Dive into Deep Learning

SECOND EDITION

Alexander J Smola

Aston Zhang · Zachary C Lipton · Mu Li · Alexander J Smola (1st edition)

Dive into Deep Learning（2026.11.27 版）。

Table of contents

まえがき **3**

インストール **15**

記法 **21**

イントロダクション **27**

基礎 **43**

1 準備 **3**
1.1 データ操作 . . . 3
1.1.1 テンソルの生成 . . . 4
1.1.2 形状変換とインデックス参照 . . . 6
1.1.3 演算 . . . 11
1.1.4 メモリと変換 . . . 14
1.1.5 まとめ . . . 16
1.1.6 演習問題 . . . 16
1.2 データの前処理 . . . 17
1.2.1 読み込みと確認 . . . 18
1.2.2 クリーニングとエンコーディング . . . 20
1.2.3 スケーリングと変換 . . . 23
1.2.4 まとめ . . . 25
1.2.5 演習問題 . . . 25
1.3 線形代数 . . . 26
1.3.1 スカラー、ベクトル、行列、テンソル . . . 27
1.3.2 要素ごとの演算とリダクション . . . 30
1.3.3 積 . . . 31
1.3.4 ノルム . . . 34
1.3.5 固有ベクトルと固有値 . . . 37
1.3.6 まとめ . . . 39
1.3.7 演習問題 . . . 39
1.4 微分積分 . . . 41
1.4.1 導関数 . . . 42
1.4.2 偏微分と勾配 . . . 45
1.4.3 連鎖律 . . . 50
1.4.4 まとめ . . . 51
1.4.5 演習問題 . . . 51

1.5 自動微分 . . . 53
1.5.1 勾配の計算 . . . 53
1.5.2 グラフの制御 . . . 55
1.5.3 プログラムの微分 . . . 57
1.5.4 フォワードモードとリバースモード . . . 58
1.5.5 まとめ . . . 58
1.5.6 演習問題 . . . 59
1.6 確率と統計 . . . 59
1.6.1 サンプリング . . . 60
1.6.2 事象と確率変数 . . . 61
1.6.3 期待値と分散 . . . 63
1.6.4 推定誤差 . . . 65
1.6.5 条件付き確率 . . . 67
1.6.6 まとめ . . . 69
1.6.7 演習問題 . . . 70

2 線形回帰 73
2.1 線形回帰 . . . 73
2.1.1 モデルと損失 . . . 74
2.1.2 損失の最小化 . . . 77
2.1.3 二乗損失とガウスノイズ . . . 82
2.1.4 ニューラルネットワークとしての線形回帰 . . . 85
2.1.5 まとめ . . . 86
2.1.6 演習問題 . . . 86
2.2 合成データ . . . 87
2.2.1 データの生成 . . . 88
2.2.2 データの観察 . . . 89
2.2.3 パラメータの復元 . . . 89
2.2.4 まとめ . . . 90
2.2.5 演習問題 . . . 90
2.3 線形回帰のスクラッチ実装 . . . 91
2.3.1 モデル、損失、勾配 . . . 91
2.3.2 スクラッチからの訓練 . . . 93
2.3.3 実装の検証 . . . 94
2.3.4 まとめ . . . 97
2.3.5 演習問題 . . . 98
2.4 実装のためのオブジェクト指向設計 . . . 99
2.4.1 ユーティリティ . . . 99
2.4.2 モデル . . . 101
2.4.3 データ . . . 103
2.4.4 訓練 . . . 105
2.4.5 線形回帰の再訓練 . . . 110
2.4.6 まとめ . . . 111
2.4.7 演習問題 . . . 111
2.5 線形回帰の簡潔な実装 . . . 112
2.5.1 全結合層 . . . 112
2.5.2 損失関数 . . . 113
2.5.3 オプティマイザ . . . 113
2.5.4 フレームワークの構成要素を用いた訓練 . . . 113
2.5.5 まとめ . . . 114

2.5.6 演習問題 . . . 115
2.6 汎化 . . . 116
2.6.1 訓練誤差と汎化誤差 . . . 116
2.6.2 学習不足と過学習 . . . 117
2.6.3 モデル選択 . . . 123
2.6.4 まとめ . . . 126
2.6.5 演習問題 . . . 126
2.7 重み減衰 . . . 128
2.7.1 ノルムペナルティ . . . 128
2.7.2 重み減衰のスクラッチ実装 . . . 131
2.7.3 固有ベクトルに沿った縮小 . . . 133
2.7.4 簡潔な実装 . . . 137
2.7.5 まとめ . . . 139
2.7.6 演習問題 . . . 139

3 線形分類 143
3.1 ソフトマックス回帰 . . . 144
3.1.1 モデル . . . 144
3.1.2 交差エントロピー損失 . . . 147
3.1.3 正解率と確信度 . . . 149
3.1.4 まとめ . . . 151
3.1.5 演習問題 . . . 151
3.2 画像分類データ . . . 153
3.2.1 データセットの読み込み . . . 154
3.2.2 ミニバッチの読み込み . . . 155
3.2.3 画像の確認 . . . 156
3.2.4 まとめ . . . 157
3.2.5 演習問題 . . . 157
3.3 ソフトマックス回帰の実装 . . . 158
3.3.1 ソフトマックス . . . 158
3.3.2 損失と正解率 . . . 159
3.3.3 モデルと損失 . . . 161
3.3.4 訓練と評価 . . . 162
3.3.5 簡潔な実装 . . . 166
3.3.6 まとめ . . . 168
3.3.7 演習問題 . . . 169
3.4 分類器の評価 . . . 170
3.4.1 正解率の限界 . . . 171
3.4.2 較正 . . . 172
3.4.3 まとめ . . . 175
3.4.4 演習問題 . . . 175
3.5 分類における汎化 . . . 177
3.5.1 テストセット . . . 177
3.5.2 テストセットの再利用 . . . 179
3.5.3 統計学習理論 . . . 181
3.5.4 まとめ . . . 182
3.5.5 演習問題 . . . 183
3.6 環境と分布シフト . . . 184
3.6.1 分布シフトの種類 . . . 184
3.6.2 共変量シフトの補正 . . . 187

3.6.3 ラベルシフトの補正 . . . 192
3.6.4 概念シフトとフィードバック . . . 193
3.6.5 まとめ . . . 194
3.6.6 演習問題 . . . 194

4 多層パーセプトロン 197
4.1 多層パーセプトロン . . . 198
4.1.1 線形モデルの限界：XOR 問題 . . . 199
4.1.2 隠れ層 . . . 200
4.1.3 表現力 . . . 202
4.1.4 活性化関数 . . . 203
4.1.5 まとめ . . . 205
4.1.6 演習問題 . . . 205
4.2 実装 . . . 206
4.2.1 スクラッチ実装 . . . 206
4.2.2 簡潔な実装 . . . 209
4.2.3 ソフトマックス回帰との比較 . . . 210
4.2.4 まとめ . . . 213
4.2.5 演習問題 . . . 213
4.3 誤差逆伝播法 . . . 214
4.3.1 計算グラフ . . . 215
4.3.2 MLP における誤差逆伝播法 . . . 216
4.3.3 メモリと計算量 . . . 219
4.3.4 まとめ . . . 220
4.3.5 演習問題 . . . 220
4.4 数値安定性と初期化 . . . 221
4.4.1 勾配消失と勾配爆発 . . . 222
4.4.2 パラメータの初期化 . . . 224
4.4.3 隠れユニット間の対称性 . . . 228
4.4.4 まとめ . . . 228
4.4.5 演習問題 . . . 228
4.5 深層学習における汎化 . . . 230
4.5.1 記憶と容量 . . . 230
4.5.2 二重降下 . . . 233
4.5.3 勾配降下法の暗黙的バイアス . . . 235
4.5.4 早期停止 . . . 237
4.5.5 まとめ . . . 238
4.5.6 演習問題 . . . 239
4.6 ドロップアウト . . . 240
4.6.1 隠れ層におけるノイズ . . . 241
4.6.2 実装 . . . 241
4.6.3 ラベルノイズ下におけるドロップアウトの効果 . . . 244
4.6.4 まとめ . . . 246
4.6.5 演習問題 . . . 246
4.7 Kaggle での住宅価格予測 . . . 247
4.7.1 データセットと log-RMSE 指標 . . . 247
4.7.2 データリークのない前処理 . . . 248
4.7.3 K 分割交差検証によるモデル選択 . . . 250
4.7.4 予測の提出 . . . 252
4.7.5 まとめ . . . 253

4.7.6 演習問題 . . . 253

5 計算 257
参考資料と関連文献 . . . 258
5.1 モジュールとモデル構築 . . . 259
5.1.1 モジュールの抽象化 . . . 259
5.1.2 モジュールコンテナ . . . 262
5.1.3 順伝播の計算 . . . 263
5.1.4 遅延初期化 . . . 265
5.1.5 設定からのモデル構築 . . . 267
5.1.6 まとめ . . . 268
5.1.7 演習問題 . . . 268
5.2 パラメータ、状態、メモリ . . . 269
5.2.1 状態木 . . . 270
5.2.2 訓練時のメモリ . . . 273
5.2.3 パラメータの共有と凍結 . . . 274
5.2.4 まとめ . . . 278
5.2.5 演習問題 . . . 278
5.3 初期化 . . . 279
5.3.1 イニシャライザの適用 . . . 279
5.3.2 カスタムイニシャライザと直接編集 . . . 281
5.3.3 残差ネットワークの初期化 . . . 282
5.3.4 まとめ . . . 285
5.3.5 演習問題 . . . 286
5.4 カスタム層とカスタム関数 . . . 287
5.4.1 パラメータをもたない層 . . . 287
5.4.2 パラメータをもつ層 . . . 288
5.4.3 固定状態をもつ層 . . . 291
5.4.4 カスタム勾配をもつ関数 . . . 292
5.4.5 まとめ . . . 296
5.4.6 演習問題 . . . 296
5.5 数値形式 . . . 297
5.5.1 浮動小数点形式 . . . 297
5.5.2 dtype の昇格とモデルのキャスト . . . 300
5.5.3 混合精度訓練 . . . 301
5.5.4 数値計算の失敗モード . . . 306
5.5.5 まとめ . . . 307
5.5.6 演習問題 . . . 308
5.6 保存、読み込み、事前学習済みの重み . . . 309
5.6.1 モデルの状態 . . . 310
5.6.2 Pickle と safetensors . . . 310
5.6.3 訓練チェックポイント . . . 313
5.6.4 事前学習済みの重み . . . 317
5.6.5 まとめ . . . 318
5.6.6 演習問題 . . . 319
5.7 GPU、デバイス、メモリ . . . 320
5.7.1 デバイスと配置 . . . 320
5.7.2 非同期実行 . . . 323
5.7.3 GPU メモリ . . . 325
5.7.4 デバイス対応の Trainer . . . 331

5.7.5 まとめ . . . 332
5.7.6 演習問題 . . . 333
5.8 再現性と検査 . . . 334
5.8.1 シードと乱数生成器 . . . 334
5.8.2 決定論的な算術演算 . . . 337
5.8.3 フックによるモデルの検査 . . . 338
5.8.4 まとめ . . . 342
5.8.5 演習問題 . . . 342

6 畳み込みニューラルネットワーク 345
参考資料と関連文献 . . . 346
6.1 全結合層から畳み込みへ . . . 347
6.1.1 同変性と不変性 . . . 348
6.1.2 全結合層への制約 . . . 349
6.1.3 畳み込みと相互相関 . . . 352
6.1.4 チャネル . . . 353
6.1.5 まとめ . . . 354
6.1.6 演習問題 . . . 354
6.2 画像に対する畳み込み . . . 355
6.2.1 相互相関 . . . 356
6.2.2 畳み込み層 . . . 357
6.2.3 行列積としての畳み込み . . . 360
6.2.4 特徴マップと受容野 . . . 361
6.2.5 まとめ . . . 362
6.2.6 演習問題 . . . 364
6.3 パディング、ストライド、ダイレーション . . . 365
6.3.1 パディング . . . 365
6.3.2 ストライド . . . 368
6.3.3 ダイレーション . . . 370
6.3.4 まとめ . . . 372
6.3.5 演習問題 . . . 372
6.4 複数の入力チャネルと複数の出力チャネル . . . 373
6.4.1 複数の入力チャネル . . . 374
6.4.2 複数の出力チャネル . . . 375
6.4.3 1×1 畳み込み . . . 377
6.4.4 まとめ . . . 378
6.4.5 演習問題 . . . 378
6.5 プーリング . . . 379
6.5.1 最大プーリングと平均プーリング . . . 379
6.5.2 小さなシフトに対する耐性 . . . 381
6.5.3 グローバルプーリング . . . 382
6.5.4 パディングとストライド . . . 382
6.5.5 複数チャネル . . . 384
6.5.6 まとめ . . . 385
6.5.7 演習問題 . . . 385
6.6 LeNet . . . 386
6.6.1 LeNet のアーキテクチャ . . . 387
6.6.2 各層の形状 . . . 388
6.6.3 パラメータと積和演算 . . . 390
6.6.4 訓練 . . . 391

6.6.5 まとめ . . . 395
6.6.6 演習問題 . . . 395

7 現代の畳み込みニューラルネットワーク 399
7.1 AlexNet と学習された画像表現 . . . 400
7.1.1 学習された特徴、データ、ハードウェア . . . 400
7.1.2 AlexNet のアーキテクチャ . . . 401
7.1.3 Fashion-MNIST での訓練 . . . 405
7.1.4 まとめ . . . 408
7.1.5 演習問題 . . . 409
7.2 ブロック、ボトルネック、ブランチ：VGG、NiN、GoogLeNet . . . 409
7.2.1 VGG：設計単位としてのブロック . . . 410
7.2.2 NiN：1×1 畳み込みとグローバル平均プーリング . . . 414
7.2.3 GoogLeNet：マルチブランチブロック . . . 419
7.2.4 ネットワークの比較 . . . 423
7.2.5 まとめ . . . 424
7.2.6 演習問題 . . . 424
7.3 正規化層 . . . 425
7.3.1 バッチ正規化 . . . 426
7.3.2 ゼロからの実装 . . . 428
7.3.3 バッチ正規化を用いた LeNet . . . 429
7.3.4 バッチ統計量を使わない正規化 . . . 433
7.3.5 まとめ . . . 435
7.3.6 演習問題 . . . 435
7.4 残差ネットワーク . . . 436
7.4.1 関数クラスと深さ . . . 437
7.4.2 残差ブロック . . . 437
7.4.3 プレーンネットワークと残差ネットワークの比較 . . . 441
7.4.4 ResNet-18 . . . 443
7.4.5 ResNeXt と DenseNet . . . 447
7.4.6 まとめ . . . 451
7.4.7 演習問題 . . . 452
7.5 効率的な畳み込みネットワーク . . . 453
7.5.1 デプスワイズ分離可能ネットワーク . . . 454
7.5.2 スケーリングと探索 . . . 464
7.5.3 構造的再パラメータ化 . . . 466
7.5.4 まとめ . . . 469
7.5.5 演習問題 . . . 469
7.6 畳み込みネットワークの現代的な訓練レシピ . . . 471
7.6.1 2015 年から 2022 年にかけての訓練レシピ . . . 471
7.6.2 各構成要素の実装 . . . 472
7.6.3 1 つのネットワークに対する 2 つのレシピ . . . 478
7.6.4 まとめ . . . 481
7.6.5 演習問題 . . . 481
7.7 ConvNeXt：ResNet の段階的な現代化 . . . 482
7.7.1 現代化のロードマップ . . . 483
7.7.2 実装 . . . 485
7.7.3 Fashion-MNIST での小規模ロードマップ . . . 490
7.7.4 限界と未解決問題 . . . 493
7.7.5 まとめ . . . 493

7.7.6 演習問題 . . . 493
7.8 畳み込みネットワークの設計空間 . . . 494
7.8.1 AnyNet 設計空間 . . . 495
7.8.2 設計空間の比較 . . . 499
7.8.3 RegNet . . . 503
7.8.4 まとめ . . . 507
7.8.5 演習問題 . . . 507

8 系列モデル 509
8.1 系列の扱い . . . 510
8.1.1 系列データ . . . 510
8.1.2 自己回帰モデル . . . 511
8.1.3 マルコフモデルと定常性 . . . 512
8.1.4 ノイズを含む正弦波の予測 . . . 513
8.1.5 まとめ . . . 520
8.1.6 演習問題 . . . 520
8.2 テキストからトークンへ . . . 521
8.2.1 データセットの読み込み . . . 521
8.2.2 文字、単語、バイト . . . 522
8.2.3 バイトペアエンコーディング . . . 526
8.2.4 特殊トークン . . . 534
8.2.5 実世界のトークナイザ . . . 534
8.2.6 まとめ . . . 538
8.2.7 演習問題 . . . 538
8.3 言語モデル . . . 539
8.3.1 n グラム言語モデル . . . 540
8.3.2 パープレキシティと平滑化 . . . 546
8.3.3 系列の分割 . . . 552
8.3.4 まとめ . . . 554
8.3.5 演習問題 . . . 554
8.4 リカレントニューラルネットワーク . . . 556
8.4.1 隠れ状態を持つリカレントニューラルネットワーク . . . 556
8.4.2 RNN 言語モデルのゼロからの実装 . . . 560
8.4.3 勾配クリッピング . . . 562
8.4.4 訓練と評価 . . . 563
8.4.5 テキスト生成 . . . 570
8.4.6 簡潔な実装 . . . 570
8.4.7 まとめ . . . 572
8.4.8 演習問題 . . . 572
8.5 通時的誤差逆伝播法 . . . 575
8.5.1 勾配消失と勾配爆発 . . . 575
8.5.2 打ち切り型通時的誤差逆伝播法 . . . 580
8.5.3 まとめ . . . 580
8.5.4 演習問題 . . . 581
8.6 デコーディングと生成 . . . 582
8.6.1 デコーディング問題 . . . 582
8.6.2 貪欲デコーディング . . . 585
8.6.3 ビームサーチ . . . 586
8.6.4 サンプリング . . . 589
8.6.5 評価とコスト . . . 594

8.6.6 まとめ . . . 596
8.6.7 演習問題 . . . 596

発展 **599**

9 最適化アルゴリズム **603**
9.1 ランドスケープと勾配降下法 . . . 604
9.1.1 最適化ランドスケープと条件数 . . . 605
9.1.2 臨界点、鞍点、プラトー . . . 608
9.1.3 降下保証と極小点の安定性 . . . 610
9.1.4 安定性の縁 . . . 612
9.1.5 ニュートン法と前処理 . . . 615
9.1.6 まとめ . . . 616
9.1.7 演習問題 . . . 617
9.2 確率的勾配降下法とミニバッチ勾配降下法 . . . 618
9.2.1 確率的勾配降下法 . . . 618
9.2.2 ミニバッチ . . . 621
9.2.3 まとめ . . . 625
9.2.4 演習問題 . . . 626
9.3 2 つのテストモデル . . . 626
9.3.1 言語モデル TinyLM . . . 627
9.3.2 画像分類器 FashionCNN . . . 630
9.3.3 訓練ハーネスと評価補助関数 . . . 631
9.3.4 ミニバッチ SGD のベースライン実行と実験プロトコル . . . 634
9.3.5 まとめ . . . 636
9.3.6 演習問題 . . . 636
9.4 モメンタム . . . 637
9.4.1 モメンタム法 . . . 638
9.4.2 加速と臨界減衰 . . . 640
9.4.3 データ上でのモメンタム . . . 642
9.4.4 ネステロフのモメンタム . . . 644
9.4.5 まとめ . . . 646
9.4.6 演習問題 . . . 646
9.5 Adam と AdamW . . . 646
9.5.1 座標ごとの適応的学習率 . . . 647
9.5.2 言語モデルにおける Adam . . . 651
9.5.3 分離型重み減衰（AdamW） . . . 655
9.5.4 まとめ . . . 659
9.5.5 演習問題 . . . 659
9.6 Muon . . . 660
9.6.1 ノルムと最急降下 . . . 661
9.6.2 ニュートン・シュルツ反復による直交化 . . . 663
9.6.3 ゼロからの Muon の実装 . . . 665
9.6.4 まとめ . . . 669
9.6.5 演習問題 . . . 670
9.7 スケジュール . . . 670
9.7.1 ノイズのある谷における学習率減衰 . . . 670
9.7.2 言語モデルにおける減衰形状 . . . 672
9.7.3 学習率ウォームアップ . . . 675
9.7.4 ウォームアップ・安定・減衰（WSD）とチェックポイント分岐 . . . 676

9.7.5 まとめ . . . 678
9.7.6 演習問題 . . . 678
9.8 バッチサイズ . . . 679
9.8.1 勾配ノイズスケール . . . 679
9.8.2 バッチサイズ、ステップ効率、学習率のスケーリング . . . 681
9.8.3 実験的検証：目標到達ステップ数とスケーリング則 . . . 683
9.8.4 まとめ . . . 688
9.8.5 演習問題 . . . 688
9.9 スケールアップ . . . 689
9.9.1 標準パラメータ化における学習率と幅 . . . 689
9.9.2 最大更新パラメータ化（μP） . . . 693
9.9.3 Transformer とスペクトルオプティマイザへの拡張 . . . 697
9.9.4 まとめ . . . 698
9.9.5 演習問題 . . . 698
9.10 実践 . . . 699
9.10.1 勾配クリッピングとスパイクからの回復 . . . 699
9.10.2 重み平均化（EMA） . . . 702
9.10.3 エンドツーエンド設定の実例 . . . 704
9.10.4 まとめ . . . 707
9.10.5 演習問題 . . . 707
9.11 関連文献 . . . 708
9.11.1 基礎的な教科書、サーベイ、プレイブック . . . 708
9.11.2 損失曲面 . . . 708
9.11.3 古典的な 2 次手法と分散低減手法 . . . 709
9.11.4 行列オプティマイザと構造的オプティマイザ . . . 709
9.11.5 スケジュール、平均化、大規模ベンチマーク . . . 710

10 アテンション **711**
10.1 クエリ、キー、値 . . . 712
10.1.1 微分可能な参照としてのアテンション . . . 712
10.1.2 固定カーネルによるアテンションプーリング . . . 713
10.1.3 診断手段としてのアテンション重み . . . 715
10.1.4 まとめ . . . 716
10.1.5 演習問題 . . . 716
10.2 アテンションのスコアリングとマスキング . . . 718
10.2.1 ドット積アテンション . . . 719
10.2.2 マスキング . . . 722
10.2.3 ミニバッチに対するアテンション . . . 726
10.2.4 学習されたアライメントとしてのアテンション . . . 728
10.2.5 まとめ . . . 731
10.2.6 演習問題 . . . 731
10.3 自己アテンションと複数のヘッド . . . 733
10.3.1 自己アテンション . . . 733
10.3.2 単一ヘッドの限界 . . . 736
10.3.3 マルチヘッドアテンション . . . 737
10.3.4 クロスアテンション . . . 743
10.3.5 まとめ . . . 743
10.3.6 演習問題 . . . 743
10.4 位置情報 . . . 745
10.4.1 置換同変性 . . . 746

10.4.2 絶対位置埋め込み . . . 746
10.4.3 回転位置埋め込み . . . 748
10.4.4 距離ペナルティとエンコーディングなし . . . 750
10.4.5 訓練長を超える外挿 . . . 751
10.4.6 まとめ . . . 756
10.4.7 演習問題 . . . 756
10.5 アテンションのコスト . . . 758
10.5.1 二次の時間とメモリ . . . 759
10.5.2 行列を保持しない厳密なアテンション . . . 760
10.5.3 まとめ . . . 764
10.5.4 演習問題 . . . 764
10.6 ウィンドウアテンションと線形アテンション . . . 766
10.6.1 スライディングウィンドウアテンション . . . 766
10.6.2 線形アテンション . . . 771
10.6.3 機構の選び方 . . . 774
10.6.4 まとめ . . . 775
10.6.5 演習問題 . . . 775
10.7 アテンション回路 . . . 776
10.7.1 回路と残差ストリーム . . . 777
10.7.2 コピーと誘導 . . . 779
10.7.3 回路の裏付け . . . 782
10.7.4 コンテキスト内学習 . . . 787
10.7.5 説明としてのアテンション重み . . . 789
10.7.6 まとめ . . . 789
10.7.7 演習問題 . . . 789
10.8 関連文献 . . . 792
10.8.1 ルックアップとその正規化 . . . 792
10.8.2 自己アテンションとヘッド . . . 793
10.8.3 位置 . . . 794
10.8.4 コスト . . . 794
10.8.5 大規模モデルにおける回路 . . . 795
10.8.6 推奨される参考資料 . . . 796

11 Transformer **797**
11.1 Transformer ブロック . . . 798
11.1.1 Transformer ブロックの構成要素 . . . 798
11.1.2 正規化 . . . 799
11.1.3 フィードフォワードネットワーク：GELU から SwiGLU へ . . . 806
11.1.4 設定可能な `TransformerBlock` . . . 807
11.1.5 まとめ . . . 808
11.1.6 演習問題 . . . 808
11.2 デコーダのみの言語モデル（GPT） . . . 809
11.2.1 ブロックから言語モデルへ . . . 810
11.2.2 WikiText-103 におけるバイトレベル GPT の訓練 . . . 813
11.2.3 統制された比較 . . . 816
11.2.4 モデルからのサンプリング . . . 818
11.2.5 事前学習済み GPT-2（124M）の読み込み . . . 819
11.2.6 現代的なレシピ . . . 822
11.2.7 まとめ . . . 823
11.2.8 演習問題 . . . 823

11.3 KV キャッシュとキャッシュ削減 824
11.3.1 再計算からキャッシュへ 825
11.3.2 メモリ容量とハードウェアのルーフライン 830
11.3.3 KV キャッシュの縮小 834
11.3.4 キャッシュ削減手法の比較 839
11.3.5 まとめ 840
11.3.6 演習問題 840
11.4 エンコーダ、デコーダ、およびクロスアテンション 841
11.4.1 エンコーダ、デコーダ、およびエンコーダ・デコーダアーキテクチャ . . 842
11.4.2 双方向文脈とマスクトークン予測 842
11.4.3 エンコーダ・デコーダにおけるクロスアテンション 846
11.4.4 インターフェースとしてのクロスアテンション：固定サイズの潜在ボトルネック 850
11.4.5 3 つの配線の使い分け 851
11.4.6 まとめ 852
11.4.7 演習問題 852
11.5 Vision Transformer（ViT） 853
11.5.1 トークンとしてのパッチ（PatchEmbedding） 853
11.5.2 Vision Transformer のアーキテクチャ 854
11.5.3 Fashion-MNIST での訓練 856
11.5.4 パラメータ数を揃えた畳み込みネットワークとの比較 859
11.5.5 まとめ 862
11.5.6 演習問題 862
11.6 スケーリング則 863
11.6.1 パラメータ数（$N \approx 12Ld^2$）と FLOPs（$C \approx 6ND$）の算出 863
11.6.2 WikiText-103 における小規模スケーリング実験 865
11.6.3 計算量最適な配分（Chinchilla スケーリング） 868
11.6.4 まとめ 871
11.6.5 演習問題 871
11.7 混合エキスパート（MoE） 873
11.7.1 条件付き計算：保持パラメータとアクティブパラメータ 873
11.7.2 混合エキスパート層 874
11.7.3 ルーティング崩壊と負荷分散 878
11.7.4 混合エキスパート言語モデル 882
11.7.5 まとめ 885
11.7.6 演習問題 885
11.8 関連文献 887
11.8.1 正規化の配置と残差ストリームの派生形 887
11.8.2 フィードフォワードネットワークと活性化関数 888
11.8.3 長文脈推論とキャッシュ圧縮 888
11.8.4 エンコーダ、クロスアテンション、ビジョンバックボーン 889
11.8.5 スケーリング則と混合エキスパートの設計 889

12 状態空間モデル **891**
12.1 ゲート付きリカレントネットワーク 892
12.1.1 ゲート 892
12.1.2 長・短期記憶 893
12.1.3 ゲート付きリカレントユニット 896
12.1.4 ゲート付きネットワークの訓練 897
12.1.5 まとめ 905

12.1.6 演習問題 . . . 905
12.2 系列対系列学習 . . . 907
12.2.1 エンコーダ・デコーダの抽象化 . . . 907
12.2.2 翻訳のスコアリング . . . 908
12.2.3 固定状態のボトルネック . . . 910
12.2.4 まとめ . . . 912
12.2.5 演習問題 . . . 912
12.3 線形漸化式と並列スキャン . . . 913
12.3.1 線形漸化式 . . . 914
12.3.2 並列スキャン . . . 915
12.3.3 線形リカレント言語モデル . . . 917
12.3.4 アテンションとのコスト比較 . . . 919
12.3.5 まとめ . . . 921
12.3.6 演習問題 . . . 921
12.4 状態空間モデル . . . 922
12.4.1 連続時間から漸化式へ . . . 923
12.4.2 対角状態空間層 . . . 927
12.4.3 まとめ . . . 931
12.4.4 演習問題 . . . 931
12.5 選択的状態空間モデル . . . 932
12.5.1 選択的コピー . . . 932
12.5.2 選択的ダイナミクス . . . 934
12.5.3 Mamba 層 . . . 936
12.5.4 訓練済み Mamba モデル . . . 940
12.5.5 まとめ . . . 944
12.5.6 演習問題 . . . 945
12.6 行列状態漸化式 . . . 946
12.6.1 連想メモリ . . . 946
12.6.2 減衰 . . . 949
12.6.3 双対形式 . . . 950
12.6.4 ゲート付き線形アテンション層 . . . 954
12.6.5 まとめ . . . 959
12.6.6 演習問題 . . . 959
12.7 デルタ則 . . . 961
12.7.1 加算的書き込み . . . 961
12.7.2 勾配ステップとしての書き込み . . . 962
12.7.3 Gated DeltaNet . . . 965
12.7.4 回帰としてのメモリ . . . 972
12.7.5 まとめ . . . 973
12.7.6 演習問題 . . . 973
12.8 ハイブリッドアーキテクチャ . . . 974
12.8.1 固定サイズの状態の限界 . . . 975
12.8.2 1 つのアテンション層による想起 . . . 976
12.8.3 同規模での言語モデリング . . . 978
12.8.4 推論時のメモリ . . . 982
12.8.5 本章のメモリの比較 . . . 982
12.8.6 まとめ . . . 985
12.8.7 演習問題 . . . 985
12.9 関連文献 . . . 986
12.9.1 再帰とアテンションの組み合わせ . . . 986

12.9.2 行列メモリへの書き込み . . . 987
12.9.3 状態空間モデルと長い畳み込み . . . 988
12.9.4 言語を超えて . . . 989
12.9.5 参照実装とチュートリアル . . . 989

13 計算性能 **991**
13.1 性能モデル . . . 992
13.1.1 演算密度とルーフライン . . . 992
13.1.2 GPU コードの時間計測 . . . 995
13.1.3 3 つのレジーム . . . 996
13.1.4 訓練ステップのプロファイリング . . . 1000
13.1.5 まとめ . . . 1002
13.1.6 演習問題 . . . 1003
13.2 ハードウェア . . . 1004
13.2.1 メモリ階層 . . . 1004
13.2.2 演算性能とメモリ帯域幅 . . . 1008
13.2.3 GPU . . . 1009
13.2.4 GPU 外のデータ経路 . . . 1011
13.2.5 ルーフライン上のワークロード . . . 1013
13.2.6 まとめ . . . 1014
13.2.7 演習問題 . . . 1014
13.3 計算グラフとコンパイル . . . 1016
13.3.1 グラフキャプチャ . . . 1016
13.3.2 演算融合 . . . 1019
13.3.3 訓練ステップのコンパイル . . . 1021
13.3.4 カーネルリプレイ . . . 1022
13.3.5 コンパイルの限界 . . . 1023
13.3.6 まとめ . . . 1024
13.3.7 演習問題 . . . 1024
13.4 メモリと精度 . . . 1025
13.4.1 訓練ステップのメモリ . . . 1025
13.4.2 混合精度 . . . 1029
13.4.3 活性化チェックポイント . . . 1031
13.4.4 勾配累積 . . . 1032
13.4.5 メモリ手法の選択 . . . 1034
13.4.6 まとめ . . . 1034
13.4.7 演習問題 . . . 1035
13.5 第一原理からのマルチ GPU . . . 1036
13.5.1 3 つの分割方法 . . . 1036
13.5.2 手作業によるデータ並列 . . . 1037
13.5.3 リング allreduce . . . 1040
13.5.4 通信コストとスケーリング . . . 1043
13.5.5 まとめ . . . 1048
13.5.6 演習問題 . . . 1048
13.6 データ並列の実践 . . . 1049
13.6.1 実用データ並列 . . . 1050
13.6.2 4 基の GPU でのスケーリング . . . 1054
13.6.3 まとめ . . . 1058
13.6.4 演習問題 . . . 1058
13.7 訓練状態のシャーディング . . . 1059

13.7.1 オプティマイザ状態、勾配、重みの分割 1060
13.7.2 メモリと通信 1062
13.7.3 2 基の GPU によるシャーディング訓練 1063
13.7.4 まとめ 1065
13.7.5 演習問題 1066
13.8 ケーススタディ：Transformer の最適化 1067
13.8.1 実験設定 1067
13.8.2 単一 GPU 1069
13.8.3 データ並列 1076
13.8.4 結果 1078
13.8.5 まとめ 1081
13.8.6 演習問題 1081
13.9 関連文献 1083
13.9.1 計測とモデル化 1083
13.9.2 ハードウェアと数値形式 1084
13.9.3 コンパイラとカーネル 1084
13.9.4 メモリ 1085
13.9.5 通信とスケール 1086

14 強化学習 **1087**
参考資料と関連文献 1088
14.1 マルコフ決定過程 1089
14.1.1 モデル 1091
14.1.2 収益・割引・ホライズン 1093
14.1.3 状態の選び方 1097
14.1.4 報酬設計 1098
14.1.5 2 つの特別な場合 1100
14.1.6 まとめ 1102
14.1.7 演習問題 1102
14.2 動的計画法 1105
14.2.1 価値とベルマン方程式 1105
14.2.2 価値反復法 1108
14.2.3 方策評価と方策反復法 1117
14.2.4 滑る氷の上での最適方策 1120
14.2.5 報酬シェイピングと最適方策 1123
14.2.6 まとめ 1125
14.2.7 演習問題 1126
14.3 時間差分学習 1128
14.3.1 サンプルによるバックアップ 1129
14.3.2 滑る湖における Q 学習 1133
14.3.3 オフポリシー学習 1139
14.3.4 学習されたモデルにおける計画 1140
14.3.5 まとめ 1142
14.3.6 演習問題 1142
14.4 探索 1145
14.4.1 多腕バンディット問題 1146
14.4.2 楽観主義と事後サンプリング 1150
14.4.3 MDP における探索 1154
14.4.4 検証器タスクにおける探索 1158
14.4.5 まとめ 1160

14.4.6 演習問題 1160
14.5 デモンストレーションからの学習 1162
14.5.1 行動クローニング 1163
14.5.2 分布シフト 1167
14.5.3 複合誤差 1170
14.5.4 対話的なデータ収集 1172
14.5.5 まとめ 1176
14.5.6 演習問題 1176
14.6 最適制御、LQR、モデル予測制御 1179
14.6.1 線形二次制御 1179
14.6.2 平衡点周りでの非線形システムの線形化 1183
14.6.3 反復 LQR による非線形軌跡最適化 1185
14.6.4 移動ホライズンモデル予測制御 1188
14.6.5 まとめ 1192
14.6.6 演習問題 1192
14.7 方策勾配 1194
14.7.1 方策のパラメータ化 1195
14.7.2 方策勾配 1196
14.7.3 共通例題における REINFORCE 1199
14.7.4 方策勾配定理 1205
14.7.5 コストと限界 1206
14.7.6 まとめ 1209
14.7.7 演習問題 1209
14.8 ベースラインと分散低減 1212
14.8.1 ゼロ平均恒等式 1213
14.8.2 残りの累積報酬、ベースライン、制御変量 1214
14.8.3 中心化、スケーリング、正規化 1217
14.8.4 5 つの推定量の比較 1222
14.8.5 まとめ 1229
14.8.6 演習問題 1230
14.9 強化学習における関数近似 1233
14.9.1 テーブルに代わるネットワーク 1234
14.9.2 CartPole における REINFORCE 1237
14.9.3 連続行動 1241
14.9.4 状態間の汎化 1246
14.9.5 手法の見取り図 1247
14.9.6 まとめ 1250
14.9.7 演習問題 1250

15 深層強化学習 **1255**
15.1 アクター・クリティック法と複数ステップ収益 1256
15.1.1 時間差分アドバンテージ 1257
15.1.2 アクター・クリティックの更新 1260
15.1.3 複数ステップ収益と GAE 1266
15.1.4 λ 収益のバイアスと分散 1268
15.1.5 まとめ 1273
15.1.6 演習問題 1273
15.2 信頼領域と近接方策最適化 1276
15.2.1 パラメータ空間と方策空間 1277
15.2.2 重要度サンプリング 1278

15.2.3 信頼領域 . . . 1280
15.2.4 PPO の実践 . . . 1283
15.2.5 まとめ . . . 1296
15.2.6 演習問題 . . . 1296
15.3 KL 正則化方策最適化 . . . 1300
15.3.1 比較からの報酬の学習 . . . 1300
15.3.2 当てはめ報酬の最適化 . . . 1304
15.3.3 KL 正則化目的関数 . . . 1305
15.3.4 ソフトバックアップ . . . 1309
15.3.5 まとめ . . . 1313
15.3.6 演習問題 . . . 1313
15.4 深層 Q ネットワーク . . . 1315
15.4.1 不安定性の要因 . . . 1317
15.4.2 経験再生とターゲットネットワーク . . . 1319
15.4.3 過大評価 . . . 1329
15.4.4 まとめ . . . 1333
15.4.5 演習問題 . . . 1333
15.5 ソフトアクタークリティック . . . 1336
15.5.1 最大エントロピー目的関数 . . . 1337
15.5.2 スクワッシュされたガウス方策 . . . 1340
15.5.3 アルゴリズム . . . 1345
15.5.4 サンプル効率と較正 . . . 1351
15.5.5 まとめ . . . 1358
15.5.6 演習問題 . . . 1358
15.6 学習されたモデルと計画 . . . 1361
15.6.1 データからのダイナミクスモデル . . . 1362
15.6.2 モデル内での計画 . . . 1367
15.6.3 方策改善としての探索 . . . 1372
15.6.4 まとめ . . . 1377
15.6.5 演習問題 . . . 1377
15.7 オフライン強化学習 . . . 1379
15.7.1 固定された記録からの学習 . . . 1379
15.7.2 悲観主義 . . . 1387
15.7.3 表形式を超えて . . . 1391
15.7.4 まとめ . . . 1393
15.7.5 演習問題 . . . 1393
15.8 系列生成のための強化学習 . . . 1395
15.8.1 軌跡としての応答 . . . 1395
15.8.2 検証器・報酬モデルとその抜け穴 . . . 1398
15.8.3 事後学習の更新則 . . . 1399
15.8.4 まとめ . . . 1402
15.8.5 演習問題 . . . 1402
15.9 関連文献 . . . 1404
15.9.1 実践における方策最適化 . . . 1405
15.9.2 DQN 以降の価値学習 . . . 1406
15.9.3 2018 年以降の連続制御 . . . 1406
15.9.4 モデル、想像、探索 . . . 1407
15.9.5 大規模なオフライン手法 . . . 1407
15.9.6 事後学習の変種 . . . 1408
15.9.7 本章の先の話題 . . . 1409

15.9.8 参考書、ベンチマーク、コードベース 1409
15.9.9 プロジェクト 1409

16 敵対的生成ネットワーク 1411
記法 1412
16.1 敵対的生成ネットワーク 1412
16.1.1 尤度から 2 サンプル比較へ 1413
16.1.2 対数損失の目的関数 1414
16.1.3 生成器の勾配 1417
16.1.4 2 次元ガウス分布での理論の検証 1420
16.1.5 Fashion-MNIST での MLP GAN の訓練 1427
16.1.6 まとめ 1429
16.1.7 演習問題 1429
16.2 敵対的目的関数とダイバージェンス 1431
16.2.1 一般的なテンプレート 1431
16.2.2 分類損失と f ダイバージェンス 1432
16.2.3 積分確率指標 1437
16.2.4 データから遠い領域と近い領域における勾配 1439
16.2.5 まとめ 1447
16.2.6 演習問題 1447
16.3 相対論的目的関数 1449
16.3.1 ペアのスコア付け 1449
16.3.2 ペアリングダイバージェンス 1450
16.3.3 ランキングとモード被覆 1453
16.3.4 まとめ 1456
16.3.5 演習問題 1456
16.4 勾配ペナルティと収束 1457
16.4.1 Dirac-GAN 1458
16.4.2 ゼロ中心勾配ペナルティ 1459
16.4.3 勾配ペナルティが価値関数に及ぼす影響 1462
16.4.4 R3GAN レシピ 1464
16.4.5 まとめ 1469
16.4.6 演習問題 1470
16.5 敵対的画像生成 1471
16.5.1 DCGAN アーキテクチャ 1472
16.5.2 正規化なしバックボーン 1477
16.5.3 統制された損失の比較 1480
16.5.4 定量的評価指標 1485
16.5.5 潜在補間とクリティックの過学習 1488
16.5.6 まとめ 1490
16.5.7 演習問題 1490
16.6 条件付き生成 1492
16.6.1 条件付き目的関数 1492
16.6.2 条件付けのアーキテクチャ 1493
16.6.3 CIFAR-10 におけるクラス条件付き生成 1495
16.6.4 条件付き生成器の評価 1498
16.6.5 まとめ 1502
16.6.6 演習問題 1502
16.7 GAN を超える敵対的損失 1504
16.7.1 条件付き平均と多峰性ターゲット 1504

16.7.2 拡散蒸留における識別器 1507
16.7.3 本章の敵対的目的関数の比較 1510
16.7.4 まとめ 1511
16.7.5 演習問題 1511
16.8 関連文献 1513
16.8.1 推奨される入門文献とコードベース 1513
16.8.2 目的関数とミニマックスダイナミクス 1513
16.8.3 固定された差異と学習される差異 1514
16.8.4 アーキテクチャ、特徴空間、データ効率 1514
16.8.5 条件付け、画像変換、現代のハイブリッドシステム 1515

17 拡散モデル **1517**
17.1 ノイズ除去とスコア 1518
17.1.1 通し例 1518
17.1.2 最適ノイズ除去器 1521
17.1.3 単一ノイズレベルの限界 1526
17.1.4 画像のノイズ除去 1531
17.1.5 まとめ 1533
17.1.6 演習問題 1534
17.2 ガウス経路：拡散とフローマッチング 1536
17.2.1 データからノイズへの経路 1536
17.2.2 周辺場への回帰 1542
17.2.3 まとめ 1550
17.2.4 演習問題 1550
17.3 積分によるサンプリング 1552
17.3.1 確率質量の輸送 1553
17.3.2 確率フロー ODE に対するオイラー法 1555
17.3.3 祖先連鎖と逆 SDE 1559
17.3.4 両サンプラからのサンプル 1561
17.3.5 まとめ 1563
17.3.6 演習問題 1563
17.4 画像の拡散モデル 1565
17.4.1 ノイズ除去 U-Net 1565
17.4.2 訓練とサンプリング 1569
17.4.3 サンプル品質の測定 1573
17.4.4 ガイダンス 1576
17.4.5 まとめ 1582
17.4.6 演習問題 1582
17.5 サンプラ、逆変換、編集 1584
17.5.1 決定論的サンプラと確率的サンプラ 1584
17.5.2 ステップ数の削減 1587
17.5.3 逆変換と編集 1597
17.5.4 まとめ 1603
17.5.5 演習問題 1603
17.6 尤度、重み付け、設計空間 1605
17.6.1 尤度境界 1605
17.6.2 重み付けとしての目的関数 1614
17.6.3 設計空間 1617
17.6.4 まとめ 1626
17.6.5 演習問題 1626

17.7 潜在拡散と Transformer 1628
17.7.1 知覚圧縮 1629
17.7.2 潜在空間での拡散 1633
17.7.3 Transformer バックボーン 1638
17.7.4 表現とバックボーンの選択 1644
17.7.5 まとめ 1645
17.7.6 演習問題 1646
17.8 少数ステップ生成 1647
17.8.1 長いオイラーステップ 1648
17.8.2 フロー写像 1649
17.8.3 フロー写像の訓練 1654
17.8.4 分布マッチング 1661
17.8.5 まとめ 1662
17.8.6 演習問題 1663
17.9 離散拡散 1664
17.9.1 トークンの破損 1665
17.9.2 マスク拡散の変分上界 1667
17.9.3 マスク解除によるサンプリング 1670
17.9.4 訓練されたノイズ除去器 1673
17.9.5 隣接するモデルと振り返り 1680
17.9.6 まとめ 1682
17.9.7 演習問題 1682
17.10関連文献 1685
17.10.1専門書、講義、コード 1685
17.10.2エネルギーベースモデルとスコアマッチング 1686
17.10.3画像ネットワークとガイダンス 1687
17.10.4評価、暗記、汎化 1687
17.10.5サンプラ、編集、逆問題 1687
17.10.6スケジュール、重み付け、尤度 1688
17.10.7潜在表現、テキスト、動画 1688
17.10.8少数ステップ生成 1688
17.10.9離散拡散 1688
17.10.10事後学習と視覚・テキスト以外の応用 1689

深層学習のための数学 **1691**

33 線形代数 **1693**
33.1 ベクトル、角度、射影 1694
33.1.1 ベクトルとその幾何学 1694
33.1.2 部分空間と最小二乗法 1699
33.1.3 高次元における類似度 1704
33.1.4 超平面と決定境界 1708
33.1.5 まとめ 1712
33.1.6 演習問題 1712
33.2 線形変換としての行列 1715
33.2.1 線形変換 1716
33.2.2 ランクと可逆性 1723
33.2.3 行列式 1729
33.2.4 まとめ 1737
33.2.5 演習問題 1738

33.3 固有値分解 1740
33.3.1 固有値と対角化 1741
33.3.2 対称行列と二次形式 1751
33.3.3 まとめ 1756
33.3.4 演習問題 1757
33.4 実践におけるスペクトル 1758
33.4.1 固有値の位置特定と計算 1759
33.4.2 反復と安定性 1764
33.4.3 まとめ 1770
33.4.4 演習問題 1770
33.5 特異値分解と低ランク近似 1772
33.5.1 特異値分解 1773
33.5.2 低ランク近似 1781
33.5.3 SVD による連立一次方程式の解法 1788
33.5.4 深層学習における SVD 1792
33.5.5 まとめ 1797
33.5.6 演習問題 1797
33.6 関連文献 1800
33.6.1 幾何学と線形変換 1800
33.6.2 固有値とスペクトル 1801
33.6.3 打ち切り SVD を超える低ランク構造 1802
33.6.4 書籍、講義、講義ノート 1804

34 微分積分と自動微分 1805
34.1 1 変数の微分積分 1806
34.1.1 導関数 1806
34.1.2 導関数の計算 1809
34.1.3 平均値の定理 1812
34.1.4 1 変数の勾配降下法 1814
34.1.5 曲率とテイラー級数 1816
34.1.6 微分不可能な点 1826
34.1.7 まとめ 1830
34.1.8 演習問題 1830
34.2 多変数の微分積分 1833
34.2.1 偏微分と勾配 1834
34.2.2 勾配の幾何学 1836
34.2.3 ヘッセ行列 1840
34.2.4 制約付き最適化とラグランジュ乗数 1847
34.2.5 まとめ 1850
34.2.6 演習問題 1850
34.3 計算グラフ上の連鎖律 1852
34.3.1 経路にわたる和としての連鎖律 1852
34.3.2 誤差逆伝播法 1855
34.3.3 まとめ 1859
34.3.4 演習問題 1859
34.4 行列微分 1860
34.4.1 ヤコビ行列 1861
34.4.2 ヤコビ行列の連鎖律 1864
34.4.3 ベクトルと行列に関する導関数 1868
34.4.4 まとめ 1873

34.4.5 演習問題 . . . 1873
34.5 自動微分 . . . 1875
34.5.1 フォワードモードと二重数 . . . 1876
34.5.2 リバースモードとテープ . . . 1880
34.5.3 ヤコビ行列積の合成 . . . 1888
34.5.4 まとめ . . . 1892
34.5.5 演習問題 . . . 1893
34.6 積分法 . . . 1895
34.6.1 定積分 . . . 1896
34.6.2 微分積分学の基本定理 . . . 1898
34.6.3 重積分 . . . 1906
34.6.4 まとめ . . . 1912
34.6.5 演習問題 . . . 1912
34.7 期待値とその勾配 . . . 1913
34.7.1 積分としての期待値 . . . 1914
34.7.2 積分記号下の微分 . . . 1915
34.7.3 スコア関数勾配とパスワイズ勾配 . . . 1917
34.7.4 まとめ . . . 1920
34.7.5 演習問題 . . . 1920
34.8 関連文献 . . . 1921
34.8.1 折れ点における微分 . . . 1922
34.8.2 ベクトル写像と行列写像の微分 . . . 1922
34.8.3 自動微分 . . . 1923
34.8.4 積分 . . . 1924
34.8.5 書籍、講義、チュートリアル . . . 1925

35 最適化 **1927**
記法 . . . 1928
35.1 勾配降下法と曲率 . . . 1928
35.1.1 降下方向と滑らかさ . . . 1929
35.1.2 二次モデルと条件数 . . . 1936
35.1.3 モメンタムと加速 . . . 1941
35.1.4 まとめ . . . 1945
35.1.5 演習問題 . . . 1945
35.2 曲率法：ニュートン法、準ニュートン法、信頼領域法 . . . 1948
35.2.1 ニュートン法 . . . 1949
35.2.2 準ニュートン法 . . . 1954
35.2.3 信頼領域 . . . 1956
35.2.4 まとめ . . . 1959
35.2.5 演習問題 . . . 1959
35.3 凸性 . . . 1960
35.3.1 凸集合 . . . 1961
35.3.2 凸関数 . . . 1962
35.3.3 凸性の帰結 . . . 1971
35.3.4 凸性の判定 . . . 1978
35.3.5 凸性を超えて . . . 1980
35.3.6 まとめ . . . 1984
35.3.7 演習問題 . . . 1984
35.4 確率的勾配法 . . . 1987
35.4.1 ミニバッチ勾配 . . . 1988

35.4.2 ステップサイズの減衰 . . . 1993
35.4.3 固定予算における収束率 . . . 1999
35.4.4 分散低減 . . . 2004
35.4.5 まとめ . . . 2007
35.4.6 演習問題 . . . 2008
35.5 適応的手法 . . . 2010
35.5.1 座標ごとのステップサイズ . . . 2011
35.5.2 座標スケーリングとその限界 . . . 2015
35.5.3 分離型重み減衰 . . . 2021
35.5.4 ウォームアップ . . . 2024
35.5.5 構造化前処理行列 . . . 2025
35.5.6 まとめ . . . 2027
35.5.7 演習問題 . . . 2028
35.6 制約付き最適化 . . . 2030
35.6.1 等式制約とラグランジュ乗数 . . . 2031
35.6.2 不等式制約と KKT 条件 . . . 2034
35.6.3 射影と射影勾配降下法 . . . 2037
35.6.4 近接作用素と座標降下法 . . . 2044
35.6.5 まとめ . . . 2050
35.6.6 演習問題 . . . 2050
35.7 双対性 . . . 2053
35.7.1 双対問題 . . . 2054
35.7.2 強双対性 . . . 2058
35.7.3 価格としての乗数 . . . 2061
35.7.4 凸共役 . . . 2064
35.7.5 具体例 . . . 2068
35.7.6 まとめ . . . 2074
35.7.7 演習問題 . . . 2075
35.8 数値安定性と条件性 . . . 2078
35.8.1 浮動小数点演算 . . . 2078
35.8.2 安定なソフトマックスと交差エントロピー . . . 2083
35.8.3 桁落ち . . . 2087
35.8.4 条件性 . . . 2093
35.8.5 まとめ . . . 2100
35.8.6 演習問題 . . . 2101
35.9 関連文献 . . . 2104
35.9.1 勾配法 . . . 2104
35.9.2 確率的手法と適応的手法 . . . 2105
35.9.3 制約と双対性 . . . 2106
35.9.4 浮動小数点 . . . 2107
35.9.5 最適化全般の参考資料 . . . 2108

36 確率と統計学習 **2109**
36.1 確率変数 . . . 2110
36.1.1 密度 . . . 2110
36.1.2 平均と分散 . . . 2116
36.1.3 分布の裾 . . . 2120
36.1.4 まとめ . . . 2125
36.1.5 演習問題 . . . 2125
36.2 同時分布と変換 . . . 2127

36.2.1 同時密度と条件付き密度 . . . 2128
36.2.2 共分散と相関 . . . 2134
36.2.3 確率変数の変換 . . . 2139
36.2.4 まとめ . . . 2143
36.2.5 演習問題 . . . 2144
36.3 分布 . . . 2145
36.3.1 離散分布 . . . 2146
36.3.2 連続分布 . . . 2153
36.3.3 多変量ガウス分布 . . . 2162
36.3.4 まとめ . . . 2166
36.3.5 演習問題 . . . 2167
36.4 指数分布族 . . . 2170
36.4.1 多様な分布を統一する単一の形式 . . . 2170
36.4.2 最大エントロピー . . . 2174
36.4.3 対数分配関数 . . . 2177
36.4.4 ソフトマックスの起源 . . . 2181
36.4.5 共役事前分布 . . . 2184
36.4.6 まとめ . . . 2189
36.4.7 演習問題 . . . 2189
36.5 統計学 . . . 2191
36.5.1 推定量とその標本分布 . . . 2192
36.5.2 仮説検定 . . . 2203
36.5.3 信頼区間 . . . 2212
36.5.4 ブートストラップ法 . . . 2216
36.5.5 まとめ . . . 2219
36.5.6 演習問題 . . . 2219
36.6 最尤法 . . . 2223
36.6.1 最尤法の原理 . . . 2224
36.6.2 負の対数尤度としての損失 . . . 2227
36.6.3 最尤推定量の精度 . . . 2230
36.6.4 MAP 推定と正則化 . . . 2239
36.6.5 まとめ . . . 2243
36.6.6 演習問題 . . . 2243
36.7 潜在変数、EM、ELBO . . . 2246
36.7.1 混合モデル . . . 2247
36.7.2 エビデンス下界 . . . 2248
36.7.3 期待値最大化法 . . . 2249
36.7.4 まとめ . . . 2254
36.7.5 演習問題 . . . 2254
36.8 ベイズ計算 . . . 2255
36.8.1 ベイズロジスティック回帰の事後分布 . . . 2256
36.8.2 棄却サンプリングと重要度サンプリング . . . 2263
36.8.3 マルコフ連鎖モンテカルロ法 . . . 2267
36.8.4 変分推論 . . . 2274
36.8.5 各手法の比較 . . . 2277
36.8.6 まとめ . . . 2279
36.8.7 演習問題 . . . 2280
36.9 集中不等式 . . . 2283
36.9.1 指数型の裾上界 . . . 2284
36.9.2 高次元における確率 . . . 2294

36.9.3 まとめ . . . 2299
36.9.4 演習問題 . . . 2300
36.10一様収束と汎化 . . . 2302
36.10.1一様収束 . . . 2302
36.10.2ラデマッハ複雑度 . . . 2306
36.10.3補間と二重降下 . . . 2314
36.10.4まとめ . . . 2319
36.10.5演習問題 . . . 2320
36.11ナイーブベイズ . . . 2323
36.11.1分類のためのベイズの定理 . . . 2324
36.11.2数え上げによる学習 . . . 2326
36.11.3数え上げによる Fashion-MNIST 分類 . . . 2329
36.11.4分類器の評価 . . . 2334
36.11.5まとめ . . . 2340
36.11.6演習問題 . . . 2340
36.12関連文献 . . . 2343
36.12.1確率モデル . . . 2344
36.12.2推定と検定 . . . 2345
36.12.3事後分布を用いた計算 . . . 2346
36.12.4集中と学習理論 . . . 2347
36.12.5カウントに基づく分類器 . . . 2348
36.12.6確率・統計の一般的な参考資料 . . . 2348

37 情報理論とダイバージェンス 2351
37.1 エントロピー、交差エントロピー、KL ダイバージェンス . . . 2352
37.1.1 情報量とエントロピー . . . 2353
37.1.2 KL ダイバージェンスと交差エントロピー . . . 2358
37.1.3 分類損失 . . . 2362
37.1.4 確率密度に対するエントロピーと KL ダイバージェンス . . . 2368
37.1.5 まとめ . . . 2372
37.1.6 演習問題 . . . 2373
37.2 符号、圧縮、パープレキシティ . . . 2375
37.2.1 記号符号 . . . 2376
37.2.2 ストリーム符号と記述長 . . . 2381
37.2.3 エントロピー率と典型系列 . . . 2386
37.2.4 パープレキシティ . . . 2390
37.2.5 まとめ . . . 2392
37.2.6 演習問題 . . . 2392
37.3 f ダイバージェンス . . . 2395
37.3.1 f ダイバージェンス族 . . . 2397
37.3.2 順方向 KL と逆方向 KL . . . 2404
37.3.3 全変動、ピンスカーの不等式、データ処理 . . . 2407
37.3.4 フェンシェル共役とクリティック下界 . . . 2412
37.3.5 まとめ . . . 2416
37.3.6 演習問題 . . . 2416
37.4 テスト関数と輸送 . . . 2418
37.4.1 積分確率指標と MMD . . . 2420
37.4.2 最適輸送とワッサースタイン距離 . . . 2425
37.4.3 固定カーネルと訓練されたクリティック . . . 2431
37.4.4 まとめ . . . 2432

37.4.5 演習問題 . . . 2432
37.5 スコアに基づく比較 . . . 2435
37.5.1 スコアとフィッシャーダイバージェンス . . . 2436
37.5.2 スタインの恒等式 . . . 2439
37.5.3 カーネルスタイン不一致度 . . . 2442
37.5.4 訓練目的関数と対応するダイバージェンス . . . 2445
37.5.5 まとめ . . . 2447
37.5.6 演習問題 . . . 2447
37.6 相互情報量 . . . 2449
37.6.1 従属性の尺度 . . . 2450
37.6.2 データ処理と予測 . . . 2457
37.6.3 通信と圧縮 . . . 2463
37.6.4 情報ボトルネック . . . 2466
37.6.5 まとめ . . . 2471
37.6.6 演習問題 . . . 2471
37.7 相互情報量の推定 . . . 2474
37.7.1 既知の情報量を持つペア . . . 2475
37.7.2 ノンパラメトリック推定 . . . 2476
37.7.3 変分下界 . . . 2483
37.7.4 InfoNCE . . . 2486
37.7.5 まとめ . . . 2496
37.7.6 演習問題 . . . 2496
37.8 関連文献 . . . 2499
37.8.1 エントロピーと損失 . . . 2499
37.8.2 符号と圧縮 . . . 2500
37.8.3 分布の比較 . . . 2502
37.8.4 相互情報量 . . . 2504
37.8.5 相互情報量の推定 . . . 2505
37.8.6 情報理論全般の参考資料 . . . 2506

38 ダイナミクス：微分方程式と生成フロー 2509
記法 . . . 2510
38.1 常微分方程式 . . . 2511
38.1.1 ベクトル場とフロー . . . 2512
38.1.2 線形 ODE と安定性 . . . 2518
38.1.3 まとめ . . . 2526
38.1.4 演習問題 . . . 2527
38.2 数値ソルバ . . . 2529
38.2.1 前進オイラー法 . . . 2529
38.2.2 ルンゲ・クッタ法 . . . 2531
38.2.3 硬さと陰的手法 . . . 2536
38.2.4 ソルバとしての勾配降下法 . . . 2538
38.2.5 まとめ . . . 2541
38.2.6 演習問題 . . . 2542
38.3 ニューラル ODE、随伴法、連続フロー . . . 2544
38.3.1 ニューラル ODE . . . 2546
38.3.2 随伴法 . . . 2551
38.3.3 連続正規化フロー . . . 2556
38.3.4 まとめ . . . 2562
38.3.5 演習問題 . . . 2563

38.4 確率微分方程式 . . . 2566
38.4.1 ブラウン運動 . . . 2567
38.4.2 伊藤解析 . . . 2570
38.4.3 オルンシュタイン・ウーレンベック過程 . . . 2580
38.4.4 シミュレーション . . . 2584
38.4.5 まとめ . . . 2589
38.4.6 演習問題 . . . 2589
38.5 フォッカー・プランク方程式と確率フロー . . . 2592
38.5.1 経路から密度へ . . . 2593
38.5.2 フォッカー・プランク方程式 . . . 2595
38.5.3 スコア関数 . . . 2602
38.5.4 連続の式と確率フロー ODE . . . 2605
38.5.5 時間反転 . . . 2612
38.5.6 まとめ . . . 2619
38.5.7 演習問題 . . . 2619
38.6 スコアマッチングと拡散モデル . . . 2623
38.6.1 スコアの学習 . . . 2623
38.6.2 スコアを用いたサンプリング . . . 2631
38.6.3 複数ノイズレベルにわたるスコア . . . 2634
38.6.4 決定論的サンプリングとガイダンス . . . 2638
38.6.5 まとめ . . . 2645
38.6.6 演習問題 . . . 2645
38.7 フローマッチングと最適輸送 . . . 2648
38.7.1 フローマッチング . . . 2649
38.7.2 Rectified Flow . . . 2656
38.7.3 最適輸送と直線性 . . . 2665
38.7.4 学習された速度場によるサンプリング . . . 2669
38.7.5 まとめ . . . 2673
38.7.6 演習問題 . . . 2673
38.8 関連文献 . . . 2676
38.8.1 ソルバ . . . 2676
38.8.2 連続時間モデル . . . 2677
38.8.3 ノイズ . . . 2678
38.8.4 学習される生成器 . . . 2680
38.8.5 ダイナミクスと生成フロー全般の参考資料 . . . 2681

深層学習のためのツール 2685

39 深層学習のためのツール 2687
参考資料と関連文献 . . . 2688
39.1 ノートブック . . . 2689
39.1.1 ノートブックの実行モデル . . . 2689
39.1.2 ローカル環境での本書の実行 . . . 2691
39.1.3 エディタでの作業 . . . 2692
39.1.4 リモートマシン . . . 2693
39.1.5 まとめ . . . 2694
39.1.6 演習問題 . . . 2694
39.2 Colab と Kaggle . . . 2696
39.2.1 本書のノートブックの起動 . . . 2696
39.2.2 Colab . . . 2698

39.2.3 Kaggle . . . 2699
39.2.4 ポータブルなノートブック . . . 2700
39.2.5 まとめ . . . 2701
39.2.6 演習問題 . . . 2702
39.3 クラウドコンピューティング . . . 2704
39.3.1 レンタル市場 . . . 2704
39.3.2 レンタルマシンでの作業 . . . 2707
39.3.3 まとめ . . . 2708
39.3.4 演習問題 . . . 2708
39.4 ハードウェア . . . 2710
39.4.1 ワークロードの要件 . . . 2711
39.4.2 訓練向けのディスクリート GPU . . . 2712
39.4.3 推論向けのユニファイドメモリマシン . . . 2714
39.4.4 ワークステーション GPU . . . 2715
39.4.5 最新数値の情報源 . . . 2716
39.4.6 まとめ . . . 2716
39.4.7 演習問題 . . . 2717
39.5 エコシステム . . . 2719
39.5.1 モデル、データ、論文の入手先 . . . 2720
39.5.2 ベンチマークとリーダーボード . . . 2721
39.5.3 情報源 . . . 2721
39.5.4 外部アーティファクトの管理 . . . 2722
39.5.5 まとめ . . . 2724
39.5.6 演習問題 . . . 2724
39.6 分散モデル訓練 . . . 2726
39.6.1 並列性の形態 . . . 2726
39.6.2 ソフトウェアライブラリ . . . 2729
39.6.3 長時間実行における耐障害性 . . . 2730
39.6.4 まとめ . . . 2731
39.6.5 演習問題 . . . 2731
39.7 モデルサービング . . . 2733
39.7.1 ワークロードとレイテンシ指標 . . . 2734
39.7.2 ローカルサービング . . . 2735
39.7.3 サーバーエンジン . . . 2735
39.7.4 エンジンの技法 . . . 2736
39.7.5 運用 . . . 2738
39.7.6 まとめ . . . 2738
39.7.7 演習問題 . . . 2739
39.8 貢献者ガイド . . . 2742
39.8.1 ビルドの仕組み . . . 2742
39.8.2 Git の基本 . . . 2744
39.8.3 コーディングエージェント . . . 2744
39.8.4 変更の提出 . . . 2745
39.8.5 まとめ . . . 2746
39.8.6 演習問題 . . . 2746

参考文献 **2749**

コード・数学・ディスカッションを備えた対話型の深層学習書。

まえがき

第 2 版へのまえがき

振り返ってみると、初版はまるで別の世界で書かれたかのように感じられる。本文、図表、コードはすべて手作業で作られ、スタイルガイドは共著者たちによって熱心に徹底され、新しいフレームワークを追加するには熟練した専門家の手が必要であった。具体的には、Anirudh Dagar（PyTorch と JAX）および Yuan Tang（TensorFlow）の尽力によるものである。本書が 1,000 ページを超え、何百もの GitHub の Issue や PR、Discourse のコメントが寄せられるようになると、すべてを最新の状態に保つことはフルタイムの仕事になった。その結果、著者全員がそれぞれ別の場へと移った後、本書は 5 年間にわたりそのまま置かれることとなった。わかりやすいコード、例を通じた解説、ノートブック形式といった本書の前提の多くは、時の試練に耐え抜いた。これらは 2020 年当時と変わらず今も新鮮である。内容のおよそ半分は今なお色あせていないが、残りの部分はその後、はるかに魅力的な問題や手法、応用によって取って代わられた。ハードウェアとそれに伴うソフトウェアツールが飛躍的な進歩を遂げたため、コードについては D2L ライブラリの形式を除けば、そのまま通用するものはほとんど残らなかった。

2026 年に起きた 2 つの出来事をきっかけに、このプロジェクトを再び手に取り、D2L を「改装」できるか試してみることにした。第 1 に、ギリシャの出版社から、本書をギリシャ語に翻訳して印刷・出版することに同意してもらえないかという依頼を受けたことである。最後に改良を加えてから 5 年が経ってもなお、新たに書籍として印刷したいと考える人がいるほどの価値が本書に残っていることを、この依頼は示していた。第 2 に、誰もが目の当たりにしてきたエージェント型 AI の急速な進歩により、Anthropic、OpenAI、Google のツールを使えば、以前なら到底手が回らなかった大規模な更新や修復を劇的に容易に行えるようになったことである。これらのツールのおかげで、コードを現代のフレームワークに合わせて刷新し、未対応だった GitHub の Issue や PR を取り込み、Discourse スレッドのバグ報告やコメントを確認し、コード・文章・実験・数式の整合性を検証して修正できた。出版スタックも刷新しており、現在本書では Quarto と uv を使用し、ディスカッションフォーラムは Discourse 自体のサーバー上でホストしている。さらにエージェントの活用により、すべてのフレームワークが同じ問題に対して同等の性能を発揮するように揃えられた。すなわち、不適切な JIT コンパイルの利用、初期化の違い、最適化スケジュール、あるいは「魔法の」乱数シードの使用に起因する性能低下を解消したのである。これは以前には望むべくもなかった贅沢だが、全編にわたって各フレームワークのベストプラクティスを示せるという点で、教育上の質を大きく高めていると確信している。

こうした基盤の刷新と並行して、内容自体も大きく進化した。PDF で 3,000 ページを超える規模となった本書は、もはや単一巻の枠に収まらなくなり、現在はモジュール化された複数のパートで構成されている。基礎パートでは深層学習の中核となる土台を築き、入門講義にそのまま使える自己完結した伴走教材となるよう設計されている。発展パートでは、最適化、状態空間モデル、Transformer、強化学習、GAN、拡散モデルなど多くの分野について、特定のテーマに深入りしすぎることなく現在の最先端に近い水準まで導く。これらの各分野における情報爆発を考えれば、すべてに深く踏み込むことは不可能だからである。深層学習のための数学パートも大幅に拡充さ

れ、拡散とフロー、情報理論、最適化をより手厚く扱うようになった。このパートは本分野の入門的な数学講義の教科書としても十分に活用でき、論文（あるいは本書の一部の章）を読む際に当然の前提とされがちな背景知識を補う助けとなる。強化学習と拡散モデルの章を寄稿してくれた Rasool Fakoor に感謝する。

やるべきことはまだ数多く残っている。言語モデルパートは未執筆であり、本書では現代のコンピュータビジョン（「画像モデル」パートは初版から引き継いだものである）、グラフィックス、オーディオ、音声、半教師あり学習をまだ扱っておらず、エージェントについての詳細な解説も備えていない。少しの時間と大量のトークン、そして惜しみないキュレーションがあれば、後者（エージェント）が前者（本書）を完成させる助けとなってくれるだろう。さらにその仕組みは、新たな科学的発見や新手法の発明があるたびに本書を最新の状態に保つ上でも役立つはずである。

Alex Smola、2026 年 11 月

初版へのまえがき

ほんの数年前まで、大企業やスタートアップで知的な製品やサービスを開発する深層学習の研究者・技術者の大群など存在しなかった。筆者らがこの分野に足を踏み入れた頃、機械学習が日刊紙の見出しを飾ることはなかった。筆者らの両親は機械学習とは何かまったく知らず、まして医学や法曹の道よりもそちらを選ぶ理由など見当もつかなかった。機械学習は基礎研究志向の学術分野であり、産業上の重要性は音声認識やコンピュータビジョンなど限られた実世界の応用にとどまっていた。しかも、そうした応用の多くは膨大なドメイン知識を必要としたため、機械学習はそのごく一部を占めるにすぎない、まったく別の専門分野とみなされることが多かった。当時、本書で焦点を当てる深層学習手法の前身であるニューラルネットワークは、一般に時代遅れとみなされていたのである。

ところがわずか数年のうちに、深層学習は世界を驚かせ、コンピュータビジョン、自然言語処理、自動音声認識、強化学習、生物医学情報学といった多様な分野で急速な進歩を牽引するようになった。さらに、実用上重要な数々のタスクにおける深層学習の成功は、理論的な機械学習や統計学の発展さえも促している。こうした進歩を手にしたいま、かつてないほど高い自律性で走行する自動運転車（もっとも、一部の企業が宣伝するほどの自律性には達していないが）、確認の質問を投げかけながらコードをデバッグする対話システム、さらには囲碁のようなボードゲームで世界最高峰の人間の棋士を破るソフトウェアエージェント（かつては何十年も先のことと考えられていた偉業である）を構築できるようになった。すでにこれらのツールは産業と社会にますます幅広い影響を及ぼしており、映画制作や疾病診断のあり方を変え、天体物理学から気候モデリング、気象予測、生物医学に至るまで、基礎科学においても存在感を高めつつある。

Aston Zhang、Zach Lipton、Mu Li、Alex Smola、2021 年

本書について

本書は、**概念・文脈・コード**を 1 か所で学ぶことで、深層学習を親しみやすいものにする試みである。

コード、数式、HTML を統合した単一の媒体

いかなるコンピューティング技術も、その真価を十分に発揮するには、よく理解され、十分に文書化され、成熟して適切に保守されたツールによって支えられていなければならない。核心となる考え方は明確に整理され、新たな実務者が最前線に追いつくまでの時間が最小限に抑えられているべきである。また、成熟したライブラリが定型的な作業を自動化し、模範となるコードによ

って、実務者が標準的なモデルを自身の目的に合わせて容易に改変・応用・拡張できなければならない。

動的なウェブアプリケーションはその好例である。Amazon などの企業は 1990 年代にデータベース駆動型のウェブアプリケーションを開発して成功を収めたが、この技術が創造的な起業家を支援する幅広い可能性を解き放ったのは、続く 10 年間に強力で十分に文書化されたウェブフレームワークが登場してからのことであった。

深層学習の可能性を引き出すには特有の難しさがある。というのも、どの応用ひとつをとっても複数の専門分野が絡み合うからである。深層学習を応用するには、(i) 問題を特定の形で定式化する動機、(ii) 与えられたモデルの数学的形式、(iii) モデルをデータに適合させる最適化アルゴリズム、(iv) モデルが未知のデータにいつ汎化（generalization）すると期待できるかを示す統計的原理と、実際に汎化したことを検証する実践的手法、そして (v) 数値計算の落とし穴を回避し、利用可能なハードウェアを最大限に活かしてモデルを効率的に訓練するエンジニアリング技術、これらすべてを同時に理解する必要がある。問題を定式化するための批判的思考力、それを解くための数学、そしてその解法を実装するためのソフトウェアツールを 1 か所で教えることは容易ならざる挑戦である。本書における目標は、意欲ある実務者が迅速に実力を身につけられる統合的なリソースを提供することにある。

本書のプロジェクトを開始した当時、(i) 常に最新の状態に保たれ、(ii) 現代の機械学習の広がりを十分な技術的深さで網羅し、(iii) 教科書に求められる厳密な解説と実践的なチュートリアルに求められる明快で実行可能なコードとを織り交ぜたリソースは、単一のものとしては存在しなかった。特定の深層学習フレームワークの使い方（たとえば TensorFlow における基本的な行列計算）や、特定のアーキテクチャ（LeNet、AlexNet、ResNet など）の実装方法を示すコード例は、ブログ記事や GitHub リポジトリの随所に数多く見られた。しかし、そうした例の多くは手法を**どのように**実装するかは示していても、アルゴリズムの設計上**なぜ**その選択がなされたのかは省かれていた。Distill や個人ブログに掲載されたインタラクティブな記事は、特定のトピックを視覚的に掘り下げてはいたものの、扱う範囲は分野の一部に限られ、コードが伴わないことも多かった。一方、深層学習の基礎を包括的に概観した Goodfellow et al. (2016) のように、当時登場していた深層学習の教科書は数学的な記述を動作する実装と結びつけておらず、理論とコードの橋渡しに戸惑う読者も少なくなかった。さらに、あまりに多くの学習リソースが商用講座のペイウォールの向こう側に閉ざされていた。

そこで本書では、次のようなリソースを作ることを目指した。すなわち、(i) 誰もが無料で利用でき、(ii) 応用機械学習の研究者・技術者へと飛躍するための足がかりとなる十分な技術的深さを備え、(iii) 実際に問題を**どのように**解くかを読者に示す実行可能なコードを含み、(iv) 著者陣およびコミュニティ全体による迅速な更新が可能であり、(v) 技術的な詳細や未解決の疑問について双方向に議論できるディスカッションフォーラムによって補完されるリソースである。

これらの目標はしばしば互いに相反した。数式、定理、引用の執筆と組版には LaTeX が最も適している。コードは Python で表現するのが最も自然である。そしてウェブページの本来の基盤は HTML と JavaScript である。さらに、実行可能なノートブック、印刷書籍、ダウンロード可能な PDF、インタラクティブなウェブサイトといういずれの形態でも同等にアクセスできるようにしたかった。これらすべての要求を満たす既存の出版ワークフローは存在しなかったため、独自の仕組みを構築した（39.8.4 節）。ソースの管理とコミュニティからの貢献の調整には GitHub を、文章・数式・コードを混在させそこから Jupyter ノートブックを自動生成する単一のソース形式には Markdown を、レンダリングエンジンには Quarto を、そして議論のプラットフォームには Discourse を採用した。完璧なシステムは存在しないものの、これらの選択は相反する要求の間で効果的なバランスを実現している。**Dive into Deep Learning** は、このような統合ワークフローを用いて出版された最初の書籍であると考えられる。

実践による学習

多くの教科書は概念を厳密な順序で提示し、各トピックを徹底的に掘り下げてから次へと進む。たとえば、Bishop (2006) による古典的な教科書は各主題をきわめて網羅的に扱っているため、線形回帰（linear regression）の章にたどり着くだけでも相当な労力を要する。専門家が同書を高く評価するのはまさにその網羅性ゆえであるが、この構成は初心者向けの入門書としては敷居を高くしている。

本書では、ほとんどの概念を**ジャストインタイム**で導入する。言い換えれば、具体的で実践的な目標を達成するために必要となったまさにその瞬間に、新しい考え方に触れることになる。冒頭の短い章で線形代数や確率といった不可欠な基礎事項を扱い、さらに深く学びたい読者のために包括的な数学の付録も用意しているが、より抽象的な理論に頭を悩ませる前に、まずは実際に動く最初のモデルを訓練する達成感を味わってほしいと考えている。

基本的な数学的背景を短期集中で学ぶ冒頭の数本のノートブックを除けば、以降の各章では無理のない範囲の新しい概念を、実データセット上で動作する自己完結した例とともに導入する。この方針には構成上の難しさも伴った。論理的には 1 つのノートブックにまとめられるモデル群もあれば、複数のモデルを順に実行することで最もよく理解できる考え方もあるからである。それでも、**1 つの動作例につき 1 つのノートブック**という方針を貫くことには大きな実用的利点がある。本書のコードをそのまま土台にして、読者自身の研究プロジェクトや開発プロジェクトをかつてないほど容易に立ち上げられるのである。ノートブックをコピーし、書き換え始めるだけでよい。

本書全体を通して、実行可能なコードと理論的背景を必要に応じて織り交ぜている。原則として、仕組みの細部をすべて説明する前にまずツールを読者の手に渡し、実際に動く様子を確認してからより深い背景を補う方針をとる。たとえば、初期のモデルの訓練には**確率的勾配降下法**（stochastic gradient descent）を用いるが、それがなぜこれほど効果的に機能するのかを詳しく分析するのはずっと後の章である。このアプローチには、途中の教育上の選択をいくつか信頼してもらうというわずかな代償こそ伴うものの、実務者が素早く問題を解くために必要なツールをいち早く手にできるという利点がある。

また本書では、深層学習の中核となる概念をゼロから実装しながら学ぶ。現代の深層学習フレームワークが通常はユーザーから隠蔽しているモデルの細部にも、しばしば踏み込んで中身を確かめる。とりわけ基礎を扱う各章では、層、損失関数、オプティマイザの内部で起きていることすべてを理解できるよう、こうした解説を重視している。その際、2 つの実装を並べて示すのが通例である。1 つは基本的なテンソル演算と自動微分（automatic differentiation）のみに依拠してゼロから構築する実装であり、もう 1 つは深層学習フレームワークの高レベル API を用いて簡潔に記述した実装である。ある構成要素の仕組みを一から説明した後は、以降の章ではフレームワークの高レベル実装を利用する。

内容と構成

本書は 3 つの短い導入章から始まる。インストール ではコードを実行するための環境構築の手順を説明し、記法 では本書の数学的記法をまとめ、イントロダクション では深層学習の全体像を概観する。本書の残りの部分は 7 つのパートで構成されており、中核をなす「基礎」と「発展」の 2 つのパートの構成を Figure 1 に示す。

- **基礎**。第 1 章では、データをテンソルとして格納・操作する方法、導関数を自動計算する方法、確率と不確実性を扱う考え方など、実践的な深層学習に欠かせない前提知識を手早く身につける。続く第 2 章と第 3 章では、線形モデルをデータに適合させながら、本書のほぼすべての手法に共通する 3 つの基本要素、すなわちパラメータ化されたモデル、その誤差を測る損失関数、そして損失を減らす最適化アルゴリズムを導入する。第 4 章では隠れ層（hidden layer）を追加して深層ネットワークを構築し、これほど多くのパラメータを持つ

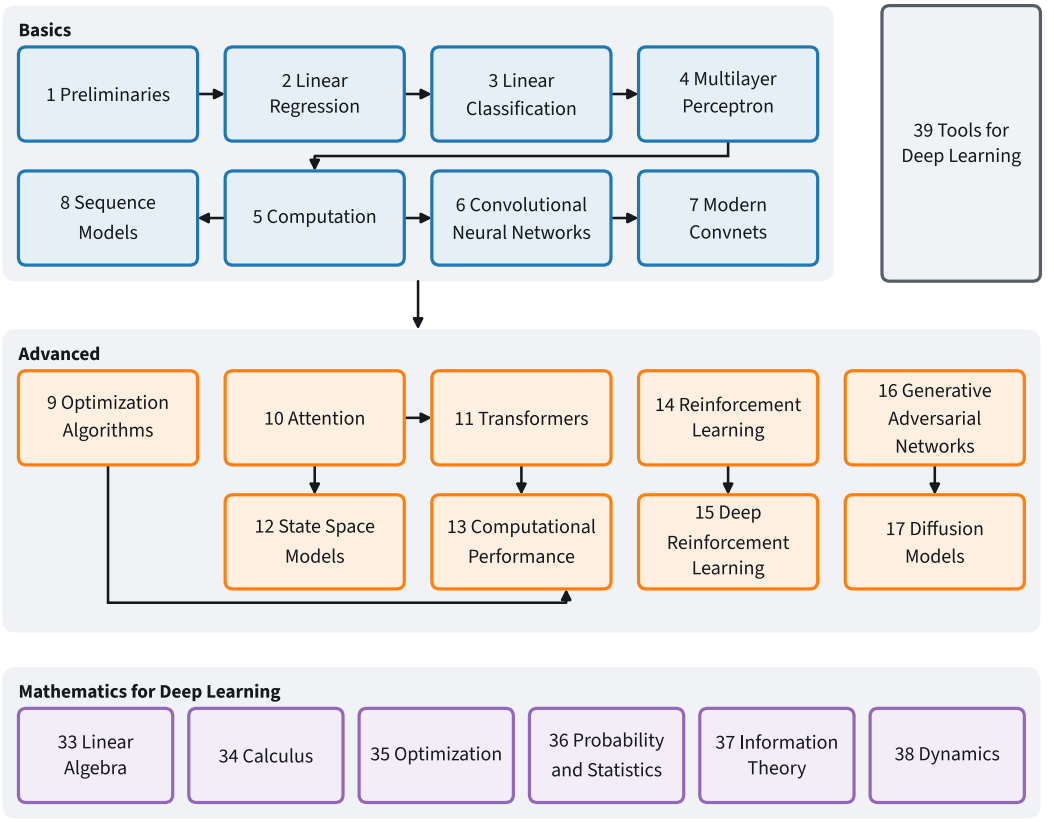


Figure 1: 本書の構成。ある章から別の章への矢印は、後者が前者を前提とすることを意味し、「基礎」パートから「発展」パートへの矢印は、「発展」パートのすべての章が「基礎」パートを前提とすることを意味する。「言語モデル」、「画像モデル」、「アーカイブ」の各パートは省略している。

モデルがなぜ新しいデータに対しても汎化できるのかを問う。次に第 5 章では、再利用可能なモジュールからのモデルの組み立て、初期化、保存、GPU による高速化など、深層学習ソフトウェアの仕組みに目を向ける。本パートの最後の 3 章では、領域固有の構造をアーキテクチャに直接組み込む。すなわち、第 6 章と第 7 章の畳み込みニューラルネットワーク（convolutional neural network、CNN）は画像の空間的対称性を活用し、第 8 章のリカレント系列モデルはテキストや時系列の順序構造を捉える。最初の 5 章は順番に読むのが望ましいが、その後の画像に関する章と系列に関する章は互いに独立して読むことができる。

- **発展**。これらの章では、「基礎」パートの土台の上に立ち、今日の最前線で用いられているアーキテクチャと訓練手法を展開する。読む順序は複数あり得、Figure 1 の矢印が章間の依存関係を示している。「基礎」パートではオプティマイザ、学習率スケジュール、バッチサイズを主に実用上のレシピとして扱ったのに対し、第 9 章ではそれらがなぜ機能し、モデルの大規模化に伴ってどう適応させるべきかを検証する。第 10 章では、系列内の各位置が他の任意の位置から直接情報を取得できるようにする**アテンション**機構を導入し、第 11 章ではこれらの層を積み重ねて、現代の大規模言語モデルを支える Transformer アーキテクチャを構築する。第 12 章では状態空間モデルを通じて再帰性を再考し、固定サイズの状態を保ちながらも系列のすべての位置にわたって並列に訓練する仕組みを学ぶ。第 13 章では Transformer を一貫した例として用い、単一 GPU および複数 GPU にまたがる訓練ステップの速度を何が律速するのかを探る。第 14 章と第 15 章では教師あり学習から離れ、環境との相互作用を通じて報酬から学習するエージェントへと軸足を移し、厳密に解ける小さな表形式の問題から始めてニューラルネットワークによる大規模化へと進む。最後に第 16 章と第 17 章では生成モデルを扱い、生成器と識別器の間のミニマックスゲーム、あるいはデータ破壊過程を反復的に逆転させる学習によって新しいデータを合成する方法を学ぶ。
- **言語モデルと画像モデル**。現在準備中の「言語モデル」パートでは、「発展」パートの手法を現代の大規模言語モデルへと拡張し、エンコーダ・埋め込み・検索（第 19 章（英語版））、事前学習（第 21 章（英語版））、ファインチューニング（第 24 章（英語版））、選好と検証可能な報酬によるアライメント（第 25 章（英語版））、エージェント（第 27 章（英語版））、そして効率的なサービング（第 30 章（英語版））を扱う。「画像モデル」パート（第 32 章（英語版））では物体検出、セマンティックセグメンテーション、ニューラルスタイル変換を取り上げており、初版から引き継がれた内容として今後全面的な改稿を予定している。
- **深層学習のための数学**。この付録では、本編の根底にある数学的基礎を展開する。本編の各章では数学的な結果を直観とともに述べるにとどめ、完全な証明や導出、より深い背景についてはこの付録を参照する形をとっている。第 33 章では幾何学的な射影、固有値分解、特異値分解を扱い、第 34 章では多変数の導関数、勾配、自動微分の仕組みを展開する。第 35 章では、曲率がなぜ学習率の上限を定めるのかといった、第 9 章で観察される経験的な振る舞いの背後にある理論的保証を与える。第 36 章では統計的推定、ベイズ推論、汎化限界を扱い、第 37 章では交差エントロピーが何を測っているのか、そしてダイバージェンスや最適輸送によって確率分布をどのように比較できるのかを説明し、第 38 章では拡散ベースおよびフローベースの生成モデルの背後にある微分方程式と確率過程を展開する。あるトピックにすでに精通している読者は該当する章を読み飛ばすか参考資料として使えばよく、それ以外の読者にとっては、この付録が自己完結した入門教科書として機能するよう設計されている。
- **深層学習のためのツール**。第 39 章では、本書のノートブックをローカル環境やクラウド環境で実行する方法、GPU ハードウェアの選定とプロビジョニング、分散訓練のスケールアップ、本番環境でのモデルのサービング、本書への貢献方法といった実践的なエンジニアリングのワークフローを扱う。あわせて `d21` ライブラリの API リファレンス（オンライン付録「`d21` API リファレンス」）も収録している。
- **アーカイブ**。最後のパートには、ガウス過程、ハイパーパラメータ最適化、推薦システム

（第 40 章（英語版） から 第 42 章（英語版） まで）など、初版から引き継がれ今版では積極的な拡充の対象外となった章が収められている。

コード

本書のほとんどの節には実行可能なコードが含まれている。深層学習における重要な直観の多くは、コードを少しずつ書き換えて結果がどう変化するかを観察するという、手を動かす実験を通じてこそ最もよく養われると考えられる。理想的には、包括的な数学理論によって、あらゆるアーキテクチャやハイパーパラメータの調整がどのような振る舞いをもたらすかを事前に知ることができればよいだろう。しかし現実には、深層学習の実務者は形式的な理論に先んじて試行錯誤しなければならない場面が多い。急速な進歩にもかかわらず、多くの手法がなぜ有効なのかについての完全な理論的説明はいまだ得られていない。強い非線形性を持つネットワークを特徴づける数学は容易ではなく、汎化は単純な定式化を拒む実世界データの構造的性質に左右されることが多く、現代のアーキテクチャに対する体系的な理論研究は経験的な発見を今なお追いかけている段階にある。深層学習の理論が成熟するにつれて、本書の将来の版により深い形式的知見を取り入れていくことを楽しみにしている。

本書のコードは最大 4 つの深層学習フレームワークで提供されており、それぞれ独立したインタラクティブなタブで表示される。「基礎」パートと付録「深層学習のための数学」は PyTorch、JAX、TensorFlow、MXNet で実装されており、「発展」パートは PyTorch と JAX で実装されている。フレームワークの選択と動作環境の構築については インストール が手引きとなる。すべてのノートブックは共通インポートをまとめた標準セルから始まり、その内容は 節で解説している。本文の焦点を絞るため、このプリアンブルのセルはそこでのみ一度だけ表示し、以降の節では扱うトピックに直接関係するコードからすぐに始められるようにしている。

章ごとに定型コードを繰り返すのを避けるため、再利用可能な関数やクラスは軽量な `d2l` ライブラリにまとめている。本書全体を通して、`#@save` が付いたコードブロック（補助関数、基底クラス、標準インポートなど）は本文中で一度だけ定義・説明されると同時に `d2l` パッケージに保存され、後の章では `d2l` 経由で直接呼び出せるようになっている。これらの関数とクラスの完全な一覧はオンライン付録「`d2l` API リファレンス」に掲載している。フレームワークに依存しない `d2l` ライブラリの中核部分が要するのは、以下の標準的な Python インポートのみである。

```
import inspect
import collections
from collections import defaultdict
from dataclasses import asdict
from IPython import display
import json
import math
from matplotlib import pyplot as plt
from matplotlib import ticker
from matplotlib_inline import backend_inline
import os
import pandas as pd
import random
import re
import shutil
import sys
import tarfile
import time
import requests
```

```
import zipfile
import gzip
import hashlib
d2l = sys.modules[__name__]
```

現在表示されているのは本書の JAX 版であり、「基礎」パート、「発展」パート、および付録「深層学習のための数学」の実装が提供されている。JAX は純粋な数値計算用の Python 関数を変換することで動作し、自動微分、コンパイル、ベクトル化を施す。本書全体を通して、ニューラルネットワークのモデルは Flax の `nnx` モジュールを用いて定義し、Optax のオプティマイザで訓練する。コードは、インストール の環境設定で固定された JAX エコシステムの各バージョンを用いて実行している。これらのライブラリは急速に進化するため、新しいリリースで非互換性が生じた場合は、その固定された環境の使用を勧める。`d2l` ライブラリの JAX バックエンドでは、以下のインポートが追加される。

```
from functools import partial
from flax import nnx
import jax
from jax import numpy as jnp
from jax import grad, vmap
import numpy as np
import optax
import tensorflow as tf
from typing import Any, Callable
```

対象読者

本書は、現代の深層学習を実践的かつ着実に身につけたい学部生・大学院生、ソフトウェアエンジニア、研究者を対象に執筆されている。すべての概念を一から積み上げて解説するため、深層学習や機械学習に関する予備知識は必要ない。深層学習の手法を丁寧に説明するには数学とプログラミングの双方が欠かせないが、前提とするのは線形代数、微分積分、確率、および Python の基礎的な素養のみである。数学の前提知識を復習したい場合や、理論をより深く掘り下げたい場合には、付録の深層学習のための数学が本書で用いる数学のほぼすべてを網羅している。本編の各章では、形式的な数学的一般性よりも明快な直観と核心的な考え方を一貫して優先する。

数学的基礎をさらに深く学びたい読者には、いくつかの古典的な名著を強く勧める。Bollobás (1999) による『Linear Analysis』は線形代数と関数解析を深く扱い、『All of Statistics』は統計的推論への簡潔で明快な入門書であり (Wasserman 2013)、確率に関する Joe Blitzstein の教科書と講義は教育的にも珠玉の内容である。最後に、Python に初めて触れる場合は、コードを読み進める前にインタラクティブな Python チュートリアルに取り組んでおくことを勧める。

ノートブック、ウェブサイト、GitHub、フォーラム

本書の実行可能なノートブックはすべて、D2L.ai ウェブサイトおよび GitHub リポジトリからダウンロードできる。また、d2l.discourse.group では活発なコミュニティのディスカッションフォーラムも運営している。本書のどの節についても、疑問が生じた際にはノートブック末尾にある直通リンクから対応するディスカッションスレッドへアクセスできる。

謝辞

英語版と中国語版の双方に携わった何百人もの貢献者に深く感謝する。その尽力により内容は改善され、数々の貴重なフィードバックが寄せられた。本書は当初、MXNet を主たるフレームワークとして実装されていた。初期の MXNet コードの大部分をそれぞれ PyTorch 実装と TensorFlow 実装へ移植してくれた Anirudh Dagar と Yuan Tang に感謝する。2021 年 7 月以降、PyTorch を主たるフレームワークに据え、本書を PyTorch、MXNet、TensorFlow で再設計・再実装した。その新しい PyTorch コードの大部分を JAX 実装へ移植してくれた Anirudh Dagar に改めて感謝する。また、中国語版においてその PyTorch コードの大部分を PaddlePaddle 実装へ移植してくれた Baidu の Gaosheng Wu、Liujun Hu、Ge Zhang、Jiehang Xie に感謝する。出版社の LaTeX スタイルを PDF ビルドへ統合してくれた Shuai Zhang にも謝意を表する。

GitHub 上で初版をすべての読者にとってより良いものにしてくれた貢献者全員にも感謝する。その GitHub ID または氏名は次のとおりである（順不同）。alxnorden、avinashingit、bowen0701、brettkoonce、Chaitanya Prakash Bapat、cryptonaut、Davide Fiocco、edgarroman、gkutiel、John Mitro、Liang Pu、Rahul Agarwal、Mohamed Ali Jamaoui、Michael (Stu) Stewart、Mike Müller、NRauschmayr、Prakhar Srivastav、sad-、sfermigier、Sheng Zha、sundeepteki、topecongiro、tpdi、vermicelli、Vishaal Kapoor、Vishwesh Ravi Shrimali、YaYaB、Yuhong Chen、Evgeniy Smirnov、lgov、Simon Corston-Oliver、Igor Dzreyev、Ha Nguyen、pmuens、Andrei Lukovenko、senorcinco、vfdev-5、dsweet、Mohammad Mahdi Rahimi、Abhishek Gupta、uwsd、DomKM、Lisa Oakley、Bowen Li、Aarush Ahuja、Prasanth Buddareddygari、brianhendee、mani2106、mtn、lkevinzc、caojilin、Lakshya、Fiete Lüer、Surbhi Vijayvargeeya、Muhyun Kim、dennismalmgren、adursun、Anirudh Dagar、liqingnz、Pedro Larroy、lgov、ati-ozgur、Jun Wu、Matthias Blume、Lin Yuan、geogunow、Josh Gardner、Maximilian Böther、Rakib Islam、Leonard Lausen、Abhinav Upadhyay、rongruosong、Steve Sedlmeyer、Ruslan Baratov、Rafael Schlatter、liusy182、Giannis Pappas、ati-ozgur、qbaza、dchoi77、Adam Gerson、Phuc Le、Mark Atwood、christabella、vn09、Haibin Lin、jjangga0214、RichyChen、noelo、hansent、Giel Dops、dvincent1337、WhiteD3vil、Peter Kulits、codypenta、joseppinilla、ahmaurya、karolszk、heytitle、Peter Goetz、rigtorp、Tiep Vu、sfilip、mlxd、Kale-ab Tessera、Sanjar Adilov、MatteoFerrara、hsneto、Katarzyna Biesialska、Gregory Bruss、Duy–Thanh Doan、paulaurel、graytowne、Duc Pham、sl7423、Jaedong Hwang、Yida Wang、cys4、clhm、Jean Kaddour、austinmw、trebeljahr、tbaums、Cuong V. Nguyen、pavelkomarov、vzlamal、NotAnotherSystem、J-Arun-Mani、jancio、eldarkurtic、the-great-shazbot、doctorcolossus、gducharme、cclauss、Daniel-Mietchen、hoonose、biagiom、abhinavsp0730、jonathanhrandall、ysraell、Nodar Okroshiashvili、UgurKap、Jiyang Kang、StevenJokes、Tomer Kaftan、liweiwp、netyster、ypandya、NishantTharani、heiligerl、SportsTHU、Hoa Nguyen、manuel-arno-korfmann-webentwicklung、aterzis-personal、nxby、Xiaoting He、Josiah Yoder、mathresearch、mzz2017、jroberayalas、iluu、ghejc、BSharmi、vkramdev、simonwardjones、LakshKD、TalNeoran、djliden、Nikhil95、Oren Barkan、guoweis、haozhu233、pratikhack、Yue Ying、tayfununal、steinsag、charleybeller、Andrew Lumsdaine、Jiekui Zhang、Deepak Pathak、Florian Donhauser、Tim Gates、Adriaan Tijsseling、Ron Medina、Gaurav Saha、Murat Semerci、Lei Mao、Levi McClenny、Joshua Broyde、jake221、jonbally、zyhazwraith、Brian Pulfer、Nick Tomasino、Lefan Zhang、Hongshen Yang、Vinney Cavallo、yuntai、Yuanxiang Zhu、amarazov、pasricha、Ben Greenawald、Shivam Upadhyay、Quanshangze Du、Biswajit Sahoo、Parthe Pandit、Ishan Kumar、HomunculusK、Lane Schwartz、varadgunjal、Jason Wiener、Armin Gholampoor、Shreshtha13、eigen-arnav、Hyeonggyu Kim、EmilyOng、Bálint Mucsányi、Chase DuBois、Juntian Tao、Wenxiang Xu、Lifu Huang、filevich、quake2005、nils-werner、Yiming Li、Marsel Khisamutdinov、Francesco “Fuma” Fumagalli、Peilin Sun、Vincent Gurgul、qingfengtommy、Janmey Shukla、Mo Shan、Kaan Sancak、regob、AlexSauer、Gopalakrishna Ramachandra、Tobias Uelwer、Chao Wang、Tian

Cao、Nicolas Corthorn、akash5474、kxxt、zxydi1992、Jacob Britton、Shuangchi He、zhmou、krahets、Jie-Han Chen、Atishay Garg、Marcel Flygare、adtygan、Nik Vaessen、bolded、Louis Schlessinger、Balaji Varatharajan、atgctg、Kaixin Li、Victor Barbaros、Riccardo Musto、Elizabeth Ho、azimjonn、Guilherme Miotto、Alessandro Finamore、Joji Joseph、Anthony Biel、Zeming Zhao、shjustinbaek、gab-chen、nantekoto、Yutaro Nishiyama、Oren Amsalem、TianMaoMao、Amin Allahyar、Gijs van Tulder、Mikhail Berkov、iamorphen、Matthew Caseres、Andrew Walsh、pggPL、RohanKarthikeyan、Ryan Choi、Likun Lei。

本書の執筆にあたり惜しみない支援を提供してくれた Amazon Web Services、とりわけ Wen-Ming Ye、George Karypis、Swami Sivasubramanian、Peter DeSantis、Adam Selipsky、Andrew Jassy に感謝する。十分な時間やリソース、同僚たちとの議論、そして絶え間ない励ましがなければ、本書が世に出ることはなかっただろう。出版に向けた準備の過程では、Cambridge University Press から素晴らしい支援を受けた。担当編集者である David Tranah の尽力とプロフェッショナリズムに深く感謝する。

第 2 版への貢献者

また、過去 4 年間にわたり GitHub やディスカッションフォーラムを通じて誤植の報告、数学的・教育的な明確化の提案、リンク切れやコードのバグの修正、プルリクエストの投稿を寄せてくれた多くのコミュニティの参加者にも感謝する。その中には、Abhishek Gupta (@abhishek-gupta2707)、Akshaya Bali (@akshayabali)、Alankrit (@alankrit02)、Alexhead (@Alexhead)、Amirhossein (@amirhosein-mesbah)、Andrey (@v4lproik)、Andrii Oriekhov (@andriyor)、Ankit Kumar Singh (@ankitkumarsingh19091995)、Ashwinpathak20 (@ashwinpathak20)、Avinash (@avinashscs)、Baptiste (@baptiste-pasquier)、Basti564 (@Basti564)、Berke (@berkeoral)、Bhavesh Laddagiri (@bhaveshladdagiri)、Bjoernenz (@bjoernenz)、Brian Bartoldson (@bartoldson)、Catalin (@catalin-sipos)、Chaitanya (@chaitanyab2001)、Cuong V. Nguyen (@ncv)、Daniel (@dakenf)、Daniel (@dani2442)、Danishjeet Singh (@danishjeet)、dark_master (@Dark_Master)、Dd3607 (@Dd3607)、Denis Kazakov (@Denis-Kazakov)、Dilip (@DilipSCU)、doc_karl (@Doc_Karl)、DomonkosNagy (@DomonkosNagy)、Dongryeol Lee (@dongryeol-lee)、Dr.rer.nat (@Dr.rer.nat)、Duc (@nguyenduc1102)、Ebad Sayed (@ebadsayed)、Elvis (@elvisesp)、Emmett (@emmett9001)、EnesBerkSak (@EnesBerkSak)、Eurus (@Eurus-Holmes)、Evan (@evanmiller)、Evgenii (@kaiser179)、Fardin (@fardinabbasi)、Fenilgandhi (@fenilgandhi)、Fredrick (@FriskyFred)、Fzyzcjy (@fzyzcjy)、Georgy (@goshik92)、Glenn (@glennmatlin)、GoodMan (@GoodMan666)、Hannes (@hannesdornbusch)、Haocong Wang (@HaocongWang)、HappyB1rd (@HappyB1rd)、Harsh (@Harsh8604)、HenryJia (@HenryJia)、Himanshu (@himanshurawlani)、HyeongjinKim (@HyeongjinKim)、Inaki (@InakiLPP)、James (@james-Howard)、Jayanth (@jayanthjayanth)、Jcbl1 (@jcbl1)、Jiawei He (@h-jiawei)、Jinyang (@jinyangyuan)、Joe (@joekakone)、Jon (@jontonsoup4)、Jonathan (@JonathanRayner)、Juanlu (@Juanlu001)、Juha (@kamiju)、kishan_kc (@Kishan_KC)、L. Jia (@L.Jia)、Legin (@Legin2022)、Lieven (@lievengovaerts)、Lin (@Lin13579)、Lockon (@Lockon007)、Lorenzoh (@lorenzoh)、Luohao (@luohao-brian)、Mahmood (@mahmoodtaghavi)、Manimaran (@Manimaran-T)、Marc (@MarcBresson)、Mark (@Mark1001)、Martin (@mjirik)、Matthieu (@matthieubulte)、Mhamed (@mhamedbenhadj)、Michael (@michaelfonder)、Mieszko (@Mieszko4)、Minato (@Minato-namikaze)、Moises (@moises-b2)、MrDiver (@MrDiver)、Noam (@noamgot)、Notplus (@notplus)、OvO (@OvO)、Pan (@pan-x-c)、Phumtawan (@Phumtawan)、Piotrek (@piotrek-k)、prajwal_pitlehra (@Prajwal_Pitlehra)、Qaq (@qaq)、Ranis (@ranisalt)、Robert (@robert-abc)、Saketh (@sakethram)、Sam (@SamFancher)、Satvik (@Satvik-Garg)、Sepehr (@sepehr-gh)、Sepehr (@sepehrnoey)、Shichao (@shichaoy)、Shiva (@shiva-sankaran)、Smsxgz (@smsxgz)、Sravan (@sravan-s)、Stest2012 (@stest2012)、Tirth (@Tirth27)、Tushar (@tusharmaurya)、Verify (@verify-code)、Victor

(@victor-iyi)、Wj (@wj-Mcat)、Xinyu (@xinyu-dev)、Yibo (@yibotongxue) が名を連ねる。

さらに、フォーラムやリポジトリにおいて活発な参加やフィードバック、技術的な議論を寄せてくれた以下の読者にも感謝する。

Amritansh (@Amritansh)、Angad (@Angad_Sandhu)、anh_tuan_bui (@Anh_Tuan_Bui)、arijit_das (@Arijit_Das)、Artyom (@artyompal)、carlos_arriaga (@Carlos_Arriaga)、Daniel、David (@d_c)、Fan (@Fan_Gu)、Hanker (@Hanker)、Igor (@igormorgado)、Jeet (@Jeet)、Jonathan (@jonathan-code-r)、mahmoud_taha (@Mahmoud_Taha)、Marios (@marios_g)、Mark (@Mark_Hall)、max_ah (@Max_Ah)、Maxe (@Maxe)、Mugen (@Mugen)、Nishant (@Nishant_Thakur)、Porter (@Porter)、Raj (@Raj)、Rling (@Rling)、Sansui (@Sansui)。

インストール

本書を最大限に活用するには、実際にコードを動かし、書き換え、自分の手で実験してみるのがよい。すべての章は、実行可能な Jupyter ノートブックとして提供されている。ローカル環境の構築をできる限りスムーズにするため、対応する各フレームワーク向けに自己完結型の `.zip` バンドルを配布している（ページ上部の **Notebooks** メニューから入手できる）。各バンドルには、出力結果や図があらかじめ保存された状態でそのまま閲覧できるノートブック一式に加え、対応する `d2l` ライブラリのソースコード、および CPU 環境と GPU 環境の両方に対応したバージョン固定済みの uv ロックファイルが含まれている。

ノートブック内の計算済み出力を読むだけであれば深層学習ライブラリをインストールする必要はないが、コードを実行・変更するには Python、JupyterLab、および選択したフレームワークが必要になる。同梱の `uv` ロックファイルを使えば、数個のコマンドで動作確認済みの完全な環境をインストールできる。

フレームワークの選択

特に強いこだわりがなければ、本書のすべての章を網羅している唯一のフレームワークである **PyTorch** から始めることを勧める。「基礎」（第 1 章から第 8 章）、「画像モデル」（第 32 章（英語版））、「深層学習のための数学」（第 33 章から第 38 章）、「深層学習のためのツール」（第 39 章）の各パートは、4 つのフレームワークすべて（PyTorch、JAX、TensorFlow、MXNet）で実装されている。「発展」パート（第 9 章から第 17 章）は PyTorch と JAX で実装されており、「言語モデル」パート（第 18 章（英語版） 以降）は PyTorch で実装されている。

始めるにあたって特別なハードウェアは必要ない。最初のいくつかの章は、最近の一般的なノートパソコンの CPU で快適に動作する。GPU は、後半でより大規模な畳み込みネットワーク、Transformer、生成モデルを訓練するときに有用となる。

ダウンロード可能なノートブックのバンドルは、本ウェブサイトで公開されている本書の版と一致している。最新の開発版ソースファイル、ビルドスクリプト、開発環境を調べたい場合は、GitHub からリポジトリを直接クローンすることもできる。

```
git clone https://github.com/smolix/d2l-neu.git
cd d2l-neu
```

ただし、単にノートブックを読んで実行するだけであれば、フレームワーク別の `.zip` バンドルの方がはるかにサイズが小さく、ビルド時の依存関係も大幅に少なくて済む。

uv のインストール

本書では、ノートブック用の Python インタプリタとパッケージの依存関係の両方を uv で管理するため、Conda や既存の Python 環境は必要ない。macOS または Linux では、スタンドアロンインストーラ（またはシステムのパッケージマネージャ）を使って uv をインストールする。

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Windows PowerShell では、次のコマンドを実行する。

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

インストーラによって PATH が更新された場合は、新しいターミナルウィンドウを開き、uv が利用可能であることを確認する。

```
uv --version
```

本書のノートブック環境は Python 3.12 を対象としている。マシンに Python 3.12 がまだインストールされていない場合でも、uv が独立したインタプリタを自動的にダウンロードして管理する。

ノートブックのダウンロードと展開

ページ上部の **Notebooks** メニューから選択したフレームワークのアーカイブをダウンロードし、展開する。たとえば、macOS または Linux で PyTorch バンドルを展開するには、ダウンロード先のディレクトリで次のコマンドを実行する。

```
unzip d2l-pytorch.zip
cd d2l-pytorch
```

Windows では、File Explorer または PowerShell を使って `.zip` アーカイブを展開できる。以降のコマンドはすべて、展開されたフォルダの中で実行する。このフォルダには `pyproject.toml`、`d2l/` パッケージのソース、および uv ロックファイルが含まれている。

CPU 環境の構築

まずはセットアップを手早く確認するため、CPU 環境から始めることを勧める。展開したノートブックのディレクトリ内で、独立した Python 3.12 仮想環境を作成し、バージョン固定済みの CPU ロックファイルと同期する。

```
uv venv --python 3.12
uv pip sync pylock.cpu.toml
```

Apple Silicon 搭載の Mac で MXNet を実行する場合は、代わりに macOS 専用のロックファイルと同期する。

```
uv pip sync pylock.cpu-macos.toml
```

Linux 上の MXNet では、`pylock.cpu.toml` は Linux `x86-64` 用の wheel として、本プロジェクトでメンテナンスされている MXNet 2.0 リリースで配布されているものを使用する。なお、MXNet のビルドは Linux（`x86-64`）および macOS（Apple Silicon）向けにのみ提供されている。Windows では、PyTorch、JAX、TensorFlow を使用するか、ホスト型のクラウド環境でノートブックを実行されたい。PyTorch、JAX、TensorFlow については、`pylock.cpu.toml` がオペレーティングシステムとアーキテクチャに適したパッケージを自動的に選択する。

環境の同期が完了したら、JupyterLab を起動する。

```
uv run --no-sync jupyter lab
```

通常、JupyterLab はデフォルトのウェブブラウザで自動的にタブを開く。開かない場合は、ターミナルに表示されるローカルの `http://localhost:...` という URL をコピーして貼り付ける。`--no-sync` を指定することで、`uv` は直前に `uv pip sync` で同期した CPU 環境または GPU 環境の中でそのまま実行される。

「言語モデル」パート（第 18 章（英語版） 以降）向けに、PyTorch バンドル（`d2l-pytorch.zip`）には `pylock.lm-cpu.toml` と `pylock.lm-gpu.toml` も同梱されている。これらは PyTorch に加えて、Hugging Face の `transformers` や `accelerate`、OpenAI クライアント、および検索・データキュレーション用のライブラリを追加するものである。こうした追加の依存関係は別の `.venv-lm` 仮想環境に分離し、メインの `.venv` を軽量に保つ。

```
uv venv --python 3.12 .venv-lm
VIRTUAL_ENV=.venv-lm uv pip sync pylock.lm-cpu.toml
VIRTUAL_ENV=.venv-lm uv run --no-sync jupyter lab
```

GPU 環境の構築

同梱の GPU ロックファイルは、本書の実行に使用されたハードウェアアクセラレーション環境を再現するものであり、NVIDIA GPU を搭載した Linux（`x86-64`）マシンを対象としている。GPU 環境をインストールする前に、NVIDIA ドライバが有効で認識されていることを確認する。

```
nvidia-smi
```

展開したノートブックのディレクトリで、（まだ作成していない場合は）仮想環境を作成し、GPU ロックファイルと同期する。

```
uv venv --python 3.12
uv pip sync pylock.gpu.toml
uv run --no-sync jupyter lab
```

CPU から GPU へ切り替えるために別の仮想環境を用意する必要はない。既存の環境で `uv pip sync pylock.gpu.toml` を実行すれば、CPU パッケージが対応する GPU パッケージへとクリーンに置き換えられる（同様に、PyTorch バンドルの「言語モデル」環境も `VIRTUAL_ENV=.venv-lm uv pip sync pylock.lm-gpu.toml` でアップグレードできる）。GPU ロックファイルでは、次のアクセラレータスタックがバージョン固定されている。

- **PyTorch**：CUDA 12.8 の wheel
- **JAX**：CUDA 12 の pip ディストリビューション

- **TensorFlow**：`and-cuda` ディストリビューション（CUDA ライブラリ同梱）
- **MXNet**：メンテナンス版の CUDA 13.3 ビルド（`smolix/mxnet` 提供）

GPU のソフトウェアスタックはオペレーティングシステム、カーネルドライバ、GPU アーキテクチャに依存するため、バージョン固定済みのロックファイルが使用中のマシンの要件と異なる場合がある。`pylock.gpu.toml` がハードウェアや OS に合わない場合は、PyTorch、JAX、または TensorFlow の公式インストールガイドを参照されたい。序盤の章では CPU ロックファイルを使用し、本書の後半で計算負荷の高い章に進んだ段階で、ホスト型ノートブック（39.2 節）やクラウド GPU インスタンス（39.3 節）に切り替えることもいつでも可能である。

インストールの確認

フレームワークが正しくインストールされているか、また GPU アクセラレータが検出されているかは、1 行のコマンドで確認できる。

```
uv run --no-sync python -c "import jax; print(jax.__version__); print(jax.devices())"
```

CPU のみの環境では、`jax.devices()` は単一の CPU デバイスを表示し、NVIDIA GPU が利用可能な場合は代わりに GPU デバイスの一覧を表示する。

標準インポート

本書のすべてのノートブックは、標準のプリアンブルセルから始まる。このセルでは、スカラー関数や定数用の Python 組み込み `math` モジュール、CPU 配列用の NumPy（標準エイリアス `np`）、深層学習フレームワーク本体、そして `d2l` という名前で本書付属ライブラリの対応バックエンドをインポートする。

```
import math
import numpy as np
import jax
from jax import numpy as jnp
from flax import nnx
import optax
from d2l import jax as d2l
```

ここで、`jax` は関数変換（`jax.grad` や `jax.jit` など）および明示的な PRNG キー管理を提供し、`jnp` はアクセラレータに対応した NumPy 互換の配列インターフェースを公開している。ニューラルネットワークの層やモデルは Flax の `nnx` モジュールで構築し、`optax` のオプティマイザとスケジュールを使って訓練する。

最後のインポート文は、本書付属パッケージの対応バックエンドを共通の名前 `d2l` に結び付ける。各節で `#@save` マーク付きの再利用可能な補助関数やクラスを定義・解説すると、それらは `d2l` にエクスポートされ、後の章で定義を繰り返すことなく簡潔に呼び出せるようになる。

本書を見やすく保つため、この標準インポートセルは、まさにこの箇所に一度だけ掲載する。ダウンロード版およびホスト型の Jupyter ノートブックでは、各ノートブックが単体で動作するようすべてのファイルの先頭にこのセルが含まれているが、ウェブ版と PDF 版では各節がそのトピック固有のコードから直接始まる。ノートブックで追加のライブラリ（強化学習の章における Gymnasium など）が必要な場合や、フレームワークを初期化する前にコンパイラフラグを設定

する必要がある場合は、その追加のセットアップセルを明示的に掲載する。プロットのために特別な定型コードは必要なく、Jupyter はデフォルトで図をインライン描画する。

データ、モデル、ディスク容量

ノートブックの `.zip` アーカイブにはすべてのノートブック、計算済みのセル出力、および図が含まれているが、大規模な訓練データセットや事前学習済みモデルの重みは同梱されていない。その代わり、ノートブックは初回使用時にこれらのアセットを自動的にダウンロードし、ローカルのキャッシュディレクトリに保存する。そのため、ノートブックを読んで保存済みの出力を確認する作業は完全にオフラインでできるが、データを使用するノートブックを実行するにはインターネット接続と追加のディスク容量が必要になる。

最後に、本書後半の一部の発展的な章では、一般的なノートパソコンが備える以上のメモリや計算資源が必要になる点に留意されたい。基礎的な章では CPU 環境から始め、処理負荷に応じてローカル GPU やクラウドマシンへ移行するとよい。深層学習用の専用ハードウェアの購入を検討している場合は、39.4 節に選ぶ際のポイントをまとめた実践的なガイドがある。

記法

線形代数、微積分、確率、最適化、そして現代のニューラルアーキテクチャにまたがる本書において、明確で一貫した記法は、数学的な概念とコードとを結びつける大きな助けとなる。本書全体を通して、統一された組版上の階層に従う。すなわち、小文字の斜体（x）はスカラー、小文字の太字（$\mathbf{x}$）はベクトル、大文字の太字（$\mathbf{X}$）は行列、サンセリフ体の大文字（X）は高階のテンソル、カリグラフィー体（$\mathcal{X}$）は集合、大文字の斜体（X）は確率変数を表す。

以下の一覧には、**一般的な組版上の慣例**を示す項目（たとえば「x：スカラー」は、小文字の斜体が一般にスカラーを表すことを意味する）と、「$\mathbb{R}$：実数全体の集合」のように**特定の固定された意味**を持つ記号とが含まれる。

使える文字の数には限りがあるため、いくつかの標準的な文字は分野をまたいで複数の意味を持つ。いずれの場合も、記号の構文上の形式と文脈から意味は一意に定まる。たとえば、シグモイド関数 $\sigma(\cdot)$ は括弧付きの引数をとり、標準偏差 σ_X は確率変数を下付き添字に持ち、特異値 $\sigma_i(\mathbf{X})$ は添字付きで行列に適用され、拡散モデル（diffusion model）のノイズスケジュール σ_t は時刻 t を添字にとる。混同しやすい記号については、以下で明示的に注意を付す。また、分野固有の語彙が多く導入される章では、最適化（Table 35.1）、情報理論（Table 37.1）、ダイナミクス（Table 38.1）、強化学習（Table 14.1）、敵対的生成ネットワーク（Table 16.1）、拡散モデル（Table 17.1）をはじめ、章の冒頭に専用の記法一覧表を設けている。本章の後半の各節では、いくつかの発展的なトピックについて主要な記法をまとめる。

数値オブジェクト

- x：スカラー
- $\mathbf{x}$：ベクトル（デフォルトでは列ベクトル）
- $(x_1, \dots, x_n)$：成分 $x_1, \dots, x_n$ を持つ列ベクトルのインライン表記。地の文では「行 $(0, 1, 2)$ と $(3, 4, 5)$」のように、行列の行も同じ形式で表記する
- $\mathbf{X}$：行列
- X：2 つより多くの軸を持つテンソル
- $\mathbf{I}$：文脈から定まる次元の単位行列（主対角成分が 1 で、それ以外の成分がすべて 0 の正方行列）
- $\mathbf{0}$、$\mathbf{1}$：文脈から定まる次元の、すべての成分が 0 のベクトルおよびすべての成分が 1 のベクトル
- x_i：ベクトル $\mathbf{x}$ の i^{th} 要素
- x_{ij}、$[\mathbf{X}]_{i,j}$：行列 $\mathbf{X}$ の第 i 行第 j 列の要素

集合論

- $\mathcal{X}$：集合

- $\mathbb{R}$：実数全体の集合
- $\mathbb{R}^n$：n 次元実ベクトル全体の集合
- $\mathbb{R}^{a\times b}$：a 行 b 列の実数行列全体の集合
- $|\mathcal{X}|$：集合 $\mathcal{X}$ の濃度（要素数）
- $\mathcal{A}\cup\mathcal{B}$：集合 $\mathcal{A}$ と $\mathcal{B}$ の和集合
- $\mathcal{A}\cap\mathcal{B}$：集合 $\mathcal{A}$ と $\mathcal{B}$ の共通部分
- $\mathcal{A}\setminus\mathcal{B}$：差集合（$\mathcal{A}$ の要素のうち $\mathcal{B}$ に属さないものからなる集合）

関数と演算子

- $f(\cdot)$：関数
- $\log(\cdot)$：自然対数（底は e）
- $\log_2(\cdot)$：底を 2 とする対数
- $\exp(\cdot)$：指数関数
- $\sigma(\cdot)$：ロジスティックシグモイド関数 $\sigma(x)=1/(1+e^{-x})$
- $\mathrm{softmax}(\mathbf{o})$：各成分が $\exp(o_i)/\sum_j \exp(o_j)$ であるソフトマックスベクトル
- $\mathbf{1}(\cdot)$：指示関数（indicator function）。真偽値の引数が真のとき 1、それ以外のとき 0 をとる（引数をとることで、すべての成分が 1 のベクトル $\mathbf{1}$ と区別される）
- $\operatorname{argmax}_x f(x)$、$\operatorname{argmin}_x f(x)$：$x$ の値のうち、$f(x)$ を最大化または最小化するもの
- $\sum$：要素の集まりにわたる総和
- $\prod$：要素の集まりにわたる総乗
- $O(\cdot)$：$O(n^2)$ 回の演算コストのように、定数倍を除いた漸近的な上界
- $\propto$：比例関係（着目している変数に依存しない定数倍だけ両辺が異なることを表す）
- $\stackrel{\text{def}}{=}$：左辺の記号を定義する等号

線形代数

- $(\cdot)^\top$：ベクトルまたは行列の転置
- $\mathbf{X}^{-1}$：正方行列 $\mathbf{X}$ の逆行列
- $\mathbf{X}^{+}$：行列 $\mathbf{X}$ のムーア・ペンローズ擬似逆行列（pseudoinverse）
- $\langle\mathbf{x},\mathbf{y}\rangle$：ベクトル $\mathbf{x}$ と $\mathbf{y}$ の内積（ドット積）。$\mathbf{x}^\top\mathbf{y}$ とも表記する
- $\odot$：アダマール積（要素ごとの積）
- $[\cdot,\cdot]$：ベクトルまたは行列の連結（concatenation）
- $\|\cdot\|_p$：ベクトルの ℓ_p ノルム
- $\|\cdot\|$：ユークリッドノルム（ℓ_2 ノルム）
- $\|\mathbf{X}\|_2$：行列 $\mathbf{X}$ のスペクトルノルム（作用素ノルム）。最大の特異値に等しい
- $\|\mathbf{X}\|_F$：行列 $\mathbf{X}$ のフロベニウスノルム。全成分の ℓ_2 ノルムに等しい
- $\|\mathbf{X}\|_*$：行列 $\mathbf{X}$ の核ノルム（トレースノルム）。特異値の総和に等しい
- $\mathrm{diag}(\mathbf{x})$：ベクトル $\mathbf{x}$ の成分を主対角線上に並べた対角行列
- $\mathrm{rank}\,\mathbf{X}$、$\mathrm{tr}\,\mathbf{X}$、$\det\mathbf{X}$：行列 $\mathbf{X}$ のランク、トレース、行列式
- λ_i、$\lambda_{\max}(\mathbf{X})$：正方行列の固有値、および対称行列 $\mathbf{X}$ の最大固有値
- $\sigma_i(\mathbf{X})$：行列 $\mathbf{X}$ の大きい方から i^{th} の特異値（シグモイド関数 $\sigma(\cdot)$ と混同しないこと）
- $\kappa(\mathbf{X})$：行列 $\mathbf{X}$ の条件数（condition number）。最大の特異値と最小の非ゼロ特異値との比に等しい
- $\rho(\mathbf{X})$：正方行列 $\mathbf{X}$ のスペクトル半径（spectral radius）。固有値の絶対値（複素絶対値）の最大値に等しい（後述の相関係数 $\rho(X,Y)$ と混同しないこと）
- $\mathbf{X}\succeq 0$、$\mathbf{X}\succ 0$：対称行列 $\mathbf{X}$ が半正定値（positive semidefinite、$\mathbf{X}\succeq 0$）または正定値（positive definite、$\mathbf{X}\succ 0$）であること

微積分

- $\frac{dy}{dx}$、$f'(x)$：y の x に関する導関数、および関数 f の導関数の点 x における値
- $\frac{\partial y}{\partial x}$：$y$ の x に関する偏微分
- $\nabla_{\mathbf{x}} y$：スカラー y のベクトル $\mathbf{x}$ に関する勾配
- $\mathbf{J}$、$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$：ベクトル値関数 $\mathbf{f}$ の $\mathbf{x}$ に関するヤコビ行列
- $\nabla^2 f$：f の 2 階偏微分を並べたヘッセ行列
- $\nabla \cdot \mathbf{f}$、Δf：ベクトル場 $\mathbf{f}$ の発散（divergence）、およびスカラー関数 f のラプラシアン
- $\int_a^b f(x)\,dx$：関数 f の a から b までの x に関する定積分
- $\int f(x)\,dx$：関数 f の x に関する不定積分（原始関数）

確率と情報理論

- X：確率変数
- P：確率分布
- $X \sim P$：確率変数 X が分布 P に従うこと
- $P(X = x)$：確率変数 X が値 x をとる事象に割り当てられた確率
- $P(X \mid Y)$：Y が与えられたもとでの X の条件付き確率分布
- $p(\cdot)$：分布 P に対応する確率密度関数（probability density function、PDF）または確率質量関数（probability mass function、PMF）
- $\mathbb{E}[X]$：確率変数 X の期待値
- $\mathbb{E}_{x \sim P}[f(x)]$：$x$ が分布 P からサンプリングされるときの $f(x)$ の期待値
- $X \perp Y$：確率変数 X と Y が統計的に独立であること
- $X \perp Y \mid Z$：確率変数 X と Y が、Z が与えられたもとで条件付き独立であること
- σ、σ_X：一般の標準偏差、および特に確率変数 X の標準偏差
- $\mathrm{Var}(X)$：確率変数 X の分散。σ_X^2 に等しい
- $\mathrm{Cov}(X, Y)$：確率変数 X と Y の共分散
- Σ：共分散行列
- $\rho(X, Y)$：X と Y のピアソン相関係数。$\frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}$ に等しい
- $\mathcal{N}(\mu, \sigma^2)$：平均 μ、分散 σ^2 の単変量ガウス分布（正規分布）。$\mathcal{N}(\mu, \Sigma)$ はその多変量版を表す
- $H(X)$、$h(X)$：離散確率変数 X のシャノンエントロピー、および連続確率変数 X の微分エントロピー（differential entropy）
- $I(X; Y)$：確率変数 X と Y の相互情報量（mutual information）
- $D_{\mathrm{KL}}(P \| Q)$：分布 P から分布 Q へのカルバック・ライブラー（Kullback–Leibler、KL）ダイバージェンス（相対エントロピー）

データ、モデル、訓練

- n：データセットに含まれるデータ例の数
- d：入力特徴量（feature）の数（1 つのデータ例の次元数）
- $\mathbf{x}^{(i)}$、$y^{(i)}$：i^{th} のデータ例の特徴ベクトルと目的ラベル
- $\mathcal{D}$：$\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ などのデータセット
- $\mathbf{X} \in \mathbb{R}^{n \times d}$：1 行に 1 つのデータ例を並べた計画行列（design matrix）またはミニバッチ行列
- $\mathcal{B}$：サイズ $|\mathcal{B}|$ のデータ例のミニバッチ
- $\hat{y}$：目的値 y に対するモデルの予測

- $\mathbf{w}$、b：線形モデルの重みベクトルとスカラーバイアス
- $\mathbf{W}^{(\ell)}$、$\mathbf{b}^{(\ell)}$：第 ℓ 層の重み行列とバイアスベクトル（例：$\mathbf{W}^{(1)}$）
- θ、θ：モデルのパラメータ（1 つのベクトルにまとめる場合は太字で θ と表記する）
- l、ℓ：データ例ごとの損失関数（loss function）
- L：データセットまたはミニバッチ全体で平均した経験損失（empirical loss）
- η：勾配ベースのオプティマイザにおける学習率（learning rate、ステップサイズ）
- λ：ℓ_2 重み減衰（weight decay）ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$ の正則化強度
- t、T：離散または連続のタイムステップ t、および系列（sequence）の全長 T
- $\mathbf{x}_t$、$\mathbf{h}_t$：タイムステップ t における入力と隠れ状態（hidden state）

アテンションと Transformer

これらの記号は第 10 章で導入され、第 11 章およびそれに基づく後続の章全体で用いられる。

- $\mathbf{q}$、$\mathbf{k}_i$、$\mathbf{v}_i$：クエリベクトル、および項目 i のキーベクトルと値ベクトル（10.1 節）
- $a(\mathbf{q}, \mathbf{k})$：アテンションスコア関数（スケール化ドット積アテンションにおける $\mathbf{q}^\top \mathbf{k}/\sqrt{d}$ など）
- $\alpha(\mathbf{q}, \mathbf{k}_i)$：すべてのキーにわたるスコアにソフトマックス関数を適用して得られる、キー i に割り当てられたアテンション重み
- $\mathbf{Q}$、$\mathbf{K}$、$\mathbf{V}$：n 個のクエリ、m 個のキー、m 個の値を 1 行に 1 つずつ積み重ねた行列。スケール化ドット積アテンションはこれらを $\mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d})\,\mathbf{V}$ に写す（10.2 節）
- d、d_v：クエリとキーの次元（幅、d）、および値の次元（d_v）
- H、d_h：アテンションヘッドの数、およびヘッドごとの次元 $d_h = d/H$（10.3 節）

状態空間モデル

これらの記号は第 12 章で導入される。

- $\mathbf{A}$、$\mathbf{B}$、$\mathbf{C}$、D：状態空間モデル（12.4.1.1 節）における状態遷移行列、入力射影、出力読み出し（readout）、および直達項（feedthrough、スキップ項）
- N：チャネルごとの潜在状態の次元
- λ_n：対角状態行列の n^{th} の対角成分（レート）。対応する状態成分はステップごとに $e^{\Delta\lambda_n}$ 倍に減衰する
- Δ、Δ_t：離散化（discretization）のステップサイズ（12.4.1.2 節）、および Mamba における入力依存（選択的）なステップサイズ（12.5 節）
- $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$：外積 $\mathbf{v}_t\mathbf{k}_t^\top$ によって更新され、クエリ $\mathbf{q}_t$ で読み出される行列値の再帰状態（12.6 節）。ここでは、キーの次元を d_k と表記し、値の次元 d_v と並べて用いる
- β_t：デルタ更新則（12.7 節）における書き込み強度（学習率）

強化学習

これらの記号は第 14 章および第 15 章全体で用いられる。各記号が定義される節を含む完全な一覧は Table 14.1 に示す。

- $\mathcal{S}$、$\mathcal{A}$：状態空間と行動空間
- s_t、a_t、r_t：タイムステップ t における状態、行動、およびスカラー報酬（reward）
- $P(s' \mid s, a)$：状態 s で行動 a をとったときに状態 s' へ遷移する確率を与えるマルコフ遷移カーネル
- $r(s, a)$：状態 s で行動 a をとったときの期待即時報酬

- γ：割引率（discount factor）、$\gamma \in [0, 1)$
- τ、$R(\tau)$：軌跡（trajectory）$(s_0, a_0, r_0, s_1, ...)$、およびその割引収益（return）$R(\tau) = \sum_t \gamma^t r_t$
- $\pi(a \mid s)$：状態 s で行動 a を選択する確率を与える確率的方策（policy）。決定論的方策では、$\pi(s)$ が選択される行動を表す
- π_θ、$J(\theta)$：パラメータ θ によってパラメータ化された方策、およびその期待割引収益
- $V^\pi(s)$、$Q^\pi(s, a)$：方策 π のもとでの状態価値関数と行動価値関数（状態 s から開始したときの期待収益、および状態 s で行動 a をとったときの期待収益）
- V^*、Q^*：最適状態価値関数と最適行動価値関数
- $A^\pi(s, a)$：アドバンテージ関数 $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$
- δ_t：ステップ t における時間差分誤差（temporal-difference error、TD 誤差）
- ϵ：ϵ 貪欲な行動選択における探索（exploration）の確率

敵対的生成ネットワーク

これらの記号は第 16 章で用いられるものである。Table 16.1 には、すべての記号が定義箇所の節および文献でよく使われる別記法とともにまとめられている。

- p、q、m：真のデータ密度、生成器（generator）のモデル密度、およびそれらの等重み混合密度 $m = \frac{1}{2}(p + q)$
- $x \sim p$、$x' \sim q$：データ分布から得られた実サンプルと、生成器から得られた合成サンプル
- y：サンプルの出所を表す二値ラベル（実データでは $y = 1$、生成サンプルでは $y = 0$）
- z、G：潜在ノイズ変数と、$x' = G(z)$ を出力する生成器ネットワーク
- D：クリティック（または識別器）が出力する真実性のロジット。これにより $\sigma(D(x))$ はサンプル x が実データである推定確率を表す
- V：対数損失ミニマックスゲームの価値関数
- $\rho = p/q$、$\lambda = \log \rho$：尤度比（likelihood ratio、密度比）と対数密度比
- $\eta = p/(p + q)$：サンプルが実データであるベイズ最適クラス事後確率（なお第 16 章の中では、オプティマイザのステップサイズには h を用い、η は使わない）
- f、D_f：ダイバージェンス関数とその f ダイバージェンス
- Φ、d_{Rp}：ペアリング（相対論的、relativistic）目的関数と、最適クリティックにおけるその値
- γ、R_1、R_2：勾配ペナルティの重み、ならびに実データ上（R_1）および生成サンプル上（R_2）で評価されるゼロ中心の勾配ペナルティ
- c：クラスラベルやテキストプロンプトなどの条件付け変数

拡散モデル

これらの記号は第 17 章で用いられるもので、第 38 章（Table 38.1）の慣例と一致している。本書全体を通して、時間の向きは単一のものに統一する。すなわち、クリーンデータは $t = 0$ に、純粋なノイズは $t = 1$ にあり、生成サンプラは $t = 1$ から $t = 0$ へと逆向きに積分する（フローマッチングの論文の中にはこの向きを逆にするものもある。17.2 節を参照）。この章で使われる記号は、論文で用いられる表記とともに Table 17.1 にまとめられている。

- $\mathbf{x}_0$、ϵ：クリーンなデータ点と、独立な標準ガウスノイズのベクトル $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$：ガウス確率経路に沿った時刻 $t \in [0, 1]$ におけるノイズ付き状態
- α_t、σ_t：経路に沿った信号スケジュールとノイズスケジュールの係数（学習率やデータの標準偏差とは異なる）

- $\bar{\alpha}_t = \alpha_t^2$、$\beta(t)$：分散保持（variance-preserving）経路の二乗信号係数とそのレート（17.2 節）
- k、T、β_k：DDPM 連鎖のステップ、総ステップ数、およびステップ k で加えられる分散（17.2 節）
- $\lambda_t = \log(\alpha_t^2/\sigma_t^2)$：対数信号対雑音比（log signal-to-noise ratio、log-SNR）。$+\infty$（クリーンデータ、$t = 0$）から $-\infty$（純粋なノイズ、$t = 1$）まで単調に減少する
- p_t、$\nabla \log p_t$：$\mathbf{x}_t$ の時刻 t における周辺密度、およびそのスコア関数（score function）
- $\hat{\mathbf{x}}_0$、$\hat{\epsilon}$：ノイズ除去ネットワークが予測する事後期待値 $\mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]$ と $\mathbb{E}[\epsilon \mid \mathbf{x}_t]$
- $\mathbf{u}_t$、$\mathbf{u}_\theta$、$\hat{\mathbf{u}}$：経路に沿ってサンプルを輸送する確率フロー（probability flow）の速度場、それを近似するよう訓練されたパラメータ θ を持つニューラルネットワーク、およびそのネットワークによる予測速度
- $\bar{\mathbf{x}} = \mathbf{x}_t/\alpha_t$、$\bar{\sigma} = \sigma_t/\alpha_t$：すべてのガウス経路が $\bar{\mathbf{x}} = \mathbf{x}_0 + \bar{\sigma}\epsilon$ の形になり、サンプラがステップを進める座標系（17.3 節）。ここでの上線は α_t による除算を意味し、$\bar{\alpha}_t = \alpha_t^2$ における二乗とは異なる
- $\mathbf{v} = \alpha_t \epsilon - \sigma_t \mathbf{x}_0$：$\mathbf{v}$ 予測（17.2 節）における目標ベクトル
- $\tilde{\mathbf{u}}$：分類器なしガイダンス（classifier-free guidance）を適用した速度場（17.4.4.2 節）
- $\bar{\mathbf{u}}(\mathbf{x}_t, s, t)$、$\mathbf{u}_{\text{ref}}$：時刻 t と s の間の平均速度（average velocity）、および少数ステップ蒸留の損失（17.8 節）で用いられる参照速度（教師モデルの予測 $\hat{\mathbf{u}}$ または条件付き速度 $\epsilon - \mathbf{x}_0$）
- c、γ：条件付け信号およびガイダンス強度（17.4.4.2 節）

イントロダクション

つい最近まで、日常的に触れるほぼすべてのコンピュータプログラムは、どのように振る舞うべきかを定めた厳密なルールの集合として記述されていた。たとえば、オンラインストアを運営するアプリケーションを作りたいとしよう。ホワイトボードを囲んで数時間議論すれば、実用的な設計の大枠が固まるだろう。ユーザーはブラウザやモバイルアプリを通じてストアを操作し、データベースが在庫・顧客のカート・過去の注文を追跡し、システムの中核では**ビジネスロジック**が想定されるあらゆる事象を対応する処理へと結びつけるルールを規定する。顧客が商品をカートに入れるためにクリックすると、プログラムはそのユーザーの ID と商品の ID を紐づける行を挿入する。境界ケース（空のカートで決済しようとしたらどうなるかなど）を一つひとつ確認し、テストでいくつかのバグを修正すれば、実際の顧客を 1 人も迎える**前に**システムを稼働させることができる。常に 100% 機能する明示的なルールを考案できるなら、機械学習の出番はない。

機械学習コミュニティにとって幸いなことに、自動化したいたくさんのタスクは、人間の創意工夫だけではそう簡単には解けない。知り合いの最も優秀なプログラマと一緒にあのホワイトボードの前に戻り、次のいずれかの問題に取り組むところを想像してみよう。

- 地理情報、衛星画像、および直近の一定期間にわたる過去の観測データが与えられたとき、明日の天気を予測するプログラムを書く。
- 自由形式のテキストで書かれた質問を受け取り、計算が必要な場合にはコードを書いて実行しながら、正確に回答するプログラムを書く。
- 街路の風景に写っているすべての人物と車両を識別し、それぞれの輪郭をなぞるプログラムを書く。
- 自力では決して出会わないような何百万もの候補の中から、ユーザーが気に入りそうな本や音楽を推薦するプログラムを書く。
- アミノ酸配列だけから、タンパク質が折りたたまれる 3 次元構造を予測するプログラムを書く。

これらのどのタスクについても、一流のプログラマでさえ明示的なルールを一から書き下すのには苦労するだろう。背後にあるパターンが時間とともに変化するため、実用的なシステムには継続的な適応が求められることもある。さらに多くの場合、生のピクセルと物体の境界の関係であれ、アミノ酸配列と折りたたまれた立体構造の関係であれ、入力と出力の関係は何百万もの微妙な相互作用に依存している。人間の目や耳がそのタスクを難なくこなせる場合であっても、そのルールを意識的に言語化することはできない。

機械学習（machine learning）とは、典型的には観測データや環境との相互作用という形の経験を通じて、タスクの遂行能力を向上させるアルゴリズムの研究分野である。ルールベースのオンラインストアは、開発者がコードを書き換えない限り 100 万人目の顧客に対しても最初の顧客とまったく同じように振る舞うが、機械学習システムはデータから自身の振る舞いを洗練させていく。本書では、**深層学習**（deep learning）の基礎と実践を学ぶ。深層学習とは、言語、視覚、音声、科学、意思決定にわたる現代の進歩を牽引する、多層構造の学習手法の総称である。

動機づけとなる例

車に乗り込み、「**Hey Siri**」と声をかけて、最寄りのコーヒーショップへの道順を尋ねたとしよう。1〜2 秒のうちにスマートフォンが起動し、要求を文字に起こし、経路を選び、到着時刻を推定する。この短いやり取りだけでも、すでに複数の機械学習モデルが関わっている。

最初のステップ、すなわち「**Alexa**」、「**OK Google**」、「**Hey Siri**」といった**ウェイクワード**への応答だけを考えてみよう（Figure 2）。静かな部屋でコンピュータとコードエディタだけを前に座り、第一原理からウェイクワード認識器を書くところを想像してほしい。スマートフォンのマイクは毎秒約 16,000 個（音楽 CD では 44,100 個）の数値を記録しており、その一つひとつがある瞬間における空気圧の振幅を測っている。声質、訛り、部屋の音響、背後の交通騒音がさまざまに異なる状況で、1 秒間の 16,000 個の生の数値配列を確実に $\{\text{yes}, \text{no}\}$ へと写像する明示的なルールとはどのようなものだろうか。見当がつかなくても心配はいらない。そのルールを手作業で書く方法など、誰にもわからないからである。

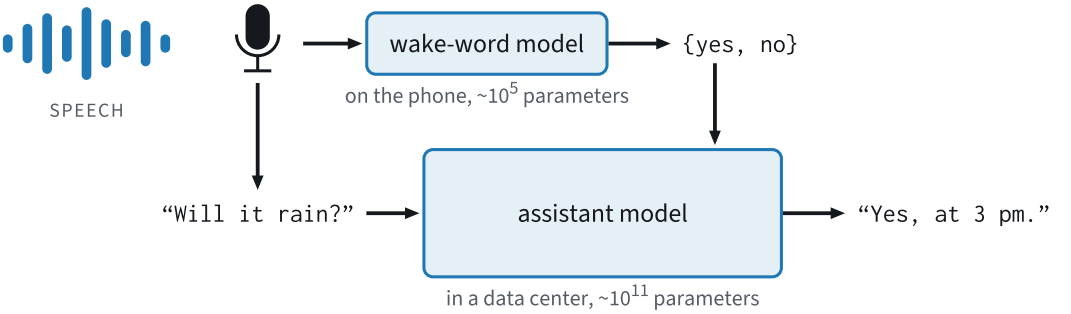


Figure 2: 音声による要求の背後にある 2 つのモデル。スマートフォン上の小さなウェイクワードモデルが常時待ち受け、yes か no かを判定する。これが反応すると、データセンタにある大規模なアシスタントモデルが要求そのものを処理する。どちらも同じ手順で訓練される。

ここに発想の転換がある。音波を単語へ写像するルールを書き下すことはできなくても、人間自身の手で例にラベルを付けることなら容易にできる。したがって、数千個の短い音声断片と、それぞれにウェイクワードが含まれるかどうかを示すラベルを対にした**データセット**を収集できる。認識器をハードコーディングする代わりに、調整可能な数値の**パラメータ**（回して調節できるつまみのようなもの）によって入出力の振る舞いが決まる柔軟なプログラムを書くのである。パラメータを固定したプログラムを**モデル**と呼び、つまみを回すことで到達できるすべての写像の集合を**モデル族**（model family）、データを使ってつまみを調整するメタプログラムを**学習アルゴリズム**（learning algorithm）と呼ぶ。

適切に選べば、同じモデル族でも、あるつまみの設定では「**Alexa**」を認識し、別の設定では「**Apricot**」を認識できるようになる。もちろん、つまみの設定がランダムなままではどちらも認識できない。**訓練**（training）とは、データによってつまみを有用な値へと導く反復ループのことである（Figure 3）。

1. モデルのパラメータをランダムに初期化し、何も知らない状態から始める。
2. データセットから例（音声断片と $\{\text{yes}, \text{no}\}$ ラベルの対）をサンプリングする。
3. それらの例に対するモデルの予測がラベルにより良く一致するよう、パラメータをわずかに調整する。
4. 性能が改善しなくなるまで、ステップ 2 と 3 を繰り返す。

このパラダイムは**データによるプログラミング**と捉えることができる。驚くべきことに、Figure 2 を振り返ると、音声による要求の両端を動かしているのはまさにこの同じ手順である。一方はス

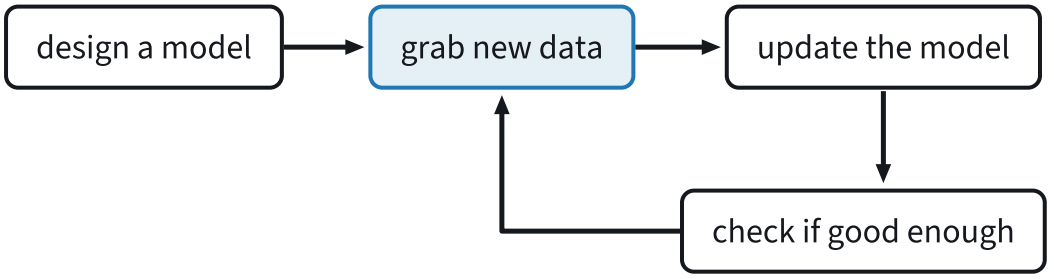


Figure 3: 典型的な訓練プロセス。

マートフォン上で数ミリワットの電力で常時動作する小さなウェイクワード検出器（パラメータ数は約 10 万個）であり、もう一方はデータセンタで稼働し、自由形式の質問への回答や言語の翻訳、ソフトウェアのデバッグまでこなす対話型アシスタント（パラメータ数は数千億個）である。

主要な構成要素

ウェイクワード検出のように、入力から既知のラベルを予測する枠組みは**教師あり学習**（supervised learning）と呼ばれる。教師あり学習をはじめ、本書で扱うあらゆるパラダイムに共通して、次の 4 つの中核的な要素が繰り返し現れる。

1. 学習の元となる**データ**。
2. 入力を出力へ変換する方法を定める**モデル**。
3. モデルの性能を定量化する**目的関数**（objective function）。
4. 目的関数を改善するようにモデルのパラメータを調整する**最適化アルゴリズム**（optimization algorithm）。

データ

データセットは**例**（example。**データ点**、**インスタンス**、**サンプル**とも呼ばれる）の集まりである。各例は**特徴量**（feature。**入力**または**共変量**とも呼ばれる）という数値属性を持ち、教師あり学習の問題ではさらに**ラベル**と呼ばれる目的値を持つ。

すべての例が同じ個数の特徴量を持ち、固定長のベクトルをなす場合がある。このベクトルの長さが入力の**次元数**（dimensionality）である。たとえば、200×200 のカラー写真は赤・緑・青のチャネルにわたって $200 \times 200 \times 3 = 120{,}000$ 個の輝度値を保持し、病院の入院記録であれば 30 日生存率を予測するためにバイタルサイン、検査値、診断名をまとめた固定長ベクトルになるだろう。固定長ベクトルは扱いやすいが、現実世界のデータの多くは定型の枠にきれいには収まらない。ウェブ上の写真は解像度もアスペクト比もさまざまであり、顧客レビューはわずか 2 語（「**it stinks!**」）のものから数ページに及ぶものまである。実際には、テキストを離散的な**トークン**（単語やサブワード断片）に分割し、各文書をトークン ID の可変長な系列として表現する。現代の深層学習アーキテクチャが際立っているのは、可変長の画像・音声・テキストをそのまま自然に扱える点にある。

データにおいては量と質の両方が重要である。表現力の高い深層ネットワークは規模が大きいほど真価を発揮する。現代の言語モデルは 10 兆（10^{13}）トークン規模のコーパスで訓練されており、そこでは窮屈な単純化の仮定に頼らずとも豊かな構造を学習できる。しかし、どれほど量を増やしても、破損したデータや偏ったデータを救うことはできない（**ゴミを入れればゴミしか出てこない**［garbage in, garbage out］）。明るい肌のデータだけで訓練された皮膚病変の分類器は

暗い肌に対して危険な誤判定を起こすし、偏見を含んだ過去の採用結果で訓練された履歴書選考システムは、エンジニアが意図しなくても過去の偏見を自動的に再現してしまう。データセットが何をカバーし、誰を取りこぼしているかを問うことは、後から付け足す配慮ではなくシステム設計そのものの一部である。

モデル

モデルとは、ある型の入力を別の型の予測へと変換する、パラメータ化された計算上の写像である。深層学習モデルは、パラメータを持つ線形変換と非線形変換の層（layer）を下から上へと幾重にも連ねており、これが**深層**学習という名前の由来となっている。その規模は、1 本の直線を定める 2 つの数値から、最先端の言語モデルにおける数千億個の**重み**（weight）にまで及ぶ。

目的関数

「経験を通じて改善する」ことを数学的に語るには、**目的関数**と呼ばれるスカラーの誤差指標が必要である（慣例として小さい値ほど良いため、**損失関数**［loss function］とも呼ばれる）。連続的な数値の目的値に対して最もよく使われる損失は、予測と正解の差を二乗した**二乗誤差**（squared error）である。一方、分類において本来関心があるのは誤り率（誤分類された例の割合）であることが多いが、誤り率はパラメータを動かしたときに不連続に変化するため、たどるべき滑らかな勾配が得られない。そこで代わりに、滑らかな**代理**損失（surrogate loss）を最小化する。その代表格が**交差エントロピー**（cross-entropy）であり、モデルが正解クラスに低い確率を割り当てるたびに大きなペナルティを課す（第 37 章）。

重要な点として、つまみの調整に使った例で損失を 0 に近づけたからといって、新しい入力に対する正解率が保証されるわけではない。本当に重要なのは、未知の例に対する**汎化**（generalization）である。去年の模擬試験を丸暗記した学生でも初見の問題が出題される期末試験で惨敗しうるのと同じように、表現力の高いモデルは訓練例固有の癖まで丸暗記してしまうことがあり、この失敗を**過学習**（overfitting）と呼ぶ。そのため、データをパラメータの当てはめに使う**訓練セット**（training set）と、学習したモデルがどれほど汎化するかを測るためだけに手元に残しておく**テストセット**（さらに実務ではモデル選択［model selection］のための**検証セット**［validation set］）に分割する。

最適化アルゴリズム

データ、モデル族、損失関数が揃ったら、良いパラメータ値を探索するアルゴリズムが必要になる。本書に登場するほぼすべてのモデルは、**勾配降下法**（gradient descent）の派生手法で訓練される。各ステップにおいて、各パラメータをほんの少し動かしたときに損失がどう変化するかを計算し、損失が下る方向へ一歩進むのである。ただし、この計算を 1 ステップごとに何百万、何兆という訓練例すべてにわたって平均していては時間がかかりすぎるため、更新のたびにランダムにサンプリングした少数の例からなる**ミニバッチ**上で勾配を推定する。このアルゴリズムが**確率的勾配降下法**（stochastic gradient descent、SGD）であり、これから出会うすべての深層ネットワークはその現代的な適応型派生手法（第 9 章）によって訓練されている。訓練が終わり重みが固定されたあと、新しい入力に対してモデルを順方向に実行することを**推論**（inference）と呼ぶ。

機械学習問題の種類

以上の 4 つの構成要素を踏まえ、本書全体を通して登場する主な問題の定式化を概観していこう。

教師あり学習

教師あり学習では、特徴量とラベルの対からなるデータセットを用いて学習アルゴリズムを動かし、新しい入力を予測ラベルへと写像するモデルを出力させる（Figure 4）。確率的に言えば、特徴量が与えられた下でのラベルの条件付き分布（conditional distribution）を推定することに相当する。CT スキャンからの腫瘍診断から 1 時間後の電力需要予測に至るまで、驚くほど幅広い実務上の意思決定がこの枠組みに当てはまる。

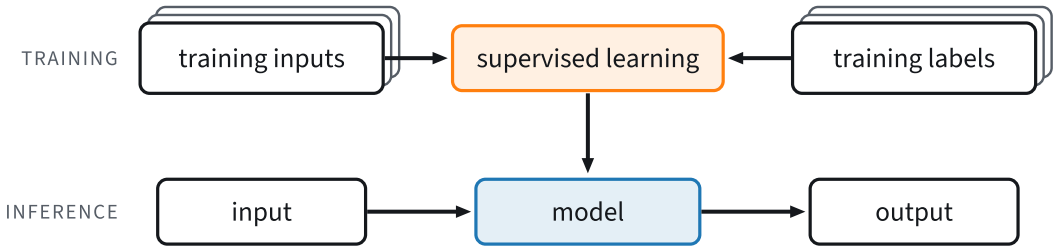


Figure 4: 教師あり学習。

回帰

目的値が連続的な数値であり、**どのくらいか**や**いくつか**という問いに答える場合、そのタスクは**回帰**（regression）である。たとえば、住宅の売却価格を `[square footage, bedrooms, bathrooms, walk-minutes to downtown]`（床面積、寝室数、浴室数、中心街までの徒歩分数）から推定する問題を考えよう。マンハッタンなら `[600, 1, 1, 60]`、ピッツバーグなら `[3000, 4, 3, 10]` といった具合である。入院期間、手術時間、6 時間降水量の予測も同じ型に従う。測定されていない要因によって常にばらつきが生じるため、予測が目的値に寸分違わず一致することはめったにない。のちに見るように、二乗誤差の最小化は、そうした残差（residual）をガウスノイズとしてモデル化することに対応している。

分類

目的値が離散的なカテゴリの集合から選ばれ、**どれか**という問いに答える場合、そのタスクは**分類**（classification）である。クラスが 2 つ（`{cat, dog}` や `{benign, malignant}`）であれば二**値分類**（binary classification）、それより多ければ（数字の `{0, ..., 9}` など）**多クラス分類**（multiclass classification）となる。分類器は融通の利かない確定ラベルをそのまま出すのではなく、クラス上の確率分布を出力することで較正された確信度（猫 90%、犬 10% など）を表し、交差エントロピーによって滑らかに訓練できるようになっている。

ただし、確率はまだ意思決定そのものではない。誤りの種類によって、伴うコストが大きく異なるからである。庭でキノコを見つけ（Figure 5）、写真分類器にかけたところ、猛毒のタマゴテングタケである確率が 0.2（安全である可能性が 80%）と推定されたとしよう。

最も確率の高いクラスを選ぶルールに従えば「**食べる**」ことになるが、それに従うのは愚かである。おいしい夕食のために 20% の確率で肝不全になる危険は冒せない。食べた場合の期待損失（$0.2 \times \infty + 0.8 \times 0 = \infty$）に比べれば、食べられるキノコを捨ててしまう惜しさ（リグレット 0.8）など微々たるものである。合理的な意思決定では、非対称な行動のコストで予測確率を重み付けする。実際、Figure 5 は本物のタマゴテングタケであり、この慎重さが命を救うことになる。

多クラス分類は、巨大なラベル集合にもスケールする。現代の**言語モデル**（language model）の本質は、約 $100{,}000$ トークンの語彙（vocabulary）を対象とする多クラス分類器であり、**これまでのテキストが与えられたとき、次に来るのはどのトークンか**という 1 つの問いに繰り返し答え

Figure 5: タマゴテングタケ（絶対に食べてはいけない）

ている（Figure 6）。その分布からトークンを 1 つサンプリングしてプロンプトの末尾に追加し、同じ操作を繰り返すことで、筋の通った文章や数式、コードを 1 トークンずつ生成していく。

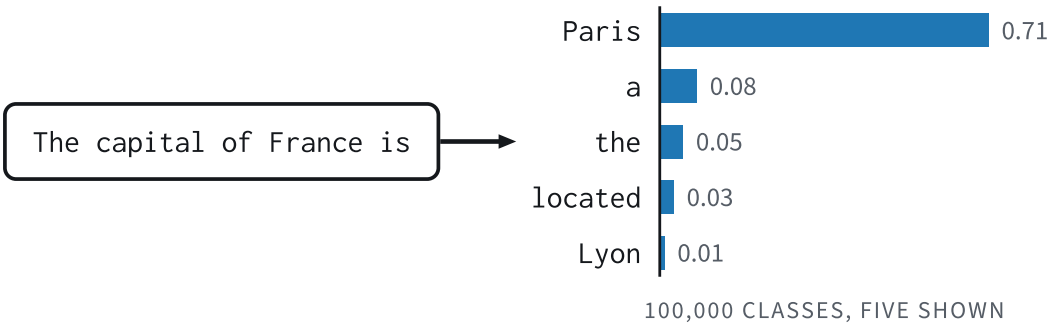


Figure 6: 分類としての次トークン予測。これまでのテキストが与えられると、モデルは語彙内のすべてのトークンに確率を割り当てる。約 100,000 クラスのうち、最も確率の高い上位 5 つを示している。

検出、セグメンテーション、タグ付け

単一ラベル分類は、1 つのカテゴリで入力全体を表せると仮定している。たとえ完璧に近い猫対犬の分類器であっても、ロバの上に犬が乗り、その上に猫が乗り、さらにその上に雄鶏が止まっている『**ブレーメンの音楽隊**』の挿絵（Figure 7）を渡されれば対応に窮してしまう。

複雑なシーンに対して、実際にはどのような出力が欲しいのだろうか。位置を問わず**どの**カテゴリが含まれているかだけを知りたいなら、**マルチラベル分類**（multi-label classification。**タグ付け**とも呼ばれる）によって各候補ラベル（`donkey`、`dog`、`cat`、`rooster`）に対する独立な二値確率を予測すればよい。しかしコンピュータビジョン（第 32 章（英語版））では、空間的な構造も必要になるのが普通である。すなわち、**物体検出**（object detection）は個々の動物ごとにバウンディングボックスとカテゴリを予測し、**インスタンスセグメンテーション**は動物同士が重なり合う箇所でもピクセル単位で正確なマスクを描き出し、マルチモーダルな視覚言語モデルはその配置をそのまま自然言語で記述する。

Figure 7: ロバ、犬、猫、雄鶏。

検索、ランキング、推薦

検索や情報検索（retrieval）における目標は、クエリに対して候補にスコアを付け、最も関連性の高い項目を上位に返すことである。ウェブページのランキングであれ、言語モデルが根拠に基づいて回答できるよう文書集合から参照パッセージを検索して取り出す処理であれ同じである（第 19 章（英語版））。ここに**パーソナライゼーション**を加えたものが**推薦システム**（recommender system）であり、明示的フィードバック（explicit feedback。星の数による評価など）と暗黙的なシグナル（クリック、視聴時間、曲のスキップなど）の両方を使って、何百万もの映画・楽曲・商品を一人ひとりの好みに結びつける。

単純な予測型の推薦システムには、見落としやすい落とし穴が潜んでいる。まず、人々は平凡な映画よりも大好きだった映画や大嫌いだった映画をはるかに頻繁に評価するため、フィードバックは**打ち切り**（censored）のある偏ったものになる。さらに厄介なことに、稼働中の推薦はユーザーが次に目にする候補そのものを変えてしまう。その結果、人気のある項目は表示されたというだけでクリックを集め、再訓練されたモデルがそれをいっそう強く推薦するという**フィードバックループ**が生じる。

系列学習

回帰や画像分類は、連続して届く入力を互いに独立なものとして扱うため、1 つの例にスコアを付け終えるとモデルはそれを忘れてしまう。しかし、音声、テキスト、動画、集中治療室（ICU）のバイタルサインのストリームなどは**可変長の系列**（variable-length sequence）であり、その意味は時間的な順序や長距離の文脈の中に宿っている。**系列対系列**（sequence-to-sequence）の問題では、入力系列を長さやアライメントが大きく異なりうる出力系列へと写像する。

- **自動音声認識と音声合成**。発話されたフレーズ（Figure 8）は、わずか数文字のテキストに対して毎秒数万個の音声サンプルに及び、サンプルと文字の間に 1 対 1 のアライメントは存在しない。テキスト音声合成はこの写像を逆向きに実行する。

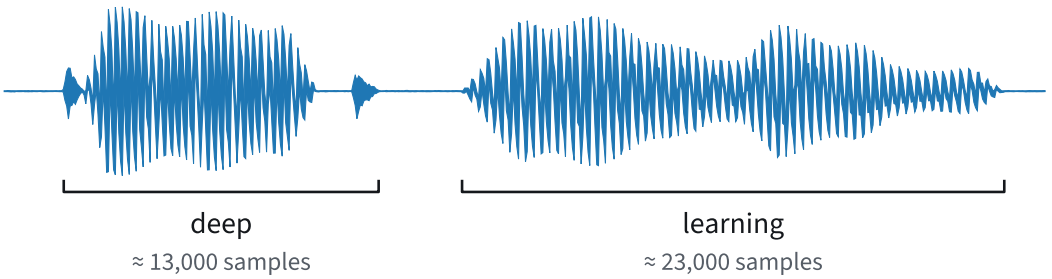


Figure 8: 音楽 CD と同じく毎秒 44,100 回サンプリングされた音声録音とその書き起こし。発話された各単語は数千個の振幅サンプルにまたがるため、入力と出力の間に 1 対 1 の対応はない。

- **機械翻訳と言語タスク**。言語間の翻訳では、節の語順が非単調に入れ替わる。たとえばドイツ語では、よく知られているように動詞が節の末尾に置かれる。

```
German:           Haben Sie sich schon dieses grossartige Lehrwerk angeschaut?
English:          Have you already looked at this excellent textbook?
Word-for-word:    Have you yourself already this excellent textbook looked at?
```

10 年前であれば音声の書き起こし、翻訳、要約、対話にはそれぞれ専用のパイプラインが必要だったが、現代の系列アーキテクチャ（第 8 章から第 12 章まで）では、事前学習された単一のモデルでそれらすべてを扱える。

教師なし学習と自己教師あり学習

人手で付与されたラベルは高コストなうえタスク固有であるのに対し、生のテキスト、画像、音声、科学計測データは豊富に存在する。**教師なし学習**（unsupervised learning）は、ラベルなしデータだけからどのような構造を抽出できるかを問う枠組みである。

- **クラスタリングと部分空間推定：**あらかじめカテゴリを定義せずに写真や顧客行動をまとまりのあるクラスタへとグループ化したり、高次元の観測値が持つ本質的な自由度を捉える低次元座標（**主成分分析**［principal component analysis］や非線形多様体）を見つけたりする。
- **表現学習：**離散的な記号やマルチモーダルな対象をユークリッド空間へ埋め込み、意味的な関係を幾何学的な関係へと変換する。古典的な例として、`"Rome"` − `"Italy"` ＋ `"France"` ≈ `"Paris"` のようなベクトルの類推がよく知られている。
- **生成モデリング：**高次元データをラベルへ写像する（写真を入力し、`"cat"` を出力する）**識別**モデル（discriminative model）に対し、**生成**モデル（generative model）はデータ分布を十分に学習して写実的な新しいサンプルを合成する（Figure 9）。**変分オートエンコーダ**（variational autoencoder）(Kingma and Welling 2014; Rezende et al. 2014)、**敵対的生成ネットワーク**（generative adversarial network）(Goodfellow et al. 2014)、**正規化フロー**（normalizing flow）(Dinh et al. 2014; Dinh, Sohl-Dickstein, et al. 2017) による初期の飛躍に続き、現在では**拡散**モデル（diffusion model）やフローマッチングモデル (Sohl-Dickstein et al. 2015; Song and Ermon 2019; Ho et al. 2020; Y. Song, Sohl-Dickstein, et al. 2021) が、ランダムなノイズを段階的に構造化データへと変換する方法を学習することで、高解像度の画像、音声、動画、さらには分子構造までも生成している。

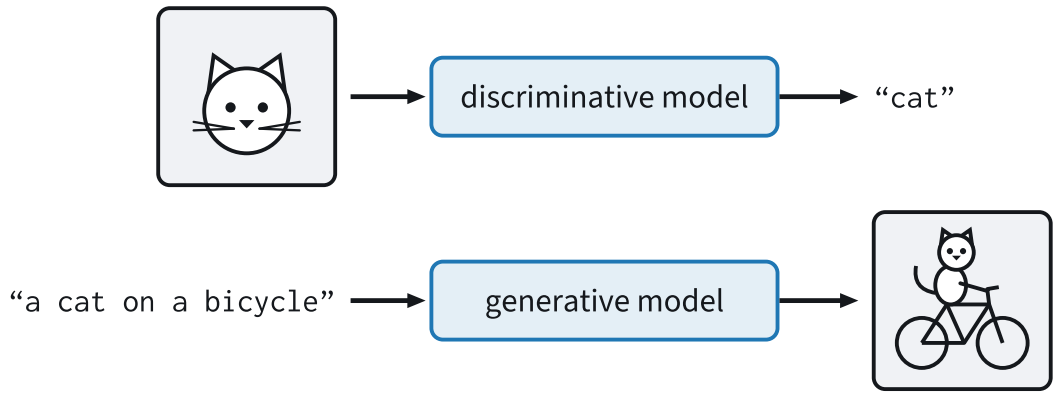


Figure 9: 識別モデルはデータをラベルへ写像し、生成モデルは記述をデータへ写像する。

とりわけ大きな影響をもたらしたのが**自己教師あり学習**（self-supervised learning）であり、ラベルなしデータから教師信号を直接作り出す。たとえば、ウェブテキストにおける次のトークンの予測、マスクされた単語の穴埋め (Devlin et al. 2018)、マスクされた画像パッチの再構成 (He et al. 2022) などである。数兆語にわたって次に来るすべてのトークンを予測させることで、人手のラベルを１つも使わずに、構文、世界知識、コードの構造、論理的推論のパターンを単一のネットワークの内部に獲得させることができる。このような汎用的な**ベースモデル**を一度訓練することを**事前学習**（pretraining）と呼ぶ。その後、教師ありの**ファインチューニング**、人間の選好へのアライメント、ツールの利用を通じて適応させることで、事前学習済みの１つのバックボーンが分類器、アシスタント、あるいは自律的なコーディングエージェントへと姿を変える（Figure 10）。

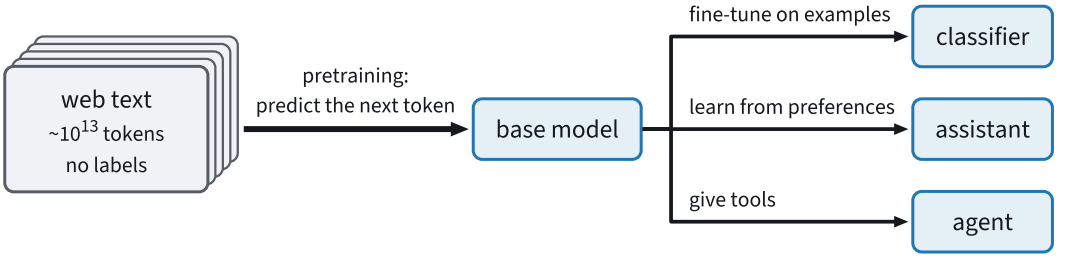


Figure 10: 一度事前学習し、何度も適応させる。単一のモデルを、ラベルを一切使わずに次のトークンを予測させることでウェブテキスト上で事前学習する。ラベル付き例でのファインチューニング、人間の選好からの学習、ツールへのアクセスを加えることで、得られたベースモデルは分類器、アシスタント、あるいはエージェントになる。

環境との相互作用

教師あり学習や自己教師ありの事前学習は**オフライン**で行われる。すなわち、現実世界から一度だけデータを収集し、閉じた部屋の中で学習を進める（Figure 11）。

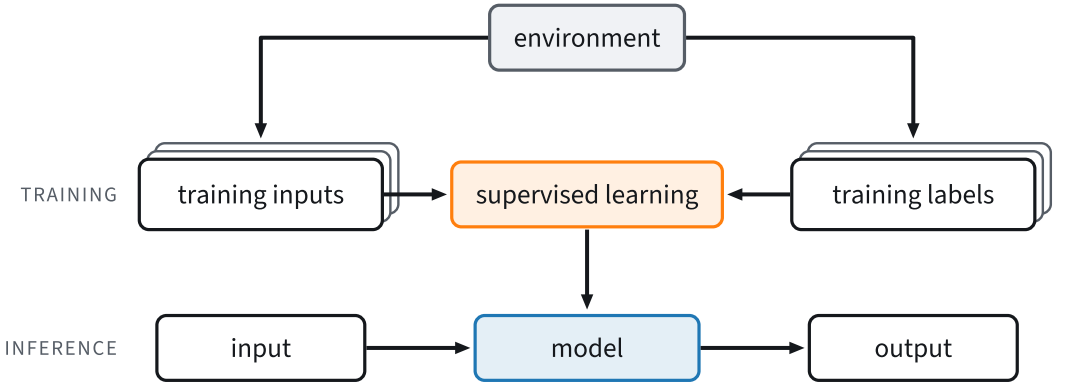


Figure 11: 教師あり学習のための環境からのデータ収集。

しかし、ひとたびモデルが実世界に配備されると、その整然とした分離は崩れ去る。環境は時間とともに自然に変化するうえ、攻撃者やユーザーがモデルそのものに反応するからである。たとえばスパム送信者は、昨日のスパムフィルタをすり抜けるよう巧妙に細工したメッセージを作成するため、訓練データと実運用データの間に**分布シフト**（distribution shift）が生じる。

強化学習とエージェント

システムが選択する**行動**（action）の結果によって世界の将来の状態が変化する場合、問題は**強化学習**（reinforcement learning）の領域に入る（Figure 12）。各ステップにおいて、**エージェント**は**観測**（observation）を受け取り、自身の**方策**（policy）に従って**行動**を選択し、環境が新しい状態へと遷移するなかでスカラーの**報酬**（reward）を受け取る。

深層強化学習が最初に世間の注目を集めたのは、ピクセル入力だけから Atari のゲームを攻略し (Mnih et al. 2015)、囲碁で人間の世界チャンピオンを破ったときであった (Silver et al. 2016)。今日では、言語モデルの事後学習（post-training）の中核にも据えられている。そこでは応答の候補に対し、学習済みの人間の選好モデル (Christiano et al. 2017; Ouyang et al. 2022)、あ

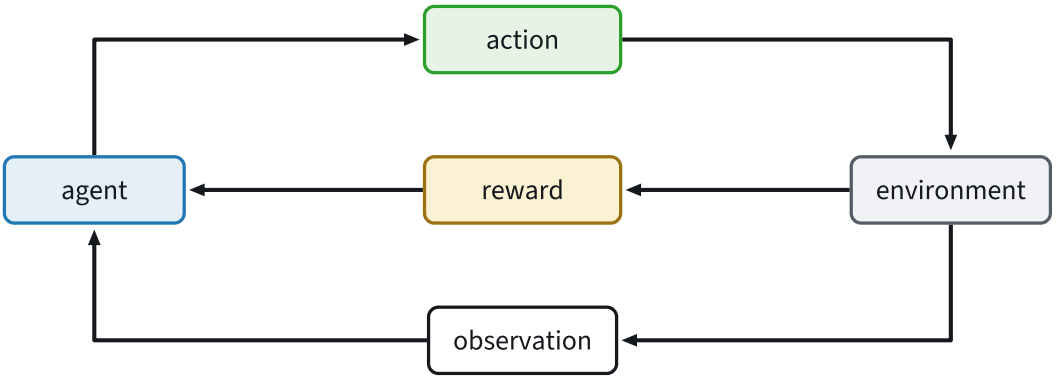


Figure 12: 強化学習と環境の相互作用。

るいは単体テストの実行や数学的証明の検証を行う自動検証器によってスコアが付けられる (DeepSeek-AI 2025)。複数ターンの観測・行動ループの中で外部**ツール**（シェルコマンドの実行、ファイルの読み取り、検索エンジンへのクエリ送信など）を備えることで、言語モデルは自律的な**エージェント**となる（第 27 章（英語版））。

強化学習は、標準的な教師あり学習にはない 3 つの課題に直面する。第 1 に**信頼割り当て**（credit assignment。チェスの対局が 60 手の末に勝敗を決したとき、それ以前のどの手が功績または失策に値したのかという問題）、第 2 に**部分観測性**（partial observability。観測からは世界の真の状態の一部が隠れている場合がある）、第 3 に**活用**（exploitation。既知の最良の戦略から報酬を得ること）と**探索**（exploration。より良い結果をもたらすかもしれない未知の行動を試すこと）の間のトレードオフである。完全な状態が観測できる場合（**マルコフ決定過程**［Markov decision process］）、行動が将来の状態を変えない場合（**文脈付きバンディット**［contextual bandit］）、そもそも状態が存在しない場合（**多腕バンディット問題**［multi-armed bandit］）は、それぞれ有用な特殊ケースとなる。

深層学習への道のり

現代のアーキテクチャが登場したのはごく最近だが、ノイズを含むデータから学ぼうとする数学的動機は科学の歴史に深く根ざしている。Bernoulli の確率論、Legendre と Gauss による最小二乗法（method of least squares）(Legendre 1805)、Shannon の情報理論、Turing による 1950 年の機械知能の定式化 (Turing 1950)、そしてシナプス強化に関する Hebb の仮説 (Hebb 1949) などがその系譜である。生体の神経回路から大まかな着想を得た初期の**ニューラルネットワーク** (McCulloch and Pitts 1943) は、今日まで受け継がれる 2 つの柱を打ち立てた。積み重ねた**層**にわたって線形変換と要素ごとの非線形変換を交互に適用すること、そして連鎖律（chain rule）によって積層全体を端から端まで微分する**誤差逆伝播法**（backpropagation）である (Rumelhart et al. 1986)。

しかし 1995 年から 2005 年頃にかけて、ニューラルネットワークはカーネル法、決定木、グラフィカルモデルに大きく水をあけられた。これらの手法の方が理論的保証が強く、小規模なデータセットでも安定して訓練できたからである（1936 年に発表された Fisher の 150 個のアヤメ［Iris］データが依然として論文に頻繁に登場し、MNIST の 60,000 枚の手書き数字が「大規模」と見なされていた時代である）。この評価を覆したのは、ウェブや安価なセンサから生み出されるデジタルデータの数十年にわたる爆発的増加と、もともとビデオゲーム向けに作られた GPU による並列計算であった（Table 1）。計算性能がメモリ帯域幅やメモリ容量よりもはるかに速く向上したため、ミニバッチ SGD で何度もデータを走査して訓練するパラメータ数の豊富なニューラルネッ

トワークはハードウェアの特性と見事にかみ合い、畳み込みネットワーク (LeCun, Léon Bottou, et al. 1998)、ゲート付き再帰構造 (Hochreiter and Schmidhuber 1997)、Q 学習 (Watkins and Dayan 1992) の中核的な着想を桁違いの規模で復活させた。

Table 1: データセット、それで訓練された最大のモデル、および 1 回の訓練に費やされた計算量の桁数による推移。データセットのサイズは例数（花、数字、画像）またはトークン数で表す。

年	データセット	最大の訓練済みモデル（パラメータ数）	1 回の訓練あたりの計算量（浮動小数点演算回数）
1970	Iris、10^2 個の花	10^1	10^5
1998	MNIST、10^5 枚の手書き数字	10^5	10^{13}
2012	ImageNet、10^6 枚のラベル付き画像	10^8	10^{18}
2020	ウェブテキスト、10^{11} トークン	10^{11}	10^{23}
2025	ウェブテキスト、10^{13} トークン	10^{12}	10^{26}

もっとも、単純な規模の拡大だけでは不十分であった。現代の深層学習を切り開いたのは、アルゴリズムとシステムの両面におけるいくつもの飛躍である。

- **エンドツーエンドの視覚階層**。2012 年、2 基のゲーム用 GPU で訓練された 8 層の畳み込みネットワークである AlexNet が、ImageNet の top-5 誤り率を約 26% から 15% へと引き下げた (Krizhevsky et al. 2012)（ほどなくして 3% 未満にまで低下した (Hu et al. 2018)）。**ドロップアウト** (Bishop 1995; Srivastava et al. 2014) などの正則化（regularization）手法にも支えられ、ピクセルから直接訓練された学習済みフィルタが、何十年もかけて人手で設計されてきた視覚処理パイプラインを凌駕したのである。
- **アテンションと Transformer**。系列モデルにおいて、**アテンション機構** (Bahdanau et al. 2015) は任意の長さの入力を単一の固定長ベクトルへ押し込める制約を取り払い、内容に基づく学習可能な参照機構として機能した。アテンションのみを軸に構築された **Transformer** (Vaswani et al. 2017) は、言語 (Devlin et al. 2018; Brown et al. 2020)、視覚 (Dosovitskiy et al. 2021; Liu et al. 2021)、音声 (Gulati et al. 2020)、意思決定 (L. Chen, Lu, Rajeswaran, Lee, Grover, Misha Laskin, et al. 2021) にわたるモデリングを統一した。
- **基盤モデル、アライメント、論理的推論**。ウェブ規模のコーパスにおける次トークン予測の事前学習 (Brown et al. 2020; Hoffmann et al. 2022) に続き、選好アライメント (Christiano et al. 2017; Ouyang et al. 2022) やコードおよび数学の検証器を用いた強化学習 (DeepSeek-AI 2025) を組み合わせることで、言語モデルは汎用アシスタントや多段階の推論を行うエージェントへと進化した。
- **深層生成による合成**。敵対的訓練（adversarial training）(Goodfellow et al. 2014) や**拡散モデル**による反復的なノイズ除去（denoising）(Sohl-Dickstein et al. 2015; Ho et al. 2020; Y. Song, Sohl-Dickstein, et al. 2021) によって、テキストの記述から写実的な画像・音声・動画を合成することが日常的な技術となった (Rombach et al. 2022; Ramesh et al. 2022; Saharia et al. 2022)。
- **分散システムとオープンなフレームワーク**。今日では分散訓練アルゴリズムが 1 回の実行で数千基のアクセラレータを協調動作させる一方、自動微分（automatic differentiation）とコンパイル機能を備えた NumPy 風の命令型フレームワーク（PyTorch、JAX、TensorFlow）の登場により、2014 年には Carnegie Mellon 大学の博士課程で 1 週間かかっていた課題が、読みやすいわずか 6 行ほどの Python コードへと凝縮された。

深層学習の本質

古典的なパイプラインと深層学習を決定的に分けているのは、**エンドツーエンドの表現学習**（end-to-end representation learning）である。2012 年以前、視覚や音声における応用機械学習といえば、Canny エッジ検出器 (Canny 1987) や SIFT 記述子 (Lowe 2004) のような領域固有の特徴抽出器を何年もかけて人手で設計し、その固定された出力を浅い分類器に渡すことを意味していた。深層学習はパイプライン全体を微分可能な層に置き換え、最終的な目的関数に向けて一括して調整する。領域固有の特徴量エンジニアリングが不要になったことで、分野間を隔てていた壁も取り払われた。現在では同じ中核アーキテクチャの派生形が、音声を翻訳し (Xiong et al. 2018)、10 日間の全球気象予測を数秒で計算し、登録されている 2 億種以上のタンパク質の 3 次元構造を予測している（この飛躍的成果に対して 2024 年のノーベル化学賞の半分が授与された）。

同様に注目すべきなのが、**予測可能なスケーリング**である（Figure 13）。モデルサイズ、データセットサイズ、訓練計算量（training compute）を連動して拡大していくと、交差エントロピー損失は両対数軸上で滑らかなべき乗則のフロンティアを描く (Kaplan et al. 2020; Hoffmann et al. 2022)。スケールアップは当て推量ではなく定量的な工学分野となり、Rich Sutton（2019）が**苦い教訓**（bitter lesson）と呼んだ知見を体現している。すなわち、計算量とともにスケールする汎用的な学習・探索手法は、人間が手書きしたヒューリスティクスを中心に据えたアプローチを一貫して追い抜くのである。もちろん、規模はひとりでに設計されるわけではない。メモリと計算資源を効率よく使うアーキテクチャや、信頼できる振る舞いを引き出す目的関数をいかに設計するかが、本書の「発展」パートおよび「言語モデル」パートの動機となっている。

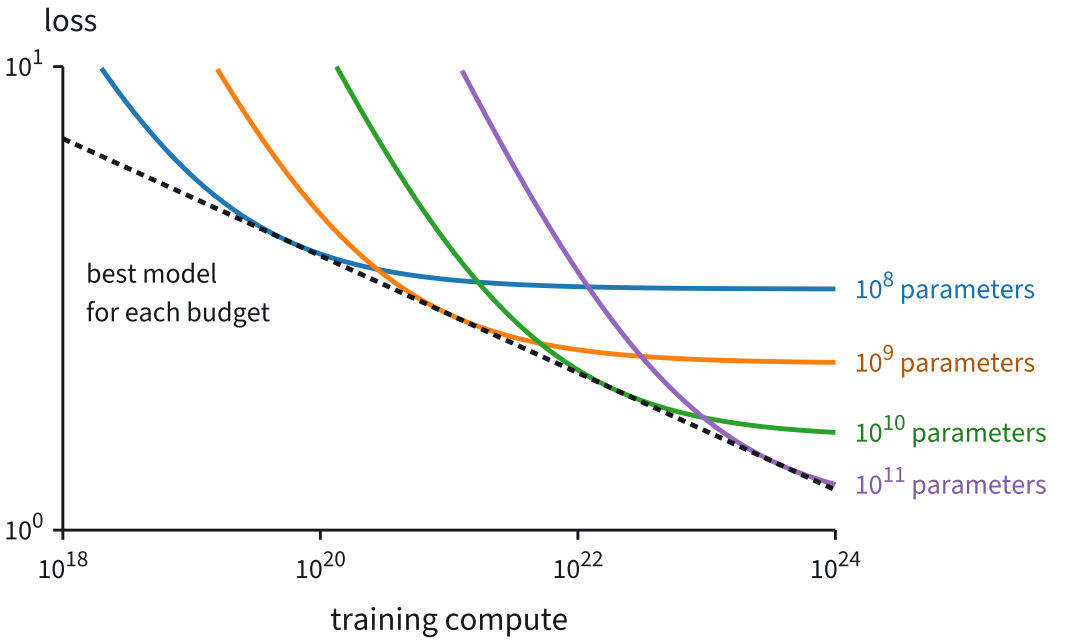


Figure 13: スケーリング則の模式図。サイズを大きくした一連のモデルについて、両対数軸上で訓練計算量に対する損失をプロットしたもの。各モデルは途中で頭打ちになるが、モデル族の包絡線（各計算予算における最良のモデル）は直線になる。この図は概念図であるため軸には 10 のべき乗のみを記しており、実測曲線は 11.6 節に示す。

それでもなお、実世界への配備には謙虚さが求められる。自動運転車がすでに複数の都市で商用運行されているとはいえ、あらゆる場所での完全自動運転がいまだに難しいのは、訓練ログに類

似した前例がない 100 万回に 1 度のまれな路上状況、すなわち**ロングテール**が存在するためである。より広く見ても、AI システムは魔法でもなければ自我の芽生えでもなく、人間が収集した有限のデータ上で人間が選んだ目的関数を最適化する統計モデルにすぎない。モデルが融資を審査し、臨床医を支援し、定型的な知的作業を自動化するとき、問うべき事柄は具体的である。すなわち、**誰が目的関数を選び、訓練データは何をカバーしており、経験の範囲外にある入力を渡されたときにシステムはどう振る舞うのか**という点である。

深層学習の進歩は当初から、アーキテクチャの論文公開、オープンソースのライブラリやモデルの重みの公開、そして学術界と産業界の垣根を越えて再現可能なコードを利用可能にするという、オープンな共有の文化によって加速されてきた。本書のすべての章で数学的基礎と実行可能なノートブックを直接対にしているのも、まさにその精神に基づいている。さあ、始めよう。

まとめ

機械学習は、手書きのルールではなくデータや相互作用を通じて振る舞いを改善するプログラムを構築する。どの定式化も、データ、パラメータ化されたモデル、目的関数（損失関数）、最適化アルゴリズム（ほぼ常に確率的勾配降下法のミニバッチ派生手法）という 4 つの要素を組み合わせ、汎化性能を検証するためにホールドアウトデータ上で評価を行う。教師あり学習（回帰、分類、構造化された空間認識、ランキング、系列変換）、教師なしおよび自己教師ありの事前学習、生成モデリング、強化学習にわたって、深層学習が生の入力から多層的な表現を**エンドツーエンド**で学習する点は際立っている。ウェブ規模のデータセット、GPU による並列計算、予測可能なスケーリング則、オープンソースの自動微分フレームワークを原動力として、統一された一連のアーキテクチャが今や言語、視覚、科学、自律エージェントにまたがる現代の AI を支えている。

演習問題

1. **ルールと学習**。現在書いているコードのうち、「学習」可能な部分、すなわちコード内で行っている設計上の選択を学習によって自動的に決定し改善できる部分はどれか。手作業で調整した閾値や優先度ルールなど、ヒューリスティックな設計上の選択が含まれているか。
    1. そのような判断を 1 つ取り上げて入力から出力への写像として述べ、そのヒューリスティクスが失敗する入力の例を挙げよ。
    2. 代わりに望ましい振る舞いを学習するためにどのようなデータとラベルを収集するか、訓練中にどの目的関数を最小化するか、そして学習したモデルをそのヒューリスティクスとどのように比較評価するかを答えよ。
2. **問題の定式化**。以下の各タスクを本章で扱った定式化（**回帰**、**分類**、**マルチラベル分類**、**物体検出**、**系列対系列学習**、**クラスタリング**、**強化学習**）のいずれかに分類し、モデルがラベル、報酬、あるいはそのどちらでもないものから学習するかを答えよ。
    1. 中古車の年式、走行距離、車種から再販価格を推定する。
    2. ドライブレコーダのフレームに写っているすべての道路標識の周囲にバウンディングボックスとカテゴリラベルを付与する。
    3. 受信した各サポートメールに対し、あらかじめ定めたリストの中からそのメールが関係する製品領域をすべて割り当てる。
    4. ニュース記事の 1 段落の要約を作成する。
    5. あらかじめグループが定義されていない状態で、多数の実験にわたる発現パターンによって遺伝子をグループ化する。
    6. 各調整のあとに得られる電力計測値と快適性に関する苦情のみから学習しながら、室温を快適に保ちつつエネルギー消費を抑えるように建物の暖房を 10 分ごとに調整する。

3. **決定閾値**。ある分類器が、キノコがタマゴテングタケである確率 p を出力するとする。タマゴテングタケを食べると $L > 0$ の損失が生じ、食用のキノコを捨てると 1 の損失が生じる。食用のキノコを食べた場合とタマゴテングタケを捨てた場合には損失は生じない。
   1. L の関数として、p がその値を超えるとキノコを捨てる方が食べるよりも期待損失が小さくなる閾値を導出せよ。
   2. $L = 1$ および $L = 1000$ に対してこの閾値を計算し、単に確率の高い方のクラスを選ぶルールとそれぞれ比較せよ。
4. **アルゴリズム、データ、計算量**。ここで Table 1 を用いる。n はデータセットのサイズ（例数またはトークン数）、p は最大モデルのパラメータ数、c は 1 回の訓練における浮動小数点演算回数で表した計算量である。
   1. 各行について、データ 1 件あたりのパラメータ数 p/n とパラメータ 1 個あたりの計算量 c/p を計算し、1970 年から 2025 年にかけてこれらの比率がどのように変化したかを述べよ。
   2. 各トークンを 1 回処理する密な Transformer は、1 トークンあたり約 $6p$ 回の演算（順伝播で $2p$ 回、逆伝播で $4p$ 回）、すなわちデータ全体を 1 パス処理するごとに約 $6pn$ 回の演算を費やす。この見積もりを 2020 年および 2025 年の行と照合し、2012 年の ImageNet の行（画像を約 90 パス走査して訓練された畳み込みネットワーク）に対してはこの見積もりが数桁もの過小評価になる理由を説明せよ。
   3. （厳密なカーネル法のように）訓練例のすべての対を比較する手法は、$n \times n$ 行列を保持し、n^2 程度の演算を必要とする。各行について、1 要素あたり 4 バイトで $n \times n$ 行列を保存するのに必要なメモリを計算し、その行列が 1 テラバイト（10^{12} バイト）に収まらなくなる最初の行を求め、なぜこうした傾向が深層ニューラルネットワークに有利に働いたのかを説明せよ。
5. **エンドツーエンドの訓練**。カメラの各ピクセルを車両・歩行者・背景のいずれかにラベル付けするよう訓練された認識ネットワーク（誤ってラベル付けされたピクセルの割合で評価される）と、それに続く手作業で調整された動作計画器および制御器からなる 3 段階の運転支援システムを考える。
   1. 誤ってラベル付けされたピクセル全体の割合はほとんど変化させないにもかかわらず、動作計画器に危険な軌跡を選択させてしまうような認識ネットワークのピクセルラベル付け誤りの例を挙げよ。
   2. もし動作計画器と制御器もニューラルネットワークであり、認識ネットワークと結合してエンドツーエンドで一括訓練するとしたら、どのようなデータと単一の目的関数が必要になるか。また、それでもなおエンジニアがモジュール化された段階構成を好むことがあるのはなぜか。
6. **次トークン予測と事前学習**。ウェブテキスト上で事前学習された、100,000 トークンの語彙を持つ言語モデルを考える。
   1. 本章の用語を用いて問題の定式化、入力、出力、損失関数を述べ、人手によるラベル付けが不要であった理由を説明せよ。
   2. 事前学習だけでこのモデルが遂行できるタスクを 1 つ挙げ、その後のファインチューニングや選好アライメントを必要とするタスクを 1 つ挙げよ。
7. **推薦システムにおけるフィードバックループ**。昨年のクリック履歴で推薦システムを訓練し、1 年間実運用したあと、その運用期間中に収集されたクリック履歴で再訓練するとする。
   1. 新しい訓練データがユーザーの選好の偏りのない標本とどのように異なるかを説明し、そのデータで過大評価される項目と過小評価される項目の例をそれぞれ 1 つ挙げよ。
   2. 実運用の 1 年間のうちにこのフィードバックループを検出し測定するためのオンライン実験を提案せよ。

基礎

イントロダクションでは、機械学習が人手で書かれたルールを、データ例から直接学習するパラメータ化されたプログラムに置き換える仕組みを見た。ここで、衣類を写した小さなグレースケール写真を数万枚手渡され、それぞれの画像に写っているのが T シャツ、スニーカー、コート、ハンドバッグのどれかを識別するシステムを構築するよう求められたとする。生のピクセル強度を衣類のカテゴリへ確実に対応づける明示的な `if-else` ルールをどう書けばよいか、わかるプログラマはいない。しかし、本パートの最初の 3 章を終える頃には、まさにそれを実現する実用的な分類器を実際に構築し、訓練し終えているはずである。さらに本パートを終える頃には、複雑な視覚シーンを認識する深い畳み込みネットワークや、英語の散文をゼロから生成することを学習するリカレントネットワークにまで進んでいるだろう。

「基礎」パート全体を通じたアプローチは、シンプルな原則に基づいている。深層学習を真に理解するには、ブラックボックスのライブラリ呼び出しに頼る前に、その中核となる仕組みを自らの手で構築すべきだということである。これら 8 つの章を通して、基礎となるほぼすべてのモデル、層、損失関数、オプティマイザを 2 回ずつ実装する。最初は**ゼロから**、基本的なテンソル演算と自動微分（automatic differentiation）のみに頼り、すべての数学的演算がコード上で見える形で実装する。次に、現代の深層学習フレームワーク（PyTorch、JAX、TensorFlow、MXNet）の高水準な抽象化を用いて**簡潔に**実装する。各コンポーネントが土台からどのように動作するかを確認した後は、以降の章ではその簡潔なフレームワーク実装を構成要素として用いる。

本パートの 8 つの章は、次の 3 つの段階で展開される。

- **ツールキットと基本レシピ（第 1 章〜第 3 章）：**まずは、完全に解析できるほどシンプルなモデルから始める。第 1 章では、多次元配列（**テンソル**）へのデータの格納と操作、表形式データの前処理、コードによる線形代数と微分積分の適用、**autograd** による導関数の自動計算、そして確率を用いた不確実性の扱いといった、深層学習に不可欠な実用的ツールを身につける。続く第 2 章と第 3 章では、線形モデルを題材に機械学習の普遍的な訓練ループを導入する。第 2 章では、線形回帰、二乗誤差損失、ミニバッチ確率的勾配降下法を用いて連続値のターゲットにモデルを適合させ、統計学の中心的な課題である**汎化**（generalization）に向き合う。すなわち、訓練データに過度に適合させると過学習を招く理由と、重み減衰などの正則化手法がいかにモデルを健全に保つかを考察する。第 3 章では連続値から離散的なカテゴリへと対象を移し、ソフトマックス回帰と交差エントロピー損失を構築して Fashion-MNIST ベンチマーク上の衣類写真を分類する。あわせて、実世界のデータ分布が訓練セットからずれたときに、デプロイされた分類器がどのように振る舞うかも検証する。
- **深層化とソフトウェアエンジンの習得（第 4 章〜第 5 章）：**線形モデルに引けるのは、クラス間の平坦な決定境界だけである。第 4 章では、非線形な活性化関数を備えたユニットの隠れ層を積み重ねて**多層パーセプトロン**（multilayer perceptron、MLP）を構築し、この限界を打破する。ここでは、誤差逆伝播法（backpropagation）によって深いネットワーク内を信号と勾配が順方向・逆方向に流れる仕組みをたどり、不用意に初期化すると深いネットワークが勾配消失や勾配爆発に陥る理由を探る。さらに、容量の大きいネットワークに

対する強力な正則化手法として**ドロップアウト**を導入する。続く第 5 章では、現代の深層学習フレームワークのソフトウェアアーキテクチャを解き明かす。再利用可能なモジュール式のブロックから複雑なネットワークを組み立てる方法、パラメータの共有・検査・初期化、カスタム層と順伝播の定義、訓練済みの重みの保存と読み込み、そして GPU 上への計算の移行を扱う。

- **ドメイン構造のアーキテクチャへの組み込み（第 6 章〜第 8 章）：**全結合ネットワークはすべての入力特徴量を対称に扱うため、入力が 2 次元の画像グリッド上に並んだピクセルなのか、文の中に順序よく並んだ単語なのかを区別しない。最後の 3 章では、データが本来持つ対称性をネットワークのアーキテクチャに直接組み込むことで、統計的効率と計算効率が劇的に向上することを示す。画像に対しては、第 6 章で平行移動不変性と空間的局所性の原理から**畳み込みニューラルネットワーク**（convolutional neural network、CNN）を展開し、パディング、ストライド、複数の入力・出力チャネル、プーリングを導入して古典的な LeNet アーキテクチャを構築する。続く第 7 章では、コンピュータビジョンを現代へと押し上げたアーキテクチャ上のマイルストーンとして、AlexNet、繰り返し可能なブロック設計（VGG）、1×1 畳み込み（NiN）、複数分岐の並列ブロック（GoogLeNet）、**バッチ正規化**（batch normalization）、ResNet の**残差接続**（residual connection）、DenseNet をたどり、原理に基づいたアーキテクチャ設計空間（AnyNet と RegNet）へと至る。最後に第 8 章ではテキストや時系列などの系列データに目を向け、自己回帰言語モデリング、バイトレベル BPE トークン化、n-gram モデルとパープレキシティ、**リカレントニューラルネットワーク**（RNN）、通時的誤差逆伝播法（backpropagation through time）、デコーディングを構築し、H. G. Wells の『The Time Machine』風の新しい文章を生成する BPE レベルの言語モデルを訓練してこれらすべてを検証する。

最初の 5 章（第 1 章〜第 5 章）は一貫した背骨をなしており、順番どおりに読み進めるのが最もよい。これらを終えた後は、コンピュータビジョンの章（第 6 章と第 7 章）と系列モデリングの章（第 8 章）は互いに依存しないため、どちらの順序で学んでも構わない。これら全体を通じて、「発展」パートが土台とする概念的・工学的な習慣が築かれる。

Chapter 1

準備

深層学習の世界に飛び込む準備として、いくつか欠かせない必須スキルを身につけておく必要がある。具体的には、(i) 多次元データをメモリに格納して操作する技法（1.1 節）、(ii) 欠損値やカテゴリ型のテキスト項目を含む生ファイルから表形式データを取り込み、クリーニングするツール（1.2 節）、(iii) 高次元表現を変換し、その大きさを測る線形代数の演算を自在に扱う力（1.3 節）、(iv) 損失関数（loss function）を減少させるために各パラメータをどの方向へ調整すべきかを定める微分法（1.4 節）、(v) 数百万個のパラメータにわたる連鎖律（chain rule）をフレームワークに処理させる自動微分（automatic differentiation）の仕組み（1.5 節）、そして (vi) 不確実性のもとで推論し、サンプリングノイズから真のシグナルを見分けるための主要な言語である確率と統計の実践的な理解（1.6 節）である。

古典的な線形代数、微分積分、確率に深層学習が付け加えるものは**規模**（scale）である。手計算など到底不可能な、数百万から数十億もの変数に対してこれらを適用する。これを支えるのが、以降のすべての章で前提となる 2 つのソフトウェアの柱、すなわち CPU や GPU 上で数値配列全体にコンパイル済み演算を並列適用する**テンソルライブラリ**と、任意の数値計算プログラムを機械的に微分する**自動微分**である。

本章では、これらの前提知識を手を動かしながら短時間で身につける速習コースを提供する。NumPy、線形代数、多変数の微分積分にすでに慣れている場合は既知の部分を流し読みしてもよいが、自動微分を扱う 1.5 節と、標準誤差（standard error）および統計的推定を扱う 1.6 節は、以降の章で絶えず用いられるため一読を勧める。完全な証明と導出を備えたより深い理論的基礎については、付録の**深層学習のための数学**で線形代数（第 33 章）、微分積分（第 34 章）、確率と統計（第 36 章）を詳細に展開している。

1.1 データ操作

機械学習で何かを実現するには、メモリ上でデータを効率よく格納し操作する手段が欠かせない。まずは、深層学習において**テンソル**と呼ばれる n 次元配列から始めよう。

専用の配列コンテナが不可欠である理由を見るため、256 枚のグレースケール写真（解像度 28×28 ピクセル）からなる 1 つの**ミニバッチ**を考えよう。ここには $256 \times 28 \times 28 = 200{,}704$ 個の数値が含まれる。画像ファイルは各ピクセルを $\{0, \dots, 255\}$ の整数として格納しており、これを $[0, 1]$ にスケーリングするため 255 で割る。素の Python では、3 重の入れ子ループによってインタプリタ上で 200,704 回のスカラー除算を個別に実行することになる。一方、テンソルライブラリでは `images / 255` という 1 つの式を書くだけでコンパイル済みの C++ や CUDA のカー

ネルが呼び出され、200,704 個の数値すべてが CPU や GPU 上で並列に除算される。深層学習では一貫して、Python のループで個々のスカラー要素を反復処理するのではなく、テンソル全体に対して一度に演算を適用する。

現代のあらゆる深層学習フレームワークにおいて、テンソルクラスは NumPy の `ndarray` に酷似しているが (Harris et al. 2020)、次の 2 つの重要な機能が加わっている。

1. **ハードウェアアクセラレーション**：NumPy が CPU 上で動作するのに対し、フレームワークのテンソルは GPU や専用の AI アクセラレータ上で実行される。
2. **自動微分**（automatic differentiation）：フレームワークはテンソル上の計算を追跡し、誤差逆伝播法（backpropagation）によって勾配（gradient）を自動的に計算する（1.5 節）。

JAX では、テンソルクラスは `jax.Array` である。`jax` と、その NumPy 互換配列インターフェースである `jax.numpy` を `jnp` としてインポートする。

```
import random
import jax
from jax import numpy as jnp
```

1.1.1 テンソルの生成

1.1.1.1 形状

テンソルは、1 つ以上の**軸**（axis）に沿って数値を並べたものである。軸が 1 つのテンソルは**ベクトル**、2 つのテンソルは**行列**、$k > 2$ 個の軸を持つテンソルは k^{th} **階テンソル**である（軸の数をそのテンソルの**階数**（order）といい、コード上では `ndim` として参照できる）。例として挙げたグレースケール画像のミニバッチは 4^{th} 階テンソルであり、その形状（shape）は $(256, 1, 28, 28)$ となる。軸 0 はバッチ内の画像、軸 1 はカラーチャネル（グレースケールでは 1、RGB では 3）、軸 2 は縦方向のピクセル行、軸 3 は横方向のピクセル列をそれぞれ指定する。

`arange(n)` を呼び出すと、0（含む）から n（含まない）まで等間隔に並んだ値の 1 次元ベクトルが生成される。新しく生成されたテンソルはデフォルトで CPU 計算用のメインメモリ上に配置される。ここでは 32 ビット浮動小数点（floating point）の値（`float32`）を指定する。

```
x = jnp.arange(12, dtype=jnp.float32)
x
```

```
Array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.],      dtype=float32)
```

テンソル内の各値は**要素**（element）と呼ばれる。`x` に含まれる要素の総数（**サイズ**）は次のように取得できる。

```
x.size
```

```
12
```

次に、`x.shape` によってテンソルの**形状**タプルを確認する。これは各軸に沿った長さを並べたものである（ここでは 12 要素のベクトルに対する 1 要素タプル `(12,)` となる）。

```
x.shape
```

```
(12,)
```

Figure 1.1 に、階数が 0 から 3 までのテンソルを示す。単一の数値（**スカラー**）は、空の形状 `()` を持つ 0^{th} 階テンソルである。どのようなテンソルでも、そのサイズは形状タプルに含まれる長さの積に等しい（先の画像ミニバッチでは $256 \times 1 \times 28 \times 28 = 200{,}704$）。

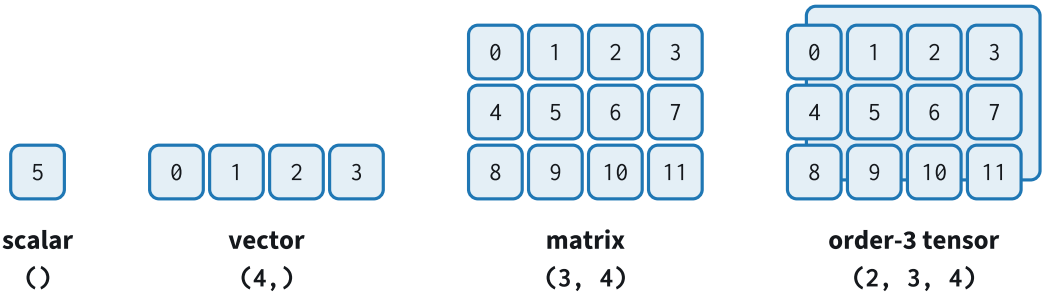


Figure 1.1: 軸が 0、1、2、3 個のテンソル。形状は各軸の長さを並べたものであり、サイズはそれらの長さの積（それぞれ 1、4、12、24 要素）である。

1.1.1.2 データ型

すべてのテンソルは、各要素のバイナリ表現を定める**データ型**（data type、`dtype`）を持つ。深層学習モデルは算術演算の大半を 32 ビット浮動小数点（`float32`、1 要素あたり 4 バイト、10 進で ≈ 7 桁の精度）または 16 ビット形式（`float16` / `bfloat16`、1 要素あたり 2 バイト、5.5 節を参照）で実行する。生の画像ピクセルは通常 8 ビット符号なし整数（`uint8`、各ピクセル 1 バイトで $\{0, \dots, 255\}$ を表す）として格納されるため、この 200,704 ピクセルのミニバッチは `uint8` では 0.2 MB、`float32` では 0.8 MB を占める。

整数引数に対する `jnp.arange` のデフォルトは `int32` であり、キャストには `x.astype(jnp.int32)` を使う。

1.1.1.3 初期値

関数 `zeros` と `ones` は、それぞれ 0 または 1 で満たされた任意の形状のテンソルを生成する（バイアスやバッファに使われる）。

```
jnp.zeros((2, 3, 4))
```

```
Array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]], dtype=float32)
```

ニューラルネットワークの重み（weight）は通常、確率分布から各要素を独立にサンプリングして初期化（initialization）する（4.4 節）。以下では、3×4 行列を標準ガウス分布（平均 0、標準偏差 1）から抽出する。

```
# A key determines the sample: the same key always gives the same numbers
key = jax.random.key(random.getrandbits(32))  # A new seed on each run
```

```
jax.random.normal(key, (3, 4))
```

```
Array([[ 0.21832727,  1.0848601 ,  0.29606727, -0.6687802 ],
       [-0.3902798 ,  0.5969642 , -0.4338682 , -0.09060081],
       [-0.19822368, -0.5707412 , -0.9283469 , -2.4065652 ]],      dtype=float32)
```

コンピュータが生成する乱数は**擬似乱数**（pseudorandom）である。決定論的なアルゴリズムが整数の**シード**から計算するため、同じシードからやり直せばまったく同じ乱数系列が再現される（以下では a と c は一致するが、b は異なる）。

JAX は純粋関数型であり、グローバルな乱数状態を持たない。すべての乱数関数は明示的な PRNG **キー**を受け取り、`jax.random.split` が新しい独立したサブキーを決定論的に導出する。以下では、c が `key_a` を再利用するため、a と同じ結果になる。

```
key_a, key_b = jax.random.split(jax.random.key(42))
a = jax.random.normal(key_a, (3,))
b = jax.random.normal(key_b, (3,))  # A different key
c = jax.random.normal(key_a, (3,))  # Same key: c repeats a
a, b, c
```

```
(Array([ 0.07592554, -0.48634264,  1.2903206 ], dtype=float32),
 Array([ 0.60576403,  0.7990441 , -0.908927  ], dtype=float32),
 Array([ 0.07592554, -0.48634264,  1.2903206 ], dtype=float32))
```

小さなテンソルの各要素があらかじめわかっている場合は、入れ子になった Python のリストから直接生成できる（ここでは 3×4 行列を生成しており、外側のリストが 3 つの行を、内側の各リストが 4 つの列を与える）。

```
jnp.array([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
```

```
Array([[2, 1, 4, 3],
       [1, 2, 3, 4],
       [4, 3, 2, 1]], dtype=int32)
```

最後に、本節を通して用いる合成ミニバッチ `images` を、形状 `(256, 1, 28, 28)`・データ型 `uint8` として、$\{0, \ldots, 255\}$ のランダムな整数で満たして生成する。

```
# 256 grayscale images of 28 x 28 pixels, each pixel an integer in [0, 255]
key, subkey = jax.random.split(key)
images = jax.random.randint(subkey, (256, 1, 28, 28), 0, 256,
                            dtype=jnp.uint8)
images.shape, images.dtype
```

```
((256, 1, 28, 28), dtype('uint8'))
```

1.1.2 形状変換とインデックス参照

1.1.2.1 形状変換

`reshape` を呼び出すと、要素数とメモリ上の行優先順（row-major order）を保ったまま、テンソルの軸を再構成する（Figure 1.2）。ここでは、12 要素のベクトル `x` を 3×4 行列 `X` に形状変換する。

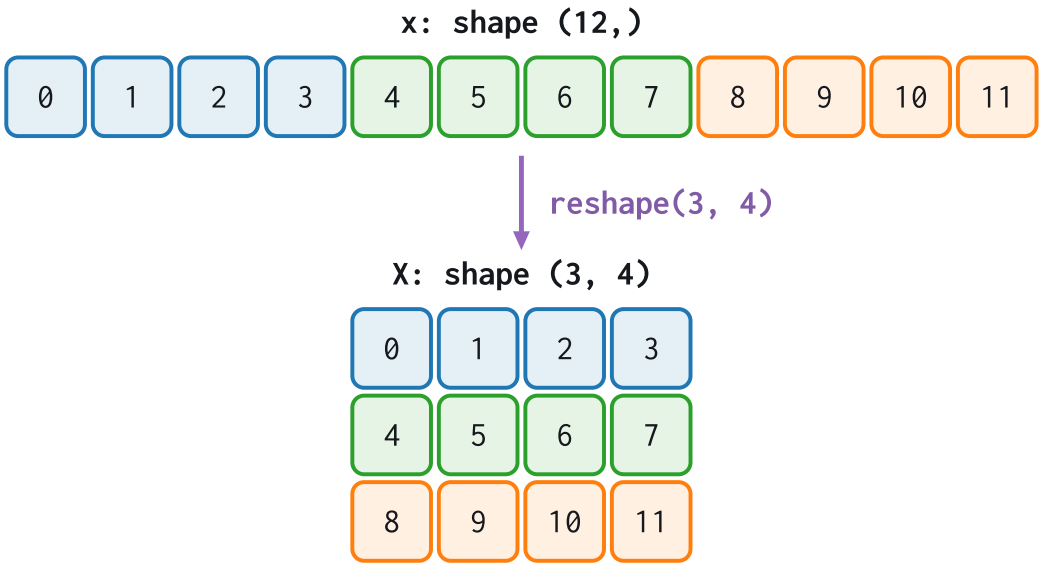


Figure 1.2: 12 要素のベクトルから 3×4 行列への形状変換。要素の順序は保たれ、行列を 1 行ずつ埋めていく。

```
X = x.reshape(3, 4)
X
```

```
Array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]], dtype=float32)
```

全体のサイズは固定されているため、1 つの次元に -1 を渡すと、その長さがフレームワークによって自動的に決まる。たとえば、$(256, 1, 28, 28)$ の画像ミニバッチを、1 行あたり 1 枚の画像を持つ $(256, 784)$ 行列へと平坦化（flatten）できる。

```
images.reshape(256, -1).shape
```

```
(256, 784)
```

None を使ってインデックス参照すると、その位置に長さ 1 の新しい単一軸（singleton axis）が挿入され、一方で squeeze は長さ 1 の軸を取り除く。

```
image = images[0]
image[None].shape, image.squeeze(0).shape
```

```
((1, 1, 28, 28), (28, 28))
```

形状変換は要素の順序を保ったままグループ分けを変えるだけなので、2 つの軸を入れ替えることは決してできない。ライブラリによっては、チャネル軸を末尾に置いた $(256, 28, 28, 1)$ の形式で画像ミニバッチを受け取るものがある。ここまでの $(256, 1, 28, 28)$ のミニバッチからこの形式を得るには、軸 1 を末尾へ移す必要がある。これが**転置**（transpose）であり、元の軸を新し

い順序（ここでは $(0, 2, 3, 1)$）で指定する。行列における軸の並べ替えは行と列の入れ替えの 1 通りしかなく、`X.T` と書く。これを同じ形状への形状変換と比較してみよう。

```
images.transpose(0, 2, 3, 1).shape, X.reshape(4, 3), X.T
```

```
((256, 28, 28, 1),
 Array([[ 0.,  1.,  2.],
        [ 3.,  4.,  5.],
        [ 6.,  7.,  8.],
        [ 9., 10., 11.]], dtype=float32),
 Array([[ 0.,  4.,  8.],
        [ 1.,  5.,  9.],
        [ 2.,  6., 10.],
        [ 3.,  7., 11.]], dtype=float32))
```

どちらの行列も 4×3 であるが、形状変換の方は依然として行方向に $0, 1, 2, \ldots$ と並んでいるのに対し、転置の方は `X` の行が列へと変わっている。形状は一致するため、転置すべき箇所に誤って形状変換を書いてもエラーは発生しない。正しい数値を誤った配置のまま次の層へ渡してしまうだけである。

1.1.2.2 インデックス参照とスライス

要素や部分テンソルには、軸ごとにカンマで区切った 0 始まりの整数インデックス参照（負のインデックスは末尾から逆向きに数える）および半開区間のスライス `start:stop`（`start` を含み、`stop` を含まない）でアクセスする。

```
X[-1], X[1:3]
```

```
(Array([ 8.,  9., 10., 11.], dtype=float32),
 Array([[ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]], dtype=float32))
```

形状の違いに注目しよう（Figure 1.3）。整数インデックス（`X[-1]`）はその軸をつぶして形状 `(4,)` のベクトルを返すのに対し、スライス（`X[1:3]`）は軸を保ったまま形状 `(2, 4)` の行列を返す。

形状 $(256, 1, 28, 28)$ の画像ミニバッチに対して、`images[0]` は最初の画像 $(1, 28, 28)$ を取り出し、`images[:32]` は最初の 32 枚の画像 $(32, 1, 28, 28)$ をスライスし、`images[:, 0, 14, 14]` は全 256 枚の画像にわたって中央のピクセル $(256,)$ を取り出す。

```
images[0].shape, images[:32].shape, images[:, 0, 14, 14].shape
```

```
((1, 28, 28), (32, 1, 28, 28), (256,))
```

整数インデックステンソル（integer index tensor）を使ってテンソルをインデックス参照することもできる（**ギャザー**、gather）。2 つの 1 次元整数テンソル `rows = (0, 1, 2)` と `cols = (3, 0, 2)` を渡すと、要素 `(0, 3)`、`(1, 0)`、`(2, 2)` が選択される。

```
rows, cols = jnp.arange(3), jnp.array([3, 0, 2])
X[rows, cols]
```

```
Array([ 3.,  4., 10.], dtype=float32)
```

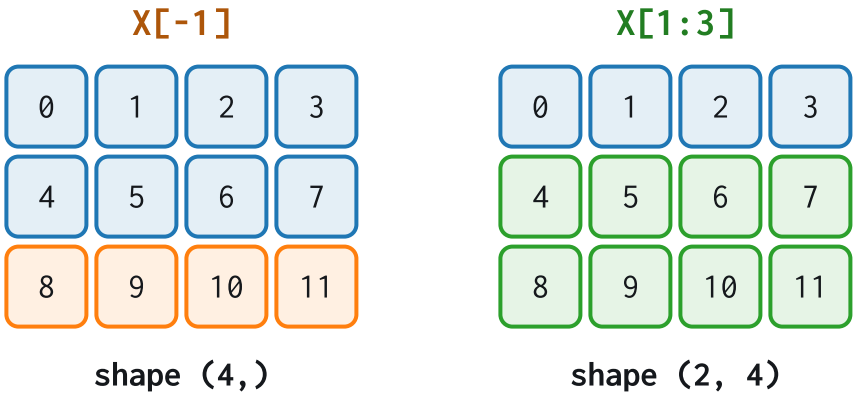


Figure 1.3: 3×4 行列のインデックス参照。整数インデックス `X[-1]` は最後の行を選択して軸 0 を取り除き、形状 $(4,)$ を残す。スライス `X[1:3]` は行 1 と行 2 を選択して軸 0 を保持し、形状 $(2, 4)$ を残す。

同様に、要素ごとの比較からは `True` と `False` の値からなる**ブールマスク**が得られる（`&`、`|`、`~` で組み合わせ可能）。`X[mask]` でインデックス参照すると、`mask` が `True` であるすべての要素が平坦化された 1 次元ベクトルとして取り出される。

`X[mask]` の出力長はデータに依存するため、ブールマスクによるインデックス参照は即時実行（eager）モードでのみ動作する。`jax.jit` でコンパイルされた関数の内部では `jnp.where` を使う。

```
mask = (X > 2) & (X < 8)
X[mask]
```

```
Array([3., 4., 5., 6., 7.], dtype=float32)
```

これに対し、3×4 という `X` の形状を保ちたい場合は、`where(mask, X, 0)` を使うことで、`X` のうち `mask` が `True` の位置の要素を選択し、残りのすべての位置を `0` で埋めることができる。

```
jnp.where(mask, X, 0)
```

```
Array([[0., 0., 0., 3.],
       [4., 5., 6., 7.],
       [0., 0., 0., 0.]], dtype=float32)
```

Figure 1.4 に、これら 3 つの選択方法を並べて示す。違いは返される形状にあり、インデックステンソルは各行から 1 つの要素を返し、マスクはデータに応じた長さのベクトルを返し、`where` は `X` と同じ形状のテンソルを返す。

1.1.2.3 要素の書き込み

テンソルをインプレースで上書きできるかどうかは、フレームワークの設計に依存する。

JAX の配列は**イミュータブル**である。JAX は関数型の更新 `X.at[1, 2].set(17)` を備えており、`X` は変更せずに新しい配列を返す。

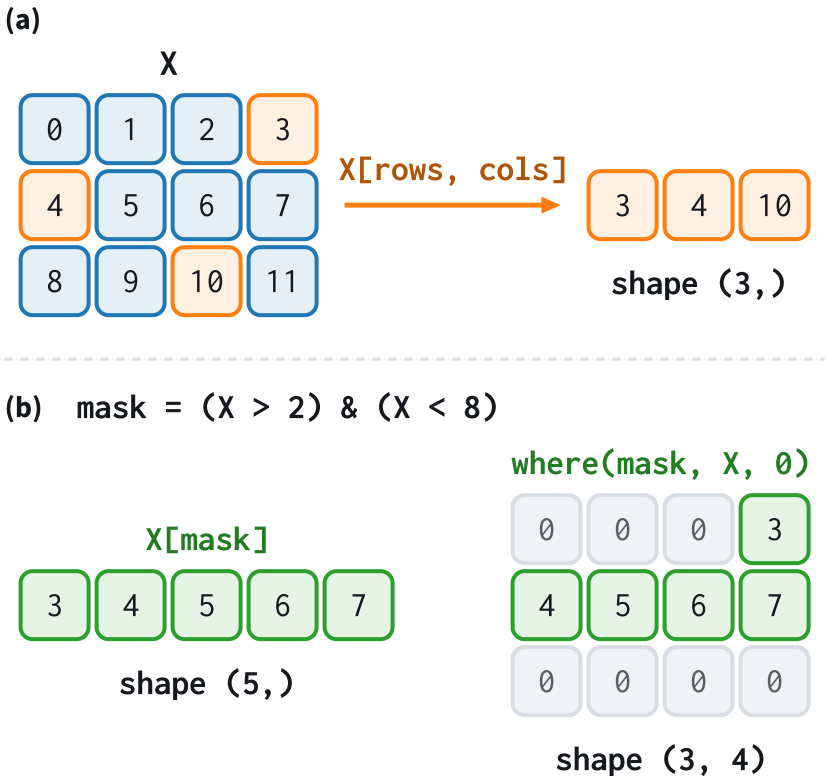


Figure 1.4: 3×4 行列 `X` からの要素の選択。(a) インデックステンソル `rows = (0, 1, 2)` と `cols = (3, 0, 2)` は各行から 1 つの要素を選択する。(b) マスク `(X > 2) & (X < 8)` は 2 より大きく 8 より小さい 5 つの要素をベクトルとして取り出すのに対し、`where` は `X` の形状を保ち、それ以外のすべての位置に 0 を書き込む。

```
X_new = X.at[1, 2].set(17)  # X is unchanged
X_new
```

```
Array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5., 17.,  7.],
       [ 8.,  9., 10., 11.]], dtype=float32)
```

代入の左辺でスライスを使うと、ブロック全体に一度に値を代入できる。

```
X_new = X_new.at[:2, :].set(12)
X_new
```

```
Array([[12., 12., 12., 12.],
       [12., 12., 12., 12.],
       [ 8.,  9., 10., 11.]], dtype=float32)
```

1.1.3 演算

1.1.3.1 要素ごとの演算

標準的な二項算術演算子（+、-、*、/、**）はスカラーの算術演算を拡張し、同じ形状を持つ 2 つのテンソル x と y の対応する要素の組ごとに独立して作用する。

```
x = jnp.array([1.0, 2, 4, 8])
y = jnp.array([2.0, 2, 2, 2])
x + y, x - y, x * y, x / y, x ** y
```

```
(Array([ 3.,  4.,  6., 10.], dtype=float32),
 Array([-1.,  0.,  2.,  6.], dtype=float32),
 Array([ 2.,  4.,  8., 16.], dtype=float32),
 Array([0.5, 1. , 2. , 4. ], dtype=float32),
 Array([ 1.,  4., 16., 64.], dtype=float32))
```

exp のような単項数学関数も同様に要素ごとに適用され、e^{x_i} を x の各要素に対して計算する。

```
jnp.exp(x)
```

```
Array([2.7182817e+00, 7.3890562e+00, 5.4598148e+01, 2.9809580e+03],      dtype=float32)
```

uint8 型のミニバッチ images を 255 で割ると、全 200,704 ピクセルが一度にスケーリングされ、結果の型は float32 に昇格（type promotion）する。

```
scaled = images / 255
print(scaled.dtype, images[0, 0, 0, :4], scaled[0, 0, 0, :4], sep='\n')
```

```
float32
[204 119 139 153]
[0.79999995 0.46666664 0.545098   0.59999996]
```

x == y のような関係演算子も要素ごとに作用し、True を x と y が一致する位置に持つブールテンソルを返す（ここでは両方とも 2.0 に等しいインデックス 1 のみ）。

```
x == y
```

```
Array([False,  True, False, False], dtype=bool)
```

sum() でテンソルの総和をとると、すべての要素が1つのスカラーに縮約される（$1+2+4+8=15$ が x の総和）。True は 1、False は 0 として数えられるため、ブールマスクの総和 (x == y).sum() によって一致する要素の個数を数えられる。

```
x.sum(), (x == y).sum()
```

```
(Array(15., dtype=float32), Array(1, dtype=int32))
```

1.1.3.2 ブロードキャスト

4 要素のバイアスベクトル v を 3×4 行列 X のすべての行に加える場合のように、**異なる**形状のテンソル同士を組み合わせることは多い。

```
X = jnp.arange(12, dtype=jnp.float32).reshape(3, 4)
v = jnp.array([100.0, 200, 300, 400])
X + v
```

```
Array([[100., 201., 302., 403.],
       [104., 205., 306., 407.],
       [108., 209., 310., 411.]], dtype=float32)
```

テンソルライブラリは物理的なコピーを実体化する代わりに、2 段階の**ブロードキャスト**規則を適用する。

1. **形状を右揃えにする：**両方の形状タプルが同じ長さになるまで、階数の低いテンソルの形状タプルの左側に 1 を補う。揃えられた 2 つの次元は、長さが等しいか、または片方が 1 であれば互換性がある。
2. **単一軸を引き伸ばす：**メモリをコピーすることなく、長さ 1 の軸をもう一方のテンソルの長さ $k > 1$ に合わせて仮想的に複製し、その上で要素ごとの演算を適用する。

両方のテンソルが同時に引き伸ばされる様子を見るため（Figure 1.5）、3×1 の列ベクトル a と 1×2 の行ベクトル b を作ってみよう。

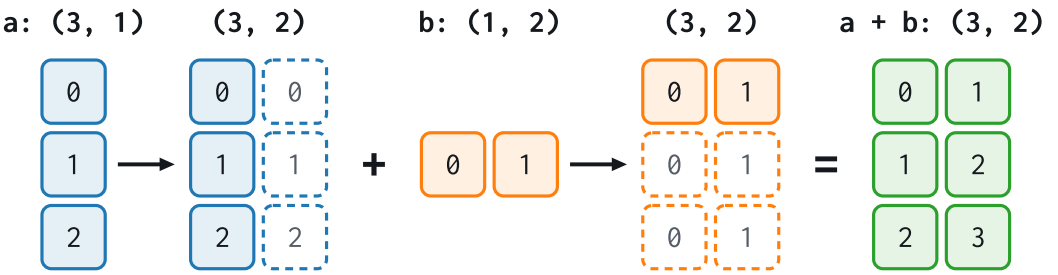


Figure 1.5: ブロードキャストは長さ 1 の軸を引き伸ばす。a（形状 $(3,1)$ の列）と b（形状 $(1,2)$ の行）は、要素ごとの加算の前にどちらも形状 $(3,2)$ へと拡張される（破線は仮想的なコピー）。

```
a = jnp.arange(3).reshape(3, 1)
b = jnp.arange(2).reshape(1, 2)
a, b
```

```
(Array([[0],
        [1],
        [2]], dtype=int32),
 Array([[0, 1]], dtype=int32))
```

`a + b` を計算すると、`a` が 2 つの列に、`b` が 3 つの行に仮想的に複製され、3×2 行列としてすべての対の和 $a_{i0} + b_{0j}$ が生成される。

```
a + b
```

```
Array([[0, 1],
       [1, 2],
       [2, 3]], dtype=int32)
```

揃えられた次元の組のうち、長さが異なりかつどちらも 1 でないものがある場合（たとえば `(3, 2)` と `(2, 3)`）、ブロードキャストはエラーを送出する。

```
try:
    jnp.ones((3, 2)) + jnp.ones((2, 3))
except Exception as e:
    print(e)
```

```
add got incompatible shapes for broadcasting: (3, 2), (2, 3).
```

したがって、$(28, 28)$ の平均画像を `images`（形状 $(256, 1, 28, 28)$）から引くと $(1, 1, 28, 28)$ として問題なくブロードキャストされる一方、画像ごとの値を持つ形状 $(256,)$ のベクトルを平坦化された画像 $(256, 784)$ に足す演算は、列 $(256, 1)$ に形状変換しない限り失敗する。

エラーメッセージが出るのはむしろ幸運な場合である。形状 $(3,)$ のラベルを形状 $(3, 1)$ の予測から引く演算は同じ規則のもとで正当とみなされ、本来欲しかった 3 個の誤差ではなく、すべての対の差からなる 3×3 行列を返してしまう（Figure 1.6）。その行列全体で平均をとった損失もやはり 1 つの数値になるため、モデルが学習に失敗するまで異常に気づきにくい。

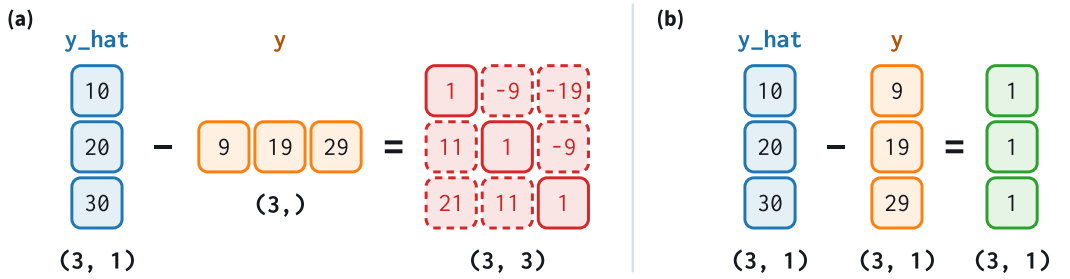


Figure 1.6: エラーメッセージの出ないブロードキャスト。(a) 予測 `y_hat`（形状 $(3, 1)$）からラベル `y`（形状 $(3,)$）を引くと、すべての対の差 9 個が得られ、対角成分（実線）だけが本来欲しかった 3 個の誤差を保持する。(b) `y` を $(3, 1)$ に形状変換すれば、同じ減算からその 3 個の誤差だけが返される。

1.1.3.3 連結

concatenate（または cat / concat）は、**既存**の軸に沿ってテンソルを端から端へ連結（concatenation）する（それ以外のすべての軸は長さが一致していなければならない）。2 つの 3×4 行列 X と Y を axis=0 に沿って連結すると行が積み重なって形状 (6, 4) となり、axis=1 に沿って連結すると列がつながって形状 (3, 8) となる。

```
X = jnp.arange(12, dtype=jnp.float32).reshape(3, 4)
Y = jnp.array([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
jnp.concatenate((X, Y), axis=0), jnp.concatenate((X, Y), axis=1)
```

```
(Array([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.],
        [ 2.,  1.,  4.,  3.],
        [ 1.,  2.,  3.,  4.],
        [ 4.,  3.,  2.,  1.]], dtype=float32),
 Array([[ 0.,  1.,  2.,  3.,  2.,  1.,  4.,  3.],
        [ 4.,  5.,  6.,  7.,  1.,  2.,  3.,  4.],
        [ 8.,  9., 10., 11.,  4.,  3.,  2.,  1.]], dtype=float32))
```

これに対し、stack は同じ形状のテンソルを**新しい**軸に沿って結合し、X と Y を束ねて 3^{rd} 階テンソルを作る。形状はデフォルトで (2, 3, 4)（axis=0）となり、(3, 4, 2) になるのは axis=2 に沿って重ねた場合である。

```
jnp.stack((X, Y)).shape, jnp.stack((X, Y), axis=2).shape
```

```
((2, 3, 4), (3, 4, 2))
```

1.1.4 メモリと変換

1.1.4.1 メモリ確保とインプレース更新

Y = Y + X のような標準的な式は新しいバッファを確保し、Y をその新しいアドレスに再束縛する。これは Python の id() で確認できる。

```
before = id(Y)
Y = Y + X
id(Y) == before
```

```
False
```

数ギガバイトものパラメータを持つモデルでは、不要なメモリ確保はメモリ帯域幅（memory bandwidth）を浪費し、他の参照が古いバッファを指したままになる原因にもなる（ただし順伝播（forward pass）の内部では、自動微分でまだ必要とされる活性化値（activation）を上書きしてはならない。1.5 節を参照）。

JAX では配列はイミュータブルであり、バッファの再利用は @jax.jit のもとで XLA コンパイラが自動的に処理する。

X の古い値がもはや不要なとき、累積代入 X += Y は和を X 自体に書き込むよう要求する。実際にそうできたかどうかは id(X) で確認できる。

```
before = id(X)
X += Y
id(X) == before
```

```
False
```

X はイミュータブルなテンソルであるため、`id(X) == before` は `False` となる。

1.1.4.2 他の Python オブジェクトへの変換

描画ライブラリやデータライブラリとの境界では、フレームワークのテンソルと NumPy 配列の相互変換がよく使われるが、変換時にメモリを共有するかコピーするかはフレームワークによって異なる。

`jax.device_get(X)` は読み取り専用の NumPy 配列を返し（CPU 上ではゼロコピーになりうる）、`jax.device_put(A)` がそれを JAX 側へ戻す。この NumPy ビューは読み取り専用であるため、以下の書き込みは `ValueError` を送出する。

```
A = jax.device_get(X)
B = jax.device_put(A)
type(A), type(B)
```

```
(numpy.ndarray, jaxlib._jax.ArrayImpl)
```

A、B、X がメモリを共有しているかどうかは、NumPy 配列上で `A[0, 0] = 100` と上書きしてから `X[0, 0]` と `B[0, 0]` を調べることで確認できる。

```
try:
    A[0, 0] = 100
except ValueError as e:
    print(e)
```

```
assignment destination is read-only
```

単一要素の（0^{th} 階）テンソルを Python の `float` や `int` として取り出すには（たとえばスカラーの損失をログに記録するとき）、`.item()`、`float()`、または `int()` を呼び出す。ただし、これによって実行待ちの GPU カーネルと CPU との同期が強制される点に注意しよう。

```
a = jnp.array(3.5)
a, a.item(), float(a), int(a)
```

```
(Array(3.5, dtype=float32, weak_type=True), 3.5, 3.5, 3)
```

> **ヒント：フレームワーク API の調べ方。**見慣れないモジュールや関数に出会ったら、`dir(obj)`（または Tab 補完）で利用可能な属性を一覧し、Jupyter 上で `help(fn)` や `fn?` を使ってシグネチャや docstring を確認し（Python のソースコードを見るには `fn??`）、手計算で期待される出力を予測できるごく小さなテスト入力で理解を確かめよう。

1.1.5 まとめ

テンソルは、入力、パラメータ、活性化値、勾配を、`shape` と `dtype` によって特徴づけられる n 次元配列として格納する。深層学習のコードはテンソル全体に対する計算を一度に記述し、ミニバッチの全 $200{,}704$ ピクセルのスケーリングもコンパイル済みの 1 回の演算（`images / 255`）で処理する。軸の再構成には `reshape`、部分集合の抽出や変更には整数インデックス・スライス・ブールマスク（`where`）、テンソルの結合には `concatenate` と `stack` を用い、形状の異なるテンソル同士は**ブロードキャスト**（形状を右揃えにして単一軸を引き伸ばす仕組み）によって組み合わせる。最後に、テンソルがミュータブルである場合はインプレース更新（`Z[:] = ...`、`X += Y`）によってメモリを再利用し、そうでない場合はミュータブルなコンテナやコンパイラがその役割を担う。

1.1.6 演習問題

1. [code] **形状変換における軸の自動決定**。n 枚のグレースケール画像（解像度 28×28）からなるミニバッチが形状 $(n, 1, 28, 28)$ のテンソルとして与えられ、各画像を 784 個のピクセル値からなる行ベクトルへ平坦化しなければならないとする。以下の各予測をコードで検証せよ。

    1. ミニバッチを `(-1, 784)` に形状変換した場合と `(256, -1)` に形状変換した場合の出力形状を、まず $n = 256$ 枚の通常のミニバッチについて、次にエポック最後の $n = 100$ 枚の小さなミニバッチについて予測せよ。2 つの目標形状のうち、**任意**のバッチサイズ n に対して確実に動作するのはどちらか。
    2. 画像が誤って 3 チャネルの RGB 形式（形状 $(n, 3, 28, 28)$）で読み込まれたとする。このテンソルを `(-1, 784)` に形状変換するとどうなるか。`reshape(-1, 784)` はチャネル数の誤りを検出するか、それともエラーを出さずにバッチ次元をかき混ぜてしまうか。

2. [code] **ブロードキャストの形状と暗黙のバグ**。ブロードキャストは 2 つの形状タプルを右から揃える。

    1. すべての軸で長さが異なるにもかかわらずブロードキャスト可能な 2 つの 3^{rd} 階テンソル `a` と `b` を構成せよ。`a + b` の形状を手計算で予測し、コードを実行して予測を検証せよ。
    2. ある回帰モデルが、5 個のデータ例に対して予測の列 `y_hat`（形状 $(5, 1)$）を出力し、正解ターゲット `y` が形状 $(5,)$ の 1 次元ベクトルとして格納されているとする。残差 `y_hat - y` の形状を予測せよ。次に、`y` に `[0.0, 1.0, 2.0, 3.0, 4.0]` を格納し、すべての予測が真のラベルと完全に一致するよう `y_hat = y.reshape(5, 1)` と設定せよ。`y_hat - y` の各要素の二乗の平均を計算し、予測が完璧であるにもかかわらずこの「平均二乗誤差」が 0 にならない理由を説明せよ。さらに、各予測が自身のラベルとのみ比較されるように `y`（または `y_hat`）を形状変換する方法を示せ。

3. [code] **データ型の算術的な限界**。`images` の生のピクセルは 8 ビット符号なし整数（`uint8`）として格納される一方、ニューラルネットワークの層は浮動小数点（`float32`）で計算する。

    1. `100` を、`uint8` テンソルの値 `200` を保持する要素に加算したときの数値結果を予測し、コードで検証せよ。このようなラップアラウンド（桁あふれによる回り込み）が起きる理由を説明し、整数のオーバーフローを起こさずに `images` の最初の画像を輝度 100 だけ明るくする方法（クランプまたは型の昇格による方法）を示せ。
    2. IEEE 754 の `float32` 数値は仮数部に 24 ビットの精度を割り当てている。`float32` が $2^{24} = 16{,}777{,}216$ までのすべての連続する整数を正確に表現できる理由を説明し、`float32(2**24) + float32(1)` を評価しても値が $16{,}777{,}216$ のまま変化しないことをコードで確認せよ。訓練ループで処理したデータ例またはトークンの総数を `float32` のスカラーカウンタ（毎回 1 ずつ加算）に累積していくと、何個のデータ例を処理した時点でカウンタが完全に停止するか。また、代わりにどのデータ型を使うべきか。

4. [code] **インプレース演算の合成とコンパイラ融合。**●JAX の配列はイミュータブルであるため、即時実行での算術演算はすべて新しい配列を確保する。`(A + B) * (-A / 2)` を Python 関数 `f(A, B)` の内部に記述し、即時実行によって中間配列がいくつ生成されるかを数えよ。次に、`f` を `jax.jit(f)` でコンパイルし、`jax.jit(f).lower(A, B).compile().as_text()` によって変換後の HLO プログラムを調べ、融合されたカーネルがいくつ残るかを確認せよ。このコンパイル実行を、`A` を上書きするミュータブルな NumPy のインプレース実装と比較せよ。`jax.jit` は即時実行時の中間確保のうちどれを自動的に除去するか。また、コンパイルされた関数が依然として出力バッファを確保するのはどのような場合か（`donate_argnums` を指定しない場合）。

   Nicolas P. Rougier「100 numpy exercises」問題 35 を改変。

5. [code] **Python のループとブロードキャスト。**●200 要素の 1 次元ベクトル `a` と 300 要素の 1 次元ベクトル `b` が与えられたとき、すべての対の絶対差からなる 200×300 行列 D（要素は $d_{ij} = |a_i - b_j|$）を計算したい。

   1. まず 2 重の入れ子になった Python の `for` ループを使って D を計算し、次に `a` を `(200, 1)` に、`b` を `(1, 300)` に形状変換した上で 1 回のブロードキャスト減算によって計算せよ。両方の行列が数値的に一致することを確認し、各手法の実経過時間（ウォールクロック時間）を計測して高速化倍率を報告せよ。
   2. ブロードキャストは入力 `a` と `b` のコピーを回避するが、出力行列 D は依然としてメモリ上に実体化されなければならない。D を `float32` で保持するとき、`a` と `b` がそれぞれ 10,000 要素を含むならば何メガバイトを占めるか。この結果から、ブロードキャストを用いた外積型の演算におけるピーク時のメモリ使用量についてどのような教訓が得られるか。

1.2 データの前処理

これまでは、望みどおりの形状とデータ型を備えた既製のテンソルとして与えられる合成データ（synthetic data）を扱ってきた。しかし、実世界で深層学習を応用するには、任意のファイル形式に保存された雑多なデータを取り出し、整った数値テンソルへと前処理しなければならない。生のレコードの表をモデルに入力できるテンソルへと変換する作業には、**前処理**（preprocessing）という控えめな言葉から想像される以上に多くの統計的な判断が伴う。

本節を通して用いる例として、10 件の住宅販売記録からなる小さな表を考えよう。この表には、部屋数（`NumRooms`）、屋根材（`RoofType`）、平方フィート単位の床面積（`Area`）、ドル単位の販売価格（`Price`）が記録されている。小さいながらも、この表には生の表形式データをそのままニューラルネットワークに入力することを阻む 3 つの障害が現れている。

1. **欠損値：**10 件の住宅のうち 3 件で部屋数が欠けており、4 件で屋根材が欠けている。
2. **カテゴリ型のテキスト項目：**屋根材は単語（`Slate` または `Tile`）として記録されているが、ニューラルネットワークは数値のみで計算する。
3. **スケールの不一致：**平方フィート単位の床面積は、数値として部屋数の約 500 倍も大きい。

いずれの障害に対しても統計的な選択が求められる。不完全な行を破棄すれば表の大半が失われ、順序のない屋根材を `0`, `1`, `2` と符号化すれば架空の順序関係を作り出すことになり、床面積をスケーリングしないままにすれば距離や勾配が床面積に支配されてしまう。Python における表形式データの標準ツールは pandas ライブラリである (McKinney 2010)。以下では、Figure 1.7 に示す前処理パイプラインに沿って、生の CSV ファイルから `float32` の特徴量テンソルとターゲットテンソルを作成する手順を順に見ていく。

まず、`pandas` を慣例的な別名 `pd` としてインポートし、あわせてファイルパス操作のための Python の `os` モジュールと、深層学習フレームワーク（整えた表を最後のステップでテンソルに変換する

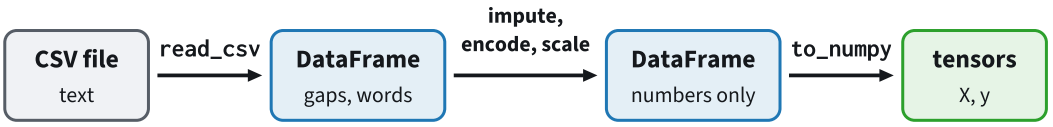


Figure 1.7: ファイルからテンソルへ。read_csv は欠損やテキスト列を含むテキストファイルを DataFrame に読み込む。補完、エンコーディング、スケーリングによってすべての要素を数値に変換し、to_numpy がその結果をテンソル X および y としてフレームワークに渡す。

際に必要となる）をインポートする。

```
import os
import pandas as pd
from jax import numpy as jnp
```

1.2.1 読み込みと確認

1.2.1.1 CSV ファイル

カンマ区切り値（comma-separated values、CSV）ファイルは、1 行目に列見出し、2 行目以降にカンマで区切られたレコードを 1 行ずつ並べることで、表形式データをプレーンテキストとして保存する。ここでは 10 件の住宅データセットを ../data/house_tiny.csv に書き出す。連続するカンマの間に挟まれた空のフィールド（,Tile,2100,290000 など）は、記録されていない値を表す。

```
os.makedirs(os.path.join('..', 'data'), exist_ok=True)
data_file = os.path.join('..', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('''NumRooms,RoofType,Area,Price
3,Slate,1500,210000
,Tile,2100,290000
2,,850,127500
4,Slate,1940,258000
,,1200,168000
3,Tile,1650,225000
5,Slate,2600,375000
2,,900,142000
,Tile,1750,240000
4,,2050,295000''')
```

pd.read_csv を呼び出すと、ファイルが解析されて pandas の **DataFrame** に読み込まれる。これは名前と型を持つ列からなる 2 次元の表であり、左端には行インデックス（0 から 9）が付いている。

```
data = pd.read_csv(data_file)
data
```

```
   NumRooms RoofType  Area   Price
0       3.0    Slate  1500  210000
1       NaN     Tile  2100  290000
2       2.0      NaN   850  127500
```

```
3       4.0    Slate  1940  258000
4       NaN      NaN  1200  168000
5       3.0     Tile  1650  225000
6       5.0    Slate  2600  375000
7       2.0      NaN   900  142000
8       NaN     Tile  1750  240000
9       4.0      NaN  2050  295000
```

pandas は、すべての空フィールド（および NA のような標準的なトークン）を、NaN（**非数**、not a number）へと自動的に置き換えている点に注目しよう。これは**欠損値**（missing value）を表すセンチネル値である。

1.2.1.2 データ型と要約

新しく読み込んだデータセットに変更を加える前に、data.info() を使って行数・列数、非欠損の要素数、および自動判定されたデータ型を確認しよう。

```
data.info()
```

```
<class 'pandas.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   NumRooms  7 non-null      float64
 1   RoofType  6 non-null      str
 2   Area      10 non-null     int64
 3   Price     10 non-null     int64
dtypes: float64(1), int64(2), str(1)
memory usage: 481.0 bytes
```

全 10 行のうち、Area と Price には欠損がなく（int64）、NumRooms には 7 個の観測値があり（float64）、RoofType には 6 個の観測値がある（str、古いバージョンの pandas では object）。NumRooms は整数の部屋数を表すにもかかわらず、float64 に格上げされている点に注目しよう。IEEE 754 規格では NaN は浮動小数点のビットパターンとして定義されているため、標準的な整数配列では欠損要素を表現できないからである。

数値列については、data.describe() によって観測値の個数、平均、標本標準偏差（std）、最小値、四分位数、最大値が表示される。

```
data.describe()
```

```
       NumRooms         Area          Price
count  7.000000    10.000000      10.000000
mean   3.285714  1654.000000  233050.000000
std    1.112697   556.421104   76128.710455
min    2.000000   850.000000  127500.000000
25%    2.500000  1275.000000  178500.000000
50%    3.000000  1700.000000  232500.000000
75%    4.000000  2022.500000  282000.000000
max    5.000000  2600.000000  375000.000000
```

NumRooms は 2 から 5（平均 3.29）の範囲にあるのに対し、Area は 850 から 2,600 平方フィート（平均 1,654）、Price は $127,500 から $375,000 の範囲にわたっている。

1.2.1.3 入力とターゲットの分離

教師あり学習（supervised learning）では、指定された**ターゲット**（または**ラベル**）列である Price を、残りの**入力**列、すなわち**特徴量**（feature）列（NumRooms、RoofType、Area）から予測する。最初に特徴量をターゲットから分離しておけば、後続の特徴量変換がターゲットに手を加えたりターゲットの情報を漏洩させたりするのを防げる。.iloc による 0 始まりの位置スライシングを使うと、data.iloc[:, :3] で最初の 3 列が選択され、data.iloc[:, 3] で 4 列目が選択される。

```
inputs, targets = data.iloc[:, :3], data.iloc[:, 3]
list(inputs.columns), targets.name
```

```
(['NumRooms', 'RoofType', 'Area'], 'Price')
```

位置によるスライシングは簡潔だが、実運用のパイプラインでは CSV の列が挿入されたり並び替えられたりすると、data.iloc[:, 3] が気づかないうちに誤ったターゲットを取得してしまう。列名で選択すれば（targets = data['Price'] および inputs = data.drop(columns='Price')）、このリスクを回避できる（演習問題 1）。

1.2.2 クリーニングとエンコーディング

1.2.2.1 欠損値

センサの脱落やアンケートの未回答、過去の記録漏れなどにより、欠損値は日常的に発生する。.isna() に .sum() をつなげて呼び出すと、各列の欠損要素数が集計される（True が 1 として数えられる）。

```
inputs.isna().sum()
```

```
NumRooms    3
RoofType    4
Area        0
dtype: int64
```

これには 3 つの標準的な対処法があり、それぞれ異なる統計的トレードオフを持つ（Figure 1.8）。

- **削除**（deletion）は、欠損値を含むすべての行（または列）を破棄する手法である。実装はきわめて簡単だが、データが失われるうえ、値が完全にランダムに欠損しているのでない限り（たとえばワンルームマンションで部屋数の記載漏れが偏って多い場合など）、残った標本に統計的バイアスが生じる。
- **補完**（imputation）は、各 NaN を列の平均や中央値などの推定値で置き換える手法である。すべての行を保持できる反面、近似値で代用するため、その列の見かけ上のばらつきが小さくなる。
- **欠損性インジケータ**（missingness indicator）は、各要素がもともと欠損していた**かどうか**を記録する追加の二値（0/1）列を補完と組み合わせる手法である。補完した列ごとに特徴量が 1 つ増える引き換えに、情報を持つ欠損性（たとえば部屋数の未報告が主に高級邸宅で生じる場合など）をモデルが活用できるようになる。

この 10 件の住宅の表では、行の削除（inputs.dropna()）を実行するとデータがほとんど残らない。

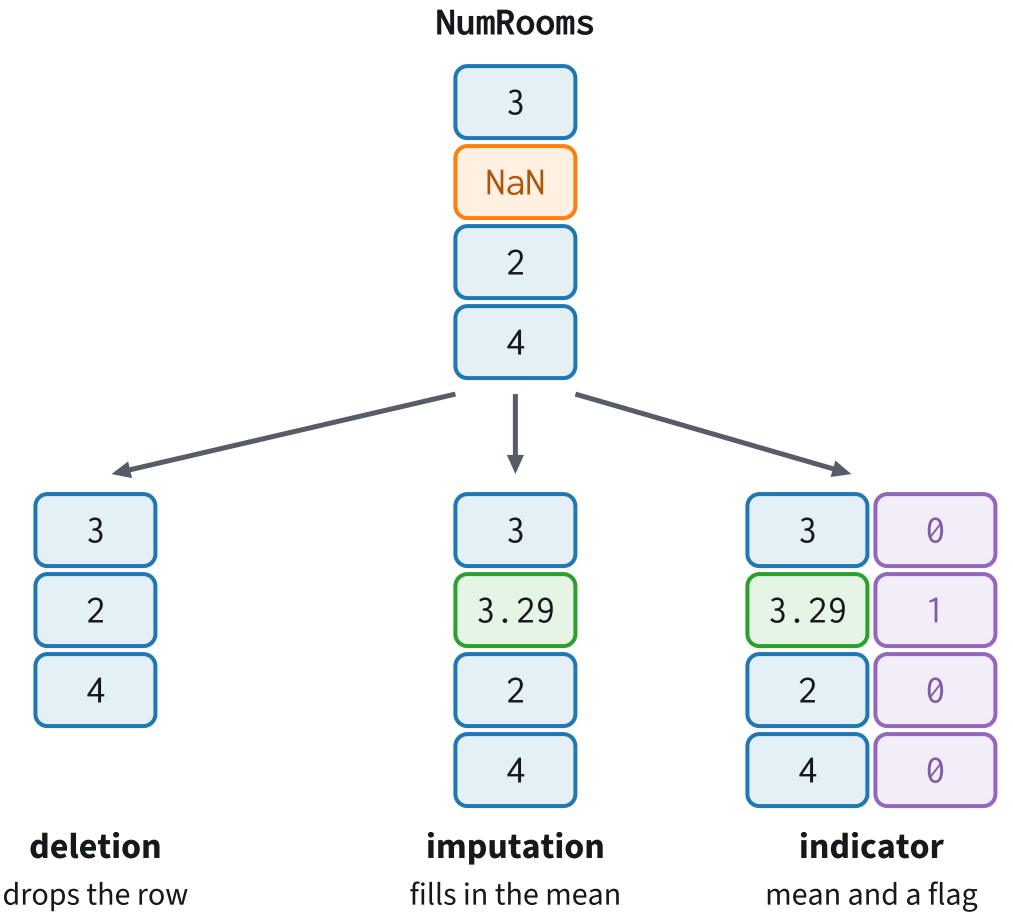


Figure 1.8: NumRooms の最初の 4 行に対する、部屋数の欠損への 3 通りの対処法。削除はその行を除外する。補完は観測された 7 個の値の平均 $23/7 \approx 3.29$ で埋める。欠損性インジケータは同じ平均で埋めたうえで、どこに欠損があったかを記録する列を追加する。

```
len(inputs.dropna()), len(inputs)
```

```
(4, 10)
```

不完全な行を除外すると 6 件（全 10 件中、データセットの 60%）の住宅が破棄されてしまうため、代わりに補完を用いる。`inputs.fillna(inputs.mean(numeric_only=True))` を呼び出すと、数値列の各 NaN がその列の観測値の平均で置き換えられ（外れ値によって平均が歪む場合は中央値の方が望ましい）、テキスト列である RoofType は次のカテゴリ特徴量のステップへとそのまま残される。

```
inputs = inputs.fillna(inputs.mean(numeric_only=True))
inputs
```

```
   NumRooms RoofType  Area
0  3.000000    Slate  1500
1  3.285714     Tile  2100
2  2.000000      NaN   850
3  4.000000    Slate  1940
4  3.285714      NaN  1200
5  3.000000     Tile  1650
6  5.000000    Slate  2600
7  2.000000      NaN   900
8  3.285714     Tile  1750
9  4.000000      NaN  2050
```

行 1、4、8 には、観測された 7 件の部屋数の平均である $23/7 \approx 3.285714$ が格納された。現実の住宅に 3.29 部屋というものは存在しないが、補完された平均は中立的な統計的プレースホルダとして機能する。

1.2.2.2 カテゴリ特徴量

欠損要素を独立した 1 つの状態として含めると、RoofType は順序のない 3 つのカテゴリ（Slate、Tile、または NaN）に分類される。Slate $= 0$、Tile $= 1$、missing $= 2$ のように単一の整数コードを割り当てると、特徴量のあらゆる線形結合において Tile が Slate と未記録の屋根の中間として扱われることになり、人工的な距離と順序関係を押し付けてしまう。

ワンホットエンコーディング、すなわち**ダミー変数エンコーディング**（dummy variable encoding）は、k 個のカテゴリを持つ列を k 本の二値（0/1）インジケータ列に置き換え、該当するカテゴリの要素を 1.0、残りの $k-1$ 個の要素を 0.0 に設定することでこの落とし穴を回避する。pandas では、`pd.get_dummies` が数値列をそのまま残しつつすべての文字列列・カテゴリ列をエンコードする。`dummy_na=True` を指定すると明示的な欠損性インジケータである RoofType_nan が追加され、`dtype=float` を指定するとブール値ではなく 0.0/1.0 の浮動小数点数が格納される。

```
inputs = pd.get_dummies(inputs, dummy_na=True, dtype=float)
inputs
```

```
   NumRooms  Area  RoofType_Slate  RoofType_Tile  RoofType_nan
0  3.000000  1500             1.0            0.0           0.0
1  3.285714  2100             0.0            1.0           0.0
2  2.000000   850             0.0            0.0           1.0
3  4.000000  1940             1.0            0.0           0.0
4  3.285714  1200             0.0            0.0           1.0
```

```
5  3.000000  1650            0.0           1.0           0.0
6  5.000000  2600            1.0           0.0           0.0
7  2.000000   900            0.0           0.0           1.0
8  3.285714  1750            0.0           1.0           0.0
9  4.000000  2050            0.0           0.0           1.0
```

どの行でも、RoofType_Slate、RoofType_Tile、RoofType_nan の和は 1.0 になる（そのため切片バイアスと併用すると 1 列が冗長になる。同様に、欠損値のない 2 状態の列には $0/1$ インジケータが 1 列あれば足りる）。ワンホットエンコーディングはカテゴリ数が少ない場合に適しているが、数万ものカテゴリを持つ列（郵便番号、ユーザー ID、語彙の単語など）は、代わりにコンパクトで学習可能な**埋め込み**（embedding）へと写像する（8.4 節）。

1.2.3 スケーリングと変換

1.2.3.1 標準化

これで inputs のすべての列が数値になったが、NumRooms と Area の測定スケールは大きく食い違っている。生の単位のままでは、2 部屋の差は数値上 3 平方フィートよりも小さい。床面積をスケーリングしないままにしておくと、ユークリッド距離（Euclidean distance）や重みのペナルティが床面積に支配されてしまう。さらに 2.3 節および第 9 章で見るように、何桁も異なるスケール差は損失ランドスケープ（loss surface）を細長い谷へと引き伸ばし、勾配降下法（gradient descent）の収束を遅らせる。

標準化（standardization）、すなわち **z スコア正規化**（z-score normalization）は、連続値の特徴量を共通のスケールに揃えるため、各要素 x を次の式へと変換する。

$$\frac{x - \mu}{\sigma}, \tag{1.1}$$

ここで、μ はその列の標本平均、$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$ は標本標準偏差である（Bessel の分母 $n-1$ は 1.6 節で導出する）。標準化されたすべての列は平均が 0、標準偏差が 1 となり、床面積が平方フィートで記録されていようと平方メートルで記録されていようと、観測値は無次元の標準偏差を単位として測られる。ワンホットインジケータはすでに $\{0, 1\}$ の値をとるため、ここでは `['NumRooms', 'Area']` のみを標準化する。

```
numerical = ['NumRooms', 'Area']
mu, sigma = inputs[numerical].mean(), inputs[numerical].std()
inputs[numerical] = (inputs[numerical] - mu) / sigma
inputs
```

```
   NumRooms      Area  RoofType_Slate  RoofType_Tile  RoofType_nan
0 -0.314485 -0.276769             1.0            0.0           0.0
1  0.000000  0.801551             0.0            1.0           0.0
2 -1.415185 -1.444949             0.0            0.0           1.0
3  0.786214  0.513999             1.0            0.0           0.0
4  0.000000 -0.815929             0.0            0.0           1.0
5 -0.314485 -0.007189             0.0            1.0           0.0
6  1.886913  1.700151             1.0            0.0           0.0
7 -1.415185 -1.355089             0.0            0.0           1.0
8  0.000000  0.172531             0.0            1.0           0.0
9  0.786214  0.711691             0.0            0.0           1.0
```

行 6（5 部屋、2,600 平方フィート）は $(1.89, 1.70)$ となり、どちらの次元でも平均より標準偏差の約 1.7 から 1.9 倍だけ上に位置している。一方、平均で補完された行 1、4、8 の 3 つの部屋数は 0.0 になっている。ここで、Figure 1.9 に示す副作用に注目しよう。補完された 3 行を μ に置くと、σ の分子にはゼロしか加わらない一方で n が 7 から 10 に増えるため、部屋数の標準偏差が 1.11 から 0.91 へと縮小し、観測値の z スコアが（1.54 から 1.89 へと、行 6 において）膨らむのである。

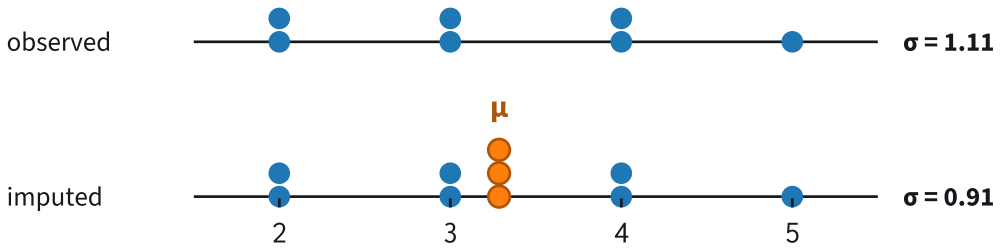


Figure 1.9: 平均補完の前後の部屋数。補完された 3 つの値（オレンジ）はすべて、観測された 7 つの値の平均 μ 上に重なるため、この列は元のデータよりもばらつきが小さくなり、標準偏差は 1.11 から 0.91 へと低下する。

データセットを**訓練セット**（training set）と**テストセット**に分割する際には、方法論上の 1 つの原則が何より重要となる。すなわち、**補完に用いる平均や中央値、カテゴリの語彙、標準化の (μ, σ) を含むすべての前処理統計量は、訓練セットのみから算出しなければならず**、それを変更せずにそのままテストセットへ適用する（Figure 1.10）。分割前のデータセット全体で統計量を算出すると、評価用に確保したテストデータから訓練へと情報が漏洩し（**データリーケージ**）、テストスコアが不当に高く出てしまう。4.7 節では、本格的な不動産ベンチマークに対してこの原則を徹底する（演習問題 4 も参照）。

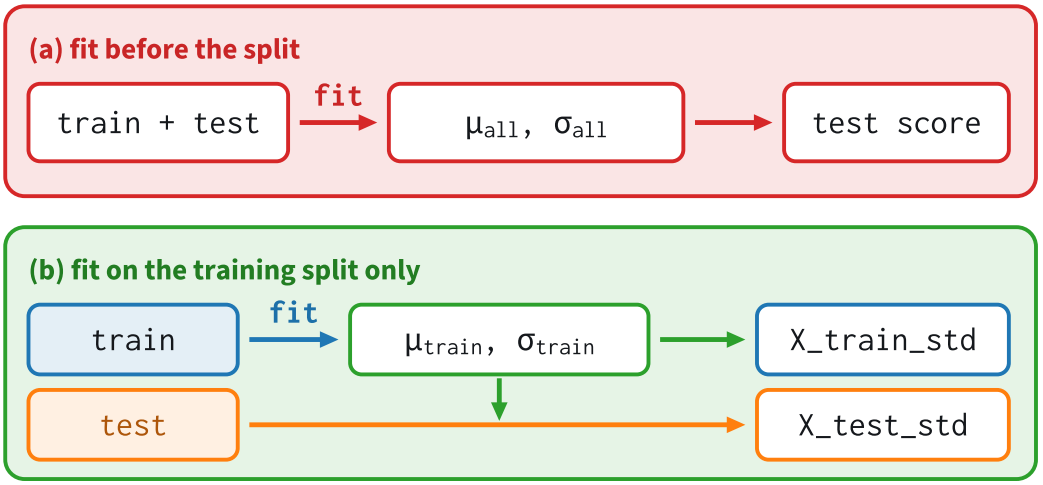


Figure 1.10: 統計量の算出とデータ分割の順序。(a) すべての行で算出した統計量はすでにテストセットを見ているため、テストスコアが楽観的になりすぎる。(b) 訓練データ分割のみで算出した統計量を変更せずに両方の分割へ適用すれば、テストセットは未知のまま保たれる。

1.2.3.2 テンソルへの変換

最後に、`.to_numpy(dtype='float32')` を使って整形済みの列を連続した 32 ビット浮動小数点の NumPy 配列として取り出し（pandas のデフォルトである `float64` からダウンキャストする）、フレームワークのテンソルコンストラクタに渡せる状態にする（1.1 節）。

```
X = jnp.array(inputs.to_numpy(dtype='float32'))
y = jnp.array(targets.to_numpy(dtype='float32'))
X.shape, X.dtype, y.shape, y.dtype
```

```
((10, 5), dtype('float32'), (10,), dtype('float32'))
```

特徴行列 `X` の形状は `(10, 5)`（住宅 10 件に対し、標準化された数値特徴量 2 列と屋根材インジケータ 3 列）、ターゲットベクトル `y` の形状は `(10,)` であり、いずれもデータ型は `float32` である。実運用のパイプラインでは、この段階に至る前に複数のリレーショナルテーブルを結合し、破損レコードやセンチネルコード（不明な座標を表す `-9999` など）を検査することが多い。こうして計画行列（design matrix）$\mathbf{X}$ にまとめられたデータは、次節で扱う線形代数の演算へと引き渡される。

1.2.4 まとめ

実世界の表形式データセットには、ほぼ例外なく欠損値やカテゴリ型のテキスト項目、大きく異なるスケールで測定された数値列が含まれているが、ニューラルネットワークには整った `float32` テンソルが必要である。pandas を用いれば、`read_csv` で CSV ファイルを DataFrame に読み込み、`info()` と `describe()` で列の型や欠損値の数、値の分布を確認し、最初の段階で入力特徴量からターゲット列を分離できる。欠損値は、削除、補完（数値の欠損を訓練セットの平均や中央値で埋めるなど）、または欠損性インジケータ列によって対処できる。順序のないカテゴリ列は、架空の数値的順序を作り出さないようワンホットエンコーディング（`get_dummies`）によって二値（$0/1$）のインジケータ列へと変換し、連続値の数値列は訓練データ分割のみで計算した統計量を用いて平均 0 と分散 1 へと標準化する（$x \mapsto (x - \mu)/\sigma$）。最後に、`.to_numpy(dtype='float32')` によって整形済みの DataFrame を深層学習フレームワークのテンソルへと直接橋渡しする。

1.2.5 演習問題

1. [code] **列名による選択と位置による選択の比較**。本節では、`data` を `data.iloc[:, :3]` と `data.iloc[:, 3]` を用いて整数の位置により特徴量とターゲットに分割した。ここで、`house_tiny.csv` の更新版において `YearBuilt` という新しい列が `Area` と `Price` の間に挿入されたとする。
    1. 更新されたファイルに対して位置による分割 `data.iloc[:, :3]` と `data.iloc[:, 3]` を実行すると、`inputs` と `targets` にはそれぞれ何が入るか。本節の後続のコード行で例外が発生するか、それともパイプラインは気づかないうちに誤ったターゲットを予測するモデルを訓練してしまうか。
    2. 列の順序や新たに追加された特徴量にかかわらず、列名を使って（たとえば `data['Price']` と `data.drop(...)` により）`targets` が常に `Price` を抽出し、`inputs` が他のすべての列を保持するよう、特徴量とターゲットの分割を書き換えよ。詳細については、データのインデックス参照と選択に関する pandas ユーザーガイドを参照せよ。
2. [code] **補完戦略の比較**。`NumRooms` の欠損要素を次の 3 つの方式で補完せよ。(i) 平均による補完、(ii) 中央値による補完、(iii) 平均による補完に二値の `NumRooms_isna` インジケータ列を追加する方式。

1. 平均による補完と中央値による補完のそれぞれについて、補完後の NumRooms 列の標本平均と標本標準偏差を計算し、元の 7 個の観測値の平均および標準偏差と比較したうえで、なぜ標準偏差がその方向に変化するのかを説明せよ。
2. より大規模な不動産データセットにおいて、NumRooms の未報告が主に高級邸宅で発生しているとする。平均による補完のみを用いた場合、モデルにはそれらの邸宅の部屋数がいくつとして見えるか。また、線形モデル（入力列の重み付き和にバイアスを加えて価格を予測するモデル）が、追加された NumRooms_isna インジケータ列をどのように利用してその系統的な過小評価を補正できるか説明せよ。
3. 3 つの対処法（削除、平均による補完のみ、インジケータを伴う補完）のそれぞれが、値の欠損を引き起こすメカニズムについてどのような統計的仮定を置いているかを明確に述べよ。

Kaggle Learn の Intermediate Machine Learning 演習問題 2 を改変。

3. **高カーディナリティのカテゴリ特徴量。**ワンホットエンコーディングは、k 個の異なるカテゴリを持つカテゴリ列を k 本の二値列（dummy_na=True の場合は $k+1$ 本）に展開する。

   1. $n = 10^6$ 件の住宅からなる表に、欠損値のない $k = 10^4$ 通りの郵便番号を持つ ZipCode 列が含まれているとする。ワンホットエンコーディング後の行列を 64 ビット浮動小数点（float64）の密行列として保存した場合に何 GB を占めるかを計算し、非ゼロ要素の割合を答えよ。
   2. さらにこの表に、すべての行に対して一意な識別子文字列を持つ ListingID 列も含まれているとする。ListingID を扱う 3 つの方法、すなわち (i) ワンホットエンコーディング、(ii) 各 ID を学習可能な埋め込みベクトルに写像する方法、(iii) その列を完全に除外する方法を比較せよ。それぞれの方法のもとで、モデルは ListingID から、ListingID が訓練セットに一度も現れなかった新規登録の住宅へ汎化するような何を学習できるか。また、どの方法を選ぶべきか。

   Kaggle Learn の Intermediate Machine Learning 演習問題 3 を改変。

4. [code] **データリーケージの定量化。**本文では、NumRooms の欠損要素の補完と NumRooms および Area の標準化に、全 10 行にわたって計算した標本平均と標準偏差を用いた。ここでは data を、最初の 8 行からなる訓練セット（data.iloc[:8]）と最後の 2 行からなるテストセット（data.iloc[8:]）に分割せよ。

   1. 補完の平均および標準化パラメータ（μ と σ）を**訓練用の 8 行のみから**計算するように前処理パイプラインを再実行し、その訓練パラメータを変更せずに用いて訓練データ分割とテストデータ分割の両方を補完・標準化せよ。得られた標準化後のテスト行を、μ と σ を全 10 行から計算したときの値と比較せよ。
   2. 表全体を使う手順のもとでは、評価用に確保したテスト住宅に関するどのような情報が訓練特徴量へ漏洩したのかを正確に説明せよ。
   3. あるカテゴリ特徴量または数値特徴量が、たまたま訓練行のすべてで一定の値をとり（したがって訓練データの標準偏差が $\sigma = 0$ となり）、テスト行では変化するとする。訓練データのみに基づく標準化の式 $(x-\mu)/\sigma$ は、訓練データ分割とテストデータ分割のそれぞれで何を生成するか。また、堅牢な前処理パイプラインは分散ゼロの特徴量をどのように扱うべきか。

1.3 線形代数

画像のミニバッチを 255 で割る操作は各ピクセルを個別にスケーリングするだけだが、予測モデルでは数百から数千もの入力特徴量を組み合わせて判断を下さなければならない。**線形代数**（linear algebra）はそのような重み付き結合を記述する数学の言語であり、ミニバッチ全体に対

する 1 つの層の順伝播（forward pass）を簡潔な 1 本の式で表し、出力の形状や浮動小数点演算のコストを実行前に読み取ることを可能にする。

28×28 のグレースケール衣類画像（第 3 章）を 10 個のカテゴリに分類する線形画像分類器（image classifier）を考えよう。各写真を平坦化すると $28 \times 28 = 784$ ピクセルになる。平坦化した 256 枚の画像のミニバッチをデータ行列 $\mathbf{X} \in \mathbb{R}^{256\times784}$ の行として積み重ね、784 個ずつの重みを 10 個のカテゴリそれぞれについて $\mathbf{W} \in \mathbb{R}^{784\times10}$ の列として並べ、10 個のカテゴリごとのオフセットを $\mathbf{b} \in \mathbb{R}^{10}$ にまとめると（パラメータ数は合計 7,850 個）、ミニバッチ全体の $256 \times 10 = 2{,}560$ 個のカテゴリスコアは次の式で与えられる。

$$\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}, \tag{1.2}$$

ここで、$\mathbf{b}$ はブロードキャストによってすべての行に加算される（Figure 1.11）。

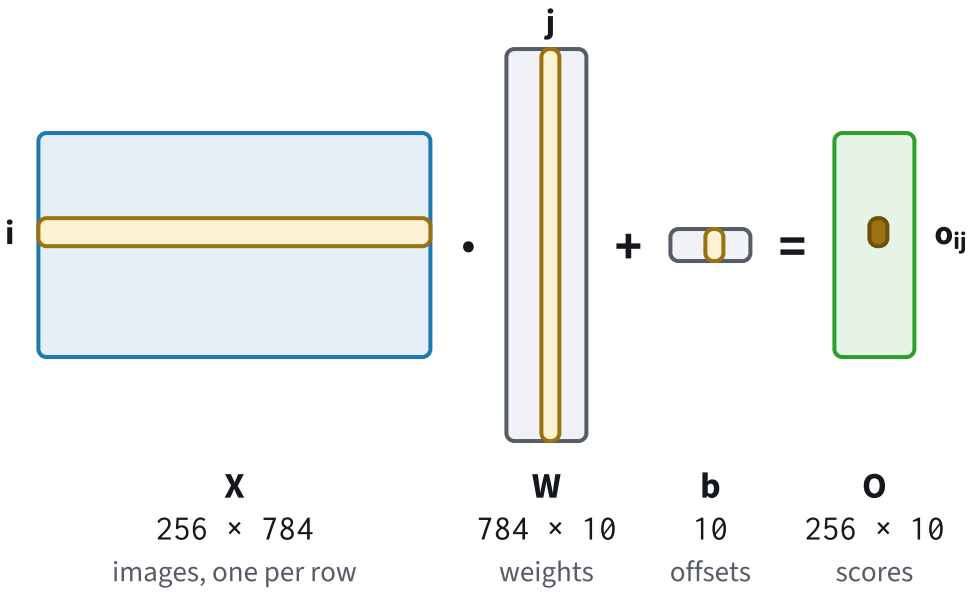


Figure 1.11: 256 枚の画像と 10 個のカテゴリからなるミニバッチに対する線形分類器のスコア。$\mathbf{X}$ の第 i 行は画像 i を保持し、$\mathbf{W}$ の第 j 列はカテゴリ j の重みを保持し、この 2 つからスコア o_{ij} が定まる。

本節では、スカラー、ベクトル、行列、テンソルから出発し、リダクション、ドット積、行列積を経て $\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}$ へと積み上げ、最後に**ノルム**と**固有値**（eigenvalue）を扱う（記法の規約は記法 にまとめられている）。

```
import random
import jax
from jax import numpy as jnp
```

1.3.1 スカラー、ベクトル、行列、テンソル

0^{th}、1^{st}、2^{nd}、4^{th} 階の配列の作成と形状変更はすでに 1.1 節で扱った。ここでは、各階数（コードでは `ndim`）を標準的な数学上の名称および記法と対応づける。

1.3.1.1 スカラー

単一の数値は**スカラー**と呼ばれ、通常の小文字イタリック体（x, y, z）で表される。$x \in \mathbb{R}$ と書けば、x が実数であることを表す。コードでは、スカラーは 0^{th} 階のテンソルであり、空の形状 `()` を持つ。

```
x = jnp.array(3.0)
x, x.shape
```

```
(Array(3., dtype=float32, weak_type=True), ())
```

1.3.1.2 ベクトル

ベクトルは固定長のスカラーの 1 次元配列（スカラーはその**要素**または**成分**）であり、太字の小文字（$\mathbf{x}, \mathbf{y}, \mathbf{w}$）で表される。$\mathbf{x} \in \mathbb{R}^n$ は n 次元の実ベクトルを表し、特に断らない限り**列ベクトル**（column vector）として扱う。

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \tag{1.3}$$

文中にインラインで書く場合は $\mathbf{x} = (x_1, \ldots, x_n)$ とも表記する。コードでは、ベクトルは 1^{st} 階のテンソルである。

```
x = jnp.arange(3, dtype=jnp.float32)
x
```

```
Array([0., 1., 2.], dtype=float32)
```

数学の記法では 1 始まりのインデックス（$x_1, \ldots, x_n$）を使うのに対し、Python では 0 始まりのインデックスを使うため、3 番目の成分 x_3 には `x[2]` でアクセスする。

```
x[2]
```

```
Array(2., dtype=float32)
```

ベクトルの**次元数**（dimensionality）n（成分の個数）は、Python の組み込み関数 `len(x)` によって、あるいは要素数 1 のタプルである形状 `x.shape` の唯一の要素として得られる。

```
len(x), x.shape
```

```
(3, (3,))
```

「次元」という言葉が曖昧に使われることによる混乱を防ぐため、本書ではテンソルの軸の数（`ndim`）には**階数**（order）、ベクトルの成分の数には**次元数**という用語を充てる。

1.3.1.3 行列

行列は 2^{nd} 階のテンソルであり、太字の大文字（$\mathbf{A}, \mathbf{X}, \mathbf{W}$）で表される。$\mathbf{A} \in \mathbb{R}^{m \times n}$ と書けば m 行 n 列であることを示し（$m = n$ のとき $\mathbf{A}$ は**正方行列**（square matrix）である）、成分 a_{ij} が第 i 行・第 j 列に並ぶ。

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \tag{1.4}$$

表形式データや平坦化したミニバッチの行列 $\mathbf{X} \in \mathbb{R}^{n \times d}$ では、1 行につき 1 つのデータ例、1 列につき 1 つの特徴量を格納する。以下では、2×3 行列 `A` を構築する。

```
A = jnp.arange(6, dtype=jnp.float32).reshape(2, 3)
A
```

```
Array([[0., 1., 2.],
       [3., 4., 5.]], dtype=float32)
```

行列の行と列を入れ替えると、その**転置**（transpose）$\mathbf{A}^\top \in \mathbb{R}^{n \times m}$（成分は $[\mathbf{A}^\top]_{ij} = a_{ji}$）が得られ、コードでは `.T` でアクセスする。

```
A.T
```

```
Array([[0., 3.],
       [1., 4.],
       [2., 5.]], dtype=float32)
```

正方行列は、$\mathbf{S} = \mathbf{S}^\top$（$s_{ij} = s_{ji}$ がすべての i, j で成り立つ）のとき**対称行列**（symmetric matrix）である。

```
S = jnp.array([[2.0, 1.0], [1.0, 2.0]])
(S == S.T).all()
```

```
Array(True, dtype=bool)
```

1.3.1.4 高階テンソル

3 つ以上の軸を持つテンソルは、サンセリフ体の大文字（`X`, `Y`, `Z`）で表される。以下では、合成した 4 次元の画像ミニバッチ `images`（形状は $(256, 1, 28, 28)$、要素は $[0, 1]$ の範囲）を生成し、各画像を平坦化して 2 次元行列 $\mathbf{X} \in \mathbb{R}^{256 \times 784}$ を作る。

```
key = jax.random.key(random.getrandbits(32))  # A new seed on each run
key, subkey = jax.random.split(key)
images = jax.random.uniform(subkey, (256, 1, 28, 28))
X = images.reshape(256, -1)  # One row of 784 pixels per image
images.shape, X.shape
```

```
((256, 1, 28, 28), (256, 784))
```

1.3.2 要素ごとの演算とリダクション

1.3.2.1 要素ごとの演算とアダマール積

同じ形状を持つ 2 つの行列 $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ の要素ごとの積は**アダマール積**（Hadamard product）$\mathbf{A} \odot \mathbf{B}$ と呼ばれ、その成分は $[\mathbf{A} \odot \mathbf{B}]_{ij} = a_{ij} b_{ij}$ であり、コードでは `*` で計算する。

```
A * A
```

```
Array([[ 0.,  1.,  4.],
       [ 9., 16., 25.]], dtype=float32)
```

1.3.2.2 軸に沿ったリダクション

`axis=0` を `.sum()` に渡すと行方向にわたって和をとり列和のベクトル（$\sum_{i=1}^{m} a_{ij}$）を生成するのに対し、`axis=1` を渡すと列方向にわたって和をとり行和のベクトル（$\sum_{j=1}^{n} a_{ij}$）を生成する。`axis` を省略すると全要素の和をとる（Figure 1.12）。

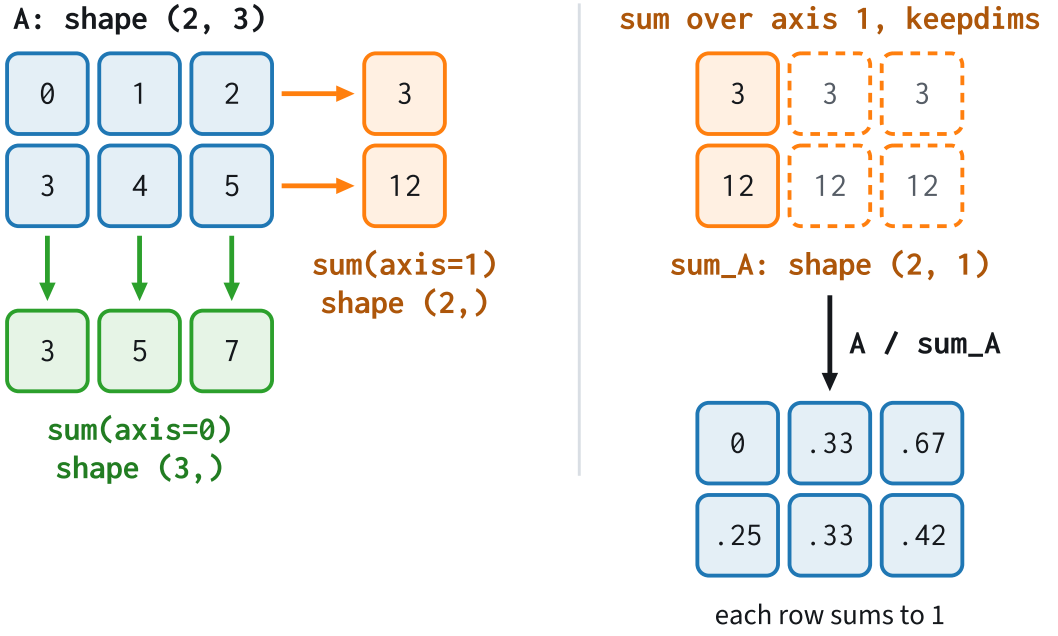


Figure 1.12: 2×3 行列のリダクション。軸 0 にわたって和をとると列ごとに 1 つの和が残り、軸 1 にわたって和をとると行ごとに 1 つの和が残る（リダクションされた軸は形状から消える）。`keepdims=True` を指定すると行和は形状 $(2, 1)$ を保つため、ブロードキャストによって 3 つの列へと引き伸ばされ、除算 `A / sum_A` によって各行の和が 1 になるようスケーリングされる。

```
A.sum(), A.sum(axis=0), A.sum(axis=1)
```

```
(Array(15., dtype=float32),
 Array([3., 5., 7.], dtype=float32),
 Array([ 3., 12.], dtype=float32))
```

1.3.2.3 ブロードキャストのためのリダクション軸の保持

`A`（形状 `(2, 3)`）の各行をその行和で正規化しようとしても、`A.sum(axis=1)`（形状 `(2,)`）は `(2, 3)` に対してブロードキャストできない。末尾の次元 2 と 3 が衝突するためである。`keepdims=True` を渡すとリダクションされた軸が長さ 1 のまま保持され、`sum_A` の形状は `(2, 1)` になる。

```
sum_A = A.sum(axis=1, keepdims=True)
sum_A, sum_A.shape
```

```
(Array([[ 3.],
        [12.]], dtype=float32),
 (2, 1))
```

`sum_A` の形状が `(2, 1)` であるため、`A / sum_A` で割ると各行の和が 3 つの列すべてにブロードキャストされ、得られる 2×3 行列のどの行も和が 1 になる。

```
A / sum_A
```

```
Array([[0.        , 0.33333334, 0.6666667 ],
       [0.25      , 0.33333334, 0.4166667 ]], dtype=float32)
```

256×784 のミニバッチ $\mathbf{X}$ において、ピクセルごとの列平均（`axis=0`、形状 `(784,)`）を引く操作はそのままブロードキャストされるが、画像ごとの行平均（`axis=1`）を引くには `keepdims=True`（形状 `(256, 1)`）が必要である。

```
col_means = X.mean(axis=0)
row_means = X.mean(axis=1, keepdims=True)
col_means.shape, row_means.shape, (X - row_means).shape
```

```
((784,), (256, 1), (256, 784))
```

注意すべき点として、もし $\mathbf{X}$ がたまたま正方行列（784×784）であった場合、`keepdims=True` を `X - X.mean(axis=1)` で省略してもエラーは発生せず、画像 j の平均がすべての画像の第 j 列から暗黙のうちに引かれてしまう（演習問題 2）。

1.3.3 積

1.3.3.1 ドット積

2 つのベクトル $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ が与えられたとき、それらの**ドット積**、すなわち**内積**（inner product）$\mathbf{x}^\top \mathbf{y}$（$\langle \mathbf{x}, \mathbf{y} \rangle$ とも書く）は、対応する成分同士の積の総和である。

$$\mathbf{x}^\top \mathbf{y} = \sum_{i=1}^{d} x_i y_i. \tag{1.5}$$

コードでは、2 つの 1 次元ベクトル `x` と `y` のドット積をフレームワークの `dot` 関数で直接計算する。

```
y = jnp.ones(3)
x, y, jnp.dot(x, y)
```

```
(Array([0., 1., 2.], dtype=float32),
 Array([1., 1., 1.], dtype=float32),
 Array(3., dtype=float32))
```

同じ計算は、`x * y` で要素ごとに掛けてから得られた積を `sum` で足し合わせることでも実行できる（$\sum_i x_i = 0 + 1 + 2 = 3$ が $\mathbf{y} = \mathbf{1}$ のときに得られる）。

```
jnp.sum(x * y)
```

```
Array(3., dtype=float32)
```

1.3.3.2 行列・ベクトル積

$\mathbf{A} \in \mathbb{R}^{m \times n}$ を m 個の行ベクトル $\mathbf{a}_1^\top, \ldots, \mathbf{a}_m^\top \in \mathbb{R}^n$ に分割すると、**行列・ベクトル積**（matrix–vector product）$\mathbf{Ax}$（ただし $\mathbf{x} \in \mathbb{R}^n$）は、各行とのドット積を並べた m 次元ベクトルになる（Figure 1.13）。

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{a}_1^\top \\ \vdots \\ \mathbf{a}_m^\top \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1^\top \mathbf{x} \\ \vdots \\ \mathbf{a}_m^\top \mathbf{x} \end{bmatrix} \in \mathbb{R}^m.$$

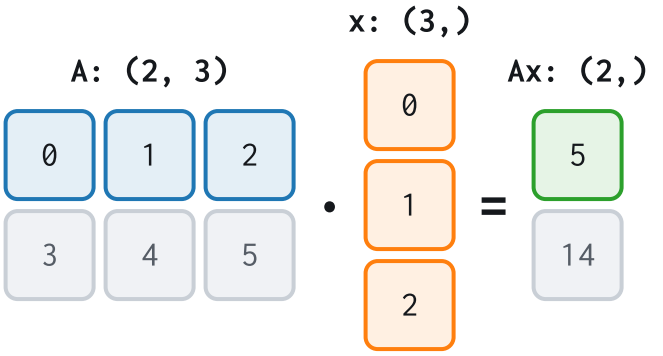


Figure 1.13: 行列・ベクトル積 $\mathbf{Ax}$ は $\mathbf{A}$ の行ごとに 1 つのドット積をとるため、2×3 行列は長さ 3 のベクトルを長さ 2 のベクトルに写す。1 行目からは $0 \cdot 0 + 1 \cdot 1 + 2 \cdot 2 = 5$ が得られる。

Python では、中置演算子 `@` によって行列・ベクトル積と行列・行列積の両方を計算できる。

```
A.shape, x.shape, A @ x
```

```
((2, 3), (3,), Array([ 5., 14.], dtype=float32))
```

列ごとに見れば、$\mathbf{Ax} = \sum_{j=1}^{n} x_j (\mathbf{Ae}_j)$ は $\mathbf{A}$ の列ベクトルの線形結合であり、ここで j^{th} 列 $\mathbf{Ae}_j$ は j^{th} 標準基底ベクトル（basis vector）$\mathbf{e}_j$ の像である。$\mathbb{R}^2$ において、2×2 行列は $\mathbf{e}_1, \mathbf{e}_2$ が張る単位正方形を、その列ベクトルが張る平行四辺形へと写す（Figure 1.14）。

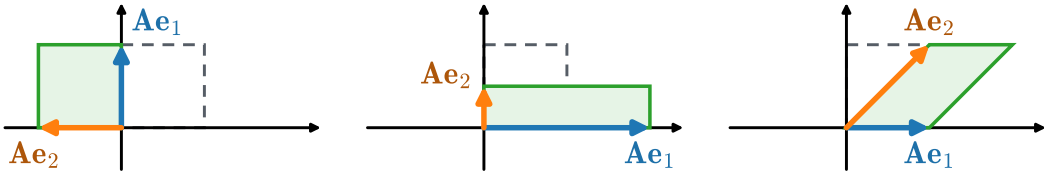


Figure 1.14: 2×2 行列は単位正方形（破線）を、その列ベクトル $\mathbf{Ae}_1$ と $\mathbf{Ae}_2$ が張る平行四辺形に写す。左から順に、$90°$ の回転、2 倍（$\mathbf{e}_1$ 方向）と $1/2$ 倍（$\mathbf{e}_2$ 方向）のスケーリング、せん断。

1.3.3.3 行列・行列積

$\mathbf{A} \in \mathbb{R}^{m \times k}$ と $\mathbf{B} \in \mathbb{R}^{k \times n}$ が与えられたとき、それらの行列積 $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times n}$ の (i, j) 成分は、$\mathbf{A}$ の第 i 行と $\mathbf{B}$ の第 j 列のドット積に等しい。

$$c_{ij} = \sum_{l=1}^{k} a_{il}\, b_{lj}. \tag{1.6}$$

```
B = jnp.ones((3, 4))
A @ B
```

```
Array([[ 3.,  3.,  3.,  3.],
       [12., 12., 12., 12.]], dtype=float32)
```

これで、10 クラスの線形分類器 $\mathbf{O} = \mathbf{XW} + \mathbf{b}$ を 256×784 のミニバッチ `X` に対して評価し、`argmax` を軸 1 に沿って適用することで各画像のスコアが最も高いカテゴリを選べる。

```
key, subkey = jax.random.split(key)
W = 0.01 * jax.random.normal(subkey, (784, 10))  # Column j: category j
b = jnp.zeros(10)                                # Entry j: category j
O = X @ W + b                                    # Row i: image i
O.shape, O.argmax(axis=1).shape
```

```
((256, 10), (256,))
```

$m \times k$ 行列と $k \times n$ 行列の積では、mn 個の長さ k のドット積を計算するため、コストは $\approx 2mkn$ FLOPs となる（$2 \times 256 \times 784 \times 10 \approx 4 \times 10^6$ FLOPs が `X @ W` にかかり、また $2n^3$ FLOPs がメモリ上のわずか $3n^2$ 個の要素に対して 2 つの $n \times n$ 行列の積で実行される。第 13 章を参照）。線形代数では 2 つの特別な正方行列が繰り返し登場する。1 つは**単位行列**（identity matrix）$\mathbf{I} \in \mathbb{R}^{n \times n}$（主対角成分が 1 でそれ以外が 0）であり、任意のベクトルを変化させない（$\mathbf{Ix} = \mathbf{x}$）。もう 1 つは**逆行列**（inverse）$\mathbf{A}^{-1}$ であり、可逆行列（invertible matrix）$\mathbf{A}$ に対して $\mathbf{A}^{-1}\mathbf{A} = \mathbf{AA}^{-1} = \mathbf{I}$ によって定義される（33.2.2.3 節）。

後の章で扱うモデルでは、ミニバッチ内のデータ例ごとに 1 つずつ用意された行列のスタック全体を一度に掛け合わせる。`@` 演算子はこれをループなしで実行する。すなわち、3 つ以上の軸を持つテンソルに対しては最後の 2 つの軸で行列積を計算し、先頭側の軸にわたってブロードキャストする（1.1.3.2 節）。`@` では対応できない場合でも、`einsum` を使えば Equation 1.6 の和を添字ごとに書き下せる。文字列 `'bil,lj->bij'` は各因子と結果の軸に名前を付けたものであり、結果に含まれない添字（ここでは `l`）について和がとられる。`A` の定数倍を 3 つ積み重ね、それらすべてに `B` を掛けてみよう。

```
As = jnp.stack([A, 2 * A, 3 * A])  # Three 2 x 3 matrices, shape (3, 2, 3)
C = As @ B                         # B is broadcast across the batch
C.shape, (jnp.einsum('bil,lj->bij', As, B) == C).all()
```

```
((3, 2, 4), Array(True, dtype=bool))
```

結果にはスタック内の行列ごとに 1 つずつ、計 3 個の 2×4 の積が格納されており、`einsum` の結果はすべての成分で `@` と一致する。

1.3.4 ノルム

ノルムはベクトルや行列の大きさを測る量であり、機械学習では予測誤差 $\hat{\mathbf{y}} - \mathbf{y}$ を測る**損失関数**（loss function）としても、重みの大きさを制約する**正則化ペナルティ**（regularization penalty）としても用いられる。

1.3.4.1 ユークリッド長と角度

ℓ_2 **ノルム**（または**ユークリッドノルム**）は、$\mathbf{x} \in \mathbb{R}^n$ に対して次の式で定義される。

$$\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_{i=1}^{n} x_i^2}, \tag{1.7}$$

これはしばしば $\|\mathbf{x}\|$ と略記される。平面上の 2 つのベクトル $\mathbf{x}, \mathbf{y}$ が角度 θ をなすとき、余弦定理 $\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\|\mathbf{x}\|\|\mathbf{y}\|\cos\theta$ と双線形展開 $(\mathbf{x}-\mathbf{y})^\top(\mathbf{x}-\mathbf{y}) = \|\mathbf{x}\|^2 - 2\mathbf{x}^\top\mathbf{y} + \|\mathbf{y}\|^2$ を等置すると、次式が得られる（Figure 1.15）。

$$\mathbf{x}^\top \mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\|\cos\theta. \tag{1.8}$$

任意の次元 d において、Equation 1.8 を逆に読むことで、非ゼロベクトル間の**コサイン類似度**（cosine similarity）を次のように定義する。

$$\cos\theta = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}, \tag{1.9}$$

この量は向きのみに依存し、$+1$（同じ向き）から 0（直交）を経て -1（反対向き）までの値をとる（Figure 1.16）。

コーシー・シュワルツの不等式（Cauchy–Schwarz inequality）により $|\mathbf{x}^\top\mathbf{y}| \le \|\mathbf{x}\|\|\mathbf{y}\|$ が保証される（したがって $\cos\theta \in [-1, 1]$）。実際、単位ベクトル $\mathbf{u} = \mathbf{x}/\|\mathbf{x}\|$ と $\mathbf{v} = \mathbf{y}/\|\mathbf{y}\|$ に対して $0 \le \|\mathbf{u} \mp \mathbf{v}\|^2 = 2 \mp 2\mathbf{u}^\top\mathbf{v}$ を展開すれば $|\mathbf{u}^\top\mathbf{v}| \le 1$ が得られる。

```
def cosine(x, y):
    return jnp.dot(x, y) / (jnp.linalg.norm(x) * jnp.linalg.norm(y))

x, y = jnp.array([1.0, 1, 1, 1]), jnp.array([1.0, 1, 1, -1])
cosine(x, y), cosine(2 * x, y), jnp.arccos(cosine(x, y))
```

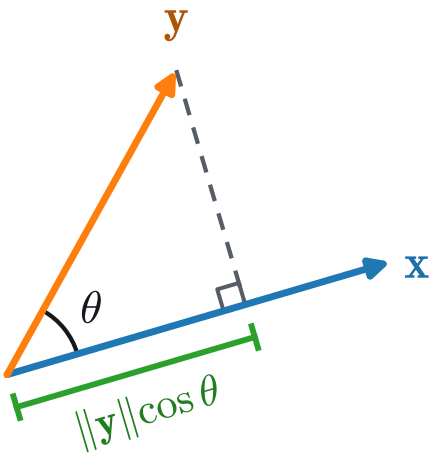


Figure 1.15: ドット積 $\mathbf{x}^\top\mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\|\cos\theta$ は、$\mathbf{x}$ の長さと符号付き長さ $\|\mathbf{y}\|\cos\theta$（$\mathbf{y}$ を $\mathbf{x}$ を通る直線へ射影した長さ）の積である。

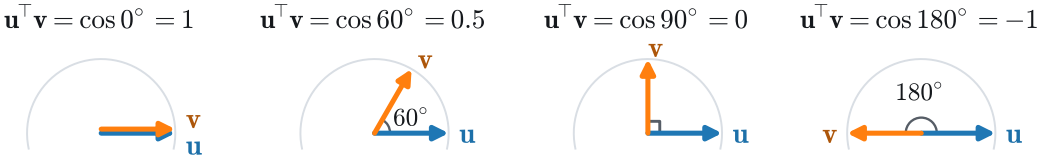


Figure 1.16: 角度 θ をなす 2 つの単位ベクトル。それらのドット積は $\cos\theta$ に等しく、1（同じ向き）から 0（垂直）を経て -1（反対向き）までの値をとる。

```
(Array(0.5, dtype=float32),
 Array(0.5, dtype=float32),
 Array(1.0471976, dtype=float32))
```

1.3.4.2 ℓ_1 ノルム、ℓ_p ノルム、フロベニウスノルム

ℓ_1 **ノルム**（または**マンハッタン距離**、Manhattan distance）は、各成分の絶対値の総和をとる（Figure 1.17）。

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|. \tag{1.10}$$

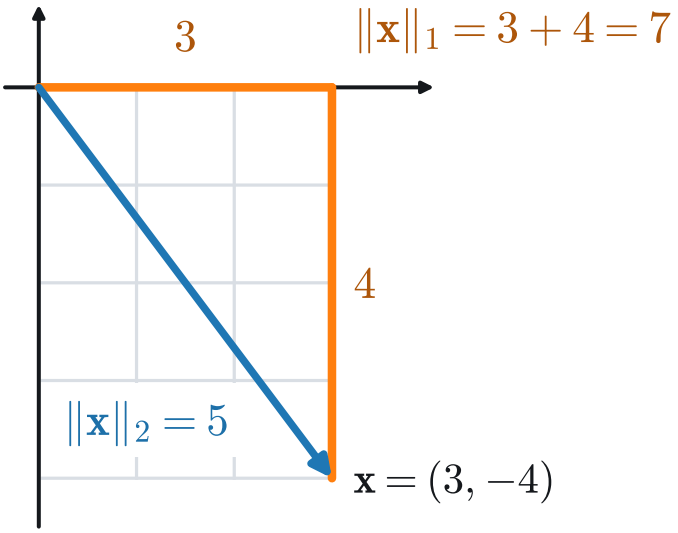


Figure 1.17: $\mathbf{x}$ の ℓ_2 ノルムは $\mathbf{x}$ への直線経路の長さであり、ℓ_1 ノルムは格子に沿った経路の長さである。

```
x = jnp.array([3.0, -4.0])
jnp.linalg.norm(x), jnp.abs(x).sum()
```

```
(Array(5., dtype=float32), Array(7., dtype=float32))
```

ℓ_2 は成分を二乗するのに対し ℓ_1 は二乗しないため、ℓ_1 損失は大きな外れ値（outlier）の誤差に対してより頑健であり、ℓ_1 の重みペナルティは重みを厳密な 0 へと導く（スパース性、sparsity）。どちらも ℓ_p **ノルム**の族 $\|\mathbf{x}\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$（$p \geq 1$）に属し、$\|\mathbf{x}\|_\infty = \max_i |x_i|$ が $p \to \infty$ の極限に対応する。あらゆる正当なノルムは、(i) 絶対斉次性 $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$、(ii) 三角不等式（triangle inequality）$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$、(iii) 正定値性（$\|\mathbf{x}\| > 0$ が $\mathbf{x} \neq \mathbf{0}$ に対して成り立つ）を満たす。

$m \times n$ 行列 $\mathbf{A}$ に対して、**フロベニウスノルム**は ℓ_2 ノルムをその mn 個の成分全体について計算したものである。

$$\|\mathbf{A}\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}. \tag{1.11}$$

```
jnp.linalg.norm(jnp.ones((4, 9)))
```

```
Array(6., dtype=float32)
```

1.3.5 固有ベクトルと固有値

50 層のネットワークが入力ベクトル $\mathbf{x}$ に行列を繰り返し掛けると何が起きるだろうか。非ゼロベクトル $\mathbf{v} \neq \mathbf{0}$ が $\mathbf{A} \in \mathbb{R}^{n\times n}$ のスカラー**固有値** λ に対応する**固有ベクトル**（eigenvector）であるとは、$\mathbf{A}$ を掛けても $\mathbf{v}$ が回転せず単に定数倍されることをいう。

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \quad \Longrightarrow \quad \mathbf{A}^k\mathbf{v} = \lambda^k\mathbf{v}. \tag{1.12}$$

先ほどの対称行列 $\mathbf{S} = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$ では、$(1,1)$ は $\lambda = 3$ に対応する固有ベクトルであり、$(-1,1)$ は $\lambda = 1$ に対応する固有ベクトルである（Figure 1.18）。

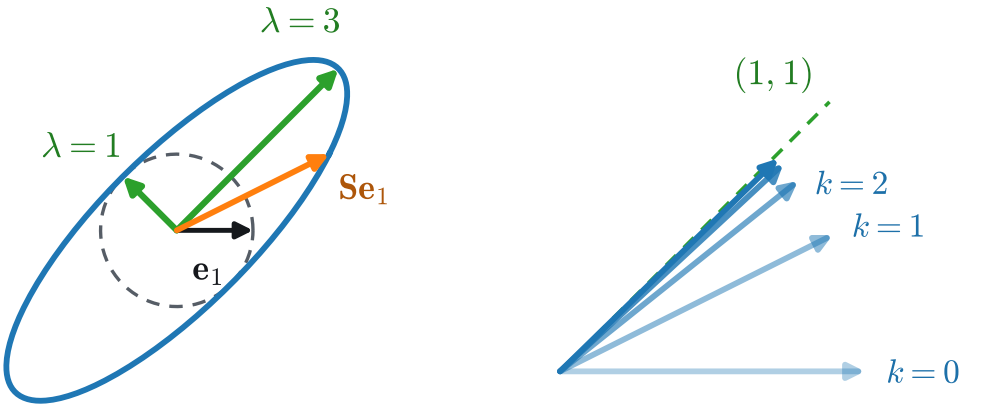


Figure 1.18: 左：$\mathbf{S}$ は単位円（破線）を、固有ベクトルを主軸とし $\lambda = 3$ 倍および $\lambda = 1$ 倍に引き伸ばされた楕円へと写し、$\mathbf{e}_1$ は $\mathbf{S}\mathbf{e}_1 = (2,1)$ へと向きを変える。右：$\mathbf{S}^k\mathbf{e}_1$（$k = 0, \ldots, 4$）の向きは、最大固有値の固有ベクトル $(1,1)$ に近づいていく。

実対称行列の固有値と単位固有ベクトルは `linalg.eigh` を使って計算する。

```
S = jnp.array([[2.0, 1.0], [1.0, 2.0]])
eigenvalues, eigenvectors = jnp.linalg.eigh(S)
eigenvalues, eigenvectors
```

```
(Array([1., 3.], dtype=float32),
 Array([[-0.70710677,  0.70710677],
        [ 0.70710677,  0.70710677]], dtype=float32))
```

スペクトル定理（spectral theorem、33.3 節）により、すべての $n \times n$ 実対称行列 $\mathbf{A}$ は n 個の実固有値と、固有ベクトルからなる完全な正規直交基底（orthonormal basis）$\mathbf{v}_1, \dots, \mathbf{v}_n$ を持つ。任意のベクトルを $\mathbf{x} = \sum_{i=1}^{n} c_i \mathbf{v}_i$（ただし $c_i = \mathbf{v}_i^\top \mathbf{x}$）と展開すると、次式が得られる。

$$\mathbf{A}^k \mathbf{x} = \sum_{i=1}^{n} c_i \lambda_i^k \mathbf{v}_i. \tag{1.13}$$

k が大きくなるにつれて $|\lambda_i|$ が最大の項が支配的になり、$\mathbf{A}^k \mathbf{x}$ は主固有ベクトルの向きに揃いながら 1 ステップあたり $\max_i |\lambda_i|$ 倍で成長する（**べき乗法**、power iteration）。対称行列では、この成長率は任意のベクトルをその行列が引き伸ばせる最大の倍率、すなわち**スペクトルノルム** $\|\mathbf{A}\|_2 = \max_{\mathbf{x} \neq \mathbf{0}} \|\mathbf{A}\mathbf{x}\| / \|\mathbf{x}\|$ にも一致する。

```
v = jnp.array([1.0, 0.0])
for _ in range(10):
    v_prev, v = v, S @ v
growth = jnp.linalg.norm(v) / jnp.linalg.norm(v_prev)
growth, v / jnp.linalg.norm(v)
```

```
(Array(3.0000002, dtype=float32), Array([0.7071187, 0.7070948], dtype=float32))
```

50 層の思考実験に戻ろう。$\mathbf{M} = \left[\begin{smallmatrix} 1 & 0.1 \\ 0.1 & 1 \end{smallmatrix}\right]$ の固有値 $\lambda_1 = 1.1$ と $\lambda_2 = 0.9$ は 1 からわずか 10% しか離れていないが、その 2 つの単位固有ベクトルに $\mathbf{M}$ を 50 回掛けると、1 つ目は $1.1^{50} \approx 117.4$ 倍に増幅され、2 つ目は $0.9^{50} \approx 0.0052$ 倍に縮小する。

```
M = jnp.array([[1.0, 0.1], [0.1, 1.0]])
# The eigenvectors (1, 1) and (-1, 1) of M, as columns of length 1
V = jnp.array([[1.0, -1.0], [1.0, 1.0]]) / 2 ** 0.5
for _ in range(50):  # 50 "layers"
    V = M @ V
jnp.linalg.norm(V, axis=0)
```

```
Array([1.1739086e+02, 5.1537761e-03], dtype=float32)
```

ネットワークの重み行列は対称行列ではなく、その大半は正方行列ですらない。そのような行列がベクトルをどれだけ引き伸ばすかも、固有値からわかるのだろうか。答えは否である。$\mathbf{A} = \left[\begin{smallmatrix} 0 & 2 \\ 0 & 0 \end{smallmatrix}\right]$ の固有値はどちらも 0 であり $\mathbf{A}^2 = \mathbf{0}$ となるが、それでも $\mathbf{A}$ は $\mathbf{e}_2$ を $2\mathbf{e}_1$ に写すため長さを 2 倍にする。$\|\mathbf{A}\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathbf{A}^\top \mathbf{A} \mathbf{x}$ であるから、ベクトルの伸縮を支配するのは対称行列 $\mathbf{A}^\top \mathbf{A}$ である。その固有値の平方根 $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^\top \mathbf{A})}$ が $\mathbf{A}$ の**特異値**（singular value）であり、正方行列か否かにかかわらずすべての行列に対して存在する。$\mathbf{S}$ と $\mathbf{A}$ について計算してみよう。

```
A = jnp.array([[0.0, 2.0], [0.0, 0.0]])  # Both eigenvalues are 0
(jnp.linalg.svdvals(S), jnp.linalg.svdvals(A),
 jnp.sqrt(jnp.linalg.eigvalsh(A.T @ A)), jnp.linalg.matrix_rank(A))
```

```
(Array([3., 1.], dtype=float32),
 Array([2., 0.], dtype=float32),
 Array([0., 2.], dtype=float32),
 Array(1, dtype=int32))
```

対称行列 $\mathbf{S}$ の特異値は（丸め誤差を除けば）3 と 1 であり、固有値の絶対値に等しい。すなわち、対称行列ではこれら 2 つの概念は一致する。$\mathbf{A}$ の特異値は 2 と 0 であり、$\mathbf{A}^\top\mathbf{A}$ の固有値からも同じ 2 つの数値が昇順で得られる。任意の行列において、最大の特異値はそのスペクトルノルムに等しい。また、非ゼロの特異値の個数は行列の**ランク**、すなわち積によって潰れずに残る独立な方向の数であり、$\mathbf{A}$ では 1 である。これら 2 つの事実はいずれも特異値分解（singular value decomposition、33.5 節）から導かれる。

各層の重み行列がベクトルの長さを平均して 1 の近くに保たない限り、活性化値や勾配は深さとともに指数関数的に**爆発**（explode）または**消失**（vanish）する（4.4 節）。

1.3.6 まとめ

線形代数はミニバッチ全体を扱うモデルを簡潔に表現する。平坦化した 256 枚の画像を $\mathbf{X} \in \mathbb{R}^{256\times 784}$ として積み重ねれば、全 2,560 個のカテゴリスコアを $\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}$ によって一括計算できる。リダクションは指定した軸に沿って値を集約する（`keepdims=True` はブロードキャストのために軸を長さ 1 として保持する）。ドット積 $\mathbf{x}^\top\mathbf{y} = \sum_i x_i y_i$ は行列・ベクトル積 $\mathbf{A}\mathbf{x}$ と行列・行列積 $\mathbf{A}\mathbf{B}$（計算量は $\approx 2mkn$ FLOPs）を構成する。ノルム（ℓ_2、ℓ_1、ℓ_p、フロベニウス、スペクトル）はベクトルや行列の大きさを測り、正規化したドット積はコサイン類似度 $\cos\theta = \mathbf{x}^\top\mathbf{y}/(\|\mathbf{x}\|\|\mathbf{y}\|)$ を与え、固有ベクトル（$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$）は行列積の反復によって各成分が λ^k 倍にスケーリングされる理由を説明する。対称行列でない行列に対しては特異値がベクトルの伸縮を測り、その最大値がスペクトルノルムとなる。

1.3.7 演習問題

1. **転置と行列積**。次の代数恒等式を証明せよ。
    1. 行列積の転置が因子の順序を逆転させること、すなわち $(\mathbf{A}\mathbf{B})^\top = \mathbf{B}^\top\mathbf{A}^\top$ が（$\mathbf{A}\mathbf{B}$ が定義される限り常に）成り立つことを示せ。（両辺の (i,j) 成分を Equation 1.6 を用いて書き出せ。）
    2. **任意の**長方行列 $\mathbf{X} \in \mathbb{R}^{m\times n}$ に対して、積 $\mathbf{X}\mathbf{X}^\top$ は常に正方行列かつ対称行列になることを示せ（$\mathbf{X}^\top\mathbf{X}$ についても同様）。本節の 256×784 の画像ミニバッチ $\mathbf{X}$ に対して、$\mathbf{X}\mathbf{X}^\top$ の形状はどうなるか。また、その (i,j) 成分は何を測っているか。
2. [code] **リダクションとブロードキャストの形状予測**。コードを一切実行せずに、`images`（形状 $(256, 1, 28, 28)$）と `X`（形状 $(256, 784)$）に対する以下の各式の結果の形状（またはエラー）を予測し、その後コードで答えを確認せよ。
    1. `len(images)` は何を返すか。`images.sum(axis=(2, 3))` の形状は `keepdims=False` の場合と `keepdims=True` の場合でそれぞれどうなるか。ブロードキャストによって `images` をそのまま割れるのはどちらか。また、`images / images.sum(axis=(2, 3), keepdims=True)` は何を計算しているか。
    2. `X - X.mean(axis=1)` を 256×784 行列 `X` に対して評価すると何が起きるか。また、`keepdims=True` によってそれがどのように修正されるか。
    3. ここで、ミニバッチがたまたま $n = 784$ 枚の画像を含んでおり、`X` が 784×784 の正方行列であるとする。`X - X.mean(axis=1)` が**エラーを一切出さずに**実行されることを示せ。`X - X.mean(axis=1)` の (i,j) 成分が実際に何を計算しているかを `X` の成分を用いて書き出し、正方行列における暗黙のブロードキャストバグが形状エラーよりもはるかに危険である理由を説明せよ。
3. **ベクトルの集合は互いにどこまで鈍角になれるか**。以下の設問では、k 個のベクトル間のすべての対のドット積が同時にどこまで負になれるかを探る。

   1. 平面上の 3 つのベクトル $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^2$ は、$\mathbf{u}^\top \mathbf{v} < 0$、$\mathbf{v}^\top \mathbf{w} < 0$、$\mathbf{u}^\top \mathbf{w} < 0$ を同時に満たせるか。具体的な数値例を挙げるか、存在しないことを証明せよ。
   2. $k \geq 2$ 個の単位ベクトル $\mathbf{u}_1, \dots, \mathbf{u}_k \in \mathbb{R}^d$（$d \geq k-1$）が対称に配置され、異なるすべての対がまったく同じドット積 $\mathbf{u}_i^\top \mathbf{u}_j = c$（すべての $i \neq j$）を持つとする。$0 \leq \|\mathbf{u}_1 + \cdots + \mathbf{u}_k\|_2^2$ を展開することで、$c \geq -\frac{1}{k-1}$ を証明せよ。10 クラス分類器における単位長のクラステンプレート 10 個がすべて等しい角度で離れているとき、この下界は 10 個のテンプレート間の対ごとのコサイン類似度について何を意味するか。

   MIT 18.06、問題セット 1、1.2 節、問題 28 より改題。

4. [code] **複数のパラメータテンソルにわたるノルム。**本節で導入したベクトルノルムのルーチンは、高階テンソルにも適用できる。

   1. 形状 $(2, 3, 4)$ の 3 次元テンソルに対してデフォルト引数でベクトルノルムを評価し、どのような数式を計算しているかを言葉で述べ、各要素を手動で二乗・総和・平方根をとることで答えを確認せよ。
   2. 深層ニューラルネットワークは、学習可能なパラメータをさまざまな階数と形状を持つ重みテンソルおよびバイアステンソルのリストに分けて保持する。訓練の監視や勾配クリッピング（8.4.2 節）の際には、全パラメータを 1 つの巨大なベクトルに連結したときの単一の全体 ℓ_2 ノルムがしばしば必要になる。テンソルを平坦化したり連結したりせずに、この全体 ℓ_2 ノルムを個々のテンソルノルムを用いて表し、形状 $(3, 4)$、$(4,)$、$(2, 3, 5)$ の 3 つのテンソルに対してコードで式を検証せよ。

5. **行列積の結合法則と計算コスト。**行列積は結合法則を満たし、数学的には $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$ が成り立つが、連鎖する行列積に括弧を付ける順序によって計算コストは何桁も変わり得る。$\mathbf{A} \in \mathbb{R}^{1000 \times 50000}$、$\mathbf{B} \in \mathbb{R}^{50000 \times 20}$、$\mathbf{C} \in \mathbb{R}^{20 \times 10000}$ が与えられたとき、$(\mathbf{AB})\mathbf{C}$ と $\mathbf{A}(\mathbf{BC})$ の両方について、総 FLOP 数と中間の積行列の要素数を求めよ。2 つの評価順序で、演算量と中間メモリ量はそれぞれ何倍異なるか。

6. [code] **ループなしの全対ユークリッド距離行列。**●50 個ずつの $\mathbb{R}^8$ のベクトルからなる 2 つの集合が $\mathbf{X} \in \mathbb{R}^{50 \times 8}$ と $\mathbf{Y} \in \mathbb{R}^{50 \times 8}$ の行として積み重ねられているとき、Python のループを使わずに二乗ユークリッド距離の 50×50 行列 $\mathbf{D}$（成分 $D_{ij} = \|\mathbf{x}_i - \mathbf{y}_j\|_2^2$）を計算したい。

   1. まず 3 次元ブロードキャストによって $\mathbf{D}$ を計算せよ。すなわち、$\mathbf{X}$ を $(50, 1, 8)$ に、$\mathbf{Y}$ を $(1, 50, 8)$ に形状変更し、差をとって二乗し、最後の軸に沿って和をとれ。次に、$\mathbf{D}$ を二次式の展開 $\|\mathbf{x}_i - \mathbf{y}_j\|_2^2 = \|\mathbf{x}_i\|_2^2 + \|\mathbf{y}_j\|_2^2 - 2\mathbf{x}_i^\top \mathbf{y}_j$ によって計算せよ。これには行ノルムのリダクション、2 次元ブロードキャスト、および 1 回の行列積 $\mathbf{X}\mathbf{Y}^\top$ しか必要ない。両者の結果が浮動小数点精度の範囲で一致することを確認せよ。
   2. $\mathbf{X}$ と $\mathbf{Y}$ がそれぞれ 10,000 個の $\mathbb{R}^{128}$ のベクトルを含むとき、3 次元ブロードキャスト法が確保しなければならない中間の浮動小数点数の個数は、行列積を用いる方法と比べていくつになるか。大規模データではどちらの方法を使うべきか。
   3. $\|\mathbf{x}_i\|_2^2 + \|\mathbf{y}_j\|_2^2 - 2\mathbf{x}_i^\top \mathbf{y}_j$ における浮動小数点の丸め誤差によって、ごく小さな負の値（-10^{-7} など）が $\mathbf{x}_i = \mathbf{y}_j$ のときに生じ得る理由を説明し、平方根をとる前に出力をクランプする方法を示せ。

   Stanford CS231n、課題 1（k 近傍距離）、および fast.ai、Deep Learning from the Foundations、Lesson 8 より改題。

7. [code] **べき乗法と収束率。**●$\mathbf{v}_{k+1} = \mathbf{A}\mathbf{v}_k / \|\mathbf{A}\mathbf{v}_k\|_2$ をランダムな単位ベクトル $\mathbf{v}_0$ から繰り返し更新する正規化べき乗法を、次の 3×3 対称行列に対して実装せよ。

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 3 & 4 & 5 \end{bmatrix}. \tag{1.14}$$

1. 最初の反復回数 k（連続する反復ベクトル間のコサイン類似度の絶対値 $|\mathbf{v}_k^\top \mathbf{v}_{k-1}|$ が 0.999 を初めて超える時点）を記録せよ。収束したベクトル $\mathbf{v}_k$ を `eigh(A)` が返す主固有ベクトルと比較し、2 つのベクトルが全体の符号 ± 1 だけ異なり得る理由を説明せよ。
2. 各ステップにおいて、$\mathbf{v}_k$ と真の主固有ベクトル $\mathbf{u}_{\max}$ がなす角の正弦 $\sin\theta_k = \sqrt{1 - (\mathbf{v}_k^\top \mathbf{u}_{\max})^2}$ を測定し、1 ステップあたりの縮小比 $\sin\theta_{k+1} / \sin\theta_k$ を計算せよ。この漸近的な比を $\mathbf{A}$ の固有値を用いて Equation 1.13 から表し、べき乗法がごく少数のステップで収束した理由を説明せよ。
3. 次に、シフトした行列 $\mathbf{B} = \mathbf{A} - 9\mathbf{I}$ に対してべき乗法を実行せよ。べき乗法は $\mathbf{A}$ のどの固有値および固有ベクトルに収束するか。また、連続する反復ベクトル $\mathbf{v}_k$ と $\mathbf{v}_{k+1}$ の符号はどうなるか、その理由とともに答えよ。

MIT 18.06、問題セット 1、2.1 節、問題 29〜30 より改題。

1.4 微分積分

古代ギリシャにおいて、Archimedes は辺の数を $n \to \infty$ と増やした正多角形を円に内接させることで円の面積を計算した（Figure 1.19）。三角形が細くなるにつれて、その高さは r に近づき、底辺の合計は円周 $2\pi r$ に近づくため、面積は $\frac{1}{2} r(2\pi r) = \pi r^2$ となる。微小なステップの極限において曲がった量を単純な線形の断片で近似するという考え方は、積分法と**微分法**（differential calculus）の両方の根底にある。

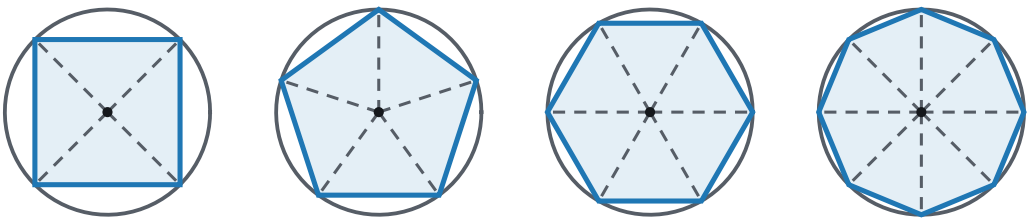


Figure 1.19: 円に内接する辺の数 4、5、6、8 の正多角形。辺の数が増えるにつれて、多角形の面積は円の面積に近づく。

深層学習において、微分法は**最適化**（optimization）の原動力である。モデルの訓練とは、モデルが訓練例にどれだけ当てはまっていないかを測るスカラー値の損失関数を最小化するように、パラメータを調整することを意味する。数百万から数十億個のパラメータがある場合、試行錯誤で各パラメータを探ろうとすれば、更新ごとに数百万回の順伝播が必要になるだろう。その代わりに、**導関数**（derivative）は各パラメータに関する損失の瞬間的な変化率を与える。すべての偏微分を**勾配**（gradient）ベクトルにまとめれば最急上昇の方向がわかり、さらに**連鎖律**（chain rule）を用いれば入れ子になった層全体の勾配を 1 回の逆伝播で組み立てられる。

本節では、すべての数値を手計算で確かめられるほど小さな線形回帰（linear regression）の例を一貫して用いながら、導関数、勾配、連鎖律を展開する。1.5 節では、深層学習フレームワークが任意のプログラムに対してこれらの計算をどのように自動化するかを学ぶ。

1.4.1 導関数

1.4.1.1 極限としての導関数

単一パラメータのモデルがすべてのデータ例に対して定数値 w を予測するとし、ラベルが 1、1、0 である 3 つの訓練例でこのモデルを評価する。二乗誤差（squared error）の和をとると、次のようになる。

$$L(w) = (w-1)^2 + (w-1)^2 + (w-0)^2 = 3w^2 - 4w + 2. \tag{1.15}$$

$w = 1$ における損失は $L(1) = 1$ である。

```
def loss(w):
    return 3 * w ** 2 - 4 * w + 2
```

$L(w)$ が微小な摂動 $h \neq 0$ に対してどのように変化するかを見るため、差分商（difference quotient）$\frac{L(1+h)-L(1)}{h}$ を、h が 10^{-1} から 10^{-5} まで小さくなるときについて計算してみよう。

```
for h in 10.0 ** -np.arange(1, 6):
    print(f'h = {h:.0e}   ratio = {(loss(1 + h) - loss(1)) / h:.5f}')
```

```
h = 1e-01   ratio = 2.30000
h = 1e-02   ratio = 2.03000
h = 1e-03   ratio = 2.00300
h = 1e-04   ratio = 2.00030
h = 1e-05   ratio = 2.00003
```

$L(1+h) = 1 + 2h + 3h^2$ を展開すると、$\frac{L(1+h)-L(1)}{h} = 2 + 3h \to 2$（$h \to 0$ のとき）となることがわかる。厳密には、$f : \mathbb{R} \to \mathbb{R}$ に対し、関数 f の点 x における**導関数**は次の極限で定義される。

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{1.16}$$

幾何学的には、差分商は点 $P = (x, f(x))$ と $Q = (x+h, f(x+h))$ を通る**割線**（secant line）の傾きである。$h \to 0$ につれて、この割線は回転し、点 P における傾き $f'(x)$ の**接線**（tangent line）へと近づく（Figure 1.20）。

すべての関数がいたるところで微分可能とは限らない。たとえば $|x|$ は $x = 0$ に角を持つ（左からの傾きは -1、右からの傾きは $+1$）。さらに悪いことに、離散的な分類の正解率はほとんどいたるところで平坦（$f'(x) = 0$）であり、予測が反転する箇所で不連続に跳び上がるため、勾配信号をまったく与えない。分類器を訓練する際に、交差エントロピー（cross-entropy）のような滑らかな代理損失（surrogate loss）を最小化するのはこのためである（Figure 1.21）。

1.4.1.2 微分の公式

$y = f(x)$ が与えられたとき、導関数を $f'(x) = y' = \frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx} f(x) = Df(x)$ と書く。よく使われる初等関数の導関数は次の通りである。

$$\frac{d}{dx} C = 0, \quad \frac{d}{dx} x^n = n x^{n-1}, \quad \frac{d}{dx} e^x = e^x, \quad \frac{d}{dx} \log x = \frac{1}{x} \quad (x > 0),$$

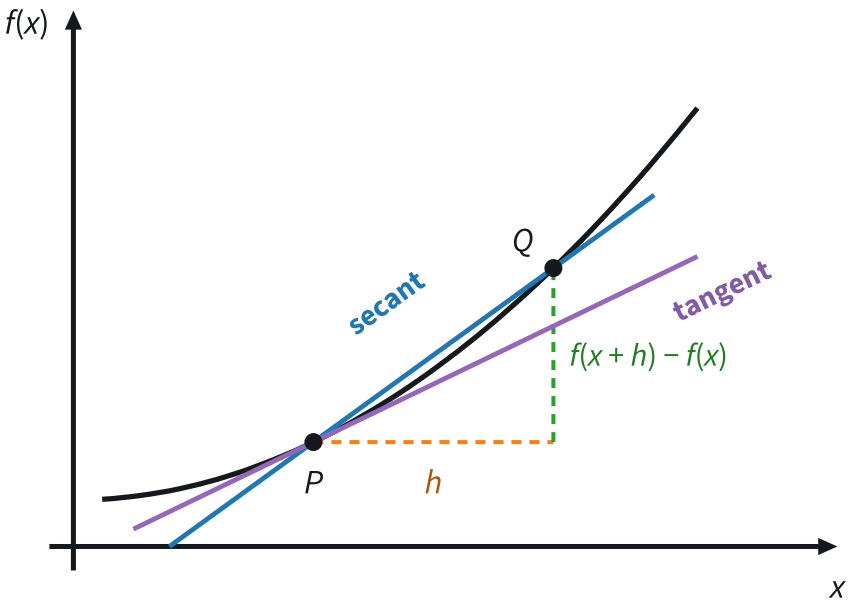


Figure 1.20: 差分商 $(f(x+h)-f(x))/h$ は P と Q を通る割線の傾きである。$h \to 0$ につれて Q は P に向かって移動し、割線は P における接線へと変化して、その傾きは $f'(x)$ となる。

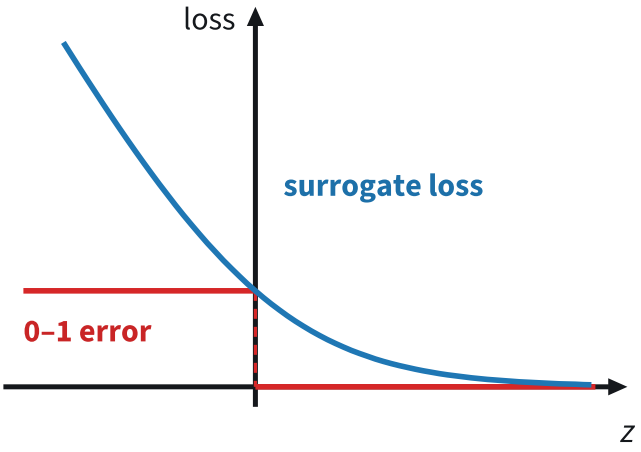


Figure 1.21: 0-1 誤差（誤予測で 1、正解で 0）と滑らかな代理損失を、分類器が正しいときに正となるスコア z に対してプロットしたもの。0-1 誤差は不連続点の両側で平坦であるため、その傾きからは何もわからない。一方、代理損失はいたるところで右下がりの傾きを持ち、パラメータをどちらへ動かすべきかを教えてくれる。

また、微分可能な関数 f, g の組み合わせは次の法則に従う。

$$\frac{d}{dx}[Cf(x)] = Cf'(x), \qquad \frac{d}{dx}[f(x) + g(x)] = f'(x) + g'(x),$$
$$\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + f(x)g'(x), \qquad \frac{d}{dx}\frac{f(x)}{g(x)} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}.$$

Figure 1.22 は、積の微分公式を $f(x) \times g(x)$ の長方形の面積によって幾何学的に示したものである（完全な証明は 34.1 節を参照）。これらの公式を $L(w) = 3w^2 - 4w + 2$ に適用すると直ちに $L'(w) = 6w - 4$ が得られ、したがって $L'(1) = 2$ となる。

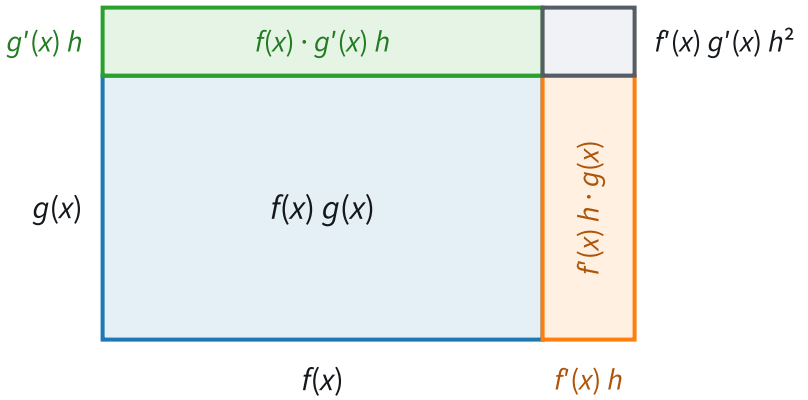


Figure 1.22: 長方形の面積として見た積の微分公式。x が h だけ増えると、辺 $f(x)$ と $g(x)$ はそれぞれ約 $f'(x)h$ と $g'(x)h$ だけ伸び、面積は 2 つの細い帯状の部分と面積 $f'(x)g'(x)h^2$ の角の部分だけ増えるが、この角は小さい h では無視できる。

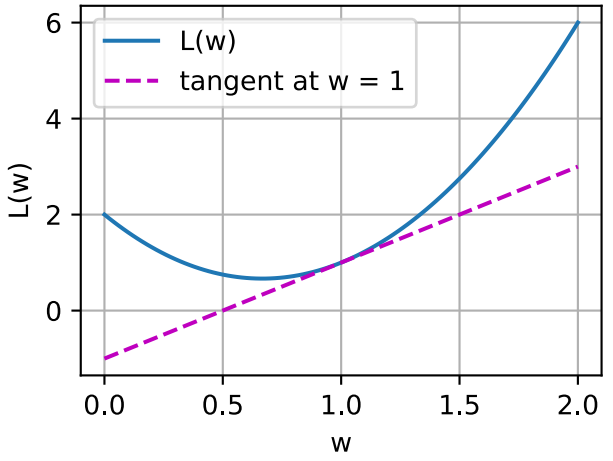
1.4.1.3 一次線形近似と最小値

$\frac{f(x+h)-f(x)}{h} \approx f'(x)$ の両辺に h を掛けると、次の**一次線形近似**（first-order linear approximation）が得られる。

$$f(x + h) \approx f(x) + f'(x)\, h. \tag{1.17}$$

$L'(1) = 2 > 0$ であるため、w を減らせば損失は小さくなる。導関数と逆向きに $w \leftarrow w - \eta L'(w)$（ステップサイズ $\eta = 0.1$）と進むと、w は 1 から $1 - 0.1(2) = 0.8$ へ移動し、L は 1 から $L(0.8) = 0.72$ へと下がる。これは接線による予測値 $1 + 2(-0.2) = 0.60$ に近い（0.12 の差は無視した二次の曲率 $3h^2$ によるものである）。滑らかな極小値（local minimum）では必ず接線が水平（$f'(x) = 0$）になる。そこで $L'(w) = 6w - 4 = 0$ と置くと、唯一の大域的最小点（global minimizer）として $w^* = 2/3$、すなわち 3 つのラベル $1, 1, 0$ の標本平均（sample mean）が得られる。

```
w = np.linspace(0, 2, 101)
d2l.plot(w, [loss(w), 2 * w - 1], 'w', 'L(w)',
         legend=['L(w)', 'tangent at w = 1'])
```



1.4.2 偏微分と勾配

1.4.2.1 偏微分

次に、先ほどの 3 つのデータ例の問題にスカラー入力 0、1、2（対応するラベルはそれぞれ 1、1、0）を与え、アフィン直線 $w_1 + w_2 \cdot \text{input}$ を当てはめてみよう。

$$L(w_1, w_2) = (w_1 - 1)^2 + (w_1 + w_2 - 1)^2 + (w_1 + 2w_2 - 0)^2. \tag{1.18}$$

多変数関数 $y = f(x_1, \dots, x_n)$ において、x_i に関する**偏微分**（partial derivative）は、他のすべての変数を固定したまま座標 i のみを微小変化させる。

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \dots, x_i + h, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{h}. \tag{1.19}$$

$(w_1, w_2) = (1, 0)$ において、$w_2 = 0$ に固定すると $L(w_1, 0) = 3w_1^2 - 4w_1 + 2$ となるため $\frac{\partial L}{\partial w_1}(1, 0) = 2$ である。一方、$w_1 = 1$ に固定すると $L(1, w_2) = 5w_2^2 + 4w_2 + 1$ となり、その導関数は $10w_2 + 4$ であるため $\frac{\partial L}{\partial w_2}(1, 0) = 4$ となる（Figure 1.23）。

行列・ベクトル形式（1.3 節）では、各データ例を $(1, \text{input}_i)$ という $\mathbf{X}$ の行として書くと、次のようになる。

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad L(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2. \tag{1.20}$$

これらの偏微分を数値的に確かめるには、`line_loss(w)` と有限差分（finite difference）の補助関数 `numerical_gradient(f, w, h=1e-6)` を定義すればよい。この補助関数は、`w` の座標を 1 つずつ $h = 10^{-6}$ だけ `float64` で摂動させる。

```
X = np.array([[1.0, 0.0], [1.0, 1.0], [1.0, 2.0]])  # rows [1, input]
y = np.array([1.0, 1.0, 0.0])
```

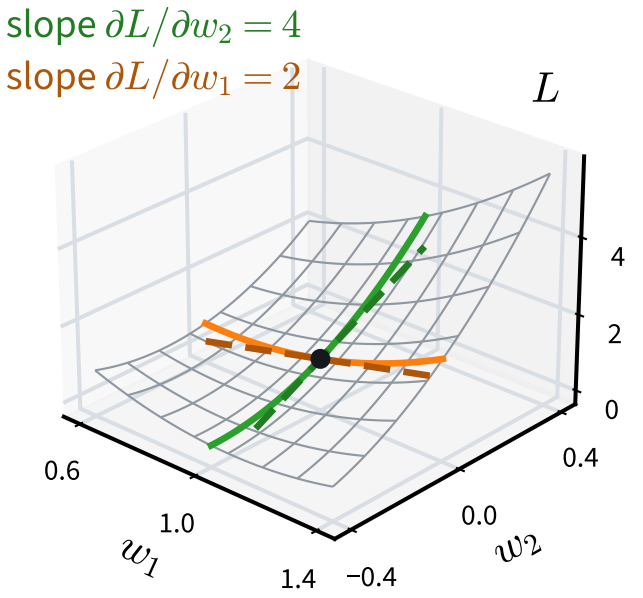


Figure 1.23: 直線モデルの損失 $L(w_1, w_2)$ を $\mathbf{w} = (1, 0)$（黒点）の周辺で示したもの。$w_2 = 0$ に固定すると定数モデルの損失であるオレンジ色の曲線が切り出され、$w_1 = 1$ に固定すると緑色の曲線が切り出される。黒点におけるそれぞれの接線（破線）の傾きは $\partial L/\partial w_1 = 2$ と $\partial L/\partial w_2 = 4$ である。

```
def line_loss(w):
    return np.sum((X @ w - y) ** 2)

def numerical_gradient(f, w, h=1e-6):
    """Finite differences of f at w, moving one coordinate at a time."""
    grad = np.zeros_like(w)
    for i in range(len(w)):
        step = np.zeros_like(w)
        step[i] = h
        grad[i] = (f(w + step) - f(w)) / h
    return grad
```

`numerical_gradient(line_loss, w)` を $\mathbf{w} = (1, 0)$ で評価すると、解析的に求めた偏微分 $(2, 4)$ が $O(h) \approx 10^{-6}$ の打ち切り誤差（truncation error）の範囲内で再現される。

```
w = np.array([1.0, 0.0])
numerical_gradient(line_loss, w)
```

```
array([2.000003, 4.000005])
```

なお、`numerical_gradient` は n 回余分に f を評価する必要があり（パラメータ数が n の場合）、$n = 10^8$ ともなると非現実的になる。

1.4.2.2 有限差分における数値精度の限界

また、有限差分では h が小さすぎるときにも浮動小数点の桁落ちという問題が生じる。有限精度のもとでは、計算機イプシロンが $\epsilon \approx 2.2 \times 10^{-16}$（`float64`）や $\epsilon \approx 1.2 \times 10^{-7}$（`float32`）であるため、ほぼ等しい数値の差 $f(x + h) - f(x)$ をとると上位の有効桁が失われる**桁落ち**（catastrophic cancellation）が起き、$\sim \epsilon$ 程度の丸め残差が h で割られることになる。したがって、**打ち切り誤差**（$\propto h$）と**丸め誤差**（rounding error、$\propto \epsilon/h$）が競合する。

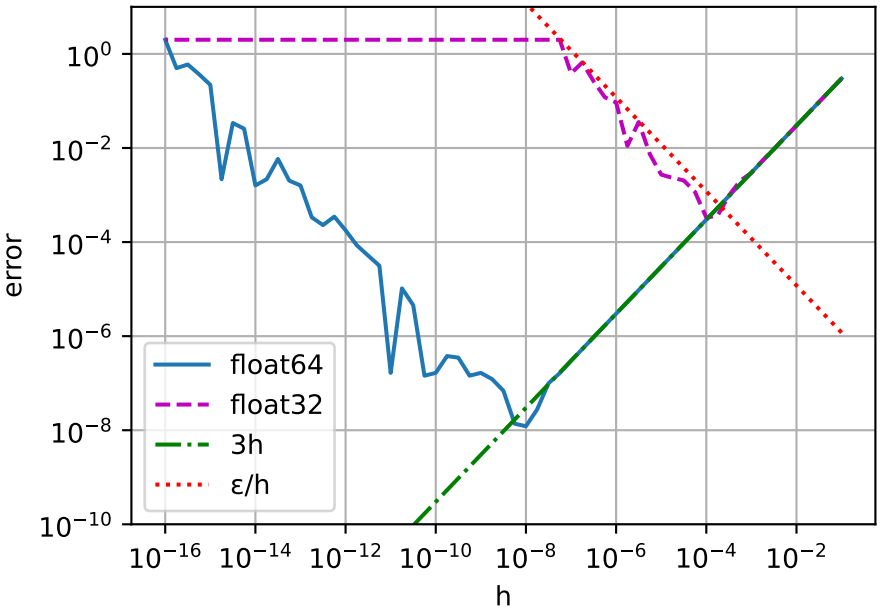
```
def quotient_error(f, x, df, h):
    """Error of (f(x + h) - f(x)) / h as an estimate of f'(x) = df."""
    return abs((f(x + h) - f(x)) / h - df)

h = np.logspace(-1, -16, 61)  # 61 values of h, evenly spaced on a log scale
```

64 ビットの NumPy 演算を、深層学習でデフォルトとして使われる 32 ビット浮動小数点形式（floating-point format）と比較するため、ステップのグリッド `h` を `float32` のフレームワークテンソル `h32` にキャストする。

```
h32 = jnp.asarray(h, dtype=jnp.float32)
```

ここで、`quotient_error(loss, 1, 2, ...)` を $w = 1$（厳密な導関数の値は $L'(1) = 2$）において全 61 個のステップサイズにわたり `float64`（`h`）と `float32`（`h32`）の両方で評価し、打ち切り誤差の直線 $3h$ と `float32` の丸め誤差の漸近線 ϵ/h を重ねて描画する。

```
eps32 = np.finfo(np.float32).eps
d2l.plot(h, [quotient_error(loss, 1, 2, h), quotient_error(loss, 1, 2, h32),
             3 * h, eps32 / h], 'h', 'error', xscale='log', yscale='log',
         ylim=(1e-10, 1e1), legend=['float64', 'float32', '3h', 'ε/h'],
         figsize=(5, 3.5))
```



$3h \approx \epsilon/h$ を釣り合わせると最適なステップサイズは $h^* \approx \sqrt{\epsilon/3}$ となり、その形式が持つ有効桁数の半分が失われる（最小誤差は $\sim 10^{-8}$ に達するのが $h \approx 10^{-8}$ の `float64` であり、$\sim 10^{-4}$ に達するのが $h \approx 10^{-4}$ の `float32` である）。連鎖律を用いた解析的な微分は、$O(n)$ の計算コストと $\sqrt{\epsilon}$ への精度低下の両方を回避できるため、`float64` による有限差分は単体テストでの勾配チェックの用途に残されている。

1.4.2.3 勾配と最急降下

全 n 個の偏微分（関数 $f : \mathbb{R}^n \to \mathbb{R}$ の各変数に関するもの）を連結すると、**勾配**ベクトル $\nabla_{\mathbf{x}} f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})^\top$（$\mathbf{x}$ と同じ形状）が得られる。これにより、Equation 1.17 は同時摂動 $\mathbf{h} \in \mathbb{R}^n$ へと一般化される。

$$f(\mathbf{x} + \mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{h}. \tag{1.21}$$

ステップ $\mathbf{h} = t\mathbf{d}$（長さ $t > 0$、単位ベクトル $\mathbf{d}$ は $\|\mathbf{d}\|_2 = 1$）に対して、Equation 1.8 より $f(\mathbf{x}+t\mathbf{d}) - f(\mathbf{x}) \approx t\,\|\nabla f(\mathbf{x})\| \cos\theta$ となる。したがって、$\nabla f(\mathbf{x})$ は**最急上昇**（steepest ascent）の方向（$\theta = 0°$）を指し、$-\nabla f(\mathbf{x})$ は**最急降下**（steepest descent）の方向（$\theta = 180°$）を指す。また、$\nabla f(\mathbf{x})$ は点 $\mathbf{x}$ を通る等位集合（level set）と直交する（$\theta = 90°$、Figure 1.24）。

これが**勾配降下法**（gradient descent）$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$ の動機となる。$\mathbf{w} = (1, 0)$ から学習率（learning rate）$\eta = 0.1$ で 1 ステップ進めると、$(1, 0) - 0.1(2, 4) = (0.8, -0.4)$ へ移動し、損失は 1.0 から 0.4 に減少する。なぜ Figure 1.24 の最急降下は谷をジグザグに進み、大域

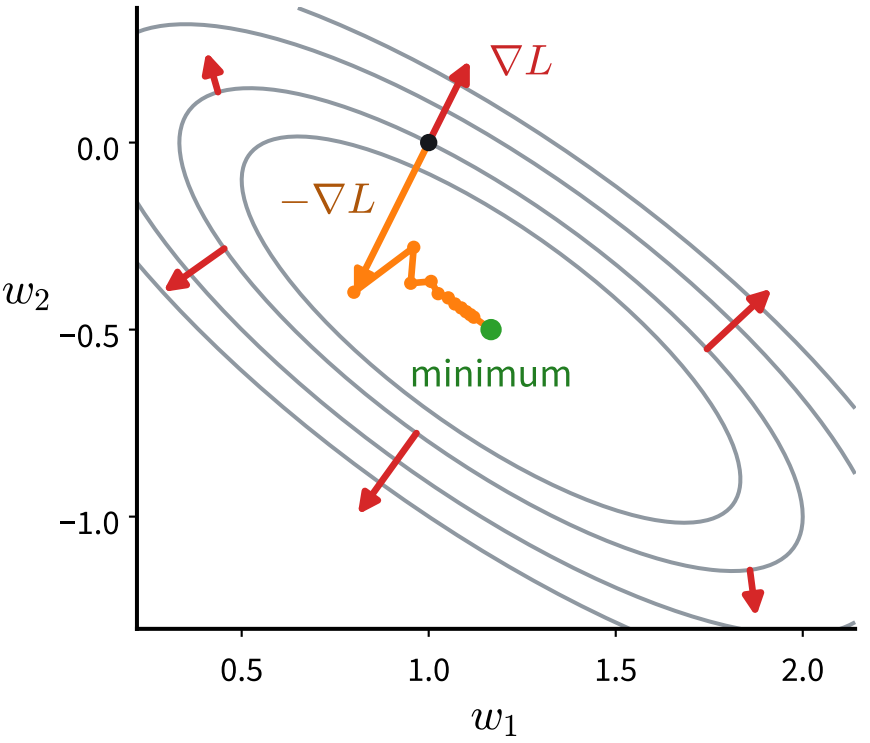


Figure 1.24: 直線モデルの損失 $L(w_1, w_2)$ の等位集合。勾配（赤）はそれぞれの始点を通る等位集合に直交し、上り方向を指す。$\mathbf{w} = (1, 0)$（黒点）から出発すると、$\eta = 0.1$ の勾配降下法（オレンジ）は最小点を直接指していない $-\nabla L$ に沿って進み始め、ジグザグに最小点へと向かう。

的最小値 $\mathbf{w}^{*} = (7/6, -1/2)$ をまっすぐ目指さないのだろうか。それは、「最急」がユークリッド距離でのステップ長 $\|\mathbf{h}\|_2$ の単位あたりで測られ、各座標の単位に依存するからである。1 つの特徴量の軸を引き伸ばすと等高線が潰れ、$-\nabla L$ が最小点から逸れる方向に傾く。実務において特徴量の標準化（1.2 節）や曲率の前処理（preconditioning、第 9 章）が極めて重要になるのはこのためである。

1.4.2.4 行列微分の恒等式

ベクトル勾配に関する 3 つの恒等式（34.4 節）を用いると、座標を書き下すことなくベクトル表現を微分できる。

$$\nabla_{\mathbf{x}} (\mathbf{a}^{\top}\mathbf{x}) = \mathbf{a}, \qquad \nabla_{\mathbf{x}} (\mathbf{x}^{\top}\mathbf{A}\mathbf{x}) = (\mathbf{A} + \mathbf{A}^{\top})\mathbf{x}, \\ \nabla_{\mathbf{x}} \|\mathbf{x}\|^2 = 2\mathbf{x}.$$

$L(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 = \mathbf{w}^{\top}(\mathbf{X}^{\top}\mathbf{X})\mathbf{w} - 2(\mathbf{X}^{\top}\mathbf{y})^{\top}\mathbf{w} + \mathbf{y}^{\top}\mathbf{y}$ を展開してこれらの公式を適用すると、次の式が得られる。

$$\nabla L(\mathbf{w}) = 2\mathbf{X}^{\top}\mathbf{X}\mathbf{w} - 2\mathbf{X}^{\top}\mathbf{y} = 2\mathbf{X}^{\top}(\mathbf{X}\mathbf{w} - \mathbf{y}). \tag{1.22}$$

言葉で言い換えると、勾配の各成分は、$\mathbf{X}$ の対応する特徴量の列と残差（residual）ベクトル $\mathbf{r} = \mathbf{X}\mathbf{w} - \mathbf{y}$ とのドット積（dot product）の 2 倍である。最適解 $\nabla L(\mathbf{w}^{*}) = \mathbf{0}$ においては、予測誤差はすべての特徴量と直交する。

1.4.3 連鎖律

深層ネットワークは、入れ子になった多数の層 $y = f_L(f_{L-1}(\cdots f_1(\mathbf{x})\cdots))$ を合成したものである。合成関数を展開せずに微分するには**連鎖律**を用いる。

スカラー関数 $z = f(u)$ と $u = g(x)$ では、変化率は $\frac{dz}{dx} = \frac{dz}{du}\frac{du}{dx}$ のように掛け合わされる。より一般に、$z = f(u_1, \dots, u_m)$ であり各変数が $u_j = g_j(x_1, \dots, x_n)$ で与えられるとき、変化率は各経路 $x_i \to u_j \to z$ に沿って掛け合わされ、すべての並列経路にわたって足し合わされる。

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial z}{\partial u_j}\frac{\partial u_j}{\partial x_i} \quad \Longrightarrow \quad \nabla_{\mathbf{x}} z = \mathbf{J}^{\top}\nabla_{\mathbf{u}} z, \tag{1.23}$$

ここで、$\mathbf{J} \in \mathbb{R}^{m \times n}$ は $\mathbf{u}$ の $\mathbf{x}$ に関する**ヤコビ行列**（Jacobian matrix）であり、その成分は $J_{ji} = \frac{\partial u_j}{\partial x_i}$ である。アフィン写像 $\mathbf{u} = \mathbf{A}\mathbf{x} - \mathbf{y}$ の場合、ヤコビ行列は単に $\mathbf{J} = \mathbf{A}$ となる。

小さな例を手計算で追ってみよう。$z = u^2 + v$（ただし $u = x_1 x_2$、$v = x_2^2$）とし、$(x_1, x_2) = (1, 2)$ で評価する。入力 x_2 から z へは、u を通る経路と v を通る経路の 2 つの経路がある（Figure 1.25）。それぞれの経路は局所的な導関数の積をもたらし、2 つの寄与を足し合わせることで $\partial z / \partial x_2 = 4 \cdot 1 + 1 \cdot 4 = 8$ が得られる。

直線モデルの損失 $L(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ を 2 つの段階、すなわち（1）内側の残差 $\mathbf{r} = \mathbf{X}\mathbf{w} - \mathbf{y}$（ヤコビ行列は $\mathbf{J} = \mathbf{X}$）と（2）外側の二乗ノルム $L = \|\mathbf{r}\|^2$（勾配は $\nabla_{\mathbf{r}} L = 2\mathbf{r}$）に分解すると、多変数の連鎖律によって $\nabla_{\mathbf{w}} L = \mathbf{X}^{\top}(2\mathbf{r})$ が 2 回のパスで復元される。

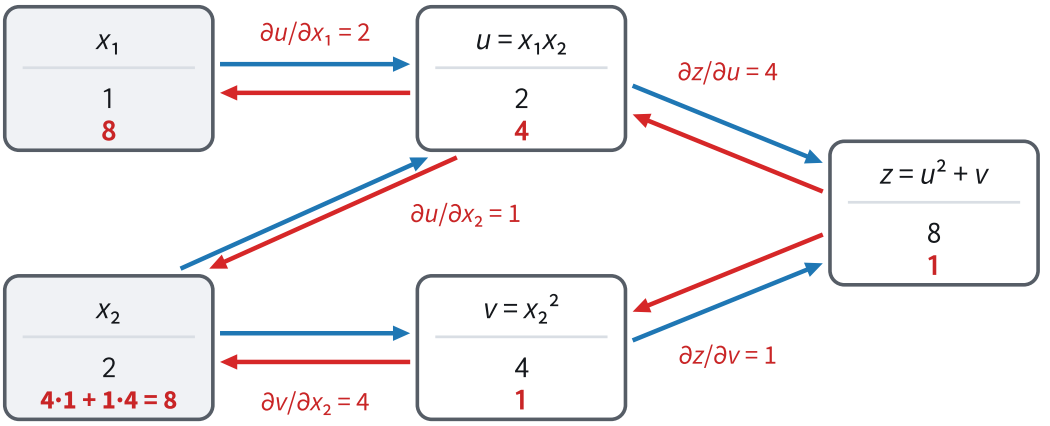


Figure 1.25: $z = u^2 + v$（$u = x_1 x_2$、$v = x_2^2$）の $(x_1, x_2) = (1, 2)$ における連鎖律。各ボックスは変数の値（黒）と z のその変数に関する導関数（赤）を示し、各赤い矢印は局所的な導関数を表す。導関数は経路に沿って掛け合わされ、x_2 のように 2 つの経路が合流する箇所で足し合わされる。

```
r = X @ w - y    # forward: errors, from which the loss is r @ r
grad_r = 2 * r   # backward: gradient of the loss with respect to r
X.T @ grad_r     # backward: gradient with respect to w
```

```
array([2., 4.])
```

順伝播（forward pass）では $\mathbf{r} = \mathbf{X}\mathbf{w} - \mathbf{y}$ と $L = \mathbf{r}^\top \mathbf{r}$ を計算する。一方、**逆伝播**（backward pass）、すなわち**誤差逆伝播法**（backpropagation）では $2\mathbf{r}$ に $\mathbf{X}^\top$ を掛け、おおよそ 1 回の順伝播評価と同じコストですべてのパラメータに関する厳密な勾配を計算する。1.5 節では、この機械的な逆伝播をフレームワークの自動微分（automatic differentiation）エンジンに委ねる。

1.4.4 まとめ

導関数 $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ は接線の傾きと局所線形近似 $f(x+h) \approx f(x) + f'(x)h$ を与える。$f : \mathbb{R}^n \to \mathbb{R}$ に対しては、勾配 $\nabla_{\mathbf{x}} f(\mathbf{x})$ がすべての偏微分をまとめたものとなり、$f(\mathbf{x}+\mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{h}$ を満たし、等位集合に直交して最急上昇の方向を指すことから、勾配降下法 $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$ の根拠となる。$O(n)$ 回の評価コストがかかり桁落ちによって精度が失われる（$\sim \sqrt{\epsilon}$）有限差分とは異なり、多変数の連鎖律 $\nabla_{\mathbf{x}} z = \mathbf{J}^\top \nabla_{\mathbf{u}} z$ は、1 回の逆伝播（**誤差逆伝播法**）によって損失からすべてのパラメータへ厳密な勾配を逆方向に伝播させる。

1.4.5 演習問題

1. **底と指数の両方が変化するときの x^x の微分。** $x > 0$ に対し、関数 $f(x) = x^x$（x が底と指数の両方に現れる）を考える。
    1. べき乗の微分公式のみを適用すること（指数を定数とみなして $x \cdot x^{x-1} = x^x$ を得ること）と、指数関数の微分公式のみを適用すること（底を定数とみなして $x^x \log x$ を得ること）がいずれも誤りである理由を説明せよ。
    2. $x^x = e^{x \log x}$ と書き換え、連鎖律と積の微分公式を適用して $f'(x)$ を求めよ。真の導関数が小問 1 の 2 つの単純な答えの**和**になっていることに注目し、なぜそうなるのか

を、$g(u, v) = u^v$ に対して $u = x, v = x$ に沿って 2 変数の連鎖律を適用することで説明せよ。

2. **なぜ二乗誤差からは平均が得られ、絶対誤差からは中央値が得られるのか。**単一パラメータのモデルが定数値 $w \in \mathbb{R}$ を、n 個の訓練例（スカラーラベルは $y_1, \dots, y_n$）に対して予測するとする。
   1. 二乗誤差の和 $L_2(w) = \sum_{i=1}^{n} (w - y_i)^2$ を w に関して微分し、導関数をゼロと置いて、唯一の最小点が標本**平均** $w^* = \frac{1}{n} \sum_{i=1}^{n} y_i$ であることを証明せよ。
   2. 次に、絶対誤差の和 $L_1(w) = \sum_{i=1}^{n} |w - y_i|$ を考える。任意の点 w（どのラベル y_i とも等しくない点）において、導関数 $L_1'(w)$ は w より真に小さいラベルの個数から w より真に大きいラベルの個数を引いた値に等しいことを示せ。w がどの値をとるとき $L_1'(w)$ は存在しないか。以上から、$L_1(w)$ は $\{y_1, \dots, y_n\}$ の任意の標本**中央値**によって最小化されることを導け。
   3. 4 つのラベル $\{1, 1, 0, 10\}$ に対し、L_2 の最小点（平均）と L_1 の最小点（中央値）を計算せよ。外れ値のラベル 10 が破損して 100 になった場合、それぞれの最小点はどうなるか。

3. **ユークリッドノルムの勾配。**勾配 $\nabla_{\mathbf{x}} f(\mathbf{x})$ を、二乗していないユークリッドノルム $f(\mathbf{x}) = \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$ に対して任意の非ゼロベクトル $\mathbf{x} \neq \mathbf{0}$ で計算し、勾配が存在するすべての点で $\|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2 = 1$ となることを確かめよ。原点 $\mathbf{x} = \mathbf{0}$ では勾配はどうなるか。また、それはなぜか。

4. **勾配降下法のステップサイズ安定性。**等方的な二次関数 $f(\mathbf{x}) = \|\mathbf{x}\|_2^2$ を、勾配降下法の更新式 $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$（学習率 $\eta > 0$）によって最小化することを考える。
   1. $\mathbf{x}_0 = (1, 1)^\top$ から出発し、1 ステップ後の $\mathbf{x}_1$ と新しい目的関数値 $f(\mathbf{x}_1)$ を、3 通りの学習率 $\eta = 0.1$、$\eta = 1.0$、$\eta = 2.0$ について計算せよ。
   2. $\mathbf{x}_{k+1}$ を $\mathbf{x}_k$ と η を用いた閉じた式で表せ。学習率 $\eta > 0$ がどの範囲にあるとき、勾配降下法の 1 ステップによって $f(\mathbf{x})$ は任意の非ゼロの初期点 $\mathbf{x}_k \neq \mathbf{0}$ から真に減少するか。また、η がどの特別な値のとき 1 ステップで大域的最小値 $\mathbf{x}^* = \mathbf{0}$ へ直接到達するか。さらに、$\eta > 1$ のときは何が起きるか。
   3. 勾配降下法のステップでパラメータベクトルが変化しない停留点 $\mathbf{x}$（$\mathbf{x}_{k+1} = \mathbf{x}_k$ を満たす点）を、$f(\mathbf{x}) = \|\mathbf{x}\|_2^2$ の場合と一般の微分可能な目的関数 f の場合の両方について特徴づけよ。

5. [code] **中心差分と前進差分。**●本節で解析した片側の**前進差分** $\frac{f(x+h) - f(x)}{h}$ は $O(h)$ の打ち切り誤差を持つ。対称な**中心差分**

$$\frac{f(x + h) - f(x - h)}{2h} \tag{1.24}$$

は関数 f を $x + h$ と $x - h$ で評価することで、$O(h)$ となる二次の項が相殺されるようにしている。中心差分版の `quotient_error` を実装し、対数グリッド上の h の値にわたって `float64` で評価せよ。

   1. 二次の損失 $L(w) = 3w^2 - 4w + 2$ の $w = 1$ において、前進差分と中心差分の誤差曲線を同じ両対数軸上にプロットせよ。二次関数では、なぜ中心差分の誤差が計算機イプシロン（$\sim 10^{-16}$）まで、大きな $h = 10^{-1}$ であっても一気に下がるのか。
   2. 次に、二次関数ではない $f(x) = e^x$ の $x = 0$（$f'(0) = 1$）において同じ比較を繰り返せ。大きい h における中心差分曲線の打ち切り誤差領域の傾きを、傾き 1 の前進差

分曲線と比較し、各手法における最適なステップサイズ h^* と達成可能な最小誤差を求め、テイラー展開 $e^{\pm h} = 1 \pm h + \frac{h^2}{2} \pm \frac{h^3}{6} + \cdots$ を用いてその両方を説明せよ。

6. [code] **`float64` と `float32` における勾配チェック**。●有限差分は訓練には遅すぎるものの、中心有限差分は手計算で導出した勾配やカスタムカーネルの勾配を単体テストする際の定番手法である。
    1. 関数 `central_gradient(f, x, h=1e-5)` を実装し、中心差分によって勾配 $\nabla f(\mathbf{x})$（$f : \mathbb{R}^n \to \mathbb{R}$）を推定せよ。これを `float64` で用い、二次形式の勾配の恒等式 $\nabla_{\mathbf{x}}(\mathbf{x}^\top \mathbf{A}\mathbf{x}) = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$ をランダムな**非対称** 5×5 行列 $\mathbf{A}$ とランダムなベクトル $\mathbf{x} \in \mathbb{R}^5$ に対してテストし、5 つの座標にわたる最大相対誤差を求めよ。
    2. 次に、**バグのある**式 $2\mathbf{A}\mathbf{x}$（$\mathbf{A}$ が対称でないことを忘れたときによくある誤り）に対して同じチェックを実行せよ。このとき相対誤差はどのくらい大きくなるか。
    3. 最後に、正しい式 $(\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$ のチェックを、32 ビット `float32` のフレームワークテンソルを用いて $h \in \{10^{-1}, 10^{-2}, \ldots, 10^{-6}\}$ にわたって繰り返せ。どの h が `float32` で最小の相対誤差を達成するか。また、それが前進差分の最適値 $h^* \approx 10^{-4}$ と異なるのはなぜか。さらに、`float32` において正しい勾配実装とバグのある実装を明確に切り分けられる相対誤差のしきい値はどの程度か。

1.5 自動微分

1.4 節では、2 つのパラメータをもつ直線モデルの勾配 $\nabla_{\mathbf{w}} L = \mathbf{X}^\top(2\mathbf{r}) = (2, 4)$ を手計算で容易に導出できたが、数十の層と数十億個のパラメータにわたって手計算で導出するのは絶望的である。現代の深層学習フレームワークは、この計算を**自動微分**（automatic differentiation、**autograd**）によって自動化する。プログラムがスカラーの損失を評価する際、フレームワークは基本演算をトレースして**計算グラフ**（computational graph）を構築し、そのグラフを逆向きにたどることで（**誤差逆伝播法**、backpropagation）、連鎖律（chain rule）を機械的に適用する。驚くべきことに、モデルのパラメータが 2 個であっても 10 億個であっても、1 回の逆方向スイープだけで**すべて**のパラメータに関する厳密な勾配を、順伝播（forward pass）のコストのわずかな定数倍で計算できる。

```
import jax
from jax import numpy as jnp
```

1.5.1 勾配の計算

1.5.1.1 順伝播

まず、1.4 節で扱った直線モデル $L(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ を $\mathbf{w} = (1, 0)$ においてフレームワークのテンソルで定義し、順伝播のみを記述する。

```
X = jnp.array([[1.0, 0.0], [1.0, 1.0], [1.0, 2.0]])  # [1, input]
y = jnp.array([1.0, 1.0, 0.0])
w = jnp.array([1.0, 0.0])

def loss(w):
    return ((X @ w - y) ** 2).sum()
```

JAX は純粋関数型のアプローチをとる。配列は可変な勾配バッファをもたないため、`loss(w)` をそのまま評価しても通常の配列演算が実行されて `1.0` が返るだけである。その代わり、以下で微分

する際には jax.grad(loss) が loss をトレースする。

```
loss(w)
```

```
Array(1., dtype=float32)
```

記録されたトレースは Figure 1.26 に示す**計算グラフ**を形成し、損失を $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$、$\mathbf{r} = \hat{\mathbf{y}} - \mathbf{y}$、$\mathbf{s} = \mathbf{r} \odot \mathbf{r}$、$L = \sum_i s_i$ という 4 つの基本演算に分解する。

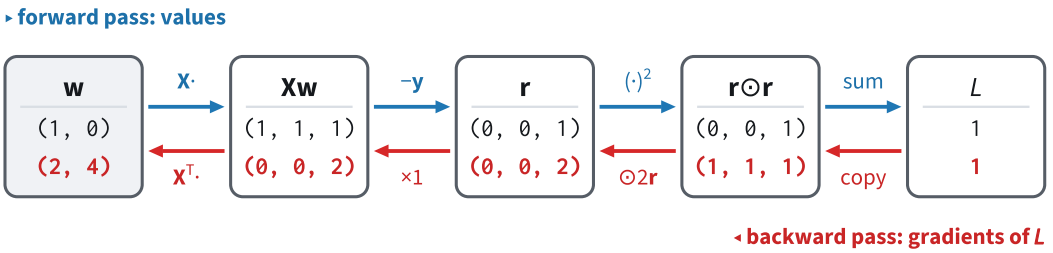


Figure 1.26: $L = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ の $\mathbf{w} = (1, 0)$ における計算グラフ。$\mathbf{w}$ およびそこから計算される各値に 1 つずつノードがあり、定数 $\mathbf{X}$ と $\mathbf{y}$ は省略されている。また、$\mathbf{r} \odot \mathbf{r}$ は要素ごとの二乗を表す。各ノードにはその値（黒）と、それに関する L の勾配（赤）を示す。順伝播（青）は左から右へ進み、逆伝播（赤）は 1 を L に与えて右から左へ各演算の局所規則を適用する。

1.5.1.2 逆伝播

逆伝播（backward pass）を実行すると、出力における $\partial L/\partial L = 1$ から各ノードの局所微分規則を通じて $\mathbf{w}$ へと感度が逆向きに伝播する。

$$\nabla_{\mathbf{s}} L = \mathbf{1} \to \nabla_{\mathbf{r}} L = 2\mathbf{r} \to \nabla_{\hat{\mathbf{y}}} L = 2\mathbf{r} \to \nabla_{\mathbf{w}} L = \mathbf{X}^\top (2\mathbf{r}), \tag{1.25}$$

これにより、先ほどと同じ $(2, 4)$ が得られる。

jax.grad(loss)(w) は勾配関数を w で評価する。

```
jax.grad(loss)(w)
```

```
Array([2., 4.], dtype=float32)
```

逆方向の局所規則では順伝播の中間活性化値（intermediate activation、ここでは $\mathbf{r}$ や $\mathbf{X}$ など）が必要になるため、訓練は推論（inference）よりも大幅に多くのメモリを消費する（Figure 1.27）。

1.5.1.3 逆伝播の繰り返し

同じパラメータベクトル $\mathbf{w}$ で勾配をもう一度評価するとどうなるだろうか。

jax.grad(loss)(w) は呼び出しのたびに新しい配列を返し、w に状態を保持しない純粋関数であるため、繰り返し呼び出しても副作用なしに常に (2, 4) を返す。マイクロバッチにわたって勾配を累積するには、返された配列を明示的に足し合わせればよい。

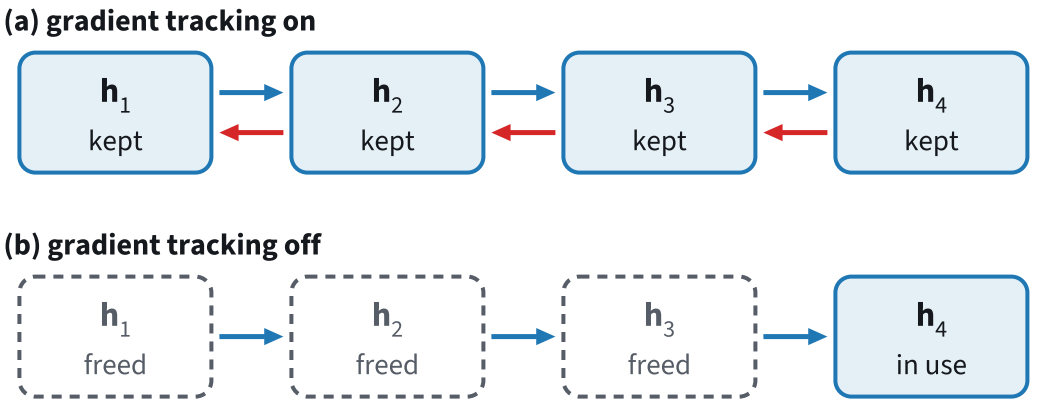


Figure 1.27: 出力 $\mathbf{h}_1, \ldots, \mathbf{h}_4$ をもつ 4 つの層を通る順伝播でのメモリ使用状況。(a) 勾配を追跡する場合、逆伝播（赤）で必要になるため、すべての出力が保持される。(b) 推論時のように勾配を追跡しない場合、次の層が使い終えた時点で出力を解放できる。

1.5.1.4 ベクトル値関数

ある演算が m 個の出力からなるベクトル $\mathbf{u} \in \mathbb{R}^m$ を n 個の入力 $\mathbf{x} \in \mathbb{R}^n$ から生成し、その $m \times n$ のヤコビ行列（Jacobian）を $\mathbf{J}$ とする。このとき、重みベクトル $\mathbf{v} \in \mathbb{R}^m$ をシードとして逆伝播を実行すると、転置を用いた**ベクトル・ヤコビ行列積**（vector–Jacobian product）$\mathbf{J}^\top \mathbf{v} = \nabla_{\mathbf{x}}(\mathbf{v}^\top \mathbf{u})$ が、$\mathbf{J}$ を明示的に構築することなく計算される。$\mathbf{v} = \mathbf{1}$ を例ごとの二乗誤差 (r_1^2, r_2^2, r_3^2) に渡すと、その和が微分される。

`jax.vjp(losses, w)` は順伝播の値と、$\mathbf{v}$ を $(\mathbf{J}^\top \mathbf{v},)$ に写す関数を返す。

```
def losses(w):
    return (X @ w - y) ** 2  # one loss per example

_, vjp = jax.vjp(losses, w)  # vjp maps v to J^T v
vjp(jnp.ones(3))[0]  # v = 1
```

```
Array([2., 4.], dtype=float32)
```

1.5.2 グラフの制御

1.5.2.1 値の切り離し

テンソルの**切り離し**（detaching）とは、数値は同じだが計算グラフ上の祖先とのつながりをもたないビューまたはコピーを作る操作であり、誤差逆伝播法ではそのテンソルが定数として扱われる。$u = x^2$ と $z = ux$ を $\mathbf{x} = (0, 1, 2, 3)$ 上で考えよう（Figure 1.28）。u がグラフに接続されたままであれば、勾配は両方の経路を流れて $\partial z/\partial x = 3x^2$ となる。一方、u を切り離して c とすると、$z = cx$ の微分では c が定数として扱われ、$\partial z/\partial x = c = x^2$ が得られる（15.4 節の深層 Q 学習のように、ネットワークが自身のブートストラップターゲットを計算する場合には不可欠である）。

`jax.lax.stop_gradient(u)` は、微分対象の関数内で勾配を遮断する。

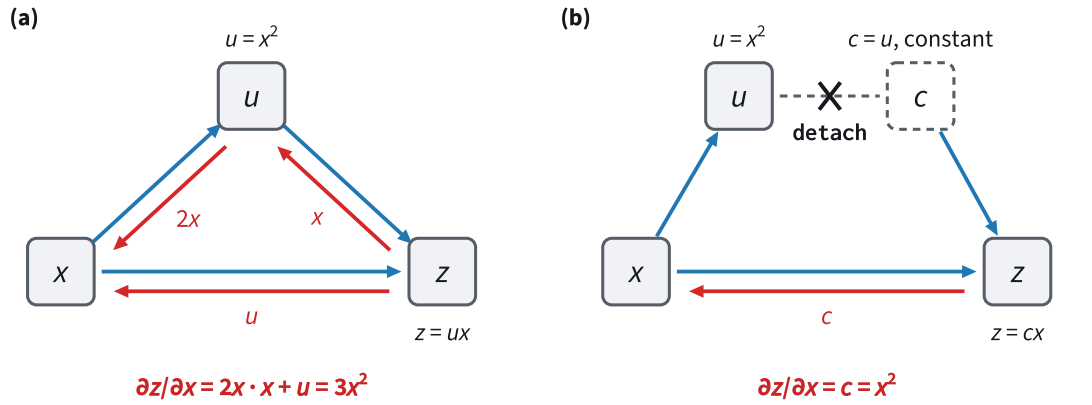


Figure 1.28: 積 $z = ux$（ただし $u = x^2$）における、(a) 接続したままの場合と (b) 切り離した場合。青い矢印は順伝播の演算を表し、赤い矢印は逆伝播とそこで乗じられる局所導関数を表す。u を切り離すと、同じ値をもつ定数 c が作られ、x とのつながりは断たれる。

```
x = jnp.arange(4.0)

def z(x):
    u = x * x
    c = jax.lax.stop_gradient(u)  # values of u, no history
    return c * x

jax.grad(lambda x: z(x).sum())(x)
```

```
Array([0., 1., 4., 9.], dtype=float32)
```

対照的に、`z = u * x` を `u` の切り離しなしに直接計算すると、誤差逆伝播法は $z = x^3$ の両方の因子を通って伝播し、$\partial z/\partial x = 3x^2 = (0, 3, 12, 27)$ を返す。

```
jax.grad(lambda x: (x * x * x).sum())(x)
```

```
Array([ 0.,  3., 12., 27.], dtype=float32)
```

1.5.2.2 勾配追跡の無効化

検証、推論、あるいはパラメータ更新 $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$ の際には、それらの演算が損失の計算グラフに記録されないよう勾配追跡を無効化する。

`jax.grad` の外側では、配列演算はトレースされずに実行される。

```
w = w - 0.1 * jax.grad(loss)(w)
w, loss(w)
```

```
(Array([ 0.8, -0.4], dtype=float32), Array(0.4, dtype=float32))
```

1.4 節と同様、$\eta = 0.1$ による 1 ステップで $\mathbf{w}$ は $(1, 0)$ から $(0.8, -0.4)$ へ移動し、損失は 1 から 0.4 へと下がる。

1.5.3 プログラムの微分

1.5.3.1 制御フロー

自動微分は与えられた入力に対して**実際に実行された**基本演算の系列を記録するため、データに依存する `while` ループや `if` と `else` の分岐を含む Python 関数もそのまま微分できる。

```
def f(a):
    b = a * 2
    while jnp.linalg.norm(b) < 1000:
        b = b * 2
    if b.sum() > 0:
        c = b
    else:
        c = 100 * b
    return c
```

任意の非ゼロのスカラー a に対して（$a = 0$ では `while` ループが終了しない）、`f(a)` は 2（および場合によっては 100）を掛ける定まった回数の乗算を実行するため、`f(a) = k * a` は局所的な傾き $k = f(a)/a$ をもつ区分線形（piecewise linear）関数となる（Figure 1.29）。具体的には、$a = 0.3$ のときは 12 回倍増されて $0.3 \times 2^{12} = 1228.8 \geq 1000$ に達し、正の分岐を通る（$k = 2^{12} = 4096$）。一方、$a = -0.75$ のときは 11 回倍増されて -1536 となり、`else` 分岐を通る（$k = 100 \times 2^{11} = 204{,}800$）。自動微分はこれら両方の傾きを正しく求める。

```
for value in (0.3, -0.75):
    a = jnp.array(value)
    print(f'a = {value:5.2f}   f(a) / a = {f(a) / a:6.0f}   '
          f'gradient = {jax.grad(f)(a):6.0f}')
```

```
a =  0.30   f(a) / a =   4096   gradient =   4096
a = -0.75   f(a) / a = 204800   gradient = 204800
```

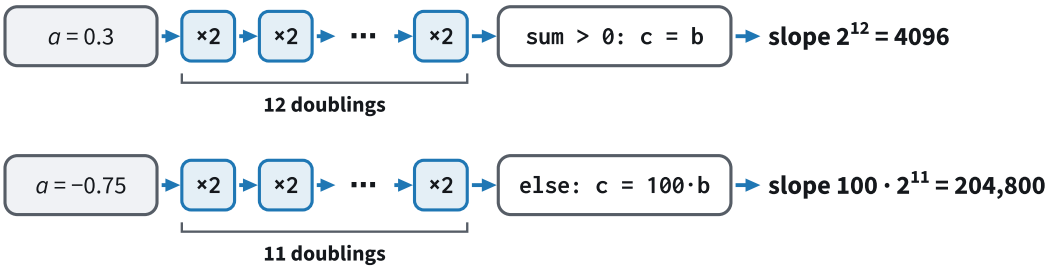


Figure 1.29: 2 つの入力に対して `f` が記録した計算グラフ。$a = 0.3$ では、`f` は a を 12 回倍増させて最初の分岐を実行する。$a = -0.75$ では、a を 11 回倍増させて 2 番目の分岐を実行する。いずれのグラフも a に関して線形であり、それぞれ固有の傾きをもつ。

なお、自動微分が微分するのは抽象的な数学的関数ではなく、**実行されたトレース**であることに注意しよう。ループ回数が切り替わる閾値では $f(a)$ は不連続になるが、自動微分は実行された側の分岐の傾きをそのまま返す。

1.5.4 フォワードモードとリバースモード

写像 $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ とその $m \times n$ ヤコビ行列 $\mathbf{J}$ に対して、連鎖律は順方向と逆方向のどちらの向きにもたどることができる（Figure 1.30、詳細は Wengert (1964); Linnainmaa (1970) および 34.5 節を参照）。

- **リバースモード（ベクトル・ヤコビ行列積、VJP）：**出力に $\mathbf{v} \in \mathbb{R}^m$ をシードとして与える逆方向スイープは $\mathbf{J}^\top \mathbf{v}$ を計算する。j^{th} の標準基底ベクトル $\mathbf{v} = \mathbf{e}_j$ をシードとすれば、第 j 行を $\mathbf{J}$ から取り出せる（ヤコビ行列全体を得るには m 回のスイープを要する）。
- **フォワードモード（ヤコビ行列・ベクトル積、JVP）：**入力に接ベクトル $\mathbf{d} \in \mathbb{R}^n$ をシードとして与える順方向スイープは、値とともに微分値を順方向に伝播させて $\mathbf{Jd}$ を計算する。$\mathbf{d} = \mathbf{e}_i$ をシードとすれば、第 i 列を $\mathbf{J}$ から取り出せる（ヤコビ行列全体を得るには n 回のスイープを要し、メモリオーバーヘッドは定数の $2\times$ で済む）。

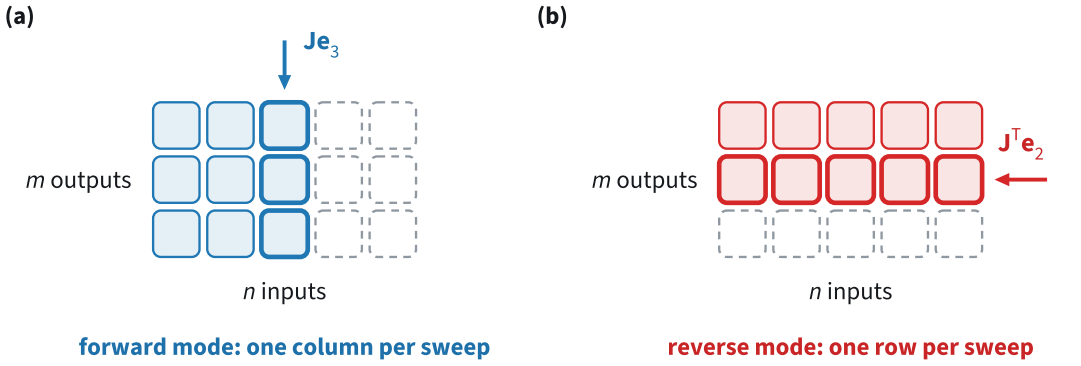


Figure 1.30: $m \times n$ ヤコビ行列 $\mathbf{J}$（n 入力 m 出力の関数のもの）を埋める 2 つの方法。(a) フォワードモード（計 n 回のスイープ）、(b) リバースモード（計 m 回のスイープ）。

JAX は `jax.vjp`、`jax.jvp`、`jax.jacrev`、`jax.jacfwd` を提供している。

スカラーの損失関数は出力が $m = 1$ 個、パラメータが n 個であるため、そのヤコビ行列は 1 行のみの $\nabla_{\mathbf{w}} L^\top$ となる。リバースモード（**誤差逆伝播法**）は勾配の全 n 成分を **1 回のスイープ**で計算できるのに対し、フォワードモードでは n 回のスイープが必要になる。深層ネットワークの訓練でリバースモードが普遍的に使われているのはこのためである。

1.5.5 まとめ

自動微分は、順伝播の基本演算をトレースして計算グラフを構築し、局所微分規則を逆向きに適用することで厳密な勾配を計算する。これにより、直線モデルでも手書きの微分コードなしに $\nabla_{\mathbf{w}} L = (2, 4)$ を再現できた。中間値を切り離すと計算履歴が断たれ、誤差逆伝播法ではその値が定数として扱われる。一方、勾配追跡を無効化すれば、パラメータ更新や推論時の評価が計算グラフに記録されず、順伝播の活性化値のキャッシュも回避できる。計算グラフは与えられた入力に対して実際に実行された演算を記録するため、自動微分は Python のループや条件分岐もシームレスに微分できる。最後に、フォワードモードの 1 回のスイープはヤコビ行列・ベクトル積 $\mathbf{Jd}$（基底ベクトルのシード 1 つにつき 1 列）を計算するのに対し、リバースモードの 1 回のスイープはベクトル・ヤコビ行列積 $\mathbf{J}^\top \mathbf{v}$（基底ベクトルのシード 1 つにつき 1 行）を計算する。したがって、スカラーの損失（$m = 1$）に対しては 1 回の逆方向スイープだけで n 個すべてのパラメータの勾配が得られる。

1.5.6 演習問題

1. [code] **中間値の切り離し**。`x` の関数を 3 つ書き、それぞれ `u = x * x` と `c = jax.lax.stop_gradient(u)` を計算して `c * c`、`u * c`、`u * u` を返すようにせよ。コードを実行する前に、各結果の要素の和の `x` に関する勾配を予測せよ。`jax.grad` で予測を確かめ、`u * c` と `u * u` に対する結果の比を説明せよ。

2. [code] **手動でのパラメータ更新**。`w = jnp.arange(4.0)` および `g = lambda x: (w * x).sum()`（外側のスコープから `w` を参照する関数）とする。`jax.grad(g)(x)` を予測して確かめ、`jax.grad` がどの配列に関して微分しているかを述べよ。次に、`g` を書き換えて `jax.grad` が `(w * x).sum()` の `w` に関する勾配を返すようにし、それを用いて `w` を学習率 0.1 で 1 回更新せよ。また、この結果から JAX の訓練ループにおいて損失関数へパラメータをどのように渡さなければならないかを説明せよ。

3. [code] **スカラー入力からベクトル入力へ**。スカラー入力 `a`（1.5.3.1 節で定義した関数 `f` の入力）を長さ 5 のランダムなベクトルに置き換え、`fa = f(a)` の要素の和を `a` に関して微分せよ。

    1. `f(a)` を `a` を用いて表し、`f` の `a` におけるヤコビ行列、および `fa` の要素の和の勾配を求めよ。
    2. 自動微分でこの勾配を計算し、要素ごとに `fa / a` と一致するかどうかを確かめよ。
    3. 入力 `a` のうち `f` が不連続に跳ぶ点の集合を記述せよ。この集合のうち、スカラー入力の場合には対応するものがない部分はどれか。

4. [code] **プログラムと関数**。自動微分が微分するのはプログラムが実行する演算であり、プログラムが計算する数学的関数そのものではない。

    1. 関数 $|x|$ と $\sqrt{x^2}$ は等しい。フレームワークの絶対値と平方根を用いて、$x = 0$ におけるそれぞれの微分値を自動微分で計算せよ。得られた両方の結果を、関与する演算の局所導関数から説明せよ。どちらの関数も 0 で導関数をもたないが、訓練においてより大きな悪影響を及ぼすのはどちらの結果か。また、それはなぜか。
    2. 出力として x^2 をすべての入力 x に対して返すが、$x = 0$ における自動微分の結果が 1（0 ではなく）になるプログラムを書け。自動微分がその違いを検出できない理由を説明せよ。

5. **二階導関数のコスト**。●1.5.4 節の回数評価の議論によれば、n 個のパラメータをもつスカラー損失の勾配のコストは逆方向スイープ 1 回分である。同じ議論を用いて、ヘッセ行列全体（二階導関数からなる $n \times n$ 行列）のコストと、1 回のヘッセ行列・ベクトル積のコストを関数評価回数を単位として見積もれ。また、この 2 つのコストの違いを説明せよ。

1.6 確率と統計

機械学習の本質は、不確実性のもとでの推論にある。患者が心臓発作を起こす可能性を予測すること、センサの測定値が異常かどうかを判断すること、あるいは正解率（accuracy）があるミニバッチの 0.90（テスト画像 $230/256$ 枚）から次のミニバッチで 0.87 に変化したとき、それが実質的な変化なのかランダムなサンプリングによる揺らぎなのかを見極めることなど、いずれもその例である。

確率論における最も明快な思考実験は、やはりコイン投げである。コインを 100 回投げて表が 56 回出たとしたら、このコインには偏りがあると疑うべきだろうか。公平なコインであっても、表がちょうど 50 回出ることはまれであるため、表が 6 回多いというずれが偶然よく生じる範囲を超えているかどうかを問わなければならない。**確率**は既知のモデル（公平なコイン）から順方向

に推論して結果の生じる見込みを導くのに対し、**統計**は観測データ（100 回投げて表が 56 回）から逆方向に推論して背後にある過程の性質を推測する。

本節では、コイン投げをシミュレーションしてサンプリングによる揺らぎがどのように現れるかを観察し、標本空間、事象、確率変数、同時分布、独立性を導入したうえで、期待値と分散から**標準誤差**（standard error）$\sigma/\sqrt{n}$ を導出して先ほどのコインの疑問に決着をつける。続いて、2 段階の医療診断検査を題材に、条件付き確率とベイズの定理を展開する。

```
import random
```

1.6.1 サンプリング

1.6.1.1 コイン投げのシミュレーション

公平なコインでは、表と裏の対称性から $P(\text{heads}) = 0.5$ となる。確率過程（random process）から実現値を引き出す操作を**サンプリング**と呼ぶ。以下では 100 回のコイン投げをシミュレーションする。まず区間 $[0, 1)$ からの一様乱数が 0.5 を超えるかどうかを判定する方法を用い、次にフレームワークの**多項**（multinomial）サンプラを $n = 100$、確率 $(0.5, 0.5)$ として用いる。

```
num_tosses = 100
heads = sum([random.random() > 0.5 for _ in range(num_tosses)])
tails = num_tosses - heads
print("heads, tails: ", [heads, tails])
```

```
heads, tails:  [56, 44]
```

JAX では、実行するたびに新しいカウントが得られるよう、`random` をシードとする明示的な PRNG キーを分割する。

```
key = jax.random.key(random.getrandbits(32))  # A new seed on each run
key, subkey = jax.random.split(key)
fair_probs = jnp.array([0.5, 0.5])
jax.random.multinomial(subkey, 100, fair_probs)
```

```
Array([48., 52.], dtype=float32)
```

1.6.1.2 頻度と確率

観測されたカウントを n で割ると経験**頻度**（frequency）$\hat{p}$ が得られる。これは真の確率 $p = P(\text{heads}) = 0.5$ の**推定量**（estimator）である。100 回投げの実験を同時に 5 回繰り返すと、経験頻度を並べた 5×2 行列が得られ、その値は 0.5 の周りで目立って揺らいでいる。

```
key, subkey = jax.random.split(key)
jax.random.multinomial(subkey, 100, fair_probs, shape=(5,)) / 100
```

```
Array([[0.51      , 0.48999998],
       [0.5       , 0.5       ],
       [0.48      , 0.52      ],
       [0.45      , 0.55      ],
       [0.47      , 0.53      ]], dtype=float32)
```

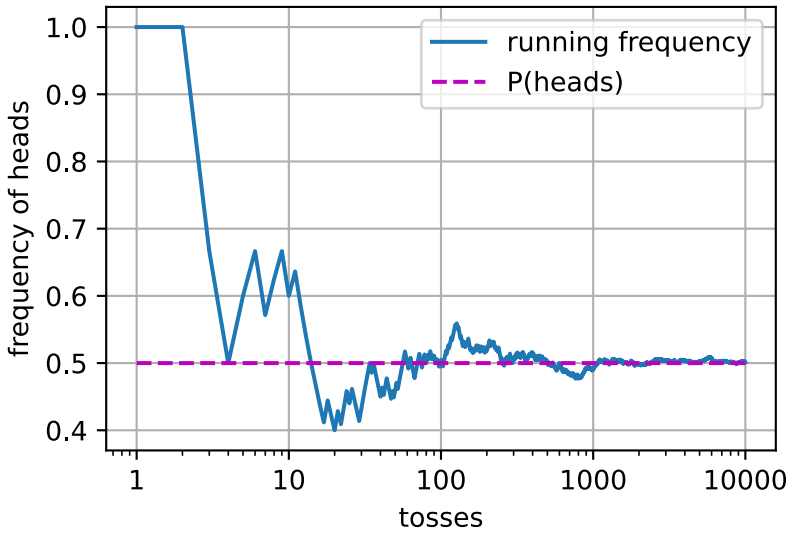
サンプルサイズを 100 倍の $n = 10{,}000$ 回に増やすと、典型的な標本誤差（sampling error）は $\sqrt{100} = 10$ 分の 1 に縮まり、表と裏の経験頻度は $(0.5, 0.5)$ にはるかに近づく。

```
key, subkey = jax.random.split(key)
jax.random.multinomial(subkey, 10000, fair_probs) / 10000
```

```
Array([0.4965, 0.5035], dtype=float32)
```

1 回目から 10,000 回目まで、1 回投げるごとにその時点までの逐次頻度を `cumsum` で追跡すると、**大数の法則**（law of large numbers、36.5 節）を見て取れる。この法則は、$\hat{p} \to p$ という収束が $n \to \infty$ の極限で成り立つことを保証する。

```
n = 10000
key, subkey = jax.random.split(key)
tosses = jax.random.uniform(subkey, (n,)) < 0.5  # True is heads
estimates = tosses.cumsum() / jnp.arange(1, n + 1)
d2l.plot(jnp.arange(1, n + 1), [estimates, jnp.full(n, 0.5)], 'tosses',
         'frequency of heads', legend=['running frequency', 'P(heads)'],
         xscale='log', figsize=(4.5, 3))
```



どれほどの速さで $\hat{p}$ が収束するのか、そして $56/100$ という結果が疑わしいのかどうかを知るために、ここで確率変数とそのばらつきを定式化しよう。

1.6.2 事象と確率変数

1.6.2.1 標本空間と事象

ランダムな試行において起こりうるすべての結果からなる集合を**標本空間**（sample space）$\mathcal{S}$ と呼ぶ（サイコロを 1 回振る場合、$\mathcal{S} = \{1, 2, 3, 4, 5, 6\}$ である）。**事象**（event）とは任意の部分集合 $\mathcal{A} \subseteq \mathcal{S}$ のことである（たとえば「偶数の目が出る」という $\mathcal{A} = \{2, 4, 6\}$ や、「5 以上の目が出る」という $\mathcal{B} = \{5, 6\}$ など）。**確率**関数 P は、Kolmogorov の 3 つの公理を満たすように、すべての事象 $\mathcal{A} \subseteq \mathcal{S}$ に実数を割り当てる (Kolmogorov 1933)。

- **非負性**：$P(\mathcal{A}) \geq 0$
- **正規化**：$P(\mathcal{S}) = 1$
- **可算加法性**：互いに排反な事象 $\mathcal{A}_1, \mathcal{A}_2, \ldots$（$\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$、ただし $i \neq j$）に対して $P(\bigcup_i \mathcal{A}_i) = \sum_i P(\mathcal{A}_i)$ が成り立つ。

これらの公理から、余事象の公式 $P(\mathcal{A}') = 1 - P(\mathcal{A})$ や $P(\emptyset) = 0$、そして次の包除原理（inclusion-exclusion formula、Figure 1.31）が導かれる。

$$P(\mathcal{A} \cup \mathcal{B}) = P(\mathcal{A}) + P(\mathcal{B}) - P(\mathcal{A} \cap \mathcal{B}). \tag{1.26}$$

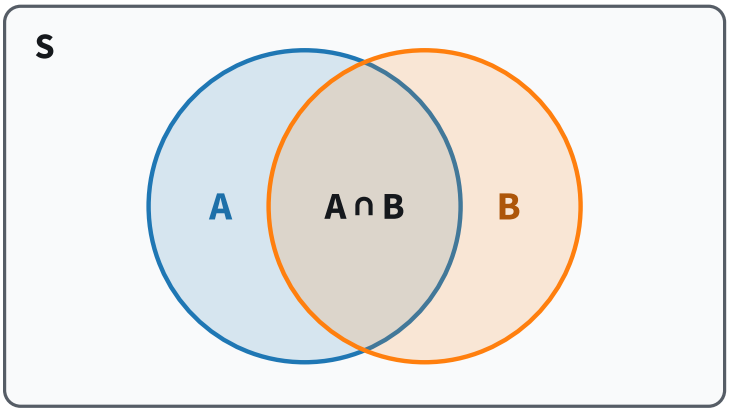


Figure 1.31: 2 つの事象 $\mathcal{A}$ と $\mathcal{B}$（標本空間 $\mathcal{S}$ 内）。和 $P(\mathcal{A}) + P(\mathcal{B})$ は共通部分 $\mathcal{A} \cap \mathcal{B}$ を 2 回数えている。

1.6.2.2 確率変数と分布

確率変数（random variable）X は、$\mathcal{S}$ 内の結果を数値に写す（たとえば 100 回のコイン投げで表が出る回数は、2^{100} 通りの系列を $\{0, \ldots, 100\}$ に写す）。**離散**（discrete）確率変数は、確率質量 $P(X = v)$ を各値 v に割り当てる（これをその**分布**（distribution）$P(X)$ と呼ぶ）。一方、**連続**（continuous）確率変数（人の正確な身長など）は**確率密度**（probability density）$p(x)$ によって記述され（Figure 1.32）、値が区間 $[a, b]$ に入る確率は密度曲線の下の面積 $P(a \leq X \leq b) = \int_a^b p(x)\,dx$ で与えられる（36.1 節）。

1.6.2.3 同時分布、周辺分布、独立な分布

2 つの確率変数の**同時分布**（joint distribution）$P(X = x, Y = y)$ は、$X = x$ と $Y = y$ が同時に起こる確率を与える。たとえば、ある無作為な日における雨の有無（X）と傘の携帯（Y）について次の表を考える。

$P(X,Y)$	傘あり	傘なし
雨	0.25	0.05
雨なし	0.10	0.60

一方の変数について和をとって消去すると、もう一方の変数の**周辺分布**（marginal distribution）が得られる。この操作を**周辺化**（marginalization）と呼ぶ（Figure 1.33）。

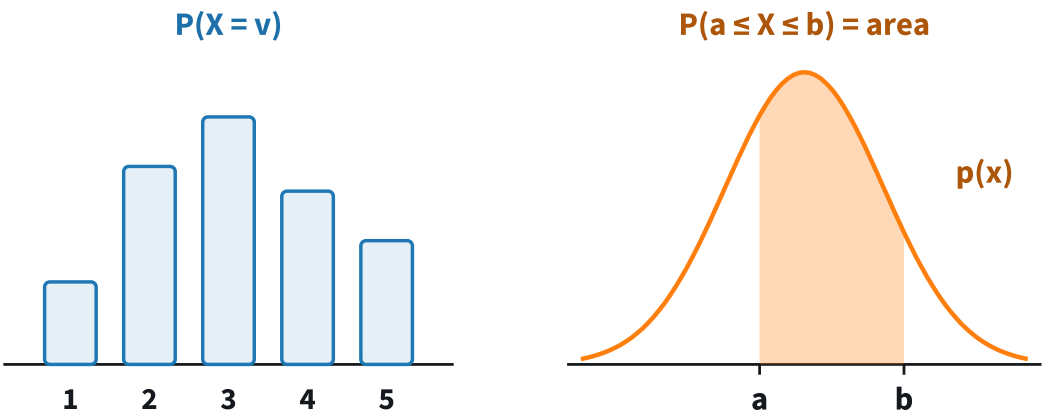


Figure 1.32: 離散確率変数は個々の値に確率を割り当てる（左）。連続確率変数は密度を持ち、ある区間の確率はその密度の下の面積となる（右）。

$$P(X = x) = \sum_{y} P(X = x, Y = y), \tag{1.27}$$

したがって、$P(\text{rain}) = 0.25 + 0.05 = 0.30$ および $P(\text{umbrella}) = 0.25 + 0.10 = 0.35$ となる。

2 つの確率変数 X と Y は、次の式のように同時分布がそれぞれの周辺分布の積に分解できるとき**独立**（independent）である（$X \perp Y$）。

$$P(X, Y) = P(X)P(Y). \tag{1.28}$$

連続するコイン投げの結果は互いに独立であるが、雨と傘は独立ではない（$0.25 \neq 0.30 \cdot 0.35 = 0.105$）。

1.6.3 期待値と分散

1.6.3.1 期待値

離散確率変数 X の**期待値**（expectation）または**平均**は、確率で重み付けした平均として次のように定義される。

$$\mathbb{E}[X] = \sum_{x} x\, P(X = x), \tag{1.29}$$

また、任意の関数 f に対して $\mathbb{E}[f(X)] = \sum_x f(x)\, P(X = x)$ が成り立つ。たとえば、収益が $0\times$ になる確率が 0.5、$2\times$ になる確率が 0.4、$10\times$ になる確率が 0.1 の投資では、$\mathbb{E}[X] = 0.5(0) + 0.4(2) + 0.1(10) = 1.8$ となる。コイン投げの指示変数（indicator）$X = \mathbf{1}(\text{heads}) \in \{0, 1\}$ については、$\mathbb{E}[X] = 1 \cdot p + 0 \cdot (1 - p) = p$ である。

期待値は常に**線形**（linear）である。すなわち、$\mathbb{E}[aX + bY] = a\,\mathbb{E}[X] + b\,\mathbb{E}[Y]$ は X と Y が独立であるかどうかにかかわらず成り立つ。したがって、標本頻度 $\hat{p} = \frac{1}{n}\sum_{i=1}^{n} X_i$ は $\mathbb{E}[\hat{p}] = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[X_i] = p$ を満たし、$\hat{p}$ は p の**不偏**（unbiased）推定量となる。

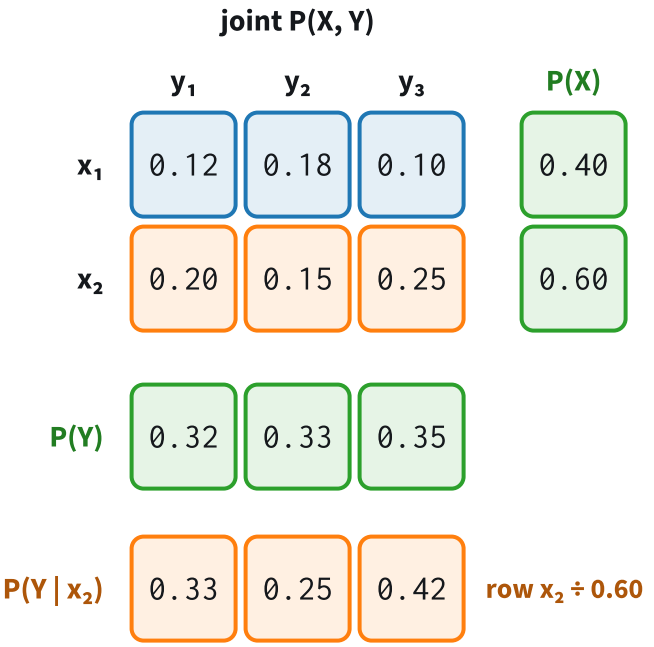


Figure 1.33: 表として表した同時分布 $P(X,Y)$。行または列の和をとると周辺分布が得られる。行をその行の和で割ると条件付き分布（ここでは $P(Y \mid X = x_2)$）が得られる。

1.6.3.2 分散と独立な変数の和

$\mu = \mathbb{E}[X]$ の周りのばらつきを測るため、**分散**（variance）は偏差の二乗の平均をとり、**標準偏差**（standard deviation）$\sigma = \sqrt{\mathrm{Var}(X)}$ は元の単位に戻す。

$$\mathrm{Var}(X) = \mathbb{E}\left[(X-\mu)^2\right] = \mathbb{E}[X^2] - \mu^2, \qquad \mathrm{Var}(aX+b) = a^2\,\mathrm{Var}(X). \tag{1.30}$$

先ほどの投資の例では、$\mathbb{E}[X^2] = 0.5(0) + 0.4(4) + 0.1(100) = 11.6$ であるから、$\mathrm{Var}(X) = 11.6 - 1.8^2 = 8.36$（$\sigma \approx 2.9$）となる。1 回のコイン投げの指示変数 $X \in \{0,1\}$ では $X^2 = X$ であるため、$\mathrm{Var}(X) = p - p^2 = p(1-p)$ となり、最大値 $1/4$ を $p = 0.5$ のときにとる。1.2 節のように n 個の観測値から標本平均（sample mean）の周りの分散を推定する際には、$n-1$ で割る**ベッセル補正**（Bessel's correction）によって、n で割る際に生じるわずかな下方向の統計的バイアス（bias）を補正する。

2 つの確率変数の和について、$\mathbb{E}[(X-\mu_X + Y-\mu_Y)^2]$ を展開すると次式が得られる。

$$\begin{aligned}\mathrm{Var}(X+Y) &= \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\,\mathrm{Cov}(X,Y),\\ \mathrm{Cov}(X,Y) &= \mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right],\end{aligned}$$

ここで、**共分散**（covariance）$\mathrm{Cov}(X,Y)$ と無次元の**相関**（correlation）$\rho = \mathrm{Cov}(X,Y)/(\sigma_X\sigma_Y) \in [-1,1]$ は、線形な連動性を測る指標である（Figure 1.34）。より一般に、確率ベクトル（random vector）$\mathbf{x} \in \mathbb{R}^d$（平均 $\mu = \mathbb{E}[\mathbf{x}]$）に対して、すべての対の共分散 $\Sigma_{ij} = \mathrm{Cov}(x_i, x_j)$ を集めると対称な $d \times d$ の**共分散行列**（covariance matrix）$\Sigma = \mathbb{E}[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^\top]$ が構成され、任意の線形結合 $\mathbf{v}^\top\mathbf{x}$ の分散は $\mathrm{Var}(\mathbf{v}^\top\mathbf{x}) = \mathbf{v}^\top\Sigma\mathbf{v}$ となる。

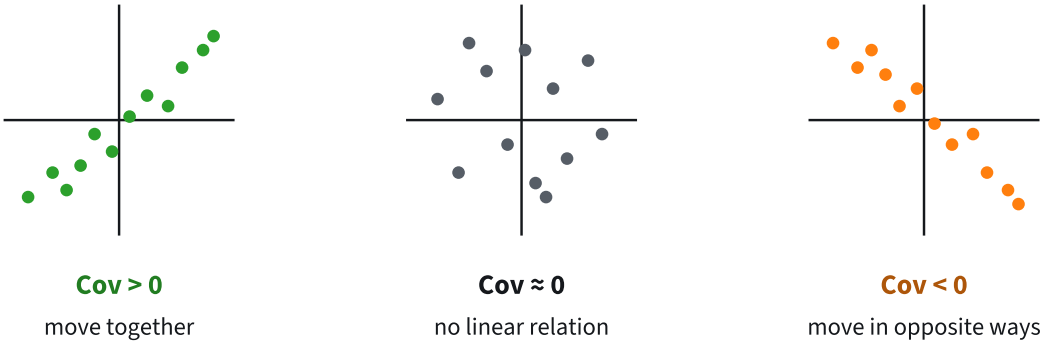


Figure 1.34: 正、ほぼゼロ、負の共分散を持つ 3 組の確率変数のサンプル。

X と Y が**独立**であるとき、期待値は積に分解されるため $\mathrm{Cov}(X,Y) = \mathbb{E}[X-\mu_X]\mathbb{E}[Y-\mu_Y] = 0$ となり、**独立な変数の分散は足し合わされる**。

$$\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y). \tag{1.31}$$

しかし、共分散がゼロであることは独立性を意味しない。たとえば X が $-1, 0, +1$ を等確率でとり $Y = X^2$ とすると、Y は X によって完全に決まるが、対称性により $\mathbb{E}[X] = 0$ かつ $\mathbb{E}[XY] = \mathbb{E}[X^3] = 0$ となるため、$\mathrm{Cov}(X,Y) = 0$ となる。

1.6.4 推定誤差

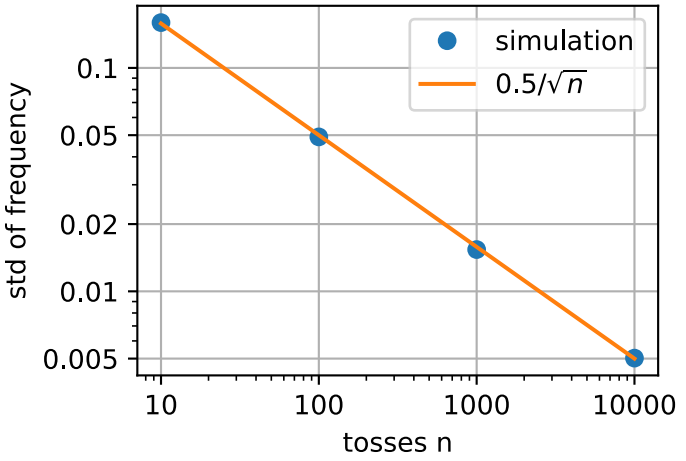
1.6.4.1 頻度の標準誤差

独立な n 回のコイン投げはそれぞれ分散 $p(1-p)$ を持つため、その和の分散は $np(1-p)$ となり、さらに n で割ると分散は $1/n^2$ 倍になる。

$$\mathrm{Var}(\hat{p}) = \frac{p(1-p)}{n}. \tag{1.32}$$

平方根をとると $\hat{p}$ の**標準誤差** $\sqrt{p(1-p)/n}$ が得られる。公平なコイン（$p = 0.5$）では $0.5/\sqrt{n}$ となり、その値は 0.05（$n = 100$ のとき）、0.005（$n = 10{,}000$ のとき）である。$1/\sqrt{n}$ の速さで減少するため、誤差を半分にするには $4\times$ のデータが必要になる。より一般に、独立な n 個のサンプルがそれぞれ分散 σ^2 を持つとき、その平均の標準誤差は $\sigma/\sqrt{n}$ となる。1000 回の反復を各 $n \in \{10, 100, 1000, 10000\}$ でシミュレーションすると、$0.5/\sqrt{n}$ が表す傾き $-1/2$ の直線が確認できる。

```
ns = [10, 100, 1000, 10000]
key, *subkeys = jax.random.split(key, len(ns) + 1)
stds = [float((jax.random.uniform(k, (1000, n)) < 0.5).mean(axis=1).std())
        for k, n in zip(subkeys, ns)]
d2l.plot(ns, [stds, [0.5 / n ** 0.5 for n in ns]], 'tosses n',
         'std of frequency', legend=['simulation', r'$0.5/\sqrt{n}$'],
         xscale='log', yscale='log', fmts=('o', '-'))
```



1.6.4.2 偏りと不確実性の検定

これで最初のコインの疑問に答えることができる。100 回投げでは、$\hat{p} = 0.56$ は標準誤差の $0.06/0.05 = 1.2$ 倍だけ 0.5 から離れているにすぎないが、10,000 回投げでは同じ頻度 $\hat{p} = 0.56$（表が 5,600 回）でも標準誤差の $0.06/0.005 = 12$ 倍も離れている。公平なコインの実験を 1000 回シミュレーションすると、公平なコインが少なくともこれだけ大きく外れる頻度（**p 値**（p-value）の推定値）がわかる。

```
def fraction_as_far(key, heads, n, num_experiments=1000):
    """Fraction of fair-coin experiments at least as far from n/2 as heads."""
    counts = (jax.random.uniform(key, (num_experiments, n)) < 0.5).sum(axis=1)
    far = jnp.abs(counts - n / 2) >= abs(heads - n / 2)
    return far.sum().item() / num_experiments

key, k1, k2 = jax.random.split(key, 3)
fraction_as_far(k1, 56, 100), fraction_as_far(k2, 5600, 10000)
```

```
(0.291, 0.0)
```

公平なコインを 100 回投げる試行の約 4 分の 1 では表の回数が 6 回以上ずれるため、$56/100$ は偏りの証拠にはならない。一方、公平なコインを 10,000 回投げる試行では、1000 回中 1 回も 5,600 の近くには達しない。こうした裾確率は 2 つの一般的な定理によって定量化される。**チェビシェフの不等式**（Chebyshev's inequality、36.1 節）は分散が有限である任意の分布に対して $P(|X - \mu| \geq k\sigma) \leq 1/k^2$ を保証し、**中心極限定理**（central limit theorem、36.3 節）は独立なサンプルの平均がガウス分布の釣鐘曲線に近づき、約 95% の確率で標準誤差の 2 倍以内に収まることを示す。同じ計算はモデルの評価にも当てはまる。真の正解率が 0.9 の分類器では、画像 256 枚の単一ミニバッチにおける標準誤差は $\sqrt{0.9 \cdot 0.1/256} \approx 0.019$（したがって $\pm 2\%$ の変動は通常のノイズである）であるのに対し、画像 10,000 枚のテストセット全体では 0.003 となる。

最後に、データを多く集めれば**認識的不確実性**（epistemic uncertainty、すなわちモデルパラメータ p に関する不確実性であり $1/\sqrt{n}$ で減衰する）は縮小するが、**偶然的不確実性**（aleatoric uncertainty、すなわち次のコイン投げそのものが持つ本質的な 50/50 のランダム性や、そもそも曖昧な画像）は決して減らないことに注意しよう (Der Kiureghian and Ditlevsen 2009)。

1.6.5 条件付き確率

1.6.5.1 条件付け、ベイズの定理、条件付き独立

対象を $X = x$（$P(X = x) > 0$）となる結果に限定して再正規化することで、$X = x$ が与えられたもとでの $Y = y$ の**条件付き確率**（conditional probability）が次のように定義される。

$$P(Y = y \mid X = x) = \frac{P(X = x, Y = y)}{P(X = x)} \quad \Longrightarrow \quad P(X, Y) = P(Y \mid X)P(X). \tag{1.33}$$

先ほどの天気の表において、雨という条件を付けると、雨の行 $(0.25, 0.05)$ を 0.30 で割り直すことになり、$P(\text{umbrella} \mid \text{rain}) = 0.25/0.30 \approx 0.83$ が得られる（条件なしでは 0.35）。同時確率（joint probability）を 2 通りの順序で $P(X, Y) = P(Y \mid X)P(X) = P(X \mid Y)P(Y)$ と書き、$P(Y) = \sum_x P(Y \mid X = x)P(X = x)$ で割ると、次の**ベイズの定理**（Bayes' theorem）が得られる (Bayes 1763)。

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)}. \tag{1.34}$$

仮説 H と観測データ D に対して、ベイズの定理は**事前分布**（prior）$P(H)$ を**尤度**（likelihood）$P(D \mid H)$ とエビデンス $P(D)$ によって更新し、**事後分布**（posterior）$P(H \mid D) \propto P(D \mid H)P(H)$ を導く。

2 つの変数 X と Y は、Z が与えられたもとで（$X \perp Y \mid Z$）、$P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z)$ を満たすとき**条件付き独立**（conditionally independent）であるという。条件付けは従属性を取り除くことも、新たに生み出すこともある（Figure 1.35）。たとえば**共通原因**（common cause）Z（子どもの年齢など）を観測すると靴のサイズと読解力の相関は消え、現在の状態 Y を**マルコフ連鎖**（Markov chain）$X \to Y \to Z$ において観測すると過去の X と未来の Z は独立になる。逆に、**共通結果**（common effect）Z で条件付けると（**説明し尽くし**、explaining away）、独立だった 2 つの原因が結びつく。たとえば入院患者が骨折しているとわかれば、肺がんで入院した確率は下がる。

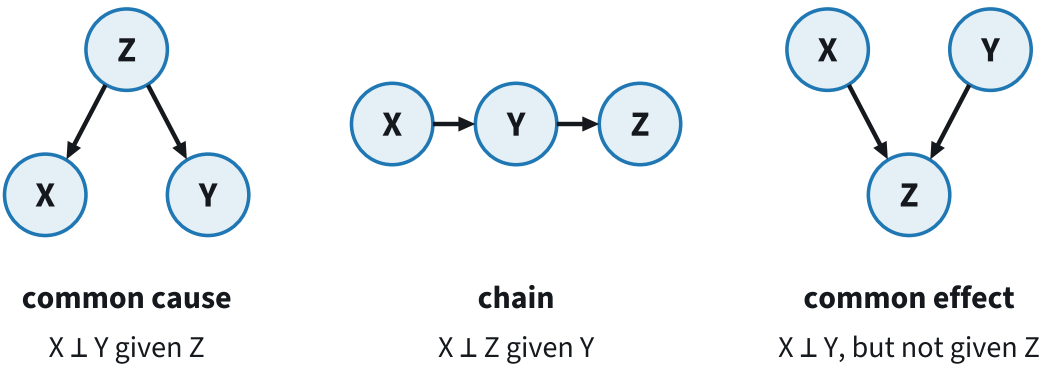


Figure 1.35: 条件付けは従属性を変化させる。共通原因 Z は X と Y を従属させるが、Z を観測するとその従属性は消える。連鎖構造では、X と Z は Y が与えられると独立になる。共通結果 Z は、Z が観測されると独立な原因どうしを従属させる。

1.6.5.2 例：診断検査

ある患者が HIV スクリーニング検査 $D_1 \in \{0,1\}$ を受け、真の HIV 感染状態が $H \in \{0,1\}$ であるとする。この検査は感度（sensitivity）が 100%、偽陽性率（false-positive rate）が 1% である。

$P(D_1 \mid H)$	$H=1$（HIV）	$H=0$（健康）
検査陽性（$D_1=1$）	1.00	0.01
検査陰性（$D_1=0$）	0.00	0.99

集団における有病率（prevalence）が $P(H=1)=0.0015$（$P(H=0)=0.9985$）であり、患者の検査結果が陽性（$D_1=1$）であったとすると、周辺化により $P(D_1=1)=0.01(0.9985)+1(0.0015)=0.011485$ となり、ベイズの定理から次式が得られる。

$$P(H=1 \mid D_1=1)=\frac{P(D_1=1 \mid H=1)P(H=1)}{P(D_1=1)}=\frac{0.0015}{0.011485}=0.1306. \quad (1.35)$$

10,000 人の集団（Figure 1.36）を考えると、感染している 15 人全員が陽性となる一方で、健康な人の 1%（9,985 人中 ≈ 100 件の誤報）も陽性となるため、陽性結果のうち本当に感染している割合は $15/115 \approx 13\%$ にとどまる。

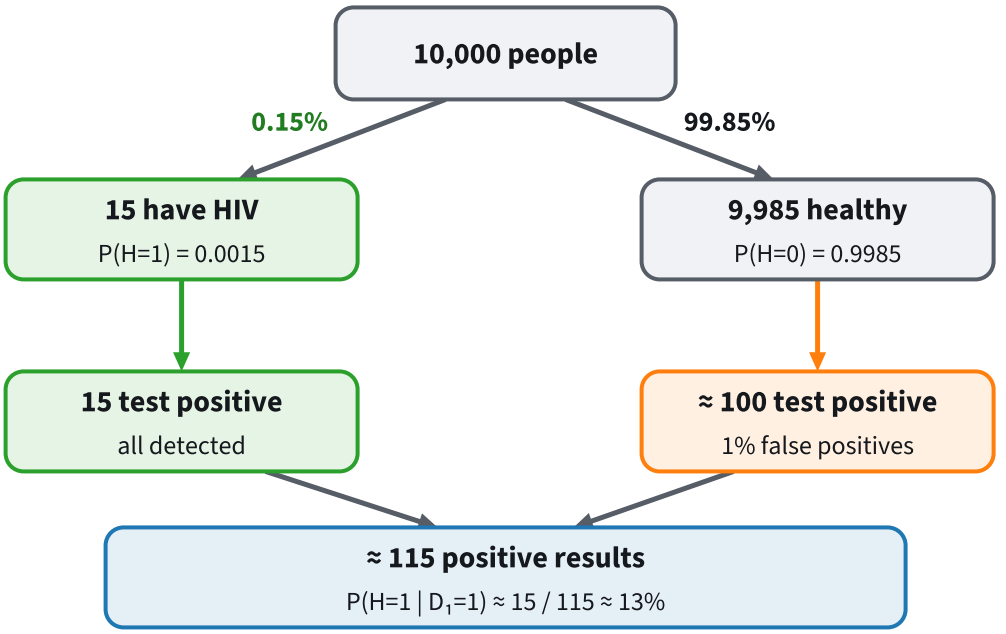


Figure 1.36: 10,000 人に HIV 検査を実施した例。感染者 15 人は全員陽性となるが、健康な 9,985 人のうち約 100 人も陽性となるため、約 115 件の陽性結果のうち正しいのは 15 件にすぎない。

ここで、追加検査 D_2（感度 $P(D_2=1 \mid H=1)=0.98$、偽陽性率 $P(D_2=1 \mid H=0)=0.03$）も陽性になったとする。D_1 と D_2 が H のもとで条件付き独立であると仮定すると、2 回の検査がともに偽陽性となる確率は $0.01 \cdot 0.03 = 0.0003$ に下がる一方、感度は $1 \cdot 0.98 = 0.98$ と

なる。周辺化により $P(D_1 = 1, D_2 = 1) = 0.0003(0.9985) + 0.98(0.0015) = 0.00176955$ が得られ、ベイズの定理によって感染の事後確率（posterior probability、Figure 1.37）は次の値まで上昇する。

$$P(H = 1 \mid D_1 = 1, D_2 = 1) = \frac{0.98 \cdot 0.0015}{0.00176955} = 0.8307. \tag{1.36}$$

仮想的な患者 200 万人をシミュレーションすると、経験的な割合は 0.82–0.83 前後となり、標準誤差 ± 0.006（$\sqrt{0.83 \cdot 0.17/3500}$）の 2 倍の範囲内で厳密値 0.8307 と一致する（両方陽性となる患者は $\approx 3{,}500$ 人である）。

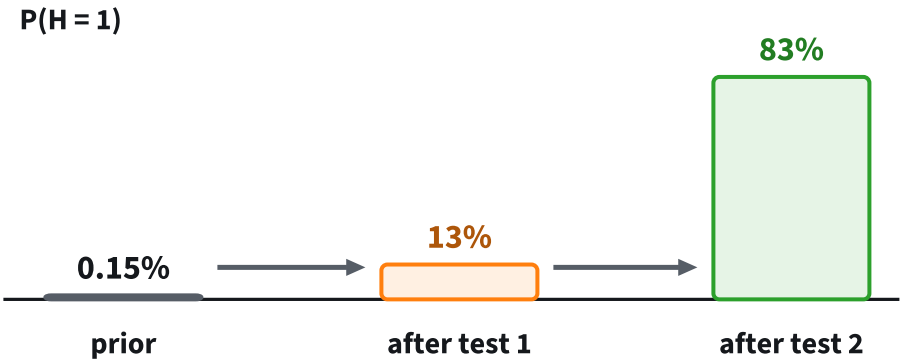


Figure 1.37: 患者が HIV に感染している確率。事前確率、1 回の陽性判定後、および条件付き独立な 2 回の陽性判定後。

```
n = 2000000
key, k1, k2, k3 = jax.random.split(key, 4)
H = jax.random.uniform(k1, (n,)) < 0.0015
D1 = jax.random.uniform(k2, (n,)) < jnp.where(H, 1.00, 0.01)
D2 = jax.random.uniform(k3, (n,)) < jnp.where(H, 0.98, 0.03)
H[D1 & D2].mean()  # Exact value: 0.8307
```

```
Array(0.82047725, dtype=float32)
```

1.6.6 まとめ

確率はモデルから順方向に推論して結果の生じる見込みを導くのに対し、統計は観測データから逆方向に推論して背後にある過程の性質を推測する。Kolmogorov の 3 つの公理に基づく同時分布 $P(X, Y)$ は、変数の和をとって消去することで周辺化でき、変数が独立なときには $P(X)P(Y)$ と積に分解される。期待値 $\mathbb{E}[X]$ は常に線形であり、分散 $\mathrm{Var}(X) = \mathbb{E}[(X - \mu)^2]$ は独立な変数どうしで足し合わされるため、独立な n 個のサンプルの平均は標準誤差 $\sigma/\sqrt{n}$ を持つ。偏差を標準誤差の単位で測ることで、表が $56/100$ 回（標準誤差の 1.2 倍）出るのは日常的な偶然の範囲内である一方、$5{,}600/10{,}000$ 回（標準誤差の 12 倍）は決定的な証拠となる理由、そしてデータの追加収集によって**認識的**不確実性が $1/\sqrt{n}$ の速さで縮小する一方、**偶然的**不確実性は変わらない理由が明らかになる。最後に、ベイズの定理 $P(H \mid D) = P(D \mid H)P(H)/P(D)$ はエビデンスに照らして事前の信念を更新するものであり、まれな疾患（有病率 0.15%、偽陽性率 1%）に対する 1 回の陽性判定では事後確率が 13.1% にとどまり、条件付き独立な 2 回目の検査で陽性となって初めて 83.1% に達する理由を説明する。

1.6.7 演習問題

1. **ポートフォリオの収益とリスク**。n 個の銘柄からなるポートフォリオを運用しており、各銘柄の収益 $\mathbf{s} \in \mathbb{R}^n$ は平均 $\mu = \mathbb{E}[\mathbf{s}]$、共分散行列 Σ を持つ確率変数であるとする。資金の割合 $\alpha_i \geq 0$ を銘柄 i に投資し、$\sum_i \alpha_i = 1$ とする。
   1. ポートフォリオ α の期待収益と収益の分散を計算せよ。
   2. すべての銘柄が分散 σ^2 を持ち、異なる銘柄のすべての対が相関 $\rho \geq 0$ を持つと仮定する。等重みポートフォリオ $\alpha_i = 1/n$ の分散と、その $n \to \infty$ における極限を計算せよ。また、$\rho = 0$ の場合を Equation 1.32 と比較せよ。
   3. （発展）分散をある上界以下に制約しつつ期待収益を最大化する最適化問題を定式化せよ。これは Markowitz のポートフォリオ問題である (Markowitz 1952)。この種の制約付き問題に対する手法は 35.6 節で展開する。

2. [code] **コイン投げ推定量に対するチェビシェフの不等式**。 Equation 1.32 は、表の頻度 $\hat{p}$（n 回投げた後）の分散を与える。
   1. 公平なコインで $n = 100$ のとき、チェビシェフの不等式を用いて $P(|\hat{p} - p| \geq 0.1)$ の上界を求めよ。
   2. 100 回投げの実験を 10,000 回反復するシミュレーションによって同じ確率を推定し、求めた上界と比較したうえで、その差が生じる理由を説明せよ。
   3. 公平なコインに対して $P(|\hat{p} - p| \geq 0.01) \leq 0.05$ を保証するためにチェビシェフの不等式が必要とする試行回数を求め、n のうち標準誤差 $\sqrt{p(1-p)/n}$ 自体が 0.01 に等しくなる値と比較せよ。

3. **2 つの分類器の比較**。●2 つの分類器を、画像 $n = 10{,}000$ 枚からなる同一のテストセット上で評価する。分類器 A は 9,050 枚の画像に正しいラベルを付け、分類器 B は 9,000 枚に正しいラベルを付けた。テスト画像は独立なサンプルとして扱え。
   1. それぞれの正解率の標準誤差を計算せよ。
   2. 2 つの正解率が独立であると仮定し、その差の標準誤差を計算せよ。0.005 という差はノイズと比べて大きいか。
   3. 2 つの正解率は同じ画像上で測定されているため、実際には独立ではない。300 枚の画像で両分類器の判定が食い違い、そのうち 175 枚では A のみが正解し、125 枚では B のみが正解したとする。画像 i について、A のみが正解なら $Z_i = 1$、B のみが正解なら $Z_i = -1$、それ以外なら $Z_i = 0$ とおくと、正解率の差は Z_i の平均となる。この平均の標準誤差を計算し、前問の答えと比較せよ。
   4. $\mathrm{Var}(X - Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) - 2\,\mathrm{Cov}(X, Y)$ を用いてこの違いを説明せよ。1 枚の画像に対する 2 つの分類器の正誤のあいだの共分散はどの符号を持ち、それはなぜか。

4. **連鎖構造の分解**。確率変数 X、Y、Z において、Figure 1.35 の連鎖構造のように、Z が X に対して Y を与えたもとで条件付き独立であるとする。
   1. 同時確率 $P(X, Y, Z)$ を、それぞれが高々 2 つの変数しか含まない因子の積として表せ。
   2. 系列 $X_1, \ldots, X_T$ において、各 X_t がそれ以前の変数に X_{t-1} を通じてのみ依存する場合へと一般化せよ。すべての X_t が k 通りの値をとるとき、同時分布を定めるのに必要な数値の個数は、そのような T 個の変数の任意の同時分布と比べていくつになるか。

5. **共通結果による条件付け**。独立な 2 枚の公平なコインを投げ、$X, Y \in \{0, 1\}$ をそれぞれで表が出たことを表す指示変数とする。また、$Z = \max(X, Y)$ を少なくとも 1 枚のコインで表が出たことを表す変数とする。$P(X = 1)$、$P(X = 1 \mid Z = 1)$、および

$P(X = 1 \mid Z = 1, Y = 1)$ を計算せよ。これらの値を用いて、X と Y が $Z = 1$ のもとで条件付き独立であるかどうかを判定し、2 番目の値から 3 番目の値への変化を Figure 1.35 と関連付けて説明せよ。

6. **相関のある 2 回目の検査**。●HIV 感染患者に対しては結果が条件付き独立であるが、健康な患者に対しては条件付き独立でない 2 つの新しい検査を用いて、1.6.5.2 節を再考せよ。いずれの検査も単独では偽陽性率が 10%、偽陰性率が 1% であるとする。

$$P(D_i = 1 \mid H = 0) = 0.1, \quad P(D_i = 0 \mid H = 1) = 0.01 \quad \text{for } i = 1, 2. \tag{1.37}$$

$H = 1$ に対しては、同時分布は $P(D_1, D_2 \mid H = 1) = P(D_1 \mid H = 1)\, P(D_2 \mid H = 1)$ と積に分解される。$H = 0$ に対しては、結果は $P(D_1 = D_2 = 1 \mid H = 0) = 0.02$ によって連動している。基準となる事前確率は先ほどと同じく $P(H = 1) = 0.0015$ である。

   1. D_1 と D_2 の（$H = 0$ のもとでの）同時確率表を求めよ。
   2. 1 回目の検査が陽性となり、2 回目の検査を実施する前の時点で、患者が HIV に感染している確率を導出せよ。
   3. 両方の検査が陽性となった後に、患者が HIV に感染している確率を導出せよ。
   4. $H = 0$ のもとでも条件付き独立である検査について前問の計算を繰り返し、2 つの事後確率が異なる理由を説明せよ。

7. **弁護人の統計**。ある容疑者には、殺害された妻を過去に虐待していた記録があった。弁護人は、統計的に妻を虐待する夫のうち実際に妻を殺害するに至るのは 1,000 人に 1 人にすぎないため、虐待歴は弱い証拠でしかなく陪審員は無罪とすべきだと主張している。「容疑者が妻を殺害した」を M、「妻が殺害された」を K、「容疑者に妻への虐待歴があった」を B と表す。

   1. 弁護人が挙げた 1,000 人に 1 人という数値と、有罪の事後確率をこれらの記号で表し、弁護人がどの確率とどの確率を取り違えているかを特定せよ。
   2. 弁護人の統計が対象とする期間において、虐待を受けている妻が夫以外の誰かに殺害される確率が 10^{-5} であると仮定する。有罪の事後確率を計算し、弁護人の数値と比較せよ。

   MIT 18.05、問題セット 7、問題 4 を改変。

Chapter 2

線形回帰

回帰（regression）は、床面積と築年数からの住宅販売価格の推定、バイタルサインからの患者の入院期間の予測、次の四半期の電力需要の予測のように、対象が離散的なカテゴリではなく意味のある数値であるあらゆる場面で、定量的な予測の主力となる。また、深層学習にとっても理想的な検証の場である。ニューラルネットワークの訓練を構成する要素、すなわちパラメータ化された順伝播、最尤法に基づく損失関数、ミニバッチ確率的勾配降下法、再利用可能な訓練ハーネス、汎化の診断、そして重みノルムによる正則化のほぼすべてが、**線形回帰**（linear regression）において最も単純で見通しのよい形で、すでに現れている。

まず 2.1 節では、アフィンモデルと二乗誤差損失を定式化し、直交射影による閉形式の解法とミニバッチ確率的勾配降下法（SGD）による反復的な解法の両方でこの問題を解いたうえで、二乗損失をガウス分布の下での最尤法と結びつける。実データを扱う前に、2.2 節では既知の正解パラメータ $(\mathbf{w}^*, b^*)$ とノイズレベル σ をもつ合成データのベンチマークを構築し、コードを検証するための厳密な基準を用意する。続く 3 つの節では、訓練パイプラインを段階的に構築していく。まず生のテンソルと自動微分（automatic differentiation）のみから実装し（2.3 節）、次に本書の以降の章を支える再利用可能な `Module`、`DataModule`、`Trainer` クラスへと整理し（2.4 節）、最後にフレームワークの高水準な層とオプティマイザを用いてわずか数行で表現する（2.5 節）。

訓練データへの当てはめは、問題の半分にすぎない。2.6 節では多項式曲線フィッティングを題材に、訓練誤差と汎化誤差（generalization error）を切り分け、期待テスト誤差を統計的バイアスの二乗、分散、ノイズへと分解したうえで、テストデータを覗き見ることなく検証データへの分割や K 分割交差検証（cross-validation）によってモデルの複雑さを選択する方法を示す。最後に 2.7 節では、二乗 ℓ_2 ペナルティを加えることで高次元における過学習（overfitting）を抑え、重み減衰（weight decay）が訓練入力からは十分に定まらない方向を選択的に縮小させる仕組みを明らかにする。

2.1　線形回帰

住宅の床面積と築年数から売却価格を見積もりたいとしよう。入力を離散的なカテゴリに割り当てる分類（classification）とは異なり、**回帰**（regression）は連続的な数値を予測する。定量的入力を数値出力に写す規則を求める試みには、2 世紀にわたる輝かしい歴史がある。1805 年、Adrien-Marie Legendre はノイズを含む天体観測から彗星の軌道を決定するために**最小二乗法**（method of least squares）を発表した (Legendre 1805)。その 4 年後、Carl Friedrich Gauss は、どのような測定誤差の分布のもとで最小二乗法による当てはめが最も確からしくなるかを問うことで最小二乗法を確率論で基礎づけ、ガウス分布に到達した (Gauss 1809)（もっとも Gauss 自身

は、新発見の準惑星ケレスが太陽のまぶしさの陰に消えた後、その位置を再特定するために 1801 年にはすでにこの手法を用いていた)。

今日、回帰は臨床リスクのスコアリングや需要予測から物体検出器のバウンディングボックスヘッドに至るまで、あらゆる応用を支えている。さらに都合のよいことに、予測が入力のアフィン結合となる**線形回帰**（linear regression）には、手計算で完全に解析できるほど単純な設定のなかに、深層学習のほぼすべての中核概念が詰まっている。計算グラフ（computational graph）として見れば、線形回帰は 1 個の人工ニューロン、すなわち出力ユニットを 1 つ持つ**全結合層**（fully connected layer）である。本節では、モデルと二乗損失（squared loss）を定式化し、正規方程式（normal equations）による解析的な解法と（深層学習の主力オプティマイザである）**ミニバッチ確率的勾配降下法**（minibatch stochastic gradient descent）による反復的な解法の両方でこれを解く。さらに、ベクトル化された配列演算と Python のループの速度を比較し、異なるノイズ分布のもとでの最尤推定（maximum likelihood estimation）へと損失関数（loss function）を結びつける。

```
import time
import jax
from jax import numpy as jnp
import numpy as np
```

2.1.1 モデルと損失

2.1.1.1 特徴量とラベル

住宅の面積（平方メートル単位）と築年数（年単位）から売却価格を推定するために、過去の売買記録からなる**訓練データセット**（training dataset）を集める。その記録に含まれる各住宅が 1 つの**例**（example。**データ点**、**サンプル**、**インスタンス**とも呼ばれる）である。予測したい量（ここでは価格）が**ラベル**（または**目的変数**、target）であり、予測の根拠となる測定可能な属性（面積と築年数）が**特徴量**（feature。または**共変量**、covariate）である。

データセットに n 個の例が含まれるとしよう。これらに $i \in \{1, \dots, n\}$ で添字を付け、$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})^\top \in \mathbb{R}^2$ で i^{th} の住宅の特徴ベクトル（feature vector）を、$y^{(i)} \in \mathbb{R}$ でその観測された売却価格を表す。より一般に、各例が d 個の特徴量で記述されるとき $\mathbf{x}^{(i)} \in \mathbb{R}^d$ となり、データセット全体は対の集合 $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$ となる。

2.1.1.2 線形モデル

あらゆる教師あり学習（supervised learning）の手法は、特徴量を予測（prediction）へと写すパラメータ化された関数族、すなわち**モデルクラス**から出発する。線形の仮定では、目的変数を特徴量の重み付き和にオフセットを加えた形で近似できるとみなす。

$$\text{price} = w_{\text{area}} \cdot \text{area} + w_{\text{age}} \cdot \text{age} + b. \tag{2.1}$$

ここで、w_{area} と w_{age} は**重み**（weight）であり、b は**バイアス**（**オフセット**、あるいは**切片**、intercept とも呼ばれる）である。各重みはその特徴量の限界的な影響度（築年数を固定したとき、床面積が 1 平方メートル増えるごとに予測価格がどれだけ変化するか）を定め、一方バイアス b はすべての特徴量が 0 のときの予測値を決める。現実には面積が 0 の住宅は存在しなくても、b を残しておくことは不可欠である。これがないと当てはめる平面は必ず原点を通ることになり、モデルが表現できる関数が著しく制限されてしまうからである。厳密に言えば、Equation 2.1 は特徴量

の**アフィン**関数（線形変換に定数の平行移動を加えたもの）であるが、機械学習ではこれを一様に線形モデルと呼ぶ。

特徴量が d 個あるとき、モデルの予測値 $\hat{y}$（ハット記号は観測ラベル y と推定値を区別するためのもの）は次の形をとる。

$$\hat{y} = w_1 x_1 + \cdots + w_d x_d + b = \mathbf{w}^\top \mathbf{x} + b, \tag{2.2}$$

ここで、$\mathbf{w} = (w_1, \ldots, w_d)^\top \in \mathbb{R}^d$ は重みを 1 つの列ベクトルにまとめたものである。全 n 個の例に対してモデルを一度に評価する際には、特徴ベクトル $\mathbf{x}^{(i)}$ を行として積み重ねた**計画行列**（design matrix）$\mathbf{X} \in \mathbb{R}^{n \times d}$ と、予測値 $\hat{y}^{(i)}$ をまとめた列ベクトル $\hat{\mathbf{y}} \in \mathbb{R}^n$ を用いて次のように書く。

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b, \tag{2.3}$$

ここで、スカラーのバイアス b はブロードキャスト（1.1.3.2 節）によって $\mathbf{X}\mathbf{w}$ のすべての成分に加算される。関係する各配列の形状を Figure 2.1 に示す。これにより、最良の線形予測器（linear predictor）を探す問題は、パラメータ $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ のうち、$\hat{\mathbf{y}}$ を観測ラベル $\mathbf{y} = (y^{(1)}, \ldots, y^{(n)})^\top$ にできる限り近づけるものを見つける問題へと帰着する。特徴量とラベルの真の関係が近似的にしか線形でなく、未測定の要因や測定ノイズが混入している場合でさえ、線形モデルは高速で解釈しやすく、驚くほど競争力のあるベースラインとなることが多い。

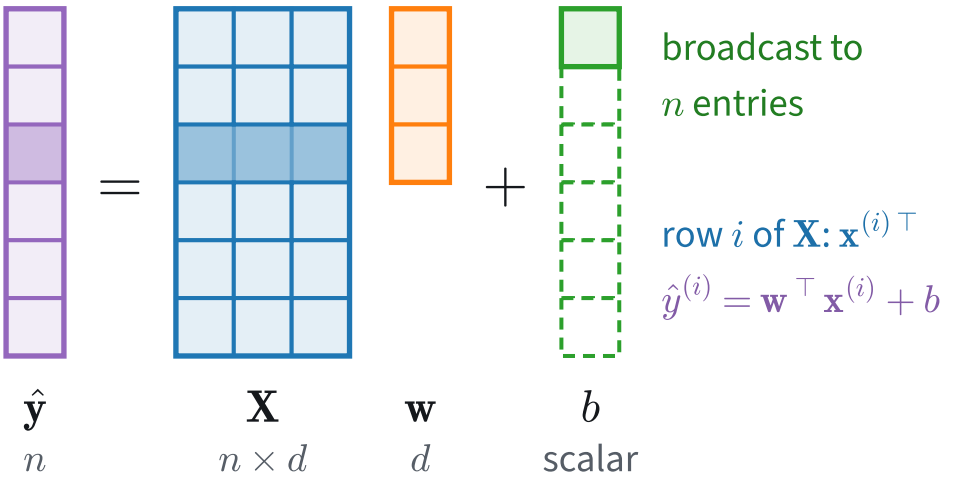


Figure 2.1: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$ における各配列の形状（$n = 6$ 個の例と $d = 3$ 個の特徴量の場合）。$\mathbf{X}$ の各行は 1 つの例を表し、$\hat{\mathbf{y}}$ の 1 つの要素（網掛け部分）を生成する。$\mathbf{w}$ は列ごとに 1 つの重みを持ち、スカラー b は全 n 個の要素にブロードキャストされる。

2.1.1.3 二乗損失

最良のパラメータを探索するには、それに先立って、任意の候補 $(\mathbf{w}, b)$ が観測ラベルからどれほど大きく外れているかを測る定量的な尺度が必要になる。**損失関数**は予測値 $\hat{y}$ と真の値 y のずれに対して非負のペナルティを割り当て、両者が一致するときに 0 となる。回帰における標準的な選択は**二乗誤差**（squared error）である。

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2. \tag{2.4}$$

係数 $\frac{1}{2}$ は純粋に式を見やすくするための便宜的なものであり、二乗を微分した際に降りてくる係数 2 を打ち消すだけで、最小点の位置は変えない。差 $\hat{y}^{(i)} - y^{(i)}$ が二乗されるため、同じ大きさの過大予測と過小予測には対称なペナルティが科される一方、大きな外れには小さな外れよりもはるかに重いペナルティが科される。たとえば 3 だけ外れた予測は、1 だけ外れた予測の 9 倍の損失を生む。幾何学的には（Figure 2.2）、各残差（residual）$\hat{y}^{(i)} - y^{(i)}$ はデータ点と回帰直線のあいだの鉛直距離であり、$2\,l^{(i)}$ はその鉛直線分を 1 辺とする正方形の面積にあたる。

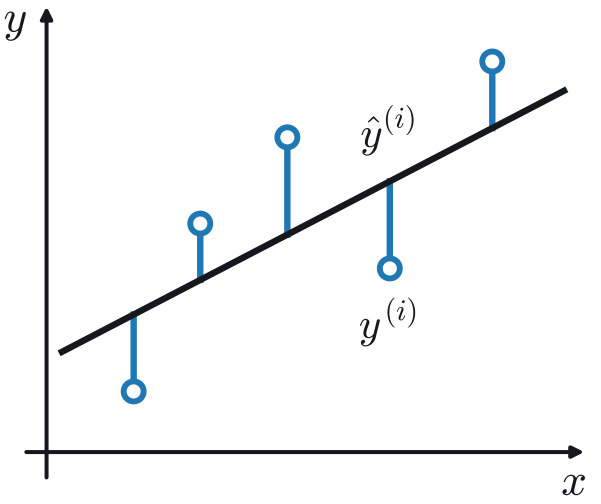


Figure 2.2: 鉛直方向の残差の二乗和を最小化することによるデータへの直線の当てはめ。

$(\mathbf{w}, b)$ を n 個の例からなる訓練セット（training set）全体で評価するため、例ごとの損失を平均して経験**損失**（または**コスト・目的関数**）を定義する。

$$\begin{aligned} L(\mathbf{w}, b) &= \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2 \\ &= \frac{1}{2n}\|\mathbf{X}\mathbf{w} + b - \mathbf{y}\|_2^2. \end{aligned}$$

n で割ることで目的関数が 1 例あたりのスケールになるため、データセットのサイズが 2 倍になっても損失やその勾配（gradient）の数値的な大きさは 2 倍にならない。モデルを訓練するとは、次の最適化（optimization）問題を解くことを意味する。

$$\mathbf{w}^*, b^* = \underset{\mathbf{w}, b}{\operatorname{argmin}}\ L(\mathbf{w}, b). \tag{2.5}$$

誤差が二乗されるため、10 だけ外れた予測は 1 だけ外れた予測の 100 倍のペナルティを被り、たった 1 つの破損したラベルが当てはめ全体を支配してしまうことがある。その影響がどれほど深刻かを見るために、直線 $y = 2x$ 上に 20 個の点を置き、（データ入力ミスで起こりうるよう

に）1 つのラベルだけを $y^{(6)} = 10{,}000$ に破損させたうえで、原点を通る直線 $\hat{y} = wx$ を 2 通りの方法で当てはめてみよう。1 つ目は二乗損失の閉形式（closed-form）の最小点（minimizer）$w = \sum_i x^{(i)} y^{(i)} / \sum_i (x^{(i)})^2$ による方法、2 つ目は傾きの候補を細かく並べたグリッド上で**平均絶対誤差**（mean absolute error）$\frac{1}{n}\sum_i |wx^{(i)} - y^{(i)}|$ を最小化する方法である。

```
x = np.arange(1.0, 21.0)
y = 2 * x
y[5] = 10000                            # corrupt a single label
w_sq = (x * y).sum() / (x * x).sum()    # squared-loss minimizer
w_grid = np.linspace(0, 30, 3001)       # slopes 0, 0.01, ..., 30
mae = np.abs(w_grid[:, None] * x - y).mean(axis=1)
w_mae = w_grid[mae.argmin()]            # MAE minimizer on the grid
print(f'true w: 2.00, squared loss: {w_sq:.2f}, MAE: {w_mae:.2f}')
```

```
true w: 2.00, squared loss: 22.88, MAE: 2.00
```

20 個のうちわずか 1 個の破損した点が二乗損失による傾きを 2.00 から 22.88 まで一気に引きずり込むのに対し、絶対誤差による傾きは 2.00 から微動だにしない。二乗損失のもとでは、ほぼ 10,000 に達する 1 つの誤差が残り 19 個の正常な点の引き戻す力の合計を圧倒してしまうが、絶対誤差のもとではその引っ張る力が線形にしか増えないため、19 個の正常な点が容易に多数決で勝つのである。どちらの振る舞いが望ましいかはラベルにどのようなノイズを想定するかに依存する。この問いには 2.1.3 節で第一原理から答える。

2.1.2 損失の最小化

$(\mathbf{w}^*, b^*)$ は、特徴量が d 個、例が n 個あるときにどうやって求めればよいだろうか。線形回帰は、厳密な閉形式の公式とスケーラブルな反復オプティマイザの両方を持つ、機械学習でもごくまれなモデルの 1 つである。この 2 つを比較すると、深層学習がなぜ圧倒的に反復的な勾配法を好むのかが明らかになる。

2.1.2.1 正規方程式

まず、スカラーのバイアス b は、計画行列に定数 1 の列を追加することで重みベクトルに吸収できる。各入力は $(\mathbf{x}^{(i)}, 1)^\top \in \mathbb{R}^{d+1}$ となり、パラメータベクトルは $(\mathbf{w}, b)^\top \in \mathbb{R}^{d+1}$ となる（あるいは同値な方法として、$\mathbf{X}$ と $\mathbf{y}$ の両方をそれぞれの標本平均で中心化してもよい。演習問題 1 を参照）。この拡大表記で $\mathbf{X}$ と $\mathbf{w}$ を書けば、損失は単純に次の形になる。

$$L(\mathbf{w}) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2. \tag{2.6}$$

$\mathbf{X}^\top\mathbf{X}$ が正則である限り（これは $\mathbf{X}$ の列が線形独立であるとき、かつそのときに限り成り立ち、$n \geq d$ が必要となる）、導関数を計算せず平方完成という純粋な代数操作だけで大域的最小値（global minimum）を求めることができる。$\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$ と定義すると、$\mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^\top\mathbf{y}$ が成り立つ。ノルムの内部で $\mathbf{X}\hat{\mathbf{w}}$ を足して引き、二乗を展開すると次を得る。

$$\begin{aligned}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 &= \|(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) + \mathbf{X}(\hat{\mathbf{w}} - \mathbf{w})\|_2^2 \\ &= \|\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}\|_2^2 + \|\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w})\|_2^2 + 2(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^\top\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}).\end{aligned} \tag{2.7}$$

末尾の交差項に注目しよう。

$$(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^\top \mathbf{X} = (\mathbf{X}^\top \mathbf{y} - \mathbf{X}^\top \mathbf{X}\hat{\mathbf{w}})^\top = \mathbf{0}^\top.$$

この項は**あらゆる** $\mathbf{w}$ に対して恒等的に 0 になる。残るのは 2 つの非負の二乗ノルムの和である。第 1 項 $\|\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}\|_2^2$ は $\mathbf{w}$ にまったく依存せず、第 2 項 $\|\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w})\|_2^2$ は $\mathbf{w} = \hat{\mathbf{w}}$ のとき、かつそのときに限り 0 になる（$\mathbf{X}$ が列フルランクを持つため）。したがって、$\hat{\mathbf{w}}$ が唯一の大域的最小点である。微分法を用いて勾配 $\nabla_{\mathbf{w}} L(\mathbf{w}) = -\frac{1}{n}\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$ を 0 とおいても同じ方程式が得られ、古典的な**正規方程式**が導かれる。

$$\mathbf{X}^\top \mathbf{X}\mathbf{w}^* = \mathbf{X}^\top \mathbf{y}, \qquad \text{or} \qquad \mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}. \tag{2.8}$$

$\mathbf{X}$ の列が線形従属である場合（たとえば 2 つの特徴量が冗長である場合や $d > n$ の場合）、$\mathbf{X}^\top \mathbf{X}$ は特異になる。このときも最小二乗法の最小点は依然として存在するが、無限に多くの重みベクトルが同じ最小損失を達成する（演習問題 6）。

コード上では、$(\mathbf{X}^\top \mathbf{X})^{-1}$ を明示的に計算すべき場合はまずない。専用の最小二乗ソルバ（`lstsq`）は QR 分解や特異値分解（singular value decomposition、SVD）を用いるため、同程度の計算コストではるかに数値的に安定である。1.4 節の 3 点の直線モデル（入力 $0, 1, 2$、ラベル $1, 1, 0$、拡大行 $(1, \text{input})$）で正規方程式を検証してみよう。このとき $\hat{\mathbf{w}} = (w_1, w_2)^\top = (7/6, -1/2)^\top$ は第 1 成分に切片、第 2 成分に傾きを持つ。あわせて、残差ベクトル $\mathbf{r} = \mathbf{X}\hat{\mathbf{w}} - \mathbf{y} = (1/6, -1/3, 1/6)^\top$ が `float32` の精度の範囲内で $\mathbf{X}^\top \mathbf{r} = \mathbf{0}$ を満たすことも確認しよう。

```
X = jnp.array([[1.0, 0.0], [1.0, 1.0], [1.0, 2.0]])  # rows (1, input)
y = jnp.array([[1.0], [1.0], [0.0]])
w_hat = jnp.linalg.lstsq(X, y)[0]
r = X @ w_hat - y                                    # errors
print('w_hat:', tuple(round(float(v), 4) for v in w_hat[:, 0]))
print('r:', tuple(round(float(v), 4) for v in r[:, 0]))
print(f'max |X^T r|: {float(jnp.abs(X.T @ r).max()):.0e}')
```

```
w_hat: (1.1667, -0.5)
r: (0.1667, -0.3333, 0.1667)
max |X^T r|: 3e-07
```

2.1.2.2 射影としての最小二乗法

なぜこれは**正規**（normal）方程式と呼ばれるのだろうか。交差項が消える Equation 2.7 の恒等式は、文字どおり $\mathbb{R}^n$ におけるピタゴラスの定理にほかならない（Figure 2.3）。$\mathbf{w}$ が $\mathbb{R}^d$ 全体を動くとき、とりうる予測ベクトル $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ は d 次元の部分空間（subspace）を $\mathbb{R}^n$ 内に描き出す。これが列空間（column space）、すなわち $\mathbf{X}$ が張る空間（span）である。通常、観測されたラベルベクトル $\mathbf{y} \in \mathbb{R}^n$ はこの部分空間上には乗らない。d 本の特徴量の列をどのように線形結合しても、ノイズを含む n 個の観測値すべてを再現することはできないからである。

$\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$ を最小化するとは、点 $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ として、$\mathbf{X}$ の列空間内でユークリッド距離（Euclidean distance）において $\mathbf{y}$ に最も近い点を見つけることを意味する。初等幾何学より、平坦な部分空間内で最も近い点は、その部分空間への $\mathbf{y}$ の**直交射影**（orthogonal projection）、すなわち差ベクトル $\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}$ が $\mathbf{X}$ のすべての列に垂直（**法線**方向、normal）となる唯一の点である。$\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}$ が $\mathbf{X}$ のすべての列と直交するという条件は $\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) = \mathbf{0}$ と書け、これこそが正規方程式にほかならない。

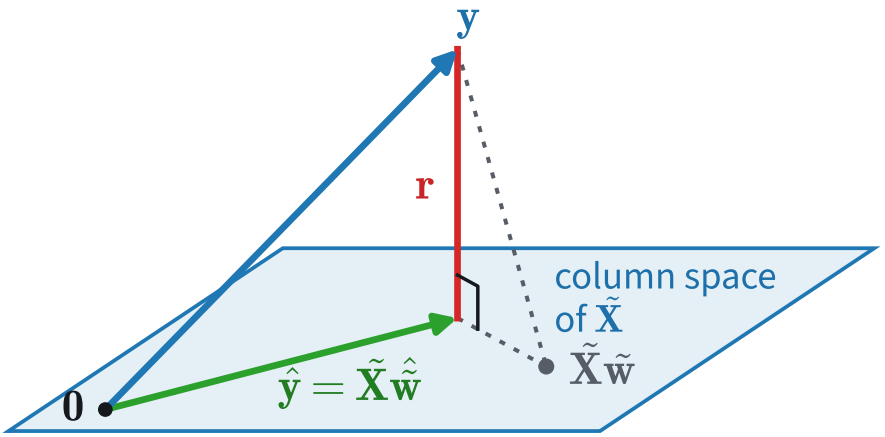


Figure 2.3: $\mathbb{R}^n$ における直交射影としての最小二乗法。$\mathbf{X}$ の列空間はあらゆる予測ベクトル $\mathbf{Xw}$ を含み、$\mathbf{y}$ に最も近い点が直交射影 $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ である。その残差 $\mathbf{y} - \hat{\mathbf{y}}$ は $\mathbf{X}$ のすべての列と直交する（法線をなす）。

2.1.2.3 ミニバッチ確率的勾配降下法

線形回帰に 1 行で書ける解析解があるなら、なぜここで終わりにしないのだろうか。現代の深層学習全般において閉形式解を阻む障壁が 2 つある。

1. **計算量のスケーリング：** $\mathbf{X}^\top\mathbf{X} \in \mathbb{R}^{d\times d}$ の計算には $O(nd^2)$ 回の演算コストがかかり、$d \times d$ の連立方程式を解くには $O(d^3)$ のコストがかかる。データセットが数百万個の例を持ち、モデルが数百万〜数十億個のパラメータを持つとき、$d \times d$ 行列を保存したり逆行列を求めたりすることは到底不可能である。
2. **非線形性：** 二乗損失を交差エントロピー（cross-entropy、第 3 章）のような別の目的関数に置き換えたり、非線形な隠れ層（hidden layer、第 4 章）を積み重ねたりした途端に、閉形式解は一切存在しなくなる。

その代わりに、ほぼすべての深層学習モデルは**勾配降下法**（gradient descent）とその確率的変種を用いて反復的に訓練する。1.4 節で見たように、勾配 $\nabla L(\mathbf{w}, b)$ は損失が最も急激に増加する方向を指す。したがってその**逆**方向に進めば損失は減少するが、全勾配 $\nabla L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \nabla l^{(i)}(\mathbf{w}, b)$ を評価するには、パラメータをたった 1 ステップ更新するだけでも全 n 個の訓練例を走査しなければならない。$n = 10^7$ のとき、フルバッチ勾配降下法は気が遠くなるほど遅いうえに統計的にも無駄が多い。無作為に選んだ 100 個の例にわたる平均勾配で、すでに 1000 万個にわたる平均の実用的な推定値になるからである。その推定値がどれほど正確かは標本の大きさに依存するのであり、抽出元のデータセットの大きさには依存しない。

ミニバッチ確率的勾配降下法（ミニバッチ SGD）は、この統計的冗長性を活用する。各ステップ t において、小さな無作為**ミニバッチ** $\mathcal{B}_t \subset \{1, \dots, n\}$（例数 $|\mathcal{B}|$）を抽出し、$\mathcal{B}_t$ のみにわたって損失の勾配を平均し、正の**学習率**（learning rate、ステップサイズ）$\eta > 0$ を用いてパラメータを次のように更新する。

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}_t} \partial_{(\mathbf{w},b)}\, l^{(i)}(\mathbf{w}, b). \tag{2.9}$$

$\mathcal{B}_t$ は訓練セットから一様にサンプリングされるため、ミニバッチ勾配は真のフルバッチ勾配の**不偏推定量**（unbiased estimator）となり、その標準誤差（standard error）は $1/\sqrt{|\mathcal{B}|}$ のオーダーで減少する。二乗損失 Equation 2.4 のもとでの線形回帰では、$l^{(i)} = \frac{1}{2}(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)})^2$ を明示的に微分することで次の更新則が得られる。

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}_t} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right), \\ b &\leftarrow b - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}_t} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right). \end{aligned} \tag{2.10}$$

各項の働きを見てみよう。スカラー $\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}$ は、例 i に対する符号付き予測誤差である。モデルが過大に予測した場合（$\hat{y}^{(i)} > y^{(i)}$）、更新によって $\mathbf{x}^{(i)}$ の定数倍が $\mathbf{w}$ から引かれ、b も下方へ修正されるため、次の予測値は $y^{(i)}$ に近づく。概念上の区別も重要である。$\mathbf{w}$ と b は最適化によって学習される**モデルパラメータ**であるのに対し、ミニバッチサイズ $|\mathcal{B}|$ と学習率 η は訓練開始前に選択される**ハイパーパラメータ**である（これらは 2.6 節で議論するように、別途取り置いた検証データ上で調整する）。

2.1.2.4 勾配ノイズと学習率

ミニバッチ SGD の軌跡（trajectory）はフルバッチ勾配降下法とどのように異なるだろうか。経験損失 $L(\mathbf{w})$ の最小二乗解 $\hat{\mathbf{w}}$ のまわりにおける等高線を考えよう（Figure 2.4）。フルバッチ勾配降下法は毎ステップで L の全 n 個の例にわたる勾配を計算し、楕円状の等高線を滑らかに横切って最小点 $\hat{\mathbf{w}}$ へと進む（ただし η が大きすぎて行き過ぎて発散しないものとする）。

ミニバッチ SGD はこれとは異なる経路をたどる。大域的最小点 $\hat{\mathbf{w}}$ では全 n 個の例にわたる平均勾配こそ 0 になるものの、ノイズを含む個々の例には依然として 0 でない残差 $\hat{y}^{(i)} - y^{(i)} \neq 0$ がある。無作為なミニバッチ $\mathcal{B}_t$ において、例ごとの勾配が互いに打ち消し合って 0 になることはまずない。その結果、学習率 η を一定に保つと、ミニバッチ SGD は $\hat{\mathbf{w}}$ の小さな近傍へ急速に降下したのち、$\hat{\mathbf{w}}$ の周囲にある***ノイズ球***（noise ball）の内部で永続的に揺らぎ続ける。この球の半径の二乗は $\eta/|\mathcal{B}|$ に比例する。

η を半分にするか $|\mathcal{B}|$ を 2 倍にすればこの終端のノイズ球は縮小し、訓練の過程で $\eta \to 0$ と減衰させれば反復列は $\hat{\mathbf{w}}$ に収束する（第 9 章）。しかし実際には、訓練損失の底を小数点以下 10 桁まで突き詰めることが望ましい場合はめったにない。最終的な目標は**未知の**データに対する誤差を小さくすることであり、$\hat{\mathbf{w}}$ 自体が真の母集団パラメータから $1/\sqrt{n}$ のオーダーの統計的推定誤差だけずれている以上、その統計的不確実性の範囲内で生じるわずかな最適化の揺らぎは汎化（generalization）において何のコストももたらさないからである。

2.1.2.5 高速化のためのベクトル化

ミニバッチ更新 Equation 2.10 を実装する際、例 $i \in \mathcal{B}_t$ を Python で 1 つずつループ処理してはならない。代わりに計算を**ベクトル化**し、ミニバッチ全体の順伝播（forward pass）と勾配ステップを配列・行列演算として表現することで、SIMD ベクトル演算器や GPU コア上で並列実行される C++/CUDA ライブラリへと処理を委ねる。

その違いを実際に確かめるため、$n = 1{,}000$ 個の例に対するラベル予測を、重み $\mathbf{w} = (2, -3.4)^\top$、バイアス $b = 4.2$ のもとでベンチマークしてみよう。まず Python の `for` ループで 1 例ずつ計算し、次に全 $1{,}000$ 個の例を単一の行列積 `X @ w + b` で一度に計算する。

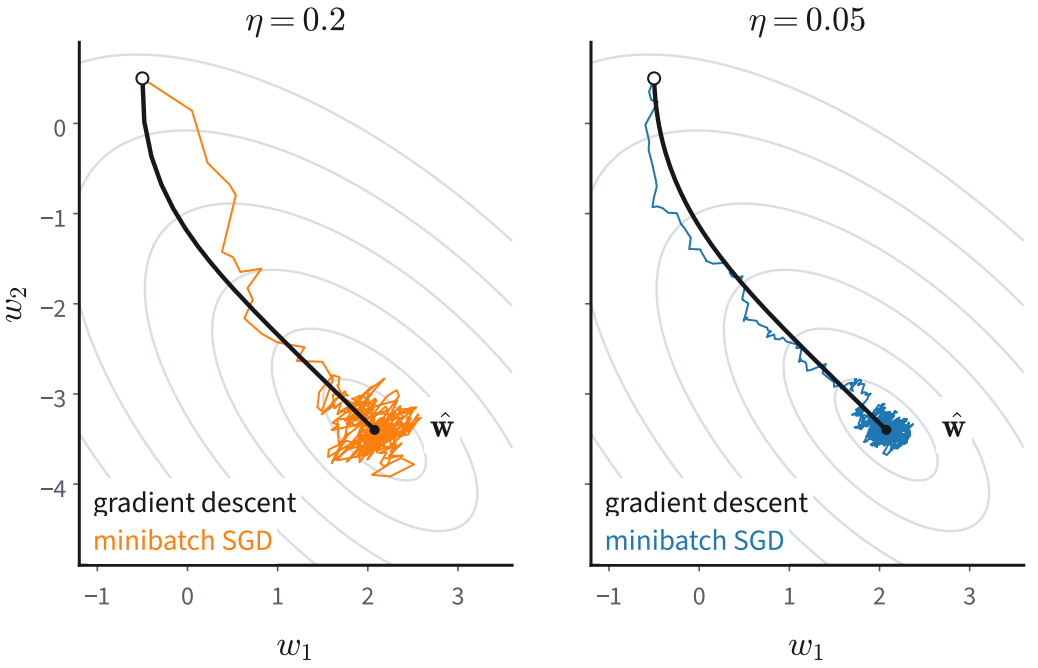


Figure 2.4: フルバッチ勾配降下法（黒）は L の勾配に沿って最小点 $\hat{\mathbf{w}}$ へ滑らかに進むが、1 ステップごとに全 n 個の例を 1 回走査する。ミニバッチ SGD（カラー）はノイズを含む勾配推定値を用いて 1 回の走査あたり $n/|\mathcal{B}|$ 回の低コストなステップを踏み、$\hat{\mathbf{w}}$ の周囲の小さなノイズ球へ急速に降下する。学習率を 4 分の 1 にすると（右）、この球は縮小する。

```
n = 1000
X = jnp.ones((n, 2))               # the values do not affect the timing
w = jnp.array([[2.0], [-3.4]])
b = 4.2
wait = jax.block_until_ready       # JAX returns before it has finished
```

計測用ヘルパ関数 seconds は、まずウォームアップ呼び出しを 1 回実行して（初回のみのセットアップや JIT コンパイルのコストを吸収するため）、計測終了前に wait を呼び出す。GPU やアクセラレータのフレームワークは演算を非同期にディスパッチするため、ハードウェアがまだ計算中であっても Python に制御を戻すことがあるからである。

```
def seconds(f):
    """Time one call of f, after a warm-up call."""
    wait(f())
    start = time.perf_counter()
    wait(f())
    return time.perf_counter() - start
```

```
loop = seconds(lambda: [X[i:i + 1] @ w + b for i in range(n)])
vectorized = seconds(lambda: X @ w + b)
print(f'one example at a time: {loop * 1e3:8.3f} ms')
print(f'all examples at once:  {vectorized * 1e3:8.3f} ms')
print(f'speedup: {loop / vectorized:.0f}x')
```

```
one example at a time:  485.697 ms
all examples at once:     0.559 ms
speedup: 869x
```

どちらの方法も同じ算術演算をしているにもかかわらず、全 1,000 個の予測を 1 回のベクトル化された呼び出しで計算する方が数百倍も高速である。ループでは 1 例ごとに Python インタプリタのオーバーヘッドとカーネル起動のレイテンシが発生するのに対し、行列積ではそのオーバーヘッドを 1 度支払うだけでバッチ全体をコンパイル済みの単一の線形代数ルーチンに引き渡せる。本書を通じて、ミニバッチ単位でベクトル化すること、すなわち n 個の内積 $\mathbf{w}^\top \mathbf{x}^{(i)}$ を 1 つの行列ベクトル積 $\mathbf{X}\mathbf{w}$ に置き換えることは、高速な深層学習コードを書くうえで最も重要な習慣である。

2.1.3 二乗損失とガウスノイズ

ここまでは、幾何学的な扱いやすさと導関数の簡潔さから二乗誤差損失 $\frac{1}{2}(\hat{y} - y)^2$ を動機づけてきた。ここで、Gauss が立てたより深い問いに進もう。すなわち、**データに関するどのような確率論的仮定のもとで、二乗誤差の最小化が統計的に最適な選択になるのだろうか。**

2.1.3.1 正規分布

平均 μ、分散 σ^2（標準偏差 σ）を持つ正規分布（normal distribution、ガウス分布とも呼ばれる）を振り返ろう。その確率密度（probability density）は次の式で与えられる。

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right).$$

この密度は μ を中心とする対称な釣鐘型の曲線を描き、その幅は標準偏差 σ によって決まる（確率の約 68% が平均から σ 以内に、約 95% が 2σ 以内に収まる）。偏差の二**乗** $(x-\mu)^2$ が負の指数関数の内部に現れるため、極端な外れが生じる確率は無視できるほど小さい。たとえば平均から 5σ 離れた値の密度は、ピーク時の $e^{-12.5} \approx 4 \times 10^{-6}$ 倍にすぎない。正規分布に従う確率変数（random variable）を $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$ と表記する。多数の小さな独立した撹乱の和は正規分布へと収束するため（36.3.2.3 節）、性質のよい測定ノイズに対する自然な標準モデルとしてガウス分布が用いられる。

定式化すると、加法的ガウスノイズを伴う線形回帰では、観測される各ラベルが線形信号と独立な正規ノイズ項の和であると仮定する。

$$y = \mathbf{w}^\top \mathbf{x} + b + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2). \tag{2.11}$$

このモデルのもとでは、条件付き平均 $\mathbb{E}[y \mid \mathbf{x}]$ は確かに $\mathbf{w}^\top \mathbf{x} + b$ となり（2.1.1.2 節と一致する）、ラベルは直線のまわりに同じノイズスケール σ で、どの入力 $\mathbf{x}$ においても対称にばらつく（Figure 2.5）。

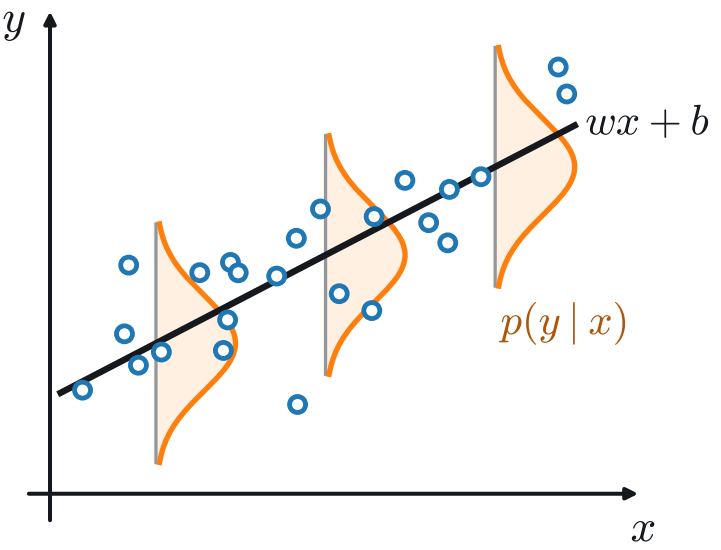


Figure 2.5: ガウスノイズモデル。どの入力 x に対しても、ラベル y は正規密度 $p(y \mid x)$（直線 $wx + b$ を中心とし、どこでも同じ幅を持つ）から抽出される。

2.1.3.2 最尤法

この加法的ガウスノイズモデルのもとでは、特徴量 $\mathbf{x}$ が与えられたときの観測ラベル y の条件付き分布（conditional distribution）は、平均 $\mathbf{w}^\top \mathbf{x} + b$、分散 σ^2 の正規分布となる。

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right). \tag{2.12}$$

最尤法（maximum likelihood）の原理に従えば、$(\mathbf{w}, b)$ はデータセット全体 $\{y^{(1)}, \ldots, y^{(n)}\}$ が（$\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$ を条件として）観測される条件付き確率密度を最大化するように選ぶべ

きである。各例が独立に抽出されると仮定すると、データセットの尤度（likelihood）は積 $\prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)})$ に分解される。正の積を最大化することは、その負の対数である**負の対数尤度**（negative log-likelihood）を最小化することと同値である。

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2. \tag{2.13}$$

右辺の 2 つの項を詳しく見てみよう。第 1 項 $\frac{n}{2}\log(2\pi\sigma^2)$ は $\mathbf{w}$ にも b にもまったく依存しない。第 2 項は単なる $\frac{n}{\sigma^2}L(\mathbf{w}, b)$、すなわち平均二乗誤差（mean squared error）損失に正の定数 n/σ^2 を掛けたものである。したがって、二乗誤差損失を最小化すると、独立な等分散（homoscedastic）ガウスノイズのもとでの $(\mathbf{w}, b)$ の**最尤推定量**（maximum-likelihood estimator）が、σ^2 の値にかかわらず得られる。

2.1.3.3 ノイズモデルからの損失の選択

ノイズがガウス分布では**ない**とき、二乗損失は何をしているのだろうか。その答えに尤度は一切不要である。入力 $\mathbf{x}$ を固定し、どの単一の数値 c が期待二乗誤差 $\mathbb{E}[(y-c)^2 \mid \mathbf{x}]$ を最小化するかを問うてみよう。導関数 $2(c - \mathbb{E}[y \mid \mathbf{x}])$ を 0 とおくと、有限の分散を持つあらゆるノイズ分布に対して条件付き平均 $c = \mathbb{E}[y \mid \mathbf{x}]$ が得られる。一方、期待絶対誤差 $\mathbb{E}[|y-c| \mid \mathbf{x}]$ を最小化するのは条件付き**中央値**（median）である。これを非対称に傾けて、単位あたりの過小予測に q のコスト、単位あたりの過大予測に $1-q$ のコストがかかるようにすると、最小点は q 分位点（quantile）になる。平均よりも 90% 需要水準を必要とすることの多い予測実務家は、これを**ピンボール損失**（pinball loss）と呼ぶ。2.1.1.3 節の破損したラベルの例はまさにこの違いを示していた。平均はたった 1 つの巨大な値に引きずられるが、中央値はそれを無視する。したがって、損失を選ぶとは、実際のノイズがどのような分布であれ、y のどの統計量をモデルに予測させるかを選ぶことにほかならない。

さらに最尤法とのつながりから、ガウスノイズが現実の適切なモデルでは**ない**場合に損失関数を設計するための原理的な指針が得られる（Figure 2.6）。

出力の種類	ノイズモデル	損失（負の対数尤度）	予測される統計量
実数値、性質のよいノイズ	ガウス分布	二乗誤差（本節）	y の平均
実数値、外れ値または重い裾	ラプラス分布	絶対誤差（演習問題 2）	y の中央値
正値、乗法的な変動（価格）	$\log y$ 上のガウス分布	$\log y$ の二乗誤差（演習問題 3）	$\log y$ の平均
計数（販売数、到着数）	ポアソン分布	$\nu(\mathbf{x}) - y\log\nu(\mathbf{x})$（演習問題 4）	y の平均

- **裾の重い外れ値（ラプラスノイズ $\to \ell_1$ 損失）：**まれに生じる破損した測定値が極端な外れ値を生む場合、$(\hat{y}-y)^2$ は二次関数的に増大するため、たった 1 つの異常なデータ点が当てはめた直線を自分の方へ引き寄せてしまう。ラプラス分布 $p(\epsilon) = \frac{1}{2\tau}\exp(-|\epsilon|/\tau)$ を用いてより重い指数関数的な裾を持つノイズとしてモデル化すると、負の対数尤度は**絶対誤差**（absolute error、ℓ_1）損失 $|\hat{y}-y|$ となり（演習問題 2）、外れ値がどれほど遠くにあっても勾配の大きさは一定の ± 1 に保たれる。
- **滑らかな最適点と外れ値への頑健性の両立（フーバー損失）：**$|\hat{y}-y|$ は 0 に鋭い角を持ち、そこで導関数が -1 と $+1$ のあいだで不連続に跳ぶため、実務では**フーバー損失**（Huber loss）がよく用いられる。これは小さな残差（$|\hat{y}-y| \le \delta$）に対する二次関数と、大きな

残差（$|\hat{y} - y| > \delta$）に対する線形の傾きを滑らかにつなぎ合わせた損失であり、外れ値への頑健性（robustness）と滑らかさを兼ね備える。

- **計数データ（ポアソンノイズ）**：目的変数 $y \in \{0, 1, 2, \ldots\}$ が離散的な計数（1 日あたりの来店客数など）である場合、ポアソン尤度からそれ固有の凸な負の対数尤度が導かれる（演習問題 4）。

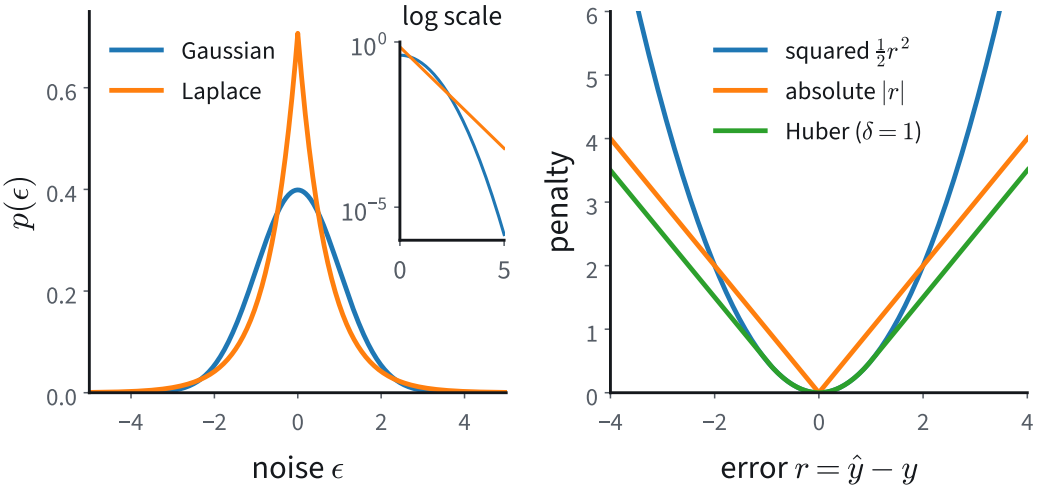


Figure 2.6: 3 つの回帰損失関数（右）と、そのうち最初の 2 つが最尤法のもとで対応するノイズ密度 $p(\epsilon) \propto \exp(-l(\epsilon))$（左）。二乗誤差（ガウス分布）と絶対誤差（ラプラス分布）のあいだにフーバー損失が位置する。対数スケールの挿入図はラプラス分布のより重い裾を示す。

2.1.4 ニューラルネットワークとしての線形回帰

線形回帰を後の章で扱う深層アーキテクチャへと結びつけるため、これを単層のニューラルネットワークとして描くことができる（Figure 2.7）。この図では、入力特徴量 $x_1, \ldots, x_d$ が**入力層**（input layer）のノードとして描かれ、それぞれ固有のシナプス重み $w_1, \ldots, w_d$ を介して**出力層**（output layer）の単一の計算ユニット o_1 へと結線されている。このユニットは重み付き入力の総和をとり、バイアス b を加えて $\hat{y} = o_1$ を出力する。

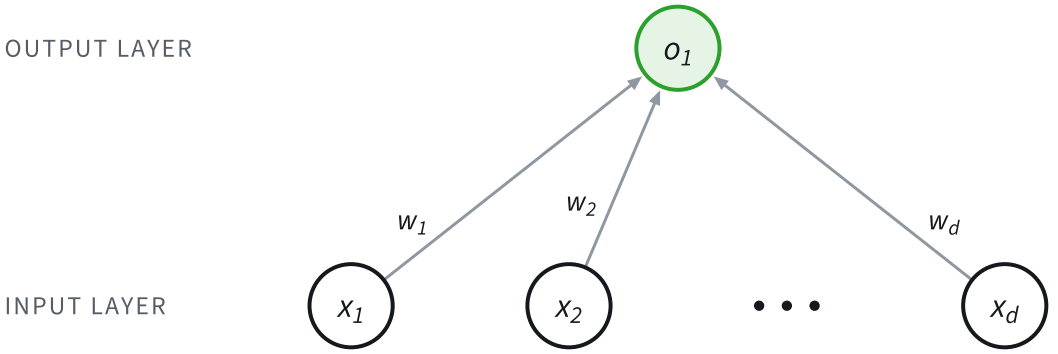


Figure 2.7: 線形回帰は単層のニューラルネットワーク（1 個の人工ニューロン）である。

慣例として、調整可能なパラメータを持つ層だけを数えるため、線形回帰は **1 層**のネットワーク

である。すべての入力特徴量が出力ノードに結合しているため、この層は**全結合層**（または**密結合層**、dense layer）と呼ばれる。大脳皮質の個々のニューロンも同様に、樹状突起上の何千ものシナプスからの重み付き信号を統合している。この類推は McCulloch と Pitts のような初期の先駆者たちに着想を与えた (McCulloch and Pitts 1943)。もっとも、本書全体を通じた焦点は、そのような層を合成して深層ネットワークを構築する数学と工学にある。

2.1.5 まとめ

線形回帰は特徴量のアフィン結合 $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$ によって連続値の目的変数を予測し、当てはまりの悪さを平均二乗誤差 $L(\mathbf{w}, b) = \frac{1}{2n}\|\mathbf{X}\mathbf{w} + b - \mathbf{y}\|_2^2$ で評価する。幾何学的には、閉形式の正規方程式 $\mathbf{X}^\top\mathbf{X}\mathbf{w}^* = \mathbf{X}^\top\mathbf{y}$ は $\mathbf{y}$ を $\mathbf{X}$ の列空間へ直交射影するが、その $d \times d$ 連立方程式の構成と求解には $O(nd^2 + d^3)$ 回の演算コストがかかり、非線形ネットワークへは一般化できない。これに対しミニバッチ確率的勾配降下法は、小さな無作為バッチ $\mathcal{B}_t$ 上でのベクトル化された行列演算によってパラメータを更新し、1 ステップあたり $O(|\mathcal{B}|d)$ の計算量で最適点の周囲の小さなノイズ球へと急速に降下する。統計学的には、二乗誤差の最小化は加法的ガウスノイズのもとでの最尤推定にあたり、ガウスモデルを裾の重い分布や離散分布に置き換えれば、絶対誤差やポアソン負の対数尤度といった代替損失が導かれる。

2.1.6 演習問題

1. **切片と中心化**。すべての入力に定数の特徴量 1 を追加すると、アフィンモデル $\mathbf{w}^\top\mathbf{x} + b$ は $(\mathbf{x}, 1)$ に関する線形モデルになる。データの中心化は、これとは別の方法でバイアスを取り除く。訓練セット全体にわたる特徴量の平均とラベルの平均をそれぞれ $\bar{\mathbf{x}}$、$\bar{y}$ とする。
   1. 二乗損失のあらゆる最小点 $(\hat{\mathbf{w}}, \hat{b})$ が $\hat{b} = \bar{y} - \hat{\mathbf{w}}^\top\bar{\mathbf{x}}$ を満たすことを示し、バイアスを持つ最小二乗法の当てはめにおける誤差 $\hat{y}^{(i)} - y^{(i)}$ の総和が 0 になることを導け。
   2. $\hat{\mathbf{w}}$ が、中心化されたデータ $(\mathbf{x}^{(i)} - \bar{\mathbf{x}},\ y^{(i)} - \bar{y})$ に対するバイアスなしの最小二乗法の当てはめになっていることを示せ。
2. **ラプラスノイズ**。加法ノイズ ϵ がスケール 1 のラプラス分布（36.3 節を参照）、すなわち $p(\epsilon) = \frac{1}{2}\exp(-|\epsilon|)$ に従うと仮定する。
   1. このモデルのもとでのデータの負の対数尤度 $-\log p(\mathbf{y} \mid \mathbf{X})$ を書き下せ。
   2. 閉形式解が存在するかどうかを判定せよ。
   3. この損失のもとでの $(\mathbf{w}, b)$ に対するミニバッチ確率的勾配降下法の更新式を Equation 2.10 の形で書き下せ。ただし、微分値として 0（$|\cdot|$ が原点で導関数を持たない場合）と、それ以外での傾き $\operatorname{sign}(\cdot) = \pm 1$ を用いよ。このような代替物は**劣勾配**と呼ばれる。各例の寄与の大きさがその誤差にどう依存するかを、この損失と二乗損失とで比較せよ。
   4. [code] この更新則を 2.1.1.3 節の破損ラベルデータに対し、原点を通る直線について適用せよ。毎ステップでデータセット全体を用い、学習率を 0.002 として $w = 0$ から 2,000 ステップ実行せよ。絶対誤差の最小点は厳密に $w = 2$ である。反復列はどこに行き着くか。最小点に落ち着かない理由を説明し、改善策を提案せよ。
3. **ガウスノイズを超えて**。住宅価格や株価を回帰によって推定したいとする。
   1. 価格を直接 $y = \mathbf{w}^\top\mathbf{x} + b + \epsilon$（ただし $\epsilon \sim \mathcal{N}(0, \sigma^2)$）とモデル化し、$\sigma = 100{,}000$ ドルとする。予測価格が 50,000 ドルの住宅に対して、このモデルが負の価格に割り当てる確率を計算せよ。単一の σ では、この住宅と予測価格が 5,000,000 ドルの住宅の両方に適合しえない理由を説明せよ。
   2. 次に価格の対数に対して回帰し、$\log \text{price} = \mathbf{w}^\top\mathbf{x} + b + \epsilon$ とする。ここでは $\epsilon \sim \mathcal{N}(0, \sigma^2)$ が対数スケール上で作用する。価格の条件付き平均と条件付き標準偏差を計算し、両者の比が $\mathbf{x}$ に依存しないことを示せ。さらに、$\exp(\mathbf{w}^\top\mathbf{x} + b)$ が価格の不偏な予測になっているかどうかを判定せよ。これと同じモデル化の選択が、オ

プション価格評価のブラック・ショールズモデルの基礎にもなっている (Black and Scholes 1973)。

4. **リンゴの個数を数える**。食料品店で売れるリンゴの**個数**を回帰によって推定したいとする。
   1. このような目的変数に対して加法的なガウスノイズモデルを用いる場合の問題点を挙げよ。
   2. 計数をモデル化するにはポアソン分布が用いられる。これは $p(k \mid \nu) = \nu^k e^{-\nu}/k!$ で与えられ、ここで $\nu > 0$ はレート、$k \in \{0, 1, 2, \ldots\}$ は観測される計数である。$\mathbb{E}[k] = \nu$ を証明せよ。
   3. 計数 k の負の対数尤度を、特徴量に依存するレート $\nu(\mathbf{x})$ のもとで（パラメータに依存しない項を除いて）導出し、2.1.3.3 節の表と一致することを確かめよ。
   4. パラメータ化 $\nu(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ と $\log \nu(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ を比較せよ。それぞれについて、すべての入力とパラメータに対して $\nu(\mathbf{x}) > 0$ が成り立つかどうかを判定し、(c) の損失にどのような帰結をもたらすかを述べよ。また、2 番目のパラメータ化について、その損失を $(\mathbf{w}, b)$ の式として書き下せ。
5. [code] **ノイズ球の大きさ**。$n = 1{,}000$ 個の例を、$\mathcal{N}(0, 1)$ から抽出した 2 つの特徴量とラベル $y = \mathbf{x}^\top \mathbf{w}^* + \epsilon$（ただし $\mathbf{w}^* = (2, -3.4)$、$\epsilon \sim \mathcal{N}(0, 1)$）によって生成し、`lstsq` で最小二乗解 $\hat{\mathbf{w}}$ を計算せよ。
   1. バイアスなしのミニバッチ SGD を $\hat{\mathbf{w}}$ を初期値として、ミニバッチサイズ 10、$\eta = 0.1$ で 20,000 ステップ実行せよ。平均二乗距離 $\mathbb{E}\|\mathbf{w} - \hat{\mathbf{w}}\|^2$（反復列と $\hat{\mathbf{w}}$ のあいだの距離の二乗平均）を推定せよ。
   2. $\eta \in \{0.05, 0.025\}$、およびミニバッチサイズ 20 と 40 について同じ実験を繰り返し、平均二乗距離がおよそ $\eta / |\mathcal{B}|$ に比例することを確かめよ。
   3. この距離が、$\hat{\mathbf{w}}$ と $\mathbf{w}^*$ がどれだけ離れているかに依存しないのはなぜか。もしラベルにノイズがまったくなければ、何が変わるか。
6. **特異な計画行列**。●列フルランクであれば最小二乗法の最小点は一意に定まる。ここでは逆に $\mathbf{X}^\top \mathbf{X}$ が特異であるとする。
   1. $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$ の最小点が依然として存在することを示し、すべての最小点からなる集合を特徴づけよ。
   2. 特異値分解を $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ と書く（ただし $\mathbf{U}$ と $\mathbf{V}$ は直交行列、Σ は非負の特異値を対角に持つ長方対角行列）。ムーア・ペンローズ擬似逆行列は次で定義される。

$$\mathbf{X}^\dagger = \mathbf{V}\Sigma^\dagger \mathbf{U}^\top, \tag{2.14}$$

   ここで $\Sigma^\dagger$ は Σ を転置し、その非ゼロ成分の逆数をとった行列である。$\mathbf{X}^\dagger \mathbf{y}$ が (a) の集合の中で最小ノルムを持つ最小点であることを示せ。
   3. $\mathbf{w} = \mathbf{0}$ から出発したとき、勾配降下法はどの最小点に収束するか。ミニバッチ確率的勾配降下法にした場合に答えが変わるかどうかも述べよ。

2.2 合成データ

実世界のデータセットは雑多であり、さらに厄介なことに「解答集」が付属していない。実際の住宅データで新しいアルゴリズムを訓練して重みベクトル $\hat{\mathbf{w}}$ を得たとしても、真のデータ生成パラメータ $\mathbf{w}^*$ や削減不能なノイズフロアを知る術はない。熟練した研究者は、実装を実データに投入する前に、まず正解（ground truth）が既知のモデルから生成した**合成データ**（synthetic data）でテストする。

合成データセットでは、特徴量の分布、真の重み $\mathbf{w}^*$ とバイアス b^*、標本サイズ n、およびノイズレベル σ を自由に制御できる。これにより、コードを直接検証する 2 つの手段が得られる。学習されたパラメータ $(\hat{\mathbf{w}}, \hat{b})$ が $(\mathbf{w}^*, b^*)$ に近づくかどうかをテストでき、また最終的な損失を理論上のノイズフロア $\frac{1}{2}\sigma^2$ と比較できる。本節では、続く 3 つの節を通して使用する合成データセッ

トを構築し、その散布図を観察したうえで、基準となるベンチマークとして最小二乗法による当てはめを計算する。

2.2.1 データの生成

手軽に可視化できるようにしつつ、複数の特徴量を扱う行列演算も実践するため、$n = 1{,}000$ 個の訓練例を $d = 2$ 個の特徴量で生成する。計画行列（design matrix）$\mathbf{X} \in \mathbb{R}^{1000\times 2}$ は各要素を独立な標準正規分布 $X_{ij} \sim \mathcal{N}(0,1)$ からサンプリングし、次の線形モデルからラベルを生成する。

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + b^* + \epsilon, \tag{2.15}$$

ここで、正解パラメータには次の値を用いる。

$$\mathbf{w}^* = (2, -3.4)^\top, \qquad b^* = 4.2, \tag{2.16}$$

さらに、加法的なガウスノイズ $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ を標準偏差 $\sigma = 0.01$ で加える。信号 $(\mathbf{X}\mathbf{w}^* + b^*)$ が数単位のスケールで変動するのに対し、ノイズ $\sigma = 0.01$ はその数百分の 1 の大きさしかない。したがって、正常に動作するオプティマイザであれば、$\mathbf{w}^*$ と b^* を数千分の 1 以内の誤差で復元できるはずである。

JAX は明示的な関数型の擬似乱数キーを用いるため（1.1 節）、このデータ生成器は追加の引数 `key` を 1 つ受け取り、その値は状態を持つキー補助関数 `d2l.get_key` から取得する。

```
def synthetic_data(w, b, noise, n, key):
    """n examples with standard normal features and labels Xw + b + noise."""
    key1, key2 = jax.random.split(key)
    X = jax.random.normal(key1, (n, w.shape[0]))
    eps = jax.random.normal(key2, (n, 1)) * noise
    return X, d2l.matmul(X, d2l.reshape(w, (-1, 1))) + b + eps
```

2,000 個の例を、$\mathbf{w}^* = (2, -3.4)$、$b^* = 4.2$、$\sigma = 0.01$ を用いて生成しよう。最初の 1,000 個は訓練用に確保し、残りの 1,000 個は検証用に残しておく。検証例は訓練例とまったく同じ分布から得られるが、モデルの当てはめ中には一切使用しない。そのため、検証例で損失を評価すれば、モデルが未知のデータをどれほど正確に予測できるかがわかる。この区別については 2.6 節で詳しく検討する。

```
w_star, b_star, noise = d2l.tensor([2, -3.4]), 4.2, 0.01
X, y = synthetic_data(w_star, b_star, noise, 2000, d2l.get_key())
```

`X` の各行には 2 次元の特徴ベクトルが格納され、`y` の各行には対応するスカラーのラベルが格納されている。最初のデータ例を調べ、そのノイズ $\epsilon^{(1)} = y^{(1)} - (\mathbf{w}^{*\top}\mathbf{x}^{(1)} + b^*)$ を逆算してみることで、データが正しく生成されているかを手早く確認できる。

```
x, label = X[0], y[0, 0]
eps = label - (d2l.reduce_sum(w_star * x) + b_star)
print(f'features: ({float(x[0]):.4f}, {float(x[1]):.4f})')
print(f'label: {float(label):.4f}, noise: {float(eps):.4f}')
```

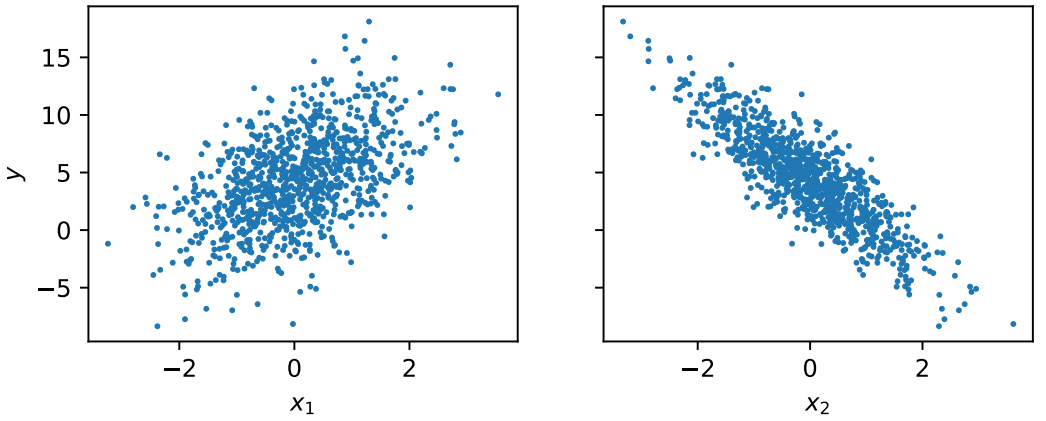
```
features: (-0.8760, 0.8323)
label: -0.3846, noise: -0.0030
```

最初の例について手計算で確かめてみると、$2x_1^{(1)} - 3.4x_2^{(1)} + 4.2$ は $y^{(1)}$ と、微小な $\mathcal{N}(0, 0.01^2)$ の摂動の範囲内で一致する。

2.2.2 データの観察

モデルを当てはめる前に、ラベル y を各特徴量 x_1、x_2 に対して個別にプロットしてみよう。

```
d2l.set_figsize((7, 2.5))
_, axes = d2l.plt.subplots(1, 2, sharey=True)
for j, ax in enumerate(axes):
    ax.scatter(d2l.numpy(X[:1000, j]), d2l.numpy(y[:1000, 0]), s=2)
    ax.set_xlabel(f'$x_{j + 1}$')
    ax.set_ylabel('$y$' if j == 0 else '')
```



どちらの散布図にも、予想どおりの符号を持つ明確な線形トレンドが現れている。y は x_1 に伴って上昇し（傾き $+2$）、x_2 に伴ってより急激に下降する（傾き -3.4）。$\sigma = 0.01$ はこれほど小さいのに、なぜ各 2D 散布図はかみそりの刃のように細い直線ではなく、太い帯のように見えるのだろうか。それは、y を x_2 だけに対してプロットすると、各点における x_1 の変動が無視されるからである。プロットされていない項 $2x_1$ は標準偏差 2 を持つため、点の縦方向の位置が数単位の幅を持つ帯状に広がるのである。多変数の関係を周辺化した 2D 射影として観察すると、プロットされていない座標の変動が常に見かけ上の縦方向のばらつきとして折り畳まれる。

2.2.3 パラメータの復元

2.1.2.1 節で閉じた形の正規方程式を導出したので、1,000 個の例に対する最小二乗法による厳密解 $(\widehat{\mathbf{w}}, \widehat{b})$ を、`lstsq` を拡大計画行列 $[\mathbf{X}, \mathbf{1}]$ に適用して早速計算してみよう。これにより、2.3 節のミニバッチ SGD が一致を目指す基準（有限標本における理想的な当てはめ）が得られる。

```
n = 1000                                     # The training examples
X1 = jnp.concatenate((X[:n], jnp.ones((n, 1))), 1)  # A column of ones for b
w_hat = jnp.linalg.lstsq(X1, y[:n])[0][:, 0]
```

ここで、実際の推定誤差を、理論的に予測される標準誤差 $\sigma/\sqrt{n}$ と並べて出力する。

```
e_w = d2l.numpy(w_hat[:2] - w_star)
print(f'error in w: ({e_w[0]:.1e}, {e_w[1]:.1e})')
print(f'error in b: {float(w_hat[2] - b_star):.1e}')
print(f'standard error: {noise / n ** 0.5:.1e}')
```

```
error in w: (3.6e-04, -5.6e-05)
error in b: -3.2e-04
standard error: 3.2e-04
```

`lstsq` で経験損失を機械精度まで解いたとしても、$(\hat{\mathbf{w}}, \hat{b})$ は真の値 $(2, -3.4, 4.2)$ と小数第 4 位で異なる。このわずかなずれは数値誤差ではなく、**統計的推定誤差**（statistical estimation error）である。1,000 個のノイズの実現値 ϵ がたまたま $\mathbf{X}$ の各列とわずかに相関し、最小二乗平面を $\sigma/\sqrt{n} = 0.01/\sqrt{1000} \approx 3 \times 10^{-4}$ 程度の大きさだけ傾けるのである（演習問題 1）。これはソルバではなくデータの限界である。1 回の実行で偶然より近い値になることはあっても、ノイズを含むこれら 1,000 個の点から計算されるいかなる不偏推定量（unbiased estimator）も、これより小さい期待二乗誤差は達成できない（**クラメール・ラオの下界**、Cramér–Rao bound）。また、演習問題 2 で調べるように、真のデータ生成関数が非線形である場合、最小二乗法は特徴量の分布上でその関数に対する最良の線形近似を求める。

2.2.4 まとめ

既知の線形モデル $\mathbf{y} = \mathbf{X}\mathbf{w}^* + b^* + \epsilon$ から、$\mathbf{w}^* = (2, -3.4)^\top$、$b^* = 4.2$、$\sigma = 0.01$ を用いて合成データを生成すると、実データにはない 2 つの定量的検証手段が得られる。すなわち、$\frac{1}{2}\sigma^2 = 5 \times 10^{-5}$ という削減不能な損失の下限と、既知の正解パラメータである。y を単一の座標 x_j に対してプロットすると、プロットされていない特徴量の変動が縦方向の広がりに折り畳まれ、ほぼノイズのない平面が幅広い 2D の帯に変わる。最小二乗法の厳密ソルバであっても、$(\mathbf{w}^*, b^*)$ からオーダー $\sigma/\sqrt{n}$（$\approx 3 \times 10^{-4}$、$n = 1{,}000$ の場合）の統計的推定誤差だけずれるため、これが次節における反復的な最適化の精度目標となる。

2.2.5 演習問題

1. [code] **ノイズ下での復元。**●以下の各設定について、2.2.3 節の最小二乗法による当てはめを 1,000 個の独立なデータセット上で繰り返し、最初の重みの推定誤差 $\hat{w}_1 - w_1^*$ の標準偏差を計測せよ。
   1. 1,000 個の訓練例を用いて `noise` を $\{0.001, 0.01, 0.1, 1\}$ の範囲で変化させ、次に訓練例の数 n を $\{10, 100, 1{,}000, 10{,}000\}$ の範囲で `noise=0.01` のもと変化させよ。両方の計測値を、理論予測 $\sigma/\sqrt{n}$ に対して両対数軸でプロットせよ。
   2. n のうち、計測値が $\sigma/\sqrt{n}$ と 10% 以内で一致する最小の値を求め、それより少ない例数ではこの予測の背後にあるどの近似が破綻するかを述べよ。
   3. $\tilde{\mathbf{X}} = [\mathbf{X}, \mathbf{1}]$、$\tilde{\mathbf{w}}^* = (\mathbf{w}^*, b^*)$、$\hat{\tilde{\mathbf{w}}} = (\hat{\mathbf{w}}, \hat{b})$ とおく。正規方程式から $\hat{\tilde{\mathbf{w}}} - \tilde{\mathbf{w}}^* = (\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^\top\epsilon$ を示し、そこから $\mathrm{Cov}(\hat{\tilde{\mathbf{w}}}) = \sigma^2(\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}})^{-1}$ を導出せよ。次に、相関を持つ特徴量 $x_2 = \rho x_1 + \sqrt{1-\rho^2}\, z$（ただし x_1 と z は独立な標準正規分布に従う）を $\rho \in \{0, 0.9, 0.99\}$ に対して生成せよ。この式から $\hat{w}_1$ の標準誤差を予測して数値的に確認し、重みが不正確であるにもかかわらず、同じ分布から得られた新しい例に対する予測 $\hat{\tilde{\mathbf{w}}}^\top \tilde{\mathbf{x}}$（ただし $\tilde{\mathbf{x}} = (\mathbf{x}, 1)$）が正確であり続ける理由を説明せよ。
2. [code] **適合しないモデル。**●すべてのラベルに項 $c\,x_1^2$（$c = 0.5$）を加えてラベルが特徴量の線形関数に従わないようにし、本節と同様に最小二乗法で線形モデルを当てはめよ。

1. $\mathbb{E}[x_1^2] = 1$ および $\mathbb{E}[x_1^3] = 0$ を用いて、$\hat{w}_1$、$\hat{w}_2$、$\hat{b}$ のうちどれが真の値から離れるか、またどの程度離れるかを予測せよ。
2. $n \in \{100, 1{,}000, 10{,}000\}$ 個の訓練例に対してその予測を確認せよ。n が大きくなるにつれて縮小する推定誤差はどれで、縮小しないものはどれか。また、縮小する誤差は $\sigma/\sqrt{n}$ と比べてどうか。
3. 標準誤差の約 3 倍にあたる 10^{-3} を超えるパラメータ誤差を検出するテストは、本節のデータ上では正しい実装を合格と判定する。このデータ上では、正しい実装に欠陥があると判定するだろうか。モデルがデータに適合していないとき、このようなテストの合否から何がわかるかを説明せよ。

3. **損失の下限とその変動**。次節では、1,000 個の検証例にわたる $\frac{1}{2}(\hat{y} - y)^2$ の平均である検証損失によって訓練の実行結果を評価する。
   1. 真のパラメータを持つモデルが新しい例に対して期待損失 $\sigma^2/2$ を持ち、他のいかなるパラメータもそれより小さい期待損失を持たないことを示せ。これは、正しい訓練の実行が最終的に落ち着く下限である。
   2. $\epsilon \sim \mathcal{N}(0, \sigma^2)$ に対して $\mathbb{E}[\epsilon^4] = 3\sigma^4$ が成り立つ。1 つの例の損失 $\frac{1}{2}\epsilon^2$ の標準偏差を計算し、そこから $n' = 1{,}000$ 個の例にわたる真のパラメータの検証損失の標準誤差を、下限に対する比として求めよ。
   3. [code] 多数の独立な検証セット上で真のパラメータを評価し、両方の数値を確かめよ。次に、小問 (a) の結果を用いて、期待損失を検証損失の標準誤差 1 つ分だけ増加させる最小のパラメータ誤差を求め、前の演習問題におけるパラメータ誤差の閾値 10^{-3} と比較せよ。

2.3 線形回帰のスクラッチ実装

線形回帰の理論（2.1 節）と、既知の正解パラメータを持つ合成データセット（2.2 節）が揃ったところで、いよいよこの両方をコードに落とし込んでいこう。本節では、基本的なテンソル演算と自動微分（automatic differentiation）のみに頼り、パラメータの初期化、線形の順伝播、二乗損失、自動微分、ミニバッチのイテレータ、SGD の更新ループといった訓練パイプラインのすべての構成要素をスクラッチで構築する。各ステップを手書きすることでブラックボックス化された仕組みが取り払われ、勾配がどのように流れ、学習率などのハイパーパラメータが収束をどのように左右するかが明確に見えてくる。全体の仕組みが明らかになった後、2.4 節ではこれらの構成要素を再利用可能なクラス（`Module`、`DataModule`、`Trainer`）へと整理し、さらに 2.5 節ではフレームワークの層、組み込みの損失関数、オプティマイザを用いて同じモデルをわずか数行で書き直す。

2.3.1 モデル、損失、勾配

まず、2,000 個の合成データ例（訓練用に 1,000 個、検証用に 1,000 個）を、真のパラメータ $\mathbf{w}^* = (2, -3.4)^\top$、$b^* = 4.2$、ノイズ $\sigma = 0.01$ で生成しよう。

```
w_star, b_star, noise = d2l.tensor([2, -3.4]), 4.2, 0.01
X, y = d2l.synthetic_data(w_star, b_star, noise, 2000, d2l.get_key())
X_train, y_train, X_val, y_val = X[:1000], y[:1000], X[1000:], y[1000:]
```

2.3.1.1 パラメータ

このモデルは、重みベクトル $\mathbf{w} \in \mathbb{R}^2$ とスカラーのバイアス $b \in \mathbb{R}$ を持つ。重みは $\mathcal{N}(0, 0.01^2)$ からの小さな乱数で初期化し、バイアスは 0 に設定する（線形回帰ではすべて 0 から始めても同

様に機能するが、これについては演習問題 1 を参照のこと。ただし、第 4 章で隠れ層を積み重ねるようになると、ランダムな重みで対称性を破ることが不可欠になる)。

JAX の配列はイミュータブルな値である。jax.grad は配列上のフラグを書き換えることなく、純粋関数をその引数に関して直接微分する。

```
def init_params(sigma=0.01):
    return jax.random.normal(d2l.get_key(), (2, 1)) * sigma, jnp.zeros(1)
```

2.3.1.2 順伝播

ミニバッチの計画行列（design matrix）$\mathbf{X} \in \mathbb{R}^{|\mathcal{B}| \times d}$、パラメータ $\mathbf{w} \in \mathbb{R}^{d \times 1}$、およびスカラーのバイアス b が与えられたとき、線形予測 $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$ は 1 回の行列とベクトルの積にブロードキャストによる加算を足し合わせたものになる。

```
def linreg(X, w, b):
    """The linear regression model."""
    return d2l.matmul(X, w) + b
```

2.3.1.3 損失

次に、平均二乗誤差（mean squared error）$L(\mathbf{w}, b) = \frac{1}{2|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\hat{y}^{(i)} - y^{(i)})^2$ を実装する。y を列ベクトル (-1, 1) に変形して y_hat の形状に揃える。もし一方のテンソルの形状が (n, 1) で他方が (n,) だった場合、両者を減算すると暗黙のうちにブロードキャストされ、すべてのペアごとの差からなる (n, n) 行列になってしまう。このバグが厄介なのは、訓練自体はエラーにならずに進んでしまう点である。n^2 個すべての二乗差の平均は、すべての予測がラベルの平均に等しいときに最小となるため、SGD は $\mathbf{w}$ を $\mathbf{0}$ へ、b を $\bar{y}$ へと向かわせ、損失はラベルの分散の半分（このデータでは $\frac{1}{2}(2^2 + 3.4^2) \approx 7.8$）まで滑らかに減少する。したがって、損失曲線が下がっているだけでは、コードが正しいことの弱い証拠にしかならない。本節の後半では、損失が到達すべき値に達しているか、またパラメータが真の値と一致しているかを検証する。

```
def squared_loss(y_hat, y):
    """Half the squared error, averaged over the minibatch."""
    return d2l.reduce_mean((y_hat - d2l.reshape(y, (-1, 1))) ** 2 / 2)
```

2.3.1.4 勾配

自動微分に処理を任せる前に、32 個の例からなる 1 つのミニバッチを用いて、自動微分が 2.1.2.3 節で手計算により導出した次の式と同じ勾配を算出することを確認しておこう。

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{1}{|\mathcal{B}|} \mathbf{X}^\top (\hat{\mathbf{y}} - \mathbf{y}), \qquad \frac{\partial L}{\partial b} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (\hat{y}^{(i)} - y^{(i)}). \tag{2.17}$$

最初の 32 個の訓練例 (Xb, yb) をスライスして取り出し、解析的な勾配と自動微分による勾配を比較してみよう。

```
Xb, yb = X_train[:32], y_train[:32]
w, b = init_params()
```

```
loss_wb = lambda w, b: squared_loss(linreg(Xb, w, b), yb)
grad_w, grad_b = jax.grad(loss_wb, argnums=(0, 1))(w, b)
r = linreg(Xb, w, b) - yb                    # One error per example
by_hand = jnp.concatenate([Xb.T @ r, r.sum(0, keepdims=True)]) / len(yb)
by_autograd = jnp.concatenate([grad_w, grad_b[None]])
print('gradient:', d2l.numpy(by_autograd).reshape(-1).round(4))
print(f'largest difference: {float(abs(by_hand - by_autograd).max()):.1g}')
```

```
gradient: [-1.2946  4.0718 -4.8165]
largest difference: 2e-07
```

要素ごとの最大の差は 0（高々 $\sim 10^{-7}$、すなわち `float32` の丸め精度）であり、autograd が期待どおりのミニバッチ勾配を計算していることが確認できる。ここからは、順伝播と損失の微分をフレームワークに自動で任せることにする。

2.3.2 スクラッチからの訓練

2.3.2.1 更新

各ステップでは、$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta\,(\nabla_{\mathbf{w}} L, \nabla_b L)$ のようにパラメータをその勾配と逆方向に更新する。このとき、パラメータ更新そのものを計算グラフ（computational graph）に記録しないよう注意する。

2.3.2.2 ミニバッチ

データセット全体をサイズ $|\mathcal{B}|$ のランダムなミニバッチで一巡するため、ジェネレータ `minibatches` は各エポックの開始時にインデックス $0, \dots, n-1$ の新しいランダムな順列を生成し、サイズ `batch_size` の連続するスライスを順に返す（演習問題 2）。

```
def minibatches(X, y, batch_size):
    """Minibatches of (X, y), in a new random order on every call."""
    indices = np.random.permutation(len(X))
    for i in range(0, len(indices), batch_size):
        batch = indices[i : i + batch_size]
        yield X[batch], y[batch]
```

1,000 個の訓練例と `batch_size=32` の場合、1 エポックで $\lceil 1000/32 \rceil = 32$ 個のミニバッチが得られる。内訳は、フルサイズのバッチが 31 個（各 32 例）と、端数のバッチが 1 個（8 例）である。このジェネレータを 1 回走査して中身を確認し、誤って**同じ**ジェネレータを 2 回目に反復処理するとどうなるか見てみよう。

```
loader = minibatches(X_train, y_train, 32)
sizes = [len(y) for X, y in loader]
print(f'{len(sizes)} minibatches, the last of size {sizes[-1]}')
print(f'second pass: {len([y for X, y in loader])} minibatches')
```

```
32 minibatches, the last of size 8
second pass: 0 minibatches
```

返される各ペア (Xb, yb) の形状は (32, 2) と (32, 1) である。1 回目の走査（1 **エポック**）では期待どおり 32 個すべてのミニバッチが返されるが、2 回目の走査では 0 個になる。Python のジェネレータは使い切りであり、一度消費されると何も警告を出さずに空になる。訓練ループで各エポックの開始時に毎回新しく minibatches(...) を呼び出すのはこのためである。

2.3.2.3 訓練ループ

ここで、完全な訓練ループを train(lr, num_epochs, batch_size) として組み立てよう。各エポックでは minibatches(X, y, batch_size) を反復処理し、ミニバッチの損失を計算して w と b に関して微分し、SGD のステップを適用する。

```
@jax.jit
def sgd_step(w, b, X, y, lr):
    loss_wb = lambda w, b: squared_loss(linreg(X, w, b), y)  # forward, loss
    grad_w, grad_b = jax.grad(loss_wb, argnums=(0, 1))(w, b)  # backward
    return w - lr * grad_w, b - lr * grad_b                   # update

def train(lr, num_epochs=10, batch_size=32):
    w, b = init_params()
    val_losses = []
    for epoch in range(num_epochs):
        for Xb, yb in minibatches(X_train, y_train, batch_size):
            w, b = sgd_step(w, b, Xb, yb, lr)
        val_losses.append(float(squared_loss(linreg(X_val, w, b), y_val)))
    return w, b, val_losses
```

中核となる訓練ループはわずか 12 行ほどのコードにすぎず、この同じ 4 つのステップ（順伝播 → 損失 → 逆伝播 → パラメータ更新、Figure 2.8 に図示）が、本書の残りの章で扱うあらゆる深層ネットワークを訓練する。

2.3.3 実装の検証

2.3.3.1 損失の期待値

訓練が成功したとき、損失はどこまで小さくなるべきだろうか。真のパラメータ $(\mathbf{w}^*, b^*)$ においてさえ、残差はガウスノイズ $\hat{y}^{(i)} - y^{(i)} = -\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ に等しいため、損失の期待値は次の**削減不能なノイズフロア**（irreducible noise floor）となる。

$$\mathbb{E}\left[L(\mathbf{w}^*, b^*)\right] = \frac{1}{2}\,\mathbb{E}\left[\epsilon^2\right] = \frac{1}{2}\sigma^2 = \frac{1}{2}(0.01)^2 = 0.00005. \tag{2.18}$$

対照的に、初期化時点（$\mathbf{w} \approx \mathbf{0}, b = 0$）での残差は $-y = -(\mathbf{x}^\top \mathbf{w}^* + b^* + \epsilon)$ であり、その平均は $-b^* = -4.2$、分散は $\|\mathbf{w}^*\|_2^2 + \sigma^2 = 2^2 + (-3.4)^2 + 0.01^2 = 15.5601$ である。したがって、初期損失の期待値は次のようになる。

$$\mathbb{E}\left[L(\mathbf{0}, 0)\right] = \frac{1}{2}\left((b^*)^2 + \|\mathbf{w}^*\|_2^2 + \sigma^2\right) = \frac{1}{2}(17.64 + 15.5601) \approx 16.6. \tag{2.19}$$

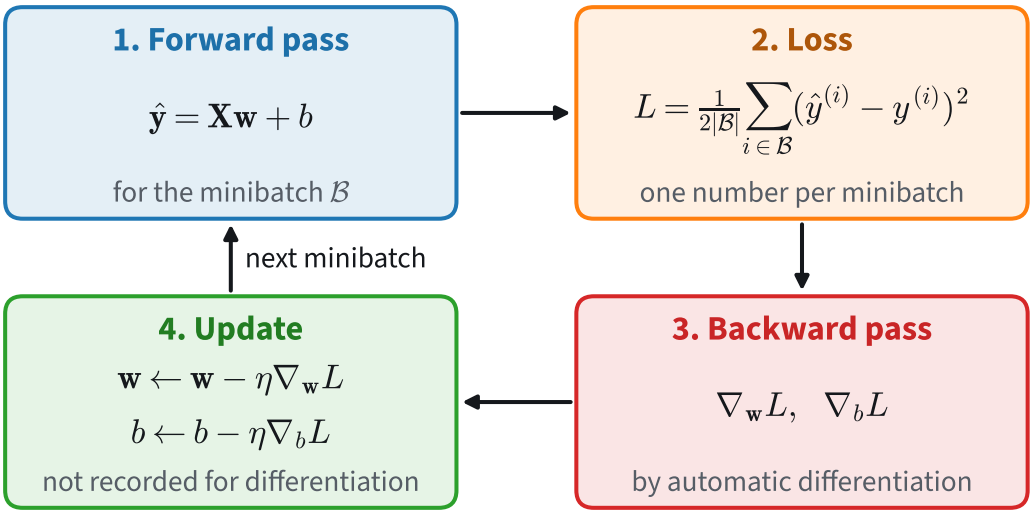


Figure 2.8: ミニバッチ SGD の 1 ステップ。順伝播でミニバッチに対する予測を計算し、損失でそれを 1 つの数値に集約し、逆伝播でその数値をパラメータに関して微分し、更新ステップでパラメータを勾配と逆方向に動かす。その後、次のミニバッチでこのサイクルを繰り返す。

したがって、訓練を通じて損失は約 16.6 からノイズフロアの 0.00005 まで、5 桁にわたって急落すると予想される。

同じ計算から、損失がどのくらい**速く**減少するかも予測できる。任意のパラメータにおいて、残差は $r = \mathbf{x}^\top(\mathbf{w} - \mathbf{w}^*) + (b - b^*) - \epsilon$ となる。2 つの特徴量は互いに独立に標準正規分布に従い、ノイズもその両方と独立であるため、すべての交差項の期待値は 0 となり、次式が成り立つ。

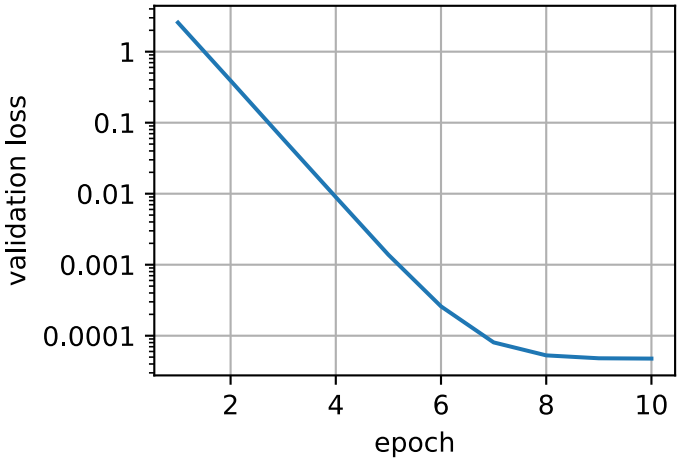
$$\mathbb{E}\left[L(\mathbf{w}, b)\right] = \frac{1}{2}\|\mathbf{w} - \mathbf{w}^*\|_2^2 + \frac{1}{2}(b - b^*)^2 + \frac{1}{2}\sigma^2. \tag{2.20}$$

この損失の期待値の $(\mathbf{w}, b)$ に関する勾配は、単に**パラメータ誤差**（parameter error）$(\mathbf{w} - \mathbf{w}^*, b - b^*)$ となる。そのため、サイズ η の勾配ステップによってパラメータ誤差は $1 - \eta$ 倍に縮小し、ノイズフロアを上回る**過剰損失**（excess loss）は $(1 - \eta)^2$ 倍に縮小する。ミニバッチ勾配が期待勾配と一致するのは平均的な意味においてのみであるため、これは厳密な恒等式ではなく近似的な軌跡を与える。$\eta = 0.03$ のとき、32 回のミニバッチ更新からなる 1 エポックで、過剰損失は約 $1/(1 - 0.03)^{64} \approx 7$ 分の 1 に減少するはずである。

2.3.3.2 訓練の実行

`train` を 10 エポックにわたり学習率 $\eta = 0.03$、ミニバッチサイズ $|\mathcal{B}| = 32$ で実行し、各エポック後の検証損失を対数スケールでプロットしてみよう。

```
w, b, val_losses = train(lr=0.03)
d2l.plot(list(range(1, 11)), val_losses, xlabel='epoch',
         ylabel='validation loss', yscale='log')
```



予想どおり、検証損失は直線に沿って急落し、エポックごとに約 7× の割合で減少した後、7〜8 エポック目あたりで予測された下限である 5×10^{-5} 付近に収まって横ばいになる。最初の 7 エポックにわたり、実測された過剰損失を閉形式による予測とエポックごとに比較してみよう。

```
floor = noise ** 2 / 2                              # sigma^2 / 2
excess = (float(d2l.reduce_sum(w_star ** 2)) + b_star ** 2) / 2  # 16.6
for epoch, loss in enumerate(val_losses[:7], start=1):
    excess *= (1 - 0.03) ** (2 * 32)                # 32 steps per epoch
    print(f'epoch {epoch}: measured {loss - floor:.1e}, '
          f'predicted {excess:.1e}')
```

```
epoch 1: measured 2.6e+00, predicted 2.4e+00
epoch 2: measured 3.9e-01, predicted 3.4e-01
epoch 3: measured 5.9e-02, predicted 4.8e-02
epoch 4: measured 8.9e-03, predicted 6.8e-03
epoch 5: measured 1.3e-03, predicted 9.7e-04
epoch 6: measured 2.1e-04, predicted 1.4e-04
epoch 7: measured 3.1e-05, predicted 2.0e-05
```

実測された過剰損失と予測された過剰損失は、序盤のエポックでは緊密に一致して推移し、その後少しずつ離れていくものの、7 エポック目でも高々 3〜4 倍程度の控えめな差にとどまる。なぜこのようなわずかなドリフトが生じるのだろうか。ここでの理想化した予測では、標本共分散行列（sample covariance matrix）$\mathbf{H} = \frac{1}{n}\sum_i \tilde{\mathbf{x}}^{(i)}(\tilde{\mathbf{x}}^{(i)})^\top$（演習問題 1 を参照）が厳密に単位行列に等しいと仮定していた。1,000 個の例からなる有限標本では、その固有値はおよそ 10% の範囲内で 1 の近くに収まっており、数エポックにわたってこの小さなずれが積み重なることで、実際の訓練は理想化した予測よりわずかに先行したり遅れたりするのである。減衰率とノイズフロアはいずれも理論と一致している。

2.3.3.3 復元されたパラメータ

このデータセットは合成データであるため、学習された $(\mathbf{w}, b)$ を真のパラメータ $\mathbf{w}^* = (2, -3.4)^\top$ および $b^* = 4.2$ と直接比較することもできる。

```
w_error = d2l.numpy(d2l.reshape(w, w_star.shape) - w_star)
b_error = d2l.numpy(b - b_star)
print('error in w:', ', '.join(f'{e:.1e}' for e in w_error))
print('error in b:', ', '.join(f'{e:.1e}' for e in b_error))
```

```
error in w: -1.9e-04, 6.9e-04
error in b: -7.1e-04
```

重みとバイアスはいずれも 1,000 分の 1 以内の誤差で復元されており、これは統計的推定の下限 $\sigma/\sqrt{n} \approx 3 \times 10^{-4}$（2.2 節で最小二乗法のソルバを用いて確認した値）と一致している。

2.3.3.4 学習率

学習率の選び方が適切でなかった場合はどうなるだろうか。まったく同じ 10 エポックのループを、3 桁にわたる 4 つの学習率 $\eta \in \{0.003, 0.03, 0.3, 3\}$ で実行し、それぞれの最終的な検証損失を表示してみよう（演習問題 1）。

```
for lr in [0.003, 0.03, 0.3, 3]:
    val_loss = train(lr)[2][-1]
    print(f'lr = {lr:<6} validation loss: {val_loss:.1e}')
```

```
lr = 0.003  validation loss: 2.7e+00
lr = 0.03   validation loss: 4.8e-05
lr = 0.3    validation loss: 5.7e-05
lr = 3      validation loss: nan
```

結果はすべてのケースで理論と一致している。$\eta = 0.003$ では、誤差が 1 ステップあたり 0.997 倍にしかならないため、10 エポック後の損失は 2 から 3 付近にとどまる（$16.6 \times 0.997^{640} \approx 2.4$ のすぐ近くである）。$\eta = 0.3$ では、最初の 1 エポックのうちにモデルが一気にノイズフロアまで到達する。そして $\eta = 3$ では、更新のたびに誤差が符号を反転させながら約 2 倍に膨れ上がり、たちまちオーバーフローして nan になる。なお、安定性の上限である 2 という値は SGD の普遍的な定数ではなく、ここでの**データのスケーリング**に由来する性質であることに注意したい。これは、特徴量の分散が 1 であることから生じている。ある特徴量を $s > 1$ 倍にスケーリングすると、対応する重み方向の損失の曲率は s^2 倍になり、安定な最大学習率は約 $2/s^2$ まで縮小する。これが、訓練前に入力特徴量を標準化する実用上の大きな理由の 1 つである（1.2 節）。

続く 2.4 節では、今構築した訓練の仕組みを再利用可能なオブジェクト指向のクラス（`Module`、`DataModule`、`Trainer`）にまとめ、2.5 節では手書きのテンソル、損失、更新処理をフレームワークの既製コンポーネントに置き換える方法を示す。

2.3.4 まとめ

完全な訓練ループに必要な基本要素は、パラメータ化された順伝播 $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$、スカラーの損失 $\frac{1}{2|\mathcal{B}|}\|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$、リバースモード自動微分（reverse-mode automatic differentiation）、そして勾配の記録対象外で実行するミニバッチ SGD の更新という 4 つだけである。損失曲線が下がっているだけではコードにバグがないことの証明にはならないため（暗黙の形状の不一致があっても、損失はラベルの分散の半分に向かって下がりうる）、合成データを用いて実装を定量的に検証した。すなわち、検証損失は予測されたエポックあたり $(1-\eta)^{64}$ の割合でノイズフロア $\frac{1}{2}\sigma^2 = 5 \times 10^{-5}$ まで指数関数的に減衰し、学習されたパラメータは $(\mathbf{w}^*, b^*)$ と $O(\sigma/\sqrt{n})$ の範囲内で一致する。学習率を変えると、小さなステップサイズでは進行が遅く、適度なステップ

サイズでは素早くノイズフロアに到達し、$2/\mu_{\max}$ を超えるステップサイズでは二次関数の谷を飛び越えて発散する。

2.3.5 演習問題

1. [code] **極端な設定での初期化**。`init_params` は標準偏差 `sigma=0.01` で重みを生成する。重みを厳密に 0 で初期化した場合と、`sigma=1000` とした場合のそれぞれについて、訓練がノイズフロアに到達するかどうか予測せよ。Equation 2.20 を用いて、学習率 0.03 のもとで 2 番目のケースが何エポック多く必要になるかを見積もり、その見積もりを検証せよ。

2. [code] **二階導関数**。ミニバッチ損失の 3×3 ヘッセ行列を $\theta = (\mathbf{w}, b)$ に関して手計算で求め、それがどの量に依存するかを述べよ。次に、合成データの固定した 1 つのミニバッチに対して、初期化時点と訓練後の 2 回、`jax.hessian`（または `jax.jacrev(jax.grad(...))`、1.5 節）を用いた自動微分によってヘッセ行列を計算せよ。両方の結果を導出した式および Equation 2.20 の曲率と比較せよ。

3. [code] **安定性とミニバッチサイズ**。●損失の期待値からは、その勾配に沿ったステップは $\eta > 2$ で発散すると予測される。$\tilde{\mathbf{x}} = (\mathbf{x}, 1)$ を、バイアスのために 1 を付加した入力とする。

   1. フルバッチの勾配降下法において、各ステップがパラメータと最小二乗法による解との差に行列 $\mathbf{I} - \eta\mathbf{H}$ を掛けることを示せ。ここで $\mathbf{H}$ は、訓練セット全体での $\tilde{\mathbf{x}}\tilde{\mathbf{x}}^\top$ の平均である。$\mathbf{H}$ の固有値を計算し、安定な最大学習率を予測せよ。
   2. サイズ 1,000、32、1 のミニバッチで訓練し、各実行の更新回数を同じ値（たとえば 320 回）に揃えよ。各ミニバッチサイズについて、任意に選んだグリッド上で最終的な訓練損失が 10^{-3} 未満となる最大の学習率を求めよ。
   3. 単一の例に対する 1 ステップが、パラメータ誤差の $\tilde{\mathbf{x}}$ 方向の成分を $1 - \eta\|\tilde{\mathbf{x}}\|^2$ 倍にすることを示し、$\mathbb{E}\|\tilde{\mathbf{x}}\|^2 = 3$ を用いてサイズ 1 のミニバッチで見られた限界を説明せよ。

4. **なぜ再シャッフルするのか**。`minibatches` は毎エポック新しいランダムな順序を生成する。これによってミニバッチ勾配のどの性質が保たれるかを説明せよ。次に、ラベル順のソートなど、再シャッフルなしのミニバッチ SGD が異なる振る舞いを示す訓練セットの固定順序を 1 つ構成し、1 エポック内および複数エポックにわたる $(\mathbf{w}, b)$ の軌跡を記述せよ。

5. [code] **ロバストな損失**。●`squared_loss` を平均絶対誤差 `d2l.reduce_mean(d2l.abs(y_hat - d2l.reshape(y, (-1, 1))))` に置き換えよ。

   1. 合成データ上で訓練し、復元されたパラメータおよび損失曲線を二乗損失の場合と比較せよ。

   2. 2.1.1.3 節で用いた外れ値と同じく 1 つの訓練ラベルを $y^{(5)} = 10{,}000$ に設定し、両方の損失で訓練せよ。ミニバッチ更新ごとに b を記録して両方の実行について反復回数に対してプロットし、最終的な値を決める要因を含めて二乗損失の曲線のパターンを説明せよ。

   3. 外れ値を含むデータに対して、損失を次のフーバー損失のミニバッチ平均に置き換えよ。

$$l_\delta(y, y') = \begin{cases} \frac{1}{2}(y - y')^2 & \text{if } |y - y'| \le \delta \\ \delta\left(|y - y'| - \frac{\delta}{2}\right) & \text{otherwise,} \end{cases} \tag{2.21}$$

   この二次関数の部分は、本節の二乗損失と一致している。$\delta \in \{1, 10, 10^2, 10^3, 10^4\}$ に対して訓練し、学習されたバイアスと真のバイアスとの距離 $|\widehat{b} - b^*|$（ここで

$b^* = 4.2$ はデータ生成時のバイアス）を δ に対して両対数軸でプロットせよ。l_δ の y' に関する導関数から曲線の形状を説明し、その結果を Figure 2.6 のペナルティ曲線と関連づけよ。

2.4 実装のためのオブジェクト指向設計

2.3 節では、生のテンソルから線形回帰を実装し、1 つのモデルと 1 つのデータセットに直接結び付けられた独立な 12 行の `train` 関数で訓練した。今後、多層パーセプトロン（multilayer perceptron）、畳み込みネットワーク（convolutional network）、リカレントネットワーク、Transformer へと進むにあたり、データローダやエポックのループ、評価指標の平均計算、リアルタイムのプロットを毎節書き直していては、新しいアイデアが反復的な定型処理の下に埋もれてしまう。

モデリングに集中しつつ、訓練ハーネスのすべての行を見通しよく変更可能な状態に保つため、本書では `d2l` ライブラリを 3 つの明快なオブジェクト指向（object-oriented）の抽象化（Figure 2.9）を中心に構成する。これは PyTorch Lightning のようなフレームワークに着想を得たものである (Falcon and The PyTorch Lightning team 2019)。

1. **`Module`（モデル）**：訓練可能なパラメータを保持し、`forward` による順伝播（forward pass）を定義し、`loss` を計算し、`training_step` と `validation_step` を定め、`configure_optimizers` を通じてオプティマイザを返す。
2. **`DataModule`（データ）**：データセットのダウンロード、前処理、分割をカプセル化し、イテラブルな `train_dataloader()` と `val_dataloader()` を公開する。
3. **`Trainer`（訓練ループ）**：任意の `Module` と任意の `DataModule` を接続し、エポックごとのミニバッチ訓練パスと検証パスを実行し、バッチ間で評価指標を平均化して、リアルタイムの学習曲線（learning curve）を `ProgressBoard` 上にプロットする。

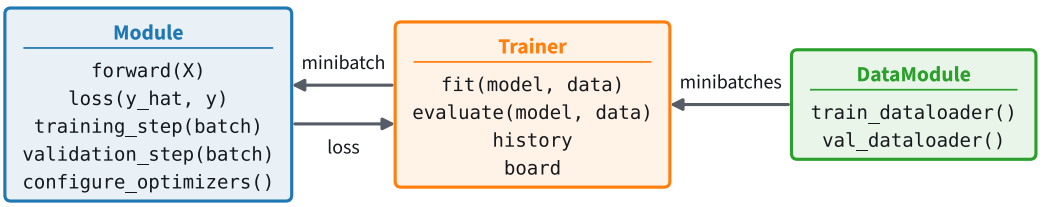


Figure 2.9: `d2l` 訓練ハーネスの中核をなす 3 つのクラス。`DataModule` は訓練用と検証用のミニバッチを供給し、`Module` はモデル・損失・オプティマイザを定義し、`Trainer` は訓練ループと検証ループを駆動する。

```
import collections
```

2.4.1 ユーティリティ

中核となる 3 つのクラスを構築する前に、ノートブック内の定義を読みやすく自己完結した形に保つ 2 つの小さな Python ユーティリティを導入する。

1. **`@add_to_class(Class)`**：教科書やノートブックにおいて、60 行のクラスを 1 つの巨大なコードブロックとして提示すると、各メソッドが登場するたびに説明するのが難しくなる。デコレータ `@add_to_class(Class)` を使えば、まずクラスの骨格だけを定義しておき、後のコードセルで新しいメソッドを段階的に追加できる。

2. **HyperParameters（save_hyperparameters()）**：self.a = a; self.b = b; ... のような記述をすべての __init__ メソッドの先頭に書くのは、純粋なボイラープレートである。HyperParameters を継承して self.save_hyperparameters() を呼び出すと、呼び出し元のフレームが検査され、すべての __init__ 引数（ignore に指定されたものを除く）が self の属性として自動的に束縛される。

```
def add_to_class(Class):
    """Register functions as methods in created class."""
    def wrapper(obj):
        setattr(Class, obj.__name__, obj)
        return obj
    return wrapper
```

簡単なクラス A と、すでにインスタンス化されたオブジェクト a で試してみよう。

```
class A:
    def __init__(self):
        self.b = 1

a = A()
```

次に、別のセルで新しいメソッド do を追加する。Python はメソッドを呼び出し時にクラス上で解決するため、既存のインスタンス a からも直ちに do を利用できる。

```
@add_to_class(A)
def do(self):
    print('Instance attribute "b" is', self.b)

a.do()
```

```
Instance attribute "b" is 1
```

2 つ目の補助クラスは、コンストラクタのボイラープレートを取り除くものである。これがないと、コンストラクタは self.lr = lr や self.batch_size = batch_size のような反復的な代入で埋め尽くされてしまう。深層学習におけるコンストラクタ引数の大半はハイパーパラメータであるため、d2l.HyperParameters を継承して self.save_hyperparameters() を呼び出すことで、すべてのコンストラクタ引数（ignore に指定されたものを除く）を self の属性として自動的に束縛する。内部では Python の inspect モジュールを使って呼び出し元のスタックフレームを検査しており、その実装はオンライン付録「ユーティリティ関数とクラス」にある。

```
class B(d2l.HyperParameters):
    def __init__(self, a, b, c):
        self.save_hyperparameters(ignore=['c'])
        print('self.a =', self.a, 'self.b =', self.b)
        print('self.c exists:', hasattr(self, 'c'))

b = B(a=1, b=2, c=3)
```

```
self.a = 1 self.b = 2
self.c exists: False
```

本節の残りで構築するすべてのクラスは HyperParameters を継承する。

2.4.2 モデル

Module クラスは、本書で実装するすべてのモデルの基底クラスである。このクラスは、各フレームワーク固有のニューラルネットワーク用コンテナと d2l.HyperParameters の両方を継承する。デフォルトの動作は次のとおりである。

- model(X) を呼び出すと model.forward(X) が実行される。
- training_step(batch) は、特徴量 batch[:-1] をモデルに通し、予測をラベル batch[-1] と比較して、ミニバッチの平均損失を返す（後の章のモデルでは、'loss' を含む評価指標の辞書を返す場合もある）。
- validation_step(batch) は、検証データのミニバッチに対して同じ損失を計算する（なお 3.3 節では、Classifier が validation_step をオーバーライドして 'loss' と 'acc' の両方を報告する）。
- 補助関数 as_metrics は、単一のスカラーまたは辞書の戻り値を評価指標の辞書へと正規化し、Trainer が任意の名前付き量を統一的に記録できるようにする。

JAX では、このコンテナは Flax の nnx.Module である。

```
class Module(nnx.Module, d2l.HyperParameters):
    """The base class of models."""
    def loss(self, y_hat, y):
        raise NotImplementedError

    def forward(self, X, *args, **kwargs):
        assert hasattr(self, 'net'), 'Neural network is not defined'
        return self.net(X, *args, **kwargs)

    def __call__(self, X, *args, **kwargs):
        return self.forward(X, *args, **kwargs)

    def training_step(self, batch):
        """The mean loss on one minibatch, or a dict of batch-mean metrics
        with key 'loss'."""
        return self.loss(self(*batch[:-1]), batch[-1])

    def validation_step(self, batch):
        """The mean loss on one minibatch of validation data, or a dict of
        batch-mean metrics with key 'loss'."""
        return self.loss(self(*batch[:-1]), batch[-1])

    def configure_optimizers(self):
        raise NotImplementedError
```

慣例として、すべてのミニバッチはタプルであり、末尾の要素がラベルテンソル、それより前の要素がモデルの入力である。training_step と validation_step はどちらも *batch[:-1] をアンパックして self(...) に渡し、batch[-1] に対する損失を計算して、その結果を単に**返す**だけである（それ自体では出力、プロット、ログ記録を一切行わない）。モデルがトレーナの存在をまったく意識しないようにしているのは意図的な設計上の選択であり、これによりどのようなカスタムループからでも隠れた副作用なしに Module を訓練・評価できる。

本章では、各ステップは単一のスカラー、すなわちミニバッチの平均損失を返す。3.3 節以降で

は、その代わりに評価指標名をミニバッチ平均に対応付ける辞書（常に 'loss' を含み、'acc' などの量も併せ持つ）を返す場合がある。小さな補助関数 as_metrics は、単なるスカラーの損失を {'loss': out} へと正規化し、トレーナが両者を統一的に扱えるようにする。

```
def as_metrics(out):
    """A step returns a loss or a dict of metrics; return a dict."""
    return out if isinstance(out, dict) else {'loss': out}
```

2.4.2.1 Module としての線形回帰

フレームワークの組み込み層を使う前に Module が実際にどう動くかを確認するため、スクラッチ実装した線形回帰のパラメータ $(\mathbf{w}, b)$、順伝播、二乗損失（squared loss）、およびスクラッチ実装の SGD 更新器を LinearRegressionScratch(d2l.Module) にまとめてみよう。

```
class LinearRegressionScratch(d2l.Module):
    """The linear regression model implemented from scratch."""
    def __init__(self, num_inputs, lr, sigma=0.01, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.w = nnx.Param(
            rngs.params.normal((num_inputs, 1)) * sigma)
        self.b = nnx.Param(jnp.zeros(1))
```

次に、forward と loss を追加する。これらは 2.3 節の linreg および squared_loss と同一であり、モデルインスタンスから self.w と self.b を読み取るだけである。

```
@d2l.add_to_class(LinearRegressionScratch)
def forward(self, X):
    return d2l.matmul(X, self.w) + self.b
```

```
@d2l.add_to_class(LinearRegressionScratch)
def loss(self, y_hat, y):
    l = (y_hat - d2l.reshape(y, (-1, 1))) ** 2 / 2
    return d2l.reduce_mean(l)
```

続いて、スクラッチ実装したミニバッチ SGD の更新則を再利用可能な SGD(d2l.HyperParameters) クラスにまとめ、LinearRegressionScratch.configure_optimizers から返すようにする。

```
class SGD(d2l.HyperParameters):
    """Minibatch stochastic gradient descent."""
    # An optax optimizer is a pair of pure functions: `init` creates the
    # optimizer state, and `update` turns gradients into parameter updates.
    def __init__(self, lr):
        self.save_hyperparameters()

    def init(self, params):
        del params  # SGD keeps no state
        return optax.EmptyState()
```

```
    def update(self, updates, state, params=None):
        del params
        updates = jax.tree_util.tree_map(lambda g: -self.lr * g, updates)
        return updates, state
```

最後に、configure_optimizers でモデルと SGD オプティマイザを結び付ける。

```
@d2l.add_to_class(LinearRegressionScratch)
def configure_optimizers(self):
    return SGD(self.lr)
```

2.4.3 データ

基底クラス DataModule は、データセットにおいて Module に対応するものである。具体的な各データセットのサブクラスは train_dataloader() と val_dataloader() を実装する（通常はどちらも get_dataloader(train=True/False) に処理を委譲する）。

```
class DataModule(d2l.HyperParameters):
    """The base class of data."""
    def __init__(self, root='../data', num_workers=4):
        self.save_hyperparameters()

    def get_dataloader(self, train):
        raise NotImplementedError

    def train_dataloader(self):
        return self.get_dataloader(train=True)

    def val_dataloader(self):
        return self.get_dataloader(train=False)
```

2.2 節で作成した合成回帰データの生成器を SyntheticRegressionData(d2l.DataModule) にまとめよう。このクラスはあらかじめ num_train + num_val 個の例（デフォルトでは訓練例 1,000 個、検証例 1,000 個）を生成しておくため、Trainer は毎エポック終了後に訓練損失（training loss）とホールドアウトされた検証損失（validation loss）の両方を追跡できる。

```
class SyntheticRegressionData(d2l.DataModule):
    """Synthetic data for linear regression."""
    def __init__(self, w, b, noise=0.01, num_train=1000, num_val=1000,
                 batch_size=32, key=None):
        super().__init__()
        self.save_hyperparameters()
        key = d2l.get_key() if key is None else key  # A new dataset per run
        self.X, self.y = d2l.synthetic_data(w, b, noise, num_train + num_val,
                                            key)
```

2.4.3.1 各エポック用のローダ

get_dataloader の内部では、エポックごとに 1 回反復でき、各エポックの開始時に訓練分割をシャッフルし直す一方で、検証分割は決定論的な順序で読み出すオブジェクトが必要になる。get_tensorloader メソッドは、分割されたテンソルからこのオブジェクトを構築する。

このオブジェクトは、2.3 節のスライスごとのインデックス順列ロジックを包む ArrayLoader クラスとしてスクラッチで記述する。

```
class ArrayLoader:
    """Minibatches of in-memory arrays, reshuffled each pass if shuffle."""
    def __init__(self, arrays, batch_size, shuffle=False, drop_last=False,
                 key=None):
        self.arrays = tuple(np.asarray(a) for a in arrays)  # On the host
        self.batch_size, self.shuffle = batch_size, shuffle
        self.drop_last = drop_last
        if shuffle and key is None:  # A new order on each run
            key = d2l.get_key()
        self.key = key

    def __len__(self):
        n = len(self.arrays[0])
        if self.drop_last:
            return n // self.batch_size
        return (n + self.batch_size - 1) // self.batch_size

    def __iter__(self):
        n = len(self.arrays[0])
        if self.shuffle:  # A new key, and hence a new order, on every pass
            self.key, key = jax.random.split(self.key)
            order = np.asarray(jax.random.permutation(key, n))
        else:
            order = np.arange(n)
        for i in range(len(self)):
            start = i * self.batch_size
            batch_indices = order[start : start + self.batch_size]
            yield tuple(a[batch_indices] for a in self.arrays)
```

get_tensorloader は指定された分割をスライスし、テンソルを ArrayLoader で包む。

```
@d2l.add_to_class(d2l.DataModule)
def get_tensorloader(self, tensors, train, indices=slice(0, None)):
    tensors = tuple(a[indices] for a in tensors)
    return ArrayLoader(tensors, self.batch_size, shuffle=train)
```

これで SyntheticRegressionData.get_dataloader はわずか 2 行になり、訓練分割または検証分割のいずれかをスライスして get_tensorloader に委譲するだけである。

```
@d2l.add_to_class(SyntheticRegressionData)
def get_dataloader(self, train):
    i = slice(0, self.num_train) if train else slice(self.num_train, None)
    return self.get_tensorloader((self.X, self.y), train, i)
```

この合成データセットをインスタンス化し、先ほど単純なジェネレータがつまずいた 2 パスのテストをもう一度試してみよう。

```
data = SyntheticRegressionData(w=d2l.tensor([2, -3.4]), b=4.2)
```

```
loader = data.train_dataloader()
sizes = [len(y) for X, y in loader]
print(f'{len(sizes)} minibatches, the last of size {sizes[-1]}')
print(f'second pass: {len([y for X, y in loader])} minibatches')
for _ in range(2):  # Each new pass starts in a new random order
    X, y = next(iter(loader))
    print('a new pass starts with labels', d2l.numpy(y[:3, 0]).round(2))
```

```
32 minibatches, the last of size 8
second pass: 32 minibatches
a new pass starts with labels [6.59 1.05 1.47]
a new pass starts with labels [6.62 7.7  4.22]
```

今度はどちらのパスでも全 32 個のミニバッチ（末尾の 8 例の端数を含む）が得られ、パスごとに新しいランダムな順序で始まる。さらに都合がよいことに、データセットを反復処理することなく `len(loader)` を直接取得できる。

```
len(loader)
```

```
32
```

2.4.4 訓練

2.4.4.1 訓練ループと履歴

次に、`Trainer` を構築する。これは `Module` と `DataModule` を結び付けるエンジンである。`trainer.fit(model, data)` を呼び出すと、データローダを準備し、モデルとそのオプティマイザを設定して、`max_epochs` 回のパスを実行する。各エポックでは、`fit_epoch` が訓練ミニバッチを一通り処理し（パラメータを更新しつつ訓練の評価指標を平均化する）、続いて検証ミニバッチに対して `validate` を呼び出し、エポック平均を `self.history` に記録して `ProgressBoard` のリアルタイムプロットを更新する。

```
class Trainer(d2l.HyperParameters):
    """Fit a Module to a DataModule, keeping a per-epoch history."""
    def __init__(self, max_epochs, num_gpus=0, gradient_clip_val=0):
        self.save_hyperparameters()
        self.board = ProgressBoard(xlabel='epoch', ylabel='loss')

    def prepare_data(self, data):
        self.train_dataloader = data.train_dataloader()
        self.val_dataloader = data.val_dataloader()

    def prepare_model(self, model):
        self.model = model
```

```
    def prepare_optimizer(self, model):
        self.optim = model.configure_optimizers()
```

prepare_model はパラメータを共有するモデルの 2 つのビュー（訓練用と評価用）を作成し、ドロップアウトやバッチ正規化（batch normalization）のような層が各モードで適切に動作するようにする。prepare_optimizer は Optax の勾配変換を nnx.Optimizer で包み、変換構造をキャッシュすることで、繰り返し訓練を行う際にコンパイル済みの JIT カーネルを再利用できるようにする。

```
@d2l.add_to_class(Trainer)
def prepare_model(self, model):
    # Two views that share the parameters: layers such as dropout behave
    # differently in training and in evaluation
    self.model = model
    self.train_model = nnx.view(model, deterministic=False,
                                use_running_average=False,
                                raise_if_not_found=False)
    self.val_model = nnx.view(model, deterministic=True,
                              use_running_average=True,
                              raise_if_not_found=False)


optimizer_cache = {}  # The most recent optax transformation, by its key

# Reusing the transformation lets a later fit of a model with the same
# structure reuse the compiled training step

@d2l.add_to_class(Trainer)
def prepare_optimizer(self, model):
    key = (jax.tree.structure(model), type(model).configure_optimizers)
    if key not in optimizer_cache:
        optimizer_cache.clear()  # One entry: it may refer to a model
        optimizer_cache[key] = model.configure_optimizers()
    self.optim = nnx.Optimizer(model, optimizer_cache[key], wrt=nnx.Param)
```

fit(model, data) メソッドは、2.3 節の train 関数と同じ外側ループを実装する。3 つの準備用フックを実行した後、max_epochs エポックにわたってループを回す。各エポックでは、fit_epoch() が訓練ローダを一通り走査して訓練の評価指標の平均を返し、validate() が検証ローダ（存在する場合）を評価し、record() がその両方を記録・プロットする。

```
@d2l.add_to_class(Trainer)
def fit(self, model, data):
    self.prepare_data(data)
    self.prepare_model(model)
    self.prepare_optimizer(model)
    self.history = collections.defaultdict(list)
    self.board.clear()
    self.board.xlim = [0, self.max_epochs]
    for self.epoch in range(self.max_epochs):
        # The training mean averages over the epoch, so it sits mid-epoch
        self.record('train', self.fit_epoch(), x=self.epoch + 0.5)
        if self.val_dataloader is not None:
```

```
            self.record('val', self.validate(), x=self.epoch + 1)
        self.board.show(final=False)
    self.board.show()

@d2l.add_to_class(Trainer)
def record(self, split, metrics, x):
    for name in sorted(metrics, key=lambda name: (name != 'loss', name)):
        self.history[f'{split}_{name}'].append(metrics[name])
        self.board.draw(x, metrics[name], f'{split}_{name}')
```

なお、`fit` は現在のエポック番号を `self.epoch` として公開し、直接は何も返さない。記録されたすべての曲線は `trainer.history` に格納される。これは分割名と評価指標名（`'train_loss'`、`'val_loss'`、および後に登場する `'val_acc'`）をキーとするリストの辞書である。したがって、`trainer.history['val_loss'][-1]` によって最終的な検証損失を直ちに取得できる。`'loss'` を先頭にソートしておくことで、モデルが異なってもプログレスボード上の曲線の色が常に一貫する。

これら 2 本の曲線は同じ位置には描画されない。その理由を Figure 2.10 に示す。エポック e の訓練損失は、そのエポック内の全ミニバッチステップにわたる平均であり、各ステップはそれぞれ異なるパラメータで計算されるため、`fit` はこれをエポックの中央である $e + \frac{1}{2}$ に描画する。一方、検証損失はそのエポック終了時点のパラメータを用いて 1 度だけ計算され、$e + 1$ に描画される。

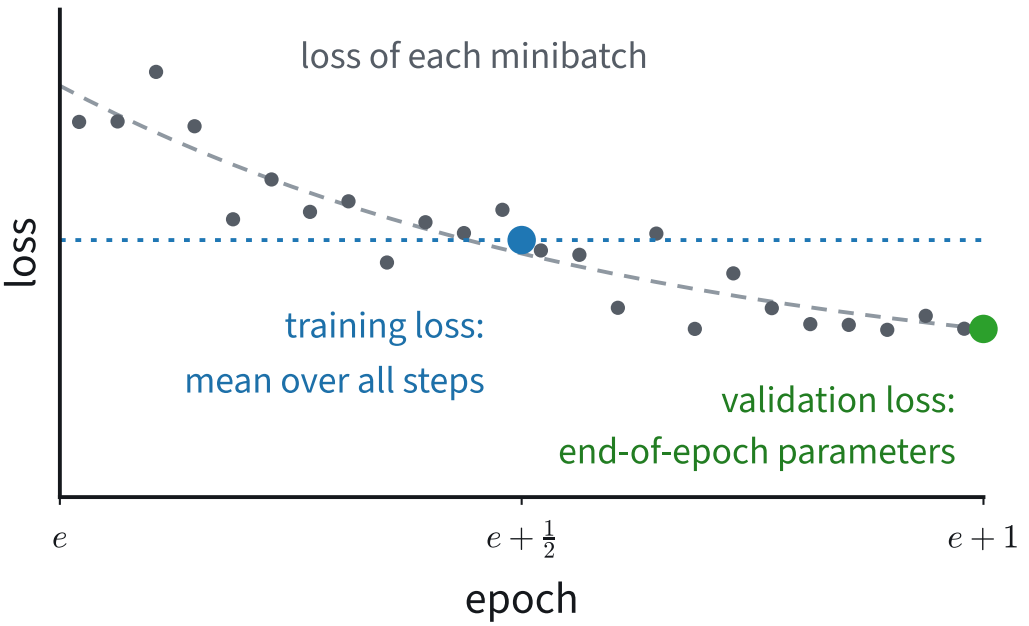


Figure 2.10: プログレスボード上の 1 エポック。パラメータが改善するにつれて個々のミニバッチの損失（灰色）は減少し、その平均である訓練損失が $e + \frac{1}{2}$ に描画される。検証損失はエポック終了時点のパラメータで計算され、$e + 1$ に描画される。

2.4.4.2 エポック平均

エポックの最後のミニバッチは batch_size より小さくなることが多いため（たとえば $1{,}000 = 31 \times 32 + 8$）、バッチごとの平均損失を重みなしで単純平均すると、最後のバッチに含まれる 8 個の例が、それ以前の各バッチに含まれる 32 個の例とまったく同じ発言力を持ってしまう。補助クラス Averager は各バッチの評価指標にそのバッチサイズ n で重み付けして累積するため、avg.mean() は常に分割全体にわたる正確な例単位の平均を返す。

```
class Averager:
    """Example-weighted means of batch metrics, summed on the device."""
    def __init__(self):
        self.sums, self.n = {}, 0

    def add(self, metrics, n):
        for name, value in metrics.items():
            self.sums[name] = self.sums.get(name, 0) + value * n
        self.n += n

    def mean(self):
        """One host synchronization per metric; the loss comes first."""
        names = sorted(self.sums, key=lambda name: (name != 'loss', name))
        return {name: float(self.sums[name]) / self.n for name in names}
```

2.4.4.3 訓練パスと検証パス

次に、validate と fit_epoch は、4 つのフレームワークそれぞれにおいてバッチごとの検証ループと訓練ループを実行する（第 8 章のリカレントネットワークに備え、任意の勾配クリッピング（gradient clipping）用の gradient_clip_val も用意してある）。

```
@d2l.add_to_class(d2l.Trainer)
def prepare_batch(self, batch):
    return batch
```

このトレーナは本書の残りの章すべてであらゆるモデルを実行するため、単純な線形回帰だけに必要なものよりわずかに長くなっている。それでもアルゴリズムの中核は Figure 2.8 とまったく同じ 4 ステップの SGD サイクル（順伝播、損失の計算、逆伝播（backward pass）、パラメータ更新）であり、後の章に向けたフックがいくつか追加されているものの、ここではまだ使われない。

_trainer_train_step と _trainer_validation_step を @nnx.jit でデコレートすることで、順伝播、損失、勾配、オプティマイザの更新が単一の融合された XLA 計算へとコンパイルされる。has_aux=True を渡すことで、nnx.grad は out['loss'] を微分しつつ、勾配とあわせて評価指標の辞書全体も返すことができる。

```
@nnx.jit
def _trainer_train_step(model, optimizer, batch):
    def loss_fn(model):
        out = d2l.as_metrics(model.training_step(batch))
        return out['loss'], out
    grads, out = nnx.grad(loss_fn, has_aux=True)(model)
    optimizer.update(model, grads)
    return out
```

```
@nnx.jit
def _trainer_validation_step(model, batch):
    return d2l.as_metrics(model.validation_step(batch))

@d2l.add_to_class(d2l.Trainer)
def fit_epoch(self):
    metrics = d2l.Averager()
    for batch in map(self.prepare_batch, self.train_dataloader):
        metrics.add(_trainer_train_step(self.train_model, self.optim, batch),
                    len(batch[-1]))
    return metrics.mean()

@d2l.add_to_class(d2l.Trainer)
def validate(self):
    metrics = d2l.Averager()
    for batch in map(self.prepare_batch, self.val_dataloader):
        metrics.add(_trainer_validation_step(self.val_model, batch),
                    len(batch[-1]))
    return metrics.mean()
```

2.4.4.4 プログレスボード

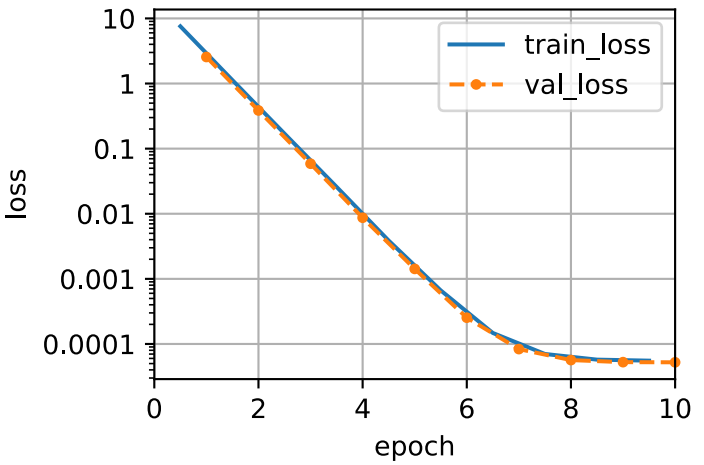
最後に、ProgressBoard はノートブックの実行中にエポックごとの学習曲線をリアルタイムで描画する（描画なしのスイープを実行する際には trainer.board.display = False によって非表示にできる）。

```
class ProgressBoard(d2l.HyperParameters):
    """Curves that grow during training; `show` redraws them."""
    def __init__(self, xlabel=None, ylabel=None, xlim=None, ylim=None,
                 xscale='linear', yscale='linear', figsize=(3.5, 2.5),
                 display=True):
        self.save_hyperparameters()
        self.clear()

    def clear(self):
        """Forget all curves; the next `show` starts a new figure."""
        self.data, self.handle = {}, None

    def draw(self, x, y, label):
        """Append the point (x, y) to the curve called `label`."""
        xs, ys = self.data.setdefault(label, ([], []))
        xs.append(x)
        ys.append(y)

    def show(self, final=True):
        raise NotImplementedError
```

2.4.5 線形回帰の再訓練

新たに組み立てた Module、DataModule、Trainer の各クラスをエンドツーエンドでテストするため、LinearRegressionScratch を SyntheticRegressionData 上で 10 エポック訓練し、復元されたパラメータを確認してみよう。

```
model = LinearRegressionScratch(2, lr=0.03)
trainer = d2l.Trainer(max_epochs=10)
trainer.board.yscale = 'log'
trainer.fit(model, data)
```



検証損失は 2.3 節とまったく同じ軌跡をたどり、エポックごとに約 $7\times$ の倍率で急減して 5×10^{-5} 付近のノイズフロアに達する。訓練曲線もその真上に重なっている（パラメータがわずか 3 個なのに対し例が 1,000 個あるため、モデルはラベルノイズをほとんど記憶しない）。最終的な検証損失とパラメータの復元誤差を数値で確認しよう。

```
w_error = d2l.numpy(d2l.reshape(model.w[...], data.w.shape) - data.w)
b_error = d2l.numpy(model.b[...] - data.b)
print(f"validation loss: {trainer.history['val_loss'][-1]:.1e}")
print('error in w:', ', '.join(f'{e:.1e}' for e in w_error))
print('error in b:', ', '.join(f'{e:.1e}' for e in b_error))
```

```
validation loss: 5.2e-05
error in w: -4.8e-04, 1.9e-04
error in b: 8.1e-06
```

訓練損失と検証損失はいずれもノイズフロアである 0.00005 まで滑らかに減少し、復元されたパラメータも $(\mathbf{w}^*, b^*)$ と約 1,000 分の 1 以内で一致している。これにより、オブジェクト指向のハーネスが独立したスクラッチ実装のループを忠実に再現しつつ、以降の章をボイラープレートから解放することが確認できた。2.5 節では次のステップに進み、手書きのテンソル、二乗損失、SGD クラスを各フレームワークの組み込みの層、損失関数、オプティマイザへと置き換える。

2.4.6 まとめ

以降の章で反復的な訓練のボイラープレートではなくモデリングに集中できるよう、d2l ライブラリはモデル（Module）、データセット（DataModule）、最適化ループ（Trainer）を分離し、それらを @add_to_class、HyperParameters、Averager、ProgressBoard で支えている。DataModule は 1 エポックで枯渇することなくパスごとに訓練例をシャッフルし直す複数エポック対応のローダを返し、Averager は各ミニバッチの評価指標に例数で重み付けすることで、端数の最終バッチによってエポック単位の平均が歪むのを防ぐ。Module は明快な戻り値の契約（training_step、validation_step、configure_optimizers）を公開しているため、LinearRegressionScratch のように生のテンソルから構築したモデルであっても、フレームワークの層から構築したモデルであっても、すべて同じ trainer.fit(model, data) の呼び出しで訓練および検証を行える。

2.4.7 演習問題

1. **訓練曲線の読み取り**。トレーナは各エポックの平均訓練損失をエポックの中央に、検証損失をその終端に描画する。
    1. 訓練の初期では、両方の例集合が同じ分布から得られているにもかかわらず、訓練曲線が検証曲線より上に位置することが多い。2 つの平均の計算方法からその理由を説明せよ。
    2. 仮に fit が検証損失と同様に、各エポック終了時点のパラメータを使って訓練損失を計算したとする。1 エポックあたりどれだけの追加コストがかかるか。また、そのとき 2 本の曲線の差は何を測ることになるか。
2. [code] **早期停止**。固定されたエポック数だけ訓練すると、停止が早すぎたり遅すぎたりすることがある。d2l.Trainer のサブクラスを作成し、その fit が、検証損失が連続して patience エポック減少しなくなった時点で self.history のみに基づいて訓練を終了するようにせよ。また、モデルやデータセットのクラスを一切変更する必要がない理由を説明せよ。実装したトレーナを、大きな max_epochs を設定した合成回帰データ上でテストせよ。
3. [code] **プランクの法則**。●同じ Module と Trainer は、線形モデルだけでなく、損失が微分可能なあらゆるモデルに当てはめることができる。プランクの法則を用いて、複数の波長における分光放射輝度の測定値から黒体の温度を求めよ。分光放射輝度は次式で与えられる。

$$B(\lambda, T) = \frac{2hc^2}{\lambda^5} \cdot \left(\exp \frac{hc}{\lambda k T} - 1 \right)^{-1}, \tag{2.22}$$

    ここで、λ は波長、T は温度、c は光速、h はプランク定数、k はボルツマン定数である。既知の T における測定値を生成した上で、d2l.Module のサブクラスとして forward が $B(\lambda, T)$ を計算し T を唯一のパラメータとして持つものを作成し、d2l.Trainer で当てはめよ。当てはめる前に、B と T が 1 のオーダーになるような単位を選べ。
4. **端数の最終ミニバッチ**。各分割に 1,000 個の例があり batch_size=32 のとき、各ローダはパスの最後に 8 例からなる端数のミニバッチを出力する。ArrayLoader に drop_last=True を指定すると、この端数ミニバッチは破棄される。
    1. 訓練ローダは毎エポック再シャッフルするのに対し、検証ローダは固定された順序で読み出す。それぞれのローダについて、端数ミニバッチの破棄によって多数のエポックにわたりどの例が除外されるかを述べ、どちらのローダであれば破棄が許容されるかを判断せよ。
    2. コンパイルされたステップは、新しい入力形状が現れるたびに 1 回コンパイルされる。端数ミニバッチを残した場合に 10 エポックにわたって発生する訓練ステップと検証

ステップのコンパイル回数を数え、そのコストを毎エポック 8 例を破棄することのコストと比較せよ。

5. [code] **オンザフライ生成**。`DataModule` のサブクラスとして、**X** や **y** を保持しないものを作成せよ。その訓練ローダは、ミニバッチが要求されるたびにデータ生成過程から毎回新たにミニバッチを抽出するものとする。
    1. 新規ミニバッチのストリームには自然な終わりがないが、`Trainer` は訓練ローダに対する 1 回のパスが終了したときにのみエポックを終了する。1 エポックが生成するミニバッチ数を決め、すべてのパスがその数で停止するようにし、`len` がその数を返すようにせよ。
    2. 検証ローダもパスごとに毎回新たな例を抽出すべきかどうかを判断し、その選択を実装せよ。また、検証損失の用途に基づいてその判断の根拠を述べよ。

2.5 線形回帰の簡潔な実装

2.3 節では、パラメータの確保、アフィン順伝播、二乗損失、リバースモード微分、ミニバッチジェネレータ、SGD の更新ステップといった線形回帰のあらゆる構成要素を生のテンソルから構築した。続く 2.4 節では、それらの構成要素を再利用可能な基底クラス（`d2l.Module`、`d2l.DataModule`、`d2l.Trainer`）へと整理し、`LinearRegressionScratch` を `trainer.fit(model, data)` で訓練することで、ベンチマーク結果が再現されることを確認した。

生のパラメータテンソルや更新規則を手書きしてテストするのは、パラメータが 3 個の線形モデルであれば簡単であった。しかし、ResNet-50（7.4 節）や GPT（11.2 節）のような現代のアーキテクチャは、数十の層にわたって数百万〜数十億個のパラメータを管理し、状態を持つ適応的オプティマイザを用いて訓練する。そのため、PyTorch (Paszke et al. 2019)、JAX (Frostig et al. 2018)（Flax と Optax を併用）、TensorFlow (Abadi et al. 2016)、MXNet (Chen et al. 2015) といった深層学習フレームワークは、ハードウェア向けに最適化された層、数値的に検証済みの損失プリミティブ、標準的なオプティマイザを標準で提供している。本節では、こうした高水準の構成要素を `d2l.Module` と `d2l.Trainer` に組み込むことで、線形回帰をいかに簡潔に記述できるかを見ていく。

2.5.1 全結合層

この簡潔な実装では、サブクラス `LinearRegression(d2l.Module)` が出力ユニット数 1 の組み込みの全結合層（fully connected layer）を 1 つラップし、重みを $\mathcal{N}(0, 0.01^2)$ から、バイアスを 0 に初期化する。

Flax NNX の層は構築時にパラメータを即座に確保するため、`nnx.Linear` は入力次元（`num_inputs`、ここでは 2 に設定）と出力次元（1）の両方を受け取る。

```
class LinearRegression(d2l.Module):
    """Linear regression built from framework components."""
    def __init__(self, num_inputs, lr, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.net = nnx.Linear(
            num_inputs, 1, kernel_init=nnx.initializers.normal(0.01),
            rngs=rngs)
```

なお、LinearRegression では forward を定義する必要すらなく、Module.forward を 2.4 節から継承してそのまま self.net(X) に処理を委譲する。

2.5.2 損失関数

$(\hat{y} - y)^2/2$ を手書きする代わりに、loss はフレームワーク組み込みの平均二乗誤差（mean squared error、MSE）プリミティブに委譲する。標準的な平均二乗誤差はスクラッチ実装の損失にあった係数 $\frac{1}{2}$ を省くため、値はスクラッチ実装の 2 倍になり、$\sigma^2 = 10^{-4}$ で頭打ちになる（$\sigma^2/2$ ではない）。

```
@d2l.add_to_class(LinearRegression)
def loss(self, y_hat, y):
    return optax.squared_error(y_hat, y).mean()
```

2.5.3 オプティマイザ

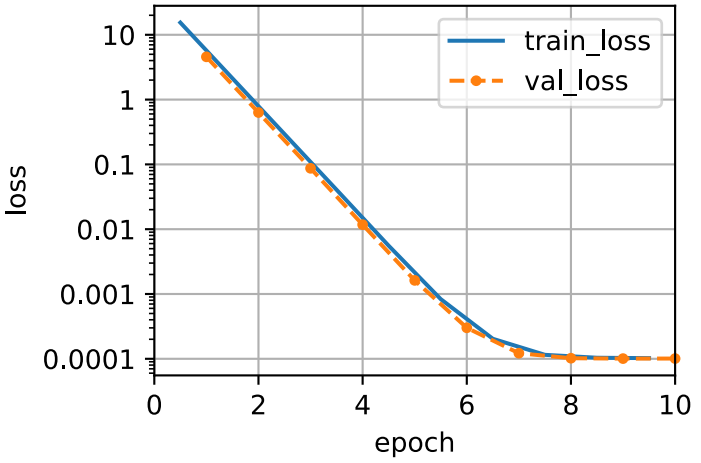
同様に、configure_optimizers はモジュールの登録済みパラメータと学習率 self.lr を接続した、フレームワーク組み込みのミニバッチ SGD オプティマイザを返す。

```
@d2l.add_to_class(LinearRegression)
def configure_optimizers(self):
    return optax.sgd(self.lr)
```

2.5.4 フレームワークの構成要素を用いた訓練

LinearRegression を定義すれば、訓練はデータモジュール、モデル（学習率は $\eta = 0.015$ とし、MSE の勾配が 2 倍になる分を相殺するためスクラッチ実装の $\eta = 0.03$ から半減させる）、および d2l.Trainer(max_epochs=10) をインスタンス化し、trainer.fit(model, data) を呼び出すだけで済む。

```
data = d2l.SyntheticRegressionData(w=d2l.tensor([2, -3.4]), b=4.2)
model = LinearRegression(2, lr=0.015)
trainer = d2l.Trainer(max_epochs=10)
trainer.board.yscale = 'log'
trainer.fit(model, data)
```




```
print(f"validation loss: {trainer.history['val_loss'][-1]:.2e}")
```

```
validation loss: 1.01e-04
```

最終的な検証損失は約 3% 以内の差で理論的なノイズフロア $\sigma^2 = 10^{-4}$ に収まり、スクラッチ実装の結果と一致する。ここで小さなヘルパーメソッド `get_w_b` を追加し、学習された重みテンソルとバイアスをフレームワークの層から取り出して $(\mathbf{w}^*, b^*)$ と比較してみよう。

```
@d2l.add_to_class(LinearRegression)
def get_w_b(self):
    return self.net.kernel[...], self.net.bias[...]
```

```
w, b = model.get_w_b()
w_error = d2l.numpy(d2l.reshape(w, data.w.shape) - data.w)
b_error = d2l.numpy(b - data.b)
print('error in w:', ', '.join(f'{e:.1e}' for e in w_error))
print('error in b:', ', '.join(f'{e:.1e}' for e in b_error))
```

```
error in w: 7.2e-05, -6.4e-05
error in b: 5.8e-05
```

今回も $\hat{\mathbf{w}}$ と $\hat{b}$ の誤差は 1,000 分の 1 程度に収まり、スクラッチ実装での実行結果と一致する。支配的な誤差要因（1,000 個の例にわたるラベルノイズと、10 エポック後に残る距離）は、誰が層を実装したかにかかわらず同一だからである。

ここまでの 3 つの節で確認してきた内容はすべて、正解（ground truth）のパラメータが既知であることに依拠していたが、これは現実のデータセットでは決して望めない贅沢である。続く 2.6 節では、ホールドアウトデータのみからモデルを評価する方法に取り組む。

2.5.5 まとめ

高水準のフレームワーク API を用いれば、線形回帰は `d2l.Module` のサブクラス内に 1 つの全結合層を置き、`loss` を組み込みの平均二乗誤差に、`configure_optimizers` を組み込みの SGD に委譲するだけで実装できる。標準的な平均二乗誤差はスクラッチ実装の目的関数にあった係数 $\frac{1}{2}$ を省く

ため、損失（$\sigma^2 = 10^{-4}$ で頭打ちになる）とその勾配がいずれも 2 倍になる。したがって、学習率を $\eta = 0.015$ へと半減させることで、スクラッチ実装と同じパラメータ更新が再現される。また、`LinearRegression` は `d2l.Module` の規約を満たしているため、データパイプラインや訓練ループを一切変更せずに `d2l.Trainer` で訓練できる。

2.5.6 演習問題

1. [code] **バイアスのない層**。バイアスを持たない層を用いて `LinearRegression` を構築し（どのフレームワークの全結合層にもバイアスを無効化する引数がある）、本節と同様に訓練せよ。
   1. 訓練の前に、2.3 節の期待二乗誤差の式を用いて（平均二乗誤差における係数 2 を考慮したうえで）、検証損失が頭打ちになる値を予測せよ。その後、予測を確かめよ。
   2. 重みが何に収束し、最終的に $\mathbf{w}^*$ からどれだけ離れた値になるかを予測せよ。特徴量の平均が 0 であるため、欠落したバイアスは標準偏差 $b^* = 4.2$ のラベルノイズのように重みに作用し、2.2.3 節の推定誤差とミニバッチ SGD のノイズ球（2.1 節）の両方に影響する。この予測を、訓練後の重みの誤差と比較せよ。
2. [code] **複数の出力**。`LinearRegression` の層の出力幅を 1 から 2 に変更せよ。`data` を構築した後、`data.y` に 2 番目のラベル列 $y_2 = \mathbf{x}^\top \mathbf{w}_2^* + b_2^* + \epsilon_2$（任意の生成パラメータと同じ大きさのノイズを用いて `data.X` から計算したもの）を追加せよ。
   1. 2 つの出力にわたって和をとった二乗損失が、出力ごとに 1 つずつの独立した 2 つの線形回帰に分解されることを示せ。そこから、一方の出力に対して学習されるパラメータが他方の出力のラベルに依存しないことを結論づけよ。
   2. フレームワークの損失関数は、例方向だけでなく、$|\mathcal{B}| \times 2$ の誤差行列（$\mathcal{B}$ はミニバッチ）のすべての要素にわたって平均をとる。単一出力モデルと比べて各出力のパラメータの勾配が何倍に縮小するか、またどの学習率を用いれば単一出力の訓練結果が再現されるかを述べよ。両方のモデルを同じ初期パラメータから同じミニバッチ上で訓練し、これを確認せよ。
3. [code] **特徴量のスケールとステップサイズ**。`data.X` の 1 列目を（`data` の構築後に）100 倍し、`data.y` はそのままにして、その特徴量の生成重みが 0.02 になるようにせよ。演習問題 1 で見たように、勾配降下法は η が $2/\mu_{\max}$ を超えると発散する（ここで $\mu_{\max}$ は損失のヘッセ行列 $\mathbf{H}$ の最大固有値である）。
   1. `LinearRegression.loss` に対する $\mathbf{H}$ を計算し、リスケーリングの前後でその固有値を推定せよ。
   2. リスケーリングの前後それぞれについて、訓練が発散しない最大の学習率を予測し、その前後の学習率で実際に訓練して両方の予測を確かめよ。
   3. 新たな上限よりわずかに小さい学習率において、2 番目の重みの誤差が 100 分の 1 に減少するまでに必要な更新回数を予測し、実測値と比較せよ。その後、リスケーリングされた特徴量を 1.2 節と同様に標準化し、`lr=0.015` が再び機能することを示せ。
4. [code] **最適化誤差と推定誤差**。●`LinearRegression` を `lr=0.015`、`max_epochs=10` として、`SyntheticRegressionData` 上で `num_train` を $5, 50, 500, 5{,}000$ のそれぞれに設定し、`batch_size=min(32, num_train)` で訓練せよ。各実行について、訓練後の重みと生成重みの距離 $\|\hat{\mathbf{w}} - \mathbf{w}^*\|_2$（生成重み $\mathbf{w}^*$ は `data.w` に格納されている）を、同じ訓練データに対する最小二乗解（パラメータ復元の項である 2.2.3 節と同様に計算したもの）の同じ距離と比較せよ。
   1. 各実行におけるパラメータ更新の回数を数え、各 `num_train` における 2 つの誤差の比を説明せよ。
   2. `num_train=5` での実行が、その最小二乗法による解の誤差の 2 倍以内に収まるまでのエポック数を求めよ。

2.6 汎化

統計学の期末試験に向けて、過去問で演習している 2 人の大学生 Ellie と Irene を想像してみよう。Ellie は過去問のすべての問題文と正確な数値の答えを一言一句丸暗記し、過去問の採点では 100% をとる。一方、Irene は背後にある概念や公式の理解に努める。時折計算ミスをするため過去問での得点は 92% にとどまるが、まったく新しい問題が出題される本番の期末試験になると、Irene が 90% をとる一方で Ellie は手も足も出ない。

機械学習モデルも同じ分岐点に直面する。最終的な目標は、手元にある有限の**訓練セット**（training set）で単に高いスコアを出すことではなく、より広い母集団から抽出される未知の例に対して正確な予測を下すことにある。この性質を**汎化**（generalization）と呼ぶ。先ほどの合成データによる線形回帰では、3 個のパラメータを 1,000 個の例に当てはめたため丸暗記は不可能であり、訓練損失は検証損失とほぼ一致して推移した。しかし、モデルクラスが個々のデータ点の周りで自在に曲がれるほど表現力をもつようになると、訓練誤差を 0 に追い込むことで、新しいデータに対する予測がかえって**悪化**しうる。本節では、訓練誤差と汎化誤差の違いを定式化し、多項式曲線フィッティングを通じて**学習不足**（underfitting）と**過学習**（overfitting）を実演したうえで、**バイアス・分散分解**（bias-variance decomposition）を解き明かし、原理に基づいたモデル選択（model selection）のための**検証セット**（validation set）と K **分割交差検証**（K-fold cross-validation）を導入する。

2.6.1 訓練誤差と汎化誤差

教師あり学習（supervised learning）では、訓練例も、実運用時に遭遇する将来のテスト例も、背後にある同一の確率分布 $P(\mathbf{x}, y)$ から独立に抽出されると仮定する。これが**独立同分布**（independent and identically distributed、i.i.d.）の仮定である。

有限の訓練データセット $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$（$n$ 個の例を P から i.i.d. に抽出したもの）が与えられたとき、予測器 f の損失 l のもとでの**訓練誤差**（training error）、すなわち**経験リスク**（empirical risk）は、$\mathcal{D}$ 上の標本平均で定義される。

$$R_{\text{emp}}[f] = \frac{1}{n} \sum_{i=1}^{n} l\left(\mathbf{x}^{(i)}, y^{(i)}, f(\mathbf{x}^{(i)})\right). \tag{2.23}$$

これに対し、**汎化誤差**（generalization error）、別名**真のリスク**（true risk）または**母集団リスク**（population risk）は、新たな例 $(\mathbf{x}, y)$ を P から抽出したときの期待損失である。

$$R[f] = \mathbb{E}_{(\mathbf{x},y)\sim P}\Big[l(\mathbf{x}, y, f(\mathbf{x}))\Big] = \int l(\mathbf{x}, y, f(\mathbf{x}))\, dP(\mathbf{x}, y). \tag{2.24}$$

求めたいのは関数 f のうち真のリスク $R[f]$ を最小化するものだが、分布 P に直接アクセスすることは決してできない。観測できるのは有限のサンプル $\mathcal{D}$ だけであるため、訓練アルゴリズムはその代わりに $R_{\text{emp}}[f]$ を最小化する。これを**経験リスク最小化**（empirical risk minimization）と呼ぶ。

なぜ $R_{\text{emp}}[\hat{f}]$ は $R[\hat{f}]$ をこれほど大きく過小評価しうるのだろうか。もし事前に関数 f を**固定**して（$\mathcal{D}$ を見る前に）選んでいれば、大数の法則（law of large numbers）により、$R_{\text{emp}}[f]$ は $R[f]$ の不偏推定値となり、その変動は $1/\sqrt{n}$ のオーダーに収まる。しかし、訓練されたモデル $\hat{f}$ は $\mathcal{D}$ と独立に固定されては**いない**。モデルクラス全体 $\mathcal{F}$ の中から、まさに $\mathcal{D}$ 上の誤差を最小化するように選び出されたものである。$\mathcal{F}$ が十分に柔軟で、$\mathcal{D}$ にたまたま含まれるノイズの癖まで拾い上げられるようになると、訓練誤差 $R_{\text{emp}}[\hat{f}]$ には真のリスク $R[\hat{f}]$ に対して下向きの楽観

的な統計的バイアスが生じる（演習問題 4）。$R[\hat{f}]$ を公正に測るには、$\hat{f}$ を、そのパラメータの当てはめに一切関与していない独立なホールドアウトの**テストセット**（test set）または**検証セット** $\mathcal{D}'$ 上で評価しなければならない。

2.6.2 学習不足と過学習

訓練誤差と検証誤差（validation error）を比較すると、正反対の 2 つの失敗モードを診断できる。

- **学習不足（高バイアス）：**モデルの自由度が低すぎてデータの背後にある真の傾向を捉えられず、訓練誤差と検証誤差の**両方**が高いまま近い値をとる。
- **過学習（高分散）：**データセットのサイズに対してモデルが柔軟すぎるため、信号だけでなく訓練例に含まれるランダムなノイズにまで適合してしまう。その結果、訓練誤差はごく小さくなる一方で検証誤差ははるかに高くなり、大きな**汎化ギャップ**（generalization gap）である $R[\hat{f}] - R_{\text{emp}}[\hat{f}]$ が開く。

2.6.2.1 多項式曲線フィッティング

線形回帰の枠組みから出ることなく、学習不足、バランスのとれた当てはめ、そして壊滅的な過学習が進行する様子を観察できる。スカラー入力 x が $[-1, 1]$ から一様に抽出され、真のラベルが滑らかな曲線にガウスノイズ（Gaussian noise）を加えた次の式で生成されるとする。

$$y = f^*(x) + \epsilon, \qquad f^*(x) = 5 + 1.2x - 3.4x^2 + 5.6x^3, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{2.25}$$

ここで $\sigma = 0.1$ とする。なお、二乗誤差 $(\hat{y} - y)^2$ における削減不能な母集団のノイズフロアは $\sigma^2 = 0.01$ である。真の関数 $f^*(x)$ でさえ 0.01 の期待二乗誤差を生じるのは、予測不可能なノイズ ϵ があるからである。

線形回帰を用いて曲線関数を当てはめるには、スカラー入力 x を単項式からなる**多項式特徴量**（polynomial features）のベクトル $(1, x, x^2, \dots, x^k)^\top \in \mathbb{R}^{k+1}$ に写像する。k 次の多項式は次のように表される。

$$\hat{y} = \sum_{j=0}^{k} w_j x^j \tag{2.26}$$

この式は x については非線形であるが、**重み** $\mathbf{w} = (w_0, \dots, w_k)^\top$ **については線形**である。したがって、任意の k 次多項式を n 個の訓練点に当てはめるには、Vandermonde 計画行列 $\mathbf{X} \in \mathbb{R}^{n \times (k+1)}$（成分は $X_{ij} = (x^{(i)})^j$）を構成して `lstsq` を呼び出せばよい（2.1.2.1 節と同様である）。それでは、$n = 20$ 個の訓練点と、テスト誤差（test error）を正確に推定するのに十分な 1,000 個のテスト点を生成し、$k = 1$（直線）、$k = 3$（3 次曲線）、$k = 19$（20 個のパラメータをもつ、20 個の点に対する 19 次多項式）の 3 つの次数を比較してみよう。

```
def target(x):
    """The cubic f* that generates the labels."""
    return 5 + 1.2 * x - 3.4 * x**2 + 5.6 * x**3

def make_data(rng, n, sigma):
    """Draw n inputs uniformly from [-1, 1] and label them by the noisy cubic."""
    x = rng.uniform(-1, 1, n)
    return x, target(x) + rng.normal(0, sigma, n)
```

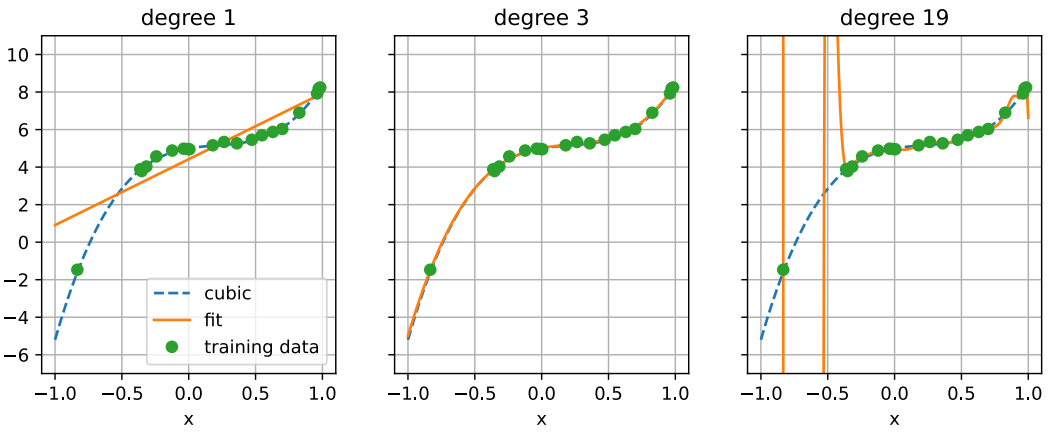
```
sigma = 0.1                              # standard deviation of the noise
rng = np.random.default_rng()            # a new random draw on every run
x_train, y_train = make_data(rng, 20, sigma)  # few, so high degrees overfit
x_test, y_test = make_data(rng, 1000, sigma)
```

以下のコードでは、features(x, d) が計画行列 $[1, x, \ldots, x^d]$ を構築し、fit_degree が最小二乗法（least squares）による厳密な重みを求め、mse が通常の平均二乗誤差（mean squared error）を計算する（係数 $\frac{1}{2}$ を省くことで、ノイズフロアがちょうど $\sigma^2 = 0.01$ になるようにしている）。

```
def features(x, d):
    """Design matrix whose columns are the powers x^0, x^1, ..., x^d."""
    return x.reshape(-1, 1) ** np.arange(d + 1)

def fit_degree(x, y, d):
    """Least-squares weights of the degree-d polynomial fitted to (x, y)."""
    return np.linalg.lstsq(features(x, d), y, rcond=None)[0]

def mse(w, x, y):
    """Mean squared error of the polynomial with weights w on (x, y)."""
    return ((features(x, len(w) - 1) @ w - y) ** 2).mean()
```

次数 1、3、19 の当てはめ結果を、ノイズを含む 20 個の訓練点および真の 3 次曲線とともにプロットしてみよう。

```
xs = np.linspace(-1, 1, 200)
_, axes = d2l.plt.subplots(1, 3, figsize=(10, 3.4), sharey=True)
for ax, d in zip(axes, (1, 3, 19)):
    w = fit_degree(x_train, y_train, d)
    d2l.plot([xs, xs, x_train], [target(xs), features(xs, d) @ w, y_train],
             xlabel='x', ylim=(-7, 11), fmts=('--', '-', 'o'),
             legend=['cubic', 'fit', 'training data'] if d == 1 else None,
             axes=ax)
    ax.set_title(f'degree {d}')
```



これら 3 つのパネルは、教師あり学習における 2 つの典型的な失敗モードのすべてを物語っている。

- **次数 $k = 1$（学習不足）：**直線は曲がることができず S 字形の 3 次曲線に追従できないため、訓練点とテスト点の双方を外してしまう。
- **次数 $k = 3$（適切な適合）：**3 次多項式は f^* とちょうど同じ柔軟性をもち、当てはめた曲線は真の曲線とほとんど見分けがつかない。
- **次数 $k = 19$（過学習）：**パラメータ数 $k + 1 = 20$ に対して相異なる訓練点が $n = 20$ 個あるため、この多項式はすべての訓練点を完全に補間できるだけの自由度をもつ（演習問題 2）。しかし、データ点の間では曲線が激しく振動し、テスト誤差は何桁も跳ね上がる。

多項式が過学習するとき、学習された重み w_j 自体には何が起きているのだろうか。3 つの次数のそれぞれについて、訓練誤差、テスト誤差、および重みの最大絶対値 $\max_j |\hat{w}_j|$ を出力してみよう。

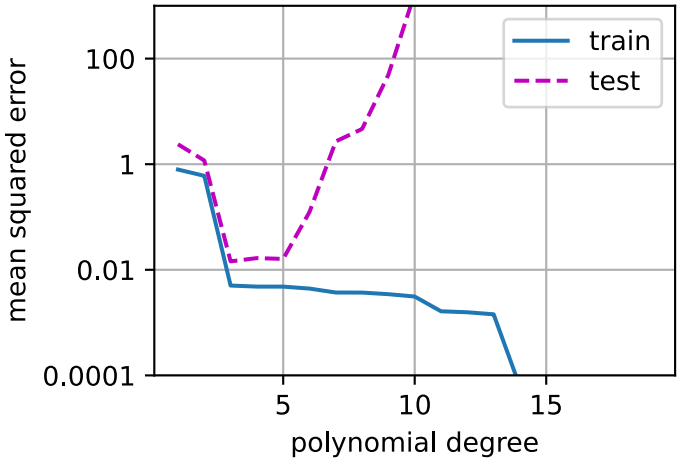
```
for d in (1, 3, 19):
    w = fit_degree(x_train, y_train, d)
    print(f'degree {d:2d}: train {mse(w, x_train, y_train):.2g}, '
          f'test {mse(w, x_test, y_test):.2g}, '
          f'largest |weight| {np.abs(w).max():.2g}')
```

```
degree  1: train 0.79, test 2.4, largest |weight| 4.4
degree  3: train 0.005, test 0.014, largest |weight| 5.4
degree 19: train 8.8e-19, test 5.3e+11, largest |weight| 1.4e+07
```

乱数の抽出ごとに正確な数値はわずかに変わるものの、その対比は劇的である。次数 1 は両方の分割で大きな誤差を抱える（テスト MSE は 2 前後と、約 200× も $\sigma^2 = 0.01$ のノイズフロアを上回る）。次数 3 は訓練誤差とテスト誤差の双方を $\sigma^2 = 0.01$ のノイズフロアまで引き下げ、最大の重みも真の最高次係数 5.6 の近くに収まる。次数 19 は訓練誤差をノイズフロアよりはるかに低く、実質的に 0 まで押し下げる一方で、テスト誤差は何桁も爆発する。その仕組みを知るには重みを見ればよい。次数 19 は巨大な正負の係数に頼っており、それらが 20 個の訓練入力では絶妙に打ち消し合う一方、それ以外のあらゆる場所では発散するのである。

次数を 1 から 19 まですべて動かすと、この推移の全体像が対数スケール上で浮かび上がる。

```
degrees = list(range(1, 20))
weights = [fit_degree(x_train, y_train, d) for d in degrees]
train_err = [mse(w, x_train, y_train) for w in weights]
test_err = [mse(w, x_test, y_test) for w in weights]
d2l.plot(degrees, [train_err, test_err], xlabel='polynomial degree',
         ylabel='mean squared error', legend=['train', 'test'],
         yscale='log', ylim=(1e-4, 1e3))
```



k が増えるにつれて、訓練誤差は単調に減少し（x^{k+1} を追加しても最小二乗法が利用できる関数の部分空間が広がるだけだからである）、σ^2 をはるかに下回る水準へ $k = 19$ までに急落する。対照的に、テスト誤差は典型的な **U 字型の曲線**を描く。学習不足が解消されるにつれて $k = 3$（またはその近傍の次数）で最小値まで急落したのち、余分な x のべき乗が 20 個の訓練点に含まれるランダムなノイズに適合していくため、何桁も上昇する。x^{17} と x^{19} は $[-1, 1]$ 上でほぼ同じ形をしているため、高次の当てはめでは一方の巨大な正の定数倍と他方の巨大な負の定数倍を打ち消し合わせることでノイズを含む 20 個の点を縫い通す。この不安定性は、2.7 節において重み減衰（weight decay）で直接解消する。

2.6.2.2 バイアスと分散

なぜモデルの複雑さ（model complexity）によって U 字型の汎化曲線が生じるのだろうか。モデル $\hat{f}_{\mathcal{D}}$（ランダムなデータセット $\mathcal{D}$ 上で訓練されたもの）のテスト点 x における期待二乗誤差を分解してみよう。ここで $y = f^*(x) + \epsilon$ であり、$\mathbb{E}[\epsilon] = 0$、$\mathrm{Var}(\epsilon) = \sigma^2$、かつ ϵ は $\mathcal{D}$ と独立であるとする。$\bar{f}(x) = \mathbb{E}_{\mathcal{D}}[\hat{f}_{\mathcal{D}}(x)]$ を、サイズ n のあらゆる訓練セットにわたる平均予測器としよう。$f^*(x)$ と $\bar{f}(x)$ を $\mathbb{E}_{\mathcal{D},\epsilon}[(\hat{f}_{\mathcal{D}}(x) - y)^2]$ の中で足して引き、交差項が消えることを用いると、有名な**バイアス・分散分解**が得られる。

$$\begin{aligned}\mathbb{E}_{\mathcal{D},\epsilon}\Big[\big(\hat{f}_{\mathcal{D}}(x) - y\big)^2\Big] &= \underbrace{\big(\bar{f}(x) - f^*(x)\big)^2}_{\mathrm{bias}^2(x)} \\ &\quad + \underbrace{\mathbb{E}_{\mathcal{D}}\Big[\big(\hat{f}_{\mathcal{D}}(x) - \bar{f}(x)\big)^2\Big]}_{\mathrm{variance}(x)} + \underbrace{\sigma^2}_{\text{noise}} .\end{aligned} \tag{2.27}$$

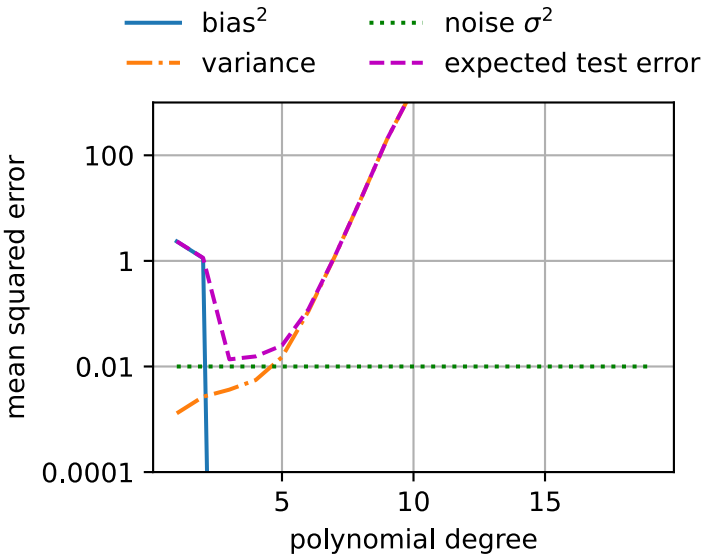
あらゆるモデルの期待テスト誤差は、次の 3 つの非負の項の和になる。

1. **バイアスの二乗** $(\bar{f}(x) - f^*(x))^2$：当てはめられたモデルの**平均**（無限に多くの訓練データセットにわたって平均したもの）が、真の関数 $f^*(x)$ からどれだけ離れているかを表す。自由度の低いモデル（3 次曲線に対する直線 $k = 1$ など）は高いバイアスをもつ。
2. **分散** $\mathbb{E}_{\mathcal{D}}[(\hat{f}_{\mathcal{D}}(x) - \bar{f}(x))^2]$：ランダムな訓練サンプル $\mathcal{D}$ が変わるたびに、当てはめられたモデルがどれほど敏感に変動するかを表す。過剰に柔軟なモデル（$k = 12$）は、20 個の点を新しく抽出するたびに大きく揺れ動き、巨大な分散を抱える。

3. **削減不能ノイズ σ^2**：ラベル y が $f^*(x)$ の周りにもつ固有の分散であり、いかなる予測器もこれを下回ることはできない。

この多項式の実験では、3 つの項すべてを直接計測できる。20 個の訓練入力を固定したまま、ラベルのノイズだけを再抽出しよう。最小二乗法の重みは $\mathbf{y}$ に線形に依存し、かつノイズの平均は 0 であるため、平均的な当てはめ $\bar{f}$ は単にノイズのないラベル $f^*(x^{(i)})$ への当てはめと一致し、ここからバイアスの二乗が厳密に求まる。また、訓練ノイズを独立に 200 回引き直して再当てはめすれば、各テスト点における分散を推定できる。

```
noise = rng.normal(0, sigma, (len(x_train), 200))   # 200 draws of the noise
bias2, var = [], []
for d in degrees:
    w_avg = fit_degree(x_train, target(x_train), d)              # average fit
    W = fit_degree(x_train, target(x_train)[:, None] + noise, d)  # fit per draw
    bias2.append(mse(w_avg, x_test, target(x_test)))
    var.append((features(x_test, d) @ W).var(axis=1).mean())
d2l.plot(degrees, [bias2, var, [sigma**2] * len(degrees),
                   np.add(bias2, var) + sigma**2],
         xlabel='polynomial degree', ylabel='mean squared error',
         fmts=('-', 'C1-.', 'g:', 'm--'), yscale='log', ylim=(1e-4, 1e3))
d2l.plt.legend(['bias$^2$', 'variance', r'noise $\sigma^2$',
                'expected test error'], ncol=2, frameon=False,
               loc='lower center', bbox_to_anchor=(0.5, 1));  # above the plot
```



これらの曲線がテスト誤差の U 字型を見事に説明していることに注目しよう。バイアスの二乗は次数 1 と 2 では大きいが、次数 3（3 次曲線を含みうる最初のモデル族）で 0 に急落する。一方、分散は次数を増やすごとに上昇し、$\sigma^2(k+1)/n$（1 例あたりのパラメータ数でスケーリングされたノイズ分散。演習問題 4 を参照）付近から始まって、やがてノイズフロアより何桁も高く跳ね上がる。学習不足とは**バイアス**が支配的な誤差であり、過学習とは**分散**が支配的な誤差である。Figure 2.11 に示す理想化されたトレードオフは、古典統計学から機械学習にわたって繰り返し現れる。複雑さが増すにつれて訓練誤差は着実に減少する一方、テスト誤差は中間のスイートスポ

ットで最小値に達する（ただし σ^2 を下回ることは決してない）。

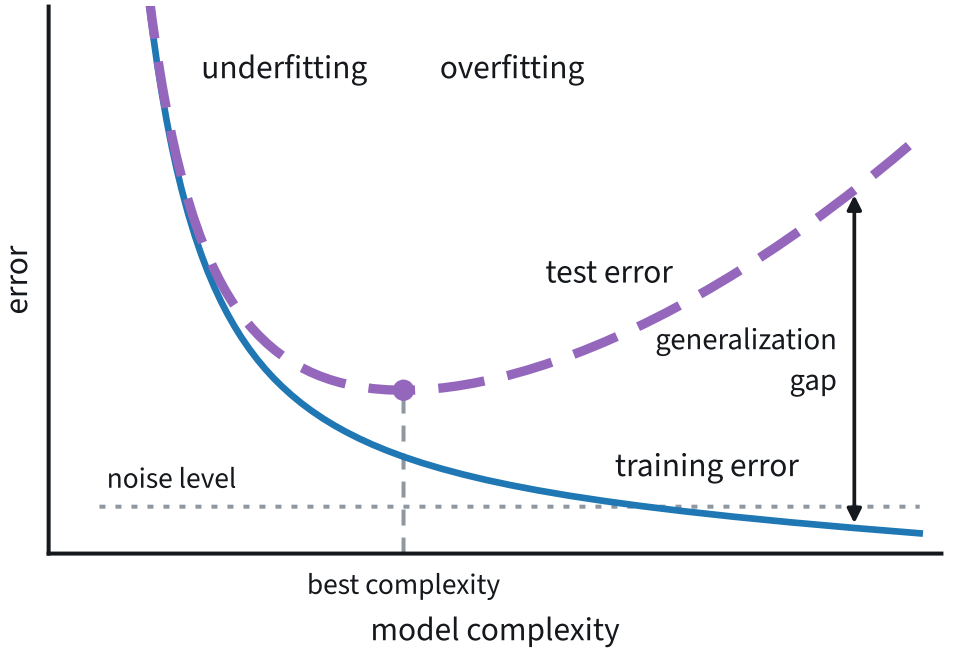


Figure 2.11: モデルの複雑さに対する訓練誤差とテスト誤差。最適な複雑さより左側ではモデルは学習不足となり両方の誤差が高い。右側では過学習し、訓練誤差はノイズレベルを下回って汎化ギャップが広がる。

2.6.2.3 モデルの複雑さ

Figure 2.11 が模式的に示すとおり、モデルの複雑さ、すなわちモデル容量（model capacity）を増やすと、バイアスは減少する一方で分散は増大する。最適な汎化性能は、バイアスの二乗と分散の和が最小となるスイートスポットにある。古典的なパラメトリックモデルでは、複雑さはパラメータ数 p と重みの大きさ $\|\mathbf{w}\|$ によって決まる。深層学習（4.5 節）では、正則化（regularization）、アーキテクチャ設計、およびデータセットの規模がこの曲線をどのように塗り替えるかを見ていく。

2.6.2.4 データセットのサイズ

モデルの複雑さを制限する以外に、分散を抑え込む最も直接的な方法は**訓練データを増やす**ことである。多項式の次数 k を固定したとき、当てはめに使う点を $n = 1{,}000$ 個に増やせば（$n = 20$ 個ではなく）、ランダムなノイズが平均化され、高次の係数でさえきれいに定まる（推定分散は $(k+1)\sigma^2/n$ に比例して減衰する）。次数 1、3、19 のモデルが、新たな訓練セットを $n = 20$ 個から $n = 1{,}000$ 個の例へと増やすにつれてどう応答するかを見てみよう。

```
for n in (20, 50, 100, 1000):
    x, y = make_data(rng, n, sigma)          # a fresh training set of size n
    errs = [mse(fit_degree(x, y, d), x_test, y_test) for d in (1, 3, 19)]
    print(f'n = {n:4d}: test error of degrees 1, 3, 19:',
          ', '.join(f'{e:.2g}' for e in errs))
```

```
n =   20: test error of degrees 1, 3, 19: 2.5, 0.011, 9.1e+14
n =   50: test error of degrees 1, 3, 19: 2, 0.011, 0.02
n =  100: test error of degrees 1, 3, 19: 1.8, 0.01, 0.33
n = 1000: test error of degrees 1, 3, 19: 1.7, 0.01, 0.01
```

この対比に注目してほしい。データセットを大きくすると、過学習していた 19 次多項式は劇的に救われる。テスト誤差は $n = 100$ までに何桁も急落し、$n = 1{,}000$ では真の 3 次曲線に並ぶ。一方、学習不足の直線はほとんど動かない（テスト誤差は 2 前後に張りついたままである）。データを増やせば**分散**は縮小するが、**バイアス**は決して解消できないからである。モデルが単独で「複雑すぎる」ということはなく、**それを制約するために利用できる信号やデータに対して**複雑すぎるにすぎない。このスケーリングの原理こそ、大規模なデジタルデータセットが当たり前になったことで深層学習が飛躍した大きな理由の 1 つである（ただし 4.5 節で見るように、深いネットワークはこの古典的な曲線に重要なひねりを加えることになる）。

2.6.3 モデル選択

2.6.3.1 検証セット

先ほどの合成データの実験では $f^*(x)$ が既知であり、1,000 個のテスト点を自在に生成できた。では現実の問題では、多項式の次数 k、学習率（learning rate）η、重み減衰の強さ λ といったハイパーパラメータをどのように選べばよいのだろうか。

訓練誤差の最小化によってハイパーパラメータを選ぶことは決してできない。訓練誤差は常に、最も複雑で丸暗記しやすいモデル（$k = 19$）を優遇してしまうからである。かといって、最終的な**テストセット**を繰り返し参照してハイパーパラメータを調整すべきでもない。そうするとテストセットのノイズが設計上の選択に漏れ出してしまう（第 3.5 章）。その代わりに、手元のデータを互いに素な 3 つの分割へと切り分ける。

1. **訓練セット**：ハイパーパラメータの各候補設定のもとで、モデルパラメータ $(\mathbf{w}, b)$ を当てはめるために用いる。
2. **検証セット**：ハイパーパラメータの候補設定どうしを比較し、最良のものを選ぶために用いる（**モデル選択**）。
3. **テストセット**：厳重に保管しておき、選択されたモデルの汎化誤差の不偏推定値を報告するために、最後の最後に **1 回だけ**評価する。

2.6.3.2 交差検証

データセットが小さい場合（たとえば $n = 20$ 件や $n = 100$ 件の臨床記録）、固定された 20% の検証セットを切り出すと、訓練に使える例も信頼できる検証に使える例も少なすぎることになる。K **分割交差検証**（Figure 2.12）は、すべての例に二役を担わせる手法である。まず、データセットをランダムに K 個の等サイズで互いに素な部分集合（**フォールド**）$\mathcal{D}_1, \dots, \mathcal{D}_K$ へと分割する（通常は $K = 5$ または $K = 10$）。ラウンド $i \in \{1, \dots, K\}$ では、$K - 1$ 個のフォールド $\mathcal{D} \setminus \mathcal{D}_i$ 上でモデルを最初から訓練し、ホールドアウトしたフォールド $\mathcal{D}_i$ 上で検証損失を評価する。この K 個のホールドアウト損失を平均したものが K 分割交差検証スコアとなり、最良のハイパーパラメータが選ばれたら、全 n 個の例を使ってモデルを再訓練する。なお、極端な $K = n$ の場合は **1 つ抜き交差検証**（leave-one-out cross-validation）と呼ばれ、各フォールドは単一のデータ点となる。線形最小二乗法では、驚くべき代数的恒等式により、**1 回**の当てはめから n 個すべての 1 つ抜き残差（residual）を計算でき、n 回の再訓練は不要になる（演習問題 6 を参照）。

実際のデータセットはソートされた状態で届くことが多く、また 1 フォールドあたり 4 点しかない場合は 1 回の分割だけではノイズが大きいため、フォールドは**ランダム**な並べ替えから作る必要がある。そこで以下の `kfold_splits` では、$K = 5$ 個のフォールドへの独立なランダム分割を

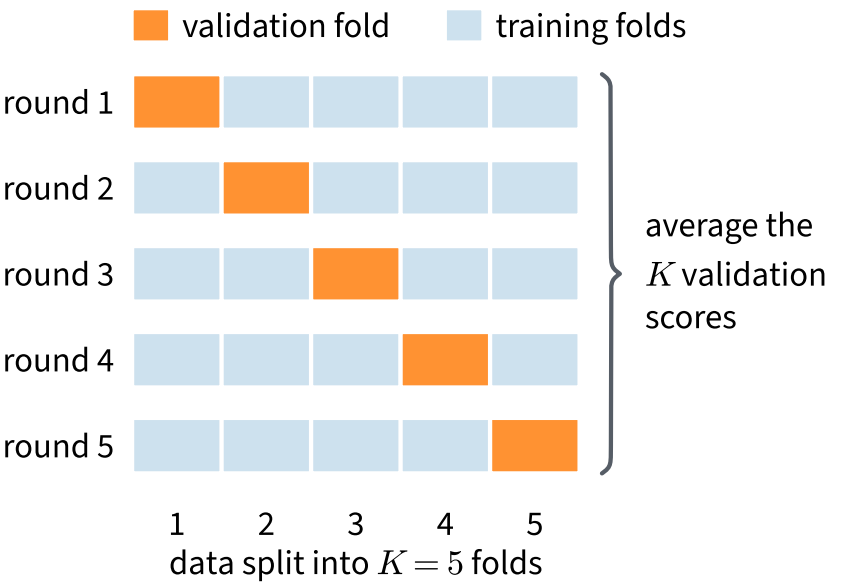


Figure 2.12: K 分割交差検証（図は $K = 5$ の場合）では、データセットを K 個のフォールドに分割し、各ラウンドで 1 つのフォールドを検証用（オレンジ色）として残し、残りの $K - 1$ 個のフォールドで訓練する。

10 回生成し（**反復**交差検証）、cv_error では多項式の当てはめの検証誤差をそれらすべてのフォールドにわたって平均する。

```
def kfold_splits(rng, n, K=5, repeats=10):
    """Random partitions of the indices 0, ..., n-1 into K folds."""
    return [np.array_split(rng.permutation(n), K) for _ in range(repeats)]

def cv_error(x, y, d, splits):
    """Validation error of the degree-d fit, averaged over all folds."""
    return np.mean([mse(fit_degree(np.delete(x, f), np.delete(y, f), d),
                        x[f], y[f]) for folds in splits for f in folds])
```

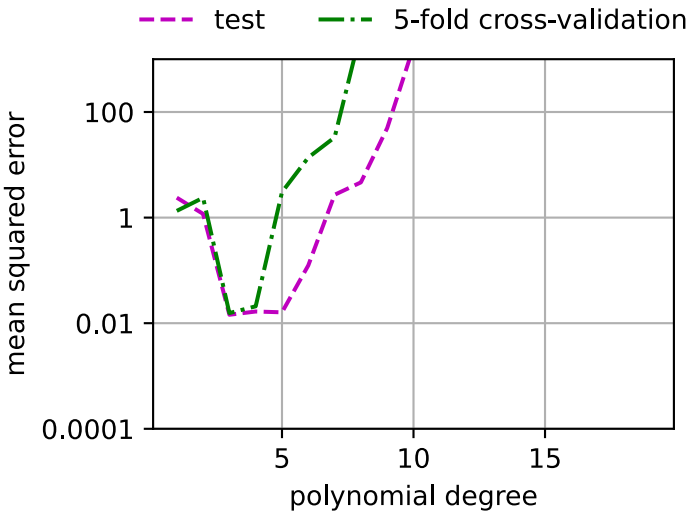
20 個の訓練例だけを使い、公平な比較のためにすべての次数を同じフォールド上で評価して多項式の次数を選択し、その選択が真のテストセット上の最適値（交差検証は一切参照しない）とどれほど一致するか確かめてみよう。

```
splits = kfold_splits(rng, len(x_train))  # the same folds for every degree
cv_err = [cv_error(x_train, y_train, d, splits) for d in degrees]
print('cross-validation picks degree:', degrees[np.argmin(cv_err)])
print('lowest test error at degree:  ', degrees[np.argmin(test_err)])
```

```
cross-validation picks degree: 3
lowest test error at degree:   3
```

全 19 個の次数にわたり、5 分割交差検証の曲線をテスト誤差の曲線と並べてプロットしてみよう。

```
d2l.plot(degrees, [test_err, cv_err], xlabel='polynomial degree',
         ylabel='mean squared error', fmts=('m--', 'g-.'), yscale='log',
         ylim=(1e-4, 1e3))
d2l.plt.legend(['test', '5-fold cross-validation'], ncol=2, frameon=False,
               loc='lower center', bbox_to_anchor=(0.5, 1));  # above the plot
```



テストセットを一度も覗き見ることなく、反復 5 分割交差検証はテスト誤差の谷底を確実に捉え、通常は次数 3 を、時折テスト誤差がほとんど区別できない近傍の次数を選択する。最小値の付近では 2 つの曲線は小さな倍率の範囲内で追従し、交差検証のほうが 20 点で平均しているため（1,000 点ではなく）、当然ながら少しノイズが多い。高次では両方の曲線が何桁も跳ね上がる。交差検証には、テスト誤差よりさらに早く上昇し始める理由がある。各フォールドは 16 点だけで訓練するため（20 点ではなく）、次数 15 で既に 16 個の訓練点すべてを補間してしまい、$[-1, 1]$ の端付近にある検証点は当てはめ範囲の外に出て高次多項式が爆発する領域に入るからである。もっとも、20 点の単一の抽出では、どちらの曲線が先に上昇することもありうる。

K はどのように選ぶべきだろうか。各フォールドはデータセットの $(K-1)/K$ の割合でしか訓練しないため、当てはめられたモデルは全 n 個の例で訓練したモデルよりわずかに劣り、K 分割交差検証にはわずかに悲観的な統計的バイアスが生じる（演習問題 5 で厳密に導出する）。$K = n$（**1 つ抜き**交差検証）まで増やせば、このバイアスはほぼ解消される。しかし一般の学習器では、候補ごとに n 回の個別の当てはめが必要になる。さらに決定木のような不安定な学習器では、1 個の例を除いただけでモデル全体が変わりうるため、ほぼ同一の n 個の訓練セットを使うことで、より粗い分割よりも推定値のノイズがかえって大きくなる場合がある。1 回の当てはめが高コストな場合に $K = 5$ や $K = 10$ が標準の既定値とされるのはこのためである (Kohavi 1995)。一方、最小二乗法は高コストでも不安定でもない。1 回の当てはめから n 個すべての 1 つ抜き残差が得られ、データに対して当てはめ結果も滑らかに変化する。

この 3 次多項式について、バイアスとノイズを計測してみよう。500 個の新たな 20 点の訓練セットのそれぞれにおいて、全 20 点に 3 次多項式を当てはめてテスト誤差を記録するとともに、その 20 点だけを使った 3 通りの誤差推定値を記録する。すなわち、1 つ抜き法（$K = 20$、各フォールド 1 点）、単一のランダム分割による 5 分割交差検証、および 10 回の分割にわたって反復した 5 分割交差検証である。

```
    loo = kfold_splits(rng, 20, K=20, repeats=1)  # K = n: one point per fold
    est = {'test error': [], 'leave-one-out': [], '5-fold, 1 split': [],
           '5-fold, 10 splits': []}
    for _ in range(500):                          # 500 fresh training sets
        x, y = make_data(rng, 20, sigma)
        est['test error'].append(mse(fit_degree(x, y, 3), x_test, y_test))
        est['leave-one-out'].append(cv_error(x, y, 3, loo))
        est['5-fold, 1 split'].append(
            cv_error(x, y, 3, kfold_splits(rng, 20, repeats=1)))
        est['5-fold, 10 splits'].append(cv_error(x, y, 3, kfold_splits(rng, 20)))
    for name, e in est.items():
        print(f'{name:18s} mean {np.mean(e):.4f}   std {np.std(e):.4f}')
```

```
test error         mean 0.0133   std 0.0051
leave-one-out      mean 0.0135   std 0.0059
5-fold, 1 split    mean 0.0152   std 0.0100
5-fold, 10 splits  mean 0.0156   std 0.0083
```

平均すると、1 つ抜き法はテスト誤差の近く、その約 10 分の 1 以内の差に収まる。5 分割の推定値はいずれも 1 つ抜き法の推定値を上回っており、19 点ではなく 16 点で訓練することによる悲観的なバイアスを反映している。3 つの交差検証推定値の中で、1 つ抜き法は 500 回の抽出にわたる標準偏差（standard deviation）も最小であり、1 つ抜き法のほうがノイズが大きいという通説はここでは成り立たない。5 分割を 10 回繰り返すとばらつきは狭まるものの、1 つ抜き法ほどには小さくならない。ばらつきの多くは、20 点をどう分割したかではなく、どの 20 点が抽出されたかという運に由来するためである。したがって最小二乗法においては、1 つ抜き法は計算が軽いだけでなく 3 つの中で最も精度が高い。一方、後の章で扱うニューラルネットワークのように 1 回の当てはめに数分から数時間かかる場合には、粗い分割が依然として現実的な選択肢となる。

ここで離散的な多項式の次数の選択がうまくいったのは、特徴量に $1, x, x^2, \dots, x^k$ という自然な階層があったからである。では、順序のない何百もの特徴量をもつデータセットが与えられ、列を 1 つずつ落とす自明な方法がない場合はどうすべきだろうか。2.7 節では、離散的な特徴選択を、重みのノルムに対する連続的なペナルティへと置き換える。

2.6.4 まとめ

教師あり学習は、有限の訓練サンプル上で経験リスク $R_{\text{emp}}[f]$ を最小化しつつ、未知のデータ上で低い母集団リスク $R[f] = \mathbb{E}[l(f(\mathbf{x}), y)]$ を達成することを目指す。最適化は訓練セットに適合するよう $\hat{f}$ を特別に選び出すため、モデルクラスがラベルのノイズを吸収できるほど柔軟である限り、$R_{\text{emp}}[\hat{f}]$ には下向きの楽観的な統計的バイアスが生じる。テスト誤差の二乗の期待値は、バイアスの二乗、推定分散、および削減不能ノイズ σ^2 へと分解される。自由度の低いモデルは高いバイアスを伴って学習不足に陥り、小さなサンプル上の過剰に柔軟なモデルは高い分散を伴って過学習し、訓練データを増やすとバイアスは減らないまま分散が $1/n$ に比例して縮小する。訓練誤差は常に丸暗記を優遇し、また最終的なテストセットへの繰り返しの問い合わせはテストセットを汚染するため、ハイパーパラメータはホールドアウトされた検証分割、または K 分割交差検証によって選ばなければならない。

2.6.5 演習問題

1. **IID が成り立たないとき。**観測どうしに従属性があるためにデータセットを IID として扱うべきでない例を 3 つ挙げよ。それぞれについて、訓練と検証へのランダム分割が汎化誤差を過大評価するか過小評価するかを述べ、その問題を回避する分割方法を説明せよ。

2. **厳密な多項式回帰**。n 個の訓練入力 $x^{(1)}, \dots, x^{(n)}$ が互いに異なるとき、次数 $n-1$ の多項式が n 個のラベルに厳密に適合することを示し、入力が互いに異なるという条件がどこで使われるかを説明せよ。2 つの入力が一致し、それらのラベルが異なるとき、最小二乗法は何を返すか。

3. [code] **次数 19 のためのより良い基底**。高次のべき乗 x^{17} と x^{19} は $[-1, 1]$ 上のどこでも差が 0.05 未満であるため、`features(x, 19)` の列ベクトルはほぼ平行になる。Chebyshev 多項式 $T_0, \dots, T_{19}$ は同じ関数空間を張るが、この区間上で -1 から 1 の間に収まる。

   1. まず予測を立てよ。Chebyshev 基底を用いて 20 個の訓練点に次数 19 を当てはめたとき、訓練誤差、テスト誤差、最大の重みのうちどれが、どの程度変化するか。
   2. 複数回実行してその予測を確かめよ。訓練入力とテスト入力の両方で `features(x, 19)` を `np.polynomial.chebyshev.chebvander(x, 19)` に置き換え、2 つの計画行列の条件数（`np.linalg.cond`）、訓練誤差とテスト誤差、および最大の重みを比較せよ。
   3. 厳密演算では 2 つの当てはめが同一の多項式になる理由と、float64 ではどの数値がなぜ異なるのかを説明せよ。より良い基底を使えば 20 点に対する次数 19 は救われるか。この結果から、複雑さの尺度としての重みの大きさについて何が言えるか。

4. **訓練誤差の楽観性**。●列フルランクの固定された $n \times p$ 計画行列 $\mathbf{X}$ と、ラベル $\mathbf{y} = \mathbf{X}\mathbf{w}^* + \epsilon$（ノイズ項は互いに独立で平均 0、分散 σ^2）に対する最小二乗法を考える。当てはめによる予測値は $\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$（ただし $\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$）である。

   1. 期待訓練誤差が $\mathbb{E}[\|\mathbf{y} - \hat{\mathbf{y}}\|^2]/n = \sigma^2(n-p)/n$ となることを示せ。ヒント：$\mathbf{y} - \hat{\mathbf{y}} = (\mathbf{I} - \mathbf{H})\epsilon$ であり、$\mathbf{I} - \mathbf{H}$ のトレースは $n - p$ である。
   2. 同じ入力に対して新たなラベル $\mathbf{y}' = \mathbf{X}\mathbf{w}^* + \epsilon'$ を抽出する。これに対する $\hat{\mathbf{y}}$ の期待誤差が $\sigma^2(n+p)/n$ となり、したがって訓練誤差はそれを $2\sigma^2 p/n$ だけ過小評価することを示せ。また、$\sigma^2 p/n$ が n 個の入力にわたって平均した予測値の分散であることを示せ。
   3. 2.6.2.1 節の 3 次多項式の当てはめ（$n = 20$、$p = 4$、$\sigma = 0.1$）について両方の式を計算し、実験を複数回行ったときの次数 3 の訓練誤差およびテスト誤差と比較せよ。その違いを説明せよ。すなわち、1 回の実行における訓練誤差はその期待値の周りにどの程度ばらつくか、また訓練入力で成り立つ (b) の公式がテスト入力にはなぜ厳密に当てはまらないのか。

5. **K 分割交差検証**。

   1. m 個の例で訓練されたモデルの期待汎化誤差が $R(m) = R_\infty + c/m$（ただし $c > 0$）であるとする。各フォールドのモデルは $(K-1)n/K$ 個の例で訓練される。K 分割交差検証の推定値がもつ統計的バイアスを、$R(n)$（全 n 個の例で訓練されたモデルの誤差）を基準として計算せよ。このバイアスを $R(n) - R_\infty$ に対する比として表し、$K = 2$、5、10、および n についてその値を求めよ。
   2. 交差検証でハイパーパラメータを選択したのち、最終モデルは全 n 個の例で再訓練される。最適な値が訓練セットのサイズに依存するハイパーパラメータがこの手順を難しくする理由を説明し、その具体例を 1 つ挙げよ。

6. **1 つ抜き法の近道**。●一般の学習器では、1 つ抜き交差検証に n 回のモデルの当てはめが必要となる。計画行列 $\mathbf{X}$（各行は $\mathbf{x}^{(i)\top}$）を用いた最小二乗法を考える。$\mathbf{A} = \mathbf{X}^\top\mathbf{X}$ とおき、$\mathbf{X}$ から任意の 1 行を取り除いても正則性が保たれると仮定する。$\hat{y}^{(i)}$ を、$\mathbf{x}^{(i)}$ における全 n 個の例への当てはめの予測値とし、$\hat{y}^{(-i)}$ を、$\mathbf{x}^{(i)}$ における例 i を除いた当てはめの予測値とし、$h_i = \mathbf{x}^{(i)\top}\mathbf{A}^{-1}\mathbf{x}^{(i)}$ とおく。ここで、Sherman–Morrison の恒等式（33.2 節）は次のとおりである。

$$(\mathbf{A} - \mathbf{u}\mathbf{u}^\top)^{-1} = \mathbf{A}^{-1} + \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{u}^\top\mathbf{A}^{-1}}{1 - \mathbf{u}^\top\mathbf{A}^{-1}\mathbf{u}}$$

この式で $\mathbf{u} = \mathbf{x}^{(i)}$ とおくことにより、次式が成り立つことを示せ。

$$y^{(i)} - \hat{y}^{(-i)} = \frac{y^{(i)} - \hat{y}^{(i)}}{1 - h_i}. \tag{2.28}$$

さらに、最小二乗法における 1 つ抜き交差検証のコストが、1 回の当てはめと比べてどうなるかを述べよ。

2.7 重み減衰

2.6 節では、多項式特徴量の離散的な個数 $k + 1$ を変えるという、最も大雑把な方法でモデルの複雑さを制御した。しかし、離散的な特徴量選択はすぐに壁に突き当たる。数百から数千の特徴量を持つ高次元の問題では、特徴量の部分集合は 2^d 通りも存在する。さらに悪いことに、すでに見たとおり、多項式が過学習すると、ほぼ共線関係にある特徴量どうしを打ち消し合うために重み w_j が数百万以上にまで爆発してしまう。

この本質的な病理を 2 つの特徴量からなる最も単純な形で確認するため、隣り合わせに置かれた 2 本のノイズを含む温度計 x_1 と x_2 から室温を予測する状況を考えよう。それぞれの温度計は、真の室温にわずかで独立なノイズの揺らぎが加わった値を示すものとする。ここで、重みの和が 2 になる 2 つの線形モデルを考える。

- **モデル A** は $\mathbf{w} = (1, 1)^\top$ とおき、2 つのセンサの平均をとることでセンサのノイズ分散を半分に抑える。
- **モデル B** は $\mathbf{w} = (501, -499)^\top$ とおく。2 本の温度計が一致しているときは $501x_1 - 499x_2 = 2x_1$ となって同じ平均を与えるが、2 つのセンサ間にわずかなずれ $x_1 - x_2$ があるだけで、それが 500 倍に増幅されてしまう。

小さな訓練サンプルにおいて、たまたま $x_1 - x_2$ がラベルのノイズとわずかに相関していた場合、正則化なしの最小二乗法は訓練誤差をほんの少し削るために $(501, -499)^\top$ を選んでしまい、新しい測定値に対して大きく破綻する。どうすれば d 個の特徴量をすべてモデルに残したまま、定まりの悪い方向に沿って重みが爆発するのを防げるだろうか。

統計学では**リッジ回帰**（ridge regression）または **Tikhonov 正則化**（Tikhonov regularization）として知られ (Hoerl and Kennard 1970; Tikhonov and Arsenin 1977)、深層学習では**重み減衰**（weight decay）と呼ばれる古典的な解決策は、二乗ユークリッドノルム $\|\mathbf{w}\|_2^2$ に対する滑らかなペナルティを損失関数に直接加えることである。本節では、相補的な 4 つの視点（ペナルティ付き最適化、SGD におけるステップごとの乗法的減衰、ベイズ的なガウス事前分布、および $\mathbf{X}^\top\mathbf{X}$ の固有ベクトルに沿ったスペクトル縮小）から重み減衰を導出し、4 つのフレームワークすべてにおいてスクラッチ実装と簡潔な実装の両方を示す。

2.7.1 ノルムペナルティ

2.7.1.1 ペナルティ付き損失

特徴量の個数を制限する代わりに、重みベクトルの二乗 ℓ_2 ノルム $\|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} w_j^2$ にペナルティを課すことで、線形モデルの実効的な容量を制御する。平均二乗誤差損失 $L(\mathbf{w}, b)$ に、非負の

正則化定数（regularization constant）$\lambda \geq 0$ を掛けたこのペナルティを加えると、次の正則化目的関数が得られる。

$$L(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2. \tag{2.29}$$

ここでも係数 $\frac{1}{2}$ は、導関数に現れる 2 を $\|\mathbf{w}\|_2^2$ の微分時に打ち消すために付けてある。Equation 2.29 の 3 つの基本的な性質に注目しよう。

1. **λ によるトレードオフ：**$\lambda = 0$ とおけば正則化なしの最小二乗法に戻る。一方、λ を大きくすると大きな重みにより厳しいペナルティが課され、$\mathbf{w}$ は $\mathbf{0}$ に引き寄せられる（そして $\lambda \to \infty$ の極限では $\mathbf{w} \to \mathbf{0}$ となる）。
2. **バイアス b にペナルティを課さない理由：**バイアス b は、出力の平均レベルを目的変数 y の平均に合わせるために平行移動させるだけの役割を持つ。もし b を 0 に向かって縮小させると、たとえば y が摂氏ではなくケルビンで測られていた場合、入力の摂動に対するモデルの感度を少しも下げないまま、すべての予測を 0 の方向へ偏らせてしまう。
3. **特徴量のスケーリングの重要性：**$\sum_j w_j^2$ は座標ごとに生の二乗係数を足し合わせるため、ある 1 つの特徴量をメートルではなくミリメートルで表し直すと、その重み w_j は $1{,}000$ で割られることになり、ペナルティへの寄与は 10^6 分の 1 に縮小する。実際には、等方的な $\|\mathbf{w}\|_2^2$ ペナルティを適用する前に、各特徴量を同程度のスケール（平均 0、単位分散）に標準化する。

なぜ二乗 ℓ_2 ノルムにペナルティを課すのだろうか。2 本の温度計の例では、$(1, 1)^\top$ のときは $\|\mathbf{w}\|_2^2 = 2$ であるのに対し、$(501, -499)^\top$ のときは $\|\mathbf{w}\|_2^2 = 500{,}002$ となる。そのため、ごく小さな $\lambda > 0$ であっても、2 つのセンサの差をとるより平均をとる方が強く選好される。

2.7.1.2 勾配ステップにおける重み減衰

ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ によって、ミニバッチ SGD の更新則はどう変わるだろうか。$\nabla_{\mathbf{w}} \left(\frac{\lambda}{2}\|\mathbf{w}\|_2^2\right) = \lambda\mathbf{w}$ であるから、ペナルティ付きの勾配ステップは次のようになる。

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\left(\frac{\partial L}{\partial \mathbf{w}} + \lambda\mathbf{w}\right) = (1 - \eta\lambda)\mathbf{w} - \eta\frac{\partial L}{\partial \mathbf{w}}. \tag{2.30}$$

右辺の第 1 項に注目しよう。**通常のデータに基づく勾配ステップ $-\eta\,\partial L/\partial \mathbf{w}$ を適用する前に、重みベクトルに $(1 - \eta\lambda) < 1$ を掛けて、更新のたびに $\mathbf{0}$ に向かってわずかに縮小（「減衰」）させている。**この乗法的な縮小因子 $(1 - \eta\lambda)$ こそ、深層学習の実践者が ℓ_2 正則化を**重み減衰**と呼ぶ理由である（通常の SGD では、損失に $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ を加えることと重みを $(1 - \eta\lambda)$ 倍に減衰させることは代数的に同一であるが、第 9 章で扱う Adam のような適応的オプティマイザでは、乗法的な重み減衰を分離して **AdamW** とすることが重要な違いを生む）。

2.7.1.3 ℓ_1 ペナルティとスパース性

二乗 ℓ_2 ノルム $\|\mathbf{w}\|_2^2 = \sum_j w_j^2$ を ℓ_1 ノルム $\|\mathbf{w}\|_1 = \sum_j |w_j|$ に置き換えたらどうなるだろうか。これは統計学で **Lasso** と呼ばれる手法である (Tibshirani 1996)。両者の制約集合の幾何学的形状を比較すると（Figure 2.13）、際立った違いが見えてくる。データ損失 $L(\mathbf{w})$ を制約 $\|\mathbf{w}\|_2^2 \leq r^2$ のもとで最小化すると、滑らかな円板（$\mathbb{R}^d$ においては球体）の内部を探索することになり、楕円状の損失の等高線はほぼ常に、**すべての**座標 w_j が小さいながらも非ゼロである点で境界に接する。これに対し、$L(\mathbf{w})$ を制約 $\|\mathbf{w}\|_1 \leq r$ のもとで最小化すると、頂点が座標軸上に直接乗った鋭い角を持つひし形（正軸体）の内部を探索することになる。楕円状の損失の等高線は、しばし

ばこうした鋭い角の 1 つに最初にぶつかるため、多くの座標 w_j が**厳密に 0** となり、自動的な特徴量選択、すなわち**スパース性**（sparsity）が実現される。

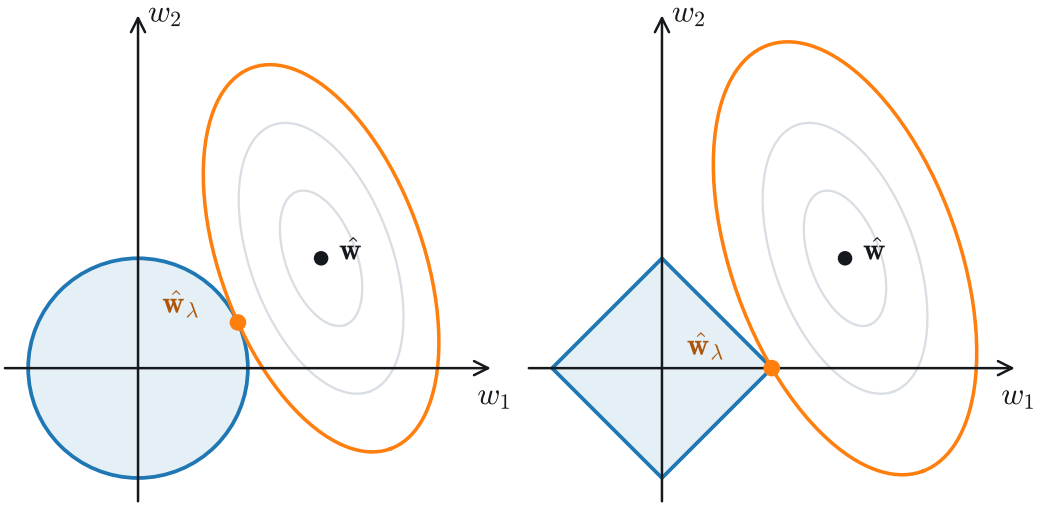


Figure 2.13: ℓ_2 リッジ回帰（左）と ℓ_1 Lasso 回帰（右）の制約の幾何学的形状。データ損失 $L(\mathbf{w})$ の楕円状の等高線は、正則化なしの最小点 $\hat{\mathbf{w}}$ から広がり、通常は軸から離れた位置で滑らかな ℓ_2 の円に接する（両方の重みが非ゼロに保たれる）のに対し、ℓ_1 のひし形では軸上にある鋭い角にぶつかることが多い（$w_2 = 0$ となる）。

2.7.1.4 重みに対するガウス事前分布

2.1.3 節においてガウス観測ノイズのもとでの最尤法から二乗データ損失が導かれたのと同様に、ベイズ統計学からは重み減衰に対しても同じくらい明快な確率的解釈が得られる。データを一切観測する前に、重みに対して平均 0 の等方的なガウス**事前分布**（prior distribution）を独立に仮定し、$w_j \sim \mathcal{N}(0, \tau^2)$、すなわち $p(\mathbf{w}) \propto \exp(-\|\mathbf{w}\|_2^2/(2\tau^2))$ とおく一方、ラベルは $y \mid \mathbf{x}, \mathbf{w}, b \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x} + b, \sigma^2)$ に従うものとする。

ベイズの定理により、**事後密度**（posterior density）$p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) \propto p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, b)\, p(\mathbf{w})$ を最大化することは、その負の対数を最小化することと同値である。これが**最大事後確率推定**（maximum a posteriori estimation、MAP 推定）であり、最小化する式は次のようになる。

$$\frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b \right)^2 + \frac{1}{2\tau^2} \|\mathbf{w}\|_2^2. \tag{2.31}$$

全体を n/σ^2 で割ると、重み減衰の目的関数 Equation 2.29 が $\lambda = \sigma^2/(n\tau^2)$ として得られる。この比がいかに理にかなっているかに注目しよう。重みが小さいという事前信念が強いほど（τ^2 が小さいほど）、あるいは観測ノイズが大きいほど（σ^2 が大きいほど）、より強い正則化 λ が求められる。一方で、サンプルサイズ $n \to \infty$ の極限では、豊富なデータ自体が十分に語るようになるため、ペナルティは $\lambda = \sigma^2/(n\tau^2) \to 0$ となる。

2.7.2 重み減衰のスクラッチ実装

訓練例の数とほぼ同じ数の重みを持つモデルを重み減衰がいかに救うかを確認するため、次元 $d = 200$ が訓練サンプルサイズ $n_{\text{train}} = 300$ に迫る高次元の合成回帰問題を構成しよう。具体的には、入力 $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{200})$ をサンプリングし、次の式からラベルを生成する。

$$y = 0.05 + \sum_{j=1}^{200} 0.1\, x_j + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1), \tag{2.32}$$

ここでは 300 個の訓練例（ミニバッチサイズ 20）と 1,000 個の検証例を、SyntheticRegressionData（2.4 節）を用いて用意する。真の重み $\mathbf{w}^* = (0.1, \ldots, 0.1)^\top$ の各座標のスケールは $\tau = 0.1$ であり、ノイズはかなり大きい。その標準偏差 $\sigma = 1$ は、ノイズのない信号の標準偏差 $\|\mathbf{w}^*\| = \sqrt{2} \approx 1.4$ に匹敵する。原理的には 300 個の例があれば 200 個すべての重みを特定できるものの、その線形結合の多くは弱くしか定まらないことがまもなくわかる。

```
tau, sigma, n = 0.1, 1.0, 300   # size of the weights, noise, examples
data = d2l.SyntheticRegressionData(w=d2l.ones(200) * tau, b=0.05,
                                   noise=sigma, num_train=n, num_val=1000,
                                   batch_size=20)
```

訓練を始める前に、検証損失に対する 2 つの基準値を設定しておこう。独立な標準正規入力 $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ とノイズ $\epsilon \sim \mathcal{N}(0, \sigma^2)$ に対し、$(\mathbf{w}, b^*)$ における期待損失は $\frac{1}{2}\|\mathbf{w} - \mathbf{w}^*\|_2^2 + \frac{1}{2}\sigma^2$ に等しい（2.3 節）。したがって、200 個の入力をすべて無視して常に b^* と予測する自明な定数モデルの期待損失は $\frac{1}{2}(\|\mathbf{w}^*\|^2 + \sigma^2) = \frac{1}{2}(2 + 1) = 1.5$ となる一方、オラクルのパラメータ $(\mathbf{w}^*, b^*)$ は $\sigma^2/2 = 0.5$ を達成する。学習されたモデルが 200 個の重みを持つ意味があるといえるには、検証損失が 1.5 を十分に下回らなければならない。

2.7.2.1 ペナルティ付きモデル

l2_penalty(w) を定義し、$\frac{1}{2}\|\mathbf{w}\|_2^2$ を返すようにする。

```
def l2_penalty(w):
    return d2l.reduce_sum(w**2) / 2
```

LinearRegressionScratch（2.4 節で実装した、2.3 節のスクラッチモデルのオブジェクト指向版）を継承してサブクラスを定義する。変更が必要なのは training_step だけであり、微分と最適化のためにペナルティ付き目的関数 Equation 2.29 を返すようにする。重要な点として、validation_step は引き続きペナルティなしの二乗損失を報告する。これにより、異なる λ の値どうしでも、検証損失を予測精度の尺度として直接比較できる。

```
class WeightDecayScratch(d2l.LinearRegressionScratch):
    def __init__(self, num_inputs, lambd, lr, sigma=0.01):
        super().__init__(num_inputs, lr, sigma)
        self.save_hyperparameters()

    def training_step(self, batch):
        loss = super().training_step(batch)
        return loss + self.lambd * l2_penalty(self.w)
```

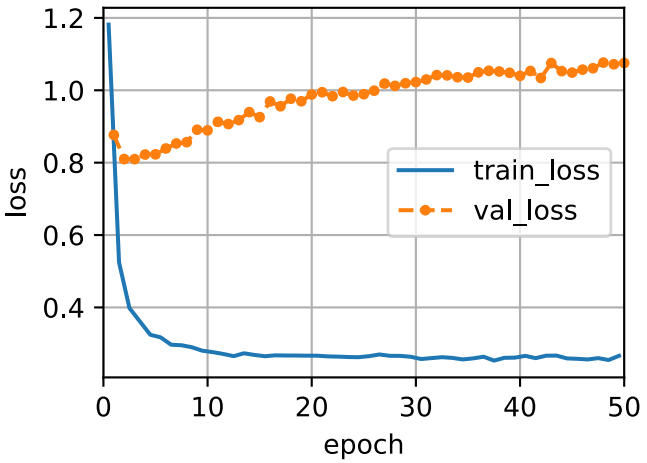
2.7.2.2 ペナルティあり・なしでの訓練

`train_scratch(lambd, data)` を作成し、`WeightDecayScratch` を 50 エポック、学習率 $\eta = 0.05$（750 回の SGD ステップ）で訓練しながら訓練曲線と検証曲線をプロットし、当てはめたモデル、エポックごとの検証損失、および学習された重みのノルム $\|\mathbf{w}\|_2$ を返すようにしよう（比較のため、真の重みベクトル $\mathbf{w}^* = (0.1, \ldots, 0.1)^\top \in \mathbb{R}^{200}$ のノルムは $\|\mathbf{w}^*\|_2 = \sqrt{200 \times 0.1^2} \approx 1.4$ である）。

```
def train_scratch(lambd, data):
    trainer = d2l.Trainer(max_epochs=50)
    model = WeightDecayScratch(num_inputs=200, lambd=lambd, lr=0.05)
    trainer.fit(model, data)
    norm = float(jnp.linalg.norm(model.w[...]))
    return model, [float(v) for v in trainer.history['val_loss']], norm
```

まず `lambd = 0` とし、正則化を完全に無効にして訓練する。

```
model_0, val_losses, norm = train_scratch(0, data)
```




```
print(f'validation loss {val_losses[-1]:.3f} (lowest {min(val_losses):.3f}, '
      f'epoch {val_losses.index(min(val_losses)) + 1}), norm of w {norm:.2f}')
```

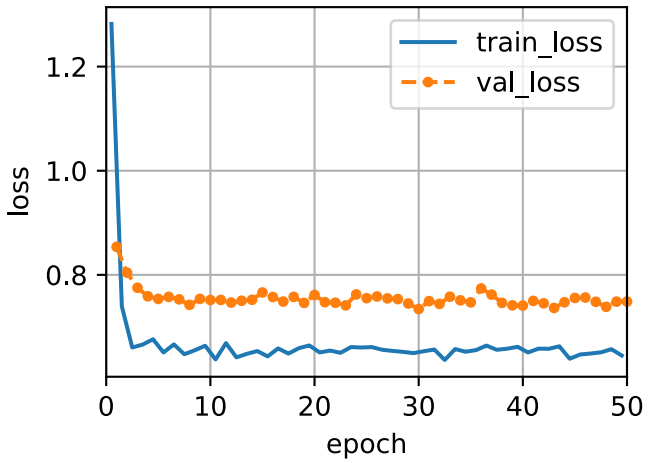
```
validation loss 1.075 (lowest 0.810, epoch 3), norm of w 1.96
```

訓練損失は約 0.2〜0.3 まで急落する。これは真のパラメータが達成する削減不能なノイズ下限 0.5 を大きく下回っており、モデルが 300 個の訓練ラベルに含まれるノイズを丸暗記していることを裏づけている。一方、検証損失はわずか 2〜3 エポック後に最小値 0.8〜1.0 に達したのち着実に上昇し、多くの場合 1.1〜1.9 の間で終了する。50 エポック後には、200 個の重みを持つこのモデルは序盤の改善の大半を失い、自明な定数予測 b^* の損失 1.5 と同程度（ときにはそれ以下）の性能にとどまる。まさに教科書どおりの過学習である。なぜ検証損失は、わずか 30〜45 回の勾配ステップというこれほど早い段階で底を打つのだろうか。その理由は、勾配降下法が重み空間のどの方向を先に学習するかを調べれば明らかになる。

次に、重み減衰を有効にする。λ を当て推量で決めるのではなく、ガウス事前分布の式を参照しよ

う。$\sigma = 1$、$n = 300$、真の重みのスケール $\tau = 0.1$ に対して、MAP 推定は $\lambda = \sigma^2/(n\tau^2) = \frac{1}{3}$ を与える。

```
lambd = sigma**2 / (n * tau**2)   # the value the prior suggests, 1/3
model_lambd, val_losses, norm = train_scratch(lambd, data)
```




```
print(f'validation loss {val_losses[-1]:.3f}, norm of w {norm:.2f}')
```

```
validation loss 0.749, norm of w 1.20
```

今度は訓練曲線がペナルティ付き目的関数を追跡し、0.5 を安定して上回り続ける。ペナルティによって、モデルが訓練ラベルのノイズを追いかけるのが抑えられているためである。検証損失は約 0.75〜1.05 に落ち着いて横ばいとなり、定数予測器の基準値 1.5 を大きく下回る。同時に重みのノルムは約 2 から約 1.2 へと縮小し、実際には真のノルム $\|\mathbf{w}^*\| \approx 1.4$ よりも**短く**なる。正則化された重みは真の長さより短く縮小されているにもかかわらず、正則化なしの重みより劇的によく汎化する。

ここで得られた効果を率直に評価しておこう。50 エポック最後まで訓練したペナルティなしモデルと比べれば、重み減衰は検証損失を 4 分の 1 から半分も削減している。しかし、同じペナルティなしモデルを最良のエポックで止めた場合と比べると、改善幅は小さく、どちらも約 0.8〜1.0 に達する。実用上の違いは安定性にある。重み減衰は目的関数そのものを変えるため、どれほど長く訓練しても解の良さが保たれるのに対し、ペナルティなしの実行は訓練損失だけでは見当もつかない絶妙なタイミングで止めなければならない。正則化と早期停止（early stopping）がいずれも**なぜ**このように振る舞うのかを理解するため、重みベクトルの内部に踏み込み、ペナルティが実際にどの成分を取り除いたのかを調べてみよう。

2.7.3 固有ベクトルに沿った縮小

$\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ を加えることで、単にすべての重みを一様に縮小するのではなく、なぜこれほど劇的に検証誤差が改善するのだろうか。特徴量の共分散行列の固有基底においてリッジ目的関数の厳密な最小点を調べることで、この問いに閉じた形で答えることができる。

2.7.3.1 リッジ推定量

特徴量とラベルが中心化されていて $b = 0$ であるとし、リッジ目的関数 Equation 2.29 を行列形式で書き下す。

$$L_\lambda(\mathbf{w}) = \frac{1}{2n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2. \tag{2.33}$$

勾配を $\nabla_{\mathbf{w}} L_\lambda(\mathbf{w}) = \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w} = \mathbf{0}$ とおくと、次の正則化正規方程式が得られる。

$$\begin{aligned}\left(\frac{1}{n}\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}\right)\hat{\mathbf{w}}_\lambda &= \frac{1}{n}\mathbf{X}^\top\mathbf{y}, \\ \hat{\mathbf{w}}_\lambda &= \left(\mathbf{X}^\top\mathbf{X} + n\lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y}.\end{aligned} \tag{2.34}$$

ここから直ちに導かれる代数的帰結に注目しよう。たとえ $\mathbf{X}^\top\mathbf{X}$ が特異であっても（訓練例が特徴量より少ないときは常にそうなる。たとえば $n = 20$、$d = 200$ のとき、$\mathbf{X}^\top\mathbf{X}$ は 180 個のゼロ固有値を持つ。演習問題 6 を参照）、$n\lambda\mathbf{I}$ を任意の $\lambda > 0$ に対して加えれば、すべての固有値が $n\lambda > 0$ だけ上方にシフトし、$\mathbf{X}^\top\mathbf{X} + n\lambda\mathbf{I}$ は狭義正定値となって必ず正則（可逆）になる。

2.7.3.2 縮小因子

Equation 2.34 が特徴空間の各方向に沿ってどのように作用するかを調べるため、$\mathbf{S} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$ を特徴量の $d \times d$ 標本二次モーメント（共分散）行列とする。$\mathbf{S}$ は対称かつ半正定値であるため（1.3 節）、正規直交固有値分解 $\mathbf{S} = \sum_{j=1}^{d} \mu_j \mathbf{v}_j \mathbf{v}_j^\top$ を持つ。ここで $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_d \geq 0$ は固有値、$\mathbf{v}_1, \ldots, \mathbf{v}_d$ は正規直交固有ベクトルである。このとき $\mu_j = \frac{1}{n}\|\mathbf{X}\mathbf{v}_j\|_2^2$ は、訓練入力の経験分散（広がり）を方向 $\mathbf{v}_j$ に沿って測った量になっている。

この固有基底では、$\mathbf{S}$ は $\mathbf{v}_j$ に μ_j を掛ける作用素として働き、$(\mathbf{S} + \lambda\mathbf{I})^{-1}$ は $\mathbf{v}_j$ を $\mu_j + \lambda$ で割る作用素として働く。$\mu_j > 0$ である限り、リッジ推定量の成分 $\mathbf{v}_j^\top\hat{\mathbf{w}}_\lambda$ を正則化なしの最小二乗推定量の成分 $\mathbf{v}_j^\top\hat{\mathbf{w}}_0$ と比較すると、次の関係が得られる。

$$\begin{aligned}\mathbf{v}_j^\top\hat{\mathbf{w}}_\lambda &= \frac{\mu_j}{\mu_j + \lambda}\left(\mathbf{v}_j^\top\hat{\mathbf{w}}_0\right), \\ \mathbf{v}_j^\top\hat{\mathbf{w}}_\lambda &= 0 \quad \text{whenever } \mu_j = 0.\end{aligned}$$

各固有方向 $\mathbf{v}_j$ には、それぞれ固有の**縮小因子**（shrinkage factor）が掛けられる（Figure 2.14）。

$$\frac{\mu_j}{\mu_j + \lambda} \in [0, 1). \tag{2.35}$$

ここにリッジ回帰の本質がある。**重み減衰とは一様な縮小ではなく、分散に適応した方向別のフィルタリングなのである。**

- データが豊かに変動する**よく定まる方向**（$\mu_j \gg \lambda$）では、傾きが観測値によってしっかりと特定されており、縮小因子 $\mu_j/(\mu_j + \lambda) \approx 1$ によって最小二乗推定量はほぼそのまま保たれる。

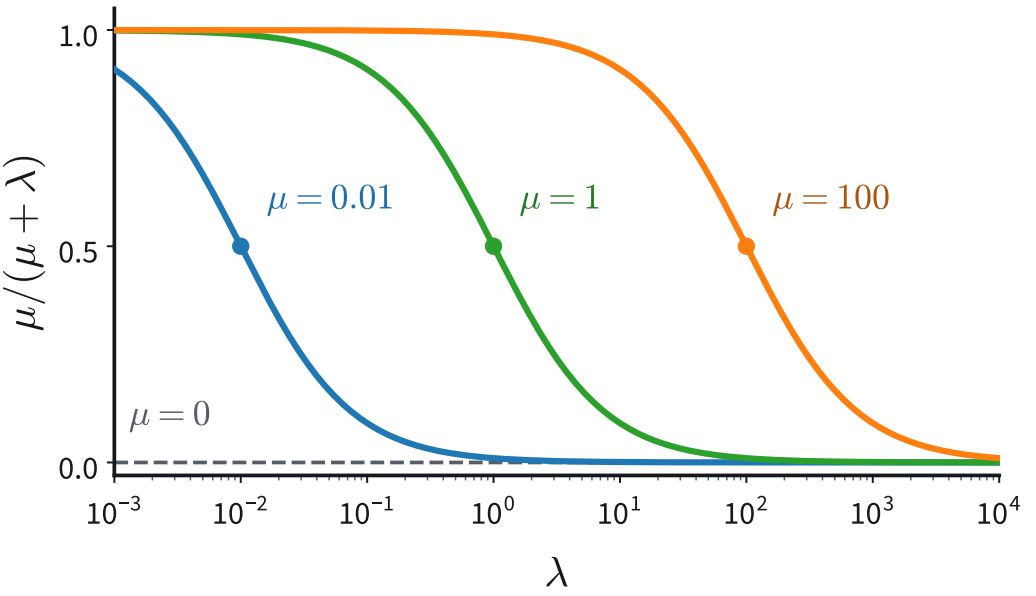


Figure 2.14: リッジの縮小因子 $\mu/(\mu+\lambda)$（固有値 μ を持つ固有ベクトルに沿った値）を正則化定数 λ（対数スケール）に対してプロットしたもの。データによってよく定まる方向は $\lambda \ll \mu$ の間ほぼそのまま保たれ、$\lambda \gg \mu$ になると 0 に向かって強く縮小される。また、この因子が $\frac{1}{2}$ となるのは $\lambda = \mu$ のときである（丸印）。$\mu = 0$ の方向はすべての $\lambda > 0$ に対して 0 に設定される（破線）。

- データがほとんど変動しない**定まりの悪い方向**（$\mu_j \ll \lambda$）、たとえばほぼ同一な 2 本の温度計の差 $x_1 - x_2$ の方向では、正則化なしの最小二乗法はノイズをごく小さな μ_j で割るため爆発してしまう（係数の分散は $\sigma^2/(n\mu_j)$ となる）。これに対し、リッジ回帰は $\mu_j/(\mu_j+\lambda) \approx \mu_j/\lambda \ll 1$ を掛けることでノイズの多い成分を抑え込む。
- **観測されない零空間の方向**（$\mu_j = 0$）では、リッジ回帰は重み成分を恒等的に 0 にする。

2.7.3.3 データが定める方向

このスペクトル的な描像を $d = 200$ のデータセット `data` で検証しよう。まず、固有値 μ_j（$\mathbf{S} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$ の固有値）を 300 個の訓練入力（中心化なしだが、スペクトルはほとんど変わらない）に対して計算し、あわせて $\lambda = \frac{1}{3}$ における縮小因子も求める。1.3 節と同様に、`eigh` は固有値を昇順で返し、第 2 戻り値の各列として固有ベクトルを返す。さらに、λ を下回る固有値の個数も数えておく。これは、リッジ回帰が最小二乗座標の半分超を取り除く、定まりの悪い方向を表す。

```
X = data.X[:n]                                  # the 300 training inputs
mu, V = jnp.linalg.eigh(X.T @ X / n)            # increasing eigenvalues
factors = mu / (mu + lambd)                     # shrinkage at lambda = 1/3
k = int((mu < lambd).sum())                     # the poorly determined ones
print(f'mu from {float(mu[0]):.2f} to {float(mu[-1]):.1f}, '
      f'{k} of {len(mu)} below lambda')
print(f'factors from {float(factors[0]):.2f} to {float(factors[-1]):.2f}, '
      f'df = {float(factors.sum()):.0f}')
```

```
mu from 0.04 to 3.2, 54 of 200 below lambda
factors from 0.11 to 0.90, df = 127
```

ゼロになる固有値は 1 つもない。300 個の例があれば、重み空間の 200 次元すべてを張るのに十分だからである。しかし、データが各方向を制約する強さには大きな偏りがある。固有値は数百分の一から約 3.2 までほぼ 2 桁にわたって広がっており、その約 4 分の 1（50〜60 方向）が $\lambda = \frac{1}{3}$ を下回る。対応する縮小因子は約 0.1〜0.9 の範囲に分布する。これらの和は**有効パラメータ数**（effective number of parameters）$\mathrm{df}(\lambda) = \sum_j \mu_j/(\mu_j+\lambda)$ を定義し、ここでは $\mathrm{df}(\frac{1}{3}) \approx 127$ となる。つまり、元の 200 個の重みのうち、正則化されたモデルは実質的に約 127 個の有効パラメータを持つかのように振る舞う。

微小な固有値は最小二乗法にどれほどの悪影響を及ぼすのだろうか。固有ベクトル $\mathbf{v}_j$ に沿って、ラベルのノイズは $\frac{1}{n}(\mathbf{X}\mathbf{v}_j)^\top\epsilon$ を $\frac{1}{n}\mathbf{v}_j^\top\mathbf{X}^\top\mathbf{y}$ に混入させる。これは標準偏差 $\sigma\sqrt{\mu_j/n}$ を持つランダムな摂動である（$\|\mathbf{X}\mathbf{v}_j\|^2 = n\mu_j$ であるため）。最小二乗法はこれを μ_j で割るため、その座標 $\mathbf{v}_j^\top\hat{\mathbf{w}}_0$ は約 $\sigma/\sqrt{n\mu_j}$ だけずれる。このずれは約 0.06（$\mu_j = 1$ のとき）で済むが、実に 0.3（$\mu_j = 0.04$ のとき）にまで達する。真の座標の大きさは $\|\mathbf{w}^*\|/\sqrt{200} = 0.1$ 程度にすぎないため、最小固有値の方向に沿った最小二乗推定量はノイズに完全に埋もれてしまう。一方、検証入力は等方的な標準正規分布に従うため、$\|\mathbf{w} - \mathbf{w}^*\|_2^2 = \sum_j c_j^2$ の直交不変性により、誤差 c_j（$\mathbf{v}_j$ 方向の成分）は期待検証損失を $c_j^2/2$ だけ押し上げることになり、これは μ_j の値によらない。訓練済みの両モデルについて、パラメータ誤差 $\mathbf{w} - \mathbf{w}^*$ を固有ベクトルへ射影し（`V.T @ (w - w_star)`）、定まりの悪い k 個の方向に蓄積した二乗誤差の合計を残りの方向と比較して、これを直接確かめてみよう。

```
w_star = d2l.reshape(data.w, (-1, 1))
for name, model in (('lambda = 0', model_0), ('lambda = 1/3', model_lambd)):
    c = V.T @ (model.w[...] - w_star)         # error along each eigenvector
    print(f'{name:>12}: squared error {float((c[:k] ** 2).sum()):.2f} in the '
          f'{k} poorly determined directions, '
          f'{float((c[k:] ** 2).sum()):.2f} in the rest')
```

```
  lambda = 0: squared error 0.79 in the 54 poorly determined directions, 0.55 in the rest
lambda = 1/3: squared error 0.34 in the 54 poorly determined directions, 0.34 in the rest
```

この内訳がすべてを物語っている。正則化なしでは、全体の約 4 分の 1 にすぎない固有ベクトルが、パラメータの総二乗誤差の半分超を占めている。残りのすべての方向を合わせても約 0.5 であるのに対し、この部分だけで通常 0.8〜2.2 に上る。重み減衰を有効にすると、この定まりの悪い方向の誤差が 2 分の 1 から 6 分の 1 に激減するとともに、残りの方向の誤差も緩やかに削られ、どちらのグループも約 0.3〜0.5 に収まる。よりよく定まる方向においてさえ、真の座標に対してラベルのノイズは無視できない大きさを持つため、0.5〜0.9 倍に縮小することで信号よりも多くのノイズが取り除かれるのである。簡単な検算として、パラメータの総二乗誤差の半分に $\sigma^2/2 = 0.5$ を足すと、両方の実行で観測された検証損失とほぼ一致する。要するに重み減衰は、約 50 個の悪条件な方向にわたって同時に、あの 2 本の温度計の差の方向 $x_1 - x_2$ に対して行ったのと同じ選択的な減衰を実行したのである。

このスペクトル的な視点は、先ほど保留にしていた謎も解き明かしてくれる。なぜペナルティなしの実行は、わずか 2〜3 エポックで最小の検証損失に達したのだろうか。ペナルティなしの損失では、各勾配ステップによって $\mathbf{v}_j$ 方向の現在の座標と最小二乗解との差が $1 - \eta\mu_j$ 倍に収縮する。15 ステップからなる 1 エポック（$\eta = 0.05$）を終えた時点で、大固有値の方向（$\mu_j = 3.2$）ではすでに差の $1 - 0.84^{15} \approx 93\%$ が縮まっているのに対し、小固有値の方向（$\mu_j = 0.04$）ではわずか $1 - 0.998^{15} \approx 3\%$ しか縮まっていない。2 エポック後には、勾配降下法はよく定まる方向ではほぼ収束している一方、ノイズの多い方向ではほとんど動いておらず、リッジ推定量をよく模倣した状態にある。この類似性は定量的に確かめられる。0 に近い重みから出発すると、t ステップ後には割合 $1 - (1 - \eta\mu_j)^t$ だけ $\mathbf{v}_j$ 方向の最小二乗座標が構築されるが、これはリッジ因子 $\mu_j/(\mu_j + \lambda)$ において $\lambda \approx 1/(\eta t)$ とおいたものに近い。実際、どちらもほぼ 1（$\mu_j \gg 1/(\eta t)$ のとき）となり、またほぼ $\eta t \mu_j$（$\mu_j \ll 1/(\eta t)$ のとき）となる。30〜45 ステップ（$\eta = 0.05$）という最良の停止点では、$1/(\eta t)$ は 0.45〜0.67 となり、事前分布が与えた $\lambda = \frac{1}{3}$ と同じオーダーになる。しかし訓練を続けると、定まりの悪い方向がラベルのノイズを着実に吸い上げ、検証損失を再び押し上げてしまう。したがって**早期停止**はそれ自体がれっきとした正則化の一形態であり、この実験では重み減衰に匹敵する精度を達成している。ただし、最適な停止点は学習率やミニバッチノイズによって変化し、実行のたびに検証データ上で監視し続けなければならないため、実運用はより難しい（演習問題 5）。

2.7.4 簡潔な実装

重み減衰はニューラルネットワークの訓練で至るところに使われるため、主要な深層学習フレームワークはいずれもオプティマイザ側で更新則 Equation 2.30 を適用できるようになっており、損失関数を変更する必要はない。便利なだけでなく、この統合は計算面でも無駄がない。オプティマイザはもともとステップごとにすべてのパラメータを 1 回ずつ走査するため、減衰因子を適用してもオーバーヘッドはほとんど生じないのである。また、各フレームワークはパラメータごとの更新則もサポートしており、バイアスには手をつけずに重みだけを減衰させることができる。

ここでは単に `d2l.LinearRegression` を継承してサブクラス `WeightDecay` を定義し、重み減衰のハイパーパラメータ `wd` を `configure_optimizers` 内のフレームワークの SGD オプティマイザに渡す

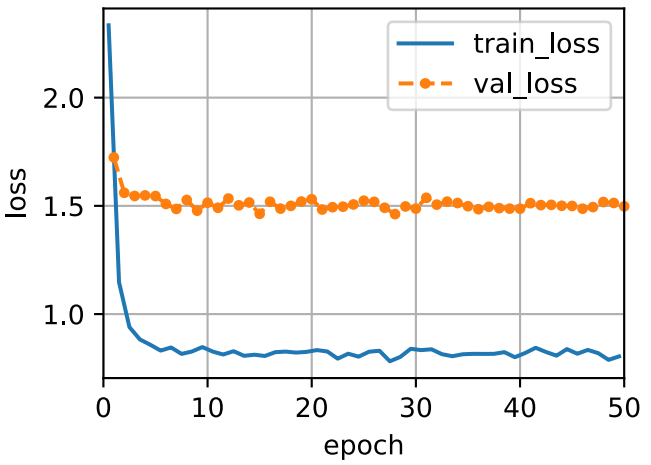
だけでよい（ただしバイアスパラメータは減衰の対象から外し、重み $\mathbf{w}$ のみにペナルティが課されるようにして Equation 2.29 と一致させる）。

`optax.sgd` 自体は重み減衰を直接含まないが、Optax では勾配変換を連鎖させることでオプティマイザを合成する。ここでは `add_decayed_weights` が各パラメータの `wd` 倍をその勾配に加え、`masked` がその変換を述語に一致するパラメータ、すなわち 2 次元以上の配列（バイアスベクトルを除いた重み行列）のみに制限する。

```
def is_matrix(params):
    """Mask that selects the parameters with more than one dimension."""
    return jax.tree.map(lambda p: p.ndim > 1, params)

class WeightDecay(d2l.LinearRegression):
    def __init__(self, wd, num_inputs, lr, rngs=None):
        super().__init__(num_inputs, lr, rngs)
        self.save_hyperparameters(ignore=['rngs'])

    def configure_optimizers(self):
        return optax.chain(
            optax.masked(optax.add_decayed_weights(self.wd), is_matrix),
            optax.sgd(self.lr))
```

`train_scratch(lambd, data)` と厳密に一致させるには、両方のハイパーパラメータをリスケールする必要がある。組み込みの二乗誤差損失には係数 $\frac{1}{2}$ が付いていないため、学習率は 0.025 へと半分にする（2.5 節と同様である）。しかし、減衰項 $\lambda\mathbf{w}$ は損失の勾配から生じるのではなく、オプティマイザの内部で直接加えられる。したがって、学習率を半分にした分、オプティマイザに渡す重み減衰は $2\lambda = \frac{2}{3}$ へと 2 倍にし、各ステップがまったく同じ収縮因子 $1 - 0.025 \cdot \frac{2}{3} = 1 - \frac{1}{60}$ を保つようにしなければならない。

```
trainer = d2l.Trainer(max_epochs=50)
model = WeightDecay(wd=2 * lambd, num_inputs=200, lr=0.025)
trainer.fit(model, data)
norm = float(jnp.linalg.norm(model.get_w_b()[0]))
```



```
print(f'validation loss {trainer.history["val_loss"][-1]:.3f}, '
      f'norm of w {norm:.2f}')
```

```
validation loss 1.498, norm of w 1.22
```

得られた重みのノルム（約 1.2）は $\lambda = \frac{1}{3}$ におけるスクラッチ実装の結果を再現しており、検証損失（スクラッチ実装の約 2 倍の値）もフレームワークの平均二乗誤差における係数 2 の違いを考慮すれば一致する。なお、減衰処理が損失関数ではなくオプティマイザの内部にあるため、プロットされる訓練曲線はペナルティ付き損失ではなく通常の平均二乗誤差を追跡している。

日常的な実践において重み減衰を有効にするには、このオプティマイザのハイパーパラメータを 1 つ渡すだけでよい。合成ベンチマーク以外では、ノイズレベル σ や事前分布のスケール τ が事前にわかっていることはまれなため、通常は複数の候補を実行して検証データ上で λ を調整する。なお、ここで述べたノルムに基づく議論がそのまま当てはまるのは、重みが関数そのものの傾きを表す線形モデルであることに留意しておこう。非線形関数に対して複雑さを厳密に測るのはより繊細な問題であり、再生核ヒルベルト空間（reproducing kernel Hilbert space、RKHS）のような枠組みがノルムペナルティを非線形関数へと拡張しているものの、古典的なカーネル法は大規模データセットに対するスケーラビリティに乏しい。それでもなお、ニューラルネットワークの各層の重みに重み減衰を適用することは、現代の深層学習において最も信頼性の高い正則化手法の 1 つであり続けている。

2.7.5 まとめ

訓練の目的関数に二乗 ℓ_2 ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$ を加えること（ただしバイアス b にはペナルティを課さない）により、高次元モデルがほぼ共線な特徴量の結合方向に沿ってノイズを増幅するのを防ぐことができる。ミニバッチ SGD において、このペナルティは各勾配ステップの前に $\mathbf{w}$ に $(1-\eta\lambda)$ を掛けることと等価であり（これが**重み減衰**という名称の由来である）、等方的なガウス事前分布 $w_j \sim \mathcal{N}(0, \tau^2)$ のもとでの最大事後確率推定に $\lambda = \sigma^2/(n\tau^2)$ として対応する。ℓ_1 ペナルティの鋭い角が重みを厳密に 0 へと追い込むのに対し、滑らかな ℓ_2 ペナルティは $\frac{1}{n}\mathbf{X}^\top\mathbf{X}$ の固有基底における方向別フィルタとして働く。すなわち、各最小二乗成分に $\mu_j/(\mu_j+\lambda)$ を掛けることで、データによってよく定まる方向（$\mu_j \gg \lambda$）を保持し、ノイズの多い方向（$\mu_j \ll \lambda$）を強く減衰させ、観測されない零空間の方向（$\mu_j = 0$）を 0 にする。

2.7.6 演習問題

1. **ℓ_1 の更新則**。ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$ を $\lambda\sum_j |w_j|$ に置き換えよ。
    1. $\mathbf{w}$ に対するミニバッチ SGD の更新則を導出せよ。ただし、劣勾配 $\operatorname{sign}(w_j)$（演習問題 2）を $|w_j|$ の導関数の代わりに用いよ。
    2. この更新則の 1 ステップにおいてペナルティ単独で w_j に生じる変化量を、更新則 Equation 2.30 における変化量と、$|w_j|$ が大きい場合と小さい場合のそれぞれについて比較せよ。
    3. (a) の更新則では、w_j が厳密に 0 になることはほとんどない。これを L に関する勾配ステップとそれに続くペナルティステップとに分割し、ペナルティステップが w_j の符号を反転させてしまう場合には常に w_j を 0 に設定するよう修正せよ。その結果を Lasso が行う特徴量選択と関連づけて説明せよ。この修正されたステップは**ソフト閾値処理**であり、ℓ_1 ノルムの近接ステップ（35.6.4.1 節）である。
2. **縮小とペナルティの比較**。重み減衰は、勾配ステップの前に重みを縮小する方法でも、$\frac{\lambda}{2}\|\mathbf{w}\|^2$ を損失 L に加える方法でも適用できる。

1. 更新則 $\mathbf{w} \leftarrow (1-\eta\lambda)\mathbf{w} - \eta\nabla L$ が、ペナルティ付き損失 $L + \frac{\lambda}{2}\|\mathbf{w}\|^2$ に対する勾配ステップと同一であることを示せ。

2. ステップの前に、勾配が座標ごとに固定された定数 $a_j > 0$ でリスケールされるとする。先に縮小してからステップを進める方法では下の第 1 式が得られ、ペナルティ付き勾配でステップを進める方法では第 2 式が得られる。

$$w_j \leftarrow (1-\eta\lambda)\, w_j - \eta a_j\, \partial_{w_j} L \qquad \text{and} \qquad w_j \leftarrow w_j - \eta a_j \left(\partial_{w_j} L + \lambda w_j\right). \tag{2.36}$$

どのような a_j に対して両者が一致するか、また第 2 式が w_j を何倍に縮小するか（$\partial_{w_j} L = 0$ となるステップにおいて）を求めよ。深層学習で用いられる多くのオプティマイザは、このように勾配をリスケールする。2 つの更新則のうち、すべての重みに対して λ を同一の減衰率に保つのはどちらか。

Simon Prince『Understanding Deep Learning』問題 9.5 を改変。

3. [code] **拡張最小二乗法としてのリッジ回帰。** 300 個の訓練入力とラベルを中心化してバイアス項を消去し、$\lambda = \frac{1}{3}$ に対するリッジ推定量を次の 2 通りの方法で計算せよ。

   1. 連立一次方程式 Equation 2.34 を、たとえば `np.linalg.solve` を用いて直接解け。
   2. `lstsq` を用いて、$\mathbf{X}$ の下に $\sqrt{n\lambda}\,\mathbf{I}$ を縦に連結した行列と、ラベル $\mathbf{y}$ の後に d 個の 0 を続けたベクトルに対して通常の最小二乗法を実行せよ。

   両者が丸め誤差の範囲で一致することを確かめ、その理由を説明せよ。さらに、得られた解のノルムを `train_scratch(lambd, data)` が返したノルムと比較し、両者の差を生みうる 2 つの手順の違いを挙げよ。

   Hastie、Tibshirani、Friedman『The Elements of Statistical Learning』演習 3.12 を改変。

4. [code] λ **のスイープ。** `WeightDecayScratch` を `data` 上で $\lambda \in \{0.01, 0.1, 0.3, 1, 3, 10, 30\}$ の各値について訓練せよ。

   1. 最終的な検証損失と $\mathbf{w}$ のノルムを対数軸の λ に対してプロットせよ。その際、`train_scratch(0, data)` の値を水平線として描き添え、検証損失が最小となる λ を報告せよ。
   2. 同じ各値に対する $\mathrm{df}(\lambda)$ を 2.7.3 節の固有値から計算し、検証損失を $\mathrm{df}(\lambda)$ に対してプロットせよ。その曲線の形状を Figure 2.11 と比較せよ。
   3. 事前分布が示唆する値 $\lambda = \frac{1}{3}$ は、スイープにおける最良値に近いか。ノイズ $\sigma = 0.5$ のデータと $\sigma = 2$ のデータをそれぞれ生成し、各場合について事前分布による λ を再計算して、スイープの最良値に追随するかどうかを確かめよ。

5. [code] **早期停止と重み減衰の比較。** 本節のペナルティなしの実行は、2〜3 エポック後に検証損失の最小値に達し、その値は $\lambda = \frac{1}{3}$ の実行が最終的に到達する値とほぼ同じくらい低い。

   1. 0 に近い重みから出発すると、ペナルティなし損失に対する勾配降下法は、$\mathbf{v}_j$ 方向の座標とその最小二乗値との差を 1 ステップごとに $1-\eta\mu_j$ 倍するため、t ステップ後にはその座標の割合 $1-(1-\eta\mu_j)^t$ を構築する。この因子を μ_j（2.7.3 節の固有値）に対してプロットせよ。ただし、t は 1 エポック・2 エポック・50 エポック（1 エポックあたり 15 ステップ、$\eta = 0.05$）とし、リッジ因子 $\mu_j/(\mu_j+\lambda)$（$\lambda = \frac{1}{3}$）も並べて描け。どの t がリッジに最も近いか。

2. `WeightDecayScratch` を $\lambda = 0$ で 2 エポック訓練し、2.7.3 節と同様に固有ベクトルに沿って誤差を分解せよ。定まりの悪い k 個の方向と残りの方向における二乗誤差を、同項で出力された 50 エポック訓練時（$\lambda = 0$ および $\lambda = \frac{1}{3}$）の値と比較せよ。
3. ペナルティなしの実行を $\eta = 0.01$ と $\eta = 0.1$ で繰り返し、`trainer.history` から最良のエポックを読み取れ。最良エポックがどのように移動するかを説明し、ペナルティには不要だが早期停止では実用上必要となるものを述べよ。

6. [code] **訓練例より多い重み。**●200 個の特徴量とわずか 20 個の訓練例を持ち、真の重みが 0.01、バイアスが 0.05、ノイズが $\sigma = 0.01$ であるデータをミニバッチサイズ 5 で生成せよ。
   1. $\frac{1}{n}\mathbf{X}^\top\mathbf{X}$ の固有値を 20 個の訓練入力に対して計算せよ。0 になる固有値は何個あるか。また、それらの固有ベクトルに沿った重みを勾配降下法が決して変化させないのはなぜか。
   2. $\lambda = 0$ と $\lambda = 3$ のそれぞれについて、30 エポック、学習率 0.01 で訓練せよ。その際、初期値を `WeightDecayScratch` のランダム初期化とした場合と、$\mathbf{w} = \mathbf{0}$（`sigma=0` を渡す）とした場合の 2 通りを試せ。実行する前に、定まらない方向における初期重みのノルムから 4 つの検証損失を予測し、その後に実行結果と比較せよ。
   3. 4 つの損失すべてを、常に b^* と予測するモデルの損失と比較せよ。ここで重み減衰は何を達成し、またどのような λ の値をもってしても達成できないことは何か。

Chapter 3

線形分類

第 2 章では、住宅の販売価格のような連続量を予測することが目標であった。しかし、多くのタスクでは代わりに離散的なカテゴリが求められる。たとえばオンラインカタログの写真に写っているのはシャツ、コート、サンダルのいずれかであり、何かの「5.5」では決してない。カテゴリに 1 から 10 までの番号を単純に振って線形回帰を適用しても、まったくうまくいかない。プルオーバーがズボンとドレスのちょうど中間にあるかのような恣意的な順序を押しつけてしまううえ、較正（calibration）された不確実性（「おそらくシャツだが、プルオーバーかもしれない」）をモデルが表現できないからである。

幸い、ミニバッチデータローダ、ベクトル化された順伝播、自動微分、ミニバッチ SGD といった回帰の訓練機構は、そのほぼすべてがそのまま**分類**（classification）に引き継がれる。変わるのは 3 つの要素だけである。順序のないカテゴリを**ワンホットベクトル**で表し、カテゴリごとに 1 つの線形スコア（**ロジット**）を計算して**ソフトマックス**関数により確率分布へと正規化し、二乗誤差を**交差エントロピー損失**（cross-entropy loss）に置き換える。この損失は、二乗誤差がガウスノイズから導かれたのと同様に、カテゴリラベルのモデルにおける最尤法から導かれる。本書で扱うニューラル分類器は、大規模言語モデルに至るまで、そのほぼすべてが最終層にこのソフトマックス層を持ち、この損失のもとで訓練される。

本章ではまず、ソフトマックス回帰（softmax regression）と交差エントロピー損失を定式化し、その残差勾配 $\hat{\mathbf{y}} - \mathbf{y}$ を導出して、Shannon の情報理論と結びつける（3.1 節）。次に、再利用可能な `FashionMNIST` 画像パイプラインを構築し（3.2 節）、ソフトマックス回帰をゼロからの方法と簡潔な方法の両方で実装する（3.3 節）。そこでは、混同行列（confusion matrix）と学習された 28×28 のテンプレートを用いて誤りを診断するとともに、融合された log-sum-exp 損失によって浮動小数点のオーバーフローとアンダーフローを解消する。

分類が回帰と大きく異なるのは、本章の後半を占める**評価**と**デプロイ**である。陽性例がまれな場合や誤りによってコストが異なる場合には、単純な正解率（accuracy）は誤解を招く。これが適合率（precision）、再現率（recall）、コスト考慮型の閾値、および較正の診断を導入する動機となる（3.4 節）。クラスが均衡していても、有限のテストセットには統計的なエラーバーが伴い、繰り返し再利用するとその保証が損なわれる。そこで Hoeffding 限界、適応的過学習（adaptive overfitting）、および Vapnik–Chervonenkis（VC）理論へと進む（第 3.5 章）。最後に 3.6 節では、デプロイ時のデータが訓練分布から乖離したときに何が起こるかを問い、重点重み付けによる共変量シフト（covariate shift）の補正、混同行列を用いたラベルシフトの推定、および暴走するフィードバックループへの防御策を展開する。

3.1 ソフトマックス回帰

回帰は、**どれくらいの量か**や**いくつあるか**といった問いに答えるための手法である。しかし、教師あり学習の多くの問題では、代わりに**どのカテゴリか**が問われる。たとえば、スパムフィルタはメールを 2 つのクラスに振り分け、臨床支援ツールは複数の診断候補から選択し、言語モデルは数万語の語彙から次のトークンを予測する。

あるオンライン小売業者が、商品の写真を T シャツ、ズボン、プルオーバー、ドレス、コート、サンダル、シャツ、スニーカー、バッグ、アンクルブーツという 10 個のアパレルカテゴリに分類したいとする。これらのカテゴリに 1 から 10 までの番号を振り、線形回帰をそのまま適用してはいけないのはなぜか。順序のないカテゴリに対する整数コードは単なる便宜上の名前にすぎず、数値的な量ではないからである。たとえば、5.5 という予測は写真をコートとサンダルのちょうど中間に位置づけてしまう。また二乗誤差では、T シャツ（1）をアンクルブーツ（10）と誤ると、ズボン（2）と誤る場合より 81 倍も重く罰せられる。さらに、最終的に**ハード割り当て**（hard assignment、単一のカテゴリを選ぶこと）が必要な場合であっても、後続の意思決定で確信度と誤りのコストを比較衡量できるよう、**ソフト割り当て**（soft assignment、すべての候補クラスに対する較正された確率）を出力するモデルがほぼ常に求められる。

ソフトマックス回帰（softmax regression）は**多項ロジスティック回帰**（multinomial logistic regression）とも呼ばれ、この両方の問題を明快に解決する。すなわち、クラスごとに 1 つのアフィンスコアを計算し、ソフトマックス演算子によってそれらのスコアを有効な確率分布へと写像する。2.1 節の最尤法（maximum likelihood）の手順に従い、観測されたカテゴリの負の対数尤度（negative log-likelihood）を最小化すると、**交差エントロピー損失**（cross-entropy loss）が得られる。

3.1.1 モデル

3.1.1.1 クラスラベル

まずは具体例として、2×2 のグレースケール画像分類（image classification）問題を考えよう。この問題は 4 つのスカラーのピクセル特徴量 x_1, x_2, x_3, x_4 を持ち、猫（1）、ニワトリ（2）、犬（3）の 3 つのカテゴリを扱う。一般に、クラス数が q のとき、各データ例のラベルは整数 $y \in \{1, \dots, q\}$ である。

順序のないカテゴリに誤った数値的順序を持ち込まずに表現するため、統計学では**ワンホットエンコーディング**を用いる。これはベクトル $\mathbf{y} \in \{0, 1\}^q$ であり、正解クラスの成分を 1（$y_y = 1$）、それ以外のすべての成分を 0 とする表現である。先ほどの 3 クラスの例では、猫は $(1, 0, 0)$、ニワトリは $(0, 1, 0)$、犬は $(0, 0, 1)$ となる。これにより、異なるどの 2 クラス間も等しいユークリッド距離 $\sqrt{2}$ に配置される。また $\mathbf{y}$ 自体が有効な確率分布（すべての確率質量を正解クラスに置く退化分布）になっており、モデルが予測する分布 $\hat{\mathbf{y}}$ と直接比較できる。（数式では 1 始まりのクラスインデックス y とワンホットベクトル $\mathbf{y}$ を用いるが、Python コードでは 0 始まりの整数インデックスをそのまま格納する。）

3.1.1.2 線形スコア

すべての q 個のクラスに同時にスコアを付けるため、ソフトマックス回帰は各クラス $j \in \{1, \dots, q\}$ に対して、入力特徴量の独立したアフィン関数を割り当てる。重み w_{kj} を特徴量 k からクラス j への結合重み、b_j をクラス j のバイアスと表すと、この 2×2 の画像分類器は正規化されていない 3 つのスコア（**ロジット**）を次の式で計算する。

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41} + b_1,\\
o_2 &= x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42} + b_2,\\
o_3 &= x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43} + b_3.
\end{aligned}
$$

ニューラルネットワークとして見ると（Figure 3.1）、ソフトマックス回帰は d 個の入力と q 個の出力ユニットを持つ単層の全結合（fully connected）ネットワークである（重みは $d \cdot q$ 個、バイアスは q 個）。

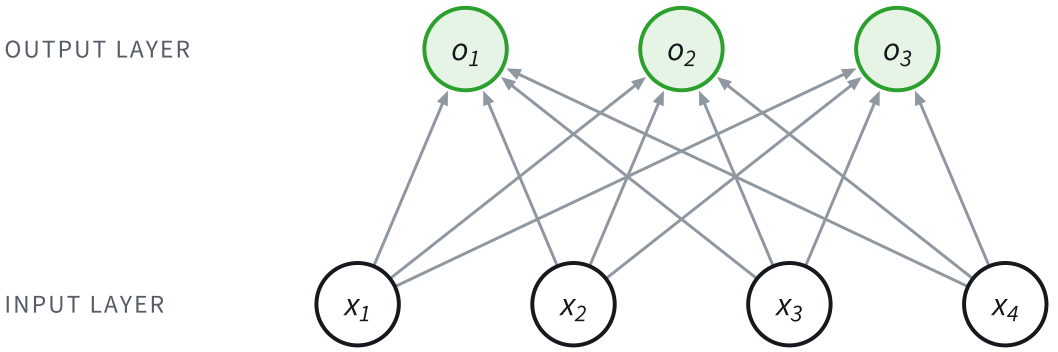


Figure 3.1: 単層の全結合ニューラルネットワークとして表したソフトマックス回帰。各入力 x_k は各出力ロジット o_j に重み w_{kj} を介して結合する。

重みを行列 $\mathbf{W} \in \mathbb{R}^{d \times q}$（特徴量ごとに 1 行、クラスごとに 1 列 $\mathbf{w}_j \in \mathbb{R}^d$）にまとめ、バイアスを $\mathbf{b} \in \mathbb{R}^q$ にまとめると、入力 $\mathbf{x} \in \mathbb{R}^d$ に対するロジットベクトルは次のように簡潔に書ける。

$$
\mathbf{o} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}, \tag{3.1}
$$

ここで、$o_j = \mathbf{w}_j^\top \mathbf{x} + b_j$ である。28×28 の Fashion-MNIST 衣類画像を 3.2 節で分類する際には、$d = 784$、$q = 10$ となり、7,840 個の重みと 10 個のバイアスが得られる。

3.1.1.3 ソフトマックス

線形層が、猫、ニワトリ、犬に対してロジット $\mathbf{o} = (1.0, 2.2, 0.3)$ を出力したとする。ニワトリのスコアが最も高いが、生のロジットは確率ではない。その和は 3.5 であって 1 ではなく、アフィンスコアが負になったり 1 を超えたりすることを防ぐ仕組みもない。任意の実数ベクトル $\mathbf{o} \in \mathbb{R}^q$ を、スコアのランキングを保ったまま有効な確率分布 $\hat{\mathbf{y}}$（q 個の正の数からなり、その和が 1 になるもの）に変換するため、**ソフトマックス**関数は各ロジットの指数をとり、その和で割る。

$$
\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)}. \tag{3.2}
$$

$\mathbf{o} = (1.0, 2.2, 0.3)$ の場合、各成分の指数は $(2.72, 9.03, 1.35)$、その和は 13.09 となり、$\hat{\mathbf{y}} = (0.21, 0.69, 0.10)$ が得られる。すなわち、ニワトリの予測確率は 69% である。exp は狭義単調増加関数であるため、ソフトマックスはロジットの大小関係を保つ。したがって、最も確率の高いクラスは常に最大のロジットを持つクラスと一致する。

$$\operatorname*{argmax}_j \hat{y}_j = \operatorname*{argmax}_j o_j. \tag{3.3}$$

推論時に最も予測確率の高いカテゴリだけが必要な場合は、指数の計算をすべて省き、単に $\operatorname{argmax}_j o_j$ を求めればよい。

なぜ他の正の単調増加関数ではなく、指数関数を使うのだろうか。それは、指数関数が加法を乗法に変換することで、ソフトマックスに次の 3 つの基本的な性質をもたらすからである。

1. **シフト不変性（ロジット間の差のみが重要）：**任意のスカラー定数 c を $\mathbf{o}$ のすべての成分に加えても、Equation 3.2 の分子と分母の両方に $\exp(c)$ が掛かるため、確率は変化しない。すなわち、$\operatorname{softmax}(\mathbf{o} + c\mathbf{1}) = \operatorname{softmax}(\mathbf{o})$ が成り立つ。言い換えれば、任意の 2 つのロジットの差は予測確率の対数オッズ比（log-odds ratio）に等しい。

$$o_i - o_j = \log \frac{\hat{y}_i}{\hat{y}_j}. \tag{3.4}$$

   先ほどの例では $o_2 - o_1 = 1.2$ であるため、ニワトリは猫の $e^{1.2} \approx 3.3$ 倍の確率となる（$0.69/0.21$）。このシフト不変性（shift invariance）により、重要な数値計算上の安全策も得られる（3.3 節）。指数をとる前に $o_{\max} = \max_k o_k$ を引いておけば、計算結果を変えずに浮動小数点オーバーフローを防げるのである。
2. **$q = 2$ におけるロジスティック回帰への帰着：**2 クラスの場合、確率は単一の差 $o = o_1 - o_2$ のみに依存する。

$$\hat{y}_1 = \frac{\exp(o_1)}{\exp(o_1) + \exp(o_2)} = \frac{1}{1 + \exp(-o)} = \sigma(o), \tag{3.5}$$

   ここで、$\sigma(o)$ は**ロジスティックシグモイド**である。したがって、2 クラスのソフトマックス回帰は古典的な**ロジスティック回帰**（logistic regression）と同一であり、$o = \log(\hat{y}_1/\hat{y}_2)$ は対数オッズ（log-odds）を表す。これが**ロジット**という語の歴史的な由来である。
3. **最尤法における簡潔な勾配：**後述するように、ソフトマックスの指数関数と尤度の対数関数が対になることで、損失の勾配は残差（residual）$\hat{\mathbf{y}} - \mathbf{y}$ という簡潔な形に帰着する。

3.1.1.4 ミニバッチ

n 個のデータ例（各例は d 個の特徴量を持つ）からなるミニバッチに対してベクトル化するため、入力を $\mathbf{X} \in \mathbb{R}^{n \times d}$、重みを $\mathbf{W} \in \mathbb{R}^{d \times q}$、バイアスを $\mathbf{b} \in \mathbb{R}^{1 \times q}$ にまとめる。

$$\begin{aligned} \mathbf{O} &= \mathbf{X}\mathbf{W} + \mathbf{b}, \\ \hat{\mathbf{Y}} &= \operatorname{softmax}(\mathbf{O}), \end{aligned} \tag{3.6}$$

ここで、$\mathbf{b}$ は n 個の行（$\mathbf{O} \in \mathbb{R}^{n \times q}$ の各行）にわたってブロードキャストされ、softmax は**行ごと**に適用される（$\mathbf{O}$ の要素ごとに指数をとり、各行をその行の和で割る）。

3.1.1.5 決定領域

ソフトマックスはアフィンロジット $o_j = \mathbf{w}_j^\top \mathbf{x} + b_j$ の大小関係を保つため、ソフトマックス回帰は入力空間 $\mathbb{R}^d$ を平坦な面で囲まれた凸多面体へと分割する（Figure 3.2）。クラス j が選ばれるのは、$o_j \geq o_k$ がすべての $k \neq j$ に対して成り立つ領域であり、これは $q - 1$ 個の半空間 $(\mathbf{w}_j - \mathbf{w}_k)^\top \mathbf{x} + (b_j - b_k) \geq 0$ の共通部分にほかならない。隣接するクラス同士は、2 つのロジ

ットが等しくなる超平面に沿って接する。線形回帰と同様に、クラス間の境界は常に**平坦な**超平面であり、曲がった境界や非凸な決定領域（decision region）を切り出すには隠れ層が必要となる（第 4 章）。

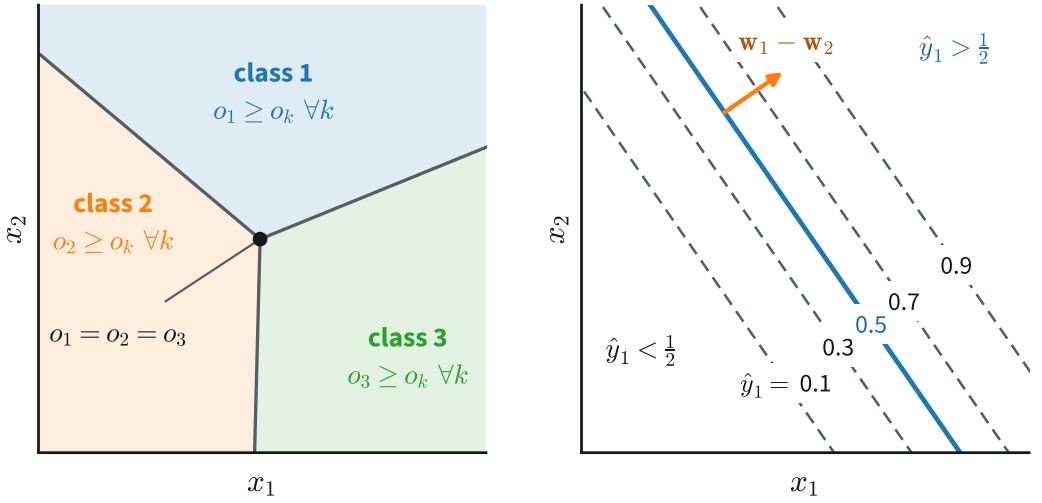


Figure 3.2: $\mathbb{R}^2$ におけるソフトマックス回帰の決定領域。左：3 クラスの場合、平面は 3 つの凸領域に分割され、ロジットの対が等しくなる直線に沿って接する。右：2 クラスの場合、等確率の等高線 $\hat{y}_1 = \sigma(o)$ は $\mathbf{w}_1 - \mathbf{w}_2$ に直交する平行線となり、決定境界は $\hat{y}_1 = 1/2$ にある。

ここで前提としている構造上の基本仮定に留意しておこう。それは、各データ例が**順序のない**集合から選ばれた**ただ 1 つの**クラスに属するという点である。同一の入力に複数のタグが同時に当てはまりうる場合（**マルチラベル分類**（multi-label classification））には、競合するソフトマックスではなくタグごとに独立した二値シグモイドを用いる (Tsoumakas and Katakis 2007)。また、クラスが自然な順序に従う場合（**順序回帰**（ordinal regression））には、その順序構造を活用する専用の閾値モデルを用いる (Moon et al. 2010)。

3.1.2 交差エントロピー損失

3.1.2.1 最尤法

予測された確率分布 $\hat{\mathbf{y}}$ と観測されたラベル y の食い違いは、どのように測ればよいだろうか。2.1 節の最尤法の原理に従い、$\hat{y}_j = P(y = j \mid \mathbf{x})$ をクラス j の条件付き確率（conditional probability）と解釈する。ワンホットベクトル $\mathbf{y} \in \{0, 1\}^q$ を用いると、観測されたクラス y に割り当てられる確率は次のように書ける。

$$P(y \mid \mathbf{x}) = \hat{y}_y = \prod_{j=1}^{q} \hat{y}_j^{y_j}.$$

n 個の独立な訓練例にわたる負の対数尤度は、データ例ごとの損失の和となる。

$$
\begin{aligned}
-\log \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}) &= \sum_{i=1}^{n} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}), \\
\text{where} \quad l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \hat{y}_j = -\log \hat{y}_y.
\end{aligned}
\tag{3.7}
$$

この目的関数が**交差エントロピー損失**である。その振る舞いを確認しておこう。

- $\hat{y}_y \in (0, 1)$ であるため、常に $l(\mathbf{y}, \hat{\mathbf{y}}) > 0$ である。また、$\hat{y}_y \to 1$ のとき 0 に近づき、$\hat{y}_y \to 0$ のとき $+\infty$ に発散する（確信を持った誤りに重いペナルティを科す）。
- ロジットが $(1.0, 2.2, 0.3)$、確率が $(0.21, 0.69, 0.10)$ であった先ほどの画像において、正解ラベルがニワトリ（$y = 2$）なら損失は $-\log 0.69 \approx 0.37$ nat であるが、もし実際は犬（$y = 3$）であれば、損失は 6 倍の $-\log 0.10 \approx 2.30$ nat へと跳ね上がる。

3.1.2.2 勾配

交差エントロピー損失をロジット $\mathbf{o}$ で微分してみよう。Equation 3.2 を Equation 3.7 に代入し、$\sum_{j=1}^{q} y_j = 1$ を用いると、損失は $\mathbf{o}$ によって直接次のように表される。

$$
\begin{aligned}
l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} \\
&= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \\
&= \log \sum_{k=1}^{q} \exp(o_k) - o_y.
\end{aligned}
$$

第 1 項の $\log \sum_{k=1}^{q} \exp(o_k)$ は **log-sum-exp** 関数であり、$\max_k o_k$ の滑らかな凸上界となっていて、$\max_k o_k$ を超える量は高々 $\log q$ に抑えられる（演習問題 4）。これを o_j で微分すると、次式が得られる。

$$
\begin{aligned}
\frac{\partial l}{\partial o_j} &= \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \operatorname{softmax}(\mathbf{o})_j - y_j, \\
\text{or} \quad \nabla_{\mathbf{o}} l &= \hat{\mathbf{y}} - \mathbf{y}.
\end{aligned}
$$

二乗損失のもとでの線形回帰とまったく同様に、**自然パラメータ $\mathbf{o}$ に関する勾配は、単に予測から観測を引いた差 $\hat{\mathbf{y}} - \mathbf{y}$ となる**。これは指数分布族（exponential family）の一般化線形モデル（generalized linear model）に共通する普遍的な性質である（36.4.3.3 節）。先ほどのニワトリの画像（$\mathbf{y} = (0, 1, 0)$）では $\nabla_{\mathbf{o}} l = (0.21, -0.31, 0.10)$ となり、勾配降下法を 1 ステップ進めると、ニワトリのロジットが押し上げられる一方、猫と犬のロジットはそれぞれの競合クラスが誤って奪った確率質量に厳密に比例して押し下げられる。さらに $\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}$ を介して連鎖律（chain rule）を適用すると、$\mathbf{W}$ に関するミニバッチ勾配は $\frac{1}{n}\mathbf{X}^\top(\hat{\mathbf{Y}} - \mathbf{Y})$ となり、損失は $(\mathbf{W}, \mathbf{b})$ に関して大域的に凹（convex）となる。実際、$\mathbf{o}$ に関するヘッセ行列（Hessian）は $\hat{\mathbf{y}}$ からのカテゴリカルな抽出の共分散行列（covariance matrix）に一致する（演習問題 2）。

	線形回帰	ソフトマックス回帰
目的変数	連続値 $y \in \mathbb{R}$	クラス $y \in \{1, \dots, q\}$、ワンホット $\mathbf{y}$
予測	$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$	$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$
確率モデル	ガウス分布 $\mathcal{N}(\hat{y}, \sigma^2)$	カテゴリカル分布 $\mathrm{Cat}(\hat{\mathbf{y}})$
損失（NLL）	$\frac{1}{2}(\hat{y} - y)^2$	$-\sum_{j=1}^{q} y_j \log \hat{y}_j = -\log \hat{y}_y$
勾配	$\hat{y} - y$（$\hat{y}$ に関して）	$\hat{\mathbf{y}} - \mathbf{y}$（$\mathbf{o}$ に関して）

3.1.2.3 エントロピーと交差エントロピー

交差エントロピー（cross-entropy）という名称は、Claude Shannon の情報理論（information theory）に由来する（37.1 節）。確率 p の事象が生起したとき、Shannon はその**サプライザル**（情報量、information content）を $-\log p$ と定義した（自然対数を用いる場合は **nat**、$\log_2$ を用いる場合は **bit** を単位とし、$1\ \text{nat} = 1/\log 2 \approx 1.44\ \text{bits}$ である）。真の分布 $\mathbf{p} = (p_1, \dots, p_q)$（$q$ 個のクラス上の分布）に対して、サプライザルの期待値がその**エントロピー**である。

$$H(\mathbf{p}) = -\sum_{j=1}^{q} p_j \log p_j, \tag{3.8}$$

この値は 0（1 つの事象が確実に起こる場合）から $\log q$（q 個すべての事象が等確率で起こる場合）までの範囲をとる。ここで、$\mathbf{p}$ から生じる事象を、モデル分布 $\hat{\mathbf{p}}$ に最適化された符号を使って符号化すると、その平均符号長（サプライザルの期待値）が**交差エントロピー**となる。

$$H(\mathbf{p}, \hat{\mathbf{p}}) = -\sum_{j=1}^{q} p_j \log \hat{p}_j. \tag{3.9}$$

ギブスの不等式（Gibbs' inequality）により $H(\mathbf{p}, \hat{\mathbf{p}}) \geq H(\mathbf{p})$ が成り立ち、等号が成立するのは $\hat{\mathbf{p}} = \mathbf{p}$ のとき、かつそのときに限る（両者の差はカルバック・ライブラー・ダイバージェンス $D_{\mathrm{KL}}(\mathbf{p} \,\|\, \hat{\mathbf{p}}) = H(\mathbf{p}, \hat{\mathbf{p}}) - H(\mathbf{p}) \geq 0$ である）。$H(\mathbf{p}, \hat{\mathbf{p}})$ を Equation 3.7 と比較すると、データ例ごとの損失 $l(\mathbf{y}, \hat{\mathbf{y}}) = H(\mathbf{y}, \hat{\mathbf{y}})$ は、文字通り経験ラベル分布 $\mathbf{y}$ と予測分布 $\hat{\mathbf{y}}$ の交差エントロピーにほかならない。さらに都合のよいことに、入力が本質的に曖昧な場合（たとえば、シャツである確率が 60%、T シャツである確率が 40% のぼやけた写真）でも、モデルが過剰に確信した推測ではなく真の較正された確率 $(0.6, 0.4)$ を出力するときに交差エントロピー損失の期待値が一意に最小化されることが、ギブスの不等式によって保証される。

3.1.3 正解率と確信度

3.1.3.1 誤り率

分類器の評価には**正解率**（accuracy）や**誤り率**（error rate、$= 1 - \text{accuracy}$）を用いるが、0–1 誤差を直接微分してモデルを訓練することはできない。データ例の**マージン**を、正解クラスのロジットから競合クラスの中で最大のロジットを引いた値として定義しよう。

$$m = o_y - \max_{j \neq y} o_j. \tag{3.10}$$

分類器が誤るか（$m < 0$）正しい予測をするか（$m > 0$）は m の不連続な階段関数で決まるが（Figure 3.3）、その導関数は存在するすべての点で 0 となるため、勾配降下法には学習のための信号がまったく届かない。交差エントロピーはこの平坦な階段関数を、滑らかで狭義単調減少する凸な代理損失（$\log(1 + e^{-m})$、ただし $q = 2$ の場合）に置き換える。この代理損失は（$\log 2$ で割って bit 単位で測れば）0–1 誤差の上界となり、すべてのデータ例に対して非ゼロの勾配を与える。

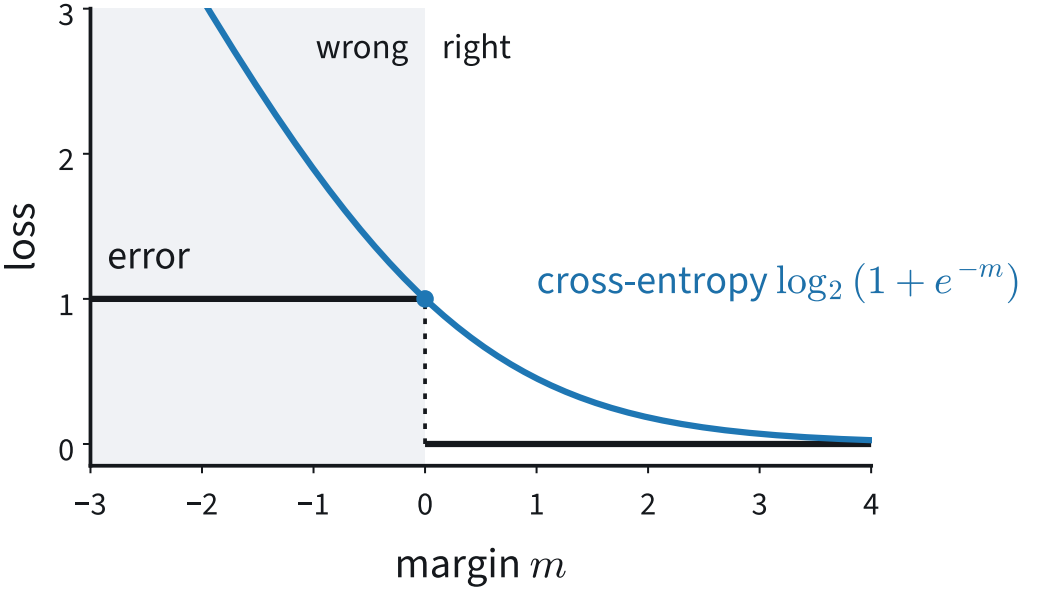


Figure 3.3: 0–1 分類誤差と 2 クラスの交差エントロピー損失（bit 単位、$\log_2(1 + e^{-m})$）をマージン $m = o_y - \max_{j \neq y} o_j$ の関数として描いた図。

3.1.3.2 較正と温度

Figure 3.3 の滑らかな曲線がもたらす 1 つの帰結に注目しよう。交差エントロピーは、$m > 0$ となって分類の判定がすでに正解した後も減少し続ける。もし訓練データセットが**線形分離可能**（linearly separable）であれば、すなわちある $(\mathbf{W}, \mathbf{b})$ がすべての訓練例を正しく分類できる（$m^{(i)} > 0$ がすべての i で成り立つ）ならば、$(\mathbf{W}, \mathbf{b})$ に任意のスケール因子 $c > 1$ を掛けるとすべてのマージンが c 倍になり、訓練損失はどこまでも 0 に近づいていく（演習問題 6）。その結果、決定境界（decision boundary）はまったく動かないまま、すべての予測確率が 0 と 1 に張り付いていく。

より一般に、ソフトマックスを適用する前にすべてのロジットを正の**温度**（temperature）$T > 0$ の逆数でスケーリングすると、次のようになる。

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o}/T), \tag{3.11}$$

この操作は、分布をワンホットな argmax へと先鋭化させ（$T \to 0$）、逆に一様分布 $(1/q, \ldots, 1/q)$ へと平坦化させるが（$T \to \infty$、Figure 3.4）、ロジットのランキング（したがって分類器の正解率）は変えない。訓練されたモデルの予測確率が実際の経験的な正解率と一致しているか（**較正**、calibration）は実証的な問いであり、3.4 節でホールドアウトデータを用いて検証する。

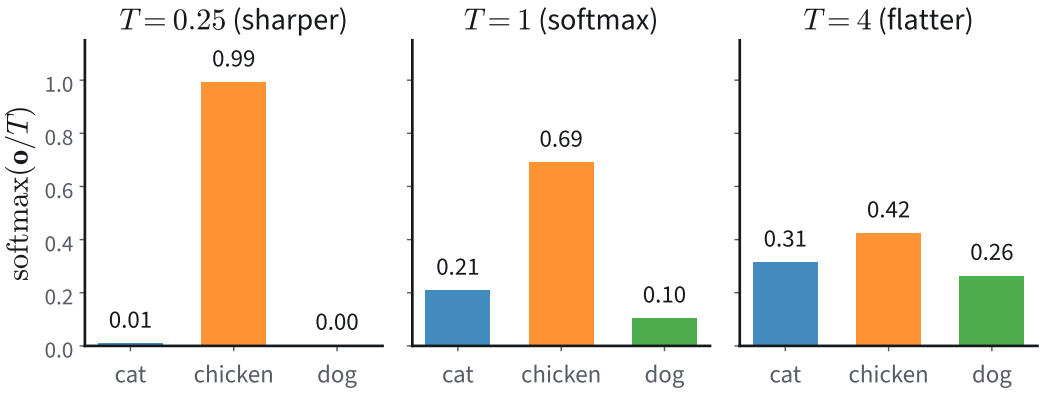


Figure 3.4: ロジット $\mathbf{o} = (1.0, 2.2, 0.3)$ に対する 3 つの温度でのソフトマックス確率。$T = 0.25$（鋭いピーク）、$T = 1$（標準のソフトマックス）、$T = 4$（ほぼ一様）。

3.1.4 まとめ

ソフトマックス回帰は、クラスごとに 1 つのアフィンロジット $\mathbf{o} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$ を計算し、それを確率 $\hat{y}_j = \exp(o_j) / \sum_k \exp(o_k)$ へと正規化することで順序のない離散カテゴリを予測し、入力空間を超平面で囲まれた凸多面体の決定領域へと分割する。ソフトマックス演算子はロジットの対ごとの差 $o_i - o_j = \log(\hat{y}_i / \hat{y}_j)$ のみに依存するため、$\mathbf{o}$ の一様なシフトに対して不変であり、$q = 2$ のときには二値のロジスティック回帰に帰着する。ワンホットの目的変数 $\mathbf{y}$ に対するカテゴリカル尤度を最大化すると、凸な交差エントロピー損失 $l(\mathbf{y}, \hat{\mathbf{y}}) = -\log \hat{y}_y = \log \sum_k \exp(o_k) - o_y$ が得られる。これは 0–1 分類誤差の滑らかな上界であり、ロジットに関する勾配は残差 $\hat{\mathbf{y}} - \mathbf{y}$ へと簡約される。ロジットを温度 $T > 0$ で割れば、クラスのランキングや正解率を変えることなく、決定論的な argmax から一様分布まで予測の確信度を調整できる。

3.1.5 演習問題

1. **ソフトマックスとシグモイド。** Equation 3.5 により、ロジスティック回帰は冗長なロジットを 1 つ取り除いた 2 クラスのソフトマックス回帰である。同僚から、当てはめ済みのロジスティックモデル $P(y = 1 \mid x) = \sigma(\hat{\beta}_0 + \hat{\beta}_1 x)$（ただし $\hat{\beta}_0 = 2$、$\hat{\beta}_1 = -1$）を渡されたとする。

    1. 2 クラスのソフトマックス係数 (w_1, b_1) と (w_2, b_2) のうち、すべての x に対してこのロジスティックモデルと予測が一致するものをすべて特徴づけ、さらに $o_2 \equiv 0$ を満たす組を特定せよ。
    2. 2 クラスのソフトマックスモデルの交差エントロピー損失に対する勾配降下法が、$w_1 + w_2$ および $b_1 + b_2$ を決して変化させないことを示せ。同僚の訓練データ上で勾配降下法を $w_1 = w_2 = b_1 = b_2 = 0$ から開始し、このロジスティックモデルを再現する係数に収束したとする。どの組に到達するかを求めよ。

    James、Witten、Hastie、Tibshirani 著『An Introduction to Statistical Learning』の演習問題 4.12 を改変。

2. **ソフトマックスのヘッセ行列。** 3.1.2.2 節と同様に、交差エントロピー損失をロジットの関数として $l(\mathbf{y}, \mathbf{o}) = g(\mathbf{o}) - \mathbf{y}^\top \mathbf{o}$（ただし $g(\mathbf{o}) = \log \sum_k \exp(o_k)$）と表す。

    1. ヘッセ行列 $\partial_{\mathbf{o}}^2 \, l(\mathbf{y}, \mathbf{o})$ を計算し、それが $\mathbf{y}$ に依存しないことを示せ。

   2. $\mathbf{e}_Y$ を、クラス Y（$\mathrm{softmax}(\mathbf{o})$ から抽出される）のワンホットベクトルとする。$\mathbf{e}_Y$ の共分散行列を計算し、前の小問で求めたヘッセ行列と一致することを示せ。
   3. g が凸関数であり、交差エントロピー損失がロジットに関して凸性を持つことを結論づけよ。

3. **Bradley–Terry モデル**。Bradley–Terry モデル (Bradley and Terry 1952) は、ロジスティックモデルを用いて選好を表現する。リンゴとオレンジのどちらかを選ぶユーザーは、2 つのスコア o_{apple} と o_{orange} のソフトマックスで与えられる確率に従って各選択肢を選ぶ。ここに、どちらも選ばないというデフォルトの選択肢を固定スコア $o_{\text{none}} = 0$ で追加する。

   1. o_{apple} と o_{orange} に同じ定数を加えたとき、選択確率が変化するかどうかを、まずデフォルト選択肢がない場合、次にデフォルト選択肢がある場合について調べよ。それぞれのモデルにおいて、観測された選択からどのスコアが一意に定まるかを結論づけよ。
   2. デフォルト選択肢がある場合とない場合で、比 $P(\text{apple})/P(\text{orange})$ を比較せよ。実際のユーザーがこの比を一定に保たないような選択状況を説明せよ。

4. **log-sum-exp**。関数 $\mathrm{lse}(a, b) = \log(\exp(a) + \exp(b))$ は 3.1.2.2 節の log-sum-exp を 2 引数にしたものであるから、その勾配は (a, b) のソフトマックスとなる。

   1. 次の不等式が成り立ち、

$$\max(a, b) < \mathrm{lse}(a, b) \le \max(a, b) + \log 2, \tag{3.12}$$

      右側の等号が $a = b$ のとき、かつそのときに限り成立することを証明し、さらに $\max(a, b)$ との差を任意に小さくできることを示せ。ヒント：一般性を失うことなく $b = 0$ かつ $a \ge 0$ と置いてよい。
   2. 任意の $\lambda > 0$ に対して

$$\max(a, b) < \lambda^{-1}\mathrm{lse}(\lambda a, \lambda b) \le \max(a, b) + \lambda^{-1}\log 2, \tag{3.13}$$

      が成り立つことを証明し、$\lambda^{-1}\mathrm{lse}(\lambda a, \lambda b) \to \max(a, b)$（$\lambda \to \infty$ のとき）を結論づけよ。
   3. 同様にして滑らかな最小値関数を構成せよ。
   4. 定義と両側の上界・下界を q 個の数に拡張し、$\max_i a_i$ との差が高々 $\log q$ であることを示せ。

5. **温度**。$P = \mathrm{softmax}(\mathbf{o})$ とし、温度 T における分布を $\mathrm{softmax}(\mathbf{o}/T)$ とする。また、別の分布 Q を $Q(i) \propto P(i)^{\alpha}$（$\alpha > 0$）により定める。

   1. 温度を 2 倍にすることに対応する α の値は何か。また、温度を半分にすることに対応する値は何か。
   2. ちょうど s 個のロジット（全 q 個のうち）が最大値をとるとする。$\mathrm{softmax}(\mathbf{o}/T)$ の $T \to 0$ における極限を求めよ。
   3. $\bar{o} = \frac{1}{q}\sum_j o_j$ をロジットの平均とする。大きな T に対して次式が成り立つことを示せ。

$$\mathrm{softmax}(\mathbf{o}/T)_i = \frac{1}{q} + \frac{o_i - \bar{o}}{qT} + O(T^{-2}). \tag{3.14}$$

   4. 温度スケーリングでは、分類器のロジットを固定したままホールドアウト集合上の交差エントロピーを最小化することで単一の T を当てはめる。この交差エントロピーが $\beta = 1/T$ の凸関数であり、したがってこの当てはめが 1 次元の凸最適化問題になることを示せ。ヒント：g の凸性（演習問題 2）を用いればよい。

6. **分離可能なデータ**。●あるパラメータ $(\mathbf{W}, \mathbf{b})$ がすべての訓練例を引き分けなしに正しく分類するとする。すなわち、どのデータ例においても正解クラスのロジットが他のすべてのロ

ジットを厳密に上回るとする。ここで、スケールされたパラメータ $(c\mathbf{W}, c\mathbf{b})$（$c > 0$）を考える。

1. すべての訓練例の交差エントロピー損失が、c の増加に伴って狭義単調減少し、0 に収束する（$c \to \infty$ のとき）ことを示せ。
2. 訓練損失が最小点を持たず、訓練損失が 0 に収束するような任意のパラメータ列は非有界なノルムを持つことを結論づけよ。
3. 2.7 節の重み減衰ペナルティを重みとバイアスの両方に適用した項 $\frac{\lambda}{2}(\|\mathbf{W}\|^2 + \|\mathbf{b}\|^2)$（$\lambda > 0$）を訓練損失に加える。このペナルティ付き損失が最小点を持つことを示せ。

7. **確率に対する二乗誤差**。交差エントロピーの代わりに、二乗損失 $\frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2$ を用いて予測確率をワンホットラベルに当てはめることも考えられる。
   1. ソフトマックスの偏導関数が $\partial \hat{y}_k / \partial o_j = \hat{y}_k(\mathbf{1}(j = k) - \hat{y}_j)$ となることを示し、これを用いてロジットに関する二乗損失の勾配を計算せよ。
   2. 2 クラスの場合について、両方の損失をマージン m の関数として表し、m に関する導関数を計算せよ。さらに $m \to -\infty$ のとき、すなわち確信を持った誤りに対して、二乗損失の導関数は 0 に収束する一方、交差エントロピーの導関数は -1 に収束することを示せ。
   3. 確信を持った誤った予測から訓練を開始するモデルにとって、この違いが何を意味するかを説明せよ。

8. [code] **Gumbel-max トリック**。●各ロジットに独立なノイズを加えて摂動させ、摂動後のスコアが最大となるクラス $y = \operatorname{argmax}_i (o_i + \gamma_i)$ を出力することでラベルが生成されるとする。ガウスノイズを用いるとプロビットモデルが得られるが、そのクラス確率は閉じた形を持たない。一方、ある別のノイズ分布を用いるとソフトマックス確率が厳密に再現される。$\gamma_1, \dots, \gamma_q$ を、累積分布関数が $P(\gamma \le z) = \exp(-\exp(-z))$ で与えられる標準ガンベル分布からの独立同一分布に従うサンプルとする。
   1. $P(o_k + \gamma_k \le z) = \exp(-\exp(o_k)\exp(-z))$ を示せ。
   2. $P\left(\operatorname{argmax}_i (o_i + \gamma_i) = k\right) = \operatorname{softmax}(\mathbf{o})_k$ を示せ。ヒント：$o_k + \gamma_k$ の値で条件付けし、置換 $u = \exp(-z)$ を用いればよい。
   3. ノイズを $\gamma_i \mapsto T\gamma_i$ とスケーリングすると、$\operatorname{softmax}(\mathbf{o}/T)$ からのサンプリングになることを結論づけよ。
   4. 結果を数値的に検証せよ。$\mathbf{o} = (1.0, 2.2, 0.3)$ および $T \in \{0.25, 1, 4\}$ に対して、10^5 個の $\operatorname{argmax}_i(o_i + T\gamma_i)$ のサンプルを生成し、ラベルの頻度がモンテカルロ誤差の範囲内で $\operatorname{softmax}(\mathbf{o}/T)$ と一致することを確認せよ。

3.2 画像分類データ

ソフトマックス回帰（softmax regression）を訓練してテストするには、弱いモデルと強いモデルを見分けられるほど十分に豊かでありながら、対話型ノートブック内で数秒のうちに訓練できるほどコンパクトな画像分類（image classification）ベンチマークが必要である。古典的な手書き数字の MNIST (LeCun, Léon Bottou, et al. 1998)（70,000 枚のグレースケール 28×28 画像、10 個の数字クラス）は 1990 年代には画期的なベンチマークであったが、今日ではごく単純なモデルでさえ 97% の正解率（accuracy）を超え、畳み込みニューラルネットワークは 99% を上回るため、アーキテクチャの改善効果を確認する余地がほとんど残っていない。その対極にある ImageNet (Deng et al. 2009)（1.2 百万枚のカラー写真、1,000 カテゴリ）では、実行ごとに数時間の GPU 計算を要する。

Fashion-MNIST (Xiao et al. 2017) はちょうど良いバランスを備えている。MNIST と同じ規模（訓練用 60,000 枚、テスト用 10,000 枚からなるサイズ 28×28 のグレースケール画像で、均等

な 10 クラス）を保ちつつ、数字を衣料品（T シャツ、ズボン、プルオーバー、ドレス、コート、サンダル、シャツ、スニーカー、バッグ、アンクルブーツ）に置き換えている。Fashion-MNIST では、線形ソフトマックス回帰の正解率は約 83% に達する一方、畳み込みネットワークは 90% を超える。本節では、以降のいくつかの章にわたって画像分類器にデータを供給する `FashionMNIST(d2l.DataModule)` クラスを構築する。

```
import tensorflow as tf
import time
```

```
d2l.use_svg_display()
```

3.2.1 データセットの読み込み

Fashion-MNIST には訓練画像 6,000 枚とテスト画像 1,000 枚が 10 個の衣類カテゴリそれぞれに含まれており（合計で 60,000 枚と 10,000 枚）、8 ビット符号なし整数として $[0, 255]$ の範囲で格納されている（0 が暗い背景を表す）。オブジェクト指向設計（2.4 節）に従い、`FashionMNIST` は `d2l.DataModule` を継承し、`batch_size` および任意の空間サイズ変更タプル `resize`（デフォルトは $(28, 28)$）を受け取る。

`tf.keras.datasets.fashion_mnist.load_data()` は、公式の訓練分割とテスト分割を `(images, labels)` の NumPy 配列として返す。

```
class FashionMNIST(d2l.DataModule):
    """The Fashion-MNIST dataset."""
    def __init__(self, batch_size=64, resize=(28, 28)):
        super().__init__()
        self.save_hyperparameters()
        self.train, self.val = tf.keras.datasets.fashion_mnist.load_data()
```

データセットをインスタンス化し、各分割のサイズを確認しよう。

```
data = FashionMNIST()
len(data.train[0]), len(data.val[0])
```

```
(60000, 10000)
```

Fashion-MNIST には 2 つの分割しか用意されていないため、訓練中は 10,000 枚のテスト分割を検証セット（validation set）として使用する（テストセットで繰り返し評価することの統計的影響については第 3.5 章で分析する）。さらに、`text_labels` を追加し、整数のクラスインデックス $0, \ldots, 9$ を人間が読める衣類名に変換できるようにしよう。

```
@d2l.add_to_class(FashionMNIST)
def text_labels(self, indices):
    """Return text labels."""
    labels = ['T-shirt', 'trouser', 'pullover', 'dress', 'coat',
              'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]
```

3.2.2 ミニバッチの読み込み

深層学習フレームワークでは、サイズ n のミニバッチに含まれる、チャネル数 c（グレースケールなら $c = 1$、RGB カラーなら $c = 3$）、高さ h、幅 w の各画像は、フレームワークに応じて**チャネルファースト**のレイアウト (n, c, h, w) または**チャネルラスト**のレイアウト (n, h, w, c) のいずれかで格納される。

JAX は画像をチャネルラストのレイアウト (n, h, w, c) で格納する。ここでは `tf.data.Dataset` を軽量なヘルパー `TensorFlowDataLoader` でラップして `.as_numpy_iterator()` と `__len__` を公開し、各エポックで NumPy のミニバッチが直接得られるようにする。

```
class TensorFlowDataLoader:
    """Expose a tf.data.Dataset as re-iterable NumPy batches."""
    def __init__(self, dataset):
        self.dataset = dataset

    def __iter__(self):
        return self.dataset.as_numpy_iterator()

    def __len__(self):
        return len(self.dataset)
```

次に、`get_dataloader(self, train)` を `FashionMNIST` に追加し、フレームワーク標準のデータローダをラップする。これにより、エポックごとに訓練分割を再シャッフルするとともに、最後の端数のミニバッチ（$60{,}000 = 937 \times 64 + 32$）も保持して、すべての画像がエポックごとに 1 回ずつ処理されるようにする。

```
@d2l.add_to_class(FashionMNIST)
def get_dataloader(self, train):
    X, y = self.train if train else self.val
    X = tf.expand_dims(X, axis=3) / 255  # Channel axis last; values in [0, 1]
    resize = lambda X, y: (tf.image.resize(X, self.resize), y)
    return TensorFlowDataLoader(tf.data.Dataset.from_tensor_slices(
        (X, tf.cast(y, tf.int32))).shuffle(len(y) if train else 1).batch(
            self.batch_size).map(resize))
```

1 つのミニバッチ `(X, y)` を `train_dataloader()` から取得し、単一画像のレイアウトならびにミニバッチ全体のテンソルの形状と dtype を確認しよう。

```
X, y = next(iter(data.train_dataloader()))
X[0].shape  # Channel-last: (height, width, channels)
```

```
(28, 28, 1)
```

データ例を束ねてミニバッチにすると、画像の先頭にバッチ軸が追加される。一方、ラベルは画像ごとに 1 つの要素を持つ 1 次元ベクトルとして得られる。

```
print(X.shape, X.dtype, y.shape, y.dtype)
```

```
(64, 28, 28, 1) float32 (64,) int32
```

ミニバッチあたり $64 \times 28 \times 28 \times 4 \approx 200$ KB（$\sim$ 190 MB で 60,000 枚の float32 訓練画像すべてに相当）と小さいため、データセット全体が RAM に余裕をもって収まる。それでも、データローダ単体でベンチマークをとること（モデルを接続せずに 1 パス全体の時間を `train_dataloader()` について計測すること）は極めて重要な習慣である。どの訓練エポックも、そのデータパイプラインより速く実行することはできないからである。

```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

```
'0.55 sec'
```

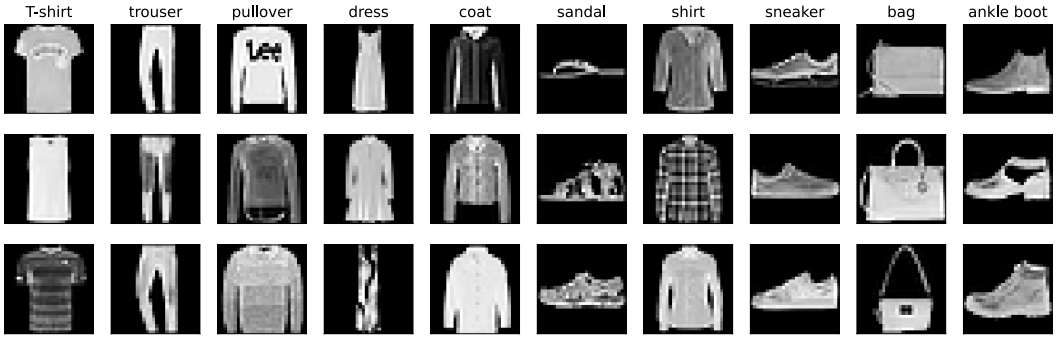
3.2.3 画像の確認

どのようなモデルを訓練する前にも、必ずラベル付き画像のサンプルを目視で確認すべきである。ラベルのずれ、輝度の反転、あるいは誤ったリシェイプによる画像の崩れは、実行時エラーを起こさないまま訓練を台無しにするからである。そこで、単一チャネルの軸を除去し、テキストラベルをタイトルに添えたグレースケール画像のグリッドを描画する `visualize` メソッドを `FashionMNIST` に追加する。

```
@d2l.add_to_class(FashionMNIST)
def visualize(self, batch, nrows=1, ncols=8, labels=None):
    """Show the images of a batch in grayscale, titled by their labels."""
    X, y = batch
    if labels is None:
        labels = self.text_labels(y)
    X = d2l.reshape(X, (-1, *self.resize))  # Drop the channel axis
    d2l.show_images(X, nrows, ncols, titles=labels, cmap='gray')
```

検証ローダが常に決定論的な順序で画像を返すことを利用して、各クラスの最初の 3 枚の検証画像をカテゴリごとに 1 列ずつ表示してみよう。

```
X, y = next(iter(data.val_dataloader()))
labels = d2l.numpy(y)
idx = [int(np.flatnonzero(labels == c)[r]) for r in range(3) for c in range(10)]
titles = data.text_labels(range(10)) + [''] * 20
data.visualize([d2l.stack([X[i] for i in idx]), None], nrows=3, ncols=10,
               labels=titles)
```



ざっと目を通すだけでも、線形分類器が直面する難しさが予見できる。ズボンとバッグは紛れもないシルエットを持ち、サンダル・スニーカー・アンクルブーツは明確な履物のクラスタを形成している。一方、4 つの上半身の衣類（T シャツ、プルオーバー、コート、シャツ）はほぼ同じ箱型の胴体輪郭を共有しており、28×28 の解像度ではかろうじて判別できる程度の襟、袖、ボタンのわずかな違いしか見られない。

3.2.4 まとめ

Fashion-MNIST は、訓練用 60,000 枚およびテスト用 10,000 枚のグレースケール 28×28 画像を均等な 10 個の衣類カテゴリにわたって提供しており、線形モデル（$\sim 83\%$）と畳み込みネットワーク（$> 90\%$）の性能差を見分けられるほど豊かでありながら、数秒で訓練できるほど小さい。本節で構築した `FashionMNIST(d2l.DataModule)` パイプラインは、生の `uint8` ピクセルを `float32` で $[0, 1]$ の範囲に再スケーリングし、任意の空間リサイズに対応するとともに、再シャッフルされたミニバッチと整数ラベルを各フレームワーク標準のチャネルレイアウトで供給する。ローダ単体のベンチマークはエポック時間の下界を与え、ラベル付き画像の一覧を目視確認することで、訓練開始前に表面化しない形状やラベルの不整合を発見できる。

3.2.5 演習問題

1. [code] **クラス平均**。訓練セット全体で各クラスの平均画像を計算し、10 個の平均画像を画像のグリッドとして表示せよ。
    1. クラス平均のすべてのペア間のユークリッド距離を計算し、その距離を 10×10 の表として示せ。どのペアが最も近く、他のすべてのクラスから最も遠いのはどのクラスか。
    2. この表から、線形分類器が最も混同しやすいクラスと、ほとんど混同しないクラスを予測せよ。この予測は控えておけ。3.3 節でその答えを計算する。
    3. 平均画像は近いにもかかわらず人間なら容易に見分けられるクラスのペアを挙げ、どのピクセルがその違いを担っているかを述べよ。
2. [code] **ローダのスループット**。訓練セットに対する 1 パスの時間は、ミニバッチのサイズと、パイプラインのどの程度が並列実行されるかに依存する。
    1. 1 パス全体の時間を `train_dataloader` について計測し（`FashionMNIST` は `batch_size` を 1、16、64、256、1024 として構築する）、毎秒の画像枚数によるスループットを `batch_size` に対してプロットせよ。さらに、バッチごとのオーバーヘッドと画像ごとの処理量の観点から曲線の形状を説明せよ。
    2. `tf.data` パイプライン（`get_dataloader` 内）は、`num_workers` という `DataModule` の属性を使用せず、ステージごとに並列性を設定する。まず実装されたままの状態で 1 パスの時間を計測し、次に `num_parallel_calls=tf.data.AUTOTUNE` を `map` の呼び出しに指定し末尾に `.prefetch(tf.data.AUTOTUNE)` を追加した状態で計測せよ。どのような条件

のもとで並列性を増やしても高速化に寄与しなくなるかを述べよ。

3. [code] **アップサンプリングと線形モデル。**●28×28 から 32×32 へのリサイズを、線形変換 $\mathbf{x}' = \mathbf{A}\mathbf{x}$（元の 784 ピクセルからリサイズ後の 1024 ピクセルへの変換であり、$\mathbf{A} \in \mathbb{R}^{1024 \times 784}$ の各列は線形独立）として扱え。
   1. $\mathbf{x}'$ 上のソフトマックス回帰が表現できる分類器の集合は、$\mathbf{x}$ 上のソフトマックス回帰が表現できる分類器の集合と完全に一致することを示せ。
   2. 3.3 節を終えた後、`SoftmaxRegressionScratch` を `FashionMNIST()` のデータと `FashionMNIST(resize=(32, 32))` のデータでそれぞれ訓練せよ（`num_inputs` は画像あたりのピクセル数に設定する）。検証正解率を比較し、前問の結果を踏まえて差異があればその理由を説明せよ。

3.3 ソフトマックス回帰の実装

第 2 章で線形回帰をスクラッチと簡潔な方法の両方で実装したのと同様に、本節では Fashion-MNIST 上でソフトマックス回帰（softmax regression）をこの 2 通りの方法で構築する。まずは生のテンソルから出発し、行ごとのソフトマックス、ベクトル化した交差エントロピー損失（cross-entropy loss）、`SoftmaxRegressionScratch` のパラメータ（$\mathbf{W} \in \mathbb{R}^{784 \times 10}, \mathbf{b} \in \mathbb{R}^{10}$）を、再利用可能な基底クラス `Classifier(d2l.Module)` とあわせて実装する。これを用いて検証正解率が約 83% に達するまで訓練し、10×10 の混同行列（confusion matrix）と 28×28 の重みテンプレートを通して線形モデルが何を学習したかを調べる。続く 3.3.5 節では、手動のテンソル管理と確率のクランプ処理を組み込みの線形層および生のロジットから直接計算するシフト付き **log-sum-exp** 交差エントロピー損失に置き換え、浮動小数点のオーバーフローとアンダーフローを一挙に解消する。

3.3.1 ソフトマックス

3.3.1.1 各行の正規化

n 個の例からなるミニバッチに対して、ロジットは $n \times q$ 行列 $\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}$ をなす（3.1.1.4 節）。ソフトマックスは各行を独立に正規化する。

$$\text{softmax}(\mathbf{O})_{ij} = \frac{\exp(O_{ij})}{\sum_{k=1}^{q} \exp(O_{ik})}. \tag{3.15}$$

コードでは、`O` の指数をとり、列方向（`axis=1`）に `keepdims=True` で和をとることで行和の形状を `(n, 1)` に保ち、ブロードキャストによって除算する。

```
def softmax(O):
    O_exp = d2l.exp(O)
    return O_exp / d2l.reduce_sum(O_exp, 1, keepdims=True)  # Broadcast
```

2 つの行で `softmax` の動作を確認してみよう。猫・鶏・犬のロジット $(1.0, 2.2, 0.3)$（3.1 節）と、同じ 0 が 3 つ並ぶ行である。

```
O = d2l.tensor([[1.0, 2.2, 0.3], [0.0, 0.0, 0.0]])
Y_hat = softmax(O)
Y_hat, d2l.reduce_sum(Y_hat, 1)
```

```
(Array([[0.20761092, 0.68929255, 0.10309654],
        [0.33333334, 0.33333334, 0.33333334]], dtype=float32),
 Array([1., 1.], dtype=float32))
```

3.3.1.2 オーバーフロー

なぜこの 3 行の `softmax` の式を本番環境で決して使ってはならないのだろうか。32 ビット浮動小数点（`float32`）では表現可能な数値の上限が 3.4×10^{38} 付近にあるため、$\exp(o)$ は $o > 88.7$ になった途端に `inf` へオーバーフローするからである。`softmax` にロジット $(1000, 0, 0)$ を入力すると ∞/∞ が計算されて `NaN` になるが、真の確率分布は機械精度の範囲で $(1, 0, 0)$ である。

3.1 節で証明したように、ソフトマックスは行内のすべてのロジットを同じ定数だけシフトしても不変である。指数をとる前に行の最大値 $o_{\max} = \max_k o_k$ を引けば、すべての指数 $o_j - o_{\max}$ が ≤ 0（最大値が 0）となるため、オーバーフローは起こり得ない。どのフレームワークの組み込みソフトマックスも、内部でこのシフトを適用している。

```
O = d2l.tensor([[1000.0, 0.0, 0.0]])
O_max = O.max(axis=1, keepdims=True)
softmax(O), softmax(O - O_max), jax.nn.softmax(O, axis=1)
```

```
(Array([[nan,  0.,  0.]], dtype=float32),
 Array([[1., 0., 0.]], dtype=float32),
 Array([[1., 0., 0.]], dtype=float32))
```

以下の Fashion-MNIST のスクラッチ実行では、ピクセル値が $[0, 1]$ に収まり重みも小さいままであるため、訓練中に `softmax` がオーバーフローすることはない。しかし、シフトだけでは非常に小さな負のロジットに対して**アンダーフロー**が生じる危険が残る（float32 では $\exp(-200) = 0.0$ となり、$\log 0$ が $-\infty$ になる）。後述の 3.3.5 節では、対数空間でソフトマックスと対数を融合させることで、この両方の危険を恒久的に取り除く。

3.3.2 損失と正解率

3.3.2.1 ロジットの 2 つの用途

勾配降下法は滑らかな交差エントロピー損失を最適化するが、人間は分類器を**正解率**（accuracy）、すなわち最高スコアのクラスが正解ラベル $y^{(i)}$ と一致する例の割合によって評価する。

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\Big(\operatorname*{argmax}_j o_j^{(i)} = y^{(i)}\Big). \tag{3.16}$$

どちらの指標もまったく同じロジットベクトルから計算され（Figure 3.5）、検証時には両方を追跡することが不可欠である。モデルが一部の誤分類に対して過剰に確信を持つようになると、検証正解率が上昇し続けているにもかかわらず、検証交差エントロピー損失が増加することがあるからである。

3.3.2.2 `Classifier` クラス

この 2 指標による検証の振る舞いを、本書におけるすべての分類モデルの基底クラスとなる `Classifier(d2l.Module)` にまとめる。まずは各検証ミニバッチに対して `validation_step` が `'loss'` と `'acc'` の両方を返すように定義する。

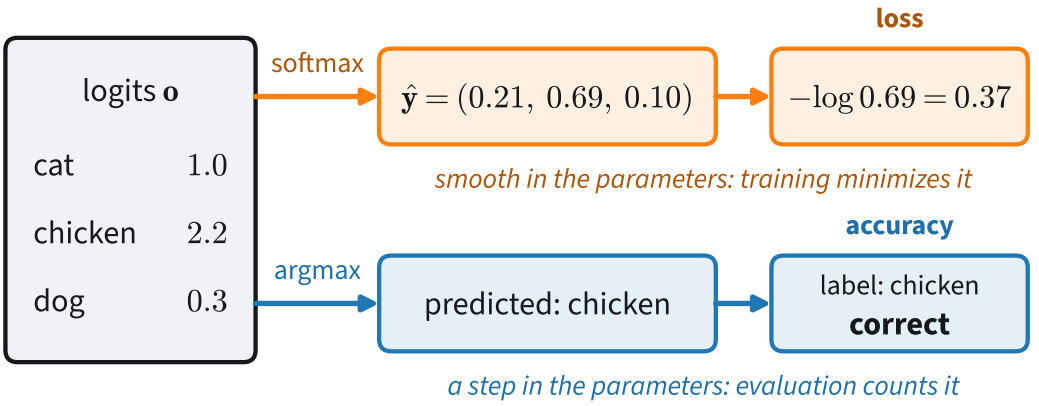


Figure 3.5: 訓練時（上）はロジットをソフトマックスに通して滑らかな交差エントロピー損失へ写像し、評価時（下）は $\mathrm{argmax}_j o_j$ をラベル y と比較して離散的な正解率を計算する。

```
class Classifier(d2l.Module):
    """The base class of classification models."""
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        return {'loss': self.loss(Y_hat, batch[-1]),
                'acc': self.accuracy(Y_hat, batch[-1])}
```

次に、`accuracy(self, Y_hat, Y, averaged=True)` を `Classifier` に追加する。このメソッドは `Y_hat` を 1 例あたり 1 行のクラススコアへと平坦化し、`argmax` を `axis=1` に沿って適用して予測クラスのインデックスを求め、`Y.dtype` にキャストしたうえで、平均値または例ごとの float32 等値マスク `preds == Y` を返す。

```
@d2l.add_to_class(Classifier)
def accuracy(self, Y_hat, Y, averaged=True):
    """Compute the fraction of correct predictions."""
    Y_hat = d2l.reshape(Y_hat, (-1, Y_hat.shape[-1]))
    preds = d2l.astype(d2l.argmax(Y_hat, axis=1), Y.dtype)
    compare = d2l.astype(preds == d2l.reshape(Y, (-1,)), d2l.float32)
    return d2l.reduce_mean(compare) if averaged else compare
```

最後に、すべての分類器がそのまま使えるオプティマイザを継承できるよう、学習率 `self.lr` の通常のミニバッチ SGD をデフォルトの `configure_optimizers` メソッドとして `d2l.Module` に組み込む。

```
@d2l.add_to_class(d2l.Module)
def configure_optimizers(self):
    return optax.sgd(self.lr)
```

3.3.3 モデルと損失

3.3.3.1 パラメータと順伝播

28×28 の各 Fashion-MNIST 画像を平坦化すると、入力特徴量は $d = 784$ 個、クラス数は $q = 10$ 個となるため、`SoftmaxRegressionScratch` では $\mathbf{W} \in \mathbb{R}^{784\times 10}$（$\mathcal{N}(0, 0.01^2)$ から初期化）と $\mathbf{b} \in \mathbb{R}^{10}$（0 で初期化）の計 7,850 個のパラメータを確保する。ここで簡単な妥当性確認をしておこう。ゼロに近い初期重みからはゼロに近いロジットが生じるため、未訓練のモデルは一様な予測（$\hat{y}_j \approx 0.1$）に近い状態から始まり、初期の交差エントロピー損失は $-\log(0.1) = \log 10 \approx 2.30$ 付近となる。

```
class SoftmaxRegressionScratch(Classifier):
    """Softmax regression implemented from scratch."""
    def __init__(self, num_inputs, num_outputs, lr, std=0.01, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.W = nnx.Param(
            rngs.params.normal((num_inputs, num_outputs)) * std)
        self.b = nnx.Param(jnp.zeros(num_outputs))
```

`forward(self, X)` では、ミニバッチ内の各画像を長さ `self.num_inputs`（784）の行ベクトルへと `d2l.reshape(X, (-1, self.num_inputs))` によって平坦化し、アフィン変換によるロジット $\mathbf{XW} + \mathbf{b}$ を `softmax` に通す。

```
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
    X = d2l.reshape(X, (-1, self.num_inputs))  # One flattened image per row
    return softmax(d2l.matmul(X, self.W) + self.b)
```

3.3.3.2 交差エントロピー損失

3.1.2 節で見たように、$l(\mathbf{y}, \hat{\mathbf{y}}) = -\log \hat{y}_y$ であった。ワンホット行列を実体化して内積をとる代わりに、ベクトル化された 1 回のインデックス参照によって、正解クラスの確率 $\hat{y}^{(i)}_{y^{(i)}}$ をすべての行 i について抜き出す。

```
y = d2l.tensor([0, 2])
y_hat = d2l.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
```

```
Array([0.1, 0.5], dtype=float32)
```

これにより、1 つ目の例から 0.1、2 つ目の例から 0.5 が抽出される。これを `cross_entropy` にまとめ、ミニバッチ全体で負の対数の平均をとる。対数をとる前に、防御策として下限を 10^{-12} にクランプしている点に注目してほしい。こうすることで、アンダーフローして 0 になった確率が $-\infty$ へと発散するのを防ぎ、1 例あたりの損失の上限を 27.6 に抑えられる。ただし、クランプが作動するとその箇所の勾配も 0 になってしまう。本節の後半（3.3.5 節）では、この場しのぎの処置を、ロジットから直接計算する融合型の損失に置き換える。

```
def cross_entropy(y_hat, y):
    # Tiny clip to keep log finite when softmax outputs underflow to 0.
    p = jnp.clip(y_hat[jnp.arange(len(y_hat)), y], min=1e-12)
    return -d2l.reduce_mean(d2l.log(p))

cross_entropy(y_hat, y)
```

```
Array(1.4978662, dtype=float32)
```

結果は $-\log 0.1 \approx 2.30$ と $-\log 0.5 \approx 0.69$ の平均、すなわち約 1.50 となる。これをモデルの loss メソッドとして追加しよう。

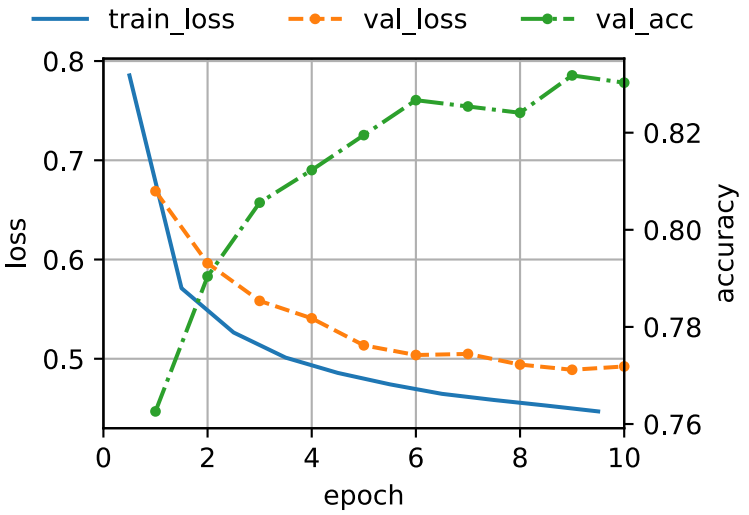
```
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)
```

3.3.4 訓練と評価

3.3.4.1 訓練の実行

SoftmaxRegressionScratch を Fashion-MNIST 上で 10 エポック、ミニバッチサイズ 256、学習率 $\eta = 0.1$ で訓練してみよう。

```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



訓練済みモデルを任意のタイミングでデータセット分割全体に対して評価できるよう、補助メソッド evaluate(self, model, data, train=False) を d2l.Trainer に追加する。このメソッドは self.val_dataloader に検証ローダまたは訓練ローダを紐づけて self.validate() を実行する。

```
@d2l.add_to_class(d2l.Trainer)
def evaluate(self, model, data, train=False):
    """Metrics of `model` on the validation data of `data` (on its training
    data if `train`), averaged over examples."""
    self.train_dataloader = None
    self.val_dataloader = (data.train_dataloader() if train
                           else data.val_dataloader())
    self.prepare_model(model)
    return self.validate()
```

全 10,000 枚の検証画像にわたって平均したこれらの指標は、`trainer.history['val_loss'][-1]` および `trainer.history['val_acc'][-1]` に記録された最終値と一致する。

```
trainer.evaluate(model, data)
```

```
{'loss': 0.492251513671875, 'acc': 0.8303}
```

この分類器は検証画像の約 83% を正しく分類できている。`train=True` を渡せば、まったく同じ重みを全 $60{,}000$ 枚の訓練画像に対して評価できる。

```
trainer.evaluate(model, data, train=True)
```

```
{'loss': 0.45724899088541665, 'acc': 0.8453333333333334}
```

10 エポック後、検証正解率は 83% 付近（乱数シードにより $\pm 1\%$ 変動）に落ち着き、訓練正解率は 84% 付近となる。この 1% というごく小さな汎化ギャップ（generalization gap）は、線形分類器の表現能力の限界を反映している。$7{,}850$ 個のパラメータでは $60{,}000$ 枚の画像を丸暗記することはできない（第 3.5 章）。

3.3.4.2 誤分類された画像

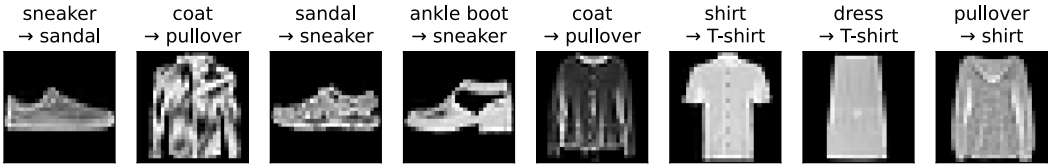
モデルが**どのように**誤るかを確認するため、最初の検証ミニバッチ（256 枚の画像）を取り出して誤分類の数を数えてみよう。

```
X, y = next(iter(data.val_dataloader()))
preds = d2l.astype(d2l.argmax(model(X), axis=1), y.dtype)
wrong = preds != y
print(f'{int(d2l.reduce_sum(d2l.astype(wrong, d2l.int32)))} of {len(y)} wrong')
```

```
35 of 256 wrong
```

誤分類された最初の 8 枚の画像を、正解ラベルをモデルの予測の上に添えて（`true` → `pred`）表示してみよう。

```
labels = [f'{true}\n→ {pred}' for true, pred in zip(
    data.text_labels(y[wrong]), data.text_labels(preds[wrong]))]
data.visualize([X[wrong], y[wrong]], labels=labels)
```



3.3.4.3 混同行列

全 10,000 枚の検証画像にわたる誤りのパターンを要約するため、10×10 の**混同行列** $\mathbf{N} \in \mathbb{R}^{q \times q}$ を集計する。ここでは、行が予測クラス $\hat{c}^{(m)} = \operatorname{argmax}_j o_j^{(m)}$ を、列が正解クラス $y^{(m)}$ を表す。

$$N_{ij} = \sum_{m=1}^{n} \mathbf{1}(\hat{c}^{(m)} = i)\, \mathbf{1}(y^{(m)} = j). \tag{3.17}$$

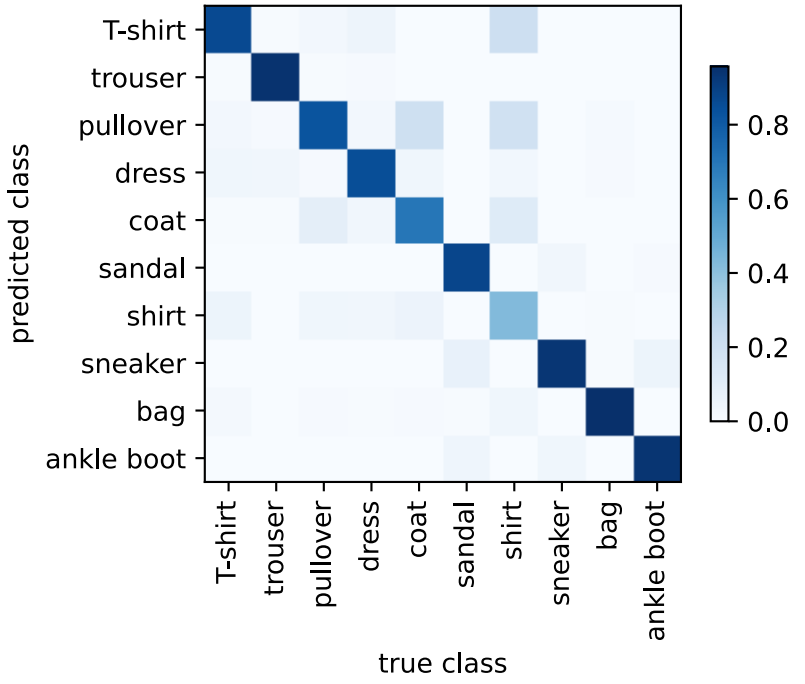
（正解クラスを列、予測クラスを行に配置しておくと、列正規化した条件付き確率の積が $\mu = \mathbf{Cp}$ とすっきり書け、3.6 節でラベルシフト（label shift）を扱う際に便利である。）ミニバッチの予測クラスと正解クラスを表すワンホット行列 $\mathbf{P}, \mathbf{Y} \in \{0,1\}^{n \times q}$ を用いれば、積 $\mathbf{P}^\top \mathbf{Y}$ によってミニバッチあたり 1 回の行列積ですべての 100 マス分のカウントを累積できる。

```
def one_hot(y, q):
    """Rows of the q x q identity matrix, one per class index in y."""
    return d2l.astype(d2l.reshape(y, (-1, 1)) == d2l.arange(q, dtype=y.dtype),
                      d2l.float32)

N = d2l.zeros((10, 10))
for X, y in data.val_dataloader():
    P = one_hot(d2l.argmax(model(X), axis=1), 10)
    N += d2l.matmul(d2l.transpose(P), one_hot(y, 10))
int(d2l.reduce_sum(N))
```

```
10000
```

これで全 10,000 枚の検証画像が集計された。生のカウント数はクラスの出現頻度に依存するため、$\mathbf{N}$ の各列をその列和で正規化して $C_{ij} = N_{ij} / \sum_{i'} N_{i'j}$ を求める。これは条件付き確率 $P(\hat{c} = i \mid y = j)$ の経験推定値である。各列は正解クラス j の例がどの予測クラスに振り分けられたかを示し、対角成分はクラスごとの認識率を与える。$\mathbf{C}$ を `d2l.show_heatmaps` でヒートマップとして描画してみよう。

```
C = N / d2l.reduce_sum(N, 0, keepdims=True)  # Each column sums to 1
names = data.text_labels(range(10))
d2l.show_heatmaps(d2l.reshape(C, (1, 1, 10, 10)), xlabel='true class',
                  ylabel='predicted class', figsize=(4, 4), cmap='Blues',
                  xticklabels=names, yticklabels=names)
```



最も頻繁に生じている混同を浮き彫りにするため、非対角成分の上位 5 件も出力してみよう。

```
mistakes = d2l.numpy(N) * (1 - np.eye(10))  # Zero the diagonal
for k in np.argsort(mistakes, axis=None)[::-1][:5]:
    i, j = divmod(int(k), 10)
    print(f'{names[j]} taken for {names[i]}: {int(mistakes[i, j])}')
```

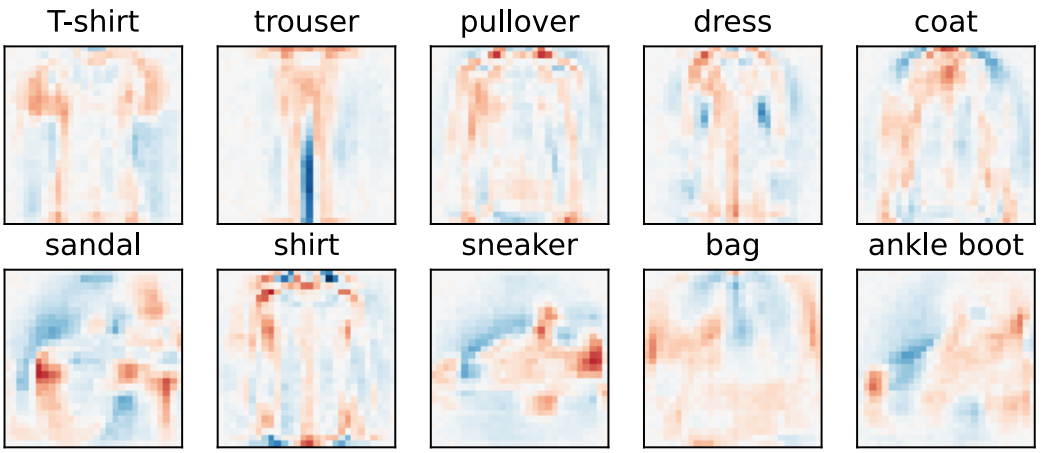
```
shirt taken for T-shirt: 203
coat taken for pullover: 201
shirt taken for pullover: 189
shirt taken for coat: 117
pullover taken for coat: 91
```

この混同行列は、3.2 節での目視確認から予想された結果を裏づけている。ズボン、バッグ、履物は $\sim 90\%$ 以上の割合で正しく認識される一方、非対角成分の上位 5 件の混同はすべて、見た目のよく似た上半身のトップス 4 種（T シャツ、プルオーバー、コート、シャツ）の間で起きている。なかでもシャツはシルエットが他の 3 種のちょうど中間に位置するため、半分程度以下しか正しく認識されない。

3.3.4.4 学習されたテンプレート

各ロジット $o_j = \mathbf{w}_j^\top \mathbf{x} + b_j$ は、平坦化された画像 $\mathbf{x}$ と $\mathbf{W}$ の第 j 列との内積であるため、784 要素の各列 $\mathbf{w}_j$ を 28×28 の形状に戻せば、分類器全体を 10 枚の視覚的な**テンプレート**（赤は正の重み、青は負の重み）として観察できる。

```
W = d2l.numpy(model.W[...])
templates = W.T.reshape(10, 28, 28)  # Column j of W as an image
v = np.abs(templates).max()
d2l.show_images(templates, 2, 5, titles=names, cmap='RdBu_r', vmin=-v, vmax=v);
```



各テンプレートには対応する衣類の淡いシルエットがはっきりと浮かび上がっている。ドレスに対して負の重みを与えるズボンの両脚の間の青い帯や、主に襟元と袖周りしか違わない 4 種のトップスのよく似た胴体形状に注目してほしい。線形層はクラスごとに固定された大域的なドット積を 1 回とることしかできないため、襟やボタンのような微細な局所的特徴ではシルエット全体の重なりを克服できない。$\sim 83\%$ の壁を超えるには、非線形な隠れ層（第 4 章）や空間的な畳み込み（第 6 章）が必要となる。

3.3.5 簡潔な実装

ここからは、深層学習フレームワークを用いることで、素朴な `softmax` やクランプ付きの `cross_entropy` が抱えていたオーバーフローとアンダーフローの危険を恒久的に解消しつつ、ソフトマックス回帰をいかに簡潔に記述できるかを見ていこう。

3.3.5.1 モデルの定義

`SoftmaxRegression(d2l.Classifier)` では、平坦化ステップと出力数 10 の組み込み全結合層（fully connected layer）1 つによってロジット $\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}$ を計算する（スクラッチ版のベースラインと揃えるため、重みは $\mathcal{N}(0, 0.01^2)$、バイアスは 0 で初期化する）。ここで決定的に重要なのは、**`forward` がソフトマックスを適用せず、生のロジット $\mathbf{O}$ を直接返す**ことである。

```
class SoftmaxRegression(d2l.Classifier):
    """The softmax regression model."""
    def __init__(self, num_outputs, lr, num_inputs=784, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.net = nnx.Linear(num_inputs, num_outputs,
                              kernel_init=nnx.initializers.normal(0.01),
                              rngs=rngs)
```

```
    def forward(self, X):
        X = X.reshape((X.shape[0], -1))  # Flatten
        return self.net(X)
```

3.3.5.2 安定な交差エントロピー損失

なぜ SoftmaxRegression.forward は確率を返さず、ロジットの段階で止めるのだろうか。まず $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o})$ を計算し、次に $-\log \hat{y}_y$ を float32 で計算するときに生じる浮動小数点の 2 つの危険を思い出そう。

1. **オーバーフロー：**$\exp(o)$ は $o > 88.7$ で inf へオーバーフローする。すべてのロジットから $o_{\max} = \max_k o_k$ を引けば、シフト後のすべての指数 $o_k - o_{\max}$ が ≤ 0 となるため、オーバーフローは解消される。
2. **アンダーフロー：**$o_{\max}$ によるシフト後であっても、ロジットが 104 を超えて $o_{\max}$ を下回るクラス（たとえばコード上でラベル 1 にあたる $\mathbf{o} = (0, -200, 0)$ の第 2 クラス）では、float32 において $\exp(o_y - o_{\max})$ が厳密な 0.0 へとアンダーフローする。すると $\log(0.0)$ をとることで -inf が生じる（あるいはスクラッチ版の 1e-12 クランプを使った場合は損失が 27.6 で頭打ちになり、**勾配が 0 になって**、自信満々に犯した大間違いからモデルが学習できなくなる）。

中間の確率を一切実体化しないことで、オーバーフローとアンダーフローの両方を回避できる。**LogSumExp トリック**によって対数とシフト付きソフトマックスを代数的に結合すると、除算は減算へと変わる。

$$l(y, \mathbf{o}) = \log \sum_{k=1}^{q} \exp(o_k) - o_y = o_{\max} + \log \sum_{k=1}^{q} \exp(o_k - o_{\max}) - o_y. \tag{3.18}$$

シフト後のすべての指数 $o_k - o_{\max}$ は ≤ 0 であり、かつ少なくとも 1 つは 0 に等しいため、対数の中の和は $[1, q]$ に、その対数は $[0, \log q]$ に収まる。オーバーフローも $\log 0$ も起こり得ず、自動微分（automatic differentiation）によって有限の勾配 $\mathrm{softmax}(\mathbf{o}) - \mathbf{y}$ が得られる。

どのフレームワークも、この融合型 log-sum-exp 交差エントロピーを提供している。これを d2l.Classifier.loss に一度組み込んでおけば、本書の残りのすべての分類器がそのまま継承できる。

```
@d2l.add_to_class(d2l.Classifier)
def loss(self, Y_hat, Y, averaged=True):
    Y_hat = d2l.reshape(Y_hat, (-1, Y_hat.shape[-1]))
    Y = d2l.reshape(Y, (-1,))
    fn = optax.softmax_cross_entropy_with_integer_labels
    return fn(Y_hat, Y).mean() if averaged else fn(Y_hat, Y)
```

SoftmaxRegression.loss を極端なロジット O = [[1000.0, 0.0, 0.0], [1000.0, 0.0, 0.0]] と正解ラベル y = [0, 1] でテストすると、融合型 LogSumExp 実装がオーバーフローもアンダーフローも起こすことなく、例ごとの正確な損失 [0., 1000.] を返すことが確認できる。

```
O = d2l.tensor([[1000.0, 0.0, 0.0], [1000.0, 0.0, 0.0]])
y = d2l.tensor([0, 1])
```

```
SoftmaxRegression(num_outputs=3, lr=0.1).loss(O, y, averaged=False)
```

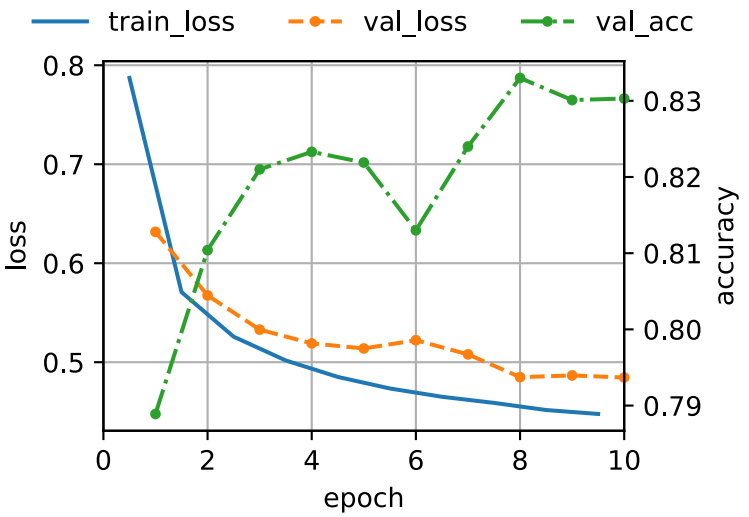
```
Array([   0., 1000.], dtype=float32)
```

（典型的な落とし穴に注意しよう。`Classifier.loss` は生のロジットを受け取って内部で log-softmax を適用するため、誤って `forward` の末尾にソフトマックスを置くとソフトマックスが **2 回**適用され、訓練損失が正の下限値より下がらなくなってしまう。以下の演習問題 4 を参照のこと。）

3.3.5.3 訓練

`SoftmaxRegression` の `d2l.Trainer` による訓練は 4 行で記述でき、盤石な数値安定性（numerical stability）のもとで $\sim 83\%$ の検証正解率を再現する。

```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegression(num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



3.3.6 まとめ

ソフトマックス回帰をスクラッチで実装する過程で、再利用可能な基底クラス `Classifier` を導入した。このクラスは訓練時の滑らかな交差エントロピーの最適化と検証時の離散的な argmax 正解率を組み合わせ、Fashion-MNIST 上で 10 エポックのうちに約 83% の検証正解率に到達する。ワンホット行列の積 $\mathbf{P}^\top\mathbf{Y}$ を累積して 10×10 の混同行列を構築すると、学習された 28×28 の重みテンプレートが大きく重なり合う上半身の衣類 4 種に線形モデルの誤りが集中していることがわかる。`softmax` と `log` を別々に計算すると大きな正のロジットでオーバーフローし、大きな負のロジットで $\log 0$ へとアンダーフローするため、簡潔な実装では `forward` から生のロジットを返し、シフト付き log-sum-exp 交差エントロピー $o_{\max} + \log \sum_k \exp(o_k - o_{\max}) - o_y$ を評価することで、あらゆる大きさのロジットに対して有限の値と損なわれない勾配を保証する。

3.3.7 演習問題

1. [code] **ワンホットラベルからの交差エントロピー**。本節の `cross_entropy` はインデックス参照によって正解クラスの確率を選択している。定義 $-\sum_j y_j \log \hat{y}_j$ に文字通り従い、`y` をワンホット行列に変換して行ごとの内積をとる別実装を作成せよ。
   1. これを `loss` として `SoftmaxRegressionScratch` に用い、訓練結果が変わらないことを確認せよ。
   2. 256 行、$q \in \{10, 10^3, 10^5\}$ 列のランダムな確率行列に対して両バージョンの実行時間を計測し、各バージョンが確保するメモリ量が q に応じてどのように増大するか述べよ。
   3. 内積による形式は適用できるが、インデックス参照による形式は適用できない設定を 1 つ挙げて説明せよ。
2. **大規模な語彙**。ソフトマックス回帰によって、次元 $d = 1024$ の特徴ベクトルから次の単語を予測するとする。語彙サイズは $V = 10^6$ 語であり、各ミニバッチは $n = 256$ 個の例を含む。`forward` と `cross_entropy` に含まれる演算のうち、コストが V に応じて増大するものを列挙せよ。また、行列積の積和演算回数、`W` が占める float32 メモリ量、および $n \times V$ の確率行列が占める float32 メモリ量を計算し、それぞれの数値を本節の Fashion-MNIST モデルと比較せよ。
3. [code] **テンプレートとクラス平均**。テンプレートとは、ミニバッチ SGD が観測したピクセルから作り上げたものである。
   1. `SoftmaxRegressionScratch` を `std=0` で再訓練し、すべての重みが 0 から始まり、すべてのクラスが同じロジットから始まるようにせよ。それでも訓練が問題なく進む理由を説明し、正解率とテンプレートを本文の結果と比較せよ。
   2. ゼロ重みから SGD を 1 ステップ進めた後、$\mathbf{W}$ の第 j 列が最初のミニバッチ上の $\sum_i \left(y_j^{(i)} - \frac{1}{10}\right) \mathbf{x}^{(i)}$、すなわちクラス j の画像の総和から全画像の総和の 10 分の 1 を引いたものに比例することを示せ。
   3. 訓練セット全体で各クラスの平均画像を計算し、目視およびピクセル間の相関によってそのクラスの訓練済みテンプレートと比較せよ。両者はどこで一致し、テンプレートはどこで平均から乖離しているか。また、単にクラス平均をコピーしただけのテンプレートでは、トップスの混同がさらに悪化する理由を説明せよ。
4. [code] **二重のソフトマックス**。本節の `loss` はロジットを入力として想定している。モデルの `forward` の末尾に誤ってソフトマックスを入れてしまったとしよう。このとき、損失関数はロジットの代わりに確率ベクトル $\mathbf{p}$ を受け取り、$-\log \mathrm{softmax}(\mathbf{p})_y$ を評価することになる。
   1. すべての確率ベクトル $\mathbf{p}$ とラベル y にわたってこの損失がとり得る最小値を、クラス数 q の関数として求めよ。また、$q = 10$ のときの値を計算せよ。
   2. 本節の前半で定義したスクラッチ版の `softmax`（`d2l` パッケージには含まれない）を用い、`SoftmaxRegression` を継承して `forward` がロジットにこの `softmax` を適用するようにせよ。これを `lr=0.1`、`d2l.FashionMNIST(batch_size=256)`、`d2l.Trainer(max_epochs=10)` で訓練せよ。その訓練損失と検証正解率を `SoftmaxRegression` と比較し、最終的な訓練損失を前問で求めた値と関連づけて説明せよ。
5. [code] **安定なソフトプラス**。ロジット $(x, 0)$ を持つ 2 クラスの場合、損失の第 1 項は $\log(1 + e^x)$、すなわち x の**ソフトプラス**となる。
   1. $\log(1 + e^x)$ と $\max(x, 0)$ を区間 $[-5, 5]$ 上でプロットせよ。前者が後者を高々 $\log 2$ だけ上回る（最大差をとるのは $x = 0$ のとき）ことを示し、log-sum-exp が滑らかな最大値と呼ばれる理由を 1 文で説明せよ。

   2. $\log(1+e^x) = \max(x,0) + \log(1+e^{-|x|})$ が成り立つことを示せ。
   3. 単精度において、素朴な式 `log(1 + exp(x))`、変形後の式、および `log1p` で `log(1 + ...)` を置き換えた変形後の式を $x \in \{-100, -20, -5, 0, 5, 20, 100\}$ に対して評価せよ。各値を倍精度の基準値と比較し、それぞれのバージョンがどこでオーバーフローし、どこで相対精度を完全に失うかを報告せよ。
   4. $x = -20$ における素朴な式の導関数を自動微分によって計算せよ。式の値自体は不正確であるにもかかわらず導関数が正確である理由と、これが訓練および高い確信度で分類された例の損失報告に対して何を意味するかを説明せよ。

6. [code] **生き残る勾配**。正解ラベル 1 に対するロジット $(0, -200, 0)$ では、クラス 1 のシフト付きソフトマックス確率が float32 で 0 にアンダーフローするため、クランプ付きのスクラッチ損失 `cross_entropy(softmax(0 - 0_max), y)` は下限 `1e-12` に張り付き、ロジットに関する勾配が 0 になる。同じロジットとラベルを融合型の `loss`（`SoftmaxRegression` のメソッド）に渡し、自動微分によってロジットに関する勾配を計算せよ。それを $\hat{\mathbf{y}} - \mathbf{y}$（3.1 節）と比較し、クランプ付きのスクラッチ損失ではクラス 1 のロジットが動かないのに対し、融合型の損失では動く理由を説明せよ。

7. **数値形式**。半精度形式は数値を 16 ビットで格納する。FP16 は指数部 5 ビット・仮数部 10 ビット、BFLOAT16 は 8 ビットと 7 ビット、FP32 は 8 ビットと 23 ビットを持つ。各形式について、ロジット o に対して $\exp(o)$ がオーバーフローしない最大の値を求めよ。次に、ロジットが 20 に達するネットワークは BFLOAT16 ではそのまま訓練できるが、FP16 ではソフトマックスを FP32 で計算しない限り訓練できない理由を説明せよ。最後に、Equation 3.18 のシフト付きの和を $q = 10^5$ クラス（各項は高々 1）について FP16 で累積する場合を考えよ。累積和が 2,048 を超えると何が起こるか、またそれと 10 ビットの仮数部はどのように関係しているか。

3.4 分類器の評価

Fashion-MNIST のソフトマックス回帰モデルは、検証画像の約 83% を正しく分類する。では、正解率（accuracy）が高いほど常に優れた分類器といえるだろうか。それが成り立つのは、すべてのクラスが同等に重要であり、どの種類の誤りもまったく同じコストを伴う場合に限られる。実運用では、次の 3 つの日常的な条件がこの対称性を崩す。

1. **クラス不均衡：**注目する陽性クラス（positive class）がまれな場合（たとえば、患者の 1% が疾患をもつ場合や、取引の 0.1% が不正である場合）、常に多数派クラスを予測する役に立たない分類器でも、陽性を 1 件も検出しないまま 99% や 99.9% の正解率を達成してしまう。
2. **非対称な誤りコスト：**命に関わる疾患を見逃すこと（偽陰性、false negative）は、追加検査を指示すること（偽陽性、false positive）よりもはるかにコストが大きい。
3. **確率に基づく意思決定：**後段の決定規則は予測確率 $\hat{\mathbf{y}}$ の数値そのものに基づいて動作するため、この値が信頼できるのは分類器が**較正**（calibration）されている場合に限られる。

本節では、これら 3 つすべてに対応する評価指標と診断手法として、**適合率**（precision）と**再現率**（recall）、**コスト考慮型の決定閾値**（cost-sensitive decision threshold）、**信頼性図**（reliability diagram）と**期待較正誤差**（expected calibration error、ECE）、そして事後的な**温度スケーリング**（temperature scaling）を順に導入する。

3.4.1 正解率の限界

3.4.1.1 クラス不均衡

100,000 人を対象に、有病率（prevalence）が 1% の疾患（病気 1,000 人、健康 99,000 人）のスクリーニング検査を実施するとしよう。すべての検査結果を無視して常に**健康**と予測する自明な規則は、病気の患者を 1 人も検出できないにもかかわらず 99% の正解率を達成する。これを、病気の患者の 90% を検出しつつ健康な人の 5% を誤って陽性と判定する実際のスクリーニング検査と比較してみよう。

	病気（陽性）	健康（陰性）
病気と予測	900 (TP)	4,950 (FP)
健康と予測	100 (FN)	94,050 (TN)

この 2×2 の二値の混同行列（confusion matrix）において、4 つのセルはそれぞれ**真陽性**（true positive、TP $= 900$）、**偽陽性**（FP $= 4{,}950$）、**偽陰性**（FN $= 100$）、**真陰性**（true negative、TN $= 94{,}050$）である。対角成分の和をとると正解率は $94{,}950/100{,}000 = 95\%$ となるため、単純な正解率で比較すると、役に立たない「常に健康」規則（99%）が人命を救う検査（95%）よりも上位に来てしまう。

陽性クラスに特化して性能を評価するため、2 つの比率を組にして用いる。すなわち、**感度**（sensitivity）や**真陽性率**（true-positive rate）とも呼ばれる**再現率**（実際の陽性のうち検出できた割合）と、**陽性的中率**（positive predictive value）とも呼ばれる**適合率**（陽性判定のうち本当に陽性である割合）である。

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad \text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3.19}$$

1 列目を縦に読むと recall $= 900/1{,}000 = 0.90$ となる（「常に健康」では 0）。1 行目を横に読むと precision $= 900/5{,}850 \approx 0.15$ となる。陽性と判定された人のうち実際に病気なのはわずか 15% にすぎないが、これは大多数を占める健康な人々からの 5% の誤報（4,950 人）が 900 人の真陽性を上回るためである（1.6.5.2 節で見たベイズの定理の効果）。2 段階の臨床スクリーニングにおいては、再現率 90%、適合率 15% という特性は理想的である。安価な一次検査で病気の患者のほぼ全員を拾い上げ、陽性判定となった 5,850 人だけを確定診断のための精密検査へ回せばよいからである。

この検査固有の再現率（0.90）と誤報率（0.05）を固定したまま、疾患の有病率を $\{0.1\%, 1\%, 10\%, 50\%\}$ と変化させてみよう。

```
recall, false_alarms = 0.9, 0.05  # Properties of the test
for prevalence in [0.001, 0.01, 0.1, 0.5]:
    tp, fp = recall * prevalence, false_alarms * (1 - prevalence)
    accuracy = tp + (1 - false_alarms) * (1 - prevalence)
    print(f'{prevalence:5.1%} sick: accuracy {accuracy:.3f}, '
          f'recall {recall:.2f}, precision {tp / (tp + fp):.3f}')
```

```
 0.1% sick: accuracy 0.950, recall 0.90, precision 0.018
 1.0% sick: accuracy 0.950, recall 0.90, precision 0.154
10.0% sick: accuracy 0.945, recall 0.90, precision 0.667
50.0% sick: accuracy 0.925, recall 0.90, precision 0.947
```

何が変化し、何が不変であるかに注目しよう。再現率と誤報率は**分類器のみ**の性質であるのに対し、適合率は**集団の有病率**に大きく依存する。

3.4.1.2 非対称なコスト

病気の患者を見逃すコストを $\kappa_{\text{FN}} = 100$、誤報のコストを $\kappa_{\text{FP}} = 1$ とする（正しい判定のコストは 0 とする）。これらのペナルティで重み付けすると、「常に健康」規則では 1 人あたりの期待コストが $100 \times 1{,}000/100{,}000 = 1.00$ かかるのに対し、先ほどのスクリーニング検査では 1 人あたり $(100 \times 100 + 1 \times 4{,}950)/100{,}000 \approx 0.15$ となり、ほぼ 7 分の 1 に抑えられる。

より一般に、分類器がある個人に対して確率 $p = P(\text{sick} \mid \mathbf{x})$ を推定するとき、**健康**と予測する場合の期待コストは $p\,\kappa_{\text{FN}}$、**病気**と予測する場合の期待コストは $(1-p)\,\kappa_{\text{FP}}$ である。したがって、次の条件を満たすときに**病気**と予測すれば期待コストが最小になる。

$$p\,\kappa_{\text{FN}} > (1-p)\,\kappa_{\text{FP}}, \quad \text{that is,} \quad p > \frac{\kappa_{\text{FP}}}{\kappa_{\text{FP}} + \kappa_{\text{FN}}}. \tag{3.20}$$

コストが等しい場合（$\kappa_{\text{FN}} = \kappa_{\text{FP}}$）には、おなじみの閾値 $1/2$（argmax 分類）に戻る。しかし、$\kappa_{\text{FN}} = 100$、$\kappa_{\text{FP}} = 1$ の場合、最適な閾値は $1/101 \approx 0.01$ まで下がり、予測リスクがわずか 2% であっても患者を陽性と判定すべきことになる。閾値を下げると適合率を犠牲にして再現率が向上する（コストが明示されていない場合、実務ではこのトレードオフを適合率と再現率の調和平均である **F1 スコア** $= 2\,\text{precision}\cdot\text{recall}/(\text{precision}+\text{recall})$ で要約するか、ROC 曲線や適合率・再現率曲線によってすべての閾値を走査することが多い。演習問題 4 および 5 を参照）。

3.4.2 較正

3.4.2.1 信頼性図

Equation 3.20 のようなコスト考慮型の決定規則は p の数値に直接作用するため、分類器が**較正**されていることを前提としている。すなわち、分類器が確信度（confidence）$\hat{p} \approx 0.80$ と予測したすべての例のうち、実際に約 80% が正解でなければならない。

q クラス問題において最大クラスの較正を確認するには、各検証予測の**確信度** $\hat{p}^{(i)} = \max_j \hat{y}_j^{(i)}$ と正誤の指示変数 $c^{(i)} = \mathbf{1}(\text{argmax}_j\, \hat{y}_j^{(i)} = y^{(i)})$ を記録し、データ例を確信度ビン $[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1.0]$ に分割して、各ビンの平均確信度 $\bar{p}_b$ と経験正解率（empirical accuracy）a_b を比較する。a_b を $\bar{p}_b$ に対してプロットしたものが**信頼性図**であり（完全に較正されたモデルは対角線 $a_b = \bar{p}_b$ 上に並ぶ）、両者の差をサンプル数で重み付け平均したものが**期待較正誤差**（ECE）として定義される。

$$\text{ECE} = \sum_b \frac{n_b}{n} \left| a_b - \bar{p}_b \right|. \tag{3.21}$$

簡潔な実装の `SoftmaxRegression` モデルを Fashion-MNIST で訓練し、検証セットのロジット `O` とラベル `y` を収集して、ビンごとの信頼性統計量、ECE、および信頼性図を計算してみよう。

```
data = d2l.FashionMNIST(batch_size=256)
model = d2l.SoftmaxRegression(num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.board.display = False
trainer.fit(model, data)
```

```
O, y = [], []
for X, Y in data.val_dataloader():
    O.append(d2l.numpy(model(X)))
    y.append(d2l.numpy(Y))
O, y = np.concatenate(O), np.concatenate(y)
print(f'validation accuracy {(O.argmax(axis=1) == y).mean():.3f}')
```

```
validation accuracy 0.831
```

各予測の確信度を計算するには、数値的にシフトしたソフトマックス（3.3 節）を NumPy でロジットに適用し、行ごとの最大確率をとる。

```
def softmax(O):
    """Row-wise softmax of a NumPy array of logits."""
    E = np.exp(O - O.max(axis=1, keepdims=True))
    return E / E.sum(axis=1, keepdims=True)

conf = softmax(O).max(axis=1)  # Confidence of each prediction
correct = (O.argmax(axis=1) == y).astype(float)
```

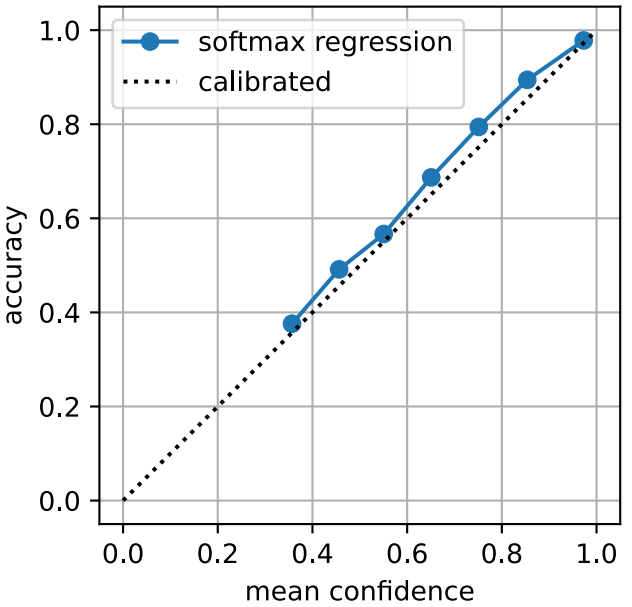
次に、`reliability(conf, correct, num_bins=10)` は `np.bincount` を用いて検証予測を等幅の 10 個の確信度ビンにグループ化し、ビンごとのサンプル数 `n`、平均確信度 `mean_conf`、経験正解率 `acc`、およびサンプル数で重み付けした `ece` を返す。

```
def reliability(conf, correct, num_bins=10):
    """Count, mean confidence, and accuracy per confidence bin; the ECE."""
    bins = np.minimum((conf * num_bins).astype(int), num_bins - 1)
    n = np.bincount(bins, minlength=num_bins)
    mean_conf = np.bincount(bins, conf, num_bins) / np.maximum(n, 1)
    acc = np.bincount(bins, correct, num_bins) / np.maximum(n, 1)
    return n, mean_conf, acc, np.sum(n * np.abs(acc - mean_conf)) / len(conf)

n, mean_conf, acc, ece = reliability(conf, correct)
for b in np.flatnonzero(n):
    print(f'confidence {b / 10:.1f} to {(b + 1) / 10:.1f}: {n[b]:4d} '
          f'predictions, mean {mean_conf[b]:.3f}, accuracy {acc[b]:.3f}')
print(f'ECE: {ece:.3f}')
```

```
confidence 0.1 to 0.2:    2 predictions, mean 0.186, accuracy 0.000
confidence 0.2 to 0.3:   51 predictions, mean 0.266, accuracy 0.275
confidence 0.3 to 0.4:  274 predictions, mean 0.357, accuracy 0.376
confidence 0.4 to 0.5:  708 predictions, mean 0.456, accuracy 0.492
confidence 0.5 to 0.6:  886 predictions, mean 0.550, accuracy 0.567
confidence 0.6 to 0.7:  888 predictions, mean 0.651, accuracy 0.687
confidence 0.7 to 0.8:  982 predictions, mean 0.751, accuracy 0.794
confidence 0.8 to 0.9: 1392 predictions, mean 0.854, accuracy 0.894
confidence 0.9 to 1.0: 4817 predictions, mean 0.973, accuracy 0.978
ECE: 0.020
```

少数サンプルのばらつきで曲線が歪まないよう、少なくとも 100 個の予測を含む十分なサンプル数のビンに絞り込み（`used = n >= 100`）、`acc[used]` を `mean_conf[used]` に対して理想的な対角線 $y = x$ とともにプロットする。

```
used = n >= 100  # Fewer predictions give too noisy an accuracy
d2l.plot([mean_conf[used], [0, 1]], [acc[used], [0, 1]], 'mean confidence',
         'accuracy', legend=['softmax regression', 'calibrated'],
         fmts=('o-', 'k:'), figsize=(3.5, 3.5))
```



十分なサンプル数をもつすべてのビン（$n_b \geq 100$）にわたり、経験正解率は平均確信度と 8 パーセントポイント以内で追随しており、対角線よりわずかに**上方**に位置している（平均確信度 $\sim 81\%$ に対して正解率 $\sim 83\%$ であり、ECE は 0.03 未満）。深刻な過信（overconfidence）に陥りやすい過剰パラメータ化された深層ネットワークとは異なり、Fashion-MNIST 上の浅いソフトマックス回帰はわずかに過小確信（underconfidence）の傾向にあるものの、すでに良好に較正されている。

3.4.2.2 温度スケーリング

信頼性図によって較正のずれが明らかになった場合、**温度スケーリング** (Guo et al. 2017) は 1 パラメータによる事後的な補正手段となる。すなわち、訓練済みの重みを固定したまま、ロジットをスカラー温度 $T > 0$（3.1 節）で割り、ホールドアウト検証データ上で交差エントロピーを最小化するように T を当てはめる。$T > 0$ で割ってもロジットの順序は保たれるため、温度スケーリングによってクラス予測が 1 つでも変わったり正解率が変化したりすることは決してなく、確信度のスケールのみが調整される（$T > 1$ は過信を抑え、$T < 1$ は過小確信を補正して分布を鋭くする）。Fashion-MNIST の検証セットの前半を使って T を当てはめ、後半で ECE を確認してみよう。

```
def nll(O, y, T):
    """Mean cross-entropy of softmax(O / T), computed from the logits."""
    Z = O / T
    m = Z.max(axis=1)
    lse = m + np.log(np.exp(Z - m[:, None]).sum(axis=1))
```

```
    return np.mean(lse - Z[np.arange(len(y)), y])

half = len(y) // 2
Ts = np.linspace(0.5, 2, 151)
T = Ts[np.argmin([nll(O[:half], y[:half], t) for t in Ts])]
ece = lambda Z: reliability(softmax(Z).max(axis=1), correct[half:])[3]
print(f'T = {T:.2f}; ECE on the second half: {ece(O[half:]):.3f} before, '
      f'{ece(O[half:] / T):.3f} after scaling')
```

```
T = 0.90; ECE on the second half: 0.016 before, 0.009 after scaling
```

わずかに過小確信の傾向にあった信頼性曲線と整合的に、当てはめた温度は 1 をわずかに下回り（$T \approx 0.9$）、もともと非常に小さかった ECE は実質的に変化しない。

3.4.3 まとめ

単純な正解率が適切な評価指標となるのは、クラスが均衡しており、すべての誤りのコストが等しい場合に限られる。陽性例がまれな場合には真陰性が全体の正解率を支配するため、陽性検出を**再現率**（$\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$）、分類器固有の性質）と**適合率**（$\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$、ベイズの定理により集団の有病率に応じて変化する）に切り分けて評価する。偽陰性と偽陽性が異なるコスト κ_{FN} と κ_{FP} を伴う場合、$p > \kappa_{\mathrm{FP}}/(\kappa_{\mathrm{FP}}+\kappa_{\mathrm{FN}})$ のときに陽性と予測することで期待コストが最小化されるが、この規則は予測確率が較正されていることを前提としている。信頼性図と期待較正誤差はその一致度を検証するものであり、ホールドアウトデータ上で単一の事後的な温度 $T > 0$ を当てはめることで、正解率を変えずに確信度を再スケーリングできる。

3.4.4 演習問題

1. **最適な意思決定と棄却**。多クラス問題において、κ_{ij} をクラス i と予測し真のクラスが j であったときのコストとし、$p(j \mid x)$ をクラス j の確率（入力 x が与えられた場合）とする。
   1. クラス i と予測する場合の期待コストを各コストおよび $p(j \mid x)$ を用いて表し、最適な i を選ぶ規則を述べよ。また、2 クラスの場合に Equation 3.20 に帰着することを確認せよ。
   2. 正しい推測のコストを 0、誤った推測のコストを λ_s とし、分類器が推測する代わりに固定コスト $\lambda_r < \lambda_s$ で**棄権**できるようにする。最大クラスの確率がどの閾値を超えたときに推測が棄権よりも有利になるかを導出せよ。
   3. 医療診断では、誤診に比べて専門医への引き継ぎは低コストであり、たとえば $\lambda_r/\lambda_s = 0.05$ であるとする。このときの閾値を計算し、確信度 0.9 の予測に対して分類器がどう振る舞うべきかを述べよ。さらに、推測や棄権のいずれよりも、分布 $p(y \mid x)$ 全体を提示する方が医師にとって有益となる場合を説明せよ。

   Kevin Murphy 著『Probabilistic Machine Learning: An Introduction』演習問題 5.1 を改変。

2. **クラス重み付き損失**。先に 3.4.1 節で見たスクリーニングの例では、病気の患者の見逃しは誤報よりもはるかに大きなコストを伴う。各クラス j に重み $w_j > 0$ を与え、モデル f を次のクラス重み付き平均損失で訓練する。

$$\frac{1}{n}\sum_{i=1}^{n} w_{y^{(i)}}\, l\big(f(\mathbf{x}^{(i)}), y^{(i)}\big), \tag{3.22}$$

ここで、l は交差エントロピー損失、$y^{(i)}$ はデータ例 i のクラスであり、クラス j の各データ例は w_j 回分として数えられる。

1. 訓練例がクラスの事前分布 $p(y)$ とクラス条件付き密度 $p(\mathbf{x} \mid y)$ に従って抽出されるとする。クラス重み付き損失の期待値は、同じクラス条件付き密度をもちクラスの事前分布が $w_y\, p(y)$ に比例する分布のもとでの重みなし損失の期待値に比例することを示せ。また、損失ではなく訓練データをどのように変更すれば同じ効果が得られるか答えよ。
2. 入力 $\mathbf{x}$（クラス確率は $p(j \mid \mathbf{x})$）を固定し、モデルが任意の分布 $\hat{\mathbf{y}}$ を出力できると仮定する。重み付き損失の期待値

$$-\sum_j w_j\, p(j \mid \mathbf{x}) \log \hat{y}_j \tag{3.23}$$

が $\hat{y}_j \propto w_j\, p(j \mid \mathbf{x})$ によって最小化されることを示せ。
3. 健康と病気の 2 クラスについて、$p(\text{sick} \mid \mathbf{x})$ の閾値であって、それを超えると $\arg\max$ をこの $\hat{\mathbf{y}}$ にとった結果が病気となるものを求め、その閾値が Equation 3.20 の閾値と等しくなるように重みを選べ。さらに、$w_{\text{sick}} > w_{\text{healthy}}$ と選ぶとなぜ再現率が上がり、偽陽性の数にどのような影響が生じるかを説明せよ。

3. **較正と有用性**。ある 10 クラス問題では、どのクラスも同じ数のデータ例をもつとする。

1. ある分類器は入力を無視してすべてのクラスに確率 0.1 を出力し、同点の場合はランダムにクラスを選ぶ。この分類器が較正されていることを示し、その正解率を計算せよ。
2. 別の分類器はデータ例の 95% で正解し、常に確信度 1 を出力する。この分類器の信頼性図を描き、期待較正誤差を計算せよ。
3. 較正と正解率がそれぞれ異なる性質を測っている理由、および意思決定に閾値 Equation 3.20 を用いる際にどちらも他方の代わりにはならない理由を説明せよ。

4. [code] **ROC 曲線**。陽性クラスに対してスコア $s \in (0, 1)$ を出力する二値分類器は、任意の $\tau \in [0, 1]$（$\frac{1}{2}$ に限らない）を閾値にでき、$s > \tau$ のときに陽性と予測する。**受信者動作特性**（ROC）曲線は、真陽性率（再現率）を偽陽性率 $\text{FP}/(\text{FP} + \text{TN})$ に対して、τ を 1 から 0 まで動かしながらプロットしたものである。1000 個の陽性例と 1000 個の陰性例をスコア $s = \sigma(z)$ でシミュレーションせよ。ただし、$\sigma(z) = 1/(1 + \exp(-z))$ はシグモイドであり、陽性例では $z \sim \mathcal{N}(\mu, 1)$、陰性例では $z \sim \mathcal{N}(0, 1)$ とする。

1. $\mu \in \{0, 1, 2\}$ に対して ROC 曲線を計算してプロットし、端点 $\tau = 1$ と $\tau = 0$ がプロット上のどの点を与えるかを述べよ。
2. スコアがラベルと独立である分類器は、期待値において対角線を描くことを示せ。
3. $\mu = 2$ に対して、適合率を再現率に対して τ を動かしながらプロットした**適合率・再現率曲線**を、陽性例と陰性例が同数の場合と、陽性例が全体の 1% を占める場合（陰性例 1000 個を保ち陽性例 10 個を抽出するか、重み付けし直す）のそれぞれについて描け。陽性率が変わっても ROC 曲線が変化しない一方で適合率・再現率曲線が低下する理由、および両者のうちどちらが本節のスクリーニング検査をより適切に表しているかを説明せよ。

5. [code] **ランキング確率としての AUC**。●この問題では、問題 4 の ROC 曲線とシミュレーションしたスコアを用いる。

1. どの 2 つのスコアも同点にならないとき、ROC 曲線の下側の面積（AUC）は、ランダムに選んだ陽性例がランダムに選んだ陰性例よりも高いスコアを得る確率に等しく、したがって閾値に依存しないランキング品質の要約指標となることを示せ。

2. シミュレーションしたスコアについてこの確率を閉じた式で求め、各曲線の下側の台形近似による面積と比較せよ。
3. 訓練済みのソフトマックス回帰に対し、2 つのクラス a と b の検証画像上で、a を陽性クラス、スコアを 2 つのロジットの差のシグモイド $s = \sigma(o_a - o_b)$ として ROC 曲線と AUC を適用せよ。対象はスニーカー対サンダル、およびシャツ対 T シャツの 2 組とする。2 つの AUC の大小関係を、同じペアに対する混同行列 **C**（3.3 節）の成分と比較し、これら 2 つの診断指標が必ずしも一致しない理由を説明せよ。

6. [code] **温度スケーリング**。本文では、検証画像の前半を使って T を当てはめた。
   1. スケーリング前後の後半データに対する信頼性図を描き、正解率が変化しないことを示せ。
   2. 代わりに訓練画像上で T を当てはめ、本文の T と比較せよ。モデルの訓練に使っていないデータで当てはめるべき理由を説明せよ。
   3. スケーリング前のモデルの期待較正誤差を 5、10、20 個のビンでそれぞれ計算せよ。その値はビン分割にどの程度依存するか。

3.5 分類における汎化

Fashion-MNIST の分類器は、約 83% の正解率を 10,000 枚の検証画像に対して達成した。将来の衣類画像に対する性能保証として、この数値をどこまで信頼できるだろうか。そこには 2 つの根本的な障害が立ちはだかっている。

1. **有限標本の不確実性とテストセットの再利用：**未使用のテストセットであっても、$n = 10{,}000$ 枚の画像は有限個の無作為標本（random sample）にすぎない。さらに悪いことに、実際の開発ではモデルの比較やハイパーパラメータの調整を進めるなかで、同じ評価分割を繰り返し参照してしまう。
2. **暗記と学習：**すべての訓練画像が互いに異なる高次元空間では、ルックアップテーブル型の暗記器（memorizer）は 100% の訓練正解率を達成しながら、未知の画像に対してはランダムに推測するだけである。暗記器の訓練誤差がゼロであっても何も保証されないのに、**線形**分類器の低い訓練誤差が低い母集団誤差を保証するのはなぜだろうか。

2.6 節では回帰における過学習と検証分割を定性的に導入したが、本節では分類における汎化（generalization）を定量的に扱う。具体的には、サイズ n のテストセットに対する厳密な信頼区間（confidence interval）と漸近的な信頼区間を導出し、テストセットの再利用のもとで選択バイアス（selection bias）と適応的過学習（adaptive overfitting）がいかにスコアを水増しするかを定量化したうえで、統計学習理論から **Vapnik–Chervonenkis（VC）次元**と**一様収束**（uniform convergence）を導入する。

3.5.1 テストセット

3.5.1.1 経験誤差と母集団誤差

単一の分類器 f は、独立なテストセット $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ を母集団分布 $P(X, Y)$ から独立同分布（i.i.d.）に抽出する**前に**固定しておく。このとき、$\mathcal{D}$ 上での**経験誤差**（empirical error）と真の**母集団誤差**（population error）は次式で定義される。

$$\epsilon_{\mathcal{D}}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\left(f(\mathbf{x}^{(i)}) \neq y^{(i)}\right), \qquad \epsilon(f) = P\left(f(X) \neq Y\right). \tag{3.24}$$

f は事前に固定されているため、各誤り指示変数 $\mathbf{1}(f(\mathbf{x}^{(i)}) \neq y^{(i)})$ は独立な Bernoulli$(\epsilon(f))$ のコイン投げとなり、その平均は $\epsilon(f)$、分散は $\epsilon(f)(1-\epsilon(f)) \leq 1/4$ である。したがって、テスト誤差 $\epsilon_{\mathcal{D}}(f)$ は不偏推定量（unbiased estimator）であり（$\mathbb{E}[\epsilon_{\mathcal{D}}(f)] = \epsilon(f)$）、その標準誤差（standard error）は次式となる。

$$\mathrm{SE}(\epsilon_{\mathcal{D}}(f)) = \sqrt{\frac{\epsilon(f)(1-\epsilon(f))}{n}} \leq \frac{0.5}{\sqrt{n}}. \tag{3.25}$$

Fashion-MNIST の分類器（$\epsilon(f) \approx 0.17$）を $n = 10{,}000$ 枚のテスト画像で評価した場合、標準誤差は $\sqrt{0.17 \times 0.83/10{,}000} \approx 0.0038$（0.38 パーセントポイント）となる。ここで $1/\sqrt{n}$ のスケーリング則（scaling law）に注目しよう。測定の不確実性を半分にするには、テスト例が **4 倍**必要になる。

3.5.1.2 信頼区間と Hoeffding の不等式

標準誤差を 95% 信頼区間 $\epsilon_{\mathcal{D}}(f) \pm r$ に変換するには、2 つの手法がある。

1. **漸近正規区間（中心極限定理、central limit theorem）：**n が大きいとき、$\epsilon_{\mathcal{D}}(f)$ は近似的に正規分布に従い、標準誤差の 1.96 倍以内の幅で $\epsilon(f)$ の周りに確率 ≈ 0.95 で収まる。

$$P\left(|\epsilon_{\mathcal{D}}(f) - \epsilon(f)| \leq 1.96\sqrt{\frac{\epsilon(f)(1-\epsilon(f))}{n}}\right) \approx 0.95. \tag{3.26}$$

   $\epsilon \approx 0.17$、$n = 10{,}000$ のとき、95% 正規信頼半径は $1.96 \times 0.0038 \approx 0.007$（$\pm 0.7$ パーセントポイント）である。最悪の場合（$\epsilon = 1/2$）でも、95% 信頼半径 ≤ 0.01 を保証するには $n \geq (1.96 \times 0.5/0.01)^2 = 9{,}604$ 個の例があればよい。これは MNIST、Fashion-MNIST、CIFAR-10 のテストセットが 10,000 例で構成されている理由を理解するうえで有益な見方であり、いずれも正解率を約 1 ポイントの精度で特定できる。

2. **有限標本 Hoeffding 区間：**$\epsilon(f)$ が 0 に非常に近い場合、正規近似では被覆率が著しく低下し得るため（小さなテストセットでは誤分類がゼロとなってプラグイン正規半径が 0 に潰れることがある）、**Hoeffding の不等式** (Hoeffding 1963) は各誤りが $[0,1]$ の範囲にあるという事実のみを用いて、あらゆる n とあらゆる分布に対して成り立つ厳密な上界を与える（36.9 節）。

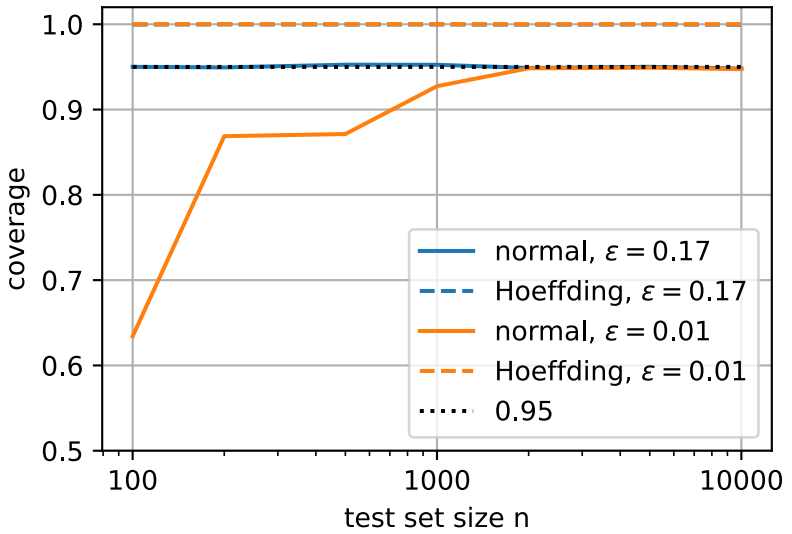
$$P\left(|\epsilon_{\mathcal{D}}(f) - \epsilon(f)| \geq t\right) \leq 2\exp\left(-2nt^2\right). \tag{3.27}$$

   右辺を $\delta = 0.05$ とおくと、95% Hoeffding 信頼半径 $r = \sqrt{\log(2/\delta)/(2n)}$ が得られる（これには $n \geq \log(40)/(2 \times 0.01^2) \approx 18{,}445$ 個の例が $r \leq 0.01$ の達成に必要となり、正規近似による必要数の約 2 倍にあたる。どちらも最悪ケースの分散 $1/4$ を仮定しているが、Hoeffding の指数型裾評価はすべての n で成り立つぶん、正規分布の厳密な分位点（quantile）より緩いためである）。

10,000 個の独立なテストセットをサイズ $n \in [100, 10{,}000]$ にわたって 2 つの分類器（$\epsilon = 0.17$ と $\epsilon = 0.01$）についてシミュレーションし、両方の 95% 信頼区間の経験被覆率（empirical coverage）をプロットしてみよう。

```
rng = np.random.default_rng()
num_sets = 10000
ns = np.array([100, 200, 500, 1000, 2000, 5000, 10000])
curves, legend = [], []
for eps in [0.17, 0.01]:
```

```
    # The number of errors on a test set of size n is Binomial(n, eps)
    errs = [rng.binomial(n, eps, num_sets) / n for n in ns]
    normal = [1.96 * np.sqrt(e * (1 - e) / n) for e, n in zip(errs, ns)]
    hoeffding = [np.sqrt(np.log(2 / 0.05) / (2 * n)) for n in ns]
    for name, radii in [('normal', normal), ('Hoeffding', hoeffding)]:
        curves.append([np.mean(np.abs(e - eps) <= r)  # coverage
                       for e, r in zip(errs, radii)])
        legend.append(rf'{name}, $\epsilon = {eps}$')
d2l.plot(ns, curves + [[0.95] * len(ns)], 'test set size n', 'coverage',
         legend=legend + ['0.95'], xscale='log', ylim=(0.5, 1.02),
         fmts=('C0-', 'C0--', 'C1-', 'C1--', 'k:'), figsize=(4.5, 3))
```



$\epsilon = 0.17$ では、プラグイン正規区間はほぼ 95% の被覆率をすべての n で維持している。一方、$\epsilon = 0.01$ で n が小さいときには、誤差ゼロのテストセットが生じるたびに正規区間の被覆率は $\sim 63\%$ まで落ち込むのに対し、Hoeffding の保守的な区間はあらゆる領域で 95% の被覆率を上回っている。

3.5.2 テストセットの再利用

3.5.2.1 多数の分類器からの選択

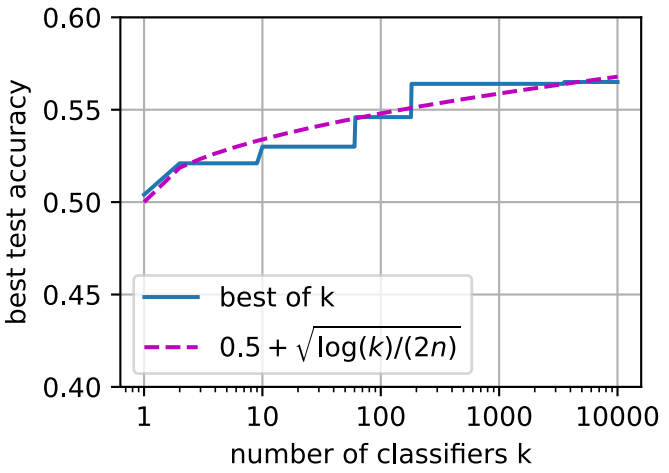
k 個の固定された候補分類器 $f_1, \ldots, f_k$ を**同じ**テストセット $\mathcal{D}$ 上で評価し、最も成績の良かったものを報告すると何が起きるだろうか。たとえ k 個の候補すべてが $\mathcal{D}$ の抽出前に固定されていたとしても、最高スコアを選ぶことは、$\mathcal{D}$ 上でたまたま最も運が良かった候補をつまみ食い（チェリーピッキング）することになる。**ユニオンバウンド**（k 個の悪い事象のうち少なくとも 1 つが起こる確率は、個々の確率の和で上から抑えられるという性質）を Equation 3.27 に適用すると、次式が得られる。

$$P\left(\max_{j=1,\ldots,k} |\epsilon_{\mathcal{D}}(f_j) - \epsilon(f_j)| \geq t\right) \leq 2k \exp\left(-2nt^2\right). \tag{3.28}$$

右辺を δ とおくと、信頼度 $1-\delta$ の精度を k 個すべてのモデルに対して同時に保証するには、信

頼半径が $\sqrt{\log(2k/\delta)/(2n)}$ まで広がることがわかる。この穏やかな $\sqrt{\log k}$ の増大に注目しよう。$k = 1{,}000$ 個の固定モデルを比較しても半径は対数因子ぶんしか大きくならないが、それでも $n = 1{,}000$ 個のコイン投げで決めたテストラベルに対し、純粋にランダムなコイン投げで推測する $k = 10{,}000$ 個の予測器（いずれも真の正解率は 0.50）を評価すると、最も運の良かった予測器のテスト正解率は $0.5 + \sqrt{\log(k)/(2n)}$ に沿って着実に上昇し、~ 0.56 に達する。

```
n, k = 1000, 10000
labels = rng.random(n) < 0.5            # one test set with coin-flip labels
guesses = rng.random((k, n)) < 0.5      # k classifiers that guess at random
accuracy = (guesses == labels).mean(axis=1)
ks = np.arange(1, k + 1)
d2l.plot(ks, [np.maximum.accumulate(accuracy),
              0.5 + np.sqrt(np.log(ks) / (2 * n))],
         'number of classifiers k', 'best test accuracy',
         legend=['best of k', r'$0.5 + \sqrt{\log(k) / (2n)}$'], xscale='log',
         ylim=(0.4, 0.6))
```



3.5.2.2 適応的過学習

過去のテストスコアに基づいて新たな候補を**適応的に**選ぶようになると、選択バイアスははるかに危険なものとなる (Dwork et al. 2015)。テスト結果のフィードバックがモデル設計を導くようになると、分類器はもはや $\mathcal{D}$ と独立ではなくなり、ユニオンバウンド Equation 3.28 は完全に破綻する。Blum and Hardt (2015) に倣い、先ほどのランダムなコイン投げ予測器のうち最初の 1,000 個だけを取り出し、テストセット上でたまたま 0.50 を超えるスコアを出したものだけを選んで、各テスト例に対して多数決をとるとどうなるか見てみよう。

```
kept = guesses[:1000][accuracy[:1000] > 0.5]  # guessers that scored above 0.5
vote = kept.mean(axis=0) > 0.5                # their majority vote
print(f'majority vote: {(vote == labels).mean():.3f}, '
      f'best single guesser: {accuracy[:1000].max():.3f}')
```

```
majority vote: 0.724, best single guesser: 0.564
```

個々のテストラベルを一度も直接見ることなく、わずか 1,000 回のスカラーの正解率クエリだけ

で、純粋なコイン投げ予測器の多数決はテスト正解率で $\sim 70\%$ に跳ね上がる一方、未知のデータに対する母集団正解率は依然として 50% のままである（実際、$O(n)$ 回のクエリがあればすべてのテストラベルを復元できる。演習問題 3 を参照）。だからこそ、ハイパーパラメータ最適化とアーキテクチャ探索は検証分割上で行い（演習問題 4）、テストセットは一度限りの最終評価のために残しておかなければならない。

10 年にわたる再利用は、標準的なベンチマークにどれほどの悪影響を与えてきたのだろうか。Recht et al. (2019) が CIFAR-10 と ImageNet の新しいテストセットを収集したところ、正解率は 3 から 15 ポイント低下したが、モデル間のランキングはほとんど変わらなかった。上述のような最悪の事態は起こり得るものの、通常のモデル開発におけるテストセットへの過学習は、理論が許容する最悪ケースよりはるかに小さかったのである。正解率の低下そのものは、データ分布（data distribution）のわずかな変化によってより適切に説明される。これについては 3.6 節で扱う。

3.5.3 統計学習理論

3.5.3.1 一様収束

分類器 $f_{\mathcal{S}}$ を仮説クラス（hypothesis class）$\mathcal{F}$ の中から、訓練誤差 $\epsilon_{\mathcal{S}}(f)$（訓練セット $\mathcal{S}$、サイズ n 上）の最小化によって選んだとき、それが母集団へ汎化することをどうすれば**事前に**（a priori）保証できるだろうか。学習器による $f_{\mathcal{S}} \in \mathcal{F}$ の選択は $\mathcal{S}$ を調べたあとに行われるため、訓練誤差が母集団誤差の近くにとどまるという性質が**すべての** $f \in \mathcal{F}$ **に対して同時に**成り立つ必要がある（**一様収束**）。$\mathcal{F}$ が有限（$|\mathcal{F}| < \infty$）であれば、ユニオンバウンド Equation 3.28 を全 $k = |\mathcal{F}|$ 個の仮説に対して $\mathcal{S}$ 上で適用することで、少なくとも確率 $1 - \delta$ で次式が成り立つ。

$$|\epsilon_{\mathcal{S}}(f) - \epsilon(f)| \le \sqrt{\frac{\log(2|\mathcal{F}|/\delta)}{2n}} \quad \text{for every } f \in \mathcal{F}. \tag{3.29}$$

3.5.3.2 VC 次元と VC 上界

線形半空間 $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b)$（$\mathbb{R}^d$ 上）のような連続パラメータ族では $|\mathcal{F}| = \infty$ となるため、Equation 3.29 は自明（vacuous）となり意味をなさない。しかし Vapnik and Chervonenkis (1971) は、任意の有限な n 個の点の集合上では、まったく同じ ± 1 の二値ラベル付けを生成する 2 つの分類器は $\mathcal{S}$ 上で区別できないことに着目した。$\mathcal{F}$ が m 個の点の集合を**粉砕する**（shatter）とは、$\mathcal{F}$ 内の関数によって全 2^m 通りの可能な二値ラベル付けをその m 点上で実現できることをいう。そして **Vapnik–Chervonenkis（VC）次元** $v = \mathrm{VC}(\mathcal{F})$ とは、$\mathcal{F}$ が粉砕できる最大の点集合のサイズである。

- **平面（$\mathbb{R}^2$）上の直線の VC 次元は 3（Figure 3.6）：**同一直線上にない任意の 3 点は直線によって全 $2^3 = 8$ 通りに粉砕できるのに対し、平面 $\mathbb{R}^2$ 上のいかなる 4 点も決して粉砕できない（凸四角形の向かい合う頂点同士を分離するには対角線を交差させる必要があり、また三角形の 3 頂点を含む半平面はその内部の任意の点も自動的に含んでしまうためである）。
- **$\mathbb{R}^d$ 内の線形半空間の VC 次元は $d + 1$（演習問題 5）：**線形半空間は原点と d 個の標準基底ベクトル（basis vector）を粉砕でき、Radon の定理により $d + 2$ 個の点（$\mathbb{R}^d$ 内）は決して粉砕できない。
- **制約のないルックアップテーブル型の暗記器は $v = \infty$：**互いに異なる点のあらゆる有限集合を粉砕できる。

Sauer の補題により、$n \ge v$ となると、VC 次元 v のクラスが n 個の点上に誘導できる異なるラベル付けの総数は多項式オーダーでしか増大せず、高々 $(en/v)^v$ で抑えられる（2^n にはならない）。これらのラベル付けに対して対称化したユニオンバウンドを適用すると（36.10 節）、**VC 上**

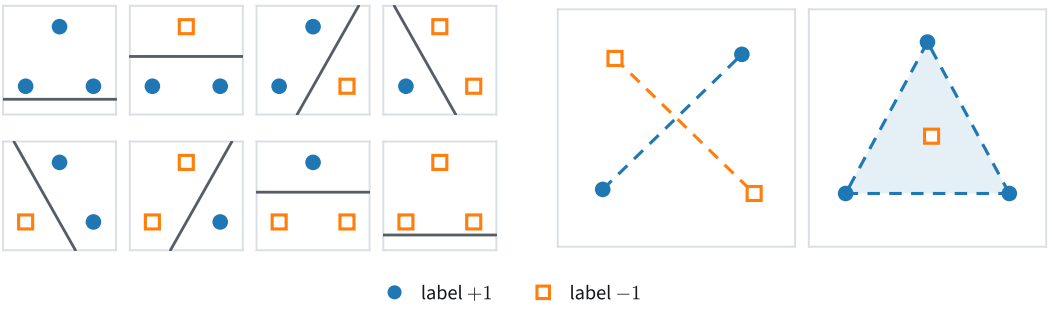


Figure 3.6: $\mathbb{R}^2$ 上の直線は同一直線上にない 3 点を粉砕できるが（左、全 $2^3 = 8$ 通りのラベル付け）、いかなる 4 点も粉砕できない（中央および右）。

界（VC bound）が得られる。すなわち、ある絶対定数 c に対し、確率 $1-\delta$ 以上（訓練セット $\mathcal{S}$ の抽出に関して）で次式が成り立つ。

$$\epsilon(f) \le \epsilon_{\mathcal{S}}(f) + c\sqrt{\frac{v + \log(1/\delta)}{n}} \quad \text{for every } f \in \mathcal{F}. \tag{3.30}$$

この定理は冒頭の概念的な問いに答えてくれる。$\mathbb{R}^d$ 内の線形分類器は有限の VC 次元 $v = d+1$ を持つため、$n \gg d$ のときに訓練誤差を小さくすれば、**いかなる**分布 P に対しても低い母集団誤差が保証されるのに対し、VC 次元が無限大の暗記器にはそのような上界が一切存在しない。同時に、最悪ケースの一様収束上界は敵対的な分布に対しても成り立たなければならないため、Equation 3.30 は自然画像に対しては数値的に緩くなり（1 対他の単一の半空間でも $\sqrt{785/60{,}000} \approx 0.11$ にさらに定数 c が掛かり、全 10 クラスを合わせた容量はさらに大きくなるのに対し、Fashion-MNIST での実測の訓練・テスト間ギャップは 0.01 である）、ランダムなラベルさえ暗記できる過剰パラメータ化された深層ニューラルネットワークに対しては自明（$v \ge n$）になってしまう (Zhang et al. 2021)。この謎については 4.5 節で改めて立ち返ることとし、当面は鋭い経験的測定のためにホールドアウトされたテストセットを頼りにする。

3.5.4 まとめ

n 個の例からなる未使用のテストセット上では、固定された分類器の経験誤差 $\epsilon_{\mathcal{D}}(f)$ の標準誤差は $\sqrt{\epsilon(1-\epsilon)/n} \le 0.5/\sqrt{n}$ となるため、正解率を 1 パーセントポイント以内の精度に信頼度 95% で特定するには、正規近似のもとで約 10,000 個、Hoeffding の有限標本上界のもとでは約 18,500 個のテスト例が必要になる。同じテストセットを再利用して k 個の固定モデルから選択すると、この不確実性は $\sqrt{\log(2k/\delta)/(2n)}$ に従って緩やかに広がる一方、過去のテストスコアに導かれる適応的な設計ははるかに少ないクエリ数でテストセットに過学習し得る。訓練誤差のみから汎化を上から抑えるため、統計学習理論では仮説クラスの容量を VC 次元 v（そのクラスが粉砕できる最大の点集合のサイズであり、線形半空間では $v = d+1$、ただし入力は $\mathbb{R}^d$）によって測り、一様な $O(\sqrt{v/n})$ の上界を導く。これにより、線形モデルが大規模なデータセットを暗記できない理由が説明される一方で、過剰パラメータ化された深層ネットワークに対してはこの上界は自明なままにとどまる。

3.5.5 演習問題

1. **2 つのモデルと 1 つのテストセット**。ある論文で、モデル A は 91.2%、モデル B は 91.6% の正解率を 10,000 枚の画像のテストセット上で達成したと報告されている。
   1. 各正解率の標準誤差と、正規近似による 95% 信頼半径を計算せよ。もし 0.4 ポイントの差が、2 つのモデルをこのサイズの独立な 2 つのテストセットで評価して得られたものだったとしたら、意味のある差といえるか。
   2. 実際には、両方のモデルが同じ 10,000 枚の画像上で評価されていた。この場合、問題の本質が 2 つのモデルの予測が食い違う画像にある理由を説明し、もし論文がすべてのテスト画像についてどちらのモデルが正解したかを報告していたなら、どのように判定するか述べよ。
   3. 0.4 ポイントの差（独立に評価された 2 つのモデル間の差）が信頼半径の外に出るには、どれほど大きなテストセットが必要か。
2. **テストセット上での誤差ゼロ**。ある分類器が n 個の例からなるテストセット上で誤りを一切出さなかったとする。
   1. 正規近似において $\epsilon_{\mathcal{D}}(f)$ を $\epsilon(f)$ の代わりに代入したときに報告される区間と、Hoeffding 区間を示せ。
   2. 母集団誤差 ϵ を持つ分類器が n 個の独立なテスト例で誤りを出さない確率が $(1-\epsilon)^n \le e^{-n\epsilon}$ であることを示せ。さらに、$\epsilon \ge 3/n$ ならば誤りのないテストセットが生じる確率は 0.05 未満になることを導け（「3 の法則」）。
   3. $3/n$ と Hoeffding 信頼半径を $n = 100$ および $n = 10{,}000$ の場合について比較し、その違いを説明せよ。
3. **テストセットの漏洩**。第三者がテストセット $\mathcal{D}$（二値ラベル付きの n 個の例）を保持しており、入力のみを公開しているとする。モデルクラスに制限はなく任意のモデル f を提出でき、そのテスト誤差 $\epsilon_{\mathcal{D}}(f)$ のみを受け取れるとする。
   1. すべてのテストラベルを復元し、したがって母集団誤差がどうあれテスト誤差 0 を達成するモデルを得る戦略を与え、その戦略が使用する提出回数を求めよ。
   2. 各応答 $\epsilon_{\mathcal{D}}(f)$ は $n+1$ 通りの値のいずれかをとる。これを用いて、いかなる戦略であっても全 2^n 通りのラベル付けを識別するために必要な提出回数の下界を求めよ。
4. [code] **Fashion-MNIST の検証分割**。本章では 10,000 枚のテスト画像上でモデルを選択した。代わりに 10,000 枚を 60,000 枚の訓練画像から検証分割として取り置き、残りの 50,000 枚でソフトマックス回帰を訓練せよ。
   1. 学習率を $\{0.01, 0.03, 0.1, 0.3, 1\}$ の中から検証分割上の正解率によって選択せよ。
   2. 選択したモデルをテスト画像上で一度だけ評価し、テスト正解率を検証正解率と比較せよ。その差は 95% 信頼区間の半径（10,000 例に対するもの）以内に収まっているか。
   3. 選択後に一度だけ測定したテスト正解率は選択されたモデルに対する不偏推定値であるのに対し、最良の検証正解率はそうではない理由を説明せよ。
5. **線形分類器の VC 次元**。●線形分類器 $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b)$（$\mathbb{R}^d$ 上）を考える。
   1. これらが $d+1$ 個の点 $\mathbf{0}, \mathbf{e}_1, \dots, \mathbf{e}_d$ を粉砕することを示せ。任意のラベル $y_0, y_1, \dots, y_d \in \{\pm 1\}$ が与えられたとき、それらを生成する b と $\mathbf{w}$ を構成せよ。ヒント：$f(\mathbf{0})$ は b のみに依存するため、まず b を選ぶとよい。
   2. $\mathbf{x}_1, \dots, \mathbf{x}_{d+2} \in \mathbb{R}^d$ とする。$d+2$ 個のベクトル $[\mathbf{x}_i, 1] \in \mathbb{R}^{d+1}$（各 $\mathbf{x}_i$ の末尾に 1 を付加したもの）が線形従属であり、したがって $\sum_i a_i [\mathbf{x}_i, 1] = \mathbf{0}$ がある係数 a_i（すべてがゼロとは限らない）に対して成り立つことを示せ。添字集合 I を $a_i > 0$ となる添字全体、J を $a_i < 0$ となる添字全体とする。どちらも空集合ではなく、いかなる線形分類器も I に属する点を $+1$、J に属する点を -1 とラベル付けできないことを示せ。ヒント：$\sum_i a_i (\mathbf{w}^\top \mathbf{x}_i + b)$ を 2 通りの方法で計算するとよい。
   3. 以上から VC 次元が $d+1$ であることを結論づけ、さらに $\{\mathbf{x}_i : i \in I\}$ の凸包と $\{\mathbf{x}_j : j \in J\}$ の凸包が交わることを示せ。

6. **多項式の VC 次元**。分類器 $f(x) = \text{sign}(p(x))$（ただし p は $x \in \mathbb{R}$ に関する高々 5 次の多項式）からなるクラスの VC 次元はいくつか。また、$\mathbf{x} \in \mathbb{R}^d$ の場合はいくつか。ヒント：このような分類器は $\mathbf{x}$ の単項式上の線形分類器である。さらに、Fashion-MNIST 上の 5 次多項式分類器の訓練誤差に対して VC 上界が何を予測するか、そしてなぜ誰もそのような分類器を訓練しないのかを述べよ。

3.6 環境と分布シフト

第 3.5 章におけるすべての保証は、ある暗黙の仮定の上に成り立っていた。それは、運用時のデータが訓練セットやテストセットとまったく同じ分布から抽出されるという仮定である。現実の世界では、この仮定は絶えず破られる。スキャナやカメラはアップグレードされ、季節性インフルエンザの流行は疾患の有病率を一晩で変え、地域の俗語は変化していく。さらに厄介なことに、モデルを運用すること**それ自体**が集団の行動を変化させる場合さえある。たとえば、ある銀行が過去の記録から、オックスフォードシューズを履いた融資申込者はスニーカーの申込者よりも返済率が高いことを発見し、オックスフォードシューズを承認してスニーカーを却下する分類器を導入したとしよう。数週間もすれば、申込者は全員オックスフォードシューズを履いて現れるようになるが、根本的な債務不履行率は少しも変わらない。これは **Goodhart の法則**（統計的な相関指標が意思決定の目標になると、もはや有効な相関指標ではなくなること）の典型例である。

本節では、**ソース分布**（source distribution）である $p_S(\mathbf{x}, y)$（ラベル付き訓練データが得られる）と、**ターゲット分布**（target distribution）である $p_T(\mathbf{x}, y)$（ラベルなしの運用時入力 $\mathbf{x}$ のみが観測される）の間の分布シフト（distribution shift）を検出し、補正する方法を学ぶ。同時分布（joint distribution）のどの因子が不変に保たれるかによって受動的なシフトの代表的な 3 つの類型を整理し、**重要度重み付き共変量シフト補正**（importance-weighted covariate shift correction）と**混同行列によるラベルシフト補正**（confusion-matrix label shift correction）を導出して実装したうえで、最後に概念シフト（concept shift）とフィードバックループを扱う。

3.6.1 分布シフトの種類

$p_S(\mathbf{x}, y)$ と $p_T(\mathbf{x}, y)$ を結びつける**何らかの**構造的仮定がなければ、ラベルなしのターゲットドメインへ適応することは数学的に不可能である。もし $p_T(\mathbf{x}) = p_S(\mathbf{x})$ が同一のまま、猫と犬のラベルが一晩のうちに密かに入れ替わったとしても（$p_T(y = \text{cat} \mid \mathbf{x}) = p_S(y = \text{dog} \mid \mathbf{x})$）、ラベルなしのターゲット画像しか見ないアルゴリズムがその入れ替わりを検出することは決してできない。

あらゆる同時分布は、次の 2 通りに分解できる。

$$p(\mathbf{x}, y) = p(\mathbf{x})\, p(y \mid \mathbf{x}) = p(y)\, p(\mathbf{x} \mid y). \tag{3.31}$$

1 つの条件付き因子を不変に保ちながら周辺分布がドリフトする場合を考えることで、分布シフトの古典的な 3 つの形態が定義される（Figure 3.7）。

シフトの形態	変化するもの	不変に保たれるもの	自然な因果構造
共変量シフト	$p(\mathbf{x})$	$p(y \mid \mathbf{x})$	特徴量 $\mathbf{x}$ がラベル y の原因
ラベルシフト	$p(y)$	$p(\mathbf{x} \mid y)$	ラベル y が特徴量 $\mathbf{x}$ の原因
概念シフト	$p(y \mid \mathbf{x})$	多くの場合 $p(\mathbf{x})$	ラベルの定義が時間・空間とともにドリフト

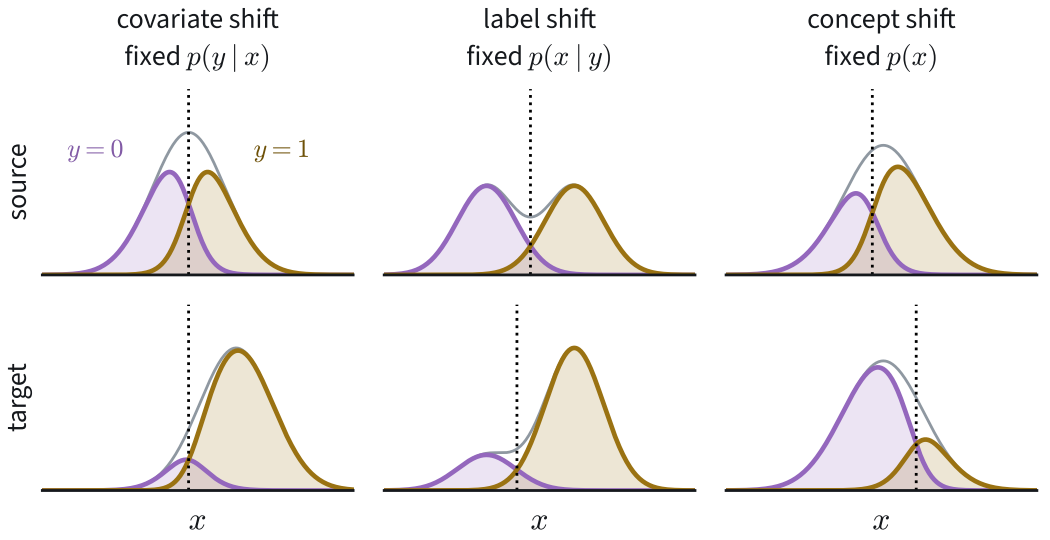


Figure 3.7: 1 次元入力 x と 2 クラスに対する分布シフトの 3 つの形態（上段：ソース p_S、下段：ターゲット p_T、点線：最適決定境界）。共変量シフトは $p(x)$ を移動させるが $p(y \mid x)$ と真の境界は固定されたままである。ラベルシフトはクラス事前分布 $p(y)$ を変化させる一方で $p(x \mid y)$ を固定したままにする（ベイズ境界は希少なクラスの側へ移動する）。概念シフトは $p(y \mid x)$ を直接変化させる。

3.6.1.1 共変量シフト、ラベルシフト、概念シフト

1. **共変量シフト（$p(y \mid \mathbf{x})$ が不変）：** $\mathbf{x}$ が y の原因となるときに自然に生じる。たとえば、猫と犬の分類器を実写写真（Figure 3.8）で訓練し、イラスト（Figure 3.9）に対して運用する場合を考えよう。イラストの猫も依然として猫であるが（$p(y \mid \mathbf{x})$ は不変）、周辺入力分布 $p(\mathbf{x})$ はピクセル空間の新たな領域へとシフトしている。真のベイズ境界は動いていなくても、ソースの実写写真に当てはめた誤特定（misspecified）されたモデルや表現力の限られたモデルは、$p_S(\mathbf{x})$ の密度が高い領域に正解率を集中させるため、$p_T(\mathbf{x})$ が分布する領域では大きく破綻し得る。
2. **ラベルシフト（$p(\mathbf{x} \mid y)$ が不変）：** y が $\mathbf{x}$ の原因となるときに自然に生じる。たとえば疾患は症状を引き起こすため、インフルエンザの流行期にはインフルエンザの有病率 $p(y)$ が急増する一方、インフルエンザ患者の症状分布 $p(\mathbf{x} \mid y)$ は変化しない。ベイズの定理により $p(y \mid \mathbf{x}) \propto p(\mathbf{x} \mid y)\,p(y)$ が成り立つため、クラスの事前分布（prior）がシフトすると最適な決定境界（decision boundary）も移動する。ラベルシフトの補正で推定が必要なのは q 個のクラス事前分布比だけであり、$\mathbf{x}$ 上の高次元の密度比（density ratio）を推定する必要はない。
3. **概念シフト（$p(y \mid \mathbf{x})$ が変化）：** 米国の郡ごとの炭酸飲料の地域的な呼び名（Figure 3.10）や、時代とともに改定される診断基準、季節ごとの購買意図など、地域や時間によってラベルそのものの意味が変化する。

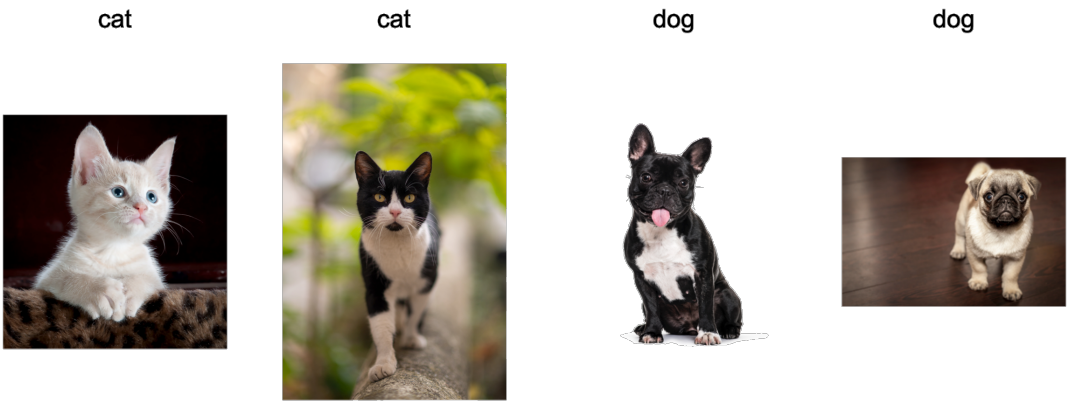


Figure 3.8: 猫と犬の分類におけるソース訓練画像：実写写真（画像提供：Lafeez Hossain / 500px / Getty Images、ilkermetinkursova / iStock / Getty Images Plus、GlobalP / iStock / Getty Images Plus、Musthafa Aboobakuru / 500px / Getty Images）。

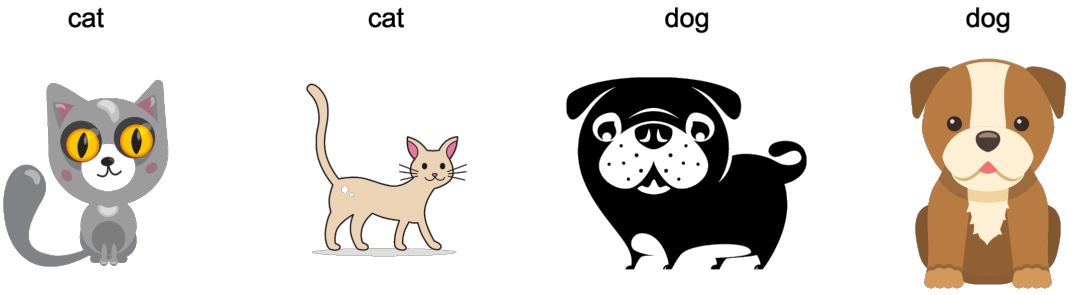


Figure 3.9: 猫と犬の分類におけるターゲット運用画像：イラスト（画像提供：SIBAS_minich / iStock / Getty Images Plus、Ghrzuzudu / iStock / Getty Images Plus、id-work / DigitalVision Vectors / Getty Images、Yime / iStock / Getty Images Plus）。

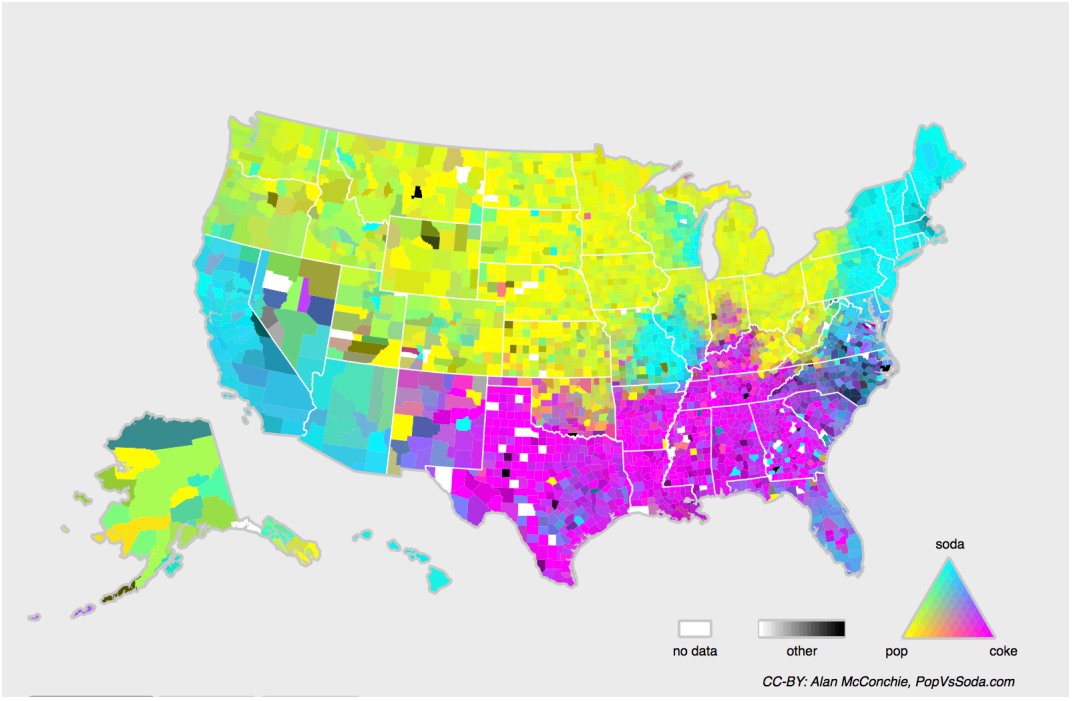


Figure 3.10: 地理的な概念シフト：米国の郡ごとに加糖炭酸飲料を指す最も一般的な呼び名（「pop」「soda」「coke」）（CC-BY: Alan McConchie, PopVsSoda.com）。

3.6.2 共変量シフトの補正

3.6.2.1 重要度重み

運用時に最小化したいのは、**ターゲットリスク** $R_T[f] = \mathbb{E}_{(\mathbf{x},y)\sim p_T}[l(f(\mathbf{x}), y)]$ である。共変量シフト（$p_T(y \mid \mathbf{x}) = p_S(y \mid \mathbf{x}) = p(y \mid \mathbf{x})$）のもとでは、ソースのサポートがターゲットを覆っている限り（すなわち $p_S(\mathbf{x}) > 0$ が $p_T(\mathbf{x}) > 0$ なるすべての点で成り立つとき）、$p_S(\mathbf{x})$ を掛けて割ることにより、ターゲットリスクを**ソース**分布に関する期待値へと書き換えられる (Shimodaira 2000)。

$$
\begin{aligned}
R_T[f] &= \sum_y \int l(f(\mathbf{x}), y)\, p(y \mid \mathbf{x})\, p_T(\mathbf{x})\, d\mathbf{x} \\
&= \sum_y \int l(f(\mathbf{x}), y)\, p(y \mid \mathbf{x})\, p_S(\mathbf{x})\, \frac{p_T(\mathbf{x})}{p_S(\mathbf{x})}\, d\mathbf{x} \\
&= \mathbb{E}_{(\mathbf{x},y)\sim p_S}\left[\beta(\mathbf{x})\, l(f(\mathbf{x}), y)\right],
\end{aligned}
\tag{3.32}
$$

ここで、$\beta(\mathbf{x}) = p_T(\mathbf{x})/p_S(\mathbf{x})$ は入力 $\mathbf{x}$ の**重要度重み**（importance weight）である。Equation 3.32 を n 個のラベル付きソース例で近似すると、次の**重み付き経験リスク最小化**（weighted empirical risk minimization）が得られる。

$$\begin{aligned} &\underset{f}{\text{minimize}}\ \frac{1}{n}\sum_{i=1}^{n}\beta_i\, l(f(\mathbf{x}^{(i)}), y^{(i)}), \\ &\text{where } \beta_i = \frac{p_T(\mathbf{x}^{(i)})}{p_S(\mathbf{x}^{(i)})}. \end{aligned} \tag{3.33}$$

サポートの重なり（support overlap、すなわち $p_S(\mathbf{x}) > 0$ が $p_T(\mathbf{x}) > 0$ なるすべての点で成り立つこと）が譲れない条件である理由に注目しよう。もし医療系スタートアップが、疾患のある血液サンプルを高齢患者から集め、健常な対照サンプルを 20 歳の大学生のみから集めたとすると、どれほど再重み付けしても $p_S(\mathbf{x}) = 0$ である領域に高齢の健常対照データを作り出すことは決してできない。たとえサポートが重なっていても、右裾で密度比が極端に大きくなると（Figure 3.11）、ごく少数の点が和を支配して**有効標本サイズ**（effective sample size）$n_{\text{eff}} = (\sum_i \beta_i)^2 / \sum_i \beta_i^2$ が激減してしまう。重みを上限で打ち切る $\min(\beta_i, c)$ のクリッピングを用いれば、わずかな統計的バイアスと引き換えに n_{eff} を大幅に増やすことができる。

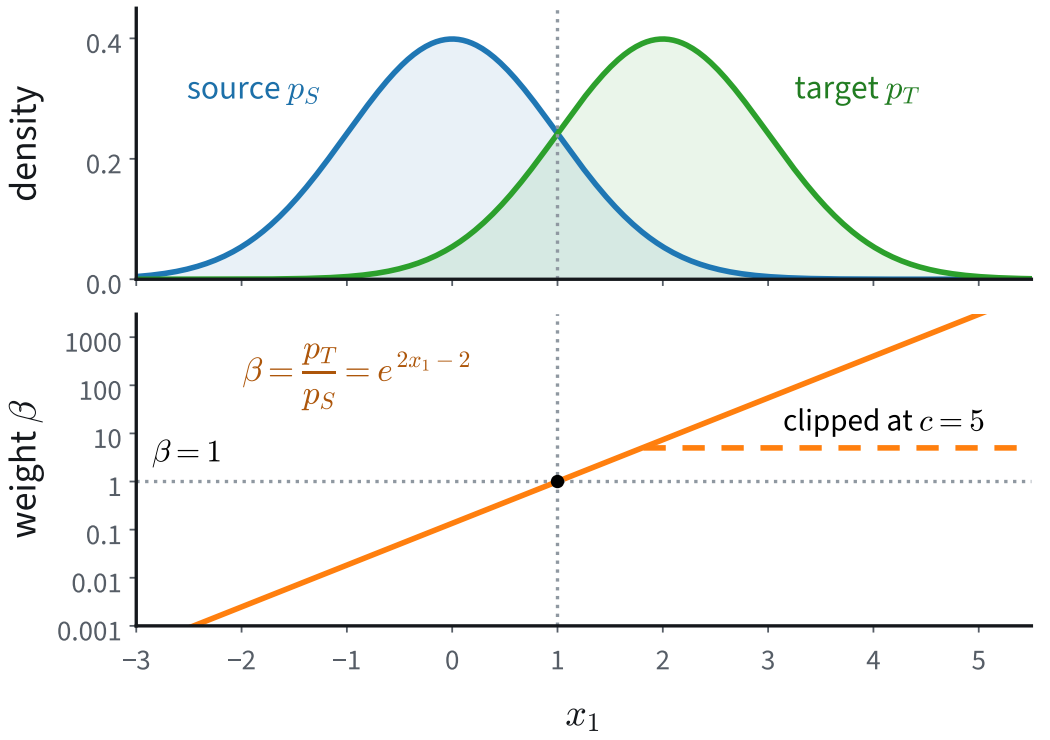


Figure 3.11: ソース密度 $p_S = \mathcal{N}(0, 1)$ とシフトしたターゲット密度 $p_T = \mathcal{N}(2, 1)$（上段）、および結果として得られる重要度重み $\beta(x_1) = p_T(x_1)/p_S(x_1) = e^{2x_1 - 2}$ をクリッピング閾値 $c = 5$ とともに対数スケールで示したもの（下段）。

3.6.2.2 ドメイン分類器による重みの推定

$\beta(\mathbf{x}) = p_T(\mathbf{x})/p_S(\mathbf{x})$ を推定するのに、高次元の密度 p_S と p_T を別々に推定せずに済ませるにはどうすればよいだろうか。n 個のソース入力（ドメインラベル $z = 0$ を付与）と m 個のラベルなしターゲット入力（ドメインラベル $z = 1$ を付与）を統合し、二値ロジスティック回帰の**ド**

メイン分類器（domain classifier）$P(z=1 \mid \mathbf{x}) = \sigma(h(\mathbf{x}))$（ロジットは $h(\mathbf{x})$）を訓練すればよい。ベイズの定理により、次式が成り立つ。

$$\frac{P(z=1 \mid \mathbf{x})}{P(z=0 \mid \mathbf{x})} = \frac{m}{n}\,\frac{p_T(\mathbf{x})}{p_S(\mathbf{x})} = \exp(h(\mathbf{x})),$$
$$\text{so} \quad \beta(\mathbf{x}) = \frac{n}{m}\exp(h(\mathbf{x})).$$

定数因子 n/m（目的関数を定数倍するだけで最小点を動かさない）を除けば、各ソース例を単に $\beta_i = \exp(h(\mathbf{x}^{(i)}))$（あるいは $\min(\exp(h(\mathbf{x}^{(i)})), c)$）で重み付けすればよい。

ドメイン分類器には 2 つ目の用途がある。補正すべきシフトが存在するかどうかを判定できることである。もし $p_T(\mathbf{x}) = p_S(\mathbf{x})$ であれば、ソース入力とターゲット入力を区別する手がかりは何もなく、訓練に使っていない統合データに対するドメイン分類器の性能は偶然レベルにとどまる（正解率は $1/2$、ただし $n = m$ のとき）。したがって、偶然レベルを明確に上回る正解率は、入力がシフトしたことの証拠となる。この検査は**分類器二標本検定**（classifier two-sample test）として知られている (Lopez-Paz and Oquab 2017)。ラベルシフトにはさらに手軽な検査法がある。運用中の分類器をラベルなしのターゲット入力に適用し、各クラスを予測する頻度をソース検証データ上での頻度と比較すればよい。シフトはまず気づかなければ補正できないため、運用システムではこれら両方の数値を監視すべきである。

3.6.2.3 2 次元の例

$p_S = \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$、$p_T = \mathcal{N}((2,0)^\top, \mathbf{I}_2)$ であり、真のラベルが $y = \mathbf{1}(x_2 > \frac{1}{2}x_1^2 - 1)$ で与えられる 2 次元問題において、ドメイン分類器による再重み付けが線形分類器をどのように救うかを見てみよう。真の放物線状の境界は上向きに湾曲しているため、$(0,0)$ 付近で最良の線形境界はほぼ水平となり、$(2,0)$ 付近では惨憺たる結果に終わる。そこで $n = 1{,}000$ 個のソース点と $1{,}000$ 個のターゲット点を生成し、ロジスティックドメイン分類器 $h(\mathbf{x})$ を当てはめて $\beta_i = \exp(h(\mathbf{x}^{(i)}))$ を推定したうえで、重みなし、重要度重み付き、およびクリッピング付き（$c = 5$）のロジスティック回帰を比較する。

```
rng = np.random.default_rng()
n = 1000
X_src = rng.normal(0, 1, (n, 2))            # source p_S: centered at (0, 0)
X_tgt = rng.normal(0, 1, (n, 2)) + [2, 0]   # target p_T: centered at (2, 0)
label = lambda X: (X[:, 1] > 0.5 * X[:, 0]**2 - 1).astype(float)
y_src, y_tgt = label(X_src), label(X_tgt)   # one labeling rule for both
```

ここで用いる学習の基本要素は、勾配降下法で訓練する重み付きロジスティック回帰ただ 1 つである。3.1.2.2 節より、ロジットに関する交差エントロピーの勾配は残差（residual）$\sigma(\mathbf{w}^\top\mathbf{x}+b)-y$ となるため、データ例 i を β_i で重み付けすることは、単にその残差に β_i を掛けることに帰着する。

```
def fit_logreg(X, y, weights=None, lr=0.1, steps=2000):
    """Logistic regression by gradient descent on the weighted mean loss."""
    w, b = np.zeros(X.shape[1]), 0.0
    v = np.ones(len(y)) if weights is None else weights / weights.mean()
    for _ in range(steps):
        r = v * (1 / (1 + np.exp(-(X @ w + b))) - y)   # weighted error
        w -= lr * X.T @ r / len(y)
```

```
        b -= lr * r.mean()
    return w, b
```

まず、X_src（ドメインラベル $z = 0$）と X_tgt（$z = 1$）を統合してドメイン分類器 (w_h, b_h) を fit_logreg により訓練し、学習されたロジット $h(\mathbf{x})$ を解析的なガウス対数密度比 $2x_1 - 2$ と比較したうえで、得られたソースの重要度重み beta とその有効標本サイズ ess を確認する。最後の数行では、新たなソース入力とターゲット入力をサンプリングし、ドメイン分類器がどの程度の割合で両者を識別できるかを測定する。

```
z = np.concatenate([np.zeros(n), np.ones(n)])      # domain label: 1 = target
w_h, b_h = fit_logreg(np.concatenate([X_src, X_tgt]), z)
beta = np.exp(X_src @ w_h + b_h)                   # weights of source examples
ess = lambda v: v.sum()**2 / (v**2).sum()          # effective sample size
print(f'learned h(x) = {w_h[0]:.2f} x1 {w_h[1]:+.2f} x2 {b_h:+.2f}'
      f'   (true log-ratio: 2 x1 - 2)')
print(f'largest weight: {beta.max():.0f}; effective sample size: '
      f'{ess(beta):.0f} of {n}, {ess(np.minimum(beta, 5)):.0f} if clipped at 5')
X_out = np.concatenate([rng.normal(0, 1, (n, 2)),          # fresh source and
                        rng.normal(0, 1, (n, 2)) + [2, 0]])  # target inputs
print(f'domain classifier accuracy on held-out inputs: '
      f'{((X_out @ w_h + b_h > 0) == (z == 1)).mean():.2f} (chance: 0.50)')
```

```
learned h(x) = 2.03 x1 -0.10 x2 -2.01   (true log-ratio: 2 x1 - 2)
largest weight: 91; effective sample size: 54 of 1000, 216 if clipped at 5
domain classifier accuracy on held-out inputs: 0.83 (chance: 0.50)
```

新たな入力に対して、ドメイン分類器は約 0.84 の正解率でソースとターゲットを識別しており、偶然レベルの 0.50 を大きく上回るため、このシフトの検出は容易である。この設定では、これ以上高い性能を達成できる分類器は存在しない。2 つのガウス分布は x_1 の平均のみが異なり、最良の判定規則は $x_1 > 1$ であって、その正答確率は $\Phi(1) \approx 0.84$（ただし Φ は標準正規分布関数）となるからである。

次に、fit_logreg をラベル付きソースデータ (X_src, y_src) に対して 3 通りの重み付け方式（重みなし 'unweighted'、生の重要度重み 'weighted'、クリッピング重み 'clipped'（上限 5））で訓練し、各分類器の X_tgt および X_src 上での正解率を評価する。

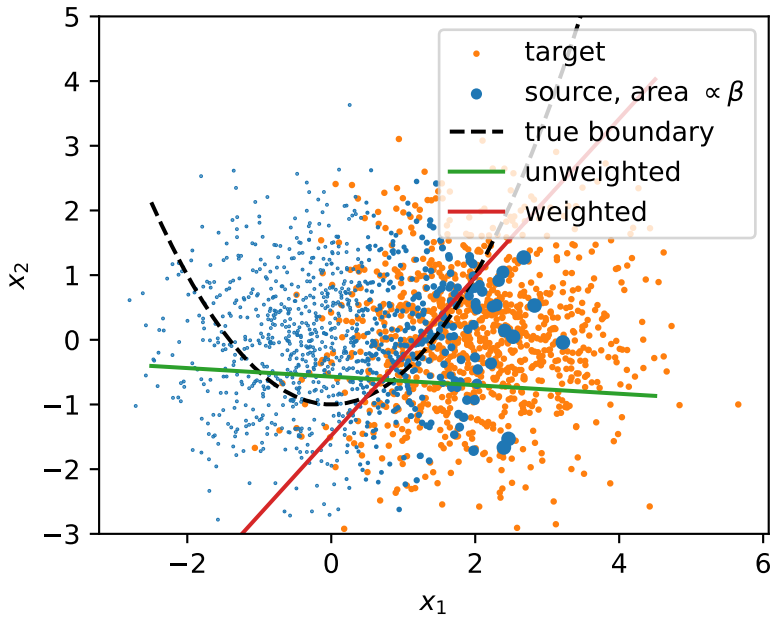
```
acc = lambda wb, X, y: ((X @ wb[0] + wb[1] > 0) == (y == 1)).mean()
models = {name: fit_logreg(X_src, y_src, v) for name, v in [
    ('unweighted', None), ('weighted', beta), ('clipped', np.minimum(beta, 5))]}
for name, wb in models.items():
    print(f'{name:10s}  target accuracy {acc(wb, X_tgt, y_tgt):.3f}'
          f'   source accuracy {acc(wb, X_src, y_src):.3f}')
print(f'always predicting the most common target class: '
      f'{max(y_tgt.mean(), 1 - y_tgt.mean()):.3f}')
```

```
unweighted  target accuracy 0.494   source accuracy 0.853
weighted    target accuracy 0.947   source accuracy 0.799
clipped     target accuracy 0.953   source accuracy 0.807
always predicting the most common target class: 0.716
```

再重み付けがこれほど劇的に効く理由を可視化するため、ターゲットとソースの点群（各ソース点のマーカー面積を beta に比例させる）を、真の放物線 $x_2 = \frac{1}{2}x_1^2 - 1$ および 'unweighted' と

'weighted' の線形決定境界とともにプロットする。

```
x1 = np.linspace(-2.5, 4.5, 50)
d2l.set_figsize((4.5, 3.5))
d2l.plt.scatter(*X_tgt.T, s=2, c='C1', label='target')
d2l.plt.scatter(*X_src.T, s=np.clip(beta, 0.2, 20), c='C0',
                label=r'source, area $\propto \beta$')
d2l.plt.plot(x1, 0.5 * x1**2 - 1, 'k--', label='true boundary')
for name, fmt in [('unweighted', 'C2'), ('weighted', 'C3')]:
    (w1, w2), b = models[name]
    d2l.plt.plot(x1, -(w1 * x1 + b) / w2, fmt, label=name)
d2l.plt.xlabel('$x_1$'), d2l.plt.ylabel('$x_2$'), d2l.plt.ylim(-3, 5)
d2l.plt.legend();
```



結果を見てみよう。シフトしたターゲット点群に対して、重みなしの訓練ではスコアが $\sim 50\%$〜60% にとどまる（多数派クラスを常に予測する場合の $\sim 70\%$ より低い）のに対し、重要度による再重み付けは線形決定境界を放物線の右側の枝に沿うように傾け、ターゲットラベル**ゼロ**でターゲット正解率を $\sim 95\%$ まで引き上げている。

この結果から、共変量シフトのもとでは常に再重み付けすべきだと結論づけるのは誤りである。ここで再重み付けが有効だったのは、直線が放物線に沿えないからである。すなわち、モデルが誤特定されており、どこで高い精度を出すかを選ばざるを得ないためである。ソースデータが存在するすべての領域で $p(y \mid \mathbf{x})$ に適合できるほど柔軟なモデルは何も得るところがない。その場合、重み付きの目的関数と重みなしの目的関数は同じ最小点を共有するからである (Shimodaira 2000)。深層ネットワークはこの状況に近い。ひとたび低い訓練損失（training loss）まで訓練されると、重要度重みの効果は薄れていき (Byrd and Lipton 2019)、実世界のシフトを集めた WILDS ベンチマークでも、分布シフト向けに設計された手法が通常の経験リスク最小化を上回ることはまれである (Koh et al. 2021)。実務において共変量シフトへの対処は、主により幅広いデータで訓練することと、少数のラベル付きターゲット例でファインチューニングすることによってなされる。

3.6.3 ラベルシフトの補正

ラベルシフト（$p_T(\mathbf{x} \mid y) = p_S(\mathbf{x} \mid y)$ が保たれる一方でクラス比率 $p_T(y) \neq p_S(y)$ がシフトする状況）のもとでは、まったく同じ重要度重み付けの恒等式により、クラス y の各ソース例をクラス事前分布の比 $\beta(y) = p_T(y)/p_S(y)$ で重み付けすることになる。$p_S(y)$ は訓練ラベルから既知であるため、唯一の課題は**ラベルなしの**ターゲットデータからターゲットのクラス比率 $\mathbf{p}_T \in \mathbb{R}^q$ を推定することである。

3.6.3.1 混同行列によるターゲット比率の推定

$\mathbf{C} \in \mathbb{R}^{q \times q}$ を、3.3 節で扱ったソース分類器の列正規化された混同行列とし、その各成分 $C_{ij} = P_S(\hat{c} = i \mid y = j)$ をラベル付きソース検証データから推定する。$\hat{c}$ は $\mathbf{x}$ のみの関数であり、かつラベルシフトのもとでは $p(\mathbf{x} \mid y)$ が不変であるため、ターゲットドメイン上でも $P_T(\hat{c} = i \mid y = j) = C_{ij}$ が成り立つ。全確率の法則により、ラベルなしターゲット入力における予測クラス頻度のベクトル $\mu \in \mathbb{R}^q$ は次式を満たす (Saerens et al. 2002; Lipton et al. 2018)。

$$\mu_i = \sum_{j=1}^{q} P(\hat{c} = i \mid y = j)\, p_T(y = j) = \sum_{j=1}^{q} C_{ij}\, p_T(y = j), \quad \text{or} \quad \mathbf{C}\,\mathbf{p}_T = \mu. \tag{3.34}$$

$\mathbf{C}$ が正則である限り（分類器が各クラスの半分超を正しく分類していれば、すなわち $C_{jj} > 1/2$ であれば保証される。演習問題 3 を参照）、$q \times q$ の連立一次方程式 $\hat{\mathbf{p}}_T = \mathbf{C}^{-1}\mu$ を解くことで、ターゲットラベルを 1 つも使わずに未知のターゲットクラス比率を復元できる。さらに、ソース分類器の出力確率がソースの事後分布（posterior distribution）$p_S(y \mid \mathbf{x})$ をよく近似しているならば（これは 3.4 節における最上位クラスの較正（calibration）よりも強い条件である）、各クラスのロジットに対数事前分布のシフト $\log(\hat{p}_T(y)/p_S(y))$ を加えるだけで、ベイズ則により再訓練なしで直ちに予測をターゲットドメインへ適応させられる。

Fashion-MNIST を用いて、混同行列の逆行列計算とベイズロジット調整を試してみよう。まず、クラス比率の均等な訓練分割で `d2l.SoftmaxRegression` を訓練し、検証データに対するロジット `O`、ラベル `y`、および予測クラス `pred = O.argmax(axis=1)` を収集する。

```
data = d2l.FashionMNIST(batch_size=256)
model = d2l.SoftmaxRegression(num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.board.display = False
trainer.fit(model, data)
O, y = [], []
for X, Y in data.val_dataloader():
    O.append(d2l.numpy(model(X)))
    y.append(d2l.numpy(Y))
O, y = np.concatenate(O), np.concatenate(y)
pred = O.argmax(axis=1)
```

次に、10,000 個の検証例をランダムに分割し、5,000 枚のソース検証セット `src`（列正規化された 10×10 の混同行列 `C` の推定に使用）とターゲットプール `pool` に分ける。このプールから、衣類全体の 50% をシャツが占める偏ったターゲットセット `tgt`（シャツ 400 枚、他の各クラスは 45 枚ずつ）を抽出し、`p_hat = np.linalg.solve(C, mu)` をラベルなしターゲットの予測頻度 `mu` から解く。

```
perm = rng.permutation(len(y))
src, pool = perm[:5000], perm[5000:]  # Source validation set; target pool
C = np.zeros((10, 10))
np.add.at(C, (pred[src], y[src]), 1)  # The confusion matrix of the source
C /= C.sum(axis=0, keepdims=True)     # Each column sums to 1
counts = np.full(10, 45)
counts[6] = 400                       # Half of the target images are shirts
p_T = counts / counts.sum()
tgt = np.concatenate([rng.choice(pool[y[pool] == j], counts[j], replace=False)
                      for j in range(10)])
mu = np.bincount(pred[tgt], minlength=10) / len(tgt)  # Needs no labels
p_hat = np.linalg.solve(C, mu)
for j, name in enumerate(data.text_labels(range(10))):
    print(f'{name:10s} true {p_T[j]:.2f}, predicted {mu[j]:.3f}, '
          f'estimated {p_hat[j]:.3f}')
```

```
T-shirt    true 0.06, predicted 0.153, estimated 0.114
trouser    true 0.06, predicted 0.052, estimated 0.052
pullover   true 0.06, predicted 0.118, estimated 0.048
dress      true 0.06, predicted 0.088, estimated 0.069
coat       true 0.06, predicted 0.129, estimated 0.077
sandal     true 0.06, predicted 0.055, estimated 0.057
shirt      true 0.50, predicted 0.215, estimated 0.411
sneaker    true 0.06, predicted 0.065, estimated 0.062
bag        true 0.06, predicted 0.076, estimated 0.062
ankle boot true 0.06, predicted 0.050, estimated 0.049
```

最後に、対数事前分布のシフト `log_w = np.log(np.maximum(p_hat, 1e-6) / 0.1)` をターゲットのロジット `O[tgt]` に加算することで分類器の予測を推定されたターゲット分布へ適応させ、調整前後のターゲット正解率を比較する。

```
log_w = np.log(np.maximum(p_hat, 1e-6) / 0.1)  # log p_T(y) - log p_S(y)
accuracy = lambda O: (O.argmax(axis=1) == y[tgt]).mean()
print(f'target accuracy: {accuracy(O[tgt]):.3f} as trained, '
      f'{accuracy(O[tgt] + log_w):.3f} adapted to the estimated proportions')
```

```
target accuracy: 0.666 as trained, 0.799 adapted to the estimated proportions
```

元の分類器がターゲット画像のうち「シャツ」と予測するのは $\sim 20\%$〜40% にすぎないにもかかわらず（残りは似た形状のトップスに紛れ込む）、$\mathbf{C}^{-1}\mu$ はその混同を解きほぐして真の値 0.50 に近いシャツ比率を復元し（通常は小数第 2 位の数単位以内、まれに 1 割近くずれる程度）、ロジットに $\log(\hat{\mathbf{p}}_T/0.1)$ を加えることで、ターゲットラベルも勾配ステップも一切要さずにターゲット正解率を約 5〜15 パーセントポイント引き上げる。

3.6.4 概念シフトとフィードバック

概念シフトによって $p(y \mid \mathbf{x})$ そのものが変化すると、過去のラベルは陳腐化し、古いデータにどのような重要度重み付けを施しても新しいラベル付け規則を学習させることはできない。新たにラベル付けされた少量のデータ例による継続的な監視と、定期的なファインチューニングや再訓練が不可欠である。

モデルの予測が**将来の訓練データを作り変える行動**を引き起こす場合（**遂行的予測**（performative prediction）および**フィードバックループ**）には、さらなる注意が必要である。予測的警察活動

（Figure 3.12）において、犯罪予測値の高い地区へ重点的にパトロールを派遣すると、その地区でより多くの事案が記録されるようになる。こうしてバイアスのかかった逮捕記録を次の訓練ラウンドへ戻すと、予測が自己強化されて暴走ループに陥る。このようなループを断ち切ったり監査したりするには、ランダム化された探索（exploration）や独立したデータ収集が必要となり、受動的な教師あり学習から**強化学習**（reinforcement learning）（第 14 章）の領域へと踏み込むことになる。

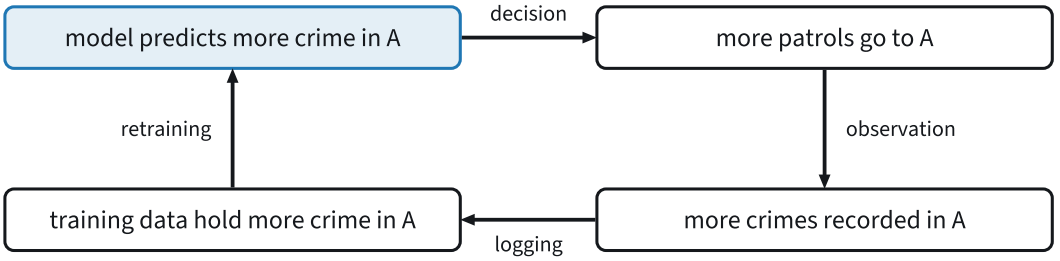


Figure 3.12: 予測的警察活動における暴走フィードバックループ：モデルの予測がパトロール配置を左右し、パトロールの有無が事案の観測場所を決定し、記録された事案が次の訓練データセットとなる。

3.6.5 まとめ

ラベルなしの運用環境への適応は、ソース $p_S(\mathbf{x}, y)$ とターゲット $p_T(\mathbf{x}, y)$ を結ぶ構造的な不変性なしには不可能であり、その不変性は共変量シフト（$p(y \mid \mathbf{x})$ が固定され $p(\mathbf{x})$ がドリフトする）、ラベルシフト（$p(\mathbf{x} \mid y)$ が固定され $p(y)$ がドリフトする）、あるいは概念シフト（$p(y \mid \mathbf{x})$ がドリフトする）として整理される。サポートが重なる共変量シフトのもとでは、ソース入力とラベルなしターゲット入力を識別するロジスティックドメイン分類器で推定した $\beta(\mathbf{x}) = p_T(\mathbf{x})/p_S(\mathbf{x}) \propto \exp(h(\mathbf{x}))$ によってソースの損失を重み付けすることで、誤特定されたモデルをターゲットリスクに整合させられる。ラベルシフトのもとでは、ソースの混同行列の逆行列を用いた $\hat{\mathbf{p}}_T = \mathbf{C}^{-1}\mu$ によりラベルなしターゲット予測からターゲットのクラス比率を復元でき、較正されたロジットに $\log(\hat{\mathbf{p}}_T/\mathbf{p}_S)$ を加えることで直接適応できる。概念シフトや意思決定が駆動するフィードバックループによって環境そのものが変化する場合には、受動的な再重み付けだけでは不十分であり、運用システムには新たなターゲットラベルと独立した探索が必要となる。

3.6.6 演習問題

1. **サポートの欠如。** $p_S(\mathbf{x}) = 0$ が入力のある領域 $\mathcal{A}$（ターゲットのもとで確率 $\pi > 0$ を持つ）上で成り立ち、それ以外のすべての点では $p_S(\mathbf{x}) > 0$ であるとする。$\beta(\mathbf{x}) = p_T(\mathbf{x})/p_S(\mathbf{x})$ を $\mathcal{A}$ の外側で定義する。
   1. 重み付きソースリスク $\mathbb{E}_{(\mathbf{x},y)\sim p_S}[\beta(\mathbf{x})\, l(f(\mathbf{x}), y)]$ が、ターゲットリスクから $\mathcal{A}$ の寄与を取り除いたものに等しいことを示せ。また、欠落した項を π と条件付き期待値の積として表せ。
   2. ソース例に対する重みをどのように選んでも、この欠落した項を復元できない理由を説明せよ。さらに、$\mathcal{A}$ の外側では一致し、同じ重み付きソースリスクを持ちながらターゲットリスクが異なる 2 つのモデルを構成せよ。
2. [code] **再重み付けが失敗するとき。** 3.6.2 節のコード（`fit_logreg`、`X_src`、`X_tgt`、`label`）ではターゲットの平均を $(2, 0)$ に置いている。ターゲットの平均 $(s, 0)$ を

$s \in \{0, 0.5, 1, 2, 3, 4, 6\}$ にわたって変化させよ。ただし、$n = 1000$ とラベル付け規則は固定したままとする。

1. 各 s について、ホールドアウトされた統合データに対するドメイン分類器 h の正解率を報告し、s の増大に伴うその上昇をシフトの検出しやすさの観点から解釈せよ。
2. 各 s について、重みなし、重み付き、およびクリッピング付き（$c = 5$）の各モデルと、最も頻度の高いターゲットクラスを常に予測する規則のターゲット正解率を記録せよ。また、大きな s においてはターゲット正解率だけでは再重み付けが機能したかどうかを判断できない理由を説明せよ。

3. **ラベルシフトの連立方程式。** $\mathbf{C}$ を Equation 3.34 の列正規化された混同行列とする。

1. $\mathbf{C}$ は、条件 $C_{jj} > \frac{1}{2}$ がすべてのクラス j で成り立つ限り正則であることを示せ。ヒント：$\mathbf{v}^\top \mathbf{C} = \mathbf{0}$ がある $\mathbf{v} \neq \mathbf{0}$ に対して成り立つと仮定し、列 j として $|v_j|$ が最大となるものに注目して、その列の和が 1 であることを用いるとよい。
2. $\mathbf{C}$ が特異となる分類器の例を 1 つ挙げ、その場合でも μ から $\mathbf{p}_T$ について何がわかるかを述べよ。
3. 2 クラスの場合、誤差 δ が陽性予測の割合 μ_1 に生じると、推定される陽性割合が $\delta / (C_{11} - C_{10})$ だけ変化することを示せ。標準誤差 $\sqrt{\mu_1(1-\mu_1)/m}$（m 個のターゲット入力から推定した割合の標準誤差）を用いて、スクリーニングの例において罹患者割合の推定値の標準誤差を 0.01 にするために必要なターゲット入力数を求めよ。同じ集団に対して、より精度の低い検査（$C_{11} = 0.6$、$C_{10} = 0.4$）を用いた場合についても同様に求めよ。

4. [code] **学習された重みと真の重み。**●演習問題 2 におけるターゲット平均 $(s, 0)$ のスイープを再利用せよ。

1. 真の対数密度比 $\log(p_T(\mathbf{x})/p_S(\mathbf{x}))$ を s の関数として導出し、学習された h と各 s において比較せよ。また、大きな s においてドメイン分類器が高い正解率を達成しながら、h が真の対数密度比から大きく乖離し得る理由を説明せよ。
2. ソース分布のもとで真の重みが $\mathbb{E}[\beta(\mathbf{x})] = 1$ および $\mathbb{E}[\beta(\mathbf{x})^2] = e^{s^2}$ を満たし、したがって大きな n に対して有効標本サイズが約 $n e^{-s^2}$ となることを示せ。この値を $n = 1000$、$s = 1, 2, 3$ について計算せよ。
3. 各 s について、実際のソースサンプル上での真の重みと学習された重みの有効標本サイズを計算せよ。$s = 2$ においてサンプルから計算した値が通常 $1000\, e^{-4} \approx 18$ よりはるかに大きくなる理由と、サンプルのみから再重み付け推定の信頼性を判断するうえでそれが何を意味するかを説明せよ。$c = 5$ でのクリッピングはこのデータ不足を解消するか。

5. **ラベルなしデータからのシフト仮定の識別。**●二値のソースドメインにおいて $p_S(y = 1) = p_S(y = 0) = \frac{1}{2}$ であり、スカラー入力が $x \mid y = 1 \sim \mathcal{N}(1, 1)$ および $x \mid y = 0 \sim \mathcal{N}(-1, 1)$ に従うとする。このとき $p_S(y = 1 \mid x) = 1/(1 + \exp(-2x))$ となる。ラベルなしターゲット入力が次の混合分布に従うとする。

$$p_T(x) = 0.8\, \mathcal{N}(x; 1, 1) + 0.2\, \mathcal{N}(x; -1, 1).$$

1. ラベルシフトを仮定せよ。$p_T(y = 1)$ と、ベイズ最適なターゲット分類器の x に関する閾値を求めよ。
2. 代わりに同じ $p_T(x)$ のもとで共変量シフトを仮定せよ。$p_T(y = 1)$ を数値的に計算し、ベイズ最適な閾値を求めよ。
3. 以上の結果を用いて、ラベルなしターゲットサンプルのいかなる統計量によっても、本節の 2 つの補正手法のどちらを適用すべきか決定できない理由を説明せよ。

Chapter 4

多層パーセプトロン

本章では、最初の**深層**ニューラルネットワークを導入する。第 3 章では、ソフトマックス回帰が Fashion-MNIST の衣類画像を構成する 784 ピクセルそれぞれに固定の重みを割り当ててその根拠を合計していたため、周囲に何があるかにかかわらず、すべてのピクセルがクラスに対して同じ票を投じていた。しかし、明るいピクセルは袖、ハンドバッグのストラップ、襟のいずれにも属しうるため、そのどれであるかは近傍のピクセルを見なければわからない。4 つの二値点においてさえ、単一の線形層では 2 ビットの排他的論理和（exclusive-OR、XOR）を計算できない。これは Marvin Minsky と Seymour Papert が 1969 年に、Frank Rosenblatt の単層**パーセプトロン**について指摘した限界である。

多層パーセプトロン（multilayer perceptron、MLP）は、それぞれアフィン変換と非線形な活性化関数を組み合わせた全結合の**隠れ層**（hidden layer）を出力層の前に 1 つ以上積み重ね、すべての層を**誤差逆伝播法**（backpropagation）で同時に訓練することによって、この線形の壁を乗り越える (Rumelhart et al. 1986)。まず、なぜ線形モデルが XOR のような特徴量の相互作用を扱えないのか、隠れ層が深さに応じて入力空間をどのように指数関数的に多くの線形領域へと折り畳むのか、そして代表的な活性化関数（ReLU、GELU、シグモイド、tanh）が微分においてどのように振る舞うのかを示す（4.1 節）。続いて、256 ユニットの ReLU MLP を Fashion-MNIST 上でゼロから、また簡潔に実装すると（4.2 節）、ソフトマックス回帰の分類誤差が約 5 分の 1 削減され（正解率 $\sim 87\%$ 対 $\sim 84\%$）、GPU でのエポック時間はほぼ同じまま、ぼやけた衣類全体のテンプレートが局所的なコントラスト検出器に置き換わる。

隠れ層を追加すると、深層学習における 3 つの根本的な問いも生じる。これらが本章の残りの節の主題である。第 1 に、4.3 節では MLP を通じたリバースモード微分の 3 つの行列微分則を導出し、自動微分（automatic differentiation）と機械精度まで一致することを検証したうえで、訓練時の活性化メモリと FLOP の所要量を明らかにする。第 2 に、4.4 節では K 個の層にわたる信号と勾配の分散を追跡し、10 層の MLP における勾配消失と勾配爆発を防ぐ **Xavier** 初期化と **He** 初期化を導出するとともに、隠れユニットの対称性を破るためになぜランダムな重みが必要なのかを示す。第 3 に、4.5 節と 4.6 節では、大幅に過剰パラメータ化されたネットワークがなぜ汎化（generalization）するのかを検討し、ランダムラベルの記憶、二重降下（double descent）、勾配降下法の暗黙的バイアス（implicit bias）、早期停止、**ドロップアウト**による乗法的ノイズ正則化を扱う。そして最後に、4.7 節でこれら一連の手法を Kaggle の Ames Housing 回帰ベンチマークに適用する。

4.1 多層パーセプトロン

第 3 章では、ソフトマックス回帰は入力ピクセルのアフィン重み付き和 $o_j = \sum_{i=1}^{d} x_i w_{ij} + b_j$ を取ることで、衣類の各カテゴリにスコアを付けた。この線形形式には、2 つの厳しい制約が組み込まれている。

1. **単調性：**特徴量 x_i を増やすと、o_j は常に同じ方向へ動く（$w_{ij} > 0$ なら増加し、$w_{ij} < 0$ なら減少する）。しかし、現実の多くの関係は非単調である。たとえば人間の健康リスクは、体温が 37°C から**どちらの**方向に外れても上昇する。
2. **特徴量の相互作用の欠如：**x_i を 1 単位増やすと、他のすべての座標の値にかかわらず、o_j は w_{ij} だけ変化する。しかし画像においては、位置 $(13, 17)$ にある明るいピクセルが意味するものは、周囲のピクセルが襟、袖、ハンドバッグのストラップのどれを形作っているかによって全く異なる。

Figure 4.1 にこの体温の例をプロットした。最良の線形スコアは温度とともに一貫して上昇するため、低体温症の患者を誰よりも安全だと評価してしまう。

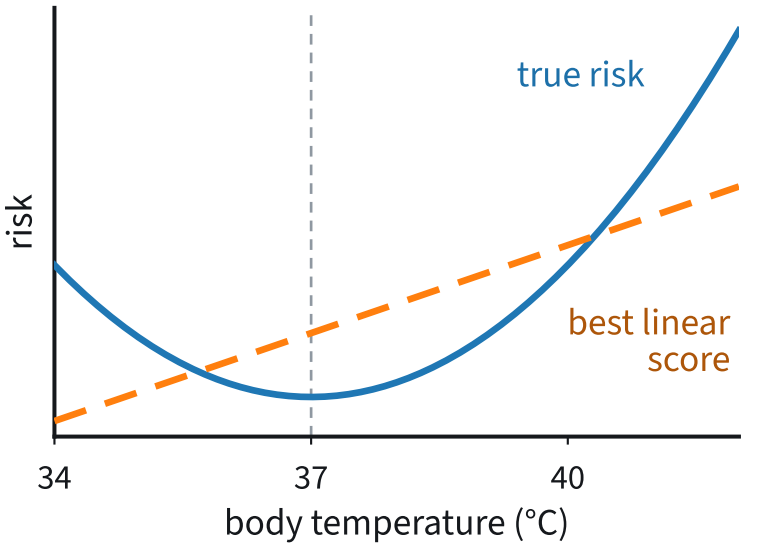


Figure 4.1: 体温に対する健康リスク。リスクは 37°C 付近で最も低く、その両側で上昇するが、線形スコアは単調に上昇または下降することしかできない。破線はプロット範囲における最小二乗法による線形当てはめである。

多層パーセプトロン（multilayer perceptron、MLP）は、生の入力を学習された中間表現へと変換する 1 つ以上の**隠れ層**（hidden layer）を挿入することで、これら両方の限界を克服する。各隠れ層ではアフィン写像に続けて非線形な**活性化関数**（activation function）を適用し、その後に線形の出力層が最終的な予測を下す。本節では、いかなる線形モデルも 4 点の XOR 問題を解けないことを証明し、2 個の ReLU 隠れユニットで XOR を明示的に解き、浅いネットワークと深いネットワークがどのような関数を表現できるかを調べ、現代の深層学習で広く用いられている活性化関数を比較する。

4.1.1 線形モデルの限界：XOR 問題

特徴量の相互作用を最も鮮明に示す例は、わずか 4 つのデータ点しか使わない。単位正方形 $\{0,1\}^2$ の各頂点に**排他的論理和**（exclusive-OR、XOR）でラベルを付けたものであり、$y=1$ となるのは**ちょうど 1 つ**の座標が 1 に等しいとき、かつそのときに限る。

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

この相互作用に注目しよう。x_1 を 0 から 1 に反転させると、$x_2=0$ のときには y が増加するが、$x_2=1$ のときには y が減少する。

命題：任意の $w_1, w_2, b \in \mathbb{R}$ に対し、線形決定規則「$w_1 x_1 + w_2 x_2 + b > 0$ ならばクラス 1 と予測する」は、XOR の 4 点のうち少なくとも 1 点を誤分類する。

証明：背理法のため、ある w_1, w_2, b が 4 つの頂点すべてを正しく分類すると仮定する。クラス 1 の 2 点 $(0,1)$ と $(1,0)$ からは $w_2 + b > 0$ および $w_1 + b > 0$ が必要となり、辺々を加えると $w_1 + w_2 + 2b > 0$ を得る。一方、クラス 0 の 2 点 $(0,0)$ と $(1,1)$ からは $b \le 0$ および $w_1 + w_2 + b \le 0$ が必要となり、辺々を加えると $w_1 + w_2 + 2b \le 0$ を得るが、これは直ちに矛盾する。■

幾何学的には、単位正方形の 2 本の対角線はどちらも中点 $(1/2, 1/2)$ を共有している（Figure 4.2）。この中点において任意のアフィン関数はどちらの対角線の両端点の値の平均に等しくなるため、いかなる直線も 2 本の対角線を分離することはできない。

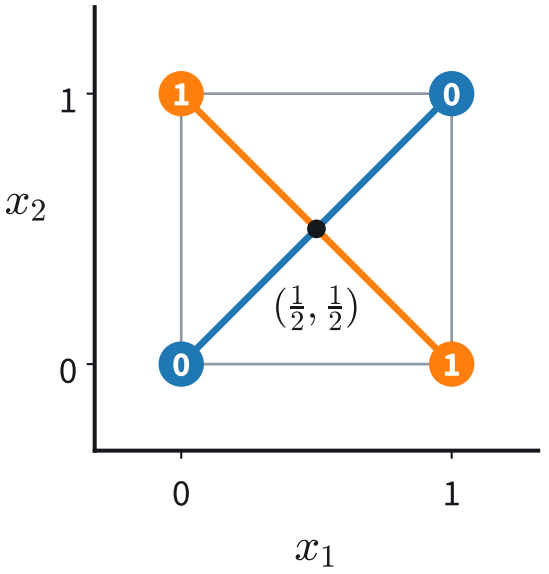


Figure 4.2: クラスでラベル付けされた XOR の 4 点。各対角線は同一クラスの 2 点を結び、2 本の対角線は $(1/2, 1/2)$ で交差するため、いかなる直線もクラス間を分離できない。

4.1.2 隠れ層

4.1.2.1 モデル

線形性から脱却するため、MLP は入力特徴量と出力層の間に 1 つ以上の**隠れ層**を挿入する（Figure 4.3）。ミニバッチの計画行列 $\mathbf{X} \in \mathbb{R}^{n \times d}$（データ例数 n、入力特徴量数 d）に対し、幅 h の単一の隠れ層は重み $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ とバイアス $\mathbf{b}^{(1)} \in \mathbb{R}^{h}$ を持ち、出力層は重み $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ とバイアス $\mathbf{b}^{(2)} \in \mathbb{R}^{q}$ を持つ。2 層（隠れ層 1 層）の MLP は次式を計算する。

$$\begin{aligned} \mathbf{H} &= \phi\left(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right), \\ \mathbf{O} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \end{aligned} \tag{4.1}$$

ここで、バイアスは各行にブロードキャストされ、非線形な**活性化関数** ϕ が**活性化前入力**（pre-activation）$\mathbf{Z} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$ に要素ごとに作用して、隠れ層の**活性化値**（activation）$\mathbf{H} \in \mathbb{R}^{n \times h}$ を生成する。パラメータ数が $(d+1)h + (h+1)q$ であることに注目しよう。Fashion-MNIST（$d = 784, q = 10$）では、$h = 256$ ユニットの隠れ層を持つモデルのパラメータ数は $785 \times 256 + 257 \times 10 = 203{,}530$ となり、ソフトマックス回帰の 7,850 の 26 倍に上る。

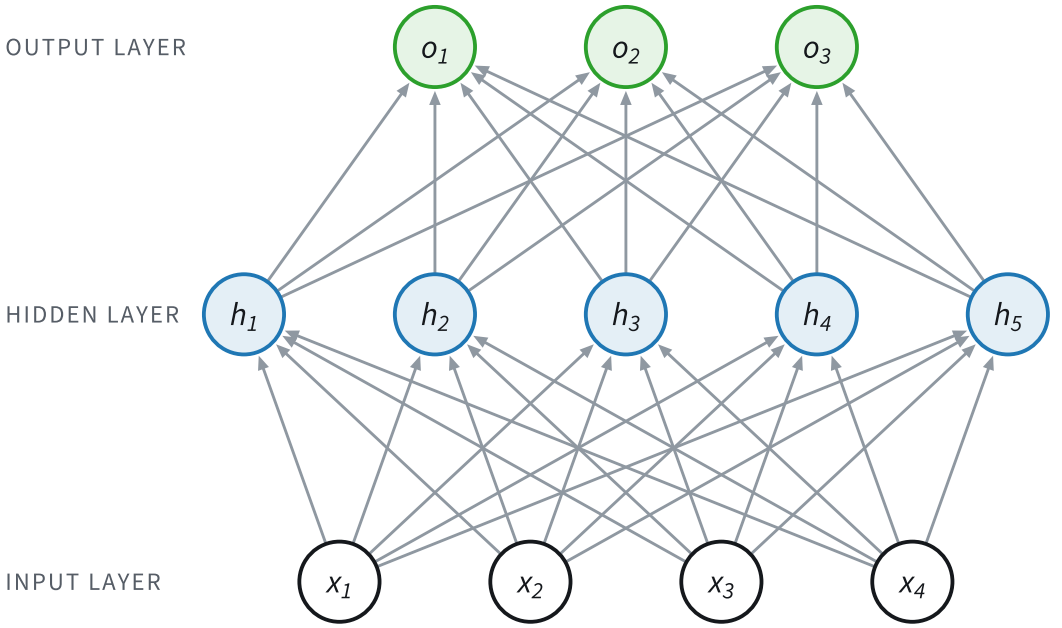


Figure 4.3: 4 個の入力、5 個の隠れユニットからなる 1 つの隠れ層、および 3 個の出力を持つ MLP。パラメータを持つ両方の層は全結合層である。

4.1.2.2 非線形性の役割

なぜ非線形な活性化関数 ϕ が不可欠なのだろうか。仮に ϕ を省略し（$\phi(z) = z$ とおき）、$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$ としたとしよう。この $\mathbf{H}$ を出力の式に代入すると、2 つの層は単一のアフィン変換（affine transformation）へと縮退してしまう。

$$\mathbf{O} = \left(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}\right)\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\underbrace{\mathbf{W}^{(1)}\mathbf{W}^{(2)}}_{\mathbf{W}} + \underbrace{\mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}}_{\mathbf{b}}.$$

非線形性（nonlinearity）がなければ、全結合層（fully connected layer）を何層積み重ねても、表現できる関数の族は単一の線形層から決して広がらない（しかも、隠れ層の幅が $\min(d, q)$ より狭い場合、実際には $\mathbf{W}$ が低ランクに制限される。演習問題 1 を参照）。そこで、次式の**整流線形ユニット**（rectified linear unit、ReLU）のような要素ごとの非線形性を挿入する。

$$\mathrm{ReLU}(z) = \max(z, 0), \tag{4.2}$$

これにより、この代数的な縮退を防ぐことができる。

4.1.2.3 2 個の ReLU ユニットによる XOR の解法

ReLU 活性化関数を用いれば、わずか $h = 2$ ユニットの隠れ層で XOR を鮮やかに解くことができる (Goodfellow et al. 2016)。パラメータを次のように設定しよう。

$$\mathbf{W}^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{b}^{(1)} = (0, -1), \quad \mathbf{w}^{(2)} = (1, -2), \quad b^{(2)} = 0. \tag{4.3}$$

両方の隠れユニットは座標の和 $s = x_1 + x_2$ に依存する。ユニット 1 は $h_1 = \mathrm{ReLU}(s) = s$ を計算する一方、ユニット 2 は $h_2 = \mathrm{ReLU}(s - 1)$ を計算し、$s = 2$ のとき（点 $(1, 1)$）に**のみ**オンになるため、$o = h_1 - 2h_2$ が 2 を引いて出力を再び 0 へと折り返す。

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
W1, b1 = np.array([[1, 1], [1, 1]]), np.array([0, -1])
w2, b2 = np.array([1, -2]), 0
H = np.maximum(X @ W1 + b1, 0)  # hidden layer with ReLU
o = H @ w2 + b2  # output layer
for x, h, out in zip(X, H, o):
    print(f'x = {x}, h = {h}, o = {out}')
```

```
x = [0 0], h = [0 0], o = 0
x = [0 1], h = [1 0], o = 1
x = [1 0], h = [1 0], o = 1
x = [1 1], h = [2 1], o = 0
```

Figure 4.4 の幾何学的な構造を見てみよう。隠れ層はクラス 1 の**両方の**頂点 $(0, 1)$ と $(1, 0)$ を全く同じ隠れ空間上の点 $(1, 0)$ に写像するため、隠れ空間（右）における直線 $h_1 - 2h_2 = 1/2$ によって、入力空間（左）における半空間ではない対角帯 $1/2 < x_1 + x_2 < 3/2$ を切り出すことができる。

4.1.2.4 より深いネットワーク

より一般に、K 層の MLP は $K - 1$ 個の隠れ層を $\mathbf{H}^{(0)} = \mathbf{X}$ の上に次のように積み重ねる。

$$\begin{aligned} \mathbf{H}^{(k)} &= \phi\left(\mathbf{H}^{(k-1)}\mathbf{W}^{(k)} + \mathbf{b}^{(k)}\right) \quad (k = 1, \dots, K-1), \\ \mathbf{O} &= \mathbf{H}^{(K-1)}\mathbf{W}^{(K)} + \mathbf{b}^{(K)}, \end{aligned}$$

こうして、生の入力からクラスのロジットに至る表現の合成的な階層構造を築き上げる。

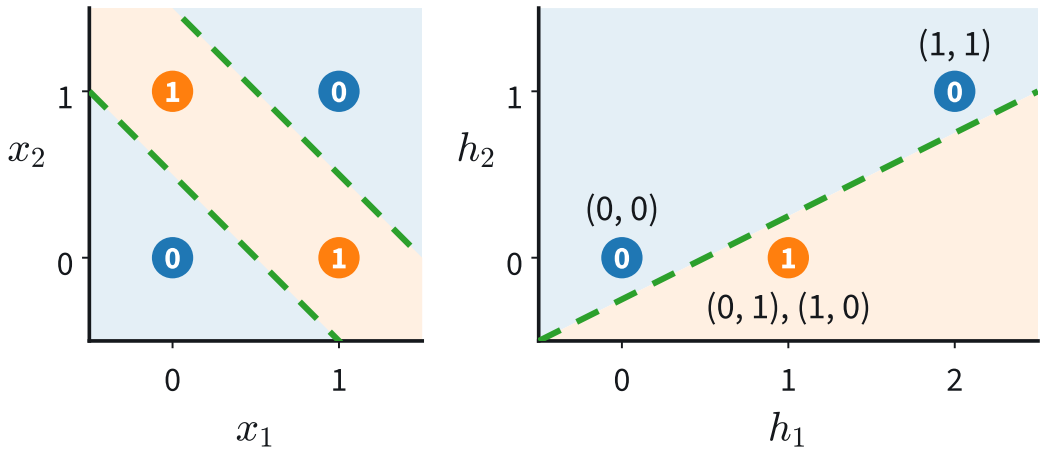


Figure 4.4: 2 個の ReLU 隠れユニットによる XOR の解法。左：入力空間 (x_1, x_2) では、ネットワークは対角帯 $1/2 < x_1 + x_2 < 3/2$ の内側でクラス 1 と予測する。右：隠れ空間 (h_1, h_2) では、クラス 1 の 2 点が $(1, 0)$ に縮退し、1 本の直線 $h_1 - 2h_2 = 1/2$ がクラス間を分離する。

4.1.3 表現力

4.1.3.1 万能近似

単一の隠れ層（$K = 2$）で計算できる関数のクラスはどれほど豊かなのだろうか。実直線上（$d = 1$）では、各 ReLU 隠れユニット $a_j \,\mathrm{ReLU}(x - t_j)$ は、節点（knot）t_j の左側で平坦であり右側で傾きを a_j だけ変化させる**ヒンジ**として寄与する（Figure 4.5）。h 個のヒンジの和に定数を加えれば、h 個の内部節点を持つ任意の連続な区分線形折れ線を作ることができる。有界閉区間 $[a, b]$ 上の任意の連続関数は、十分に細かい折れ線によって任意の $\epsilon > 0$ 以内で一様近似できるため、十分に幅の広い単一の ReLU 隠れ層は、任意の 1 次元連続関数を任意の精度で近似できる。

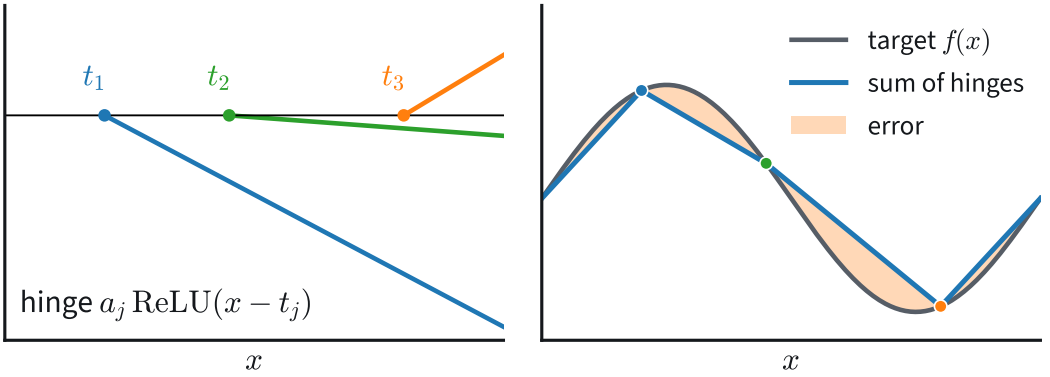


Figure 4.5: 区間上の滑らかな曲線を、傾きの変化ごとに 1 つの隠れユニットを割り当てた ReLU ヒンジ $a_j \,\mathrm{ReLU}(x - t_j)$ の和で近似する。

万能近似定理（universal approximation theorem）は、この事実を任意の入力次元 d へと一般化する。すなわち、隠れ層 1 層の MLP は、$\mathbb{R}^d$ の任意のコンパクト部分集合上の**任意の**連続関数を、**任意の**連続な非多項式の活性化関数 ϕ を用いて任意の一様精度で近似できる (Cybenko 1989;

Leshno et al. 1993)（リッジ関数と Stone–Weierstrass の定理を用いた 2 次元での証明については、演習問題 5 を参照）。

4.1.3.2 深さと空間の折りたたみ

隠れ層 1 つですでに何でも近似できるなら、なぜ**深い**ネットワークを構築するのだろうか。それは、万能近似が単一の層に必要な**ユニット数**（d に対して指数関数的に増大し得る）について何も語っておらず、また勾配降下法（gradient descent）が有限のデータから学習・汎化（generalization）できるかどうかも保証しないからである。

深さは、空間を折りたたむ写像を**合成**することで指数関数的なパラメータ効率を達成する (Telgarsky 2016)。2 個の ReLU ユニットによって計算される $[0,1]$ 上の 1 次元テント関数を考えよう。

$$g(z) = 2\,\mathrm{ReLU}(z) - 4\,\mathrm{ReLU}(z - 1/2), \tag{4.4}$$

この関数は $[0, 1/2]$ 上で 0 から 1 まで上昇し、$[1/2, 1]$ 上で再び 0 まで下降する（Figure 4.6）。g をそれ自身と合成すると、層ごとに $[0,1]$ が半分に折りたたまれる。$g \circ g$ は 4 個の線形区間を持ち、$g \circ g \circ g$ は 8 個を持ち、m 回の合成 $g^{\circ m}$ は m 個の隠れ層（各層 2 個の ReLU ユニット、合計 $2m$ ユニット）で計算され、2^m 個の線形区間を持つのこぎり波を生成する。この同じ 2^m 区間ののこぎり波を**単一の**隠れ層で表現するには、少なくとも $2^m - 1$ 個の隠れユニットが必要になる（$m = 10$ のとき、1 層では 1,023 ユニット必要なのに対し、10 層にわたる構成なら 20 ユニットで済む）。

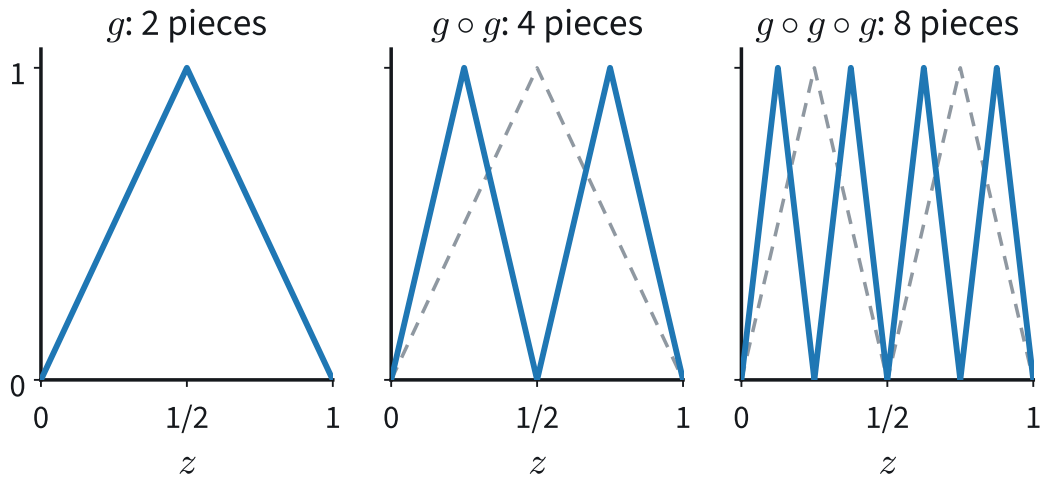


Figure 4.6: 2 個の ReLU からなるテント関数 g をそれ自身と m 回合成すると、層ごとに線形区間の数が倍増し、2^m 個の区間がわずか $2m$ 個の隠れユニットで生成される。

4.1.4 活性化関数

万能近似には非多項式であればほぼどのような ϕ でも十分であるが、勾配降下法による最適化は導関数 $\phi'(z)$ に決定的に依存する。隠れユニット j において、活性化前入力を $z_j = \sum_i x_i w_{ij}^{(1)} + b_j^{(1)}$、

活性化値を $h_j = \phi(z_j)$ とすると、連鎖律（chain rule）により入力側の各重みの勾配は局所的な傾き $\phi'(z_j)$ 倍にスケーリングされる。

$$\frac{\partial l}{\partial w_{ij}^{(1)}} = \frac{\partial l}{\partial h_j}\,\phi'(z_j)\,x_i. \tag{4.5}$$

$\phi'(z_j) \approx 0$ となる場所ではどこでも、勾配フロー（gradient flow）が停滞する。Figure 4.7 に 4 つの主要な活性化関数とその導関数を比較する。

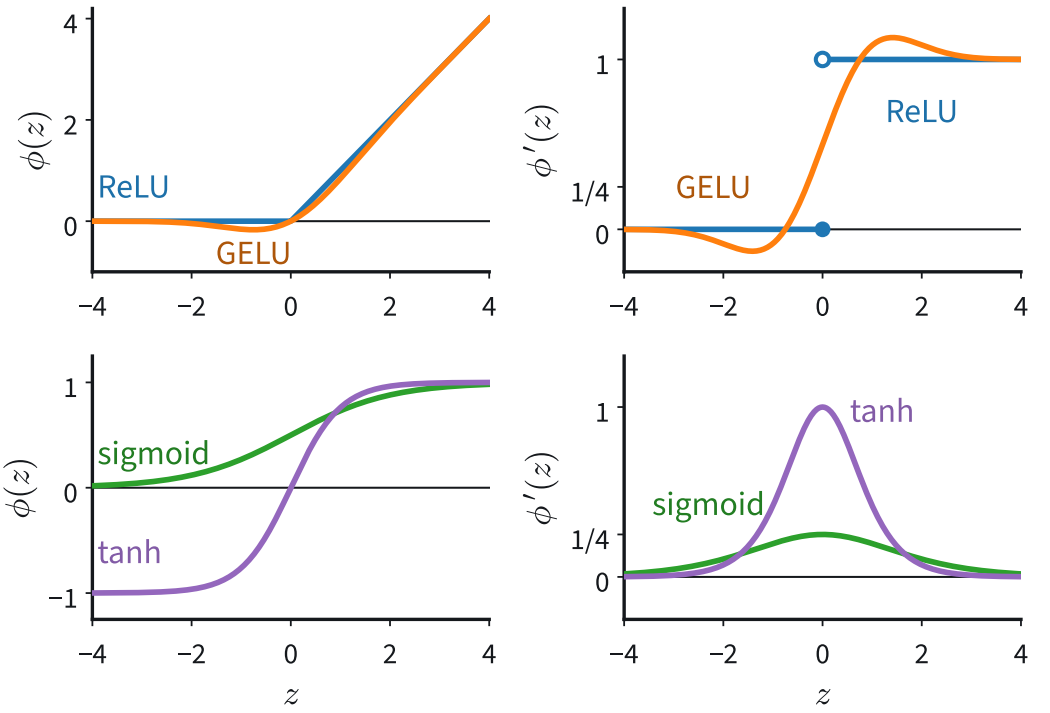


Figure 4.7: 4 つの活性化関数 $\phi(z)$（左）とその導関数 $\phi'(z)$（右）。ReLU と GELU（上）、シグモイドと tanh（下）。

- **ReLU と Leaky ReLU**：$\mathrm{ReLU}(z) = \max(z, 0)$ の導関数は $z > 0$ で 1、$z < 0$ で 0（慣例として $z = 0$ では 0）である。正の半直線全体にわたって傾きが恒等的に 1 であるため、活性な ReLU ユニットでは z が大きくなっても勾配が決して減衰しない (Nair and Hinton 2010; Glorot et al. 2011)。しかし、ユニットの重みが変化してすべての訓練入力に対して $z < 0$ になると、その勾配は恒久的に消失する（**死んだ ReLU**、dead ReLU）。そこで **Leaky ReLU** すなわち $\max(z, \alpha z)$（$\alpha \approx 0.01$）は、$z < 0$ に対して小さな正の傾きを保つことで、死んだユニットが回復できるようにする。
- **GELU と SiLU（Swish）**：このうち**ガウス誤差線形ユニット**（Gaussian error linear unit）$\mathrm{GELU}(z) = z\,\Phi(z)$（ただし Φ は標準正規分布の累積分布関数）とその類縁である $\mathrm{SiLU}(z) = z\,\sigma(z)$ は、原点における ReLU の折れ角を滑らかにし（$z = 0$ を傾き $1/2$ で通り、$z \approx -0.75$ 付近に緩やかな負のくぼみを持つ）、現代の Transformer においてデフォルトの活性化関数として用いられている (Hendrycks and Gimpel 2016)。

- **シグモイドと Tanh**：ロジスティックシグモイド $\sigma(z) = 1/(1+e^{-z}) \in (0,1)$ の導関数は $\sigma'(z) = \sigma(z)(1-\sigma(z)) \le 1/4$ である一方、ゼロ中心の双曲線正接（hyperbolic tangent）$\tanh(z) = (1-e^{-2z})/(1+e^{-2z}) = 2\sigma(2z) - 1 \in (-1,1)$（演習問題 3）の導関数は $1 - \tanh^2(z) \le 1$ である。どちらも、両裾で平坦な水平漸近線へと飽和して $\phi'(z) \to 0$ となり（$|z|$ が ~ 3 を超えて大きくなるとき）、$\sigma'(z) \le 1/4$ を K 層にわたって掛け合わせると勾配は少なくとも 4^{-K} 倍に縮小する（**勾配消失**、vanishing gradients。4.4 節で解析する）。現在、シグモイドと tanh は主にリカレントアーキテクチャにおける有界な**ゲート**や状態の圧縮関数として用いられている（第 12 章）。

4.1.5 まとめ

線形モデルはすべての座標に関して単調であり、XOR のような特徴量の相互作用を捉えることができない。多層パーセプトロンは、線形の出力層の前に隠れ層 $\mathbf{H}^{(k)} = \phi(\mathbf{H}^{(k-1)}\mathbf{W}^{(k)} + \mathbf{b}^{(k)})$ を挿入することでこれら両方の限界を克服する。非線形な活性化関数 ϕ がなければ、アフィン変換を何層積み重ねても単一の線形変換へと縮退してしまう。任意の連続な非多項式の活性化関数を持つ十分に幅広い単一の隠れ層は、コンパクト集合上の任意の連続関数を近似できる一方、m 個の隠れ層を合成すれば空間が繰り返し折りたたまれ、2^m 個の線形領域を $O(m)$ 個のユニットで作り出せる。活性化関数のなかでも、ReLU（$\max(z,0)$）や GELU（$z\Phi(z)$）のような整流型活性化関数は正の入力に対して 1 または 1 に近い傾きを維持するため、シグモイドや tanh ユニットを深く積み重ねた際に勾配を減衰させる裾の飽和を回避できる。

4.1.6 演習問題

1. **線形ネットワーク**：活性化関数がなければ、層を積み重ねてもアフィン写像が合成されて別のアフィン写像になるだけである。入力次元 d、出力次元 q、隠れ層の幅 $h_1, \dots, h_{K-1}$ を持つこのようなネットワークを考える。このネットワークが表現する関数が、アフィン写像 $\mathbf{XW} + \mathbf{b}$ のうち重み行列 $\mathbf{W}$ のランクが高々 $\min(h_1, \dots, h_{K-1})$ であるもの全体とちょうど一致することを示せ。この結果を用いて、線形モデルに隠れ層を追加するとその関数クラスが真に縮小する例を挙げよ。
2. **手計算による XOR**：@fig-mdl-mlp-xor は、クラス 1 の 2 つの入力を同じ隠れ空間上の点に写像することで XOR を計算する 2 ユニットの ReLU ネットワークを示している。
   1. 同じく XOR を計算するが、クラス 1 の 2 つの入力を**異なる**隠れ空間上の点に写像する 2 ユニットの ReLU ネットワークの重みを求め、4 つの入力で検証せよ。
   2. 単一の ReLU 隠れユニットと線形出力を持ついかなるネットワークも、その出力にどのようなしきい値を適用しても XOR を正しく分類できないことを示せ。
3. **シグモイドと tanh**：これら 2 つの圧縮関数は密接に関連している。
   1. $\tanh(z) = 2\sigma(2z) - 1$ を示せ。
   2. 隠れ層の活性化関数に tanh を持つ任意の MLP に対し、同じアーキテクチャで隠れ層の活性化関数にシグモイドを持ち同じ関数を計算する MLP が存在すること、またその逆も成り立つことを示せ。重みとバイアスがどのように変化するかを述べよ。
4. **深さと線形区間**：●実直線上で h 個の隠れユニットを持つ隠れ層 1 層の ReLU ネットワークは高々 $h+1$ 個の線形区間しか持たない一方、Figure 4.6 の 2 ユニットのテント関数を合成するたびに区間数は倍増する。本問では、1 層あたり 3 ユニットを持つネットワークについて深さによる利得を厳密に求める。
   1. 3 個の隠れユニット $\mathrm{ReLU}(w_j x + b_j)$ をとり、入力の重みを $w_1 = w_2 = 1$ および $w_3 = -1$、折れ点 $-b_j/w_j$ を $1/6$、$2/6$、$4/6$ とする。ネットワークの出力 f が値 $0, 1, 0, 1, 0$ を $x = 0, 1/6, 2/6, 4/6, 1$ で順にとるように、出力の重みと出力バイアスを選べ。
   2. $f \circ f$ が $[0,1]$ 上で持つ線形区間の数、および m 回の合成 $f \circ \cdots \circ f$ の線形区間の数

を数えよ。f との合成がグラフをどのように折りたたむかという観点から、合成のたびに区間数が加算ではなく乗算される理由を説明せよ。

3. この m 回の合成は、各 3 ユニットからなる m 個の隠れ層を持つネットワークによって計算される。単一の隠れ層を持つネットワークで同じ関数を表現するには、何個の隠れユニットが必要か。2 つのネットワークのパラメータ数を比較せよ。

Simon Prince『Understanding Deep Learning』問題 4.8 より改変。

5. **単一の隠れ層で十分 :●**実直線上では、単一の隠れ層は折れ点ごとに 1 つのヒンジを配置した折れ線によって連続関数を近似する。平面上には配置すべき折れ点が存在しないため、証明には別のアイデア、すなわち 1 つの方向に沿ってのみ変化する関数が必要になる。
   1. ψ を $\mathbb{R}$ 上の連続関数とし、$\mathbf{w} \in \mathbb{R}^2$ とする。単一の隠れ層が、**リッジ関数** $\psi(\mathbf{w}^\top \mathbf{x})$（$\mathbf{w}$ に直交するすべての直線上で定数となる関数）を平面の任意のコンパクト部分集合上で一様に近似できることを示せ。
   2. Stone–Weierstrass の定理は次のように述べる。コンパクト集合上の連続関数の集合が、和、積、スカラー倍に関して閉じており、定数関数を含み、任意の異なる 2 点に対してその 2 点で異なる値をとる関数を含むとする。このとき、そのコンパクト集合上のすべての連続関数は、この集合に属する関数の一様極限となる。$\cos(\omega^\top \mathbf{x})$ と $\sin(\omega^\top \mathbf{x})$（$\omega \in \mathbb{R}^2$）の有限線形結合全体がこのような集合をなすことを示せ。
   3. 以上より、単一の隠れ層が平面のコンパクト部分集合上の任意の連続関数を近似できることを結論づけよ。

Michael Nielsen『Neural Networks and Deep Learning』第 4 章より改変。

4.2 実装

4.1 節では、XOR を解くために 2 ユニットの ReLU ネットワークの重みを手作業で設計した。今度は、ミニバッチ SGD によって Fashion-MNIST 画像から隠れ特徴を直接学習させてみよう。隠れ層（hidden layer）として $h = 256$ 個の ReLU ユニットからなる単一の層を、$d = 784$ 個の入力ピクセルと $q = 10$ 個のクラスロジットの間に挿入すると、パラメータ数（parameter count）と画像あたりの FLOPs は 26 倍になる（パラメータ数は 203,530 個に対し、ソフトマックス回帰（softmax regression）では 7,850 個）。

本節では、この 2 層 MLP をスクラッチ（`MLPScratch`）と、簡潔な方法（`MLP`、フレームワークの `Sequential` コンテナを使用）の両方で実装する。さらに、訓練前の手軽な 3 つの妥当性確認によって未訓練のネットワークを検証し、その学習曲線（learning curve）と GPU エポック時間をソフトマックス回帰と直接比較したうえで、隠れユニット（hidden unit）が $\mathbf{W}^{(1)}$ 内に学習する重みパターンを観察する。

```
import time
```

4.2.1 スクラッチ実装

4.2.1.1 パラメータと順伝播

各 28×28 画像を $d = 784$ 個の特徴量（feature）に平坦化すると、$h = 256$ 個の隠れユニットと $q = 10$ 個の出力を持つ隠れ層 1 層の MLP（Figure 4.8）は、4 つのパラメータテンソルを確保する。

$$\mathbf{W}^{(1)} \in \mathbb{R}^{784 \times 256}, \quad \mathbf{b}^{(1)} \in \mathbb{R}^{256}, \quad \mathbf{W}^{(2)} \in \mathbb{R}^{256 \times 10}, \quad \mathbf{b}^{(2)} \in \mathbb{R}^{10}. \tag{4.6}$$

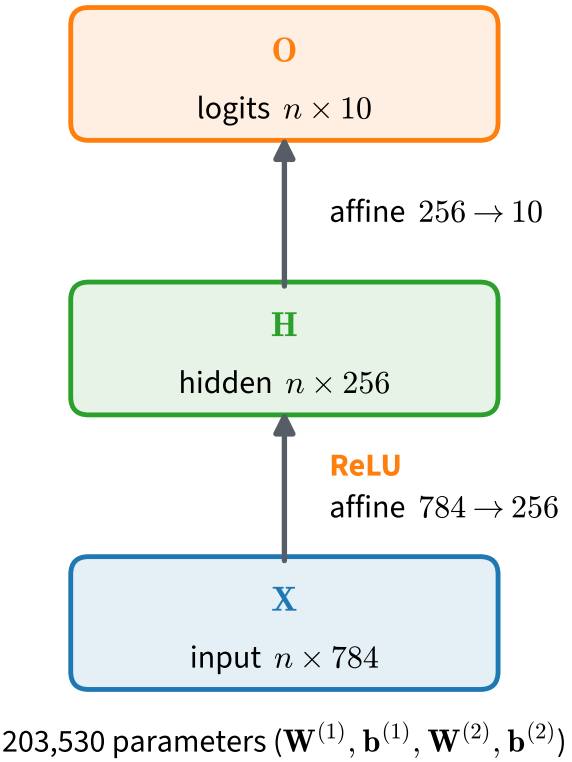


Figure 4.8: Fashion-MNIST 用 MLP のアーキテクチャ。n 枚の画像からなるミニバッチを $n \times 784$ に平坦化し、アフィン層と ReLU を通して $n \times 256$ の隠れ表現 $\mathbf{H}$ に写像したのち、$n \times 10$ のクラスロジット $\mathbf{O}$ へ射影する。

ソフトマックス回帰との直接比較で条件を揃えるため、両方の重み行列を $\mathcal{N}(0, 0.01^2)$ から、両方のバイアスを 0 に初期化する（4.4 節では、隠れユニット間の対称性を破るためにランダム初期化（random initialization）が不可欠である理由と、He 初期化（He initialization）が `std=0.01` をどのように改善するかを見る）。

```
class MLPScratch(d2l.Classifier):
    """An MLP with one hidden layer, implemented from scratch."""
    def __init__(self, num_inputs, num_outputs, num_hiddens, lr, std=0.01,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.W1 = nnx.Param(
            rngs.params.normal((num_inputs, num_hiddens)) * std)
        self.b1 = nnx.Param(jnp.zeros(num_hiddens))
        self.W2 = nnx.Param(
            rngs.params.normal((num_hiddens, num_outputs)) * std)
        self.b2 = nnx.Param(jnp.zeros(num_outputs))
```

次に、要素ごとの活性化関数 `relu(X)` をスクラッチで実装する。`X` にブールマスク `X > 0` を掛けることで、正の要素はそのまま通過させ、負の要素を 0 にする。

```
def relu(X):
    return X * (X > 0)
```

続いて隠れ層の処理を `hidden(self, X)` として切り出す。このメソッドは各画像ミニバッチを `(-1, self.num_inputs)` に平坦化して $\mathbf{H} = \mathrm{relu}(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$ を計算するため、`forward(self, X)` は生のクラスロジット $\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$ を返すだけでよい（この出力は `Classifier.loss` にそのまま渡される）。

```
@d2l.add_to_class(MLPScratch)
def hidden(self, X):
    X = d2l.reshape(X, (-1, self.num_inputs))  # One flattened image per row
    return relu(d2l.matmul(X, self.W1) + self.b1)

@d2l.add_to_class(MLPScratch)
def forward(self, X):
    return d2l.matmul(self.hidden(X), self.W2) + self.b2  # The logits
```

4.2.1.2 訓練前の確認

時間のかかる訓練を開始する前に、8 枚の画像からなる小さなスライスを**未訓練**のネットワークに通すことで、3 つの不変条件を数ミリ秒で検証できる。

1. **形状：** `H` の形状は `(8, 256)`、`O` の形状は `(8, 10)` である。
2. **初期損失：** ゼロに近い重みからはゼロに近いロジット（$\hat{y}_j \approx 1/10$）が生じるため、初期の交差エントロピー損失（cross-entropy loss）は $\log 10 \approx 2.303$ のすぐ近くに収まる。
3. **活性な ReLU の割合：** バイアスが 0 で重みが対称なため、隠れ層の活性化前の値（pre-activation）の約 50% が正になる。

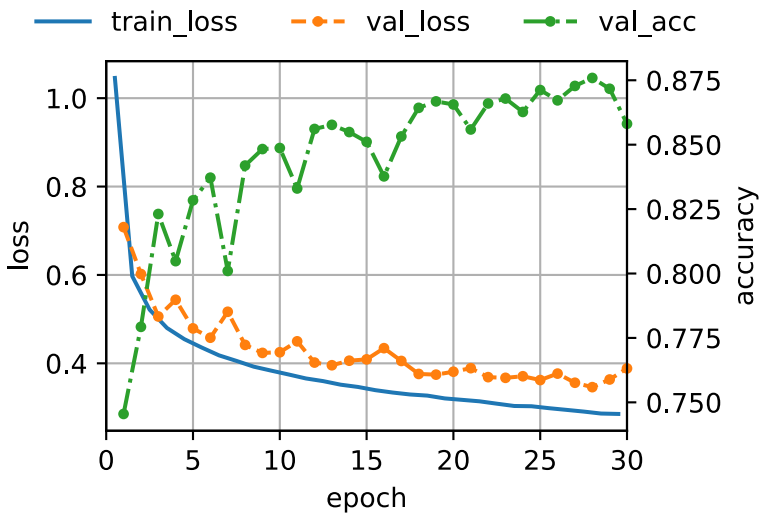
```
data = d2l.FashionMNIST(batch_size=256)
model = MLPScratch(num_inputs=784, num_outputs=10, num_hiddens=256, lr=0.1)
X, y = next(iter(data.train_dataloader()))
X, y = X[:8], y[:8]  # Eight images suffice for the checks
H, O = model.hidden(X), model(X)
active = d2l.reduce_mean(d2l.astype(H > 0, d2l.float32))
print('H:', H.shape, 'O:', O.shape)
print(f'loss {d2l.numpy(model.loss(O, y)):.3f}, '
      f'fraction of active units {d2l.numpy(active):.2f}')
```

```
H: (8, 256) O: (8, 10)
loss 2.294, fraction of active units 0.48
```

4.2.1.3 訓練

それでは、Fashion-MNIST 上で MLPScratch を 30 エポックにわたり、ミニバッチサイズを 256、学習率（learning rate）を $\eta = 0.1$ として訓練しよう（num_gpus=1 を渡し、GPU が利用可能なときには Trainer がそれを使うようにする）。

```
trainer = d2l.Trainer(max_epochs=30, num_gpus=1)
trainer.fit(model, data)
```



4.2.2 簡潔な実装

各フレームワークの Sequential コンテナと組み込みの線形層および ReLU 層を使えば、パラメータ配列を手動で定義することなく、まったく同じアーキテクチャを数行で記述できる（重みを $\mathcal{N}(0, 0.01^2)$ から、バイアスを 0 に初期化して MLPScratch と揃える）。

```
class MLP(d2l.Classifier):
    """An MLP with one hidden layer, built from framework layers."""
```

```
    def __init__(self, num_outputs, num_hiddens, lr, num_inputs=784,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        normal = nnx.initializers.normal(0.01)
        self.net = nnx.Sequential(
            nnx.Linear(num_inputs, num_hiddens, kernel_init=normal,
                       rngs=rngs),
            nnx.relu,
            nnx.Linear(num_hiddens, num_outputs, kernel_init=normal,
                       rngs=rngs))

    def forward(self, X):
        return self.net(X.reshape((X.shape[0], -1)))  # Flatten each image
```

4.2.3 ソフトマックス回帰との比較

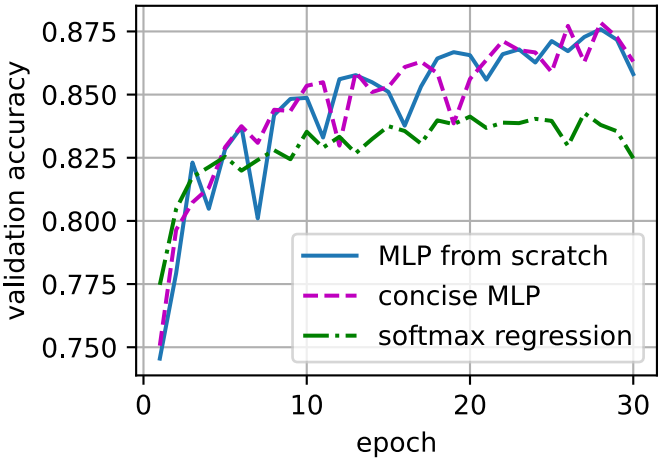
4.2.3.1 正解率

簡潔な実装の `MLP` と `d2l.SoftmaxRegression` の両方を同一の設定のもとで 30 エポック訓練し、各モデルのエポックあたりの平均実時間（wall-clock time）を `seconds` に、検証データでの正解率（accuracy）の推移を `curves` に記録しよう。

```
curves, seconds = {'MLP from scratch': trainer.history['val_acc']}, {}
trainer.board.display = False
models = {'concise MLP': MLP(num_outputs=10, num_hiddens=256, lr=0.1),
          'softmax regression': d2l.SoftmaxRegression(num_outputs=10, lr=0.1)}
for name, net in models.items():
    start = time.perf_counter()
    trainer.fit(net, data)
    seconds[name] = (time.perf_counter() - start) / trainer.max_epochs
    curves[name] = trainer.history['val_acc']
```

`curves` に記録した 3 つの検証正解率の履歴をエポック 1 から 30 にわたってプロットし、スクラッチ実装と簡潔な実装の両 MLP を線形のソフトマックスベースラインと直接比較する。

```
d2l.plot(list(range(1, trainer.max_epochs + 1)), list(curves.values()),
         'epoch', 'validation accuracy', legend=list(curves))
```



単一エポックごとの検証値のばらつきをならすため、最後の 5 エポック（`acc[-5:]`）における各モデルの平均検証正解率を、`MLPScratch` の訓練セット（training set）での正解率とあわせて出力する。

```
print('validation accuracy averaged over the last 5 epochs')
for name, acc in curves.items():  # A single epoch can land a few points low
    print(f'  {name}: {sum(acc[-5:]) / 5:.3f}')
train_acc = trainer.evaluate(model, data, train=True)['acc']
print(f'training accuracy of the MLP from scratch: {train_acc:.3f}')
```

```
validation accuracy averaged over the last 5 epochs
  MLP from scratch: 0.869
  concise MLP: 0.871
  softmax regression: 0.834
training accuracy of the MLP from scratch: 0.884
```

この学習曲線には、示唆に富む 2 つのパターンが見て取れる。

1. **`std=0.01` による遅いウォームアップ：**エポック 1 では、実はどちらの MLP もソフトマックス回帰に遅れをとる。$\mathbf{W}^{(1)}$ と $\mathbf{W}^{(2)}$ の両方が 0.01 前後の要素から始まるため、隠れ層の活性化値（activation）$\mathbf{H}$ および $\mathbf{W}^{(1)}$ へ逆伝播される勾配が 0.01 倍に抑制され、その状態は $\mathbf{W}^{(2)}$ が成長するまで続く。このボトルネックは 4.4 節で He 初期化によって解消する。
2. **分類誤りの 5 分の 1 の削減：**エポック 26〜30 までに、ソフトマックス回帰は検証正解率 84% 前後（誤り率 16%）で頭打ちになるのに対し、両 MLP は検証正解率 ～ 87%（誤り率 ～ 13%、訓練正解率 87〜90%）に達し、ソフトマックス回帰の誤分類の約 20% を解消する。

残っている誤分類はどのようなものだろうか。3.2 節で見たように、上半身のトップス 4 種（T シャツ、プルオーバー、コート、シャツ）は互いに似ているため、当然の候補となる。いま訓練した 2 つのモデルそれぞれについて、検証データでの誤分類数と、そのうちトップス同士を取り違えた件数を数えてみよう。

```
tops = [0, 2, 4, 6]  # T-shirt/top, pullover, coat, shirt
for name, net in models.items():
    errors = among_tops = 0
```

```
    for batch in data.val_dataloader():
        X, y = trainer.prepare_batch(batch)
        pred, y = d2l.numpy(net(X)).argmax(axis=1), d2l.numpy(y)
        errors += (pred != y).sum()
        among_tops += ((pred != y) & np.isin(y, tops) & np.isin(pred, tops)).sum()
    print(f'{name}: {errors} errors, {among_tops} of them between two tops')
```

```
concise MLP: 1369 errors, 864 of them between two tops
softmax regression: 1752 errors, 1061 of them between two tops
```

どちらのモデルでも、全誤分類の半数以上が 2 種のトップスの混同である。MLP はソフトマックス回帰よりもこうした誤りを減らしているものの、トップス類の識別は依然として Fashion-MNIST の難所である。

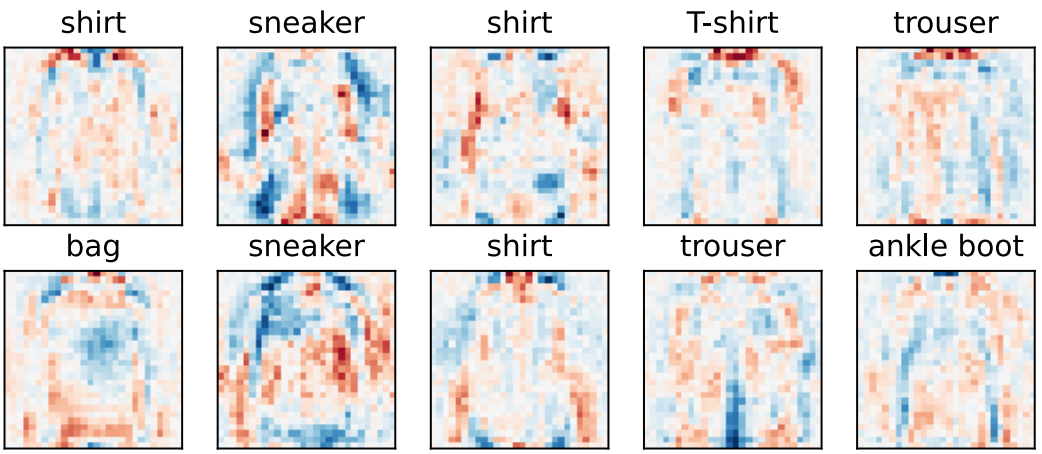
4.2.3.2 学習された特徴

3.3 節では、ソフトマックス回帰の重み行列 $\mathbf{W}$ の 10 個の列が、それぞれ衣類全体のテンプレートとして機能せざるを得なかった。これに対し、MLP の 256 個の隠れユニットが何を学習したかを調べるため、訓練済みの重み行列 $\mathbf{W}^{(1)}$ と $\mathbf{W}^{(2)}$ を NumPy 配列として取り出す。

```
W1, W2 = d2l.numpy(model.W1[...]), d2l.numpy(model.W2[...])
```

重みのユークリッドノルムが最も大きい 10 個の隠れユニットを 28×28 の画像として表示し、各ユニットには出力側の正の重みが最大となる衣類クラスのラベルを添える。

```
names = data.text_labels(range(10))
units = np.argsort(np.linalg.norm(W1, axis=0))[::-1][:10]  # Largest weights
images = W1[:, units].T.reshape(10, 28, 28)
images /= np.abs(images).max(axis=(1, 2), keepdims=True)  # Own color scale
d2l.show_images(images, 2, 5, titles=[names[k] for k in W2[units].argmax(1)],
                cmap='RdBu_r', vmin=-1, vmax=1);
```



これらの画像は、衣類全体のテンプレートには見えない。ノルムの大きい各隠れユニットは、画像の一部の領域に正の重み（赤）を、隣接する領域に負の重み（青）を与えており、たとえば肩の

輪郭や胴体の両脇に沿った「ある領域の明るいピクセルとその隣の暗いピクセル」という**コントラスト**に反応する。もっとも、パターンの大半は依然として衣類の広い範囲にまたがっているため、小さなエッジ検出器とみなすべきではない。出力層は、このようなコントラストを 256 個組み合わせることで 10 個のスコアを作る。

4.2.3.3 パラメータ、計算量、時間

モデル	パラメータ数	画像あたりの積和演算数	検証正解率（エポック 26〜30）
ソフトマックス回帰	7,850	7,840	$\sim 84\%$
MLP（$h = 256$ 個の ReLU ユニット）	203,530	203,264	$\sim 87\%$

この MLP は画像あたりの積和演算数がソフトマックス回帰の 26 倍にのぼるにもかかわらず、エポックあたりの実時間がどうなるか比較してみよう。

```
    for name, s in seconds.items():
        print(f'{name}: {s:.1f} s per epoch')
```

```
concise MLP: 0.8 s per epoch
softmax regression: 0.7 s per epoch
```

現代のハードウェア（特に GPU）では、どちらのモデルもエポックあたりの実時間はほぼ同じである。幅 $h = 256$ では、データの読み込みや Python またはカーネル起動のオーバーヘッドが実行時間の大部分を占めるため、密行列演算が 26 倍に増えても実時間上のコストはほとんど増えない（演習問題 3 では、どの幅から行列演算がデータ読み込みのオーバーヘッドを上回るかを調べる）。

4.2.4 まとめ

訓練を開始する前に、小さなバッチに対して順伝播（forward pass）を 1 回実行することで、中間テンソルと出力テンソルの形状を検証し、初期の交差エントロピー損失が $\log q \approx 2.30$ 付近にあることを確かめ、さらに ReLU 隠れユニットの約半数が活性化することを確認できる。Fashion-MNIST では、256 ユニットの ReLU 隠れ層を 1 層挿入することで検証正解率が $\sim 84\%$ から $\sim 87\%$ へと向上し、エポックあたりの GPU 実時間をほぼ変えずに誤り率を約 5 分の 1 削減できる。このとき隠れユニットは、ぼやけた衣類全体のテンプレートを作るのではなく、隣接する領域に逆符号の重みを与えるコントラストパターンを学習し、それらが出力層で組み合わされる。

4.2.5 演習問題

1. [code] **活性化関数なしの隠れ層**。`relu` を `MLPScratch` 内で恒等写像に置き換え、得られたモデルを本節と同様に訓練せよ。
    1. 30 エポック後の検証正解率を、MLP およびソフトマックス回帰の検証正解率と比較せよ。
    2. 訓練後に $\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)}$ と $\mathbf{b} = \mathbf{W}^{(2)\top}\mathbf{b}^{(1)} + \mathbf{b}^{(2)}$ を計算し、ミニバッチ上で $\mathbf{XW} + \mathbf{b}$ がモデルのロジットを再現することを確かめよ。
    3. 演習問題 1 で示したとおり、このようなネットワークが計算するのは、重み行列のランクが高々隠れ層の幅であるようなアフィン写像にちょうど一致する。ここでこの制

約が実質的に何の制限にもならない理由を説明し、さらに隠れ層の幅がいくつ未満になると Fashion-MNIST 上でモデルが表現できる範囲が縮小するかを述べよ。

2. [code] **単一の隠れユニット**。`num_hiddens=1` を `MLPScratch` に設定し、隠れ層が重み $\mathbf{w}^{(1)} \in \mathbb{R}^{784\times 1}$ とバイアス $b^{(1)} \in \mathbb{R}$ を持つ単一の ReLU ユニットからなるようにせよ。このとき各入力 $\mathbf{x} \in \mathbb{R}^{1\times 784}$ はスカラー $t = \text{ReLU}(\mathbf{x}\mathbf{w}^{(1)} + b^{(1)}) \geq 0$ に縮約され、ロジットは $\mathbf{o} = t\,\mathbf{w}^{(2)} + \mathbf{b}^{(2)}$（ただし $\mathbf{w}^{(2)} \in \mathbb{R}^{1\times 10}$、$\mathbf{b}^{(2)} \in \mathbb{R}^{10}$）となる。
   1. 各クラスが予測されるのは t の高々 1 つの区間上であること、および t 軸に沿ったそれらの区間の順序が $\mathbf{w}^{(2)}$ の各成分によって定まることを示せ。さらに、モデルが各クラスに割り当てる入力空間の領域を記述し、それが Fashion-MNIST の 10 クラスを分離するうえで何を意味するかを述べよ。
   2. ある更新の後に、すべての訓練例に対して $\mathbf{x}\mathbf{w}^{(1)} + b^{(1)} < 0$ になったとする。それ以降、0 でない勾配を受け取るパラメータはどれか、またネットワークは何を予測するか。本節の 256 個の隠れユニットのうち 1 つに同じ現象が起きてもほとんど悪影響がないのはなぜか。
   3. モデルを 5 回訓練し、各実行について検証正解率と、検証例のうち $t > 0$ となる割合（`model.hidden` から得られる）を報告せよ。この割合が 0 になった実行があれば、その結果を (b) の解答と比較せよ。
3. [code] **幅**。本節と同様に `MLP` を学習率 0.1 で 30 エポック訓練せよ。ただし隠れ層の幅を $h \in \{16, 64, 256, 1024, 4096\}$ と変化させ、最終的な検証正解率とエポックあたりの時間を記録せよ。
   1. 両方を対数軸上の h に対してプロットせよ。どの幅を超えると、正解率の改善幅がエポック間の変動を超えなくなるか。
   2. 積和演算数は h に比例して増加する。モデルを一切使わずミニバッチの読み込みだけを行う訓練データ 1 周分の時間を計測し、それを用いて時間曲線のうち h に伴って増加しない部分を説明せよ。使用しているハードウェアでは、どの幅から演算時間が支配的になるか。
   3. 幅 64 と 1024 について、学習率を $\{0.03, 0.1, 0.3\}$ として訓練を繰り返せ。最良の学習率は幅に依存するか。

4.3 誤差逆伝播法

4.2 節では、導関数を 1 つも手書きすることなく Fashion-MNIST の MLP がもつ 203,530 個すべてのパラメータを訓練した。自動微分（automatic differentiation）があらゆる勾配を計算してくれたからである。1.5 節ではその一般原理を見た。すなわち、リバースモード自動微分は順伝播（forward pass）を計算グラフ（computational graph）として記録し、スカラー損失から逆向きに連鎖律（chain rule）をたどるのである。

本節では、多層パーセプトロン（multilayer perceptron、MLP）についてそのブラックボックスの中身を開いてみよう。MLP 全体に必要なのは、行列積、ブロードキャストされたバイアスの加算、要素ごとの活性化という **3 つの局所的な行列微分の規則**だけである。これらを順にたどることで、工学上重要な 3 つの問いに答えることができる。

1. どのようにして 1 回の逆伝播（backward pass）は、1 回の順伝播の高々約 $2\times$ の演算コストで数十万個もの偏微分（partial derivative）を計算するのか。
2. なぜ訓練は推論（inference）よりもはるかに多くの GPU メモリを消費し、そのメモリ量はミニバッチサイズ n と深さ K に対して線形に増加するのか。
3. 死んだ ReLU ユニット（dead ReLU unit）や勾配消失・勾配爆発（vanishing/exploding gradients）は、逆伝播の漸化式から機械的にどのように生じるのか。

4.3.1 計算グラフ

4.3.1.1 小さなグラフ

リバースモード微分を最も単純な形で復習するため、3 つのスカラーからなる式 $e = (a + b)\,c$ を $a = 2, b = 1, c = -3$ において考えよう（Figure 4.9）。**順伝播**（左から右）では $u = a + b = 3$ と $e = u\,c = -9$ を計算する。**逆伝播**（右から左）では $\partial e/\partial e = 1$ をシードとして与え、ノードごとに 1 つの局所導関数を適用する。

- 乗算ノード $e = u\,c$ では、$\partial e/\partial u = c = -3$ および $\partial e/\partial c = u = 3$ となる。
- 加算ノード $u = a + b$ では、$\partial u/\partial a = \partial u/\partial b = 1$ であるため、$\partial e/\partial a = \partial e/\partial b = -3$ となる。

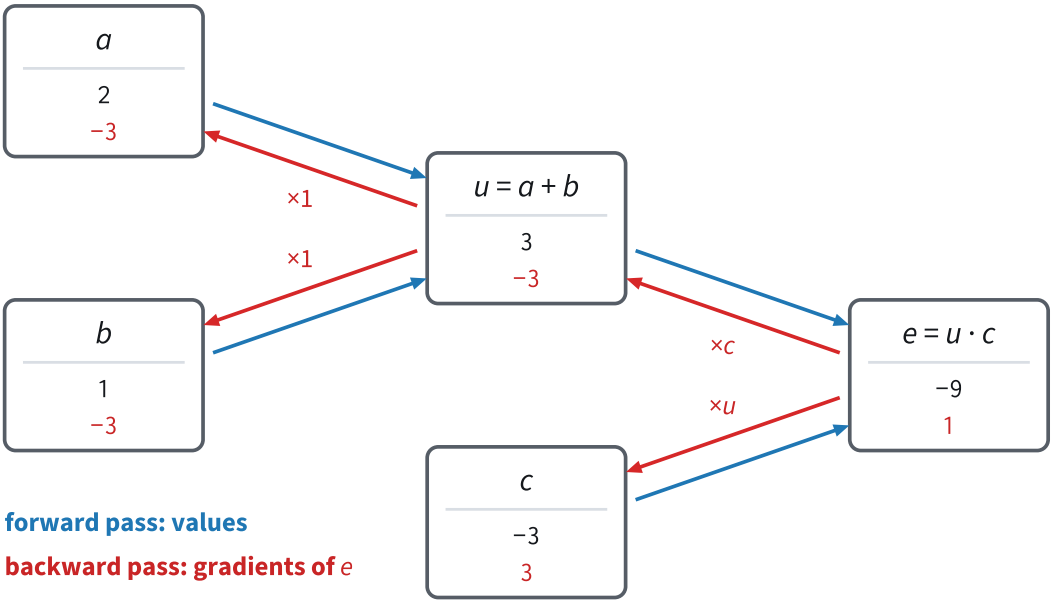


Figure 4.9: $e = (a + b)\,c$ の $a = 2, b = 1, c = -3$ における計算グラフ。順伝播の値（黒）とリバースモードの導関数 $\partial e/\partial(\cdot)$（赤）を示す。

ここにすでに表れている 2 つの普遍的な規則に注目しよう。(i) 加算ノードは入力された勾配をそのまま各被加数へコピーするのに対し、乗算ノードは入力された勾配に**他方**の因子を掛ける（そのため順伝播で u と c をキャッシュしておく必要がある）。(ii) 逆伝播はグラフを**逆トポロジカル順序**（reverse topological order）で走査する。

4.3.1.2 MLP のグラフ

次に、隠れ層（hidden layer）を 1 つもつ MLP（Figure 4.10）をミニバッチ $\mathbf{X} \in \mathbb{R}^{n \times d}$ 上で考えよう。パラメータを $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$、$\mathbf{b}^{(1)} \in \mathbb{R}^{h}$、$\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$、$\mathbf{b}^{(2)} \in \mathbb{R}^{q}$ とし、ℓ_2 重み減衰（weight decay）ペナルティを $s = \frac{\lambda}{2}\left(\|\mathbf{W}^{(1)}\|_{\mathrm{F}}^2 + \|\mathbf{W}^{(2)}\|_{\mathrm{F}}^2\right)$ とすると、次のようになる。

$$
\begin{aligned}
\mathbf{Z} &= \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}, \\
\mathbf{H} &= \phi(\mathbf{Z}), \\
\mathbf{O} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}, \\
J &= L(\mathbf{O}, \mathbf{Y}) + s,
\end{aligned}
\tag{4.7}
$$

ここで、L はミニバッチ上の平均交差エントロピー損失（cross-entropy loss）であり、$\|\mathbf{W}\|_{\mathrm{F}}^2 = \sum_{r,j} w_{rj}^2$ は二乗フロベニウスノルムである。

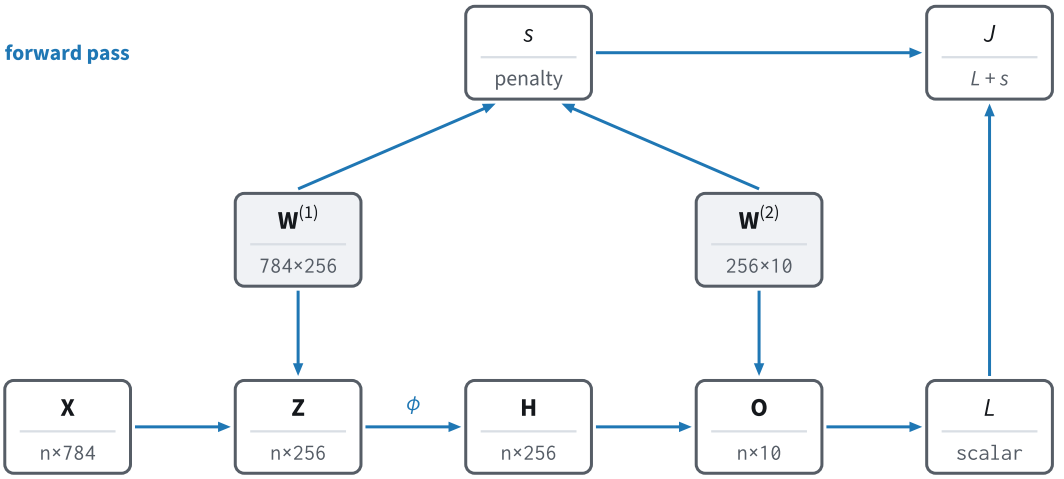


Figure 4.10: 重み減衰を伴う 1 隠れ層 MLP（$d = 784, h = 256, q = 10$）における順伝播の計算グラフ。パラメータノードは網掛けされており、各重み行列はそのアフィン層と正則化ペナルティ s の両方に入力される。

4.3.2 MLP における誤差逆伝播法

4.3.2.1 行列に対する局所規則

任意の行列 $\mathbf{A}$ に対し、$\partial J/\partial \mathbf{A}$ は偏微分の行列 $(\partial J/\partial a_{ij})$ を表し、$\mathbf{A}$ と**まったく同じ形状**をもつものとする。3 つの局所的なベクトル・ヤコビ行列規則（34.4.3.2 節）によって、MLP のすべての層を扱うことができる。

1. **行列積（$\mathbf{A} = \mathbf{X}\mathbf{W}$、ただし $\mathbf{X} \in \mathbb{R}^{n\times d}, \mathbf{W} \in \mathbb{R}^{d\times h}$）：**上流の勾配 $\mathbf{G} = \partial J/\partial \mathbf{A} \in \mathbb{R}^{n\times h}$ が与えられたとき、$a_{ij} = \sum_r x_{ir} w_{rj}$ であるから、連鎖律より $\partial J/\partial w_{rj} = \sum_i x_{ir} g_{ij}$ および $\partial J/\partial x_{ir} = \sum_j g_{ij} w_{rj}$ が得られる。行列形式で書けば次のようになる。

$$
\frac{\partial J}{\partial \mathbf{W}} = \mathbf{X}^{\top}\mathbf{G}, \qquad \frac{\partial J}{\partial \mathbf{X}} = \mathbf{G}\,\mathbf{W}^{\top}.
\tag{4.8}
$$

   形状がぴったりかみ合うことに注目しよう。$\mathbf{X}^{\top}\mathbf{G}$ は $(d\times n)(n\times h) = d\times h$（$\mathbf{W}$ と一致）となり、$\mathbf{G}\mathbf{W}^{\top}$ は $(n\times h)(h\times d) = n\times d$（$\mathbf{X}$ と一致）となる。スカラーの乗算とまったく同様に、$\partial J/\partial \mathbf{W}$ の計算にはキャッシュされた入力 $\mathbf{X}$ が必要であり、一方で $\partial J/\partial \mathbf{X}$ の計算は $\mathbf{W}^{\top}$ を通じて誤差信号を逆方向へ伝播させる。

2. **ブロードキャストされたバイアスの加算（$\mathbf{Z} = \mathbf{A} + \mathbf{b}$、ただし $\mathbf{b} \in \mathbb{R}^h$）：**$b_j$ はすべての行 i にわたって列 j に加算されるため、$\partial J/\partial \mathbf{A} = \mathbf{G}$ は変化せずそのまま通過し、一方で $\partial J/\partial \mathbf{b} = \mathbf{G}^\top \mathbf{1}_n$ は $\mathbf{G}$ を n 個のデータ例（行）にわたって合計する。
3. **要素ごとの活性化（$\mathbf{H} = \phi(\mathbf{Z})$）：**$h_{ij} = \phi(z_{ij})$ は成分 (i, j) のみに依存するため、上流の勾配は局所的な活性化の傾きによって成分ごとにゲートされる（アダマール積 $\odot$）。

$$\frac{\partial J}{\partial \mathbf{Z}} = \frac{\partial J}{\partial \mathbf{H}} \odot \phi'(\mathbf{Z}).$$

4.3.2.2 逆伝播

$\partial J/\partial \mathbf{O} = \frac{1}{n}(\hat{\mathbf{Y}} - \mathbf{Y})$（3.1.2.2 節参照。ここで $\hat{\mathbf{Y}} = \text{softmax}(\mathbf{O})$ であり、$\mathbf{Y} \in \{0, 1\}^{n \times q}$ はワンホットラベル行列である）および $\partial s/\partial \mathbf{W}^{(k)} = \lambda \mathbf{W}^{(k)}$ から出発し、3 つの規則を出力から入力へ向かって逆トポロジカル順序で適用すると、MLP の完全な逆伝播が得られる。

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{O}} &= \frac{1}{n}(\hat{\mathbf{Y}} - \mathbf{Y}), \\
\frac{\partial J}{\partial \mathbf{W}^{(2)}} &= \mathbf{H}^\top \frac{\partial J}{\partial \mathbf{O}} + \lambda \mathbf{W}^{(2)}, \qquad \frac{\partial J}{\partial \mathbf{b}^{(2)}} = \left(\frac{\partial J}{\partial \mathbf{O}}\right)^\top \mathbf{1}, \\
\frac{\partial J}{\partial \mathbf{H}} &= \frac{\partial J}{\partial \mathbf{O}} \mathbf{W}^{(2)\top}, \\
\frac{\partial J}{\partial \mathbf{Z}} &= \frac{\partial J}{\partial \mathbf{H}} \odot \phi'(\mathbf{Z}), \\
\frac{\partial J}{\partial \mathbf{W}^{(1)}} &= \mathbf{X}^\top \frac{\partial J}{\partial \mathbf{Z}} + \lambda \mathbf{W}^{(1)}, \qquad \frac{\partial J}{\partial \mathbf{b}^{(1)}} = \left(\frac{\partial J}{\partial \mathbf{Z}}\right)^\top \mathbf{1}.
\end{aligned} \tag{4.9}$$

一般の K 層 MLP（$\mathbf{Z}^{(k)} = \mathbf{H}^{(k-1)}\mathbf{W}^{(k)} + \mathbf{b}^{(k)}, \mathbf{H}^{(k)} = \phi(\mathbf{Z}^{(k)})$）では、各層 $k = K-1, \ldots, 1$ がまったく同じ 2 ステップのモジュールを実行する。

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{Z}^{(k)}} &= \left(\frac{\partial J}{\partial \mathbf{Z}^{(k+1)}} \mathbf{W}^{(k+1)\top}\right) \odot \phi'(\mathbf{Z}^{(k)}), \\
\frac{\partial J}{\partial \mathbf{W}^{(k)}} &= \mathbf{H}^{(k-1)\top} \frac{\partial J}{\partial \mathbf{Z}^{(k)}} + \lambda \mathbf{W}^{(k)}.
\end{aligned}$$

4.3.2.3 計算例

Equation 4.9 を、ごく小さなネットワーク（$n = 1, d = h = 2, q = 1$、ReLU 活性化、バイアスは 0、$\lambda = 0$、二乗損失（squared loss）$L = \frac{1}{2}(o - y)^2$）上で手計算により追ってみよう。

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \end{bmatrix}, \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad \mathbf{W}^{(2)} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad y = 0. \tag{4.10}$$

順伝播（Figure 4.11）では、$\mathbf{Z} = \mathbf{X}\mathbf{W}^{(1)} = \begin{bmatrix} -1 & 2 \end{bmatrix}$、$\mathbf{H} = \text{ReLU}(\mathbf{Z}) = \begin{bmatrix} 0 & 2 \end{bmatrix}$（ユニット 1 は非活性）、$o = \mathbf{H}\mathbf{W}^{(2)} = -2$、$L = \frac{1}{2}(-2)^2 = 2$ となる。逆伝播では、$\partial L/\partial o = o - y = -2$ であり、各勾配は次のようになる。

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}^{(2)}} &= \mathbf{H}^\top \frac{\partial L}{\partial o} = \begin{bmatrix} 0 \\ 2 \end{bmatrix} (-2) = \begin{bmatrix} 0 \\ -4 \end{bmatrix}, \\
\frac{\partial L}{\partial \mathbf{H}} &= \frac{\partial L}{\partial o} \mathbf{W}^{(2)\top} = [-4 \quad 2], \\
\frac{\partial L}{\partial \mathbf{Z}} &= [-4 \quad 2] \odot [0 \quad 1] = [0 \quad 2], \\
\frac{\partial L}{\partial \mathbf{W}^{(1)}} &= \mathbf{X}^\top \frac{\partial L}{\partial \mathbf{Z}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} [0 \quad 2] = \begin{bmatrix} 0 & 2 \\ 0 & 4 \end{bmatrix}.
\end{aligned}
$$

$\partial L/\partial \mathbf{W}^{(1)}$ の 1 列目がすべて 0 になっていることに注目しよう。$z_1 = -1 < 0$ であるため、$\phi'(z_1) = 0$ によって上流の勾配 $\partial L/\partial h_1 = -4$ が遮断され、ユニット 1 へ入力されるどの重みにも届かないからである。

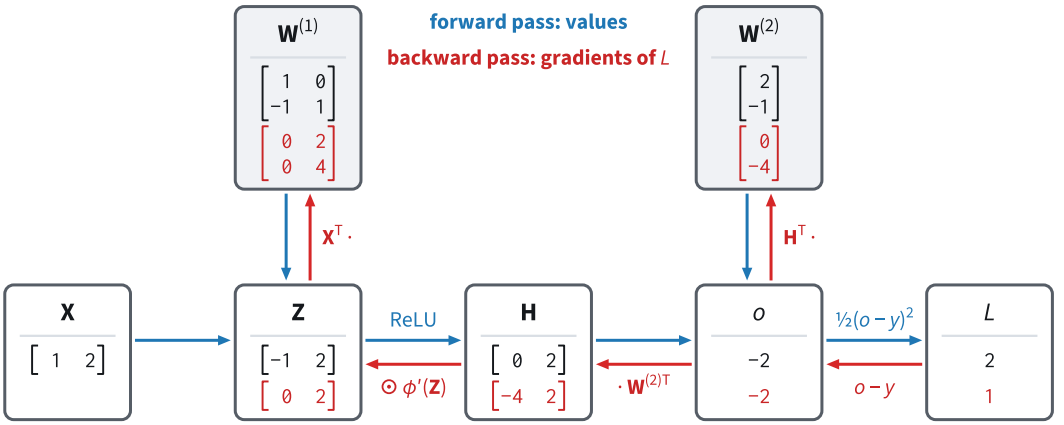


Figure 4.11: 計算例（Equation 4.10）における順伝播の値（黒）と逆伝播の勾配（赤）。$z_1 = -1 < 0$ であるため、ReLU ゲートによって $\partial L/\partial z_1$ および $\partial L/\partial \mathbf{W}^{(1)}$ の 1 列目が 0 になる。

4.3.2.4 ミニバッチ上での逆伝播の検証

Equation 4.9 を `mlp_backward` として実装し、64 ビット浮動小数点（floating point）を用いてランダムな 256×784 のミニバッチ（$h = 256, q = 10, \lambda = 10^{-3}$）上で各フレームワークの自動微分と比較して検証しよう。

```
def mlp_backward(X, Y, W1, b1, W2, b2, lambd):
    """Gradients of J = cross-entropy + weight decay, by backprop."""
    Z = X @ W1 + b1  # forward pass, keeping X, Z, H, and Y_hat
    H = jax.nn.relu(Z)
    Y_hat = jax.nn.softmax(H @ W2 + b2, axis=1)
    G_O = (Y_hat - Y) / X.shape[0]  # dJ/dO
    dW2, db2 = H.T @ G_O + lambd * W2, G_O.sum(axis=0)
    G_Z = (G_O @ W2.T) * (Z > 0)  # dJ/dH, then dJ/dZ
    dW1, db1 = X.T @ G_Z + lambd * W1, G_Z.sum(axis=0)
    return dW1, db1, dW2, db2
```

256 個のデータ例からなるランダムなミニバッチを、4.2 節とまったく同じ次元および初期化（initialization）に $\lambda = 10^{-3}$ を加えた設定で生成し、`mlp_backward` を各フレームワーク組み込みの自動微分と比較しよう。勾配チェックの定石に従い、64 ビット浮動小数点（丸め誤差の下限が約 10^{-16} と小さいため、わずかな代数的バグも確実に検出できる）で実行し、パラメータテンソルごとに最大の相対差を出力する。

```
jax.config.update('jax_enable_x64', True)  # 64-bit is off by default
n, d, h, q, lambd = 256, 784, 256, 10, 1e-3
f64 = jnp.float64  # 64-bit floating point throughout the check
keys = jax.random.split(d2l.get_key(), 4)
X = jax.random.normal(keys[0], (n, d), f64)
y = jax.random.randint(keys[1], (n,), 0, q)
params = [jax.random.normal(keys[2], (d, h), f64) * 0.01, jnp.zeros(h, f64),
          jax.random.normal(keys[3], (h, q), f64) * 0.01, jnp.zeros(q, f64)]

def objective(params):
    W1, b1, W2, b2 = params
    O = jax.nn.relu(X @ W1 + b1) @ W2 + b2
    return (optax.softmax_cross_entropy_with_integer_labels(O, y).mean()
            + lambd / 2 * ((W1 ** 2).sum() + (W2 ** 2).sum()))

auto = jax.grad(objective)(params)  # automatic differentiation
grads = mlp_backward(X, jax.nn.one_hot(y, q, dtype=f64), *params, lambd)
for name, a, g in zip(['W1', 'b1', 'W2', 'b2'], auto, grads):
    rel = jnp.abs(a - g).max() / jnp.abs(a).max()
    print(f'{name}: relative difference {float(rel):.1e}')
```

```
W1: relative difference 3.5e-16
b1: relative difference 1.7e-16
W2: relative difference 2.2e-16
b2: relative difference 1.8e-16
```

すべてのパラメータ勾配は自動微分の結果と相対誤差 10^{-15} 以内で一致しており、これは倍精度（double precision）のマシンイプシロン 2.2×10^{-16} の数倍程度にすぎない。

4.3.3 メモリと計算量

4.3.3.1 保存される活性化値とパラメータメモリ

Equation 4.9 を見ると、逆伝播において $\partial J/\partial \mathbf{W}^{(2)}$ と $\partial J/\partial \mathbf{W}^{(1)}$ を計算するには、順伝播で $\mathbf{X}$、$\mathbf{H}$（ReLU の場合はマスク $\mathbf{Z} > 0$ の情報も兼ねる）、および $\hat{\mathbf{Y}}$ を保持しておく必要がある。Fashion-MNIST の MLP（$d = 784, h = 256, q = 10$）においてミニバッチサイズを $n = 256$ とすると、キャッシュされたこれらのテンソルが保持する要素数は次のようになる。

$$256 \times 784 + 256 \times 256 + 256 \times 10 = 268{,}800 \text{ floats} \approx 1.1 \text{ MB}, \tag{4.11}$$

これはすでに 203,530 個のパラメータ（≈ 0.8 MB）を上回っている。K 層のネットワークでは、活性化メモリ（activation memory）はバッチサイズ n と深さ K の**両方**に対して線形に増大するのに対し、推論では次の層を計算した直後に各層の活性化値を破棄できる。

4.3.3.2 演算コスト

順伝播では、各データ例あたり $dh + hq = 784 \times 256 + 256 \times 10 = 203{,}264$ 回の積和演算（multiply-add）を実行する。逆伝播では、内部層の各行列積 $\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}$ から同じサイズの逆方向の行列積が **2 つ**（重みに対する $\mathbf{H}^{(k-1)\top}\mathbf{G}$ と下層の活性化値に対する $\mathbf{G}\mathbf{W}^{(k)\top}$）生じる一方、最初の層で必要なのは $\mathbf{X}^\top \mathbf{G}_Z$ のみである（$\partial J/\partial \mathbf{X}$ は不要なため）。したがって、この 2 層 MLP の逆伝播は $dh + 2hq = 205{,}824$ 回の積和演算で済み、順伝播とほぼ同じである。計算量の大部分を最初の層が占めており、そこでは逆方向の積が 1 回しか必要ないからである。深層ネットワークではほぼすべての層が内部層となるため、逆伝播には順伝播の $\sim 2\times$ のコストがかかり、訓練 1 ステップ全体では順伝播による推論のおよそ $3\times$ となる。いずれにせよ、$203{,}530$ 個すべての導関数が追加の順伝播 1〜2 回分のコストで求まるのに対し、片側有限差分（finite difference）では**たった 1 つ**のスカラーパラメータにつき 1 回分の追加の順伝播を費やすことになる。

4.3.4 まとめ

多層パーセプトロンにおける誤差逆伝播法（backpropagation）は、逆トポロジカル順序で適用される 3 つの局所的な行列規則に帰着する。すなわち、$\partial J/\partial \mathbf{W} = \mathbf{X}^\top \mathbf{G}$ と $\partial J/\partial \mathbf{X} = \mathbf{G}\mathbf{W}^\top$（行列積 $\mathbf{A} = \mathbf{X}\mathbf{W}$ に対して）、行方向の和 $\mathbf{G}^\top \mathbf{1}$（ブロードキャストされたバイアスに対して）、および要素ごとのゲート処理 $\mathbf{G} \odot \phi'(\mathbf{Z})$（活性化に対して）であり、パラメータがデータ損失と重み減衰の両方に入力される箇所では流入する勾配を合算する。$\mathbf{X}^\top \mathbf{G}$ は各層への順伝播の入力に依存するため、訓練時には順方向スイープ全体にわたって中間の活性化値をキャッシュする必要があり、バッチサイズと深さに対して線形に増大する $O(nKh)$ のメモリを要する。その見返りとして、1 回の逆方向スイープだけで、1 回の順伝播の高々約 2 倍の FLOPs ですべてのパラメータ勾配を計算できる。

4.3.5 演習問題

1. **層のヤコビ行列**。スカラー J の $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ に関する勾配は、$\mathbf{W}^{(2)}$ と同じ形状をもつ。一方、行列 $\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} \in \mathbb{R}^{n \times q}$ の $\mathbf{W}^{(2)}$ に関する導関数はより大きな対象であり、4 つの添字のあらゆる組み合わせに対して 1 つずつ成分 $\partial o_{ij}/\partial w_{rt}$ をもつ。
    1. $\partial \mathbf{O}/\partial \mathbf{W}^{(2)}$ の成分数はいくつか。また、そのうち非ゼロになり得るものはいくつあるか。
    2. 連鎖律 $\partial J/\partial w_{rt} = \sum_{i,j} (\partial J/\partial o_{ij})(\partial o_{ij}/\partial w_{rt})$ を計算し、$\mathbf{H}^\top \partial J/\partial \mathbf{O}$ を導出せよ。
    3. $\partial \mathbf{O}/\partial \mathbf{W}^{(2)}$ は、4.2 節の MLP で $n = 256$ としたとき、32 ビット浮動小数点でどれだけのメモリを消費するか。また、誤差逆伝播法が決してこれを構成しない理由を説明せよ。
2. **より深いネットワークのメモリ**。d 個の入力、q 個の出力、および K 個の層（そのうち $K-1$ 個の隠れ層は幅 h の ReLU 層）をもち、n 個のデータ例からなるミニバッチで訓練される MLP を考える。
    1. 本文中の K 層に対するステップを用いて、逆伝播が読み取る値を列挙し、その個数を求めよ。また、パラメータの個数を求めよ。
    2. 予測時には、次の層が使い終わり次第、各値を破棄できる。予測が任意の時点で保持する値の最大個数を求めよ。
    3. $d = 784$、$h = 256$、$q = 10$ のとき、保存される値の個数がパラメータ数を上回る最小のミニバッチサイズを、$K = 2$ および $K = 11$ のそれぞれについて求めよ。
3. [code] **勾配チェック**。自動微分が登場する以前、手書きの逆伝播は有限差分と比較して検証されていた。

1. 1 つの成分 w を $\mathbf{W}^{(1)}$ から選び、`mlp_backward` による勾配を、中心差分 $\big(J(w+\epsilon) - J(w-\epsilon)\big)/(2\epsilon)$ ($\epsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-7}$) と 32 ビット浮動小数点および 64 ビット浮動小数点のそれぞれで比較せよ。相対誤差を ϵ に対して両対数軸でプロットせよ。
2. 曲線の形状を説明せよ。ϵ を小さくするとき減少するのはどの誤差で、増加するのはどの誤差か。また、それぞれの精度において両者が釣り合うのはどこか。
3. 活性化前の値が 0 から ϵ 以内にあるとき、勾配が正しくてもこのチェックが失敗することがある。その理由と回避方法を説明せよ。

4. **フォワードモード**。誤差逆伝播法は出力に $\partial J/\partial J = 1$ をシードとして与え、グラフを逆方向にスイープする。これに対し、フォワードモード微分は選んだ 1 つのスカラーパラメータに $\partial w/\partial w = 1$（対象パラメータ w に対する微分）をシードとして与え、順方向にスイープする。各ノードは、入力の導関数から自身の値の w に関する導関数を計算する。

   1. フォワードモードを用いて、$\partial L/\partial \mathbf{W}^{(1)}$ を Equation 4.10 のネットワークについて再計算せよ（$\mathbf{W}^{(1)}$ の成分ごとに 1 回のスイープを行うこと）。
   2. このネットワークの 6 個すべての重み勾配を得るには、各モードで何回ずつスイープが必要か。リバースモードでは 1 回しか計算しない積のうち、フォワードモードが毎回のスイープで繰り返し計算するのはどの積か。

   Simon Prince『Understanding Deep Learning』問題 7.12 および 7.13 を改変。

5. [extended] **自動微分エンジン。**●小型の自動微分エンジンを構築し、それを用いて Equation 4.10 のネットワークの勾配を再計算せよ。

   1. 各計算結果について、その入力と生成元の演算（計算グラフ）を記録するスカラーの `Value` クラスを作成し、`+`、`*`、`relu` をサポートせよ（**ヒント：** `__add__` と `__mul__` を実装し、それぞれが親ノードへの参照と勾配を 1 ステップ伝播させる小さな関数を保持する新しい `Value` を返すようにする）。
   2. `backward()` を実装せよ。グラフをトポロジカルソートし、出力の勾配に 1 をシードとして与え、逆順にノードをたどって各ノードから親ノードへ勾配を渡すようにせよ。1 つの値が複数回使われる場合に勾配が**累積**される（`+=` であり、`=` ではない）ようにすること。
   3. 作成したエンジンを用いて Figure 4.9 の計算グラフと Equation 4.10 の計算例を再計算し（行列積はスカラー演算に展開せよ）、得られた勾配を本節で導出した勾配と比較せよ。
   4. スカラー入力 x から $v_i = f_i(x)$ ($i = 1, 2, 3$)、中間値 $m = v_1 + v_2 + v_3$、値 $p_j = g_j(m)$ ($j = 1, 2, 3$)、および損失 $L = p_1 + p_2 + p_3$ が順に生成される連鎖を考える。$\alpha_i = f_i'(x)$、$\beta_j = g_j'(m)$ とおく。経路上の和としての連鎖律は $3 \times 3 = 9$ 本の経路（x から L への経路）を列挙し、各経路が積 $\alpha_i \beta_j$ を寄与する。これに対し、作成したエンジンは $(\alpha_1 + \alpha_2 + \alpha_3)(\beta_1 + \beta_2 + \beta_3)$ を計算すること（m で勾配を 1 回だけ累積することによる）を示し、この共有によって深層ネットワークでも誤差逆伝播法が現実的な計算量に収まる理由を説明せよ。

   Andrej Karpathy の micrograd（`micrograd/engine.py`）を改変。

4.4 数値安定性と初期化

4.2 節では、隠れ層が 1 層の MLP の重みを $\mathcal{N}(0, 0.01^2)$ で初期化すると最初の 1 エポックは進みが鈍かったものの、その後ネットワークは持ち直して 87% の正解率に到達した。では、幅 256 の隠れ層を **10** 層積み重ね、同じ $\mathcal{N}(0, 0.01^2)$ で初期化するとどうなるだろうか。10 エポック経過しても損失は $\log 10 \approx 2.30$ からまったく動かず、正解率は当て推量と同じ 10% にとどまる。逆に、重みを $\mathcal{N}(0, 1)$ で初期化すると、活性化値（activation）と勾配が天文学的に大きくなり、

たった 1 回の SGD 更新で重みが大きく飛びすぎて、次の順伝播で `inf` や `NaN` へとオーバーフローしてしまう。

4.3 節で見たように、K 層のネットワークでは、順伝播の活性化値も逆伝播の勾配も、重みと傾きからなる K 個の行列の積を通過する。各層が信号の分散を保ち、その乗数を 1 付近に維持しない限り、信号は指数関数的にゼロへ**消失**するか、無限大へ**爆発**するかのいずれかになる。本節では、層ごとの分散の伝播を定量化して **Xavier** 初期化と **He** 初期化を導出し、Fashion-MNIST 上の 10 層 MLP で検証したうえで、隠れユニット間の置換対称性（permutation symmetry）を破るために初期重みが**ランダム**でなければならない理由を証明する。

4.4.1 勾配消失と勾配爆発

4.4.1.1 層にわたる積

K 層の MLP において、隠れ層が $\mathbf{Z}^{(k)} = \mathbf{H}^{(k-1)}\mathbf{W}^{(k)} + \mathbf{b}^{(k)}$ および $\mathbf{H}^{(k)} = \phi(\mathbf{Z}^{(k)})$（幅 h）で与えられるとき、4.3 節の逆伝播の漸化式は、活性化値の勾配を次の式によって 1 層下へ伝播させる。

$$\frac{\partial L}{\partial \mathbf{H}^{(k-1)}} = \left(\frac{\partial L}{\partial \mathbf{H}^{(k)}} \odot \phi'\big(\mathbf{Z}^{(k)}\big) \right) \mathbf{W}^{(k)\top}. \tag{4.12}$$

Equation 4.12 を最上位の隠れ層 $K-1$ から第 k 層まで展開すると、第 k 層における勾配は、行列と傾きからなる $K-1-k$ 個の因子の積として表される。

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{H}^{(k)}} = \frac{\partial L}{\partial \mathbf{H}^{(K-1)}} & \underbrace{\odot\, \phi'\big(\mathbf{Z}^{(K-1)}\big)\, \mathbf{W}^{(K-1)\top}}_{\text{layer } K-1} \\ & \cdots \underbrace{\odot\, \phi'\big(\mathbf{Z}^{(k+1)}\big)\, \mathbf{W}^{(k+1)\top}}_{\text{layer } k+1}. \end{aligned} \tag{4.13}$$

1 つのデータ例に対する上流の勾配行を $\mathbf{g}'$、下流の勾配行を $\mathbf{g}$ とし、幅 h の層の重み w_{ij} が平均 0、分散 τ^2 の独立同分布から抽出されるとする。初期化時点において重み、傾き、上流の勾配を近似的に独立とみなせば、各成分 $g_i = \sum_{j=1}^{h} w_{ij}\, \phi'(z_j)\, g'_j$ の二乗の期待値は次の式で与えられる。

$$\begin{aligned} \mathbb{E}[g_i^2] &= \sum_{j=1}^{h} \mathbb{E}[w_{ij}^2]\, \mathbb{E}[\phi'(z_j)^2]\, \mathbb{E}[(g'_j)^2] \\ &= \underbrace{h\, \tau^2\, \mathbb{E}\big[\phi'(z)^2\big]}_{\text{per-layer factor}}\, \mathbb{E}[(g'_j)^2]. \end{aligned} \tag{4.14}$$

t 個の層を通過すると、勾配の二乗平均（mean square）は $\big(h\,\tau^2\,\mathbb{E}[\phi'(z)^2]\big)^t$ 倍になる。活性化前の値（pre-activation）の分布が対称で活性化関数が ReLU の場合、$\phi'(z) \in \{0, 1\}$ は半分の確率で 1 になるため、$\mathbb{E}[\phi'(z)^2] = 1/2$ となり、層ごとの乗数は $\frac{1}{2}h\tau^2$ となる。幅 $h = 256$ では次のようになる。

- $\tau = 0.01$ の場合、乗数は $256 \times 10^{-4}/2 = 0.0128$ となり、9 段の伝播で勾配の二乗平均は $0.0128^9 \approx 10^{-17}$ 倍にまで縮小し、**勾配消失**（vanishing gradients）が生じる。
- $\tau = 1$ の場合、乗数は $256 \times 1/2 = 128$ となり、9 段の伝播で勾配の二乗平均は $128^9 \approx 10^{19}$ 倍にまで膨張し、**勾配爆発**（exploding gradients）が生じる。

- シグモイド活性化関数（$\sigma'(z) \le 1/4$ より $\mathbb{E}[\sigma'(z)^2] \le 1/16$）の場合、たとえ $h\tau^2 = 1$ であっても、各層を通るたびに勾配の二乗平均は少なくとも 16 分の 1 に縮小する (Bengio et al. 1994)。

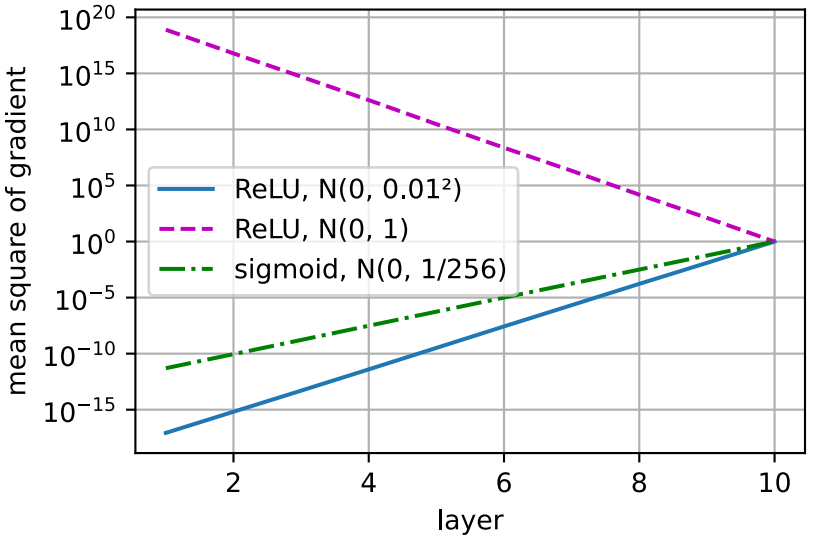
`moments` を用いて、初期化時点における 10 層・幅 256 のネットワーク上でこれら 3 つの予測を直接確認してみよう。

```
def moments(phi, dphi, scale, depth=10, width=256, n=256):
    """Mean squares of the activations and of their gradients, per layer."""
    Ws = [scale * jax.random.normal(d2l.get_key(), (width, width))
          for _ in range(depth)]
    H, Zs, acts = jax.random.normal(d2l.get_key(), (n, width)), [], []
    for W in Ws:  # forward pass, keeping the pre-activations
        Zs.append(H @ W)
        H = phi(Zs[-1])
        acts.append(float((H ** 2).mean()))
    G, grads = jax.random.normal(d2l.get_key(), (n, width)), []  # dL/dH
    for W, Z in zip(reversed(Ws), reversed(Zs)):  # backward pass
        grads.insert(0, float((G ** 2).mean()))
        G = (G * dphi(Z)) @ W.T  # dL/dH one layer down
    return acts, grads

relu = (jax.nn.relu, lambda Z: (Z > 0).astype(jnp.float32))
sigmoid = (jax.nn.sigmoid,
           lambda Z: jax.nn.sigmoid(Z) * (1 - jax.nn.sigmoid(Z)))
```

3 つの設定を比較しよう。すなわち、$\tau = 0.01$ の ReLU（4.2 節と同じ設定）、$\tau = 1$ の ReLU、そして $\tau^2 = 1/256$ のシグモイド（$h\tau^2 = 1$ として活性化関数のみによる減衰を切り出した設定）である。

```
runs = {'ReLU, N(0, 0.01²)': moments(*relu, 0.01),
        'ReLU, N(0, 1)': moments(*relu, 1.0),
        'sigmoid, N(0, 1/256)': moments(*sigmoid, 1 / 16)}
for name, (acts, grads) in runs.items():
    print(f'{name}: gradient mean square at layer 1: {grads[0]:.1e}')
d2l.plot(list(range(1, 11)), [grads for acts, grads in runs.values()],
         'layer', 'mean square of gradient', legend=list(runs),
         yscale='log', figsize=(4.5, 3))
```

```
ReLU, N(0, 0.01²): gradient mean square at layer 1: 8.5e-18
ReLU, N(0, 1): gradient mean square at layer 1: 7.9e+18
sigmoid, N(0, 1/256): gradient mean square at layer 1: 5.0e-12
```



縦軸を対数スケールにとると、3 本の曲線はいずれも直線となり、その傾きは解析的に導いた乗数 Equation 4.14 と一致する（第 1 層でそれぞれ 10^{-17}、10^{19}、5×10^{-12}）。

4.4.2 パラメータの初期化

4.4.2.1 Xavier 初期化

初期重みの分散 τ^2 をどのように選べば、すべての層の乗数を 1 にできるだろうか。線形層 $o_j = \sum_{i=1}^{n_{\text{in}}} x_i w_{ij}$ を考えよう。この層は n_{in} 個の入力（層の**ファンイン**）と n_{out} 個の出力（**ファンアウト**）を持ち、重み w_{ij} は平均 0、分散 τ^2 の独立同分布に従い、入力 x_i は互いに無相関で平均 0、分散 γ^2 を持つとする。順伝播では $\mathbb{E}[o_j] = 0$ であり、分散は次のようになる。

$$\text{Var}[o_j] = \mathbb{E}[o_j^2] = \sum_{i=1}^{n_{\text{in}}} \mathbb{E}[x_i^2]\,\mathbb{E}[w_{ij}^2] = n_{\text{in}}\,\tau^2\gamma^2.$$

層をまたいで順伝播の分散を保つには $n_{\text{in}}\tau^2 = 1$ が必要である。また、傾きを 1 とした Equation 4.14 より、逆伝播の勾配の分散を保つには $n_{\text{out}}\tau^2 = 1$ が必要となる。$n_{\text{in}} \neq n_{\text{out}}$ のとき、**Xavier 初期化**（または **Glorot 初期化** (Glorot and Bengio 2010)）はこれら 2 つのファン数を次のように平均する。

$$\frac{1}{2}(n_{\text{in}} + n_{\text{out}})\,\tau^2 = 1 \quad \Longleftrightarrow \quad \tau^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}}, \tag{4.15}$$

すなわち、重みを正規分布 $\mathcal{N}\left(0, \frac{2}{n_{\text{in}}+n_{\text{out}}}\right)$、またはそれと分散を揃えた一様区間 $U\left(-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right)$ から抽出する。Xavier 初期化は、線形層や tanh のようなゼロ中心で傾き 1 の活性化関数に対する標準的な選択肢である。

4.4.2.2 He 初期化

ReLU 層では、平均 0 で対称な活性化前の値 z の負の領域がゼロに切り捨てられるため、その二乗平均は半分になる（$\mathbb{E}[\mathrm{ReLU}(z)^2] = \frac{1}{2}\mathbb{E}[z^2]$）。重みが独立で平均 0 であれば、$\mathbb{E}[o_j^2] = n_{\text{in}}\tau^2\,\mathbb{E}[x_i^2]$ は入力 $x_i = \mathrm{ReLU}(z_i)$ が正の平均を持つ場合にも成り立つため、活性化値の二乗平均 $v^{(k)} = \mathbb{E}[(h_i^{(k)})^2]$ は ReLU 層を通過すると次のように変化する。

$$v^{(k)} = \frac{1}{2}\, n_{\text{in}}\, \tau^2\, v^{(k-1)}. \tag{4.16}$$

したがって、深さにわたって $v^{(k)} = v^{(k-1)}$ を一定に保つには、分散を 2 倍にする必要がある。

$$\tau^2 = \frac{2}{n_{\text{in}}}. \tag{4.17}$$

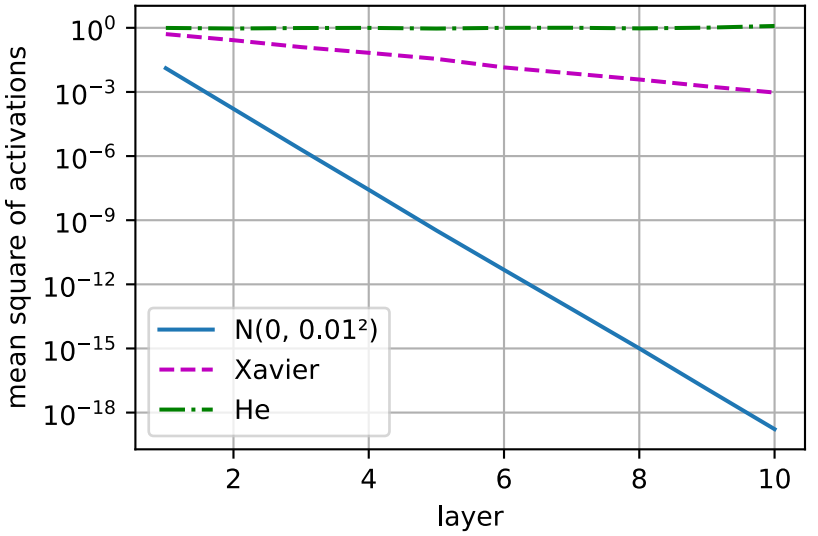
これが **He 初期化**（または **Kaiming 初期化** (He et al. 2015)）であり、ReLU ネットワークにおける標準の手法である（なお、フレームワークの標準的な層では、He 初期化を明示的に指定しない限り、デフォルトが $1/(3n_{\text{in}})$ や $1/n_{\text{in}}$、あるいは Xavier 初期化になっていることが多い）。

4.4.2.3 深さにわたる活性化値、勾配、訓練

まず、$\mathcal{N}(0, 0.01^2)$、Xavier（$\tau^2 = 1/256$）、He（$\tau^2 = 2/256$）を、10 層の合成 ReLU 検証コード `moments(*relu, ...)` で比較してみよう。

```
runs = {'N(0, 0.01²)': moments(*relu, 0.01),
        'Xavier': moments(*relu, (1 / 256) ** 0.5),
        'He': moments(*relu, (2 / 256) ** 0.5)}
for name, (acts, grads) in runs.items():
    print(f'{name}: activations at layer 10: {acts[-1]:.1e}, '
          f'gradient at layer 1: {grads[0]:.1e}')
d2l.plot(list(range(1, 11)), [acts for acts, grads in runs.values()],
         'layer', 'mean square of activations', legend=list(runs),
         yscale='log', figsize=(4.5, 3))
```

```
N(0, 0.01²): activations at layer 10: 1.7e-19, gradient at layer 1: 9.8e-18
Xavier: activations at layer 10: 9.4e-04, gradient at layer 1: 1.9e-03
He: activations at layer 10: 1.2e+00, gradient at layer 1: 1.2e+00
```



$\mathcal{N}(0, 0.01^2)$ では第 10 層の活性化値が $\sim 10^{-19}$ まで潰れてしまい、Xavier でも各 ReLU 層で活性化値の二乗平均が半減する（第 10 層までに $\sim 2^{-10} \approx 10^{-3}$）のに対し、He 初期化では活性化値と勾配の両方が $O(1)$ 付近に保たれている点に注目しよう。この分散の伝播が実際の訓練をどのように左右するかを確かめるため、隠れ層 10 層の ReLU 分類器 `DeepMLP` を定義し、層ごとの標準偏差の規則 `std(n_in, n_out)` を引数として渡せるようにする。

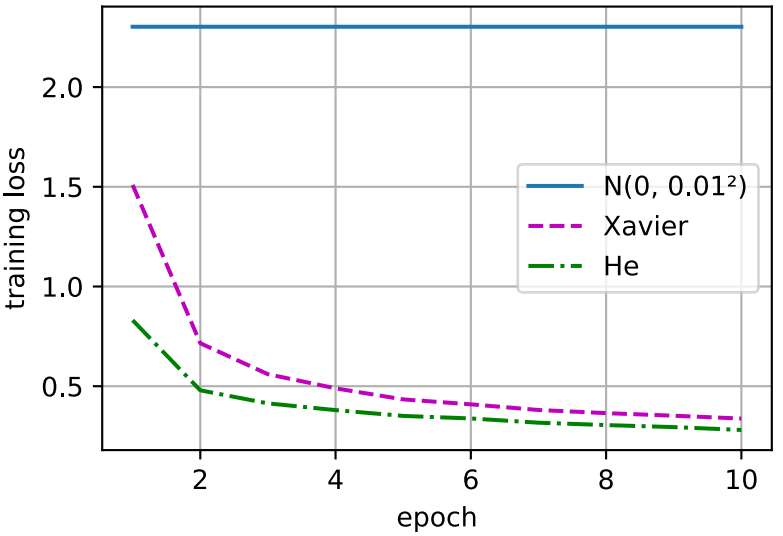
```
class DeepMLP(d2l.Classifier):
    """A deep MLP whose weights have the standard deviation std(n_in, n_out)."""
    def __init__(self, std, lr, depth=10, width=256, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['std', 'rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        sizes = [784] + [width] * depth + [10]
        layers = []
        for n_in, n_out in zip(sizes[:-1], sizes[1:]):
            normal = nnx.initializers.normal(std(n_in, n_out))
            layers += [nnx.Linear(n_in, n_out, kernel_init=normal, rngs=rngs),
                       nnx.relu]
        self.net = nnx.Sequential(*layers[:-1])  # No ReLU after the logits

    def forward(self, X):
        return self.net(X.reshape((X.shape[0], -1)))  # Flatten each image
```

3 つの実行はすべてまったく同じアーキテクチャとハイパーパラメータを共有しており、異なるのは `std(n_in, n_out)` のみである。これらの訓練曲線と最終的な検証正解率を比較しよう。

```
inits = {'N(0, 0.01²)': lambda n_in, n_out: 0.01,
         'Xavier': lambda n_in, n_out: (2 / (n_in + n_out)) ** 0.5,
         'He': lambda n_in, n_out: (2 / n_in) ** 0.5}
data = d2l.FashionMNIST(batch_size=256)
losses = {}
for name, std in inits.items():
```

```
    trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
    trainer.board.display = False
    trainer.fit(DeepMLP(std, lr=0.05), data)
    losses[name] = trainer.history['train_loss']
    print(f'{name}: validation accuracy {trainer.history["val_acc"][-1]:.2f}')
d2l.plot(list(range(1, 11)), list(losses.values()), 'epoch', 'training loss',
         legend=list(losses), figsize=(4.5, 3))
```

```
N(0, 0.01²): validation accuracy 0.10
Xavier: validation accuracy 0.81
He: validation accuracy 0.86
```



10 エポックにわたる訓練曲線が、先ほどの分散解析とどれほどよく一致しているか見てみよう。

- $\mathcal{N}(0, 0.01^2)$ では、第 10 層の活性化値（$\sim 10^{-19}$）と第 1 層の勾配（$\sim 10^{-17}$）が完全に消失し、訓練損失は $\log 10 \approx 2.30$ に張り付いたまま、検証正解率も 10% にとどまる。
- Xavier では、ReLU 層ごとに信号が 2 分の 1 へと減衰して全体で $2^{-10} \approx 10^{-3}$ 倍に縮小するため、最初の 1 エポックは立ち上がりが遅いものの、その後訓練が回復して 80% を超える正解率に達する。
- He 初期化では、全 10 層にわたって活性化値と勾配の両方がオーダー 1 付近に保たれ、最初のエポックから訓練損失が滑らかに低下する。

もっとも、初期化だけで深さの問題がすべて解決するわけではない。ここまでの分散の議論はステップ 0 におけるネットワークを記述しているにすぎず、ひとたび SGD が重みを動かせば、Equation 4.14 の層ごとの乗数を 1 付近に保ち続ける保証は何もない。そのため、数十層から数百層に及ぶネットワークでは、訓練全体を通して機能する仕組みに頼ることになる。正規化層（normalization layer、7.3 節）は毎ステップ活性化値のスケールを調整し、残差接続（residual connection、7.4 節）は信号と勾配に各層を迂回する経路を与え、勾配クリッピング（gradient clipping、8.4 節）は時折生じる勾配爆発に上限を設ける。適切な初期化は訓練を軌道に乗せるためのものであり、その後の信号のスケールを維持するのはこれらの仕組みである。

4.4.3 隠れユニット間の対称性

なぜ $\mathbf{W}^{(1)}$ のすべての成分を決定論的な定数 $c = \sqrt{2/n_{\text{in}}}$ に設定するのではなく、初期重みを**ランダム**にしなければならないのだろうか。理由は、同一層内の隠れユニットが置換対称だからである。もし $\mathbf{W}^{(1)}$ のすべての列が同一の値から始まり（かつ $\mathbf{W}^{(2)}$ のすべての行も同一の値から始まり）、そのまま訓練を進めると、どの隠れユニットもまったく同じ順伝播の活性化値を計算し、同じ逆伝播の勾配を受け取り、その後もずっと同じ SGD 更新ステップをたどり続けるため、256 ユニットの層が実質的に 1 個のニューロンへと縮退してしまう。16 ユニットの層を用いて、He のスケールに合わせた定数の重みでは勾配の各列が同一になる（ユニット間の差が `0.00`）のに対し、He のスケールに合わせたランダムな重みでは各ユニットが異なる勾配方向を持つことを確認してみよう。

```
def hidden_grad(W1, W2, X, y):
    """Gradient of the squared loss with respect to the hidden weights."""
    L = lambda W1: ((jax.nn.relu(X @ W1) @ W2 - y) ** 2).mean() / 2
    return jax.grad(L)(W1)

d, h = 20, 16
X = jax.random.normal(d2l.get_key(), (64, d))
y = jax.random.normal(d2l.get_key(), (64, 1))
c, c_out = (2 / d) ** 0.5, (1 / h) ** 0.5
inits = {'constant': (jnp.full((d, h), c), jnp.full((h, 1), c_out)),
         'random': (c * jax.random.normal(d2l.get_key(), (d, h)),
                    c_out * jax.random.normal(d2l.get_key(), (h, 1)))}
for name, (W1, W2) in inits.items():
    G = hidden_grad(W1, W2, X, y)
    print(f'{name}: largest difference between two hidden units '
          f'{float(jnp.abs(G - G[:, :1]).max()):.2f}')
```

```
constant: largest difference between two hidden units 0.00
random: largest difference between two hidden units 0.09
```

4.4.4 まとめ

K 層の MLP（幅 h、初期重みの分散 τ^2）では、各層が活性化値と勾配の二乗平均をおよそ $h\tau^2\,\mathbb{E}[\phi'(z)^2]$ 倍するため、この乗数が 1 付近に保たれない限り、信号は深さに対して指数関数的に消失または爆発する。Xavier 初期化は $\tau^2 = 2/(n_{\text{in}} + n_{\text{out}})$ と設定することで線形層や tanh 層における順伝播と逆伝播の分散のバランスをとり、一方 He 初期化は分散を 2 倍の $\tau^2 = 2/n_{\text{in}}$ とすることで ReLU が負の半直線をゼロに切り捨てる影響を相殺し、深い ReLU 積層でも最初のエポックから安定して訓練できるようにする。また、同一層内の隠れユニットは置換対称であるため、重みは定数にするのではなく独立かつランダムに抽出しなければならない。そうしなければ、すべてのユニットが訓練を通じて同一の勾配と更新を受け続けることになる。さらにステップ 0 以降についても、より深いアーキテクチャでは分散保持の初期化に正規化層や残差接続を組み合わせることで、重みが変化しても信号のスケールを適切に維持する。

4.4.5 演習問題

1. **スペクトルノルムによる上界**。1 つのデータ例について列ベクトルとして表記すると、Equation 4.12 のステップは、勾配 $\mathbf{g}'$（第 k 層より上流）に行列 $\mathbf{M}^{(k)} = \mathbf{W}^{(k)}\,\text{diag}\big(\phi'(\mathbf{z}^{(k)})\big)$ を掛ける操作となる。ここで $\text{diag}(\cdot)$ はベクトルの各成分を対角成分に並べ、他の成分をゼロとした行列を表す。したがって、Equation 4.13 の勾配は、このような $K-1-k$ 個の

行列の積と最上位層の勾配との積になる。スペクトルノルム（最大特異値）は劣乗法的であり、$\|\mathbf{AB}\|_2 \le \|\mathbf{A}\|_2 \|\mathbf{B}\|_2$ を満たす。

1. 定義 $\|\mathbf{A}\|_2 = \max_{\mathbf{x}\neq 0} \|\mathbf{Ax}\|/\|\mathbf{x}\|$ から劣乗法性を証明せよ。
2. これを用いて、勾配のノルムを各因子のノルム $\|\mathbf{M}^{(k)}\|_2$ によって上から評価せよ。深さに対してこの上界が幾何級数的に増大しないためには、これらのノルムがどのような条件を満たせばよいか。
3. この評価はあくまで上界にすぎない。前問の条件によって勾配爆発は排除できるが勾配消失は排除できない理由を説明し、さらにその条件を等号で満たすにもかかわらず、積のスペクトルノルムが真に 1 より小さくなる 2 つの 2×2 行列の例を挙げよ。

2. **その他の対称性**。MLP が表す関数を不変に保つパラメータの変化は、隠れユニットの置換だけではない。隠れ層が 1 層の MLP を考え、隠れユニット j の入力重みを $\mathbf{w}_j$、バイアスを b_j、出力重みを $\mathbf{v}_j$ とする。
   1. $\mathrm{ReLU}(\alpha z) = \alpha\,\mathrm{ReLU}(z)$ が $\alpha > 0$ に対して成り立つことを証明せよ。この恒等式を用いて、$(\mathbf{w}_j, b_j, \mathbf{v}_j)$ を $(\alpha\mathbf{w}_j, \alpha b_j, \mathbf{v}_j/\alpha)$ に置き換えても ReLU ネットワークの関数が変化しないことを示せ。
   2. 単一の隠れユニットのパラメータに対する変換であって、tanh ネットワークの関数を不変に保つものを求めよ。
   3. 同一の隠れユニットからなる状態は置換の不動点であり、その不動点から開始した勾配降下法はそこにとどまり続ける。前問の変換によって不変に保たれるユニットのパラメータを求めよ。さらに、tanh ネットワークのすべての重みとバイアスをゼロで初期化すると、勾配降下法はどのステップにおいても出力バイアスしか変化させないことを示せ。

   Simon Prince『Understanding Deep Learning』問題 3.5、4.3、7.15 より改変。
3. **ファンインと平均ファン数の比較**。@He.Zhang.Ren.ea.2015 に従い、K 層の線形ネットワークを考える。第 k 層は n_{k-1} 個の入力を n_k 個の出力へ写し、その重みは分散 τ_k^2 で抽出されるとする。4.4.2.1 節の計算より、第 k 層は順伝播信号の分散を $n_{k-1}\tau_k^2$ 倍し、逆伝播信号の分散を $n_k\tau_k^2$ 倍する。
   1. すべての層で $\tau_k^2 = 1/n_{k-1}$ と設定する。このとき任意の深さにおいて、順伝播の分散が保たれ、逆伝播の分散は全体で n_K/n_0 倍変化することを示せ。
   2. 各層の幅が $100, 400, 100, 400, \ldots$ と交互に並ぶものとし、すべての層で Xavier 初期化を用いるとする。$100 \to 400 \to 100$ という 1 組の層が順伝播の分散を何倍にするか、またその組を 50 回繰り返した後に何倍になるかを計算せよ。
   3. あらゆる幅の系列に対して、順伝播と逆伝播の両方の分散を初期値から深さに依存しない定数倍の範囲内に保つのは、2 つの規則のうちどちらか。
4. **ReLU のもとでの分散**。●4.4.2.2 節の導出では、分散ではなく二乗平均 $\mathbb{E}[\mathrm{ReLU}(z)^2]$ を追跡している。これは、次の層の活性化前の値 $o = \sum_i x_i w_i$ において、入力を $x_i = \mathrm{ReLU}(z_i)$ とし重みが独立で平均 0 であるとき、$\mathrm{Var}[o] = n_{\text{in}}\tau^2\mathbb{E}[x_i^2]$ が x_i の平均によらず成り立つからである。
   1. $z \sim \mathcal{N}(0, \nu^2)$ に対して $\mathbb{E}[\mathrm{ReLU}(z)]$ と $\mathrm{Var}[\mathrm{ReLU}(z)]$ を計算し、分散が $\left(\frac{1}{2} - \frac{1}{2\pi}\right)\nu^2$ となり、$\frac{1}{2}\nu^2$ にはならないことを示せ。
   2. 一見もっともらしい近道として、Xavier の計算で $\mathrm{Var}[x_i]$ を $\mathbb{E}[x_i^2]$ の代わりに代入する方法が考えられる。この近道からは $\tau^2 = 2/(n_{\text{in}}(1 - 1/\pi))$ が導かれ、$2/n_{\text{in}}$ とはならないことを示せ。さらに、この方法で初期化した各 ReLU 層が実際には二乗平均を何倍にするか、また 50 層全体で何倍になるかを計算せよ。
5. [code] **Tanh ネットワーク**。●4.4.2.3 節の計測を、ReLU の代わりに tanh を用い、Xavier 初期化かつ 50 層の設定で繰り返せ。このとき、1 つの層は入力の二乗平均 m を $m' = \mathbb{E}[\tanh(z)^2]$ へ写す（ここで z は近似的に $\mathcal{N}(0, m)$ に従う）。
   1. $\tanh(z)^2 \approx z^2 - \frac{2}{3}z^4$（小さい z での近似）を用いて、$m' \approx m - 2m^2$ となること

を示せ。多くの層を経たとき、二乗平均は一定に保たれるか、幾何級数的に減衰するか、それともより緩やかに減衰するか。また、第 k 層における二乗平均が約 $1/(2k)$（大きい k に対して）となることを示せ。
2. 実測した二乗平均を層に対して対数軸上でプロットし、この予測を確かめよ。勾配には何が起きるか。

4.5 深層学習における汎化

2.6 節では、20 個のパラメータを持つ 19 次の多項式がノイズを含む 20 個のデータ点を補間（interpolation）し、テストデータ上で発散した。この古典的な直感に従えば、4.2 節の Fashion-MNIST 用 MLP は 203,530 個のパラメータを 60,000 枚の訓練画像に対して持つため（画像 1 枚あたり 3.4 個のパラメータ）、とりわけすべての訓練画像を補間するまで訓練した場合には激しく過学習するはずである。ところが実際には、未知の画像に対して $\sim 87\%$ の正解率に達した。さらに以下で見るように、同じアーキテクチャにわずか 1,000 枚の訓練画像しか与えず、訓練正解率が 100% になるまで訓練しても、なお未知の画像の約 80% を正しく分類する。

極度に過剰パラメータ化（overparameterization）されたネットワークがなぜ汎化（generalization）するのかを理解することは、現代の機械学習における中心的な未解決問題の 1 つである。本節では、統制された実験と解析可能なモデルに対する厳密な結果を組み合わせ、このパズルを解く 4 つの重要な手がかりを明らかにする。

1. **ランダムラベルの記憶：**標準的な MLP は、完全にランダムなラベルに対しても容易に訓練正解率 100% に達する (Zhang et al. 2021)。この事実は、分布に依存しない古典的な容量上界（VC 次元など）では実際のラベルに対する汎化を説明できないことを証明している。
2. **二重降下：**モデル幅を補間閾値（interpolation threshold）$h = n$ を超えて変化させると、特徴行列がほぼ特異になる $h = n$ でテスト誤差が急増し、その後、余分なパラメータによってよりノルムの小さい補間解が可能になるにつれて**第 2 の降下**が現れる (Belkin et al. 2019)。
3. **勾配降下法の暗黙的バイアス：**損失がゼロになる無数の解の中から、勾配降下法（gradient descent）は二乗損失のもとでは最小ノルム補間解を、ロジスティック損失のもとでは最大マージン分離器を自動的に選択する (Soudry et al. 2018)。
4. **クリーンなパターンの優先学習と早期停止：**ノイズを含むデータセットでは、勾配降下法は破損したラベルを記憶するよりずっと前に、クリーンな構造的パターンに適合する (Arpit et al. 2017)。そのため、**早期停止**（early stopping）が強力な正則化手法となる。

4.5.1 記憶と容量

4.5.1.1 ランダムラベル

Zhang et al. (2021) に倣って生の記憶容量（memorization capacity）を検証するため、Fashion-MNIST の訓練画像を 2,000 枚（訓練用に 1,000 枚、検証用に 1,000 枚）とテスト画像を 2,000 枚抽出し、補助クラス `ArrayData` と `Tracker`、および He 初期化を施した 256 ユニットの `MLP` を定義して、真のラベル（`fraction=0`）と完全にランダムなラベル（`fraction=1`）のそれぞれで 200 エポック訓練する。

```
data = d2l.FashionMNIST(batch_size=2000)
X, y = next(iter(data.train_dataloader()))          # 2,000 random images
X_test, y_test = next(iter(data.val_dataloader()))  # the first 2,000
X, y, X_test, y_test = (d2l.numpy(a) for a in (X, y, X_test, y_test))
X, X_test = X.reshape(2000, -1), X_test.reshape(2000, -1)
```

```
X_train, y_train, X_val, y_val = X[:1000], y[:1000], X[1000:], y[1000:]
```

以降の実験を簡潔に進めるため、2 つの軽量な補助クラスを用意する。`ArrayData` は NumPy 配列をシャッフルされたミニバッチにまとめ、`Tracker` は `Trainer` を継承して、名前付きデータセットの任意の辞書にわたってエポックごとの正解率を記録する。

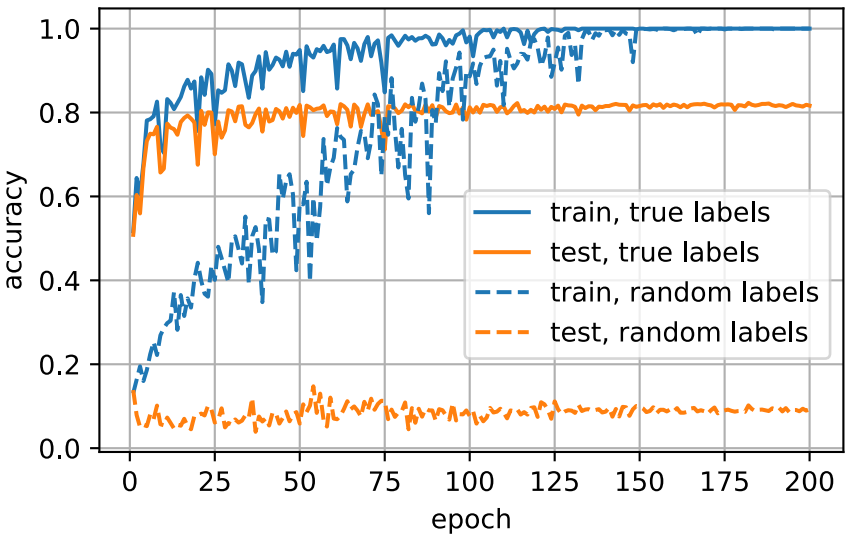
```
class ArrayData(d2l.DataModule):
    """Minibatches of the arrays X and y; training batches are shuffled."""
    def __init__(self, X, y, batch_size=64):
        super().__init__()
        self.save_hyperparameters()

    def get_dataloader(self, train):
        return self.get_tensorloader(
            (d2l.tensor(self.X), d2l.tensor(self.y)), train)

class Tracker(d2l.Trainer):
    """A Trainer that records the accuracy on named datasets every epoch."""
    def __init__(self, max_epochs, datasets):
        super().__init__(max_epochs, num_gpus=1)
        self.datasets = datasets
        self.accuracy = {name: [] for name in datasets}
        self.board.display = False  # We plot the accuracies ourselves

    def validate(self):
        for name, dataset in self.datasets.items():
            self.val_dataloader = dataset.val_dataloader()
            self.accuracy[name].append(super().validate()['acc'])
        return {}
```

モデルは 4.2 節と同じ 256 ユニットの ReLU MLP であるが、He 初期化（4.4.2.2 節）を `std=0.01` の代わりに用いることで、第 1 エポックから直ちに訓練が進むように改良している（バイアスはゼロとし、入力は事前に平坦化している）。

```
class MLP(d2l.Classifier):
    def __init__(self, num_hiddens, lr, num_inputs=784, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        self.net = nnx.Sequential(
            nnx.Linear(num_inputs, num_hiddens, kernel_init=he, rngs=rngs),
            nnx.relu,
            nnx.Linear(num_hiddens, 10, kernel_init=he, rngs=rngs))
```

次に、`corrupt(y, fraction, rng)` を定義して各訓練ラベルを $\{0, \ldots, 9\}$ から一様ランダムに選んだクラスへ確率 `fraction` で置き換え、`MLP(num_hiddens=256, lr=0.1)` を 200 エポックにわたって `fraction = 0`（真のラベル）と `fraction = 1`（純粋なランダムラベル）で訓練し、訓練正解率とテスト正解率の推移をプロットする。

```
def corrupt(y, fraction, rng):
    """Replace each label, with probability fraction, by a random class."""
    redraw = rng.random(len(y)) < fraction
    y = y.copy()
    y[redraw] = rng.integers(0, 10, redraw.sum())
    return y

rng = np.random.default_rng()
curves = []
for fraction in (0, 1):
    labels = corrupt(y_train, fraction, rng)
    trainer = Tracker(max_epochs=200, datasets={
        'train': ArrayData(X_train, labels, 1000),
        'test': ArrayData(X_test, y_test, 1000)})
    trainer.fit(MLP(num_hiddens=256, lr=0.1), ArrayData(X_train, labels))
    curves += [trainer.accuracy['train'], trainer.accuracy['test']]
d2l.plot(list(range(1, 201)), curves, 'epoch', 'accuracy',
         fmts=('C0-', 'C1-', 'C0--', 'C1--'), figsize=(5, 3))
d2l.plt.legend(['train, true labels', 'test, true labels',
                'train, random labels', 'test, random labels'],
               loc='center right', bbox_to_anchor=(1, 0.4));
```



この 2 回の実行結果は、次の事実を証明している。

- **真のラベル**では、訓練正解率が 60〜90 エポック以内に 100% に達する一方、テスト正解率は 80% 超で横ばいになる（わずか 1,000 枚の訓練画像での結果である）。
- **完全にランダムなラベル**でも、まったく同じアーキテクチャとオプティマイザで依然として訓練正解率 100% に達する（やや長く ～ 120〜150 エポックを要する）一方、テスト正解率は当て推量の 10% にとどまる。

4.5.1.2 容量上界の限界と帰納バイアス

この MLP は訓練誤差ゼロ（$\epsilon_S(f) = 0$）を、真の母集団誤差が $\epsilon(f) = 0.90$ であるランダムラベル上で達成するため、$\epsilon(f) \le \epsilon_S(f) + B(\mathcal{F}, n, \delta)$ という形をとり仮説クラス（hypothesis class）$\mathcal{F}$ のみに依存する分布に依存しない上界は、**いかなるものでも** $B(\mathcal{F}, n, \delta) \ge 0.90$ とならざるを得ず、実際のラベルに対しても空虚（vacuous）なものになる（演習問題 1）。

ノーフリーランチ定理（no free lunch theorem）(Wolpert 1996) により、あらゆる分布にわたって汎化できる学習器は存在しない。汎化するかどうかは、学習器の**帰納バイアス**（inductive bias）、すなわち無数にある補間関数のうちどれを選好するかが実データの構造と一致しているかにかかっている。これは、1 最近傍分類器が訓練データの 100% を補間しながらも、近傍の入力同士が同じラベルを共有していさえすれば良好に汎化するのと同様である (Cover and Hart 1967)。

4.5.2 二重降下

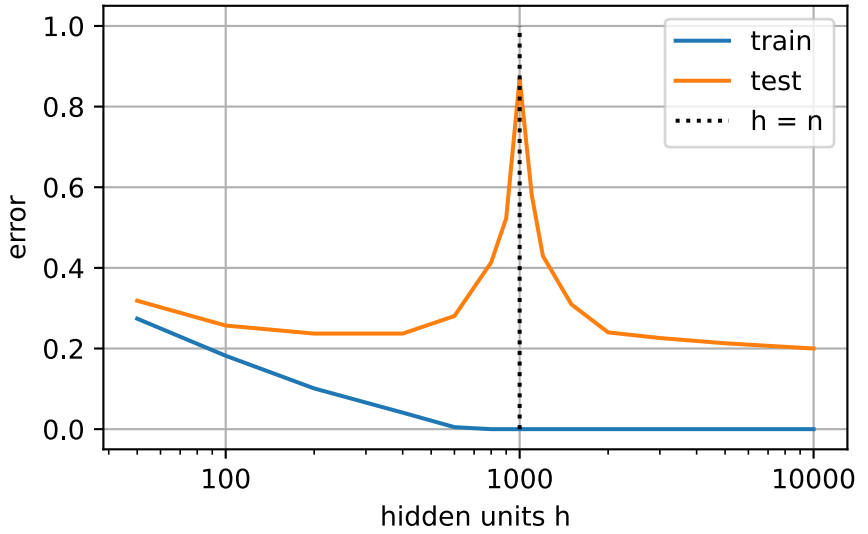
4.5.2.1 補間閾値と行列の条件

モデル幅 h をサンプルサイズ n をちょうど超えるまで増やすと何が起きるだろうか。解析しやすい**ランダム特徴量**（random-features）モデル (Rahimi and Recht 2007) で検証してみよう。He 初期化した第 1 層 $\mathbf{W}^{(1)} \in \mathbb{R}^{784 \times 10000}$ を固定し、中心化した $n = 1{,}000$ 枚の訓練画像から 10,000 個のランダム ReLU 特徴量 `H` と `H_test` を事前計算する。さらに、ラベルをワンホット行列 `Y` に符号化し、`error(H, W2, y)` を定義する。

```
W1 = rng.normal(0, np.sqrt(2 / 784), (784, 10000))  # He-initialized, frozen
mean = X_train.mean(axis=0)                         # The mean training image
H = np.maximum((X_train - mean) @ W1, 0)
H_test = np.maximum((X_test - mean) @ W1, 0)
Y = np.eye(10)[y_train]                             # One-hot labels

def error(H, W2, y):
    """Fraction of examples whose largest output is not their label."""
    return (np.argmax(H @ W2, axis=1) != y).mean()
```

ここで隠れ層の幅 `h` を 50 から 10,000 までの 15 通りの値にわたって変化させ、出力重み $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times 10}$ を先頭 `h` 個の特徴量の列 `H[:, :h]` に対して最小ノルム最小二乗法（`np.linalg.lstsq`）で当てはめつつ、最小特異値（singular value）`s_min[h]` とフロベニウスノルム `norm[h]` を記録する。

```
widths = [50, 100, 200, 400, 600, 800, 900, 1000, 1100, 1200, 1500, 2000,
          3000, 5000, 10000]
train_err, test_err, s_min, norm = [], [], {}, {}
for h in widths:
    W2, _, _, s = np.linalg.lstsq(H[:, :h], Y, rcond=None)
    train_err.append(error(H[:, :h], W2, y_train))
    test_err.append(error(H_test[:, :h], W2, y_test))
    s_min[h], norm[h] = s.min(), np.linalg.norm(W2)
d2l.plot([widths, widths, [1000, 1000]], [train_err, test_err, [0, 1]],
         'hidden units h', 'error', legend=['train', 'test', 'h = n'],
         xscale='log', fmts=('C0-', 'C1-', 'k:'), figsize=(5, 3))
```



これがよく知られた二**重降下**（double descent）曲線である (Belkin et al. 2019)。

1. **古典的な U 字曲線**（$h < n$）：テスト誤差は $\sim 30\%$（$h = 50$ のとき）から $\sim 23\%$（$h = 200$ 付近）まで下がるが、やがて $\sim 90\%$ へと、ちょうど**補間閾値** $h = n = 1{,}000$ において急上昇する。
2. **第 2 の降下**（$h > n$）：h が $n = 1{,}000$ を超えて 10,000 まで増えるにつれ、どのモデルも訓練セットを補間するにもかかわらず、テスト誤差は再び $\sim 20\%$ まで急落し、過少パラメータ化（underparameterized）された最良のモデルを上回る。

なぜテスト誤差は $h = n$ でこれほど激しく急増し、$h \gg n$ では回復するのだろうか。最小特異値 $s_{\min}$（$\mathbf{H}_{:,:h}$ に対する値）とフロベニウスノルム $\|\mathbf{W}^{(2)}\|_{\mathrm{F}}$ を確認してみよう。

```
for h in (200, 900, 1000, 1100, 10000):
    print(f'h = {h:5d}: smallest singular value {s_min[h]:.1e}, '
          f'norm of W2 {norm[h]:6.1f}')
```

```
h =   200: smallest singular value 1.7e+00, norm of W2    3.7
h =   900: smallest singular value 1.1e-01, norm of W2   22.2
h =  1000: smallest singular value 1.7e-03, norm of W2  410.4
h =  1100: smallest singular value 9.7e-02, norm of W2   24.2
h = 10000: smallest singular value 2.1e+00, norm of W2    2.4
```

$h = n = 1{,}000$ のとき、$\mathbf{H}$ は 1000×1000 の正方行列であり、最小特異値 $s_{\min}$ がきわめて小さいため、一意な厳密逆行列による解 $\mathbf{H}^{-1}\mathbf{Y}$ はラベルノイズを $s_{\min}$ で割ることになり、$\|\mathbf{W}^{(2)}\|_{\mathrm{F}}$ を数百以上にまで爆発させる。一方、$h \gg n$ では余分な列によって $(h - n)$ 次元の部分空間（subspace）をなす厳密な補間解が生じるため、**最小ノルム**解は悪条件（ill-conditioned）の方向を避けて $\|\mathbf{W}^{(2)}\|_{\mathrm{F}}$ を ~ 2 にまで縮小できる。実際、$h = 1{,}000$ を固定したまま訓練セットを $n = 500$ から $n = 1{,}000$ へと倍増させると、正則化なしの当てはめはちょうど $h = n$ の崖の上に乗り、テスト誤差は $\sim 30\%$ から 85–90% へと**上昇**してしまう。

```
for n in (500, 1000):
    W2 = np.linalg.lstsq(H[:n, :1000], Y[:n], rcond=None)[0]
```

```
    test_error = error(H_test[:, :1000], W2, y_test)
    print(f'{n} training images: test error {test_error:.2f}')
```

```
500 training images: test error 0.29
1000 training images: test error 0.87
```

4.5.3 勾配降下法の暗黙的バイアス

上の第 2 の降下は、損失がゼロになる無数の解の中から `lstsq` が**最小ノルム**補間解を選ぶことに依拠していた。驚くべきことに、勾配降下法にはまさにこれと同じ選好が組み込まれている。これは、明示的なペナルティ項がなくても働く**暗黙的バイアス**（implicit bias）、別名**暗黙的正則化**（implicit regularization）である。

4.5.3.1 二乗損失のもとでの最小ノルム補間

過剰パラメータ化された連立一次方程式 $\mathbf{H}\mathbf{w} = \mathbf{y}$（$h > n$、行フルランク）において、一意な最小ノルム補間解は $\mathbf{w}^\star = \mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top)^{-1}\mathbf{y}$ である。

命題。 $L(\mathbf{w}) = \frac{1}{2}\|\mathbf{H}\mathbf{w} - \mathbf{y}\|_2^2$ に対する勾配降下法は、初期値を $\mathbf{w}_0 = \mathbf{0}$ とすると（収束するステップサイズのもとで）最小ノルム補間解 $\mathbf{w}^\star$ に収束する。

証明。 すべての勾配 $\nabla L(\mathbf{w}) = \mathbf{H}^\top(\mathbf{H}\mathbf{w} - \mathbf{y})$ は $\mathbf{H}$ の行空間に属する。初期値を $\mathbf{w}_0 = \mathbf{0}$ とすると、各反復点 $\mathbf{w}_t$、ひいては極限 $\mathbf{w}_\infty = \mathbf{H}^\top\mathbf{a}$ も $\mathbf{H}$ の行空間にとどまる（Figure 4.12）。$\mathbf{H}\mathbf{w}_\infty = \mathbf{H}\mathbf{H}^\top\mathbf{a} = \mathbf{y}$ であるから、$\mathbf{a} = (\mathbf{H}\mathbf{H}^\top)^{-1}\mathbf{y}$ となり、$\mathbf{w}_\infty = \mathbf{w}^\star$ が成り立つ。他の任意の補間解 $\mathbf{w}^\star + \mathbf{v}$ は $\mathbf{H}\mathbf{v} = \mathbf{0}$ を満たし（したがって $\mathbf{v} \perp \mathbf{w}^\star$）、二乗ノルムは $\|\mathbf{w}^\star\|_2^2 + \|\mathbf{v}\|_2^2$ となって真に大きくなる。■

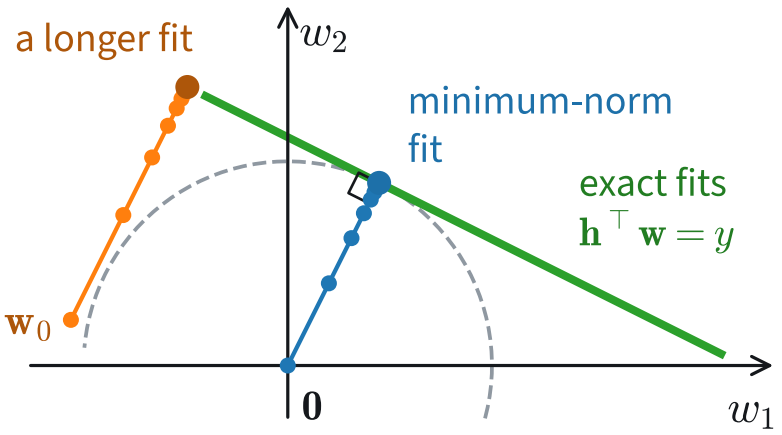


Figure 4.12: 劣決定な一次方程式 $\mathbf{h}^\top\mathbf{w} = y$ に対する勾配降下法は解超平面に対して厳密に垂直に動くため、$\mathbf{0}$（青）から出発すると最小ノルム解 $\mathbf{w}^\star$ に到達する一方、$\mathbf{w}_0 \neq \mathbf{0}$（オレンジ）から出発すると $\mathbf{w}_0$ の零空間成分がそのまま残る。

この命題を、式が 20 本、未知数が 100 個の系で数値的に検証し、$\mathbf{w}_0 = \mathbf{0}$ とランダム初期化を比較してみよう。

```
H_small = rng.normal(size=(20, 100))
y_small = rng.normal(size=20)
w_star = np.linalg.lstsq(H_small, y_small, rcond=None)[0]
lr = 1 / np.linalg.norm(H_small, 2) ** 2  # Below 2 / largest curvature
for name, w in (('zero', np.zeros(100)), ('random', rng.normal(size=100))):
    for _ in range(2000):
        w = w - lr * H_small.T @ (H_small @ w - y_small)
    residual = np.linalg.norm(H_small @ w - y_small)
    print(f'start {name:6s}: residual {residual:.0e}, norm '
          f'{np.linalg.norm(w):5.2f}, distance to w* '
          f'{np.linalg.norm(w - w_star):.1f}')
```

```
start zero  : residual 2e-16, norm  0.47, distance to w* 0.0
start random: residual 2e-15, norm  9.14, distance to w* 9.1
```

4.5.3.2 ロジスティック損失のもとでの最大マージン分類

線形分離可能な二値分類データ（$y_i \in \{-1, +1\}$）に対し、ロジスティック（交差エントロピー）損失 $\sum_i \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i))$ のもとでは $\|\mathbf{w}_t\|_2 \to \infty$（$t \to \infty$ のとき）となる。ここで Soudry et al. (2018) は、その**方向** $\mathbf{w}_t / \|\mathbf{w}_t\|_2$ が（$1/\log t$ の速さで）ハードマージン SVM の**最大マージン分離器**へ収束することを証明した（Figure 4.13）。これは、最小のマージンを持つサポートベクトルが $\log(1 + e^{-m})$ の指数関数的に減衰する裾を支配するためである（多層 ReLU ネットワークにおいて層のリスケーリングがパラメータノルムによるマージンをどのように複雑にするかについては、演習問題 5 を参照）。

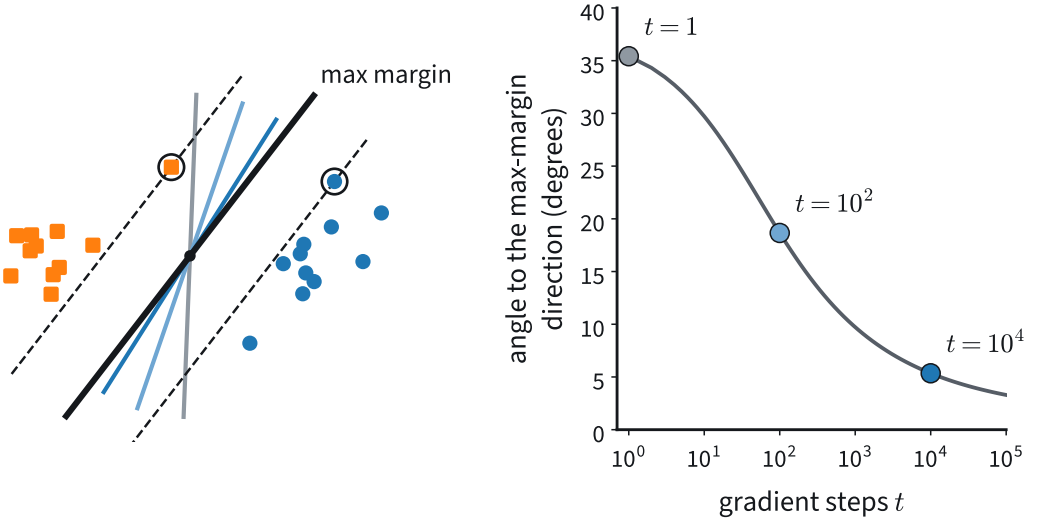
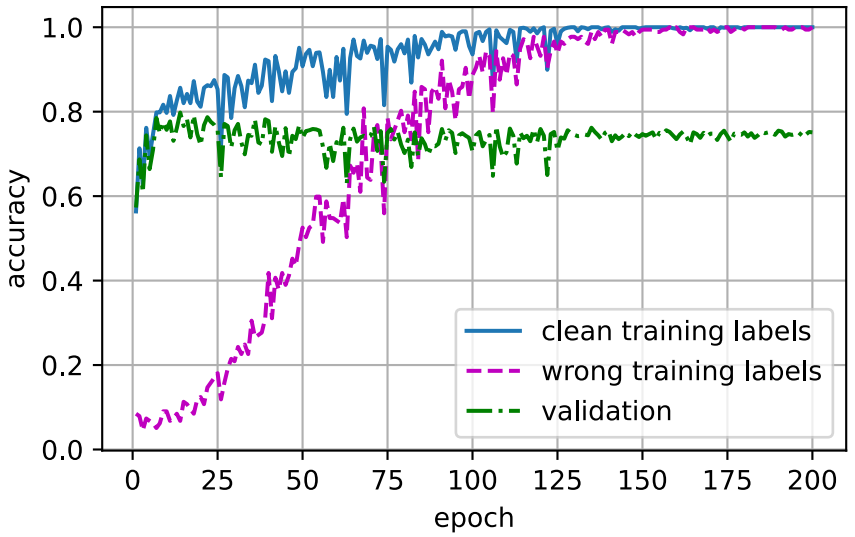


Figure 4.13: 線形分離可能なロジスティック回帰に対する勾配降下法は、$\mathbf{w}_t / \|\mathbf{w}_t\|_2$ を最大マージン超平面へ向けて対数的な速度で回転させ（左）、その回転は訓練正解率が 100% に達した後も長く続く（右）。

4.5.4 早期停止

破損したラベルを記憶するのに十分な容量を MLP が備えている場合でも、勾配降下法は**まずクリーンな構造的パターンを学習**し、**ラベルノイズはずっと後になって記憶**する (Arpit et al. 2017)。ここではラベルの 20%（画像 1,000 枚の訓練セット中）を破損させ、クリーンな訓練例、誤ラベル付きの訓練例（破損したラベルを基準に評価）、および 1,000 枚のクリーンな検証画像それぞれに対する正解率を 200 エポックにわたって個別に追跡してみよう。

```
labels = corrupt(y_train, 0.2, rng)
wrong = labels != y_train
trainer = Tracker(max_epochs=200, datasets={
    'clean': ArrayData(X_train[~wrong], labels[~wrong], 1000),
    'mislabeled': ArrayData(X_train[wrong], labels[wrong], 1000),
    'validation': ArrayData(X_val, y_val, 1000),
    'test': ArrayData(X_test, y_test, 1000)})
trainer.fit(MLP(num_hiddens=256, lr=0.1), ArrayData(X_train, labels))
acc = trainer.accuracy
d2l.plot(list(range(1, 201)), [acc['clean'], acc['mislabeled'],
                               acc['validation']], 'epoch', 'accuracy',
         legend=['clean training labels', 'wrong training labels',
                 'validation'], figsize=(5, 3))
```



エポック 20 までに、MLP はクリーンな訓練画像の 80% 超を正しく分類する一方で、破損したラベルの記憶は 20% 未満にとどまる。検証正解率はこの初期の区間でピークに達し、その後のエポック終盤におけるノイズの記憶によって押し下げられる。**早期停止**とは、指定した**ペイシェンス**の期間にわたって検証性能が改善しなくなった時点で訓練を打ち切り（演習問題 2）、検証性能が最良であったチェックポイントを復元する手法である。これにより計算量を節約しつつテスト正解率を数パーセントポイント回復できる（なお線形最小二乗法においては、ℓ_2 重み減衰（weight decay）と類似した働きをする。演習問題 4 を参照）。

```
best = int(np.argmax(acc['validation']))
print(f'best validation accuracy after epoch {best + 1}: '
      f'test accuracy {acc["test"][best]:.3f}')
print(f'after all 200 epochs: test accuracy {acc["test"][-1]:.3f}')
```

```
best validation accuracy after epoch 14: test accuracy 0.766
after all 200 epochs: test accuracy 0.735
```

早期停止の有名な例外として**グロッキング** (Power et al. 2022) がある（Figure 4.14、演習問題 6)。剰余加算のような小規模なアルゴリズム的タスクで重み減衰を伴ってネットワークを訓練すると、訓練データの対応表は急速に記憶される一方、検証正解率は当て推量付近のまま $100\times$ 長く停滞した後に突然 100% へと跳ね上がる。

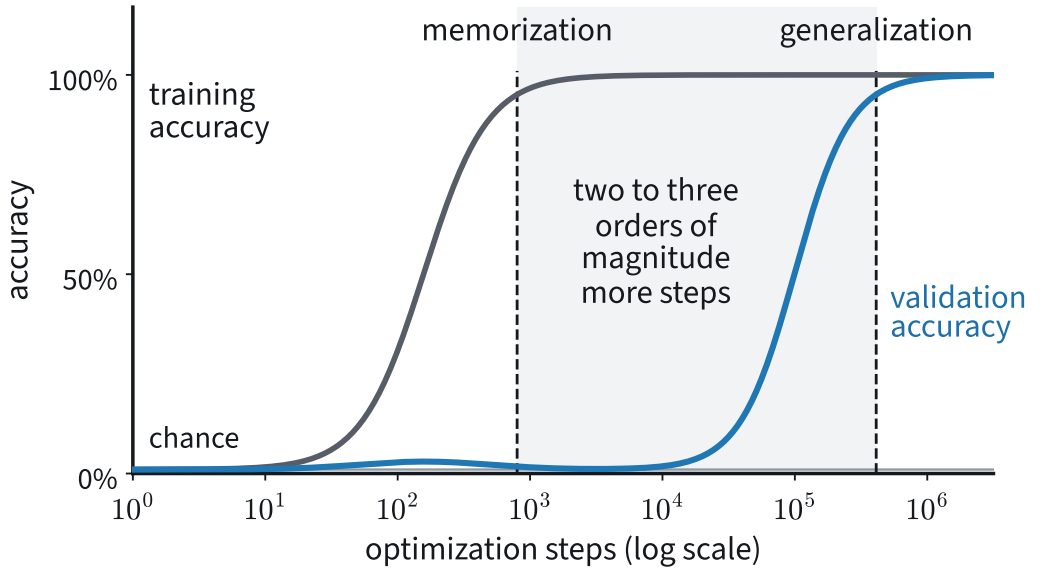


Figure 4.14: グロッキング（Power et al. (2022)）：モジュラ算術において、訓練正解率は $\sim 10^3$ ステップ以内に飽和する一方、検証正解率は $\sim 10^5$ ステップを経た後に初めて汎化へと跳ね上がる。

4.5.5 まとめ

過剰パラメータ化されたニューラルネットワークは、完全にランダムなラベルを記憶できる一方で構造を持つデータには良好に汎化するため、分布に依存しない容量上界は空虚なものとなり、汎化を説明する鍵は帰納バイアスに委ねられる。ランダム特徴量最小二乗法では、テスト誤差は補間閾値 $h = n$ で急増する。そこでは極端に小さい最小特異値 $s_{\min}(\mathbf{H})$ が $\|\mathbf{W}^{(2)}\|$ を膨張させるためであるが、$h \gg n$ になると最小ノルム補間が冗長な特徴量にわたって重みを分散させることで、テスト誤差は再び降下する。驚くべきことに、ゼロ初期化からの勾配降下法は、二乗損失のもとではこの最小ノルム補間解を自動的に見つけ出し、ロジスティック損失のもとでは方向において最大マージン分離器へと収束する。さらに勾配降下法は破損したラベルを記憶するより先に整合的なパターンへ適合するため、検証セットを用いた早期停止が単純かつ有効な正則化手法として機能する。

4.5.6 演習問題

1. **複雑さ尺度の限界**。あらゆるデータ分布に対して成り立つ汎化上界を考える。すなわち、少なくとも確率 $1-\delta$ で（n 個の訓練例の抽出に関して）、任意の分類器 f（クラス $\mathcal{F}$ の要素）が次式を満たすとする。

$$\epsilon(f) \le \epsilon_{\mathcal{S}}(f) + B(\mathcal{F}, n, \delta). \tag{4.18}$$

   ここで、$\epsilon(f)$ は母集団誤差、$\epsilon_{\mathcal{S}}(f)$ は訓練誤差であり、項 B は $\mathcal{F}$、n、δ のみに依存する（VC 上界における項 $c\sqrt{(v+\log(1/\delta))/n}$ と同様である）。本文では、このような上界に対して 1,000 枚の画像を用いたランダムラベルの議論を展開した。

   1. Fashion-MNIST の全 60,000 枚の訓練画像に対してラベルを一様ランダムに引き直したときにも、訓練によって $\mathcal{F}$ の中から訓練誤差ゼロの要素が見つかるとする（これは Zhang et al. (2021) が画像分類器について観察した通りである）。このとき $B(\mathcal{F}, 60{,}000, \delta)$ に対してどのような下界が導かれるか。また、その場合この上界は、真のラベルで訓練されたネットワークに対して何を保証するか。
   2. 近年の多くの汎化上界では、右辺の第 2 項が訓練の出力する分類器に依存することを許している（たとえば重みのノルムや、訓練例に対するマージンを通じて依存する）。この場合、上の議論のどの段階が成り立たなくなるか。また、真のラベルに対して意味のある主張を与えるためには、そのような量はランダムラベルでの実行時にどのような振る舞いをしなければならないか。

2. [code] **ペイシェンスに基づく停止**。`FashionMNIST` は Fashion-MNIST のテストセットを検証データとして使用しているため、代わりに 10,000 枚の訓練画像を検証セットとして取り分けておけ。残りの画像上で、256 ユニットの隠れ層を 1 つ持つ MLP を 4.2 節と同様に、過学習するまで十分に大きな固定エポック数だけ訓練し、各エポック終了時の検証損失とテスト正解率を記録せよ。許容幅 θ とペイシェンス k を持つペイシェンス規則は、検証損失がこれまでの最良値から θ を超えて改善しない状態が k エポック連続した時点で訓練を停止し、そこまでのうち検証損失が最小であったエポックのパラメータを採用する。

   1. θ を固定し、記録した曲線に対して $k \in \{1, 5, 20\}$ のそれぞれでこの規則を適用せよ。各 k について、停止エポック、採用されたエポック、およびその両方におけるテスト正解率を報告せよ。
   2. 採用されたエポックにおけるテスト正解率は k にどのように依存するか。その依存性を、エポックごとの検証損失の揺らぎと関連づけて説明せよ。

   Michael Nielsen『Neural Networks and Deep Learning』第 3 章より改作。

3. [code] **閾値における重み減衰**。二重降下の実験における最小ノルム解による当てはめを、リッジ解 $\mathbf{W}^{(2)} = (\frac{1}{n}\mathbf{H}^\top\mathbf{H} + \lambda\mathbf{I})^{-1}\frac{1}{n}\mathbf{H}^\top\mathbf{Y}$（2.7 節と同じく n 個の例にわたって二乗誤差の平均をとったもの）に置き換えよ。

   1. $\lambda \in \{10^{-5}, 10^{-3}, 10^{-1}\}$ に対して、幅に対するテスト誤差をプロットせよ。$h = n$ におけるピークはどうなるか。
   2. リッジ解が $\lambda \to 0$ のとき（$h > n$ において）最小ノルム解に近づくことを示せ。ヒント：$(\frac{1}{n}\mathbf{H}^\top\mathbf{H} + \lambda\mathbf{I})^{-1}\frac{1}{n}\mathbf{H}^\top = \mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top + n\lambda\mathbf{I})^{-1}$ と変形する。
   3. 各幅に対して、1,000 枚の検証画像を用いて λ を選択せよ。調整後のモデルの曲線は依然として 2 回降下するか。

4. **早期停止と重み減衰**。●重み減衰が重みの大きさを制限するのに対し、早期停止は訓練エポック数を制限する。線形最小二乗法ではこの対応関係を厳密に記述できる。$\mathbf{X}^\top\mathbf{X}/n$ が固有

値 $\mu_i > 0$ を持つとし、$\mathbf{w}^*$ を $L(\mathbf{w}) = \frac{1}{2n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ の最小点とし、L に対する勾配降下法を初期値 $\mathbf{w}_0 = \mathbf{0}$、学習率 $\eta < 1/\max_i \mu_i$ で実行する。

1. t ステップ後における $\mathbf{w}_t$ の第 i 固有ベクトル方向の座標が、係数 $1 - (1 - \eta\mu_i)^t$ を $\mathbf{w}^*$ の同じ座標に掛けたものに等しいことを示せ。
2. $L(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$ の最小点は、同じ座標を $\mu_i/(\mu_i + \lambda)$ 倍に縮小する（2.7.3 節）。λ を η と t の関数としてどのように選べば、これら 2 つの係数が $\eta\mu_i t \ll 1$ の方向と $\eta\mu_i t \gg 1$ の方向の両方で近似的に一致するか。

Ian Goodfellow、Yoshua Bengio、Aaron Courville『Deep Learning』7.8 節より改作。

5. **ロジスティック損失の暗黙的バイアス。**●線形分離可能なデータに対して、ロジスティック損失上の勾配降下法は方向において最大マージン分離器へ収束する (Soudry et al. 2018)。ラベルを $y_i \in \{-1, +1\}$ とする。
   1. 線形予測器 $f(\mathbf{x}) = \mathbf{w}^\top\mathbf{x}$ について、分離可能なデータ上では損失 $\sum_i \log\left(1 + \exp(-y_i\mathbf{w}^\top\mathbf{x}_i)\right)$ が任意の $\mathbf{w}$ に対して正である一方、その下限はゼロであることを示せ。ここから、勾配降下法が損失をゼロへ近づけるとき $\|\mathbf{w}_t\|$ は際限なく増大し、したがって方向 $\mathbf{w}_t/\|\mathbf{w}_t\|$ のみが収束し得ることを結論づけよ。
   2. 2 層の ReLU ネットワーク $f(\mathbf{x}) = \sum_j v_j \max(0, \mathbf{w}_j^\top\mathbf{x})$ について、$(\mathbf{w}_j, v_j)$ を $(c_j\mathbf{w}_j, v_j/c_j)$（ただし $c_j > 0$）に置き換えても f が変化しないことを示せ。このリスケーリング全体にわたる $\sum_j(\|\mathbf{w}_j\|^2 + v_j^2)$ の最小値を求めよ。また、ネットワークのパラメータノルムで測ったマージンが、線形予測器に対して $\|\mathbf{w}\|$ で測ったマージンとは異なり、関数 f のみで決まる性質ではない理由を説明せよ。
6. [extended] **グロッキング。**●●Power et al. (2022) の設定を再現せよ。すなわち、小規模なネットワーク（元論文では 2 層の Transformer を用いているが、ワンホット表現の組を入力とする幅広の MLP でもよい）を用い、$c = (a + b) \bmod 97$ を組 (a, b) から予測するよう、全ペアからランダムに選んだ 50% を訓練データとし、重み減衰を加えて 10^5 ステップ以上訓練せよ。ステップ数の対数に対して訓練正解率と検証正解率をプロットし、Figure 4.14 と比較せよ。汎化が起こるまでの遅れは、訓練データの割合や重み減衰の強さによってどのように変化するか。

4.6 ドロップアウト

過剰パラメータ化されたネットワークを希少なデータやノイズの多いデータで何エポックも訓練すると、やがて訓練データ特有の癖や破損したラベルまで暗記してしまう（4.5 節）。重み減衰（weight decay、2.7 節）が大きな重みにペナルティを課し、早期停止（early stopping、4.5.4 節）が終盤の暗記が始まる前に訓練を打ち切るのに対し、**ドロップアウト**は訓練中に乗法的なノイズを隠れ層へ直接注入することでネットワークを正則化する (Hinton et al. 2012; Srivastava et al. 2014)。すなわち、各順伝播において各隠れユニット（hidden unit）をそれぞれ独立に確率 p でゼロにすることで、特定のパートナーユニットに頼る**共適応**（co-adaptation）を起こさず、どのユニットにも単独で確実に機能する特徴を学習させる。テスト時にはドロップアウトを無効にし、ネットワーク全体を決定論的に動作させる。

本節では、逆ドロップアウト（inverted dropout）におけるマスク処理と再スケーリングを定式化し、Fashion-MNIST 上で隠れ層を 2 つ持つ MLP にゼロからの実装と簡潔な実装の両方を施す。さらに、ラベルを大幅に破損させたラベルノイズ実験を通じて、ドロップアウトがいかに暗記を遅らせ、検証正解率の低下をどれだけ食い止めるかを実証する。

4.6.1 隠れ層におけるノイズ

4.6.1.1 マスク処理と再スケーリング（逆ドロップアウト）

訓練中に入力へランダムなノイズを注入すると、摂動に強く滑らかな関数が学習されやすくなる(Bishop 1995)。実際、二乗誤差（squared error）のもとでの線形モデル $\mathbf{w}^\top \mathbf{x} + b$ において、入力に平均 0・分散 s^2 の独立なノイズを加えると、期待損失には $s^2\|\mathbf{w}\|_2^2$ が加算される。これは ℓ_2 重み減衰にほかならない。ドロップアウトは、ランダムな二値マスクによってこのノイズ注入を隠れ層へと拡張する。

隠れ層の活性化値（activation）h_j と**ドロップアウト確率** $p \in [0, 1)$ に対し、各訓練ステップで独立なベルヌーイマスク $\xi_j \in \{0, 1\}$（$P(\xi_j = 0) = p$、$P(\xi_j = 1) = 1 - p$）をサンプリングし、h_j を次の式で置き換える。

$$h_j' = \frac{\xi_j h_j}{1 - p} = \begin{cases} 0 & \text{with probability } p, \\ \dfrac{h_j}{1 - p} & \text{with probability } 1 - p. \end{cases} \tag{4.19}$$

訓練中に生き残った活性化値を $1 - p$ で割ること（**逆ドロップアウト**）により、期待値は不偏に保たれる。

$$\mathbb{E}[h_j'] = p \cdot 0 + (1 - p)\,\frac{h_j}{1 - p} = h_j, \qquad \mathrm{Var}[h_j'] = \frac{p}{1 - p}\,h_j^2.$$

訓練中に $\mathbb{E}[h_j'] = h_j$ が保たれるため、**推論時には重みの再スケーリングを一切せず、単にドロップアウトをオフにする（$h_j' = h_j$）だけでよい**。（なお、入力ドロップアウトを伴う線形回帰（linear regression）モデルでは、分散の式 $\mathrm{Var}[x_j'] = \frac{p}{1-p} x_j^2$ により、二乗損失（squared loss）に適応的なリッジペナルティ $\mathbb{E}[(\mathbf{w}^\top \mathbf{x}' + b - y)^2] = (\mathbf{w}^\top \mathbf{x} + b - y)^2 + \frac{p}{1-p}\sum_j x_j^2 w_j^2$ が加わる。一方、深い非線形ネットワークでは $\mathbb{E}[\mathrm{ReLU}(z)] \neq \mathrm{ReLU}(\mathbb{E}[z])$ であるため、テスト時の決定論的なネットワークは、間引かれた 2^h 個のサブネットワークからなるアンサンブルに対する実用的な近似となっている。演習問題 3 および 5 を参照。）

4.6.2 実装

4.6.2.1 ドロップアウト層

ゼロから実装する `dropout_layer(X, dropout)` は、区間 $[0, 1)$ の一様分布からテンソルをサンプリングし、値が $\geq$ `dropout` である要素を残して `1.0 - dropout` で割る（ただし `dropout == 1` のときはゼロのテンソルを返す）。

```
def dropout_layer(X, dropout, key):
    assert 0 <= dropout <= 1
    if dropout == 1: return jnp.zeros_like(X)
    mask = jax.random.uniform(key, X.shape) >= dropout
    return mask.astype(X.dtype) * X / (1.0 - dropout)
```

`dropout_layer` を 2×8 のテンソルに対し、ドロップアウト確率 0、0.5、1 でテストしてみよう。また、100 万個の 1 からなるベクトルに $p = 0.2$ を適用し、標本平均と標本分散が理論値である 1 および $p/(1-p) = 0.25$ と一致することを確かめる。

```
X = jnp.arange(16, dtype=jnp.float32).reshape(2, 8)
for p in (0, 0.5, 1):
    print(f'p = {p}:', dropout_layer(X, p, d2l.get_key()))
Y = dropout_layer(jnp.ones(10**6), 0.2, d2l.get_key())
print(f'mean {float(Y.mean()):.3f}, '
      f'variance {float(((Y - 1) ** 2).mean()):.3f}')
```

```
p = 0: [[ 0.  1.  2.  3.  4.  5.  6.  7.]
 [ 8.  9. 10. 11. 12. 13. 14. 15.]]
p = 0.5: [[ 0.  2.  0.  0.  0. 10. 12.  0.]
 [16.  0.  0.  0.  0.  0.  0. 30.]]
p = 1: [[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
mean 0.999, variance 0.251
```

4.6.2.2 モデルのゼロからの定義と訓練

各隠れ層の出力に対し、ReLU 活性化の直後にドロップアウトを適用する（Figure 4.15）。ただし、有効にするのは訓練中**のみ**である。以下では、He 初期化（He initialization）を施した隠れ層 2 層の分類器 `DropoutMLPScratch`（$784 \to 256 \to 256 \to 10$）を、層ごとに独立したドロップアウト確率 `dropout_1` と `dropout_2` を持たせて定義する。

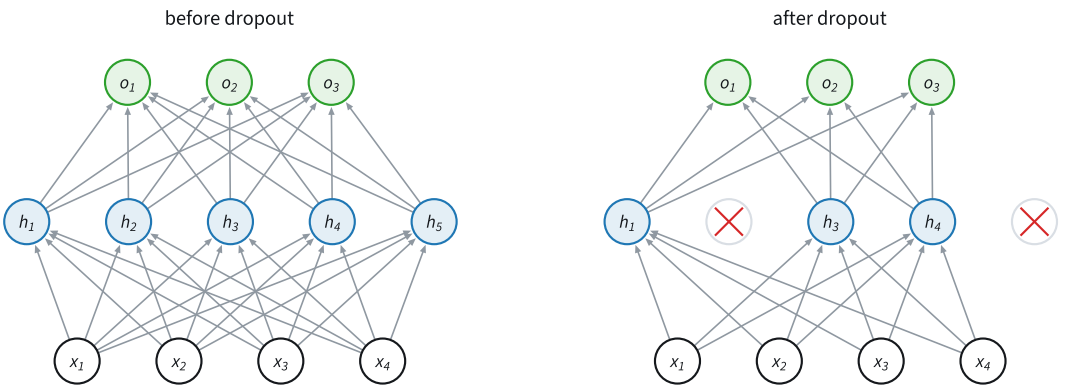


Figure 4.15: ドロップアウトは各訓練ステップにおいて、隠れユニットのランダムな部分集合（図では h_2 と h_5）およびそれらに接続する結合を無効化する。

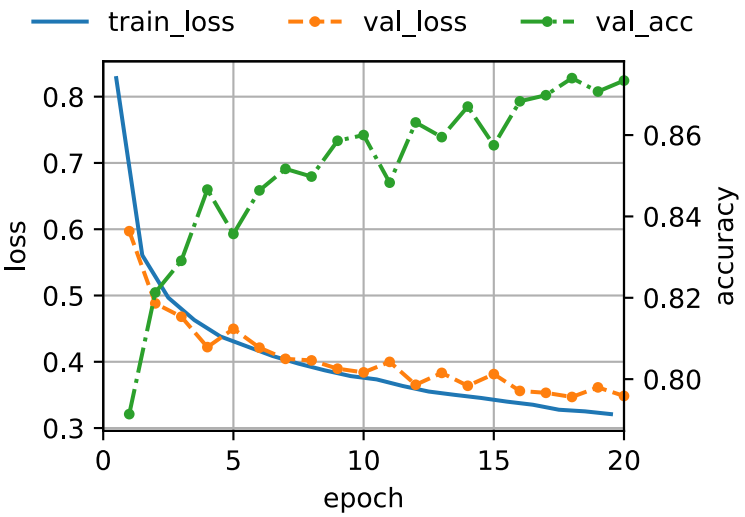
各層は `self.deterministic` を判定する。トレーナの評価用ビューはこれを `True` に設定する。

```
class DropoutMLPScratch(d2l.Classifier):
    def __init__(self, num_outputs, num_hiddens_1, num_hiddens_2,
                 dropout_1, dropout_2, lr, num_inputs=784, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        self.rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        he = lambda n_in, n_out: nnx.Param(  # Variance 2 / n_in
            self.rngs.params.normal((n_in, n_out)) * (2 / n_in) ** 0.5)
        self.W1 = he(num_inputs, num_hiddens_1)
        self.W2 = he(num_hiddens_1, num_hiddens_2)
```

```
        self.W3 = he(num_hiddens_2, num_outputs)
        self.b1 = nnx.Param(jnp.zeros(num_hiddens_1))
        self.b2 = nnx.Param(jnp.zeros(num_hiddens_2))
        self.b3 = nnx.Param(jnp.zeros(num_outputs))
        self.deterministic = False

    def set_view(self, *, deterministic):
        self.deterministic = deterministic

    def forward(self, X):
        X = X.reshape(-1, self.num_inputs)
        H1 = nnx.relu(X @ self.W1 + self.b1)
        if not self.deterministic:
            H1 = dropout_layer(H1, self.dropout_1, self.rngs.dropout())
        H2 = nnx.relu(H1 @ self.W2 + self.b2)
        if not self.deterministic:
            H2 = dropout_layer(H2, self.dropout_2, self.rngs.dropout())
        return H2 @ self.W3 + self.b3
```

ここで DropoutMLPScratch を、第 1 隠れ層の後に dropout_1 = 0.2、第 2 隠れ層の後に dropout_2 = 0.5、学習率（learning rate）に lr = 0.1 を指定してインスタンス化し（後で再利用するためハイパーパラメータは hparams にまとめておく）、Fashion-MNIST 上で 20 エポック訓練する。

```
hparams = {'num_outputs': 10, 'num_hiddens_1': 256, 'num_hiddens_2': 256,
           'dropout_1': 0.2, 'dropout_2': 0.5, 'lr': 0.1}
model = DropoutMLPScratch(**hparams)
data = d2l.FashionMNIST(batch_size=256)
trainer = d2l.Trainer(max_epochs=20, num_gpus=1)
trainer.fit(model, data)
```



最初の数エポックでは、訓練損失（training loss）が検証損失（validation loss）よりも上に位置していることがわかる。これは、訓練損失がノイズを伴うドロップアウトマスクを有効にした状態で評価されるのに対し、検証損失は決定論的なネットワーク全体を使って計算されるためであ

る。

4.6.2.3 簡潔な実装

フレームワークのドロップアウト層 `Dropout` を `Sequential` の内部で用いれば、まったく同じ隠れ層 2 層のアーキテクチャを数行で記述できる。

```
class DropoutMLP(d2l.Classifier):
    def __init__(self, num_outputs, num_hiddens_1, num_hiddens_2,
                 dropout_1, dropout_2, lr, num_inputs=784, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        self.net = nnx.Sequential(
            nnx.Linear(num_inputs, num_hiddens_1, kernel_init=he, rngs=rngs),
            nnx.relu, nnx.Dropout(dropout_1, rngs=rngs),
            nnx.Linear(num_hiddens_1, num_hiddens_2, kernel_init=he,
                       rngs=rngs),
            nnx.relu, nnx.Dropout(dropout_2, rngs=rngs),
            nnx.Linear(num_hiddens_2, num_outputs, kernel_init=he, rngs=rngs))

    def forward(self, X):
        return self.net(X.reshape((X.shape[0], -1)))  # Flatten each image
```

4.6.3 ラベルノイズ下におけるドロップアウトの効果

60,000 枚の画像すべてを含む Fashion-MNIST データセット全体で 20 エポック訓練する程度では、正則化なしの MLP でもほとんど過学習（overfitting）しないため、ドロップアウトが真価を発揮する余地は小さい。深刻な暗記にドロップアウトがどう対抗するかを観察するため、4.5.4 節で用いた 1,000 枚の画像からなるラベルノイズ付きベンチマークに立ち返り、`d2l.corrupt` によって訓練ラベルの 40% をランダムに破損させる（誤ラベルは全体の $\sim 36\%$ となり、ブールマスク `wrong = labels != y` で参照する）。

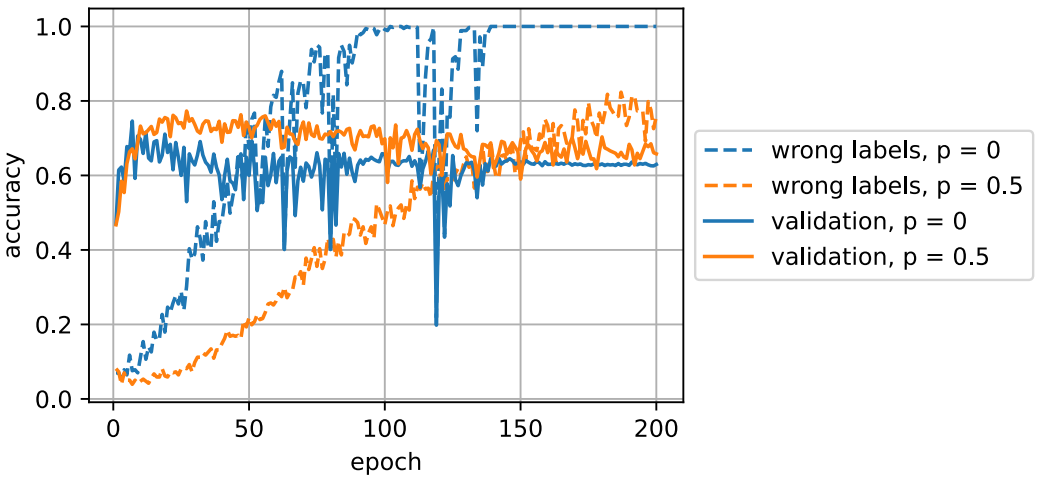
```
images = d2l.FashionMNIST(batch_size=2000)
X, y = (d2l.numpy(a) for a in next(iter(images.train_dataloader())))
X_val, y_val = (d2l.numpy(a) for a in next(iter(images.val_dataloader())))
X, y, X_val = X[:1000].reshape(1000, -1), y[:1000], X_val.reshape(2000, -1)
labels = d2l.corrupt(y, 0.4, np.random.default_rng())
wrong = labels != y  # The training labels that are now wrong
```

ここで `DropoutMLP` を 200 エポックにわたり `(X, labels)` 上で訓練し、両方の隠れ層のドロップアウト確率を `p = 0`（ドロップアウトなし）とした場合と `p = 0.5` とした場合を比較する。その際、`d2l.Tracker` を用いて、破損した訓練サブセット `X[wrong]` に対する正解率（accuracy）と、ホールドアウトされた `X_val` に対する検証正解率を記録する。

```
acc = {}
for p in (0, 0.5):
    trainer = d2l.Tracker(max_epochs=200, datasets={
        'wrong labels': d2l.ArrayData(X[wrong], labels[wrong], 1000),
```

```
            'validation': d2l.ArrayData(X_val, y_val, 1000)})
        trainer.fit(DropoutMLP(**{**hparams, 'dropout_1': p, 'dropout_2': p}),
                    d2l.ArrayData(X, labels))
        acc[p] = trainer.accuracy
        print(f'dropout {p}: wrong labels reproduced '
              f'{acc[p]["wrong labels"][-1]:.2f}, validation accuracy over the '
              f'last 10 epochs {np.mean(acc[p]["validation"][-10:]):.3f}')
    d2l.plot(list(range(1, 201)), [acc[0]['wrong labels'], acc[0.5]['wrong labels'],
                                   acc[0]['validation'], acc[0.5]['validation']],
             'epoch', 'accuracy', fmts=('C0--', 'C1--', 'C0-', 'C1-'),
             legend=['wrong labels, p = 0', 'wrong labels, p = 0.5',
                     'validation, p = 0', 'validation, p = 0.5'],
             legend_loc='outside', figsize=(4.5, 3))
```

```
dropout 0: wrong labels reproduced 1.00, validation accuracy over the last 10 epochs 0.630
dropout 0.5: wrong labels reproduced 0.76, validation accuracy over the last 10 epochs 0.663
```



2 つの軌跡を見てみよう。

- **ドロップアウトなし** ($p = 0$)：100 エポック目までに MLP は破損した訓練ラベルを 100% 暗記してしまい、検証正解率は序盤のピークから ~ 10 パーセントポイント低下する。
- **ドロップアウトあり** ($p = 0.5$)：誤ラベルの暗記が減速する。200 エポック目でも MLP が再現する誤ラベルはすべてではなく約 $70\text{〜}85\%$ にとどまり、最後の 10 エポックにおける検証正解率は正則化なしのネットワークより **3〜6 パーセントポイント高い**。もっとも、誤ラベルの正解率曲線はなお上昇を続けている。すなわち、ドロップアウトは悪影響を取り除くのではなく遅らせる手法であり、だからこそ早期停止と組み合わせて使われるのである。

実務上、ドロップアウトが最も威力を発揮するのは、パラメータ数（parameter count）がデータセットのサイズを圧倒的に上回り、かつ何エポックにもわたって訓練する場合である（7.1 節の AlexNet のような 2010 年代初頭の画像分類器がその典型である）。一方、データを 1 周のみ処理する基盤モデルの事前学習（pretraining）や、バッチ正規化（batch normalization）層の直前（7.3 節、訓練時にドロップアウトが分散を $1/(1-p)$ 倍に膨張させることで訓練時とテスト時の間に分散のずれが生じる (Li et al. 2019)）では、通常ドロップアウトを弱めるか省略する。

4.6.4 まとめ

逆ドロップアウトは、訓練中に各隠れ層の活性化値をそれぞれ独立に確率 p でゼロにし、生き残った活性化値を $1/(1-p)$ 倍に再スケーリングする。この再スケーリングは $\mathbb{E}[h'_j] = h_j$ を保ちつつ分散 $\frac{p}{1-p}h_j^2$ を注入するため、テスト時には重みの再スケーリングを一切せずにマスクなしのネットワーク全体を決定論的に動作させられる。パラメータ数がサンプル数を大きく上回る小規模なデータセットやノイズの多いデータセットでは、ドロップアウトによって隠れユニット間の脆い共適応が抑えられ、破損したラベルの暗記が遅くなり、ホールドアウト正解率が数パーセントポイント向上する。ただし、ドロップアウトは最終的な暗記を防ぐのではなく遅らせるものであるため、早期停止や重み減衰と併用するときに最も効果を発揮する。

4.6.5 演習問題

1. [code] **ドロップアウト確率**。本節のモデルは、第 1 隠れ層の後に確率 0.2 で、第 2 隠れ層の後に確率 0.5 でユニットをドロップする。
    1. 2 つの確率を入れ替えた場合、両方を 0.5 に設定した場合、両方を 0.8 に設定した場合のそれぞれについてモデルを訓練せよ。各実行における最良の検証正解率を報告せよ。より高いドロップアウト確率に耐えられるのはどちらの層か。
    2. `Trainer` がプロットする訓練損失は、マスクをオンにした状態で計算されている。訓練後にマスクをオフにした状態で訓練画像に対する損失を評価し、ドロップアウトありで訓練したネットワークとドロップアウトなしで訓練したネットワークとで、訓練損失と検証損失のギャップを比較せよ。
2. **ドロップアウトと He 初期化**。He 初期化（4.4.2.2 節）では、活性化前の値の二乗平均が層から層へと一定に保たれるように重みの分散を選ぶ。
    1. 訓練中、確率 p のドロップアウトは次の層の活性化前の値の二乗平均を何倍に変化させるか。本節のモデルにおける 2 つのドロップアウト層のそれぞれについて、その倍率を求めよ。
    2. $K-1$ 個の隠れ層を持ち、各隠れ層の後に確率 p のドロップアウトが続くネットワークにおいて、この倍率は K に伴ってどのように増大するか。テスト時にはドロップアウトなしでネットワークが動作することを踏まえると、初期化でこの倍率を補正すべきか。バッチ正規化層の直前にドロップアウトを置いたときに生じる訓練時とテスト時の不一致と関連づけて答えよ。
3. [code] **モンテカルロドロップアウト**。テスト時にドロップアウトをオフにする代わりに、訓練済みの `DropoutMLPScratch` でドロップアウトをオンのままにし、各検証画像に対して $T = 20$ 回の順伝播を実行してソフトマックス出力を平均せよ (Gal and Ghahramani 2016)。その正解率と較正（最大のソフトマックス確率の平均と正解率との一致度）を、ネットワーク全体による 1 回の順伝播の結果と比較せよ。T を 1 から 20 まで増やすにつれて、正解率はどのように変化するか。
4. [code] **ドロップアウトと重み減衰**。重み減衰とドロップアウトはいずれも、異なる仕組みによって本節のモデルを正則化する。各実行を、最良の検証正解率、およびマスクをオフにした状態で評価した訓練損失と検証損失のギャップによって測定せよ。
    1. 両方のドロップアウト確率を 0 に設定したうえで、2.7 節と同様に強度 $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}\}$ の重み減衰を加えよ。これらの実行結果を、本節のドロップアウトありおよびドロップアウトなしの実行結果と比較せよ。
    2. ドロップアウト確率を 0.2 と 0.5 に保ったまま、同じ強度の重み減衰を加えよ。2 つの正則化手法の効果は加法的に積み重なるか、収穫逓減を示すか、それとも互いに打ち消し合うか。

5. **テスト時のドロップアウト。**●dropout_1 を 0 に設定して第 2 隠れ層のみがマスクされるようにし、ξ がそのマスク全体を動くとする。$\mathbf{o}(\xi)$ と $\hat{\mathbf{y}}(\xi)$ を、マスク ξ（確率 $P(\xi)$ で生じる）のもとでのロジットおよびソフトマックス出力とする。ネットワーク全体のロジットが $\sum_\xi P(\xi)\,\mathbf{o}(\xi)$ に等しいことを示せ。さらに、そのソフトマックス出力が次の正規化された重み付き幾何平均に等しいことを示せ。

$$\tilde{y}_j = \frac{\prod_\xi \hat{y}_j(\xi)^{P(\xi)}}{\sum_k \prod_\xi \hat{y}_k(\xi)^{P(\xi)}}. \tag{4.20}$$

また、dropout_1 が非ゼロになると、いずれの等式も一般には成り立たない理由を説明せよ。

4.7 Kaggle での住宅価格予測

Fashion-MNIST のような整った画像ベンチマークは、$[0, 1]$ の均一なピクセル格子として提供され、欠損値も含まれない。実世界の表形式データ（tabular data）がこれほど整然としていることは稀である。本章の集大成となる本節では、Kaggle の House Prices: Advanced Regression Techniques コンペティションに取り組む。このデータセットは De Cock (2011) が米国アイオワ州エイムズの不動産売買記録（2006〜2010 年）から編纂したもので、ラベル付きの訓練用住宅 1,460 軒とラベルなしのテスト用住宅 1,459 軒からなる。各住宅はスケールが 5 桁にわたる異種混合の数値属性およびカテゴリ属性 79 個で記述されており、19 列に欠損値がある。

このエンドツーエンドの表形式パイプラインを一通り構築することで、第 1 章から第 4 章までで学んだツールが一つに結びつく。具体的には、データリークのない pandas による前処理、ターゲットの対数変換、共有フォールドを用いた K 分割交差検証（cross-validation）、線形回帰のベースライン、そして重み減衰（weight decay）で正則化した He 初期化 MLP である。

```
import pandas as pd
```

4.7.1 データセットと log-RMSE 指標

d2l.download を使って訓練用およびテスト用の CSV ファイルをダウンロードしてキャッシュし、pandas の DataFrame に読み込む（1.2 節）。

```
def read_house_prices(root='../data'):
    """Download and read the training and test tables of the competition."""
    train = pd.read_csv(d2l.download(
        d2l.DATA_URL + 'kaggle_house_pred_train.csv', root,
        sha1_hash='585e9cc93e70b39160e7921475f9bcd7d31219ce'))
    test = pd.read_csv(d2l.download(
        d2l.DATA_URL + 'kaggle_house_pred_test.csv', root,
        sha1_hash='fa19780a7b011d9b009e8bff8e99922a8ee2eb90'))
    return train, test
```

read_house_prices() を呼び出すと 2 つのテーブルが raw_train と raw_test に読み込まれ、その形状から訓練用住宅が 1,460 軒で 81 列（SalePrice を含む）、テスト用住宅が 1,459 軒で 80 列であることを確認できる。

```
raw_train, raw_test = read_house_prices()
print(raw_train.shape, raw_test.shape)
```

```
(1460, 81) (1459, 80)
```

raw_train の先頭と末尾の数列について最初の 4 行を確認すると、整数コード（Id、MSSubClass）、文字列のカテゴリ（MSZoning、SaleType、SaleCondition）、連続値の計測値（LotFrontage）、そしてドル建てのターゲットである SalePrice が混在している様子がわかる。

```
print(raw_train.iloc[:4, [0, 1, 2, 3, -3, -2, -1]])
```

```
   Id  MSSubClass MSZoning  LotFrontage SaleType SaleCondition  SalePrice
0   1          60       RL         65.0       WD        Normal     208500
1   2          20       RL         80.0       WD        Normal     181500
2   3          60       RL         68.0       WD        Normal     223500
3   4          70       RL         60.0       WD       Abnorml     140000
```

予測に用いる 79 個の属性（情報を持たない行インデックス Id を除いたもの）のうち、36 個は数値として、43 個は文字列として格納されており、欠損値は設備そのものが存在しないことを表している場合が多い（たとえばプールがない住宅での PoolQC = NaN など）。

住宅価格は \$34,900 から \$755,000 まで幅があるため、\$50,000 の誤差は手頃な住宅では致命的である一方、豪邸ではわずかな誤差にすぎない。そのため Kaggle では、価格の**対数**の二乗平均平方根誤差（root-mean-squared error、RMSE）を用いて、予測の**相対的な**精度によってスコアを算出する。

$$\text{log-RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log \hat{y}_i)^2}. \tag{4.21}$$

$\delta = |\log \hat{y} - \log y|$ が小さいとき、$\hat{y}/y = e^{\pm\delta} \approx 1 \pm \delta$ となるため、log-RMSE が 0.135 であることは典型的な相対誤差が約 13.5% であることに相当する。二乗誤差損失のもとで $\log y$ を予測するようにモデルを訓練すれば、コンペティションの評価指標の二乗を直接最適化できる。

4.7.2 データリークのない前処理

元の DataFrame を float32 の特徴量行列に変換するには、以下の preprocess(train, *others) に実装されている 4 つの慎重なステップが必要である。

1. **数値のカテゴリコードを文字列に変換する：**MSSubClass は数値（$20, 60, 90, \ldots$）を使って住居タイプ（1 階建て、2 階建て、二世帯住宅）を表している。これを str に変換することで、モデルがコード 90 をコード 20 の定数倍として扱うのを防ぐ。
2. **数値特徴量を訓練セットの統計量のみで標準化する：**平均 μ と標準偏差 s を train のみから計算し、訓練テーブルと検証・テストテーブルの両方に $(x - \mu)/s$ を適用することで、SGD と重み減衰のためにすべての数値特徴量を単位スケールに揃えつつ、検証行やテスト行からの**データリーク**を防ぐ。
3. **欠損している数値を 0（訓練平均）で補完する：**標準化後の NaN 要素を 0 で埋めることで、欠損している数値の計測値を訓練セットの平均で置き換える（演習問題 1）。
4. **カテゴリ特徴量をワンホットエンコーディングする（pd.get_dummies(..., dummy_na=True)）：**各カテゴリ列を二値インジケータ（NaN に対する明示的なインジケータを

含む）に展開し、`reindex(..., fill_value=0)` によって検証・テスト列を訓練列の語彙と厳密に一致させる。

```
def preprocess(train, *others):
    """Fit the feature preprocessing on train; apply it to all tables."""
    frames = [df.drop(columns=['Id', 'SalePrice'], errors='ignore')
                .astype({'MSSubClass': str})  # A code, not a quantity
              for df in (train, *others)]
    numeric = frames[0].select_dtypes(include='number').columns
    mean, std = frames[0][numeric].mean(), frames[0][numeric].std()
    for df in frames:
        df[numeric] = ((df[numeric] - mean) / std).fillna(0)
    frames = [pd.get_dummies(df, dummy_na=True) for df in frames]
    return [df.reindex(columns=frames[0].columns, fill_value=0)
              .to_numpy(dtype=np.float32) for df in frames]
```

コンペティションのテーブル全体に対して `preprocess(raw_train, raw_test)` を実行すると、元の 79 個の属性が展開され、列の揃った `float32` の計画行列（design matrix）である `X_train` と `X_test`（各住宅につき 345 個の特徴量）が得られる。

```
X_train, X_test = preprocess(raw_train, raw_test)
X_train.shape, X_test.shape
```

```
((1460, 345), (1459, 345))
```

最後に、`KaggleHouse(d2l.DataModule)` は `preprocess` と対数ターゲットの中心化を組み合わせ、訓練分割における対数価格の平均 `self.offset`（≈ 12.02）を $\log(\text{SalePrice})$ から差し引くことで、ネットワークがステップ 1 から適切なベースラインで出発できるようにする（演習問題 2）。

```
class KaggleHouse(d2l.DataModule):
    """Features and centered log prices, fitted on the training rows."""
    def __init__(self, raw_train, raw_val, batch_size=64):
        super().__init__()
        self.save_hyperparameters(ignore=['raw_train', 'raw_val'])
        self.X_train, self.X_val = preprocess(raw_train, raw_val)
        self.num_inputs = self.X_train.shape[1]
        log_price = lambda df: np.log(
            df['SalePrice'].to_numpy(np.float32)).reshape(-1, 1)
        self.offset = log_price(raw_train).mean()  # Mean log price
        self.y_train = log_price(raw_train) - self.offset
        self.y_val = (log_price(raw_val) - self.offset
                      if 'SalePrice' in raw_val else None)

    def get_dataloader(self, train):
        X, y = ((self.X_train, self.y_train) if train
                else (self.X_val, self.y_val))
        if y is None:  # No prices to validate against
            return None
        return self.get_tensorloader((d2l.tensor(X), d2l.tensor(y)), train)
```

4.7.3 K 分割交差検証によるモデル選択

4.7.3.1 共有 5 分割と線形ベースライン

ラベル付き住宅が 1,460 軒しかないため、単一の 20% 検証分割（292 軒）では、フォールド間の標準誤差が log-RMSE で 0.01 の数倍に達する。そこでヘルパー関数 `k_fold_data(raw, k)` を用いて `raw` を K 個のランダムなフォールドに分割し、検証フォールドが訓練特徴量の統計量に決してリークしないよう、ラウンドごとに個別の `KaggleHouse` データモジュールを構築する。

```
def k_fold_data(raw, k):
    """Split the table raw into k random folds; one data module per round."""
    folds = np.array_split(np.random.default_rng().permutation(len(raw)), k)
    return [KaggleHouse(raw.drop(index=raw.index[f]), raw.iloc[f])
            for f in folds]
```

事前に構築した `folds` のリストとモデル生成関数 `make_model` を受け取ると、`k_fold` は各フォールドで新しいモデルを `max_epochs` エポック訓練する。さらに、1 エポックごとのばらつきを抑えるために最後の 5 エポック分の二乗損失を平均してから平方根をとり、全フォールドにわたる訓練および検証 log-RMSE の平均を表示する。

```
def k_fold(folds, make_model, max_epochs=100):
    """Train a fresh model in each round; return the RMSEs of every fold."""
    rmse = []
    for fold in folds:
        trainer = d2l.Trainer(max_epochs=max_epochs)
        trainer.board.display = False
        trainer.fit(make_model(fold.num_inputs), fold)
        rmse.append([np.mean(trainer.history[f'{split}_loss'][-5:]) ** 0.5
                     for split in ('train', 'val')])  # Last five epochs
    train, val = np.array(rmse).T
    print(f'training RMSE {train.mean():.3f}, validation RMSE '
          f'{val.mean():.3f} (folds {val.min():.3f} to {val.max():.3f})')
    return train, val
```

ここで、共有の 5 分割 `folds` を 1 回だけ生成しておく。これにより、以降のすべての候補モデルがまったく同じ 5 組の訓練・検証分割で評価される。

```
folds = k_fold_data(raw_train, 5)
```

まず、`make_linear` を定義して `d2l.LinearRegression` のベースライン（100 エポック、バッチサイズ 64、学習率 $\eta = 0.02$）を用意しよう。

```
make_linear = lambda num_inputs: d2l.LinearRegression(num_inputs, lr=0.02)
```

`k_fold(folds, make_linear)` を実行すると、正則化なしの線形ベースラインが 5 つのフォールドすべてで訓練され、訓練および検証 log-RMSE のベクトルが `linear` に記録される。

```
linear = k_fold(folds, make_linear)
```

```
training RMSE 0.113, validation RMSE 0.154 (folds 0.126 to 0.224)
```

線形ベースラインは、5 分割の検証 log-RMSE で ~ 0.15（典型的な価格誤差で $\sim 15\%$）を達成し、対する訓練 log-RMSE は ~ 0.11 である。

4.7.3.2 重み減衰を伴う MLP

次に、`HouseMLP` を構築しよう。これは 64 個の ReLU ユニットを持つ隠れ層 1 層の MLP（パラメータ数は $\sim 22{,}000$ 個、すなわち 1 フォールド内の訓練用住宅 1 軒あたり ~ 19 個のパラメータ）であり、He 初期化された重みと、重み行列のみ（バイアスを除く）に適用される ℓ_2 重み減衰 `wd` を備えている。

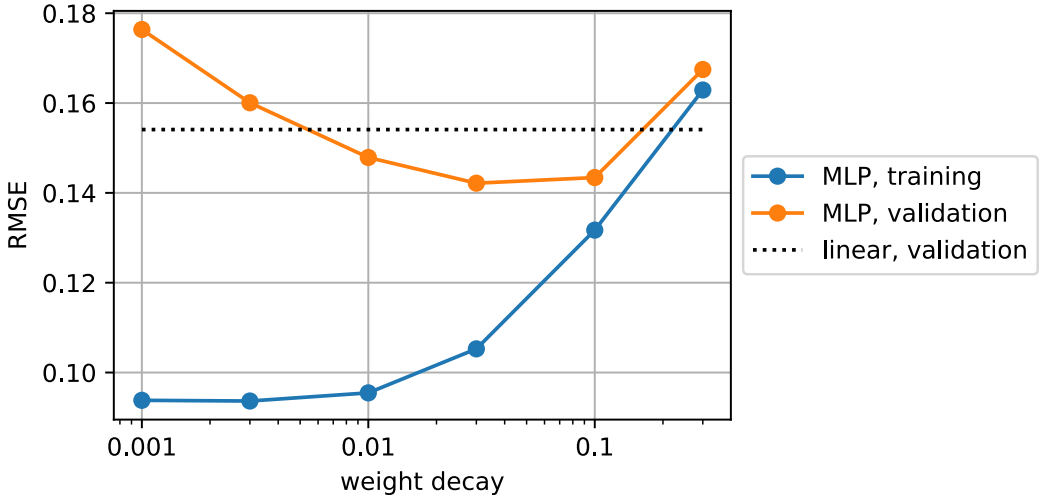
```
def is_matrix(params):
    """Mask that selects the parameters with more than one dimension."""
    return jax.tree.map(lambda p: p.ndim > 1, params)

class HouseMLP(d2l.Module):
    """An MLP with one hidden layer, squared loss, and weight decay."""
    loss = d2l.LinearRegression.loss

    def __init__(self, num_inputs, num_hiddens, wd, lr, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        self.net = nnx.Sequential(
            nnx.Linear(num_inputs, num_hiddens, kernel_init=he, rngs=rngs),
            nnx.relu,
            nnx.Linear(num_hiddens, 1, kernel_init=he, rngs=rngs))

    def configure_optimizers(self):
        return optax.chain(
            optax.masked(optax.add_decayed_weights(self.wd), is_matrix),
            optax.sgd(self.lr))

make_mlp = lambda num_inputs, wd: HouseMLP(num_inputs, num_hiddens=64,
                                           wd=wd, lr=0.02)
```

ここで、`wd` を `wds = [0.001, 0.003, 0.01, 0.03, 0.1, 0.3]` にわたって共有の 5 分割 `folds` 上で変化させ、訓練 RMSE と検証 RMSE の平均を線形ベースラインと比較してプロットする。

```
wds = [0.001, 0.003, 0.01, 0.03, 0.1, 0.3]
mlp = {}
for wd in wds:
    print(f'weight decay {wd}: ', end='')
    mlp[wd] = k_fold(folds, lambda num_inputs: make_mlp(num_inputs, wd))
d2l.plot(wds, [[mlp[wd][0].mean() for wd in wds],
               [mlp[wd][1].mean() for wd in wds], [linear[1].mean()] * 6],
         'weight decay', 'RMSE', xscale='log', fmts=('C0-o', 'C1-o', 'k:'),
         legend=['MLP, training', 'MLP, validation', 'linear, validation'],
         legend_loc='outside', figsize=(4.5, 3))
```

```
weight decay 0.001: training RMSE 0.094, validation RMSE 0.176 (folds 0.141 to 0.296)
weight decay 0.003: training RMSE 0.094, validation RMSE 0.160 (folds 0.134 to 0.244)
```

```
weight decay 0.01: training RMSE 0.095, validation RMSE 0.148 (folds 0.118 to 0.212)
weight decay 0.03: training RMSE 0.105, validation RMSE 0.142 (folds 0.117 to 0.214)
weight decay 0.1: training RMSE 0.132, validation RMSE 0.143 (folds 0.128 to 0.188)
weight decay 0.3: training RMSE 0.163, validation RMSE 0.167 (folds 0.153 to 0.201)
```



5 分割の検証 RMSE の平均が最小となる重み減衰 `best` を選択し、フォールドのサンプリングノイズが相殺されるフォールドごとの対応のある差 `mlp[best][1] - linear[1]` も確認できる。

```
best = min(wds, key=lambda wd: mlp[wd][1].mean())
print(f'weight decay {best}: MLP minus linear, per fold',
      np.round(mlp[best][1] - linear[1], 3))
```

```
weight decay 0.03: MLP minus linear, per fold [-0.011 -0.004 -0.011 -0.028 -0.006]
```

交差検証の曲線は教科書通りの U 字型を描く。弱い重み減衰（`wd = 0.001`）では訓練フォールドに過学習し（訓練 RMSE が ~ 0.09、検証 RMSE が ~ 0.15〜0.18 で線形回帰より悪い）、中程度の重み減衰（`wd = 0.03` または `0.1`）では検証 RMSE が ~ 0.13〜0.14 というスイートスポットに入り（大半のフォールドで線形ベースラインを上回る）、強すぎる減衰（`wd = 0.3`）では学習不足に陥る。

4.7.4 予測の提出

5 分割交差検証で選ばれた最良の重み減衰 `best` を用いて、ラベル付き住宅全 $1{,}460$ 軒で `HouseMLP` を再訓練し（あるいは 5 つのフォールドモデルをアンサンブルしてもよい。演習問題 5 を参照）、訓練セットの対数価格の平均 `data.offset` を足し戻し、指数変換でドル単位の予測値に戻して `submission.csv` に保存する。

```
data = KaggleHouse(raw_train, raw_test)
model = make_mlp(data.num_inputs, wd=best)
trainer = d2l.Trainer(max_epochs=100)
trainer.board.display = False
trainer.fit(model, data)
log_price = d2l.numpy(model(d2l.tensor(data.X_val))).reshape(-1) + data.offset
```

```
submission = pd.DataFrame({'Id': raw_test['Id'],
                           'SalePrice': np.exp(log_price)})
submission.to_csv('submission.csv', index=False)
submission.head()
```

```
     Id      SalePrice
0  1461  118287.554688
1  1462  163739.953125
2  1463  180315.218750
3  1464  198616.843750
4  1465  189772.375000
```

4.7.5 まとめ

実世界の表形式回帰では、データリークのない規律ある前処理が鍵を握る。具体的には、任意の行識別子の除去、数値カテゴリコードの文字列への変換、訓練分割の平均と標準偏差による数値列の標準化とそれに続く欠損値のゼロ補完、明示的な欠損性インジケータを伴うカテゴリ変数のワンホットエンコーディング、そして対数変換したターゲットの訓練平均による中心化である。すべての候補モデルを同一の $K = 5$ 交差検証フォールド（各ラウンドで 4 つの訓練フォールドのみを用いて前処理器を再適合させる）で評価することで、モデル同士を直接比較する際にフォールドのサンプリングノイズが相殺される。Ames Housing データセットでは、ℓ_2 重み減衰を調整することで、22,000 パラメータの ReLU MLP は過学習の状態（log-RMSE で ~ 0.15〜0.18）から線形ベースライン（~ 0.15）を超え、log-RMSE で ~ 0.13〜0.14 に到達する。もっとも、異種混合の表形式特徴量においては、勾配ブースティング決定木が依然として強力な競合手法である（演習問題 3）。

4.7.6 演習問題

1. [code] **平均値補完**。 `preprocess` は欠損しているすべての数値を、その列の訓練セットの平均で置き換える。また、`pd.get_dummies` を `dummy_na=True` で呼び出すことで、すべてのカテゴリ特徴量に対して欠損値インジケータ列も追加される。

    1. このデータセットの中から、販売価格に対して欠損性がランダムではない数値特徴量を 1 つ特定せよ。
    2. その特徴量に対して平均値補完のみを適用するとどのような統計的バイアスが生じるか、また該当する住宅の予測がどの方向にずれるかを説明せよ。続いて、`preprocess` がすでに作成しているインジケータ列によって、線形モデルがこのバイアスを除去できるかどうかを判定せよ。
    3. 平均値補完を中央値補完に置き換え、それとは別に補完した各特徴量に対して二値の欠損性インジケータ列を追加せよ。これら 3 つの方式の交差検証誤差を比較せよ。

    Kaggle Learn の Intermediate Machine Learning の演習問題 2 を改変。

2. [code] **特徴量とターゲットのスケーリング**。 `preprocess` から標準化を取り除け。`fillna(0)` が列の平均を補完するのは標準化された列に限られるため、代わりに欠損している数値を訓練セットの列平均で明示的に埋めよ。

    1. 本節と同じ学習率 0.02 で線形モデルの交差検証を再実行し、訓練損失の挙動を述べよ。
    2. 標準化していない線形モデルが発散せずに訓練できる最大の学習率を 1 桁（10 倍）の精度で求めよ。100 エポック後の交差検証誤差を標準化ありのパイプラインと比較し、両方の観測結果を説明せよ。

3. 標準化を元に戻したうえで、対数価格から平均を引かずに訓練せよ。100 エポック後の交差検証誤差を中心化したターゲットの場合と比較し、その違いを説明せよ。
4. 本節のパイプラインを用い、毎エポック後の検証フォールドにおける線形モデルの検証 RMSE と残差の平均を記録せよ。RMSE が跳ね上がるエポックでは何が動いているか。どの住宅もちょうど 44 個のワンホット特徴量が 1 に等しい（バイアスを含めれば 45 個の入力が 1 に等しい）。SGD の 1 ステップがいずれか単一の重みの変化量よりも平均予測を大きく変化させる理由を説明し、学習率を半分にするとこの跳ね上がりが消えるかどうかを確認せよ。

3. [code] **勾配ブースティング木**。このような表形式データでは、決定木のアンサンブルがニューラルネットワークよりも高い精度で予測できることが多い (Grinsztajn et al. 2022; Shwartz-Ziv and Armon 2022)。勾配ブースティング木モデル（たとえば scikit-learn の `GradientBoostingRegressor`、あるいはインストールされていれば XGBoost や LightGBM）を、`folds` の各フォールドの訓練行における中心化対数価格に適合させ、各フォールドでの検証 RMSE を計算せよ。その平均を本節のモデルと比較せよ。データのどのような性質が決定木に有利に働くか。

4. [code] **ターゲットリーク**。予測対象の事象が起きた後、あるいはその事象が起きた結果として初めて記録される特徴量は、実運用モデルには役立たないまま検証スコアだけを不当に押し上げる。

   1. このデータセットの元の特徴量の中から、売却後に初めて記入されうると考えられるものを少なくとも 1 つ特定し、交差検証ではなぜこの問題を検出できないのかを説明せよ。
   2. この仕組みを実験的に確認せよ。対数販売価格に小さなガウスノイズを加えた合成特徴量を追加して交差検証を再実行し、検証 RMSE がどう変化するかを報告せよ。

   Kaggle Learn の Intermediate Machine Learning の演習問題 7 を改変。

5. [code] **フォールドアンサンブルと再訓練の比較**。4.7.4 節では、選択した MLP を全住宅で再訓練している。別の方法として、交差検証中に訓練した 5 つのモデルにそれぞれのラウンドの前処理を用いてテスト住宅を予測させ、その対数価格の予測を平均することもできる。

   1. このフォールドアンサンブルを実装し、その予測と再訓練モデルの予測の両方を提出せよ。2 つのリーダーボードのスコアを互いに比較するとともに、交差検証 RMSE とも比較せよ。
   2. アンサンブルの各モデルは全住宅の 5 分の 4 でしか訓練されていない。それにもかかわらず、全住宅で再訓練したモデルよりアンサンブルのほうが精度よく予測しうるのはなぜか。
   3. 再訓練モデルの訓練誤差が交差検証による推定値の代わりにならない理由を説明せよ。

6. [code] **ターゲットエンコーディング**。●**ターゲットエンコーディング**は、カテゴリ列の各カテゴリをそのカテゴリに属する訓練行の対数販売価格の平均で置き換える手法であり、`preprocess` が作成するカテゴリごとの列ではなく単一の列で特徴量を表現できる。

   1. 10 個を超えるカテゴリを持つすべてのカテゴリ列に対して、ワンホットエンコーディングの代わりに各フォールドの訓練行からカテゴリ平均を計算してターゲットエンコーディングを適用せよ。訓練行に出現しないカテゴリには全体の平均を与え、エンコード後の列を数値特徴量と同様に標準化せよ。その交差検証誤差をワンホットエンコーディングの場合と比較せよ。
   2. 上記のエンコーディングでは、各訓練行の値を計算する平均にその行自身のラベルが含まれている。代わりに訓練行をフォールド外でエンコードせよ。すなわち、各フォールドの訓練行をさらに 5 分割し、残りの 4 分割から計算したカテゴリ平均で各分割

をエンコードせよ。交差検証誤差をサンプル内エンコーディングの場合と比較し、カテゴリのサイズ（行数）が 2 つのエンコーディングの差にどう影響するかを説明せよ。

7. **幾何平均。**●K 個のモデルからなるアンサンブルが住宅の対数価格 $z_1, \dots, z_K$ を予測するとき、指数変換の前にそれらを平均でき、これは価格空間での幾何平均に相当する。

   1. $\exp\big(\frac{1}{K}\sum_{k=1}^{K} z_k\big)$ と価格空間での平均 $\frac{1}{K}\sum_{k=1}^{K} e^{z_k}$ を比較せよ。どちらがもう一方を決して上回らないかを示し、両者が一致する条件を述べよ。

   2. 住宅の真の対数価格を $\log y$ とする。対数空間での平均が次式を満たすことを示せ。

$$\Big(\log y - \frac{1}{K}\sum_{k=1}^{K} z_k\Big)^2 \le \frac{1}{K}\sum_{k=1}^{K} \big(\log y - z_k\big)^2 . \tag{4.22}$$

   また、価格空間での平均の対数も、すべての予測 $z_1, \dots, z_K$ に対して同じ上界を満たすかどうかを判定せよ。

Chapter 5

計算

ここまでは、モデルの構築、パラメータの管理、初期化、デバイス上での実行をフレームワークの裏側の処理に任せてきた。しかし、教科書レベルのプロトタイプから本番規模のシステムへと進むには、そのブラックボックスを開ける必要がある。80 億パラメータの言語モデルを 32 ビット浮動小数点の Adam で訓練する場合、活性化値を 1 つ保持する前の段階で、パラメータ、勾配、オプティマイザ状態（optimizer state）だけで 128 GB のメモリを要するうえ、数時間おきにハードウェア障害で訓練が中断される数千基の GPU 上で実行することになる。この規模では、フレームワークがモデルをどのように表現し、初期化し、計算に用い、永続化するかを理解することが不可欠である。

本章のあらゆる内容は、1 つの中心的なデータ構造の上に築かれている。すなわち、ニューラルネットワークとは**モジュールの木**（tree of modules）であり、その葉がモデルの**状態**（state、訓練可能なパラメータと固定バッファの両方）を保持し、それぞれが根からのパスによって指定されるという構造である。本章の前半では、この構造的な抽象化を展開し、フレームワークの各操作がそれをどのように走査するかを示す。まず、モジュールの階層構造、コンテナへの登録、残差スキップ接続、形状の遅延決定（lazy shape inference）、設定駆動のビルダから始める（5.1 節）。続いて状態木を開き、訓練可能なパラメータと訓練対象外のバッファを切り分け、fp32 の Adam 訓練におけるパラメータあたり 16 バイトというメモリ下限を導出し、ストレージを確保せずにパラメータ数を数え上げ、重みの共有と凍結を扱う（5.2 節）。次に、層ごとの分散規則から木全体および残差構造の初期化へと進み、スケーリングしない残差分岐が活性化値の分散を $(1+g^2)^N$ 倍に増幅させる理由と、$1/\sqrt{N}$ またはゼロによる出力射影（output projection）のスケーリングが深い積層モデルを安定させる仕組みを示す（5.3 節）。そのうえで、状態をもたない層、パラメータをもつ層（`RMSNorm`）、固定バッファをもつ層（`RandomFeatures`）、およびストレートスルー推定器（straight-through estimator）によるカスタム逆伝播規則を用いてフレームワークを拡張する（5.4 節）。

後半では、モデルの構造から数値表現、永続化、ハードウェア実行、および診断へと目を向ける。まず fp32、fp16、bf16 を比較し、16 ビットの重みでは微小な更新が丸めによって消えてしまう理由を説明したうえで、混合精度訓練（mixed-precision training）がいかに fp32 のマスター重みと 16 ビット演算、および fp16 の損失スケーリング（loss scaling）を組み合わせるかを示す（5.5 節）。次に、安全でない `pickle` アーカイブとコードを含まない `safetensors` ファイルを対比し、ビット単位で完全に再開できるアトミックな 5 要素の訓練チェックポイントを構築するとともに、事前学習済みバックボーンを読み込む際の欠落キーや予期しないキーを検査する（5.6 節）。最後に、テンソルとモデルを GPU 上に配置し、非同期コマンドキューの実行時間を正確に計測し、$O(\sqrt{K})$ の活性化チェックポイント（activation checkpointing）とあわせて GPU メモリの内訳を詳しく調べ（5.7 節）、擬似乱数シードの制御および GPU の決定論的実行を非侵襲的な順

伝播・逆伝播フックと組み合わせる（5.8 節）。

参考資料と関連文献

以下の資料は本章の各節に対応しており、それぞれ関連する節を併記している。特記のない限り、すべてオンラインで無料公開されている。

書籍

- Deep Learning with PyTorch (Stevens, Antiga & Viehmann)：PyTorch チームによる無料の PDF 版（アーカイブ版）。第 1 部でテンソル、ストレージ、自動微分、`nn.Module` を 5.1 節および 5.2 節と同じ水準で解説している。
- Machine Learning Systems (Vijay Janapa Reddi)：オンラインで無料公開。フレームワーク、データパイプライン、訓練インフラストラクチャ、効率的なデプロイメントなど、本章を取り巻くシステム面の視点を扱っている（5.7 節、5.8 節）。

講義コースと講義動画

- CMU 10-414/714: Deep Learning Systems (Chen & Kolter)：無料の講義と課題。自動微分、モジュール、初期化、GPU サポートを備えた小型フレームワーク「needle」を受講生自ら構築する（5.1 節、5.3 節、5.7 節）。
- Neural Networks: Zero to Hero (Andrej Karpathy)：無料の動画シリーズ。標準の Python のみを用いて、自動微分、モジュール、訓練ループをゼロから構築する（5.1 節、5.4 節）。
- fast.ai Part 2: Deep Learning Foundations (Howard et al.)：無料の講義コース。モジュール、初期化、混合精度を含め、テンソル演算から積み上げて訓練フレームワークを再構築する（5.1 節、5.3 節、5.5 節）。

チュートリアル・解説記事・インタラクティブ教材

- PyTorch internals (Edward Yang)：PyTorch 内部のテンソル、ストライド、ディスパッチ、自動微分を順を追って解説する記事であり、5.1 節より 1 段低い抽象度を扱っている。
- JAX: The Sharp Bits：純粋関数、明示的な PRNG キー、JIT の制約など、JAX の背後にある関数型設計を実践上の注意点としてまとめた解説である（5.2 節、5.8 節）。
- Train With Mixed Precision (NVIDIA)：fp16/bf16 演算と損失スケーリングに関するベンダー公式ガイドであり、5.5 節はその要点を凝縮したものである。
- What Every Computer Scientist Should Know About Floating-Point Arithmetic (David Goldberg)：あらゆる丸めやオーバーフローの問題（5.5 節）の根底にある標準的な参考文献である。
- Making Deep Learning Go Brrrr From First Principles (Horace He)：演算能力、メモリ帯域幅、フレームワークのオーバーヘッドを第一原理から解き明かす解説記事である（5.7 節）。
- safetensors (Hugging Face)：読み込み時にコードを実行せずに重みを共有するためのファイル形式と、それが pickle ベースのチェックポイントを置き換えた理由を解説するドキュメントである（5.6 節）。
- Reproducibility (PyTorch notes)：決定論的実行のためのフラグ、その計算性能上のコスト、および限界についての解説であり、5.8 節を補完する。

基礎となった論文

- Automatic Differentiation in Machine Learning: a Survey (Baydin, Pearlmutter, Radul & Siskind, 2018, **JMLR**)：フォワードモードおよびリバースモードの自動微分に関する決定版のサーベイ論文であり、5.4 節のあらゆるカスタム勾配の背後にあるアルゴリズムを解説している。

- Mixed Precision Training (Micikevicius et al., 2018)：fp16・マスター重み・損失スケーリングを組み合わせたレシピ（5.5 節）の原典となった論文である。
- PyTorch: An Imperative Style, High-Performance Deep Learning Library (Paszke et al., 2019)：即時実行、自動微分、キャッシングアロケータなど、5.1 節および 5.7 節の命令型スタイルを支える設計思想を述べた論文である。

5.1 モジュールとモデル構築

4.2 節の多層パーセプトロン（multilayer perceptron、MLP）が持つパラメータ配列はわずか 4 個であり、手作業でも管理できるほど小さかった。しかし、実世界のアーキテクチャははるかに複雑である。コンピュータビジョン向けの現代的な残差ネットワーク（residual network）は何十層、何百層もの層を積み重ねており、Llama 3 8B のような言語モデル（language model）では (Grattafiori et al. 2024)、32 個の同一の Transformer ブロックにわたる 80 億個のパラメータの重みが 291 個もの個別の配列に分散している。これらの配列は 1 つ残らずオプティマイザに渡り、チェックポイントに書き出され、GPU 上へ転送されなければならない。さらに厄介なことに、配列が 1 つ漏れても構文エラーにはならず、訓練も保存もされないまま静かに放置されてしまう。

深層学習フレームワークは、この煩雑な管理の問題を**モジュール**という単一の再帰的な抽象化によって解決する。モジュールは、個々の層、層のグループ、あるいはモデル全体を表すことができる。モジュールはパラメータと子モジュールを所有し、入力を出力へ写像する順伝播の計算を定義する。モジュールは他のモジュールの内部に入れ子にできるため、あらゆるアーキテクチャはモジュールの木構造をなす（Figure 5.1）。各子モジュールが親モジュールに正しく**登録**（register）されてさえいれば、どのフレームワークもこの木を走査することで全パラメータを自動的に見つけ出す。

本節は実践的な内容であり、すでにカスタムモジュールを実装した経験がある読者は基礎的な部分を流し読みしてもよい。ただし、登録の仕組みとそれが静かに失敗する挙動だけは読み飛ばさないようにしよう。これは非常に踏みやすいバグとして知られているからである。また本節では、残差接続（residual connection）などのカスタム順伝播計算の書き方、遅延初期化（lazy initialization）によって実行時に形状を自動的に決める仕組み、そしてアーキテクチャ上のハイパーパラメータをコンパクトな設定（configuration）オブジェクトへ分離する方法も示す。後続の節では、同じ木の走査を再利用して、パラメータ数の集計（5.2 節）、重みの初期化（5.3 節）、チェックポイントの保存（5.6 節）、および GPU 上へのモデル配置（5.7 節）を扱う。

```
from dataclasses import dataclass
```

5.1.1 モジュールの抽象化

5.1.1.1 逐次コンテナとレジストリ

2.4 節以降に記述してきたモデルはすべてモジュールであった。すなわち、`d2l.Module` は各フレームワークにおける基底モジュールクラスのサブクラスである。層のパイプラインを最も手早く組み立てる方法は、子モジュールを順番につなぎ合わせる**逐次コンテナ**（sequential container）を使うことである。以下では、4.2 節と同様に 256 個の ReLU ユニットを持つ 1 つの隠れ層と 10 個の出力を備えた MLP を構築し、それぞれ 20 個の特徴量を持つ 2 個の例からなるミニバッチを入力する。

Flax NNX では基底クラスが `nnx.Module` であり、`nnx.Sequential` は引数を順番につなぎ、各層は構築時に `rngs` を用いてパラメータを確保する。

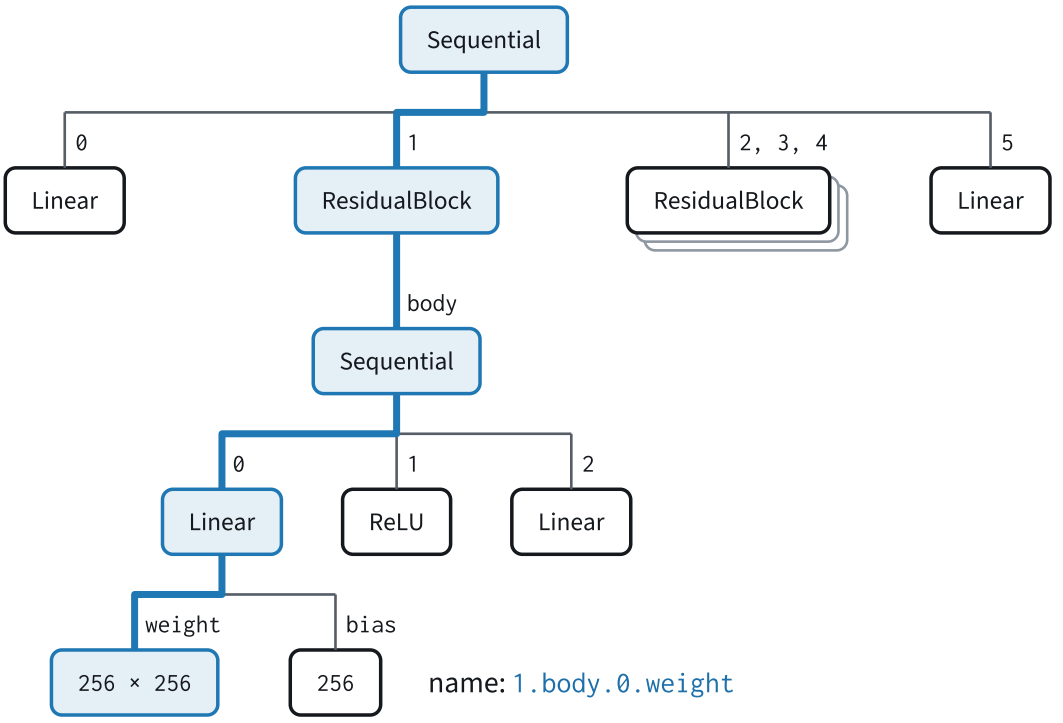


Figure 5.1: 本節の最後で構築する残差 MLP のモジュール木。各辺には親が子を登録した際の名前が付いており、根からパラメータに至る名前のパスがそのパラメータの名前（ここでは `1.body.0.weight`）になる。ブロック 2〜4 はブロック 1 と同じ構造を繰り返す。

```
rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(nnx.Linear(20, 256, rngs=rngs), nnx.relu,
                     nnx.Linear(256, 10, rngs=rngs))

X = jax.random.uniform(d2l.get_key(), (2, 20))
net(X).shape
```

```
(2, 10)
```

内部では、コンテナは名前を子モジュールに対応付ける順序付きの**レジストリ**に子モジュールを格納しており、順伝播ではそのテーブルを挿入順に反復処理する。

`nnx.state(net, nnx.Param)` は、`layers` を経由するパスごとにすべてのパラメータを列挙する。

```
[(path, value.shape)
 for path, value in nnx.state(net, nnx.Param).flat_state()]
```

```
[(('layers', 0, 'bias'), (256,)),
 (('layers', 0, 'kernel'), (20, 256)),
 (('layers', 2, 'bias'), (10,)),
 (('layers', 2, 'kernel'), (256, 10))]
```

モデルに対するあらゆる木の走査は、このレジストリを起点とする。オプティマイザ用のパラメータ収集、ディスクへの重みのシリアライズ、GPU へのテンソル転送はいずれも、登録された子モジュールを再帰的にたどってすべての葉パラメータに到達する。これらの操作にとって、未登録の層は存在しないも同然である。

5.1.1.2 モジュールクラスのサブクラス

逐次コンテナは一直線の連鎖を簡潔に扱えるが、多くのアーキテクチャは分岐や合流を含んだり、構成要素を再利用したりする。単純な直列構造を超えるモデルでは、フレームワークの基底モジュールを継承してサブクラスを作り、子モジュールを生成するコンストラクタと、それらの間をデータがどう流れるかを規定する順伝播メソッドの 2 つを実装する。

Flax NNX では、順伝播メソッドは `__call__` である。

先ほどと同じ MLP を改めて以下に示す。

```
class MLP(nnx.Module):
    def __init__(self, rngs):
        self.hidden = nnx.Linear(20, 256, rngs=rngs)
        self.out = nnx.Linear(256, 10, rngs=rngs)

    def __call__(self, X):
        return self.out(nnx.relu(self.hidden(X)))
```

`MLP` をインスタンス化して `net(X)` を呼び出すと、定義したカスタム順伝播メソッドへと処理が渡り、2×20 の入力バッチ `X` が形状 `(2, 10)` の出力テンソルへと写像される。

JAX では、コンストラクタが明示的な `nnx.Rngs` を受け取る。

```
net = MLP(nnx.Rngs(d2l.get_key()))
net(X).shape
```

```
(2, 10)
```

裏側では 2 つの仕組みが重要な役割を果たしている。第 1 に、`self.hidden = ...` という代入は単なる Python の属性保存ではない。基底モジュールが `__setattr__` を介して代入を捕捉し、その値がモジュールであることを検知して、属性名（`hidden` と `out`）のもとで登録する。この自動登録だけで、両方の層のパラメータがオプティマイザへ渡るようになる。第 2 に、逆伝播メソッドを自分で書く必要は一切ない。1.5 節で扱った自動微分（automatic differentiation）が、順伝播で実行された演算から直接勾配を導出するためである。

モジュールの呼び出しは常に `net(X)` とする。

5.1.2 モジュールコンテナ

5.1.2.1 コンテナの実装

逐次コンテナの役割は、追加される各子モジュールを登録することと、それらに順番に入力を通すことの 2 つだけである。最小限のバージョンを自作してみることで、登録が正確にどのように機能し、どこで問題が生じうるのかが明確になる。

```
class MySequential(nnx.Module):
    def __init__(self, *modules):
        self.modules = nnx.List(modules)

    def __call__(self, X):
        for module in self.modules:
            X = module(X)
        return X
```

`nnx.List` はモジュールの並びをグラフ追跡対象のコンテナとしてラップし、`modules` の下に保持する。

自作した `MySequential` は、組み込みのコンテナとまったく同じように動作する。

```
rngs = nnx.Rngs(d2l.get_key())
net = MySequential(nnx.Linear(20, 256, rngs=rngs), nnx.relu,
                   nnx.Linear(256, 10, rngs=rngs))
net(X).shape
```

```
(2, 10)
```

5.1.2.2 通常のリストに入れた層

モジュール登録に不慣れな人がつい書いてしまうように、通常の Python の `list` に層を格納するとどうなるだろうか。順伝播ではそのリストをループ処理できるものの、通常の `list` はモジュールではない。それでもフレームワークはリスト内部の層を見つけ出せるのだろうか。

```
class PlainListMLP(nnx.Module):
    def __init__(self, rngs):
        self.layers = [nnx.Linear(20, 256, rngs=rngs),
                       nnx.Linear(256, 10, rngs=rngs)]  # Plain list

    def __call__(self, X):
        return self.layers[1](nnx.relu(self.layers[0](X)))

try:
    net = PlainListMLP(nnx.Rngs(d2l.get_key()))
except ValueError as e:
    print(str(e).split(' of Pytree')[0])  # The rest lists the repairs
```

```
Found data on value of type '<class 'list'>' assigned to static attribute 'layers'
```

NNX では、通常の Python リストがモジュールを保持していると構築時に即座に `ValueError` が送出され、修正方法として `nnx.List` が示されるため、一部だけ登録されたモデルが気づかないうちに構築されることはない。

```
class ListMLP(PlainListMLP):
    def __init__(self, rngs):
        self.layers = nnx.List([nnx.Linear(20, 256, rngs=rngs),
                                nnx.Linear(256, 10, rngs=rngs)])

net = ListMLP(nnx.Rngs(d2l.get_key()))
state = nnx.state(net, nnx.Param)
net(X).shape, sum(x.size for _, x in state.flat_state())
```

```
((2, 10), 7946)
```

`nnx.List`（マッピングの場合は `nnx.Dict`）を使えば、7,946 個すべてのパラメータが問題なく登録される。

5.1.3 順伝播の計算

モジュールの順伝播メソッドは通常の Python コードであるため、任意のテンソル演算、ループ、条件分岐、スキップ経路を呼び出すことができる。この柔軟性を活かして残差接続を実装し、続いてフレームワークが訓練ステップをコンパイルする際にどのような制約が生じるかを見ていこう。

5.1.3.1 残差接続

単純な直列構造ではないアーキテクチャのうち、広く使われている最も基本的なものが、ブロックの入力をその出力へ直接加算する**残差接続**である (He et al. 2016b)。ここでは間に ReLU を挟んだ 2 つの全結合層（fully connected layer）からなるブロックの**本体**を f と書くと、このブロックは次の式を計算する。

$$\mathbf{y} = \mathbf{x} + f(\mathbf{x}). \tag{5.1}$$

$f(\mathbf{x}) = \mathbf{0}$ であれば常にこのブロックは恒等写像になるため、深い積層を通して信号がそのまま伝わり、各ブロックは表現をゼロから作り直すのではなく補正分 $f(\mathbf{x})$ だけを学習すればよい。7.4 節の ResNet アーキテクチャはこのパターンに基づいている。残差接続のために特別な組み込

み層を用意する必要はどのフレームワークにもなく、カスタムモジュールでは `X + self.body(X)` というわずか 1 行で書ける。

```
class ResidualBlock(nnx.Module):
    def __init__(self, num_hiddens, *, rngs):
        self.body = nnx.Sequential(
            nnx.Linear(num_hiddens, num_hiddens, rngs=rngs), nnx.relu,
            nnx.Linear(num_hiddens, num_hiddens, rngs=rngs))

    def __call__(self, X):
        return X + self.body(X)
```

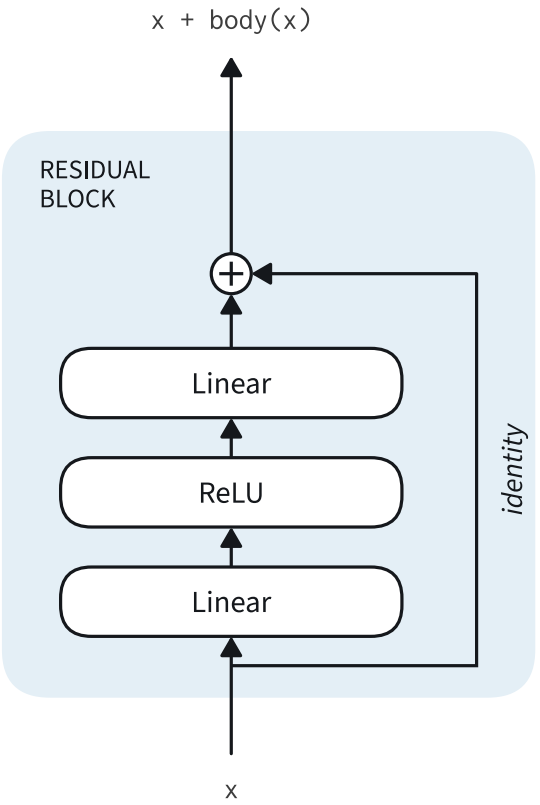


Figure 5.2: 残差ブロック。入力は本体（Linear、ReLU、Linear）と恒等経路に分岐し、加算によって再び合流する。

Figure 5.2 にその配線を示す。$\mathbf{x}$ と $f(\mathbf{x})$ を加算するには両者の形状が一致している必要があるため、残差ブロックは単一の隠れ層の幅 `num_hiddens` を受け取る。

```
block = ResidualBlock(24, rngs=nnx.Rngs(d2l.get_key()))
X24 = jax.random.normal(d2l.get_key(), (2, 24))
block(X24).shape
```

```
(2, 24)
```

順伝播メソッドは、入力でも訓練可能なパラメータでもない値に依存することもある。スカラー定数は通常の Python 属性として保持できる一方、データから追跡されるテンソル（入力の移動統計量など）は、オプティマイザに更新されることなく、モデルとともにチェックポイントに保存されデバイス間を移動しなければならない。こうした訓練対象外の**バッファ**を登録する方法は 5.2.1.2 節で扱う。

5.1.3.2 即時実行とコンパイル実行

ここまでの順伝播は通常の Python と同様に、演算を 1 つずつ実行してきた。しかし、Python のオーバーヘッドをなくして演算を融合するために、フレームワークは計算を一度トレースし、以降の呼び出しではコンパイル済みのグラフを再生できる。この高速化にはトレードオフがあり、コンパイル実行では順伝播内部でのデータ依存の Python 制御フローが制限される。

直接の呼び出しは即時実行されるが、`d2l.Trainer` は `nnx.jit` を用いて、形状とデータ型だけを持ち具体的な値を持たない抽象的な**トレーサ**上で `__call__` を一度トレースすることで訓練ステップをコンパイルする。Python の `while` ループでトレーサを条件に分岐しようとするとエラーが発生する。

```
class HalvingNet(nnx.Module):
    def __init__(self, rngs):
        self.proj = nnx.Linear(20, 24, rngs=rngs)

    def __call__(self, X):
        X = self.proj(X)
        while jnp.abs(X).sum() > 1:  # Depends on the values of X
            X = X / 2
        return X.sum()

net = HalvingNet(nnx.Rngs(d2l.get_key()))
print(net(X))
try:
    nnx.jit(lambda net, X: net(X))(net, X)
except jax.errors.TracerBoolConversionError as e:
    print(type(e).__name__)
```

```
-0.2457959
TracerBoolConversionError
```

即時実行（eager execution）ではデータ依存のループがそのまま動作する一方、トレース時にはシンボリックテンソルやトレーサを Python の `bool` へ変換することが拒否される（`tf.while_loop` や `jax.lax.while_loop` が必要になる）。`self.modules` の反復処理のような静的なモデル構造に対するループは、トレース時に問題なく展開される。

5.1.4 遅延初期化

5.1.4.1 データからの形状決定

多くの層ライブラリでは、構築時に出力次元だけを指定しておき、最初のミニバッチから入力幅を自動的に決めることができる。

NNX の層は明示的な入力幅と出力幅を必要とし、`__init__` の中で即座にパラメータを確保する。

```
rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(nnx.Linear(20, 256, rngs=rngs), nnx.relu,
                     nnx.Linear(256, 10, rngs=rngs))
[(path, value.shape)
 for path, value in nnx.state(net, nnx.Param).flat_state()]
```

```
[(('layers', 0, 'bias'), (256,)),
 (('layers', 0, 'kernel'), (20, 256)),
 (('layers', 2, 'bias'), (10,)),
 (('layers', 2, 'kernel'), (256, 10))]
```

5.1.4.2 ドライラン後の初期化

遅延層を使えば、深いモデルですぐにミスのもとになる層から層への入力次元の手計算から解放される。その代償として、データがネットワークを流れるまでパラメータが存在しないため、最初の順伝播より前にはパラメータ数を調べたり、Xavier 初期化（4.4.2.1 節）のような形状に依存するイニシャライザを実行したりできない。したがって遅延モデルでは、イニシャライザをあらかじめ記録しておくか、すべての形状を実体化させる**ドライラン**（出力を捨てるダミーのミニバッチに対する 1 回の順伝播）の後に適用する必要がある。

`nnx.Linear` は `kernel_init` を受け取り、コンストラクタ内で即座に重みをサンプリングして生成する。

```
class TinyMLP(d2l.Module):
    def __init__(self, rngs):
        super().__init__()
        self.net = nnx.Sequential(
            nnx.Linear(20, 256,
                       kernel_init=nnx.initializers.xavier_uniform(),
                       rngs=rngs), nnx.relu,
            nnx.Linear(256, 10, rngs=rngs))

model = TinyMLP(nnx.Rngs(d2l.get_key()))
[(path, value.shape)
 for path, value in nnx.state(model, nnx.Param).flat_state()]
```

```
[(('net', 'layers', 0, 'bias'), (256,)),
 (('net', 'layers', 0, 'kernel'), (20, 256)),
 (('net', 'layers', 2, 'bias'), (10,)),
 (('net', 'layers', 2, 'kernel'), (256, 10))]
```

遅延層と明示的な次元指定はどのように使い分けるべきだろうか。未知のデータセットで素早くプロトタイピングを行う際には、遅延層が便利である。一方、パラメータ数を数えたり、プログラムから初期化したり、ダミーデータなしで設定ファイルからインスタンス化したりしたい本番用のモデルでは、入力幅と出力幅を明示的に宣言する方が見通しがよく、本章の残りの部分でもこの流儀を採用する。

5.1.5 設定からのモデル構築

5.1.5.1 設定とビルダ

保存された重みテンソルだけから、訓練済みモデルを復元できるだろうか。完全には不可能である。訓練済みモデルは、数値配列とそれを解釈するコードの両方で構成されているからである。先ほどまでに書いた幅や深さはすべてコンストラクタ呼び出しに直接ハードコードされていたが、重みファイルはそうした選択を間接的にしか反映しない（テンソルの形状から層の幅はわかっても、ReLU か tanh かといったパラメータを持たない選択は配列のどこにも残らない）。

したがって、実験ごとに変わるアーキテクチャ上のハイパーパラメータは、評価指標とともに記録したりチェックポイントと並べて保存したりできる専用の**設定**オブジェクト（**config**）にまとめるべきであり、トポロジーそのもの（各ブロックが何を計算し、ブロック同士がどう接続されるか）はコード側に置く。以下では、Python の `dataclass` でアーキテクチャの 4 つのフィールドを保持し、ファクトリ関数 `build` によって任意の設定を完全なモジュール木へと変換する。

```
@dataclass
class MLPConfig:
    num_inputs: int = 784
    num_hiddens: int = 256
    num_blocks: int = 4
    num_outputs: int = 10

def build(cfg: MLPConfig, rngs=None) -> nnx.Module:
    rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
    blocks = [ResidualBlock(cfg.num_hiddens, rngs=rngs)
              for _ in range(cfg.num_blocks)]
    return nnx.Sequential(
        nnx.Linear(cfg.num_inputs, cfg.num_hiddens, rngs=rngs), *blocks,
        nnx.Linear(cfg.num_hiddens, cfg.num_outputs, rngs=rngs))
```

1 つの設定が、`num_inputs` から `num_hiddens` への入力射影（projection）、`num_blocks` 個の残差ブロック、および出力ヘッドからなるアーキテクチャのインスタンスを完全に規定する。なお、`blocks` は一時的な Python リストとして作られているが、逐次コンテナへアンパックして渡すことで各ブロックが即座に登録されるため、ここではまったく安全である。すべての層が入力幅と出力幅の両方を指定しているため、ドライランなしでただちにパラメータを調べたり数えたりできる。

```
net = build(MLPConfig())
[(path, value.shape)
 for path, value in nnx.state(net, nnx.Param).flat_state()][:4]
```

```
[(('layers', 0, 'bias'), (256,)),
 (('layers', 0, 'kernel'), (784, 256)),
 (('layers', 1, 'body', 'layers', 0, 'bias'), (256,)),
 (('layers', 1, 'body', 'layers', 0, 'kernel'), (256, 256))]
```

構築されたモデルは、784 次元ベクトル（平坦化した Fashion-MNIST 画像など）のミニバッチを 10 個の出力ロジットへ写像する。

```
net(jax.random.uniform(d2l.get_key(), (2, 784))).shape
```

```
(2, 10)
```

5.1.5.2 パラメータ数の集計

設定があれば、メモリを確保する前にモデルの正確なパラメータ数を紙の上で計算することもできる。入力射影は $784 \cdot 256 + 256 = 200{,}960$ 個のパラメータを持ち、各残差ブロックは 2 つの全結合層で合計 $2 \cdot (256 \cdot 256 + 256) = 131{,}584$ 個のパラメータを含み、出力層が $256 \cdot 10 + 10 = 2{,}570$ 個を加えるため、次のようになる。

$$200{,}960 + 4 \cdot 131{,}584 + 2{,}570 = 729{,}866 \tag{5.2}$$

全体ではこれだけのパラメータ数になり、その大半を残差ブロックが占めている。モデルをスケールアップするには、モデルのコードを書き換えるのではなく設定の 2 つのフィールドを変更するだけでよく、パラメータ木を走査すればこの計算が正しいことを検証できる。

```
for cfg in (MLPConfig(), MLPConfig(num_hiddens=512, num_blocks=8)):
    n = sum(x.size for _, x in nnx.state(build(cfg), nnx.Param).flat_state())
    print(f'num_hiddens={cfg.num_hiddens}, num_blocks={cfg.num_blocks}: '
          f'{n:,} parameters')
```

```
num_hiddens=256, num_blocks=4: 729,866 parameters
num_hiddens=512, num_blocks=8: 4,609,546 parameters
```

隠れ層の幅を 2 倍にすると各ブロック内のパラメータ数は 4 倍になり、ブロック数を 2 倍にするとその合計がさらに 2 倍になるため、大きい方の設定は 6 倍以上のパラメータを含むことになる。実世界の基盤モデル（foundation model）もまったく同じパターンに従っており、冒頭で挙げた 8B の言語モデルは、コンパクトな設定（たとえば 32 ブロック、隠れ層の幅 4,096）とモジュール木を実体化するビルダコードの組によって規定される。

なお、設定が保持するのはハイパーパラメータであり、実装コードではない点に注意が必要である。チェックポイントからモデルを復元するには、重み、設定、そして対応する `build` 関数およびブロック定義の 3 つすべてが揃っていなければならない。5.6 節では重みと設定を一緒に保存し、コードはバージョン管理に任せる。

ここまではパラメータの場所を特定して数を集計しただけで、その中身はまだ調べていない。5.2 節では、モデルが所有するテンソルの中身を開き、訓練がどれだけのメモリを消費するかを計算し、重みを共有したり凍結したりする方法を見ていく。

5.1.6 まとめ

モデル構築の根幹をなすのは 3 つの考え方である。第 1 に、すべてのニューラルネットワークはモジュールの木であり、各パラメータは根からのパスによって一意に指定される。第 2 に、オプティマイザ、チェックポイント、デバイス転送はいずれも登録された子モジュールを走査して動作するため、未登録のコンテナの内部に層を隠してしまうと、訓練や保存の対象から静かに除外される。第 3 に、可変な次元を軽量な設定へ切り出して `build` 関数と組み合わせることで、アーキテクチャの記録、スケールアップ、再現が容易になる。これらの基礎の上に、カスタムの順伝播メソッドは Python ならではの柔軟性をもたらし（ただしコンパイル実行下ではトレース上の制約を受ける）、遅延層は形状の生成をドライランまで遅らせる引き換えとして素早い実験に利便性を提供する。

5.1.7 演習問題

1. [code] **未登録のコンテナ。** NNX はモジュールを保持する通常の Python リストを拒否し、修正方法として `nnx.List` と `nnx.data` を挙げる。

   1. `ListMLP` を 2 通りに書き直せ。1 つは 2 つの層を `nnx.Dict` に `'hidden'` と `'out'` をキーとして格納したもの、もう 1 つは通常のリストを `nnx.data` でラップしたものである。両方のバージョンで 7,946 個すべてのパラメータが `nnx.state(net, nnx.Param)` に現れることを確認せよ。
   2. 2 つのバージョンのパラメータパスを `ListMLP` のものと比較せよ。チェックポイントは、パスから配列へのマッピングとして状態を保存する。`ListMLP` から保存されたチェックポイントを復元できるのは 2 つの書き直しのうちどちらか。また、それはなぜか。

2. **残差ブロックの幅の拡大。** `ResidualBlock` は入力幅と出力幅が一致していることを要求する。あるブロックが幅 d をより大きな幅 d' へ写像するものとし、その本体が幅 d' の層で終わり、スキップ経路も幅 d' を出力しなければならないとする。このスキップ経路を構築する方法を、パラメータを持つ場合と持たない場合でそれぞれ 1 つずつ挙げよ。各方法について、追加されるパラメータ数を求め、どの出力座標が引き続き恒等経路を受け取るかを答えよ。

3. [code] **ゼロからのモジュール実装。**●フレームワークの基底クラスを一切使わず、通常の Python と NumPy のみを用いて、配列をラップする `Parameter` クラスと、最小限の `Module` クラス（`__setattr__` によって代入されたすべての `Parameter` または `Module` を登録するもの）を実装せよ。`Module` には、登録された子モジュールを再帰的にたどる `parameters()` メソッドを持たせよ。さらに、重みとバイアスを保持する `Linear` を `Module` のサブクラスとして実装し、その 2 つのインスタンスから 2 層の MLP を構築して、`parameters()` が 4 つすべての配列を返すことを確認せよ。続いて、その 2 つの層を通常のリストに格納して `parameters()` が何を返すかを確認し、リスト版でも 4 つすべての配列を見つけられるように `Module` を拡張せよ。

   CMU 10-714、Homework 2、問題 2 を改変。

4. [code] **共有された子モジュール。** 1 つの子モジュールを複数の箇所で再利用すると、Figure 5.1 のモジュール木はグラフになる。同じ子モジュールを 2 箇所で使う小さなネットワークを構築せよ。共有された子モジュールのパラメータ集合は、`nnx.state(net, nnx.Param)` の中に 1 回現れるか、それとも 2 回現れるか。予測したうえで検証せよ。また、関数 `nnx.split(net)` はモジュールを静的なグラフ定義とその状態に分離し、`nnx.merge` はその 2 つからモジュールを再構築する。この往復変換の後も、2 つの箇所は `is` で判定して同一のオブジェクトのままであるか。

5.2 パラメータ、状態、メモリ

前節では、フレームワークがすべてのパラメータを自動的に見つけられるよう、モデルをモジュールの木構造として構成した。これまでは、モデルが他に何を保持しているかや、訓練にどれだけのメモリが必要かを問うことなく、こうしたテンソルの管理をフレームワークの裏側の処理に任せておけばよかった。小規模なうちは、そうした細部が問題になることはめったにない。しかし大規模になると、あらゆる設計判断がそれに左右される。5.1 節で取り上げた Llama 3 のような 80 億パラメータのモデルを考えてみよう。その重みを 32 ビット浮動小数点数で格納するだけでも 32 GB を要するが、これを**訓練**するには、データを 1 例も処理する前の段階でその 4 倍（128 GB）ものメモリが必要になる。すべてのパラメータに対応する勾配が必要であるうえ、Adam オプティマイザ（9.5 節）が勾配と二乗勾配の移動平均のために同じサイズのテンソルをさらに 2 つ保持するからである。これだけで、すでに 80 GB GPU のメモリを超えてしまう。

このようなメモリ予算を正確に計算するには、モデルが所有するすべてのテンソルとそれぞれの役割を把握する必要がある。日常的なワークフローも、まったく同じ一覧に依存している。チェックポイントには順伝播が依存するすべての永続テンソルを記録しなければならず、GPU への転

送ではそれらをまとめて移動する必要があり、ファインチューニングではオプティマイザが更新してよいテンソルを明確に指定しなければならない。

フレームワークは、これらすべての操作を**登録**（registration、5.1 節）によって統一している。各層が構造化された名前の下で自身のテンソルをモジュール木（module tree）に登録することで、オプティマイザ、チェックポイント保存処理、デバイス転送ルーチンがその単一の木を走査できるようになる。本節では、この状態木（state tree）を 3 つのパートに分けて見ていく。第 1 に、訓練可能な**パラメータ**と訓練対象外の**バッファ**を区別する。第 2 に、訓練時のメモリをパラメータあたりのバイト数で見積もり、ストレージを確保せずにパラメータ数を数える方法を示す。第 3 に、言語モデリングや転移学習（transfer learning）の随所で用いられる 2 つの代表的なパラメータのパターンとして、重み結合（weight tying）とパラメータ凍結（parameter freezing）を取り上げる。

5.2.1 状態木

5.2.1.1 パスによるアクセス

本節を通して用いる例は、コンパクトな残差 MLP である。これは、20 個の特徴量を 64 個の隠れユニット（hidden unit）に射影する入力層、幅 64 の残差ブロック（residual block）2 つ、そして 10 ユニットの出力層からなる。このノートブック単体で動作するよう、5.1 節の `ResidualBlock` をここで再定義しておく。各ブロックは、2 層の**本体**（body）の出力を入力に直接加算する。

```
class ResidualBlock(nnx.Module):
    def __init__(self, num_hiddens, *, rngs):
        self.body = nnx.Sequential(
            nnx.Linear(num_hiddens, num_hiddens, rngs=rngs), nnx.relu,
            nnx.Linear(num_hiddens, num_hiddens, rngs=rngs))

    def __call__(self, X):
        return X + self.body(X)

rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(nnx.Linear(20, 64, rngs=rngs),
                     ResidualBlock(64, rngs=rngs),
                     ResidualBlock(64, rngs=rngs),
                     nnx.Linear(64, 10, rngs=rngs))
X = jax.random.normal(rngs(), (2, 20))
net(X).shape
```

```
(2, 10)
```

パラメータはモジュール木の葉に位置する。任意の葉に至るドット区切りのパス（Figure 5.1）は Python でのインデックス参照に対応しており、デバッグ時に単一の層を調べるのに便利である。一方、オプティマイザやチェックポイントの書き出し処理は**すべて**の葉を一度に必要とするため、各フレームワークにはすべてのパラメータをそのパスとともに取得する木全体の走査手段が用意されている。

NNX では `nnx.state(net, nnx.Param)` がこれにあたり、フラットなビューから各配列とそのパスが得られる。パラメータ自体は勾配を保持せず、`nnx.grad` が同じパスを持つ別の状態木として勾配を返す。

この残差 MLP には合計 12 個のパラメータテンソルが含まれている。以下では、その総数とともに、最初の残差ブロックに属する 4 つのテンソルを確認する。

```
params = nnx.state(net, nnx.Param)
flat = list(params.flat_state())
len(flat), [(path, leaf.shape) for path, leaf in flat[2:6]]
```

```
(12,
 [(('layers', 1, 'body', 'layers', 0, 'bias'), (64,)),
  (('layers', 1, 'body', 'layers', 0, 'kernel'), (64, 64)),
  (('layers', 1, 'body', 'layers', 2, 'bias'), (64,)),
  (('layers', 1, 'body', 'layers', 2, 'kernel'), (64, 64))])
```

`('layers', 1, 'body', 'layers', 0, 'kernel')` のようなパスのタプルは、`net.layers[1].body.layers[0].kernel` に対応している。訓練可能なパラメータのみが存在する場合、`nnx.state(net)` は `nnx.state(net, nnx.Param)` と一致する。

```
([path for path, _ in nnx.state(net).flat_state()]
 == [path for path, _ in params.flat_state()])
```

```
True
```

フレームワークのあらゆるユーティリティは、これらのパスの下で同じ状態木を走査する。上記の等価性が成り立つのは、この残差 MLP 内のすべてのテンソルがたまたま訓練可能だからであり、すべての層でそうとは限らない。

5.2.1.2 パラメータとバッファ

訓練セットから一度だけ計算した特徴量ごとの平均と標準偏差を用いて、入力を標準化するモデルを考える。これらの統計量はチェックポイント内に保存しなければならず（保存しなければ推論時の予測が誤ったものになる）、重みとともに GPU へ移動する必要もある。しかし、これらは学習される重みではなくデータの統計量であるため、オプティマイザが勾配降下法（gradient descent）によって更新しては**決して**ならない。モジュールの走査では通常の Python 属性は無視されるため（5.1 節）、通常の属性として保持するだけではうまくいかない。バッチ正規化（batch normalization、7.3 節、(Ioffe and Szegedy 2015)）も、まったく同じ種類の状態に依存している。すなわち、訓練時の順伝播で更新され、推論時に適用される移動平均と移動分散である。

このような永続的だが訓練対象外のテンソルは一般に**バッファ**と呼ばれ、どのフレームワークでも登録が可能になっている。登録しておけば、チェックポイント保存やデバイス間の移動には含まれる一方、オプティマイザからはスキップされる。ここから、層内のあらゆるテンソルに対して明快な 3 通りの使い分けが導かれる。オプティマイザで更新すべきものは**パラメータ**として登録し、勾配は不要だがチェックポイントやデバイス転送で必要なものは**バッファ**として登録し、1 回の順伝播・逆伝播の間だけ存在するもの（4.3 節で扱う誤差逆伝播法（backpropagation）のためにキャッシュされる中間活性化値（activation）など）は通常のローカル変数のままにする。層の幅のようなアーキテクチャ上の定数は、通常の Python 属性のままでよい。後に参照できるよう、Table 5.1 にこれらの規則をまとめる。

Table 5.1: 層内の 4 種類の値と、オプティマイザ・チェックポイント・デバイス移動におけるそれぞれの扱い。

種類	例	オプティマイザによる更新	チェックポイントへの保存	モデルと一緒の移動
パラメータ	重み、バイアス	あり	あり	あり

種類	例	オプティマイザによる更新	チェックポイントへの保存	モデルと一緒の移動
バッファ	入力統計量、BatchNorm の移動平均	なし	あり	あり
通常の属性	層の幅	なし	なし	なし
ローカル変数	中間活性化値	なし	なし	なし

具体例として、事前計算された特徴量ごとの平均と標準偏差で入力を正規化してから線形層を適用する `Standardizer` モジュールを構築する。

`nnx.Variable` をサブクラス化すること（ここでは `class Buffer(nnx.Variable)`）で、バッファを `nnx.Param` と区別する。

```
class Buffer(nnx.Variable):
    pass

class Standardizer(nnx.Module):
    def __init__(self, mean, std, *, rngs):
        self.mean = Buffer(mean)
        self.std = Buffer(std)
        self.out = nnx.Linear(4, 2, rngs=rngs)

    def __call__(self, X):
        return self.out((X - self.mean) / self.std)

sample = jax.random.normal(rngs(), (100, 4)) * jnp.arange(1., 5.)
standardizer = Standardizer(sample.mean(0), sample.std(0), rngs=rngs)
[(path, value.shape) for path, value in nnx.state(standardizer).flat_state()]
```

```
[(('mean',), (4,)),
 (('out', 'bias'), (2,)),
 (('out', 'kernel'), (4, 2)),
 (('std',), (4,))]
```

訓練可能な状態と訓練対象外の状態を調べると、チェックポイントからは 4 つのテンソルすべてが見える一方、オプティマイザからは線形層の重みとバイアスしか見えないことが確認できる。

```
{'params': list(nnx.state(standardizer, nnx.Param)),
 'buffers': list(nnx.state(standardizer, Buffer))}
```

```
{'params': ['out'], 'buffers': ['mean', 'std']}
```

登録が制御するのはチェックポイント保存と最適化だけではない。dtype のキャスト（5.5 節）や GPU への転送（5.7 節）といったモデル全体に対する操作もまったく同じ状態木を走査し、パラメータとバッファを同様に移動する一方で、未登録の通常の属性はそのまま残す。

5.2.2 訓練時のメモリ

5.2.2.1 パラメータあたりのバイト数

この残差 MLP を訓練するには、どれだけのメモリが必要だろうか。モデルとオプティマイザの状態だけに注目すると、メモリ量はパラメータごとの 3 つの要素、すなわちパラメータそのもの、その勾配、そしてパラメータごとのオプティマイザ状態に比例する。通常のミニバッチ SGD は状態を保持しないため、32 ビット浮動小数点数（fp32、1 数値あたり 4 バイト）ではパラメータあたり 8 バイトを消費する。Adam はこれに加えて、勾配と二乗勾配それぞれの指数移動平均（exponential moving average）として、パラメータごとに 2 つの fp32 テンソルを保持する。大規模な訓練のほぼすべてが Adam またはその派生手法を用いるため、この 16 バイトという値が実用上の基準となる。

訓練時の状態	精度	パラメータあたりのバイト数
重み	fp32	4
勾配	fp32	4
Adam の勾配の平均	fp32	4
Adam の二乗勾配の平均	fp32	4
合計		16

このパラメータあたり 16 バイトという値は、厳密な下界にすぎない。ここには、誤差逆伝播法のために順伝播中に保存され、バッチサイズや系列長（sequence length）に応じて増大する中間**活性化値**や、一時的なカーネル作業領域、アロケータの断片化は含まれていない。これらはすべて 5.7 節で GPU 上で直接計測する。この残差 MLP の場合、基準値は次のようになる。

```
leaves = jax.tree_util.tree_leaves(nnx.state(net, nnx.Param))
n = sum(x.size for x in leaves)
weights = sum(x.size * x.dtype.itemsize for x in leaves)  # fp32: 4 bytes
grads, adam_state = weights, 2 * weights
print(f'{n} parameters: {(weights + grads + adam_state) / 2**20:.2f} MiB '
      'for weights + gradients + Adam state')
```

```
18634 parameters: 0.28 MiB for weights + gradients + Adam state
```

このパラメータ数は手計算でも確かめられる。入力層が $20 \times 64 + 64 = 1344$ 個、各残差ブロックが $2 \times (64 \times 64 + 64) = 8320$ 個、出力層が $64 \times 10 + 10 = 650$ 個で、合計 18,634 個のパラメータとなる。Adam のオプティマイザ状態は表の上だけの理論値ではなく、テンソルごとに実際に確保されるメモリであり、モデルの完全なコピーを 2 つ分追加する。

```
optimizer = nnx.Optimizer(net, optax.adam(1e-3), wrt=nnx.Param)
grads = nnx.grad(lambda model: model(X).sum())(net)
optimizer.update(net, grads)
moments = sum(x.size for x in jax.tree.leaves(optimizer.opt_state)
              if hasattr(x, 'ndim') and x.ndim > 0)
moments == 2 * n
```

```
True
```

`nnx.Optimizer` は更新が行われる前の構築時点で、両方のモーメントを確保する。

この小さな MLP では、その基準値は 0.28 MiB とごくわずかであり、これまでの章でメモリをほとんど気にせずに済んだのはそのためである。しかし、まったく同じ計算を 80 億パラメータ

のモデルに拡大すると、バッチサイズにかかわらずたちまち 128 GB に達する。16 ビット**混合精度**（mixed-precision、5.5 節）訓練を使えば、この静的な下限を半分に減らせると期待するかもしれない。ところが意外なことに、そうはならない。混合精度では微小な更新が丸めによって消えてしまわないよう、重みのマスターコピーと Adam のモーメントを fp32 で保持しており (Micikevicius et al. 2018)、順伝播で用いる 16 ビット値はそれらからキャストして作られるからである。

5.2.2.2 メモリ確保なしのカウント

ここまでのパラメータ数カウントはすべて、すでにメモリ上に確保されたテンソルを調べていた。8B モデルではこれでは順序が逆転してしまう。32 GB の重みが RAM に収まるかを確認するために、まず 32 GB を確保しなければならないようでは本末転倒である。幸い、パラメータ数を数えるのに必要なのは形状（shape）と dtype だけであり、どちらもコンストラクタの引数のみから決まる。例として、4,096 次元の隠れ状態（hidden state）を語彙の 128,256 個の**トークン**に対するロジットへ写す Llama 3 8B の出力射影（output projection）を考えよう。その重み行列は $4096 \times 128{,}256 = 525{,}336{,}576$ 個の要素を持つ（fp32 で 2.1 GB）。以下では、ストレージを確保せずにこの層をインスタンス化し、パラメータ数を数える。

`nnx.eval_shape` はコンストラクタを抽象的にトレースし、配列を一切確保することなく形状と dtype を持つ `jax.ShapeDtypeStruct` の葉を返す。

```
key = d2l.get_key()  # drawn outside the abstract trace
head = nnx.eval_shape(lambda: nnx.Linear(
    4096, 128256, use_bias=False, rngs=nnx.Rngs(key)))
leaves = jax.tree.leaves(nnx.state(head, nnx.Param))
n_head = sum(x.size for x in leaves)
print(leaves, f'{n_head:,} params, {4 * n_head / 1e9:.1f} GB in fp32')
```

```
[ShapeDtypeStruct(shape=(4096, 128256), dtype=float32)] 525,336,576 params, 2.1 GB in fp32
```

8B モデルのすべての層にわたって同じ抽象的な検査を実行すれば、テンソル用のメモリを 1 バイトも確保することなく、正確な 80 億個のパラメータ数と訓練時の下限である 128 GB が得られる。

5.2.3 パラメータの共有と凍結

5.2.3.1 パラメータの結合

単一のパラメータテンソルが、ネットワーク内で複数の役割を担うことがある。典型的な例は言語モデル（8.3 節）であり、そこでは語彙 V からトークンを読み取って次のトークンを予測する。その入力**埋め込み**（embedding）は $|V| \times d$ のルックアップテーブルであり、v^{th} の行は d 次元で表されたトークン v の表現になっている。一方の出力ヘッドは、d 次元の隠れ状態を $|V|$ 個のトークンロジットへと射影するものであり、まったく同じ幾何構造と意味を持つもう 1 つの $|V| \times d$ テーブルである。両方の層を 1 つの共有重み行列に**結合**すると、$|V| \times d$ 個のパラメータが削減されるうえ、汎化性能（generalization）も向上することが多い (Press and Wolf 2017; Inan et al. 2017)。入力として互換的に振る舞うトークンは出力としても当然似たスコアを得るべきであり、重み結合によって単一の統一されたトークン空間の幾何構造が保たれるからである。メモリの節約効果も劇的になり得る。GPT-2 small (Radford et al. 2019) では、結合された 50257×768 のテーブルがモデル全体の 1 億 2400 万パラメータのうち約 3860 万個、すなわちネットワーク全体のほぼ 3 分の 1 を占めている。

Figure 5.3 にこの 2 つの呼び出し箇所を示す。語彙サイズ 100 トークン、隠れ層の幅 $d = 16$ の小規模な言語モデル `ToyLM` でその仕組みを確かめてみよう。

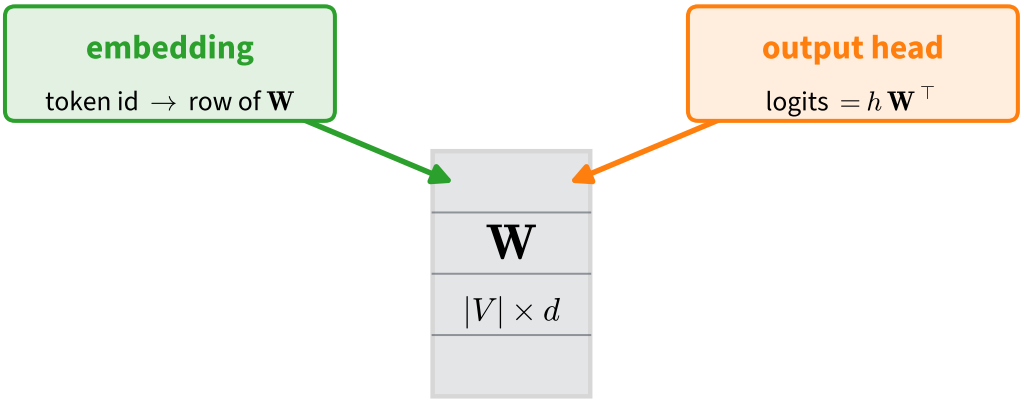


Figure 5.3: 1 つの重み行列 W と 2 つの呼び出し箇所。埋め込みはトークン ID で行を参照し、出力ヘッドはその転置を掛けてロジットを生成する。結合とは両者が同じテンソルを指すことを意味し、両方の使用箇所からの勾配は 1 か所に合算される。

`nnx.Embed.attend(h)` は、`self.emb` の埋め込みテーブルの転置を直接用いて隠れ状態を射影する。

```
class ToyLM(nnx.Module):
    def __init__(self, vocab_size, d, tied=True, *, rngs):
        self.tied = tied
        self.emb = nnx.Embed(vocab_size, d, rngs=rngs)
        self.body = nnx.Linear(d, d, rngs=rngs)
        self.head = None if tied else nnx.Linear(
            d, vocab_size, use_bias=False, rngs=rngs)

    def __call__(self, tokens):
        h = nnx.relu(self.body(self.emb(tokens)))
        if self.tied:
            return self.emb.attend(h)   # the embedding table, used as the head
        return self.head(h)

lm = ToyLM(vocab_size=100, d=16, rngs=rngs)
tokens = jax.random.randint(rngs(), (2, 8), 0, 100)
[(path, value.shape)
 for path, value in nnx.state(lm, nnx.Param).flat_state()]
```

```
[(('body', 'bias'), (16,)),
 (('body', 'kernel'), (16, 16)),
 (('emb', 'embedding'), (100, 16))]
```

共有された行列は 1 回数えられるのだろうか、それとも 2 回数えられるのだろうか。重みを結合したモデルと結合しないモデルを比較してみよう。

```
untied = ToyLM(vocab_size=100, d=16, tied=False, rngs=rngs)
(sum(x.size for x in jax.tree.leaves(nnx.state(lm, nnx.Param))),
 sum(x.size for x in jax.tree.leaves(nnx.state(untied, nnx.Param))))
```

```
(1872, 3472)
```

重みを結合したモデルのパラメータ数は $1600 + 272 = 1872$ 個（埋め込みテーブル 1 つと本体の層）であるのに対し、結合しないモデルでは 2 つ目の 100×16 テーブルが加わり、合計 3472 個となる。

1 つのテンソルが 2 回使われるとき、誤差逆伝播法では何が起きるだろうか。多変数の連鎖律（chain rule）により、各呼び出し箇所がそれぞれの勾配項をもたらし、その両方が共有テンソルの勾配に合算される。重みを結合しないモデルの 2 つのテーブルを $\mathbf{W}_{\text{emb}}$ および $\mathbf{W}_{\text{head}}$ と書き、結合行列 $\mathbf{W}$ と同じ値に初期化すると、2 つのモデルはまったく同じ関数を計算し、次の関係を満たす。

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{W}_{\text{emb}}} + \frac{\partial L}{\partial \mathbf{W}_{\text{head}}}. \tag{5.3}$$

結合された重みを結合しないモデルへコピーし、同じバッチに対する勾配を比較して、この恒等式を数値的に確かめよう。

```
untied.emb.embedding[...] = lm.emb.embedding
untied.body.kernel[...] = lm.body.kernel
untied.body.bias[...] = lm.body.bias
untied.head.kernel[...] = lm.emb.embedding.T
g_tied = nnx.grad(lambda model: model(tokens).sum())(lm)
g_untied = nnx.grad(lambda model: model(tokens).sum())(untied)
jnp.allclose(g_tied['emb']['embedding'],
             g_untied['emb']['embedding'] + g_untied['head']['kernel'].T)
```

```
Array(True, dtype=bool)
```

5.2.3.2 パラメータの凍結

事前学習済みのネットワークを新しいタスクでファインチューニングする際（32.2 節（英語版））には、事前学習済みのバックボーンを固定したまま新しい出力ヘッドだけを訓練する戦略がよくとられる。これにより、必要なデータ量とオプティマイザのメモリの双方を大幅に削減できる。勾配による更新が行われないようにパラメータを固定することを**凍結**と呼ぶ。以下では、新しく用意した残差 MLP の出力ヘッド以外のすべての層を凍結し、18,634 個のパラメータのうち 650 個だけを訓練可能にする。

NNX では、`trainable` フィルタで出力ヘッド（`layers[3]`）を選択し、それを `nnx.grad` と `nnx.Optimizer(..., wrt=trainable)` の両方に渡す。

```
finetune = nnx.Sequential(nnx.Linear(20, 64, rngs=rngs),
                          ResidualBlock(64, rngs=rngs),
                          ResidualBlock(64, rngs=rngs),
                          nnx.Linear(64, 10, rngs=rngs))
```

```
trainable = nnx.All(nnx.Param, lambda path, _: path[:2] == ('layers', 3))

def count(state):
    return sum(x.size for x in jax.tree.leaves(state))

n_train = count(nnx.state(finetune, trainable))
(n_train, count(nnx.state(finetune, nnx.Param)))
```

```
(650, 18634)
```

この状態で Adam のステップを 1 回実行すると、出力ヘッドだけが更新され、それより前の層はすべて変化しないはずである。ステップの前後ですべてのパラメータを確認してみよう。

```
head_opt = nnx.Optimizer(finetune, optax.adam(0.1), wrt=trainable)
before = nnx.to_pure_dict(nnx.state(finetune, nnx.Param))
grads = nnx.grad(lambda model: model(X).sum(),
                 argnums=nnx.DiffState(0, trainable))(finetune)
head_opt.update(finetune, grads)
stepped = nnx.to_pure_dict(nnx.state(finetune, nnx.Param))
jax.tree_util.tree_map(lambda a, b: bool((a != b).any()), before, stepped)
```

```
{'layers': {0: {'bias': False, 'kernel': False},
  1: {'body': {'layers': {0: {'bias': False, 'kernel': False},
     2: {'bias': False, 'kernel': False}}}},
  2: {'body': {'layers': {0: {'bias': False, 'kernel': False},
     2: {'bias': False, 'kernel': False}}}},
  3: {'bias': True, 'kernel': True}}}
```

出力ヘッドの重みとバイアスだけが変化し、オプティマイザは訓練可能なその 650 個のパラメータに対してのみ Adam のモーメントを確保した。パラメータの凍結には見落としやすい落とし穴が 2 つあり、どちらもエラーを送出しない。第 1 に、パラメータを凍結しても、すでに確保されたオプティマイザ状態は解放され**ない**。すべてのパラメータを対象に構築した Adam オプティマイザですでにステップを実行していた場合、後から凍結した層についてもパラメータあたり 8 バイトの移動モーメントが保持され続ける。18,634 パラメータであれば無害だが、80 億パラメータ規模では大きな負担となる（演習問題で扱う）。

第 2 の落とし穴はさらに見落としやすい。凍結は勾配を遮断するが、層の状態の中には勾配**なしに**書き換わるものがある。特にバッチ正規化（7.3 節）は、訓練モードで動作している限り、オプティマイザを一切経由せず**順伝播**の途中で移動平均と移動分散を更新する。そのため、BatchNorm 層の訓練可能な重みを凍結したときに移動統計量も固定されるかどうかは、フレームワークによって異なる。以下では、オプティマイザのステップを一切行わずに、訓練モードのまま凍結した BatchNorm 層へ平均をずらしたバッチを通し、移動平均がドリフトするかどうかを確認する。

```
bn = nnx.BatchNorm(4, use_running_average=False, momentum=0.9,
                   use_fast_variance=False, rngs=rngs)
before = bn.mean[...]
bn(jax.random.normal(rngs(), (8, 4)) + 3)   # training mode
bool(jnp.allclose(bn.mean, before))
```

```
False
```

移動平均が変化した。パラメータの勾配の凍結（`nnx.Param` フィルタ）と、層の評価モードへの切

り替え（`use_running_average=True`）は別物である。ファインチューニング中に BatchNorm 層を完全に固定するには、両方を設定する必要がある。

本節で見てきたすべてのパラメータ（結合されたもの、凍結されたもの、平均をとるもの）は、フレームワークが与えた初期値から出発していた。次節では、そうした初期値がどのように選ばれるのか、またモジュール木全体にわたって初期化（initialization）をどのように制御するのかを調べる。

5.2.4 まとめ

モデルの永続状態は名前付きテンソルの木をなしており、フレームワークの各操作からどのテンソルが見えるかは登録によって決まる。オプティマイザは**パラメータ**を更新し、チェックポイントやデバイス転送には**バッファ**（移動統計量のような勾配を持たない永続テンソル）も含まれる一方、未登録の属性は黙ってスキップされる。fp32 と Adam による訓練では、活性化値を考慮する前の段階でパラメータあたり 16 バイト（重み、勾配、2 つの Adam モーメントであり、8B モデルでは 128 GB）の基準メモリが必要となるが、抽象的な形状検査を用いればメモリを確保せずにその予算を計算できる。重み結合は複数の役割にわたって 1 つのテンソルを共有させることで、カウントを 1 回に抑えつつ全呼び出し箇所からの勾配の和を受け取る。また、凍結は勾配による更新を止めるが、既存のオプティマイザ状態を解放するわけではなく、順伝播における BatchNorm の移動統計量の更新も止めるとは限らない。

5.2.5 演習問題

1. [code] **移動統計量**。この問題で必要なのは、7.3 節のバッチ正規化層に関する次の更新式だけである。訓練モードでは、この層はミニバッチ平均 $\widehat{\mu}_{\mathcal{B}}$ と分散を用いて正規化し、各訓練ステップで移動平均 $\bar{\mu}$ を次式により更新する。

$$\bar{\mu} \leftarrow \rho\,\bar{\mu} + (1-\rho)\,\widehat{\mu}_{\mathcal{B}}. \tag{5.4}$$

   1. 新しい層は $\bar{\mu} = \mathbf{0}$ から始まる。すべての特徴量でミニバッチ平均が 3 となるバッチを訓練モードで処理したとき、$\bar{\mu}$ が k バッチ後にどの値になるか予測せよ。オプティマイザのステップを一切行わずに、4 個の特徴量に対する新しい層で $k = 1, 5, 20$ の場合の予測を確かめ、この層がデフォルトで用いる ρ の値を読み取れ。
   2. $\bar{\mu}$ を訓練可能なパラメータとして登録し、上記の更新を取り除いたとする。訓練損失は $\bar{\mu}$ にどのような勾配を送るか。また、そのとき訓練損失と評価時の正解率は、変更を加えていないモデルと比べてどうなるか。
   3. $\bar{\mu}$ のみに依存する損失であって、各ミニバッチに対する学習率 η の勾配ステップが上記の取り除いた更新を再現するものを求め、一致させる η を ρ を用いて答えよ。

2. [code] **メモリ使用量**。すべてのパラメータを訓練する fp32 の Adam 訓練において、`net` の各トップレベルブロックについて状態要素数とバイト数を報告する補助関数を書け。

   1. どのブロックが支配的か。出力層の幅がいくつになると、1 つの残差ブロックを上回るか。
   2. 2 つの残差ブロックの間に、64 個の隠れ特徴量に対するバッチ正規化層を挿入せよ。この層の移動統計量のコストを正しく見積もるように補助関数を拡張し、各ブロックの 1 要素あたりのバイト数を報告せよ。全バイト数に占める割合が全要素数に占める割合より大きくなるのはどのブロックで、小さくなるのはどのブロックか。両方の理由を説明せよ。

3. [code] **結合と凍結**。本節の重み結合された `ToyLM` を考える。

1. たとえば `nnx.All(nnx.Param, nnx.Not(nnx.PathContains('emb')))` を用いて埋め込みテーブルを訓練可能フィルタから除外し、そのフィルタを `nnx.grad` と `nnx.Optimizer` に渡せ。訓練されるパラメータはいくつで、ヘッドはなお適応できるか。ここから、結合と凍結の相互作用について何がわかるか。
2. `lm` の値から出発し、埋め込みは固定したままヘッドだけが適応するモデルを構築し、1 回の更新後に両方の性質が成り立つことを確かめよ。全体を訓練可能にした `lm` と比べて、格納するパラメータ数と訓練するパラメータ数はそれぞれいくつか。

4. **ファインチューニングのメモリ**。ある言語モデルが 80 億個のパラメータを持つとする。出力層は埋め込みとは結合されていない $4096 \times 128{,}256$ の行列であり、状態はすべて fp32、オプティマイザは Adam である。

   1. すべてのパラメータを訓練するとき、重み、勾配、および Adam の移動平均は何バイトを占めるか。その状態だけで 80 GB GPU 何枚分を埋めるか。
   2. 出力層以外をすべて凍結し、出力層のみを対象にオプティマイザを構築せよ。このとき永続状態は何バイトを占め、1 枚の 80 GB GPU に収まるか。
   3. 代わりに、全パラメータを対象に構築したオプティマイザで凍結前に 1 ステップを実行し、その後もそのまま使い続けたとする。確保されたまま残るのは何バイトで、その状態はなお収まるか。

5.3 初期化

これまで訓練してきたすべてのニューラルネットワークは、通常フレームワークのデフォルトのイニシャライザによって抽出されたランダムな重みから出発していた。しかし 4.4 節で見たように、この出発点はきわめてシビアになり得る。各層が活性化値をわずかに増幅または減衰させるだけでも、層を深く積み重ねるとその倍率が指数関数的に積み重なり、訓練が始まる前に立ち往生してしまうからである。

同節では、単一の層に対する解決策も導出した。すなわち、標準偏差 $g/\sqrt{n_{\text{in}}}$ で重みを抽出する方法である。ここで、n_{in} は層の入力幅であり、**ゲイン** g は直後に続く活性化関数を補正する係数で、ReLU の前では $g = \sqrt{2}$（He 初期化、4.4.2.2 節）、線形出力では $g = 1$ とする。この規則のもとでは、各層は入力の二乗平均を保つ。

この規則をすべての層に適用すれば、初期化の問題は完全に解決すると思うかもしれない。ところが驚くべきことに、そうはならない。残差ネットワーク（residual network）では、個々の層がすべて He 初期化に従っていても、深さとともに活性化値が爆発してしまうのである。各残差ブロックは、その本体 f（5.1 節）の出力を入力に加算する。本体が入力の二乗平均を保ち、かつ入力と無相関な出力を生成するなら、両者の和の二乗平均は **2 倍**になる。32 個の残差ブロックを経ると、勾配ステップを 1 回も進める前に、活性化値の標準偏差は $2^{16} \approx 65{,}000$ 倍に膨れ上がる。

したがって、適切な初期化は個々の層の種類だけでなく**アーキテクチャの配線**にも依存するため、モジュールツリー全体にカスタム規則を適用するための見通しのよい仕組みが必要になる。本節の前半では、モジュールツリー全体へのイニシャライザの適用、切断正規分布による極端な抽出値の抑制、カスタムイニシャライザの作成、重みのインプレース編集といった具体的な手法を扱う。後半ではじっくりと、残差ブロックをまたぐ分散の増大を解析し、各ブロックの出力射影（output projection）を縮小またはゼロ化することでこれを解決する。

5.3.1 イニシャライザの適用

4.4 節で見たように、線形層のデフォルトのイニシャライザは He 初期化（4.4.2.2 節）を適用しない。モデル全体に He 初期化（あるいは後ほど導出する深さ依存の残差スケーリング）を適用するには、モジュールツリーを走査する（5.1 節）。以下では、幅 256 の 10 層 ReLU スタック（バ

イアスなし）に標準正規分布から抽出した 1,024 個の入力を与え、デフォルトの初期化と He 初期化を比較する。

NNX のデフォルトは LeCun の正規初期化（He の分散の半分）である。`nnx.iter_modules(model)` を走査し、`nnx.Linear` 層ごとに新しい乱数キーを分割して、`layer.kernel[...]` を `nnx.initializers.he_normal()` でインプレースに更新する。

```
rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(*[module for _ in range(10) for module in
                       (nnx.Linear(256, 256, use_bias=False, rngs=rngs),
                        nnx.relu)])
X = jax.random.normal(d2l.get_key(), (1024, 256))
print(f'default: mean square of the output {(net(X) ** 2).mean():.2g}')

def reinit(model, key, init_fn):
    layers = [m for _, m in nnx.iter_modules(model)
              if isinstance(m, nnx.Linear)]
    for layer, subkey in zip(layers, jax.random.split(key, len(layers))):
        layer.kernel[...] = init_fn(
            subkey, layer.kernel.shape, layer.kernel.dtype)

reinit(net, d2l.get_key(), nnx.initializers.he_normal())
print(f'He:      mean square of the output {(net(X) ** 2).mean():.2g}')
```

```
default: mean square of the output 0.00095
He:      mean square of the output 0.56
```

デフォルトのイニシャライザのもとでは、すべての層が分散に 1 未満の係数を掛けるため、出力の二乗平均は何桁も潰れてしまう。He 初期化に切り替えると、出力の二乗平均は 1（入力の二乗平均）から約 2 倍以内の範囲へと一気に戻る。残りのばらつきは、有限幅に起因するランダムな揺らぎが 10 層にわたって積み重なったものにすぎない。

5.3.1.1 切断正規分布

標準ガウス分布は分散こそ適切になるものの、その裾は無限遠まで広がっている。小さな層ではめったに問題にならないが、大規模モデルが持つ 10^8 個の重み全体で見ると、標準偏差の 5 倍を超える要素が数十個はほぼ確実に生じる。極端な初期の外れ値を避けるため、BERT や Vision Transformer などの現代的な言語モデルや視覚モデルは、σ の定数倍の外側にあるサンプルを棄却して引き直す**切断正規分布**（truncated normal）から抽出する (Devlin et al. 2018; Dosovitskiy et al. 2021)。裾を切り落とすと分散もわずかに小さくなるが、フレームワークの実装によっては再スケーリングでこれを補正するものもあれば、そのままにするものもある。

`nnx.initializers.truncated_normal` は σ の倍数で切断位置を受け取る（デフォルトは `lower=-2.0, upper=2.0`）。

```
w = nnx.initializers.normal(stddev=0.02)(d2l.get_key(), (1000, 1000))
print(f'normal:    std {w.std():.4f}, max weight {jnp.abs(w).max():.4f}')
w = nnx.initializers.truncated_normal(stddev=0.02)(d2l.get_key(), (1000, 1000))
print(f'truncated: std {w.std():.4f}, max weight {jnp.abs(w).max():.4f}')
```

```
normal:    std 0.0200, max weight 0.0957
truncated: std 0.0176, max weight 0.0400
```

$\sigma = 0.02$ の通常の正規分布から 100 万個を抽出すると、極端な重みは $5\sigma = 0.1$ に迫るが、切断正規分布では $|w| \leq 0.04$ が保証される。裾を切り落とすことで、経験標準偏差は 0.02 から約 0.0176 に低下する。BERT のように固定した σ を用いるレシピではこのわずかな減少をそのまま受け入れる一方、分散保持（variance-preserving）型の方式では切断後に再スケーリングして目標の分散に正確に合わせる。

5.3.2 カスタムイニシャライザと直接編集

組み込みのイニシャライザが論文やアーキテクチャの要件に合わない場合は、独自のイニシャライザを作成するか、パラメータテンソルを直接編集できる。いずれも各フレームワークのイニシャライザのインターフェースに基づいている。

NNX では、イニシャライザはシグネチャ `(key, shape, dtype) -> array` を持つ任意の関数である。

5.3.2.1 任意のゲインを持つ規則

それでは、冒頭で述べたゲイン付きの正規分布規則を実装しよう。すなわち、重みを標準偏差 $g/\sqrt{n_{\text{in}}}$（g は任意のゲイン）で抽出する規則である。He 初期化は $g = \sqrt{2}$ に対応し、以下の残差ネットワークの実験では各ブロックの出力射影に対して $g = 1$、$g = 1/\sqrt{N}$（ブロック数 N の場合）、および $g = 0$ も試す。再利用可能なイニシャライザを 1 つ書いておけば、4 つのケースすべてをすっきりと扱える。

ゲインを引数にとる関数がイニシャライザを返す。NNX のカーネルの形状は $(n_{\text{in}}, n_{\text{out}})$ である。

```
def scaled_normal(gain):
    def init(key, shape, dtype=jnp.float32):
        std = gain / math.sqrt(shape[0])
        return std * jax.random.normal(key, shape, dtype)
    return init
```

5.3.2.2 直接編集

初期化の直後や重みを読み込んだ後に、パラメータの値を直接変更したい場合がある。ReLU は正斉次（positively homogeneous）であるため、先ほどの 10 層スタックで最終層の重みを半分にすると、すべての出力の活性化値も半分になり、したがって出力の二乗平均は 4 分の 1 に減少するはずである。

`kernel[...] *= 0.5` は、追跡対象の `nnx.Param` 変数内の配列を直接書き換える。

```
before = (net(X) ** 2).mean()
net.layers[-2].kernel[...] *= 0.5
print(f'{before:.2g} before, {(net(X) ** 2).mean():.2g} after')
```

```
0.56 before, 0.14 after
```

予想どおり、出力の二乗平均は元の値のちょうど 4 分の 1 に下がる。

5.3.3 残差ネットワークの初期化

5.3.3.1 ブロックをまたぐ分散の増大

先ほど、個々の層がすべて He の規則に従っていても、深い残差スタックは初期化時点で爆発すると述べた。これを示すため、残差ブロックが $\mathbf{x}_{k+1} = \mathbf{x}_k + f_k(\mathbf{x}_k)$ を計算することに着目しよう。ここでは 5.1 節の 2 層の本体を取り上げ、多くの現代的な残差ブロックや Transformer と同様に中間で幅を 4 倍に拡大する構成を考える。すなわち、幅 d から $4d$ への線形層、ReLU、そして $4d$ から d へ戻す線形の**出力射影**である。バイアスはすべて 0 で初期化する。最初の層は直後に ReLU が続くため He 初期化（$g = \sqrt{2}$）を用い、出力射影は標準偏差 $g/\sqrt{4d}$ を用いる。ここでゲイン g は自由パラメータとして残しておく（$g = 1$ が 4.4 節で導出した線形層の標準規則である）。

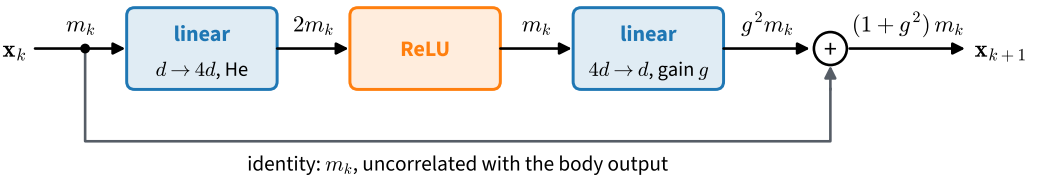


Figure 5.4: 1 つの残差ブロックの各時点における信号の二乗平均。He 層が m_k を 2 倍にし、ReLU が再び半分にし、出力射影が g^2 倍にする。恒等経路は本体の外側で m_k をそのまま伝え、加算点で 2 つの二乗平均が足し合わされる。

m_k を $\mathbf{x}_k$ の各要素の二乗平均としよう。Figure 5.4 は、ブロック内でのその変化を追ったものである。最初の線形層を通過すると m_k は 2 倍になり、ReLU がそれを半分にして m_k に戻す（4.4.2.2 節）。続いて出力射影が二乗平均を $4d \cdot g^2/(4d) = g^2$ 倍にするため、本体の出力 $f_k(\mathbf{x}_k)$ の二乗平均は $g^2 m_k$ となる。ここが重要なポイントである。出力射影の重みは平均 0 で $\mathbf{x}_k$ と独立であるため、本体の出力 $f_k(\mathbf{x}_k)$ はスキップ信号 $\mathbf{x}_k$ と無相関になる。したがって、両者の二乗平均は次のように足し合わされる。

$$m_{k+1} = m_k + g^2 m_k = (1 + g^2)\, m_k. \tag{5.5}$$

どのブロックも、活性化値の二乗平均を $1 + g^2 > 1$ 倍（$g \neq 0$ である限り）に膨張させる。$g = 1$ では各ブロックが分散を 2 倍にするため、N ブロックを経ると活性化値の標準偏差は $2^{N/2}$ 倍に増大する。すなわち、2 ブロック後には $2\times$、8 ブロック後には $16\times$、32 ブロック後には $65{,}536\times$ となる。個々の層はすべて単一層の分散規則を守っていたにもかかわらず、ネットワーク全体が爆発してしまった。その局所的な規則は、ブロックごとにスキップ接続が入力を足し戻すことを考慮していないからである。

5.3.3.2 出力射影の縮小

N ブロック全体では、ブロックあたりの倍率が積み重なって $(1 + g^2)^N$ となる。したがって、任意の深さで活性化値のスケールを適切に保つには、g を N の増加に伴って縮小する必要がある。$g^2 = 1/N$ と選び、すべての本体の出力射影を $1/\sqrt{N}$ 倍にスケーリングすると、この増幅は次のようにきれいに抑えられる。

$$m_N = \left(1 + \frac{1}{N}\right)^N m_0 \leq e\, m_0, \tag{5.6}$$

これは $(1+1/N)^N$ が e に向かって単調増加するためである。これにより、活性化値の標準偏差の増大は深さによらず高々 $\sqrt{e} \approx 1.65$ 倍に抑えられる（2 ブロック後で 1.5、8 ブロック後で 1.60、32 ブロック後で 1.64）。同じ $1/\sqrt{N}$ スケーリングは実用上も広く使われている。たとえば GPT-2 は、すべての残差**ブランチ**の出力射影における初期標準偏差を $1/\sqrt{N}$ 倍（N はネットワーク内の残差ブランチの総数）に縮小する (Radford et al. 2019)。GPT-2 の各ブロックは入力を正規化するため、増大は乗法的ではなく加法的になるが（演習問題を参照）、同じスケーリングによって和が有界に保たれる。Fixup は、正規化層を一切持たない残差ネットワークに対して、この種の深さ依存スケーリングを導出している (Hongyi Zhang et al. 2019)。

5.3.3.3 ゼロ出力射影

この考えを極限まで推し進め、$g = 0$ としたらどうなるだろうか。一見すると、ゼロ初期化は危険に思える。**すべての**重みをゼロにすると、全隠れユニットが永久に同一の対称性に閉じ込められてしまうからである（4.4 節）。しかし、各残差ブロックの本体の出力射影**のみ**をゼロにする場合はまったく異なる。ステップ 0 ではすべての本体が 0 を出力するため、各ブロックは厳密な恒等写像として振る舞い、ネットワーク全体が安定した状態から出発して、訓練中に深さが徐々に働き始める。本体の最初の層はランダムな重みを保持しているため、対称性によって学習が凍りつくことは決してない。最初の逆伝播において、出力射影は（その ReLU 入力が非ゼロであるため）非ゼロの勾配を受け取って即座に 0 から離れ、2 ステップ目以降は最初の層にも勾配が自由に流れるようになる。Figure 5.5 に、この 3 つの段階と全重みゼロ化の罠との対比を示す。

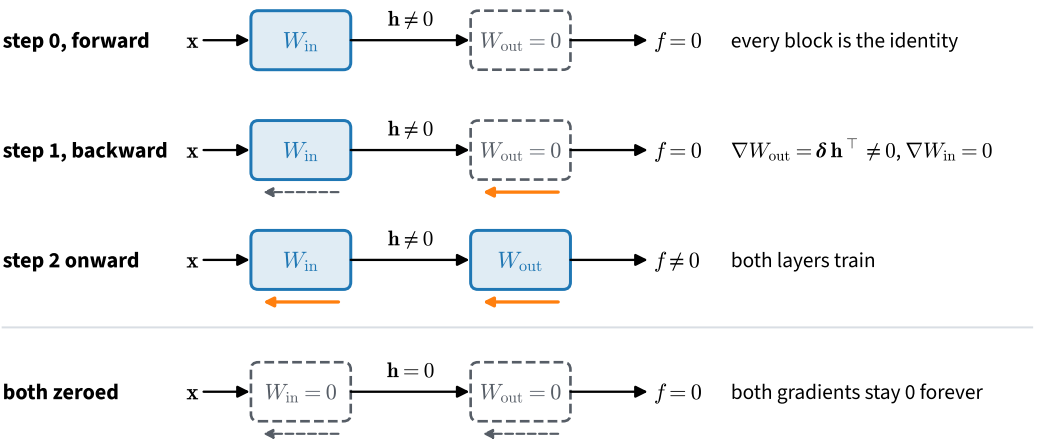


Figure 5.5: 残差ブロックの本体の出力射影 W_{out} のみをゼロ化した場合。最初の順伝播では本体の出力が $f = 0$ となるためブロックは恒等写像となるが、隠れ層の活性化値 $\mathbf{h}$ は非ゼロである。したがって最初の逆伝播では、W_{out} は非ゼロの勾配 $\delta\mathbf{h}^\top$（δ は本体の出力に到達する勾配）を受け取る一方、W_{in} は勾配を受け取らない。次のステップからは両方の層が訓練される。下段：両方の層をゼロ化すると $\mathbf{h} = 0$ となり、どちらの層も永久に勾配を受け取らない。

ゼロ初期化したブランチには明確な系譜がある。Goyal et al. (2017) は、大きなミニバッチで ResNet を訓練するために各残差ブロックの最終スケール係数を 0 に初期化した。Fixup はゼロ出力射影を深さ依存のスケーリングと組み合わせている。その後の研究では、ゼロ行列の代わりに 0 から始まるブランチあたり 1 つの学習可能なスカラーを用いている (De and Smith 2020; Bachlechner et al. 2021)。

5.3.3.4 深さにわたる分散の計測

それでは、3 つの予測すべてを実験で検証しよう。`WideResidualBlock` は、先ほど作成したゲイン付き正規イニシャライザを用いて $d \to 4d \to d$ の残差ブロックを実装したものであり、拡大層にはゲイン $\sqrt{2}$ を、出力射影には設定可能な `out_gain` を用いる。

```
class WideResidualBlock(nnx.Module):
    def __init__(self, d, out_gain, *, rngs):
        self.body = nnx.Sequential(
            nnx.Linear(d, 4 * d, kernel_init=scaled_normal(math.sqrt(2)),
                       rngs=rngs),
            nnx.relu,
            nnx.Linear(4 * d, d, kernel_init=scaled_normal(out_gain),
                       rngs=rngs))

    def __call__(self, X):
        return X + self.body(X)
```

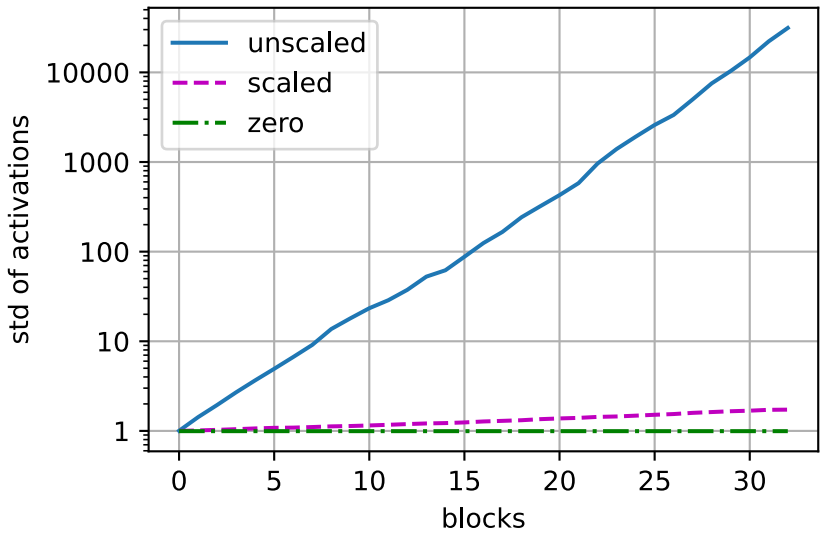
次に、`std_across_depth` は幅 64 のブロックを `num_blocks` 個積み重ねたスタックを構築し、256 個の $\mathcal{N}(0,1)$ 入力を与えて、入力時点および各ブロック通過後の活性化値の標準偏差を記録する。

```
def std_across_depth(num_blocks, out_gain):
    rngs = nnx.Rngs(d2l.get_key())
    net = nnx.Sequential(*[WideResidualBlock(64, out_gain, rngs=rngs)
                           for _ in range(num_blocks)])
    X = jax.random.normal(rngs(), (256, 64))
    stds = [float(X.std())]
    for block in net.layers:
        X = block(X)
        stds.append(float(X.std()))
    return stds
```

スケーリングなし（$g = 1$）、スケーリングあり（$g = 1/\sqrt{N}$）、ゼロ（$g = 0$）の 3 つのゲインスケジュールすべてを比較し、$N = 2, 8, 32$ に対する最終的な標準偏差を表示するとともに、$N = 32$ における深さ全体のプロファイルをプロットする。

```
gains = {'unscaled': lambda n: 1.0, 'scaled': lambda n: n ** -0.5,
         'zero': lambda n: 0.0}
profiles = {n: [std_across_depth(n, gain(n)) for gain in gains.values()]
            for n in (2, 8, 32)}
print(f'{"N":>3}' + ''.join(f'{name:>10}' for name in gains))
for n, stds in profiles.items():
    print(f'{n:>3}' + ''.join(f'{s[-1]:>10.3g}' for s in stds))
d2l.plot(list(range(33)), profiles[32], 'blocks', 'std of activations',
         legend=list(gains), yscale='log', figsize=(4.5, 3))
```

```
  N  unscaled    scaled      zero
  2      2.06      1.48      1.01
  8      18.6      1.59     0.997
 32  3.13e+04      1.73     0.994
```



実測された曲線はこれまでの解析を裏づけている。スケーリングなしのスタックはブロックごとに約 $\sqrt{2}$ 倍で増大し（対数スケール上で直線となり）、32 ブロック後には約 20,000〜200,000 に達する（ランダムな正の係数を掛け合わせると右に裾を引く分布になり、その中央値が平均の平方根を下回るため、個々の試行の多くは理論値 $\sqrt{\mathbb{E}[m_{32}]} = 65{,}536$ よりわずかに下に位置する）。初期の活性化値がすでに数万に達している状態では、スケーリングなしのスタックの訓練を試みる意味はほとんどない。対照的に、$1/\sqrt{N}$ でスケーリングしたスタックはどの深さでも 1.4〜1.9 の間に余裕をもって収まっており、1.5〜1.65 という予測と一致する。一方、出力射影をゼロ化した場合は 32 ブロックすべてにわたって標準偏差が 1 のまま一定に保たれる。

このような簡単な順伝播による診断を実行すれば、訓練に GPU 時間を浪費する前に、破綻した初期化を即座に見つけ出せる。ただし、いざ訓練が始まった後に $1/\sqrt{N}$ スケーリングとゼロ初期化のどちらがより速く最適化されるかは、この診断からはわからないことに注意しよう。どちらもステップ 0 で信号のスケールを保っており、違いは残差ブランチが最初から機能しているか、最初の数回の更新で立ち上がるかという点だけである。

最後に、初期化が制御するのはステップ 0 だけであることを忘れてはならない。勾配ステップによって重みが更新されるにつれ、初期化だけでは活性化値が適切なスケールに保たれる保証はなくなる。そこで出番となるのが正規化層（7.3 節）である。

5.3.4 まとめ

フレームワークのデフォルトのイニシャライザは He 初期化を用いないため、正規化層を持たない深いネットワークでは（コンストラクタ引数、ツリー全体へのコールバック、あるいは直接のインプレース編集による）明示的な初期化が必要であり、さらに大規模モデルでは切断正規分布によってまれに生じる極端な外れ値を防ぐ。とりわけ重要な点として、層ごとの分散規則（4.4 節）だけでは残差アーキテクチャには不十分である。各ブロックの本体を入力に足し戻すことで活性化値の二乗平均が $1 + g^2$ 倍になるため、$g = 1$ ではブロックごとに $\sqrt{2}$ 倍ずつ爆発してしまう。各ブロックの出力射影を $1/\sqrt{N}$ 倍にスケーリングすれば活性化値の増大は $\sqrt{e}$ 倍以下（任意の深さ N に対して）に抑えられ、一方で出力射影をゼロ化すればネットワークを対称性に閉じ込めることなくすべてのブロックを恒等写像として開始できる。

5.3.5 演習問題

1. **切断正規分布**。$\pm 2\sigma$ で切断された正規分布について、切断によって棄却される抽出値の割合はいくらか、また標準偏差はどれだけ縮小するか。いずれの量も初等的な閉じた形を持たない。両者を標準正規分布の密度関数と分布関数を用いて表し、数表の値を使って数値的に評価せよ。さらに、標準偏差を 5.3.1.1 節の実演における出力結果と照合し、通常の正規分布からの抽出値のうち $\pm 2\sigma$ を超える要素を数えることで棄却割合を検証せよ。

2. [code] **正規化された入力**。多くの残差ネットワークは各本体の入力を正規化する。すなわち、本体は入力として $\mathbf{x}_k / r_k$（r_k は各データ例における $\mathbf{x}_k$ の要素の二乗平均平方根）を $\mathbf{x}_k$ の代わりに受け取る。

    1. このとき本体の出力の二乗平均が g^2（$\mathbf{x}_k$ のスケールによらず）になること、および $m_0 = 1$ のとき N ブロック後の二乗平均が $1 + Ng^2$ になることを示せ。深さに伴う増大は、本節の正規化なしブロックと比べてどう異なるか。
    2. スタック末端での二乗平均を 2（すべての N に対して）に保つゲインはいくらか。これを GPT-2 のスケーリングと比較せよ。
    3. `WideResidualBlock` にこの正規化を追加し、$g = 1$ と $g = 1/\sqrt{N}$ について $N = 32$ における深さにわたる標準偏差を計測し、自身の予測および 5.3.3.4 節の実測プロットと比較せよ。

3. [code] **Kaiming 初期化**。4.4.2.2 節の Kaiming（He）イニシャライザ（一様分布版と正規分布版）を、明示的な `nonlinearity` 引数（ゲインを `'relu'` と `'linear'` の間で切り替える）付きで実装せよ。

    1. 入力数 512、出力数 256 の層の重みに対して両者を適用し、平均、標準偏差、値域を確認せよ。すなわち、正規イニシャライザについては本節のゲイン付き正規イニシャライザと比較し、一様イニシャライザについては理論上の境界および標準偏差と比較せよ。値域に違いがあればその理由を説明せよ。
    2. 実装した正規イニシャライザを 5.3.3.4 節のスタックに適用し、すべてのブロックの両方の層に `'relu'` のゲインを設定せよ。各ブロックが二乗平均を何倍にするかを予測し、5.3.3.4 節のプロットに実測曲線を追加して検証せよ。

    CMU 10-714 の Homework 2 および Homework 4 に基づく。

4. [code] **深さにわたる勾配**。●ブロックを通過する逆伝播では、勾配に $\mathbf{x} \mapsto \mathbf{x} + f(\mathbf{x})$ の転置ヤコビ行列が掛けられる。

    1. 本節の議論を逆伝播に対して繰り返せ。逆伝播では各層の入力ではなく出力について和をとるにもかかわらず、本体が勾配の二乗平均を g^2 倍にし、その結果として各ブロックがやはり二乗平均を $1 + g^2$ 倍にすることを示せ。
    2. $N = 32$ における 3 種類のゲインの選択についてこの予測を検証せよ。すなわち、スタックの出力の総和の、その入力に関する勾配を計算し、その二乗平均を出力時点での勾配の二乗平均と比較せよ。

5. [code] **データ駆動の較正**。●5.3.3.4 節のスタックをスケーリングなしのゲインで初期化したうえで、単位分散の入力からなる 1 つのミニバッチを用いてブロックごとに較正せよ。各ブロックについて順番に、そのブロックまでミニバッチを順伝播させ、本体の出力 $f_k(\mathbf{x}_k)$ の標準偏差を計測し、その標準偏差が $1/\sqrt{N}$ になるようブロックの出力射影を再スケーリングする。これは Mishkin and Matas (2016) による層逐次の単位分散初期化を残差スタックに適合させたものである。

    1. 1 回の順伝播から全ブロックを一度に較正するのではなく、ブロックを順番に較正しなければならない理由を説明せよ。
    2. 較正の対象をブロック出力 $\mathbf{x}_k + f_k(\mathbf{x}_k)$ ではなく本体の出力とする理由を説明せよ。

3. 較正後のスタック末端における標準偏差を予測せよ。較正後の深さ対標準偏差の曲線をスケーリングありのゲインの曲線と並べてプロットし、両者の違いを説明せよ。

5.4 カスタム層とカスタム関数

深層学習フレームワークの組み込みライブラリには、線形射影、活性化関数、ドロップアウトなど、これまで誰もが定番として必要としてきた標準的な層が揃っている。しかし研究はリリースサイクルよりも速く進むため、新たに提案された層が公式ライブラリに加わるまでには何年もかかることがある。各活性化値ベクトルの二乗平均平方根（root mean square、RMS）の大きさを単位長に再スケーリングする正規化層（normalization layer）である **RMSNorm** は、まさにその好例である。2019 年に発表されてから (Zhang and Sennrich 2019)、標準の層ライブラリに採用されるまでに数年を要した。それまでの間、RMSNorm を使いたいコードベースはすべて、自前で実装するかサードパーティ製の実装をインポートするしかなかった。自作するカスタム層が一流の組み込み層と同じように振る舞うには、フレームワークにシームレスに統合されなければならない。すなわち、パラメータがオプティマイザに渡り、状態がチェックポイントへシリアライズされ GPU 上へ移動でき、さらに組み込み層と並べてシーケンシャルコンテナやカスタムコンテナに組み込めなければならない。

ときには、順方向の演算そのものではなく、その導関数が障壁になることもある。80 億パラメータのモデル（5.1 節）を限られたメモリに収めるため、実務では重みを 4 ビット整数へ量子化（quantization）し、ストレージ容量をわずか 4 GB（16 ビット浮動小数点の 4 分の 1）に削減することが多い。訓練済みのモデルを量子化するだけであれば勾配は不要である。しかし、順伝播で量子化された重みを用いて**訓練**するには勾配が必要であり、そこでは丸め処理が立ちはだかる。丸め関数の導関数はほとんど至るところで 0 であるため、順伝播の内部で量子化された重みはすべて受け取る勾配が 0 になり、学習が止まってしまうからである。

本節では、この両方の種類の拡張を構築する。フレームワークのあらゆる操作（最適化、チェックポイント保存、デバイス間の移動）はモジュール木に登録された状態をたどるため（5.1 節）、基底モジュールをサブクラス化してテンソルを登録するだけで、カスタム層はこれらの仕組みに自動的に組み込まれる。以下では 4 つの具体例を順に進めていく。状態をもたない層、学習されるパラメータをもつ `RMSNorm`、訓練されない固定バッファを保持するランダム特徴量の層、そして丸め処理に勾配を通すカスタム逆伝播としての**ストレートスルー推定器**（straight-through estimator）である。

```
from flax import serialization
from jax.test_util import check_grads
```

5.4.1 パラメータをもたない層

まずは最も単純なカスタム層、すなわち状態を一切保持しない層から始めよう。`CenteredLayer` は入力テンソルから全体の平均を引く層である。これを構築するには、フレームワークの基底モジュールをサブクラス化し、順伝播だけを実装すればよい。

```
class CenteredLayer(nnx.Module):
    def __call__(self, X):
        return X - X.mean()
```

ベクトル `[1, 2, 3, 4, 5]` の平均は 3 であるため、`CenteredLayer` はこれを -2 から 2 までの範囲にシフトする。

```
layer = CenteredLayer()
layer(jnp.array([1.0, 2, 3, 4, 5]))
```

```
Array([-2., -1.,  0.,  1.,  2.], dtype=float32)
```

CenteredLayer は組み込み層とまったく同じモジュールインターフェースを共有しているため、シーケンシャルコンテナ内で線形層の直後にそのまま配置して出力を中心化できる。結果の平均は 0 になるはずだが、浮動小数点演算において 10^{-8} 程度以下のごく小さな残差が見られるのは通常の丸め誤差であり、バグではない。

NNX はコンストラクタ内で Linear のパラメータを生成するため、入出力の両方の幅をそこで指定する。

```
net = nnx.Sequential(nnx.Linear(8, 128, rngs=nnx.Rngs(d2l.get_key())),
                     CenteredLayer())
Y = net(jax.random.uniform(d2l.get_key(), (4, 8)))
Y.mean()
```

```
Array(1.8626451e-09, dtype=float32)
```

5.4.2 パラメータをもつ層

4.4 節では、層の重みに応じて活性化値のスケールが深さにわたってドリフトしうることを見た。**正規化層**は、各活性化値ベクトルを制御された大きさに再スケーリングし、代わりにそれを調整するための学習されるスケールパラメータをネットワークに与えることで、この感度を解消する。この層の族については 7.3 節以降で詳しく調べるが、ここではその最も簡潔な一例である **RMSNorm** を実装する。

5.4.2.1 RMSNorm

入力ベクトル $\mathbf{x} \in \mathbb{R}^d$ の二乗平均平方根は $\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$ で与えられる。RMSNorm は $\mathbf{x}$ をこの RMS で割り、学習される**ゲイン**ベクトル $\mathbf{g} \in \mathbb{R}^d$ を要素ごとに掛けてスケーリングする。

$$\mathrm{RMSNorm}(\mathbf{x}) = \frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}} \odot \mathbf{g}.$$

小さな定数 ϵ（たとえば 10^{-6}）はゼロ除算を防ぐためのものである。$\mathbf{g}$ を $\mathbf{1}$ に初期化すると、入力のスケールにかかわらず、出力の二乗平均平方根は常に 1 に近くなる。たとえば $\mathbf{x} = (3, 4)$ の場合、二乗平均は $(9+16)/2 = 12.5$、RMS は 3.536 となって出力は $(0.849, 1.131)$ になり、入力を $(300, 400)$ に再スケーリングしてもまったく同じ出力が得られる。幾何学的には（Figure 5.6）、RMSNorm は $\mathbf{x}$ の向きを保ったまま半径 $\sqrt{d}$ の球面上に射影し、その後 $\mathbf{g}$ が各座標軸を引き伸ばす。

RMSNorm は**層正規化**（layer normalization）を簡素化したものである (Ba et al. 2016)。層正規化では再スケーリングに加えて、各例の平均を引き（行ごとの CenteredLayer に相当）、再スケーリング後に学習されるバイアスを加える。Zhang と Sennrich は、平均の中心化を省いても性能が損なわれることはめったになく計算量を削減できることを示した。そのため、Llama 3 8B (Grattafiori et al. 2024)（5.1 節）をはじめとする多くの現代的な言語モデルがモデル全体で RMSNorm を採用している。

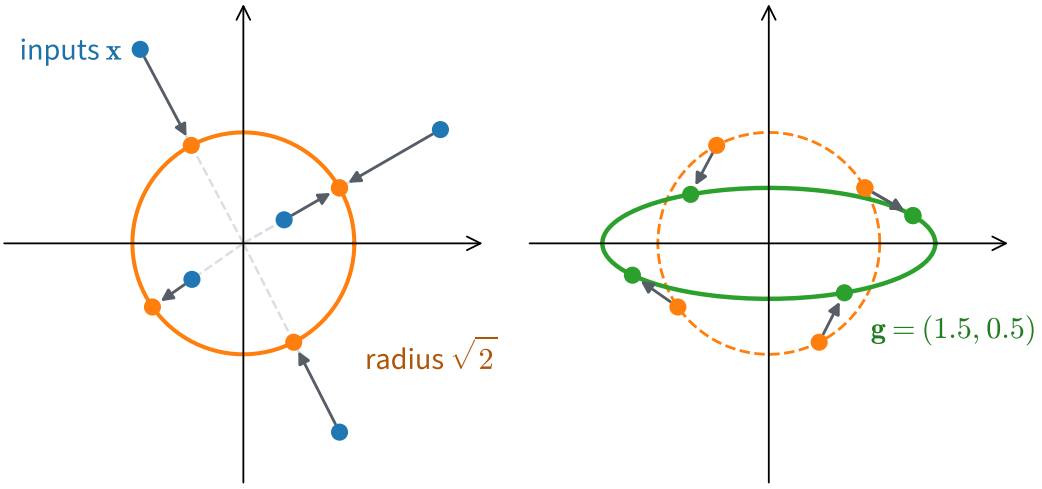


Figure 5.6: 2 次元における RMSNorm。左：ゲインが 1 のとき、すべての入力（青）は原点からの半直線に沿って半径 $\sqrt{2}$ の円周（オレンジ）上へ移り、同じ半直線上にある長さ 0.6 と 2.9 の 2 つの入力は同じ点に着地する。右：ゲイン $\mathbf{g} = (1.5, 0.5)$ は、この円を半軸 $1.5\sqrt{2}$ と $0.5\sqrt{2}$ の楕円へと引き伸ばす。

g は訓練する必要があるため、層はこれをパラメータとして登録する（5.2 節）。実装の残りの部分は定義式をそのまま反映している。

NNX では、`nnx.Param` でラップした配列を属性に代入することでモジュールのパラメータになる。

```
class RMSNorm(nnx.Module):
    def __init__(self, d, eps=1e-6):
        self.gain = nnx.Param(jnp.ones(d))
        self.eps = eps

    def __call__(self, X):
        rms = jnp.sqrt((X ** 2).mean(-1, keepdims=True) + self.eps)
        return self.gain * X / rms
```

`RMSNorm` を $(3, 4)$ と、それを 100 倍に再スケーリングしたベクトル $(300, 400)$ でテストしてみよう。どちらの行も $(0.849, 1.131)$ に写されるはずである。

```
norm = RMSNorm(2)
norm(jnp.array([[3.0, 4.0], [300.0, 400.0]]))
```

```
Array([[0.84852815, 1.1313709 ],
       [0.84852815, 1.1313709 ]], dtype=float32)
```

5.4.2.2 登録

単独のゲインテンソルを受け取る通常の関数を呼び出すのではなく、なぜ RMSNorm をモジュールのサブクラスに包むのだろうか。それは、モジュールへの登録によって、オプティマイザによるパラメータの自動検出、コンテナ内での合成可能性、チェックポイントのシリアライズ、デバ

イス間の移動という組み込み層が備える 4 つの機能すべてがカスタム層に自動的に付与されるからである（5.1 節、5.2 節）。このことをエンドツーエンドで確かめるため、2 つの線形層の間に RMSNorm を挟み、勾配ステップを 1 回進めてゲインを 1 から変化させたうえで、状態全体を新しく初期化したクローンにコピーしてみよう。すると両方のモデルは完全に同一の出力を生成するはずである（ディスク上のチェックポイントへの拡張は 5.6 節で扱う）。

```
rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(nnx.Linear(20, 8, rngs=rngs), RMSNorm(8),
                     nnx.Linear(8, 2, rngs=rngs))
X = jax.random.normal(d2l.get_key(), (4, 20))
optimizer = nnx.Optimizer(net, optax.sgd(0.1), wrt=nnx.Param)
optimizer.update(net, nnx.grad(lambda net: (net(X) ** 2).sum())(net))
raw = serialization.to_bytes(nnx.to_pure_dict(nnx.state(net)))
clone = nnx.Sequential(nnx.Linear(20, 8, rngs=rngs), RMSNorm(8),
                       nnx.Linear(8, 2, rngs=rngs))
state = nnx.state(clone)
nnx.replace_by_pure_dict(
    state, serialization.from_bytes(nnx.to_pure_dict(state), raw))
nnx.update(clone, state)
net.layers[1].gain[...], bool(jnp.array_equal(net(X), clone(X)))
```

```
(Array([0.91795534, 0.6222538 , 0.9029598 , 0.84584   , 0.76988167,
        0.84932446, 0.87185675, 0.9712586 ], dtype=float32),
 True)
```

実際、ゲインは 1 から更新され、クローンは訓練後のモデルとビット単位まで一致する。しかも RMSNorm の内部には、self.gain を登録する以外の定型コードは一切不要である。

5.4.2.3 フレームワーク実装との比較

統合テストに通ったからといって証明されるのは、自作の層がフレームワークに正しく組み込まれていることであり、数式処理にバグがないことではない。ライブラリにネイティブの同等実装が用意されている場合は常に、自明でない重みを使ってカスタム実装をそれと突き合わせることで、数値的な妥当性を即座に確認できる。

自明でないゲイン（linspace(0.5, 1.5, 8)）を自作の RMSNorm とネイティブの nnx.RMSNorm の両方に設定し、出力が一致することを確かめる。

```
ours = RMSNorm(8)
native = nnx.RMSNorm(8, epsilon=1e-6, rngs=nnx.Rngs(d2l.get_key()))
gain = jnp.linspace(0.5, 1.5, 8)
X = jax.random.normal(d2l.get_key(), (16, 8))
ours.gain[...] = gain
native.scale[...] = gain
bool(jnp.allclose(ours(X), native(X)))
```

```
True
```

組み込みライブラリの層が仕様に合致する場合は常に、保守性と融合カーネルによる計算速度の観点から本番環境では組み込み層を優先すべきである。カスタム層を書くのは、ライブラリにない新しい演算や改良が必要なときに限るとよい。

5.4.3 固定状態をもつ層

層によっては、勾配降下法（gradient descent）で学習されるのでも呼び出しのたびにゼロから再計算されるのでもなく、構築時に一度だけ生成されてモデルの存続期間を通じて固定されたままになる配列に依存するものがある。古典的な例が**ランダム特徴量**（random features）層であり (Rahimi and Recht 2007)、ミニバッチ $\mathbf{X}$ を固定された標準正規分布の行列 $\mathbf{W} \in \mathbb{R}^{d \times D}$ で射影し、ランダムな位相 $\mathbf{b} \in \mathbb{R}^D$（$[0, 2\pi]$ から一様にサンプリングしたもの）だけシフトして、要素ごとにコサインを適用する。

$$\mathbf{Z} = \sqrt{2/D}\,\cos(\mathbf{X}\mathbf{W} + \mathbf{b}). \tag{5.7}$$

$\mathbf{Z}$ の各行は D 個の固定された非線形特徴量を与える。驚くべきことに、$\mathbf{W}$ や $\mathbf{b}$ を一切更新しなくても、その上に訓練可能な線形層を 1 つ載せるだけで多様な非線形関数を近似できる。

`Standardizer` の移動統計量（5.2.1.2 節）とまったく同様に、$\mathbf{W}$ と $\mathbf{b}$ は最適化の間は固定されたままでありながら、チェックポイントへの保存やデバイス間の移動ではモデルと行動をともにしなければならない。したがって、これらをバッファとして登録する。

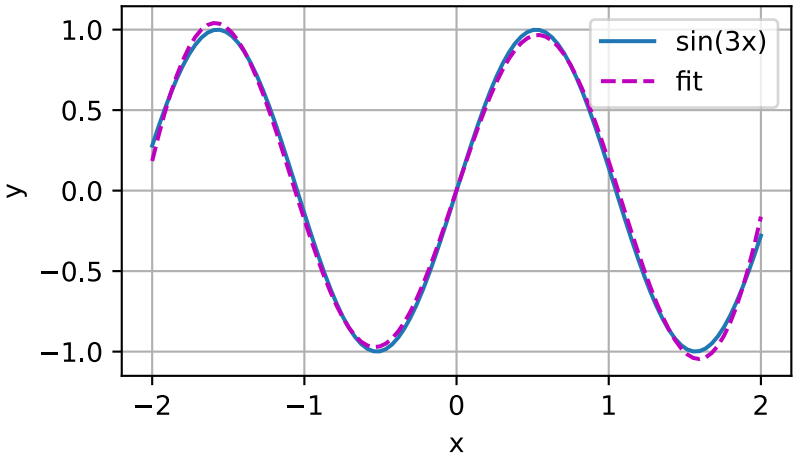
```
class Buffer(nnx.Variable):
    pass

class RandomFeatures(nnx.Module):
    def __init__(self, d, num_features, *, rngs):
        self.W = Buffer(jax.random.normal(rngs.params(), (d, num_features)))
        self.b = Buffer(
            2 * jnp.pi * jax.random.uniform(rngs.params(), (num_features,)))

    def __call__(self, X):
        return jnp.sqrt(2 / self.b.shape[0]) * jnp.cos(X @ self.W + self.b)
```

ここで実際に `RandomFeatures` を動かし、$y = \sin(3x)$ を $[-2, 2]$ 内の等間隔な 100 点にわたって近似してみよう。500 個のランダム特徴量を単一の線形層に入力し、Adam で訓練する。$\mathbf{W}$ と $\mathbf{b}$ はパラメータではなくバッファとして登録されているため、モデル全体をオプティマイザに渡しても線形ヘッドだけが更新される。

```
x = jnp.linspace(-2, 2, 100)
X, y = x.reshape(-1, 1), jnp.sin(3 * x)

def mse(net, X, y):
    return ((net(X)[:, 0] - y) ** 2).mean()

@nnx.jit
def train_step(net, optimizer, X, y):
    optimizer.update(net, nnx.grad(mse)(net, X, y))

rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(RandomFeatures(1, 500, rngs=rngs),
                     nnx.Linear(500, 1, rngs=rngs))
optimizer = nnx.Optimizer(net, optax.adam(0.05), wrt=nnx.Param)
for _ in range(500):
    train_step(net, optimizer, X, y)
```

```
print(f'mean squared error {mse(net, X, y):.2g}')
d2l.plot(x, [y, net(X)[:, 0]], 'x', 'y',
         legend=['sin(3x)', 'fit'], figsize=(4.5, 2.5))
```

```
mean squared error 0.0013
```



学習された線形結合は $\sin(3x)$ をよく捉えており、平均二乗誤差（mean squared error、MSE）は 0.01 未満まで下がる（定数 0 を出力するベースラインでは 0.5 である）。

ここで、チェックポイントに訓練済みの線形ヘッドだけを保存した場合に何が起きるか見てみよう。新しくインスタンス化されたモデルはまったく別の $\mathbf{W}$ と $\mathbf{b}$ を引き直すため、線形層には訓練時に一度も見たことのない特徴量が入力されてしまう。以下では、線形ヘッドだけを復元した場合と、バッファを含む状態全体を復元した場合とを比較する。

```
fresh = nnx.Sequential(RandomFeatures(1, 500, rngs=rngs),
                       nnx.Linear(500, 1, rngs=rngs))
nnx.update(fresh.layers[1], nnx.state(net.layers[1]))
print(f'linear layer only: mean squared error {mse(fresh, X, y):.2g}')
nnx.update(fresh, nnx.state(net))
print(f'full state:        mean squared error {mse(fresh, X, y):.2g}')
```

```
linear layer only: mean squared error 0.7
full state:        mean squared error 0.0013
```

線形層の重みだけを読み込むと MSE は 2 桁以上も跳ね上がり（定数 0 を予測するのと変わらない）、一方でバッファを含む状態全体を復元すれば訓練済みの近似が正確に再現される。

これとまったく同じバッファの仕組みは、バッチ正規化（batch normalization）の移動平均や移動分散（7.3 節）をはじめ、後の章でも訓練対象外のテーブルや移動統計量を保持するために使われる。

5.4.4 カスタム勾配をもつ関数

ここまでは順伝播だけを書き、逆伝播の導出は自動微分（automatic differentiation）に任せてきた。自動微分はあらゆる演算を忠実に微分するが、ときにはまさにその忠実さが邪魔になるこ

とがある。冒頭の 4 ビット量子化の例に戻ろう。順伝播で重みを整数に丸める演算は区分的に定数であるため、真の導関数はほとんど至るところで 0 になり、丸め処理の手前にあるすべての重みの勾配が 0 になってしまう。

```
w = jnp.array([0.9, 1.4, 2.6])
jax.grad(lambda w: jnp.round(w).sum())(w)
```

```
Array([0., 0., 0.], dtype=float32)
```

数学的には勾配が 0 であることは厳密に正しいが、実用上はその場で学習が完全に止まってしまう。**ストレートスルー推定器**（STE）(Bengio, Léonard, et al. 2013) は、順方向と逆方向の計算を切り離すことでこの問題を回避する。順伝播では従来どおり整数への丸めを行う一方、逆伝播では丸めが恒等関数であったかのように見なし、上流からの勾配をそのまま通過させる（導関数を 1 とする）。これは意図的に真の導関数とは**異なる**ものを用いているが、階段関数 $\mathrm{round}(x)$ と恒等関数とのずれは高々 $1/2$ に収まるため（Figure 5.7）、実際にはよく機能する。自動微分は順方向の関数とは異なる代理導関数（surrogate derivative）を勝手に作り出すことはできないため、逆伝播の規則を明示的に与える。

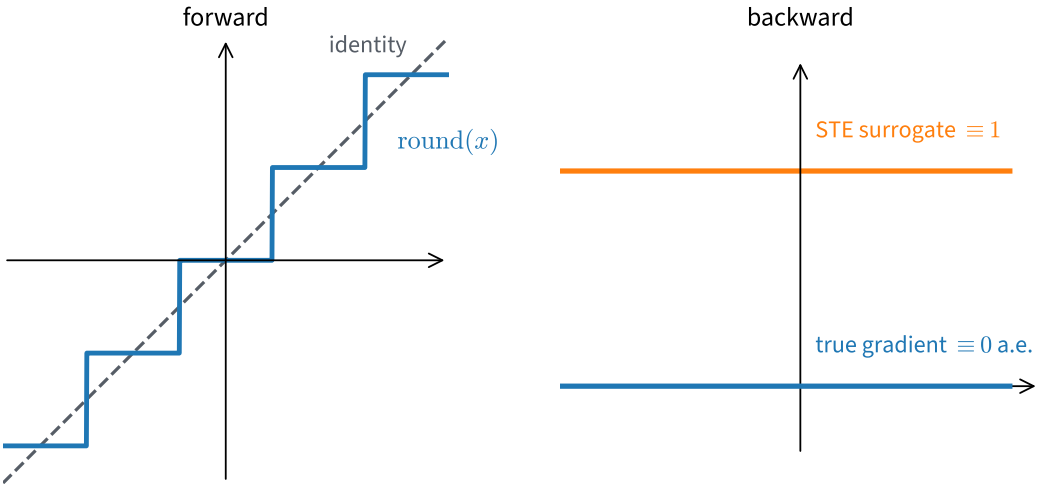


Figure 5.7: ストレートスルー推定器。順伝播：階段関数 round(x) は恒等関数（破線）の近くにとどまる。逆伝播：丸めの真の導関数はほとんど至るところで 0 であるが、この推定器は代わりに恒等関数の導関数である 1 を用いる。

JAX では `jax.custom_vjp` を用い、手書きの逆伝播規則（ベクトル・ヤコビ行列積）を通常の関数に紐づける。

```
@jax.custom_vjp
def round_ste(X):
    return jnp.round(X)

def round_ste_fwd(X):
    return jnp.round(X), None  # (output, residuals); we need no residuals

def round_ste_bwd(residuals, grad_output):
```

```
    # Treat the forward pass as the identity: pass the gradient on
    return (grad_output,)

round_ste.defvjp(round_ste_fwd, round_ste_bwd)
```

round_ste_fwd は (output, residuals)（順伝播の値が不要な場合は None）を返し、round_ste_bwd(residuals, grad_output) は入力引数に対応する 1 要素のタプル (grad_output,) を返す。

丸めた重みに $(1, 2, 3)$ を掛けて和をとることで、このカスタム逆伝播規則を確かめてみよう。ストレートスルー推定器は、$(1, 2, 3)$ をそのまま $\mathbf{w}$ に関する勾配として伝えるはずである。

```
jax.grad(lambda w: (round_ste(w) * jnp.array([1.0, 2.0, 3.0])).sum())(w)
```

```
Array([1., 2., 3.], dtype=float32)
```

順伝播は引き続き整数に丸められた重みで計算される一方、勾配はあたかも丸めが恒等関数であるかのようにそのまま $\mathbf{w}$ へと逆伝播する。勾配降下法は、この代理を通じて実際に学習できるのだろうか。以下では、順伝播中に重みが整数へ丸められる 8 個の重みをもつ線形モデルを、$[-7, 7]$（4 ビット符号付き整数の範囲 $[-8, 7]$ の対称な部分）から一様に引いた重みで生成した合成データ（synthetic data）に適合させる。$\mathbf{w} = \mathbf{0}$ から出発し、通常の丸めとストレートスルー推定器を、初期損失および基準点となる真の重みを丸めたときの損失（各重みを個別に丸めたものが最良の整数近似とは限らない）と並べて比較する。

```
key_X, key_w = jax.random.split(d2l.get_key())
X = jax.random.normal(key_X, (256, 8))
w_true = jax.random.uniform(key_w, (8,), minval=-7, maxval=7)
y = X @ w_true

def fit(rnd, X, y, num_steps=100, lr=0.05):
    loss = lambda w: ((X @ rnd(w) - y) ** 2).mean()
    step = lambda _, w: w - lr * jax.grad(loss)(w)
    return loss(jax.lax.fori_loop(0, num_steps, step, jnp.zeros(8)))

print(f'start {(y ** 2).mean():.3g}, '
      f'rounding {fit(jnp.round, X, y):.3g}, '
      f'straight-through {fit(round_ste, X, y):.3g}, '
      f'rounded target {((X @ jnp.round(w_true) - y) ** 2).mean():.3g}')
```

```
start 191, rounding 191, straight-through 0.772, rounded target 0.793
```

通常の丸めではすべての勾配が 0 になるため、損失は初期値から一歩も動かない。一方、ストレートスルー推定器を用いると、損失は数十〜数百から 1 または 2 程度まで急減し、正解の重みを丸めたときの損失の数倍以内に収まる。意図的に近似された代理勾配（surrogate gradient）によって、量子化モデルの訓練が確かに成功するのである。

恒等関数は数ある代理の 1 つにすぎない。完全な 4 ビット量子化器は入力を $[-8, 7]$ にクランプする処理も含むため、実用的な実装では表現可能な範囲の内側でのみ勾配を通し、飽和した外側では勾配を 0 にすることが多い（演習問題を参照）。適切な代理を選ぶことは、近似対象の演算に合わせたモデリング上の設計判断である。

代理が恒等関数**である**場合には、カスタムの自動微分関数を書く必要を完全になくせる洗練された 1 行のテクニックがあり、本番コードで最もよく目にするのもこの書き方である。順伝播では

恒等関数として振る舞い逆伝播では導関数が 0 になる**勾配停止**（stop-gradient）演算子を用いて $X + \text{stop}(\text{round}(X) - X)$ と書くと、順伝播では $\text{round}(X)$ と評価される一方、逆伝播では先頭の X のみを通じて微分される。クリップ付きの量子化器のように恒等関数でない代理には依然としてカスタム逆伝播の仕組み全体が必要だが、単純なストレートスルー丸めであれば、両方の定義は値も勾配も完全に一致する。

JAX における勾配停止演算は `jax.lax.stop_gradient` である。

```
def round_ste2(X):
    return X + jax.lax.stop_gradient(jnp.round(X) - X)

def loss(f, w):
    return (f(w) * jnp.array([1.0, 2.0, 3.0])).sum()

(bool(jnp.allclose(round_ste(w), round_ste2(w))),
 bool(jnp.allclose(jax.grad(loss, 1)(round_ste, w),
                   jax.grad(loss, 1)(round_ste2, w))))
```

```
(True, True)
```

手書きした逆伝播規則には誤りが紛れ込む可能性があり、順伝播の結果だけではそれに気づけない。標準的なテスト手法は**勾配チェック**（gradient check）である。各入力を微小量だけ摂動させて出力の変化量を測り、この有限差分（finite difference）による傾きと逆伝播規則が返す勾配とを比較する。以下では、滑らかな組み込み関数と先ほどのストレートスルー規則をチェックしてみよう。

```
def check(f):
    try:
        check_grads(f, (jnp.array([0.9, 1.4, 2.6]),), order=1, modes=['rev'])
        return True
    except AssertionError:
        return False

check(jnp.tanh), check(round_ste)
```

```
(True, False)
```

組み込み関数はテストに合格し、ストレートスルー推定器は不合格となる。この不合格は設計どおりの結果である。順伝播は階段関数であり、有限差分で測れる場所では傾きが 0 であるのに対し、ここで定義した逆伝播規則は傾き 1 を返すからである。つまり勾配チェックが教えてくれるのは、ある規則がその順伝播の真の導関数になっているかどうかであり、その規則が有用かどうかではない。演習問題のクランプのように、厳密な導関数であることを意図したすべての規則に対して勾配チェックを実行するとよい。

なお、カスタム勾配は既存のテンソルプリミティブから構築された式の導関数を上書きするものである。カスタム GPU カーネルをゼロから書くことはシステム分野の別領域であり本書の範囲外だが、フレームワークがすでに提供しているプリミティブを使って計算速度を最大化する方法については第 13 章で扱う。

`RMSNorm` のような層については、重要な疑問が 1 つ残っている。その浮動小数点演算は本当に安全なのだろうか。活性化値を二乗して和をとる計算は、16 ビット形式では容易にオーバーフローを起こしうる。5.5 節では、数値形式と混合精度（mixed precision）訓練を直接取り上げて詳しく調べる。

5.4.5 まとめ

カスタム層とは、フレームワークの基底モジュールをサブクラス化して順伝播を実装したものにほかならない。テンソルを登録しておけば、最適化、コンテナへの組み込み、チェックポイント保存、デバイス配置が自動的に処理される。学習される重みは（`RMSNorm` のゲインのように）パラメータとして登録し、訓練されない固定テンソルは（`RandomFeatures` のランダム射影のように）バッファとして登録することで、オプティマイザに公開することなくチェックポイントに保持できる。丸め処理のように演算の真の導関数から有益な情報が得られない場合には、ストレートスルー推定器（あるいはその勾配停止によるショートカット）のようなカスタム逆伝播規則を用いることで訓練を進められる。仕様に合致する組み込みライブラリの層がある場合は常に、保守性と融合カーネルの計算速度の観点から組み込み層を優先すべきである。

5.4.6 演習問題

1. [code] **RMSNorm のスケール不変性**。●$\epsilon = 0$ とする。
   1. $\mathrm{RMSNorm}(c\mathbf{x}) = \mathrm{RMSNorm}(\mathbf{x})$ がすべての $c > 0$ に対して成り立つことを示せ。
   2. 両辺を c に関して $c = 1$ で微分し、RMSNorm のヤコビ行列 $\mathbf{J}$（点 $\mathbf{x}$ におけるもの）が $\mathbf{J}\mathbf{x} = \mathbf{0}$ を満たすことを導け。
   3. 任意の損失の RMSNorm の入力 $\mathbf{x}$ に関する勾配が $\mathbf{x}$ と直交することを導き、ランダムな $\mathbf{x}$ とランダムな線形損失に対して自動微分でこれを確かめよ。微小な勾配ステップは $\mathbf{x}$ の長さにどのような影響を与えるか。

2. [code] **RMSNorm のバイアス**。オプションの学習されるバイアス $\mathbf{b}$ を `RMSNorm` に追加し（ゲインの後に適用するシフトであり、層正規化がもっていた要素の一部を復活させるもの）、これを含むモデルのパラメータ状態に期待どおりの項目が増えることを確かめよ。次に、この層を重み $\mathbf{W}$ とバイアス $\mathbf{c}$ をもつ線形層の直前に配置し、出力を変えることなく $\mathbf{b}$ を $\mathbf{c}$ に吸収できることを示せ。吸収後のバイアスはどうなるか。

3. [code] **ランダム特徴量によるカーネルの近似**。$\mathbf{w}$ の各成分は独立な標準正規分布に従い、b は $[0, 2\pi]$ 上の一様分布に従うとする。
   1. $\cos\alpha\cos\beta = \frac{1}{2}(\cos(\alpha-\beta)+\cos(\alpha+\beta))$ および $\mathbb{E}[\cos(\mathbf{w}^\top\mathbf{u})] = \exp(-\|\mathbf{u}\|^2/2)$ を用いて、`RandomFeatures` の特徴量 $\mathbf{z}(\mathbf{x})$ が $\mathbb{E}[\mathbf{z}(\mathbf{x})^\top\mathbf{z}(\mathbf{x}')] = \exp(-\|\mathbf{x}-\mathbf{x}'\|^2/2)$ を任意の特徴量数 D に対して満たすことを示せ。
   2. 距離が 1 である 2 つの入力に対して、$D = 10$、100、1,000、10,000 としてこれを数値的に確かめよ（各 D について複数の層にわたって誤差を平均せよ）。D が大きくなるにつれて誤差はどのように減少するか。
   3. 本節の $\sin(3x)$ の近似を、特徴量数 $D = 5$ と $D = 50$ の場合、およびランダム特徴量の代わりに x のみを入力とする線形モデルの場合で繰り返せ。それぞれどの程度の誤差に達するか。また、それはなぜか。

   (Rahimi and Recht 2007) に基づく。

4. [code] **カスタム勾配をもつクランプ**。`custom_vjp` 関数を実装し、順伝播を `jnp.clip(X, lo, hi)` とし、逆伝播では入力がクランプ範囲の厳密に内側にあった箇所でのみ勾配を通すようにせよ（順伝播規則では `X` を残差として保存すること）。範囲の内側、外側、およびちょうど境界上にある入力について、その勾配をネイティブの `jnp.clip` の勾配と比較せよ。ネイティブの演算は境界上でどの規約を採用しているか。

   PyTorch チュートリアル Learning PyTorch with Examples および JAX ノートブック Custom derivative rules に基づく。

5.5 数値形式

これまでのモデルでは、ニューラルネットワークに本当にそれだけのビット数が必要かを問うことなく、すべてのテンソルに 32 ビット浮動小数点（floating point、fp32）を使ってきた。ビット幅を半分にすれば、メモリとスループットの両面ですぐに恩恵が得られる。たとえば、本章で一貫して扱う 80 億パラメータのモデルは、重みに 32 ビット精度では 32 GB を要するが 16 ビットなら 16 GB で済み、最新の GPU は 16 ビットの行列積を 32 ビットの少なくとも 2 倍高速に実行する。では、16 ビットで訓練しても同じモデルが得られるだろうか。

単純にはうまくいかない。16 ビット形式は目に見えて粗く、実行時エラーを起こさない 2 つのサイレントな失敗モードを招く。fp16 では 300^2 の時点で `inf` にオーバーフローし、訓練中に `inf` と NaN の値があふれ返る。bf16 では、0.001 を重み 1 に足しても丸められてそのまま 1 に戻ってしまい、勾配が健全に見えていても学習が静かに停滞する。

本節では、これら両方の落とし穴を避けつつ 16 ビットで確実に訓練する方法を示す。すべてのテンソルは **dtype**（数値形式）を持ち、どの形式も**ダイナミックレンジ**（表現可能な数値がどこまで大きく、または小さくなれるか）と**精度**（precision、隣り合う数値がどれだけ細かく並ぶか）のトレードオフの上にある。**混合精度訓練**（mixed-precision training）では、微小な勾配による更新が忠実に累積するよう重みの fp32 マスターコピーを保持する一方で、計算負荷の高い行列積の実行と活性化値（activation）の保存を 16 ビットで行い、さらに fp16 を使う際には小さな勾配が 0 にアンダーフローしないよう**損失スケーリング**（loss scaling）を組み合わせる。5.2 節ではパラメータあたりのバイト数を数えたのに対し、ここでは数値 1 個あたりのバイト数を制御する。2 番目の項（dtype の昇格とモデルのキャスト）は実用上のリファレンスであり、ざっと目を通すだけでよい。すべての例は CPU でも GPU でも動作するが、16 ビットによる実際の高速化には GPU の行列演算ユニットが必要であり、その計測は 5.7 節で行う。

5.5.1 浮動小数点形式

5.5.1.1 範囲と精度

300 を両方の 16 ビット形式で 2 乗してみるだけで、低精度が引き起こす 2 つの異なる問題が浮かび上がる（300 という値はニューラルネットワークの内部では決して珍しいものではなく、正規化前のロジットや分散計算の中間和は容易にこの大きさに達する）。

```
x = jnp.array(300.0)
x.astype(jnp.float16)**2, x.astype(jnp.bfloat16)**2
```

```
(Array(inf, dtype=float16), Array(90112, dtype=bfloat16))
```

fp16（**半精度**、half precision）では、300^2 はいきなり `inf` にオーバーフローする。最大の有限値がわずか 65504 しかないためである。bf16（**brain floating point**）では結果は有限にとどまるものの（90000 ではなく 90112）、すでに有効数字 3 桁目で丸め誤差（rounding error）が生じている。どちらの形式も 16 ビットに収まるが、そのビットの割り当て方が異なるにすぎない。任意の正規化浮動小数点数は、符号ビット s、指数 e、および t ビットの仮数部（小数部 $f \in [0, 1)$ を符号化する）を格納し、次の値を表す。

$$x = (-1)^s \cdot 2^e \cdot (1 + f). \tag{5.8}$$

指数部のビット数はダイナミックレンジ（その形式で表現できる最大および最小の絶対値 $|x|$）を決定し、仮数部のビット数は相対精度を決定する。具体的には、1 から 2 の間では隣り合う数値

の間隔は 2^{-t} であり、2^e から 2^{e+1} の間では 2^{e-t} となって、スケールによらず一様な相対間隔が保たれる。ここで扱う 3 つの形式のビット配分は次のとおりである。

形式	符号	指数部	仮数部
fp32	1	8	23
bf16	1	8	7
fp16	1	5	10

トレードオフは一目瞭然である。bf16 は fp32 の 8 ビットの指数部をすべて維持しつつ仮数部を 7 ビットに切り詰めることで、解像度は粗くなる代わりに fp32 と同じダイナミックレンジを実現している。fp16 はその逆で、仮数部を 10 ビット残す代わりに指数部を 5 ビットに抑えており、局所的な刻み幅は細かくなるものの表現範囲ははるかに狭くなる。先ほどの 2 つの出力は手計算でも確かめられる。指数部が 5 ビットのとき、e は $[-14, 15]$ の範囲をとり、fp16 の上限は $2^{15}(2 - 2^{-10}) = 65504$ となる。仮数部が 7 ビットのとき、2^{16} と 2^{17} の間にある bf16 の値の間隔は $2^9 = 512$ であるため、90000 は $175 \cdot 512 = 89600$ と $176 \cdot 512 = 90112$ の間に位置し、より近い値である 90112 に丸められる。

Figure 5.8 にこれらのビット配分を並べて示す（IEEE 浮動小数点演算のより詳しい解説については 35.8 節を参照）。

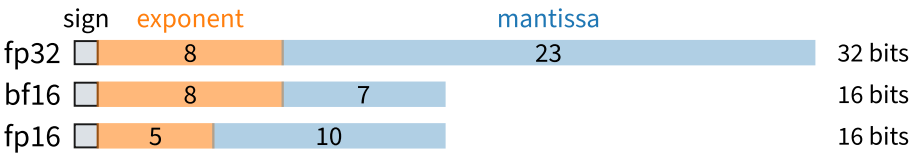


Figure 5.8: 表の各形式のビット配置を符号ビットの位置で揃えたもの。棒が長いほどビット数が多いことを表す。bf16 は fp32 の 8 ビット指数部を維持し、fp16 と bf16 は同じ 16 ビットを正反対のやり方で分割している。

各形式の正確な数値限界は `finfo` で照会でき、ここでは `max`（最大の有限値）、`tiny`（正の最小の**正規化数**、normal number）、`eps`（1 の直上における間隔 2^{-t}）の 3 つのフィールドに注目する。指数が下限に達する `tiny` 未満の領域では、等間隔に並ぶ**非正規化数**（subnormal number）を通じて 0 まで段階的に精度が低下する。fp16 ではこれらは 2^{-24} の倍数となるため、3×10^{-5} 付近の値はまだ 9 ビットの有効ビットを保つのに対し、10^{-7} 付近の値はわずか 1 ビットしか保持できない。

```
for dtype in (jnp.float32, jnp.bfloat16, jnp.float16):
    fi = jnp.finfo(dtype)
    print(f'{fi.dtype.name:10} max {float(fi.max):10.3e}'
          f'  tiny {float(fi.tiny):9.3e}  eps {float(fi.eps):8.3e}')
```

```
float32    max  3.403e+38  tiny 1.175e-38  eps 1.192e-07
bfloat16   max  3.390e+38  tiny 1.175e-38  eps 7.812e-03
float16    max  6.550e+04  tiny 6.104e-05  eps 9.766e-04
```

bf16 は fp32 と同じ 8 ビットの指数部を共有するため、`tiny` は fp32 と一致し、`max` もほぼ一致する一方、`eps` は 0.0078 であり 10 進で 2〜3 桁の精度となる。fp16 は 10 進で 3〜4 桁を分解できるものの、65504 を超えるとオーバーフローし、6.1×10^{-5} 未満では非正規化数に落ち込み、$2^{-24} \approx 6.0 \times 10^{-8}$ 未満の正の値は一切表現できない。深層学習にとって、どちらの不足がよ

り深刻だろうか。活性化値や勾配は日常的に何桁にもわたって変化する。表現範囲を超えると数値が inf や 0 に変わって破滅的な結果を招くのに対し、精度がやや粗い程度であれば穏やかな丸めノイズが加わるだけで済む。最新のハードウェアで bf16 が標準の 16 ビット訓練形式となり、fp16 が主に旧世代の GPU や推論（inference）向けに残されているのはそのためである。

5.5.1.2 小さな更新の丸め

bf16 の粗い間隔が**実際に**悪影響を及ぼすのは、パラメータ更新（parameter update）の場面である。勾配降下法（gradient descent）の 1 ステップは重みを ηg だけ動かすが、その大きさは重みの絶対値の 10^{-3} 以下であることが多い。最近傍丸め（round-to-nearest）のもとでは、局所的な格子間隔の半分より小さい更新ステップは完全に消えてしまう。以下では、各形式で 10^{-3} と 10^{-4} を重み 1 に加算してみる。

```
w = jnp.array(1.0)
for dtype in (jnp.float32, jnp.bfloat16, jnp.float16):
    print(f'{jnp.dtype(dtype).name:8}: {float(w.astype(dtype) + 1e-3):.7f}, '
          f'{float(w.astype(dtype) + 1e-4):.7f}')
```

```
float32 : 1.0010000, 1.0001000
bfloat16: 1.0000000, 1.0000000
float16 : 1.0009766, 1.0000000
```

fp32 では、どちらの増分も問題なく反映される。1 付近での間隔が $2^{-7} \approx 0.0078$ である bf16 では、どちらも丸められて 1 に戻り、重みは完全に凍りついたままとなる。fp16（間隔 $2^{-10} \approx 0.00098$）では、10^{-3} のステップは（格子 1 目盛り分に丸められて）生き残るが、10^{-4} は消えてしまう。Figure 5.9 は両方の格子を同じ縮尺で描いたものであり、更新が反映されるのは重みを格子間隔の半分より大きく動かしたときに限られる。したがって、重みを直接 16 ビットで保持すると、たとえすべての勾配が正確であっても最適化が停滞し得る。訓練においてマスター重み（master weights）を fp32 で保持するのはこのためである。

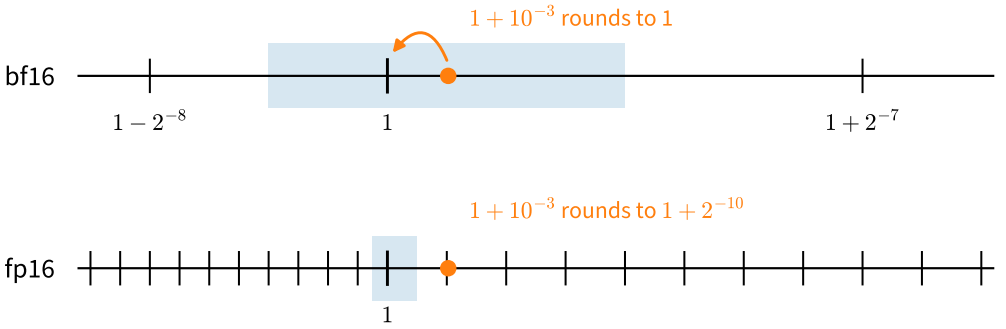


Figure 5.9: 重み 1 付近における bf16 の格子（上）と fp16 の格子（下）を 1 つの縮尺で描いた図。網掛けの区間は丸められて 1 に戻る値の範囲を表し、左右それぞれに格子間隔の半分まで広がっている（1 より下側の間隔は 1 より上側の間隔の半分である）。bf16 では更新後の $1 + 10^{-3}$ がこの区間の内側に収まるため失われる。fp16 では区間の外側に出るため、次の格子点 $1 + 2^{-10}$ に丸められる。

5.5.2 dtype の昇格とモデルのキャスト

5.5.2.1 昇格

型が一致しないオペランドを式の中で組み合わせると、結果の dtype はどうなるだろうか。fp16 テンソルに、fp32 テンソル、Python の float、整数テンソルをそれぞれ加算して確かめてみよう。

```
x16 = jnp.ones(3, dtype=jnp.float16)
x32 = jnp.ones(3, dtype=jnp.float32)
(x16 + x32).dtype, (x16 + 1.0).dtype, (x16 + jnp.arange(3)).dtype
```

```
(dtype('float32'), dtype('float16'), dtype('float16'))
```

2 つの浮動小数点テンソルを組み合わせるとより広い形式へ昇格する（また fp16 + bf16 は fp32 へ昇格する）。そのため、紛れ込んだたった 1 つの fp32 テンソルによって、後続の 16 ビット計算全体が暗黙のうちに fp32 へ昇格してしまう。Python の float リテラルや整数テンソルは、浮動小数点テンソルを昇格させない。

5.5.2.2 モデルのキャスト

パラメータは単なるテンソルであるため、モデル全体を 16 ビットにキャストできる。本節を通して、4.2 節で構築した Fashion-MNIST 用の MLP を使用する。これは 784 個の入力、256 ユニットの ReLU 隠れ層（hidden layer）、10 個の出力を持ち、重みを $\mathcal{N}(0, 0.01^2)$、バイアスを 0 で初期化したものである（パラメータ数は 203,530 個で、fp32 では 814,120 バイトを占める）。

```
def mlp(rngs, dtype=None):
    normal = nnx.initializers.normal(0.01)
    return nnx.Sequential(
        lambda x: x.reshape((x.shape[0], -1)),
        nnx.Linear(784, 256, kernel_init=normal, dtype=dtype, rngs=rngs),
        nnx.relu,
        nnx.Linear(256, 10, kernel_init=normal, dtype=dtype, rngs=rngs))

def param_bytes(model):
    return sum(p.size * p.dtype.itemsize
               for _, p in nnx.state(model, nnx.Param).flat_state())

net = mlp(nnx.Rngs(d2l.get_key()))
param_bytes(net)
```

```
814120
```

Flax NNX の層は保存用の型（`param_dtype`、デフォルトは fp32）と計算用の型（`dtype`）を分離している。`.astype(jnp.bfloat16)` を `nnx.Param` にマップすると、保存されている重みがキャストされる。

```
def cast(model, dtype):
    """Return the model with its parameters stored in dtype."""
    graph, params = nnx.split(model, nnx.Param)
    return nnx.merge(graph, jax.tree.map(lambda p: p.astype(dtype), params))

net16 = cast(net, jnp.bfloat16)
```

```
X = jax.random.normal(d2l.get_key(), (8, 28, 28, 1))
print(net16(X).dtype, param_bytes(net16))
print(net16(X.astype(jnp.bfloat16)).dtype)
```

```
float32 407060
bfloat16
```

キャストによってパラメータのメモリ使用量は即座に半減するが、16 ビットのモデルが fp32 の入力を自動的にキャストするか、dtype の不一致として拒否するかはフレームワークによって異なる。

推論においては、訓練済みモデルを 16 ビットにキャストすれば（タスクの正解率が維持されることを確認した上で）重みのメモリ使用量をただちに半減できる。しかし**訓練**においては、純粋な 16 ビットの重みは先ほど見た小さな更新の丸めという落とし穴に直面する。以下ではこれを実験で確かめる。

5.5.3 混合精度訓練

5.5.3.1 マスター重みと 16 ビット計算

一見するとジレンマに直面しているように思える。信頼できる重み更新には 32 ビットでの保存が求められる一方で、ハードウェアによる高速化は 16 ビット演算にあるからである。**混合精度訓練** (Micikevicius et al. 2018) は、パラメータの保存と計算を切り離すことでこの対立を解消する。fp32 の**マスター重み**を保持して小さな更新がすべて正確に累積するようにしつつ、計算負荷の高い順伝播と逆伝播の行列積および活性化値にはそれを 16 ビットにキャストして使い、損失のような精度に敏感なリダクションは fp32 のまま保つのである。Figure 5.10 に 1 回の訓練ステップの概要を示す。

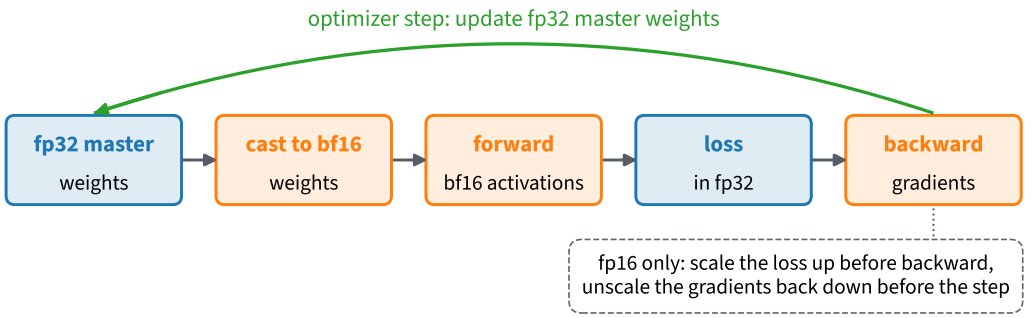


Figure 5.10: 混合精度訓練の 1 ステップ。順伝播のために fp32 のマスター重みが bf16 にキャストされ、bf16 の活性化値が生成される。損失は fp32 で計算され、逆伝播によって勾配が求められ、オプティマイザがそれを fp32 のマスター重みに適用する。fp16 を用いる場合は、逆伝播の前に損失がスケールアップされ、更新の前に勾配が元のスケールに戻される。

NNX の層を `dtype=jnp.bfloat16` で構築すると（`param_dtype` はデフォルトの fp32 のまま残す）、fp32 のマスター重みが保存され、順伝播は bf16 で計算される。

```
net = mlp(nnx.Rngs(d2l.get_key()), dtype=jnp.bfloat16)
net(X).dtype, jax.tree.leaves(nnx.state(net, nnx.Param))[0].dtype
```

```
(dtype(bfloat16), dtype('float32'))
```

5.5.3.2 fp32 と 16 ビットでの訓練

16 ビット演算は最適化にどの程度影響するのか、また fp32 のマスターコピーを省くと何が起きるのだろうか。以下では、256 枚の画像からなる Fashion-MNIST のミニバッチ上で、通常の SGD を用いて MLP を 3 通りで訓練する。すべて fp32、純粋な bf16 重み（fp32 のマスターコピーなし）、および混合精度の 3 つである。3 回の実行はいずれも同一のランダムな初期重みから開始し、まったく同じミニバッチを同じ順序で処理する。

学習率（learning rate）に注目してほしい。ここでは $\eta = 0.01$（4.2 節で用いた値の 10 分の 1）に設定し、1000 ステップ（約 4 エポック）実行する。なぜ η がこれほど大きく影響するかは、先ほどの丸めの解析から説明できる。更新量 ηg は局所的な bf16 の格子間隔の半分を下回ると消えてしまうため、学習率が小さいほど多くの更新が失われるのである。小さな学習率は決して人為的な設定ではない。実際のどの訓練でも、学習率が減衰すればいずれこの領域に到達する。訓練を始める直前に、これら 2 つの学習率のそれぞれでどれだけの更新が失われるかを計測しよう。

```
data = d2l.FashionMNIST(batch_size=256)
images, labels = data.train
batches = [(jnp.asarray(images[k:k+256], jnp.float32) / 255,
            jnp.asarray(labels[k:k+256], jnp.int32))
           for k in range(0, len(images) - 255, 256)]  # One epoch
val_images, val_labels = data.val
val_batch = (jnp.asarray(val_images[:1024, :, :, None], jnp.float32) / 255,
             jnp.asarray(val_labels[:1024], jnp.int32))
```

訓練用のミニバッチと 1,024 枚の画像からなる検証用スライスをあらかじめメモリに読み込んだ上で、`train` は 1,000 ステップの SGD を $\eta = 0.01$ で実行し、ステップごとの損失と最終的な検証正解率（accuracy）を返す。

```
@nnx.jit
def train_step(model, optimizer, X, y):
    def loss_fn(model):
        logits = model(X).astype(jnp.float32)
        return optax.softmax_cross_entropy_with_integer_labels(
            logits, y).mean()
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    optimizer.update(model, grads)
    return loss

def train(net, num_steps=1000):
    optimizer = nnx.Optimizer(net, optax.sgd(0.01), wrt=nnx.Param)
    losses = [train_step(net, optimizer, *batches[step % len(batches)])
              for step in range(num_steps)]
    Xv, yv = val_batch
    accuracy = float((net(Xv).argmax(1) == yv).mean())
    return [float(l) for l in losses], accuracy
```

`train` の中にある 1 つの実装上の工夫は、精度とは無関係にシステム面で良い習慣である。ループ中はステップごとの損失テンソルをデバイス上に置いたままにし、最後にまとめて Python の float へ変換することで、毎ステップの CPU と GPU 間の同期（synchronization）による待ち時間を回避している（5.7 節）。

訓練の前に、最初のミニバッチを使って丸めの議論を確かめておこう。初期化（initialization）時点における第 1 層の非ゼロの更新量 ηg のうち、bf16 で保存した場合に $\eta = 0.01$ と $\eta = 0.1$ でそれぞれどれだけの割合が丸めで消えてしまい、失われた更新が更新量全体の絶対値和のうちどれだけの割合を占めるだろうか。

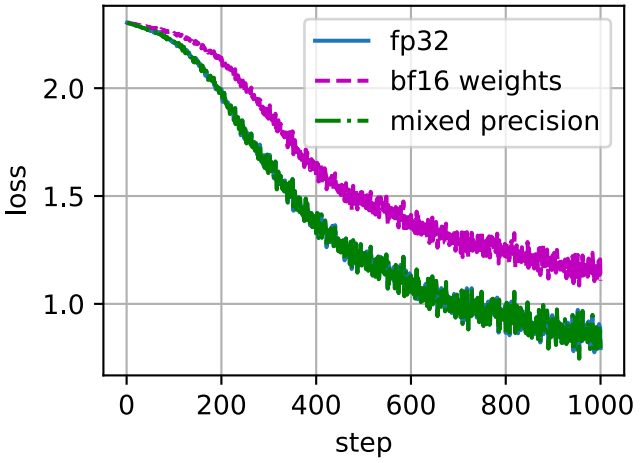
```
net = mlp(nnx.Rngs(d2l.get_key()))
X, y = batches[0]
def loss_fn(model):
    return optax.softmax_cross_entropy_with_integer_labels(
        model(X), y).mean()
g = nnx.grad(loss_fn)(net)['layers'][1]['kernel'][...]
w16 = net.layers[1].kernel[...].astype(jnp.bfloat16)
moved = g != 0  # Count only the nonzero updates
for lr in (0.01, 0.1):
    new = (w16.astype(jnp.float32) - lr * g).astype(jnp.bfloat16)
    lost = (new == w16) & moved
    share = float(lost.sum() / moved.sum())
    mass = float(jnp.abs(g[lost]).sum() / jnp.abs(g).sum())
    print(f'lr {lr}: {share:.0%} of updates rounded away, '
          f'{mass:.0%} of the update mass')
```

```
lr 0.01: 91% of updates rounded away, 77% of the update mass
lr 0.1: 50% of updates rounded away, 14% of the update mass
```

$\eta = 0.01$ では更新の約 10 個中 9 個が丸めで消え、それらは更新量全体の大きさ（出力中の「mass」）の約 5 分の 4 を占めている。つまり、勾配ステップの大部分が重みにまったく届かない。$\eta = 0.1$ でも依然として約半分の更新が失われるが、それらは小さな成分であり、大きさの合計に占める割合は約 15% にすぎない。そのためステップの大半は反映され、学習率が減衰するまで問題は隠れたままになる。それでは、$\eta = 0.01$ ですべての 3 通りを訓練してみよう。

```
key = d2l.get_key()  # One random key, shared by all runs
loss32, acc32 = train(mlp(nnx.Rngs(key)))
loss_bf16, acc_bf16 = train(cast(mlp(nnx.Rngs(key)), jnp.bfloat16),
                            jnp.bfloat16))
loss_mixed, acc_mixed = train(mlp(nnx.Rngs(key)), jnp.bfloat16))
print(f'validation accuracy: fp32 {acc32:.3f}, bf16 weights {acc_bf16:.3f}, '
      f'mixed {acc_mixed:.3f}')
d2l.plot(list(range(1, 1001)), [loss32, loss_bf16, loss_mixed], 'step',
         'loss', legend=['fp32', 'bf16 weights', 'mixed precision'])
```

```
validation accuracy: fp32 0.687, bf16 weights 0.646, mixed 0.688
```



fp32 と混合精度の損失曲線はほぼ完全に重なっており、検証正解率も約 1 パーセントポイント以内で一致する。対照的に、純粋な bf16 重みは最初の 100 ステップのうちに遅れをとり始める。1000 ステップ後の損失は 0.85 に対して約 1.15 にとどまり、検証正解率も 68% に対して約 64% と後れをとっている。スカラーでの実験が予測したとおり、重みを直接 bf16 で保持すると第 1 層の更新の大半が丸めで消えてしまうのに対し、混合精度では fp32 のマスター重みへ正確に適用される。3 回の実行はいずれも収束（convergence）までは訓練していないが、重要なのは同一のデータとステップ数での対等な比較である。

混合精度で何が得られ、何が得られないかについて、2 つの重要な注意点がある。第 1 に、CPU 上の小規模な MLP では実時間の高速化は見られない。16 ビットによる高速化は、大規模な行列積を処理する専用の GPU テンソルコアによってもたらされるものであり、これについては 5.7.2.1 節でベンチマークを行う。第 2 に、混合精度は 5.2 節で見たパラメータあたりの静的なメモリ下限を縮小**しない**。マスター重みとオプティマイザのモーメントが fp32 のまま残るからである。冒頭で述べた 16 GB というメモリ使用量は、**推論**用にキャストした 8B モデルに対する値である。**訓練**中に混合精度が削減するのは、活性化メモリ（activation memory、16 ビットで保存される活性化値が半減する）と行列積の計算時間である。

5.5.3.3 fp16 のための損失スケーリング

bf16 であれば、マスター重みだけで手法として完結する。しかし fp16 では、指数部の範囲が狭いために下限側で 2 つ目の落とし穴が生じる。小さな勾配が**アンダーフロー**するのである。絶対値が $2^{-14} \approx 6.1 \times 10^{-5}$ を下回ると非正規化数に落ちて有効ビットが失われ、最小の非正規化数の半分（$\approx 3 \times 10^{-8}$）を下回る値はすべて 0 に丸められる。10^{-8} の勾配は fp16 では消えてしまうが、これに 2^{16} を掛ければ安全な正規化数の範囲へと引き上げられる。

```
g = jnp.array(1e-8)
g.astype(jnp.float16), (g * 2**16).astype(jnp.float16)
```

```
(Array(0., dtype=float16), Array(0.000655, dtype=float16))
```

実際には、微小な勾配はどれほど頻繁に現れるのだろうか。以下では、最初のミニバッチに対する MLP の第 1 層の fp32 勾配を計算し、非ゼロ要素のうち fp16 の非正規化数または 0 の領域に落ちる割合を、スケーリングなしの場合と 2^{16} 倍にスケーリングした場合の両方で計測する。

```
net = mlp(nnx.Rngs(d2l.get_key()))
X, y = batches[0]
def loss_fn(model):
    return optax.softmax_cross_entropy_with_integer_labels(
        model(X), y).mean()
g = nnx.grad(loss_fn)(net)['layers'][1]['kernel'][...].reshape(-1)
g = g[g != 0]
for scale in (1, 2**16):
    print(f'scale {scale:5}: '
          f'{float((jnp.abs(g * scale) < 2**-14).mean()):.1%} subnormal, '
          f'{float(((g * scale).astype(jnp.float16) == 0).mean()):.2%} zero')
```

```
scale     1: 31.9% subnormal, 0.14% zero
scale 65536: 0.0% subnormal, 0.00% zero
```

第 1 層の非ゼロ勾配のおよそ 3 分の 1 が fp16 の非正規化数の範囲に入り、約 1000 個に 2 個は完全に 0 までアンダーフローしている。この小規模な MLP ではごく微小な要素しか消えないため影響は軽微であり、実際、fp16 が完全に破綻し損失スケーリングによって救われるようなトイモデルを作るのは難しい。深刻なアンダーフローが直撃するのは、損失を多数のトークンやピクセルにわたって平均し、勾配が何十もの層を通過して減衰する大規模で深いモデルである。Micikevicius et al. (2018) は、いくつかの ImageNet 分類モデルはスケーリングなしの fp16 でも問題なく訓練できた一方で、物体検出モデルは発散し、大規模な再帰型言語モデル（language model）は損失スケーリングを有効にして初めて fp32 と同等の正解率に到達したことを報告している。

勾配に 2^{16} を掛けると、fp16 のオーバーフロー上限である 65504 より十分下に収まったまま、非正規化数となる要素がすべて解消される。これこそが**損失スケーリング**の本質である。逆伝播の前に、スカラーの損失に大きな定数 S を掛ける。微分の線形性により、誤差逆伝播法（backpropagation）の過程で生じるすべての中間勾配は S 倍に引き上げられる。そしてオプティマイザのステップを実行する前に、微小な値を忠実に表現できる fp32 上で勾配を S で割り戻す。これにより、数学的なパラメータ更新を一切変えることなく、中間的な fp16 勾配をアンダーフロー領域の外に保つことができる。

S を静的に選ぶ方法は脆い。小さすぎれば勾配がアンダーフローし、大きすぎれば大きな勾配が `inf` にオーバーフローする。**動的損失スケーリング**（dynamic loss scaling）はこの選択を自動化する。大きな S から開始して毎ステップ勾配を検査し、`inf` や NaN が現れたらオプティマイザの更新をスキップして S を半分にする。逆に正常なステップが一定回数続いたら、fp16 の表現範囲の上限付近を維持するよう S を 2 倍にする。Figure 5.11 にこのループの 1 ステップを示す。

Optax には自動損失スケーラが用意されていない（JAX は主に bf16 をネイティブサポートする TPU や最新の GPU を対象としているためである）。fp16 で訓練する場合は、逆伝播の前に損失へ S を掛け、オプティマイザによる更新の前に勾配を S で割る。

したがって、bf16 をネイティブサポートする最新のハードウェア（近年の GPU、TPU、サーバー CPU）では、損失スケーリングなしの bf16 混合精度が標準的な手法となっており、動的損失スケーリングを伴う fp16 は旧世代の GPU において引き続き不可欠である。

最後に、テンソルの dtype を一切変更せずに fp32 モデルを高速化するハードウェア機能を 1 つ紹介しておく。NVIDIA の Ampere アーキテクチャ以降では、fp32 の行列積をテンソルコア上の **tf32** モードで実行できる。入力は fp32 の 8 ビット指数部を保ったまま fp16 と同じ 10 ビット仮数部に切り詰められ、内積は完全な fp32 で累積されるため、厳密な fp32 と有効数字 3〜4 桁まで一致しつつ大幅な高速化が得られる。

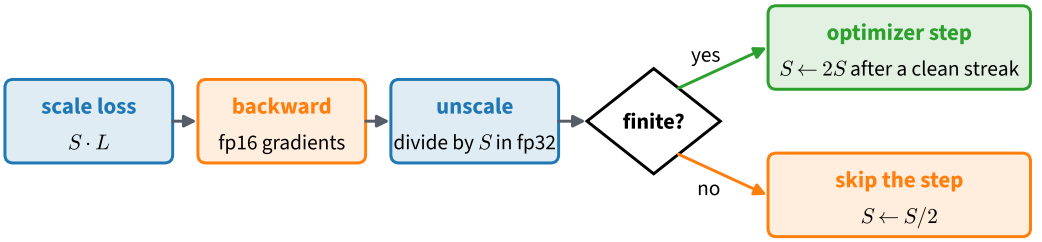


Figure 5.11: 動的損失スケーリングの 1 ステップ。損失に S を掛け、逆伝播を fp16 で実行し、勾配を fp32 で S で割る。すべての勾配が有限であればオプティマイザがステップを進め、そのようなステップが一定回数続くと S を 2 倍にする。いずれかが `inf` または NaN であれば、そのステップをスキップして S を半分にする。

JAX は対応 GPU 上でデフォルトで tf32 を有効にする（`jax.default_matmul_precision('float32')` で制御できる）。

5.5.4 数値計算の失敗モード

16 ビットのコードにおける数値バグの大半は、3 つの具体的な落とし穴で説明できる。上流で起きたオーバーフローが下流で NaN として表面化すること、シフトしていない指数関数が最後の対数で縮められる前に発散すること、そして長いリダクションにおいて個々の項がアキュムレータの丸め閾値を下回った途端に和が増えなくなることである。

5.5.4.1 オーバーフローから NaN へ

損失が NaN になること自体は根本原因ではない場合が多い。たいていは、それより前に起きた `inf` へのオーバーフローの後遺症である。`inf - inf` や `inf / inf` のような不定形の演算が NaN を生み出すためである。

```
s = jnp.array(60000., dtype=jnp.float16) * 2  # Overflows
s, s - s
```

```
(Array(inf, dtype=float16), Array(nan, dtype=float16))
```

最初のオーバーフローは NaN が損失に到達するより何層も上流で起きていることがあり、一度でも NaN を含むパラメータ更新が行われると、それ以降のすべてのステップが汚染される。損失が NaN になったときは、学習率やアーキテクチャを疑う前に、まず値の範囲を確認すべきである（fp16 で実行されている演算が上限の 65504 付近に達していないかなど）。

デバッグ中に `inf` または NaN を生成した最初の演算で即座に停止させるには、`jax.config.update('jax_debug_nans', True)` を有効にする。

5.5.4.2 Log-Sum-Exp

ソフトマックス交差エントロピー（cross-entropy）では $\log \sum_i \exp(x_i)$ を計算する必要がある。この式を文字どおり評価すると、最終的な答えが小さい場合でさえ fp16 ではオーバーフローする。

```
x = jnp.array([12.0, 11.0, 10.0], dtype=jnp.float16)
jnp.log(jnp.exp(x).sum()), jax.scipy.special.logsumexp(x)
```

```
(Array(inf, dtype=float16), Array(12.41, dtype=float16))
```

真の値は 12.4 という穏やかな大きさであるにもかかわらず、中間項 $e^{12} \approx 163{,}000$ が fp16 の上限である 65504 を軽々と超えてしまう。解決策は 3.3.5.2 節で導入した厳密なシフトであり、指数をとる前に最大要素 $m = \max_i x_i$ を引き、あとから m を足し戻せばよい。

$$\log \sum_i \exp(x_i) = m + \log \sum_i \exp(x_i - m). \tag{5.9}$$

右辺の指数はすべて非正となるためオーバーフローは起こり得ず、最大の項は厳密に $e^0 = 1$ となるため $\log 0$ へのアンダーフローも防げる。実際にはこのシフトを自分で実装する必要はほとんどなく、組み込みの log-softmax や交差エントロピーのルーチンが自動的に適用してくれる。

5.5.4.3 fp32 での累積

16 ビットで長い総和を累積すると、気づかないうちに計算が停滞する。途中の累積和が十分に大きくなると、足し込む小さな各項が局所的な格子間隔の半分を下回り、0 に丸められてしまうからである。以下では、（fp16 で**厳密に**表現できる）2^{-10} を 4096 回、fp16 のアキュムレータと fp32 のアキュムレータにそれぞれ加算して比較する。

```
total16, total32 = np.float16(0), np.float32(0)
term = np.float16(2**-10)
for _ in range(4096):
    total16 += term  # fp16 accumulator
    total32 += term  # fp32 accumulator, same fp16 terms
total16, total32
```

```
(np.float16(2.0), np.float32(4.0))
```

厳密な合計は 4 であり、fp32 のアキュムレータはピタリとこの値に一致する。一方、fp16 のアキュムレータは 2048 ステップ後に 2 で凍りつく。2 と 4 の間では隣り合う fp16 の値の間隔が 2^{-9} であるため、$2 + 2^{-10}$ はちょうど中点に落ち、最近傍偶数丸め（round-to-nearest-even）によって永久に 2 へと丸め戻されるからである。バッチ平均、エポック全体の損失合計、正規化層（normalization layer）における分散計算はすべてこの落とし穴を抱えている。混合精度のポリシーがリダクションと正規化を fp32 に保つ理由も、16 ビットテンソルに対するカスタムのリダクションを fp32 で累積すべき理由もここにある。

これら 3 つの失敗モードはすべて同じ根源に行き着く。最終的な答えが余裕をもって収まる場合でも、中間量が 16 ビット形式の範囲や精度を超えてしまうことである。また、混合精度と tf32 は設計上、演算の最下位ビット群を変化させるため、純粋な fp32 とビット単位で一致することは決してない点も覚えておこう。実行結果をいつビット単位で再現**できる**のか、そしてそれをどう実現するかについては 5.8 節で扱う。

5.5.5 まとめ

浮動小数点形式はダイナミックレンジ（指数ビット）と精度（仮数ビット）のトレードオフの上にある。fp16 は局所的な刻み幅が細かいものの表現範囲が狭く（65504 を超えるとオーバーフロ

ーし、微小な勾配はアンダーフローする）、一方で bf16 は精度が 10 進で 2〜3 桁にとどまる代わりに fp32 の 8 ビット指数部をすべて維持する。重みを直接 16 ビットで保存すると小さな勾配更新が丸めで消えて訓練が停滞するため、混合精度訓練では fp32 のマスター重みを保持しつつ行列積の実行と活性化値の保存を 16 ビットで行う。これにより、純粋な bf16 重みが遅れをとる Fashion-MNIST の MLP でも fp32 と同等の正解率を達成できる。fp16 では動的損失スケーリングによって誤差逆伝播法の間、勾配を非正規化数・アンダーフロー領域の外へ引き上げるが、bf16 ではこの手順自体が不要となる。最後に、オーバーフロー起因の NaN から低精度コードを守るには、シフト付きの log-sum-exp 演算を用い、長いリダクションや正規化には fp32 のアキュムレータを使えばよい。

5.5.6 演習問題

1. **混合精度のメモリ**。fp32 で Adam を用いて訓練すると、パラメータあたり 16 バイトが保存される（5.2 節）。`train` が SGD の代わりに Adam を使うと仮定する。
   1. 使用しているフレームワークの混合精度の手法では、ステップ間において重み、勾配、および Adam の移動平均はそれぞれどの dtype で保存されるか。混合精度によってパラメータあたりのバイト数は減少するか。
   2. 各線形層の入力とロジットを保存対象の活性化値として数えるものとする。`train` の MLP について、1 例あたりの要素数を fp32 の場合と `train` が用いる 16 ビット形式の場合のバイト数へそれぞれ換算せよ。どのバッチサイズで、活性化メモリの削減量がこの MLP の永続的な訓練状態のサイズと等しくなるか。
2. **オーバーフロー時の損失スケーリング**。動的損失スケーリング（5.5.3.3 節）のもとでは、スケール解除後の勾配に `inf` または NaN を含むステップはスキップされ、スケールが半分にされる。正常なステップが N 回連続すると、スケールは 2 倍にされる。
   1. スキップされたステップでは、Adam の移動平均を一切変更してはならない。仮に非有限の勾配でそのまま更新した場合に何が問題になるか、また非有限の要素を 0 に置き換えて更新した場合に何が問題になるかをそれぞれ説明せよ。
   2. スケールを縮小するだけでなく、正常なステップが続いた後に再び増大させなければならない理由を説明せよ。
   3. スケールがある固定の閾値を超えると毎ステップ必ずオーバーフローし、それを下回ると一切オーバーフローしないと仮定する。スケールが落ち着いた後、スキップされるステップの割合を N の関数として求めよ。固定閾値の仮定を置かない場合、N を大きくするとどのようなコストが生じるか。
3. [code] **fp16 での正規化**。混合精度の手法では正規化層を fp32 で実行する。その理由を見るため、ゲインを 1 に設定した RMSNorm を考える（5.4 節でモジュールとして実装した）。これは入力行 $\mathbf{x}$（幅 d）をその二乗平均平方根で再スケーリングする。

$$\frac{\mathbf{x}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \epsilon}}, \tag{5.10}$$

   ここで、小さな定数 ϵ は 0 除算を防ぐためのものである。要素の標準偏差が約 100 である幅 $d = 1024$ の行 64 個からなるバッチに対し、この式をすべて fp16 で計算せよ。
   1. どの中間量が最初に `inf` になり、影響を受けた行に対してこの層は何を返すか。
   2. 2 乗の計算自体は厳密であるとする。入力の標準偏差がいくらになるとその平均が fp16 の最大値を超え、またいくらになると d 個の要素にわたる総和が fp16 の最大値を超えるか。

4. [code] **e4m3fn 形式**。●`jnp.finfo(jnp.float8_e4m3fn)` を出力し、その `max`、`tiny`、`eps` を fp16 および bf16 の値と比較せよ。また、`e4m3fn` という名前の意味を説明せよ。4 ビットの指数部と 3 ビットの仮数部を持ち IEEE の規約に従う形式であれば、すべて 1 の指数部を `inf` と NaN のために予約するはずである。その形式における最大の有限値を計算し、`e4m3fn` がどのようにして `max` として 448 を達成しているかを説明せよ。

5.6 保存、読み込み、事前学習済みの重み

訓練済みのニューラルネットワークは驚くほど脆いものであり、それを訓練したプロセスのメモリ上にしか存在しない。スクリプトが正常終了したか、エラーに遭遇したか、サーバーの電源コードが抜けたか、あるいはプリエンプティブルなクラウドインスタンス上で実行されていたかにかかわらず、そのプロセスが終了した瞬間に、学習したすべての重みはディスクに保存していない限り消えてしまう。これまでの章で扱ったトイモデルであれば、途中経過を失っても数分を失うだけで済む。しかし現代の大規模な訓練では数週間分の計算が水泡に帰しかねず、しかもハードウェア障害は珍しい例外ではなく日常茶飯事である。実際、最大 16,384 基の GPU を用いて Llama 3 405B を事前学習（pre-training）した 54 日間のうちに、Meta は予期しない中断を 419 回、すなわち約 3 時間に 1 回の頻度で記録している (Grattafiori et al. 2024)。信頼できるチェックポイント保存（checkpointing）と再開の仕組みがなければ、この規模の実行は決して完走できないだろう。

重みの保存には、もう 1 つ同様に広く浸透した目的がある。それは訓練済みモデルを共有し、他者が一から訓練することなく推論（inference）を実行したりファインチューニングしたりできるようにすることである（32.2 節（英語版））。どちらのワークフローでも、実務上同じ疑問が生じる。ディスクには具体的に何を保存すべきか、どのファイル形式なら安全に読み込めるか、そして重みが対象のアーキテクチャと一致していることをどう検証すべきか。

何を保存すべきかという問いへの答えは、5.2 節から直接導かれる。モデルはコードと状態からなり、コードがアーキテクチャと順伝播（forward pass）を定義する一方、**状態**（state）は訓練によって生成される名前付きテンソル（パラメータとバッファの両方）の集まりである。シリアライズするのは名前をテンソルに対応付けるこの状態のみであり、モデルの再構築はコード上でアーキテクチャをインスタンス化して、そこにテンソルを読み込み直すことで行う。中断された訓練を再開するには、さらに多くの情報が必要になる。モデルの重みに加え、オプティマイザ状態（optimizer state、たとえば Adam の移動モーメントなど）、乱数生成器（random-number generator、RNG）の状態、ステップカウンタ、およびアーキテクチャ設定も保存しなければならず、そのため訓練チェックポイントは推論用の重みファイルよりも数倍大きくなる。

この実践的な節では、まずモデルの状態を確認してシリアライズし、次に危険性を伴う pickle ファイルと安全な `safetensors` 形式を対比する。続いて、ビット単位で完全に再開可能なアトミックな訓練チェックポイントを構築し、最後に事前学習済み ResNet-50 の重みを改変したアーキテクチャへ読み込みつつ、欠落キーと予期しないキーを検証する。

```
import json
import orbax.checkpoint as ocp
import os
import struct
from collections import Counter
from d2l.nnx_resnet import ResNet50
from dataclasses import asdict, dataclass
from flax import traverse_util
from safetensors.flax import load_file, save_file
```

5.6.1 モデルの状態

モデルは数十から数百ものテンソルを含むため、その保存には名前からテンソルへの対応付けを記録する必要がある。各フレームワークはモジュールツリー（module tree）を走査することでこの対応付けを自動生成し（5.1 節）、すべてのテンソルにルートからのパスをキーとして割り当てる。以下では、256 個の隠れユニットを持つ単純な単一隠れ層 MLP の状態を確認する。

`nnx.state` は、モジュールの属性パス（`hidden.kernel`、`output.bias`）をキーとする入れ子状の状態ツリーを返す。

```
class MLP(nnx.Module):
    def __init__(self, rngs):
        self.hidden = nnx.Linear(20, 256, rngs=rngs)
        self.output = nnx.Linear(256, 10, rngs=rngs)

    def __call__(self, x):
        return self.output(nnx.relu(self.hidden(x)))

net = MLP(nnx.Rngs(d2l.get_key()))
X = jax.random.normal(d2l.get_key(), (2, 20))
Y = net(X)
params = nnx.state(net, nnx.Param)
[(path, tuple(value.shape)) for path, value in params.flat_state()]
```

```
[(('hidden', 'bias'), (256,)),
 (('hidden', 'kernel'), (20, 256)),
 (('output', 'bias'), (10,)),
 (('output', 'kernel'), (256, 10))]
```

この状態辞書（state dictionary）には名前、形状、数値のみが記録されており、それ以外の情報は一切含まれない点に注目してほしい。`MLP` クラスや順伝播のロジック、活性化関数（activation function）の痕跡は残っていないのである。したがってモデルを復元するには、完全に一致する名前と形状を再構築する Python コードが必要になる。属性名を `output` から `head` に変更しただけでも、計算する関数は同一であるにもかかわらず対応付けは壊れてしまう。このような不一致を検出し診断する方法は、本節の後半で示す。

5.6.2 Pickle と safetensors

チェックポイントファイルが自身のディスク外に出ない限り、ほぼどのようなシリアライズ形式でも事足りる。しかし、公開ハブから事前学習済みの重みをダウンロードしたり、他者からチェックポイントを受け取ったりした瞬間、セキュリティ上の疑問が生じる。そのファイルを読み込むだけで、手元のマシン上で任意のコードが実行されうるのだろうか。残念ながら、Python のデフォルトのシリアライズ形式では、その答えは「イエス」である。

5.6.2.1 読み込み時のコード実行

Python 標準の `pickle` モジュールは、受動的なデータを保存しているわけではない。Python オブジェクトを再構築するバイトコードプログラムを保存しており、pickle の復元処理はそのプログラムを**実行する**ため、インポート可能な任意の関数を呼び出せてしまう。どのクラスも `__reduce__` を通じて呼び出し可能オブジェクトとその引数を返すことで、pickle が実行する内容をカスタマイズできる。以下の `Tripwire` の例は無害に `print` を呼び出すだけだが、悪意あるチェックポイントであれば同じくらい容易に `os.system` を呼び出せる。

```
class Tripwire:
    def __reduce__(self):  # pickle calls print(...) to rebuild the object
        return (print, ('*** payload executed while loading ***',))
```

NumPy はプリミティブでない Python オブジェクトを保存する際につねに `pickle` を使用する。`allow_pickle=True` を指定して読み込むと、ペイロードが即座に実行される。

```
np.save('tripwire.npy', np.array(Tripwire(), dtype=object))  # pickled
_ = np.load('tripwire.npy', allow_pickle=True)  # the payload runs here
```

```
*** payload executed while loading ***
```

デフォルトでは、各ローダは任意の Python オブジェクトを拒否する。

`np.load` は `allow_pickle=True` を指定しない限り pickle の読み込みを拒否する。

```
try:
    np.load('tripwire.npy')  # allow_pickle=False, the default
except ValueError as e:
    print(type(e).__name__, e)
```

```
ValueError Object arrays cannot be loaded when allow_pickle=False
```

制限付き Unpickler による保護は許可リスト（allowlist）にバグがない範囲でしか機能せず、過去には強化された制限付きローダさえ回避する脆弱性が発見されている。実行可能な命令をそもそもエンコード**できない**ファイル形式であれば、設計上その攻撃対象領域を丸ごと排除できる。

5.6.2.2 safetensors 形式

safetensors 形式は生のテンソルデータとメタデータのみを厳密に格納し、コードは一切含まない。そのバイナリレイアウトは連続した 3 つのセグメントだけで構成されている（Figure 5.12）。すなわち、ヘッダ長を表す 8 バイトのリトルエンディアン整数、各テンソル名をその dtype・形状・バイトオフセットに対応付ける UTF-8 の JSON ヘッダ、そして隙間なく詰め込まれた生のテンソルバッファである。読み込み時にコードは一切実行されず、リーダは JSON ヘッダを検証してすべてのバイト範囲がファイル内に収まっていることを確認するだけである。システム面のさらなる利点として、正確なバイトオフセットがあらかじめわかっているため、メモリマッピング（`mmap`）や、ファイル全体を読み込まずに個々のテンソルへランダムアクセスすることが可能になる。フレームワークに依存せず高速な safetensors は、Hugging Face Hub における標準の重み配布形式となっている。

この MLP の状態を `mlp.safetensors` に保存し、新しく作成したクローンに読み込んで、両方のインスタンスが同一の出力を生成することを確認しよう。

```
def to_flat(tree):
    return {'.'.join(path): value
            for path, value in traverse_util.flatten_dict(tree).items()}

def from_flat(flat):
    return traverse_util.unflatten_dict(
        {tuple(name.split('.')): value for name, value in flat.items()})
```

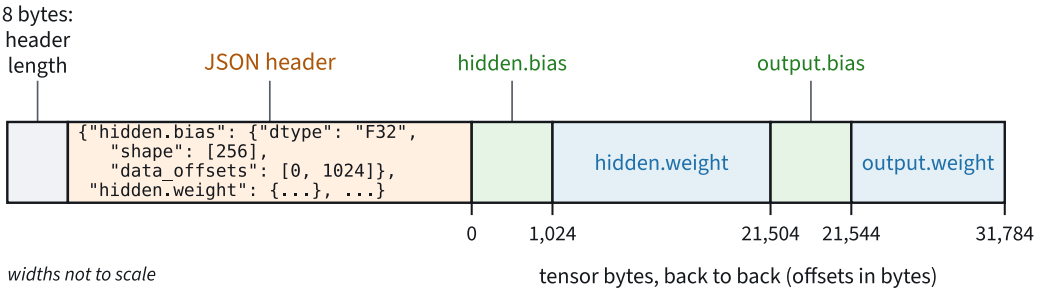


Figure 5.12: 後述する MLP の safetensors ファイルを 1 本のバイト列として示したもの。8 バイトの長さ、各テンソルの dtype・形状・バイト範囲を記した JSON ヘッダ、そして 4 つのテンソルの生バイト列が名前順に隙間なく並ぶ。データの下のオフセットは、ヘッダが記録している範囲を表す。

```
save_file(to_flat(nnx.to_pure_dict(params)), 'mlp.safetensors')
clone = MLP(nnx.Rngs(d2l.get_key()))
clone_params = nnx.state(clone, nnx.Param)
nnx.replace_by_pure_dict(clone_params, from_flat(load_file('mlp.safetensors')))
nnx.update(clone, clone_params)
bool(jnp.array_equal(clone(X), Y))
```

```
True
```

保存と読み込みの往復変換はビット単位で完全に一致する。さらに都合がよいことに、ヘッダは固定オフセットにあるプレーンな JSON であるため、Python の標準ライブラリだけでファイルの中身を調べられる。

```
with open('mlp.safetensors', 'rb') as f:
    n = struct.unpack('<Q', f.read(8))[0]  # header length, little-endian
    header = json.loads(f.read(n))
print('header length:', n, 'bytes')
for name, entry in sorted(header.items(), key=lambda e: e[1]['data_offsets']):
    print(f"{name:18s} {entry['dtype']} {str(entry['shape']):10s} "
          f"bytes {entry['data_offsets']}")
```

```
header length: 296 bytes
hidden.bias        F32 [256]      bytes [0, 1024]
hidden.kernel      F32 [20, 256]  bytes [1024, 21504]
output.bias        F32 [10]       bytes [21504, 21544]
output.kernel      F32 [256, 10]  bytes [21544, 31784]
```

どのバイト区間も、テンソルの要素数に 4 バイト（F32 = float32）を掛けた長さに等しい。256 要素の隠れ層バイアスが最初の 1,024 バイトを占め、$256 \times 20 = 5{,}120$ 要素の隠れ層重みが続く 20,480 バイトを占め、各ブロックは直前のブロックが終わった位置からすぐ始まる。隠れたコードを一切含まない、純粋なテンソルデータである。

手元のマシン上で一時的に使う個人用のチェックポイントであれば、フレームワーク標準の形式でまったく問題ない。しかし、ユーザー向けに公開したりウェブからダウンロードしたりするなど、重みが信頼境界（trust boundary）を越える場合には、safetensors のようなコードを含まな

い形式を用い、受け手が完全性を検証できるよう暗号学的チェックサムを併記すべきである。

5.6.3 訓練チェックポイント

単なる重みファイルは推論には十分だが、訓練の再開には**不十分**である。その理由は、状態を持つ最適化にある。現代の深層学習の実行の多くは **Adam**（9.5 節）を用いており、パラメータごとに 2 つの移動平均を追跡する（5.2 節）。すなわち、$\mathbf{m}$（勾配 $\mathbf{g}$ 用）と $\mathbf{v}$（要素ごとの二乗 $\mathbf{g} \odot \mathbf{g}$ 用）であり、直近の勾配が大きかった箇所では更新幅を抑えつつ、平滑化された勾配と逆方向へ各パラメータを次の式で更新する。

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \, \frac{\hat{\mathbf{m}}}{\sqrt{\hat{\mathbf{v}}} + \epsilon}. \tag{5.11}$$

ここで、η は学習率（learning rate）、ϵ は小さな定数（通常は 10^{-8}）であり、除算と平方根は要素ごとに作用し、ハット記号はゼロ初期化を補うバイアス補正（bias correction）項を表す（導出は 9.5 節を参照）。重要なのは、$\mathbf{m}$ と $\mathbf{v}$ が最適化の履歴を保持している点である。これらを破棄すれば、Adam はあたかも訓練が一から始まったかのように振る舞う。

したがって、真に再開可能な訓練チェックポイントには 5 つの異なる要素が詰め込まれる（Figure 5.13）。すなわち、(1) モデル状態（パラメータとバッファ）、(2) オプティマイザ状態（$\mathbf{m}$ と $\mathbf{v}$）、(3) ミニバッチのシャッフルやドロップアウトを切れ目なく継続するための乱数生成器の状態、(4) Adam のバイアス補正や学習率スケジュール（learning-rate schedule）を制御するステップカウンタ、そして (5) アーキテクチャの再構築に必要なモデル設定である。勾配は次のステップですぐに再計算されるため省略される。結果として、fp32 での Adam 訓練中に保持されるパラメータあたり 16 バイトのうち 12 バイト（重みと 2 つのモーメント）が残る（5.2 節）。そのため、80 億パラメータのモデルに対する再開可能なチェックポイントは 96 GB に達し、これは 32 GB の fp32 重みの 3 倍、16 GB の bf16 推論用ファイルの 6 倍にあたる（5.5 節）。

チェックポイント保存は**クラッシュセーフ**でもなければならない。既存のチェックポイントをその場で上書きすると、書き込み途中でジョブが異常終了した際に新旧両方のバージョンを失う危険がある。そこで代わりに一時ファイルへ書き出し、完了後に保存先へアトミックにリネームすることで、どのリーダから見ても完全な前回のチェックポイントか完全な新しいチェックポイントのいずれか一方だけが見えることを保証する。このリネームによりプロセスのクラッシュからは保護される。電源喪失にまで耐えるにはファイルとそのディレクトリの両方に対する `os.fsync` も必要になるが、これについては演習問題で扱う。

チェックポイントからの再開をテストするため、64 個の隠れユニットを持つ小さな MLP 回帰モデルを Adam（$\eta = 0.05$）により 20 次元のノイズ付き線形データ 256 例で訓練する。その際、シードを 0 に固定した専用の乱数生成器を使って 64 例ずつのミニバッチを抽出し、ミニバッチの順序もチェックポイントの状態に含まれるようにする。5.1 節と同様、`Config` データクラスにアーキテクチャと訓練のすべてのハイパーパラメータをまとめ、チェックポイント内部に直接同梱できるようにする。

```
@dataclass
class Config:
    num_inputs: int = 20
    num_hiddens: int = 64
    num_outputs: int = 1
    batch_size: int = 64
    lr: float = 0.05
```

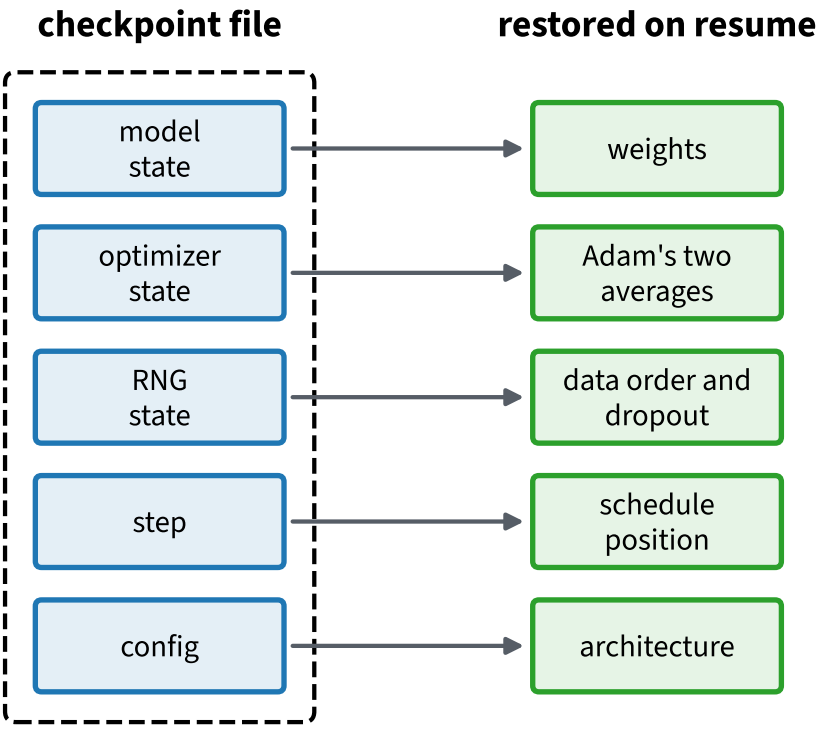


Figure 5.13: チェックポイントファイルの 5 つの区画と、再開時にそれぞれが復元する対象を矢印で対応付けた図。モデル状態は重み、オプティマイザ状態は Adam の 2 つの移動平均、RNG 状態はデータ順序とドロップアウト、ステップはスケジュール内の位置、設定はアーキテクチャを復元する。

まず 200 ステップ訓練してチェックポイントを書き出し、そのまま同じ実行をさらに 5 ステップ続けて、中断なしの 205 ステップ分の正解を作成する。`train` の各呼び出しは、各更新ステップの直前にミニバッチ損失を記録する。

モデル状態ツリー、オプティマイザ状態ツリー、PRNG キー配列、ステップ、設定辞書を 1 つの辞書にまとめ、Orbax（`ocp.StandardCheckpointer`）を介してアトミックに保存する。

```
def build(cfg, rngs):
    xavier = nnx.initializers.xavier_uniform()  # biases start at zero
    return nnx.Sequential(
        nnx.Linear(cfg.num_inputs, cfg.num_hiddens, kernel_init=xavier,
                   rngs=rngs),
        nnx.relu,
        nnx.Linear(cfg.num_hiddens, cfg.num_outputs, kernel_init=xavier,
                   rngs=rngs))

def fresh_state(cfg):
    model = build(cfg, nnx.Rngs(d2l.get_key()))
    return model, nnx.Optimizer(model, optax.adam(cfg.lr), wrt=nnx.Param)

def checkpoint_tree(model, optimizer, key, step, cfg):
    return {'model': nnx.to_pure_dict(nnx.state(model)),
            'optimizer': nnx.to_pure_dict(nnx.state(optimizer)),
            'rng': key, 'step': step, 'cfg': asdict(cfg)}

cfg = Config()
data = jax.random.normal(d2l.get_key(), (256, cfg.num_inputs))
target = (data @ jax.random.normal(d2l.get_key(), (cfg.num_inputs, 1))
          + 0.1 * jax.random.normal(d2l.get_key(), (256, 1)))

def loss(model, X, y):
    return jnp.mean((model(X) - y) ** 2)

@nnx.jit
def step(model, optimizer, idx):
    l, grads = nnx.value_and_grad(loss)(model, data[idx], target[idx])
    optimizer.update(model, grads)
    return l

def train(model, optimizer, key, num_steps):
    losses = []
    for _ in range(num_steps):
        key, subkey = jax.random.split(key)
        idx = jax.random.randint(subkey, (cfg.batch_size,), 0, len(data))
        losses.append(step(model, optimizer, idx))
    return key, [float(l) for l in losses]

net, opt = fresh_state(cfg)
key = jax.random.key(0)                    # draws the minibatches
key, _ = train(net, opt, key, 200)
path = os.path.abspath('run-jax')          # orbax wants an absolute path
ckptr = ocp.StandardCheckpointer()
ckptr.save(path, checkpoint_tree(net, opt, key, 200, cfg), force=True)
```

```
ckptr.wait_until_finished()
key, uninterrupted = train(net, opt, key, 5)
```

ここで、新しいプロセスでの再起動をシミュレートしよう。resume は保存された設定からモデルとオプティマイザを再構築し、重みとあわせてオプティマイザ状態や RNG 状態を任意で復元したうえで、num_steps に達するまで残りのステップを実行する。また、max_diff は 2 つのモデル間における要素ごとの重みの最大誤差を返す。

```
def load_into(obj, saved):
    state = nnx.state(obj)
    nnx.replace_by_pure_dict(state, saved)
    nnx.update(obj, state)

def resume(path, num_steps, restore_optimizer=True, restore_rng=True):
    ckpt = ckptr.restore(path)              # the saved tree, as it is
    model, opt = fresh_state(Config(**ckpt['cfg']))
    load_into(model, ckpt['model'])
    if restore_optimizer:
        load_into(opt, ckpt['optimizer'])
    key = jax.random.key(0)                 # seeded as at the start
    if restore_rng:
        key = ckpt['rng']
    _, losses = train(model, opt, key, num_steps - ckpt['step'])
    return model, losses

def max_diff(a, b):
    return max(float(jnp.abs(p - q).max()) for p, q in
               zip(jax.tree.leaves(nnx.state(a)), jax.tree.leaves(nnx.state(b))))
```

完全なチェックポイントを復元した場合と、乱数生成器だけを省いた場合（restore_rng=False、スクリプトが単にシード 0 を再設定したかのような状態）とを比較し、中断なしで 205 ステップ実行した net に対する重みの最大誤差を測定する。

```
for name, kwargs in (('full checkpoint', {}),
                     ('without the generator', {'restore_rng': False})):
    resumed, _ = resume(path, 205, **kwargs)
    print(f'{name:22s} max |difference| {max_diff(resumed, net):.3g}')
```

```
full checkpoint        max |difference| 0
without the generator  max |difference| 0.0076
```

完全なチェックポイントを復元すると、中断なしの重みが誤差ゼロ（ビット単位の一致）で再現される。一方、RNG 状態を省くと、再開後のループは次のミニバッチを抽出する代わりに最初のミニバッチを再生してしまい、重みに約 0.004〜0.02 の差が生じる（なお、ビット単位の完全一致には決定論的な演算も前提となるが、これについては 5.8 節で詳しく調べる）。

オプティマイザ状態の破棄ははるかに破壊的である。以下では、保存された重みと RNG 状態はそのまま引き継ぎつつ、新しく構築した Adam オプティマイザを使って再開し、再開後 5 ステップのミニバッチ損失を並べて比較する。

```
_, full = resume(path, 205)
_, fresh = resume(path, 205, restore_optimizer=False)
print('saved optimizer:', [round(l, 4) for l in full])
print('fresh optimizer:', [round(l, 4) for l in fresh])
```

```
saved optimizer: [0.0026, 0.0035, 0.0017, 0.003, 0.0022]
fresh optimizer: [0.0026, 4.4987, 0.5426, 1.8454, 2.1844]
```

保存されたオプティマイザ状態を用いれば、訓練は滑らかに再開し、損失は 0.01 前後かそれ以下で推移する。ところが新規の Adam オプティマイザを用いると、**たった 1** ステップで損失が約 1,000 倍に跳ね上がり、4 ステップ後でも再開前の 100 倍以上の水準にとどまってしまう。

なぜ新規の Adam を 1 ステップ実行しただけで、ほぼ収束したモデルが壊れてしまうのだろうか。Equation 5.11 がその理由をすべて物語っている。$\mathbf{m} = \mathbf{0}$ と $\mathbf{v} = \mathbf{0}$ から出発すると、ステップ 1 における Adam のバイアス補正が初期化の減衰因子をちょうど打ち消すため、$\hat{\mathbf{m}} = \mathbf{g}$ および $\hat{\mathbf{v}} = \mathbf{g} \odot \mathbf{g}$ となる。したがって、非ゼロの勾配を持つすべてのパラメータは、勾配がどれほど微小であっても $\eta\,|g_i|/(|g_i| + \epsilon) \approx \eta = 0.05$、すなわち学習率そのものの大きさで移動してしまう。対照的に、保存された $\mathbf{v}$ は訓練序盤のはるかに大きな二乗勾配を記憶しているため、訓練終盤の微小な勾配に対して $\sqrt{\hat{\mathbf{v}}}$ が大きく保たれ、更新は穏やかなまま維持される。モデルが収束（convergence）に近づいているほど、未初期化の Adam による再開の衝撃は激しくなる。重みだけでは、訓練の再開には決して十分ではないのである。

5.6.4 事前学習済みの重み

ここからは、保存された状態のもう 1 つの主要な用途である、他者が訓練した重みからの初期化に目を向けよう。ImageNet（1,000 カテゴリにわたる 120 万枚の画像）で訓練されたバックボーンは、新しいタスクへきれいに転移できる豊かな視覚表現を学習している。これを 10 クラスの分類問題に適応させるのは簡単で、最終の分類器**ヘッド**を除くすべての層を読み込み、新しい 10 出力の線形層に差し替えるだけでよい。以下では ResNet-50（約 2500 万個のパラメータ、1,000 クラスのヘッドを除くと 2350 万個）を使ってこれを実演する。畳み込みネットワークについては第 6 章と第 7 章で学ぶが、重みの読み込みという観点では、ResNet-50 は単に数百個の名前付きテンソルからなる状態辞書にすぎない。

```
net = ResNet50.from_pretrained()                  # ~100 MB once
net.fc = nnx.Linear(2048, 10, rngs=nnx.Rngs(d2l.get_key()))  # new head
net.fc.kernel.shape
```

```
(2048, 10)
```

外部の重みファイルが独自の対象モデルと 1 対 1 で一致することはまれであり、不一致を黙って無視するのは悪名高いバグの温床である。慎重なローダが部分的な転移をどのように監査するかを見るため、事前学習済み ResNet-50 の状態から 2 つのテンソル群を取り除く。1 つは置き換えを**意図して**いる分類ヘッド、もう 1 つは破損または不一致のチェックポイントを模したバックボーンの最終ステージ全体である。残りのテンソルを新規の 10 クラス ResNet-50 に読み込むと、2 つの診断用集合を確認できる。すなわち、**欠落**キー（missing keys、ファイルに存在せずランダム初期化のまま残されるモデルパラメータ）と、**予期しない**キー（unexpected keys、モデル側で使われないファイル内のエントリ）である。

欠落キーと予期しないキーを、モデルのパラメータパスとチェックポイントのキーとの差集合として計算する。

```
def flat_state(model):
    return traverse_util.flatten_dict(nnx.to_pure_dict(nnx.state(model)))

damaged = {path: v for path, v in flat_state(net).items()
           if path[0] != 'fc' and path[:2] != ('stages', 3)}
target = ResNet50(num_classes=10, rngs=nnx.Rngs(d2l.get_key()))
fresh = flat_state(target)
state = nnx.state(target)
nnx.replace_by_pure_dict(state, traverse_util.unflatten_dict(
    {**fresh, **{p: v for p, v in damaged.items() if p in fresh}}))
nnx.update(target, state)
print('missing by block:', dict(Counter(
    f'stages.{p[1]}' if p[0] == 'stages' else p[0]
    for p in sorted(set(fresh) - set(damaged)))))
print('unexpected:', sorted(set(damaged) - set(fresh)))
```

```
missing by block: {'fc': 2, 'stages.3': 50}
unexpected: []
```

新しい 10 クラス用ヘッドの重みとバイアスに対する欠落キーは、そのヘッドがランダム初期化から始まるため想定どおりである。一方、バックボーン最終ステージの欠落キーは危険信号である。もし部分一致のまま漫然と読み込んでいれば、「事前学習済み」ResNet-50 の大部分が人知れずランダムノイズのまま残るところだった。今回の例では予期しないキーの集合は空だが、これが空でない場合は通常、モジュール名の変更によって重みがまったく読み込まれなかったことを示唆する。本番コードでは、欠落キーと予期しないキーの集合が置き換えや削除を意図した層**のみ**と一致することを必ず明示的にアサートし、それ以外が含まれていればエラーを送出すべきである。

検証が済めば、転移学習（transfer learning）では通常、まず事前学習済みバックボーンを**凍結**（freezing）して（5.2.3.2 節）初期の更新で新しいヘッドのみを訓練し、その後より深い層の凍結を選択的に解除していく（32.2 節（英語版））。

本章でここまで扱ってきたものはすべて、ホスト CPU メモリ上に置かれていた。続く 5.7 節では、状態と計算の両方を GPU 上へ移す。

5.6.5 まとめ

モデルの永続化とはその**状態**（モジュールツリーのパスをパラメータやバッファのテンソルに対応付ける辞書）を保存することであり、アーキテクチャ自体はコード内に存在する。Python の `pickle` は読み込み時に任意のコードを実行するため、信頼境界を越えて共有する重みには `safetensors`（8 バイトの長さプレフィックス、JSON ヘッダ、および隙間なく詰め込まれたテンソルバイト列）のようなコードを含まない形式を用いるべきである。クラッシュセーフな訓練チェックポイントは一時ファイルを介してアトミックに書き出され、モデル状態、オプティマイザ状態、RNG 状態、ステップカウンタ、設定という 5 つの要素を保持する。5 つすべてを復元すれば訓練はビット単位で再開されるが、RNG 状態を省くとデータストリームがドリフトし、Adam のモーメントを省くと再開後の最初のステップですべてのパラメータが学習率そのものの大きさで移動して損失が何桁も急騰する。事前学習済みの重みを改変したアーキテクチャへ読み込む際は、不一致を黙って無視するのではなく、欠落キーと予期しないキーの集合を自身の想定と照らし合わせて必ず監査すべきである。

5.6.6 演習問題

1. [code] **Safetensors ヘッダ**。MLP の safetensors ファイルにおけるヘッダ長は、ファイル先頭にある 8 バイトのリトルエンディアン整数である（Figure 5.12）。
   1. 隠れ層の幅を 2 倍にしたとき、および 3 番目の層を追加したときに、その長さがどう変化するかを予測したうえで実測せよ。ヘッダのどの部分がテンソルのサイズに依存するかを説明せよ。
   2. safetensors ライブラリを使わず、ヘッダとファイルのシーク操作のみを用いてファイルから隠れ層のバイアスを読み取れ。ヘッダに記録された dtype と形状を使ってバイト列をデコードし、その結果が `load_file` の返すテンソルと一致することを確かめよ。
   3. pickle ベースのチェックポイントでは、オブジェクトグラフ全体をデシリアライズせずに同様の単一テンソル読み取りを行えない理由を説明せよ。
2. [code] **部分的な読み込み**。`MLP2` を、`MLP` の出力層の直前に幅 256 の 2 番目の隠れ層 `hidden2` を挿入し、属性 `output` を `head` に改名したコピーとして定義せよ。
   1. 破損した ResNet-50 の状態を読み込んだときと同様のキー集合のマージを用いて、`mlp.safetensors` を `MLP2` に読み込め。欠落キーと予期しないキーの集合を報告し、各エントリについて説明せよ。
   2. マージする前に、ファイルのキーを `MLP2` が使う名前に変更せよ。`hidden2` のみが新規の初期化を保っていることを確認し、予期しないキーや `hidden2` 以外の欠落キーがあればマージ時に例外を送出するようにせよ。

   Matthew Inkawhich「Saving and Loading Models」内の「Warmstarting Model Using Parameters from a Different Model」を改変。
3. [code] **dtype の往復変換**。●`float16` の数値は指数部に 5 ビット、仮数部に 10 ビットを格納するのに対し、`float32` ではそれぞれ 8 ビットと 23 ビットを格納する（5.5 節）。`MLP` のパラメータを `float16` にキャストして保存し、それを `float32` の `MLP` に読み込み直せ。
   1. 仮数部のビット幅から、復元されたパラメータの最大相対誤差を予測せよ。復元後の値と元の値のあいだで、最大絶対誤差、および非ゼロのパラメータ全体にわたる最大相対誤差を測定せよ。相対誤差が予測を上回るパラメータがあれば、その理由を説明せよ。
   2. 元のモデルと復元された `MLP` の `X` に対する出力を比較し、この往復変換が推論において許容できるかどうかを判断せよ。
   3. Adam は二乗勾配の移動平均を保持する。`float16` が勾配要素 10^{-4} の二乗をどのような値として格納するかを計算し、再開可能なチェックポイントがオプティマイザ状態を `float32` で保持する理由を説明せよ。
4. [extended] **クラッシュセーフなマニフェスト**。●
   1. 新しいステップ番号のプレフィックスのもとでモデル状態、オプティマイザ状態、JSON メタデータファイルを書き出し、各ファイルの名前・サイズ・チェックサムを記録した 1 つのマニフェストを所定の位置へリネームすることでこれら 3 つを公開するユーティリティを作成せよ。
   2. 2 種類の障害をシミュレートせよ。1 つ目は 3 つのファイルを書き出した後、マニフェストをリネームする前のクラッシュであり、2 つ目は公開済みチェックポイントに含まれる 1 ファイルの切り詰め（破損）である。再読み込みを行うと、1 つ目のケースでは前回の完全なチェックポイントから再開し、2 つ目のケースでは破損したチェックポイントを拒否することを示せ。
   3. リネームはアトミックであるため、リーダからは古いマニフェストか新しいマニフェ

ストのいずれか一方だけが見える。しかし POSIX は、リネームされたファイルのデータがリネーム自体より先にディスクへ到達することを保証していない。`os.fsync` は、ファイルのバッファリングされた書き込みやディレクトリのエントリを強制的にディスクへ書き出す。電源喪失が起きても前回のチェックポイントか新しいチェックポイントのいずれかが無傷で残るよう、このユーティリティのどこで `os.fsync` を呼び出すべきかを述べ、各呼び出しの役割を説明せよ。

5.7 GPU、デバイス、メモリ

2.4 節以降、`Trainer` は `num_gpus` 引数を受け取りつつも、それを黙って無視してきた。これまでの章で扱った小さなモデルでは、CPU での訓練が数秒で終わるため、それでもほとんど問題にならなかった。しかし、モデルが大規模になると、デバイス配置（device placement）が数秒で済むか数時間かかるかの違いを生む。現代のデスクトップ GPU は 2 つの 4096×4096 行列の積を 2 ms 未満で計算するが、CPU では 0.1 秒以上かかるからである（5.7.2.1 節でベンチマークする）。この圧倒的な計算速度には、厳しいハードウェア上の制約が伴う。GPU は専用のメモリ（たとえばデスクトップ向けカードなら 24 GB）を備えており、GPU が演算するすべてのテンソルはそのメモリ上に置かれていなければならない。

そもそも目的のモデルはメモリに収まるのだろうか。ここまで例に挙げてきた 80 億パラメータのモデルは、bf16 の重みに 16 GB を必要とするため、推論であれば余裕をもって収まる。しかし、このモデルを fp32 の Adam で訓練するには、活性化値（activation）を 1 つ保存する前でも 128 GB の静的な状態が必要となり（5.2 節）、単一の 24 GB や 80 GB のカードの容量をはるかに超えてしまう。モデルがデバイスに収まるかどうか、そして GPU が休みなく計算を続けられるかデータ待ちで停止してしまうかは、すべてのテンソルがどこに存在し、何バイトを消費するかに帰着する。

本節では、3 つの中心的な問いに答える。すなわち、テンソルとモデルをどのように CPU や GPU 上に配置し、デバイス間でコピーするか、GPU の非同期実行（asynchronous execution）はどのように動作するか（また GPU カーネルを正確にベンチマークするにはどうすればよいか）、そして訓練中に実際何が GPU メモリを占めるのか（余分な計算と引き換えにメモリ使用量を劇的に削減する**活性化チェックポイント**（activation checkpointing）の仕組みを含む）である。そのうえで、これらすべてを本書の残りの章で用いるデバイス対応の `Trainer` に組み込む。5.2 節と 5.5 節ではパラメータあたり・数値あたりのバイト数を数えたが、本節ではそれらのバイトを実際のハードウェア上に載せる（マルチ GPU 訓練は第 13 章に譲る）。すべてのコードセルは、GPU が利用できない環境でも CPU へスムーズにフォールバックし、実行時間の計測には GPU が必要であっても動作の正しさは確認できるようになっている。

```
import time
```

5.7.1 デバイスと配置

5.7.1.1 デバイスオブジェクト

すべてのテンソルは特定の**デバイス**上に存在する。すなわち、ホスト RAM と対になった CPU か、オンボードメモリと対になった個々の GPU のいずれかである。ターミナルで `nvidia-smi` を実行すると、マシン上のすべての NVIDIA GPU、その総メモリ量と使用中メモリ量、および実行中のプロセスが一覧表示される。デバイスの命名法はフレームワークごとに多少異なるものの、4 つのフレームワークはすべて 1 つの基本原則に従っている。それは、演算は単一のデバイス上で実行されるため、そのオペランドはすべて同じデバイス上に存在しなければならないという原則である。

`jax.devices('cpu')[0]` と `jax.devices('gpu')[i]` は、ハードウェアデバイスのハンドルを返す。

デバイスの取得処理を、本書の以降の章でインポートする 3 つの小さなヘルパ関数にまとめておこう。CPU 用のもの、GPU i 用のもの、そして利用可能な GPU の数を数えるものの 3 つである。

```
def cpu():
    """Get the CPU device."""
    return jax.devices('cpu')[0]

def gpu(i=0):
    """Get a GPU device."""
    return jax.devices('gpu')[i]

def num_gpus():
    """Get the number of available GPUs."""
    try:
        return jax.device_count('gpu')
    except:
        return 0  # No GPU backend found
```

`jax.devices('gpu')` は実機のハードウェアを即座に照会し、GPU が存在しない場合はエラーを送出する。

`num_gpus` を土台として、GPU があればそれを選択し、なければ CPU にフォールバックする 2 つのヘルパ関数を定義する。これにより、ノート PC でも GPU サーバーでも同じノートブックのコードを一切変更せずに実行できる。

```
def try_gpu(i=0):
    """Return gpu(i) if exists, otherwise return cpu()."""
    if num_gpus() >= i + 1:
        return gpu(i)
    return cpu()

def try_all_gpus():
    """Return all available GPUs, or [cpu(),] if no GPU exists."""
    devices = [gpu(i) for i in range(num_gpus())]
    return devices if devices else [cpu()]

try_gpu(), try_gpu(10), try_all_gpus()
```

```
(CudaDevice(id=0), CpuDevice(id=0), [CudaDevice(id=0), CudaDevice(id=1)])
```

Apple シリコンの GPU（`mps`）や Google TPU などの他のアクセラレータも独自のデバイス識別子を用いるが、以下で述べる配置の原則はそれらすべてに等しく当てはまる。本書全体を通して、CPU と CUDA GPU に焦点を当てる。

5.7.1.2 デバイス上の配列

すべてのテンソルは、`device` 属性を通じて自身の配置先を公開している。

JAX の配列は、GPU が利用可能であればデフォルトで最初の GPU に配置される（または `jax.device_put` で明示的に配置できる）。

目的のデバイス上に直接メモリを確保すれば、CPU から GPU への余分なコピーを避けられる。以下では、X を 1 番目の GPU に、Y を 2 番目の GPU に配置し（GPU が 2 基未満の場合は CPU にフォールバックする）、両者の加算を試みる。

```
X = jax.device_put(jnp.ones((2, 3)), try_gpu())
Y = jax.device_put(jax.random.uniform(d2l.get_key(), (2, 3)), try_gpu(1))
try:
    print(X + Y)
except ValueError as e:
    print(str(e).split('. ')[0])  # The first sentence names the problem
```

```
Received incompatible devices for jitted computation
```

異なるデバイス上のテンソル同士を加算しようとすると即座にエラーが送出され、意図しないデバイス間転送を防ぐ。

5.7.1.3 デバイス間のコピー

デバイス間でテンソルを移動する処理は、デバイス上での計算よりはるかに遅いため（この差は 5.7.2.1 節で計測する）、異なるデバイス上のオペランドを暗黙的にコピーしてしまうと深刻な速度低下のバグが隠れてしまう。そこで、X を Y のデバイス上へ明示的にコピーしてから（Figure 5.14）加算する。

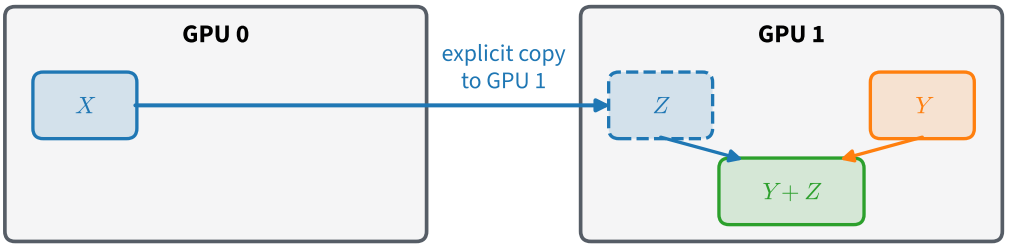


Figure 5.14: X は GPU 0 上に、Y は GPU 1 上に存在する。GPU 1 上に X のコピー Z（破線）を作ることで、Y + Z をすべて GPU 1 上で実行できる。

```
Z = jax.device_put(X, try_gpu(1))
(Y + Z).device
```

```
CudaDevice(id=1)
```

これで加算が実行され、結果も両オペランドと同じ 2 番目の GPU 上に置かれる。実践上の原則は単純である。計算の境界で一度だけデータを転送し（すでにそのデバイス上にあるテンソルの移動は実質的に何もしない操作である）、内側ループの演算はすべてその単一のデバイス上に留めることである。

5.7.1.4 デバイス上のモデル

順伝播（forward pass）では入力とモデルパラメータを組み合わせるため、モデル全体がミニバッチと同じデバイス上に存在しなければならない。5.1 節で見たモジュールツリーのおかげで、どのフレームワークでも 1 回の呼び出しですべてのパラメータを移動または初期化できる。

```
net = nnx.Sequential(nnx.Linear(3, 1, rngs=nnx.Rngs(d2l.get_key())))
nnx.update(net, jax.device_put(nnx.state(net), try_gpu()))
net(X)
```

```
Array([[0.8409575],
       [0.8409575]], dtype=float32)
```

入力とパラメータが同じデバイス上に揃っていれば、順伝播はそのデバイス上で実行され、出力も同じデバイス上に返される。

配置に関する 2 つの原則を心に留めておこう。1 つ目は、オプティマイザを構築する**前**にモデルを目的のデバイス上へ移動または初期化すること（これによりオプティマイザのモーメントも同じデバイス上に確保される）、2 つ目は、`forward` 内で新たに生成するテンソルをデフォルトの CPU ではなく `X.device` 上に確保することである。

5.7.2 非同期実行

5.7.2.1 実行キュー

GPU は、Python から演算が呼び出された瞬間にそれを同期的に実行するわけではない。その代わり、Python は各カーネル起動（kernel launch）をデバイスのコマンドキューに追加して即座に制御を返し、GPU がキュー内の処理を順番にこなしている間に CPU が後続の処理を準備できるようにする。このパイプライン化によって Python のオーバーヘッドは隠蔽されるが、時間計測においては重要な落とし穴がある。Python ループの所要時間を測っても、計測されるのは**実行**ではなく**キューへの投入**にかかる時間なのである。以下では、4096×4096 行列の積を 32 回キューに投入し、実時間（ウォールクロック時間）を 2 回確認する。Python がキューへの投入を終えた直後と、デバイスが計算を終えた直後の 2 回である。

```
A = jax.random.normal(d2l.get_key(), (4096, 4096))
(A @ A).block_until_ready()  # Warm up and wait
t = time.perf_counter()
for _ in range(32):
    B = A @ A
print(f'time to queue 32 matrix products: {time.perf_counter() - t:.4f} sec')
B.block_until_ready()  # The device runs the products in order
print(f'time until they all finished:     {time.perf_counter() - t:.4f} sec')
```

```
time to queue 32 matrix products: 0.0020 sec
time until they all finished:     0.0517 sec
```

JAX は CPU と GPU の両方で非同期にディスパッチする。`B.block_until_ready()` は計算が完了するまでブロックする。

Python は、実際の計算時間のほんの一瞬のうちに 32 個すべての行列積をキューに投入し終える。では、CPU が実際にブロックして待機するのはいつだろうか。それは、テンソルの表示、スカラーから Python の浮動小数点数への変換、NumPy への変換、あるいは `if` 文でのテンソル値に基づく分岐など、Python がホスト側で具体的な値を必要とするときだけである。このような**同期ポイント**（synchronization point）のたびに、GPU がその値までのキューを消化し終えるまで Python は停止し、その後 Python が追いつくまでの間、今度は GPU がアイドル状態になる（Figure 5.15）。ミニバッチごとに損失を Python の数値として記録すると、毎ステップ同期の空白時間（バブル）が生じてしまう。`Trainer` が `d2l.Averager` 内で損失をデバイス上に保持し、エポックごとに 1 回だけ Python の浮動小数点数に変換しているのはこのためである（2.4 節）。

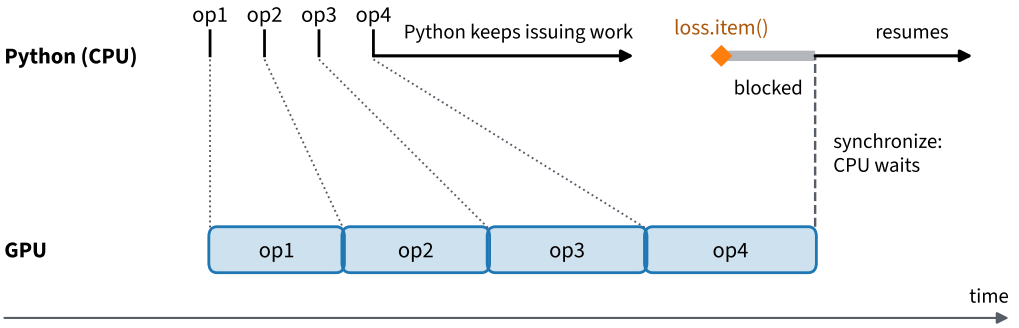


Figure 5.15: Python は演算 op1 から op4 までをキューに投入して先へ進み、その間に GPU がそれらを順番に実行する。損失を Python の数値として読み出す処理は同期ポイントとなる。GPU がキューを消化し終えるまで CPU はブロックし、その後 Python が再開する間、GPU は次の処理を受け取るまでアイドル状態になる。

非同期ディスパッチの仕組みから、信頼できる GPU ベンチマークのための 4 つの原則も導かれる。(1) 初回限りのメモリ確保、ライブラリの初期化、JIT コンパイルを除外するため、最初にウォームアップ呼び出しを実行すること、(2) タイマの開始**直前**と停止**直前**の両方でデバイスを同期すること、(3) 複数回の反復にわたって平均をとること、(4) サブミリ秒の分解能を得るために `time.perf_counter` を使用することである。(上の実演セルではウォームアップと同期は実施したものの、32 回の積の合計時間を報告していた。以下で定義する `Benchmark` ヘルパは、本章および以降の章のためにこれら 4 つの原則をすべて自動化する。)

```
class Benchmark:
    """Time a callable: warmup, then device-synchronized average seconds."""
    def __init__(self, f, warmup=3, repeats=10, desc='time'):
        self.desc, out = desc, None
        for _ in range(warmup):
            out = f()
        jax.block_until_ready(out)
        t0 = time.perf_counter()
        for _ in range(repeats):
            out = f()
        jax.block_until_ready(out)
        self.time = (time.perf_counter() - t0) / repeats

    def __repr__(self):
        return f'{self.desc}: {1000 * self.time:.2f} ms/call'
```

JAX はデバイス全体ではなく特定の配列の完了を待機するため、計測対象の関数は計算結果を返す必要があり、`Benchmark` は最後の呼び出しが返した値の完了を待機する。

さっそく `Benchmark` を使って、先ほどの 2 つの主張を検証しよう。すなわち、大規模な行列積において GPU が CPU よりどれだけ高速か、そして計算に比べてホストからデバイスへの転送がいかに高コストかである。以下では、4096×4096 行列の積を CPU (fp32)、GPU の tf32 (5.5 節) および 16 ビット精度で計測し、あわせて単一の 4096×4096 行列を CPU RAM から GPU へコピーする時間も計測する。

```
if num_gpus() > 0:
    A_cpu = jax.device_put(A, cpu())  # The same matrix in main memory
    A16 = A.astype(jnp.bfloat16)
    print(Benchmark(lambda: A_cpu @ A_cpu, warmup=1, repeats=3,
                    desc='product on the CPU, fp32'))
    print(Benchmark(lambda: A @ A, desc='product on the GPU, tf32'))
    print(Benchmark(lambda: A16 @ A16, desc='product on the GPU, bf16'))
    # Wait for each copy, so that the copies are timed one at a time
    print(Benchmark(lambda: jax.device_put(A_cpu, gpu()).block_until_ready(),
                    desc='copy from CPU to the GPU'))
else:
    print('No GPU: there is nothing to compare the CPU with.')
```

```
product on the CPU, fp32: 136.53 ms/call
product on the GPU, tf32: 1.61 ms/call
product on the GPU, bf16: 0.86 ms/call
copy from CPU to the GPU: 8.55 ms/call
```

JAX のデフォルトの行列積の精度設定では、この GPU 上で tf32 が選択される。

現代の GPU では、4096×4096 の積は 2 ms 未満で完了するのに対し、CPU では約 100 ms から 2 s を要する（CPU のコア数に応じて約 $100\times$ から $1000\times$ の高速化である）。16 ビット精度に落とすと GPU の計算時間はさらにほぼ半減し、5.5 節で予告したテンソルコアによる高速化が実現される。

最後の行に特に注目してほしい。行列の積ではコピーされた要素 1 つあたり 4096 回の積和演算が実行されるにもかかわらず、CPU メモリから GPU へ行列を単に**コピーする**だけで、到着後に積を計算するよりも 3〜6 倍**長く**かかっている。通常のホストからのコピーでは、GPU の DMA エンジンが直接データを引き出せるよう、まず**ピン留め**（pinned、ページロック）ホストメモリにデータを中継する。バッチをあらかじめピン留めしておけば、この余分なコピーが不要になり、データ転送と計算をオーバーラップさせることができる（13.2 節）。

5.7.3 GPU メモリ

5.7.3.1 メモリの 3 つのビュー

訓練スクリプトがどれだけの GPU メモリを使用しているかを尋ねると、ソフトウェアスタックのどの層に問い合わせるかによって 3 つの異なる数値が返ってくる。CUDA ドライバにメモリを直接要求する処理は遅いため、各フレームワークは内部で**キャッシングアロケータ**を動かし、ドライバから大きなメモリ領域（スラブ）をまとめて取得してテンソルへ細分化して割り当て、解放されたスラブも即座に返却せず再利用のために保持する。これにより、入れ子になった 3 つのメモリビューが生じる（Figure 5.16）。すなわち、生存中のテンソルが現在占有しているバイト数、フレームワークのアロケータが予約しているより大きなプール、そして CUDA ドライバがそのプロセスに割り当てた総メモリ量（`nvidia-smi` に表示される数値）である。

`gpu().memory_stats()` は、JAX が事前確保した GPU プール内での使用中バイト数（`'bytes_in_use'`）とピーク使用バイト数（`'peak_bytes_in_use'`）を報告する。

1 GiB の float32 テンソルを確保して直ちに削除し、これらのカウンタがどう反応するか見てみよう。

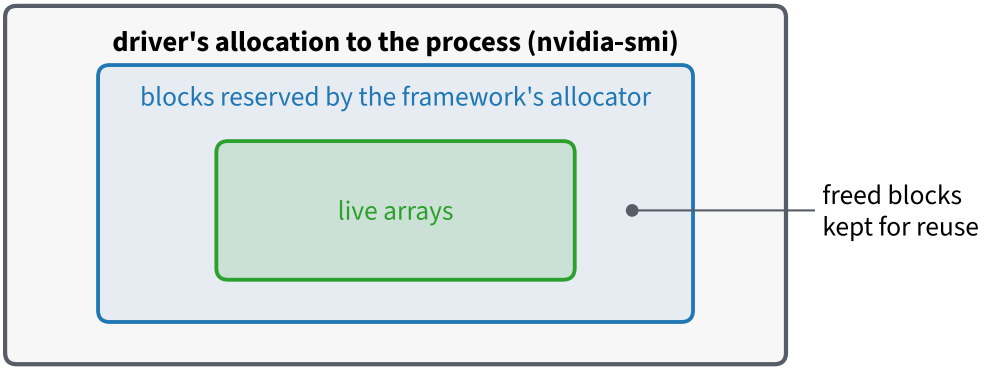


Figure 5.16: 3 つのメモリビューは入れ子になっている。nvidia-smi が報告するプロセスへのドライバ割り当ての中にアロケータの予約済みブロックが含まれ、さらにその中に生存中の配列が含まれる。

```
if num_gpus() > 0:
    def report(tag):
        s = gpu().memory_stats()
        print(f'{tag}: {s["bytes_in_use"] / 2**20:7.1f} MiB in use, '
              f'{s["peak_bytes_in_use"] / 2**20:7.1f} MiB peak')
    big = jnp.zeros((256, 1024, 1024))  # 1 GiB of float32
    big.block_until_ready()
    report('after creating big')
    del big
    report('after deleting it ')
else:
    print('No GPU: memory_stats() returns None on the CPU backend.')
```

```
after creating big:  1184.0 MiB in use,  1184.0 MiB peak
after deleting it :   160.0 MiB in use,  1184.0 MiB peak
```

`big` を削除すると生存中のテンソル 1 GiB 分が解放されるが、フレームワークのキャッシングアロケータは解放された 1 GiB のスラブを予約プール内に保持し続けるため、`nvidia-smi` の表示はピクリとも動かない。（訓練ループの内部でスラブを強制的に CUDA ドライバへ返却してはならない。毎ステップ低速なドライバ割り当てが発生してしまうからである。）最も外側のドライバビューは、メモリ不足の絶対的な上限を定める。ドライバはカードが物理的に搭載している以上の RAM を貸し出せないためである。コード内の**どこに**メモリが使われているかを診断するには、生存バイト数のカウンタ、とりわけ呼び出し中に達した**ピーク**メモリを調べる。このピーク値が、ステップが成功するかメモリ不足（out-of-memory、OOM）でクラッシュするかを決定する。

XLA はコンパイル済み関数のメモリを静的に計画するため、`compiled.memory_analysis().temp_size_in_bytes` によって入出力以外に必要な一時ワークスペース・活性化メモリのバイト数が報告される。

```
def peak_memory(f, *args):
    """Temporary memory that XLA reserves to run jax.jit(f) on args, in
    bytes."""
    compiled = jax.jit(f).lower(*args).compile()
    return compiled.memory_analysis().temp_size_in_bytes
```

5.7.3.2 訓練時のメモリ

5.2 節では、fp32 の Adam 訓練における静的メモリの下限を $16N$ バイト（パラメータ数を N として、重みに $4N$、勾配に $4N$、Adam のモーメントに $8N$）と導出した。実際の訓練ステップでは、ここに 4 つ目の項として**活性化値**が加わる。誤差逆伝播法（backpropagation）では、順伝播の中間出力を逆伝播（backward pass）で消費するまで保持しなければならず、静的な $16N$ とは異なり、活性化メモリはミニバッチサイズと層の幅に比例して増大する。実測する前に計算してみよう。1024 次元の入力を受け取る幅 4096・隠れ層 3 層の MLP（パラメータ数は $N \approx 3.8 \times 10^7$、したがって重みは $4N \approx 144$ MiB）を、ミニバッチサイズ 4096 で動かすとする。3 つの隠れ層の出力はそれぞれ 64 MiB の 4096×4096 fp32 テンソル（合計 192 MiB）となり、この時点ですでに重み自体より 3 分の 1 大きい。

```
if num_gpus() > 0:
    start = gpu().memory_stats()['bytes_in_use']
    def in_use():  # Memory in use beyond that at the start of the cell
        s = gpu().memory_stats()
        return f'{(s["bytes_in_use"] - start) / 2**20:7.1f} MiB'
    rngs = nnx.Rngs(d2l.get_key())
    net = nnx.Sequential(nnx.Linear(1024, 4096, rngs=rngs), nnx.relu,
                         nnx.Linear(4096, 4096, rngs=rngs), nnx.relu,
                         nnx.Linear(4096, 4096, rngs=rngs), nnx.relu,
                         nnx.Linear(4096, 10, rngs=rngs))
    key1, key2 = jax.random.split(d2l.get_key())
    X = jax.random.normal(key1, (4096, 1024))
    y = jax.random.randint(key2, (4096,), 0, 10)
    def loss(model):
        logits = model(X)
        return optax.softmax_cross_entropy_with_integer_labels(
            logits, y).mean()
    jax.block_until_ready(nnx.state(net))
    print('weights                ', in_use())
    grads = jax.block_until_ready(nnx.grad(loss)(net))
    print('+ gradients            ', in_use())
    optimizer = nnx.Optimizer(net, optax.adam(1e-3), wrt=nnx.Param)
    jax.block_until_ready(optimizer.opt_state)
    print('+ optimizer state      ', in_use())
    graphdef, params = nnx.split(net)
    grad_fn = jax.grad(lambda params: loss(nnx.merge(graphdef, params)))
    planned = peak_memory(grad_fn, params)
    print(f'activations (planned)  {planned / 2**20:7.1f} MiB')
else:
    print('No GPU: run this cell on a GPU to see the terms arrive.')
```

```
weights                 160.2 MiB
+ gradients             304.4 MiB
```

```
+ optimizer state          593.0 MiB
activations (planned)     192.0 MiB
```

出力された数値は、訓練ステップのライフサイクルを正確にたどっている。まず重み（$4N$ バイト）と入力がメモリ上に置かれ、順伝播によって保存された活性化値（約 192 MiB、重みを上回る）が加わり、逆伝播がその活性化値を消費・解放しつつ勾配（$4N$）を実体化し、最初のオプティマイザステップが Adam の 2 つのモーメント（$8N$）を確保する。Figure 5.17 は、16 MiB の入力バッチを除いたこれら 4 つの段階を示したものである。2 ステップ目以降は、毎ステップの中でメモリが増減する。順伝播で重みと Adam のモーメントの上に活性化値が保存され、逆伝播でそれが勾配に置き換わる。ピーク使用量は高々、静的な $16N$ に活性化値を加えた量であり、次の順伝播の前に古い勾配を解放するループでは最大 $4N$ だけ低くなる。活性化値が原因でメモリ上限を超えてしまう場合、アーキテクチャを変えずに活性化メモリを削減できるミニバッチサイズの縮小が最も手軽な対策となる。対照的に、メモリが**ステップを追うごとに**着実に増え続ける場合は、古い計算グラフを生かし続けているメモリリークを疑うべきである。典型的な原因は、生の損失テンソルを切り離さずそのまま Python のリストに追加してしまうことである。

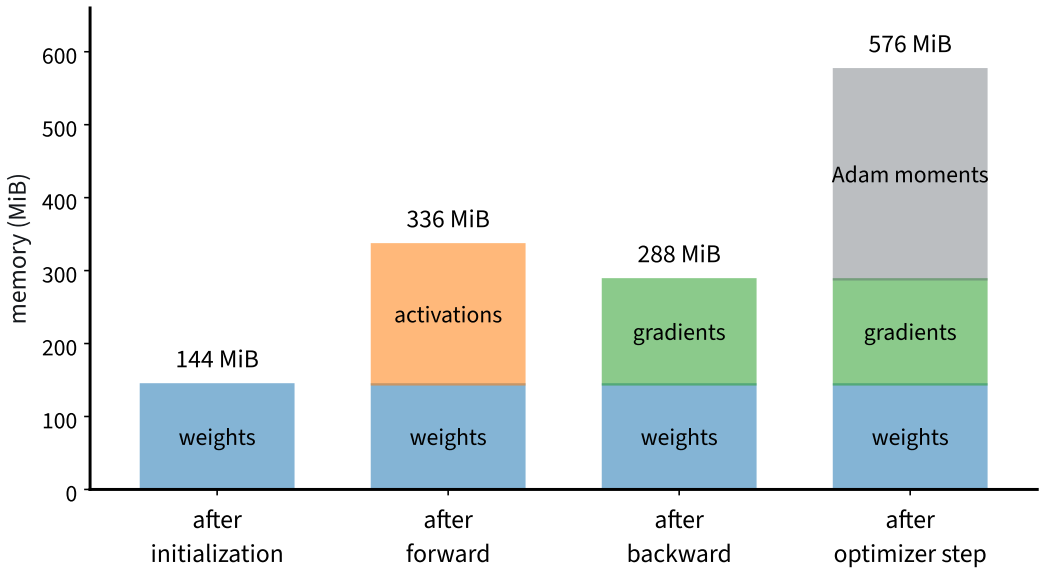


Figure 5.17: 幅 4096 の MLP をミニバッチサイズ 4096 で訓練した際の、最初の訓練ステップの各段階後における使用中メモリ（入力バッチは含めない）。順伝播では重み（$4N$）に 3 つの保存された活性化値が加わる。逆伝播では活性化値が解放されて勾配（$4N$）が残る。オプティマイザステップでは Adam の 2 つのモーメント（$8N$）が加わる。

5.7.3.3 活性化チェックポイント

活性化メモリを抑えるには、アーキテクチャ面での 2 つ目の手段がある。通常の誤差逆伝播法は、逆方向の計算で一度読み出すためだけに、すべての層の中間出力を保存している。**活性化チェックポイント**（**勾配チェックポイント**（gradient checkpointing）や**再実体化**（rematerialization）とも呼ばれる (Chen et al. 2016)）は、順伝播中に大半の中間活性化値を破棄して選択したセグメントの境界にある入力だけを保持し、逆伝播中に各セグメント内部の活性化値をその場で再計算する。（同じ「チェックポイント」という語を含むが、このメモリ上の手法は 5.6 節で扱ったディスクへのチェックポイント保存とは無関係である。）

均一な K 個の層を s 層ずつのセグメントに分割すると、保持されるのは K/s 個のセグメント境

界と、処理中のセグメント内部で生存している s 個の活性化値のみとなる（約 $K/s + s$ 個のテンソルであり、K 個すべてではない）。$s = \sqrt{K}$ と選べば、ピークの活性化メモリは $O(K)$ から $O(\sqrt{K})$ へと減少する。その代償は、誤差逆伝播法におけるセグメントあたり 1 回分の追加の順伝播である（Figure 5.18）。線形層の逆伝播が 2 回の行列積（入力勾配と重み勾配）を計算するのに対し、順伝播は 1 回の行列積で済むため、順伝播が 1 回増えてもステップ全体の計算量は順伝播換算で約 3 回分から 4 回分に増えるだけであり、ハードウェアや層の構成にもよるが大まかな上限として約 33% の増加にとどまる。

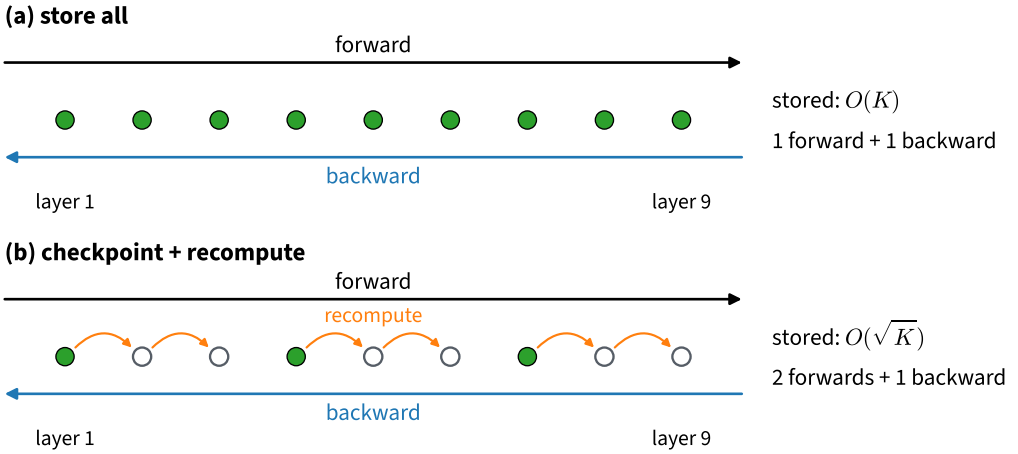


Figure 5.18: (a) 通常の誤差逆伝播法はすべての層の活性化値を保存する。(b) チェックポイントを用いると、セグメントの入力（塗りつぶし）のみを保存し、各セグメント内部の活性化値（白抜き）は逆伝播中に再計算する。K 個の層を $\sqrt{K}$ 層ずつのセグメントに分けると、1 回分の追加の順伝播をコストとして、保存される活性化値は $O(K)$ から $O(\sqrt{K})$ に減少する。

活性化チェックポイントが真価を発揮するのは、5.1 節の残差 MLP や現代の Transformer のように、同一構造のブロックを深く積み重ねたモデルである。以下では、重みを $\mathcal{N}(0, 0.01^2)$、バイアスを 0 で初期化した `ResidualBlock` と、ブロックをセグメントにまとめて任意でチェックポイントを適用する `run_stack` ヘルパを定義する。

```
class ResidualBlock(nnx.Module):
    def __init__(self, num_hiddens, *, rngs):
        normal = nnx.initializers.normal(0.01)
        self.body = nnx.Sequential(
            nnx.Linear(num_hiddens, num_hiddens, kernel_init=normal,
                       rngs=rngs), nnx.relu,
            nnx.Linear(num_hiddens, num_hiddens, kernel_init=normal,
                       rngs=rngs))

    def __call__(self, X):
        return X + self.body(X)

def run_stack(graphdef, states, X, use_checkpoint=False, segment_size=4):
    def segment(segment_states, X):
        for state in segment_states:
```

```
            X = nnx.merge(graphdef, state)(X)
        return X
    f = jax.checkpoint(segment) if use_checkpoint else segment
    for start in range(0, len(states), segment_size):
        X = f(states[start:start + segment_size], X)
    return X
```

jax.checkpoint（または jax.remat）は（nnx.split で得られる）純粋関数を変換し、jax.grad がセグメント境界の入力のみを保存して誤差逆伝播法の実行中に内部の活性化値を再実体化するようにする。

まず健全性チェックとして、活性化値の再計算によって数学的な計算結果が一切変わらないことを確認しておこう。8 ブロックのスタックを 2 つのチェックポイント付きセグメントに分割し、すべての勾配が通常の誤差逆伝播法と一致することを検証する。

```
X = jax.random.normal(d2l.get_key(), (32, 64))
blocks = [ResidualBlock(64, rngs=nnx.Rngs(d2l.get_key())) for _ in range(8)]
split_blocks = [nnx.split(block) for block in blocks]
graphdef = split_blocks[0][0]
states = [state for _, state in split_blocks]

def stack_grads(use_checkpoint):
    loss = lambda states: run_stack(graphdef, states, X,
                                    use_checkpoint).sum()
    return jax.tree_util.tree_leaves(jax.grad(loss)(states))

for g, g_ckpt in zip(stack_grads(False), stack_grads(True)):
    assert jnp.allclose(g, g_ckpt)
print('checkpointed gradients match the ordinary ones')
```

```
checkpointed gradients match the ordinary ones
```

この正しさの確認はどの環境でも実行できるが、メモリと計算量のトレードオフを実測するため、次は GPU 上でより深い 16 ブロックのスタック（幅 1024、バッチサイズ 8192）を 4 ブロックずつのセグメントにまとめ、**ステップメモリ**（パラメータと入力を除いたピークメモリ）と Benchmark による実時間の両方を計測する。

```
if num_gpus() > 0:
    X = jax.random.normal(d2l.get_key(), (8192, 1024))
    blocks = [ResidualBlock(1024, rngs=nnx.Rngs(d2l.get_key()))
              for _ in range(16)]
    split_blocks = [nnx.split(block) for block in blocks]
    graphdef = split_blocks[0][0]
    states = [state for _, state in split_blocks]
    for use_checkpoint in (False, True):
        step = jax.jit(jax.grad(lambda states: run_stack(
            graphdef, states, X, use_checkpoint).sum()))
        plan = step.lower(states).compile().memory_analysis()
        timing = Benchmark(lambda: step(states))
        peak = plan.temp_size_in_bytes + plan.output_size_in_bytes
        print(f'checkpointing={use_checkpoint!s:5}  step memory '
```

```
                f'{peak / 2**20:5.0f} MiB, {timing.time * 1e3:5.1f} ms per step')
else:
    print('No GPU: the peak-memory comparison needs a GPU.')
```

```
checkpointing=False  step memory  1152 MiB,  22.7 ms per step
checkpointing=True   step memory   512 MiB,  27.6 ms per step
```

ここでのステップメモリとは、コンパイルされたステップが入力以外に使用する計画のメモリ、すなわち一時配列（主に活性化値）とその出力である勾配の合計である。JAX では peak_bytes_in_use をリセットできず、前のセルの実行ですでに値が上がっているため、この目的には使えない。

チェックポイントなしでは、逆伝播は 16 ブロックすべての活性化値を同時に保持する。4 ブロックずつのセグメントにまとめると、保持されるのは 4 つのセグメント境界と処理中の 1 セグメントのみとなり、計算時間が約 4 分の 1 増える（先ほどの概算による上限 33% の範囲内である）だけで、ステップメモリは約 1.1 GiB から 0.4〜0.5 GiB へと削減される。この有利なトレードオフにより、大規模な Transformer や深い視覚モデルの訓練において、活性化チェックポイントは標準的な手法となっている。

5.7.4 デバイス対応の Trainer

これでようやく num_gpus を d2l.Trainer に組み込むことができる。ここまでの原則を実践に移すには、たった 2 つの手順で済む。訓練が始まる前にモデルパラメータを目的のデバイス上に配置すること、そして毎ステップの冒頭で各ミニバッチをそのデバイスへ移動することである。

```
@d2l.add_to_class(d2l.Trainer)
def __init__(self, max_epochs, num_gpus=0, gradient_clip_val=0):
    self.save_hyperparameters()
    self.gpus = [d2l.gpu(i) for i in range(min(num_gpus, d2l.num_gpus()))]
    self.board = d2l.ProgressBoard(xlabel='epoch', ylabel='loss')

@d2l.add_to_class(d2l.Trainer)
def prepare_batch(self, batch):
    if self.gpus:
        batch = [d2l.to(a, self.gpus[0]) for a in batch]
    return batch
```

min(num_gpus, d2l.num_gpus()) を使うことで、num_gpus は必須条件ではなく上限として扱われる。CPU のみのマシンでは self.gpus が空になり、訓練はそのままシームレスに CPU 上で実行される。本書の残りの章におけるすべての Trainer(num_gpus=1) 呼び出しの背後にあるのは、まさにこのフォールバック機構である。

小さなモデルの訓練は、GPU によって実際どれほど速くなるのだろうか。4096×4096 の行列積は約 $100\times$ 以上高速化されたが、エンドツーエンドの訓練ループにはデータ読み込み、Python のディスパッチ、そしてより小さな層が含まれる。以下では、幅 256 の残差ブロック（5.7.3.3 節）を 2 つ重ね、重みを $\mathcal{N}(0, 0.01^2)$、バイアスを 0 で初期化した Fashion-MNIST 分類器で計測してみよう。

```
class ResMLPClassifier(d2l.Classifier):
    def __init__(self, num_hiddens=256, num_blocks=2, lr=0.1, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
```

```
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        normal = nnx.initializers.normal(0.01)
        self.net = nnx.Sequential(
            lambda x: x.reshape((x.shape[0], -1)),
            nnx.Linear(784, num_hiddens, kernel_init=normal, rngs=rngs),
            nnx.relu,
            *[ResidualBlock(num_hiddens, rngs=rngs) for _ in range(num_blocks)],
            nnx.Linear(num_hiddens, 10, kernel_init=normal, rngs=rngs))
```

Fashion-MNIST におけるウォームアップ後の 1 エポックの実行時間を、`num_gpus=0` と `num_gpus=1` で比較する。

```
data = d2l.FashionMNIST(batch_size=256)

def epoch_time(num_gpus):
    # JAX creates the parameters, and runs on the minibatches, on the default
    # device, the GPU whenever there is one: num_gpus=0 alone would not move
    # the run to the CPU, so we set the default device
    with jax.default_device(try_gpu() if num_gpus else cpu()):
        model = ResMLPClassifier()
        trainer = d2l.Trainer(max_epochs=1, num_gpus=num_gpus)
        trainer.board.display = False
        trainer.fit(model, data)  # The first epoch warms up and compiles
        return Benchmark(trainer.fit_epoch, warmup=0, repeats=1).time

for gpus in (0, 1):
    print(f'num_gpus={gpus}: {epoch_time(num_gpus=gpus):.1f} sec per epoch')
```

```
num_gpus=0: 1.3 sec per epoch
num_gpus=1: 0.7 sec per epoch
```

この高速化倍率は良い現実確認になる。この小さな Fashion-MNIST ネットワークでは、GPU での 1 エポックは CPU より高々数倍速いだけであり、100× の向上（4096×4096 の行列積での倍率）には遠く及ばない。ここでは最大の行列でも 784×256 にすぎないため、GPU の計算能力の大部分が遊んでしまい、データの読み込みとカーネル起動のオーバーヘッドが実行時間の大半を占めている。次章以降で扱うように、大きな畳み込み、アテンションヘッド、幅の広い線形層が計算負荷の主役になって初めて、GPU は本領を発揮する。

メモリの観点では、この小さな SGD モデルは数 GB もの余裕を残して楽に収まる。しかし、80 億パラメータのモデルに立ち返ってみよう。バッチサイズを縮小し活性化チェックポイントを有効にすれば活性化メモリは抑えられるものの、fp32 の重み・勾配・Adam のモーメントが占める 128 GB の静的な下限は手つかずのままであり、しかも `Trainer` は OOM に直面する前に事前警告を出すことはできない。概算によるメモリ見積もりは依然として不可欠であり、単一デバイスのメモリを超えてスケールさせるには第 13 章で扱うマルチ GPU の並列性（parallelism）へと進むことになる。

5.7.5 まとめ

すべてのテンソルとモデルパラメータは特定のデバイス上に存在し、あらゆる演算はそのオペランドが同じデバイス上にあることを要求する。ホスト・デバイス間の転送は GPU の計算よりはるかに遅いため、モデルは最初に一度だけ配置し、バッチはステップの境界で転送し、内側ループはデバイス上に留める必要がある。GPU はコマンドキューから非同期に実行するため、Python

が停止するのは同期ポイント（スカラーから Python の浮動小数点数への変換や NumPy への変換など）のみであり、信頼できるベンチマークにはウォームアップ、明示的な同期、および反復試行が欠かせない。訓練時のメモリは静的なモデル状態・オプティマイザ状態とバッチ依存の活性化値からなり、活性化チェックポイントは約 1 回分の追加の順伝播をコストとして活性化メモリを $O(K)$ から $O(\sqrt{K})$ へと削減する。最後に、デバイス対応の `Trainer` は単一 GPU への配置を自動化する一方で、GPU がない環境では CPU へとスムーズにフォールバックする。

5.7.6 演習問題

1. [code] **ピークメモリ**。幅が 1024、4096、4096、4096 で出力数が 10 の 4 層 MLP を、5.7.3.2 節と同様に Adam で訓練する場合を考える。同項の計算方法を用いて、重み、勾配、オプティマイザ状態、および逆伝播が保持する活性化値を数え上げ、バッチサイズ 64、256、1024 における 1 訓練ステップのピークメモリを予測せよ。次に、GPU 上で実際のピーク値を計測せよ。この予測はどこで崩れ、何を見落としていたか。

2. **チェックポイントの計算**。幅 1024 の `ResidualBlock` を 16 個積み重ねたスタックをバッチサイズ 8192 で動かす場合について、Figure 5.18 のスケジュールを紙の上で書き出せ。

    1. 1 つのブロック内部で逆伝播が保持しなければならない配列を挙げ、その要素数を数えよ。
    2. チェックポイントなしの場合と、セグメントサイズを 1、2、4、8 ブロックとした場合について、逆伝播のピーク時に保持される活性化メモリを計算せよ。また、各スケジュールがどれだけの順伝播計算を繰り返すか述べよ。
    3. この設定ではどのセグメントサイズを選ぶべきか。また、その選択は 5.7.3.3 節で述べた約 $\sqrt{K}$ 層ごとのセグメントという規則とどのように関係するか。

3. [code] **非同期ディスパッチ**。●以下の両問とも、5.7.2.1 節の時間計測の原則を用いよ。

    1. 256 から 8192 までの n について、$n \times n$ 行列を CPU から GPU へ送り再び戻す往復時間と、デバイス上での 1 回の $n \times n$ 行列積の時間を計測せよ。両者の時間が等しくなる n を求めよ（この範囲内で交差しない場合は、各時間が n に対してどう増加するかから外挿せよ）。使用している GPU の公称ホスト・デバイス間帯域幅と行列積スループットからこの交差点を予測し、予測値と実測値の間に差があればその理由を説明せよ。
    2. GPU が 2 基ある場合、2 つの 4096×4096 行列の積を 1 基の GPU 上で 1000 回実行する時間と、同じループから 2 基の GPU にそれぞれ 500 回ずつ発行して実行する時間を計測せよ。スケーリングを計測し、非同期ディスパッチによって Python がどのように両デバイスへ処理を発行できるかを説明したうえで、線形スケーリングから乖離する要因を特定せよ。実験コードは `num_gpus() >= 2` でガードせよ。

4. [code] **同期ポイント**。●GPU 上における `ResMLPClassifier` の訓練実行（5.7.4 節）を考える。

    1. モデルのコードに加えて `Trainer.fit_epoch`、`Trainer.prepare_batch`、`Averager` を読み、1 訓練ステップ内のすべての同期ポイントを列挙せよ。それぞれを「正しいループにとって不可避なもの」か「回避可能な計測・記録上の副産物」かに分類し、その分類の根拠を述べよ。
    2. ループが毎ステップ後に損失を Python の浮動小数点数へ読み出すとする。作成したリストをもとに、この読み出しによってデフォルトの `ResMLPClassifier` と `ResMLPClassifier(num_hiddens=2048, num_blocks=8)` がそれぞれどれだけ遅くなるかを予測せよ。次に、各モデルについて読み出しあり・なしの 1 エポックを計測し、4 つの計測結果を説明せよ。

    PyTorch ドキュメントの CUDA semantics に基づき改変。

5.8 再現性と検査

訓練スクリプトを 1 行だけ書き換えたところ、Fashion-MNIST における検証正解率が 0.3 パーセントポイント上がったとしよう。ぬか喜びする前に、スクリプトを**変更しない**まま 2 回実行しただけでどの程度結果がばらつくかを知る必要がある。5.8.1.5 節では、乱数シード（random seed）だけを変えてソフトマックス回帰（softmax regression）を 5 回実行する。すると、初期重みとバッチのシャッフルが異なるだけで最終的な正解率には 1〜4 パーセントポイントもの幅が生じ、0.3 ポイントの改善など完全に埋もれてしまうことがわかる。

クラッシュする訓練のデバッグは、これと表裏一体の問題である。数時間の訓練の後に損失が爆発して NaN になった場合、クラッシュに至るまでのステップ列を正確に再現し、実行中のネットワークの内部を調べて、どの層が最初に非有限値を出力したのかを突き止めたい。

本節では、この両面をまとめて扱う。まず、擬似乱数生成器（pseudorandom number generator）、すなわちグローバルシード、専用のジェネレータオブジェクト、マルチワーカーのデータローダを制御して実行を再現可能にし、続いて GPU 特有のさらに厄介な問題である、並列アトミックリダクションのもとでの浮動小数点演算の非結合性に取り組む。次に、**フック**（および各フレームワークにおける同等の仕組み）を用いて、モデルのコードを変更することなく実行中のモデルを検査し、活性化値（activation）のスケールの記録、最初に NaN を出す層の捕捉、勾配ノルム（gradient norm）の監視を行う。この両面は互いに補完し合う関係にある。内部の診断結果を 2 つのコードバージョン間で比較できるのは、実行間で他の条件が一切変わらない場合に限られるからである。初読時はジェネレータオブジェクトとデータローダのワーカーに関する項を軽く読み流し、データパイプラインでランダムなデータ拡張（data augmentation）を扱うようになってから読み返してもよい。

5.8.1 シードと乱数生成器

訓練パイプラインには、多くの実務者が意識している以上に多様な箇所からランダム性が入り込む。たとえば、重みの初期化（5.3 節）、毎ステップのドロップアウトマスク、エポックごとのミニバッチのシャッフル、ランダムなデータ拡張（切り抜き、反転、色ジッター）、そして並列に動作するデータローダのワーカープロセスなどである。すべての乱数は**擬似乱数生成器**（PRNG）から生成される。これは、初期**シード**によって出力系列全体が決定される決定論的なアルゴリズムである。したがって、パイプラインが利用するすべての乱数生成器にシードを設定すれば、あらゆる乱数抽出を固定できる。

5.8.1.1 グローバルシード

深層学習フレームワークは、共有されたグローバルな乱数生成器から乱数を引き出すか、あるいは生成器の状態を引数として明示的に渡すことを要求する。以下では、重みと合成入力のすべてをシード付きの 1 つの乱数源から生成する、5 ステップの小規模な訓練ループでシード設定の実例を示す。

JAX にはグローバル RNG がない。すべての乱数関数は明示的な**キー**（`jax.random.key(seed)`）を受け取り、`jax.random.split` によって独立したストリームを派生させる。

```
def train_once(seed):
    key_init, key_X, key_y = jax.random.split(jax.random.key(seed), 3)
    rngs = nnx.Rngs(key_init)
    net = nnx.Sequential(nnx.Linear(20, 32, rngs=rngs),
                         nnx.relu,
                         nnx.Linear(32, 1, rngs=rngs))
```

```
    X = jax.random.normal(key_X, (128, 20))
    y = jax.random.normal(key_y, (128, 1))
    optimizer = nnx.Optimizer(net, optax.sgd(0.1), wrt=nnx.Param)
    loss_fn = lambda model: ((model(X) - y) ** 2).mean()
    for _ in range(5):
        loss, grads = nnx.value_and_grad(loss_fn)(net)
        optimizer.update(net, grads)
    return loss, net.layers[0].kernel[...]

loss_a, w_a = train_once(seed=0)
loss_b, w_b = train_once(seed=0)
loss_c, _ = train_once(seed=1)
print('same seed, identical loss and weights:',
      bool(loss_a == loss_b), bool(jnp.array_equal(w_a, w_b)))
print('different seed:', loss_a, 'vs', loss_c)
```

```
same seed, identical loss and weights: True True
different seed: 1.0159674 vs 0.79100513
```

`seed=0` を指定した 2 回の実行は**ビット単位で**一致する（すべての重みと最終損失が同一になる）。同一の初期重み、データ、および決定論的な算術演算からは同一の勾配が生じるからである（演算順序については 5.8.2 節で調べる）。シードを 1 に変更すると、異なる軌跡（trajectory）と損失が得られる。

5.8.1.2 ジェネレータオブジェクト

単一のグローバル PRNG に依存すると、そこから乱数を引くすべてのコードが結合してしまう。層を 1 つ追加したり、デバッグ中に乱数を伴う一時的な表示処理を入れたりするだけで、後続の乱数ストリーム全体がずれ、訓練・検証データの分割、重みの初期化、ドロップアウトマスクが気づかないうちに一斉に変わってしまう。専用の**ジェネレータオブジェクト**を使えば、独自のシードの背後にストリームを分離できるため、他の箇所での無関係な乱数抽出によって乱されることは決してない。

JAX のキーはすべて最初から状態を持たない独立した値であるため、`key_other` を使った乱数抽出が `key_split` に影響することは決してない。

```
key_split, key_other = jax.random.split(jax.random.key(42))
split = jax.random.permutation(key_split, 10)
_ = jax.random.normal(key_other, (1000,))  # unrelated draws, their own key
split_again = jax.random.permutation(key_split, 10)
print(bool(jnp.array_equal(split, split_again)), split)
```

```
True [1 3 9 5 0 2 6 7 4 8]
```

5.8.1.3 データローダのワーカー

データローダが並列の**ワーカー**プロセスでミニバッチを準備する場合、ランダムなデータ拡張には気づきにくい 2 つの失敗モードがある。ワーカーが OS の `fork` によってメインプロセスのクローンとして起動され、シードが再設定されない場合、すべてのワーカーがまったく同じ生成器の状態を引き継ぎ、同一の「ランダム」なデータ拡張を生成してしまう。逆に、各ワーカーが OS のエントロピーから初期化されると、2 回の実行が一致することは二度となくなる。

JAX では、ワーカーは `jax.random.split(key, num_workers)` や `jax.random.fold_in(key, epoch)` で派生させた明示的なキーを受け取るため、2 つのワーカーに同じキーを渡さない限り、意図しない状態の共有は起こり得ない。

5.8.1.4 値としてのランダム性

上述のワーカーに関するバグはいずれも、隠れた可変な生成器の状態が裏側で暗黙のうちにコピーまたは共有されることに起因する。関数型の乱数 API は、ランダム性を明示的な値として扱うことで隠れた状態を排除する。すなわち、すべての乱数呼び出しはキーを受け取り、同じキーで 2 回呼び出せば同一の数値が返り、独立したストリームはキーを明示的に分割することで得る。誤ってキーを再利用してしまう可能性は残るものの、その再利用はコード上にそのまま現れる。その代償は、確率的なすべての関数にキーを引き回す手間である。グローバルシードとはこの同じ契約を暗黙に略記したものにすぎず、専用のジェネレータオブジェクトを使えば必要な箇所で明示的な分離を取り戻せる。

JAX の `jax.random` はこの方式で動作する。

5.8.1.5 シード間のばらつき

ここで、本節の冒頭で触れたシード間の分散を実際に測定してみよう。以下では、Fashion-MNIST に対するソフトマックス回帰（3.3.5 節）を GPU 上で 3 エポック、5 つのシード（0〜4）にわたって訓練する。関係するすべての乱数生成器にシードを設定し、各実行の最終的な検証正解率と重みを返す。

```
class KeyedFashionMNIST(d2l.FashionMNIST):
    """Fashion-MNIST whose shuffle order is drawn from a key."""
    def __init__(self, key, batch_size=256):
        super().__init__(batch_size)
        self.key = key

    def get_dataloader(self, train):
        X, y = self.train if train else self.val
        X = (X[..., None] / 255).astype(np.float32)  # channel axis last
        return d2l.ArrayLoader((X, y.astype(np.int32)), self.batch_size,
                               shuffle=train, key=self.key)

def train_classifier(seed):
    key_init, key_order = jax.random.split(jax.random.key(seed))
    model = d2l.SoftmaxRegression(num_outputs=10, lr=0.1,
                                  rngs=nnx.Rngs(key_init))
    trainer = d2l.Trainer(max_epochs=3, num_gpus=1)
    trainer.board.display = False
    trainer.fit(model, KeyedFashionMNIST(key_order))
    return trainer.history['val_acc'][-1], model.net.kernel[...]

runs = [train_classifier(seed) for seed in range(5)]
accs = 100 * np.array([acc for acc, _ in runs])
print('validation accuracy (%):', ' '.join(f'{a:.1f}' for a in accs))
print(f'mean {accs.mean():.1f}, max - min {np.ptp(accs):.1f} points')
```

```
validation accuracy (%): 82.1 82.0 82.1 81.0 81.1
mean 81.7, max - min 1.2 points
```

5 回の実行すべてで同一のコードとハイパーパラメータを実行しているにもかかわらず、最終的な正解率は 81% 前後で 1〜4 パーセントポイントの広がりを見せており、1 回の実行で得られた 0.3 ポイントの上昇は単なるノイズにすぎないことが確認できる。信頼できる実験比較では、必ず複数のシードにわたる平均とばらつきを報告する。

この差がシードの選択だけによって生じたことを確かめるため、`seed=0` を再実行し、正解率とすべての重みを最初の実行と比較してみよう。

```
acc, w = train_classifier(0)
print('same accuracy:', acc == runs[0][0],
      '| same weights:', bool(jnp.array_equal(w, runs[0][1])),
      f'| largest difference {float(jnp.abs(w - runs[0][1]).max()):.2g}')
```

```
same accuracy: True | same weights: True | largest difference 0
```

再実行の検証正解率は最初の実行と完全に一致する。個々の重みが最下位ビットまで一致するかどうかは、GPU が浮動小数点数を一定の順序で足し合わせるかどうかに依存しており、これはシードだけでは制御できない（5.8.2 節）。

5.8.2 決定論的な算術演算

シードの設定は、実行時に**どの**乱数を引くかを固定するものであり、浮動小数点演算がそれらを**どのように**組み合わせるかまで固定するわけではない。浮動小数点の加算は毎回、表現可能な最も近い値へと丸められるため、結合法則を満たさ**ない**。すなわち、足し合わせる順序を変えると結果が変わり得る。

3 つの float32 の数値を使えば、この影響は一目瞭然となる。10^8 付近では、隣り合う float32 の数値の間隔が 8 になるため（5.5 節）、$-10^8 + 1$ は丸められて元の -10^8 に戻ってしまう。

```
a, b, c = [jnp.array(v, dtype=jnp.float32) for v in (1e8, -1e8, 1.0)]
print(float((a + b) + c), float(a + (b + c)))  # same numbers, regrouped
```

```
1.0 0.0
```

括弧の位置をずらすだけで、答えが `1.0` から `0.0` に変わってしまう。長いリダクションではこのずれはより微妙になるが、至る所で生じる。100 万個の浮動小数点数を先頭から足す場合と末尾から足す場合とでは、最下位ビット付近が日常的に食い違う。

スレッド数が固定された CPU 上では、リダクションの順序は通常一定に保たれるため、シードを設定した実行はきれいに再現される。一方 GPU 上では、数千ものスレッドが並列に競合しており、いくつかの高速なカーネル（埋め込みの勾配、スキャッタ加算（scatter-add）、一部の畳み込みの逆伝播、各種リダクションなど）は、スレッドが完了した順に到着する**アトミック加算**（atomic addition）によって部分結果を累積する。そのため、同じ GPU 上で同一のシードを設定した 2 回の実行であっても最下位ビットに差異が生じ、それがカオス的な勾配ステップによってやがて目に見えるほど異なる損失曲線へと増幅され得る。どのフレームワークにも、通常いくらかの速度コストと引き換えに一定の加算順序を強制する決定論モード（または代替カーネル）が用意されている。以下では、GPU 上の**スキャッタ加算**で 100 万個の乱数を 10 個のビンに 2 回続けて足し合わせ、この現象を直接観察してみよう。

`compiler_options={'xla_gpu_deterministic_ops': True}` を `jax.jit` に渡す（または `XLA_FLAGS=--xla_gpu_deterministic_ops=true` を設定する）と、決定論的な XLA GPU リダクションが強制される。

```
def scatter_add(idx, src):
    return jnp.zeros(10).at[idx].add(src)

if jax.default_backend() == 'gpu':
    src = jax.random.normal(d2l.get_key(), (2**20,))  # a million numbers
    idx = jax.random.randint(d2l.get_key(), (2**20,), 0, 10)  # their bins
    for deterministic in (False, True):
        f = jax.jit(scatter_add, compiler_options={
            'xla_gpu_deterministic_ops': deterministic})
        a, b = f(idx, src), f(idx, src)
        print(f'deterministic {deterministic!s:5}: bitwise equal '
              f'{bool(jnp.array_equal(a, b))}, largest difference '
              f'{float(jnp.abs(a - b).max()):.2g}')
else:
    print('no GPU: on a CPU this sum adds in a fixed order')
```

```
deterministic False: bitwise equal False, largest difference 0.0021
deterministic True : bitwise equal True, largest difference 0
```

決定論モードを有効にしない場合、まったく同じ GPU データに対する連続した 2 回のスキャッタ加算では、数百規模の和に対して数千分の 1 程度の差が生じる。決定論的カーネルを使えば、両方の呼び出しはビット単位で一致する。

ビット単位の再現性が成り立つ範囲を正しく認識しておく必要がある。それが保証されるのは同一のマシン、ソフトウェアスタック、ハードウェア構成の上だけであり、ライブラリのバージョンアップや異なる GPU アーキテクチャ、あるいは CPU と GPU の間でビット単位の一致を約束するフレームワークは存在しない。ビット単位の決定論は、2 回の実行が分岐する正確なステップを二分探索で特定するための非常に有用な**デバッグ**手段である。一方、科学的な再現性はこれより緩やかであると同時に、より強い要請でもある。本質的な改善（「このアーキテクチャ変更によって正解率が向上する」といった主張）は複数のシードにわたって頑健に成り立たなければならず、5.8.1.5 節と同様に平均とばらつき（約 81%、幅 1〜4 ポイント）として報告すべきである。16 ビット形式の場合（5.5 節）と同じく、シードの変更や最下位ビットの丸めによって覆るような主張には、より強い根拠が必要である。

5.8.3 フックによるモデルの検査

バグを確実に再現できたとしても、それで証明されるのはバグが実在することだけであり、ネットワーク内部の**どこで**発生しているかまではわからない。すべての層の `forward` メソッドに print 文を挿入するのは不格好であり、ソースコードをフォークしたくないサードパーティ製ライブラリのモデルでは現実的でもない。**フック**を使えば、この問題をきれいに解決できる。フックとはモデルの外部から登録するコールバックであり、層の実行前または実行後にフレームワークが自動的に呼び出す仕組みである。

`nnx.capture(net, nnx.Intermediate, method_outputs=nnx.Intermediate)(X)` は、モデルのコードを変更することなく、1 回の関数呼び出しですべてのサブモジュールの戻り値を取得する。

検査の例として、入力数 20 の層と出力数 10 のヘッドの間に幅 64 の残差ブロック（residual block）を 8 個積み重ねたネットワークを構築する（5.1 節）。各ブロックは $\mathbf{h} = \mathbf{x} + \mathrm{body}(\mathbf{x})$ を計算し、重みには Xavier 初期化（4.4.2.1 節）、バイアスにはゼロを用いる。実行ごとに新しいランダムな重みが引かれるよう、このネットワークにはあえてシードを設定しないでおく。以下の診断結果の測定値を典型的な範囲として示しているのはそのためである。

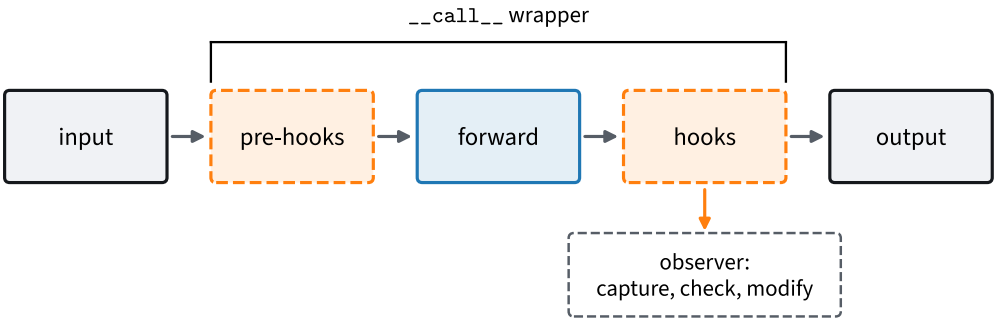


Figure 5.19: パイプラインとしての __call__ ラッパー。入力は事前フック、forward、フックを順に通って出力へと流れる。forward の青い実線に対し 2 つのフック段階はオレンジ色の破線で示されており、フック段階から横に出る矢印は値の取得・検査・変更を行うオブザーバを表す。

```
xavier = nnx.initializers.xavier_uniform()  # biases start at zero

class ResidualBlock(nnx.Module):
    def __init__(self, num_hiddens, *, rngs):
        self.body = nnx.Sequential(
            nnx.Linear(num_hiddens, num_hiddens, kernel_init=xavier,
                       rngs=rngs),
            nnx.relu,
            nnx.Linear(num_hiddens, num_hiddens, kernel_init=xavier,
                       rngs=rngs))

    def __call__(self, X):
        return X + self.body(X)

rngs = nnx.Rngs(d2l.get_key())
net = nnx.Sequential(nnx.Linear(20, 64, kernel_init=xavier, rngs=rngs),
                     *[ResidualBlock(64, rngs=rngs) for _ in range(8)],
                     nnx.Linear(64, 10, kernel_init=xavier, rngs=rngs))
X = jax.random.normal(d2l.get_key(), (256, 20))
```

5.8.3.1 活性化値の統計量

5.3 節では、ブロックを手動でループして深さ方向の活性化値の標準偏差を測定した。フックを使えば、モデルに一切手を加えることなく、標準正規分布に従う 256 個の入力からなるミニバッチに対してわずか数行でまったく同じプロファイルを収集できる。

```
_, inter = nnx.capture(
    net, nnx.Intermediate, method_outputs=nnx.Intermediate)(X)
for k, layer in enumerate(net.layers):
    out = inter['layers'][k]['__call__'][0]
    print(f'{type(layer).__name__:15s} std {out.std():.2f}')
```

```
Linear          std 0.70
ResidualBlock   std 0.85
ResidualBlock   std 1.02
ResidualBlock   std 1.31
ResidualBlock   std 1.61
ResidualBlock   std 1.95
ResidualBlock   std 2.51
ResidualBlock   std 3.21
ResidualBlock   std 4.16
Linear          std 5.71
```

活性化値の標準偏差は深さとともに着実に増加し、入力層の直後では約 0.7 だったものが 8 番目のブロックの後では 3〜4.5 前後に達しており、理論値とよく一致している。Xavier 初期化により $20 \to 64$ の入力層の重みの分散は $2/84$ となり、出力の分散は $20 \cdot 2/84 \approx 0.48$（標準偏差 0.69）となる。続く各残差ブロックは、入力とほぼ無相関で分散が入力分散の半分となる body 出力を加算するため（最初の線形層は分散を保ち、ReLU が二乗平均を半分にし、2 番目の線形層がその半分を保つ）、ブロックごとに全体の分散は 1.5 倍（標準偏差は $\sqrt{1.5} \approx 1.22$ 倍）になる。8 ブロック全体では $0.69 \times 1.5^4 \approx 3.5$ となり、実測値のちょうど中央に収まる。これは 5.3 節で行った分散増加の解析そのものであり（ここでは係数が 1.5 であり、He 初期化のもとでの 2 とは異なる）、深い残差ブランチにおいて出力射影（output projection）のスケールを縮小することが有効な理由を改めて裏づけている。

`nnx.capture` はモデルに永続的な状態を付与しないため、後から切り離したり削除したりする必要はない。

5.8.3.2 NaN 検出器

ステップ 40,000 で訓練損失（training loss）が突然 NaN になったとき、最初の診断上の疑問は**どの層が最初に非有限値を出力したのか**である。フックは順伝播（forward pass）の実行順に呼び出されるため、すべてのリーフ層に有限性チェックのフックを取り付ければ、1 回の順伝播で原因箇所を特定できる。この検出器を試すため、3 番目の残差ブロック（インデックス 3）の重みの 1 つに意図的に NaN を注入してみよう。

```
saved = net.layers[3].body.layers[0].kernel[0, 0]
net.layers[3].body.layers[0].kernel[0, 0] = float('nan')  # break one layer
_, inter = nnx.capture(
    net, nnx.Intermediate, method_outputs=nnx.Intermediate)(X)

def get_path(tree, path):
    for key in path:
        tree = tree[key]
    return tree

for path, module in nnx.iter_modules(net):
    if isinstance(module, nnx.Linear):
        out = get_path(inter, path)['__call__'][0]
        if not jnp.isfinite(out).all():
            print('first non-finite output in', path)
            break

net.layers[3].body.layers[0].kernel[0, 0] = saved  # restore the weight
```

```
first non-finite output in ('layers', 3, 'body', 'layers', 0)
```

層のクラス定義の中に print 文を 1 つも追加することなく、フックによってブロック 3 内の破損した層が即座に特定される。もちろん、意図的に仕込んだ NaN を捕まえるのはわかりやすいデモの例にすぎない。実際の訓練クラッシュでは、この検出器が示すのは最初の**症状**であり、根本的な原因（活性化値の暴走的な増大が最終的にオーバーフローに至るなど）はさらに数層手前から始まっていることもある。

5.8.3.3 勾配の検査

逆伝播（backward pass）も同じ精度で監視できる。8 個の残差ブロックを通るにつれて順伝播の活性化値が増大したのだから、勾配も逆方向に増大するのだろうか。標準正規分布に従う 256 個の入力に対する出力の二乗平均を損失として用い、各残差ブロックの出力における勾配ノルムと、ブロック 3 の最初の線形層における重みおよびバイアスの勾配ノルムを記録してみよう。

逆伝播フックを使わずに中間出力の勾配を検査するため、各ブロックの出力にゼロテンソル z を加算し、z に関して微分する。

```
def loss_fn(model, zeros):  # zeros: added to each block's output
    H = model.layers[0](X)
    for block, Z in zip(model.layers[1:-1], zeros):
        H = block(H) + Z
    return (model.layers[-1](H) ** 2).mean()

zeros = [jnp.zeros((256, 64)) for _ in range(8)]
grads, block_grads = nnx.grad(loss_fn, argnums=(0, 1))(net, zeros)
print('block output gradients:',
      ' '.join(f'{float(jnp.linalg.norm(g)):.3g}' for g in block_grads))
layer = grads['layers'][3]['body']['layers'][0]  # block 3, first layer
print(f"kernel {float(jnp.linalg.norm(layer['kernel'][...])):.3g}, "
      f"bias {float(jnp.linalg.norm(layer['bias'][...])):.3g}")
```

```
block output gradients: 1.84 1.47 1.16 0.914 0.741 0.543 0.422 0.318
kernel 26.4, bias 5.91
```

8 番目のブロックから 1 番目のブロックへと逆向きにたどると、出力の勾配ノルムはブロックごとに約 1.2× から 1.4× ずつ（スタック全体では 5〜6 倍）増大しており、順伝播での 1.22× の増大とちょうど対称になっている。理由もまったく同じで、逆伝播において各残差ブロックがブランチ側の勾配をスキップ接続（skip connection）の勾配に足し戻すからである。個々のパラメータの勾配ノルムを単独で見てもあまり意味はないが、深さ方向に比較したり訓練ステップにわたって追跡したりすれば、勾配の消失や爆発の挙動が即座に浮かび上がる（演習問題を参照）。

これらのデバッグツールが何を解決し、何を解決しないのかを押さえておこう。異なる 2 台のマシンをビット単位で一致させることはできないし、1 回の実行だけで実験上の改善が本物かどうかを決着させることもできない（そのためには複数のシードにわたる分布を比較する）。しかし、再開可能なチェックポイント（5.6 節）と組み合わせれば、失敗するジョブをクラッシュ直前まで正確に再生し、診断用のフックを取り付けて原因となっている層をピンポイントで特定するために必要な道具はすべて揃ったことになる。

本章の冒頭で取り上げた 80 億パラメータの言語モデルが小さくなったわけではない（fp32 では依然として 32 GB を占める）。しかし、フレームワークがそれをどのように扱うかについて、もはやブラックボックスは何もない。設定からのアーキテクチャ構築、パラメータとバッファの監査、深いスタックの初期化、カスタム層とカスタム勾配の実装、混合精度（mixed precision）での訓練、クラッシュに強いチェックポイントの保存、GPU メモリの管理、そして実行中のモデルの検査は、すべてまったく同じモジュール木の上で動作している。このエンジニアリングの道具立て

を備えた今、実世界のデータの構造に合わせて設計されたアーキテクチャへと進む準備が整った。まずは第 6 章の畳み込みニューラルネットワーク（convolutional neural network）から始めよう。

5.8.4 まとめ

訓練におけるあらゆるランダムな選択（初期化、シャッフル、ドロップアウト、データ拡張）は、シードによって制御される擬似乱数生成器に行き着く。また、専用のジェネレータオブジェクトや明示的なキーを用いれば、無関係なコードによってストリームが乱されるのを防げる。浮動小数点の加算は結合法則を満たさず、並列な GPU のアトミック演算は任意のスレッド順で和を計算するため、ビット単位の再現性を得るには決定論的カーネルも必要であり、しかもそれが成り立つのは同一のマシンとソフトウェアスタックの上だけである。したがって、ビット単位の決定論はデバッグのための手段であり、科学的な結論は複数のシードにわたって成り立たなければならない。最後に、順伝播および逆伝播のフック（ならびに各フレームワークにおける同等の仕組み）を使えば、モデルに手を加えることなく、活性化値のスケール、勾配ノルム、最初に発生した NaN を内部から検査できる。

5.8.5 演習問題

1. [code] **残差の分散収支**。$\mathbf{x}_k$、$\mathbf{b}_k$、$\mathbf{h}_k$ をそれぞれブロック k の入力、body の出力、ブロックの出力とし、各分散は全要素にわたって計算するものとする。`nnx.capture`（`method_outputs=nnx.Intermediate`）の 1 回の呼び出しから、すべての層の出力とすべての body の出力を収集せよ。ブロック k の入力は、その直前の層の出力である。5.8.3.1 節の一覧における標準偏差が増大するのは、各ブロックが自身の body の出力をストリームに加算するためである。次の等式に注目する。

$$\mathrm{Var}[\mathbf{h}_k] = \mathrm{Var}[\mathbf{x}_k] + \mathrm{Var}[\mathbf{b}_k] \tag{5.12}$$

   すべてのブロックについてこの等式がどの程度よく成り立つかを測定し、この等式が省略している項を答えよ。

2. [code] **シードとデータ読み込み**。●標準の Python のみで 4 ワーカーの入力パイプラインをシミュレートせよ。20 個の特徴量を持つ 64 個の例からなるデータセットをワーカーごとに 1 つずつ計 4 つのシャードに分割し、各ワーカーが `jax.random.normal` を用いて担当する例にガウスノイズを加えるものとする。

   1. 1 つのベースキーを `jax.random.split(key, 4)` でワーカーごとのキーに分割し、それぞれに `jax.random.fold_in` でエポック番号を畳み込むことでノイズ用のキーを派生させよ。同じベースキーからの 2 回の実行が同一のノイズを生成することを検証せよ。次にワーカー数を 2 に減らし、各例が依然として同じノイズを受けるかどうかを確認せよ。
   2. 今度は、ベースキーにエポック番号を畳み込んでから例のインデックスを畳み込む方法で、各例のキーを派生させよ。両方の確認を繰り返し、例のワーカーへの割り当て方に依存しないノイズを与えるのがどちらの派生方法かを説明せよ。

3. [code] **発散モニタ**。●5.8.3.2 節の有限性チェックを、ステップごとの訓練モニタに作り変えよ。`net` を再構築し、標準正規分布に従う 256 個の入力からなる固定した 1 つのバッチを用い、出力と標準正規分布に従うターゲットとの平均二乗誤差を損失として、学習率 1 の SGD で訓練せよ。いずれかの層の出力が最初に非有限になるステップと、そのステップで深さ順で最初に非有限となる層を報告せよ。毎ステップの損失も確認し、いずれかの層の出力より先に損失が非有限になるかどうかを報告せよ。

4. [code] **勾配ノルム。**●`net` を再構築し、残差ブロック内のすべての線形層のカーネルを 3 倍にし、固定したバッチに対する出力の二乗平均を損失とせよ。

    1. 本節と同様にして各ブロックの出力勾配のノルムを計算し、さらに `nnx.grad` が返すツリーから各ブロックのパラメータ勾配のノルムを計算せよ。両方のノルムを深さに対してプロットせよ。
    2. 線形層のカーネルの勾配を、その層の入力と出力勾配を用いて表せ。この式を使って 2 つの深さ方向プロファイルが異なる理由を説明し、逆伝播信号の爆発を監視するにはどちらの系列を見るべきかを述べよ。

Chapter 6

畳み込みニューラルネットワーク

これまでに構築してきた画像分類器はすべて、画像の空間的な幾何構造を無視し、2 次元のピクセルグリッドを 1 次元のベクトルへと平坦化していた。第 4 章の MLP は、28×28 の Fashion-MNIST 写真を順序のない 784 個の数値のリストとして扱う。そのため、すべての画像のピクセルに同じ固定のランダム置換を施して再訓練しても、並べ替えられた画像が人間の目にはただの砂嵐にしか見えないにもかかわらず、MLP はまったく同じ正解率に到達する。人間はスニーカーを輪郭やテクスチャ、すなわち**隣接する**ピクセルが形作るパターンによって認識しており、輪郭はフレーム内のどこに現れても同じ意味を持つ。

初期の手がかりは神経科学から得られた。ネコとサルを用いた古典的な実験において、Hubel と Wiesel は初期視覚野のニューロンが視野内の局所的な領域、すなわち各ニューロンの**受容野**（receptive field）に現れる特定の向きを持つエッジやバーに応答することを発見した (Hubel and Wiesel 1959, 1962, 1968)。その後、Field (1987) は画像上のあらゆる位置にわたって走査する少数の局所フィルタからなるバンクを用いて、自然画像に対するこれらの応答をモデル化した（Figure 6.1）。

これらの知見に触発され、Fukushima の Neocognitron (Fukushima 1980) は局所的な検出器を空間全体にわたって繰り返される階層的な層へと組織化し、続いて LeCun らは手書きの郵便番号を認識するために、そのようなアーキテクチャを誤差逆伝播法（backpropagation）によってエンドツーエンドで訓練する方法を示した (LeCun et al. 1989, 1995)。さらに、訓練された視覚ネットワークのより深い層は、霊長類の視覚経路における高次の段階と表現上の類似性さえ示す (Kuzovkin et al. 2018)。もっとも、生物学が果たしたのはアーキテクチャ上の着想を与える役割であり、シリコン上の計算に対する設計図ではなかった。

これらのアーキテクチャの中核にあるのが**畳み込み層**（convolutional layer）であり、共有された小さな重みのカーネルを画像のあらゆる近傍にわたってスライドさせる。各出力ユニットは近くのピクセルのみに依存し、入力を平行移動させれば出力もそのまま平行移動する。これにより、局所性（locality）と平行移動同変性（translation equivariance）をネットワークに直接組み込みつつ、パラメータ数を何桁も削減できる。こうした**畳み込みニューラルネットワーク**（convolutional neural network、CNN）は、ImageNet における分水嶺となるブレークスルーを牽引し (Deng et al. 2009; Krizhevsky et al. 2012)、ビジョン Transformer（第 11 章）と並んで今なおコンピュータビジョンの礎であり続けている。

本章では、3 つのステップで第一原理から CNN を導出し構築する。まず、全結合層に局所性と平行移動同変性を課すと、線形演算としては畳み込み（convolution）が唯一の選択肢になることを示す（6.1 節）。続いて 2 次元の相互相関（cross-correlation）を実装し、勾配降下法（gradient descent）によってエッジ検出カーネルを学習させ、畳み込みを行列積へと書き換え（**im2col**）、

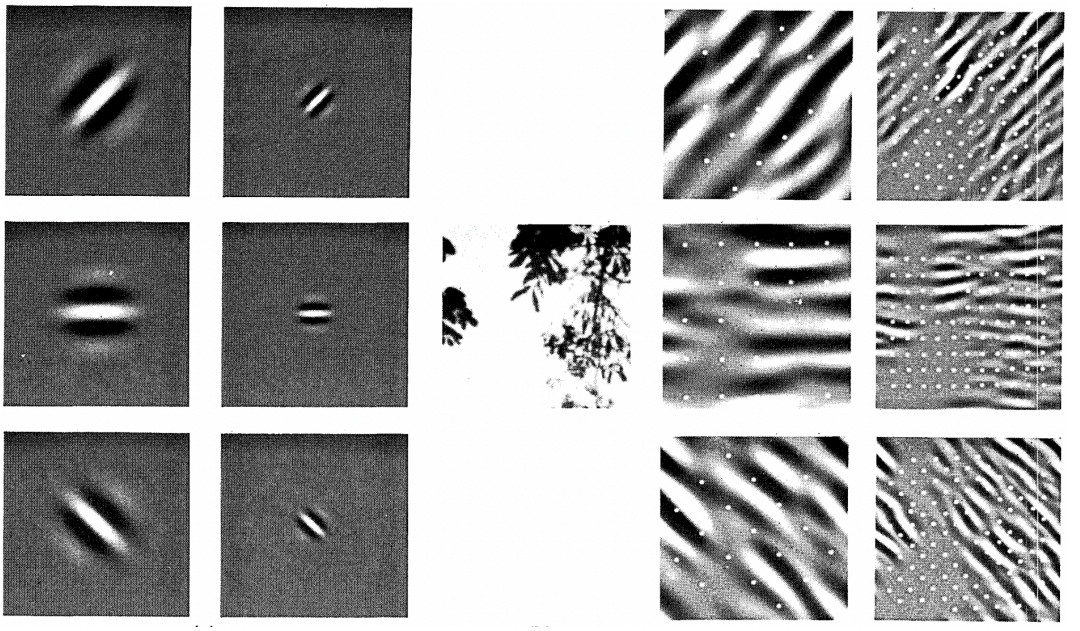


Figure 6.1: 6 つの受容野フィルタと自然画像に対するその応答（Field (1987) より改変）。左のパネルはフィルタ、中央のパネルは入力画像、右のパネルは各フィルタのサイズに比例した間隔でサンプリングされたフィルタ応答を示す。異なるフィルタは、それぞれ異なる局所的なエッジパターンに応答する。

層を積み重ねることで受容野がどのように広がるかをたどる（6.2 節）。次に、パディング・ストライド・ダイレーション（6.3 節）、マルチチャネルのフィルタバンクと 1×1 ボトルネック（6.4 節）、および局所プーリングとグローバルプーリング（6.5 節）を通じて、空間解像度、視野、特徴の表現容量を制御する。最後に、これらの要素を LeNet（6.6 節）へと組み立て、Fashion-MNIST 上で MLP と比較し、学習された最初の層のフィルタを観察する。さらに、対称性から MLP の正解率は変化しないピクセルシャッフル画像上で LeNet を再訓練することで、空間的な帰納バイアス（inductive bias）の寄与を切り分ける。現代の深層畳み込みアーキテクチャは、続く第 7 章で扱う。

参考資料と関連文献

以下の資料は本章の各節に対応しており、それぞれ関連する節を併記している。特記のない限り、すべてオンラインで無料公開されている。

書籍

- Deep Learning (Goodfellow, Bengio & Courville)：無料の HTML 版。第 9 章で疎な相互作用、パラメータ共有、同変性を詳しく展開している（6.1 節、6.5 節）。
- Understanding Deep Learning (Simon J. D. Prince)：無料の PDF 版。第 10 章で畳み込み、ストライド、ダイレーション、受容野を明快な幾何学的図解とともに解説している（6.2 節、6.3 節）。
- Neural Networks and Deep Learning (Michael Nielsen)：オンラインで無料公開。第 6 章で畳み込み層、共有された重み、プーリングを第一原理から導入している（6.2 節、6.5 節）。

講義コースと講義動画

- Stanford CS231n: Deep Learning for Computer Vision：無料の講義ノート。「Convolutional Networks」モジュール（層のサイズ計算、パラメータ数の計算、層の配置パターン）は、本章とあわせて広く利用されているオンライン副教材である（6.3 節、6.4 節）。
- Michigan EECS 498-007: Deep Learning for Computer Vision (Justin Johnson)：無料の講義動画。本章と同じペースで畳み込みとプーリングの計算を順を追って解説している（6.2 節、6.5 節）。

チュートリアル・解説記事・インタラクティブ教材

- A guide to convolution arithmetic for deep learning (Dumoulin & Visin, 2016)：無料、付属のアニメーションあり。パディング、ストライド、ダイレーション、転置畳み込みの視覚的なリファレンスである（6.3 節）。
- CNN Explainer (Wang et al., Polo Club)：活性化値、カーネル、受容野をブラウザ上で対話的に確認できる畳み込みネットワークの可視化ツールである（6.4 節、6.6 節）。
- Computing Receptive Fields of Convolutional Neural Networks (Araujo, Norris & Sim, 2019, **Distill**)：ストライドを含む場合や複数経路を持つ場合も含め、6.2 節で導入する受容野の閉形式の計算式を導出している。
- Conv Nets: A Modular Perspective (Chris Olah)：構造化された重み共有としての畳み込みについて直感を養う、簡潔な定番記事である（6.1 節）。
- Image Kernels Explained Visually (Victor Powell)：人手で設計したカーネルを画像上でスライドさせて試せるインタラクティブツールであり、6.2 節を補完する。
- Feature Visualization (Olah, Mordvintsev & Schubert, 2017, **Distill**)：畳み込みネットワークの各ユニットが何に応答するかを視覚的に探求する記事であり、6.6 節に対応する。

基礎となった論文

- Gradient-Based Learning Applied to Document Recognition (LeCun, Bottou, Bengio & Haffner, 1998, **Proc. IEEE**)：無料の PDF 版。6.6 節の背景にある LeNet-5 の論文である。
- Backpropagation Applied to Handwritten Zip Code Recognition (LeCun et al., 1989, **Neural Computation**)：誤差逆伝播法によってエンドツーエンドで訓練された最初の畳み込みネットワークの論文である（購読制限あり）（6.6 節）。
- Neocognitron (Fukushima, 1980, **Biological Cybernetics**)：畳み込みとプーリングの段階を交互に重ねる構造の、誤差逆伝播法以前における先駆的研究である（購読制限あり）（6.5 節）。
- Receptive fields, binocular interaction and functional architecture in the cat's visual cortex (Hubel & Wiesel, 1962, **J. Physiology**)：無料公開（PMC）。局所受容野と階層的な特徴検出の生物学的起源となった論文である（6.1 節）。

6.1 全結合層から畳み込みへ

第 4 章の多層パーセプトロン（multilayer perceptron、MLP）は、あらゆる画像を順序のない数値ベクトルへと平坦化する。これは表形式データには適しているが、どのピクセル同士が隣接しているかという情報を捨ててしまうため、画像に対しては無駄が多い。1 メガピクセル（1000×1000）のグレースケール写真では、わずか 1,000 ユニットの控えめな隠れ層でさえ $10^6 \times 10^3 = 10^9$ 個の重み（fp32 で 4 GB）を必要とし、元の解像度を保つ 1000×1000 の隠れマップともなれば 10^{12} 個もの重みを要求する。しかし、自然画像は任意の 100 万次元ベクトルとはまったく異なる。隣接するピクセル同士が強く相関する**局所性**（locality）があり、動物のひげのような視覚的特徴がフレーム内のどこに現れても同じ意味を持つ**平行移動同変性**（translation equivariance）を備えているからである。

2 次元のピクセルグリッドを 2 次元の隠れグリッドへ写す密結合層（dense layer）に、これら

2 つの幾何学的原理を直接課すとどうなるだろうか。平行移動同変性だけでも 10^{12} 個の重みは約 4×10^6 個へと激減し、さらに各隠れユニットが参照する範囲を小さな局所近傍に制限すれば、重みは数百個にまで削減される。この両方の制約を課して残るものこそが、**畳み込み層**（convolutional layer）である。

6.1.1 同変性と不変性

子どもの遊びである「ウォーリーをさがせ !」（Figure 6.2）は、視覚認識における本質的な課題を的確に捉えている。ごちゃごちゃした各場面には気を散らすものが溢れており、ウォーリーは思いがけない場所に隠れているが、**ウォーリーがどう見えるか**は**どこに立っているか**に依存しない。したがって、単一のウォーリー検出器で画像全体を走査し、あらゆる位置でまったく同じ規則を用いて、各パッチがどれだけ強くウォーリーと一致するかのスコアを計算できる。現代の物体検出（object detection）(Girshick et al. 2014) やセマンティックセグメンテーション (Long et al. 2015) のパイプラインは、この考え方を直接の土台としている。

Figure 6.2: ウォーリーを見つけられるだろうか（画像提供：William Murphy (Infomatique)）。

この直観から、視覚ネットワークの初期の層に対する 2 つの設計指針が導かれる。

1. 同じ局所パッチに対しては、それがどこに現れても同一の応答を返すべきである。すなわち、入力をシフトさせれば出力も同じ量だけシフトする。
2. 画像の離れた領域は気にせず、小さな局所近傍のみに着目すべきである。これが**局所性**の原理である。

後続の層はこれらの局所的な応答を組み合わせることで、段階的により大きく抽象的なパターンを構成し、最終的に画像全体の予測へと至る。

第 1 の原理を定式化しよう。シフト作用素 T_v を、画像をオフセット $v = (v_1, v_2)$ だけ平行移動させる作用素とし、ピクセル (i, j) を $(i + v_1, j + v_2)$ へ移すことで $[T_v \mathbf{X}]_{i,j} = [\mathbf{X}]_{i-v_1, j-v_2}$ が成り立つものとする。画像から画像への写像 f は、入力をシフトさせると出力もまったく同じベクトルだけシフトするとき**平行移動同変**であり、一方で画像からラベルへの写像は、入力をシフトさせても出力がまったく変化しないとき**平行移動不変**（translation invariant）である。すなわち、次式が成り立つ。

$$\begin{aligned} f(T_v \mathbf{X}) &= T_v f(\mathbf{X}) \quad \text{(equivariance)}, \\ f(T_v \mathbf{X}) &= f(\mathbf{X}) \quad \text{(invariance)}. \end{aligned} \tag{6.1}$$

Figure 6.3 は、平行移動同変性を可換図式として示したものである。ここで f は、画像のあらゆるウィンドウに対して小さなテンプレートを照合する演算であり、これこそが本節でこれから導出しようとしている演算にほかならない。先にシフトしてから照合しても、先に照合してからシフトしても、まったく同じマップが得られる。

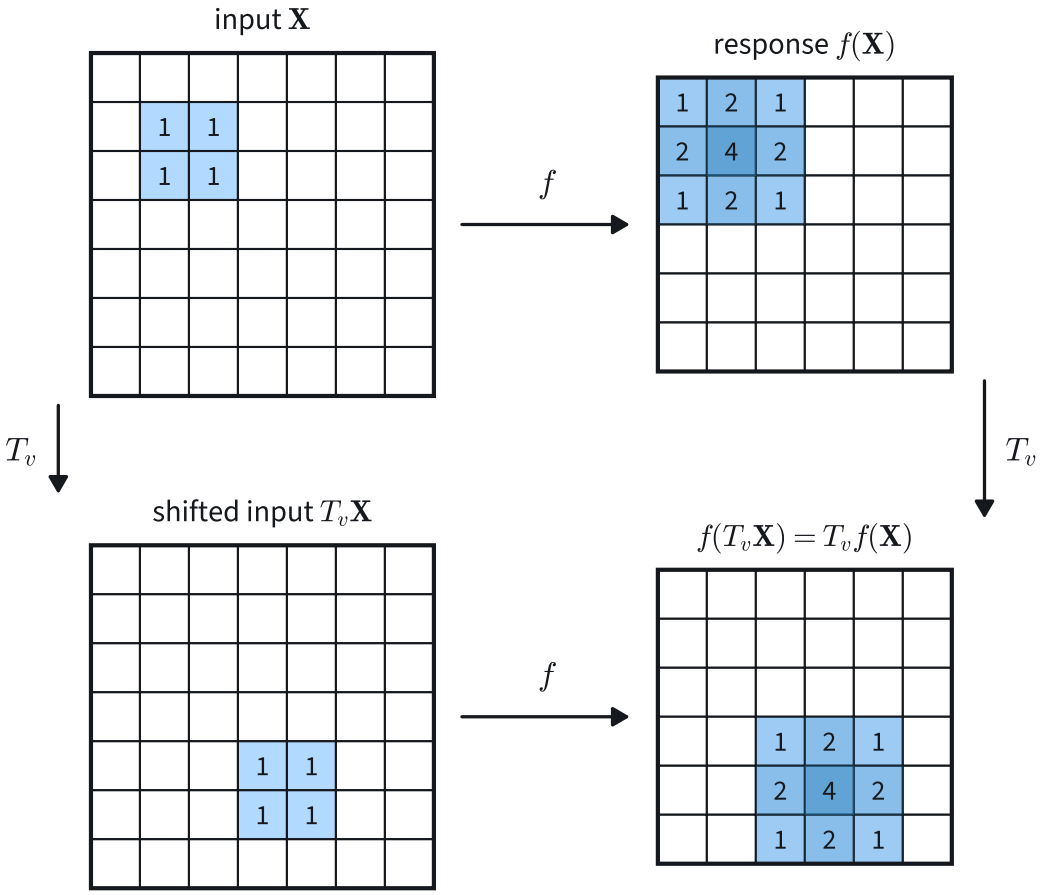


Figure 6.3: 平行移動同変性。入力にはすべて 1 からなる 2×2 のパッチが含まれており、f はすべての 2×2 ウィンドウをそのパッチと比較するため、ウィンドウがパッチ全体を覆う位置で応答は 4 になる。入力を $v = (3, 2)$ だけシフトしてから f を適用したマップ（下段）は、先に f を適用してから応答をシフトしたマップ（右列）と一致する。

ここで、明確な役割分担に注目しよう。**同変性**はパターンが**どこに**現れるかを保持するのに対し、**不変性**は位置情報を捨てる。ウォーリー検出器は、そのスコアマップが場面内のウォーリーを追跡できるよう同変であるべきだ。一方、「この画像にウォーリーはいるか」というイエス・ノーの問いへの回答は不変であるべきであり、これは最後に同変なマップを空間全体にわたって集約する（たとえば最大値をとる）ことで達成される。

6.1.2 全結合層への制約

重み行列全体を図示できる 1 次元で考えると、これら両方の制約はきわめて明快になる。8 サンプルの信号 $x_0, \ldots, x_7$ が全結合層（fully connected layer）によって 8 個の隠れユニットへ

$h_i = u_i + \sum_k W_{i,k} x_k$ と写される場合を考えよう。この 8×8 の重み行列は 64 個の独立な成分を持ち、Figure 6.4 のパネル (a) に示されている。

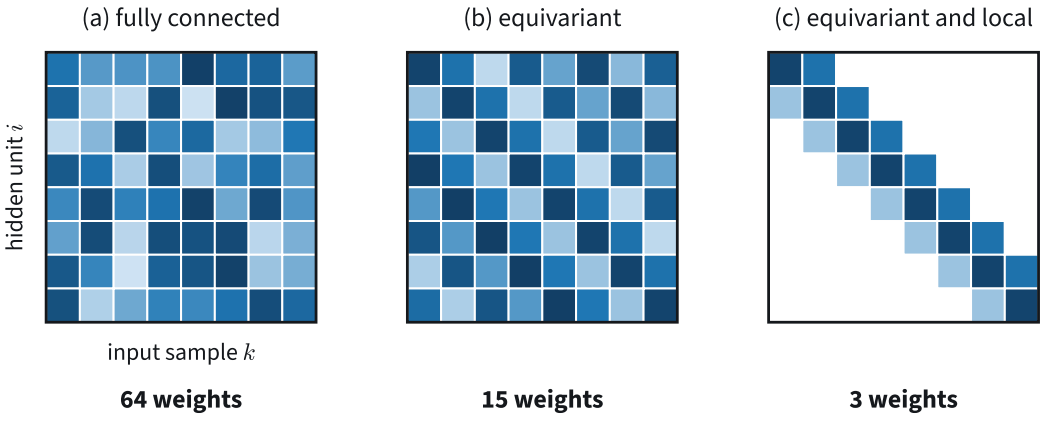


Figure 6.4: 8 サンプル（列）の 1 次元信号を 8 個の隠れユニット（行）へ写す線形層の重み。同じ色は等しい重みを表し、白は 0 を表す。(a) 全結合：すべての重みが自由パラメータである。(b) 平行移動同変：重みは入力と出力のオフセットのみに依存するため、行列は各対角線に沿って一定となる。(c) 同変かつ局所的：オフセット -1、0、1 のみが非ゼロの重みを持つ。

平行移動同変性は $\mathbf{W}$ にどのような条件を課すだろうか。いったんバイアスを無視して（後述の証明で、すべてのバイアスが一致しなければならないことを示す）、サンプル k にのみ単一の 1 を持つ単位インパルスを入力すると、層は列番号 k に対応する $\mathbf{W}$ の列を出力する。このインパルスを $k+1$ へシフトさせると、層は第 $k+1$ 列を出力する。同変性により、第 $k+1$ 列は第 k 列を 1 行下にシフトしたものと等しくなければならない。すなわち $W_{i+1,k+1} = W_{i,k}$ である。その結果、パネル (b) のように、$\mathbf{W}$ はすべての対角線に沿って一定でなければならない（**テプリッツ**行列）。各対角線は単一の相対オフセット $a = k - i$ に対応するため、64 個の自由な重みはわずか 15 個（-7 から 7 までのオフセット）へと縮約される。

局所性は、残った 15 個の重みのほぼすべてを刈り込む。各隠れユニットが距離 1 以内の近傍のみを参照するように制限すると、$\{-1, 0, 1\}$ 以外のすべてのオフセットが 0 になり、パネル (c) のようにすべての行で共有されるわずか 3 個の重みだけが残る。これにより、この層は重み 3 個からなる 1 つの小さなテンプレートで信号全体を走査することになる。これこそが導出全体の縮図であり、2 次元画像への拡張は 2 つ目の空間インデックスを追加するだけでよい。

画像の場合、入力 $\mathbf{X}$ と隠れ表現 $\mathbf{H}$ はどちらも 2 次元グリッドであり、入力ピクセル $[\mathbf{X}]_{k,l}$ が (k,l) に、隠れユニット $[\mathbf{H}]_{i,j}$ が (i,j) に位置する。すべての隠れユニットをすべてのピクセルに結合すると、重みは 4 階テンソル W となり、これにバイアス行列 $\mathbf{U}$ が加わる。

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l}.$$

両方の制約とも、隠れユニット (i,j) に対するピクセルの**相対的な**位置に依存するため、オフセット $a = k - i$ および $b = l - j$ によってインデックスを張り替え、$[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}$ と定義する。

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}. \tag{6.2}$$

ここで、a と b は正と負の両方のオフセットにわたる。このインデックスの張り替えは純粋に表記上のものであり、W を対角線に沿って読み替えているだけで、パラメータはまだ削減されていない。1000×1000 の画像を 1000×1000 の隠れグリッドへ写す場合、この単一の層は依然として $10^6 \times 10^6 = 10^{12}$ 個の重みを必要とする。

6.1.2.1 平行移動同変性

1 次元の場合と同様に、Equation 6.2 の形をした層が平行移動同変となるのは、その重みとバイアスが位置 (i, j) に依存しないとき、かつそのときに限る。重みの共有から同変性が導かれることの確認は 1 行で済むが、重要なのはその逆である。同変性を課すと線形な選択肢がほかに残らないため、畳み込みが必然的に導かれるからである。

境界（シフトによって有限フレームの端からピクセルが押し出される箇所）を見通しよく扱うため、台（support）の外側を 0 でパディングした無限グリッド上に画像を埋め込み、情報がグリッドの外へ失われないようにする。有限サイズの画像では、同変性は境界から離れた領域で厳密に成り立つ。この詳細については 6.3 節で扱う。

命題： 入力（非ゼロのピクセルを有限個のみ持つ）と隠れ表現が無限グリッド上にあるとする。このとき、Equation 6.2 の層があらゆるシフト v に対して平行移動同変となるのは、あるスカラー u と行列 $\mathbf{K}$ が存在して、$[\mathbf{U}]_{i,j} = u$ かつ $[\mathsf{V}]_{i,j,a,b} = [\mathbf{K}]_{a,b}$ がすべての i, j, a, b について成り立つとき、かつそのときに限る。

証明： 重みもバイアスも (i, j) に依存しないならば、入力を v だけシフトさせると出力もそのまま v だけシフトする。

$$
\begin{aligned}
[f(T_v\mathbf{X})]_{i,j} &= u + \sum_a \sum_b [\mathbf{K}]_{a,b} [\mathbf{X}]_{i-v_1+a, j-v_2+b} \\
&= [f(\mathbf{X})]_{i-v_1, j-v_2} = [T_v f(\mathbf{X})]_{i,j}.
\end{aligned}
$$

逆に、この層が同変であると仮定する。すべて 0 の画像はあらゆるシフトに対して不変であり、$\mathbf{U}$ へと写されることから、$\mathbf{U} = T_v\mathbf{U}$ がすべての v について成り立つ必要がある。したがって、バイアスは定数 u でなければならない。両辺から u を引けば、線形部分単独でも同変であることがわかる。ここで、(k, l) のみが単一の 1 で他がすべて 0 である画像を入力すると、(i, j) における応答は $[\mathsf{W}]_{i,j,k,l}$ となる。このインパルスを v だけシフトさせると $(k + v_1, l + v_2)$ へ移り、同変性から $[\mathsf{W}]_{i,j,k+v_1,l+v_2} = [\mathsf{W}]_{i-v_1,j-v_2,k,l}$ が要請される。$(k, l) = (a, b)$ かつ $v = (i, j)$ とおけば、$[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b} = [\mathsf{W}]_{0,0,a,b}$ が得られ、これは (i, j) に依存しない。■

要するに、単一の明るいピクセルでこの層を探ることで、すべての位置がまったく同じ重みを共有せざるを得なくなり、層は次式へと縮約される。

$$
[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{K}]_{a,b} [\mathbf{X}]_{i+a, j+b}. \tag{6.3}
$$

これにより、すべての隠れユニットは自身の座標を中心として同じパターン $\mathbf{K}$ を適用するようになる。ゼロパディングされた 1000×1000 の画像では、非ゼロのピクセルに触れうるのはオフセット $a, b \in [-999, 999]$ のみであるため、重みの数は 10^{12} 個から $1999^2 \approx 4 \times 10^6$ 個へと激減する。

6.1.2.2 局所性

単一のフィルタに 400 万個もの重みがあるのは依然として多すぎるうえ、その大半は隠れユニットを画像の反対側にある遠いピクセルと結びつけている。そこで**局所性**（$[\mathbf{H}]_{i,j}$ が各軸に沿って距離 Δ 以内のピクセルのみに依存するという条件）を課すと、$[\mathbf{K}]_{a,b} = 0$ が $|a| > \Delta$ または $|b| > \Delta$ のときに成り立ち、次式が得られる。

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{K}]_{a,b} [\mathbf{X}]_{i+a,j+b}. \tag{6.4}$$

これにより、各隠れユニットはサイズ $(2\Delta+1) \times (2\Delta+1)$ の (i,j) を中心とする局所ウィンドウ上で重み付き和を計算するようになり、使用するパラメータはわずか $(2\Delta+1)^2$ 個の共有された重みと 1 個のバイアスだけになる。すなわち、重みの数は $\Delta = 1$（3×3 ウィンドウ）なら 9 個、$\Delta = 5$ なら 121 個である。通常 Δ は 10 よりずっと小さいため、隠れマップの空間解像度を完全に保ったまま、4×10^6 個の重みが数十〜数百個の重みへと置き換わる。

Equation 6.4 は、**畳み込みニューラルネットワーク**（convolutional neural network、CNN）の基本構成要素である**畳み込み層**を定義している。共有される行列 $\mathbf{K}$ は**畳み込みカーネル**（convolution kernel）または**フィルタ**と呼ばれる。空間的および時間的な重み共有（weight sharing）の先駆的な研究としては、音声認識のための時間遅れニューラルネットワーク (Waibel et al. 1989) や、Zhang et al. (1988) によるシフト不変ネットワークが挙げられる。

重要な点として、パラメータ数が 10^{12} から $(2\Delta+1)^2$ へと縮小したのは、仮説クラス（hypothesis class）を局所的かつ平行移動同変な写像に制限した結果である。この組み込まれた**帰納バイアス**（inductive bias、4.5 節）により、エッジやテクスチャのように特徴が局所的で位置に依存しない場合にはサンプル効率が劇的に向上する。もっとも、単一の層だけでは、同一のパッチを絶対位置のみによって区別することはできない（たとえば、パスポート写真の上部付近にある目と、下の方にある鼻孔の区別など）。非線形性を挟んで複数の層を積み重ねることで、より深い層のユニットは段階的に広い領域にわたる文脈を統合できるようになる。

6.1.3 畳み込みと相互相関

厳密に言うと、深層学習における「畳み込み」という名称は少し不正確である。数学では、2 つの 1 次元関数の離散**畳み込み**（convolution）は $(f * g)(i) = \sum_a f(a) g(i-a)$（$\mathbb{R}^d$ 上では積分）で定義され、f と、**反転**させてシフトした g のコピーとの重なり具合を測る。2 次元では次のようになる。

$$(f * g)(i,j) = \sum_a \sum_b f(a,b) g(i-a, j-b). \tag{6.5}$$

これを Equation 6.4 と比較してみよう。そこでは $\mathbf{X}$ のインデックスが $(i+a, j+b)$ となっており、$(i-a, j-b)$ ではない。$-a$ を a に、$-b$ を b に代入すると、Equation 6.4 は $\mathbf{X}$ と反転させたカーネル $[\mathbf{K}']_{a,b} = [\mathbf{K}]_{-a,-b}$ との数学的な畳み込みであることがわかる。

$$\sum_a \sum_b [\mathbf{K}]_{a,b} [\mathbf{X}]_{i+a,j+b} = \sum_a \sum_b [\mathbf{K}']_{a,b} [\mathbf{X}]_{i-a,j-b} = (\mathbf{K}' * \mathbf{X})(i,j). \tag{6.6}$$

この反転操作はカーネルを両方の軸に沿って逆向きにするため、左上の重みが右下の重みへと移る（Figure 6.5）。

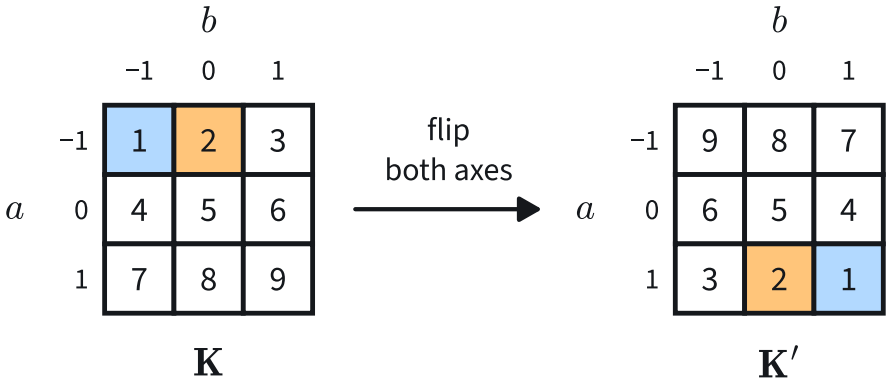


Figure 6.5: 3×3 カーネル $\mathbf{K}$ とその反転コピー $\mathbf{K}'$。行と列はオフセット a と b でインデックス付けされている。反転によって両方の軸が逆向きになり、色付きの 2 つの重みはオフセット $(-1,-1)$ と $(-1,0)$ から $(1,1)$ と $(1,0)$ へと移動する。$\mathbf{K}$ との相互相関は $\mathbf{K}'$ との畳み込みに等しい。

反転を伴わないこの演算は、正しくは**相互相関**（cross-correlation）と呼ばれる。CNN のカーネルはデータから学習されるため、この違いは表現力にまったく影響しない。すなわち、$\mathbf{K}$ との相互相関を計算する層は、$\mathbf{K}'$ との畳み込みを計算する層とまったく同じ関数族を学習する。深層学習フレームワークは相互相関を計算してそれを畳み込みと呼んでおり、本書でも全体を通してこの標準的な慣習に従う。

6.1.4 チャネル

ここまでは、入力と隠れマップの両方を単一のグレースケールグリッドとして扱ってきた。実際のカラー写真は 1 ピクセルあたり 3 つの輝度値（赤、緑、青）を持つため、画像は 3 階テンソル X となり、その形状は $3 \times \text{height} \times \text{width}$、各成分は $[\mathsf{X}]_{c,i,j}$ で表される。先頭の**チャネル**軸は各空間ピクセルに特徴ベクトルを対応付け、カーネルは局所的な空間ウィンドウとすべての入力チャネルの両方にわたって和をとる。

同様に重要な点として、単一のカーネルは単一の応答マップ（Figure 6.6 のウォーリーのスコアマップのようなもの）しか生成せず、1 つの視覚パターンしか捉えられない。豊かな視覚表現を得るには、方向を持つエッジ、コーナー、色のコントラスト、テクスチャなど、多数のパターンを並列に検出する必要があるため、隠れ表現 H もまた**出力チャネル**（output channel）軸に沿って積み重ねられた 3 階テンソルとなる。各出力チャネル o はそれぞれ固有の学習されたカーネルスライスとバイアス u_o を持ち、これによって K は 4 階テンソルとなる。

$$[\mathsf{H}]_{o,i,j} = u_o + \sum_c \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathsf{K}]_{o,c,a,b} [\mathsf{X}]_{c,i+a,j+b}. \tag{6.7}$$

ここで、c は入力チャネル、o は出力チャネルのインデックスであり、本章全体を通してこの「出力チャネルが先、入力チャネルが後」という順序の慣習を用いる。各出力チャネルは特定の学習された特徴がグリッド上のどこに現れるかを表すため、隠れ層のチャネルは**特徴マップ**（feature map）とも呼ばれ、そのスタック全体が次の層への入力となる。入力チャネル数を c_i、出力チャネル数を c_o とすると、この層は $(2\Delta+1)^2 c_\text{i} c_\text{o}$ 個の重みと c_o 個のバイアスを必要と

する。したがって、RGB 画像を 64 枚の特徴マップへ写す 3×3 の層が持つパラメータはわずか $9 \cdot 3 \cdot 64 + 64 = 1{,}792$ 個であり、画像の解像度にはまったく依存しない。

Figure 6.6: ウォーリー検出器：ウォーリーに一致するカーネルの応答は、ウォーリーの位置でピークに達する（画像提供：William Murphy (Infomatique)）。

この導出には 2 次元に固有の要素は一切ない。まったく同じ論理から、音声サンプルやテキストのトークン上をスライドする 1 次元畳み込みや、動画フレームや 3 次元医用スキャン上をスライドする 3 次元畳み込みが導かれる。

Equation 6.7 は、本章の残りの節すべての基礎となる。ただし、これは理想化された無限グリッド上で導出されたものであるため、実用上は 3 つの課題が残っている。すなわち、有限な画像境界の扱い、より深い層のユニットが見渡せる画像領域の拡大、そして空間的な特徴マップを画像全体の単一の判定（たとえば、ウォーリーがシーン内の**どこかに**現れているかどうか）へと集約する方法である。しかしその前に、まずは Equation 6.4 をコードとして実装してみよう。

6.1.5 まとめ

画像を構造のないベクトルへと平坦化すると、密結合層は扱いきれないほど巨大になる（1 メガピクセルの画像を同サイズの隠れグリッドへ写すのに 10^{12} 個の重みが必要となる）。画像に関する 2 つの自然な仮定がこのパラメータ数を数百個へと激減させる。すなわち、**平行移動同変性**（無限グリッド上で証明したとおり）によってすべての空間位置が 1 つのカーネルを共有し、**局所性**によってそのカーネルが小さなウィンドウに制限される。結果として得られる層は相互相関（深層学習では慣例的に畳み込みと呼ばれる）を計算し、複数の入力チャネルと出力チャネルを重ねることで、パラメータ数を画像サイズに依存させないまま単一の層で多数の視覚パターンを同時に検出できるようになる。これらの利点は帰納バイアスから直接もたらされるものである。畳み込み層は、位置に依存する任意の写像を表現する自由度を手放す代わりに、空間的構造を持つデータに対する高い汎化性能を獲得しているのである。

6.1.6 演習問題

1. **円環上の重み共有**。最後のサンプルの次が再び最初のサンプルに戻る周期グリッド上の 4 サンプルの信号（シフトによって情報が一切失われない）に対して、Figure 6.4 の議論を繰り返せ。
    1. 各サンプルの位置に順番に単一の 1 を層へ入力し、平行移動同変な層の 4×4 重み行列を書き下せ。自由な重みはいくつあるか。周囲を 0 で囲まれた 4 サンプルの場合の

$2 \cdot 4 - 1 = 7$ 個のオフセットと比較せよ。
2. オフセット -1、0、1 の局所性を課せ。行列のどの成分が非ゼロとなり、そのうち図のパネル (c) に対応する成分がないものはどれか。

2. **畳み込みの可換性**。
1. Equation 6.5 の畳み込みが可換であること、すなわち $f * g = g * f$ を証明せよ。
2. 相互相関 $(f \star g)(i,j) = \sum_a \sum_b f(i+a, j+b)\, g(a,b)$（入力 f とカーネル g に対する Equation 6.4 の演算）を定義する。$(g \star f)(i,j)$ を $f \star g$ の値を用いて表し、その式から相互相関が可換であるかどうかを判定せよ。

3. **音声に対する畳み込み**。音声データは多くの場合、サンプルの 1 次元系列として表現される。
1. Equation 6.4 の 1 次元版を導出せよ。
2. 時間方向に局所的かつ平行移動同変な層が適している音声タスクを 1 つ、適していない音声タスクを 1 つ挙げよ。それぞれの理由を説明せよ。
3. スペクトログラムは、録音を時間と周波数の 2 軸からなる 2 次元配列として表現する（周波数はヘルツ単位の線形スケールとする）。それぞれの軸について、平行移動同変性が妥当な仮定であるかどうかを述べ、その軸に沿ったシフトが音にどのような変化を与えるかという観点から理由を説明せよ。さらに、周波数軸が対数スケールである場合についても同じ問いに答えよ。

4. **同変性と不変性**。畳み込み分類器が、Equation 6.7 の層と非線形性を交互に積み重ねて特徴マップ $h(\mathsf{X})$ を計算し、続いて予測 $g(h(\mathsf{X}))$ をヘッド g によって計算するものとする。
1. 最後の列の次が再び最初の列になるよう折り返す周期グリッドを仮定すると、この上では命題の議論により h は平行移動同変となる。ここで g は $h(\mathsf{X})$ の各チャネルをすべての位置にわたって平均するものとする。Equation 6.1 を用いて、この予測が平行移動不変であることを示せ。
2. ある画像には 3×3 グリッドの 9 つのセルのいずれか 1 つに単一の数字が描かれており、どのセルに描かれているかを当てるタスクを考える。前半の分類器がこのタスクを解けない理由を説明せよ。また、ヘッド g として、同じ特徴マップ $h(\mathsf{X})$ からこのタスクを解けるものを記述せよ。

5. **回転同変性**。●本節の命題では、シフトに対する同変性から重み共有を導出した。回転についても同様の問いを考えよう。命題と同じ無限グリッド上で、R を原点まわりに画像を 90° 回転させる作用素 $[R\mathbf{X}]_{i,j} = [\mathbf{X}]_{-j,i}$ とする。
1. Equation 6.4 の畳み込み層において、$f(R\mathbf{X}) = Rf(\mathbf{X})$ がすべての画像 $\mathbf{X}$ について成り立つのは、$[\mathbf{K}]_{a,b} = [\mathbf{K}]_{-b,a}$ がすべてのオフセット (a,b) について成り立つとき、かつそのときに限ることを示せ。
2. $\Delta = 1$ のとき、このようなカーネルは自由な重みをいくつ持つか。評価対象のピクセルを中心に画像を 90° 回転させてもこのカーネルの応答が変化せず、したがって垂直エッジと水平エッジを区別できないことを示せ。
3. 1 つのカーネルに制約を課す代わりに、制約のない 1 つのカーネル $\mathbf{K}$ をそれぞれ 0°、90°、180°、270° 回転させたコピーをカーネルとして持つ 4 つの出力チャネルを用いるとする。入力を回転させると、各出力マップが回転するとともに 4 つのチャネルが巡回置換されることを示せ。前半の制約付きカーネルでは失われていたが、この層では保持されているものは何か。

6.2 画像に対する畳み込み

6.1 節では、平行移動同変性（translation equivariance）と局所性（locality）という 2 つの幾何学的原理から、畳み込み層（convolutional layer）を数式の上で導出した。ここでは、その式が画像上で実際に何を計算しているのか、そして勾配降下法によってデータから有用なカーネルを自動的に見つけ出せるのかを見ていく。

明るい領域と暗い領域が接する垂直方向のエッジを見つけることを考えよう。各ピクセルから右隣のピクセルを引くと（$[\mathbf{X}]_{i,j} - [\mathbf{X}]_{i,j+1}$）、一様な領域の内部では 0 になり、白から黒へ変化する箇所では $+1$、黒から白へ戻る箇所では -1 になる。2 つの重みからなる検出器 $(1, -1)$ を画像のすべての有効な位置にわたって走査すると、**エッジマップ**が得られる。本節では、2 次元相互相関（two-dimensional cross-correlation）を実装し、手作業で設計したエッジ検出器を試し、勾配降下法によってデータから直接 $(1, -1)$ を**学習**させる。さらに、畳み込みが単一の行列積（**im2col**）に帰着されることを示し、層を積み重ねることでユニットの受容野（receptive field）がどのように広がるかを導出する。

6.2.1 相互相関

6.2.1.1 演算の定義

ひとまずチャネルのことは脇に置き、入力 $\mathbf{X}$ を $n_\mathrm{h} \times n_\mathrm{w}$ 行列、カーネル $\mathbf{K}$ を $k_\mathrm{h} \times k_\mathrm{w}$ 行列とする。このカーネルは、形状 $k_\mathrm{h} \times k_\mathrm{w}$ のスライディング**ウィンドウ**を定める。ウィンドウの左上隅を (i, j) に置き、重なった入力パッチと $\mathbf{K}$ を要素ごとに掛けてその総和をとることで、位置 (i, j) の出力要素が得られる。

$$[\mathbf{Y}]_{i,j} = \sum_{a=0}^{k_\mathrm{h}-1} \sum_{b=0}^{k_\mathrm{w}-1} [\mathbf{K}]_{a,b} \, [\mathbf{X}]_{i+a,j+b}. \tag{6.8}$$

これが $\mathbf{X}$ と $\mathbf{K}$ の **2 次元相互相関**であり、コードに合わせて 0 始まりの添字を用い、$\mathbf{Y} = \mathbf{X} \star \mathbf{K}$ と表記する。6.1 節ではオフセットを $-\Delta$ から Δ まで対称に中心化したが、左上隅を基準点とする方式は一定のシフト Δ だけが異なるにすぎず、中心ピクセルを持たない偶数サイズのカーネル（2×2 カーネルや先ほどの 1×2 エッジ検出器など）も自然に扱える。

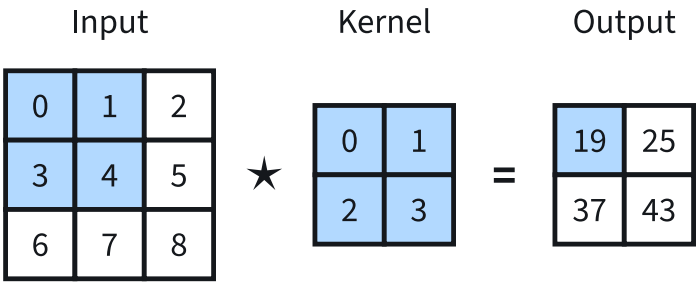


Figure 6.7: 2 次元相互相関。網掛け部分は、最初の出力要素と、その計算に使われる入力要素およびカーネル要素である（$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$）。

Figure 6.7 では、2×2 カーネルを 3×3 の入力上で左から右、上から下へと 1 ピクセルずつスライドさせている。ウィンドウは 4 つの位置にぴったり収まり、次の 2×2 の出力が得られる。

$$\begin{aligned}
0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 &= 19, \\
1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 &= 25, \\
3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 &= 37, \\
4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 &= 43.
\end{aligned}$$

カーネルは入力の内部に完全に収まらなければならないため、出力は各軸に沿ってわずかに縮小する。幅 k のウィンドウを長さ n の範囲に収めると有効な位置は $n - k + 1$ 個残るため、出力の形状は次のようになる。

$$(n_{\mathrm{h}} - k_{\mathrm{h}} + 1) \times (n_{\mathrm{w}} - k_{\mathrm{w}} + 1). \tag{6.9}$$

したがって、3×3 の入力と 2×2 カーネルからは 2×2 が、6×8 の画像と 1×2 カーネルからは 6×7 が、28×28 の Fashion-MNIST 画像と 5×5 カーネルからは 24×24 が得られる。空間次元を保ちたい場合は、入力の周囲を 0 の枠で囲む（**パディング**）。これについては、ストライドやダイレーションとあわせて 6.3 節で詳しく調べる。

6.2.1.2 実装

すべての出力座標を回る二重ループを用いて、Equation 6.8 を `corr2d` として直接実装する。

```
def corr2d(X, K):
    """Compute 2D cross-correlation."""
    h, w = K.shape
    Y = jnp.zeros((X.shape[0] - h + 1, X.shape[1] - w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y = Y.at[i, j].set((X[i:i + h, j:j + w] * K).sum())
    return Y
```

`corr2d` を Figure 6.7 の手計算と照合するため、3×3 の入力テンソル `X` と 2×2 のカーネル `K` を作成し、`corr2d(X, K)` を評価する。

```
X = d2l.tensor([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
K = d2l.tensor([[0.0, 1.0], [2.0, 3.0]])
corr2d(X, K)
```

```
Array([[19., 25.],
       [37., 43.]], dtype=float32)
```

出力は Figure 6.7 と要素ごとに一致する（また、6.1 節で示したように、反転させたカーネル $\mathbf{K}'$ を用いる数学的に厳密な畳み込みもまったく同じ層の族を学習するため、本書でも深層学習の標準的な用語法に従い、相互相関を「畳み込み」と呼ぶ）。

6.2.2 畳み込み層

6.2.2.1 層の定義

2 次元畳み込み層は、入力と学習されるカーネルの相互相関を計算し、スカラーのバイアスを加える。MLP の重み（第 4 章）と同様に、カーネルは $\mathcal{N}(0, 0.01^2)$ から初期化し、バイアスは 0 に初期化する。以下の `Conv2D` は両方のパラメータを登録し、順伝播（forward pass）で `corr2d` を呼び出す。高さ h、幅 w のカーネルを持つ層は、$h \times w$ 畳み込み層と呼ばれる。

```
class Conv2D(nnx.Module):
    def __init__(self, kernel_size, rngs=None):
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
```

```
        self.weight = nnx.Param(
            nnx.initializers.normal(stddev=0.01)(rngs.params(), kernel_size))
        self.bias = nnx.Param(jnp.zeros(1))

    def __call__(self, x):
        return corr2d(x, self.weight) + self.bias
```

全結合層とは異なり、畳み込み層のパラメータはカーネルサイズのみに依存するため、固定された入力解像度に縛られない。まったく同じ 2×2 の層（パラメータ数は合計 5 個）で、先ほどの 3×3 テンソル `X` と 6×8 の画像の両方を処理でき、それぞれ形状 2×2 と 5×7 の出力を返す。

```
layer = Conv2D((2, 2))
layer(X).shape, layer(d2l.ones((6, 8))).shape
```

```
((2, 2), (5, 7))
```

6.2.2.2 エッジ検出

ここで、冒頭で述べた垂直エッジ検出器を 6×8 の合成画像で試してみよう。この画像は、中央の 4 列が黒（`0`）で外側の列が白（`1`）になっている。

```
X = jnp.ones((6, 8))
X = X.at[:, 2:6].set(0)
X
```

```
Array([[1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.],
       [1., 1., 0., 0., 0., 0., 1., 1.]], dtype=float32)
```

次の 1×2 カーネルを定義する。

$$\mathbf{K} = \begin{bmatrix} 1 & -1 \end{bmatrix}. \tag{6.10}$$

すべての位置 (i, j) において、このカーネルは水平方向の**有限差分**（finite difference）である $[\mathbf{X}]_{i,j} - [\mathbf{X}]_{i,j+1}$ を計算する。これは輝度の水平方向の導関数を（符号を除いて）離散近似したものであり、平坦な領域ではゼロになり、画素の強度が急変する箇所で鋭く立ち上がる。

```
K = d2l.tensor([[1.0, -1.0]])
```

6×8 の縞模様の画像 `X` と 1×2 の水平差分カーネル `K` の相互相関を `corr2d(X, K)` で計算すると、6×7 のエッジマップ `Y` が得られる。

```
Y = corr2d(X, K)
Y
```

```
Array([[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
       [ 0.,  1.,  0.,  0.,  0., -1.,  0.]], dtype=float32)
```

予想どおり、`Y` は白から黒への境界に沿って +1、黒から白への境界に沿って -1 を出力し、それ以外の場所ではすべて `0` になる。まったく同じカーネルを転置した画像（縦縞を横縞に変えたもの）に適用すると、出力はすべて 0 になる。

```
corr2d(d2l.transpose(X), K)
```

```
Array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]], dtype=float32)
```

明らかに、`[1, -1]` は垂直方向のエッジにのみ反応する。水平方向のエッジや斜め線、コーナー、テクスチャを検出するには、まったく別のカーネルが必要である。

6.2.2.3 カーネルの学習

欲しい特徴が正確にわかっていれば `[1, -1]` を手作業で作るのは簡単だが、深いネットワーク全体にわたる何千もの中間フィルタを人手で指定することは誰にもできない。では、勾配降下法はカーネル `[1, -1]` を入出力の組 `(X, Y)` だけから復元できるだろうか。各フレームワークの組み込み 2 次元畳み込み層（バイアスは省略し、重みは $\mathcal{N}(0, 0.01^2)$ から初期化）を用い、全 42 個の出力要素にわたる二乗誤差（squared error）の和を、10 回の勾配ステップにわたり学習率 0.03 で最小化してこれを検証する。組み込みの層はバッチ軸とチャネル軸を含む 4 次元テンソルを入力として受け取るため（6.4 節）、それに合わせて `X` と `Y` の形状を変換する。

```
# padding='VALID' evaluates only where the window fits, as corr2d does
conv2d = nnx.Conv(1, 1, kernel_size=(1, 2), use_bias=False, padding='VALID',
                  kernel_init=nnx.initializers.normal(stddev=0.01),
                  rngs=nnx.Rngs(d2l.get_key()))
# Layers take (example, height, width, channel): one example, one channel
X = X.reshape((1, 6, 8, 1))
Y = Y.reshape((1, 6, 7, 1))
lr = 3e-2  # Learning rate

def loss(model, X, Y):
    return ((model(X) - Y) ** 2).sum()

for i in range(10):
    l, grads = nnx.value_and_grad(loss)(conv2d, X, Y)
    conv2d.kernel[...] -= lr * grads.kernel[...]
    if i % 3 == 0:
        print(f'step {i}, loss {l:.3f}')
```

```
step 0, loss 11.927
step 3, loss 0.820
step 6, loss 0.056
step 9, loss 0.004
```

初期状態では損失は 12 付近にある（ゼロに近いランダムなカーネルは、12 個ある ± 1 のエッジピクセルすべてを約 1 だけ外すためである）。しかし、わずか 9 ステップ以内に損失は 0.01 未満まで低下する。学習されたカーネルを調べると、何が発見されたかを確認できる。

```
conv2d.kernel[...].reshape((1, 2))
```

```
Array([[ 0.98783964, -0.98917454]], dtype=float32)
```

どちらの要素も `[1, -1]` から約 0.02 以内に収まっている。このループにはエッジに関する事前知識は何も組み込まれていない。損失の信号を与えるだけで、勾配降下法が自力でフィルタを見つけ出したのである。ここでは単一の層に `Y` を直接教師信号として与えたが、深い CNN では、誤差逆伝播法（backpropagation）によって最終的なタスク損失だけからすべての層のカーネルがエンドツーエンドで学習される。

6.2.3 行列積としての畳み込み

`corr2d` の入れ子ループは数式の意味を理解しやすいものの、Python で出力ピクセルを 1 つずつ処理するのでは実用的なワークロードには遅すぎる。古典的な解決策であり、現在でもライブラリが選択するアルゴリズムの 1 つとなっているのが、相互相関を 1 回の大きな行列積に書き換える方法である。行列積こそ、GPU が最高速度で実行できる演算にほかならない。Figure 6.7 の各出力要素は、平坦化した $k_\mathrm{h} k_\mathrm{w}$ 要素のカーネルと、対応する平坦化した入力ウィンドウとのドット積にすぎないことに注目しよう。$(n_\mathrm{h} - k_\mathrm{h} + 1)(n_\mathrm{w} - k_\mathrm{w} + 1)$ 個すべてのウィンドウを取り出して行として積み重ねるとパッチ行列が得られ、これに平坦化したカーネルを掛ければ相互相関全体を一度に計算できる。この標準的なメモリ配置の変換は **im2col**（「image to columns」の略だが、ここでは行に並べている）と呼ばれる。Figure 6.8 は、Figure 6.7 の例に対してこの変換を実行した様子である。

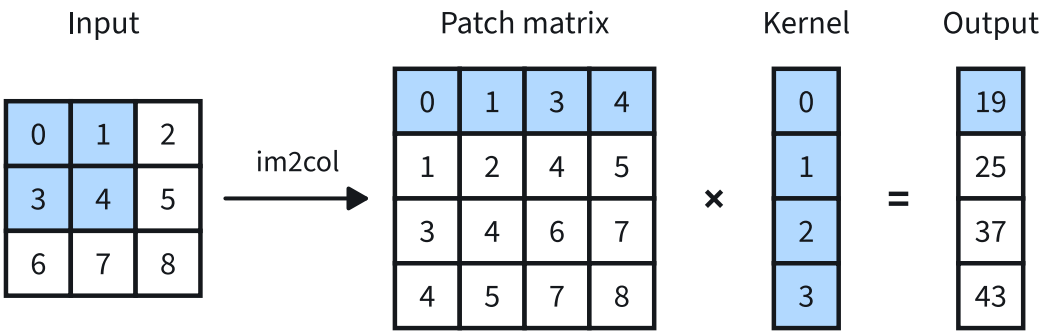


Figure 6.8: 1 回の行列積として表した Figure 6.7 の相互相関。入力の 4 つの 2×2 ウィンドウがそれぞれパッチ行列の 1 行になり、平坦化したカーネルを掛けることで平坦化された出力が得られる。網掛けのセルは、最初のウィンドウから対応する行、そして出力 19 までの対応を追ったものである。中央のピクセル 4 は 4 つすべてのウィンドウに含まれるため、すべての行に現れる。

引き換えとなるのは追加の作業用メモリである。内部の各ピクセルが最大 $k_\mathrm{h} k_\mathrm{w}$ 個のパッチに重複してコピーされるからである。以下では Figure 6.7 の例を用い、この 1 回の行列積が `corr2d` と

完全に一致することを確かめる。

```
X = d2l.tensor([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
K = d2l.tensor([[0.0, 1.0], [2.0, 3.0]])
h, w = K.shape
h_o, w_o = X.shape[0] - h + 1, X.shape[1] - w + 1
patches = d2l.stack([d2l.reshape(X[i:i + h, j:j + w], (-1,))
                     for i in range(h_o) for j in range(w_o)])
Y_mat = d2l.reshape(d2l.matmul(patches, d2l.reshape(K, (-1, 1))), (h_o, w_o))
Y_mat, d2l.reduce_sum(d2l.abs(Y_mat - corr2d(X, K)))
```

```
(Array([[19., 25.],
        [37., 43.]], dtype=float32),
 Array(0., dtype=float32))
```

差は恒等的に 0 であり、1 回の行列積によって `corr2d` が完全に再現されている。

これと等価な方法として、入力ではなく**カーネル**の方をスパースな帯行列（banded matrix）$\mathbf{W}$ に展開し、平坦化した画像に掛けることもできる。これは Figure 6.4 のパネル (c) のちょうど 2 次元版にあたり、どの行にも同じ少数のカーネル重みが繰り返し現れる。畳み込みを構造化された全結合層とみなすことで、6.1 節の導出へと一周して戻ってきたことになる（演習問題では $\mathbf{W}$ を具体的に構成する）。

6.2.4 特徴マップと受容野

単一の畳み込み層が見ているのはカーネルサイズのパッチだけだが、物体全体を認識するには、はるかに広い領域にわたって文脈を統合する必要がある。6.1.4 節で述べたように、畳み込み層の出力は**特徴マップ**（feature map）と呼ばれる（先ほどの垂直エッジマップ `Y` もその一例である）。特徴マップ内の任意の要素 x に対し、その**受容野**とは、順伝播において x の値に影響を与えうる入力画像（またはより前の層）の領域のことである。これは神経生理学の用語を借りたものである（Figure 6.1）。

6.2.4.1 層の積み重ね

Figure 6.7 では、網掛けされた出力 19 は 2×2 の受容野を持ち、$\mathbf{X}$ の左上の 4 ピクセルをカバーしている。ここで、**2 層目**の 2×2 畳み込み層を、2×2 の出力 $\mathbf{Y}$ の上に重ねて単一のスカラー z を生成してみよう。z は $\mathbf{Y}$ の 4 要素すべてに依存し、それらは合わせて $\mathbf{X}$ の 9 要素すべてに依存する。層を積み重ねることで、深さとともに受容野が滑らかに広がっていく。

では、どのくらいの速さで広がるのだろうか。各 $k \times k$ 畳み込み層は、その入力の 1 要素よりも $k-1$ ピクセルだけ遠くまで届くため、L 個の層（カーネルサイズ $k_1, \ldots, k_L$）を積み重ねると、受容野の 1 辺の長さは次のようになる。

$$r = 1 + \sum_{i=1}^{L} (k_i - 1). \tag{6.11}$$

連続する L 個の 3×3 層を積み重ねた場合、各層が 2 ピクセルずつ範囲を広げ、$(2L+1)\times(2L+1)$ 個の入力をカバーする。すなわち、2 層後には 5×5、3 層後には 7×7 となる（Figure 6.9）。この線形な拡大は、巨大なカーネルに頼る理由ではなく、むしろ大きな利点である。3 層の 3×3 層を積み重ねれば、同じ 7×7 の受容野を単一の 7×7 層よりもはるかに少ないパラメータ数（チ

ャネル対あたりの重みが $3 \times 9 = 27$ 個に対し 49 個）で達成しつつ、非線形性を 1 回ではなく 3 回挟み込めるからである。

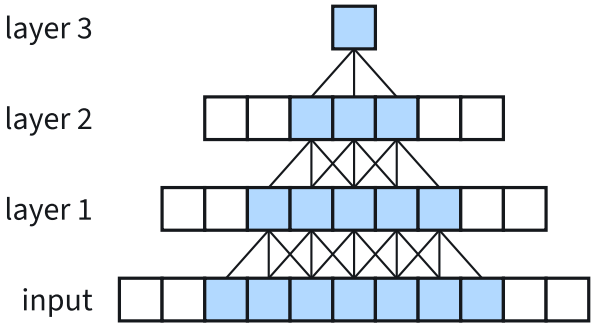


Figure 6.9: カーネルサイズ 3 の 1 次元の層を 3 層重ねた後の 1 ユニットの受容野。網掛けのセルは最上部のユニットが依存するユニットであり、各線はそれぞれのユニットとその入力を結んでいる。層ごとに受容野が 2 ずつ広がり、7 個の入力に達する。

6.2.4.2 有効受容野

ただし、r が測っているのは**理論的**受容野（theoretical receptive field）、すなわち出力ユニットへの**何らかの**経路を持つすべての入力ピクセルの範囲であり、各ピクセルが実際にどれほど強く寄与するかは表していないことに留意したい。入力勾配を直接計測することで、Luo et al. (2016) は、未訓練のネットワークと訓練済みのネットワークの両方において、**有効**受容野（effective receptive field）が中心から離れるにつれてほぼガウス型の釣鐘曲線を描いて急激に減衰することを示した。深い層のユニットは理論上は画像の広い範囲を見渡せるものの、その感度は中心付近に強く集中しているのである。

経路の数を数えれば、この釣鐘型の形状を説明できる。受容野の端にある入力は単一の結合の連鎖を通ってのみ出力ユニットに到達するのに対し、中央の入力は中間の層を通る数多くの経路をたどることができる。Figure 6.10 は、Figure 6.9 の積層構造について経路数を数えたものである。最上部のユニットに至る経路は、中央の入力では 7 本あるのに対し、両端の入力ではそれぞれ 1 本しかない。すべての重みが同程度の大きさであれば、ある入力の影響力は経路の数とともに大きくなる。

この基本的な層には、実用上の課題がまだ 2 つ残っている。パディングなしの畳み込みを繰り返すと空間次元が縮小し、かつ境界付近のピクセルがあまり使われないこと、そして単一のカーネルでは 1 つのパターンしか検出できないことである。空間サイズの調整と境界の扱いについては続く 6.3 節で、マルチチャネルの特徴マップについては 6.4 節で取り上げる。

6.2.5 まとめ

畳み込み層は、入力のすべての有効なウィンドウにわたってカーネルをスライドさせながら、学習されたカーネルとの 2 次元相互相関にスカラーのバイアスを加える。単純な 1×2 の有限差分カーネル `[1, -1]` は画像内のどこにある垂直エッジも検出でき、勾配降下法は入出力データだけからその重みを正確に復元する。これと同様に、深い CNN はフィルタの階層全体をエンドツーエンドで学習する。数学的な畳み込みはカーネルを反転させるが、カーネルをデータから学習する以上、実質的な違いはない。また、畳み込みは単一の行列積（**im2col**）として計算でき、L 個の畳み込み層を積み重ねると、各ユニットの受容野は層ごとに $k_i - 1$ ずつ線形に広がる。

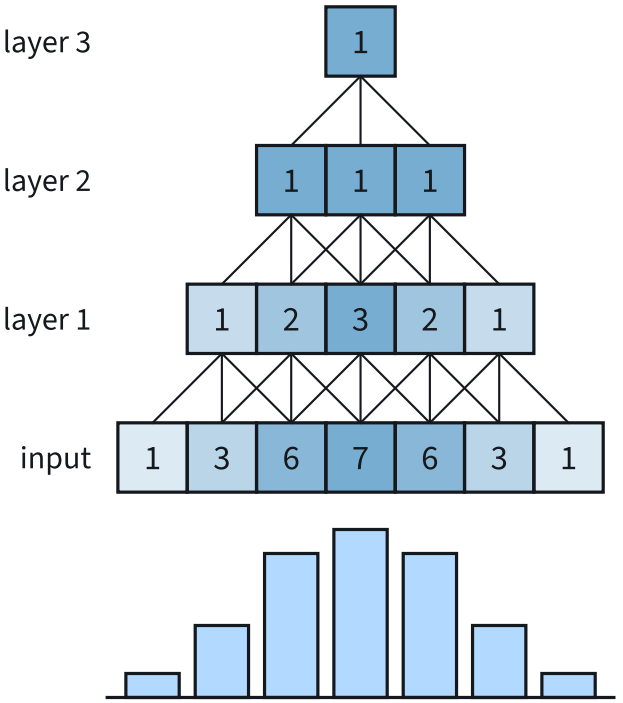


Figure 6.10: Figure 6.9（カーネルサイズ 3 の 1 次元の層 3 層）における各ユニットから最上部のユニットへの経路数。各セルの数値は、そのセルが入力を送る先のユニットの経路数の和になっている。色が濃いセルほど同じ層の他のセルより経路が多く、下部の棒グラフは入力行の経路数（1、3、6、7、6、3、1）をそのまま表している。

6.2.6 演習問題

1. [code] **手作業によるカーネル設計**。手作業でいくつかのカーネルを設計せよ。
    1. 方向ベクトル $\mathbf{v} = (v_1, v_2)$ が与えられたとき、$\mathbf{v}$ に直交するエッジ、すなわち方向 $(v_2, -v_1)$ のエッジを検出するエッジ検出カーネルを導出せよ。
    2. 二階導関数に対する有限差分作用素を導出せよ。画像内のどのような構造がこれに最も強く反応するか。
    3. d 階の導関数を得るために必要なカーネルの最小サイズはいくらか。
    4. ぼかしカーネルをどのように設計するか。また、そのようなカーネルを使いたくなるのはなぜか。
    5. 写真を 1 枚読み込み、1 チャネルに変換して約 100×100 ピクセルにダウンサンプリングせよ（`corr2d` は Python でループ処理を行うため）。エッジ検出器、ぼかしカーネル、および先鋭化カーネル「恒等変換の 2 倍からぼかしを引いたもの」を適用し、それぞれの結果を元の画像の隣に表示せよ。写真のエッジマップには、黒い帯のエッジマップでは見られなかった何が現れているか。

    Stanford CS231n の Assignment 2（ConvolutionalNetworks ノートブック）を改変。

2. [code] **帯行列としての畳み込み**。6.2.3 節のパッチ行列は入力を展開するものであった。これとは逆の構成として、カーネルを展開する。$\mathbf{X}$ を 4×4 の入力、$\mathbf{K}$ を 3×3 のカーネルとし、$\mathrm{vec}(\cdot)$ を行ごとの平坦化とする。
    1. 4×16 行列 $\mathbf{W}$ を、$\mathbf{W}\,\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{Y})$ を満たすように構成せよ。ここで $\mathbf{Y}$ は $\mathbf{X}$ と $\mathbf{K}$ の相互相関である。各行には非ゼロ要素がいくつあり、$\mathbf{W}$ には何種類の異なる値が現れるか。
    2. $\mathbf{W}$ を `K` からコード上で構築し、その行列積が `corr2d(X, K)` を再現することを確かめよ。
    3. $n \times n$ の入力と $k \times k$ のカーネルに対し、密配列として格納した $\mathbf{W}$ の要素数、$\mathbf{W}$ の非ゼロ要素数、およびパッチ行列の要素数を数えよ。それぞれの数は n に伴ってどのように増加するか。
    4. $\mathbf{W}$ のスパース性のパターンを保ったまま、各行に固有の k^2 個の値を持たせる。すると局所結合層が得られる。これは 4 階テンソル V（6.1 節の $[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_{a,b} [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}$ におけるもの）を、平行移動同変性によって重みを共有させる前に、$k \times k$ ウィンドウのオフセット (a, b) に制限したものにあたる。$n \times n$ の入力に対し、この層、同じ入出力形状を持つ全結合層、および畳み込みのパラメータ数をそれぞれ数えよ。

    CMU 11-785 の Homework 2 Part 1 を改変。

3. [code] **有効受容野**。●本節の理論的受容野 r はどの入力が出力に**影響を与えうる**かの範囲を与えるが、有効受容野は実際にどれだけ影響するかを測る (Luo et al. 2016)。L 個の単一チャネル 3×3 畳み込み層（パディングなし、バイアスなし、非線形性なし）を積み重ねたネットワークを 32×32 の入力上に構築せよ（6.2.2.3 節の組み込み層を用いること）。カーネルの各要素は標準正規分布から独立にサンプリングせよ。
    1. $L = 3$ とし、1 回サンプリングしたカーネルに対して、最終層の中央付近にある出力ユニットから入力まで単位勾配を逆伝播させ、入力勾配の大きさをプロットせよ。その台を 7×7 の理論的受容野と比較し、大きさがどこで最大になるかを答えよ。
    2. カーネルを 500 回サンプリングして入力勾配の二乗を平均し、これを $L = 3$ と $L = 10$ のそれぞれについて求めよ。平均した各マップを行方向に足し合わせ、得られた列方向のプロファイルを総和が 1 になるように正規化せよ。プロファイルの標準偏差と理論上の 1 辺の長さ $2L + 1$ との比は、深さに伴ってどのように変化するか。

3. Figure 6.10 は 1 次元における $L = 3$ の経路数を数えたものである。一般の L に対し、行方向に足し合わせた入力勾配の二乗の期待値が、L 個のベクトル $(1, 1, 1)$ の畳み込みに比例することを示せ。また、正規化されたプロファイルの標準偏差が L に伴ってどのように増大するかを導出せよ。

4. [code] **畳み込みの逆伝播。●Y** を入力 **X** とカーネル **K** の単一チャネル相互相関（`corr2d` が計算するもの）とし、ℓ を **X** と **K** に **Y** を介してのみ依存するスカラー損失とする。
   1. $\partial\ell/\partial\mathbf{K}$ と $\partial\ell/\partial\mathbf{X}$ を $\partial\ell/\partial\mathbf{Y}$ を用いて表す式を、4.3 節の誤差逆伝播法の方程式の形式で導出せよ。
   2. 導出した $\partial\ell/\partial\mathbf{K}$ の式を、損失 $\ell = \sum_{i,j} G_{ij} Y_{ij}$ に対して自動微分を使わずに実装せよ（ここで **G** は **Y** と同じ形状の固定されたランダム行列である）。ステップサイズ 10^{-2} の中心有限差分による勾配と小さなランダムな `X` および `K` 上で比較し、最大の相対誤差を報告せよ。
   3. 導出した $\partial\ell/\partial\mathbf{K}$ の式を、6.2.2.3 節の実験で自動微分が計算するカーネル勾配と照合して確認せよ。

Michael Nielsen『Neural Networks and Deep Learning』第 6 章、および Stanford CS231n の Assignment 2（ConvolutionalNetworks ノートブック）を改変。

6.3 パディング、ストライド、ダイレーション

パディングなしの $k \times k$ 畳み込みは、入力を各軸に沿って $k-1$ ピクセル縮小する（6.2 節）。5×5 の層を 10 層重ねると、240×240 の画像は 200×200 に縮小し、境界に沿って面積の 30% が失われる。しかし、ストライド 1 ですべての層を元の解像度に保つと、今度は逆の問題が生じる。各 3×3 層はユニットの受容野（receptive field）を 2 ピクセルしか広げないため、224×224 の画像全体をカバーするには 100 層以上が必要になるのである。さらに、セマンティックセグメンテーションのような密な予測（dense prediction）タスクでは、巨大なカーネルによる二乗オーダーのコストを避けつつ、広い視野と元の空間解像度（spatial resolution）の**両方**を確保したい。

3 つの幾何学的な制御が、これらのトレードオフを調整する。**パディング**は入力の周囲に枠を付加することで、カーネルの中心を端のピクセルに合わせ、空間次元を維持できるようにする。**ストライド**はウィンドウを 1 ピクセルより大きい幅で移動させることで、特徴マップ（feature map）をダウンサンプリングし、受容野の拡大を加速させる。そして**ダイレーション**はカーネルの重みの間隔を広げることで、小さなカーネルでも元の解像度のままより広い領域をカバーできるようにする。

6.3.1 パディング

6.3.1.1 境界ピクセル

層を重ねて画像の外周が削られていく以前に、境界付近のピクセルはそもそもほとんど活用されない。Figure 6.11 は、パディングなしの畳み込みにおいて各ピクセルを覆うスライディングウィンドウの数を示したものである。$k \times k$ カーネルでは、すべての境界から少なくとも $k-1$ ステップ離れた内部ピクセルは k^2 個のウィンドウに含まれるのに対し、辺上のピクセルは高々 k 個、角のピクセルはわずか 1 個のウィンドウにしか現れない。5×5 カーネルの場合、内部ピクセルは 25 個の出力位置に寄与するが、角のピクセルは 1 か所にしか寄与しないため、角の近くにある物体は次の層にほとんど反映されない。何らかの対処をしなければ、各層は入力の中心部に偏って情報を抽出することになる。

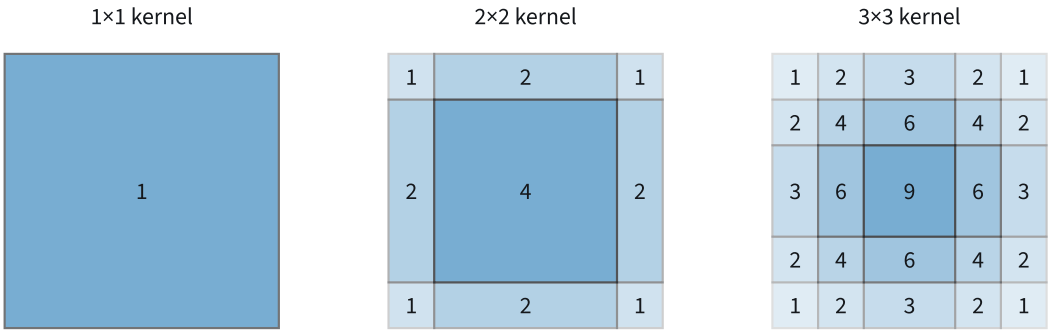


Figure 6.11: パディングなしで大きな入力上を 1×1、2×2、または 3×3 のカーネルがスライドするとき、各ピクセルを含むウィンドウの数。内部ピクセルは k^2 回、角のピクセルは 1 回覆われる。細長い各セルは、個数にかかわらず境界からその距離にあるすべてのピクセルを表す。

6.3.1.2 ゼロパディング

最も単純な解決策は、カーネルをスライドさせる前に、入力の周囲を余分な詰め物ピクセル（通常はゼロ）の枠で囲むことである。この手法を**パディング**と呼ぶ。Figure 6.12 では、上下にゼロの行を 1 行ずつ、左右に列を 1 列ずつ追加することで、3×3 の入力を 5×5 のグリッドへと拡張している。これにより、2×2 カーネルによって 4×4 の出力が得られる（パディングなしでは 2×2 にとどまる）。網掛けされた左上の出力要素は、パディングされた 3 個のゼロと元の入力の角の要素を組み合わせたものであり、$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$ となる。

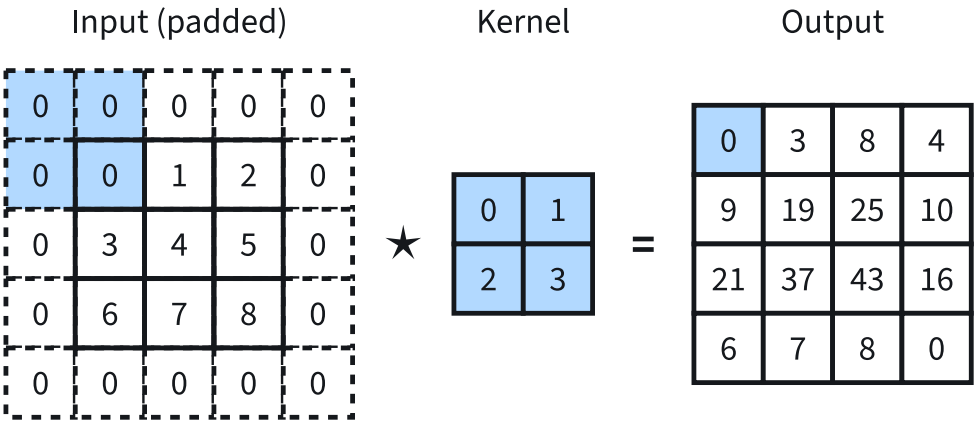


Figure 6.12: 四方に 1 ピクセルのゼロパディングを施した 2 次元相互相関。

深層学習フレームワークの慣例に従い、パディングは片側あたりの大きさで数える。すなわち、入力の上に p_h 行、下に p_h 行を追加し、左右にはそれぞれ p_w 列を追加する。このとき、パディング後の入力の形状は $(n_\mathrm{h} + 2p_\mathrm{h}) \times (n_\mathrm{w} + 2p_\mathrm{w})$ となり、出力の形状は次のようになる。

$$(n_\mathrm{h} + 2p_\mathrm{h} - k_\mathrm{h} + 1) \times (n_\mathrm{w} + 2p_\mathrm{w} - k_\mathrm{w} + 1). \tag{6.12}$$

上下どちらの側であってもパディングを 1 行追加するごとに出力は 1 行増え、1 列追加するごとに出力は 1 列増える。カーネルによって $k_\mathrm{h} - 1$ 行と $k_\mathrm{w} - 1$ 列が削られるため、$p_\mathrm{h} = (k_\mathrm{h} - 1)/2$

および $p_w = (k_w - 1)/2$ と設定すれば、入力の高さと幅がそのまま維持される。実務ではこれが圧倒的に最もよく使われる選択である。形状を維持する層を積み重ねれば、どの深さでも同じ空間解像度が保たれるため、ネットワークアーキテクチャの設計や考察がはるかに容易になる。

$k - 1$ を 2 で割ることから、CNN でほぼ常に 1、3、5、7 といった奇数のカーネルサイズが使われ、それに 0、1、2、3 のパディングが組み合わされる理由も説明できる。奇数サイズのカーネルには明確な中心ピクセルが存在するため、対称なパディングを行えば、出力 `Y[i, j]` を計算するウィンドウの中心が入力ピクセル `X[i, j]` の真上に置かれ、入力と出力のグリッドの位置が揃ったまま保たれる。一方、偶数サイズのカーネルでは $k_h - 1$ が奇数になるため、空間次元を維持するには非対称なパディング（片側に他方より 1 行多く追加すること）が必要になり、出力グリッドが入力に対して半ピクセルずれてしまう。

これらの出力形状の公式をコードで確認するため、まず小さなヘルパ関数を定義する。組み込みの畳み込み層（convolutional layer）はバッチ軸とチャネル軸を持つ 4 次元の入力を前提としているため、`comp_conv2d` は 2D 入力の先頭にサイズ 1 の次元を 2 つ挿入して層を適用し、結果からそれらの余分な次元を取り除く。

```
def comp_conv2d(conv2d, X):
    """Apply a convolutional layer to a single-channel 2D input."""
    Y = conv2d(X.reshape((1,) + X.shape + (1,)))  # input (1, h, w, 1)
    return Y.reshape(Y.shape[1:3])
```

3×3 カーネルの場合、形状を維持するパディングには $p = (3 - 1)/2 = 1$（コードでは `padding=1`）が必要である。この層を 8×8 の入力に適用すると、空間形状は 8×8 のまま変化しないはずである。

```
X = jax.random.uniform(d2l.get_key(), shape=(8, 8))
conv2d = nnx.Conv(1, 1, kernel_size=(3, 3), padding=1,
                  rngs=nnx.Rngs(d2l.get_key()))
comp_conv2d(conv2d, X).shape
```

```
(8, 8)
```

`nnx.Conv` のデフォルトは `padding='SAME'` であり、このほかに `'VALID'`、整数（または軸ごとのタプル）、`'REFLECT'`、`'CIRCULAR'` も受け取る。入力はチャネルラスト形式（`batch, height, width, channel`）である。

6.3.1.3 パディングの値

ゼロパディングは、フレームの外側が黒いピクセルで囲まれていると暗黙のうちに仮定している。より形式的に言えば、パディングとは**境界条件**（boundary condition）、すなわち観測値が存在しない画像境界の外側に値を与える規則を指定するものである。ゼロパディングが出力にどう影響するかを見るため、すべての要素が 1 である一様な 5×5 画像の四方に 1 ピクセルのゼロをパディングし、すべての要素が 1 の 3×3 カーネルとの相互相関を 6.2 節の `corr2d` で計算してみよう。このとき、出力の各要素は、ウィンドウ内に含まれる本来の画像ピクセルの数を表す。

```
img = d2l.ones((5, 5))
d2l.corr2d(jnp.pad(img, 1), d2l.ones((3, 3)))
```

```
Array([[4., 6., 6., 6., 4.],
       [6., 9., 9., 9., 6.],
```

```
       [6., 9., 9., 9., 6.],
       [6., 9., 9., 9., 6.],
       [4., 6., 6., 6., 4.]], dtype=float32)
```

内部の出力は 9 であるのに対し、辺では 6、角では 4 に低下する。元の画像が完全に一様であっても、層にはその周囲に暗い枠があるように見えているのである。この人工的な枠は後続の層を通じて内側へと伝播し、暗黙的な位置の手がかりとして学習に利用されうる。同時に、境界のアーティファクトによって端の近くに死角が生じ、ネットワークの反応が弱くなることもある (Alsallakh et al. 2020)。ゼロを導入しない代表的な代替手法が 2 つある。**反射パディング**（reflect padding）は境界を軸にピクセルを反転させ、**複製パディング**（replicate padding）は境界の値を外側へ繰り返す。どちらも定数画像を定数出力に写すため、境界の暗転が生成ピクセルに直接現れてしまう画像生成や画像復元のネットワークでよく使われる。しかし、分類ネットワークでは依然としてゼロパディングが標準的な選択肢である。単純であるうえ、パディング済みの大きなテンソルを別途確保することなく高速なカーネル内で暗黙的に処理できるからである。

6.3.2 ストライド

6.3.2.1 ストライド付き畳み込み

224×224 の画像をネットワーク全体にわたって元の解像度のまま保持すると、計算コストが高くつくうえ、大域的なコンテキストを集約するのにも時間がかかる。1 層あたり各チャネルに 50,176 個の空間位置が含まれ、各 3×3 畳み込みがユニットの視野を 2 ピクセルしか広げないため、1 つのユニットが画像全体を見渡せるようになるまでに 100 層以上が必要になるからである。畳み込みウィンドウをより大きな刻み幅で入力上に移動させれば、空間解像度を下げるとともに大域的なコンテキストを得るのに必要な深さも抑えられ、両方の問題を一挙に解決できる。

ここまでは、スライディングウィンドウを各行・各列に沿って 1 ピクセルずつ進めてきた。この高さ方向および幅方向のステップサイズを**ストライド**と呼ぶ。縦方向のストライドを s_h、横方向のストライドを s_w とすると、層は s_h 行ごと、s_w 列ごとにのみカーネルを評価するため、出力要素の数はおよそ $s_\mathrm{h}s_\mathrm{w}$ 分の 1 に減る。Figure 6.13 は、Figure 6.12 のパディング済み入力に対して縦方向のストライドを 3、横方向のストライドを 2 とした例を示している（2 つの軸を区別しやすいよう非対称な値を選んでいる）。網掛けされた各出力は、それを生成したウィンドウと同じ色で表されている。青い要素は $0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$ であり、最初のウィンドウから右に 2 列ずらした位置から得られる。橙色の要素は $0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$ であり、下に 3 行ずらしたウィンドウから得られる。さらに右へ 2 列進めるとウィンドウがパディング済み入力の右端をはみ出してしまうため、1 行あたりの出力は 2 個となる。

一般の場合の出力の高さを求めるため、ウィンドウがパディング済み入力の第 0 行から始まり、パディング済みの $n_\mathrm{h} + 2p_\mathrm{h}$ 行の内部に完全に収まらなければならない点に注目しよう。有効な開始行は $0, s_\mathrm{h}, 2s_\mathrm{h}, \ldots$ であり、上限は $n_\mathrm{h} + 2p_\mathrm{h} - k_\mathrm{h}$ となる。幅についても同様に考えると、出力の形状は次の式で与えられる。

$$\left(\left\lfloor \frac{n_\mathrm{h} + 2p_\mathrm{h} - k_\mathrm{h}}{s_\mathrm{h}} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{n_\mathrm{w} + 2p_\mathrm{w} - k_\mathrm{w}}{s_\mathrm{w}} \right\rfloor + 1\right). \tag{6.13}$$

ここで、床関数の項は最初の配置の後にウィンドウが進めるステップ数を表し、$+1$ はその開始位置の分である。Figure 6.13 に当てはめて確認すると、$\lfloor 3/3 \rfloor + 1 = 2$ 行、$\lfloor 3/2 \rfloor + 1 = 2$ 列となり、$s_\mathrm{h} = s_\mathrm{w} = 1$ と置けば先ほどのパディングの公式に戻る。ストライドを形状維持パディング（$2p_\mathrm{h} = k_\mathrm{h} - 1$）と組み合わせると、出力の高さは $\lfloor (n_\mathrm{h} - 1)/s_\mathrm{h} \rfloor + 1 = \lceil n_\mathrm{h}/s_\mathrm{h} \rceil$ に簡約される。すなわち、ストライド s は空間解像度を s で割り、端数を切り上げる。

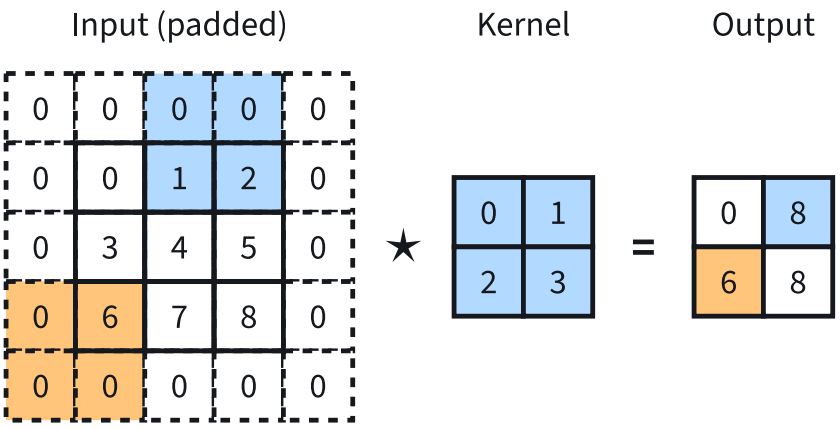


Figure 6.13: 高さ方向に 3、幅方向に 2 のストライドを用いた相互相関。網掛けされた各出力は、それを生成するウィンドウと同じ色になっている。

ResNet 族（7.4 節）全体を通して使われている非常に一般的なダウンサンプリング層は、パディング 1、ストライド 2 の 3×3 畳み込みである。先ほどの 8×8 の入力に適用すると、両方の空間次元が半分になり 4×4 となる。

```
conv2d = nnx.Conv(1, 1, kernel_size=(3, 3), padding=1, strides=2,
                  rngs=nnx.Rngs(d2l.get_key()))
comp_conv2d(conv2d, X).shape
```

```
(4, 4)
```

高さと幅で同じカーネルサイズ、パディング、ストライドを共有する必要はなく、出力形状の公式は各軸に独立して適用される。

ストライドがカーネルサイズに等しく、パディングがゼロである場合は重要な特殊ケースとなる。このとき、ウィンドウは入力を重なりのないタイルへと分割する。たとえば、ストライド 16 の 16×16 畳み込みは、224×224 の画像を 14×14 のパッチのグリッドに分割し、各パッチをチャネル出力のベクトルへ射影する。この**パッチ化**（patchify）ステムは、ビジョン Transformer が画像をトークンの系列に変換する仕組みであり、近年のいくつかの CNN も同じ演算から始まる。

高さと幅で同じ設定を共有する場合は、添字を省略して単に p および s と書く。特に断らない限り、畳み込み層のデフォルトはストライド 1、パディング 0（$p = 0$）である。

ただし、`nnx.Conv` ではデフォルトが `padding='SAME'` である。

6.3.2.2 ストライドと受容野

6.2 節で見たように、ストライド 1 では各 $k \times k$ 層が各軸に沿って受容野を $k - 1$ ピクセルずつ広げるため、深さに対して線形にしか拡大しない。ストライドはこの拡大を劇的に加速させる。ストライド 2 の層を通過すると、出力の隣接する要素は元の画像で 2 ピクセル離れた位置に対応する。そのため、後続の 3×3 カーネルは、受容野を 2 ピクセルではなく $(3-1) \cdot 2 = 4$ 入力ピクセル広げる。2. より一般に、第 i 層への入力における 1 ステップは元の画像の $\prod_{j<i} s_j$ ピクセ

ルにまたがるため、L 層のスタック（各層のカーネルサイズ k_i、ストライド s_i）の最上位にあるユニットの受容野の幅は次の式で与えられる。

$$r = 1 + \sum_{i=1}^{L} (k_i - 1) \prod_{j=1}^{i-1} s_j, \tag{6.14}$$

ここで、$i = 1$ に対する空積は 1 と定義する。各層は、元の画像のピクセル単位で測ったカーネルの広がり分だけ寄与する。すべてのストライドを 1 に設定すると各項は $k_i - 1$ になり、6.2 節で導いた線形の式に戻る。この公式にパディングが含まれないことに注目しよう。パディングはどの出力位置が存在するかを決めるものであり、各ユニットが観測するパッチの広さを変えるものではないからである。3×3 畳み込みの 3 層スタックを 2 通り比較すると、ストライド $(1, 1, 1)$ では $r = 1 + 2 + 2 + 2 = 7$ であるのに対し、ストライド $(2, 2, 1)$ では到達範囲が 2 倍以上の $r = 1 + 2 + 4 + 8 = 15$ に広がる。

各ストライドはそれより上位にあるすべての層の実効的な到達範囲を定数倍するため、ダウンサンプリングが続く限り受容野は幾何級数的に拡大する。ストライド 2 のステージを 5 回経ると（224×224 の入力を扱う画像分類器（image classifier）で典型的な構成である）、特徴マップ上の 1 ステップは 32 入力ピクセルにまたがるようになり、最上位のユニットはわずか数層で画像全体を見渡せるようになる。

6.3.3 ダイレーション

6.3.3.1 ダイレーション畳み込み

ストライド付き畳み込み（strided convolution）は、空間解像度を犠牲にして広い受容野を得る。しかしタスクによっては、広いコンテキストとすべての入力ピクセルに対する出力の両方が求められる。単にカーネルを大きくすれば視野は広がるものの、パラメータ数（parameter count）と計算コストがカーネルサイズの二乗で増大してしまう。パディングとストライドがカーネルを**どこに**配置するかを制御するのに対し、ダイレーションは**カーネル自体の間隔**を変化させる。すなわち、**ダイレーション畳み込み**（dilated convolution）は隣接する重みの間に隙間を挿入し、重みを d ピクセル間隔で評価する (Yu and Koltun 2016)。ここで d は**ダイレーション**率である。$k \times k$ カーネルは、ダイレーション d を用いても学習可能な重みを k^2 個しか含まないが、次の式で与えられる**実効カーネルサイズ**（effective kernel size）を持つ。

$$k' = k + (k - 1)(d - 1) = d(k - 1) + 1 \tag{6.15}$$

これは各軸に沿ったピクセル数であり、連続するタップの間で $d - 1$ ピクセルをスキップする。$d = 1$ と設定すれば通常の密な畳み込みに戻る。Figure 6.14 は、3×3 カーネルにおける $d = 1$ と $d = 2$ を対比したものである。ダイレーション 2 では、1 ピクセルおきにサンプリングすることで同じ 9 個の重みが 5×5 の領域をカバーしつつ、出力位置ごとに必要な乗算は依然として 9 回だけで済む。

出力次元に関しては、ダイレーションを施したカーネルはサイズ k' の密なカーネルとまったく同じように振る舞う。ストライド付きの公式において k を $k' = d(k - 1) + 1$ に置き換えると、任意のパディング、ストライド、ダイレーションに対する軸方向の出力サイズを表す単一の統一式が得られる。

$$\left\lfloor \frac{n + 2p - d(k - 1) - 1}{s} \right\rfloor + 1, \tag{6.16}$$

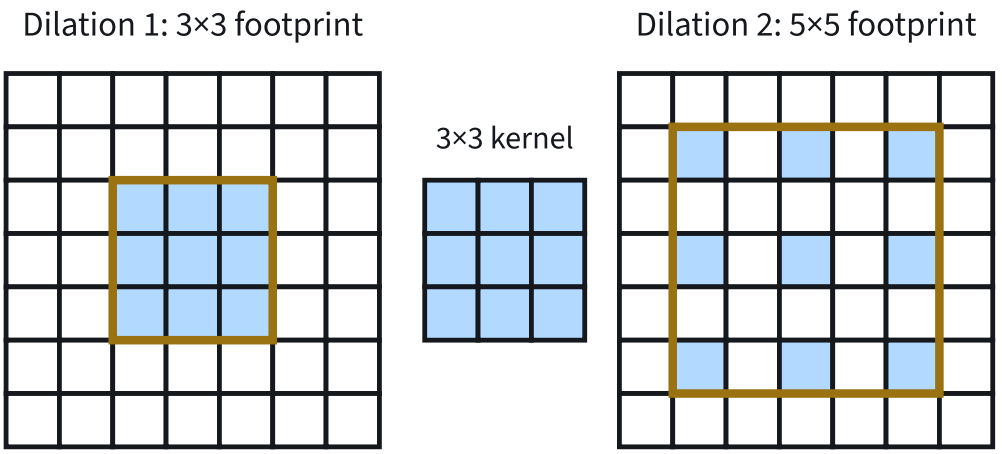


Figure 6.14: ダイレーション 1 およびダイレーション 2 における 3×3 カーネル。ダイレーション 2 では、9 個の重みが 5×5 の領域に広がる。

この式は、それぞれのパラメータ n、k、p、s、d を用いて高さと幅に独立して適用される。$d = 1$ と置けばストライド付きの公式が再現され、さらに $s = 1$ かつ $p = 0$ と置けば最初の $n - k + 1$ に戻る。したがって、先ほどの 8×8 の入力に対し、ダイレーション 2 の 3×3 カーネルは 4×4 の出力形状を生成し、これは密な 5×5 カーネルの出力形状と一致するはずである。

```
conv2d = nnx.Conv(1, 1, kernel_size=(3, 3), padding='VALID',
                  kernel_dilation=2, rngs=nnx.Rngs(d2l.get_key()))
conv5 = nnx.Conv(1, 1, kernel_size=(5, 5), padding='VALID',
                 rngs=nnx.Rngs(d2l.get_key()))
comp_conv2d(conv2d, X).shape, comp_conv2d(conv5, X).shape
```

```
((4, 4), (4, 4))
```

6.3.3.2 ダイレーションと受容野

ストライド 1 では、ダイレーションを用いた層は受容野を $k' - 1 = d(k-1)$ ピクセル広げるため、各 k_i を実効サイズ k'_i に置き換えれば Equation 6.14 がそのまま成り立つ。各層でダイレーション率を $d_i = 1, 2, 4, \ldots, 2^{L-1}$ のように倍増させていくと、L 層の 3×3 カーネルからなるスタックは幅 $1 + 2(1 + 2 + \cdots + 2^{L-1}) = 2^{L+1} - 1$ の受容野を達成する。ストライド付きスタックとまったく同様に受容野は深さに対して指数関数的に拡大するが、すべての層で入力の空間解像度がそのまま維持される。

Figure 6.15 は、これら 2 つの戦略を並べて示したものである。パネル (a) はストライド $(2, 2, 1)$ のストライド付きスタックを示しており、3 つの層が最上位ユニットの視野にそれぞれ 2、4、8 ピクセルを加え、合計 15 入力ピクセルに達する。パネル (b) はストライド 1 とダイレーション $(1, 2, 4)$ を用いてまったく同じ 15 ピクセルの受容野を実現しており、最上位ユニットと入力を同一の接続パターンで結んでいる。唯一の違いは空間的な密度である。ストライド付きスタックが空間位置を 4 つごとにしか保持しないのに対し、ダイレーションスタックはすべての位置を評価する。実際、パディングなしで重みを共有すれば、各層の後に要素ごとの非線形性を適用した場合であっても、ストライド付きスタックの出力はダイレーションスタックの出力を 4 要素ごとに取り出したものにほかならない。このため、ダイレーション畳み込みはセマンティックセグメンテ

ーションのような**密な予測**タスクで特に威力を発揮する。こうしたタスクでは、周囲の広いコンテキストを参照しつつ、すべてのピクセルにクラスラベルを割り当てなければならないからである (Yu and Koltun 2016)。トレードオフとなるのは計算量とメモリであり、すべての層が元の解像度の特徴マップを処理しなければならない。もう 1 つの選択肢は、ストライドによって一度ダウンサンプリングし、後から 32.10 節（英語版）で扱う転置畳み込み（transposed convolution）のような学習可能なアップサンプリング層を用いて空間解像度を復元する方法である。

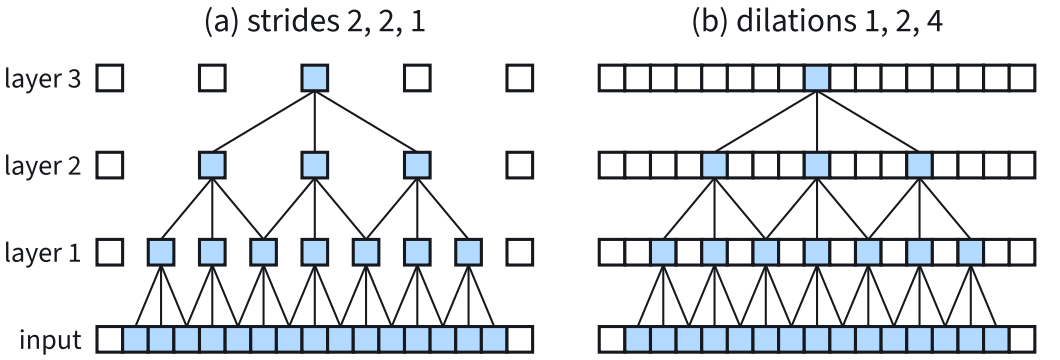


Figure 6.15: 1 次元においてカーネルサイズ 3 の層を 3 層重ねたときの、最上位ユニットの受容野。各セルは各層が計算する位置、網掛けのセルは最上位ユニットが依存する位置、線はカーネルのタップを表す。(a) ストライド 2、2、1 の場合、第 1 層は 2 つごとの位置を保持し、第 2 層と第 3 層は 4 つごとの位置を保持する。(b) ストライド 1、ダイレーション 1、2、4 の場合、すべての層が全位置を保持する。どちらの最上位ユニットも同じ 15 個の入力ピクセルを参照する。

本節のここまでの例はすべて、単一チャネルの入力に対して単一のカーネルを作用させるものであった。実世界の画像は複数の色チャネルを含み、実用的な隠れ層（hidden layer）は数十から数百もの異なるパターンを同時に検出する。6.4 節では、この両方の制約を取り除く。

6.3.4 まとめ

パディングを行わない場合、畳み込みは入力を各軸に沿って $k-1$ ピクセル縮小させ、境界ピクセルを内部ピクセルよりはるかに少ないウィンドウでしかサンプリングしない。奇数サイズのカーネルの両側に $(k-1)/2$ ピクセルのパディングを追加すれば、入力の形状が維持され、各出力の中心が対応する入力ピクセルの上に保たれる。ただし、ゼロパディングは深い層が検出しうる境界条件を導入する。ストライド $s > 1$ を用いると s 番目ごとのウィンドウ位置のみが評価され、空間解像度と計算量が各軸あたりおよそ s 分の 1 に減る一方で、上位層の受容野は幾何級数的に拡大する。最後に、ダイレーション d はカーネルの重みを実効サイズ $d(k-1)+1$ の領域にわたって配置し、パラメータを増やすことなく元の解像度のまま受容野を指数関数的に拡大する。これら 3 つの仕組みはすべて、単一の出力形状公式 Equation 6.16 に集約される。

6.3.5 演習問題

1. [code] **ミラーパディング**。ミラーパディング（反射パディングとも呼ばれる）は、境界の要素を繰り返さずに境界を軸として要素を反転させることでテンソルを拡張する。たとえば、ベクトル $(1,2,3)$ の両側に 1 要素ずつパディングすると $(2,1,2,3,2)$ になる。
    1. 2 次元テンソルの四方に 1 要素ずつミラーパディングを行う処理を実装し、互いに異なる要素を持つ 3×3 テンソルに対して手計算で結果を確かめよ。

   2. ゼロパディングのかわりにミラーパディングを用いて 6.3.1.3 節の計算を繰り返し、その結果を説明せよ。
   3. 5×5 の傾斜（ランプ）データ $x_{ij} = j$ と、3×3 カーネル（すべての要素が $\frac{1}{9}$）を考える。内部では出力が入力に一致することを示せ。また、ゼロパディングおよびミラーパディングのもとで最初と最後の列の出力を計算し、この傾斜データを厳密に再現するようなパディングの方法を見つけよ。
2. **ダイレーションとグリッディング**。あるネットワークが、ストライド 1 の 3×3 畳み込みを 4 層積み重ねているとする。ダイレーションを倍増させるスケジュール $(1, 2, 4, 8)$ では、1 つの出力要素は 1 辺が 31 の受容野を持つ。これを一定のスケジュール $(4, 4, 4, 4)$ と比較せよ。
   1. 各カーネルをその実効サイズに置き換えた Equation 6.14 を用いて、一定のスケジュールにおける受容野を計算せよ。
   2. 各スケジュールについて、受容野内のどのピクセルに出力が実際に依存するかを特定し、その個数を数えよ。受容野内の疎な格子上のピクセルにしか出力が依存しないスケジュールは、**グリッディング**を起こすという。
   3. セグメンテーションのような密な予測タスクにおいて、グリッディングが有害となる理由を 1 つ挙げよ。
   4. ●ダイレーションを $d_1, \ldots, d_L$ とするストライド 1 の 3×3 畳み込みのスタックにおいて、出力が受容野内のすべてのピクセルに依存することを保証する条件を述べ、その条件が十分であることを示せ。
3. **ストライドとダイレーションの比較**。@fig-conv-rf-towers の 2 つの 1 次元スタックを、パディングなし、両スタックで共有されるサイズ 3 のカーネル $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$、および各層の後の要素ごとの非線形性のもとで考える。
   1. 各スタックの位置 i における出力を入力の関数として書き下し、ストライド付きスタックの位置 i における出力が、ダイレーションスタックの位置 $4i$ における出力に等しいことを示せ。
   2. 長さ n の入力に対し、境界の影響を無視して各スタックが実行する乗算の回数を数えよ。
   3. 最初の設問を一般化せよ。ストライド $s_1, \ldots, s_L$ を持つスタックを再現するには、ダイレーションをどのように設定すればよいか。
4. **ステムを通過する形状**。224×224 の画像が 4 つの層を通過するとする。すなわち、パディング 3・ストライド 2 の 7×7 畳み込み、パディング 1・ストライド 2 の 3×3 畳み込み、およびパディング 1・ストライド 1 の 2 つの 3×3 畳み込みである。
   1. すべての層について出力の高さを求めよ。
   2. Equation 6.14 を用いて、最終層のユニットの受容野を計算せよ。そのユニットは画像の幅のどれだけの割合を見ているか。
   3. 最初の 2 層を、パディングなし・ストライド 4 の単一の 4×4 畳み込み（本節で扱ったパッチ化層）に置き換える。改めて出力の高さと受容野を求めよ。第 1 層の隣接するウィンドウは依然として重なり合うか。

6.4 複数の入力チャネルと複数の出力チャネル

ここまでの畳み込みは、単一の 2 次元グリッドを単一の 2 次元グリッドへ写すものであった。しかし実際の画像や中間表現は、入出力の両側に複数の**チャネル**を持つ。たとえば RGB 写真は $3 \times h \times w$ のテンソルであり（等輝度で赤と緑が接する境界を検出するにはチャネル間の比較が必要になる）、隠れ層は何十、何百もの異なる特徴マップ（feature map）を並列に抽出する。

6.1 節で導出したように、c_{i} 個の入力チャネルと c_{o} 個の出力チャネルを持つ層は、出力チャネルごとに $c_{\mathrm{i}} \times k_{\mathrm{h}} \times k_{\mathrm{w}}$ のカーネルを 1 つずつ保持し、すべての入力チャネルにわたって相互相関をとってその結果を足し合わせる。本節では、多チャネル畳み込みを実装し、パラメータ数・FLOP・

活性化メモリのコスト（いずれも積 $c_\mathrm{i}c_\mathrm{o}$ に支配される）を整理したうえで、1×1 ボトルネック畳み込みを導入する。

6.4.1 複数の入力チャネル

まず入力側から見ていこう。入力が c_i 個のチャネルを持つとき、カーネルは各入力チャネルに対して $k_\mathrm{h} \times k_\mathrm{w}$ のフィルタを 1 つずつ用意する必要があり、その結果、形状 $c_\mathrm{i} \times k_\mathrm{h} \times k_\mathrm{w}$ の 3 階カーネルテンソルとなる。各入力チャネル $\mathbf{X}_c$ と対応するカーネルスライス $\mathbf{K}_c$ について 2 次元の相互相関（6.2 節、`corr2d` で計算）をとり、得られた c_i 個の中間グリッドを足し合わせる。

$$\mathbf{Y} = \sum_{c=1}^{c_\mathrm{i}} \mathbf{X}_c \star \mathbf{K}_c. \tag{6.17}$$

したがって、得られる 2 次元出力の各要素は、すべての入力チャネルにわたる同じ空間ウィンドウ上の重み付き和であり、$c_\mathrm{i}k_\mathrm{h}k_\mathrm{w}$ 個のスカラー積を合わせたものになる。Figure 6.16 に 2 つの入力チャネルに対するこの計算を示す。網掛けされた左上の出力要素は $(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$ に等しい。

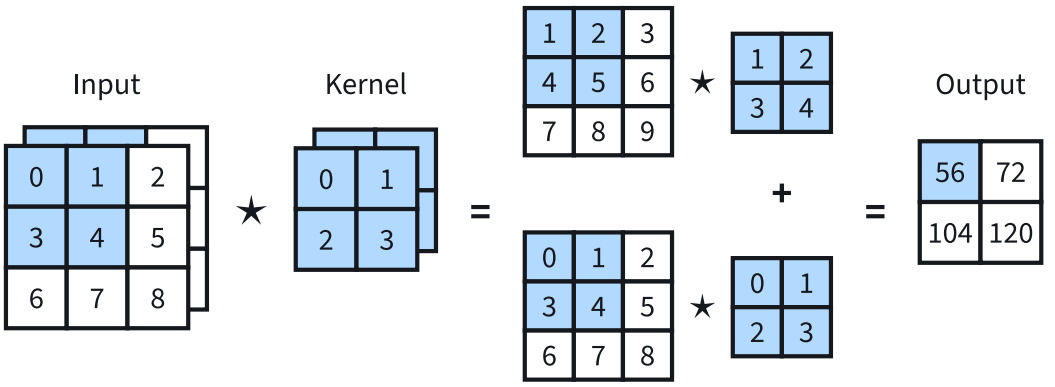


Figure 6.16: 2 つの入力チャネルに対する相互相関。チャネルごとに相互相関を計算し、それらを足し合わせる。

チャネル間で和をとることこそが、色のように異なるチャネルが担う情報をカーネルが比較できる理由である。例として、1×1 カーネルで赤チャネルの重みを $+1$、緑チャネルの重みを -1 としたものを考えてみよう。このカーネルは各ピクセルで「赤マイナス緑」を計算するため、明るさにかかわらず、赤い領域では正の値、緑の領域では負の値、無彩色の灰色の領域では 0 を出力する。隣接するピクセル間で符号が反転する場所では、赤の領域と緑の領域が接している。たとえ両領域の輝度がまったく同じで、グレースケールの明るさフィルタではエッジが一切見えない場合でも、この境界を検出できる。

多入力の相互相関の実装は、この数式をそのまま Python に書き下すだけである。チャネルごとに `corr2d` を 1 回呼び出し、その結果を足し合わせればよい。

```
def corr2d_multi_in(X, K):
    """Cross-correlate each channel of X with its channel of K and add."""
    return sum(d2l.corr2d(x, k) for x, k in zip(X, K))
```

Figure 6.16 の入力テンソルとカーネルテンソルを使って、この実装を検証しよう。左上の出力要素は 56 になるはずである。

```
X = d2l.tensor([[[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]],
                [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]])
K = d2l.tensor([[[0.0, 1.0], [2.0, 3.0]], [[1.0, 2.0], [3.0, 4.0]]])

corr2d_multi_in(X, K)
```

```
Array([[ 56.,  72.],
       [104., 120.]], dtype=float32)
```

6.4.2 複数の出力チャネル

どれほど多くの入力チャネルを読み込もうとも、単一の多入力カーネルが生成するのは 1 枚の 2 次元**特徴マップ**（6.1.4 節）だけである。異なる c_o 枚の特徴マップを同時に抽出するため、畳み込み層はそのようなカーネルを c_o 個保持する（各カーネルの形状は $c_\mathrm{i} \times k_\mathrm{h} \times k_\mathrm{w}$ である）。これらを新しい軸に沿って積み重ねると、形状 $c_\mathrm{o} \times c_\mathrm{i} \times k_\mathrm{h} \times k_\mathrm{w}$ の 4 階カーネルテンソルが得られ、出力チャネル o は次の式で与えられる。

$$\mathbf{Y}_o = \sum_{c=1}^{c_\mathrm{i}} \mathbf{X}_c \star \mathbf{K}_{o,c}, \quad o = 1, \ldots, c_\mathrm{o}. \tag{6.18}$$

ここで、各出力チャネルは固有の $k_\mathrm{h} \times k_\mathrm{w}$ スライス $\mathbf{K}_{o,c}$ を介して**すべての**入力チャネルと結合している。この密な全対全のチャネル結合こそが、この層の計算コストを押し上げる主な要因である。出力チャネルごとにスカラーのバイアス u_o を加えれば、6.1 節で導出した多チャネル畳み込み層が得られる。典型的な CNN アーキテクチャでは、ストライドやプーリングによって空間解像度が縮小するにつれてチャネル数を増やし、きめ細かい空間配置と引き換えに、各位置で共同して学習されたより豊かな特徴記述を得る。

Flax はカーネルをチャネルラストの $(k_\mathrm{h}, k_\mathrm{w}, c_\mathrm{i}, c_\mathrm{o})$ として格納する。

多出力チャネルの相互相関を実装するには、出力カーネルを順に取り出してそれぞれに `corr2d_multi_in` を適用し、得られた特徴マップを軸 0 に沿って積み重ねる。

```
def corr2d_multi_in_out(X, K):
    """Apply corr2d_multi_in once per output channel of K and stack."""
    return d2l.stack([corr2d_multi_in(X, k) for k in K], 0)
```

この関数をテストするため、`K`、`K + 1`、`K + 2` を積み重ねて 3 つの出力チャネルを持つカーネルテンソルを作ってみよう。

```
K = d2l.stack((K, K + 1, K + 2), 0)
K.shape
```

```
(3, 2, 2, 2)
```

`corr2d_multi_in_out` を 2 チャネルの入力 `X` と積み重ねた $3 \times 2 \times 2 \times 2$ のカーネル `K` に適用すると、形状 $(3, 2, 2)$ の 3 チャネル出力テンソルが得られる。

```
corr2d_multi_in_out(X, K)
```

```
Array([[[ 56.,  72.],
        [104., 120.]],

       [[ 76., 100.],
        [148., 172.]],

       [[ 96., 128.],
        [192., 224.]]], dtype=float32)
```

1 番目の出力チャネルは、カーネルが元の K のままであるため先ほどの結果と一致する。2 番目のチャネルも暗算で確かめられる。カーネルのすべての重みに 1 を加えると、各出力要素は、両方の入力チャネルにわたる対応ウィンドウ内の全要素の和だけ増加する。左上のウィンドウではその和が 20 なので、2 番目のチャネルの先頭要素は $56 + 20 = 76$ となる。

6.4.2.1 パラメータとコスト

チャネルを増やすとネットワークの表現力は高まるが、同時にメモリ使用量と計算量も急激に増大する。出力チャネルごとに 1 つのスカラーバイアスを含めると、畳み込み層が持つ要素数は次の式で表される。

$$c_{\mathrm{o}} c_{\mathrm{i}} k_{\mathrm{h}} k_{\mathrm{w}} + c_{\mathrm{o}} \tag{6.19}$$

これが学習可能なパラメータの数である。ネットワークの入力付近では、この数は控えめである。たとえば、3 つの RGB チャネルを 64 チャネルへ写す 3×3 の層（6.1 節）が持つパラメータは 1,792 個にすぎない。しかしネットワークの深い位置ではチャネル数が大きくなり、256 チャネルを 256 チャネルへ写す 3×3 の層では、589,824 個の重みと 256 個のバイアスが必要になる。

次に計算コストを考えると、各出力要素の計算には、その出力チャネルの各重みにつき 1 回の積和演算（multiply-add）が必要になる（合計 $c_{\mathrm{i}} k_{\mathrm{h}} k_{\mathrm{w}}$ 回）。空間サイズ $h_{\mathrm{o}} \times w_{\mathrm{o}}$、出力チャネル数 c_{o} の出力特徴マップ全体にわたってこの層が実行する演算回数は、次の式で与えられる。

$$h_{\mathrm{o}} w_{\mathrm{o}} \cdot c_{\mathrm{o}} c_{\mathrm{i}} k_{\mathrm{h}} k_{\mathrm{w}} \tag{6.20}$$

これだけの積和演算が実行される。56×56 の特徴マップに適用した場合、先ほどの 256 チャネルから 256 チャネルへの層は、画像 1 枚あたり $56^2 \cdot 589{,}824 \approx 1.8 \times 10^9$ 回の積和演算を実行する。パラメータ数と演算コストはいずれも積 $c_{\mathrm{i}} c_{\mathrm{o}}$ に比例するため、各層の幅を 2 倍にすると両コストとも 4 倍になる。また、同じ重みがすべての空間位置で再利用されるため、演算コストは出力面積 $h_{\mathrm{o}} w_{\mathrm{o}}$ にもそのまま比例する。

3 つ目のリソースコストは、訓練時に同等に重要でありながら見落とされがちな**活性化メモリ**（activation memory）である。逆伝播で勾配を計算するため、フレームワークは各層が出力した活性化値を保持しなければならず、訓練例 1 つあたり $h_{\mathrm{o}} w_{\mathrm{o}} c_{\mathrm{o}}$ 個の数値を保存する（5.7.3.2 節）。パラメータのメモリとは異なり、活性化メモリは**最も初期の**層で最大になる。そこではパラメータ数こそごくわずかだが、空間解像度が最も高いからである。たとえば、3 チャネルから 64 チャネルへ写して 224×224 の出力を生成する層は、訓練例 1 つあたり $224^2 \cdot 64 \approx 3.2 \times 10^6$ 個の活性化値をキャッシュする必要があり、これは 1,792 個のパラメータの約 2,000 倍もの数値にあたる。

6.4.3 1×1 畳み込み

一見すると、1×1 畳み込み（$k_\mathrm{h} = k_\mathrm{w} = 1$）は無意味に思えるかもしれない。空間ウィンドウが 1 ピクセルしか覆わないため、エッジやコーナーなど近傍の位置にまたがるパターンを一切検出できないからである。しかし複数のチャネルが存在するときには、先ほどの「赤マイナス緑」の色フィルタと同様に、きわめて有用な演算となる。各空間座標 (i, j) において、1×1 畳み込みは c_i 次元のチャネルベクトルに共有された $c_\mathrm{o} \times c_\mathrm{i}$ の重み行列を掛け、c_o 次元の出力ベクトルを生成する。言い換えれば、すべての空間位置で重みを共有しつつ、各ピクセルに独立して適用される全結合層（fully connected layer）として機能するのである。Figure 6.17 に、高さと幅を保ったまま 3 つの入力チャネルを 2 つの出力チャネルへ写す 1×1 畳み込みを示す。

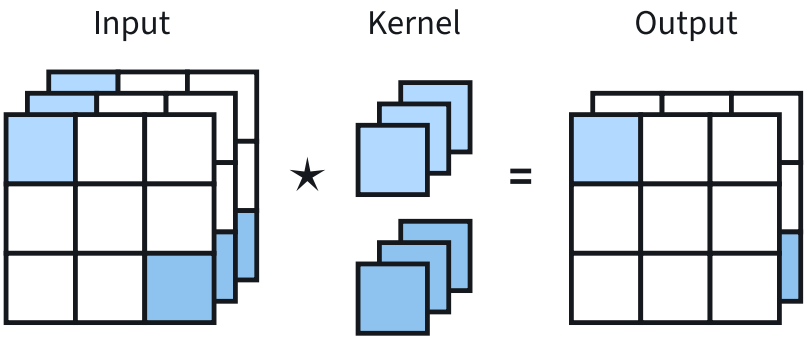


Figure 6.17: 3 つの入力チャネルと 2 つの出力チャネルを持つ 1×1 畳み込み。各出力要素は、同じ位置にある入力要素の線形結合である。

すべてのピクセルでまったく同じ線形変換が適用されるため、1×1 畳み込み全体は 1 回の行列積で計算できる。すなわち、入力を $c_\mathrm{i} \times hw$ の行列に変形し、左から $c_\mathrm{o} \times c_\mathrm{i}$ のカーネル行列を掛け、得られた出力を (c_o, h, w) の形状に戻せばよい。

```
def corr2d_multi_in_out_1x1(X, K):
    """A 1x1 convolution as one matrix product."""
    c_i, h, w = X.shape
    c_o = K.shape[0]
    Y = d2l.matmul(d2l.reshape(K, (c_o, c_i)), d2l.reshape(X, (c_i, h * w)))
    return d2l.reshape(Y, (c_o, h, w))
```

ランダムな入力を用いて、この行列積による定式化が `corr2d_multi_in_out` の結果と一致することを確かめよう。

```
X = jax.random.normal(d2l.get_key(), (3, 3, 3))
K = jax.random.normal(d2l.get_key(), (2, 3, 1, 1))
Y1 = corr2d_multi_in_out_1x1(X, K)
Y2 = corr2d_multi_in_out(X, K)
float(d2l.reduce_sum(d2l.abs(Y1 - Y2)))
```

```
1.0617077350616455e-06
```

期待どおり、2 つの結果は浮動小数点の丸め誤差の範囲内で一致する。

なぜ現代のアーキテクチャでは 1×1 畳み込みがこれほど広く使われているのだろうか。最大

の強みは、チャネル次元を低コストで調整し、高コストな空間畳み込みをより狭い表現の上で実行できるようにする点にある。先ほどの 256 チャネルから 256 チャネルへの 3×3 層に戻ろう。3×3 カーネルを 256 チャネルすべてに直接適用するかわりに、まず 1×1 畳み込みで 256 チャネルを 64 チャネルへ射影し、その 64 チャネルに対して 3×3 畳み込みを適用したのち、2 つ目の 1×1 畳み込みで 256 チャネルへ戻すことができる。3 つの層を合わせても使用する重みは $16{,}384 + 36{,}864 + 16{,}384 = 69{,}632$ 個にとどまり、$589{,}824$ 個と比べてパラメータ数と積和演算の両方で約 $8.5\times$ の削減となる。このような**ボトルネック**は中間での空間的な混合を 64 チャネルの部分空間に制限するが、ResNet（7.4 節）などのアーキテクチャは、はるかに深いネットワークを実現する引き換えとしてこのトレードオフを積極的に受け入れている。また、1×1 畳み込みの直後に活性化関数を置けば、各ピクセルで評価される小さな非線形 MLP となる (Lin et al. 2013)。非線形性がなければ、連続する線形な畳み込みは単一の線形変換に縮退してしまう。

どの入力チャネルを各出力チャネルに入力するかを制限することで $c_\mathrm{i}c_\mathrm{o}$ の因子を削減するアーキテクチャ上の仕組みは、ほかにも 2 つある。ResNeXt（7.4.5.1 節）で広く知られるようになった**グループ畳み込み**（grouped convolution）と、軽量なモバイルモデル（7.5.1.1 節）の基盤をなす**深さ方向分離可能畳み込み**（depthwise-separable convolution）である。これらについては次章で、それぞれを用いるアーキテクチャを構築しながら導入する。

なお、チャネルを導入しただけでは、1 つの基本的な性質は変わらないままである。すなわち、入力がずれると畳み込みの出力もやはりピクセル単位でそのまま移動する。6.5 節では、局所近傍を要約して小さな空間的ずれに対する耐性を獲得するために、プーリング層を導入する。

6.4.4 まとめ

多チャネル入力を扱うため、畳み込み層は各入力チャネルと固有の空間カーネルとの相互相関をとってその結果を足し合わせる。これにより、単一のフィルタで色のようなチャネル間の手がかりを統合できる。c_o 枚の出力特徴マップを生成するため、この層はそのような多入力カーネルを c_o 個積み重ねて形状 $c_\mathrm{o} \times c_\mathrm{i} \times k_\mathrm{h} \times k_\mathrm{w}$ のテンソルを構成する。これには $c_\mathrm{o}c_\mathrm{i}k_\mathrm{h}k_\mathrm{w} + c_\mathrm{o}$ 個のパラメータと $h_\mathrm{o}w_\mathrm{o}c_\mathrm{o}c_\mathrm{i}k_\mathrm{h}k_\mathrm{w}$ 回の積和演算（いずれも $c_\mathrm{i}c_\mathrm{o}$ に比例）を要し、さらに訓練時には訓練例 1 つあたり $h_\mathrm{o}w_\mathrm{o}c_\mathrm{o}$ 個の活性化値を保存しなければならない。1×1 畳み込みは各ピクセルで共有される全結合層として機能し、チャネルを混合してパラメータ効率・計算効率の高いボトルネックを構築することを可能にする。

6.4.5 演習問題

1. **畳み込みの合成**。ストライド 1、パディングなしで、間に非線形性を含まない 2 つの畳み込み層があり、それぞれのカーネルの形状を $c_\mathrm{m} \times c_\mathrm{i} \times k_1 \times k_1$ および $c_\mathrm{o} \times c_\mathrm{m} \times k_2 \times k_2$ とする。
    1. これら 2 層の合成が単一の畳み込みになることを証明し、そのカーネルの形状を求めよ。
    2. この 2 層の組が持つ重みの数と、単一の畳み込みが持つ重みの数を比較せよ。どのような中間幅 c_m のときに 2 層の組のほうが低コストになるか。
    3. 6.2 節で示したように、積み重ねた 3 つの単一チャネル 3×3 カーネルは 7×7 の受容野を持つ。すべての単一チャネル 7×7 カーネルが 3 つの 3×3 カーネルの合成として表せるかどうかを判定し、その理由を述べよ。
2. **3 つのコストと 3 つの層**。224×224 の RGB 画像を入力とするネットワークが、パディング 1 の 4 つの 3×3 畳み込みから始まるとする。すなわち、3 チャネルから 64 チャネル（ストライド 1）、64 チャネルから 128 チャネル（ストライド 2）、128 チャネルから 128 チャネル（ストライド 1）、そして 128 チャネルから 256 チャネル（ストライド 2）である。

1. 各層について、出力の形状、重みの数、画像 1 枚あたりの積和演算の回数、および逆伝播のために訓練時に保持しなければならない出力値の個数を求めよ。
2. 重みの数が最も多い層、積和演算の回数が最も多い層、保持する出力値の個数が最も多い層はそれぞれどれか。これらがすべて異なる 3 つの層になる理由を説明せよ。
3. 入力画像を 448×448 の画像に置き換える。3 つのコストのうちどれが変化し、何倍になるか。

3. [code] **多チャネルの im2col**。6.2.3 節のパッチ行列を複数の入力チャネルと出力チャネルに拡張せよ。本節の後半のセルで上書きされているため、Figure 6.16 の 2 チャネル入力 `X` と 6.4.2 節の 3 出力チャネルカーネル `K` を再構築せよ。形状 $c_\mathrm{i} \times k_\mathrm{h} \times k_\mathrm{w}$ の入力パッチをその軸順で平坦化したものを各行とするパッチ行列を作成し、同様に `K` も同じ順序で行を並べた $c_\mathrm{i} k_\mathrm{h} k_\mathrm{w} \times c_\mathrm{o}$ のカーネル行列に並べ替えよ。両者の積を出力の形状に並べ替えると `corr2d_multi_in_out(X, K)` が再現されることを確認せよ。

Simon Prince『Understanding Deep Learning』問題 10.6、10.7 より改作。

6.5 プーリング

畳み込み層は（ピクセル (i, j) にエッジがあるかといった）局所的な問いに答えるのに対し、分類器は画像全体に関する大域的な問いに答えなければならない。局所的な特徴マップから大域的な予測へと橋渡しをするには、広い空間的到達範囲（6.3.2.2 節）と、カメラの 1 ピクセルの手ぶれで予測が変わらないような微小な空間的シフトに対する頑健性（robustness）の両方が求められる。

プーリング層（pooling layer）は、学習可能なパラメータを追加することなくこの両方の目的を果たす。各チャネルの上で独立にウィンドウをスライドさせ、パッチごとにその**最大値**または**平均**によって要約するのである。小さなブロック内の最大値はピーク応答がそのブロック内にとどまる限り変化しないため、局所プーリングは小さなシフトに対する**耐性**（tolerance）をもたらす。一方、特徴マップ全体にわたる**グローバルプーリング**は、空間座標を潰してチャネルごとの単一のベクトルに集約する。

```
from functools import partial
```

6.5.1 最大プーリングと平均プーリング

畳み込み層と同様に、プーリング演算子は固定形状のウィンドウ（**プーリングウィンドウ**）を入力上でスライドさせ、各位置で単一のスカラー出力を計算する。しかし畳み込みとは異なり、プーリング層は学習可能なカーネルの重みを持たず、ウィンドウ内の要素に対して固定された決定論的な縮約を適用する。**最大プーリング**（maximum pooling、略して **max-pooling**）はウィンドウ内の最大の要素を返し、**平均プーリング**（average pooling）はそれらの算術平均を返す。

Figure 6.18 では、2×2 のウィンドウが 3×3 の入力上をストライド 1 で左から右、上から下へとスライドする。ウィンドウは各軸に沿って 2 つの位置に収まるため、出力の形状は 2×2 となり、各要素はそれぞれのウィンドウの最大値で与えられる。

$$\begin{aligned}
\max(0, 1, 3, 4) &= 4, \\
\max(1, 2, 4, 5) &= 5, \\
\max(3, 4, 6, 7) &= 7, \\
\max(4, 5, 7, 8) &= 8.
\end{aligned}$$

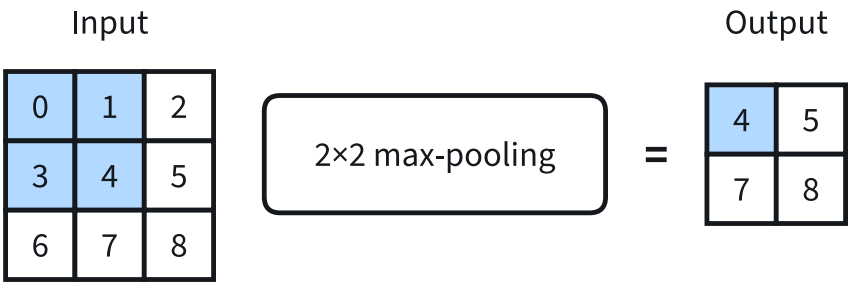


Figure 6.18: プーリングウィンドウの形状が 2×2 の最大プーリング。網掛け部分は最初の出力要素と、その出力の計算に使われる入力テンソルの要素を表す（$\max(0, 1, 3, 4) = 4$）。

9 個の入力要素のうち 5 個（0、1、2、3、6）は、どのウィンドウでも最大値にならないことに注目しよう。最大プーリングは各ウィンドウで最大となる要素のみに依存するため、これらの要素をわずかに摂動させても出力はまったく変化しない。同じスライディングウィンドウの構成は、任意の $k_{\text{h}} \times k_{\text{w}}$ プーリングウィンドウへそのまま一般化できる。

以下では、2 次元プーリングの順伝播を `pool2d` として実装する。その入れ子のループは `corr2d`（6.2 節）と同じ構造であり、重み付き内積を `max` または `mean` に置き換えている。また、次にダウンサンプリングを調べられるよう `stride` 引数も用意しておく。

```
def pool2d(X, pool_size, mode='max', stride=1):
    k_h, k_w = pool_size
    Y = jnp.zeros(((X.shape[0] - k_h) // stride + 1,
                   (X.shape[1] - k_w) // stride + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            window = X[i * stride: i * stride + k_h,
                       j * stride: j * stride + k_w]
            Y = Y.at[i, j].set(window.max() if mode == 'max'
                               else window.mean())
    return Y
```

`pool2d` を 3×3 テンソル（Figure 6.18）でテストすると、最大プーリングによって 4、5、7、8 が得られることが確認できる。

```
X = d2l.tensor([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
pool2d(X, (2, 2))
```

```
Array([[4., 5.],
       [7., 8.]], dtype=float32)
```

同じ入力に対して平均プーリングに切り替えると、4 つのウィンドウの平均である 2、3、5、6 を返す。

```
pool2d(X, (2, 2), 'avg')
```

```
Array([[2., 3.],
       [5., 6.]], dtype=float32)
```

6.5.2 小さなシフトに対する耐性

プーリングは空間的なシフトをどの程度まで吸収できるのだろうか。最も単純な設定で試してみよう。ストライド 2 の 2×2 プーリング層は、入力を重なりのない 2×2 のブロックに分割し、各ブロックを単一の値に要約する。（孤立したエッジにエッジ検出器が反応したときのように）1 という非ゼロの応答をただ 1 つだけ含む 4×4 の特徴マップを考え、その応答を行 1 に沿って 1 ピクセルずつスライドさせてみよう。以下では、連続する 3 つの列位置にわたって最大プーリングと平均プーリングの出力を比較する。

```
def single_response(i, j, n=4):
    """An n x n feature map that is 1 at (i, j) and 0 elsewhere."""
    return d2l.tensor([[float(r == i and c == j) for c in range(n)]
                       for r in range(n)])

for j in range(3):
    Z = single_response(1, j)
    print(f'response at (1, {j}): max',
          d2l.numpy(pool2d(Z, (2, 2), stride=2)).tolist(),
          'avg', d2l.numpy(pool2d(Z, (2, 2), 'avg', stride=2)).tolist())
```

```
response at (1, 0): max [[1.0, 0.0], [0.0, 0.0]] avg [[0.25, 0.0], [0.0, 0.0]]
response at (1, 1): max [[1.0, 0.0], [0.0, 0.0]] avg [[0.25, 0.0], [0.0, 0.0]]
response at (1, 2): max [[0.0, 1.0], [0.0, 0.0]] avg [[0.0, 0.25], [0.0, 0.0]]
```

応答を列 0 から列 1 へシフトさせても、両方の位置が同じ 2×2 ブロック内に収まるため、どちらのプーリング出力もまったく変化しない。さらに 1 ピクセル進めて列 2 へシフトさせるとブロック境界をまたぐため、プーリングされた応答は隣接する出力セルへとジャンプする。したがって、水平方向の 1 ピクセルのシフトは、応答が元のブロック内にとどまる場合（全開始列の半分）には吸収され、そうでない場合には出力の位置を 1 つずらす。ゆえに局所プーリングは、表現にある種の小さなシフトに対する**耐性**を持たせるのであり、厳密に不変にするわけではない。Figure 6.19 に、4 つの列位置すべてにわたる最大プーリングの出力を示す。

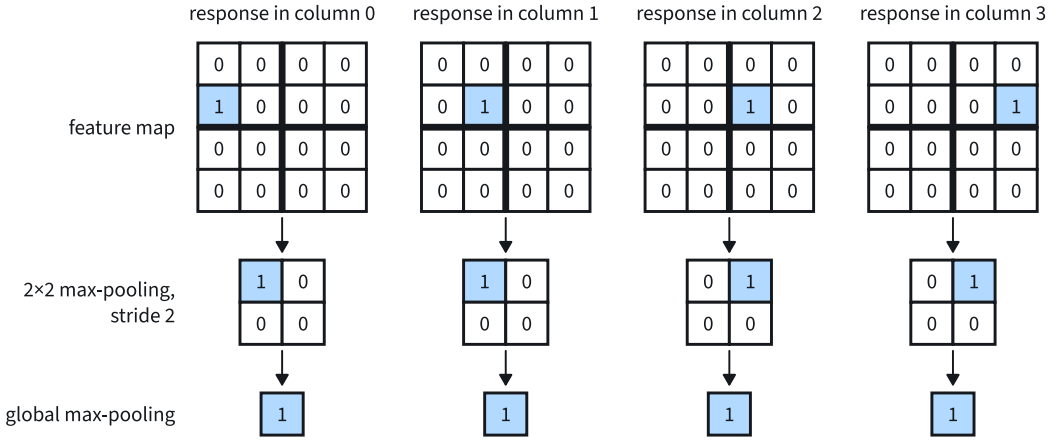


Figure 6.19: 4×4 の特徴マップの 4 つの列を移動する単一の応答（上段）、ストライド 2 による 2×2 最大プーリング後のマップ（中段）、およびその大域的最大値（下段）。太線はプーリングブロックを表す。プーリングされた応答は、元の応答がブロック内にとどまる間は同じ位置に保たれ、次のブロックへ境界を越えるとジャンプするが、大域的最大値は決して動かない。

また、最大プーリングと平均プーリングとでブロックの要約の仕方が異なることにも注目しよう。最大プーリングはブロック内のどこであっても活性化値（activation）の強さ（1）をそのまま保持するため、実質的にその近傍のどこかに特徴が現れたかどうかを問うていることになる。一方、平均プーリングでは特徴が 4 つのセルのうち 1 つしか占めていないため 0.25 が出力され、特徴が空間的に占める割合を測るとともに、周囲の 0 によって強い孤立応答を薄めてしまう。

また、ストライド 2 によるサブサンプリングには、特徴がサンプリング境界をまたぐ際に信号処理上の厄介な落とし穴が生じる。極端な例として、高周波で振動する信号 $1, 0, 1, 0, \ldots$ から 1 つおきにサンプルを取り出すと、1 ピクセルのシフト次第で出力がすべて 1 になるかすべて 0 になるかが切り替わってしまう。サンプリングレートで分解できる限界より速く信号が変化するときに生じるこの歪みは**エイリアシング**と呼ばれ、プーリングか畳み込みかを問わず、ストライド 2 のあらゆるダウンサンプラに影響を及ぼす。サブサンプリングの前に小さな平滑化フィルタを適用するとエイリアシングは軽減される (Zhang 2019)。この手法は本節末の演習問題で取り上げる。

歴史的には、シフト耐性こそがプーリングの主な動機であった。Fukushima の Neocognitron (Fukushima 1980) は、特徴を検出する細胞の層ごとに、局所近傍内のどこかに特徴が現れれば発火するプーリング様の細胞を対にして配置した。最大プーリングの初期の形態は音声認識で登場し (Yamaguchi et al. 1990)、その後、計算論的神経科学において視覚野における階層的な証拠集約のモデルとして最大プーリングが提案された (Riesenhuber and Poggio 1999)。

6.5.3 グローバルプーリング

画像内の**どこかに** Waldo が現れているかを問う際、6.1 節で求めた完全な平行移動不変性（translation invariance）はどうなるだろうか。今見たとおり、小さな固定ウィンドウではそれを保証できない。しかし、各特徴マップの空間領域**全体**を覆うようにプーリングウィンドウを広げれば実現できる。すべての空間位置にわたる最大値や平均は、応答がどこにあってもまったく同じスカラーになるからである（Figure 6.19 の下段）。この演算は**グローバルプーリング**と呼ばれ、各チャネルを単一の数値へと潰す。

しかし CNN 全体の中では、グローバルプーリングはゼロパディングされた畳み込み層の上に置かれており、そうした畳み込み層が厳密にシフト同変であるのは画像の境界から離れた領域に限られる（6.1 節）。6.3.1.3 節で見たように、内部のピクセルは 3×3 カーネルの 9 つのウィンドウすべてに関与するのに対し、境界のピクセルは 6 つ（四隅のピクセルは 4 つ）にしか関与しない。そのため、孤立した応答に対する大域的最大値は列によらず一定に保たれる一方、大域的な平均は特徴がパディングされた境界へ移動すると低下する。したがって、最後にグローバルプーリングを置くストライド 1 の畳み込み層の積み重ねは、顕著な内容が境界から離れたままであるようなシフトに対しては厳密に不変であり、それ以外では近似的に不変となる。ストライドを持つ層はこの保証を弱める。上で見たエイリアシングのため、不変性が厳密に成り立つのは各層のストライドの積の倍数だけシフトした場合に限られるからである。

第 7 章の現代的なアーキテクチャでは、プーリングが担う 2 つの役割は切り離されることが多い。中間層での空間的ダウンサンプリングは、固定された最大値や平均に決めてしまうのではなく各ウィンドウへの重み付けを学習するストライド 2 の畳み込みで処理されることが多く、ネットワークの末尾では最終の線形分類器の直前に**グローバル平均プーリング**（global average pooling）を置く (Lin et al. 2013)。次節で構築する古典的アーキテクチャである LeNet は、これら両方の慣行より前の時代のものである。

6.5.4 パディングとストライド

プーリングは畳み込みとまったく同じ方法で空間グリッド上にウィンドウをスライドさせるため、深層学習フレームワークでは同じウィンドウ形状・パディング・ストライドの引数でプーリング

層を指定でき、出力形状も 6.3 節と同一の式に従う。すなわち、両端に p ピクセルずつパディングした長さ n の軸に対し、ウィンドウサイズ k とストライド s を適用すると、その軸に沿った出力の数は $\lfloor (n + 2p - k)/s \rfloor + 1$ となる。唯一の微妙な違いは、パディングされた境界をどの**値**で埋めるかである。組み込みのプーリング層でこれらの挙動を確認するため、まず 0 から 15 までの要素を持つ単一の 4×4 特徴マップを格納した 4 次元テンソルを構築しよう。

```
# (example, height, width, channel), as in the previous sections
X = d2l.reshape(d2l.arange(16, dtype=d2l.float32), (1, 4, 4, 1))
X
```

```
Array([[[[ 0.],
         [ 1.],
         [ 2.],
         [ 3.]],

        [[ 4.],
         [ 5.],
         [ 6.],
         [ 7.]],

        [[ 8.],
         [ 9.],
         [10.],
         [11.]],

        [[12.],
         [13.],
         [14.],
         [15.]]], dtype=float32)
```

プーリングは通常、重なりのない領域を要約するために使われるので、大半のフレームワークではウィンドウが重ならずに入力を敷き詰めるよう、デフォルトでストライドをプーリングウィンドウのサイズと等しく設定する。この 4×4 の入力に対しては、ストライド 3 の 3×3 ウィンドウは 1 回しか収まらず、左上の 3×3 ブロックの最大値（10）を返す。プーリングには学習可能な重みがないため、この層ではパラメータの初期化が不要である。

ただし、関数 `nnx.max_pool` はデフォルトでストライドが 1 になるため、`strides` を明示的に渡す。

```
# The stride of nnx.max_pool defaults to 1, so we set it to the window shape
nnx.max_pool(X, window_shape=(3, 3), strides=(3, 3))
```

```
Array([[[[10.]]]], dtype=float32)
```

重なりのあるウィンドウや明示的なパディングを使いたい場合は、これらのデフォルト設定を上書きできる。パディングを 1、ストライドを 2 に設定すると、高さと幅の両方に沿ってサイズ $\lfloor (4 + 2 - 3)/2 \rfloor + 1 = 2$ の出力が得られる。

```
# One (before, after) pair of padding for each spatial axis
pool = partial(nnx.max_pool, window_shape=(3, 3), strides=(2, 2),
               padding=((1, 1), (1, 1)))
pool(X)
```

```
Array([[[[ 5.],
         [ 7.]],

        [[13.],
         [15.]]]], dtype=float32)
```

プーリング層のパディングされた境界には、どのような値を入れるべきだろうか。畳み込みでは、0 を掛けても重み付き和に何も寄与しないため、0 が自然な単位元（neutral element）となる。しかし最大プーリングでは、単位元は $-\infty$ である。パディングされたセルが実際の活性化値を上回ることは決してないため、各出力はウィンドウ内にある本来の入力要素の最大値と必ず一致する。上の例では、ウィンドウが覆う本来の行および列は $\{0, 1\}$ と $\{1, 2, 3\}$ であり、出力として 5、7、13、15 が得られる。また、$-\infty$ によるパディングは**値**に関するシフト同変性も保つ。すなわち、すべての入力要素に定数を加えると、プーリング後の各出力もまったく同じ定数だけシフトする。要素がすべて負である `X - 16` でこれを確認できる。

```
pool(X - 16)
```

```
Array([[[[-11.],
         [ -9.]],

        [[ -3.],
         [ -1.]]]], dtype=float32)
```

予想どおり、すべての出力がちょうど 16 だけ減少する。もし仮にこの層が 0 でパディングしていたなら、境界に接する 3 つのウィンドウは 0 を返してしまい、単に 0 がすべての負の入力より大きいというだけの理由で結果が損なわれていたはずである。

平均プーリングには、これに相当する中立な定数が存在しない。パディングされた 0 を分母に含めると、境界での平均に 0 へ引き寄せるバイアスが生じるため、各ウィンドウ内に収まる本来の入力要素の個数だけで割る方がすっきりとした方法である。

ここでの平均プーリングはデフォルトでパディングされた 0 を個数に含めるが（`count_include_pad=True`）、`count_include_pad=False` を指定すると本来の要素のみの平均をとる。

6.5.5 複数チャネル

畳み込み層がすべての入力チャネルにわたって和をとることでチャネル間の情報を混合するのに対し（6.4 節）、プーリング層は各チャネルに対して独立に作用する。その結果、出力のチャネル数は常に入力のチャネル数と完全に一致する。この挙動を確認するため、チャネル軸に沿って `X` と `X + 1` を連結し、2 チャネルのテンソルを作成しよう。

```
X = d2l.concat([X, X + 1], 3)  # the channel axis is last
X
```

```
Array([[[[ 0.,  1.],
         [ 1.,  2.],
         [ 2.,  3.],
         [ 3.,  4.]],

        [[ 4.,  5.],
         [ 5.,  6.],
         [ 6.,  7.],
         [ 7.,  8.]],
```

```
       [[ 8.,  9.],
        [ 9., 10.],
        [10., 11.],
        [11., 12.]],

       [[12., 13.],
        [13., 14.],
        [14., 15.],
        [15., 16.]]], dtype=float32)
```

先ほどのプーリング演算（パディング 1、ストライド 2）を適用すると、2 つの出力チャネルが得られる。1 つ目のチャネルは 5、7、13、15 を再現し、2 つ目のチャネルはそれらに 1 を加えた値を返す。

```
pool(X)
```

```
Array([[[[ 5.,  6.],
         [ 7.,  8.]],

        [[13., 14.],
         [15., 16.]]], dtype=float32)
```

結果はチャネルラスト形式であるため、2 つのチャネルは最後の軸に沿って並んで現れる。

プーリングは学習可能な重みを持たないため、それ自体でパターンを見つけ出すことはできず、先行する畳み込み層が検出した特徴を要約することしかできない。畳み込み、パディング、ストライド、ダイレーション、複数チャネルの特徴マップ、そしてプーリングがすべて揃ったことで、完全な畳み込みニューラルネットワークを組み立てて訓練し、これまでの MLP と直接比較するために必要な構成要素がすべて手に入ったことになる。

6.5.6 まとめ

プーリング層は、入力特徴マップの各空間ウィンドウをその最大値または平均によって要約する。学習可能なパラメータを一切追加せず、各チャネルを独立に処理し、畳み込みとまったく同じパディングおよびストライドの出力形状の規則に従う。また、最大プーリングではパディングされた位置が最大値に決して影響しないよう、境界を $-\infty$ でパディングする。ストライド 2 の 2×2 プーリング層は各軸に沿った空間解像度を半分にし、ブロック内にとどまる小さなシフトを吸収するが、応答がブロック境界をまたぐとやはりジャンプする。最大プーリングは近傍内のピークの活性化値を保持するのに対し、平均プーリングは孤立したピークをウィンドウ全体で薄めてしまう。最後に、グローバルプーリングは各特徴マップ全体を単一のスカラーへと潰す。ストライド 1 の層の上に置けば、内部のパターンに対して完全な平行移動不変性（パディングされた境界付近では近似的な不変性）が得られ、ストライドを持つ層と組み合わせた場合、不変性が厳密に成り立つのは各層のストライドの積の倍数だけシフトした場合に限られる。

6.5.7 演習問題

1. **プーリングの代替手法。**
    1. 独立した最小プーリング層を用意する必要があるか、それとも本節ですでに扱った演算を使って表現できるか。

2. 温度 $\tau > 0$ を持つ**ソフトマックスプーリング**は、ウィンドウ内の要素 $x_1, \dots, x_m$ を次のソフトマックス重み付き平均で置き換える。

$$\sum_{i=1}^{m} x_i \frac{\exp(x_i/\tau)}{\sum_{j=1}^{m} \exp(x_j/\tau)}. \tag{6.21}$$

$\tau \to 0$ および $\tau \to \infty$ としたときの極限を求めよ。また、最大プーリングと比べてソフトマックスプーリングが追加で要するコストを 1 つ挙げよ。

2. **エイリアシングとブラープール**。単純なストライド 2 のダウンサンプリングは、入力の要素を 1 つおきに残す。周波数を入力 1 サンプルあたりの周期数で測ることにすると、1 次元信号 $(1, 0, 1, 0, 1, 0)$ と $(0, 1, 0, 1, 0, 1)$ の周波数はいずれも $\frac{1}{2}$ である。
   1. 1 マス分のシフトだけが異なるこれら両方の信号に単純なストライド 2 のダウンサンプリングを適用し、2 つの出力を比較せよ。なぜこの挙動は**エイリアシング**と呼ばれるのか。
   2. ストライド 2 のサブサンプラが忠実に表現できる入力周波数はどの範囲か。
   3. ブラープール (Zhang 2019) は、サブサンプリングの前に小さな平均化フィルタを適用することでエイリアシングを抑える。2 タップのボックスフィルタ $(\frac{1}{2}, \frac{1}{2})$ を用いたブラープールが上の 2 つの信号に対して何を計算するか求めよ。これは本節のどのプーリング演算と一致するか。

3. [code] **最大プーリングの逆伝播**。$k_\mathrm{h} \times k_\mathrm{w}$ ウィンドウおよびウィンドウサイズと等しいストライドを持つ 2 次元最大プーリングについて、入力に関する勾配を導出し、自動微分を使わずに実装せよ。本節の組み込み最大プーリング演算に対して自動微分が計算する勾配と、小さなランダムテンソル上で相対誤差 10^{-4} 未満で一致することを検証せよ。

   Stanford CS231n、課題 2、ConvolutionalNetworks の素朴な最大プーリングの逆伝播より改変。

4. **ReLU による最大プーリング**。●畳み込みは線形であるため、それ単独では最大値を計算できない。
   1. $\max(a, b)$ を、アフィン演算と $\mathrm{ReLU}(x) = \max(0, x)$ のみを用いて表せ。
   2. これを利用して、畳み込みと ReLU 層の積み重ねとして最大プーリングを実装せよ。
   3. その構成が 2×2 ウィンドウおよび 3×3 ウィンドウに対して使用する層ごとのチャネル数と ReLU 層の数を数えよ。ReLU 層の数はウィンドウ内の要素数に伴ってどのように増加するか。

6.6 LeNet

ここまでで、畳み込みニューラルネットワーク（convolutional neural network）の構成要素がすべて出そろった。すなわち、畳み込み層（6.2 節）、パディングとストライド（6.3 節）、複数チャネル（6.4 節）、およびプーリング（6.5 節）である。いよいよこれらを組み合わせて完全な画像分類器を構築し、これまでの議論を検証してみよう。2 次元の空間構造を尊重することで、実際に標準的な MLP を上回る性能が得られるのだろうか。

ベースラインとするのは 4.2 節の MLP であり、Fashion-MNIST において 30 エポック後に約 87% の検証正解率に達する。MLP が各画像を順序のない 784 個の数値へと平坦化する（6.1 節）のに対し、CNN はピクセルの格子構造を保ったまま、共有された局所カーネルをすべての空間位置に適用する。2 つのアーキテクチャを公平に比較するため、同じ交差エントロピー損失（cross-entropy loss）、SGD オプティマイザ、30 エポックという計算予算で CNN を訓練する。正解率という 1 つの数値だけでは全体像は見えないため、第 1 畳み込み層が学習したフィルタを観

察するとともに、ピクセルをシャッフルした画像を用いた対照実験によって CNN の優位性がどこから生じるのかを確かめる。

本節で実装するアーキテクチャは **LeNet** である。手書き数字を認識するために AT&T Bell Labs の同僚らとともにこれを開発した Yann LeCun にちなんで名付けられた (LeCun, Léon Bottou, et al. 1998)。LeNet は、誤差逆伝播法（backpropagation）で訓練された最初期のエンドツーエンド畳み込みネットワークの 1 つに端を発する、10 年におよぶ研究の集大成であった (LeCun et al. 1989)。1998 年までに LeNet-5 は手書き数字に対して 1% 未満の誤り率を達成し、当時の機械学習を席巻していたサポートベクターマシンに匹敵する性能を示した。またその派生モデルは、銀行の小切手や ATM 預け入れ時の手書き金額を読み取るシステムとして商用展開された。エンドツーエンドで学習したネットワークが、人手で入念に設計されたビジョンパイプラインに取って代われることを、LeNet は早い段階で説得力をもって実証したのである。

6.6.1 LeNet のアーキテクチャ

大まかに見ると、LeNet は Figure 6.20 に示すように 2 つの段階へ明確に分かれる。2 つの畳み込みブロックから構築された**畳み込みエンコーダ**（convolutional encoder）と、それに続く 3 つの全結合層（fully connected layer）からなる**全結合分類ヘッド**（dense classification head）である。

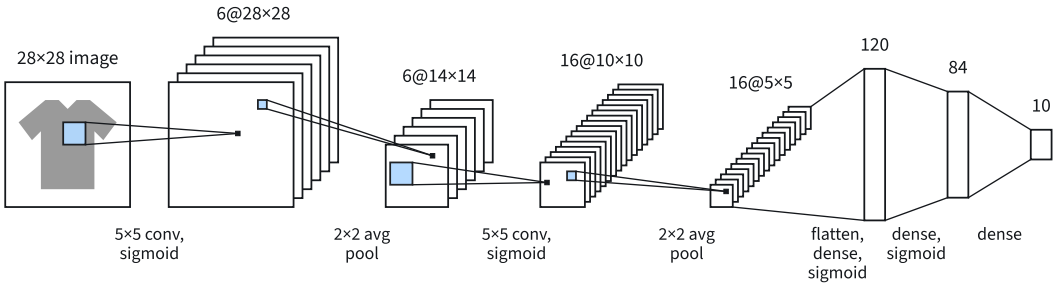


Figure 6.20: 1 枚の 28×28 Fashion-MNIST 画像に対する LeNet 内のデータフロー（特徴マップは実寸比で描画）。網掛けされた各ウィンドウから、矢印が指す次の層の濃いユニットが計算される。3 つの全結合層はそれぞれ 120、84、10 個の出力を持つ。

各畳み込みブロックは、5×5 の畳み込み層およびシグモイド活性化関数と、ストライド 2 の 2×2 平均プーリング層を組み合わせたものである。第 1 畳み込み層は 6 チャネルを出力し、第 2 畳み込み層は 16 チャネルを出力する。

エンコーダの出力は、チャネルラストの形状 `(batch_size, height, width, channels)` を持つ。

畳み込みエンコーダを全結合ブロックに接続するため、各データ例の空間的な特徴マップを 1 本のベクトルへと平坦化し、ミニバッチ内の例ごとに 1 行を持つ 2 次元行列を作る。続いて 3 つの全結合層がこのベクトルを 120 個および 84 個の隠れユニットを経由して、ソフトマックス回帰と同様にクラスごとの 10 個の出力ロジットへと射影する。

なお、ここで扱うのは LeNet-5 の標準的な教育用バージョンであり、当時の実装を細部まで忠実に再現したものではない。1998 年のネットワークにおける畳み込みとプーリングの交互配置や各層の幅（6、16、120、84）は維持しつつ、今日ではほとんど使われない当時の細かな仕様（スケーリングされた tanh 活性化関数、学習可能なプーリングのスケールとバイアス、2 つの畳み込み間の疎なチャネル結線、動径基底関数（radial basis function）による出力ユニットなど）は簡略化している。ただし、コードには歴史的な名残が 1 つ現れている。元の LeNet-5 は 32×32 の入力を受け取っていたという点である。MNIST の数字画像は 28×28 であるが、1998 年の論文

では数字の端近くにある筆画も上位の特徴検出器の受容野（receptive field）に収まるよう、四方を 2 ピクセル分の背景で囲んでいた。本節の第 1 畳み込み層における 2 ピクセルのパディングはこれと同じ役割を果たし、最初の 5×5 畳み込みは再び 28×28 の出力を生成する。

現代の深層学習フレームワークを使えば、LeNet の組み立てに必要なのは、2 つの畳み込み・プーリング段階と 3 つの全結合層を連結し、すべてのバイアスを 0 に設定する単一の `Sequential` ブロック `self.net` だけである。

```
class LeNet(d2l.Classifier):
    """The layout of LeNet-5, with sigmoid activations and average pooling."""
    def __init__(self, lr=0.1, num_classes=10, kernel_init=None, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs', 'kernel_init'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        kernel_init = (nnx.initializers.xavier_uniform() if kernel_init is None
                       else kernel_init)

        def avg_pool(x):  # 2x2 windows, stride 2
            return nnx.avg_pool(x, window_shape=(2, 2), strides=(2, 2))

        def flatten(x):
            return x.reshape((x.shape[0], -1))

        self.net = nnx.Sequential(
            nnx.Conv(1, 6, kernel_size=(5, 5), padding=2,
                     kernel_init=kernel_init, rngs=rngs), nnx.sigmoid,
            avg_pool,
            nnx.Conv(6, 16, kernel_size=(5, 5), padding='VALID',
                     kernel_init=kernel_init, rngs=rngs), nnx.sigmoid,
            avg_pool,
            flatten,
            nnx.Linear(400, 120, kernel_init=kernel_init, rngs=rngs),
            nnx.sigmoid,
            nnx.Linear(120, 84, kernel_init=kernel_init, rngs=rngs),
            nnx.sigmoid,
            nnx.Linear(84, num_classes, kernel_init=kernel_init, rngs=rngs))
```

6.6.2 各層の形状

パディングの値が 1 つずれたり平坦化層を入れ忘れたりするだけで後続すべてのテンソル形状が変わってしまうため、訓練を始める前に中間層の形状を確認しておくのがよい習慣である。Figure 6.21 は、簡略記法で表した LeNet の図に、各境界で想定されるテンソル形状を添えたものである。実装がこの図と一致するか確かめるため、単一チャネルの 28×28 ダミーテンソルをモデルに通し、各層を通過した後の出力形状を表示する。次章のより深いアーキテクチャでも再利用できるよう、この診断処理を `layer_summary` メソッドとして `d2l.Classifier` に実装しておこう。

```
@d2l.add_to_class(d2l.Classifier)
def layer_summary(self, X_shape):
    X = jnp.zeros(X_shape)
    for layer in self.net.layers:
        X = layer(X)
```

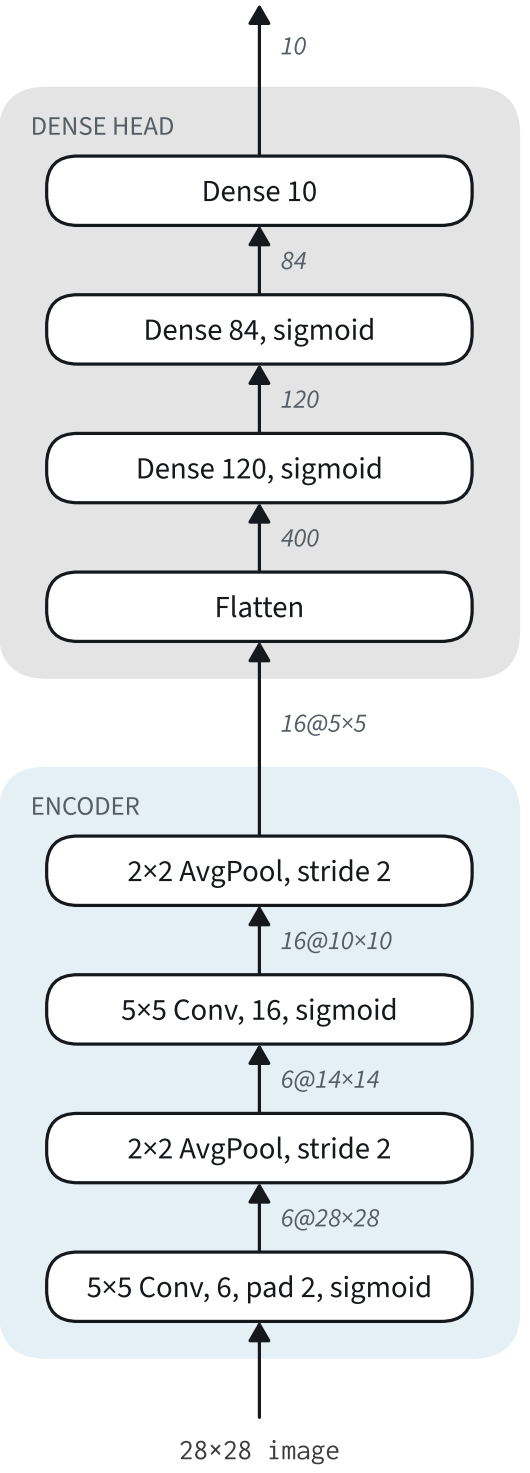


Figure 6.21: LeNet の簡略記法。各矢印の横には、そこを流れるテンソルの形状が「チャネル数 @ 高さ × 幅」の形式で記されている。

```
        # Activations and pooling are plain functions: print their names
        name = getattr(layer, '__name__', type(layer).__name__)
        print(f'{name:<22} {tuple(X.shape)}')

model = LeNet()
model.layer_summary((1, 28, 28, 1))
```

```
Conv                   (1, 28, 28, 6)
sigmoid                (1, 28, 28, 6)
avg_pool               (1, 14, 14, 6)
Conv                   (1, 10, 10, 16)
sigmoid                (1, 10, 10, 16)
avg_pool               (1, 5, 5, 16)
flatten                (1, 400)
Linear                 (1, 120)
sigmoid                (1, 120)
Linear                 (1, 84)
sigmoid                (1, 84)
Linear                 (1, 10)
```

表示された形状は Figure 6.21 と完全に一致しており、6.3 節で導いた出力形状の規則に従っている。パディングなしの 5×5 カーネルは高さと幅をそれぞれ 4 減らすため、第 1 畳み込み層では各辺に 2 ピクセルのパディングを加えて 28×28 の空間解像度を維持し、第 2 畳み込み層ではパディングを行わずに 14×14 を 10×10 へと縮小する。各 2×2 平均プーリング層は両方の空間次元を半分にする（$28 \to 14$ および $10 \to 5$）。その一方で、エンコーダが空間解像度と引き換えにより豊かな特徴表現を獲得するにつれ、チャネルの深さは 1 から 6、さらに 16 へと拡大する。最後の $16 \times 5 \times 5$ テンソルを平坦化すると 400 次元のベクトルが得られ、全結合ブロックがこれを段階的に 10 個のクラスロジットへと圧縮する。

6.6.3 パラメータと積和演算

すべての層の出力形状が確認できたところで、パラメータのメモリ量と演算コストの両方を集計してみよう。6.4.2 節で見たように、入力チャネル数 c_{i}、出力チャネル数 c_{o}、カーネルサイズ $k \times k$ の畳み込み層は、$c_{\mathrm{i}} c_{\mathrm{o}} k^2$ 個の重みと c_{o} 個のバイアスを保持し、出力の空間位置ごとにすべての重みを 1 回ずつ再利用する。対照的に、全結合層は 1 つの例につき各重みを 1 回しか使わない。Table 6.1 に LeNet 全体での内訳を示す。

Table 6.1: LeNet の各層における 1 例あたりのパラメータ数と積和演算数。プーリング層と活性化層はパラメータを持たない。

層	出力	パラメータ数	積和演算数
5×5 畳み込み、1 から 6 チャネル	6@28×28	$150 + 6 = 156$	$150 \times 784 = 117{,}600$
5×5 畳み込み、6 から 16 チャネル	16@10×10	$2{,}400 + 16 = 2{,}416$	$2{,}400 \times 100 = 240{,}000$
全結合、400 から 120	120	$48{,}000 + 120 = 48{,}120$	$48{,}000$
全結合、120 から 84	84	$10{,}080 + 84 = 10{,}164$	$10{,}080$
全結合、84 から 10	10	$840 + 10 = 850$	840
合計		$61{,}706$	$416{,}520$

この内訳から、CNN の設計全般に繰り返し現れるメモリと計算量の際立った非対称性が浮かび上

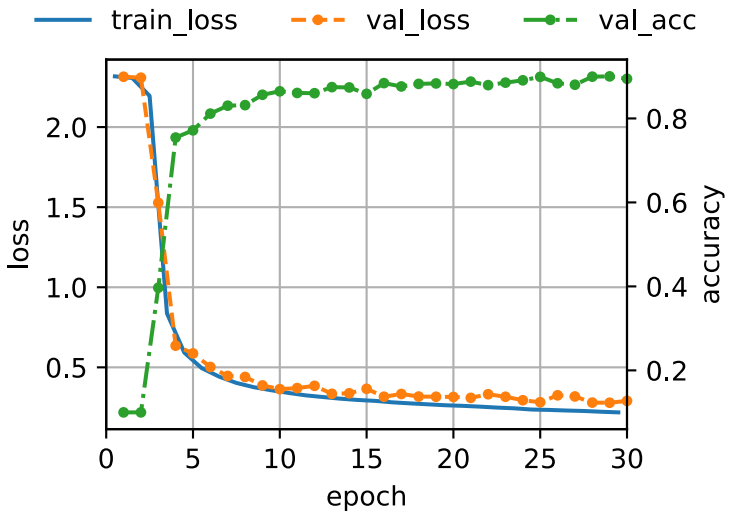
がる。2 つの畳み込み層が占めるのは LeNet のパラメータのわずか 4% にすぎないが、カーネルの各重みが空間格子全体にわたって評価されるため、積和演算の 86% を担っている。逆に、最初の全結合層だけでパラメータの 78% を保持する一方、演算量への寄与は 12% にとどまる。先に扱った Fashion-MNIST の MLP（パラメータ数 203,530、画像あたりの積和演算数約 203,000）と比べると、LeNet はパラメータ数が 3 分の 1 未満である一方、約 2 倍の計算を実行する。

6.6.4 訓練

MLP のときと同様に、ミニバッチサイズ 128（MLP の 256 の半分）のミニバッチ SGD によって交差エントロピー損失を最小化し、LeNet を 30 エポック訓練する。ただし、1 つだけ大きく変更しなければならないハイパーパラメータがある。学習率（learning rate）である。シグモイド活性化関数の導関数は決して $1/4$ を超えないため（4.1.4 節）、第 1 畳み込み層へ逆伝播する勾配は 4 つの連続するシグモイドによって減衰する。MLP と同じ 0.1 の学習率では訓練が極端に遅くなり、検証正解率は最初の 2〜4 エポックの間ランダム予測の水準付近に留まり、10 エポック後でも約 73% までしか上がらない。したがって、学習率を 2.0（より小さな勾配スケールを反映し、経験的に調整した 20 倍の値）へと引き上げる。

5.7.4 節と同様、GPU が利用可能な場合には `d2l.Trainer` が自動的に GPU 上で訓練を実行する。

```
trainer = d2l.Trainer(max_epochs=30, num_gpus=1)
data = d2l.FashionMNIST(batch_size=128)
model = LeNet(lr=2.0)
trainer.fit(model, data)
```



30 エポック後、LeNet はパラメータ数が 3 分の 1 未満であるにもかかわらず、MLP の約 87% を上回る約 88〜90% の検証正解率を達成する。

訓練曲線の立ち上がりにも注目してほしい。ランダム初期化（random initialization）の違いだけで、正解率が直ちに上昇することもあれば、損失が $\ln 10 \approx 2.3$ 付近に張り付いたまま数エポックにわたってランダム予測の水準（10%）付近をさまよい、その後急激に立ち上がることもある。この初期のプラトーの間、ネットワークはどの画像に対してもほぼ同一の一様な確率を予測している。4 つのシグモイド層を通過するうちに、ロジットにおける入力依存の変動が当初は小

さくなりすぎて、クラスを分離できないからである。この信号消失の効果については後述の演習問題で定量化する。また、同じプラトーがピクセルシャッフル実験でも再び現れることになる。

6.6.4.1 学習された特徴

ソフトマックス回帰では各クラスの重みをそのまま 28×28 のテンプレート画像として可視化でき（3.3 節）、同様に MLP の各隠れユニットも固定されたピクセル位置に重みを付けていた（4.2 節）。対照的に、畳み込みカーネルは特定のピクセルに紐づいておらず、単なる 5×5 の重みパッチであるため単独では解釈しにくい。そこで第 1 畳み込み層が何を発見したのかを探るため、まず検証ミニバッチからスニーカー（クラス 7）1 枚とシャツ（クラス 6）1 枚を取り出し、2 枚の画像からなるテンソル `X` に格納する。

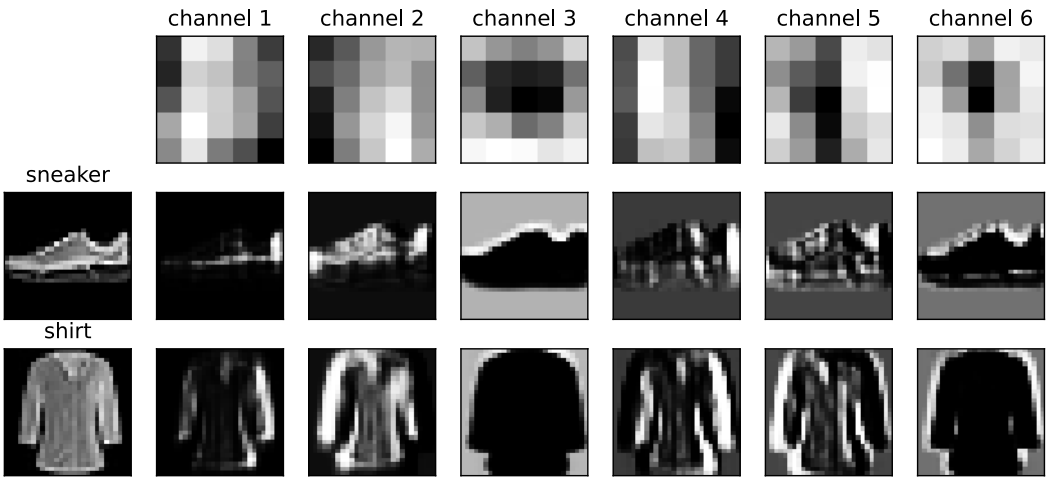
```
X, y = trainer.prepare_batch(next(iter(data.val_dataloader())))
labels = d2l.numpy(y)
X = d2l.stack([X[int(np.argmax(labels == c))] for c in (7, 6)], 0)
```

次に、`X` を第 1 畳み込み層とそのシグモイド活性化関数に通し、6 枚の 28×28 特徴マップ `maps` と学習済みの 5×5 カーネル `kernels` を抽出する。

```
layers = model.net.layers
maps = layers[1](layers[0](X))  # The first convolution and its sigmoid
# Channels-last to channels-first, so that maps[k, c] is one 28x28 map
images, maps = d2l.numpy(X[..., 0]), d2l.numpy(maps).transpose(0, 3, 1, 2)
kernels = d2l.numpy(layers[0].kernel[...])[:, :, 0].transpose(2, 0, 1)
```

これで、最上段に 6 個の `kernels` を並べ、その下に入力画像 `images` とシグモイド通過後の 6 枚の特徴マップ `maps` を表示できる。各パネルはそれぞれの最小値（黒）から最大値（白）まで独立にスケーリングしている。

```
names = data.text_labels([7, 6])
axes = d2l.show_images(
    [kernels[0], *kernels, images[0], *maps[0], images[1], *maps[1]], 3, 7,
    titles=['', *[f'channel {c}' for c in range(1, 7)],
            names[0], *[''] * 6, names[1], *[''] * 6], cmap='gray')
axes[0].set_visible(False)  # No kernel above the input images
```



訓練を繰り返しても、一貫して 2 種類の明確に異なる特徴マップ群が立ち現れる。一部のチャネルは局所的な明るさや前景の検出器として働き、衣類の塗りつぶされたシルエット（暗い背景に明るい領域、あるいはその反転）を再現する。残りのチャネルは衣類の輪郭、とりわけ左右の境界に沿って選択的に強く反応する。これらの 5×5 カーネルを調べるとその仕組みがわかる。明るい列と暗い列が隣り合って対をなしており、6.2.2.2 節で人手により設計した $(1, -1)$ 垂直エッジカーネルの複数行版を学習によって獲得しているのである。以前は勾配降下法（gradient descent）にエッジフィルタを学習させるため明示的な正解エッジマップを与える必要があったが、ここでは生の衣類ピクセルとクラスラベルだけから、エッジとシルエットが視覚的分類に有用な構成要素であることを LeNet 自らが発見している。（6 つのチャネルへの具体的な役割分担は実行ごとに異なる。）

6.6.4.2 シャッフルされたピクセル

MLP に対する 1〜3 パーセントポイントの改善を、どの程度重く受け止めるべきだろうか。このマージンだけを見ればノイズのせいかもしれない。MLP の検証正解率はエポックごとに約 1 ポイント変動するうえ（4.2 節）、LeNet は深さ、活性化関数（activation function）、学習率も異なっているからである。より純粋な対照実験は、畳み込みだけが活用している唯一の情報、すなわち空間的隣接性を取り除くことである。本章の冒頭で述べた思考実験に立ち返り、784 個のピクセル位置に対する固定されたランダムな置換を 1 つ生成し、まったく同じ並び替えをすべての訓練画像と検証画像に適用しよう。

```
perm = np.random.permutation(28 * 28)  # One fixed order for every image

def shuffle_pixels(images):
    """Rearrange the 784 pixels of every image into the order perm."""
    return images.reshape(len(images), -1)[:, perm].reshape(images.shape)
```

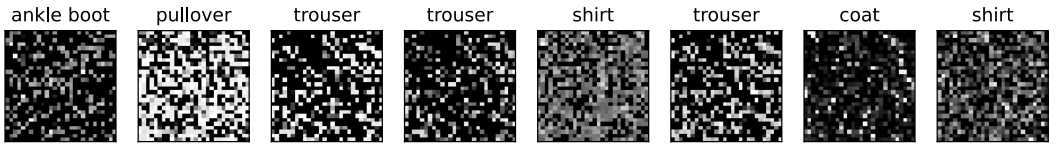
それでは `shuffle_pixels` を使い、この共通の置換 `perm` を `data_shuffled` の訓練データと検証データのすべての画像に適用しよう。

```
data_shuffled = d2l.FashionMNIST(batch_size=128)
data_shuffled.train, data_shuffled.val = [
    (shuffle_pixels(images), labels)
```

```
    for images, labels in (data_shuffled.train, data_shuffled.val)]
```

data_shuffled から検証ミニバッチを取り出して可視化すると、784 個のピクセル座標を置換したことで判別可能な衣類の輪郭がすべて失われ、人間の目には純粋な砂嵐ノイズにしか見えないパターンになっていることが確認できる。

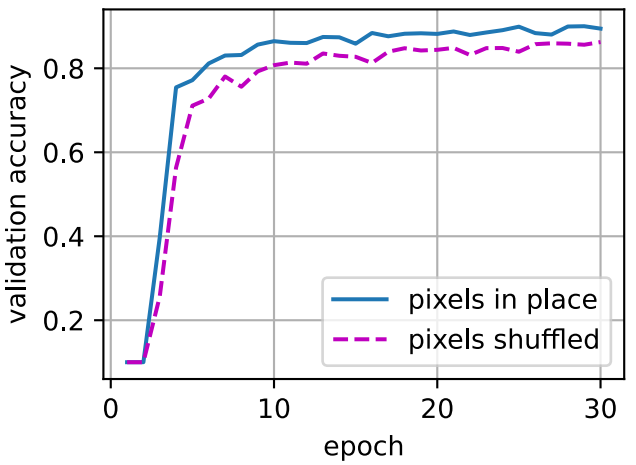
```
data_shuffled.visualize(next(iter(data_shuffled.val_dataloader())))
```



MLP はこの置換の影響をまったく受けない。その理由を証明するのに再訓練すら不要である。入力座標を置換しても、第 1 層の重み行列 $\mathbf{W}^{(1)}$ の行が入れ替わるにすぎないからである。各重みは独立同分布で初期化されるため、シャッフルされたピクセルでの訓練は元の画像での訓練と統計的に同一であり、同じ約 87% の正解率をもたらす。しかし LeNet にとっては、シャッフルによって空間的局所性が破壊され、すべての 5×5 カーネルが元の衣類上にランダムに散らばった 25 ピクセルを読み取ることになる。まったく同じ設定のままシャッフル後のデータセットで LeNet を再訓練し、両方の検証曲線を並べてプロットしてみよう。

```
curves = {'pixels in place': trainer.history['val_acc']}
trainer.board.display = False
trainer.fit(LeNet(lr=2.0), data_shuffled)
curves['pixels shuffled'] = trainer.history['val_acc']
d2l.plot(list(range(1, trainer.max_epochs + 1)), list(curves.values()),
         'epoch', 'validation accuracy', legend=list(curves))
for name, acc in curves.items():  # A single epoch can land a point low
    print(f'{name}: {sum(acc[-5:]) / 5:.3f} averaged over the last 5 epochs')
```

```
pixels in place: 0.892 averaged over the last 5 epochs
pixels shuffled: 0.859 averaged over the last 5 epochs
```



シャッフルされたピクセル上では、LeNet の正解率は典型的な実行で数パーセントポイント低下して 85〜86% 前後となり、MLP と同等かやや下回る水準に落ち込む。空間的な近傍構造が破壊されると LeNet の優位性は消失し、MLP に対するリードが畳み込みの帰納バイアス（inductive bias）から直接もたらされていたことが裏付けられる。

なお、初期のランダム予測水準のプラトーはここでも同様に予測不能な形で現れ得る。通常の画像では数エポックを超えることはまれだが、シャッフルされた画像では運悪く初期化が偏ると 30 エポックの大半にわたってランダム予測の水準付近から抜け出せなくなり、最終的な正解率がさらに押し下げられることがある。この不安定性はピクセルの順序そのものではなく 4 つのシグモイド活性化関数を重ねていることに起因しており、プラトーを即座に脱出した実行であっても、シャッフルされたピクセル上の LeNet が MLP を上回ることはない。

最後に、LeNet を 6.1 節の対称性の原理と改めて結びつけよう。理想的な画像分類器は厳密に平行移動同変な特徴層と厳密に平行移動不変な出力を備えるはずだが、LeNet はどちらも厳密には達成していない。パディングを伴う畳み込みは境界付近で同変性を破り、プーリングが吸収できるのはプーリングブロック内に収まるシフトに限られ（6.5 節）、最後の 5×5 マップを平坦化する処理はすべての空間セルに別々の全結合の重みを割り当てるからである。したがって LeNet のシフト耐性は部分的なものにとどまる。すべての衣類がきれいに中央配置された Fashion-MNIST には十分だが、制約のない一般のシーンには不十分である。

2 つの実験を総合すると、CNN が 3 分の 1 未満のパラメータ数で MLP に匹敵し、さらには上回ることができるのは、まさに重み共有（weight sharing）によって各カーネルが局所的な視覚パターンを一度学習するだけで格子上のどこででも検出できるからだとわかる。小さな 28×28 の画像ではこの効率性はささやかな利点にすぎないが、6.1 節で議論したメガピクセル級の写真（単一の全結合層だけで 10 億個の重みを要する）においては、同じ原理がそもそもモデルを構築できるかどうかを左右する。深い畳み込みネットワークを大規模環境で**訓練しやすくする**ことが次の課題であり、LeNet の 4 つのシグモイド層が見せた鈍く不安定な収束は、現代のアーキテクチャがなぜ 1998 年の姿から大きく進化しなければならなかったのかをすでに示唆している。

6.6.5 まとめ

LeNet は、空間解像度を段階的に縮小してより深いチャネル表現へと変換する畳み込みエンコーダと、3 層の全結合分類ヘッドを組み合わせたモデルである。その畳み込み層はパラメータのわずか 4% しか持たないにもかかわらず積和演算の 86% を実行し、一方で最初の全結合層だけでパラメータの 78% を占める。4.2 節の MLP と同じ 30 エポックの SGD 設定で訓練すると、LeNet は 3 分の 1 未満のパラメータ数で Fashion-MNIST において約 88〜90% の検証正解率（MLP は約 87%）に到達し、クラスラベルのみからシルエットフィルタや垂直エッジフィルタを自動的に学習する。ピクセル位置をシャッフルするとこの改善が消えることから、優位性の源泉が空間的局所性にあると特定できる。同時に、4 つのシグモイド層を通して勾配を伝播させるため、はるかに大きな学習率（0.1 ではなく 2.0）が必要となり、訓練初期に予測不能なプラトーが生じる。

エンコーダとヘッドを組み合わせる全体の基本設計は今なお礎であり続けているが、LeNet を構成する個々の部品はその後の発展でほぼすべて現代化された。ReLU がシグモイドに取って代わり、最大プーリングやストライド付き畳み込みが平均プーリングを置き換え、グローバル平均プーリングと単一の線形層の組み合わせが重い全結合ヘッドに取って代わり、正規化層が深いモデルの訓練を安定させたのである。次章では AlexNet（7.1 節）を皮切りに、ネットワークが深さ・幅・解像度の面でスケールアップしていく過程に沿って、こうしたアーキテクチャ上の進歩を 1 つずつたどっていく。

6.6.6 演習問題

1. [code] **現代的な構成要素**。本節の訓練設定を用い、以下の変更を加えて `LeNet` を再訓練し、

30 エポック後の各変種の検証正解率を LeNet と比較せよ。

1. 平均プーリングを最大プーリングに置き換えよ。
2. シグモイド活性化関数を ReLU に置き換えよ。損失が発散するか正解率がランダム予測の水準にとどまる場合は、訓練が成功するまで学習率を下げ、使用した学習率を報告せよ。
3. 両方の変更を同時に行い、その結果得られるモデルを現代化版 LeNet と呼ぶことにする。正解率の差の大部分をもたらしているのは、2 つの変更のうちどちらか。
4. ●訓練開始時点の重みを用いて、LeNet と現代化版 LeNet のそれぞれについて、第 1 畳み込みカーネルに関する損失の勾配のノルムと、ミニバッチ全体にわたるロジットの標準偏差を測定せよ。4.4.2.1 節における Xavier 初期化の導出では、平均ゼロの入力と非線形性がないことを仮定している。シグモイド出力の平均とその傾き $\sigma'(0) = 1/4$ を用いて、測定結果、LeNet が必要とする学習率、および訓練初期にランダム予測の水準にとどまるエポックが生じる理由を説明せよ。

2. [code] **元の MNIST**。MNIST データクラスを d2l.FashionMNIST を改変して作成し、それぞれ Fashion-MNIST で用いた設定のまま、LeNet と 4.2 節の MLP の両方をそのデータ上で訓練せよ。LeCun, Léon Bottou, et al. (1998) は、MNIST における LeNet-5 のテスト誤差率が 1% 未満であると報告している。

   1. LeNet の検証誤差率をこの数値と比較せよ。両者は同じ画像上で測定されている（Fashion-MNIST と同様、本書の検証セットはこのデータセットの公式テストセットである）。
   2. 各データセットについて、2 つのネットワークの検証誤差率の差と比を計算せよ。3.2 節では、MNIST ではモデルクラス間の違いが見えにくくなると述べられている。2 つの指標はそれぞれこの記述を裏付けているか。

3. [code] **より深い層の特徴マップ**。本文では第 1 畳み込み層の特徴マップを描画した。

   1. プルオーバーとコートのように混同されやすい 2 つのクラスの検証画像について、訓練済み LeNet の第 2 畳み込み層のシグモイド通過後の特徴マップ 16 枚を表示せよ。各チャネルが何に反応しているかをまだ説明できるか。2 つのクラス間で最も違いが大きいのはどのチャネルか。
   2. 問題 1 の現代化版 LeNet を訓練し、Fashion-MNIST の検証画像、猫や自動車などの分布外の写真、および一様ランダムノイズのそれぞれに対して、2 つの畳み込み層それぞれの ReLU 通過後における活性化値の最大値を測定せよ。各写真は 28×28 のグレースケール画像に変換し、すべての入力を $[0, 1]$（d2l.FashionMNIST のピクセル値の範囲）に収めること。写真およびノイズに対する最大値は、検証画像で観測される範囲の内側に収まるか、それとも明確に外れるか。

4. [code] **シフトさせた衣類画像**。本文では、LeNet のシフトに対する耐性は部分的なものにすぎないと論じた。

   1. すべての検証画像を右に 1、2、4 ピクセルシフトさせ（空いた列はゼロで埋める）、シフトさせた各データセット上で訓練済み LeNet の検証正解率を測定せよ。ネットワークが耐性を持つのはどのシフトか。得られた結果を 2 つのプーリング層および全結合ヘッドと関連づけて考察せよ。
   2. LeNet の畳み込み層はすべての位置で重みを共有しているにもかかわらず、平行移動させた訓練画像のコピーを用いて訓練すると、シフトさせた検証画像に対する正解率が向上し得る理由を説明せよ。

   Michael Nielsen『Neural Networks and Deep Learning』第 6 章より改変。

5. [code] **全結合ヘッド**。●Table 6.1 が示すように、最初の全結合層は LeNet のパラメータの

78% を保持している。

1. 平坦化層から出力層までの全結合ヘッド全体を、グローバル平均プーリングとそれに続く 16 ユニットから 10 ユニットへの単一の線形層に置き換えよ。このネットワークのパラメータ数はいくつになるか。訓練する前に、検証正解率が `LeNet` と比べて 1 パーセントポイント以上変化するかどうかを予測し、そのうえで再訓練して比較せよ。
2. 今度は、平坦化された 400 次元の畳み込み出力を 10 出力の線形出力層へ直接接続せよ。このネットワークのパラメータ数はいくつになるか。再訓練し、縮小された 2 種類のヘッドの正解率を互いに、また事前の予測と比較せよ。この比較から、LeNet の表現能力がどこで使われているかについて何が言えるか。

Michael Nielsen『Neural Networks and Deep Learning』第 6 章より改変。

Chapter 7

現代の畳み込みニューラルネットワーク

畳み込み層と LeNet の仕組みを押さえたところで（第 6 章）、本章では 2012 年の ImageNet における AlexNet のブレークスルーから、2020 年代初頭の畳み込みネットワークと Vision Transformer の対話に至るまで、10 年にわたる畳み込みアーキテクチャ設計の歩みをたどる。節目となった各モデルを孤立した産物として扱うのではなく、どのようなボトルネックがそれぞれの設計ステップを促したのかを問い、章全体を通して累積的なツールキットを引き継いでいく。

冒頭の各節では、個別に手作業で組まれた層からモジュール式の構成ブロックへと進む。AlexNet（7.1 節）は、大規模なラベル付きデータセットと GPU が結びついたことで、エンドツーエンドの特徴学習（feature learning）がいかに人手で設計されたビジョンパイプラインを凌駕したかを実証している。同モデルは ReLU 活性化関数、ドロップアウト、データ拡張（data augmentation）、および本章全体で再利用するウォームアップ付きコサイン学習率スケジュールを導入した一方で、畳み込みの計算量と全結合ヘッドのパラメータ数との間にある際立った不均衡も浮き彫りにした。続く VGG、Network in Network（NiN）、GoogLeNet（7.2 節）は、繰り返し可能な 3×3 ブロックによるネットワークの構成、1×1 畳み込みによるピクセルごとのチャネル混合、巨大な全結合ヘッドからグローバル平均プーリング（global average pooling）への置き換え、そしてステム・ボディ・ヘッド構造の内部での並列マルチスケール分岐の組み合わせによって、この不均衡に対処する。

中盤の各節が扱うのは、深さとハードウェア効率のスケーリングである。正規化層（7.3 節）は、ミニバッチ全体または個々のデータ例の内部で中間の活性化値を標準化し、隠れ層を重みのスケールに対して不感にすることで、実用可能な学習率の幅を何桁も広げる。残差ネットワーク（7.4 節）は、加算ショートカット $\mathbf{x} + g(\mathbf{x})$ により $g = 0$ で恒等写像（identity mapping）を保つことで、深いプレーンな積層ネットワークにおける最適化の劣化を解消し、さらにグループ畳み込み（ResNeXt）および特徴の連結（DenseNet）によってブロックを拡張する。効率的な畳み込みネットワーク（7.5 節）では、正解率と並ぶ重要な設計目標としてレイテンシ、メモリ、エネルギーを位置づけ、深さ方向分離可能な分解（depthwise-separable factorization）、逆ボトルネック（inverted bottleneck）、複合スケーリング（compound scaling）、Squeeze-and-Excitation ゲート、構造的再パラメータ化（structural re-parameterization）を導入するとともに、演算回数が GPU の実経過時間としばしば乖離する理由を明らかにする。

終盤の各節では、訓練手法とアーキテクチャを切り分ける。現代の訓練レシピ（7.6 節）では、AdamW、コサインウォームアップ、ラベル平滑化（label smoothing）、Mixup、CutMix、確率的深さ（stochastic depth）、重み平均化といった現代的な最適化と正則化だけでも、アーキテク

チャを変えない ResNet において ImageNet の正解率の数パーセントポイント分を担っていることを示す。このようなレシピを固定したうえで、ConvNeXt（7.7 節）は ResNet を段階的に現代化し、アテンションを用いずに階層型 Vision Transformer に匹敵する性能を達成する。そして畳み込みネットワークの設計空間（7.8 節）では、サンプリングされたモデル集団にわたる経験誤差の分布を比較することで、RegNet ファミリーの簡潔な構造規則を抽出する。

7.1 AlexNet と学習された画像表現

6.6 節では、LeNet が畳み込みフィルタから最終的な分類器に至るパイプラインのすべての段階を、生のピクセルからエンドツーエンドで訓練した。しかし、1989 年の LeCun の成果から 20 年間、主流のコンピュータビジョンでは特徴抽出（feature extraction）と分類を切り離していた。すなわち、SIFT (Lowe 2004)、SURF (Bay et al. 2006)、HOG (Dalal and Triggs 2005)、Bag-of-Visual-Words (Sivic and Zisserman 2003) といった人手で設計された記述子がピクセルを固定の特徴ベクトルへと写像し、それを線形分類器やカーネル分類器に入力していたのである。訓練損失は分類器のみを更新し、特徴を決して更新しなかったため、進歩は人間による特徴量エンジニアリングに依存していた。

転換点は 2012 年の ImageNet コンペティション（検証画像のうち正解ラベルがモデルの上位 5 件の予測に含まれない割合である **top-5 誤差**（top-5 error）によって 1,000 個の物体カテゴリを評価する競技会）で訪れた。**AlexNet** (Krizhevsky et al. 2012) は、8 層の畳み込みネットワークを生のピクセルからエンドツーエンドで訓練したモデルである。コンペティションへの出場システムはこのようなネットワーク 7 個のアンサンブルであり、top-5 誤差を（人手で設計された最良のパイプラインによる）26.2% から一躍 15.3% へと引き下げた。単一のネットワークだけでもすでに約 18% に達していた。アーキテクチャの面では AlexNet は LeNet を大規模化したものであるため、この劇的な勝利は 2012 年に揃った 3 つの推進要因を直接示している。すなわち、100 万枚規模のラベル付きデータセット（ImageNet）、数千万個の重みを数か月ではなく数日で訓練できるほど高速な GPU、そして ReLU 活性化関数、ドロップアウト、データ拡張（data augmentation）といった、深いネットワークを訓練可能にし暗記しにくくした重要なモデリング手法の 3 つである。

7.1.1 学習された特徴、データ、ハードウェア

人手で設計された記述子が見落とし、誤差逆伝播法（backpropagation）が ImageNet 上で発見したものは何だったのだろうか。Figure 7.1 に、AlexNet の最初の畳み込み層が学習した 96 個の 11×11 カーネルを示す。方向選択性を持つエッジ検出器や周波数検出器と並んで反対色ブロブ（color-opponent blob）が見られ、これらは分類損失の最小化のみによって見いだされ、より深い層によってテクスチャ、部品、物体カテゴリへと合成される。

7.1.1.1 データとハードウェア

6000 万個のパラメータを持つモデルは、PASCAL VOC（約 10^4 枚の画像）のような小規模なベンチマークをすぐに暗記してしまう。2012 年の ImageNet の訓練データ分割 (Deng et al. 2009; Russakovsky et al. 2015) は 1,000 カテゴリにわたる 120 万枚のラベル付き写真を提供し、さらに AlexNet はデータ拡張によって実効的なサンプル多様性を何倍にも高めた。具体的には、224×224 のランダムクロップと水平反転を 256×256 の画像から抽出することで、写真 1 枚あたり最大 $32 \cdot 32 \cdot 2 = 2{,}048$ 通りのビューを生成した。

ブレークスルーのもう一方の立役者はハードウェアであった。あらゆる空間位置とチャネルにわたってフィルタを畳み込む処理では、数百万回の独立した積和演算（multiply-add）を実行する。これはまさに、高帯域幅メモリに接続された数百個の並列コアで実行するために GPU

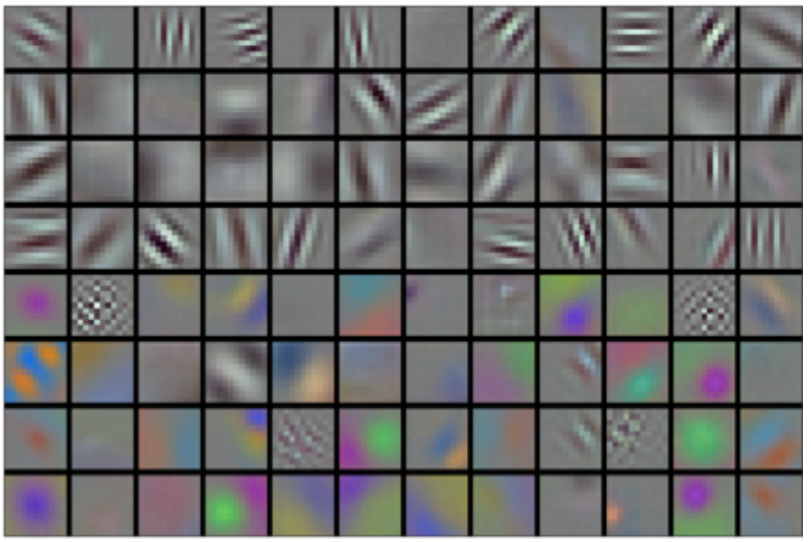


Figure 7.1: ImageNet 上で学習された、AlexNet の最初の畳み込み層における 96 個のカーネル。Krizhevsky et al. (2012) の厚意により転載。

（graphics processing unit）が設計されたワークロードそのものである。AlexNet に使われた 2 基の NVIDIA GTX 580 GPU は、それぞれ約 1.6 TFLOPS の演算性能と 192 GB/s のメモリ帯域幅を備えていたのに対し、当時のクアッドコア CPU は約 0.2 TFLOPS と 20 GB/s にとどまっていた。224×224 の RGB 画像 1 枚に対する AlexNet の 1 回の順伝播には約 10^9 回の積和演算がかかる。逆伝播を順伝播の 2 倍、積和演算 1 回あたり 2 FLOPs と数えると、90 エポックにわたって 120 万枚の画像を訓練するには次の演算量が必要になる。

$$10^9 \cdot 3 \cdot 2 \cdot 1.2 \times 10^6 \cdot 90 \approx 6.5 \times 10^{17} \text{ FLOPs} \tag{7.1}$$

これには 2 基の GTX 580 上で Krizhevsky のカスタム CUDA カーネル（cuda-convnet）を用いて 5〜6 日を要したが、CPU では優に 1 か月以上かかる計算量であった。

7.1.2 AlexNet のアーキテクチャ

Figure 7.2 では、LeNet（6.6 節）と、以下で Fashion-MNIST 上で訓練する AlexNet の変種を並べて比較している。どちらも畳み込みステージの積み重ねから密結合の分類ヘッドへとつなぐ同じ大局的な設計に従っているが、AlexNet はあらゆる次元を拡大するとともに、活性化関数、プーリング、正則化という 3 つの重要な構成要素を近代化している。

7.1.2.1 深さ、幅、ReLU、ドロップアウト

AlexNet はパラメータを持つ層の数を LeNet の 5 層から、5 つの畳み込み層と 3 つの密結合層からなる 8 層へと拡張している（Figure 7.2）。224×224 の入力は 28×28 のサムネイルより幅が 8 倍大きいため、第 1 層にはストライド 4 の 11×11 カーネル、第 2 層には 5×5 カーネル、第 3〜5 層には 3×3 カーネルを用い、第 1・第 2・第 5 の畳み込みの後に領域が重なり合う 3×3（ストライド 2）の最大プーリングを配置する。チャネル幅は 96 から 384 へと増え（LeNet では 6 と 16）、2 つの隠れ密結合層はそれぞれ 4096 ユニットへと広がる（LeNet では 120 と 84）。本書の実装では、2012 年当時の歴史的な事情による 2 つの要素は省略する。3 GB の GPU 2 基にわた

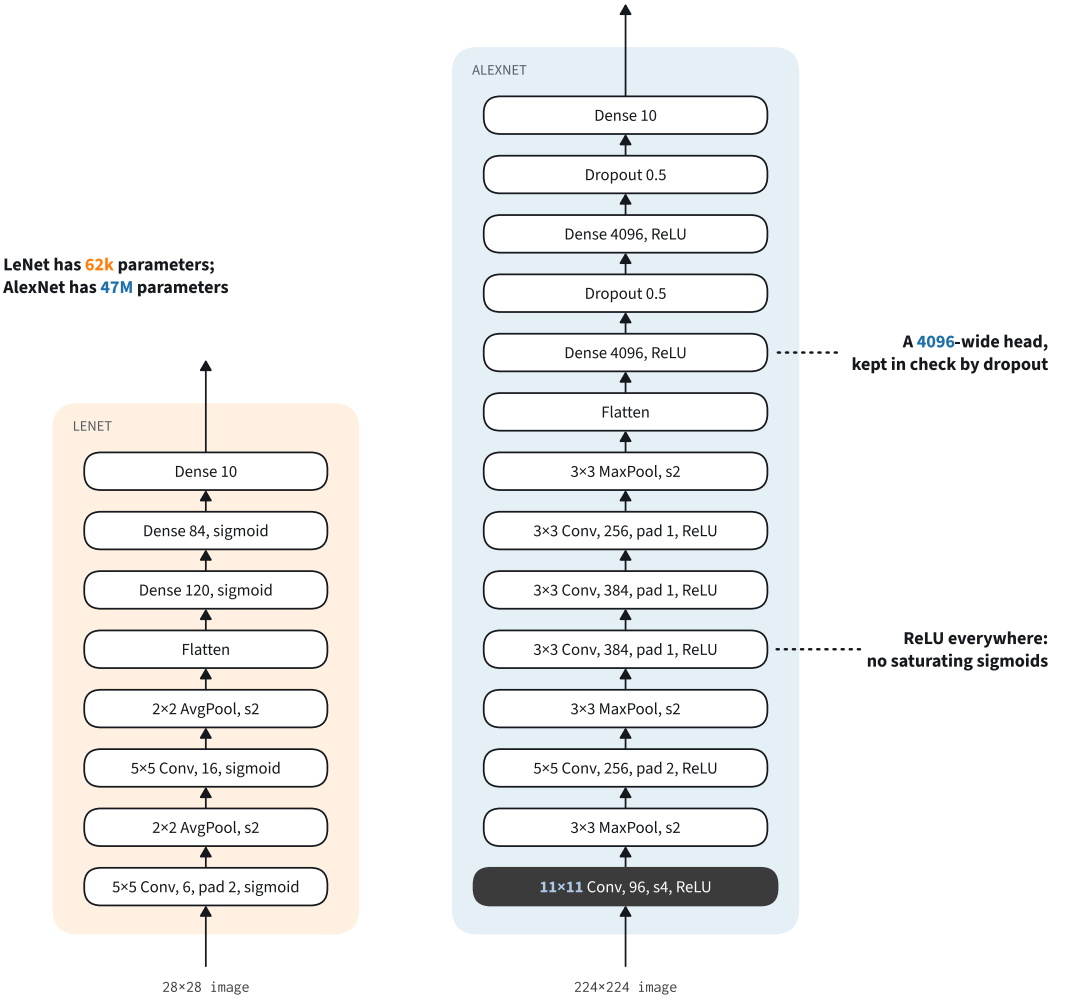


Figure 7.2: 6.6 節で構築した LeNet（左）と本節で構築する AlexNet（右）。いずれも単一チャネルの Fashion-MNIST 画像と 10 クラスを対象としており、入力は下端にある。AlexNet はより深く幅広であり、ReLU 活性化関数と最大プーリングを用い、密結合層をドロップアウトで正則化する。

る層の分割と、局所応答正規化（local response normalization）であり、後者は 7.3 節で扱うバッチ正規化（batch normalization）に取って代わられている。

この大規模化したネットワークを有効に機能させているのは、3 つのモデリング上の選択である。最初の 2 つは AlexNet 自身のものであり、3 つ目は本書による改良である。

1. **ReLU 活性化関数：**飽和するシグモイド（導関数が $\leq 1/4$ であるため深い層の勾配を 4^{-K} 倍に減衰させる）を $\mathrm{ReLU}(x) = \max(x, 0)$ に置き換えることで、活性なユニットでの導関数が 1 に保たれ、CIFAR-10 の訓練誤差 25% に tanh より 6 倍速く到達した (Krizhevsky et al. 2012)。
2. **ドロップアウト：**4096 ユニットの密結合層の両方の後にドロップアウト（$p = 0.5$、4.6 節）を適用することで、分類器ヘッドが訓練セットを暗記するのを防ぐ。
3. **He 初期化：**AlexNet は標準偏差 0.01 に固定したガウス分布から重みを抽出していた。本書ではこれを、3 年後に He et al. (2015) が ReLU ネットワーク向けに導出した初期化に置き換える。ReLU は入力の約半分を 0 にするため、重みを $\mathcal{N}(0, 2/n_{\text{in}})$（4.4.2.2 節、畳み込みでは $n_{\text{in}} = c_{\text{i}}k^2$）から抽出し、バイアスは 0 に初期化する。

NNX の層はコンストラクタ内で重みを抽出するため、すべての層に初期化器を渡す。

```
class AlexNet(d2l.Classifier):
    def __init__(self, lr=0.1, num_steps=4690, num_classes=10,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = (nnx.Rngs(params=d2l.get_key(), dropout=d2l.get_key())
                if rngs is None else rngs)
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        init = dict(kernel_init=he, rngs=rngs)

        def max_pool(x):  # 3x3 windows, stride 2
            return nnx.max_pool(x, window_shape=(3, 3), strides=(2, 2))

        def flatten(x):  # One vector per example
            return x.reshape((x.shape[0], -1))

        self.net = nnx.Sequential(
            nnx.Conv(1, 96, kernel_size=(11, 11), strides=4, padding='VALID',
                     **init), nnx.relu, max_pool,
            nnx.Conv(96, 256, kernel_size=(5, 5), **init), nnx.relu,
            max_pool,
            nnx.Conv(256, 384, kernel_size=(3, 3), **init), nnx.relu,
            nnx.Conv(384, 384, kernel_size=(3, 3), **init), nnx.relu,
            nnx.Conv(384, 256, kernel_size=(3, 3), **init), nnx.relu,
            max_pool,
            flatten,  # 5 x 5 x 256 values per image
            nnx.Linear(5 * 5 * 256, 4096, **init), nnx.relu,
            nnx.Dropout(0.5, rngs=rngs),
            nnx.Linear(4096, 4096, **init), nnx.relu,
            nnx.Dropout(0.5, rngs=rngs),
            nnx.Linear(4096, num_classes, **init))
```

コードが Figure 7.2 と一致していることを確認するため、単一チャネルの 224×224 のダミー画像を `layer_summary` に通し、各出力テンソルの形状を調べる。

```
AlexNet().layer_summary((1, 224, 224, 1))
```

```
Conv                 (1, 54, 54, 96)
relu                 (1, 54, 54, 96)
max_pool             (1, 26, 26, 96)
Conv                 (1, 26, 26, 256)
relu                 (1, 26, 26, 256)
max_pool             (1, 12, 12, 256)
Conv                 (1, 12, 12, 384)
relu                 (1, 12, 12, 384)
Conv                 (1, 12, 12, 384)
relu                 (1, 12, 12, 384)
Conv                 (1, 12, 12, 256)
relu                 (1, 12, 12, 256)
max_pool             (1, 5, 5, 256)
flatten              (1, 6400)
Linear               (1, 4096)
relu                 (1, 4096)
Dropout              (1, 4096)
Linear               (1, 4096)
relu                 (1, 4096)
Dropout              (1, 4096)
Linear               (1, 10)
```

すべての次元が 6.3 節の公式と一致している。最初のストライド 4 の畳み込みは 224 を $\lfloor (224-11)/4 \rfloor + 1 = 54$ に写し、領域が重なり合う 3×3（ストライド 2）の各最大プーリング層は n を $\lfloor (n-3)/2 \rfloor + 1$ に写す（$54 \to 26 \to 12 \to 5$）。中間のパディング付き畳み込みは空間次元を保つため、最終的な畳み込みステージは平坦化されて $256 \times 5 \times 5 = 6400$ 要素のベクトルとなり、密結合の分類器ヘッドに入力される。

7.1.2.2 パラメータと積和演算

各層の形状が確認できたところで、LeNet（6.6 節）のときと同様にパラメータ数と算術演算コストを集計しよう。畳み込み層は $c_{\mathrm{i}} c_{\mathrm{o}} k^2$ 個の各重みをすべての空間出力位置で再利用するのに対し、全結合層（fully connected layer）は 1 例あたり各重みを 1 回だけ乗算する。単一チャネル・10 クラスの AlexNet についての結果を Table 7.1 にまとめる。

Table 7.1: 単一チャネルの 224×224 画像 1 枚・10 クラスに対する AlexNet の各層のパラメータ数と 1 例あたりの積和演算数。プーリング層、活性化層、ドロップアウト層にはパラメータがない。

層	出力	パラメータ数	積和演算数
11×11 畳み込み、1 → 96、ストライド 4	96@54×54	11,712	33.9M
5×5 畳み込み、96 → 256	256@26×26	614,656	415.3M
3×3 畳み込み、256 → 384	384@12×12	885,120	127.4M
3×3 畳み込み、384 → 384	384@12×12	1,327,488	191.1M
3×3 畳み込み、384 → 256	256@12×12	884,992	127.4M

層	出力	パラメータ数	積和演算数
全結合、6400 →4096	4096	26,218,496	26.2M
全結合、4096 →4096	4096	16,781,312	16.8M
全結合、4096 →10	10	40,970	0.04M
合計		46,764,746	938.1M

パラメータと計算量の配分は、LeNet よりもさらに極端に偏っている。5 つの畳み込み層が占めるのは AlexNet の 4700 万個のパラメータのうちわずか 8% であるが、積和演算の 95% を実行している。一方、3 つの全結合層はパラメータの 92%（32 ビット浮動小数点で約 170 MB）を消費しながら、算術演算のわずか 5% しか担っていない。画像 1 枚あたり 61,706 個のパラメータと 416,520 回の積和演算であった LeNet と比べると、AlexNet はパラメータ数を約 $760\times$、算術演算量を $2{,}250\times$ に拡大している。ImageNet（RGB 3 チャネル、1000 クラス）では、完全なモデルは約 6000 万個のパラメータに達するが、依然としてその大半が同じ密結合層に集中している。

7.1.3 Fashion-MNIST での訓練

現代の GPU でも ImageNet 上で AlexNet を訓練するには数時間かかり、さらに約 150 GB のデータセットと、GPU にデータを供給し続ける高速な入力パイプライン（input pipeline）が必要になる。そこで本節では、対話的に実験を進められるよう Fashion-MNIST 上で訓練する。Fashion-MNIST 本来の 28×28 画像は小さすぎて、11×11（ストライド 4）の第 1 層と 3 回のプーリングステージを通過できないため、224×224 へと `resize` 引数（`d2l.FashionMNIST` の指定）によってアップサンプリングする。実応用においては、画素数を 64 倍にアップサンプリングしても視覚的な情報は一切増えず計算量だけが増大するが、ここでは変更なしの AlexNet アーキテクチャをそのまま動かすためだけにこのオーバーヘッドを受け入れる。

バッチサイズ 128 のミニバッチ SGD を用いて、10 エポック（LeNet の 30 エポックの 3 分の 1）訓練する。ReLU 活性化関数のおかげでシグモイドによる減衰なしに勾配が流れるため、LeNet の 2.0 よりはるかに小さい学習率できれいに学習が進む。

7.1.3.1 学習率スケジュール

固定された学習率が訓練の初期と終盤の両方で最適になることはまれである。重みがランダムな初期段階では、過度に大きなステップは最適化を不安定にしやすく、終盤近くでは大きなステップによってパラメータが最小点の周囲で跳ね回り、そこへ落ち着かなくなる。AlexNet の原論文の著者らは、検証誤差（validation error）が頭打ちになるたびに手動で学習率を 10 で割って減衰させていた。その代わりに本書では、現代の標準となっている滑らかで人手のいらない学習率スケジュール（learning-rate schedule）を採用し、すべてのモデルを公平に比較できるよう本章の残り全体でもこれとまったく同じスケジュールを再利用する。具体的には、線形の**ウォームアップ**によって学習率を 0 からピーク値 η_0 まで最初の t_{w} ステップかけて引き上げ、続いて半コサイン減衰（half-cosine decay）により 0 へと最終ステップ T で戻す。

$$\eta_t = \frac{\eta_0}{2}\left(1 + \cos\frac{\pi(t - t_{\mathrm{w}})}{T - t_{\mathrm{w}}}\right) \quad \text{for } t \geq t_{\mathrm{w}}. \tag{7.2}$$

$t = t_{\mathrm{w}}$ ではコサイン項が 1 に等しくなり $\eta_t = \eta_0$ となる。減衰の中間地点では $\eta_0/2$ に下がり、$t = T$ で 0 に達する。緩やかにウォームアップすることで、ステップ 1 からいきなり使うには危険なほど高いピーク学習率を利用できるようになる。ここではピーク学習率 $\eta_0 = 0.03$ で AlexNet を訓練する（7.2 節の各アーキテクチャでも同様に、安定して訓練できる最大のピーク学習率を用いる）。以下の `cosine_warmup` はステップ t における乗算スケジュール係数を計算する関

数である。本章全体で再利用するためこれを d2l に保存しておくが、学習率スケジュールについては 9.7 節で体系的に学ぶ。Figure 7.3 にこのスケジュールの形状を示す。

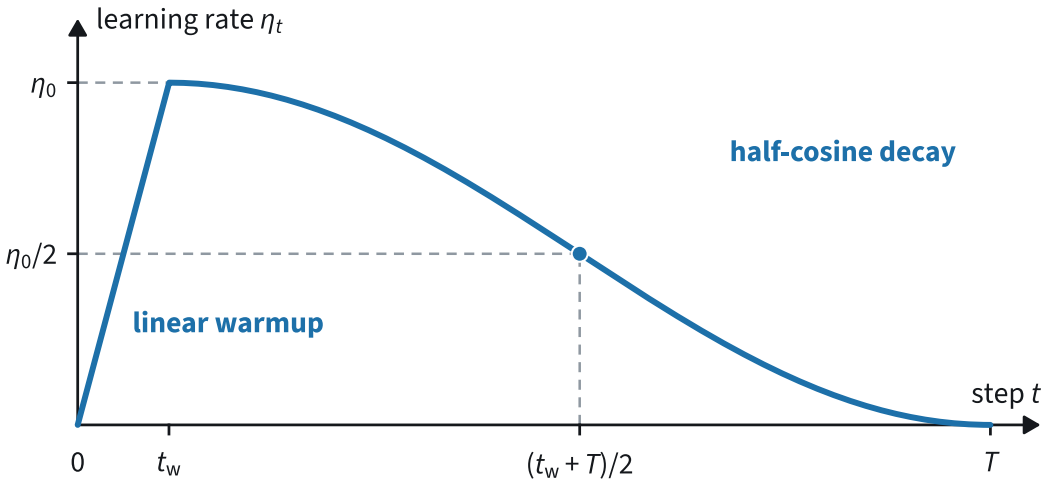


Figure 7.3: 本章全体で使用する学習率スケジュール。線形ウォームアップが学習率を 0 からピーク値 η_0 まで最初の t_w ステップ（ここでは訓練全体の 10 分の 1）かけて引き上げる。続いて Equation 7.2 の半コサイン減衰がこれを 0 へと最終ステップ T で引き下げ、減衰の中間地点で $\eta_0/2$ を通る。

```
def cosine_warmup(t, num_steps, warmup_steps):
    """Learning-rate factor: linear warmup from 0, then a half-cosine to 0."""
    t = jnp.asarray(t, jnp.float32)  # Arguments may be traced arrays
    warmup = jnp.where(t < warmup_steps, t / warmup_steps, 1.0)
    progress = jnp.clip((t - warmup_steps) / (num_steps - warmup_steps), 0, 1)
    return warmup * 0.5 * (1 + jnp.cos(jnp.pi * progress))
```

AlexNet はミニバッチごとにこのスケジュールを更新する。num_steps は実行全体の総ステップ数 T を指定し、ウォームアップは訓練の最初の 10 分の 1（10 エポック中の 1 エポック）を占める。

```
@d2l.add_to_class(AlexNet)
def configure_optimizers(self):
    return optax.sgd(lambda t: self.lr * cosine_warmup(
        t, self.num_steps, self.num_steps // 10))
```

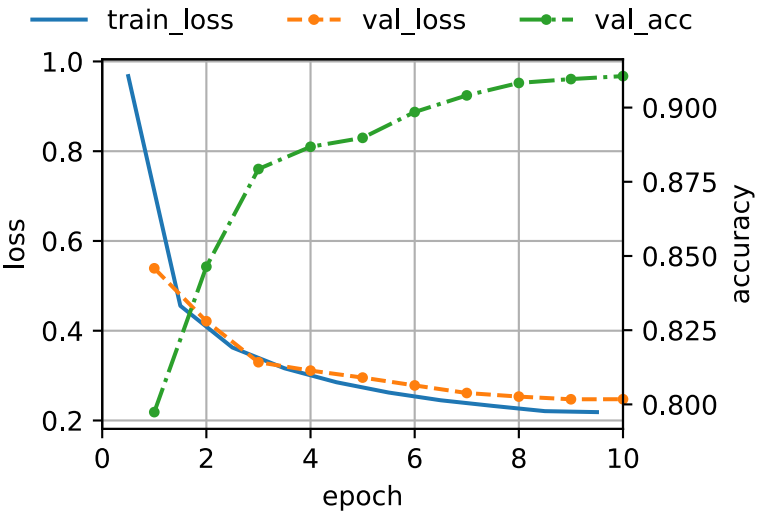
7.1.3.2 LeNet との比較

訓練ループを実行する前に、あらかじめ見通しを立てておこう。Fashion-MNIST には AlexNet の 4700 万個のパラメータを十分に活用するだけのデータセット規模も視覚的な複雑さもないため、LeNet に対する正解率（accuracy）の向上はわずかなものにとどまると予想すべきである。ここでの目的は、実装を検証し、計算プロファイルを確認し、第 1 層が何を学習するかを調べることにある。その観察を可能にするため、訓練開始前に第 1 層の 96 個の初期カーネルをスナップショットとして保存し、最適化によってそれらがどう変化するかを正確に測れるようにしておく。

```
def first_kernels(model):
    """The 96 kernels of 11 x 11 weights of the first layer, as NumPy."""
    kernel = d2l.numpy(model.net.layers[0].kernel[...])  # (11, 11, 1, 96)
    return kernel[:, :, 0].transpose(2, 0, 1)

data = d2l.FashionMNIST(batch_size=128, resize=(224, 224))
model = AlexNet(lr=0.03, num_steps=10 * len(data.train_dataloader()))
K0 = first_kernels(model)
```

初期の 11 × 11 カーネルを `K0` に記録したところで、`AlexNet` をリサイズ済みの Fashion-MNIST データセット `data` 上で 10 エポック訓練する。

```
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
```



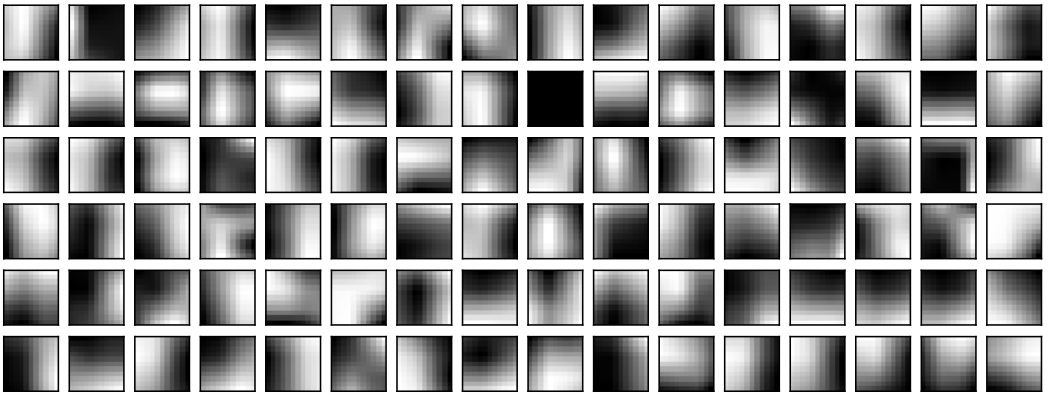
10 エポック後、AlexNet は約 91% の検証正解率に達し、訓練損失は 0.22 付近、検証損失（validation loss）は 0.25 付近となる。コサインスケジュールが終盤のエポックで学習率を 0 に向けて徐々に減衰させるため、終盤での正解率の変動は 1 パーセントポイント未満に収まり、最終エポックの値が実行全体の信頼できる要約となる。30 エポック後に約 88〜90% であった LeNet と比べると、AlexNet は 3 分の 1 のエポック数で 1〜2 パーセントポイント上回るが、その代償として画像 1 枚あたり 2,250× の算術演算を要している。しかも、そのわずかな改善でさえ、モデル容量（model capacity）の大きさと同じくらい ReLU 活性化関数、He 初期化、学習率スケジュールに負うところが大きい。

7.1.3.3 学習された第 1 層

LeNet の 5 × 5 カーネルは小さすぎて視覚的に読み取れなかったが（6.6 節）、AlexNet のサイズ 11 × 11 の 96 個のカーネルは、ImageNet 上の Figure 7.1 で見たように、小さな画像として直接解釈できる十分な大きさがある。このモデルが Fashion-MNIST 上で何を学習したかを見るため、訓練後の第 1 層のカーネルからランダムな初期重み `K0` を引き、各重みの差分をそれぞれのグレースケール範囲でプロットする。

```
change = first_kernels(model) - K0
d2l.show_images(change, 6, 16, scale=0.6, cmap='gray')
print(f'mean size of the change: {np.abs(change).mean() / np.abs(K0).mean():.3f}'
      ' of the mean size of the initial weights')
```

```
mean size of the change: 0.049 of the mean size of the initial weights
```



10 エポックの訓練で第 1 層の重みが変化する量は初期の大きさの約 4% にすぎないため、訓練後のカーネルをそのままプロットしても、大部分は初期のランダムノイズのように見えてしまう。しかし、**重みの更新量**そのものには明確な構造が現れている。滑らかな輝度勾配や幅広の方向性バーであり、これらは Figure 7.1 で見た鮮明な ImageNet のエッジフィルタに対応する低周波版である。欠けている 2 つの要素（細かな高周波エッジと色ブロブ）はいずれも入力データから直接説明できる。ここでの画像はグレースケールであり、また縦横 8 倍のアップサンプリングによって約 8 ピクセルより細かい本来のディテールが存在しないため、11×11 カーネルからは緩やかな輝度勾配として捉えられるのである。要するに、第 1 層はそのデータセットの空間統計量に忠実に適応している。

最後に、Table 7.1 で見たいくつかの数値を振り返ろう。AlexNet のパラメータの 92% は巨大な全結合ヘッドに存在し、算術演算にはほとんど寄与しない一方で暗記を抑えるために強いドロップアウトを必要としていた。次節では、再利用可能なブロックから深いアーキテクチャを体系的に構築する方法、そしてあの巨大な密結合ヘッドがそもそも必要なのかどうかを問う。

7.1.4 まとめ

エンドツーエンドで学習される視覚表現が人手で設計された特徴パイプラインを凌駕したのは、2012 年に 3 つの要因が揃ったときであった。大規模なラベル付きデータセット（ImageNet）、数千万個のパラメータを数日で訓練できる超並列 GPU、そして近代的な訓練の構成要素（ReLU 活性化関数、ドロップアウト、データ拡張）である。これらの道具を用いて LeNet の基本テンプレートを拡大した AlexNet の出場システムは、ImageNet の top-5 誤差を 26.2% から 15.3% へと大幅に引き下げた。層ごとの内訳を集計すると際立った非対称性が明らかになり、畳み込みの幹線部が算術演算の 95% を実行する一方、密結合ヘッドがパラメータの 92% を保持している。これほど膨大なモデル容量を使い切れない Fashion-MNIST 上では、AlexNet は画像あたり約 2,250× の算術演算を費やしつつ、3 分の 1 のエポック数で LeNet を 1〜2 パーセントポイント上回る。

7.1.5 演習問題

1. **メモリと計算量の内訳**。32 ビット浮動小数点において Table 7.1 を `AlexNet` が必要とするメモリ量へと拡張せよ。
    1. 224×224 画像 1 枚の順伝播が保存する活性化値の個数を、すべての畳み込み層、プーリング層、全結合層の出力を数えて層ごとに計算せよ。活性化メモリはどこに集中しており、パラメータのメモリはどこに集中しているか。
    2. 訓練中には、パラメータ、その勾配、およびミニバッチ全体の保存された活性化値がメモリに収まらなければならない。どのバッチサイズで、活性化メモリがパラメータと勾配のメモリを上回るか。
    3. Table 7.1 のパラメータ数の計算を ImageNet 版（3 入力チャネル、1000 クラス、227×227 入力により最終畳み込みステージが 6×6（5×5 ではなく）になる設定）に対して繰り返し、合計を Krizhevsky et al. (2012) で報告されている 6000 万個のパラメータと比較せよ。
2. **初期層の受容野**。カーネルサイズ k_i とストライド s_i を持つ層の積み重ねにおいて、最上位のユニットが持つ受容野の一辺は $r = 1 + \sum_i (k_i - 1) \prod_{j<i} s_j$ で与えられる（6.3 節）。各最大プーリング層もそれぞれのカーネルサイズとストライドを持つ層として数え、`AlexNet` の 5 つの畳み込み層それぞれの直後における 1 ユニットの受容野を計算せよ。各受容野は 224×224 の入力の一辺に対してどの割合を占めるか。また、いずれかの畳み込み層のユニットは画像全体を見ているか。
3. **演算密度**。●層の**演算密度**とは、メモリから読み出すパラメータ 1 バイトあたりにその層が実行する積和演算の回数である。Table 7.1 を用いて次を答えよ。
    1. 画像 1 枚のバッチと画像 128 枚のバッチのそれぞれについて、`AlexNet` のすべての層の演算密度を計算せよ（各パラメータは 1 バッチにつき 1 回読み出されると仮定する）。
    2. メモリから 1 バイトを読み出す時間に 10 回の積和演算を実行するアクセラレータを考え、各層の処理時間は計算時間とパラメータ読み出し時間の最大値で決まるとする。画像 1 枚の推論（$B = 1$）をより高速化するのは、計算スループットを 2 倍にすることか、それともメモリ帯域幅を 2 倍にすることか。また、$B = 128$ での訓練をより高速化するのはどちらか。
4. [code] **バッチサイズ、スループット、メモリ**。●バッチサイズのみを変えながら、本節と同様に `AlexNet` を訓練せよ。
    1. 問題 1 を用いて、手元の GPU のメモリに収まる最大の訓練バッチサイズを予測せよ。
    2. バッチサイズ 32 から始めて 2 倍ずつ増やし、メモリ不足にならずに訓練できる最大のバッチサイズを見つけ、予測との差が生じる主な要因を説明せよ。
    3. その上限までの少なくとも 4 通りのバッチサイズにわたって訓練スループット（枚/秒）を測定せよ。どのバッチサイズを超えるとスループットが頭打ちになるか。また、それはなぜか。

7.2 ブロック、ボトルネック、ブランチ：VGG、NiN、GoogLeNet

AlexNet（7.1 節）は 1 層ずつ手作業で設計されていた。ここには 11×11 カーネル、あそこには 384 チャネルを置き、1 番目、2 番目、5 番目の畳み込みの後に最大プーリングを挿入するといった具合である。8 層のモデルであれば全層を手動で調整することも可能だが、20 層以上へとスケールさせるには体系的な設計規則が求められる。同様に注目すべき点として、AlexNet は 4700 万個のパラメータのうち 92% を全結合分類ヘッドに集中させているが、この部分は演算量のわ

ずか 5% しか担っておらず、訓練セットの丸暗記を防ぐために強いドロップアウトに頼っている。2013〜2014 年にかけて、3 つの画期的なアーキテクチャがこれらのボトルネックに順次取り組んだ。今日の実務でこれらのモデルがそのまま使われることはほとんどないが、そこで導入された構造上のアイデアは、現代のほぼすべての畳み込みネットワークに受け継がれている。

第 1 に、**VGG** は設計の単位を個々の層から、小さな 3×3 畳み込みとそれに続くプーリングからなる反復可能な**ブロック**へと移行させ、アーキテクチャ全体をブロック仕様の簡潔なリストとして表現できるようにした (Simonyan and Zisserman 2015)。第 2 に、**Network in Network**（NiN）は肥大化した全結合ヘッドを完全に排除し、1×1 畳み込みによって各ピクセルでチャネルを混合したうえで、グローバル平均プーリング（global average pooling）により最終の特徴マップを直接クラススコアへと縮約した (Lin et al. 2013)。第 3 に、**GoogLeNet** は単一のカーネルサイズに固定するのを避け、各ブロック内で複数のフィルタサイズを並列に実行しつつ、チャネルを削減する 1×1 ボトルネックによって大きな畳み込みの計算コストを抑えた (Szegedy et al. 2015)。以下では、これら 3 つのアーキテクチャをすべて構築し、AlexNet と同様にパラメータ数と積和演算（multiply-add）の回数を数え、同一のコサインスケジュールのもとで Fashion-MNIST 上で訓練して横並びで比較する。

7.2.1 VGG：設計単位としてのブロック

7.2.1.1 3×3 畳み込みの積み重ね

もしすべての畳み込み層の後にストライド 2 のプーリングを置いたとすると、幅 d の入力は高々 $\lfloor \log_2 d \rfloor$ 層で 1×1 に潰れてしまう（$224\to112\to56\to28\to14\to7\to3\to1$）。ネットワークをより深くするには、ダウンサンプリングの前に、各空間解像度（spatial resolution）で形状を保つストライド 1 の畳み込みを複数層積み重ねる必要がある。

VGG は、左右および上下の隣接ピクセルを捉える最小の対称フィルタである 3×3 カーネルに標準化した (Simonyan and Zisserman 2015)。2 つ積み重ねた 3×3 層は、同じ 5×5 の受容野（receptive field）を 1 つの 5×5 層と同様にカバーしつつ、重みの数を $2\cdot9c^2=18c^2$ 個に抑えられる（単一層では $25c^2$ 個）。また、3 つ積み重ねた 3×3 層は、7×7 のウィンドウを $27c^2$ 個の重みでカバーし（単一層では $49c^2$ 個）、同時に ReLU の非線形性を 3 つ挿入できる（単一層では 1 つ）。**VGG ブロック**は、パディング付きの 3×3 畳み込みと ReLU を `num_convs` 個積み重ね、その後に 2×2（ストライド 2）の最大プーリングを続ける構造である。

```
def vgg_block(num_convs, in_channels, out_channels, init):
    layers = []
    for _ in range(num_convs):
        layers.append(nnx.Conv(in_channels, out_channels, kernel_size=(3, 3),
                               padding=(1, 1), **init))
        layers.append(nnx.relu)
        in_channels = out_channels
    layers.append(max_pool)
    return nnx.Sequential(*layers)

def max_pool(x):  # 2x2 windows, stride 2
    return nnx.max_pool(x, window_shape=(2, 2), strides=(2, 2))
```

7.2.1.2 VGG ネットワーク

完全な VGG ネットワークは、これらのブロックの列を連結し、その末尾に AlexNet の密な分類ヘッド（ドロップアウト付きの 4096 ユニットの隠れ層 2 つと、それに続く出力層）を接続する。Figure 7.4 に VGG-11 と AlexNet の対比を示す。5 段階のプーリングステージによって

224×224 の入力は半分ずつ縮小されて 7×7 になり、一方でチャネル幅はステージごとに倍増して 512 に達する。

すべての VGG ブロックは 2 つの整数（畳み込み層の数と出力チャネル数）で完全に指定できるため、畳み込みトランク全体は `(num_convs, out_channels)` というタプルのリスト（`arch`）として渡すことができる。VGG は単一の固定されたアーキテクチャを定義するのではなく、**パラメータ化されたモデル族**（parameterized family）を定義する。すなわち、VGG-11、VGG-16、VGG-19 はまったく同じクラスを共有し、`arch` だけが異なる。ネットワークのトポロジを手書きのコードではなくモジュール化されたデータとして扱うことこそ、VGG が残した最大の遺産である。7.1 節と同様に、本節のすべてのネットワークは重みを He 正規初期化し、バイアスを 0 に初期化する。

辞書 `init` は、重みの初期化子と乱数ストリームをすべての層に渡す。

```
class VGG(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_steps=4690, num_classes=10,
                 input_shape=(224, 224, 1), rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = (nnx.Rngs(params=d2l.get_key(), dropout=d2l.get_key())
                if rngs is None else rngs)
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        init = dict(kernel_init=he, rngs=rngs)
        conv_blks, in_channels = [], input_shape[-1]
        for num_convs, out_channels in arch:
            conv_blks.append(vgg_block(num_convs, in_channels, out_channels,
                                       init))
            in_channels = out_channels
        # Each block halves the height and the width
        height, width = (n // 2 ** len(arch) for n in input_shape[:2])
        self.net = nnx.Sequential(
            *conv_blks, *self.head((height, width, in_channels), init))

    def head(self, shape, init):  # AlexNet's fully connected head
        return [flatten,
                nnx.Linear(math.prod(shape), 4096, **init), nnx.relu,
                nnx.Dropout(0.5, rngs=init['rngs']),
                nnx.Linear(4096, 4096, **init), nnx.relu,
                nnx.Dropout(0.5, rngs=init['rngs']),
                nnx.Linear(4096, self.num_classes, **init)]

def flatten(x):  # One vector per example
    return x.reshape((x.shape[0], -1))
```

NNX の層は構築時に入力サイズを必要とするため、`VGG` は画像サイズを `input_shape` として受け取る。最初の全結合層は hwc 個の数値を読み込むが、これは $h \times w$ ピクセル、c チャネルの最終特徴マップに対応する。

元論文で最小の変種である VGG-11 は、(1, 1, 2, 2, 2) 個の畳み込み層と (64, 128, 256, 512, 512) チャネルを持つ 5 つのブロックを積み重ねており、8 つの畳み込み層と 3 つの全結合層を合わせて計 11 個のパラメータ付き層からなる。単一の 224×224 画像に対する中間テンソルの形状を確認してみよう。

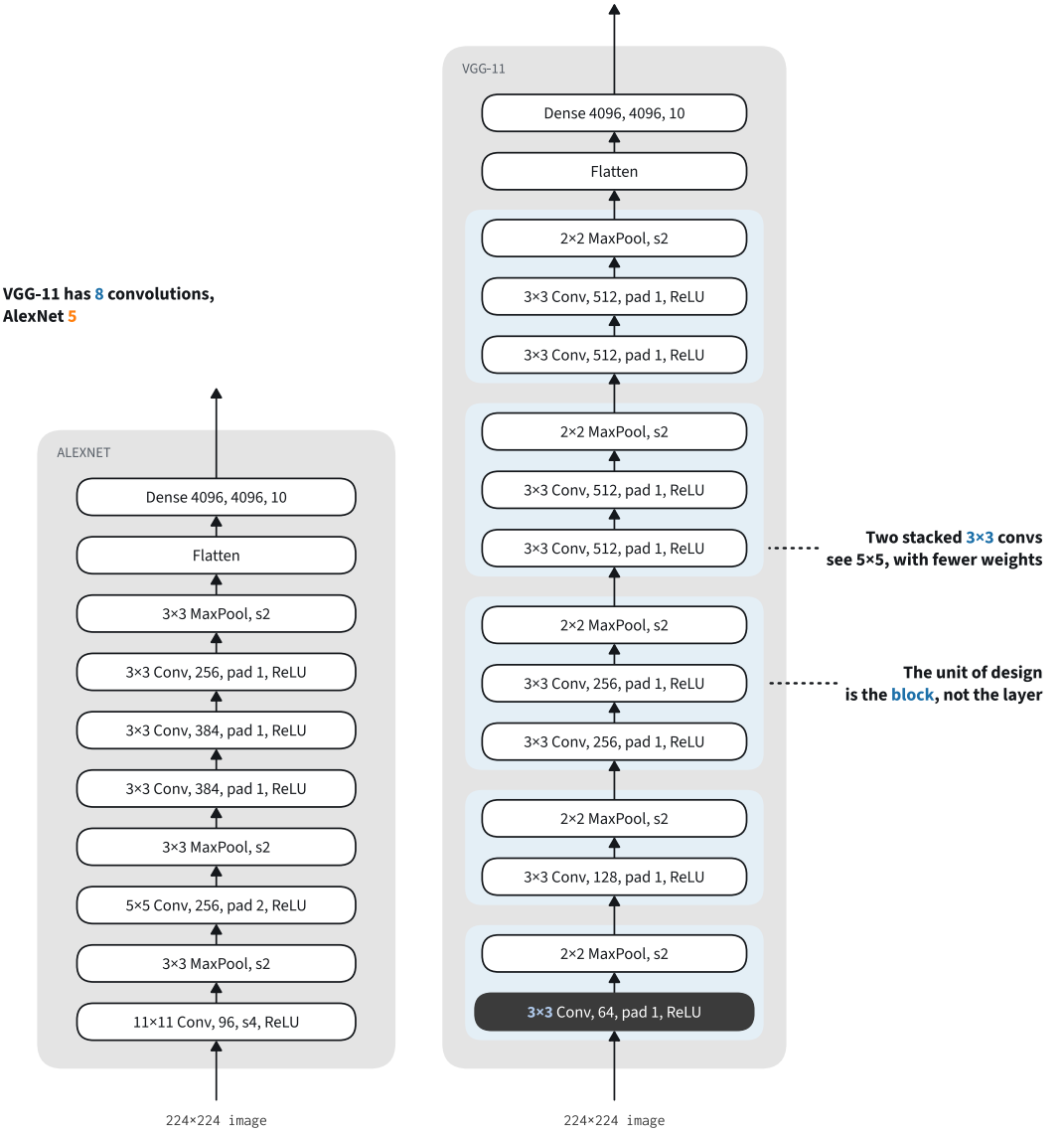


Figure 7.4: 単一チャネルの 224×224 画像と 10 クラスに対する AlexNet（左）と VGG-11（右）。入力は下側にある。AlexNet の層は 1 つずつ個別に選ばれているのに対し、VGG は複数の 3×3 畳み込みと最大プーリングからなる 1 種類のブロックを、チャネル数を増やしながら繰り返す。

```
VGG(arch=((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))).layer_summary(
    (1, 224, 224, 1))
```

```
Sequential            (1, 112, 112, 64)
Sequential            (1, 56, 56, 128)
Sequential            (1, 28, 28, 256)
Sequential            (1, 14, 14, 512)
Sequential            (1, 7, 7, 512)
flatten               (1, 25088)
Linear                (1, 4096)
relu                  (1, 4096)
Dropout               (1, 4096)
Linear                (1, 4096)
relu                  (1, 4096)
Dropout               (1, 4096)
Linear                (1, 10)
```

最終の畳み込みブロックは $512 \times 7 \times 7$ のテンソルを出力し、これが 25,088 個の特徴量へと平坦化されて密なヘッドに入力される。AlexNet と同様に各層を集計すると、VGG-11 は 1 億 2880 万個のパラメータを持ち、画像 1 枚あたり 75 億回の積和演算（AlexNet の計算量の 8 倍）を実行することがわかる。ここでもやはり、パラメータと計算量はまったく別の場所に偏在している。密なヘッドが全重みの 93% を占めており、最初の $25{,}088 \times 4096$ の行列だけで 1 億 300 万個のパラメータに達する。

7.2.1.3 訓練

Fashion-MNIST の実演に AlexNet の 8 倍もの計算量を費やすのは不必要に重いため、チャネル数を 16、32、64、128、128 とした 1/4 幅の VGG-11 を訓練する。このモデルの画像 1 枚あたりの積和演算は 5 億 1200 万回であり、AlexNet の 9 億 3800 万回の約半分に収まる。その他の訓練設定はすべて 7.1 節と同一とし、224×224 にアップサンプリングした画像、バッチサイズ 128、および 7.1 節のウォームアップ付きコサインスケジュールによる 10 エポックの SGD を用い、AlexNet の `configure_optimizers` を再利用する。

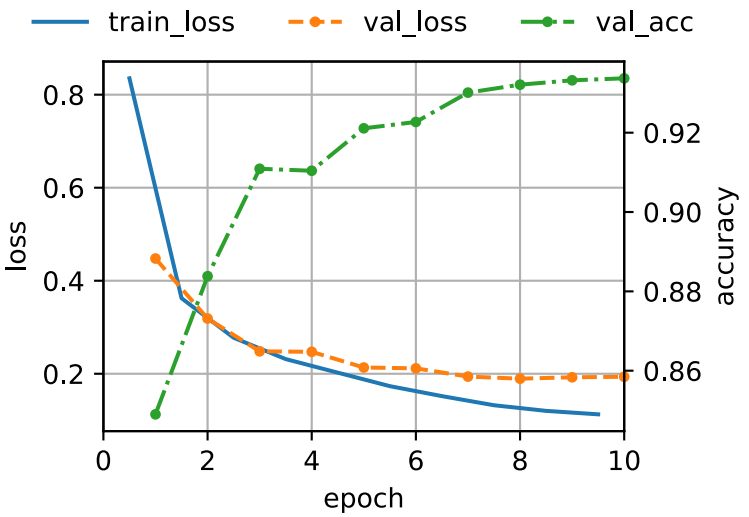
本節のネットワーク間で変更する唯一のハイパーパラメータは、ピーク学習率である。ピーク値を 0.1 にすると、以下で扱う正規化なしのアーキテクチャのいくつかは不安定になる寸前の状態に陥り、ときおり発散したりランダム推測の水準で停滞したりするため、それらのモデルは代わりに 0.03 で訓練する。1/4 幅の VGG は 0.1 でも安定して訓練でき、0.03 に比べて 1.5〜2 パーセントポイント高い精度が得られるため、0.1 で実行する。

```
@d2l.add_to_class(VGG)
def configure_optimizers(self):
    return optax.sgd(lambda t: self.lr * d2l.cosine_warmup(
        t, self.num_steps, self.num_steps // 10))
```

コサインウォームアップスケジュールを設定したうえで、1/4 幅の `VGG` を 224×224 の Fashion-MNIST 上で 10 エポック訓練し、検証正解率の軌跡を `val_acc['VGG']` に保存する。

```
data = d2l.FashionMNIST(batch_size=128, resize=(224, 224))
model = VGG(arch=((1, 16), (1, 32), (2, 64), (2, 128), (2, 128)), lr=0.1,
            num_steps=10 * len(data.train_dataloader()))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
```

```
trainer.fit(model, data)
val_acc = {'VGG': trainer.history['val_acc']}  # for the comparison below
```



この 1/4 幅の VGG は約 93% の検証正解率に達し、約半分の演算量で AlexNet を 2 パーセントポイント上回る。もっとも公平を期すなら、この差の多くは学習率の高さによるものである。後ほど確認するように、AlexNet と同じピーク値 0.03 では、同じ VGG でも 92% 弱にとどまる。しかし、畳み込みトランクの幅を狭めても解決しなかった問題に注目してほしい。畳み込み層のパラメータはわずか 60 万個まで縮小したにもかかわらず、密なヘッドは依然として $128 \times 7 \times 7 = 6272$ 個の活性化値を 4096 個の隠れユニットへと写像しており、モデル全体の 4310 万パラメータのうち 4250 万個（実に 99%）を占めている。ブロックによってネットワークのボディは規則的に整理されたが、ヘッドは依然として肥大化したままである。

7.2.2 NiN：1×1 畳み込みとグローバル平均プーリング

密な分類ヘッドが数千万個ものパラメータを消費するのは、すべてのチャネルのすべての空間座標と、すべての隠れユニットを結ぶ個別の重みを割り当てるからである。しかも、その膨大な容量は、最終の特徴マップをクラスのロジットへと要約するというただ 1 つの役割のためだけに使われている。**Network in Network**（NiN）は、この密なヘッドを完全に廃止しつつ、同時に各畳み込みブロック内の非線形性を高められないかと問う。標準的な畳み込みの後に ReLU を続けただけでは、各局所パッチにしきい値付きの線形フィルタを適用しているにすぎなかった。カーネルを大きくせずに同じ空間ウィンドウ上でより豊かな非線形関数を計算するには、ピクセルごとにチャネルを混合する層、まさに 1×1 畳み込み（6.4.3 節）を積み重ねればよい。

7.2.2.1 NiN ブロック

1×1 畳み込みはすべてのピクセルに対して同一の全結合層を独立に適用するため、空間畳み込みの後に 2 つの 1×1 畳み込み（それぞれ ReLU を伴う）を続けると、すべての空間位置に隠れ層 1 層の小さな MLP を配置することになる。これが **Network in Network** という名前の由来である。Figure 7.5 に VGG の密なヘッドと NiN の末尾の対比を示す。

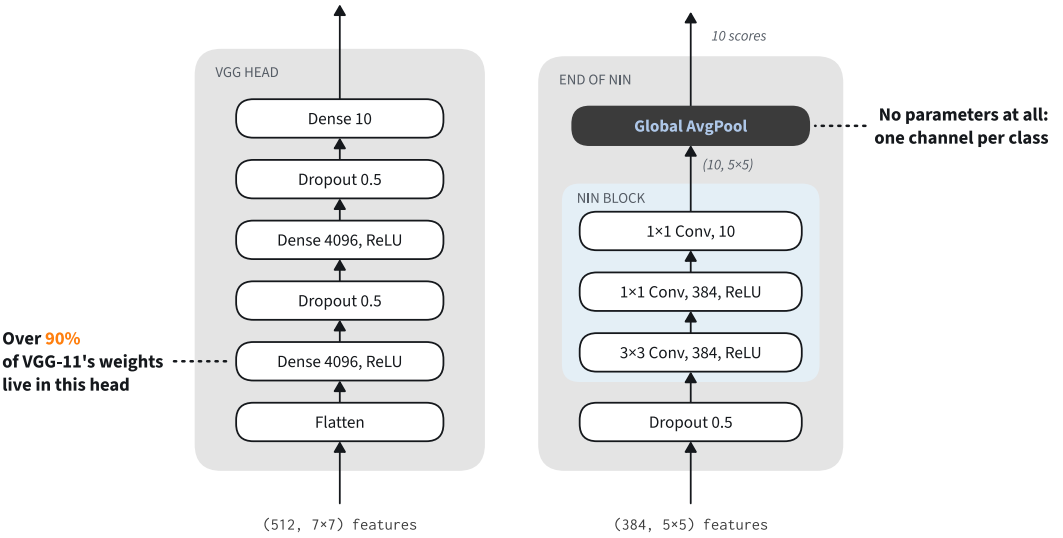


Figure 7.5: いずれも 10 クラス分類に対する VGG のヘッド（左）と NiN の末尾（右）。入力は下側にある。VGG は最終の特徴マップを平坦化して全結合層に入力する。NiN の最終ブロックは、ドロップアウトの後に 3×3 畳み込みと 2 つの 1×1 畳み込み（最後の畳み込みの後には ReLU を置かない）を適用してクラスごとに 1 チャネルを出力し、グローバル平均プーリングによって各チャネルを 1 つのスコアに変換する。

```
def nin_block(in_channels, out_channels, kernel_size, strides, padding,
              init, num_classes=None):
    layers = [
        nnx.Conv(in_channels, out_channels, kernel_size, strides=strides,
                 padding=padding, **init), nnx.relu,
        nnx.Conv(out_channels, out_channels, kernel_size=(1, 1), **init),
        nnx.relu]
    if num_classes is None:
        layers += [nnx.Conv(out_channels, out_channels, kernel_size=(1, 1),
                            **init), nnx.relu]
    else:  # Last block: one map of scores per class, no ReLU
        layers.append(nnx.Conv(out_channels, num_classes, kernel_size=(1, 1),
                               **init))
    return nnx.Sequential(*layers)
```

7.2.2.2 グローバル平均プーリング

バックボーンについては、NiN は AlexNet の序盤のカーネルサイズとチャネル数を踏襲しており、幅 96、256、384 の 11×11、5×5、3×3 畳み込みから始まるブロックを並べ、それぞれの後にストライド 2 の 3×3 最大プーリングを置く。しかし、ネットワークの最上部において NiN は AlexNet や VGG と決定的に袂を分かつ。ドロップアウトの後、4 番目の 384 チャネル NiN ブロックは出力チャネル数がクラス数（Fashion-MNIST では 10）に等しい 1×1 畳み込みで終わり、続いて**グローバル平均プーリング**がそれら 10 枚の特徴マップをそれぞれ全空間座標にわたって平均し、10 個のクラスロジットを直接生成する。プーリングヘッドにはパラメータが 1 つもないため、正解クラスに対応するチャネルで高い空間平均応答を出すよう、最終ブロック自体が学

習しなければならない。

ここで、実装上の重要な細部に注目してほしい。num_classes が nin_block に渡されたとき、最後の 1×1 畳み込みの出力は生のクラスロジットであるため、ReLU 活性化関数を省いている。もしロジットに ReLU を適用すると、あるクラスが永久に死んでしまう危険がある。あるクラスチャネルのすべての空間位置がデータセット全体にわたって負になった場合、そのロジットと勾配は恒等的に 0 となって消失し、SGD がそのクラスを二度と復活させられなくなるからである。

グローバル平均プーリングには、パラメータの削減に加えて概念上の第 2 の利点がある。畳み込みは平行移動同変（translation-equivariant）であり、全空間座標にわたる平均は置換に対して不変であるため、得られるクラススコアは（境界処理やストライドの影響を除けば）物体が画像内の**どこに**現れるかに依存しない。対照的に、平坦化された密なヘッドは空間セルごとに異なる重みを割り当てるため、あらゆる物体をあらゆる位置で学習し直さなければならない。

```
class NiN(d2l.Classifier):
    def __init__(self, lr=0.1, num_steps=4690, num_classes=10, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = (nnx.Rngs(params=d2l.get_key(), dropout=d2l.get_key())
                if rngs is None else rngs)
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        init = dict(kernel_init=he, rngs=rngs)

        def max_pool(x):  # 3x3 windows, stride 2
            return nnx.max_pool(x, (3, 3), strides=(2, 2))

        self.net = nnx.Sequential(
            nin_block(1, 96, (11, 11), (4, 4), (0, 0), init), max_pool,
            nin_block(96, 256, (5, 5), (1, 1), (2, 2), init), max_pool,
            nin_block(256, 384, (3, 3), (1, 1), (1, 1), init), max_pool,
            nnx.Dropout(0.5, rngs=rngs),
            nin_block(384, 384, (3, 3), (1, 1), (1, 1), init,
                      num_classes=num_classes),
            global_avg_pool)

def global_avg_pool(x):  # Average over height and width (NHWC)
    return x.mean(axis=(1, 2))
```

単一の 224×224 入力に対するテンソルの形状を追ってみよう。

```
NiN().layer_summary((1, 224, 224, 1))
```

```
Sequential              (1, 54, 54, 96)
max_pool                (1, 26, 26, 96)
Sequential              (1, 26, 26, 256)
max_pool                (1, 12, 12, 256)
Sequential              (1, 12, 12, 384)
max_pool                (1, 5, 5, 384)
Dropout                 (1, 5, 5, 384)
Sequential              (1, 5, 5, 10)
global_avg_pool         (1, 10)
```

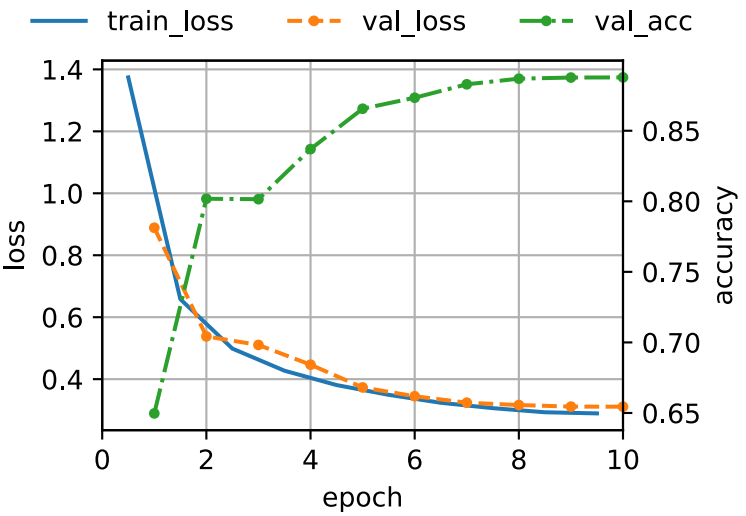
AlexNet と同様に、空間次元は 224 から 54、26、12、5 へと段階的に縮小し、その後に最終の

NiN ブロックが 10 枚の 5×5 スコアマップを出力して、グローバル平均プーリングがそれらを 10 個のロジットへと縮約する。密なヘッドを取り除くことで、パラメータ数はわずか 340 万個（AlexNet のサイズの 7% にすぎない）まで激減する一方、追加された 1×1 畳み込みによって計算が畳み込みボディへと移るため、画像 1 枚あたりの総演算量は積和演算 7 億 9800 万回と、AlexNet の 9 億 3800 万回と同程度にとどまる。

7.2.2.3 NiN の訓練

VGG と同じオプティマイザスケジュールで NiN を訓練する。ただし、0.1 ではときおり発散するため、ピーク学習率には 0.03 を用いる。

```
NiN.configure_optimizers = VGG.configure_optimizers  # AlexNet's schedule
data = d2l.FashionMNIST(batch_size=128, resize=(224, 224))
model = NiN(lr=0.03, num_steps=10 * len(data.train_dataloader()))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
val_acc['NiN'] = trainer.history['val_acc']
```



10 エポック後、NiN の検証正解率は約 89% に達するが、これは AlexNet より約 2 ポイント、1/4 幅の VGG より 4 ポイント低い。訓練損失は 0.30 付近で終了し（AlexNet では約 0.22）、検証損失もそれにほぼ追随していることから、10 エポックでは過学習を起こさずわずかに学習不足の状態にあることがわかる。単独のアーキテクチャとして見れば、NiN の性能は演算コストの割に控えめである。その永続的な影響力は、チャネルを混合する 1×1 畳み込みとグローバル平均プーリングヘッドという、再利用可能な 2 つの基本要素を広めた点にある。

7.2.2.4 プーリングヘッドを持つ VGG

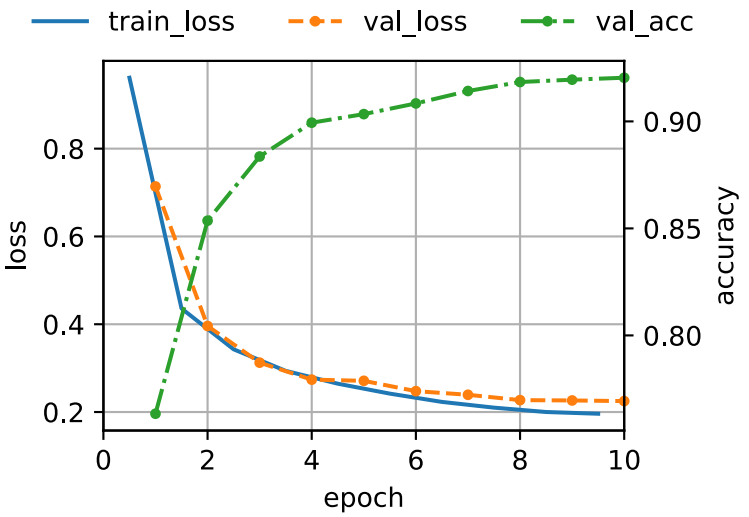
NiN は畳み込みボディと分類ヘッドの両方を同時に変更しているため、そのスコアだけでは密なヘッドが実際にどれほど寄与していたのかを切り分けられない。ヘッドの影響だけを純粋に取り出すため、1/4 幅の VGG ボディをまったく同じまま保ち、`VGG.head` のみをオーバーライドする。`PooledVGG` では、グローバル平均プーリングが最終ブロックの 128 個の出力チャネルを要約し、単

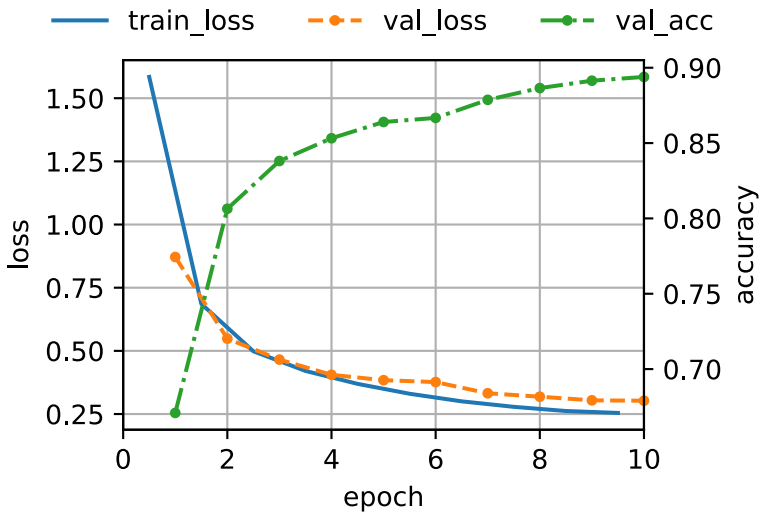
一の線形層がそれら 128 個の特徴量を 10 クラスへ直接写像する。これにより、ネットワーク全体の 4310 万パラメータのうち 4250 万個（および同数の積和演算）が削減される。

```
class PooledVGG(VGG):
    def head(self, shape, init):  # Global average pooling, one linear layer
        return [global_avg_pool,
                nnx.Linear(shape[-1], self.num_classes, **init)]
```

PooledVGG も NiN と同様に 0.1 では発散や崩壊を起こすことがあるため、VGG と PooledVGG の両方を揃えたピーク学習率 `lr=0.03` で訓練し、最終的な検証正解率を heads に保存する。

```
heads = {}  # Both heads at peak 0.03, the rate the pooled head tolerates
for net in (VGG, PooledVGG):
    model = net(arch=((1, 16), (1, 32), (2, 64), (2, 128), (2, 128)),
                lr=0.03, num_steps=10 * len(data.train_dataloader()))
    trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
    trainer.fit(model, data)
    heads[net.__name__] = round(trainer.history['val_acc'][-1], 3)
val_acc['PooledVGG'] = trainer.history['val_acc']
heads
```






```
{'VGG': 0.92, 'PooledVGG': 0.894}
```

PooledVGG の検証正解率は 89〜90% となり、密なヘッドを持つ VGG（同じピーク値 0.03 で 91〜92%）を約 2 パーセントポイント下回り、訓練損失も高くなる（約 0.25 対約 0.20）。したがって、Fashion-MNIST 上で 10 エポック訓練したこの幅の狭い畳み込みボディにおいては、密なヘッドの追加容量はモデル全体のパラメータの 99% と引き換えに、正解率で約 2 ポイント分の働きをしていることになる。

7.2.3 GoogLeNet：マルチブランチブロック

ここまでのアーキテクチャでは、設計者は各層で単一のカーネルサイズを選ばなければならなかった。1×1 は局所的にチャネルを混合し、3×3 は細かいテクスチャを捉え、5×5 はより広い近傍をカバーするが、実際の写真に写る物体はあらゆるスケールで現れる。GoogLeNet はこの選択そのものを回避する。その **Inception ブロック**は、同じ入力を複数のカーネルサイズで並列に処理し、それらの出力をチャネル軸に沿って連結するのである。複数のフィルタサイズをそのまま同時に走らせればパラメータも計算量も爆発してしまうため、GoogLeNet は高コストな各空間フィルタの手前に、次元削減（dimensionality reduction）のボトルネックとして NiN の 1×1 畳み込みを取り入れている。

7.2.3.1 Inception ブロック

Figure 7.6 に示すように、各 Inception ブロックは入力を 4 つの並列ブランチに分岐させる。(1) 1×1 畳み込み、(2) チャネル削減の 1×1 畳み込みとそれに続く 3×3 畳み込み、(3) チャネル削減の 1×1 畳み込みとそれに続く 5×5 畳み込み、(4) ストライド 1 の 3×3 最大プーリング層とそれに続く 1×1 畳み込みである。4 つの出力をチャネル次元に沿ってきれいに連結できるよう、すべてのブランチで空間的な高さと幅を保つように入力をパディングする。ブランチごとのチャネル数は、空間スケール間で容量を配分する調整つまみの役割を果たす（このブロック名は、映画 **Inception** と深層学習のミーム「もっと深くする必要がある（we need to go deeper）」に掛けた遊び心のある命名である）。

```
class Inception(nnx.Module):
    # c1--c4 are the numbers of output channels of the four branches
```

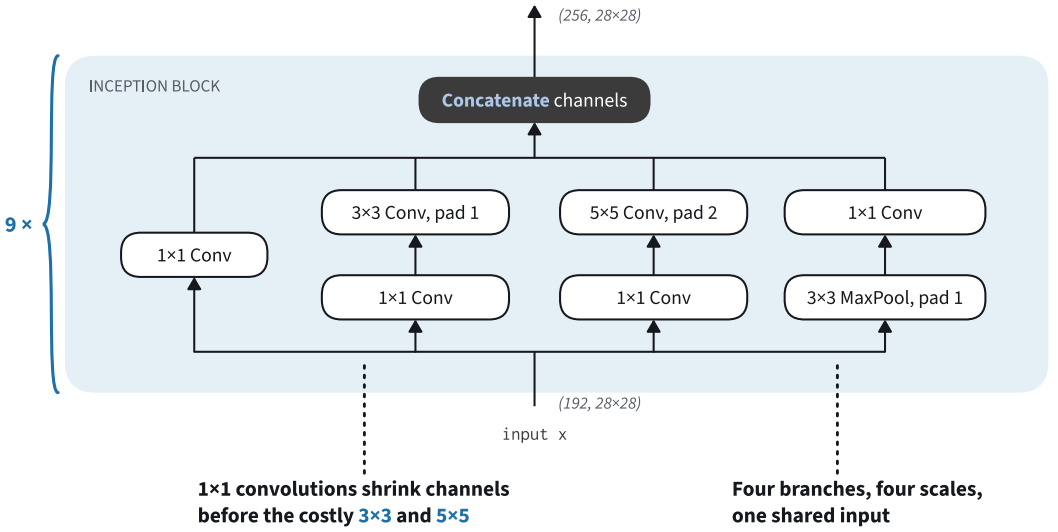


Figure 7.6: Inception ブロック（入力は下側）。4 つの並列ブランチが同じ入力を異なるカーネルサイズで読み込み、1×1 畳み込みが 3×3 および 5×5 畳み込みの手前でチャネル数を削減し、各ブランチの出力がチャネル次元に沿って連結される。図中の形状は、224×224 入力に対する GoogLeNet の最初の Inception ブロックのものである。

```
    def __init__(self, in_channels, c1, c2, c3, c4, init):
        self.b1_1 = nnx.Conv(in_channels, c1, kernel_size=(1, 1), **init)
        self.b2_1 = nnx.Conv(in_channels, c2[0], kernel_size=(1, 1), **init)
        self.b2_2 = nnx.Conv(c2[0], c2[1], kernel_size=(3, 3),
                             padding='SAME', **init)
        self.b3_1 = nnx.Conv(in_channels, c3[0], kernel_size=(1, 1), **init)
        self.b3_2 = nnx.Conv(c3[0], c3[1], kernel_size=(5, 5),
                             padding='SAME', **init)
        self.b4_2 = nnx.Conv(in_channels, c4, kernel_size=(1, 1), **init)

    def __call__(self, x):
        b1 = nnx.relu(self.b1_1(x))
        b2 = nnx.relu(self.b2_2(nnx.relu(self.b2_1(x))))
        b3 = nnx.relu(self.b3_2(nnx.relu(self.b3_1(x))))
        pooled = nnx.max_pool(x, window_shape=(3, 3), strides=(1, 1),
                              padding='SAME')
        b4 = nnx.relu(self.b4_2(pooled))
        return jnp.concatenate((b1, b2, b3, b4), axis=-1)
```

GoogLeNet の最初の Inception ブロックは、1×1 によるチャネル削減の効果を鮮やかに示している。224×224 の入力に対し、このブロックは解像度 28×28 で 192 チャネルを受け取る（Figure 7.6）。ブランチ幅を $c_1 = 64$、$c_2 = (96, 128)$、$c_3 = (16, 32)$、$c_4 = 32$ とすると、$64 + 128 + 32 + 32 = 256$ チャネルを同じ解像度 28×28 で出力する。もし 192 チャネルから 32 チャネルへ 5×5 畳み込みを直接適用すれば、$25 \cdot 192 \cdot 32 = 153{,}600$ 個の重みが必要になる。しかし、先に 192 チャネルを 16 チャネルへ射影しておけば、重みは $192 \cdot 16 = 3{,}072$ 個（1×1 による削減分）と $25 \cdot 16 \cdot 32 = 12{,}800$ 個（5×5 フィルタ分）の合計 15,872 個で済み、パラメータ数と積和演算の両方で約 10 分の 1 に削減される。高コストな空間畳み込みの手前に幅

の狭い 1×1 の射影（projection）を置くこの構造は**ボトルネック**と呼ばれ、深層視覚モデルで最も広く採用される基本要素の 1 つとなり、7.4.5.1 節や 7.5 節でも再び登場する。

7.2.3.2 ステム、ボディ、ヘッド

マクロな視点で見ると、GoogLeNet は現代のほぼすべての視覚バックボーンに共通する 3 部構成の**ステム・ボディ・ヘッド**という設計図を確立した。**ステム**は標準的な畳み込みとプーリング層の短い積み重ね（ここでは 3 つの畳み込みと 2 つの最大プーリング）を通じて生のピクセルを取り込み、空間解像度を素早く 8 分の 1 に縮小しつつ 192 チャネルへと拡張する。**ボディ**は 2 個、5 個、2 個の 3 ステージに配置された計 9 個の Inception ブロックを連結し、ステージ間をストライド 2 の最大プーリングで区切る。最後に**ヘッド**は NiN のコンパクトな設計を採用し、グローバル平均プーリングの後に単一の線形分類層を置く。

9 個の Inception ブロックにおける各ブランチの幅は、単純な数式から導かれたものではなく、元論文で手作業により調整されたものである。VGG と同様に、この構成全体を構造化データ、すなわち各 Inception ブロックにつき 1 つの設定タプルを収めたステージのタプルとして記述する。

```
googlenet_arch = (
    ((64, (96, 128), (16, 32), 32), (128, (128, 192), (32, 96), 64)),
    ((192, (96, 208), (16, 48), 64), (160, (112, 224), (24, 64), 64),
     (128, (128, 256), (24, 64), 64), (112, (144, 288), (32, 64), 64),
     (256, (160, 320), (32, 128), 128)),
    ((256, (160, 320), (32, 128), 128), (384, (192, 384), (48, 128), 128)))
```

`arch` を `GoogLeNet` に渡すことで、ステム、最大プーリングで区切られた 3 つの Inception ステージ、およびグローバルプーリングヘッドが組み立てられる。

```
class GoogLeNet(d2l.Classifier):
    def __init__(self, arch=googlenet_arch, lr=0.1, num_steps=4690,
                 num_classes=10, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        init = dict(kernel_init=he, rngs=rngs)

        def pool(x):  # 3x3 windows, stride 2; 'SAME' would pad unevenly
            return nnx.max_pool(x, window_shape=(3, 3), strides=(2, 2),
                                padding=((1, 1), (1, 1)))

        stem = nnx.Sequential(
            nnx.Conv(1, 64, kernel_size=(7, 7), strides=(2, 2), padding=3,
                     **init), nnx.relu, pool,
            nnx.Conv(64, 64, kernel_size=(1, 1), **init), nnx.relu,
            nnx.Conv(64, 192, kernel_size=(3, 3), padding='SAME', **init),
            nnx.relu, pool)
        body, in_channels = [], 192
        for group in arch:
            blocks = []
            for c1, c2, c3, c4 in group:
                blocks.append(Inception(in_channels, c1, c2, c3, c4, init))
                in_channels = c1 + c2[1] + c3[1] + c4
```

```
            body.append(nnx.Sequential(*blocks))
        head = nnx.Sequential(global_avg_pool,
                              nnx.Linear(in_channels, num_classes, **init))
        self.net = nnx.Sequential(stem, body[0], pool, body[1], pool,
                                  body[2], head)
```

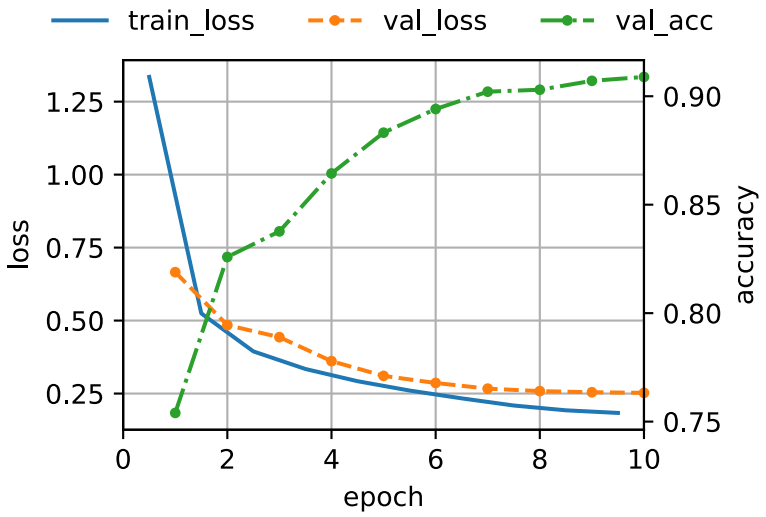
ステム内の 3 回（最初の 7×7 畳み込みと 2 回の最大プーリング）と Inception ステージ間の 2 回を合わせた計 5 回のストライド 2 のダウンサンプリング演算により、GoogLeNet は空間解像度を 32 分の 1 に縮小する。Fashion-MNIST の訓練を軽快に保つため、入力を 96×96 にリサイズすると、3 つの Inception ステージはそれぞれ 12×12、6×6、3×3 で動作し、チャネル数はステム直後の 192 から 480、832、1024 へと増加する。

```
GoogLeNet().layer_summary((1, 96, 96, 1))
```

```
Sequential            (1, 12, 12, 192)
Sequential            (1, 12, 12, 480)
pool                  (1, 6, 6, 480)
Sequential            (1, 6, 6, 832)
pool                  (1, 3, 3, 832)
Sequential            (1, 3, 3, 1024)
Sequential            (1, 10)
```

GoogLeNet は本節で群を抜いて深いアーキテクチャであり、最長経路に沿って 22 個のパラメータ付き層が連なる（ステムに 3 層、9 個の Inception ブロックに各 2 層、そして線形ヘッド）。NiN と同様に、0.1 ではときおりランダム推測の水準で停滞するため、ピーク学習率 0.03 で訓練する。以下の比較にあたり、実験上の細部を 1 点念頭に置いてほしい。入力を 96×96 にして訓練するため（224×224 ではなく）、GoogLeNet の 600 万個のパラメータが画像 1 枚あたりに要する積和演算はわずか 2 億 7600 万回にとどまる。

```
GoogLeNet.configure_optimizers = VGG.configure_optimizers
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
model = GoogLeNet(lr=0.03, num_steps=10 * len(data.train_dataloader()))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
val_acc['GoogLeNet'] = trainer.history['val_acc']
```



10 エポック後、GoogLeNet は検証正解率 91% 付近に達して同じピーク学習率の AlexNet に並び、訓練損失はわずかに低く（0.20 弱）、検証損失は同程度（約 0.25）となる。

7.2.4 ネットワークの比較

コサインスケジュールは学習率を滑らかに 0 まで減衰させるため、最終エポックの検証正解率は 4 つのモデルすべてで安定しており、直接比較できる。

```
{name: round(acc[-1], 3) for name, acc in val_acc.items()}  # After 10 epochs
```

```
{'VGG': 0.934, 'NiN': 0.888, 'PooledVGG': 0.894, 'GoogLeNet': 0.909}
```

Table 7.2 に、LeNet、AlexNet、および本節の 4 つのアーキテクチャをまとめ、入力解像度、パラメータ数、画像 1 枚あたりの積和演算数、ピーク学習率、Fashion-MNIST の検証正解率を一覧で示す。

Table 7.2: 各ネットワーク（6.6 節、7.1 節、および本節）のパラメータ数、積和演算数、および Fashion-MNIST 検証正解率。積和演算数は訓練時の入力サイズにおける画像 1 枚あたりの回数である。正解率は示されたピーク学習率で 10 エポック訓練した後の結果であり、複数回の実行にわたる範囲として示す（単一の値は、各実行の差が 0.5 ポイント以内に収まったことを意味する）。LeNet の結果は一定の学習率で 30 エポック訓練した後の値である。

ネットワーク	入力	パラメータ数	積和演算数	ピーク学習率	検証正解率
LeNet	28×28	0.06M	0.4M	2.0	88〜90%（30 エポック）
AlexNet	224×224	46.8M	938M	0.03	91%
VGG（1/4 幅）	224×224	43.1M	512M	0.1	93%
VGG（1/4 幅）	224×224	43.1M	512M	0.03	91〜92%
VGG（1/4 幅、プーリングヘッド）	224×224	0.6M	470M	0.03	89〜90%
NiN	224×224	3.4M	798M	0.03	89%
GoogLeNet	96×96	6.0M	276M	0.03	91%

ネットワーク	入力	パラメータ数	積和演算数	ピーク学習率	検証正解率

Table 7.2 からは、3 つの教訓が浮かび上がる。

1. **単純な演算量は正解率を決定しない：**1/4 幅の VGG は AlexNet の半分の積和演算で最高の正解率を達成する一方、NiN は AlexNet とほぼ同等の計算量を費やしながら 2 ポイント低い結果に終わる。
2. **ボトルネックとステムのダウンサンプリングがパラメータとピクセルの予算を大幅に削る：**GoogLeNet は $8\times$ のパラメータ削減（6.0M 対 46.8M）と 96×96 の入力（積和演算 276M 回。ただし 224×224 では約 1.5B 回の積和演算を実行する）により、AlexNet の 91% の正解率に並ぶ。
3. **グローバル平均プーリングは、幅の狭いトランクでのわずかな正解率のマージンと引き換えに膨大なパラメータを削減する：**`VGG` の密なヘッドをグローバル平均プーリングに置き換える（`PooledVGG`）と、Fashion-MNIST で約 2 パーセントポイントのコストを払う代わりにパラメータの 99%（43.1M $\rightarrow$ 0.6M）が削減される。一方 ImageNet では、より深いボディにプーリングヘッドを組み合わせた GoogLeNet が、144M パラメータの VGG モデルに圧勝した。

また、4 つのネットワークすべてに共通する限界にも注目してほしい。いずれも中間の活性化値を正規化しておらず、補助ヘッドや段階的な事前学習なしには約 22 層を超えられない。さらに、ピーク学習率を 0.03 から 0.1 に上げると、NiN、`PooledVGG`、GoogLeNet はいずれも不安定になる。7.3 節では、このスケールに対する脆弱性を直接解決する。

7.2.5 まとめ

AlexNet 以降、深層視覚モデルの設計は、個々の層を手作業で設計する手法からモジュール化されたブロックを組み合わせる手法へと移行した。VGG はパディング付き 3×3 畳み込みとプーリングの反復可能な積み重ねを導入し、小さなカーネルの積み重ねによってより少ない重みとより多くの非線形性で広い受容野を捉えられることを示すとともに、ネットワークトポロジをデータへと変えた。NiN はピクセルごとにチャネルを混合する 1×1 畳み込みを導入し、AlexNet や VGG で重みの 90% 超を占めていた巨大な密な分類ヘッドを、パラメータ不要のグローバル平均プーリングに置き換えた。GoogLeNet は、明快なステム・ボディ・ヘッド構成の内部で、マルチスケールの並列ブランチと 1×1 ボトルネックを融合させた。共通のコサインウォームアップスケジュールを用いた Fashion-MNIST 上での比較では、1/4 幅の VGG が最も高い正解率を示し、GoogLeNet は 8 分の 1 のパラメータ数で AlexNet に並び、VGG の密なヘッドをグローバル平均プーリングに置き換えると約 2 パーセントポイントの低下と引き換えにパラメータの 99% を削減できる。

7.2.6 演習問題

1. **設定から見る VGG-16**。VGG-16 は、224×224 の RGB 画像に対し、`arch=((2, 64), (2, 128), (3, 256), (3, 512), (3, 512))` の 5 つのブロックで 13 個の畳み込み層を積み重ね、その後に 4096、4096、1000 ユニットの 3 つの全結合層を続ける。
   1. 畳み込みボディと全結合ヘッドのそれぞれについてパラメータ数と積和演算数を手計算で求め、Simonyan and Zisserman (2015) が報告している 1 億 3800 万パラメータと合計が一致することを確認せよ。
   2. 単一の層として最も多くのパラメータを持つのはどの層か。最も多くの積和演算を実行するのはどの層か。また、複数の畳み込み層が同数で並ぶのはなぜか。
2. [code] **元の解像度**。Fashion-MNIST の画像は 28×28 であるため、224×224 へのアップサンプリングは情報を増やすことなく計算量を消費している。

1. 1/4 幅の VGG（arch=((1, 16), (1, 32), (2, 64), (2, 128), (2, 128))、lr=0.1）を、56×56 および 84×84 にリサイズした画像で訓練せよ。各解像度において密なヘッドに入力される特徴マップの空間サイズを示し、検証正解率と 1 エポックあたりの実行時間を 224×224 での実行結果と比較せよ。
2. GoogLeNet はすべてのストライド 2 のステップで解像度が半分（切り上げ）になるようパディングしているため、変更なしで 28×28 の入力を受け付ける。28×28 入力に対して、3 つの Inception ステージがそれぞれどの空間解像度で動作するか計算せよ。最終ステージにおいて、3×3 ブランチと 5×5 ブランチは何を計算しているか。

3. **削減効果の源泉**。●本節の層ごとの集計と Table 7.1 を用いて、以下の問いに答えよ。
   1. 10 クラスの単一チャネル 224×224 画像に対する VGG-11、NiN、GoogLeNet について、パラメータ数と積和演算数を畳み込み層と分類ヘッドにそれぞれ切り分けよ。
   2. VGG-11 と比べて、NiN がより多くのパラメータを削減しているのはボディか、それともヘッドか。GoogLeNet についてはどうか。積和演算数についても同じ問いに答え、それぞれの削減をもたらしている設計上の選択を挙げよ。
4. [code] **単一ブランチ Inception のアブレーション**。●Inception ブロックにおいて 3×3 ブランチ（1×1 による削減を含む）のみを残した GoogLeNet の変種を構築せよ。その際、3×3 畳み込みには元の 4 ブランチブロック全体と同じチャネル数を出力させ、各ブロックのパラメータ数が元の 10% 以内に収まるように 1×1 削減の幅を選べ。両方のネットワークを 96×96 の Fashion-MNIST 上で訓練し、検証正解率を比較せよ。この結果から、1 つのブロック内で複数のカーネルサイズを混合することについて何が言えるか。

7.3 正規化層

6.6 節で LeNet を訓練した際、MLP では問題なく機能した学習率が通用しなくなった。0.1 では検証正解率が数エポックにわたりランダム推測レベルに留まり、10 エポック後でも約 73% にしか達しなかったため、学習率を 20 倍の 2.0 に引き上げ、30 エポック訓練してようやく約 89% に到達させる必要があった。同様の脆さは 7.2 節でも見られた。そこではピーク学習率 0.1 で VGG は順調に訓練できた一方、NiN と GoogLeNet は停滞または発散してしまった。5 層のネットワークがステップサイズに対してこれほど敏感であるはずがない。なぜ実用的な学習率がアーキテクチャにこれほど微妙に依存するのか、そしてネットワーク内部に層を挿入してその脆さを取り除くことはできるのだろうか。

根本的な原因は、信号と勾配のスケールが深さ方向に伝播するにつれてドリフトすることにある。LeNet の 4 つのシグモイド層はそれぞれ、逆伝播される勾配を元の $1/4$ 以下に縮小するため、浅い層の更新量は深い層に比べて極端に小さくなり、単一の学習率ではネットワーク全体に適合しなくなる。各層が次の層へ入る活性化値のスケールを変化させるため、深さを増すほどこの問題は深刻になる。ReLU 活性化関数と慎重な初期化（4.4.2.2 節）によってステップ 0 における活性化値の分散は均衡させられるものの、勾配更新によって重みが動き始めると、それらのスケールを揃え続ける仕組みは何もない。

バッチ正規化（batch normalization）は、訓練全体を通じてこの問題を動的に解決する (Ioffe and Szegedy 2015)。層の非線形性の直前に配置されたバッチ正規化は、現在のミニバッチの平均と分散を使って各特徴量を標準化し、さらに学習されたスケールとシフトを適用する。この単純な変換によって順伝播の活性化値と逆伝播の勾配の両方が安定し、正規化された各層は重みのリスケーリングに対して不変になる。以下で確認するように、バッチ正規化を加えることで、LeNet は MLP と同じ学習率だけでなく、300 倍もの幅にわたるステップサイズで順調に訓練できるようになる。次節で扱う残差接続と並び、バッチ正規化は 100 層を超える深さのネットワークを実現可能にした。ただし、その高い経験的性能を支える正確なメカニズムについては、現在も研究者の間で議論が続いている。一方で、バッチ正規化はミニバッチ内の複数の例を結合するため、ミニバッチが非常に小さい場合には性能が低下する。そこで本節の後半では、各例を単独で正規化

する**層正規化**（layer normalization）と**グループ正規化**（group normalization）を導入する。

7.3.1 バッチ正規化

7.3.1.1 ミニバッチにわたる標準化

ミニバッチ $\mathcal{B}$（例数 m）にわたって全結合層が計算する活性化前の単一の特徴量を考え、得られるスカラー値を $x_1, \ldots, x_m$ とする。バッチ正規化は、このミニバッチにわたる標本平均と標本分散を次式のように計算する。

$$\hat{\mu}_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i,$$
$$\hat{\sigma}^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} (x_i - \hat{\mu}_{\mathcal{B}})^2,$$

そして、各値を次の式で変換する。

$$\mathrm{BN}(x_i) = \gamma \frac{x_i - \hat{\mu}_{\mathcal{B}}}{\sqrt{\hat{\sigma}^2_{\mathcal{B}} + \epsilon}} + \beta. \tag{7.3}$$

この分数項は、ミニバッチにわたって特徴量を平均 0、分散 $\hat{\sigma}^2_{\mathcal{B}}/(\hat{\sigma}^2_{\mathcal{B}} + \epsilon)$（1 よりわずかに小さい値）へと標準化する。微小な定数 $\epsilon > 0$（通常は 10^{-5}）は、ミニバッチ内のすべての値が一致した際のゼロ除算を防ぐためのものである。複数の特徴量を持つ層では、各特徴量がそれぞれのミニバッチ平均と分散を用いて独立に正規化される。

具体的な確認として、4 つの値 $(2, 4, 6, 8)$ は平均が 5、分散が $(9 + 1 + 1 + 9)/4 = 5$ であり、（ϵ を無視すれば）$(x_i - 5)/\sqrt{5} \approx (-1.34, -0.45, 0.45, 1.34)$ に標準化される。ここで、$(20, 40, 60, 80)$ や $(102, 104, 106, 108)$ もまったく同じ 4 つの数値に写されることに注目しよう。標準化は、入力される特徴量の任意のスケールとオフセットの両方を取り除くのである。

しかし、平均 0・分散 1 という分布が、後続のあらゆる活性化関数にとって理想的とは限らない。たとえばシグモイドは、分散 1 の入力領域ではほぼ線形に振る舞ってしまう。表現の自由度を完全に回復するため、Equation 7.3 では特徴量ごとに学習可能な**スケール** γ と**シフト** β を導入し、$\gamma = 1$、$\beta = 0$ に初期化してネットワークの重みとともに訓練する。γ と β をデータセットの真の標準偏差と平均にそれぞれ設定すれば（ミニバッチのサンプリングによる変動を除いて）正規化前の元の特徴量が復元されるため、バッチ正規化がネットワークの表現力を損なうことはない。

7.3.1.2 スケール不変性

特徴量のスケールを取り除くことは、最適化に対して 2 つの強力な効果をもたらす。1 つ目は直接的な効果であり、バッチ正規化がなぜ LeNet を救うのかを端的に説明する。ミニバッチ内で特徴量がほとんど変化しない場合（たとえば飽和しかけたシグモイドを通過して $1/2$ 付近に集中している場合）、その微小な標準偏差 $\hat{\sigma}_{\mathcal{B}}$ で割ることで順伝播において分散 1 が回復され、連鎖律によって逆伝播される勾配も同じ係数 $1/\hat{\sigma}_{\mathcal{B}}$ 倍に増幅される。信号の消失と勾配消失の双方が自動的にリスケーリングされるのである。

2 つ目の効果はより巧妙で、層の重み自体に作用する。$x_i = \mathbf{w}^\top \mathbf{h}_i + b$ を例 i の活性化前の値とする。$\mathbf{w}$ と b を任意の正の定数 $a > 0$ 倍にスケーリングすると、すべての x_i、その平均、および標準偏差が a 倍になり、（ϵ を無視すれば）正規化された出力は変化しない。

$$\mathrm{BN}(a\mathbf{w}^\top \mathbf{h}_i + ab) = \mathrm{BN}(\mathbf{w}^\top \mathbf{h}_i + b).$$

その結果、損失は $\mathbf{w}$ の**向き**のみに依存し、ユークリッドノルムには依存しない。両辺を $\mathbf{w}$ に関して微分すると、$a\mathbf{w}$ における勾配は、係数 $1/a$ を $\mathbf{w}$ における勾配に掛けたものに等しいことがわかる (Ioffe and Szegedy 2015)。すなわち、小さい重みは大きな勾配を受け取り、大きい重みは小さな勾配を受け取る。

過度に大きな学習率 η を選んだ場合に何が起きるか考えてみよう。損失は $\mathbf{w}$ の動径方向の半直線上で一定であるため、勾配 $\mathbf{g}$ は $\mathbf{w}$ と直交し、SGD の各ステップは重みのノルムを厳密に増加させる。すなわち、$\|\mathbf{w} - \eta\mathbf{g}\|^2 = \|\mathbf{w}\|^2 + \eta^2\|\mathbf{g}\|^2$ が成り立つ。過大なステップは $\|\mathbf{w}\|$ を急速に膨張させ、それによって後続の勾配はすべて同じ比率で縮小する。こうして、大きな学習率に対する自己安定化のフィードバックループが自然に形成されるのである (Arora et al. 2019)。

ただし、この解析の前提条件には留意しておく必要がある。厳密には、この議論は重み減衰（weight decay）なしの単純な SGD を仮定しており（重み減衰の下では、外向きの勾配による増大と内向きの減衰が釣り合う平衡状態に重みノルムが落ち着く）、直後に正規化が続く層のみに適用でき（分類器の最終出力層には当てはまらない）、さらに重みノルムが成長する前のごく初期の数ステップにおける発散を防ぐことはできない。こうした留保はあるものの、スケール不変性（scale invariance）は、正規化されたネットワークがはるかに広い学習率の範囲を許容できる理由を最も明快に説明してくれる。

7.3.1.3 全結合層と畳み込み層

重み $\mathbf{W}$、バイアス $\mathbf{b}$、非線形性 ϕ を持つ全結合層では、Ioffe and Szegedy (2015) で提案されたように、アフィン変換と活性化関数の間にバッチ正規化を配置する。

$$\mathbf{h} = \phi(\mathrm{BN}(\mathbf{W}\mathbf{x} + \mathbf{b})). \tag{7.4}$$

ここでは、各出力ユニットを独立した特徴量として扱い、それぞれ固有のミニバッチ平均、分散、γ_j、β_j を持たせる。

畳み込み層では、バッチ正規化は平行移動同変性（translation equivariance）を保つため、各出力**チャネル**を 1 つの特徴量として扱い、m 個のミニバッチ例**および** $p \times q$ 個のすべての空間位置の両方にわたって単一の平均と分散を計算する（チャネルあたり $m \cdot p \cdot q$ 個の値）。LeNet の最初の畳み込み層（チャネル数 6、空間サイズ 28×28、バッチサイズ 128）では、各チャネルの統計量は $128 \cdot 28 \cdot 28 = 100{,}352$ 個の要素を集約するため、全結合層のユニットあたり 128 個の要素よりもはるかに安定する。同様に、正規化された層が平行移動同変であり続けるよう、各チャネルはすべての空間座標にわたって単一のスカラー γ と β を共有する。

7.3.1.4 訓練と予測

推論時には、デプロイされたモデルが画像を 1 枚ずつ評価することも多く、ある入力に対する予測は、たまたま同じミニバッチに入った他の画像に依存するのではなく決定論的でなければならない。そのため、バッチ正規化は 2 つの異なるモードで動作する。**訓練モード**（training mode）では、Equation 7.3 のように現在のミニバッチの統計量を用いて正規化しつつ、各ステップの後に指数移動平均（exponential moving average）である $\hat{\mu}$ と $\hat{\sigma}^2$（それぞれ 0 と 1 に初期化）を次式のように更新する。

$$\begin{aligned} \hat{\mu} &\leftarrow \rho\,\hat{\mu} + (1-\rho)\,\hat{\mu}_{\mathcal{B}}, \\ \hat{\sigma}^2 &\leftarrow \rho\,\hat{\sigma}^2 + (1-\rho)\,\hat{\sigma}^2_{\mathcal{B}}. \end{aligned} \tag{7.5}$$

減衰（decay）を表す係数 ρ は、この移動統計量の実効的なウィンドウ幅を決める。k ステップ前のミニバッチは重み $(1-\rho)\rho^k$ を持つ。$\rho = 0.9$ のとき、重みは約 7 ステップごとに半減し、直近の約 $1/(1-\rho) = 10$ 個のミニバッチ（Fashion-MNIST の 1 エポックあたり 469 個あるサイズ 128 のミニバッチのうち）にわたって平均することになる。**予測モード**（prediction mode）では、層はこれらの移動推定値を固定し、それを用いてすべての入力を正規化する。これにより、バッチ正規化は固定された決定論的なアフィン変換に帰着する。このように、ドロップアウト（4.6 節）と同様、バッチ正規化は訓練時と評価時で異なる振る舞いをする。

ライブラリ間における悪名高い API の不統一には注意が必要である。パラメータ ρ を `momentum` と呼ぶフレームワークがある一方、$1-\rho$ を `momentum` と呼ぶフレームワークもあり、しかもそのどちらも最適化のモメンタムとは何の関係もない。本章全体を通して $\rho = 0.9$ および $\epsilon = 10^{-5}$ に統一し、フレームワークのデフォルト値が異なる箇所ではこれらを明示的に渡すことにする。

7.3.2 ゼロからの実装

以下の `batch_norm` は、全結合層（2D）と畳み込み層（4D）の両方の入力に対し、訓練モードと予測モードにわたって Equation 7.3 と Equation 7.5 の双方を実装したものである。

画像は（例、高さ、幅、チャネル）の形式で格納されるため、チャネル軸は末尾の軸である。

```
def batch_norm(X, gamma, beta, moving_mean, moving_var, eps, momentum,
               training):
    if not training:
        # Prediction mode: normalize with the running estimates
        X_hat = (X - moving_mean[...]) / jnp.sqrt(moving_var[...] + eps)
    else:
        assert X.ndim in (2, 4)
        # Channels last: average over every axis except the last one
        axes = tuple(range(X.ndim - 1))
        mean = X.mean(axis=axes, keepdims=True)
        var = ((X - mean) ** 2).mean(axis=axes, keepdims=True)
        X_hat = (X - mean) / jnp.sqrt(var + eps)
        # Update the running estimates in place; `momentum` is the decay rho
        moving_mean[...] = momentum * moving_mean[...] + (1.0 - momentum) * mean
        moving_var[...] = momentum * moving_var[...] + (1.0 - momentum) * var
    return gamma * X_hat + beta  # Scale and shift
```

`batch_norm` を `BatchNorm` 層にラップするには、2 種類の異なる状態を管理する必要がある。1 つはオプティマイザによって更新される学習可能なパラメータ（`gamma` と `beta`）であり、もう 1 つはオプティマイザからは無視されるものの、重みとともにモジュールがディスクへ保存し GPU へ転送する訓練対象外のバッファ（`moving_mean` と `moving_var`）である。

```
class BatchNorm(nnx.Module):
    # `num_features`: outputs of a fully connected layer or channels of a
    # convolutional layer. `num_dims`: 2 or 4, the number of input axes
    def __init__(self, num_features, num_dims, deterministic=False):
        self.deterministic = deterministic  # True in prediction mode
        shape = (1, num_features) if num_dims == 2 else (1, 1, 1, num_features)
        # Scale and shift, learned, start at 1 and 0
        self.gamma = nnx.Param(jnp.ones(shape))
        self.beta = nnx.Param(jnp.zeros(shape))
        # Running estimates, not learned, start at 0 and 1
```

```
        self.moving_mean = nnx.BatchStat(jnp.zeros(shape))
        self.moving_var = nnx.BatchStat(jnp.ones(shape))

    def set_view(self, *, deterministic):
        self.deterministic = deterministic

    def __call__(self, X):
        return batch_norm(X, self.gamma, self.beta, self.moving_mean,
                          self.moving_var, eps=1e-5, momentum=0.9,
                          training=not self.deterministic)
```

NNX は移動推定値を BatchStat 変数に保持する。オプティマイザは Param のみを更新するため、これらは無視される。Trainer は deterministic=False でモデルの訓練ビューを、deterministic=True で予測ビューを作成し、set_view がこのフラグを受け取る。

ゼロから構築した実装を検証するため、BatchNorm とフレームワークの組み込み層（いずれも $\rho = 0.9$、$\epsilon = 10^{-5}$）の両方を、平均 2、標準偏差 3 で生成した 128 個の 6 チャネル 28×28 特徴マップからなる同一のミニバッチに対して実行する。以下では、すべての出力チャネルが平均 0、分散 1 になること、出力がフレームワークの組み込み層と一致すること、そして 1 ステップ後に移動平均が適切に更新されることを確認する。

```
X = 2 + 3 * jax.random.normal(d2l.get_key(), (128, 28, 28, 6))
bn = BatchNorm(6, num_dims=4)  # Training mode
bn_ref = nnx.BatchNorm(6, momentum=0.9, epsilon=1e-5, use_fast_variance=False,
                       use_running_average=False,
                       rngs=nnx.Rngs(d2l.get_key()))
Y, Y_ref = bn(X), bn_ref(X)
{'largest channel mean': float(jnp.abs(Y.mean(axis=(0, 1, 2))).max()),
 'average channel variance': float(Y.var(axis=(0, 1, 2)).mean()),
 'largest difference': float(jnp.abs(Y - Y_ref).max()),
 'running means': bn.moving_mean[...].reshape(-1)}
```

```
{'largest channel mean': 9.396854494525542e-08,
 'average channel variance': 0.9999990463256836,
 'largest difference': 4.76837158203125e-07,
 'running means': Array([0.20013797, 0.20052281, 0.1993001 , 0.20045717, 0.1992819 ,
        0.19969386], dtype=float32)}
```

単精度の丸め誤差の範囲内ですべてのチャネルが平均 0、分散 1 となり、実装した層の出力は組み込み層と 10^{-6} 以内で一致している。1 回のミニバッチ処理の後、移動平均は初期値の 0 からミニバッチ平均の 2 に向けて $1 - \rho = 0.1$ の割合だけ進んでいる。

7.3.3 バッチ正規化を用いた LeNet

ここで LeNet（6.6 節）に戻り、バッチ正規化の効果を試してみよう。まったく同じ Xavier 初期化を保ったまま、BNLeNet では LeNet の 4 つの隠れ層（6 チャネルと 16 チャネルの畳み込み、および 120 ユニットと 84 ユニットの全結合層）のそれぞれの直後、各シグモイド活性化関数の直前に組み込みのバッチ正規化層を挿入する。これら 4 つの層ではバイアスを無効化している点に注目してほしい。ミニバッチ平均を引く操作によってバッチ正規化の直前にある加法的なバイアスは完全に相殺され、β がその役割を引き継ぐからである。正規化される全 $6 + 16 + 120 + 84 = 226$ 個の特徴量にわたり、226 個のバイアスを 226 組の (γ, β) ペアに置き換えることで、LeNet の 61,706 個のパラメータに加わる訓練可能なパラメータはわずか 226 個にとどまる。

```
class BNLeNet(d2l.Classifier):
    """LeNet with batch normalization before each hidden sigmoid."""
    def __init__(self, lr=0.1, num_steps=4690, num_classes=10, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        init = dict(kernel_init=nnx.initializers.xavier_uniform(), rngs=rngs)
        bn = lambda n: nnx.BatchNorm(n, momentum=0.9,  # rho = 0.9
                                     use_fast_variance=False, rngs=rngs)
        nb = dict(use_bias=False, **init)  # No biases before BN

        def avg_pool(x):  # 2x2 windows, stride 2
            return nnx.avg_pool(x, window_shape=(2, 2), strides=(2, 2))

        def flatten(x):  # One vector per example
            return x.reshape((x.shape[0], -1))

        self.net = nnx.Sequential(
            nnx.Conv(1, 6, kernel_size=(5, 5), padding=2, **nb), bn(6),
            nnx.sigmoid, avg_pool,
            nnx.Conv(6, 16, kernel_size=(5, 5), padding='VALID', **nb),
            bn(16), nnx.sigmoid, avg_pool, flatten,
            nnx.Linear(400, 120, **nb), bn(120), nnx.sigmoid,
            nnx.Linear(120, 84, **nb), bn(84), nnx.sigmoid,
            nnx.Linear(84, num_classes, **init))
```

Flax の `nnx.BatchNorm` はデフォルトで `momentum=0.99` となっているため $\rho = 0.9$ を渡すが、`epsilon` のデフォルトは 10^{-5} である。また、デフォルトでは二乗の平均から平均の二乗を引く形で分散を計算するため、標準偏差に比べて平均が大きい場合に数値精度が低下する。`use_fast_variance=False` を指定すると、前節の実装と同様に二乗する前に平均を引くようになる。Flax はデフォルトで `'SAME'` パディングを行うため、2 つ目の畳み込みでは `'VALID'` を指定しており、平坦化された出力は $16 \cdot 5 \cdot 5 = 400$ 個の要素を持つ。

訓練に入る前に、予測モードと学習率の間の微妙な相互作用に注目しておこう。移動平均は約 10 ミニバッチ分の遅れを伴うため、ネットワークの重みがまだ急速に変化している間は、少し古い時点のネットワークのスナップショットを表すことになる。これは単なる理論上の話ではない。一定の学習率 0.1 の下では、`BNLeNet` の予測モードにおける検証正解率が、最終エポックを含む個々のエポックで突然 10 パーセントポイント以上も急落することがある。学習率を滑らかに 0 へと減衰させれば、この問題はきれいに解消する。ステップサイズが縮むにつれて重みが落ち着き、移動平均が最終的なネットワークに収束するからである。そこで、7.1 節で用いたウォームアップ付きコサインスケジュール（詳しくは第 9 章で学ぶ）を再利用し、`num_steps` の最初の 10 分の 1 でウォームアップを行うとともに、両モデルがまったく同じスケジュールを共有できるよう `d2l.LeNet` をサブクラス `CosineLeNet` でラップする。

```
@d2l.add_to_class(BNLeNet)
def configure_optimizers(self):
    return optax.sgd(lambda t: self.lr * d2l.cosine_warmup(
        t, self.num_steps, self.num_steps // 10))

class CosineLeNet(d2l.LeNet):
    def __init__(self, lr=0.1, num_steps=4690, num_classes=10, rngs=None):
```

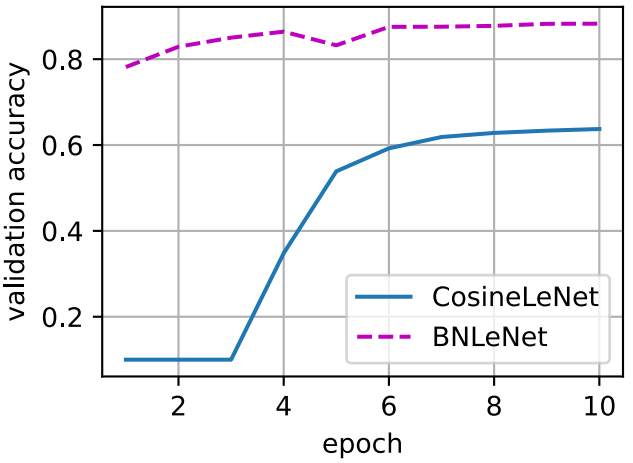
```
        super().__init__(lr, num_classes, rngs=rngs)
        self.num_steps = num_steps

    configure_optimizers = BNLeNet.configure_optimizers
```

CosineLeNet と BNLeNet の両方を同一の設定（ピーク学習率 0.1、バッチサイズ 128、10 エポック）で訓練し、エポックごとの検証正解率を追跡してみよう。

```
data = d2l.FashionMNIST(batch_size=128)
num_steps = 10 * len(data.train_dataloader())
curves = {}
for model in (CosineLeNet(0.1, num_steps), BNLeNet(0.1, num_steps)):
    trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
    trainer.board.display = False  # We plot the accuracies ourselves
    trainer.fit(model, data)
    curves[type(model).__name__] = trainer.history['val_acc']
```

curves に記録されたエポックごとの検証正解率の推移をプロットすると、CosineLeNet と BNLeNet の収束速度を直接対比できる。

```
d2l.plot(list(range(1, 11)), list(curves.values()), 'epoch',
         'validation accuracy', legend=list(curves))
```



また、最終的な到達性能を比較するため、両モデルの 10 エポック後の検証正解率 acc[-1] を curves から取り出して表示する。

```
{name: round(acc[-1], 3) for name, acc in curves.items()}  # After 10 epochs
```

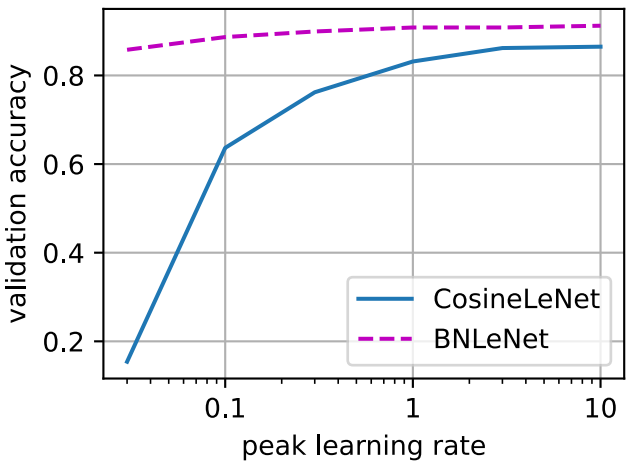
```
{'CosineLeNet': 0.637, 'BNLeNet': 0.883}
```

バッチ正規化なしの LeNet は、最初の 2〜5 エポックにわたりランダム推測レベル（10%）で停滞し、10 エポック後でも約 60% にしか達しない（6.6 節の 73% より低いのは、コサイン減衰によって平均ステップサイズが半分になるためと考えられる）。一方、バッチ正規化を加えると、ま

ったく同じアーキテクチャが最初の 1 エポック目で 70% を超え、10 エポック後には約 88% まで上昇する。これは、正規化なしの LeNet が 20 倍の学習率と 3 倍のエポック数を費やしてようやく到達した 88〜90% にほぼ匹敵する水準である。また、学習率が 0 へと減衰するにつれて、終盤のエポックで曲線が滑らかに平坦化していることにも注目してほしい。わずか 226 個のパラメータを追加したことでネットワークの表現容量が広がったのではなく、単にすべての層の活性化値と勾配が健全なスケールに保たれるようになったのである。

しかし、単一の学習率で比較しただけでは重要な疑問が残る。バッチ正規化は最適なステップサイズの位置を単にずらしただけなのか、それとも実用的な学習率の範囲を本当に広げたのだろうか。これを確かめるため、ピーク学習率を 0.03 から 10 までの 300 倍の幅にわたって変化させ、両モデルの 10 エポック後の最終検証正解率をプロットする。

```
lrs = (0.03, 0.1, 0.3, 1, 3, 10)
final = {'CosineLeNet': [], 'BNLeNet': []}
for lr in lrs:
    for cls in (CosineLeNet, BNLeNet):
        trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
        trainer.board.display = False
        trainer.fit(cls(lr, num_steps), data)
        final[cls.__name__].append(trainer.history['val_acc'][-1])
d2l.plot(list(lrs), list(final.values()), 'peak learning rate',
         'validation accuracy', legend=list(final), xscale='log')
```



このスイープ実験は決定的な答えを示している。通常の LeNet がうまく訓練できるのは 1〜3 付近の狭い帯域（84〜87% に到達）に限られる。この範囲より下では学習が遅すぎて 10 エポックでは収束せず、10 では挙動が不安定になってランダム推測レベルで停滞することもある。対照的に、BNLeNet は 0.03 から 10 までの**あらゆる**学習率で約 85% を超え、1 以上の学習率では約 91% に達する（0.1 のときより丸々 3 ポイント高い）。正規化なしのネットワークが 3 倍以内の範囲で学習率を調整しなければならないのに対し、正規化されたネットワークはスケール不変性の解析が予測したとおり、300 倍もの範囲にわたって安定して機能する（ただし、最終出力層は正規化されていないため、上限が際限なく伸びるわけではない）。

バッチ正規化が深いネットワークにこれほど劇的な効果をもたらす理由は、当初提唱された内部共変量シフト（internal covariate shift）の軽減にとどまらない (Ioffe and Szegedy 2015)。その後の研究により、損失ランドスケープと勾配を滑らかにすること (Santurkar et al. 2018)、深さ

方向にわたって活性化値と勾配のスケールを均衡させること、隠れ層の重みをスケール不変にして大きな学習率を自己安定化させること、そしてドロップアウトのように正則化として働く適度なミニバッチノイズを注入すること (Luo et al. 2018; Brock et al. 2021) が明らかになっている。

7.3.4 バッチ統計量を使わない正規化

ある例の出力を同じミニバッチ内の他の例と結合させるというのは、ニューラルネットワークの層としては異例の性質であり、実用上 2 つの弱点をもたらす。第 1 に、バッチサイズが小さくなるにつれてミニバッチ統計量の質が悪化する。極端な例として、1 例のみのミニバッチに対する全結合層では、すべての要素が自身の平均と一致するため、標準化された活性化値はすべて強制的に 0 となり、層は入力にかかわらず定数 β を出力してしまう。物体検出やセマンティックセグメンテーションのようにメモリ消費の大きいタスクでは、GPU あたり 1〜2 枚の高解像度画像しか載せられないことが多く、まさにバッチ統計量のノイズが最も大きくなる領域にぶつかる。第 2 に、バッチ正規化は訓練モードと予測モードで 2 つの異なる関数を計算する。たとえばドメインをまたぐファインチューニングの後など、移動推定値がテスト時の入力分布とずれてしまうと、気づかないうちに正解率が低下し、デバッグが極めて困難な不具合を引き起こすことがある。

正規化の統計量を**各例の内部で独立に**計算すれば、これら 2 つの制約はともに解消する。Figure 7.7 は、バッチ軸 N、チャネル軸 C、空間軸 $H \cdot W$ を持つ畳み込みの活性化値テンソル上で、こうして得られる正規化層の族を比較したものである。どの手法もまったく同じ標準化の式を用いており、どの要素をまとめて 1 つの平均と分散を共有させるかという点のみが異なる。

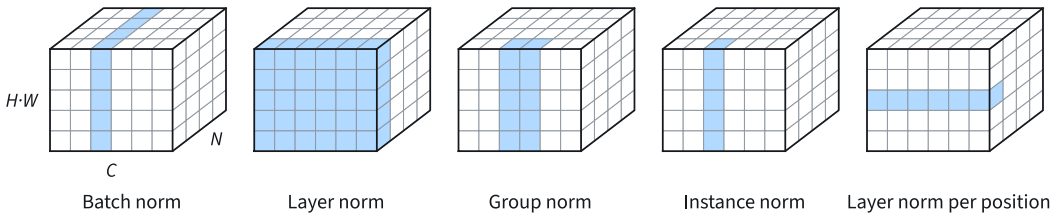


Figure 7.7: 例 N、チャネル C、空間位置 $H \cdot W$ を持つ活性化値に対する各種正規化手法。色付きの要素が 1 つの平均と 1 つの分散を共有する。バッチ正規化は各チャネルをミニバッチと全位置にわたって平均し、層正規化は各例を全チャネルと全位置にわたって平均し、グループ正規化はチャネルのグループと全位置にわたり、インスタンス正規化は単一チャネルにわたって平均する。Transformer や ConvNeXt における層正規化は、1 つの位置におけるチャネル方向にわたって平均する。

7.3.4.1 層正規化

層正規化 (Ba et al. 2016) は、単一の例が持つ全 n 個の特徴量 $\mathbf{x} = (x_1, \dots, x_n)$ を独立に標準化する。

$$\mathrm{LN}(\mathbf{x})_j = \gamma_j \frac{x_j - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}} + \beta_j,$$
$$\text{with} \quad \hat{\mu} = \frac{1}{n}\sum_{j=1}^{n} x_j \quad \text{and} \quad \hat{\sigma}^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \hat{\mu})^2.$$

これは Equation 7.3 を転置したものにあたる。1 つの特徴量を複数の例にわたって平均する代わりに、特徴量ごとの学習可能なスケール γ_j とシフト β_j は保ったまま、1 つの例をその特徴量

方向にわたって平均するのである。計算に他の例が一切関与しないため、層正規化はあらゆるバッチサイズにおいて、また訓練モードと予測モードの両方でまったく同じように動作し、移動統計量を一切必要としない。なお、実際に「層正規化」が集約する軸はアーキテクチャによって異なる。畳み込みテンソルでは 1 つの例の全チャネルと全空間位置にわたって集約する場合がある（Figure 7.7 の左から 2 番目のパネル）のに対し、Transformer は各トークンの特徴ベクトルを独立に正規化し、ConvNeXt（7.7 節）も同様に空間方向に集約せず各ピクセルのチャネルベクトルを個別に正規化する（右端のパネル）。

7.3.4.2 グループ正規化とインスタンス正規化

畳み込み層に特化して設計された**グループ正規化** (Wu and He 2018) は、c 個のチャネルを G 個のグループ（各グループ c/G チャネル）に分割し、各例をグループごとの $(c/G) \cdot p \cdot q$ 個の要素にわたって標準化した後、チャネルごとに学習されたスケールとシフトを適用する。これは 2 つの両極端を滑らかに補間する。$G = 1$ では全チャネルと全位置にわたって集約し（Figure 7.7 の空間的な層正規化）、$G = c$ では各例の各チャネルを単独で正規化する**インスタンス正規化**（instance normalization）になる。中間の値（通常は $G = 32$）を選べば、無関係な特徴量のチャネルに単一の平均と分散の共有を強いることなく、信頼できる統計量を得るのに十分なグループあたりの要素数を確保できる。以下では、32 個のテンソルからなるバッチ（16 チャネル、4 グループ）を用いて、すべての `(example, group)` スライスが平均 0、分散 1 に標準化されること、そして先頭の 1 例のみを単独で正規化した結果がバッチ全体の中で正規化した結果と完全に一致することを確認する。

```
X = jax.random.normal(d2l.get_key(), (32, 8, 8, 16))  # Channels last
gn = nnx.GroupNorm(16, num_groups=4, epsilon=1e-5,
                   rngs=nnx.Rngs(d2l.get_key()))
Y = gn(X)
# Collect each (example, group) pair's numbers
G = jnp.transpose(Y, (0, 3, 1, 2)).reshape(32, 4, -1)
(jnp.abs(G.mean(axis=2)).max(), G.var(axis=2).mean(),
 jnp.abs(gn(X[:1]) - Y[:1]).max())
```

```
(Array(2.6077032e-08, dtype=float32),
 Array(0.9999897, dtype=float32),
 Array(4.7683716e-07, dtype=float32))
```

予想どおり、丸め誤差の範囲内ですべてのグループが平均 0、分散 1 となり、先頭の例の出力はバッチの残りの例から厳密に独立している。この独立性は正解率とのトレードオフを伴うのだろうか。バッチサイズが大きい場合、その差はごくわずかである。ImageNet 上の ResNet-50 を GPU あたり 32 枚の画像で訓練した実験では、Wu and He (2018) は top-1 誤り率がバッチ正規化の 23.6% に対してグループ正規化では 24.1% であると報告している。しかし、GPU あたり 2 枚の画像に減らすと、バッチ正規化の誤り率が 34.7% へと急増する一方、グループ正規化は 24.1% のまま安定している。このため、GPU あたりの小バッチで訓練される物体検出モデルやセグメンテーションモデルでは、グループ正規化が標準的な選択肢となっている。

7.3.4.3 ノーマライザフリーネットワーク

深い畳み込みネットワークにおいて信号のスケールを適切に保つ方法は、活性化値の正規化だけではない。**ノーマライザフリーネットワーク**（normalizer-free network、NFNet）は、正規化層を 1 つも使わずに深い ResNet 風のアーキテクチャを訓練し、（ImageNet のみで訓練されたモデルの中で）ImageNet の最先端の正解率を達成した (Brock et al. 2021)。NFNet は活性化値を正規化する代わりに、各畳み込みカーネルの重みを直接標準化することで、初期化（4.4.2.2 節）が

ステップ 0 でしか確立できない分散の均衡を訓練全体にわたって維持し、さらに各重みテンソルのノルムに応じた適応的勾配クリッピング（adaptive gradient clipping）を組み合わせる。本節で NFNet の実装は扱わないが、その成功は示唆に富む証拠を与えてくれる。活性化値ではなく重みを通じて信号スケールを制御しても深い畳み込みネットワークが同等にうまく訓練できるという事実は、正規化層の核心的な役割がスケール制御にあるという見方を強く裏付けている。

7.3.5 まとめ

バッチ正規化は、現在のミニバッチの平均と分散を用いて全結合層の各特徴量または畳み込み層の各チャネルを標準化し、続いて学習されたスケールとシフトを適用する。順伝播の信号と逆伝播の勾配を適切なスケールに保ち、正規化された層を重みノルムに対して不変にすることで、LeNet は 0.03 から 10 にわたるピーク学習率（正規化なしの 3 倍の帯域に対し 300 倍の帯域）で安定して訓練できるようになる。もっとも、深いネットワークにおけるその効果を完全に説明する理論は今なお未解決のままである。推論時には、バッチ正規化は訓練統計量の固定された指数移動平均へと切り替わる。例をまたぐ結合は極小のバッチサイズで破綻するため、層正規化とグループ正規化は各例の内部のみで厳密に統計量を計算し、バッチサイズへの感度と訓練・テスト間のモード分離の両方を解消する。次節では、バッチ正規化を残差接続と組み合わせ、ネットワークの深さを 100 層超へと押し広げる。

7.3.6 演習問題

1. **バッチ正規化直前のバイアス**。`BNLeNet` は正規化を伴う 4 つの層をバイアスなしで構築している。Equation 7.3 から、バッチ正規化の前にバイアス $\mathbf{b}$ を加えても訓練モードの出力も重みの勾配も変化せず、$\partial L/\partial \mathbf{b} = \mathbf{0}$ となることを示せ。また、予測モードではこのようなバイアスはどうなるか。

2. [code] **学習率の上限**。本節のスイープ実験では、`BNLeNet` は 10 までのすべてのピーク学習率で順調に訓練できた。

    1. `BNLeNet` のスイープをピーク学習率 30、100、300 まで拡張せよ。どこで訓練が破綻するか。
    2. 出力の線形層は、直後にバッチ正規化を持たない唯一の重み層である。出力層の学習率を 0.1 に固定したまま、正規化された各層のピーク学習率を 30、100、300 にわたってスイープし、この出力層が上限を決めているかどうか検証せよ。
    3. `CosineLeNet` と `BNLeNet` の両方でシグモイドを ReLU に置き換え（He 初期化を使用）、学習率のスイープを再度実行せよ。正規化なしのネットワークの実用的な帯域はどのように変化するか。また、それはなぜか。

3. [code] **移動平均と再計算された統計量**。@eq-bn-running の減衰 ρ は、移動統計量が直近いくつのミニバッチにわたって平均するかを決定する。

    1. $\rho = 0.9$ と $\rho = 0.99$ のそれぞれについて、移動平均におけるあるミニバッチの重みが最新のミニバッチの重みの 1% 未満になるのは何ステップ後か。両者の期間を Fashion-MNIST の 1 エポックにあたる 469 ステップと比較せよ。
    2. `BNLeNet` を一定の学習率 0.1（`d2l.Classifier.configure_optimizers`）で 10 エポック訓練せよ。各エポックの後にモデルを 2 通りの方法で評価せよ。1 つは移動平均を用いた予測モードでの評価、もう 1 つは重みを固定した状態で 100 個の訓練ミニバッチから統計量を新たに再計算した後の評価である。両方の検証曲線をコサインスケジュールの曲線と並べてプロットし、その差が生じる理由を説明せよ。

4. [code] **ミニバッチ内の結合**。バッチ正規化はミニバッチ内の複数の例を結合するため、訓練済みの `BNLeNet` を検証セット上で次の 3 通りの方法で評価せよ（訓練モードの実行で移

動統計量が上書きされないよう、予測モードを最初に計測すること）。すなわち、(i) 予測モード、(ii) 検証バッチサイズ 128 での訓練モード、(iii) 検証バッチサイズ 2（または 1）での訓練モードである。得られた 3 つの正解率について説明せよ。

5. [code] **小バッチ訓練：BatchNorm 対 GroupNorm・LayerNorm**。●BNLeNet の派生モデルとして、2 つの畳み込み層の後の BatchNorm をグループ正規化（2 グループ）に、2 つの隠れ全結合層の後の正規化を層正規化（7.3.4.1 節）に置き換えたものを構築せよ。BNLeNet とこの派生モデルの両方を、バッチサイズ 4 およびバッチサイズ 128 で 2 エポック訓練せよ。バッチサイズが 128 から 4 に下がると各モデルの検証正解率はどのように変化するか。また、それはなぜか。

6. [code] **バッチ正規化の逆伝播**。●Equation 7.3 の $\hat{\mu}_{\mathcal{B}}$ と $\hat{\sigma}^2_{\mathcal{B}}$ はミニバッチのすべての要素に依存するため、バッチ正規化の逆伝播は全 m 個の例を結合する。
   1. 1D バッチ正規化層における勾配 $\partial L/\partial x_i$（スカラー損失 L の各入力 x_i に関する偏微分）を手計算で導出し、得られた式から恒等的に $\sum_{i=1}^{m} \partial L/\partial x_i = 0$ となることを示せ。
   2. 導出した逆伝播の式を実装し、自動微分の結果とマシン精度で一致することを検証せよ。

7.4 残差ネットワーク

ImageNet においてはここまで、AlexNet の 8 層から GoogLeNet の 22 層に至るまで、ネットワークを深くすることは一貫して高い正解率を意味しており、バッチ正規化（7.3 節）によって深さをさらに拡大するうえでの残りの障壁も取り除かれたかのように見えた。当然ながら、さらに多くの層を積み重ねれば、悪くとも過学習が始まるまでは訓練誤差が下がり続けると予想されるだろう。ところが、He et al. (2016a) がバッチ正規化と ReLU を備えた VGG 風の**プレーン**な畳み込みの積層を深さを増しながら訓練してこの直感を検証したところ、まったく逆の結果が明らかになった。CIFAR-10 では、56 層のプレーンネットワークが 20 層のネットワークよりも大幅に高い**訓練**誤差で訓練を終えたのである。訓練セットへの当てはまりさえ悪化しているため、この**劣化問題**（degradation problem）は過学習ではなく、最適化の失敗である。この謎は深刻だ。原理的には、56 層のネットワークは 20 層の解をそのままコピーし、残りの 36 層にわたって恒等写像（identity mapping）として振る舞うだけで、少なくとも同じくらい低い訓練損失を保証できるはずである。なぜ SGD はそのような解を見つけられないのか、そして見つけられるようにするには基本ブロックをどう再設計すればよいのだろうか。

残差ネットワーク（residual network、ResNet）は、拍子抜けするほど単純な再パラメータ化（reparameterization）によってこの逆説を解決する。各ブロックに出力変換全体をゼロから学習させる代わりに、残差ブロック（residual block）は学習した補正項を入力に加算する。そのため、この補正項を 0 に近づけるだけで直ちに恒等写像が復元される。たった 1 本のこのショートカットが前例のない深さを可能にした。ImageNet では、34 層のプレーンネットワークが 18 層のモデルに遅れをとったのに対し、34 層の ResNet は 18 層の ResNet を 3 パーセントポイント近く上回り、152 層の ResNet が 2015 年の ImageNet コンペティションで優勝した。

本節ではまず、入れ子の関数クラスを通じて深さを捉え直し、残差ブロックを設計する。次に、Fashion-MNIST 上で 20 層と 56 層のプレーンネットワークおよび残差ネットワークを並べて訓練することで劣化現象を直接再現し、劣化が現れては消える様子を目の当たりにする。続いて ResNet-18 を組み立てて訓練し、最後に影響力の大きい 2 つの後継モデルを取り上げる。すなわち、グループ畳み込み（grouped convolution）を用いて同等のコストでブロックの幅を広げる **ResNeXt** と、特徴マップを加算するのではなく連結する **DenseNet** である。

7.4.1 関数クラスと深さ

あるアーキテクチャの**関数クラス**（function class）$\mathcal{F}$、すなわちパラメータを変化させたときに達成可能な入出力写像の集合を考えよう。より大きなアーキテクチャ $\mathcal{F}'$ が、$\mathcal{F}$ 内の最良の解と少なくとも同じくらい低い訓練損失をもつ解を必ず含むのはどのようなときか。それは、関数クラスが**入れ子**（nested）になっており、$\mathcal{F} \subseteq \mathcal{F}'$（Figure 7.8）を満たすときである。もし $\mathcal{F} \not\subseteq \mathcal{F}'$ であれば、クラスを大きくしても目標の関数 f^* からかえって遠ざかることさえある。

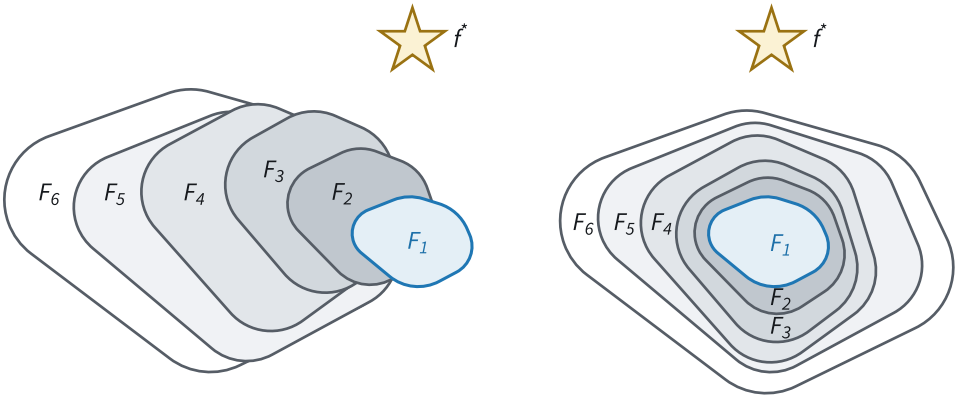


Figure 7.8: 次第に大きくなる関数クラス。左は入れ子になっていない場合であり、大きなクラスが小さなクラスの関数を含むとは限らず、目標 f^* に近づくとも限らない。右は各クラスが前のクラスをすべて含んでいるため、クラスが大きくなるにつれて利用可能な最良の関数は改善する一方である。

追加した層が恒等写像 $f(\mathbf{x}) = \mathbf{x}$ を表現できる限り、層を追加することで関数クラスは入れ子になる。プレーンブロック $\mathbf{x} \mapsto \mathrm{ReLU}(\mathrm{BN}(\mathbf{W}\mathbf{x} + \mathbf{b}))$ は非負の入力に対して恒等写像を表現**できる**ため、プレーンネットワークはすでに Figure 7.8 の右パネルに示した入れ子の状況にある。つまり、その劣化は表現能力の不足ではなく最適化の失敗であり、SGD が何十もの非線形層にわたって恒等写像を近似するのに苦戦しているのである。残差ブロックは層を再パラメータ化し、残差分岐の重みがゼロになる原点 $g(\mathbf{x}) = 0$ に恒等写像が位置するようにする。

7.4.2 残差ブロック

目標の写像 $f(\mathbf{x})$ を直接学習する代わりに、残差ブロック（Figure 7.9）は**残差**（residual）$g(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}$ を学習し、**ショートカット**（または**スキップ**）接続を通じて入力を足し戻すことで $\mathbf{x} + g(\mathbf{x})$ を出力する。このブロックを恒等写像にするには、g の最終層をゼロにするだけでよい。

ショートカットは勾配の伝わり方も変える。積み重ねた 2 つのブロック $\mathbf{x}_1 = \mathbf{x}_0 + g_1(\mathbf{x}_0)$ と $\mathbf{x}_2 = \mathbf{x}_1 + g_2(\mathbf{x}_1)$ に対して、ヤコビ行列 $\partial \mathbf{x}_2 / \partial \mathbf{x}_0 = (\mathbf{I} + \partial g_2 / \partial \mathbf{x}_1)(\mathbf{I} + \partial g_1 / \partial \mathbf{x}_0)$ を展開すると単位行列 $\mathbf{I}$ と分岐項の和になるため、損失の勾配は途中のすべての重み行列との積を経ることなく初期のブロックに到達する（連続時間の観点からは、$\mathbf{x}_{l+1} = \mathbf{x}_l + g_l(\mathbf{x}_l)$ は常微分方程式（ODE）$\dot{\mathbf{x}} = g(\mathbf{x}, t)$ の単位ステップのオイラー離散化にあたる。38.3 節を参照）。

プレーンネットワークが劣化するのは勾配が消失するためだと結論づけたくなるかもしれない。しかしそうではない。He et al. (2016a) は、バッチ正規化を施したプレーンネットワークがどの深さでも健全な勾配ノルムを維持していることを確認した。破綻の原因はより微妙であり、その後の研究によって 2 つの相補的な説明が与えられている。深いプレーンネットワークでは、深さが増すにつれて近傍の入力における勾配の相関が失われて白色雑音に近づき、SGD が従うべき

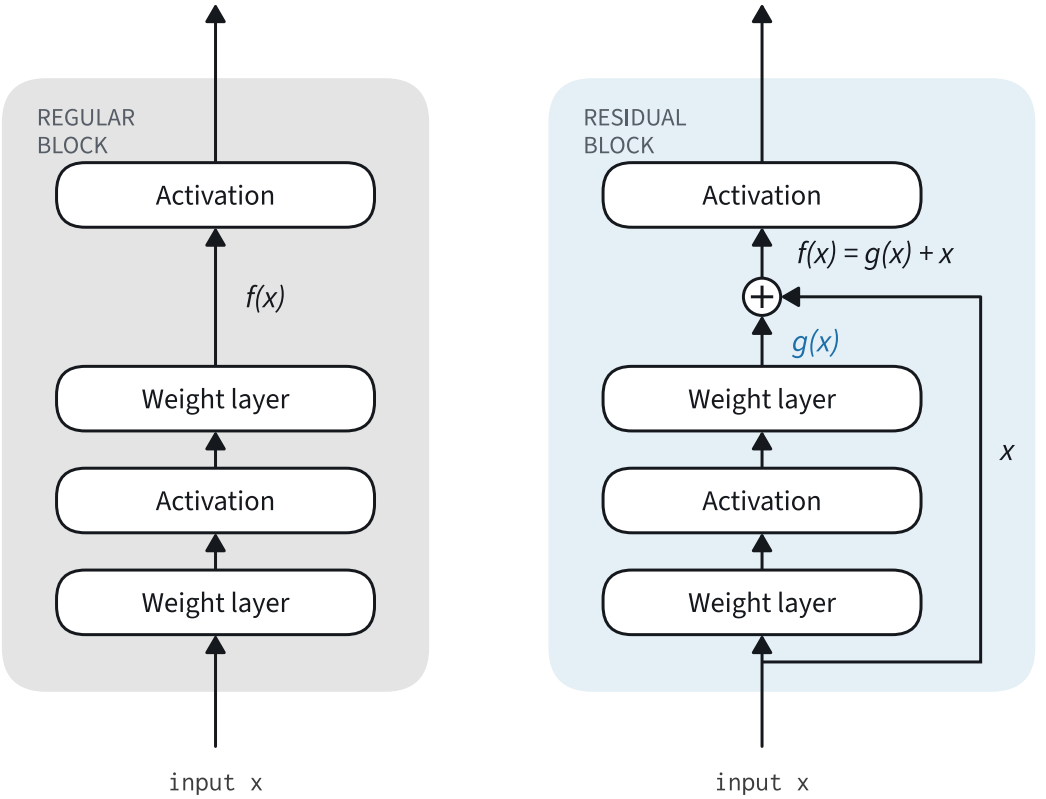


Figure 7.9: 通常のブロック（左）は写像 $f(\mathbf{x})$ を直接学習しなければならない。残差ブロック（右）は残差写像 $g(\mathbf{x}) = f(\mathbf{x}) - \mathbf{x}$ を学習して入力を足し戻すため、恒等写像 $f(\mathbf{x}) = \mathbf{x}$ は $g(\mathbf{x}) = 0$ に対応する。

一貫した方向を見失うのに対し、ショートカットがあると相関の減衰ははるかに緩やかになる (Balduzzi et al. 2017)。また重みの側から見ると、深いプレーンネットワークの損失ランドスケープは粗くカオス的であるのに対し、ショートカットは最小点の周囲でそれを滑らかに保つ (Li et al. 2018)。上のヤコビ行列はその理由を示唆している。L 個のブロックにわたって展開すると、因子 $\mathbf{I} + \partial g_l/\partial \mathbf{x}_{l-1}$ の積は 2^L 通りの経路にわたる和となり、勾配の大半は短い経路を通って流れるため、ResNet は比較的浅いネットワークのアンサンブルのように訓練される (Veit et al. 2016)。このようなアンサンブルから 1 つのブロックを取り除いてもほとんど悪影響はなく、7.6 節で扱う確率的深さ（stochastic depth）はこの性質を正則化手法へと転用している。

具体的には、ResNet の基本的な `Residual` ブロックは、バイアスなしの 2 つの 3×3 畳み込みをバッチ正規化および中間の ReLU とともに積み重ね、2 番目のバッチ正規化の出力にショートカット入力 $\mathbf{x}$ を加算し、最後に ReLU を適用する。2 番目のバッチ正規化で $\gamma = \beta = 0$ と設定すれば分岐の出力はゼロになり、非負の入力に対して $\mathrm{ReLU}(\mathbf{x}) = \mathbf{x}$ が復元される。大規模訓練のレシピではこれを初期化に利用し、すべての残差分岐の最後の γ を 0 から始めることで、最初のステップでは各ブロックを恒等写像とし、分岐が徐々に育っていく浅いネットワークとして深いネットワークの訓練を開始する (Goyal et al. 2017; He et al. 2019)。本節で扱うネットワークはデフォルトの $\gamma = 1$ でも十分に訓練できる浅さであるため、そのままにしておく。`use_1x1conv=True` または `strides != 1` によってチャネル幅や空間解像度（spatial resolution）が変化するときは、加算が適切に定義されるよう、ショートカット側にもストライドを合わせた 1×1 畳み込みとバッチ正規化を適用する。

```
class Residual(nnx.Module):
    """The Residual block of ResNet models."""
    def __init__(self, num_channels, use_1x1conv=False, strides=(1, 1),
                 in_channels=None, rngs=None):
        in_channels = num_channels if in_channels is None else in_channels
        strides = (strides, strides) if isinstance(strides, int) else strides
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        # He initialization; no conv biases, since the batch normalization
        # after each conv removes them
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        # padding=1 on every side: 'SAME' pads only right and bottom when the
        # stride is 2 and the sizes are even
        self.conv1 = nnx.Conv(in_channels, num_channels, kernel_size=(3, 3),
                              padding=1, strides=strides, use_bias=False,
                              kernel_init=he, rngs=rngs)
        self.conv2 = nnx.Conv(num_channels, num_channels, kernel_size=(3, 3),
                              padding='same', use_bias=False,
                              kernel_init=he, rngs=rngs)
        # Running averages decay 0.9 and a two-pass variance (the Flax
        # defaults are 0.99 and a one-pass variance)
        bn = lambda: nnx.BatchNorm(num_channels, momentum=0.9,
                                   use_fast_variance=False, rngs=rngs)
        # Auto-enable 1x1 conv when downsampling so the residual shape matches.
        if use_1x1conv or any(s != 1 for s in strides):
            self.conv3 = nnx.Conv(in_channels, num_channels,
                                  kernel_size=(1, 1), strides=strides,
                                  use_bias=False, kernel_init=he, rngs=rngs)
            self.bn3 = bn()
        else:
            self.conv3 = None
        self.bn1, self.bn2 = bn(), bn()
```

```
    def __call__(self, X):
        Y = nnx.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.bn3(self.conv3(X))
        Y += X
        return nnx.relu(Y)
```

NNX の層は構築時に入力チャネル数を必要とするため、JAX 版の `Residual` は引数 `in_channels`（デフォルトは `num_channels`）を受け取る。

Figure 7.10 に両方の構成を示す。`use_1x1conv=True` を渡すか、1 より大きいストライドを設定すると、右側の射影ショートカットが有効になる。

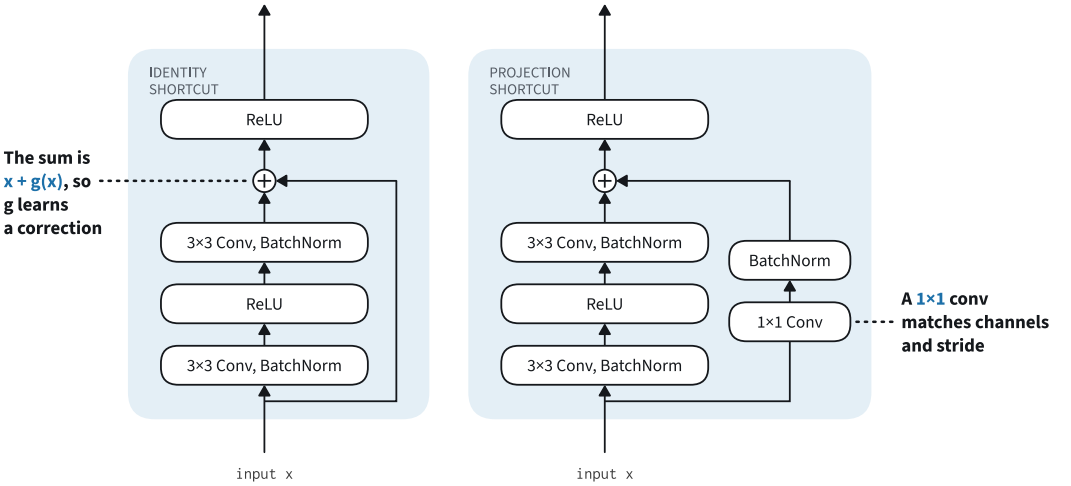


Figure 7.10: ResNet ブロックの 2 つの変種。入出力の形状が一致するときは恒等ショートカットを用い（左）、形状が異なるときはショートカット上の 1×1 畳み込みとバッチ正規化によってチャネル数と解像度を調整し、加算が定義されるようにする（右）。

出力の形状を確認してみよう。入出力のチャネル数が一致しストライドが 1 のとき、このブロックはテンソルの形状 `(4, 3, 6, 6)` をそのまま保つ。

```
blk = Residual(3)
X = jax.random.normal(d2l.get_key(), (4, 6, 6, 3))
blk(X).shape
```

```
(4, 6, 6, 3)
```

`num_channels=6`、`use_1x1conv=True`、`strides=2` と設定すると、チャネル数が 2 倍になると同時に高さと幅がいずれも半分になる。これは、以下で構築する ResNet のステージ 2、3、4 の先頭に置かれるダウンサンプリングブロックそのものである。

```
blk = Residual(6, use_1x1conv=True, strides=(2, 2), in_channels=3)
blk(X).shape
```

```
(4, 3, 3, 6)
```

7.4.3 プレーンネットワークと残差ネットワークの比較

Fashion-MNIST 上で劣化問題を直接再現し、残差ショートカットがそれを解消する様子を観察できるだろうか。条件を揃えた明確な対照実験を行うため、ショートカット接続の有無**のみ**が異なる 2 つのネットワーク族を構築する。He et al. (2016a) による CIFAR-10 向けの設計を Fashion-MNIST 本来の 28×28 画像に適合させることで、深いモデルでも高速に訓練できる。**プレーンブロック**は、迂回経路の加算を除けば `Residual` と層単位で同一であり、2 つの 3×3 畳み込みのそれぞれにバッチ正規化と ReLU が続く構成をとる。

```
def plain_block(num_channels, strides, in_channels, rngs):
    """The layers of `Residual` without the shortcut."""
    he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
    bn = lambda: nnx.BatchNorm(num_channels, momentum=0.9,
                               use_fast_variance=False, rngs=rngs)
    return nnx.Sequential(
        nnx.Conv(in_channels, num_channels, kernel_size=(3, 3), padding=1,
                 strides=strides, use_bias=False, kernel_init=he, rngs=rngs),
        bn(), nnx.relu,
        nnx.Conv(num_channels, num_channels, kernel_size=(3, 3), padding=1,
                 use_bias=False, kernel_init=he, rngs=rngs),
        bn(), nnx.relu)
```

各ネットワークは、16 チャネルの 3×3 畳み込み、バッチ正規化、ReLU から始まり、チャネル数が 16、32、64 である n ブロックずつの 3 つのステージが続き（ステージ 2 と 3 の最初のブロックはストライド 2 でダウンサンプリングする）、最後はグローバル平均プーリングと線形分類器で終わる。ステム、ブロックあたり 2 つの畳み込み、および線形のヘッドを数えると合計 $6n + 2$ 層になり、$n = 3$ では 20 層、$n = 9$ では 56 層となる。`plain_block` または `Residual` のいずれかを `block` として渡すことで、条件を揃えた対となるモデルを構築できる。これまでと同様に、すべての重みを He 初期化（4.4.2.2 節）で初期化し、`d2l.cosine_warmup`（7.1 節）を用いてピーク学習率 0.1 で訓練する。この学習率は 7.2 節では正規化なしのネットワークを不安定化させたが、ここではすべての畳み込みの後にバッチ正規化（7.3 節）があるため滑らかに訓練が進む。

```
class DeepNet(d2l.Classifier):
    """6n + 2 layers: a 3x3 stem, three stages of n blocks, a linear head."""
    def __init__(self, block, n, lr=0.1, num_steps=2345, num_classes=10,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['block', 'rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        # Fashion-MNIST images have one channel
        layers = [nnx.Conv(1, 16, kernel_size=(3, 3), padding=1,
                           use_bias=False, kernel_init=he, rngs=rngs),
                  nnx.BatchNorm(16, momentum=0.9, use_fast_variance=False,
                                rngs=rngs), nnx.relu]
        in_channels = 16
        for i, num_channels in enumerate((16, 32, 64)):
            for j in range(n):  # The first block of stages 2 and 3 downsamples
                strides = 2 if i > 0 and j == 0 else 1
```

```
                layers.append(block(num_channels, strides=strides,
                                    in_channels=in_channels, rngs=rngs))
                in_channels = num_channels
        layers += [global_avg_pool,
                   nnx.Linear(64, num_classes, kernel_init=he, rngs=rngs)]
        self.net = nnx.Sequential(*layers)

    def configure_optimizers(self):
        return optax.sgd(lambda t: self.lr * d2l.cosine_warmup(
            t, self.num_steps, self.num_steps // 10))

def global_avg_pool(X):  # Average over height and width (NHWC)
    return X.mean(axis=(1, 2))
```

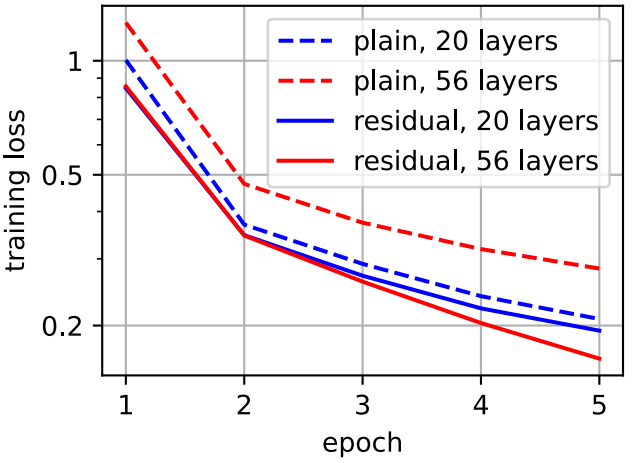
NNX の層は構築時に重みをサンプリングするため、Residual の内部も含むすべての層が He 初期化（variance_scaling(2.0, 'fan_in', 'normal')）を kernel_init として受け取る。

4 つの構成すべて（20 層および 56 層のプレーンネットワークと残差ネットワーク）をバッチサイズ 128 で 5 エポック訓練し、訓練履歴をすべて記録する。

```
data = d2l.FashionMNIST(batch_size=128)
num_steps = 5 * len(data.train_dataloader())
history = {}
for name, block in (('plain', plain_block), ('residual', Residual)):
    for n in (3, 9):
        model = DeepNet(block, n, lr=0.1, num_steps=num_steps)
        trainer = d2l.Trainer(max_epochs=5, num_gpus=1)
        trainer.board.display = False  # We plot the losses ourselves
        trainer.fit(model, data)
        history[f'{name}, {6 * n + 2} layers'] = trainer.history
```

4 本すべての訓練損失の曲線を対数スケールの y 軸上に重ねてプロットしてみよう（青が 20 層、赤が 56 層であり、破線がプレーンネットワーク、実線が残差ネットワークを表す）。

```
d2l.plot(list(range(1, 6)), [h['train_loss'] for h in history.values()],
         'epoch', 'training loss', legend=list(history), yscale='log',
         fmts=('b--', 'r--', 'b-', 'r-'))
```



また、20 層と 56 層の 4 つの構成すべてについて、最終エポックの `train_loss` と `val_acc` を `history` から確認する。

```
{name: (round(h['train_loss'][-1], 3), round(h['val_acc'][-1], 3))
 for name, h in history.items()}  # Training loss, validation accuracy
```

```
{'plain, 20 layers': (0.208, 0.906),
 'plain, 56 layers': (0.283, 0.884),
 'residual, 20 layers': (0.194, 0.904),
 'residual, 56 layers': (0.164, 0.908)}
```

プレーンネットワークには劣化現象が紛れもなく現れている。56 層のプレーンネットワークは最終的な訓練損失が 0.25〜0.30 前後と、20 層のプレーンネットワークの約 0.20 よりも高く、すべてのエポックにおいて浅いモデルの曲線よりも厳密に上に位置している。訓練損失そのものが悪化しているため、過学習の可能性は除外される。一方、残差ショートカットを有効にすると順序は完全に逆転する。56 層の残差ネットワークは訓練損失が約 0.15〜0.17 まで下がり、20 層の両ネットワーク（ショートカットの有無にかかわらず 0.19 付近で終了）を上回る。20 層ではショートカットの影響はほとんど見られないが、56 層では追加の 36 層が最適化を妨げるか改善するかをショートカットが決定づけている。これら 4 回の実験で変えたのは深さとショートカット接続だけであるため、この差がショートカットによるものであることは明白だ。検証正解率にも同じ傾向がより緩やかに現れており、5 エポック後の時点で 56 層のプレーンネットワークが 88〜90% と遅れをとる一方、残りの 3 つは 90.5〜91.5% の狭い範囲に集まっている。

7.4.4 ResNet-18

7.4.4.1 ステム、ステージ、ヘッド

本格的な ImageNet スケールにおいて、ResNet は GoogLeNet（7.2.3 節）が確立したステム・ボディ・ヘッドのアーキテクチャを採用している。その**ステム**は、ストライド 2 で 64 チャネルの 7×7 畳み込み、バッチ正規化、ReLU、およびストライド 2 の 3×3 最大プーリングから始まり、空間次元をいきなり 4 分の 1 に縮小する。以下の `ResNet.stem` はこの入り口のモジュールを構築する。

```
class ResNet(d2l.Classifier):
    def __init__(self, arch, lr=0.1, num_steps=4690, num_classes=10,
                 in_channels=1, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.net = self.create_net(in_channels, rngs)

    def stem(self, in_channels, rngs):
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        return nnx.Sequential(
            nnx.Conv(in_channels, 64, kernel_size=(7, 7), strides=(2, 2),
                     padding=3, use_bias=False, kernel_init=he, rngs=rngs),
            nnx.BatchNorm(64, momentum=0.9, use_fast_variance=False,
                          rngs=rngs), nnx.relu, max_pool)

def max_pool(X):  # 3x3 windows, stride 2, padding 1 on every side
    return nnx.max_pool(X, window_shape=(3, 3), strides=(2, 2),
                        padding=((1, 1), (1, 1)))
```

ボディは 4 つの**ステージ**からなり、各ステージは一定のチャネル幅で num_residuals 個の残差ブロックを積み重ねる。ステムがすでに入力を 4 分の 1 にダウンサンプリングしているため、ステージ 1 は解像度と 64 チャネルをそのまま維持し、それ以降の各ステージは先頭にストライド 2 の射影ブロックを置いて高さと幅を半分にしつつチャネル数を 2 倍にする。stage メソッドはそのようなステージを 1 つ構築する。

```
@d2l.add_to_class(ResNet)
def stage(self, num_residuals, num_channels, in_channels,
          first_block=False, rngs=None):
    blk = []
    for i in range(num_residuals):
        if i == 0 and not first_block:
            blk.append(Residual(num_channels, use_1x1conv=True,
                                strides=(2, 2), in_channels=in_channels,
                                rngs=rngs))
        else:
            blk.append(Residual(num_channels, in_channels=in_channels,
                                rngs=rngs))
        in_channels = num_channels
    return nnx.Sequential(*blk)
```

モデルを締めくくる**ヘッド**は NiN や GoogLeNet に倣い、グローバル平均プーリングに単一の線形分類層が続く構成をとる。VGG と同様に、引数 arch で各ステージの (num_residuals, num_channels) を指定し、すべての畳み込み層と線形層の重みには He 初期化を用い、出力層にある唯一のバイアスは 0 で初期化する。

```
@d2l.add_to_class(ResNet)
def create_net(self, in_channels, rngs):
    layers = [self.stem(in_channels, rngs)]
    stage_channels = 64
    for i, (num_residuals, num_channels) in enumerate(self.arch):
```

```
            layers.append(self.stage(num_residuals, num_channels, stage_channels,
                                     first_block=(i == 0), rngs=rngs))
            stage_channels = num_channels
        layers.append(nnx.Sequential(
            global_avg_pool,
            nnx.Linear(stage_channels, self.num_classes, rngs=rngs,
                       kernel_init=nnx.initializers.variance_scaling(
                           2.0, 'fan_in', 'normal'))))
        return nnx.Sequential(*layers)
```

4 つの各ステージに 2 つずつ残差ブロックを配置すると **ResNet-18** になる。4 ステージ $\times$ 4 つの 3×3 畳み込み（ショートカットの 1×1 射影は除く）に、ステムの 7×7 畳み込みと最終の線形層を加えると、パラメータをもつ層は合計 18 層になる。ResNet-34 はステージごとに `(3, 4, 6, 3)` 個のブロックを用い、さらに深いモデル（ResNet-50、ResNet-101、ResNet-152）では 3 層の**ボトルネック**ブロック（$1 \times 1 \to 3 \times 3 \to 1 \times 1$）に置き換える。これは後述する ResNeXt ブロックでグループ数を 1 とした場合に一致する。Figure 7.11 に ResNet-18 の全体構造を示す。

```
class ResNet18(ResNet):
    def __init__(self, lr=0.1, num_steps=4690, num_classes=10, in_channels=1,
                 rngs=None):
        super().__init__(((2, 64), (2, 128), (2, 256), (2, 512)),
                         lr, num_steps, num_classes, in_channels, rngs)
```

1 チャネルの 96×96 画像を ResNet-18 に通し、グローバル平均プーリングに向かって解像度が段階的に下がりチャネル幅が広がっていく際のテンソル形状をたどってみよう。

```
ResNet18().layer_summary((1, 96, 96, 1))
```

```
Sequential              (1, 24, 24, 64)
Sequential              (1, 24, 24, 64)
Sequential              (1, 12, 12, 128)
Sequential              (1, 6, 6, 256)
Sequential              (1, 3, 3, 512)
Sequential              (1, 10)
```

7.4.4.2 訓練

`ResNet18` を訓練するため、まず `DeepNet` のコサインウォームアップ付き SGD スケジュールを `configure_optimizers` によって設定する。

```
ResNet.configure_optimizers = DeepNet.configure_optimizers  # Same schedule
```

それでは、`ResNet18` を 96×96 にリサイズした Fashion-MNIST 上で、バッチサイズ 128、ピーク学習率 `lr=0.1` で 10 エポック訓練しよう。

```
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
model = ResNet18(lr=0.1, num_steps=10 * len(data.train_dataloader()))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
```

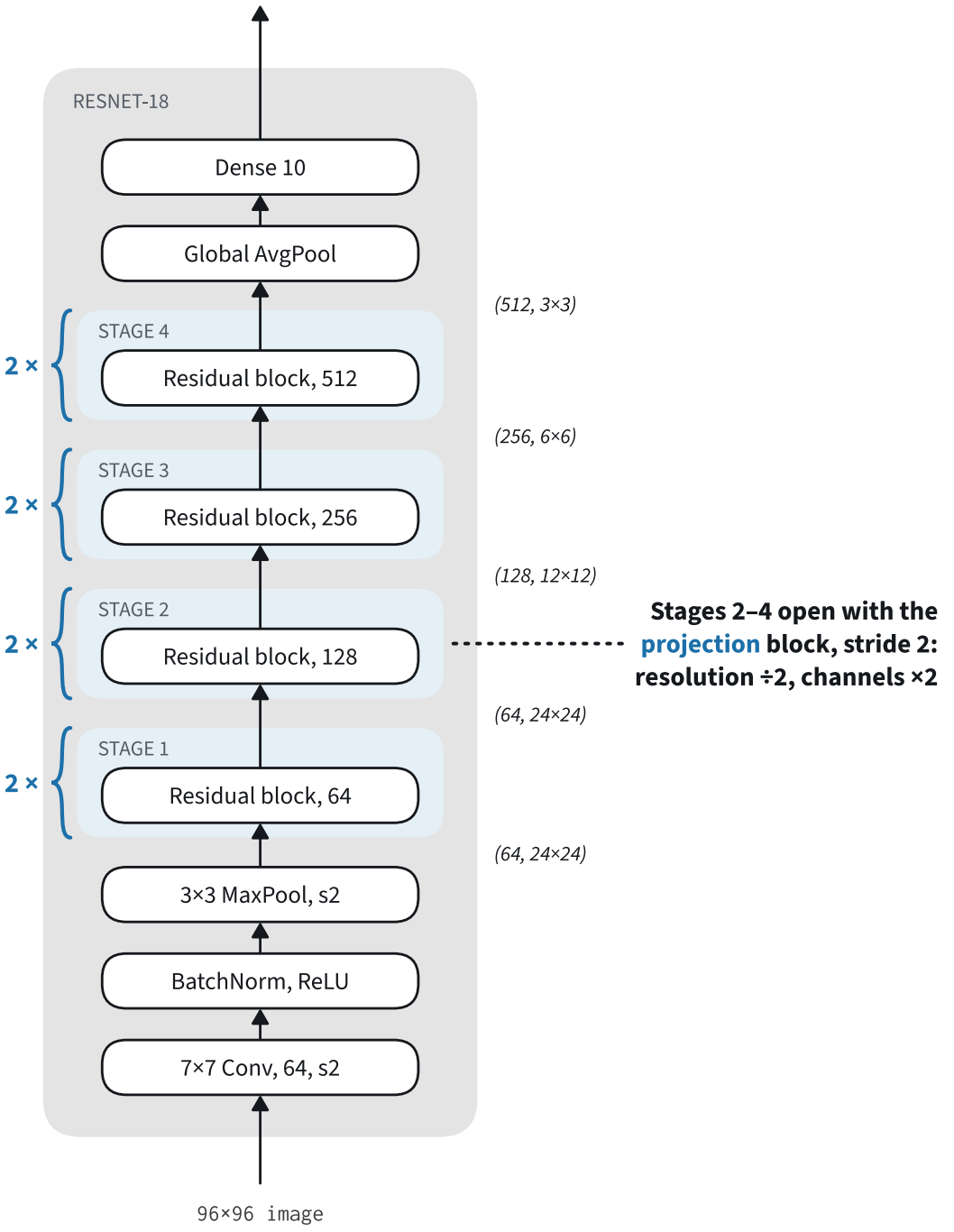
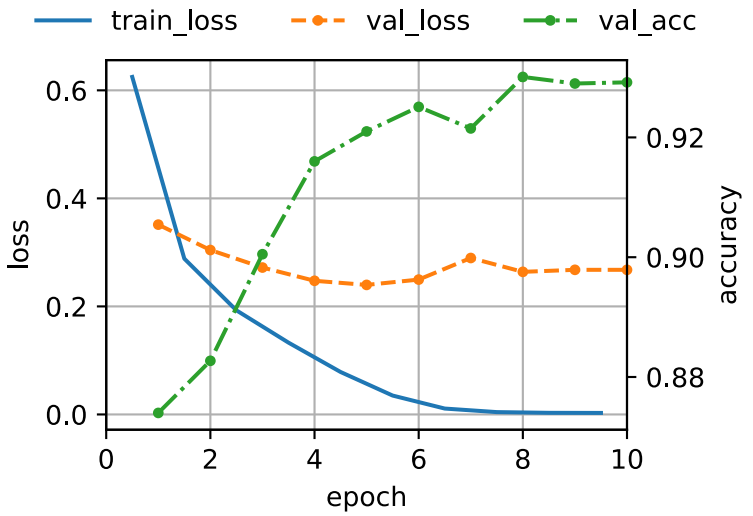


Figure 7.11: ResNet-18 の構造。7 × 7 畳み込みのステム、残差ブロックを 2 つずつもつ 4 つのステージ（ステージ 2、3、4 の最初のブロックで解像度を半分にしチャネル数を 2 倍にする）、およびグローバル平均プーリングと 1 つの全結合層からなるヘッドで構成される。形状は 96 × 96 入力に対するものである。



`trainer.history['val_acc']` の最後の要素を読み取ることで、10 エポックのコサインスケジュール終了時点での検証正解率が得られる。

```
round(trainer.history['val_acc'][-1], 3)  # Validation accuracy after 10 epochs
```

```
0.929
```

10 エポック後、ResNet-18 は約 93% の検証正解率に達し（最後の 3 エポックでの変動は 1 ポイント未満）、7.2 節の 4 分の 1 幅の VGG と同等の精度を、わずか 4 分の 1 のパラメータ数、かつ 96×96 ピクセル（224×224 ピクセルではなく）の入力で達成している。もっとも、アーキテクチャ、正規化、入力解像度がすべて同時に変化している異なるモデル族間の単一の比較を過大解釈しないよう注意すべきである。残差接続（residual connection）の真価を示す明確な証拠は、7.4.3 節で行った 20 層対 56 層の対照実験にある。

また、過学習の典型的な兆候にも注目しよう。ResNet-18 は訓練損失を 0.01 未満まで引き下げる一方、検証損失は最初の数エポック以降 0.27 前後で頭打ちになる。60,000 枚のグレースケール衣類画像に対して 1,120 万個のパラメータを訓練しているため、訓練セットの丸暗記は容易である。この汎化ギャップ（generalization gap）を縮めるにはより強力なデータ拡張（data augmentation）と正則化が必要であり、これについては 7.6 節で体系的に取り組む。

7.4.5 ResNeXt と DenseNet

残差接続は、ブロックが学習した変換に入力を加算するという形で、ブロックが入力を**どのように**先へ伝えるかを規定する。代表的な 2 つの後継アーキテクチャは、ショートカット配線という思想を受け継ぎながらブロック自体を改良している。**ResNeXt** はブロック内部の畳み込みを再設計して計算量を増やさずにチャネル幅を広げ、**DenseNet** は加算型のショートカットをチャネルの連結（concatenation）に置き換える。ResNeXt ブロックについては、7.8 節で体系的なネットワーク設計空間を探究する際に再び取り上げる。

7.4.5.1 ResNeXt

幅広い層はより豊かな表現を学習できるが、c_i チャネルを c_o チャネルに写す標準的な畳み込みのコストは $c_\text{i}c_\text{o}$ に比例して増大するため（6.4 節）、幅を 2 倍にすると重みと演算量の双方が 4 倍に

なる。**ResNeXt** (Xie et al. 2017) は、GoogLeNet の多分岐 Inception の発想（7.2.3 節）を、すべての分岐がまったく同じトポロジーをもつ完全に均一な構造へと発展させたものである。

具体的には、**グループ畳み込み**は c_i 個の入力チャネルを互いに素な g 個のグループ（各 c_i/g チャネル）に分割し、各グループを独立に畳み込んで c_o/g 個の出力チャネルを生成する（ただし g は c_i と c_o の両方を割り切ると仮定する）。各グループに必要なカーネルのサイズは $(c_\text{i}/g) \times (c_\text{o}/g) \times k \times k$ であり、全 g グループにわたる総パラメータ数と積和演算（multiply-add）の回数は g 分の 1 に削減され、$c_\text{i}c_\text{o}k^2$ から $c_\text{i}c_\text{o}k^2/g$ へと減る（実際には、メモリアクセス量が演算量と同じ割合では減らないため、実時間での高速化は通常 g 倍未満にとどまる。この違いについては 7.5 節で詳しく調べる）。歴史的には、グループ畳み込みは AlexNet (Krizhevsky et al. 2012) で初めて導入され、2 基の GPU に収めるために 5 つの畳み込み層のうち 3 つを 2 グループに分割するのに使われた。

独立したグループ同士はそれ自体では情報を交換できないため、ResNeXt ブロック（Figure 7.12）はグループ化した 3×3 畳み込みを 2 つの密な 1×1 畳み込みで挟み込む。1 つ目の畳み込みが c 個の入力チャネルを b 個の**ボトルネック**チャネルへと射影し、2 つ目の畳み込みが再び c チャネルへ射影して全グループ間の情報を混合する。2 つの 1×1 層は $2cb$ 個の重みを用い、グループ化した 3×3 層は $9b^2/g$ 個の重みを用いる。ダウンサンプリング時には、ストライド 2 の処理を 3×3 畳み込みに持たせ（最初の 1×1 射影ではなく）、ショートカットは `Residual` と同じ形にする。

「同等のコストでより幅広く」が数値的に何を意味するかを見るため、標準的な ResNet-50 のボトルネックブロック（$c = 256$、$b = 64$、$g = 1$）で使用される $2 \cdot 256 \cdot 64 + 9 \cdot 64^2 = 69{,}632$ 個の重みと、Figure 7.12 に示す ResNeXt ブロック（$b = 128$、$g = 32$）で使用される $2 \cdot 256 \cdot 128 + 9 \cdot 128^2/32 = 70{,}144$ 個の重みを比較してみよう。重みの増加を 1% 未満に抑えながら、内部のボトルネック幅が 2 倍になっている。ImageNet では、この単純なトレードオフによって ResNet-50 の top-1 誤差が 23.9% から 22.2% へと低下する (Xie et al. 2017)。

以下の `ResNeXtBlock` は、出力幅 `num_channels`（c）、グループ数 `groups`（g）、およびボトルネック比 `bot_mul`（b/c）を受け取る。`Residual` と同様に、`strides=2` を渡すと高さと幅が半分になり、ショートカットの 1×1 射影が自動的に有効になる。

```
class ResNeXtBlock(nnx.Module):
    """The ResNeXt block."""
    def __init__(self, num_channels, groups, bot_mul, use_1x1conv=False,
                 strides=(1, 1), in_channels=None, rngs=None):
        in_channels = num_channels if in_channels is None else in_channels
        strides = (strides, strides) if isinstance(strides, int) else strides
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        bot_channels = int(round(num_channels * bot_mul))
        # He initialization; no conv biases, since the batch normalization
        # after each conv removes them
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        self.conv1 = nnx.Conv(in_channels, bot_channels, kernel_size=(1, 1),
                              strides=(1, 1), use_bias=False,
                              kernel_init=he, rngs=rngs)
        self.conv2 = nnx.Conv(bot_channels, bot_channels,
                              kernel_size=(3, 3), strides=strides,
                              padding=1, feature_group_count=groups,
                              use_bias=False, kernel_init=he, rngs=rngs)
        self.conv3 = nnx.Conv(bot_channels, num_channels, kernel_size=(1, 1),
                              strides=(1, 1), use_bias=False,
```

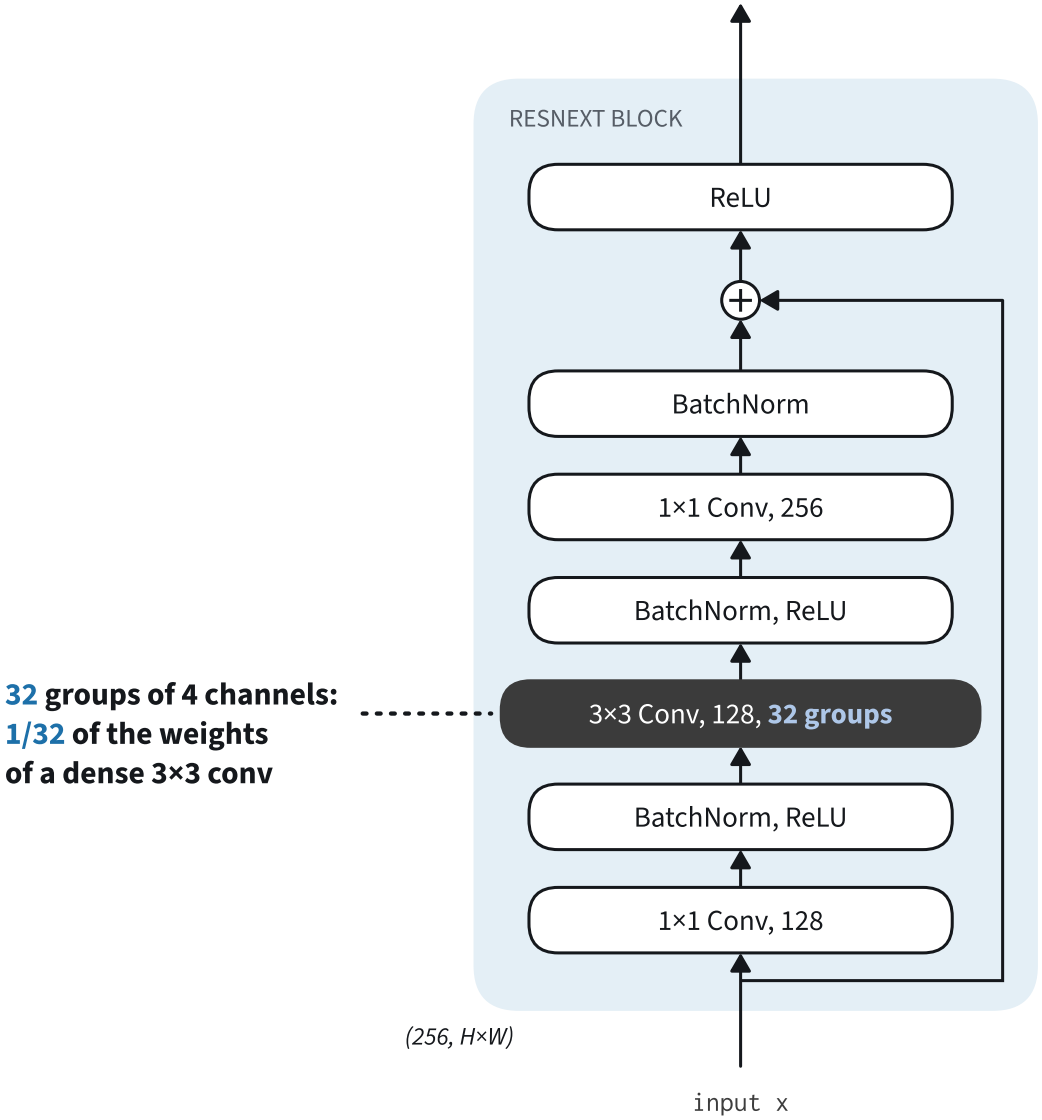


Figure 7.12: ResNeXt ブロック。3×3 のグループ畳み込み（グループ数 g）を、グループ間で情報を混合する 2 つの 1×1 畳み込みで挟む。図のブロックはチャネル数 $c = 256$、ボトルネックチャネル数 $b = 128$、4 チャネルずつのグループ数 $g = 32$ である。

```
                            kernel_init=he, rngs=rngs)
        # Running averages decay 0.9 and a two-pass variance (the Flax
        # defaults are 0.99 and a one-pass variance)
        bn = lambda n: nnx.BatchNorm(n, momentum=0.9, use_fast_variance=False,
                                     rngs=rngs)
        self.bn1, self.bn2 = bn(bot_channels), bn(bot_channels)
        self.bn3 = bn(num_channels)
        # Auto-enable 1x1 conv when downsampling so the residual shape matches.
        if use_1x1conv or any(s != 1 for s in strides):
            self.conv4 = nnx.Conv(in_channels, num_channels,
                                  kernel_size=(1, 1), strides=strides,
                                  use_bias=False, kernel_init=he, rngs=rngs)
            self.bn4 = bn(num_channels)
        else:
            self.conv4 = None

    def __call__(self, X):
        Y = nnx.relu(self.bn1(self.conv1(X)))
        Y = nnx.relu(self.bn2(self.conv2(Y)))
        Y = self.bn3(self.conv3(Y))
        if self.conv4:
            X = self.bn4(self.conv4(X))
        return nnx.relu(Y + X)
```

32 チャネル、16 グループ、`bot_mul=1` のブロックが入力テンソルの形状を保つことを確認しよう。

```
blk = ResNeXtBlock(32, 16, 1)
X = jax.random.normal(d2l.get_key(), (4, 96, 96, 32))
blk(X).shape
```

```
(4, 96, 96, 32)
```

7.4.5.2 DenseNet

残差ブロックが出力を入力に加算し（$\mathbf{x}_k = \mathbf{x}_{k-1} + g_k(\mathbf{x}_{k-1})$）、チャネル幅を一定に保ちながら古い特徴と新しい特徴を混ぜ合わせるのに対し、**DenseNet**（dense convolutional network、密結合畳み込みネットワーク）(Huang, Liu, et al. 2017) は各層の出力を入力に対してチャネル次元に沿って**連結**する。

$$\mathbf{x}_k = [\mathbf{x}_{k-1}, f_k(\mathbf{x}_{k-1})]\,,$$

ここで、$[\cdot,\cdot]$ はチャネル方向の連結を表す。いかなる情報も上書きされることはない。$\mathbf{x}_k$ は元のブロック入力と先行する k 個すべての層の出力をそのまま保持するため、ブロック内のすべての層がそれ以前のすべての層と直接接続される。Figure 7.13 に 2 層の密ブロック（dense block）を示す。

密ブロックの内部では、各畳み込みはプレアクティベーションの順序（バッチ正規化 → ReLU → バイアスなしの畳み込み）を採用する。この順序を ResNet に適用した場合については後述の演習問題で調べる。各層が新たに追加するチャネル数（Figure 7.13 では 32）は**成長率**（growth rate）と呼ばれる。どの密ブロックの内部でもチャネル幅が線形に増加するため、DenseNet は連続するブロックの間に**遷移層**（transition layer）を挿入し、1×1 畳み込みでチャネル数を圧

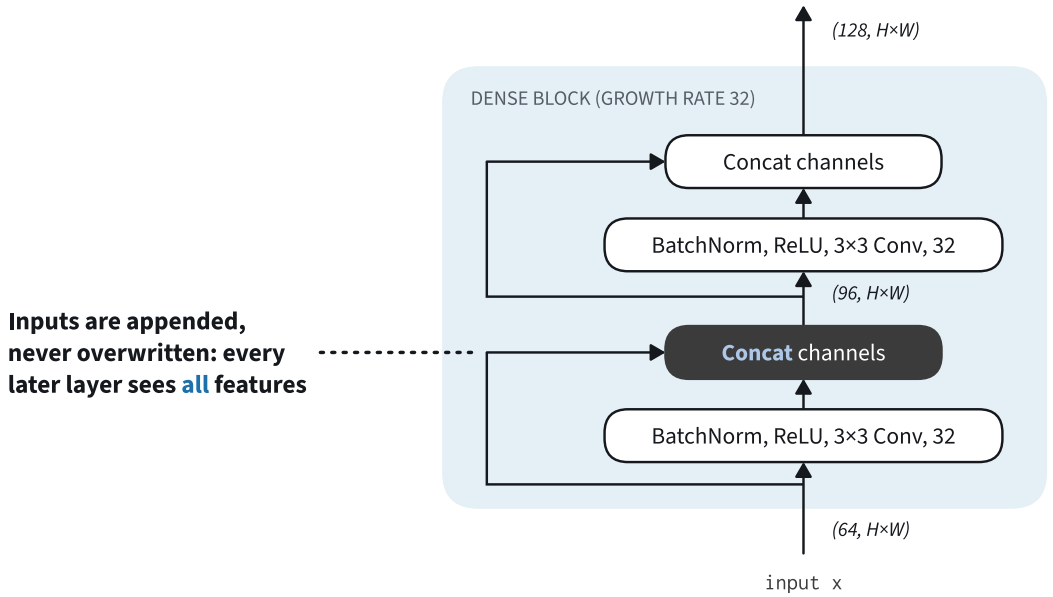


Figure 7.13: 2 層で成長率が 32 の密ブロック。各層が 32 個の出力チャネルを入力に追加するため、64 チャネルの入力は 96 チャネル、次いで 128 チャネルへと増加し、すべての層がそれ以前の全層の出力を参照する。

縮するとともにストライド 2 の平均プーリングで高さと幅を半分にする。完全な DenseNet は遷移層を挟んで 4 つの密ブロックを積み重ねており、ResNet-18 の 4 ステージ構成と対応している（その構築と訓練の手順は 7 で扱う）。

DenseNet はより少ないパラメータ数で ResNet に匹敵する ImageNet 正解率を達成し、著者らはその要因を徹底した特徴の再利用に帰した。しかし実用上の欠点は活性化メモリ（activation memory）にある。**固有の**特徴マップの数はブロックの深さに対して線形にしか増えないにもかかわらず、素朴な実装では層ごとにより幅の広い新たな連結テンソル（およびその正規化後のコピー）を確保して誤差逆伝播法のためにキャッシュするため、逆伝播の際に中間テンソルを再計算する特殊な実装を用いない限り、メモリ使用量が深さの二**乗**に比例して増大してしまう (Pleiss et al. 2017)。加算型の残差ショートカットはこのメモリオーバーヘッドを完全に回避できる。これが、加算型のスキップ接続（skip connection）が深層学習全般で普遍的な標準となった主要な理由の 1 つである。

それでもなお、連結型のスキップ接続は 1 つの主要なアーキテクチャ族において中心的な役割を担い続けている。それが画像セグメンテーション（image segmentation）向けの **U-Net** (Ronneberger et al. 2015) であり、画像を段階的にダウンサンプリングしたのち元の解像度までアップサンプリングし、アップサンプリングされた各特徴マップをエンコーダ経路の対応する高解像度マップと連結する。連結は層ごとではなく空間スケールごとに 1 回しか行われないため、メモリ消費は控えめに抑えられる。この同じ U-Net バックボーンはのちに現代の画像拡散モデル（diffusion model、17.4 節）の原動力ともなった。

7.4.6 まとめ

驚くべきことに、バッチ正規化を施したプレーンネットワークでは、深いアーキテクチャの関数クラスが浅いモデルの関数クラスを厳密に含んでいるにもかかわらず、層を積み増すことで浅

いモデルよりも訓練損失が**増加**してしまうことがある。残差ブロックは加算的な補正 $\mathbf{x} + g(\mathbf{x})$ を学習することでこの劣化問題を解決する。これにより恒等写像が $g = 0$ に位置するようになり、深さが増してもショートカットが勾配の一貫性と損失ランドスケープの滑らかさを保つ。Fashion-MNIST では、56 層のプレーンネットワークは 20 層のモデルより訓練結果が悪化するのに対し、残差ショートカットを導入すると 56 層のモデルの方が良好に訓練される。残差ブロックをバッチ正規化とともに 4 つのステージに配置することで ResNet-18（Fashion-MNIST で 10 エポック後に検証正解率約 93%）が得られ、150 層を超える深さへも無理なくスケールする。ResNeXt はグループ畳み込みによって同等のコストでボトルネックブロックの幅を広げ、DenseNet はメモリコストの増加と引き換えに加算型ショートカットを特徴の連結へと置き換える。以来、残差接続は第 11 章の Transformer をはじめ、現代のほぼすべてのアーキテクチャに不可欠な基盤となっている。

7.4.7 演習問題

1. **Inception ブロックと残差ブロックの比較**。GoogLeNet の `arch`（7.2.3 節）において、中間ステージの 2 番目のブロックは分岐幅 $c_1 = 160$、$c_2 = (112, 224)$、$c_3 = (24, 64)$、$c_4 = 64$ をもち、512 個の入力チャネルを 512 個の出力チャネルへ写像する。これを恒等ショートカット付きの 512 チャネル `Residual` ブロックと比較せよ。
   1. 両ブロックの畳み込みの重み数と出力ピクセルあたりの積和演算回数を数えよ（バイアスとバッチ正規化のパラメータは無視してよい）。
   2. 分岐の出力がゼロである `Residual` ブロックは、非負の入力（ブロックが ReLU の直後に続く場合は常に成り立つ）に対して恒等写像を計算する。訓練モードと予測モードにおいて `bn2` が何を出力するかを `conv2` をゼロにした場合について確認し、両モードでブロックを厳密な恒等写像にする `bn2` のパラメータを述べよ。また、4 分岐の Inception ブロックが非負の入力に対して恒等写像を計算するようなパラメータの選び方は存在するか。
2. **プレアクティベーションの順序**。@{He.Zhang.Ren.ea.2016**1}** **はバッチ正規化と ReLU を各畳み込みの**前 * に移し（BN → ReLU → Conv）、加算後の ReLU を取り除くことで、各ブロックが手つかずの恒等ハイウェイを保ったまま $\mathbf{x}_k = \mathbf{x}_{k-1} + g_k(\mathbf{x}_{k-1})$ を計算するようにした（7.4.5.2 節の DenseNet ブロック内部と同じ順序である）。
   1. 積み重ねた K 個のプレアクティベーションブロックについて、$\mathbf{x}_K$ を $\mathbf{x}_0$ と各分岐出力を用いて表し、ヤコビ行列 $\partial \mathbf{x}_K / \partial \mathbf{x}_0$ が加算項として厳密な単位行列 $\mathbf{I}$ を含むことを示せ。
   2. 標準的なポストアクティベーション型の `Residual` ブロック（それぞれ加算後に ReLU を適用する）を K 個積み重ねた場合、$\mathbf{I}$ に代わって $\partial \mathbf{x}_K / \partial \mathbf{x}_0$ に現れる因子は何か。
3. [code] **プレーンネットワークの勾配**。●He et al. (2016a) は、バッチ正規化を施したプレーンネットワークが深さとともに劣化するにもかかわらず勾配消失を起こしていないことを指摘した。7.4.3 節の 20 層および 56 層の 4 つのネットワークでこのことを確認せよ。
   1. 初期化直後と 1 エポックの訓練後に、1 つのミニバッチ上で損失の勾配を計算し、4 つのネットワークすべてについて各畳み込み層の重み勾配のフロベニウスノルムを層のインデックスに対してプロットせよ。56 層のプレーンネットワークの勾配は入力層に向かうにつれて消失しているか。
   2. 56 層のプレーンネットワークを 5 エポックではなく 20 エポック訓練せよ。その訓練損失は 20 層のプレーンネットワークに追いつくか。
4. [code] **ResNet-34 と ResNet-50 のボトルネック**。●He et al. (2016a) の表 1 は、ステージの深さとブロックの種類によって ResNet-18 から ResNet-152 までを定義している。

1. ResNet-34（`arch=((3, 64), (4, 128), (6, 256), (3, 512))`）を `ResNet` を用いて構築し、そのパラメータ数を `ResNet18` と比較せよ。
2. ResNet-50 は 2 層の `Residual` ブロックを 3 層のボトルネックブロック（1×1 で幅を $4\times$ 縮小し、3×3 を経て、1×1 で幅を復元する構造であり、`ResNeXtBlock` で `groups=1, bot_mul=0.25` としたもの）に置き換え、ステージ出力幅を $4\times$ 広い（`256, 512, 1024, 2048`）にする。幅 $4c$ の ResNet-50 ボトルネックブロック 1 個のパラメータ数と積和演算回数を幅 c の ResNet-34 の `Residual` ブロック 1 個と比較し、続いて `stage` を `ResNet` のサブクラスでオーバーライドして ResNet-50 を構築し、総パラメータ数を確認せよ。

5. [code] **ResNet-D のダウンサンプリングショートカット**。ストライド 2 の `Residual` ブロックでは、ショートカットの 1×1 でストライド 2 の畳み込みは 4 つの空間位置のうち 1 つしかサンプリングせず、残りの 3 つを捨ててしまう。He et al. (2019) はこれを、2×2 でストライド 2 の平均プーリングに続けてストライド 1 の 1×1 畳み込みとバッチ正規化を行う構成に置き換えている。この ResNet-D ショートカットを `Residual` のサブクラスとして実装し、出力の形状とパラメータ数が変わらないことを確認したうえで、Fashion-MNIST における 10 エポック訓練後の検証正解率を複数回の実行にわたって `ResNet18` と比較せよ。

6. [code] **プレアクティベーション ResNet-18**。●He et al. (2016b) のプレアクティベーションブロックを `Residual` の変種として実装し、`stage` を `ResNet` のサブクラスでオーバーライドして積み重ね、その訓練曲線を `ResNet18` と比較せよ。

7. [code] **DenseNet**。●7.4.5.2 節の DenseNet を構築して訓練し、そのパラメータ数と活性化メモリを計算せよ。

   1. `DenseBlock`（`num_convs` 個の $(\text{BN} \rightarrow \text{ReLU} \rightarrow \text{Conv}_{3\times3} \text{ without bias})$ ユニットを適用し、`growth_rate` チャネルの各出力を入力にチャネル軸に沿って連結するブロック）と、遷移層 $(\text{BN} \rightarrow \text{ReLU} \rightarrow \text{Conv}_{1\times1} \text{ without bias} \rightarrow \text{AvgPool}_{2\times2,s=2})$ を実装せよ。3 チャネルの入力に対し、畳み込みが 2 個で成長率 10 の密ブロックが $3 + 10 + 10 = 23$ チャネルを返し、10 チャネルの遷移層がそれを 10 チャネルに減らしつつ高さと幅を半分にすることを確認せよ。
   2. `ResNet` のステム、畳み込み 4 個・成長率 32 の 4 つの密ブロック、連続する密ブロック間でチャネル数を半分にする遷移層、および BN $\rightarrow$ ReLU に続く `ResNet` のヘッドから DenseNet を組み立てよ。これを 96×96 の Fashion-MNIST 上で訓練し、パラメータ数、検証正解率、1 エポックあたりの実行時間を `ResNet18` と比較せよ。
   3. n 個の畳み込みがそれぞれ r チャネルを出力する密ブロック（入力は c チャネル）と、n 個の `Residual` ブロック（一定の幅 c）を比較せよ。連結されたすべての入力テンソルを逆伝播のために保持する素朴な実装において、i 番目の畳み込みは何チャネルを入力として消費するか。また $i = 1, \ldots, n$ にわたるその総和はいくらか。この総和を、保持される $2nc$ チャネル（n 個の残差ブロックの場合）および $c + nr$ 個の固有チャネル（Pleiss et al. (2017) の手法による場合）と、$c = 64, r = 32, n = 6$ のもとで比較せよ。

7.5 効率的な畳み込みネットワーク

これまでに学んだアーキテクチャは、データセンターの GPU を用いて学術ベンチマークの正解率リーダーボードで首位に立つよう設計されたものであった。しかし実環境での運用では、膨大な数の畳み込みネットワークがスマートフォン、スマートカメラ、自動運転車、組み込みボード上で動作しており、そこではレイテンシ、メモリ使用量、バッテリー消費が厳格な予算制約となる。ベンチマークの正解率がどれほど高くても、予算を超過するモデルは製品として出荷できないのである。カメラからのフレームをリアルタイムに分類するスマートフォンを考えてみよう。

VGG-16 は 224×224 の単一フレームに対して 153 億回の積和演算を必要とし、毎秒 30 フレームでは毎秒 4600 億回の積和演算に相当する。さらに、1 億 3800 万個の単精度パラメータは、端末上の他のあらゆるアプリと並行して 550 MB の RAM を消費する。畳み込みネットワークのコストを 1 桁削減するにはどうすればよいか、またそれによって正解率にどのようなトレードオフが生じるのだろうか。

本節では、効率的な畳み込みネットワークの設計を支える 3 つの主要なアイデアを探る。**因子分解**（factorization）は、各密な畳み込みを、軽量なチャネルごとの空間フィルタと 1×1 のチャネル混合層のペアへと分解する。**スケーリングと探索**は、強力な基本ブロックを固定したうえで、各ステージの深さ、幅、入力解像度を体系的に最適化する。**構造的再パラメータ化**（structural re-parameterization）は、最適化しやすいマルチブランチ構造を訓練し、その後それらのブランチを代数的に統合して、実運用向けの高速な単一の畳み込みへと変換する。その過程で、因子分解を正解率と実経過時間の両面から直接ベンチマークし、示唆に富む教訓を明らかにする。すなわち、理論上の積和演算数だけでは実際の速度はほとんど語れず、因子分解が単に密なモデルの幅を狭める手法を上回るかどうかは、タスクの規模に強く依存するのである。

```
import time
```

7.5.1 デプスワイズ分離可能ネットワーク

密な空間畳み込みは畳み込みネットワークの演算の大半を占めるため、効率化への最も直接的な道筋は畳み込み層そのものを因子分解することである。

7.5.1.1 デプスワイズ分離可能畳み込み

標準的な密な畳み込みでは、すべての出力チャネルが独自の空間フィルタを通してすべての入力チャネルを読み取るため、計算量と重み数は $c_\mathrm{i} c_\mathrm{o}$ に比例して増加する。7.4.5.1 節で見た ResNeXt のグループ畳み込み（grouped convolution）は、チャネルを互いに素な g 個のグループに分割し、各出力グループを対応する入力グループのみに制限することでこの全対全の結合を断ち切り、重みと積和演算の両方を g で割った規模に削減した。Figure 7.14 に示すように、グループ化によって密なチャネル接続行列はブロック対角パターンへと変わる。

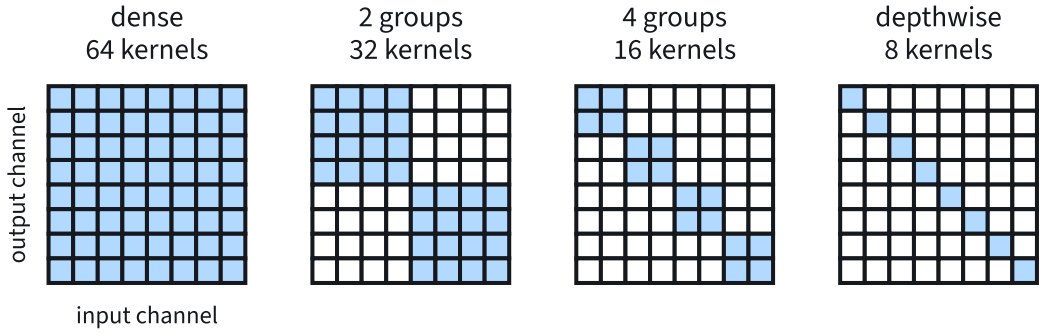


Figure 7.14: 8 個の入力チャネルと 8 個の出力チャネルにおいて、各出力チャネルが読み取る入力チャネル。網掛けの各セルは 1 つの $k_\mathrm{h} \times k_\mathrm{w}$ カーネルを表し、g 個のグループに分けると全体の $1/g$ だけが残る。

ここで、グループ化を極限である $g = c_\mathrm{i} = c_\mathrm{o}$ まで推し進め、各グループが単一のチャネルのみを含むようにしてみよう。この**デプスワイズ畳み込み**（depthwise convolution）は、チャネル

間の混合を一切伴わずに入力チャネルごとに 1 つの $k \times k$ 空間フィルタを独立に適用する。これは、空間的な相互作用を一切伴わずに各ピクセルですべてのチャネルを混合する 1×1 畳み込み（6.4.3 節）とちょうど相補的な関係にある。デプスワイズな $k \times k$ 畳み込みと**ポイントワイズ**な 1×1 畳み込みを連結すれば、2 つの低コストなステップで空間フィルタリングとチャネル混合の両方を回復でき、**デプスワイズ分離可能畳み込み**（depthwise-separable convolution）が得られる (Chollet 2017; Howard et al. 2017)。Figure 7.15 に、この 2 段階の因子分解と置き換え元の密な畳み込みの対比を示す。

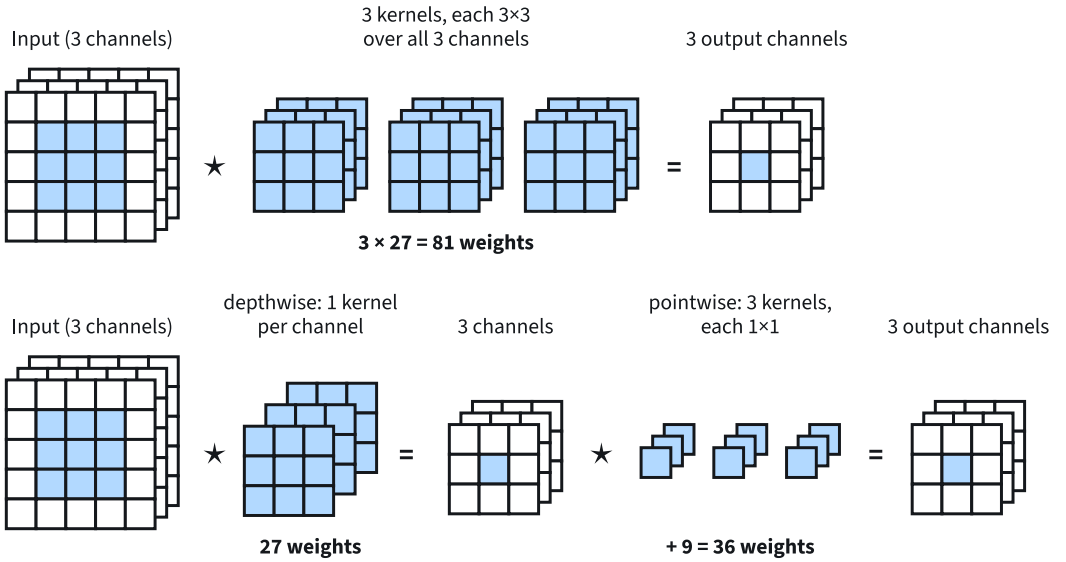


Figure 7.15: 3 個の入力チャネルと 3 個の出力チャネルに対する 3×3 カーネル。上：密な畳み込みでは、各出力チャネルが独自のカーネルスタックを通してすべての入力チャネルを読み取る。下：デプスワイズ畳み込みが全出力チャネルに共通して各チャネルを個別にフィルタリングし、続いてポイントワイズ畳み込みが出力チャネルごとに 1 つの 1×1 カーネルスタックを用いてチャネルを混合する。

削減効果を定量化してみよう。出力を $h \times w$、カーネルを $k \times k$ とすると、密な畳み込みは $hw \cdot c_{\mathrm{i}} c_{\mathrm{o}} k^2$ 回の積和演算を必要とするのに対し、デプスワイズ分離可能ペアが必要とするのは空間フィルタリングの $hw \cdot c_{\mathrm{i}} k^2$ 回とポイントワイズ混合の $hw \cdot c_{\mathrm{i}} c_{\mathrm{o}}$ 回の和である。両者の比は次の式で与えられる。

$$\frac{hw \cdot c_{\mathrm{i}} k^2 + hw \cdot c_{\mathrm{i}} c_{\mathrm{o}}}{hw \cdot c_{\mathrm{i}} c_{\mathrm{o}} k^2} = \frac{1}{c_{\mathrm{o}}} + \frac{1}{k^2}, \tag{7.6}$$

パラメータ数についてもまったく同じ比が成り立つ。Figure 7.15（$c_{\mathrm{i}} = c_{\mathrm{o}} = k = 3$）では、密な層が $3 \cdot 27 = 81$ 個の重みを使うのに対し、分離可能ペアは $27 + 9 = 36$ 個で済む（比は $1/3 + 1/9 = 4/9$）。出力チャネルをもう 1 つ増やすのに必要な新しい重みは、密な層では 27 個であるのに対し、分離可能層ではわずか 3 個である。c_{o} が大きく $k = 3$ のとき、この比は $1/9$ に近づき、重みと演算量の両方でほぼ 9 分の 1 への削減となる。

むろん、この削減は無償ではなく、実効的なフィルタテンソルに対する構造的制約から生じている。分離可能ペアは単一の空間フィルタ $\mathbf{D}_c$ を入力チャネル c に適用し、フィルタリングされたマップをスカラーのポイントワイズ重み $P_{o,c}$ で線形結合するため、カーネルが $\mathbf{K}_{o,c} = P_{o,c} \mathbf{D}_c$

という分解形式に制約された密な畳み込みと等価になる。すべての出力チャネルが入力チャネル c 上で同じ空間フィルタを（スカラー倍を除いて）共有しなければならないのに対し、密な畳み込みは独立した任意の $k \times k$ フィルタをすべての (o, c) の組に対して学習できる。効率的なネットワークはこのランク 1 の制約を受け入れ、浮いた計算予算をより深い、あるいはより広いステージへと再投資するのである。

バイアスなしの 3×3 層で 128 チャネルを 128 チャネルに写像する場合について、パラメータ比をコードで確認してみよう。この設定では、Equation 7.6 から削減倍率は $(1/128 + 1/9)^{-1} \approx 8.4$ と予測される。

Flax の層はチャネルラストの入力を受け取るため、以下の `X` の形状は $(1, 32, 32, c_\text{i})$ である。

```
c_i, c_o, k = 128, 128, 3
X = jax.random.normal(d2l.get_key(), (1, 32, 32, c_i))
dense = nnx.Conv(c_i, c_o, kernel_size=(k, k), padding='SAME',
                 use_bias=False, rngs=nnx.Rngs(d2l.get_key()))
depthwise = nnx.Conv(c_i, c_i, kernel_size=(k, k), padding='SAME',
                     feature_group_count=c_i, use_bias=False,
                     rngs=nnx.Rngs(d2l.get_key()))
pointwise = nnx.Conv(c_i, c_o, kernel_size=(1, 1), use_bias=False,
                     rngs=nnx.Rngs(d2l.get_key()))
assert dense(X).shape == pointwise(depthwise(X)).shape
p_dense = dense.kernel[...].size
p_sep = depthwise.kernel[...].size + pointwise.kernel[...].size
p_dense, p_sep, p_dense / p_sep
```

```
(147456, 17536, 8.408759124087592)
```

密な層が 147,456 個の重みを使用するのに対し、分離可能ペアは 17,536 個であり、ちょうど $8.4\times$ の削減となっている。

Flax では、`feature_group_count=c_i` を指定すると `nnx.Conv` がデプスワイズ畳み込みになる。この引数はグループ数 g を表す。

7.5.1.2 MobileNet

MobileNet は、デプスワイズ分離可能畳み込みをモバイル向けの完全なバックボーンへと発展させた (Howard et al. 2017)。最初の単一の密なステム畳み込みの後、すべての層で Figure 7.16 のブロック、すなわちデプスワイズ 3×3 畳み込みに続けてポイントワイズ 1×1 畳み込みを配置し、それぞれにバッチ正規化（batch normalization）と ReLU を組み合わせた構造を用いる。空間的なダウンサンプリングはストライド 2 のデプスワイズ畳み込みで直接処理し（独立したプーリング層を省く）、モデルの末尾には NiN のグローバル平均プーリング（global average pooling）と単一の線形層を置く（7.2.2 節）。ImageNet における効果は劇的であった。MobileNet は VGG-16 と 1 パーセントポイント以内の差（top-1 正解率で 70.6% 対 71.5%）に迫りながら、わずか 5 億 7000 万回の積和演算と 420 万個のパラメータ、すなわち演算量で $27\times$ 減、パラメータ数で $33\times$ 減を達成したのである。

7.5.1.3 逆ボトルネック

古典的な ResNet や ResNeXt のボトルネック（7.4.5.1 節）では、空間フィルタの手前でチャネルを**圧縮**していた。すなわち、1×1 層が広い表現を狭い部分空間へと絞り込むことで高コストな密な 3×3 畳み込みの計算量を抑え、2 つ目の 1×1 層で再び元の幅へと拡張していた。しかし、3×3 層が**デプスワイズ**である場合、この論理は完全に逆転する。デプスワイズ畳み込みは広い

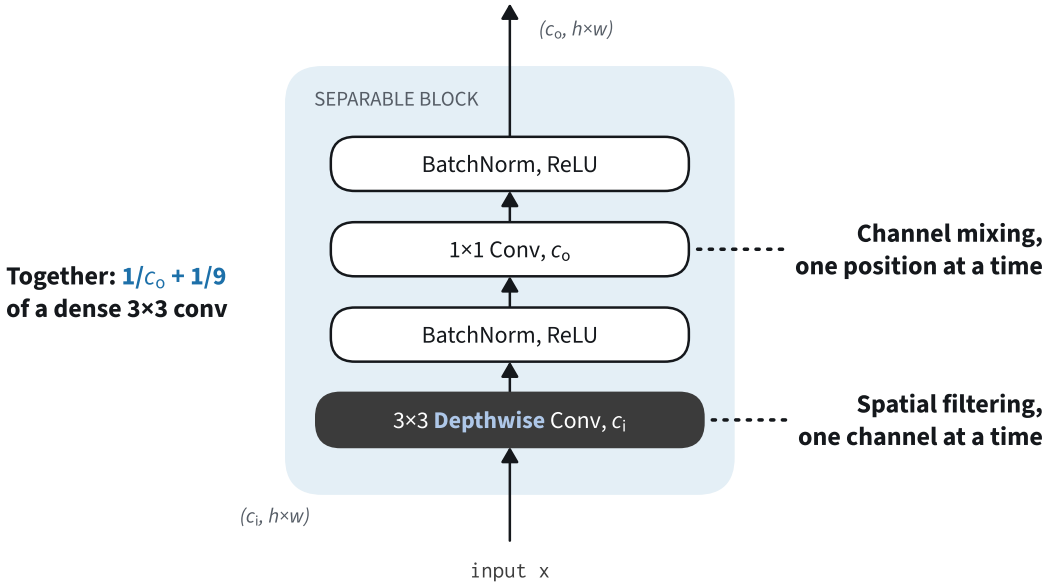


Figure 7.16: c_i チャネルから c_o チャネルへのデプスワイズ分離可能ブロック。デプスワイズ 3×3 畳み込みが各入力チャネルを個別にフィルタリングし、ポイントワイズ 1×1 畳み込みが各位置でチャネルを混合する。各畳み込みの後にはバッチ正規化と ReLU が続く。

特徴マップ上でも低コストであり、しかもチャネルごとに 1 つの空間フィルタしか学習しないため、むしろ広い幅で実行する方が有利になる。

そのため、**MobileNetV2** はボトルネックを**反転**させる (Sandler et al. 2018)。残差ハイウェイは狭いまま（たとえば 24 チャネル）に保ち、ブロック内部では 1×1 の拡張によって 6 倍の 144 チャネルへと引き上げ、デプスワイズ 3×3 畳み込みがその拡張された空間をフィルタリングした後、線形の 1×1 射影で再び 24 チャネルへと圧縮する。ここで決定的に重要なのは、最終の射影で ReLU を省く点である。きわめて細いボトルネックに ReLU を適用すると負の座標が 0 に潰され、後続の層では回復できない情報が永久に失われてしまう。Figure 7.17 に標準的なボトルネックと逆ボトルネックを並べて示す。

残差ショートカットが広い内部ではなく細い両端を結ぶため、ブロック間でメモリ上に保持されるのはコンパクトなテンソルだけで済む。これは、重み用のフラッシュストレージよりも活性化用の RAM の方が逼迫しやすいモバイルプロセッサにおいて大きな利点となる。この逆ボトルネック（inverted bottleneck）とまったく同じ構造には、より大きな 7×7 デプスワイズカーネルを先頭に移動させた形で、ConvNeXt（7.7 節）において再び出会うことになる。

7.5.1.4 Mini-MobileNet

デプスワイズ分離可能因子分解の効果を Fashion-MNIST 上で直接確かめるため、因子分解そのものに焦点を絞れるよう Figure 7.16 のシンプルなブロックを用いてコンパクトな MobileNetV1 風ネットワークを構築する。すべての畳み込みの直後にバッチ正規化が続くため、畳み込みのバイアスはすべて省略し、畳み込み層と線形層の重みにはすべて He 初期化（4.4 節）を用いる。

```
he_init = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
```

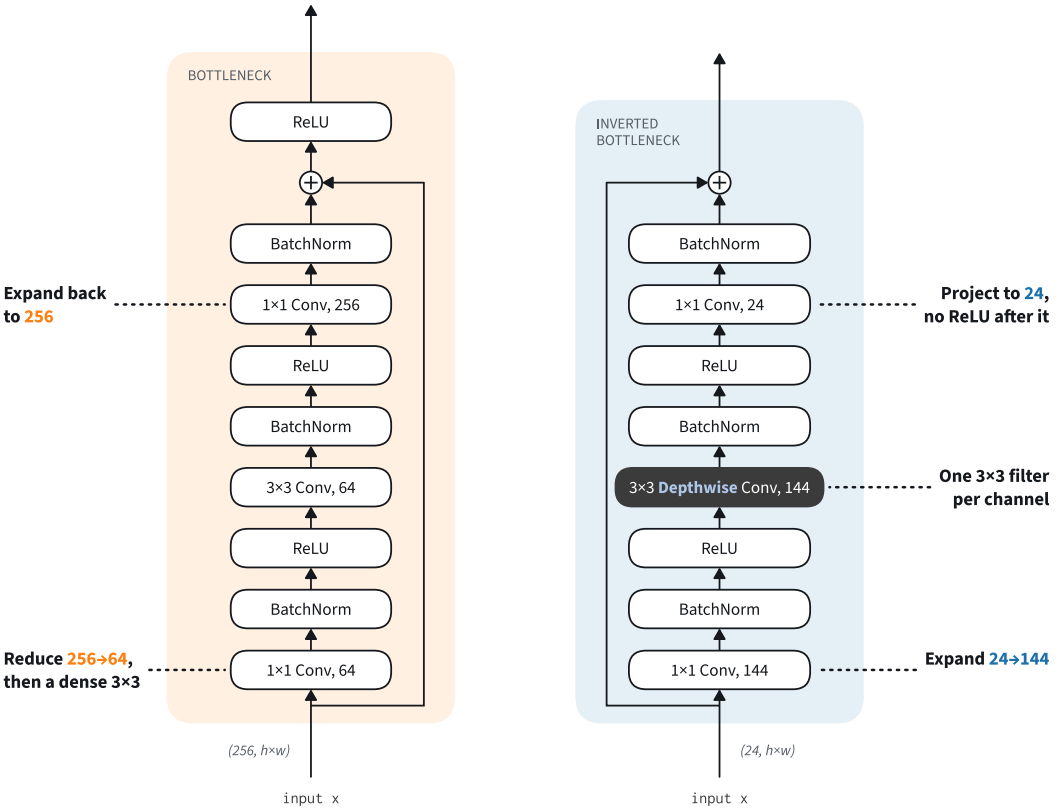


Figure 7.17: ResNet のボトルネックブロック（左）と MobileNetV2 の逆ボトルネック（右）。入力は下側にある。通常のボトルネックは密な 3×3 畳み込みのために 256 チャネルを 64 チャネルに狭めてから再び広げる。逆ボトルネックはチャネルごとに 1 つのフィルタを学習するデプスワイズ 3×3 畳み込みのために 24 チャネルを 144 チャネルに広げ、ReLU なしで 24 チャネルへ射影して戻す。残差接続は細い両端どうしを結ぶ。

```
def global_avg_pool(X):  # Average over height and width (NHWC)
    return X.mean(axis=(1, 2))

class DWSBlock(nnx.Module):
    """Depthwise 3x3 and pointwise 1x1 convolutions, each with BN and ReLU."""
    def __init__(self, in_channels, out_channels, strides=1, *, rngs):
        self.depthwise = nnx.Conv(
            in_channels, in_channels, (3, 3), strides=strides,
            padding=1, feature_group_count=in_channels, use_bias=False,
            kernel_init=he_init, rngs=rngs)
        self.bn1 = nnx.BatchNorm(in_channels, momentum=0.9,
                                 use_fast_variance=False, rngs=rngs)
        self.pointwise = nnx.Conv(in_channels, out_channels, (1, 1),
                                  use_bias=False, kernel_init=he_init,
                                  rngs=rngs)
        self.bn2 = nnx.BatchNorm(out_channels, momentum=0.9,
                                 use_fast_variance=False, rngs=rngs)

    def __call__(self, X):
        X = nnx.relu(self.bn1(self.depthwise(X)))
        return nnx.relu(self.bn2(self.pointwise(X)))
```

VGG（7.2.1 節）と同様に、アーキテクチャは `(out_channels, stride)` のペアのリストとして簡潔に指定する。ストライド 2 の 3×3 ステムが単一チャネルの入力を半分の解像度で 32 チャネルへと引き上げ、続く 7 個のデプスワイズ分離可能ブロックがチャネル数の倍増と空間解像度の半減を 3 回繰り返し、最後にグローバル平均プーリングと線形ヘッドが 10 クラスのロジットを出力する。

ここではオプティマイザもモメンタム 0.9 付きの SGD へと強化する。これは過去の勾配の指数移動平均を蓄積することで、同じ学習率でも実効的なステップサイズを通常の SGD の最大 10 倍にまで高められる手法である（9.4 節）。このオプティマイザの変更があるため、ここでの生の正解率を 7.4 節における通常の SGD を用いた ResNet-18 の実行結果と直接比較してはならない。これまでと同様、学習率は半コサイン減衰（7.1 節）に沿って `num_steps` 回の訓練ステップ（10 エポック分で 4,690 ミニバッチ）にわたり滑らかに 0 へと減衰させるが、ここではウォームアップなしで各フレームワーク組み込みのコサインスケジューラを使用する。

```
class MiniMobileNet(d2l.Classifier):
    def __init__(self, arch=((64, 1), (128, 2), (128, 1), (256, 2),
                             (256, 1), (512, 2), (512, 1)),
                 lr=0.05, num_steps=4690, num_classes=10, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        layers = [nnx.Conv(1, 32, (3, 3), strides=2, padding=1,
                           use_bias=False, kernel_init=he_init, rngs=rngs),
                  nnx.BatchNorm(32, momentum=0.9, use_fast_variance=False,
                                rngs=rngs),
                  nnx.relu]
        c = 32
        for c_out, strides in arch:
            layers.append(DWSBlock(c, c_out, strides, rngs=rngs))
            c = c_out
```

```
        layers += [global_avg_pool,
                   nnx.Linear(c, num_classes, kernel_init=he_init,
                              rngs=rngs)]
        self.net = nnx.Sequential(*layers)

    def configure_optimizers(self):
        schedule = optax.cosine_decay_schedule(self.lr, self.num_steps)
        return optax.sgd(schedule, momentum=0.9)
```

MiniMobileNet の訓練可能なパラメータ数を数え、対応する密なネットワークと比較してみよう。なお、後の節で再利用するため num_params を d2l に保存しておく。

```
def num_params(model):
    """The number of trainable parameters."""
    return sum(p.size for _, p in nnx.state(model, nnx.Param).flat_state())

model = MiniMobileNet()
assert model(jnp.zeros((1, 96, 96, 1))).shape == (1, 10)
num_params(model)
```

```
542474
```

MiniMobileNet は 542,474 個の訓練可能なパラメータを持つ。各分離可能ブロックを同じ幅どうしの通常の密な 3×3 畳み込みに置き換えると、7 個のブロックだけでも $9\sum_l c_{\mathrm{i},l}\,c_{\mathrm{o},l} = 4{,}663{,}296$ 個の重みが必要になる。これはこのモデルの分離可能層が持つ 530,528 個の重みの $8.8\times$ にあたり、チャネル幅が 512 に達するにつれて漸近的な上限である $9\times$ に近づいている。

7.5.1.5 演算量・正解率・実行時間

因子分解はどれだけの正解率を引き換えにし、実際にどれだけの実測速度をもたらすのだろうか。それを確かめるため、まず VGGSmall を構築する。これは密な VGG 風ネットワーク（7.2.1 節）であり、MiniMobileNet の**パラメータ数**に合わせたものである。密な 3×3 層はそれぞれ分離可能ブロックの約 9 倍の重みを使うため、パラメータ予算を揃えると、VGGSmall の上限は 128 チャネルとなるのに対し、MiniMobileNet は 512 チャネルに達する。両モデルはバッチ正規化、He 初期化、グローバル平均プーリングヘッド、およびまったく同じ訓練スケジュールを共有する。なお、パラメータ数を揃える比較では、層の種類とチャネル幅の両方が同時に変化する点に留意しておこう。

```
class VGGSmall(d2l.Classifier):
    """A dense VGG-style network with the same parameter budget."""
    def __init__(self, arch=((2, 32), (2, 64), (2, 128), (2, 128)),
                 lr=0.05, num_steps=4690, num_classes=10, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        layers, c = [], 1
        for num_convs, c_out in arch:
            for _ in range(num_convs):
                layers += [nnx.Conv(c, c_out, (3, 3), padding='same',
                                    use_bias=False, kernel_init=he_init,
                                    rngs=rngs),
                           nnx.BatchNorm(c_out, momentum=0.9,
```

```
                                        use_fast_variance=False, rngs=rngs),
                            nnx.relu]
                c = c_out
            layers.append(max_pool)
        layers += [global_avg_pool,
                   nnx.Linear(c, num_classes, kernel_init=he_init,
                              rngs=rngs)]
        self.net = nnx.Sequential(*layers)

    def configure_optimizers(self):
        schedule = optax.cosine_decay_schedule(self.lr, self.num_steps)
        return optax.sgd(schedule, momentum=0.9)

def max_pool(X):  # 2x2 windows, stride 2
    return nnx.max_pool(X, (2, 2), (2, 2))
```

訓練を開始する前に、96×96 の入力に対する正確な積和演算数を集計しておこう。「same」パディングとストライド s を用いるとき、$h \times h$ の入力は一辺 $\lceil h/s \rceil$ へと写像され、$k \times k$ のグループ畳み込み（グループ数 g）は出力要素あたり $k^2 c_\mathrm{i}/g$ 回の積和演算を要する。

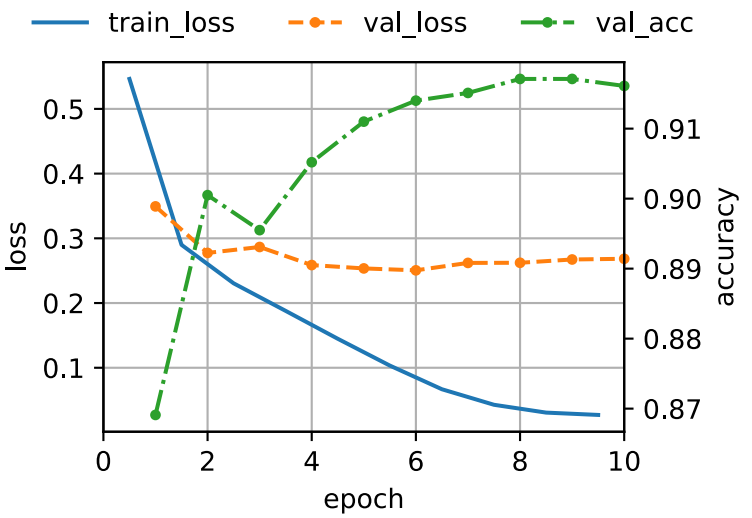
```
def conv_macs(h, c_in, c_out, k=3, stride=1, groups=1):
    """Output side and multiply-adds of a k x k convolution, 'same' padding."""
    h = -(-h // stride)  # ceil(h / stride)
    return h, h * h * k * k * c_in * c_out // groups

def mobilenet_macs(arch, h=96, num_classes=10):
    h, total = conv_macs(h, 1, 32, stride=2)  # The stem
    c = 32
    for c_out, stride in arch:
        h, depthwise = conv_macs(h, c, c, stride=stride, groups=c)
        h, pointwise = conv_macs(h, c, c_out, k=1)
        total, c = total + depthwise + pointwise, c_out
    return total + c * num_classes

def vgg_macs(arch, h=96, num_classes=10):
    total, c = 0, 1
    for num_convs, c_out in arch:
        for _ in range(num_convs):
            h, macs = conv_macs(h, c, c_out)
            total, c = total + macs, c_out
        h //= 2  # 2 x 2 max-pooling
    return total + c * num_classes

narrow = ((2, 11), (2, 23), (2, 46), (2, 46))  # VGGSmall's widths / 2.8
macs = {'Mini-MobileNet': mobilenet_macs(MiniMobileNet().arch),
        'VGG-style': vgg_macs(VGGSmall().arch),
        'VGG-style, narrow': vgg_macs(narrow)}
macs, macs['VGG-style'] / macs['Mini-MobileNet']
```

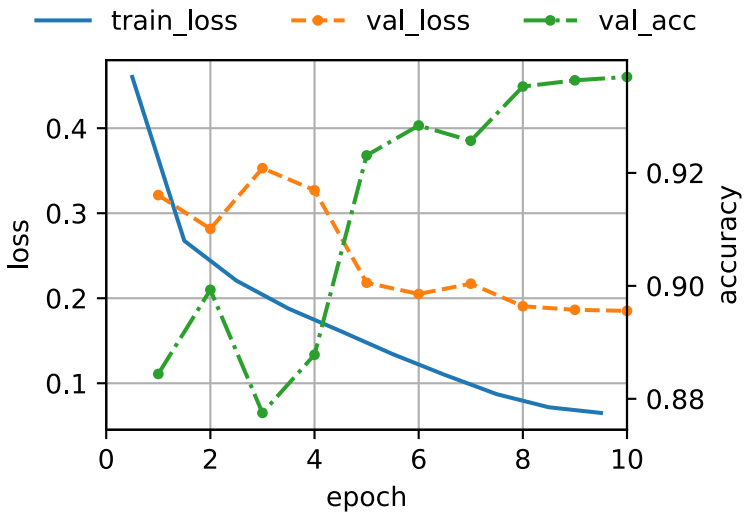
```
({'Mini-MobileNet': 50259968,
  'VGG-style': 384861440,
  'VGG-style, narrow': 49103308},
 7.657415142007253)
```

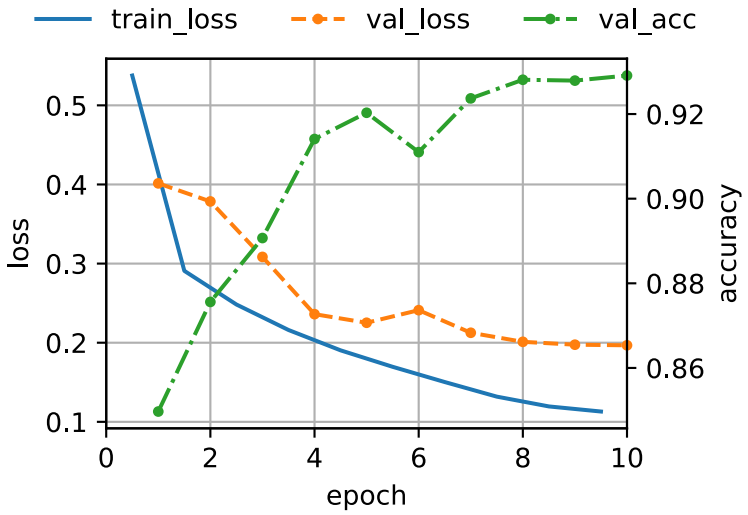
パラメータ数が揃っているにもかかわらず、画像 1 枚あたりの積和演算数は VGGSmall が 3 億 8500 万回を実行するのに対し、MiniMobileNet はわずか 5000 万回、すなわち 7.7× 少ない演算量で済む。この差は、各モデルが重みを**どこに**配置しているかを反映している。VGGSmall が 96×96 から 24×24 までの高解像度の特徴マップにわたって密なフィルタを適用するのに対し、MiniMobileNet の重みのほぼ 4 分の 3 は、小さな 6×6 のグリッド上で作用する最後の 2 つのポイントワイズ層に集中しているためである。

しかしエッジハードウェアにおいて決定的な意味を持つのは、**同一の演算量**での比較である。VGGSmall を MiniMobileNet と同じ 5000 万回の積和演算まで抑えるため、すべてのチャネル数を約 2.8 分の 1 に縮小して (11, 23, 46, 46)（上の narrow）とすると、パラメータ数はわずか 76,000 個、すなわち MiniMobileNet の重み数の 7 分の 1 になる。これら 3 つのモデルすべてを 10 エポックにわたり 96×96 の Fashion-MNIST 上で訓練してみよう。

```
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
num_steps = 10 * len(data.train_dataloader())  # 10 epochs of 469 steps
models = {'Mini-MobileNet': MiniMobileNet(lr=0.05, num_steps=num_steps),
          'VGG-style': VGGSmall(lr=0.05, num_steps=num_steps),
          'VGG-style, narrow': VGGSmall(narrow, lr=0.05, num_steps=num_steps)}
accuracy = {}
for name, model in models.items():
    trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
    trainer.fit(model, data)
    accuracy[name] = trainer.history['val_acc'][-1]
```







演算回数が測るのは算術演算の量であり、経過時間ではない。積和演算数が実際のレイテンシにどう結びつくかを調べるため、GPU 上で 128 枚の画像からなるバッチに対する推論の順伝播をベンチマークする。この際、1 回ウォームアップを実行し、実経過時間を読み取る前にデバイスを同期させる（5.7.2.1 節）。

```
def forward_ms(model, num_reps=50):
    """Milliseconds per forward pass of a batch of 128 images."""
    X = jax.random.normal(d2l.get_key(), (128, 96, 96, 1))
    graphdef, state = nnx.split(nnx.view(model, use_running_average=True))
    forward = jax.jit(lambda state, X: nnx.merge(graphdef, state)(X))
    forward(state, X).block_until_ready()  # Compile, warm up, and wait
    start = time.perf_counter()
    for _ in range(num_reps):
        Y = forward(state, X)
```

```
    Y.block_until_ready()
    return 1000 * (time.perf_counter() - start) / num_reps

for name, model in models.items():
    print(f'{name}: {num_params(model):,} parameters, '
          f'{macs[name] / 1e6:.1f}M multiply-adds, '
          f'accuracy {accuracy[name]:.3f}, '
          f'{forward_ms(model):.1f} ms/batch')
```

```
Mini-MobileNet: 542,474 parameters, 50.3M multiply-adds, accuracy 0.916, 2.9 ms/batch
VGG-style: 583,594 parameters, 384.9M multiply-adds, accuracy 0.937, 7.2 ms/batch
VGG-style, narrow: 75,854 parameters, 49.1M multiply-adds, accuracy 0.929, 1.8 ms/batch
```

Table 7.3 に、複数回の実行にわたる 3 つのモデルすべての結果をまとめる。検証正解率とあわせて、MiniMobileNet を基準に正規化した各モデルの順伝播レイテンシを示している。

Table 7.3: パラメータ数をほぼ揃えた密な VGG 風ネットワークおよび積和演算数をほぼ揃えた狭幅の VGG 風ネットワークと Mini-MobileNet の比較（96×96 の Fashion-MNIST、10 エポック。複数回の実行にわたる検証正解率と順伝播時間の比）。

モデル	パラメータ数	積和演算数	検証正解率	順伝播時間
Mini-MobileNet	542,474	5030 万	91〜92%	1
VGG 風	583,594	3 億 8490 万	93.5〜94%	2〜3.5
VGG 風（狭幅）	75,854	4910 万	93%	0.5〜0.8

Table 7.3 からは 3 つの教訓が浮かび上がる。

1. **パラメータ数を揃えても計算量は揃わない：**約 55 万パラメータにおいて、VGGSmall は MiniMobileNet を約 2 パーセントポイント上回るが、同時に 7.7× 多い積和演算を実行している（最初のステージで 96×96 の完全な解像度を保ち、制約のない密なカーネルを学習するためである）。
2. **小規模なデータセットで演算量を揃えた場合、因子分解より幅の縮小が勝る：**VGGSmall を 5000 万回の積和演算まで縮小したモデル（VGG-style, narrow、約 7.6 万パラメータ）は、MiniMobileNet を約 1 ポイント上回り、**しかも**高速に動作する。Equation 7.6 によれば、デプスワイズ因子分解が最も効果を発揮するのはチャネル幅が広い領域である（削減倍率は $5\times$ にとどまる $c = 11$ に対し、ほぼ $9\times$ に達する $c = 512$ で大きく開く）。10 クラスの Fashion-MNIST は幅の狭い密な層でも十分に表現できるのに対し、1,000 クラスの ImageNet では広いステージが不可欠であり、密な畳み込みネットワークの幅を狭めると表現容量が不足してしまう。
3. **積和演算数は GPU の実測レイテンシの指標としては不十分である：**演算量を 7.7× 削減しても速度向上が 2–3.5× にとどまるのは、密な 3×3 層が入力の読み出し 1 回あたり $9c_o$ 回の積和演算を実行する（演算密度（arithmetic intensity）が高い）のに対し、デプスワイズ 3×3 層は入力の読み出し 1 回あたりわずか 9 回の積和演算しか実行せず、GPU がメモリ帯域幅律速（memory-bandwidth bound）となるからである。

7.5.2 スケーリングと探索

MobileNet は効率的な基本ブロックを固定し、各ステージの幅を人手で選択している。ここから自然な疑問が 2 つ生じる。利用可能な計算資源が増えたとき、ベースとなるアーキテクチャをどのようにスケールアップすべきだろうか。また、自動化された最適化によって、人間の試行錯誤よりも優れたステージの幅・深さ・解像度を選択できるのだろうか。

7.5.2.1 複合スケーリング

計算量を数えてみれば、スケーリングの問題はただちに明確になる。畳み込みネットワークの演算量は、深さに対して線形に、チャネル幅に対して 2 次（$c_\mathrm{i}c_\mathrm{o}$ 経由）に、入力解像度に対しても 2 次（hw 経由）に増加する。どれか 1 つの次元だけを伸ばしても、すぐに収穫逓減に突き当たる。極小のサムネイル画像に対して非常に深いモデルを適用しても分析すべき細部がそもそも欠けており、逆に高解像度の画像では受容野が大きな物体全体を覆えるようにより多くの層が必要になるからである。**複合スケーリング**（compound scaling）は、これら 3 つの次元すべてを歩調を揃えて拡大する (Tan and Le 2019)。各スケーリングステップで深さを 1.2 倍、幅を 1.1 倍、解像度を 1.15 倍にすると、総計算量はおよそ 2 倍になる（$1.2 \cdot 1.1^2 \cdot 1.15^2 \approx 1.92$）。これにより、単一の整数のつまみだけでモデル族全体を生成できるようになる。たとえば 3 ステップ進めると、深さが $1.7\times$、幅が $1.3\times$ になり、$1.5\times$ 大きい画像を入力として計算量が約 $7\times$ となるネットワークが得られる。

7.5.2.2 アーキテクチャ探索

ベースとなるネットワーク自体の設計には、EfficientNet は逆ボトルネックブロックの積層を対象とした**ニューラルアーキテクチャ探索**（neural architecture search、NAS）を用い、正解率と積和演算数のバランスをとる目的関数を最適化した。しかし、先ほどの GPU 実行時間ベンチマークが示したとおり、理論上の積和演算数を最適化すると実ハードウェアでは誤った結論を招きかねない。**EfficientNetV2** は探索の目的関数において積和演算数を実測の訓練速度とパラメータ数に置き換えた (Tan and Le 2021)。注目すべきことに、探索の結果、初期の高解像度ステージにあるデプスワイズブロックは、演算量が多くともアクセラレータ上で高速に動作する標準的な密な 3×3 畳み込みへと自動的に置き換えられた。いずれの世代でも、中核となる基本演算は人間が設計し、各ステージへの幅・深さ・解像度の配分を自動探索が担っている。

7.5.2.3 Squeeze-and-Excitation ゲート

EfficientNet が探索した逆ボトルネックブロックには、デプスワイズ畳み込みの直後にもう 1 つ軽量なモジュールとして **Squeeze-and-Excitation**（SE）ゲートが含まれている (Hu et al. 2018)。畳み込みが局所的な空間ウィンドウ内でのみ情報を集約するのに対し、SE ゲートは大域的な画像コンテキストを用いて特徴チャネル全体の重みを動的に付け直すことができる。具体的には、グローバル平均プーリングによって c 個の各チャネルを単一のスカラーへと**圧縮**（squeeze）し、その c 次元の要約を小さな 2 層のボトルネック MLP（ReLU を伴う隠れ層の幅 c/r、シグモイドを伴う出力幅 c による**励起**（excitation））に通して、特徴マップの各チャネルを $(0, 1)$ 内の予測ゲート値でスケーリングする。縮小比を r とすると、この MLP が追加するパラメータは約 $2c^2/r$ 個にとどまり、全ピクセルではなく画像あたり単一のベクトルに対して作用するため、演算コストは無視できるほど小さい。概念的には、SE はチャネルごとの単純なアテンション機構として機能しており、第 10 章で扱う一般的なアテンション層の先駆けとなっている。

SE ゲートはほぼゼロの計算コストで一貫した正解率の向上をもたらすため、MobileNetV3 や RegNetY（7.8 節、その演習問題 3 では RegNet に SE を追加する）の随所で採用されている。SE ブロックの実装はほんの数行で済み、グローバルプーリング、2 つの線形層、およびチャネルごとの再スケーリングからなる。

```
class SE(nnx.Module):
    def __init__(self, num_channels, ratio=4, rngs=None):
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.squeeze = nnx.Linear(num_channels, num_channels // ratio,
                                  kernel_init=he_init, rngs=rngs)
        self.excite = nnx.Linear(num_channels // ratio, num_channels,
```

```
                                kernel_init=he_init, rngs=rngs)

    def __call__(self, X):
        s = X.mean(axis=(1, 2))
        s = nnx.relu(self.squeeze(s))
        s = nnx.sigmoid(self.excite(s))
        return X * s[:, None, None, :]

X = jax.random.normal(d2l.get_key(), (2, 16, 16, 32))
SE(32)(X).shape
```

```
(2, 16, 16, 32)
```

$c = 32$、$r = 4$ のとき、SE モジュールが追加するのはわずか $2 \cdot 32^2/4 = 512$ 個の重みと 40 個のバイアスだけであり、テンソルの形状を完全に保つため、任意のブロックにそのまま組み込むことができる。

7.5.3 構造的再パラメータ化

7.5.3.1 RepVGG ブロック

ResNet（7.4 節）は、ショートカット経路を持つマルチブランチブロックが、単純な直列積層よりも劇的に最適化しやすいことを確立した。しかし推論時には、並列ブランチはシステム上の実質的なオーバーヘッドを伴う。各ブランチを個別に起動し、加算地点までメモリ上に保持し続けなければならず、ハードウェアの利用効率が低下するためである。対照的に、3×3 畳み込みの単純な直列積層はアクセラレータにとって理想的であり、層ごとに高度に最適化された密なカーネル 1 つと、保持すべき単一の活性化バッファだけで済む。**RepVGG** は、訓練時のトポロジーと推論時のトポロジーを切り離すことで両者の長所を兼ね備える (X. Ding et al. 2021)。訓練時には、すべての層が ReLU の手前で 3 つの並列ブランチ、すなわち 3×3 畳み込み、1×1 畳み込み、恒等経路（それぞれ独自のバッチ正規化を伴う）の和をとる（Figure 7.18）。訓練が完了すると、各ブランチは入力に関して純粋に線形となるため、3 つすべてを代数的に**単一の 3×3 畳み込みへと融合**し、まったく同じ関数を計算できる。つまり、訓練時にはマルチブランチの残差ネットワークを用い、実運用時には高速な単純直列の VGG スタックを配備するのである。

```
class RepVGGBlock(nnx.Module):
    """Train-time block: 3x3, 1x1, and identity branches, each with BN."""
    def __init__(self, num_channels, rngs=None):
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.conv3 = nnx.Conv(num_channels, num_channels, (3, 3),
                              padding='same', use_bias=False, rngs=rngs)
        self.conv1 = nnx.Conv(num_channels, num_channels, (1, 1),
                              use_bias=False, rngs=rngs)
        self.bn3, self.bn1, self.bn0 = (
            nnx.BatchNorm(num_channels, momentum=0.9,
                          use_fast_variance=False, rngs=rngs)
            for _ in range(3))

    def __call__(self, X):
        return nnx.relu(self.bn3(self.conv3(X)) + self.bn1(self.conv1(X))
                        + self.bn0(X))
```

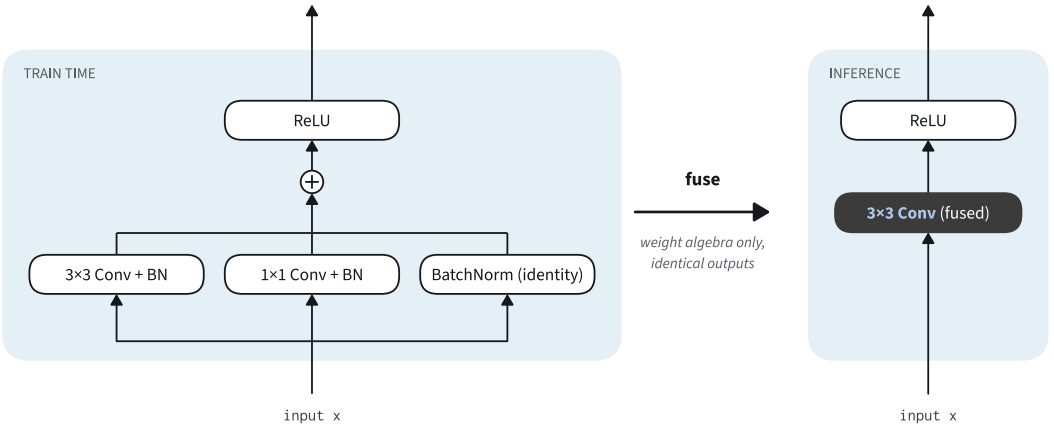


Figure 7.18: 構造的再パラメータ化。訓練時には、RepVGG の各層はそれぞれバッチ正規化を伴う 3×3 畳み込み、1×1 畳み込み、恒等写像の 3 つのブランチの和をとる。推論時には、これらのブランチはバイアス付きの 1 つの 3×3 畳み込みへと折り込まれ、2 つのネットワークは同一の関数を計算する。

7.5.3.2 ブランチの融合

3 つのブランチすべてを 1 つの畳み込みに集約するには、線形代数の単純な 3 つのステップを踏めばよい。第 1 に、推論モードにおけるバッチ正規化は、移動平均 μ_o を引き、$\sqrt{\sigma_o^2 + \epsilon}$ で割り、γ_o を掛けて β_o を足すという、チャネルごとの固定されたアフィン写像である。このアフィン写像を直前の畳み込みのカーネル $\mathbf{W}$ とバイアス b_o（出力チャネル o で添字付けされる）に直接吸収させると、等価な融合後のカーネルとバイアスが次の式で得られる。

$$\begin{aligned} \hat{\mathbf{W}}_o &= \frac{\gamma_o}{\sqrt{\sigma_o^2 + \epsilon}} \mathbf{W}_o, \\ \hat{b}_o &= \beta_o + \frac{\gamma_o (b_o - \mu_o)}{\sqrt{\sigma_o^2 + \epsilon}}. \end{aligned} \tag{7.7}$$

各出力チャネルのカーネルは正規化係数によってスケーリングされるだけであり、融合後のバイアスが入力に依存しないオフセットをすべて吸収する（たとえば、$\gamma_o = 2$、$\sigma_o^2 + \epsilon = 16$、$\mu_o = 1$、$\beta_o = 1$、$b_o = 0$ のとき、$\hat{\mathbf{W}}_o = \mathbf{W}_o / 2$ および $\hat{b}_o = 1 - 0.5 = 0.5$ となる）。`RepVGGBlock` の畳み込みはバイアスを持たないため、以下の `fuse_bn` は $b_o = 0$ の場合を実装している。なお、BN の折り込み（BN folding）は完全に一般的な手法であり、推論コンパイラによって任意の `Conv + BN` ペアに日常的に適用されている。

第 2 に、残りの 2 つのブランチを 3×3 畳み込みとして表す。1×1 カーネルは中心の周囲を 1 ピクセル分ゼロパディングした 3×3 カーネルにほかならず、恒等写像は 3×3 畳み込みとして、カーネル $\mathbf{W}^{\text{id}}_{o,i}$ が $i = o$ の中心タップで 1、それ以外のすべての位置で 0 となるものに一致する。第 3 に、畳み込みの線形性により、同一の入力に対する 3 つの並列な畳み込みの出力を足し合わせることは、折り込み後の 3 つのカーネルの和をカーネルとし、折り込み後の 3 つのバイアスの和をバイアスとする単一の畳み込みと完全に一致する（Figure 7.19）。

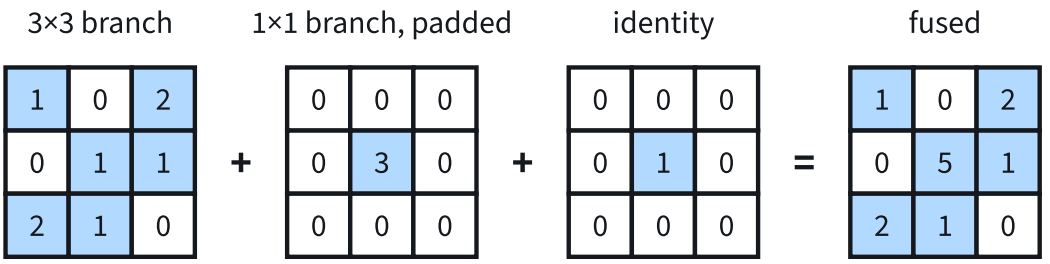


Figure 7.19: バッチ正規化を各ブランチに折り込んだ後の、1 つの出力チャネル o が入力チャネル $i = o$ に適用するカーネル（数値は例示用。折り込み後、恒等ブランチの中心成分は 1 ではなく $\gamma_o/\sqrt{\sigma_o^2 + \epsilon}$ になる）。パディングされた 1×1 カーネルと恒等カーネルは中心のみで非ゼロとなるため、融合後のカーネルは中心のみで 3×3 ブランチと異なる。$i \neq o$ では恒等カーネルは 0 である。

```
def fuse_bn(W, bn):
    """Fold inference-mode batch normalization into a bias-free conv."""
    scale = bn.scale[...] / jnp.sqrt(bn.var[...] + bn.epsilon)
    return W * scale, bn.bias[...] - bn.mean[...] * scale

def fuse_block(blk):
    c = blk.conv3.kernel.shape[-1]
    pad = ((1, 1), (1, 1), (0, 0), (0, 0))
    W3, b3 = fuse_bn(blk.conv3.kernel[...], blk.bn3)
    W1, b1 = fuse_bn(jnp.pad(blk.conv1.kernel[...], pad), blk.bn1)
    I = jnp.pad(jnp.eye(c).reshape(1, 1, c, c), pad)
    W0, b0 = fuse_bn(I, blk.bn0)
    fused = nnx.Conv(c, c, (3, 3), padding='same',
                     rngs=nnx.Rngs(d2l.get_key()))
    fused.kernel[...], fused.bias[...] = W3 + W1 + W0, b3 + b1 + b0
    return fused
```

融合は推論モードの移動統計量に依存するため、fuse_block を厳密にテストする。まず、いくつかの訓練バッチ (平均 1、標準偏差 2) を RepVGGBlock に通して移動統計量を 0 と 1 から変化させ、γ_o と β_o をランダム化して Equation 7.7 のすべての項が働くようにしたうえで、新たなデータに対して推論モードのマルチブランチブロックと単一の融合済み畳み込みの最大絶対誤差を比較する。

```
blk = RepVGGBlock(8)
for _ in range(4):  # Move the BN running statistics
    nnx.view(blk, use_running_average=False)(
        1 + 2 * jax.random.normal(d2l.get_key(), (32, 16, 16, 8)))
for bn in (blk.bn3, blk.bn1, blk.bn0):  # Nontrivial scales and shifts
    bn.scale[...] = jax.random.uniform(d2l.get_key(), (8,), minval=0.5,
                                       maxval=1.5)
    bn.bias[...] = jax.random.normal(d2l.get_key(), (8,))
fused = fuse_block(blk)
X = 1 + 2 * jax.random.normal(d2l.get_key(), (32, 16, 16, 8))
Y = nnx.view(blk, use_running_average=True)(X)
float(jnp.abs(Y - nnx.relu(fused(X))).max())
```

```
5.7220458984375e-06
```

最大誤差は 10^{-6} のオーダー（純粋な単精度浮動小数点の丸め誤差）であり、関数として厳密に等価であることが確認できる。重み数の削減は 10 分の 1（$10c^2$ から $9c^2$）にとどまるものの、実行時の処理は劇的に単純化される。2 つの畳み込み、3 つの正規化、2 つの加算、および同時に保持される 3 つの特徴バッファが、1 つの密な畳み込みと 1 つの保持される活性化テンソルへと置き換わるからである。

7.5.3.3 融合後の量子化

実環境への配備では、厳密な浮動小数点の代数だけでは見落とされる繊細な注意点が 1 つ生じる。消費エネルギーとメモリ帯域幅を節約するため、エッジアクセラレータでは重みと活性化値を 8 ビット整数へと量子化（quantization）し、各テンソルのダイナミックレンジを等間隔なわずか 256 段階へと写像することが広く行われている。RepVGG の融合カーネルは独立にバッチ正規化された 3 つのブランチの和をとり、折り込み後の各スケールが何桁も異なりうるため、融合後の重みや活性化値は裾の重いダイナミックレンジを持ちやすく、256 個の離散ビンではうまく表現できなくなる。その影響は壊滅的になりうる。単純な訓練後の 8 ビット量子化を適用すると、融合済み RepVGG-B0 の ImageNet top-1 正解率は 75.1% から 40.2% へと急落するのに対し、標準的な ResNet の低下は 1 ポイント未満にとどまる (Chu et al. 2024)。訓練中に各ブランチのスケールを正則化すればこの問題は解消されるが、ここには重要な工学的教訓がある。すなわち、FP32 における厳密な代数的等価性は、低精度量子化のもとでの等価性を保証しないのである。

7.5.4 まとめ

効率的な畳み込みネットワークでは、正解率と並んでレイテンシ、メモリ、消費エネルギーが最優先の設計目標へと引き上げられる。デプスワイズ分離可能畳み込みは空間フィルタリングとチャネル混合を分離することで層のコストを密な畳み込みの $1/c_{\mathrm{o}} + 1/k^2$ に削減し、MobileNetV2 の逆ボトルネックは残差ハイウェイを細く保ってピーク時の活性化メモリを最小化する。Fashion-MNIST 上では、`MiniMobileNet` はパラメータ数を揃えた密なネットワークに対して積和演算数が 7.7× 少ない一方で正解率は約 2 ポイント劣り、演算密度の低いデプスワイズ層がメモリ帯域幅のボトルネックとなるため実行速度の向上も 2–3.5× にとどまる。さらにこの小規模なタスクでは、密なネットワークの幅を同じ演算予算まで狭めた方が、正解率と速度の両面で `MiniMobileNet` を上回る。層の因子分解にとどまらず、複合スケーリングとレイテンシ考慮型アーキテクチャ探索は各ステージの深さ・幅・解像度を配分し、Squeeze-and-Excitation ゲートは無視できるほどの演算コストで大域的なチャネル重み付けを追加し、構造的再パラメータ化は訓練時のマルチブランチブロックを推論時に高速な単純畳み込みへと折り込む（ただし融合後の量子化には注意を要する）。人手による設計と単一モデルの探索による効率化を見てきたところで、次節以降では訓練レシピ、ResNet の体系的な現代化、および集団レベルのアーキテクチャ設計空間へと進む。

7.5.5 演習問題

1. **グループ間のチャネル置換（ShuffleNet）**。●グループ畳み込みは、密な 1×1 重みを用いずとも混合できる (X. Zhang et al. 2018)。$k \times k$ のグループ畳み込み（$c_{\mathrm{i}} \to c_{\mathrm{o}}$ チャネル、g グループ）の後に、c_{o} 個の中間チャネルに対する固定の置換と、2 層目のグループ畳み込み（$c_{\mathrm{o}} \to c_{\mathrm{o}}$、$g$ グループ）を続けるとする。g と c_{o} について、2 層目のすべての出力チャネルがある置換のもとで全 c_{i} 個の入力チャネルに依存できるようになる条件を求め、その置換を具体的に構成せよ。

2. [code] **幅乗数**。すべてのステージのチャネル数を m 倍にスケールする幅乗数 m を `MiniMobileNet` に追加せよ (Howard et al. 2017)。パラメータ数と積和演算数が m に対し

てどのようにスケールするかを予測し、$m = 0.5$ において `num_params` と `mobilenet_macs` を用いてその予測を検証せよ。

3. [code] **バイアス付きバッチ正規化の折り込み**。@eq-bn-fold を用いて `fuse_bn` を非ゼロのバイアス b_o を持つ直前の畳み込みへと拡張し、推論モードの `Conv + BN` ペアが折り込み後の単一の畳み込みと 10^{-6} 以内で数値的に一致することを検証せよ。バッチ正規化が**訓練**モードで動作しているとき、いかなる固定のカーネルとバイアスによっても `Conv + BN` の出力を再現できないのはなぜか。

4. [code] **MobileNetV2 の逆ボトルネック**。●MobileNetV2 (Sandler et al. 2018) は Figure 7.16 のブロックを逆ボトルネック、すなわち 1×1 による倍率 $t = 6$ の拡張、デプスワイズ 3×3 畳み込み、**線形**の 1×1 射影（ReLU なし）、および入出力の形状が一致する場合の残差ショートカットに置き換えている。

   1. $t = 6$ の逆ボトルネックブロックを実装して `MiniMobileNet` の各ブロックを置き換え、総パラメータ数が 542,474 付近に保たれるよう各ステージの幅を選べ。Table 7.3 の設定のもとで訓練し、検証正解率と順伝播レイテンシを `MiniMobileNet` と比較せよ。
   2. 最終の 1×1 射影の後に ReLU を追加し、線形版と ReLU 版の両方をそれぞれ 3 回ずつ訓練して、この規模でも線形射影が ReLU 付き射影を上回るかどうかを確認せよ。
   3. 96×96 の入力に対し、各ブロックの入力テンソル・出力テンソル・拡張された内部テンソルの要素数を表にまとめよ。拡張とデプスワイズフィルタリングをチャネルごとに融合またはタイリングする推論エンジンでは、拡張されたテンソル全体を RAM 上に実体化する必要が一切ない理由を説明せよ。

5. [code] **演算密度とバッチサイズ**。●`MiniMobileNet` の積和演算数は $7.7\times$ も `VGGSmall` より少ないにもかかわらず、Table 7.3 における GPU 順伝播は 2–$3.5\times$ しか高速にならない。

   1. ある層の演算密度を、その積和演算数を読み書きするテンソル要素の総数（重み + 入力活性化値 + 出力活性化値）で割った値として定義せよ。デプスワイズ畳み込み、ポイントワイズ畳み込み、および等価な密な 3×3 畳み込みの演算密度を `MiniMobileNet` のすべてのステージについて計算せよ。デプスワイズ畳み込みの演算密度が最も低くなるのはどのステージか。また、そのことから EfficientNetV2 (Tan and Le 2021) が初期のデプスワイズステージを密な畳み込みに置き換える理由をどのように説明できるか。
   2. `forward_ms` を `MiniMobileNet` と `VGGSmall` についてバッチサイズ $B \in \{1, 16, 128, 512\}$ にわたり計測し、画像 1 枚あたりのレイテンシを B に対してプロットせよ。両者の速度比は B とともにどのように変化するか、またそれはなぜか。

6. [code] **RepVGG ブロックの訓練と量子化**。●推論時に単一の 3×3 畳み込みへと融合されるにもかかわらず、なぜ 3 ブランチの `RepVGGBlock` を訓練するのだろうか。

   1. チャネル幅を変更するストライド 2 のダウンサンプリングブロックに対応できるよう `RepVGGBlock` と `fuse_block` を拡張せよ（入出力の形状が異なる場合は 3×3 ブランチと 1×1 ブランチのみを残す (X. Ding et al. 2021)）。`RepVGGBlock` から小規模な畳み込みネットワークを構築するとともに、訓練中も単一の 3×3 ブランチのみを残す対応する単純な畳み込みネットワークを構築せよ。両者を Fashion-MNIST 上で訓練して収束を比較し、訓練済みの RepVGG ネットワークを融合しても検証正解率が変化しないことを確認せよ。
   2. 単純なネットワークのバッチ正規化を `fuse_bn` で折り込んだうえで、配備された両ネットワークの重みをテンソルごとの対称スケーリング（$\hat{w} = s \cdot \mathrm{round}(w/s)$、ただし $s = \max|w|/127$）を用いて 8 ビット整数へと量子化せよ。融合済み RepVGG ネットワークの正解率低下を単純なネットワークと比較し、Chu et al. (2024) の重み分布解析と関連づけて論じよ。

7.6 畳み込みネットワークの現代的な訓練レシピ

本章全体を通して、正解率の差をアーキテクチャの違いに帰属できるよう、各節の中で訓練手順を一定に保つよう努めてきた。そのような統制された条件下でさえ、ピーク学習率を変更しただけで VGG の正解率は 1.5 パーセントポイント変化した（7.2 節）。公開されているベンチマークは、これよりはるかに統制されていないことが多い。新たに提案されたネットワークは最新の最適化や正則化の工夫を用いて訓練され、何年も前に書かれた論文から正解率を転記した古典的なベースラインと比較されるのである。報告される飛躍的な性能向上のうち、アーキテクチャそのものではなく**訓練レシピ**（training recipe：オプティマイザ、学習率スケジュール、データ拡張、正則化、訓練期間）に由来する部分は、実際にはどれほどあるのだろうか。

7.4 節で構築した ResNet-18 をさらに深くした兄弟モデルである ResNet-50 が、驚くべき答えを与えてくれる。2015 年のアーキテクチャの層を 1 つも変更することなく、Wightman et al. (2021) は 2020 年代初頭の訓練手法を用いて ResNet-50 を再訓練し、ImageNet の top-1 正解率を 76.1% から 80.4% へと引き上げた。純粋に訓練レシピだけで 4.3 ポイントもの向上を達成したのである。この伸びは、その間に発表された数多くの新規アーキテクチャが主張する改善幅に匹敵するか、それを上回る。したがって、時代遅れのレシピで訓練されたベースラインに対して 2 ポイントの優位を主張する論文があったとしても、その功績がアーキテクチャにあるのかレシピにあるのかは判別できない。

本節では、2015 年から 2022 年の間に何が変わったのかを調べ、中核となる各構成要素を数行のコードで実装したうえで、同一の ResNet-18 を古典的なレシピと現代的なレシピの両方で訓練してその違いを直接観察する。これらの手法はいずれも畳み込みに固有のものではない。まったく同じツール群が現代の Transformer の訓練を支えており、現代の深層学習の実践における汎用的な基盤となっている。

7.6.1 2015 年から 2022 年にかけての訓練レシピ

2015 年のオリジナルの ResNet レシピは、その時点ですでに入念に調整されていた。モメンタム付き SGD（7.5 節）、重み減衰、段階的な学習率減衰（学習率を 2 回にわたり 10 分の 1 にする）、そしてランダムクロップと水平反転（horizontal flip）である (He et al. 2016a)。その後の数年間にわたり、研究者たちはこのパイプラインのあらゆる要素を、まずは個別に、続いて大規模かつ体系的な組み合わせによって見直していった (He et al. 2019; Wightman et al. 2021)。Table 7.4 に、2015 年のベースラインと 2020 年代初頭の代表的なレシピの対比を示す。直後の節では、それぞれの新しい構成要素を紐解きながら実装していく。

Table 7.4: 2015 年の ImageNet レシピと、2020 年代初頭の代表的な高正解率レシピ。右の列は万能の標準規格ではなく、強力なレシピは依然としてモデルや計算予算によって異なる。

構成要素	2015 年のレシピ	2022 年のレシピ
オプティマイザ	モメンタム付き SGD	AdamW または LAMB（調整された SGD も依然として有力）
スケジュール	ステップ減衰（10 分の 1 を 2 回）	線形ウォームアップ付きコサイン減衰
データ拡張	ランダムクロップ、水平反転	RandAugment、Mixup、CutMix、ランダム消去
正則化	重み減衰	+ ラベル平滑化、確率的深さ
訓練期間	90 エポック	300〜600 エポック
評価時の重み	最終時点の重み	重みの指数移動平均

Table 7.4 を読む際には、2 つの重要な点がある。第 1 に、表の各行は独立に働くのではなく密接

に相互作用する。データ拡張と正則化を強めると訓練目的関数への当てはめが難しくなり、その分だけはるかに多くの訓練エポックが必要になる。訓練スケジュールが 90 エポックから 300〜600 エポックへと伸びた理由はここにある。正則化強度と訓練期間は合わせて拡大しなければならない。短い訓練に強力なデータ拡張を詰め込むと、軽めのレシピに比べて正解率をかえって**損なう**ことが少なくない。

第 2 に、オプティマイザの行は万能の必須要件ではなく、選択肢の広がりを反映している。Adam のような適応的オプティマイザ（詳細は第 9 章で扱う）は、各パラメータの勾配をその大きさの移動推定値で正規化し、すべての座標がおよそ学習率のスケールで更新されるようにする。従来の実装ではこの再スケーリングの前に重み減衰を勾配に組み込んでいたため、意図せずペナルティが歪められていた。**AdamW** は重み減衰を適応的な勾配ステップから切り離し（分離型重み減衰、decoupled weight decay）、すべての重みを「学習率 × 減衰係数」に等しい割合で直接縮小する (Loshchilov and Hutter 2019)。この明快な修正により、画像系の各種ベンチマークにおいて適応的オプティマイザは入念に調整された SGD と互角に戦えるようになった。（LAMB は大バッチ向けの AdamW の派生手法である。前述の正解率 80.4% の ResNet-50 は、LAMB を用い、二値交差エントロピー（binary cross-entropy）の損失のもと重みの平均化なしで 600 エポック訓練されたものである。本節では LAMB は使用しない。）

7.6.2 各構成要素の実装

現代的な各構成要素は、ほんの数行のコードで記述できる。以下では、ラベル平滑化、ウォームアップ付きコサインスケジュール、Mixup、確率的深さ、そして指数移動平均による重みの平均化を実装する。（RandAugment は標準的な画像変換のランダムな系列を調整された強度で適用する手法であるが (Cubuk et al. 2020)、実務上は前処理ライブラリの呼び出しとして利用され、その真価は実装よりも強度調整にあるため、ここでは省略する。）

7.6.2.1 ラベル平滑化

ワンホットな正解ターゲットを用いた標準的な交差エントロピーのもとでは、正しく分類された例に対する損失は、正解クラスのロジットが $+\infty$ へ向かって大きくなるにつれて際限なく減少し続ける。有限のロジットでは目的関数が最小化されないため、ネットワークは較正（calibration）の不十分な極端な確信度へと絶えず押しやられる。**ラベル平滑化**（label smoothing）は、重み $1-\epsilon$ を正解クラス y に割り当て、残りの ϵ を全 K クラスに一様に分配したソフトな分布へとハードなワンホットターゲットを置き換えることで、この過信を抑止する (Szegedy et al. 2016)。予測されたソフトマックス確率を p_k と書くと、1 例あたりの損失は次のようになる。

$$l_\epsilon = -(1-\epsilon)\log p_y - \frac{\epsilon}{K}\sum_{k=1}^{K}\log p_k. \tag{7.8}$$

第 1 項は通常の交差エントロピーを $1-\epsilon$ 倍したものであり、第 2 項はいずれかのクラス確率を 0 に近づけることに対してペナルティを課す。この損失は $p_y = 1-\epsilon+\epsilon/K$ かつ $p_k = \epsilon/K$（$k \neq y$）のときに最小値をとり、これはロジット間の明確な有限のマージンに対応する。すでに高い確信度で正解している予測に対し、平滑化なしの交差エントロピーと平滑化ありの交差エントロピーのもとで何が起きるか見てみよう。

```
logits, Y = jnp.array([[10.0, 0.0, 0.0]]), jax.nn.one_hot(jnp.array([0]), 3)
for epsilon in (0.0, 0.1):
    loss = optax.softmax_cross_entropy(logits,
                                       optax.smooth_labels(Y, epsilon))
```

```
    print(f'epsilon={epsilon}: loss={float(loss[0]):.4f}')
```

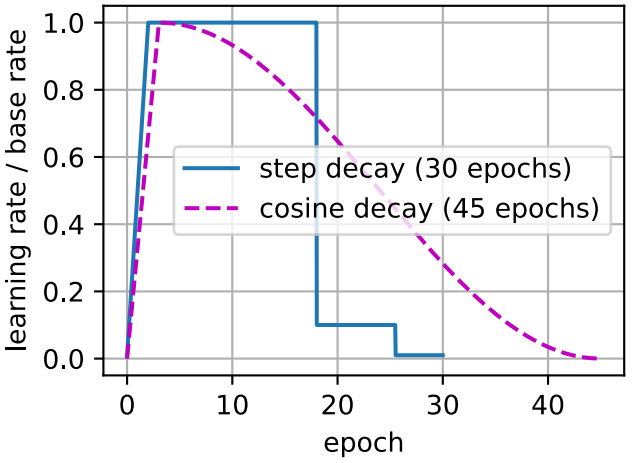
```
epsilon=0.0: loss=0.0001
epsilon=0.1: loss=0.6668
```

$\epsilon = 0.1$ とすると、損失は 0.0001 から 0.67 へと跳ね上がる。3 クラスの場合、平滑化された損失の最小値は約 0.29 であり、最大ロジットが他のロジットを約 3.3 上回るときに達成される。したがってマージン 10 は過信としてペナルティを受け、勾配の向きが反転して、ロジットをさらに引き離すのではなく適度な差へと引き戻すように働く。

7.6.2.2 ウォームアップ付きコサインスケジュール

従来のステップスケジュールは、数十エポックにわたって学習率を一定に保ったのち突如として 10 分の 1 に引き下げるため、減衰させるエポック数が調整すべき追加のハイパーパラメータとして生じてしまう。現代の標準手法 (Loshchilov and Hutter 2016) は、AlexNet 以降（7.1 節）ですでにおなじみのものである。すなわち、0 からピーク学習率 η_0 まで t_{w} ステップかけて線形に引き上げる**ウォームアップ**に続き、滑らかな半コサイン曲線でステップ T において 0 になるまで**減衰**させる（コサイン減衰、cosine decay）。ウォームアップは初期の発散を防ぐだけでなく、本格的なステップ幅での更新が始まる前にバッチ正規化の移動統計量や適応的オプティマイザのモーメントが落ち着く時間を与え、コサイン減衰は急激な減衰の閾値そのものを不要にする。以下では、1 エポックあたり 79 ミニバッチとして、`step_decay`（30 エポック、2 エポックのウォームアップ付き）と `d2l.cosine_warmup`（45 エポック、3 エポックのウォームアップ付き）を並べてプロットする。これは本節後半の実験で用いるホライズンと完全に一致している。

```
def step_decay(t, num_steps, warmup_steps):
    """Recipe A: linear warmup, then divide by 10 at 60% and 85% of steps."""
    warmup = min((t + 1) / warmup_steps, 1)
    return warmup * 0.1 ** sum(t >= int(f * num_steps) for f in (0.6, 0.85))

steps_per_epoch = 79
steps_a = list(range(30 * steps_per_epoch))  # Recipe A: 30 epochs
steps_b = list(range(45 * steps_per_epoch))  # Recipe B: 45 epochs
d2l.plot([[t / steps_per_epoch for t in steps_a],
          [t / steps_per_epoch for t in steps_b]],
         [[step_decay(t, len(steps_a), 2 * steps_per_epoch) for t in steps_a],
          [float(d2l.cosine_warmup(t, len(steps_b), 3 * steps_per_epoch))
           for t in steps_b]],
         xlabel='epoch', ylabel='learning rate / base rate',
         legend=['step decay (30 epochs)', 'cosine decay (45 epochs)'])
```



7.6.2.3 Mixup

Mixup は、訓練例のペアの凸結合（convex combination）を用いて訓練することで分類器を正則化する手法である (H. Zhang et al. 2018)。ミニバッチごとに混合係数 $\lambda \sim \text{Beta}(\alpha, \alpha)$ をサンプリングし、各データ例 i をランダムにシャッフルした相手のデータ例 j と組にして次を構成する。

$$\begin{aligned}\tilde{\mathbf{x}} &= \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j,\\ \tilde{\mathbf{y}} &= \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j,\end{aligned} \tag{7.9}$$

ここで、$\mathbf{y}_i$ と $\mathbf{y}_j$ はワンホットなクラスベクトルであるため、$\tilde{\mathbf{y}}$ はソフトなターゲット分布となる。交差エントロピーはターゲット分布に関して線形であるため、$\tilde{\mathbf{y}}$ に対する損失は、λ で重み付けした y_i および y_j に対する交差エントロピーの和に等しくなり、より簡潔かつ高速に計算できる。訓練例のペア間で予測が線形に補間されるよう促すことで、個々のデータ点周辺での急峻な丸暗記が抑制される。典型的な小さい値（$\alpha \in [0.1, 0.2]$）ではベータ分布が 0 と 1 の近くに集中するため、大半のミニバッチには軽い摂動が加わるにとどまり、一部のミニバッチだけが強く混合される。以下では、ミニバッチごとに共通の λ を 1 つ用いる `mixup` を実装し、7.7 節で再利用できるよう `d2l` に保存する。

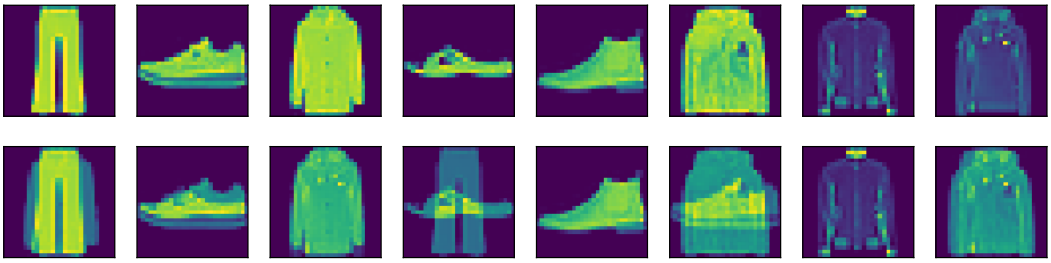
```
def mixup(key, X, y, alpha):
    """Mix a minibatch with a shuffled copy; return both label sets."""
    key_lam, key_perm = jax.random.split(key)
    lam = jax.random.beta(key_lam, alpha, alpha)
    perm = jax.random.permutation(key_perm, X.shape[0])
    return lam * X + (1 - lam) * X[perm], y, y[perm], lam
```

効果を視覚的にわかりやすく確認するため、Fashion-MNIST の衣類画像のミニバッチを $\alpha = 2$ で混合する。この設定では λ が 0.5 付近に集中する（実際の訓練でははるかに小さい α を用いる）。上段は元の画像、下段は重ね合わされた混合画像である。

```
data = d2l.FashionMNIST(batch_size=8)
X, y = map(jnp.asarray, next(iter(data.train_dataloader())))
```

```
X_mix, y_a, y_b, lam = mixup(d2l.get_key(), X, y, alpha=2.0)
d2l.show_images(jnp.concatenate([X, X_mix]).squeeze(-1), 2, 8)
print(f'lambda = {float(lam):.2f}')
```

```
lambda = 0.59
```



Mixup は実際の写真には決して現れない幽霊のような二重露光画像を作り出すため、**CutMix** は代わりに空間的な混合を行う (Yun et al. 2019)。すなわち、一方の画像から矩形パッチを切り出して他方の画像に貼り付け、パッチの面積に比例してラベルを混合することで、すべてのピクセルを局所的に自然なまま保つ。現代の画像系レシピの多くは、ミニバッチごとに Mixup と CutMix をランダムに切り替えている（後述の演習問題を参照）。

7.6.2.4 確率的深さ

ドロップアウト（4.6 節）が訓練中に個々の活性化値をランダムに 0 にするのに対し、**確率的深さ**（stochastic depth）は残差分岐全体をランダムに無効化する (Huang et al. 2016)。確率 p でブロックの残差分岐がバイパスされて入力がそのまま通過するため、実質的により浅いサブネットワークのアンサンブルを訓練することになる。恒等ショートカット（identity shortcut）が常に信号を通し続けるため、残差接続のもとではこの操作を安全に行える。$\mathbf{x} + f(\mathbf{x})$ を計算するブロックに対し、確率的深さは分岐を次の式に置き換える。

$$\mathbf{x} + \frac{b}{1-p} f(\mathbf{x}), \qquad b \sim \text{Bernoulli}(1-p). \tag{7.10}$$

逆ドロップアウト（inverted dropout）と同様に、$1/(1-p)$ 倍でスケーリングすることで訓練中の分岐出力の期待値が不変に保たれるため、評価時の再スケーリングは不要である。現代の実装では、ミニバッチ内のデータ例ごとに独立して b をサンプリングする（以下の `drop_path` であり、ConvNeXt 用に `d2l` に保存する）。`d2l.Residual`（7.4 節）をサブクラス化し、ショートカットを加算する直前で残差分岐をドロップする。訓練モードで $p = 0.5$ とし、1,000 個のランダムな入力をこのブロックに通すと、全出力のおよそ半分が $\text{ReLU}(\mathbf{x})$ と完全に一致するはずである。

```
def drop_path(key, Y, p, training):
    """Zero each example of Y with probability p, rescale the rest."""
    if not training or p == 0:
        return Y
    shape = (Y.shape[0],) + (1,) * (Y.ndim - 1)  # One draw per example
    keep = jax.random.bernoulli(key, 1 - p, shape)
    return Y * keep / (1 - p)

class StochasticResidual(d2l.Residual):
```

```
    """A residual block whose branch is dropped with probability p."""
    deterministic: bool

    def __init__(self, num_channels, p, rngs=None):
        rngs = (nnx.Rngs(params=d2l.get_key(), drop_path=d2l.get_key())
                if rngs is None else rngs)
        super().__init__(num_channels, rngs=rngs)
        self.p, self.deterministic, self.rngs = p, False, rngs

    def set_view(self, *, deterministic):  # The Trainer switches this
        self.deterministic = deterministic

    def __call__(self, X):
        Y = self.bn2(self.conv2(nnx.relu(self.bn1(self.conv1(X)))))
        Y = drop_path(self.rngs.drop_path(), Y, self.p,
                      not self.deterministic)
        return nnx.relu(X + Y)

blk = StochasticResidual(3, p=0.5)
X = jax.random.normal(d2l.get_key(), (1000, 8, 8, 3))
Y = blk(X)
dropped = jnp.all(Y == nnx.relu(X), axis=(1, 2, 3)).mean()
print(f'fraction of examples with a dropped branch: {float(dropped):.3f}')
```

```
fraction of examples with a dropped branch: 0.509
```

深いアーキテクチャでは、ドロップ確率 p を最初のブロックの 0 から最深ブロックのピーク値 0.1〜0.5 まで線形に増加させるのが一般的である。なお、データ例ごとのマスク処理では、ドロップされた例を 0 にする前にすべての分岐をひととおり計算するため、正則化にはなるが訓練の高速化にはつながらない。これに対し、元のミニバッチ単位の定式化ではブロックごとに単一の b を引くため、ドロップされた分岐の計算を完全にスキップして計算量を節約できる。

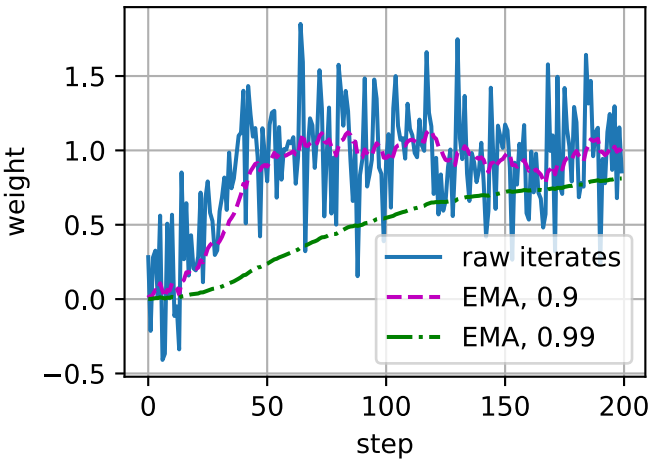
7.6.2.5 重みの平均化

Table 7.4 の最後の行にある手法は、追加コストをほぼゼロに抑えつつ、より良好な汎化をもたらす。訓練の終盤では、ミニバッチノイズによってパラメータが低損失な盆地の底で跳ね回るため、直近の反復点を平均化すると、単一のチェックポイントよりも盆地の中心に近い位置に到達できる。**確率的重み平均化**（stochastic weight averaging、SWA）が訓練終盤のスナップショットを一様に平均するのに対し (Izmailov et al. 2018)、現代の画像系レシピでは一般に**指数移動平均**（exponential moving average、EMA）を保持する。これはバッチ正規化が移動統計量の更新に用いる漸化式（7.3 節）とまったく同じものである。オプティマイザの各ステップのあと、訓練可能なパラメータの**シャドウコピー** $\bar{\theta}$ に最新の重み θ を減衰率 ρ で混合する。

$$\bar{\theta} \leftarrow \rho\bar{\theta} + (1-\rho)\theta. \tag{7.11}$$

k ステップ前のパラメータは重み $(1-\rho)\rho^k$ を持ち、実効的な平均化ホライズンは約 $1/(1-\rho)$ ステップ（$\rho = 0.99$ なら 100 ステップ）となる。最適化は通常どおり θ を更新し続け、評価時には $\bar{\theta}$ に差し替える。（バッチ正規化の移動統計量は訓練可能な重みではなくバッファであるため、EMA 重みとどう組み合わせるのが最善かは後述の演習問題で検討する。）ρ におけるトレードオフを確認するため、50 ステップかけて 0 から 1 まで上昇したのち 1 の周りでノイズを伴って振動する単一の合成重みを追跡してみよう。

```
class EMA:
    """A shadow copy of a model's parameters, updated after each step."""
    def __init__(self, model, decay=0.99):
        self.decay = decay
        # A copy: nnx.state returns references to the model's arrays
        self.shadow = jax.tree.map(jnp.copy, nnx.state(model, nnx.Param))

    def update(self, model):  # beta * avg + (1 - beta) * current
        self.shadow = optax.incremental_update(
            nnx.state(model, nnx.Param), self.shadow, 1 - self.decay)

net = nnx.Linear(1, 1, use_bias=False, kernel_init=nnx.initializers.zeros,
                 rngs=nnx.Rngs(d2l.get_key()))  # Training starts at 0
emas = {decay: EMA(net, decay) for decay in (0.9, 0.99)}
raw, avg = [], {decay: [] for decay in emas}
for t in range(200):  # Iterates rise to 1 over 50 steps, then stay noisy
    net.kernel[...] = (min(t / 50, 1)
                       + 0.3 * jax.random.normal(d2l.get_key(), (1, 1)))
    raw.append(float(net.kernel[0, 0]))
    for decay, ema in emas.items():
        ema.update(net)
        avg[decay].append(float(ema.shadow['kernel'][...][0, 0]))
d2l.plot(list(range(200)), [raw, *avg.values()], xlabel='step',
         ylabel='weight', legend=['raw iterates', 'EMA, 0.9', 'EMA, 0.99'])
```



$\rho = 0.9$（ホライズン約 10 ステップ）では、シャドウ重みは約 10 ステップ遅れて上昇に追従し、その後はノイズを劇的に減衰させる。$\rho = 0.99$（ホライズン約 100 ステップ）ではさらに滑らかになるものの、遅れがあまりに大きく、200 ステップ経過後でも依然として 1 より約 20% 低い位置にとどまっている。これは EMA の主な落とし穴を示している。実行ステップ数に比べて $1/(1-\rho)$ が大きいと、シャドウコピーは改善途上の重みに遅れをとり、最終チェックポイントよりも性能が劣ってしまうのである。したがって、0.999 や 0.9999 のような非常に大きい減衰率は、長期の訓練にのみ適している。

7.6.3 1 つのネットワークに対する 2 つのレシピ

ここまでの要素を組み合わせ、単一の固定アーキテクチャ上で直接比較テストを行ってみよう。用いるのは 7.4 節の ResNet-18 であり、`d2l.Residual` と He 初期化（4.4 節）を用いて再構築する。96×96 にリサイズした Fashion-MNIST 画像からランダムに抽出した 10,000 枚のサブセット（各試行内で両レシピが共有）で訓練する。訓練例を絞ることで、反復実験を高速に保ちつつ正則化の役割を際立たせることができる。両レシピとも、`lr` と `num_steps` から直接ステップスケジュールを計算する。

```
class FashionMNIST10k(d2l.FashionMNIST):
    """Fashion-MNIST with a random subset of num_train training images."""
    def __init__(self, batch_size=128, resize=(96, 96), num_train=10000):
        super().__init__(batch_size, resize)
        idx = jax.random.permutation(d2l.get_key(), len(self.train[1]))
        self.train = tuple(a[idx[:num_train]] for a in self.train)

class ResNet18(d2l.Classifier):
    """The ResNet-18 of the residual-networks section, from d2l.Residual."""
    def __init__(self, lr, num_steps, num_classes=10, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        self.rngs = (nnx.Rngs(params=d2l.get_key(), mixup=d2l.get_key())
                     if rngs is None else rngs)
        he_init = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        layers = [nnx.Conv(1, 64, (7, 7), strides=2, padding=3,
                           use_bias=False, kernel_init=he_init,
                           rngs=self.rngs),
                  nnx.BatchNorm(64, momentum=0.9, use_fast_variance=False,
                                rngs=self.rngs),
                  nnx.relu, max_pool]
        in_channels = 64
        for i, c in enumerate((64, 128, 256, 512)):
            for j in range(2):
                down = i > 0 and j == 0
                layers.append(d2l.Residual(
                    c, use_1x1conv=down, strides=(2, 2) if down else (1, 1),
                    in_channels=in_channels, rngs=self.rngs))
                in_channels = c
        layers += [global_avg_pool,
                   nnx.Linear(512, num_classes, kernel_init=he_init,
                              rngs=self.rngs)]
        self.net = nnx.Sequential(*layers)

def max_pool(X):  # 3x3 windows, stride 2, padding 1
    return nnx.max_pool(X, (3, 3), (2, 2), padding=((1, 1), (1, 1)))

def global_avg_pool(X):  # Average over height and width (NHWC)
    return X.mean(axis=(1, 2))
```

両モデルはまったく同じアーキテクチャと訓練データ分割を共有する一方、最適化と正則化のレシピを全面的に変更している。**レシピ A** は 2015 年当時の実践を踏襲したもので、モメンタム 0.9 の SGD、結合型重み減衰 5×10^{-4}、ピーク学習率 0.05（2 エポックの線形ウォームアップののち、訓練の 60% と 85% の時点で 10 分の 1 に減衰）、通常の交差エントロピー損失、30 エポック

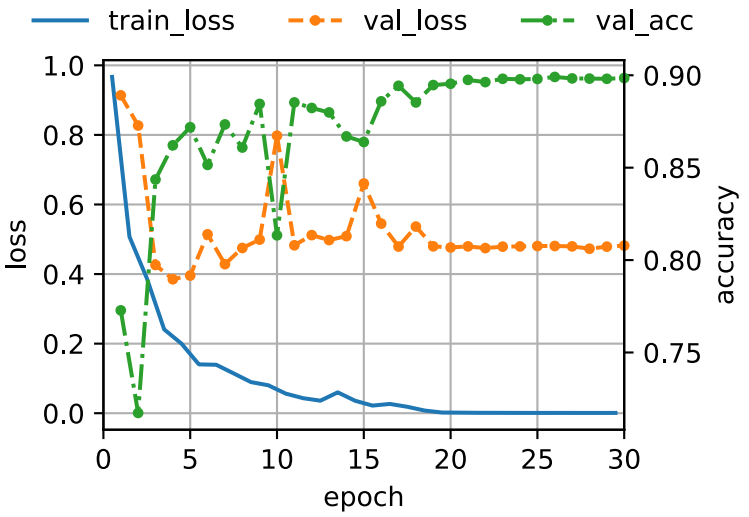
という設定である。（2 エポックの短いウォームアップは、Goyal et al. (2017) が大規模ミニバッチ SGD 向けに導入した現代の軽微な安定化処置である。これがないと、学習率 0.05 のバッチ正規化付き SGD は試行のおよそ 3 分の 1 でエポック 1 のうちに発散した。）一方、本節の**簡潔な現代的レシピ**では、重み減衰 0.05 の AdamW、ピーク学習率 0.002（3 エポックのウォームアップとコサイン減衰）、ラベル平滑化（$\epsilon = 0.1$）、Mixup（$\alpha = 0.2$）、そして（現代的な正則化が必要とする長いホライズンを反映した）45 エポックを用いる。実行時間を現実的な範囲に収めるため、RandAugment、CutMix、確率的深さ、EMA は省き、フルメニューから厳選したサブセットを検証する。

2 つの重み減衰の数値にある 100 倍の差（0.05 vs. 5×10^{-4}）に惑わされてはならない。1 ステップあたりの実際の縮小率は「学習率 × 重み減衰」であり、ピーク学習率においては AdamW が $0.002 \times 0.05 = 10^{-4}$ であるのに対し、SGD は $0.05 \times 5 \times 10^{-4} = 2.5 \times 10^{-5}$ となる。さらに SGD ではモメンタムがこれを最大 10 倍まで増幅するため、実効的な減衰はむしろ SGD の方が強い。最後に、どちらのレシピも混合していない検証画像に対する通常の交差エントロピーで検証損失を評価する。ラベル平滑化と Mixup が変更するのは訓練目的関数のみである。

```
class ClassicRecipe(ResNet18):
    """Recipe A: SGD with momentum, weight decay, warmup, step decay."""
    def configure_optimizers(self):
        decay = optax.piecewise_constant_schedule(self.lr, {
            int(0.6 * self.num_steps): 0.1, int(0.85 * self.num_steps): 0.1})
        warmup = 2 * self.num_steps // 30  # Two of the 30 epochs
        schedule = lambda t: decay(t) * jnp.minimum((t + 1) / warmup, 1.0)
        return optax.chain(optax.add_decayed_weights(5e-4),  # In the gradient
                           optax.sgd(schedule, momentum=0.9))

class ModernRecipe(ResNet18):
    """Compact modern recipe: AdamW, warmup and cosine, smoothing, Mixup."""
    def configure_optimizers(self):
        warmup = 3 * self.num_steps // 45  # Three of the 45 epochs
        schedule = lambda t: self.lr * d2l.cosine_warmup(
            t, self.num_steps, warmup)
        return optax.adamw(schedule, weight_decay=0.05)

    def training_step(self, batch):
        X, y_a, y_b, lam = mixup(self.rngs.mixup(), *batch, alpha=0.2)
        Y_hat = self(X)
        targets = lambda y: optax.smooth_labels(
            jax.nn.one_hot(y, self.num_classes), 0.1)
        smooth = lambda y: optax.softmax_cross_entropy(
            Y_hat, targets(y)).mean()
        return lam * smooth(y_a) + (1 - lam) * smooth(y_b)
```

まず、レシピ A を 30 エポック訓練し、10,000 枚の検証データ全体で評価する。

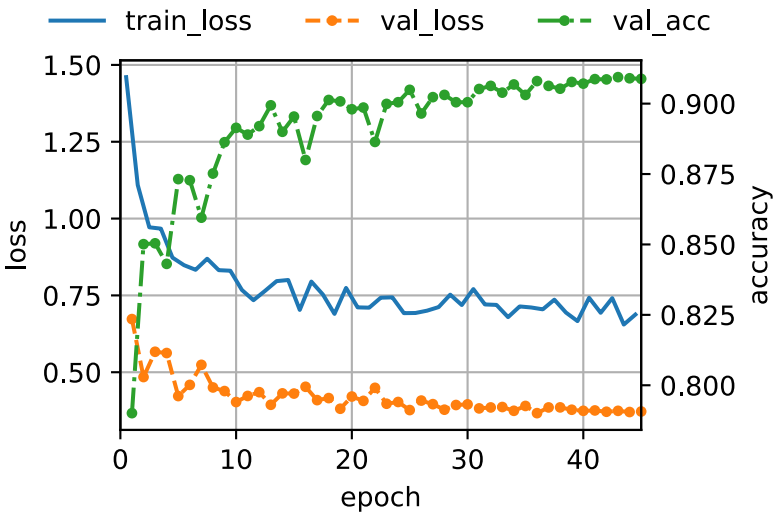
```
data = FashionMNIST10k()
steps_per_epoch = len(data.train_dataloader())  # 79 minibatches of <= 128
model = ClassicRecipe(lr=0.05, num_steps=30 * steps_per_epoch)
trainer = d2l.Trainer(max_epochs=30, num_gpus=1)
trainer.fit(model, data)
trainer.history['val_acc'][-1]
```




```
0.8985
```

次に、まったく同じアーキテクチャをまったく同じ訓練サブセット上で、簡潔な現代的レシピを用いて 45 エポック訓練する。

```
model = ModernRecipe(lr=2e-3, num_steps=45 * steps_per_epoch)
trainer = d2l.Trainer(max_epochs=45, num_gpus=1)
trainer.fit(model, data)
trainer.history['val_acc'][-1]
```




```
0.9088
```

反復試行（毎回新たな 10,000 枚の画像サブセット、初期化、ミニバッチ順序を抽選）にわたる検証正解率の範囲を Table 7.5 に示す。

Table 7.5: 両レシピのもとでの同一の ResNet-18（96×96 の Fashion-MNIST 訓練画像 10,000 枚、反復試行にわたる検証正解率の範囲）。

レシピ	エポック数	検証正解率
A（2015 年）：SGD、ウォームアップ、ステップ減衰	30	89〜90.5%
簡潔な現代的レシピ：AdamW、コサイン、平滑化、Mixup	45	90.5〜91%

同じ 10,000 枚の画像サブセットを用いたすべての対比較において、（RandAugment、CutMix、確率的深さ、EMA を省いているにもかかわらず）`ModernRecipe` は `ClassicRecipe` を 0.5〜1.5 パーセントポイント上回る。この比較からは 3 つの実践的な教訓が得られる。

1. **一括比較とアブレーション：**オプティマイザ、スケジュール、損失、データ拡張、エポック予算を一度に変更しているため、どの構成要素が最も寄与したかを切り分けるには 1 要素ずつのアブレーション（演習問題 4）を行うほかない。また、最適な学習率には $25\times$ もの開き（AdamW の 0.002 対 SGD の 0.05）があるため、オプティマイザを入れ替える際には必ず学習率を再調整しなければならない。
2. **レシピ間で生の訓練損失を比較してはならない：**`ClassicRecipe` のもとでは訓練損失が 0 に向かって急落する一方で、ロジットが過信状態になるにつれて検証損失が 0.4〜0.5 へと反発する。対照的に `ModernRecipe` のもとでは、検証損失が 0.4 未満まで下がる一方、Mixup とラベル平滑化がソフトに混合された訓練目的関数を評価するため、訓練損失は検証損失よりも**高い**まま推移する。
3. **公開リーダーボードの読み方：**訓練レシピだけで固定された ResNet-50 の ImageNet 正解率が > 4 ポイントも動く以上 (Wightman et al. 2021)、過去の論文から転記したベースラインに対する 1 ポイントの改善という主張は、レシピとアーキテクチャを混同している。たとえレシピを揃えたとしても、ImageNet にはテストセットのノイズや複数物体の曖昧さがあるため (Recht et al. 2019; Beyer et al. 2020)、1 ポイント未満の ImageNet top-1 正解率の差よりも、物体検出やセグメンテーションへの転移性能の方がバックボーンの品質を鋭く測る指標となることが多い。

7.6.4 まとめ

2015 年から 2020 年代初頭にかけて、畳み込みネットワークの訓練レシピはあらゆる軸で進化を遂げた。分離型重み減衰を備えた適応的オプティマイザ（AdamW、LAMB）、ウォームアップ付きコサインスケジュール、RandAugment・Mixup・CutMix、ラベル平滑化、確率的深さ、より長い訓練ホライズン、そして EMA による重みの平均化である。そうしたレシピの 1 つ（LAMB、コサインスケジュール、RandAugment と Mixup および CutMix、確率的深さ、600 エポック、重み平均化なし）は、無変更の ResNet-50 の ImageNet 正解率を 76.1% から 80.4% へと引き上げた。各構成要素はわずか数行のコードで実装でき、その簡潔なサブセットだけでも `FashionMNIST10k` 上の ResNet-18 を着実に改善する。続く 7.7 節では、現代的なレシピを固定したうえで、ResNet のアーキテクチャ自体を 1 段階ずつ現代化していく。

7.6.5 演習問題

1. **二値交差エントロピーと Mixup**。ImageNet 分類器の訓練において、Wightman et al. (2021) は K クラスのソフトマックス交差エントロピーではなく、二値交差エントロピー（K 個の各クラスを独立したシグモイド二値分類タスクとして扱う）を用いている。混合された画像に異なる 2 つのクラスの物体が同時に含まれるとき、独立したシグモイド出力が Mixup や CutMix と自然に相性がよい理由を説明せよ。

2. [code] **CutMix**。CutMix (Yun et al. 2019) は $\lambda \sim \mathrm{Beta}(1,1) = \mathrm{Uniform}(0,1)$ をサンプリングし、シャッフルした相手の画像から目標面積比 $1-\lambda$ を覆うランダムな矩形パッチを貼り付ける。`cutmix` を `mixup` とそのまま差し替え可能な変種として実装し、矩形を画像境界でクリッピングしたあとに残るピクセルの**厳密な**割合を用いてワンホットラベルを混合するようにせよ。`ModernRecipe` を `FashionMNIST10k` 上で、Mixup の代わりに CutMix を用いた場合と、ミニバッチごとに Mixup と CutMix をランダムに切り替えた場合のそれぞれで訓練し、検証正解率を Table 7.5 と比較せよ。

3. [code] **重み平均化のホライズン（EMA）**。オプティマイザの各ステップ後にシャドウ重みを更新し、シャドウパラメータを用いて検証正解率を評価するように、`EMA` クラスを `ModernRecipe` に組み込め。減衰率 $\rho \in \{0.9, 0.99, 0.999\}$ を Equation 7.11 において変化させて実験せよ。最も性能のよい ρ を実効ホライズン $1/(1-\rho)$ および総ステップ数（$45 \times 79 = 3{,}555$）と関連付けて考察し、EMA 重みを評価する際にバッチ正規化の移動統計量をどのように扱うべきかを説明せよ。

4. [code] **1 要素ずつのレシピ・アブレーション**。●Table 7.5 では、5 つの構成要素をすべて同時に変更したときに `ModernRecipe` が `ClassicRecipe` を 0.5〜1.5 パーセントポイント上回った。

    1. `ClassicRecipe` と `ModernRecipe` の両方を `FashionMNIST10k` 上で 5 回ずつ実行し、試行間の検証正解率の標準偏差を推定せよ。
    2. `ModernRecipe` を出発点として、構成要素を 1 つずつ `ClassicRecipe` の対応する設定に戻せ。すなわち、(i) Mixup を外す、(ii) ラベル平滑化を外す、(iii) コサインウォームアップを `step_decay` に置き換える、(iv) AdamW をモメンタム付き SGD に置き換える（ピーク学習率は 0.05 に再調整する）、(v) 訓練を 45 エポックから 30 エポックに短縮する。試行間のノイズ水準を超える変化をもたらすのはどの個別変更か。
    3. Mixup とラベル平滑化を有効にした状態で、2×2 の「オプティマイザ（SGD vs. AdamW）$\times$ スケジュール（`step_decay` vs. `cosine_warmup`）」の表を完成させよ。オプティマイザとスケジュールの間に相互作用はあるか。

5. [code] **レシピへの確率的深さの導入**。●本節の簡潔な現代的レシピでは確率的深さを省略した。これを `ResNet18` に追加せよ。

    1. 1×1 ショートカットを持つストライド 2 のダウンサンプリングブロックに対応できるよう `StochasticResidual` を拡張し、ドロップ確率がブロック 1 の 0 からブロック 8 の $p_{\max}$ まで線形に増加する `ResNet18` を構築せよ。1 回の順伝播あたりに有効となる残差分岐の個数の期待値を $p_{\max}$ の関数として求めよ。
    2. $p_{\max} \in \{0.1, 0.2, 0.5\}$ を変化させて `ModernRecipe` を `FashionMNIST10k` 上で実行せよ（$p_{\max} = 0$ と比較すること）。8 ブロックの浅いネットワークでは、50 ブロックや 100 ブロックのネットワークに比べて大きい $p_{\max}$ の恩恵が小さいのはなぜか。
    3. Huang et al. (2016) によるミニバッチ単位の変種（ブロックごとに単一のベルヌーイ変数 b をサンプリングし、$b = 0$ のときは残差分岐の評価を完全にスキップする方式）を実装せよ。$p_{\max} = 0.5$ において、スキップされる分岐の理論上の割合と実測された訓練の実時間の短縮率を比較せよ。

7.7 ConvNeXt：ResNet の段階的な現代化

本章でここまで扱ってきたアーキテクチャはすべて畳み込みをベースとしており、2020 年までは畳み込みネットワークがあらゆる主要な画像分類のリーダーボードで頂点に君臨していた。そこへ、重なりのある空間畳み込みを 1 つも持たず、**アテンション**を中心に構築されたアーキテクチャが登場して首位を奪った。最初のビジョン Transformer は 2020 年後半、ImageNet よりはるかに大規模な事前学習データセットの力を借りてこれを成し遂げた。しかし 2021 年の

Swin Transformer (Liu et al. 2021) はそのような助けを必要とせず、ImageNet のみで訓練して 81.3% の top-1 正解率を達成した。これに対し、同規模の ResNet-50 は 76.1% であった。この 5 ポイントもの飛躍はアテンション自体がもたらしたものなのだろうか。そしてもしそうなら、畳み込みネットワークは時代遅れになったのだろうか。

ここではビジョン Transformer の概略を述べるにとどめる（詳細は第 11 章で展開する）。**ビジョン Transformer** (Dosovitskiy et al. 2021) は、画像をパッチの格子に分割し、各パッチを特徴ベクトルとして埋め込み、その系列を同一構造のブロックの積層に通して処理する。各ブロックは 2 つのサブ層を持つ。すなわち、すべてのパッチが他のパッチのデータ依存な重み付き平均を計算できるようにする**アテンション**層と、それに続いて各パッチベクトルに独立に適用されるポイントワイズ MLP である。Swin は、局所的な 7×7 ウィンドウ内でアテンションを計算し、ステージ間でダウンサンプリングすることで、階層的な畳み込みネットワークの構造を復活させている。

7.6 節を踏まえれば、5 ポイントの差をアテンションだけの功績とみなすことにはすぐさま疑問を持つべきだろう。Swin はアテンション以外にも、少なくとも十数個の点で 2015 年の ResNet と異なっている。300 エポックの現代的な訓練レシピ（training recipe）で訓練し、パッチステムで画像をスライスし、ステージ間で深さの配分を変え、さらに異なる活性化関数と正規化を採用しているのである。これらの要因を切り分けるため、Z. Liu et al. (2022) は画期的な対照実験を実施した。標準的な ResNet-50 から出発し、アテンションを一切追加することなく Swin に着想を得た現代化を 1 つずつ導入し、変更のたびに ImageNet の top-1 正解率を測定したのである。こうして得られた純粋な畳み込みネットワーク **ConvNeXt** は 82.0% に達し、本章ですでに構築してきた構成要素、すなわち 7.6 節の現代的な訓練レシピ、7.5 節のデプスワイズ畳み込み（depthwise convolution）と逆ボトルネック（inverted bottleneck）、および 7.3 節の層正規化（layer normalization）だけで Swin-T をわずかに上回った。このアブレーションの各行には ImageNet 全体の訓練が必要となるため、本節ではまず公開されたロードマップを段階的にたどり、続いて ConvNeXt をゼロから実装したうえで、パラメータ数を揃えた ResNet-18 から出発する独自のコンパクトな 3 ステップの現代化ロードマップを Fashion-MNIST 上で実行する。

7.7.1 現代化のロードマップ

Table 7.6 に、Z. Liu et al. (2022) が検証した一連の変更を順番通りに示す。各行はそれより上のすべての変更を累積しつつ、画像あたり約 45 億回の積和演算（multiply-add）という Swin-T の計算予算付近に計算量を保っている（論文中では 4.5 GFLOPs と報告されている）。

Table 7.6: ConvNeXt の現代化ロードマップ (Z. Liu et al. 2022)。設計変更を累積したときの ResNet-50 の ImageNet top-1 正解率であり、各行はほぼ一定のコストのもとでそれより上のすべての行を含む。同程度のコストの Transformer である Swin-T は 81.3% に達する。

変更項目	内容	Top-1
（出発点）	2015 年のレシピによる ResNet-50	76.1%
現代的な訓練レシピ	300 エポック、AdamW、Mixup、ラベル平滑化、確率的深さ（7.6 節）	78.8%
ステージ比	ステージごとのブロック数 $(3,4,6,3) \to (3,3,9,3)$	79.4%
パッチ化ステム	ストライド 2 の 7×7 畳み込みとプーリング $\to$ ストライド 4 の単一の 4×4 畳み込み	79.5%
デプスワイズ畳み込み	3×3 畳み込みをデプスワイズ化し、幅を 64 から 96 チャネルに拡大	80.5%

変更項目	内容	Top-1
逆ボトルネック	ブロック内で圧縮する代わりに 4× に拡大	80.6%
大カーネル	デプスワイズ畳み込みを先頭へ移動し、7×7 に拡大	80.6%
ReLU → GELU	Transformer と同様の滑らかな活性化関数	80.6%
活性化関数の削減	ブロックあたりの活性化関数を 3 個から 1 個に削減	81.3%
正規化の削減	ブロックあたりの正規化を 3 個から 1 個に削減	81.4%
BN → LN	各位置でのチャネル方向の層正規化	81.5%
独立したダウンサンプリング	ステージ間に LN とストライド 2 の 2×2 畳み込みを配置	82.0%

いきなり最初の行が、ネットワークの層に一切手を加えることなく最大の単一の上昇をもたらしている。7.6 節と呼応するように、訓練レシピを刷新するだけで、無変更の ResNet-50 が 2.7 パーセントポイント向上する。2015 年の ResNet-50 と ConvNeXt を隔てる合計 5.9 ポイントの改善のうち、（この適用順序では）2.7 ポイントが訓練、3.2 ポイントがアーキテクチャに由来しており、現代的な Transformer を 2015 年のレシピによる ResNet と競わせていた比較では、レシピとアーキテクチャが交絡していたことが裏づけられる。

7.7.1.1 マクロ設計

続く 2 行は、ネットワークのマクロ構造を組み替える。Swin-T は 4 つのステージに 1:1:3:1 の比率でブロックを配分し、特徴マップが小さく計算コストが安いステージ 3 に深さを集中させている。ResNet-50 の $(3, 4, 6, 3)$ という配置を $(3, 3, 9, 3)$ に変更すると、0.6 ポイント向上する。次はステムである。ResNet が重なりのあるストライド 2 の 7×7 畳み込みに最大プーリングを続けて 4 分の 1 にダウンサンプリングするのに対し、ビジョン Transformer は画像を重なりのないパッチに分割し、各パッチを共有の線形層で射影する。これは代数的には、重なりのないストライド 4 の 4×4 畳み込みと同一である（6.3 節）。この**パッチ化**（patchify）ステムに置き換えても正解率は維持され（79.5%）、Transformer とまったく同様に 4×4 パッチごとに 1 つのトークンベクトルが生成される。

7.7.1.2 デプスワイズ畳み込みと逆ボトルネック

Transformer ブロックは計算を 2 つの段階にきれいに分解する。アテンションが**空間位置にわたって**情報を混合し、MLP が各位置で**チャネルにわたって**情報を混合する。7.5.1.1 節で見たように、畳み込みネットワークもこれとまったく同じ分解を実行できる。すなわち、**デプスワイズ**畳み込みが各チャネル内で空間方向に情報を混合し、1×1 畳み込みが各ピクセルでチャネル方向に情報を混合する。ResNet のボトルネック内の密な 3×3 層をデプスワイズ畳み込みに置き換えると、計算量が大幅に削減されるため、ステージ 1 の幅を 64 から 96 チャネルへと（Swin-T に合わせて）広げることができ、正解率は 80.5% へと跳ね上がる。

空間畳み込みが密ではなくデプスワイズになると、そのコストは幅に対して二乗ではなく線形で増加するため、フィルタリングの前にチャネルを圧縮する理由がなくなる。そこで ConvNeXt は MobileNetV2 の**逆ボトルネック**（7.5 節）を採用し、細い残差ストリームを保ちながらブロック内部で 4× に拡大する（Transformer の MLP と同じ $1\times \to 4\times \to 1\times$ の構造）。これにより正解率は 80.6% へとわずかに上昇する。

空間フィルタリングが軽量になったことで、カーネルサイズの拡大も容易になる。まず、作用対象が c チャネル（$4c$ ではなく）になるようデプスワイズ畳み込みをブロックの先頭へ移動させると、正解率は 0.7 ポイント低下する（79.9% となり、ロードマップ上で唯一の低下である）。しかし、そのカーネルを 3×3 から 7×7 に拡大すると、その低下分は即座に取り戻される。1 つの 7×7 フィルタは、3 層重ねた 3×3 層の受容野と一致する（6.3 節）。どちらも同じ範囲を覆うのに、なぜ単一の大カーネルが有効なのだろうか。積層構造はウィンドウ内を不均一に覆うからである。境界のピクセルよりも中央のピクセルから出力へ至る経路の方がはるかに多いため、積層構造の**有効**受容野（effective receptive field）はほぼガウス分布状になり、名目上の受容野よりずっと小さくなる (Luo et al. 2016)。これに対し、単一の 7×7 カーネルであれば、ウィンドウ内のあらゆる位置に学習した任意の重みを 1 層で与えられる。かつての VGG（7.2.1 節）では、密な 7×7 カーネルは計算コストが高すぎて実用的でなかったが（$k^2c_{\text{i}}c_{\text{o}}$）、デプスワイズカーネル（$k^2c$）であれば、$3\times3$ から 7×7 へ拡大してもブロック全体の計算量のごく一部が $49/9\approx5$ 倍になるにすぎない。正解率はちょうど 7×7 で飽和し、Swin の局所アテンションのウィンドウサイズとぴったり一致する。

7.7.1.3 ミクロ設計

最後の変更群は、ブロック内部の個々の層を洗練させるものである。ReLU を、Transformer で好まれる滑らかな活性化関数である **GELU** (Hendrycks and Gimpel 2016) に置き換えても、それ単独では正解率は変わらない（80.6%）。**実際に**大きな効果をもたらすのは、活性化関数と正規化の**数**を減らすことである。従来の ResNet のボトルネックは 3 つの畳み込みすべてに正規化と活性化関数を続けるのに対し、Transformer ブロックは MLP 内部に単一の活性化関数しか持たず、正規化の数も少ない。3 つの GELU のうち 2 つを取り除き、2 つの 1×1 層の間にある 1 つだけを残すと、正解率は 0.7 ポイント跳ね上がり、表中のミクロ設計として最大の改善となる（2 つの非線形性を取り除くとなぜ正解率が向上するのかについて明快な理論的証明はまだなく、純粋な経験的知見とみなすのが妥当である）。さらに正規化を 3 つから 1 つに減らすと 0.1 ポイント向上し、残った唯一の正規化をバッチ正規化から位置ごとのチャネル方向の**層正規化**（7.3.4.1 節）へ切り替えるとさらに 0.1 ポイント向上する（81.5%）。Figure 7.20 に、元の ResNet のボトルネックと、こうして得られた ConvNeXt ブロック（後述するチャネルごとの学習可能な層スケール（layer scale）γ を含む）の比較を示す。

最後に、ResNet が各ステージの先頭ブロック内部でストライド 2 の畳み込みによりダウンサンプリングするのに対し、Swin はステージ間に専用のパッチ統合層を挿入する。独立したダウンサンプリング層（ステージ間の層正規化とそれに続く 2×2 のストライド 2 畳み込みに加え、訓練を安定させるためのステム直後および線形ヘッド直前の正規化）を採用することで、ロードマップは 82.0% で完結する。Figure 7.21 に、古典的なステム・ボディ・ヘッド構成（7.2.3 節）のもとで $(3,3,9,3)$ 個のブロックを幅 $(96,192,384,768)$ で積み重ねた ConvNeXt-T バックボーンの全体像を示す。

同等の計算量のもとで、ConvNeXt-T はアテンション層を 1 つも使わずに Swin-T を上回る。ここから得られる教訓は、畳み込みが本質的にアテンションより優れているということではなく、2021 年時点での正解率の差はアテンション単独ではなく訓練レシピとアーキテクチャ設計の洗練に起因していたということである。条件を揃えて調整すれば、両方のパラダイムは同等にスケールする。

7.7.2 実装

現代化のロードマップでは追加した層よりも削ぎ落とした層の方が多いため、ConvNeXt ブロックは驚くほど簡潔に実装できる。

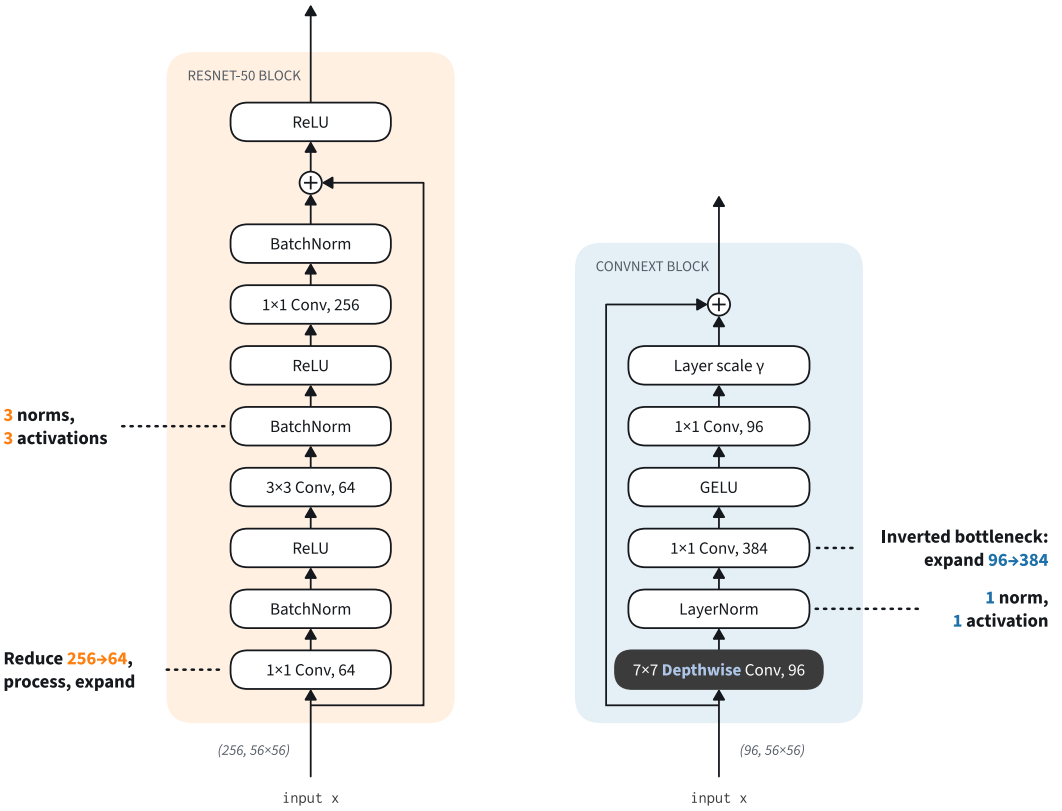


Figure 7.20: ResNet-50 のボトルネックブロックと、ロードマップによって変換された後の ConvNeXt ブロック。3 つの正規化と 3 つの活性化関数がそれぞれ 1 つずつになり、圧縮・処理・拡大のボトルネックが 7×7 デプスワイズ畳み込みを先頭に持つ逆ボトルネックへと変わり、さらにチャネルごとの学習可能な層スケール γ が加算前のブランチに乗算される。

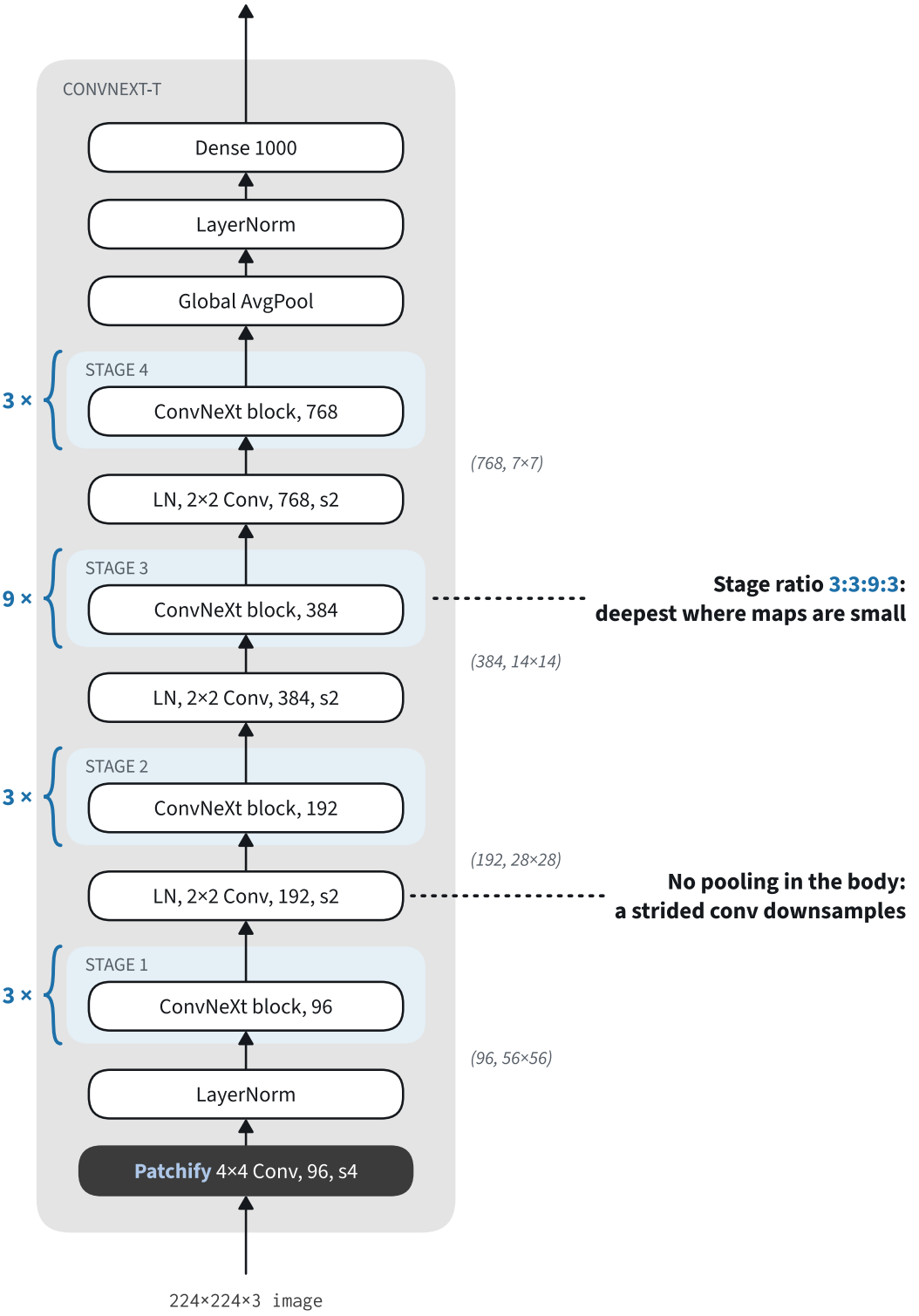


Figure 7.21: ConvNeXt-T。パッチ化ステムが畳み込みとプーリングを置き換え、3、3、9、3 ブロックの各ステージが独立した LN と 2×2 のストライド 2 ダウンサンプリング層で接続される。ボディのどこにもプーリングは存在しない。

7.7.2.1 ConvNeXt ブロック

各ブロックは 7×7 デプスワイズ畳み込み、位置ごとの層正規化、1×1 の拡大 ($c \to 4c$)、GELU、および 1×1 の射影（$4c \to c$）を順に適用し、最後に残差入力を加算する。このブロックには、論文で導入された訓練安定化の仕組みが 2 つ組み合わされている。**層スケール** (Touvron, Cord, Sablayrolles, et al. 2021) は、学習可能なチャネルごとのベクトル γ（初期値 10^{-6}、バッチ正規化のスケールとは別物）を残差ブランチに乗算することで、すべてのブロックがほぼ恒等写像として始まるようにし、深いネットワークの最適化を滑らかに立ち上げる。これは層正規化を用いるブロックにおいて、ResNet で残差ブランチ最後のバッチ正規化スケールを 0 から始める手法（7.4.2 節）と同じ役割を果たす。**確率的深さ**（stochastic depth）は、訓練中に `d2l.drop_path`（7.6 節）を用いて残差ブランチ全体をランダムにドロップする。

層正規化は各空間位置のチャネルベクトルに対して作用するため、チャネルラストのテンソル (`batch, height, width, channels`) においては最終軸に対する標準的な `LayerNorm` となり、2 つの 1×1 畳み込みも同じ軸に対する通常の線形層となる。Z. Liu et al. (2022) に従い、すべての畳み込み層と線形層の重みを $\mathcal{N}(0, 0.02^2)$ から初期化し、バイアスは 0 とする。

画像はチャネルラストで格納されるため、ブロックはライブラリの層正規化と全結合層をそのまま使用する。

```
def layer_norm(dim, rngs):
    return nnx.LayerNorm(dim, epsilon=1e-6, rngs=rngs)

class ConvNeXtBlock(nnx.Module):
    """Depthwise 7x7, norm, 1x1 expand, GELU, 1x1 project, scaled residual."""
    def __init__(self, dim, drop_prob=0.0, layer_scale=1e-6, norm=layer_norm,
                 rngs=None):
        rngs = (nnx.Rngs(params=d2l.get_key(), dropout=d2l.get_key())
                if rngs is None else rngs)
        w_init = nnx.initializers.normal(0.02)
        self.dwconv = nnx.Conv(dim, dim, kernel_size=(7, 7), padding='same',
                               feature_group_count=dim, kernel_init=w_init,
                               rngs=rngs)
        self.norm = norm(dim, rngs)
        self.pwconv1 = nnx.Linear(dim, 4 * dim, kernel_init=w_init,
                                  rngs=rngs)
        self.pwconv2 = nnx.Linear(4 * dim, dim, kernel_init=w_init,
                                  rngs=rngs)
        self.gamma = nnx.Param(layer_scale * jnp.ones(dim))
        self.drop_prob, self.rngs = drop_prob, rngs
        self.deterministic = False

    def set_view(self, *, deterministic):
        self.deterministic = deterministic

    def __call__(self, X):
        Y = self.norm(self.dwconv(X))
        Y = self.pwconv2(nnx.gelu(self.pwconv1(Y), approximate=False))
        return X + d2l.drop_path(self.rngs.dropout(), self.gamma * Y,
                                 self.drop_prob, not self.deterministic)
```

7.7.2.2 ネットワーク全体

VGG（7.2.1 節）と同様に、ネットワーク全体はステージ仕様（depth, channels）のタプルを順にたどり、先頭にパッチ化ステムと層正規化を置き、ステージ間に専用のダウンサンプリング層を挿入する（ボディのどこにも空間プーリングは置かない）。確率的深さのドロップ確率は、最初のブロックの 0 から最後のブロックの drop_path_max まで線形に増加させる。Fashion-MNIST に対しては、幅 $(40, 80, 160, 320)$、ステージ深さ $(2, 2, 6, 2)$ を持つコンパクトな **ConvNeXt-atto** の構成を採用し、ConvNeXt-T と同様に第 3 ステージを最も深く保つ。norm を引数として、stem をメソッドとして公開しておくことで、後続のアブレーションにおける差し替えが容易になる。

```
class ConvNeXt(d2l.Classifier):
    def __init__(self, arch=((2, 40), (2, 80), (6, 160), (2, 320)),
                 lr=2e-3, num_classes=10, drop_path_max=0.1, norm=layer_norm,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = (nnx.Rngs(params=d2l.get_key(), dropout=d2l.get_key(),
                         mixup=d2l.get_key()) if rngs is None else rngs)
        self.rngs = rngs
        w_init = nnx.initializers.normal(0.02)
        depths = [d for d, c in arch]
        rates = jnp.linspace(0, drop_path_max, sum(depths)).tolist()
        layers = self.stem(arch[0][1], norm, rngs)
        c_prev, b = arch[0][1], 0
        for i, (depth, c) in enumerate(arch):
            if i > 0:  # Separate downsampling layer between stages
                layers += [norm(c_prev, rngs),
                           nnx.Conv(c_prev, c, kernel_size=(2, 2),
                                    strides=(2, 2), kernel_init=w_init,
                                    rngs=rngs)]
            for _ in range(depth):
                layers.append(ConvNeXtBlock(c, rates[b], norm=norm, rngs=rngs))
                b += 1
            c_prev = c
        layers += [global_avg_pool, norm(c_prev, rngs),
                   nnx.Linear(c_prev, num_classes, kernel_init=w_init,
                              rngs=rngs)]
        self.net = nnx.Sequential(*layers)

    def stem(self, c, norm, rngs):  # Patchify: a 4x4 convolution, stride 4
        return [nnx.Conv(1, c, kernel_size=(4, 4), strides=(4, 4),
                         kernel_init=nnx.initializers.normal(0.02),
                         rngs=rngs),
                norm(c, rngs)]

def global_avg_pool(X):  # Average over height and width (NHWC)
    return X.mean(axis=(1, 2))
```

64×64 の Fashion-MNIST 入力に対し、パッチ化ステムは 16×16 の格子を生成し、3 つのダウンサンプリング層がグローバル平均プーリングの前に解像度を 8×8、4×4、2×2 へと段階的に縮小する。出力の形状を確認し、訓練可能なパラメータ数を数えてみよう。

```
model = ConvNeXt()
assert model(jnp.zeros((1, 64, 64, 1))).shape == (1, 10)
d2l.num_params(model)
```

```
3376450
```

この ConvNeXt-atto モデルは正確に 3,376,450 個のパラメータを持ち、これは ResNet-18 の 1120 万個（7.4 節）のおよそ 3 分の 1 である。幅やカーネルサイズの設定ミスがあればこの正確な数値はただちにずれるため、パラメータ数の一致を確認することは再実装における信頼性の高い検証手段となる。

7.7.3 Fashion-MNIST での小規模ロードマップ

Table 7.6 のすべての選択は、130 万枚の 224×224 の ImageNet 写真で検証されたものであった。同じアーキテクチャ上の判断は、小さな 64×64 のグレースケール衣類画像にもそのまま通用するのだろうか。これを確かめるため、独自のコンパクトな累積ロードマップを実行する。パラメータ数を揃えた対照モデルとして、基本幅を 64 から 35 チャネルに狭めた ResNet-18（パラメータ数 3.35M）から出発し、ConvNeXt の変更を 3 つの累積ステップで適用していく。すべてのモデルは、7.6 節のコンパクトな現代的レシピ（分離型重み減衰（decoupled weight decay）を伴う AdamW、10% の線形ウォームアップとコサイン減衰、ラベル平滑化（label smoothing）、および Mixup。AdamW の詳細は第 9 章で扱う）のもとで、Fashion-MNIST 全体に対して 20 エポック訓練する。アーキテクチャの影響を純粋に切り分けるため、ConvNeXt の確率的深さの率は 0 に設定し、両モデル族で正則化を完全に同一にする。

訓練手順を `ModernRecipe` ミックスインとしてまとめておくことで、検証損失を通常の交差エントロピーに保ったまま、ロードマップ内のすべてのモデルにまったく同じオプティマイザ、スケジュール、および混合訓練ステップを継承させられる。

```
class ModernRecipe:
    """AdamW, linear warmup then cosine decay, label smoothing, Mixup."""
    def configure_optimizers(self):
        warmup = self.num_steps // 10
        schedule = lambda t: self.lr * d2l.cosine_warmup(t, self.num_steps,
                                                         warmup)
        return optax.adamw(schedule, weight_decay=0.05)

    def training_step(self, batch):
        X, y_a, y_b, lam = d2l.mixup(self.rngs.mixup(), *batch, alpha=0.2)
        Y_hat = self(X)
        smooth = lambda y: optax.smooth_labels(
            jax.nn.one_hot(y, Y_hat.shape[1]), 0.1)
        loss = lambda y: optax.softmax_cross_entropy(Y_hat, smooth(y)).mean()
        return lam * loss(y_a) + (1 - lam) * loss(y_b)

class ModernConvNeXt(ModernRecipe, ConvNeXt):
    def __init__(self, num_steps, lr=2e-3, norm=layer_norm):
        super().__init__(lr=lr, drop_path_max=0.0, norm=norm)
        self.num_steps = num_steps
```

ベースラインとなる `CompactResNet18` は、ResNet 標準の 7×7 ストライド 2 ステムと最大プーリングの後ろに、幅 35、70、140、280 にわたって `d2l.Residual` ブロック（7.4 節）を積み重ねる。

```
class CompactResNet18(ModernRecipe, d2l.Classifier):
    """ResNet-18 with base width 35 instead of 64, trained by the recipe."""
    def __init__(self, num_steps, lr=2e-3, num_classes=10, base=35,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = (nnx.Rngs(params=d2l.get_key(), mixup=d2l.get_key())
                if rngs is None else rngs)
        self.rngs = rngs
        w_init = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        layers = [nnx.Conv(1, base, (7, 7), strides=(2, 2), padding=3,
                           use_bias=False, kernel_init=w_init, rngs=rngs),
                  nnx.BatchNorm(base, momentum=0.9, use_fast_variance=False,
                                rngs=rngs), nnx.relu, max_pool]
        c_prev = base
        for i, c in enumerate((base, 2 * base, 4 * base, 8 * base)):
            for j in range(2):
                down = i > 0 and j == 0
                layers.append(d2l.Residual(
                    c, use_1x1conv=down, strides=(2, 2) if down else (1, 1),
                    in_channels=c_prev, rngs=rngs))
                c_prev = c
        layers += [global_avg_pool,
                   nnx.Linear(c_prev, num_classes, kernel_init=w_init,
                              rngs=rngs)]
        self.net = nnx.Sequential(*layers)

def max_pool(X):  # 3x3 windows, stride 2, padding 1
    return nnx.max_pool(X, (3, 3), (2, 2), padding=((1, 1), (1, 1)))
```

論文の変更点をアーキテクチャの部位ごとに 3 つの明快なステップにまとめる。(1) ResNet のステムとバッチ正規化は維持したまま、ブロックとステージを ConvNeXt の 7×7 逆ボトルネック、$(2, 2, 6, 2)$ のステージ比、単一の GELU と正規化、層スケール、および独立したダウンサンプリングに置き換える（Table 7.6 の 8 行分に相当）。(2) バッチ正規化を層正規化に置き換える。(3) ResNet のステムを 4×4 パッチ化ステムに置き換えて完全な ConvNeXt に到達する（元の論文ではステムをより早い段階で置き換えていたが、各ステップへの寄与の割り当てがわずかに変わり得るとしても、最終的なアーキテクチャは順序に依存しない）。`stem` を `ResNetStemConvNeXt` でオーバーライドし、`norm` を切り替えることで、4 つのモデルすべてを簡潔に実装できる。

```
def batch_norm(dim, rngs):  # Decay 0.9 and a two-pass variance, as in ResNet
    return nnx.BatchNorm(dim, momentum=0.9, use_fast_variance=False,
                         rngs=rngs)

class ResNetStemConvNeXt(ModernConvNeXt):
    """ConvNeXt with the stem of ResNet-18 instead of the patchify stem."""
    def stem(self, c, norm, rngs):  # 7x7 convolution, stride 2, max-pooling
        return [nnx.Conv(1, c, kernel_size=(7, 7), strides=(2, 2), padding=3,
                         kernel_init=nnx.initializers.normal(0.02),
                         rngs=rngs),
                norm(c, rngs), nnx.relu, max_pool]
```

```
roadmap = {'compact ResNet-18': CompactResNet18,
           'ConvNeXt blocks and stages, BN': (
               lambda num_steps: ResNetStemConvNeXt(num_steps,
                                                    norm=batch_norm)),
           '+ BN -> LN': ResNetStemConvNeXt,
           '+ patchify stem = ConvNeXt': ModernConvNeXt}
```

4 つのモデルすべてを順番に訓練し、それぞれのパラメータ数と最終的な検証正解率を記録しよう。

```
data = d2l.FashionMNIST(batch_size=128, resize=(64, 64))
num_steps = 20 * len(data.train_dataloader())
results = {}
for name, make_model in roadmap.items():
    model = make_model(num_steps)
    trainer = d2l.Trainer(max_epochs=20, num_gpus=1)
    trainer.board.display = False  # Four runs; we tabulate the results
    trainer.fit(model, data)
    results[name] = (d2l.num_params(model), trainer.history['val_acc'][-1])
```

続いて `results` を順にたどり、`roadmap` の各ステップにおけるパラメータ数と 20 エポック後の検証正解率を表示する。

```
for name, (params, accuracy) in results.items():
    print(f'{name:34s} {params:>9,} parameters, '
          f'validation accuracy {accuracy:.3f}')
```

```
compact ResNet-18                  3,346,675 parameters, validation accuracy 0.939
ConvNeXt blocks and stages, BN     3,377,770 parameters, validation accuracy 0.937
+ BN -> LN                         3,377,770 parameters, validation accuracy 0.939
+ patchify stem = ConvNeXt         3,376,450 parameters, validation accuracy 0.917
```

Table 7.7: Fashion-MNIST における本節のロードマップと論文のロードマップの対比。最後の列は 64×64 において現代的なレシピで 20 エポック訓練した後の検証正解率であり、複数回実行した際の安定範囲に丸めている。論文の列は、各ステップが対応する Table 7.6 の行の ImageNet top-1 改善幅を合計したものである。

ステップ	論文の対応行	ImageNet での改善幅	パラメータ数	検証正解率
コンパクトな ResNet-18	（現代的レシピによる ResNet-50）		3.35M	≈94%
ConvNeXt のブロックとステージ、BN	ステージ比〜正規化の削減、独立したダウンサンプリング	+3.0	3.38M	≈94%
BN → LN	BN → LN	+0.1	3.38M	≈94%
パッチ化ステム（本節の ConvNeXt）	パッチ化ステム	+0.1	3.38M	≈92%

Table 7.7 の結果は、64×64 の Fashion-MNIST に何が転移し、何が転移しないかを明確に示し

ている。ResNet の重なりのあるステムを維持すれば、ConvNeXt の現代化されたブロック、ステージ比、および層正規化は、パラメータ数を揃えた `CompactResNet18` と並ぶ（約 94%、画像あたり約 43〜46M 回の積和演算の範囲内）。約 92% への約 2 ポイントの低下はすべて、224×224 の ImageNet では中立だったたった 1 つのステップに起因している。すなわち、重なりのない 4×4 ストライド 4 のパッチ化ステムであり、これは 1 回の線形射影で粗い 16×16 の格子へと一気に飛び越えてしまう。ImageNet の解像度に合わせて調整されたアーキテクチャ上の選択が低解像度の領域へ無条件に転移するとは限らず、累積アブレーションを実施すれば適応させるべき構成要素をただちに特定できる。

7.7.4 限界と未解決問題

ConvNeXt は、レシピを揃えれば純粋な畳み込みネットワークが教師あり分類において階層的なビジョン Transformer に匹敵することを示したが、なお 2 つの活発な研究領域が残されている。

1. **マスク付き自己教師あり事前学習：**マスク付きオートエンコーダ (He et al. 2022) のようにランダムな画像パッチをドロップする処理はパッチトークンの系列には自然になじむが、密にスライドする畳み込みではマスクの穴を越えて情報が染み出してしまう。**ConvNeXt V2** (Woo et al. 2023) は、スパース畳み込み（sparse convolution）と Global Response Normalization（演習問題 4）によってこの隔たりを埋めている。
2. **大域的コンテキスト：**7×7 のデプスワイズフィルタであっても局所的であることに変わりはなく、超大カーネルのデプスワイズ畳み込み (Ding et al. 2022)、変形可能畳み込み（deformable convolution）、および畳み込みとアテンションのハイブリッド積層が最先端のスケールでどう比較されるかは未解決のままである。

7.7.5 まとめ

標準的な ResNet-50 から出発し、ConvNeXt は Swin に着想を得た改良を 1 つずつ適用した。すなわち、300 エポックの現代的なレシピ（+2.7 ポイント）、1:1:3:1 のステージ比、4×4 ストライド 4 のパッチ化ステム、7×7 デプスワイズ畳み込みを先頭に置いた $4\times$ 逆ボトルネック、ブロックあたり単一の GELU と LayerNorm、層スケール、そして独立した 2×2 ストライド 2 のダウンサンプリング層である。これらの変更を組み合わせることで、アテンションを使わずに ImageNet で 82.0% に到達する。64×64 の Fashion-MNIST では、ConvNeXt のボディと LayerNorm はパラメータ数を揃えた ResNet-18 と同等の性能（約 94%）を示す一方、粗い 4×4 パッチ化ステムは約 2 ポイントの低下を招く。

7.7.6 演習問題

1. **デプスワイズ層の移動とカーネル拡大の演算量。**本節の Table 7.6 において、「大カーネル」の行の直前では、各ブロックは $c \to 4c$ へ 1×1 畳み込みで拡大し、$4c$ チャネルを 3×3 デプスワイズ畳み込みでフィルタリングし、$4c \to c$ へ 2 つ目の 1×1 畳み込みで射影している。「大カーネル」のステップでは、まずデプスワイズ畳み込みをブロックの**先頭**へ移動させ（作用対象が c チャネルになり、$4c$ ではなくなる）、その後にカーネルを 7×7 へ拡大する。
    1. デプスワイズ層の空間ピクセルあたりの積和演算数を、移動前（3×3 を $4c$ に適用）、移動後（3×3 を c に適用）、および 7×7 へ（c 上で）拡大した後のそれぞれについて数えよ。正味の変化を、7×7 へ**元の位置のまま** $4c$ チャネル上で拡大した場合にかかっていたはずのコストと比較せよ。
    2. `ConvNeXtBlock` 全体のピクセルあたり積和演算数（$49c + 8c^2$）のうち、7×7 デプスワイズ畳み込みが占める割合は、$c = 96$ および $c = 384$ においてそれぞれいくらか。

2. **ConvNeXt におけるパラメータの分布。**
   1. 本節の `ConvNeXt` モデル（パラメータ数 $3{,}376{,}450$）について、(i) 7×7 デプスワイズ畳み込み、(ii) 1×1 の拡大層と射影層、(iii) ブロック内部の正規化および層スケールのベクトル、(iv) ダウンサンプリング層、(v) ステムとヘッドがそれぞれ占める正確なパラメータ割合を計算せよ。
   2. `ConvNeXtBlock` と 2 層の `Residual` ブロック（7.4 節、1×1 ショートカットなし）のパラメータ数を、幅 c の関数として表せ。大きな c に対して `ConvNeXtBlock` は漸近的に何倍パラメータ効率が高いか。また、どの設計上の選択がその倍率をもたらしているか。

3. [code] **層スケール対確率的深さ。** 層スケールの初期化を $\gamma_0 = 10^{-6}$（デフォルト）とした場合、$\gamma_0 = 1$ とした場合、および層スケールを完全に除去した場合のそれぞれについて、Table 7.7 のロードマップのレシピのもとで `ConvNeXt` を訓練し、訓練曲線を比較せよ。初期値 10^{-6} の層スケールと確率的深さ（7.6 節）はいずれも、訓練開始時に深いネットワークを実質的により浅くする。それぞれの機構がどのように働くか、訓練中のいつ有効であるか、そして評価時のネットワークを変化させるかどうかを対比せよ。

4. [code] **Global Response Normalization（ConvNeXt V2）。** ConvNeXt V2 (Woo et al. 2023) は層スケールを取り除き、各ブロック内部の GELU の直後に Global Response Normalization（GRN）を挿入する。特徴マップ $\mathbf{X}$（チャネル数 C）に対し、$g_c = \|\mathbf{X}_c\|_2$ を空間方向の ℓ_2 ノルム（チャネル c に対するもの）とし、$\bar{g} = \frac{1}{C}\sum_{c=1}^{C} g_c$ とおく。GRN は学習可能なアフィン変換と残差を用いて、相対的な応答により各チャネルを再スケーリングする。

$$\mathrm{GRN}(\mathbf{X})_c = \gamma_c \frac{g_c}{\bar{g} + \epsilon}\mathbf{X}_c + \beta_c + \mathbf{X}_c, \tag{7.12}$$

   ここで $\epsilon = 10^{-6}$ であり、GRN が恒等写像として始まるよう γ, β は 0 に初期化される。GRN を数行のコードで実装し、層スケールの代わりに `ConvNeXtBlock` へ挿入して、Table 7.7 のロードマップのレシピのもとで検証正解率を比較せよ。

5. [code] **デプスワイズカーネルサイズのスイープ。** ●Table 7.7 のレシピのもとで、3×3、7×7、および 11×11 のデプスワイズカーネル（パディングはそれぞれ 1、3、5）を用いて `ConvNeXt` を訓練し、パラメータ数、積和演算数、エポックあたりの実時間、および検証正解率を比較せよ。7×7 から 11×11 にしたとき、エポックあたりの実時間が総積和演算数から予想されるよりもはるかに急激に増大するのはなぜか。

6. [code] **アーキテクチャとレシピの 2×2 アブレーション。** ●Table 7.7 の最初と最後の行は、現代的な AdamW・Mixup・平滑化のレシピのもとで `CompactResNet18` と `ModernConvNeXt` を比較している。7.6 節のレシピ A（モメンタム SGD、ステップ減衰、Mixup およびラベル平滑化なし、ピーク学習率は再調整）のもとで両方のアーキテクチャを訓練し、2×2 のアブレーション表を完成させよ。現代的な訓練レシピは両方のアーキテクチャを同程度に押し上げるか。

7.8 畳み込みネットワークの設計空間

本章で構築してきたネットワークの大半は人手で設計されており、各ステージで少数の構造的ハイパーパラメータを固定していた。すなわち、積み重ねるブロックの数、各ステージが出力するチャネル数、そしてどのブロックトポロジを使うかである。これらの数値はどこから来ており、そこには一般的な原理が存在するのだろうか。これまでのモデルを振り返っても、驚くほど手がかりは少ない。ステージごとのブロック数は、ResNet-18 では $(2,2,2,2)$、ResNet-50 では $(3,4,6,3)$、ConvNeXt-T では Swin から借用して 1 度検証されただけの $(3,3,9,3)$ である。さらに厄介なことに、これらのパラメータは密接に相互作用する。固定された計算予算（compute budget）

のもとでは、あるステージを深くすれば幅を狭くせざるを得ず、グループ畳み込み（grouped convolution）を使えば計算資源が浮いてチャネル幅を広げられる。網羅的な探索は問題外である。4 つのステージにわたってステージごとに 4 つの数値があるため、各数値にわずか 2 通りの選択肢を設けるだけでも $2^{16} = 65{,}536$ 通りの異なるネットワークが生じ、それぞれに個別の訓練を実行しなければならない。仮に力任せの探索で 1 つの優れたネットワークを見つけ出したとしても、単一の点推定（point estimate）からは**どの**設計上の選択が重要だったのかも、新しい計算予算や構成ブロックへどう適応させればよいのかもわからない。

ニューラルアーキテクチャ探索（neural architecture search、NAS）(Zoph and Le 2016; Liu et al. 2018) は EfficientNet（7.5 節）を生み出し、単一モデルの選択を自動化したものの、人手による設計と同様に再利用可能な科学的規則ではなく 1 つの点解を返すにとどまる。これに対し、Radosavovic et al. (2020) は根本的に異なる道をとった。共通のテンプレートと少数の制約（constraint）から生成されるネットワークの母集団、すなわち**設計空間**（design space）全体を比較するアプローチである。設計空間からネットワークをサンプリングして短時間だけ訓練し、提案した制約によって母集団全体の**誤差分布**（error distribution）が改善されるかを調べることで、単純で一般化可能な設計規則を抽出し、**RegNet** 族を定義した。以下では、この AnyNet テンプレートを実装し、ImageNet における母集団の実験結果（空間あたり 500 個のモデル）をたどったうえで、最も強力な規則を Fashion-MNIST の 2 つの小規模な母集団で自ら検証し、コンパクトな RegNet を ResNet-18 と直接対決させて訓練する。

7.8.1 AnyNet 設計空間

本章を通じて、現代の畳み込みネットワークは 7.2.3 節で導入したステム・ボディ・ヘッドの 3 部構成テンプレートへと収束してきた。**ステム**は 1 つまたは少数の畳み込みで生のピクセルを処理し、通常は直ちにダウンサンプリングする。**ボディ**は一連の**ステージ**を通じて計算の大半を担い、各ステージは固定された解像度で同一構造の**ブロック**を繰り返しながら、ステージの境界ごとに空間次元を半分にし、チャネル幅を広げる。そして**ヘッド**は、グローバル平均プーリングと線形層を介して最終的な特徴テンソルをクラスのロジットへと写像する。ResNet の 4 つのステージは入力解像度の $1/4$、$1/8$、$1/16$、$1/32$ で特徴を出力する。このマルチスケールなピラミッド構造が ConvNeXt に至るまで受け継がれてきた大きな理由は、後段の物体検出器やセグメンテーションヘッドが 4 つのスケールすべてを同時に利用するためである（32.5 節（英語版））。

7.8.1.1 人手で設計されたネットワークのステージパラメータ

Table 7.8 に、本章で扱った 4 つの代表的なアーキテクチャを並べて比較する。なお、どの行でもチャネル幅はステージが進むごとにきれいに 2 倍になっているのに対し、ステージの深さには明白な規則が見られない。

Table 7.8: 人手で設計された 4 つのネットワークにおけるステージの深さと幅（ImageNet 版）。

ネットワーク	ステージあたりのブロック数	ステージあたりの出力チャネル数	ブロック
ResNet-18（7.4 節）	2, 2, 2, 2	64, 128, 256, 512	2 つの 3×3 畳み込み
ResNet-50 (He et al. 2016a)	3, 4, 6, 3	256, 512, 1024, 2048	1×1、3×3、1×1 ボトルネック
ResNeXt-50（7.4.5.1 節）	3, 4, 6, 3	256, 512, 1024, 2048	32 グループのボトルネック
ConvNeXt-T（7.7 節）	3, 3, 9, 3	96, 192, 384, 768	7×7 デプスワイズ、逆ボトルネック

Radosavovic et al. (2020) による **AnyNet** 設計空間は、このマクロなテンプレートを固定したま

ま、ステージのハイパーパラメータを自由に残す（Figure 7.22）。ステムはストライド 2 の単一の 3×3 畳み込みで $c_0 = 32$ チャネルを出力し、その後にバッチ正規化（batch normalization）と ReLU が続く（ImageNet では RGB 画像を、Fashion-MNIST では単一のグレースケールチャネルを読み込む）。ボディは 4 つのステージからなり、ステージ i は d_i 個の ResNeXt ブロック（Figure 7.12）を積み重ねる。各ブロックでは、1×1 畳み込みが c_i チャネルを c_i/b_i へ射影し、3×3 グループ畳み込みが g_i チャネルずつのグループを処理し、2 つ目の 1×1 畳み込みが c_i チャネルへと戻す構造になっており、各畳み込みの後にバッチ正規化を備え、残差ショートカットを持つ（なお原論文の記法では、b_i は**ボトルネック比**（bottleneck ratio）を表すため、7.4.5.1 節で b と表記したボトルネック幅はここでは c_i/b_i となる）。各ステージの先頭ブロックはストライド 2 で空間解像度（spatial resolution）を半分にし、ストライド 2 の 1×1 畳み込みでショートカットを射影する。グローバル平均プーリングと線形分類器がヘッドを形成する。

したがって、各ステージには 4 つの自由度がある。深さ d_i、出力幅 c_i、ボトルネック比 $b_i \geq 1$、および**グループ幅**（group width）g_i である（グループあたりのチャネル数であり、**グループ数**（group count）g（7.4.5.1 節）と混同しないこと。groups 引数が d2l.ResNeXtBlock で表すのは引き続きこのグループ数である）。4 つのステージ全体では、これら 16 個の自由な数値（最大 16 までの深さ、8 刻みで最大 1024 までの幅、$\{1, 2, 4\}$ のボトルネック比、最大 32 までの 2 のべき乗のグループ幅をサンプリングする）によって、およそ $(16 \cdot 128 \cdot 3 \cdot 6)^4 \approx 2 \times 10^{18}$ 通りの候補アーキテクチャが張られる。すべてを列挙することは不可能だが、ランダムにサンプリングするのは容易である。224×224 の ImageNet 入力に対しては、ステムと 4 つのステージで空間解像度を 5 回半分にし、最終的に $224/2^5 = 7 \times 7$ となる。

7.8.1.2 AnyNet の実装

この空間をコードで探索するため、ステージのタプルで駆動される汎用的な AnyNet コンストラクタを構築する。まずは 3 層からなるステムから始めよう。

```
he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')

class AnyNet(d2l.Classifier):
    def __init__(self, arch, stem_channels, lr=0.1, num_classes=10,
                 in_channels=1, rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        rngs = nnx.Rngs(d2l.get_key()) if rngs is None else rngs
        self.net = self.create_net(in_channels, rngs)

    def stem(self, in_channels, num_channels, rngs):
        return nnx.Sequential(
            nnx.Conv(in_channels, num_channels, kernel_size=(3, 3),
                     strides=(2, 2), padding=(1, 1), use_bias=False,
                     kernel_init=he, rngs=rngs),
            nnx.BatchNorm(num_channels, momentum=0.9,
                          use_fast_variance=False, rngs=rngs), nnx.relu)
```

各ステージでは depth 個の d2l.ResNeXtBlock（7.4.5.1 節）を積み重ねる。各ブロックは出力幅 c_i、グループ数、および bot_mul $= 1/b_i$（したがってグループ幅は (c_i/b_i)/groups に等しい）によってパラメータ化され、先頭のブロックにのみストライド 2 を適用する。

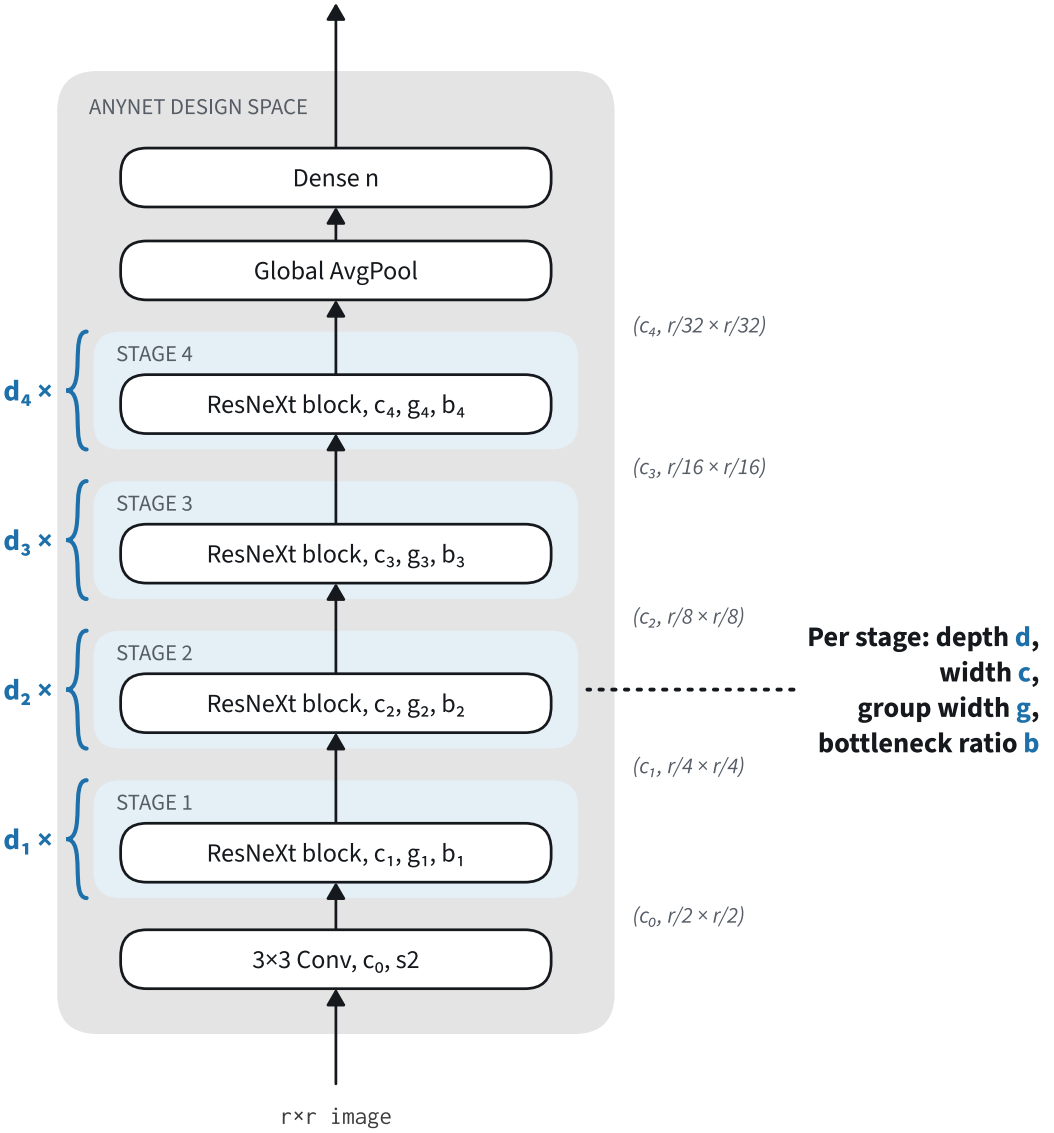


Figure 7.22: AnyNet 設計空間。ステム、4 ステージからなるボディ、およびヘッドで構成される。ステージ i は ResNeXt ブロックを d_i 回繰り返し、各ステージの最初のブロックが解像度を半分にする。図中の注記は $r \times r$ の入力に対する各パート後のチャネル数と解像度を示し、ヘッドは n 個のクラススコアを出力する。ステージごとの自由な選択肢は、深さ d_i、幅 c_i、グループ幅 g_i、ボトルネック比 b_i である。

```
@d2l.add_to_class(AnyNet)
def stage(self, depth, num_channels, groups, bot_mul, in_channels, rngs):
    blk = []
    for i in range(depth):
        if i == 0:
            blk.append(d2l.ResNeXtBlock(num_channels, groups, bot_mul,
                use_1x1conv=True, strides=(2, 2), in_channels=in_channels,
                rngs=rngs))
        else:
            blk.append(d2l.ResNeXtBlock(num_channels, groups, bot_mul,
                                        in_channels=num_channels, rngs=rngs))
    return nnx.Sequential(*blk)
```

ステム、(depth, num_channels, groups, bot_mul) の形式で arch の各要素ごとに作られるステージ、およびグローバルプーリングヘッドを組み立てれば AnyNet が完成する。7.4 節と同様に、すべての畳み込み層と線形層の重みには He 初期化（4.4.2.2 節）を用い、バッチ正規化の直前にある畳み込みではバイアスを省略する。

```
@d2l.add_to_class(AnyNet)
def create_net(self, in_channels, rngs):
    layers = [self.stem(in_channels, self.stem_channels, rngs)]
    stage_channels = self.stem_channels
    for s in self.arch:
        layers.append(self.stage(*s, stage_channels, rngs))
        stage_channels = s[1]
    layers.append(nnx.Sequential(
        global_avg_pool,
        nnx.Linear(stage_channels, self.num_classes, kernel_init=he,
                   rngs=rngs)))
    return nnx.Sequential(*layers)

def global_avg_pool(X):  # Average over height and width (NHWC)
    return X.mean(axis=(1, 2))
```

設計空間からサンプリングされたすべてのモデルをまったく同じ訓練プロトコルのもとで評価する必要があるため、オプティマイザを直接 AnyNet に組み込む。これは Radosavovic et al. (2020) が用いたプロトコル（および 7.6 節のレシピ A）に合わせたものであり、モメンタム 0.9 の SGD（7.5 節、第 9 章）、結合型重み減衰（coupled weight decay、2.7 節）、そして本章で共通して用いている d2l.cosine_warmup スケジュール（7.1 節、num_steps の最初の 10% で lr までウォームアップし、半コサイン曲線に沿って 0 まで減衰させる）からなる。原論文では ImageNet の 100 エポック訓練に対して重み減衰を 5×10^{-5} に設定しているが、ここでのより短い Fashion-MNIST の実行では 10^{-3} を用いる。この値は RegNetX32（7.8.3 節）に最適でありながら、ResNet18 でも同等に良く機能した。

```
@d2l.add_to_class(AnyNet)
def configure_optimizers(self):
    warmup = self.num_steps // 10
    schedule = lambda t: self.lr * d2l.cosine_warmup(
        t, self.num_steps, warmup)
    return optax.chain(optax.add_decayed_weights(1e-3),  # In the gradient
```

```
optax.sgd(schedule, momentum=0.9))
```

7.8.2 設計空間の比較

2 つの巨大な設計空間を、それぞれの最良アーキテクチャを 1 つも見つけることなく比較するにはどうすればよいだろうか。Radosavovic et al. (2019) は、短時間だけ訓練した適度な数のサンプルモデルで空間全体を忠実に代表させるため、次の 4 つの作業仮説を置いている。

1. **優れた設計原理は豊富に存在する**。妥当な設計規則を満たすネットワークの多くは高い性能を示す（干し草の山の中に良質な針が多数ある）ため、アーキテクチャ上の**分布**を最適化することには意味がある。
2. **短時間訓練のランキングは転移する**。短時間の訓練後の検証正解率は、収束後の最終的な正解率とよく相関する（これにより原論文での ImageNet 10 エポック訓練や、以下で行う Fashion-MNIST 1 エポック訓練が可能になる）。
3. **小規模での知見は外挿できる**。小さな計算予算で見つかった規則は、大規模なモデルへスケールアップしても引き続き成り立つ（最後に大規模環境で検証する）。
4. **設計因子はおおむね分離できる**。個々の構造的制約は、深刻な隠れた相互作用を伴わずに順次評価できる。

これら 4 つの仮説はいずれも先験的に保証されているわけではなく（特に 2 番目と 3 番目は強い仮説である）、末尾の演習問題では小規模環境でこれらを直接検証する。

これらの仮説を受け入れたうえで、ある空間からランダムに n 個のネットワークを抽出し、それぞれを短時間訓練して検証誤差（validation error）$e_1, \dots, e_n$ を記録する。**経験累積分布関数**（empirical cumulative distribution function、CDF）は、各誤差閾値 e に対して、誤差が高々 e に収まるサンプルモデルの割合を与える。

$$\hat{F}(e) = \frac{1}{n} \sum_{j=1}^{n} \mathbf{1}(e_j \leq e). \tag{7.13}$$

CDF の曲線が左上へ高く立ち上がるほど、その空間には正解率の高いネットワークが豊富に含まれている。ある空間の CDF が別の空間の CDF より一様に上側にあるとき、両空間における最良の単一モデルがたまたま一致していたとしても、最初の空間からランダムに抽出したモデルは任意の誤差閾値をより高い確率で下回る。

この視点を用いて、Radosavovic et al. (2020) は約 4 億回の積和演算（multiply-add、400 MFLOPs）で空間ごとに 500 個のネットワークをサンプリングし、それぞれを ImageNet 上で 10 エポック訓練して得られた CDF を Figure 7.23 にプロットしながら、制約なしの AnyNetX$_A$ 空間を 1 つずつ規則で絞り込んでいった。全 4 ステージで単一のボトルネック比 b を共有し（AnyNetX$_B$）、続いて単一のグループ幅 g も共有しても（AnyNetX$_C$）、誤差分布は変わらない（平均誤差は約 49%）。正解率を損なうことなく 16 個の自由なハイパーパラメータのうち 6 個を削減できるため、これ自体が大きな成果である。劇的な飛躍をもたらすのは**ステージ幅**への制約である。非減少の幅（$c_{i+1} \geq c_i$）を課すと母集団の平均誤差は丸 6 ポイント下がって約 43% になるのに対し、一定または縮小する幅では平均誤差がそれぞれ 56% と 63% へ悪化する。さらに非減少の深さ（$d_{i+1} \geq d_i$）を課すと平均誤差はもう 0.5 ポイント下がり、一定または縮小する深さは後れを取る。

経験則はデータセット、計算予算、ブロックの種類に依存するため、最も効果が大きかった「幅の増加対縮小」という単一の規則について、まったく同じ母集団手法を Fashion-MNIST に適用してみるのは有益である。32 チャネルのステムと ResNeXt ブロック（$b = 1$、グループ幅 16）を用いた

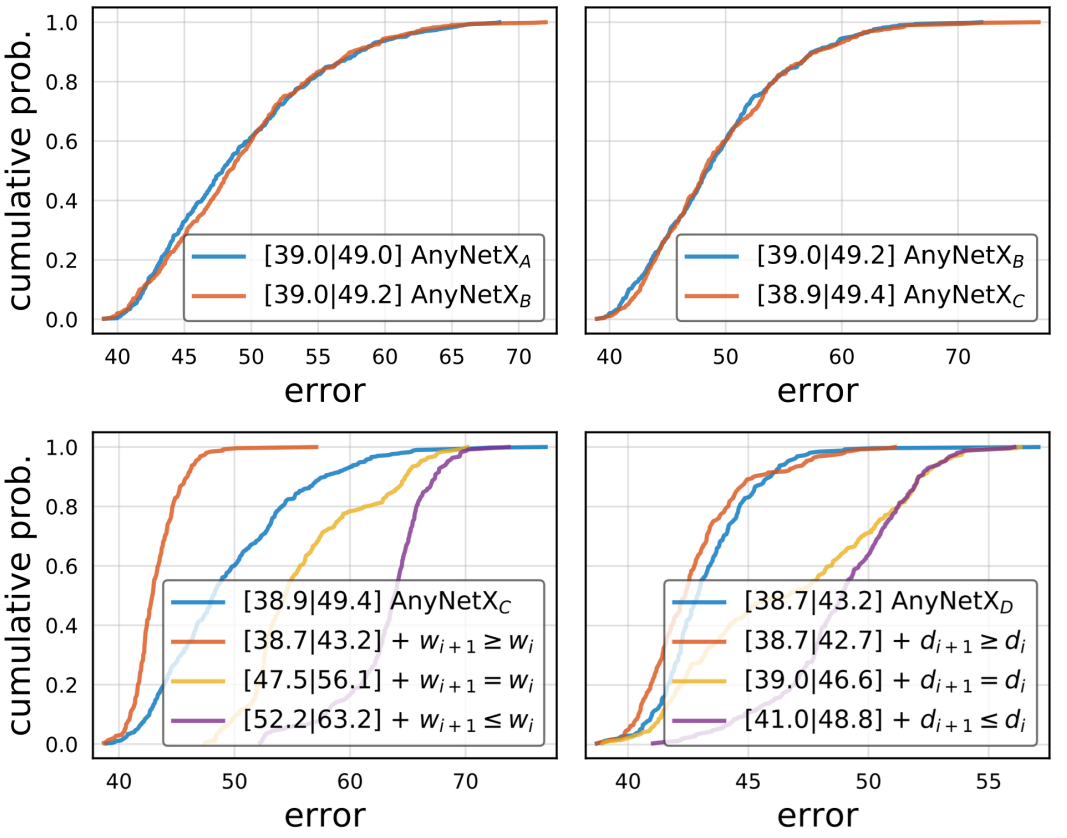


Figure 7.23: Radosavovic et al. (2020) より転載した設計空間の誤差 CDF。各空間から 500 個のネットワークをサンプリングし、それぞれ約 4 億回の積和演算のもと ImageNet 上で 10 エポック訓練したもの。凡例は top-1 誤差の最小値と平均値（パーセント表記）を示し、w_i は幅 c_i に対する原論文の記号である。上段：全ステージで単一のボトルネック比を共有する場合（AnyNetX$_B$）と制約なしの場合（AnyNetX$_A$）の比較、およびグループ幅も単一にした場合（AnyNetX$_C$）。下段：ステージ間で幅が増加・一定・減少する場合の比較、および幅が増加する空間（AnyNetX$_D$）において深さが同様に増加・一定・減少する場合の比較。

2 ステージの設計空間を 2 つ定義し、ステージ深さ $d_i \in \{1, \ldots, 8\}$ と幅 $c_i \in \{16, 32, \ldots, 256\}$ を、一方の空間では $c_1 < c_2$、もう一方の空間では $c_1 > c_2$ という制約のもとで抽出する。原論文と同様に、ステム、すべてのブロック、およびすべての射影ショートカットにわたる積和演算を集計して計算予算を揃える（7.1 節）。

```
def mult_adds(arch, r, stem_channels=32, group_width=16, num_classes=10):
    """Multiply-adds of an AnyNet with k = 1 for one r x r image; arch holds
    (depth, width) for each stage."""
    r //= 2  # The stem halves the resolution
    total, c_in = r * r * 9 * stem_channels, stem_channels  # Grayscale input
    for d, c in arch:
        total += r * r * c_in * c  # First 1 x 1 convolution, before the stride
        r //= 2
        total += r * r * c_in * c  # 1 x 1 convolution of the shortcut
        total += r * r * c * c * (2 * d - 1)  # The other 1 x 1 convolutions
        total += r * r * 9 * group_width * c * d  # Grouped 3 x 3 convolutions
        c_in = c
    return total + c_in * num_classes  # Linear head
```

候補モデルを抽出するには、2 つの深さと互いに異なる 2 つの幅をサンプリングし、空間の制約に従って幅を並べ替え、96×96 画像 1 枚あたりの積和演算が予算である 112M 回の 20% 以内に収まる場合に限り採択する（この予算は 7.8.3 節で訓練する `RegNetX32` モデルに合わせたものである）。

```
def sample_arch(rng, increasing, r=96, budget=112e6):
    """Depths and widths of two stages at about `budget` multiply-adds."""
    while True:
        depths = rng.integers(1, 9, size=2).tolist()
        widths = sorted(rng.choice(np.arange(16, 257, 16), 2, replace=False)
                        .tolist(), reverse=not increasing)
        arch = tuple(zip(depths, widths))
        if abs(mult_adds(arch, r) / budget - 1) <= 0.2:
            return arch

rng = np.random.default_rng()
for increasing in (True, False):
    arch = sample_arch(rng, increasing)
    print(arch, f'{mult_adds(arch, 96) / 1e6:.0f} million multiply-adds')
```

```
((6, 96), (2, 112)) 133 million multiply-adds
((3, 128), (6, 48)) 106 million multiply-adds
```

計算量を揃えた条件では、幅が減少するモデルは初期の高解像度な特徴マップにチャネルを費やし、出力側に向かうにつれて幅を狭めていく。以下では、サンプリングした各アーキテクチャを `AnyNet` のオプティマイザを用いて 1 エポック（7.8.3 節における本訓練の 10 分の 1）だけ訓練し、その検証誤差を記録する。

```
def population_errors(rng, increasing, data, n, num_epochs=1):
    """Validation errors of n networks sampled from one design space."""
    errors = []
    for _ in range(n):
```

```
            arch = sample_arch(rng, increasing)
            model = AnyNet(tuple((d, c, c // 16, 1) for d, c in arch), 32)
            model.num_steps = num_epochs * len(data.train_dataloader())
            trainer = d2l.Trainer(max_epochs=num_epochs, num_gpus=1)
            trainer.board.display = False  # We plot the errors ourselves
            trainer.fit(model, data)
            errors.append(1 - float(trainer.history['val_acc'][-1]))
        return np.array(errors)
```

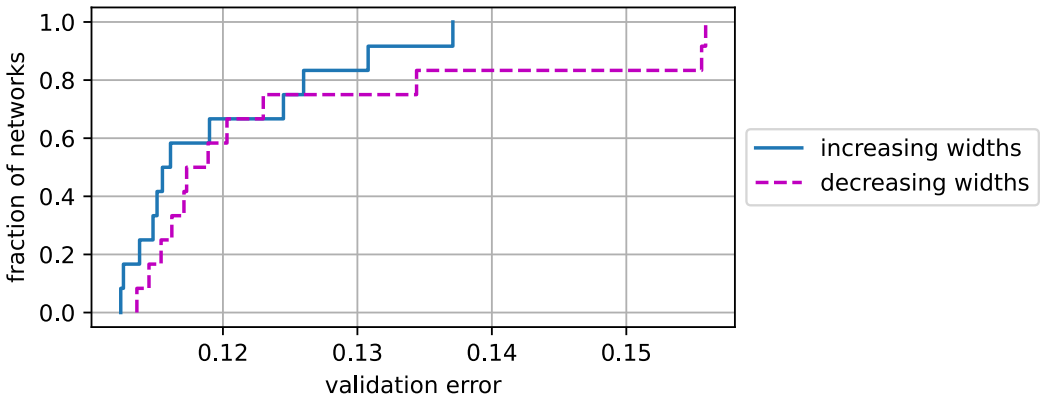
空間ごとに 12 個のモデルを抽出して訓練する（合計 24 回の 1 エポック訓練となる）。

```
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
spaces = {'increasing widths': True, 'decreasing widths': False}
errors = {name: population_errors(rng, increasing, data, 12)
          for name, increasing in spaces.items()}
```

経験 CDF Equation 7.13 をプロットすると、ソートされた各検証誤差の位置で $1/n$ ずつ跳び上がる階段関数が得られる。

```
def ecdf(e):
    """Corners of the step function of the empirical CDF of e."""
    e = np.sort(e)
    return np.repeat(e, 2), np.repeat(np.arange(len(e) + 1) / len(e), 2)[1:-1]

X, Y = zip(*(ecdf(e) for e in errors.values()))
d2l.plot(list(X), list(Y), xlabel='validation error',
         ylabel='fraction of networks', legend=list(errors),
         legend_loc='outside', figsize=(5, 2.5))
for name, e in errors.items():
    print(f'{name}: mean error {e.mean():.3f}, minimum {e.min():.3f}')
```

```
increasing widths: mean error 0.120, minimum 0.112
decreasing widths: mean error 0.125, minimum 0.114
```



ここでは 2 つの CDF は大きく重なっている。繰り返し抽出を行っても、平均誤差は 12% 付近にとどまって差は 1 パーセントポイント未満であり（多くの場合、幅が増加する空間がわずかに優位となる）、最良モデルはいずれの空間でも 11% 前後に着地する。ImageNet では平均誤差にし

て丸 20 ポイントもの価値があった制約が、Fashion-MNIST では 1 ポイント未満へと縮小しており、空間あたり 12 サンプルでは明確に分離できないほど小さい。これは冷静な見方を促すと同時に示唆に富む結果である。まともな畳み込みネットワークならほぼどれでも 1 エポックで約 88% に達する簡単な 10 クラスのデータセットでは、アーキテクチャ上の制約が母集団を引き離す余地はほとんどない。設計規則は決して普遍的な法則ではなく、その効果の大きさは発見元のタスクの複雑さに直接依存するのである。

7.8.3 RegNet

7.8.3.1 設計原理

ImageNet の検証を勝ち残った制約を組み合わせることで、コンパクトな AnyNetX$_E$ 設計空間が定義される。

- 全ステージにわたって単一のボトルネック比 b を共有する。
- 全ステージにわたって単一のグループ幅 g を共有する。
- ステージ幅を非減少とする（$c_i \leq c_{i+1}$）。
- ステージ深さを非減少とする（$d_i \leq d_{i+1}$）。

AnyNetX$_E$ 内で上位の性能を示すネットワークを詳しく調べると、さらに際立ったパターンがもう 1 つ浮かび上がる。入力側から出力側へ向かってブロックに $j = 0, 1, \ldots$ と順に通し番号を振ると、最適なブロック幅は次のような単純な線形ランプに従う。

$$u_j = u_0 + aj, \tag{7.14}$$

ここで、初期幅 u_0 と傾き $a > 0$ がそのパラメータである。1 つのステージ内のすべてのブロックは同じ幅を共有するため、RegNet は Table 7.8 に示した人手設計ネットワークの倍々になる幅と同様に、各 u_j を等比グリッド上へ量子化（quantization）し、同じ量子化幅に丸められた連続するブロックを 1 つのステージにまとめてきれいな階段構造を作る。Figure 7.24 は RegNet の 1 例における階段構造を示しており、問題 2 では別の例を実際に計算する。興味深いことに、著者らは $b = 1$（内部でのボトルネック圧縮を一切行わない構成）が ResNet-50 や ResNeXt-50 の $b = 4$ のボトルネックを一貫して上回ることも発見した。量子化線形幅と $b = 1$ を組み合わせることで **RegNetX** 族が定義され、16 個の恣意的なステージ数値は計算予算ごとにわずか数個の解釈可能なパラメータへと集約される。

7.8.3.2 Fashion-MNIST における RegNetX32

それでは、これらの規則に従うコンパクトな RegNet 風ネットワークを構築し、ResNet-18 と直接比較してみよう。アップサンプリングされた 96×96 の Fashion-MNIST 画像はもともと 28×28 の縮小画像であるため、4 ステージ（5 回の半減）を実行すると最終盤のステージは 6×6 や 3×3 という極小のグリッドに縮んでしまう。そこで、ここでは 24×24 と 12×12 で動作する 2 ステージ構成を用いる（問題 1 で 4 ステージ構成を検証する）。32 チャネルのステムの後に、増加する深さと幅（ステージ 1 は $d_1 = 4$、$c_1 = 64$、ステージ 2 は $d_2 = 6$、$c_2 = 160$）を人手で選び、$b = 1$ および共通のグループ幅 $g = 16$（グループ数はそれぞれ $64/16 = 4$ と $160/16 = 10$）と組み合わせる。ステムの畳み込み 1 層、3 層構成のブロック 10 個、線形分類器 1 層で合計 32 層となり、これが `RegNetX32` である。

```
class RegNetX32(AnyNet):
    def __init__(self, lr=0.1, num_classes=10, num_steps=None,
                 in_channels=1, rngs=None):
```

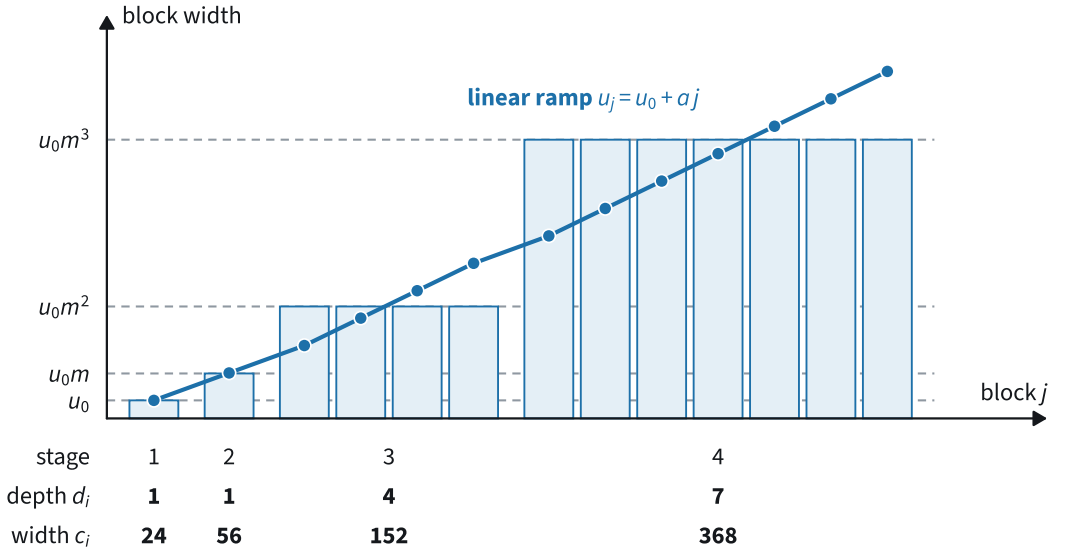


Figure 7.24: 13 ブロックからなるネットワーク RegNetX-200MF における量子化線形幅。線形ランプ $u_j = u_0 + aj$ 上の各ブロックの幅（点）は等比グリッド $u_0, u_0m, u_0m^2, \ldots$（破線）の最も近い水準へ丸められ、同じ水準に属する連続したブロックが 1 つのステージを形成する。$u_0 = 24$、$a = 36.44$、$m = 2.49$ とすると深さ 1、1、4、7 の 4 つのステージが得られ、8 の倍数へ丸めた後の各ステージの幅は 24、56、152、368 チャネルとなる。

```
        stem_channels, group_width, bot_mul = 32, 16, 1
        depths, channels = (4, 6), (64, 160)
        super().__init__(  # The number of groups is c_i / g
            tuple((d, c, c // group_width, bot_mul)
                  for d, c in zip(depths, channels)),
            stem_channels, lr, num_classes, in_channels, rngs)
        self.num_steps = num_steps  # Length of the learning-rate schedule
```

`layer_summary` を確認すると、チャネル数が増えるにつれて空間解像度が $96 \to 48 \to 24 \to 12$ ときれいに段階的に縮小していくことがわかる。

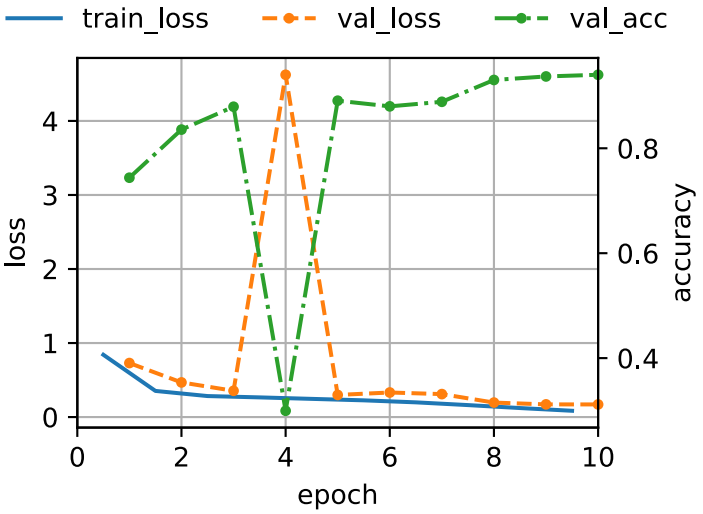
```
RegNetX32().layer_summary((1, 96, 96, 1))
```

```
Sequential            (1, 48, 48, 32)
Sequential            (1, 24, 24, 64)
Sequential            (1, 12, 12, 160)
Sequential            (1, 10)
```

7.1 節と同様に演算数を集計すると、`RegNetX32` は 96×96 画像 1 枚あたり 1 億 1200 万回の積和演算を実行する。これは ResNet-18 の 3 億 1900 万回と比べてほぼ 3 分の 1 の少なさである。

`RegNetX32` を 96×96 の Fashion-MNIST 上で、バッチサイズ 128、`AnyNet` のモメンタム付き SGD オプティマイザ、ピーク学習率 0.1（1 エポックのウォームアップ付き）の設定により 10 エポック訓練しよう。

```
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
model = RegNetX32(lr=0.1, num_steps=10 * len(data.train_dataloader()))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
```



10 エポックの訓練が完了したら、`d2l.num_params(model)` を呼び出して `RegNetX32` の総パラメータ数（parameter count）を最終検証正解率 `trainer.history["val_acc"][-1]` とともに表示する。

```
print(f'{d2l.num_params(model):,} parameters, validation accuracy '
      f'{trainer.history["val_acc"][-1]:.3f}')
```

```
519,658 parameters, validation accuracy 0.940
```

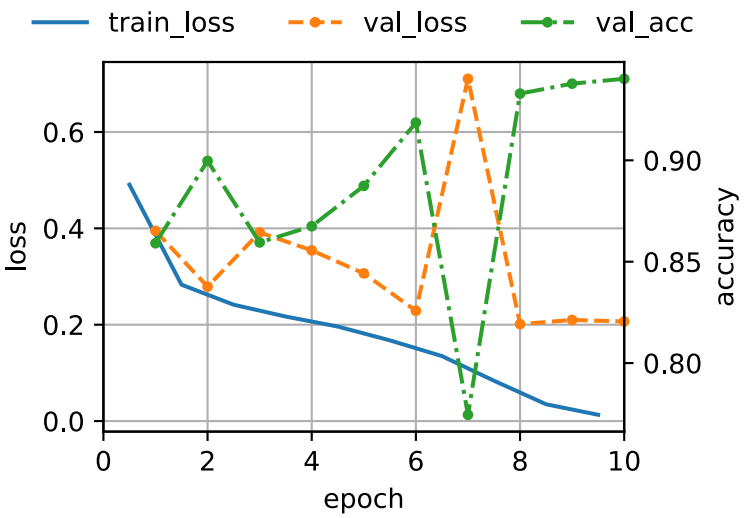
比較条件を厳密に揃えるため、ResNet-18 もまったく同じ `AnyNet` オプティマイザのもとで訓練する。ResNet-18 は同じステム・ボディ・ヘッド構造を共有しているため、単に `AnyNet` をサブクラス化し、`stem` と `stage` を `d2l.Residual`（7.4 節）でオーバーライドして各ステージに `(depth, width, stride)` を渡すだけでよい。

```
class ResNet18(AnyNet):
    def __init__(self, lr=0.1, num_classes=10, num_steps=None,
                 in_channels=1, rngs=None):
        super().__init__(((2, 64, 1), (2, 128, 2), (2, 256, 2), (2, 512, 2)),
                         64, lr, num_classes, in_channels, rngs)
        self.num_steps = num_steps

    def stem(self, in_channels, num_channels, rngs):
        return nnx.Sequential(
            nnx.Conv(in_channels, num_channels, kernel_size=(7, 7),
                     strides=(2, 2), padding=3, use_bias=False,
                     kernel_init=he, rngs=rngs),
            nnx.BatchNorm(num_channels, momentum=0.9,
```

```
                           use_fast_variance=False, rngs=rngs), nnx.relu,
            max_pool)

    def stage(self, depth, num_channels, stride, in_channels, rngs):
        blk = [d2l.Residual(num_channels, use_1x1conv=stride > 1,
                            strides=(stride, stride), in_channels=in_channels,
                            rngs=rngs)]
        blk += [d2l.Residual(num_channels, in_channels=num_channels,
                             rngs=rngs) for _ in range(depth - 1)]
        return nnx.Sequential(*blk)

def max_pool(X):  # 3x3 windows, stride 2, padding 1 on every side
    return nnx.max_pool(X, window_shape=(3, 3), strides=(2, 2),
                        padding=((1, 1), (1, 1)))
```

それでは、まったく同じデータ上で ResNet18 を 10 エポック訓練する。

```
model = ResNet18(lr=0.1, num_steps=10 * len(data.train_dataloader()))
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
```



先ほどと同様に d2l.num_params(model) と最終検証正解率 trainer.history["val_acc"][-1] を出力し、ResNet18 を RegNetX32 と直接比較できるようにする。

```
print(f'{d2l.num_params(model):,} parameters, validation accuracy '
      f'{trainer.history["val_acc"][-1]:.3f}')
```

```
11,175,370 parameters, validation accuracy 0.940
```

どちらのアーキテクチャも検証正解率は 94% 付近（複数回の実行で 93.7〜94.2%）に達するが、パラメータ数は ResNet-18 の 11,175,370 個に対して RegNetX32 はわずか 519,658 個であり、重み数は 5% 未満、積和演算も 3 分の 1（112M 対 319M）にすぎない。この結果を正しく理解する

には、次の 3 つの点に留意する必要がある。

1. **アーキテクチャとオプティマイザの結合：**モメンタムも重み減衰も使わない素の SGD（7.4 節）では、ResNet-18 が約 93% に達するのに対し `RegNetX32` は約 91% まで低下する。`RegNetX32` が肩を並べるのは、RegNet の設計空間が調整された前提であるモメンタム＋重み減衰のプロトコルのもとでのみである。
2. **データセットの上限：**4 ステージの RegNet（RegNetX-400MF、-800MF、-1.6GF、パラメータ数 4.8〜8.3M）へとスケールアップしても、Fashion-MNIST では同じ 93〜94% の上限で頭打ちになる。
3. **演算数対 GPU レイテンシ：**`RegNetX32` の 1 エポックには、積和演算が 3 分の 1 であるにもかかわらず ResNet-18 の約 $1.5\times$ の時間がかかる。積和演算が数えるのは算術演算の回数であって実行時間ではない（7.5 節）。`RegNetX32` は深さがほぼ 2 倍あり、幅の狭い多数のグループ畳み込み層が画像 1 枚あたり約 $2.5\times$ の活性化値（activation）を書き出すため、算術演算ではなくメモリ転送が律速となる。RegNet の著者らも ImageNet で同じ現象を確認している (Radosavovic et al. 2020)。すなわち、GPU の実行時間は積和演算よりも活性化値の数に強く連動し、活性化値の数が等しい ResNet と比較すると RegNetX モデルはほぼ同等の速度で訓練・実行でき、同程度の積和演算で比較すれば（積和演算に比例して活性化値が増える）EfficientNet より最大 $5\times$ 高速に動作する。

7.8.4 まとめ

設計空間を比較するアプローチは、アーキテクチャ分析の対象を単一の点推定から、パラメータ化されたテンプレート（`AnyNet`）よりサンプリングした母集団上の誤差分布へと転換させる。空間あたり 500 個の ImageNet サンプルを短時間訓練した検証では、ステージ間で単一のボトルネック比 b とグループ幅 g を共有しても性能は一切損なわれない一方、ステージの幅（階段状に量子化された $u_j = u_0 + aj$）と深さに非減少の制約を課すことで母集団の平均誤差が 49% から 43% へと下がり、RegNetX 族が導かれる。Fashion-MNIST において、`RegNetX32` は ResNet-18 の 5% 未満のパラメータ数と 3 分の 1 の積和演算で、その約 94% という正解率に匹敵する。本章全体を通じて見てきたように、少数の基本的な構造プリミティブ（反復可能なブロック、活性化値の正規化、残差ショートカット）と、アーキテクチャによる改善を訓練レシピ（training recipe）による改善から切り分ける規律こそが、現代の視覚バックボーンの礎をなしている。

7.8.5 演習問題

1. [code] **予算を揃えた 4 ステージ構成。**`RegNetX32` は 96×96 の Fashion-MNIST に対して 2 つのステージを用いている。ステージ間で幅と深さを非減少に保ち、かつ総パラメータ数を `RegNetX32` の 10% 以内に収めつつ、これを Figure 7.22 の 4 ステージ構成へ拡張せよ。検証正解率と 1 エポックあたりの実行時間を `RegNetX32` と比較し、4 つの各ステージが動作する特徴マップの空間解像度を述べよ。

2. **量子化線形幅。**@Radosavovic.Kosaraju.Girshick.ea.2020 は、d ブロックからなるネットワークのブロック幅を線形ランプ $u_j = u_0 + a\,j$（$j = 0, \dots, d-1$、7.8.3 節）でパラメータ化し、次の式によって公比 m の等比グリッドへ各ブロックを丸める。

$$s_j = \operatorname{round}\left(\frac{\log(u_j/u_0)}{\log m}\right), \qquad \tilde{u}_j = u_0\, m^{s_j}, \tag{7.15}$$

   そして $\tilde{u}_j$ が等しいブロックをまとめて単一のステージとする。

   1. Figure 7.24 には RegNetX-200MF の結果を示した。$d = 22$、$u_0 = 24$、$a = 24.48$、$m = 2.54$（原論文の記号では w_0, w_a, w_m）を持つ RegNetX-400MF について同じ

計算を行い、ステージ数、各ステージの深さ d_i、および最も近い 8 の倍数に丸めたステージ幅を求めよ。

2. ステージ数と階段近似 $\tilde{u}_j \approx u_j$ は、$m \to 1$ のときと m が大きくなるときとでそれぞれどのように振る舞うか。
3. (a) で丸めたステージ幅のうち、共通のグループ幅 $g = 16$ で割り切れないものはどれか。また、それらはどのように調整すべきか。

3. [code] **RegNetY（Squeeze-and-Excitation の追加）**。RegNetY は Squeeze-and-Excitation ゲート `SE`（7.5.2.3 節、縮小比 $r = 4$）を、各 `ResNeXtBlock` 内の 3×3 グループ畳み込みの直後に挿入する (Radosavovic et al. 2020)。

   1. `RegNetY32` を `RegNetX32` と同じステージ構成で構築せよ。式 $2c^2/r$ を用いて追加されるパラメータ数を予測し、`d2l.num_params` と照合して検証せよ。
   2. `RegNetY32` を 96×96 の Fashion-MNIST 上で訓練し、検証正解率と 1 エポックあたりの実行時間を `RegNetX32` と比較せよ。積和演算の増加がほぼゼロであるにもかかわらず、`SE` によって無視できない実時間が増えるのはなぜか。

4. [code] **RegNet の規則を破る**。●パラメータ数をほぼ揃えたまま `RegNetX32` の派生モデルを 3 つ実装し、それぞれが AnyNetX$_E$ の規則（7.8.3 節）を 1 つずつ破るようにせよ。すなわち、(i) 減少するステージ深さ（$d_1 > d_2$）、(ii) ステージ間で不均一なグループ幅、(iii) ステージ間で異なるボトルネック圧縮 $b_i > 1$ である。各派生モデルを Fashion-MNIST 上で 3 回ずつ訓練して `RegNetX32` と比較し、得られた結果を Figure 7.23 の母集団 CDF と対比せよ。規則違反ごとに人手で選んだ単一のネットワークを検証するだけでは、2 つの経験 CDF を比較する場合よりも証拠として弱いのはなぜか。

5. [extended] **ConvNeXt の設計空間**。●`ConvNeXtBlock`（7.7 節、拡大比 $t \in \{2, 4\}$ を引数として追加）と `ConvNeXt` のステムおよびダウンサンプリング層を用いて 2 ステージの AnyNet 設計空間を構築し、ステージ深さ $d_i \in \{1, 2, 4\}$ と幅 $c_i \in \{32, 64, 128\}$ をサンプリングできるようにせよ。

   1. AnyNet におけるステージごとの 4 つの自由度（d_i, c_i, g_i, b_i）のうち、`ConvNeXtBlock` に直接対応するものがあるのはどれで、デプスワイズ分離構造によって固定されるのはどれか。
   2. 制約なしの空間から 16 個、拡大比を共有した空間（$t_1 = t_2$）から 16 個、さらに $c_1 \le c_2$ かつ $d_1 \le d_2$ を課した空間から 16 個のネットワークをサンプリングせよ。それぞれを 96×96 の Fashion-MNIST 上で 2 エポック訓練し、3 つの経験誤差 CDF Equation 7.13 をプロットして、RegNet のどの設計原理が ConvNeXt ブロックにも引き継がれるかを確認せよ。

Chapter 8

系列モデル

ここまで 2 章にわたって画像を扱ってきたが、本書のこのパートの締めくくりとして、ここからは系列データに目を向けよう。これまでに構築してきたすべてのモデルは、属性の表形式ベクトルや 28×28 のピクセル格子のように固定された形状の入力を前提とし、各例を独立に扱っていた。しかし、文、音声録音、動画、あるいはセンサ計測値のストリームは、それとはまったく異なる振る舞いをする。すなわち、長さは例ごとに異なり、各要素は互いに対して、また到着する順序に対して強く依存している。本章全体を通して共通のデータセットとして用いる H. G. Wells の『The Time Machine』の冒頭の一文、「The Time Traveller (for so it will be convenient to speak of him) was expounding a recondite matter to us.」（タイム・トラベラー、便宜上彼をそう呼ぶことにするが、彼は難解な事柄を我々に説いていた）を考えてみよう。最後の単語を言い当てるには、その前にある 19 語を順番に読む必要がある。先行するテキストが任意に長くなりうるとき、単一のモデルでどのようにしてその予測ができるだろうか。

Claude Shannon はこの問いを推測ゲームとして捉え (Shannon 1951)、人間の読者に一節を 1 文字ずつ推測させることで、印刷された英語の予測可能性を推定した。**言語モデル**（language model）も同じゲームをプレイする。すなわち、先行するトークンが与えられた下で次の**トークン**（文字、サブワード、単語全体のいずれであってもよい）に確率を割り当て、選ばれた各トークンを次の入力として戻すことで新しいテキストを生成できる。

すでに読み込んだ履歴はステップごとに長くなるため、直近のトークンからなる固定長のウィンドウだけに着目すると、それ以前の情報はすべて捨てられてしまう。そこで**リカレントニューラルネットワーク**（RNN）は、これまでに読み込んだ系列を要約する固定サイズの**隠れ状態**（hidden state）を保持し、トークンを 1 つ処理するたびに同じパラメータでその状態を更新する。現代の大規模言語モデル（large language model）が再帰構造をアテンション（第 10 章）や構造化状態空間モデル（第 12 章）に置き換えている場面でも、サブワードトークン化（subword tokenization）、パープレキシティとバイトあたりビット数（bits per byte）による評価、自己回帰デコーディング（autoregressive decoding）といった本章の中核的な仕組みは、そのまま変わらず引き継がれる。

まず、自己回帰型の系列予測を定式化し、多ステップのロールアウトが 1 ステップ先の予測よりもはるかに速く劣化する理由を確認する（8.1 節）。次に、バイトレベルのバイトペアエンコーディングによって生のテキストをサブワードトークン系列へ変換し（8.2 節）、パープレキシティおよびバイトあたりビット数の評価指標とあわせてカウントベースの n-gram 言語モデルを構築する（8.3 節）。続いて、リカレントニューラルネットワーク言語モデルを定式化してゼロからの実装と高レベル API による実装の両方で構築し（8.4 節）、勾配クリッピング（gradient clipping）によって訓練を安定化させたうえで、学習された表現を Kneser–Ney n-gram と比較する。最後に、

8.5 節では通時的誤差逆伝播法（backpropagation through time）を導出して素朴な RNN の勾配が消失または爆発する理由を説明し、8.6 節ではテキストを生成するための貪欲探索（greedy search）、ビームサーチ、および打ち切りサンプリング手法を検討する。

8.1 系列の扱い

メールクライアントは「Thank you for your」という書き出しを目にすると、その続きとして「help」や「time」といった語を自然に提案する。同様に、1 時間ごとに気温を記録する気象サービスは、直近の観測値をもとに次の値を予測する。いずれの状況でも、良い予測を行うには、それ以前に何が現れたかだけでなく、どの順序で現れたかが重要になる。単語を「your thank for you」と並べ替えれば続く語はまったく変わってしまうし、下降中の気温曲線と上昇中の気温曲線とでは求められる予測が大きく異なる。これまでの章で扱ったモデルは、この構造を直接捉えることができない。固定された形状の単一の入力（ベクトル $\mathbf{x} \in \mathbb{R}^d$）を出力へと写像し、各例を他の例とは独立なものとして扱うからである。文、音声録音、あるいはセンサ計測値のストリームは、この両方の仮定に反している。長さがインスタンスごとに異なり、しかも内部の要素同士が強く依存し合っているためである。

本節では、系列モデリング（sequence modeling）を既知の問題である回帰へと帰着させる。各要素をそれに先行する要素から予測するにあたり、直近の観測値からなるウィンドウを保持するか、それまでに観測したすべてについての逐次的な要約を維持することで、増え続ける履歴を固定サイズの入力へと縮約する。厳密な正解（ground truth）がわかっている設定でこれらの考え方を検証するため、ノイズを含む正弦波に対して固定ウィンドウ上の線形な自己回帰モデル（autoregressive model）を当てはめる。のちに見るように、1 ステップ先を予測するのは容易である一方、モデル自身の予測を入力に戻して何ステップも先を予測するのは格段に難しい。本章の残りの節では、これと同じ原理をテキストへと引き継いでいく。

8.1.1 系列データ

系列（sequence）とは、特徴ベクトル $\mathbf{x}_1, \ldots, \mathbf{x}_T$（スカラーの場合は値 $x_1, \ldots, x_T$）の順序付きリストである。ここで添字 $t \in \mathbb{Z}^+$ は**タイムステップ**を表し、長さ T は系列ごとに異なってよい。身近な例は枚挙にいとまがない。テキストは文字、サブワード、または単語の系列であり、音声（発話であれ音楽であれ）は波形サンプルやスペクトログラムフレームの系列であり、株価、センサの計測値、患者のバイタルサインといった時系列（time series）では測定値が 1 つずつ順番に届く。

文書全体や入院期間全体といった系列**全体**が、系列上の単一の基底分布から独立に抽出されるという仮定は引き続き置くものの、系列**内部**の要素が互いに独立であると仮定することは到底できない。入院 10 日目に患者へ投与される薬剤は、それまでの 9 日間に起きた経過に大きく依存する。この内部の依存性は厄介どころか、まさに過去を有益な情報源にしている要因そのものである。もし連続する要素が独立であれば、過去の観測値で条件付けても次の要素について何もわからないだろう。また、系列内のすべての位置が同じ周辺分布（marginal distribution）を共有すると仮定する必要もない。文書の結びの段落は書き出しの一文とは大きく異なるのが普通であり、患者の容態も治療の経過とともに回復や悪化へと変化していく。

系列予測のタスクは、いくつかの異なる形で現れる。系列全体に単一のラベルを割り当てる問題（映画レビューの感情分類など）、固定された入力から系列を生成する問題（画像のキャプション生成など）、あるいは入力系列を出力系列へ写像する問題（文の翻訳や音声の書き起こしなど）がある。しかし、いかなる教師ありの目的値を扱うにせよ、まずは同時確率（joint probability）$P(x_1, \ldots, x_T)$ を推定することで系列そのものをモデル化してみるのが有益である。各要素がそれに先行する要素に対する目的値として自然に機能するため、この分布の推定には外部ラベルが一切必要ない。さらに、系列に確率を割り当てるモデルは、新しい系列を生成できるだけでなく、

競合する候補を順位付けすることもできる。たとえば、同じ音声クリップに対する 2 つの書き起こしのうち、もっともらしい方を選択できる。

8.1.2 自己回帰モデル

株価指数や 1 時間ごとの気温記録のような数値系列を考えよう。次の測定値に対してどう行動するかを判断する予測者にとって最大の関心事は、これまでに得られたすべての観測値が与えられたもとでの次の値の条件付き分布（conditional distribution）である。

$$P(x_t \mid x_1, \ldots, x_{t-1}). \tag{8.1}$$

プレフィックス $x_1, \ldots, x_{t-1}$ は本章全体を通して登場するため、これを $x_{<t}$ と略記する。条件付き分布全体を推定するのが難しい場合でも、その条件付き期待値（conditional expectation）$\mathbb{E}[x_t \mid x_{<t}]$ だけで十分に有用な要約となり、線形回帰（2.1 節）での当てはめを試みることができる。信号をそれ自身の過去の値に回帰させるモデルは、特徴量と目的値の両方が同じ系列から得られることから、**自己回帰モデル**と呼ばれる。

しかし、ここでただちに問題が生じる。$x_{<t}$ に含まれる条件付け入力の個数は t とともに増大するため、タイムステップごとに特徴量の数が異なり、固定入力のモデルへそのまま渡すことができないのである。この不一致は、履歴を固定サイズの表現へと圧縮する 2 つの一般的な戦略によって解決できる（Figure 8.1）。どちらも自己回帰モデルであることに変わりはなく、過去の情報をどれだけ保持するかという点だけが異なる。

8.1.2.1 固定ウィンドウ

第 1 の戦略は、直近 m 個の観測値 $x_{t-m}, \ldots, x_{t-1}$ のみで条件付け、それ以前の履歴をすべて破棄する方法である。すべての $t > m$ に対して各訓練例はちょうど m 個の特徴量を持つため、線形モデルであれ深層モデルであれ、任意の固定入力アーキテクチャをそのまま適用できる。これは自己回帰の古典的な形式であり、m は**ウィンドウ長**（window length）と呼ばれる。固定ウィンドウは実装が単純であり、各予測が系列の局所的なスライスのみに依存するためタイムステップ間で並列化しやすい。その欠点は明確なホライズンの限界があることで、m ステップより前の過去はモデルからまったく見えなくなる。

8.1.2.2 潜在要約

第 2 の戦略は、固定サイズの逐次的な要約 $\mathbf{h}_t$ によってすべての観測値 $x_1, \ldots, x_t$ の情報を保持し、新しい値が届くたびに $\mathbf{h}_t = g(\mathbf{h}_{t-1}, x_t)$ によって再帰的に更新しながら、その要約から $\hat{x}_{t+1} = f(\mathbf{h}_t)$ によって次の要素を予測する方法である。$\mathbf{h}_t$ は直接観測されないため、このようなモデルは**潜在自己回帰モデル**（latent autoregressive model）と呼ばれる。状態ベクトル $\mathbf{h}_t$ が占めるメモリ量は系列がどれほど長くなっても一定であり、任意の長さの履歴を有界な空間内に要約できる。リカレントニューラルネットワーク（8.4 節）は学習される遷移関数 g を備えた潜在自己回帰モデルであり、第 12 章の状態空間モデル（state space model）も同様である。隠れ状態（hidden state）に何を記憶させ、どのように安定して更新するかが、それらの章における中心的な設計上の問いとなる。

8.1.2.3 条件付き分布から系列へ

どちらの表現を選ぶにせよ、1 ステップの条件付き分布によって系列全体にわたる確率モデルが完全に定まる。**確率の連鎖律**（chain rule of probability）により、任意の同時分布（joint distribution）は、タイムステップごとに 1 つずつ、それ以前のすべての要素で条件付けられた条件付き確率の積へと分解される。

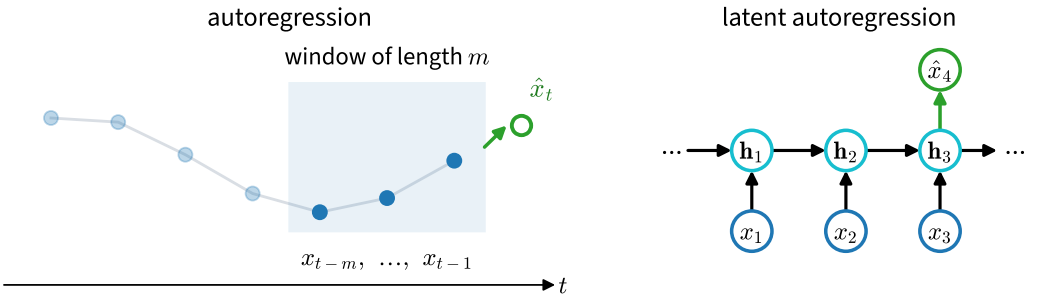


Figure 8.1: 系列モデルに固定サイズの入力を与える 2 つの方法。（左）固定ウィンドウは直近 m 個の観測値で条件付け、残りを破棄する。（右）潜在要約は過去全体を固定サイズに要約する状態 $\mathbf{h}_t$ を引き継ぐ。

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t}). \tag{8.2}$$

$t = 1$ のときプレフィックス $x_{<1}$ は空であり、最初の因子は単に $P(x_1)$ となる。この恒等式は系列上のあらゆる同時分布に対して厳密に成り立つため、次の要素をモデル化することは、系列全体をモデル化することと数学的に同値である。たとえば 3 語のフレーズであれば、次のようになる。

$$P(\text{the}, \text{cat}, \text{sat}) = P(\text{the})\, P(\text{cat} \mid \text{the})\, P(\text{sat} \mid \text{the}, \text{cat}), \tag{8.3}$$

したがって、モデルがこれら 3 つの因子に確率 0.05、0.01、0.2 を割り当てるなら、この文の同時確率は 10^{-4} となる。実際には、長い系列での数値アンダーフローを避けるため、条件付き確率を直接掛け合わせるのではなく、その対数の和をとって系列にスコアを付ける。新しい系列を生成するには、この手順を逆にたどる。まず x_1 をサンプリングし、次に x_2 を x_1 が与えられたもとでサンプリングし、以降も新たに得られた値を次の条件付き分布へと次々に渡していく。

なお、Equation 8.2 では暗黙のうちに順序を選んでおり、系列を左から右へと分解している。原理的には、添字をどのように置換しても数学的に正しい分解が得られる。しかし因果関係のある領域では、通常は順方向の方がはるかにモデル化しやすい。原因から結果を予測する方が、その過程を逆向きにたどるよりも単純になる傾向があるためである (Peters et al. 2017)。過去から未来を予測することは実務上の目的とも一致するため、本章全体を通して系列は左から右へと分解する。

系列の要素が離散的なトークンである場合、各条件付き分布は語彙（vocabulary）上の分類器となり、$P(x_1, \ldots, x_T)$ を推定することは**言語モデリング**（language modeling、8.3 節）と呼ばれる。本節の残りでは、自己回帰予測の仕組みそのものに直接焦点を当てられるよう、連続値を扱うことにする。

8.1.3 マルコフモデルと定常性

固定ウィンドウを用いることは古い観測値を無視しても問題ないと暗黙に仮定しており、すべてのタイムステップにわたって単一のモデルを当てはめることは根底にある予測規則が時間とともにドリフトしないと仮定している。これら 2 つの仮定はいずれも、確率論と時系列解析における古典的な概念である。

8.1.3.1 マルコフ条件

直近 m ステップでの条件付けが履歴全体での条件付けと同等の予測力を持つとき、その系列は**マルコフ条件**（Markov condition）を満たすという。すなわち、直近の過去が与えられたもとで、未来は遠い過去と条件付き独立である。$m = 1$ のとき、この過程は **1 次**マルコフモデルと呼ばれ、一般には m^{th} 次マルコフモデルと呼ばれる。1 次の条件のもとでは、Equation 8.2 の各因子は直前の要素のみに依存し、同時確率は隣接する対にわたる積へと簡約される。

$$P(x_1, \dots, x_T) = P(x_1) \prod_{t=2}^{T} P(x_t \mid x_{t-1}). \tag{8.4}$$

現実世界の系列がマルコフ条件に厳密に従うことはめったにない。自然言語のテキストでは、トークンを追加するにつれて得られる効果は次第に薄れるものの、コンテキストを広げれば有用な情報が得られ続ける。それでもなお、m^{th} 次マルコフ条件を仮定することは実用上不可欠な場合が多い。有限のデータからモデルを計算可能かつ推定可能な範囲に保てるからであり、現代の大規模言語モデル（large language model）でさえ無限の履歴ではなく有界なコンテキストウィンドウ上で動作する。離散系列に対する最も単純なマルコフモデルは、各コンテキストの直後に各後続要素が何回現れるかを数え上げることで $P(x_t \mid x_{t-1})$ を直接推定する。これは、8.3 節で構築する n-gram 言語モデルの原理そのものである。

8.1.3.2 定常性

すべてのタイムステップにわたって単一の予測器を当てはめることは、第 2 の仮定、すなわち先行する要素から各要素を生成する**ダイナミクス**が時間とともに変化しないという仮定に依拠している。時間不変なダイナミクスを持つ過程は**時間斉次**（time-homogeneous）、あるいは機械学習ではより緩やかかつはるかに一般的な呼び方として**定常**（stationary）と呼ばれる。観測される値そのものは大きく変動してもよく、一定に保たれるのは条件付き分布 $P(x_t \mid x_{t-m}, \dots, x_{t-1})$ である。重要な点として、遷移規則が固定されていても、各要素が同一分布に従うとは限らない。同じ規則が異なるコンテキスト上で評価されるため、文書の末尾は冒頭とは異なる振る舞いを示す。系列に沿ってウィンドウをスライドさせ、単一の系列の異なる部分や多数の系列から例を集めて 1 つの共有訓練セットを構築できるのは、この定常性があるからである。

8.1.4 ノイズを含む正弦波の予測

厳密な正解がわかっている設定でこれらの概念が実際にどう機能するかを確かめるため、合成データ（synthetic data）に単純な自己回帰モデルを当てはめ、1 ステップ先予測と複数ステップのロールアウトを比較してみよう。

8.1.4.1 データとモデル

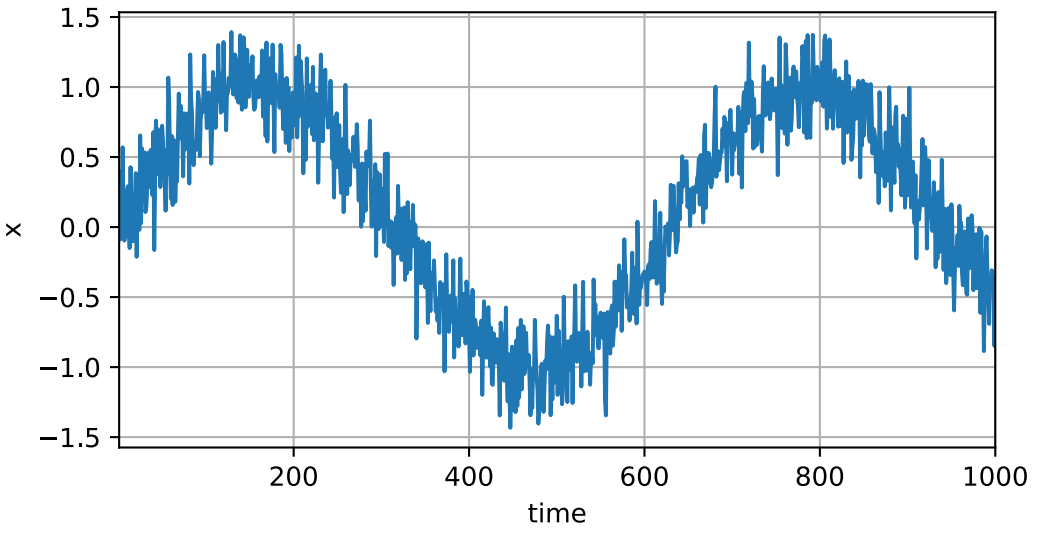
$T = 1000$ タイムステップの系列を、ノイズを含む正弦波 $x_t = \sin(0.01\, t) + \epsilon_t$ から生成する。ここで加法ノイズ $\epsilon_t \sim \mathcal{N}(0, 0.2^2)$ は各ステップで独立に抽出される。この周波数では、1 周期は $2\pi/0.01 \approx 628$ ステップに相当する。

```
class Data(d2l.DataModule):
    def __init__(self, batch_size=16, T=1000, num_train=600, m=4):
        self.save_hyperparameters()
        self.time = d2l.arange(1, T + 1, dtype=d2l.float32)
        noise = jax.random.normal(d2l.get_key(), (T,))
```

```
        self.x = d2l.sin(0.01 * self.time) + noise * 0.2
```

Data をインスタンス化し、data.x を data.time に対してプロットすると、ノイズを含んだ振動が約 1.5 周期分表示される。

```
data = Data()
d2l.plot(data.time, data.x, 'time', 'x', xlim=[1, 1000], figsize=(6, 3))
```



この系列を教師ありの訓練例へ変換するため、ウィンドウ長 m の固定ウィンドウ戦略を適用する。各例は、特徴ベクトル $(x_{t-m}, \ldots, x_{t-1})$ と目的ラベル $y = x_t$ の対からなる。これによって合計 $T - m$ 個の例が得られる。最初の m 個の要素には十分な履歴がないため、ここでは単に破棄する（ゼロパディングもよく使われる代替策である）。lag_features メソッドはラグごとに 1 列を持つ特徴行列全体を一度に構築し、データローダは最初の 600 個の例（正弦波の 1 周期にわずかに満たない長さ）を訓練に使用する。Figure 8.2 に、$m = 4$、$T = 1000$ の設定で得られる表を示す。

```
@d2l.add_to_class(Data)
def lag_features(self):
    """Features (x_{t-m}, ..., x_{t-1}) and labels x_t for t > m."""
    features = [self.x[i : self.T-self.m+i] for i in range(self.m)]
    return d2l.stack(features, 1), d2l.reshape(self.x[self.m:], (-1, 1))

@d2l.add_to_class(Data)
def get_dataloader(self, train):
    i = slice(0, self.num_train) if train else slice(self.num_train, None)
    return self.get_tensorloader(self.lag_features(), train, i)
```

予測器には、$m = 4$ 個のラグ特徴量に対する通常の線形回帰を用いる。考えられる限り最も単純な自己回帰モデルといえる。

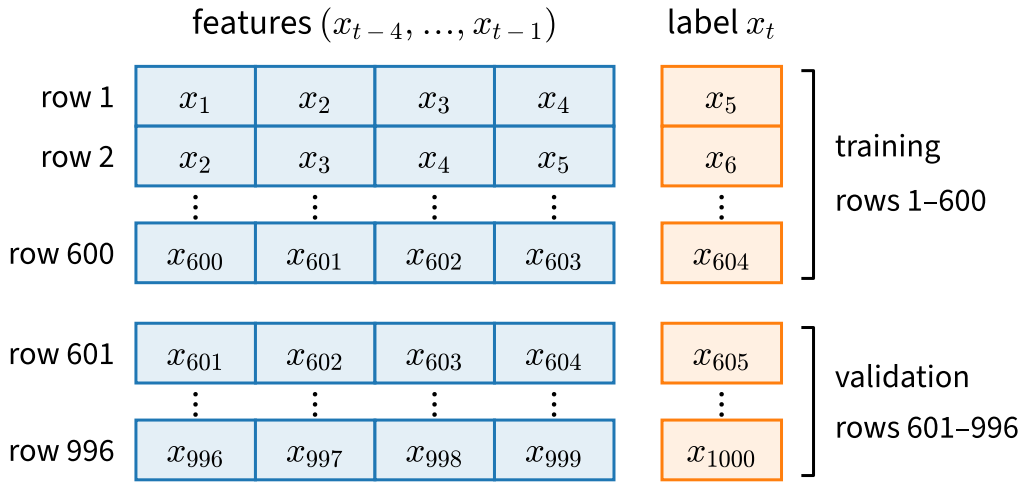
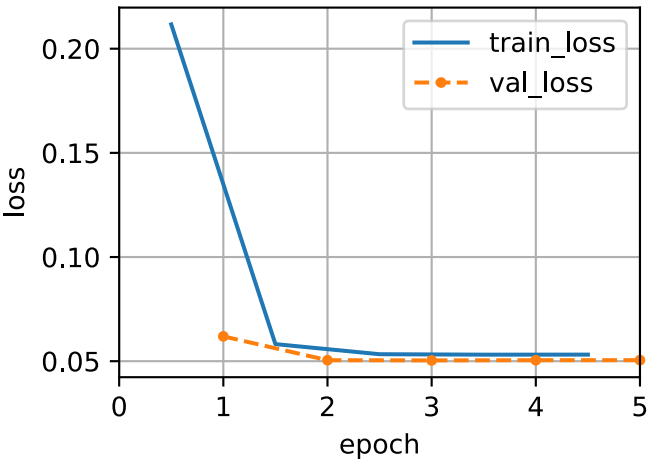


Figure 8.2: $m = 4$、$T = 1000$ におけるラグ特徴量とラベル。第 i 行は 4 つの値 $(x_i, \ldots, x_{i+3})$ とラベル x_{i+4} を対にしており、全 996 行の末尾は x_{1000} となる。最初の 600 行を訓練に、残りの 396 行を検証に使用する。

```
model = d2l.LinearRegression(num_inputs=data.m, lr=0.01)
trainer = d2l.Trainer(max_epochs=5)
trainer.fit(model, data)
```



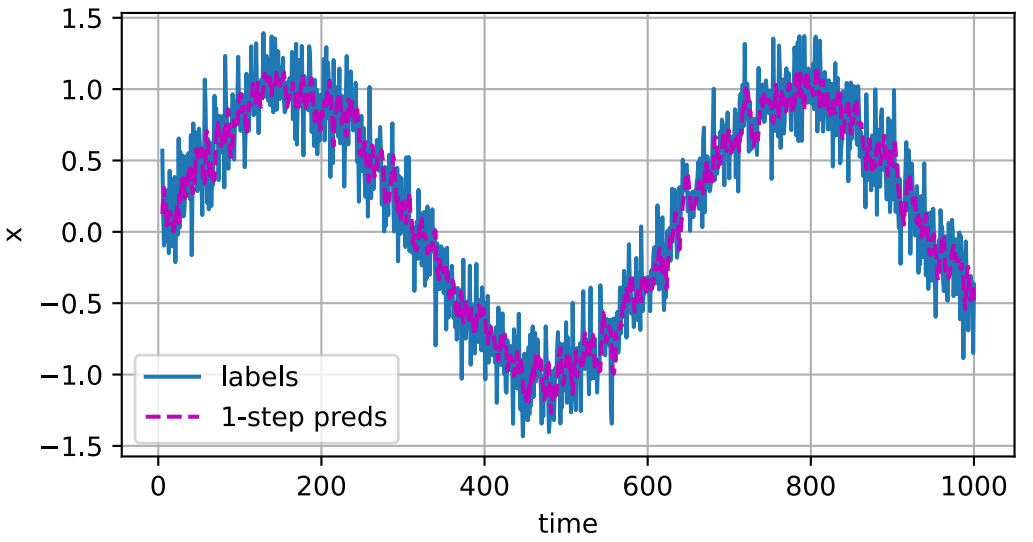
8.1.4.2 1 ステップ先予測

まずは簡単なケースから始めよう。すなわち、$\hat{x}_t$ を、**観測された**直前 m 個の値からすべてのタイムステップ $t > m$ において予測する。

```
features, labels = data.lag_features()
onestep_preds = d2l.numpy(model(features))
```

1 ステップ予測 onestep_preds を正解の labels に重ねて、全 $T - m = 996$ タイムステップにわたりプロットする。

```
d2l.plot(data.time[data.m:], [labels, onestep_preds], 'time', 'x',
         legend=['labels', '1-step preds'], figsize=(6, 3))
```



1 ステップ予測は元の波形によく追従しており、訓練分割の外側のデータでモデルが評価されるステップ 600 以降でもその精度は保たれている。

8.1.4.3 複数ステップのロールアウト

複数ステップ先の未来を予測するのは格段に難しくなる。タイムステップ t で観測が途切れ、$\hat{x}_{t+5}$ を予測したいとしよう。$x_{t+1}, \ldots, x_{t+4}$ は観測されていないため、1 ステップ予測器を直接評価することはできない。そこで代わりに、モデル自身の予測をあたかも真の観測値であるかのように入力へ戻し、1 タイムステップずつ先へ進めていく。

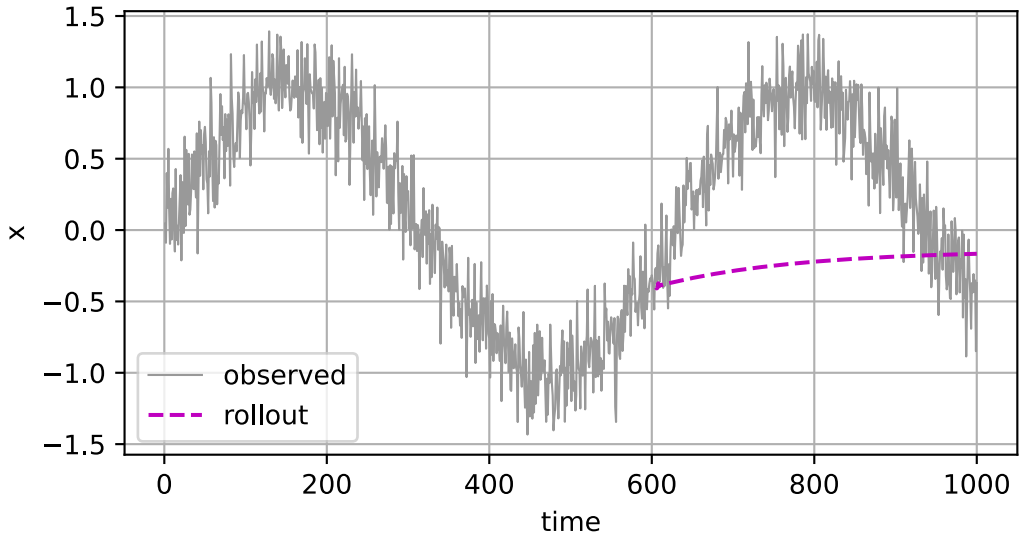
$$\begin{aligned}
\hat{x}_{t+1} &= f(x_{t-3}, x_{t-2}, x_{t-1}, x_t),\\
\hat{x}_{t+2} &= f(x_{t-2}, x_{t-1}, x_t, \hat{x}_{t+1}),\\
\hat{x}_{t+3} &= f(x_{t-1}, x_t, \hat{x}_{t+1}, \hat{x}_{t+2}),\\
\hat{x}_{t+4} &= f(x_t, \hat{x}_{t+1}, \hat{x}_{t+2}, \hat{x}_{t+3}),\\
\hat{x}_{t+5} &= f(\hat{x}_{t+1}, \hat{x}_{t+2}, \hat{x}_{t+3}, \hat{x}_{t+4}).
\end{aligned}$$

5 ステップ目以降、モデルはそれ以前の自身の予測のみに頼ることになる。$\hat{x}_{t+k}$ を k **ステップ先予測**と呼び、1 ステップ予測を連鎖的に先へとつないでいく過程を**ロールアウト**と呼ぶ。以下のコードは、訓練分割で使われた最後の値である $t = 604$ から系列の末尾まで、線形モデルをロールアウトする。

```
def rollout_step(window, _):
    pred = model(window[None])[0, 0]  # window: the last m values
    return jnp.append(window[1:], pred), pred

start = data.num_train + data.m
_, preds = jax.lax.scan(rollout_step, data.x[start-data.m : start],
                        length=data.T - start)
multistep_preds = d2l.numpy(data.x.at[start:].set(preds))
```

JAX では、タイムステップにわたるループを jax.lax.scan で記述する。これによってウィンドウを 1 ステップずつ次へ引き継ぎながら、予測を積み重ねていく。

```
start = data.num_train + data.m
d2l.plot([data.time, data.time[start:]], [data.x, multistep_preds[start:]],
         'time', 'x', figsize=(6, 3))
ax = d2l.plt.gca()
ax.lines[0].set(color='0.6', linewidth=0.8)  # The series: thin and grey
ax.legend(['observed', 'rollout']);
```



ロールアウトは振動にまったく追従できていない。最後に観測されたウィンドウの水準付近から始まったあと、真の波形が上下し続けるのをよそに、緩やかに定数へとドリフトしていく。1 ステップ誤差がこれほど小さいモデルが、なぜここまで徹底的に破綻するのだろうか。学習された重みを調べれば、その理由はただちに明らかになる。

```
w, b = model.get_w_b()
print(f'weights on x[t-{data.m}], ..., x[t-1]:',
      d2l.numpy(d2l.reshape(w, -1)).round(3))
```

```
weights on x[t-4], ..., x[t-1]: [0.236 0.258 0.255 0.237]
```

4 つの重みはいずれも $1/4$ の近くに収まっており、モデルは単に直前 4 個の平均として次の観測

値を予測することを学習している。1 ステップ予測においては、これは理にかなった戦略である。正弦波の変化量は 1 ステップあたり高々 0.01 であるのに対し、加法ノイズの標準偏差は 0.2 もある。ノイズを含む 4 つの測定値を平均すればノイズ分散は 4 分の 1（0.04 から 0.01）に減少し、緩やかに変化する曲線が 1〜2 ステップ遅れることで生じる統計的バイアス（statistical bias）はごくわずかである。ところが、移動平均（moving average）にそれ自身の出力を入力し続けると、現在の水準を維持することしかできない。しかも重みの和が 1 をわずかに下回るため、その水準はロールアウトの不動点（fixed point）$b/(1-\sum_i w_i)$ へとゆっくり減衰していく。ここで b はバイアスである（2 つの小さな数値の比であるため、厳密な漸近先は実行ごとに異なる）。1 ステップの目的関数では正弦波の曲率を学習してもほとんど有利にならないため、モデルはその曲率をまったく学習しておらず、長いホライズンにわたるとその欠落が誤差の大部分を占めるようになる。重要な点として、線形性そのものが原因ではない。ノイズがなければ、正弦波は厳密な 2 階漸化式 $x_t = 2\cos(0.01)\,x_{t-1} - x_{t-2}$ を満たすため、2 と -1 付近の重みを持つわずか 2 つのラグだけで振動を無限に再現できるはずである。しかしノイズが存在する状況では、そのような重みはノイズ分散を 5 倍に増幅してしまうため、1 ステップの二乗誤差を最小化すると当てはめは平均化へと押しやられるのである（演習問題 1）。

8.1.4.4 ホライズンに伴う誤差の増大

ホライズン k が大きくなるにつれて予測がどれほど急速に劣化するかを測るため、系列内のすべての有効な位置から同時に出発して k ステップ先予測を計算する。補助関数 `k_step_pred` は開始インデックスごとに 1 つの要素を持つ列のリストを保持する。まず観測された m 個のラグから始め、直近 m 列から計算した各段階の予測を順に追加していき、$(j-1)^{\text{th}}$ の要素に j ステップ先予測が格納されたリストを返す。

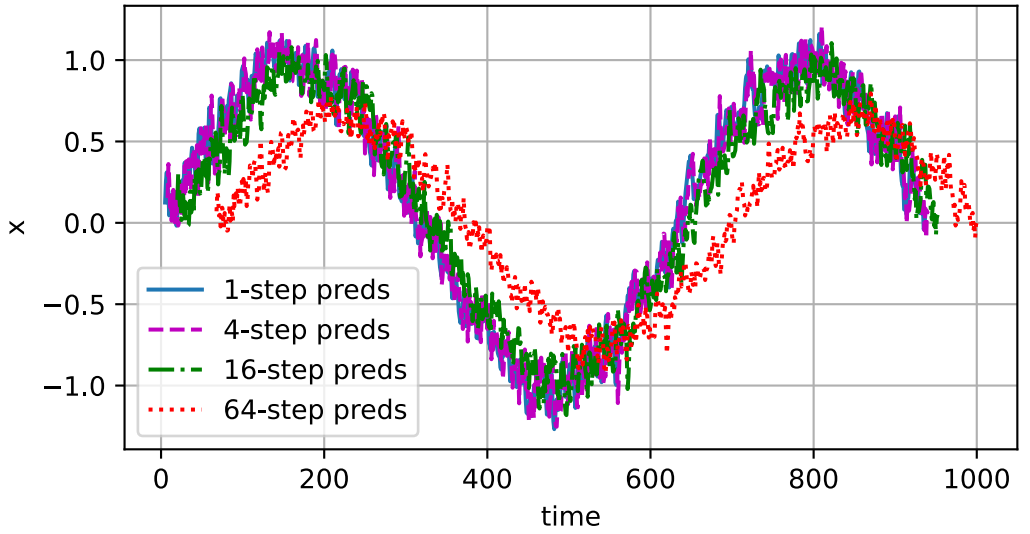
```
def k_step_pred(model, x, m, k):
    """Predictions 1, 2, ..., k steps ahead from every start position."""
    n = len(x) - m - k + 1                      # number of start positions
    columns = [x[i : i + n] for i in range(m)]  # observed lags
    for j in range(k):
        preds = model(d2l.stack(columns[j : j + m], 1))
        columns.append(d2l.reshape(preds, -1))
    return columns[m:]                          # entry j-1: j steps ahead
```

`k_step_pred` を用いて、訓練済みモデルを最大 $k = 64$ ステップ先まで、`data.x` 内のすべての有効な開始ウィンドウからロールアウトする。

```
steps = (1, 4, 16, 64)
preds = k_step_pred(model, data.x, data.m, steps[-1])
```

ホライズン $k \in \{1, 4, 16, 64\}$ における予測曲線を `preds` から取り出してプロットし、ホライズン長とともに誤差がどのように蓄積するかを確認しよう。

```
d2l.plot([data.time[data.m+k-1 : data.T-steps[-1]+k] for k in steps],
         [d2l.numpy(preds[k-1]) for k in steps], 'time', 'x',
         legend=[f'{k}-step preds' for k in steps], figsize=(6, 3))
```



1 ステップ予測と 4 ステップ予測は系列によく追従しているが、16 ステップ予測では遅れが目立ち始め、64 ステップ予測になると波形を約 64 ステップずらして振幅を目に見えて減衰させたコピーのようになる（縮小の度合いは、重みの和が 1 をどれだけ下回るかで決まる）。系列の分散（平均を予測する自明な定数予測の誤差）をベースラインとして、観測値に対する平均二乗誤差（mean squared error）によってこの劣化を定量化できる。

```
x = d2l.numpy(data.x)
for k in steps:
    err = d2l.numpy(preds[k-1]) - x[data.m+k-1 : data.T-steps[-1]+k]
    print(f'{k:2d} steps ahead: mean squared error {(err**2).mean():.3f}')
print(f'variance of the series: {x.var():.3f}')
```

```
 1 steps ahead: mean squared error 0.052
 4 steps ahead: mean squared error 0.055
16 steps ahead: mean squared error 0.072
64 steps ahead: mean squared error 0.242
variance of the series: 0.493
```

1 ステップ先では、平均二乗誤差は 0.05 付近にある。これは削減不可能なノイズ分散 $0.2^2 = 0.04$ に、4 つの観測値の平均に残るノイズ分散 0.01 を加えた値である。k に伴うその後の増大は、ほぼすべて位相遅れによって説明できる。k ステップ予測は k ステップ前の系列の水準をほぼそのまま再現するため、追加で生じる誤差は正弦波が k ステップの間にどれだけ動くかに等しい。波形の位相 a にわたって平均すると、その変化量の二乗は次の式になる。

$$\frac{1}{2\pi}\int_0^{2\pi} \big(\sin(a + 0.01\,k) - \sin a\big)^2\, da = 1 - \cos(0.01\,k), \tag{8.5}$$

この式の値は 0.001（$k = 4$）、0.013（$k = 16$）、0.20（$k = 64$）となる。これにベースラインの下限である 0.05 を加えると、予測される誤差は約 0.05、0.05、0.065、0.25 となり、実測値とよく一致する。64 ステップ先までに誤差は 1 ステップ誤差の 4〜6 倍に達し、すでに系列全体の分散の約半分に及んでいる。

ここで重要な機微に目を向けよう。この誤差は、暴走的な誤差蓄積から生じたものではない。平均化モデルはフィードバックされる入力を常に減衰させるため、ロールアウトは発散するのではなく定数へと滑らかに減衰していく。k ステップ予測が目的値を外すのは、小さな誤差が指数関数的に増幅したからではなく、真の波形が先へ進んでしまったからである。より一般に、1 ステップの目的関数のみで厳密に訓練されたモデルは、1 ステップ先の予測に役立つ特徴しか学習しない。多数のステップにわたってロールアウトしたときの性能は真の基底ダイナミクスを捉えられているかどうかに左右されるが、1 ステップ損失が小さいからといってそのダイナミクスが学習された保証にはならない。

複数ステップのロールアウトは、この減衰型の線形モデルには見られない第 2 の、より広く知られた失敗モードにも直面する。すべての予測誤差がそれ以降の全ステップにおける入力の一部となるため、摂動を減衰させるのではなく増幅してしまうモデルは、データ分布（data distribution）から次第に大きくドリフトしうるのである。言語モデリングでは、運悪くサンプリングされた数個のトークンによって生成器が訓練時には一度も現れなかった不自然な言い回しへと誘導され、話題から逸れたり反復ループに陥ったりすることがある。学習された物理シミュレータが長い軌跡にわたって安定性を失うのも、ほぼ同じ理由による。この不一致を緩和するには、大きく分けて 2 つの戦略が有効である。訓練時には、正解の履歴とあわせてモデル自身の予測にも触れさせることで、不完全なプレフィックスが分布外（out-of-distribution）にならないようにできる。推論時には、各予測分布を具体的な出力へと変換するデコーディング規則を工夫し、ロールアウトを脱線させる低確率な選択を抑制できる。テキストにおけるこの話題には 8.6 節で改めて立ち返るが、その前にまず、生のテキストを系列モデルが処理できる表現へと変換する必要がある。

8.1.5 まとめ

自己回帰モデルは、直近 m 個の観測値からなる固定ウィンドウ（n-gram の背後にある原理）、または過去全体を要約する有界サイズの潜在状態（リカレントネットワークの背後にある原理）のいずれかを用いて各要素を先行する要素から推定することで、系列モデリングを教師あり予測へと帰着させる。確率の連鎖律により、1 ステップの条件付き分布を掛け合わせることで、系列のスコアリングと生成の両方が可能な完全な確率モデル（probabilistic model）が得られる。すべてのタイムステップにわたって単一のモデルを当てはめることは、履歴から各要素を生成する条件付き規則が時間とともに変化しないという定常性の仮定に依拠している。最後に、高い 1 ステップ予測精度は高い複数ステップ予測性能を保証しない。1 ステップの訓練は短距離の特徴しか評価しないためである。ノイズを含む正弦波の例では、平均化を行う線形モデルは 1 ステップのノイズ下限に迫る一方で、64 ステップ先では 4〜6 倍もの誤差を被った。これは、ロールアウト中に誤差が増幅するモデルを悩ませる累積的なドリフトを考慮に入れる前の段階でもすでに生じる現象である。

8.1.6 演習問題

1. [code] **ウィンドウ長**。`Data` は `m=4` 個のラグ特徴量を与え、`d2l.LinearRegression` がそれを当てはめる。
    1. `m` を増やして再訓練せよ。検証誤差の改善が止まるまでに、過去の観測値はいくつ必要か。
    2. `Data` のノイズ項を取り除き、`m=2` で当てはめよ。得られた 2 つの重みを $2\cos(0.01)$ および -1 と比較し、モデルをロールアウトせよ。次に、標準偏差 0.001 から 0.2 までの範囲でノイズを戻せ。どのノイズレベルで重みは曲率の表現を諦めて平均化へと転じるか。
2. [code] **非線形自己回帰器**。同じ 4 つのラグ特徴量上で `d2l.LinearRegression` を小さな MLP に置き換え、必要に応じてエポック数を増やして再訓練せよ。1 ステップ予測と複数ステップ予測はどのように変化するか。

3. [code] **ホライズンに伴う誤差の増大**。各 k について、`k_step_pred(model, data.x, data.m, k)` が返すリストの最後の要素には、`data.x[data.m+k-1:]` に対する k ステップ先予測が格納されている。
    1. $k \in \{1, 2, 4, \dots, 256\}$ について、k ステップの平均二乗誤差を対数軸上の k に対してプロットせよ。曲線はどこで上向きに折れ曲がり、その折れ曲がりは 16 ステップ予測および 64 ステップ予測の減衰とどのように関係しているか。
    2. k ステップの平均二乗誤差が系列の分散を超える最小の k を求めよ。このホライズンを超える予測には、定数予測と比べてどのような価値があるか。
4. [code] **膨張ラグと平均化ラグ**。●`d2l.LinearRegression` の特徴量は 4 つのまま保ちつつ、より過去まで届くようにせよ。1 つはラグ $x_{t-1}, x_{t-2}, x_{t-4}, x_{t-8}$ とし、もう 1 つは 4 つのブロック平均 $\frac{1}{4}\sum_{i=0}^{3} x_{t-4j+i}$ $(j = 1, \dots, 4)$ とする。それぞれの検証誤差を $m = 4$ および $m = 16$ の場合と比較せよ。どちらの構成が有効であり、それはなぜか。
5. [code] **レジームスイッチング**。●`Data` の単一の正弦波を、200 タイムステップごとに $\sin(0.01\,t)$ と $\sin(0.05\,t)$ が交互に切り替わる系列に置き換え、加法ノイズはそのまま残せ。すると訓練例と検証例の両方に 2 つのレジームが含まれることになる。この切り替わり系列は 8.1.3.2 節の定常性の仮定に反している。それでも $m = 4$ の単一の線形モデルで両方のレジームに当てはめられるか、またそれはなぜか。各レジームの訓練誤差と検証誤差を別々に求め、系列上で当てはめが最も悪化する位置を特定せよ。

8.2 テキストからトークンへ

本章でモデル化する系列の大半はテキストである。ここには 8.1 節の合成数値系列にはなかった課題が生じる。ニューラルネットワークは数値のベクトルを扱うのに対し、テキストは生の文字列として与えられるからである。どのような言語モデルを訓練するにせよ、まず文字列をどのように**トークン**（観測 x_t として扱われる最小単位）へ分割し、各トークンをどのように数値インデックスへ対応付けるかを決めなければならない。一見すると地味なこの前処理の選択は、極めて広範な影響を及ぼす。モデルの埋め込み（embedding）層と出力層のサイズ、モデルが読み込むあらゆる文書の系列長（sequence length）、さらには算術のようなタスクの難易度までもがこれによって決まるのである。『The Time Machine』を個々の文字に分割すると、モデルは約 170,000 タイムステップを順に進まなければならなくなる。一方、単語単位に分割すると、約 4,600 種類の異なる単語それぞれに出力クラスが必要になるうえ、訓練時に出現しなかった未知の単語には対応できない。こうした理由から、現代のすべての言語モデルには専用のトークナイザが付属しており、トークナイザはモデル自体の不可分な一部と捉えるのが最善である。

まずは、語彙（vocabulary）サイズと系列長のトレードオフにおいて両極端に位置する、文字レベルのトークン化（tokenization）と単語レベルのトークン化を比較することから始める。続いて、普遍的な基底アルファベットとして生のバイト列へと降り、GPT 系モデルの基盤となっている手法である**バイトペアエンコーディング**（BPE）を用いてデータから直接サブワード語彙を学習する (Sennrich et al. 2016; Radford et al. 2019)。ゼロから構築するこの実装は Python でわずか数十行にすぎないが、GPT-2 の実用トークナイザをトークン単位で完全に再現する。

8.2.1 データセットの読み込み

本章を通して共通のコーパスとして用いるのは、H. G. Wells の『The Time Machine』（30,000 語強の中編小説）である。実世界の事前学習（pretraining）コーパスは数兆トークン規模に及ぶが、このデータセットは自然言語の統計的性質を示すのに十分な豊かさを備えつつ、数秒でトークン化できるほど小さい。以下の `_download` メソッドは、この書籍を取得し、生のテキストを 1 つの文字列として読み込む。

```
import collections
import re
import tiktoken
```

```
class TimeMachine(d2l.DataModule):
    """The Time Machine dataset."""
    def _download(self):
        fname = d2l.download(d2l.DATA_URL + 'timemachine.txt', self.root,
                             '090b5e7e70c295757f55df93cb0a180b9691891a')
        with open(fname) as f:
            return f.read()

data = TimeMachine()
raw_text = data._download()
raw_text[:60]
```

```
'The Time Machine, by H. G. Wells [1898]\n\n\n\n\nI\n\n\nThe Time Tra'
```

古典的な NLP パイプラインでは、句読点を除去し、すべての文字を小文字に統一してから、整形後の文字列をトークンに分割するという積極的な正規化を生テキストに施していた。ここでも、以下で行う文字レベルと単語レベルの比較や、このパイプラインを引き続き使用する後続の節のために、単純な正規化ステップを残しておく。もっとも、このような前処理が大雑把であることは否めない。大文字・小文字の区別（「Bill」と「bill」の違いなど）のような有用な手がかりを捨て去り、どの文字が重要かについて脆い仮定を組み込んでしまうからである。対照的に、本節の後半で構築するバイトレベルのトークナイザは、正規化を一切必要としない。

```
@d2l.add_to_class(TimeMachine)
def _preprocess(self, text):
    return re.sub('[^A-Za-z]+', ' ', text).lower()

text = data._preprocess(raw_text)
text[:60]
```

```
'the time machine by h g wells i the time traveller for so it'
```

8.2.2 文字、単語、バイト

8.2.2.1 文字と単語

系列モデルは 1 タイムステップにつき 1 つのトークンを処理するため、トークンの単位として最も自然な 2 つの選択肢は、個々の文字と単語全体である。まずは文字から順に、両方を見てみよう。

```
@d2l.add_to_class(TimeMachine)
def _tokenize(self, text):
    return list(text)

tokens = data._tokenize(text)
','.join(tokens[:30])
```

```
't,h,e, ,t,i,m,e, ,m,a,c,h,i,n,e, ,b,y, ,h, ,g, ,w,e,l,l,s, '
```

代わりに空白文字で分割すると、単語トークンが得られる。それぞれの方式について、**異なる**トークン種別の数（これが語彙サイズを決める）と、コーパス内の総トークン数（これが系列長を決める）の両方を数えてみよう。

```
words = text.split()
print(f'characters: {len(set(tokens)):>6} distinct, '
      f'{len(tokens):>7} tokens in the corpus')
print(f'words:      {len(set(words)):>6} distinct, '
      f'{len(words):>7} tokens in the corpus')
```

```
characters:     27 distinct,  173428 tokens in the corpus
words:        4579 distinct,   32775 tokens in the corpus
```

正規化したテキストを文字レベルでトークン化する場合、必要な記号はわずか 27 個であり、未知の文字に遭遇することも決してない。しかし、この中編小説は 170,000 以上のタイムステップに引き伸ばされ、主語と動詞の一致のような長距離の依存関係が数十ステップにまたがることになる。単語レベルのトークン化では、同じテキストが約 5 分の 1 のトークン数に圧縮され、各トークンははるかに豊かな意味内容を持つ。しかし、この短い中編小説でさえ、単語に分割するとすでに数千種類の異なるトークン種別が生じ、コーパスが大きくなるにつれて語彙は際限なく膨張する。固有名詞、誤植、語形変化（「traveller」、「travellers」、「travelled」）のすべてが独自の項目を占め、その大半は 1〜2 回しか出現しない（8.3 節）。さらに悪いことに、訓練セットに含まれない単語はすべて、テスト時に**語彙外**（out-of-vocabulary、OOV）となる。Figure 8.3 はこの基本的なトレードオフを示している。語彙を大きくすれば得られる系列は短くなるため、設計上の中心的な問いはこのスペクトラム上のどこで運用するかという点にある。サブワードトークナイザを使えばその動作点を柔軟に選ぶことができ、本節の残りの大部分はその構築に充てる。

8.2.2.2 トークンからインデックスへ

サブワードに進む前に、本書の後続のいくつかの節でも引き続き使用する古典的なパイプラインを完成させておこう。ニューラルネットワークは文字列ではなく整数インデックスを扱うため、文字や単語のパイプラインでは、出現頻度の高い順に各トークン種別へインデックスを割り当て、未知のトークンを予約済みの <unk> 項目に対応付ける**語彙**オブジェクトを用いる。出現回数が `min_freq` 回未満の低頻度トークンも、語彙をコンパクトに保つために枝刈りでき、同様に <unk> へとフォールバックされる。

```
class Vocab:
    """Vocabulary for text."""
    def __init__(self, tokens=(), min_freq=0, reserved_tokens=()):
        # Flatten a 2D list if needed
        if tokens and isinstance(tokens[0], list):
            tokens = [token for line in tokens for token in line]
        # Count token frequencies
        counter = collections.Counter(tokens)
        self.token_freqs = sorted(counter.items(), key=lambda x: x[1],
                                  reverse=True)
        # The list of unique tokens, ordered by descending frequency.
        # Reserve <unk> at index 0 so vocab[0] is the unknown token.
        self.idx_to_token = ['<unk>'] + list(reserved_tokens) + [
            token for token, freq in self.token_freqs
```

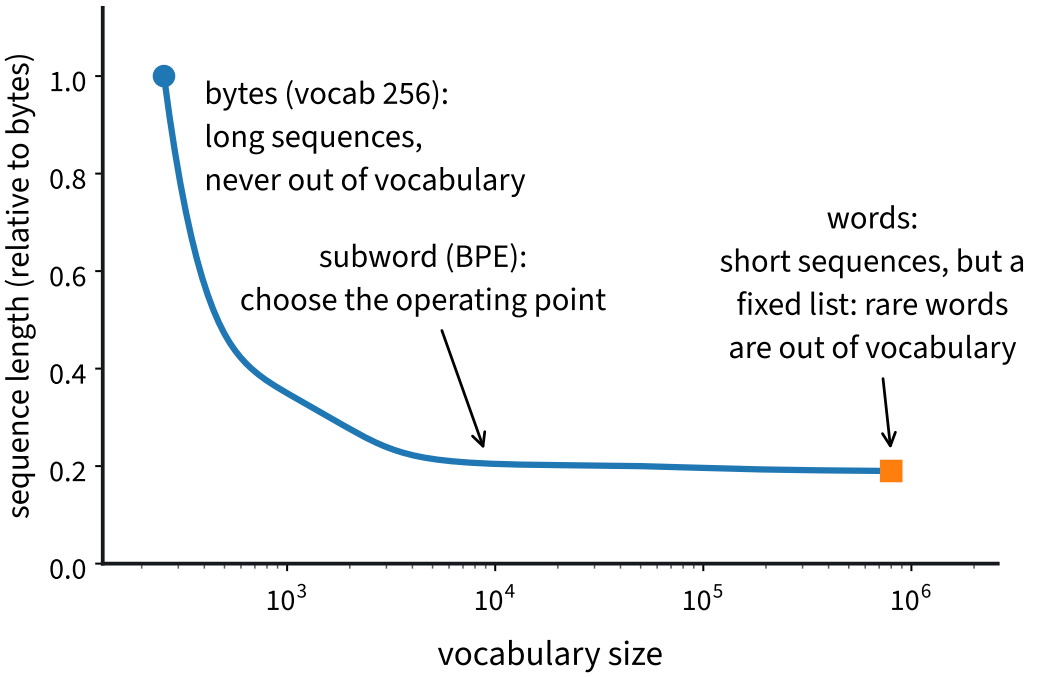


Figure 8.3: 1 本のトレードオフ曲線上に並ぶトークン化の粒度。語彙が大きいほど、同じテキストをより少ないトークンで表現できる。バイト列は決して語彙外にならないが、固定された単語リストでは語彙外が生じる。バイトペアエンコーディングはこの両極端の間を補間する。

```
            if freq >= min_freq and token not in reserved_tokens]
        self.token_to_idx = {token: idx
                             for idx, token in enumerate(self.idx_to_token)}

    def __len__(self):
        return len(self.idx_to_token)

    def __getitem__(self, tokens):
        if not isinstance(tokens, (list, tuple)):
            return self.token_to_idx.get(tokens, self.unk)
        return [self.__getitem__(token) for token in tokens]

    def to_tokens(self, indices):
        if isinstance(indices, (list, tuple)) or getattr(indices, 'ndim', 0):
            return [self.idx_to_token[int(index)] for index in indices]
        return self.idx_to_token[int(indices)]

    @property
    def unk(self):  # Index for the unknown token
        return self.token_to_idx['<unk>']
```

Vocab をテストするため、文字のリスト tokens から語彙を構築し、先頭の 10 文字を整数の indices に変換したうえで、vocab.to_tokens によって元のトークンへ戻してみよう。

```
vocab = Vocab(tokens)
indices = vocab[tokens[:10]]
print('indices:', indices)
print('tokens:', vocab.to_tokens(indices))
```

```
indices: [3, 9, 2, 1, 3, 5, 13, 2, 1, 13]
tokens: ['t', 'h', 'e', ' ', 't', 'i', 'm', 'e', ' ', 'm']
```

補助メソッド build は、この古典的なパイプライン全体（前処理、トークン化、インデックス付け）を 1 回の呼び出しにまとめ、整数インデックスからなるコーパスとその語彙を返す。

```
@d2l.add_to_class(TimeMachine)
def build(self, raw_text, vocab=None):
    tokens = self._tokenize(self._preprocess(raw_text))
    if vocab is None: vocab = Vocab(tokens)
    corpus = [vocab[token] for token in tokens]
    return corpus, vocab

corpus, vocab = data.build(raw_text)
len(corpus), len(vocab)
```

```
(173428, 28)
```

8.2.2.3 バイトとしてのテキスト

文字と単語の間を滑らかに補間するには、まずあらゆる文字列を表現できる基底アルファベットが必要になる。驚くべきことに、文字そのものはこの役割に向いていない。Unicode は多様な文字体系、ダイアクリティカルマーク（発音区別符号）、数学記号、絵文字にわたって約 150,000 文

字を定義しているため、平易な英語から一歩踏み出した途端に、文字レベルの語彙はもはや小さなものではなくなってしまう。そこで現代のトークナイザは、もう 1 段階低いレベルへと降りる。コンピュータ上に保存されるあらゆる文字列は、文字エンコーディング（ほぼ例外なく UTF-8）のもとで最終的に**バイト**の系列として表現される。1 バイトが取り得る値はわずか $2^8 = 256$ 通りしかないため、バイトレベルの基底語彙は 256 個の記号しか持たず、語彙外の入力に遭遇することが**決して**ない。どのような言語のどのような文字列であっても、句読点やコード、誤植を含め、余すところなくバイト列へと分解できる。

UTF-8 は、標準的な ASCII 文字には 1 バイトを、あまり一般的でない文字体系や記号には最大 4 バイトを用いる可変長エンコーディングによって、この普遍性を実現している。

```
s = 'Hello, naïve café: 你好  '
print(len(s), 'characters,', len(s.encode('utf-8')), 'bytes')
for ch in 'a é 你  '.split():
    print(repr(ch), '->', list(ch.encode('utf-8')))
n_bytes = len(raw_text.encode('utf-8'))
print(f'The novella: {n_bytes} bytes, {n_bytes / len(words):.1f} per word')
```

```
23 characters, 32 bytes
'a' -> [97]
'é' -> [195, 169]
'你' -> [228, 189, 160]
' ' -> [240, 159, 154, 128]
The novella: 178979 bytes, 5.5 per word
```

この普遍的なアルファベットの代償は系列長である。先ほどの中編小説は約 180,000 バイト（1 単語あたり 5 トークン超）へと膨らみ、さらに「é」のようなアクセント付き文字は 2 バイト、中国語の漢字は 3 バイト、絵文字は 4 バイトを要するため、生のバイト系列では英語以外のテキストがより一層引き伸ばされる。以下で扱う手法はすべて、バイトの普遍性を保ちながら、系列を扱いやすい長さへと圧縮し直すものである。

8.2.3 バイトペアエンコーディング

8.2.3.1 マージ規則

バイトペアエンコーディング（BPE）はもともと、最も頻度の高い隣接バイトのペアを新しい記号に置き換える操作を繰り返すデータ圧縮手法として考案された (Gage 1994)。Sennrich et al. (2016) は機械翻訳（machine translation）向けのサブワード単位を得るために同じ規則を単語内の文字へ適用し、その後 GPT-2 がこれを生のバイト列へと戻した (Radford et al. 2019)。本節で構築するのは、このバイトレベルの変種である。まず 256 個の単一バイト記号から出発し、貪欲に語彙を拡大していく。各反復において、訓練コーパス内のあらゆる隣接トークンの**ペア**を数え上げ、最も頻度の高いペアを 1 つの新しいトークンへとマージする操作を、語彙が目標サイズに達するまで繰り返す。各マージは置き換えた出現回数とちょうど同じ数だけコーパスを短くするため、BPE は貪欲な圧縮アルゴリズムとして機能し、学習された語彙はコーパス内で頻出する断片の辞書を形成する。

アルゴリズムを手作業でトレースしてみると、その動作が明快にわかる。5 種類の単語が計 8 回出現する次のトイコーパスを考えよう。

```
hug pug pun bun hugs hug hug pug
```

生のコーパスの長さは 32 バイトである。最も頻度の高い隣接ペアは (u, g) であり、「hug」、「pug」、「hugs」の中に計 6 回出現するため、最初のマージでは新しいトーク

ン ug が導入される。すべての箇所を ug に置き換えると、最も頻度の高い隣接ペアは (h, ug) となり、以降も同様に処理が続く。

ステップ	最頻出ペア	出現回数	新トークン	コーパス長
0	（バイト列から開始）			32
1	u, g	6	ug	26
2	h, ug	4（同数）	hug	22
3	hug, ' '	3（同数）	'hug '	19

なお、ステップ 2 とステップ 3 には出現回数が同数のペアが存在する。ステップ 2 では (ug, ' ') も 4 回出現し、ステップ 3 では (' ', p) と (' ', hug) も 3 回出現する。3. 標準的な慣例に従い、同数の場合はコーパス内で最も早く出現するペアを優先する。

このトレースから 2 つの重要な性質が直接浮かび上がる。第 1 に、マージは階層的に合成される。hug が h と ug から形成されるように、学習された語彙はバイトアルファベット上の**マージ木**（merge tree）を定義する（Figure 8.4）。これにより、学習済みのマージを順に再生することで、いかなる未知のテキストも分割できる。たとえば「hugs」という単語は hug と s にきれいに分割される一方、低頻度語の「bun」は生の 3 バイトのまま残る。学習されたマージの対象とならなかった部分文字列は、単に個々のバイトへとフォールバックされるのである。第 2 に、ステップ 3 で単語とその**末尾の空白**が結合された点に注目しよう。末尾の空白自体は害がないものの、その後のマージで後続する単語の先頭まで結合されるのを防ぐ仕組みは何もない。実際のコーパスでは、制約のない BPE は複数単語にまたがるトークンを容易に作り出してしまう。この欠点はまもなく解消する。

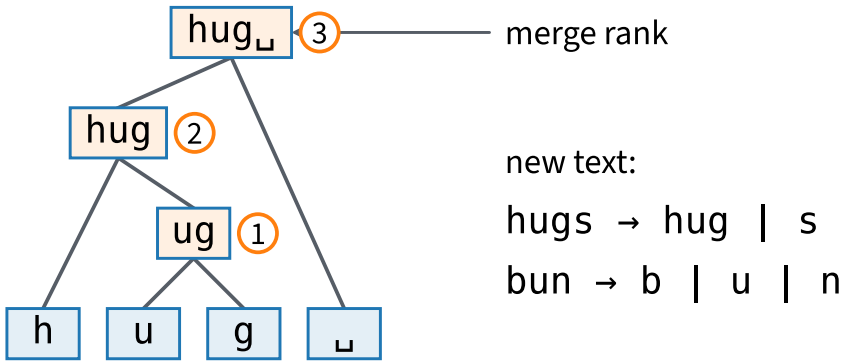


Figure 8.4: トイコーパスで学習された最初の 3 回のマージは、バイトアルファベット上の木を形成する。新しいテキストはこの木を再利用し、「hugs」は 2 トークンになる一方、未知の断片はバイトへとフォールバックされる。

8.2.3.2 実装

BPE アルゴリズムを 3 つの簡潔な要素に分けて実装する。まず、`BPETokenizer` クラスがトークナイザの状態を保持する。トークン ID の 0〜255 は生のバイトを表し、学習された各マージには次に利用可能な整数 ID が順に割り当てられるため、ID はマージの**ランク**も兼ねる（ID が小さいほど早い段階のマージであることを示す）。辞書 `merges` はマージされた ID ペアをその新しい ID に

対応付け、vocab はすべての ID を対応するバイト列に対応付ける。引数 pattern と specials、および byte_ids は本節の後半で登場するため、現時点では無視してよい。また、regex パッケージもインポートする。これは Python 標準の re モジュールを拡張したもので、事前トークン化に必要な Unicode 文字クラス（任意の文字を表す \p{L} など）をサポートしている。

```
import regex

class BPETokenizer:
    """Byte-level BPE tokenizer trained by iterated most-frequent-pair
    merges.

    Token ids 0..255 are raw bytes; learned merges get ids
    256..vocab_size-1; special tokens sit above those and are never
    produced by BPE itself. `pattern` is an optional pre-tokenization
    regex: text is split into chunks and merges never cross chunk
    boundaries.
    """

    GPT2_PATTERN = (r"'s|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+"
                    r"| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+")

    def __init__(self, vocab_size=1024, pattern=None,
                 specials=('<pad>', '<bos>', '<eos>')):
        self.vocab_size, self.pattern = vocab_size, pattern
        self.merges = {}                                # (id, id) -> new id
        self.vocab = {i: bytes([i]) for i in range(256)}  # id -> bytes
        self.byte_ids = list(range(256))                # byte value -> id
        self.specials = {s: vocab_size + i for i, s in enumerate(specials)}

    def __len__(self):
        return self.vocab_size + len(self.specials)

    @property
    def pad(self):
        return self.specials['<pad>']

    @property
    def bos(self):
        return self.specials['<bos>']

    @property
    def eos(self):
        return self.specials['<eos>']

    def _chunks(self, text):
        return [text] if self.pattern is None else regex.findall(
            self.pattern, text)
```

訓練とエンコードはいずれも、対象となる隣接 ID ペアのすべての出現箇所をマージ後のトークン ID に置き換えるという単一の基本操作に依存している。補助メソッド _merge は左から右へ走査し、new_id を書き込んで位置を 2 つ進める処理を pair が一致するたびに行い、一致しない場合は ID を 1 つコピーする。その結果、文字列 aaa に対してペア (a, a) は重複して適用されるのではなく、左側で 1 回だけ置き換えられる。

```
@d2l.add_to_class(BPETokenizer)
def _merge(self, ids, pair, new_id):
    """Replace every occurrence of pair in ids by new_id."""
    out, i = [], 0
    while i < len(ids):
        if i + 1 < len(ids) and (ids[i], ids[i + 1]) == pair:
            out.append(new_id)  # Merge the pair and skip past it
            i += 2
        else:
            out.append(ids[i])
            i += 1
    return out
```

訓練では、手作業でトレースしたのと同じループを繰り返す。すなわち、すべての隣接ペアを数え上げ、最頻出ペアをすべての箇所でマージし、得られた規則を記録する。トレースのときと同様、出現回数が同数の場合は最初に出現したペアが優先されるが、これは Python の `max` の挙動そのものである。実用上の最適化として、同一のテキストチャンクは 1 回だけ保持し、そのペア出現回数をコーパス内でのチャンクの出現頻度で重み付けしている。事前トークン化を行わない場合、コーパス全体が重み 1 の単一チャンクとなるが、事前トークン化によってテキストが単語に分割されれば、頻出語は何千回も処理される代わりに 1 回だけ処理される。実用レベルの BPE トレーナはさらに踏み込み、各マージの前後で変化した箇所のみペア出現回数を差分更新するが、ここでは明快さを優先して速度とのトレードオフを受け入れ、各ステップの後に単純に数え直す。

```
@d2l.add_to_class(BPETokenizer)
def train(self, text):
    """Learn vocab_size - 256 merges, most frequent pair first."""
    chunk_freq = collections.Counter(self._chunks(text))
    seqs = [list(chunk.encode('utf-8')) for chunk in chunk_freq]
    for new_id in range(256, self.vocab_size):
        pairs = collections.Counter()
        for seq, w in zip(seqs, chunk_freq.values()):
            for pair in zip(seq, seq[1:]):
                pairs[pair] += w  # A chunk that occurs w times counts w times
        if not pairs:
            break  # nothing left to merge
        pair = max(pairs, key=pairs.get)
        self.merges[pair] = new_id
        self.vocab[new_id] = self.vocab[pair[0]] + self.vocab[pair[1]]
        seqs = [self._merge(seq, pair, new_id) for seq in seqs]
```

トイコーパス上で `BPETokenizer` を 3 回のマージ（`vocab_size=256 + 3`）だけ訓練し、`toy_tok.merges` を確認すると、先ほど手作業でトレースした 3 つのマージステップがそのまま再現される。

```
toy_tok = BPETokenizer(vocab_size=256 + 3)
toy_tok.train('hug pug pun bun hugs hug hug pug')
for (a, b), new_id in toy_tok.merges.items():
    print(f'{toy_tok.vocab[a]} + {toy_tok.vocab[b]} -> '
          f'{toy_tok.vocab[new_id]}')
```

```
b'u' + b'g' -> b'ug'
b'h' + b'ug' -> b'hug'
```

```
b'hug' + b' ' -> b'hug '
```

8.2.3.3 エンコードとデコード

訓練によってマージテーブルが得られる。新しいテキストを**エンコード**するには、生の UTF-8 バイト列から始め、テーブル内に該当する隣接ペアがなくなるまで、最もマージランクの低い隣接ペアを繰り返しマージする。注目すべき点として、エンコード時に参照されるのはマージランクのみであり、コーパスでの出現回数は一切参照されない。つまりエンコードは学習済みテーブルの純粋な関数であり、**トークナイザはそのマージランクによって完全に定義される**。各訓練マージはそれ以前のすべてのマージが適用済みのテキスト上で学習されたため、最もランクの低いマージから順に適用することで、訓練時の過程が忠実に再現される（この性質は演習問題で定式化する）。デコードはさらに単純である。すべてのトークン ID はバイト系列へ直接対応付けられるため、バイトを連結して UTF-8 としてデコードすればよい。この際、生成モデル（generative model）が出力した不完全または不正なバイト系列に対して例外を送出するのではなく穏やかに代替文字へフォールバックできるよう、`errors='replace'` を指定する。

```
@d2l.add_to_class(BPETokenizer)
def _encode_chunk(self, text_bytes):
    ids = [self.byte_ids[b] for b in text_bytes]
    while len(ids) > 1:
        # The lowest-rank merge applicable anywhere in this chunk
        pair = min(zip(ids, ids[1:]),
                   key=lambda p: self.merges.get(p, float('inf')))
        if pair not in self.merges:
            break
        ids = self._merge(ids, pair, self.merges[pair])
    return ids

@d2l.add_to_class(BPETokenizer)
def encode(self, text, allow_special=False):
    if allow_special and self.specials:
        pat = '(' + '|'.join(regex.escape(s) for s in self.specials) + ')'
        ids = []
        for part in regex.split(pat, text):
            if part in self.specials:
                ids.append(self.specials[part])
            elif part:
                ids.extend(self.encode(part))
        return ids
    return [i for chunk in self._chunks(text)
            for i in self._encode_chunk(chunk.encode('utf-8'))]

@d2l.add_to_class(BPETokenizer)
def decode(self, ids):
    specials = {i: s.encode('utf-8') for s, i in self.specials.items()}
    data = b''.join(self.vocab[i] if i in self.vocab else specials[i]
                    for i in ids)
    return data.decode('utf-8', errors='replace')
```

トイトークナイザにおいて、エンコード結果は手作業のトレースと一致し、デコードによって元の文字列がきれいに復元される。

```
print([toy_tok.vocab[i] for i in toy_tok.encode('hugs')])
print([toy_tok.vocab[i] for i in toy_tok.encode('bun')])
toy_tok.decode(toy_tok.encode('hug pug hugs'))
```

```
[b'hug', b's']
[b'b', b'u', b'n']
```

```
'hug pug hugs'
```

8.2.3.4 『The Time Machine』での訓練

実際の散文から BPE が何を学習するかを確認するため、『The Time Machine』で訓練してみよう。トークンのリストや生成された文章は印刷ページの幅を超えることが多いため、まず長い行を見やすく折り返す小さな表示用補助関数を定義し、ここだけでなく本書の以降の箇所でも利用する。

```
import textwrap

def print_wrapped(*args, width=76):
    """Print like `print`, folding each line to the width of a page."""
    for line in ' '.join(str(a) for a in args).split('\n'):
        print(textwrap.fill(line, width, subsequent_indent='    '))
```

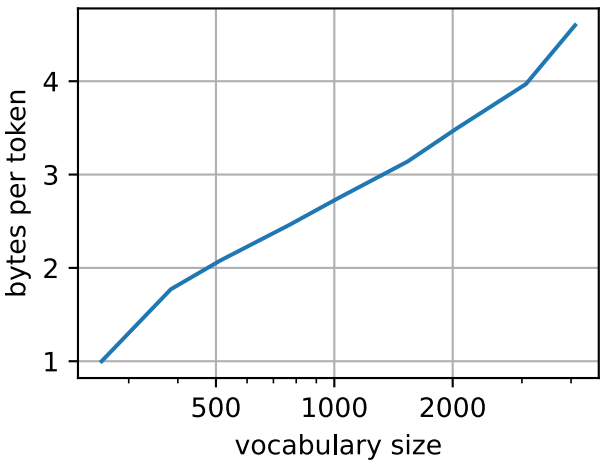
それでは、正規化していないこの中編小説の生テキストから直接、1,024 トークンの語彙を訓練する。初期のマージは頻出する文字ペアを捉えるのに対し、終盤のマージは Wells の散文に登場する単語全体を組み立てるようになる。ただし、以下のエンコードされたサンプルの中に、空白まで含めて 2 単語が結合した複合トークン `"Time Traveller "` が含まれている点に注目してほしい。この現象については次で詳しく調べる。

```
tok = BPETokenizer(vocab_size=1024)
tok.train(raw_text)
merged_ids = list(tok.merges.values())
print('first:', [tok.vocab[i] for i in merged_ids[:8]])
print('last: ', [tok.vocab[i] for i in merged_ids[-8:]])
ids = tok.encode('The Time Traveller smiled. Are you sure?')
print_wrapped(len(ids), 'tokens:', [tok.vocab[i] for i in ids])
```

```
first: [b'e ', b'th', b'd ', b'in', b't ', b's ', b'an', b'er']
last:  [b'appe', b'vo', b'ering ', b'too ', b'door', b'acro', b'clo', b'after ']
13 tokens: [b'The ', b'Time Traveller ', b's', b'mi', b'le', b'd', b'. A',
    b'r', b'e ', b'you ', b'su', b're', b'?']
```

語彙を大きくすると、系列はどれだけ短くなるだろうか。マージには学習された順序どおりに番号が振られるため、同じコーパスで訓練されたより小さな BPE 語彙は、より大きな語彙の**接頭辞**（prefix）となる。すなわち、ID が $|\mathcal{V}|$ 未満のマージだけを残せば、サイズ $|\mathcal{V}|$ で訓練を打ち切ったときに得られるトークナイザが厳密に再現される。したがって、1 回の訓練を実行するだけで、語彙サイズを 256 から 4,096 まで変化させて調べることができる。以下では、中編小説の先頭 10,000 文字に対して 1 トークンあたりのバイト数で圧縮率を測定する（演習問題 5 ではホールドアウトされたテキストで評価する）。

```
big = BPETokenizer(vocab_size=4096)
big.train(raw_text)
excerpt = raw_text[:10000]
sizes = [256, 384, 512, 768, 1024, 1536, 2048, 3072, 4096]
ratios = []
for v in sizes:
    sub = BPETokenizer(vocab_size=v)  # the first v - 256 merges of big
    sub.merges = {p: i for p, i in big.merges.items() if i < v}
    sub.vocab = {i: b for i, b in big.vocab.items() if i < v}
    ratios.append(len(excerpt.encode('utf-8')) / len(sub.encode(excerpt)))
d2l.plot(sizes, [ratios], 'vocabulary size', 'bytes per token',
         xscale='log')
```



語彙を 2 倍にするごとにほぼ一定の圧縮率向上が得られるため、1 トークンあたりのバイト数は語彙サイズに対しておよそ対数的に増加する。このスケーリングは貪欲なマージ規則から直接導かれる。`"e "`、`"th"`、`"d "` のような初期のマージはそれぞれ数千箇所の出現を一気に縮約するのに対し、4,000 番目のマージはわずか数箇所しか縮約しないからである。ただし、この曲線を 50,000〜200,000 トークンの実用語彙へ無批判に外挿しないよう注意が必要である。プロットの上端が良く見えているのは、トークナイザの訓練に使ったのと同じテキスト上で評価しているためであり、そこでは 4,096 トークンの語彙が『The Time Machine』固有の単語やフレーズを記憶してしまっている。一般的な英語の散文では、後ほど検証するように実用語彙はおおむね 1 トークンあたり 4〜5 バイト（およそ 1 単語あたり 1 トークン）となり、余剰の語彙容量はより低頻度な単語やソースコード、そして（`o200k_base` のような多言語語彙では）他言語のために使われる。

8.2.3.5 事前トークン化

複合トークン `"Time Traveller "` は決して例外的な事例ではない。制約のないこのトークナイザは、本来の単語断片と並んで、単語境界をまたぐトークンを何十個も蓄積してしまう。

```
crossers = [v.decode('utf-8', 'replace') for i, v in tok.vocab.items()
            if i >= 256 and b' ' in v.strip(b' ')]
print_wrapped(len(crossers), 'tokens cross a word boundary, e.g.',
              crossers[:8])
```

```
78 tokens cross a word boundary, e.g. [', and ', 'of the ', '. I ', '. Th',
    '. A', '. I', 'in the ', 'ed to ']
```

"of the " のような複数単語のマージは訓練テキストの圧縮には役立つものの、脆い複合表現である。固定フレーズに語彙枠を費やす一方で、構成単語の統計的な手がかりを分断してしまうからである（"of" が多数の異なるトークンに分散して出現するようになる）。そのため、すべての実用 BPE トークナイザはマージの前に**事前トークン化**（pre-tokenization）の段階を挿入する。まず正規表現（regular expression）によって入力文字列をチャンクに分割し、チャンク境界を越えてマージが行われないよう、厳密に各チャンクの**内部**でのみ BPE を訓練・適用する。GPT-2 のパターン (Radford et al. 2019) は `BPETokenizer.GPT2_PATTERN` として格納されており、次の 5 つの分岐から構成される。

```
's|'t|'re|'ve|'m|'ll|'d        common English contractions, split off
| ?\p{L}+                      a run of letters, with its leading space
| ?\p{N}+                      a run of digits, with its leading space
| ?[^\s\p{L}\p{N}]+            a run of punctuation or other symbols
|\s+(?!\S)|\s+                 remaining whitespace
```

このパターンには 3 つの意図的な規約が組み込まれている。第 1 に、空白が単語自体と一体になって移動するよう、各単語はその**先頭の空白**を保持する（" the" であり、"the " ではない）。第 2 に、数値や記号が決して単語と融合しないよう、文字・数字・句読点は別々のチャンクに分離される。第 3 に、英語の一般的な短縮形は独立したチャンクとして切り離される。

```
regex.findall(BPETokenizer.GPT2_PATTERN,
              "The traveller's clock struck 12,345 times.")
```

```
['The',
 ' traveller',
 "'s",
 ' clock',
 ' struck',
 ' 12',
 ',',
 '345',
 ' times',
 '.']
```

事前トークン化を有効にするには、`pattern` を `BPETokenizer` に渡すだけでよい。Figure 8.5 に、生テキストからトークン ID までのパイプライン全体を示す。制約なしの実行結果と比べると、単語境界をまたぐ複合トークンはすべて消失し、解放された語彙枠は先頭の空白を伴う完全な単語へと割り当てられる。また、初期のマージは頻出文字を先頭の空白に結合するもの（" t"、" a"）や、頻出する文字ペアを結合するもの（"he"、"in"）になる。

```
bpe = BPETokenizer(vocab_size=1024, pattern=BPETokenizer.GPT2_PATTERN)
bpe.train(raw_text)
crossers = [v for i, v in bpe.vocab.items()
            if i >= 256 and b' ' in v.strip(b' ')]
print('boundary-crossing tokens now:', len(crossers))
gained = sorted(set(bpe.vocab.values()) - set(tok.vocab.values()),
                key=lambda v: (-len(v), v))
print_wrapped('longest tokens gained instead:',
              [v.decode('utf-8', 'replace') for v in gained[:6]])
print('first merges:', [bpe.vocab[i] for i in list(bpe.merges.values())[:6]])
```

```
boundary-crossing tokens now: 0
longest tokens gained instead: [' Psychologist', 'sychologist', '
    Traveller', ' Morlocks', ' darkness', ' Machine']
first merges: [b' t', b'he', b' a', b'in', b' the', b' s']
```

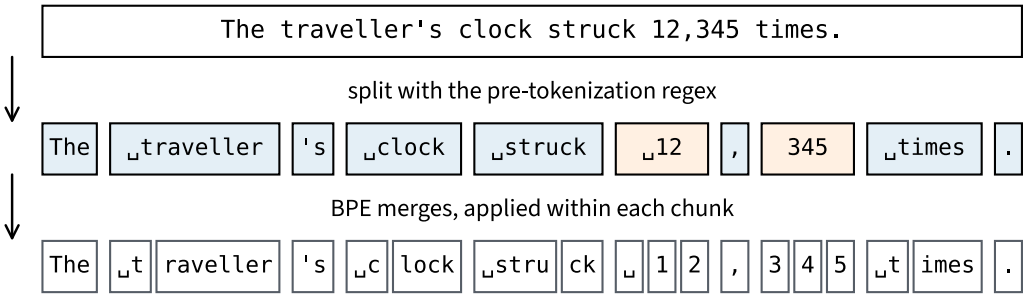


Figure 8.5: 実際の文に対する事前トークン化の動作。正規表現がまずテキストを、先頭の空白を伴う単語、数字の並び、句読点というチャンクに切り分ける。続いて各チャンクの内部で BPE マージが適用され、チャンク境界をまたぐことは決してない。

8.2.4 特殊トークン

訓練やサービングのパイプラインでは、テキストそのものではなく**インフラストラクチャ**にとって意味を持つ少数の制御トークンも必要になる。<pad> は短い系列を共通のバッチ長に揃えるためにパディングし、<bos> は系列の先頭を表し、<eos> は生成が完了したことをモデルが通知するために使われる（8.6 節のデコーダはこれを停止の合図として監視する）。`BPETokenizer` では、これらの特殊 ID を BPE の範囲よりも**上**に配置している。これは、通常のテキストが誤って特殊トークンへとエンコードされるのを防ぐための意図的な安全上の不変条件（invariant）である。そうでなければ、"<eos>" という文字そのものを含む文書やユーザーメッセージがモデルの入力ストリームへ制御信号を直接注入できてしまい、信頼できないユーザーテキストがシステム命令として解釈される**プロンプトインジェクション**の典型的な攻撃経路となってしまう。そのため、`encode` メソッドは呼び出し側が明示的に `allow_special=True` を渡さない限り特殊トークン文字列を通常のバイト列として扱い、`tiktoken` の `allowed_special` ガード（デフォルトではエラーを送出する）と同様の防御を実現している。

```
    print('as plain text:  ', bpe.encode('<bos>the time machine'))
    print('as control code:', bpe.encode('<bos>the time machine',
                                         allow_special=True))
    print('pad/bos/eos ids:', bpe.pad, bpe.bos, bpe.eos, '| len:', len(bpe))
```

```
as plain text:   [60, 98, 111, 115, 62, 341, 513, 680]
as control code: [1025, 341, 513, 680]
pad/bos/eos ids: 1024 1025 1026 | len: 1027
```

8.2.5 実世界のトークナイザ

8.2.5.1 実用トークナイザのランクテーブル

GPT 系やバージョン 3 以降の Llama をはじめ、広く運用されている言語モデルの大半は、これとまったく同じ仕組みで構築され、はるかに大規模なコーパスで訓練されたバイトレベル BPE ト

ークナイザを採用している。GPT-2 のトークナイザ (Radford et al. 2019) は、各トークンのバイト系列をそのマージランクへ対応付けるルックアップテーブルに、事前トークン化用の正規表現と特殊トークンの登録情報を組み合わせた形でコンパクトに保存されており、その内部にニューラルネットワークは存在しない（WordPiece や Unigram などの他のサブワード手法では、これに加えてテキスト正規化規則や、競合する分割候補の中から選択するためのトークン確率も保持する）。OpenAI の `tiktoken` ライブラリには、GPT-2 のランクテーブルがそのまま同梱されている。以下では、単一バイトと学習済みマージの境界をまたぐ項目を確認してみよう。なお、`_mergeable_ranks` の先頭にあるアンダースコアは内部属性であることを示しており、ライブラリのリリースによって名前が変わる可能性がある。

```
enc = tiktoken.get_encoding('gpt2')
print(list(enc._mergeable_ranks.items())[254:260])
print(enc._special_tokens)
```

```
[(b'\xa0', 254), (b'\xad', 255), (b' t', 256), (b' a', 257), (b'he', 258), (b'in', 259)]
{'<|endoftext|>': 50256}
```

ランク 0〜255 は並び替えられた順序での個々のバイトに対応し（GPT-2 は印字可能な ASCII 文字を先に並べている）、ランク 256 以降には学習されたマージが訓練順に続く。これは先ほど作成した `BPETokenizer` とまったく同じ構成である。驚くべきことに、40 GB のウェブテキストから学習された GPT-2 の最初の 4 回のマージ（`" t"`、`" a"`、`"he"`、`"in"`）は、19 世紀の 1 編の中編小説から `bpe` が学習した最初の 4 回のマージと完全に一致しており、英語における最頻出の文字パターンの安定性を物語っている。

8.2.5.2 実装の検証

BPE トークナイザはそのランクテーブルによって完全に規定されるため、公開されている GPT-2 のランクをそのまま自作の `BPETokenizer` エンコーダに読み込むことができる。各ランクがそのままトークン ID となり、256 個の単一バイト項目がバイトの置換順序を定義する。公開されているテーブルには各トークンを形成した親ペアではなく最終的なバイト列のみが記録されているため、各トークンのバイト列に対してよりランクの低いマージをすべて再生し、ちょうど 2 つの断片が残るまで縮約することで各マージペアを再構築する。以下の再構築の詳細は流し読みして構わない。重要なのはその後に続く検証用のセルである。

```
@d2l.add_to_class(BPETokenizer)
@classmethod
def from_tiktoken(cls, name):
    """Load a published bytes->rank table (e.g. 'gpt2') into our encoder."""
    import tiktoken  # lazy: d2l itself does not require tiktoken
    enc = tiktoken.get_encoding(name)
    ranks = enc._mergeable_ranks
    tok = cls(vocab_size=len(ranks), pattern=enc._pat_str, specials=())
    tok.specials = dict(enc._special_tokens)
    tok.vocab = {rank: b for b, rank in ranks.items()}
    tok.byte_ids = [ranks[bytes([b])] for b in range(256)]
    for token, rank in ranks.items():
        if len(token) > 1:  # recover which pair merged into this token
            parts = [bytes([b]) for b in token]
            while len(parts) > 2:
                a, b = min(zip(parts, parts[1:]),
                           key=lambda p: ranks.get(p[0] + p[1],
```

```
                                                float('inf')))
                    i = list(zip(parts, parts[1:])).index((a, b))
                    parts[i:i + 2] = [a + b]
                tok.merges[ranks[parts[0]], ranks[parts[1]]] = rank
    return tok
```

散文、数値、アクセント付き文字、通貨記号、絵文字を織り交ぜたテスト段落において、このエンコーダは tiktoken とトークン単位で完全に一致する。このアサーションは本書がビルドされるたびに自動的に検証されている。

```
ours = BPETokenizer.from_tiktoken('gpt2')
para = ('The Time Traveller (for so it will be convenient to speak of '
        'him) was expounding a recondite matter to us; his pale grey '
        'eyes shone and twinkled. Prices rose 3.5% in 1895, and naïve '
        'café patrons paid £12,345 more!  ')
assert ours.encode(para) == enc.encode(para)
print(len(ours.encode(para)), 'tokens, identical to tiktoken')
print([ours.vocab[i].decode('utf-8', 'replace')
       for i in ours.encode(para)[:8]])
```

```
60 tokens, identical to tiktoken
['The', ' Time', ' Trave', 'ller', ' (', 'for', ' so', ' it']
```

大規模な実用語彙は『The Time Machine』をどの程度うまく圧縮できるだろうか。以下では、ここで作成した 1,024 トークンのトークナイザを、GPT-2 の 50,257 トークンの語彙、および GPT-4o で採用されている約 200,000 トークンの語彙 o200k_base と比較する。

```
rows = [('ours (1k, in-domain)', len(bpe.encode(raw_text)))]
for name in ('gpt2', 'o200k_base'):
    e = tiktoken.get_encoding(name)
    rows.append((f'{name} ({len(e._mergeable_ranks) // 1000}k)',
                 len(e.encode(raw_text))))
for name, n in rows:
    print(f'{name:>22}: {n:>6} tokens, {n_bytes / n:.2f} bytes/token')
```

```
  ours (1k, in-domain):  65741 tokens, 2.72 bytes/token
           gpt2 (50k):  44073 tokens, 4.06 bytes/token
    o200k_base (199k):  42624 tokens, 4.20 bytes/token
```

ここで作成した語彙はサイズが 50〜200 分の 1 にすぎないにもかかわらず、この中編小説に特化して訓練されているため、実用語彙のおよそ 3 分の 2 の圧縮率を達成している。他のあらゆる統計モデルと同様、トークナイザもその訓練分布（training distribution）に対する圧縮モデルであり、構築元のコーパスに似たテキストに対して最も高い性能を発揮する。

8.2.5.3 肥沃度、数字、グリッチトークン

ある言語においてトークナイザが 1 単語あたりに費やす平均トークン数は、**肥沃度**（fertility）と呼ばれる。GPT-2 のトークナイザは主に英語のウェブテキストで訓練されたため、まったく同じ文を 3 つの言語に翻訳すると、トークン数に際立った差が生じる。

```
sentences = {
    'English': 'The time traveller returned after many years.',
    'German': 'Der Zeitreisende kehrte nach vielen Jahren zurück.',
    'Greek': 'Ο ταξιδιώτης του χρόνου επέστρεψε μετά από πολλά χρόνια.'}
for lang, s in sentences.items():
    gpt2, o200k = [len(tiktoken.get_encoding(e).encode(s)) / len(s.split())
                   for e in ('gpt2', 'o200k_base')]
    print(f'{lang:>8}: {gpt2:.2f} (gpt2), {o200k:.2f} (o200k_base) '
          'tokens per word')
```

```
English: 1.14 (gpt2), 1.14 (o200k_base) tokens per word
 German: 2.86 (gpt2), 1.57 (o200k_base) tokens per word
  Greek: 7.33 (gpt2), 1.89 (o200k_base) tokens per word
```

GPT-2 のトークナイザでは、英単語が 1 語あたり約 1 トークンであるのに対し、ギリシャ語の単語は約 7 トークンも消費する（ギリシャ文字は 1 文字あたり UTF-8 で 2 バイトを要するうえ、ギリシャ語のマージがほとんど学習されなかったためである）。その結果、1,024 トークンのコンテキストウィンドウには英語なら約 900 語収まるのに対し、ギリシャ語は約 140 語しか収まらず、英語圏以外のユーザーにとっては推論がより低速かつ高コストになる。対照的に、多言語対応の語彙 `o200k_base` はギリシャ語の肥沃度を 1 語あたり約 2 トークンまで引き下げる。

数値の例を見ると、トークン化の微妙な仕様がいかにモデルの学習対象を左右するかがわかる。GPT-2 は BPE がたまたま頻出断片を見つけた位置で数字の並びを分割するのに対し、より新しいトークナイザは分割をより均一にするため、事前トークン化の段階で数字チャンクの上限を 3 文字に制限している。

```
for name in ('gpt2', 'o200k_base'):
    e = tiktoken.get_encoding(name)
    print(f'{name}: 12345678 ->',
          [e.decode([t]) for t in e.encode('12345678')])
```

```
gpt2: 12345678 -> ['123', '45', '678']
o200k_base: 12345678 -> ['123', '456', '78']
```

それでもなお、いずれの分割も、右から左へ一の位・十の位・百の位と桁を束ねる位取り記数法の筆算の列とは一致しない。左から右へ 3 桁ずつ区切る上限のもとでは、`"12345678"` は `"123"`、`"456"`、`"78"` へと分割されるため、言語モデルは桁のずれた境界をまたいで繰り上がりを処理しなければならなくなる。この不一致は言語モデルにおける算術誤りのよく知られた一因であり、標準的な 3 桁区切りのカンマ（`"12,345,678"`）を挿入すると数値タスクが容易になり得る理由の説明にもなっている。

最後に、語彙項目がどれほど信頼できるかは、訓練中にモデルがそのトークンにどれだけ触れたかに依存する。初期の GPT-2 や GPT-3 の語彙には `" SolidGoldMagikarp"` のようなトークンが含まれていた。これは Reddit のユーザー名であり、**トークナイザ**の訓練コーパスには専用のマージを獲得するほど頻繁に出現したにもかかわらず、**言語モデル**の事前学習データにはほとんど含まれていなかったものである。このような**グリッチトークン**は、その埋め込みベクトルが勾配更新をほとんど受けていないにもかかわらず、ユーザーのプロンプトから生成され得る。初期のモデルにこのトークンを繰り返すよう促すと、しばしばはぐらかすような応答や無関係な単語を返した。

```
enc.encode(' SolidGoldMagikarp')  # one token for an 18-character string
```

[43453]

8.2.6 まとめ

トークナイザは、生のテキストとニューラルネットワークが受け取る整数系列との橋渡しをする。語彙サイズは系列長と出力層サイズの間の基本的なトレードオフを支配し、さらに固定された単語語彙は未知語に対して破綻する。バイトレベル BPE は 256 個の単一バイト記号から出発することで語彙外の入力を完全に排除し、目標の語彙サイズに達するまで最頻出の隣接トークンペアを貪欲にマージする。エンコードは学習済みマージを厳密にランク順で再生するため、マージテーブルだけでトークナイザが定義される。実用上は、事前トークン化の正規表現によってマージが単語境界をまたぐのを防ぎ、通常のテキストから合成されないよう特殊な制御トークンには BPE の範囲より上の ID を割り当てる。GPT-2 の実用トークナイザはこの構成をより大規模にそのまま踏襲したものであり、本節の実装はそれをトークン単位で再現する。言語ごとに異なる肥沃度、数字分割の癖、グリッチトークンといった現象はいずれも、訓練データとマージ規則から直接生じた帰結である。次節では、トークナイザが生成するトークン系列のモデル化へと進む。

8.2.7 演習問題

1. **ランクによるエンコード**。`BPETokenizer`（`vocab_size=258`）が、まず ID 256 として (b, c)、次に ID 257 として (a, b) というちょうど 2 つのマージを学習したとする。`_encode_chunk` の**ランク規則**は常に、適用可能なマージのうち最もランクの低いものを適用する。一方、**最左規則**は常に、マージテーブルに存在する隣接ペアのうち最も左にあるものをマージする。

    1. 2 つの規則のそれぞれのもとで、文字列「abc」を分割せよ。
    2. 訓練終了時点において、訓練コーパス内のすべての「abc」の出現箇所がどの分割になっていたかを示し、どちらの規則がそれと一致するかを答えよ。

2. [code] **特殊トークンの往復変換**。通常のテキストから決して生成されないよう、8.2.4 節では特殊トークンを BPE の ID 範囲より上に配置した。`BPETokenizer` を、`specials` がデフォルトの 3 つに加えて文書区切り `<sep>` を含むように構築し、`raw_text` 上で訓練せよ。`x` を、2 つの短い文書を `<sep>` で連結した文字列とする。

    1. `x` を `allow_special=True` ありと、なしのそれぞれで 1 回ずつエンコードし、両方の ID 系列が `x` にデコードされることを確認せよ。
    2. デコードされたテキストを `allow_special=True` ありと、なしのそれぞれで 1 回ずつ再エンコードせよ。それぞれの再エンコードは、前の小問で得られた 2 つの ID 系列のうちどちらを再現するか。モデルの出力をデコードし、後でそのテキストを再び入力として与えるパイプラインにとって、これが何を意味するかを述べよ。
    3. 特殊文字列の混入が許可されていないテキストにその文字列が現れたとき、`tiktoken` はエラーを送出するのに対し、`encode` はその文字列を通常のバイト列として扱う。それぞれの方針が望ましい状況を 1 つずつ挙げよ。

    Andrej Karpathy の minbpe、Step 4 を改変。

3. [code] **数字のチャンク化**。加算プロンプト「1234+5678=」、「1234+567=」、「567+678=」およびそれぞれの和「6912」、「1801」、「1245」を、`gpt2` と `o200k_base` の各エンコーディング（`tiktoken`）でトークン化し、各数値のトークン境界を書き出せ。

    1. `o200k_base` の 3 桁上限のもとで、各数字トークンが担う位取りの値（一の位、十の位、百の位、千の位）をラベル付けせよ。各プロンプトについて、足される 2 つの数と和の対応する位置にあるトークンが同じ位取りの値を担っているかどうかを述べよ。また、同じでない場合、トークン単位で加算するモデルにどのような問題が生じるかを説明せよ。

   2. トークンを位取りの値と揃える事前トークン化規則を 1 つ提案し、その規則の欠点を 1 つ述べよ。

4. **グリッチトークン**。グリッチトークンの原因について、8.2.5.3 節ではトークナイザの訓練コーパスでは頻出したものの言語モデルの訓練コーパスではまれであった文字列に帰着させた。

   1. なぜ `BPETokenizer.train` のマージ規則はそのような文字列を複数の断片ではなく単一のトークンにしてしまうのか。また、事前トークン化はそのトークンの範囲を確定するうえでどのような役割を果たすか。
   2. 訓練済みの埋め込みテーブルとモデルの訓練コーパスが与えられたとき、グリッチトークンを検出するテストを 1 つ提案し、2 つの入力がそれぞれ何に寄与するかを述べよ。

5. [code] **事前トークン化のアブレーション**。●`raw_text` を前半の 90% と後半の 10% に分割せよ。前半部分を用いて、パターンの有無のみが異なる 4,096 トークンのトークナイザを 2 つ訓練せよ（1 つは `big` と同様にパターンなし、もう 1 つは `BPETokenizer.GPT2_PATTERN` ありとする）。

   1. 各トークナイザが学習した最も長い 20 個のトークンを列挙し、複数の単語にまたがるものに印を付けよ。
   2. 2 つのうちどちらが前半部分をより高い圧縮率で圧縮するかを測定し、その結果をマージ規則の観点から説明せよ。
   3. 後半の 10% に対して両トークナイザの 1 トークンあたりのバイト数を測定せよ。このホールドアウトされたテキストにおいて、圧縮率の低下割合が大きいのはどちらのトークナイザか。また、より高い圧縮率を達成するのはどちらか。両方の答えを、先ほど列挙したトークンと関連付けて説明せよ。

8.3 言語モデル

次の単語を提案するスマートフォンのキーボードも、話者が「to recognize speech」と言ったのか「to wreck a nice beach」と言ったのかを判定する音声認識器も、同じ能力に依存している。それは、あるテキストの一節がどれほどもっともらしいかを表す数値スコアを割り当てる能力である。これら 2 つのフレーズは発音がほぼ同じであるため、音響情報だけでは区別できない。曖昧さを解消するのは、前者が自然な英語表現であるのに対し、後者が奇抜で不自然だという知識である。**言語モデル**（language model）はこの判断を担う。すなわち、トークン系列 $x_1, \dots, x_T$（語彙 $\mathcal{V}$ から選ばれた、たとえば 8.2 節で構築した BPE トークンなど）が与えられたとき、その同時確率（joint probability）$P(x_1, \dots, x_T)$ を推定する。

一見すると、起こり得るすべての系列に確率を割り当てるのは絶望的に思える。1,024 トークンという控えめな語彙であっても、長さ 10 の系列はすでに $1024^{10} \approx 10^{30}$ 通りも存在するからである。幸いなことに、確率の連鎖律（chain rule、8.1 節）を使えば、この直接扱えない同時分布（joint distribution）を近似なしで次トークンの条件付き確率の積へと変換できる。各項は、直前までのプレフィックス $x_{<t} = x_1, \dots, x_{t-1}$ で条件付けられる。

$$P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t}). \tag{8.6}$$

したがって、言語モデルは本質的に次トークン予測器である。同じ条件付き分布（conditional distribution）を用いることで、既存の系列に**スコアを付ける**こと（確率を掛け合わせる、またはその対数を足し合わせる）も、新しいテキストを**生成する**こと（まず x_1 をサンプリングし、次に

x_2 を直前の x_1 のもとでサンプリングし、段階的に続けていく）も可能になる。注目すべきことに、幅広いタスクをテキストの続きを書く問題としてそのまま定式化できる。たとえば、「The novella The Time Machine was written by」というプロンプトの続きを補完すれば事実に関する質問への回答となり、これこそが現代の大規模言語モデル（large language model、11.2 節）の根底にあるパラダイムである。

まずは、こうした条件付き確率（conditional probability）に対する最も単純な推定量、すなわちコーパス内での出現回数を数える方法から始め、それがなぜ破綻するのかを調べる。改善を客観的に評価するため、パープレキシティを導入して 2 つの古典的な平滑化手法を比較し、続いて異なるトークナイザを持つモデル同士を公平に比較できるようバイトあたりビット数を定義する。最後に、トークン化された系列を次節以降のニューラル言語モデルに供給するミニバッチデータローダを構築する。

```
import collections
import random
import re
```

8.3.1 n グラム言語モデル

8.3.1.1 マルコフモデルとカウント

Equation 8.6 の各因子はプレフィックス全体 $x_{<t}$ に条件付けられているため、t が大きくなるにつれて異なる文脈の数は際限なく増大する。そこでマルコフ条件（8.1.3.1 節）を導入し、文脈を直近の $n-1$ 個のトークンに打ち切る。

$$P(x_t \mid x_{<t}) \approx P(x_t \mid x_{t-n+1}, \dots, x_{t-1}). \tag{8.7}$$

この仮定が粗いことは否定できないが（文が 2 語進んだだけで主語を忘れることはまずない）、取り得る文脈の集合を有限個に絞ることができる。文脈を持たないモデル（$n=1$）は**ユニグラム**モデル、直前の 1 トークンのみに条件付けるモデル（$n=2$）は**バイグラム**モデル、直前の 2 トークンに条件付けるモデル（$n=3$）は**トライグラム**モデルと呼ばれる。最尤推定（maximum likelihood estimation）のもとでは、各条件付き確率はコーパス内の出現回数の比にほかならない。トークンの組が訓練コーパスに出現する回数を $N(\cdot)$ と書けば、次式を得る。

$$\hat{P}(x_t \mid x_{t-n+1}, \dots, x_{t-1}) = \frac{N(x_{t-n+1}, \dots, x_t)}{N(x_{t-n+1}, \dots, x_{t-1})}, \tag{8.8}$$

これは、文脈 $(x_{t-n+1}, \dots, x_{t-1})$ の直後に x_t が続いた経験的な割合である。n グラム言語モデルとは、こうした条件付き相対頻度のルックアップテーブルにすぎない。

『The Time Machine』を用いてユニグラム、バイグラム、トライグラムのテーブルを推定してみよう。本章の後半で扱うニューラル言語モデルは BPE トークンを対象に動作するが、ここでは生成された文章を目視で確認しやすいよう、この中編小説を単語単位でトークン化する。8.2 節の単純な小文字英字への正規化を再利用し、空白で分割したうえで、テキストの末尾 10% を評価用にホールドアウトする。語彙は訓練分割に出現するすべての単語からなり、訓練時に出現しなかったホールドアウト内の単語はすべて未知トークン（unknown token）<unk> に写像する。

```
fname = d2l.download(d2l.DATA_URL + 'timemachine.txt',
                     sha1_hash='090b5e7e70c295757f55df93cb0a180b9691891a')
```

```
raw_text = open(fname).read()
text = re.sub('[^A-Za-z]+', ' ', raw_text).lower()
split = text.rindex(' ', 0, int(0.9 * len(text)))
words_train = text[:split].split()
known = set(words_train)  # A held-out word never seen in training is <unk>
words_val = [w if w in known else '<unk>' for w in text[split:].split()]
len(words_train), len(words_val)
```

```
(29402, 3373)
```

以下の NGramLM クラスは、観測されたすべての $(n-1)$ トークンの文脈に対し、訓練中にその直後に続いたトークンのカウンタを保持する。これだけで Equation 8.8 を計算するのに必要な情報がすべて揃う（alpha 属性は、後ほど平滑化を導入する際に使用する）。また、sample メソッドは与えられたプレフィックスを延長するため、直近 $n-1$ 個のトークンのもとでの出現回数に比例して次のトークンを繰り返し抽出する。

```
class NGramLM:
    """An n-gram language model estimated from counts."""
    def __init__(self, tokens, n, vocab_size, alpha=0.01):
        self.n, self.vocab_size, self.alpha = n, vocab_size, alpha
        cont = collections.defaultdict(collections.Counter)
        for gram in zip(*(tokens[i:] for i in range(n))):
            cont[gram[:-1]][gram[-1]] += 1
        self.cont = dict(cont)
        self.total = {ctx: sum(c.values()) for ctx, c in self.cont.items()}

    def sample(self, prefix, num_tokens, rng):
        """Extend prefix by sampling from the unsmoothed count model."""
        out = list(prefix)
        for _ in range(num_tokens):
            ctx = tuple(out[len(out) - self.n + 1:])
            counts = self.cont[ctx]
            out.append(rng.choices(list(counts), counts.values())[0])
        return out

vocab_size = len(known) + 1  # The training words and <unk>
models = {n: NGramLM(words_train, n, vocab_size) for n in (1, 2, 3)}
```

これらのテーブルが実際に何を捉えているかを調べるため、タイムトラベルを扱ったこの中編小説全体で頻繁に登場する文脈 "the time" について、トライグラムモデルにクエリを実行してみよう。

```
trigram = models[3]
total = trigram.total[('the', 'time')]
for word, count in trigram.cont[('the', 'time')].most_common(3):
    print(f'P({word} | the time) = {count}/{total} = {count / total:.2f}')
single = sum(len(c) == 1 for c in trigram.cont.values()) / len(trigram.cont)
print(f'word pairs followed by a single word in training: {single:.0%}')
```

```
P(traveller | the time) = 47/87 = 0.54
P(machine | the time) = 27/87 = 0.31
P(i | the time) = 6/87 = 0.07
```

```
word pairs followed by a single word in training: 84%
```

訓練分割において "the time" は 87 回出現し、その直後には約半数の割合で "traveller" が、約 3 分の 1 の割合で "machine" が続いており、このテーブルが作品の核となるフレーズを的確に捉えていることがわかる。しかし、このような高頻度の文脈はまれな例外にすぎない。観測された単語ペアの 5 分の 4 以上では、訓練テーブルに記録された後続語がただ 1 種類しかないのである。

8.3.1.2 n グラムモデルからのサンプリング

テキストの生成には Equation 8.6 をそのまま適用する。すなわち、そこまでのプレフィックスを条件とする条件付き分布から次のトークンを引き、末尾に追加する操作を繰り返す。もっとも、一見流暢な出力であっても丸暗記の丸写しが覆い隠されていることが少なくないため、生成サンプルとあわせて診断関数 longest_copy を用いる。この関数は、生成された単語列のうち訓練セット内に一言一句そのまま現れる最長の連続区間を測定する。

```
def longest_copy(words, train_words):
    """Length of the longest run of words that also occurs in train_words."""
    corpus, best = ' ' + ' '.join(train_words) + ' ', 0
    for i in range(len(words)):
        while (i + best < len(words) and
               ' ' + ' '.join(words[i:i + best + 1]) + ' ' in corpus):
            best += 1
    return best
```

各モデルにプレフィックス "the time" をシードとして与えて 25 語を生成する。実行するたびに新しいサンプルが得られるよう、乱数生成器（random number generator）にはシードを固定しないでおく。

```
rng = random.Random()
for n, model in models.items():
    sample = model.sample(['the', 'time'], 25, rng)
    copied = longest_copy(sample, words_train)
    d2l.print_wrapped(f'{n}-gram (longest copied run: {copied} words): '
                      + ' '.join(sample))
```

```
1-gram (longest copied run: 2 words): the time we time it and my toil winged
    first was shape the it the came again in steadily once down of s flash a
    that of
2-gram (longest copied run: 4 words): the time machine looking for it was no
    specialist in fact is no vestige of the peculiar shrinking was a ghastly
    half bleached colour swim before us
3-gram (longest copied run: 7 words): the time traveller s absence and i did
    not trace this line i trace with my last view of our old world savage
    animal only the vaguest
```

ユニグラムモデルは文法的な一貫性なしにコーパス内の頻度に従って単語を出力し、バイグラムモデルは各単語が直前の 1 語のみに依存するため、局所的にはもっともらしいペアをつなぎ合わせながらもあてどなくさまよう。一方、トライグラムのサンプルは早くも驚くほど H. G. Wells らしく聞こえる。

しかし、30,000 語のコーパスにおいてトライグラムが Wells らしく聞こえるのは、その大部分が文字どおり Wells の文章**そのもの**だからである。コピーされた最長連続区間を見ればこの振る舞いは一目瞭然であり、ユニグラムでは 2〜3 語、バイグラムでは数語であるのに対し、トライグラ

ムでは典型的に 7〜12 語もの連続した単語に達する。サンプラが訓練時に後続語が 1 種類しかなかった単語ペアに踏み込むたびに、モデルは単一の決定論的な経路へと強制され、小説の節全体をそのまま再生してしまう。モデルは汎化（generalization）しているのではなく、記憶した断片をつぎはぎしているのである（もっとも、数十億語規模で推定すれば、n グラムテーブルは英語の豊かな局所的統語構造を確かに捉えるため、何十年にもわたり音声認識や統計的機械翻訳を支えてきた）。

8.3.1.3 カウントのスパース性

この丸暗記は、より根深い統計的ボトルネックが表面化した症状である。**取り得る** n グラムの総数は $|\mathcal{V}|^n$ であり、n に対して指数関数的に増大するのに対し、コーパス内に実際に現れる n グラムの数はコーパス長に対して線形にしか増えない。結果として生じるスパース性は、ホールドアウトの n グラムのうち訓練時に 1 度も出現しなかったものの割合を調べることで直接測定できる。

```
for n in (1, 2, 3):
    seen = set(zip(*(words_train[i:] for i in range(n))))
    grams = list(zip(*(words_val[i:] for i in range(n))))
    rate = sum(gram not in seen for gram in grams) / len(grams)
    print(f'unseen held-out {n}-grams: {rate:.1%} '
          f'({len(seen)} distinct in training)')
```

```
unseen held-out 1-grams: 8.4% (4315 distinct in training)
unseen held-out 2-grams: 51.5% (18535 distinct in training)
unseen held-out 3-grams: 87.0% (26982 distinct in training)
```

ホールドアウトの単語の約 12 分の 1 は訓練時に 1 度も出現しておらず、ホールドアウトの単語ペアの半数は未知であり、3 語組（トライグラム）に至っては 10 個中ほぼ 9 個がカウント 0 である。最尤推定量（maximum likelihood estimate）Equation 8.8 のもとでは、未知の n グラムにはすべて確率 0 が割り当てられる。そして Equation 8.6 の中に 0 の因子が 1 つでもあれば、英語として完全に妥当な文章全体に確率 0 が割り当てられてしまう。

8.3.1.4 単語頻度とジップの法則

なぜカウントテーブルはこれほど急速にスパースになるのだろうか。根本的な原因は、自然言語における単語頻度（word frequency）の分布が極端に偏っていることにある。トークンを頻度の降順に順位付けする `Vocab` クラス（8.2 節）を用いて、『The Time Machine』における上位 10 個の最頻出語を調べてみよう。

```
words = text.split()
vocab = d2l.Vocab(words)
vocab.token_freqs[:10]
```

```
[('the', 2261),
 ('i', 1267),
 ('and', 1245),
 ('of', 1155),
 ('a', 816),
 ('to', 695),
 ('was', 552),
 ('in', 541),
```

```
 ('that', 443),
 ('my', 440)]
```

これら上位 10 個の最頻出語からは、タイムトラベルに関する手がかりは何一つ得られない。並んでいるのは冠詞、代名詞、前置詞（しばしば**ストップワード**と呼ばれる）であり、ヴィクトリア朝のどの小説でもほぼ同じリストになるだろう。目を引くのは、頻度がどれほど急激に落ち込むかである。10 番目に多い単語の頻度は、早くも最頻出語の 5 分の 1 未満にすぎない。両対数軸上に頻度と順位をプロットすると、最上位の数語を除けば直線にきわめてよく乗り、べき乗則の特徴が現れる。具体的には、単語の頻度 n_i（最頻出から i^{th} の単語）は**ジップの法則**（Zipf's law）に従う。

$$n_i \propto \frac{1}{i^s}, \quad \text{equivalently} \quad \log n_i = -s \log i + c, \tag{8.9}$$

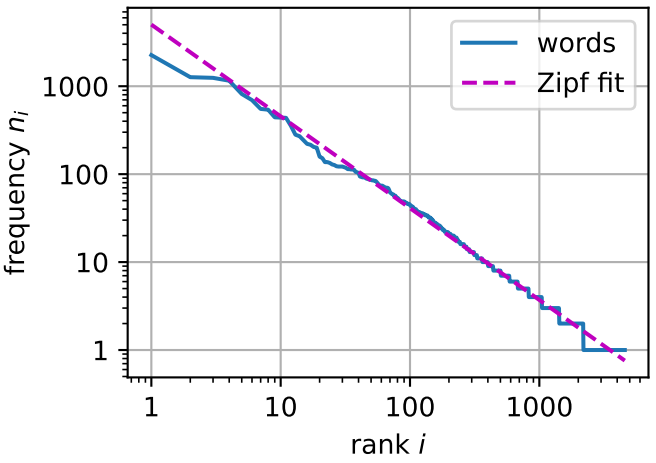
ここで s は指数、c は定数である。以下の補助関数 `zipf_fit` は、順位 10 から 1,000 までの範囲で両対数空間に直線を当てはめることで s と c を推定する。

```
def zipf_fit(freqs, lo=10, hi=1000):
    """Fit log n_i = -s log i + c over ranks lo..hi; return (s, c)."""
    ranks = np.arange(lo, hi + 1)
    slope, c = np.polyfit(np.log(ranks), np.log(freqs[lo - 1:hi]), 1)
    return -slope, c
```

`zipf_fit` を単語頻度 `freqs`（`vocab.token_freqs` から取得）に適用し、推定された指数 s を表示するとともに、経験的な出現回数にべき乗則の直線を重ねて描画する。

```
freqs = [freq for token, freq in vocab.token_freqs]
s, c = zipf_fit(freqs)
print(f'Zipf exponent s = {s:.2f}')
ranks = np.arange(1, len(freqs) + 1)
d2l.plot(ranks, [freqs, np.exp(c) * ranks ** -s], xlabel='rank $i$',
         ylabel='frequency $n_i$', xscale='log', yscale='log',
         legend=['words', 'Zipf fit'])
```

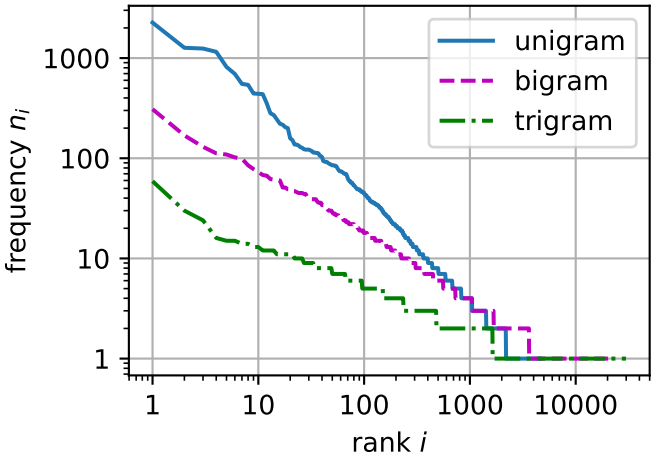
```
Zipf exponent s = 1.04
```



当てはめた指数は 1 に近い値となり、さまざまな自然言語にわたる Zipf の古典的な観測と一致する。べき乗則に従う分布は裾が重い（heavy-tailed）。個々の希少語が出現する頻度は低くとも、異なる希少語があまりに数多く存在するため、それらを合わせるとどのテキストでもかなりの割合を占めるのである。バイグラムやトライグラムでも同様の性質が見られるかを確かめるため、それぞれの頻度曲線を計算してジップ指数を当てはめ、1 回しか出現しない異なる n グラムの割合を測定し、3 本の曲線をまとめてプロットする。

```
counts = [sorted(collections.Counter(zip(*(words[i:] for i in range(n))))
                 .values(), reverse=True) for n in (1, 2, 3)]
for n, f in zip((1, 2, 3), counts):
    print(f'{n}-grams: s = {zipf_fit(f)[0]:.2f}, '
          f'{np.mean(np.array(f) == 1):.0%} of distinct {n}-grams occur once')
d2l.plot([np.arange(1, len(f) + 1) for f in counts], counts,
         xlabel='rank $i$', ylabel='frequency $n_i$', xscale='log',
         yscale='log', legend=['unigram', 'bigram', 'trigram'])
```

```
1-grams: s = 1.04, 52% of distinct 1-grams occur once
2-grams: s = 0.69, 82% of distinct 2-grams occur once
3-grams: s = 0.47, 95% of distinct 3-grams occur once
```



単語の 2 語組や 3 語組もやはりべき乗則に従うが、その指数はより小さくなる（それぞれ約 0.7 と約 0.5）。カウントベースのモデルにとって決定的なのは、n が増えるにつれて分布の裾が劇的に重くなる点である。異なる単語の半数が 1 回しか出現しないのに対し、異なるトライグラムでは 10 個中 9 個以上が、たった 1 回の観測に確率推定を依存するシングルトンとなる。ホールドアウトテキストで未知のトライグラムが大量発生する原因はまさにこの重い裾にあり、各文脈を独立に推定するモデルは、文脈が長くなるにつれて指数関数的に多くのデータを必要とする。

8.3.2 パープレキシティと平滑化

8.3.2.1 パープレキシティ

カウントテーブルを詳しく調べたことで、サンプル生成では隠れていた深刻なスパース性が明らかになった。これらのテーブルを修復する前に、変更によって本当に汎化が改善したかを判定できるよう、原理に基づいた定量的な評価指標が必要である。ホールドアウトテキストの同時確率を評価するのは自然な出発点だが、系列の生の確率は文書長に対して指数関数的に小さくなってしまう。そのため、モデルの良し悪しにかかわらず、どのモデルも Tolstoy の『戦争と平和』には Saint-Exupéry の『星の王子さま』よりはるかに小さな確率を割り当ててしまう。必要なのは、トークンあたりの平均である。

情報理論（information theory）はまさにその指標を与えてくれる。3.1.2.3 節で見たように、確率的分類器の交差エントロピー損失（cross-entropy loss）は、正解ラベルの**サプライザル**、すなわち正解ラベルに割り当てられた確率の負の対数に等しい。言語モデルはあらゆる位置における次トークン分類器であるため、系列の全 T トークンにわたる平均サプライザルによって評価する。

$$\frac{1}{T}\sum_{t=1}^{T} -\log P(x_t \mid x_{<t}), \tag{8.10}$$

ここで、P はモデルの予測分布（predictive distribution）であり、x_t はステップ t における実際のトークンである。この平均は長さで正規化されるため、長さの異なる文書同士を比較できるようになる。自然言語処理では、Equation 8.10 の指数を取った値を報告するのが慣例であり、これを**パープレキシティ**と呼ぶ (Jelinek et al. 1977)。

$$\mathrm{PPL} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P(x_t \mid x_{<t})\right). \tag{8.11}$$

言い換えれば、パープレキシティとは観測された各トークンに割り当てられた確率の逆数の幾何平均である。すべての目標トークンに確率 $1/4$ を割り当てるモデルはパープレキシティ 4 を達成し、等確率な 4 つの候補から一様に選ぶときの不確実性と一致する。したがって、パープレキシティはトークンあたりのモデルの**実効分岐数**（effective branching factor）として自然に解釈できる。以下の 3 つの極限ケースがこのスケールの基準点となる。

- 常に正解トークンを確率 1 で予測するモデルは、取り得る最小のパープレキシティ 1 を達成する。
- 観測されたトークンのうち 1 つにでも確率 0 を割り当てるモデルは、パープレキシティが無限大になる。
- 語彙全体にわたる一様分布を予測するベースラインモデルはパープレキシティ $|\mathcal{V}|$ となり、有用なモデルであれば必ず下回るべき上界を与える。

平滑化していない最尤法による n グラムモデルは、直ちに 2 番目のケースに直面する。未知の n グラムすべてに確率 0 を割り当てるため、ホールドアウトテキストに未知のトライグラムが 1 つ現れただけでパープレキシティが無限大に発散してしまうからである。カウントモデルを有意義に評価・比較するには、まずカウントを**平滑化**してゼロを取り除かなければならない。

8.3.2.2 ラプラス平滑化

古典的な解決策は**ラプラス平滑化**（Laplace smoothing、加算 α 平滑化とも呼ばれる）である。この手法では、正の定数 $\alpha > 0$ をすべてのカウントに加え、得られる分布が正規化された状態を保つよう分母に $\alpha|\mathcal{V}|$ を加える。

$$\hat{P}(x_t \mid x_{t-n+1}, \ldots, x_{t-1}) = \frac{N(x_{t-n+1}, \ldots, x_t) + \alpha}{N(x_{t-n+1}, \ldots, x_{t-1}) + \alpha|\mathcal{V}|}. \tag{8.12}$$

これにより、すべての未知の n グラムは小さな正の確率を受け取り、まったく未知の文脈は一様分布 $1/|\mathcal{V}|$ へと帰着する。この発想は言語モデルより 2 世紀も古く、Laplace は今日まで毎日太陽が昇ってきたという事実のもとで明日も太陽が昇る確率を推定するために $\alpha = 1$ としてこれを導入した。

まず、ラプラス平滑化版の `prob` メソッドを `NGramLM` に追加しよう。このメソッドは、モデルに保持された平滑化定数 `alpha` を用いて Equation 8.12 を実装する。

```
@d2l.add_to_class(NGramLM)
def prob(self, context, token):
    """Laplace-smoothed P(token | last n-1 tokens of context)."""
    ctx = tuple(context[len(context) - self.n + 1:])
    num = self.cont.get(ctx, {}).get(token, 0) + self.alpha
    return num / (self.total.get(ctx, 0) + self.alpha * self.vocab_size)
```

`prob` が用意できたので、`eval_nll` でトークン系列全体の負の対数尤度（negative log-likelihood）の総和を計算し、`words_val` 上でのホールドアウト・パープレキシティを各 n グラム次数について求められるようにする。

```
def eval_nll(model, tokens):
    """Total negative log-likelihood of tokens, in nats."""
    return -sum(math.log(model.prob(tokens[max(0, t - model.n + 1):t],
                                    tokens[t]))
                for t in range(len(tokens)))

for n, model in models.items():
    ppl = math.exp(eval_nll(model, words_val) / len(words_val))
    print(f'{n}-gram held-out perplexity, Laplace smoothing: {ppl:7.1f}')
```

```
1-gram held-out perplexity, Laplace smoothing:   840.6
2-gram held-out perplexity, Laplace smoothing:   753.6
3-gram held-out perplexity, Laplace smoothing:  2481.4
```

ラプラス平滑化のもとでは、生成サンプルが最も説得力に富んでいたはずのトライグラムモデルが、実際には大差で**最悪**のホールドアウト・パープレキシティを記録し、バイグラムもユニグラムをわずかに上回るにとどまる。カウントのスパース性がこの両方の結果を説明する。ホールドアウトのトライグラムの 87% が未知であるとき、トライグラムの予測の大部分はほぼ一様な平滑化の当て推量へと潰れてしまうのである。α を調整しても救済はできない。演習問題 2 で求める最適な α を用いてもなお、トライグラムのパープレキシティは 2,400 を超えたままである。流暢に見えるサンプルは、汎化を測る指標としては当てにならないことが判明した。

これらの数値は、ラプラス平滑化が抱える 2 つの根本的な欠陥を浮き彫りにしている。第 1 に、未知の後続語すべてに確率質量を均等に配分してしまい、各単語が一般にどれほど出現しやすいかを無視している。第 2 に、より短い文脈へフォールバックできない。未知のトライグラムは、その末尾 2 語が高頻度のバイグラムをなしていようと不自然なペアであろうと、まったく同じ平坦な推定値を受け取ってしまうのである。

8.3.2.3 Kneser–Ney 平滑化

Kneser–Ney 平滑化 (Kneser and Ney 1995)（ここでは Chen and Goodman (1999) による補間型の定式化を用いる）はこれら両方の欠陥を解消し、20 年にわたり最強のカウントベース言語モデルとして君臨した。記号として、u を $n-1$ トークンの文脈、w を次のトークン、u' を u の最も古いトークンを落として得られるより短い文脈とする。この手法は、互いに連動する 3 つのアイデアを組み合わせている。

第 1 のアイデアは**絶対ディスカウンティング**（absolute discounting）である。経験的なカウントは希少な事象の真の頻度を系統的に過大評価する。希少な n グラムの膨大な母集団から選ばれる際の選択バイアス（selection bias）により、訓練中に 1 回観測された n グラムは、同じサイズの新たなサンプルでは平均して 1 回未満しか出現しないからである。すべての非ゼロカウントから一定のディスカウント $0 < D < 1$ を差し引くことでこの上方向の統計的バイアスを補正でき、歪みが最も大きい小さなカウントほど強く補正されるとともに、合計 $D\,S(u)/N(u)$ の確率質量が文脈 u において解放される。ここで $S(u)$ は、訓練中に u の直後に観測された異なるトークン種類の数である。

第 2 のアイデアは**補間**（interpolation）である。差し引いて得られた確率質量を一様に広げるのではなく、1 つ下の次数の推定値 $P_{\text{KN}}(w \mid u')$ に配分する。この推定値もまったく同じ形で再帰的に定義され、最下層の一様な下限 $1/|\mathcal{V}|$ まで続く。

$$P_{\text{KN}}(w \mid u) = \frac{\max(N(u,w) - D, 0)}{N(u)} + \frac{D\,S(u)}{N(u)}\,P_{\text{KN}}(w \mid u'). \tag{8.13}$$

この再帰のもとでは、頻繁に観測される文脈は主として自身の経験カウントに依拠する一方、多種多様な後続語とともに観測された文脈はより短い文脈へと多くの重みを移す。これは、過去に多様な続き方をした文脈ほど未知の続き方も許容しやすいという直観を反映している。もし文脈 u が訓練時に 1 度も観測されていなければ、確率質量はすべてそのまま下の次数へと引き継がれ、$P_{\text{KN}}(w \mid u) = P_{\text{KN}}(w \mid u')$ となる。

第 3 のアイデアは、低次のモデルで**何を**数えるかを変えることである。低次の分布は主として長い文脈では出現しなかった単語にスコアを付けるために呼び出されるため、コーパスでの生の出現頻度ではなく、その単語がどれほど容易に**新しい**文脈に入り込めるかを測るべきである。訓練テキストにおいて、`"traveller"` は 49 回出現するが直前の単語は 2 種類しかなく（`"time"` が 48 回、`"real"` が 1 回）、一方で `"said"` は 76 回出現し、48 種類の異なる先行語の後に続いている。生のユニグラム頻度を用いると、未知の文脈でも `"traveller"` を `"said"` とほぼ同じくらい積極的に予測してしまうが、異なる先行語の数を数えれば、`"traveller"` が `"time"` なしにはめったに現れないことを見抜ける（自然言語処理における古典的な例は `"Francisco"` であり、ニュース記事で頻出するにもかかわらず、その直前はほぼ例外なく `"San"` である）。そのため Kneser–Ney 平滑化では、最高次より下のすべての次数において $N(u, w)$ を**継続カウント**（continuation count）、すなわち訓練中に (u, w) の直前に現れた異なるトークンの数に置き換え、$N(u)$ と $S(u)$ も同様にその継続カウントから計算する。

さらに都合のよいことに、ディスカウントパラメータ D には Ney et al. (1994) が導出した簡潔な閉形式の推定値があり、各次数で $D = c_1/(c_1 + 2c_2)$ と与えられる。ここで c_1 と c_2 は、その次数においてちょうど 1 回および 2 回出現した異なる n グラムの数である（最高次より下では継続カウントを用いる）。

以下の関数 `kneser_ney` は Equation 8.13 を完全に実装したものである。最高次の n グラムを集計し、1 つ上の次数の異なるタプルからすべての低次の継続カウントを導出し、$N(u)$、$S(u)$、D を計算したうえで、トークン系列全体にわたって再帰式を評価し、利用可能なプレフィックスのもとでの各トークンの対数確率（log probability、単位は nat）を返す。

```
def kneser_ney(train, tokens, vocab_size, n=3):
    """Interpolated Kneser-Ney log-probabilities (nats) of each of tokens."""
    counts = {n: collections.Counter(zip(*(train[i:] for i in range(n))))}
    for k in range(n - 1, 0, -1):  # Lower orders: continuation counts
        counts[k] = collections.Counter(g[1:] for g in counts[k + 1])
    stats = {}
    for k, c in counts.items():
        c1, c2 = (sum(x == r for x in c.values()) for r in (1, 2))
        N, S = collections.Counter(), collections.Counter()
        for g, x in c.items():
            N[g[:-1]] += x
            S[g[:-1]] += 1
        stats[k] = c, N, S, c1 / max(c1 + 2 * c2, 1)
    logp = []
    for i in range(len(tokens)):
        p = 1 / vocab_size  # Below the unigram: uniform
        for k in range(1, min(n, i + 1) + 1):
            c, N, S, D = stats[k]
            g = tuple(tokens[i - k + 1:i + 1])
            if N[g[:-1]]:  # An unseen context passes p down unchanged
                p = (max(c[g] - D, 0) + D * S[g[:-1]] * p) / N[g[:-1]]
        logp.append(math.log(p))
    return logp
```

検証分割全体にスコアを付ける前に、おなじみの文脈 "the time" で Kneser–Ney 平滑化の動きを見てみよう。以下では、トライグラムのディスカウント D を表示し、3 つの後続語候補に対して Kneser–Ney 平滑化とラプラス平滑化が割り当てる確率を比較する。候補は、"traveller"（訓練中に "the time" が 87 回出現したうち 47 回続いた語）と、訓練中に 1 度も後に続かなかった 2 語 "was" および "filby" である。

```
c3 = collections.Counter(zip(words_train, words_train[1:], words_train[2:]))
c1, c2 = (sum(x == r for x in c3.values()) for r in (1, 2))
print(f'trigram discount D = {c1 / (c1 + 2 * c2):.2f}')
for w in ('traveller', 'was', 'filby'):
    lp = kneser_ney(words_train, ['the', 'time', w], vocab_size)[-1]
    print(f'P({w} | the time): Kneser-Ney {math.exp(lp):.2g}, '
          f'Laplace {models[3].prob(["the", "time"], w):.2g}')
```

```
trigram discount D = 0.93
P(traveller | the time): Kneser-Ney 0.53, Laplace 0.36
P(was | the time): Kneser-Ney 0.00037, Laplace 7.7e-05
P(filby | the time): Kneser-Ney 2.3e-05, Laplace 7.7e-05
```

トライグラムのディスカウントは約 0.9 であり、観測されたすべてのトライグラムからほぼ丸々 1 回分を差し引いている。これは 1 回きりの観測がいかにノイズを多く含むかを鮮やかに物語っている。カウントが 47 ある頻出トライグラムでは、0.9 を引いてもほとんど影響はなく、"traveller" は経験頻度 $47/87$ の近くに保たれるのに対し、ラプラス平滑化はその確率質量の 3 分の 1 近くを語彙全体へと浪費してしまう。未知の後続語における対比はさらに鮮明である。ラプラス平滑化が "was" と "filby" をまったく同じに扱うのに対し、Kneser–Ney 平滑化は "was" が幅広い文脈の後に続く一方で "filby" はごく限られた文脈にしか現れないことを見抜き、両者の確率に 1 桁以上の差をつける。これによってホールドアウトテキストでどれほど改善するかを測定してみよう。

```
for n in (1, 2, 3):
    kn = kneser_ney(words_train, words_val, vocab_size, n)
    print(f'{n}-gram held-out perplexity, Kneser-Ney: '
          f'{math.exp(-sum(kn) / len(kn)):7.1f}')
```

```
1-gram held-out perplexity, Kneser-Ney:   592.8
2-gram held-out perplexity, Kneser-Ney:   312.3
3-gram held-out perplexity, Kneser-Ney:   300.1
```

Kneser–Ney 平滑化によってランキングは完全に逆転する。ユニグラムでさえラプラス平滑化を上回り、バイグラムはパープレキシティをほぼ半減させ、トライグラムは他のすべてのカウントモデルを凌駕する。未知のトライグラムが一様な当て推量に潰れるのではなく、適切に較正（calibration）されたバイグラムおよびユニグラムの継続推定値へと滑らかにバックオフするからである。もっとも、トライグラムによるバイグラムからの改善幅は控えめである。ホールドアウトのトライグラムの大半は依然として未知であるため、トライグラムモデルは過半の位置でバイグラムへとフォールバックしているからである。**未知の**テキストで低いパープレキシティを達成することこそ言語モデルの決定的な試金石であり、続くニューラルモデルでもパープレキシティを主要なベンチマークとして用いる。

それでもなお、Kneser–Ney 平滑化はあらゆるカウントベースの手法と共通する根本的な限界を抱えている。平滑化はゼロから確率質量を再配分するだけであり、未知の組み合わせのうちどれが意味的にもっともらしいかを推論することはできない。カウントテーブルは各トークンを分割不能で互いに関係のない記号として扱うため、"cat" と "feline" が統語的・意味的な役割を共有

していることも、"I want to eat, grandma" が無害であるのに対し "I want to eat grandma" が物騒であることも知る由がない。関連する単語や文脈の間で統計的な強みを共有するには、学習された連続表現が不可欠であり、これこそが 8.4 節でニューラルネットワークへと向かう直接の動機となる。

8.3.2.4 バイトあたりビット数

パープレキシティには 1 つ重大な制約がある。それはトークン化の選択に依存するという点である。文字レベルのモデルは各ステップで 27 個の記号から選ぶのに対し、単語レベルのモデルは数千語から選ぶため、生のパープレキシティはまったく異なるスケール上にあり、直接比較することができない。しかしトークン化はモデルの内部設計上の選択にすぎず（8.2 節）、公正な評価指標はそれに対して不変でなければならない。幸い、解決策は単純であり、分母を置き換えればよい。総サプライザルをトークン数で割る代わりに、評価対象テキストの**バイト**数（テキスト自体のみによって決まる量）で割るのである。サプライザルをビット単位（底 2 の対数）で測れば、**バイトあたりビット数**（bits per byte、bpb）が得られる。

$$\text{bpb} = \frac{1}{B}\sum_{t=1}^{T} -\log_2 P(x_t \mid x_{<t}), \tag{8.14}$$

ここで、B は評価対象テキストの UTF-8 バイト長である。バイトあたりビット数は、可逆圧縮（lossless compression）の圧縮率として直ちに実用的な意味を持つ。算術符号器と組み合わせれば、bpb バイトあたりビット数を達成するモデルはテキストを元のバイトサイズの $\text{bpb}/8$ に圧縮できる。

パープレキシティとバイトあたりビット数を直接比較するため、正規化した同じ訓練テキスト上で、まずラプラス平滑化、次に Kneser–Ney 平滑化を用いて、文字、単語、および 1,024 トークンの BPE 語彙（8.2 節のトークナイザを使って正規化済み訓練分割で訓練したもの）という 3 通りのトークン化のもとでトライグラムモデルを当てはめる。ただし、比較を公正に保つには 1 つ注意すべき細部がある。単語モデルが <unk> を予測した時点では、語彙外（out-of-vocabulary）の**どの**単語が出現したのかをまだ特定していないのである。ホールドアウトテキストを可逆的に符号化するには、単語モデルは未知語を 1 文字ずつ綴り出す必要があるため、1 文字あたり（末尾の空白も含めて）$\log_2 27$ ビットを加算する。これは正規化された 27 種類の記号から各文字を一様に符号化するコストである。

```
bpe = d2l.BPETokenizer(vocab_size=1024, pattern=d2l.BPETokenizer.GPT2_PATTERN)
bpe.train(text[:split])
streams = {'character': (list(text[:split]), list(text[split:]),
                         len(set(text))),
           'BPE (1,024)': (bpe.encode(text[:split]), bpe.encode(text[split:]),
                           bpe.vocab_size),
           'word': (words_train, words_val, vocab_size)}
n_bytes = len(text[split:].encode('utf-8'))
spell = sum((len(w) + 1) * math.log(27)  # Nats to spell out the <unk> words
            for w in text[split:].split() if w not in known)
print(f'{"tokenization":>12} {"|V|":>5} {"tokens":>7} {"ppl":>7} '
      f'{"bpb":>5} {"KN bpb":>6}')
for name, (train, val, m) in streams.items():
    nll = eval_nll(NGramLM(train, 3, m), val)
    nll_kn = -sum(kneser_ney(train, val, m))
    extra = spell if name == 'word' else 0
```

```
        print(f'{name:>12} {m:>5} {len(val):>7} {math.exp(nll / len(val)):>7.1f} '
              f'{(nll + extra) / math.log(2) / n_bytes:>5.2f} '
              f'{(nll_kn + extra) / math.log(2) / n_bytes:>6.2f}')
```

```
tokenization   |V|  tokens     ppl   bpb KN bpb
   character    27   17345     6.4  2.68   2.67
 BPE (1,024)  1024    5509   281.6  2.58   1.90
        word  4316    3373  2481.4  2.81   2.22
```

生のパープレキシティで測ると、ラプラス平滑化を用いた 3 つのモデルには 2 桁以上の開きがあるように見える。しかしバイトあたりビット数で測ると、3 つは互いに 3 分の 1 ビット以内の範囲に収まり、パープレキシティでは圧倒的に優位に見えた文字モデルはもはや首位ではなくなる。固定されたマルコフ次数 n に対して、粒度の粗いトークンほどテキスト上のより長い範囲をカバーする。2 語の文脈は約 10 文字先まで遡れるのに対し、2 文字では 1 音節すらほとんどカバーできないからである。もし語彙外単語のペナルティがなければ単語モデルが首位に立つところだが、その綴り出しコストが約 0.6 バイトあたりビット数だけ上乗せされるため、ラプラス平滑化では最下位に転落する。バイトレベルの BPE には語彙外（OOV）のペナルティが一切なく、最も高い圧縮性能を達成しており、現代の言語モデルがサブワードトークン化（subword tokenization）を採用するもう 1 つの理由を裏付けている。

最後の列を見ると、Kneser–Ney 平滑化は BPE と単語のトライグラムの両方でバイトあたりビット数を 0.5 ビット以上削り落としている一方、文字トライグラムはほとんど変化していない。記号が 27 種類しかなければ未知の文字トライグラムはごくわずかであり、平滑化が働く余地がほとんどないためである。約 1.9 バイトあたりビット数を達成する Kneser–Ney 平滑化の BPE トライグラムが、カウントベースの最強のベースラインとなる。

したがって、異なる語彙上に構築された現代の言語モデル同士（たとえば 50,000 トークンのモデルと 200,000 トークンのモデル）を比較する際には、バイトあたりビット数が標準的な評価指標となる。スケール感を掴むには 3 つの基準が参考になる。Shannon による人間の予測実験という古典的研究では、印刷された英語のエントロピーが 1 文字あたり約 1 ビットと推定された (Shannon 1951)。現代のニューラル言語モデルは多様なウェブテキストを 1 バイトあたり 1 ビットを優に下回る水準まで圧縮する。そして本節のカウントベースのモデルは、およそ 1.9〜2.8 バイトあたりビット数の間に位置している。

8.3.3 系列の分割

次節以降のニューラル言語モデルは、8.2 節のトークナイザが生成した固定長の BPE トークン部分系列のミニバッチを入力として受け取る。ここでそのデータパイプラインを組み立てよう。

トークン化されたコーパスを、トークン ID の単一の連続ストリーム $x_1, x_2, x_3, \ldots$ として思い描いてみよう。長さ T でインデックス t から始まる入力ウィンドウは $x_t, \ldots, x_{t+T-1}$ を含み、それと対になる目標は同じウィンドウを右に 1 トークンずらした $x_{t+1}, \ldots, x_{t+T}$ である。モデルが入力ウィンドウを順にたどる際、すべての位置で次のトークンを予測するため、各ウィンドウは教師ありの次トークン予測（next-token prediction）目標を T 個供給する。コーパス末尾の T トークンを除くすべての開始オフセットが有効なウィンドウを定め、隣接するウィンドウ同士は 1 トークンを除いてすべて重なり合う（Figure 8.6）。訓練時には、確率的勾配降下法（stochastic gradient descent）が必要とするシャッフルを実現するため、これらのウィンドウからランダムにミニバッチをサンプリングする。

以下の `TimeMachine` コンストラクタがこのパイプライン全体をまとめる。中編小説をダウンロードし、8.2 節の BPE トークナイザを訓練してテキストを整数 ID に符号化し、長さ `num_steps` の重複する全ウィンドウを対応する入力（`X`）および目標（`Y`）の配列へと積み重ねる。訓練済みトー

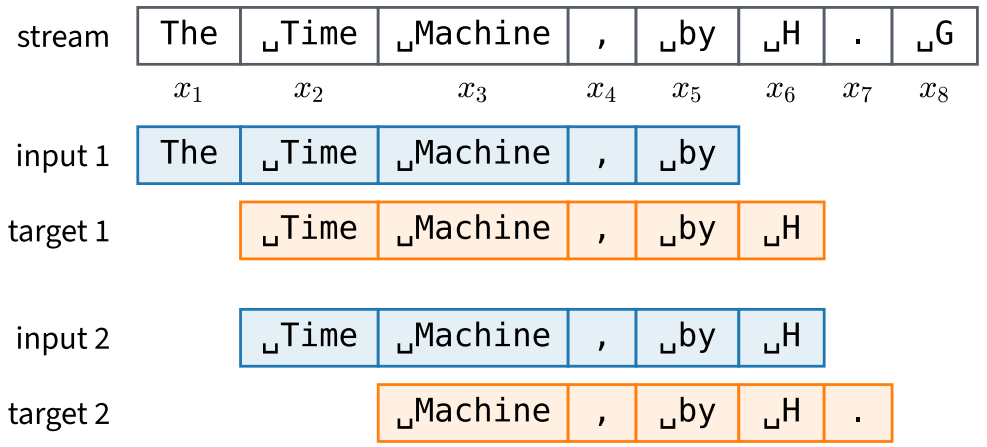


Figure 8.6: 『The Time Machine』冒頭の BPE トークンに対する長さ $T = 5$ の最初の 2 つの入力ウィンドウ（␣は先頭の空白を表す）。各目標は入力を 1 トークン先にずらしたものであり、連続するウィンドウは 1 トークンを除いて重なり合う。ミニバッチはランダムに抽出したウィンドウを集めて構成される。

クナイザは `data.tokenizer` と `data.vocab` の両方に格納されるため、ID をテキストへデコードできるほか、どのトークン化モードでも統一的に `len(data.vocab)` でクエリできる。

```
@d2l.add_to_class(d2l.TimeMachine)
def __init__(self, batch_size, num_steps, num_train=50000, num_val=5000,
             tokenization='bpe', vocab_size=1024):
    super(d2l.TimeMachine, self).__init__()
    self.save_hyperparameters()
    raw_text = self._download()
    if tokenization == 'bpe':
        self.tokenizer = d2l.BPETokenizer(
            vocab_size, pattern=d2l.BPETokenizer.GPT2_PATTERN)
        self.tokenizer.train(raw_text)
        corpus, self.vocab = self.tokenizer.encode(raw_text), self.tokenizer
    else:  # tokenization='char': characters, via TimeMachine.build
        corpus, self.vocab = self.build(raw_text)
    windows = [corpus[i:i+num_steps+1] for i in range(len(corpus)-num_steps)]
    # Drop the windows that straddle the split, so that no validation token
    # is a training target
    windows = windows[:num_train] + windows[num_train+num_steps:]
    array = d2l.tensor(windows)
    self.X, self.Y = array[:,:-1], array[:,1:]
```

先頭の `num_train` 個のウィンドウを訓練用に確保し、続く `num_val` 個のウィンドウを検証用に充てる。隣接するウィンドウは大きく重なり合っているため、訓練分割の直後に続くウィンドウをそのまま使うと、すでに訓練の目標として現れたトークンが含まれてしまう。そこで境界にある `num_steps` 個のウィンドウをバッファとして捨てることで、訓練と検証の間のデータリークを防ぐ。検証時には、ミニバッチはシャッフルせず先頭から順番に反復処理する。

```
@d2l.add_to_class(d2l.TimeMachine)
def get_dataloader(self, train):
    idx = slice(0, self.num_train) if train else slice(
        self.num_train, self.num_train + self.num_val)
    return self.get_tensorloader([self.X, self.Y], train, idx)
```

各ミニバッチは形状（batch_size、num_steps）の 2 つの整数テンソルからなる。サンプルの 1 行をデコードすると、X に Wells の文章が収められており、Y がちょうど 1 トークンだけ先にずれていることが確認できる。

```
data = d2l.TimeMachine(batch_size=2, num_steps=10)
for X, Y in data.train_dataloader():
    print('X:', X, '\nY:', Y)
    break
print('X[0]:', repr(data.tokenizer.decode([int(i) for i in X[0]])))
print('Y[0]:', repr(data.tokenizer.decode([int(i) for i in Y[0]])))
```

```
X: [[ 373  282   44  277  438  287   10  111  325  463]
 [ 732  313  316  298  437 1019  996  867  118  697]]
Y: [[ 282   44  277  438  287   10  111  325  463  273]
 [ 313  316  298  437 1019  996  867  118  697  270]]
X[0]: 'loor, and fell\nover one'
Y[0]: 'or, and fell\nover one of'
```

BPE トークンのウィンドウは、いかなるカウントテーブルが活用できるよりもはるかに長い文脈を提供する（先ほどの単語テーブルは、わずか 2 語の履歴で早くもデータ不足に陥っていた）。次節では、ルックアップテーブルをニューラルネットワークに置き換え、任意の長さのプレフィックスを学習された隠れ状態（hidden state）へと圧縮する。

8.3.4 まとめ

言語モデルはトークン系列に確率を割り当て、確率の連鎖律によってこの課題は次トークン予測へと帰着するため、単一のモデルでテキストのスコア付けと生成の両方が可能になる。マルコフ仮定のもとで、n グラムモデルはコーパス内のカウントから次トークンの確率を直接推定する。構築は容易で局所的には流暢なサンプルを生成できるものの、指数関数的に膨張する文脈空間とジップ則に従う重い裾に阻まれ、観測された n グラムの大半がシングルトンとなって汎化ではなく丸暗記を引き起こす。トークンあたりの平均交差エントロピーの指数であるパープレキシティはモデルの実効分岐数を測り、ラプラス平滑化を用いたトライグラムがホールドアウトテキストで振るわない理由を明らかにする。補間型 Kneser–Ney 平滑化は観測カウントをディスカウントし、解放された質量をより低次の継続カウントへと再配分することで最強のカウントベースモデルをもたらすが、意味的に関連するトークン間で汎化することは依然としてできない。また、パープレキシティは語彙の粒度に依存するため、バイトあたりビット数がトークン化に不変な圧縮率としてモデル間の公平な比較を可能にする。最後に、次節以降のニューラル言語モデルへ入力するため、BPE トークン化されたコーパスを固定長の重複する入力・目標ウィンドウへと変換し、検証分割へのリークを防ぐ明確なギャップを設けた。

8.3.5 演習問題

1. **密な格納とスパースな格納**。語彙サイズ $|\mathcal{V}| = 50{,}000$ の単語トークン上の 5 グラムモデルを考える。

1. 条件付き確率 Equation 8.8 の密なテーブルは何個のエントリを含むか。
2. 代わりに、1 兆トークンのコーパスで観測された 5 グラムのみを 1 エントリあたり 16 バイトで格納するとする。必要なメモリは最大でどれほどか。また、取り得るすべての 5 グラムのうちどれだけの割合がカバーされるか。
3. 8.3.1 節の単語語彙は $|\mathcal{V}| = 4{,}316$ エントリであり、そこのコードは訓練テキスト内の異なる単語トライグラムの数を表示している。取り得るすべてのトライグラムのうち、訓練に出現する割合はどれだけか。また、ホールドアウトのトライグラムのうち訓練に出現する割合はどれだけか。したがって、(b) で計算したカバー率は、新しいテキストに対するクエリがテーブル内で該当する 5 グラムを見つけられる頻度の良い推定値といえるか。

2. [code] **平滑化定数**。単語のユニグラム、バイグラム、トライグラムの各モデルについて、Equation 8.12 の α を $\alpha = 10^{k/4}$（$k = -16, -15, \ldots, 2$）の 19 個の値にわたって変化させ、ホールドアウト・パープレキシティを対数軸の α に対してプロットせよ。

   1. 最良の α は n に応じてどのように変化するか。また、それはなぜか。
   2. 3 つのモデルをそれぞれの最良の α におけるホールドアウト・パープレキシティによって順位付けせよ。このランキングは、8.3.2 節で `NGramLM` のデフォルト値 $\alpha = 0.01$ を用いて表示したランキングとどのように異なるか。

3. **偏りのある分布でのパープレキシティ**。10 個の記号からなる訓練コーパスにおいて、1 つの記号が 91% の割合で出現し、残りの 9 個の記号がそれぞれ 1% の割合で出現するとする。

   1. このコーパスから推定されたユニグラムモデルについて、同じ分布から抽出された長いテストテキスト上でのパープレキシティを計算せよ。語彙サイズの 10 と比べてどうなるか。
   2. 8.3.2 節の単語レベルのユニグラムモデルは数千種類の単語から選択する。そのホールドアウト・パープレキシティを等確率な選択肢の実効数として読み取り、ジップの法則 Equation 8.9 を用いて、この値が $|\mathcal{V}|$ よりはるかに小さくなる理由を説明せよ。

   Daniel Jurafsky、James H. Martin『Speech and Language Processing』第 3 版ドラフト、演習問題 3.12 を改変。

4. **重複するウィンドウ**。`d2l.TimeMachine` は、コーパス内の取り得るすべての開始位置に対して連続する `num_steps + 1` トークンからなる行を 1 行ずつ構築する。各行の最初の `num_steps` トークンが `X` の 1 行をなし、最後の `num_steps` トークンが対応する `Y` の 1 行をなす。最初の `num_train` 行が訓練に回り、次の `num_steps` 行は捨てられ、その後の `num_val` 行が検証に回る（8.3.3 節）。ここで `num_steps=32`、`num_train=10000`、`num_val=5000` とする。

   1. 入力トークンと目標トークンを合わせて数えたとき、訓練行はコーパスのどの位置に触れ、検証行はどの位置に触れるか。もし行をまったく捨てなかった場合、両方が触れる位置は何個になり、なぜそれによって検証スコアが不当に押し上げられるのか。
   2. 訓練範囲の内部にあるトークンは、1 エポックあたり何回、目標として使われるか。
   3. 約 66,000 トークンのコーパス全体を重複しない 32 トークンのウィンドウに分割すると、何個のウィンドウが得られるか。この分割を 10,000 個の重複する訓練行と、ウィンドウ数およびカバーされる異なるコーパス位置の数の両面で比較せよ。

5. [code] **異なる長さにわたる正規化**。●`NGramLM` には系列終了トークンがない。3 記号の語彙上の短い文字列で、バイグラムの `NGramLM` を $\alpha = 1$、`vocab_size=3` として訓練せよ。`prob` と連鎖律 Equation 8.6 を用いて、長さ 2 のすべての系列の確率の和と、長さ 3 のすべての系列の確率の和をそれぞれ別個に求めよ。

   1. それぞれの和は何に等しいか。また、この 2 つの和は、長さが事前に固定されていない系列上の分布としてのこのモデルについて何を示しているか。

2. 語彙に系列終了トークンを追加して `vocab_size=4` とし、訓練文字列の末尾にもそのトークンを追加せよ。このトークンを出力した時点で系列が完結するものとして扱う。完結するすべての系列の確率の総和が厳密に 1 になることを示せ。決して終わらない系列に確率質量が残る可能性を排除しているのは、モデルのどの性質か。

Daniel Jurafsky、James H. Martin『Speech and Language Processing』第 3 版ドラフト、演習問題 3.5 を改変。

6. [code] **既知のトライグラムと未知のトライグラム。**●ホールドアウトの 3 語目以降の位置を、その位置で終わる単語トライグラムが訓練中に出現したかどうかによって 2 つのグループに分けよ。バイグラムとトライグラムの単語モデルを、それぞれ問題 2 の探索で得られた最良の α に設定し、各グループ上での各モデルの平均の負の対数尤度を計算せよ。トライグラムの方が優れたモデルとなるのはどちらのグループか。また、Kneser–Ney 平滑化 Equation 8.13 はこれをどのように活用しているか。

8.4 リカレントニューラルネットワーク

8.3 節のカウントに基づく言語モデルは、直前の短いウィンドウ内のトークンから各トークンを予測する。トライグラムモデルが条件づけるのは直前の 2 トークンのみであり、文書内のそれより前の部分は予測に一切影響しない。単にウィンドウを広げようとしてもすぐに壁にぶつかる。20 個の BPE トークンにまたがるカウント表は $1024^{20} \approx 10^{60}$ 個もの文脈を追跡しなければならず、連結した m 個のトークン埋め込みからなる固定ウィンドウを読み取るニューラルネットワーク (Bengio et al. 2003) でさえ、$m+1$ ステップ以上前のトークンはまったく参照できないうえ、パラメータ数が m に対して線形に増加してしまう。

8.1 節では、根本的に異なるアプローチとして、履歴全体を固定サイズの**隠れ状態**（hidden state）ベクトル $\mathbf{h}_{t-1}$ に圧縮し、その要約から次のトークンを予測する**潜在**自己回帰モデル（latent autoregressive model）を導入した。

$$P(x_t \mid x_{<t}) \approx P(x_t \mid \mathbf{h}_{t-1}), \qquad \mathbf{h}_t = g(\mathbf{h}_{t-1}, x_t). \tag{8.15}$$

この漸化式のもとでは、プレフィックスがどれほど長くなっても、1 ステップあたりの計算コストは一定に保たれる。**リカレントニューラルネットワーク**（RNN）は、すべてのタイムステップで重みを共有するニューラルネットワークの層によって遷移関数 g をパラメータ化する (Elman 1990)。（なお、タイムステップをまたいで引き継がれる**隠れ状態**は、単一ステップ内で入力と出力の間にあるフィードフォワードの**隠れ層**（hidden layer）とは異なる。）本節では、RNN 言語モデルを定式化し、そのパラメータ数と計算量を数え上げたうえで、『The Time Machine』を対象にゼロからの実装とフレームワークの高水準層を用いた実装の両方を作成する。

```
import random
```

8.4.1 隠れ状態を持つリカレントニューラルネットワーク

RNN は、単一隠れ層の MLP（4.1 節）から出発し、そこにリカレント結合を 1 つ追加する形で考えると最も理解しやすい。b 個の例からなり各例が d 個の特徴量を持つミニバッチ $\mathbf{X} \in \mathbb{R}^{b \times d}$ が与えられたとき（本章を通じてバッチサイズには b を用いるが、これは n が n-gram の次数を表すためである）、活性化関数 ϕ を持つ通常の隠れ層は次式を計算する。

$$\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{\text{xh}} + \mathbf{b}_{\text{h}}), \tag{8.16}$$

ここで、$\mathbf{W}_{\text{xh}} \in \mathbb{R}^{d \times h}$ および $\mathbf{b}_{\text{h}} \in \mathbb{R}^{1 \times h}$ であり、これに続いて出力射影（output projection）$\mathbf{O} = \mathbf{H}\mathbf{W}_{\text{hq}} + \mathbf{b}_{\text{q}}$ により q 個のロジットへと写す。

ここで、ミニバッチ入力の系列 $\mathbf{X}_t \in \mathbb{R}^{b \times d}$（タイムステップ t で添字づけられる）を受け取るとしよう。隠れ表現に前のステップを記憶させるため、直前の隠れ状態 $\mathbf{H}_{t-1} \in \mathbb{R}^{b \times h}$ をリカレント重み行列 $\mathbf{W}_{\text{hh}} \in \mathbb{R}^{h \times h}$ を介して層へフィードバックする。

$$\begin{aligned} \mathbf{H}_t &= \phi(\mathbf{X}_t \mathbf{W}_{\text{xh}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{hh}} + \mathbf{b}_{\text{h}}), \\ \mathbf{O}_t &= \mathbf{H}_t \mathbf{W}_{\text{hq}} + \mathbf{b}_{\text{q}}. \end{aligned} \tag{8.17}$$

同じ 5 つのパラメータテンソル（リカレント層の $\mathbf{W}_{\text{xh}}$、$\mathbf{W}_{\text{hh}}$、$\mathbf{b}_{\text{h}}$ と出力層の $\mathbf{W}_{\text{hq}}$、$\mathbf{b}_{\text{q}}$）が（$\mathbf{H}_0 = \mathbf{0}$ から始めて）すべてのステップで再利用されるため、パラメータ数は系列長に一切依存しない（Figure 8.7）。時間方向に展開した形で見ると、T ステップにわたって動作する RNN は、すべての層が同じ重みを共有する T 層のフィードフォワードネットワークである。

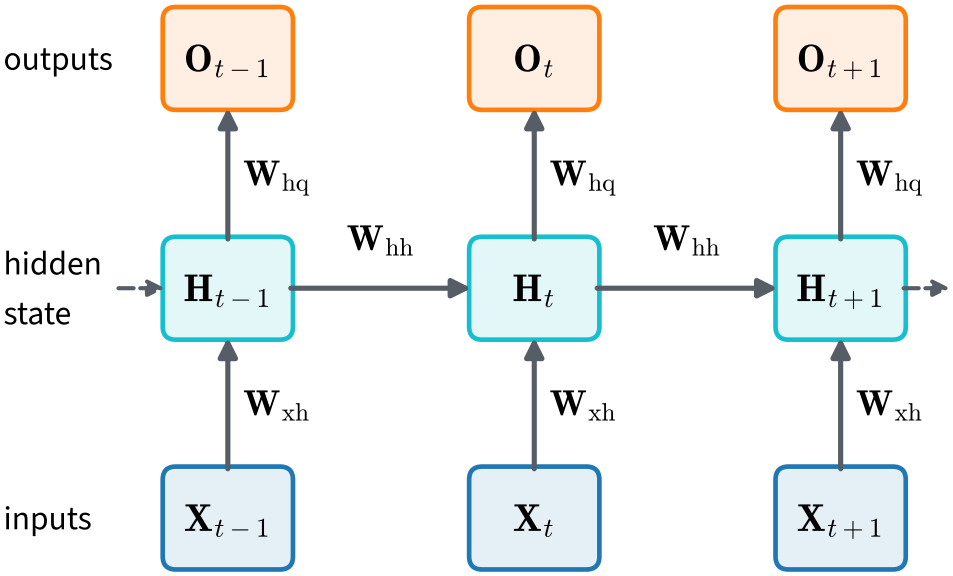


Figure 8.7: 隣接する 3 つのタイムステップにわたって展開した、隠れ状態を持つ RNN。すべてのステップで同じ重みが再利用される。

Equation 8.17 における 2 つの行列積は、水平方向に連結した入力と垂直方向に積み重ねた重みとの単一の積に等しい。

$$[\mathbf{X}_t, \mathbf{H}_{t-1}] \begin{bmatrix} \mathbf{W}_{\text{xh}} \\ \mathbf{W}_{\text{hh}} \end{bmatrix} = \mathbf{X}_t \mathbf{W}_{\text{xh}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{hh}}.$$

それにもかかわらずライブラリがこの 2 つの積を分けているのは、訓練時にはすべての入力 $\mathbf{X}_1, \ldots, \mathbf{X}_T$ が事前にわかっているため、$\mathbf{X}_t \mathbf{W}_{\text{xh}}$ を全 T ステップ分まとめて逐次的な時間ループの外でバッチ計算できるからである（演習問題 1）。

さらに、再帰状態（recurrent state）を用いれば、ビット列に含まれる 1 の累積個数が偶数か奇数か（パリティ）を毎ステップ報告する課題のように、**いかなる**固定ウィンドウモデルにも不可能なタスクを解くことができる。任意の固定ウィンドウ長 m より前のビットを 1 つ反転させると、直近 m ビットは変わらないまま真のパリティだけが反転してしまうのに対し、2 ユニットの再帰状態であれば任意の長さでパリティを追跡できる。

8.4.1.1 ステップあたり一定のメモリ量

$\mathbf{H}_t$ の計算に必要なのは $\mathbf{H}_{t-1}$ と $\mathbf{X}_t$ だけであるため、1 ステップあたりのメモリ量と計算量は系列長にかかわらず一定に保たれる。$h = 128$ 個の `float32` 隠れユニットを用いる場合、系列の状態が占めるのは、モデルが 10 トークン読んだ後でも 100 万トークン読んだ後でもわずか 512 バイトである。しかし、この一定のメモリ使用量は本質的に**非可逆な**（lossy）ボトルネックでもある。プレフィックスから記憶されるあらゆる詳細がこの 512 バイトを通過しなければならず、系列長とともに増大するコストと引き換えに過去の全トークンの表現を保持する**アテンション**（第 10 章）とは対照的である。

8.4.1.2 RNN 言語モデルと教師強制

RNN 言語モデルは 1 ステップにつき 1 つのトークン ID を受け取り、全 $q = |\mathcal{V}|$ 個の語彙項目にわたる次トークンの分布を予測する（Figure 8.8）。ステップ t では、ミニバッチ内の b 個のトークン ID が学習済みの埋め込みテーブル（embedding table）$\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ から対応する行を参照し、$\mathbf{X}_t \in \mathbb{R}^{b \times d}$ を形成する（これは $b \times |\mathcal{V}|$ のワンホット行列に $\mathbf{E}$ を掛けることと数学的に同一だが、ゼロを実体化したり乗算したりせずに済む）。リカレント層が $\mathbf{H}_t$ を更新し、$\mathrm{softmax}(\mathbf{O}_t)$ が $P(x_{t+1} \mid x_{\le t})$ を定める。これは連鎖律による分解 $P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$ を構成する条件付き確率である（8.3 節）。

パラメータと FLOPs はどこに集中しているのだろうか。BPE 語彙サイズ $|\mathcal{V}| = 1{,}027$ 個の ID（1,024 個の BPE トークンと 3 つの特殊トークン）、埋め込み幅 $d = 64$、隠れ状態のサイズ $h = 128$ の場合、内訳は次のようになる。

構成要素	形状	パラメータ数
埋め込みテーブル $\mathbf{E}$	$1{,}027 \times 64$	65,728
入力重み $\mathbf{W}_{\text{xh}}$	64×128	8,192
リカレント重み $\mathbf{W}_{\text{hh}}$	128×128	16,384
隠れバイアス $\mathbf{b}_{\text{h}}$	128	128
出力重み $\mathbf{W}_{\text{hq}}$	$128 \times 1{,}027$	131,456
出力バイアス $\mathbf{b}_{\text{q}}$	$1{,}027$	1,027
合計		222,915

リカレント層が占めるのはわずか 24,704 パラメータ（モデル全体の 11%）および 1 トークンあたり 24,576 回の積和演算であるのに対し、語彙サイズの埋め込み層と出力層がパラメータの 89% を保持し、出力射影だけで $128 \cdot 1{,}027 = 131{,}456$ 回の積和演算を要する。

訓練時には、前のステップでモデルが何を予測したかにかかわらず、毎ステップの入力にはコーパスからの**正解**（ground-truth）トークンが与えられる。この方式を**教師強制**（teacher forcing）と呼ぶ。ウィンドウ内のすべての正解入力が事前にわかっているため、ウィンドウ全体の交差エントロピー損失（cross-entropy loss、パープレキシティの対数）を 1 回の順伝播で評価できる。しかし生成時には、サンプリングまたは貪欲選択された各トークンが次の入力としてフィードバックされるため、8.1 節の多段階正弦波予測と同様に、ロールアウトは自身の誤差にさらされることになる。

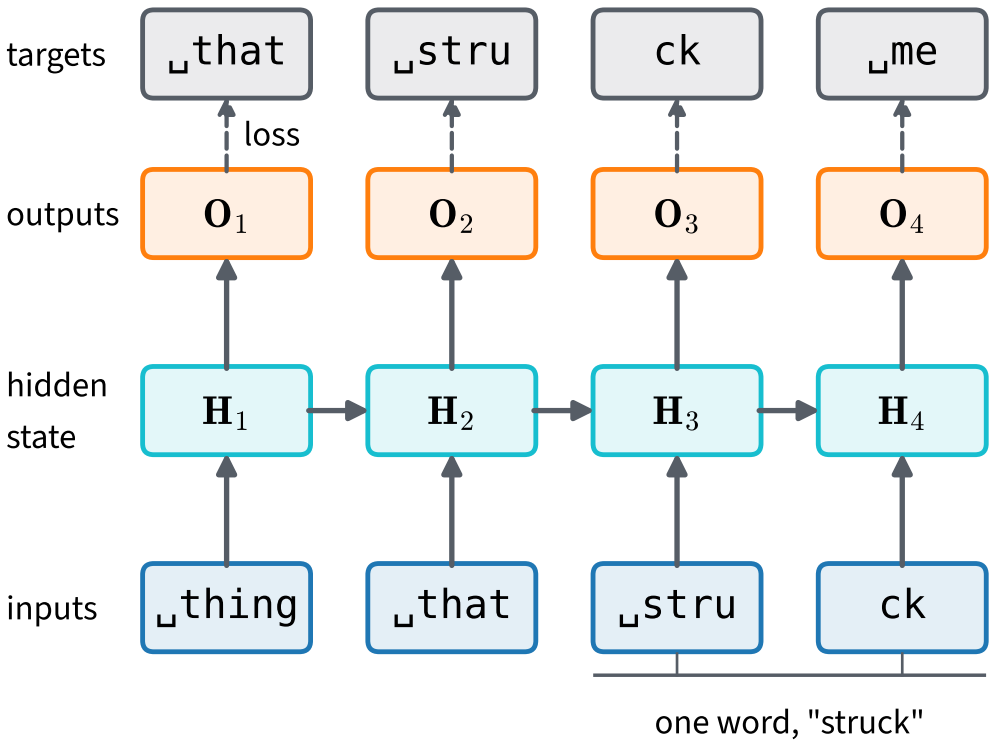


Figure 8.8: 『The Time Machine』の BPE トークン上で教師強制により訓練される RNN 言語モデル。ターゲット系列は入力を 1 トークン前にずらしたものであり、各ステップは次のトークンを予測するよう訓練される。

8.4.2 RNN 言語モデルのゼロからの実装

ここからは、この言語モデルの 3 つの構成要素（リカレントセル、埋め込み参照、出力層）をゼロから実装する。

8.4.2.1 リカレントセル

RNNScratch クラスは Equation 8.17 のパラメータを $\phi = \tanh$ として保持する。$\mathbf{W}_{\text{xh}}$ と $\mathbf{W}_{\text{hh}}$ は $\mathcal{N}(0, \sigma^2)$（ただし $\sigma = 0.01$）から抽出し、$\mathbf{b}_{\text{h}}$ はゼロで初期化する。このスケールでは、$\mathbf{W}_{\text{hh}}$ のスペクトル半径（spectral radius）は約 $\sigma\sqrt{h} \approx 0.11$ となるため、未訓練のセルは過去の状態を急速に減衰させ、必要に応じて訓練によってリカレント重みを成長させる設計になっている。

```
class RNNScratch(nnx.Module):
    """The RNN model implemented from scratch."""
    def __init__(self, num_inputs, num_hiddens, sigma=0.01, rngs=None):
        rngs = nnx.Rngs(random.getrandbits(32)) if rngs is None else rngs
        self.num_inputs, self.num_hiddens = num_inputs, num_hiddens
        self.sigma = sigma
        self.W_xh = nnx.Param(
            rngs.params.normal((num_inputs, num_hiddens)) * sigma)
        self.W_hh = nnx.Param(
            rngs.params.normal((num_hiddens, num_hiddens)) * sigma)
        self.b_h = nnx.Param(jnp.zeros(num_hiddens))
```

順伝播では、inputs（形状は（num_steps、batch_size、num_inputs））の先頭にある時間軸に沿ってループを回し、隠れ状態が与えられない場合はゼロで初期化したうえで、ステップごとの全隠れ状態のリストを最終状態とともに返す。

訓練ステップをコンパイルする際、Python のループはタイムステップごとに 1 つずつセルのコピーへと展開されるため、コンパイル時間はステップ数に応じて増加する。本節末尾の簡潔な実装では nnx.RNN を使用する。これは jax.lax.scan のもとでセルを実行し、1 回だけコンパイルする。

```
@d2l.add_to_class(RNNScratch)
def __call__(self, inputs, state=None):
    if state is None:
        # Initial state with shape: (batch_size, num_hiddens)
        state = jnp.zeros((inputs.shape[1], self.num_hiddens))
    outputs = []
    for X in inputs:  # Shape of inputs: (num_steps, batch_size, num_inputs)
        state = d2l.tanh(d2l.matmul(X, self.W_xh) +
                         d2l.matmul(state, self.W_hh) + self.b_h)
        outputs.append(state)
    return outputs, state
```

リカレントセルの次元を確認するため、まず RNNScratch の順伝播をダミーのミニバッチ X（形状 (num_steps, batch_size, num_inputs)）に対して実行する。

```
batch_size, num_inputs, num_hiddens, num_steps = 2, 16, 32, 100
rnn = RNNScratch(num_inputs, num_hiddens)
X = d2l.ones((num_steps, batch_size, num_inputs))
```

```
outputs, state = rnn(X)
```

ここでヘルパー関数 check_shape（以降の章でも再利用する）を定義し、outputs をスタックすると (num_steps, batch_size, num_hiddens) になり、state の形状が (batch_size, num_hiddens) になることを確認する。

```
def check_shape(a, shape):
    """Check the shape of a tensor."""
    assert a.shape == shape, \
            f'tensor\'s shape {a.shape} != expected shape {shape}'

check_shape(d2l.stack(outputs, 0), (num_steps, batch_size, num_hiddens))
check_shape(state, (batch_size, num_hiddens))
```

8.4.2.2 埋め込み層と出力層

RNNLMScratch は、リカレントセル rnn を埋め込みテーブル E と出力射影（W_hq、b_q）で包み、さらに埋め込みと隠れ状態に対する任意のドロップアウト率を備える。埋め込みの各行は（単位スケールのワンホット入力に合わせて）$\mathcal{N}(0, 1)$ から初期化し、出力重みはスケール $\sigma = 0.01$、バイアスはゼロとするため、未訓練のモデルはゼロに近いロジットを出力し、初期パープレキシティは $|\mathcal{V}|$ に近くなる。d2l.Classifier を継承することで平均交差エントロピー損失が提供され、validation_step をオーバーライドしてその損失を直接記録することで、その指数から検証パープレキシティが得られる。

```
class RNNLMScratch(d2l.Classifier):
    """The RNN-based language model implemented from scratch."""
    def __init__(self, rnn, vocab_size, lr=0.01, dropout=0, emb_dropout=0,
                 rngs=None):
        super().__init__()
        self.save_hyperparameters(ignore=['rnn', 'rngs'])
        self.rnn = rnn
        rngs = nnx.Rngs(random.getrandbits(32)) if rngs is None else rngs
        sigma = getattr(rnn, 'sigma', 0.01)  # Scale of the output weights
        self.E = nnx.Param(rngs.params.normal((vocab_size, rnn.num_inputs)))
        self.W_hq = nnx.Param(rngs.params.normal(
            (rnn.num_hiddens, vocab_size)) * sigma)
        self.b_q = nnx.Param(jnp.zeros(vocab_size))
        self.drop = nnx.Dropout(dropout, rngs=rngs)
        self.emb_drop = nnx.Dropout(emb_dropout, rngs=rngs)

    def validation_step(self, batch):
        return self.loss(self(*batch[:-1]), batch[-1])
```

データローダは形状（batch_size、num_steps）のミニバッチを生成するため、embedding はトークン ID を転置してから E を参照し、先頭の軸を時間とする（num_steps、batch_size、num_inputs）テンソルを返す。

```
@d2l.add_to_class(RNNLMScratch)
def embedding(self, X):
```

```
    # Output shape: (num_steps, batch_size, num_inputs)
    return self.E[X.T]
```

output_layer はステップごとの状態を形状（batch_size、num_steps、num_hiddens）にスタックし、1 回のバッチ行列積で共有の出力射影を適用する。一方、forward は埋め込み参照、ドロップアウト、リカレントセル、出力射影を順に連結する。

```
@d2l.add_to_class(RNNLMScratch)
def output_layer(self, rnn_outputs):
    # A list of per-step states, or a tensor whose first axis is time
    H = d2l.stack(list(rnn_outputs), 1)  # (batch_size, num_steps, num_hiddens)
    return d2l.matmul(H, self.W_hq) + self.b_q

@d2l.add_to_class(RNNLMScratch)
def forward(self, X, state=None):
    embs = self.emb_drop(self.embedding(X))
    rnn_outputs, _ = self.rnn(embs, state)
    H = d2l.stack(list(rnn_outputs), 0)  # (num_steps, batch_size, num_hiddens)
    return self.output_layer(self.drop(H))
```

形状 (batch_size, num_steps) のトークンインデックスからなるダミーのミニバッチを RNNLMScratch に通すと、返されるロジットテンソル outputs の形状が (batch_size, num_steps, vocab_size) であることが確認できる。

```
model = RNNLMScratch(rnn, vocab_size=1024)
outputs = model(d2l.ones((batch_size, num_steps), dtype=d2l.int32))
check_shape(outputs, (batch_size, num_steps, 1024))
```

8.4.3 勾配クリッピング

展開された RNN は T ステップにわたって同じ遷移ヤコビ行列を適用するため、勾配は T に対して指数関数的に縮小または増大しうる（8.5 節）。突発的な勾配のスパイクが生じると、パラメータはユークリッド距離で $\eta\|\mathbf{g}\|$ だけ移動し、1 次のテイラー近似が成り立つ局所領域を大きく飛び越えて、何十回分もの更新を 1 ステップで台無しにしてしまう。η を一律に小さくする代わりに、**勾配クリッピング**（gradient clipping）は（すべてのモデルパラメータにわたって一括計算した）大域的な勾配ノルムをしきい値 θ で頭打ちにする。具体的には、$\mathbf{g}$ を半径 θ の球へ射影する。

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right)\mathbf{g}. \tag{8.18}$$

長さが θ を超える勾配は方向を変えずに長さ θ に縮小され、それより短い勾配はそのまま保たれる（Figure 8.9）。

gradient_clip_val が正のとき、d2l.Trainer（2.4 節）は誤差逆伝播法の直後かつオプティマイザのステップの前に Equation 8.18 を適用する。

JAX のトレーナはコンパイルされた訓練ステップ内で optax 変換を用いてパラメータを更新するため、クリッピングもそこに組み込まれる。prepare_optimizer は clip_gradients を optax 変換に変換し、モデルのオプティマイザの手前に連結する。

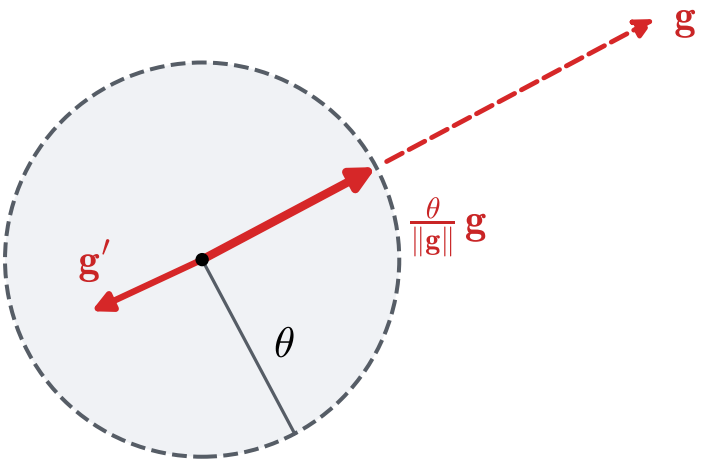


Figure 8.9: 半径 θ の球への射影としての勾配クリッピング。長い勾配 $\mathbf{g}$ は自身の方向に沿って長さ θ に再スケーリングされる一方、球の内部にある勾配 $\mathbf{g}'$ は変化しない。

```
def clip_gradients(grad_clip_val, grads):
    """Rescale the gradients to global norm at most grad_clip_val."""
    grad_leaves, _ = jax.tree_util.tree_flatten(grads)
    norm = jnp.sqrt(sum(jnp.vdot(x, x) for x in grad_leaves))
    clip = lambda grad: jnp.where(norm < grad_clip_val,
                                  grad, grad * (grad_clip_val / norm))
    return jax.tree_util.tree_map(clip, grads)

@d2l.add_to_class(d2l.Trainer)
def prepare_optimizer(self, model):
    val = self.gradient_clip_val
    key = (jax.tree.structure(model), type(model).configure_optimizers, val)
    if key not in d2l.optimizer_cache:
        tx = model.configure_optimizers()
        if val > 0:  # The closure holds only val, not the trainer
            clip = lambda grads, params: clip_gradients(val, grads)
            tx = optax.chain(optax.stateless(clip), tx)
        d2l.optimizer_cache.clear()
        d2l.optimizer_cache[key] = tx
    self.optim = nnx.Optimizer(model, d2l.optimizer_cache[key],
                               wrt=nnx.Param)
```

8.4.4 訓練と評価

`d2l.TimeMachine`（8.3 節）はこの中編小説の 65,741 個の BPE トークンを符号化し、長さ $T = 32$ の重複するウィンドウからなるミニバッチを生成する。訓練用が 50,000 個、検証用が 5,000 個であり、検証トークンが訓練ターゲットに現れないよう両者の間には 32 ウィンドウ分の間隔が空けられている。各ウィンドウは $\mathbf{H}_0 = \mathbf{0}$ から始まるため、誤差逆伝播法は 32 ステップで**打ち切られる**（truncated、8.5 節）。

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
```

8.4.4.1 単純なモデルとクリッピングの役割

まず、222,915 パラメータのベースラインモデル（$d = 64$、$h = 128$、ドロップアウトなし）を SGD の学習率 $\eta = 4$ でインスタンス化する。

```
rnn = RNNScratch(num_inputs=64, num_hiddens=128)
model = RNNLMScratch(rnn, vocab_size=len(data.vocab), lr=4)
```

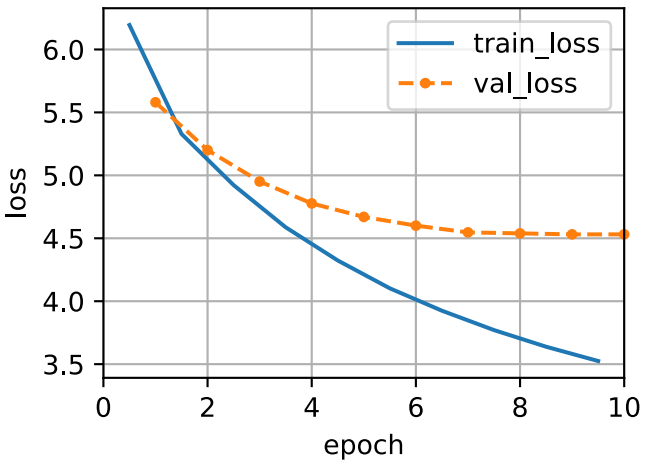
訓練を始める前に、ランダムな初期埋め込み行列 `model.E` の NumPy コピーを `E_init` に保存しておき、SGD によってどれだけ移動するかを測定できるようにする。

```
E_init = d2l.numpy(model.E[...]).copy()
```

次に、未訓練モデルの初期検証パープレキシティ `ppl_init` を評価してから、`model` を 10 エポックにわたり `gradient_clip_val=1` の勾配クリッピング付きで訓練する。

```
trainer = d2l.Trainer(max_epochs=10, gradient_clip_val=1, num_gpus=1)
ppl_init = math.exp(trainer.evaluate(model, data)['loss'])
print(f'initial validation perplexity {ppl_init:.0f}')
trainer.fit(model, data)
```

```
initial validation perplexity 1027
```



最終的な検証交差エントロピー損失 `trainer.history['val_loss'][-1]` の指数をとると、訓練済みベースラインの検証パープレキシティ `ppl_plain` が得られる。

```
ppl_plain = math.exp(trainer.history['val_loss'][-1])
print(f'validation perplexity {ppl_plain:.1f}')
```

```
validation perplexity 92.9
```

初期検証パープレキシティは $|\mathcal{V}| = 1{,}027$ 付近にあり、10 エポック後にクリッピングありのモデルは検証パープレキシティ約 85〜100 に達する。ただし最初のエポックの途中では、損失が初期値を大きく上回って跳ね上がることがよくある（一方、222,915 個のパラメータが 50,000 個の訓練トークンに過学習し始めるため、訓練パープレキシティはその半分以下にまで下がる）。

$\eta = 4$ において勾配クリッピングが不可欠である理由を見るため、まったく同じアーキテクチャと学習率を持つ新しいベースラインインスタンスを構築するヘルパー関数 `unclipped_model()` を定義する。

```
def unclipped_model():
    return RNNLMScratch(RNNScratch(num_inputs=64, num_hiddens=128),
                        vocab_size=len(data.vocab), lr=4)
```

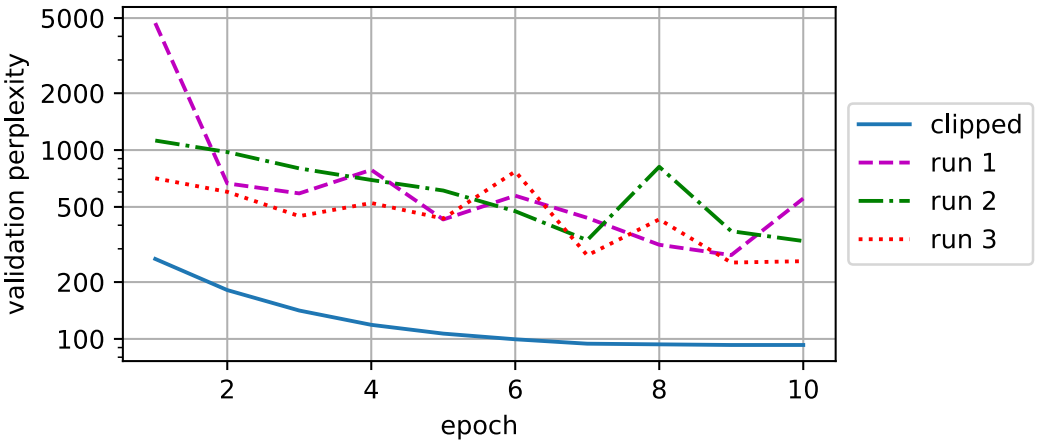
ここで、シード未固定の独立した 3 つのインスタンスを勾配クリッピングなしで訓練し、各実行のエポックごとの検証パープレキシティを `ppl_runs` に記録して、その最大値と最終値を報告する。

```
ppl_runs = []
for run in range(3):  # Three unseeded runs, no clipping
    trainer_unclipped = d2l.Trainer(max_epochs=10, num_gpus=1)
    trainer_unclipped.board.display = False  # One plot for all runs below
    trainer_unclipped.fit(unclipped_model(), data)
    ppls = [math.exp(loss) for loss in trainer_unclipped.history['val_loss']]
    ppl_runs.append(ppls)
    print(f'run {run + 1}: highest {max(ppls):.3g}, final {ppls[-1]:.3g}')
finals = [ppls[-1] for ppls in ppl_runs]
print(f'{sum(f > 2 * ppl_plain for f in finals)} of 3 runs end above twice '
      f'the clipped {ppl_plain:.3g}; finals from {min(finals):.3g} to '
      f'{max(finals):.3g}')
```

```
run 1: highest 4.7e+03, final 554
run 2: highest 1.12e+03, final 330
run 3: highest 770, final 258
3 of 3 runs end above twice the clipped 92.9; finals from 258 to 554
```

`ppl_runs` に格納したクリッピングなしの 3 本の軌跡を、クリッピングありのベースライン `ppl_clipped` とともに対数軸上にプロットすると、その不安定さは一目瞭然となる。

```
ppl_clipped = [math.exp(loss) for loss in trainer.history['val_loss']]
d2l.plot(list(range(1, 11)), [ppl_clipped] + ppl_runs, 'epoch',
         'validation perplexity', yscale='log',
         legend=['clipped', 'run 1', 'run 2', 'run 3'],
         legend_loc='outside', figsize=(5, 2.5))
```



クリッピングなしでは、勾配のスパイクによって検証パープレキシティが激しく跳ね上がり、未訓練ベースラインの 1,027 をはるかに超えることさえある。そして大半の実行は、クリッピングありのモデルの 2 倍を超えるパープレキシティで終了する。クリッピングありの検証パープレキシティをバイトあたりビット数（bits per byte、8.3.2.4 節）に変換するには、$\log_2$ ppl を BPE トークンあたりの平均 UTF-8 バイト数（≈ 2.8）で割る。

```
ids = d2l.numpy(data.X[data.num_train:data.num_train+data.num_val, 0]).tolist()
bytes_per_token = len(data.tokenizer.decode(ids).encode('utf-8')) / len(ids)
print(f'{bytes_per_token:.2f} bytes/token, '
      f'{math.log2(ppl_plain) / bytes_per_token:.2f} bits per byte')
```

```
2.78 bytes/token, 2.35 bits per byte
```

8.4.4.2 SGD 下での埋め込みとドロップアウト付き幅広モデル

10 エポックの SGD を通じて $\mathbf{E}$ の各行がどれだけ移動したかを調べると、第 2 のボトルネックが浮かび上がる。まず、訓練済みの埋め込みテーブルを NumPy 配列 `E_final` に取り出す。

```
E_final = d2l.numpy(model.E[...])
```

すべての語彙行にわたって相対ユークリッド変位 `change`（`E_final` と `E_init` の差）を計算すると、各トークンベクトルがいかにわずかしか動いていないかが明らかになる。

```
change = (((E_final - E_init) ** 2).sum(1) / (E_init ** 2).sum(1)) ** 0.5
print(f'mean change of an embedding row: {change.mean():.1%} of its length')
```

```
mean change of an embedding row: 4.2% of its length
```

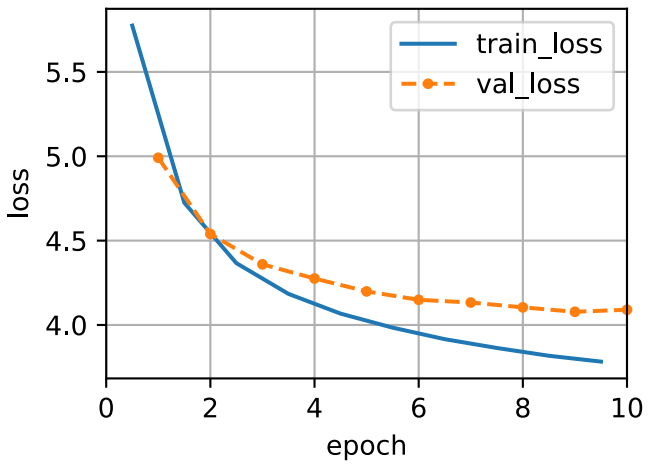
第 i 行はトークン i が出現した位置からのみ勾配を受け取るうえ、損失はミニバッチ内の全 $1{,}024 \times 32 = 32{,}768$ トークンにわたって平均されるため、一様な学習率の SGD では低頻度や中頻度のトークンが受け取る勾配はごく小さくなり、$\mathbf{E}$ はほぼ凍結されたままになる。Adam（9.5 節）のような適応的オプティマイザは座標ごとのステップサイズを再スケーリングして埋め込みをはるかに大きく動かすが（演習問題で扱う）、SGD の枠内で直接対処することもできる。埋め込み幅を $d = 64$ から $d = 512$ へと広げれば、密に更新される入力行列 $\mathbf{W}_{\text{xh}}$ にはるかに豊か

なランダム特徴量基底が与えられ、さらにドロップアウト（埋め込みに 0.1、隠れ状態に 0.5）を加えることで過学習を抑えられる。

```
rnn = RNNScratch(num_inputs=512, num_hiddens=128)
model = RNNLMScratch(rnn, vocab_size=len(data.vocab), lr=4, dropout=0.5,
                     emb_dropout=0.1)
```

この正則化された幅広モデルを、同じ学習率 `lr=4` とクリッピングしきい値 `gradient_clip_val=1` を用いて 10 エポック訓練する。

```
trainer = d2l.Trainer(max_epochs=10, gradient_clip_val=1, num_gpus=1)
trainer.fit(model, data)
```



最終的な検証損失から、検証パープレキシティ `ppl_scratch` と、それに対応する圧縮率であるバイトあたりビット数の両方を計算する。

```
ppl_scratch = math.exp(trainer.history['val_loss'][-1])
print(f'validation perplexity {ppl_scratch:.1f}, '
      f'{math.log2(ppl_scratch) / bytes_per_token:.2f} bits per byte')
```

```
validation perplexity 59.8, 2.12 bits per byte
```

検証パープレキシティは 85〜100 から 60 前後へと低下し、バイトあたりビット数は 2.3〜2.4 から約 2.1 へと下がる。

8.4.4.3 Kneser–Ney 法との比較と実効コンテキスト範囲

バイトあたり 2.1 ビットという値は、補間 Kneser–Ney 平滑化（8.3.2.3 節）と比べてどうだろうか。Kneser–Ney トライグラムが達成したバイトあたり 1.9 ビット（8.3 節）は、そのまま比較基準にはできない。この数値は英小文字と空白のみに整形したテキスト上で測定されたものであるのに対し、本節では大文字・句読点・改行を含む元の小説をそのままモデル化しているからである。そこで、Kneser–Ney n-gram（$n = 2, 3, 4$）をまったく同じ 50,000 個の BPE 訓練トークンに当てはめ、`val[32:]` 上で評価する。ここでは各トークンが、それぞれの 32 トークン検証ウィンドウの最終ターゲットとしてスコア付けされる。

```
first_tokens = d2l.numpy(data.X[:, 0]).tolist()  # First token of each window
train = first_tokens[:data.num_train]
val = first_tokens[data.num_train:data.num_train + data.num_val]
lp_kns = {n: d2l.kneser_ney(train, val, len(data.vocab), n)[32:]
          for n in (2, 3, 4)}
for n, lp in lp_kns.items():
    bits = -sum(lp) / len(lp) / math.log(2)
    print(f'KN {n}-gram: {bits / bytes_per_token:.2f} bits per byte')
lp_kn = lp_kns[3]
```

```
KN 2-gram: 2.24 bits per byte
KN 3-gram: 2.09 bits per byte
KN 4-gram: 2.08 bits per byte
```

訓練済み RNN をまったく同じホールドアウトトークン `val[32:]` 上でスコア付けするため、検証ウィンドウのスライス `[start:stop]`（その最終ターゲットが `val[32:]` を網羅する範囲）を選択する。

```
start, stop = data.num_train, data.num_train + data.num_val - 32  # val[32:]
```

`model` を評価モードにして `data.X[start:stop]` 上で 1,024 ウィンドウずつのバッチとして実行し、ウィンドウ内の全 32 ステップにわたる位置ごとのターゲット対数確率の行列 `logp` を収集する。

```
model_eval = nnx.view(model, deterministic=True, raise_if_not_found=False)
X_val, Y_val = data.X[start:stop], data.Y[start:stop]
logp = d2l.numpy(jnp.concatenate([  # Log-probability of every target
    jnp.take_along_axis(jax.nn.log_softmax(model_eval(X_val[i:i + 1024])),
                        Y_val[i:i + 1024, :, None], -1)[..., 0]
    for i in range(0, len(X_val), 1024)]))
```

RNN とトライグラムはほぼ同じ平均値に到達する。では、両者は同じ予測をしているのだろうか。`logp` の最終列は、トークン列 `val[32:]` に対して `lp_kn` と同じ対象をスコア付けしているため、2 つのモデルを 1 トークンずつ比較できる。あわせて、ニューラル言語モデルとカウントに基づくモデルを組み合わせる定石である等重み混合 $0.5\,P_{\text{RNN}} + 0.5\,P_{\text{KN}}$ もスコア付けする (Mikolov et al. 2011)。

```
lp_rnn, lp_tri = logp[:, -1], np.array(lp_kn)
lp_mix = np.logaddexp(lp_rnn, lp_tri) + math.log(0.5)  # 0.5 P_RNN + 0.5 P_KN
for name, lp in [('RNN', lp_rnn), ('KN trigram', lp_tri), ('mixture', lp_mix)]:
    print(f'{name}: {-lp.mean() / math.log(2) / bytes_per_token:.2f} '
          'bits per byte')
print(f'RNN more confident on {np.mean(lp_rnn > lp_tri):.0%} of the tokens')
```

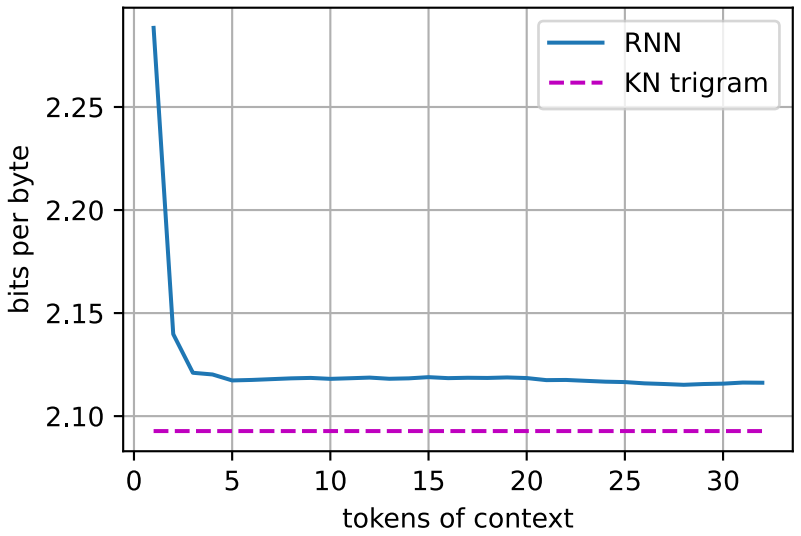
```
RNN: 2.12 bits per byte
KN trigram: 2.09 bits per byte
mixture: 2.03 bits per byte
RNN more confident on 50% of the tokens
```

2 つのモデルは平均では引き分けるが、トークン単位ではそうではない。約半数のトークンでは RNN が、残りの半数ではトライグラムがより高い確率を割り当てており、混合モデルはトライグラムよりバイトあたり約 0.06 ビット低くなり、単独のどちらのモデルよりも優位に立つ。

混合モデルが改善を得られるのは構成モデル同士の予測が食い違う箇所に限られるため、2 つのモデルはテキストについて部分的に異なる事柄を学習していることになる。そうなると、この引き分けはいっそう謎めいて見える。32 トークン分の履歴を読み取るリカレントネットワークが、わずか 2 トークンしか振り返らない Kneser–Ney トライグラムと同じバイトあたり ≈ 2.1 ビット付近に着地するのはなぜだろうか。訓練済み RNN が実際にどれだけの文脈を利用しているかを調べるため、各ウィンドウ位置 $1, \ldots, 32$ における平均検証バイトあたりビット数を Kneser–Ney トライグラムのベースラインに対してプロットする。

```
bits = -logp.mean(0) / math.log(2) / bytes_per_token  # One per position
kn_bits = -sum(lp_kn) / len(lp_kn) / math.log(2) / bytes_per_token
positions = np.arange(1, data.num_steps + 1)
d2l.plot(positions, [bits, [kn_bits] * data.num_steps],
         'tokens of context', 'bits per byte', legend=['RNN', 'KN trigram'],
         figsize=(4.5, 3))
print('bits per byte with 1, 2, 3, 4, 8, 32 tokens of context:',
      [round(float(bits[k - 1]), 2) for k in (1, 2, 3, 4, 8, 32)])
```

```
bits per byte with 1, 2, 3, 4, 8, 32 tokens of context: [2.29, 2.14, 2.12, 2.12, 2.12, 2.12]
```



この診断曲線から引き分けの理由がわかる。改善のほぼすべては 2 トークン目の文脈で得られており、損失はバイトあたり約 2.3 ビットから約 2.14 ビットへと約 0.15 下がる。3 トークン目がもたらす改善は 0.01〜0.02 程度にとどまり、位置 4 から 32 まで曲線は平坦である。順伝播において隠れ状態は全 32 トークンに依存**しうる**ものの、訓練されたモデルが実際に活用しているのは直近の 2〜3 トークンだけである。

なぜこれほど少ないのだろうか。1 つの説明は最適化にある。離れたトークンを利用するようセルに教えるはずの勾配が、逆方向へ伝わるステップごとに縮小してしまうというものであり、これは 8.5 節で測定する。もう 1 つの説明はデータにある。50,000 個の訓練トークンでは、2〜3 個以上前の位置にあるトークンから学習できる情報がそもそもほとんど含まれていないのかもしれない。上で出力した Kneser–Ney モデルの結果は 2 番目の説明を裏づけている。これらのモデルはカウントによって当てはめられるため消失しうる勾配が存在しないにもかかわらず、同じ箇所で飽和する。すなわち、トライグラムはバイグラムよりバイトあたり 0.15 ビット改善するのに対し、4-gram のトライグラムからの改善はわずか 0.01 にすぎない。ここでの実験だけでは、この 2 つ

の説明を切り分けることはできない。明確に検証するには、同じトークン列で訓練されつつ長いラグにわたって勾配が生き残るモデルが必要であり、第 12 章ではそのようなモデルを構築する。

8.4.5 テキスト生成

訓練が完了すると、この言語モデルは `predict` を通じて自己回帰的にテキストを生成する。まずユーザーのプレフィックスのトークン列上で隠れ状態をウォームアップし、続いて選択した各トークンを次の入力としてフィードバックしながら `num_tokens` 個の後続トークンを生成する（`temperature=0.0` のときは貪欲な argmax を用い、それ以外のときは温度スケーリングされたソフトマックスからサンプリングする。8.6 節を参照）。

```
@d2l.add_to_class(RNNLMScratch)
def predict(self, prefix, num_tokens, tok, device=None, temperature=0.0):
    model = nnx.view(self, deterministic=True, use_running_average=True,
                     raise_if_not_found=False)
    outputs, state = tok.encode(prefix), None
    for i in range(len(outputs) - 1):  # Warm up on the prefix
        X = d2l.tensor([[outputs[i]]])
        _, state = model.rnn(model.embedding(X), state)
    for _ in range(num_tokens):  # Generate num_tokens continuation tokens
        X = d2l.tensor([[outputs[-1]]])
        rnn_outputs, state = model.rnn(model.embedding(X), state)
        logits = model.output_layer(rnn_outputs)[0, 0]
        if temperature == 0:  # Greedy: the highest-scoring token
            token = logits.argmax()
        else:  # Sample from the softmax of logits / temperature
            token = jax.random.categorical(d2l.get_key(), logits / temperature)
        outputs.append(int(token))
    return tok.decode(outputs)
```

`model.predict` を貪欲デコーディング（greedy decoding、`temperature=0.0`）でテストし、プロンプト `'The Time Traveller'` から 50 トークンの続きを生成してみよう。

```
d2l.print_wrapped(repr(model.predict('The Time Traveller', 50,
                                     data.tokenizer)))
```

```
'The Time Traveller. I had the same\nfounding the bushes. I had the
    same\nfounding the bushes. I had the same\nfounding the bushes. I had
    the same\nfounding the b'
```

貪欲デコーディングは決定論的であるため、隠れ状態が直近のフレーズに近い状態を再訪すると、そのループを際限なく繰り返してしまう。この失敗モードについては 8.6 節で詳しく調べ、対処する。

8.4.6 簡潔な実装

フレームワークが提供するリカレント層は、リカレントステップと時間ループを最適化された低水準カーネルに融合している。以下の `RNN` モジュールは `RNNScratch` のインターフェースおよび初期化と一致しており、`RNNLM` は `RNNLMScratch` を継承して組み込みの埋め込み層と線形層に差し替える。

Flax NNX は単純なリカレントセルとして `SimpleCell` を提供している。これは 1 タイムステップ分の入力と明示的な隠れ状態を受け取り、`nnx.RNN` がそれを入力系列全体にわたってスキャンする。セルの入力射影がバイアスを持ち、リカレント射影にはバイアスがないため、`RNNScratch` と同じくセルのバイアスベクトルは 1 つである。

```
class RNN(nnx.Module):
    """The RNN model implemented with high-level APIs."""
    def __init__(self, num_inputs, num_hiddens, sigma=0.01, rngs=None):
        rngs = nnx.Rngs(random.getrandbits(32)) if rngs is None else rngs
        self.num_inputs, self.num_hiddens = num_inputs, num_hiddens
        self.sigma = sigma
        normal = nnx.initializers.normal(sigma)
        cell = nnx.SimpleCell(num_inputs, num_hiddens, kernel_init=normal,
                              recurrent_kernel_init=normal, rngs=rngs)
        self.rnn = nnx.RNN(cell, time_major=True, return_carry=True,
                           rngs=rngs)

    def __call__(self, inputs, H=None):
        H, outputs = self.rnn(inputs, initial_carry=H)
        return outputs, H
```

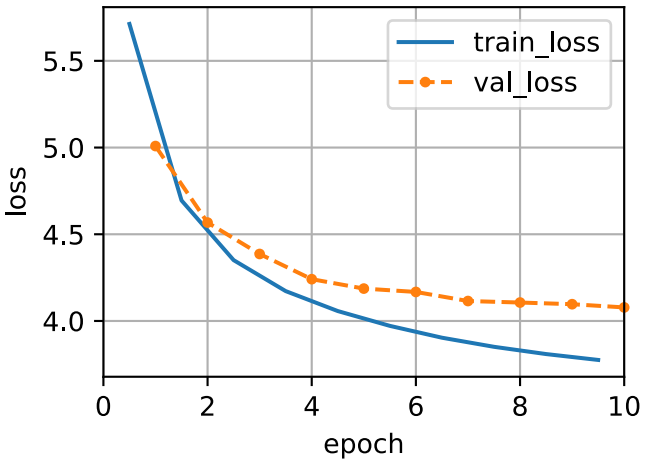
```
class RNNLM(d2l.RNNLMScratch):
    """The RNN-based language model implemented with high-level APIs."""
    def __init__(self, rnn, vocab_size, lr=0.01, dropout=0, emb_dropout=0,
                 rngs=None):
        d2l.Classifier.__init__(self)
        self.save_hyperparameters(ignore=['rnn', 'rngs'])
        self.rnn = rnn
        rngs = nnx.Rngs(random.getrandbits(32)) if rngs is None else rngs
        sigma = getattr(rnn, 'sigma', 0.01)  # Scale of the output weights
        self.emb = nnx.Embed(vocab_size, rnn.num_inputs,
                             embedding_init=nnx.initializers.normal(1.0),
                             rngs=rngs)
        self.linear = nnx.Linear(
            rnn.num_hiddens, vocab_size,
            kernel_init=nnx.initializers.normal(sigma), rngs=rngs)
        self.drop = nnx.Dropout(dropout, rngs=rngs)
        self.emb_drop = nnx.Dropout(emb_dropout, rngs=rngs)

    def embedding(self, X):
        return self.emb(X.T)

    def output_layer(self, hiddens):
        return d2l.swapaxes(self.linear(hiddens), 0, 1)
```

この高水準 API による実装を検証するため、`RNN` と `RNNLM` を同じハイパーパラメータ（$d = 512$、$h = 128$、およびドロップアウト）でインスタンス化する。

```
rnn = RNN(num_inputs=512, num_hiddens=128)
model = RNNLM(rnn, vocab_size=len(data.vocab), lr=4, dropout=0.5,
              emb_dropout=0.1)
```

ここで RNNLM を 10 エポックにわたり gradient_clip_val=1 で訓練する。融合されたリカレントカーネルが動作するため、ゼロからの実装に比べてごくわずかな実時間で完了する。

```
trainer = d2l.Trainer(max_epochs=10, gradient_clip_val=1, num_gpus=1)
trainer.fit(model, data)
```



最後に、この簡潔なモデルの検証パープレキシティ ppl_concise を、'The Time Traveller' から貪欲に生成した 30 トークンの続きとともに報告する。

```
ppl_concise = math.exp(trainer.history['val_loss'][-1])
pred = model.predict('The Time Traveller', 30, data.tokenizer)
d2l.print_wrapped(f'perplexity {ppl_concise:.1f}, {pred!r}')
```

```
perplexity 59.0, 'The Time Traveller. I had a faintness. I was a\nface of
    the\ndeliculture. I was a\nface of'
```

8.4.7 まとめ

リカレントニューラルネットワークは、すべてのタイムステップで共有される重みを用いて現在の入力と直前の状態から固定サイズの隠れ状態を更新し、履歴を非可逆な状態ベクトルに圧縮する代償として、1 ステップあたり一定のメモリ量と計算量を実現する。RNN 言語モデルは、埋め込みテーブル、tanh リカレントセル、および語彙サイズの出力射影（ベースラインのパラメータの 89% がここに存在する）を組み合わせ、教師強制によってトークンあたりの交差エントロピーを最小化するよう訓練される。大域的な勾配クリッピング（Equation 8.18）は、高い学習率のもとで勾配爆発のスパイクが訓練を破綻させるのを防ぐ。『The Time Machine』では、ドロップアウト付きの幅広な埋め込みにより、SGD で訓練した RNN はバイトあたり ≈ 2.1 ビットに達し、補間 Kneser–Ney トライグラムと引き分ける。訓練されたバニラ RNN が実質的に活用しているのは直近の 2〜3 トークンの文脈のみであり、その勾配がどこまで届くかは 8.5 節で測定する。

8.4.8 演習問題

1. **入力積の分離。** $\mathbf{X}_t \in \mathbb{R}^{b\times d}$ と $\mathbf{H}_{t-1} \in \mathbb{R}^{b\times h}$ を Equation 8.17 における入力と状態とする。

   1. 訓練時、ウィンドウ内の入力 $\mathbf{X}_1, \ldots, \mathbf{X}_T$ は最初のタイムステップを計算する前にすべて既知である。2 つの積 $\mathbf{X}_t\mathbf{W}_{\text{xh}}$ と $\mathbf{H}_{t-1}\mathbf{W}_{\text{hh}}$ のうち、全 T ステップ分を 1 回の行列積でまとめて計算できるのはどちらか、またその各因子の形状は何か。タイムステップに関する逐次ループの内側に残さなければならない積はどちらか、またそれはなぜか。
   2. ループ内に残る 1 ステップあたりの計算量を、連結形式 $[\mathbf{X}_t, \mathbf{H}_{t-1}]$ の 1 ステップあたりの計算量と比較せよ。生成時には、ステップ t の入力はステップ $t-1$ で生成されたトークンとなる。このときも分離形式の利点は保たれるか。

2. **露出バイアス**。教師強制（8.4.1.2 節）ではすべての訓練ステップが真のプレフィックスに条件づけられるのに対し、生成時には各ステップがモデル自身の過去の出力に条件づけられる。この不一致を**露出バイアス**と呼ぶ (Ranzato et al. 2016)。

   1. 生成時に各トークンをモデルの次トークン分布からサンプリングするものとし、プレフィックスが正しい限りモデルは正しい次トークンに確率 $1-\epsilon$ を割り当てるとする。生成された T 個のトークンに誤りが 1 つも含まれない確率はいくらか。また、$\epsilon = 0.01$、$T = 100$ のときの値を求めよ。
   2. **スケジュールドサンプリング** (Bengio et al. 2015) は、訓練中にモデルを自身の出力にさらす手法であり、各ステップである確率に従い、真のトークンの代わりに直前のステップにおけるモデルの予測からサンプリングしたトークンを入力とする。これは 8.4.1.2 節が依拠している教師強制のどの性質を手放すことになり、それによって訓練ウィンドウの計算はどのように変わるか。

3. **漸化式の内部でのウィンドウ表現**。ニューラル n-gram は直近 m 個の入力埋め込みからなる固定ウィンドウを読み取る。Equation 8.17 において ϕ を恒等写像、$\mathbf{b}_{\text{h}} = \mathbf{0}$ とし、状態に $h = md$ 個のユニットを持たせる。

   1. $\mathbf{W}_{\text{xh}}$ と $\mathbf{W}_{\text{hh}}$ を適切に選び、$\mathbf{H}_0 = \mathbf{0}$ から始めたとき、状態が連結 $\mathbf{H}_t = [\mathbf{X}_t, \mathbf{X}_{t-1}, \ldots, \mathbf{X}_{t-m+1}]$（すべての $t \geq m$ に対して）となるようにせよ。
   2. $\mathbf{W}_{\text{hh}}^m = \mathbf{0}$ であることを示し、このことから m ステップより前の入力が $\mathbf{H}_t$ に影響しない理由を説明せよ。
   3. $d = 256$、$m = 20$ のとき、この構成には何個の状態ユニットが必要か。状態が 256 ユニットしかない RNN は、ウィンドウをそのまま保持する代わりに何をしなければならないか説明せよ。

4. [code] **積としての埋め込み参照**。$\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ を埋め込みテーブルとし、$\mathbf{x}$ に b トークンからなるミニバッチのトークン ID を格納するものとする。

   1. 入力を、一度は $\mathbf{E}$ の行のうち $\mathbf{x}$ が指定するものを選択する方法で、もう一度は $b \times |\mathcal{V}|$ の $\mathbf{x}$ のワンホット行列と $\mathbf{E}$ との積として計算し、両者が一致することを確認せよ。$|\mathcal{V}| \in \{10^3, 10^4, 10^5\}$ および $d = b = 256$ に対して両方の実行時間を計測し、それぞれの実行時間が $|\mathcal{V}|$ に対してどのように増加するか述べよ。
   2. 選択された行の全要素の和を、自動微分によって $\mathbf{E}$ に関して微分せよ。勾配のうち非ゼロとなるのはどの行か、またあるトークンがミニバッチ内に複数回出現するとき、その行には何が入るか。$|\mathcal{V}| = 10^4$ とし、一様ランダムに抽出した $b = 256$ 個の ID に対して、非ゼロの勾配を受け取る行の割合を見積もれ。

5. **任意のウィンドウを超えるパリティ**。●$x_1, x_2, \ldots$ をビット列とし、ステップ t で求める出力を $x_1 + \cdots + x_t$ のパリティとする。

   1. 有限の m をどのように選んでも、直近 m 個の入力しか読み取らないモデルは、m より長い系列のすべてのステップで正しい出力を生成できないことを示せ。
   2. ϕ をしきい値活性化関数とする。任意の系列長に対して毎ステップの出力がパリティに一致するような、隠れユニット数 2 の Equation 8.17 形式の RNN を構成せよ。具体的な重みを与え、隠れユニット 1 個では不十分であることを示せ。

3. 以降のすべてのステップで出力が正しくなるために、隠れ状態は $x_1, \ldots, x_t$ について何ビットの情報を保持しなければならないか。これを 8.4.1.1 節における「非可逆な固定サイズの要約」としての状態の説明と関連づけよ。

6. [code] **勾配クリッピングのアブレーション**。 `d2l.Trainer` は、`gradient_clip_val` が正のとき常に射影 Equation 8.18 を適用する。
   1. 最初のエポックの各更新におけるクリッピング前の勾配ノルムを記録し、プロットせよ。クリッピングはどのくらいの頻度で、どのタイミングで作動するか。
   2. 本節のクリッピングなし実行の大半は `lr=4` で跳ね上がった。3 回連続でクリッピングなし実行が検証パープレキシティの急騰を起こさずに訓練できるまで、学習率を半分ずつ下げよ。その 10 エポック後の検証パープレキシティは、`lr=4` でクリッピングした実行と比べてどうか。このことから、クリッピングは何を可能にしているといえるか。
   3. `RNNScratch` の tanh を ReLU に置き換え、クリッピングありとなしの両方で再訓練せよ。クリッピングは依然として必要か。tanh セルとの違いを隠れ状態の値域の観点から説明せよ。

7. [code] **利用可能な履歴**。すべての訓練ウィンドウはゼロ状態から始まるため、モデルは `num_steps` トークンを超える文脈を使うようには一切訓練されておらず、位置別の損失からはそれよりはるかに少ない文脈しか使っていないことがわかった。対照的に `predict` は、プレフィックス全体と生成されたすべてのトークンにわたって状態を引き継ぐ。訓練済みモデルを連続する 512 個の検証トークン上で次の 2 通りの方法で評価せよ。すなわち、訓練時と同様に 32 トークンごとに状態をゼロにリセットする方法と、全 512 トークンにわたって状態を引き継ぐ方法である。両者の平均交差エントロピーを比較せよ。状態の引き継ぎは性能を改善するか、悪化させるか、あるいは差を生まないか。また、その答えは訓練時のコンテキスト範囲を超えてモデルが到達する状態について何を示唆しているか。

8. [code] **Adam**。幅広の埋め込みを持つモデルを、SGD の代わりに 9.5 節の Adam オプティマイザで訓練せよ。具体的には、`RNNLMScratch` を継承し、学習率 0.003 の Adam を返すよう `configure_optimizers` をオーバーライドし、クリッピングは 1 のまま維持せよ。
   1. `emb_dropout=0.2` とし、上の比較と同様にトークン列 `val[32:]` をスコア付けし、そのバイトあたりビット数を SGD モデル、Kneser–Ney トライグラムおよび 4-gram、さらに $\lambda = 0.5$ における RNN とトライグラムの混合モデルと比較せよ。
   2. `emb_dropout=0.1` と 0.2 のそれぞれについて、各エポック後の検証損失を記録せよ。検証損失が最小になるのは何エポック目か、またその後はどうなるか。
   3. 訓練中に埋め込みの各行がどれだけ移動するかを再度測定せよ。Adam が SGD よりも大きく埋め込みを動かすのはなぜか。

9. [code] **語彙サイズとバイトあたりビット数**。●幅広の埋め込みを持つモデルを、`vocab_size` を 512、1,024、2,048、4,096 として `d2l.TimeMachine` 上で再訓練せよ。ウィンドウ数は語彙サイズに依存するため、割合で分割する。すなわち、各データセットを一度構築して `n = len(data.X)` を読み取り、次に `num_train=int(0.8 * n)` および `num_val=n - int(0.8 * n)` を指定して再構築せよ。（後から `data.num_train` を書き換えると、検証トークンを訓練ターゲットから除外する処理が飛ばされてしまう。）これにより、すべてのモデルがほぼ同じ範囲のテキストで訓練・検証される。トークンあたりのパープレキシティは直接比較できないため、バイトあたりビット数で比較せよ。
   1. バイトあたりビット数が最も低くなるのはどの語彙サイズか。
   2. 埋め込みテーブルと出力層のパラメータ数、およびトークン単位でのコーパス長は、語彙サイズに応じてどのように変化するか。測定した傾向を最もよく説明するのはこれらの効果のうちどれか。

10. [code] **書籍間の汎化**。●`d2l.TimeMachine` を継承し、『The War of the Worlds』（Project

Gutenberg ebook 36）など別の H. G. Wells の小説のテキストを返すよう `_download` をオーバーライドせよ。その上でトークナイザとモデルを訓練せよ。

1. 新しい小説のトークナイザで両方を符号化し、その小説自身のホールドアウトテキストと『The Time Machine』のそれぞれでモデルを評価せよ。両方についてパープレキシティとバイトあたりビット数を報告せよ。
2. $s = \log_2(\text{ppl})$ をトークンあたりのサプライザル、b を評価対象テキストのトークンあたりバイト数とすると、バイトあたりビット数は s/b となる。『The Time Machine』（TM）におけるバイトあたりビット数と、新しい小説のホールドアウトテキスト（own）におけるバイトあたりビット数の比は、次のように分解できる。

$$\frac{s_{\text{TM}}}{s_{\text{own}}} \cdot \frac{b_{\text{own}}}{b_{\text{TM}}}. \tag{8.19}$$

   第 1 の因子はモデルの予測を反映し、第 2 の因子はトークナイザの圧縮効率を反映する。両方の因子を測定せよ。1 からより大きく離れているのはどちらか。

8.5 通時的誤差逆伝播法

8.4.2 節における RNN 言語モデルの実験からは、2 つの際立った経験的知見が得られた。勾配クリッピング（gradient clipping）なしでは大半の訓練が不安定化した一方、クリッピングを行うと訓練は安定して収束したものの、学習されたネットワークは 32 トークンのコンテキストウィンドウのうち直近の数トークンしか利用していなかったのである。両方の現象は、まったく同じ数学的根源を共有している。リカレントネットワークはすべてのタイムステップで同じ重み行列を適用するため、系列を逆方向に伝播する勾配には、各ステップで状態ヤコビ行列（state Jacobian）が乗じられる。多数のステップを経るうちにこれらの因子が複利のように積み重なり、勾配は幾何級数的に増大または縮小する。

スカラーの漸化式 $h_t = w_{\text{h}} h_{t-1} + x_t$ を見るだけでも、その理由は明らかである。これを展開すると $h_t = \sum_{s \le t} w_{\text{h}}^{t-s} x_s$ となり、したがって $\partial h_t / \partial h_{t-k} = w_{\text{h}}^{k}$ となる。40 ステップにわたると、$w_{\text{h}} = 0.9$ では勾配が $0.9^{40} \approx 0.015$ まで縮小するのに対し、$w_{\text{h}} = 1.1$ では $1.1^{40} \approx 45$ まで膨張する。勾配爆発（exploding gradients）は最適化を即座に不安定化させるため、更新のクリッピングを余儀なくされる。一方、勾配消失（vanishing gradients）は目立たないがより厄介であり、遠く離れたトークンからの誤差信号を減衰させ、モデルが長期依存関係を学習できなくしてしまう。

RNN における勾配計算に、新しい微分アルゴリズムは必要ない。漸化式を時間方向に展開して重みを共有する深いフィードフォワードネットワークとし、標準的な誤差逆伝播法（4.3 節）を適用すればよい。この手順は**通時的誤差逆伝播法**（backpropagation through time、BPTT）として知られる (Werbos 1990)。以下では、逆方向の漸化式を行列形式で直接導出し、8.4.2 節で訓練した RNN の内部で勾配が実際にどこまで伝わるかを測定したうえで、打ち切り型通時的誤差逆伝播法（truncated backpropagation through time）を検討する。

8.5.1 勾配消失と勾配爆発

T ステップにわたって展開され、リカレントパラメータを共有し、系列全体の平均損失を $L = \frac{1}{T}\sum_{t=1}^{T} l(\mathbf{y}_t, \mathbf{o}_t)$ とする RNN を考えよう。誤差逆伝播法は、重み行列が現れるすべてのタイムステップからの局所的な勾配の寄与を足し合わせることで、重み共有（weight sharing）を扱う。逆伝播（backward pass）の構造を見通しよく確認するため、8.4 節の RNN を取り上げ、バイアスと tanh 非線形性を一時的に省略する（tanh は直後に復活させる）。b 個の系列からなるミニバッチと各ステップの入力 $\mathbf{X}_t \in \mathbb{R}^{b \times d}$ に対し、線形 RNN は次式を計算する。

$$\begin{aligned}\mathbf{H}_t &= \mathbf{X}_t \mathbf{W}_{\text{xh}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{hh}},\\ \mathbf{O}_t &= \mathbf{H}_t \mathbf{W}_{\text{hq}},\end{aligned} \tag{8.20}$$

ここで、$\mathbf{W}_{\text{xh}} \in \mathbb{R}^{d \times h}$、$\mathbf{W}_{\text{hh}} \in \mathbb{R}^{h \times h}$、$\mathbf{W}_{\text{hq}} \in \mathbb{R}^{h \times q}$ である。Figure 8.10 は、このネットワークを展開した 3 つのタイムステップを表しており、順伝播（forward pass）を灰色で、逆伝播を赤色で示している。

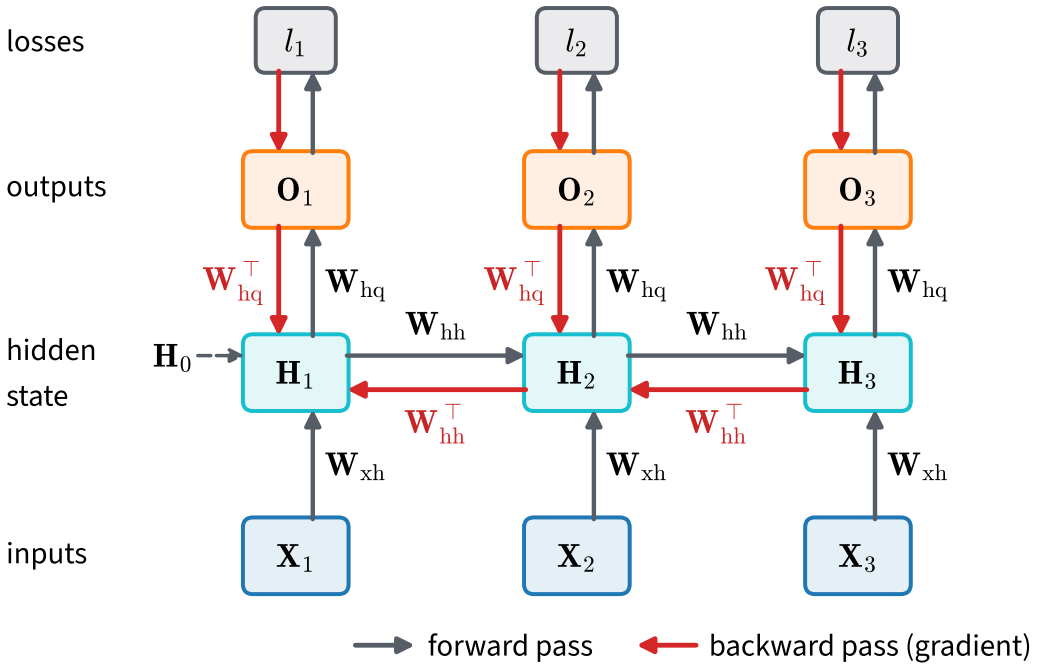


Figure 8.10: 3 つのタイムステップにわたって展開された線形 RNN の順伝播（灰色）と逆伝播（赤色）。最後のステップを除く各隠れ状態に到達する勾配は、2 本の赤い矢印の和となる。1 つは自身の出力から $\mathbf{W}_{\text{hq}}^\top$ を経る経路であり、もう 1 つは次の状態から $\mathbf{W}_{\text{hh}}^\top$ を経る経路である。

$\mathbf{G}_t = \partial L / \partial \mathbf{H}_t$ を、$\mathbf{H}_t$ に関する総損失の勾配とする。$\mathbf{H}_t$ はその時点の出力 $\mathbf{O}_t$ と次の隠れ状態 $\mathbf{H}_{t+1}$ の両方に入力され、さらに右からの行列積を通じた逆伝播では転置した重み行列が右から乗じられるため（4.3 節）、$\mathbf{G}_t$ は次の逆方向の漸化式に従う。

$$\begin{aligned}\mathbf{G}_t &= \frac{\partial L}{\partial \mathbf{O}_t} \mathbf{W}_{\text{hq}}^\top + \mathbf{G}_{t+1} \mathbf{W}_{\text{hh}}^\top,\\ \mathbf{G}_{T+1} &= \mathbf{0},\end{aligned} \tag{8.21}$$

これは Figure 8.10 において各隠れ状態に入ってくる 2 本の赤い矢印に対応する。Equation 8.21 を $\mathbf{G}_T$ から $\mathbf{G}_t$ まで逆方向に展開すると、次式が得られる。

$$\mathbf{G}_t = \sum_{j=t}^{T} \frac{\partial L}{\partial \mathbf{O}_j} \mathbf{W}_{\text{hq}}^\top \left(\mathbf{W}_{\text{hh}}^\top\right)^{j-t}. \tag{8.22}$$

したがって、未来のステップ j における損失は、$k = j - t$ ステップ前の状態へ、k^{th} 乗された $\mathbf{W}_{\text{hh}}^\top$ を通じて逆方向に伝わる（一般の非線形遷移 $\mathbf{H}_t = g(\mathbf{H}_{t-1}, \mathbf{X}_t, \mathbf{W}_{\text{hh}})$ の場合、この行列のべき乗は $j - t$ 個の中間状態ヤコビ行列の積 $\prod_{i=t+1}^{j} \partial \mathbf{H}_i / \partial \mathbf{H}_{i-1}$ に置き換わる）。1 回の逆方向スイープで $\mathbf{G}_1, \ldots, \mathbf{G}_T$ を計算してしまえば、パラメータ勾配はステップごとに 1 つの外積（outer product）を足し合わせるだけで求まる。

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}_{\text{hh}}} &= \sum_{t=1}^{T} \mathbf{H}_{t-1}^\top \mathbf{G}_t, \\ \frac{\partial L}{\partial \mathbf{W}_{\text{xh}}} &= \sum_{t=1}^{T} \mathbf{X}_t^\top \mathbf{G}_t.\end{aligned} \tag{8.23}$$

1.3.5 節および 4.4 節で見たように、行列を k^{th} 乗すると、各固有ベクトルは λ^k 倍にスケールされる。ρ を最大の $|\lambda|$、すなわち $\mathbf{W}_{\text{hh}}$ の**スペクトル半径**（spectral radius）とすると、$(\mathbf{W}_{\text{hh}}^\top)^k$ のノルムは漸近的に ρ^k のオーダーでスケールする。$\rho < 1$ のとき遠いステップからの寄与は指数関数的に消失し、$\rho > 1$ のときは指数関数的に爆発する。ここで tanh 活性化関数を復活させると、各ステップの逆伝播勾配には、局所的な導関数 $1 - \mathbf{H}_t^2 \in (0, 1]$ が要素ごとに乗じられてから $\mathbf{W}_{\text{hh}}^\top$ が掛けられる。$1 - h^2 \leq 1$ であるため、tanh が勾配を増幅させることは決してなく、隠れユニットが ± 1 付近で飽和すると勾配消失へと強く引き寄せる。ただし、$\mathbf{W}_{\text{hh}}$ が大きく、かつ活性化値が 0 付近にとどまる場合には、依然として勾配爆発が起こりうる。

8.4.2 節で訓練した言語モデルの内部で実際に何が起きているかを測定してみよう。まず、32 トークンのウィンドウで `TimeMachine` データセットを読み込む。

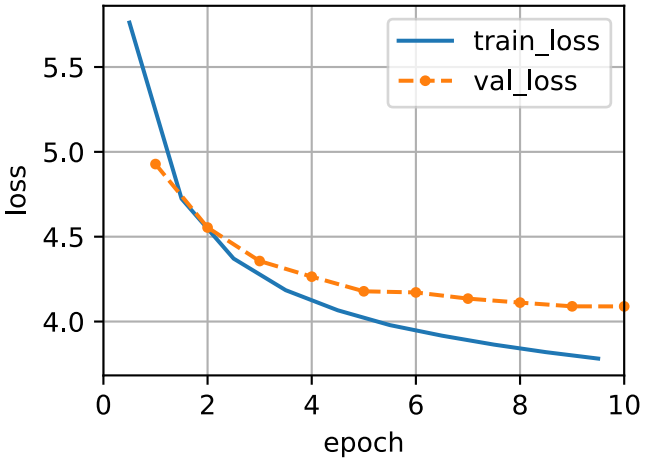
```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
```

次に、ゼロから実装したリカレント言語モデル `RNNLMScratch` を、埋め込み次元（embedding dimension）を $d = 512$、隠れ状態のサイズを $h = 128$ とし、ドロップアウトを加えてインスタンス化する。

```
rnn = d2l.RNNScratch(num_inputs=512, num_hiddens=128)
model = d2l.RNNLMScratch(rnn, vocab_size=len(data.vocab), lr=4, dropout=0.5,
                         emb_dropout=0.1)
```

`model` を 10 エポックにわたり、学習率 `lr=4`、大域的勾配クリッピング `gradient_clip_val=1` の設定で訓練する。

```
trainer = d2l.Trainer(max_epochs=10, gradient_clip_val=1, num_gpus=1)
trainer.fit(model, data)
```



より長いホライズンにわたる勾配フロー（gradient flow）を調べるため、検証データからランダムに 256 個のセグメント `seqs`（長さ $L = 64$、訓練ウィンドウの 2 倍）と、それぞれの直後のトークン `targets` をサンプリングする。

```
L = 64
val = d2l.numpy(data.X[data.num_train:data.num_train + data.num_val, 0])
starts = np.random.default_rng().choice(len(val) - L, 256, replace=False)
seqs = np.stack([val[s:s + L] for s in starts])  # (256, L) token ids
targets = val[starts + L]                        # The token after each stretch
```

ここで `model.rnn` を `seqs` にわたって順伝播させ、全 64 ステップ分の隠れ状態を `H` に収集するとともに、最終ステップの `logits`、および重み行列 `W_hh` と `W_hq` を取得する。

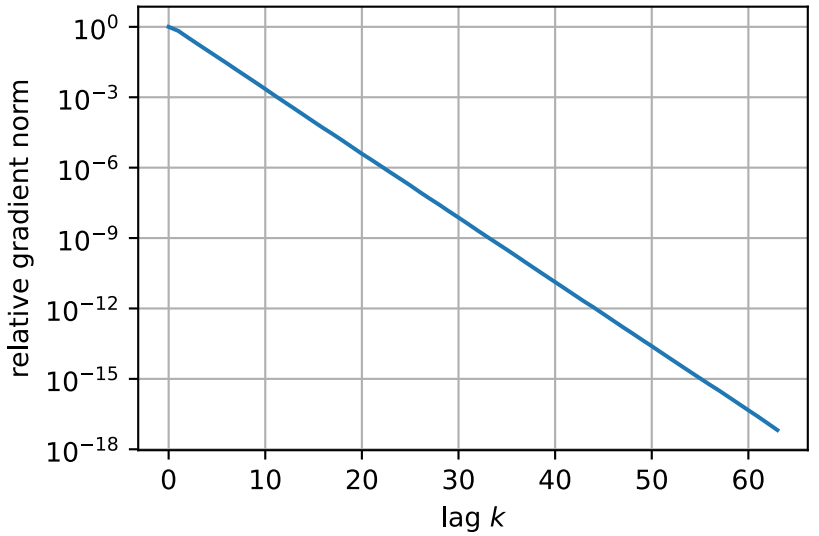
```
states, _ = model.rnn(model.embedding(jnp.array(seqs)))
logits = model.output_layer(states[-1:])[:, 0]  # The last step only
H, logits = d2l.numpy(jnp.stack(states, 0)), d2l.numpy(logits)
W_hh, W_hq = d2l.numpy(model.rnn.W_hh[...]), d2l.numpy(model.W_hq[...])
```

最終ステップの損失勾配 $\partial L_L/\partial \mathbf{O}_L = \mathrm{softmax}(\mathbf{O}_L) - \mathbf{Y}_L$ から出発し、$\mathbf{G}_{t-1} = (\mathbf{G}_t \odot (1 - \mathbf{H}_t^2))\mathbf{W}_{\mathrm{hh}}^\top$ によって時間を逆方向にたどり、相対勾配ノルム $\|\mathbf{G}_{L-k}\|/\|\mathbf{G}_L\|$ の幾何平均をラグ k の関数としてプロットする。

```
P = np.exp(logits - logits.max(1, keepdims=True))
P /= P.sum(1, keepdims=True)
P[np.arange(len(targets)), targets] -= 1  # dL/dO: softmax minus one-hot
G = (P @ W_hq.T).astype(np.float64)       # dL/dH at the last step
norms = []
for t in range(L - 1, -1, -1):            # Backward through time
    norms.append(np.linalg.norm(G, axis=1))
    G = (G * (1 - H[t] ** 2)) @ W_hh.T    # Through the tanh, then W_hh
rel = np.exp(np.log(np.array(norms) / norms[0]).mean(1))
print(f'spectral radius of W_hh: {np.abs(np.linalg.eigvals(W_hh)).max():.2f}; '
      f'mean tanh derivative: {(1 - H ** 2).mean():.2f}')
```

```
print('relative gradient norm at lags 1, 2, 4, 8, 16, 32:',
      [float(f'{rel[k]:.2g}') for k in (1, 2, 4, 8, 16, 32)])
d2l.plot(np.arange(L), rel, 'lag $k$', 'relative gradient norm',
         yscale='log', figsize=(4.5, 3))
```

```
spectral radius of W_hh: 1.79; mean tanh derivative: 0.26
relative gradient norm at lags 1, 2, 4, 8, 16, 32: [0.67, 0.35, 0.1, 0.008, 4.9e-05, 2.1e-09]
```



訓練後の $\mathbf{W}_{\text{hh}}$ のスペクトル半径は 1.8 付近であり、純粋な線形漸化式であれば爆発する値にもかかわらず、実際の勾配は急速に消失する。およそ 8 ステップ以内に初期ノルムの 1% 未満へと落ち込み、32 トークンにわたると 10 億分の数程度にまで縮小するのである。逆方向の 1 ステップごとに保たれる勾配ノルムは半分強であり（ラグ 8 での出力値から $0.008^{1/8} \approx 0.55$ となる）、大まかな見積もり $1.8 \times 0.27 \approx 0.5$（スペクトル半径に平均の tanh 導関数を掛けた値）に近い。この減衰は、位置ごとの検証損失（validation loss）が 2〜3 トークンのコンテキストで頭打ちになるという 8.4.2 節の結果と整合するが、勾配消失がその原因であることの証明にはならない。ラグ 2 と 4 では勾配はまだ元の約 3 分の 1 および約 10 分の 1 の大きさを保っており、この程度の減衰であれば SGD は問題なく学習できるうえ、8.4.2 節の Kneser–Ney カウントも勾配を一切使わずに同じコンテキスト長（context length）で飽和しているからである。ただし、データにどのような情報が含まれていようとも、はるか前のトークンからの学習がこの減衰によって阻まれることは確かである。

8.5.1.1 勾配爆発と勾配消失への対策

勾配爆発と勾配消失には、根本的に異なる対策が求められる。勾配爆発はたった 1 回の過大な更新でパラメータを破壊しかねないため、大域的勾配クリッピング（8.4 節）が簡便な安全策となる。しかし、勾配の和 Equation 8.22 全体にスカラーを掛けても、遠いラグと近いラグの**相対的な**重みは決して変わらない。したがって勾配消失を克服するには、漸化式そのものを変更して記憶方向に沿った状態ヤコビ行列を単位行列付近に保つ必要がある。その方法には、学習されたゲートで制御される加法的な記憶経路を作る**ゲート付き**リカレントセル（LSTM や GRU など）を用いるアプローチと、状態の非線形性を取り除いて固有値を単位円付近に制約する**構造化線形漸化式**（structured linear recurrences）および状態空間モデル（state space models）を用いるアプローチの 2 系統があり、どちらも第 12 章で詳しく扱う。

8.5.2 打ち切り型通時的誤差逆伝播法

T トークンからなるコーパス全体にわたって Equation 8.21 を評価するには、順伝播における全 T 個の隠れ状態をメモリに保持し、1 回のパラメータ更新のために全 T ステップを走査する必要がある。**打ち切り型通時的誤差逆伝播法**は、長いストリームを長さ T_0 の固定長セグメントへとスライスし、各セグメントの境界で誤差逆伝播法を打ち切ることで、1 回の更新あたりのメモリ量と計算量の両方に上限を設ける（Figure 8.11）。

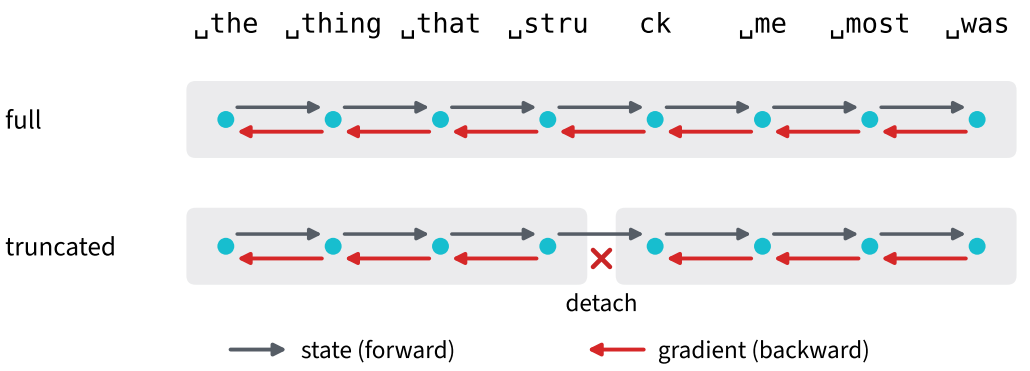


Figure 8.11: トークン系列に対する完全な通時的誤差逆伝播法と打ち切り型通時的誤差逆伝播法。どちらも同じ順伝播（灰色）を実行する。完全な誤差逆伝播法は系列全体に沿って勾配（赤色）を逆方向へ送る。打ち切りを行う場合は系列を 4 トークンのセグメントに分割し、各境界で状態を計算グラフから切り離す。その値は次のセグメントへと引き継がれるが、勾配が境界を越えて逆方向に流れることはない。

各ウィンドウで $\mathbf{H}_0 = \mathbf{0}$ にリセットする場合（8.4.2 節）でも、前のセグメントの最終状態を計算グラフから**切り離し**（detach）ながら順方向に引き継ぐ場合（Figure 8.11）でも、勾配がセグメントの境界を越えることは決してなく、高々 T_0 ステップという厳格な学習ホライズンが課される (Jaeger 2002)。しかし素の RNN では、32 トークンのウィンドウのずっと手前であるおよそ 8 ステップ後には勾配消失によって逆方向の信号がすでに途絶えてしまうため、ウィンドウを長くするだけではモデルの有効な記憶を延ばすことはできない。

8.5.3 まとめ

通時的誤差逆伝播法は、展開された RNN に通常の誤差逆伝播法を適用し、Equation 8.21 によって損失の感度を逆方向に伝播させるとともに、タイムステップ全体でパラメータ勾配を足し合わせる（Equation 8.23）。k ステップ逆方向に進む勾配には k 個の状態ヤコビ行列が乗じられ、線形 RNN ではこれが $(\mathbf{W}_{\text{hh}}^{\top})^k \sim \rho^k$ に帰着するため、勾配は $\rho > 1$ のとき爆発し、$\rho < 1$ のとき消失する。tanh を含めると各ステップで $1 - \mathbf{H}_t^2 \in (0, 1]$ が乗じられるため、訓練した RNN では $\rho \approx 1.8$ であるにもかかわらず、勾配はおよそ 8 ステップ以内に 1% 未満まで低下した。勾配クリッピングは急激な勾配爆発を抑え込み、打ち切り型 BPTT は 1 回の更新あたりのメモリ量と計算量を制限するが、長いホライズンにわたって勾配を保持するにはアーキテクチャ上の革新（第 12 章）が不可欠である。

8.5.4 演習問題

1. **手計算による逆伝播。** `RNNScratch`（8.4.2 節）の 1 ステップは次式を計算する。

$$\mathbf{H}_t = \tanh(\mathbf{X}_t \mathbf{W}_{\text{xh}} + \mathbf{H}_{t-1} \mathbf{W}_{\text{hh}} + \mathbf{b}_{\text{h}}).$$

   $\mathbf{G}_t = \partial L / \partial \mathbf{H}_t$ を、後続のステップおよびステップ t の出力から到達する勾配とする。

   1. このステップから $\partial L / \partial \mathbf{W}_{\text{xh}}$、$\partial L / \partial \mathbf{W}_{\text{hh}}$、$\partial L / \partial \mathbf{b}_{\text{h}}$、および $\partial L / \partial \mathbf{H}_{t-1}$ への寄与を、$\mathbf{G}_t$、$\mathbf{X}_t$、$\mathbf{H}_t$、$\mathbf{H}_{t-1}$、$\mathbf{W}_{\text{hh}}$ の関数として導出せよ。
   2. ステップ $t-1$ における出力からの寄与を $\partial L / \partial \mathbf{H}_{t-1}$ への寄与に加えると、$\mathbf{G}_{t-1}$ の漸化式が得られる。tanh とバイアスを取り除いたときにこれが Equation 8.21 に帰着することを示し、さらに tanh が各ステップで付け加える因子を答えよ。また、ウィンドウの T ステップにわたってパラメータ勾配がどのように累積するかを説明し、Equation 8.23 と比較せよ。

   CMU 11-785 の Homework 3 Part 1 を改変。

2. **べき乗法とアライメント。** $\mathbf{M} \in \mathbb{R}^{h \times h}$ を対称行列とし、その固有値 λ_i と固有ベクトル $\mathbf{v}_i$（$i = 1, \ldots, h$）が $|\lambda_i| \geq |\lambda_{i+1}|$ を満たすように並べられているとする。

   1. $\mathbf{M}^k$ が同じ固有ベクトルに対して固有値 λ_i^k を持つことを示せ。
   2. 標準正規分布に従う独立な成分を持つ確率ベクトル $\mathbf{x} \in \mathbb{R}^h$ に対し、$\mathbf{M}^k \mathbf{x}$ は $\mathbf{v}_1$ の方向へ k の増加とともに揃っていく（アライメントする）ことを示せ。これに必要な固有値の条件と、どのような意味でアライメントが成り立つかを述べよ。
   3. この結果は、RNN の勾配 Equation 8.22 に対して、ノルムの変化以外に何を意味するか。

3. [code] **コードによる逆伝播。**●`d2l.RNNLMScratch` を `d2l.TimeMachine` 上で 8.4.2 節と同様に訓練せよ。演習問題 1 で求めた 1 ステップの勾配と漸化式を用い、ロジットに関する損失の勾配（3.1.2.2 節）から出発して、`RNNScratch` セルのウィンドウ全体にわたる逆伝播をテンソル演算のみで実装せよ。それを 1 つの訓練ミニバッチ上で実行し、得られた `W_xh`、`W_hh`、`b_h` の勾配を自動微分によって計算された勾配と比較せよ。

   CMU 11-785 の Homework 3 Part 1 を改変。

4. [code] **非線形ヤコビ行列。**●ランダムな 100×100 の対称行列を生成してスペクトル半径 $\rho \in \{0.9, 1.0, 1.1\}$ にリスケールし、その k^{th} 乗のノルム $\|\mathbf{J}^k\|$ を k に対して片対数軸でプロットせよ。固有値から予測されるとおり、3 本の直線が得られるはずである。ここから対称かつ線形な場合を離れる。

   1. 対称行列を、各スペクトル半径にリスケールした一般の非対称行列 $\mathbf{W}_{\text{hh}}$ に置き換え、$\|\mathbf{J}^k\|$ を k に対してプロットせよ。$\|\mathbf{J}^k\|$ の $\rho = 0.9$ における最大値と、それが現れるラグを報告せよ。また、比 $\|\mathbf{J}^k\| / \rho^k$ がなぜ 1 を超えうるのか、そして k が大きくなっても有界にとどまるのはなぜか説明せよ（ランダム行列は確率 1 で対角化可能である）。
   2. 対称な $\mathbf{W}_{\text{hh}}$ に戻り、次の tanh RNN を実行せよ。

$$\mathbf{H}_t = \tanh(\mathbf{H}_{t-1} \mathbf{W}_{\text{hh}} + \mathbf{X}_t), \qquad \mathbf{H}_0 = \mathbf{0}, \tag{8.24}$$

      ただし 1 つの系列（$\mathbf{H}_t \in \mathbb{R}^{1 \times h}$）とし、入力 $\mathbf{X}_t$ の各成分は $\mathcal{N}(0, s^2)$ から独立に抽出するものとする。軌跡に沿ったヤコビ行列 $\partial \mathbf{H}_j / \partial \mathbf{H}_{j-1}$ の積のノルムを、$s = 0.1$ と $s = 1$ についてプロットせよ。非線形性と入力スケールによってノルムの増大はどのように変化するか。また、それはなぜか。
   3. $\mathbf{W}_{\text{hh}}$ の最大特異値が 1 未満であることは積が縮小するための十分条件であるのに対し、最大特異値が 1 より大きいことは積が増大するための必要条件ではあっても十分

条件ではない理由を説明せよ (Bengio et al. 1994; Pascanu et al. 2013)。この議論を、得られた tanh の曲線（$\rho = 0.9$ および $\rho = 1.1$）と関連づけよ。

5. [code] **ウィンドウ長と記憶ホライズン**。●8.5.1 節のモデルでは、最後の損失の勾配はおよそ 8 ステップ以内に元の大きさの 1% 未満まで低下し、これは 32 トークンの訓練ウィンドウの十分内側に収まっている。
   1. コードを実行する前に考えよ。このモデルを `num_steps=128` で再訓練した場合、このホライズンは移動すると予測されるか。Equation 8.22 における状態ヤコビ行列の積に基づいて論じよ。
   2. `num_steps` を 128 に設定して再訓練し、検証データの 256 トークンの区間上でラグに対する勾配を測定して、結果を自身の予測と比較せよ。
   3. 再訓練したモデルに対して、8.4.2 節の位置ごとの損失の測定を再現せよ。コンテキストの 3 番目のトークンを超えても損失は改善し続けるようになるか。

8.6 デコーディングと生成

本章ではここまで、次のトークンにどれだけ適切に確率を割り当てられるかという基準だけで言語モデルを評価してきた。しかし、テキストを**生成**するには、予測された分布から各ステップで具体的なトークンを 1 つ選択する必要もある。この選択は予想以上に重要である。というのも、選択規則が異なれば、まったく同じモデルからでも劇的に異なる文章が生成されるからである。8.4.2 節では、最も単純な規則がうまく機能しないことを見た。各ステップで最も確率の高い単一のトークンを貪欲に選び続けると、生成器はたちまち反復ループに陥ってしまう。では、次トークンの分布をどのようにすれば流暢なテキストへと変換できるのだろうか。

次トークン分布の系列を出力トークンへと変換するアルゴリズムは**デコーディング**と呼ばれる (Graves 2013; Holtzman et al. 2020)。デコーディング戦略は大きく 2 つの系統に分けられる。**最大化**（maximization）アルゴリズムはモデルの下で高い同時確率を持つ出力系列を探索するのに対し、**サンプリング**アルゴリズムは分布の信頼できない裾（tail）を変形または刈り込んだうえで各トークンを確率的に抽出する。以下では、この両系統を順に構築し比較していく。重要な点として、どのデコーディング規則も次トークン分布のみを介して言語モデルとやり取りするため、ここで構築するアルゴリズムは RNN、状態空間モデル、Transformer のいずれであっても、任意の自己回帰言語モデルにそのまま適用できる。

8.6.1 デコーディング問題

訓練済みの言語モデルは、次トークンの条件付き分布を与える。任意のプレフィックス $x_{<t} = x_1, \dots, x_{t-1}$ が与えられると $P(x_t \mid x_{<t})$ を出力し、確率の連鎖律（8.1 節）により、これらの条件付き確率から任意の完全な系列の同時確率が $P(x_1, \dots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$ と定まる。

デコーディングはその逆問題を突きつける。すなわち、これらの条件付き分布が与えられたとき、どの系列を出力すべきかという問題である。一見すると、**最も確率の高い**系列を見つけることが自明な目標に思えるが、厳密な最大化は計算量的にまったく実行不可能である。語彙 $\mathcal{V}$ とホライズン T に対して、可能な続きは $|\mathcal{V}|^T$ 通り存在する。本節で用いる 1,027 トークンの語彙と 50 トークンの続きでは $1027^{50} > 10^{150}$ 通りの候補となり、到底列挙できる数ではない。したがって、実用的なデコーダはいずれも探索木のごく小さなスライスのみを探索し、各ステップでどのプレフィックスを残すかを決定しなければならない。

仮に厳密な探索が可能であったとしても、確率が最大の単一の系列は、必ずしも求めているものとは**限らない**ことが多い。音声の文字起こしや技術文の翻訳のように、正解となる出力の幅が狭いタスクでは、高確率の系列を求めるアプローチがうまく機能する。対照的に、対話や創作執筆

のように多くの続きが同等に妥当である場合、よく較正されたモデルは多数の生じうる軌跡に確率を分散させるため、個々の続きが持つ確率質量はどうしても微小になる。モデルに最も確率の高い単一の経路を強制的に選ばせると、その自然な多様性が損なわれてしまうが、サンプリングは学習された分布から直接抽出することで多様性を保つ。

Figure 8.12 は、この対比に沿ってデコーディング戦略を整理したものである。最大化手法である貪欲デコーディング（greedy decoding）とビームサーチは、妥当な出力が限られる制約付きタスクにおける標準的な選択肢であり続ける一方、温度スケーリング（temperature scaling）や打ち切り（truncation）を組み合わせたサンプリング手法は、オープンエンドな生成におけるデフォルトとなっている。

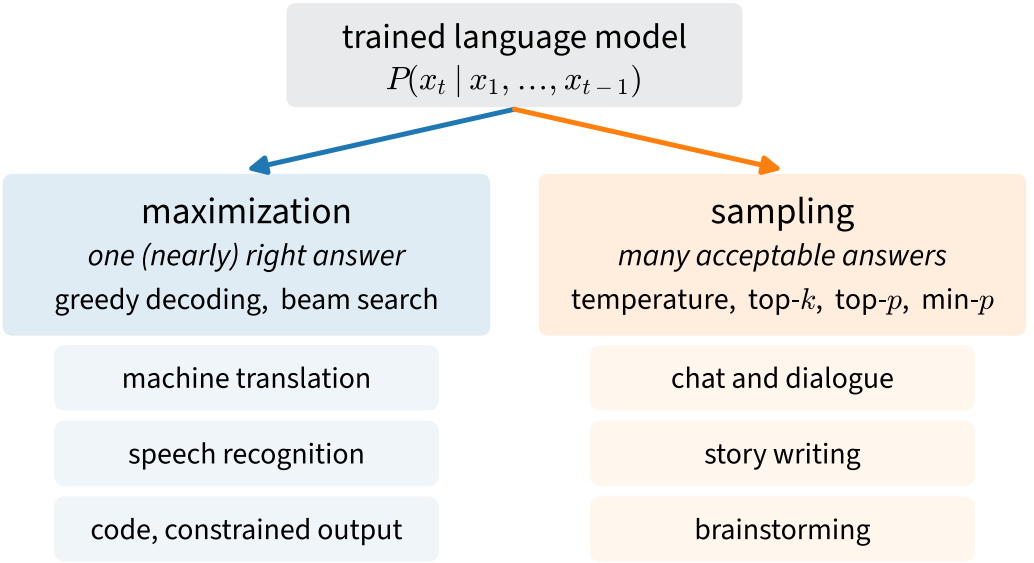


Figure 8.12: デコーディング規則の 2 つの系統。高確率の系列を探す手法は許容される出力が少ないタスクに適しており、サンプリングはオープンエンドな生成において多様性を保つ。

8.6.1.1 デコード対象のモデル

実験には、8.4.6 節で構築した簡潔な実装の RNN 言語モデルを用いる。このモデルは 512 次元の埋め込み、128 個の隠れユニット、埋め込みと隠れ状態の両方に対するドロップアウト、および『The Time Machine』の 1,027 個の BPE トークンに対する出力層を備えている。まず、`d2l.TimeMachine` から長さ 32 トークンの訓練ウィンドウと検証ウィンドウを読み込む。

```
data = d2l.TimeMachine(batch_size=1024, num_steps=32)
```

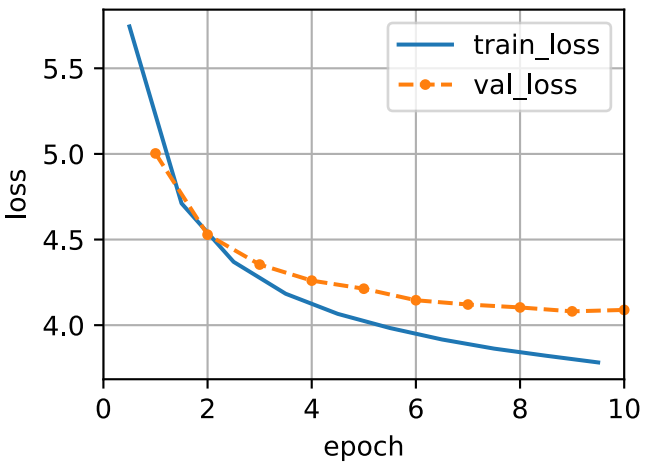
次に、簡潔な実装の再帰セル `d2l.RNN` と言語モデル `d2l.RNNLM` を、$d = 512$、$h = 128$、ドロップアウト、学習率 `lr=4` の設定で構築する。

```
rnn = d2l.RNN(num_inputs=512, num_hiddens=128)
model = d2l.RNNLM(rnn, vocab_size=len(data.vocab), lr=4, dropout=0.5,
```

```
                    emb_dropout=0.1)
```

model を gradient_clip_val=1 の勾配クリッピング付きで 10 エポック訓練すると、『The Time Machine』における検証パープレキシティは約 60 となる。

```
trainer = d2l.Trainer(max_epochs=10, gradient_clip_val=1, num_gpus=1)
trainer.fit(model, data)
print(f'validation perplexity {math.exp(trainer.history["val_loss"][-1]):.1f}')
```




```
validation perplexity 59.7
```

本節のすべてのデコーディングアルゴリズムは、モデルに依存しない単一のインターフェース step_fn(ids) を介して言語モデルにアクセスする。この関数は、それまでのプレフィックスのトークン ID を受け取り、次トークンのロジットからなる 1 次元 NumPy ベクトルを返す。通常の NumPy 配列を返すことで、4 つのフレームワークすべてに対して各デコーダを一度書くだけで済む。また、明示的な隠れ状態を省いているのは意図的な設計であり、リカレント型か Transformer かを問わず、あらゆる自己回帰モデルがこのシグネチャにそのまま適合する。

```
@nnx.jit
def forward(model, X):
    return model(X)

model_eval = nnx.view(model, deterministic=True, raise_if_not_found=False)

def seq_logits(ids):  # Token ids in, NumPy logits at every position out
    padded = ids + [0] * (-len(ids) % 32)  # Few distinct lengths to compile
    logits = forward(model_eval, d2l.tensor([padded]))  # Dropout off
    return d2l.numpy(logits)[0, :len(ids)]

def step_fn(ids):  # Token ids in, NumPy logits for the next token out
    return seq_logits(ids)[-1]
```

nnx.jit は入力形状ごとに順伝播を一度コンパイルするため、各プレフィックスを 32 トークンの倍数になるようゼロでパディングし、少数の形状だけをコンパイルするようにしている。ネット

ワークは系列を左から右へと読み込むため、パディングによって実際の末尾位置（読み取る対象の位置）のロジットが変化することはない。

そのトレードオフは効率性にある。1 回の呼び出しごとに、ロジットのベクトルを 1 つ取り出すためだけにプレフィックス**全体**にわたってネットワークを再評価してしまうのである。RNN ではこの余分な計算は回避可能であり、その計算コストについては本節の最後で改めて取り上げる。

8.6.2 貪欲デコーディング

考えられる限り最も単純なデコーダは、各ステップで条件付き確率が最も高い単一のトークンを確定させるものである。

$$x_t = \operatorname*{argmax}_{x \in \mathcal{V}} P(x \mid x_{<t}). \tag{8.25}$$

プレフィックスのスコアを計算し、argmax を選び、末尾に追加して繰り返す。貪欲デコーディングは生成トークンあたり 1 回のモデル評価しか必要とせず、完全に決定論的である。すなわち、同じプレフィックスからは常にまったく同じ続きが得られる。以下の補助関数 show は、トークン ID をデコードして引用符付きの文字列に変換し、改行を \n として表示するとともに、印字不可能なバイトを U+FFFD に置き換える。

```
def show(ids):  # Decoded text as a literal; control characters -> U+FFFD
    text = data.tokenizer.decode(ids)
    return repr(''.join(c if c.isprintable() or c == '\n' else '\ufffd'
                        for c in text))

def decode_greedy(text, num_tokens=50):
    ids = data.tokenizer.encode(text)
    for _ in range(num_tokens):
        ids.append(int(step_fn(ids).argmax()))
    return ids

for text in ('The Time Traveller', 'It seemed to me that'):
    d2l.print_wrapped(show(decode_greedy(text)))
```

```
'The Time Traveller. I was a vastly. I was a vastly. I was a vastly. I was a
    vastly. I was a vastly. I was a vastly. I was a vastly.'
"It seemed to me that the Time Traveller. 'I'I think I was a champ.
    The\nday. The\nday. The\nday. The\nday. The\nday. The\nday. The\nd"
```

8.4.2 節と同様に、どちらの続きもすぐに反復ループへと陥ってしまう。貪欲デコーディングには 2 つの構造的な弱点が内在している。1 つ目は**近視眼性**（myopia）である。局所的に最適な選択をつなぎ合わせても、大域的に最適な系列が得られるとは限らない。たとえば、モデルが最初のトークンに確率 $P(A) = 0.5$ と $P(B) = 0.4$ を割り当て、続く 2 ステップ目では A の後は不確実（最上位の続きの確率が 0.3）である一方、B の後は非常に確信度が高い（最上位の確率が 0.9）としよう。貪欲デコーディングは即座に A を確定させ、同時確率 $0.5 \times 0.3 = 0.15$ で終了してしまい、B から始まるはるかに確率の高い系列（同時確率 $0.4 \times 0.9 = 0.36$）を見逃してしまう（Figure 8.13）。

2 つ目の弱点は反復そのものである。貪欲デコーディングは各ステップで自身の決定論的な argmax を再帰構造にそのまま入力するため、次の隠れ状態は現在の状態の決定論的な関数となる。そのため、軌跡がいったん引き込みサイクル（attracting cycle）に落ち込むと、そこから

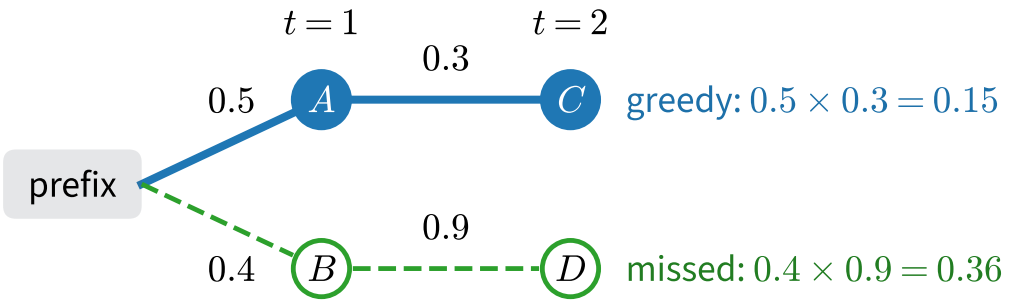


Figure 8.13: 2 ステップの例における貪欲デコーディングの近視眼性。貪欲デコーディングはより確率の高い最初のトークン A を選び、系列 AC（同時確率 $0.5 \times 0.3 = 0.15$）で終了する。一度も探索されない B を経由する経路は $0.4 \times 0.9 = 0.36$ に達する。

抜け出させるランダム性が存在しない。しかも、これは本節の小さな RNN 特有の癖ではない。Holtzman et al. (2020) は、1,000 倍規模の大きい言語モデルにおいてもまったく同じ**ニューラルテキスト退化**（neural text degeneration）が生じることを実証し、オープンエンドなテキストに対する尤度最大化が一貫して反復的で味気ない、人間の文章とは明らかに異なる散文を生み出すことを示した。

8.6.3 ビームサーチ

貪欲デコーディングが直ちに単一の経路を確定させ、全探索（exhaustive search）が可能なすべての経路を保持するのに対し、**ビームサーチ**は単一のハイパーパラメータである**ビーム幅**（beam width）w によって制御される中間的な手法をとる。ビームサーチは各ステップで、最もスコアの高い w 個のプレフィックスを残す。最も確率の高い w 個の最初のトークンから出発し、続く各ステップでは残っているすべてのプレフィックスを語彙全体にわたって展開し、$w \cdot |\mathcal{V}|$ 個の拡張候補すべてを累積対数確率でスコア付けしたうえで、上位 w 個の系列へと刈り込む。$w = 1$ とすれば貪欲デコーディングに一致し、候補が一切刈り込まれないほど w を大きくすれば全探索に一致する。

Figure 8.14 は、$w = 2$ とし、5 トークンの語彙 $\mathcal{V} = \{A, B, C, D, E\}$ 上で 3 ステップにわたって行うビームサーチを示している（共通のプロンプトは表記から省略している）。A と C が最も確率の高い上位 2 つの最初のトークンであるとしよう。ステップ 2 では、10 通りすべての拡張 $x' \in \mathcal{V}$ のスコアを次の式で計算する。

$$
\begin{aligned}
P(A, x') &= P(A)\, P(x' \mid A), \\
P(C, x') &= P(C)\, P(x' \mid C),
\end{aligned}
$$

そして上位 2 つ（たとえば AB と CE）を保持し、ステップ 3 でも同じ展開を繰り返して ABD と CED を残す。ビームサーチは $5^3 = 125$ 通りの系列すべてをスコア付けするのではなく、$5 + 10 + 10 = 25$ 通りのみを評価する。一般に、T トークンの生成にかかるコストは $O(w|\mathcal{V}|T)$ であり、貪欲デコーディングのわずか w 倍のコストで済み、全探索より指数関数的に少ない。

系列終了トークン（end-of-sequence token）を出力して系列を早期終了できる場合、異なる長さの候補を比較するには長さの補正が必要になる。各トークンの対数確率は非正であるため、生の累積対数確率は系列が長くなるにつれて単調に減少し、短い仮説が不当に有利になってしまう。

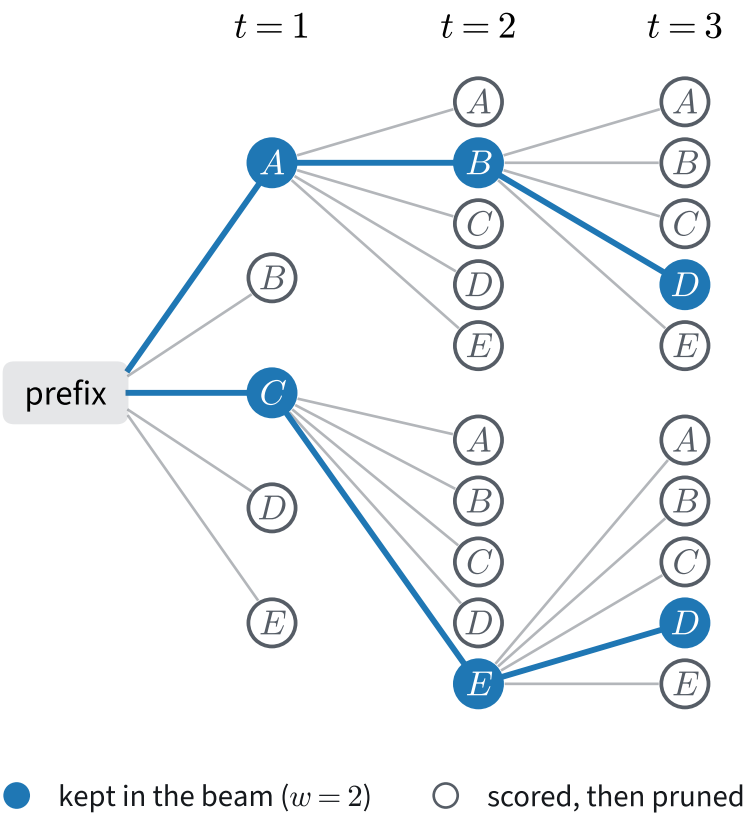


Figure 8.14: 5 トークンの語彙に対するビーム幅 2 のビームサーチ。各ステップで残っている各プレフィックスを語彙全体にわたって展開し、スコアが上位 2 つのプレフィックスのみを保持する。ここで残るのはまず A と C、次に AB と CE、続いて ABD と CED である。

標準的な対策は**長さ正規化**（length normalization）であり (Wu et al. 2016)、T トークンを長さ t_0 のプレフィックスに追加した候補を次の式でスコア付けする。

$$\frac{1}{T^{\beta}} \sum_{t=t_0+1}^{t_0+T} \log P(x_t \mid x_{<t}), \tag{8.26}$$

ここで、指数 β（通常は 0.75 前後）は、正規化なしの対数確率（$\beta = 0$）とトークンあたりの平均対数確率（$\beta = 1$）の間を補間する。以下のコードでは、`alpha` という引数名を Wu et al. (2016) に倣って用いる。

以下の `beam_search` の実装では、完了した仮説（`eos_id` で終わるもの）をそのまま次のステップへ引き継ぎ、アクティブなビームのみを展開する。なお、ここでは 1 つ簡潔な最適化を施している。全体の上位 w 個の候補には、任意の単一プレフィックスからの拡張は高々 w 個しか含まれないため、各アクティブビームからは上位 w 個の次トークンだけを取り出せばよく、$|\mathcal{V}|$ 個の拡張すべてをソートする必要はない。

```
def beam_search(step_fn, prefix, num_tokens, beam_size=4,
                alpha=0.75, eos_id=None):
    """Length-normalized beam search (score = log P / len^alpha).
    Returns candidates sorted best-first; [0][1] is the argmax sequence."""
    def log_softmax(logits):
        logits = [float(l) for l in logits]
        m = max(logits)
        lse = m + math.log(sum(math.exp(l - m) for l in logits))
        return [l - lse for l in logits]
    score = lambda logp, ids: logp / max(1, len(ids) - len(prefix))**alpha
    beams = [(0.0, list(prefix), False)]  # (log-prob, sequence, finished)
    for _ in range(num_tokens):
        if all(done for _, _, done in beams):
            break
        candidates = [b for b in beams if b[2]]  # Finished pass through
        for logp, ids, _ in (b for b in beams if not b[2]):
            logprobs = log_softmax(step_fn(ids))
            best = sorted(range(len(logprobs)),
                          key=lambda i: -logprobs[i])[:beam_size]
            candidates += [(logp + logprobs[i], ids + [i], i == eos_id)
                           for i in best]
        beams = sorted(candidates, key=lambda b: score(b[0], b[1]),
                       reverse=True)[:beam_size]
    return sorted([(score(logp, ids), ids) for logp, ids, _ in beams],
                  reverse=True)
```

同じ 2 つのプロンプトを $w = 1$（貪欲デコーディングの再現）と $w = 4$ でデコードし、最上位仮説の長さ正規化スコア Equation 8.26 を出力してみよう。このコーパスには系列終了トークンがないため、すべての候補が長さ 30 に達し、定数の分母 $30^{0.75} \approx 12.8$ によって相対的なランキングは変化しない。

```
for text in ('The Time Traveller', 'It seemed to me that'):
    prefix = data.tokenizer.encode(text)
    for w in (1, 4):
```

```
        score, ids = beam_search(step_fn, prefix, 30, beam_size=w)[0]
        d2l.print_wrapped(f'w={w}, score {score:.2f}: {show(ids)}')
```

```
w=1, score -5.46: 'The Time Traveller. I was a vastly. I was a vastly. I was
    a vastly. I was a vastly. I'
w=4, score -4.19: "The Time Traveller.\n\n'What is the slopes of the
    preason. I felt assured to me.\n\n'"
w=1, score -5.09: "It seemed to me that the Time Traveller. 'I'I think I was
    a champ. The\nday. The\nday. The\nd"
w=4, score -4.22: "It seemed to me that the Time Traveller.\n\n'What is the
    slopes of the preason. I felt assured to me."
```

ビームを広げると、一般には貪欲デコーディングよりも確率の高い続きが見つかる。ただし、近似的な刈り込みによって後から有望になるはずだったプレフィックスが捨てられることもあるため、確実とは限らない。しかも、確率が高いからといって自動的に**より良い**文章になるとは限らない。オープンエンドな生成では、大規模言語モデル（large language model、LLM）におけるビームサーチは人間の文章よりも顕著に反復の多いテキストを生成する (Holtzman et al. 2020)。この対比については後ほど定量化する。対照的に、音声認識や機械翻訳のような制約の強いタスクでは、適度な幅（通常 4〜10）のビームサーチが依然として標準的なデコーダである (Sutskever et al. 2014; Wu et al. 2016)。

8.6.4 サンプリング

モデルが続きに関する妥当な分布を学習できているなら、最大化に代わる最も自然な選択肢は直接的な**祖先サンプリング**（ancestral sampling）である。すなわち、$x_t \sim P(x_t \mid x_{<t})$ を抽出し、末尾に追加して繰り返す。サンプリングは最大化が抑え込んでしまう豊かな多様性を取り戻すが、生のサンプリングはそのまま分布の裾の問題に直面する。ソフトマックスは語彙のすべての項目（ここでは約 1,000 トークン、現代の LLM では 10 万トークン以上）に非ゼロの確率を割り当てるため、何千もの不自然なトークンにわたる微小な確率が合計されると、各ステップで無視できない大きさの確率質量になってしまう。その結果、純粋なサンプリングではまったく不自然なトークンが頻繁に抽出され、それをコンテキストとして再入力することで残りの文章全体が脱線しかねない。これはまさに 8.1 節で議論したロールアウト誤差の蓄積そのものである。Holtzman et al. (2020) はこれを**信頼できない裾**（unreliable tail）と呼んだ。言語モデルの確率は分布の上位付近ではよく較正されているが、裾の奥深くではノイズが多いのである。実用的な対策はサンプリングの前に分布を変形または打ち切ることであり、広く使われている 4 つの制御手法がそれぞれ異なる方法でこれを実現する。

温度スケーリングは、ソフトマックスの前にすべてのロジット o_x を正のスカラー $\tau > 0$ で割る手法である。

$$P_\tau(x \mid x_{<t}) = \frac{\exp(o_x/\tau)}{\sum_{x' \in \mathcal{V}} \exp(o_{x'}/\tau)} \propto P(x \mid x_{<t})^{1/\tau}. \tag{8.27}$$

τ を 1 より小さくすると分布は鋭くなり（$\tau \to 0$ で貪欲デコーディングに収束する）、逆に τ を 1 より大きくすると一様分布に近づくよう平坦化される。しかし、温度は分布全体を一様に再スケーリングするため、大雑把な調整手段にすぎない。信頼できない裾を抑え込めるほどソフトマックスを冷やすと、もっともらしい上位トークン間の正当な多様性まで同時に潰れてしまうからである。

Top-k サンプリングは順位によって打ち切りを行い、最も確率の高い上位 k 個のトークンのみを残して再正規化し、サンプリングする (Fan et al. 2018; Radford et al. 2019)。一方、**Top-p**（また

はニュークリアス）サンプリングは累積確率質量によって打ち切りを行い、確率の和が少なくとも p（一般には $p = 0.9$）に達する最小の上位トークン集合を保持する (Holtzman et al. 2020)。この違いが重要なのは、モデルのエントロピーがトークンごとに大きく変動するためである。次の単語がほぼ確実な場面では、固定された $k = 20$ は 19 個の不適切な候補を通過させてしまう一方、何十もの単語が同等に自然な場面では、$k = 20$ が本来の多様性を切り捨ててしまうおそれがある。Top-p は動的に適応し、鋭い分布では候補集合を縮小し、平坦な分布では拡大する。ただし、単一の最上位トークンだけで p を超える場合（たとえば確率が 0.95 であり、$p = 0.9$ の下で有力な 2 番手の候補を除外してしまう場合）には、依然として過度に貪欲な振る舞いをしうる。

Min-p サンプリングは、モデルの最上位確率に対する相対的な閾値を設定する手法であり (Nguyen et al. 2025)、最も確率の高い単一トークンの少なくとも p_{min} 倍（たとえば $p_{\text{min}} = 0.05$）の確率を持つすべてのトークンを保持する。モデルの確信度が高いときにはカットオフが自動的に上昇して有力な候補のみを通し、モデルが不確実なときにはカットオフが下がってより幅広い探索を許容する。Nguyen et al. (2025) は、絶対的な質量閾値では平坦化された裾の大部分まで拾い込んでしまうような高い温度においても、この相対閾値であれば top-p よりはるかに穏やかに性能を保てることを示している。

8.6.4.1 統一サンプラ

温度、top-k、top-p、min-p はいずれもまったく同じ次トークン分布に対して作用するため、コンパクトな単一の関数ですべての組合せを扱える。以下の `sample_next` は、`temperature == 0` のときに貪欲な argmax を返し、それ以外のときは Equation 8.27 によって P_τ を構成してトークンを確率の降順にソートし、有効な各カットオフが許容する上位トークン数を計算して最も厳しい上限を採用し、再正規化したうえで NumPy の乱数生成器 `rng` を用いてトークンを 1 つ抽出する。

```
def sample_next(logits, temperature=1.0, k=None, p=None, min_p=None,
                rng=None):
    """Draw a token id from 1-D NumPy logits; temperature 0 is greedy.
    Top-k, top-p, and min-p truncate the temperature-scaled distribution."""
    logits = np.asarray(logits, dtype=np.float64)
    if temperature == 0:
        return int(logits.argmax())
    probs = np.exp((logits - logits.max()) / temperature)
    order = np.argsort(-probs)               # token ids, most probable first
    probs = probs[order] / probs.sum()
    keep = len(probs)
    if k is not None:                        # top-k: a fixed count
        keep = min(keep, k)
    if p is not None:                        # top-p: smallest head of mass p
        keep = min(keep, int(np.searchsorted(probs.cumsum(), p)) + 1)
    if min_p is not None:                    # min-p: a fraction of the top
        keep = min(keep, int((probs >= min_p * probs[0]).sum()))
    rng = np.random.default_rng() if rng is None else rng
    return int(order[rng.choice(keep, p=probs[:keep] / probs[:keep].sum())])
```

ラッパー関数 `generate` は、自己回帰デコーディング（autoregressive decoding）のループをまとめたものである。`step_fn` を呼び出し、`sample_next` で次のトークンを抽出し（戦略に関するキーワード引数はそのまま渡す）、末尾に追加し、`eos_id` が出力されたら早期終了する。『The Time Machine』のオープンエンドな文章には系列終了トークンがないため、本節の例では固定個数のトークンを生成する。

```
def generate(step_fn, prefix, num_tokens, eos_id=None, **strategy):
    """Extend prefix by up to num_tokens ids, each chosen by sample_next
    from step_fn(ids), the next-token logits; stop early at eos_id."""
    ids = list(prefix)
    for _ in range(num_tokens):
        ids.append(sample_next(step_fn(ids), **strategy))
        if eos_id is not None and ids[-1] == eos_id:
            break
    return ids
```

確率が $0.45, 0.25, 0.15, 0.10$ および 0.05 である 5 トークンのトイ分布を使うと、各カットオフの動作を直接確認できる。top-k（$k = 2$）はトークン 0 と 1 のみを保持し、top-p（$p = 0.8$）はトークン 0、1、2（累積質量が 0.85 に達する）を保持し、min-p（$p_{\min} = 0.5$）は少なくとも $0.5 \times 0.45 = 0.225$ の確率を持つトークン、すなわち再びトークン 0 と 1 を保持する。各規則の下で 1,000 個のサンプルを抽出すると、その台（support）が正確に一致することが確認できる。

```
logits = np.log([0.45, 0.25, 0.15, 0.10, 0.05])
print('greedy:', sample_next(logits, temperature=0))
for cutoff in (dict(k=2), dict(p=0.8), dict(min_p=0.5)):
    drawn = {sample_next(logits, **cutoff) for _ in range(1000)}
    print(cutoff, 'tokens drawn:', sorted(drawn))
```

```
greedy: 0
{'k': 2} tokens drawn: [0, 1]
{'p': 0.8} tokens drawn: [0, 1, 2]
{'min_p': 0.5} tokens drawn: [0, 1]
```

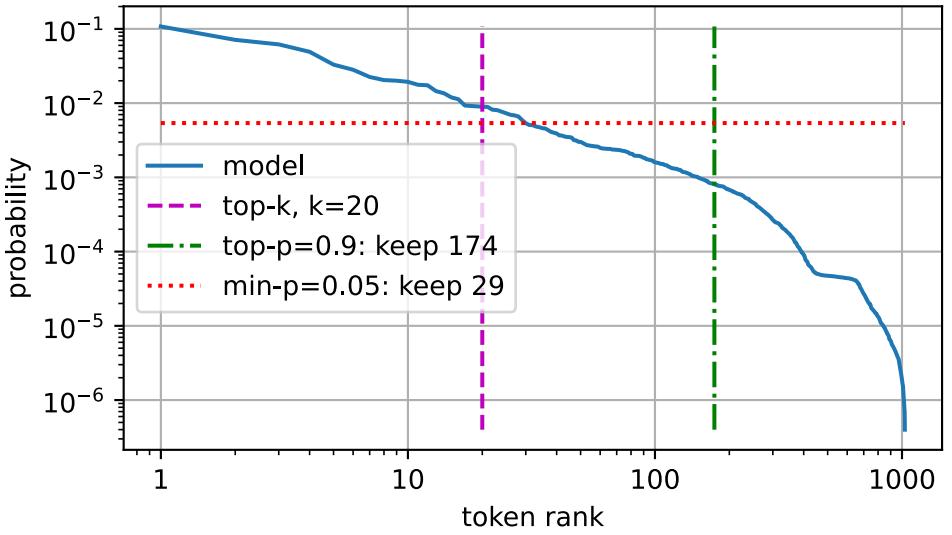
予想どおり、どの規則も残った台のみから厳密にサンプリングしている。すなわち、トークン 0 と 1 が top-k と min-p で抽出され、トークン 0〜2 が top-p で抽出されている。

8.6.4.2 訓練済みモデルにおけるカットオフ

これら 3 つの規則が実際の言語モデル上でどのように振る舞うかを確認するため、`"The Time Traveller"` に続く RNN の次トークン分布をソートして両対数軸（log-log axes）上にプロットしてみよう。このように表示すると、各打ち切り規則は明快な幾何学的解釈を持つ。top-k は曲線の形状にかかわらず順位 k に固定の垂直線を引き、top-p は確率質量 p が蓄積される順位に垂直線を引き、min-p は最上位確率の $p_{\min}$ 倍の高さに**水平**線を引き、モデルの確信度に応じて上下する。垂直カットオフより右側、あるいは水平カットオフより下側にあるトークンはすべて刈り込まれる。

```
def next_probs(text):  # Next-token probabilities after text, sorted
    logits = step_fn(data.tokenizer.encode(text))
    probs = np.exp(logits - logits.max())
    return np.sort(probs / probs.sum())[::-1]

def cutoffs(probs, p=0.9, min_p=0.05):  # Tokens kept by top-p and min-p
    return (int(np.searchsorted(probs.cumsum(), p)) + 1,
            int((probs >= min_p * probs[0]).sum()))

probs = next_probs('The Time Traveller')
n_p, n_m = cutoffs(probs)
```

```
lo, hi = probs[-1], probs[0]
d2l.plot([np.arange(1, len(probs) + 1), [20, 20], [n_p, n_p],
          [1, len(probs)]], [probs, [lo, hi], [lo, hi], [0.05 * hi] * 2],
         xlabel='token rank', ylabel='probability', xscale='log',
         yscale='log', figsize=(5.5, 3),
         legend=['model', 'top-k, k=20', f'top-p=0.9: keep {n_p}',
                 f'min-p=0.05: keep {n_m}'])
```



両対数軸上では、ソートされた確率は順位 1 における 0.1 から 0.25 の間から順位 100 付近の約 10^{-3} まで着実に低下した後、残りの数百トークンにわたってさらに急激に落ち込んでいる。これが打ち切りによって除去される広大な低確率の裾である。モデルの確信度に対して各規則がどれだけ適応的に反応するかを観察するため、`"The Time Traveller"` をより短いプレフィックス `"The Time"` と比較する。後者は小説中の全 13 回の出現のうち 11 回で直後に `" Traveller"` が続くため、鋭く尖った分布をもたらす。

```
for text in ('The Time Traveller', 'The Time'):
    probs = next_probs(text)
    n_p, n_m = cutoffs(probs)
    print(f'{text!r}: top probability {probs[0]:.2f}; kept: top-k 20, '
          f'top-p {n_p}, min-p {n_m}')
```

```
'The Time Traveller': top probability 0.11; kept: top-k 20, top-p 174, min-p 29
'The Time': top probability 0.80; kept: top-k 20, top-p 2, min-p 2
```

top-k がコンテキストにかかわらず機械的に 20 トークンを保持するのに対し、2 つの適応的な規則はモデルの確信度が高い場面で急激に絞り込まれる。最上位トークンが支配的な `"The Time"` の後ではどちらも積極的に刈り込みを行い、とりわけ min-p はごく少数の候補しか残さない。しかし、`"The Time Traveller"` の後のようにモデルが不確実な場面では、両者の振る舞いは大きく分かれる。top-p は確率質量の 90% を集めるために裾の奥深くまで踏み込む必要があり、その大半が個々の確率 1% 未満である 100 個を優に超えるトークンを通過させてしまう。一方、min-p は最上位候補のみを基準とするため、最上位確率の 20 分の 1 以内に収まる高々数十個のトークンしか保持しない。

8.6.4.3 単一プレフィックス上での戦略の比較

まず $\tau = 0.5$、1.0、2.0 における純粋な温度スケーリングから始めて、各戦略が実際に生成する続きを確認してみよう。

```
prefix = data.tokenizer.encode('The Time Traveller')
for tau in (0.5, 1.0, 2.0):
    out = generate(step_fn, prefix, 50, temperature=tau)
    d2l.print_wrapped(f'tau={tau}: {show(out)}')
```

```
tau=0.5: "The Time Traveller regued that the old faint\ndelight of the
    beautifulation. I suppose I had seen in the despient, I had don't know
    how I remember of the"
tau=1.0: 'The Time Traveller looked now, part is a\ntheena□ of the burned to
    going to myself. The sky of an,\nmeweight my necies. I tried rising dist
    or soistibly norbe, who'
tau=2.0: 'The Time Traveller a gheautite laroterllow For NayN Andest perhaps
    Cor none round my poper indungual toofround towardsigapf found Thereles
    creatical face, was at□ve oldmekes□'
```

$\tau = 0.5$ では、サンプラは高確率のトークン付近に保守的にとどまる。$\tau = 1.0$ では出力が目立って多様になる一方、時折崩れた単語が混じるようになる。そして $\tau = 2.0$ では、分布の平坦化によって裾のトークンが出力列を圧倒し、英語から逸脱するばかりか、有効な UTF-8 文字をなさないバイト列まで出力してしまう。次に、同じプロンプトに対して主要なデコーディング戦略をそれぞれの標準的な設定で直接比較してみよう。

```
strategies = {'greedy': dict(temperature=0), 'tau=1': dict(),
              'top-k=20': dict(k=20), 'top-p=0.9': dict(p=0.9),
              'min-p=0.05': dict(min_p=0.05)}
for name, strategy in strategies.items():
    out = generate(step_fn, prefix, 50, **strategy)
    d2l.print_wrapped(f'{name:>10}: {show(out)}')
```

```
    greedy: 'The Time Traveller. I was a vastly. I was a vastly. I was a
    vastly. I was a vastly. I was a vastly. I was a vastly. I was a vastly.'
     tau=1: 'The Time Traveller\ncove about the hail going. And the I had
    been nunningaryhinguntively. It seemed metmey of the tinught and leave
    me to overhas that put mees.'
  top-k=20: 'The Time Traveller I had the softs and the sphinx, and the
    fruit of his paintrieve, to the\nfeltighing\nof the pallications of the
    sight of that I will sh'
 top-p=0.9: "The Time Traveller remained it, and I was determist\nwemed of
    the stotst of the little people were a little tural?\nAnd for any
    kind.'\n\n'I sat, the machine was n"
min-p=0.05: 'The Time Traveller\nsume of a miles of the gradual story of the
    simply draat of the low. Thing had a next. The Editor came upon the ran,
    and the\nlighter'
```

当然ながら、1 編の中編小説で訓練された小さな単層 RNN では、いかなるデコーダを用いても完璧な文章は生成できず、規則ごとに 1 サンプルだけではノイズの多い証拠にすぎない。それでも定性的な対比は明白である。貪欲デコーディングは決定論的であり反復サイクルに陥るのに対し、どのサンプラも呼び出すたびにループのない新しいテキストを生成する。実務では、適度な温度と top-p または min-p による打ち切りを組み合わせることで流暢さと多様性のバランスがよくと

れるため、これがオープンエンドな生成のデフォルトとなっている。

8.6.5 評価とコスト

8.6.5.1 生成テキストの品質

個々の生成文を目視で確認する以外に、デコーディング戦略を体系的に比較するにはどうすればよいだろうか。パープレキシティはこの問いに答えられない。パープレキシティはホールドアウトされた人間のテキストに対する**モデル**の評価指標であって特定のデコーディング規則を評価するものではなく、生成されたサンプルを自身のモデル尤度でスコア付けすれば、サンプリングがピークの尤度と引き換えに意図的に多様性を確保しているにもかかわらず、同語反復的に最大化手法が勝者となってしまうからである。

それでも、自動診断指標を使えば特定の失敗モードを的確に捉えられる。反復を定量化するため、ここでは **distinct-3** を用いる。これは生成された続きにおける全 3-gram に対するユニークな 3-gram の比率であり、3 トークンの連続列が一度も繰り返されないときに 1 となり、テキストがサイクルに陥ると 0 に近づく。以下では、検証データからランダムに選んだ長さ 6 トークンのプレフィックス 20 個に対し、貪欲デコーディング、ビームサーチ ($w = 4$)、および各サンプラの下で 100 トークンの続きを生成し、モデルの下でのトークンあたり平均対数確率と平均 distinct-3 の両方を報告する。

```
def log_prob(ids, start):  # Mean log-probability of ids[start:] in one pass
    logits = seq_logits(ids[:-1])[start - 1:]
    lse = logits.max(1) + np.log(np.exp(logits - logits.max(1, keepdims=True)).sum(1))
    return float(np.mean(logits[np.arange(len(logits)), ids[start:]] - lse))

def distinct(ids, n=3):  # Fraction of distinct n-grams in ids
    grams = [tuple(ids[i:i + n]) for i in range(len(ids) - n + 1)]
    return len(set(grams)) / len(grams)

val = d2l.numpy(data.X[data.num_train:data.num_train + data.num_val])
rows = np.random.default_rng().choice(len(val), 20, replace=False)
prefixes = [[int(i) for i in val[r, :6]] for r in rows]
decoders = {
    'greedy': lambda pre: generate(step_fn, pre, 100, temperature=0),
    'beam, w=4': lambda pre: beam_search(step_fn, pre, 100, beam_size=4)[0][1],
    'tau=1': lambda pre: generate(step_fn, pre, 100),
    'top-k=20': lambda pre: generate(step_fn, pre, 100, k=20),
    'top-p=0.9': lambda pre: generate(step_fn, pre, 100, p=0.9),
    'min-p=0.05': lambda pre: generate(step_fn, pre, 100, min_p=0.05)}
for name, decode in decoders.items():
    outs = [decode(pre) for pre in prefixes]
    print(f'{name:>10}: log-prob per token '
          f'{np.mean([log_prob(o, 6) for o in outs]):.2f}, distinct-3 '
          f'{np.mean([distinct(o[6:]) for o in outs]):.2f}')
```

```
   greedy: log-prob per token -2.17, distinct-3 0.09
beam, w=4: log-prob per token -1.82, distinct-3 0.35
    tau=1: log-prob per token -4.05, distinct-3 1.00
 top-k=20: log-prob per token -2.82, distinct-3 0.99
top-p=0.9: log-prob per token -3.51, distinct-3 1.00
min-p=0.05: log-prob per token -2.80, distinct-3 0.99
```

これら 2 つの指標は、見事なまでに正反対の方向へ引っ張り合う。2 つの最大化戦略はいずれも任意のサンプラより顕著に高いトークンあたり対数確率を達成する（通常はビームサーチが貪欲デコーディングをわずかに上回る）が、3-gram の大半を繰り返すため distinct-3 のスコアは急落する。一方、4 つのサンプラはすべて 1 に近い distinct-3 を達成しており、Holtzman et al. (2020) が報告した尤度と多様性のトレードオフそのものをミニチュア規模で再現している。

サンプリング系統の内部では、4 つの規則すべてで多様性がほぼ飽和している一方、トークンあたり対数確率には明確な差が表れる。純粋な祖先サンプリング（$\tau = 1$）は平均対数確率が最も低く（トークンあたり約 -4 nat）、3 つの打ち切りサンプラはいずれも distinct-3 を損なうことなく尤度を大幅に引き上げる。先ほどのカットオフの保持数とも整合的に、最も広い台を残す top-p（0.9）が純粋なサンプリングに最も近く、より積極的に刈り込む top-k（20）と min-p（0.05）はトークンあたり対数確率を約 1 nat 押し上げており、裾の打ち切りが意図どおりに機能していることを直接示している。

distinct-3 のような自動指標は具体的な失敗モードを検出するうえで有用だが、一貫性、事実性、文体までは測れない。結局のところ、オープンエンドなテキスト生成は人間の読み手（あるいは較正された代理評価器）によって判断される一方、翻訳のように正解の参照文が存在する制約付きタスクは参照出力と照らし合わせて測定できる。要するに、パープレキシティは**モデル**の選択を導き、人間の選好（またはタスク固有のベンチマーク）が**デコーダ**の選択を導くのである。

8.6.5.2 生成のコスト

任意の自己回帰モデルをラップできるよう `step_fn` をステートレスに保ったため、新しいトークンを生成するたびにプレフィックス**全体**にわたって RNN を再実行することになり、T トークンの続きを生成するのに $O(T^2)$ 回のセル更新コストがかかる。また、自己回帰生成（autoregressive generation）は本質的に**逐次的**（sequential）でもある。すべての位置があらかじめ既知である教師強制（teacher forcing）による訓練とは異なり、トークン $t+1$ のスコアはトークン t が選ばれない限り計算できない。この逐次的な依存関係は本質的で避けられないものの、二次の再計算は RNN にとってはまったく不要である。隠れ状態がすでにプレフィックスを要約しているため、（`predict` が 8.4.2 節で行ったように）その状態をステップからステップへと引き継げば、系列長によらずトークンあたりちょうど 1 回のセル更新までコストを下げられる。再帰状態（recurrent state）を持たないモデルでは、過去の各トークンに対して計算した内容をキャッシュすることで同じ再計算を回避するが、その代わりプレフィックスの長さに比例して増大するメモリコストが生じる（第 10 章）。

生成トークンあたりの計算量とメモリが一定であることは、リカレント言語モデルを特徴づける効率上の利点である。しかし、まさにその固定サイズの状態がモデルのボトルネックにもなり、プレフィックスから記憶すべきあらゆる情報をわずか 128 個の浮動小数点数へと押し込めることになる（8.4.1.1 節）。次トークン分布が長距離の構造を捉えられていないモデルから、デコーディングの工夫だけで繊細な表現を引き出すことはできない。とはいえ、本節のデコーダはモデルを純粋なブラックボックスとして扱うため、ここで構築したすべてのアルゴリズムは、はるかに強力なモデルにもそのまま移行できる。

これをもって「**基礎**」パートは幕を閉じる。そしてその締めくくりは、まさにアーキテクチャ上の明確な境界線に位置している。本節の RNN は一定のコストで 1 トークンずつテキストを予測するが、実質的には直近 2〜3 トークンの履歴しか活用できておらず（8.4.2 節）、わずか数ステップで勾配が消失してしまうため（8.5 節）、それ以上多くの履歴を使えるよう学習できる見込みはほとんどない。ここからは 2 つの道が先へと続いている。1 つ目の道は、固定サイズの状態を保ちつつ再帰構造を再設計し、長いホライズンにわたって勾配が生き残るようにするアプローチである（第 12 章）。2 つ目の道は、固定サイズへの圧縮そのものを捨て去り、過去のすべてのトークンの表現を保持して、系列長に応じて増大するコストと引き換えにそれらへ直接注意を向けるアプローチ、すなわち現代の大規模言語モデルの中核をなす**アテンション**機構（第 10 章）である。

根底にあるアーキテクチャがいずれであっても、本章で学んだサブワードトークナイザ、バイトあたりビット数（bits-per-byte）による評価、そしてデコーディングアルゴリズムはそのまま引き継がれる。

8.6.6 まとめ

デコーディングは、言語モデルの次トークン分布を具体的なトークン系列へと変換する処理である。最大化手法は高確率の出力を求めるものであり、貪欲デコーディングは各ステップで局所的に確定させるため近視眼性と反復ループの両方に悩まされる一方、ビームサーチは w 倍のコストで上位 w 個の仮説を追跡し（候補が早期終了しうる場合は長さ正規化を用いる）、出力空間の狭い制約付きタスクで優れた性能を発揮する。オープンエンドな生成では、サンプリングが自然な多様性を保ち、温度スケーリング（Equation 8.27）と top-k、top-p、または min-p による裾の打ち切りによって低確率の裾のトークンがロールアウトを破綻させるのを防ぐことで、distinct-3 の多様性では純粋なサンプリングに匹敵しつつ、はるかに確率の高いテキストを生成できる。最後に、パープレキシティが評価するのはデコーダではなくモデルであり、再帰的な隠れ状態を引き継ぐことで素朴なステートレスデコーディングにおける $O(T^2)$ の再計算を回避できる。

8.6.7 演習問題

1. **ビームサーチの性質。**
    1. ビーム幅 2、Figure 8.14 の 5 トークンの語彙、3 ステップを考える。ビームサーチが最も確率の高い 3 トークンの系列を返さないような条件付き確率を割り当て、両方の系列の確率を計算せよ。この構成例から、ビームサーチが最も確率の高い系列を見逃しうる仕組みについて何が言えるか。
    2. モデルがいかなるものであっても、T ステップにわたるビームサーチが全 $|\mathcal{V}|^T$ 通りの系列のうち最も確率の高いものを返すことを保証する最小のビーム幅はいくらか。
    3. $w \le |\mathcal{V}|$ とし、系列終了トークンは出現しないと仮定する。`step_fn` に対する呼び出しのうち、`beam_search` が T ステップにわたって行う回数を求めよ。また、全探索がすべてのプレフィックスを 1 回ずつ評価する場合の呼び出し回数も求めよ。$|\mathcal{V}| = 1{,}027$、$T = 50$、$w = 4$ のときに両方の回数を計算せよ。

2. [code] **長さ正規化。** `beam_search` を $w = 4$ として、`'The Time Traveller'` と `'It seemed to me that'` の後に最大 40 トークンまで実行せよ。改行文字の ID である `data.tokenizer.encode('\n')[0]` を `eos_id` として渡し、β（引数 `alpha`）を $\{0, 0.75, 1.5\}$ にわたって変化させよ。最上位となった候補の長さと合計対数確率を比較し、Equation 8.26 を用いてその傾向を説明せよ。

3. **温度の下での打ち切り。** `sample_next` は、すべてのカットオフを温度スケーリング後の分布 P_τ（Equation 8.27）に対して適用する。min-p の条件 $P_\tau(x \mid x_{<t}) \ge p_{\min} \max_{x'} P_\tau(x' \mid x_{<t})$ を、温度スケーリング前の確率に関する条件として書き表せ。5 トークンの分布 0.45、0.25、0.15、0.10、0.05 に対して、top-k ($k = 2$)、top-p ($p = 0.85$)、min-p ($p_{\min} = 0.5$) が $\tau = 2$ において保持する集合を計算せよ。3 つの規則のうち、あらゆる温度で同じ集合を保持するものはどれか。

4. [code] **Top-p と min-p の比較。●** $\tau = 1.5$ において、グリッド上の値にわたって p（top-p 用）と $p_{\min}$（min-p 用）を変化させよ。各設定について、任意に選んだ 5 つのプレフィックスから 50 トークンの続きを生成し、生成トークン全体にわたる 2 つの平均値、すなわちその規則が保持するトークン数と、温度スケーリング前のモデルの下で抽出されたトークンの対数確率を記録せよ。規則ごとに 1 本の曲線として、1 つ目の平均値を横軸、2 つ目の平均値を縦軸にとりプロットせよ。保持トークン数の平均が同じとき、より確率の高いトー

クンを抽出するのはどちらの規則か。得られた結果を Nguyen et al. (2025) の知見と比較せよ。

5. [code] **トークンあたりの定数コスト**。●`step_fn` は生成トークンごとにプレフィックス全体にわたってネットワークを再実行するため、T トークンの続きには $O(T^2)$ 回のセル更新コストがかかる（8.6.5.2 節）。

    1. `step_fn` のステートフルな代替関数を作成せよ。ただし、`model.rnn` の隠れ状態を保持して最新のトークンのみを消費し、1 回の呼び出しにつき 1 回のセル更新だけを行うようにすること。両方のバージョンで `generate` が同じ貪欲な続きを生成することを検証せよ。
    2. 50、200、800 トークンに対する `generate` の実行時間を両方のバージョンで計測せよ。その比から、それぞれの実行時間がトークン数に対してどのように増加するかを推定し、セル更新回数が予測する二次増加および線形増加からの乖離があればその理由を説明せよ。
    3. 同じ置き換えがそのままでは `beam_search` に適用できないのはなぜか。また、どのような状態管理を行えば動作させられるか。

発展

「**基礎**」パートの締めくくりでは、H. G. Wells の『The Time Machine』を用いてリカレントニューラルネットワークを訓練した。「**The Time Traveller**」という言葉をプロンプトとして与えると、この小さなモデルは英語として認識できる単語や句読点こそ出力するものの、十数文字もすると文法的な一貫性が崩れてしまう。対照的に、今日の最前線にある基盤モデル（foundation model）は、パラメータ化された層、損失関数、自動微分（automatic differentiation）、勾配ベースのオプティマイザといったまったく同じ基本的な構成要素から築かれているにもかかわらず、立ち入った技術的な対話を長く続け、ソフトウェアを書いてデバッグし、競技数学の問題を解き、1 文のテキストから写実的な画像や動画を合成する。

教室で作るプロトタイプと現代の最前線のシステムを分かつものは何か。その隔たりを越えるには、トイデータセット上の小さなモデルでは決して直面しない 4 つの根本的な課題を解決する必要がある。

1. **最適化の安定性とスケーリング**。「**基礎**」パートでは、ミニバッチ確率的勾配降下法、学習率スケジュール、バッチサイズを主に便利なレシピとして扱ってきた。しかし、ネットワークが数百万から数十億個のパラメータへと大規模化するにつれ、幅の狭い層を滑らかに訓練できる学習率であっても、より幅広い、あるいはより深いネットワークでは瞬時に発散を引き起こすことがあり、悪条件の損失曲面によって数百万ステップにわたり進行が停滞することもある。
2. **長距離の記憶と並列な系列処理**。古典的なリカレントネットワークは、どの細部が後から重要になるかわからないうちに、それまでに読み込んだすべてを固定幅の単一の隠れベクトルへと圧縮せざるを得ない。しかも、各タイムステップが直前のタイムステップの完了を待つため、再帰的な訓練では現代の GPU が備える大規模な並列性を活かせない。
3. **ハードウェアのボトルネックと分散システム**。現代のアクセラレータは、オフチップメモリとオンチップレジスタの間、あるいは GPU 間のネットワークリンク越しにテンソルを移動させる速度よりも、はるかに高速に浮動小数点演算を実行する。大規模環境では、モデルの実環境における訓練速度や推論速度は、単純な演算回数と同程度に、メモリ帯域幅、数値精度、通信トポロジーによって左右される。
4. **静的なラベルを超えた学習**。教師ありの交差エントロピーは、すべての入力に一意な正解ラベルがある場合には機能する。しかし、環境内で行動する自律エージェントは、自身の方策（policy）が変化するにつれて分布がシフトする、遅延した疎なスカラー報酬から学習しなければならない。また、高解像度画像を合成する生成モデルには一致させるべき単一の目標ラベルが存在しないため、確率分布全体を比較する目的関数や、連続的な破壊過程を逆転させる目的関数が必要となる。

PyTorch と JAX の両方で実装された本パートの 9 つの章では、これら 4 つの課題に体系的に取り組む。

- **最適化とスケーリング（第 9 章）**。まずは、悪条件の曲率、鞍点（saddle point）、ミニバッチ勾配のノイズといった要因によって高次元の非凸最適化（nonconvex optimization）がな

ぜ難しくなるのか、そして現代のオプティマイザがそれらの障壁をいかに克服するのかを検証する。確率的勾配降下法（stochastic gradient descent）からモメンタム、AdaGrad、RMSProp、Adam、分離型重み減衰（decoupled weight decay、AdamW）への発展を、ウォームアップやコサイン学習率スケジュールと組み合わせてたどり、最後に、臨界バッチサイズや最大更新パラメータ化（maximal update parametrization、μP）を通じてバッチサイズや幅の拡大に伴って学習率をどう変化させるべきかを明らかにし、完全な訓練プロトコルで締めくくる。

- **アテンション機構と Transformer（第 10 章と第 11 章）**。第 10 章では、キー、値、クエリに対する微分可能な内容参照型検索（content-addressable lookup）である**アテンション**を導入し、古典的な系列モデルにおける固定長ベクトルの記憶ボトルネックを打破する。Nadaraya–Watson カーネル回帰から出発し、加法アテンションとスケール化ドット積アテンション、系列対系列のアライメントのための Bahdanau アテンション、そしてマルチヘッド自己アテンションを構築する。続く第 11 章では再帰構造を完全に排し、自己アテンション、位置ごとのフィードフォワードネットワーク、層正規化（layer normalization）、残差接続、および回転位置埋め込み（rotary position embedding、RoPE）を含む位置エンコーディングを組み合わせて、双方向エンコーダ（BERT）、因果的自己回帰デコーダ（GPT）、Vision Transformer（ViT）という 3 つの基本的な **Transformer** パラダイムへと統合する。あわせて、長文脈アテンションの計算上のボトルネック、生成時のキー・値キャッシュ（KV キャッシュ）、スパースな混合エキスパート（mixture of experts、MoE）層も掘り下げる。
- **現代のリカレントネットワークと状態空間モデル（第 12 章）**。すべてのトークンが他のすべてのトークンに注意を向けなければならない場面では二次の自己アテンションが主流となるが、数十万トークンにわたって肥大化し続けるキー・値キャッシュを保持することは推論時に高コストとなる。そこで第 12 章では、「**推論時には固定サイズの状態を保ちながら、系列のすべての位置にわたって並列に訓練するにはどうすればよいか**」という新たな問いを携えて再帰構造へと立ち返る。状態の非線形な再帰を取り除くことで結合的な**スキャン**（associative scan）による厳密な並列評価が可能になることを示し、線形漸化式（linear recurrence）を連続時間の**状態空間モデル**（state space model、SSM）や HiPPO 多項式射影と結びつけ、さらに行列値の状態（線形アテンション）、入力依存の**選択的**（selective）状態空間モデル（Mamba）、誤り訂正型のデルタ則（DeltaNet）へと進む。
- **計算性能（第 13 章）**。第 13 章では、訓練ステップの時間とメモリがどこに費やされるのかを問う。ルーフラインモデルによって算術演算に律速される演算とメモリトラフィックに律速される演算とを切り分け、さらに 3 つ目の制約である、すべての演算を発行するホスト側の処理によって、細かな演算がなぜ GPU をアイドル状態のまま遊ばせてしまうのかを説明する。そのうえで、コンパイル（`torch.compile` と `jax.jit`）、bf16 演算、より大きなミニバッチ、活性化チェックポイント（activation checkpointing）を組み合わせることで、1 基の GPU 上における GPT の訓練ステップのスループットを約 2 倍に高める。続いて複数の GPU へと拡張し、手書き実装と実用ライブラリの両方によるデータ並列（data parallelism）、allreduce とそのコストモデル、通信と逆伝播のオーバーラップ、そして単一のカードには収まらないモデルを訓練するための重み・勾配・オプティマイザ状態のシャーディング（ZeRO と FSDP）を扱う。
- **強化学習と深層強化学習（第 14 章と第 15 章）**。第 14 章では、不確実性の下での意思決定に関する数学的・アルゴリズム的基礎を築く。具体的には、マルコフ決定過程（Markov decision process）、ベルマン最適方程式、価値反復法と方策反復法、モデルフリーの時間差分学習（Q 学習）、多腕バンディット問題における探索戦略、模倣学習、既知のダイナミクスの下での連続最適制御（倒立振子における LQR、iLQR、モデル予測制御）、REINFORCE と分散低減ベースラインによる直接的な方策最適化、およびニューラル方策勾配を扱う。続く第 15 章では強化学習を深層関数近似へとスケールさせ、深層 Q ネットワーク（DQN）における経験再生とターゲットネットワークによって不安定性の「死の三要素」（deadly triad）に対処したうえで、一般化アドバンテージ推定（generalized

advantage estimation、GAE）を備えたアクタークリティックのアーキテクチャ、近接方策最適化（proximal policy optimization、PPO）、最大エントロピーに基づくソフトアクタークリティック（soft actor-critic、SAC）、オフライン強化学習とモデルベース強化学習、KL 正則化制御、さらには自己回帰言語モデルのための強化学習（GRPO）を展開する。

- **敵対的生成ネットワークと拡散モデル（第 16 章と第 17 章）**。最後の 2 つの章では、現代の深層生成モデリングを発展させる。第 16 章では、生成サンプルと本物のサンプルを見分けるよう学習するクリティックと生成器との間のゲームとして生成を捉える。そこでは、各クリティックの損失がどのダイバージェンス（Jensen–Shannon、他の f ダイバージェンス、MMD、W_1）を測っているのか、また本物から離れた生成器にもなお勾配が得られるのかを問い、本物と生成サンプルをペアでスコア付けするクリティックを比較するとともに、ゼロ中心の勾配ペナルティである R_1 と R_2 がなぜ勾配による訓練をデータ近傍へと収束させるのかを小さなゲーム上で示す。さらに、条件付きモデルを含めた画像生成器を構築してスコア付けし、今日なおボコーダ、画像オートエンコーダ、および拡散モデルから蒸留された 1 ステップ生成器の訓練に用いられている敵対的項で締めくくる。続く第 17 章では、今日の主力となる画像・音声・動画モデルの背後にある連続時間の生成パラダイムを展開する。信号対雑音比（signal-to-noise ratio）によってパラメータ化されたガウス確率経路に沿ってデータを破壊し、クリーンなデータ、ノイズ、スコア関数、あるいは**速度場**（velocity field）を予測するよう深層ノイズ除去器（denoiser）を訓練し、逆時間の確率微分方程式（stochastic differential equation、SDE）または確率フロー ODE を積分して新しいサンプルを生成する。その過程で、ノイズ除去拡散（DDPM/DDIM）と**フローマッチング**を統一し、分類器なしガイダンス（classifier-free guidance）を備えた U-Net および Diffusion Transformer（DiT）のバックボーンを構築し、一貫性（consistency）と平均速度（average velocity）の蒸留によってサンプリングをわずか数ステップへと高速化する。

これらの章は、必ずしも先頭から順番どおりに読む必要はない。どの章がどの章を前提としているかは、まえがきの依存関係図（Figure 1）に示されている。これらの章が一体となって、「**言語モデル**」パートおよび「**画像モデル**」パートが大規模な基盤モデルに応用するアーキテクチャ・数学・システム面のツールキットを提供する。

Chapter 9

最適化アルゴリズム

前章までに登場したすべての深層ネットワークは、モメンタム 0.9 のミニバッチ SGD や、線形ウォームアップとコサイン減衰を組み合わせた AdamW といった標準的なレシピで訓練してきた（7.6 節）。訓練の実行中に値が nan へ発散すれば学習率を下げ、進行が停滞すればスケジュールやバッチサイズを調整する。本章ではオプティマイザのブラックボックスを開き、3 つの実践的な問いに答える。安全に使える最大の学習率とそれ未満での収束速度は何によって決まるのか、現代的な訓練レシピの各要素は更新ステップをどのように作り変えるのか、そしてモデルやバッチサイズが大きくなるにつれてそれらのハイパーパラメータをどのようにスケーリングしなければならないのか。

勾配降下法そのものは現代のコンピュータより古く、1847 年に Augustin-Louis Cauchy が惑星の軌道を計算するために提案し、1951 年には Herbert Robbins と Sutton Monro がノイズを含む観測のもとで減衰するステップサイズを用いた求根問題を解析した。古典的な設定と現代的な設定のいずれにおいても、勾配ベースのあらゆるオプティマイザは次の 2 つの根本的な障害によって制約される。

1. **悪条件性（不均一な曲率）：**損失曲面がある方向には急激に曲がり、別の方向には緩やかに曲がっているとき、最も急な方向で安定を保てるほど小さい学習率を選ぶと、最も平坦な方向への進行が極端に遅くなる。
2. **勾配ノイズ：**勾配はデータセット全体ではなく有限サイズのミニバッチから推定されるため、各更新にはサンプリング分散が伴い、一定のステップサイズでは反復点が最小点に落ち着かない。

Figure 9.1 は、同一の 2 次元の谷におけるこれら両方の障害を示している。現代のオプティマイザにおけるあらゆる設計上の選択は、次の 3 つの制御軸を通じてそのいずれかに対処する。

- **ステップ方向：**モメンタム（9.4 節）、Adam や AdamW のような座標ごとの適応的手法（9.5 節）、および Muon のような行列レベルで直交化する手法（9.6 節）は、生の勾配を前処理（preconditioning）や平滑化を施した方向に置き換えることで、悪条件（ill-conditioned）な谷に沿ってより速く進む。
- **ステップサイズスケジュール：**学習率のウォームアップ、コサイン減衰や線形減衰、およびウォームアップ・安定・減衰（warmup–stable–decay、WSD）スケジュール（9.7 節）は、序盤の速やかな移動と終盤の低いノイズとの間のトレードオフを調整する。
- **ミニバッチサイズとスケーリング：**臨界バッチサイズ（critical batch size、9.8 節）は、ミニバッチを大きくしても最適化ステップ数がそれ以上減らなくなる境界を定める。一方、最大更新パラメータ化（maximal update parametrization、μP、9.9 節）を用いると、小規

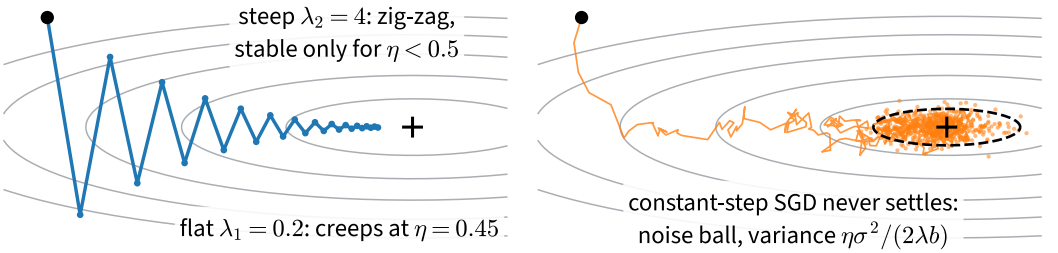


Figure 9.1: 谷を横切る方向の曲率が谷に沿う方向の 20 倍大きい谷（$\kappa = 20$）における 2 つの障害。左：勾配降下法は急な方向をジグザグに行き来しつつ、平坦な方向には少しずつしか進まない。右：ノイズを含む勾配と一定のステップサイズのもとでは、反復点は最小点に到達するものの決して落ち着かず、ノイズ球（破線の楕円：2 標準偏差）の内部をさまよい続ける。

模なプロキシモデルで調整した学習率をはるかに幅の広いネットワークへそのまま転移できる。

本章では各手法を展開するにあたり、条件数（condition number）$\kappa = 20$ の 2 次元の二次関数の谷という解析モデルと、9.3 節で導入する 2 つのニューラルネットワークベンチマーク（畳み込み分類器 `FashionCNN` とバイトレベルの Transformer 言語モデル `TinyLM`）での体系的な実証計測とを組み合わせる。最後に 9.10 節では、勾配クリッピング、重み平均化、ハイパーパラメータのスケーリングを統合し、大規模な訓練の実行を設定・診断するためのエンドツーエンドのプロトコルを構築する。

本章では、勾配、対称行列の固有値、および基本的なミニバッチ SGD（2.1.2.3 節）に関する知識を前提とする。ハイパーパラメータの選択に直結する箇所では簡潔な導出を示し、より長い数学的証明は第 35 章に譲る。教科書、論文、および実務者向けプレイブックに関する注釈付きの参考文献は 9.11 節にまとめている。

9.1 ランドスケープと勾配降下法

モメンタム、適応的な前処理（preconditioning）、学習率スケジュールといった違いはあるものの、深層学習における標準的なオプティマイザはすべて、損失の勾配を評価してその逆方向へ進むという同じ基本操作を反復している。では、なぜその 1 ステップがある目的関数では急速に前進し、別の目的関数では這うようにしか進まないのだろうか。また、学習率（learning rate）をあるしきい値より大きくすると、なぜ突然発散してしまうのだろうか。

d 個のパラメータをもつネットワークは、経験リスク（empirical risk）$f : \mathbb{R}^d \to \mathbb{R}$ を d 次元のパラメータベクトル $\mathbf{x}$ 上に定める。汎化（generalization）は最終的に未知データに対する期待リスクに依存するものの（2.6.1 節）、最適化アルゴリズムの主たる役割は、経験目的関数 $f(\mathbf{x})$ を可能な限り速く確実に減少させることにある。従来の非凸最適化の説明では、質の悪い極小点に陥る危険性が強調されることが多い。しかし高次元のニューラルネットワークでは、訓練ダイナミクスを支配するのは別の 2 つの幾何学的特徴、すなわち**悪条件性**（ill-conditioning、方向によって曲率が大きく異なること）と**鞍点**（saddle point）である。

本節ではまず、悪条件な 2 次元の二次関数の谷における勾配降下法（gradient descent）を解析し、曲率、安定な最大学習率、反復回数の間の厳密な関係を導く。次に、高次元の損失ランドスケープ（loss landscape）における臨界点（critical point）・鞍点・プラトーを調べ、滑らかな関数に対する一般的な降下保証を証明する。さらに、学習率が極小点をその鋭さ（sharpness）

によって選別する仕組み（これがニューラルネットワークの訓練における**安定性の縁**（edge of stability）につながる）を示し、悪条件性への対処法として**前処理**を導入する。

```
import functools
```

9.1.1 最適化ランドスケープと条件数

不均一な曲率の影響を切り分けるため、次の 2 次元の二次目的関数を考えよう。

$$f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2, \tag{9.1}$$

この関数の唯一の最小点は原点にある。二階導関数（曲率）は $\lambda_1 = 0.2$（平坦な軸 x_1 方向）および $\lambda_2 = 4$（急峻な軸 x_2 方向）であり、細長い楕円状の等位集合（level set）が狭い谷を形成している。

学習率 $\eta > 0$ の勾配降下法は、反復点を $\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f(\mathbf{x})$ により更新する。$\nabla f(\mathbf{x}) = (0.2x_1, 4x_2)$ であるため、2 つの座標は互いに分離し、各ステップで乗法的に発展する。

$$\begin{aligned} x_1 &\leftarrow (1 - 0.2\eta)\, x_1, \\ x_2 &\leftarrow (1 - 4\eta)\, x_2. \end{aligned}$$

それぞれの座標は、η に対して相反する制約を課す。

- **安定性の上限（急峻な方向）：**急峻な座標 x_2 が縮小するのは $|1 - 4\eta| < 1$ のとき、かつそのときに限り、ここから厳格な安定性の上限 $\eta < 0.5$ が課される。
- **収束の速さ（平坦な方向）：**安定な任意のステップサイズ $\eta < 0.5$ に対して、平坦な方向の縮小率は $1 - 0.2\eta > 0.9$ を満たす。つまり、各ステップで取り除かれるのは残存誤差の 10% 未満（x_1 方向）である。

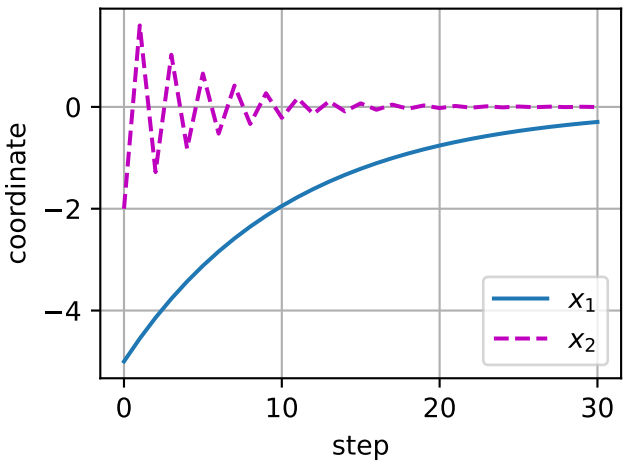
谷の目的関数 `f_valley`、その勾配 `valley_grad`、2 次元の軌跡実行関数 `train_2d`（初期点 $(-5, -2)$）、および通常の勾配降下法の 1 ステップ `gd_2d` を `d2l` に保存し、後続の節ですべてのオプティマイザをまったく同じ曲面上でベンチマークできるようにしておく。

```
VALLEY_CURVATURES = (0.2, 4.0)
h1, h2 = VALLEY_CURVATURES  # Curvatures of the valley along x1 and x2

def f_valley(x1, x2):
    """The ill-conditioned quadratic valley h1/2 x1^2 + h2/2 x2^2, with
    (h1, h2) = VALLEY_CURVATURES."""
    h1, h2 = VALLEY_CURVATURES
    return h1 / 2 * x1 ** 2 + h2 / 2 * x2 ** 2

def valley_grad(x1, x2):
    """Gradient of f_valley."""
    h1, h2 = VALLEY_CURVATURES
    return h1 * x1, h2 * x2

def train_2d(trainer, f_grad, steps=20, **hparams):
    """Run a 2D update rule from (-5, -2) and return the iterates."""
    x1, x2, s1, s2 = -5, -2, 0, 0  # s1, s2: state of stateful optimizers
```

```
    results = [(x1, x2)]
    for _ in range(steps):
        x1, x2, s1, s2 = trainer(x1, x2, s1, s2, f_grad, **hparams)
        results.append((x1, x2))
    print(f'step {steps}, x1: {float(x1):f}, x2: {float(x2):f}')
    return results

def gd_2d(x1, x2, s1, s2, f_grad, eta):
    """One step of gradient descent in 2D; the state stays zero."""
    g1, g2 = f_grad(x1, x2)
    return x1 - eta * g1, x2 - eta * g2, 0, 0
```

安定性の上限のわずかに下である $\eta = 0.45$ では、急峻な方向の乗数は $1 - 4(0.45) = -0.8$ となるため、x_2 は谷底を行き過ぎて毎ステップ符号を反転させながら、絶対値で 20% ずつ縮小する。一方、平坦な方向の乗数は $1 - 0.2(0.45) = 0.91$ であるため、30 ステップを $(-5, -2)$ から進めた後でも、平坦な座標は依然として $-5 \times 0.91^{30} \approx -0.3$ 付近にとどまる。

```
h = np.array([0.2, 4.0])  # Curvatures h1, h2 along x1 and x2

def gd_valley(eta, steps, x0=(-5.0, -2.0)):
    """Gradient descent on the valley; returns all iterates."""
    xs = [np.array(x0)]
    for _ in range(steps):
        xs.append(xs[-1] - eta * h * xs[-1])  # The gradient is h * x
    return np.stack(xs)

xs = gd_valley(0.45, steps=30)
d2l.plot(np.arange(31), [xs[:, 0], xs[:, 1]], 'step', 'coordinate',
         legend=['$x_1$', '$x_2$'])
```



η を $(0, 0.5]$ の範囲で変化させると、全体の収束時間（両方の座標を原点からの初期距離の 1% 以内に収めるまでに要するステップ数で測る）が、これら 2 つの因子のバランスにどう依存するかがわかる。

```
    print('  eta   factor x1   factor x2   steps to 1%')
    for eta in [0.1, 0.2, 0.3, 0.4, 0.45, 2 / 4.2, 0.49, 0.5]:
        xs = gd_valley(eta, steps=1000)
        done = np.all(np.abs(xs) <= 0.01 * np.abs(xs[0]), axis=1)
        steps = int(np.argmax(done)) if done.any() else 'never'
        print(f'{eta:5.3f}   {1 - 0.2 * eta:9.3f}   {1 - 4 * eta:9.3f}   '
              f'{steps:>11}')
```

```
  eta   factor x1   factor x2   steps to 1%
0.100       0.980       0.600           228
0.200       0.960       0.200           113
0.300       0.940      -0.200            75
0.400       0.920      -0.600            56
0.450       0.910      -0.800            49
0.476       0.905      -0.905            47
0.490       0.902      -0.960           113
0.500       0.900      -1.000         never
```

最適なステップサイズは平坦な方向と急峻な方向の乗数の絶対値を等しくする条件 $1-0.2\eta = -(1-4\eta)$ から求まり、$\eta^\star = 2/4.2 \approx 0.476$ となる。このとき両方の因子の絶対値は 0.905 となり、1% への収束には 47 ステップを要する。安定性のしきい値付近における急激な非対称性に注目しよう。$\eta^\star$ よりわずかに小さい $\eta = 0.45$ に下げてもステップ数は 2 ステップ増えるだけ（49 ステップ）であるのに対し、わずかに大きい $\eta = 0.49$ に上げるとステップ数は 2 倍以上に跳ね上がる（113 ステップ）。

この解析は、任意の d 次元の正定値二次関数 $f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\top \mathbf{H}\mathbf{x}$（対称なヘッセ行列 $\mathbf{H} \succ \mathbf{0}$ をもつ）へ直ちに拡張できる。$\mathbf{x} = \sum_{i=1}^{d} c_i \mathbf{v}_i$ と $\mathbf{H}$ の正規直交固有基底（$\mathbf{H}\mathbf{v}_i = \lambda_i \mathbf{v}_i$ かつ $0 < \lambda_{\min} = \lambda_1 \le \cdots \le \lambda_d = \lambda_{\max}$）で展開すると、勾配降下法のステップは各固有ベクトルに沿って分離する。

$$(\mathbf{I} - \eta \mathbf{H}) \sum_{i=1}^{d} c_i \mathbf{v}_i = \sum_{i=1}^{d} (1 - \eta \lambda_i)\, c_i \mathbf{v}_i. \tag{9.2}$$

安定であるためには $|1-\eta\lambda_i| < 1$ がすべての i について成り立つ必要があり、これは最大固有値によって決まる。

$$\eta < \frac{2}{\lambda_{\max}}. \tag{9.3}$$

この上限のもとでは、最も平坦な固有ベクトル $\mathbf{v}_1$ に沿った進み具合は、次の**条件数**（condition number）のみに依存する。

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{9.4}$$

目的関数 f を定数 $c > 0$ 倍に再スケーリングすると、すべての固有値は c 倍になり、安定な学習率は c で割られるため、軌跡は不変に保たれる。無次元の比 κ こそが、曲面の本質的な幾何学的難しさを捉えている。両端の縮小率をつり合わせる条件 $1-\eta\lambda_{\min} = -(1-\eta\lambda_{\max})$ から、最適な定数学習率 $\eta^\star = 2/(\lambda_{\min}+\lambda_{\max})$ と、1 ステップあたりの最悪ケースの縮小率

$(\kappa-1)/(\kappa+1) \approx 1-2/\kappa$ が得られる（35.1.2 節）。したがって、誤差を ϵ 倍に減らすのに必要なステップ数は次式となる。

$$t \approx \frac{\kappa}{2}\log\frac{1}{\epsilon} \tag{9.5}$$

この谷（$\kappa = 4/0.2 = 20$、$\epsilon = 0.01$）に対して、この式は $10\log 100 \approx 46$ ステップを予測し、上で観測された 47 ステップと一致する。t は κ に比例して増大するため、いかなるスカラーの学習率を用いても悪条件性を克服することはできない。収束を加速するには、更新方向そのものを変える必要がある。

9.1.2 臨界点、鞍点、プラトー

凸二次関数の椀型の曲面とは異なり、ニューラルネットワークの損失ランドスケープは非凸であり、$\nabla f(\mathbf{x}) = \mathbf{0}$ となる多数の**臨界点**や、$\|\nabla f(\mathbf{x})\|$ が極めて小さくなる**プラトー**をもつ。2 回微分可能な関数 $f : \mathbb{R}^d \to \mathbb{R}$ の任意の臨界点において、ヘッセ行列 $\mathbf{H} = \nabla^2 f(\mathbf{x})$ の固有値は局所的な幾何構造を次のように分類する。

- **極小点：**d 個の固有値がすべて真に正である（$\mathbf{H} \succ \mathbf{0}$）。
- **極大点：**d 個の固有値がすべて真に負である（$\mathbf{H} \prec \mathbf{0}$）。
- **鞍点：**$\mathbf{H}$ が正と負の両方の固有値をもち、ある固有ベクトルに沿っては上向きに曲がり、別の固有ベクトルに沿っては下向きに曲がる。原点における $f(x, y) = x^2 - y^2$ がその例である（Figure 9.2）。
- **退化臨界点：**$\mathbf{H}$ が 0 の固有値をもち、かつ異符号の固有値をもたない点（degenerate critical point）であり、極値点（x^4 など）か変曲点（x^3 など）かは高階の導関数によって決まる。

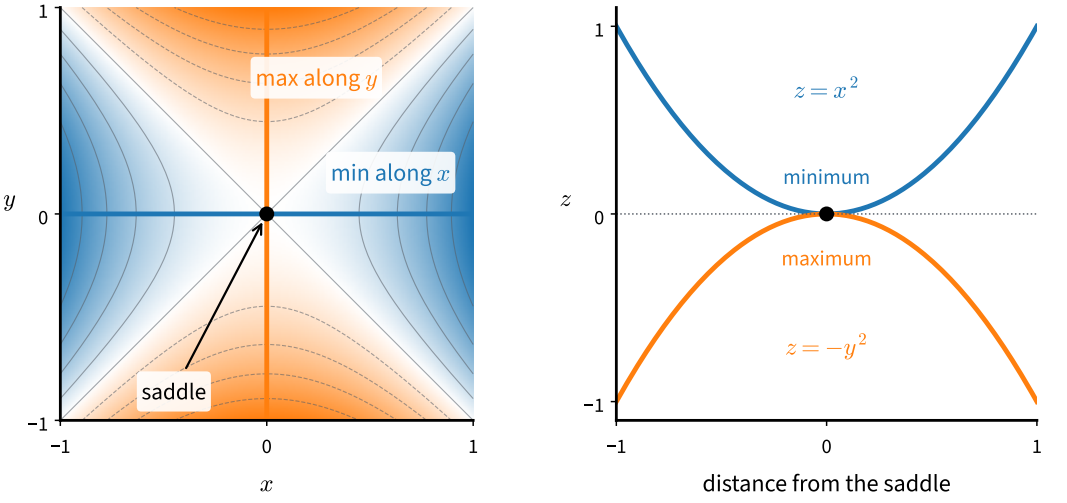


Figure 9.2: 鞍面 $z = x^2 - y^2$ は x 方向に極小、y 方向に極大をもち（左）、原点を通る 2 つの断面にもそれが現れている（右）。

高次元では、鞍点の数が極小点を圧倒的に上回る。臨界点が極小点であるためには、d 個のヘッセ行列の固有値がすべて正でなければならない。単純なランダム符号のヒューリスティックのもとでは、d 個すべての固有値が正となる確率は 2^{-d} にすぎない（またランダム行列理論によれ

ば、基準となる損失が十分に低くなるまでは、対称ランダム行列の固有値は 0 の周りに均等に分かれることが確認されている。演習問題 4 を参照)。ニューラルネットワークにおけるヘッセ行列の実測でも、高い損失値にある臨界点は負の固有値を多数もつ鞍点で占められており、負の固有値の割合が 0 に近づくのは低い損失値の領域に限られることが確かめられている (Dauphin et al. 2014)。

重要なことに、**狭義鞍点**（strict saddle point、少なくとも 1 つの負の固有値 $\lambda < 0$ をもつ臨界点）は勾配降下法を閉じ込めない。曲率 $-\mu < 0$ で下向きに曲がる固有ベクトルに沿っては、勾配ステップのたびに変位が $(1 + \eta\mu) > 1$ 倍に増幅される。狭義鞍点へ収束するような初期値の安定多様体（stable manifold）は測度 0 であるため、ランダム初期化からの勾配降下法は確率 1 で鞍点から脱出する (Lee et al. 2016)。尾根から距離 δ の位置から出発すると、不安定方向に沿って距離 1 に達するまでのステップ数は $\log(1/\delta)/\log(1 + \eta\mu) \approx \log(1/\delta)/(\eta\mu)$ となる。これは δ については対数的であるが、負の曲率の大きさ μ には反比例する。さらに、ミニバッチ勾配のノイズは $\delta = 0$ の場合であっても反復点を尾根から積極的に弾き出す（演習問題 1)。

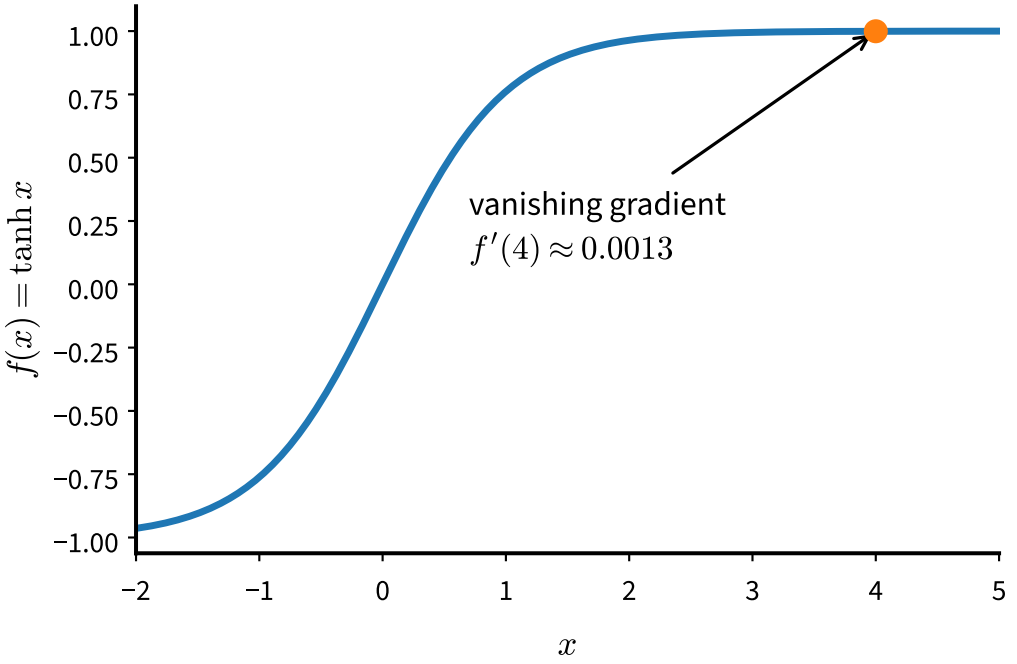


Figure 9.3: $f(x) = \tanh x$ を $x = 4$ から最小化すると、臨界点が一切ないにもかかわらず、$f'(4) \approx 0.0013$ となる活性化プラトー上で停滞する。

対照的に、**プラトー**は活性化関数の導関数が飽和するとき、臨界点がなくても生じる。たとえば $f(x) = \tanh(x)$ の $x = 4$ 付近（Figure 9.3）では、$f'(4) = 1 - \tanh^2(4) \approx 0.0013$ であるため、$\eta = 1$ の勾配降下法は 370 ステップもかけて $x = 4$ から $x = 2$ まで這うように進むことになる。深層ネットワークにおいて、このような勾配消失（4.4 節）は主にアーキテクチャの設計、すなわち非飽和型の活性化関数（4.1.4 節)、正規化層（7.3 節)、残差接続（7.4 節）によって解消されており、オプティマイザにとっての中心的な課題として残るのは悪条件性である。

ニューラルネットワークの損失は大域的には非凸であるものの（たとえば、ある層の d 個の隠れユニットを並べ替えると同一の損失をもつ $d!$ 個の異なるパラメータベクトルが得られ、それらの平均はより損失の高い単一ユニットのネットワークに潰れてしまう。演習問題 3)、任意の反復点の周りにおける局所的な幾何構造はそのヘッセ行列のスペクトルによって決まる。深層ネットワ

ークの実測スペクトルは、0 付近に密集する膨大な固有値のバルクと、少数の大きな正の外れ値固有値を示す (Ghorbani et al. 2019)。**鋭さ** $\lambda_{\max}$ と呼ばれる最大固有値が安定な学習率の上限を決める一方、平坦なバルクは $\eta\lambda_i \ll 1$ の速さでしか進まず、高次元空間において狭い谷のボトルネックが再現される。

9.1.3 降下保証と極小点の安定性

非二次の目的関数では、曲率は場所ごとに変化する。それにもかかわらず、なぜ $2/\lambda_{\max}$ という規則が依然としてステップサイズの安定性を支配するのだろうか。

1 次のテイラー展開 $f(\mathbf{x}+\epsilon) \approx f(\mathbf{x}) + \epsilon^\top \nabla f(\mathbf{x})$ とコーシー・シュワルツの不等式により、負の勾配 $-\nabla f(\mathbf{x})$ は局所的な最急降下方向となる。有限のステップ $\mathbf{x} - \eta\nabla f(\mathbf{x})$ が実際に f を減少させることを保証するため、f が L **滑らか**（L-smooth）、すなわちその勾配が L リプシッツ連続であると仮定しよう（1 変数の場合は $|f'(y) - f'(x)| \le L|y-x|$、あるいは $|f''(x)| \le L$）。ステップ ϵ に沿って傾きの差を積分すると、f は曲率 L の接する放物線によって上から抑えられる。

$$
\begin{aligned}
f(x+\epsilon) - f(x) - \epsilon f'(x) &= \int_0^\epsilon \left(f'(x+s) - f'(x)\right) ds \\
&\le \int_0^\epsilon L s \, ds = \frac{L}{2}\epsilon^2.
\end{aligned}
$$

ここに勾配ステップ $\epsilon = -\eta f'(x)$（あるいは $\epsilon = -\eta\nabla f(\mathbf{x})$、すなわち d 次元で $\|\nabla^2 f(\mathbf{x})\|_2 \le L$ の場合。35.1.1 節）を代入すると、次の**降下補題**（descent lemma）が得られる。

$$
f(\mathbf{x} - \eta\nabla f(\mathbf{x})) \le f(\mathbf{x}) - \eta\Big(1 - \frac{\eta L}{2}\Big)\|\nabla f(\mathbf{x})\|^2. \tag{9.6}
$$

$0 < \eta < 2/L$ である限り、臨界点以外の任意の点で各勾配ステップは f を真に減少させ、1 ステップあたりの保証減少量は $\|\nabla f(\mathbf{x})\|^2/(2L)$（$\eta = 1/L$ のとき）で最大となる。

ここで視点を逆転させ、曲率の異なる複数の極小点をもつ損失ランドスケープ上で学習率 η を固定してみよう。任意の極小点 $x^\star$（曲率 $\lambda = f''(x^\star) > 0$）の近傍では、1 回の勾配ステップによって微小な変位 $x - x^\star$ が $1 - \eta\lambda$ 倍になる。その結果、次のことが成り立つ。

- $\lambda < 2/\eta$ ならば $|1 - \eta\lambda| < 1$ となり、この極小点は**線形安定**（linearly stable）である（近傍の反復点を引き寄せる）。
- $\lambda > 2/\eta$ ならば $1 - \eta\lambda < -1$ となり、この極小点は**不安定**である（厳密に $x^\star$ 上に着地しない限り、あらゆる反復点を弾き出す）。

したがって、学習率 η は極小点を深さではなく**鋭さ**によって選別する。すなわち、勾配降下法が収束できるのは鋭さが $\lambda_{\max} \le 2/\eta$ を満たす極小点に限られる。この選別は $f(x) = x\cos(0.15\pi x)$ 上で直接観察できる。この関数は $x = 7.27$ 付近に深くより鋭い極小点（$f'' \approx 1.81$）をもち、$x = -1.83$ 付近により浅く平坦な極小点（$f'' \approx 0.98$）をもつ（Table 9.1 にまとめる）。

Table 9.1: 局所曲率 f'' と 1 ステップあたりの変位の乗数 $1 - \eta f''$（$f(x) = x\cos(0.15\pi x)$ の 2 つの極小点における値）。

極小点	曲率 f''	$\eta = 0.5$ での乗数	$\eta = 2$ での乗数
深い極小点（$x = 7.27$）	1.81	0.09（安定）	−2.62（不安定）

極小点	曲率 f''	$\eta = 0.5$ での乗数	$\eta = 2$ での乗数
浅い極小点 ($x = -1.83$)	0.98	0.51（安定）	-0.96（安定）

まず、1 次元の勾配降下法を実行する関数 `gd` と軌跡を描画する関数 `show_trace` を定義する。

```
def gd(eta, f_grad, x=10.0, steps=10):
    """Gradient descent from x; returns all iterates."""
    xs = [x]
    for _ in range(steps):
        x = x - eta * f_grad(x)
        xs.append(float(x))
    return xs

def show_trace(f, *runs, legend=None):
    """Draw the iterates of one or more runs on the graph of f."""
    n = max(abs(x) for xs in runs for x in xs)
    grid = d2l.arange(-n, n, 0.01)
    d2l.plot([grid, *runs], [f(grid), *[[f(x) for x in xs] for xs in runs]],
             '$x$', '$f(x)$', legend=legend, fmts=['-', '-o', '-s'])
```

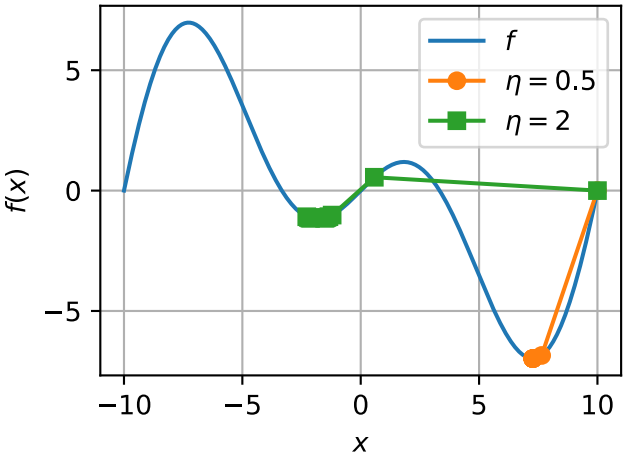
次に、$f(x) = x\cos(0.15\pi x)$ とその一階導関数 `f_cos_grad` および二階導関数 `f_cos_hess` を定義する。

```
c = d2l.tensor(0.15 * np.pi)

def f_cos(x):  # Objective function
    return x * d2l.cos(c * x)

def f_cos_grad(x):  # Its derivative
    return d2l.cos(c * x) - c * x * d2l.sin(c * x)

def f_cos_hess(x):  # Its second derivative
    return -2 * c * d2l.sin(c * x) - c**2 * x * d2l.cos(c * x)
```

2 つの極小点で `f_cos_hess` を評価し、`gd` を $x = 10$ から $\eta \in \{0.5, 2\}$ で実行すると、安定性の上限 $2/\eta$ が深く鋭い極小点と浅く平坦な極小点のどちらを選ぶかがよくわかる。

```
minima = [gd(0.5, f_cos_grad, x=x, steps=50)[-1] for x in (10.0, 0.0)]
print(', '.join(f"f''({x:.2f}) = {float(f_cos_hess(x)):.2f}" for x in minima))
runs = {eta: gd(eta, f_cos_grad, steps=100) for eta in (0.5, 2)}
for eta, xs in runs.items():
    print(f'eta {eta}, 2/eta = {2/eta:.0f}: x = {xs[10]:.3f} after 10 steps, '
          f'{xs[100]:.3f} after 100')
show_trace(f_cos, *[xs[:11] for xs in runs.values()],
           legend=['$f$', r'$\eta = 0.5$', r'$\eta = 2$'])
```

```
f''(7.27) = 1.81, f''(-1.83) = 0.98
eta 0.5, 2/eta = 4: x = 7.269 after 10 steps, 7.269 after 100
eta 2, 2/eta = 1: x = -1.528 after 10 steps, -1.821 after 100
```



$x = 10$ から出発すると、$\eta = 0.5$（$2/\eta = 4 > 1.81$）での実行は $x = 7.269$ の深い極小点へ急速に収束する。一方、$\eta = 2$（$2/\eta = 1 < 1.81$）では深い極小点が不安定（$1 - \eta f'' = -2.62$）となり、反復点は代わりに $x = -1.83$ 付近のより平坦な極小点（$f'' = 0.98 < 2/\eta$）へと落ち着く。

9.1.4 安定性の縁

ニューラルネットワークの訓練では、鋭さ $\lambda_{\max}$ と安定性の限界 $2/\eta$ の関係は動的なものになる。Cohen et al. (2021) は、フルバッチ勾配降下法のもとで損失が下がるにつれてヘッセ行列の鋭さ $\lambda_{\max}(\nabla^2 f(\mathbf{x}_t))$ が着実に上昇し（**漸進的鋭化**（progressive sharpening））、やがて安定性のしきい値 $2/\eta$ に達することを示した。その後、訓練は発散するのではなく、残りの訓練期間を通じて鋭さが $2/\eta$ 付近にとどまり、損失は非単調に減少し続ける。この動作領域は**安定性の縁**として知られている。

この現象は、パラメータ数 25 の単一隠れ層 tanh ネットワークを 16 個の点（$y = \sin(3x)$、区間 $[-2, 2]$）に当てはめる実験で直接測定できる。$d = 25$ であるため、厳密な 25×25 のヘッセ行列とその最大固有値 $\lambda_{\max}$ を 500 ステップごとに、10 個のランダム初期化にわたって $\eta = 0.4$（$2/\eta = 5$）および $\eta = 0.25$（$2/\eta = 8$）のそれぞれについて計算できる。

```
X = jnp.linspace(-2, 2, 16)
Y = jnp.sin(3 * X)

def mlp_loss(p):  # 8 hidden units: weights, biases, output weights, bias
    W1, b1, W2, b2 = p[:8], p[8:16], p[16:24], p[24]
    return 0.5 * ((jnp.tanh(X[:, None] * W1 + b1) @ W2 + b2 - Y) ** 2).mean()

def sharpness(P):  # Largest eigenvalue of the Hessian, for each row of P
    return jnp.linalg.eigvalsh(jax.vmap(jax.hessian(mlp_loss))(P))[:, -1]

@functools.partial(jax.jit, static_argnames=('steps', 'every', 'last'))
def gd_sharpness(eta, P, steps=20000, every=500, last=1000):
    """Full-batch gradient descent from each row of P. Returns the sharpness
    every `every` steps, the final loss, and the fraction of the last `last`
    steps on which the loss rose."""
    grad_and_loss = jax.vmap(jax.value_and_grad(mlp_loss))
```

```
    def step(P, _):
        loss, g = grad_and_loss(P)
        return P - eta * g, loss
    def block(P, _):  # Measure the sharpness, then take `every` steps
        P_next, losses = jax.lax.scan(step, P, length=every)
        return P_next, (sharpness(P), losses)
    P, (sharp, losses) = jax.lax.scan(block, P, length=steps // every)
    losses = jnp.vstack([losses.reshape(steps, -1), jax.vmap(mlp_loss)(P)])
    losses = losses[-last - 1:]  # (last + 1, starts)
    return (jnp.vstack([sharp, sharpness(P)]).T, losses[-1],
            jnp.mean(losses[1:] > losses[:-1], axis=0))

P0 = 0.5 * jax.random.normal(d2l.get_key(), (10, 25))  # Ten random starts
```

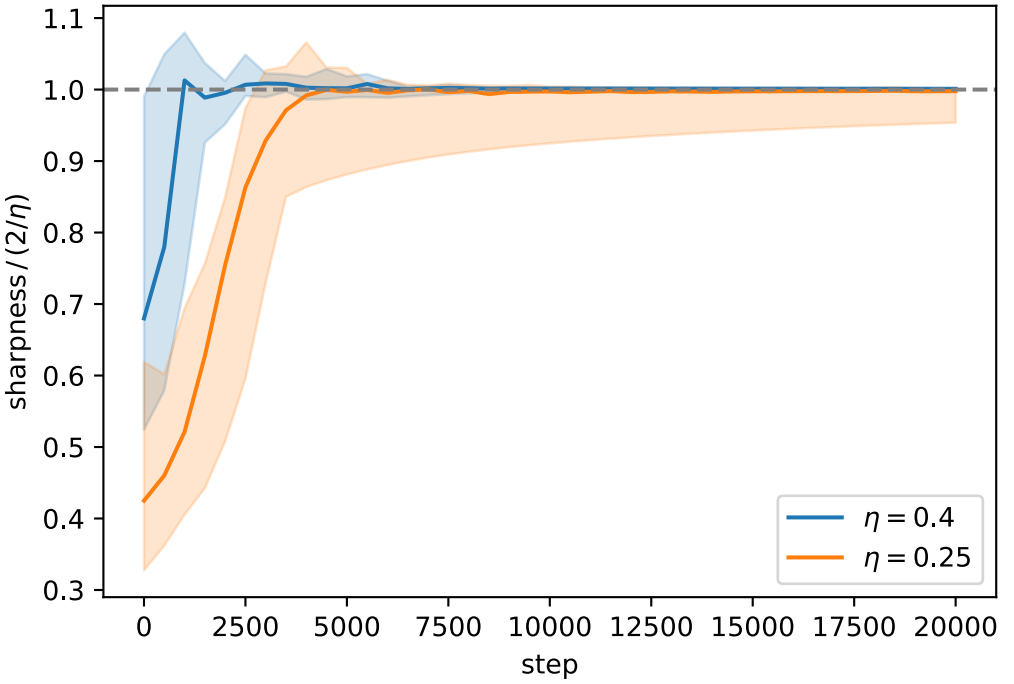
関数 gd_sharpness を共通の 10 通りの初期化 P0 から $\eta = 0.4$ $(2/\eta = 5)$ および $\eta = 0.25$ $(2/\eta = 8)$ で実行し、初期および最終の鋭さ、最終損失、ならびに終盤のステップのうち損失が上昇した割合を表にまとめよう。

```
runs = {eta: gd_sharpness(eta, P0) for eta in (0.4, 0.25)}
print('      start      |' + '|'.join(
    f'    eta {eta}, 2/eta = {2 / eta:.0f}     ' for eta in runs))
print('sharpness   loss |' + '|'.join([' sharpness    loss  rising '] * 2))
for i in range(len(P0)):
    print(f'{float(runs[0.4][0][i, 0]):9.2f} {float(mlp_loss(P0[i])):6.3f} |'
          + '|'.join(f' {float(s[i, -1]):9.2f} {float(l[i]):7.1e} '
                     f'{float(r[i]):6.0%} ' for s, l, r in runs.values()))
```

```
      start      |    eta 0.4, 2/eta = 5     |    eta 0.25, 2/eta = 8
sharpness   loss | sharpness    loss  rising | sharpness    loss  rising
     3.49  0.932 |      5.01 5.1e-04    50% |      8.01 1.8e-04    50%
     4.11  0.422 |      5.01 3.7e-04    50% |      7.63 6.8e-05     0%
     4.95  0.585 |      5.00 9.8e-04    50% |      7.98 1.7e-04    50%
     2.80  0.597 |      5.01 5.0e-04    50% |      7.98 1.1e-04    50%
     4.15  0.599 |      5.00 3.4e-04    50% |      7.98 7.2e-05     7%
     3.78  0.237 |      5.00 4.2e-04    50% |      8.01 1.1e-04    50%
     3.31  0.271 |      5.00 3.1e-04    50% |      7.94 6.4e-05     0%
     2.83  0.306 |      4.99 4.3e-04    50% |      7.98 2.0e-04    50%
     2.62  0.229 |      5.01 3.9e-04    50% |      7.99 4.5e-05     0%
     2.68  0.767 |      5.01 3.7e-04    50% |      8.01 1.8e-04    50%
```

安定性のしきい値で正規化した鋭さ $\lambda_{\max}/(2/\eta)$ を全 20,000 ステップにわたってプロットすると、どちらの学習率でも鋭さが 1 の安定性の縁に向かって上昇していく様子がわかる。

```
curves = {rf'$\eta = {eta}$': eta / 2 * s for eta, (s, _, _) in runs.items()}
d2l.plot_curves(curves, 'step', r'sharpness$\,/\,(2/\eta)$',
                x=np.arange(0, 20001, 500), reference=1)
```



まったく同じ初期化から出発しても、正規化された鋭さ $\lambda_{\max}/(2/\eta)$ は 1 まで上昇してそこで頭打ちになる。すなわち、$\eta = 0.4$ の実行は鋭さ $\lambda_{\max} \approx 5.0$ で停止する一方、$\eta = 0.25$ の大半の実行は $\lambda_{\max} \approx 8.0$ まで上昇する。すべての実行がそこに到達するわけではなく、$\eta = 0.25$ ではいくつかの初期値が鋭さ 8 に達する前にデータに適合し、しきい値のわずかに下にとどまったまま、損失が上昇するステップをほとんど伴わずに降下する。しきい値 $2/\eta$ に達した実行では、損失は典型的に約 50% のステップで上昇する。これは、反復点が最も鋭い固有ベクトルに沿って振動する（$1 - \eta\lambda_{\max} \approx -1$）一方で、残りの 24 個のより平坦な方向に沿っては降下し続けるためである。$\lambda_{\max}$ が一時的に $2/\eta$ を超えるたびに、最も鋭い固有ベクトルに沿った振動の増大が損失の 3 階導関数と結合してより平坦な領域への復元ドリフトを引き起こし、$\lambda_{\max}$ を $2/\eta$ に安定化させる (Damian et al. 2023)。この自己安定化は、訓練開始時に η を徐々に引き上げる手法（**学習率のウォームアップ**、9.7 節）によって、ピーク学習率に耐えられるほど平坦な領域へとネットワークが誘導される理由も説明している (Kalra and Barkeshli 2024)。

この実験ではフルバッチ勾配降下法を用いたが、本章の残りの節ではモメンタム、Adam、およびミニバッチを用いて訓練する。安定性の縁はこれら 3 つすべてにおいても保たれ、それぞれ異なるしきい値をもつ。係数 β のヘビーボールモメンタム（9.4 節）は、安定性の限界を $\eta\lambda_{\max} < 2$ から $\eta\lambda_{\max} < 2(1+\beta)$ へと引き上げ、それに応じて鋭さも $2(1+\beta)/\eta$ に落ち着く (Cohen et al. 2021)。Adam（9.5 節）はステップの各座標をそれぞれの勾配スケールで割るため、定常値に落ち着く量は同じ再スケーリングを施した後のヘッセ行列の鋭さ、すなわち**前処理後の鋭さ**（preconditioned sharpness）となる。Adam で通常用いられるモメンタム係数 $\beta_1 = 0.9$ の場合、そのしきい値は約 $38/\eta$ である (Cohen et al. 2022)。ミニバッチ勾配を用いると、鋭さは $2/\eta$ より手前で上昇を止め、バッチが小さいほどより早い段階で止まる。いずれの場合も、学習率が大きいほど鋭さの上限は低くなるため、ウォームアップに関する議論はこれ以降で用いるオプティマイザにもそのまま当てはまる。

9.1.5 ニュートン法と前処理

9.1.5.1 二次ステップとニュートン法

なぜ勾配降下法は谷 Equation 9.1 で苦戦するのだろうか。点 $(-5, -2)$ において、負の勾配 $(1, 8)$ は原点 $(0, 0)$ ではなく、ほぼ真横の急峻な壁を向いている。急峻な座標 x_2 が 1 ステップで谷底に到達するには $\eta = 1/4 = 0.25$ が必要であるのに対し、平坦な座標 x_1 には $\eta = 1/0.2 = 5$ が必要となる。

ニュートン法（Newton's method）は、$\mathbf{x}$ の周りの 2 次テイラー近似を最小化することで、各方向にその局所曲率の逆数を与える。

$$f(\mathbf{x} + \epsilon) \approx f(\mathbf{x}) + \epsilon^\top \nabla f(\mathbf{x}) + \frac{1}{2}\epsilon^\top \mathbf{H} \epsilon. \tag{9.7}$$

ϵ に関する勾配を 0 とおくと、ニュートン法の更新式が得られる。

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}^{-1} \nabla f(\mathbf{x}). \tag{9.8}$$

任意の正定値二次関数では 2 次展開が厳密に一致するため、条件数 κ にかかわらず 1 回のニュートンステップで最小点に到達する。一般の滑らかな関数の正則な極小点の近傍でも、ニュートン法は**二次収束**（quadratic convergence）を達成し、ステップごとに有効桁数が倍増する（35.2 節）。

このような高速な局所収束性にもかかわらず、厳密なニュートンステップは次の 3 つの理由から深層ニューラルネットワークには非現実的である。

1. **計算コスト：**ヘッセ行列・ベクトル積（34.5.3.2 節）を用いて $d \times d$ のヘッセ行列（要素数 10^{18} 個、$d = 10^9$ の場合）の保存を回避したとしても、$\mathbf{H}\epsilon = -\nabla f(\mathbf{x})$ を反復法で解くには 1 回のパラメータ更新あたり数十回の誤差逆伝播法の実行が必要になる。
2. **負の曲率：**$\mathbf{H}$ はすべての臨界点に対して対称に作用するため、ダンピングや絶対値スペクトルへの射影による修正（35.2.1 節）を加えない限り、負の固有値がステップ方向を反転させ、反復点を鞍点や極大点に向かって上り方向へと駆動してしまう。
3. **ミニバッチノイズへの感度：**0 付近のノイズを含んだ曲率推定値の逆数をとると、サンプリングノイズが壊滅的に増幅される。

9.1.5.2 対角前処理

実用的な深層学習のオプティマイザは、ヘッセ行列全体の逆行列を求める代わりに、各座標（または層の行列）を安価な曲率推定値で再スケーリングするという二次手法の手頃な中核部分だけを残す。一般に、$\mathbf{x} \leftarrow \mathbf{x} - \eta \mathbf{P}^{-1} \nabla f(\mathbf{x})$ の形をとる更新は**前処理付き勾配降下法**（preconditioned gradient descent）と呼ばれ、ここで**前処理行列**（preconditioner）$\mathbf{P}$ は $\mathbf{P}^{-1/2}\mathbf{H}\mathbf{P}^{-1/2}$ の有効条件数を小さくする役割を果たす。

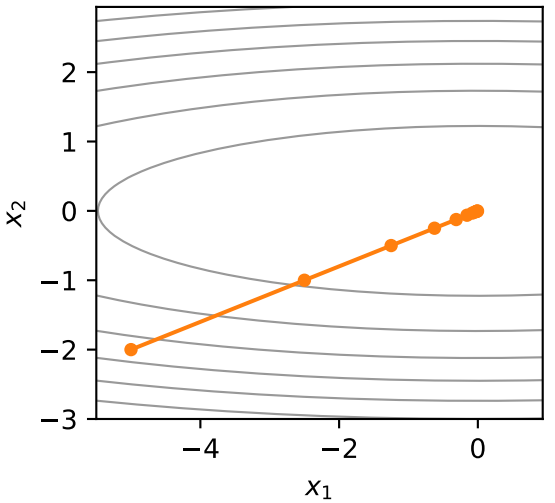
最も単純な選択は**対角前処理行列**（diagonal preconditioner）$\mathbf{P} = \operatorname{diag}(\mathbf{H})$ であり、勾配の各座標をそれぞれの二階導関数で割る。

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \operatorname{diag}(\mathbf{H})^{-1} \nabla f(\mathbf{x}). \tag{9.9}$$

$\operatorname{diag}(\mathbf{H}) = (0.2, 4)$ である軸に沿った谷では、Equation 9.9 は毎ステップ両方の座標を同一の因子 $1 - \eta$ で縮小させ、有効条件数を $\kappa = 20$ から 1 へと引き下げる。$\eta = 0.5$ のとき、両方の座標が初期値の 1% 未満に下がるまでのステップ数は、わずか 7 ステップ（$0.5^7 \approx 0.0078$）となり、従来の 47 ステップから大幅に短縮される。

```
def precond_gd_2d(x1, x2, s1, s2, f_grad, eta):
    g1, g2 = f_grad(x1, x2)
    return x1 - eta * g1 / h1, x2 - eta * g2 / h2, 0, 0  # diag(H) = (h1, h2)

def steps_to_1pct(xs):  # First step with both coordinates within 1% of start
    return next(t for t, (x1, x2) in enumerate(xs)
                if abs(x1) <= 0.01 * 5 and abs(x2) <= 0.01 * 2)

plain = train_2d(gd_2d, valley_grad, steps=100, eta=2 / (h1 + h2))
precond = train_2d(precond_gd_2d, valley_grad, steps=10, eta=0.5)
print(f'steps to 1%: plain at its best rate {steps_to_1pct(plain)}, '
      f'preconditioned {steps_to_1pct(precond)}')
d2l.show_trace_2d(f_valley, precond)
```

```
step 100, x1: -0.000225, x2: -0.000090
step 10, x1: -0.004883, x2: -0.001953
steps to 1%: plain at its best rate 47, preconditioned 7
```



深層ネットワークでは $\mathrm{diag}(\mathbf{H})$ が直接与えられるわけではなく、主曲率の方向が座標軸と一致しているわけでもない（谷を 45° 回転させると $\mathbf{H}$ の対角成分が等しくなり、対角前処理は効果を失う。演習問題 7)。後続の節ではこの両方の限界に対処する。すなわち、Adam（9.5 節）は二乗勾配から座標ごとのスケールをオンラインで推定し、モメンタム（9.4 節）と Muon（9.6 節）は座標軸に沿わない方向への進行を加速する。ただしその前にまず、厳密な勾配をノイズを含んだミニバッチ推定値に置き換えたときに何が起こるかを調べよう（9.2 節)。

9.1.6 まとめ

ヘッセ行列の固有値 $\lambda_{\min} \leq \cdots \leq \lambda_{\max}$ をもつ正定値二次関数において、勾配降下法が安定であるのは $\eta < 2/\lambda_{\max}$ のとき、かつそのときに限られる。また、相対精度 ϵ に最適な定数ステップサイズ $\eta^\star = 2/(\lambda_{\min} + \lambda_{\max})$ で到達するには $t \approx (\kappa/2)\log(1/\epsilon)$ ステップを要する（ここで $\kappa = \lambda_{\max}/\lambda_{\min}$ は条件数である)。高次元の非凸ランドスケープでは、高い損失域にある臨界点は主として狭義鞍点であり、勾配降下法は負の曲率をもつ固有ベクトルに沿ってそこから脱出する一方、活性化プラトーは現代的な活性化関数、正規化層、残差接続によって概ね防がれている。

任意の L 滑らかな目的関数に対しては、降下補題により $0 < \eta < 2/L$ である限り単調減少が保証される。逆に、学習率 η を固定すると $2/\eta$ より鋭い極小点はすべて不安定化し、フルバッチでのニューラルネットワークの訓練では、$\lambda_{\max}$ がこの**安定性の縁** $2/\eta$ 付近にとどまるまで漸進的鋭化が進む。曲率に基づく前処理行列によって勾配を再スケーリングすると有効条件数が減少し、対角前処理は $O(d)$ のメモリで座標軸に沿った悪条件性を解消するため、9.5 節で扱う座標ごとの適応的手法の動機となる。

9.1.7 演習問題

1. [code] **鞍点からの脱出時間**。鞍面 $f(x, y) = x^2 - \mu y^2$（$\mu > 0$、下向きの曲率 2μ）を考え、学習率 η の勾配降下法を $(1, \delta)$ から開始する。
   1. $|y| \geq 1$ に達するまでのステップ数が約 $\log(1/\delta)/(2\eta\mu)$（$\eta\mu$ が小さいとき）となることを示せ。この推定値を $\mu \in \{1, 0.01\}$、$\delta = 10^{-2}$、$\eta = 0.1$ について数値的に検証し、曲率 2μ をもつ正曲率の座標を δ 倍に縮小するのに要するステップ数と比較せよ。
   2. $\mu = 1$、$\delta = 0$ とし、各勾配評価に標準偏差 $\sigma \in \{10^{-3}, 10^{-1}\}$ をもつ平均 0 のガウスノイズを加えて脱出時間を測定せよ。得られた結果を、9.1.2 節における測度 0 の安定多様体の結果と整合的に説明せよ。
2. [code] **学習率と鋭さ**。2 層のスカラー線形ネットワークは、ab を入力 1 に対する出力として予測し、二乗損失 $\ell(a, b) = \tfrac{1}{2}(ab - 1)^2$ をとるため、双曲線 $ab = 1$ 上のすべての点が大域的最小点となる。
   1. 最小点 (a, b)（$ab = 1$ 上）におけるヘッセ行列の最大固有値が $\lambda_{\max} = a^2 + b^2$ であることを示せ。最も平坦な最小点はどれか。
   2. 200 ステップの勾配降下法を $(a, b) = (4, 0.1)$ から $\eta \in \{0.05, 0.1, 0.15, 0.2\}$ で実行し、最終的な鋭さ $\lambda_{\max}$ を安定性のしきい値 $2/\eta$ と比較せよ。
3. **置換対称な極小点と非凸性**。●$\theta^\star$ を、$d \geq 2$ 個の隠れユニットをもつ単一隠れ層 MLP の極小点とし、どの 2 つの隠れユニットも同一の入力重みを共有していないものとする。
   1. 隠れユニットを置換すると同一の損失をもつ $d!$ 個の異なるパラメータベクトルが得られることを示し、もし損失が凸であればそれらの平均もまた単一隠れユニットのネットワークと等価な大域的最小点になることを証明せよ。
   2. tanh 活性化関数を用いる場合について、各隠れユニットの符号反転対称性を特定し、その結果として生じる等価な極小点の総数を求めよ。
4. **対称ランダム行列の固有値**。●$\mathbf{M}$ を $k \times k$ の対称ランダム行列とし、その独立な成分 $M_{ij} = M_{ji}$（$i \leq j$）が対称分布 $p_{ij}(x) = p_{ij}(-x)$ から抽出されるものとする (Wigner 1958)。
   1. 一様に選ばれた固有値 λ（$\mathbf{M}$ の固有値）が $P(\lambda > 0) = P(\lambda < 0)$ を満たすことを証明し、区間 $[-2\sigma, 2\sigma]$ 上のウィグナーの半円則を用いて、$\mathbf{M}/\sqrt{k}$ の十分大きな 1 つの実現値における負の固有値の極限割合を求めよ。
   2. $k = 2$ かつ $M_{ii} \sim \mathcal{N}(0, 1)$、$M_{12} \sim \mathcal{N}(0, 1/2)$ のとき、両方の固有値が正となる厳密な確率を計算し、独立なコイン投げによる推定値 2^{-2} と比較せよ。
5. **多数ステップにわたる降下保証**。f を L 滑らかで下に $f^\star$ で有界な関数とし、勾配降下法を $\eta = 1/L$ で $\mathbf{x}_0$ から実行する。
   1. Equation 9.6 を T ステップにわたって足し合わせることで、$\min_{0 \leq t < T} \|\nabla f(\mathbf{x}_t)\|^2 \leq 2L\,(f(\mathbf{x}_0) - f^\star)/T$ を示せ。この上界に凸性は必要か。
   2. 谷の目的関数において初期点 $(-5, -2)$、$T = 100$ としたとき、この上界と反復点が実際に到達する二乗勾配ノルムとを比較せよ。
6. [code] **到達可能な極小点と安定性の縁より下での動作**。
   1. $f(x) = x\cos(0.15\pi x)$ の区間 $(-20, 30)$ における 4 つの極小点を求め、それぞれにおける f'' と安定性のしきい値 $2/f''$ を計算し、各極小点から 0.01 離れた初期値からしきい値の $0.9\times$ および $1.1\times$ の学習率で勾配降下法を試せ。

2. パラメータ数 25 のネットワーク（9.1.4 節）において、`gd_sharpness(0.5, P0)` を実行する前に最終的な鋭さを予測せよ。次に $\eta = 0.1$ で 50,000 ステップ（$2/\eta = 20$）試行し、$2/\eta$ に達する前にデータへ適合した実行がなぜ単調減少のまま（`rising = 0%`）となるのかを説明せよ。

7. [code] **回転下での対角前処理とニュートン法の収束次数。●**
   1. 谷を角度 θ だけ回転させ、$\mathbf{H}_\theta = \mathbf{R}_\theta \,\mathrm{diag}(0.2, 4)\mathbf{R}_\theta^\top$ とする。対角前処理を施したヘッセ行列 $\mathbf{D}^{-1/2}\mathbf{H}_\theta\mathbf{D}^{-1/2}$（ただし $\mathbf{D} = \mathrm{diag}(\mathbf{H}_\theta)$）の条件数を θ の関数として計算し、$\theta = 45^\circ$ において対角前処理が単なるスカラー学習率の変更に帰着し $\kappa = 20$ のまま変わらないことを数値的に検証せよ。
   2. 1 次元のニュートン法 $x_{t+1} = x_t - f'(x_t)/f''(x_t)$ を $f(x) = \cosh(0.5x)$ に対して $x_0 = 10$ から実行し、局所的な収束次数 p（$|x_{t+1}| \approx C|x_t|^p$、0 の近傍）を推定せよ。さらに、$p = 3$ となることを $x - 2\tanh(0.5x)$ のテイラー展開から導出せよ。

9.2 確率的勾配降下法とミニバッチ勾配降下法

9.1 節では、すべてのパラメータ更新に目的関数の厳密な勾配を用いたため、ステップごとに訓練セット全体にわたって損失を評価する必要があった。深層学習において、フルバッチ勾配降下法は計算コストの面から現実的ではない。ImageNet の 1.28 百万枚の画像で ResNet-50 を 90 エポックにわたり 256 枚のミニバッチで訓練すると 450,000 回の更新を要するが (Goyal et al. 2017)、もしフルバッチで 450,000 ステップ実行すればデータセット全体を 450,000 周走査することになり、5,000 倍もの計算量が必要になる。

その代わりに、**確率的勾配降下法**（stochastic gradient descent、SGD）はランダムにサンプリングした 1 つの例、あるいは b 個の例からなる小さな**ミニバッチ**から勾配を推定する（2.1.2.3 節）。この推定量は不偏であり、その分散は $1/b$ に比例して減少するが、サンプリングによるノイズは最適化の終盤の挙動を根本的に変えてしまう。すなわち、一定の学習率 η のもとでは、反復点はある 1 点の最小値へ収束せず、定常的な**ノイズ球**（noise ball）の内部で揺らぎ続ける。

本節では、勾配ノイズ（gradient noise）と不均一な曲率を組み合わせて考察する。9.1.1 節の 2 次元の谷を用いて、一定ステップサイズの SGD における各曲率方向の定常分散を導出し、そこから得られる過剰損失（excess loss）の予測を実際の回帰データセットで検証する。さらに、非復元サンプリング（sampling without replacement）とハードウェアの並列性がミニバッチの選択をどのように左右するかを調べ、学習率減衰（9.7 節）およびモメンタム（9.4 節）を導入する動機を明らかにする。

9.2.1 確率的勾配降下法

9.2.1.1 サンプリングされた勾配と谷

経験リスク（empirical risk）$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$ が n 個の訓練例に対して定義されるとき、厳密な勾配は例ごとの勾配の平均となる。

$$\nabla f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x}). \tag{9.10}$$

1 例ずつ用いる SGD は、各ステップでインデックス $i \in \{1, \dots, n\}$ を一様ランダムに引き、次のように更新する。

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f_i(\mathbf{x}). \tag{9.11}$$

$\mathbb{E}_i[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$ であるため、確率的勾配（stochastic gradient）は $\nabla f_i(\mathbf{x}) = \nabla f(\mathbf{x}) + \xi$ と分解できる。ここで、勾配ノイズ ξ の平均は 0 である。

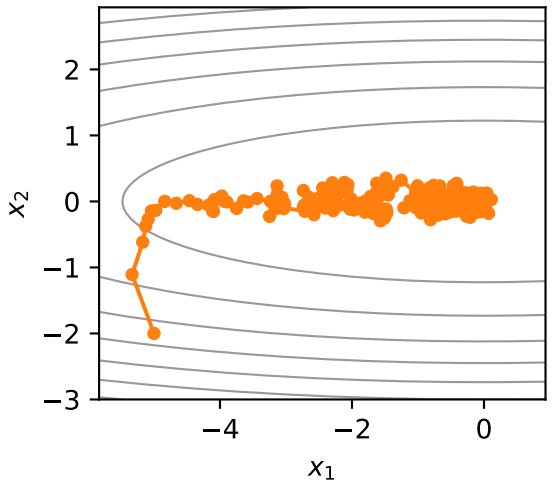
ξ が曲率とどのように相互作用するかを確かめるため、まず勾配の両座標に平均 0・分散 1 の独立なガウスノイズを加える sgd_2d を定義する。

```
def sgd_2d(x1, x2, s1, s2, f_grad, eta, rng):
    g1, g2 = f_grad(x1, x2)
    n1, n2 = rng.normal(size=2)  # Unit-variance gradient noise
    return x1 - eta * (g1 + n1), x2 - eta * (g2 + n2), 0, 0
```

続いて、300 ステップの sgd_2d を谷の関数 $f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$（9.1.1 節）上で、学習率 $\eta = 0.1$、初期点 $(-5, -2)$ から実行する。

```
rng = np.random.default_rng()
trace = d2l.train_2d(sgd_2d, d2l.valley_grad, steps=300, eta=0.1, rng=rng)
d2l.show_trace_2d(d2l.f_valley, trace)
```

```
step 300, x1: -0.708950, x2: 0.035603
```



最初の ~ 100 ステップでは、決定論的な勾配が支配的となり、反復点は原点へと近づく。原点付近に達すると、ノイズの振幅が一定のままであるのに対し $\|\nabla f(\mathbf{x})\|$ は小さくなるため、反復点は平坦な軸 x_1 に沿って引き伸ばされた細長い雲の中に閉じ込められる。

9.2.1.2 ノイズ球

この雲の大きさは解析的に計算できる。曲率 $\lambda > 0$ をもつ固有ベクトルの方向に沿って、平均 0 の独立な勾配ノイズ ξ_t（分散 σ^2）が加わるとき、SGD の 1 ステップは次のように時間発展する。

$$x_{t+1} = x_t - \eta\,(\lambda x_t + \xi_t) = (1 - \eta\lambda)\,x_t - \eta\,\xi_t. \tag{9.12}$$

両辺を二乗して期待値をとると（$\mathbb{E}[x_t \xi_t] = 0$ を用いる）、二次モーメント（second moment）の漸化式が得られる。

$$\mathbb{E}[x_{t+1}^2] = (1-\eta\lambda)^2\,\mathbb{E}[x_t^2] + \eta^2\sigma^2. \tag{9.13}$$

任意の安定なステップサイズ $0 < \eta < 2/\lambda$ に対して、平均 $\mathbb{E}[x_t]$ は 0 へ減衰し、二次モーメントは決定論的な縮小とノイズの注入が釣り合う定常分散へと収束する。すなわち $\mathbb{E}[x_\infty^2] = (1-\eta\lambda)^2\mathbb{E}[x_\infty^2] + \eta^2\sigma^2$ より、次式が成り立つ。

$$\mathbb{E}[x_\infty^2] = \frac{\eta\,\sigma^2}{\lambda\,(2-\eta\lambda)} \approx \frac{\eta\,\sigma^2}{2\lambda} \quad \text{for } \eta\lambda \ll 1. \tag{9.14}$$

Equation 9.14 は**ノイズ球**の性質を特徴づけている（有限時間における完全な軌跡の導出は 35.4.1 節を参照）。

1. **安定性の上限の維持：** $\eta \to 2/\lambda$ に近づくにつれて、分母は $(2-\eta\lambda) \to 0$ となり、分散は発散する。
2. **曲率に反比例するスケーリング：** 定常分散は曲率 λ に反比例する。谷の関数において $\eta = 0.1$、$\sigma^2 = 1$ とすると、定常分散は $0.1/(0.2 \times 1.98) \approx 0.2525$（平坦な座標 x_1 方向、std ≈ 0.50）および $0.1/(4 \times 1.6) \approx 0.0156$（急峻な座標 x_2 方向、std ≈ 0.125）となる。平坦な方向は二重の不利を被る。過渡期には収束が遅く、定常状態では大きく揺らぐからである。
3. **ステップサイズに対する線形スケーリング：** $\eta\lambda \ll 1$ のとき、学習率 η を半分にすると、あらゆる方向で定常分散も半分になる。これは 10,000 本の独立な軌跡をシミュレーションすることで確認できる。

```
def noise_ball(eta, lam, num_runs=10000, num_steps=2000):
    """Variance of SGD iterates on sum(lam * x**2 / 2) with unit noise."""
    rng = np.random.default_rng()
    x = np.zeros((num_runs, len(lam)))
    for _ in range(num_steps):
        x = x - eta * (lam * x + rng.normal(size=x.shape))
    return x.var(axis=0)

lam = np.array(d21.VALLEY_CURVATURES)  # Curvatures of the valley
for eta in (0.1, 0.05):
    print(f'eta={eta}: measured {noise_ball(eta, lam).round(4)}, '
          f'predicted {(eta / (lam * (2 - eta * lam))).round(4)}')
```

```
eta=0.1: measured [0.2498 0.0154], predicted [0.2525 0.0156]
eta=0.05: measured [0.1244 0.0069], predicted [0.1256 0.0069]
```

9.2.1.3 学習率の減衰

一定ステップサイズの SGD は根本的なトレードオフに直面する。学習率 η を小さくすればノイズ球は $\propto \eta$ で縮小するが、1 ステップあたりの縮小率 $\eta\lambda$ が下がるため初期の収束が遅くなる。ノイズフロアを取り除くには、時間とともに学習率 η_t を減衰させる必要がある。Robbins and Monro (1951) は、勾配の分散が有界である滑らかな強凸目的関数（strongly convex objective）に対し、ステップサイズが次の条件を満たせば SGD が厳密な最小点へ収束することを示した。

$$\sum_{t=1}^{\infty} \eta_t = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \eta_t^2 < \infty. \tag{9.15}$$

第 1 の条件は累積ステップ総量が無限大であることを保証し、任意の初期化から最適点へ到達できるようにする。一方、第 2 の条件は注入される累積ノイズ分散 $\propto \sum_t \eta_t^2$ が有限にとどまることを保証する。$\eta_t = c/(t+t_0)$（ただし $c > 1/(2\lambda_{\min})$）のような多項式スケジュールは両方の条件を満たし、期待二乗誤差を $\mathbb{E}[\|\mathbf{x}_t - \mathbf{x}^\star\|^2] = O(1/t)$ のレートで減少させる（35.4.2 節、演習問題 2）。ステップ数 T の固定された訓練予算のもとでは、実用的な深層学習は漸近的な $1/t$ スケジュールに代えて、コサイン、線形、あるいはウォームアップ・安定・減衰（warmup–stable–decay）スケジュールを用いる（9.7 節）。

9.2.2 ミニバッチ

9.2.2.1 勾配の分散とバッチサイズ

1 つの例を用いる代わりに、ミニバッチ SGD はランダムな部分集合 $\mathcal{B}_t$（b 個の例からなる）にわたって勾配を平均する。

$$\mathbf{g}_t = \frac{1}{b}\sum_{i \in \mathcal{B}_t} \nabla f_i(\mathbf{x}_t). \tag{9.16}$$

例ごとの勾配の共分散行列を $\Sigma(\mathbf{x}) = \mathbb{E}_i\big[(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x}))(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x}))^\top\big]$ とおく。そのトレース $\operatorname{tr}\Sigma = \mathbb{E}_i\|\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\|^2$ は、例ごとの勾配の分散の総和を表す。独立に復元サンプリングされた b 個の例に対して平均をとると、共分散は b で割られる。

$$\mathbb{E}\big\|\mathbf{g}_t - \nabla f(\mathbf{x}_t)\big\|^2 = \frac{\operatorname{tr}\Sigma}{b}. \tag{9.17}$$

$\sigma^2 \leftarrow \sigma^2/b$ を Equation 9.14 に代入すると、曲率 λ の方向に沿ったミニバッチ SGD の定常分散が得られる。

$$\mathbb{E}\big[x_\infty^2\big] = \frac{\eta\,\sigma^2}{b\,\lambda\,(2 - \eta\lambda)} \approx \frac{\eta}{b}\cdot\frac{\sigma^2}{2\lambda} \quad \text{for } \eta\lambda \ll 1. \tag{9.18}$$

ステップサイズが小さい領域 $\eta\lambda \ll 1$ では、学習率とバッチサイズはそれらの比 η/b のみを通じて定常的なノイズ球を支配する。すなわち、バッチサイズ b を 2 倍にすることは、ノイズフロアに対して学習率 η を半分にするのと同じ効果をもつ。

9.2.2.2 データにおける過剰損失

ノイズ球内部での揺らぎは、訓練損失をその最小値 $f(\mathbf{x}^\star)$ からどれだけ押し上げるだろうか。k 番目の固有ベクトル（ヘッセ行列 $\mathbf{H}$ の固有値 λ_k、例ごとの勾配の分散 σ_k^2 をもつ方向）に沿った分散 $v_k \approx \frac{\eta\sigma_k^2}{2\lambda_k b}$ の定常的な変位は、二次損失の期待値に $\frac{1}{2}\lambda_k v_k$ だけ寄与する。

$$\frac{1}{2}\lambda_k\left(\frac{\eta\,\sigma_k^2}{2\lambda_k b}\right) = \frac{\eta\,\sigma_k^2}{4b}. \tag{9.19}$$

ここで曲率 λ_k が約分されて消えることに注目しよう。平坦な方向ほど広い半径にわたって揺らぐが（$\propto 1/\lambda_k$）、平坦な方向への単位変位が損失を押し上げる量は小さい（$\propto \lambda_k$）ためである。

直交する d 個すべての固有ベクトルにわたって和をとると（$\sum_{k=1}^{d} \sigma_k^2 = \operatorname{tr}\Sigma$）、定常状態の**過剰損失**が得られる。

$$\mathbb{E}\big[f(\mathbf{x}_\infty)\big] - f(\mathbf{x}^\star) \approx \frac{\eta \operatorname{tr}\Sigma(\mathbf{x}^\star)}{4b}. \tag{9.20}$$

Equation 9.20 を NASA 翼型自己騒音（airfoil self-noise）回帰データセット（$n = 1{,}500$ 件の風洞測定値からなり、標準化された 5 個の特徴量と標準化された 1 個の音圧ターゲットをもつ）で検証する。

```
d2l.DATA_HUB['airfoil'] = (d2l.DATA_URL + 'airfoil_self_noise.dat',
                           '76e5be1548fd8222e5074cf0faae75edff8cf93f')


def get_airfoil(batch_size=10, n=1500):
    """Load the standardized airfoil self-noise data as minibatches."""
    data = np.genfromtxt(d2l.download('airfoil'),
                         dtype=np.float32, delimiter='\t')
    data = (data - data.mean(axis=0)) / data.std(axis=0)
    data_iter = d2l.load_array(
        (jnp.array(data[:n, :-1]), jnp.array(data[:n, -1])),
        batch_size, is_train=True)
    return data_iter, data.shape[1]-1
```

二乗損失を用いた線形回帰に対し（バイアスは定数特徴量に対する 6 番目の重みとして表す）、厳密なヘッセ行列 $\mathbf{H} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$、閉形式の最小点 $\mathbf{w}^\star$、最大曲率 $L = \lambda_{\max}(\mathbf{H}) \approx 2.10$、および例ごとの勾配の分散 $\operatorname{tr}\Sigma(\mathbf{w}^\star) \approx 3.24$ を計算する。

```
data_iter, num_features = get_airfoil(batch_size=1500)
X, y = map(d2l.numpy, next(iter(data_iter)))  # All 1,500 examples
X = np.hstack([X, np.ones((len(X), 1), np.float32)])  # Bias as a 6th weight
H = X.T @ X / len(X)  # Hessian of the loss
w_star = np.linalg.lstsq(X, y)[0]  # The minimizer, in closed form
G = X * (X @ w_star - y)[:, None]  # Per-example gradients at w_star
tr_Sigma = (G ** 2).sum(axis=1).mean()
print(f'L = {np.linalg.eigvalsh(H).max():.2f}, tr Sigma = {tr_Sigma:.2f}')
```

```
L = 2.10, tr Sigma = 3.24
```

この予測に対して経験的な過剰損失を測定するため、`sgd_runs` は復元サンプリングによるベクトル化されたミニバッチ SGD の軌跡を `num_runs=400` 本シミュレーションし、`excess_loss` は $\frac{1}{2}(\mathbf{w} - \mathbf{w}^\star)^\top \mathbf{H} (\mathbf{w} - \mathbf{w}^\star)$ を評価する。

```
def sgd_runs(X, y, w, eta, b, num_steps, rng, num_runs=400):
    """Run minibatch SGD on the squared loss num_runs times from w, drawing
    examples with replacement; return every iterate, (steps, runs, d)."""
    W, trace = np.tile(w, (num_runs, 1)), []
    for idx in rng.integers(0, len(X), (num_steps, num_runs, b)):
        r = np.einsum('rbd,rd->rb', X[idx], W) - y[idx]  # Residuals
        W = W - eta / b * np.einsum('rbd,rb->rd', X[idx], r)
```

```
            trace.append(W)
        return np.array(trace)

    def excess_loss(W, w_star, H):
        """f(w) - f(w_star) of the quadratic loss, for every w in the last axis."""
        D = W - w_star
        return 0.5 * np.einsum('...d,de,...e->...', D, H, D)
```

初期点を $\mathbf{w}^\star$、バッチサイズを $b = 10$ として、3 エポック目（150 ステップ分）にわたる `excess_loss` の平均を $\eta \in \{0.05, 0.025\}$ のそれぞれで求め、$\eta \operatorname{tr}\Sigma/(4b)$ と比較する。

```
rng = np.random.default_rng()
for eta in (0.05, 0.025):  # b = 10, three epochs of 150 steps from w_star
    trace = sgd_runs(X, y, w_star, eta, 10, 450, rng)
    measured = excess_loss(trace[-150:], w_star, H).mean()  # The last epoch
    print(f'eta={eta}: excess loss {measured:.4f}, '
          f'predicted {eta * tr_Sigma / (4 * 10):.4f}')
```

```
eta=0.05: excess loss 0.0043, predicted 0.0041
eta=0.025: excess loss 0.0021, predicted 0.0020
```

$b = 10$ において、実測された過剰損失は、約 0.004（$\eta = 0.05$ のとき）および 0.002（$\eta = 0.025$ のとき）となり、予測値 $\eta \operatorname{tr}\Sigma/(4b)$ とよく一致している。ここで、モデルがすべての訓練例に厳密に適合できるだけの表現力をもち、$\nabla f_i(\mathbf{x}^\star) = \mathbf{0}$ がすべての i について成り立つ**補間レジーム**（interpolating regime）では何が起きるかに注目しよう。この極限では $\operatorname{tr}\Sigma(\mathbf{x}^\star) = 0$ となり、反復点が $\mathbf{x}^\star$ に近づくにつれてノイズ球は消失し、一定ステップサイズの SGD は厳密な最小点へと幾何級数的に収束する。多数のエポックにわたって訓練される過剰パラメータ化された分類器はこの補間レジームに近づく一方、巨大なコーパスを 1 パスだけ走査して訓練される大規模言語モデルは、$\operatorname{tr}\Sigma(\mathbf{x}^\star) > 0$ となるノイズ制限レジーム（noise-limited regime）にとどまる。

9.2.2.3 非復元サンプリング

実際には、データローダはミニバッチを独立に復元サンプリングするのではなく、各エポックの開始時にデータセットをランダムに並べ替え、互いに素なサイズ b のミニバッチへと分割する。b 個の異なる例をサイズ n の有限母集団から非復元サンプリングすると、例ごとの勾配誤差の間に負の相関が生じる（$\sum_{i=1}^{n}(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})) = \mathbf{0}$ であるため）。その結果、ミニバッチ勾配の分散 Equation 9.17 には有限母集団修正係数（finite-population correction factor）が掛かる（演習問題 3）。

$$\mathbb{E}\left\|\mathbf{g}_t - \nabla f(\mathbf{x}_t)\right\|^2 = \frac{\operatorname{tr}\Sigma}{b} \cdot \frac{n-b}{n-1}. \tag{9.21}$$

$b \ll n$ のとき（10 個を 1,500 個から選ぶ場合）、この係数はほぼ 1 に等しく（0.994）、$b = n$ のときはフルバッチ勾配降下法で要請されるとおり分散が 0 になる。さらに 1 エポック全体を通してみると、ランダムな再シャッフル（random reshuffling）によってエポック内の n/b ステップにわたるミニバッチ勾配誤差が 1 次のオーダーで打ち消し合うため、学習率が小さいときには復元サンプリングよりも定常誤差が小さくなる (Mishchenko et al. 2020)。

9.2.2.4 バッチサイズとハードウェアのスループット

$\eta\lambda \ll 1$ であり、かつ η/b を一定に保つとき、サイズ b のミニバッチに対して学習率 η で 1 ステップ更新すると、期待誤差は $(1-\eta\lambda)$ 倍に縮小し、1 例ずつ b ステップを学習率 η/b で進めた場合と同じノイズ球に到達する。これは $(1-\eta\lambda/b)^b \approx 1-\eta\lambda$ が成り立つためである。例ごとの勾配評価の回数だけで数えるなら、この線形レジームの範囲内で b を大きくしても、学習は速くも遅くもならない。

ミニバッチを不可欠なものにしているのは、**ハードウェアによる償却**（hardware amortization）である。各パラメータ更新には b に依存しない一定の Python ディスパッチおよびカーネル起動のオーバーヘッドが生じる上、ベクトルプロセッサ（CPU や GPU）はモデルの重みをメモリから 1 度読み出すだけで b 個の例を並列に評価できる（第 13 章）。翼型データセットにおいて、1 エポックあたりの実経過時間と処理した例数に対する損失の両方をベンチマークするため、`train_sgd` は $\mathbf{w}=\mathbf{0}$ から線形回帰に対してミニバッチ SGD を実行し、更新ごとのデータセット全体の損失と 1 エポックあたりの秒数を返す。

```
@jax.jit
def sgd_step(w, X, y, lr):
    """One minibatch SGD step on the squared loss of linear regression."""
    loss = lambda w: jnp.mean((X @ w[:-1] + w[-1] - y) ** 2) / 2
    return w - lr * jax.grad(loss)(w)

def train_sgd(lr, batch_size, num_epochs=2):
    """Minibatch SGD on linear regression from w = 0; return the loss after
    every update and the seconds per epoch."""
    data_iter, num_features = get_airfoil(batch_size)
    X, y = (jnp.concatenate(t) for t in zip(*data_iter))  # The whole dataset
    ws = [jnp.zeros(num_features + 1)]
    sgd_step(ws[0], X[:batch_size], y[:batch_size], lr)  # Compile first

    def epoch():
        for X_b, y_b in data_iter:
            ws.append(sgd_step(ws[-1], X_b, y_b, lr))
        return ws[-1]  # Benchmark waits for it

    seconds = d2l.Benchmark(epoch, warmup=0, repeats=num_epochs).time
    W = jnp.stack(ws[1:])  # The weights after every update, (updates, 6)
    losses = ((X @ W[:, :-1].T + W[:, -1] - y[:, None]) ** 2).mean(0) / 2
    return losses, seconds
```

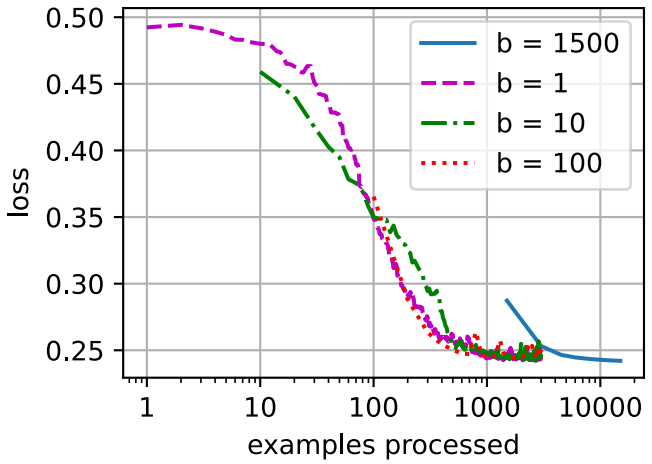
それでは、`train_sgd` を $b \in \{1, 10, 100\}$ に対して一定の比 $\eta/b = 0.005$ で実行し、あわせてフルバッチ勾配降下法（$b = 1{,}500$、$\eta = 0.9 < 2/L \approx 0.95$）も実行しよう。

```
runs = {}
for b, lr, num_epochs in ((1500, 0.9, 10), (1, 0.005, 2), (10, 0.05, 2),
                          (100, 0.5, 2)):  # lr / b = 0.005 for SGD
    runs[b], seconds = train_sgd(lr, b, num_epochs)
    updates = 1500 // b
    print(f'b={b:4d}: {updates:4d} updates/epoch, {seconds:.3f} s/epoch '
          f'({1000 * seconds / updates:.1f} ms/update), '
          f'last-epoch loss {float(runs[b][-updates:].mean()):.3f}')
```

```
b=1500:    1 updates/epoch, 0.002 s/epoch (2.0 ms/update), last-epoch loss 0.242
b=   1: 1500 updates/epoch, 0.930 s/epoch (0.6 ms/update), last-epoch loss 0.245
b=  10:  150 updates/epoch, 0.111 s/epoch (0.7 ms/update), last-epoch loss 0.247
b= 100:   15 updates/epoch, 0.014 s/epoch (0.9 ms/update), last-epoch loss 0.248
```

処理した総例数（$b \times$ updates）に対して訓練損失をプロットすると、4 つのバッチサイズすべてのサンプル効率を重ねて比較できる。

```
d2l.plot([b * np.arange(1, len(losses) + 1) for b, losses in runs.items()],
         [d2l.numpy(losses) for losses in runs.values()],
         'examples processed', 'loss', legend=[f'b = {b}' for b in runs],
         xscale='log')
```



$b = 100$ 個の例に対する 1 回の更新は、$b = 1$ での更新とほぼ同じ実経過時間で完了するため、$b = 100$ の 1 エポックは約 50–100× 高速になりながら、処理した例数あたりではまったく同じ損失曲線をたどる。同時に、線形スケーリング $\eta \propto b$ がどこで破綻するかにも注目しよう。$\eta/b = 0.005$ を保ったまま $b = 100$ にすると学習率は $\eta = 0.5$ まで上がり、$\eta L = 0.5 \times 2.10 = 1.05$ となって 2 と比べて十分に小さいとは言えなくなる（その結果、Equation 9.14 の分母の逆数 $(2 - \eta\lambda)^{-1}$ が増大する）。さらに $b \approx 190$ を超えるバッチサイズでは、安定性の上限 $\eta < 2/L \approx 0.95$ に違反してしまう。この臨界バッチサイズ（critical batch size）の遷移については、9.8 節で体系的に解析する。

9.2.3 まとめ

ミニバッチ SGD は b 個の例から勾配を推定し、サンプリングによる分散を $\mathrm{tr}(\Sigma)/b$（非復元サンプリングではこれに $(n-b)/(n-1)$ を掛けた値）まで低減しつつ、ステップごとのディスパッチおよびメモリ帯域幅のコストをミニバッチ全体で償却する。一定の学習率 η のもとでは、反復点は定常的な**ノイズ球**へと収束し、曲率 λ の方向に沿った座標分散は $\frac{\eta\sigma^2}{b\lambda(2-\eta\lambda)} \approx \frac{\eta\sigma^2}{2\lambda b}$ となって最も平坦な方向で最も広くなる。全モードにわたって二次損失への寄与を足し合わせると λ が相殺され、定常状態の期待**過剰損失** $\mathbb{E}[f(\mathbf{x}_\infty)] - f(\mathbf{x}^\star) \approx \frac{\eta\,\mathrm{tr}\,\Sigma(\mathbf{x}^\star)}{4b}$ が残る。この過剰損失が 0 になるのは、最適点においてモデルがすべての訓練例を補間する場合に限られる。$\eta L \ll 1$ が成り立つ限り、1 例あたりの過渡的な進捗と定常ノイズフロアはいずれも η と b にその比 η/b のみを通じて依存するが、b を $\eta \propto b$ に従って増やし続けると、やがて決定論的な曲率の上限 $\eta < 2/L$ に近づきこの性質は成り立たなくなる。

9.2.4 演習問題

1. **ランダムなデータセットとしてのノイズ。** 分散 1 のガウス勾配ノイズを伴う `sgd_2d` を $f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$ 上で実行することは、$f(\mathbf{x}, \mathbf{w}) = 0.1(x_1 - w_1)^2 + 2(x_2 - w_2)^2$（ただし $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{C})$）上で 1 例ずつの SGD を実行することと等価であることを示し、共分散 $\mathbf{C}$ を求めよ。

2. [code] **$1/t$ スケジュールの定数。**●曲率 λ の 1 次元二次関数に対し、分散 σ^2 の勾配ノイズと学習率 $\eta_t = c/(t + t_0)$ のもとで以下を行え。

   1. $e_t = \mathbb{E}[x_t^2]$ に対する厳密な漸化式を書き下し、漸近的な仮定形 $e_t \approx A/t$ を代入することで、$A = c^2\sigma^2/(2c\lambda - 1)$ が（$c > 1/(2\lambda)$ のとき常に）成り立つことを示せ。どの c の値が A を最小化するか。
   2. e_t の決定論的な漸化式を $\lambda = 0.2, \sigma = 1$、初期値 $e_0 = 25$、$t_0 = 20$、$c \in \{1, 2.5, 5, 10\}$ として $t = 10^6$ まで反復計算し、両対数軸上で $t\,e_t$ を t に対してプロットせよ。さらに、単一の c では $\lambda_1 = 0.2$ と $\lambda_2 = 4$ の両方に対して同時に最適になり得ない理由を説明せよ。

3. [code] **有限母集団係数と再シャッフル。**

   1. $\mathbf{e}_i = \nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})$ とおく。$\sum_{i=1}^{n} \mathbf{e}_i = \mathbf{0}$ を用いて $\mathbb{E}_{i \neq j}[\mathbf{e}_i^\top \mathbf{e}_j] = -\operatorname{tr}\Sigma/(n-1)$ を示し、Equation 9.21 を導出せよ。さらに翼型データセットの例ごとの勾配 `G` を用い、$b \in \{10, 100, 500, 1000\}$ を復元サンプリングおよび非復元サンプリングした場合について、この公式を実験的に検証せよ。
   2. $n = 100$ 個のベクトル $\mathbf{w}_i$ を演習問題 1 の分布から生成し、復元サンプリングとエポックごとの再シャッフルによる 20 エポックの SGD の軌跡を $\eta \in \{0.1, 0.01\}$ で比較せよ。

4. [code] **訓練中の勾配の分散。**●

   1. 翼型の線形モデルにおいて、$\operatorname{tr}\Sigma$ の値を $\mathbf{w} = \mathbf{0}$ と $\mathbf{w} = \mathbf{w}^\star$ で比較せよ。
   2. 2 層の ReLU MLP（隠れユニット数 64）を翼型データで 10 エポック（$b = 10, \eta = 0.05$）訓練し、$\operatorname{tr}\Sigma$（`W1`、`b1`、`W2`、`b2` ごとの内訳）を初期化時点と訓練後とで比較せよ。

5. [code] **バッチサイズと学習率のスケーリング。**●翼型の線形回帰モデルにおいて $\eta = 0.05$（$b = 10$ のとき）を基準に固定し、$b \in \{1, 10, 100, 1500\}$ を線形スケーリング（$\eta \propto b$）と平方根スケーリング（$\eta \propto \sqrt{b}$）の両方で試せ。$L = 2.10$ を用いてどの設定が発散するかを説明し、さらに `H` の全 6 個の固有ベクトルにわたる厳密な定常過剰損失を Equation 9.14 から計算して、$b = 100, \eta = 0.5$ のフロアが $b = 10, \eta = 0.05$ よりもわずかに高くなる理由を説明せよ。

9.3 2 つのテストモデル

2 次元の二次関数の谷のような解析的モデルは、曲率と勾配ノイズが単純な更新則をどのように支配するかを説明してくれるが、現代のオプティマイザを評価するには実際のニューラルネットワーク上での経験的な測定が欠かせない。なぜ AdamW は Transformer では大差で SGD を上回る一方、畳み込みネットワークでははるかに小さな優位性しか示さないのだろうか。こうした問いに答えるには、統制された実験プロトコルのもとで実際のモデルを訓練する必要がある。

確率的な訓練の実行結果はランダム初期化やミニバッチの抽出ごとに変動するため、2 つのオプティマイザ設定のあいだの差が意味を持つのは、それが単一の設定における実行間のばらつきを上回る場合に限られる。本節では、本章の残りの部分を通して使用する 2 つのベンチマークアーキテクチャ、すなわちバイトレベルの Transformer 言語モデル `TinyLM`（パラメータ数 470,784）

と畳み込み分類器 FashionCNN（パラメータ数 421,642）を定義し、あわせて共通の訓練ループ、学習率スイープ用ハーネス、ベースラインとなる SGD の測定結果、および有意性のしきい値を導入する。

```
import os
import pandas as pd
import textwrap
```

9.3.1 言語モデル TinyLM

TinyLM は、次の UTF-8 バイト（$V = 256$）を連続する 64 バイトのコンテキストウィンドウから予測する自己回帰型のバイトレベル言語モデルである（8.2 節）。標準的な 2,000 ステップの実行では、64 個のウィンドウからなるミニバッチを用いて $2{,}000 \times 64 \times 64 \approx 8.2 \times 10^6$ 個のターゲットバイトを処理する。訓練例の繰り返し（最適化の速度と暗記を混同させてしまう）を避けるため、訓練には WikiText-103 の第 1 シャード（Wikipedia 記事 $\sim 2.7 \times 10^8$ UTF-8 バイト分）を用いる (Merity et al. 2016)。各訓練ミニバッチはランダムなバイトオフセットから 64 個のウィンドウをサンプリングするため、2,000 ステップの実行で目にするのはコーパスのわずか $\sim 3\%$ にとどまり、すべての訓練ステップで新しいテキストを評価することになる。検証パープレキシティは、1.1×10^6 バイトの検証分割から切り出した連続する非重複ウィンドウ上で測定する。

```
class WikiText103(d2l.DataModule):
    """WikiText-103 as UTF-8 bytes, cut into windows of num_steps bytes."""
    def __init__(self, batch_size, num_steps, num_train=None, num_val=None):
        super().__init__()
        self.save_hyperparameters()
        self.vocab = d2l.BPETokenizer(vocab_size=256, specials=())  # Bytes
        self.train = self._read('train-00000-of-00002',
                                '3cbc792dfc0eeedf8d7333acdc15a9f536032d9a')
        self.val = self._read('validation-00000-of-00001',
                              '49d15019b0476afc752bd5f010b5a2aab2de89ca')
        self.train, self.val = self.train[:num_train], self.val[:num_val]

    def _read(self, split, sha1):
        url = ('https://huggingface.co/datasets/Salesforce/wikitext/resolve/'
               f'main/wikitext-103-raw-v1/{split}.parquet')
        fname = d2l.download(url, os.path.join(self.root, 'wikitext-103'),
                             sha1)
        lines = pd.read_parquet(fname)['text']  # Each line ends with '\n'
        return np.frombuffer(b''.join(s.encode() for s in lines), np.uint8)

    def windows(self, corpus, starts):
        """Inputs and next-byte targets of the windows at `starts`."""
        idx = np.asarray(starts)[:, None] + np.arange(self.num_steps + 1)
        array = corpus[idx].astype(np.int32)
        return array[:, :-1], array[:, 1:]

    def train_dataloader(self):  # A pass of minibatches at random offsets
        n = len(self.train) - self.num_steps
        rng = np.random.default_rng(d2l.get_seed())
        for _ in range(n // (self.batch_size * self.num_steps)):
```

```
            yield self.windows(self.train,
                               rng.integers(n, size=self.batch_size))

    def val_dataloader(self):  # Consecutive, non-overlapping windows
        starts = np.arange(0, len(self.val) - self.num_steps, self.num_steps)
        for i in range(0, len(starts), self.batch_size):
            yield self.windows(self.val, starts[i:i + self.batch_size])
```

`WikiText103(batch_size=64, num_steps=64)` をインスタンス化すると、訓練シャードと検証シャードがダウンロードされ、コーパス先頭の 300 バイトがデコードされる。

```
data = WikiText103(batch_size=64, num_steps=64)
print(f'{len(data.train):,} training bytes, '
      f'{len(data.val):,} validation bytes')
text = data.vocab.decode(data.train[:300])
print('\n'.join(textwrap.fill(line, 72) for line in text.splitlines()))
```

```
269,630,498 training bytes, 1,144,248 validation bytes
 = Valkyria Chronicles III =
 Senjō no Valkyria 3 : Unrecorded Chronicles ( Japanese : 戦場のヴァルキュリア 3 ,
lit . Valkyria of the Battlefield 3 ) , commonly referred to as Valkyria
Chronicles III outside Japan , is a tactical role @-@ playing video game
developed by Sega and Media.Vis
```

9.3.1.1 アーキテクチャとパラメータグループ

`TinyLM` は 2 ブロックのデコーダのみの Transformer（第 11 章）であり、埋め込み幅は $d_{\text{model}} = 128$、各ブロックは 2 個の因果アテンション（causal attention）ヘッドと、隠れ層幅 $4d_{\text{model}} = 512$ の 2 層 GELU MLP を持つ。各サブ層の前には LayerNorm（7.3.4.1 節）が置かれて残差接続で囲まれ、最後に最終 LayerNorm と 256 個のバイトロジットへ射影する線形出力ヘッドが続く。埋め込みの各行は $\mathcal{N}(0, 1)$ から初期化され、すべての線形層の重みとバイアスは $[-1/\sqrt{\text{fan-in}}, 1/\sqrt{\text{fan-in}}]$ から一様に抽出される。

```
class TinyLM(nnx.Module):
    """A small decoder-only transformer language model."""
    def __init__(self, vocab_size, d_model=128, num_heads=2, num_blks=2,
                 max_len=64, *, rngs):
        self.num_heads = num_heads
        def linear(d_in, d_out):
            a = 1 / math.sqrt(d_in)  # Weights and biases ~ U(-a, a)
            init = lambda key, shape, dtype: jax.random.uniform(
                key, shape, dtype, -a, a)
            return nnx.Linear(d_in, d_out, kernel_init=init, bias_init=init,
                              rngs=rngs)
        def embed(n):  # Embedding rows ~ N(0, I)
            return d2l.Embed(n, d_model, rngs=rngs,
                             embedding_init=nnx.initializers.normal(1.0))
        def norm():
            return nnx.LayerNorm(d_model, epsilon=1e-5, rngs=rngs)
        self.token_emb, self.pos_emb = embed(vocab_size), embed(max_len)
```

```
        self.blks = nnx.List([nnx.Dict(
            norm1=norm(), qkv=linear(d_model, 3 * d_model),
            proj=linear(d_model, d_model), norm2=norm(),
            mlp1=linear(d_model, 4 * d_model),
            mlp2=linear(4 * d_model, d_model))
            for _ in range(num_blks)])
        self.norm = norm()
        self.head = linear(d_model, vocab_size)

    def attention(self, blk, X):
        B, T, D = X.shape
        q, k, v = jnp.split(blk['qkv'](X), 3, axis=-1)
        q, k, v = (u.reshape(B, T, self.num_heads, -1) for u in (q, k, v))
        Y = jax.nn.dot_product_attention(q, k, v, is_causal=True)
        return blk['proj'](Y.reshape(B, T, D))

    def __call__(self, X):
        H = self.token_emb(X) + self.pos_emb(jnp.arange(X.shape[1]))
        for blk in self.blks:
            H = H + self.attention(blk, blk['norm1'](H))
            H = H + blk['mlp2'](jax.nn.gelu(blk['mlp1'](blk['norm2'](H)),
                                            approximate=False))
        return self.head(self.norm(H))
```

オプティマイザの観点からは、`TinyLM` の 470,784 個のパラメータは構造的に異なる 3 つのグループに分かれ、後の節ではそれぞれ異なる扱いを受ける（たとえば 9.5.3 節の分離型重み減衰（decoupled weight decay）や 9.6 節の行列直交化など）。

- **埋め込み**（9%）：ルックアップテーブル（`token_emb`、`pos_emb`）であり、その各行はトークン頻度に比例したスパースな勾配を受け取る。
- **重み行列**（90%）：密な 2 次元射影行列（各ブロックの `qkv`、`proj`、2 つの MLP 層、および `head`）である。
- **1 次元ベクトル**（$< 1\%$）：線形層のバイアスおよび LayerNorm のスケール・シフトベクトルである。

```
# Flax's default initializers differ, so TinyLM passes the ones described
# above (and approximate=False for the exact GELU); d2l.Embed computes
# the embedding gradient as one matrix product, far faster than XLA's
# default on frequent bytes such as the space
model = TinyLM(vocab_size=len(data.vocab), rngs=nnx.Rngs(d2l.get_key()))
kinds = {'embeddings': 0, 'matrices': 0, 'vectors': 0}
print(f'{"parameter":<22} {"shape":<14} {"count":>7}')
for path, p in nnx.to_flat_state(nnx.state(model, nnx.Param)):
    name = '.'.join(str(k) for k in path)
    kind = ('embeddings' if 'emb' in name else
            'matrices' if p.ndim == 2 else 'vectors')
    kinds[kind] += p.size
    print(f'{name:<22} {str(tuple(p.shape)):<14} {p.size:>7}')
for kind, count in kinds.items():
    print(f'{kind:>37} {count:>7}')
print(f'{"total":>37} {sum(kinds.values()):>7}')
```

```
parameter              shape           count
blks.0.mlp1.bias       (512,)            512
blks.0.mlp1.kernel     (128, 512)      65536
blks.0.mlp2.bias       (128,)            128
blks.0.mlp2.kernel     (512, 128)      65536
blks.0.norm1.bias      (128,)            128
blks.0.norm1.scale     (128,)            128
blks.0.norm2.bias      (128,)            128
blks.0.norm2.scale     (128,)            128
blks.0.proj.bias       (128,)            128
blks.0.proj.kernel     (128, 128)      16384
blks.0.qkv.bias        (384,)            384
blks.0.qkv.kernel      (128, 384)      49152
blks.1.mlp1.bias       (512,)            512
blks.1.mlp1.kernel     (128, 512)      65536
blks.1.mlp2.bias       (128,)            128
blks.1.mlp2.kernel     (512, 128)      65536
blks.1.norm1.bias      (128,)            128
blks.1.norm1.scale     (128,)            128
blks.1.norm2.bias      (128,)            128
blks.1.norm2.scale     (128,)            128
blks.1.proj.bias       (128,)            128
blks.1.proj.kernel     (128, 128)      16384
blks.1.qkv.bias        (384,)            384
blks.1.qkv.kernel      (128, 384)      49152
head.bias              (256,)            256
head.kernel            (128, 256)      32768
norm.bias              (128,)            128
norm.scale             (128,)            128
pos_emb.embedding      (64, 128)        8192
token_emb.embedding    (256, 128)      32768
                           embeddings  40960
                             matrices 425984
                              vectors   3840
                                total 470784
```

9.3.2 画像分類器 FashionCNN

Transformer におけるオプティマイザのランキングは畳み込みネットワークにおけるランキングと大きく異なるため、2 つ目のベンチマークとして 28×28 の Fashion-MNIST 画像（3.2 節）に対する分類器 `FashionCNN` を導入する。2 段の 3×3 畳み込み + ReLU + 2×2 最大プーリングのステージ（32 チャネルと 64 チャネル）によって空間解像度を $64 \times 7 \times 7$ に縮小し、その後に 2 つの線形層（$3{,}136 \to 128 \to 10$）を続ける。すべての層で一様分布 $\pm 1/\sqrt{\text{fan-in}}$ による初期化を用い、パラメータ数は合計 421,642 となる。

```
class FashionCNN(nnx.Module):
    """A compact CNN for Fashion-MNIST: two conv-pool stages, two linear."""
    def __init__(self, *, rngs):
        def init(fan_in):  # Weights and biases ~ U(-a, a), a = 1/sqrt(fan_in)
            a = 1 / math.sqrt(fan_in)
            f = lambda key, shape, dtype: jax.random.uniform(
                key, shape, dtype, -a, a)
```

```
            return dict(kernel_init=f, bias_init=f, rngs=rngs)
        self.conv1 = nnx.Conv(1, 32, kernel_size=(3, 3), **init(9))
        self.conv2 = nnx.Conv(32, 64, kernel_size=(3, 3), **init(9 * 32))
        self.fc1 = nnx.Linear(64 * 7 * 7, 128, **init(64 * 7 * 7))
        self.fc2 = nnx.Linear(128, 10, **init(128))

    def __call__(self, X):
        X = nnx.max_pool(nnx.relu(self.conv1(X)), window_shape=(2, 2),
                         strides=(2, 2))
        X = nnx.max_pool(nnx.relu(self.conv2(X)), window_shape=(2, 2),
                         strides=(2, 2))
        return self.fc2(nnx.relu(self.fc1(X.reshape(X.shape[0], -1))))
```

d2l.num_params を用いて、FashionCNN が 421,642 個の訓練可能なパラメータを持つことを確認する。

```
# Flax's default initializers differ, so FashionCNN passes U(-a, a)
# explicitly; images arrive as (batch, height, width, channel)
print(f'{d2l.num_params(FashionCNN(rngs=nnx.Rngs(d2l.get_key()))):,} '
      'parameters')
```

```
421,642 parameters
```

バッチサイズ 256 の場合、2,000 ステップの実行では ∼ 8.5 周にわたって 60,000 枚の Fashion-MNIST 訓練画像を走査することになるため、evaluate_accuracy を用いて訓練損失とあわせてホールドアウトのテスト正解率も報告する。

```
fashion = d2l.FashionMNIST(batch_size=256)

@nnx.jit
def num_correct(model, X, y):
    """Number of examples in (X, y) that model classifies correctly."""
    return (model(X).argmax(axis=1) == y).sum()

def evaluate_accuracy(model, batches):
    """Fraction of the examples in batches that model classifies correctly."""
    # A view with dropout off and batch norm on running statistics
    model = nnx.view(model, deterministic=True, use_running_average=True,
                     raise_if_not_found=False)
    correct, total = 0, 0
    for X, y in batches:
        correct += num_correct(model, jnp.asarray(X), jnp.asarray(y))
        total += len(y)
    return float(correct) / total
```

9.3.3 訓練ハーネスと評価補助関数

どちらのモデルもミニバッチをロジットへ写像し、一定のステップ予算にわたって平均交差エントロピー損失を最小化する。Table 9.2 に、d2l に保存される訓練ループと評価ユーティリティをまとめる。

Table 9.2: d21 に保存される共通の訓練・評価関数。

関数	目的
train_steps	num_steps ステップ（または stop(step) が成り立つまで）訓練し、ステップごとの交差エントロピー損失を返す
lm_perplexity	バッチのイテレータ上でトークンごとのパープレキシティ exp(mean cross-entropy) を計算する（8.3 節）
evaluate_accuracy	バッチのイテレータ上で分類正解率を計算する
final_loss	直近 $k = 100$ ステップの損失の平均（発散した場合は inf）
smooth	プロット用の連続する $k = 25$ 個の損失ウィンドウにわたるブロック平均
steps_to_reach	25 ステップ平滑化損失が target 以下に下がる最初のステップ（到達しない場合は None）
lr_sweep	同一の初期重みから学習率のリストにわたって訓練し、損失曲線と評価スコアを返す

まず train_steps を実装する。この関数は、num_steps 個のミニバッチにわたって（パディングラベル -100 を無視しつつ）、あるいは任意の述語 stop(step) が成り立つまで、平均交差エントロピーを最小化する。

```
@nnx.jit
def _cross_entropy_step(model, optimizer, X, Y):
    """One optimizer step on the mean cross-entropy of the logits model(X)
    for the labels Y; negative labels, such as -100, add no loss."""
    def loss_fn(model):
        logits = model(X)
        logits, y = logits.reshape(-1, logits.shape[-1]), Y.reshape(-1)
        valid = y >= 0
        losses = optax.softmax_cross_entropy_with_integer_labels(
            logits, jnp.where(valid, y, 0))
        return (losses * valid).sum() / valid.sum()
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    optimizer.update(model, grads)
    return loss

def train_steps(model, data, optimizer, num_steps, *, stop=None):
    """Minimize the mean cross-entropy of the logits model(X) for the labels
    Y over the minibatches (X, Y) of data, leaving out the labels -100, for
    num_steps steps or until stop(step) is true; return every step's loss."""
    losses, pending, step = [], [], 0
    while step < num_steps:
        for X, Y in data.train_dataloader():
            pending.append(_cross_entropy_step(
                model, optimizer, jnp.asarray(X), jnp.asarray(Y)))
            step += 1
            done = step >= num_steps or (stop is not None and stop(step))
            if len(pending) == 50 or done:  # Sync now and then
                losses.extend(float(l) for l in jax.device_get(pending))
```

```
                pending = []
            if done:
                return losses
```

次に、`lm_perplexity` はミニバッチのイテレータ上で言語モデルを推論モードで評価し、トークンあたりの exp(mean cross-entropy) を返す。

```
@jax.jit(static_argnums=0)
def lm_loss_sum(graphdef, state, X, Y):
    """Summed next-token cross-entropy of a batch, and the model's state."""
    model = nnx.merge(graphdef, state)
    loss = optax.softmax_cross_entropy_with_integer_labels(model(X), Y).sum()
    return loss, nnx.state(model)

def lm_perplexity(model, batches):
    """Perplexity of a language model on the minibatches (X, Y) of batches."""
    # A view with dropout off, sharing the weights; one walk through it
    graphdef, state = nnx.split(nnx.view(model, deterministic=True,
                                         raise_if_not_found=False))
    total, count = 0, 0
    for X, Y in batches:
        loss, state = lm_loss_sum(graphdef, state, X, Y)
        total, count = total + loss, count + Y.size
    nnx.update(model, state)  # Keep statistics the model records, if any
    return math.exp(float(total) / count)
```

ノイズを含むステップ損失の履歴を要約・プロットするため、`final_loss` は直近 $k = 100$ ステップの損失を平均し、`smooth` は 25 ステップごとのブロック平均を計算し、`steps_to_reach` は平滑化された損失が `target` に到達する最初のステップを求める。

```
def final_loss(losses, k=100):
    """Mean of the last k losses; infinity if the run diverged."""
    v = sum(losses[-k:]) / k
    return v if math.isfinite(v) else float('inf')

def smooth(losses, k=25):
    """Means over consecutive blocks of k losses, for plotting."""
    return [sum(losses[i:i + k]) / k
            for i in range(0, len(losses) - k + 1, k)]

def steps_to_reach(losses, target, k=25):
    """First step at which the smoothed losses reach target; None if they
    never do."""
    for i, v in enumerate(smooth(losses, k)):
        if v <= target:
            return k * (i + 1)
    return None
```

最後に、`lr_sweep` は同じ乱数 `seed` から初期化した新しいモデルを `lrs` 内の各学習率で訓練し、指定があれば `evaluate(model)` を `every` ステップごとに実行する。

```
def lr_sweep(make_model, make_tx, data, lrs, seed,
             num_steps=2000, *, evaluate=None, metric='score', every=None,
             verbose=True):
    """Train make_model(rngs=...) with the optax transformation make_tx(lr)
    for each lr, from one initialization; return the loss curves and the
    evaluations."""
    every = every or num_steps
    curves, scores = {}, {}
    for lr in lrs:  # The same initialization for every lr
        model = make_model(rngs=nnx.Rngs(seed))
        optimizer = nnx.Optimizer(model, make_tx(lr), wrt=nnx.Param)
        curves[lr], scores[lr] = [], []
        for _ in range(num_steps // every):
            curves[lr] += train_steps(model, data, optimizer, every)
            if evaluate is not None:
                scores[lr].append(evaluate(model))
        line = f'lr {lr:g}: final loss {final_loss(curves[lr]):.3f}'
        if scores[lr]:
            line += f', {metric} {scores[lr][-1]:.3f}'
        if verbose:
            print(line)
    return curves, scores
```

9.3.4 ミニバッチ SGD のベースライン実行と実験プロトコル

ここで、定数学習率のミニバッチ SGD（2,000 ステップ、モメンタムも重み減衰もなし）によるベースラインを確立しよう。TinyLM では、未訓練のモデルは一様分布の交差エントロピー $\ln 256 \approx 5.55$ nats/byte よりわずかに高い値から出発する（初期ロジットの分散のため ~ 5.71 nats/byte となる。演習問題 1 を参照）。WikiText-103 における参照用のランドマークとして、ユニグラムのバイト頻度テーブルは ~ 3.2 nats/byte、バイグラムのバイトペアテーブルは ~ 2.5 nats/byte を達成する（演習問題 6）。Figure 9.4 はこれらのランドマークを 1 つの尺度上に並べたものであり、本章におけるすべての損失はこの尺度に照らして読み取ることができる。

まず、モデルと素の SGD を生成するファクトリクロージャ `make_lm`、`sgd`、`new_seed` を定義する。

```
make_lm = lambda rngs: TinyLM(len(data.vocab), rngs=rngs)
sgd = optax.sgd
new_seed = d2l.get_seed
```

`lr_sweep` を TinyLM 上で $\eta \in \{0.1, 0.5, 2.5\}$ にわたって実行し、ステップごとの損失を一様な当て推量のベースライン $\ln 256$ とともにプロットすると、典型的な 3 つの学習率領域が現れる。

```
lm_sgd, _ = lr_sweep(make_lm, sgd, data, lrs=[0.1, 0.5, 2.5], seed=new_seed())
d2l.plot(list(range(1, 2001)),
         list(lm_sgd.values()) + [[math.log(256)] * 2000],
         'step', 'training loss', xscale='log', ylim=[1.5, 6],
         legend=[f'lr {lr:g}' for lr in lm_sgd] + ['uniform guess'],
         legend_loc='outside', figsize=(5, 2.5))
```

```
lr 0.1: final loss 2.485
lr 0.5: final loss 2.131
```

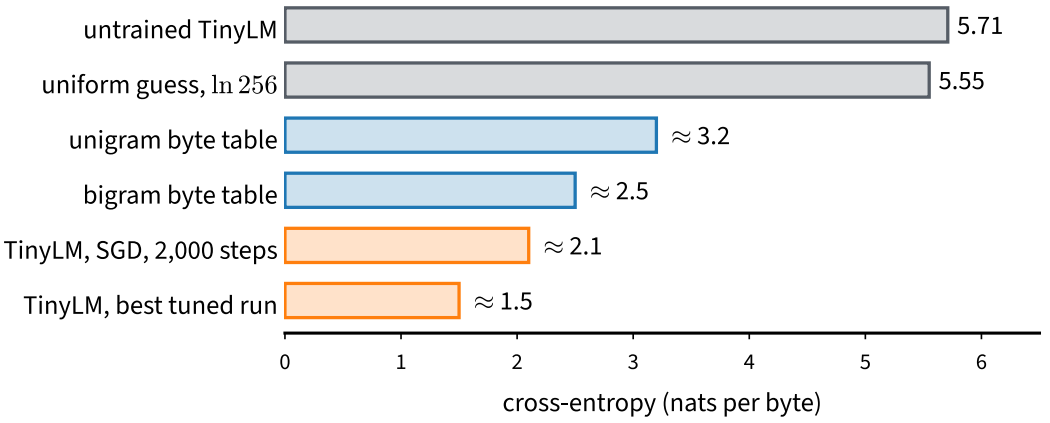
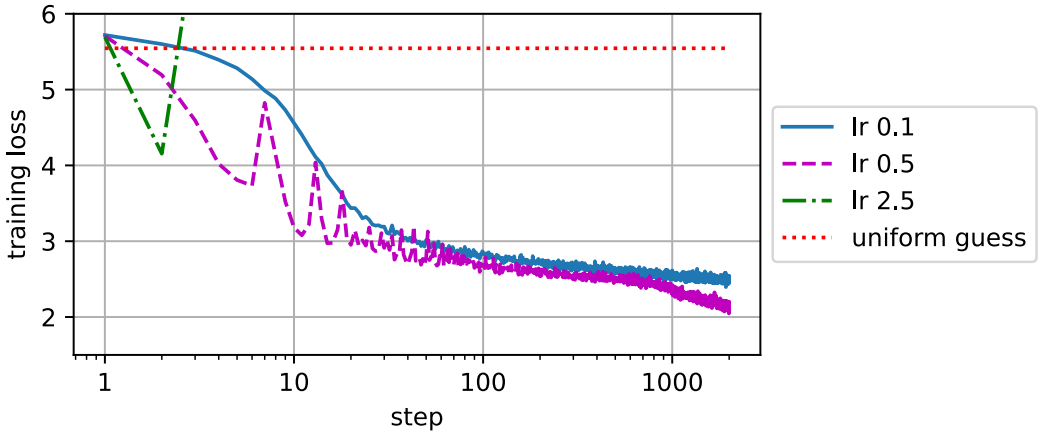


Figure 9.4: WikiText-103 におけるバイトレベル言語モデリングの損失ランドマーク。灰色：未訓練の TinyLM と一様な当て推量。青：ユニグラムおよびバイグラムのバイトテーブル。オレンジ：以下で測定する 2,000 ステップの SGD 実行後の TinyLM、および本章後半で最もよく調整された実行後の結果。

```
lr 2.5: final loss inf
```



- **高すぎる場合**（$\eta = 2.5$）：最初のステップの後に局所的な安定性しきい値 $2/L$ を超え、10 ステップ以内に `inf` へと発散する。
- **低すぎる場合**（$\eta = 0.1$）：滑らかだがゆっくりと降下し、到達するのは $\sim$ 2.5 nats/byte（バイグラムの水準、ステップ 2,000 時点）にとどまる。
- **適切に調整された場合**（$\eta = 0.5$）：ネットワークが初期の高曲率領域を通過する際に序盤で一時的なスパイクを経験するが、その後回復して約 2.1 nats/byte に到達する。

FashionCNN では、$\eta = 0.3$ の素の SGD が訓練損失 $\sim$ 0.22、ホールドアウトのテスト正解率 $\sim$ 90% に到達する。

```
test_acc = lambda model: evaluate_accuracy(model, fashion.val_dataloader())
cnn_sgd, _ = lr_sweep(FashionCNN, sgd, fashion, lrs=[0.3], seed=new_seed(),
```

```
                    evaluate=test_acc, metric='test accuracy')
```

```
lr 0.3: final loss 0.216, test accuracy 0.900
```

9.3.4.1 比較プロトコルと有意性のしきい値

Deep Learning Tuning Playbook (Godbole et al. 2023) に従い、2 つのオプティマイザを公平に比較するには、学習率 η を手法ごとに独立にスイープすべき**局外ハイパーパラメータ**（nuisance hyperparameter）として扱い（共通の初期重みから対数等間隔グリッド上で `lr_sweep` を実行する）、任意の共通学習率で 2 つのオプティマイザを比較することは避けなければならない (Schmidt et al. 2021)。各オプティマイザの最良の学習率で達成された最小損失とあわせて、その**使用可能な学習率範囲**（usable learning-rate range：最適値から 0.15 nats 以内で終了するグリッド点の範囲）も追跡する。

適切に調整された学習率のもとで 5 個の独立な乱数シードにわたって実行すると、末尾 100 ステップの訓練損失（`final_loss`）および検証損失は `TinyLM` では ~ 0.01–0.015 nats の範囲で変動し、一方 `FashionCNN` では訓練損失で ~ 0.02、テスト正解率で $\sim 1\%$ 変動する。保守的な $3\times$ の範囲マージンを採用し、単一実行での差を統計的に有意とみなすのは、`TinyLM` で 0.05 **nats**、または `FashionCNN` で 0.06 **の訓練損失（3% のテスト正解率）**を超える場合に限ることとする。それより小さな差は、複数のシードにわたって確認されない限り引き分けとして扱う。

9.3.5 まとめ

本章の経験的な比較では、相補的な 2 つのニューラルネットワークベンチマークを組み合わせる。1 つ目は `TinyLM`（$470{,}784$ パラメータ：90% が 2 次元射影行列、9% が埋め込み、$< 1\%$ が 1 次元ベクトル）であり、ランダムな 64 バイトウィンドウを WikiText-103 から切り出すシングルパスのストリーミング設定で訓練するため、$2{,}000$ ステップの実行において毎ステップ新しいテキストを評価できる。2 つ目は `FashionCNN`（$421{,}642$ パラメータ）であり、~ 8.5 エポックにわたり 256 枚の画像からなる Fashion-MNIST のミニバッチ上で訓練し、訓練損失とホールドアウトのテスト正解率の両方で評価する。また、閉形式の一様・ユニグラム・バイグラムのバイトエントロピーによって、訓練を始める前の段階で `TinyLM` の損失スケールに基準を与える。アルゴリズム本来の改善をハイパーパラメータやシードに対する感度から切り分けるため、すべての比較において同一の初期重みからオプティマイザごとに学習率 η を独立にスイープし、複数シードに基づく有意性のしきい値である 0.05 nats（`TinyLM`）および損失 0.06 または正解率 3%（`FashionCNN`）に照らして、最適損失と使用可能な学習率範囲の両方を報告する。

9.3.6 演習問題

1. **訓練前の初期損失。**
    1. ロジット $z_1, \dots, z_V$ がターゲットとは独立に $\mathcal{N}(0, s^2)$ から独立同分布に抽出されるとき、$\sum_{i=1}^{V} e^{z_i} \approx V\mathbb{E}[e^{z_i}] = V e^{s^2/2}$ を用いて、初期交差エントロピーの期待値がおよそ $\ln V + s^2/2$ になることを示せ。
    2. 平均 0、分散 1 の LayerNorm 出力と、$\mathcal{U}(-1/\sqrt{128}, 1/\sqrt{128})$ から抽出されたヘッドの重み・バイアスが与えられたとき、$s^2 = 129/384 \approx 1/3$ かつ $\ln 256 + 1/6 \approx 5.71$ nats となり、これがステップ 1 における `TinyLM` の損失と一致することを示せ。ヘッドの重みとバイアスを 0 で初期化した場合、初期損失はいくらになるか。
2. [code] **`FashionCNN` の損失曲線の診断。** `FashionCNN` を $\eta \in \{0.003, 0.03, 0.3, 1.0\}$ にわたって素の SGD で訓練し、ステップ損失を対数スケールのステップ軸上で一様分布のベースラ

イン $\ln 10 \approx 2.3026$ とともにプロットせよ。$\ln 10$ 付近で停滞する実行について、1 ミニバッチにわたる死んだ ReLU の活性化値の割合を調べ、なぜ勾配が消失するのかを説明せよ。

3. [code] **TinyLM における発散の境界**。TinyLM 上で素の SGD を $\eta \in \{0.7, 1.0, 1.4, 2.0\}$ および 5 個のシードにわたってスイープせよ（`lr_sweep(..., verbose=False)`）。生存率、発散する実行が破綻するステップ、および発散しなかった実行における最小損失を記録せよ。

4. [code] **複数シードのばらつきと標準誤差**。●20 個のシードで TinyLM を AdamW（9.5 節）により $\eta = 3 \times 10^{-3}$（`weight_decay=0`）で実行し、`final_loss` の標本標準偏差 σ を正規分布の範囲推定量 $\text{range}/3.7$ と比較したうえで、必要なシード数 k（0.005 nats の差を標準誤差の 2 倍 $\sigma\sqrt{2/k}$ の水準で検出できる数）を求めよ。

5. [code] **訓練損失と検証損失の比較および小規模コーパスでの過学習**。

   1. なぜ `final_loss`（直近 100 訓練ステップの平均）がわずかに（0.02–0.03 nats ほど）ステップ 2000 終了後の TinyLM の検証損失より高くなるのかを説明し、最終チェックポイントをホールドアウトの訓練ウィンドウ上で評価して確認せよ。
   2. コーパスを 100,000 バイトに制限し（`WikiText103(batch_size=64, num_steps=64, num_train=100000)` により ~ 80 エポックを 2,000 ステップで回る設定）、訓練損失と検証損失を 250 ステップごと（`every=250`）にプロットせよ。

6. [code] **カウントベースの n グラムランドマーク**。経験的なバイトユニグラムのエントロピー（~ 3.2 nats）、ならびに 1 加算平滑化を用いたバイグラム（~ 2.5 nats）およびトライグラムの検証分割上での交差エントロピーを、先頭 10^7 訓練バイトの WikiText103 から計算し、2,000 ステップの SGD（約 2.1 nats）および AdamW（1.59 nats、9.5 節）による TinyLM の損失と比較せよ。

9.4 モメンタム

9.1 節では、通常の勾配降下法が悪条件（ill-conditioned）の谷で苦戦する理由を見た。最も急峻な曲率 $\lambda_{\max}$ によって安定性の上限 $\eta < 2/\lambda_{\max}$ が課されるため、最も平坦な方向 $\lambda_{\min}$ はステップごとに高々 $O(1/\kappa)$ しか縮小しない（ここで $\kappa = \lambda_{\max}/\lambda_{\min}$ は条件数である）。通常の勾配降下法が遅いのは、各ステップがその時点の勾配 $\mathbf{g}_t$ のみに依存しているからである。しかし、直近の勾配の履歴には明確な幾何学的情報が含まれている。谷底に沿っては連続する勾配が一貫して同じ方向を指すのに対し、急峻な壁を横切る方向では反復点が左右にオーバーシュートするため勾配の符号が交互に反転する。

Polyak (1964) は、過去の勾配の指数重み付き和である**速度**（velocity）に沿ってステップを進める手法を提案した。これにより、平坦な方向に沿う持続的な成分は蓄積され、急峻な方向を横切る振動成分は打ち消し合う。摩擦を受けながら溝を転がり落ちる重い球になぞらえられる**ヘビーボールモメンタム**は、パラメータごとに単一の速度バッファを保持するコストと引き換えに、二次ボウル上での最悪ケースの反復回数を $O(\kappa)$ から $O(\sqrt{\kappa})$ へと削減する。

本節では、速度の漸化式、その実効ステップサイズ（effective step size）と記憶の時間スケール、および臨界減衰（critical damping）による $\sqrt{\kappa}$ 加速則を解析する。続いて、フルバッチのソフトマックス回帰と FashionCNN のミニバッチ訓練でモメンタムを検証し、ネステロフの先読みモメンタムを導入したうえで、β と η を調整するための実践的な指針を確立する。

9.4.1 モメンタム法

9.4.1.1 速度と実効ステップサイズ

勾配 $\mathbf{g}_t$（ステップ t における厳密勾配またはミニバッチ勾配）が与えられたとき、ヘビーボールモメンタムは速度ベクトル $\mathbf{v}_t$（初期値 $\mathbf{v}_0 = \mathbf{0}$）を保持し、次のように更新する。

$$\begin{aligned} \mathbf{v}_t &= \beta \mathbf{v}_{t-1} + \mathbf{g}_t, \\ \mathbf{x}_{t+1} &= \mathbf{x}_t - \eta \mathbf{v}_t, \end{aligned} \tag{9.22}$$

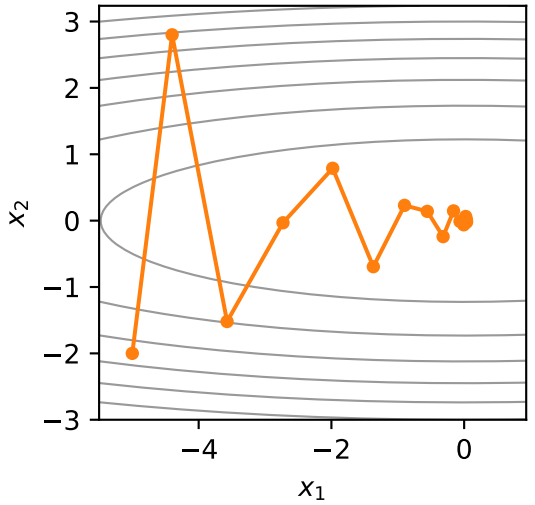
ここで、$\beta \in [0, 1)$ はモメンタム係数である（$\beta = 0$ とすると勾配降下法に戻る）。この漸化式を展開すると、速度は過去のすべての勾配の指数重み付き和として表される。

$$\mathbf{v}_t = \sum_{k=0}^{t-1} \beta^k \mathbf{g}_{t-k} = \mathbf{g}_t + \beta \mathbf{g}_{t-1} + \beta^2 \mathbf{g}_{t-2} + \cdots + \beta^{t-1} \mathbf{g}_1.$$

この重み付き和が、谷 $f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$（$\lambda_1 = 0.2, \lambda_2 = 4$）の 2 つの座標に対して $\eta = 0.6$ でどのように働くか考えてみよう。この学習率は通常の勾配降下法の安定限界 $2/\lambda_2 = 0.5$ を超えており、通常の勾配降下法では x_2 方向に沿ってステップごとに $1 - 4(0.6) = -1.4$ 倍となって発散する。ここにモメンタム $\beta = 0.5$ を加えると軌跡が安定化し、両座標とも原点の 0.01 以内に 20 ステップで到達する。

```
h1, h2 = d2l.VALLEY_CURVATURES
f_valley, valley_grad = d2l.f_valley, d2l.valley_grad

def momentum_2d(x1, x2, v1, v2, f_grad, eta, beta):
    g1, g2 = f_grad(x1, x2)
    v1, v2 = beta * v1 + g1, beta * v2 + g2
    return x1 - eta * v1, x2 - eta * v2, v1, v2

d2l.show_trace_2d(f_valley, d2l.train_2d(momentum_2d, valley_grad, eta=0.6,
                                         beta=0.5))
```

```
step 20, x1: 0.007188, x2: 0.002553
```



モメンタムが平坦な座標を加速しつつ同時に急峻な座標を安定化できる理由は、次の 2 つの極限的なケースから説明できる。

- **持続する勾配（平坦な谷底）：**連続する勾配がほぼ一定（$\mathbf{g}_{t-k} \approx \mathbf{g}$）にとどまるとき、速度は $\mathbf{v}_t \approx (1 + \beta + \beta^2 + \cdots)\mathbf{g} = \frac{1}{1-\beta}\mathbf{g}$ まで蓄積し、次の**実効ステップサイズ**をもたらす。

$$\eta_{\text{eff}} = \frac{\eta}{1 - \beta}. \tag{9.23}$$

- **符号が反転する勾配（急峻な壁）：**連続する勾配の符号が交互に反転する（$\mathbf{g}_{t-k} \approx (-1)^k \mathbf{g}$）とき、幾何級数の和は $1 - \beta + \beta^2 - \cdots = \frac{1}{1+\beta}$ となり、実効ステップサイズは $\frac{\eta}{1+\beta}$ へと減衰する。

Figure 9.5 に、単位サイズの勾配に対する両ケースを示す。

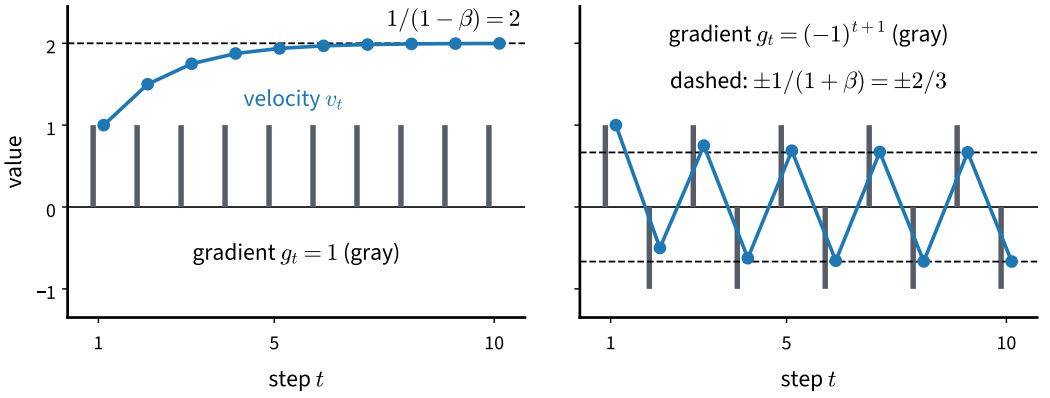


Figure 9.5: 速度 $v_t = \beta v_{t-1} + g_t$（青）の推移（単位勾配 g_t は灰色の棒、$\beta = 0.5$）。左：同符号の勾配が足し合わさり、v_t は $1/(1-\beta) = 2$ に近づく。右：符号が交互に反転する勾配は大半が打ち消し合い、v_t は振幅 $1/(1+\beta) = 2/3$ の振動に落ち着く。

$\eta = 0.6$、$\beta = 0.5$ のとき、モメンタムは谷底に沿った前進を $\eta_{\text{eff}} = 0.6/0.5 = 1.2$ へと増幅す

る一方、急峻な壁を横切る振動を $0.6/1.5 = 0.4 < 0.5$ へと減衰させる。その結果、曲率 λ に対する安定性の上限は $\eta\lambda < 2$ から $\eta\lambda < 2(1+\beta)$ へと引き上げられる（演習問題 3）。

9.4.1.2 β の時間スケールとミニバッチノイズ

重み $1, \beta, \beta^2, \ldots$ の和は $1/(1-\beta)$ となるため、速度は $\tau = 1/(1-\beta)$ ステップの実効記憶ウィンドウと $\log(1/2)/\log\beta$ ステップの半減期を持つ。

```
print('beta   window   half-life   weight inside the window')
for beta in [0.5, 0.9, 0.99]:
    window = 1 / (1 - beta)
    half_life = np.log(0.5) / np.log(beta)
    inside = 1 - beta ** window
    print(f'{beta:4}   {window:6.0f}   {half_life:9.1f}   {inside:24.2f}')
```

```
beta   window   half-life   weight inside the window
 0.5        2         1.0                      0.75
 0.9       10         6.6                      0.65
0.99      100        69.0                      0.63
```

β を 0.9 から 0.99 に増やすと、記憶ホライズンと実効ステップサイズ $\eta_{\text{eff}} = \eta/(1-\beta)$ の両方が 10 倍（10 ステップから 100 ステップへ）に拡大する。

確率的勾配 $\mathbf{g}_t = \nabla f(\mathbf{x}_t) + \xi_t$ のもとで、$1/(1-\beta)$ ステップにわたる平均化は 9.2 節の定常ノイズ球（noise ball）を縮小させるだろうか。答えは否である。各ノイズ摂動 ξ_t は $\mathbf{v}_t$ に重み 1 で加わり、その後のステップで β^k として減衰するため、累積変位として $\eta(1+\beta+\beta^2+\cdots)\xi_t = \eta_{\text{eff}}\xi_t$ だけ寄与する。したがって、実効ステップサイズ $\eta_{\text{eff}} = \eta/(1-\beta)$ を揃えた場合、モメンタムありの SGD とモメンタムなしの SGD はほぼ同一の分散を持つ定常ノイズ球に落ち着く（演習問題 5）。ゆえに、ノイズ律速（noise-limited）の訓練で β を大きくする際には、η を $1-\beta$ に比例してスケーリングし、η_{eff} を一定に保つべきである。

9.4.2 加速と臨界減衰

組 (η, β) としてどのような選択が、固有値 $\lambda_{\min} \le \cdots \le \lambda_{\max}$ と条件数 $\kappa = \lambda_{\max}/\lambda_{\min}$ を持つ二次形式において最適となるだろうか。Polyak (1964) は、次のように設定すればよいことを証明した。

$$\eta^\star = \frac{4}{\left(\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}\right)^2}, \qquad \beta^\star = \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^2 \tag{9.24}$$

このとき、最悪ケースにおけるステップごとの縮小率（contraction factor）は次の値に達する。

$$\rho^\star = \sqrt{\beta^\star} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \approx 1 - \frac{2}{\sqrt{\kappa}},$$

これにより、相対精度 ϵ に到達するために必要なステップ数は $\frac{\kappa}{2}\log(1/\epsilon)$ から $\frac{\sqrt{\kappa}}{2}\log(1/\epsilon)$ へと削減される（35.1.3 節）。

縮小率が $\sqrt{\beta^\star}$ に等しくなる理由を見るため、$\eta v_t = x_t - x_{t+1}$ を曲率 λ の固有ベクトルに沿った 1 次元の速度更新式に代入すると、次の 2 階線形漸化式が得られる。

$$x_{t+1} = (1 + \beta - \eta\lambda)\, x_t - \beta\, x_{t-1}, \tag{9.25}$$

この漸化式の特性多項式は $z^2 - (1 + \beta - \eta\lambda)z + \beta = 0$ である。解と係数の関係により、2 つの根の積は $z_1 z_2 = \beta$ となる。判別式 $\Delta = (1 + \beta - \eta\lambda)^2 - 4\beta$ が非正であるとき（これはすべての $\beta \geq \beta_{\mathrm{c}}(\lambda) = (1 - \sqrt{\eta\lambda})^2$ で生じる）、根は同じ絶対値 $|z_1| = |z_2| = \sqrt{\beta}$ を持つ複素共役対をなし、その大きさは曲率 λ に依存しない（Figure 9.6）。

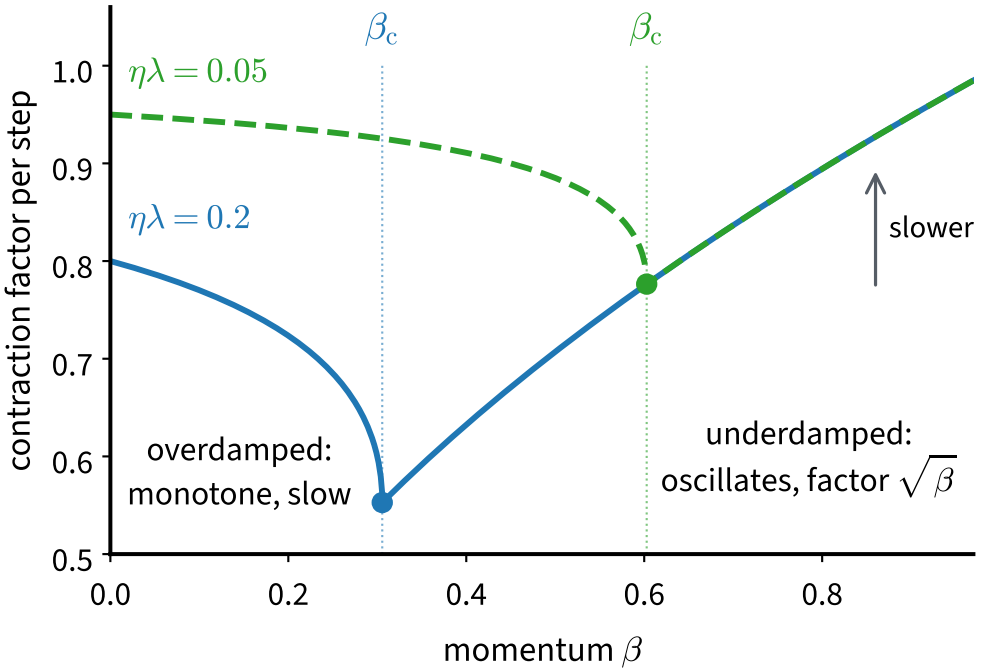


Figure 9.6: 曲率 λ に沿ったヘビーボールモメンタムのステップごとの縮小率（β の関数）。収束は臨界減衰のしきい値 $\beta_{\mathrm{c}} = (1 - \sqrt{\eta\lambda})^2$ で最速となり、β_{c} 未満では過減衰、β_{c} を超えると絶対値 $\sqrt{\beta}$ で振動する。

Equation 9.24 の最適パラメータは、最も平坦な固有値と最も急峻な固有値の臨界減衰しきい値を $(1 - \sqrt{\eta^\star \lambda_{\min}})^2 = (\sqrt{\eta^\star \lambda_{\max}} - 1)^2 = \beta^\star$ として等しくする。これにより両端のモードを臨界減衰させるとともに、すべての中間固有値 $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ が不足減衰（underdamped）の複素根領域に入り、縮小率 $\sqrt{\beta^\star}$ で縮小することを保証する。

先ほどの谷（$\kappa = 20$）では、Equation 9.24 から $\eta^\star \approx 0.668, \beta^\star \approx 0.403$ が得られる。これにより漸近縮小率は 0.905 から $\sqrt{\beta^\star} \approx 0.635$ に低下し、相対誤差 10^{-6} に達するまでのステップ数は 139 から 40 へと減少する。

```
def steps_to(tol, eta, beta, steps=400):
    """Steps after which |x1|, |x2| stay below tol times their start."""
    x1, x2, v1, v2, last = -5.0, -2.0, 0.0, 0.0, 0
    for t in range(1, steps + 1):
        x1, x2, v1, v2 = momentum_2d(x1, x2, v1, v2, valley_grad, eta, beta)
        if abs(x1) > 5 * tol or abs(x2) > 2 * tol:
```

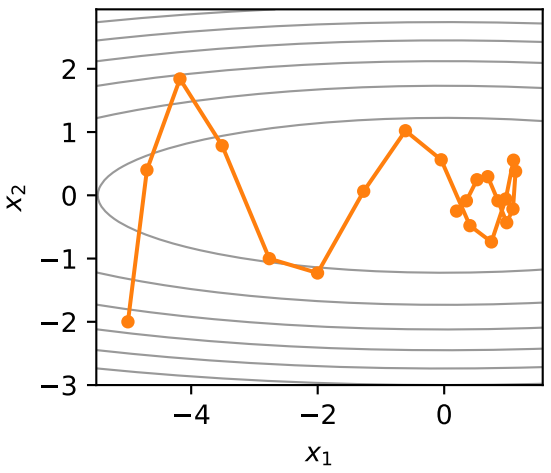
```
            last = t
        return last + 1

    def contraction(eta, beta):
        return max(abs(np.roots([1, -(1 + beta - eta * lam), beta])).max()
                   for lam in (h1, h2))

    eta_opt = 4 / (np.sqrt(h2) + np.sqrt(h1)) ** 2
    beta_opt = ((np.sqrt(h2 / h1) - 1) / (np.sqrt(h2 / h1) + 1)) ** 2
    print('  eta    beta    factor    steps to 1%    steps to 1e-6')
    for eta, beta in [(2 / (h1 + h2), 0), (0.6, 0.5), (eta_opt, beta_opt),
                      (0.3, 0.8)]:
        print(f'{eta:5.3f}  {beta:5.3f}  {contraction(eta, beta):7.3f}  '
              f'{steps_to(1e-2, eta, beta):12d}  {steps_to(1e-6, eta, beta):14d}')
```

```
  eta    beta    factor    steps to 1%    steps to 1e-6
0.476  0.000     0.905             47              139
0.600  0.500     0.707             16               42
0.668  0.403     0.635             18               40
0.300  0.800     0.894             42              126
```

4 行目（$\eta = 0.3, \beta = 0.8$）に注目しよう。β を臨界しきい値 β_c より大幅に大きく選ぶと、両座標とも絶対値 $\sqrt{0.8} \approx 0.894$ の不足減衰となり、軌跡はオーバーシュートして原点周りで減衰振動（リンギング）を起こす。

```
d2l.show_trace_2d(f_valley, d2l.train_2d(momentum_2d, valley_grad, eta=0.3,
                                         beta=0.8))
```

```
step 20, x1: 0.195696, x2: -0.249649
```



9.4.3 データ上でのモメンタム

これまでの解析から、データ上で訓練する際には次の 2 つの異なるレジームが予測される。

1. **曲率律速レジーム（高精度な勾配、大きな κ）：**モメンタムによってより大きな実効ステップサイズ η_{eff} が可能になり、必要な反復回数が大幅に削減される。
2. **ノイズ律速レジーム（小さなミニバッチ）：**定常ノイズフロアは $\eta_{\text{eff}} = \eta/(1-\beta)$ に支配されるため、$\beta = 0.9$ を加えると最適学習率は $1-\beta = 0.1$ 倍に下がる一方、収束速度は同程度にとどまる。

Fashion-MNIST 上で $\beta = 0$ と $\beta = 0.9$ を比較し、両レジームを検証しよう。対象は、(1) フルバッチのソフトマックス回帰（$b = 60{,}000$、厳密な勾配、非負のピクセル強度により悪条件）と、(2) `FashionCNN`（9.3.2 節）の小ミニバッチ訓練（$b = 32$）である。まず、正規化した Fashion-MNIST テンソルをメモリ内ローダ `Batches` としてアクセラレータ上に事前読み込みする。

```
fashion = d2l.FashionMNIST()
X = jnp.asarray(fashion.train[0][..., None], jnp.float32) / 255  # NHWC
y = jnp.asarray(fashion.train[1], jnp.int32)

class Batches:
    """The examples X, y on the device, in a new random order on every pass."""
    def __init__(self, X, y, batch_size):
        self.X, self.y, self.batch_size = X, y, batch_size
        self.key = d2l.get_key()  # Advanced on every pass

    def train_dataloader(self):
        self.key, key = jax.random.split(self.key)
        order = jax.random.permutation(key, len(self.y))
        return ((self.X[i], self.y[i])
                for i in order.reshape(-1, self.batch_size))
```

次に、ゼロ初期化した線形モデル `softmax_regression` と、モメンタムを引数にとる SGD コンストラクタ `make_sgd(beta)` を定義する。

```
def softmax_regression(rngs):
    """A linear layer from the pixels to the logits, starting at zero."""
    return nnx.Sequential(lambda X: X.reshape(len(X), -1),
                          nnx.Linear(28 * 28, 10, rngs=rngs,
                                     kernel_init=nnx.initializers.zeros))

make_sgd = lambda beta: lambda lr: optax.sgd(lr, momentum=beta)
new_seed = d2l.get_seed
```

ここで `d2l.lr_sweep` を用いて `grids` にわたる学習率スイープを両モデルと $\beta \in \{0.0, 0.9\}$ に対して実行し、各設定における最良の学習率、目標損失への到達ステップ数、および最終損失を記録する。

```
# Model, batch size, steps, target loss
setups = {'softmax regression': (softmax_regression, 60000, 1000, 0.5),
          'FashionCNN': (d2l.FashionCNN, 32, 3000, 0.3)}
grids = {('softmax regression', 0.0): [0.0625, 0.125, 0.25, 0.5, 1],
         ('softmax regression', 0.9): [0.1, 0.2, 0.4, 0.8],
         ('FashionCNN', 0.0): [0.05, 0.1, 0.2, 0.4],
         ('FashionCNN', 0.9): [0.005, 0.01, 0.02, 0.04]}
best = {}
```

```
for (name, beta), lrs in grids.items():
    make_model, b, num_steps, target = setups[name]
    curves, _ = d2l.lr_sweep(make_model, make_sgd(beta), Batches(X, y, b),
                             lrs, new_seed(), num_steps, verbose=False)
    best[name, beta] = min(lrs, key=lambda lr: d2l.final_loss(curves[lr]))
    c = curves[best[name, beta]]
    print(f'{name:<18}  beta={beta}  best lr {best[name, beta]:<5g}  steps to '
          f'{target}: {d2l.steps_to_reach(c, target)}  final loss '
          f'{d2l.final_loss(c):.3f}')
```

```
softmax regression  beta=0.0  best lr 0.125  steps to 0.5: 625  final loss 0.472
softmax regression  beta=0.9  best lr 0.4    steps to 0.5: 100  final loss 0.365
FashionCNN          beta=0.0  best lr 0.1    steps to 0.3: 2000  final loss 0.288
FashionCNN          beta=0.9  best lr 0.02   steps to 0.3: 1475  final loss 0.277
```

いずれの予測も定量的に成り立つことがわかる。

- フルバッチの**ソフトマックス回帰**では、モメンタム $\beta = 0.9$ を加えることで最適学習率が $\eta = 0.125$ から $\eta = 0.4$ に上昇し（$\eta_{\text{eff}} = 4.0$ と実効ステップサイズが $32\times$ に拡大）、損失 0.5 に到達するまでのステップ数は $6\times$ 以上削減され（$625 \to 100$ ステップ）、最終損失も 0.472 から 0.365 へと低下する。
- 小バッチの **FashionCNN**（$b = 32$）では、訓練はノイズ律速となる。最適学習率は $\eta \approx 0.1$〜0.2（$\beta = 0$ 時）から $\eta \approx 0.02$〜0.04（$\beta = 0.9$ 時）へと低下し（比率にして 5〜10 倍の変化であり、実効ステップサイズ η_{eff} は $\beta = 0$ での値から公比 2 のグリッド上で 1 段階以内に保たれる）、いずれの設定でも損失 0.3 に到達し、最終損失も 0.26〜0.30 と 9.3 節の単一シード変動の範囲内に収まる。

9.4.4 ネステロフのモメンタム

不足減衰の実行例（$\eta = 0.3, \beta = 0.8$）においてヘビーボールモメンタムが最小点をオーバーシュートしたのは、速度によってまさに到達しようとしている地点ではなく、現在位置 $\mathbf{x}_t$ で勾配を評価していたためである。Nesterov (1983) は、**先読み点**（look-ahead point）$\mathbf{x}_t - \eta\beta\mathbf{v}_{t-1}$ で勾配を評価する手法を提案した（Figure 9.7）。

$$\begin{aligned} \mathbf{v}_t &= \beta\,\mathbf{v}_{t-1} + \nabla f(\mathbf{x}_t - \eta\beta\,\mathbf{v}_{t-1}), \\ \mathbf{x}_{t+1} &= \mathbf{x}_t - \eta\,\mathbf{v}_t. \end{aligned} \tag{9.26}$$

先読み点で勾配を評価すると、急峻な方向の振動を抑える先回り減衰（anticipatory damping）が導入される。さらに、理論的保証もヘビーボールモメンタムより広い。定数 $\beta = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ を用いれば、ネステロフのモメンタムは二次関数だけでなく、あらゆる滑らかな強凸関数で $\sqrt{\kappa}$ のレートを達成する。また、増加スケジュール $\beta_t = (t-1)/(t+2)$ を用いれば、一般の滑らかな凸目的関数に対して最適な $O(1/t^2)$ の収束率を達成する (Nesterov 2018)（35.1.3 節）。ここでは定数の β を用いる。

```
def nesterov_2d(x1, x2, v1, v2, f_grad, eta, beta):
    g1, g2 = f_grad(x1 - eta * beta * v1,  # Gradient at the look-ahead
                    x2 - eta * beta * v2)  # point, not at (x1, x2)
    v1, v2 = beta * v1 + g1, beta * v2 + g2
    return x1 - eta * v1, x2 - eta * v2, v1, v2
```

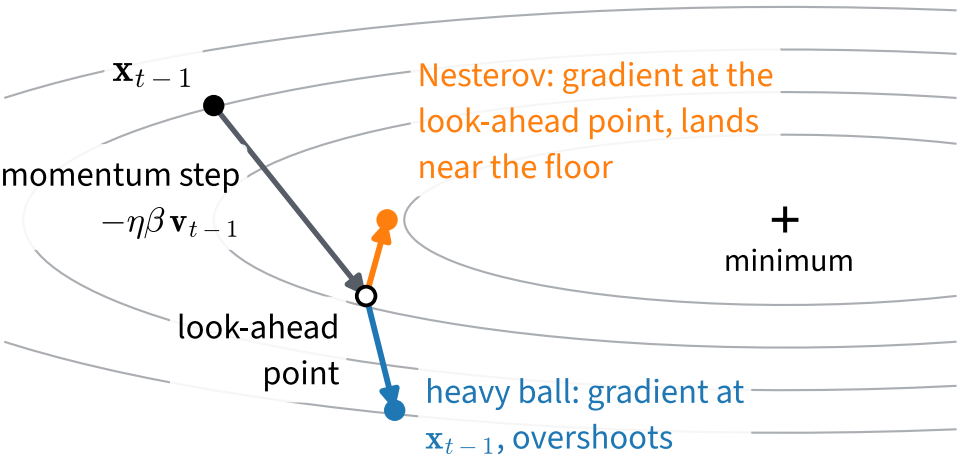
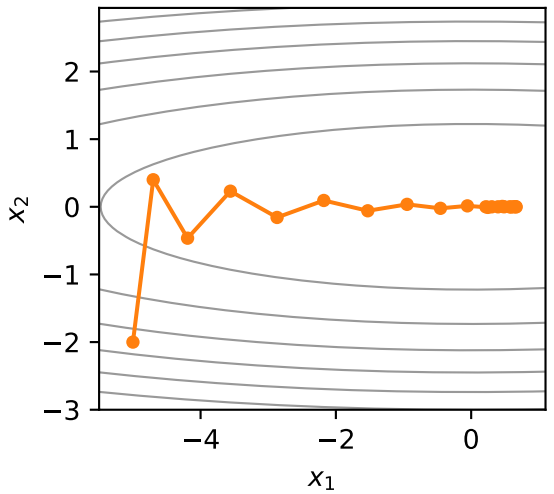


Figure 9.7: 同じ反復点と速度からのヘビーボールモメンタムとネステロフのモメンタムの 1 ステップの比較。ヘビーボールモメンタムは $\mathbf{x}_t$ で勾配を評価してオーバーシュートするが、ネステロフのモメンタムは先読み点 $\mathbf{x}_t - \eta\beta\mathbf{v}_{t-1}$ で勾配を評価し、即座に引き戻す補正を加える。

```
d2l.show_trace_2d(f_valley, d2l.train_2d(nesterov_2d, valley_grad, eta=0.3,
                                         beta=0.8))
```

```
step 20, x1: 0.217791, x2: -0.000070
```



実務上は、ほとんどのアーキテクチャにおいて $\beta = 0.9$ が頑健なデフォルト値となり、学習率は実効ステップサイズ $\eta_{\text{eff}} = \eta/(1-\beta)$ を通じて調整される。しかし、モメンタムは依然としてすべての座標に単一のスカラーのステップサイズを適用する。Transformer のように層によって勾配のスケールが何桁も異なる場合には、座標ごとの適応的なスケーリングが必要となる（9.5 節）。

9.4.5 まとめ

ヘビーボールモメンタムは、指数減衰する速度 $\mathbf{v}_t = \beta \mathbf{v}_{t-1} + \mathbf{g}_t$（記憶ホライズン $1/(1-\beta)$）に沿ってパラメータを更新し、持続的な勾配方向を実効ステップサイズ $\eta_{\text{eff}} = \eta/(1-\beta)$ へと増幅する一方、符号が交互に反転する方向を $\eta/(1+\beta)$ へと減衰させ、安定性の上限を $\eta\lambda < 2(1+\beta)$ に引き上げる。条件数 κ の二次ボウル上では、$(\eta^\star, \beta^\star)$ によって両端の固有値を臨界減衰させることで、反復計算量は $O(\kappa)$ から $O(\sqrt{\kappa})$ へと低下する。ネステロフのモメンタムは先読み点 $\mathbf{x}_t - \eta\beta\mathbf{v}_{t-1}$ で勾配を評価することで先回り減衰をもたらし、不足減衰によるリンギングを抑えるとともに、加速効果を一般の滑らかな凸関数へと拡張する。ミニバッチノイズのもとでは、実効ステップサイズ $\eta_{\text{eff}} = \eta/(1-\beta)$ を揃えれば定常ノイズ球が保たれるため、モメンタムは曲率律速（curvature-limited）レジームで大幅な高速化をもたらす一方、ノイズ律速レジームでは $\eta \propto 1-\beta$ のスケーリングを必要とする。

9.4.6 演習問題

1. [code] **モメンタムと学習率のグリッド**。`momentum_2d` を `f_valley` 上で $\beta \in \{0, 0.25, 0.5, 0.8\}$ および $\eta \in \{0.1, 0.3, 0.6, 1.0\}$ にわたり最大 1,000 ステップ実行せよ。$\|\mathbf{x}_t\| \le 10^{-3}$ に到達するまでのステップ数（または発散）を表にまとめ、各 η における経験的な最適値を $\beta^\star$（Equation 9.24）および $\beta_{\text{c}}(\lambda) = (1-\sqrt{\eta\lambda})^2$ と比較せよ。
2. **速度のノイズ対信号比**。$\mathbf{g}_t = \mathbf{g} + \xi_t$ において、座標ごとに独立同分布で平均ゼロ・分散 σ^2 のノイズが加わるとき、$\mathbf{v}_t$（Equation 9.22）の定常平均が $\mathbf{g}/(1-\beta)$、定常分散が $\sigma^2/(1-\beta^2)$ となり、速度のノイズ対信号比が $\sqrt{(1-\beta)/(1+\beta)}$ 倍に縮小することを示せ。
3. **ネステロフのモメンタムの同値な形式と安定性**。
    1. ライブラリで用いられる形式 $\mathbf{v}_t = \beta\mathbf{v}_{t-1} + \nabla f(\mathbf{y}_t)$、$\mathbf{y}_{t+1} = \mathbf{y}_t - \eta(\nabla f(\mathbf{y}_t) + \beta\mathbf{v}_t)$ が、Equation 9.26 と $\mathbf{y}_t = \mathbf{x}_t - \eta\beta\mathbf{v}_{t-1}$ のもとでまったく同じ速度を生成することを数学的帰納法により示せ。
    2. Equation 9.25 から、$z^2 - (1+\beta-\eta\lambda)z + \beta = 0$ の両根が $0 \le \beta < 1$ において単位円の真に内部に存在するのは、$0 < \eta\lambda < 2(1+\beta)$ のとき、かつそのときに限ることを証明せよ。
4. **指数平均形式のモメンタム**。@eq-momentum を、正規化された EMA 形式 $\mathbf{m}_t = \beta\mathbf{m}_{t-1} + (1-\beta)\mathbf{g}_t$、$\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha\mathbf{m}_t$（Adam で使用、9.5 節）と比較せよ。$\mathbf{m}_t = (1-\beta)\mathbf{v}_t$ となり、したがって $\alpha = \eta/(1-\beta) = \eta_{\text{eff}}$ のとき両者の軌跡が一致することを示せ。
5. [code] **バッチサイズにわたるミニバッチノイズ下のモメンタム**。●
    1. $f(x) = \frac{\lambda}{2}x^2$ 上でノイズ付き勾配 $g_t = \lambda x_t + \xi_t$ を用いるとき、ステップサイズ α の通常の SGD と、モメンタム（$\beta = 0.9$）で $\eta = (1-\beta)\alpha$ とした場合の反復点の定常分散を $\alpha\lambda \in \{0.05, 0.2\}$ について比較せよ。
    2. `d2l.FashionCNN` を 1,000 ステップにわたり、バッチサイズ $b \in \{32, 500, 6000\}$、$\beta \in \{0, 0.9\}$、および揃えた実効ステップサイズ $\eta/(1-\beta) = 0.2$ で訓練せよ。続いて η を $b = 6000$ においてスイープし、バッチサイズが曲率律速レジームに入るほど大きくなるとモメンタムが訓練を加速することを検証せよ。

9.5 Adam と AdamW

9.3 節では、単純なミニバッチ SGD は `TinyLM` において学習率 $\eta = 0.5$ を用いてようやく約 2.1 nats/byte の損失に到達した。これに対し、100 分の 1 未満の学習率である $\eta = 3 \times 10^{-3}$ を用

いた Adam は、以下で測定するように 1.59 nats/byte という損失に同じ 2,000 ステップで到達する。公称学習率が 100 分の 1 ほど小さいオプティマイザがなぜより速く訓練を進められるのか、そしてなぜ Transformer における普遍的なデフォルトとなっているのだろうか。

モメンタム（9.4 節）は勾配を時間方向に平均化するが、それでもすべてのパラメータに単一のスカラーのステップサイズを適用する。しかし深層ネットワーク、とりわけ Transformer では、埋め込み、アテンション射影、MLP 層、正規化パラメータの間で勾配の大きさが何桁も異なる。**Adam**（**Adaptive Moment Estimation**、Kingma and Ba (2015)）は、モメンタムと、**AdaGrad** (Duchi et al. 2011) および **RMSProp** (Tieleman and Hinton 2012) を経て発展した座標ごとの二乗平均平方根（root mean square、RMS）勾配の前処理行列（preconditioner）を組み合わせる。各座標の更新量をその座標自身の過去の勾配スケールで割ることにより、学習率 η は生の勾配に対する乗数ではなく、パラメータ空間における直接的なステップ長となる。

しかし、まさにこの座標ごとの除算が標準的な ℓ_2 正則化（regularization）を破綻させる。損失に $\frac{\lambda}{2}\|\mathbf{x}\|^2$ を加えると、各重みの減衰率がその勾配スケールの逆数で再スケーリングされてしまうからである。Loshchilov and Hutter (2019) は重み減衰（weight decay）を適応的な前処理行列から分離することでこの問題を解決し、**AdamW** を生み出した。本節では、AdaGrad と RMSProp から Adam への発展をたどり、`TinyLM` 上で座標ごとの勾配スケールを測定してモメンタム付き SGD と Adam を比較し、さらに AdamW とその重み減衰のタイムスケール $\tau = 1/(\eta\lambda)$ および平衡ノルムを導出したうえで、どのパラメータグループを重み減衰の対象外とすべきかを詳しく述べる。

9.5.1 座標ごとの適応的学習率

9.5.1.1 AdaGrad と RMSProp

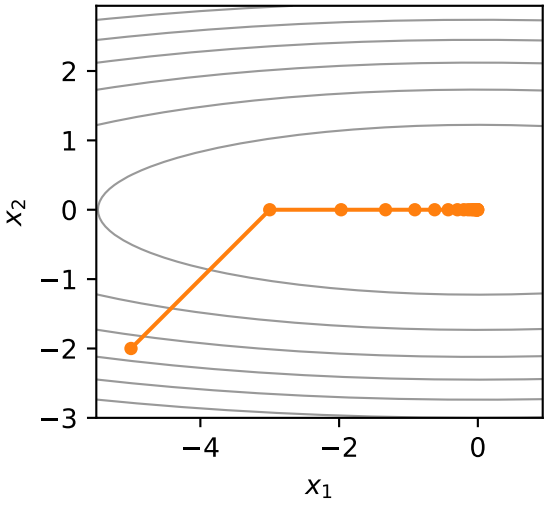
疎または不均一な特徴量を持つモデル（たとえば、頻出トークンは毎回のミニバッチに現れる一方、希少なトークンはごくまれにしか現れないトークン埋め込みなど）では、単一の大域的な学習率ですべての座標に適切に対応することはできない。**AdaGrad** (Duchi et al. 2011) は二乗勾配の累積和 $\mathbf{s}_t$ を要素ごとに蓄え、各座標のステップを $\sqrt{\mathbf{s}_t + \epsilon}$ に反比例させてスケーリングする。

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathbf{g}_t^2, \qquad \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t, \tag{9.27}$$

ここで、$\mathbf{s}_0 = \mathbf{0}$ であり、$\epsilon > 0$ はゼロ除算を防ぐための定数である。座標軸に沿った谷においては、各座標をそれ自身の勾配の大きさでスケーリングすることで方向ごとの曲率（curvature）の差が自動的に打ち消され、9.1.5 節の対角前処理行列に対する勾配のみを用いた近似として機能する（35.5.1 節）。

```
def adagrad_2d(x1, x2, s1, s2, f_grad, eta=2.0, eps=1e-6):
    g1, g2 = f_grad(x1, x2)
    s1, s2 = s1 + g1 ** 2, s2 + g2 ** 2
    x1 -= eta / math.sqrt(s1 + eps) * g1
    x2 -= eta / math.sqrt(s2 + eps) * g2
    return x1, x2, s1, s2

d2l.show_trace_2d(d2l.f_valley, d2l.train_2d(adagrad_2d, d2l.valley_grad))
```

```
step 20, x1: -0.002295, x2: -0.000000
```



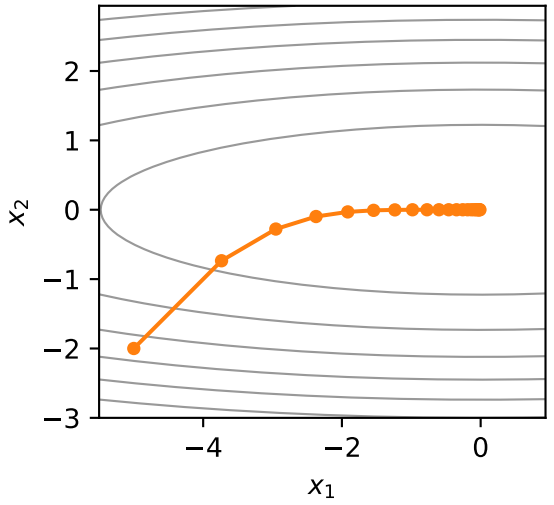
$\mathbf{s}_t$ は二乗勾配を上界なく単調に蓄積するため、持続的な確率的勾配のもとでは AdaGrad の実効ステップサイズが $O(t^{-1/2})$ で減衰し、非凸な深層ネットワークではやがて学習の進行が止まってしまう。**RMSProp** (Tieleman and Hinton 2012) はこの累積和を、二乗勾配の指数移動平均（exponential moving average、EMA）$\mathbf{v}_t$（ウィンドウ幅は $\sim 1/(1-\beta_2)$ ステップ）に置き換える。

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\,\mathbf{g}_t^2, \qquad \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}} \odot \mathbf{g}_t. \tag{9.28}$$

（補足：標準的な最適化の記法に従い、ここでの $\mathbf{v}_t$ は 9.4 節の速度ベクトルではなく、二次モーメントである二乗勾配の平均を表す。）古い勾配を忘却することで、RMSProp は中程度の学習率 $\eta = 0.4$ からでも消失しないステップサイズを維持する。

```
def rmsprop_2d(x1, x2, v1, v2, f_grad, eta=0.4, beta2=0.9, eps=1e-6):
    g1, g2 = f_grad(x1, x2)
    v1 = beta2 * v1 + (1 - beta2) * g1 ** 2
    v2 = beta2 * v2 + (1 - beta2) * g2 ** 2
    x1 -= eta / math.sqrt(v1 + eps) * g1
    x2 -= eta / math.sqrt(v2 + eps) * g2
    return x1, x2, v1, v2

d2l.show_trace_2d(d2l.f_valley, d2l.train_2d(rmsprop_2d, d2l.valley_grad))
```

```
step 20, x1: -0.010599, x2: 0.000000
```



Equation 9.28 の 2 つの性質に注目しよう。

1. **バイアス補正なしでの初期ステップの増幅：**$\mathbf{v}_0 = \mathbf{0}$ であるため、最初のステップでは $\mathbf{v}_1 = (1-\beta_2)\mathbf{g}_1^2$ となり、初回のジャンプが大きさ $\eta/\sqrt{1-\beta_2} \approx 3.16\,\eta$（$\beta_2 = 0.9$ のとき）、さらには $31.6\,\eta$（$\beta_2 = 0.999$ のとき）と過大になる。
2. **正規化されたステップ長：**ある座標の勾配の大きさがほぼ定常であるときは常に $\sqrt{v_t} \approx |g_t|$ となり、更新量の大きさは $\approx \eta$ となって $|g_t|$ がどれほど小さくなっても変わらない。したがって極小値の近傍では、定学習率の適応的手法は $O(\eta)$ の近傍内で振動し続け、スケジュールによって η を減衰させるまで収まらない（9.7 節）。

9.5.1.2 Adam とバイアス補正

Adam (Kingma and Ba 2015) は、一次モーメント（first moment）の EMA（モメンタム、$\mathbf{m}_t$）と非中心二次モーメント（second moment）の EMA（$\mathbf{v}_t$）を組み合わせ、両方の推定量に厳密な**バイアス補正**（bias correction）を適用する。$\mathbf{v}_t = (1-\beta_2)\sum_{s=1}^{t}\beta_2^{t-s}\mathbf{g}_s^2$ を初期値 $\mathbf{v}_0 = \mathbf{0}$ から展開すると、重みの総和は $1-\beta_2^t$ となり、1 にはならないことがわかる。そこで $1-\beta_1^t$ と $1-\beta_2^t$ で割ることにより、不偏モーメント推定量 $\hat{\mathbf{m}}_t$ と $\hat{\mathbf{v}}_t$ が得られる。

$$\begin{aligned} \mathbf{m}_t &= \beta_1\,\mathbf{m}_{t-1} + (1-\beta_1)\,\mathbf{g}_t, & \hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-\beta_1^t}, \\ \mathbf{v}_t &= \beta_2\,\mathbf{v}_{t-1} + (1-\beta_2)\,\mathbf{g}_t^2, & \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1-\beta_2^t}. \end{aligned} \tag{9.29}$$

パラメータ更新では、バイアス補正された一次モーメントを座標ごとの RMS スケール $\sqrt{\hat{\mathbf{v}}_t}$ で割る。

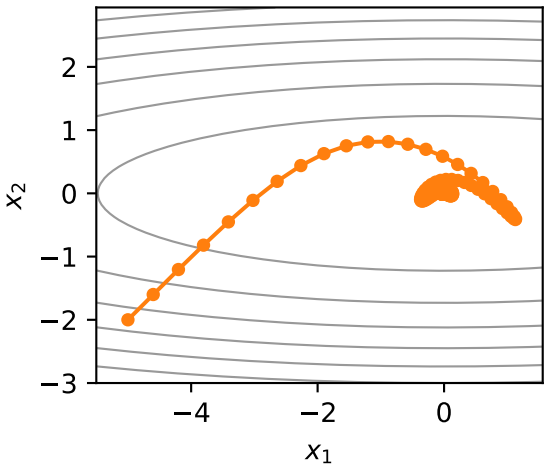
$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}. \tag{9.30}$$

標準的なデフォルト値は $\beta_1 = 0.9$、$\beta_2 = 0.999$（大規模言語モデルの訓練では $\beta_2 = 0.95$ も用いられる (Brown et al. 2020)）、$\epsilon = 10^{-8}$ である。$t = 1$ では $\hat{\mathbf{m}}_1 = \mathbf{g}_1$ かつ $\hat{\mathbf{v}}_1 = \mathbf{g}_1^2$ となるため、初回のステップは厳密に $-\eta\ \mathrm{sign}(\mathbf{g}_1)$ となる。

より一般に、目的関数の任意の座標を正の定数 $c > 0$ で再スケーリングすると、$\hat{m}_{t,i}$ と $\sqrt{\hat{v}_{t,i}}$ の両方が c 倍されるため、それらの比は（$\epsilon \to 0$ において）不変に保たれる。したがって、**Adam のもとでは、η はパラメータ空間における座標ごとの直接的なステップ長を指定する**ことになる。すなわち、勾配の符号が一貫している座標はステップごとに $\approx \eta$ だけ移動する一方、平均ゼロのノイズが支配的な座標は $\approx \eta\sqrt{(1-\beta_1)/(1+\beta_1)} \approx 0.23\,\eta$ だけ移動する。

```
def adam_2d(x1, x2, s1, s2, f_grad, eta=0.4, beta1=0.9, beta2=0.999,
            eps=1e-8):
    def update(x, g, state):  # state: (m, v, t), 0 before the first step
        m, v, t = state if state else (0, 0, 0)
        m, v = beta1 * m + (1 - beta1) * g, beta2 * v + (1 - beta2) * g ** 2
        m_hat, v_hat = m / (1 - beta1 ** (t + 1)), v / (1 - beta2 ** (t + 1))
        return x - eta * m_hat / (math.sqrt(v_hat) + eps), (m, v, t + 1)
    g1, g2 = f_grad(x1, x2)
    (x1, s1), (x2, s2) = update(x1, g1, s1), update(x2, g2, s2)
    return x1, x2, s1, s2

path = d2l.train_2d(adam_2d, d2l.valley_grad, steps=200, beta2=0.999)
d2l.show_trace_2d(d2l.f_valley, path)
```

```
step 200, x1: 0.000022, x2: 0.000042
```



Adam をゼロから実装するには、各パラメータテンソルに対してステップカウンタ t と 2 つのバッファ $(\mathbf{m}, \mathbf{v})$ を保持する必要がある。

```
def scale_by_adam(b1=0.9, b2=0.999, eps=1e-8):
    """Adam's direction m_hat / (sqrt(v_hat) + eps), from scratch."""
    def init(params):
        zeros = jax.tree.map(jnp.zeros_like, params)
        return dict(t=jnp.zeros((), jnp.int32), m=zeros, v=zeros)

    def update(grads, state, params=None):
        t = state['t'] + 1
```

```
        m = jax.tree.map(lambda m, g: b1 * m + (1 - b1) * g,
                         state['m'], grads)
        v = jax.tree.map(lambda v, g: b2 * v + (1 - b2) * g ** 2,
                         state['v'], grads)
        def direction(m, v):
            return m / (1 - b1 ** t) / (jnp.sqrt(v / (1 - b2 ** t)) + eps)
        return jax.tree.map(direction, m, v), dict(t=t, m=m, v=v)
    return optax.GradientTransformation(init, update)

def adam(learning_rate):
    return optax.chain(scale_by_adam(), optax.scale(-learning_rate))
```

9.5.2 言語モデルにおける Adam

9.5.2.1 TinyLM における勾配スケールと SGD との比較

なぜ Transformer にとって座標ごとのスケーリングがこれほど不可欠なのだろうか。TinyLM（9.3.1 節）全体の勾配スケールを測定するため、まず WikiText103 データセット（$b = 64$、ウィンドウ長 64）を読み込む。

```
data = d2l.WikiText103(batch_size=64, num_steps=64)
```

次に、各パラメータテンソルの二乗平均平方根（RMS）勾配を 10 ミニバッチにわたって計算する gradients(model, data)、パラメータ値を複製する snapshot(model)、および初期状態の TinyLM インスタンスと Adam（$\eta = 3 \times 10^{-3}$）を定義する。

```
def flat(state):  # {'blks.0.qkv.kernel': array, ...}
    return {'.'.join(map(str, path)): v[...]
            for path, v in nnx.to_flat_state(state)}

@nnx.jit
def gradient(model, X, Y):
    loss = lambda model: optax.softmax_cross_entropy_with_integer_labels(
        model(X), Y).mean()
    return nnx.grad(loss)(model)

def gradients(model, data, num_batches=10):
    """RMS over num_batches minibatches of each parameter's gradient."""
    sq = {}
    for _, (X, Y) in zip(range(num_batches), data.train_dataloader()):
        for name, g in flat(gradient(model, jnp.asarray(X),
                                     jnp.asarray(Y))).items():
            sq[name] = sq.get(name, 0) + g ** 2 / num_batches
    return {name: jnp.sqrt(v) for name, v in sq.items()}

def snapshot(model):
    return flat(nnx.state(model, nnx.Param))

model = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(d2l.get_seed()))
optimizer = nnx.Optimizer(model, optax.adam(3e-3), wrt=nnx.Param)
```

テンソルを 8 種類の機能別パラメータ群に `rms_by_kind` で分類し、ステップ 0 とステップ 500 における RMS 勾配の大きさを、正規化された Adam のステップ長 $\|\Delta\mathbf{x}\|_{\mathrm{RMS}}/\eta$ と並べて比較する。

```
KINDS = ['token_emb', 'pos_emb', 'qkv', 'proj', 'mlp', 'head']

def rms_by_kind(tensors):
    """RMS of the entries of each kind of parameter, from {name: tensor}."""
    sums, counts = {}, {}
    for name, t in tensors.items():
        kind = (('LayerNorm' if 'norm' in name else 'biases')
                if len(t.shape) == 1 else
                next(k for k in KINDS if k in name))
        sums[kind] = sums.get(kind, 0) + float((t ** 2).sum())
        counts[kind] = counts.get(kind, 0) + math.prod(t.shape)
    return {kind: math.sqrt(sums[kind] / counts[kind]) for kind in sums}

grad_init = rms_by_kind(gradients(model, data))
d2l.train_steps(model, data, optimizer, num_steps=500)
grad_500 = rms_by_kind(gradients(model, data))
before = snapshot(model)
d2l.train_steps(model, data, optimizer, num_steps=1)
after = snapshot(model)
step = rms_by_kind({name: (before[name] - after[name]) / 3e-3
                    for name in before})
print(f'{"":<11}{"RMS gradient at step 0":>24}{"at step 500":>13}'
      f'{"Adam step / lr":>16}')
for kind in KINDS + ['LayerNorm', 'biases']:
    print(f'{kind:<11}{grad_init[kind]:>24.1e}{grad_500[kind]:>13.1e}'
          f'{step[kind]:>16.2f}')
ratio = lambda column: max(column.values()) / min(column.values())
print(f'{"max / min":<11}{ratio(grad_init):>24.0f}{ratio(grad_500):>13.0f}'
      f'{ratio(step):>16.1f}')
```

```
            RMS gradient at step 0  at step 500  Adam step / lr
token_emb                  1.9e-04      8.1e-05            0.20
pos_emb                    1.8e-04      1.4e-04            0.28
qkv                        7.3e-04      3.6e-04            0.23
proj                       1.3e-03      5.7e-04            0.21
mlp                        9.8e-04      2.5e-04            0.24
head                       4.4e-03      1.3e-03            0.13
LayerNorm                  4.8e-03      1.2e-03            0.22
biases                     7.0e-03      6.0e-04            0.18
max / min                       39           16             2.2
```

生の勾配の大きさは、初期化時点で $40\times$ **の範囲**（トークン埋め込みの $\sim 2\times10^{-4}$ からバイアスの $\sim 7\times10^{-3}$ まで）に及び、$15\times$ **の範囲**がステップ 500 の時点でも残る。SGD のもとでは、単一のスカラー学習率によって勾配が最大のグループ（バイアス、LayerNorm、出力ヘッド）が埋め込みよりも 15–40× 速く動かされる。一方 Adam のもとでは、8 つすべてのパラメータグループにわたる正規化されたステップサイズは狭い $2\times$ の範囲内（0.1η–0.3η であり、純粋なノイズに対する比 $\sqrt{(1-\beta_1)/(1+\beta_1)} \approx 0.23$ に近い）に収まる。

`TinyLM` の勾配スケールは 10^{-4}–10^{-2} 付近にあり、かつ Adam の EMA には $(1-\beta_1)=0.1$ が含まれるため、$\eta\sim3\times10^{-3}$ の Adam は、モメンタム 0.9 かつ $\eta\sim0.3$ の SGD と同程度の大

きさのパラメータステップを生み出す。共通の 3 つのシードにわたって両オプティマイザを比較するため、まずモデルとオプティマイザを生成するクロージャ `make_lm`、`make_sgd`、`make_adam` を用意する。

```
make_lm = lambda rngs: d2l.TinyLM(len(data.vocab), rngs=rngs)
make_sgd = lambda lr: optax.sgd(lr, momentum=0.9)
make_adam = optax.adam
seeds = [d2l.get_seed() for _ in range(3)]
```

続いて、シードごとに同一の初期重みから出発し、モメンタム付き SGD を $\eta \in \{0.1, 0.3, 0.5, 1.0\}$ にわたって、Adam を $\eta \in \{3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, 10^{-2}, 3 \times 10^{-2}\}$ にわたってスイープする。

```
def sweep(make_optimizer, lrs):
    """Loss curves of one run per learning rate and seed, printing the final
    losses; the runs of one seed start from the same initialization."""
    runs = [d2l.lr_sweep(make_lm, make_optimizer, data, lrs, seed,
                         verbose=False)[0] for seed in seeds]
    for lr in lrs:
        print(f'  lr {lr:<7g}', *(f'{d2l.final_loss(run[lr]):6.3f}'
                                  for run in runs))
    return {lr: [run[lr] for run in runs] for lr in lrs}

print('final loss of three runs, SGD with momentum')
sgd_lm = sweep(make_sgd, [0.1, 0.3, 0.5, 1.0])
print('Adam')
adam_lm = sweep(make_adam, [3e-4, 1e-3, 3e-3, 1e-2, 3e-2])
```

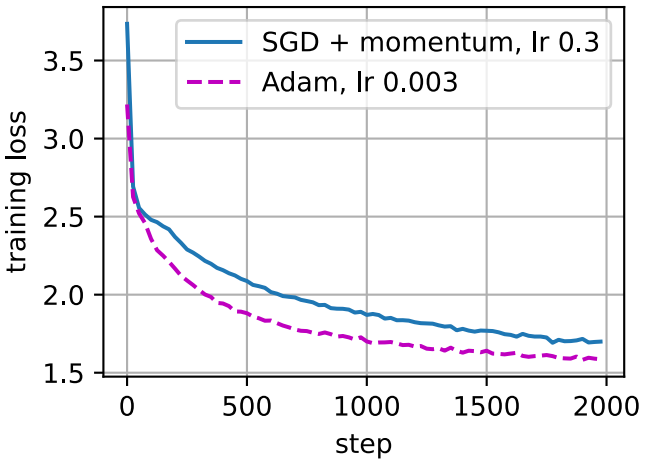
```
final loss of three runs, SGD with momentum
  lr 0.1      1.865  1.841  1.866
  lr 0.3      1.702  1.667  1.670
  lr 0.5        inf    inf    inf
  lr 1          inf    inf    inf
Adam
  lr 0.0003   1.930  1.970  1.939
  lr 0.001    1.660  1.661  1.657
  lr 0.003    1.589  1.589  1.586
  lr 0.01     1.634  1.644  1.644
  lr 0.03     2.729  2.766  2.788
```

各オプティマイザを最良の学習率（`best_rate`）で集計し、実用可能な学習率の範囲（最良値から 0.15 nats 以内）、1.8 nats に到達するまでのステップ数、1 ステップでの最大の損失スパイク、および平滑化された損失の軌跡を比較する。

```
def best_rate(curves):
    """The lr with the lowest mean final loss, and each lr's mean."""
    mean = {lr: sum(map(d2l.final_loss, runs)) / len(runs)
            for lr, runs in curves.items()}
    return min(mean, key=mean.get), mean

print(f'{"":6}{"best lr":>8}{"final loss":>17}   {"within 0.15 nats":<18}'
```

```
        f'{"steps to 1.8":<16}largest one-step rise')
for name, curves in [('SGD', sgd_lm), ('Adam', adam_lm)]:
    best, mean = best_rate(curves)
    runs = curves[best]
    finals = [d2l.final_loss(run) for run in runs]
    usable = [f'{lr:g}' for lr in curves if mean[lr] <= mean[best] + 0.15]
    steps = ' '.join(str(d2l.steps_to_reach(run, 1.8)) for run in runs)
    rises = ' '.join(f'{max(b - a for a, b in zip(run, run[1:])):.2f}'
                     for run in runs)
    print(f'{name:<6}{best:>8g}{min(finals):>8.3f} to {max(finals):.3f}   '
          f'{" ".join(usable):<18}{steps:<16}{rises}')
best_sgd, best_adam = best_rate(sgd_lm)[0], best_rate(adam_lm)[0]
d2l.plot(list(range(0, 2000, 25)),
         [d2l.smooth(sgd_lm[best_sgd][0]),
          d2l.smooth(adam_lm[best_adam][0])],
         'step', 'training loss',
         legend=[f'SGD + momentum, lr {best_sgd:g}',
                 f'Adam, lr {best_adam:g}'])
```

```
       best lr       final loss   within 0.15 nats  steps to 1.8    largest one-step rise
SGD        0.3   1.667 to 1.702   0.3               1350 1225 1250  1.07 0.90 0.52
Adam     0.003   1.586 to 1.589   0.001 0.003 0.01  700 750 775     0.16 0.16 0.19
```



TinyLM において、Adam は目標損失 1.8 nats にモメンタム付き SGD のほぼ半分のステップ数（650–800 対 1,100–1,500）で到達し、より低い最終損失で訓練を終える。さらに 10× の実用可能な学習率ウィンドウ（10^{-3}〜10^{-2}）で動作し、$\eta = 3 \times 10^{-2}$ においてさえ正規化された更新長が η で上から抑えられるため有限に留まる。一方、SGD の最良学習率 $\eta = 0.3$ は発散のわずか 1 グリッド手前に位置しており、1 ステップでの最悪の損失スパイクは Adam の数倍に達し、$\eta = 0.5$ では大半の実行が `inf` へと発散する。サブワード語彙を持つより大規模な Transformer では、ブロック単位のヘッセ行列の不均一性 (Y. Zhang et al. 2024) と裾の重いトークン頻度 (Kunstner et al. 2023, 2024) によって、この差はさらに拡大する。

9.5.2.2 ϵ の役割と二次モーメント推定の限界

Equation 9.30 において、ϵ は座標ごとの最大実効学習率を η/ϵ で頭打ちにする。$\epsilon \gg \sqrt{\hat{v}_{t,i}}$ の極限では、Adam はステップサイズ $(1-\beta_1)\eta/\epsilon$ のモメンタム付き SGD に帰着する。逆に β_2 が小さすぎると、記憶の短い二次モーメント推定（second moment estimate）が、まれに生じる逆向きの大きな勾配よりも、頻繁に現れる小さな勾配に大きな重みを与えてしまうことがある。Reddi et al. (2018) はこの現象を利用して人工的な周期凸反例（$+4, -1, -1$）を構成し、小さな β_2 を持つ Adam が収束しないことを示した（35.5.2.2 節、演習問題 4）。実用上は $\beta_2 \in [0.95, 0.999]$ に保つことで、単調な累積最大値（AMSGrad）を導入しなくてもこの病理を防ぐことができる。

9.5.3 分離型重み減衰（AdamW）

9.5.3.1 結合型 ℓ_2 正則化と分離型重み減衰

単純な SGD（2.7 節）のもとでは、目的関数に ℓ_2 ペナルティ $\frac{\lambda}{2}\|\mathbf{x}\|^2$ を加えることは勾配に $\lambda \mathbf{x}_t$ を加えることになり、これは各ステップの前に重みを $(1-\eta\lambda)$ 倍に縮小（shrinkage）することと数学的に同一である。

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\,(\mathbf{g}_t + \lambda \mathbf{x}_t) = (1-\eta\lambda)\,\mathbf{x}_t - \eta\,\mathbf{g}_t. \tag{9.31}$$

しかし、ペナルティ付きの勾配 $\mathbf{g}_t + \lambda \mathbf{x}_t$ を Adam に入力すると、ペナルティ項まで座標ごとの前処理行列 $\sqrt{\hat{v}_{t,i}} + \epsilon$ で割られてしまう。

$$\underbrace{\frac{\eta\,\lambda}{\sqrt{\hat{v}_{t,i}} + \epsilon}\,x_{t,i}}_{\text{coupled } \ell_2 \text{ penalty in Adam}} \qquad \text{versus} \qquad \underbrace{\eta\,\lambda\,x_{t,i}}_{\text{decoupled weight decay}} \quad . \tag{9.32}$$

結合型 ℓ_2 正則化のもとでは、損失勾配の小さい重みは勾配の大きい重みよりも最大 $10^3\times$ も強く減衰させられる。あるいは、$\lambda|x_{t,i}|$ が損失勾配に対して支配的になると、Adam はペナルティをそれ自身の大きさで正規化し、重みをゼロへ向けてステップあたり速度 η で押し縮め、$\lambda|x| \sim |g_{\text{loss}}|$ になるまでそれが続く。

Loshchilov and Hutter (2019) は、重み減衰を適応的な勾配ステップから分離した **AdamW** を導入した。

$$\mathbf{x}_{t+1} = (1-\eta\lambda)\,\mathbf{x}_t - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}, \tag{9.33}$$

ここで、$\hat{\mathbf{m}}_t$ と $\hat{\mathbf{v}}_t$ はデータ損失の勾配 $\mathbf{g}_t$ のみから計算される。

```
def adamw(learning_rate, weight_decay):
    return optax.chain(optax.scale_by_adam(),
                       optax.add_decayed_weights(weight_decay),
                       optax.scale(-learning_rate))
```

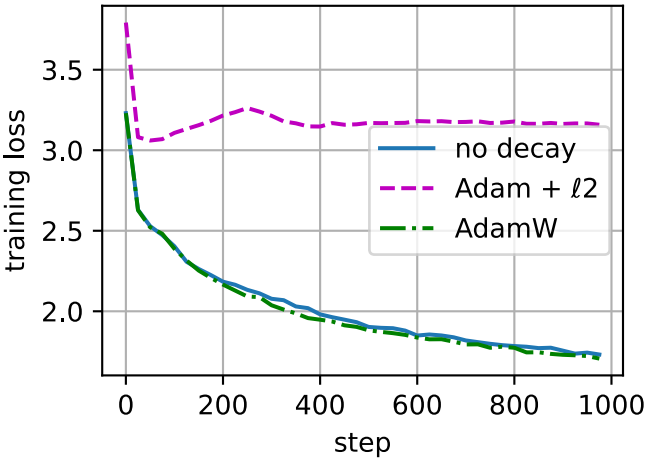
減衰なし、結合型の `Adam + l2`、および分離型の `AdamW` を `TinyLM` 上で比較するため、まず埋め込み以外の 2D 重み行列の要素ごとの RMS を測る `matrix_rms` と、ステップごとの損失および行列の RMS を 250 ステップごとに記録する `train_tracked` を定義する。

```
def matrix_rms(model):
    """RMS of the entries of the weight matrices, embeddings excluded."""
    leaves = jax.tree_util.tree_leaves_with_path(nnx.state(model, nnx.Param))
    W = jnp.concatenate([p.ravel() for path, p in leaves
                         if p.ndim == 2 and 'emb' not in str(path)])
    return float(jnp.sqrt((W ** 2).mean()))

def train_tracked(tx, data, seed, num_steps, every=250):
    """Train TinyLM from seed with the optax transformation tx; return the
    losses, the matrix RMS at the start and every `every` steps, the model."""
    model = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(seed))
    optimizer = nnx.Optimizer(model, tx, wrt=nnx.Param)
    losses, rms = [], [matrix_rms(model)]
    for _ in range(num_steps // every):
        losses += d2l.train_steps(model, data, optimizer, every)
        rms.append(matrix_rms(model))
    return losses, rms, model
```

共通のシードから 3 つの設定すべてを 1,000 ステップ、$\eta = 3 \times 10^{-3}$ および $\lambda = 0.1$ で実行すると、結合型 ℓ_2 ペナルティと分離型重み減衰（decoupled weight decay）の劇的な違いが現れる。

```
def coupled(lr, wd):  # The penalty's gradient enters Adam's moments
    return optax.chain(optax.add_decayed_weights(wd), optax.adam(lr))

seed = d2l.get_seed()
curves = {}
for name, tx in [('no decay', optax.adam(3e-3)),
                 ('Adam + ℓ2', coupled(3e-3, 0.1)),
                 ('AdamW', optax.adamw(3e-3, weight_decay=0.1))]:
    curves[name], rms, _ = train_tracked(tx, data, seed, 1000)
    print(f'{name:<10} final loss {d2l.final_loss(curves[name]):.3f}, RMS of '
          f'the weight matrices {rms[0]:.4f} -> {rms[-1]:.4f}')
d2l.plot(list(range(0, 1000, 25)), [d2l.smooth(c) for c in curves.values()],
         'step', 'training loss', legend=list(curves))
```

```
no decay   final loss 1.742, RMS of the weight matrices 0.0447 -> 0.1104
Adam + ℓ2  final loss 3.164, RMS of the weight matrices 0.0447 -> 0.0027
AdamW      final loss 1.721, RMS of the weight matrices 0.0447 -> 0.0999
```



結合型の `Adam + l2`（$\lambda = 0.1$）は隠れ層の重み行列を初期 RMS の 0.045 から < 0.003 まで崩壊させ、ネットワークの表現を破壊してユニグラムのバイト頻度エントロピー（~ 3.2 nats/byte）で停滞してしまう。一方、分離型の `AdamW` は正則化なしのベースラインと並んで順調に訓練が進みつつ、重みの増大を抑え込んでいる。

9.5.3.2 減衰のタイムスケールと平衡ノルム

Adam の正規化された更新方向を $\mathbf{u}_t = \widehat{\mathbf{m}}_t / (\sqrt{\widehat{\mathbf{v}}_t} + \epsilon)$ と書き、Equation 9.33 を t ステップにわたって展開すると次式が得られる。

$$\mathbf{x}_t = (1 - \eta\lambda)^t \, \mathbf{x}_0 - \eta \sum_{s=0}^{t-1} (1 - \eta\lambda)^{t-1-s} \, \mathbf{u}_s. \tag{9.34}$$

したがって、AdamW はパラメータ $\mathbf{x}_t$ を、減衰係数 $1 - \eta\lambda \approx e^{-\eta\lambda}$ を持つ過去の更新の指数加重移動平均として表しており、その特性**減衰タイムスケール**（decay timescale）は次式で与えられる。

$$\tau = \frac{1}{\eta\lambda}$$

単位はステップ数である (Wang and Aitchison 2025)。$\eta = 3 \times 10^{-3}$、$\lambda = 0.1$ のとき、減衰タイムスケールは $\tau = 1/0.0003 \approx 3{,}333$ ステップとなる。T ステップの訓練全体では、重み減衰は $T/\tau = T\eta\lambda$ タイムスケールにわたって作用する。

$\tau = 1/(\eta\lambda)$ より長い訓練では、初期重みは減衰して消え去り、各座標 $x_{t,i}$ は確率的な**平衡ノルム**（equilibrium norm）に到達する。そこではステップごとの縮小量 $\eta\lambda x_{t,i}$ と正規化された更新 $-\eta n_{t,i}$（ただし $n_{t,i} = g_{t,i}/\sqrt{\widehat{v}_{t,i}}$ は二乗平均が単位大きさ $\sigma^2 \approx 1$）の注入とが釣り合う。SGD のノイズ球（noise ball、9.2 節）において曲率を λ_{decay} に置き換えたものとまったく同じ分散の漸化式により、次式が成り立つ。

$$\begin{aligned} \mathbb{E}[x_{t+1}^2] &= (1-\eta\lambda)^2\,\mathbb{E}[x_t^2] + \eta^2\sigma^2 \\ \implies \quad \mathbb{E}[x_\infty^2] &\approx \frac{\eta\,\sigma^2}{2\lambda} \approx \frac{\eta}{2\lambda}. \end{aligned} \tag{9.35}$$

重みの二乗平均平方根の大きさが $\mathrm{RMS}(\mathbf{x}_\infty) \propto \sqrt{\eta/(2\lambda)}$ に安定することで、正規化された重み行列の**相対角度更新率**（relative angular update rate）も $\eta/\mathrm{RMS}(\mathbf{x}_\infty) \propto \sqrt{2\eta\lambda}$ に安定する（**回転平衡**、Kosson et al. (2024)）。これにより、正則化なしの重みが制御不能な $\sqrt{t}$ のランダムウォーク成長を起こし、長期の事前学習において実効的な角度ステップサイズが $1/\sqrt{t}$ で減衰してしまう事態が防がれる (D'Angelo et al. 2024)。η をスイープする際には、ステップごとの縮小率 $s = \eta\lambda$（したがってタイムスケール $\tau = 1/s$）を固定しておけば、学習率の探索を減衰タイムスケールから切り離すことができる (Loshchilov and Hutter 2019; Wang and Aitchison 2025; Bergsma et al. 2025)。

9.5.3.3 重み減衰の対象外となるパラメータとオプティマイザ状態のメモリ

標準的な Transformer の訓練レシピでは、重み行列に分離型重み減衰を適用し、バイアスと正規化パラメータは対象外とする。OLMo 2 はさらに埋め込みテーブルも対象外としており、本書でもその選択を採用する。その結果、重み減衰は **2D の内部射影行列のみ**（TinyLM の 90%）に適用され、次の 2 つのパラメータグループは対象外（$\lambda = 0$）となる。

1. **1D のバイアスおよび LayerNorm・RMSNorm のゲイン・シフトベクトル：**正規化のスケールは 1 付近から始まり、特徴量の分散を直接制御する。これらをゼロへ向けて減衰させると、角度更新速度を調整するどころか層の活性化値を歪めてしまう。
2. **トークン埋め込みテーブルと位置埋め込みテーブル（OLMo 2 での除外）：**トークン埋め込みの各行は、そのトークンがミニバッチに現れたときにしか勾配を受け取らないのに対し、分離型重み減衰は毎ステップすべての行を縮小するため、希少なトークンの表現を崩壊させ、下流の LayerNorm の勾配を膨張させてしまう (Team OLMo et al. 2025)。

```
model = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(0))

def decay_mask(params):  # True for the matrices, False for the rest
    return jax.tree_util.tree_map_with_path(
        lambda path, p: p.ndim == 2 and 'emb' not in str(path), params)

optimizer = nnx.Optimizer(
    model, optax.adamw(3e-3, weight_decay=0.1, mask=decay_mask),
    wrt=nnx.Param)
params = nnx.state(model, nnx.Param)
n_decay = sum(int(p.size) for p, m in zip(
    jax.tree.leaves(params), jax.tree.leaves(decay_mask(params))) if m)
n = sum(int(p.size) for p in jax.tree.leaves(params))
print(f'decayed: {n_decay} of {n} parameters')
```

```
decayed: 425984 of 470784 parameters
```

9.5.3.4 オプティマイザ状態のメモリ

最後に、パラメータごとに 2 つのバッファ $(\mathbf{m}_t, \mathbf{v}_t)$ を保持することで、静的なパラメータのメモリ使用量は 2 倍になる。32 ビット浮動小数点（スカラーあたり 4 バイト）では、訓練可能な各パラメータに 16 **バイト**のデバイスメモリが必要になる。内訳は、4 バイトがパラメータ $\mathbf{x}$、4 バイ

トが勾配 **g**、そして 8 バイトが AdamW の状態 $(\mathbf{m}, \mathbf{v})$ である（13.4 節）。パラメータ数が 10^9 のモデルでは、活性化値を一切保存する前の段階で、オプティマイザ状態だけで 8 GB を消費する。パラメータを凍結する（オプティマイザから除外する）か、$(\mathbf{m}, \mathbf{v})$ を量子化または因子分解することで、このパラメータあたり 8 バイトのオーバーヘッドを直接解消または圧縮できる。

AdamW は重みの各要素を独立に正規化するが、2D 重み行列は活性化値のベクトルに作用する線形作用素として働く。9.6 節では、各 2D 重み行列の更新を要素ごとではなく作用素全体として正規化することで、さらに優れた結果が得られるかを問う。

9.5.4 まとめ

AdaGrad は各座標を累積二乗勾配の平方根に反比例させてスケーリングし、**RMSProp** はその上界のない和を忘却を伴う指数移動平均 $\mathbf{v}_t$ に置き換え、**Adam** はさらに一次モーメントのモメンタム $\mathbf{m}_t$ とバイアス補正 $1-\beta_1^t$、$1-\beta_2^t$ を加える。$\hat{\mathbf{m}}_t$ を $\sqrt{\hat{\mathbf{v}}_t}$ で割ることで、Adam は勾配の対角再スケーリングに対して不変となり、η をパラメータ空間における座標ごとの直接的なステップ長へと変え、生の勾配が 15–40× も異なる Transformer の各パラメータグループ間で更新量の大きさを均一化する。**AdamW** は重み減衰 $\mathbf{x}_{t+1} = (1-\eta\lambda)\mathbf{x}_t - \eta\hat{\mathbf{m}}_t/(\sqrt{\hat{\mathbf{v}}_t}+\epsilon)$ を適応的な前処理行列から分離し、一様な減衰タイムスケール $\tau = 1/(\eta\lambda)$、$\sqrt{\eta/(2\lambda)}$ に比例する平衡時の重み RMS、および $\sqrt{2\eta\lambda}$ に比例する定常的な相対角度ステップを確立する。標準的なレシピでは 2D 重み行列のみを減衰させて 1D のバイアスと正規化パラメータ（さらに OLMo 2 に従えば埋め込みテーブル）を対象外とし、$(\mathbf{m}_t, \mathbf{v}_t)$ は `float32` における訓練可能なパラメータあたり 8 バイト（全 16 バイト中）を占める。

9.5.5 演習問題

1. [code] **座標回転と対角スケール不変性**。
    1. `adam_2d` を `d2l.f_valley` 上で実行し、目的関数を $c > 0$（ただし $c \gg \epsilon$）で再スケーリングしても、また λ_2 を 4 から 400 に変更しても、Adam が不変であることを検証せよ。
    2. 谷を 45° 回転させて $f(\mathbf{x}) = 0.05(x_1+x_2)^2 + (x_1-x_2)^2$ とせよ。`adagrad_2d`、`rmsprop_2d`、`adam_2d` を $\eta = 0.4$ で 2,000 ステップにわたり回転前と回転後の谷（および $\lambda_2 = 400$ の場合）で実行し、固有基底を座標軸から回転させると対角前処理行列に対して悪条件性が復活する理由を説明せよ。
2. [code] **定学習率のリミットサイクルと AMSGrad**。●
    1. Adam のもとで 1D 座標が $x_t = (-1)^t a$ のように $f(x) = \frac{\lambda}{2}x^2$ 上で交互に振動するとき、その振幅が $a = \frac{1-\beta_1}{1+\beta_1}\frac{\eta}{2}$ となり λ に依存しないことを示せ。
    2. `adam_2d` を `f_valley` 上で 20,000 ステップ実行し、反復点が原点の 10^{-40} 以内まで近づく（$\hat{v}_t$ が初期の勾配を記憶している間）にもかかわらず、$\hat{v}_t$ がモメンタムの安定性しきい値を下回って減衰すると弾き出される理由を説明せよ。また、$\hat{v}_t$ を $\max_{s\le t}\hat{v}_s$（AMSGrad）に置き換えるとこの弾き出しが防げることを検証せよ。
3. [extended] **頻出トークン・希少トークンとグループ別学習率**。
    1. Kunstner et al. (2024) に従い、256 個のバイト値を `WikiText103` における出現頻度に基づいて頻出トークン（$\ge 0.1\%$）と希少トークン（$< 0.1\%$）に分割し、チューニングされたモメンタム付き SGD、Adam、およびモメンタム付き符号降下法（$\eta\ \mathrm{sign}(\mathbf{m}_t)$）のもとで各グループの検証損失を比較せよ。
    2. モメンタム付き SGD のもとで `TinyLM` の 3 つのパラメータグループ（埋め込み、2D 行列、1D ベクトル）に個別の学習率を割り当て、SGD と Adam の差のうちどれだけが

グループ単位のスケーリングと座標単位のスケーリングのそれぞれによって縮まるかを測定せよ。

4. [code] **敵対的な周期勾配と二次モーメントの修正手法**。周期 3 の系列として、$f_t(x) = 4x$（$t \equiv 1 \pmod 3$ のとき）およびそれ以外では $f_t(x) = -x$ と定義された $x \in [-1, 1]$ 上の関数列において (Reddi et al. 2018)、$\beta_1 = 0, \beta_2 = 1/17, \eta_t = 0.5/\sqrt{t}$ で Adam を実行し、AMSGrad および Yogi (Zaheer et al. 2018) と比較せよ。

5. **AdamW の 2 つの極限と回転平衡**。

   1. $x_{t+1} = (1 - \eta\lambda)x_t - \eta n_t$ について、決定論的な平衡 $|x_\infty| = 1/\lambda$（$n_t = -1$ が定数のとき）、平均ゼロのノイズ平衡 $\mathbb{E}[x_\infty^2] \approx \eta/(2\lambda)$（$n_t$ が独立同分布で $\mathbb{E}[n_t] = 0, \mathbb{E}[n_t^2] = 1$ のとき）、および混合平衡 $\mathbb{E}[x_\infty^2] \approx \bar{n}^2/\lambda^2 + \eta/(2\lambda)$（$n_t = \bar{n} + \xi_t$ のとき）を導出せよ。
   2. $\mathbf{x}_{t+1} = (1 - \eta\lambda)\mathbf{x}_t - \eta\mathbf{u}_t$ を $\mathbb{R}^{1000}$ 上で $\mathbf{u}_t \perp \mathbf{x}_t$ の条件のもとでシミュレーションし (Kosson et al. 2024)、平衡時の RMS が $\sqrt{\eta/(2\lambda)}$ に一致し、ステップごとの回転角が $\sqrt{2\eta\lambda}$ に一致することを検証せよ。

6. [code] (η, λ) **の同時探索とパラメータグループの除外**。●

   1. `TinyLM` を $8{,}000$ 個の重複ウィンドウ（`d2l.TimeMachine`）上で 800 ステップにわたり、$\eta \in \{10^{-3}, 3 \times 10^{-3}, 10^{-2}\}$ と λ のグリッド全体で訓練し、λ によるパラメータ化とステップごとの縮小率 $s = \eta\lambda$ によるパラメータ化を比較せよ。
   2. `WikiText103` 上の検証パープレキシティを $\lambda \in \{0.5, 1.0\}$ において、すべてのパラメータを減衰させる場合と 2D 重み行列のみを減衰させる場合（9.5.3.3 節）とで比較し、改善の大部分をもたらしているのが埋め込みの除外なのか 1D ベクトルの除外なのかを特定せよ。

9.6 Muon

これまでに見てきたすべてのオプティマイザ（SGD、モメンタム、AdamW）は、ニューラルネットワークのパラメータを構造のない平坦なベクトルとして扱っている。AdamW でさえ、2 つのスカラー平均 $(m_{t,ij}, v_{t,ij})$ を各要素 W_{ij}（重み行列 $\mathbf{W}$ の成分）に対して独立に保持しているにすぎない。仮に $\mathbf{W}$ の要素を任意に並べ替えたとしても、AdamW はまったく同じ更新ステップを踏むことになる。

しかし、Transformer のパラメータの 80% 以上は 2 次元の隠れ射影行列 $\mathbf{W} \in \mathbb{R}^{m \times n}$ にあり、これらの行列は層の活性化値 $\mathbf{x} \in \mathbb{R}^n$ を $\mathbf{y} = \mathbf{W}\mathbf{x} \in \mathbb{R}^m$ へ写す線形作用素として働く。重み行列を**線形作用素とみなしたとき**、その最急降下（steepest descent）の方向はどうなるだろうか。

Bernstein and Newhouse (2024) が強調するように、最急降下は更新ステップを抑えるために選ぶノルムに対して相対的に定義される。ステップをユークリッド（ℓ_2）ノルムで測れば SGD の生の勾配方向が得られ、座標ごとの ℓ_∞ ボックスノルムで測れば Adam の記憶なし極限である符号降下法（sign descent）が得られる。その代わりに、更新 $\Delta\mathbf{W}$ が層の出力にもたらし得る最大変化量 $\|\Delta\mathbf{W}\mathbf{x}\|_2 / \|\mathbf{x}\|_2$、すなわち**スペクトルノルム** $\|\Delta\mathbf{W}\|_2$ を抑えることにすると、最急降下方向は勾配行列のすべての非ゼロ特異値を 1 に置き換えたものとなり、**直交化勾配**（orthogonalized gradient）$\mathbf{U}\mathbf{V}^\top$ が得られる。

各ステップで厳密な特異値分解（singular value decomposition、SVD）を計算するのは GPU にとって遅すぎるが、5 ステップの奇の行列多項式（**ニュートン・シュルツ反復**、Newton–Schulz iteration）を用いれば、行列積のみを用いて $\mathbf{U}\mathbf{V}^\top$ を近似できる。これを隠れ層の 2 次元重み行列に対するネステロフのモメンタムバッファに適用し（埋め込み、1 次元パラメータ、出力ヘッドは AdamW に任せる）、得られるのが **Muon**（**Momentum Orthogonalized by Newton–Schulz**、

Jordan et al. (2024)）である。本節では、ユークリッドノルム球、ℓ_∞ ノルム球、スペクトルノルム球における最急降下を導出し、調整済みの 5 次ニュートン・シュルツ反復を解析したうえで、ハイブリッドな Muon+AdamW オプティマイザをゼロから実装し、`TinyLM` 上で AdamW とベンチマーク比較する。

9.6.1 ノルムと最急降下

9.6.1.1 ノルム球における最急降下

1 次テイラー展開 $f(\mathbf{x}+\mathbf{d}) \approx f(\mathbf{x}) + \langle \mathbf{g}, \mathbf{d} \rangle$（ただし勾配は $\mathbf{g} = \nabla f(\mathbf{x})$）が与えられたとき、半径 $\eta > 0$、ノルム $\|\cdot\|$ のもとでの最急降下ステップは次の最適化問題を解くことで得られる。

$$\mathbf{d}^\star = \operatorname*{argmin}_{\|\mathbf{d}\| \le \eta} \langle \mathbf{g}, \mathbf{d} \rangle. \tag{9.36}$$

ノルムの幾何構造が異なれば、まったく異なる更新方向が導かれる (Bernstein and Newhouse 2024)（Figure 9.8 および Table 9.3）。

1. **ユークリッド球（$\|\mathbf{d}\|_2 \le \eta$）：**コーシー・シュワルツの不等式により、最小点は負の勾配と同じ方向を向き、$\mathbf{d}^\star = -\eta\, \mathbf{g}/\|\mathbf{g}\|_2$ となって線形化された目的関数を $\eta\|\mathbf{g}\|_2$ だけ減少させる（SGD の方向）。
2. **座標ボックス（$\|\mathbf{d}\|_\infty = \max_i |d_i| \le \eta$）：**$\sum_i g_i d_i$ を制約 $|d_i| \le \eta$ のもとで最小化すると、すべての座標がその境界まで押し出されて $\mathbf{d}^\star = -\eta\, \operatorname{sign}(\mathbf{g})$ となり、線形化された目的関数は双対な ℓ_1 ノルム $\eta\|\mathbf{g}\|_1$ だけ減少する。これは Adam の記憶なし（$\beta_1 = \beta_2 = 0$）極限である符号降下法にほかならない（9.5 節）。

9.6.1.2 行列とスペクトルノルム

隠れ重み行列 $\mathbf{W} \in \mathbb{R}^{m \times n}$（写像 $\mathbf{x} \mapsto \mathbf{W}\mathbf{x}$）に対し、摂動 $\Delta\mathbf{W}$ を加えると出力の変化は $\|\Delta\mathbf{W}\mathbf{x}\|_2 \le \|\Delta\mathbf{W}\|_2 \|\mathbf{x}\|_2$ で抑えられる。ここで、$\|\Delta\mathbf{W}\|_2 = \sigma_1(\Delta\mathbf{W})$ は**スペクトルノルム**（作用素ノルム、1.3.5 節）である。勾配行列 $\mathbf{G} \in \mathbb{R}^{m \times n}$ の簡約特異値分解（33.5 節）を次式で表す。

$$\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top = \sum_{i=1}^{r} \sigma_i\, \mathbf{u}_i \mathbf{v}_i^\top,$$

ここで、$\mathbf{U} \in \mathbb{R}^{m \times r}$ と $\mathbf{V} \in \mathbb{R}^{n \times r}$ は正規直交列をもち、$\sigma_1 \ge \cdots \ge \sigma_r > 0$ である。なお、フロベニウスノルム $\|\mathbf{G}\|_F = \sqrt{\sum_i \sigma_i^2}$（平坦化した行列のユークリッドノルム）は、特異値の大きさが同程度であるとき、スペクトルノルム σ_1 を最大で $\sqrt{\min(m,n)}$ 倍上回り得る。

スペクトルノルム制約 $\|\Delta\mathbf{W}\|_2 \le \eta$ のもとで Equation 9.36 を解くと、次の**直交化勾配**が得られる。

$$\Delta\mathbf{W}^\star = \operatorname*{argmin}_{\|\Delta\mathbf{W}\|_2 \le \eta} \langle \mathbf{G}, \Delta\mathbf{W} \rangle = -\eta\, \mathbf{U}\mathbf{V}^\top, \tag{9.37}$$

これは $\mathbf{G}$ のすべての非ゼロ特異値を 1 に設定する操作である。証明は直ちに得られる。任意の行列 $\mathbf{A}$（ただし $\|\mathbf{A}\|_2 \le 1$）に対し、トレースの巡回不変性より次式が成り立つ。

$$\langle \mathbf{G}, \mathbf{A} \rangle = \operatorname{tr}(\mathbf{G}^\top \mathbf{A}) = \operatorname{tr}(\Sigma\, \mathbf{U}^\top \mathbf{A} \mathbf{V}) = \sum_{i=1}^{r} \sigma_i \, (\mathbf{U}^\top \mathbf{A} \mathbf{V})_{ii} \le \sum_{i=1}^{r} \sigma_i,$$

これは、$\|(\mathbf{U}^\top \mathbf{A}\mathbf{V})\|_2 \le \|\mathbf{A}\|_2 \le 1$ から $|(\mathbf{U}^\top \mathbf{A}\mathbf{V})_{ii}| \le 1$ が導かれるためである（演習問題 1）。ここで $\mathbf{A} = -\mathbf{U}\mathbf{V}^\top$ とおけば $\mathbf{U}^\top \mathbf{A}\mathbf{V} = -\mathbf{I}_r$ となり、最小値 $-\eta \sum_i \sigma_i = -\eta \|\mathbf{G}\|_*$（双対ノルムである**核ノルム**、nuclear norm）が達成される。

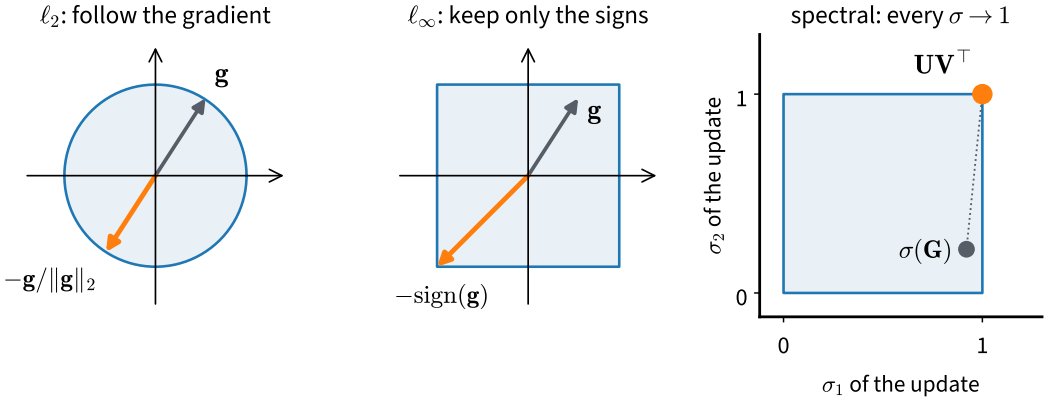


Figure 9.8: 3 つのノルム球のもとでの最急降下方向。ユークリッド ℓ_2 球（勾配方向）、座標ごとの ℓ_∞ ボックス（符号降下法）、およびスペクトルノルム $\|\cdot\|_2$（直交化勾配 $\mathbf{U}\mathbf{V}^\top$ であり、有効なすべての特異値が 1 に等しい）。

Adam は**行列の個々の要素**にわたって更新の大きさを揃える（それでもなおランク 1 の方向へ潰れることがあり、たとえば $\operatorname{sign}\begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ の特異値は 2 と 0 になる）。これに対し、スペクトル最急降下 $-\eta \mathbf{U}\mathbf{V}^\top$ は、等しいステップサイズ η で勾配のすべての有効な特異方向 $(\mathbf{u}_i, \mathbf{v}_i)$ に沿って進む。Figure 9.9 はこれがスペクトルに与える作用を示している。少数の方向に支配された勾配が、すべての有効な方向を等しく重み付けする更新へと変換されるのである。

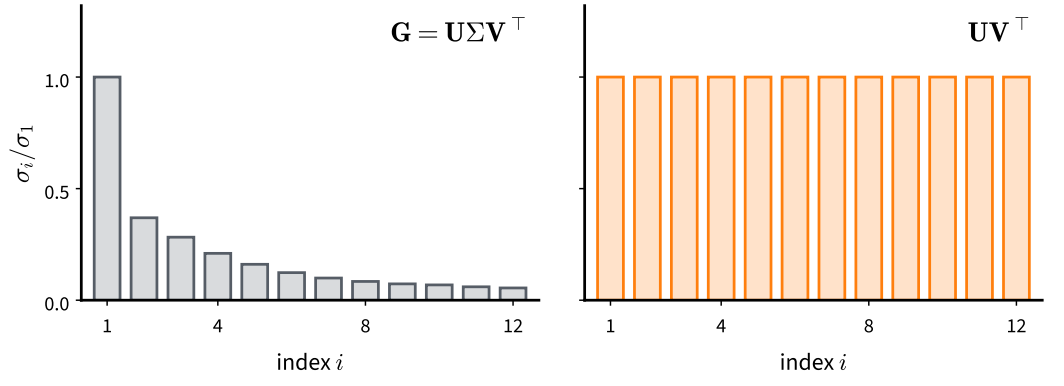


Figure 9.9: 勾配行列 $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top$ の最大特異値に対する相対特異値 σ_i/σ_1（左）と、その直交化 $\mathbf{U}\mathbf{V}^\top$ の特異値（右）。直交化は特異ベクトルを保ったまますべての非ゼロ特異値を 1 にするため、更新は各特異方向に沿って η ずつ進む。

Table 9.3: 3 つのノルム幾何のもとでの最急降下。

更新に対するノルム球	最急降下方向	線形化減少量（双対ノルム）	適したパラメータ群
ユークリッド球 $(\|\mathbf{d}\|_2 \le \eta)$	$-\eta\,\mathbf{g}/\|\mathbf{g}\|_2$	$\eta\,\|\mathbf{g}\|_2$	非構造化ベクトル（SGD）
座標ボックス $(\|\mathbf{d}\|_\infty \le \eta)$	$-\eta\,\mathrm{sign}(\mathbf{g})$	$\eta\,\|\mathbf{g}\|_1$	非一様な座標・埋め込み（AdamW）
スペクトルノルム $(\|\Delta\mathbf{W}\|_2 \le \eta)$	$-\eta\,\mathbf{U}\mathbf{V}^\top$	$\eta\|\mathbf{G}\|_* = \eta\sum_i \sigma_i$	密な 2 次元隠れ重み行列（Muon）

9.6.1.3 行列型の前処理行列（Shampoo）との関連とパラメータの分割

直交化勾配 $\mathbf{U}\mathbf{V}^\top$ は、クロネッカー分解型の行列前処理における記憶なし極限としても現れる。**Shampoo** (Gupta et al. 2018) は左右のグラム行列 $\mathbf{L}_t = \sum_{s\le t}\mathbf{G}_s\mathbf{G}_s^\top$ と $\mathbf{R}_t = \sum_{s\le t}\mathbf{G}_s^\top\mathbf{G}_s$ を蓄積し、$\mathbf{L}_t^{-1/4}\mathbf{G}_t\mathbf{R}_t^{-1/4}$ によって勾配に前処理を施す。これを過去の履歴を蓄積せず単一の勾配 $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top$ に対して評価すると、特異値は恒等的に相殺する。

$$(\mathbf{G}\mathbf{G}^\top)^{-1/4}\,\mathbf{G}\,(\mathbf{G}^\top\mathbf{G})^{-1/4} = \mathbf{U}\Sigma^{-1/2}\mathbf{U}^\top\cdot\mathbf{U}\Sigma\mathbf{V}^\top\cdot\mathbf{V}\Sigma^{-1/2}\mathbf{V}^\top = \mathbf{U}\mathbf{V}^\top. \tag{9.38}$$

$\mathbf{L}_t$ と $\mathbf{R}_t$ を保持して逆行列べき乗を計算する代わりに、標準的な一次モーメントのモメンタムバッファに直交化を適用することで、Muon は行列あたりわずか 1 つのモメンタムバッファで全行列前処理の効果を取り込む（35.5.5 節）。

スペクトルノルムの幾何構造は活性化値空間のあいだの密な線形変換に固有のものであるため、Muon は**隠れ層の 2 次元重み行列にのみ**（アテンションの `qkv` や `proj`、および MLP の重み）適用される。ワンホットな行参照を行う埋め込みテーブル、1 次元のバイアスや LayerNorm のスケール、および最終層の語彙読み出しヘッドは AdamW で最適化される (Jordan et al. 2024)。

9.6.2 ニュートン・シュルツ反復による直交化

訓練中に明示的な SVD によって $\mathbf{U}\mathbf{V}^\top$ を計算するのは時間がかかりすぎて実用的ではない。そこで、任意の**奇の行列多項式**（odd matrix polynomial）が $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ の特異ベクトルを保ったまま各特異値に独立に作用することに着目する。

$$p(\mathbf{X}) = a\mathbf{X} + b\,(\mathbf{X}\mathbf{X}^\top)\mathbf{X} + c\,(\mathbf{X}\mathbf{X}^\top)^2\mathbf{X} = \mathbf{U}\,p(\Sigma)\,\mathbf{V}^\top, \tag{9.39}$$

これは $\mathbf{X}\mathbf{X}^\top = \mathbf{U}\Sigma^2\mathbf{U}^\top$ が成り立つためである。まず $\mathbf{X} \leftarrow \mathbf{X}/\|\mathbf{X}\|_F$ と正規化すれば、すべての特異値が $[0,1]$ に収まる。したがって、奇のスカラー多項式 $p(x)$ として $(0,1]$ を 1 に近づけるものを反復適用すれば（**ニュートン・シュルツ反復**、33.5.4.2 節）、高速な行列積のみを用いて $\mathbf{X}$ を直交化できる。

古典的な 3 次多項式 $p(x) = \frac{3}{2}x - \frac{1}{2}x^3$（原点での傾きが $p'(0) = 1.5$ と小さいため、5 ステップでは小さな特異値の持ち上げが遅すぎる）の代わりに、Jordan et al. (2024) は急な初期の傾き $p'(0) = 3.4445$ をもつ調整済みの 5 次多項式を用いている。

$$p(x) = 3.4445\,x - 4.7750\,x^3 + 2.0315\,x^5. \tag{9.40}$$

Figure 9.10 に示すように、Equation 9.40 を 5 回合成すると、$[0.01, 1]$ 内のすべての特異値が狭い帯域 $[0.68, 1.13]$ に持ち上げられる（さらに 0.0015 より大きいすべての特異値も少なくとも 0.68 に達する）。これは 1 への厳密な漸近収束と引き換えに、小さな特異方向を 5 ステップで急速に増幅させる設計である。

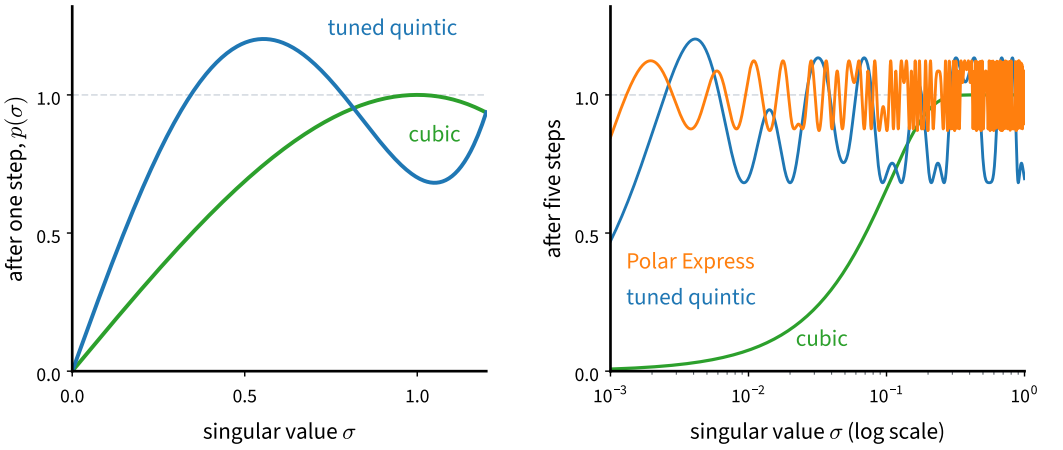


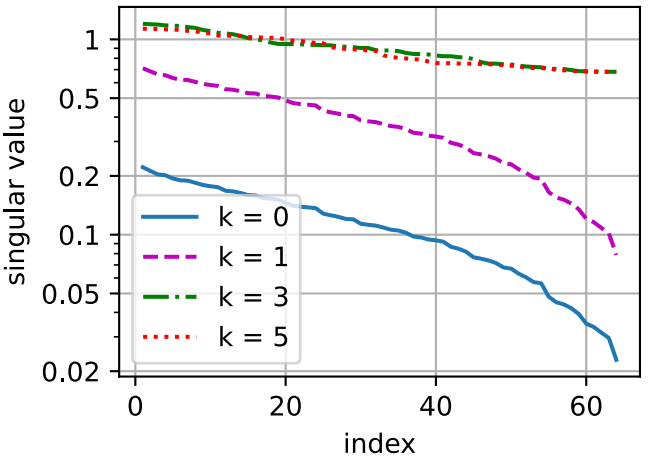
Figure 9.10: 特異値に作用するニュートン・シュルツ多項式。左：古典的な 3 次多項式と調整済み 5 次多項式（Equation 9.40）の 1 回の適用。右：$[10^{-3}, 1]$ 上での 5 回の合成反復（多段階多項式を用いる Polar Express 法 (Amsel et al. 2026) も併記）。

```
def newton_schulz(M, num_iters=5, a=3.4445, b=-4.7750, c=2.0315, eps=1e-7):
    """Approximate the orthogonalization U V^T of M by num_iters steps of
    the odd polynomial a x + b x^3 + c x^5 (Jordan et al.'s tuned quintic)."""
    tall = M.shape[0] > M.shape[1]
    X = M.T if tall else M  # Keep the Gram factor X @ X.T small
    X = X / (jnp.linalg.norm(X) + eps)
    for _ in range(num_iters):
        A = X @ X.T
        X = a * X + (b * A + c * A @ A) @ X
    return X.T if tall else X
```

$m \times n$ 行列（$m \leq n$）とトークンバッチサイズ B に対し、5 回のニュートン・シュルツ反復にかかる計算コストは $5(4m^2n + 2m^3)$ FLOPs であるのに対し、順伝播と逆伝播は $6mnB$ FLOPs である。すなわちオーバーヘッドは $\sim 5m/B$ であり、$< 1\%$（実用規模のバッチサイズ $B \sim 5 \times 10^5$ のとき）、`TinyLM` 上でも $\sim 12\%$（$B = 4{,}096$ のとき）にとどまる。`newton_schulz` を 96×64 のガウス行列に適用すると、反復によって特異値スペクトルが $[0.68, 1.20]$（$k = 3$ の後）、さらに $[0.68, 1.13]$（$k = 5$ の後）へと急速に圧縮される様子が確認できる。

```
G = jax.random.normal(d2l.get_key(), (96, 64))
sigmas = [jnp.linalg.svd(newton_schulz(G, num_iters=k), compute_uv=False)
          for k in (0, 1, 3, 5)]
d2l.plot(jnp.arange(1, 65), sigmas, 'index', 'singular value',
         legend=[f'k = {k}' for k in (0, 1, 3, 5)], yscale='log')
for k, sigma in zip((3, 5), sigmas[2:]):
    print(f'after {k} iterations: [{sigma.min():.2f}, {sigma.max():.2f}]')
```

```
after 3 iterations: [0.68, 1.20]
after 5 iterations: [0.68, 1.13]
```



実際のネットワークの勾配では、2 つの点が Muon に有利に働く。第 1 に、Muon はノイズの多い単一バッチの勾配ではなく、ネステロフのモメンタムバッファ $\mathbf{G}_t + \beta\mathbf{M}_t$（$\sim 1/(1-\beta) = 20$ ステップにわたる平均、ただし $\beta = 0.95$）を直交化する。第 2 に、フロベニウスノルムの 0.15% 未満の特異値（$3.4445^5 \approx 485$ 倍の増幅では 0.68 に届かない領域）は 1 まで押し上げられるのではなく線形に増幅されるにとどまるため、スペクトル末尾の純粋なノイズ方向が自然に減衰される。

9.6.3 ゼロからの Muon の実装

9.6.3.1 更新式と RMS アライメント

$m \times n$ の隠れ重み行列 $\mathbf{W}$ とその勾配 $\mathbf{G}_t$ に対し、Muon はモメンタムバッファ $\mathbf{M}_t$（$\beta = 0.95$）を更新し、5 回のニュートン・シュルツ反復 NS_5 をネステロフの先読み勾配 $\mathbf{G}_t + \beta\mathbf{M}_t$ に適用するとともに、分離型重み減衰（decoupled weight decay）を組み合わせる (Jingyuan Liu et al. 2025)。

$$\begin{aligned} \mathbf{M}_t &= \beta\,\mathbf{M}_{t-1} + \mathbf{G}_t, \\ \mathbf{W}_{t+1} &= (1 - \eta\lambda)\,\mathbf{W}_t - \eta \cdot 0.2\sqrt{\max(m,n)}\;\mathrm{NS}_5(\mathbf{G}_t + \beta\,\mathbf{M}_t). \end{aligned} \tag{9.41}$$

なぜ $0.2\sqrt{\max(m,n)}$ を掛けるのだろうか。$m \times n$ の直交行列 $\mathbf{U}\mathbf{V}^\top$ は $\min(m,n)$ 個の単位特異値をもち、フロベニウスノルムは $\sqrt{\min(m,n)}$、要素ごとの RMS は $\sqrt{\min(m,n)/(mn)} = 1/\sqrt{\max(m,n)}$ となる。したがって $0.2\sqrt{\max(m,n)}$ を掛けることで、行列の次元にかかわらず更新の要素ごとの RMS は $0.2\,\eta$ になる。これは 9.5 節で計測した AdamW の経験的な要素ごとの RMS ステップ幅 $\sim 0.2\,\eta$ と一致するため、両方のオプティマイザで同じ学習率 η と重み減衰 λ を共有できる (Jingyuan Liu et al. 2025)。

```
def scale_by_muon(momentum=0.95):
    """Orthogonalized Nesterov momentum, scaled to AdamW's update RMS."""
    def init(params):
        return jax.tree.map(jnp.zeros_like, params)
```

```
        def update(grads, bufs, params=None):
            bufs = jax.tree.map(lambda b, g: momentum * b + g, bufs, grads)
            def direction(g, b):
                G = g + momentum * b            # Nesterov momentum
                M = G.reshape(-1, G.shape[-1])  # Flattens conv kernels
                O = newton_schulz(M).reshape(G.shape)
                return 0.2 * math.sqrt(max(M.shape)) * O  # AdamW's RMS
            return jax.tree.map(direction, grads, bufs), bufs
        return optax.GradientTransformation(init, update)

    def scratch_muon(learning_rate, momentum=0.95, weight_decay=0.0):
        return optax.chain(scale_by_muon(momentum),
                           optax.add_decayed_weights(weight_decay),  # Decoupled
                           optax.scale_by_learning_rate(learning_rate))
```

各処理を確認するために Muon を自ら実装したが、フレームワークにも標準実装が用意されている。その設定は Equation 9.41 の係数 $0.2\sqrt{\max(m,n)}$ とどのように対応しているのだろうか。これを確かめるため、ゼロ初期化した重みと 96×64 のガウス勾配に対して自前の実装とライブラリの実装でそれぞれ 1 ステップを実行し、2 つの更新を比較してみよう。

```
params = {'w': jnp.zeros((96, 64))}
grads = {'w': jax.random.normal(d2l.get_key(), (96, 64))}
steps = []
for tx in (scratch_muon(0.01),
           optax.contrib.muon(0.01, consistent_rms=0.2)):
    updates, _ = tx.update(grads, tx.init(params), params)
    steps.append(updates['w'])
for name, step in zip(('ours', 'optax.contrib.muon'), steps):
    print(f'{name:18s} update RMS / lr = '
          f'{jnp.sqrt(jnp.mean(step ** 2)) / 0.01:.3f}')
gap = jnp.linalg.norm(steps[0] - steps[1]) / jnp.linalg.norm(steps[0])
print(f'relative difference: {gap:.1e}')
```

```
ours               update RMS / lr = 0.176
optax.contrib.muon update RMS / lr = 0.176
relative difference: 2.5e-03
```

2 つの更新の差は約 1% 以内に収まっており、要素ごとの RMS はいずれも約 $0.18\,\eta$ である。これが公称値の $0.2\,\eta$ をわずかに下回るのは、5 回のニュートン・シュルツ反復後の特異値が 0.68 から 1.13 のあいだに広がっており、厳密な 1 にはならないためである。以下の実験では、本節で作成した自前の実装を引き続き使用する。

`optax.contrib.muon` は係数の定数部分を `consistent_rms` として受け取る。値を 0.2 に指定すると $0.2\sqrt{\max(m,n)}$ となり、指定しない場合は代わりに $\sqrt{\max(1, \text{fan-out}/\text{fan-in})}$ が掛けられる。この実装は 2 次元パラメータに Muon を、それ以外のすべてのパラメータに Adam を適用するため、モデル全体をそのまま渡すことができる。

9.6.3.2 パラメータ群とハイブリッドオプティマイザ

`TinyLM` において、内部ブロックの 8 個の行列（`qkv`、`proj`、`mlp1`、`mlp2` が 2 ブロック分）を Muon に振り分け、残る 22 個のテンソル（埋め込み、LayerNorm ベクトル、バイアス、出力ヘッド）を AdamW に振り分けると、393,216 個のパラメータ（全 470,784 個のうち 84%）が Muon に

割り当てられる。Muon はパラメータあたり AdamW の 2 つのモーメントではなく 1 つのモメンタムバッファしか保持しないため、このハイブリッドオプティマイザはオプティマイザ状態の総メモリ使用量を 42% 削減する。

```
data = d2l.WikiText103(batch_size=64, num_steps=64)

def muon_labels(params):
    """'muon' for the hidden matrices, 'adamw' for every other parameter."""
    def label(path, p):
        name = jax.tree_util.keystr(path)
        is_hidden = p.ndim == 2 and 'emb' not in name and 'head' not in name
        return 'muon' if is_hidden else 'adamw'
    return jax.tree_util.tree_map_with_path(label, params)

model = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(d2l.get_key()))
params = nnx.state(model, nnx.Param)
labels = jax.tree.leaves(muon_labels(params))
for group, name in (('muon', 'Muon'), ('adamw', 'AdamW')):
    sizes = [p.size for p, label in zip(jax.tree.leaves(params), labels)
             if label == group]
    print(f'{name + ":":<6} {len(sizes):>2} tensors, {sum(sizes):>7} '
          'parameters')
```

```
Muon:   8 tensors,  393216 parameters
AdamW: 22 tensors,   77568 parameters
```

そこで、隠れ層の 2 次元行列に分離型重み減衰を適用する Muon と、残りのパラメータに重み減衰を適用しない AdamW を、muon_adamw によって単一のハイブリッドオプティマイザにまとめる。

```
def muon_adamw(lr, weight_decay):
    """Muon with decoupled weight decay on the hidden matrices, AdamW without
    it on the rest."""
    return optax.multi_transform(
        {'muon': scratch_muon(lr, weight_decay=weight_decay),
         'adamw': optax.adamw(lr, weight_decay=0.0)}, muon_labels)
```

9.6.3.3 AdamW との比較

共通の初期重みから出発し、$\eta \in \{10^{-3}, 3\times10^{-3}, 10^{-2}, 3\times10^{-2}, 10^{-1}\}$ の範囲および隠れ行列に対する $\lambda = 0.1$ のもとで 2,000 ステップ訓練を行い、250 ステップごとに検証損失を評価して Muon+AdamW ハイブリッドを標準の AdamW とベンチマーク比較する。まず、ベースラインとなる adamw のスイープを run_lm で実行する。

```
def val_loss(model):
    """Mean cross-entropy per byte over the validation split."""
    return math.log(d2l.lm_perplexity(model, data.val_dataloader()))

def run_lm(make_tx, lrs, seed):
    """d2l.lr_sweep of TinyLM, validated every 250 steps."""
    return d2l.lr_sweep(lambda rngs: d2l.TinyLM(len(data.vocab), rngs=rngs),
                        make_tx, data, lrs, seed, evaluate=val_loss,
```

```
                        metric='validation loss', every=250)

    def hidden_only(params):  # Decay on the hidden matrices only, as for Muon
        return jax.tree.map(lambda label: label == 'muon', muon_labels(params))

    seed = d2l.get_seed()  # One draw, shared by every run below
    lrs, wd = [1e-3, 3e-3, 1e-2, 3e-2, 1e-1], 0.1
    adamw_lm, adamw_val = run_lm(
        lambda lr: optax.adamw(lr, weight_decay=wd, mask=hidden_only), lrs, seed)
```

```
lr 0.001: final loss 1.669, validation loss 1.645
lr 0.003: final loss 1.603, validation loss 1.582
lr 0.01: final loss 1.651, validation loss 1.635
lr 0.03: final loss 2.497, validation loss 2.477
lr 0.1: final loss 3.069, validation loss 3.082
```

次に、muon_adamw ハイブリッドについて、まったく同じ学習率グリッド lrs と初期シード seed でスイープを実行する。

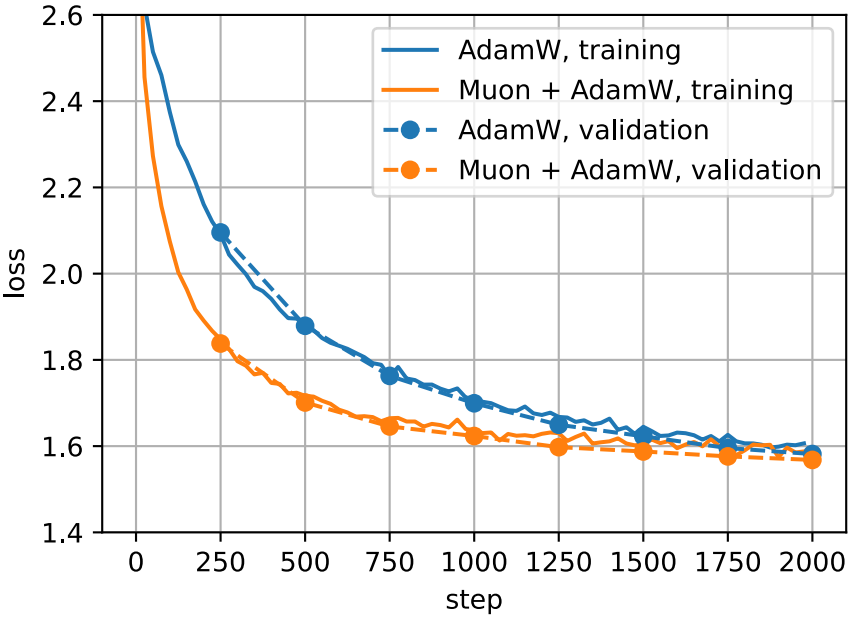
```
    muon_lm, muon_val = run_lm(lambda lr: muon_adamw(lr, wd), lrs, seed)
```

```
lr 0.001: final loss 1.625, validation loss 1.598
lr 0.003: final loss 1.585, validation loss 1.569
lr 0.01: final loss 1.586, validation loss 1.568
lr 0.03: final loss 1.687, validation loss 1.664
lr 0.1: final loss 1.969, validation loss 1.950
```

各オプティマイザを検証損失が最良となった学習率（best_adamw と best_muon）で比較し、平滑化した訓練曲線と検証曲線をプロットするとともに、検証損失が 1.6 nat を下回るステップ数および実用可能な学習率の範囲を表にまとめる。

```
    best_adamw = min(adamw_val, key=lambda lr: adamw_val[lr][-1])
    best_muon = min(muon_val, key=lambda lr: muon_val[lr][-1])
    d2l.plot([list(range(0, 2000, 25))] * 2 + [list(range(250, 2001, 250))] * 2,
             [d2l.smooth(adamw_lm[best_adamw]), d2l.smooth(muon_lm[best_muon]),
              adamw_val[best_adamw], muon_val[best_muon]], 'step', 'loss',
             legend=['AdamW, training', 'Muon + AdamW, training',
                     'AdamW, validation', 'Muon + AdamW, validation'],
             fmts=('C0-', 'C1-', 'C0o--', 'C1o--'), ylim=(1.4, 2.6),
             figsize=(5, 3.5))
    print('validation loss  lr     step 500  step 2000  below 1.6  within 0.15')
    for name, lr, vals in [('AdamW', best_adamw, adamw_val),
                           ('Muon+AdamW', best_muon, muon_val)]:
        below = d2l.steps_to_reach(vals[lr], 1.6, k=1)  # Checkpoint 1, 2, ...
        near = [r for r in lrs if vals[r][-1] < vals[lr][-1] + 0.15]
        print(f'{name:<16} {lr:<6g} {vals[lr][1]:<9.3f} {vals[lr][-1]:<10.3f} '
              f'{250 * below if below else "never":<10} '
              f'{near[0]:g} to {near[-1]:g}')
```

```
validation loss  lr     step 500  step 2000  below 1.6  within 0.15
AdamW            0.003  1.879     1.582      1750       0.001 to 0.01
Muon+AdamW       0.01   1.701     1.568      1250       0.001 to 0.03
```



TinyLM 上では、Muon+AdamW は検証損失で > 0.1 nat 以上のリードをステップ 500 の時点で築き、1.6 nat のしきい値を AdamW より少なくとも 1 チェックポイント（250 ステップ）早く通過する。さらに、実用可能な学習率の上限を $\eta = 3 \times 10^{-2}$ まで広げる（この学習率では AdamW は 2 nat を大きく上回って終了するが、これは要素ごとの正規化のもとで初期のランク 1 的な勾配が、直交化された更新より最大 $\sqrt{\min(m,n)}$ 倍大きいスペクトルノルムをもち得るためである）。

より大規模な領域では、この優位性は小さくなり、検証事例もまだ限られている。完全な学習率減衰を用いた実験において、Wen et al. (2025) は入念に調整した AdamW に対するステップ数の削減率を計測しており、モデルが大きくなるにつれて約 $1.4\times$（パラメータ数 0.1B）から約 $1.1\times$（1.2B）へと縮小することを報告している。一方、Jingyuan Liu et al. (2025) は独自のスケーリング実験において AdamW の約 2 倍の計算効率を報告している。1 兆パラメータの Kimi K2 は Muon とアテンションロジットのクリッピングを用いて訓練されたが (Kimi Team 2025a)、その規模で比較対象となる AdamW の訓練実験は存在しない。ただし、オプティマイザ状態をほぼ半減できる利点は規模によらず成り立つ。

9.6.4 まとめ

最急降下は、更新を抑えるノルム球に対して相対的に定義される。ユークリッド ℓ_2 球からは SGD が、座標ごとの ℓ_∞ ボックスからは符号降下法（平滑化すると Adam）が、そしてスペクトルノルム球 $\|\Delta \mathbf{W}\|_2 \le \eta$ からは行列作用素のすべての有効な特異方向に沿って等しく進む直交化勾配 $-\eta \mathbf{U}\mathbf{V}^\top$ が得られる。**Muon** は、各隠れ 2 次元重み行列のネステロフのモメンタムバッファ上で、調整済み 5 次ニュートン・シュルツ多項式（Equation 9.40）の 5 回の反復によって $\mathbf{U}\mathbf{V}^\top$ を近似し、更新に $0.2\sqrt{\max(m,n)}$ を掛けて AdamW の要素ごとの RMS ステップサイズに合わせるとともに分離型重み減衰を適用し、埋め込み、1 次元ベクトル、読み出しヘッドは AdamW に任せる。TinyLM において、この Muon+AdamW ハイブリッドは初期の収束を加速し、AdamW より $3\times$ 高い学習率でも安定性を保ち、オプティマイザ状態のメモリを 42% 削減する。

9.6.5 演習問題

1. **対角成分の上界とスペクトルステップの一意性**。$\mathbf{U} \in \mathbb{R}^{m\times r}$ と $\mathbf{V} \in \mathbb{R}^{n\times r}$ が正規直交列をもつとする。$\|\mathbf{U}^\top \mathbf{A}\mathbf{V}\|_2 \le \|\mathbf{A}\|_2$ および $|(\mathbf{U}^\top \mathbf{A}\mathbf{V})_{ii}| \le \|\mathbf{A}\|_2$ が成り立つことを示し、さらに Equation 9.37 における最小点 $\Delta\mathbf{W}^\star = -\eta\mathbf{U}\mathbf{V}^\top$ が一意となる条件を求めよ。

2. [code] **経験的な更新 RMS と特異値スペクトル**。Muon に計測処理を追加して TinyLM 上で実行し、$\Delta\mathbf{W}/\eta$ の要素ごとの RMS を 2,000 ステップにわたって追跡するとともに、単一のミニバッチ勾配 $\mathbf{G}_t$ とモメンタムバッファ $\mathbf{M}_t$ の特異値スペクトルをステップ 0 およびステップ 500 で比較せよ。

3. [code] **ニュートン・シュルツの反復回数と多項式の次数**。調整済み 5 次多項式（Equation 9.40）で num_iters $\in \{1, 3, 5, 10\}$ とした場合と、5 回の反復による古典的な 3 次多項式 $p(x) = 1.5x - 0.5x^3$ を TinyLM 上で比較し、検証損失と 1 ステップあたりの実経過時間を報告せよ。

4. [code] **埋め込みを AdamW に残す理由**。

   1. $m \times n$ の勾配行列 $\mathbf{M}$ が $r \le n$ 個の非ゼロかつ線形独立な行のみをもつ（ミニバッチ内の r 個の異なるトークンに対応する）とする。このとき、$\mathbf{U}\mathbf{V}^\top$ のすべての有効な行が単位 ℓ_2 ノルムをもつことが、該当トークンの出現回数が 1 回であるか 500 回であるかによらず成り立つことを示せ。
   2. token_emb を TinyLM の Muon パラメータ群に移し、token_emb を AdamW 群に残した場合と比べて検証パープレキシティが悪化することを実験的に確かめよ。

9.7 スケジュール

これまでの節では、訓練全体を通じて学習率 η を一定に保ち、固定されたステップサイズが最適ではない理由を 2 度見てきた。ミニバッチ SGD（9.2 節）では、一定の学習率は反復点を分散 $O(\eta\sigma^2/\lambda)$ の定常なノイズ球（noise ball）に閉じ込める。η を半分にするとノイズフロアも半分になるが、そこに到達するまでに必要な過渡ステップ数は 2 倍になる。AdamW（9.5 節）や Muon（9.6 節）のような正規化オプティマイザでは、勾配が 0 に近づいても更新の大きさは $O(\eta)$ のままであり、η を小さくするまで反復点は最小点に落ち着くことができない。

現代の訓練レシピでは、**学習率スケジュール**（learning rate schedule）η_t をステップ $t = 0, \dots, T-1$ にわたって変化させることで、このトレードオフを解決する。現在のほぼすべてのスケジュールは、0 からピーク学習率 η_0 まで立ち上げる初期の**ウォームアップ**と、それに続く 0 に向けた**減衰**（decay）という 2 つのフェーズを組み合わせている。本節ではまず、ノイズのある二次関数の谷における学習率減衰を、**累積学習率予算**（cumulative learning-rate budget）$B_T = \sum_{t=0}^{T-1} \eta_t$ と**終端学習率**（terminal learning rate）η_T という 2 つの量の観点から解析する。次に、TinyLM 上で定数、コサイン、およびウォームアップ・安定・減衰（warmup–stable–decay、WSD）の各スケジュールを比較し、高いピーク学習率において線形ウォームアップが初期ステップの不安定性を防ぐ理由を実証する。さらに、一定の学習率を保つプラトーから短い減衰フェーズを分岐させることで、複数の訓練ホライズンにわたって競争力のあるチェックポイントが得られることを示す。

9.7.1 ノイズのある谷における学習率減衰

曲率 $\lambda > 0$ を持つ二次関数の谷の単一座標において、平均 0 の勾配ノイズ ξ_t（分散 σ^2）と時間変化するステップサイズ η_t の下での更新を考えよう。

$$x_{t+1} = (1 - \eta_t \lambda)\, x_t - \eta_t\, \xi_t. \tag{9.42}$$

t ステップにわたって、決定論的な平均は $\prod_{s<t}(1-\eta_s\lambda) \approx e^{-\lambda B_t}$ だけ収縮する。ただし、その指数に現れる量は次式で与えられる。

$$B_t = \sum_{s=0}^{t-1} \eta_s \tag{9.43}$$

これが**累積学習率予算**（ステップ t までの学習率曲線の下側の面積）である。η_t が局所的な緩和窓 $1/(\eta_t\lambda)$ にわたって緩やかに変化する限り、分散は瞬間的なノイズ球 $\eta_t\sigma^2/(2\lambda)$ に追随し、次の平均二乗誤差の分解が得られる。

$$\mathbb{E}\big[x_t^2\big] \approx e^{-2\lambda B_t}\, x_0^2 + \frac{\eta_t\, \sigma^2}{2\lambda}. \tag{9.44}$$

Equation 9.44 から、T ステップにわたる任意の有限ホライズンのスケジュールに対する 2 つの要件が明らかになる。

1. **十分な累積予算 B_T**：総予算 $B_T = \sum_{t=0}^{T-1}\eta_t$ は、最も平坦な方向に沿った初期バイアス $e^{-2\lambda_{\min} B_T} x_0^2$ が消失するほど十分に大きくなければならない。
2. **十分な残余予算を伴う小さな終端学習率 η_T**：終端学習率 η_T は、中間ステップ s で注入されたノイズが残余予算 $R_s = \sum_{u=s}^{T-1}\eta_u \gtrsim 1/\lambda_{\min}$ によって収縮されるよう、十分に緩やかに 0 へと減衰しなければならない。

古典的なパラメトリックなスケジュールの 3 つの族として、区分定数減衰（piecewise-constant decay、ステップ減衰）、指数減衰（exponential decay）、多項式減衰（polynomial decay）がある。

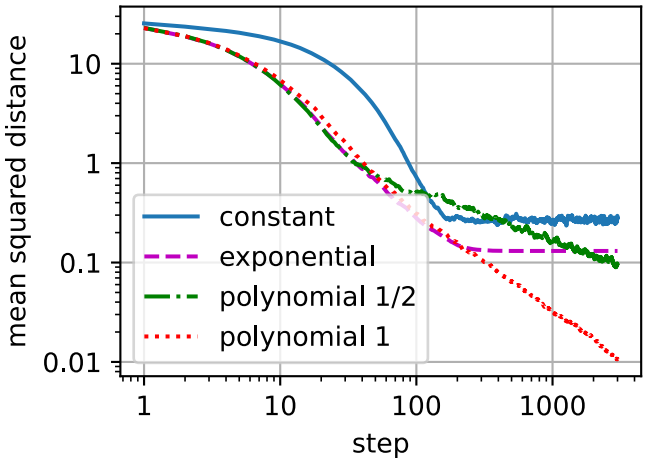
$$\begin{aligned} \eta_t &= \eta_i \quad \text{for } t_i \le t < t_{i+1} && \text{(piecewise constant)},\\ \eta_t &= \eta_0\, \gamma^t && \text{(exponential)},\\ \eta_t &= \eta_0\, (1 + t/\tau)^{-\alpha} && \text{(polynomial)}, \end{aligned} \tag{9.45}$$

ここで、$\gamma \in (0,1)$ は減衰率、$\tau > 0$ は時間スケール、$\alpha > 0$ は指数である。独立した 1,000 回の SGD 実行を谷 $f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$ 上で $T = 3{,}000$ ステップにわたりシミュレーションすると、累積予算 B_T と終端学習率 η_T がいかに収束を支配するか、そして急激すぎる指数減衰がなぜノイズを凍結させるかがわかる。

```
def sgd_valley(schedule, num_steps=3000, num_runs=1000):
    """Mean squared distance to the minimum of SGD runs on the valley."""
    rng = np.random.default_rng()
    h = np.array(d2l.VALLEY_CURVATURES)  # Curvatures along x1 and x2
    x = np.tile([-5.0, -2.0], (num_runs, 1))
    dist = []
    for t in range(num_steps):
        x = x - schedule(t) * (h * x + rng.normal(size=x.shape))
        dist.append((x ** 2).sum(axis=1).mean())
    return dist
```

```
valley_schedules = {
    'constant': lambda t: 0.1,
    'exponential': lambda t: 0.4 * 0.98 ** t,
    'polynomial 1/2': lambda t: 0.4 / (1 + t / 25) ** 0.5,
    'polynomial 1': lambda t: 0.4 / (1 + t / 25)}
dists = {}
for name, schedule in valley_schedules.items():
    dists[name] = sgd_valley(schedule)
    budget = sum(schedule(t) for t in range(3000))
    print(f'{name:15s} budget {budget:6.1f}, final rate {schedule(2999):.2g}, '
          f'mean squared distance {dists[name][-1]:.3f}')
d2l.plot(np.arange(1, 3001), list(dists.values()), 'step',
         'mean squared distance', legend=list(dists), xscale='log',
         yscale='log')
```

```
constant        budget  300.0, final rate 0.1, mean squared distance 0.276
exponential     budget   20.0, final rate 1.9e-27, mean squared distance 0.131
polynomial 1/2  budget  200.2, final rate 0.036, mean squared distance 0.097
polynomial 1    budget   48.2, final rate 0.0033, mean squared distance 0.011
```



一定の学習率（$\eta = 0.1$）は、定常ノイズ球（$0.25 + 0.016 \approx 0.27$）で頭打ち（プラトー）になる。$\alpha = 1/2$（$\eta_T \approx 0.036$）および $\alpha = 1$（$\eta_T \approx 0.0033$）の多項式減衰は Equation 9.44 に追随し、それぞれ 0.09–0.10 および 0.011–0.012 まで低下する。対照的に、指数減衰（$\gamma = 0.98$）は終端学習率が 10^{-27} に達するにもかかわらず、平均二乗誤差 0.12–0.13 で凍結してしまう。ステップ s 以降の残余予算が $R_s \approx \eta_s/(1-\gamma) = 50\eta_s$ しかないため、$\eta_s \lesssim 0.1$（$R_s\lambda_1 \lesssim 1$）となった後に注入されたノイズは決して収縮して消えることがなく、$(1-\gamma)\sigma^2/(4\lambda_1^2) = 0.125$ の凍結分散が残るからである（演習問題 1）。

9.7.2 言語モデルにおける減衰形状

固定されたステップ予算 T のもとでは、現代の深層学習において次の 2 つの減衰プロファイルが主流となっている。

- **コサイン減衰**（cosine decay）(Loshchilov and Hutter 2016) は、ピーク学習率 η_0（$t = 0$）

から終端学習率 η_T（$t = T$）まで半コサイン波をたどる。

$$\eta_t = \eta_T + \frac{\eta_0 - \eta_T}{2}\left(1 + \cos\frac{\pi t}{T}\right). \tag{9.46}$$

$\eta_T = 0$ のとき、コサイン減衰の累積予算は $B_T = \frac{1}{2}\eta_0 T$（一定の学習率 η_0 の予算の半分）となる。

- **安定後減衰（終盤線形減衰）**（stable–then–decay）は、ピーク学習率 η_0 をステップ t_{decay}（ここでは $0.8\,T = 1{,}600$）まで一定に保ち、その後 0 まで訓練の最後の 20% をかけて線形に減衰させる。これにより、一定の学習率の予算の 90%（$B_T = 0.9\,\eta_0 T$）を維持しつつ、$\eta_T = 0$ で訓練を終えることができる。

定数、コサイン、安定後減衰の各スケジュール（$\eta_0 = 3 \times 10^{-3}, T = 2{,}000$）を AdamW（$\lambda = 0$）を用いて `TinyLM` 上で比較するため、まず `WikiText103` を読み込み、モデルのコンストラクタ `tiny_lm(seed)` とスケジュール付きオプティマイザのラッパ関数 `adamw(model, schedule)` を定義する。

```
data = d2l.WikiText103(batch_size=64, num_steps=64)
new_seed = d2l.get_seed

def tiny_lm(seed):
    return d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(seed))

def adamw(model, schedule):
    """AdamW without weight decay at the learning rate schedule(t)."""
    return nnx.Optimizer(model, optax.adamw(schedule, weight_decay=0),
                         wrt=nnx.Param)
```

次に、スケジュール関数 `constant`、`cosine`、`stable_decay`（η_0 を `decay_start=1600` まで保った後、$T = 2{,}000$ で 0 になるよう線形に減衰させる）と、線形ランプのラッパ関数 `warmup` を定義する。

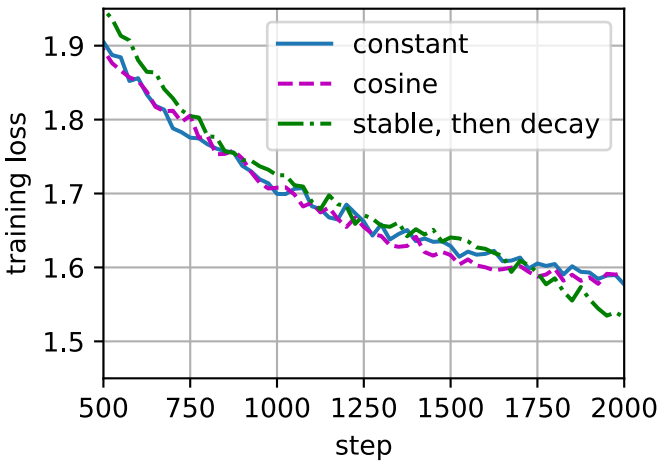
```
def constant(t, eta=3e-3):
    return eta

def cosine(t, eta=3e-3, T=2000):
    return eta * (1 + jnp.cos(jnp.pi * t / T)) / 2

def stable_decay(t, eta=3e-3, decay_start=1600, T=2000):
    """eta until decay_start, then a linear decay to zero at step T."""
    return eta * jnp.minimum(1, (T - t) / (T - decay_start))

def warmup(schedule, num_steps):
    """schedule, ramped up linearly over its first num_steps steps."""
    return lambda t: jnp.minimum(1, (t + 1) / num_steps) * schedule(t)
```

それでは、2 つのシードにわたって 3 つの減衰プロファイルそれぞれのもとで `TinyLM` を訓練し、累積予算 B_T と、ステップ 1,500（終盤減衰が始まる前）およびステップ 2,000 における検証損失を記録しよう。

```
schedules = {'constant': constant, 'cosine': cosine,
             'stable, then decay': stable_decay}
```

```
val_loss = lambda model: math.log(d2l.lm_perplexity(model,
                                                    data.val_dataloader()))
seeds, curves = [new_seed(), new_seed()], {}
print(f'{"schedule":<20}{"budget":>7}  validation loss at step 1500, 2000')
for name, schedule in schedules.items():
    at_1500, at_2000 = [], []
    for seed in seeds:
        model = tiny_lm(seed)
        optimizer = adamw(model, schedule)
        losses = d2l.train_steps(model, data, optimizer, 1500)
        at_1500.append(val_loss(model))
        losses += d2l.train_steps(model, data, optimizer, 500)
        at_2000.append(val_loss(model))
    curves[name] = d2l.smooth(losses)
    budget = sum(float(schedule(t)) for t in range(2000))
    print(f'{name:<20}{budget:7.2f}  ', ' '.join(f'{v:.3f}' for v in at_1500),
          '  ', ' '.join(f'{v:.3f}' for v in at_2000))
d2l.plot(list(range(25, 2001, 25)), list(curves.values()), 'step',
         'training loss', legend=list(curves), xlim=[500, 2000],
         ylim=[1.45, 1.95])
```

```
schedule             budget  validation loss at step 1500, 2000
constant               6.00   1.609 1.606    1.563 1.564
cosine                 3.00   1.576 1.591    1.551 1.565
stable, then decay     5.40   1.610 1.615    1.506 1.514
```



ステップ 1,500 の時点では、コサイン減衰の学習率がすでにピークの 15% まで低下して高周波の振動を抑えているため、コサイン減衰が優位に立つ。しかし、ステップ 1,600 から 2,000 の間に `stable_decay` が $\eta_t \to 0$ へと線形に減衰させると、その損失は ~ 0.10 nat 急落し、定数スケジュールより 0.05–0.06 nat 低く、累積移動予算が半分しかなかったコサイン減衰よりも ~ 0.05 nat 低い損失で終了する（$B_T = 3.0$ 対 5.4）。なお、学習率減衰が完了する前にチェックポイントを比較すると、訓練終了時の真のランキングが逆転し得る (Wen et al. 2025)。

9.7.3 学習率ウォームアップ

減衰が訓練終盤で反復点を落ち着かせるのに対し、**学習率ウォームアップ**（learning-rate warmup）は η_t を 0 から η_0 まで W ステップ（通常は 1%–5% 程度の割合、総ステップ数は T）かけて線形に立ち上げることで、初期ステップを安定化させる。

なぜウォームアップが必要なのだろうか。第 1 に、Adam（9.5 節）のもとでは、ステップ 1 において単一のミニバッチに基づきすべての座標が $\pm\eta_0$ だけ移動してしまうが、この時点ではモメンタム平均 $\mathbf{m}_t$ と二次モーメント平均 $\mathbf{v}_t$ に複数バッチの統計量がまだ蓄積されていない。初期の線形重み行列が $\mathrm{RMS}(\mathbf{W}_0) \approx 0.045$ である `TinyLM` では、$\eta_0 = 0.03$ でのコールドスタートはすべての重みを初期の大きさの $\sim 67\%$ だけステップ 1 で変化させてしまい、初期化を破壊して訓練を 2.8 nat 付近で停滞させる。第 2 に、9.1.4 節で示したように、η_t を徐々に引き上げると鋭さの上限（勾配降下法では $2/\eta_t$、Adam の前処理後の鋭さでは約 $38/\eta_t$）が段階的に下がり、ピーク学習率に耐えられる平坦な領域へとネットワークを誘導する (Goyal et al. 2017; Kalra and Barkeshli 2024)。

20 ステップの線形ランプ（`warmup(peak, 20)`、全体のわずか 1%、総数 $2{,}000$ ステップの実行中）を加えると、初期の相対的な重み変化が約 10% に抑えられて $\eta_0 = 0.03$ での訓練が救われ、最終的な検証損失は約 1 nat 低下する。ステップごとの重み変化を測るため、`linear_weights` で `TinyLM` の埋め込み以外の 2 次元重み行列を抽出する。

```
def linear_weights(model):
    params = nnx.to_flat_state(nnx.state(model, nnx.Param))
    return [p[...] for path, p in params
            if p.ndim == 2 and 'emb' not in str(path)]
```

`relative_change` を用いて $\|\Delta\mathbf{W}\|_{\mathrm{RMS}}/\|\mathbf{W}\|_{\mathrm{RMS}}$ をステップ $1, 5, 20,$ および 100 で計算し、$\eta_0 = 0.03$ でのコールドスタートと 20 ステップの線形ウォームアップを $2{,}000$ ステップにわたって比較する。

```
def relative_change(old, new):
    """RMS change of a list of weight matrices relative to their RMS."""
    return math.sqrt(float(sum(((n - o) ** 2).sum() for o, n in zip(old, new))
                           / sum((o ** 2).sum() for o in old)))

peak = lambda t: constant(t, eta=3e-2)
seed = new_seed()
print('step size / weight size at steps 1, 5, 20, 100; final loss')
for name, schedule in [('cold start', peak),
                       ('20-step ramp', warmup(peak, 20))]:
    model = tiny_lm(seed)
    optimizer = adamw(model, schedule)
    losses, sizes = [], []
    for step in range(1, 101):
        before = linear_weights(model)
        losses += d2l.train_steps(model, data, optimizer, 1)
        if step in (1, 5, 20, 100):
            sizes.append(relative_change(before, linear_weights(model)))
    losses += d2l.train_steps(model, data, optimizer, 1900)
    print(f'{name:<14}', ' '.join(f'{s:.2f}' for s in sizes),
          f'  training {d2l.final_loss(losses):.3f},',
          f'validation {val_loss(model):.3f}')
```

```
step size / weight size at steps 1, 5, 20, 100; final loss
cold start     0.67 0.15 0.02 0.01   training 2.795, validation 2.788
20-step ramp   0.03 0.10 0.05 0.01   training 1.829, validation 1.766
```

9.7.4 ウォームアップ・安定・減衰（WSD）とチェックポイント分岐

コサイン減衰 Equation 9.46 では、総ステップホライズン T をステップ 1 の前に固定する必要がある。途中のチェックポイントは依然として高い学習率にあり、T を超えて訓練を延長するにはスケジュールの再開や継ぎ足しが必要になる。短いウォームアップ、ピーク学習率を一定に保つプラトー、および短い最終減衰を組み合わせることで、**ウォームアップ・安定・減衰（WSD）**スケジュールが得られる (Hu et al. 2024; Hägele et al. 2024)。

幾何学的には、Wen et al. (2024) は WSD の軌跡を**河谷**（river valley）のランドスケープ（Figure 9.11）によってモデル化している。高学習率のプラトーの間、反復点は急峻な横断方向の谷壁の間で振動しつつ、平坦な縦断方向の河床に沿って下流へ急速に前進する。そして短い最終減衰が横断方向の振動（曲率 λ_{steep} が大きいため、わずかな残余予算 $R \gtrsim 1/\lambda_{\text{steep}}$ しか必要としない）を減衰させ、反復点を直接谷底へと落ち着かせる。

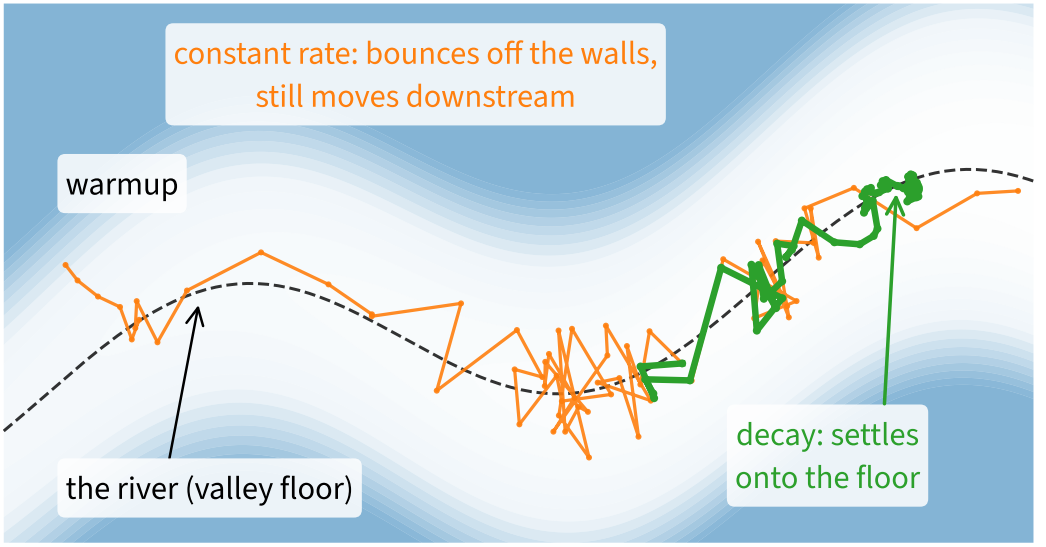


Figure 9.11: ウォームアップ・安定・減衰（WSD）の河谷の幾何学。高い定数プラトーは急峻な横断方向の谷壁の間で振動しながら下流へと急速に前進し、短い減衰ブランチは速やかに谷底へと落ち着く。

9.7.4.1 プラトーからの分岐

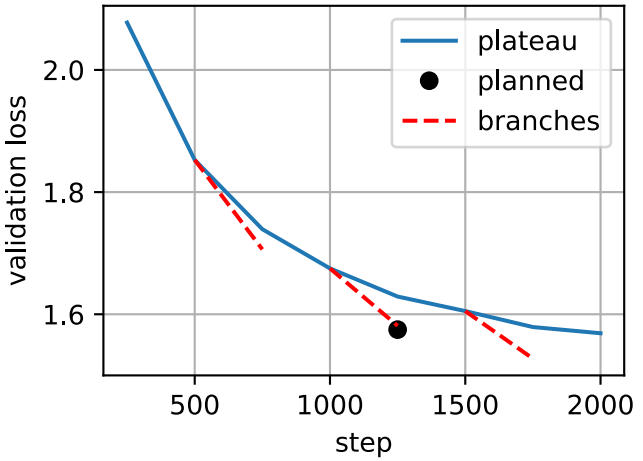
プラトーフェーズは目標ホライズン T に依存しないため、単一の継続的なプラトー実行を維持したまま、任意のチェックポイントから短い線形クールダウン（250 ステップ）を**分岐**（branch）させることができる。まず、`branch(model, optimizer, schedule)` はモデルの重みと AdamW のモーメントバッファ $(\mathbf{m}_t, \mathbf{v}_t)$ の両方を複製し、クールダウン用の `schedule` のもとで訓練を継続できるようにする。

```
def branch(model, optimizer, schedule):
    """Copies of model and optimizer that continue under schedule."""
    model = nnx.clone(model)
    twin = adamw(model, schedule)
    nnx.update(twin, nnx.clone(nnx.state(optimizer)))  # Moments, step count
    return model, twin
```

それでは、`TinyLM` 上で 2,000 ステップの定数学習率プラトーを実行し、250 ステップの線形減衰ブランチをステップ 500, 1,000, および 1,500 で分岐させて、ステップ 1,000 → 1,250 のブランチをゼロから計画された専用の 1,250 ステップ実行と比較しよう。

```
seed = new_seed()  # The planned run below starts from the same weights
model = tiny_lm(seed)
optimizer = adamw(model, constant)
plateau, branched = {}, {}
for step in range(250, 2001, 250):
    d2l.train_steps(model, data, optimizer, 250)
    plateau[step] = val_loss(model)
    if step in (500, 1000, 1500):  # A 250-step decay on a copy
        decay = lambda t, s=step: stable_decay(t, decay_start=s, T=s + 250)
        twin, twin_optimizer = branch(model, optimizer, decay)
        d2l.train_steps(twin, data, twin_optimizer, 250)
        branched[step + 250] = val_loss(twin)
planned = tiny_lm(seed)  # Decays from step 1000 to 1250 by design
d2l.train_steps(planned, data, adamw(planned, lambda t: stable_decay(
    t, decay_start=1000, T=1250)), 1250)
print('step  plateau  branch')
for step, v in branched.items():
    print(f'{step:>4} {plateau[step]:8.3f} {v:7.3f}')
planned_loss = val_loss(planned)
print(f'planned run of 1250 steps: {planned_loss:.3f}')
d2l.plot([list(plateau), [1250]] + [[s - 250, s] for s in branched],
         [list(plateau.values()), [planned_loss]] +
         [[plateau[s - 250], v] for s, v in branched.items()],
         'step', 'validation loss', legend=['plateau', 'planned', 'branches'],
         fmts=('-', 'ko', 'r--', 'r--', 'r--'))
```

```
step  plateau  branch
 750    1.739   1.706
1250    1.629   1.581
1750    1.579   1.527
planned run of 1250 steps: 1.575
```



分岐させたすべてのクールダウンにおいて、検証損失は継続するプラトーよりも急激に低下し（0.05–0.06 nat の低下、ステップ 1,250 および 1,750 時点）、ステップ 1,000 → 1,250 のブランチはゼロから計画された専用の 1,250 ステップ実行に匹敵する。これにより、3 つの訓練ホライズンにわたる完成モデルを、合計 2,250 ステップ（個別実行なら 3,750 ステップ）で得ることができる。

9.7.5 まとめ

学習率スケジュール η_t は、2 つの量によって支配される。1 つは平坦な方向に沿って反復点がどれだけ進むかを決める**累積学習率予算** $B_T = \sum_{t=0}^{T-1} \eta_t$ であり、もう 1 つは最終的なノイズフロアを決定する**終端学習率** η_T である（ただし、以前のノイズを減衰させるために残余予算 $R_s \gtrsim 1/\lambda$ が十分に緩やかに減少する必要がある）。`TinyLM` では、訓練終盤に学習率を 0 まで減衰させることで、定数学習率の AdamW における $O(\eta)$ のステップ揺らぎが除去され、検証損失が 0.05–0.06 nat 低下する。最初の 1%–5% のステップにわたる**線形ウォームアップ**は、$(\mathbf{m}_t, \mathbf{v}_t)$ が安定する前の Adam の過大な初期ジャンプを防ぎ、軌跡を鋭さの上限より下へと導く。最後に、**ウォームアップ・安定・減衰（WSD）**はウォームアップ後に η_0 を一定に保ち、最後の 10%–20% のステップでのみ減衰させることで、移動予算の約 90% を維持しつつ、任意の途中チェックポイントから短いクールダウンブランチを分岐させることを可能にする。

9.7.6 演習問題

1. [code] **指数減衰における凍結ノイズ**。$\eta_s = \eta_0 \gamma^s$ および残余予算 $R_s \approx \eta_s/(1-\gamma)$ のもとで $\sum_s \eta_s^2 \sigma^2 e^{-2\lambda R_s}$ を積分により評価し、曲率 λ に沿った漸近的な凍結分散が $(1-\gamma)\sigma^2/(4\lambda^2)$ となること（ただし $2\lambda\eta_0 \gg 1-\gamma$ とする）を示せ。また、この予測を `sgd_valley` を用いて $\gamma \in \{0.98, 0.99, 0.995\}$ について検証せよ。

2. [code] **区分定数のステップ減衰**。@eq-scheduler-bias-noise を用いて、3 段階の区分定数スケジュール（3,000 ステップ、`sgd_valley` 上）を設計し、その最終的な平均二乗距離を $\alpha = 1$ の多項式減衰（約 0.01）と比較せよ。

3. [code] **同一の累積予算 B_T と終端学習率 η_T を持つスケジュール**。●全ホライズンの線形減衰 $\eta_t = \eta_0(1 - t/T)$ とコサイン減衰 Equation 9.46（$\eta_T = 0$ としたもの。いずれも $B_T = \eta_0 T/2$ かつ $\eta_T = 0$ を満たす）を `sgd_valley`（$\eta_0 = 0.4, T = 3{,}000$）上で比較し、

残余予算 R_t を $t \approx T$ の近傍で比較することで、なぜコサイン減衰の方が小さい終端距離で終了するのかを説明せよ。

4. [code] **ウォームアップ期間と許容可能な最大ピーク学習率。**●TinyLM において線形ウォームアップの長さを $W \in \{2, 5, 10, 20, 50\}$ ステップにわたって変化させ（$\eta_0 = 0.03$）、安定する最小のウォームアップ長を Adam の一次モーメントの窓幅 $1/(1-\beta_1) = 10$ と比較せよ。さらに、20 ステップのウォームアップによって訓練可能な最大ピーク学習率 η_0 がどれだけ大きくなるかを求めよ。

9.8 バッチサイズ

実際の現場では、ミニバッチサイズ b はアクセラレータのメモリを埋めるように選ばれ、そのまま変更されないことが多い。しかし、訓練を複数の GPU にスケールさせたり、訓練の実経過時間と総計算量（FLOPs）をトレードオフしたりする場面では、根本的な問いに直面する。ミニバッチサイズ b を 2 倍にして（ステップあたりの並列計算量を 2 倍にして）訓練したとき、半分のステップ数で目標の損失に到達するのだろうか。

9.2 節で見たように、b 個の例からなるミニバッチを用いると、勾配のサンプリング分散は $\operatorname{tr}(\Sigma)/b$ に減少する。ここから、次の 2 つの異なる領域が導かれる。

1. **小バッチ（ノイズ支配）領域：**サンプリングノイズが真の勾配より大きいとき、b を 2 倍にするとノイズ分散が半分になり、より大きな学習率を使えるようになる。その結果、必要なステップ数 $S(b)$ は半分に減り、処理される総データ例数 $E(b) = b\,S(b)$ は一定に保たれる（**完全な線形スケーリング**）。
2. **大バッチ（曲率支配）領域：**b が十分に大きくなり、ミニバッチ勾配が真のフルバッチ勾配を精度よく近似するようになると、さらに b を増やしても $S(b)$ の減少幅は逓減する一方、データと計算のコスト $E(b)$ は線形に増大する。

McCandlish et al. (2018) は、これら 2 つの領域の遷移が**臨界バッチサイズ**（critical batch size）b_{crit} で生じること、そしてこの値が**勾配ノイズスケール**（gradient-noise scale）$b_{\text{noise}} = \operatorname{tr}(\Sigma)/\|\nabla f\|^2$ によって予測できることを示した。本節では、b_{noise} の 2 バッチ推定量を構築し、ステップ数とデータ量のトレードオフのフロンティア、ならびに SGD（$\eta \propto b$）と Adam（$\eta \propto \sqrt{b}$）における学習率のスケーリング則を導出し、その両方の予測を `TinyLM` と `FashionCNN` で検証する。

9.8.1 勾配ノイズスケール

9.8.1.1 信号、ノイズ、2 バッチ推定量

データセット全体の厳密な勾配を $\nabla f(\mathbf{x})$ とし、ミニバッチ勾配 $\hat{\mathbf{g}}_b$ を独立な b 個の例にわたる平均（1 例あたりの勾配の共分散は Σ）とする。$\mathbb{E}[\hat{\mathbf{g}}_b] = \nabla f$ かつ $\mathbb{E}[\|\hat{\mathbf{g}}_b - \nabla f\|^2] = \operatorname{tr}(\Sigma)/b$ であるため、ミニバッチ勾配の二乗ノルムの期待値は、信号パワー $\|\nabla f\|^2$ とノイズパワー $\operatorname{tr}(\Sigma)/b$ の和に分解される。

$$\mathbb{E}\big[\|\hat{\mathbf{g}}_b\|^2\big] = \|\nabla f\|^2 + \frac{\operatorname{tr}\Sigma}{b}. \tag{9.47}$$

信号パワーとノイズパワーは、次の**単純勾配ノイズスケール**（simple gradient-noise scale）において等しくなる (McCandlish et al. 2018)。

$$b_{\text{noise}} = \frac{\operatorname{tr}\Sigma}{\|\nabla f\|^2}. \tag{9.48}$$

$\operatorname{tr}\Sigma$ と $\|\nabla f\|^2$ はいずれも、データセット全体を走査しなくても推定できる。具体的には、二乗勾配ノルムの平均 $N_b = \mathbb{E}[\|\hat{\mathbf{g}}_b\|^2]$ を 2 つのバッチサイズ $b_\mathrm{s} < b_\mathrm{l}$ で測定し、2 元連立一次方程式 Equation 9.47 を解けばよい。

$$\operatorname{tr}\Sigma \approx \frac{N_{b_\mathrm{s}} - N_{b_\mathrm{l}}}{1/b_\mathrm{s} - 1/b_\mathrm{l}}, \qquad \|\nabla f\|^2 \approx \frac{b_\mathrm{l} N_{b_\mathrm{l}} - b_\mathrm{s} N_{b_\mathrm{s}}}{b_\mathrm{l} - b_\mathrm{s}}.$$

まず WikiText-103 と Fashion-MNIST をアクセラレータに読み込み、ランダムなミニバッチサンプラ `sample_fashion(b)` と `sample_lm(b)` を定義する。

```
data = d2l.WikiText103(batch_size=64, num_steps=64)
fashion = d2l.FashionMNIST(batch_size=256)
X_f = jnp.asarray(fashion.train[0], jnp.float32)[..., None] / 255
y_f = jnp.asarray(fashion.train[1], jnp.int32)
rng = np.random.default_rng(d2l.get_seed())

def sample_fashion(b):  # b random training images
    i = rng.integers(len(y_f), size=b)
    return X_f[i], y_f[i]

def sample_lm(b):  # b windows at random offsets in the training text
    return data.windows(data.train,
                        rng.integers(len(data.train) - data.num_steps, size=b))
```

これらのサンプラを用いて、交差エントロピーの補助関数 `ce_loss`、全パラメータにわたる二乗勾配ノルム `grad_sq_norm`、および 2 バッチ推定量 `noise_scale` を `d2l` に保存する。

```
def ce_loss(model, X, Y):
    """Mean cross-entropy of the logits model(X) for the labels Y."""
    logits = model(X)
    return optax.softmax_cross_entropy_with_integer_labels(
        logits.reshape(-1, logits.shape[-1]), Y.reshape(-1)).mean()

@nnx.jit
def grad_sq_norm(model, X, Y):
    """Squared norm of the gradient of ce_loss over all parameters."""
    grads = nnx.grad(ce_loss)(model, X, Y)
    return sum((g ** 2).sum() for g in jax.tree.leaves(grads))

def noise_scale(model, sample, b_small=16, b_big=2048, m=400):
    """Gradient-noise scale from the mean squared gradient norms of m
    minibatches of size b_small and m // 8 of size b_big from sample(b)."""
    def mean_sq_norm(b, m):  # N_b, averaged over m random minibatches
        return jnp.stack([grad_sq_norm(model, *sample(b))
                          for _ in range(m)]).mean()
    n_small, n_big = mean_sq_norm(b_small, m), mean_sq_norm(b_big, m // 8)
    tr_sigma = (n_small - n_big) / (1 / b_small - 1 / b_big)
    sq_norm = (b_big * n_big - b_small * n_small) / (b_big - b_small)
    return float(tr_sigma / sq_norm)
```

それでは、`noise_scale` を `TinyLM`（$b_\mathrm{s} = 8, b_\mathrm{l} = 256$）に対して、初期化時点と 500 ステップの Adam（$\eta = 3 \times 10^{-3}$）を実行した後の時点で評価してみよう。

```
lm = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(d2l.get_key()))
print(f'noise scale at initialization: '
      f'{noise_scale(lm, sample_lm, b_small=8, b_big=256):.1f}')
optimizer = nnx.Optimizer(lm, optax.adam(0.003), wrt=nnx.Param)
d2l.train_steps(lm, data, optimizer, num_steps=500)
print(f'after 500 steps of Adam: '
      f'{noise_scale(lm, sample_lm, b_small=8, b_big=256):.0f}')
```

```
noise scale at initialization: 2.0
after 500 steps of Adam: 91
```

画像に対する畳み込みネットワークのノイズスケールも同様に増大するのだろうか。デフォルトの $b_{\mathrm{s}} = 16$ 枚および $b_{\mathrm{l}} = 2{,}048$ 枚の画像を用いて、`FashionCNN` でも初期化時点と 200 ステップの Adam（$\eta = 0.004$、後述のスイープで $b = 256$ のときに用いる学習率）を実行した後の時点で同じ測定を繰り返す。

```
cnn = d2l.FashionCNN(rngs=nnx.Rngs(d2l.get_key()))
print(f'noise scale at initialization: {noise_scale(cnn, sample_fashion):.0f}')
optimizer = nnx.Optimizer(cnn, optax.adam(0.004), wrt=nnx.Param)
d2l.train_steps(cnn, fashion, optimizer, num_steps=200)
print(f'after 200 steps of Adam: {noise_scale(cnn, sample_fashion):.0f}')
```

```
noise scale at initialization: 95
after 200 steps of Adam: 76
```

初期化時点の `TinyLM` では、どのウィンドウも大まかなユニグラム分布への同じ調整を要求するため、信号 $\|\nabla f\|^2$ が大きく、$b_{\mathrm{noise}} \approx 2$ 系列にとどまる。頻出バイトの統計量が学習されると（ステップ 500 までには）、$\|\nabla f\|^2$ が低下する一方で系列間の分散 $\operatorname{tr}\Sigma$ は残るため、b_{noise} は 10 倍以上に増大して ~ 50–150 系列となる。

`FashionCNN` はこれとは異なる挙動を示す。10 個のクラスが均衡しているため、未訓練のネットワークにはすべての画像で一致する調整方向が存在せず、ノイズスケールは最初のステップから 100 枚のオーダーになる。200 ステップの間、推定値の変動は実行ごとに増減いずれの方向にも 2 倍未満にとどまり、100 枚のオーダーのままである。

9.8.2 バッチサイズ、ステップ効率、学習率のスケーリング

9.8.2.1 最適ステップサイズと効率フロンティア

L 滑らかな降下上界 $f(\mathbf{x}-\eta\hat{\mathbf{g}}_b)-f(\mathbf{x}) \le -\eta\nabla f^\top\hat{\mathbf{g}}_b+\frac{L}{2}\eta^2\|\hat{\mathbf{g}}_b\|^2$ の期待値をとり、Equation 9.47 を代入すると次式が得られる。

$$\mathbb{E}\big[f(\mathbf{x}-\eta\,\hat{\mathbf{g}}_b)\big]-f(\mathbf{x}) \le -\eta\,\|\nabla f\|^2+\frac{L}{2}\,\eta^2\Big(\|\nabla f\|^2+\frac{\operatorname{tr}\Sigma}{b}\Big). \tag{9.49}$$

右辺を η について最小化すると、最適ステップサイズ $\eta_{\mathrm{opt}}(b)$ と 1 ステップあたりの保証減少量が得られる。

$$\eta_{\text{opt}}(b) = \frac{1}{L} \cdot \frac{1}{1 + b_{\text{noise}}/b},$$
$$f(\mathbf{x}) - \mathbb{E}\big[f(\mathbf{x} - \eta_{\text{opt}}\, \hat{\mathbf{g}}_b)\big] \geq \frac{\|\nabla f\|^2}{2L} \cdot \frac{1}{1 + b_{\text{noise}}/b}. \tag{9.50}$$

（スカラーの滑らかさの上界 $L\mathbf{I}$ を局所ヘッセ行列 $\mathbf{H}$ に置き換えても、曲率重み付きノイズスケール $\operatorname{tr}(\mathbf{H}\Sigma)/(\nabla f^\top \mathbf{H} \nabla f)$ を用いたまったく同じ形の式が得られ (McCandlish et al. 2018)、これに対して Equation 9.48 の b_{noise} はヘッセ行列不要の代理指標として機能する。）

バッチサイズ b における各ステップは、厳密なフルバッチステップに対して $(1 + b_{\text{noise}}/b)^{-1}$ の割合だけ前進するため、フルバッチで $S_{\min}$ ステップを要する目標損失に到達するには、ステップ数 $S(b)$ と総訓練例数 $E(b) = b\,S(b)$ が次式のように必要となる。

$$\frac{S(b)}{S_{\min}} = 1 + \frac{b_{\text{noise}}}{b}, \qquad \frac{E(b)}{E_{\min}} = 1 + \frac{b}{b_{\text{noise}}}, \tag{9.51}$$

ここで、$E_{\min} = b_{\text{noise}} S_{\min}$ は小バッチ極限 $b \ll b_{\text{noise}}$ で達成される最小データ予算である。**臨界バッチサイズ** $b_{\text{crit}} = b_{\text{noise}}$ においては、ステップ数とデータ例数の双方が理論的最小値の 2 倍 $(S = 2S_{\min}, E = 2E_{\min})$ となり、パレートフロンティア $((E/E_{\min} - 1)(S/S_{\min} - 1) = 1)$ の屈曲点（knee）をなす。

9.8.2.2 学習率の線形スケーリングと平方根スケーリング

バッチサイズを b_0 から b へ増やすとき、ノイズ支配領域（$b_0, b \ll b_{\text{noise}}$）では学習率 η をどのようにスケーリングすべきだろうか。

- **SGD とモメンタム（線形スケーリング）：**先の Equation 9.50 より、$b \ll b_{\text{noise}}$ のとき $\eta_{\text{opt}}(b) \approx \frac{b}{L b_{\text{noise}}} \propto b$ となる。言い換えれば、k 回のステップ（サイズ η_0、バッチサイズ b_0）は、サイズ $k\eta_0$、バッチサイズ $k b_0$ による 1 回のステップと一致し、同時に定常ノイズフロア $\propto \eta/b$ も保たれる (Goyal et al. 2017; Smith et al. 2018)。
- **Adam と AdamW（平方根スケーリング）：**ある座標の勾配が平均 $\bar{g}$ と標準偏差 $\sigma/\sqrt{b}$ を持つノイズ支配領域では、Adam は勾配をその RMS スケール $\sqrt{\widehat{v}_t} \approx \sigma/\sqrt{b}$ で正規化するため、1 ステップあたりのドリフト $-\eta(\sqrt{b}/\sigma)\bar{g}$ とステップノイズ $-\eta z$（$z \sim \mathcal{N}(0,1)$）が生じる。分母の $\sigma/\sqrt{b}$ がすでに $\sqrt{b/b_0}$ 倍の因子を与えるため、$k = b/b_0$ 回の小ステップのドリフトと累積分散に 1 回の大きなステップで一致させるには、η を $\sqrt{b/b_0}$ 倍にスケーリングする必要がある (Malladi et al. 2022)。

$$\eta(b) = \frac{b}{b_0}\,\eta_0 \quad \text{(SGD)},$$
$$\eta(b) = \sqrt{\frac{b}{b_0}}\,\eta_0 \quad \text{(Adam / AdamW)}. \tag{9.52}$$

b が b_{crit} を超えると、勾配ノイズはもはや分母で支配的ではなくなり、最適学習率は b とともに増大し続けるのではなく、決定論的な曲率による上限で飽和する (Shallue et al. 2019)。

9.8.3 実験的検証：目標到達ステップ数とスケーリング則

固定したホールドアウトの目標損失に到達するまでのステップ数 $S(b)$ を測定するため、まず `eval_loss` を定義し、検証テンソル上の `ce_loss` を 4,096 例ごとのチャンクに分けて評価できるようにする。

```
eval_chunk = nnx.jit(ce_loss)

def eval_loss(model, X, Y, chunk=4096):
    return float(jnp.stack([eval_chunk(model, X[i:i + chunk], Y[i:i + chunk])
                            for i in range(0, len(Y), chunk)]).mean())
```

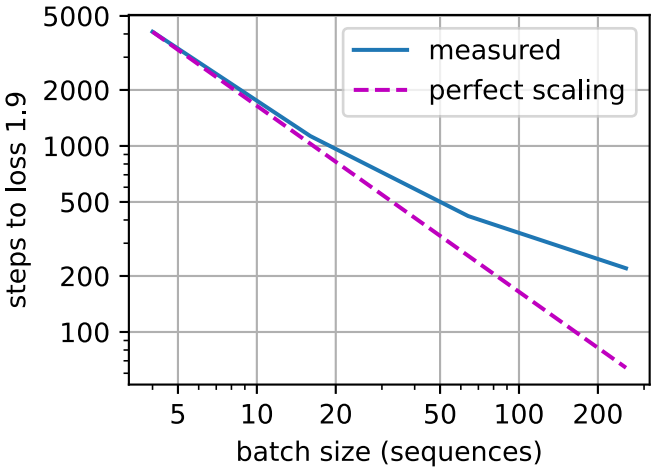
停止ハーネス `train_to_target` は、`eval_loss` を `eval_every` ステップごとに `d2l.train_steps` 内で確認し、`target` に到達したステップ数を返す（`max_steps` を使い切った場合は `inf` を返す）。

```
def train_to_target(model, data, optimizer, X_eval, Y_eval, target,
                    max_steps, eval_every=10):
    """Return the number of steps until the eval loss reaches target."""
    def reached(step):
        return (step % eval_every == 0
                and eval_loss(model, X_eval, Y_eval) <= target)
    steps = len(d2l.train_steps(model, data, optimizer, max_steps,
                                stop=reached))
    return steps if reached(steps) else float('inf')
```

まず共通の初期化から、`train_to_target` を `TinyLM`（目標検証損失 1.9 nats/byte）に対して $b \in \{4, 16, 64, 256\}$ 系列で実行し、Adam の学習率を $\eta = 0.003\sqrt{b/64}$ に従ってスケーリングする。

```
starts = np.arange(0, len(data.val) - data.num_steps, data.num_steps)
X_val, Y_val = data.windows(data.val, starts[::8][:2048])
lm_bs, lm_steps = [4, 16, 64, 256], []
seed = d2l.get_seed()  # One initialization for all batch sizes
for b in lm_bs:
    data.batch_size = b
    model = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(seed))
    optimizer = nnx.Optimizer(model,
                              optax.adam(0.003 * math.sqrt(b / 64)),
                              wrt=nnx.Param)
    lm_steps.append(train_to_target(model, data, optimizer, X_val, Y_val,
                                    target=1.9, max_steps=8000))
    print(f'b={b:4d}  steps={lm_steps[-1]:6.0f}  '
          f'examples={b * lm_steps[-1]:8.0f}')
d2l.plot(lm_bs, [lm_steps, [lm_steps[0] * lm_bs[0] / b for b in lm_bs]],
         'batch size (sequences)', 'steps to loss 1.9', xscale='log',
         yscale='log', legend=['measured', 'perfect scaling'])
```

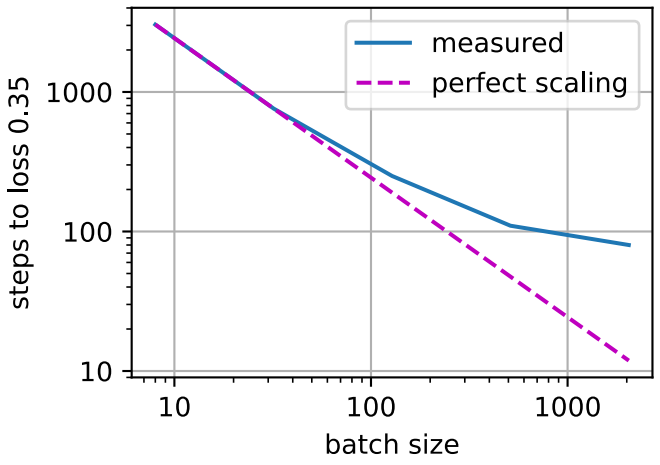
```
b=   4  steps=  4110  examples=   16440
b=  16  steps=  1130  examples=   18080
b=  64  steps=   420  examples=   26880
b= 256  steps=   220  examples=   56320
```



同じ目標到達ステップ数のスイープを FashionCNN（目標損失 0.35）に対しても $b \in \{8, 32, 128, 512, 2048\}$ 枚の画像で繰り返し、学習率を $\eta = 0.004\sqrt{\min(b, 512)/256}$ として $b = 512$ までスケーリングする。

```
cnn_bs, cnn_steps = [8, 32, 128, 512, 2048], []
seed = d2l.get_seed()  # One initialization for all batch sizes
for b in cnn_bs:
    fashion_b = d2l.FashionMNIST(batch_size=b)
    model = d2l.FashionCNN(rngs=nnx.Rngs(seed))
    lr = 0.004 * math.sqrt(min(b, 512) / 256)  # Held fixed above 512
    optimizer = nnx.Optimizer(model, optax.adam(lr), wrt=nnx.Param)
    cnn_steps.append(train_to_target(model, fashion_b, optimizer,
                                     X_f[:8192], y_f[:8192], target=0.35,
                                     max_steps=8000))
    print(f'b={b:4d}  steps={cnn_steps[-1]:6.0f}  '
          f'examples={b * cnn_steps[-1]:8.0f}')
d2l.plot(cnn_bs, [cnn_steps,
                  [cnn_steps[0] * cnn_bs[0] / b for b in cnn_bs]],
         'batch size', 'steps to loss 0.35', xscale='log', yscale='log',
         legend=['measured', 'perfect scaling'])
```

```
b=   8  steps=  3040  examples=   24320
b=  32  steps=   760  examples=   24320
b= 128  steps=   250  examples=   32000
b= 512  steps=   110  examples=   56320
b=2048  steps=    80  examples=  163840
```



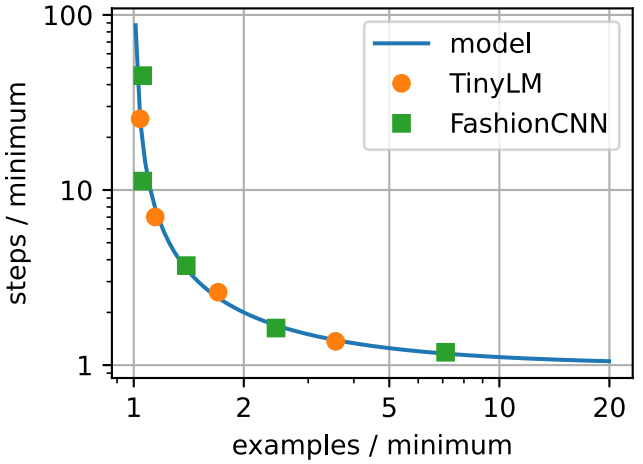
$S(b) = S_{\min}(1 + b_{\text{crit}}/b)$ は $1/b$ に関して線形であるため、`fit_steps` は重み付き最小二乗法によって両モデルの漸近的な最小ステップ数 $S_{\min}$ と実験的な臨界バッチサイズ b_{crit} を抽出する。

```
def fit_steps(bs, steps):
    """Fit S(b) = S_min (1 + b_crit / b) by least squares in 1/b."""
    b, S = np.array(bs, dtype=float), np.array(steps, dtype=float)
    ok = np.isfinite(S)  # Leave out runs that never reached the target
    slope, S_min = np.polyfit(1 / b[ok], S[ok], 1, w=1 / S[ok])
    assert S_min > 0, 'step counts never level off: no floor to fit'
    return S_min, slope / S_min

fits = {'TinyLM': fit_steps(lm_bs, lm_steps),
        'FashionCNN': fit_steps(cnn_bs, cnn_steps)}
for name, (S_min, b_crit) in fits.items():
    print(f'{name:10s}  S_min = {S_min:4.0f} steps  b_crit = {b_crit:4.0f}')
```

```
TinyLM      S_min =  161 steps  b_crit =   98
FashionCNN  S_min =   68 steps  b_crit =  340
```

各実行のデータ例数 $E(b) = b\,S(b)$ を $E_{\min} = b_{\text{crit}} S_{\min}$ で正規化し、ステップ数を $S_{\min}$ で正規化することで、両モデルを普遍的なパレートフロンティア Equation 9.51 と重ねてプロットする。

```
x = np.logspace(0.005, 1.3, 100)  # E / E_min along the model curve
xs, ys = [x], [1 + 1 / (x - 1)]
for (S_min, b_crit), bs, steps in zip(fits.values(), (lm_bs, cnn_bs),
                                      (lm_steps, cnn_steps)):
    xs.append([b * S / (b_crit * S_min) for b, S in zip(bs, steps)])
    ys.append([S / S_min for S in steps])
d2l.plot(xs, ys, 'examples / minimum', 'steps / minimum', xscale='log',
         yscale='log', fmts=('-', 'o', 's'),
         legend=['model', 'TinyLM', 'FashionCNN'])
```



いずれのアーキテクチャも普遍曲線 $(E/E_{\min}-1)(S/S_{\min}-1)=1$ の近くに並ぶ（$E=E_{\min}$ に近い点は曲線の急峻な壁の上に位置するため、測定されたステップ数のわずかな誤差によって上下に大きく動く）。当てはめられた臨界バッチサイズは、TinyLM では $b_{\text{crit}}\sim 100$ 系列、FashionCNN では $b_{\text{crit}}\sim 200$–400 枚である。TinyLM では、この当てはめ結果は先ほど測定した b_{noise} の 2 バッチ推定値（100 系列をやや下回る値）とよく一致する。一方、FashionCNN では約 100 枚という推定値の 2〜4 倍となっている。単純ノイズスケールは曲率を無視するため、b_{crit} の正確な値ではなくオーダー（桁）を与えるのである。

最後に、線形 $(\eta\propto b)$ と平方根 $(\eta\propto\sqrt{b})$ の学習率スケーリング則 Equation 9.52 を、FashionCNN 上で 12,800 枚（1/5 エポック）の固定データ予算のもと直接検証する。補助関数 `final_loss_at` は、`d2l.lr_sweep` を `num_examples // b` ステップだけバッチサイズ `b` で実行し、各実行の最後の 1/4 区間にわたって平均した最終損失を返す。

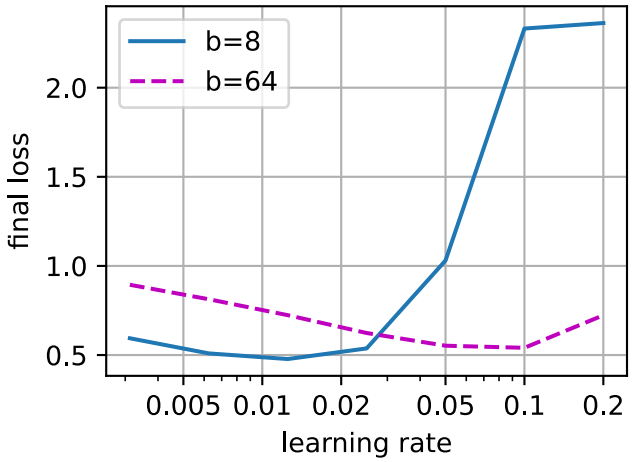
```
def final_loss_at(make_tx, lrs, b, seed, num_examples=12800):
    """Train the CNN once per learning rate; return the final losses."""
    init_key, order_key = jax.random.split(jax.random.key(seed))
    data = d2l.DataModule()  # Every run sees the images in one order
    data.train_dataloader = lambda: d2l.ArrayLoader(
        (X_f, y_f), b, shuffle=True, key=order_key)
    curves, _ = d2l.lr_sweep(d2l.FashionCNN, make_tx, data, lrs, init_key,
                             num_examples // b, verbose=False)
    return [d2l.final_loss(c, k=len(c) // 4) for c in curves.values()]
```

モメンタム付き SGD（$\beta=0.9$）では、バッチサイズを $8\times$（$b=8$ から $b=64$ へ）増やして `sgd_lrs` にわたり探索すると、最適学習率は右へ 4–8× シフトする（$b=64$ では最終損失が $\eta=0.05$ から 0.1 の間でほぼ平坦になる）。

```
sgd_lrs = [0.003125, 0.00625, 0.0125, 0.025, 0.05, 0.1, 0.2]
make_sgd = lambda lr: optax.sgd(lr, momentum=0.9)
seed = d2l.get_seed()
sgd_small = final_loss_at(make_sgd, sgd_lrs, b=8, seed=seed)
sgd_big = final_loss_at(make_sgd, sgd_lrs, b=64, seed=seed)
for b, loss in ((8, sgd_small), (64, sgd_big)):
    print(f'b={b:2d}: best lr {sgd_lrs[np.nanargmin(loss)]:g}')
```

```
d2l.plot(sgd_lrs, [sgd_small, sgd_big], 'learning rate', 'final loss',
         xscale='log', legend=['b=8', 'b=64'])
```

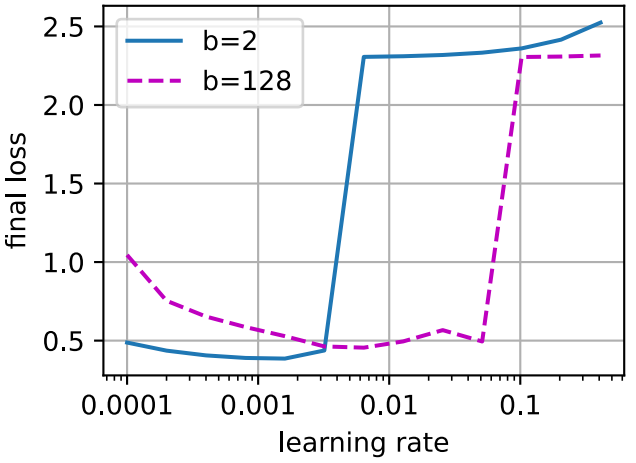
```
b= 8: best lr 0.0125
b=64: best lr 0.1
```



対照的に、Adam のもとでバッチサイズを $64\times$（$b = 2$ から $b = 128$ へ）増やして `adam_lrs` にわたり探索すると、最適学習率のシフトはわずか 4–8×（2 から 3 段階分の $2\times$ グリッドステップ）にとどまり、$\sqrt{64} = 8\times$ と一致して $64\times$ にはならない。

```
adam_lrs = [0.0001 * 2 ** k for k in range(13)]  # 0.0001 to 0.4096
make_adam = optax.adam
seed = d2l.get_seed()
adam_small = final_loss_at(make_adam, adam_lrs, b=2, seed=seed)
adam_big = final_loss_at(make_adam, adam_lrs, b=128, seed=seed)
i, j = int(np.nanargmin(adam_small)), int(np.nanargmin(adam_big))
print(f'best lr: {adam_lrs[i]:g} at b=2; {adam_lrs[j]:g} at b=128, '
      f'final loss {adam_big[j]:.3f}')
for rule, k in (('no scaling', 0), ('square root', 3), ('linear', 6)):
    if i + k < len(adam_lrs):
        print(f'{rule:11s}: lr {adam_lrs[i + k]:g} at b=128, '
              f'final loss {adam_big[i + k]:.3f}')
    else:
        print(f'{rule:11s}: lr {adam_lrs[i] * 2 ** k:g}, beyond the grid')
d2l.plot(adam_lrs, [adam_small, adam_big], 'learning rate', 'final loss',
         xscale='log', legend=['b=2', 'b=128'])
```

```
best lr: 0.0016 at b=2; 0.0064 at b=128, final loss 0.455
no scaling : lr 0.0016 at b=128, final loss 0.528
square root: lr 0.0128 at b=128, final loss 0.495
linear     : lr 0.1024 at b=128, final loss 2.305
```



9.8.3.1 バッチサイズを選択する実践的手順

1. **まずハードウェアの演算器を飽和させる：**1 ステップあたりの実行時間が b に対して線形に増加し始めるまで、少なくとも b を大きくする（9.2 節）。
2. **ウォームアップ後に b_{noise} を推定する：**初期の損失低下の後（たとえば訓練開始から数百ステップ後）に `noise_scale` を測定する。`TinyLM` で見られたように、b_{noise} は初期化時点ではきわめて小さく、訓練中に 1–2 桁増大することがあるためである（これが GPT-3 などの大規模な事前学習でバッチサイズのウォームアップランプが用いられる理由である (Brown et al. 2020)）。
3. **b を $b_{\text{crit}} \approx b_{\text{noise}}$ に照らして設定する：**計算量やデータがボトルネックである場合は $b \lesssim 0.3\, b_{\text{crit}}$ を選ぶ。並列アクセラレータを用いた実経過時間がボトルネックである場合は $b \approx b_{\text{crit}}$（このとき $S = 2S_{\min}$ かつ $E = 2E_{\min}$）を選び、$b \gg b_{\text{crit}}$ は避ける。
4. **η を Equation 9.52 に従ってスケーリングする：**SGD・モメンタムでは $\eta \propto b$、AdamW では $\eta \propto \sqrt{b}$ に従って b_{crit} までスケーリングし、線形ウォームアップと組み合わせる（9.7.3 節）。

9.8.4 まとめ

勾配ノイズスケール $b_{\text{noise}} = \operatorname{tr}(\Sigma)/\|\nabla f\|^2$ は、サンプリングのノイズパワーと信号パワーが等しくなるバッチサイズを表し、データセット全体の走査を行わずに 2 つのバッチサイズ $b_s < b_l$ における二乗勾配ノルムからオンラインで推定できる（Equation 9.47）。適切に調整された学習率のもとでは、目標損失への到達に必要なステップ数 $S(b)$ とデータ例数 $E(b) = b\,S(b)$ は、双曲線型のトレードオフ $S(b)/S_{\min} = 1 + b_{\text{crit}}/b$ および $E(b)/E_{\min} = 1 + b/b_{\text{crit}}$ に従う。この関係は屈曲点 $b_{\text{crit}} \approx b_{\text{noise}}$ を境として、データ効率の高い小バッチ領域（$b \ll b_{\text{crit}}$）とステップ数が飽和する大バッチ領域（$b \gg b_{\text{crit}}$）とを分ける。b_{crit} 未満では、最適学習率は SGD とモメンタムでは線形（$\eta \propto b$）に、Adam と AdamW では平方根（$\eta \propto \sqrt{b}$）に従ってスケーリングし、b が b_{crit} を超えると頭打ちになる。

9.8.5 演習問題

1. **学習率を固定したままのバッチサイズ拡大。**本節の Equation 9.49 と Equation 9.50 を用いて、η を $\eta_0 = \eta_{\text{opt}}(b_0)$ に固定したままバッチサイズを $b > b_0$ に増やすと、1 ステッ

プあたりの保証減少量は $2-(1+b_{\text{noise}}/b)/(1+b_{\text{noise}}/b_0) < 2$ 倍に改善されるものの、$b\to\infty$ としても決して 2 倍を超えないことを示せ。

2. **時間最適バッチサイズと計算量最適バッチサイズ**。バッチサイズ b における 1 ステップの実経過時間が $t(b)=t_0(1+b/b_{\text{sat}})$ であるとき、Equation 9.51 を用いて最適なバッチサイズ $b^\star$（総実経過時間 $T(b)=t(b)S(b)$ を最小化する値）を導出し、そのデータオーバーヘッド $E(b^\star)/E_{\min}$ を求めよ。

3. [code] **訓練に伴う信号とノイズの時間変化**。関数 `noise_scale` を変更して $\operatorname{tr}\Sigma$、$\|\nabla f\|^2$、およびそれらの比 b_{noise} を個別に返すようにし、`TinyLM` 上でステップ $0, 25, 50, 100, 200, 500, 1000, 2000, 3000$ においてこれら 3 つすべてを追跡せよ。そして、$100\times$ におよぶ b_{noise} の増大が、主に信号 $\|\nabla f\|^2$ の低下（$\operatorname{tr}\Sigma$ の増加ではなく）によって生じていることを確認せよ。

4. [code] **臨界バッチサイズを超えたスケーリング**。本節の `TinyLM` に対する目標到達ステップ数のスイープ（9.8.3 節）を $b=1{,}024$ まで拡張し、平方根則 $\eta\propto\sqrt{b}$ に従う場合と η を $b=256$ での値に固定する場合とを比較せよ。さらに、η を b_{crit} を超えてスケーリングし続けると収束が悪化する理由を説明せよ。

9.9 スケールアップ

ニューラルネットワークは小規模な設定で開発され、大規模な設定で運用される。数十億パラメータ規模の事前学習（pretraining）では学習率をグリッド探索（grid search）する余裕がないため、実務では同じアーキテクチャ族に属する小規模な**プロキシモデル**でハイパーパラメータを調整し、それを目的のスケールへと転移する。

深層学習フレームワークでデフォルトとして用いられる**標準パラメータ化**（standard parametrization、SP）では、各層の重みを分散 $1/\text{fan-in}$ で初期化し、すべての層で単一の大域的な学習率 η を共有するが、この設定では最適な学習率が**モデル幅をまたいで転移しない**。層の幅 n が増すにつれて、Adam の 1 ステップは隠れ層の活性化前の値を $O(\eta n)$ 変化させる。なぜなら、n 個の項からなる行列ベクトル積 $\Delta\mathbf{W}\,\mathbf{h}$ において、各項の符号が入力の活性化値 $\mathbf{h}$ と揃うからである。その結果、モデルが広くなるにつれて最適学習率は下方へドリフトする。

最大更新パラメータ化（maximal update parametrization、μP、Yang and Hu (2021); Yang et al. (2022)）は、各層のファンインとファンアウトの増加に応じて学習率と出力乗数をスケーリングすることでこのドリフトを解消し、すべての隠れ層が $\Theta(1)$ の特徴更新を受けることを幅 $n\to\infty$ において保証する。本節では、$64\times$ の範囲にわたる MLP の幅（$n=128$〜$8{,}192$）で SP における学習率のドリフトを測定し、第 1 ステップのスケーリング則を導出する。続いて μP を実装して 1 ステップの**座標チェック**（coordinate check）により検証し (Yang et al. 2022; Dey et al. 2024)、幅をまたいだゼロショット学習率転移を実証したうえで、この規則を Transformer、スペクトルノルムオプティマイザ、および深さへと拡張する。

```
import random
```

9.9.1 標準パラメータ化における学習率と幅

9.9.1.1 幅の系列と学習率のドリフト

Fashion-MNIST 上の隠れ層 3 層からなる ReLU MLP を考え、その隠れ層の幅 n を $\{128, 512, 2048, 8192\}$（パラメータ数は 1.35×10^5〜1.41×10^8 個）に変化させる。各重み行列が n に伴ってどのようにスケーリングするかに注目しよう。

- **入力層（`fc_in`、$784 \to n$）**：ファンインは固定（784）であり、ファンアウトが n に伴って増加する。
- **隠れ層（`fc_h1`、`fc_h2`、$n \to n$）**：ファンインとファンアウトの両方が n に伴って増加する。
- **出力読み出し（`fc_out`、$n \to 10$）**：ファンインが n に伴って増加し、ファンアウトは固定（10）である。

標準パラメータ化（SP）では、すべての重み行列を $\mathcal{N}(0, 1/\text{fan-in})$ から初期化し、バイアスを 0 に初期化して、全パラメータで単一の Adam 学習率 η を共有する。まず 60,000 枚の Fashion-MNIST 訓練画像を平坦化して正規化し、アクセラレータ上に配置する。

```
fashion = d2l.FashionMNIST()
X = jnp.asarray(fashion.train[0], dtype=jnp.float32).reshape(-1, 784) / 255
Y = jnp.asarray(fashion.train[1], dtype=jnp.int32)
```

次に `MLP(width)` を定義する。各層 $784 \to n \to n \to n \to 10$ は標準偏差 $\text{fan-in}^{-1/2}$ で初期化し、全パラメータで単一の Adam 学習率を共有する。

```
class MLP(nnx.Module):
    """A three-hidden-layer ReLU network under standard parametrization."""
    def __init__(self, width, *, rngs):
        init = nnx.initializers.variance_scaling(1.0, 'fan_in', 'normal')
        self.fc_in = nnx.Linear(784, width, kernel_init=init, rngs=rngs)
        self.fc_h1 = nnx.Linear(width, width, kernel_init=init, rngs=rngs)
        self.fc_h2 = nnx.Linear(width, width, kernel_init=init, rngs=rngs)
        self.fc_out = nnx.Linear(width, 10, kernel_init=init, rngs=rngs)

    def features(self, X):
        h = nnx.relu(self.fc_h1(nnx.relu(self.fc_in(X))))
        return nnx.relu(self.fc_h2(h))

    def __call__(self, X):
        return self.fc_out(self.features(X))

    def configure_adam(self, lr):
        return nnx.Optimizer(self, optax.adam(lr), wrt=nnx.Param)
```

最適学習率が幅 n に応じてどのように変化するかを測定するため、`train_mlp` は `arch(width)` を Adam で 400 ステップ（$b = 512$）訓練し、最後の 50 ステップにわたる平均損失を返す（発散した実行については上限を 2.5 とする）。

```
@nnx.jit
def train_step(model, optimizer, Xb, Yb):
    def loss_fn(model):
        return optax.softmax_cross_entropy_with_integer_labels(
            model(Xb), Yb).mean()
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    optimizer.update(model, grads)
    return loss

def train_mlp(arch, width, lr, seed, num_steps=400, batch_size=512):
    model = arch(width, rngs=nnx.Rngs(seed))
```

```
    step = nnx.cached_partial(train_step, model, model.configure_adam(lr))
    idx = jnp.asarray(np.random.default_rng(seed).integers(
        0, X.shape[0], (num_steps, batch_size)))
    losses = jnp.stack([step(X[i], Y[i]) for i in idx])
    v = float(losses[-50:].mean())  # Mean minibatch loss of the last 50 steps
    return min(v, 2.5) if math.isfinite(v) else 2.5
```

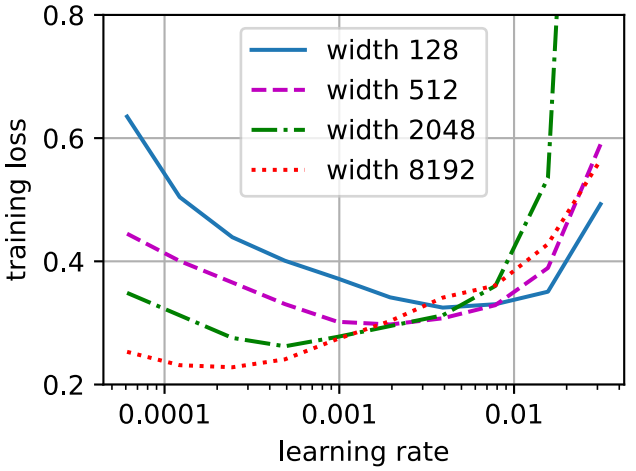
補助関数 sweep、print_optimum、plot_sweep は、幅・学習率・シードのグリッドにわたって train_mlp を評価し、最適学習率 $\eta^\star(n)$ の n に対する両対数傾きを当てはめ、シード平均した損失曲線をプロットする。

```
def sweep(arch, widths, lrs, seeds):
    """Score of every width and learning rate, one column per seed."""
    return {w: np.array([[train_mlp(arch, w, lr, s) for s in seeds]
                         for lr in lrs]) for w in widths}

def print_optimum(loss, lrs):
    """Best rate of the seed-averaged loss per width; slope in log-log."""
    widths = list(loss)
    best = [lrs[loss[w].mean(1).argmin()] for w in widths]
    slope = np.polyfit(np.log(widths), np.log(best), 1)[0]
    print('best lr:', '  '.join(f'{lr:.1e}' for lr in best),
          f'   slope {slope:+.2f}')

def plot_sweep(loss, lrs):
    d2l.plot(lrs, [l.mean(1) for l in loss.values()], 'learning rate',
             'training loss', xscale='log', ylim=[0.2, 0.8],
             legend=[f'width {w}' for w in loss])
```

それでは、sweep を MLP に対して実行し、幅 $n \in \{128, 512, 2048, 8192\}$ および対数等間隔の 10 個の学習率 $\eta \in \{2^{-14}, \ldots, 2^{-5}\}$ を 2 つの共有シードにわたって走査しよう。

```
widths = [128, 512, 2048, 8192]
lrs = [2.0 ** k for k in range(-14, -4)]
seeds = [random.getrandbits(32) for _ in range(2)]  # Shared by every sweep
sp_loss = sweep(MLP, widths, lrs, seeds)
print_optimum(sp_loss, lrs)
plot_sweep(sp_loss, lrs)
```

```
best lr: 3.9e-03  2.0e-03  4.9e-04  2.4e-04    slope -0.70
```



標準パラメータ化では、幅が大きくなるにつれて U 字型の損失曲線が一貫して左へ移動し、最適学習率は 16〜32× 低下する（$\sim 3.9 \times 10^{-3}$（幅 128）から ~ 1〜2×10^{-4}（幅 8,192）への変化であり、両対数傾きは -0.7〜-0.8 となる）。幅 128 での最適学習率を幅 8,192 のモデルにそのまま適用すると、曲線の不安定な右側へと大きく押し上げられ、1,000× 大きいパラメータ数の恩恵が帳消しになってしまう。

9.9.1.2 Adam の第 1 ステップにおける最適学習率のドリフト

なぜ $\eta^\star$ が幅 n に伴って低下しなければならないのかを見るために、正方の隠れ重み行列 $\mathbf{W} \in \mathbb{R}^{n \times n}$ を考え、これが活性化値ベクトル $\mathbf{h} \in \mathbb{R}^n$ を $\mathbf{z} = \mathbf{W}\mathbf{h}$ に写すとする。初期化時には $W_{ij} \sim \mathcal{N}(0, 1/n)$ が $\mathbf{h}$ と独立であるため、各座標 z_i は分散 $\Theta(1)$ をもち、これは n に依存しない。

逆伝播された出力勾配 $\delta = \partial \ell / \partial \mathbf{z}$ をもつ単一の例に対して、重み勾配は外積（outer product）$\mathbf{G} = \delta \mathbf{h}^\top$ となる。ステップ 1 において、Adam のバイアス補正された更新は $\Delta \mathbf{W} = -\eta \ \mathrm{sign}(\mathbf{G}) = -\eta \ \mathrm{sign}(\delta)\, \mathrm{sign}(\mathbf{h})^\top$ である。同じ活性化値ベクトル $\mathbf{h}$ に対して評価すると、活性化前の値 z_i に生じる変化は次のようになる。

$$
\begin{aligned}
(\Delta \mathbf{W}\, \mathbf{h})_i &= -\eta \sum_{j=1}^{n} \mathrm{sign}(\delta_i h_j)\, h_j = -\eta \ \mathrm{sign}(\delta_i) \sum_{j=1}^{n} |h_j| \\
&= -\eta \ \mathrm{sign}(\delta_i)\, n\, \overline{|h|},
\end{aligned}
\tag{9.53}
$$

ここで、$\overline{|h|} = \frac{1}{n} \sum_{j=1}^{n} |h_j| = \Theta(1)$ である。ΔW_{ij} は h_j と相関しているため、和に含まれる n 個の項はすべて同じ符号を共有してコヒーレントに足し合わされ（$O(n)$）、ランダムな符号をもつ初期化時の項のように打ち消し合うことはない（$O(\sqrt{n})$）。特徴更新 $(\Delta \mathbf{W}\, \mathbf{h})_i = \Theta(1)$ を幅 n に対して不変に保つには、ファンイン n をもつ隠れ行列の学習率を $\eta \propto 1/n$ にスケーリングしなければならない（両対数傾きは -1 となる。MLP 全体での実験的な傾きが -0.7〜-0.8 であったのは、$n \times n$ の隠れ行列と、ファンインが固定された入力層やバイアスとの折衷を反映している）。

9.9.2 最大更新パラメータ化（μP）

9.9.2.1 μP のスケーリング則

ファンインのスケーリングが異なる層に単一の大域的学習率を強制するのではなく、**最大更新パラメータ化**（μP、Yang and Hu (2021); Yang et al. (2022)）は参照用の**基準幅**（base width）n_{base} に対する**幅乗数**（width multiplier）$m = n/n_{\text{base}}$ を用いて各層をスケーリングする（Table 9.4）。

1. **入力重みおよび 1D バイアス・ノルム（ファンイン固定）**：初期分散 $1/\text{fan-in}$ と学習率 η をそのまま維持する。
2. **隠れ行列（ファンイン・ファンアウトともに $\propto n$）**：初期分散 $1/\text{fan-in} = 1/n$ を維持し、Adam の学習率を η/m にスケーリングすることで、あらゆる幅において $(\eta/m)\, n\, \overline{|h|} = \eta\, n_{\text{base}}\, \overline{|h|} = \Theta(1)$ となるようにする。
3. **出力読み出し行列（ファンイン $\propto n$、ファンアウト固定）**：$\Delta\mathbf{W}_{\text{out}}\,\mathbf{h}$ と $\mathbf{W}_{\text{out}}\,\Delta\mathbf{h}$ はいずれも相関した n 個の項を出力ロジットへと足し合わせるため、$\mathbf{W}_{\text{out}}$ の各成分の大きさは $O(1/n)$ でなければならない（$O(1/\sqrt{n})$ ではない）。スケーリング前の重みの初期化分散を $1/n_{\text{base}}$ に固定し、順伝播で出力ロジットに $1/m$ を掛け、Adam の学習率を η に保つことで、実効的な出力重みとその更新の双方が $O(1/n)$ でスケーリングされ、あらゆる幅で $\Theta(1)$ のロジット更新が得られる。

Table 9.4: Adam に対する最大更新パラメータ化(μP)の規則(基準幅 n_{base}、幅乗数 $m = n/n_{\text{base}}$)。

パラメータグループ	初期分散	Adam 学習率	順伝播乗数
入力重みと全バイアス	$1/\text{fan-in}$（バイアスは 0）	η	1
隠れ行列（$n \times n$）	$1/\text{fan-in} = 1/n$	η/m	1
出力読み出し（$n \times d_{\text{out}}$）	$1/n_{\text{base}}$	η	$1/m$

```
class MuMLP(MLP):
    """The same network under muP, relative to a width-8,192 base."""
    def __init__(self, width, base_width=8192, *, rngs):
        super().__init__(width, rngs=rngs)
        self.m = width / base_width
        # Rule 3: output init variance 1/base_width at every width
        self.fc_out.kernel[...] *= self.m ** 0.5

    def __call__(self, X):
        # Rule 3: scale the output matrix's logits; the bias is untouched
        return self.fc_out(self.features(X) / self.m)

    def configure_adam(self, lr):
        def label(path, _):
            name = jax.tree_util.keystr(path)
            return 'hidden' if 'fc_h' in name and 'kernel' in name else 'rest'
        tx = optax.multi_transform(
            {'hidden': optax.adam(lr / self.m),  # Rule 2: hidden LR / m
             'rest': optax.adam(lr)},
            lambda params: jax.tree_util.tree_map_with_path(label, params))
        return nnx.Optimizer(self, tx, wrt=nnx.Param)
```

（**base_width=8192** に設定すると、最大の幅 $n = 8{,}192$ において **MuMLP** が **MLP** と一致するため、上記の **SP** のスイープと直接比較できる。実運用では、**base_width** を小規模なプロキシの幅に設定

し、$m > 1$ によって目的のモデルへとスケールアップする。)

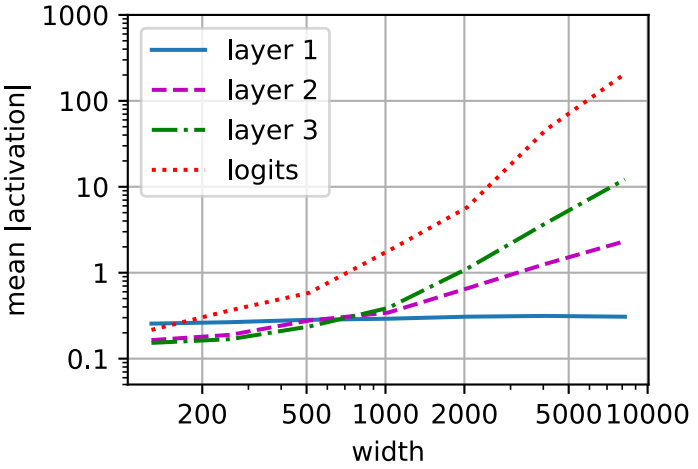
9.9.2.2 座標チェックと学習率転移

スイープを開始する前に μP の実装を検証する手法が**座標チェック**である (Yang et al. 2022; Dey et al. 2024)。ここでは幅 $n \in \{128, 256, \dots, 8192\}$ にわたって Adam を 1 ステップ実行し、各層の活性化値の平均絶対値 $\overline{|h_\ell|}$ を `activations` と `coord_check` で測定する。

```
def activations(model, Xb):
    h1 = nnx.relu(model.fc_in(Xb))
    h2 = nnx.relu(model.fc_h1(h1))
    h3 = nnx.relu(model.fc_h2(h2))
    return [float(jnp.abs(h).mean()) for h in (h1, h2, h3, model(Xb))]

def coord_check(arch, widths, lr, seed, num_steps=1):
    Xb, Yb = X[:256], Y[:256]
    sizes = []
    for width in widths:
        model = arch(width, rngs=nnx.Rngs(seed))
        optimizer = model.configure_adam(lr)
        for _ in range(num_steps):
            train_step(model, optimizer, Xb, Yb)
        sizes.append(activations(model, Xb))
    return list(zip(*sizes))
```

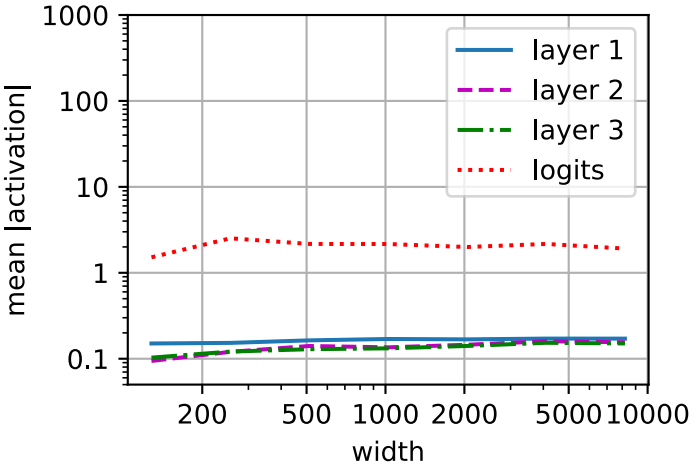
`coord_check` を標準パラメータ化の `MLP` に対して、7 つの幅 $n \in \{128, 256, \dots, 8192\}$ にわたり幅 128 での最適学習率 `lr_128` で実行すると、幅の拡大に伴う深刻な活性化値の発散が明らかになる。

```
check_widths = [128 * 2 ** k for k in range(7)]  # 128 to 8,192
lr_128 = lrs[sp_loss[128].mean(1).argmin()]  # SP's best rate at width 128
sp_acts = coord_check(MLP, check_widths, lr_128, seeds[0])
d2l.plot(check_widths, list(sp_acts), 'width', 'mean |activation|',
         xscale='log', yscale='log', ylim=[0.05, 1000],
         legend=['layer 1', 'layer 2', 'layer 3', 'logits'])
```



まったく同じ 1 ステップの `coord_check` を `MuMLP` に対して基準幅の学習率 `lr_base` で実行すると、μP によって 3 つの隠れ層すべてと出力ロジットが幅に対して不変に保たれることが確認できる。

```
lr_base = lrs[sp_loss[8192].mean(1).argmin()]  # Best at the base width
mup_acts = coord_check(MuMLP, check_widths, lr_base, seeds[0])
d2l.plot(check_widths, list(mup_acts), 'width', 'mean |activation|',
         xscale='log', yscale='log', ylim=[0.05, 1000],
         legend=['layer 1', 'layer 2', 'layer 3', 'logits'])
```

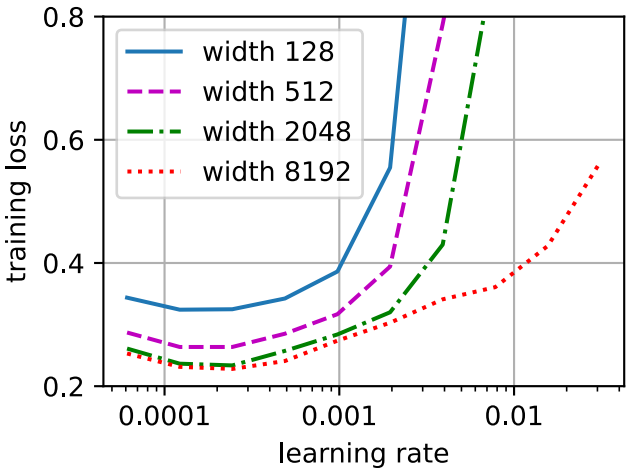


SP（`MLP`）では、最大の幅付近において第 2 層、第 3 層、および出力ロジットがそれぞれ $O(n)$、$O(n^2)$、$O(n^3)$ の増加に近づく（1 ステップ後にはロジットが約 1,000× 上昇しており、これは幅 128 から幅 8,192 への変化に相当する）。一方、μP（`MuMLP`）では、すべての層の活性化値とロジットのスケールが 7 つの幅すべてにわたって平坦なままである。

400 ステップの学習率スイープを μP のもとで繰り返すと、64× の幅の範囲全体にわたって最適学習率が揃うことが確認できる。

```
mup_loss = sweep(MuMLP, widths[:-1], lrs, seeds)
mup_loss[widths[-1]] = sp_loss[widths[-1]]  # At the base width muP is SP
print_optimum(mup_loss, lrs)
plot_sweep(mup_loss, lrs)
```

```
best lr: 1.2e-04  1.2e-04  2.4e-04  2.4e-04    slope +0.20
```



最後に、幅 8,192 で達成される訓練損失について、狭い幅 128 のプロキシで調整した学習率を転移した場合と、幅 8,192 で直接調整した場合とを、SP と μP の双方で比較する。

```
for name, loss in (('SP', sp_loss), ('muP', mup_loss)):
    mean = {w: l.mean(1) for w, l in loss.items()}
    print(f'{name:3s} best loss per width:',
          '  '.join(f'{m.min():.3f}' for m in mean.values()))
    small, wide = widths[0], widths[-1]
    tuned, own = mean[small].argmin(), mean[wide].argmin()
    for label, i in ((f'the width-{small} lr', tuned), ('its own lr', own)):
        runs = '  '.join(f'{v:.3f}' for v in loss[wide][i])
        print(f'    width {wide} at {label:16s} {mean[wide][i]:.3f}'
              f'  (seeds {runs})')
```

```
SP  best loss per width: 0.325  0.297  0.262  0.228
    width 8192 at the width-128 lr 0.341  (seeds 0.332  0.350)
    width 8192 at its own lr       0.228  (seeds 0.227  0.229)
muP best loss per width: 0.324  0.264  0.234  0.228
    width 8192 at the width-128 lr 0.231  (seeds 0.232  0.231)
    width 8192 at its own lr       0.228  (seeds 0.227  0.229)
```

μP では、$\eta^\star(n)$ の両対数傾きは SP における -0.7 以下の急な傾きからほぼ 0 へと縮小する（最適値の移動は $64\times$ の幅の範囲全体で高々 1 グリッドステップに収まる）。幅 128 で調整した学習率を幅 8,192 へそのまま転移した際の準最適性は ≤ 0.003 nat（シードによるノイズを十分に下回る）にとどまり、SP における 0.10–0.15 nat の損失悪化とは対照的である。

9.9.3 Transformer とスペクトルオプティマイザへの拡張

9.9.3.1 Transformer に対する μP の規則

μP を `TinyLM`（9.3.1 節）のようなモデル次元 d、幅乗数 $m = d/d_{\text{base}}$ の Transformer に適用する場合も、ファンイン・ファンアウトの分類はまったく同じであり、アテンションロジットに関する規則が 1 つ加わるだけである（Table 9.5）。訓練中に学習されたクエリとキーのベクトルが整列するため、両者のドット積 $\mathbf{q}^\top \mathbf{k}$ はヘッド次元（head dimension）$d_k \propto d$ にわたって $O(d_k)$ で増大する。したがって、アテンションには $1/d_k$ によるスケーリングが必要となる（追加係数 $1/\sqrt{m}$ を標準的な $1/\sqrt{d_k}$ スケーリングに掛けることに相当する (Yang et al. 2022)）。

Table 9.5: Adam における Transformer の最大更新パラメータ化（μP）の規則（$m = d/d_{\text{base}}$）。

構成要素	スケーリングの理由	μP の規則
埋め込み、1D バイアス、LayerNorm のスケール・シフト	ファンイン固定（参照または要素ごと）	学習率 η
ブロック内部の行列（`qkv`、`proj`、`mlp1`、`mlp2`）	ファンイン・ファンアウトともに $\propto d$	学習率 η/m、初期分散 $1/\text{fan-in}$
出力読み出し `head`	ファンイン $\propto d$、語彙サイズ V は固定	初期分散 $1/d_{\text{base}}$、ロジット $\times 1/m$、学習率 η
アテンションのドット積スコア	整列した d_k 次元ベクトルの内積	スコアを $1/d_k$ でスケーリング（追加係数 $1/\sqrt{m}$）
残差ブランチ（深さ L も増加する場合）	L 個のブランチ出力が残差ストリームに加算	各ブランチに L_{base}/L を乗算（9.9.3.3 節）

9.9.3.2 スペクトルスケーリング条件と Muon

Yang et al. (2023) が示すように、μP のすべての規則は単一の作用素ノルム（operator norm）の原理のもとに統一できる。すなわち、$\mathbf{h} \in \mathbb{R}^{d_{\text{in}}}$（ただし $\|\mathbf{h}\|_2 = \Theta(\sqrt{d_{\text{in}}})$）を $\mathbf{z} \in \mathbb{R}^{d_{\text{out}}}$（ただし $\|\mathbf{z}\|_2 = \Theta(\sqrt{d_{\text{out}}})$）に写す層において、成分ごとの活性化値と更新を $\Theta(1)$ に保つには、重み行列 $\mathbf{W}$ とその更新 $\Delta\mathbf{W}$ の双方が次のスペクトルノルム（spectral norm）をもつ必要がある。

$$\|\Delta\mathbf{W}\|_2 = \Theta\left(\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}}\right). \tag{9.54}$$

正方の $n \times n$ 隠れ行列に対して、Equation 9.54 は $\|\Delta\mathbf{W}\|_2 = \Theta(1)$ を要求する。Adam では、ランク 1 の符号更新が $\|\Delta\mathbf{W}\|_2 = \Theta(\eta n)$ となるため、$\eta \propto 1/n$ が必要になる。Muon（9.6 節）では、直交化された更新 $\text{NS}_5(\cdot)$ は構成上、単位スペクトルノルムをもつ。これを $\sqrt{d_{\text{out}}/d_{\text{in}}}$ でスケーリングすれば (Bernstein 2025)、学習率は $1/m$ の補正なしで幅をまたいでそのまま転移する（一方、AdamW に合わせた係数 $0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$（9.6 節）はスペクトルノルムが $\Theta(\eta\sqrt{n})$ となるため、$\eta \propto 1/\sqrt{n}$ のスケーリングとなる。演習問題 3 を参照）。

9.9.3.3 深さ・重み減衰・有限サイズ

ここまでは幅のみを変化させてきた。しかし実際のスケールアップでは層も増やすことになり、さらに AdamW のもとでは重み減衰（weight decay）も学習率への変更に追随させなければならない。いずれにも固有の規則が必要である。

まず深さを考えよう。L 個の残差ブロック（residual block）からなるネットワークでは、残差ストリーム（residual stream）は L 個のブランチ出力の和となる。各ブランチの寄与が $\Theta(1)$

であれば、その和は L に伴って増大し、すべてのブロックが出力を動かすためオプティマイザの 1 ステップの影響も同様に増大する。対策はロジット乗数と同じ形をとり、各残差ブランチに $(L_{\text{base}}/L)^{\alpha}$ を掛ける。これには 2 つの指数が提案されている。$\alpha = 1/2$ とすると、ブランチ出力は独立なランダム項として足し合わされることになり、これは初期化時の状態に対応する (Bordelon et al. 2024; G. Yang et al. 2024)。一方、$\alpha = 1$ とすると整列した項として足し合わされることになり、これは訓練によって各項が相関した後の状態に対応する。これは読み出し層に $1/n$（$1/\sqrt{n}$ ではなく）を与えたのと同じ議論である。Dey et al. (2025) は、Transformer において $\alpha = 1$ のみが L の増加に伴ってもすべてのブロックに特徴を学習させ続け、その結果、学習率が幅の場合と同様に深さをまたいでも転移することを示している。

次に重み減衰を考える。AdamW では、減衰の時間スケール $\tau = 1/(\eta\lambda)$ が、重みが平衡ノルムへ近づく速さを決める（9.5 節）。ある層の学習率を m で割りながら λ をそのままにする規則では、その層の τ が m 倍に伸びてしまい、広いモデルには調整元の小規模モデルとは異なる正則化がかかることになる。一貫した選択肢の 1 つは τ を一定に保つことであり、具体的には λ に m を掛ける変更を、η を m で割るすべての箇所に適用する (Wang and Aitchison 2025)。

最後に、これらの規則はいずれも無限大サイズの極限でのみ厳密に成り立つ。実用上の幅では最適値は依然としてわずかにドリフトし、本節のスイープでも 1 グリッドステップ分移動した。したがって、転移した学習率はターゲットサイズでのスイープの代わりとするのではなく、狭い範囲のスイープの中心として用いるのが最善である。

9.9.4 まとめ

標準パラメータ化（SP）では、Adam の 1 ステップがファンイン n の隠れ層の出力を $\Theta(\eta n)$ 変化させるため、ネットワークが広くなるにつれて最適学習率 $\eta^{\star}(n)$ は下方へドリフトする。**最大更新パラメータ化（μP）**は、隠れ行列の学習率を $1/m$（ただし $m = n/n_{\text{base}}$）でスケーリングし、出力層の初期化分散を $1/n_{\text{base}}$ に固定しつつロジットを $1/m$ でスケーリングし、さらに Transformer のアテンションスコアを $1/d_k$ でスケーリングすることで、$\Theta(1)$ の特徴更新と幅をまたいだゼロショット学習率転移を実現する。1 ステップの**座標チェック**は、等比数列をなす複数の幅にわたって、オプティマイザの 1 ステップ後も各層の活性化値の大きさが平坦に保たれることを検証する。ハイパーパラメータ転移を本格的なスケールアップへと拡張する際には、さらに深さに応じて残差ブランチを L_{base}/L でスケーリングし、層ごとの学習率の変化に合わせて AdamW の減衰の時間スケール $\tau = 1/(\eta\lambda)$ を一定に保つことで、ターゲットスケールにおける狭いスイープの中心となる漸近的な基準点が得られる。

9.9.5 演習問題

1. [code] **座標チェックによる誤った μP 規則の診断**。`base_width=128` を `MuMLP` に設定し、`coord_check` を `lr_128` で用いて、次の 3 つの意図的なバグを 1 つずつテストせよ。(a) ロジット乗数 $1/m$ の省略、(b) 隠れ層学習率のスケーリング η/m の省略、(c) 隠れ層学習率の η/m^2 によるスケーリング。どの誤りを `coord_check` が即座に検出できるか、また更新を過剰に縮小する誤り（η/m^2）の検出になぜ複数ステップの損失スイープが必要となるのかを明らかにせよ。

2. [code] **参照用の基準幅の影響**。μP のスイープを `base_width=128` として幅 $\{128, 512, 2048, 8192\}$ にわたり再実行し、$\eta^{\star}(n)$ の両対数傾きがほぼ 0 のままであることを確認せよ。また、どの定数比（入力層と隠れ層の学習率比、あるいは初期ロジットスケール）が幅 8,192 における損失の差を生み出しているかを特定せよ。

3. **層およびオプティマイザにわたるスペクトルノルム条件**。●Equation 9.54 を用いて、Table 9.4 の μP 規則が入力層・隠れ層・出力層にスペクトルノルム $\Theta(\sqrt{d_{\text{out}}/d_{\text{in}}})$ の更新

を与えること（ランク 1 の Adam ステップのもとで）を確認し、さらに Muon（9.6 節）において `TinyLM` の 4 つのブロック行列に必要となる η の幅スケーリングを導出せよ。

4. [code] **`TinyLM` における μP と座標チェックの実装**。●Table 9.5 を `d2l.TinyLM` に対して `d_model` $\in \{64, 128, 256, 512, 1024\}$（`num_heads=2`、`base_width=64`）にわたり適用し、$\eta = 3 \times 10^{-3}$ での 1 ステップ座標チェックによって、すべてのサブ層出力およびロジットの大きさが幅に対して平坦に保たれることを確認せよ。

9.10 実践

これまでの小規模な実験では、1 回の実行あたりの訓練時間はわずか数秒であったため、学習率をグリッド状に探索したり、更新が発散するたびに実行をやり直したりできた。しかし、本番の大規模な訓練は事情が異なる。1 回の訓練に数日から数週間を要する場合、大半の設定を事前に固定しなければならず、無防備なまま 1 度の勾配スパイクが生じるだけで何時間もの進捗が損なわれかねない。さらに、最終チェックポイントの品質が最後のミニバッチ抽出の運に左右されてはならない。

極端な大規模環境（たとえば、パラメータ数 540B の PaLM モデルを 6,144 個の TPU チップで訓練した際には、勾配クリッピングを有効にしていたにもかかわらず約 20 回の突発的な損失スパイクが発生した (Chowdhery et al. 2022)）であっても、信頼性の高い訓練は 3 つの運用上の柱に支えられている。

1. **一過性の不安定性に対するガード：**大域ノルムによる**勾配クリッピング**（gradient clipping、8.4.2 節）は、SGD における異常なステップに上限を設けるとともに、AdamW の二次モーメント蓄積変数 $\mathbf{v}_t$ を一過性の勾配スパイクから保護し、チェックポイント保存と巻き戻しによる回復と組み合わせて用いられる。
2. **反復点の平滑化：**モデルの重みの**指数移動平均**（exponential moving average、EMA）を保持することで、学習率をゼロまで完全に減衰させなくとも、ノイズ球内の高周波な変動を除去できる。
3. **短い代理実行による体系的な較正：**デバイスのスループットと b_{noise}（9.8 節）からバッチサイズを選択し、実績のあるオプティマイザのデフォルト設定を採用したうえで、短い実行（ホライズンの 10%）でピーク学習率を較正してから、ウォームアップと減衰の全スケジュール（9.7 節）を実行する。

本節では、これら 3 つの柱をそれぞれ詳しく検討し、未調整のパラメータ数 3.3M、4 ブロックの `TinyLM` に対する完全なエンドツーエンド設定へと組み立てる。

9.10.1 勾配クリッピングとスパイクからの回復

8.4.2 節でリカレントネットワーク向けに導入したように、大域ノルムによる**勾配クリッピング**は全パラメータにわたって連結した勾配を単一のベクトル $\mathbf{g}$ として扱い、そのユークリッドノルム $\|\mathbf{g}\|_2$ がしきい値 θ を超えるたびに再スケーリングする。

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|_2}\right)\mathbf{g}. \tag{9.55}$$

スケーリング係数 $\theta/\|\mathbf{g}\|_2$ は大域的であるため、クリッピングはすべての層にわたる相対的な方向を保ったまま、オプティマイザに入力される勾配のユークリッドノルムを θ 以下に抑える。

```
def record_norms(num_steps):
    """Pass the gradients through unchanged, storing each step's norm."""
    def init(params):
        return dict(norms=jnp.zeros(num_steps), step=jnp.zeros((), jnp.int32))

    def update(grads, state, params=None):
        norm = optax.global_norm(grads)
        norms = state['norms'].at[state['step']].set(norm)
        return grads, dict(norms=norms, step=state['step'] + 1)

    return optax.GradientTransformation(init, update)

def clipped(tx, num_steps, max_norm=1.0):
    """Clip the global gradient norm before an optimizer's update, recording
    the norm before clipping on each of the first num_steps steps."""
    return optax.chain(record_norms(num_steps),
                       optax.clip_by_global_norm(max_norm), tx)
```

9.10.1.1 ガードとしてのクリッピングと常時ブレーキ

9.5 節では、モメンタム 0.9 の SGD を `TinyLM` に適用したところ、`inf` への発散が $\eta = 1.0$ では最初の 20 ステップ以内に生じた。$\eta = 1.0$ は損失ランドスケープ全体に対して大きすぎるのだろうか、それとも初期化直後の重みが大きな勾配を生む冒頭の一過性のステップでのみ大きすぎるのだろうか。まず、`WikiText103` ストリーム（$b = 64$、ウィンドウ長 64）を読み込む。

```
data = d2l.WikiText103(batch_size=64, num_steps=64)
```

次に、`run(seed, lr, max_norm)` は `TinyLM` を 2,000 ステップにわたり、モメンタム 0.9 かつ `max_norm` でクリッピングした SGD で訓練し、各ステップの損失とクリッピング前の勾配ノルムの両方を返す。

```
new_seed = d2l.get_seed

def run(seed, lr, max_norm, num_steps=2000):
    """SGD with momentum 0.9 on TinyLM, clipped at max_norm (math.inf: not
    at all); return the losses and the gradient norms before clipping."""
    model = d2l.TinyLM(len(data.vocab), rngs=nnx.Rngs(seed))
    tx = clipped(optax.sgd(lr, momentum=0.9), num_steps, max_norm)
    optimizer = nnx.Optimizer(model, tx, wrt=nnx.Param)
    losses = d2l.train_steps(model, data, optimizer, num_steps)
    return losses, np.asarray(optimizer.opt_state[0]['norms'][...])
```

ここで、$(\eta, \theta) \in \{(1.0, \infty), (1.0, 1.0), (0.3, \infty), (3.0, 1.0)\}$ を 2 つのシードにわたって検証し、クリッピングが作動したステップ数（$\|\mathbf{g}_t\|_2 > \theta$）、最後にクリッピングが作動したステップ、および勾配ノルムの中央値を記録する。

```
settings = [(1.0, math.inf), (1.0, 1.0), (0.3, math.inf), (3.0, 1.0)]
seeds, runs = [new_seed() for _ in range(2)], {}
print(f'{"lr":>4}{"clip at":>9}{"final loss":>12}{"steps clipped":>15}'
```

```
        f'{"last":>6}{"median norm":>13}{"largest / median":>18}')
  for seed in seeds:
      for lr, max_norm in settings:
          losses, norms = runs[seed, lr, max_norm] = run(seed, lr, max_norm)
          line = f'{lr:4g}{max_norm:9g}{d2l.final_loss(losses):12.3f}'
          if max_norm < math.inf:
              fired = np.nonzero(norms > max_norm)[0]  # Steps clipping changed
              median = np.median(norms)
              line += (f'{len(fired):15d}{fired.max(initial=0):6d}{median:13.2f}'
                       f'{norms[100:].max() / median:18.1f}')  # After step 100
          print(line)
```

lr	clip at	final loss	steps clipped	last	median norm	largest / median
1	inf	inf				
1	1	1.613	14	13	0.26	1.7
0.3	inf	1.669				
3	1	197.921	2000	1999	43.59	1.7
1	inf	inf				
1	1	1.616	15	23	0.26	1.5
0.3	inf	1.679				
3	1	103.020	2000	1999	35.99	13.0

$\eta = 1.0$ において $\theta = 1.0$ でクリッピングすると、最初の数十ステップ（すべてステップ 50 より前）でのみ作動し、残りの 1,950 ステップ（勾配ノルムの中央値は ~ 0.25）では二度と作動せず、`inf` への発散を防いで損失を ~ 1.61–1.64 nat まで引き下げる。ここでクリッピングは**ガードとして機能している**。すなわち、$< 2\%$ のステップにのみ介入して初期の一過性の発散を防いでいる（同時に裾の重い勾配ノイズにも対処する (Zhang, Karimireddy, et al. 2020)）。対照的に、$\eta = 3.0$ では勾配ノルムが有限である限り毎ステップでクリッピングが作動し、それでも訓練は失敗する（損失が 100–200 nat で停滞するか、`inf` にオーバーフローする）。クリッピングが**ブレーキ**として常時作動している場合、単に学習率 η が大きすぎるのである (Godbole et al. 2023)。実務上は、定常状態における勾配ノルムの中央値より θ をおよそ 3–5× 高い値（典型的には $\theta = 1.0$）に設定し、クリッピングの作動率を監視するとよい。

9.10.1.2 AdamW におけるクリッピングと運用上のスパイク回復

AdamW（9.5 節）のもとでは、単一の異常な勾配 $g \gg \sqrt{v_{t-1}}$ が生じても、その直後のステップはそれ自体で高々 $\eta(1-\beta_1)/\sqrt{1-\beta_2}$ 以下に抑えられる（具体的には $\approx 3.2\,\eta$ が $\beta_2 = 0.999$ の値、$0.45\,\eta$ が $\beta_2 = 0.95$ の値である）。直後のステップを抑える代わりに、**AdamW におけるクリッピングは二次モーメント状態 $\mathbf{v}_t$ を保護する**。クリッピングなしの 100× の勾配スパイクは、通常の二乗勾配の $(1-\beta_2) \times 10^4$ 倍を v_t に注入して分母 $\sqrt{\hat{v}_t}$ を肥大化させ、続く $\sim 1/(1-\beta_2)$ ステップ（1,000 ステップが $\beta_2 = 0.999$ の場合、20 ステップが $\beta_2 = 0.95$ の場合）にわたって学習を麻痺させるからである。

勾配クリッピング（および PaLM の z 損失 $10^{-4}\log^2(\sum_j e^{o_j})$ のような補助的なロジット正則化項。演習問題 2 を参照）にもかかわらず突発的な損失スパイクが生じた場合、標準的な運用上の対処法 (Chowdhery et al. 2022; Zhang et al. 2022) は、**スパイクの ~ 100 ステップ前に保存したチェックポイントまで巻き戻し、発生箇所の前後 200–500 個のミニバッチをスキップすること**である。これには、モデルの重み、オプティマイザのモーメント $(\mathbf{m}_t, \mathbf{v}_t)$、スケジューラのステップカウンタ、およびデータストリームのカーソルを一定間隔でチェックポイント保存しておく必要がある。

9.10.2 重み平均化（EMA）

定数または非ゼロの学習率では、確率的な反復点はノイズ球（9.2 節）の内部で最適解の周囲を絶えずゆらぎ続け、曲率 λ_{curv} のモードは $\sim 1/(\eta\lambda_{\text{curv}})$ ステップの自己相関時間を持つ。そこで、次のように**指数移動平均（EMA）** $\bar{\mathbf{x}}_t$ をライブパラメータ $\mathbf{x}_t$ に対して減衰係数 $\alpha \in [0.99, 0.9999]$ で保持する (Polyak and Juditsky 1992)。

$$\bar{\mathbf{x}}_t = \alpha\,\bar{\mathbf{x}}_{t-1} + (1-\alpha)\,\mathbf{x}_t, \tag{9.56}$$

この移動平均は直近 $1/(1-\alpha)$ ステップのウィンドウにわたって平均をとることで、ライブの訓練軌跡を変えることなく評価時にノイズ球の変動を打ち消す。

定数学習率の SGD（$\eta = 0.3$）における重み EMA あり・なし（$\alpha = 0.999$、ウィンドウ幅 $1{,}000$ ステップ ≈ 4.2 エポック、初期の遅れを避けるためエポック 5 の終了後に開始）と、15 エポック（$3{,}525$ ステップ）にわたるコサイン学習率減衰とを `FashionCNN` 上で比較する。まず `FashionMNIST`（$b = 256$）を読み込み、検証正解率の補助関数 `test_acc` を定義する。

```
fashion = d2l.FashionMNIST(batch_size=256)
num_epochs, num_batches = 15, len(fashion.train_dataloader())
test_acc = lambda model: d2l.evaluate_accuracy(model, fashion.val_dataloader())
```

次に、スケジュールの補助関数 `warmup_cosine` と重み EMA を定義する。

`ema_of_weights` はオプティマイザ状態に EMA を保持する optax の変換であり、`averaged` はその重みを持つモデルを複製する。

```
warmup_cosine = lambda peak, warmup, num_steps: (
    optax.warmup_cosine_decay_schedule(0.0, peak, warmup, num_steps))

def ema_of_weights(alpha=0.999, start=0):
    """Pass the updates through unchanged, keeping an EMA of the updated
    weights that averages from step start on (copies them before)."""
    def init(params):
        return dict(ema=params, step=jnp.zeros((), jnp.int32))

    def update(updates, state, params):
        new = optax.apply_updates(params, updates)
        rate = jnp.where(state['step'] < start, 1.0, 1 - alpha)
        ema = optax.incremental_update(new, state['ema'], rate)
        return updates, dict(ema=ema, step=state['step'] + 1)

    return optax.GradientTransformation(init, update)

def averaged(model, optimizer):
    """A copy of model with the EMA kept by the last transformation."""
    ema = nnx.clone(model)
    nnx.update(ema, optimizer.opt_state[-1]['ema'])
    return ema
```

訓練ループ `train_fashion` は、`schedule(t)` のもとで通常の SGD を `num_epochs=15` エポック実行し、各エポック終了時にライブ重みとその EMA（エポック `start=5` から開始）の両方についてホールドアウトのテスト正解率を記録する。

```
def train_fashion(seed, schedule, alpha=0.999, start=5):
    """Plain SGD on FashionCNN at the rate schedule(t); the test accuracy
    after every epoch of the weights and of their EMA from epoch start on."""
    model = d2l.FashionCNN(rngs=nnx.Rngs(seed))
    tx = optax.chain(optax.sgd(schedule),
                     ema_of_weights(alpha, start * num_batches))
    optimizer = nnx.Optimizer(model, tx, wrt=nnx.Param)
    live, avg = [], []
    for epoch in range(num_epochs):
        d2l.train_steps(model, fashion, optimizer, num_batches)
        live.append(test_acc(model))
        avg.append(test_acc(averaged(model, optimizer)) if epoch >= start
                   else math.nan)
    return live, avg
```

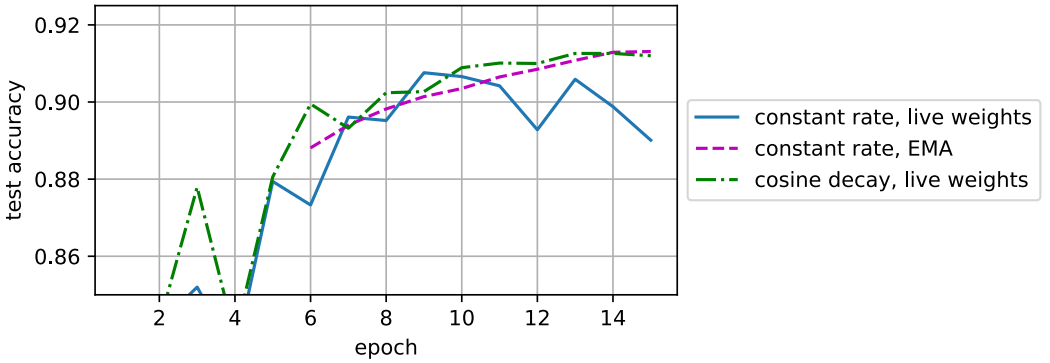
2 つのシードにわたり train_fashion を定数学習率（$\eta = 0.3$）とコサイン減衰（$\eta : 0.3 \to 0$）の両方で実行し、最後の 5 エポックにおけるライブ重みの正解率の範囲を最終的な EMA の正解率と比較する。

```
schedules = {'constant rate': lambda t: 0.3,
             'cosine decay': warmup_cosine(0.3, 0, num_epochs * num_batches)}
cnn_seeds, curves = [new_seed() for _ in range(2)], {}
print(f'{"test accuracy":<15}{"live, last five epochs":>24}{"live, final":>13}'
      f'{"EMA, final":>12}')
for seed in cnn_seeds:
    for name, schedule in schedules.items():
        live, avg = curves[seed, name] = train_fashion(seed, schedule)
        print(f'{name:<15}{min(live[-5:]):15.3f} to {max(live[-5:]):.3f}'
              f'{live[-1]:13.3f}{avg[-1]:12.3f}')
```

```
test accuracy    live, last five epochs  live, final  EMA, final
constant rate            0.890 to 0.906        0.890       0.913
cosine decay             0.910 to 0.913        0.912       0.911
constant rate            0.863 to 0.904        0.904       0.911
cosine decay             0.903 to 0.910        0.910       0.906
```

1 つ目のシードについてエポックごとのテスト正解率の軌跡をプロットすると、定数学習率の EMA がライブ重みのノイズ球によるジッタをいかに除去するかが視覚的にわかる。

```
(live, avg), (decayed, _) = (curves[cnn_seeds[0], name] for name in schedules)
d2l.plot(list(range(1, num_epochs + 1)), [live, avg, decayed], 'epoch',
         'test accuracy', ylim=[0.85, 0.925], figsize=(5, 2.5),
         legend=['constant rate, live weights', 'constant rate, EMA',
                 'cosine decay, live weights'], legend_loc='outside')
```



定数学習率では、最後の 5 エポックにわたってライブ重みのテスト正解率が数パーセントポイントの幅で小刻みに揺れ動き、1 エポックの調子が悪いときにはさらに大きく落ち込むこともある。これに対し、その EMA は帯の上端に沿って滑らかに上昇して約 91% に達し、学習率を一度も下げることなくコサイン減衰させた実行の最終正解率に匹敵する。したがって、WSD スケジュールの定数プラトー期間中において、重み EMA はクールダウン分岐（9.7.4.1 節）の低コストな近似となる。追加のステップを一切必要としないからである。ただし、完全な代替にはならない。言語モデルでは、プラトーに沿った平均化によってクールダウンによる改善の一部は回収できるものの、すべては回収できない (Hägele et al. 2024)。

9.10.3 エンドツーエンド設定の実例

ここで本章全体を総合し、3,000 ステップの訓練を、未調整の 4 ブロック、幅 256、4 ヘッドの `TinyLM`（パラメータ数 3.3M）に対して構成する。Table 9.6 に 4 つの最先端の事前学習で公開されているオプティマイザ設定をまとめ、Table 9.7 に訓練ログの症状と対処法を対応付けた診断早見表を示す。

Table 9.6: 4 つの大規模言語モデルの事前学習における公開オプティマイザ設定。

モデル	オプティマイザ	β_1, β_2	ピーク学習率・スケジュール	ウォームアップ	クリップ θ	重み減衰 λ	バッチサイズ
Llama 3 405B (Grattafiori et al. 2024)	AdamW	—	8×10^{-5}、8×10^{-7} へコサイン減衰	8k ステップ	—	—	4M → 16M トークンへ段階的増加
DeepSeek-V3 (DeepSeek-AI, Liu, et al. 2024)	AdamW	0.9, 0.95	2.2×10^{-4}、2.2×10^{-5} へ WSD コサイン減衰	2k ステップ	1.0	0.1	3,072 → 15,360 系列へ段階的増加
OLMo 2 7B (Team OLMo et al. 2025)	AdamW	0.9, 0.95	3×10^{-4}、3×10^{-5} へコサイン減衰	2k ステップ	1.0	0.1（埋め込み除外）	1,024 系列
Kimi K2 (Kimi Team 2025a)	MuonClip	—	2×10^{-4}、2×10^{-5} へ WSD コサイン減衰	500 ステップ	—	0.1	67M トークン

Table 9.7: 記録された指標（損失、クリッピング前の勾配ノルム、クリッピング率、相対ステップサイズ $\|\Delta\mathbf{W}\|_{\text{RMS}}/\|\mathbf{W}\|_{\text{RMS}}$）からの最適化失敗の診断。

ログ上の症状	診断	主な対処法
勾配ノルムの爆発（勾配爆発）を伴い最初の数ステップで損失が発散	ステップサイズが初期の局所的な鋭さ $2/L$ を超過	線形ウォームアップまたは大域ノルムクリッピングを追加
Adam で損失が高止まりし、初期の相対重み更新が $> 10\%$	コールドスタートのステップが初期化を破壊	線形ウォームアップ（全ステップの 1%–5%）を追加
ウォームアップを入れても Adam で損失が高止まり	ピーク学習率 η_0 が過大	η_0 を 2–3× 引き下げ
訓練を通じて $> 20\%$–50% のステップでクリッピングが作動	クリッピングが常時ブレーキとして作用	まず η_0 を引き下げ（θ の調整より優先）
ステップ間の大きな分散を伴い損失が早期にプラトーへ到達	現在の η/b における定常的なノイズ球	$\eta_t \to 0$ へ減衰、重み EMA で評価、または b を拡大
最終ステップまで一定の傾きで損失が滑らかに減少	累積移動量 B_T の予算不足	ピーク η_0 または総ステップ数 T を増加

Table 9.6 に倣い、パラメータ数 3.3M の `TinyLM` を以下のとおり設定する。

1. **バッチサイズ：**$b = 256$ 系列とする（ハードウェアのスループットが飽和し、かつウォームアップ後の勾配ノイズスケール $b_{\text{noise}} \sim 100$–$200$ と同程度になる領域である）。
2. **オプティマイザとガード：**$(\beta_1, \beta_2) = (0.9, 0.95)$ の AdamW、2D の隠れ層行列のみに適用する分離型重み減衰（decoupled weight decay）$\lambda = 0.1$（9.5.3.3 節）、$\theta = 1.0$ での大域ノルム勾配クリッピング、および短いウィンドウの重み EMA（$\alpha = 0.995$）を用いる。
3. **ピーク学習率の較正：**3 回の 300 ステップ（ホライズンの 10%）の試行を $\eta \in \{10^{-3}, 3 \times 10^{-3}, 10^{-2}\}$ にわたって行い、$\eta_0 = 3 \times 10^{-3}$ が最適であることを特定する（クリッピングされるステップは $< 5\%$ であり、$\theta = 1.0$ が純粋にガードとして機能していることが確認できる）。
4. **スケジュール：**60 ステップ（全体の 2%、$T = 3{,}000$）の線形ウォームアップに続けてゼロまでのコサイン減衰を行い、検証損失を定数学習率のベースラインより 0.05–0.06 nat 低減する（シード間のばらつきは < 0.01 nat）。

まず `tiny_lm(seed)` を定義して 4 ブロック、幅 256、4 ヘッドの `TinyLM` を生成し、ノイズスケール推定用にランダムな訓練ウィンドウを抽出する `sample_lm(b)` を定義する。

```
def tiny_lm(seed):
    """A TinyLM of four blocks of width 256."""
    return d2l.TinyLM(len(data.vocab), d_model=256, num_heads=4, num_blks=4,
                      rngs=nnx.Rngs(seed))

rng = np.random.default_rng(d2l.get_seed())

def sample_lm(b):  # b windows at random offsets in the training text, on the GPU
    X, Y = data.windows(data.train,
                        rng.integers(len(data.train) - data.num_steps, size=b))
    return jnp.asarray(X), jnp.asarray(Y)
```

訓練ハーネス `train(seed, schedule, num_steps)` は、$(\beta_1, \beta_2) = (0.9, 0.95)$ かつ重み減衰 0.1（埋め込み以外の 2D 行列に限定）の AdamW を設定し、大域ノルムクリッピング（`max_norm=1.0`）と重み EMA（`alpha=0.995`）でラップしたうえで、損失、ライブモデル、EMA モデル、およびクリッピング率を返す。

```
def decay_mask(params):  # True for the matrices, False for the rest
    return jax.tree_util.tree_map_with_path(
        lambda path, p: p.ndim == 2 and 'emb' not in str(path), params)

def train(seed, schedule, num_steps, max_norm=1.0, alpha=0.995):
    """AdamW with betas (0.9, 0.95) and weight decay 0.1 on the matrices at
    the rate schedule(t), clipped at max_norm, with an EMA of the weights;
    return the losses, the model, the EMA and the fraction of clipped steps."""
    model = tiny_lm(seed)
    adamw = optax.adamw(schedule, b1=0.9, b2=0.95, weight_decay=0.1,
                        mask=decay_mask)
    tx = optax.chain(clipped(adamw, num_steps, max_norm), ema_of_weights(alpha))
    optimizer = nnx.Optimizer(model, tx, wrt=nnx.Param)
    losses = d2l.train_steps(model, data, optimizer, num_steps)
    norms = optimizer.opt_state[0][0]['norms'][...]  # Recorded by clipped
    clip_rate = (norms > max_norm).mean()
    return losses, model, averaged(model, optimizer), float(clip_rate)
```

$b = 256$ において、初期化時点での `d2l.noise_scale` を計測し、3 回の 300 ステップ（ホライズンの 10%）の代理試行を $\eta \in \{10^{-3}, 3 \times 10^{-3}, 10^{-2}\}$ にわたって実行してピーク学習率 `peak` を選択し、さらに `noise_scale` を 300 ステップ後に再推定する。

```
data.batch_size = 256  # Where the throughput levels off
seed, short = new_seed(), {}
print('noise scale at initialization: '
      f'{d2l.noise_scale(tiny_lm(seed), sample_lm, b_small=8, b_big=256):.0f}')
for lr in [1e-3, 3e-3, 1e-2]:
    losses, model, _, clip_rate = train(seed, lambda t: lr, num_steps=300)
    short[lr] = (d2l.final_loss(losses), model)
    print(f'lr {lr:g}: loss {short[lr][0]:.3f} after 300 steps, '
          f'{clip_rate:.0%} of the steps clipped')
peak = min(short, key=lambda lr: short[lr][0])
print(f'peak rate {peak:g}; noise scale after its 300 steps: '
      f'{d2l.noise_scale(short[peak][1], sample_lm, b_small=8, b_big=256):.0f}')
```

```
noise scale at initialization: 2
lr 0.001: loss 1.880 after 300 steps, 2% of the steps clipped
lr 0.003: loss 1.769 after 300 steps, 3% of the steps clipped
lr 0.01: loss 2.482 after 300 steps, 6% of the steps clipped
peak rate 0.003; noise scale after its 300 steps: 173
```

最後に、2 つのシードにわたり 3,000 ステップの本番実行を行い、構築したスケジュール（`warmup_cosine(peak, 60, 3000)`）とその重み EMA を、`peak` での定数学習率ベースラインと比較する。

```
num_steps = 3000
val_loss = lambda model: math.log(d2l.lm_perplexity(model, data.val_dataloader()))
print(f'{"validation loss":<16}{"configured":>11}{"its EMA":>9}'
      f'{"clipped":>9}{"baseline":>10}{"gap":>7}')
results = []
for seed in [new_seed() for _ in range(2)]:
```

```
    _, model, ema, clip_rate = train(
        seed, warmup_cosine(peak, num_steps // 50, num_steps), num_steps)
    _, baseline, _, _ = train(seed, lambda t: peak, num_steps)
    results.append((val_loss(model), val_loss(baseline)))
    print(f'{"seed " + str(len(results)):<16}{results[-1][0]:11.3f}'
          f'{val_loss(ema):9.3f}{clip_rate:9.1%}{results[-1][1]:10.3f}'
          f'{results[-1][1] - results[-1][0]:7.3f}')
(c1, b1), (c2, b2) = results
print(f'spread between the seeds: configured {abs(c1 - c2):.3f}, '
      f'baseline {abs(b1 - b2):.3f}')
```

```
validation loss  configured  its EMA  clipped  baseline    gap
seed 1              1.247     1.247     0.6%     1.300   0.053
seed 2              1.249     1.249     0.6%     1.305   0.056
spread between the seeds: configured 0.002, baseline 0.006
```

9.10.4 まとめ

大域ノルム勾配クリッピング（$\mathbf{g} \leftarrow \min(1, \theta/\|\mathbf{g}\|_2)\mathbf{g}$）は、初期ステップや異常ステップのごく一部（$< 5\%$）でのみ作動するときには**ガード**として機能し、SGD における初期の発散を防ぎ、AdamW の二次モーメントバッファ $\mathbf{v}_t$ を勾配スパイクから保護する。一方で、絶えず作動し続ける場合は η が大きすぎてクリッピングが**ブレーキ**と化していることを示す。モデルの重みの**指数移動平均（EMA）**（$\bar{\mathbf{x}}_t = \alpha\bar{\mathbf{x}}_{t-1} + (1-\alpha)\mathbf{x}_t$）は、ライブの軌跡を変更することなく定常的なノイズ球の変動を平滑化し、定数学習率の実行において完全なコサイン減衰と同等の正解率を達成する。信頼性の高い大規模訓練では、ハードウェアを飽和させつつ b_{noise} 付近に設定したバッチサイズ、$(\beta_1, \beta_2) = (0.9, 0.95)$ かつ 2D 重み行列に限定した分離型重み減衰 $\lambda = 0.1$ を持つ AdamW、$\theta = 1.0$ での大域ノルムクリッピング、短ホライズンでのピーク学習率の較正、そして線形ウォームアップに続くコサイン減衰または WSD クールダウンを組み合わせる。

9.10.5 演習問題

1. [code] **クリッピングの領域：ガードとブレーキ**。$\theta \in \{0.01, 0.1, 0.5, 1.0, 4.0, \infty\}$ を $\eta = 1.0$ において `run` を用いて `TinyLM` 上で変化させて最終損失とクリッピング率を記録し、クリッピングを 20 ステップまたは 100 ステップの線形ウォームアップに置き換えた場合と比較せよ。

2. **ソフトマックスのロジットドリフトと PaLM の z 損失**。出力ロジット $\mathbf{o} \in \mathbb{R}^V$ と分配関数 $Z = \sum_{j=1}^{V} e^{o_j}$ に対し、交差エントロピー勾配の総和がゼロ（$\mathbf{1}^\top \nabla_{\mathbf{o}} \ell_{\text{CE}} = 0$）である一方、補助ペナルティ $10^{-4} \log^2 Z$ (Chowdhery et al. 2022) を加えると復元勾配 $2 \times 10^{-4} (\log Z)\,\mathbf{p}$ が生じ、$\log Z$ を 0 の方向へ引き戻しつつ $\mathbf{p} = \text{softmax}(\mathbf{o})$ の一様シフト不変性を損なわないことを示せ。

3. [code] **重み EMA のウィンドウ幅と開始時期**。EMA の減衰率 $\alpha \in \{0.9, 0.99, 0.999, 0.9999\}$ と開始エポック `start` $\in \{0, 5, 10\}$ を `train_fashion` 上で定数学習率 $\eta = 0.3$ のもとで変化させて実験し、正解率の軌跡をエポック単位の実効ウィンドウ幅 $1/(1-\alpha)$ と関連付けて考察せよ。

4. [code] **設定実例のアブレーション**。●4 ブロック、幅 256 の `TinyLM`（`train` で定義）において、2 シードのアブレーションを実行し、以下を検証せよ。(a) WSD の終盤線形減衰とコサイン減衰の比較、(b) ウォームアップの除去または勾配クリッピングの除去、(c) ピーク学習率

の 10^{-3} または 10^{-2} への変更、(d) 定数学習率ベースラインに対する $\alpha \in \{0.995, 0.999\}$ での重み EMA の評価。

9.11 関連文献

本節では、本章の手法を発展させた基礎的な教科書、サーベイ論文、および近年の研究をまとめる。本文中で引用した収束率（convergence rate）の数学的証明は第 35 章で与えられており（理論に関する追加の文献は 35.9 節を参照）、自動ハイパーパラメータ最適化は 第 41 章（英語版）で、複数デバイスによる分散訓練システムは第 13 章で扱っている。

9.11.1 基礎的な教科書、サーベイ、プレイブック

- **凸最適化と数値最適化：**@Boyd.Vandenberghe.2004（『Convex Optimization』）は、条件付け（conditioning）、勾配降下法、ニュートン法（9.1 節）、および一般のノルムのもとでの最急降下法（steepest descent、9.6 節）に関する標準的な参考書である。Nocedal and Wright (2006)（『Numerical Optimization』）は、ニュートン法、準ニュートン法、直線探索、信頼領域法を包括的に扱っている。
- **確率的最適化：**@Bottou.Curtis.Nocedal.2018（「Optimization Methods for Large-Scale Machine Learning」）は、確率的勾配のノイズ球、ステップサイズの減衰（9.2 節）、および分散低減（variance reduction）を概説している。
- **対話型ガイドと実務者向けプレイブック：**Gabriel Goh による「Why Momentum Really Works」（Distill、2017 年）は、臨界減衰と $\sqrt{\kappa}$ 加速則（9.4 節）の対話的な可視化を提供している。Jeremy Bernstein による「Deriving Muon」および Bernstein and Newhouse (2024) は、ノルム球にわたる最急降下法を統一的に導出している（9.6 節）。EleutherAI の「Practitioner's Guide to Maximal Update Parameterization」は μP の座標チェック（9.9 節）を詳述しており、Google Research の『Deep Learning Tuning Playbook』(Godbole et al. 2023) は、9.3 節および 9.10 節全体で用いられる科学的（scientific）・局外（nuisance）・固定（fixed）ハイパーパラメータの区別を定式化している。

9.11.2 損失曲面

9.11.2.1 高次元損失ランドスケープの可視化とモード連結性

ニューラルネットワークの損失曲面の低次元断面は、正の斉次性によるスケール不変性（$c\,\mathbf{W}_1$ と $c^{-1}\mathbf{W}_2$ は同じ ReLU ネットワークを計算する）を取り除くために摂動方向をフィルタごとに正規化しなければ、誤解を招くものになりかねない。フィルタ正規化したスライスを用いることで、Li et al. (2018) は残差接続（7.4 節）がぎざぎざした非凸な曲面を滑らかな盆地へと変えることを示した。

独立に訓練された 2 つのネットワーク間を線形補間すると高い損失の障壁を横切ることになるが、Garipov et al. (2018) および Draxler et al. (2018) は**モード連結性**（mode connectivity）を発見した。すなわち、1 回だけ折れ曲がる単純な区分線形曲線によって、訓練損失・テスト損失ともに一様に低い経路に沿って独立な極小点どうしを結ぶことができる。さらに Entezari et al. (2022) と Ainsworth et al. (2023) は、置換対称性（9.1 節）を用いて隠れユニットを揃えると、幅の広いネットワーク間の線形の障壁がほぼ消失することを実証した。

9.11.2.2 平坦な極小点と Sharpness-Aware Minimization (SAM)

より平坦な極小点ほど汎化性能が高いという仮説に基づき、**Sharpness-Aware Minimization (SAM)** (Foret et al. 2021) は、降下方向の勾配を評価する前に勾配上昇方向の摂動ステップを踏

むことで、現在の重みの周囲の ℓ_2 球内における最悪ケースの損失を最小化する（1 回の更新につき $2\times$ の勾配コストがかかる）。ただし、そのままのヘッセ行列の鋭さ（sharpness）は層ごとの再パラメータ化に対して不変ではないことに注意が必要である (Dinh, Pascanu, et al. 2017)。

9.11.3 古典的な 2 次手法と分散低減手法

9.11.3.1 準ニュートン法、直線探索、信頼領域法

準ニュートン法（quasi-Newton method）は、ヘッセ行列の逆行列 $\mathbf{H}^{-1}$ を、1 次の勾配のみを用いた連続するセカント対 $(\mathbf{x}_{t+1}-\mathbf{x}_t, \mathbf{g}_{t+1}-\mathbf{g}_t)$ から近似する。記憶制限 BFGS（**L-BFGS**、Liu and Nocedal (1989)）は直近 m 個の対を $O(md)$ のメモリに保持し、決定論的で滑らかな問題に対する定番の手法であるが（35.2.2.2 節）、ミニバッチ勾配の差分に対しては敏感である。**直線探索**（line search、Armijo の十分減少条件の判定により η_t を動的に選ぶ手法）および**信頼領域法**（trust-region method、35.2.3 節、Muon の信頼領域解析については Kovalev (2025) を参照）も同様に、決定論的最適化において威力を発揮する。

9.11.3.2 ヘッセ行列フリー法と鞍点フリーニュートン法

$d \times d$ のヘッセ行列を陽に作らずに 2 次手法を深層ネットワークへ適用するため、**ヘッセ行列フリー最適化**（Hessian-free optimization）(Martens 2010) は、半正定値であるガウス・ニュートン行列上のヘッセ行列ベクトル積（34.5.3.2 節）を用いた共役勾配法によって $\mathbf{H}\mathbf{d} = -\mathbf{g}$ を近似的に解く。一方、**鞍点フリーニュートン法**（Saddle-Free Newton）(Dauphin et al. 2014) は、負の曲率方向が反復点を引き寄せるのではなく反発させるよう、低次元のクリロフ部分空間上で $|\mathbf{H}|^{-1}$ による前処理を施す。

9.11.3.3 確率的収束率と分散低減

一般の凸目的関数に対して、ミニバッチ SGD はミニマックス最適な $O(1/\sqrt{T})$ の収束率を達成し (Nemirovski and Yudin 1983)（35.4.3 節）、非凸で滑らかな目的関数に対しては $\min_{t\le T}\mathbb{E}[\|\nabla f(\mathbf{x}_t)\|^2] = O(1/\sqrt{T})$ を保証する (Ghadimi and Lan 2013)（35.4.3.2 節）。Polyak–Ruppert の反復平均化 (Polyak and Juditsky 1992) は、最適な漸近共分散を達成する（9.10.2 節）。有限のデータセット上では、**SVRG** (Johnson and Zhang 2013)（35.4.4 節）がデータセット全体にわたる周期的な制御変量（control variate）によって定常的なノイズ球を解消するが、深層学習においてはデータ拡張や急速に変化する特徴がその有効性を制限する。

9.11.4 行列オプティマイザと構造的オプティマイザ

- **自然勾配、K-FAC、SOAP：**出力分布の空間でステップ長を測る**自然勾配**（natural gradient）(Amari 1998) は、フィッシャー情報行列の逆行列によって更新を前処理する。**K-FAC** (Martens and Grosse 2015) は、層ごとのフィッシャー行列ブロックを入力の共分散と出力勾配の共分散のクロネッカー積として近似する（35.5.5 節）。一方、**SOAP** (Vyas et al. 2024) は、Shampoo の左右の前処理行列 (Gupta et al. 2018) がなす固有基底の上で Adam を実行する。
- **Muon の理解とスケーリング：**@Wang.Zhang.Li.ea.2025 は、AdamW に対する Muon の高速化の要因が、裾の特徴（tail feature）における連想メモリ射影（アテンションの値・出力行列および MLP 行列）にあることを突き止めた。これは希少トークンにおける SGD に対する Adam の優位性 (Kunstner et al. 2023, 2024) と対をなす結果である。また Shah et al. (2025) は、Muon が AdamW よりも大きなバッチサイズまでデータ効率を維持することを明らかにしている。**Scion** (Pethick et al. 2025) は、AdamW にフォールバックすることなく、作用素ノルムによる最急降下法をすべての層（埋め込みや読み出しヘッドを含む）へと拡張する。

- **効率的・分散的な直交化：**まず **Polar Express** (Amsel et al. 2026) と Cesista et al. (2025) は、ステップごとのニュートン・シュルツ多項式の係数（Figure 9.10）を最適化する。また Kim and Oh (2026) は非厳密な q ステップ直交化のもとでの収束を解析し、**Dion** (Ahn et al. 2025) はシャーディングされた訓練におけるデバイス間通信を削減するため、低ランクのモメンタム近似を直交化する。

9.11.5 スケジュール、平均化、大規模ベンチマーク

- **ウォームリスタート、スケジュールフリー最適化、モデルスープ：**コサインアニーリングはもともと周期的なウォームリスタートを伴う手法（**SGDR**、Loshchilov and Hutter (2016)）として導入され、サイクル間で予測を平均する**スナップショットアンサンブル**（snapshot ensemble）(Huang, Li, et al. 2017) を可能にした。これに対し、**モデルスープ**（model soup）(Wortsman et al. 2022) は共通の初期化からファインチューニングされた複数のモデルの重みを平均する。**Schedule-Free SGD および AdamW** (Defazio et al. 2024) は、現在の反復点と反復点の移動平均との補間点で勾配を評価することにより、あらかじめホライズン T を定めるスケジュールを不要にする。
- **バッチサイズランプとスケーリング則：**@Malladi.Lyu.Panigrahi.ea.2022 は、バッチサイズの増大に伴う Adam のモーメント $(\beta_1, \beta_2, \epsilon)$ に対する完全な確率微分方程式（SDE）スケーリング則を導出している。H. Zhang et al. (2024) および Merrill et al. (2025) は、臨界バッチサイズがモデルサイズではなく主に処理済みトークン数に応じて増大することを示しており、Dey et al. (2023)、DeepSeek-AI, Bi, et al. (2024)、および Meta AI (2025) は、実用言語モデルにおけるスケール間のハイパーパラメータ転移を報告している。
- **オプティマイザのベンチマークと評価プロトコル：**新しいオプティマイザ（LARS/LAMB (You et al. 2017) や NorMuon (Li et al. 2026) など）による見かけ上の改善は、ベースライン側の局外ハイパーパラメータを同等の労力で調整すると、縮小または消失することが少なくない (Nado et al. 2021; Schmidt et al. 2021; Wen et al. 2025)。**AlgoPerf** (Dahl et al. 2023; Kasimbeg et al. 2025)、言語モデルのベンチマークスイート (Semenov et al. 2025)、および **NanoGPT speedrun** の最適化トラック (Jordan and contributors 2024; Jordan 2026) といった標準化されたベンチマークでは、明示的な調整予算と統計的有意性のしきい値が課されている。最後に、公開されている **OPT-175B 訓練ログブック** (Zhang et al. 2022) には、大規模環境における訓練途中の損失スパイク、チェックポイントの巻き戻し、ハードウェア復旧のありのままの日次記録が残されている。

Chapter 10

アテンション

第 8 章のリカレントモデルは、系列を 1 トークンずつ読み込み、それまでに見たすべてを固定サイズの隠れ状態に圧縮する。長い文書の末尾付近での予測が最初の段落の細部に依存する場合、リカレントネットワークはその細部が必要になると知るはるか前に、隠れ状態へ書き込んでおかなければならない。履歴を単一のベクトルに圧縮するのではなく、ネットワークがすべての位置に表現を保持し、現在の予測に必要なものだけを取り出すことはできないだろうか。

アテンションはまさにこれを実現する。各ルックアップでは、現在の位置が求めるものを表す**クエリ**ベクトルに対し、すべての候補位置にある**キー**ベクトルとの間でスコアを計算する。ソフトマックスがそれらのスコアを和が 1 になる非負の重みへと変換し、層はそれらの位置に格納された**値**（value）ベクトルの重み付き平均を返す。重み付き平均は滑らかであるため、勾配降下法によってクエリ、キー、値を生成する射影をエンドツーエンドで訓練できる。このようなルックアップを複数並列に実行すると、**マルチヘッドアテンション**が得られる。アテンションは、ニューラル機械翻訳でデコーダが原言語のすべての単語を直接参照できるように導入され (Bahdanau et al. 2015)、Transformer における唯一の位置混合演算となった (Vaswani et al. 2017)。

再帰構造を全ペア間のルックアップに置き換えると、新たに 3 つの課題が生じる。

1. **順序への不感性：**重み付き和は入力の置換に対して不変であるため、位置情報を明示的に注入しなければならない。
2. **二次スケーリング：**n 個のトークンの全ペアに対してスコアを計算するには $O(n^2)$ の演算とメモリが必要であり、これが IO を考慮した厳密カーネルと準二次近似の両方の動機となる。
3. **機構的解釈：**アテンション重みが示すのはヘッドが**どこを**読み取るかであり、層をまたいで**どのような**関数を計算するかではない。

最初の 3 つの節では、微分可能なキー・値ルックアップ（10.1 節）、スケール付きドット積スコアリングとブールマスキング（10.2 節）、そしてマルチヘッド自己アテンション（10.3 節）によってこの層を構築する。続く 10.4 節では、絶対、回転（RoPE）、および相対（ALiBi）の位置エンコーディングによって系列の順序を扱う。次の 2 つの節では計算スケールを取り上げる。10.5 節では $n \times n$ のスコア行列を実体化しないタイル化オンラインソフトマックスカーネル（FlashAttention）を導出し、10.6 節ではスライディングウィンドウアテンションと線形アテンションの変種を検討する。最後に、10.7 節では残差ストリーム（residual stream）、QK/OV 分解、および 2 層の誘導ヘッド（induction head）を通じて訓練済みアテンション回路を解析し、10.8 節では各種の拡張と一次文献を概観する。

第 11 章へ直接進む読者は、最初の 4 つの節と 10.7 節冒頭の残差ストリームの定式化だけを押さ

えればよい。計算コストに関する各節は、11.3 節および第 12 章と併せて読むことができる。

10.1 クエリ、キー、値

リカレントニューラルネットワーク（8.4 節）は、後の予測でどの詳細が必要になるかを知る前に、それまでに読み込んだすべての情報を固定サイズの状態ベクトルへと圧縮する。たとえば、隠れ次元が 768 のリカレントモデルが 1,024 トークンを読み込む場合、1,024 個すべての位置の情報を 768 個の数値に要約しなければならない。

データベースは、レコードをそのまま保存し、クエリが届いた時点で初めて何を取り出すかを決定することで、時期尚早な圧縮を回避する。しかし、厳密なデータベース参照（lookup）は不連続である。たとえば、キーと値のペア $\mathcal{D} = \{(2, 10), (5, 20)\}$ が与えられたとき、完全一致クエリ $q = 5$ は 20 を返すが、$q = 4$ はどのキーにも一致しない。勾配降下法で訓練するため、ハード参照を**ソフト**参照に置き換える。すなわち、負の二乗距離 $a(q, k) = -(q - k)^2$ などですべてのキーにスコアを付け、ソフトマックスによってそのスコアを凸重みへと変換する。$q = 4$ のとき、キー 2 と 5 のスコアはそれぞれ -4 と -1 であり、重みは $e^{-4}/(e^{-4} + e^{-1}) \approx 0.05$ と 0.95、重み付き値は $0.05 \cdot 10 + 0.95 \cdot 20 \approx 19.5$ となる。この出力はクエリに対して滑らかに変化し、保存されたレコード数がいくつであっても対応でき、クエリを実行する前に固定サイズのボトルネックを設ける必要もない。

この微分可能なキー・値参照が**アテンション機構**（attention mechanism）である (Bahdanau et al. 2015)。本節ではまず、ベクトルのクエリ・キー・値に対するアテンションプーリングを定式化し、続いてその古典的なノンパラメトリック手法の先駆けである Nadaraya–Watson カーネル回帰を検討する。そこでは、単一のバンド幅（bandwidth）パラメータが参照の鋭さを制御する。

10.1.1 微分可能な参照としてのアテンション

$\mathcal{D} \stackrel{\text{def}}{=} \{(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)\}$ を、m 個のキー・値ペアからなるデータベースとし、キーを $\mathbf{k}_i \in \mathbb{R}^d$、値を $\mathbf{v}_i \in \mathbb{R}^{d_v}$ とする。クエリ $\mathbf{q} \in \mathbb{R}^d$ が与えられたとき、**アテンションプーリング**は次の値の重み付き和を返す。

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i, \tag{10.1}$$

ここで、$\alpha(\mathbf{q}, \mathbf{k}_i) \geq 0$ かつ $\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i) = 1$ である。したがって出力は値の凸結合（convex combination）となり、完全一致参照（ワンホット重みベクトル）と平均プーリング（一様な重み $1/m$）の間を補間する。

任意の実数値をとる適合度スコア（compatibility score）$a(\mathbf{q}, \mathbf{k}_i)$ を凸重みへと写すには、ソフトマックス関数（3.1 節）を適用する。

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_j \exp(a(\mathbf{q}, \mathbf{k}_j))}. \tag{10.2}$$

$\alpha_i = 1/\big(1 + \sum_{j \neq i} e^{a_j - a_i}\big)$ であるため、重みはスコアの差 $a_j - a_i$ のみに依存し、すべてのキーにわたって定数を加えても不変である。ソフトマックスも線形結合も、$\mathbf{q}$、$\mathbf{k}_i$、$\mathbf{v}_i$ に関して微分可能である。

具体的なベクトルの例として、$a(\mathbf{q}, \mathbf{k}) = -\|\mathbf{q} - \mathbf{k}\|^2$ とし、クエリを $\mathbf{q} = (1, 1)$、キーを $\mathbf{k}_1 = (1, 1)$、$\mathbf{k}_2 = (2, 1)$、$\mathbf{k}_3 = (0, 0)$、値を $\mathbf{v}_1 = (1, 3)$、$\mathbf{v}_2 = (0, 0)$、$\mathbf{v}_3 = (4, 1)$ とする。スコアは 0、-1、-2 となり、ソフトマックス重み $(1, e^{-1}, e^{-2})/(1+e^{-1}+e^{-2}) \approx (0.67, 0.24, 0.09)$ と次の出力が得られる。

$$\mathbf{o} = 0.67\,(1, 3) + 0.24\,(0, 0) + 0.09\,(4, 1) \approx (1.0, 2.1). \tag{10.3}$$

Figure 10.1 に示されている 2 つの性質に注目しよう。第 1 に、$\mathbf{k}_1$ は $\mathbf{q}$ と完全に一致しているにもかかわらず、重みの 2/3 しか受け取らない。スコアを 10 倍にスケーリングすると $(0, -10, -20)$ になり)、重みの 99.995% が $\mathbf{k}_1$ に集中する。このように、スコアの大きさはソフトマックスがどれだけ鋭く 1 つの候補に絞り込むかを支配する。第 2 に、$\mathbf{v}_2 = (0, 0)$ は和への寄与がゼロであるにもかかわらず、その 0.24 という重みは他の値を希釈することで出力を原点の方へと引き寄せる。

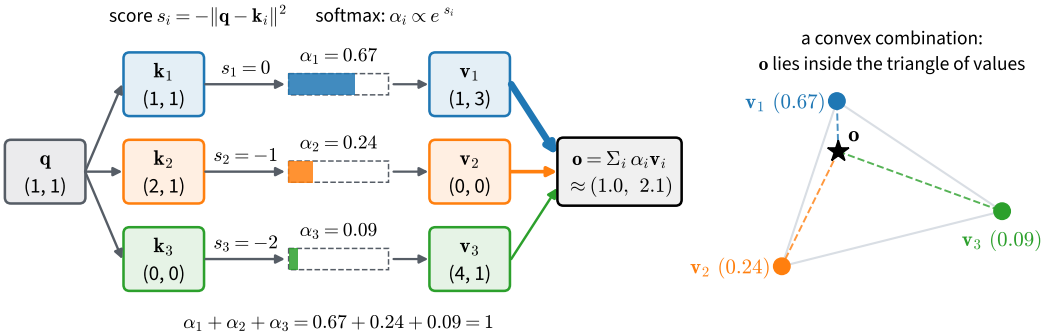


Figure 10.1: クエリ $\mathbf{q} = (1, 1)$ の 3 つのキー・値ペア（キーは $(1, 1)$、$(2, 1)$、$(0, 0)$、値は $(1, 3)$、$(0, 0)$、$(4, 1)$）に対するソフト参照。スコア $s_i = -\|\mathbf{q} - \mathbf{k}_i\|^2$ は総和が 1 となるソフトマックス重み α_i に変換され（左）、出力 $\mathbf{o} \approx (1.0, 2.1)$ は値の重み付き和として、各値を頂点とする三角形の内部の点になる（右）。

アテンションの規則 Equation 10.1 は任意のレコード数 m に対してそのまま適用できるため、単一の層で任意の長さの系列を固定サイズの状態に圧縮することなく処理できる。その代わり、コストはメモリへと移る。幅 768 のリカレント層が保持する数値は 768 個であるのに対し、GPT-2 small の 12 ヘッドアテンション層（ヘッドあたり $d = 64$）が 1,024 トークンを処理する場合、ヘッドあたり $2 \times 1{,}024 \times 64 = 131{,}072$ 個のキーと値の要素、すなわち全 12 ヘッドでは約 1.6×10^6 個の数値を保持する。

10.1.2 固定カーネルによるアテンションプーリング

深層学習が登場するずっと前から、**Nadaraya–Watson カーネル回帰**（Nadaraya–Watson kernel regression）(Nadaraya 1964; Watson 1964) はノンパラメトリック回帰に Equation 10.1 を用いていた。そこでは訓練入力をキー、訓練ラベルを値とし、固定の類似度カーネルをスコア付けの規則として扱う。

10.1.2.1 カーネルと Nadaraya–Watson 推定量

非負の類似度カーネル $K(\mathbf{q}, \mathbf{k})$、たとえば**バンド幅** $\sigma > 0$ をもつ次のガウスカーネルが与えられたとする。

$$K(\mathbf{q}, \mathbf{k}) = \exp\left(-\frac{\|\mathbf{q} - \mathbf{k}\|^2}{2\sigma^2}\right), \tag{10.4}$$

これをキーにわたって正規化すると、**Nadaraya–Watson 推定量**（Nadaraya–Watson estimator）が得られる。

$$f(\mathbf{q}) = \sum_i \mathbf{v}_i \frac{K(\mathbf{q}, \mathbf{k}_i)}{\sum_j K(\mathbf{q}, \mathbf{k}_j)}. \tag{10.5}$$

つねに正の値をとる任意のカーネルに対して、Equation 10.5 はスコアを $a(\mathbf{q}, \mathbf{k}) = \log K(\mathbf{q}, \mathbf{k})$ としたソフトマックスアテンション Equation 10.2 に一致する。ガウスカーネルでは $a(\mathbf{q}, \mathbf{k}) = -\|\mathbf{q} - \mathbf{k}\|^2/(2\sigma^2)$ となり、$2\sigma^2 = 1$ のときに冒頭の例が復元される。密度関数と目標関数に関する正則性条件のもとで、$m \to \infty$ かつ $\sigma \to 0$ を適切な速度で近づけると、$f(\mathbf{q})$ は条件付き期待値 $\mathbb{E}[y \mid \mathbf{q}]$ に収束する (Mack and Silverman 1982)。

10.1.2.2 バンド幅

バンド幅 σ が推定量をどのように制御するかを見るため、$n = 40$ 個のノイズを含む訓練観測値を $y = 2\sin(x) + x + \epsilon$（ただし $\epsilon \sim \mathcal{N}(0, 1)$、定義域は $[0, 5)$）から生成し、密なグリッド上で予測を評価する。

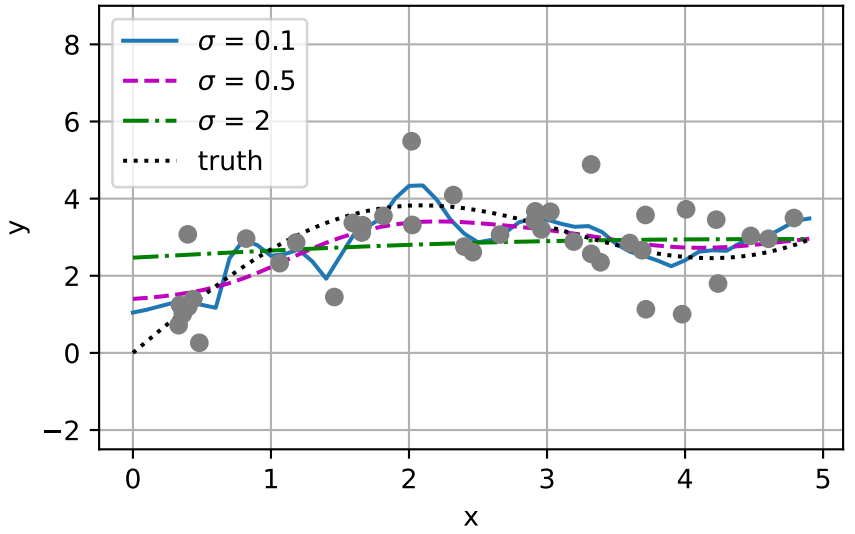
```
n = 40
x_train = jnp.sort(jax.random.uniform(d2l.get_key(), (n,)) * 5)
y_train = (2 * jnp.sin(x_train) + x_train
           + jax.random.normal(d2l.get_key(), (n,)))
x_val = jnp.arange(0, 5, 0.1)
y_val = 2 * jnp.sin(x_val) + x_val
```

実装では、クエリとキーのすべてのペア間の距離を計算し、ガウスカーネルを評価してクエリごとの各行を正規化したうえで、`y_train` を掛ける。

```
def nadaraya_watson(x_train, y_train, x_val, sigma):
    dists = x_val[:, None] - x_train[None, :]  # (queries, keys)
    k = d2l.exp(-dists**2 / (2 * sigma**2))
    attention_w = k / k.sum(1, keepdims=True)  # Normalize over the keys
    return attention_w @ y_train, attention_w
```

$\sigma \to 0$ の極限では、すべての重みが最も近い訓練点に集中し（1 最近傍補間）、$\sigma \to \infty$ の極限では、すべての重みが $1/n$（全体の標本平均）に近づく。以下に $\sigma \in \{0.1, 0.5, 2.0\}$ の場合をプロットする。

```
sigmas = (0.1, 0.5, 2.0)
estimates = [nadaraya_watson(x_train, y_train, x_val, s)[0] for s in sigmas]
d2l.plot([x_val] * 4 + [x_train], estimates + [y_val, y_train], 'x', 'y',
         legend=[rf'$\sigma$ = {s:g}' for s in sigmas] + ['truth'],
         fmts=('-', 'm--', 'g-.', 'k:', 'C7o'), ylim=(-2.5, 9),
         figsize=(5, 3), legend_loc='upper left')
```



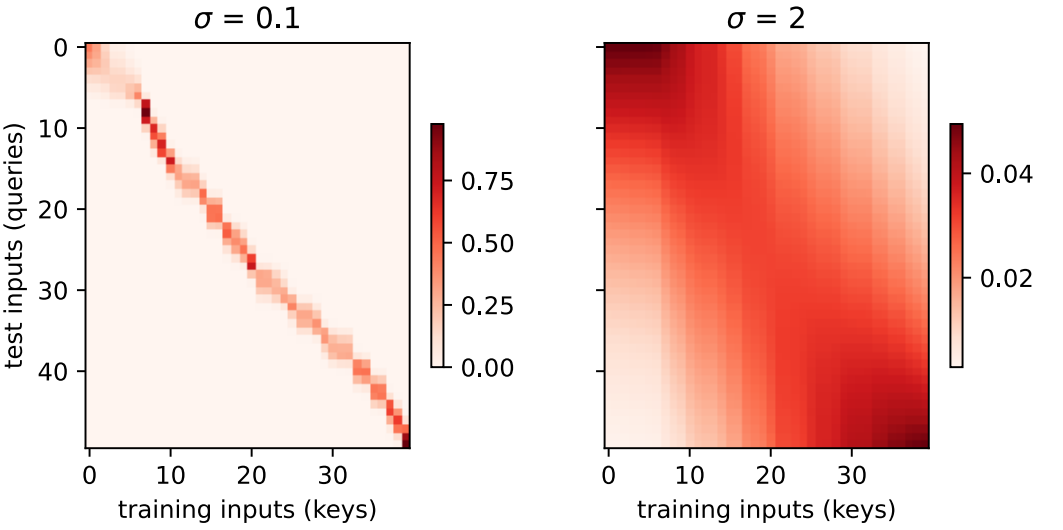
狭いバンド幅（$\sigma = 0.1$）はノイズを含む個々のラベルに過学習する一方、広いバンド幅（$\sigma = 2.0$）は過剰に平滑化して水平線へと近づく。中間の値である $\sigma = 0.5$ は、近傍点がすべて片側に偏る境界付近を除いて真の曲線によく追従する。実用上、σ は 1 つ抜き交差検証（leave-one-out cross-validation）によって選択でき、それには重み行列の対角成分を 0 にして各行を再正規化するだけでよい（演習問題 5）。

より広く見れば、$1/\sigma^2$ はソフトマックスの前に生のスコア $-\frac{1}{2}(q-k)^2$ に掛かる逆温度（inverse temperature）として働く。スコアを拡大するとペア間の差が引き伸ばされ、ソフトマックスはワンホット参照へと近づく。逆にスコアを縮小すると差が圧縮され、ソフトマックスは一様平均へと近づく。これと同じ温度スケーリングは、ドット積アテンション（10.2 節）でも再び登場する。

10.1.3 診断手段としてのアテンション重み

アテンション行列 **A** の各行はキー上の確率分布であるため、**A** をヒートマップとしてプロットすれば、各クエリがどのキーを参照しているかがわかる。

```
w_narrow, w_wide = (nadaraya_watson(x_train, y_train, x_val, s)[1]
                    for s in (0.1, 2.0))
# show_heatmaps takes (rows of panels, columns of panels, queries, keys)
d2l.show_heatmaps(d2l.stack([w_narrow, w_wide])[None],
                  xlabel='training inputs (keys)',
                  ylabel='test inputs (queries)',
                  titles=[r'$\sigma$ = 0.1', r'$\sigma$ = 2'],
                  figsize=(7, 3), cbar='each')
```



$\sigma = 0.1$ では、重みは最も近い訓練入力の周囲の狭い対角帯に集中する。一方、$\sigma = 2.0$ では重みはデータセット全体に広く分散し、どの重みも一様な値 $1/40 = 0.025$ の約 2 倍を超えない。最大となる約 0.05 の重みは端にあるクエリに属するもので、そこでも遠いキーが受け取る重みはわずか約 0.002 にとどまる。

深いネットワークでアテンションのヒートマップを観察する際には、2 つの注意点がある。

1. **重みは層がどこを見ているかを示すのであって、何を伝達するかを示すのではない：**あるキーの値ベクトルがゼロであっても（先のベクトルの例における $\mathbf{v}_2 = (0, 0)$ のように）、あるいは下流の射影の零空間（null space）に含まれていても、そのキーは大きなアテンション重みを受け取り得る。ヘッドの因果的な役割を検証するには、介入実験（interventional experiments）が必要である（10.7 節）。
2. **固定カーネルは静的な類似度計量を強いる：**すべてのクエリは Equation 10.5 において入力空間の同じユークリッド距離を用いる。しかし系列モデリングでは、関連性はコンテキストやタスクに依存する。たとえば、動詞は数トークン離れた構文上の主語を必要とするかもしれない。そのため、学習されるアテンション層では、スコアを計算する前に入力を訓練可能なクエリ・キー・値の空間へと射影する。

10.1.4 まとめ

アテンションプーリングは値ベクトルの凸結合 $\sum_i \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i$ を計算し、そのソフトマックス重み $\alpha(\mathbf{q}, \mathbf{k}_i)$ は厳密な辞書参照と一様な平均プーリングの間を滑らかに補間する。固定サイズの再帰状態とは異なり、アテンションはペア数 m に依存しない規則によって保存されたすべてのキー・値ペアに直接アクセスするが、作業メモリは m に対して線形に増大する。Nadaraya–Watson カーネル回帰は固定のユークリッド類似度カーネルを用いたアテンションプーリングに対応し、そのバンド幅 σ は統計的バイアスと分散のトレードオフを支配するソフトマックス温度として働く。アテンションのヒートマップはクエリとキーのアライメントを可視化するものの、固定された入力空間の距離を超えるには、訓練可能な射影と学習されるスコア関数が必要である。

10.1.5 演習問題

1. **2 つの値の平均。**キー $\mathbf{k}_1, \ldots, \mathbf{k}_m$ を正規直交とし、スコアを $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k}$、重みを Equation 10.2 によるものとする。

1. クエリ $\mathbf{q} = c\,(\mathbf{k}_a + \mathbf{k}_b)$（$c > 0$）に対して m 個すべての重みを計算し、$c \to \infty$ のとき出力が $\frac{1}{2}(\mathbf{v}_a + \mathbf{v}_b)$ に収束することを示せ。
2. 選択された 2 つのキーが重み全体の少なくとも 99% を受け取るために必要な c の大きさは、$m = 10$ のときどれだけか。
3. 次に、キーが互いに直交し $\|\mathbf{k}_b\| = 2\,\|\mathbf{k}_a\|$ を満たすとする。再び $\mathbf{v}_a$ と $\mathbf{v}_b$ の平均をとるクエリを求め、さらに $\mathbf{q} = c\,(\mathbf{k}_a + \mathbf{k}_b)$ が $c \to \infty$ のとき代わりに何を返すか述べよ。

Stanford CS224N、課題 4 の問題 1(b) および 1(c) を改題。

2. [code] **有効キー数。** 1 つのクエリに対する重み $\alpha_i(q)$ について、有効キー数を $n_{\text{eff}}(q) = 1/\sum_i \alpha_i(q)^2$ と定義する。

   1. 値を $y_i = g(x_i) + \epsilon_i$（ノイズは互いに独立で分散 s^2）とし、キーを固定する。推定量 Equation 10.5 の q における分散が $s^2/n_{\text{eff}}(q)$ であることを示せ。
   2. n_{eff} を `x_val` 内のすべてのクエリにおいて、`nadaraya_watson` が返す重みから `sigmas` 内の各バンド幅について計算し、3 本の曲線をプロットせよ。次に、`y_train` のノイズを 200 回引き直し（`x_train` は固定）、各クエリにおける推定値の標準偏差を $1/\sqrt{n_{\text{eff}}(q)}$ と比較せよ。

3. [code] **アテンションの勾配。** ●$a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^\top \mathbf{k}_i$ であり、かつ $i = 1, \dots, m$ に対して $\mathbf{k}_i = \mathbf{v}_i$ であるとする。ソフトマックス正規化 Equation 10.2 が定めるキー上の分布を $p(\mathbf{k}_i; \mathbf{q})$ と表す。

   1. 次式を証明せよ。
   $$\nabla_{\mathbf{q}} \operatorname{Attention}(\mathbf{q}, \mathcal{D}) = \operatorname{Cov}_{p(\mathbf{k}_i;\mathbf{q})}[\mathbf{k}_i]. \tag{10.6}$$
   2. このヤコビ行列が半正定値であることを結論づけ、どのようなデータベースに対して零行列になるかを求めよ。
   3. $\mathbf{q} = c\,\mathbf{u}$ とおく。ここで固定された方向 $\mathbf{u}$ に対し、ちょうど 1 つのキーが最大のスコア $\mathbf{u}^\top \mathbf{k}_i$ を達成するとする。$c \to \infty$ としたときのヤコビ行列の極限を求め、それが勾配降下法によるクエリの訓練に対して何を意味するか述べよ。
   4. $m = 8$、$d = 4$ のランダムなデータベース上で、自動微分を用いてこの恒等式と極限を検証せよ。

4. [code] **検索としてのソフト参照。** ●m 個のキーを $\mathbb{R}^{16}$ の単位球面上から一様に引いてデータベースを構築し、値 $\mathbf{v}_i$ をインデックス i のワンホット指示ベクトルとする。これらをバンド幅 σ のガウスカーネルで重み付けし、Equation 10.5 と同様に正規化して、出力が重みベクトルになるようにする。保存されたキーにノイズを加えたコピー $\mathbf{q} = \mathbf{k}_i + \epsilon$（ただし $\epsilon \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I})$）をクエリとする。

   1. すべての $\sigma > 0$ に対して最大の重みはクエリに最も近いキーに割り当てられ、したがってバンド幅はどのインデックスが勝者になるかには影響しないことを示せ。ただし、バンド幅は勝者が重みのどれだけの割合を占めるかには影響する。
   2. ノイズがない場合、正解キーに対する重みは次式となる。
   $$\alpha_i = \frac{1}{1 + \sum_{j \neq i} \exp\left(-\|\mathbf{k}_i - \mathbf{k}_j\|^2 / 2\sigma^2\right)}. \tag{10.7}$$
   ランダムな単位ベクトルに対して $\mathbf{k}_i^\top \mathbf{k}_j \approx 0$、すなわち $\|\mathbf{k}_i - \mathbf{k}_j\|^2 \approx 2$ と近似し、α_i が 0.9 まで下がるバンド幅を $m = 20$ および $m = 200$ について導出せよ。σ は m が大きくなるにつれてどのように縮小しなければならないか。
   3. 次の層が出力を利用できるような参照では、単に最大の割合を得るだけでなく、重みの大半が 1 つのキーに集中する必要がある。$\alpha_i \geq 0.5$ のときクエリの検索に成功し

たとよぶことにする。τ を 0 から 1 まで動かして、$m = 20$ と $m = 200$ のそれぞれで各 m に対する (b) のバンド幅を用い、検索正解率を τ に対してプロットせよ。重みはスコア $-\|\mathbf{q} - \mathbf{k}_j\|^2/2\sigma^2$ のソフトマックス Equation 10.2 として計算せよ。クエリから正解キーまでの距離が変わらないにもかかわらず、m を大きくすると固定した τ での正解率が低下する理由を説明せよ。

5. [extended] **学習される計量。**●学習された距離を用いるカーネル回帰 (Weinberger and Tesauro 2007) は、Equation 10.5 の固定類似度を学習された類似度に置き換える。$n = 200$ 個の入力 $\mathbf{x}_i$ を $[0, 5]^5$ から一様に引き、ラベルを標準ガウスノイズ付きの $y_i = 2\sin(x_{i1}) + x_{i1} + \epsilon_i$ とすることで、第 1 座標のみが意味をもつようにする。クエリとキーのスコアを

$$a(\mathbf{q}, \mathbf{k}) = -\tfrac{1}{2}\|\mathbf{W}(\mathbf{q} - \mathbf{k})\|^2, \qquad \mathbf{W} = \mathrm{diag}(w_1, \dots, w_5), \tag{10.8}$$

で与え、Equation 10.2 と同様に正規化し、$\mathbf{W}$ を 1 つ抜き誤差（$\mathbf{x}_i$ における推定値からペア $(\mathbf{x}_i, y_i)$ を除外した誤差）に当てはめよ。新たに生成した 1000 個の入力上で、ノイズのない関数に対するテスト誤差を測定せよ。

   1. 等方的なカーネル $w_1 = \cdots = w_5 = 1/\sigma$ を、σ に関するグリッド探索によって当てはめよ。
   2. 等方的な解を初期値として、5 つのスケールを勾配降下法で当てはめよ。学習されたスケールを各座標の関連度と比較し、さらにテスト誤差を等方的カーネルおよび第 1 座標のみを最良のバンド幅で用いるオラクルのテスト誤差と比較せよ。
   3. 学習された計量とオラクル計量の 1 つ抜き誤差をそれぞれのテスト誤差と比較し、同じ比較を $n = 1000$ でも繰り返せ。有限サンプルから類似度を学習することについて、この結果は何を示しているか。
   4. 一般の行列 $\mathbf{W} \in \mathbb{R}^{5\times 5}$ はクエリとキーを同一の線形変換で写すのに対し、学習されるアテンションは別々の行列 $\mathbf{W}_q$ と $\mathbf{W}_k$ でそれらを写す。共有の変換が保証し、別々の変換では失われるスコア a の性質を 2 つ挙げよ。

10.2 アテンションのスコアリングとマスキング

10.1.2 節では、アテンション重みは入力空間におけるユークリッド距離上の固定されたガウスカーネルによって決定されていた。しかし系列モデルでは、類似度は学習されなければならない。たとえば主語を探す動詞が必要とするのは、その統語的役割を果たす単語であって、生の埋め込み空間で最も近い単語ではないからである。ソフトな検索を訓練可能なネットワーク層へと変えるにあたっては、2 つの疑問が生じる。すなわち、その層はどのアテンションスコア関数 $a(\mathbf{q}, \mathbf{k})$ を計算すべきか、そしてどのクエリ・キーのペアを検索から除外すべきかである。

$n = 1{,}024$ トークンを読み取るヘッドは 1 系列あたり $n^2 = 1{,}048{,}576$ 個のクエリ・キーのペアを評価するため、スコアは低コストで計算でき、行列積のハードウェアへ無理なくマッピングできるものでなければならない。さらに、可変長のミニバッチに含まれるパディングトークンや、自己回帰言語モデルにおける未来のトークンのような無効なペアは、正規化の前に重みがゼロになるよう除外しなければならない。

本節では、**スケール化ドット積アテンション**（scaled dot-product attention、$\mathbf{q}^\top\mathbf{k}/\sqrt{d}$）を導出し、$1/\sqrt{d}$ 係数が初期化時におけるソフトマックスの飽和をどのように防ぐかを説明する。続いてブールマスキングとマスクの合成を実装し、学習されたアテンションが出力の教師信号のみから系列のアライメントを復元する様子を示す。

10.2.1 ドット積アテンション

10.1.2 節の単位帯域幅を持つガウスカーネルについて負の二乗距離を展開すると、内積と 2 つの二乗ノルムの和が現れる。

$$a(\mathbf{q}, \mathbf{k}_i) = -\frac{1}{2}\|\mathbf{q} - \mathbf{k}_i\|^2 = \mathbf{q}^\top \mathbf{k}_i - \frac{1}{2}\|\mathbf{k}_i\|^2 - \frac{1}{2}\|\mathbf{q}\|^2. \tag{10.9}$$

$-\frac{1}{2}\|\mathbf{q}\|^2$ はすべてのキー i にわたって定数であるため、ソフトマックス正規化によって相殺される。すべてのキーが等しいノルムを持つならば、$-\frac{1}{2}\|\mathbf{k}_i\|^2$ も同様に相殺される。キーのノルムが異なる場合であってもこれら両方のノルム項を落とすと、**ドット積スコア**（dot-product score）$\mathbf{q}^\top \mathbf{k}_i$ が得られる。大きな $\|\mathbf{k}_i\|$ にペナルティを課すガウスカーネルとは異なり、ドット積ではキーのノルムが乗法的な事前分布として働くため、長いキーほど同じ方向を向くあらゆるクエリからより大きな重みを引き寄せる（演習問題 1c）。

これに対し、初期のニューラル機械翻訳モデルでは、**加法アテンション**（additive attention）として知られる隠れ層 1 層の MLP が用いられた (Bahdanau et al. 2015)。

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R}, \tag{10.10}$$

ここで、$\mathbf{W}_q \in \mathbb{R}^{r \times d_q}$ と $\mathbf{W}_k \in \mathbb{R}^{r \times d_k}$ は学習される行列、$\mathbf{w}_v \in \mathbb{R}^r$ は重みベクトルである。加法スコアリングとドット積スコアリングの漸近的な演算回数は同程度であるが（$O(nmr)$ 対 $O(nmd)$。ただし n はクエリ数、m はキー数）、加法アテンションでは隠れ層の活性化値からなる $n \times m \times r$ のテンソルをメモリ上に実体化し、要素ごとに tanh を適用しなければならない。これに対し、ドット積アテンションは $n \times m$ 個のスコアすべてを 1 回の行列積 $\mathbf{Q}\mathbf{K}^\top$ で計算する。双線形スコアは MLP より一般性では劣るものの、クエリとキー自体が学習された射影 $(\mathbf{W}_q\mathbf{q})^\top(\mathbf{W}_k\mathbf{k})$ によって生成される場合には、その一般性の低下は実用上何ら問題にならない（演習問題 6 も参照）。

もう 1 つ重要なスケーリング係数が残っている。$\mathbf{q}, \mathbf{k} \in \mathbb{R}^d$ の各成分が、平均 0、分散 1 の独立な確率変数であると仮定しよう。このとき各項 $q_j k_j$ は平均 0、分散 1 となるため、その和 $\mathbf{q}^\top \mathbf{k}$ は平均 0、分散 d となる（標準偏差は 8 となる。ただし $d = 64$ の場合）。大きなドット積によってソフトマックスが勾配消失（vanishing gradients）の領域へ押し込まれるのを大きな d において防ぐため、Vaswani et al. (2017) は $\sqrt{d}$ で割って単位分散を回復させ、次の**スケール化ドット積**スコアを与えた。

$$a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^\top \mathbf{k}_i / \sqrt{d}. \tag{10.11}$$

ここで $1/\sqrt{d}$ は、10.1.2 節における逆温度 $1/\sigma^2$ の役割を果たしており、ベクトルの幅 d に応じて自動的に較正される。続いてソフトマックス正規化により、アテンション重みは次式で与えられる。

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(\mathbf{q}^\top \mathbf{k}_i / \sqrt{d})}{\sum_{j=1}^{m} \exp(\mathbf{q}^\top \mathbf{k}_j / \sqrt{d})}, \tag{10.12}$$

これにより、出力 $\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\, \mathbf{v}_i$ が得られる。

10.2.1.1 計算例

$d = 4$（したがって $1/\sqrt{d} = 1/2$）とし、2 つのクエリと 3 組のキー・値のペアを次のように定める。

$$\begin{aligned} \mathbf{q}_1 &= (2,2,0,0), & \mathbf{k}_1 &= (1,1,0,0), & \mathbf{v}_1 &= (1,0), \\ \mathbf{q}_2 &= (0,0,2,0), & \mathbf{k}_2 &= (0,0,1,1), & \mathbf{v}_2 &= (0,1), \\ & & \mathbf{k}_3 &= (0,0,1,-1), & \mathbf{v}_3 &= (1,1). \end{aligned}$$

1. **スコア：** $\mathbf{q}_1$ と 3 つのキーとの生のドット積は $(4,0,0)$ であり、$\mathbf{q}_2$ では $(0,2,2)$ となる。
2. **スケーリング：** $\sqrt{4} = 2$ で割ると、$(2,0,0)$ と $(0,1,1)$ が得られる。
3. **ソフトマックス：** $\mathbf{q}_1$ は重み $(e^2/(e^2+2), 1/(e^2+2), 1/(e^2+2)) \approx (0.787, 0.107, 0.107)$ を割り当て、$\mathbf{q}_2$ は $(1/(1+2e), e/(1+2e), e/(1+2e)) \approx (0.155, 0.422, 0.422)$ を割り当てる。
4. **重み付き和：** $\mathbf{o}_1 = 0.787\,\mathbf{v}_1 + 0.107\,\mathbf{v}_2 + 0.107\,\mathbf{v}_3 \approx (0.893, 0.213)$ および $\mathbf{o}_2 \approx (0.578, 0.845)$ となる。

直交するキーであっても非ゼロのソフトマックス重み（0.107）を受け取り、クエリの非ゼロ成分上で同点となるキー（$\mathbf{k}_2$ と $\mathbf{k}_3$、$\mathbf{q}_2$ のもとで）は残りの重みを等分する。もしスケーリング係数 $1/2$ を省くと、$\mathbf{q}_1$ が $\mathbf{k}_1$ に与える重みは 0.787 から $e^4/(e^4+2) \approx 0.96$ へと鋭くなる。

10.2.1.2 スコアの分散と $1/\sqrt{d}$ 係数

1 つのスコアが他のすべてのスコアを大きなマージンで上回るとき、重みベクトル α はワンホットベクトルに近づき、ソフトマックスは**飽和**（saturate）する。$\alpha = \mathrm{softmax}(\mathbf{a})$ のヤコビ行列（Jacobian matrix）は次式で与えられる。

$$\frac{\partial \alpha}{\partial \mathbf{a}} = \mathrm{diag}(\alpha) - \alpha\alpha^{\top}, \tag{10.13}$$

この行列は、α がワンホットになるにつれてゼロ行列へと収束する（演習問題 2）。飽和した行では、スコアに関して逆伝播される勾配が消失するため、選択されなかったキーが訓練中に重みを獲得することは困難になる。

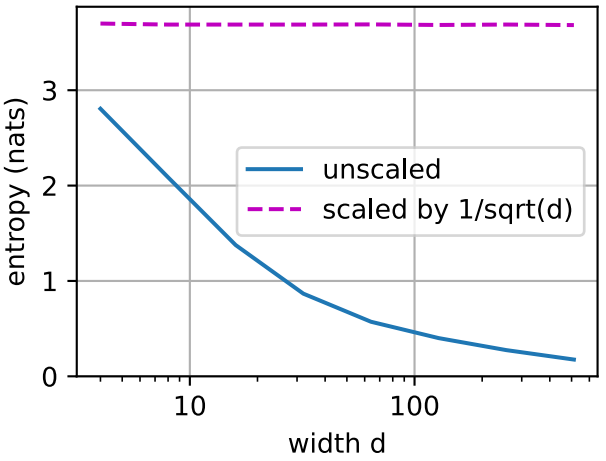
標準正規分布からランダムに生成したクエリ（2,048 個のクエリがそれぞれ 64 個のランダムなキーにアテンションを向ける設定）を用いて、次元 $d \in \{4, \ldots, 512\}$ にわたり、スケーリングなしのスコアと $1/\sqrt{d}$ でスケーリングしたスコアを比較することでこの効果を測定できる。64 個のキーに対する一様な重みのエントロピーは $\log 64 \approx 4.16$ nat であるのに対し、ワンホット分布のエントロピーは 0 である。

```
def softmax_weights(d, scale, num_keys=64, num_queries=2048):
    """Log-weights of random queries over random keys, scores scaled."""
    q = jax.random.normal(d2l.get_key(), (num_queries, d))
    keys = jax.random.normal(d2l.get_key(), (num_queries, num_keys, d))
    scores = (keys @ q[..., None]).squeeze(-1) * scale
    return jax.nn.log_softmax(scores, axis=-1)

def entropy(log_alpha):
    return float(-(jnp.exp(log_alpha) * log_alpha).sum(-1).mean())
```

```
def largest_weight(log_alpha):
    return float(jnp.exp(log_alpha).max(-1).mean())

def jacobian_norm(log_alpha):
    """Mean Frobenius norm of the Jacobian diag(alpha) - alpha alpha^T."""
    alpha = jnp.exp(log_alpha)
    J = (alpha[..., None] * jnp.eye(alpha.shape[-1])
         - alpha[..., :, None] * alpha[..., None, :])
    return float(jnp.linalg.norm(J, axis=(-2, -1)).mean())
```

ヘッド幅 d を対数軸上で 4 から 512 まで変化させると、スケーリングなしのドット積アテンションでは d の増大に伴って急激なエントロピー崩壊が生じる一方、$1/\sqrt{d}$ でスケーリングするとすべての幅にわたってアテンションのエントロピーが平坦に保たれることがわかる。

```
ds = [2**k for k in range(2, 10)]
entropies = [[entropy(softmax_weights(d, 1.0)) for d in ds],
             [entropy(softmax_weights(d, 1 / math.sqrt(d))) for d in ds]]
d2l.plot(ds, entropies, 'width d', 'entropy (nats)',
         legend=['unscaled', 'scaled by 1/sqrt(d)'], xscale='log')
```



下の表は、5 通りの幅におけるエントロピー、最大重みの平均、およびソフトマックスのヤコビ行列のフロベニウスノルム（Frobenius norm）の平均を示したものである。

```
print('      entropy (nats)    largest weight     Jacobian norm')
print('  d' + '   unscaled scaled' * 3)
for d in (4, 16, 64, 128, 512):
    stats = [(entropy(la), largest_weight(la), jacobian_norm(la))
             for la in (softmax_weights(d, 1.0),
                        softmax_weights(d, 1 / math.sqrt(d)))]
    print(f'{d:3d}' + ''.join(f'   {u:8.3f} {s:6.3f}'
                              for u, s in zip(*stats)))
```

```
    entropy (nats)    largest weight     Jacobian norm
d   unscaled scaled   unscaled scaled   unscaled scaled
```

4	2.819	3.706	0.273	0.104	0.257	0.180
16	1.367	3.682	0.583	0.109	0.293	0.183
64	0.598	3.686	0.786	0.106	0.214	0.183
128	0.383	3.687	0.855	0.106	0.163	0.183
512	0.180	3.685	0.928	0.108	0.091	0.184

$1/\sqrt{d}$ によるスケーリングを行うと、エントロピーはすべての幅にわたって $\log 64 - \frac{1}{2} \approx 3.66$ nat 付近に保たれ（演習問題 5）、最大重みは ≈ 0.11、ヤコビ行列のノルムは ≈ 0.18 となる。スケーリングがない場合、スコアの標準偏差は $\sqrt{d}$ に比例して増大する。$d = 512$ に達する頃にはエントロピーは ≈ 0.2 nat まで崩壊し、ランダム初期化の時点で最大のキーが重みの 90% 超を占め、ヤコビ行列のノルムは半減する。$1/\sqrt{d}$ によるスケーリングは、初期状態のソフトマックスの鋭さをヘッド幅から切り離しつつ、訓練中には学習された射影が重みを自由に鋭くできるようにする（クエリとキーが整列するようになるとドット積は $O(d)$ で増大するため、最大更新パラメータ化（maximal update parametrization）における $1/d$ スケーリングの動機となる。9.9 節を参照）。近年の多くのモデルでは、温度の調整を射影に一切委ねず、クエリとキーを単位長に正規化したうえでスコアのスケールを独立したパラメータとして学習する。これは **QK-norm** と呼ばれる設計であり、11.1 節で改めて扱う。

このスケーリングは幅 d に対しては鋭さを一定に保つが、キーの数 m に対しては一定に保たない。スコアが単位分散を持つとき、エントロピーは約 $\log m - \frac{1}{2}$ となるため、重みはおよそ $0.6\,m$ 個のキーに分散する。つまりコンテキストがどれほど長くなっても、その一定の割合に広がる。検索も同様に希釈される。一致するキーが他の $m-1$ 個の各キーに対してスコア差 g だけリードしている場合、そのキーが受け取る重みは $1/(1+(m-1)e^{-g})$ である。$g = 9$ のスコア差は、約 0.9 の重み（キー数が 1,000 個のとき）を与えるが、0.1 未満（キー数が 100,000 個のとき）にまで落ち込む。鋭さを維持するには、スコア差が $\log m$ のオーダーで増大しなければならない。したがって、あるコンテキスト長で訓練されたモデルは、より長いコンテキスト長ではより拡散したアテンションを向けることになり、これがコンテキスト長に応じてスコアのスケールを引き上げるモデルが存在する理由である（10.8 節）。

10.2.2 マスキング

特定のクエリ・キーのペアをアテンションから除外する代表的な条件が 2 つある。

1. **パディング：**長さの異なる系列を `<blank>` トークンでパディングしてミニバッチを構成するとき、どのクエリもパディング位置にアテンションを向けてはならない。
2. **因果性：**自己回帰型の言語モデリングでは、位置 t のクエリは位置 $j \le t$ のキーにのみアテンションを向けることができる。

これら両方の制約は、ブール型の**マスク**テンソル（`True` の位置でクエリ i がキー j にアテンションを向けることを許可する）によって表される。

10.2.2.1 マスク付きソフトマックス演算

密なテンソルの形状を保つため、**マスク付きソフトマックス**（masked softmax）は除外されたスコア（`~mask`）を大きな負の数に置き換えてから `softmax` を呼び出す。この埋め値の選択には、浮動小数点形式（floating-point format）に応じた注意が必要である。

- -10^6 のようなハードコードされた定数は、float16（表現可能な最大の有限絶対値は 65,504）でオーバーフローする。
- リテラルの $-\infty$ は、クエリ行の全体がマスクされると必ず `NaN` を生じる。数値的に安定なソフトマックスは行の最大値 $-\infty$ を引くため、$-\infty - (-\infty) = \text{NaN}$ となるからである。
- テンソルのデータ型における最小の**有限**値（`finfo(dtype).min`）を用いれば、float16、bfloat16、float32 のいずれにおいても `NaN` を生じることなく、除外された指数項を厳密

なゼロへとアンダーフローさせることができる。すべてのキーがマスクされたクエリ行（`~mask.any(-1)`）に対しては、`masked_softmax` が結果の重みを明示的にゼロにする。

```
def masked_softmax(X, mask=None):
    """Softmax over the last axis with zero weight where mask is False."""
    if mask is None:
        return jax.nn.softmax(X, axis=-1)
    # Most negative finite value: zero weight at any precision, no NaN
    X = jnp.where(mask, X, jnp.finfo(X.dtype).min)
    weights = jax.nn.softmax(X, axis=-1)
    # A query without any valid key gets zero weights, not a uniform row
    return jnp.where(mask.any(-1, keepdims=True), weights, 0.0)
```

`mask` は `X` に対してブロードキャスト可能でありさえすればよいため、系列内のすべてのクエリで共有されるパディングマスクは形状 `(batch, 1, keys)` で保持できる。

```
def padding_mask(valid_lens, num_keys):
    """Boolean mask that is True for the first valid_lens keys."""
    # valid_lens of shape (batch,) gives (batch, 1, keys); per-query
    # lengths of shape (batch, queries) give (batch, queries, keys)
    if valid_lens.ndim == 1:
        valid_lens = valid_lens[:, None]
    return jnp.arange(num_keys) < valid_lens[..., None]
```

長さ 4 で有効長が 2 と 3 であるスコアゼロの 2 系列からなるバッチに対し、`masked_softmax` は各行の有効なプレフィックス上に一様に重みを分配する。

```
masked_softmax(jnp.zeros((2, 2, 4)), padding_mask(jnp.array([2, 3]), 4))
```

```
Array([[[0.5       , 0.5       , 0.        , 0.        ],
        [0.5       , 0.5       , 0.        , 0.        ]],

       [[0.33333334, 0.33333334, 0.33333334, 0.        ],
        [0.33333334, 0.33333334, 0.33333334, 0.        ]]], dtype=float32)
```

10.2.2.2 因果マスキング

因果マスク（causal mask）は、クエリ i がキー j にアテンションを向けることを $j \leq i$ の場合に限り許可し、$n(n-1)/2$ 個を全 n^2 ペアのうちから除外する下三角行列を形成する（除外数は $523{,}776$ ペアとなる、ただし $n = 1{,}024$ の場合）。`num_queries < num_keys` のとき（キャッシュを用いた生成時など）には、`causal_mask` はクエリを系列の末尾の位置に揃える。

```
def causal_mask(num_queries, num_keys=None):
    """Boolean (queries, keys) mask, True where a query may see a key."""
    # Queries are the last positions, as when new tokens attend to a
    # cache: query i sees key j if j <= i + num_keys - num_queries
    num_keys = num_queries if num_keys is None else num_keys
    i = jnp.arange(num_queries)[:, None]
    j = jnp.arange(num_keys)[None, :]
    return j <= i + (num_keys - num_queries)
```

10.2.2.3 マスクの合成

複数の制約が同時に適用されるとき、あるペアが有効となるのはすべての制約がそのペアを許可する場合に限られるため、ブールマスクはブロードキャストのもとでの論理積（logical AND、&）によって合成される（Figure 10.2）。

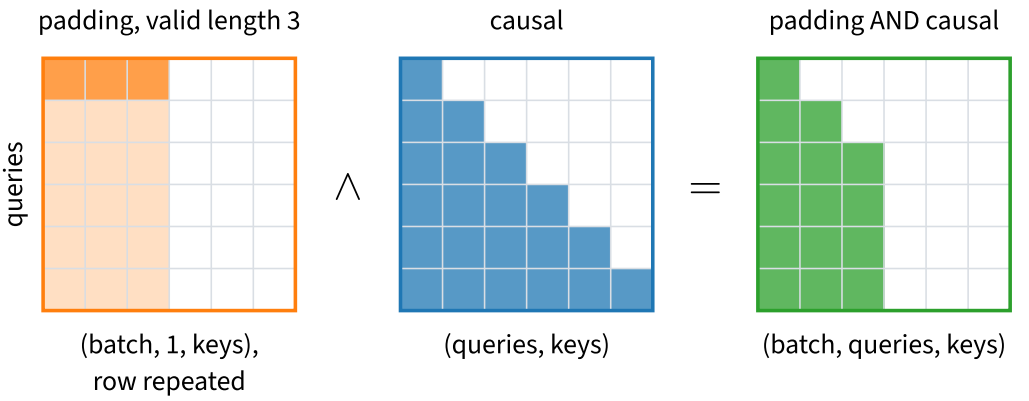
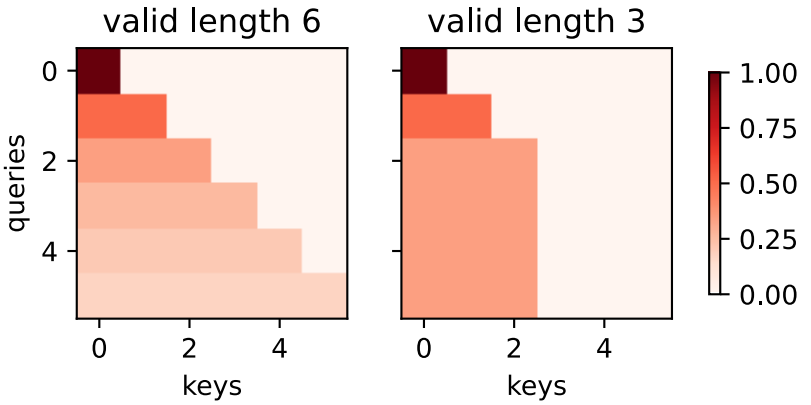


Figure 10.2: マスクは論理積で合成される。長さ 6、有効長 3 の系列に対し、パディングマスクはキーの 1 行分を保持し、ブロードキャストによってすべてのクエリに繰り返される（薄色）。因果マスクは下三角であり、両者の論理積は双方が許可するペアのみを許可する。行はクエリ、列はキーであり、塗りつぶされた要素が許可されたペアである。

等しいスコアに対してパディングマスク（有効長 6 と 3）と因果マスクを組み合わせると、1 番目の系列では完全な因果下三角が得られ、2 番目の系列ではすべての行がキー 3 で打ち切られる。

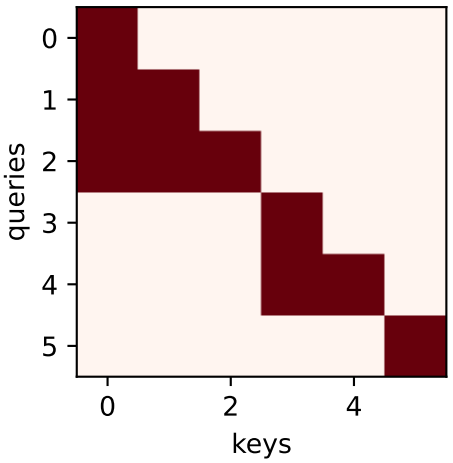
```
n = 6
padding = padding_mask(jnp.array([6, 3]), n)  # (batch, 1, key)
causal = causal_mask(n)                       # (query, key)
mask = padding & causal                       # (batch, query, key)
weights = masked_softmax(jnp.zeros((2, n, n)), mask)  # Equal scores
d2l.show_heatmaps(weights[None], xlabel='keys', ylabel='queries',
                  titles=['valid length 6', 'valid length 3'],
                  figsize=(5, 2.5))
```



短い方の系列では、クエリ 4 から 6 自体がパディングであるが、因果下三角は常にキー 1 まで遡るため、いずれのクエリも依然として有効なキーを持つ。これは論理積の性質ではなく、マスクの形状による性質である。もしパディングマスクを、各クエリが直近の少数のキーしか参照できない**スライディングウィンドウ**と組み合わせたならば、ウィンドウ全体がパディング領域内に収まるクエリは有効なキーを 1 つも持たなくなるだろう。単独では無害なマスクであっても、組み合わせると行が空になることがあり、`masked_softmax` がゼロにするのはまさにこのケースである。

同じブール代数は、パディングのオーバーヘッドを避けるために複数の短い文書を 1 行に連結する**パック化系列**（packed sequences）も扱える。同一文書を表す指示行列 `doc[:, None] == doc[None, :]` と `causal_mask` の論理積をとると、ブロック対角の因果マスクが得られる。

```
doc = jnp.array([0, 0, 0, 1, 1, 2])  # Document of each position
same_doc = doc[:, None] == doc[None, :]
packed = same_doc & causal_mask(6)
d2l.show_heatmaps(packed.astype(jnp.float32)[None, None], xlabel='keys',
                  ylabel='queries', cbar=False)
```



10.2.3 ミニバッチに対するアテンション

n 個のクエリを $\mathbf{Q} \in \mathbb{R}^{n \times d}$ に、m 個のキーを $\mathbf{K} \in \mathbb{R}^{m \times d}$ に、そして m 個の値を $\mathbf{V} \in \mathbb{R}^{m \times d_v}$ に積み重ねると、スケール化ドット積アテンションは 2 回の行列積と行ごとのソフトマックスとして表される（Figure 10.3）。

$$\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \in \mathbb{R}^{n \times d_v}. \tag{10.14}$$

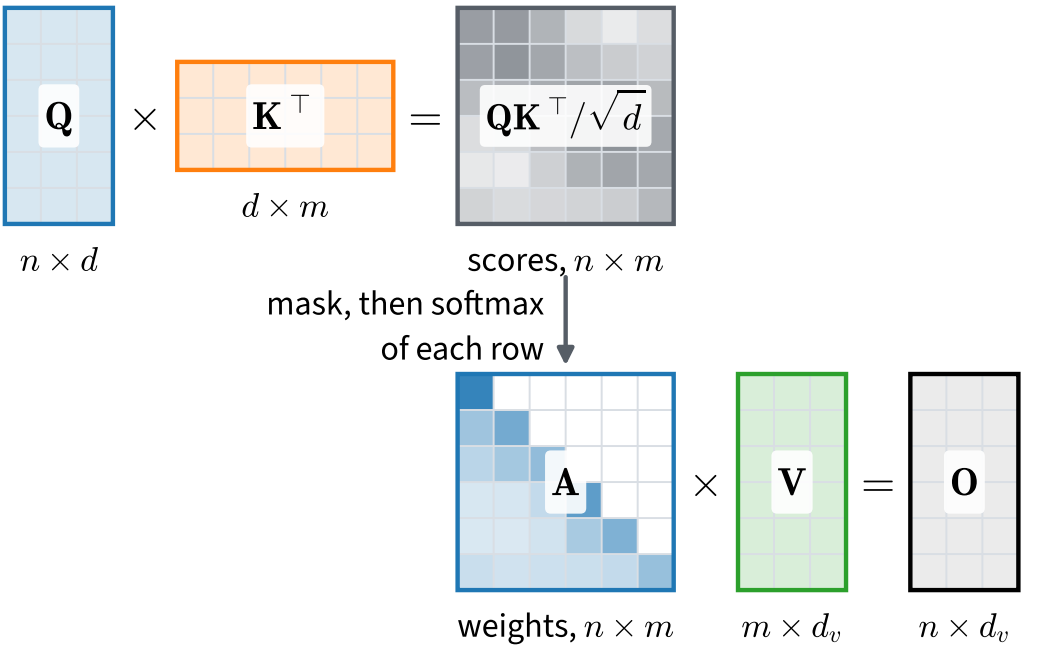


Figure 10.3: スケール化ドット積アテンションにおける形状。$\mathbf{Q}\mathbf{K}^\top/\sqrt{d}$ は $n \times m$ 個すべてのクエリ・キーのペアのスコアを保持する。マスキングと各行にわたるソフトマックスによってこれらが重み $\mathbf{A}$ へと変換され（ここでは上三角を空にする因果マスクを使用）、さらに $\mathbf{V}$ を掛けることでクエリごとに 1 つの値の重み付き和が得られる。

先頭にバッチ軸（およびヘッド軸）を持つバッチ入力に対しては、バッチ行列積 `@` が最後の 2 次元に対して演算し、先頭の各軸を自動的にブロードキャストする。GPT-2 small の 1 つのヘッド（$n = m = 1{,}024$、$d = d_v = 64$）では、スコア行列は 1,048,576 個の浮動小数点数（fp32 で 4 MiB）を保持し、2 回の積には $4n^2 d \approx 0.27$ GFLOPs を要する。いずれも n に関して二次のオーダーである。

10.2.3.1 DotProductAttention クラス

`DotProductAttention` は、任意のマスキングとアテンション重みへのドロップアウト（デフォルトは 0.0）を備えて Equation 10.14 を実装し、確認用に直近のアテンション重みを `attention_weights` に保存する。

```
class DotProductAttention(nnx.Module):
    """Scaled dot-product attention over the last two axes."""
    def __init__(self, dropout=0.0, *, rngs=None):
        # rngs is needed only to draw dropout masks (dropout > 0)
        self.dropout = nnx.Dropout(dropout, rngs=rngs)
        # A Variable, so that views of the module (nnx.view) share it
        self.last_weights = nnx.Intermediate(None)

    # Leading axes (batch, and heads if present) are carried along.
    # Shape of queries: (..., no. of queries, d)
    # Shape of keys: (..., no. of key-value pairs, d)
    # Shape of values: (..., no. of key-value pairs, value dimension)
    # mask: boolean, broadcastable to (..., no. of queries, no. of keys),
    # True where a query may attend to a key
    def __call__(self, queries, keys, values, mask=None):
        d = queries.shape[-1]
        scores = queries @ keys.swapaxes(-2, -1) / math.sqrt(d)
        weights = masked_softmax(scores, mask)
        if not isinstance(weights, jax.core.Tracer):
            # Record the weights of eager calls for inspection; inside
            # jit-compiled code skip it, so the module state keeps a
            # fixed structure and the compiled step is reused
            self.last_weights.set_value(weights)
        return self.dropout(weights) @ values

    @property
    def attention_weights(self):
        """Weights of the last eager call: (..., queries, keys)."""
        return self.last_weights.get_value()
```

DotProductAttention を 10.2.1.1 節のベクトルに対して評価すると、先ほどの手計算が再現される。

```
queries = d2l.tensor([[[2.0, 2, 0, 0], [0, 0, 2, 0]]])           # (1, 2, 4)
keys = d2l.tensor([[[1.0, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, -1]]])  # (1, 3, 4)
values = d2l.tensor([[[1.0, 0], [0, 1], [1, 1]]])                  # (1, 3, 2)
attention = DotProductAttention()
output = attention(queries, keys, values)
print('weights:', np.asarray(attention.attention_weights[0]).round(3))
print('output: ', np.asarray(output[0]).round(3))
```

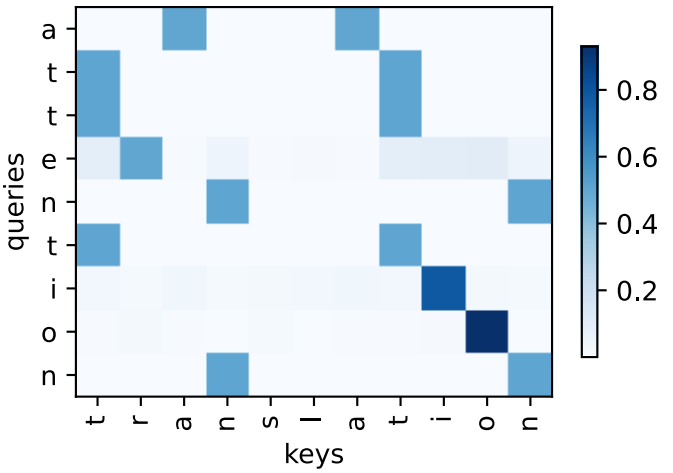
```
weights: [[0.787 0.107 0.107]
 [0.155 0.422 0.422]]
output:  [[0.893 0.213]
 [0.578 0.845]]
```

10.2.3.2 2つの単語の文字マッチング

射影を伴わない内容マッチングが何を識別でき、何を識別できないかを確かめるため、"attention" と "translation" に現れる異なる各文字に標準正規分布からランダムに生成した埋め込み $\mathbf{e} \in \mathbb{R}^{32}$ を割り当て、"attention" の各文字をクエリとして "translation" の各文字を検索させてみよう。同一の文字どうしではスケール化スコアの期待値が $\mathbb{E}[\|\mathbf{e}\|^2]/\sqrt{32} = \sqrt{32} \approx 5.7$ となる一

方、異なる文字どうしではスコアの期待値が 0、分散が単位分散となる。

```
words = 'attention', 'translation'
letters = sorted(set(''.join(words)))
emb = jax.random.normal(d2l.get_key(), (len(letters), 32))
queries, keys = (emb[jnp.array([letters.index(c) for c in w])][None]
                 for w in words)
attention = DotProductAttention()
attention(queries, keys, keys)
d2l.show_heatmaps(attention.attention_weights[None], xlabel='keys',
                  ylabel='queries', figsize=(4, 3.5), cmap='Blues',
                  xticklabels=words[1], yticklabels=words[0])
```



ここから 3 つの挙動が際立って見て取れる。

1. 一意に一致する文字（i、o）は、ほぼすべての重みを対応する相手の文字に集中させる。
2. 複数回一致する文字（a、t、n）は、"translation" 内に現れるその文字のすべての出現箇所に重みを均等に分割する。内容のみからなるキーは位置の識別情報を持たないためである。
3. 一致する相手のないクエリ（e）は、ランダムなキー全体に確率質量を分散させる。関連するキーが存在しなくてもソフトマックスの重みの総和は 1 にならなければならないからである（訓練済みの言語モデルでは、先頭トークンがこのような行き場のない質量を**アテンションシンク**として吸収することが多い (Xiao et al. 2024)）。

10.2.4 学習されたアライメントとしてのアテンション

学習可能なアテンションは、ニューラル機械翻訳 (Bahdanau et al. 2015) において、系列対系列 RNN（12.2 節）が抱えていた固定長ベクトルのボトルネックを解消するために考案された。各デコーディングステップでデコーダがすべてのエンコーダ状態にクエリを投げると、学習されたアテンション重みは原言語と目的言語の間の単語アライメントを自発的に復元した。それは単調な対角線を基本としつつ、語順が異なる箇所（たとえば英語の「**black cat**」に対するフランス語の「**chat noir**」）では非対角の交差を伴うものであった。

最小限のモデル `ToyAligner` を用いて、アライメントが創発する様子を観察できる。入力系列は 8 個のトークン（サイズ 16 の語彙から抽出）からなり、目標系列は固定された置換 π（位置 3 と 4 を入れ替える）を適用して得られる。各出力位置 j は学習されるクエリベクトルを持ち、入力側

の各キーは連結されたワンホットトークンとワンホット入力位置の線形射影であり、値は入力トークンのワンホットベクトルである。

```
class ToyAligner(nnx.Module):
    """One learned query per output position; keys from token and position."""
    def __init__(self, seq_len=8, vocab_size=16, d=16, *, rngs):
        self.vocab_size, n_in = vocab_size, vocab_size + seq_len
        self.queries = nnx.Param(
            jax.random.normal(rngs.params(), (seq_len, d)))
        self.W_k = nnx.Param(jax.random.normal(rngs.params(), (n_in, d))
                             / math.sqrt(n_in))
        self.attention = DotProductAttention()

    def __call__(self, X):
        # X: (batch, seq_len) token ids; the values are their one-hot vectors
        tokens = jax.nn.one_hot(X, self.vocab_size)
        n = X.shape[1]
        positions = jnp.broadcast_to(jnp.eye(n), (len(X), n, n))
        keys = jnp.concatenate((tokens, positions), -1) @ self.W_k
        # The queries broadcast over the batch: (batch, seq_len, vocab_size)
        return self.attention(self.queries, keys, tokens)
```

並べ替えられた目標トークンの負の対数尤度（negative log-likelihood）を最小化することで、`ToyAligner` を Adam により 100 ステップ訓練する。

```
perm = jnp.array([0, 1, 2, 4, 3, 5, 6, 7])  # Output j copies source perm[j]

def align_data(key, n, seq_len=8, vocab_size=16):
    X = jax.random.randint(key, (n, seq_len), 0, vocab_size)
    return X, X[:, perm]

@nnx.jit
def align_step(model, optimizer, key):
    X, Y = align_data(key, 128)
    loss = lambda model: -jnp.log(
        jnp.take_along_axis(model(X), Y[..., None], -1)).mean()
    optimizer.update(model, nnx.grad(loss)(model))

def train_aligner(num_steps=100):
    model = ToyAligner(rngs=nnx.Rngs(d2l.get_key()))
    optimizer = nnx.Optimizer(model, optax.adam(0.05), wrt=nnx.Param)
    for key in jax.random.split(d2l.get_key(), num_steps):
        align_step(model, optimizer, key)
    return model
```

未訓練のモデルと 3 回の独立な訓練実行を 1,000 本のテスト系列上で比較すると、勾配降下法は確実に 100% の正解率へと収束し、出力位置にわたる平均で重みの 99% 超を $\pi(j)$ に割り当てることがわかる。

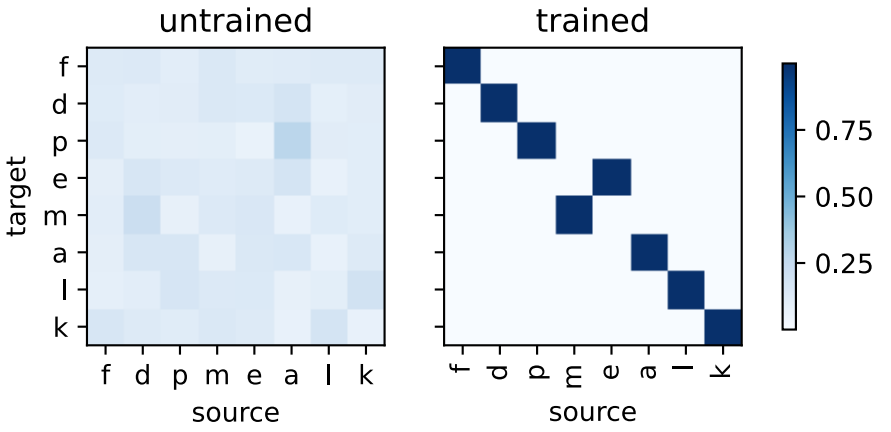
```
X, Y = align_data(d2l.get_key(), 1000)
```

```
def evaluate_aligner(model, X, Y, name):
    accuracy = (model(X).argmax(-1) == Y).mean()
    weights = model.attention.attention_weights  # (1000, 8, 8)
    largest = (weights.argmax(-1) == perm).mean()
    on_perm = weights[:, jnp.arange(8), perm].mean()
    print(f'{name:>9}: accuracy {accuracy:.3f}, largest weight on pi(j) '
          f'{largest:.3f}, mean weight on pi(j) {on_perm:.3f}')

untrained = ToyAligner(rngs=nnx.Rngs(d2l.get_key()))
evaluate_aligner(untrained, X, Y, 'untrained')
for run in range(1, 4):
    model = train_aligner()
    evaluate_aligner(model, X, Y, f'run {run}')
```

```
untrained: accuracy 0.214, largest weight on pi(j) 0.053, mean weight on pi(j) 0.120
    run 1: accuracy 1.000, largest weight on pi(j) 1.000, mean weight on pi(j) 0.999
    run 2: accuracy 1.000, largest weight on pi(j) 1.000, mean weight on pi(j) 1.000
    run 3: accuracy 1.000, largest weight on pi(j) 1.000, mean weight on pi(j) 1.000
```

最初のテスト系列に対する訓練前後のアテンション重み行列を可視化すると、ToyAligner が構造のない初期分布を π の厳密な置換行列（位置 3 と 4 に非対角の入れ替えを持つ）へと変化させていることが確認できる。

```
letters = 'abcdefghijklmnop'  # One letter per token id
maps = d2l.stack([untrained.attention.attention_weights[0],
                  model.attention.attention_weights[0]])
d2l.show_heatmaps(maps[None], xlabel='source', ylabel='target',
                  titles=['untrained', 'trained'], figsize=(5.5, 3),
                  cmap='Blues', xticklabels=[letters[int(i)] for i in X[0]],
                  yticklabels=[letters[int(i)] for i in Y[0]])
```



キーがトークンの識別情報と並んで位置を符号化しているため、入力系列内で同じトークンが繰り返される場合であっても、訓練後のアテンションヒートマップは π の厳密な置換行列を復元する。しかし一般の系列モデリングでは、クエリは系列自体から導出されなければならず、1 つの位置が複数の異なる値を同時に取り出す必要があるかもしれず（10.3 節）、さらに位置エンコーディング（positional encoding）は異なる系列長にわたって汎化しなければならない（10.4 節）。

10.2.5 まとめ

スケール化ドット積スコアリング $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k}/\sqrt{d}$ は、ガウスカーネルにおける二次のノルムペナルティを省き、すべてのペア間のスコアを 1 回の行列積 $\mathbf{Q}\mathbf{K}^\top$ で評価する。$\sqrt{d}$ で割ることにより、単位分散の入力に対してスコアの単位分散が保たれ、ヘッド幅 d が増大してもソフトマックスの飽和やスコア勾配の消失が防がれる。系列のパディングや因果デコーディングといった構造上の制約は、論理積のもとで合成されるブールマスクによって適用され、`finfo(dtype).min` と空行の明示的なゼロ化を併用することで `float16` でのオーバーフローや `NaN` の伝播を回避する。最後に、内容のみを持つキーは重複するトークン間でアテンションを均等に分割するのに対し、位置情報を備えたキーを用いれば系列のアライメントをエンドツーエンドで学習できる。

10.2.6 演習問題

1. **スコアの不変性**。ガウススコア Equation 10.9 とスケール化ドット積 Equation 10.11 は、いずれもソフトマックス Equation 10.12 を通じてアテンション重みを定義する。
   1. 2 つのスコアのそれぞれについて、次のどの変換が重みを変化させないかを求めよ。すべてのキーに 1 つのベクトル $\mathbf{c}$ を加える変換、クエリとすべてのキーに $\mathbf{c}$ を加える変換、クエリとすべてのキーに 1 つの直交行列を掛ける変換。
   2. クエリとキーは通常、入力からのアフィン写像 $\mathbf{q} = \mathbf{W}_q\mathbf{x} + \mathbf{b}_q$ および $\mathbf{k}_i = \mathbf{W}_k\mathbf{y}_i + \mathbf{b}_k$ によって計算される。$\mathbf{q}^\top \mathbf{k}_i$ を 4 つの項に展開し、重みが $\mathbf{b}_k$ に依存しないことを示せ。$\mathbf{b}_k$ に関する損失の勾配はどうなるか。
   3. $\mathbf{b}_q$ は 4 つの項のうちどれを通じて重みに影響を与えるか。また、バイアスなしのスコアでは表現できない、キーに対するどのような選好をそれによって表現できるか。
2. [code] **ソフトマックスのヤコビ行列**。`jacobian_norm` はヤコビ行列 Equation 10.13 のフロベニウスノルムを測定する。
   1. Equation 10.13 を Equation 10.12 から導出し、有限のスコアに対してその零空間が全成分 1 のベクトルによって張られることを示し、さらにランダムなスコアベクトル上で自動微分と比較してこの式を検証せよ。
   2. k 個のキー（全 m 個のうち）に一様に広がる重みに対してノルムを計算し、それを最大化する k を求め、ランダムな重みベクトルをサンプリングしてこの最大値を超えるものがないことを確認せよ。この結果を用いて、表のスケーリングなしのノルムが $d = 4$ から $d = 16$ にかけて上昇した後に低下する理由を説明せよ。
3. **完全にマスクされた行**。`masked_softmax` は除外されたスコアをそのデータ型（dtype）の最小の有限値で上書きし、続いて有効なキーを持たないクエリの行をゼロにする。
   1. すべてのキーが除外された 4 個のスコアからなる行を作成せよ。それを float16 の -10^6、float32 の $-\infty$、および float32 の `finfo(dtype).min` でそれぞれ埋め、`masked_softmax` の最後のゼロ化を行わずにフレームワークのソフトマックスをそれぞれに適用せよ。得られる結果を報告し、行の最大値を引く計算過程を追って各結果を説明せよ。
   2. クエリ・キー・値のすべてが同一の系列から得られる自己アテンション層において、10.2.2.3 節の合成マスクで起こり得るように、パディング位置にあるクエリが有効なキーを 1 つも持たないとし、除外されたスコアが $-\infty$ で上書きされてその位置の出力が NaN になったとする。その出力自体は破棄され、次のアテンション層ではすべての有効なクエリがそのパディング位置をキーから除外する。それにもかかわらず、次の層でその系列のすべての有効な位置の出力が NaN によって汚染される理由を説明せよ。
4. **合成マスク**。10.2.2.3 節ではマスクを論理積で合成し、単独では無害なマスクでも組み合わせると有効なキーを持たないクエリが生じ得ることを指摘した。幅 w の因果スライディングウィンドウは、位置 t のクエリが位置 $t - w + 1, \ldots, t$ のキーにアテンションを向けることを許可する。

1. 長さ 6、有効長 3 の系列に対し、パディングマスクと幅 2 の因果スライディングウィンドウを 6×6 のブール行列として書き下せ。それぞれのマスク単独ではすべてのクエリに少なくとも 1 つのキーが残ることを確認し、両者の論理積で行が空になるクエリを求めよ。
2. すべてのマスクが各クエリに自身の位置へのアテンションを許可するならば、それらのマスクの論理積には空の行が生じないことを示せ。パディングマスク、因果マスク、スライディングウィンドウマスク、パック化系列マスクのうち、この条件を満たすものはどれか。また、その答えは先ほど見つけた空の行について何を示唆するか。
3. 実装によっては、許可されたペアの要素を 0、除外されたペアの要素を $c < 0$ とするバイアス行列を加算し、バイアス同士の足し算によってマスクを合成するものがある。c として単精度における最小の有限値（約 -3.4×10^{38}）をとる。単精度においてこのようなバイアスを 2 つ足し合わせた和を計算し、すべてのキーが両方のマスクで除外されている行と、すべてのキーが除外されているが一部のキーは片方のマスクのみで除外されている行のそれぞれに対して、ソフトマックスが何を返すかを求めよ。

5. **ランダムなスコアのエントロピー。**●上でプロットしたスケーリングありのエントロピー曲線は 3.7 nat 付近にとどまり、一様分布の値 $\log 64 \approx 4.16$ を下回っている。1 つのクエリ $\mathbf{q} \in \mathbb{R}^d$ を固定し、`softmax_weights` と同様に独立な標準正規分布の成分を持つ m 個のキーを抽出せよ。
   1. スケール化スコア $a_i = \mathbf{q}^\top \mathbf{k}_i / \sqrt{d}$ が平均 0、分散 $s^2 = \|\mathbf{q}\|^2 / d$ の独立な正規確率変数であることを示し、s^2 の期待値を、$\mathbf{q}$ が `softmax_weights` と同様に抽出される場合について計算せよ。
   2. 重みのエントロピーは $H = \log Z - \sum_i \alpha_i a_i$（ただし $Z = \sum_j \exp(a_j)$）である。大きな m に対して Z/m と $\sum_i \exp(a_i)\, a_i / m$ をそれぞれの期待値で置き換え、次式を導出せよ。
   $$H \approx \log m - \frac{s^2}{2}.$$
   この近似式を $m = 64$ および $s = 1$ に対して評価せよ。
   3. この近似が成り立つには、Z がその平均の周りに集中する必要がある。Z の相対標準偏差を s と m の関数として計算し、そこからこの近似が成り立つための s^2 に関する条件を導出せよ。スケーリングなしのスコアでは $s^2 = \|\mathbf{q}\|^2$ であり、その期待値は d である。$m = 64$ に対し $s^2 = 4$ および $s^2 = 8$ における相対標準偏差を評価し、この近似値を上でプロットした $d = 4$ および $d = 8$ のスケーリングなしのエントロピーと比較せよ。
6. [code] **ゼロ近傍での加法スコア。**●加法スコア Equation 10.10 は、射影されたクエリと射影されたキーを tanh の内側で足し合わせる。
   1. tanh を恒等関数に置き換えると、アテンション重みがクエリに依存しなくなることを示せ。
   2. tanh をゼロの周りで 3 次まで展開し、重みがクエリに依存する最低次の項を特定し、さらに $\mathbf{W}_q$ と $\mathbf{W}_k$ の両方に小さな係数 ϵ を掛けたときにこの依存性がどのようにスケールするかを述べよ。
   3. 3 個のクエリと 6 個のキー（いずれも $\mathbb{R}^8$ 上で標準正規分布に従う成分を持つ）および隠れ層の幅 $r = 16$ に対して加法スコアを実装せよ。ここで $\mathbf{W}_q$、$\mathbf{W}_k$、$\mathbf{w}_v$ は、標準正規分布の成分をそれぞれの入力幅の平方根で割って抽出するものとする。$\mathbf{W}_q$ と $\mathbf{W}_k$ に $\epsilon \in \{1, 0.3, 0.1, 0.03\}$ を掛け、3 つのクエリの重み行間の最大差を多数回の抽出にわたって平均して測定し、導出したスケーリング則を検証せよ。これらのスケールのうち、重みがクエリに有意に依存するのはどれか。
7. [code] **アライメントの変種。**●10.2.4 節の `ToyAligner` は、出力位置ごとに学習される 1 つのクエリと、トークンおよび位置を保持するキーから置換 π を学習する。
   1. キーからワンホット位置を取り除き、トークンのみに依存するようにせよ。訓練の前

に、出力位置 j が依然として重みを $\pi(j)$ に置けるかどうか、またモデルがおおよそどの程度の正解率に到達できるかを予測し、実際に訓練して比較せよ。その結果を 10.2.3.2 節の文字マッチングと関連づけよ。

2. スコアを加法規則 Equation 10.10 に置き換え、各モデルが 99% の正解率に達するまでに要するステップ数を比較せよ。訓練の前に、演習問題 6 を用いて、射影を小さく初期化したときに加法モデルがどのような初期挙動を示すかを予測し、その予測を検証せよ。
3. クエリとキーに標準正規分布による初期化のもとで幅 $d \in \{16, 256, 1024\}$ を与え、係数 $1/\sqrt{d}$ がある場合とない場合で訓練せよ。訓練曲線を初期化時点における重みのエントロピーと関連づけよ。
4. 先行するトークンが語彙の指定された部分集合に属するときにのみ隣接する 2 要素を入れ替えるといった、トークンに依存する並べ替えを `ToyAligner` では学習できないことを示せ（ヒント：このモデルのスコアはすべての位置で共有されるトークン項と位置項の和である。隣接する 2 要素のスコアを比較せよ）。これを可能にするには、クエリまたはキーが何に依存していなければならないかを述べよ。

10.3 自己アテンションと複数のヘッド

10.2 節の例では、クエリはある系列から得られ、キーと値は別の系列から得られていた。しかし、デコーダのみの言語モデルが持つ系列は 1 つだけである。各トークンを文脈化するには、クエリ・キー・値のすべてをその系列自体から導出しなければならない。

「**Alice gave Bob the book**」という文を考えよう。**gave** を解釈するには、与え手（**Alice**）、受け手（**Bob**）、対象物（**book**）を特定する必要がある。**自己アテンション**（self-attention）では (Cheng et al. 2016; Lin et al. 2017; Vaswani et al. 2017)、すべてのトークンが自身の表現をクエリ・キー・値へと射影するため、**gave** は文中の他の単語にクエリを送り、それらの値を集約できる。

しかし、1 回のアテンション検索が生成する確率分布はクエリあたり **1 つ**だけであり、返されるのも混合された単一のベクトルにとどまる。トークンが複数の異なる項を 1 つの平均に潰してしまうことなく取り出すには、どうすればよいだろうか。本節ではまず、この 5 語の文に対する自己アテンションの単一ヘッドの動作をたどり、次に 1 つのヘッドでは独立な 2 つの値を復元できない理由を示す情報理論的な下界を証明し、最後に**マルチヘッドアテンション**と**グループ化クエリアテンション**（grouped-query attention、GQA）を構成する。

10.3.1 自己アテンション

自己アテンションでは、各トークン表現 $\mathbf{x}_i \in \mathbb{R}^d$（$\mathbf{X} \in \mathbb{R}^{n\times d}$ の第 i 行）が、学習された線形射影 $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{d\times d_h}$ および $\mathbf{W}_v \in \mathbb{R}^{d\times d_v}$ を介して 3 つの異なる役割を果たす。すなわち、**クエリ** $\mathbf{x}_i\mathbf{W}_q$ はそのトークンが求めるものを指定し、**キー** $\mathbf{x}_i\mathbf{W}_k$ は他のクエリに対して提示するものを指定し、**値** $\mathbf{x}_i\mathbf{W}_v$ は選択されたときに返される内容を指定する。これにスケール化ドット積アテンション（scaled dot-product attention）を適用すると、幅 d_h の 1 つのアテンション**ヘッド**が得られる。

$$\operatorname{softmax}\left(\frac{(\mathbf{X}\mathbf{W}_q)(\mathbf{X}\mathbf{W}_k)^\top}{\sqrt{d_h}}\right)\mathbf{X}\mathbf{W}_v \in \mathbb{R}^{n\times d_v}. \tag{10.15}$$

10.3.1.1 1 つの文に対する単一ヘッド

Equation 10.15 を具体的にたどるため、「**Alice gave Bob the book**」の各単語を、4 次元の one-hot な文法的役割（人物、動作、物、機能語）と 5 次元の one-hot な単語識別子を連結した 9 次元ベクトルで表す。$\mathbf{W}_q$ と $\mathbf{W}_k$ は 4 次元の役割のみを $d_h = 2$ 次元に読み出し、$\mathbf{W}_v$ は 5 次元の単語指示ベクトルを抽出するように設定する。

役割	クエリ	キー
人物（Alice、Bob）	$(1, -1)$	$(1, 1)$
動作（**gave**）	$(1, 1)$	$(1, -1)$
物（book）	$(1, -1)$	$(1, 1)$
機能語（「the」）	$(0, 0)$	$(0, 0)$

```
tokens = ['Alice', 'gave', 'Bob', 'the', 'book']
# Role of each token: person, action, thing, function word
roles = d2l.tensor([[1.0, 0, 0, 0], [0, 1, 0, 0], [1, 0, 0, 0],
                    [0, 0, 0, 1], [0, 0, 1, 0]])
X = d2l.concat((roles, d2l.eye(5)), 1)  # (5, 9): the role, then the word
# Queries and keys read the role part (rows: person, action, thing,
# function word), values the word part
W_q = d2l.concat((d2l.tensor([[1.0, -1], [1, 1], [1, -1], [0, 0]]),
                  d2l.zeros((5, 2))))
W_k = d2l.concat((d2l.tensor([[1.0, 1], [1, -1], [1, 1], [0, 0]]),
                  d2l.zeros((5, 2))))
W_v = d2l.concat((d2l.zeros((4, 5)), d2l.eye(5)))
Q, K, V = X @ W_q, X @ W_k, X @ W_v  # (5, 2), (5, 2), (5, 5)
```

gave について見ると、クエリ $(1, 1)$ はドット積 2（各名詞のキー $(1, 1)$ との間）および 0（**gave** と「**the**」との間）をとる。$\sqrt{2}$ で割ると、スケール化されたスコアは各名詞で 1.41、それ以外で 0 となり、ソフトマックスによって各名詞の重みは $e^{1.41}/(3e^{1.41} + 2) \approx 0.29$、**gave** と「**the**」の重みは 0.07 に変換される（Figure 10.4）。

```
attention = d2l.DotProductAttention()
Y = attention(Q[None], K[None], V[None])  # A minibatch of one sentence
weights = attention.attention_weights[0]
scores = Q[1] @ K.T / math.sqrt(Q.shape[-1])  # Scaled by sqrt(d_h)
print('scores of gave: ', np.asarray(scores).round(2))
print('weights of gave:', np.asarray(weights[1]).round(2))
print('weights of the: ', np.asarray(weights[3]).round(2))
print('output of gave: ', np.asarray(Y[0, 1]).round(2))
attention(Q[None], K[None], V[None], d2l.causal_mask(5))
print('causal weights of gave:',
      np.asarray(attention.attention_weights[0, 1]).round(2))
d2l.show_heatmaps(weights[None, None], xlabel='keys', ylabel='queries',
                  figsize=(3.2, 3.2), cmap='Blues',
                  xticklabels=tokens, yticklabels=tokens)
```

```
scores of gave:  [1.41 0.   1.41 0.   1.41]
weights of gave: [0.29 0.07 0.29 0.07 0.29]
weights of the:  [0.2 0.2 0.2 0.2 0.2]
```

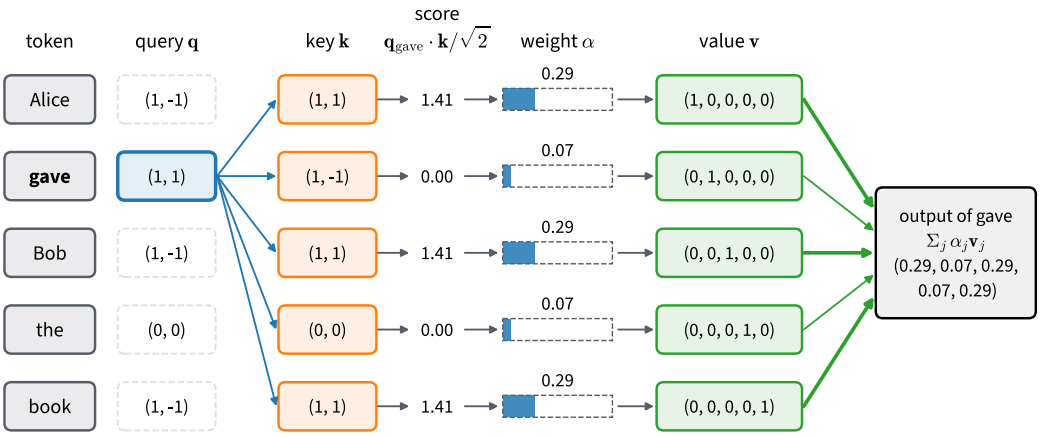
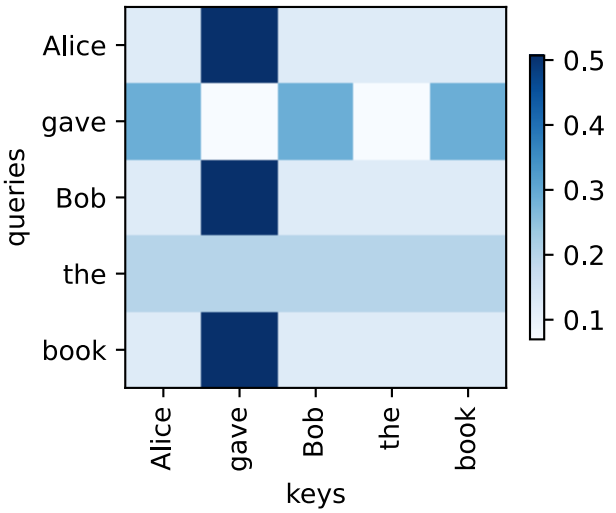


Figure 10.4: 上のコードで手動設定した写像（$d_h = 2$）を用いた、「Alice gave Bob the book」に対する自己アテンションの 1 つのヘッド。すべてのトークンがクエリ・キー・値を出力する。**gave** のクエリ $(1, 1)$ はすべてのキーに対して $\mathbf{q} \cdot \mathbf{k}/\sqrt{2}$ でスコア付けされ、ソフトマックスがスコア 1.41 と 0 を重み 0.29 と 0.07 に変換し、**gave** の出力は 5 つの単語を表す値の重み付き和となる。

```
output of gave:  [0.29 0.07 0.29 0.07 0.29]
causal weights of gave: [0.8 0.2 0.  0.  0. ]
```



逆に、各名詞のクエリ $(1, -1)$ は **gave** に対してスコア 1.41（重み 0.51）、それ以外に対して 0 を与え、一方で「**the**」はゼロクエリを持つため一様にアテンションを向ける（5 語すべてに 0.20）。因果マスク（causal mask）を適用すると、**gave** の参照先は **Alice** と自身のみに制限され、それらのスコア $(1.41, 0)$ は再正規化されて $(0.80, 0.20)$ になる。

10.3.1.2 クエリとキーに対する個別の写像

なぜ $\mathbf{W}_q$ と $\mathbf{W}_k$ に別々の行列を使うのだろうか。仮に単一の射影 $\mathbf{u}_i = \mathbf{x}_i\mathbf{W}$ がクエリとキーの両方を兼ねるとすると、スコア行列 $s_{ij} = \mathbf{u}_i^\top \mathbf{u}_j / \sqrt{d_h}$ は対称かつ半正定値（positive semidefinite）となり、コーシー・シュワルツの不等式により $s_{ij} \le \sqrt{s_{ii}\, s_{jj}} \le \max(s_{ii}, s_{jj})$ を満たす。すると、2 つのことが不可能になる。まず、$s_{ij} = s_{ji}$ であるため、動詞が名詞を求める強さを、名詞が動詞を求める強さより大きくすることはできない。また、2 つのトークンが互いに相手を自分自身より高くスコア付けすることも決してできない。動詞が自分自身より名詞を選好できるのは名詞のノルムがより大きい場合に限られ、そのとき名詞は動詞よりも自分自身を選好してしまうからである。クエリ・キー・値の射影を分けることで、**誰が探すか**、**誰が見つけられるか**、そして**どの内容を受け渡すか**が切り離される。

10.3.2 単一ヘッドの限界

先ほどの 5 語の文では、**gave** は **Alice**、**Bob**、**book** を 1 つの重み付き平均へと混合していた。クエリあたり 1 つのアテンション分布しか持たないことによるボトルネックを定量化するため、**両方コピータスク**（copy-both task）を考えよう。2 つのソース位置が、位置のみに依存する固定キーとともに独立な標準ガウス分布に従う値ベクトル $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^{d_v}$ を保持しており、単一のクエリが線形読み出し（linear readout）を介してそれらの連結 $\mathbf{t} = [\mathbf{v}_1; \mathbf{v}_2] \in \mathbb{R}^{2d_v}$ を出力しなければならないとする。

キーは位置のみに依存するため、単一ヘッドは固定のソフトマックス重み (α_1, α_2)（ただし $\alpha_1 + \alpha_2 = 1$）を割り当て、単一の混合ベクトル $\mathbf{m} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 \in \mathbb{R}^{d_v}$ を線形読み出しに渡す。

命題。位置のみに依存する 2 つの固定キー、独立な標準ガウス分布に従う値 $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^{d_v}$、および線形読み出しを持つ 1 つのアテンションヘッドにおいて、両方コピータスクで達成可能な最小の相対二乗誤差は $1/2$ であり、任意のソフトマックス対 (α_1, α_2) に対してこの値をとる。

証明。$\mathbf{m} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2$ が与えられたとき、$\mathbf{v}_1$ の最小平均二乗誤差推定量は条件付き期待値（conditional expectation）$\mathbb{E}[\mathbf{v}_1 \mid \mathbf{m}] = \frac{\alpha_1}{\alpha_1^2+\alpha_2^2}\,\mathbf{m}$ であり、これは $\mathbf{m}$ に関して線形であって、座標あたりの残差分散は $1 - \frac{\alpha_1^2}{\alpha_1^2+\alpha_2^2} = \frac{\alpha_2^2}{\alpha_1^2+\alpha_2^2}$ となる。対称性より、$\mathbf{v}_2$ の推定でも座標あたり $\frac{\alpha_1^2}{\alpha_1^2+\alpha_2^2}$ の残差分散が残る。これらの和は座標あたり 1 であり、ターゲットの全分散である座標あたり 2 に対する比をとると、相対二乗誤差は $1/2$ となる（(α_1, α_2) によらない）。■

両方コピータスクを解くには 2 つの要素が必要である。(1) キーに対する少なくとも 2 つの線形独立な**重み付け**と、(2) 取得した全 $2d_v$ 個の座標を保持するのに十分な**出力次元**（容量）である（演習問題 5）。以下では、単一ヘッドに対する二乗平均平方根相対誤差の下界 $\sqrt{1/2} \approx 0.707$ を検証するとともに、異なる重み付け（たとえば $(0.88, 0.12)$ と $(0.12, 0.88)$ のようなソフトな重み付けであっても）を持つ 2 つのヘッドがあれば、線形読み出しによって 2×2 の混合行列を浮動小数点精度まで逆変換できることを示す（Figure 10.5）。

```
N, d_v = 10000, 8
key1, key2 = jax.random.split(d2l.get_key())
V1, V2 = jax.random.normal(key1, (N, d_v)), jax.random.normal(key2, (N, d_v))
target = jnp.concatenate([V1, V2], axis=1)

def readout_error(M, target):
    """Best linear readout of target from M, relative error."""
```

```
    # Full fp32 products: TF32, the default on recent NVIDIA cards, would
    # stop the two-head errors near 1e-4
    with jax.default_matmul_precision('highest'):
        W = jnp.linalg.lstsq(M, target)[0]
        err = jnp.linalg.norm(M @ W - target) / jnp.linalg.norm(target)
    return float(err)

M = 0.7 * V1 + 0.3 * V2
print(f'one head, weights (0.70, 0.30): relative error '
      f'{readout_error(M, target):.3f}; sqrt(1/2) = {0.5 ** 0.5:.3f}')
for a in (1.0, 0.88):
    M2 = jnp.concatenate([a * V1 + (1 - a) * V2, (1 - a) * V1 + a * V2],
                         axis=1)
    print(f'two heads, weights ({a:.2f}, {1 - a:.2f}) and '
          f'({1 - a:.2f}, {a:.2f}): '
          f'relative error {readout_error(M2, target):.1e}')
```

```
one head, weights (0.70, 0.30): relative error 0.706; sqrt(1/2) = 0.707
two heads, weights (1.00, 0.00) and (0.00, 1.00): relative error 5.7e-07
two heads, weights (0.88, 0.12) and (0.12, 0.88): relative error 6.2e-07
```

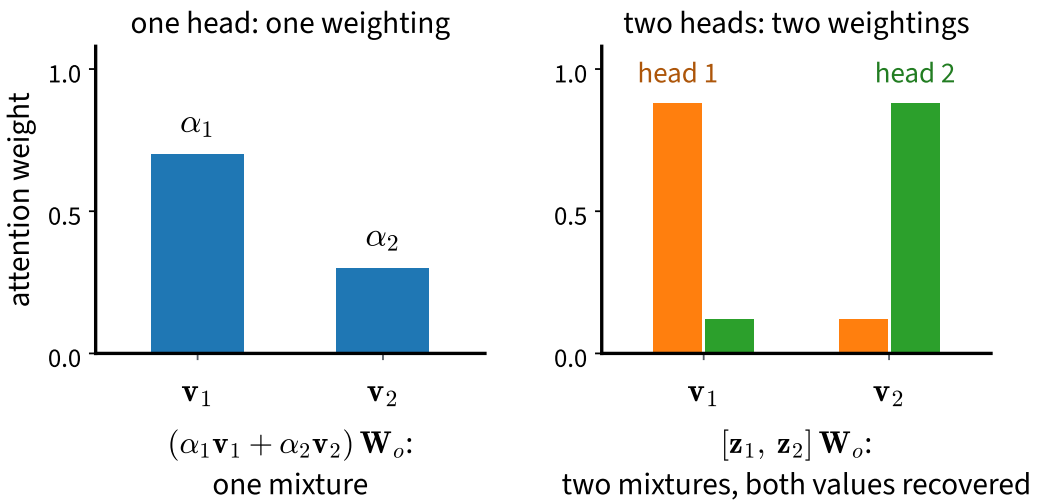


Figure 10.5: クエリあたり 1 つの重み付けと 2 つの重み付けの対比。単一ヘッドは 1 つのソフトマックス重み付けで両方コピータスクを処理するため、その出力では要求された 2 つの値が混ざり合う。異なる重み付けを持つ 2 つのヘッドは 2 つの混合ベクトルを供給し、そこから線形読み出しが両方の値を復元する。

10.3.3 マルチヘッドアテンション

10.3.3.1 射影されたヘッド

マルチヘッドアテンションでは (Vaswani et al. 2017)、H 個の独立なアテンションヘッドを並列に実行する。各ヘッドは独自のクエリ・キー・値の射影 $\mathbf{W}_i^{(q)}, \mathbf{W}_i^{(k)}, \mathbf{W}_i^{(v)} \in \mathbb{R}^{d \times d_h}$ を持ち、次のように計算される。

$$\mathbf{Z}_i = \text{softmax}\left(\frac{(\mathbf{X}\mathbf{W}_i^{(q)})(\mathbf{X}\mathbf{W}_i^{(k)})^\top}{\sqrt{d_h}}\right)\mathbf{X}\mathbf{W}_i^{(v)} \in \mathbb{R}^{n\times d_h}, \qquad i = 1, \dots, H, \tag{10.16}$$

続いて、それらの出力を連結したうえで、最終的な線形射影 $\mathbf{W}_o \in \mathbb{R}^{Hd_h\times d}$ を適用する（Figure 10.6）。

$$[\mathbf{Z}_1, \dots, \mathbf{Z}_H]\,\mathbf{W}_o \in \mathbb{R}^{n\times d}. \tag{10.17}$$

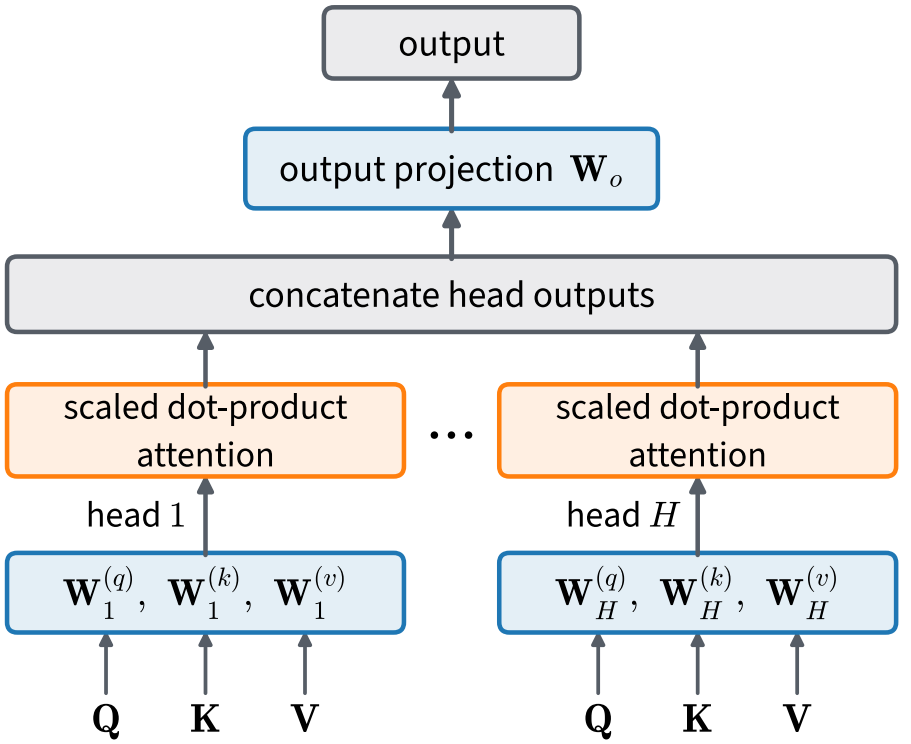


Figure 10.6: マルチヘッドアテンション。各ヘッドが同じクエリ・キー・値を読み込み、それぞれの部分空間へ射影してアテンションを計算する。ヘッドの出力は連結され、1 つの出力射影によって再結合される。

10.3.3.2 パラメータ数と FLOP 数

ヘッド幅を $d_h = d/H$ に設定すると、全ヘッドにわたる合計次元は $Hd_h = d$ に保たれる。横に並べると、H 個のクエリ射影は単一の $d \times d$ 行列をなし、キー・値・出力の各射影も同様である。1 回の積和演算を 2 FLOPs と数えると、n トークンに対するマルチヘッド自己アテンション層が必要とする計算量とパラメータ数は次のようになる。

$$\underbrace{8nd^2}_{\text{projections}} + \underbrace{4n^2d}_{\text{scores and mixing}} \text{ FLOPs}, \qquad 4d^2 \text{ parameters}. \tag{10.18}$$

GPT-2 small（$d = 768$、$n = 1{,}024$）の 1 層では、射影に 4.8 GFLOPs、スコア計算と値の混合に 3.2 GFLOPs を要し、パラメータ数は 2.4×10^6 個となる。注目すべきことに、パラメータ数も主要項の FLOP 数もヘッド数 H に依存しない。次元 d を H 個のヘッドに分割すると、各ヘッドの内積コストは $1/H$ に縮む一方で、それを H 個実行するからである（$O(Hn^2)$ のソフトマックスのオーバーヘッドのみが H に比例する）。逆に、d_h をいくらでも小さくできるわけではない。$d_h = 1$ ではスカラースコア qk_j が k_j に関して単調になるため、クエリは最大または最小の端にあるキーにしか集中できなくなる（演習問題 1）。実際には $d_h \in \{64, 128\}$ が標準的である。

10.3.3.3 畳み込み・再帰との比較

Figure 10.7 と Table 10.1 では、1 次元畳み込み（第 6 章）、再帰（8.4 節）、自己アテンションを、n 個のトークン（幅 d）に対する系列対系列の層として比較する。

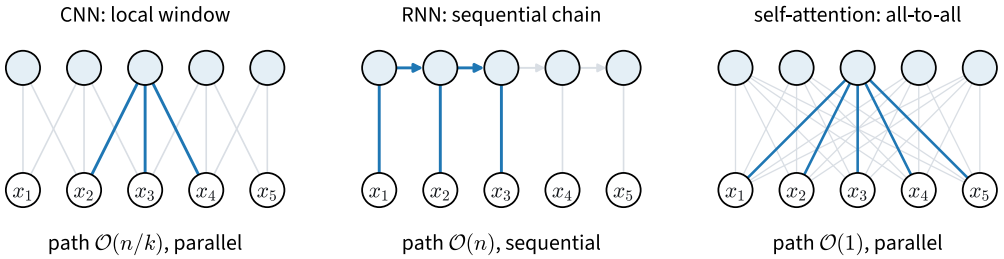


Figure 10.7: 5 トークンに対する系列層の結合構造。強調された辺は、1 層で入力が 3 番目の出力にどう到達するかを示す。カーネルサイズ $k = 3$ の畳み込みは近傍を参照し、再帰は連鎖に沿って 1 ステップずつ情報を伝え、自己アテンションはすべての位置を直接結ぶ。各パネルの下には、離れたトークン間の経路長と、出力が並列に計算されるかどうかを示す。

Table 10.1: n 個の d 次元トークンを n 個の d 次元トークンへ写す層あたりのコスト。

層の種類	計算量	逐次演算数	最大経路長
畳み込み（カーネル k）	$O(knd^2)$	$O(1)$	$O(n/k)$
再帰	$O(nd^2)$	$O(n)$	$O(n)$
自己アテンション	$O(nd^2 + n^2 d)$	$O(1)$	$O(1)$

自己アテンションは $O(1)$ の逐次ステップ数（訓練における完全な並列性）と任意のトークン対間の最大経路長 $O(1)$ を兼ね備えており、$O(n)$ ステップにわたるリカレントな勾配の連鎖を解消する（8.5 節）。その代償はスコア計算と混合にかかる二次の項 $4n^2 d$ であり、射影のコスト $8nd^2$ を $n > 2d$（GPT-2 small では $n > 1{,}536$）の領域で上回る。

10.3.3.4 実装

`MultiHeadAttention` では、単一の線形層が全 H 個のヘッドの射影を一度に計算し、`split_heads` が形状 `(batch, length, H * d_h)` を `(batch, H, length, d_h)` に変形することで、`DotProductAttention` が全ヘッドを 1 回のバッチ演算として実行できるようにする（Figure 10.8）。現代の Transformer の慣例に従い、射影ではデフォルトで加法バイアスを省略する（`bias=False`）。また、引数 `num_kv_heads` によってキー・値ヘッドの共有をサポートする（10.3.3.6 節）。

```
class Linear(nnx.Linear):
    """nnx.Linear with weights and biases drawn from U(-a, a), where
    a = 1/sqrt(in_features)."""
    def __init__(self, in_features, out_features, *, use_bias=True, rngs,
                 **kwargs):
        a = in_features ** -0.5
        init = lambda key, shape, dtype=jnp.float32: jax.random.uniform(
            key, shape, dtype, -a, a)
        super().__init__(in_features, out_features, use_bias=use_bias,
                         kernel_init=init, bias_init=init, rngs=rngs,
                         **kwargs)
```

```
class MultiHeadAttention(nnx.Module):
    """Multi-head attention with heads as an explicit axis; num_kv_heads <
    num_heads shares each key-value head across a group of query heads."""
    def __init__(self, num_hiddens, num_heads, dropout=0.0, bias=False,
                 num_kv_heads=None, *, rngs):
        assert num_hiddens % num_heads == 0, 'heads must divide num_hiddens'
        self.num_heads = num_heads
        self.num_kv_heads = num_kv_heads or num_heads
        assert num_heads % self.num_kv_heads == 0
        kv_hiddens = self.num_kv_heads * (num_hiddens // num_heads)
        self.attention = d2l.DotProductAttention(dropout, rngs=rngs)
        self.W_q = Linear(num_hiddens, num_hiddens, use_bias=bias, rngs=rngs)
        self.W_k = Linear(num_hiddens, kv_hiddens, use_bias=bias, rngs=rngs)
        self.W_v = Linear(num_hiddens, kv_hiddens, use_bias=bias, rngs=rngs)
        self.W_o = Linear(num_hiddens, num_hiddens, use_bias=bias, rngs=rngs)

    def __call__(self, queries, keys, values, mask=None):
        # queries: (batch, no. of queries, num_hiddens); keys and values:
        # (batch, no. of key-value pairs, num_hiddens). mask: boolean,
        # broadcastable to (batch, no. of queries, no. of keys), shared by
        # all heads, or to (batch, num_heads, no. of queries, no. of keys)
        q = self.split_heads(self.W_q(queries), self.num_heads)
        k = self.split_heads(self.W_k(keys), self.num_kv_heads)
        v = self.split_heads(self.W_v(values), self.num_kv_heads)
        group = self.num_heads // self.num_kv_heads
        k, v = jnp.repeat(k, group, axis=1), jnp.repeat(v, group, axis=1)
        if mask is not None and mask.ndim == 3:
            mask = mask[:, None]           # The same mask for every head
        # Shape of Y: (batch, num_heads, no. of queries, head width)
        Y = self.attention(q, k, v, mask)
        return self.W_o(self.merge_heads(Y))

    @property
    def attention_weights(self):
        """Weights of the last call: (batch, num_heads, queries, keys)."""
        return self.attention.attention_weights
```

形状変換と転置を担う 2 つの補助メソッド `split_heads` と `merge_heads`（Figure 10.8）は、アテンションの前に最後の次元 Hd_h を明示的なヘッド軸を持つ `(batch, num_heads, length, width)` へと折り畳み、アテンションの後に H 個のヘッド出力を再び連結する。

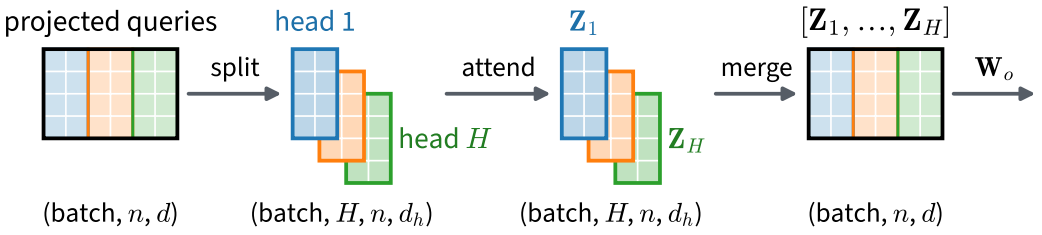


Figure 10.8: 1 つの射影のスライスとしての各ヘッド。n トークンの射影されたクエリを H 個の列スライス（幅 $d_h = d/H$）に分割し、独立したヘッド軸とする。各ヘッドが個別にアテンションを計算した後、出力射影 $\mathbf{W}_o$ の前でヘッド出力を再び横に並べる。キーと値も同様に分割される。

```
@d2l.add_to_class(MultiHeadAttention)
def split_heads(self, X, num_heads):
    """(batch, length, num_heads * width) -> (batch, num_heads, length,
    width)."""
    return X.reshape(X.shape[0], X.shape[1], num_heads, -1).swapaxes(1, 2)

@d2l.add_to_class(MultiHeadAttention)
def merge_heads(self, X):
    """Reverse split_heads: (batch, num_heads, length, width) -> (batch,
    length, num_heads * width)."""
    return X.swapaxes(1, 2).reshape(X.shape[0], X.shape[2], -1)
```

10.3.3.5 両方コピータスクにおける学習されたヘッド

ここで、幅 $d = 16$ の `MultiHeadAttention` を両方コピータスク（$d_v = 8$ をゼロパディングして幅 16 とし、キーを $\mathbf{e}_1, \mathbf{e}_2$、クエリを $\mathbf{e}_3$ とする）で検証しよう。

```
def copy_both_data(key, n, d=16, d_v=8):
    """One query, two position-only keys, values of width d_v padded to d."""
    V = jax.random.normal(key, (n, 2, d_v))
    keys = jnp.broadcast_to(jnp.eye(d)[:2], (n, 2, d))  # e_1, e_2: positions
    query = jnp.broadcast_to(jnp.eye(d)[2], (n, 1, d))  # e_3 asks for both
    values = jnp.pad(V, ((0, 0), (0, 0), (0, d - d_v)))
    return query, keys, values, V.reshape(n, 1, 2 * d_v)

@nnx.jit
def copy_both_step(model, optimizer, key):
    query, keys, values, target = copy_both_data(key, 256)
    loss = lambda model: ((model(query, keys, values) - target) ** 2).mean()
    optimizer.update(model, nnx.grad(loss)(model))

def train_copy_both(num_heads, num_kv_heads=None, num_steps=2000):
    model = MultiHeadAttention(16, num_heads, num_kv_heads=num_kv_heads,
                               rngs=nnx.Rngs(d2l.get_key()))
    optimizer = nnx.Optimizer(model, optax.adam(1e-2), wrt=nnx.Param)
```

```
    # Full fp32 products: TF32, the default on recent NVIDIA cards, would
    # stop the error of a solved task near 1e-3 instead of fp32 rounding
    with jax.default_matmul_precision('highest'):
        for key in jax.random.split(d2l.get_key(), num_steps):
            copy_both_step(model, optimizer, key)
    return model

def evaluate_copy_both(model, n=10000):
    """Relative error on fresh examples; each head's weight on v_1."""
    query, keys, values, target = copy_both_data(d2l.get_key(), n)
    with jax.default_matmul_precision('highest'):
        Y = model(query, keys, values)
    err = jnp.linalg.norm(Y - target) / jnp.linalg.norm(target)
    return float(err), model.attention_weights[0, :, 0, 0].tolist()
```

各ヘッド数 $H \in \{1, 2, 4\}$（いずれも合計幅 $d = 16$、パラメータ数 1,024 で共通）について 2 つのシードで訓練し、相対再構成誤差および各ヘッドのアテンション重み α_1（第 1 の値ベクトル $\mathbf{v}_1$ に対する重み）を比較する。

```
print('H  run  parameters  error    weight on v_1, per head')
for num_heads in (1, 2, 4):
    for run in (1, 2):
        model = train_copy_both(num_heads)
        err, alpha_1 = evaluate_copy_both(model)
        print(f'{num_heads}  {run:3d}  {d2l.num_params(model):10d}  '
              f'{err:<#8.2g} ' + ' '.join(f'{a:.2f}' for a in alpha_1))
```

```
H  run  parameters  error    weight on v_1, per head
1    1        1024  0.71     0.41
1    2        1024  0.71     0.54
2    1        1024  1.2e-07  0.96 0.08
2    2        1024  1.1e-07  0.02 0.82
4    1        1024  1.9e-07  0.00 0.98 0.23 0.54
4    2        1024  1.2e-07  0.98 0.14 0.05 0.74
```

いずれの構成もパラメータ数は $4 \times 16^2 = 1{,}024$ である。先の命題が予測したとおり、$H = 1$ は下界である $\sqrt{1/2} \approx 0.71$ で頭打ちになるのに対し、$H \in \{2, 4\}$ はヘッドごとに異なる重み付けを学習し、誤差を $\sim 10^{-7}$ まで縮小する。

10.3.3.6 キーと値の共有

自己回帰生成（autoregressive generation）の際、過去のキーと値はメモリにキャッシュされるため、新しい各トークンは過去の全位置の射影を再計算することなく参照できる（11.3 節）。**グループ化クエリアテンション**（GQA）では (Ainslie et al. 2023)、このキャッシュを H/H_{kv} 分の 1 に削減するため、各キー・値ヘッドを H/H_{kv} 個のクエリヘッドからなるグループで共有する。特に $H_{\text{kv}} = 1$ とすると、**マルチクエリアテンション**（MQA）が得られる (Shazeer 2019)。

両方コピータスクでは、共有された値ヘッドが張る空間は値あたり高々 $H_{\text{kv}} d_h$ 座標であるため、d_v 次元の値を復元するには $H_{\text{kv}} d_h \geq d_v$ が必要となる（演習問題 6）。$d = 16$、$d_v = 8$ のとき、構成 $(H, H_{\text{kv}}) = (2, 1)$ と $(4, 2)$ は $H_{\text{kv}} d_h = 8$ となって fp32 精度でタスクを解く一方、$(4, 1)$ は $H_{\text{kv}} d_h = 4$ しかなく、4 つのクエリヘッドにわたって異なる重み付けを学習しても 0.71 のボトルネックに逆戻りする。

```
print('H  H_kv  run  parameters  error    weight on v_1, per head')
for num_heads, num_kv_heads in ((2, 1), (4, 2), (4, 1)):
    for run in (1, 2):
        model = train_copy_both(num_heads, num_kv_heads)
        err, alpha_1 = evaluate_copy_both(model)
        print(f'{num_heads}  {num_kv_heads:4d}  {run:3d}  '
              f'{d2l.num_params(model):10d}  {err:<#8.2g} '
              + ' '.join(f'{a:.2f}' for a in alpha_1))
```

```
H  H_kv  run  parameters  error    weight on v_1, per head
2     1    1         768  1.0e-07  0.94 0.02
2     1    2         768  1.1e-07  0.99 0.09
4     2    1         768  1.2e-07  0.96 0.11 0.85 0.02
4     2    2         768  1.1e-07  0.18 0.98 0.04 0.91
4     1    1         640  0.71     0.11 0.94 0.05 0.96
4     1    2         640  0.71     0.91 0.03 0.94 0.07
```

10.3.4 クロスアテンション

自己アテンションが同じ系列 **X** をクエリ・キー・値として渡す（`attention(X, X, X)`）のに対し、**クロスアテンション**はターゲット系列 **A** からクエリを、ソース系列 **B** からキーと値を取り出し（`attention(A, B, B)`）、**A** のトークンごとに 1 行の出力を生成する。エンコーダ・デコーダ型の Transformer は翻訳時に符号化されたソース文を読み取るためにクロスアテンションを用い、マルチモーダルモデルはテキストトークンから視覚特徴量へクエリを送るためにこれを用いる (Alayrac et al. 2022)。

しかし、複数のヘッドを用いても、自己アテンションは依然として語順を識別できない。「**Alice gave Bob the book**」において **Alice** と **Bob** を入れ替えても、**gave** における出力は変わらないのである。系列の順序を注入する方法については 10.4 節で扱う。

10.3.5 まとめ

自己アテンションは系列の各トークンをクエリ・キー・値へと射影し、$O(1)$ の逐次ステップ数と $O(1)$ の経路長、および $O(nd^2 + n^2 d)$ FLOPs の計算量ですべての位置対を結ぶ。単一ヘッドはクエリあたり値の凸結合を 1 つしか形成できないため、独立な 2 つの値を同時に取得するタスクでは、いかなる単一ヘッドも少なくとも $1/2$ の相対二乗誤差を被る。マルチヘッドアテンションは、H 個の部分空間ヘッド（幅 $d_h = d/H$）を並列に実行して出力を $\mathbf{W}_o$ で結合することでこのボトルネックを克服し、必要なパラメータ数は $4d^2$、計算量は $8nd^2 + 4n^2 d$ FLOPs といずれも H に依存しない。グループ化クエリアテンションとマルチクエリアテンションは、$H_{\text{kv}} < H$ 個のキー・値ヘッドをクエリグループ間で共有することで生成時のキャッシュを H/H_{kv} 分の 1 に縮小しつつ、共有された値の幅 $H_{\text{kv}} d_h$ が取得対象の特徴量を収容できる限り検索能力を維持する。

10.3.6 演習問題

1. [code] **幅の狭いヘッド**。$d_h = d/H$ とすると、Equation 10.18 の主要項は H に依存しないため、ヘッド数は $H = d$ まで増やしてもコストが増えないように見える。マスクなしでスコア $\mathbf{q}^\top \mathbf{k}_j / \sqrt{d_h}$ をキー $\mathbf{k}_1, \dots, \mathbf{k}_m \in \mathbb{R}^{d_h}$ に対して計算する 1 つのヘッドを考える。クエリを限りなく増大する係数でスケールしたときに重みがキー j（すなわち $\mathbf{k}_j$）へ集中するとき、そのクエリはそのキーに対して鋭い（sharp）と呼ぶことにする。

1. $d_h = 1$ で互いに異なるスカラーのキーが与えられたとき、任意のクエリの重みが k_j に関して単調になることを示せ。これから、このようなヘッドのクエリが鋭くなりうるのは最大または最小のキーに対してのみであることを導け。
2. $d_h = 2$ のとき、クエリからキーへのいかなる割り当てに対しても、n 個の各クエリがそれぞれ任意に選んだキーに対して鋭くなるようにキーとクエリを選べることを示せ。
3. これら 2 つの結果を用いて、H が d に近づくにつれて層が表現できなくなるアテンションパターンを述べよ。
4. ドロップアウト 0 の `MultiHeadAttention` について、$d = 256$、$n = 512$、ミニバッチサイズ 8 系列の自己アテンションにおける順伝播の実行時間を、$H \in \{1, 2, 4, 8, 16, 32, 64\}$ にわたって計測せよ。最初にウォームアップ呼び出しを 1 回行い、計測を止める前に結果の完了を待つこと。最も遅い構成と最も速い構成の実行時間の比を報告せよ。さらに、H とともに増大し Equation 10.18 では省略されているコストを特定し、それがこの傾向を説明できることを確認せよ。

2. [code] **スライスとしてのヘッド**。`MultiHeadAttention` はクエリ・キー・値のそれぞれに対して幅 d の射影を 1 回ずつ計算し、`split_heads` によって各ヘッドへ分割する。
   1. ヘッド i は `W_q`、`W_k`、`W_v` の出力特徴量のスライスを担当する。`W_o` のどの入力特徴量がそのヘッドの出力を受け取るかを求めよ。さらに、ヘッドについてループを回し、スライスした射影を適用してヘッドごとに `d2l.DotProductAttention` を 1 回呼び出す順伝播を実装せよ。パディングマスク付きのランダムな入力に対し、これがドロップアウト 0 の `MultiHeadAttention` と一致することを検証し、さらに `num_kv_heads < num_heads` の場合へループを拡張せよ。
   2. `split_heads` の本体を `X.reshape(X.shape[0], num_heads, X.shape[1], -1)` に置き換えよ（これは同じ形状のテンソルを返す）。このとき各「ヘッド」がどの数値を保持するかを説明し、ランダムな入力で出力が変化することを示したうえで、形状のチェックだけではこの誤りを検出できない理由を説明せよ。2 つのバージョンの出力が一致するような入力の長さと幅は存在するか。存在するならばそれはどのような場合か。
3. **ソースの置換**。クロスアテンションにおいて、クエリは $\mathbf{A} \in \mathbb{R}^{n \times d}$ から、キーと値は $\mathbf{B} \in \mathbb{R}^{m \times d}$ から得られ、どちらも位置情報を持たないとする。Π を $m \times m$ の置換行列とする。
   1. マスクを適用しないとき、キーと値の両方の引数で $\mathbf{B}$ を $\Pi\mathbf{B}$ に置き換えても `MultiHeadAttention` の出力は変化せず、キーの引数のみを置き換えると一般に出力が変化することを示せ。
   2. 次に、パディングマスクによって先頭 ℓ 行のみが $\mathbf{B}$ の有効な行として指定されているとする。マスクを固定したままにしたとき、出力を変化させない置換はどのようなものか。
4. [code] **ほぼ同一のヘッド**。●重み $(0.5 + \epsilon, 0.5 - \epsilon)$ と $(0.5 - \epsilon, 0.5 + \epsilon)$ を持つ 2 つのヘッドは 2 つの異なる混合ベクトルを供給するため、任意の $\epsilon > 0$ に対して線形読み出しはそれらを厳密に逆変換できる。
   1. 2×2 の混合行列を書き下し、その固有値と固有ベクトルを計算して、条件数を ϵ の関数として表せ。
   2. 標準偏差 $\sigma = 0.01$ の独立なガウスノイズを両方の混合ベクトルに加え、`readout_error` を用いて $\epsilon \in \{0.4, 0.1, 0.03, 0.01, 0.003, 0.001\}$ に対する誤差を測定せよ。両対数軸上で ϵ に対して誤差をプロットし、曲線が直線になる領域の傾きを読み取って、(a) の条件数を用いてその傾きを説明せよ。
   3. 初期化直後の層の各ヘッドは、両方コピータスクの 2 つの位置に $(0.5, 0.5)$ 付近の重みを割り当てる。`train_copy_both` を変更して 20 ステップごとに相対誤差と各ヘッドの重みを記録し、$H = 2$ について両方をプロットせよ。最初の数百ステップにおける誤差曲線の形状を、混合行列の条件数（悪条件性）を用いて説明せよ。さらに $H = 8$ でも同様の実験を行い、最終的な各ヘッドを、ほぼ one-hot なもの、ソフトだが他と

異なるもの、別のヘッドとほぼ重複しているものに分類せよ。

5. **下界が数えているもの。**●10.3.2 節の命題ではソフトマックス重みを固定したが、その誤差下界はより弱い事実に依拠している。すなわち、読み出しは単一のベクトル $\mathbf{m} \in \mathbb{R}^{d_v}$ を受け取り、ターゲット $\mathbf{t} \in \mathbb{R}^{2d_v}$ を出力しなければならないという点である。
   1. $\mathbf{m} \in \mathbb{R}^{d_v}$ を $\mathbf{v}_1$ と $\mathbf{v}_2$ の任意の関数（重みが値に依存する混合ベクトルを含む）とし、$\mathbf{W}_o \in \mathbb{R}^{d_v \times 2d_v}$ を任意の行列とする。次の不等式が成り立つことを示せ。
   $$\mathbb{E}\,\|\mathbf{m}\mathbf{W}_o - \mathbf{t}\|^2 \geq \tfrac{1}{2}\,\mathbb{E}\,\|\mathbf{t}\|^2.$$
   これから、読み出しが線形であり受け取る数値が d_v 個である限り、内容依存のキー、層の追加、あるいは残差接続を導入してもこの下界は変わらないことを導け。
   2. 幅 d の層（幅 $d_h = d/2$ のヘッドを 2 つ持つ）が、値 $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^{d/2}$（ゼロパディングにより幅 d に拡張）を受け取るとする。このタスクを厳密に解くような各ヘッドのアテンション重み、値射影 $\mathbf{W}_i^{(v)}$、および出力射影 $\mathbf{W}_o$ を与えよ。また、その重みを 10.3.3.5 節で出力された重みと比較せよ。
   3. この命題を k 個のソース位置（独立な標準ガウス分布に従う値 $\mathbf{v}_1, \dots, \mathbf{v}_k \in \mathbb{R}^{d_v}$ とターゲット $[\mathbf{v}_1; \dots; \mathbf{v}_k]$ を持つ）へ一般化せよ。H 個の各ヘッドが値に対して固定の重みを適用し、ヘッド i の重みが第 i 行をなす行列を $\mathbf{A} \in \mathbb{R}^{H \times k}$ とし、線形読み出しが全 H 個の混合ベクトルを参照するとする。達成可能な最良の相対二乗誤差が次の式で与えられることを示せ。
   $$1 - \frac{\operatorname{rank}(\mathbf{A})}{k}, \tag{10.19}$$
   また、すべての成分が正であるソフトマックス重みによって $\operatorname{rank}(\mathbf{A}) = \min(H, k)$ を達成できることも示せ。
6. [code] **両方コピータスクにおけるキーと値の共有。**●幅 d の層が、H 個のクエリヘッド（幅 $d_h = d/H$）と $H_{\text{kv}} < H$ 個のキー・値ヘッドを持つとする。
   1. 演習問題 5 の次元カウントを用いて、この層がタスクを解けるのは $H_{\text{kv}} d_h \geq d_v$ の場合に限られること、またこの条件で十分であることを示せ。
   2. 2 つのクエリヘッドが 1 つのキー・値ヘッドを共有する `train_copy_both(2, 1)` で層を訓練せよ。2 つのヘッド間で異なるパラメータと共有されているパラメータを特定し、両方コピータスクの性質から、なぜその箇所に違いが生じなければならないのかを説明せよ。

10.4 位置情報

“dog bites man” と **“man bites dog”** という 2 つの文は、同じ 3 つの単語からなっている。これらを 10.3 節のマスクなし自己アテンション層に通すと、順序が入れ替わっただけのまったく同じ出力ベクトルが得られる。アテンション重みがトークンの内容のみに依存するためである。語順を区別するには、表現またはスコア計算規則に位置情報を注入しなければならない。

位置情報の導入手法は大きく 3 つの系統に分けられる。各トークン埋め込みに位置ベクトル $\mathbf{p}_i$ を加算する手法、位置に応じた角度でクエリとキーを回転させる手法（**回転位置埋め込み**（rotary position embeddings、RoPE））、そしてアテンションスコアから距離ペナルティを減算する手法（**ALiBi**）である。これら 3 系統はいずれも訓練コンテキスト長の内側では位置を区別できるが、より長い系列への外挿（extrapolation）を求められると大きく異なる振る舞いを示す。本節ではまず自己アテンションの置換対称性（permutation symmetry）を定式化し、続いて長さ 128 で訓練して最大長 512 まで評価する 2 層のアテンションのみのバイト言語モデル（`TinyCharLM`）上で 5 つの位置表現手法を構築・比較する。

10.4.1 置換同変性

$\mathbf{X} \in \mathbb{R}^{n \times d}$ を n 個のトークンベクトルを並べた行列とし、$f(\mathbf{X}) = \mathrm{softmax}(\mathbf{S})\,\mathbf{X}\mathbf{W}_v$（ただし $\mathbf{S} = \mathbf{X}\mathbf{W}_q(\mathbf{X}\mathbf{W}_k)^\top/\sqrt{d_h}$）をドロップアウトを無効化したマスクなし自己アテンション（10.3 節）とする。n 個のトークンの並び替えは、$\mathbf{X}$ に左から $n \times n$ の置換行列 Π を掛けることに対応する。

命題（置換同変性）。任意の置換行列 Π に対して、$f(\Pi\mathbf{X}) = \Pi f(\mathbf{X})$ が成り立つ。

証明。線形射影は行ごとに作用するため、$(\Pi\mathbf{X})\mathbf{W} = \Pi(\mathbf{X}\mathbf{W})$ が $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ のそれぞれについて成り立つ。したがって、スコア行列は $\Pi\mathbf{S}\Pi^\top$ となる。行ごとのソフトマックスは行および列の置換と可換であるため、$\mathrm{softmax}(\Pi\mathbf{S}\Pi^\top) = \Pi\,\mathrm{softmax}(\mathbf{S})\,\Pi^\top$ が得られる。これに置換後の値 $\Pi\mathbf{X}\mathbf{W}_v$ を掛け、$\Pi^\top\Pi = \mathbf{I}$ を用いると $\Pi f(\mathbf{X})$ となる。複数ヘッドを連結して行ごとに $\mathbf{W}_o$ を適用しても同変性は保たれる。■

4 ヘッドの `MultiHeadAttention` 層上で、置換同変性（permutation equivariance）を数値的に確認しよう。

```
attention = d2l.MultiHeadAttention(num_hiddens=64, num_heads=4,
                                   rngs=nnx.Rngs(d2l.get_key()))
key_X, key_perm = jax.random.split(d2l.get_key())
X = jax.random.normal(key_X, (1, 8, 64))
perm = jax.random.permutation(key_perm, 8)
Y = attention(X, X, X)
Y_perm = attention(X[:, perm], X[:, perm], X[:, perm])
print(f'max |Y[perm] - Y_perm|: {jnp.abs(Y[:, perm] - Y_perm).max():.2e}')
```

```
max |Y[perm] - Y_perm|: 0.00e+00
```

同変な層やトークンごとの多層パーセプトロン（MLP）を積み重ねても同変性は保たれる。ただし、**因果マスク**（causal mask）を導入すると位置 i が $i+1$ 個の先行要素を参照するようになるため、置換対称性が破れる。後述する **NoPE** はこの非対称性を利用している。

10.4.2 絶対位置埋め込み

オリジナルの Transformer (Vaswani et al. 2017) は、位置ベクトル $\mathbf{p}_i \in \mathbb{R}^d$（第 i 行にあたる $\mathbf{P}$ の行）を各トークン埋め込み $\mathbf{x}_i$ に加算することで、入力段階で置換対称性を破る。トークンをシャッフルすると $\mathbf{X}+\mathbf{P}$ は $\Pi\mathbf{X}+\mathbf{P} \neq \Pi(\mathbf{X}+\mathbf{P})$ へと写されるからである。

10.4.2.1 学習による位置表現

BERT (Devlin et al. 2018) や GPT-2 (Radford et al. 2019) などのモデルでは、$\mathbf{P} \in \mathbb{R}^{L_{\max} \times d}$ を、固定の最大長 $L_{\max}$ まで位置ごとに 1 つのパラメータ行を持つ訓練可能な埋め込みテーブル（embedding table）として扱う。訓練コンテキスト長を超える行は一度も勾配を受け取らないため、学習型のテーブルは未知の位置へ外挿できない。

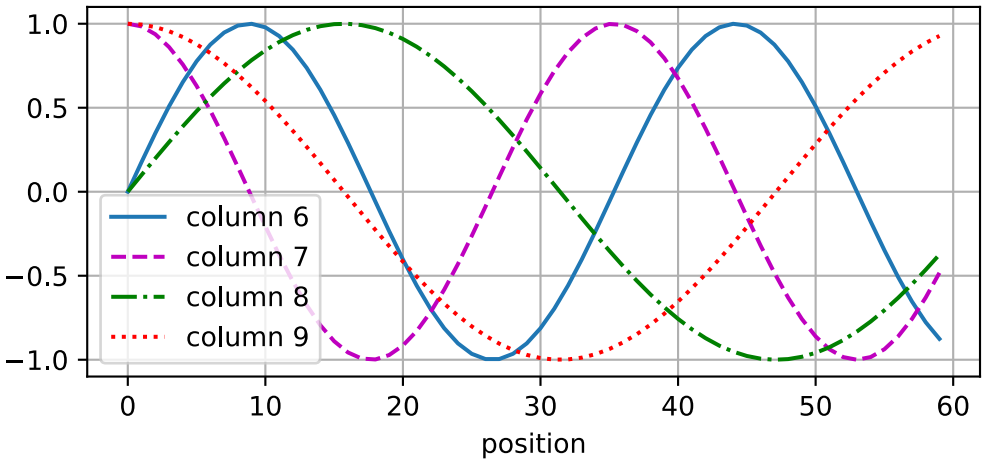
10.4.2.2 正弦波エンコーディング

学習パラメータを用いずに任意の長さに対する位置ベクトルを定義するため、Vaswani et al. (2017) は位置 i に対応する $\mathbf{P}$ の行に、等比的に配置された $d/2$ 個の周波数 $\omega_m = 10000^{-2m/d}$ における正弦と余弦を格納する。

$$p_{i,2m} = \sin\left(\frac{i}{10000^{2m/d}}\right), \qquad p_{i,2m+1} = \cos\left(\frac{i}{10000^{2m/d}}\right). \tag{10.20}$$

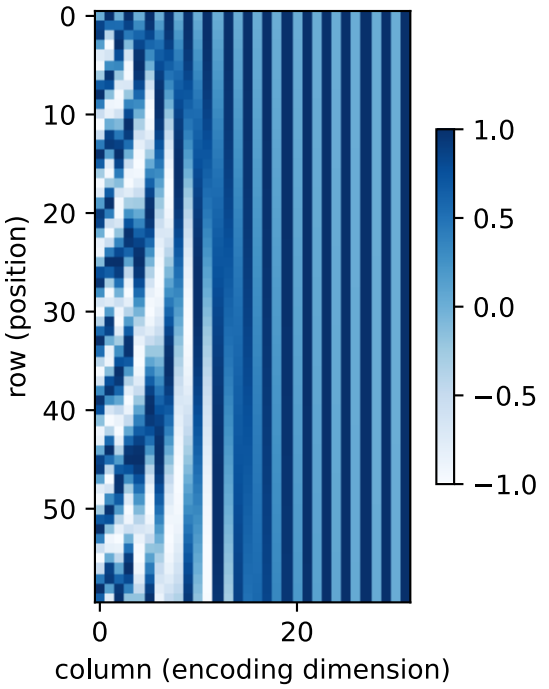
ω_m がトークンあたり 1 ラジアン ($m = 0$) から 10^{-4} ($m = d/2 - 1$) へと減少するにつれ、各列ペアは二進カウンタの連続版として機能する。すなわち、小さいインデックスの列は急速に振動して隣接する位置を区別し、大きいインデックスの列は広い領域にわたって緩やかに変化する。

```
def sinusoidal_encoding(max_len, num_hiddens):
    """The (max_len, num_hiddens) table of sinusoidal position codes."""
    theta = jnp.arange(max_len)[:, None] / 10000 ** (
        jnp.arange(0, num_hiddens, 2) / num_hiddens)
    return jnp.stack([jnp.sin(theta), jnp.cos(theta)],
                     -1).reshape(max_len, num_hiddens)

P = sinusoidal_encoding(60, 32)
d2l.plot(jnp.arange(60), P[:, 6:10].T, xlabel='position',
         figsize=(6, 2.5), legend=[f'column {d}' for d in range(6, 10)])
```



60×32 のエンコーディング行列 P 全体をヒートマップとして表示すると、整数カウンタの各ビットと同様に、左側の急速に振動する高周波ペアから右側の緩やかに変化する低周波ペアへと移り変わる様子がわかる。

```
d2l.show_heatmaps(P[None, None], xlabel='column (encoding dimension)',
                  ylabel='row (position)', figsize=(3.5, 4), cmap='Blues')
```



10.4.2.3 回転としての正弦波エンコーディング

三角関数の加法定理により、位置 i をオフセット δ だけシフトすると、2×2 回転（δ のみに依存）が各周波数ペア $(2m, 2m+1)$ に適用される。

$$\begin{bmatrix} \cos(\delta\omega_m) & \sin(\delta\omega_m) \\ -\sin(\delta\omega_m) & \cos(\delta\omega_m) \end{bmatrix} \begin{bmatrix} p_{i,2m} \\ p_{i,2m+1} \end{bmatrix} = \begin{bmatrix} \sin\left((i+\delta)\omega_m\right) \\ \cos\left((i+\delta)\omega_m\right) \end{bmatrix} = \begin{bmatrix} p_{i+\delta,2m} \\ p_{i+\delta,2m+1} \end{bmatrix}. \tag{10.21}$$

しかし、$\mathbf{p}_i$ は $\mathbf{x}_i$ に**加算**されるため、$\mathbf{A} = \mathbf{W}_q \mathbf{W}_k^\top / \sqrt{d_h}$ を用いた双線形のアテンションスコアは 4 つの項に展開される。

$$\begin{aligned} (\mathbf{x}_i + \mathbf{p}_i)^\top \mathbf{A} \, (\mathbf{x}_j + \mathbf{p}_j) = \; & \underbrace{\mathbf{x}_i^\top \mathbf{A} \, \mathbf{x}_j}_{\text{content–content}} + \underbrace{\mathbf{x}_i^\top \mathbf{A} \, \mathbf{p}_j}_{\text{content–position}} \\ & + \underbrace{\mathbf{p}_i^\top \mathbf{A} \, \mathbf{x}_j}_{\text{position–content}} + \underbrace{\mathbf{p}_i^\top \mathbf{A} \, \mathbf{p}_j}_{\text{position–position}}. \end{aligned} \tag{10.22}$$

2 つの交差項は内容と**絶対**位置（absolute position）を結合しており（i と j が一緒にシフトすると値が変化する）、さらに $\mathbf{p}_i^\top \mathbf{A} \, \mathbf{p}_j$ でさえも純粋に $j - i$ のみに依存するのは、$\mathbf{A}$ が 2×2 のスケール付き回転ブロックからなるブロック対角行列である場合に限られる（演習問題 2）。

10.4.3 回転位置埋め込み

回転位置埋め込み（RoPE）(Su et al. 2024) は、すべてのスコアが相対オフセット $j - i$ のみに依存するよう、クエリとキーに対して位置変換を**乗法的**に適用する。

$$(\mathbf{R}_i\mathbf{q})^\top(\mathbf{R}_j\mathbf{k}) = \mathbf{q}^\top\mathbf{R}_i^\top\mathbf{R}_j\mathbf{k} = \mathbf{q}^\top\mathbf{R}_{j-i}\mathbf{k} \quad \text{for all } i, j. \tag{10.23}$$

$\mathbf{R}_i^\top\mathbf{R}_j = \mathbf{R}_{j-i}$ および $\mathbf{R}_0$ がフルランクであることを要請すると、次が導かれる。(1) $\mathbf{R}_0 = \mathbf{I}$（$\mathbf{R}_0^\top\mathbf{R}_0 = \mathbf{R}_0$ より）。(2) $\mathbf{R}_i^\top\mathbf{R}_i = \mathbf{I}$ であり、したがってすべての $\mathbf{R}_i$ は直交かつノルム保存である。(3) $\mathbf{R}_{i+j} = \mathbf{R}_i\mathbf{R}_j$ であり、したがって $\mathbf{R}_i = \mathbf{R}_1^i$ となる。適切な正規直交基底（orthonormal basis、$\mathbf{W}_q, \mathbf{W}_k$ に吸収可能）のもとで、任意の直交行列 $\mathbf{R}_1$ は 2×2 の平面回転へと分解される。RoPE は、各クエリとキーの d_h 個の座標を $d_h/2$ 組のペアに分割し、ペア m を位置 i において角度 $i\omega_m$（ただし $\omega_m = 10000^{-2m/d_h}$）だけ回転させる（Figure 10.9）。

$$\begin{aligned}\mathbf{R}_i &= \begin{bmatrix} R(i\omega_0) & & \\ & \ddots & \\ & & R(i\omega_{d_h/2-1}) \end{bmatrix}, \\ R(\alpha) &= \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}.\end{aligned} \tag{10.24}$$

値は回転させないため、層の出力は大域的な位置シフトに対して不変に保たれる。RoPE は、現代のオープンウェイトの言語モデルにおける標準的な位置エンコーディング（positional encoding）である。

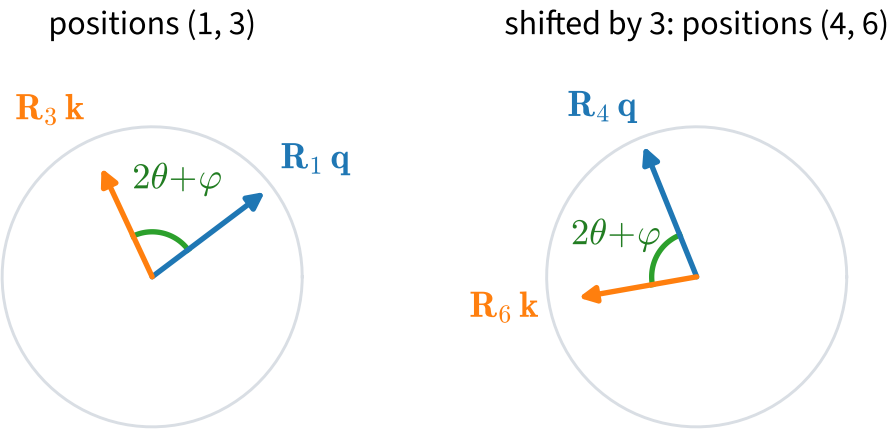


Figure 10.9: 回転埋め込みは、クエリとキーの 2 次元特徴ペアのそれぞれを位置に比例する角度だけ回転させる。両方の位置を同じ量（ここでは 3）だけシフトすると、両ベクトルが一緒に回転するため、両者のなす角（どちらのパネルでも $2\theta + \varphi$）、ひいてはアテンションスコアは変化しない。ここで、$\theta = \omega_m$ はペア m が位置ごとに回転する角度であり、φ は回転前のクエリとキーのなす角である。

```
def rope_tables(n, d, offset=0, base=10000.0, scale=None):
    """Cosines and sines, each (n, d/2), of the angles of the positions
    offset, ..., offset + n - 1."""
    pos = jnp.arange(n, dtype=jnp.float32) + offset
    omega = base ** (-jnp.arange(0, d, 2) / d)
    if scale is not None:  # One angle factor per pair, shape (d/2,)
```

```
        omega = omega * scale
    theta = pos[:, None] * omega[None, :]
    return jnp.cos(theta), jnp.sin(theta)

def rope(x, offset=0, base=10000.0, scale=None, tables=None):
    """Rotate feature pairs of x (..., num_steps, d) by position angles;
    tables from rope_tables take the place of offset, base, and scale."""
    if tables is None:
        tables = rope_tables(*x.shape[-2:], offset, base, scale)
    # Angles in fp32, tables in the dtype of x: bf16 queries stay bf16
    cos, sin = (u.astype(x.dtype) for u in tables)
    x1, x2 = x[..., 0::2], x[..., 1::2]
    return jnp.stack([x1 * cos - x2 * sin,
                      x1 * sin + x2 * cos], -1).reshape(x.shape)
```

すべての位置を任意のオフセットだけシフトしても、RoPE のスコアは float32 の丸め誤差の範囲で同一に保たれるのに対し、加算型正弦波のスコアは Equation 10.22 の内容・位置の交差項のために $O(1)$ だけ変化する。

```
T, d = 32, 64
key_q, key_k = jax.random.split(d2l.get_key())
q, k = jax.random.normal(key_q, (T, d)), jax.random.normal(key_k, (T, d))
# Full float32 matmuls: GPU default (TF32) round-off would hide the check
dot = lambda a, b: jnp.matmul(a, b.T, precision='highest') / math.sqrt(d)

scores = lambda off: dot(rope(q, off), rope(k, off))
for off in (1, 17, 480):
    print(f'RoPE, all positions shifted by {off:>3}: max score change '
          f'{jnp.abs(scores(off) - scores(0)).max():.1e}')

P = sinusoidal_encoding(600, d)
added = lambda off: dot(q + P[off:off+T], k + P[off:off+T])
print(f'additive sinusoidal, shifted by  17: max score change '
      f'{jnp.abs(added(17) - added(0)).max():.1e}')
```

```
RoPE, all positions shifted by   1: max score change 2.1e-06
RoPE, all positions shifted by  17: max score change 2.1e-06
RoPE, all positions shifted by 480: max score change 1.9e-05
additive sinusoidal, shifted by  17: max score change 2.8e+00
```

10.4.4 距離ペナルティとエンコーディングなし

さらに別の 2 つの手法では、トークン、クエリ、キーの各ベクトルには一切手を加えない。

- **ALiBi**（**Attention with Linear Biases**、(Press et al. 2022)）は、ヘッド $h \in \{1, \ldots, H\}$ における因果アテンション（causal attention）のスコアから線形の距離ペナルティを直接減算する。

$$\text{score}_{ij} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_h}} - m_h\,(i-j), \qquad m_h = 2^{-8h/H}. \tag{10.25}$$

等比的な傾き m_h（たとえば $1/4, 1/16, 1/64, 1/256$、$H = 4$ の場合）によってヘッドごとに異なる直近ホライズンが与えられ、固定のテーブルサイズを必要としない。

- **NoPE**（**No Positional Encoding**、(Kazemnejad et al. 2023)）は位置信号を一切加えず、因果マスク（$i + 1$ 個のキーを位置 i で参照可能にする）のみに頼って置換対称性を破る。

10.4.5 訓練長を超える外挿

これら 5 つの手法が分布内（in-distribution）およびより長いコンテキストへ外挿する際にどのように振る舞うかを比較するため、長さ $L = 128$ バイトで訓練し、長さ 128、256、512 でパープレキシティを評価する。

10.4.5.1 言語モデル TinyCharLM

`TinyCharLM` は 256 個のバイト値を幅 $d = 128$ に埋め込み、2 つの因果自己アテンションブロック（$H = 4$ 個のヘッド、各幅 $d_h = 32$ であり、位置情報が `pos` のみから入るように MLP や正規化層は設けない）を適用したのち、共有された埋め込み重みを用いてバイトロジットへと射影する (Press and Wolf 2017)。

```
class TinyCharLM(nnx.Module):
    """Attention-only language model."""
    def __init__(self, vocab_size, num_hiddens=128, num_heads=4, num_blks=2,
                 pos='rope', max_len=512, bias=False, *, rngs):
        assert pos in ('learned', 'sinusoidal', 'rope', 'alibi', 'none')
        self.num_heads, self.pos = num_heads, pos
        init = nnx.initializers.normal(0.02)
        self.token_emb = d2l.Embed(vocab_size, num_hiddens,
                                   embedding_init=init, rngs=rngs)
        if pos == 'learned':
            self.pos_emb = nnx.Embed(max_len, num_hiddens,
                                     embedding_init=init, rngs=rngs)
        if pos == 'sinusoidal':
            self.P = nnx.Cache(sinusoidal_encoding(max_len, num_hiddens))
        if pos == 'rope':  # Base and per-pair angle factors of the rotation
            self.rope_base = 10000.0
            self.rope_scale = nnx.Cache(
                jnp.ones(num_hiddens // num_heads // 2))
        self.blks = nnx.List([nnx.Dict(
            qkv=d2l.Linear(num_hiddens, 3 * num_hiddens, use_bias=bias,
                           rngs=rngs),
            proj=d2l.Linear(num_hiddens, num_hiddens, use_bias=bias,
                            rngs=rngs))
            for _ in range(num_blks)])

    def stream(self, X, num_blks=None):
        """Residual stream after the first num_blks blocks (default: all),
        and the attention weights of those blocks."""
        H, maps = self.embed(X), []
        for blk in self.blks[:num_blks]:
            out, weights = self.attend(blk, H)
            H, maps = H + out, maps + [weights]
        return H, maps
```

```
    def __call__(self, X):
        return self.token_emb.attend(self.stream(X)[0])  # Tied output head

    def attention_weights(self, X):
        """Per-block attention maps, each (batch, num_heads, T, T)."""
        return self.stream(X)[1]
```

embed では、重み共有された出力ロジットを変えることなく、正弦波テーブルを $1/\sqrt{d}$ 倍してその要素のスケールを $\mathcal{N}(0, 0.02^2)$ のトークン埋め込みに合わせる。

```
@d2l.add_to_class(TinyCharLM)
def embed(self, X):
    """Token embeddings plus the absolute position code, if any."""
    H = self.token_emb(X)
    if self.pos == 'learned':
        H = H + self.pos_emb(jnp.arange(X.shape[1]))
    if self.pos == 'sinusoidal':  # Table scaled to the token embeddings
        H = H + self.P[:X.shape[1]] / math.sqrt(H.shape[-1])
    return H
```

attend では、RoPE はクエリとキーを回転させるのに対し、ALiBi はソフトマックス前のスコアに距離バイアス $-m_h(i-j)$ を加算する。

```
@d2l.add_to_class(TinyCharLM)
def alibi(self, T):
    """ALiBi bias -m_h (i - j) of every head, (num_heads, T, T)."""
    h = jnp.arange(1, self.num_heads + 1)
    slopes = 2.0 ** (-8.0 * h / self.num_heads)
    pos = jnp.arange(T, dtype=jnp.float32)
    return slopes[:, None, None] * (pos[None, :] - pos[:, None])

@d2l.add_to_class(TinyCharLM)
def attend(self, blk, H):
    """Output of attention block blk on the stream H, and its weights."""
    B, T, D = H.shape
    q, k, v = jnp.split(blk['qkv'](H), 3, axis=-1)
    q, k, v = (u.reshape(B, T, self.num_heads, -1).swapaxes(1, 2)
               for u in (q, k, v))
    if self.pos == 'rope':
        q = rope(q, base=self.rope_base, scale=self.rope_scale[...])
        k = rope(k, base=self.rope_base, scale=self.rope_scale[...])
    scores = q @ k.swapaxes(-2, -1) / math.sqrt(q.shape[-1])
    if self.pos == 'alibi':
        scores = scores + self.alibi(T)
    weights = d2l.masked_softmax(scores, d2l.causal_mask(T))
    out = (weights @ v).swapaxes(1, 2).reshape(B, T, D)
    return blk['proj'](out), weights
```

10.4.5.2 実験

`d2l.WikiText103`（9.3.1 節）の 128 バイトウィンドウ上で、各手法につき 3 つのシードを 3,000 ステップの AdamW により訓練する。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
schemes = ('learned', 'sinusoidal', 'rope', 'alibi', 'none')

def train_tiny(**kwargs):
    """A TinyCharLM after 3000 steps, and its final training loss."""
    model = TinyCharLM(len(data.vocab), **kwargs,
                       rngs=nnx.Rngs(d2l.get_key()))
    optimizer = nnx.Optimizer(model, optax.adamw(1e-3, weight_decay=0.0),
                              wrt=nnx.Param)
    return model, d2l.final_loss(d2l.train_steps(model, data, optimizer,
                                                 3000))

runs = {}  # Three independently initialized models per scheme
for pos in schemes:
    trained = [train_tiny(pos=pos) for _ in range(3)]
    runs[pos] = [model for model, _ in trained]
    print(f'{pos:>10}: final training loss '
          + ', '.join(f'{loss:.2f}' for _, loss in trained))
models = {pos: ms[0] for pos, ms in runs.items()}  # The first run of each
```

```
   learned: final training loss 2.26, 2.26, 2.29
sinusoidal: final training loss 2.34, 2.22, 2.32
      rope: final training loss 2.00, 2.02, 1.98
     alibi: final training loss 2.29, 2.30, 2.31
      none: final training loss 2.47, 2.45, 2.47
```

閉形式のバイトバイグラムのベースラインとあわせて、同一の検証用バイト列を $n \in \{128, 256, 512\}$ のウィンドウに再編成したデータ上で検証パープレキシティを評価する。

```
train_len, batch = data.num_steps, 16
contexts = (train_len, 2 * train_len, 4 * train_len)
block = batch * max(contexts)  # Whole minibatches at every context
num_bytes = (len(data.val) - 1) // block * block
X, Y = data.windows(data.val, range(0, num_bytes, train_len))
eval_sets = {}
for n in contexts:  # The same bytes, regrouped into windows of n
    Xn, Yn = X.reshape(-1, n), Y.reshape(-1, n)
    eval_sets[n] = [(Xn[i:i + batch], Yn[i:i + batch])
                    for i in range(0, len(Xn), batch)]

def perplexities(model):
    return [d2l.lm_perplexity(model, eval_sets[n]) for n in contexts]

def mean_range(values):
    """Mean and (min-max) of the runs, one column of the table."""
    return f'{np.mean(values):5.1f} ({min(values):.1f}-{max(values):.1f})'

ppls = {pos: [perplexities(model) for model in ms] for pos, ms in runs.items()}
```

```
print(f'{num_bytes:,} validation bytes, mean (min-max) of three runs')
print(f'{"context":>10}: ' + ''.join(f'{n:>18}' for n in contexts))
for pos in schemes:  # zip(*...) turns runs x contexts into contexts x runs
    print(f'{pos:>10}: '
          + ''.join(f'{mean_range(col):>18}' for col in zip(*ppls[pos])))
```

```
1,138,688 validation bytes, mean (min-max) of three runs
   context:                128                256                512
   learned:      9.6 (9.4-9.7)  18.0 (15.3-20.7)  25.4 (19.8-31.8)
sinusoidal:      9.8 (9.1-10.3)  12.9 (12.4-13.4)  15.7 (14.3-16.5)
      rope:      7.3 (7.2-7.4)  12.4 (12.2-12.7)  27.4 (27.0-28.0)
     alibi:    10.0 (9.9-10.0)    9.9 (9.8-10.0)     9.9 (9.8-9.9)
      none:   11.7 (11.6-11.7)  11.8 (11.8-11.8)  11.9 (11.9-11.9)
```

これらの数値の基準点を得るため、訓練コーパス全体のラプラス平滑化された経験遷移カウントから、同じ検証ウィンドウ上での厳密なバイト単位バイグラムのパープレキシティを計算する。

```
V = len(data.vocab)
counts = np.ones((V, V))  # Byte pair counts, plus one for unseen pairs
for s in range(0, len(data.train) - 1, 10**7):  # Chunks of 10^7 bytes
    a = data.train[s:s + 10**7 + 1].astype(np.int64)
    counts += np.bincount(a[:-1] * V + a[1:], minlength=V * V).reshape(V, V)
bigram = counts / counts.sum(1, keepdims=True)  # P(next byte | this byte)
x, y = np.asarray(X).reshape(-1), np.asarray(Y).reshape(-1)
print(f'byte bigram: perplexity {np.exp(-np.log(bigram[x, y]).mean()):.1f}')
```

```
byte bigram: perplexity 11.6
```

訓練長（$n = 128$）では、RoPE が最も低いパープレキシティ（7.3）を達成し、学習型（9.6）、正弦波（9.7〜9.8）、ALiBi（9.9〜10.0）を上回る。これらはいずれもバイトバイグラムのベースライン（11.6）より改善しているが、この小さな 2 ブロックモデルでは NoPE はバイグラムの水準にとどまる。

$n = 512$ では、表現が訓練範囲外の値に遭遇する 3 つの手法が急激に劣化する。すなわち、学習型埋め込みは未訓練の行（128〜511）にあたり、正弦波の低速な列は未知の値をとり、RoPE の低周波ペアは訓練中に一度も現れなかった角度へと回転する。一方、ALiBi のパープレキシティは 9.9 のまま平坦に保たれる。その理由を調べるため、ALiBi の 4 つのヘッドが距離 ≥ 128 のキーに割り当てるアテンション重みの割合を計測し、距離 ≥ 128 のすべてのキーをマスクした場合と比較する。

```
alibi_model, T = models['alibi'], max(contexts)
X_eval = eval_sets[T][0][0]  # 16 windows of 512 bytes
i = jnp.arange(T)
far = i[:, None] - i[None, :] >= train_len  # Offsets never trained on

def far_weight(weights):
    """Per head: mean weight on far keys, over queries from train_len on."""
    return (weights * far).sum(-1)[..., train_len:].mean((0, -1)).tolist()

maps = alibi_model.attention_weights(X_eval)
bias_only = d2l.masked_softmax(alibi_model.alibi(T)[None], d2l.causal_mask(T))
print(f'{"head":>10}: ' + ''.join(f'{h:>9}' for h in range(1, 5)))
```

```
    for name, weights in [('bias alone', bias_only)] + [
            (f'block {b}', w) for b, w in enumerate(maps, 1)]:
        print(f'{name:>10}: ' + ''.join(f'{w:9.3f}' for w in far_weight(weights)))
```

```
      head:         1         2         3         4
bias alone:     0.000     0.000     0.115     0.401
   block 1:     0.000     0.000     0.115     0.401
   block 2:     0.000     0.001     0.119     0.390
```

これを WindowedALiBi によって直接確認しよう。このクラスは訓練済み ALiBi の重みをコピーし、距離 ≥ 128（window=train_len）のすべてのキーをハードマスクして長さ $T = 512$ で評価する。

```
    class WindowedALiBi(TinyCharLM):
        """ALiBi whose queries see only the window most recent keys."""
        def __init__(self, *args, window, **kwargs):
            super().__init__(*args, **kwargs)
            self.window = window

        def alibi(self, T):
            i = jnp.arange(T)
            return jnp.where(i[:, None] - i[None, :] >= self.window, -jnp.inf,
                             super().alibi(T))

    full, windowed = [], []
    for model in runs['alibi']:
        model_w = WindowedALiBi(len(data.vocab), pos='alibi', window=train_len,
                                rngs=nnx.Rngs(d2l.get_key()))
        nnx.update(model_w, nnx.state(model))  # Same trained weights
        full.append(d2l.lm_perplexity(model, eval_sets[T]))
        windowed.append(d2l.lm_perplexity(model_w, eval_sets[T]))
    for name, ppl in ('all keys', full), (f'keys < {train_len} back', windowed):
        print(f'{name:>16}: perplexity at {T} {mean_range(ppl)}')
```

```
       all keys: perplexity at 512   9.9 (9.8-9.9)
 keys < 128 back: perplexity at 512   9.9 (9.8-9.9)
```

距離 ≥ 128 のすべてのキーを取り除いても ALiBi のパープレキシティは変化しない。傾きの急なヘッドはもともと遠方のキーを無視しており、緩やかなヘッドは広い直近重み付き平均を計算していて 128 バイトより先の情報から何も得ていないためである。したがって、平坦な外挿曲線はその手法が遠方のキーで破綻しないことを示しているにすぎず、必ずしもそこから長距離の情報を抽出していることを意味しない。

10.4.5.3 訓練済みモデルのコンテキスト拡張

なぜ RoPE は訓練長 $L = 128$ を超えると破綻するのか、そしてどうすればそれを修復できるだろうか。ヘッド幅 $d_h = 32$ の場合、16 組の座標ペアは周期 $2\pi/\omega_m = 2\pi \cdot 10000^{m/16}$ を持つ。最初の 6 組の**高速**ペアは周期が $\leq 112 < 128$ であり、訓練ウィンドウ内で既に 1 周以上の回転を完了している。一方、残りの 10 組の**低速**ペアは周期が 199 から 35,000 に及び、$n = 512$ では未知の角度領域へと回転してしまう。

位置補間（position interpolation）では (Shouyuan Chen et al. 2023)、すべての角度を $128/512 = 1/4$ 倍にスケーリングして、新しいコンテキスト範囲を訓練済み区間 $[0, 128)$ に写し戻す。しかし、ファインチューニングなしに高速ペアまでスケーリングすると、隣接バイ

トの区別に使われている局所的な位相差が損なわれる。高速ペアには手を加えず低速ペア**のみ**をスケーリングする手法は、**YaRN** (Peng et al. 2024) や **NTK 考慮スケーリング**（NTK-aware scaling、基数を $b \mapsto b\, s^{d_h/(d_h-2)}$ へと大きくする手法）の中核にある考え方であり、訓練コストゼロで外挿時のパープレキシティの大半を回復できる。

```
rope_model = models['rope']
d_head = 2 * rope_model.rope_scale.shape[0]
periods = (2 * math.pi
           * rope_model.rope_base ** (jnp.arange(0, d_head, 2) / d_head))
slow = periods > train_len
print(f'{int(slow.sum())} of {d_head // 2} pairs have periods above '
      f'{train_len}')
ones = jnp.ones(d_head // 2)
shrink = train_len / max(contexts)  # Longest context into the trained one
factors = {'unscaled': ones, 'all pairs': ones * shrink,
           'slow pairs': jnp.where(slow, ones * shrink, ones)}
print(f'{"context":>10}: ' + ''.join(f'{n:>18}' for n in contexts))
for name, factor in factors.items():  # Each of the three trained RoPE runs
    ppl = []
    for rope_model in runs['rope']:
        rope_model.rope_scale[...] = factor
        ppl.append(perplexities(rope_model))
        rope_model.rope_scale[...] = ones
    print(f'{name:>10}: ' + ''.join(f'{mean_range(col):>18}'
                                    for col in zip(*ppl)))
```

```
10 of 16 pairs have periods above 128
   context:                128                256                512
  unscaled:      7.3 (7.2-7.4)  12.4 (12.2-12.7)  27.4 (27.0-28.0)
 all pairs:   39.7 (33.3-50.6)  40.1 (33.6-51.2)  40.2 (33.7-51.4)
slow pairs:      8.1 (7.9-8.5)     8.7 (8.3-9.5)     9.1 (8.6-10.1)
```

低速ペアのみを再スケーリングすると、重みを 1 つも更新することなく長さ 512 でのパープレキシティが約 27 から約 9 へと低下し、目標長で短時間のファインチューニングを行えば残りの差も解消される（演習問題 6）。

10.4.6 まとめ

マスクなし自己アテンションは置換同変であり、$f(\Pi\mathbf{X}) = \Pi f(\mathbf{X})$ を任意のトークン置換 Π に対して満たす。加算型の位置埋め込み（学習型または正弦波）は入力段階でこの対称性を破るが、絶対インデックスに結びついた内容・位置の交差項を生じさせ、訓練長を超えると劣化する。アテンションスコアが相対オフセット $j-i$ のみに依存することを要請すると、クエリとキーに対する直交平面回転（**RoPE**）が導かれるのに対し、**ALiBi** はヘッドごとの線形距離ペナルティ $-m_h(i-j)$ を減算する。訓練ウィンドウを超えると、RoPE は特に波長が訓練長を上回る低周波の座標ペアにおいて劣化するが、高周波ペアをそのまま残しつつこれらの低速ペアを選択的に補間することで、長文脈での性能が回復する。

10.4.7 演習問題

1. **異なる位置と異なる符号。** $\mathbf{p}_i$ を、第 i 行として正弦波テーブル Equation 10.20（幅 d、周波数 $\omega_m = 1/10000^{2m/d}$）から取り出した行ベクトルとする。

1. 次式を示し、
$$\mathbf{p}_i^\top \mathbf{p}_{i+\delta} = \sum_{m=0}^{d/2-1} \cos(\delta\, \omega_m),$$
これから $\|\mathbf{p}_i - \mathbf{p}_{i+\delta}\|^2$ を導出して、両方の量がオフセット δ のみに依存することを結論づけよ。
2. $\mathbf{p}_i = \mathbf{p}_{i'}$ が成り立つのは $(i - i')\,\omega_m$ が 2π の倍数である（すべての m について）場合に限られること、したがって `max_len` の値にかかわらず、相異なる 2 つの整数位置が同じ行を共有することはないことを結論づけよ。
3. 最も速いペアは周期 2π を持つため、2π の倍数に近い整数オフセットでほぼ同じ値を繰り返す。オフセット $\delta < 512$ のうち、その角度が 2π の倍数に最も近づくものを求め、残差（residual）の角度を示せ。$d = 32$ の場合について、このオフセットにおける $\|\mathbf{p}_i - \mathbf{p}_{i+\delta}\|$ を $\delta = 1$ における値と比較し、ほぼ反復する 2 つの位置をどのペアが引き離しているかを述べ、その答えを 10.4.2.2 節の二進カウンタと関連づけよ。

2. **スコアにおける加算型符号**。10.4.2.2 節の加算型エンコーディングを用いるとき、第 1 層のヘッドは位置 i と j の入力に対し、Equation 10.22 の 4 つの項（ただし $\mathbf{A} = \mathbf{W}_q \mathbf{W}_k^\top / \sqrt{d_h}$）によってスコアを計算する。
1. $\mathbf{J}$ を平面上の $\pi/2$ 回転とする。Equation 10.21 を用いて、位置・位置の項 $\mathbf{p}_i^\top \mathbf{A}\, \mathbf{p}_j$ は、$j - i$ のみに依存することを、$\mathbf{A}$ が周波数ペアごとに 1 つの 2×2 ブロック $a_m \mathbf{I} + b_m \mathbf{J}$ を持つブロック対角行列である場合について示せ。演習問題 1 は $\mathbf{A} = \mathbf{I}$ の場合を扱っている。次に、異なる 2 つの周波数ペアを結合する $\mathbf{A}$ の非ゼロ要素が 1 つでもあると、この性質が損なわれることを示せ。
2. 内容・位置の項 $\mathbf{x}_i^\top \mathbf{A}\, \mathbf{p}_j$ が j のあらゆるシフトに対して変化しない条件を求めよ。その答えを用いて、10.4.3 節のシフト実験における加算型エンコーディングのスコア変化と、RoPE に内容・位置の項が存在しない理由を説明せよ。

3. [code] **どこで外挿が破綻するか**。10.4.5.2 節におけるコンテキスト 512 でのパープレキシティは、各系列のすべての位置にわたって損失を平均したものである。
1. 5 つのモデルそれぞれにおいて、長さ 512 の系列の位置 0 から 127 におけるロジットが、先頭の 128 トークンのみから計算したロジットと浮動小数点の丸め誤差の範囲で一致することを数値的に確認せよ。`TinyCharLM` のどの 2 つの性質が、すべての手法に対してこれを保証しているか。
2. 5 つのモデルについて長さ 512 の系列の各位置における検証損失を計算し、16 位置ごとのウィンドウで平均して位置に対してプロットせよ。各手法について、曲線はどの位置で位置 128 未満で保っていた水準から離れるか。また、損失は位置 128 で不連続に跳ね上がるか、それとも徐々に増大するか。
3. `models['rope']` のすべてのクエリが自身を含む直近 128 個のキーのみを参照するよう、`attend` をオーバーライドするサブクラスで因果マスクに帯状の制約を追加し、訓練済みの重みをコピーして同じ計測を繰り返せ。位置 128 以降のウィンドウ制限付き曲線と、それより前の制限なし曲線とを比較せよ。127 を超えるオフセットは、コンテキスト 512 における RoPE の劣化を説明するか。

4. **対称性を保つバイアス**。●証明中のソフトマックス前のスコア行列に対し、固定行列 $\mathbf{B} \in \mathbb{R}^{n \times n}$ を $\mathbf{S}$ に加算して、自己アテンション層（10.3 節）が $\mathrm{softmax}(\mathbf{S} + \mathbf{B})\,\mathbf{X}\mathbf{W}_v$ を計算するものとする。
1. 置換行列 Π が $\Pi^\top \mathbf{B} \Pi = \mathbf{B}$ を満たす限り、命題が依然として成り立つことを示せ。
2. すべての置換に対してこの条件を満たす行列 $\mathbf{B}$ をすべて求め、そのそれぞれがアテンション重みをどのように変化させるかを述べよ。
3. 因果マスクがない場合、Equation 10.25 の ALiBi バイアスは対称な形 $B_{ij} = -m_h\,|i - j|$ へと拡張される。この行列を保つ置換をすべて求め、このバイアス以外に一切の位置信号を受け取らないマスクなし層に対してその結果が何を意味する

かを述べよ。また、Equation 10.25 の因果的な ALiBi がこの対称性を共有しない理由を説明せよ。

5. **RoPE におけるオフセット選好。**●クエリ $\mathbf{q}$ とキー $\mathbf{k}$ を、$d_h/2$ 組の 2 次元ペア $\mathbf{q}^{(m)}, \mathbf{k}^{(m)} \in \mathbb{R}^2$（Equation 10.24 が作用する単位）に分割し、$\delta = j - i$ をキー位置 j のクエリ位置 i からのオフセットとする。
   1. RoPE のスコア Equation 10.23 が次式に等しいことを示せ。

$$\sum_{m=0}^{d_h/2-1} \|\mathbf{q}^{(m)}\| \, \|\mathbf{k}^{(m)}\| \cos(\delta\, \omega_m - \psi_m), \tag{10.26}$$

   ここで、ψ_m は $\mathbf{k}^{(m)}$ から $\mathbf{q}^{(m)}$ への角度（符号を反転させた φ、Figure 10.9 参照）である。
   2. 角度 ψ_m を、すべてのペアが直前のトークンであるオフセット $\delta = -1$ で最大の寄与を与えるように選べ。すべての積 $\|\mathbf{q}^{(m)}\| \, \|\mathbf{k}^{(m)}\|$ が c に等しいとき、スコアは δ の関数として、c とドット積 $\mathbf{p}_i^\top \mathbf{p}_{i+\delta}$（演習問題 1 における幅 d_h の正弦波テーブルのもの）をシフトしたものの積になることを示せ。
   3. $d_h = 32$ とし、位置 127 にあるクエリ（オフセット $\delta = -127, \dots, 0$ を参照する）を考える。係数 $1/\sqrt{d_h}$ を c に吸収させ、アテンションロジットがこれらのスコアに等しいとする。このとき、c の値として、オフセット -1 がソフトマックス重みの半分を受け取るようなものを求め、それと最も強く競合するオフセットを特定せよ。オフセット -1 でのスコアとその隣接オフセットでのスコアとの差を決定づけているのはどのペアか。
6. [extended] **位置補間。**●10.4.5.3 節の最後のコードセルでは、訓練済みの RoPE モデルの角度を一切の追加訓練なしに再スケーリングしている。本問ではその結果の理由を解明し、NTK 考慮スケーリングの基数変更を加え、最後にファインチューニングによって手法を完成させる。
   1. 訓練オフセット 0 から 127 の範囲内で 1 回転を完了するペアは、より長いオフセットが生み出す任意の角度に対し、$\omega_m/2$ 以内の差（2π を法として）で既に出会っていることを示せ。演習問題 5 におけるオフセット -1 のスコアとその隣接オフセットのスコアとの差を、$d_h = 32$ の場合について、再スケーリング前、$128/512$ による一様な再スケーリング後、および周期が 128 位置を超えるペアのみを再スケーリングした後のそれぞれで計算せよ。その結果を、コードセル内のコンテキスト 128 におけるパープレキシティと関連づけよ。
   2. 基数 $b' = b\, s^{d_h/(d_h-2)}$ が ω_0 を不変に保ちつつ $\omega_{d_h/2-1}$ を s で割ることを示せ。属性 `rope_base`（`models['rope']` 内）を $s = 4$ に対するこの基数に設定し、訓練せずに長さ 128、256、512 で評価して、同セルの 2 つの再スケーリング手法と比較せよ。
   3. コンテキスト 512 のもと、`d2l.WikiText103(batch_size=16, num_steps=512)` 上で同セルの 2 つの再スケーリング手法のそれぞれについて（いずれも訓練済み `models['rope']` のコピーを初期値として）数百ステップだけファインチューニングせよ。長さ 128、256、512 で再評価し、スケーリングなしのモデルおよび 10.4.5.2 節の実験における同じ長さでの ALiBi モデルと比較せよ。

10.5 アテンションのコスト

長文脈（long-context）言語モデルは、1 回のパスで数万から数十万個のトークンを処理する。密な自己アテンションはあらゆる位置の対に対してスコアを評価するため、演算量と素朴な実装でのメモリ使用量はいずれも系列長 n の二乗に比例して増大する。$n = 8{,}192$ のとき、単一のヘッドは 6.7×10^7 個のクエリ・キー対（float32 で 256 MiB）を評価することになる。

数年にわたり、$O(n^2)$ のメモリは厳密なアテンションに避けられないコストだと広く考えられていた。その後、Rabe and Staats (2021) は Milakov and Gimelshein (2018) の**オンラインソフトマックス**のアルゴリズムを発展させ、厳密なアテンションにおいて $n \times n$ のスコア行列を実体化（materialize）する必要が一切ないことを示した。さらに **FlashAttention** (Dao et al. 2022) は、タイル化したオンラインソフトマックスを IO を考慮した GPU カーネルへと昇華させ、素朴なアテンションより高速でありながらメモリ使用量が線形にとどまる計算を実現した。

本節では、密なアテンションの FLOPs とメモリ使用量を算出して GPU 上でベンチマークし、ブロック単位のオンラインソフトマックスを導出したうえで、チャンク分割による実装および融合された FlashAttention カーネルを素朴な行列実体化と比較する。

10.5.1 二次の時間とメモリ

10.5.1.1 FLOPs の算出

幅 d、H 個のヘッド（各ヘッドの幅 $d_h = d/H$）、トークン数 n のマルチヘッド自己アテンション層では、マルチヘッドの計算量（10.3 節）に行ごとのソフトマックスを合わせると次式が得られる。

$$\underbrace{8nd^2}_{\text{projections}} + \underbrace{4n^2 d}_{\text{scores and mixing}} + O(Hn^2) \quad \text{FLOPs}. \tag{10.27}$$

二次の項 $4n^2 d$（$2n^2 d_h$ が $\mathbf{QK}^\top$ 分、$2n^2 d_h$ が $\mathbf{AV}$ 分で、H 個のヘッドの合計）は、線形な射影の項 $8nd^2$ を $n > 2d$ となった時点で上回る。GPT-2 small（$d = 768$、$n = 1{,}024$）ではまだ射影が支配的である。一方、$d = 4{,}096$ のモデルで $n = 131{,}072$ とすると $n/(2d) = 16$ となり、スコア計算と混合にはアテンションの射影に対して $16\times$ の FLOPs を要する（第 11 章のフィードフォワードネットワークを含めた Transformer ブロック全体に対しても約 $5\times$ であるが、これを含めるとブロック全体での逆転点は $n \approx 6d$ に移動する）。

10.5.1.2 メモリ使用量の算出

長文脈では、演算量よりも先にメモリがボトルネックになることが多い。素朴な実装は $n \times n$ のスコア行列とその $n \times n$ のソフトマックス重み行列の両方を実体化するため、float32 では 1 ヘッド・1 系列あたり $8n^2$ バイトを消費する（これに対し射影は $O(nd)$ である）。幅 $d_h = 64$ の 1 つのヘッドにおいて $n = 8{,}192$ のとき、$\mathbf{Q}, \mathbf{K}, \mathbf{V}$ の合計はわずか 6 MiB にすぎないが、2 つの $n \times n$ バッファは 512 MiB を消費する。さらに $n = 131{,}072$ では、単一の float32 アテンション行列だけで 64 GiB が必要になる。訓練時に GPT-2 small の全 12 層・12 ヘッドにわたってアテンション重みを $n = 8{,}192$ で保存しようとすれば、1 系列あたり 144×256 MiB $= 36$ GiB を要することになる。

$8n^2$ バイトの一時メモリ使用量を、1 つの float32 ヘッド（$d_h = 64$）上で `d2l.peak_memory`（5.7 節）を用いて確認しよう。

```
def attention(Q, K, V):
    """softmax(Q K^T / sqrt(d_h)) V through the full score matrix."""
    scores = Q @ K.swapaxes(-2, -1) / math.sqrt(Q.shape[-1])
    return jax.nn.softmax(scores, axis=-1) @ V
```

`d2l.peak_memory(attention, Q, Q, Q)` を系列長 $n \in \{2048, 4096, 8192, 16384\}$ にわたって評価すると、n が 2 倍になるごとに一時メモリの確保量が $8n^2$ バイトの理論値どおり 4 倍になることが確認できる。

```
d_h = 64
for n in [2048, 4096, 8192, 16384]:
    Q = jax.random.normal(d2l.get_key(), (n, d_h))
    mem = d2l.peak_memory(attention, Q, Q, Q)  # XLA's compile-time scratch
    print(f'n = {n:5d}: XLA temp {mem/2**20:7.1f}'
          f' MiB, predicted 8n^2 B = {8*n*n/2**20:7.1f} MiB')
```

```
n =  2048: XLA temp    32.0 MiB, predicted 8n^2 B =    32.0 MiB
n =  4096: XLA temp   128.0 MiB, predicted 8n^2 B =   128.0 MiB
n =  8192: XLA temp   512.0 MiB, predicted 8n^2 B =   512.0 MiB
n = 16384: XLA temp  2048.0 MiB, predicted 8n^2 B =  2048.0 MiB
```

10.5.1.3 実行時間の計測

同じ順伝播の実行時間を `d2l.Benchmark`（5.7 節）で計測すると、n がカーネル起動のレイテンシを償却できるほど大きくなった領域（$n \geq 8{,}192$）で、実行時間が二乗に比例して増加することが確認できる。

```
attention_jit = jax.jit(attention)
for n in [2048, 4096, 8192, 16384]:
    Q = jax.random.normal(d2l.get_key(), (n, d_h))
    print(d2l.Benchmark(lambda: attention_jit(Q, Q, Q), desc=f'n = {n:5d}'))
```

```
n =  2048: 0.15 ms/call
n =  4096: 0.25 ms/call
n =  8192: 1.21 ms/call
n = 16384: 4.66 ms/call
```

10.5.2 行列を保持しない厳密なアテンション

10.5.2.1 ブロック単位のオンラインソフトマックス

一見すると、ソフトマックスの分母 $\sum_{j'} \exp(a_{j'})$ と数値安定性のための行最大値 $\max_{j'} a_{j'}$（3.3.5.2 節）を求めるには、どの値に重み付けするよりも先に全 n 個のスコアが必要であるように思える。**オンラインソフトマックス** (Milakov and Gimelshein 2018) は、キーをサイズ c のブロック単位で処理し、クエリごとに 3 つの逐次統計量（running statistics）を保持することで、この 2 パスの要件を回避する。その 3 つとは、スコアの逐次最大値 m、分母の逐次和 $s = \sum_j e^{a_j - m}$、および未正規化の出力アキュムレータ $\mathbf{o} = \sum_j e^{a_j - m}\mathbf{v}_j$ である。

スコア $\{a_j\}$ と値 $\{\mathbf{v}_j\}$ の新しいブロックが到着したとき、逐次最大値を $m' = \max(m, \max_j a_j)$ に更新するには、それまでに累積した和 s と出力 $\mathbf{o}$ に $e^{m-m'}$ を掛けて再スケーリングする必要がある。

$$
\begin{aligned}
m' &= \max\left(m, \max_j a_j\right), \\
s' &= s\, e^{m-m'} + \sum_j e^{a_j - m'}, \\
\mathbf{o}' &= \mathbf{o}\, e^{m-m'} + \sum_j e^{a_j - m'}\, \mathbf{v}_j.
\end{aligned}
\tag{10.28}
$$

たとえば、ブロック 1 のスコアが $(1, 3)$ であれば、$m = 3$ かつ $s = e^{-2} + e^{0} \approx 1.135$ となる。次にブロック 2 のスコアが $(5, 2)$ であれば、新しい最大値は $m' = 5$ となるため、ブロック 1 の和は $e^{3-5} = e^{-2}$ 倍に再スケーリングされたうえで $e^{5-5} + e^{2-5}$ が加算され、$s' = e^{-4} + e^{-2} + e^{0} + e^{-3} \approx 1.203$ が得られる。これは大域的最大値 5 をあらかじめ知った状態で計算する標準的なソフトマックスの分母と一致する。最後のブロックを処理し終えると、$\mathbf{o}/s$ は浮動小数点の丸め誤差の範囲内で $\mathrm{softmax}(\mathbf{a})\mathbf{V}$ と等しくなり、しかも一度にメモリ上に保持するのは 1 つの $n \times c$ スコアストライプだけで済む（Figure 10.10、(Rabe and Staats 2021)）。

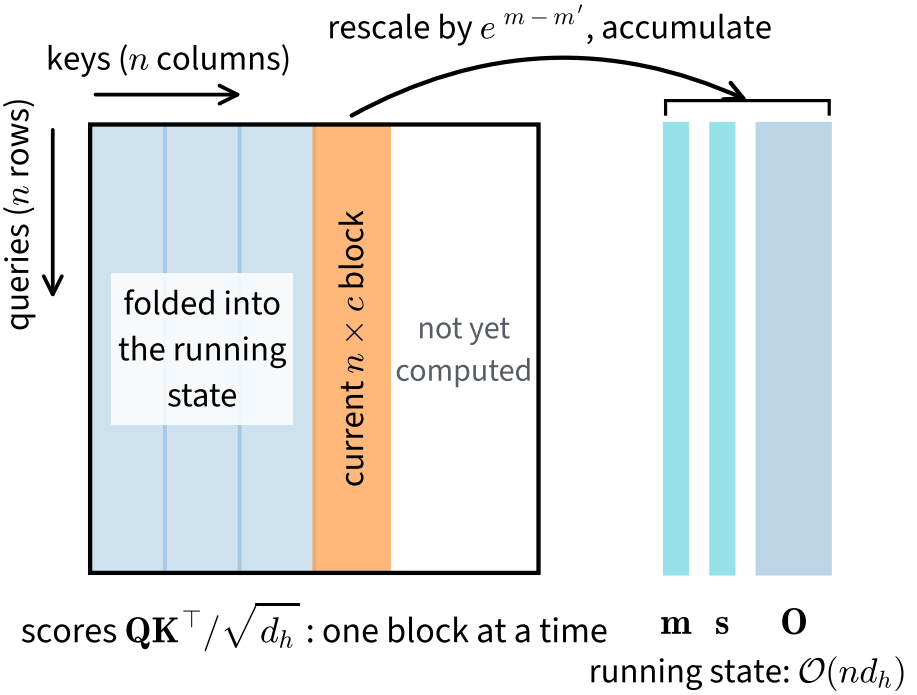


Figure 10.10: オンラインソフトマックスはスコア行列を 1 キーブロックずつ処理する。スコアのうちメモリ上に置かれるのは現在の $n \times c$ ストライプだけであり、その横に全クエリの逐次最大値・逐次和・出力 $\mathbf{m}, \mathbf{s}, \mathbf{O}$ が並ぶ。$n \times n$ の行列全体が保存されることは一切ない。

10.5.2.2 チャンク分割による実装

行列全体を用いる `causal_attention` と、`chunked_attention` を比較する。後者は幅 `chunk_size=512` のキー・値ブロックを Equation 10.28 に従って順に処理する（ブロック全体が因果マスクされるときに m の初期化と未来の位置のマスクに `finfo(dtype).min` を用いることで、$-\infty - (-\infty) =$ NaN の発生を防いでいる。演習問題 2d を参照）。

```
def causal_attention(Q, K, V):
    """Reference: causal attention with the full score matrix."""
    scores = Q @ K.swapaxes(-2, -1) / math.sqrt(Q.shape[-1])
    mask = d2l.causal_mask(Q.shape[-2])
    scores = jnp.where(mask, scores, jnp.finfo(scores.dtype).min)
    return jax.nn.softmax(scores, axis=-1) @ V
```

```
def chunked_attention(Q, K, V, chunk_size=512):
    """Exact causal attention, one n-by-chunk_size block at a time."""
    (n, d_h), d_v = Q.shape, V.shape[-1]
    assert n % chunk_size == 0, 'lax.scan needs equal-sized blocks'
    pos_q = jnp.arange(n)[:, None]
    def block(carry, chunk):
        m, s, O = carry
        Kc, Vc, start = chunk
        pos_k = start + jnp.arange(chunk_size)
        scores = Q @ Kc.T / math.sqrt(d_h)
        scores = jnp.where(pos_k[None, :] > pos_q,
                           jnp.finfo(scores.dtype).min, scores)
        m_new = jnp.maximum(m, scores.max(axis=-1, keepdims=True))
        scale = jnp.exp(m - m_new)                    # Rescale the past
        P = jnp.exp(scores - m_new)
        s = s * scale + P.sum(axis=-1, keepdims=True)
        O = O * scale + P @ Vc
        return (m_new, s, O), None
    init = (jnp.full((n, 1), jnp.finfo(Q.dtype).min), jnp.zeros((n, 1)),
            jnp.zeros((n, d_v)))
    chunks = (K.reshape(-1, chunk_size, d_h), V.reshape(-1, chunk_size, d_v),
              jnp.arange(0, n, chunk_size))           # Blocks, their starts
    (m, s, O), _ = jax.lax.scan(block, init, chunks)
    return O / s
```

chunked_attention は causal_attention と float32 の精度（$\sim 10^{-6}$）で一致しつつ、ピークメモリを $O(n^2)$ から $O(nc)$ へと削減する。

```
n, d_h = 2048, 64
Q, K, V = (jax.random.normal(k, (n, d_h))
           for k in jax.random.split(d2l.get_key(), 3))
# Full fp32 matrix products: the GPU default (TF32) would perturb the two
# computations differently, at the 1e-4 level
with jax.default_matmul_precision('highest'):
    err = jnp.abs(chunked_attention(Q, K, V)
                  - causal_attention(Q, K, V)).max()
print(f'maximum deviation: {float(err):.2e}')
```

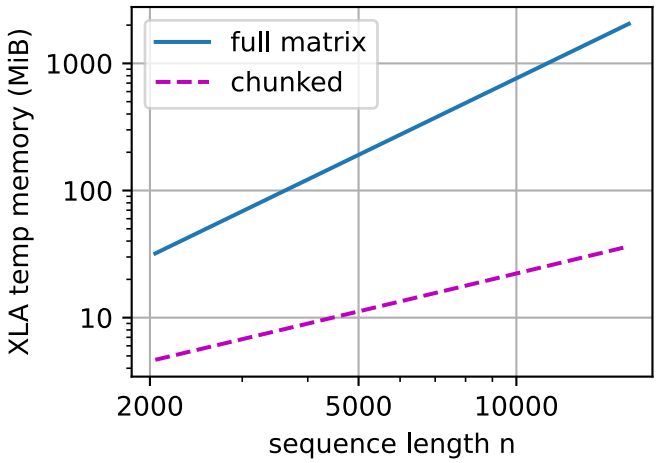
```
maximum deviation: 4.17e-07
```

d2l.peak_memory を $n \in \{2048, 4096, 8192, 16384\}$ にわたって比較すると、chunked_attention が causal_attention の二次曲線的な $O(n^2)$ のメモリ増加を線形な $O(nc)$ のスケーリングに置き換えていることが確認できる。

```
lengths, mems = [2048, 4096, 8192, 16384], [[], []]
for n in lengths:
    Q = jax.random.normal(d2l.get_key(), (n, d_h))
    mems[0].append(d2l.peak_memory(causal_attention, Q, Q, Q) / 2**20)
    mems[1].append(d2l.peak_memory(chunked_attention, Q, Q, Q) / 2**20)
print(f'n = {n:,}: full matrix {mems[0][-1]:6.1f} MiB, '
      f'chunked {mems[1][-1]:6.1f} MiB')
```

```
d2l.plot(lengths, mems, 'sequence length n', 'XLA temp memory (MiB)',
         legend=['full matrix', 'chunked'], xscale='log', yscale='log')
```

```
n = 16,384: full matrix 2048.0 MiB, chunked   36.4 MiB
```



10.5.2.3 メモリトラフィックと融合カーネル

GPU の演算ユニットは、オフチップのデバイスメモリがデータを転送できる速度よりも何桁も高速に行列演算を実行する（たとえば RTX 4090 では、float16 の ~ 165 TFLOP/s に対し、メモリ帯域幅は ~ 1 TB/s である。第 13 章を参照）。データセンター向けカードはこのメモリを広帯域メモリ（high-bandwidth memory、HBM）で構成し、RTX 4090 は GDDR6X で構成する。本節では FlashAttention の論文に倣い、いずれの種類のデバイスメモリも HBM と表記する。素朴なアテンションは**メモリ帯域幅律速**（memory-bandwidth bound）である。すなわち、$n \times n$ のスコア行列を HBM に書き込み、ソフトマックスのためにそれを読み戻し、$n \times n$ の重みを HBM に書き込み、さらに $\mathbf{V}$ を掛けるために再びそれを読み出すため、低い演算密度（arithmetic intensity）のまま $4n^2$ 個の浮動小数点数を HBM 上で移動させることになる（演習問題 4 および 5）。`chunked_attention` のような Python ループも、結局は個別の GPU カーネルを起動してすべての $n \times c$ ストライプを HBM に対して読み書きするため、ピーク時のメモリ確保量は削減できても HBM の総トラフィックは削減できない。

FlashAttention (Dao et al. 2022) はクエリとキーの両方をタイル化し、各スコアブロックとその逐次統計量 $(m, s, \mathbf{o})$ がループの間ずっと高速なオンチップ SRAM 内に留まり、HBM に一切触れないようにする。逆伝播に向けては、順伝播で $n \times n$ のアテンション重みを保存する代わりに、クエリごとにスカラーの対数正規化項（log-normalizer）$\ell = m + \log s$ のみを保存し（$O(n)$ メモリ）、誤差逆伝播法の実行中にオンチップでスコアタイルを再計算する。

フレームワークが提供する融合 FlashAttention カーネルを `causal_attention` と比較し、float16 で $B = 2$ 系列、$H = 8$ ヘッド、$d_h = 64$、$n = 8{,}192$ の条件でベンチマークする。マスクなしの融合カーネルの実行も含めることで、$O(n^2)$ の HBM トラフィック削減による高速化と、上三角の因果タイルをスキップすることによる $\sim 1.5\times$ の高速化とを切り分ける。

融合カーネルは、`jax.nn.dot_product_attention` に `implementation='cudnn'` を指定したものである。

```
def fused_attention(Q, K, V, is_causal=True):
    """Attention in the fused cuDNN kernel."""
    # The kernel takes (batch, length, heads, width): swap axes 1 and 2
    Q, K, V = (A.swapaxes(1, 2) for A in (Q, K, V))
    O = jax.nn.dot_product_attention(Q, K, V, is_causal=is_causal,
                                     implementation='cudnn')
    return O.swapaxes(1, 2)

B, H, n = 2, 8, 8192
X = jax.random.normal(d2l.get_key(), (B, H, n, d_h), dtype=jnp.float16)
# The unmasked kernel computes every tile: it shows what skipping saves
for name, f in [('naive', causal_attention), ('fused', fused_attention),
                ('fused, no mask',
                 lambda Q, K, V: fused_attention(Q, K, V, is_causal=False))]:
    f_jit = jax.jit(f)
    t = d2l.Benchmark(lambda: f_jit(X, X, X)).time
    mem = d2l.peak_memory(f, X, X, X)
    print(f'{name:>14}: {t*1e3:6.2f} ms, XLA temp {mem/2**20:7.1f} MiB')
```

```
         naive:   9.32 ms, XLA temp  4096.0 MiB
         fused:   1.29 ms, XLA temp    48.0 MiB
fused, no mask:   1.79 ms, XLA temp    48.0 MiB
```

融合カーネルは一時メモリを ~ 4 GiB（$16 \cdot 4n^2$ バイト）から 16 MiB の出力の数倍程度まで削減し、GPU のピーク行列スループットに近い速度で数倍高速に動作する。しかし、融合カーネルを用いてもなお、厳密なアテンションは $\Theta(n^2 d)$ 回の算術演算を必要とする。線形時間計算量を達成するには、アテンション機構そのものを変更する必要がある（10.6 節）。

10.5.3 まとめ

密な自己アテンションは 1 層あたり $8nd^2 + 4n^2 d$ FLOPs を要するため、線形の射影よりも二次のスコア計算・混合の演算量が $n > 2d$ となった時点で大きくなる。さらに厄介なことに、$n \times n$ のスコア行列と重み行列を実体化すると float32 で 1 ヘッドあたり $8n^2$ バイトが確保され、現代の GPU では素朴なアテンションがメモリ帯域幅律速に陥る。オンラインソフトマックスは 2 つの $n \times n$ バッファを近似なしで不要にする手法であり、キーをブロック単位で処理して逐次最大値 m、正規化項 s、値アキュムレータ $\mathbf{o}$ を保持し、その部分和を $e^{m-m'}$ 倍に再スケーリングする（m が増加するたびに行う）。FlashAttention はこのタイル化された漸化式を高速なオンチップ SRAM 内で融合し、オフチップメモリを $O(n)$ に削減して実経過時間で数倍の高速化を達成しつつ、厳密なアテンションとその $\Theta(n^2 d)$ の演算量を維持する。

10.5.4 演習問題

1. **手計算によるオンラインソフトマックス**。1 つのクエリが、2 個ずつ 2 つのブロックに分かれた 4 つのキーと照合されるとする。スコアは第 1 ブロックで $(0, 2)$、第 2 ブロックで $(1, 3)$ であり、値は第 1 ブロックで数値 $1, 2$、第 2 ブロックで $3, 4$ である。
    1. Equation 10.28 をブロックごとに適用せよ。初期値は $m = -\infty$、$s = 0$、$o = 0$ とする。各ブロックの処理後の m、s、o を記録し、除算して出力を求めよ。
    2. 4 つのソフトマックス重みを直接計算し、同じ出力が得られることを確かめよ。
    3. 最後のブロックの処理後に $\ell = m + \log s$ を作り、4 つの重みのそれぞれをそのスコアのみから $e^{a_j - \ell}$ として再計算せよ。これらが前問の重みと一致し、かつ和が 1 になることを確かめ、融合カーネルの逆伝播において ℓ のみを保持すればよい（m と s の

代わりに）理由を説明せよ。

2. [code] **逐次最大値**。`chunked_attention` から逐次最大値を取り除き、状態が e^{a_j} の単純な和を累積するように変更せよ。**Q**、**K**、**V** および状態 s、**O** を float16 とし、10 倍にスケールアップしたクエリに対してその結果を実行せよ。
   1. 実行する前に、最大の指数 x（float16 で e^x が有限となるもの）を計算し、幅 64 のクエリとキーが独立な標準正規分布に従う成分をもつとき、スケーリング後のスコアがこの値を超える割合を見積もれ。
   2. 変更した関数を実行し、最初の無限大または未定義値がどの演算で現れるかを報告せよ。
   3. m を保持することで、すべての指数が 0 以下に抑えられる理由を説明せよ。正の最小の float16 数より小さい項 $e^{a_j - m'}$ は 0 に丸められる。そのような項の s への寄与の総和を、最大スコアの項に対する相対値として、n 個のキーからなる 1 行について上から評価せよ。
   4. `chunked_attention` は m をその dtype の最小の有限値で初期化し、スコアも同じ値でマスクする。$n = 2c$（キー c 個ずつのブロック）とし、ループが 2 つのキーブロックを逆順に走査するとする。このとき、最初に走査されるブロックのスコアは、第 1 ブロック内のクエリに対してすべてマスクされる。そのクエリに対する m、s、**O** を両方の反復にわたってトレースし、最終出力が依然として厳密であることを示せ。
3. **部分状態のマージ**。スコア a_j と値 $\mathbf{v}_j$ をもつ 1 つのクエリを固定する。キーの集合 B に対し、3 つ組 $(m_B, s_B, \mathbf{o}_B)$（Equation 10.28 が B 内のキーのみから生成するもの）を B の部分状態と呼ぶことにすると、$\mathbf{o}_B / s_B$ は B に制限されたアテンション出力となる。
   1. $B \cup B'$ の部分状態を、互いに素な集合 B と B' の 2 つの部分状態から導出し、Equation 10.28 が B' を単一のブロックとした特別な場合であることを示せ。
   2. このマージ操作が結合的かつ可換であることを示せ。このことから、キーブロックは任意の順序で処理でき、あるいは並列に処理して深さ $\lceil \log_2(n/c) \rceil$ の木（ブロックあたり c 個のキー）でマージできると結論づけ、生成時に単一のクエリが長いキャッシュにアテンションを向ける場面でなぜこれが重要になるかを説明せよ。
   3. 対 (m_B, s_B) を単一の数値 $\ell_B = m_B + \log s_B$ に、3 つ組を $(\ell_B, \bar{\mathbf{o}}_B)$（ただし $\bar{\mathbf{o}}_B = \mathbf{o}_B / s_B$）に置き換えよ。マージ後の ℓ と $\bar{\mathbf{o}}$ をこれらの量で表し、$\bar{\mathbf{o}}$ が $\bar{\mathbf{o}}_B$ と $\bar{\mathbf{o}}_{B'}$ の凸結合であり、その重みが ℓ_B と $\ell_{B'}$ のみに依存することを示せ。
4. **アテンションの演算密度**。n 個のクエリとキー（幅 d_h）および値（幅 d_v）をもつ 1 つのヘッドを考える。2 つの行列積である $\mathbf{QK}^\top$ と重みおよび **V** の積には、それぞれ $2n^2 d_h$ 回と $2n^2 d_v$ 回の浮動小数点演算を要し、ソフトマックスには 1 スコアあたり数回の演算を要する。
   1. スコア行列がメモリに 1 回書き込まれてソフトマックスで 1 回読み出され、その出力が 1 回書き込まれて値との積で 1 回読み出されると仮定する（すべて単精度とする）。これら 4 回のパスで移動するバイト数と、**Q**、**K**、**V** の読み出しおよび出力の書き込みで移動するバイト数を数えよ。
   2. ソフトマックスの演算を無視してバイト数に対する演算回数の比を作り、$n \to \infty$ の極限（d_h と d_v は固定）をとれ。その極限が $(d_h + d_v)/8$ になることを示し、ソフトマックスの演算を含めるとこれがどう変わるかを述べよ。
   3. 入力とスコアを 16 ビット形式とした場合について同じ極限を求めよ。H100 または B200 の 16 ビットのスループットとメモリ帯域幅を Table 13.2 から読み取り、1 バイトあたりの演算回数の比を計算して、その GPU 上で $d_h = d_v = 64$ のアテンションが演算とメモリトラフィックのどちらに律速されるかを判定せよ。
5. **タイル化されたスケジュールのトラフィック**。●演習問題 4a より、素朴な層は主メモリとの間で $4n^2$ 個の浮動小数点数を移動させる（n に関して線形な項を除く。スコア行列と重み行列がそれぞれ 1 回書き込まれ 1 回読み出されるためである）。以下では n に関して線形なトラフィックをすべて無視せよ。
   1. `chunked_attention` が一度に 1 つずつ演算を実行し、各演算が主メモリから入力を読み

出して出力を書き戻すとする。n、c、d_h の関数として、スコアストライプのために移動する浮動小数点数、$\mathbf{Q}$ とアキュムレータ $\mathbf{O}$ のために移動する浮動小数点数、および起動される演算の回数を数えよ。素朴な層と比べて、チャンク分割はこれらの量のうちどれを削減するか。

2. 一方、融合カーネルは b 個のクエリからなるタイル、それらの出力アキュムレータ、および b 個のキーと b 個の値からなるタイルを各 1 つずつ、容量 M 個（浮動小数点数換算）の高速メモリ内に保持し、$4bd_h \le M$ を満たすとする。タイルのスコアおよび逐次統計量 m と s も高速メモリ内に留まる。このカーネルは各クエリタイルを 1 回読み込み、すべてのキータイルと値タイルをそこへ順にストリーミングし、各出力タイルを 1 回書き出す。線形な項を除いたトラフィックが $2n^2 d_h / b$ 個の浮動小数点数になること、そして許容される最大のタイルをとると $\Theta(n^2 d_h^2 / M)$（Dao et al. (2022) が FlashAttention に対して証明した上界）が得られることを示せ。
3. $d_h \in \{64, 128, 256\}$、$M = 2^{16}$ に対して素朴な層のトラフィックに対する融合カーネルのトラフィックの比を計算し、ヘッド幅がタイリングの恩恵をどのように制限するかを述べよ。

10.6 ウィンドウアテンションと線形アテンション

融合カーネル（fused kernel、10.5 節）は $O(n^2)$ の中間スコア行列を不要にするが、$\Theta(n^2 d)$ の FLOP 数とキー・値キャッシュはいずれも依然として系列長 n に伴って増大する。しかし、100,000 トークンの文書を処理する場合、大半のトークンは主に局所的な文脈に依存し、長距離の検索が必要になるのは時折生じる参照に限られる。

アテンションの計算コストを $O(n)$ に削減する主要なアーキテクチャ上の戦略は 2 つある。

1. **スライディングウィンドウアテンション** (Child et al. 2019; Beltagy et al. 2020) は、各クエリの参照先を直近 w 個のキーに制限することで、1 層あたりの計算量を $O(nw)$ に削減し、キー・値キャッシュを w トークンに抑える。
2. **線形アテンション**（linear attention、(Katharopoulos et al. 2020)）は、ソフトマックスカーネル $\exp(\mathbf{q}^\top \mathbf{k} / \sqrt{d_h})$ を積に分解されたカーネル $\phi(\mathbf{q})^\top \phi(\mathbf{k})$ に置き換えることで、キーと値を固定サイズ $d_h \times d_v$ の再帰状態行列に累積できるようにする。

いずれの手法も、密なソフトマックスアテンションとは異なる関数を計算する。本節では、スライディングウィンドウが 1 ホップの到達範囲と引き換えにどのようにタイルスパース性を得るかを分析し、続いて線形アテンションの並列形式・再帰形式・チャンク形式を導出したうえで、分解された特徴マップがソフトマックスのような先鋭化の温度をもたない理由を示す。

10.6.1 スライディングウィンドウアテンション

10.6.1.1 帯マスク

幅 w の因果的な**スライディングウィンドウアテンション**では、クエリ i はキー j に対し、$j \le i$ かつ $i - j < w$ の場合に限りアテンションを向ける。因果性と局所性を組み合わせたこの条件により、$n(n+1)/2$ 個の有効ペアからなる下三角領域は、高々 nw 個のペアからなる対角帯に置き換わる。

```
def window_mask(n, w):
    """Causal band: query i may attend to keys i - w + 1, ..., i."""
    i = jnp.arange(n)
    return d2l.causal_mask(n) & (i[:, None] - i[None, :] < w)
```

10.6.1.2 深さに伴う受容野の拡大

畳み込みを積み重ねると深さに伴って受容野が拡大するのと同様に（第 6 章）、L 層の幅 w のスライディングウィンドウ層を積み重ねると、情報は 1 層あたり $w-1$ ステップ後方へ伝播でき、理論的受容野（theoretical receptive field）は次の式で与えられる。

$$1 + L(w-1) \tag{10.29}$$

これが到達可能なトークン数である。たとえば、Mistral 7B v0.1 (Jiang et al. 2023) は $w = 4{,}096$ を $L = 32$ 層にわたって用いており、最大 131,041 トークンをカバーする。しかし、理論上の到達範囲と直接的な検索は異なる。完全な自己アテンションが任意の 2 トークンを 1 ホップで結ぶのに対し、ウィンドウの積層では層をまたぐ $\lceil \Delta/(w-1) \rceil$ 回の逐次ホップを経てようやく距離 Δ 先の事実を中継する。さらに、中継される事実はホップのたびに希釈される。最小の例を Figure 10.11 に示す。一様なウィンドウ平均（$w = 4$、$L = 4$）のもとでは、境界 $\Delta = 12$ にあるトークンは単一の 4 ホップ経路を通って重み $(1/4)^4 = 1/256$ で到達するが、単一の大域層ではこれが $1/13$ となる。

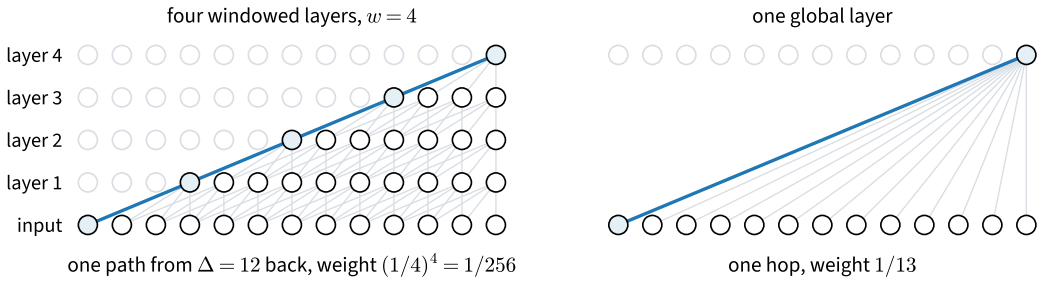


Figure 10.11: 積層ウィンドウと単一の大域層の比較。左：幅 $w = 4$ のウィンドウ層を 4 層重ねた場合、右上の出力は過去の位置からなる錐状の領域に依存し、$\Delta = 12$ 位置前の入力は 4 ホップの単一経路（青）のみを通って到達するため、一様なアテンション重みのもとでの重みは $(1/4)^4 = 1/256$ となる。右：単一の大域層は同じ 2 つの位置を 1 ホップ、重み $1/13$ で結ぶ。

10.6.1.3 大域層

局所ウィンドウによるメモリと計算量の削減効果を大部分維持しつつ、1 ホップの長距離検索を保つため、現代の長文脈（long-context）アーキテクチャは r 個のウィンドウ層と 1 つの完全な**大域**（global）アテンション層を交互に配置する。たとえば Gemma 3 は、大域層 1 つにつき $r = 5$ 個のウィンドウ層（$w = 1{,}024$）を用いている (Gemma Team 2025)。生成時、ウィンドウ層は直近 $\min(n, w)$ 個のキーと値のみをキャッシュするため（11.3 節）、この交互配置により KV キャッシュ全体の大きさは次の比率に削減される。

$$\frac{r\,\min(n, w) + n}{(r+1)\,n} \xrightarrow{\;n \gg w\;} \frac{1}{r+1}$$

これは全層を大域層とした構成に対する比率であり、$\approx 27\%$（$n = 8{,}192$ のとき）、漸近的には $1/6$（$r = 5, w = 1{,}024$ のとき）となる。

10.6.1.4 融合カーネルにおけるタイル

密な $n \times n$ 行列積の内部で `window_mask` を適用しても、依然として n^2 個すべてのスコアが計算されてしまう。$O(nw)$ の時間とメモリを実現するには、有効な対角帯のみを評価する実装が必要である。たとえば、クエリをサイズ w のブロックにまとめ、現在のキーブロックと直前のキーブロックのみにアテンションを向ければよい（合計 $2nw$ 個のスコア）。

```
def windowed_attention(Q, K, V, w):
    """Causal sliding-window attention in O(nw) time and memory."""
    n, d_h = Q.shape
    assert n % w == 0, 'blocks must tile the sequence exactly'
    Qb, Kb, Vb = (X.reshape(n // w, w, -1) for X in (Q, K, V))
    # Each query block attends to its own key block and the one before it
    prev = lambda X: jnp.concatenate([jnp.zeros_like(X[:1]), X[:-1]])
    Kb = jnp.concatenate([prev(Kb), Kb], 1)
    Vb = jnp.concatenate([prev(Vb), Vb], 1)
    i = jnp.arange(w)[:, None]                          # Query in its block
    j = jnp.arange(-w, w)[None, :]                      # Key offset from it
    first = jnp.arange(n // w)[:, None, None] == 0
    mask = (j <= i) & (i - j < w) & ~(first & (j < 0))  # No keys before 0
    scores = Qb @ Kb.swapaxes(-1, -2) / math.sqrt(d_h)
    return (d2l.masked_softmax(scores, mask) @ Vb).reshape(n, -1)
```

FlashAttention や FlexAttention (Dong et al. 2025) のようなタイル化 GPU カーネルでは、計算は $b \times b$ のタイル単位（FlexAttention のデフォルトでは $b = 128$）で進み、タイル内の b^2 個の要素が**すべて**マスクされている場合に限りそのタイルがスキップされる。$n = 2{,}048$ かつ $b = w = 128$ のとき、因果マスクは $T(T+1)/2$ 個（全 T^2 タイル中 53.1%、ただし $T = n/b = 16$）を訪問するのに対し、スライディングウィンドウは各タイル行の対角タイルとその左隣のみを訪問する（$2T - 1 = 31$ タイル、すなわち 12.1%、Figure 10.12）。

```
def tiles_visited(mask, b):
    """Fraction of b-by-b tiles of a mask that hold at least one True."""
    T = mask.shape[0] // b
    return float(mask.reshape(T, b, T, b).any(axis=(1, 3)).mean())

n, b, w = 2048, 128, 128
print(f'causal:  {tiles_visited(d2l.causal_mask(n), b):.1%} of tiles')
print(f'window:  {tiles_visited(window_mask(n, w), b):.1%} of tiles')
```

```
causal:  53.1% of tiles
window:  12.1% of tiles
```

`windowed_attention` とフレームワークの融合スライディングウィンドウカーネルはいずれも、密行列にマスクを適用する定義と float32 の丸め誤差の範囲内で一致する。

融合カーネルには、`dot_product_attention` に `local_window_size=(w - 1, 0)` を指定して用いる。

```
n, w, d_h = 2048, 128, 64
Q, K, V = (jax.random.normal(k, (n, d_h))
           for k in jax.random.split(d2l.get_key(), 3))
with jax.default_matmul_precision('highest'):
    reference = d2l.masked_softmax(Q @ K.T / math.sqrt(d_h),
```

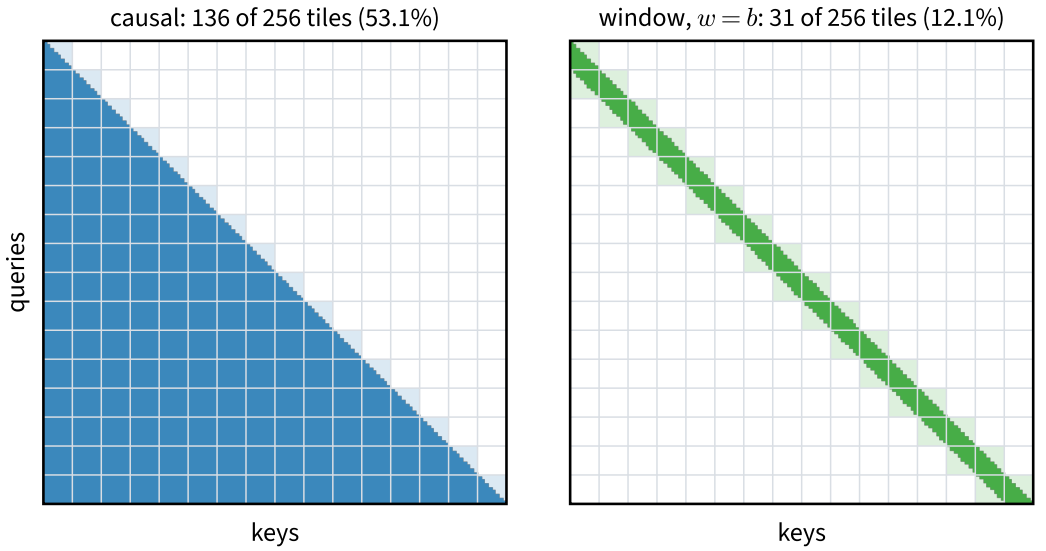


Figure 10.12: 融合カーネルが訪問するタイル（タイル行数 $T = 16$ の場合）。濃い要素は許可されたスコアを表し、正方形内の薄い残余部分はマスク境界で切断されるもののカーネルが訪問せざるを得ないタイルを示す。因果マスク（左）は $T(T+1)/2 = 136$ 個のタイルを訪問し、切断が生じるのは対角上の 16 個のみである。幅 $w = b$ のウィンドウ（右）は各行の対角タイルとその左隣、計 $2T - 1 = 31$ 個のタイルを訪問し、そのすべてがマスク境界で切断される。

```
                                    window_mask(n, w)) @ V
    # The window as an argument: w - 1 keys back, none ahead. The default
    # XLA implementation runs in float32; cuDNN's needs 16-bit inputs
    fused = jax.nn.dot_product_attention(
        Q[None, :, None], K[None, :, None], V[None, :, None],
        is_causal=True, local_window_size=(w - 1, 0))
    for name, O in (('blocked', windowed_attention(Q, K, V, w)),
                    ('fused kernel', fused[0, :, 0])):
        print(f'{name:>12}: maximum deviation '
              f'{float(jnp.abs(O - reference).max()):.2e}')
```

```
     blocked: maximum deviation 2.98e-07
fused kernel: maximum deviation 0.00e+00
```

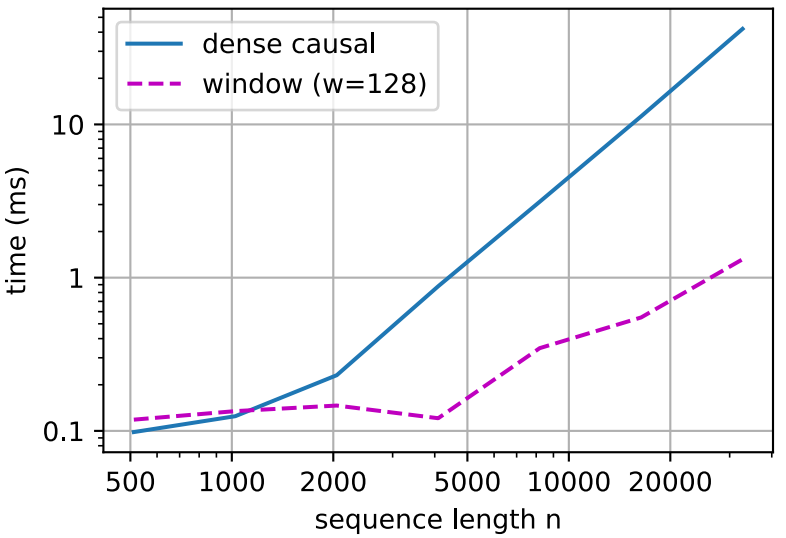
10.6.1.5 ウィンドウカーネルと密カーネルの比較

タイル行数を $T = n/128$ とすると、有効な因果タイル数と有効なウィンドウタイル数（$w = b = 128$）の比は $T(T+1)/(2(2T-1)) \approx T/4$ となり、理論上のタイル削減率は $64\times$（$n = 32{,}768$、すなわち $T = 256$ のとき）と予測される。同一の bfloat16 融合カーネル内で、両方のマスクを $n \in [512, 32{,}768]$ にわたってベンチマークしよう（ヘッド数 $H = 32$、$d_h = 64$）。

```
# cuDNN takes (batch, length, heads, width); it skips the tiles outside the
# band only together with is_causal=True
w = b = 128  # Window width; tile size of the kernel
```

```
masks = {'dense causal': {},
         f'window (w={w})': {'local_window_size': (w - 1, 0)}}
kernels = {name: jax.jit(lambda X, kw=kw: jax.nn.dot_product_attention(
    X, X, X, is_causal=True, implementation='cudnn', **kw))
    for name, kw in masks.items()}
H, lengths = 32, [512, 1024, 2048, 4096, 8192, 16384, 32768]
times = {name: [] for name in masks}
for n in lengths:
    X = jax.random.normal(d2l.get_key(), (1, n, H, d_h), dtype=jnp.bfloat16)
    for name, f in kernels.items():
        times[name].append(d2l.Benchmark(lambda: f(X)).time * 1e3)
    T = n // b  # Rows of b x b tiles; causal T(T+1)/2, window 2T - 1
    dense, window = (t[-1] for t in times.values())
    print(f'n = {n:6,}: window {dense / window:4.1f}x faster, '
          f'tile count predicts {T * (T + 1) / 2 / (2 * T - 1):4.1f}x')
d2l.plot(lengths, list(times.values()), 'sequence length n', 'time (ms)',
         legend=list(times), xscale='log', yscale='log', figsize=(4.5, 3))
```

```
n =    512: window  0.8x faster, tile count predicts  1.4x
n =  1,024: window  0.9x faster, tile count predicts  2.4x
n =  2,048: window  1.6x faster, tile count predicts  4.4x
n =  4,096: window  7.3x faster, tile count predicts  8.4x
n =  8,192: window  9.0x faster, tile count predicts 16.4x
n = 16,384: window 20.5x faster, tile count predicts 32.4x
n = 32,768: window 31.9x faster, tile count predicts 64.4x
```



$n \approx 2{,}048$ を超えると、密な因果アテンションの実行時間が二次関数的に増大するのに対し、ウィンドウアテンションの実行時間は線形に増大し、$> 30\times$ の高速化（$n = 32{,}768$ のとき）に達する。これはタイル数から予測される $64\times$ の 1/2 から 3/4 の間にあたる。タイル数による比較は訪問されるすべてのタイルを同一の計算量とみなすため、得られる高速化の上界ではあっても推定値ではない。

cuDNN はスコア行列のタイル分割方法を公開していないため、ここでは $b = 128$ と仮定してい

る。

10.6.2 線形アテンション

10.6.2.1 スコアのカーネル化

どのペアのスコアを計算するかを制限する代わりに、**線形アテンション** (Katharopoulos et al. 2020) は指数類似度 $\exp(\mathbf{q}^\top \mathbf{k}/\sqrt{d_h})$ を積に分解されたカーネル $K(\mathbf{q},\mathbf{k}) = \phi(\mathbf{q})^\top \phi(\mathbf{k})$ に置き換える。ここで $\phi : \mathbb{R}^{d_h} \to \mathbb{R}_+^{d_h}$ は要素ごとの非負の特徴マップである。行列積の結合法則により、次が成り立つ。

$$\left(\phi(\mathbf{Q})\,\phi(\mathbf{K})^\top\right)\mathbf{V} = \phi(\mathbf{Q})\left(\phi(\mathbf{K})^\top \mathbf{V}\right),$$

これにより、$n \times n$ 行列 $\phi(\mathbf{Q})\phi(\mathbf{K})^\top$ の構築を避け、代わりに $d_h \times d_v$ の要約行列を計算すれば済む。因果マスク（$s \le t$）のもとで、分子と分母の和から $\phi(\mathbf{q}_t)^\top$ をくくり出すと次式を得る。

$$\begin{aligned}\mathbf{o}_t^\top &= \frac{\sum_{s \le t} \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_s)\,\mathbf{v}_s^\top}{\sum_{s \le t} \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_s)} = \frac{\phi(\mathbf{q}_t)^\top \mathbf{S}_t}{\phi(\mathbf{q}_t)^\top \mathbf{z}_t},\\ \mathbf{S}_t &= \sum_{s \le t} \phi(\mathbf{k}_s)\mathbf{v}_s^\top, \qquad \mathbf{z}_t = \sum_{s \le t} \phi(\mathbf{k}_s).\end{aligned} \tag{10.30}$$

Katharopoulos et al. (2020) に従い、要素ごとに $\phi(x) = \mathrm{elu}(x) + 1 > 0$ と選ぶ（float32 のアンダーフローを防ぐため、分母には小さな $\epsilon = 10^{-6}$ を加える）。プレフィックス和 $(\mathbf{S}_t, \mathbf{z}_t)$ は次の線形漸化式（linear recurrence）を満たす。

$$\begin{aligned}\mathbf{S}_t &= \mathbf{S}_{t-1} + \phi(\mathbf{k}_t)\,\mathbf{v}_t^\top,\\ \mathbf{z}_t &= \mathbf{z}_{t-1} + \phi(\mathbf{k}_t),\end{aligned} \tag{10.31}$$

これにより、因果アテンションは、外積（outer product）によって更新される $d_h \times d_v$ の連想メモリ行列を隠れ状態にもつ RNN、すなわち**高速重みプログラマ**（fast weight programmer、(Schlag et al. 2021)）へと変換される。Equation 10.30 は、累積和（`cumsum`）を用いて並列に評価することも、$O(d_h d_v)$ のメモリで 1 ステップずつ再帰的に評価することもできる。

```
def phi(X):
    return jax.nn.elu(X) + 1

def linear_attention_parallel(Q, K, V, eps=1e-6):
    """Training form: all outputs at once via cumulative sums."""
    Qp, Kp = phi(Q), phi(K)
    S = jnp.cumsum(Kp[:, :, None] * V[:, None, :], axis=0)  # (n, d_h, d_v)
    z = jnp.cumsum(Kp, axis=0)                              # (n, d_h)
    den = (Qp * z).sum(-1, keepdims=True) + eps
    return (Qp[:, :, None] * S).sum(1) / den

def linear_attention_recurrent(Q, K, V, eps=1e-6):
    """Generation form: a (d_h, d_v) state, updated token by token."""
    Qp, Kp = phi(Q), phi(K)
    def step(carry, qkv):
```

```
        S, z = carry
        q, k, v = qkv
        S = S + k[:, None] * v[None, :]          # Outer-product write
        z = z + k
        return (S, z), q @ S / (q @ z + eps)     # Query the memory
    init = (jnp.zeros((Q.shape[-1], V.shape[-1])), jnp.zeros(Q.shape[-1]))
    _, outputs = jax.lax.scan(step, init, (Qp, Kp, V))
    return outputs
```

10.6.2.2 チャンク単位の計算

GPU 上での訓練時、n 個すべての中間行列 $\mathbf{S}_t \in \mathbb{R}^{d_h \times d_v}$ を `linear_attention_parallel` で実体化すると $O(nd_hd_v)$ のメモリ（$n = 16{,}384, d_h = d_v = 64$ のときヘッドあたり 256 MiB）を消費する一方、`linear_attention_recurrent` は n 回の逐次的なベクトル演算ステップを実行するため GPU の行列演算コアを十分に活用できない。系列を長さ c のチャンクに分割する手法 (S. Yang, Wang, Shen, et al. 2024) は両者の利点を兼ね備える。位置 t_0 から始まりチャンク前の状態 $(\mathbf{S}, \mathbf{z})$ をもつチャンクに対しては、次式が成り立つ。

$$\mathbf{o}_t^\top = \frac{\phi(\mathbf{q}_t)^\top \mathbf{S} + \sum_{t_0 \le s \le t} \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_s)\, \mathbf{v}_s^\top}{\phi(\mathbf{q}_t)^\top \mathbf{z} + \sum_{t_0 \le s \le t} \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_s)}, \tag{10.32}$$

したがって、チャンク内のアテンション $(c \times c)$ とチャンク間の状態伝播はいずれも $O(nc(d_h + d_v) + nd_hd_v)$ FLOPs の密行列積を用い、保存するチャンク状態も n/c 個だけで済む。

```
def linear_attention_chunked(Q, K, V, c=64, eps=1e-6):
    """Training form in chunks: quadratic within, recurrent across."""
    n, d_h = Q.shape
    assert n % c == 0, 'chunks must tile the sequence exactly'
    Qp, Kp = phi(Q).reshape(n // c, c, d_h), phi(K).reshape(n // c, c, d_h)
    Vc = V.reshape(n // c, c, -1)
    # The state each chunk starts from: sums over all earlier chunks
    S = jnp.cumsum(Kp.swapaxes(-1, -2) @ Vc, axis=0)     # (n/c, d_h, d_v)
    z = jnp.cumsum(Kp.sum(1), axis=0)                    # (n/c, d_h)
    S = jnp.concatenate([jnp.zeros_like(S[:1]), S[:-1]])
    z = jnp.concatenate([jnp.zeros_like(z[:1]), z[:-1]])
    A = jnp.tril(Qp @ Kp.swapaxes(-1, -2))               # (n/c, c, c)
    num = Qp @ S + A @ Vc                                # Past + this chunk
    den = Qp @ z[..., None] + A.sum(-1, keepdims=True) + eps
    return (num / den).reshape(n, -1)
```

3 つの形式はいずれも float32 の丸め誤差の範囲内でまったく同じ出力を計算し、再帰状態は系列長によらずヘッドあたり一定の 16 KiB（float で $64 \times 64 + 64$ 個）しか占有しない。

```
n = 512
Q, K, V = (jax.random.normal(k, (n, d_h))
           for k in jax.random.split(d2l.get_key(), 3))
with jax.default_matmul_precision('highest'):
    recurrent = linear_attention_recurrent(Q, K, V)
    for name, f in (('parallel', linear_attention_parallel),
```

```
                    ('chunked', linear_attention_chunked)):
        err = jnp.abs(f(Q, K, V) - recurrent).max()
        print(f'{name:>8}: maximum deviation {float(err):.2e}')
print(f'recurrent state: {d_h}x{d_h} + {d_h} floats '
      f'= {(d_h * d_h + d_h) * 4 / 1024:.0f} KiB at any sequence length')
```

```
parallel: maximum deviation 2.38e-07
 chunked: maximum deviation 3.87e-07
recurrent state: 64x64 + 64 floats = 16 KiB at any sequence length
```

10.6.2.3 容量と鋭さ

任意個のキー・値ペアを $d_h \times d_v$ の状態行列に圧縮することには、2 つの本質的なコストが生じる。

1. **有限の連想容量：**$\phi(\mathbf{k}_s) \in \mathbb{R}_+^{d_h}$ は非負であるため、クエリがクロストークなしに保存された単一の値を取り出せるのは、キーの特徴ベクトル同士がほぼ直交する（台がほぼ互いに素である）場合に限られ、厳密な検索は高々 d_h 個のキーに制限される。
2. **温度による先鋭化の欠如：**ソフトマックスアテンションでは、クエリを $\beta > 1$ 倍にスケーリングするとスコア差が指数的に増幅され（$e^{\beta\Delta}$）、$\beta \to \infty$ で分布は one-hot 参照へと収束する。一方、$\phi(x) = \mathrm{elu}(x) + 1$ を用いる線形アテンションでは、$\mathbf{q}$ を大きな β でスケーリングすると $\phi(\beta\mathbf{q}) \approx \beta\,\mathrm{relu}(\mathbf{q})$ となるため、スケール因子 β が分子と分母で相殺される（演習問題 2）。

```
kq, kk = jax.random.split(d2l.get_key())
Q, K = jax.random.normal(kq, (1000, d_h)), jax.random.normal(kk, (8, d_h))
for beta in (1, 4, 16):  # Scale the queries up; 8 keys: uniform is 0.125
    soft = jax.nn.softmax(beta * Q @ K.T / math.sqrt(d_h), axis=-1)
    lin = phi(beta * Q) @ phi(K).T
    lin = lin / lin.sum(-1, keepdims=True)
    print(f'queries x{beta:<2}: mean largest weight, softmax '
          f'{soft.max(-1).mean():.2f}, elu+1 {lin.max(-1).mean():.2f}')
```

```
queries x1 : mean largest weight, softmax 0.35, elu+1 0.14
queries x4 : mean largest weight, softmax 0.74, elu+1 0.15
queries x16: mean largest weight, softmax 0.93, elu+1 0.15
```

10.6.2.4 各機構における想起

これら 2 つの限界とウィンドウの限界は、考えられる最も単純な参照タスクに現れる。N 個の幅 $d_h = 64$ のランダムなキー・値ペアを保存し、保存されたキーの 1 つをクエリとして与え、出力が他のどの保存された値よりも正しい値と大きな内積をもつときにヒットと数える。このクエリに 4 つの機構で応答する。すなわち、密なソフトマックスアテンション、直近 $w = 16$ 個のペアのみを見るウィンドウ、elu+1 特徴マップを用いた線形アテンション、そして恒等特徴マップを用い正規化項をもたない線形アテンションである**外積状態**（outer-product state）$\mathbf{q}^\top \sum_i \mathbf{k}_i \mathbf{v}_i^\top$ である。最後の機構は特徴マップの影響を切り離すことで、$d_h \times d_v$ の状態そのものがもつ容量を浮き彫りにする。

```
def recall(N, w=16, trials=1000):
    """Look up one of N stored pairs by its key; fraction of hits per rule."""
```

```
    key_K, key_V, key_t = jax.random.split(d2l.get_key(), 3)
    K, V = (jax.random.normal(k, (trials, N, d_h)) for k in (key_K, key_V))
    target = jax.random.randint(key_t, (trials,), 0, N)
    q = K[jnp.arange(trials), target][:, None]         # (trials, 1, d_h)
    scores = q @ K.swapaxes(1, 2) / math.sqrt(d_h)     # (trials, 1, N)
    recent = jnp.arange(N) >= N - w                    # The last w pairs
    lin = phi(q) @ phi(K).swapaxes(1, 2)
    outputs = {'softmax': jax.nn.softmax(scores, axis=-1) @ V,
               f'window (w={w})': d2l.masked_softmax(scores, recent) @ V,
               'elu+1': lin / lin.sum(-1, keepdims=True) @ V,
               'outer-product state': q @ (K.swapaxes(1, 2) @ V)}
    # A hit: of the N stored values, the target is nearest to the output
    return {name: float(((O @ V.swapaxes(1, 2)).argmax(-1)[:, 0] == target)
                        .mean()) for name, O in outputs.items()}

for N in (16, 64, 256, 1024):
    print(f'N = {N:4}: ' + ', '.join(f'{name} {hit:.2f}'
                                     for name, hit in recall(N).items()))
```

```
N =   16: softmax 1.00, window (w=16) 1.00, elu+1 0.21, outer-product state 1.00
N =   64: softmax 1.00, window (w=16) 0.25, elu+1 0.03, outer-product state 0.99
N =  256: softmax 1.00, window (w=16) 0.07, elu+1 0.01, outer-product state 0.66
N = 1024: softmax 1.00, window (w=16) 0.02, elu+1 0.00, outer-product state 0.09
```

ソフトマックスアテンションはあらゆる N で目的のペアを見つけ出す。ウィンドウは目的のペアが直近 w 個に含まれる場合にのみ見つけられるため、その想起率は約 w/N となり、$N = 16$ では完全、$N = 64$ では 1/4、$N = 1{,}024$ では数パーセントにとどまる。さらに際立つのが elu+1 線形アテンションであり、$N = 16$、すなわち保存すべきペア数の 4 倍の行数を状態がもつ場合でさえ失敗する。重みを鋭くできないため、出力は保存されたすべての値の平均に近くなり、正しい値がほとんど際立たないのである。

外積状態ははるかに良好に機能し、容量の限界を鮮明に示す。クエリ $\mathbf{q} = \mathbf{k}_j$ に対する出力は、$\|\mathbf{k}_j\|^2 \mathbf{v}_j \approx d_h \mathbf{v}_j$ に他のペアからのクロストーク $\sum_{i \neq j} (\mathbf{k}_j^\top \mathbf{k}_i)\, \mathbf{v}_i$ を加えたものになる。各クロストーク係数の大きさは約 $\sqrt{d_h}$ であるため、信号に対するノイズの比は約 $\sqrt{(N-1)/d_h}$ となる。想起率は $N \approx d_h$ まではほぼ完全であるが、$N = 4d_h$ で約 2/3 に低下し、$N = 16d_h$ では崩壊する。ソフトマックスアテンションは N 個すべてのペアを保持してあらゆる N で想起できるのに対し、固定サイズの状態は N がその幅の付近にとどまるあいだしか想起できない。

10.6.3 機構の選び方

- **厳密な融合アテンション**は短〜中程度の文脈におけるデフォルトであり、$O(n)$ の作業メモリで厳密な 1 ホップ検索を実現する。
- **スライディングウィンドウ層と大域層の交互配置**は長文脈 Transformer（例：Gemma 3）の標準的な構成であり、局所層での KV キャッシュメモリとタイルあたりの FLOP 数を削減しつつ、周期的な大域層で 1 ホップの大域ルーティングを維持する。
- **線形アテンション**で厳密な連想想起（associative recall）を扱うには、データ依存の減衰ゲート（古い項目を忘却できるようにするため。演習問題 4 および第 12 章を参照）と、完全なソフトマックス層とのハイブリッドな交互配置（例：MiniMax-01、(MiniMax 2025)）が必要になる (Arora, Eyuboglu, Zhang, et al. 2024; S. Yang, Wang, Shen, et al. 2024)。

10.6.4 まとめ

スライディングウィンドウアテンションは、各クエリの参照先を直近 w 個のキーに制限することで、演算量を $O(nw)$、キャッシュメモリを $O(w)$ に削減しつつ、有効受容野を $1 + L(w-1)$ トークン（L 層積層時）へと拡大する。現代の長文脈モデルは、1 ホップ検索を維持するためにこれらの局所層と周期的な大域アテンション層を交互に配置する。GPU 上では、タイル化カーネルが完全にマスクされた $b \times b$ のスコアタイルをスキップすることで、境界でのマスク処理のオーバーヘッドを除きこの高速化を実現する。一方、線形アテンションは代数的なアプローチをとり、指数スコアを積に分解された類似度カーネル $\phi(\mathbf{q})^\top \phi(\mathbf{k})$ に置き換えることで、因果アテンションを、並列チャンクで訓練可能な $O(n)$ の漸化式（固定サイズ $d_h \times d_v$ の行列状態をもつ）へと帰着させる。非負の特徴量をもつ固定サイズの状態では、クエリのスケーリングによって重みを鋭くすることも、クロストークなしに $O(d_h)$ 個を超える直交キーを保持することもできないため、実用的な線形アテンションのアーキテクチャはデータ依存の減衰ゲートとハイブリッドなソフトマックス層を組み込んでいる。

10.6.5 演習問題

1. [code] **融合カーネルのタイル**。融合カーネルは $b \times b$ 個のスコアからなるタイル単位で動作し、マスクがそのタイル全体を除外する場合に限りタイルをスキップする。$T = n/b$ とする。
   1. 因果マスクが $T(T+1)/2$ 個のタイル（全 T^2 タイル中）を残すこと、および幅 $w = kb$（整数 $k < T$）のウィンドウが $T(k+1) - k(k+1)/2$ 個のタイルを残すことを示せ。両方の個数を `tiles_visited` で確認せよ。w が b の倍数でないとき、個数はどのように変化するか。
   2. 複数の文書が長さ $n = 2048$ トークンの 1 つの系列にパックされており、10.2 節のマスクと同様に、マスクが因果性と「同一文書」の条件を組み合わせているとする。$b = 128$ で `tiles_visited` を用い、512 トークンの 4 つの文書を、1,000、600、300、148 トークンの文書と比較せよ。文書の境界がタイルの境界と一致しない場合により多くのタイルを要する理由を説明せよ。
2. [code] **特徴マップ**。`phi` で用いた特徴マップ $\phi(x) = \mathrm{elu}(x) + 1$ は数ある選択肢の 1 つにすぎない。いずれも要素ごとに適用される $\phi(x) = \mathrm{relu}(x)$ および $\phi(x) = \exp(x)$ と、$n = 512$ 個の幅 `d_h` のランダムなクエリとキー上で比較せよ。
   1. 各マップについて、要素が次式で与えられる因果線形アテンションの重み行列を計算せよ。
   $$\frac{\phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_s)}{\sum_{s' \le t} \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_{s'})}, \qquad s \le t,$$
   さらに、因果ソフトマックス重み行列 `d2l.masked_softmax(Q @ K.T / math.sqrt(d_h), d2l.causal_mask(n))` との最大絶対差、および各行のエントロピーの平均を測定せよ。
   2. $\mathbf{q}_t$ とキー $\mathbf{k}_1, \dots, \mathbf{k}_t$ がどのような条件を満たすとき、ReLU マップは未定義の行を生じるか。すべての要素が独立な標準正規分布に従うとき、第 t 行でこの事象が起こる確率を導出し、$t = 1$ について幅 64 と幅 4 のそれぞれで値を求めよ。
   3. 本節ではクエリを $\beta \in \{1, 4, 16\}$ 倍して最大の重みを観察した。ReLU マップと指数マップに対して同様の実験を行い、代わりに各行の平均エントロピーを測定せよ。3 つのマップのそれぞれについて $\beta \to \infty$ としたときの正規化重みの極限を導出し、ソフトマックスが one-hot になる状況でいずれも one-hot にならない理由を説明せよ。
3. **厳密な特徴マップ**。●ソフトマックスカーネル $K(\mathbf{q}, \mathbf{k}) = \exp(\mathbf{q}^\top \mathbf{k} / \sqrt{d_h})$ も積に分解できるが、そのためには無限個の特徴量が必要になる。
   1. K をテイラー級数に展開し、p 次の項が $\phi_p(\mathbf{q})^\top \phi_p(\mathbf{k})$ に等しいことを示せ（ここで ϕ_p は次元 d_h^p の特徴マップであり、引数の p 重テンソル積から構成される）。

2. 級数を 2 次の項までで打ち切れ。打ち切ったカーネルが任意の $\mathbf{q}$ と $\mathbf{k}$ の組に対して正であり、したがって Equation 10.30 の正規化項が 0 になり得ないことを示したうえで、ϕ_0、ϕ_1、ϕ_2 を連結した特徴マップの次元を求めよ。$d_h = 64$ かつ値の幅が 64 のとき、単精度における状態 $(\mathbf{S}, \mathbf{z})$ のサイズと、ソフトマックスヘッドのキー・値キャッシュが同じサイズに達する系列長 n を計算せよ。この種の打ち切りは Based (Arora, Eyuboglu, Zhang, et al. 2024) の特徴マップとして用いられている。
3. $n \times n$ 行列（要素 $K(\mathbf{x}_i, \mathbf{x}_j)$）が相異なる $\mathbf{x}_1, \dots, \mathbf{x}_n$ に対して正定値であることは既知としてよい。クエリとキーが一致するとき、次元 $D < n$ のいかなる特徴マップもこれらの点上で K を厳密には再現できないことを示せ。

4. [code] **減衰する状態。**●Equation 10.31 の状態にスカラー減衰 $\gamma \in (0, 1)$ を導入する。

$$\mathbf{S}_t = \gamma\, \mathbf{S}_{t-1} + \phi(\mathbf{k}_t)\, \mathbf{v}_t^\top, \qquad \mathbf{z}_t = \gamma\, \mathbf{z}_{t-1} + \phi(\mathbf{k}_t). \tag{10.33}$$

1. 並列形式において、位置 s が位置 t の出力に与える寄与が γ^{t-s} で重み付けされることを示し、`linear_attention_parallel` の累積和を γ のべき乗でリスケーリングすることでこれを実装せよ。
2. `linear_attention_recurrent` に減衰を追加し、$n = 512$ かつ $\gamma = 0.99$ で 2 つの形式が一致することを検証せよ。
3. このリスケーリングは累積和の前に位置 s に γ^{-s} を掛ける。単精度でオーバーフローが生じるのはどのような n のときか。$\gamma = 0.9$ と $\gamma = 0.5$ のそれぞれについて求めよ。
4. `linear_attention_chunked` に減衰を追加せよ。すなわち、各位置をそのチャンクの先頭を基準としてリスケーリングし、減衰付きの漸化式でチャンクから次のチャンクへと状態を引き継げ。γ^{-c} より大きな因子が現れないことを示したうえで、$\gamma = 0.5$、$n = 512$、$c = 64$ で実行し、再帰形式と比較せよ。
5. 重み γ^{t-s} が 0.01 を下回るラグ（$\gamma = 0.99$ のとき）を計算せよ。あるタスクでは、2000 位置前にある 1 つのトークンを想起する必要があり、そのあいだのトークンはすべてフィラー（埋め草）であるとする。固定された γ ではそのトークンとフィラーの双方にうまく対処できない理由、および各トークン自身の入力から計算される減衰であれば代わりに何ができるかを説明せよ。

10.7 アテンション回路

10.1 節で述べたように、アテンションのヒートマップは層が何を計算しているかについての仮説であり、証明ではない。ヒートマップが示すのはヘッドがどこを見るかという点であり、それは計算の半分にすぎない。ヘッドが実際に出力へ何を書き込むかは、その値射影と出力射影に依存する。あるトークンに鋭くアテンションを向けるヘッドであっても、そのトークンの同一性をコピーする場合もあれば、部分空間の単一の特徴量を抽出する場合や、無視できるほど小さなベクトルしか寄与しない場合もある。

訓練済みのアテンションヘッドが何を計算するかを理解するため、本節では Elhage et al. (2021) による機構的回路（mechanistic circuit）の定式化を、アテンションのみの言語モデル `TinyCharLM`（10.4 節）に適用する。各ヘッドは、**残差ストリーム**（residual stream）上のランク d_h の 2 つの双線形・線形作用素へと分解される。すなわち、どの位置にアテンションを向けるかを決める **QK 回路**（QK circuit）と、アテンションを向けられたベクトルが出力ロジットをどう変化させるかを決める **OV 回路**（OV circuit）である。この分解を用いて、1 層のモデルがバイグラムやスキップトライグラム（単一トークンのコピーを含む）は表現できてもコンテキスト内のパターン補完（[A][B] … [A] → [B]）は実行できない理由を示し、対照的に 2 層のモデルでは**直前トークンヘッド**（previous-token head）と**照合コピーヘッド**（match-and-copy head）を合成して**誘導ヘッド**（induction head）を形成できることを示す (Olsson et al. 2022)。最後に、重みの検査と因果的ヘッドアブレーションによってこの回路仮説を検証する。

10.7.1 回路と残差ストリーム

10.7.1.1 QK 回路と OV 回路

`TinyCharLM`（10.4.5.1 節）では、トークン埋め込み $\mathbf{e}_{x_i} \in \mathbb{R}^d$（$d = 128$）が N 個の残差ブロックを通過する。各ブロックは $H = 4$ 個の因果アテンションヘッド（幅 $d_h = d/H = 32$）からなり、MLP、正規化、バイアス項を持たず、その後に重み共有された逆埋め込み射影（unembedding projection）$\mathbf{E} \in \mathbb{R}^{|\mathcal{V}|\times d}$ が続く。

トークンベクトルを列ベクトル $\mathbf{h}_j \in \mathbb{R}^d$ と書くと、各ヘッドは $\mathbf{h}_j$ を $\mathbf{W}_v \in \mathbb{R}^{d_h\times d}$ によって射影し、重み付き和 $\sum_{j\le i} \alpha_{ij}\mathbf{W}_v\mathbf{h}_j$ を作る。ブロックの出力行列を H 個の列スライス $\mathbf{W}_o \in \mathbb{R}^{d\times d_h}$（ヘッドごとに 1 つ）に分割すると、連結と射影のステップは各ヘッドの独立な寄与の和として表せる。

$$\mathbf{W}_o \sum_{j\le i} \alpha_{ij}\,\mathbf{W}_v\mathbf{h}_j = \sum_{j\le i} \alpha_{ij}\,\mathbf{W}_o\mathbf{W}_v\,\mathbf{h}_j.$$

同様に、クエリ位置 i とキー位置 j の間のソフトマックス前のスコアは $(\mathbf{W}_q\mathbf{h}_i)^\top(\mathbf{W}_k\mathbf{h}_j)/\sqrt{d_h}$（ただし $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{d_h\times d}$）である。したがって、各ヘッドは 2 つの $d \times d$ 行列（ランクは高々 d_h）によって完全に特徴づけられる (Elhage et al. 2021)。

$$s_{ij} = \frac{\mathbf{h}_i^\top \mathbf{W}_{\mathrm{QK}}\,\mathbf{h}_j}{\sqrt{d_h}}, \qquad \mathbf{W}_{\mathrm{QK}} = \mathbf{W}_q^\top\mathbf{W}_k, \qquad \mathbf{W}_{\mathrm{OV}} = \mathbf{W}_o\mathbf{W}_v. \tag{10.34}$$

- **QK 回路 $\mathbf{W}_{\mathrm{QK}}$**（RoPE のもとでは $\mathbf{W}_q^\top\mathbf{R}_{j-i}\mathbf{W}_k$、10.4.3 節）は、ヘッドが**どこに**アテンションを向けるかを決める。
- **OV 回路 $\mathbf{W}_{\mathrm{OV}}$** は、参照元の位置にアテンションが向けられたとき、ヘッドが参照先の位置**どの**ベクトルを書き込むかを決める。

個々の因子ではなく積 $\mathbf{W}_{\mathrm{QK}}$ と $\mathbf{W}_{\mathrm{OV}}$ を調べることで、ゲージ自由度（gauge freedom）が取り除かれ（$\mathbf{W}_o\mathbf{G}^{-1}$ と $\mathbf{G}\mathbf{W}_v$ からはまったく同じ $\mathbf{W}_{\mathrm{OV}}$ が任意の可逆行列 $\mathbf{G} \in \mathbb{R}^{d_h\times d_h}$ について得られる）、ランク d_h の部分空間ボトルネックが明示的になる。

10.7.1.2 層をまたぐ残差表現

$\mathbf{h}_i^{(0)} = \mathbf{e}_{x_i}$ とおくと、ブロック $\ell \in \{1,\dots,N\}$ は位置 i の表現を加法的に更新する。

$$\mathbf{h}_i^{(\ell)} = \mathbf{h}_i^{(\ell-1)} + \sum_{m=1}^{H}\sum_{j\le i} \alpha_{ij}^{\ell m}\,\mathbf{W}_{\mathrm{OV}}^{\ell m}\,\mathbf{h}_j^{(\ell-1)}, \tag{10.35}$$

そして、重み共有された出力ヘッドがロジット $\mathbf{z}_i = \mathbf{E}\,\mathbf{h}_i^{(N)}$ を出力する。系列 $\mathbf{h}_i^{(0)},\dots,\mathbf{h}_i^{(N)}$ は位置 i における**残差ストリーム**（Figure 10.13）であり、すべてのヘッドが加法的に読み書きする共有通信チャネルとなっている。

10.7.1.3 バイグラム・スキップトライグラムと 1 層の限界

Equation 10.35 を深さ方向に展開すると、N 層のアテンション層がどのような関数を表現できるかが明らかになる。

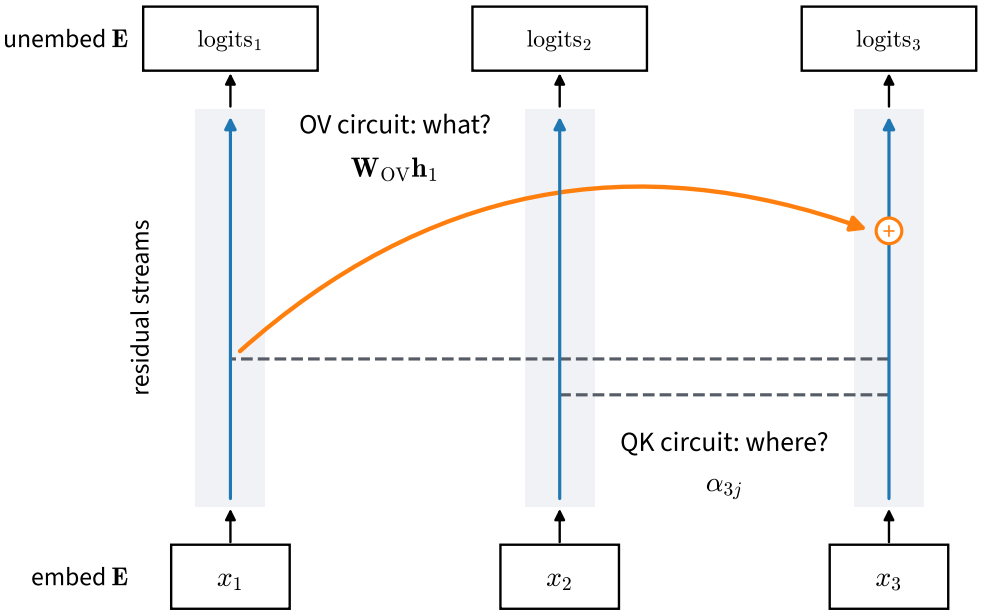


Figure 10.13: 残差ストリーム。各位置は埋め込みから出力ロジットへとベクトルを運ぶ。QK 回路がアテンション重み α_{3j} を決定し、OV 回路がアテンションを向けられた各ベクトルから参照先ストリームへの寄与を決定する。

- **0 ブロック**（$N = 0$）：$\mathbf{z}_i = \mathbf{E}\,\mathbf{e}_{x_i}$ となり、対称かつ半正定値なグラム行列 $\mathbf{E}\mathbf{E}^\top$ を介して、制約付きのトークンバイグラムのロジットしかモデル化できない（演習問題 1）。
- **1 ブロック**（$N = 1$）：

$$\mathbf{z}_i = \mathbf{E}\,\mathbf{e}_{x_i} + \sum_{m=1}^{H} \sum_{j \le i} \alpha_{ij}^{m}\, \mathbf{E}\,\mathbf{W}_{\mathrm{OV}}^{m}\,\mathbf{e}_{x_j}. \tag{10.36}$$

  各ヘッドの項は $x_j \ldots x_i \to x_{\text{next}}$ という形の**スキップトライグラム**規則を実装する（たとえば、**“in”** から先行する **“keep”** にアテンションを向けて **“mind”** のスコアを押し上げたり、x_j 自体を押し上げて**コピー**したりする）。

- **2 ブロック**（$N = 2$）**と誘導ヘッド**：コンテキストに [A][B] ... [A] が含まれており、モデルがコンテキスト内参照によって [B] を予測しなければならないとする。1 ブロックのモデルでは、内容の照合によってこれを行うことはできない。[B] を位置 j からコピーするにはヘッドが位置 j にアテンションを向ける必要があるが、Equation 10.36 では位置 j のキーは $\mathbf{e}_{x_j} = \mathbf{e}_{[\mathrm{B}]}$ のみに依存し、その直前のトークン $x_{j-1} = [\mathrm{A}]$ については何も知らないからである。

  2 ブロックあれば、**K 合成**（K-composition）によってこの問題を解決できる（Figure 10.14、演習問題 2）。ブロック 1 では、**直前トークンヘッド**がオフセット -1 にアテンションを向け、$x_{j-1} = [\mathrm{A}]$ を残差ストリーム $\mathbf{h}_j^{(1)}$（位置 j）に書き込む。続いてブロック 2 では、**誘導ヘッド**が自身のクエリ [A] を $\mathbf{h}_j^{(1)}$ 内のそのキー特徴量と照合し、OV 回路を使って $x_j = [\mathrm{B}]$

を予測ロジットへとコピーする。

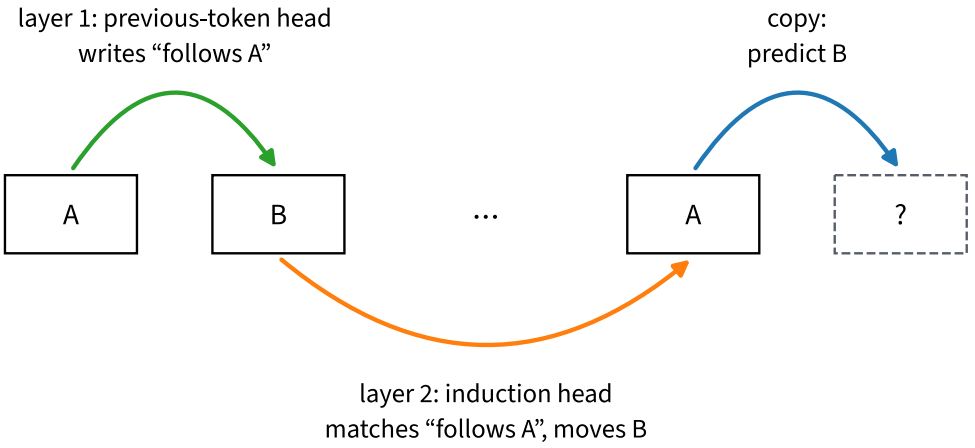


Figure 10.14: 誘導回路は 2 層にまたがる。第 1 層では直前トークンヘッドが各トークンの先行トークンをそのストリームに書き込むため、B の位置のキーは「A に続く」と告知できる。第 2 層では誘導ヘッドの QK 回路が現在の A をその告知と照合し、OV 回路が B を予測へとコピーする。

10.7.2 コピーと誘導

10.7.2.1 反復系列タスク

勾配降下法がこの 2 層の回路を発見するかどうかを検証するため、**反復ランダムトークン**の合成系列でモデルを訓練する。各系列では、長さ $L \in [L_{\min}, L_{\max}]$ のランダムなパターンをサイズ 64 の語彙から一様に抽出し、長さ 64 になるまで並べて繰り返す。パターンは毎回新たに抽出されるため、初回コピー（$t < L$）は予測不可能である（ランダム予測での交差エントロピーは $\log 64 \approx 4.16$ nats）。パターンが再開する位置 $t = L$ のトークンも同様である。そのコンテキスト内ではまだ何も繰り返されておらず、モデルは L を知り得ないからである。したがって、ターゲット $t \leq L$ を初回コピーとして数える。後続のコピー（$t > L$）は、コンテキスト内参照によってコンテキストから予測できる。

```
def repeated_batch(key, batch_size, num_steps, vocab, Lmin, Lmax):
    """Random patterns of length Lmin..Lmax, tiled to num_steps tokens."""
    key_L, key_s = jax.random.split(key)
    L = jax.random.randint(key_L, (batch_size,), Lmin, Lmax + 1)
    s = jax.random.randint(key_s, (batch_size, num_steps), 0, vocab)
    pos = jnp.arange(num_steps)
    return jnp.take_along_axis(s, pos[None, :] % L[:, None], axis=1), L

x, L = repeated_batch(d2l.get_key(), batch_size=2, num_steps=16,
                      vocab=10, Lmin=5, Lmax=8)
print(f'pattern lengths: {L.tolist()}')
print(x)
```

```
pattern lengths: [6, 5]
[[6 8 1 6 3 6 6 8 1 6 3 6 6 8 1 6]
```

```
[7 0 0 4 8 7 0 0 4 8 7 0 0 4 8 7]]
```

eval_copy は、初回コピー（$t \leq L$）での交差エントロピー、後続のコピー（$t > L$）での交差エントロピー、および後続のコピーでの top-1 正解率を測定する。

```
def eval_copy(model, x, L):
    """First-copy loss, later-copy loss, and later-copy accuracy."""
    logits, Y = model(x[:, :-1]), x[:, 1:]
    ce = optax.softmax_cross_entropy_with_integer_labels(logits, Y)
    later = jnp.arange(1, Y.shape[1] + 1)[None, :] > L[:, None]
    acc = ((logits.argmax(-1) == Y) * later).sum() / later.sum()
    first = (ce * ~later).sum() / (~later).sum()
    rep = (ce * later).sum() / later.sum()
    return [float(v) for v in jax.device_get((first, rep, acc))]

class RepeatedTokens(d2l.DataModule):
    """An endless stream of fresh repeated-token minibatches."""
    def __init__(self, Lmin, Lmax, batch_size=128, num_steps=64, vocab=64):
        super().__init__()
        self.save_hyperparameters()

    def train_dataloader(self):
        while True:
            x, _ = repeated_batch(d2l.get_key(), self.batch_size,
                                  self.num_steps, self.vocab, self.Lmin,
                                  self.Lmax)
            yield x[:, :-1], x[:, 1:]

def train_repeat(model, num_steps, Lmin, Lmax, eval_every=50):
    """Train with Adam; return (first, later) copy losses per round."""
    data = RepeatedTokens(Lmin, Lmax)
    optimizer = nnx.Optimizer(model, optax.adam(1e-3), wrt=nnx.Param)
    history = []
    for _ in range(num_steps // eval_every):
        d2l.train_steps(model, data, optimizer, eval_every)
        x, L = repeated_batch(d2l.get_key(), 256, 64, 64, Lmin, Lmax)
        history.append(eval_copy(model, x, L)[:2])
    return history

def drop_step(hist, eval_every=50):
    """First step at which the later-copy loss is below 1 nat, or None."""
    return next((eval_every * (k + 1) for k, (_, rep) in enumerate(hist)
                 if rep < 1), None)
```

10.7.2.2 固定オフセットによる解

可変周期を試す前に、パターンの周期が $L = 32$ に固定されているとしよう。**1 ブロック**のモデル（num_blks=1, bias=True）は、容易に 100% の 2 回目のコピーの正解率を達成する。

```
model_shortcut = d2l.TinyCharLM(vocab_size=64, num_blks=1, pos='rope',
                                bias=True, rngs=nnx.Rngs(d2l.get_key()))
train_repeat(model_shortcut, 800, Lmin=32, Lmax=32)
```

```
first, rep, acc = eval_copy(model_shortcut, *repeated_batch(
    d2l.get_key(), 256, 64, 64, 32, 32))
print(f'first copy {first:.2f} nats, second copy {rep:.2f} nats, '
      f'second-copy accuracy {acc:.2f}')
```

```
first copy 4.27 nats, second copy 0.08 nats, second-copy accuracy 1.00
```

回路の解析では内容に基づく誘導に 2 層が必要であると論じたのに、なぜ単一の層で後続トークンをコピーできるのだろうか。$L = 32$ が一定のとき、位置 $i + 1$ のターゲットトークンは常に固定オフセット $j = i - 31$ のトークンと一致する。定数バイアスのクエリとキーを持つ単一の RoPE ヘッドは、内容を一切照合せず純粋に相対オフセット -31 だけでアテンションを向けることができる（演習問題 3）。`model_shortcut` を周期 $L = 24$ のパターン（ターゲットはオフセット -23 にある）で評価すると、このショートカットはただちに露呈する。アテンションはオフセット -31 に固定されたままであり、正解率はランダム予測の水準（$1/64 \approx 0.02$）まで崩壊する。

```
x, L = repeated_batch(d2l.get_key(), 256, 64, 64, 24, 24)
first, rep, acc = eval_copy(model_shortcut, x, L)
m = model_shortcut.attention_weights(x[:, :-1])[0]
print(f'patterns of length 24: second-copy accuracy {acc:.2f}')
for offset in (31, 23):
    mass = jnp.diagonal(m, -offset, axis1=2, axis2=3).mean((0, 2))
    print(f'mass at offset {offset}, per head: '
          + ' '.join(f'{v:.2f}' for v in mass.tolist()))
```

```
patterns of length 24: second-copy accuracy 0.02
mass at offset 31, per head: 0.92 1.00 0.94 1.00
mass at offset 23, per head: 0.00 0.00 0.00 0.00
```

10.7.2.3 2 ブロックの誘導回路

パターンの周期を $L \in \{16, \ldots, 32\}$ から一様にサンプリングすると、いかなる固定オフセットのショートカットも使えなくなる。ここでは、独立な 3 つの 2 ブロックモデルと対照用の 1 ブロックモデル（すべて `bias=False`）を 5,000 ステップ訓練する。

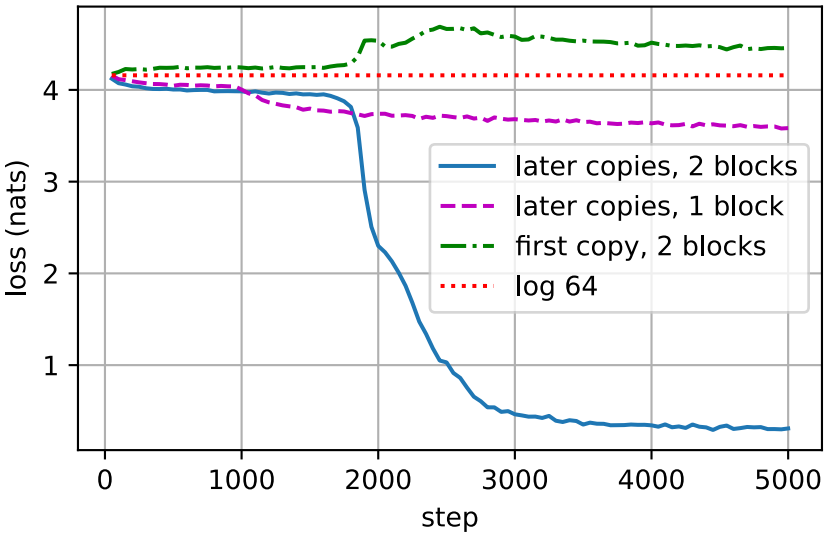
```
runs = []  # (blocks, model, history): three two-block runs, one control
for num_blks in (2, 2, 2, 1):
    model = d2l.TinyCharLM(vocab_size=64, num_blks=num_blks, pos='rope',
                           rngs=nnx.Rngs(d2l.get_key()))
    runs.append((num_blks, model, train_repeat(model, 5000, 16, 32)))
models_two = [model for _, model, _ in runs[:3]]
_, model_two, hist_two = runs[0]   # Run 1: maps and curves shown for it
_, model_one, hist_one = runs[3]
x, L = repeated_batch(d2l.get_key(), 256, 64, 64, 16, 32)
print('blocks  first copy  later copies  accuracy  below 1 nat at step')
for num_blks, model, hist in runs:
    first, rep, acc = eval_copy(model, x, L)
    print(f'{num_blks:6d}{first:12.2f}{rep:14.2f}{acc:10.2f}'
          f'{drop_step(hist) or "never":>21}')
```

```
blocks  first copy  later copies  accuracy  below 1 nat at step
     2        4.45          0.32      0.94                 2550
```

```
2        4.43          0.42       0.91              2300
2        4.43          0.30       0.94              2400
1        4.28          3.60       0.11             never
```

2 ブロックの 3 回の実行はすべて 90% を超える後続コピーの正解率（損失にして約 0.2〜0.4 nats）を達成するのに対し、1 ブロックのモデルは 3 nats を超えたところで停滞する（正解率は ∼ 10% であり、コンテキスト内に現れたトークンのユニグラム集合のスコアを押し上げているにすぎない）。初回コピーの損失は、どの実行でもランダム予測の値である $\log 64$ をわずかに**上回る**結果となる。4 つのモデルはすべてコンテキスト内に既出のトークンを選好するよう学習しており、最初の周期ではこの確信が裏目に出る。すなわち、そこで偶然トークンが重複すると、2 ブロックのモデルはそのトークンの前回の後続トークンを予測してしまうのである。さらに、訓練曲線は初期のプラトーの後に急激な相転移（phase transition）を示す。

```
steps = list(range(50, 5001, 50))
d2l.plot(steps, [[h[1] for h in hist_two], [h[1] for h in hist_one],
                 [h[0] for h in hist_two], [math.log(64)] * len(steps)],
         'step', 'loss (nats)',
         legend=['later copies, 2 blocks', 'later copies, 1 block',
                 'first copy, 2 blocks', 'log 64'], figsize=(5, 3))
```



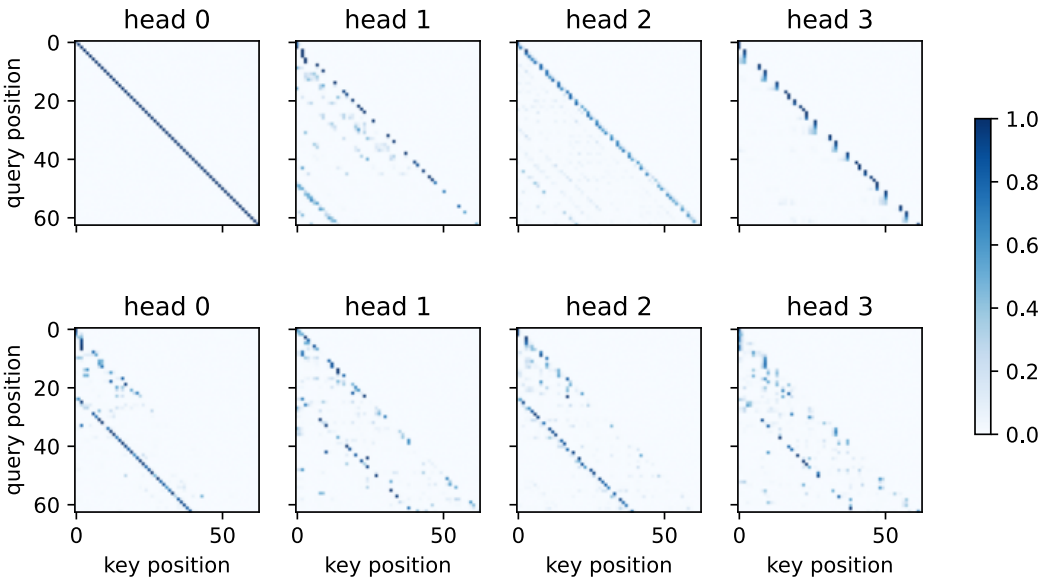
10.7.3 回路の裏付け

`model_two` が Figure 10.14 の 2 段階の誘導回路（induction circuit）を実際に実装しているかを検証するため、(1) アテンションマップ、(2) OV コピー行列、(3) ヘッドアブレーションを調べる。

10.7.3.1 学習されたヘッドのアテンションパターン

周期 $L = 24$ のテスト系列では、この回路の予測によれば、少なくとも 1 つのブロック 1 のヘッドが 1 位置前（$j = i - 1$）にアテンションを向け、ブロック 2 のヘッドは誘導ターゲット $j = i + 1 - L = i - 23$ に（$i \geq 24$ 以降で）アテンションを向けるはずである。

```
x, L = repeated_batch(d2l.get_key(), 1, 64, 64, 24, 24)
maps = jnp.stack(model_two.attention_weights(x[:, :-1]))
d2l.show_heatmaps(maps[:, 0], xlabel='key position',
                  ylabel='query position',
                  titles=[f'head {h}' for h in range(4)],
                  figsize=(9, 4.5), cmap='Blues')
```



可変のパターン長 $L \in \{16, \ldots, 32\}$ および 3 回の実行すべてにわたってこれら両方の役割を定量化するため、head_scores は直前トークン（$j = i - 1$）と誘導ターゲット（$j = i + 1 - L$、ただし $i \geq L$）に対する各ヘッドの平均アテンション重みを計算する。

```
def head_scores(model, x, L):
    """Mean attention on the previous token and on the induction target:
    shape (blocks, 2, heads)."""
    maps = jnp.stack(model.attention_weights(x[:, :-1]))
    i = jnp.arange(maps.shape[-1])
    tgt = i + 1 - L[:, None]               # Induction target of query i
    is_tgt = i == tgt[..., None]           # (batch, query, key)
    valid = tgt >= 1                       # Query token occurred before
    prev = jnp.diagonal(maps, -1, axis1=3, axis2=4).mean((1, 3))
    ind = (maps * is_tgt[:, None]).sum(-1)  # (blocks, batch, heads, query)
    ind = (ind * valid[:, None]).sum((1, 3)) / valid.sum()
    return jnp.stack((prev, ind), 1)

x, L = repeated_batch(d2l.get_key(), 256, 64, 64, 16, 32)
all_scores = [head_scores(model, x, L) for model in models_two]
scores = all_scores[0]  # Run 1, whose maps are shown above
for r, run_scores in enumerate(all_scores):
    for b, rows in enumerate(run_scores.tolist()):
        for name, row in zip(('previous', 'induction'), rows):
```

```
            print(f'run {r + 1}, block {b + 1}, {name:<9}: '
                  + ' '.join(f'{v:.2f}' for v in row))
```

```
run 1, block 1, previous : 0.00 0.46 0.33 0.50
run 1, block 1, induction: 0.00 0.01 0.01 0.00
run 1, block 2, previous : 0.09 0.08 0.06 0.09
run 1, block 2, induction: 0.64 0.39 0.38 0.35
run 2, block 1, previous : 0.04 0.39 0.02 0.49
run 2, block 1, induction: 0.01 0.00 0.01 0.01
run 2, block 2, previous : 0.06 0.06 0.06 0.05
run 2, block 2, induction: 0.52 0.31 0.32 0.35
run 3, block 1, previous : 0.06 0.42 0.02 0.50
run 3, block 1, induction: 0.02 0.01 0.00 0.01
run 3, block 2, previous : 0.05 0.05 0.09 0.06
run 3, block 2, induction: 0.60 0.34 0.38 0.33
```

どの実行でも、ブロック 1 のヘッドのうち 1〜3 個が直前トークンヘッドに特化する一方、ブロック 2 のヘッドはアテンションを誘導ターゲット $j = i + 1 - L$ に集中させる。

10.7.3.2 コピー行列

位置 j にアテンションを向けて x_j を予測できるのは、ブロック 2 の OV 回路がトークン x_j のロジットを押し上げる場合に限られる。ここで、次の量を定義する。

$$C_{ab} = \mathbf{e}_b^\top \mathbf{W}_{\mathrm{OV}} \bar{\mathbf{h}}_a \tag{10.37}$$

これはトークン b に与えられるロジットの押し上げ量であり、あるブロックの総和 OV 回路が残差ストリームの平均ベクトル $\bar{\mathbf{h}}_a$（トークン a を保持する各位置での平均、または生の埋め込み $\mathbf{e}_a$）を読み取ったときの値を表す。コピーを行う OV 回路では、対角成分（$b = a$）がほぼすべての 64 行で行内の最大値となる（ランダムな場合は $1/64 \approx 0.02$ である）。

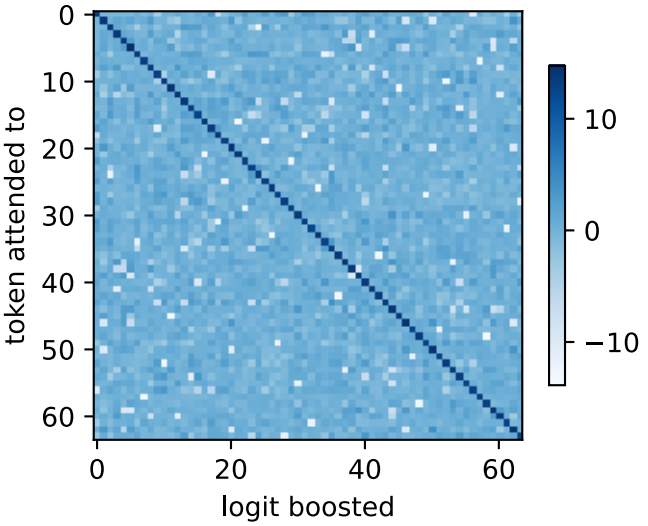
```
def circuits(model, blk):
    """W_QK and W_OV of every head of block blk, each (heads, D, D)."""
    # nnx.Linear stores its kernel as (input, output), the transpose of the
    # column-vector equations, so both kernels are transposed first
    W_qkv = model.blks[blk]['qkv'].kernel[...].T    # (3D, D): W_q; W_k; W_v
    W_o = model.blks[blk]['proj'].kernel[...].T     # (D, D): heads in columns
    H, D = model.num_heads, W_o.shape[0]
    W_q, W_k, W_v = W_qkv.reshape(3, H, D // H, D)  # Each (H, d_h, D)
    W_QK = W_q.mT @ W_k
    W_OV = W_o.reshape(D, H, D // H).swapaxes(0, 1) @ W_v
    return W_QK, W_OV

def copy_matrix(model, blk, X=None):
    """C[a, b] = e_b^T W_OV h_a: W_OV summed over the heads of blk, h_a the
    mean stream vector entering blk at the positions of token a in X, or the
    embedding e_a if X is None."""
    E = h = model.token_emb.embedding[...]
    if X is not None:
        S = model.stream(X, blk)[0]            # The stream entering blk
        h = jnp.zeros_like(E).at[X.flatten()].add(S.reshape(-1, E.shape[1]))
```

```
        h = h / jnp.bincount(X.flatten(), length=len(E))[:, None]
    W_OV = circuits(model, blk)[1].sum(0)
    return h @ W_OV.T @ E.T

def diagonal_share(C):
    """Fraction of the rows of C whose largest entry is on the diagonal."""
    return float((C.argmax(-1) == jnp.arange(len(C))).mean())
```

2 ブロックの 3 回の訓練実行すべて（ブロック 1 の残差ストリームの平均 $\bar{\mathbf{h}}_a$ と生のトークン埋め込み $\mathbf{e}_a$ の両方）および対照用の 1 ブロックモデルにわたって `diagonal_share(copy_matrix(...))` を評価すると、OV 回路がほぼ対角なコピー行列を形成していることがわかる。

```
x, L = repeated_batch(d2l.get_key(), 256, 64, 64, 16, 32)
print('rows whose largest entry is the diagonal (chance 1/64)')
print(f'{"":<32}   run 1   run 2   run 3')
for name, X in (('2 blocks, block 2, mean stream', x),
                ('2 blocks, block 2, embeddings', None)):
    shares = [diagonal_share(copy_matrix(m, 1, X)) for m in models_two]
    print(f'{name:<32}' + ''.join(f'{v:8.2f}' for v in shares))
print(f'{"1 block, mean stream":<32}'
      f'{diagonal_share(copy_matrix(model_one, 0, x)):8.2f}')
d2l.show_heatmaps(copy_matrix(model_two, 1, x)[None, None],
                  xlabel='logit boosted', ylabel='token attended to',
                  figsize=(3.6, 3.6), cmap='Blues')
```

```
rows whose largest entry is the diagonal (chance 1/64)
                                 run 1   run 2   run 3
2 blocks, block 2, mean stream    1.00    1.00    1.00
2 blocks, block 2, embeddings     0.81    0.81    0.27
1 block, mean stream              0.98
```



ここで示唆に富む 2 つの機微に注目しよう。(1) 対照用の 1 ブロックモデルも対角な OV コピー行列を学習しており、このことから QK の K 合成を伴わない OV コピーだけではコンテキスト内

の後続トークンを検索できないことがわかる（演習問題 6）。(2) ブロック 2 は、ブロック 1 のストリーム $\bar{\mathbf{h}}_a$ からのほうが、実行ごとに対角成分の割合が大きくばらつく生の埋め込み $\mathbf{e}_a$ 単体からよりも確実にコピーできる。

10.7.3.3 ヘッドアブレーション

因果的な必要性を検証するため、ablate はブロック内の選択したヘッドの出力射影スライス $\mathbf{W}_o^m$ を 0 にする。ここでは、ブロック 1 の直前トークンヘッド（スコア > 0.2）を除去する場合、ブロック 1 の残りのヘッドを除去する場合、およびブロック 2 の誘導ヘッドを強いものから順に累積的に除去する場合を比較する。

```
def ablate(model, blk, heads):
    """Copy of model with the given heads of block blk removed."""
    model = nnx.clone(model)
    W_o = model.blks[blk]['proj'].kernel   # (D, D): heads stacked in rows
    W = W_o[...].reshape(model.num_heads, -1, W_o.shape[1])
    W_o[...] = W.at[jnp.array(heads, dtype=int)].set(0).reshape(W_o.shape)
    return model
```

訓練済みの 3 つの 2 ブロックモデルすべてに ablate を適用すると、ブロック 1 の直前トークンヘッドを除去した場合とブロック 2 の上位の誘導ヘッドを除去した場合とで、後続コピーの交差エントロピーおよびコピー正解率がどれだけ悪化するかが定量化される。

```
x, L = repeated_batch(d2l.get_key(), 256, 64, 64, 16, 32)
H, table = model_two.num_heads, {}  # Variant: (loss, accuracy) per run
for r, (model, s) in enumerate(zip(models_two, all_scores)):
    prev = [h for h in range(H) if s[0, 0, h] > 0.2]  # Previous-token heads
    rest = [h for h in range(H) if h not in prev]
    order = jnp.argsort(-s[1, 1]).tolist()  # Strongest first
    print(f'run {r + 1}: block 1 previous-token heads {prev}, others {rest}')
    variants = {'none': model,
                'block 1, previous-token heads': ablate(model, 0, prev),
                'block 1, all other heads': ablate(model, 0, rest)}
    for k in range(1, H + 1):
        variants[f'block 2, top {k} induction'] = ablate(model, 1, order[:k])
    for name, variant in variants.items():
        table.setdefault(name, []).append(eval_copy(variant, x, L)[1:])
table['one-block model'] = [eval_copy(model_one, x, L)[1:]]
print(f'{"heads removed":<31}later-copy loss / accuracy: run 1, 2, 3')
for name, cols in table.items():
    print(f'{name:<31}'
          + ''.join(f'{rep:7.2f} /{acc:5.2f}' for rep, acc in cols))
```

```
run 1: block 1 previous-token heads [1, 2, 3], others [0]
run 2: block 1 previous-token heads [1, 3], others [0, 2]
run 3: block 1 previous-token heads [1, 3], others [0, 2]
heads removed                  later-copy loss / accuracy: run 1, 2, 3
none                              0.32 / 0.93   0.42 / 0.90   0.32 / 0.94
block 1, previous-token heads     4.31 / 0.05   4.55 / 0.05   4.29 / 0.04
block 1, all other heads          4.98 / 0.02   5.02 / 0.06   5.23 / 0.03
block 2, top 1 induction          0.76 / 0.91   0.70 / 0.88   0.58 / 0.91
block 2, top 2 induction          1.76 / 0.69   1.78 / 0.68   1.46 / 0.73
```

```
block 2, top 3 induction          3.34 / 0.25   2.73 / 0.44   3.16 / 0.32
block 2, top 4 induction          4.24 / 0.01   4.31 / 0.02   4.28 / 0.02
one-block model                   3.59 / 0.11
```

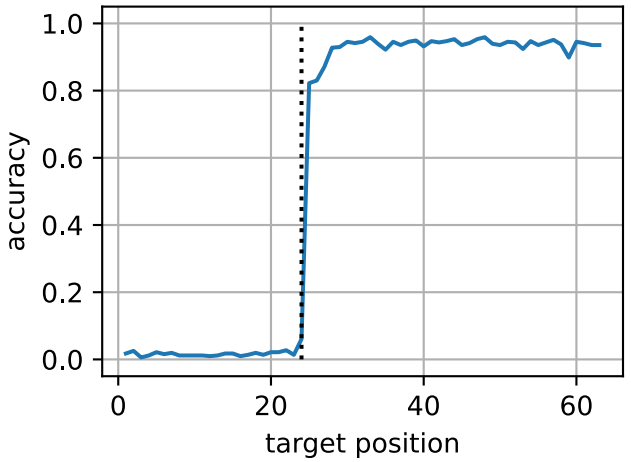
ブロック 1 の直前トークンヘッドをアブレーションすると、後続コピーの正解率は 1 ブロックモデルと同水準の 0.2 未満まで崩壊し、それらの必要性が裏付けられる。興味深いことに、ブロック 1 の**それ以外の**ヘッドをアブレーションしても検索が壊れる。`copy_matrix` がブロック 1 で変換されたストリームに依存していた（さらに正規化を持たないネットワークではゼロアブレーションによってストリームのノルムがずれる。平均アブレーションについては演習問題 5 を参照）ため、ブロック 2 はそれらのヘッドにも依存しているのである。ブロック 2 では、4 つのヘッドが誘導の負荷を冗長に分担しており、正解率がランダム予測水準まで落ちるには複数のヘッドをアブレーションする必要がある。

10.7.4 コンテキスト内学習

10.7.4.1 未知のパターンへの汎化

取り得るパターンは $64^{16} \approx 8 \times 10^{28}$ 通り存在するため、テスト系列はどれも未知であり、訓練分布全体ではすべてのトークン対が等確率で現れる。誘導回路は、コンテキストウィンドウ内で 1 度提示されただけで各トークンとその直後のトークンをその場で結びつける。これは**コンテキスト内学習**（in-context learning）の最小の例である。

```
x, L = repeated_batch(d2l.get_key(), 512, 64, 64, 24, 24)
acc = (model_two(x[:, :-1]).argmax(-1) == x[:, 1:]).mean(0)
d2l.plot([jnp.arange(1, 64), [24, 24]], [acc, [0, 1]],
         'target position', 'accuracy', fmts=('-', 'k:'))
```



正解率は、初回コピーでの $1/64$（内容照合がまだ重複トークンを見ていない再開境界 $t = 24$ でも同じ）から、約 0.8（位置 25）へと跳ね上がり、その数位置後には 0.9 を超える。

再開後に残る誤差は何に起因するのだろうか。あるトークンが $K > 1$ 回、異なる後続トークンを伴って L トークンのパターン内に現れるとき、現在のトークンだけの照合では曖昧さが残る。すなわち、K 個の出現箇所から一様に選ぶと確率 $1/K$ でしか成功しない。$L = 24$、$p = 1/64$ のとき、パターン内でトークンが重複する確率は $1 - (63/64)^{23} \approx 0.30$ であり、一様な単一トークン照合が予測する全体の正解率は次の式で与えられる。

$$\mathbb{E}\left[\frac{1}{K}\right] = \frac{1-(1-p)^L}{Lp} \approx 0.84, \qquad p = \frac{1}{64},$$

また、重複トークンに限ると正解率は ≈ 0.47 となる。テスト位置を $K = 1$（一意）と $K > 1$（重複）に分けると、3 回の実行すべてで一意なトークンでは 0.97 以上、重複トークンでも約 0.7〜0.9 に達し、単一トークン照合のベースラインである 0.47 を大きく上回る。ここでは検証しないが、もっともらしい理由として、モデルがより長い n グラムのコンテキストも照合していることや、特定の RoPE オフセットを選好していることが挙げられる（演習問題 2）。

```
period, num_steps = 24, 64
x, _ = repeated_batch(d2l.get_key(), 512, num_steps, 64, period, period)
pattern = x[:, :period]
K = (pattern[:, :, None] == pattern[:, None, :]).sum(-1)  # Count in pattern
K = K[:, jnp.arange(period, num_steps - 1) % period]  # Current token
parts = (K == 1, K > 1, K > 0)           # Unique, repeated, all
rows = {'share of targets': [p.mean() for p in parts],
        'single-token match': [(1 / K[p]).mean() for p in parts]}
for r, model in enumerate(models_two):
    hit = (model(x[:, :-1]).argmax(-1) == x[:, 1:])[:, period:]
    rows[f'run {r + 1} accuracy'] = [hit[p].mean() for p in parts]
print(f'{"current token":<18}   unique repeated      all')
for name, row in rows.items():
    print(f'{name:<18}' + ''.join(f'{v:9.2f}' for v in row))
```

```
current token        unique repeated      all
share of targets       0.70     0.30     1.00
single-token match     1.00     0.48     0.84
run 1 accuracy         0.99     0.81     0.93
run 2 accuracy         0.97     0.75     0.91
run 3 accuracy         0.99     0.81     0.94
```

10.7.4.2 より大規模なモデルにおける誘導ヘッド

Olsson et al. (2022) は、さまざまなスケールの言語モデルにおいて、誘導ヘッドが急激な相転移の過程で形成され、それが一般的なコンテキスト内学習の創発と時期を同じくすることを実証した（演習問題 8）。コピーやあいまいなパターン補完にとどまらず、アテンション層は回帰タスクにおけるコンテキスト内の勾配ステップを実装することもできるが（演習問題 7、(Oswald et al. 2023)）、事前学習済みの言語モデルが実際にこの方法でコンテキスト内学習をしているかどうかについては議論がある (Shen et al. 2024)。

ここまでの解析は正規化も MLP も持たないモデルに依拠していたが、実際の Transformer ブロック（第 11 章）はその両方を備えている。いずれも基本的な考え方は変えないが、解析手法には修正が必要になる。層正規化（layer normalization）はストリームを自身のノルムで再スケーリングするため、ストリームからロジットへの写像はそのスケールを固定したときにのみ線形になる。そこで回路解析では、スケールを観測された値に固定し、残りを従来どおり扱う。MLP はどのアテンションパターンも経由しない経路を追加し、あるヘッドの出力が後段のヘッドに読み取られる前に MLP によって書き換えられることもある。最後に、訓練済みの言語モデルにおける誘導ヘッドはあいまいな照合を行う。すなわち、以前出現したトークンと同一ではなく単に類似しているだけのトークンに対しても、$[A^*][B^*] \ldots [A] \to [B]$ を補完する。

10.7.5 説明としてのアテンション重み

ここまでの検証から、機構的な説明としてアテンションのヒートマップだけでは不十分な理由が明らかになる。

- **重要なヘッドでも目立たないマップを持ち得る：**ブロック 1 の直前トークンヘッド以外のヘッドはヒートマップ上では不活性に見えたが、それらをアブレーションするとブロック 2 の検索が壊れた。
- **鋭いマップでも効果が無視できるほど小さい場合がある：**事前学習済みの言語モデルでは、遊休状態のヘッドが重みの大半を先頭トークン（**アテンションシンク**）に置くことが頻繁にあるが、その OV ベクトルはほぼゼロのノルムしか持たない (Xiao et al. 2024; Gu et al. 2025)。

Table 10.2 に、誘導回路を検証するために用いた証拠の階層をまとめる。

Table 10.2: 誘導ヘッドの解釈に対するレベル別の証拠。

本節での検証	証拠のレベル	支持される結論	残る限界
後続コピーの損失	振る舞い	2 ブロックは反復トークンタスクを解けるが、1 ブロックでは解けない	内部の構成要素を特定しない
周期の変更	入力への介入	固定周期モデルは内容ではなく位置を利用する	入力だけでなくタスクも変えてしまう
ヘッドスコア	活性化値	各ヘッドが予測どおりのオフセットにアテンションを向ける	値の変換を考慮していない
コピー行列	重みと平均活性化値	ブロック 2 の OV 回路がトークンをコピーする	検索を行わないモデルにも存在する
ヘッドアブレーション	構成要素への介入	後続コピーの予測には直前トークンヘッドだけでなくブロック 1 全体とブロック 2 の誘導ヘッドが必要	ゼロ化によりストリームが訓練分布から外れる

10.7.6 まとめ

各アテンションヘッドは、高々ランク d_h の 2 つの行列を介して残差ストリームに作用する。すなわち、どこにアテンションを向けるかを決める **QK 回路** $\mathbf{W}_q^\top \mathbf{W}_k$ と、参照先の位置へどのベクトルを書き込むかを決める **OV 回路** $\mathbf{W}_o \mathbf{W}_v$ である。単一のアテンション層はバイグラムやスキップトライグラム（単一トークンのコピーを含む）を表現できる一方、未知のコンテキスト内パターン $[\mathrm{A}][\mathrm{B}] \ldots [\mathrm{A}] \to [\mathrm{B}]$ を補完するには、**直前トークンヘッド**と照合コピーを行う**誘導ヘッド**が残差ストリームを介して合成できるよう少なくとも 2 つの層が必要である。可変周期の反復ランダムパターンで訓練すると、2 ブロックモデルは急激な相転移を経て 90% を超えるコンテキスト内正解率に達するのに対し、固定周期では 1 ブロックモデルが脆い位置ショートカットを利用できてしまう。したがって、ヘッドが実際に何を計算しているかを確立するには、アテンションパターンのプローブ、OV 重みの検査、および因果的ヘッドアブレーションを組み合わせる必要がある。

10.7.7 演習問題

1. **重み共有バイグラム。**0 ブロックのとき、トークン b がトークン a に続くロジットは $\mathbf{e}_b^\top \mathbf{e}_a$ であるため、埋め込みの幾何構造だけでモデルが選好する後続トークンが決まる。b を a の

最上位後続トークンと呼ぶのは $\mathbf{e}_b^\top \mathbf{e}_a$ がその行で最大のロジットであるときとし、同順位はないものとする。

1. すべての埋め込みが同じノルムを持つならば、あらゆるトークンの最上位後続トークンはそのトークン自身になることを示せ。
2. 等ノルムの仮定を外す。トークン $b \neq a$ が a の最上位後続トークンになり得るのは $\|\mathbf{e}_b\| > \|\mathbf{e}_a\|$ の場合に限られることを示し、最上位後続トークンが巡回 $a_1 \to a_2 \to \cdots \to a_k \to a_1$（$k \geq 2$）を形成し得ないことを結論づけよ。
3. あるコーパスにおいて、文字 `t` の最頻の後続文字が `h`、`h` の最頻の後続文字が `e`、`e` の最頻の後続文字が空白、空白の最頻の後続文字が `t` であるとする。どれほど長く訓練しても、0 ブロックのモデルではこれら 4 つすべてを最上位後続トークンとして再現できないことを示せ。
4. 1 ブロックのモデルでは、先頭位置にあるすべてのヘッドはその位置自身にのみアテンションを向ける。そこでのロジットを $\mathbf{EMe}_{x_0}$（$\mathbf{M}$ は OV 回路から構成される行列）と書き表し、前問までの 2 つの制約がいずれも $\mathbf{EME}^\top$ には当てはまらない理由を説明せよ。

2. **2 つのブロックを通る経路**。N 個のブロック（各 H ヘッド）からなるモデルの T トークン系列上でのアテンションパターンを固定し、$\mathbf{A}^{\ell m}$ を $T \times T$ 行列（成分は $\alpha_{ij}^{\ell m}$）とする。
   1. Equation 10.35 のロジットを $N = 2$ に対して経路項の和へと展開せよ。ブロック 1 のヘッド m とブロック 2 のヘッド m' を通る経路が、パターン $\mathbf{A}^{2m'}\mathbf{A}^{1m}$ と OV 回路 $\mathbf{W}_{\text{OV}}^{2m'}\mathbf{W}_{\text{OV}}^{1m}$ を持つ単一のヘッドのように作用することを示し、その回路のランクの上界を求めよ。
   2. N ブロックに対する経路項の個数を数え、`model_two` についてその個数を計算せよ。
   3. 訓練済みモデルではブロック 2 のパターンは固定されておらず、$\mathbf{h}^{(1)}$ にブロック 2 の QK 回路を介して依存する。$\mathbf{h}_i^{(1)}$ と $\mathbf{h}_j^{(1)}$ を Equation 10.34 のスコアに代入し、クエリ側、キー側、またはその両方にブロック 1 のヘッドの出力が含まれるかどうかによって項を分類せよ。これらの項は **Q 合成**および **K 合成**と呼ばれ、最初の小問の経路は **V 合成**と呼ばれる (Elhage et al. 2021)。Figure 10.14 の誘導回路はどの種類の項に依拠しているか。
   4. 2 トークンのプレフィックス (x_{i-1}, x_i) での照合が、ブロック 1 にオフセット -1 と -2 へアテンションを向けるヘッドが含まれるときに 2 つのブロックで実現できることを示し、照合のそれぞれの半分を担う合成項の名称を答えよ。
3. [code] **バイアスなしのショートカット**。10.7.2.2 節の固定オフセットモデルは射影バイアスを持っている。
   1. 回転前のヘッドのクエリとキーが、すべての位置で同じ定数ベクトル $\mathbf{q}$ と $\mathbf{k}$ であるとする。$\mathbf{R}_i^\top \mathbf{R}_j = \mathbf{R}_{j-i}$ を用いてスコアがオフセット $j-i$ のみに依存することを示し、$\mathbf{q}$ と $\mathbf{k}$ をどのように選べばオフセット -31 でスコアが最大になるかを説明せよ。
   2. バイアスがない場合、すべてのクエリとキーはストリームの線形関数となる。ヘッドがほぼ定数のクエリとキーを生成するためには、トークン埋め込みに何が含まれていなければならないか。
   3. 本文の 1 ブロックモデルを `bias=False` で 3 回訓練し、2 回目のコピーの正解率とオフセット 31 における各ヘッドのアテンションを報告せよ。バイアスなしでショートカットはどの程度確実に形成されるか。また、周期 24 のプローブでも維持されるか。
4. **何も変えない重み**。クエリ i におけるヘッドの出力は $\mathbf{W}_{\text{OV}} \sum_j \alpha_{ij} \mathbf{h}_j$（Equation 10.35）であり、ソフトマックスによりマスクされていないキー上のすべての重み α_{ij} は正になる。Jain and Wallace (2019) は、訓練済みのテキスト分類器において大きく異なるアテンション分布がしばしば予測を変化させないことを見いだし、アテンション重みは予測を説明しないと結論づけた。
   1. n 個のキー上のアテンション分布のうち、α_i と同じヘッド出力を与えるものの集合は、α_i の近傍に少なくとも次元 $n - 1 - d_h$ のアフィン部分空間の一部を含むことを

示せ。63 トークンの入力を `model_two` に与えたときの最後のクエリについてこの下界を計算せよ。

2. 最初の小問における代替分布は、Jain and Wallace (2019) のものと同様に、1 回につき 1 つの入力に対して構成されている。Wiegreffe and Pinter (2019) は、代替分布が説明に対する反証となるのは、単一の訓練済みモデルがあらゆる入力に対してそのような代替分布を生成しつつほぼ同じ予測を行う場合に限られると論じた。最初の小問の次元計算からはそのようなモデルが存在するかどうかについて何も言えない理由を説明し、`model_two` についてそれを判定する実験を記述せよ。

5. [code] **ゼロアブレーションと平均アブレーション**。10.7.3.3 節のアブレーションでは、ヘッドの出力をゼロに設定した。
   1. `proj` のあるヘッドに対応する列をゼロにすると、残差ストリームに対するそのヘッドの寄与だけが厳密に取り除かれる理由を説明せよ。ブロック 1 のヘッドを除去したとき、どのアテンションパターンが変化せず、どのパターンが変化し得るか。
   2. 訓練中にヘッドの出力がゼロになることはないため、ゼロ化するとストリームはモデルが未経験の状態へと移動する。**平均アブレーション**は、代わりにその出力 $\mathbf{o}_m$（ヘッド m の出力）を位置と系列にわたる平均 $\bar{\mathbf{o}}_m$ で置き換える。これを実装せよ。`TinyCharLM` を `bias=True` で構築し、`model_two` の重みを（`qkv` のバイアスはゼロとして）コピーし、ヘッド m に対応する `proj` の列をゼロにしたうえで、`proj` のバイアスをゼロ化前のそれらの列と $\bar{\mathbf{o}}_m$ の積に設定せよ。本文のアブレーションを再度実行せよ。ブロック 1 の直前トークンヘッド以外のヘッドを除去しても依然として検索は壊れるか。また、その答えは 10.7.3.3 節の 2 つの解釈のうちどちらを支持するか。
   3. ブロック 2 で 2 番目に強い誘導ヘッドを単独で除去せよ。単一ヘッドをそれぞれ除去したときの後続コピーの損失の増加量の和と、両方のヘッドを同時に除去したときの増加量を比較せよ。この比較から、ヘッドがどのように役割を分担しているかについて何が言えるか。
   4. Table 10.2 に挙げた限界のうち、平均アブレーションによって解消されるものはどれで、残るものはどれか。

6. [code] **重みにおける K 合成**。●誘導ヘッドの照合が機能するのは、ブロック 2 の QK 回路がブロック 1 の OV 回路によってストリームに書き込まれた内容を読み取るからである。
   1. ブロック 1 のヘッド m とブロック 2 のヘッド m'（`model_two`）のすべての対について、64×64 行列
   $$\mathbf{E}\,\mathbf{W}_{\mathrm{QK}}^{2m'}\,\mathbf{W}_{\mathrm{OV}}^{1m}\,\mathbf{E}^{\top}, \tag{10.38}$$
   を計算せよ。その成分 (a, b) は、クエリトークン a を直前のトークンが b であったキーに対してスコア付けする。Equation 10.37 のコピーテストがコピーに対して行ったのと同様に、最大成分が対角線上に位置する行の割合を測定せよ。
   2. 同じブロック 2 のヘッドにおける直接経路 $\mathbf{E}\,\mathbf{W}_{\mathrm{QK}}^{2m'}\,\mathbf{E}^{\top}$ と比較せよ。どのヘッド対が照合を担っているか。また、それらはアテンションに基づく `head_scores` のスコアと一致するか。
   3. 上の計算は、RoPE が QK 回路の内部で適用する回転を無視している。固定オフセットに対する相対回転 $\mathbf{R}_{j-i}$ を含め、オフセット -23 でこれを行うと対のランキングが変化するかどうかを確認せよ。

7. [code] **勾配ステップを実行するヘッド**。●誘導ヘッドはコピーによってコンテキストから学習する。モデルはコンテキスト内のラベル付き例から規則を推論することもできる。系列が対 $(\mathbf{x}_i, y_i)$（$i = 1, \dots, n$）を保持し、ここで標準正規分布に従う $\mathbf{x}_i \in \mathbb{R}^p$ に対して $y_i = \mathbf{w}^{\top}\mathbf{x}_i$（$\mathbf{w}$ は系列ごとに新たに抽出されるベクトル）であり、その後にラベルを予測すべきクエリ $\mathbf{x}_q$ が続くとする。
   1. $\frac{1}{2}\sum_i (y_i - \mathbf{w}^{\top}\mathbf{x}_i)^2$ に対する勾配降下法を $\mathbf{w} = \mathbf{0}$ から学習率 η で 1 ステップ行うと、予測 $\widehat{y}_q = \eta \sum_i y_i\, \mathbf{x}_i^{\top}\mathbf{x}_q$ が得られることを示せ。これをソフトマックスなしのアテンションとして解釈したとき、クエリ、キー、値の役割を果たすのはそれぞれ何か。ま

た、10.6 節の線形アテンション（特徴マップ $\phi(\mathbf{x}) = \mathbf{x}$）と比較せよ。

2. 各例にトークン $\mathbf{z}_i = [\mathbf{x}_i; y_i] \in \mathbb{R}^{p+1}$ を、クエリにトークン $\mathbf{z}_q = [\mathbf{x}_q; 0]$ を与える。クエリトークンにおける出力が $\hat{y}_q$ となるよう、ソフトマックスなしの 1 つのヘッドのクエリ写像、キー写像、値写像を定めよ。クエリトークンが自身にもアテンションを向けることが問題にならないのはなぜか。このヘッドを実装し、勾配ステップと一致することを確認せよ。このヘッドの重みは $\mathbf{w}$ に依存するか。
3. $\eta = 1/n$ のとき、このステップにより $\mathbf{w}_1 = \left(\frac{1}{n}\sum_i \mathbf{x}_i \mathbf{x}_i^\top\right)\mathbf{w}$ および $\mathbb{E}\,\|\mathbf{w}_1 - \mathbf{w}\|^2 = \frac{p+1}{n}\|\mathbf{w}\|^2$ が成り立つことを示せ。ヒント：標準正規分布に従う $\mathbf{x}$ に対して $\mathbb{E}[(\mathbf{x}^\top\mathbf{w})^2\|\mathbf{x}\|^2] = (p+2)\|\mathbf{w}\|^2$ である。これより、$\hat{y}_q$ の期待二乗誤差（y_q の分散に対する相対値）が $(p+1)/n$ となることを結論づけよ。
4. $p = 8$ に対し、500 本の系列にわたる相対誤差を $n = 10$、100、1000 で測定し、理論予測 $9/n$ と比較せよ。$n = 10$ のとき、このヘッドは常にゼロを予測する場合と比べてどうなるか。

8. [extended] **テキストにおける誘導ヘッド。**●●Olsson et al. (2022) は、言語モデルにおける誘導ヘッドの形成を、コンテキストの前半位置から後半位置にかけての損失の低下量として測られるコンテキスト内学習の急激な向上と結びつけている。これを 10.4.5.2 節のバイトレベルモデルで検証せよ。
   1. `d2l.TinyCharLM`（`pos='rope'`）を 10.4.5.2 節の WikiText-103 のバイト列上で、1 ラウンドあたり 100 ステップの `d2l.train_steps` を 30 ラウンド実行して訓練せよ。各ラウンドの後に 2 つの量を記録せよ。1 つ目は、テキスト語彙上のパターン長 16〜32 の反復ランダム系列に対して `head_scores` で計算した、ブロック 2 のヘッドにおける誘導ターゲットスコアの最大値である。2 つ目はコンテキスト内利得、すなわちターゲット位置 0〜15 での検証損失から位置 96〜127 での検証損失を引いた差である。
   2. 両方の量を訓練ステップに対してプロットせよ。誘導スコアとコンテキスト内利得は同じ期間に上昇するか。
   3. 訓練後、誘導スコアが最も高いブロック 2 のヘッド群を `ablate` で除去し、コンテキスト内利得と検証損失を再度測定せよ。これらのヘッドは利得のどれだけの割合を担っているか。
   4. 反復ランダム系列には反復以外の構造がないのに対し、テキストでは単語や句が不規則な間隔で繰り返される。もしテキストに重複する単語が一切含まれなかったら、結果のどれが変化するか。また、それはなぜか。

10.8 関連文献

ここまでの各節では、層の各構成要素について代表的な設計を 1 つずつ展開してきた。すなわち、ソフトマックスルックアップ（10.1 節）、スケール付きドット積によるスコア計算とマスキング（10.2 節）、マルチヘッドおよびグループ化クエリ自己アテンション（10.3 節）、位置エンコーディング（10.4 節）、タイル化オンラインソフトマックス（10.5 節）、スライディングウィンドウアテンションと線形アテンション（10.6 節）、そして機構的回路（10.7 節）である。本節では、これらと密接に関連する派生手法、現代のオープンウェイトモデルにおけるアーキテクチャ上の改良、および一次文献を概観する。

10.8.1 ルックアップとその正規化

10.8.1.1 連想記憶としてのアテンション

Ramsauer et al. (2021) は、ソフトマックスアテンションの 1 ステップが、キーと値の行を記憶パターンとする連続値**現代的ホップフィールドネットワーク**（modern Hopfield network）の 1 回の更新に対応することを示し、ソフトマックスアテンションを古典的な連想記憶（associative

memory）と結びつけた。逆温度（カーネル回帰における $1/\sigma^2$、ドット積アテンションにおける $1/\sqrt{d}$）によって、不動点（fixed point）が個々の記憶パターンになるか（高逆温度）、複数のパターンにわたる準安定な平均になるか（低逆温度）が決まる。この視点から同研究は、BERT などの事前学習済みエンコーダにおいて、浅い層は主として広範な平均化レジームで動作し、深い層は鋭い検索レジームで動作することを観察している。適応的カーネルバンド幅 (Silverman 1986) は、クエリまたはヘッドに依存するソフトマックス温度に対応する。

10.8.1.2 単調アライメント

Bahdanau et al. (2015) が機械翻訳向けに制約のない内容ベースのアライメントを導入する 1 年前、Graves (2013) は手書き文字合成向けに微分可能な**単調**（monotone）位置ベースのアテンション機構を導入し、平均位置がペンストロークごとに単調に前進する混合ガウス窓をパラメータ化した。これは音声や手書き文字には適した帰納バイアス（inductive bias）であるが（(Rabiner and Juang 1993)）、翻訳における語順の逆転に対しては制約が強すぎる（10.2.4 節）。

10.8.1.3 総和が 1 である必要のない重み

ソフトマックスの各行は和が 1 にならなければならないため、関連するキーを持たないクエリは、確率質量を先頭トークンへと集中させることが多い。これが 10.2.3.2 節および 10.7 節で見た**アテンションシンク**である（11.3.3.3 節も参照）。M. Sun et al. (2024) は、アテンションシンクを残差ストリーム内で暗黙的バイアス（implicit bias）項として働く巨大な外れ値の活性化値と関連づけており、一方 Barbero, Arroyo, et al. (2025) は、シンクが無操作（no-op）のターゲットを提供することで、深い長文脈 Transformer における表現崩壊（representation collapse）を防いでいると論じている。

近年のいくつかの設計では、この総和 1 の制約を緩和し、入力トークンのシンクを使わずにヘッドを待機状態にできるようにしている。

- **シグモイドアテンション** (Ramapuram et al. 2025) は、行ごとのソフトマックスを $-\log n$ だけシフトした要素ごとのロジスティックシグモイドに置き換え、重みを $1/(n+1)$ 付近に初期化する。
- **ゲート付きアテンション**（gated attention、(Qiu et al. 2025)）は、Qwen3-Next (Qwen Team 2025) で採用されており、各ヘッドのソフトマックス出力に入力依存のシグモイドゲートを乗じる。
- **学習可能なシンクロジット**（learnable sink logits）は、`gpt-oss` (OpenAI 2025) で採用されており、対応する値ベクトルを持たない訓練可能な仮想キースコアをソフトマックスの分母に追加する。
- **差分アテンション**（differential attention、(T. Ye et al. 2025)）は、ヘッドごとに 2 つのソフトマックス分布の差をとることで、無関係なキーに対するコモンモードの背景重みを相殺する。

10.8.2 自己アテンションとヘッド

10.8.2.1 起源とヘッドの冗長性

自己アテンションは、**系列内アテンション**（intra-attention）を伴う再帰構造 (Cheng et al. 2016)、自然言語推論（natural language inference）向けの再帰を用いない分解可能アテンション (Parikh et al. 2016)、および多様性ペナルティを伴う複数行の文埋め込み (Lin et al. 2017) から発展し、その後に Vaswani et al. (2017) が Transformer において射影付きマルチヘッド自己アテンションとして統合した。

単一混合のボトルネック（10.3.2 節）を克服するには複数のヘッドが不可欠であるが、訓練済みモデルはテスト時に大きなヘッドの冗長性を示すことが多い。実際、Michel et al. (2019) および Voita et al. (2019) は、BERT や NMT モデルにおける大半のヘッドを最小限の性能劣化で事後的に枝刈りでき、位置・構文・低頻度語を担う中核的なヘッドだけが残ることを示した。現代のアーキテクチャでは、グループ化クエリアテンション（grouped-query attention、10.3.3.6 節、11.3.3 節）によって、訓練前の段階からパラメータとキャッシュの効率化が図られている。

10.8.3 位置

10.8.3.1 学習される相対位置

RoPE の登場以前は、オフセットごとに訓練可能なキー埋め込み (Shaw et al. 2018)、Music Transformer におけるスキューインデックス付き相対ロジット (Huang et al. 2018)、あるいは T5 における対数距離バケットごとのスカラーバイアス (Raffel et al. 2020) を通じて相対オフセットが注入されていた。その後、ALiBi は学習されるバケットを固定の線形スロープに置き換え、RoPE はパラメータを持たない座標回転によって、オフセットのみに厳密に依存するスコアを実現した。

10.8.3.2 訓練済みモデルにおける RoPE

オープンウェイトモデルは、より長いコンテキストに対応するため、RoPE の基底周波数 b を段階的に引き上げてきた。具体的には、Llama 2 の $b = 10^4$ から、Llama 2 Long (Xiong et al. 2024) や Llama 3 (Grattafiori et al. 2024) では 5×10^5 へ、さらに Gemma 3 (Gemma Team 2025) のグローバル層では 10^6 へと拡大している。b を大きくすると、波長が訓練ウィンドウを超える低速レジームへとより多くの座標ペアが移行する（10.4.5.3 節）。Barbero, Vitvitskyi, et al. (2025) は、訓練済みモデルが高周波の RoPE ペアを位置ヘッド（直前トークンヘッドなど）に用い、低周波のペアをほぼ位置不変な意味チャネルとして用いていることを示した。これが、極端な距離で意味チャネルがドリフトしないよう最低周波成分を完全にゼロにする **p-RoPE** の動機となっている。YaRN (Peng et al. 2024) を超える事後学習でのコンテキスト拡張として、**LongRoPE** (Ding et al. 2024) は次元ごとの再スケーリング係数に対する進化的探索を用い、コンテキストウィンドウを最大 200 万トークンまで拡張している。

10.8.3.3 長さ汎化と長文脈の活用

T.-C. Chi et al. (2023) は ALiBi の有効受容野を解析し、平坦なパープレキシティ外挿が長距離の検索ではなく局所的なウィンドウ処理の反映であるという 10.4.5.2 節の結果を裏づけている。位置エンコーディングを持たない因果モデル（NoPE、(Haviv et al. 2022)）については、Jie Wang et al. (2024) が、系列長とともにアテンションエントロピーが増大すること、またヘッドごとの温度スケーリングによってこれを打ち消せることを示した。この仕組みは Llama 4 (Meta AI 2025) の NoPE グローバル層において推論時に採用されている。最後に、長文ドキュメント全体でのパープレキシティが低いからといって、ウィンドウ全体にわたる有効な検索が保証されるわけではない。実際、N. F. Liu et al. (2024) は、関連する事実が長いコンテキストの先頭や末尾付近ではなく中央付近に現れると正解率が大幅に低下する「**中間での埋没**」（lost in the middle）現象を実証している（21.5 節（英語版） 参照）。

10.8.4 コスト

10.8.4.1 初期の線形計算量近似

FlashAttention の登場以前は、数多くのアーキテクチャが $n \times n$ のアテンション行列を近似していた。たとえば、**Performer** (Choromanski et al. 2021) は正値直交ランダム特徴量を用いて指

数カーネルを近似し、**Linformer** (Wang et al. 2020) は低ランクの系列射影を適用し、**Reformer** (Kitaev et al. 2020) は局所性鋭敏型ハッシュ（locality-sensitive hashing）を用い、**Longformer** (Beltagy et al. 2020) と **Sparse Transformer** (Child et al. 2019) は局所ウィンドウにストライド付きトークンやグローバルトークンを組み合わせた（Tay et al. (2020) によるサーベイを参照）。

10.8.4.2 公開モデルにおけるウィンドウと内容適応型スパース性

Mistral 7B (Jiang et al. 2023) はすべての層にスライディングウィンドウを適用していたが（後に (Mistral AI 2023) で廃止）、後続のアーキテクチャでは局所ウィンドウと周期的なグローバル層を交互に配置している。たとえば、Gemma 2 (Gemma Team 2024) では 1:1、Gemma 3（$w = 1{,}024$、(Gemma Team 2025)）では 5:1、`gpt-oss` (OpenAI 2025) では $w = 128$ で 1:1 となっており、Llama 4 (Meta AI 2025) ではチャンク化された局所層と NoPE グローバル層を組み合わせている。

固定の位置ウィンドウにとどまらず、長文脈の訓練とデコーディングに向けて、ハードウェアに適合した**内容ベースのスパースアテンション**（content-based sparse attention、31.5 節（英語版））も登場している。その例として、**Native Sparse Attention**（NSA、(Yuan et al. 2025)）、**Mixture of Block Attention**（MoBA、(E. Lu et al. 2025)）、および **DeepSeek Sparse Attention** (DeepSeek-AI et al. 2025) が挙げられる。

10.8.4.3 FlashAttention-2、FlashAttention-3、FlexAttention

FlashAttention (Dao et al. 2022) に続く **FlashAttention-2** (Dao 2024) は、クエリの系列次元にわたって並列化し、キーループの最後まで正規化を遅延させる。Hopper GPU 上では、**FlashAttention-3** (Shah et al. 2024) が非同期の Tensor Core 行列積とワープ特化型のソフトマックス指数計算をオーバーラップさせ、インコヒーレントな直交事前回転によって FP8 をサポートする。**FlexAttention** (Dong et al. 2025) は任意のスコア変更クロージャとブロックマスクを融合 Triton カーネルへとコンパイルし、一方で並列 KV キャッシュデコーディングカーネル（29.2 節（英語版））はキー軸を分割して、オンラインソフトマックスにより部分状態 $(m_B, s_B, \mathbf{o}_B)$ を縮約する。

10.8.4.4 ゲート付き再帰とハイブリッド

現代の線形アテンションの派生手法は、第 12 章で扱う 2 つの改良によって記憶容量の課題に対処している。すなわち、(1) データ依存の忘却ゲート（**Gated Linear Attention**、(S. Yang, Wang, Shen, et al. 2024)、12.6 節）と、(2) 誤差修正型のデルタ更新（**DeltaNet**、(Schlag et al. 2021; S. Yang, Wang, Zhang, et al. 2024)、および **Gated DeltaNet**、(S. Yang et al. 2025)、12.7 節）である。実用アーキテクチャでは、厳密な連想想起を維持するために、これらのリカレント層と周期的な完全ソフトマックスアテンション層（通常は 4 層に 1 層から 8 層に 1 層の割合）を組み合わせている (Qwen Team 2025; Kimi Team 2025b; MiniMax 2025)（12.8 節）。

10.8.5 大規模モデルにおける回路

10.8.5.1 活性化パッチング、重ね合わせ、説明

ゼロ・平均アブレーション（10.7.3.3 節）の先へ進む手法として、**活性化パッチング**（activation patching）はクリーンな実行の活性化値を破損させた実行へと代入することで因果経路を切り分ける (K. Meng et al. 2022; K. Wang et al. 2023)。この手続きは ACDC (Conmy et al. 2023) や 1 次の**帰属パッチング**（attribution patching、(Nanda 2023; Kramár et al. 2024; Zhang and Nanda 2024; Heimersheim and Nanda 2024)）によって自動化されている。特徴量の数が残差ストリームの次元数を上回る**重ね合わせ**（superposition、(Elhage et al. 2022)）の状況では、**スパー**

スオートエンコーダと帰属グラフが多義的（polysemantic）な方向を単一意味（monosemantic）の特徴量へと分離する (Bricken et al. 2023; Cunningham et al. 2024; Ameisen et al. 2025)（18.6 節（英語版））。また、こうした介入的ツールは、アテンション重みが忠実な説明になり得るかをめぐる Jain and Wallace (2019) と Wiegreffe and Pinter (2019) の古典的な論争にも決着をつけている。

10.8.5.2 アルゴリズムとしてのコンテキスト内学習

誘導ヘッド（induction head）によるコピー (Olsson et al. 2022) と並び、コンテキスト内回帰の問題族で訓練された Transformer は、勾配降下法、リッジ回帰、および最小二乗法のソルバを順伝播の中で直接実装できる (Garg et al. 2022; Oswald et al. 2023; Akyürek et al. 2023)。ただし、一般的なテキスト事前学習がどの程度まで暗黙的な勾配降下法を誘導するかについては、依然として議論が続いている (Shen et al. 2024)。

10.8.6 推奨される参考資料

- **視覚的な入門資料：**3Blue1Brown の **Attention in Transformers, Step-by-Step**（2024）、Jay Alammar の **The Illustrated Transformer**（2018）、Peter Bloem の **Transformers from Scratch**（2019）、Lilian Weng の **The Transformer Family Version 2.0**（2023）。
- **基礎および回路に関する専門文献：**Elhage らの **A Mathematical Framework for Transformer Circuits**（2021）、Olsson らの **In-Context Learning and Induction Heads**（2022）、Weiss・Goldberg・Yahav の **Thinking Like Transformers**（2021）およびこれに対応する Sasha Rush の **Transformer Puzzles**。

Chapter 11

Transformer

第 10 章では、マルチヘッドアテンション層、位置エンコーディング、および長文脈へスケールするアテンションの派生手法を構築した。しかし、アテンション単体ではトークン混合演算子（token-mixing operator）にすぎない。トークンごとの非線形変換、残差接続、および正規化がなければ、アテンション層を積み重ねても訓練可能な深層ネットワークは得られないからである。本章では、アテンションを完全な **Transformer** アーキテクチャへと組み上げ (Vaswani et al. 2017)、その訓練・サービング・スケーリングの仕組みを考察する。

まず、因果的スタックを一から構築して訓練することから始める。11.1 節では、Pre-Norm 残差ストリーム（`RMSNorm` と `SwiGLU`）の内部でマルチヘッドアテンションと位置ごとのフィードフォワードネットワークを対にし、残差射影のスケーリング（$1/\sqrt{2L}$）およびクエリ・キー正規化（query–key normalization）が深さ方向にわたって信号ノルムとアテンションロジットを有界に保つ理由を示す。11.2 節では、これらの因果的ブロックをトークン埋め込みと重み共有（weight tying）を用いた言語モデリングヘッドの間に積み重ね、条件を揃えたリカレントベースラインと比較しながら WikiText-103 上でバイトレベルの GPT を訓練する。さらに、事前学習済みの GPT-2 の重み（`124M`）を読み込んで温度と top-k サンプリングを検証し、現代のオープンウェイト LLM に共通するアーキテクチャのレシピを抽出する（11.2.6 節）。素朴な自己回帰生成ではステップごとにプレフィックス全体を再計算するため、11.3 節ではキー・値キャッシュ（key–value cache、KV キャッシュ）を導入し、ハードウェアのルーフライン上で計算律速（compute-bound）なプレフィルとメモリ帯域幅律速（memory-bandwidth-bound）なデコードを対比させたうえで、ヘッド数（グループ化クエリアテンションとマルチクエリアテンション）、チャネル幅（低ランクのキー・値射影）、および系列長（アテンションシンクを伴うスライディングウィンドウ）にわたってキャッシュを圧縮する。

すべてのタスクが因果的なテキスト生成というわけではない。11.4 節では、因果的デコーダを、マスク言語モデリング（masked language modeling）によって訓練される双方向エンコーダ (Devlin et al. 2018) や、クロスアテンションで結合された系列対系列のエンコーダ・デコーダ（固定サイズの潜在ボトルネックを含む (Jaegle et al. 2021)）と対比させる。続く 11.5 節では、重複のないピクセルパッチをトークン埋め込みへ射影することで双方向エンコーダを画像に直接適用し (Dosovitskiy et al. 2021)、学習された 2D 位置構造とデータ効率を、パラメータ数を揃えた ResNet と比較する。

最後に、スケーリングと条件付き計算（conditional computation）に目を向ける。11.6 節では、パラメータ数（$N \approx 12Ld^2$）と訓練 FLOPs（$C \approx 6ND$）を数え上げ、15 モデルのグリッドを訓練し、公開されている Chinchilla のべき乗則フィット (Kaplan et al. 2020; Hoffmann et al. 2022) を用いて IsoFLOP 曲線と計算量最適（compute-optimal）な配分を描く。あわせて、運用

期間全体の推論コスト（inference cost）を考慮すると、現代の事前学習がなぜ Chinchilla のパラメータあたりトークン比をはるかに超えて訓練された、より小さなモデルへと向かうのかを説明する。さらに 11.7 節では、密なフィードフォワードネットワークをルーティングされるエキスパート群に置き換えることで保持パラメータ容量とトークンあたり FLOPs を切り離し (Shazeer et al. 2017; Fedus et al. 2022)、補助損失による負荷分散と損失を用いないルータバイアス制御を比較する (L. Wang et al. 2024)。そして 11.8 節では、正規化の配置、潜在アテンション（latent attention）、および投機的デコーディング（speculative decoding）にわたる拡張手法を概観する。

11.1 Transformer ブロック

自己アテンション（self-attention、第 10 章）は、系列の位置間で情報を混合する。すなわち、各トークンの出力は、アテンションを向ける位置の値ベクトルの重み付き平均となる。しかし、アテンション層単体では、その重み付き和の前後に線形射影（W_Q, W_K, W_V, W_O）を適用するにすぎない。位置ごとの非線形性、残差接続、および正規化がなければ、アテンション層を積み重ねてもトークンごとの階層的な特徴量を構築できず、数十層にわたって信号を安定して伝播させることもできない。

Transformer ブロック (Vaswani et al. 2017) は、マルチヘッドアテンションと位置ごとの**フィードフォワードネットワーク（FFN）**を組み合わせ、各サブ層を加算的な**残差接続**（residual connection）で包み (He et al. 2016a)、**層正規化**（layer normalization）によって活性化値を安定化させる（Figure 11.1）。現代のブロックと 2017 年の元の Transformer を分ける設計上の改良点は、次の 3 つである。

1. **正規化の配置：**正規化を残差ストリーム（residual stream）自体（**Post-Norm**）から各サブ層の入力（**Pre-Norm**）へと移すことで、深さ方向に減衰しない恒等経路が保たれる。
2. **正規化の形式：**平均中心化を伴う `LayerNorm` を二乗平均平方根（root-mean-square）による正規化（`RMSNorm`）と任意のヘッドごとの**クエリ・キー正規化**（query–key normalization、`QK-Norm`）に置き換えることで、メモリ帯域幅のオーバーヘッドを抑えつつ活性化値とアテンションロジットを有界に保つ。
3. **ゲート付きフィードフォワードネットワーク：**2 行列の ReLU・GELU MLP（$d_{\text{ff}} = 4d$）をゲート付きの `SwiGLU` ネットワーク（$d_{\text{ff}} \approx \frac{8}{3}d$）に置き換えることで、まったく同じ $8d^2$ のパラメータ数と FLOP 予算のまま表現の質を高める。

11.1.1 Transformer ブロックの構成要素

Transformer ブロックは、共有された**残差ストリーム** $\mathbf{X} \in \mathbb{R}^{B\times n\times d}$（10.7 節）上で動作する。連続する 2 つのサブ層は $\mathbf{X}$ を上書きするのではなく、ストリームから値を読み取り、更新量を計算して、その結果をストリームに加算して戻す。

- **マルチヘッドアテンション（トークン混合、**token mixing）は、系列の位置どうしが相互作用する唯一の箇所である。各トークンは系列にクエリを送り（10.3 節）、取得したコンテキストベクトルを自身のストリームに加算する。
- **位置ごとのフィードフォワードネットワーク（チャネル混合、**channel mixing）は、同じ MLP を各トークンベクトル $\mathbf{x}_i \in \mathbb{R}^d$ に独立に適用し、アテンションが集約した特徴量を変換する。密なブロックのパラメータ数の約 3 分の 2 は FFN に存在する。

現代の Transformer で用いられる **Pre-Norm** 配置（Figure 11.1）では、各サブ層は更新量を計算する前に自身の入力を正規化する。

$$\begin{aligned}\mathbf{H} &= \mathbf{X} + \mathrm{Attn}\big(\mathrm{Norm}(\mathbf{X})\big),\\ \mathbf{Y} &= \mathbf{H} + \mathrm{FFN}\big(\mathrm{Norm}(\mathbf{H})\big).\end{aligned} \tag{11.1}$$

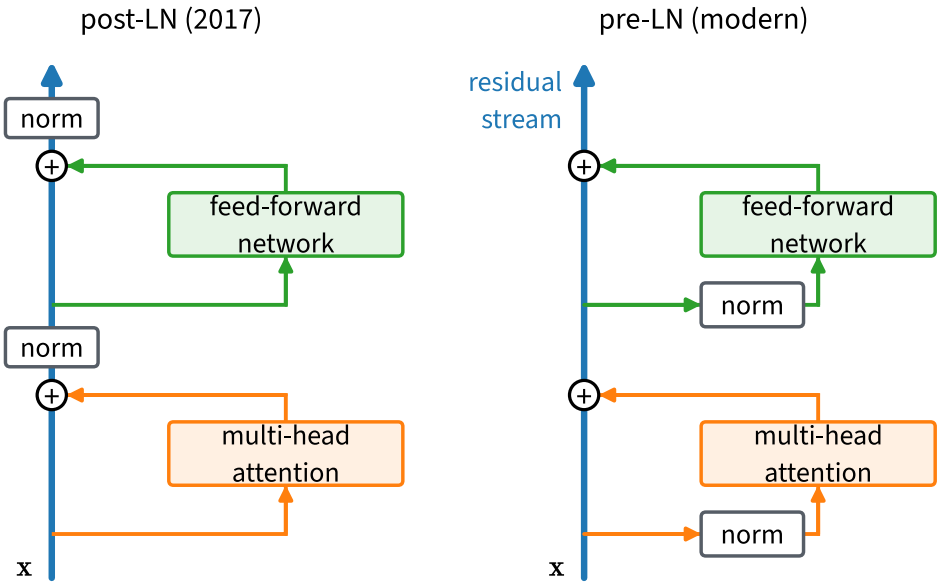


Figure 11.1: Transformer ブロック。アテンションと位置ごとの FFN が残差ストリームから値を読み取り、計算結果をストリームに加算して戻す。左は 2017 年の配置であり、正規化は各加算の後にストリーム自体の上に置かれる。右は現代の配置であり、正規化は各ブランチの上へと移り、ストリームは入力から出力まで途切れることなく通る。

11.1.2 正規化

11.1.2.1 Pre-Norm と Post-Norm

2017 年の元の Transformer (Vaswani et al. 2017) と BERT (Devlin et al. 2018) は **Post-Norm** を採用しており、各加算の後に**残差ストリーム自体の上**に正規化を置いていた。

$$\begin{aligned}\mathbf{H} &= \mathrm{Norm}\big(\mathbf{X} + \mathrm{Attn}(\mathbf{X})\big),\\ \mathbf{Y} &= \mathrm{Norm}\big(\mathbf{H} + \mathrm{FFN}(\mathbf{H})\big).\end{aligned} \tag{11.2}$$

なぜ現代のアーキテクチャは Equation 11.2 から Equation 11.1 へと切り替わったのだろうか。最小限のブロック `MiniBlock`（フラグ `pre_norm` で 2 つの配置を切り替える）を使えば、1 ステップも訓練することなく初期化の段階でその違いを直接確認できる。

```
class MiniBlock(nnx.Module):
    """Attention + MLP on a residual stream; norm placement as a flag."""
    def __init__(self, num_hiddens, pre_norm, *, rngs):
        self.pre_norm = pre_norm
        self.attention = d2l.MultiHeadAttention(num_hiddens, 4, rngs=rngs)
```

```
        self.W_1 = d2l.Linear(num_hiddens, 4 * num_hiddens, rngs=rngs)
        self.W_2 = d2l.Linear(4 * num_hiddens, num_hiddens, rngs=rngs)
        self.norm1 = nnx.LayerNorm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.norm2 = nnx.LayerNorm(num_hiddens, epsilon=1e-5, rngs=rngs)

    def ffn(self, X):  # exact GELU
        return self.W_2(jax.nn.gelu(self.W_1(X), approximate=False))

    def __call__(self, X):
        if self.pre_norm:
            Y = self.norm1(X)
            X = X + self.attention(Y, Y, Y)
            return X + self.ffn(self.norm2(X))
        X = self.norm1(X + self.attention(X, X, X))
        return self.norm2(X + self.ffn(X))
```

11.1.2.2 初期化時の信号伝播とトークン崩壊

単位分散をもつランダムなトークンベクトルのバッチ $\mathbf{X}^{(0)} \in \mathbb{R}^{B\times n\times d}$ を、初期化したばかりの $L = 32$ 個のブロックに通すことを考えよう。以下の関数 `signal_stats` は、積層された各ブロックのすべての重み行列を、2017 年の Transformer と同じ Xavier 初期化で、あるいは $\mathcal{N}(0, \text{std}^2)$ の要素（`std` が指定された場合）で新たにサンプリングする。各ブロック $\ell \in \{1, \dots, L\}$ において、次の 3 つの量を追跡する。

1. **ストリーム標準偏差**（$\mathbf{X}^{(\ell)}$ の全要素にわたる標準偏差）。
2. **トークンの広がり**（token spread）。n 個の系列位置にわたる標準偏差として測られ、n 個すべてのトークンが同じベクトルへと崩壊すると消失する。
3. **クエリ勾配ノルム** $\|\partial \mathcal{L}/\partial W_q^{(\ell)}\|_F$（スタック最上部の単位スカラー損失から逆伝播してくる勾配のノルム）。

```
def signal_stats(pre_norm, std=None, num_blks=32, num_hiddens=256):
    """Token spread, stream RMS, and gradient norms of a fresh stack whose
    weight matrices are Xavier, or N(0, std^2) if std is given."""
    rngs = nnx.Rngs(d2l.get_key())
    stack = nnx.Sequential(*[MiniBlock(num_hiddens, pre_norm, rngs=rngs)
                             for _ in range(num_blks)])
    init = (nnx.initializers.xavier_uniform() if std is None  # as in 2017
            else nnx.initializers.normal(std))
    for _, m in nnx.iter_graph(stack):
        if isinstance(m, nnx.Linear):
            m.kernel[...] = init(d2l.get_key(), m.kernel.shape)
    X = jax.random.normal(d2l.get_key(), (2, 64, num_hiddens))
    R = jax.random.normal(d2l.get_key(), X.shape)
    # Full fp32 products: on a GPU, JAX's default TF32 rounding (about 1e-3)
    # would swamp token differences of 1e-4
    with jax.default_matmul_precision('highest'):
        H, spread, rms = X, [], []
        for blk in stack.layers:
            H = blk(H)
            spread.append(jnp.linalg.norm(H - H.mean(1, keepdims=True))
                          / jnp.linalg.norm(H))
```

```
        rms.append(jnp.sqrt((H ** 2).mean()))
    g = nnx.grad(lambda m: (m(X) * R).mean())(stack)  # generic readout
gnorm = lambda k, n: jnp.linalg.norm(
    g['layers'][k]['attention'][n]['kernel'][...])
grad_q = [gnorm(k, 'W_q') for k in range(num_blks)]
proj = {n: float(gnorm(num_blks - 1, n))
        for n in ('W_q', 'W_k', 'W_v', 'W_o')}
return {'spread': jnp.stack(spread).tolist(),
        'rms': jnp.stack(rms).tolist(),
        'grad_q': jnp.stack(grad_q).tolist(), 'proj': proj}
```

なぜ Post-Norm では深さとともにトークンの多様性が失われるのだろうか。各トークンを全トークンが共有する成分と固有の成分に分け、$\mathbf{x}_i = \mu + \delta_i$ と書く。ここで $\mu = \frac{1}{n}\sum_j \mathbf{x}_j$ は n 個の位置にわたる平均である（Figure 11.2）。初期化時、アテンション重みはほぼ一様であるため、どのトークンもほぼ同じ更新量 $\mathbf{u} \approx \mathbf{W}_o\mathbf{W}_v\mu$ を受け取る。すなわち、偏差は平均化されて打ち消し合い、平均成分だけが先へ伝わる。2 つの射影はそれぞれ入力の長さの二乗を $g^2 = d\sigma^2$ 倍（σ^2 は重みの各要素の分散）にし、確率ベクトル $\mathbf{u}$ は μ とほぼ直交する。したがって、共有成分のエネルギーは $1 + g^4$ 倍に増大する一方、偏差は元のままである。続く正規化ではこれを修復できない。すべてのトークンを同じ係数で再スケーリングするため、ストリームの長さは元に戻るものの、平均と偏差のバランスは戻らないからである。

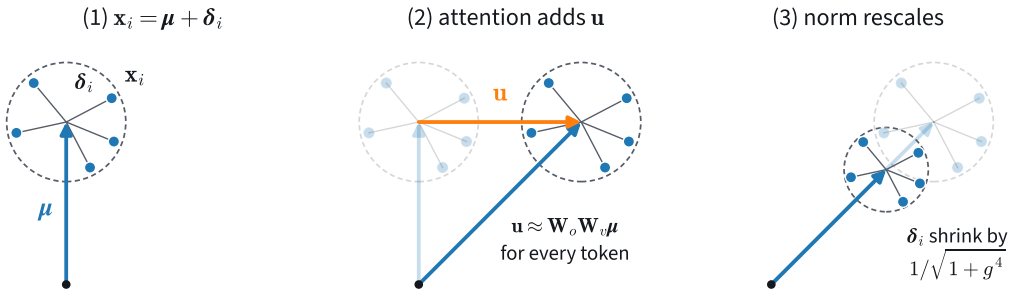


Figure 11.2: 1 つの Post-Norm ブロックにおけるトークン崩壊。各トークンは共有平均 μ と固有の偏差 δ_i に分解される。ほぼ一様なアテンションがすべてのトークンに同じ更新量 $\mathbf{u} \approx \mathbf{W}_o\mathbf{W}_v\mu$ を加算するため、平均だけが増大する。続く正規化が全トークンを単一の係数で再スケーリングすることで、偏差が縮小する。Xavier 初期化（$g = 1$）の場合を図示しており、このとき $\mathbf{u}$ は μ と同じ長さをもち、係数は $1/\sqrt{2}$ となる。

共有エネルギーとトークン固有のエネルギーの比を $\rho = n\|\mu\|^2 / \sum_i \|\delta_i\|^2$ とする。各ブロックは ρ を $1 + g^4$ 倍にする。ストリームのノルムのうち偏差が占める割合であるトークンの広がりは $s = (1+\rho)^{-1/2}$ であり、n 個の独立なランダムトークンは $\rho_0 = 1/(n-1)$ から出発する。したがって、k ブロック後には次のようになる。

$$s_k \approx \left(1 + \frac{(1+g^4)^k}{n-1}\right)^{-1/2}. \tag{11.3}$$

ひとたび平均が支配的になると、広がりはブロックごとに $1/\sqrt{1+g^4}$ 倍に縮小し、その値は $1/\sqrt{2}$ となる（Xavier 初期化、すなわち $g = 1$ の場合）。なお、Equation 11.3 に FFN は現れな

い。FFN は各トークンに個別に作用し、平均と偏差を区別できないため、両者の比をほとんど変えないからである（演習問題 5）。

Pre-Norm（Equation 11.1）のもとでは、残差ストリームが正規化の分母で割られることはない。L 個のブロックを展開すると、直接の和 $\mathbf{X}^{(L)} = \mathbf{X}^{(0)} + \sum_{\ell=1}^{2L} \mathbf{U}^{(\ell)}$ となる。$2L$ 個の独立なサブ層出力がストリームに分散を加えるため、スケーリングなしの Pre-Norm ストリームの RMS は深さとともに $\sqrt{1 + 2\ell\sigma^2}$ のように増大する（全 $2L$ 個のサブ層が加える合計分散が $O(1)$ となるよう任意の深さ L で抑えるため、GPT-2 (Radford et al. 2019) は各ブロックの残差出力射影 `W_o` と `W_2` の初期化標準偏差を $1/\sqrt{2L}$ 倍にスケーリングしており、これは 11.2 節で実装し、演習問題 4 で検証する）。一方、Post-Norm において BERT (Devlin et al. 2018) は、Equation 11.3 の崩壊を遅らせるためにすべての線形重みをより小さな標準偏差 $\sigma = 0.02$ で初期化する（これにより $g^2 \approx 0.10$、$g^4 \approx 0.01$ となる。ただし $d = 256$ の場合）。これら 3 つの構成（`post-norm`、`pre-norm`、`post-norm, std 0.02`）を 32 ブロックにわたって比較してみよう。

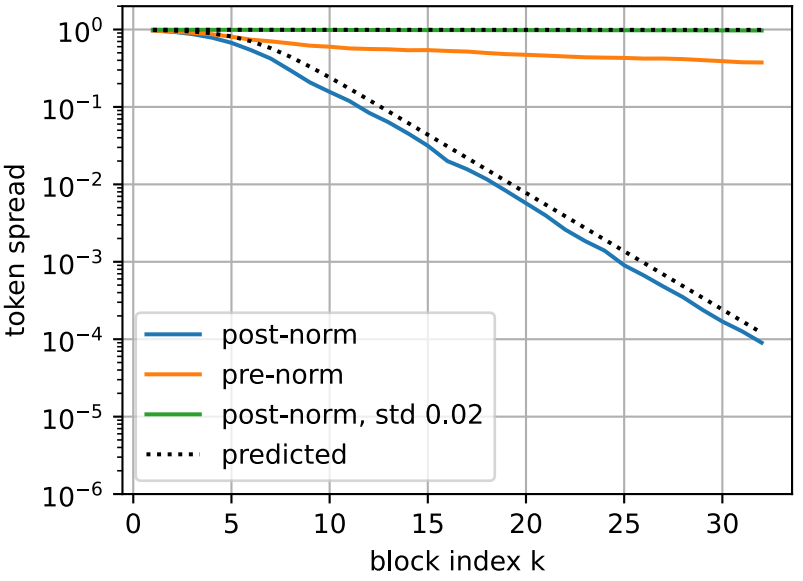
```
arms = {'post-norm': (False, None), 'pre-norm': (True, None),
        'post-norm, std 0.02': (False, 0.02)}
stats, predicted = {}, {}
at = lambda xs, ks=(8, 16, 32): ' '.join(f'{xs[k - 1]:.1e}' for k in ks)
for name, (pre_norm, std) in arms.items():
    stats[name] = s = signal_stats(pre_norm, std)
    print(f'{name}: stream RMS at k=32 {s["rms"][-1]:.2f}')
    print(f'  token spread at k=8,16,32: {at(s["spread"])}')
    if not pre_norm:  # rho_0 = 1/63 for 64 random tokens; Xavier: g = 1
        g2 = 1 if std is None else 256 * std ** 2
        predicted[name] = [(1 + (1 + g2 ** 2) ** k / 63) ** -0.5
                           for k in range(1, 33)]
        print(f'  predicted:                {at(predicted[name])}')
    print(f'  W_q grad norm at k=1,8,16,32: {at(s["grad_q"], (1, 8, 16, 32))}')
    print('  grad norms at k=32: ' + '  '.join(
        f'{n} {g:.1e}' for n, g in s['proj'].items()))
```

```
post-norm: stream RMS at k=32 1.00
  token spread at k=8,16,32: 3.0e-01 2.0e-02 9.0e-05
  predicted:                 4.4e-01 3.1e-02 1.2e-04
  W_q grad norm at k=1,8,16,32: 8.7e-03 1.6e-03 1.3e-05 3.0e-09
  grad norms at k=32: W_q 3.0e-09  W_k 3.6e-09  W_v 6.6e-02  W_o 7.1e-02
pre-norm: stream RMS at k=32 5.11
  token spread at k=8,16,32: 6.6e-01 5.3e-01 3.7e-01
  W_q grad norm at k=1,8,16,32: 5.7e-02 1.9e-02 7.1e-03 2.2e-03
  grad norms at k=32: W_q 2.2e-03  W_k 2.3e-03  W_v 8.4e-02  W_o 7.9e-02
post-norm, std 0.02: stream RMS at k=32 1.00
  token spread at k=8,16,32: 9.9e-01 9.8e-01 9.7e-01
  predicted:                 9.9e-01 9.9e-01 9.9e-01
  W_q grad norm at k=1,8,16,32: 3.9e-04 4.0e-04 4.0e-04 4.0e-04
  grad norms at k=32: W_q 4.0e-04  W_k 4.0e-04  W_v 6.1e-03  W_o 6.2e-03
```

ブロック $k = 1, \ldots, 32$ にわたるトークンの広がりの実測値を、2 つの Post-Norm 条件に対する Equation 11.3 の予測（点線、$n = 64$）と並べてプロットする。

```
d2l.plot(list(range(1, 33)), [s['spread'] for s in stats.values()]
         + list(predicted.values()), 'block index k', 'token spread',
         legend=list(arms) + ['predicted'], yscale='log', ylim=(1e-6, 2),
```

```
        fmts=('C0-', 'C1-', 'C2-', 'k:', 'k:'), figsize=(4.5, 3.3),
        legend_loc='lower left')
```

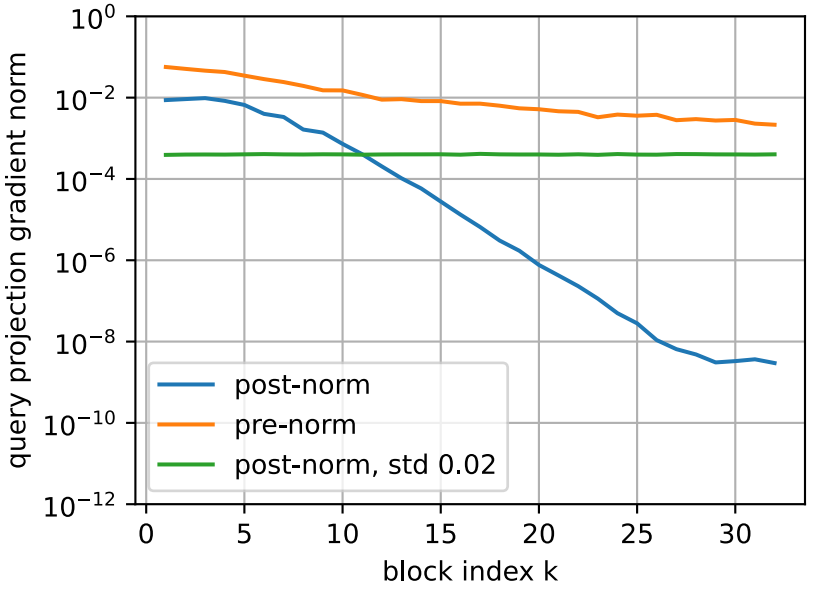


標準的な Xavier 初期化による Post-Norm（`post-norm`）では、トークンの広がりはブロック 32 までにほぼゼロ（$\sim 10^{-4}$）へと崩壊し、Equation 11.3 の予測によく一致する。系列内のすべてのトークンが、ほぼ同一のベクトルへと押し潰されているのである。実測曲線が点線をわずかに下回っているのは、FFN による小さな寄与である。Post-Norm の初期化を縮小すると（`post-norm, std 0.02`）、ブロックごとの収縮が十分に遅くなり、32 ブロックにわたってトークンの広がりが保たれる。Pre-Norm（`pre-norm`）も無縁ではなく、アテンションサブ層は同様に共有平均へ加算する。しかしストリーム自体が深さとともに増大するため、各更新の相対的な重みは前の層より小さくなり、広がりの減衰は緩やかである。Xavier スケールであっても 32 ブロック後に 3 分の 1 以上が残っている（その代わり、ストリームの RMS は $\sqrt{L}$ のオーダーで増大するが、これは $1/\sqrt{2L}$ の出力射影スケーリングと組み合わせることで抑えられる。演習問題 4 を参照）。

11.1.2.3 初期化時の勾配

トークン崩壊（token collapse）は順伝播の性質であるが、どの重みが学習できるかも決定づける。クエリ射影の勾配ノルム $\|\partial \mathcal{L}/\partial W_q^{(\ell)}\|_F$ をブロック番号 $\ell \in \{1, \dots, 32\}$ にわたってプロットする。

```
d2l.plot(list(range(1, 33)), [s['grad_q'] for s in stats.values()],
         'block index k', 'query projection gradient norm',
         legend=list(arms), yscale='log', ylim=(1e-12, 1),
         fmts=('C0-', 'C1-', 'C2-'), figsize=(4.5, 3.3),
         legend_loc='lower left')
```



`post-norm` では、クエリ勾配はブロック 1 で最大（10^{-2} 程度）となり、ブロック 32 では約 10^{-9} まで落ち込む。これは、損失から最も遠い層ほど受け取る勾配が小さくなる深い MLP の勾配消失とは逆の現象である。勾配は逆伝播の途中で失われているのではない。深いブロックが順伝播で受け取る入力に原因があるのである。ブロックに入るトークンどうしがほぼ一致してしまうと、どのキーも同じスコアを受け取り、どの値も同じになるため、クエリを動かしても出力は何も変化しない。その結果、クエリ勾配はトークンの広がりの二乗に比例して縮小し（スコアが偏差 δ_i を通じてしか異ならないことで 1 乗分、値も同様であることでさらに 1 乗分縮小する）、キー勾配もそれに伴って縮小する。他の重みは影響を受けない。ブロック 32 における `W_v` と `W_o` の勾配は、`post-norm` と `pre-norm` で同じ桁の大きさにある（上の各条件の出力の最終行）。

`pre-norm` では、恒等項 I（$\partial \mathbf{X}^{(L)}/\partial \mathbf{X}^{(\ell)} = I + \cdots$ に現れる項）が損失からの勾配をすべてのブロックへと運ぶ。こちらのクエリ勾配も一様ではなく、トークンの広がりが緩やかに減衰するにつれてスタック全体で約 30 分の 1 に低下する。それでも、Post-Norm が 6 桁以上も失うのに対し、10^{-3} 以上を保っている。Xiong et al. (2020) は、初期化時の Post-Norm Transformer における別の不均衡、すなわち出力付近のパラメータに対する勾配が大きくなる現象を解析し、Post-Norm の訓練が学習率のウォームアップに依存する一方で Pre-Norm はそれなしで訓練できる理由がここにあることを示している。

11.1.2.4 RMSNorm

標準的な LayerNorm（7.3 節）は、各トークンベクトル $\mathbf{x} \in \mathbb{R}^d$ からその平均 $\mu = \frac{1}{d}\sum_j x_j$ を引いて中心化し、標準偏差でスケーリングしたうえで、学習されるゲイン γ とバイアス β を適用する。Zhang and Sennrich (2019) は、平均中心化と加算バイアスは訓練の安定性にほとんど寄与せず、重要なのは二乗平均平方根の大きさを正規化することだと示した。μ と β を取り除くと **RMSNorm** が得られる。

$$\begin{aligned}\mathrm{RMSNorm}(\mathbf{x}) &= \frac{\mathbf{x}}{\mathrm{RMS}(\mathbf{x})} \odot \gamma, \\ \mathrm{RMS}(\mathbf{x}) &= \sqrt{\frac{1}{d}\sum_{j=1}^{d} x_j^2 + \epsilon},\end{aligned} \tag{11.4}$$

ここで、$\gamma \in \mathbb{R}^d$ は 1 で初期化され、$\epsilon = 10^{-5}$ である。RMSNorm はトークンごとに平均と分散の両方ではなく二乗平均の縮約 $\frac{1}{d}\sum_j x_j^2$ を 1 回計算するだけで済むため、LayerNorm と同等の正解率を保ちながらメモリ上の縮約演算を減らせる。

```
def rms_norm(X, weight=1, eps=1e-5):
    return X / jnp.sqrt((X * X).mean(-1, keepdims=True) + eps) * weight

x = jnp.array([1.0, 2.0, 3.0, 6.0])
rngs = nnx.Rngs(d2l.get_key())  # both start at unit gain and zero shift
rmsnorm = nnx.RMSNorm(4, epsilon=1e-5, rngs=rngs)
layernorm = nnx.LayerNorm(4, epsilon=1e-5, rngs=rngs)
show = lambda v: ' '.join(f'{a:5.2f}' for a in v.tolist())
print(f'ours vs. nnx.RMSNorm: max deviation '
      f'{jnp.abs(rms_norm(x) - rmsnorm(x)).max():.1e}')
for name, v in (('x', x), ('3x', 3 * x), ('x + 10', x + 10)):
    print(f'{name:>6}: RMSNorm {show(rms_norm(v))} | '
          f'LayerNorm {show(layernorm(v))}')
```

```
ours vs. nnx.RMSNorm: max deviation 2.4e-07
     x: RMSNorm  0.28  0.57  0.85  1.70 | LayerNorm -1.07 -0.53  0.00  1.60
    3x: RMSNorm  0.28  0.57  0.85  1.70 | LayerNorm -1.07 -0.53  0.00  1.60
x + 10: RMSNorm  0.84  0.91  0.99  1.22 | LayerNorm -1.07 -0.53  0.00  1.60
```

11.1.2.5 クエリ・キー正規化（QK-Norm）

Pre-Norm はアテンションサブ層に入る入力 $\mathbf{X}$ を正規化するが、アテンション内部ではクエリ $\mathbf{Q} = \mathrm{Norm}(\mathbf{X})\mathbf{W}_q$ とキー $\mathbf{K} = \mathrm{Norm}(\mathbf{X})\mathbf{W}_k$ に学習される射影行列 $\mathbf{W}_q, \mathbf{W}_k$ が掛けられる。訓練中に $\mathbf{W}_q$ と $\mathbf{W}_k$ の特異値が増大すると、スケール付きドット積ロジット $\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d_h}$ は重みのスケールに対して二次関数的に増大し、ソフトマックスがワンホットな飽和状態に追い込まれて勾配が消失する (Henry et al. 2020; Dehghani et al. 2023)。

QK-Norm は、アテンションスコアを計算する直前に、ヘッドごとの幅 d_h の RMSNorm を $\mathbf{Q}$ と $\mathbf{K}$ に適用する。コーシー・シュワルツの不等式により、単位 RMS をもつ 2 つのベクトルのドット積は大きさが高々 d_h に抑えられるため、スケール付きロジット $\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d_h}$ の絶対値は $\sqrt{d_h}$ 以下に有界となり（さらにランダムな方向に対しては単位分散をもつ）、これは $\|\mathbf{W}_q\|$ や $\|\mathbf{W}_k\|$ がどれほど大きくなっても変わらない。

```
def logit_stats(Q, K):
    """Std of the scaled logits; mean entropy of each query's weights."""
    S = Q @ K.T / math.sqrt(Q.shape[-1])
    entropy = -(jax.nn.softmax(S) * jax.nn.log_softmax(S)).sum(-1).mean()
    return S.std(), entropy
```

```
d, d_h = 256, 64
X = jax.random.normal(d2l.get_key(), (64, d))         # 64 random tokens
W_q, W_k = jax.random.normal(d2l.get_key(), (2, d, d_h)) / math.sqrt(d)
for lam in (1, 4, 16):
    Q, K = X @ (lam * W_q), X @ (lam * W_k)
    s, h = logit_stats(Q, K)
    s_n, h_n = logit_stats(rms_norm(Q), rms_norm(K))
    print(f'lambda {lam:2d}: logit std {s:5.1f}, entropy {h:.2f} | '
          f'QK-norm: logit std {s_n:.2f}, entropy {h_n:.2f}')
```

```
lambda  1: logit std   1.0, entropy 3.70 | QK-norm: logit std 1.01, entropy 3.68
lambda  4: logit std  15.8, entropy 0.35 | QK-norm: logit std 1.01, entropy 3.68
lambda 16: logit std 252.7, entropy 0.03 | QK-norm: logit std 1.01, entropy 3.68
```

QK-Norm なしで Q と K を $\lambda = 4\times$ および $\lambda = 16\times$ にスケーリングすると、ロジットの標準偏差は二次関数的に跳ね上がり（≈ 16 および ≈ 256）、ソフトマックスのエントロピーはゼロに向かって崩壊する。一方、`rms_norm` を Q と K に適用すれば、ロジットの広がりとアテンションのエントロピーは重みのスケールに対して完全に不変となる。

11.1.3 フィードフォワードネットワーク：GELU から SwiGLU へ

元の Transformer のフィードフォワードネットワークは、チャネル幅を d から $d_{\text{ff}} = 4d$ へと拡大し、要素ごとの活性化関数（ReLU が Vaswani et al. (2017) で、GELU が GPT-2 と BERT で使われた）を適用してから d へと射影し直す 2 層の MLP である。

$$\begin{aligned}\text{FFN}_{\text{GELU}}(\mathbf{x}) &= \text{GELU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2,\\ \mathbf{W}_1 \in \mathbb{R}^{d\times 4d}, &\qquad \mathbf{W}_2 \in \mathbb{R}^{4d\times d},\end{aligned} \tag{11.5}$$

これは $2 \cdot d \cdot 4d = 8d^2$ 個の重みパラメータを用いる。

Dauphin et al. (2017) および Shazeer (2020) に従い、現代の Transformer は Equation 11.5 をバイアスなしの **SwiGLU** ゲート付き線形ユニット（gated linear unit）に置き換える。これは、SiLU で活性化したゲート射影 $\mathbf{x}\mathbf{W}_g$（$\text{SiLU}(z) = z\sigma(z)$）と線形アップ射影 $\mathbf{x}\mathbf{W}_1$ の要素ごとの積をとってから、$\mathbb{R}^d$ へと $\mathbf{W}_2$ で射影し直すものである。

$$\text{FFN}_{\text{SwiGLU}}(\mathbf{x}) = \big(\text{SiLU}(\mathbf{x}\mathbf{W}_g) \odot (\mathbf{x}\mathbf{W}_1)\big)\mathbf{W}_2. \tag{11.6}$$

`SwiGLU` は形状 $d \times d_{\text{ff}}$（$\mathbf{W}_g, \mathbf{W}_1$）と $d_{\text{ff}} \times d$（$\mathbf{W}_2$）の **3** つの行列を使うため、$d_{\text{ff}} = \text{round}(\frac{8}{3}d)$ と設定すれば $3dd_{\text{ff}} \approx 8d^2$ となり、$4d$ の GELU MLP とパラメータ数・FLOP 数の両方が一致する。

```
class FeedForward(nnx.Module):
    """Position-wise FFN: GELU MLP or SwiGLU at matched parameter count."""
    def __init__(self, num_hiddens, act='swiglu', bias=False, *, rngs):
        assert act in ('swiglu', 'gelu'), f'unknown act: {act!r}'
        self.act = act
        if act == 'gelu':
            width = 4 * num_hiddens
        else:  # 'swiglu': three matrices; width 8d/3 matches the MLP budget
            width = round(8 * num_hiddens / 3)
```

```
            self.W_g = d2l.Linear(num_hiddens, width, use_bias=bias,
                                  rngs=rngs)
        self.W_1 = d2l.Linear(num_hiddens, width, use_bias=bias, rngs=rngs)
        self.W_2 = d2l.Linear(width, num_hiddens, use_bias=bias, rngs=rngs)

    def __call__(self, X):
        if self.act == 'gelu':  # The tanh approximation, as in GPT-2
            return self.W_2(jax.nn.gelu(self.W_1(X), approximate=True))
        return self.W_2(jax.nn.silu(self.W_g(X)) * self.W_1(X))
```

11.1.4 設定可能な TransformerBlock

ここで TransformerBlock を組み立てよう。このクラスは、Pre-Norm または Post-Norm の配置（pre_norm）、RMSNorm または LayerNorm（norm）、SwiGLU または GELU の FFN（act）、バイアスの切り替え（bias）、および差し替え可能な attn_factory と ffn_factory のフックをひとまとめにしたものである。これらのフックにより、後の節ではブロックを書き直すことなく、11.2 節の因果的 RoPE アテンションや 11.7 節の混合エキスパート（mixture of experts）層をそのまま組み込める。

```
class TransformerBlock(nnx.Module):
    """Configurable transformer block: attention + FFN on a residual
    stream."""
    def __init__(self, num_hiddens, num_heads, *, dropout=0, norm='rms',
                 act='swiglu', pre_norm=True, bias=False, attn_factory=None,
                 ffn_factory=None, rngs):
        assert norm in ('rms', 'layer'), f'unknown norm: {norm!r}'
        assert num_hiddens % num_heads == 0
        self.pre_norm = pre_norm
        make_norm = nnx.RMSNorm if norm == 'rms' else nnx.LayerNorm
        self.norm1 = make_norm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.norm2 = make_norm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.attention = (d2l.MultiHeadAttention(num_hiddens, num_heads,
                                                 dropout, bias=bias,
                                                 rngs=rngs)
                          if attn_factory is None else attn_factory(rngs))
        self.ffn = (FeedForward(num_hiddens, act, bias=bias, rngs=rngs)
                    if ffn_factory is None else ffn_factory(rngs))
        self.dropout = nnx.Dropout(dropout, rngs=rngs)

    def __call__(self, X, mask=None, **attn_args):
        # attn_args go on to the attention, e.g. the RoPE tables of d2l.GPT
        if self.pre_norm:
            Y = self.norm1(X)
            X = X + self.dropout(self.attention(Y, Y, Y, mask, **attn_args))
            return X + self.dropout(self.ffn(self.norm2(X)))
        X = self.norm1(X + self.dropout(self.attention(X, X, X, mask,
                                                       **attn_args)))
        return self.norm2(X + self.dropout(self.ffn(X)))
```

11.1.4.1 パラメータ数

デフォルトの現代的なブロック（RMSNorm、SwiGLU、bias=False）を 2017 年版・GPT-2 版の変種（GELU、LayerNorm、bias=True）と $d = 256, H = 8$ で比較すると、どの構成も $\approx 12d^2 = 786{,}432$ 個のパラメータ（アテンションに $4d^2$、FFN に $8d^2$）をもつことが確認できる。

```
rngs = nnx.Rngs(d2l.get_key())
blocks = {'default': TransformerBlock(256, 8, rngs=rngs),
          'GELU': TransformerBlock(256, 8, act='gelu', rngs=rngs),
          '2017': TransformerBlock(256, 8, norm='layer', act='gelu',
                                   pre_norm=False, bias=True, rngs=rngs)}
for name, blk in blocks.items():
    attn, ffn, total = (d2l.num_params(m)
                        for m in (blk.attention, blk.ffn, blk))
    print(f'{name:>7}: attention {attn:,}, FFN {ffn:,}, total {total:,} '
          f'= {total / 256 ** 2:.2f} d^2')
```

```
default: attention 262,144, FFN 524,544, total 787,200 = 12.01 d^2
   GELU: attention 262,144, FFN 524,288, total 786,944 = 12.01 d^2
   2017: attention 263,168, FFN 525,568, total 789,760 = 12.05 d^2
```

11.1.5 まとめ

Transformer ブロックは、**トークン混合**（系列の位置間にわたるマルチヘッドアテンション）と**チャネル混合**（d 個の特徴量にわたる位置ごとのフィードフォワードネットワーク）を分離し、両方の更新量を共有された残差ストリームに加算する。**Pre-Norm**（Equation 11.1）は各サブ層への入力のみを正規化して残差ストリームを途切れのない恒等経路として残すのに対し、**Post-Norm**（Equation 11.2）はストリーム自体を正規化する。Xavier スケールでは、深い Post-Norm スタックはトークン間の差異を指数関数的な速さで失い（Equation 11.3）、上層ブロックのクエリ勾配とキー勾配を枯渇させる一方、各ブロックの残差出力射影（W_o と W_2）を $1/\sqrt{2L}$ 倍にスケーリングすれば、Pre-Norm のストリーム分散は $O(1)$ のまま任意の深さ L にわたって保たれる。現代のブロックは、LayerNorm の平均減算と加算バイアスを省いた **RMSNorm**（Equation 11.4）とヘッドごとの **QK-Norm** を組み合わせることで、射影重みの増大にかかわらずコーシー・シュワルツの不等式によってアテンションロジットを有界に保つ。最後に、**SwiGLU**（Equation 11.6）は 2 行列・幅 $4d$ のフィードフォワードネットワークを、幅 $d_{\text{ff}} = \text{round}(\frac{8}{3}d)$ の 3 行列によるゲート付き積に置き換え、標準的な FFN と同じ $8d^2$ のパラメータ数および FLOP 予算と一致させる。

11.1.6 演習問題

1. **RMSNorm のスケール不変性とそのヤコビ行列。** $\mathbf{y} = \text{RMSNorm}(\mathbf{x})$ が Equation 11.4 に従い、$\epsilon = 0$ かつ $\gamma = \mathbf{1}$ で作用するものとする。
   1. $\text{RMSNorm}(c\mathbf{x}) = \text{RMSNorm}(\mathbf{x})$ が任意のスカラー $c > 0$ に対して成り立つことを示せ。
   2. $\mathbf{y} = \sqrt{d}\,\mathbf{x}/\|\mathbf{x}\|_2$ を微分してヤコビ行列が $J(\mathbf{x}) = \frac{\sqrt{d}}{\|\mathbf{x}\|_2}\left(\mathbf{I} - \frac{\mathbf{x}\mathbf{x}^\top}{\|\mathbf{x}\|_2^2}\right)$ となることを示し、$J(\mathbf{x})\mathbf{x} = \mathbf{0}$ を確かめよ。$\mathbf{x}$ に平行な任意の動径方向の摂動が $J(\mathbf{x})$ の零空間に属さなければならないのはなぜか。
2. **展開された残差ストリームと勾配の恒等経路。** $2L$ 個のサブ層（L ブロックの Pre-Norm スタックを構成するサブ層）に $f_1, f_2, \ldots, f_{2L}$ と番号を振る。ここで、奇数番号はアテンシ

ョンサブ層、偶数番号は FFN サブ層とする。

1. Equation 11.1 を展開し、$\mathbf{X}^{(L)}$ を $\mathbf{X}^{(0)} + \sum_{m=1}^{2L} f_m(\mathrm{Norm}(\mathbf{X}^{(m-1)/2}))$ として陽に表せ。
2. 連鎖律を適用して $\partial \mathbf{X}^{(L)} / \partial \mathbf{X}^{(0)}$ を $2L$ 個のヤコビ行列の積として書き下し、主要項を展開したうえで、得られる恒等項 $\mathbf{I}$ を、すべての因子に左から J_{Norm} が掛かる Post-Norm の積と対比せよ。

3. **`TransformerBlock` 内部の QK-Norm**。`attn_factory` を作成し、`d2l.MultiHeadAttention` をラップして、アテンションスコアの計算前にヘッドごとのクエリとキーへ幅 `d // h` の RMSNorm を適用するようにせよ。射影重みを $4\times$（または $16\times$）に定数倍した 16 ブロックのスタック上で、アテンションのエントロピーがゼロから離れた値に保たれることを確かめよ。

4. **深さ方向のストリーム分散予算**。Pre-Norm 配置では、すべてのサブ層が残差ストリームに更新量を加算する。`signal_stats` を修正し、`W_o` と `W_2` を標準偏差 $0.02/\sqrt{2L}$ で、他のすべての行列を標準偏差 0.02 でサンプリングするようにして、すべての行列を標準偏差 0.02 とした場合と $L \in \{8, 16, 32, 64\}$ にわたって比較せよ。スケーリングなしでは最終的なストリームの RMS が L とともに増大する一方、$1/\sqrt{2L}$ 則のもとではほぼ一定に保たれることを確かめよ。

5. **トークン崩壊のレート**。@eq-collapse-rate の予測には FFN に対応する因子が含まれていないが、これはトークン間を混合するのはアテンションサブ層だけであり、FFN は位置ごとに作用するという議論に基づいている。

   1. `MiniBlock` を修正してアテンションサブ層をスキップし（$\mathbf{H} = \mathrm{Norm}(\mathbf{X})$）、Post-Norm の FFN のみが実行されるようにしたうえで、`signal_stats(pre_norm=False)` を再実行せよ（アテンションの重みは使われなくなるため、勾配に関する行は削除せよ）。32 ブロック後にトークンの広がりはどれだけ残っているか。
   2. 次に FFN をスキップし、アテンションサブ層を残せ。実測される広がりは Equation 11.3 に $k = 8, 16, 32$ でどれほど近づくか。
   3. FFN のみのスタックでもわずかな収縮が起きる。位置ごとのネットワークが全トークンに共通する小さなベクトルを加算できてしまうのは、GELU のどの性質によるものか。

11.2 デコーダのみの言語モデル（GPT）

単一の `TransformerBlock`（11.1 節）は、d 次元のトークン表現の系列を同じ形状 (B, T, d) の系列へと写像する。因果的な Transformer ブロックのスタックを自己回帰（autoregressive）型の **Generative Pre-trained Transformer (GPT)** (Radford et al. 2018, 2019; Brown et al. 2020) に仕立てるには、L 個のブロックを 2 つの境界層で挟み込む（Figure 11.3）。

1. **入力埋め込み：**トークン埋め込みテーブル $\mathbf{E} \in \mathbb{R}^{V \times d}$ は、離散トークン ID $x_1, \ldots, x_T \in \{0, \ldots, V-1\}$ を $\mathbb{R}^d$ のベクトルへと写像する。なお、アテンション内部で回転位置埋め込み（rotary position embedding）を使用しない場合は、オプションで学習可能な絶対位置埋め込み（absolute position embedding）$\mathbf{P} \in \mathbb{R}^{T_{\max} \times d}$ を加算する。
2. **最終正規化と言語モデリングヘッド：**L 個の因果的な `TransformerBlock` 層を経た後、最終正規化層が残差ストリーム $\mathbf{H}^{(L)}$ を安定化し、線形ヘッドが各位置 t を次トークンのロジット $\mathbf{Z} = \mathrm{Norm}(\mathbf{H}^{(L)})\mathbf{E}^\top \in \mathbb{R}^{B \times T \times V}$ へと射影する。このとき、出力射影の重みは入力埋め込みテーブル $\mathbf{E}$ と**重み共有**（weight tying）する (Press and Wolf 2017; Inan et al. 2017)。

11.2.1 ブロックから言語モデルへ

各ブロックの内部では、CausalAttention がクエリ・キー・値の射影を融合し、**グループ化クエリアテンション**（grouped-query attention、num_kv_heads ≤ num_heads、10.3 節）をサポートする。さらに、オプションで **RoPE**（rope=True、10.4 節）によってクエリとキーを回転させ、位置 t が位置 $s \le t$ のみにアテンションを向けるよう因果マスクを適用する。

```
class CausalAttention(nnx.Module):
    """Causal self-attention, optionally rotary; num_kv_heads < num_heads
    shares each key-value head across a group of queries."""
    def __init__(self, num_hiddens, num_heads, *, bias=False, rope=False,
                 num_kv_heads=None, rngs):
        self.num_heads, self.rope = num_heads, rope
        self.num_kv_heads = num_kv_heads or num_heads
        self.kv_hiddens = self.num_kv_heads * (num_hiddens // num_heads)
        self.W_qkv = d2l.Linear(num_hiddens,
                                num_hiddens + 2 * self.kv_hiddens,
                                use_bias=bias, rngs=rngs)
        self.W_o = d2l.Linear(num_hiddens, num_hiddens, use_bias=bias,
                              rngs=rngs)

    def qkv(self, X, offset=0, tables=None):
        """Queries, keys, and values of X, each (batch, heads, length, head
        width), at positions offset, offset + 1, ...; tables, if given, are
        the d2l.rope_tables of these positions."""
        B, T, D = X.shape
        q, k, v = jnp.split(self.W_qkv(X), [D, D + self.kv_hiddens], axis=-1)
        q = q.reshape(B, T, self.num_heads, -1).swapaxes(1, 2)
        k, v = (u.reshape(B, T, self.num_kv_heads, -1).swapaxes(1, 2)
                for u in (k, v))
        if self.rope:
            q, k = (d2l.rope(u, offset, tables=tables) for u in (q, k))
        return q, k, v

    @staticmethod
    @jax.checkpoint  # The backward pass recomputes the T x T weights
    def attend(q, k, v):
        """Causal attention of (batch, length, heads, head width) queries,
        keys, and values."""
        return jax.nn.dot_product_attention(q, k, v, is_causal=True)

    def __call__(self, X, keys=None, values=None, mask=None, tables=None):
        # Self-attention: X also serves as the keys and the values, and the
        # mask is causal, so a block's mask argument must stay None
        if mask is not None:
            raise ValueError('CausalAttention applies its own mask')
        q, k, v = self.qkv(X, tables=tables)
        Y = self.attend(*(u.swapaxes(1, 2) for u in (q, k, v)))
        return self.W_o(Y.reshape(*X.shape[:2], -1))  # (batch, length, d)
```

ここで定義する GPT クラスは、設定可能な num_blks 個の TransformerBlock モジュール（デフォルトでは norm='rms'、act='swiglu'、pos='rope'、pre_norm=True であり、GPT-2 の norm='layer'、act='gelu'、pos='learned'、bias=True もサポートする）を積み重ね、出力射影をトークン埋め込

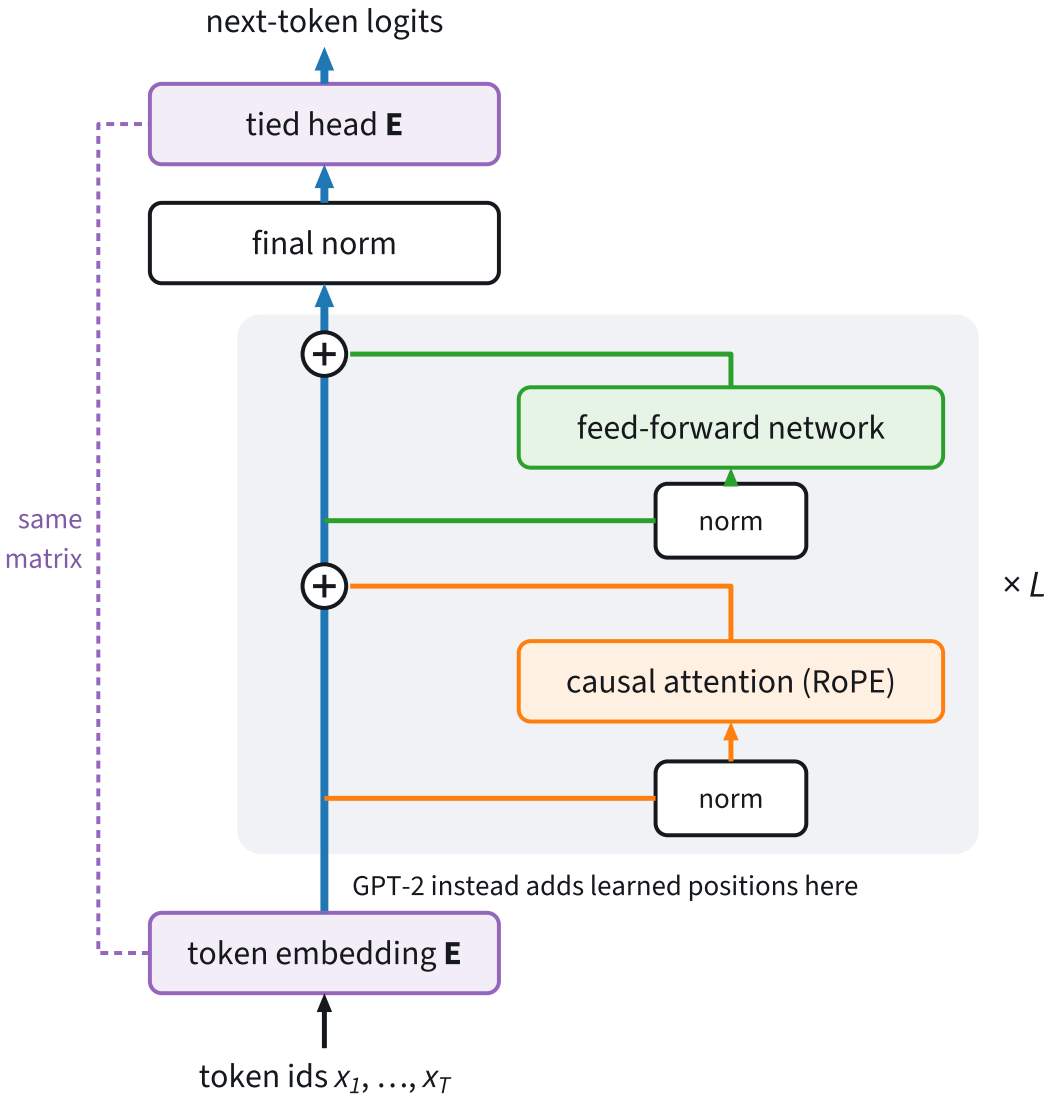


Figure 11.3: 下から上へと見た GPT の構造。テーブル $\mathbf{E}$ がトークン ID を残差ストリームに埋め込み、L 個の因果的な `TransformerBlock` 層がストリームから読み出して書き込み、最終正規化と転置テーブル $\mathbf{E}^\top$ が各位置のベクトルを語彙上のロジットへと変換する。ここに示す現代的な構成では各アテンション層の内部で RoPE によりクエリとキーを回転させるが、GPT-2 では代わりに最下部で学習された位置ベクトルを一度だけ加算する。

みテーブル token_emb と重み共有する。

```
class GPT(nnx.Module):
    """Decoder-only transformer language model built from configurable
    blocks."""
    CausalAttention = CausalAttention  # A subclass may replace it

    def __init__(self, vocab_size, *, num_hiddens=256, num_heads=8,
                 num_blks=6, max_len=1024, pos='rope', norm='rms',
                 act='swiglu', pre_norm=True, bias=False, dropout=0,
                 num_kv_heads=None, ffn_factory=None, rngs):
        assert pos in ('rope', 'learned'), f'unknown pos: {pos!r}'
        self.pos, self.max_len, self.num_hiddens = pos, max_len, num_hiddens
        self.head_width = num_hiddens // num_heads
        init = nnx.initializers.normal(0.02)
        self.token_emb = d2l.Embed(vocab_size, num_hiddens,
                                   embedding_init=init, rngs=rngs)
        if pos == 'learned':
            self.pos_emb = nnx.Embed(max_len, num_hiddens,
                                     embedding_init=init, rngs=rngs)
        attn = lambda rngs: self.CausalAttention(
            num_hiddens, num_heads, bias=bias, rope=pos == 'rope',
            num_kv_heads=num_kv_heads, rngs=rngs)
        self.blks = nnx.List([
            d2l.TransformerBlock(num_hiddens, num_heads, dropout=dropout,
                                 norm=norm, act=act, pre_norm=pre_norm,
                                 bias=bias, attn_factory=attn,
                                 ffn_factory=ffn_factory, rngs=rngs)
            for _ in range(num_blks)])
        self.norm = (nnx.RMSNorm if norm == 'rms'
                     else nnx.LayerNorm)(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.init_weights(rngs)

    def __call__(self, X):                    # X: (batch, T) token ids
        H, tables = self.token_emb(X), None   # (batch, T, d)
        if self.pos == 'learned':
            H = H + self.pos_emb(jnp.arange(X.shape[1]))
        else:  # RoPE angles of positions 0, ..., T - 1, for every block
            tables = d2l.rope_tables(X.shape[1], self.head_width)
        for blk in self.blks:                 # L blocks, causal attention
            H = blk(H, tables=tables)
        return self.token_emb.attend(self.norm(H))  # (batch, T, V), tied
```

GPT-2 (Radford et al. 2019) および 11.1.2.2 節に従い、init_weights は線形層と埋め込み層の重みを $\mathcal{N}(0, 0.02^2)$ からサンプリングしつつ、各ブロックの 2 つの残差出力射影（attention.W_o と ffn.W_2）を $1/\sqrt{2L}$ 倍にスケーリングする。

```
@d2l.add_to_class(GPT)
def init_weights(self, rngs):
    """GPT-2's initialization of the linear layers: N(0, 0.02^2) weights and
    zero biases; W_o and W_2, which write to the stream, get std
    0.02 / sqrt(2L)."""
```

```
    L = len(self.blks)
    for path, m in nnx.iter_graph(self):
        if isinstance(m, nnx.Linear):
            std = 0.02 / (2 * L) ** 0.5 if path[-1] in ('W_o', 'W_2') else 0.02
            m.kernel[...] = std * jax.random.normal(rngs.params(),
                                                    m.kernel.shape)
            if m.bias is not None:
                m.bias[...] = jnp.zeros(m.bias.shape)
```

11.2.2 WikiText-103 におけるバイトレベル GPT の訓練

ゼロからの実験を自己完結させ、トークナイザに起因するアーティファクトを排除するため、**WikiText-103** (Merity et al. 2016) の生の UTF-8 バイト列（$V = 256$）上で直接訓練する。ここでは、ウィンドウ数 $B = 64$、長さ $T = 128$ バイトのミニバッチ（ステップあたり 8,192 トークン）を用いる。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
print(f'{len(data.train) / 1e6:.0f}M training bytes, '
      f'{len(data.val) / 1e6:.2f}M validation bytes, '
      f'vocabulary of {len(data.vocab)}')
```

```
270M training bytes, 1.14M validation bytes, vocabulary of 256
```

また、補助コンストラクタ `gpt(...)` も定義しておく。これは新しく初期化された、すぐに訓練可能な `GPT` を構築する関数である。

`gpt` はモデルごとに `d2l.get_key()` から新しいランダムな重みをサンプリングする。

```
def gpt(*args, **kwargs):  # Fresh random weights for every model
    return GPT(*args, **kwargs, rngs=nnx.Rngs(d2l.get_key()))
```

11.2.2.1 自己回帰の目的関数

入力ウィンドウ $\mathbf{x} = (x_1, \dots, x_T)$ とシフトされたターゲットウィンドウ $\mathbf{y} = (x_2, \dots, x_{T+1})$ が与えられたとき、因果マスクにより、出力ロジットの第 t 行 $\mathbf{z}_t \in \mathbb{R}^V$ は $(x_1, \dots, x_t)$ のみに依存することが保証される。訓練では、全 T 個の位置にわたる平均交差エントロピー損失を並列に最小化する。

$$\mathcal{L}(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \log \operatorname{softmax}(\mathbf{z}_t)_{x_{t+1}}. \tag{11.7}$$

初期化の段階（学習が一切進む前）では、モデルはすべてのバイトにほぼ一様な確率 $1/V$ を割り当てるため、初期の交差エントロピー損失は $\ln V = \ln 256 \approx 5.55$ nat/バイトに近い値となる。まず、シフトされた入力・ターゲットのウィンドウ対 (`X[0]`, `Y[0]`) を 1 つと、5×5 の下三角因果マスクを確認してみよう。

```
start = data.train.tobytes().find(b'the European lobster')
X, Y = data.windows(data.train, [start])  # One window and its targets
```

```
print('X[0]:', repr(data.vocab.decode(X[0, :24].tolist())))
print('Y[0]:', repr(data.vocab.decode(Y[0, :24].tolist())))
print(np.asarray(d2l.causal_mask(5)).astype(int))
```

```
X[0]: 'the European lobster ) a'
Y[0]: 'he European lobster ) an'
[[1 0 0 0 0]
 [1 1 0 0 0]
 [1 1 1 0 0]
 [1 1 1 1 0]
 [1 1 1 1 1]]
```

新しく初期化した 6 ブロックの GPT（≈ 4.79M パラメータ）を検証ミニバッチ上で評価すると、未訓練時の交差エントロピーが $\ln 256 \approx 5.55$ nat/バイトに近いことが確認できる。

```
model = gpt(len(data.vocab), max_len=128)
loss = math.log(d2l.lm_perplexity(model, [next(data.val_dataloader())]))
print(f'{d2l.num_params(model) / 1e6:.2f}M parameters; loss before '
      f'training {loss:.2f} nats per byte, log 256 = {math.log(256):.2f}')
```

```
4.79M parameters; loss before training 5.51 nats per byte, log 256 = 5.55
```

11.2.2.2 訓練レシピ（LMOptimizer）

現代の LLM の事前学習では、**AdamW**（9.5 節、標準的なモメンタムパラメータとしてデフォルトでは $\beta_1 = 0.9, \beta_2 = 0.999$、大バッチでの実行では $\beta_2 = 0.95$ を使用）に、**選択的重み減衰**（埋め込み、バイアス、1D 正規化スケールを対象外とし、2D 線形射影行列にのみ重み減衰 $\lambda = 0.1$ を適用）、**線形ウォームアップとそれに続くコサイン学習率減衰**（9.7 節）、および**大域的勾配ノルムクリッピング**（$\|\nabla \mathcal{L}\|_2 \leq 1.0$）を組み合わせる。

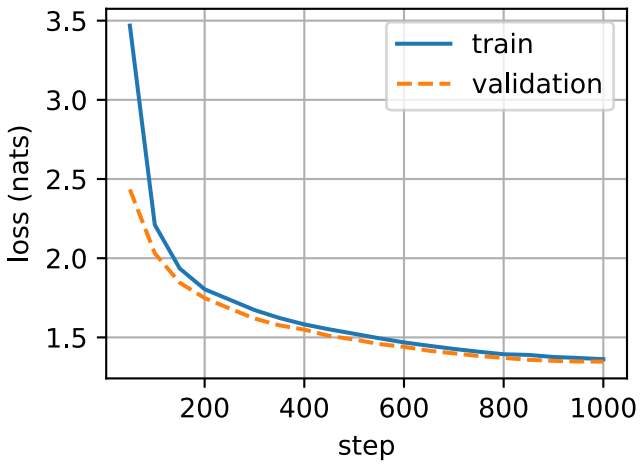
```
def lm_decay_mask(params):
    """Decay the weight matrices; exempt embeddings, gains, and biases."""
    # Embedding tables are named *emb* (token_emb, pos_emb) by convention
    return jax.tree_util.tree_map_with_path(
        lambda path, p: p.ndim == 2 and 'emb' not in str(path), params)

@optax.inject_hyperparams
def lm_adamw(lr=0.0, num_steps=1, warmup=0, weight_decay=0.0, max_norm=1.0):
    """The transformation of LMOptimizer, with its hyperparameters as state."""
    schedule = lambda t: lr * d2l.cosine_warmup(t, num_steps, warmup)
    return optax.chain(
        optax.clip_by_global_norm(max_norm),
        optax.adamw(schedule, weight_decay=weight_decay, mask=lm_decay_mask))

class LMOptimizer(nnx.Optimizer):
    """AdamW with weight decay on the weight matrices only, a linear warmup
    and a cosine decay of the learning rate, and gradient-norm clipping."""
    tx = lm_adamw()  # Shared by all instances: the step compiles once
    def __init__(self, model, lr, num_steps, warmup, weight_decay=0.1,
                 max_norm=1.0):
        super().__init__(model, LMOptimizer.tx, wrt=nnx.Param)
```

```
        hyperparams = dict(lr=lr, num_steps=num_steps, warmup=warmup,
                           weight_decay=weight_decay, max_norm=max_norm)
        for name, value in hyperparams.items():
            self.opt_state.hyperparams[name][...] = value
```

6 ブロックのバイトレベル GPT（$d = 256, H = 8, L = 6$、≈ 4.8M パラメータ）を 1,000 ステップ（≈ 8.2M バイト）訓練すると、検証交差エントロピーは英語 Wikipedia テキストの経験ユニグラムエントロピー（3.17 nat/バイト）と経験バイグラムエントロピー（2.45 nat/バイト）の両方を大きく下回る。

```
model = gpt(len(data.vocab), max_len=128)
num_steps, every = 1000, 50
optimizer = LMOptimizer(model, lr=2e-3, num_steps=num_steps, warmup=100)
steps, train_curve, val_curve, timer = [], [], [], d2l.Timer()
for step in range(every, num_steps + 1, every):
    timer.start()
    losses = d2l.train_steps(model, data, optimizer, every)
    timer.stop()  # Training time only, without the validation passes
    steps.append(step)
    train_curve.append(sum(losses) / every)
    val_curve.append(math.log(d2l.lm_perplexity(model, data.val_dataloader())))
d2l.plot(steps, [train_curve, val_curve], 'step', 'loss (nats)',
         legend=['train', 'validation'], fmts=('-', '--'))
print(f'step {num_steps}: train {train_curve[-1]:.2f}, '
      f'validation {val_curve[-1]:.2f} nats per byte; '
      f'{timer.sum():.0f} s of training')
```

```
step 1000: train 1.36, validation 1.35 nats per byte; 29 s of training
```



参考として、訓練コーパスの先頭 10M バイトから経験ユニグラムおよびバイグラムのバイトエントロピーを直接計算し、6 ブロックの Transformer が局所的な文字統計からどれほど先へ進んでいるかを確かめる。

```
text = data.train[:10**7].astype(np.int64)  # The first 10M training bytes
V = len(data.vocab)
unigram = np.bincount(text, minlength=V) / len(text)
pairs = np.bincount(text[:-1] * V + text[1:], minlength=V * V).reshape(V, V)
bigram = pairs / np.maximum(pairs.sum(1, keepdims=True), 1)
p = unigram[unigram > 0]
print(f'unigram {-(p * np.log(p)).sum():.2f}, '
      f'bigram {-np.log(bigram[text[:-1], text[1:]]).mean():.2f} '
      f'nats per byte')
```

```
unigram 3.17, bigram 2.45 nats per byte
```

11.2.3 統制された比較

11.2.3.1 実行間のばらつきと FFN 活性化関数（SwiGLU と GELU）

純粋なアーキテクチャ上の改善を乱数シードのノイズと切り分けるため、デフォルトの SwiGLU モデルを独立な 3 つのシードで訓練し、さらにパラメータ数を揃えた GELU モデル（`act='gelu'`）も 3 つのシードで訓練する。

```
def train_lm(model, data, lr=2e-3, num_steps=1000, warmup=100):
    """Train with the recipe of the main run; return milliseconds per step."""
    optimizer = LMOptimizer(model, lr=lr, num_steps=num_steps, warmup=warmup)
    # One untimed call of 50 steps warms up (JAX compiles), the rest is timed
    bench = d2l.Benchmark(lambda: d2l.train_steps(model, data, optimizer, 50),
                          warmup=1, repeats=num_steps // 50 - 1)
    return 1000 * bench.time / 50

runs = {'swiglu': [val_curve[-1]], 'gelu': []}  # The main run comes first
times = {'swiglu': [], 'gelu': []}  # Milliseconds per step
for act, repeats in (('swiglu', 2), ('gelu', 3)):
    for _ in range(repeats):
        net = gpt(len(data.vocab), act=act, max_len=128)
        times[act].append(train_lm(net, data))
        runs[act].append(math.log(d2l.lm_perplexity(net,
                                                    data.val_dataloader())))
    print(f'{act:>6} ({d2l.num_params(net):,} parameters): validation '
          + ', '.join(f'{v:.3f}' for v in runs[act])
          + f'; mean {np.mean(runs[act]):.3f}, range {np.ptp(runs[act]):.3f}')
```

```
swiglu (4,788,992 parameters): validation 1.347, 1.342, 1.348; mean 1.346, range 0.006
  gelu (4,787,456 parameters): validation 1.404, 1.398, 1.395; mean 1.399, range 0.009
```

乱数シード間の実行ごとのばらつきはわずか ~ 0.01 nat/バイトにすぎないのに対し、SwiGLU は同じパラメータ数および FLOP 予算のもとで GELU を一貫して ~ 0.06 nat/バイト上回る。

この `train_lm` を使えば、11.1 節におけるもう 1 つの設計上の選択である正規化の配置も検証できる。初期化の時点では、11.1.2.2 節で見たように、`init_weights` が用いる重みスケール 0.02 において Post-Norm に問題は生じなかった。トークン同士は離れたままであり、クエリの勾配も消失せずに残る。しかし、Post-Norm が同様にうまく**訓練できる**かどうかは別の問題である。そこで、同じモデルを Post-Norm ブロック（`pre_norm=False`）にして、ウォームアップを 500 ステップにした場合、本実験と同じ 100 ステップにした場合、およびウォームアップなしにした場合の 3 通りで訓練する。ピーク学習率はいずれの 3 通りでも本実験と同じである。

```
for warmup in (500, 100, 0):  # Post-norm: long, pre-norm's, and no warmup
    net = gpt(len(data.vocab), pre_norm=False, max_len=128)
    train_lm(net, data, warmup=warmup)
    loss = math.log(d2l.lm_perplexity(net, data.val_dataloader()))
    print(f'post-norm, warmup {warmup:>3}: validation {loss:.3f}')
```

```
post-norm, warmup 500: validation 1.375
post-norm, warmup 100: validation 3.174
post-norm, warmup   0: validation 3.174
```

その答えはウォームアップに依存する。Pre-Norm モデルを約 1.34 nat/バイトまで訓練できる 100 ステップのウォームアップでは、Post-Norm モデルは 3.17 で止まり、ウォームアップなしの実行でも同じ結果になる。この数値はすでに上で見たものである。すなわち先ほど計算したユニグラムエントロピーであり、これら 2 つのモデルは各バイトがどれくらいの頻度で出現するかだけを学習し、直前にあるバイト列については何も学習していない。原因は初期の更新が大きすぎることにある。学習率がわずか数ステップのうちにピークへ達すると、アテンションと FFN の各サブ層の出力が、加算先の正規化された残差ストリームよりもはるかに大きく成長してしまう。その結果、正規化を経た後にはすべてのトークンがほぼ同一の最終ベクトルに行き着き、どの位置でも同じベクトルしか得られないため、バイトの出現頻度しか予測できなくなる。トークン同士を再び引き離すはずの勾配もトークンの広がりとともに縮小してしまうため（11.1.2.2 節）、訓練が回復することはない。

ウォームアップを 500 ステップにすれば、サブ層の出力はストリームより小さいまま保たれ、Post-Norm モデルも Pre-Norm モデルにわずかに及ばないものの約 1.38 nat/バイトまで訓練できる。したがって、Pre-Norm には十分なウォームアップであっても Post-Norm には不十分なのである。その境界がどこにあるかは一概には言えない。境界は明確ではなく、中間のウォームアップ長では訓練が進む実行もあれば崩壊する実行もある。これは Xiong et al. (2020) が元の Transformer について報告した挙動そのものである。すなわち、Post-Norm は長いウォームアップを必要とするのに対し、Pre-Norm はウォームアップなしでも訓練でき、より速く収束する。

11.2.3.2 条件を揃えたリカレントベースラインとの比較

同じパラメータ数（$d_{\text{rnn}} = 1920$、≈ 4.7M パラメータ）と訓練ステップ数を与えたとき、6 ブロックの GPT はリカレント言語モデル（`d2l.RNNLM`、8.4.2 節）と比べてどうなるだろうか。RNN の学習率を $\{10^{-3}, 2\times10^{-3}, 4\times10^{-3}\}$ にわたってスイープし、ウィンドウ内の位置 $t \in \{0, \dots, 127\}$ の関数として検証損失を比較する（`loss_by_position`）。

```
def rnnlm(vocab_size, num_hiddens):  # The tanh RNN LM of the Basics
    rngs = nnx.Rngs(d2l.get_key())
    return d2l.RNNLM(d2l.RNN(256, num_hiddens, rngs=rngs), vocab_size,
                     rngs=rngs)

@nnx.jit
def position_loss_sum(model, X, Y):
    """Loss in nats at each position, summed over the batch."""
    return optax.softmax_cross_entropy_with_integer_labels(model(X), Y).sum(0)

def loss_by_position(model, data):
    """Mean validation loss in nats at each position of the window."""
    total, count = 0, 0
    for X, Y in data.val_dataloader():
```

```
        total += position_loss_sum(model, jnp.asarray(X), jnp.asarray(Y))
        count += len(X)
    return (total / count).tolist()
```

各候補のピーク学習率 $\{10^{-3}, 2\times10^{-3}, 4\times10^{-3}\}$ ごとに RNNLM を 2 つのシードで訓練し、最良のリカレントモデルの実行結果をウィンドウ位置 $t \in \{0, 1, 4, 16, 64, 127\}$ にわたって GPT と比較する。

```
rnn_runs = []  # (validation loss, model, ms per step) of every run
for lr in (1e-3, 2e-3, 4e-3):
    for _ in range(2):
        rnn = rnnlm(len(data.vocab), 1920)
        ms = train_lm(rnn, data, lr=lr)
        rnn_runs.append((np.mean(loss_by_position(rnn, data)), rnn, ms))
    print(f'RNN ({d2l.num_params(rnn):,} parameters), peak rate {lr:g}: '
          'validation ' + ', '.join(f'{r[0]:.3f}' for r in rnn_runs[-2:]))
best = min(rnn_runs, key=lambda run: run[0])
positions = (0, 1, 4, 16, 64, 127)
print('validation loss at positions', ', '.join(map(str, positions)))
for name, net, ms in (('GPT', model, np.mean(times['swiglu'])),
                      ('best RNN', *best[1:])):
    loss = loss_by_position(net, data)
    print(f'{name:>8}: ' + ' '.join(f'{loss[t]:.2f}' for t in positions)
          + f'; {ms:.1f} ms per training step')
```

```
RNN (4,737,152 parameters), peak rate 0.001: validation 1.454, 1.463
RNN (4,737,152 parameters), peak rate 0.002: validation 1.444, 1.482
RNN (4,737,152 parameters), peak rate 0.004: validation 1.715, 1.608
validation loss at positions 0, 1, 4, 16, 64, 127
     GPT: 2.47 2.16 1.49 1.34 1.33 1.32; 13.0 ms per training step
best RNN: 2.50 2.15 1.54 1.44 1.44 1.42; 10.9 ms per training step
```

位置 $t = 0$（文脈として 1 バイトしか利用できない位置）では、GPT と RNNLM はほぼ同じ損失から出発する。しかし、利用可能なプレフィックスの文脈が増えるにつれて（$t \to 127$）、Transformer の検証損失は RNN よりも急激に低下する。これは、自己アテンションが任意の先行バイトを $O(1)$ ホップで直接読み取り、過去の履歴を t 回の逐次的なリカレントステップを通して圧縮する必要がないからである。

11.2.4 モデルからのサンプリング

自己回帰生成（GPT.generate）では、一度に 1 トークンずつ末尾に追加していく。各ステップで現在のプレフィックスをモデルに入力し、最終位置のロジット $\mathbf{z}_t$ を**温度**（temperature）$\tau > 0$ で割り、オプションで上位の **top-k** 個を除くすべてのロジットを $-\infty$ にマスクしたうえで (Fan et al. 2018)、$\mathrm{softmax}(\mathbf{z}_t/\tau)$ から次のトークンをサンプリングする。なお、極限 $\tau \to 0$ に対応する貪欲デコーディング（greedy decoding）は、top_k=1 の場合にあたる。

```
@nnx.jit(static_argnames='top_k')
def _gpt_sample_next(model, buf, t, key, temperature, top_k):
    """Sample the token after the first t tokens of buf and append it.
```

```
    A full buffer (t equal to its length) drops its oldest token first."""
    logits = model(buf)[0, t - 1] / temperature
    if top_k is not None:
        cutoff = jax.lax.top_k(logits, top_k)[0][-1]
        logits = jnp.where(logits < cutoff, -jnp.inf, logits)
    token = jax.random.categorical(key, logits)
    n = buf.shape[1]
    buf = jnp.where(t == n, jnp.roll(buf, -1, axis=1), buf)
    return buf.at[0, jnp.minimum(t, n - 1)].set(token), token

@d2l.add_to_class(GPT)
def generate(self, prefix, num_tokens, temperature=1.0, top_k=None):
    """Sample a continuation of the token-id list prefix."""
    n, ctx = self.max_len, list(prefix)[-self.max_len:]
    # One buffer of fixed shape (1, max_len): the step compiles once
    buf, t = jnp.asarray([ctx + [0] * (n - len(ctx))]), len(ctx)
    tokens = []
    for key in jax.random.split(d2l.get_key(), num_tokens):
        buf, token = _gpt_sample_next(self, buf, t, key, temperature, top_k)
        tokens.append(token)
        t = min(t + 1, n)
    return list(prefix) + [int(i) for i in jax.device_get(tokens)]
```

'The history of the ' に続く 120 バイトの系列を 3 通りの (temperature, top_k) 設定でサンプリングすると、低い温度と top-k フィルタリングがいかに多様性と引き換えに局所的な一貫性を高めるかがわかる。

```
prefix = data.vocab.encode('The history of the ')
for temperature, top_k in ((1.0, None), (0.7, 8), (2.0, None)):
    out = model.generate(prefix, 120, temperature=temperature, top_k=top_k)
    d2l.print_wrapped(f'T={temperature}, top_k={top_k}: '
                      + repr(data.vocab.decode(out)))
```

```
T=1.0, top_k=None: 'The history of the 17th August Hackrulka – Africa in the
    WABI . " After played a " water had listing in view to a reverse task ,
    in the S'
T=0.7, top_k=8: 'The history of the side , when the former service of more
    society was really for the moved . The second became the ring of the
    central acti'
T=2.0, top_k=None: "The history of the tuh□ux Telecovcuvlly YotaNevofry
    Gate\x154□□|□f Dudtlvbur\x1aet□□2□□ ' asprarx □ieques ou82CR Fl□ Yez□□□□
    hurri□Nfi wr□gmh si□"
```

11.2.5 事前学習済み GPT-2（124M）の読み込み

本節の GPT クラスは norm='layer'、act='gelu'、pos='learned'、および bias=True をサポートしているため、OpenAI の 124M パラメータの **GPT-2 Small** アーキテクチャ（$V = 50{,}257, d = 768, H = 12, L = 12, T_{\max} = 1024$）をそのままインスタンス化し、公式の model.safetensors の重みとバイトレベル BPE トークナイザ（gpt2_tokenizer）を直接読み込める。

```
GPT2_URL = 'https://huggingface.co/openai-community/gpt2/resolve/main/'
d2l.DATA_HUB['gpt2-weights'] = (
    GPT2_URL + 'model.safetensors',
    '89a76996d7c6ee89b86618a265483aab73e61d50')
d2l.DATA_HUB['gpt2-merges'] = (
    GPT2_URL + 'merges.txt', '396d4d8ec90cb02f4d56e049e0e4add868bcd943')
d2l.DATA_HUB['gpt2-encoder'] = (
    GPT2_URL + 'vocab.json', 'f0223209235343bc067d7da838328bced8085ae1')

def gpt2_tokenizer():
    """GPT-2's byte-level BPE tokenizer, assembled by tiktoken."""
    import tiktoken  # lazy: d2l itself does not require tiktoken
    from tiktoken.load import data_gym_to_mergeable_bpe_ranks
    return tiktoken.Encoding(
        'gpt2', explicit_n_vocab=50257,
        pat_str=d2l.BPETokenizer.GPT2_PATTERN,
        mergeable_ranks=data_gym_to_mergeable_bpe_ranks(
            d2l.download('gpt2-merges', '../data/gpt2'),
            d2l.download('gpt2-encoder', '../data/gpt2')),
        special_tokens={'<|endoftext|>': 50256})

enc = gpt2_tokenizer()
ids = enc.encode('Attention is all you need.')
print(ids, [enc.decode([i]) for i in ids])
```

```
[8086, 1463, 318, 477, 345, 761, 13] ['Att', 'ention', ' is', ' all', ' you', ' need', '.']
```

Table 11.1: OpenAI の GPT-2 safetensors チェックポイントのキーと本節の GPT モジュールの属性との対応（Conv1D の重みは形状 (in, out) をもつ）。

チェックポイントのテンソル	GPT の属性	備考
wte.weight	token_emb	出力ヘッドと共有（V x d）
wpe.weight	pos_emb	1,024 個の学習済み位置ベクトル
h.i.ln_1, h.i.ln_2	blks[i].norm1, blks[i].norm2	アテンション前および FFN 前の LayerNorm
h.i.attn.c_attn	blks[i].attention.W_qkv	融合 QKV（Conv1D）
h.i.attn.c_proj	blks[i].attention.W_o	出力射影（Conv1D）
h.i.mlp.c_fc, h.i.mlp.c_proj	blks[i].ffn.W_1, blks[i].ffn.W_2	2 層 GELU MLP（Conv1D）
ln_f	norm	出力ヘッド前の最終 LayerNorm

nnx.Linear のカーネルは (in, out) すなわち Conv1D と同じ配置であるため、load_gpt2 は重みをそのままコピーする。

```
def load_gpt2():
    """GPT-2 small (124M) as a GPT with the released weights, in eval
    mode."""
    from safetensors.numpy import load_file  # lazy, like tiktoken above
    gpt2 = GPT(vocab_size=50257, num_hiddens=768, num_heads=12, num_blks=12,
               max_len=1024, pos='learned', norm='layer', act='gelu',
               pre_norm=True, bias=True, rngs=nnx.Rngs(d2l.get_key()))
```

```
    weights = {k: jnp.asarray(v) for k, v in load_file(
        d2l.download('gpt2-weights', '../data/gpt2')).items()}
    gpt2.token_emb.embedding[...] = weights['wte.weight']
    gpt2.pos_emb.embedding[...] = weights['wpe.weight']
    gpt2.norm.scale[...] = weights['ln_f.weight']
    gpt2.norm.bias[...] = weights['ln_f.bias']
    for i, blk in enumerate(gpt2.blks):
        for norm, key in ((blk.norm1, f'h.{i}.ln_1'),
                          (blk.norm2, f'h.{i}.ln_2')):
            norm.scale[...] = weights[key + '.weight']
            norm.bias[...] = weights[key + '.bias']
        for linear, key in ((blk.attention.W_qkv, f'h.{i}.attn.c_attn'),
                            (blk.attention.W_o, f'h.{i}.attn.c_proj'),
                            (blk.ffn.W_1, f'h.{i}.mlp.c_fc'),
                            (blk.ffn.W_2, f'h.{i}.mlp.c_proj')):
            # Conv1D stores (in, out), exactly nnx.Linear's kernel layout
            linear.kernel[...] = weights[key + '.weight']
            linear.bias[...] = weights[key + '.bias']
    gpt2.eval()
    return gpt2

gpt2 = load_gpt2()
print(f'{d2l.num_params(gpt2) / 1e6:.1f}M parameters')
```

```
124.4M parameters
```

読み込んだチェックポイントを検証するため、『The Time Machine』の冒頭段落における交差エントロピー（≈ 3.9 nat/トークン、すなわちパープレキシティ ≈ 49）を計算し、さらに続きのテキストを生成する。

```
text = ("The Time Machine, by H. G. Wells. The Time Traveller was "
        "expounding a recondite matter to us.")
x = jnp.asarray(enc.encode(text))[None]
loss = optax.softmax_cross_entropy_with_integer_labels(
    gpt2(x)[0, :-1], x[0, 1:]).mean()
print(f'per-token loss {loss:.2f}, perplexity {jnp.exp(loss):.0f}')
```

```
per-token loss 3.89, perplexity 49
```

事前学習済みの重みが検証できたところで、`gpt2.generate` にプロンプトを与え、貪欲デコーディング（`top_k=1`）によって決定論的な続きを生成してみよう。

```
out = gpt2.generate(enc.encode('Alan Turing theorized that computers '
                               'would one day become'), 16, top_k=1)
d2l.print_wrapped(enc.decode(out))
```

```
Alan Turing theorized that computers would one day become the most powerful
    machines on the planet.

The computer is a computer that
```

確率的な top-k サンプリング（`temperature=0.8, top_k=50`）に切り替えて 40 個のサブワードトークンを生成すると、流暢で多様な続きが得られる。

```
out = gpt2.generate(enc.encode('The secret of a good deep learning '
                               'textbook is'), 40, temperature=0.8,
                    top_k=50)
d2l.print_wrapped(enc.decode(out))
```

```
The secret of a good deep learning textbook is the fact that you can combine
    a great set of experiments with your own knowledge to create a good,
    consistent textbook that won't only be better but also be easier to read
    and understand.

This
```

11.2.6 現代的なレシピ

2019 年の `load_gpt2()` の構成を本節のデフォルトの `GPT` と比較すると、オープンウェイトである現代のデコーダのみの言語モデルを特徴づける 4 つの主要な改良点が浮かび上がる（Table 11.2）。

1. **正規化：**バイアスなしの Pre-Norm `RMSNorm`（11.1.2.4 節）を `LayerNorm` の代わりに用い、多くの場合ヘッドごとの **QK-Norm**（11.1.2.5 節）やサブ層出力の正規化（OLMo 2、Gemma 3）と組み合わせる。
2. **位置表現：**各アテンション層の内部で**回転位置埋め込み**（RoPE、10.4 節）を、固定長の学習済み位置テーブル `wpe` の代わりに用いる。
3. **フィードフォワードネットワーク：**バイアスなしのゲート付き **`SwiGLU`**（または `GeGLU`）を、2 層の `GELU` MLP の代わりに用いる。
4. **推論キャッシュとモデル容量のスケーリング：**KV キャッシュを縮小する（11.3 節）ための**グループ化クエリアテンション（GQA）**、**スライディングウィンドウアテンション**、または**低ランク潜在圧縮**（low-rank latent compression、MLA）と、トークンあたりの FLOP 数を増やさずに総パラメータ数を拡大する**混合エキスパート**（mixture of experts、MoE）層（11.7 節）。

Table 11.2: 2023〜2025 年の技術報告書で報告された 7 つのオープンウェイトモデル族のアーキテクチャ。アテンションとキャッシュ：11.3.3 節。正規化：11.1 節。位置表現：10.4 節。混合エキスパート：11.7 節。

モデル	アテンションとキャッシュ	正規化	位置表現	FFN	密または MoE
Mistral 7B（2023）(Jiang et al. 2023)	GQA 32:8、スライディングウィンドウ 4096	RMSNorm、Pre-Norm	RoPE、ベース 10^4	SwiGLU	密
Llama 3（2024）	GQA 32:8（8B）、64:8（70B）、128:8（405B）	RMSNorm、Pre-Norm	RoPE、ベース 5×10^5	SwiGLU	密
Qwen3（2025）(A. Yang et al. 2025)	GQA：64:8 密（32B）、64:4 MoE（235B-A22B）	RMSNorm、Pre-Norm、QK-Norm	RoPE、ベース 10^6、YaRN	SwiGLU	密、および MoE：128 エキスパート中 8 個活性

モデル	アテンションとキャッシュ	正規化	位置表現	FFN	密または MoE
OLMo 2/3（2024/25）(Team OLMo et al. 2025; Team OLMo 2025)	MHA（7B）、GQA（32B）、OLMo 3：4 層中 3 層でウィンドウ 4096	サブ層出力に RMSNorm、QK-Norm	RoPE	SwiGLU	密
DeepSeek-V3（2024）(DeepSeek-AI, Liu, et al. 2024)	MLA：KV を 512 次元の潜在表現に圧縮	RMSNorm、Pre-Norm	分離型 RoPE（64 次元の共有キー）	SwiGLU	MoE：256 エキスパート + 共有 1、8 個活性
Gemma 3（2025）(Gemma Team 2025)	GQA、局所対大域 5:1、ウィンドウ 1024	RMSNorm（Pre・Post 両方）、QK-Norm	RoPE、ベースは大域 10^6、局所 10^4	GeGLU	密
gpt-oss（2025）(OpenAI 2025)	GQA 64:8、交互の層でウィンドウ 128、学習型シンク	RMSNorm、Pre-Norm	RoPE、YaRN	クランプ付き SwiGLU	MoE：128 または 32 エキスパート、4 個活性

11.2.7 まとめ

デコーダのみの GPT は、L 個の因果的な `TransformerBlock` 層をトークン埋め込みテーブル $\mathbf{E} \in \mathbb{R}^{V \times d}$ と最終正規化および重み共有された出力射影 $\mathbf{E}^\top$ の間に配置し、全 T 個の系列位置における次トークンのロジットを 1 回の並列順伝播で予測する（Equation 11.7）。**AdamW**（2D 行列にのみ選択的に重み減衰を適用）、線形ウォームアップ、コサイン減衰、および勾配クリッピング（`LMOptimizer`）を用いて訓練すると、4.8M パラメータのバイトレベル `GPT` は 1,000 ステップ以内に英語テキストのバイグラムエントロピーを大きく下回る。パラメータ数と計算量を揃えた統制された比較では、`SwiGLU` は `GELU` を約 0.06 nat/バイト上回り（シード間のばらつきは 0.01 nat）、また `GPT` は条件を揃えた `RNNLM` に対して文脈位置 t が深くなるほど着実に差を広げていく。現代のオープンウェイト LLM（Table 11.2）はいずれもこの Pre-Norm の `RMSNorm` + `RoPE` + `SwiGLU`・`GeGLU` というレシピを共有しており、主な違いは KV キャッシュをどのように圧縮するか（11.3 節）、そして密な FFN を混合エキスパートに置き換えるかどうか（11.7 節）にある。

11.2.8 演習問題

1. **訓練レシピ（`LMOptimizer`）のアブレーション**。`LMOptimizer` は、選択的重み減衰（2D 重み行列のみに $\lambda = 0.1$）、線形ウォームアップ（100 ステップ）、コサイン減衰、および大域的勾配クリッピング（$\|\nabla \mathcal{L}\|_2 \le 1.0$）という 4 つの要素を組み合わせている。1 つずつ要素を無効化しながら、本節の 6 ブロックのバイトレベル `GPT` を 1,000 ステップ訓練せよ。(a) 埋め込みや RMSNorm のゲインを含む**すべて**のパラメータに重み減衰を適用する、(b) ウォームアップなし（`warmup=0`）、(c) 定数学習率（終始 `lr=2e-3`）、(d) 勾配クリッピングなし。検証損失を最も悪化させる単一の変更はどれか。

2. **GPT-2 チェックポイントのテンソル対応表の検証**。GPT-2 の `model.safetensors` ファイルには 148 個の重み・バイアステンソル（および 12 個の旧式の因果マスクバッファ `h.i.attn.bias`）が格納されている。`d2l.num_params(gpt2)` がファイル内の 148 個のパラメータテンソルの総要素数（124,439,808）と一致することを確かめる短い検証コードを書

き、さらに使用しているフレームワークの線形層の重み配置に基づいて、読み込み時に `c_attn.weight` を転置する必要があるかどうかを説明せよ。

3. **0 層の重み共有埋め込みベースライン**（$L = 0$）。`GPT`（`num_blks=0`）を考える。入力トークン x_t は $\mathbf{e}_{x_t}$ として埋め込まれ、`RMSNorm` で正規化された後、$\mathbf{E}^\top$ との内積によってロジット $\mathbf{z}_t \propto \mathbf{e}_{x_t}\mathbf{E}^\top$ を生成する。
   1. 0 ブロックのモデルは、どのクラスの統計的言語モデル（n グラムの次数）を表現できるか。
   2. 入力埋め込みと出力埋め込みを重み共有すると（$\mathbf{W}_{\text{LM}} = \mathbf{E}^\top$）、対称行列 $\mathbf{E}\mathbf{E}^\top$ が対角成分 $x_{t+1} = x_t$ に最大の成分を割り当ててしまうのはなぜか（ただし $\mathbf{E}$ のすべての行が等しいノルムをもつとする）。また、重みを共有しないモデルや 1 ブロックのモデルはどのようにしてこの対称性を破るか。
4. **トークナイザ間でのバイトあたりビット数の比較（バイトレベル GPT と BPE GPT-2）**。バイトレベルモデルは交差エントロピーを nat/バイトで報告するのに対し、GPT-2 は交差エントロピーを nat/サブワードトークンで報告する。
   1. 任意の N_B バイトの UTF-8 テキスト（`enc.encode` により N_T 個のサブワードトークンにトークン化されるもの）において、トークンあたりの平均の負の対数尤度が $\mathcal{L}_{\text{tok}}$（nat/トークン）であるサブワードモデルの厳密な**バイトあたりビット数（bpb）**が $\text{bpb} = \frac{N_T\,\mathcal{L}_{\text{tok}}}{N_B \ln 2}$ となることを示せ。
   2. WikiText-103 検証セットの先頭 10,000 バイト上で、本節の 4.8M パラメータのバイトレベル `model` と事前学習済みの 124M パラメータの `gpt2` の両方をバイトあたりビット数で評価せよ。
5. **Top-p（ニュークリアス）サンプリング**。`GPT.generate` を拡張してオプション引数 `top_p`（$0 < p < 1$）を追加し (Holtzman et al. 2020)、温度スケーリング後の確率を降順にソートしたうえで、累積確率が p を超える最小のプレフィックスより後ろのトークンをマスクするようにせよ。`gpt2` にプロンプト `"Alan Turing theorized that computers would"` を与え、`top_p=0.9, temperature=0.8` を用いた場合とフィルタリングなしの `temperature=1.2` を用いた場合とで生成結果を比較せよ。
6. **QK-Norm による学習率の安定性**。本節の 6 ブロックのバイトレベル `GPT` を、ピーク学習率 $\eta \in \{2\times10^{-3}, 6\times10^{-3}, 1.5\times10^{-2}\}$ にわたり、標準の `CausalAttention` を用いた場合と、`RMSNorm(d_h)` を `q` と `k` に適用する変種（`CausalAttention.qkv` の内部、11.1.2.5 節）を用いた場合とで訓練せよ。QK-Norm は安定に訓練できる最大学習率をどのように変化させるか。

11.3 KV キャッシュとキャッシュ削減

`GPT.generate`（11.2.4 節）がデコードステップ t で何をしているか詳しく見てみよう。トークン $t+1$ をサンプリングするために、プレフィックス $x_{1:t}$（長さ t）の全体に対して完全な順伝播を実行し、最後のロジットベクトル $\mathbf{z}_t \in \mathbb{R}^V$ のみを読み取って x_{t+1} を末尾に追加し、続くステップ $t+1$ では再び $x_{1:t+1}$ を全 L 個のブロックに最初から通し直している。因果アテンション（causal attention）は下三角構造をもつため、どの層においても位置 $s \le t$ の表現はトークン $1, \dots, s$ のみに依存する。したがって、位置 $1, \dots, t-1$ のキーと値をステップ t で再計算しても、ステップ $t-1$ で計算したものとまったく同じテンソルが再現されるだけである。T 個のトークンを生成する間に、この冗長な再計算によって延べ $O(T^2)$ 個のトークンが重み行列を通過することになるが、本来は T 個で済むはずである。

過去のキーと値を**キー・値（KV）キャッシュ**（key–value cache、Figure 11.4）に保存すれば、こ

の冗長性は解消される。ステップ t において、各層は単一の新しいトークン x_t のみを $(\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t)$ へ射影し、$(\mathbf{k}_t, \mathbf{v}_t)$ を事前確保されたバッファのスロット t に書き込み、$\mathbf{q}_t$ からキャッシュ済みのプレフィックス $(\mathbf{K}_{1:t}, \mathbf{V}_{1:t})$ に対してアテンションを計算する。

しかし、冗長な FLOPs を削減すると、今度は**メモリ容量とメモリ帯域幅**（memory bandwidth）というハードウェア上のボトルネックが露呈する。単一トークンのデコードでは、トークンを 1 つ生成するたびに、わずかな回数の行列・ベクトル積を実行するためだけにモデルの重みと KV キャッシュ全体を GPU メモリ（HBM）から読み出さなければならない。コンテキストが長い場合やバッチが大きい場合には、KV キャッシュは数十ギガバイトにまで膨れ上がり、デコードのスループットを頭打ちにする。本節の前半では、キャッシュを用いた順伝播を構築し、ハードウェアのルーフライン上で**プレフィル**、**デコード**、**バッチ化**（batching）を解析する。後半（11.3.3 節）では、KV キャッシュをその 3 つの次元、すなわちヘッド数（**GQA/MQA**）、チャネル幅（**低ランクのキー・値射影**）、系列長（**アテンションシンク付きスライディングウィンドウ**）に沿って圧縮する。

```
from functools import partial
```

11.3.1 再計算からキャッシュへ

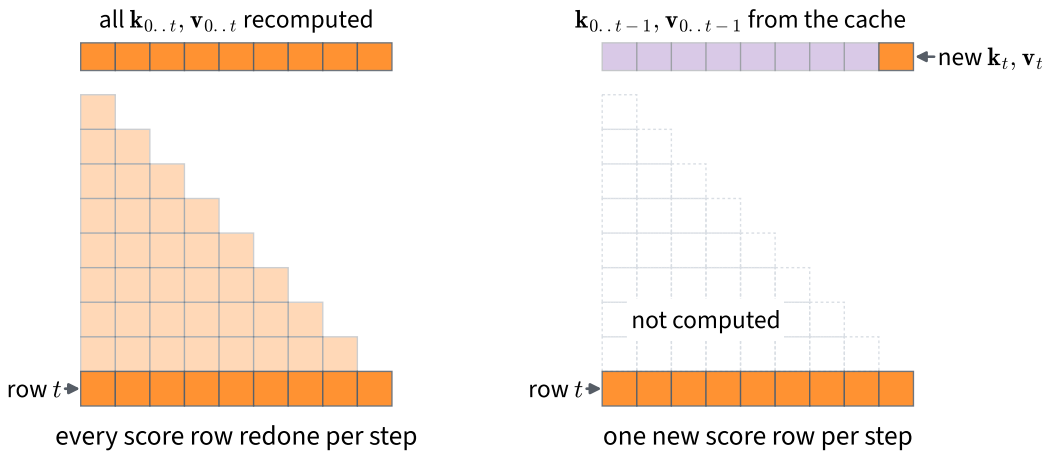


Figure 11.4: 位置 t における 1 デコードステップ。素朴な生成では、プレフィックスのすべてのキー・値・スコア行を再計算したうえで最後の行のみを使用する。KV キャッシュを用いれば、過去のキーと値はメモリから読み出され、そのステップでは新しい単一のキー・値・スコア行のみを計算する。

11.3.1.1 キャッシュを用いた順伝播

各ブロックでキャッシュするのがキーと値 $(\mathbf{K}_{1:t}, \mathbf{V}_{1:t})$ であり、クエリ $\mathbf{Q}_{1:t}$ や残差状態 $\mathbf{H}_{1:t}$ ではないのはなぜだろうか。因果マスキング（causal masking）のもとでは、クエリ $\mathbf{q}_s$（$s < t$）はトークン $s+1$ を出力するときにしか使われず、その後二度と参照されない。また、**RoPE 適用後**のキー $\mathbf{k}_s$ と値 $\mathbf{v}_s$（いずれも形状は $(B, H_{\text{kv}}, 1, d_h)$）をキャッシュしておけば、過去のトークンを $\mathbf{W}_k, \mathbf{W}_v$ で再射影したり RoPE で再回転させたりする必要も一切なくなる。

`init_cache` は、ゼロ初期化したキーと値のバッファ `ks`, `vs`（形状 $(B, H_{\text{kv}}, T_{\max}, d_h)$）を L 個の

各ブロックに対して事前確保する。Figure 11.5 に生成途中のバッファの 1 つを示す。位置 t より前のスロットはすでに埋まっており、現在のステップでスロット t に書き込み、残りのスロットは 0 のままマスクで隠されている。

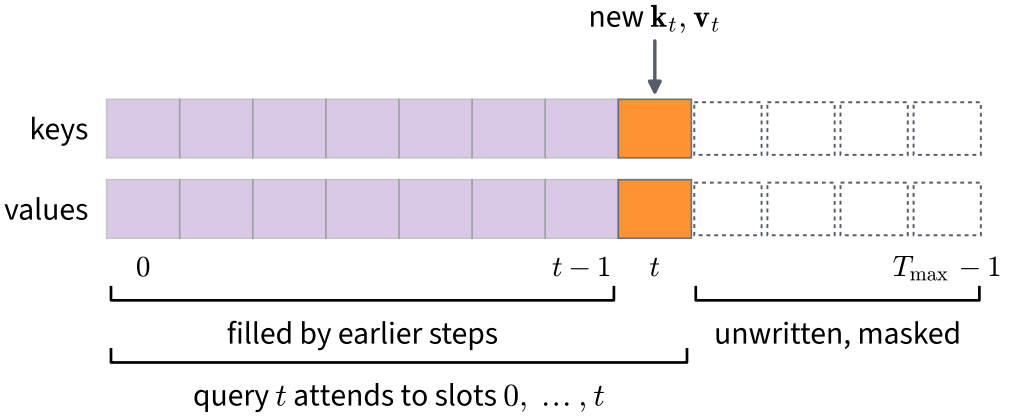


Figure 11.5: 位置 t のステップにおける 1 ブロック分のキャッシュバッファ（1 系列、1 キー・値ヘッド）。スロット $0, \ldots, t-1$ には以前のステップのキーと値が保持されている。このステップでは $\mathbf{k}_t$ と $\mathbf{v}_t$ をスロット t に書き込む。t より後のスロットは 0 のままでマスクによって隠されるため、クエリ t はスロット $0, \ldots, t$ にアテンションを向ける。

`forward_cached(model, X, ks, vs, t)` は、スライス `X` に含まれる n 個の新しいトークンを位置オフセット `t` から処理し（RoPE に `offset=t` を渡すか `pos_emb[t:t+n]` をインデックス参照する）、それらのキーと値を `[:, :, t:t+n, :]` に書き込み、`[:, :, :t+n, :]` に対してアテンションを計算する。

```
def init_cache(model, batch_size, max_len):
    """Zeroed keys and values: for every block one buffer of each, of
    shape (batch, max_len, kv heads, head width)."""
    attn = model.blks[0].attention
    shape = (batch_size, max_len, attn.num_kv_heads,
             attn.kv_hiddens // attn.num_kv_heads)
    dtype = model.token_emb.embedding[...].dtype
    cache = [jnp.zeros(shape, dtype) for _ in range(2 * len(model.blks))]
    return cache[::2], cache[1::2]

def forward_cached(model, X, ks, vs, t):
    """Logits after the last of the new tokens X, which sit at positions
    t, t + 1, ...; writes their keys and values into the cache."""
    assert all(blk.pre_norm for blk in model.blks), 'pre-norm blocks only'
    B, T = X.shape
    pos = t + jnp.arange(T)  # t: an int or a traced value
    H, tables = model.token_emb(X), None
    if model.pos == 'learned':
        H = H + model.pos_emb(pos)
    else:  # RoPE angles of positions t, ..., t + T - 1, for every block
        tables = d2l.rope_tables(T, model.head_width, t)
    # Query t + i sees the filled positions 0, ..., t + i of the buffer
```

```
    mask = jnp.arange(ks[0].shape[1]) <= pos[:, None]
    ks, vs = list(ks), list(vs)
    for i, blk in enumerate(model.blks):  # Pre-norm, as in d2l.GPT
        attn = blk.attention
        q, k, v = (u.swapaxes(1, 2)  # (batch, T, heads, head width)
                   for u in attn.qkv(blk.norm1(H), t, tables))
        # A write past the end of the buffer would not fail: XLA moves the
        # start back, so the caller keeps t + T within max_len
        ks[i] = jax.lax.dynamic_update_slice(ks[i], k, (0, t, 0, 0))
        vs[i] = jax.lax.dynamic_update_slice(vs[i], v, (0, t, 0, 0))
        Y = jax.nn.dot_product_attention(q, ks[i], vs[i], mask=mask)
        H = H + attn.W_o(Y.reshape(B, T, -1))
        H = H + blk.ffn(blk.norm2(H))
    return model.token_emb.attend(model.norm(H[:, -1])), ks, vs
```

forward_cached を用いることで、generate_cached は生成を 2 つの段階に分割する。(1) 長さ T_p のプロンプト全体を 1 回の並列呼び出し（t = 0, n = T_p）で処理する**プレフィル**、および (2) 新たにサンプリングされたトークンをステップごとに 1 つずつ処理する（n = 1、オフセット t = T_p, T_p+1, ...）**デコード**である。

```
def sample(logits, key, temperature=1.0, top_k=None):
    """One token per row of logits (batch, vocab), as generate draws it."""
    logits = logits / temperature
    if top_k is not None:
        cutoff = jax.lax.top_k(logits, top_k)[0][:, -1:]
        logits = jnp.where(logits < cutoff, -jnp.inf, logits)
    return jax.random.categorical(key, logits)[:, None]

@partial(jax.jit, static_argnames=('graphdef', 'num_tokens', 'top_k'),
         donate_argnames=('ks', 'vs'))
def decode(graphdef, params, X, ks, vs, t, num_tokens, key,
           temperature=1.0, top_k=None):
    """Feed X at positions t, t + 1, ..., then sample num_tokens tokens
    per sequence; returns them as (batch, num_tokens), and the cache."""
    model = nnx.merge(graphdef, params)
    logits, ks, vs = forward_cached(model, X, ks, vs, t)  # Prefill
    first = sample(logits, jax.random.fold_in(key, 0), temperature, top_k)
    tokens = jnp.zeros((X.shape[0], num_tokens), jnp.int32).at[:, 0].set(
        first[:, 0])
    def step(i, carry):  # Feed token i - 1, sample token i
        ks, vs, tokens = carry
        logits, ks, vs = forward_cached(model, tokens[:, i - 1, None], ks, vs,
                                        t + X.shape[1] + i - 1)
        token = sample(logits, jax.random.fold_in(key, i), temperature, top_k)
        return ks, vs, tokens.at[:, i].set(token[:, 0])
    ks, vs, tokens = jax.lax.fori_loop(1, num_tokens, step, (ks, vs, tokens))
    return tokens, ks, vs

@d2l.add_to_class(d2l.GPT)
def generate_cached(self, prefix, num_tokens, temperature=1.0, top_k=None):
    """Sample as generate does, but never recompute the prefix."""
    assert len(prefix) + num_tokens <= self.max_len, 'cache overflow'
```

```
    ks, vs = init_cache(self, 1, len(prefix) + num_tokens)
    tokens = decode(*nnx.split(self), jnp.asarray(prefix)[None], ks, vs, 0,
                    num_tokens, d2l.get_key(), temperature, top_k)[0]
    return list(prefix) + jax.device_get(tokens)[0].tolist()
```

11.3.1.2 厳密な等価性の検証

generate_cached の実行時間を計測する前に、プロンプトをプレフィルしてからキャッシュを通じて 1 トークンずつステップを進める処理が、系列全体に対して一度に model(X) を実行した場合と（浮動小数点の丸め誤差 $\sim 10^{-6}$ の範囲内で）**まったく同じロジット**を生成することを、pos='rope' と pos='learned' の双方について確認しよう。

```
def gpt(*args, **kwargs):  # Fresh random weights for every model
    return d2l.GPT(*args, **kwargs, rngs=nnx.Rngs(d2l.get_key()))
```

ここで、3 ブロックの GPT を回転位置埋め込み（pos='rope'）と学習可能な絶対位置埋め込み（pos='learned'）の両方でインスタンス化し、先頭の 16 トークン（全長 48 トークンの系列のうち）を 1 回の呼び出しでプレフィルしたのち、残りの 32 位置を 1 トークンずつ進め、キャッシュなしの完全な順伝播 model(x) に対する最大絶対誤差を計測する。

```
with jax.default_matmul_precision('highest'):
    for pos in ('rope', 'learned'):
        model = gpt(vocab_size=97, num_hiddens=128, num_heads=4,
                    num_blks=3, pos=pos)
        model.eval()
        x = jax.random.randint(d2l.get_key(), (1, 48), 0, 97)
        full = model(x)[0, 15:]  # Logits after positions 15, ..., 47
        ks, vs = init_cache(model, 1, 48)
        logits, ks, vs = forward_cached(model, x[:, :16], ks, vs, 0)
        outs = [logits]
        for t in range(16, 48):
            logits, ks, vs = forward_cached(model, x[:, t:t + 1], ks, vs, t)
            outs.append(logits)
        err = jnp.abs(jnp.concatenate(outs) - full).max()
        print(f'{pos}: max |full - cached| = {err:.1e}')
```

```
rope: max |full - cached| = 9.5e-07
learned: max |full - cached| = 6.0e-07
```

11.3.1.3 二次から線形への高速化の計測

12 層の GPT-2 規模のモデル（$d = 768, H = 12, L = 12$）において、n トークンに対するキャッシュなしの順伝播 1 回分の実時間（time_forward、素朴なデコードではこれを**すべての**ステップで繰り返す）と、コンテキスト長（context length）n におけるキャッシュありのデコードステップ 1 回分の実時間（time_decode）を比較すると、長文脈（long-context）の生成になぜキャッシュが不可欠であるかがわかる。

```
model = gpt(vocab_size=50257, num_hiddens=768, num_heads=12, num_blks=12,
            max_len=4096)  # GPT-2's vocabulary and size
```

```
model.eval()
print(f'{d2l.num_params(model) / 1e6:.0f}M parameters')
```

```
124M parameters
```

続いて、d2l.Benchmark を用いて 2 つの計測用ヘルパー関数を定義する。time_forward(model, n) は n トークンに対するコンパイル済みの完全なプレフィックス順伝播を計測し、time_decode(model, batch_size, n) は 32 回の単一トークン・キャッシュ付きデコードステップ（コンテキスト長 n から開始）の実時間レイテンシの平均を求める。

```
@partial(jax.jit, static_argnames='graphdef')
def last_logits(graphdef, params, X):
    """The step of naive generation: a forward pass over all of X."""
    return nnx.merge(graphdef, params)(X)[:, -1]

def time_forward(model, n):
    """Seconds for a forward pass over n random tokens."""
    graphdef, params = nnx.split(model)
    x = jax.random.randint(d2l.get_key(), (1, n), 0,
                           model.token_emb.num_embeddings)
    return d2l.Benchmark(lambda: last_logits(graphdef, params, x)).time

def time_decode(model, batch_size, n):
    """Seconds per decode step for batch_size sequences at context n,
    averaged over 32 steps."""
    graphdef, params = nnx.split(model)
    x = jax.random.randint(d2l.get_key(), (batch_size, 1), 0,
                           model.token_emb.num_embeddings)
    cache = init_cache(model, batch_size, n + 32)
    def run():
        nonlocal cache  # decode consumes the cache and returns it updated
        tokens, *cache = decode(graphdef, params, x, *cache, n - 1, 32,
                                d2l.get_key())
        return tokens
    return d2l.Benchmark(run).time / 32
```

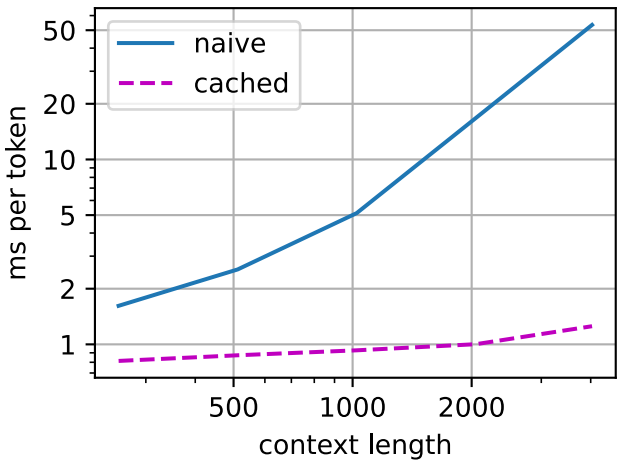
コンテキスト長 n を 256 から 4,032 トークンまで 2 のべき乗に沿って変化させると、キャッシュなしの各ステップが線形に増大するのに対し、キャッシュ付きデコードステップのレイテンシがほぼ一定に保たれるという対比が鮮明になる。

```
lengths, t_naive, t_cached = [256, 512, 1024, 2048, 4032], [], []
for n in lengths:
    t_naive.append(time_forward(model, n))
    t_cached.append(time_decode(model, 1, n))
    print(f'n={n:4d}: naive {1e3 * t_naive[-1]:5.1f} ms, '
          f'cached {1e3 * t_cached[-1]:4.1f} ms per token')
```

```
n= 256: naive   1.6 ms, cached  0.8 ms per token
n= 512: naive   2.5 ms, cached  0.9 ms per token
n=1024: naive   5.1 ms, cached  0.9 ms per token
n=2048: naive  16.7 ms, cached  1.0 ms per token
n=4032: naive  53.4 ms, cached  1.3 ms per token
```

トークンあたりのレイテンシ曲線を両対数軸でプロットすると、素朴なステップのコストがコンテキスト長 n に比例して増大する一方、キャッシュ付きステップはほぼ平坦であることがわかる（新しいクエリとキャッシュ済みの全 n 個のキーとのスコア計算は依然として必要だが、そのコストの定数係数は小さい）。

```
d2l.plot(lengths, [[1e3 * t for t in t_naive], [1e3 * t for t in t_cached]],
         'context length', 'ms per token', legend=['naive', 'cached'],
         xscale='log', yscale='log')
```



新しい 256 トークン（3,584 トークンのプロンプトに続く部分）をエンドツーエンドで生成する全体の累積時間で比べると、`model.generate` を `model.generate_cached` に置き換えることで実時間が 1 桁以上高速化される。

```
prefix = list(range(3584))  # Any token ids will do for timing
for name, gen in (('naive ', model.generate),
                  ('cached', model.generate_cached)):
    t = d2l.Benchmark(lambda: gen(prefix, 256), warmup=1, repeats=1).time
    print(f'{name}: 256 tokens in {t:4.1f} s = {256 / t:5.1f} tokens/s')
```

```
naive : 256 tokens in 15.2 s =  16.8 tokens/s
cached: 256 tokens in  0.4 s = 641.6 tokens/s
```

11.3.2 メモリ容量とハードウェアのルーフライン

ブロック数 L、バッチサイズ B、系列長（sequence length）n、および H_{kv} 個のキー・値ヘッド（各ヘッド幅 d_h）をもち、数値 1 個あたり b バイト（FP32 では $b = 4$、BF16/FP16 では 2、FP8 では 1）で保存するモデルでは、各ブロックが 2 つのテンソル（$\mathbf{K}$ と $\mathbf{V}$、形状 $(B, H_{\text{kv}}, n, d_h)$）を保持する。したがって、KV キャッシュ全体のメモリ使用量は次の式で与えられる。

$$M_{\text{KV}}(n, B) \;=\; 2\, b\, B\, L\, n\, H_{\text{kv}}\, d_h \;\text{ bytes.} \tag{11.8}$$

```
def kv_cache_bytes(model, num_tokens, batch_size=1, bytes_per_elem=4):
    """Bytes of the keys and values of num_tokens positions in every block
    of a GPT, for batch_size sequences."""
    attn = model.blks[0].attention
    return (2 * len(model.blks) * attn.kv_hiddens * num_tokens * batch_size
            * bytes_per_elem)
```

kv_cache_bytes を 124M パラメータの FP32 モデルに対して $n = 4{,}096$ トークンで評価すると、確保されたキャッシュバッファのサイズが Equation 11.8（288 MiB）と一致し、すでにモデルの重みのメモリ使用量である 471 MiB の半分を超えていることが確認できる。

```
bytes_per_param = 4  # fp32 weights
weight_bytes = bytes_per_param * d2l.num_params(model)
ks, vs = init_cache(model, 1, 4096)
print(f'cache buffers {sum(u.nbytes for u in ks + vs) / 2**20:.0f} MiB, '
      f'formula {kv_cache_bytes(model, 4096) / 2**20:.0f} MiB, '
      f'weights {weight_bytes / 2**20:.0f} MiB')
```

```
cache buffers 288 MiB, formula 288 MiB, weights 471 MiB
```

11.3.2.1 ルーフライン上のプレフィルとデコード

なぜ 2,048 トークンのプレフィル 1 回あたりの所要時間は、単一トークンのデコードステップ 15 回分程度とほぼ同じ桁に収まったのだろうか。**ルーフラインモデル**（Figure 11.6）は、この違いを**演算密度**（arithmetic intensity、HBM と演算ユニットの間で転送される 1 バイトあたりに実行される FLOPs）によって説明する。

- **プレフィル（$n = 2048$ トークンの並列処理、計算律速）：**各重み行列 **W** は HBM から 1 回読み出されて Bn 個のトークンベクトルと乗算される（$2BnN_{\text{params}}$ FLOPs の演算に対し重みの転送量は $b_w N_{\text{params}}$ バイト）。これにより演算密度は $\sim 2Bn/b_w \gg 80$ FLOPs/byte となり、GPU のリッジポイントを大きく上回ってテンソルコアを飽和させる。
- **単一ストリームのデコード（ステップあたり $B = 1, n = 1$ トークン、メモリ帯域幅律速）：**各ステップでは行列・ベクトル積としてわずか $2N_{\text{params}}$ FLOPs しか実行しない一方で、HBM から**すべての**重み $b_w N_{\text{params}}$ バイトに加えてキャッシュ $M_{\text{KV}}(n, 1)$ バイトを読み出すため、演算密度は $\leq 2/b_w \approx 0.5$ FLOPs/byte（FP32 の場合。BF16 では 1.0 FLOP/byte）となり、計算ルーフより $100\times$ 以上低い水準にとどまる。

```
t_prefill = time_forward(model, 2048)
t_decode = time_decode(model, 1, 2048)
```

得られたトークンスループットと演算密度を 1 TB/s の HBM 帯域幅の上限値と比較すると、単一ストリームのデコードがメモリ帯域幅の上限付近にとどまる一方、2,048 トークンのプレフィルがそれより $100\times$ 以上高いスループットを達成する理由がわかる。

```
bandwidth = 1e12  # bytes/s, about an RTX 4090
flops = 2 * d2l.num_params(model)  # One decode step at context 2048
moved = weight_bytes + kv_cache_bytes(model, 2048)
print(f'prefill: {2048 / t_prefill:6.0f} tokens/s')
print(f'decode:  {1 / t_decode:6.0f} tokens/s, bandwidth ceiling '
```

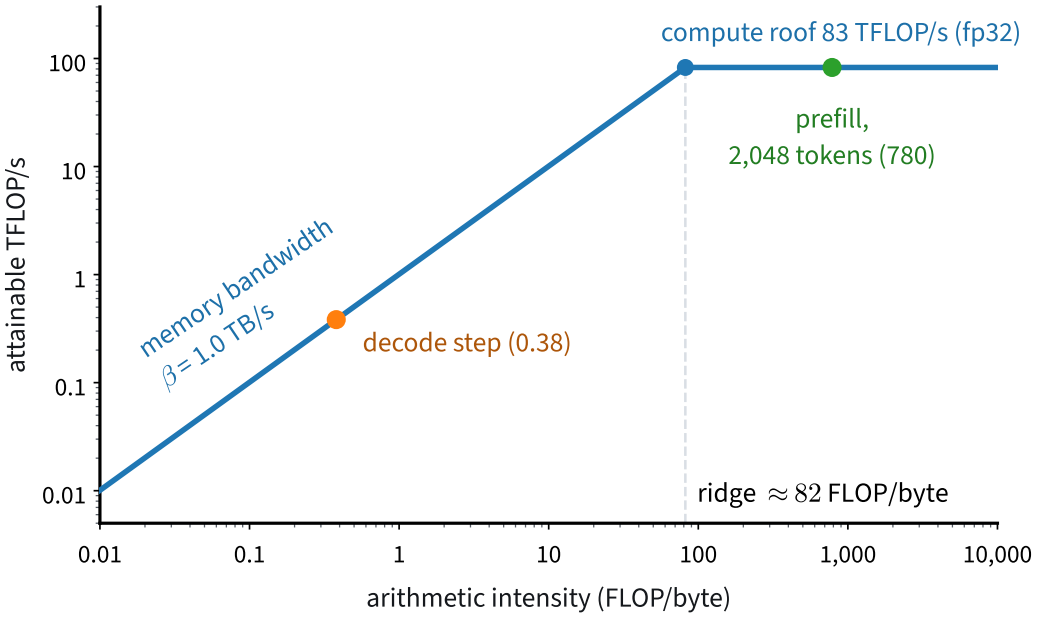


Figure 11.6: RTX 4090 における fp32 のルーフライン。達成可能なスループットは、計算ルーフ（約 83 TFLOP/s）と、演算密度にメモリ帯域幅（約 1 TB/s）を掛けた値のうち小さい方であり、両者は約 82 FLOPs/byte のリッジポイントで交わる。コンテキスト 2,048 トークンにおける GPT-2 規模のモデルのデコードステップは 0.38 FLOPs/byte であり、帯域幅の傾斜部のずっと下方に位置する。一方、2,048 トークンのプレフィルは、重みとキャッシュのみを数えた場合で約 780 FLOPs/byte となり、計算ルーフの下に位置する。

```
        f'{bandwidth / moved:.0f} tokens/s')
print(f'FLOPs per byte: decode {flops / moved:.2f}, '
      f'prefill about {2048 * flops / moved:.0f}')
```

```
prefill: 121629 tokens/s
decode:     997 tokens/s, bandwidth ceiling 1550 tokens/s
FLOPs per byte: decode 0.38, prefill about 784
```

11.3.2.2 バッチデコード

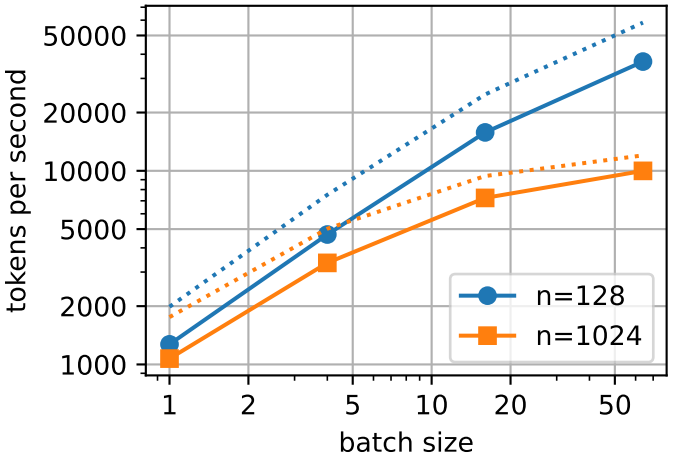
単一ストリームのデコードステップ（$B = 1$）は大半の時間を HBM からモデルの重み $M_{\text{weights}} = b_w N_{\text{params}}$ バイトを読み出すことに費やすため、B 本の系列からなるバッチを同時にデコードすれば、1 回の重みの読み出しを全 B トークンで再利用しつつ、B 個の個別の KV キャッシュ（$B\,M_{\text{KV}}(n,1)$ バイト）を読み出すだけで済む。メモリ帯域幅を $\mathcal{B}_{\text{HBM}}$（bytes/s）とすると、帯域幅で制限されるデコードスループット（tokens/s）は次の式で抑えられる。

$$\text{Throughput}(B,n) \;\le\; \frac{B\,\mathcal{B}_{\text{HBM}}}{M_{\text{weights}} + B\,M_{\text{KV}}(n,1)}. \tag{11.9}$$

バッチサイズが小さい領域（$B\,M_{\text{KV}} \ll M_{\text{weights}}$）では、スループットは B に対してほぼ線形に増加する（「無償のバッチ化」）。一方、B または n が大きくなると（$B\,M_{\text{KV}} \gg M_{\text{weights}}$）、KV キャッシュの読み出しが HBM の通信量を支配するようになり、スループットは **KV 帯域幅の上限** $\mathcal{B}_{\text{HBM}}/M_{\text{KV}}(n,1)$ で飽和する。

```
batches, measured, bounds = [1, 4, 16, 64], [], []
for n in (128, 1024):
    measured.append([b / time_decode(model, b, n) for b in batches])
    nc = kv_cache_bytes(model, n)  # One sequence's cache
    bounds.append([b * bandwidth / (weight_bytes + b * nc) for b in batches])
    print(f'n={n}: ' + ', '.join(f'{r:.0f}' for r in measured[-1])
          + ' tokens/s')
d2l.plot(batches, measured + bounds, 'batch size', 'tokens per second',
         legend=['n=128', 'n=1024'], fmts=('C0-o', 'C1-s', 'C0:', 'C1:'),
         xscale='log', yscale='log', legend_loc='lower right')
```

```
n=128: 1270, 4689, 15790, 36660 tokens/s
n=1024: 1072, 3346, 7245, 9994 tokens/s
```



11.3.3 KV キャッシュの縮小

メモリ容量（Equation 11.8）と大バッチ時のデコードスループットの上限（Equation 11.9）はどちらも系列あたりのキャッシュサイズ $M_{\text{KV}}(n,1) = 2bLnH_{\text{kv}}d_h$ によって決まるため、現代の Transformer アーキテクチャは積 $n \cdot H_{\text{kv}} \cdot d_h$ を互いに直交する 3 つの軸に沿って圧縮する。

1. **キー・値ヘッド数の削減（$H_{\text{kv}} < H$）**：グループ化クエリアテンション（grouped-query attention、GQA）(Ainslie et al. 2023) およびマルチクエリアテンション（MQA）(Shazeer 2019) は、各キー・値ヘッドを $G = H/H_{\text{kv}}$ 個のクエリヘッドで共有する（Figure 11.7）。
2. **トークンあたりの表現幅の縮小（$2H_{\text{kv}}d_h \to r$）**：低ランクのキー・値射影およびマルチヘッド潜在アテンション（Multi-Head Latent Attention、MLA）(DeepSeek-AI, Liu, et al. 2024) は、全ヘッドのキーと値を低次元の潜在ベクトルに圧縮する（11.3.3.2 節）。
3. **キャッシュするコンテキストの短縮（$n \to w + s$）**：スライディングウィンドウアテンションはウィンドウ幅 w より古いトークンを破棄する一方、先頭の $s \geq 1$ 個の**アテンションシンク**トークンを保持することで (Xiao et al. 2024)、ソフトマックスの分母を安定に保つ（11.3.3.3 節）。

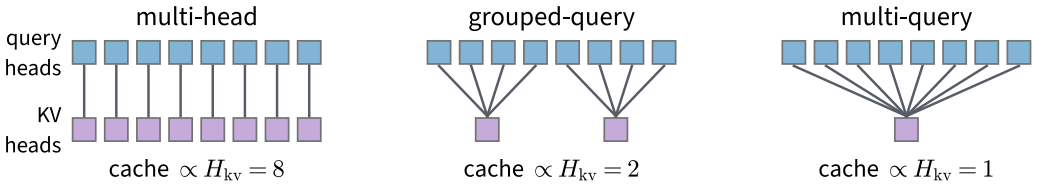


Figure 11.7: クエリヘッドとそれらが読み出すキー・値ヘッド。マルチヘッドアテンションは各クエリヘッドに固有のキー・値ヘッドを与え、グループ化クエリアテンションはクエリヘッドのグループごとに 1 つを共有し、マルチクエリアテンションはすべてのクエリヘッドで単一のヘッドを共有する。キャッシュのサイズはキー・値ヘッドの数に比例する。

11.3.3.1 軸 1：グループ化クエリアテンションとチェックポイントのアップトレーニング

`CausalAttention` クラス（11.2 節）はすでに引数 `num_kv_heads` を受け取るようになっている。$H_{\text{kv}} = H/G$ と設定すると $\mathbf{W}_{qkv}$ が小さくなり、M_{KV} は $G = H/H_{\text{kv}}$ 分の 1 に削減される。さ

らに Ainslie et al. (2023) は、既存のマルチヘッドのチェックポイント（$H_{\text{kv}} = H$）について、G 個ずつのキーおよび値の射影ヘッドのグループに**平均プーリング**（mean-pooling）を施し（`pool_heads`）、元の訓練計算量のわずかな割合（~ 5–10%）だけ**アップトレーニング**することで、GQA や MQA モデルへ変換できることを示した。

6 ブロックのバイトレベル `GPT`（$H = 8$）を 1,000 ステップにわたり $H_{\text{kv}} \in \{8, 2\}$ でゼロから訓練した場合（および $H_{\text{kv}} = 2$ で 50 ステップのみ訓練した場合）と、訓練済みの $H_{\text{kv}} = 8$ チェックポイントを平均プーリングまたは各グループの先頭ヘッドのみを残す方法で $H_{\text{kv}} = 2$ に変換した場合とを、ゼロショット（0 ステップ）および 50・100 ステップ（10%）のアップトレーニング後について、2 回の実行で比較する。

```
def pool_heads(target, source, pool):
    """Load the weights of source into target, which has fewer key-value
    heads. Each key (and value) projection of target is the mean of those
    of a group of consecutive heads of source (pool='mean'), or the first
    of them (pool='first')."""
    src, tgt = source.blks[0].attention, target.blks[0].attention
    group = src.num_kv_heads // tgt.num_kv_heads
    def convert(path, W):
        if 'W_qkv' not in jax.tree_util.keystr(path):
            return W
        d = W.shape[0]  # Kernel columns: d queries, keys, values
        q, k, v = jnp.split(W, [d, d + src.kv_hiddens], axis=1)
        k, v = (u.reshape(d, tgt.num_kv_heads, group, -1) for u in (k, v))
        k, v = ((u.mean(2) if pool == 'mean' else u[:, :, 0]).reshape(d, -1)
                for u in (k, v))
        return jnp.concatenate([q, k, v], axis=1)
    nnx.update(target, jax.tree_util.tree_map_with_path(
        convert, nnx.state(source, nnx.Param)))
    return target
```

それでは、6 ブロックのバイトレベル `GPT` を `WikiText103` 上で $H_{\text{kv}} \in \{8, 2\}$ の設定により訓練し、`pool_heads` を `'mean'` と `'first'` の両方のヘッドプーリング手法について 0、50、100 ステップのアップトレーニングにわたり評価しよう。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)

def trained(num_kv_heads, num_steps=1000):
    """A d2l.GPT trained from scratch with the recipe of the GPT section."""
    lm = gpt(len(data.vocab), num_kv_heads=num_kv_heads)
    optimizer = d2l.LMOptimizer(lm, lr=2e-3, num_steps=num_steps,
                                warmup=min(100, num_steps // 10))
    d2l.train_steps(lm, data, optimizer, num_steps)
    return lm

def val_loss(lm):
    return math.log(d2l.lm_perplexity(lm, data.val_dataloader()))

for run in (1, 2):
    mha = trained(8)
    print(f'run {run}, trained from scratch: H_kv=8 {val_loss(mha):.2f}, '
          f'H_kv=2 {val_loss(trained(2)):.2f}, '
```

```
            f'H_kv=2 for 50 steps only {val_loss(trained(2, 50)):.2f}')
    for pool in ('mean', 'first'):
        lm = pool_heads(gpt(len(data.vocab), num_kv_heads=2), mha, pool)
        losses = [val_loss(lm)]
        optimizer = d2l.LMOptimizer(lm, lr=3e-4, num_steps=10**9, warmup=0)
        for _ in range(2):  # Uptraining at a constant rate
            d2l.train_steps(lm, data, optimizer, 50)
            losses.append(val_loss(lm))
        print(f'  H_kv=8 converted by {pool:5s}: {losses[0]:.2f}, after 50 '
              f'steps {losses[1]:.2f}, after 100 steps {losses[2]:.2f}')
```

```
run 1, trained from scratch: H_kv=8 1.34, H_kv=2 1.34, H_kv=2 for 50 steps only 2.66
  H_kv=8 converted by mean : 2.85, after 50 steps 1.68, after 100 steps 1.48
  H_kv=8 converted by first: 2.96, after 50 steps 1.80, after 100 steps 1.64
run 2, trained from scratch: H_kv=8 1.34, H_kv=2 1.34, H_kv=2 for 50 steps only 2.67
  H_kv=8 converted by mean : 2.75, after 50 steps 1.62, after 100 steps 1.46
  H_kv=8 converted by first: 2.87, after 50 steps 1.72, after 100 steps 1.57
```

4 つのクエリヘッドでキーと値を共有しても（$H_{\text{kv}} = 2$、KV キャッシュを $4\times$ 縮小）、ゼロから訓練すれば測定可能なほどの品質低下は生じない（1.34〜1.35 対 1.34 nats/byte）。ヘッドプーリング直後は 2.8 nats/byte 付近から始まるが、100 ステップのアップトレーニングによって平均プーリングを施したモデルは約 1.47 まで改善し、元の性能までの差の約 90% を取り戻す。これに対し、ゼロから 50 ステップ訓練しただけでは約 2.7 にしか届かず、各グループの先頭ヘッドのみを残す方法はアップトレーニング後も平均プーリングに劣る。

11.3.3.2 軸 2：低ランクのキー・値射影

ヘッド数 H_{kv} を減らす代わりに、$2d$ 次元の連結されたキー・値ベクトル $[\mathbf{k}_t; \mathbf{v}_t] \in \mathbb{R}^{2d}$（各トークン、層 i）をランク r の部分空間（subspace、$r \ll 2d$）へ圧縮できないだろうか。これを事前学習済みの **GPT-2 Small**（124M、$d = 768$、$2d = 1536$）に対し、$T = 1024$ トークンからなる『The Time Machine』の一節を用いて直接検証する。

```
gpt2, enc = d2l.load_gpt2(), d2l.gpt2_tokenizer()
fname = d2l.download(d2l.DATA_URL + 'timemachine.txt', '../data',
                     '090b5e7e70c295757f55df93cb0a180b9691891a')
tokens = jnp.asarray(enc.encode(open(fname).read()))
x, calib = tokens[None, :1024], tokens[None, 1024:2048]
print(f'{len(tokens)} tokens of The Time Machine')
```

```
44073 tokens of The Time Machine
```

GPT を書き換えずにすべての層の内部でキーと値を横取りして変換できるよう、`forward_blocks(model, X, attn_fn)` を定義してブロック i のアテンション計算をコールバック `attn_fn(q, k, v, i)` に委譲する。そして、標準の因果アテンションが `gpt2(x)` と 10^{-5} 以内で一致することを確認するとともに、ベースラインとなる `tail_loss` を位置 $512 \ldots 1023$ 上で求めておく。

```
def forward_blocks(model, X, attn_fn):
    """The forward pass of a d2l.GPT with attention computed by
    attn_fn(q, k, v, i) in block i, on inputs of shape (batch, heads,
    length, head width)."""
    assert all(blk.pre_norm for blk in model.blks), 'pre-norm blocks only'
```

```
    H = model.token_emb(X)
    if model.pos == 'learned':
        H = H + model.pos_emb(jnp.arange(X.shape[1]))
    for i, blk in enumerate(model.blks):
        attn = blk.attention
        Y = attn_fn(*attn.qkv(blk.norm1(H)), i)
        H = H + attn.W_o(Y.swapaxes(1, 2).reshape(*X.shape, -1))
        H = H + blk.ffn(blk.norm2(H))
    return model.token_emb.attend(model.norm(H))

def tail_loss(logits, X):
    """Mean loss of the next-token predictions from position 512 on."""
    return optax.softmax_cross_entropy_with_integer_labels(
        logits[0, 512:-1], X[0, 513:]).mean()

def causal(q, k, v, i):
    """Fused causal attention; the kernel wants (batch, length, ...)."""
    return jax.nn.dot_product_attention(
        *(u.swapaxes(1, 2) for u in (q, k, v)), is_causal=True).swapaxes(1, 2)

logits = forward_blocks(gpt2, x, causal)
print(f'helper vs model: max deviation '
      f'{jnp.abs(logits - gpt2(x)).max():.1e}')
full = tail_loss(logits, x)
print(f'full attention: loss {full:.2f}, perplexity {jnp.exp(full):.0f}')
```

```
helper vs model: max deviation 0.0e+00
full attention: loss 3.58, perplexity 36
```

書籍の続く 1,024 トークン（`calib`）を用いて各層の連結されたキー・値行に対する経験的な PCA/SVD 基底 $\mathbf{U}_i \in \mathbb{R}^{2d\times r}$ を求め（`fit_bases`）、評価対象の一節 `x` のキーと値を上位 $r \in \{512, 256, 128\}$ 方向へ射影する（`projected`）ことで、再訓練なしでも GPT-2 の KV キャッシュがどこまで圧縮可能かを計測する。

```
def kv_rows(k, v):
    """Keys and values of all heads of each token as one row: (T, 2d)."""
    return jnp.concatenate(
        [u[0].swapaxes(0, 1).reshape(u.shape[2], -1) for u in (k, v)], 1)

def fit_bases(model, X):
    """Right singular vectors of the key-value rows of X in every block."""
    bases = []
    def fit(q, k, v, i):
        bases.append(jnp.linalg.svd(kv_rows(k, v), full_matrices=False)[2])
        return causal(q, k, v, i)
    forward_blocks(model, X, fit)
    return bases

def projected(bases, r):
    """Attention with keys and values projected onto the top r bases."""
    def attn_fn(q, k, v, i):
        P = bases[i][:r]  # (r, 2d), orthonormal rows
```

```
            KV = kv_rows(k, v) @ P.T @ P
            k, v = (u.reshape(u.shape[0], k.shape[1], -1).swapaxes(0, 1)[None]
                    for u in jnp.split(KV, 2, axis=1))
            return causal(q, k, v, i)
        return attn_fn

    bases = fit_bases(gpt2, calib)
    for r in (512, 256, 128):
        loss = tail_loss(forward_blocks(gpt2, x, projected(bases, r)), x)
        print(f'rank {r} of 768: loss {loss:.2f}, '
              f'perplexity {jnp.exp(loss):.0f}')
```

```
rank 512 of 768: loss 3.55, perplexity 35
rank 256 of 768: loss 3.61, perplexity 37
rank 128 of 768: loss 4.14, perplexity 63
```

各ブロックは $2d$ 個のキーと値を 1 つの d 次元入力から 1 つの行列と 1 つのバイアスで計算するため、それらは高々 $d = 768$ 次元のアフィン部分空間に収まる（出力の「of 768」はこの次元数を指す）。事後的な線形射影であっても、GPT-2 の KV ベクトルは測定可能な劣化なしに $6\times$（$r = 256$、元の次元は $2d = 1536$）まで圧縮できる（3.61 対 3.58 nats）が、$r = 128$ まで絞ると 0.5 nat 超の悪化が生じる。一方、DeepSeek-V2/V3 の**マルチヘッド潜在アテンション（MLA）**(DeepSeek-AI, Liu, et al. 2024) のように低ランクのボトルネックをエンドツーエンドで訓練すれば、単一の 512 次元潜在ベクトルで全ヘッドのキーと値を置き換え、$> 16\times$ の圧縮を達成できる。

11.3.3.3 軸 3：スライディングウィンドウとアテンションシンク

3 つ目の軸は、キャッシュに保持する系列長 n を直近 w トークンのローリングウィンドウに制限することである（10.6 節）。しかし、事前学習済み GPT-2 に対して 1,024 トークンの一節上で純粋なスライディングウィンドウ（$w = 256$）を適用すると、後半トークン（$t \geq 512$）の交差エントロピー（cross-entropy）は ~ 3.6 nats から 8 nats 超へと跳ね上がってしまう。

その理由を探るため、`sink_mass` によって位置 $t \geq 64$ のクエリが**トークン 0**（系列の先頭トークン）に割り当てる平均アテンション確率を、GPT-2 の全 12 層のそれぞれについて計測してみよう。

```
def sink_mass(model, X):
    """Mean attention weight on token 0 of the queries from position 64
    on, one entry per block."""
    masses = []
    def watch(q, k, v, i):  # Explicit causal attention
        scores = q @ k.swapaxes(-1, -2) / math.sqrt(q.shape[-1])
        alpha = d2l.masked_softmax(scores, d2l.causal_mask(q.shape[2]))
        masses.append(alpha[..., 64:, 0].mean())
        return alpha @ v
    forward_blocks(model, X, watch)
    return masses

masses = sink_mass(gpt2, x)
print('mean attention weight on token 0 (uniform would be about 0.003)')
print('block ' + ''.join(f'{i:6d}' for i in range(len(masses))))
print('weight' + ''.join(f'{m:6.2f}' for m in jax.device_get(masses)))
```

```
mean attention weight on token 0 (uniform would be about 0.003)
block      0     1     2     3     4     5     6     7     8     9    10    11
weight  0.00  0.01  0.11  0.30  0.32  0.51  0.46  0.57  0.48  0.48  0.45  0.34
```

GPT-2 の中間層から上位層にかけて、クエリは**全アテンション重みの 30% から 60% 近くをトークン 0 に割り当てる**が、これは位置 0 にどの単語があるかによらない。ソフトマックス確率の総和は 1 でなければならないため、アテンションヘッドは局所的に一致するトークンがないとき、**すべての**因果クエリから参照可能な唯一の位置である先頭付近のトークンをデフォルトの**アテンションシンク** (Xiao et al. 2024) として利用するのである。純粋なスライディングウィンドウが位置 0 を通り過ぎると（$t \geq w$）、ソフトマックスの分母からトークン 0 が取り除かれることで、その 30–60% もの確率質量が残りのウィンドウ内トークンへ無理やり押し付けられ、残差ストリーム（residual stream）がかき乱される。

先頭のわずか $s \in \{1, 4\}$ 個のシンクトークンをキャッシュの先頭に固定したまま直近 $w = 256$ 個のトークンと併せて保持すれば（キャッシュされるトークン数は合計 $w + s$ 個）、GPT-2 の検証損失はフルコンテキストのベースラインとほぼ同等の水準まで回復する。

```
j = jnp.arange(x.shape[1])
allowed = d2l.causal_mask(x.shape[1])
w = 256  # The window: each query sees itself and the 255 tokens before it
window = allowed & (j[:, None] - j[None, :] < w)
for nsink in (0, 1, 4):
    mask = window | (allowed & (j[None, :] < nsink))
    loss = tail_loss(forward_blocks(
        gpt2, x, lambda q, k, v, i: jax.nn.dot_product_attention(
            *(u.swapaxes(1, 2) for u in (q, k, v)),
            mask=mask).swapaxes(1, 2)), x)
    print(f'window {w} + {nsink} sink tokens: loss {loss:.2f}, '
          f'perplexity {jnp.exp(loss):.0f}')
print(f'full attention:            loss {full:.2f}, '
      f'perplexity {jnp.exp(full):.0f}')
```

```
window 256 + 0 sink tokens: loss 8.25, perplexity 3838
window 256 + 1 sink tokens: loss 3.86, perplexity 47
window 256 + 4 sink tokens: loss 3.76, perplexity 43
full attention:            loss 3.58, perplexity 36
```

11.3.4 キャッシュ削減手法の比較

Table 11.3: $M_{\text{KV}} = 2bBLnH_{\text{kv}}d_h$ の 3 つの因子に沿った KV キャッシュ削減機構のまとめ。

手法	削減対象の因子	系列あたりの保存要素数	訓練の要否
KV ヘッドの共有（GQA、MQA） (Shazeer 2019; Ainslie et al. 2023)	ヘッド数 $H_{\text{kv}} < H$	$2LnH_{\text{kv}}d_h$	要：ゼロからの訓練またはアップトレーニング（$\sim$ 5–10%）
低ランク潜在表現（MLA） (DeepSeek-AI, Liu, et al. 2024)	幅 $2H_{\text{kv}}d_h \to r$	$\approx Lnr$（小さな回転キーを追加）	MLA は要。事後的な SVD 基底でも $\sim 6\times$ まで有効
シンク付きスライディングウィンドウ (Xiao et al. 2024)	長さ $n \to w + s$	$2L(w+s)H_{\text{kv}}d_h$	$s \geq 1$ なら事前学習済みモデルにゼロショットで有効

11.3.5 まとめ

KV キャッシュは過去のトークンの RoPE 適用後のキーと値を保存することで、各デコードステップでの射影を単一の新しいトークン（`n = 1`）のみにとどめ、同一のロジットを生成しながら T トークン生成にわたる射影の計算量を $O(T^2)$ から $O(T)$ 回のトークン評価へと削減する（アテンションスコア計算も $O(T^3)$ から $O(T^2)$ へ減少する）。ハードウェアのルーフライン（Figure 11.6）上では、サービングは計算律速の**プレフィル**（$n = T_p$ 個のプロンプトトークンを高い演算密度で並列処理する段階）と、メモリ帯域幅律速の**デコード**（ステップあたり $n = 1$ トークンを低い演算密度で処理する段階）に分かれる。**バッチデコード**は並行する B 本の系列間で重みの読み出しを償却し（Equation 11.9）、スループットを B に比例して伸ばすが、やがて KV キャッシュの転送量が支配的になると $\mathcal{B}_{\text{HBM}}/M_{\text{KV}}(n,1)$ で飽和する。メモリ容量と大バッチ時のスループットの双方を引き上げるため、現代のアーキテクチャは $M_{\text{KV}} = 2bLnH_{\text{kv}}d_h$（Equation 11.8）を 3 つの直交軸（Table 11.3）、すなわちヘッド数（**GQA/MQA**）、チャネル幅（**低ランク KV / MLA**）、系列長（**アテンションシンク付きスライディングウィンドウ**）に沿って圧縮する。

11.3.6 演習問題

1. **70B モデルのキャッシュメモリとバッチ上限**。Llama 3.1 70B は $L = 80$ 層、隠れ層サイズ $d = 8192$、$H = 64$ 個のクエリヘッド（$d_h = 128$）、$H_{\text{kv}} = 8$ 個のキー・値ヘッドをもち、BF16（$b = 2$ バイト）で保存される。
   1. Equation 11.8 を用いて、単一系列の KV キャッシュサイズ $M_{\text{KV}}(n,1)$ を最大コンテキスト長 $n = 131{,}072$（128K）において計算し、標準の MHA（$H_{\text{kv}} = 64$）を用いた場合のキャッシュサイズと比較せよ。
   2. 合計 $8 \times 80 = 640$ GB の HBM（うち 140 GB は BF16 のモデル重みが占有）を備える 8-GPU ノードにおいて、最大バッチサイズ B（長さ 128K トークンの系列を残りの 500 GB に収められる本数）は、GQA（$H_{\text{kv}} = 8$）と MHA（$H_{\text{kv}} = 64$）のそれぞれでいくつになるか。
2. **バッチデコードのスループットと KV 帯域幅の上限**。 1 の 70B モデルを、コンテキスト長 $n = 32{,}768$、総 HBM 帯域幅 $\mathcal{B}_{\text{HBM}} = 24$ TB/s、$M_{\text{weights}} = 140$ GB のシステム上でサービングするとする。
   1. 帯域幅による上界 Equation 11.9 を、バッチサイズ $B \in \{1, 8, 32, 128, 512\}$ において GQA（$H_{\text{kv}} = 8$）と MHA（$H_{\text{kv}} = 64$）の両方について評価せよ。
   2. バッチサイズ $B_{1/2}$ がいくつのとき、各構成は漸近的な上限 $\lim_{B\to\infty} \text{Throughput}(B,n) = \mathcal{B}_{\text{HBM}}/M_{\text{KV}}(n,1)$ の半分に到達するか。
3. **K 個の出力系列間でのプレフィックス共有**。バッチ内の $K = 16$ 本の系列が $n_{\text{shared}} = 4096$ トークンの共通システムプロンプトを共有し、それぞれが固有のトークンを $n_{\text{unique}} = 256$ 個生成したとする。共有プレフィックスの KV キャッシュを保存し、ステップごとに全 K 系列に対して **1 回**だけ読み出す場合について Equation 11.9 の分母を修正し、70B GQA モデルにおける $K = 16$ でのスループット向上率を計算せよ。
4. **INT8 / FP8 による KV キャッシュの量子化**。`forward_blocks` 内で GPT-2 のキャッシュされるキーと値を、トークンごとの対称 8 ビット整数量子化を用いて保存せよ。すなわち、各トークンベクトル $\mathbf{u} \in \mathbb{R}^d$ に対してスケール $s = \max_j |u_j|/127$ を計算し、$\tilde{\mathbf{u}} = s \cdot \text{round}(\mathbf{u}/s)$ と量子化して、アテンション計算に $\tilde{\mathbf{u}}$ を用いる。`tail_loss`（`x` 上）の変化を、キーと値を 8 ビットに量子化した場合（FP32 キャッシュより 4× 縮小、BF16 より 2× 縮小）と 4 ビットに量子化した場合（`max / 7`、それぞれ 8× / 4× 縮小）とで計測せよ。
5. **アテンションシンクと仮想シンクロジット（$\textbf{softmax}_1$）**。Xiao et al. (2024) および gpt-oss

（Table 11.2）が指摘しているように、標準のソフトマックスはアテンション重みの総和を 1 に強制するため、ヘッドにはアテンションシンクが必要となる。

1. `forward_blocks` 内のソフトマックスを $\mathrm{softmax}_1(\mathbf{s})_j = \frac{e^{s_j}}{e^{b}+\sum_m e^{s_m}}$ に置き換えよ（値ベクトルを一切追加せず、非トークンのシンクロジット $b = 5.0$ を分母に加える）。
2. 純粋なスライディングウィンドウ（$w = 256, s = 0$ 個の保持シンクトークン）を GPT-2 上で $b = 5.0$ として評価せよ。分母に加えた仮想ロジットによって、$s = 0$ における損失の急増はどれだけ解消されるか。

6. **低ランクのキー・値復元の $\mathbf{W}_q$ および $\mathbf{W}_o$ への吸収**。層 ℓ が各トークン j を単一の潜在ベクトル $\mathbf{c}_j \in \mathbb{R}^r$（$r \ll d_h H$）に圧縮し、そこからヘッド m のキーと値を $\mathbf{k}_{j,m} = \mathbf{c}_j \mathbf{B}_{k,m}$ および $\mathbf{v}_{j,m} = \mathbf{c}_j \mathbf{B}_{v,m}$（$\mathbf{B}_{k,m}, \mathbf{B}_{v,m} \in \mathbb{R}^{r \times d_h}$）によって復元するとする。

   1. $\mathbf{q}_{t,m}\mathbf{k}_{j,m}^\top = (\mathbf{x}_t \mathbf{W}_{q,m}\mathbf{B}_{k,m}^\top)\mathbf{c}_j^\top$ および $\big(\sum_j \alpha_{t,j}^{(m)} \mathbf{v}_{j,m}\big)\mathbf{W}_{o,m} = \big(\sum_j \alpha_{t,j}^{(m)} \mathbf{c}_j\big)(\mathbf{B}_{v,m}\mathbf{W}_{o,m})$ が成り立つことを示せ。
   2. デコード時に $\mathbf{B}_{k,m}^\top$ を $\mathbf{W}_{q,m}$ へ、$\mathbf{B}_{v,m}$ を $\mathbf{W}_{o,m}$ へ吸収することで、完全なキー $\mathbf{k}_{j,m}$ や値 $\mathbf{v}_{j,m}$ を HBM 上に一切実体化せずに済む理由を説明せよ。また、DeepSeek-V2/V3 が RoPE を幅の狭い別個の分離されたキー $\mathbf{k}_j^{\mathrm{RoPE}}$ に適用し、$\mathbf{c}_j \mathbf{B}_{k,m}$ を直接回転させない理由を説明せよ。

7. **固定サイズのストリーミング・リングバッファキャッシュの実装**。`StreamingCache` 版の `init_cache` と `forward_cached` を作成し、層あたり $s + w$ 個のトークンスロットのみを事前確保するようにせよ（先頭の $s = 4$ 個のシンク位置をスロット $0 \ldots s-1$ に保持し、循環バッファ内のスロット $s + ((t - s) \bmod w)$ を $t \geq s$ に対して上書きする）。そして、一定の $O(s + w)$ キャッシュメモリで 2,048 トークンを生成できることを確認せよ。なお、`gpt2` は学習済みの位置を 1,024 個しかもたないため、位置インデックスを上限なく増やせる回転位置埋め込み（`pos='rope'`）をもつモデルを用いるか、あるいは `gpt2` の各トークンにそのキャッシュスロットの位置を割り当てよ（Xiao et al. (2024) と同様の手法である）。

11.4 エンコーダ、デコーダ、およびクロスアテンション

これまでに構築してきた Transformer スタックはすべて**因果デコーダ**（causal decoder）（11.2 節）であった。すなわち、三角マスクによってトークン i のアテンション先をプレフィックス位置 $j \leq i$ に制限し、1 回の順伝播を T 個の並列な次トークン予測（next-token prediction）へと変換するものであった。推論時には未来のトークンがまだ存在しないため、トークンを 1 つずつ生成する際には因果マスキング（causal masking）が不可欠である。しかし、検索用の文書埋め込み、トークン分類、翻訳のための原言語文のエンコード、あるいは画像パッチグリッドのエンコード（11.5 節）のように、入力系列の**全体**が最初から得られている場合、位置 i が位置 $j > i$ を読み取ることを禁じると、周囲の文脈の半分を捨てることになる。

計算開始時点でどのトークンが利用可能か、また入力と出力が同じ系列に属するかに応じて、同じ `TransformerBlock`（11.1 節）から 3 つの標準的な配線（wiring）が構成できる（Figure 11.8）。

1. **エンコーダのみ（双方向自己アテンション）**：すべてのトークンがすべてのトークンにアテンションを向け（`is_causal=False`）、完全な入力系列の文脈を考慮した表現を 1 回の順伝播で計算する（BERT (Devlin et al. 2018)、RoBERTa (Liu et al. 2019)、ViT (Dosovitskiy et al. 2021)）。
2. **デコーダのみ（因果自己アテンション）**：各トークンは自身とそれ以前の位置のみにアテンションを向け（`is_causal=True`）、単一の均質なスタックでプロンプトの読み取りと自己回帰生成の両方を担う（GPT、Llama 3、Qwen3）。

3. **エンコーダ・デコーダ（双方向エンコーダ＋クロスアテンション付き因果デコーダ）**：双方向エンコーダがソース系列 $\mathbf{X}_{\text{src}}$ を文脈表現 $\mathbf{H}_{\text{enc}}$ に写像し、因果デコーダがターゲット系列 $\mathbf{Y}_{\text{tgt}}$ を、**クロスアテンション**によって $\mathbf{H}_{\text{enc}}$ にクエリを送りながら 1 トークンずつ生成する（2017 年のオリジナル Transformer (Vaswani et al. 2017)、T5 (Raffel et al. 2020)、Whisper (Radford et al. 2023)）。

11.4.1 エンコーダ、デコーダ、およびエンコーダ・デコーダアーキテクチャ

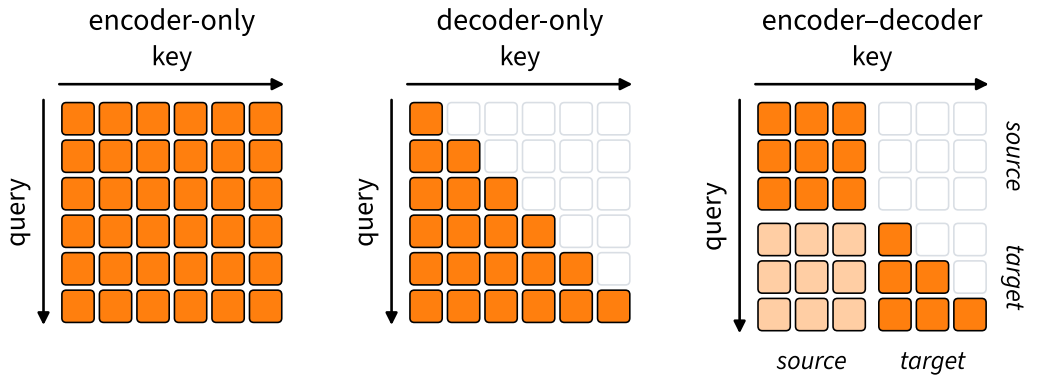


Figure 11.8: 3 つの配線とそのアテンションパターン（クエリ位置は下方向、キー位置は右方向に進み、塗りつぶされたセルはそのクエリがそのキーにアテンションを向けられることを表す）。エンコーダのみ：すべてのトークンがすべてのトークンにアテンションを向ける。デコーダのみ：各トークンは自身とその過去にのみアテンションを向ける。エンコーダ・デコーダ：ソースは自身に双方向のアテンションを向け、各ターゲットトークンはクロスアテンションによってソース全体（淡色のセル）にアテンションを向けつつ自身の過去には因果的にアテンションを向ける。3 番目のパネルには構造上の事実が 1 つ隠れている。すなわち、ソースの行とターゲットの行は 2 つの別々のスタックに属しており、すべてのデコーダブロックは最後のエンコーダブロックの出力を読み取る。

11.4.2 双方向文脈とマスクトークン予測

11.4.2.1 双方向エンコーダ

`d2l.TransformerBlock` から因果マスクを取り除くと、因果スタックは**双方向 Transformer エンコーダ**（`TransformerEncoder`）になり、トークン ID の入力バッチ（および任意のパディングマスク）を形状 (B, n, d) の文脈表現へと写像する。

```
class TransformerEncoder(nnx.Module):
    """Bidirectional encoder: embeddings plus unmasked pre-norm blocks."""
    def __init__(self, vocab_size, num_hiddens=128, num_heads=4, num_blks=4,
                 max_len=64, *, rngs):
        init = nnx.initializers.normal(0.02)
        self.token_emb = nnx.Embed(vocab_size, num_hiddens,
                                   embedding_init=init, rngs=rngs)
        self.pos_emb = nnx.Embed(max_len, num_hiddens, embedding_init=init,
                                 rngs=rngs)
        self.blks = nnx.List([
```

```
            d2l.TransformerBlock(num_hiddens, num_heads, rngs=rngs)
            for _ in range(num_blks)])
        self.norm = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)

    def __call__(self, X, mask=None):
        H = self.token_emb(X) + self.pos_emb(jnp.arange(X.shape[1]))
        for blk in self.blks:
            H = blk(H, mask)  # by default no mask: all tokens see all
        return self.norm(H)
```

11.4.2.2 マスク言語モデリング（MLM）の目的関数

なぜ双方向スタックは 11.2 節の次トークン予測の目的関数では訓練できないのだろうか。位置 t のターゲットはトークン x_{t+1} であり、そのトークンは位置 $t+1$ の入力そのものである。マスクがなければ、わずか 1 ブロックでも位置 t が位置 $t+1$ にアテンションを向けてそこにある自身のターゲットを読み取れてしまうため（Figure 11.9）、次トークン予測は自明なコピーへと退化してしまう。

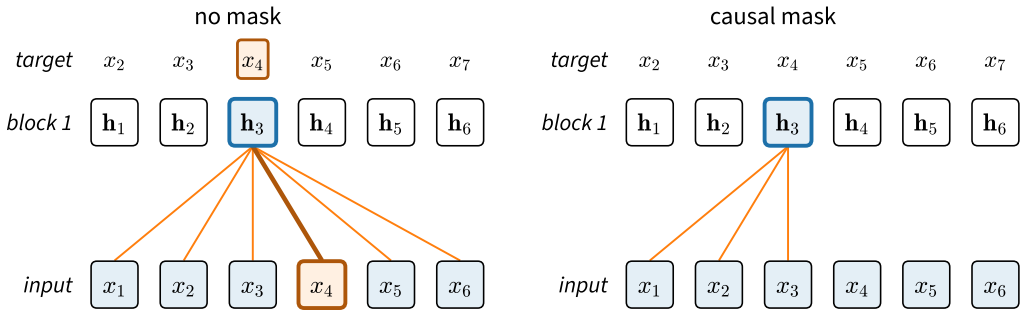


Figure 11.9: 1 ブロックにおける次トークン予測。マスクがない場合、位置 3 は自身のターゲットである x_4 を含むすべての入力にアテンションを向ける。因果マスクがある場合、位置 1 から 3 のみを読み取る。

BERT (Devlin et al. 2018) で導入された**マスク言語モデリング**（masked language modeling、MLM）は、次トークン予測を穴埋め課題（cloze task）に置き換える。すなわち、位置の部分集合 $M \subset \{1, \dots, n\}$（通常はトークンの $p = 15\%$）をランダムに選び、各 x_i（$i \in M$）を専用の <mask> トークン ID に置き換えて破損入力 $\tilde{\mathbf{x}}$ を作り、**マスクされた位置のみ**（$i \in M$）で元のトークンを復元する交差エントロピーを最小化する。

$$\mathcal{L}_{\text{MLM}}(\theta) \;=\; -\frac{1}{|M|} \sum_{i \in M} \log P_\theta(x_i \mid \tilde{\mathbf{x}}). \tag{11.10}$$

4 ブロックのバイトレベル `MaskedLM`（$d = 128, H = 4, L = 4$）を、11.2 節とまったく同じ WikiText-103 のバイトストリーム上で訓練する。その際、トークン ID `MASK = len(data.vocab)`（256）を <mask> 用に確保する。

```
data = d2l.WikiText103(batch_size=64, num_steps=64)
MASK = len(data.vocab)  # id of the extra <mask> embedding row
```

```
def mask_tokens(X, key, p=0.15):
    mask = jax.random.uniform(key, X.shape) < p
    return jnp.where(mask, MASK, X), mask

class MaskedBytes:
    """The windows of data with masked bytes; labels -100 add no loss."""
    def __init__(self, data):
        self.data = data

    def train_dataloader(self):
        return self.masked(self.data.train_dataloader())

    def val_dataloader(self):
        return self.masked(self.data.val_dataloader())

    def masked(self, batches):
        key = d2l.get_key()
        for X, _ in batches:
            key, sub = jax.random.split(key)
            Xm, mask = mask_tokens(jnp.asarray(X), sub)
            yield Xm, jnp.where(mask, X, -100)

class MaskedLM(nnx.Module):
    """An encoder with an output head tied to its token embedding."""
    def __init__(self, encoder):
        self.encoder = encoder

    def __call__(self, X):
        return self.encoder.token_emb.attend(self.encoder(X))

model = MaskedLM(TransformerEncoder(len(data.vocab) + 1,
                                    rngs=nnx.Rngs(d2l.get_key())))
optimizer = nnx.Optimizer(model, optax.adamw(1e-3, weight_decay=0.0),
                          wrt=nnx.Param)
losses = d2l.train_steps(model, MaskedBytes(data), optimizer, 2000)
print('masked loss at step 500/1000/2000: ' + '/'.join(
    f'{sum(losses[k-50:k]) / 50:.2f}' for k in (500, 1000, 2000)))
```

```
masked loss at step 500/1000/2000: 3.14/2.11/1.63
```

11.4.2.3 利用可能な文脈ごとの損失

11.2.3.2 節では、因果言語モデルの位置ごとの検証損失は左端（$t = 0$、1 バイトの文脈）から右端（$t = T-1$、完全なプレフィックス文脈）に向かって単調に減少していた。では、`loss_by_position` を用いて双方向の `MaskedLM` における各ウィンドウ位置 $t \in \{0, \dots, n-1\}$ での復元損失を評価すると、どのような曲線になるだろうか。

```
@nnx.jit
def scored_losses(model, X, Y):
    logits = model(X)
    losses = optax.softmax_cross_entropy_with_integer_labels(
```

```
            logits, jnp.maximum(Y, 0))
        return losses, logits.argmax(-1) == Y

    def loss_by_position(model, batches):
        """Mean loss at each window position over the labels that are not -100,
        and the fraction of those labels predicted exactly."""
        model.eval()
        total = count = correct = 0
        for X, Y in batches:
            Y = jnp.asarray(Y)
            losses, hit = scored_losses(model, jnp.asarray(X), Y)
            scored = Y != -100
            total += (losses * scored).sum(0)
            count += scored.sum(0)
            correct += (hit & scored).sum()
        return total / count, float(correct / count.sum())

    def show(losses):
        return ' '.join(f'{float(v):.2f}' for v in losses)

    masked = MaskedBytes(data)
    pos_loss, accuracy = loss_by_position(model, masked.val_dataloader())
    interior = slice(16, 48)  # positions 16 to 47 of the 64
    print(f'masked accuracy {accuracy:.2f}')
    print(f'loss at positions 0-3: {show(pos_loss[:4])}; 60-63: '
          f'{show(pos_loss[-4:])}; interior mean {pos_loss[interior].mean():.2f}')
```

```
masked accuracy 0.52
loss at positions 0-3: 2.42 1.76 1.56 1.55; 60-63: 1.57 1.56 1.79 2.44; interior mean 1.56
```

双方向エンコーダは左右対称な**バスタブ型（U 字型）曲線**を描く。内部のどの位置も左右両方の文脈を利用でき（約 1.5〜1.6 nats/byte）、損失が上昇するのはウィンドウ境界によって片側の文脈が断ち切られる両端（$t \approx 0$ および $t \approx n-1$、約 2.4 nats/byte）だけである。マスクされた文で `model` を評価すると、右側の文脈が内部の欠損バイトの曖昧性をどのように解消するかがわかる。

```
    snippet = 'Homarus gammarus , known as the European lobster or common '
    ids = jnp.asarray(data.vocab.encode(snippet))[None]
    for pos in (11, 22, 28):
        Xm = ids.at[0, pos].set(MASK)
        probs = jax.nn.softmax(model(Xm)[0, pos])
        top = jnp.argsort(probs)[::-1][:3]
        shown = snippet[:pos] + '_' + snippet[pos + 1:]
        print(f'{shown[:40]!r}... -> '
              + ', '.join(f'{data.vocab.decode([int(i)])!r} {probs[i]:.2f}'
                          for i in top))
```

```
'Homarus gam_arus , known as the European'... -> 'p' 0.63, ' ' 0.08, 'm' 0.06
'Homarus gammarus , kno_n as the European'... -> 'w' 0.66, 'h' 0.18, 'r' 0.04
'Homarus gammarus , known as _he European'... -> 't' 0.97, 'T' 0.02, 'w' 0.00
```

11.4.3 エンコーダ・デコーダにおけるクロスアテンション

11.4.3.1 アライメントが既知の系列対系列課題

見通しのよい系列対系列（sequence-to-sequence）のアライメント問題で**エンコーダ・デコーダ**（Figure 11.8 右）を調べるため、$V = 16$ 文字のアルファベット（a〜p）と長さ $T = 12$ からなる文字列反転でモデルを訓練する。ソース文字列 $\mathbf{x} = (x_1, \dots, x_T)$ が与えられたとき、デコーダは $\text{dec_in} = (\texttt{<bos>}, x_T, x_{T-1}, \dots, x_2)$ を受け取り、反転した文字列 $\mathbf{y} = (x_T, x_{T-1}, \dots, x_1)$ を予測する。

```
V, T = 16, 12   # 16 letters, strings of length 12
BOS = V         # decoder start token

def sample_batch(batch_size, key):
    src = jax.random.randint(key, (batch_size, T), 0, V)
    tgt = src[:, ::-1]
    dec_in = jnp.concatenate(
        [jnp.full((batch_size, 1), BOS), tgt[:, :-1]], 1)
    return src, dec_in, tgt

def to_str(ids):
    return ''.join(chr(ord('a') + int(i)) for i in ids)

src, dec_in, tgt = sample_batch(1, d2l.get_key())
print(f'source {to_str(src[0])!r} -> target {to_str(tgt[0])!r}')
```

```
source 'mcikbijmbcah' -> target 'hacbmjibkicm'
```

11.4.3.2 デコーダブロックと EncoderDecoder

DecoderBlock は、因果自己アテンションと SwiGLU FFN の間に**クロスアテンション**のサブ層（d2l.MultiHeadAttention）を挿入する。クロスアテンションでは、**クエリはデコーダから得られ**（$\mathbf{Q} = \text{Norm}(\mathbf{Y})\mathbf{W}_q \in \mathbb{R}^{B\times T_{\text{dec}}\times d}$）、**キーと値はエンコーダの最上層から得られる**（$\mathbf{K} = \mathbf{H}_{\text{enc}}\mathbf{W}_k \in \mathbb{R}^{B\times T_{\text{src}}\times d}$、$\mathbf{V} = \mathbf{H}_{\text{enc}}\mathbf{W}_v \in \mathbb{R}^{B\times T_{\text{src}}\times d}$）。エンコードされたソース $\mathbf{H}_{\text{enc}}$ の全体がデコーディング開始前にすでに存在するため、クロスアテンションでは**因果マスクを使用しない**。

```
class DecoderBlock(nnx.Module):
    """Pre-norm decoder block: causal self-attention, cross-attention,
    FFN."""
    def __init__(self, num_hiddens, num_heads, *, rngs):
        self.norm1 = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.norm2 = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.norm3 = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.self_attention = d2l.MultiHeadAttention(num_hiddens, num_heads,
                                                     rngs=rngs)
        self.cross_attention = d2l.MultiHeadAttention(num_hiddens, num_heads,
                                                      rngs=rngs)
        self.ffn = d2l.FeedForward(num_hiddens, rngs=rngs)

    def __call__(self, X, enc, mask=None, cross_mask=None):
        if mask is None:  # causal self-attention
```

```
            mask = d2l.causal_mask(X.shape[1])
        Y = self.norm1(X)
        X = X + self.self_attention(Y, Y, Y, mask)
        X = X + self.cross_attention(self.norm2(X), enc, enc, cross_mask)
        return X + self.ffn(self.norm3(X))
```

EncoderDecoder は、TransformerEncoder と DecoderBlock 層のスタックおよび線形出力ヘッドを組み合わせたものである。なお、decode(dec_in, enc) は事前計算されたエンコーダ出力 enc を受け取るため、自己回帰生成の際にはソース系列に対してエンコーダが **1 回だけ**実行され、その出力を使って decode がターゲットトークンを 1 ステップずつ進める。

```
class EncoderDecoder(nnx.Module):
    """A causal decoder cross-attending into a bidirectional encoder."""
    def __init__(self, vocab_size, num_hiddens=128, num_heads=4,
                 num_blks=1, max_len=64, *, rngs):
        init = nnx.initializers.normal(0.02)
        self.encoder = TransformerEncoder(vocab_size, num_hiddens,
                                          num_heads, num_blks, max_len,
                                          rngs=rngs)
        self.tgt_emb = nnx.Embed(vocab_size, num_hiddens,
                                 embedding_init=init, rngs=rngs)
        self.pos_emb = nnx.Embed(max_len, num_hiddens, embedding_init=init,
                                 rngs=rngs)
        self.blks = nnx.List([DecoderBlock(num_hiddens, num_heads,
                                           rngs=rngs)
                              for _ in range(num_blks)])
        self.norm = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.head = d2l.Linear(num_hiddens, vocab_size, use_bias=False,
                               rngs=rngs)

    def decode(self, dec_in, enc):
        H = self.tgt_emb(dec_in) + self.pos_emb(jnp.arange(dec_in.shape[1]))
        for blk in self.blks:
            H = blk(H, enc)
        return self.head(self.norm(H))

    def __call__(self, src, dec_in):
        return self.decode(dec_in, self.encoder(src))
```

11.4.3.3 訓練と自己回帰デコーディング

EncoderDecoder を 128 個の文字列からなるランダムバッチ上で 300 ステップ訓練し、貪欲な自己回帰デコーディング（reverse）を実行すると、ほぼ完璧な文字列全体の反転が達成される。

```
@nnx.jit
def s2s_step(model, optimizer, src, dec_in, tgt):
    def loss_fn(model):
        logits = model(src, dec_in)
        return optax.softmax_cross_entropy_with_integer_labels(
            logits.reshape(-1, logits.shape[-1]), tgt.reshape(-1)).mean()
```

```
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    optimizer.update(model, grads)
    return loss

def train_reversal(model, num_steps=300):
    """Train on fresh batches of 128 strings; return the loss of each step."""
    optimizer = d2l.LMOptimizer(model, lr=1e-3, num_steps=num_steps,
                                warmup=0, weight_decay=0.0)
    key, losses = d2l.get_key(), []
    for step in range(num_steps):
        key, sub = jax.random.split(key)
        src, dec_in, tgt = sample_batch(128, sub)
        losses.append(s2s_step(model, optimizer, src, dec_in, tgt))
    return jax.device_get(losses)  # One transfer at the end

seq2seq = EncoderDecoder(V + 1, rngs=nnx.Rngs(d2l.get_key()))
losses = train_reversal(seq2seq)
print('loss at step 25/50/100/200/300: ' + '/'.join(
    f'{sum(losses[k-10:k]) / 10:.3f}' for k in (25, 50, 100, 200, 300)))
```

```
loss at step 25/50/100/200/300: 2.090/1.126/0.120/0.004/0.001
```

推論時には、`reverse(model, src)` がソース系列を 1 回エンコードし、<bos> からステップごとに 1 つずつターゲットトークンを貪欲にデコードすることで、ホールドアウトされた 1,000 個のランダム文字列に対してほぼ完璧な完全一致を達成する。

```
@nnx.jit
def next_token(model, enc, buf, t):
    return model.decode(buf, enc)[:, t].argmax(-1)

def reverse(model, src):
    """Greedy decoding: the encoder runs once, the decoder once per token."""
    model.eval()
    enc = model.encoder(src)
    buf = jnp.full((len(src), T + 1), BOS)
    for t in range(T):
        buf = buf.at[:, t + 1].set(next_token(model, enc, buf, t))
    return buf[:, 1:]

src, _, tgt = sample_batch(1000, d2l.get_key())
pred = reverse(seq2seq, src)
print(f'exact match on 1000 fresh strings: '
      f'{(pred == tgt).all(-1).mean():.3f}')
print(f'source {to_str(src[0])!r} -> predicted {to_str(pred[0])!r}')
```

```
exact match on 1000 fresh strings: 1.000
source 'mmodhakcikni' -> predicted 'inkickahdomm'
```

11.4.3.4 クロスアテンションのアライメントの読み取り

文字列反転では、ターゲットのステップ t はソースのトークン $T-1-t$ を出力する。したがって、クロスアテンションの重み行列は反対角線（anti-diagonal）$(t, T-1-t)$ に集中しなければならないのだろうか。そこで、1 つのエンコーダブロックと 1 つのデコーダブロックを持つ訓

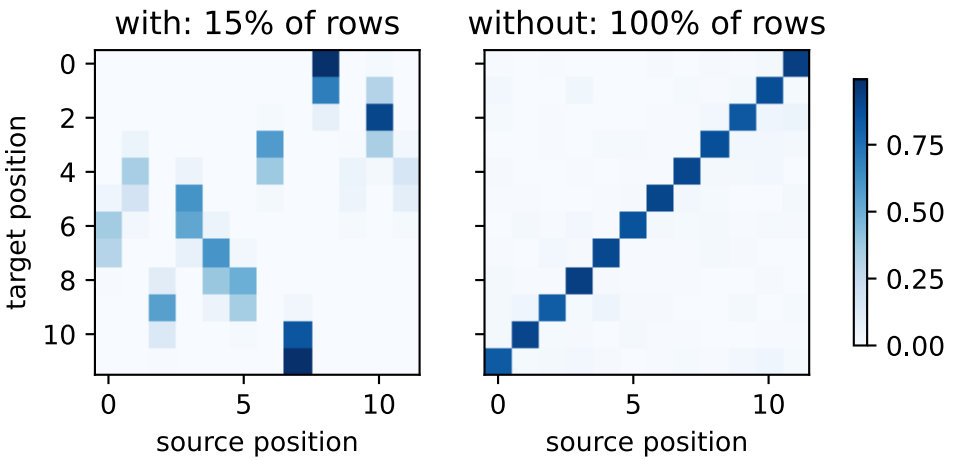
練済みの `EncoderDecoder`（エンコーダ自己アテンションあり：`with`）を、エンコーダがブロックを持たず埋め込みのみからなるそれ以外は同一のモデル（なし：`without`、したがってエンコーダ位置 j はソーストークン j の情報しか保持できない）と比較してみよう。

```
    def reversal_model(self_attention=True):
        model = EncoderDecoder(V + 1, rngs=nnx.Rngs(d2l.get_key()))
        if not self_attention:  # the encoder keeps embeddings and positions
            model.encoder = TransformerEncoder(V + 1, num_blks=0,
                                               rngs=nnx.Rngs(d2l.get_key()))
        train_reversal(model)
        return model

    def alignment(model):
        """Exact match on 1000 fresh strings; the head-averaged cross-attention
        of 256 strings, and how often its largest weight in a row falls on the
        true source position."""
        src, _, tgt = sample_batch(1000, d2l.get_key())
        match = (reverse(model, src) == tgt).all(-1).mean()
        src, dec_in, _ = sample_batch(256, d2l.get_key())
        model(src, dec_in)
        maps = model.blks[0].cross_attention.attention_weights.mean(1)
        want = jnp.arange(T - 1, -1, -1)  # row t: column T-1-t
        return match, (maps.argmax(-1) == want).mean(), maps[0]
```

エンコーダ自己アテンションあり（`with`）となし（`without`）の両方について独立した 6 つのシードを実行し、各グループで反対角線へのアライメントが最も弱いクロスアテンションのヒートマップをプロットすると、直接的な位置参照と多層ルーティングの違いが鮮明になる。

```
    encoders = {'with': [seq2seq] + [reversal_model() for _ in range(5)],
                'without': [reversal_model(False) for _ in range(6)]}
    maps, titles = [], []
    for name, models in encoders.items():
        runs = [alignment(m) for m in models]
        print(f'encoder {name} self-attention, six runs')
        print('  exact match:', ' '.join(f'{m:.3f}' for m, _, _ in runs))
        print('  rows on the true source position:',
              ' '.join(f'{100 * h:.0f}%' for _, h, _ in runs))
        _, hit, least = min(runs, key=lambda run: run[1])  # the least aligned
        maps.append(least)
        titles.append(f'{name}: {100 * hit:.0f}% of rows')
    d2l.show_heatmaps(d2l.stack(maps)[None], xlabel='source position',
                      ylabel='target position', titles=titles, figsize=(6, 3),
                      cmap='Blues')
```

```
encoder with self-attention, six runs
  exact match: 0.999 1.000 1.000 1.000 1.000 0.999
  rows on the true source position: 15% 73% 31% 100% 90% 88%
encoder without self-attention, six runs
  exact match: 1.000 1.000 1.000 1.000 1.000 1.000
  rows on the true source position: 100% 100% 100% 100% 100% 100%
```



エンコーダに自己アテンションがない場合(without)、クロスアテンションは反対角線 $(t, T-1-t)$ を直接参照してソーストークン $T-1-t$ を取り出さ**なければならない**。一方、エンコーダが双方向自己アテンションを含む場合（with)、クロスアテンションが $\mathbf{H}_{\text{enc}}$ を読み取る前に、エンコーダ層がすでに位置間でソース情報をルーティングまたは再編成できる。これは、多層ネットワークにおける生のアテンション重みがトークン対トークンの直接的な帰属マップ（attribution map）ではなく層間ルーティングを反映しているという古典的な教訓を改めて示している (Jain and Wallace 2019)。

11.4.4 インターフェースとしてのクロスアテンション：固定サイズの潜在ボトルネック

クロスアテンションで必要なのは、クエリ $\mathbf{Q} \in \mathbb{R}^{B\times M\times d}$ とキー・値 $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{B\times N\times d}$ が同じ特徴幅 d を共有することだけであるため、内部の処理長 M を生の入力長 N から切り離すことができる（Figure 11.10)。

Perceiver (Jaegle et al. 2021)（およびマルチモーダルな視覚・言語リサンプラ (Alayrac et al. 2022)）では、学習された $M \ll N$ 個の潜在クエリベクトルの配列 $\mathbf{Z}_0 \in \mathbb{R}^{M\times d}$ が、長さ N トークンの入力 $\mathbf{X} \in \mathbb{R}^{B\times N\times d}$ に対して $O(MNd)$ 回の演算でクロスアテンションを向け、その後に L 個の自己アテンションブロックが長さ M のコンパクトな潜在配列を $O(LM^2d)$ 回の演算で（$O(LN^2d)$ ではなく）処理する。

```
class PerceiverEncoder(nnx.Module):
    """M learned latents cross-attend into the input, then process among
    themselves."""
    def __init__(self, num_latents, num_hiddens, num_heads=4, num_blks=2,
                 *, rngs):
        self.latents = nnx.Param(
            0.02 * jax.random.normal(rngs.params(),
                                     (num_latents, num_hiddens)))
        self.norm_q = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)
        self.cross_attention = d2l.MultiHeadAttention(num_hiddens, num_heads,
                                                      rngs=rngs)
        self.blks = nnx.List([
            d2l.TransformerBlock(num_hiddens, num_heads, rngs=rngs)
```

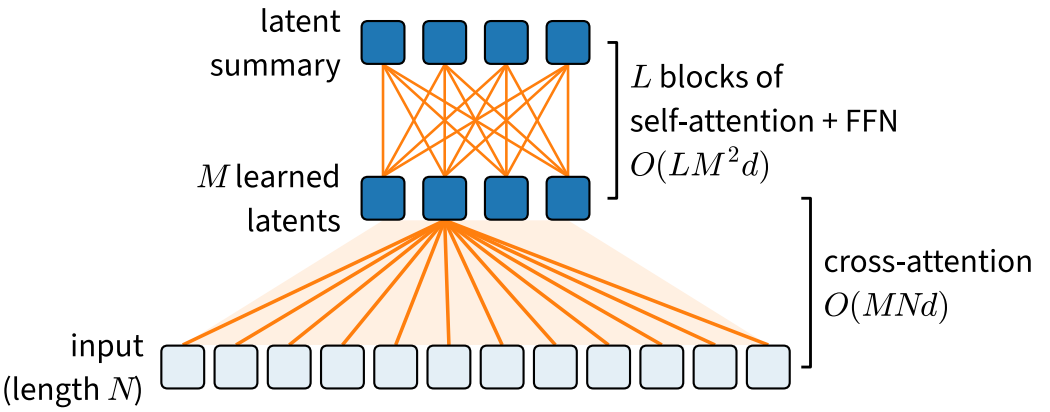


Figure 11.10: 入力長 N、学習された M 個の潜在変数、幅 d、および L 個の潜在ブロックからなる潜在ボトルネック。すべての潜在変数がすべての入力を読み取る（網掛けの帯。1 つの潜在変数による読み取りのみすべての線を描画している）。1 回のクロスアテンションによる読み取りの混合コストは $O(MNd)$ であり、後続の各潜在自己アテンション層の混合コストは $O(M^2d)$、スタック全体では $O(LM^2d)$ となる。これらの項ではバッチサイズ、ヘッド数、幅を固定とし、射影と FFN は省略している。

```
            for _ in range(num_blks)])
        self.norm = nnx.RMSNorm(num_hiddens, epsilon=1e-5, rngs=rngs)

    def __call__(self, X):
        Z = jnp.broadcast_to(self.latents[...],
                             (X.shape[0],) + self.latents.shape)
        Z = Z + self.cross_attention(self.norm_q(Z), X, X)  # O(MN)
        for blk in self.blks:
            Z = blk(Z)                                      # O(M^2)
        return self.norm(Z)
```

11.4.5 3 つの配線の使い分け

- **エンコーダのみ（`is_causal=False`）**は、1 回の順伝播ですべてのトークンに完全な双方向文脈を与えるため、密なテキスト埋め込み、リランキング、トークン分類、および非自己回帰型の視覚バックボーン（11.5 節）における標準的なバックボーンとなっている。
- **デコーダのみ（`is_causal=True`）**は、次トークン予測によって系列トークンの 100%（MLM の 15% に対して）で訓練でき、プロンプトと生成される続きを単一の統合されたパラメータスタックおよび KV キャッシュ（11.3 節）で処理できるため、汎用言語モデルの主流アーキテクチャとなっている。
- **エンコーダ・デコーダ（クロスアテンション付き）**は、Whisper (Radford et al. 2023) における音声スペクトログラムからテキストへの変換や、固定サイズの潜在クロスアテンション・インターフェースを通じて圧縮される高解像度の視覚・センサ配列（Figure 11.10）のように、入力と出力のモダリティや長さが異なるときに真価を発揮する。

11.4.6 まとめ

同じ TransformerBlock を、**双方向エンコーダ**（is_causal=False）、**因果デコーダ**（is_causal=True）、または**クロスアテンション付きエンコーダ・デコーダ**（Figure 11.8）として配線できる。マスクなしの自己アテンションでは次トークン予測のもとですべての位置が自身のターゲットを読み取れてしまうため、双方向エンコーダは**マスク言語モデリング（MLM）**（Equation 11.10）で訓練され、ウィンドウ全体にわたって対称な U 字型の loss_by_position 曲線を生じる。**エンコーダ・デコーダ**では、各 DecoderBlock がターゲットの因果自己アテンション、**クロスアテンション**（デコーダのクエリがエンコーダのキーと値にアテンションを向ける）、および位置ごとの FFN を交互に配置するため、エンコーダはソースに対して 1 回だけ実行され、デコーダが自己回帰的にステップを進める。より一般には、学習された $M \ll N$ 個のクエリベクトルによるクロスアテンション（PerceiverEncoder）を用いることで、長さ N トークンの入力を長さ M トークンの潜在ボトルネックへと $O(MNd + LM^2d)$ 時間で圧縮できる（Figure 11.10）。

11.4.7 演習問題

1. **BERT の 80 / 10 / 10 マスキングレシピ**。mask_tokens は選択されたすべての位置（バイトの 15%）を <mask> に置き換えている。<mask> はファインチューニング時や推論時には決して現れないため、Devlin et al. (2018) は選択されたトークンを 80% の割合で <mask> に、10% の割合で語彙から一様ランダムに選んだトークンに置き換え、残りの 10% の割合では元のトークンをそのまま残している。この 80 / 10 / 10 ルールを mask_tokens に実装し、10% をそのまま残し 10% をランダムにすることで、なぜエンコーダが**マスクされていない**トークンでも <mask> トークンと同様に正確な表現を維持せざるを得なくなるのか説明せよ。

2. **不揃いな系列対系列バッチに対するパディングマスク**。バッチ内に長さ $(s_1, \dots, s_B) \le S$ の可変長ソース文字列と長さ $(t_1, \dots, t_B) \le T_{\text{dec}}$ の可変長ターゲット文字列が含まれる場合、3 つのブールマスクが必要になる。すなわち、(a) $(B, 1, S, S)$ のエンコーダ用パディングマスク、(b) $(B, 1, T_{\text{dec}}, T_{\text{dec}})$ のデコーダ用因果＋パディングマスク、(c) $(B, 1, T_{\text{dec}}, S)$ の長方形のクロスアテンション用ソースパディングマスクである。これら 3 つのマスクをすべて構築するヘルパ関数 make_seq2seq_masks(src_len, tgt_len, S, T_dec) を作成し、パディングトークンを src に付加して EncoderDecoder に渡しても、パディング以外の位置におけるデコーダのロジットが変化しないことを検証せよ。

3. **MLM のマスク率 p の変化**。MaskedLM を 2,000 ステップにわたりマスク率 $p \in \{0.05, 0.15, 0.30, 0.50\}$ でそれぞれ訓練し、4 つのモデルすべてを同一の $p = 0.15$ の検証セット masked.val_dataloader() で評価せよ。p は、バッチあたりの教師あり勾配信号の数と各マスクの周囲で利用可能な破損していない文脈の量とをどのようにトレードオフしているか。

4. **因果モデルと双方向モデルにおける時間の順方向と逆方向**。因果言語モデルはテキストを左から右（$x_1 \to x_n$）へと処理する。WikiText103 のすべてのバイトウィンドウを反転させて因果モデルが右から左（$x_n \to x_1$）へと予測するようにして 1,000 ステップ訓練し、バイトあたりの平均検証損失と loss_by_position 曲線を、左から右への因果モデルおよび双方向の MaskedLM と比較せよ。同時対数尤度の合計 $\sum_{t=1}^{n} \log P(x_t \mid \text{past})$ が理論上は時間反転に対して不変であるのはなぜか。また、双方向モデルは両方の片方向モデルに対してどこで優位性を得るのか。

5. **内容ベース対位置ベースのクロスアテンション・アライメント**。本節の文字列反転実験では、ランダム文字列内のどの文字も重複し得るため、重複文字を区別するには位置情報が必要である。長さ 12 の各文字列が 12 個の**相異なる**文字のランダムな順列になるよう sample_batch を修正し、エンコーダの順伝播から位置項を取り除いた状態で EncoderDecoder

を訓練せよ。位置埋め込みを持たない 1 層のエンコーダが、12 文字すべてが相異なる場合でさえ文字列反転に失敗する一方で、ソート課題（入力文字をアルファベット順 `a..p` に出力する課題）ではエンコーダの位置埋め込みが一切なくても成功するのはなぜか。

6. **系列対系列の反転のための Perceiver IO**。Perceiver IO (Jaegle et al. 2022) は、`PerceiverEncoder`（学習された M 個の潜在クエリが長さ N トークンの入力にアテンションを向ける）と、クロスアテンション**デコーダ**（N_{out} 個のターゲット位置埋め込み $\mathbf{Q}_{\text{out}} \in \mathbb{R}^{N_{\text{out}} \times d}$ が M 個の潜在ベクトル $\mathbf{Z}^{(L)} \in \mathbb{R}^{B \times M \times d}$ にクロスアテンションを向ける）とを組み合わせる。12 文字の文字列を $M \in \{2, 4, 8, 12\}$ 個の潜在変数（$d = 128$）にエンコードし、1 回の並列なクロスアテンション読み出しパスで反転文字列をデコードする `PerceiverAutoencoder` を構築せよ。ボトルネックのサイズ M に応じて反転の正解率はどのように変化するか。

11.5 Vision Transformer（ViT）

畳み込みニューラルネットワーク（第 6 章、第 7 章）は、画像に関する 2 つの幾何学的な事前仮定をアーキテクチャに直接組み込んでいる。すなわち、**空間的局所性**（spatial locality、$k \times k$ カーネルが隣接するピクセルのみを結合する性質）と**平行移動同変性**（translation equivariance、同じカーネルの重みがすべての (r, c) 位置にわたってスライドする性質）である。では、2 次元グリッド上の近傍やシフト同変性という概念を**まったく**組み込んでいない標準的な双方向 Transformer エンコーダ（11.4 節）で、競争力のある画像分類ができるのだろうか。

個々のピクセルに直接自己アテンションを適用すると、計算量はピクセル数の二乗で増大する。96×96 の画像でさえ 9,216 ピクセルあり、ヘッドあたり $9216^2 \approx 8.5 \times 10^7$ 個のアテンションペアが生じる。そこで Dosovitskiy et al. (2021) は **Vision Transformer（ViT）**（Figure 11.11）を導入し、画像を重なりのない $p \times p$ の**パッチ**のグリッドへとトークン化した。96×96 の画像は、パッチサイズを $p = 16$ とすると、$m = (96/16)^2 = 36$ 個のパッチトークンからなる短い系列になる。各パッチを幅 d に射影し、学習可能な `<cls>` トークンを先頭に付加し、学習型の 1 次元位置埋め込み（position embedding）を加えたうえで、合計 37 個のトークンを L 個の双方向 Pre-Norm Transformer ブロックに通す。

11.5.1 トークンとしてのパッチ（`PatchEmbedding`）

形状 (B, C, H, W) の画像テンソルを重なりのない $p \times p$ のパッチに分割し、各パッチに共通の線形射影 $\mathbb{R}^{Cp^2} \to \mathbb{R}^d$ を適用する操作は、**カーネルサイズ** $p \times p$、**ストライド** p、**出力チャネル数** d **の 2 次元畳み込み**を適用した後に、高さ H/p、幅 W/p の 2 次元空間グリッドを $m = (H/p)(W/p)$ 個のトークンベクトルからなる系列へと平坦化する操作と同一である。

```
class PatchEmbedding(nnx.Module):
    """Image-to-sequence stem of the vision transformer."""
    def __init__(self, img_size=96, patch_size=16, num_hiddens=512,
                 num_channels=3, *, rngs):
        # A partial trailing patch would silently shrink the token grid
        assert img_size % patch_size == 0, 'patches must tile the image'
        self.num_patches = (img_size // patch_size) ** 2
        # PyTorch's default initialization: U(-a, a), a = 1/sqrt(fan_in)
        a = (num_channels * patch_size ** 2) ** -0.5
        init = lambda key, shape, dtype=jnp.float32: jax.random.uniform(
            key, shape, dtype, -a, a)
        self.conv = nnx.Conv(num_channels, num_hiddens,
                             kernel_size=(patch_size, patch_size),
```

```
                            strides=patch_size, padding='VALID',
                            kernel_init=init, bias_init=init, rngs=rngs)

    def __call__(self, X):  # (batch, h, w, c) -> (batch, patches, d)
        X = self.conv(X)
        return X.reshape(X.shape[0], -1, X.shape[-1])
```

4 枚の 96×96 単一チャネル画像からなるミニバッチに対し、パッチサイズ $p = 16$ で PatchEmbedding を適用すると、$(96/16)^2 = 36$ 個の幅 $d = 512$ のパッチトークンが出力される。

```
img_size, patch_size, num_hiddens, batch_size = 96, 16, 512, 4
patch_emb = PatchEmbedding(img_size, patch_size, num_hiddens, num_channels=1,
                           rngs=nnx.Rngs(d2l.get_key()))
X = jnp.zeros((batch_size, img_size, img_size, 1))
d2l.check_shape(patch_emb(X),
                (batch_size, (img_size//patch_size)**2, num_hiddens))
```

11.5.2 Vision Transformer のアーキテクチャ

11.5.2.1 エンコーダブロック（ViTBlock）

Dosovitskiy et al. (2021) に従い、ViTBlock は設定可能な d2l.TransformerBlock（11.1 節）を継承し、Pre-Norm の LayerNorm と、GELU を活性化関数とする 2 層の多層パーセプトロン（multilayer perceptron、MLP、隠れ層幅は mlp_num_hiddens）を用いる。デフォルトではアテンションの射影にバイアスを持たせない（use_bias=False）一方、MLP の 2 つの線形層と LayerNorm にはバイアスを残す。

```
class ViTBlock(d2l.TransformerBlock):
    """Pre-norm block with LayerNorm and an exact-GELU MLP."""
    def __init__(self, num_hiddens, mlp_num_hiddens, num_heads, dropout,
                 use_bias=False, *, rngs):
        super().__init__(
            num_hiddens, num_heads, dropout=dropout, norm='layer',
            bias=use_bias, rngs=rngs, ffn_factory=lambda rngs: nnx.Sequential(
                d2l.Linear(num_hiddens, mlp_num_hiddens, rngs=rngs),
                lambda X: jax.nn.gelu(X, approximate=False),  # Exact GELU
                nnx.Dropout(dropout, rngs=rngs),
                d2l.Linear(mlp_num_hiddens, num_hiddens, rngs=rngs)))
```

11.5.2.2 <cls> トークン、位置埋め込み、および ViT

特定の空間位置を優遇することなく m 個のパッチ表現を画像全体の単一ベクトルへと集約するため、ViT は学習可能なパラメータベクトル cls_token（形状 $(1, 1, d)$）をインデックス 0 の先頭に付加し、学習可能な位置埋め込みテーブル pos_embedding（形状 $(1, m + 1, d)$）を $m + 1$ 個の位置すべてに加算する。そのうえで L 層の ViTBlock を実行し、LayerNorm で正規化して、インデックス 0（X[:, 0]）を線形分類ヘッドに入力する。

```
class ViT(d2l.Classifier):
    """Vision transformer."""
```

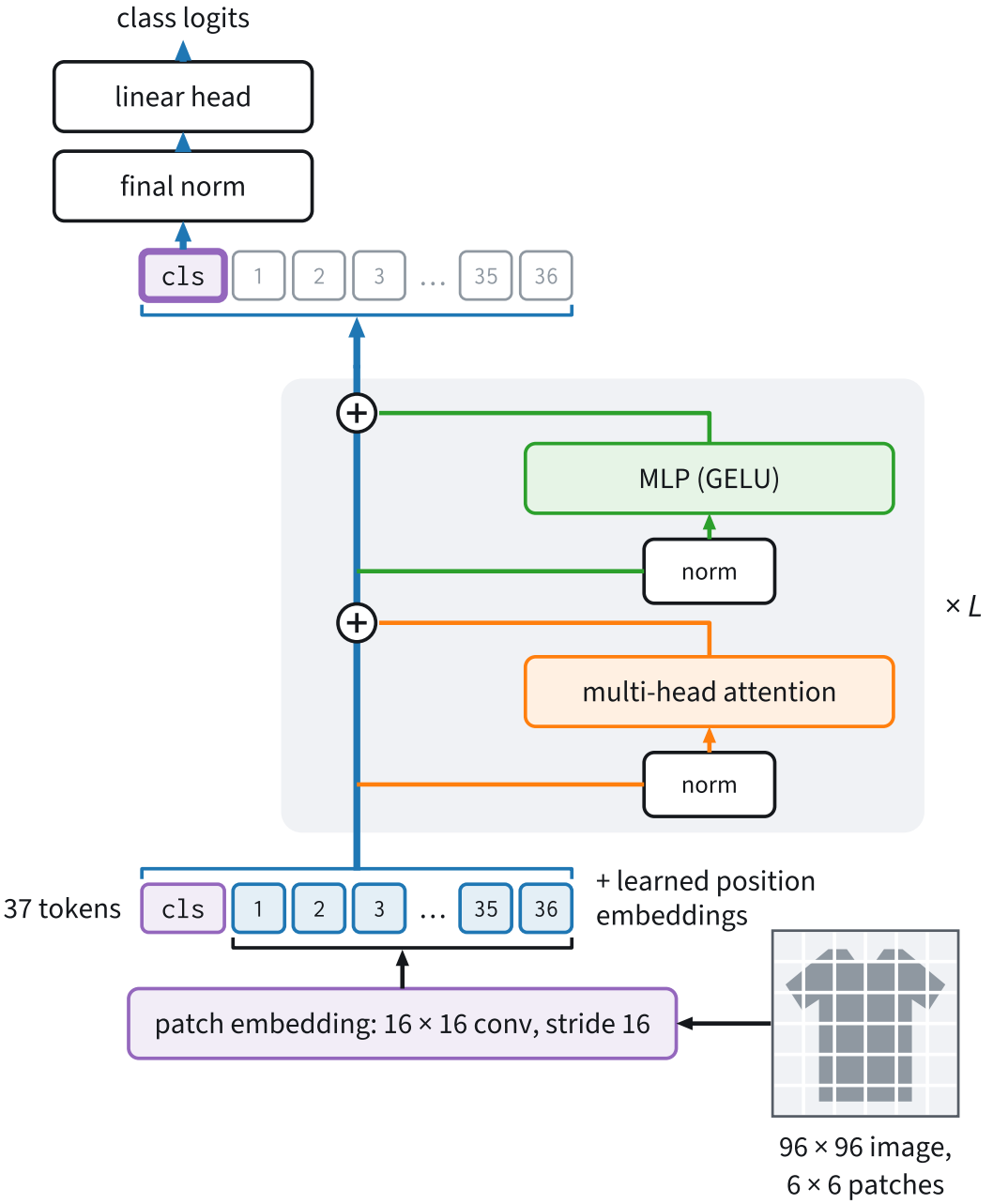


Figure 11.11: 96 × 96 画像に対する Vision Transformer。ストライド付き畳み込みによって 6 × 6 グリッドに並ぶ 16 × 16 パッチが 36 個のトークンに変換される。学習された「<cls>」トークンが先頭に付加され、学習された位置埋め込みが加算される。続いて L 個の Pre-Norm Transformer ブロックが全 37 トークンを処理し、線形ヘッドが「<cls>」トークンの最終表現を分類する。

```
    def __init__(self, *, img_size, patch_size, num_hiddens, mlp_num_hiddens,
                 num_heads, num_blks, num_steps, emb_dropout=0.0,
                 blk_dropout=0.0, lr=1e-3, use_bias=False, num_classes=10,
                 num_channels=1, rngs):
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        self.patch_embedding = PatchEmbedding(
            img_size, patch_size, num_hiddens, num_channels, rngs=rngs)
        self.cls_token = nnx.Param(jnp.zeros((1, 1, num_hiddens)))
        num_tokens = self.patch_embedding.num_patches + 1  # Add the cls token
        # Positional embeddings are learnable, initialized to small noise
        self.pos_embedding = nnx.Param(
            rngs.params.normal((1, num_tokens, num_hiddens)) * 0.02)
        self.dropout = nnx.Dropout(emb_dropout, rngs=rngs)
        self.blks = nnx.List([
            ViTBlock(num_hiddens, mlp_num_hiddens, num_heads, blk_dropout,
                     use_bias=use_bias, rngs=rngs) for _ in range(num_blks)])
        self.head = nnx.Sequential(
            nnx.LayerNorm(num_hiddens, epsilon=1e-5, rngs=rngs),
            d2l.Linear(num_hiddens, num_classes, rngs=rngs))

    def forward(self, X):
        X = self.patch_embedding(X)  # (batch, patches, d)
        cls = jnp.broadcast_to(self.cls_token[...], (len(X), 1, X.shape[2]))
        X = self.dropout(jnp.concatenate((cls, X), 1) + self.pos_embedding)
        for blk in self.blks:
            X = blk(X)
        return self.head(X[:, 0])

    def configure_optimizers(self):
        # AdamW decays weight matrices and kernels; it exempts biases, norm
        # parameters, the class token, and the position embeddings
        decay = lambda params: jax.tree_util.tree_map_with_path(
            lambda path, p: p.ndim >= 2 and path[0].key not in (
                'cls_token', 'pos_embedding'), params)
        # Linear warmup over the first tenth of the steps, then cosine
        return optax.adamw(lambda t: self.lr * d2l.cosine_warmup(
            t, self.num_steps, self.num_steps // 10), weight_decay=0.05,
            mask=decay)
```

11.5.3 Fashion-MNIST での訓練

11.5.3.1 最初の実行

2 ブロックの ViT（$d = 512, d_{\text{mlp}} = 2048, H = 8, L = 2, p = 16$、パラメータ数 ≈ 6.5M）を、96×96 の Fashion-MNIST 上で 10 エポック訓練する。最適化には線形ウォームアップとコサイン学習率減衰を伴う AdamW（$\lambda = 0.05$）を用いる。

```
data = d2l.FashionMNIST(batch_size=128, resize=(96, 96))
num_steps = 10 * len(data.train_dataloader())  # Ten epochs of minibatches
hparams = dict(img_size=96, patch_size=16, num_hiddens=512,
```

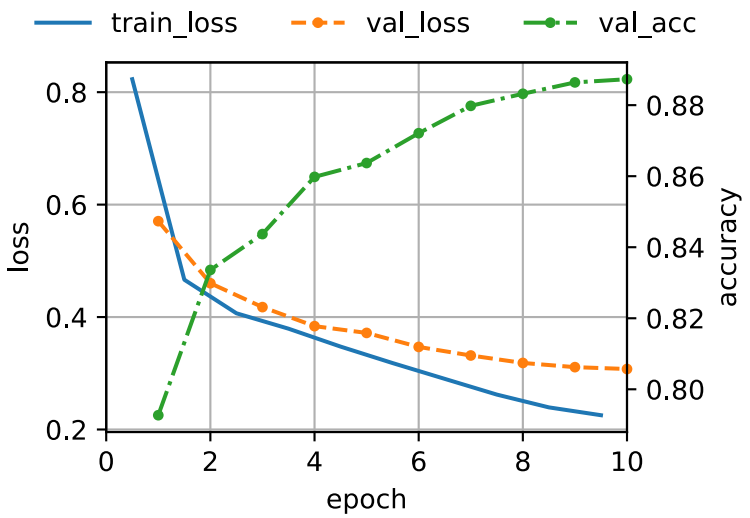
```
                mlp_num_hiddens=2048, num_heads=8, num_blks=2,
                emb_dropout=0.1, blk_dropout=0.1, num_steps=num_steps)
```

訓練を始める前に、`model = vit()` をインスタンス化し、ランダムに初期化されたパッチ位置埋め込みのコピーを `pos_init` として保存しておく。これにより、初期化時点から訓練によってどれだけの空間構造が加わったかを後で計測できる。

```
vit = lambda: ViT(**hparams, lr=3e-4, rngs=nnx.Rngs(d2l.get_key()))
model = vit()  # A fresh model at the tuned rate
pos_init = model.pos_embedding[0, 1:]  # The initial noise, a new array
```

`model` を 10 エポックにわたり `d2l.Trainer` で適合させると、検証正解率は約 89% に達する（アップサンプリングされた 96×96 の Fashion-MNIST 画像での評価）。

```
trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(model, data)
vit_history = trainer.history
vit_params, vit_acc = d2l.num_params(model), vit_history['val_acc'][-1]
print(f'ViT: {vit_params/1e6:.1f}M parameters, '
      f'validation accuracy {vit_acc:.2f}')
```




```
ViT: 6.5M parameters, validation accuracy 0.89
```

11.5.3.2 空間位置の役割と学習された 2 次元グリッドのトポロジー

`pos_embedding` がなければ、`ViT` は $m = 36$ 個のパッチに対して**置換不変**（permutation-invariant）である。36 個のパッチをどのように置換しても、すべてのアテンション行列の $1 \ldots 36$ 行目が同じように置換されるだけで、0 行目（`<cls>`）は文字通りまったく変化しないからである。学習された `pos_embedding` がある場合、パッチ k は位置ベクトル $\mathbf{p}_k$ を配置先のスロット k に応じて受け取る。訓練済みの `ViT` が空間配置にどれだけ依存しているかを検証するため、テスト時に各検証画像の 36 個のパッチを 5 通りのランダムな順列で並べ替える（`shuffled_accuracy`）。

```
def shuffle_patches(X, perm, p=16):
    """Move the p x p blocks of images X: block perm[k] goes to grid cell k."""
    n, h, w, c = X.shape
    B = X.reshape(n, h // p, p, w // p, p, c).swapaxes(2, 3)  # Grid of blocks
    B = B.reshape(n, -1, p, p, c)[:, perm]  # Blocks in raster order, moved
    B = B.reshape(n, h // p, w // p, p, p, c).swapaxes(2, 3)
    return B.reshape(X.shape)

def shuffled_accuracy(model, data, perm):
    """Validation accuracy on images whose blocks are moved by perm."""
    return d2l.evaluate_accuracy(model, ((shuffle_patches(X, perm), y)
                                         for X, y in data.val_dataloader()))
```

shuffled_accuracy を 36 個のパッチに対する 5 通りの固定されたランダム順列にわたって評価し、テスト時に空間配置をかき混ぜたときに正解率がどれだけ急激に低下するかを計測する。

```
rng = np.random.default_rng(0)
perms = [rng.permutation(36) for _ in range(5)]  # Five fixed shuffles
accs = [shuffled_accuracy(model, data, perm) for perm in perms]
print(f'trained ViT: {vit_acc:.3f} as is, '
      f'{min(accs):.2f} to {max(accs):.2f} with its patches shuffled')
```

```
trained ViT: 0.887 as is, 0.36 to 0.45 with its patches shuffled
```

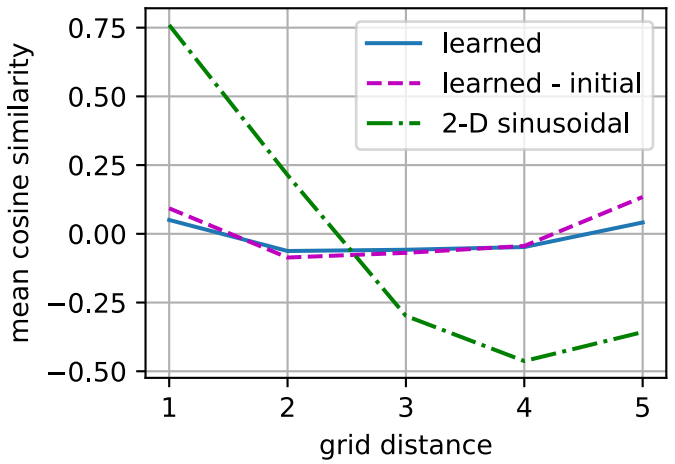
6×6 のパッチ配置をかき混ぜると、検証正解率は約 0.89 からシャッフルの仕方に応じて約 0.25〜0.45 まで低下するため、訓練済みの ViT は各パッチが置かれた位置に強く依存していることがわかる。それでもランダム予測の 0.1 を大きく上回っているのは、靴のかかとや襟の形状といった局所的な 16×16 パッチのテクスチャだけでも、「パッチの集合（bag of patches）」として大まかな衣類クラスの多くを識別できるからである。

構造を持たない 1 次元の位置テーブル pos_embedding（$36 \times d$）は、6×6 の画像グリッドが持つ 2 次元の幾何構造を実際に発見しているのだろうか。パッチの位置ベクトル間の平均コサイン類似度を、2 次元グリッド距離 $\max(|r - r'|, |c - c'|)$（6×6 グリッド上）の関数として計算すれば（grid_similarity）、この問いに答えられる。比較のため、構造上グリッドを最初からエンコードしている 2 次元正弦波符号と、そうではないランダムなベクトルについても同じ曲線を計算する。

```
def grid_similarity(P):
    """Mean cosine similarity of 36 position embeddings, by grid distance."""
    P = P - P.mean(0)                    # Drop the part shared by all
    P = P / jnp.linalg.norm(P, axis=1, keepdims=True)
    sim = P @ P.T                        # (36, 36) cosine similarities
    rows, cols = jnp.meshgrid(jnp.arange(6), jnp.arange(6), indexing='ij')
    ij = jnp.stack((rows.flatten(), cols.flatten()), 1)  # Grid cells
    dist = jnp.abs(ij[:, None] - ij[None]).max(-1)  # Max of the two offsets
    return [float(sim[dist == k].mean()) for k in range(1, 6)]

S = d2l.sinusoidal_encoding(6, 256)      # 1-D codes of the 6 rows/columns
rows, cols = jnp.meshgrid(jnp.arange(6), jnp.arange(6), indexing='ij')
fixed = jnp.concatenate((S[rows.flatten()], S[cols.flatten()]), 1)  # (36, 512)
learned = model.pos_embedding[0, 1:]     # Drop the cls slot
```

```
noise = jax.random.normal(d2l.get_key(), (36, 512))
curves = {'learned': grid_similarity(learned),
          'learned - initial': grid_similarity(learned - pos_init),
          '2-D sinusoidal': grid_similarity(fixed),
          'random': grid_similarity(noise)}
for name, curve in curves.items():
    print(f'{name:>17}:', '  '.join(f'{c:+.2f}' for c in curve))
d2l.plot(list(range(1, 6)), list(curves.values())[:3], 'grid distance',
         'mean cosine similarity', legend=list(curves)[:3])
```

```
          learned: +0.05  -0.06  -0.06  -0.05  +0.04
learned - initial: +0.09  -0.09  -0.07  -0.04  +0.13
   2-D sinusoidal: +0.76  +0.21  -0.30  -0.46  -0.36
           random: -0.02  -0.03  -0.03  -0.03  -0.02
```



10 エポック後の答えは「ごくわずか」である。隣接するパッチ（距離 1）と反対側の端にあるパッチ（距離 5）は、ランダムなベクトルよりもわずかに類似度が高くなるが（約 $+0.05$ 対 -0.03）、その中間の距離では学習された曲線は 0 をわずかに下回り、初期ノイズに訓練が付け加えた変化量は距離 5 で最大となる。対照的に、2 次元正弦波符号は距離 1 での約 0.75 から 0 を大きく下回る値まで低下する。Dosovitskiy et al. (2021) が ViT の位置埋め込みに見出した明瞭な 2 次元構造は、はるかに大規模なデータセットではるかに長く訓練することで初めて現れるものである。

11.5.4 パラメータ数を揃えた畳み込みネットワークとの比較

この 6.5M パラメータの `ViT` は、60,000 枚の画像からなるこのデータセットにおいて、パラメータ数を揃えた ResNet 風の畳み込みベースライン（`CompactResNet`、パラメータ数 ≈ 6.5M）をまったく同じコサイン減衰付きの AdamW スケジュールで 10 エポック訓練した場合と比べるとどうなるだろうか。

```
class CompactResNet(d2l.Classifier):
    """A ResNet-style CNN parameter-matched to the ViT above."""
    configure_optimizers = ViT.configure_optimizers  # The same recipe

    def __init__(self, num_steps, lr=1e-3, num_classes=10, *, rngs):
```

```
        super().__init__()
        self.save_hyperparameters(ignore=['rngs'])
        he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
        self.net = nnx.Sequential(
            # Pad 3 on every side: with stride 2 on even sizes, 'same' pads
            # one more row and column at the bottom and right
            nnx.Conv(1, 64, kernel_size=(7, 7), strides=(2, 2), padding=3,
                     use_bias=False, kernel_init=he, rngs=rngs),
            nnx.BatchNorm(64, momentum=0.9, use_fast_variance=False,
                          rngs=rngs), nnx.relu,
            lambda x: nnx.max_pool(x, window_shape=(3, 3), strides=(2, 2),
                                   padding=((1, 1), (1, 1))),
            d2l.Residual(64, in_channels=64, rngs=rngs),
            d2l.Residual(64, in_channels=64, rngs=rngs),
            d2l.Residual(128, strides=(2, 2), in_channels=64, rngs=rngs),
            d2l.Residual(128, in_channels=128, rngs=rngs),
            d2l.Residual(256, strides=(2, 2), in_channels=128, rngs=rngs),
            d2l.Residual(256, in_channels=256, rngs=rngs),
            d2l.Residual(512, strides=(2, 2), in_channels=256, rngs=rngs),
            lambda x: x.mean(axis=(1, 2)),
            nnx.Linear(512, num_classes, kernel_init=he, rngs=rngs))

    def forward(self, X):
        return self.net(X)
```

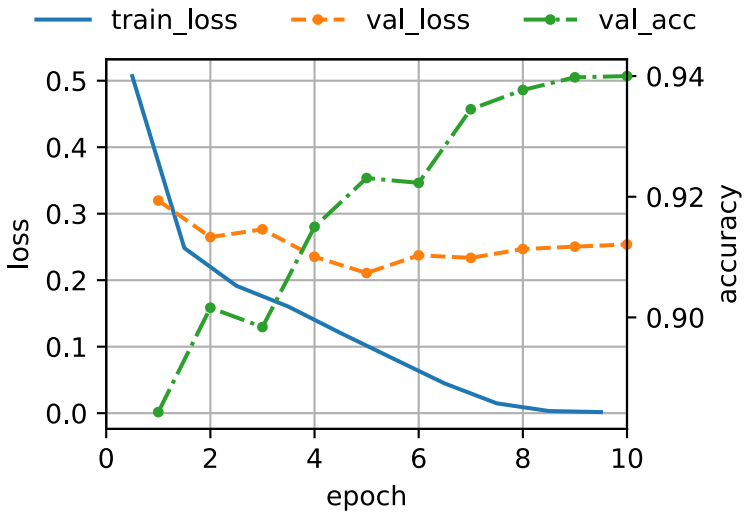
ファクトリ compact_cnn を作成し、cnn をピーク学習率 10^{-3} でインスタンス化する。

```
compact_cnn = lambda: CompactResNet(num_steps, lr=1e-3,
                                    rngs=nnx.Rngs(d2l.get_key()))
cnn = compact_cnn()  # A fresh CNN
```

CompactResNet を 10 エポック訓練し（さらに実行間の安定性を確認するため CompactResNet と ViT の両方で 2 つ目のシードでも訓練し）、まったく同じ 6.5M パラメータの予算で 2 つのアーキテクチャを比較する。

```
def final_accuracy(model, data):
    """Validation accuracy after the ten epochs of the first run."""
    trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
    trainer.board.display = False  # No progress plot for the repeats
    trainer.fit(model, data)
    return trainer.history['val_acc'][-1]

trainer = d2l.Trainer(max_epochs=10, num_gpus=1)
trainer.fit(cnn, data)
cnn_history = trainer.history
cnn_params, cnn_acc = d2l.num_params(cnn), cnn_history['val_acc'][-1]
cnn_runs = [cnn_acc, final_accuracy(compact_cnn(), data)]  # Run-to-run range
learned_runs = [vit_acc, final_accuracy(vit(), data)]  # A second ViT run
print(f'CNN: {cnn_params/1e6:.1f}M parameters, validation accuracy',
      ', '.join(f'{a:.3f}' for a in cnn_runs))
print(f'ViT: {vit_params/1e6:.1f}M parameters, validation accuracy',
```

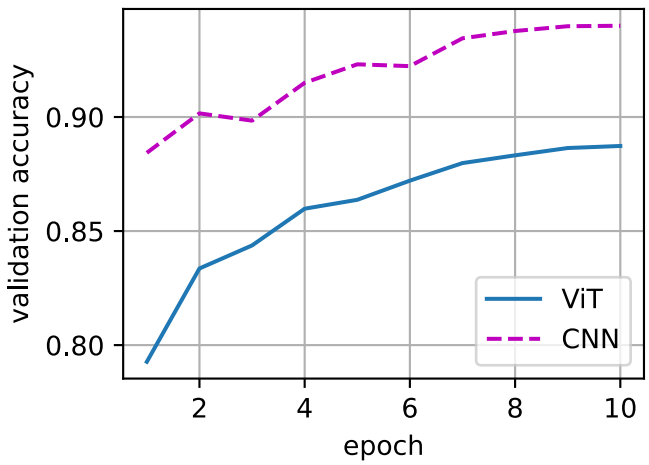
```
', '.join(f'{a:.3f}' for a in learned_runs))
```




```
CNN: 6.5M parameters, validation accuracy 0.940, 0.938
ViT: 6.5M parameters, validation accuracy 0.887, 0.892
```

10 エポックにわたる ViT と CompactResNet の検証正解率の軌跡を重ねてプロットすると、訓練全体を通じたサンプル効率の差が浮き彫りになる。

```
d2l.plot(list(range(1, 11)), [vit_history['val_acc'], cnn_history['val_acc']],
         'epoch', 'validation accuracy', legend=['ViT', 'CNN'])
```



データ拡張や大規模な事前学習なしの 60,000 枚の小さな画像データセットでは、CompactResNet の方が速く収束し、最終的に ViT を約 5 パーセントポイント上回る。この結果は、Dosovitskiy et al. (2021) が明らかにした根本的なトレードオフを示している。すなわち、CNN にハードコードされた 2 次元の局所性と平行移動同変性は小規模データ領域において強力な**帰納バイアス**

（inductive bias）を与える一方、Vision Transformer は空間的な関係をデータから柔軟に学習するため、データ拡張なしの小さなデータセットでは CNN に後れを取るものの、数千万枚の画像で事前学習した場合（あるいは強力なデータ拡張、蒸留 (Touvron, Cord, Douze, et al. 2021)、マスク付きパッチ再構成 (He et al. 2022) を用いて訓練した場合）には CNN に匹敵するか、それを凌駕する。

11.5.5 まとめ

Vision Transformer（ViT）は、形状 (C, H, W) の画像を $m = (H/p)(W/p)$ 個のトークンからなる系列へとストライド p の畳み込み（`PatchEmbedding`）によって変換し、先頭に `<cls>` トークンを付加し、学習型の 1 次元位置埋め込みを加算したうえで、双方向の Pre-Norm Transformer エンコーダを適用する（Figure 11.11）。位置埋め込みがなければ、ViT はパッチに対して厳密に置換不変である。一方、学習型の 1 次元位置埋め込みを用いると、Fashion-MNIST 上での 10 エポックの訓練では位置テーブルに 2 次元グリッドがわずかに刻まれるにすぎないものの、訓練済みモデルはパッチ配置に強く依存するようになる。ViT は畳み込みのような組み込みの 2 次元の局所性や平行移動同変性を持たないため、データ拡張なしの小さなデータセットでゼロから訓練するとパラメータ数を揃えた ResNet の方が高い性能を示すが、大規模な事前学習スケールでは ViT が CNN を追い抜き、さらにマルチモーダルモデルに対する統一されたトークンインターフェースを提供する。

11.5.6 演習問題

1. **パッチサイズ p、パラメータ数、およびアテンションの FLOPs**。本節の 96×96 画像向け `ViT` では、パッチサイズは $p = 16$（$m = 36$ パッチ）である。
   1. `PatchEmbedding` のパラメータ数、`pos_embedding` のパラメータ数、および L 層の `ViTBlock` のパラメータ数は、パッチサイズを $p = 8$（$m = 144$）に半減させたとき、あるいは $p = 4$（$m = 576$）に 4 分の 1 にしたとき、それぞれどのように変化するか。
   2. 層あたりのアテンションスコア行列のサイズ $(m+1)^2$ は、$p = 16$ から $p = 8$ および $p = 4$ にすると何倍に増大するか。$p = 8$ の `ViT` を 10 エポック訓練し、その実経過時間と検証正解率を $p = 16$ の場合と比較せよ。
2. **`<cls>` 読み出し対グローバル平均プーリング**。`<cls>` トークンによる `ViT` の読み出しを、$m = 36$ 個のパッチトークンにわたるグローバル平均プーリングに置き換えよ（`self.head(X[:, 1:].mean(1))` とするか、`cls_token` を完全に削除して全 36 個のトークンを平均する）。Fashion-MNIST 上で 10 エポック訓練し、検証正解率を `<cls>` 読み出しと比較せよ。
3. **位置埋め込みなしでの厳密な置換不変性**。`model.pos_embedding` をすべてゼロに設定し、`model` を評価モードに切り替えよ（ドロップアウトを無効化する）。次の性質を代数的に証明するとともに、検証バッチ上で誤差 10^{-6} 以内の精度で数値的に確認せよ。すなわち、`shuffle_patches(X, perm)` を任意の順列 `perm`（$\{0, \dots, 35\}$ 上の置換）に対して適用しても、出力ロジット `model(X)` はまったく変化しない。
4. **2 次元グリッド発見に関する統計的置換検定**。Δ を、隣接パッチ（グリッド距離 1）とグリッド距離 ≥ 4 のパッチとの間での `grid_similarity` の差（訓練済みの `model.pos_embedding` に対する値）とする。Δ の帰無分布を、`pos_embedding` の 36 行に対する 1,000 通りのランダムな順列にわたって計算し、経験 p 値を報告せよ。学習された位置埋め込みの微弱なグリッド構造は、偶然と区別できるか。
5. **サブパッチ単位対パッチ単位の平行移動感度**。CNN では、入力画像を s ピクセル平行移動させると、内部の特徴マップも（プーリングのストライドを除けば）s ピクセル平行移動する。一方、`ViT`（$p = 16$）では、画像を $s = 16$ ピクセル（1 パッチ幅ちょうど）平行移動

させても各パッチが隣接するパッチスロットへ移るだけであるのに対し、$s = 8$ ピクセル（半パッチ分）平行移動させるとすべての 16×16 パッチ境界を横切ることになる。訓練済みの `ViT` と `CompactResNet` の検証正解率を、右へ $s \in \{0, 2, 4, 8, 12, 16\}$ ピクセル平行移動させた検証画像（左端はゼロパディングする）上で評価せよ。なぜ `ViT` は $s = 8$ において $s = 16$ と比べ非単調な落ち込みを示すのか。

6. **深さにわたる平均アテンション距離**。@Dosovitskiy.Beyer.Kolesnikov.ea.2021 は、各ヘッドの**平均アテンション距離**を、アテンション重みで加重平均した 2 次元ユークリッドピクセル距離 $\sum_{i,j=1}^{36} \alpha_{ij}\, d_{\text{pixel}}(i, j)/36$（クエリパッチ i とキーパッチ j の間の距離）として定義している。訓練済みの `ViT` のブロック 1 とブロック 2 に含まれる各 8 個のヘッドについて、平均アテンション距離を計算せよ。CNN で受容野が拡大していくのと同様に、浅い層のヘッドは深い層のヘッドよりも局所的にアテンションを向けているか。

11.6 スケーリング則

Transformer 言語モデル（11.2 節）を事前学習（pretraining）する際、計算コストを決定する巨視的な量が 2 つある。**非埋め込みパラメータ** N（non-embedding parameters、L 個の Transformer ブロック内のパラメータ）の個数と、オプティマイザに流し込まれる**訓練トークン数** D（training tokens）である。各トークンはパラメータあたり予測可能な回数の浮動小数点演算（floating-point operations、FLOPs）を必要とするため、訓練の計算予算 C（単位は FLOPs）を固定すると、モデルサイズ N とデータ予算 D の間に直接的なトレードオフが生じる。より大きなモデルをより少ないステップ数で訓練すべきか、それともより小さなモデルをより多くのトークンで訓練すべきか。

経験的な**ニューラルスケーリング則**（neural scaling laws）の研究 (Kaplan et al. 2020) により、検証交差エントロピー損失は滑らかなべき乗則（power law）の曲面 $\mathcal{L}(N, D)$ に従い、それが N と D の何桁にもわたる範囲で成り立つことが明らかになった。続いて Hoffmann et al. (2022) は、固定された事前学習計算量 $C \approx 6ND$ のもとで損失を最小化するには、モデルパラメータ $N^*(C)$ と訓練トークン数 $D^*(C)$ を**同じ比率**（$N^* \propto C^{1/2}, D^* \propto C^{1/2}$、パラメータあたり約 $D^*/N^* \approx 20$ トークン、Figure 11.12）でスケーリングする必要があることを示した。本節では、パラメータ数の集計式 $N \approx 12Ld^2$ と FLOP 則 $C \approx 6ND$ を導出して厳密な演算子カウンタに対して両者を検証し、15 モデルのスケーリンググリッド（5 種類のサイズと 3 種類のトークン予算の組み合わせ）を WikiText-103 上で実行したうえで、**推論コスト**（inference cost）を考慮すると現代のオープンモデルが Chinchilla の比率をはるかに超えてより小さなモデルを訓練する方向へシフトする理由を説明する。

11.6.1 パラメータ数（$N \approx 12Ld^2$）と FLOPs（$C \approx 6ND$）の算出

11.6.1.1 ブロックと埋め込み

密な L ブロックの `GPT` において、隠れ層の幅を d とし、標準的なマルチヘッドアテンション（$H_{\text{kv}} = H$）および `SwiGLU` の幅 $d_{\text{ff}} = \text{round}(\frac{8}{3}d)$（11.1 節）を用いる場合を考える。

- 各アテンションサブ層は 4 つの $d \times d$ 射影行列（$\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o$）を持ち、パラメータ数は $4d^2$ 個である。
- 各 `SwiGLU` FFN は 3 つの行列（$\mathbf{W}_g, \mathbf{W}_1 \in \mathbb{R}^{d \times \frac{8}{3}d}$ および $\mathbf{W}_2 \in \mathbb{R}^{\frac{8}{3}d \times d}$）を持ち、パラメータ数は $8d^2$ 個である。
- したがって、各ブロックは $12d^2$ 個の行列パラメータ（および $2d$ 個の `RMSNorm` ゲイン）を持ち、L 個のブロック全体では次式の個数の非埋め込み行列パラメータを持つ。

$$N \;=\; 12Ld^2 \tag{11.11}$$

一方、重み共有されたトークン埋め込みテーブルが Vd 個のパラメータを追加する。

```
def gpt(*args, **kwargs):  # Fresh random weights for every model
    return d2l.GPT(*args, **kwargs, rngs=nnx.Rngs(d2l.get_key()))
```

非埋め込みパラメータ数が $12Ld^2$ に等しいこと（無視できる $O(Ld)$ 個の正規化のゲインおよびバイアスの範囲内で）を、6 ブロックのバイトレベル GPT と 12 ブロックの GPT-2 Small の両方で確認する。

```
def census(model):
    L, d = len(model.blks), model.num_hiddens
    emb = d2l.num_params(model.token_emb) + (
        d2l.num_params(model.pos_emb) if model.pos == 'learned' else 0)
    total = d2l.num_params(model)
    print(f'embedding {emb / 1e6:6.2f}M ({emb / total:5.1%}),  '
          f'blocks {(total - emb) / 1e6:6.2f}M '
          f'= {(total - emb) / (L * d ** 2):.2f} L d^2,  '
          f'total {total / 1e6:.2f}M')

census(gpt(256))
census(gpt(50257, num_hiddens=768, num_heads=12, num_blks=12,
           pos='learned', norm='layer', act='gelu', bias=True))
```

```
embedding   0.07M ( 1.4%),  blocks   4.72M = 12.01 L d^2,  total 4.79M
embedding  39.38M (31.6%),  blocks  85.06M = 12.02 L d^2,  total 124.44M
```

11.6.1.2 パラメータ・トークンあたり 6 FLOPs（$C \approx 6ND$）

なぜ 1 つの訓練トークンは、約 $6N$ FLOPs を $N = 12Ld^2$ 個の線形射影（linear projection）パラメータ全体で消費するのだろうか。

1. **順伝播（トークンあたり $2N$ FLOPs）：** トークンベクトル $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ に重み行列 $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ を掛けると、$d_{\text{in}} d_{\text{out}}$ 回の積和演算 $= 2P$ FLOPs が実行される（ここで $P = d_{\text{in}} d_{\text{out}}$ はこの行列のパラメータ数である）。全 N 個の行列パラメータと D トークンにわたって合計すると、順伝播の計算量は $2ND$ FLOPs になる。
2. **逆伝播（トークンあたり $4N$ FLOPs）：** $\mathbf{Y} = \mathbf{X}\mathbf{W}$ を通して誤差逆伝播法を実行するには、トークンあたり **2** 回のサイズ P の行列積が必要になる。すなわち $\nabla_{\mathbf{X}} \mathcal{L} = (\nabla_{\mathbf{Y}} \mathcal{L}) \mathbf{W}^\top$（$2P$ FLOPs）と $\nabla_{\mathbf{W}} \mathcal{L} = \mathbf{X}^\top (\nabla_{\mathbf{Y}} \mathcal{L})$（$2P$ FLOPs）であり、合計で逆伝播の計算量は $4ND$ FLOPs となる。

順伝播と逆伝播を足し合わせると、訓練計算量の基本的な近似式が得られる (Kaplan et al. 2020)。

$$C \;\approx\; 6ND. \tag{11.12}$$

（$QK^\top$ と AV のアテンションスコアのドット積をコンテキストウィンドウ長 T にわたって含めると、$12LTdD$ FLOPs が加わり、$6ND$ は $1 + \frac{T}{6d}$ 倍になる。この係数は 1 に近い値となる（$T \ll 6d$ が成り立つ限り）。因果アテンション行列のマスクされた半分をスキップする実装では、この補正項は半分になる。）

11.6.1.3 ハードウェアカウンタによる計算式の検証

Equation 11.12 は、フレームワークが計測する厳密な FLOP カウントと直接照らし合わせることで、`gpt(256, act='gelu')` の単一の訓練ステップ上で検証できる。

XLA のコスト解析は、コンパイルされたプログラムの FLOPs を CPU 上でカウントする。

```
model = gpt(256, act='gelu')
key_x, key_y = jax.random.split(d2l.get_key())
X = jax.random.randint(key_x, (64, 128), 0, 256)
Y = jax.random.randint(key_y, (64, 128), 0, 256)
graphdef, params, rest = nnx.split(model, nnx.Param, ...)
N = sum(p.size for p in jax.tree.leaves(params) if p.ndim == 2)
# Count on the CPU: on a GPU the count depends on the kernels XLA picks
params, rest, X, Y = jax.device_put((params, rest, X, Y), d2l.cpu())

def loss_fn(params, X, Y):
    logits = nnx.merge(graphdef, params, rest)(X)
    return optax.softmax_cross_entropy_with_integer_labels(
        logits.reshape(-1, logits.shape[-1]), Y.reshape(-1)).mean()

f = jax.jit(loss_fn).lower(params, X, Y).compile().cost_analysis()['flops']
s = jax.jit(jax.value_and_grad(loss_fn)).lower(
    params, X, Y).compile().cost_analysis()['flops']
print(f'forward {f:.3e} FLOPs, {f / (2 * N * X.size):.3f} x 2ND')
print(f'step    {s:.3e} FLOPs, {s / (6 * N * X.size):.3f} x 6ND')
print(f'backward / forward = {(s - f) / f:.2f}')
```

```
forward 8.575e+10 FLOPs, 1.094 x 2ND
step    2.625e+11 FLOPs, 1.116 x 6ND
backward / forward = 2.06
```

実測された逆伝播の計算量は順伝播の約 2 倍（2.0–2.1×）であり、$6ND$（重み共有出力ヘッドおよびアテンションスコア項 $12LTdD$ を考慮したもの）は厳密な演算子カウントと数パーセント以内で一致する。

11.6.2 WikiText-103 における小規模スケーリング実験

11.6.2.1 5 つのモデルサイズと 3 つのデータ予算

スケーリングの挙動を直接観察するため、**15 個のモデル**からなるグリッドを WikiText-103 上で訓練する。具体的には、非埋め込みパラメータ数が $N \in [0.3\text{M}, 14\text{M}]$ にわたる 5 種類のアーキテクチャ（`(d, L, H)` が `(96, 3, 4)` から `(384, 8, 8)` まで）と、3 種類のステップ予算 $s \in \{500, 1000, 2000\}$ ステップ（訓練バイト数 $D \in \{4.10\text{M}, 8.19\text{M}, 16.38\text{M}\}$ に相当）を掛け合わせる。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
print(f'{len(data.train) / 1e6:.0f}M training bytes, '
      f'{len(data.val) / 1e6:.2f}M validation bytes, '
      f'vocabulary of {len(data.vocab)}')
```

```
270M training bytes, 1.14M validation bytes, vocabulary of 256
```

コーパスを読み込んだら、全 $5 \times 3 = 15$ 通りの (size, budget) 設定を、幅に応じてスケーリングしたピーク学習率 $\eta = 2 \times 10^{-3} \cdot (256/d)$ で訓練し、各モデルの非埋め込みパラメータ数 Ns と検証交差エントロピー val を記録する。

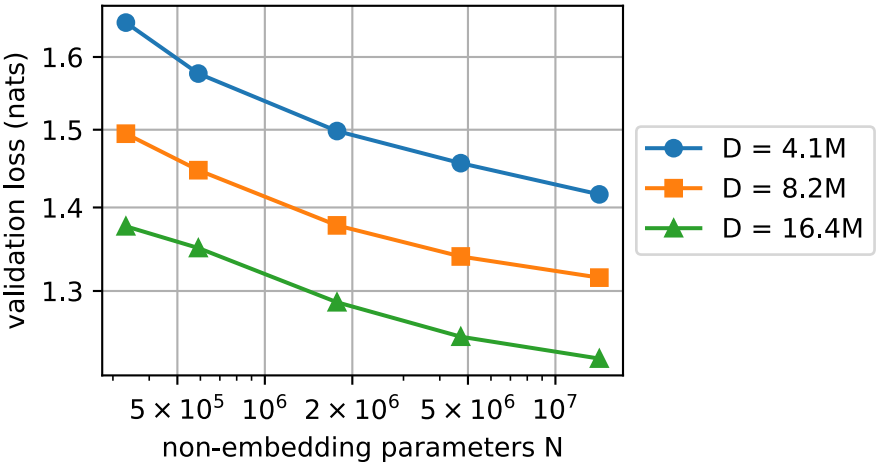
```
sizes = ((96, 3, 4), (128, 3, 4), (192, 4, 6), (256, 6, 8), (384, 8, 8))
budgets = (500, 1000, 2000)  # Steps of 64 windows of 128 bytes
Ns, train, val = [], [], {num_steps: [] for num_steps in budgets}
for d, num_blks, num_heads in sizes:
    for num_steps in budgets:
        model = gpt(len(data.vocab), num_hiddens=d, num_heads=num_heads,
                    num_blks=num_blks)
        optimizer = d2l.LMOptimizer(model, lr=2e-3 * 256 / d,
                                    num_steps=num_steps,
                                    warmup=max(100, num_steps // 10))
        losses = d2l.train_steps(model, data, optimizer, num_steps)
        val[num_steps].append(
            math.log(d2l.lm_perplexity(model, data.val_dataloader())))
    # The size of the models just trained, and the training loss of the
    # longest of their runs
    Ns.append(d2l.num_params(model) - d2l.num_params(model.token_emb))
    train.append(sum(losses[-50:]) / 50)
    print(f'd = {d:3d}, {num_blks} blocks, N = {Ns[-1] / 1e6:5.2f}M: '
          'validation ' + ' / '.join(f'{val[s][-1]:.2f}' for s in budgets)
          + f', training {train[-1]:.2f}')
```

```
d =  96, 3 blocks, N =  0.33M: validation 1.65 / 1.49 / 1.38, training 1.39
d = 128, 3 blocks, N =  0.59M: validation 1.58 / 1.45 / 1.35, training 1.37
d = 192, 4 blocks, N =  1.77M: validation 1.50 / 1.38 / 1.29, training 1.30
d = 256, 6 blocks, N =  4.72M: validation 1.46 / 1.34 / 1.25, training 1.27
d = 384, 8 blocks, N = 14.16M: validation 1.42 / 1.32 / 1.22, training 1.23
```

11.6.2.2 モデルサイズ、データ予算、および計算量に対する損失

検証交差エントロピーを、非埋め込みパラメータ数 N の対数軸に対してトークン予算 D ごとにプロットし、さらに総訓練計算量 $C = 6ND$ に対してもプロットする。

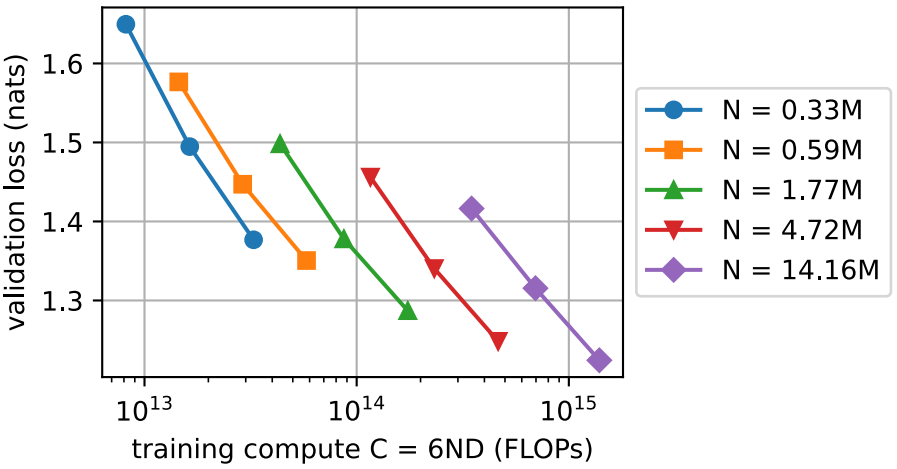
```
tokens = {s: s * 64 * 128 for s in budgets}  # Tokens read in s steps
d2l.plot(Ns, [val[s] for s in budgets], 'non-embedding parameters N',
         'validation loss (nats)',
         legend=[f'D = {tokens[s] / 1e6:.1f}M' for s in budgets],
         xscale='log', yscale='log', fmts=('C0o-', 'C1s-', 'C2^-'),
         legend_loc='outside')
```



同じ 15 個の検証損失を総訓練計算量 $C = 6ND$ に対して再プロットすると（あわせて計算量が近い 3 つの実行、すなわち $C \approx 2\text{–}3 \times 10^{14}$ FLOPs で互いに $2\times$ 以内に収まるものを出力すると）、FLOP 予算が増えるにつれて計算量最適（compute-optimal）なモデルサイズが上方へシフトする様子がわかる。

```
d2l.plot([[6 * N * tokens[s] for s in budgets] for N in Ns],
         [[val[s][i] for s in budgets] for i in range(len(Ns))],
         'training compute C = 6ND (FLOPs)', 'validation loss (nats)',
         legend=[f'N = {N / 1e6:.2f}M' for N in Ns], xscale='log',
         fmts=('C0o-', 'C1s-', 'C2^-', 'C3v-', 'C4D-'), legend_loc='outside')
for i, s in ((2, 2000), (3, 1000), (4, 500)):  # Larger N, fewer tokens
    print(f'N = {Ns[i] / 1e6:5.2f}M, D = {tokens[s] / 1e6:4.1f}M tokens: '
          f'C = {6 * Ns[i] * tokens[s]:.1e} FLOPs, '
          f'validation {val[s][i]:.2f}')
```

```
N =  1.77M, D = 16.4M tokens: C = 1.7e+14 FLOPs, validation 1.29
N =  4.72M, D =  8.2M tokens: C = 2.3e+14 FLOPs, validation 1.34
N = 14.16M, D =  4.1M tokens: C = 3.5e+14 FLOPs, validation 1.42
```



2 つのパターンが際立っている。

1. 最大のトークン予算（2,000 ステップ、$D = 16.4$M バイト）では、検証損失は小型・中型モデルにわたって $\log N$ に対してほぼ線形に減少したのち、最大のモデル（$N = 14.16$M）で横ばいに曲がる。そこでは 2,000 ステップ（パラメータあたり $D/N \approx 1.2$ バイト）であっても、14M 個のパラメータに対してはいまだデータ不足の状態にある。
2. 全 15 回の実行を総訓練 FLOPs $C = 6ND$ に対してプロットすると、その下側包絡線（lower envelope）が**パレート最適な計算フロンティア**（Pareto-optimal compute frontier）を描く。計算量が小さい領域では小型モデルが優位に立ち、計算量が大きい領域では十分なステップ数が与えられた時点で大型モデルが逆転する。

11.6.3 計算量最適な配分（Chinchilla スケーリング）

11.6.3.1 Chinchilla のパラメトリック法則

Hoffmann et al. (2022)（および Besiroglu et al. (2024) による再フィット）は、検証損失曲面 $\mathcal{L}(N, D)$ を、削減不能なテキストエントロピーの下限 L_∞ と、N および D に関する 2 つのべき乗則ペナルティの和としてモデル化する。

$$\mathcal{L}(N, D) \;=\; L_\infty \;+\; \frac{A}{N^{\alpha}} \;+\; \frac{B}{D^{\beta}}. \tag{11.13}$$

11.6.3.2 計算量最適な分割（$N^*(C)$ と $D^*(C)$）

Equation 11.13 を、N と D に関して制約 $C = 6ND$ のもとで最小化すると（$D = C/(6N)$ を代入して $\partial\mathcal{L}/\partial N = 0$ とおくと）、最適なモデルサイズ $N^*(C)$ と最適なトークン予算 $D^*(C)$ の両方についてべき乗則スケーリングが得られる。

$$N^*(C) \;=\; G\left(\frac{C}{6}\right)^{a}, \qquad D^*(C) \;=\; G^{-1}\left(\frac{C}{6}\right)^{b}, \tag{11.14}$$

ここで、各定数は次のとおりである。

$$\begin{aligned} a \;&=\; \frac{\beta}{\alpha+\beta}, \qquad b \;=\; \frac{\alpha}{\alpha+\beta}, \\ G \;&=\; \left(\frac{\alpha A}{\beta B}\right)^{\frac{1}{\alpha+\beta}}. \end{aligned} \tag{11.15}$$

$\alpha \approx \beta \approx 0.34$–$0.37$ であるため、指数は $a \approx b \approx 0.5$ を満たす。すなわち、訓練計算量 C が $100\times$ に増えると、モデルサイズ N^* と訓練トークン数 D^* はともに $\sim 10\times$ に増え、パラメータあたり約 $D^*/N^* \approx 20$ トークンという計算量最適な比率はほぼ一定に保たれる。

これは、この問いに対する最初の答えではなかった。その 2 年前、Kaplan et al. (2020) は同種のべき乗則を当てはめ、追加の計算量の大部分はモデルに投じるべきだと結論づけていた。すなわち $N^* \propto C^{0.73}$ であり、データには $D^* \propto C^{0.27}$ しか残らない（Figure 11.13）。この 2 つの研究の食い違いは、主に学習率スケジュール（learning rate schedule）に起因する。Kaplan らはすべての実行に 1 つのスケジュールを使い、より小さなトークン予算の損失を、まだ学習率が減衰していない中間チェックポイントから読み取っていた。これでは少ないトークンでの訓練が実際より悪く見えるため、データの効果が実際より小さく見積もられてしまう。また、扱ったモデルも小さく、高々 1.5B パラメータであった。Hoffmann らは最大 16B パラメータのモデルを訓

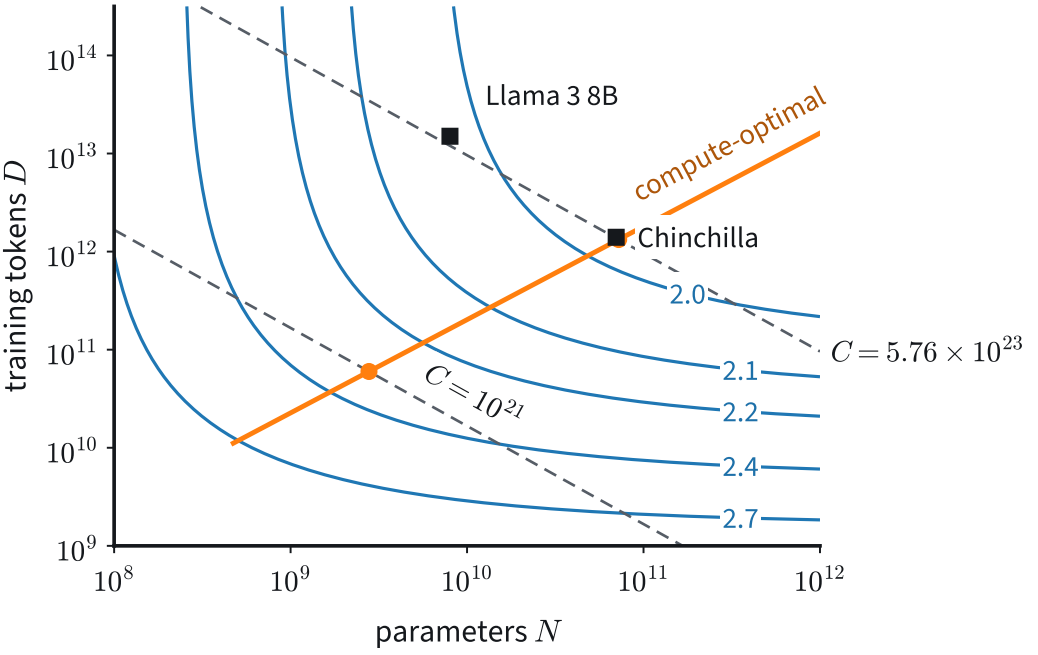


Figure 11.12: 後述する Besiroglu ら（2024）の再フィット定数を用いた、パラメータ数と訓練トークン数にわたる Chinchilla 則の損失等高線。破線：等しい計算量 $C = 6ND$ の予算線、点：各予算における計算量最適モデル、オレンジ色の実線：計算量最適経路。Chinchilla を示す四角は自身の予算の最適点上に位置し、その点を覆っている。一方、Chinchilla に近い予算で訓練された Llama 3 8B は、最適サイズの 10 分の 1 の位置にあり、最適経路から大きく外れている。

練し、コサインスケジュールを各実行の長さに合わせた。前述のグリッド実験でもこれと同じ方法をとっており、15 回の実行のそれぞれが固有のコサインホライズンを持つのはそのためである（そうしない場合に何が起きるかは、演習問題 3 で計測する）。

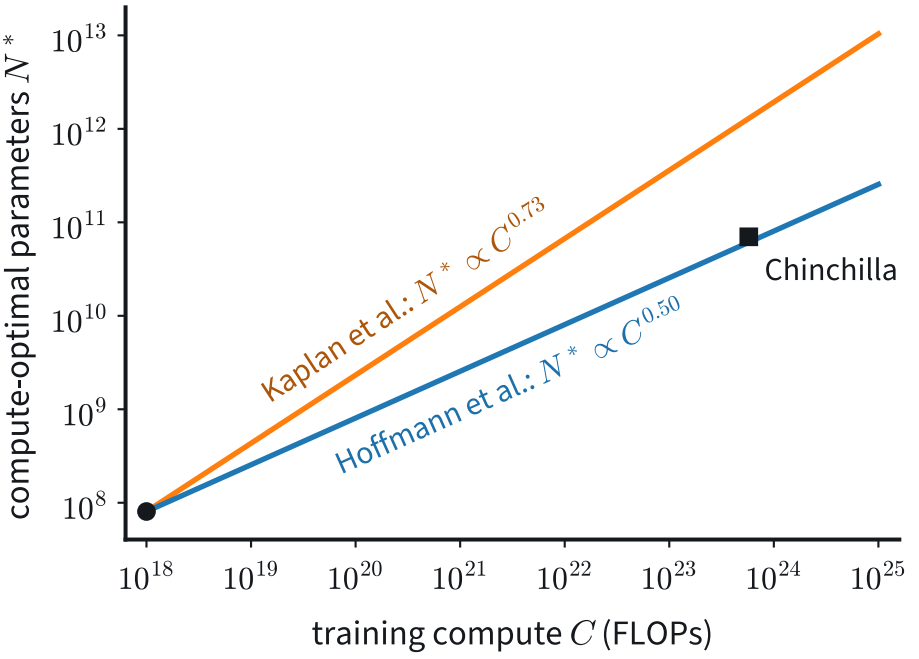


Figure 11.13: 公表された 2 つの指数、すなわち 0.73（Kaplan ら）と 0.5（Hoffmann ら）のもとでの訓練計算量に対する計算量最適モデルサイズ。いずれの直線も 10^{18} FLOPs において同じモデルを通るように引かれており、傾きのみが各論文に由来する。予算を 100× に増やすと、第 1 の指数では 29× 大きいモデルが、第 2 の指数では 10× 大きいモデルが求められる。四角は Chinchilla を示す。

以下の関数 `optimum` は、Equation 11.14 を Chinchilla の訓練予算 $C = 5.76 \times 10^{23}$ FLOPs において、Hoffmann らが公表した定数と Besiroglu らによる再フィット定数のそれぞれについて評価する。

```
fits = {'Hoffmann et al.': (1.69, 406.4, 410.7, 0.34, 0.28),
        'Besiroglu et al.': (1.82, 482.0, 2085.4, 0.348, 0.366)}

def optimum(C, L_inf, A, B, alpha, beta):
    G = (alpha * A / (beta * B)) ** (1 / (alpha + beta))
    N = G * (C / 6) ** (beta / (alpha + beta))
    return N, C / (6 * N)

for name, fit in fits.items():
    N, D = optimum(5.76e23, *fit)
    print(f'{name:17s} N* = {N / 1e9:3.0f}B, D* = {D / 1e12:.2f}T, '
          f'{D / N:3.0f} tokens per parameter')
```

```
Hoffmann et al.   N* =  32B, D* = 2.98T,  93 tokens per parameter
Besiroglu et al.  N* =  72B, D* = 1.33T,  18 tokens per parameter
```

11.6.3.3 推論を見据えた Chinchilla 最適を超える訓練

なぜ Llama 3 8B（Figure 11.12）は、8B パラメータのモデルを 15T トークン（パラメータあたり $D/N \approx 1{,}875$ トークンで、Chinchilla の比率のほぼ 100×）で訓練する道を選び、Chinchilla 最適な 70B モデルを 1.4T トークン（同じ事前学習計算量 $C \approx 7 \times 10^{23}$ FLOPs）で訓練しなかったのだろうか。

それは、Chinchilla が**事前学習の FLOPs $6ND$ のみ**を最小化し、デプロイ後のモデルのサービングに費やされる $2ND_{\text{inf}}$ の推論 FLOPs や KV キャッシュメモリを無視しているからである (Sardana et al. 2024)。一定の損失をもつ等高線 $\mathcal{L}(N, D) = \mathcal{L}_{\text{target}}$ に沿って移動してみると、8B モデルを 15T トークンで過剰訓練（over-training）すれば、Chinchilla 配分の 30B–40B モデルに近い損失に到達しつつ、その後に生成するすべての推論トークンを 4–5× 安くサービングできることがわかる。

```
L_inf, A, B, alpha, beta = fits['Besiroglu et al.']
loss = lambda N, D: L_inf + A / N ** alpha + B / D ** beta
reported = {'Chinchilla': (70e9, 1.4e12), 'Llama 3 8B': (8e9, 15e12),
            'Llama 3 405B': (405e9, 15.6e12), 'OLMo 2 7B': (7e9, 4.05e12),
            'Gemma 3 27B': (27e9, 14e12)}
for name, (N, D) in reported.items():
    N_opt, D_opt = optimum(6 * N * D, L_inf, A, B, alpha, beta)
    print(f'{name:12s} {D / N:5.0f} tokens per parameter, '
          f'{N / N_opt:4.2f} x optimal size, '
          f'loss +{loss(N, D) - loss(N_opt, D_opt):.3f}')
```

```
Chinchilla      20 tokens per parameter, 0.96 x optimal size, loss +0.000
Llama 3 8B    1875 tokens per parameter, 0.10 x optimal size, loss +0.054
Llama 3 405B    39 tokens per parameter, 0.65 x optimal size, loss +0.001
OLMo 2 7B      579 tokens per parameter, 0.18 x optimal size, loss +0.037
Gemma 3 27B    519 tokens per parameter, 0.18 x optimal size, loss +0.023
```

11.6.4 まとめ

L 層、幅 d で SwiGLU（$d_{\text{ff}} \approx \frac{8}{3}d$）を備えた密な Transformer は、$N \approx 12Ld^2$ 個の非埋め込み行列パラメータ（アテンションに $4Ld^2$、FFN に $8Ld^2$）を含み、$C \approx 6ND$ の訓練 FLOPs を D トークンにわたって消費する（順伝播で $2ND$、逆伝播で $4ND$、Equation 11.12）。**Chinchilla スケーリング則** $\mathcal{L}(N, D) = L_\infty + AN^{-\alpha} + BD^{-\beta}$（Equation 11.13）のもとでは、固定された事前学習計算量 $C = 6ND$ で検証損失を最小化すると、パラメータ数とトークン数は指数 $a = \beta/(\alpha+\beta) \approx 0.5$ および $b = \alpha/(\alpha+\beta) \approx 0.5$（Equation 11.14、Equation 11.15）に従ってスケーリングし、パラメータあたり約 $D^*/N^* \approx 20$ トークンという比率が得られる。**生涯推論計算量**（lifetime inference compute、$2ND_{\text{inf}}$）とサービング時のレイテンシを考慮すると、総計算量の最適点は等損失等高線に沿って、パラメータあたり数百〜数千トークンで過剰訓練されたより小さなモデル N の側へとシフトする。

11.6.5 演習問題

1. **アテンションスコアの FLOP 補正**（$1 + T/(6d)$）。本文の Equation 11.12 は、アテンションスコアと値の積 $\mathbf{QK}^\top$ および $\mathbf{AV}$、ならびに出力ヘッド $\mathbf{H}^{(L)}\mathbf{E}^\top$ を省略している。
    1. L ブロック、幅 d、ウィンドウ長 T のモデルにおいて、順伝播のアテンションのドット積がウィンドウあたり $4LT^2d$ FLOPs（トークンあたり $4LTd$ FLOPs）を実行し、順伝播と逆伝播を合わせたアテンションのドット積がトークンあたり $12LTd$ FLOPs

（因果対称性が FlashAttention によって利用される場合は $6LTd$）を実行することを示せ。

2. $6LTd$ と $6N = 72Ld^2$ の比をとることで、因果アテンションが加える相対的な FLOP 比率が $T/(12d)$（因果タイルスキップなしでは $T/(6d)$）となることを示せ。コンテキスト長 T^* がいくつのときに、アテンションの計算量は Llama 3 8B（$d = 4096$）の線形射影の FLOPs と一致するか。

2. **スケーリンググリッドにおける総パラメータと非埋め込みパラメータ**。本節の 5 サイズにわたる WikiText-103 スイープの各モデルではバイト語彙（$V = 256$）を用いたため、Vd は高々約 7%（対 $12Ld^2$ 比）にとどまっていた。`census` を同じ 5 つの `(d, L, H)` バックボーンに対して、代わりに BPE 語彙 $V = 128{,}256$ トークン（Llama 3）を用いた場合について実行せよ。最小モデル（`d=96, L=3`）と最大モデル（`d=384, L=8`）では、埋め込みテーブルが総パラメータのどれだけの割合を占めるか。また、なぜ Kaplan et al. (2020) はべき乗則を $N_{\text{non-emb}} = 12Ld^2$ に対して当てはめ、N_{total} に対しては当てはめなかったのか。

3. **コサイン減衰のホライズンと固定ステップホライズン**。本節の 15 実行のスケーリング実験では、各予算 $s \in \{500, 1000, 2000\}$ が固有のコサインスケジュールを実行し、学習率をステップ s でゼロまで減衰させた（スケジュールを実行の長さに合わせることは、Hoffmann et al. (2022) による Kaplan et al. (2020) への決定的な修正であった）。3 つの予算を中型モデル `(192, 4, 6)` について再実行せよ。ただし、単一の固定コサインホライズン `num_steps=2000` を `LMOptimizer` 内で用い、ステップ 500、1000、2000 における検証損失を単に読み取るものとする。$s = 500$ および $s = 1000$ で学習率を減衰させないことによって、短い実行の損失はどれだけ過大評価されるか。

4. **Chinchilla 則の 15 点グリッドへの当てはめ**。`scipy.optimize.minimize`（またはグリッド探索 + 非線形最小二乗法）を用いて、パラメータ $(L_\infty, A, B, \alpha, \beta)$（Equation 11.13）を、15 個の `(N, D, val_loss)` 点（本節の WikiText-103 スイープで計測したもの）に当てはめよ。スケーリング指数 $a = \beta/(\alpha+\beta)$ と $b = \alpha/(\alpha+\beta)$ は、このバイトレベルの 15 点グリッドからどのような値として得られるか。

5. **推論需要 D_{inf} を伴う生涯計算量の最適化**。サイズ N のモデルを D トークンで訓練し、D_{inf} 個の推論トークンをサービングするためにデプロイすると、消費される総生涯 FLOPs は $C_{\text{tot}}(N, D) = 6ND + 2ND_{\text{inf}}$ となる。

   1. 制約 $\mathcal{L}(N, D) = L_\infty + AN^{-\alpha} + BD^{-\beta} = \mathcal{L}_{\text{target}}$ に対してラグランジュ乗数法を用い、生涯計算量の最小点 (N^*, D^*) において限界比が次の関係を満たすことを示せ。

$$\frac{\beta B (D^*)^{-\beta}}{\alpha A (N^*)^{-\alpha}} = \frac{6N^*D^*}{6N^*D^* + 2N^*D_{\text{inf}}} = \frac{1}{1 + \frac{D_{\text{inf}}}{3D^*}}.$$

   2. $D_{\text{inf}} = 0$ のときにこれが Chinchilla の均衡式 $\alpha A (N^*)^{-\alpha} = \beta B (D^*)^{-\beta}$ に帰着することを確かめよ。さらに (N^*, D^*) を、$\mathcal{L}_{\text{target}} = \mathcal{L}(70\text{B}, 1.4\text{T})$ かつ $D_{\text{inf}} \in \{10\text{T}, 100\text{T}, 500\text{T}\}$ トークンの各場合について数値的に求めよ。

6. **活性化チェックポイントと $8ND$ 則**。完全なブロック単位の活性化チェックポイントは、各ブロックへの入力 $\mathbf{X}^{(\ell-1)}$ のみを順伝播（$2ND$ FLOPs）の間に保存し、誤差逆伝播法において逆方向の行列積（$4ND$ FLOPs）を計算する前に各ブロックの順伝播を一度再計算する。活性化チェックポイントによって事前学習計算量の式が $C \approx 6ND$ から $C \approx 8ND$ に変わることを示せ。また、6 を 8 に置き換えること（Equation 11.14）によって、$N^*(C)$ と $D^*(C)$ が固定された FLOP 予算 C のもとでどのようにスケーリングするかを求めよ。

11.7 混合エキスパート（MoE）

密な Transformer（11.6 節）では、すべてのトークンがすべてのパラメータを使用する。そのため、モデル容量 $N \approx 12Ld^2$ を 2 倍にすると、訓練コスト（トークンあたり $6N$ FLOPs）も推論の計算量（トークンあたり $2N$ FLOPs）も 2 倍になる。モデルの保持パラメータ容量 N_{total} を拡大しつつ、トークンごとに実行されるアクティブパラメータ数 N_{active} を増やさずに済ませることはできるだろうか。

11.1 節で見たように、フィードフォワードネットワークは密なブロックのパラメータ数の 3 分の 2（$8d^2$、ブロック全体では $12d^2$）を占め、各トークンベクトル $\mathbf{x} \in \mathbb{R}^d$ に対して独立に**位置ごと**（position-wise）に作用する。**スパース混合エキスパート**（sparse mixture-of-experts、MoE）層 (Jacobs et al. 1991; Shazeer et al. 2017; Lepikhin et al. 2021; Fedus et al. 2022) は、この独立性を活用する（Figure 11.14）。すなわち、ブロック内の単一の FFN を並列な E 個の FFN **エキスパート**群 $\{\text{FFN}_1, \ldots, \text{FFN}_E\}$ に置き換え、軽量な線形**ルータ** $\mathbf{W}_r \in \mathbb{R}^{d \times E}$ を用いて各トークン $\mathbf{x}$ を上位 k 個のエキスパートのみにディスパッチする（$1 \le k \ll E$）。

11.7.1 条件付き計算：保持パラメータとアクティブパラメータ

L 層の MoE Transformer において、各ブロックがアテンションのパラメータ P_{attn}、ルーティングされる E 個の SwiGLU エキスパート（中間幅 w、エキスパートあたり $P_{\text{exp}} = 3dw$ 個のパラメータ）、および E_{shared} 個の常時アクティブな共有エキスパート（shared expert）を持つとする (Dai et al. 2024)。このとき、全 $E + E_{\text{shared}}$ 個のエキスパートをメモリに保持する必要があるが（N_{total}）、各トークンが実行するのは $k + E_{\text{shared}}$ 個のエキスパートのみである（N_{active}）。

```python
def moe_accounting(name, d, width, num_experts, num_active, num_layers,
                   shared=0, unit=1e9):
    """Expert parameters stored, and expert and router parameters active
    per token, in a stack of SwiGLU expert layers."""
    per_expert = 3 * d * width          # SwiGLU: three d x width matrices
    store = num_layers * (num_experts + shared) * per_expert
    active = num_layers * ((num_active + shared) * per_expert
                           + d * num_experts)      # + the router
    tag = 'B' if unit == 1e9 else 'M'
    print(f'{name:>12}: {store/unit:5.3g}{tag} in experts, '
          f'{active/unit:5.3g}{tag} active per token, '
          f'ratio {store/active:4.1f}x')
    return store, active

moe_accounting('ours', 128, 341, 8, 1, 2, unit=1e6)
moe_accounting('DeepSeek-V3', 7168, 2048, 256, 8, 58, shared=1)
store, active = moe_accounting('Mixtral 8x7B', 4096, 14336, 8, 2, 32)
attn = 32 * (2 * 4096 * 4096 + 2 * 4096 * 1024)  # GQA: 32 Q, 8 KV heads
emb = 2 * 32000 * 4096                           # embedding + output head
print(f'{"":>12}  with attention and embeddings: '
      f'{(store + attn + emb)/1e9:.1f}B stored, '
      f'{(active + attn + emb)/1e9:.1f}B active')
```

```
        ours:   2.1M in experts, 0.264M active per token, ratio  7.9x
 DeepSeek-V3:   656B in experts,  23.1B active per token, ratio 28.4x
Mixtral 8x7B:  45.1B in experts,  11.3B active per token, ratio  4.0x
              with attention and embeddings: 46.7B stored, 12.9B active
```

N_{total} と N_{active} がそれぞれ異なるシステム資源を規定している点に注目しよう。

- **トークンあたり FLOPs（訓練とプレフィル）**は $N_{\textbf{active}}$ に比例してスケールする（$C_{\text{train}} \approx 6N_{\text{active}}D$）。
- **VRAM 容量**は $N_{\textbf{total}}$ に比例してスケールする。全 E 個のエキスパートをデバイスメモリ上に常駐させる必要があるためである。
- **バッチデコードの帯域幅：**バッチサイズ $B = 1$ では、1 つのトークンが HBM からストリーミングするのは k 個のエキスパートのみである（N_{active} 個分の重みバイト数）。しかし、B トークンのバッチ全体で見ると、異なるトークンが異なるエキスパートを活性化させる。$B \gg E/k$ になると、ほぼ全 E 個のエキスパートがバッチ内の少なくとも 1 つのトークンによって参照されるため、各デコードステップでは HBM からほぼ $N_{\textbf{total}}$ 個分の重みバイト数をストリーミングする一方で、実行する演算は $2BN_{\text{active}}$ FLOPs にとどまる。

11.7.2 混合エキスパート層

11.7.2.1 Top-k ルーティング

各トークン $\mathbf{x} \in \mathbb{R}^d$ に対し、ルータは確率 $\mathbf{p}(\mathbf{x}) = \text{softmax}(\mathbf{x}\mathbf{W}_r) \in \Delta^{E-1}$ を計算し、集合 $T(\mathbf{x}) \subset \{1, \dots, E\}$ としてスコア上位 k 個のエキスパートを選択する。続いて、その生の確率をそのまま重み $g_e(\mathbf{x}) = p_e(\mathbf{x})$ として保持し（$T(\mathbf{x})$ 上で再正規化すると Top-1 ルータの勾配が消失してしまう。演習問題 1）、次の重み付き和を計算する。

$$\text{MoE}(\mathbf{x}) \;=\; \sum_{e \in T(\mathbf{x})} g_e(\mathbf{x})\, \text{FFN}_e(\mathbf{x}). \tag{11.16}$$

11.7.2.2 固定容量エキスパートバッファへのトークンのディスパッチ

コンパイル実行や分散訓練では、各エキスパート $e \in \{1, \dots, E\}$ はあらかじめ容量 $S = \lceil \gamma \cdot nk/E \rceil$ スロットの固定バッファを、n トークンのバッチに対して確保する。ここで、nk/E は完全に均等化されたときのエキスパートあたりの負荷であり、$\gamma \geq 1$ は**容量係数**（capacity factor）である（$\gamma = E$ とすれば $S = nk$ となり、ドロップは無効化される）。S 個を超えるトークン・エキスパート割り当てがエキスパート e に集中した場合、超過分の割り当ては**ドロップ**される（Equation 11.16 への寄与が 0 になり、該当トークンは外側のブロックの残差スキップ接続をそのまま通過する）。Figure 11.15 は、6 個のトークンがこのディスパッチ処理を通過する様子を追ったものである。

`MoELayer` は、ルータ、S スロットのバッファへのディスパッチ、およびドロップ規則を実装する。

```
class MoELayer(nnx.Module):
    """Token-choice top-k mixture of SwiGLU d2l.FeedForward experts, each
    with a buffer of S = capacity_factor * n k / E slots for n tokens."""
    def __init__(self, num_hiddens, num_experts, num_active,
                 capacity_factor=1.25, *, rngs):
        self.num_experts, self.num_active = num_experts, num_active
        self.capacity_factor = capacity_factor
        self.router = d2l.Linear(num_hiddens, num_experts, use_bias=False,
                                 rngs=rngs)
        self.experts = nnx.List([d2l.FeedForward(num_hiddens, rngs=rngs)
                                 for _ in range(num_experts)])
        # Added to the probabilities for the selection only; zero until a
        # balancing method moves it
```

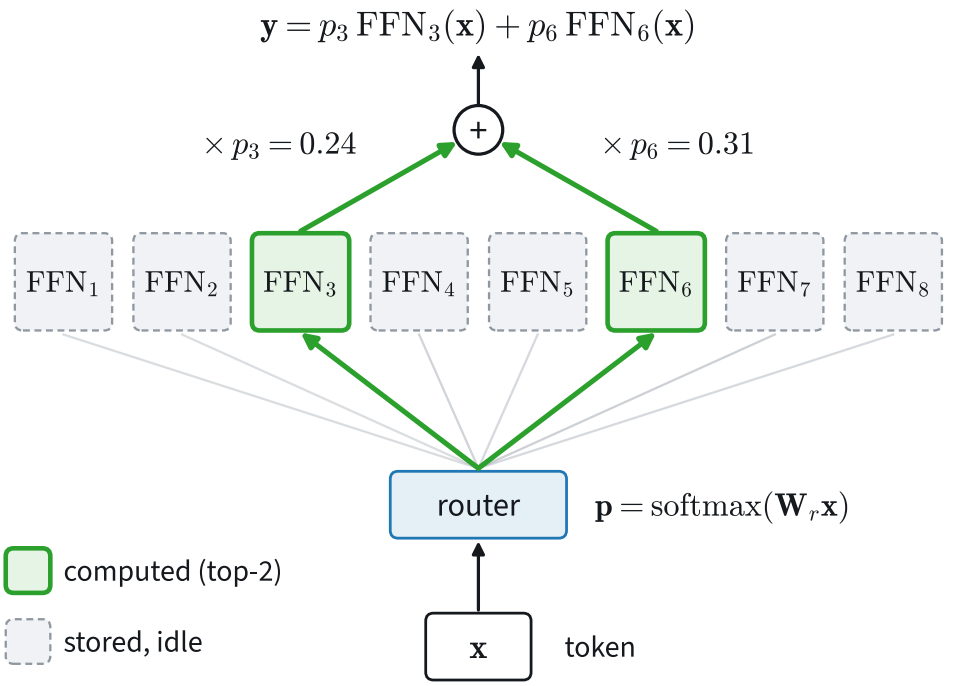


Figure 11.14: 8 個のエキスパートを持ち、トークンごとにそのうち 2 個がアクティブになる混合エキスパート層。ルータがすべてのエキスパートをスコアリングする。最も確率の高い 2 個のエキスパートが実行され、それぞれの出力が確率で重み付けされて加算される。残りの 6 個のエキスパートは保持されているが、このトークンに対しては何も処理しない。

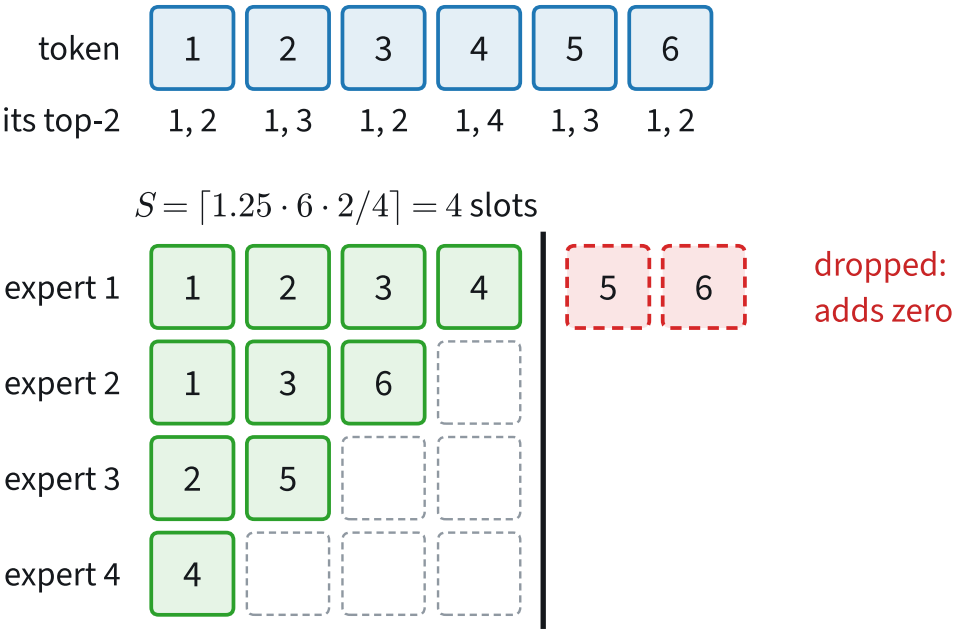


Figure 11.15: 6 個のトークンをそれぞれ上位 2 個（Top-2）ずつ 4 個のエキスパートにディスパッチする例（容量係数 $\gamma = 1.25$ により、各エキスパートのスロット数は $S = 4$）。割り当てはトークン順にエキスパートごとのキューに入る。エキスパート 1 への 5 番目と 6 番目の割り当ては S スロットを超えるためドロップされ、トークン 5 と 6 は第 2 候補のエキスパートのみで処理される。すべての割り当てがドロップされたトークンには残差のみが残る。

```
        self.expert_bias = nnx.Variable(jnp.zeros(num_experts))
        # Assignments per expert, dropped or not, and the dropped ones,
        # counted by record() until reset_stats(); the last batch's
        # balancing loss
        self.load = nnx.Variable(jnp.zeros(num_experts))
        self.dropped = nnx.Variable(jnp.zeros(()))
        self.aux_loss = nnx.Variable(jnp.zeros(()))

    def __call__(self, X):
        x = X.reshape(-1, X.shape[-1])                # (n, d): the tokens
        (n, d), E, k = x.shape, self.num_experts, self.num_active
        S = math.ceil(self.capacity_factor * n * k / E)  # slots per expert
        P = jax.nn.softmax(self.router(x), -1)        # (n, E)
        _, idx = jax.lax.top_k(P + self.expert_bias[...], k)  # (n, k)
        e = idx.ravel()                               # expert of each pair
        onehot = jax.nn.one_hot(e, E, dtype=jnp.int32)  # (n k, E)
        rank = (onehot.cumsum(0) * onehot).sum(1) - 1  # place in its queue
        slot = jnp.where(rank < S, e * S + rank, E * S)  # E S: dropped
        buf = jnp.zeros((E * S, d), x.dtype).at[slot].set(
            jnp.repeat(x, k, 0), mode='drop')
        Y = self.run_experts(buf.reshape(E, S, d)).reshape(E * S, d)
        Y = Y.at[slot].get(mode='fill', fill_value=0)  # dropped: zeros
        weights = jnp.take_along_axis(P, idx, -1)     # (n, k): the p_i
        out = (weights[..., None] * Y.reshape(n, k, d)).sum(1)
        self.record(onehot.sum(0), (rank >= S).sum(), P)
        return out.reshape(X.shape)

    def run_experts(self, H):
        """Each expert's SwiGLU on its own buffer: (E, S, d) -> (E, S, d)."""
        W = lambda name: jnp.stack(  # (E, d_in, d_out): one per expert
            [getattr(e, name).kernel[...] for e in self.experts])
        return (jax.nn.silu(H @ W('W_g')) * (H @ W('W_1'))) @ W('W_2')

    def record(self, counts, num_dropped, P):
        """Add a batch's assignments per expert and drops to the totals,
        and keep its balancing loss, a measure of imbalance."""
        self.load[...] += counts
        self.dropped[...] += num_dropped
        f = counts / counts.sum()                     # this batch's shares
        self.aux_loss[...] = self.num_experts * (f * P.mean(0)).sum()

    def reset_stats(self):
        self.load[...] = jnp.zeros_like(self.load[...])
        self.dropped[...] = jnp.zeros(())
```

2 つの小さなユーティリティ関数 `moe_layers(model)` と `active_params(model)` は、ネットワーク内のすべての `MoELayer` サブモジュールを収集し、単一トークンの順伝播に関与するパラメータ数（エキスパート群の k/E とすべての共有パラメータの合計）を集計する。

```
def moe_layers(model):
    """The MoELayer modules of a model."""
    return [m for _, m in nnx.iter_graph(model) if isinstance(m, MoELayer)]
```

```
def active_params(model):
    """Parameters used per token: all but the unselected experts."""
    return d2l.num_params(model) - sum(
        d2l.num_params(l.experts) * (l.num_experts - l.num_active)
        // l.num_experts for l in moe_layers(model))
```

d2l.TransformerBlock（11.1 節）は ffn_factory フックを受け取るため、密な FFN を 8 エキスパート・Top-2 の MoELayer に置き換えるにはラムダ式を 1 つ渡すだけでよい。

```
blk = d2l.TransformerBlock(
    256, num_heads=8, rngs=nnx.Rngs(d2l.get_key()),
    ffn_factory=lambda rngs: MoELayer(256, 8, 2, rngs=rngs))
X = jax.random.normal(d2l.get_key(), (4, 100, 256))
d2l.check_shape(blk(X), X.shape)  # The expert layer fits into the block
moe = MoELayer(256, num_experts=8, num_active=2,
               rngs=nnx.Rngs(d2l.get_key()))
print(f'{d2l.num_params(moe)} parameters stored, '
      f'{active_params(moe)} active per token')
```

```
4198400 parameters stored, 1051136 active per token
```

11.7.3 ルーティング崩壊と負荷分散

11.7.3.1 負荷の偏りと正のフィードバック

初期化の時点でさえ、ランダムなルータの重みによって、一部のエキスパートには他よりも多くのトークンが送られる。負荷分散（load balancing）の仕組みなしに訓練を進めると、**正のフィードバックループ**（positive-feedback loop）がこの非対称性を増幅し、**ルーティング崩壊**（routing collapse）を引き起こす (Shazeer et al. 2017)。すなわち、ステップ 1 でより多くのトークンを受け取ったエキスパートほど大きな勾配更新を受け、ステップ 2 で言語モデリング損失をより速く減少させ、ステップ 3 で $\nabla_{\mathbf{w}_r}\mathcal{L}$ から報酬を受けてルータのロジットがさらに高くなる。このまま放置すると、ごく少数のエキスパートが上位 k 個のスロットの大半を独占し、残りのエキスパートにはほとんどトークンが回らず何も学習できない状態に陥りうる。

11.7.3.2 2 つの解決策：補助負荷分散損失と損失不要バイアスコントローラ

1. **Switch 補助負荷分散損失** (Fedus et al. 2022)：$f_e = \frac{1}{nk}\sum_{t=1}^{n}\mathbb{I}[e \in T(\mathbf{x}_t)]$ をエキスパート e にディスパッチされたルーティングスロットの経験的な割合（n トークンのバッチ内）とし、$\bar{p}_e = \frac{1}{n}\sum_{t=1}^{n} p_e(\mathbf{x}_t)$ をエキスパート e に割り当てられたルータ確率の平均とする。訓練損失に次の補助損失による項 $\alpha\mathcal{L}_{\text{aux}}$ を加える。

$$\mathcal{L}_{\text{aux}} \;=\; E\sum_{e=1}^{E} f_e\,\bar{p}_e, \tag{11.17}$$

   この項は、高い確率 $\bar{p}_e$ が負荷の集中したエキスパート（f_e が大きいもの）に与えられることにペナルティを課し、最小値 $\mathcal{L}_{\text{aux}} = 1$ をとるのは $f_e = 1/E$ がすべての e について成り立つときである。

2. **損失不要の動的ルータバイアス制御**（loss-free dynamic router bias control、L. Wang et al. (2024)。DeepSeek-V3 (DeepSeek-AI, Liu, et al. 2024) で採用）：専門化した方が言語モデリング損失が下がる場合であっても $\nabla\mathcal{L}_{\text{aux}}$ はトークン表現を一様なルーティングへと押しやってしまうため、この手法では補助損失の勾配を用いずに負荷を分散させる（なお DeepSeek-V3 では、これに添えてごく小さな系列単位の負荷分散損失のみを併用している）。その代わりに、各エキスパートは微分不可能な選択バイアス b_e を保持し、これを Top-k 選択の**内部でのみ**加算する（$T(\mathbf{x}) = \text{argtop}_k(\mathbf{p}(\mathbf{x}) + \mathbf{b})$ となり、ゲーティング重み $g_e(\mathbf{x})$ にはバイアスがかからない）。このバイアスは各ステップ後に、更新率 η_b の符号コントローラによって次式で更新される。

$$b_e \leftarrow b_e + \eta_b \,\text{sign}\left(\frac{1}{E} - f_e\right). \tag{11.18}$$

11.7.3.3 負荷分散手法の比較

メソッド `balance(rate)` を `MoELayer` に追加し、2 ブロックのバイトレベル `GPT`（$E = 8, k = 1$ とし、各エキスパートを全幅の FFN とすることで 1 個のアクティブなエキスパートが 1 個の密な FFN と等価になる設定）を用いて、WikiText-103 上で 800 ステップにわたり 5 つの設定を比較する。(1) 初期化状態、(2) 負荷分散なし、(3) 対照実験としての**凍結ルータ**（崩壊にルータとエキスパート間のフィードバックが必要かどうかを検証）、(4) Switch 補助損失（$\alpha = 0.01$）、および (5) 損失不要のバイアス制御（$\eta_b = 10^{-2}$）である。

```
@d2l.add_to_class(MoELayer)
def balance(self, rate):
    """The controller: move each bias one step toward the even load, then
    count the next batch's assignments afresh."""
    load = self.load[...]
    self.expert_bias[...] += rate * jnp.sign(load.mean() - load)
    self.load[...] = jnp.zeros_like(load)

@nnx.jit  # aux_weight and bias_rate are traced: one compilation for all
def moe_train_step(model, optimizer, X, Y, aux_weight, bias_rate):
    def loss_fn(model):
        ce = optax.softmax_cross_entropy_with_integer_labels(
            model(X), Y).mean()
        aux = sum(l.aux_loss[...] for l in moe_layers(model))
        return ce + aux_weight * aux, ce
    (_, ce), grads = nnx.value_and_grad(loss_fn, has_aux=True)(model)
    optimizer.update(model, grads)
    for l in moe_layers(model):  # The controller: not a gradient step
        l.balance(bias_rate)
    return ce

def train_moe_lm(model, data, optimizer, num_steps, aux_weight=0.0,
                 bias_rate=0.0):
    """d2l.train_steps with a balancing loss and the bias controller."""
    for l in moe_layers(model):
        l.reset_stats()
    losses = []
    while len(losses) < num_steps:
        for X, Y in data.train_dataloader():
```

```
            losses.append(moe_train_step(model, optimizer, jnp.asarray(X),
                                         jnp.asarray(Y), aux_weight,
                                         bias_rate))
            if len(losses) == num_steps:
                return [float(l) for l in jax.device_get(losses)]
```

検証セットにおけるルーティングの均衡を調べるため、evaluate_moe は検証交差エントロピー、降順にソートしたエキスパートごとのトークン占有率、およびドロップされた割り当ての割合を記録する。一方、measure は設定された容量係数での検証損失を、上限を設けないバッファ（$S = nk$、ドロップ数 0）での値と比較する。

```
def evaluate_moe(model, data):
    """Validation loss; each expert layer's shares of the validation
    tokens, busiest expert first; and the fraction of assignments dropped."""
    layers = moe_layers(model)
    for l in layers:
        l.reset_stats()
    loss = math.log(d2l.lm_perplexity(model, data.val_dataloader()))
    shares = [sorted((l.load / l.load.sum()).tolist(), reverse=True)
              for l in layers]
    dropped = sum(float(l.dropped) for l in layers) / sum(
        float(l.load.sum()) for l in layers)
    return loss, shares, dropped

def measure(model, data):
    """The validation loss, the same with buffers that drop nothing, the
    busiest expert's load in units of the even share, and the fraction
    dropped; and the shares."""
    layers = moe_layers(model)
    val, shares, dropped = evaluate_moe(model, data)
    peak = max(l.num_experts * s[0] for l, s in zip(layers, shares))
    capacity = [l.capacity_factor for l in layers]
    for l in layers:  # S = n k slots: room for every assignment
        l.capacity_factor = l.num_experts
    val_all = evaluate_moe(model, data)[0]
    for l, c in zip(layers, capacity):
        l.capacity_factor = c
    return (val, val_all, peak, dropped), shares
```

次に、WikiText103 を読み込み、small_gpt を定義する。これは 2 ブロック・バイトレベルの GPT（幅 $d = 128$）であり、capacity_factor=None のときは密な FFN、それ以外のときは 8 エキスパート・Top-1 の MoELayer を使用する。あわせて、凍結ルータの対照アーム用に freeze_router も定義する。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)

def small_gpt(vocab_size, capacity_factor=None):
    """Two blocks of width 128; with a capacity factor, each feed-forward
    network becomes eight experts with one active per token."""
    experts = lambda rngs: MoELayer(128, 8, 1,
                                    capacity_factor=capacity_factor, rngs=rngs)
```

```
    return d2l.GPT(vocab_size, num_hiddens=128, num_heads=4, num_blks=2,
                   ffn_factory=None if capacity_factor is None else experts,
                   rngs=nnx.Rngs(d2l.get_key()))

def freeze_router(model):
    """Keep every router at its initial weights: a Variable that is not a
    Param gets no gradient and no update."""
    for l in moe_layers(model):
        l.router.kernel = nnx.Variable(l.router.kernel[...])
```

ドライバ関数 `balancing_run` は、容量係数 $\gamma = 1.25$ で初期化した新規の `small_gpt` を、指定された負荷分散設定のもとで `num_steps=800` ステップ訓練し、`measure` による集計結果を返す。

```
def balancing_run(data, num_steps=800, frozen=False, aux_weight=0.0,
                  bias_rate=0.0):
    """Measure a fresh small MoE GPT after num_steps steps of training with
    one balancing setting."""
    model = small_gpt(len(data.vocab), capacity_factor=1.25)
    if frozen:
        freeze_router(model)
    if num_steps:
        optimizer = d2l.LMOptimizer(model, lr=2e-3, num_steps=num_steps,
                                    warmup=100)
        train_moe_lm(model, data, optimizer, num_steps, aux_weight,
                     bias_rate)
    return measure(model, data)
```

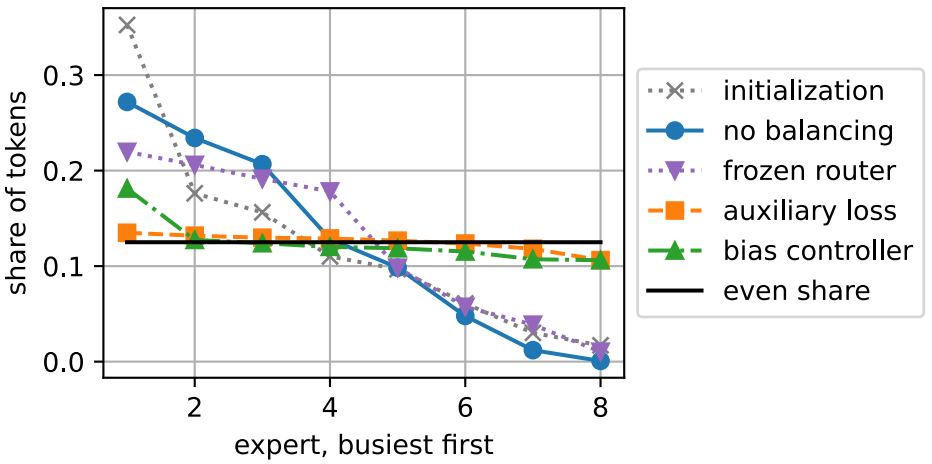
5 つの負荷分散アームのそれぞれについて独立な 2 つのシードを実行し、検証損失（バッファによるドロップあり・なしの両方）、最大エキスパート負荷、およびドロップ率を表にまとめる。

```
arms = {'initialization': dict(num_steps=0), 'no balancing': {},
        'frozen router': dict(frozen=True),
        'auxiliary loss': dict(aux_weight=0.01),
        'bias controller': dict(bias_rate=0.01)}
print(f'spaces: {(data.val == ord(" ")).mean():.1%} of the validation bytes')
print(f'{"two runs each":>15}  {"loss":^9}  {"no drops":^9}  '
      f'{"busiest":^9}  {"dropped":^7}')
shares = {}  # The first run's shares
for name, kwargs in arms.items():
    (a, shares[name]), (b, _) = [balancing_run(data, **kwargs)
                                 for _ in range(2)]
    print(f'{name:>15}  {a[0]:.2f} {b[0]:.2f}  {a[1]:.2f} {b[1]:.2f}  '
          f'{a[2]:3.1f}x {b[2]:3.1f}x  {a[3]:3.0%} {b[3]:3.0%}')
```

```
spaces: 18.9% of the validation bytes
  two runs each    loss      no drops    busiest   dropped
 initialization  5.49 5.53  5.49 5.53  3.4x 3.1x  22% 29%
   no balancing  1.71 1.76  1.66 1.69  2.4x 2.7x  24% 29%
  frozen router  1.74 1.78  1.68 1.72  1.8x 2.7x  17% 23%
 auxiliary loss  1.57 1.56  1.57 1.56  1.1x 1.2x   0%  0%
bias controller  1.60 1.56  1.59 1.55  1.8x 1.5x   3%  2%
```

両方の MoE 層にわたって平均し順位付けしたエキスパートのトークン占有率を、一様分布の基準線 $1/8 = 12.5\%$ と重ねてプロットすると、いずれの負荷分散機構もルーティング崩壊を防いでいる様子が視覚的にわかる。

```
ranks = list(range(1, 9))
mean_shares = [[sum(rank) / len(rank) for rank in zip(*s)]
               for s in shares.values()]  # Mean over the layers
d2l.plot(ranks, mean_shares + [[1 / 8] * 8], 'expert, busiest first',
         'share of tokens', legend=list(shares) + ['even share'],
         fmts=('C7x:', 'C0o-', 'C4v:', 'C1s--', 'C2^-.', 'k-'),
         legend_loc='outside')
```



負荷分散なし（no balancing）では、最も稼働率の高い上位 3 個のエキスパートがルーティング対象トークン全体の約 70% を占める一方で、最も稼働率の低いエキスパートはほとんどトークンを受け取らず、容量係数 $\gamma = 1.25$ のもとで全割り当ての 4 分の 1 から 3 分の 1 がドロップされる。ルータを凍結した場合（frozen router）は偏りがやや緩和されるものの、初期化直後の未訓練モデル（initialization）の時点ですでに負荷分散なしの場合と同程度の偏りがあるため、この規模では偏りの大部分はエキスパートの学習とルータの更新の間のフィードバックではなく入力分布に起因している。1 行あたり 2 回の実行ではこの 2 つの要因を切り分けることはできない（演習問題 3）。Switch 補助損失と損失不要のバイアスコントローラはいずれも、すべてのエキスパートを理想的な占有率 $1/E = 12.5\%$ の近くに保っている（最も負荷の高いエキスパートでもその 2 倍未満に収まり、$\gamma = 1.25$ でのドロップ率は高々数パーセントにとどまる）。

11.7.4 混合エキスパート言語モデル

11.7.4.1 アクティブパラメータ数を揃えた MoE GPT と密な GPT の比較

負荷分散された 8 エキスパート・Top-1 の MoE GPT（$N_{\text{active}} = 0.43\text{M}$、$N_{\text{total}} = 2.26\text{M}$）を、**アクティブパラメータ数とトークンあたり FLOPs が完全に同一**の密な GPT（$N_{\text{active}} = N_{\text{total}} = 0.43\text{M}$）と WikiText-103 上の 6,000 訓練ステップにわたって比較すると、スパースな条件付き容量がもたらす IsoFLOP 上の優位性が明らかになる。

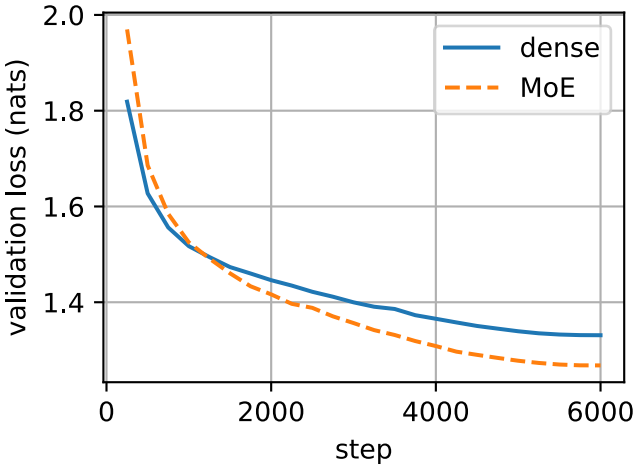
```
capacity = {'dense': None, 'MoE': 2}  # The dense model has no buffers
models = {name: small_gpt(len(data.vocab), capacity_factor=c)
          for name, c in capacity.items()}
for name, model in models.items():
    print(f'{name:>5}: {d2l.num_params(model) / 1e6:5.2f}M parameters, '
          f'{active_params(model) / 1e6:.2f}M active per token')
```

```
dense:  0.43M parameters, 0.43M active per token
  MoE:  2.26M parameters, 0.43M active per token
```

両モデルを独立に 2 回ずつ 6,000 ステップにわたり Switch 補助損失（$\alpha = 0.01$）のもとで訓練し、250 ステップごとに検証損失を評価することで、スパースモデルが密なベースラインをいつ追い抜くかを追跡する。

```
num_steps, every = 6000, 250
curves = {name: [] for name in capacity}  # Validation loss every 250 steps
for run in range(2):
    for name, c in capacity.items():  # Dense first, then MoE
        model = small_gpt(len(data.vocab), capacity_factor=c)
        optimizer = d2l.LMOptimizer(model, lr=2e-3, num_steps=num_steps,
                                    warmup=100)
        curve = []
        for _ in range(num_steps // every):  # Dense: no aux loss to add
            train_moe_lm(model, data, optimizer, every, aux_weight=0.01)
            curve.append(
                math.log(d2l.lm_perplexity(model, data.val_dataloader())))
        curves[name].append(curve)
    dense, moe = curves['dense'][-1], curves['MoE'][-1]
    ahead = next((every * (i + 1) for i, (m, d) in enumerate(zip(moe, dense))
                  if m < d), None)
    print(f'run {run + 1}: dense {dense[-1]:.3f}, MoE {moe[-1]:.3f}; '
          + (f'MoE ahead from step {ahead}, ' if ahead else 'MoE never ahead, ')
          + f'{evaluate_moe(model, data)[2]:.1%} dropped at the end')
d2l.plot(list(range(every, num_steps + 1, every)),
         [curves['dense'][0], curves['MoE'][0]], 'step',
         'validation loss (nats)', legend=['dense', 'MoE'], fmts=('-', '--'))
```

```
run 1: dense 1.331, MoE 1.268; MoE ahead from step 1250, 0.0% dropped at the end
run 2: dense 1.322, MoE 1.266; MoE ahead from step 1500, 0.0% dropped at the end
```



MoE モデルは序盤こそ密なモデルに遅れをとるものの、最初の 2,000 ステップ以内に追い抜き、同じトークンあたり FLOP コストでありながら検証交差エントロピーが約 0.05 nat 低い状態で訓練を終える。

11.7.4.2 公開モデルにおける細粒度エキスパートと共有エキスパート

近年の最先端 MoE モデル群（Table 11.4）は、Mixtral 8x7B の粗粒度な 8 エキスパート・Top-2 のレシピを次の 2 つの方向で洗練させている（Figure 11.16）。

1. **細粒度エキスパート分割**（fine-grained expert segmentation）：E 個の全幅エキスパートのそれぞれを m 個のエキスパート（幅 w/m）に分割し、各トークンを mk 個へと（全 mE 個の中から）ルーティングすると、保持パラメータ数とアクティブパラメータ数の両方を変えないまま、異なるアクティブなエキスパートの組み合わせ数を、たとえば $\binom{16}{2} = 120$ 通りから $\binom{64}{8} \approx 4.4 \times 10^9$ 通りへと拡大できる（$m = 4$ の場合）。公開モデルではさらに踏み込み、128〜256 個のエキスパートのうち 4〜8 個をアクティブにすることで、アクティブな割合も引き下げている。
2. **共有エキスパート** (Dai et al. 2024)：ルーティングを介さず**すべての**トークンに対して実行される幅の狭いエキスパートを $E_{\text{shared}} \geq 1$ 個専任させることで、共通の文法的変換や書式変換を共有エキスパートに吸収させ、ルーティングされるエキスパートが冗長性なく専門化できるようにする。

Table 11.4: 公開されている MoE 言語モデルにおけるルーティング対象エキスパートの構成。

モデル	層あたりエキスパート数（E）	アクティブ数 k	エキスパート幅 / d	共有エキスパート	保持 / アクティブパラメータ数
Mixtral 8x7B (Jiang et al. 2024)	8	2	3.5	なし	46.7B / 12.9B = 3.6
DeepSeek-V3 (DeepSeek-AI, Liu, et al. 2024)	256	8	0.29	1	671B / 37B = 18

モデル	層あたりエキスパート数（E）	アクティブ数 k	エキスパート幅 / d	共有エキスパート	保持 / アクティブパラメータ数
Qwen3-235B (A. Yang et al. 2025)	128	8	0.38	なし	235B / 22B = 11
gpt-oss-120b (OpenAI 2025)	128	4	1.0	なし	117B / 5.1B = 23

11.7.5 まとめ

混合エキスパート（MoE）層（Equation 11.16）は、密な FFN をルーティングされる E 個のエキスパート FFN に置き換え、トークンごとに上位 $k \ll E$ 個のエキスパートのみを活性化させることで、**保持パラメータ数** N_{total}（メモリ容量）を**アクティブパラメータ数** N_{active}（トークンあたり FLOPs）から切り離す。負荷分散を行わないとエキスパートの稼働率は大きく偏る。この偏りのかなりの部分は初期化時点ですでに存在しており、ルータとエキスパート間の正のフィードバックによって**ルーティング崩壊**へと増幅されうるが、**Switch 補助損失** $\mathcal{L}_{\text{aux}} = E \sum_e f_e \bar{p}_e$（Equation 11.17）と**損失不要の動的ルータバイアス** $b_e \leftarrow b_e + \eta_b \text{sign}(1/E - f_e)$（Equation 11.18）はいずれもエキスパートの負荷を $1/E$ 付近に保つ。アクティブパラメータ数と訓練 FLOPs を揃えた比較では、負荷分散された MoE 言語モデルの方が密な GPT よりも低い検証交差エントロピーに到達する。近年のオープンウェイト MoE アーキテクチャ（Table 11.4）はさらに、組み合わせによる専門化を広げるためにエキスパートを**細粒度**のバンク（$E \in [128, 256], k \in [4, 8]$）へと分割し、多くの場合そこに常時アクティブな**共有エキスパート**を併用している（Figure 11.16）。

11.7.6 演習問題

1. **$k = 1$ と $k > 1$ の再正規化におけるルータ勾配。**多くの MoE 層では、選択された確率をその和で再正規化する。すなわち $g_e(\mathbf{x}) = p_e(\mathbf{x}) / \sum_{j \in T(\mathbf{x})} p_j(\mathbf{x})$ とする。
    1. $k = 1$ のときにこれと同じ再正規化を適用すると、$g_{e^*}(\mathbf{x}) = p_e(\mathbf{x})/p_e(\mathbf{x}) \equiv 1$ が定数となるため $\partial \text{MoE}(\mathbf{x})/\partial \mathbf{W}_r = \mathbf{0}$ となり、ルータがタスク損失からの勾配を一切受け取らないことを示せ。
    2. 本節の `MoELayer` は、選択された各エキスパートを生の確率で重み付けすることで（$g_{e^*}(\mathbf{x}) = p_{e^*}(\mathbf{x})$ とする Switch Transformer (Fedus et al. 2022) と同様に）この問題を回避していることをコードで確認せよ。
2. **Switch 補助損失の最小値と勾配。**$\mathcal{L}_{\text{aux}} = E \sum_{e=1}^{E} f_e \bar{p}_e$（Equation 11.17）を考える。ただし、f_e は勾配停止された定数として扱い、$\sum_e \bar{p}_e = 1$ とする。
    1. 偏微分 $\partial \mathcal{L}_{\text{aux}} / \partial s_{t,e}$（トークン t のルータロジット $s_{t,e} = (\mathbf{x}_t \mathbf{W}_r)_e$ に関する微分）を計算し、これが 0 になるのはすべての経験的負荷が等しいとき、かつそのときに限ること（$f_1 = \cdots = f_E = 1/E$）を示せ。
    2. コーシー・シュワルツの不等式を用いて、$\bar{p}_e = f_e$ のときに $\mathcal{L}_{\text{aux}} \geq 1$ が成り立ち、等号成立は $f_e = 1/E$ のとき、かつそのときに限ることを証明せよ。また、その最大値 E（単一エキスパートへの完全な崩壊 $f_1 = \bar{p}_1 = 1$ のとき）を求めよ。
3. **正のフィードバックと入力分布の偏りの切り分け。**`balancing_run` において、`frozen router` アームと `no balancing` をステップ $100, 200, 400, 800$ で比較せよ。初期状態のモデルにおける最上位エキスパートの占有率（本節のバイトレベルモデルではすでに約 3 分の 1 に達している）を起点としたとき、判別に十分な数のシードにわたって評価すると、訓練可能な

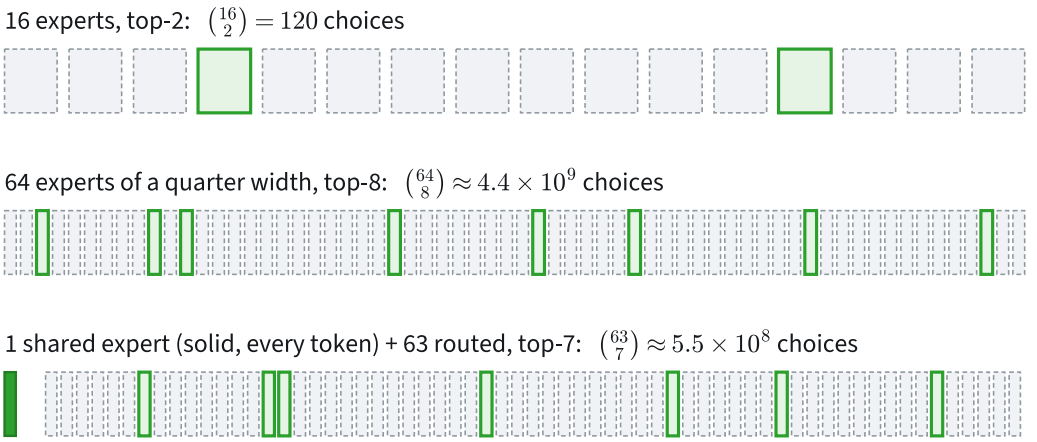


Figure 11.16: 同じエキスパートパラメータ予算を使う 3 通りの方法。各ボックスは 1 個のエキスパートを表し、その幅はサイズに比例する。強調表示されたエキスパートは、ある 1 トークンに対して選択されたものである。上段：16 個のエキスパートのうち、トークンは 2 個を使用する。中段：同じパラメータ数を 4 分の 1 幅の 64 個のエキスパートに分割し、トークンはそのうち 8 個を使用する。下段：幅の狭いエキスパートのうち 1 個をすべてのトークンが使用する共有エキスパートとし、各トークンは残りの 63 個から 7 個を選択する。3 つの行はいずれも保持パラメータ数とトークンあたりの使用パラメータ数が同一である。

ルータの最上位エキスパート占有率は凍結ルータよりもそこから大きく離れるか。また、すべてのエキスパートの出力射影 `W_2` を 0 で初期化し、初期ステップですべてのエキスパートが同一の出力を生成するようにすると何が起きるか。

4. **損失不要バイアスコントローラのステップサイズに関するダイナミクス。** `balancing_run` を `bias_rate` $\eta_b \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ で実行し、訓練ステップにわたる最上位エキスパートの占有率と検証損失をプロットせよ。コントローラのステップサイズが大きすぎる場合（$\eta_b = 10^{-1}$）、なぜエキスパートの負荷がステップごとに振動するのか。

5. **極小データ領域における MoE の過学習。** 6,000 ステップの `dense` 対 `MoE` の比較（11.7.4.1 節）を、ただし `data.train` を先頭の 65,536 バイト（64 KB）のみに制限し、モデルが同じテキストを何百回も反復するようにして再実行せよ。`dense` と `MoE` の両方について訓練損失と検証損失をプロットせよ。両者がトークンあたりで全く同じ FLOPs を使用しているにもかかわらず、なぜ 2.26M パラメータの MoE は小さなコーパスをはるかに速く暗記し、0.43M パラメータの密なモデルよりも早く過学習し始めるのか。

6. **固定予算下での共有エキスパートと細粒度化。** 常時アクティブな 1 個の共有エキスパート（幅 w）に加えて Top-k 個（ルーティングされる E 個の幅 w のエキスパートから選択）を実行する `SharedMoELayer` を実装し、保持およびアクティブなエキスパート幅を揃えた条件（ブロックあたり保持 16×170、アクティブ 2×170）のもとで、`small_gpt` 内の 3 つの FFN 構成を比較せよ。(a) 粗粒度 $E = 8, k = 1, w = 340$、(b) 中粒度 $E = 16, k = 2, w = 170$、(c) 細粒度と共有 $E_{\text{shared}} = 1, E = 15, k = 1, w = 170$ とし、800 ステップ後の検証損失を比較せよ。

11.8 関連文献

本章で構築してきたアーキテクチャは、共通の設計を備えている。すなわち、Pre-Norm の残差ストリーム（11.1 節）、因果または双方向のマルチヘッドアテンション（11.2 節、11.4 節）、ヘッド・トークン・チャネルにわたって圧縮されたキー・値キャッシュ（11.3 節）、そして密に（11.6 節）またはスパースなエキスパートルーティングを通じて（11.7 節）スケールされたフィードフォワード層である。本節では、これらの基盤の上に築かれた重要なアーキテクチャ上のバリエーションと研究の方向性を概観する。

11.8.1 正規化の配置と残差ストリームの派生形

Pre-Norm（各サブ層の前に `RMSNorm` を配置する方式）は、出力から最初の層まで減衰のない恒等経路を保つため、現代の大半の Transformer で標準となっている（11.1 節）。しかし、非常に深いモデルや長時間の FP16/BF16 訓練では、正規化されない残差ストリームが深さ方向にわたって大きな活性化値ノルムを蓄積しうるほか、アテンションスコアの大きさもソフトマックスが飽和するまで上方へドリフトしかねない。この問題に対処する改良には 2 つの系統がある（Figure 11.17）。

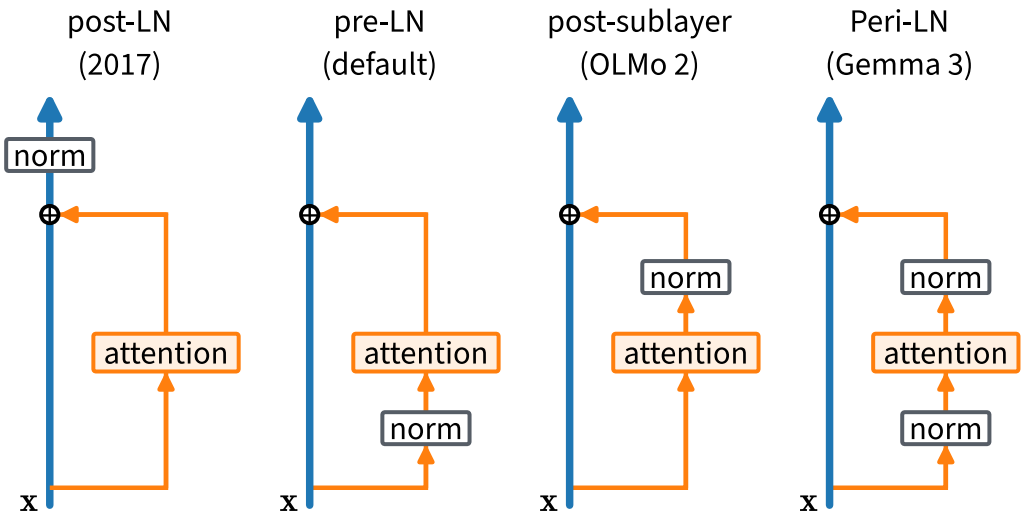


Figure 11.17: サブ層 f の前後に正規化を配置する 4 つの方法。Post-LN は残差ストリーム自体を正規化する。Pre-LN はサブ層の入力のみを正規化する。サブ層後正規化（OLMo 2）はストリームへ戻して加算する前にサブ層の出力のみを正規化する。Peri-LN（Sandwich-LN、Gemma 3）はサブ層の入力と出力の両方を正規化する。

- **サブ層出力の正規化（Sandwich-LN / Peri-LN）と Post-Norm の派生形：**各サブ層の入力と出力の両方を正規化する構成（$x + \mathrm{Norm}(f(\mathrm{Norm}(x)))$）は、恒等スキップ接続（skip connection）を損なわずに各残差更新の大きさを有界に保つものであり、CogView (M. Ding et al. 2021) や Gemma 2/3 (Gemma Team 2024, 2025) で採用されている。一方、OLMo 2 (Team OLMo et al. 2025) は RMSNorm を各サブ層の出力側へ移し（$x + \mathrm{Norm}(f(x))$）、これを QK-Norm と組み合わせることで、加算後の正規化によって恒等経路を遮ることなく残差ストリームの増大を有界に抑えている。
- **クエリ・キー正規化（QK-Norm）：**ヘッドごとにクエリとキーへ RMSNorm を適用してから $QK^\top/\sqrt{d_h}$ を計算すると（11.1.2.5 節）、残差ストリームの増大にかかわらず、コーシ

ー・シュワルツの不等式によってソフトマックス前のロジットが有界に保たれる (Henry et al. 2020; Dehghani et al. 2023)。この手法は ViT-22B、Gemma 3、Qwen3、OLMo 2 で標準となっている。

- **深さスケーリング付き Post-Norm とロジットのソフトキャッピング：**DeepNorm (H. Wang et al. 2024) は Post-LN ブロックを維持しつつ、明示的な深さ依存の定数によってスキップ経路の重みを大きくし（$\mathrm{Norm}(\alpha x + f(x))$、ただし $\alpha = (2L)^{1/4}$）、同時にブランチの重みの初期化を $\beta = (8L)^{-1/4}$ 倍に縮小する。この両方によってストリームに対する各更新が相対的に小さくなり、Post-LN Transformer を 1,000 層まで安定化できる。別の方法として、Gemma 2 は滑らかな tanh ソフトキャッピング（$c\tanh(z/c)$）によってアテンションロジットと最終ロジットを有界に抑えているが、より新しいモデルではアテンションロジットのソフトキャッピングはおおむね QK-Norm に取って代わられている。

11.8.2 フィードフォワードネットワークと活性化関数

元の Transformer は、中間幅 $d_{\text{ff}} = 4d$ をもつ 2 行列の ReLU フィードフォワードネットワークを用いていた (Vaswani et al. 2017)。GPT-2 と BERT は ReLU を GELU に置き換えた (Hendrycks and Gimpel 2016)。Shazeer (2020) 以降、現代のほぼすべての言語モデルはゲート付き線形ユニット（gated linear unit）(Dauphin et al. 2017)、すなわち **SwiGLU**（SiLU ゲート）または **GeGLU**（GELU ゲート）を採用しており、中間幅 $d_{\text{ff}} \approx \frac{8}{3}d$ をハードウェアに適した 128 または 256 の倍数に丸めることで、3 つの射影行列（$W_{\text{gate}}, W_{\text{up}}, W_{\text{down}}$）を $8d^2$ のパラメータ数・FLOP 予算（ゲートなしの幅 $4d$ の FFN と同量）に一致させている（11.1 節）。極端に大きな幅では、**並列 Transformer ブロック**（PaLM (Chowdhery et al. 2022)、ViT-22B (Dehghani et al. 2023)）が同じ正規化済み入力からアテンションと FFN を並列に計算するため（$x + \mathrm{Attn}(\mathrm{Norm}(x)) + \mathrm{FFN}(\mathrm{Norm}(x))$）、アテンションと FFN の入力射影を単一の大規模な行列積へと融合できる。

11.8.3 長文脈推論とキャッシュ圧縮

デコーディングはメモリ帯域幅律速であるため（11.3.2.1 節）、キー・値キャッシュの通信量を削減することが効率的なサービングの中心課題となる。

- **グループ化クエリアテンションとマルチヘッド潜在アテンション：**マルチクエリアテンション（MQA）(Shazeer 2019) とグループ化クエリアテンション（grouped-query attention、GQA）(Ainslie et al. 2023) は、クエリヘッドのグループ間でキーヘッドと値ヘッドを共有する（11.3.3 節）。DeepSeek-V2 (DeepSeek-AI 2024) で導入され DeepSeek-V3 (DeepSeek-AI, Liu, et al. 2024) でも引き継がれた**マルチヘッド潜在アテンション**（multi-head latent attention、MLA）は、低ランク KV 圧縮（11.3.3.2 節）を一般化した手法である。キーと値を結合的に単一の低次元潜在ベクトル $c_t^{\text{KV}} \in \mathbb{R}^{d_c}$（$d_c \ll Hd_h$）へ射影してメモリにキャッシュする一方、切り離された低次元の RoPE キーベクトルと対にすることで、デコーディング時の重み吸収（weight absorption）と回転位置エンコーディングの両立を保っている。
- **スライディングウィンドウ、アテンションシンク、ハイブリッドスタック：**直近トークンの移動ウィンドウとあわせて先頭のプロンプトトークンを保持する手法（**StreamingLLM** (Xiao et al. 2024)、11.3.3.3 節）は、ソフトマックスの分母が余剰の確率質量を先頭の「シンク」トークンで吸収するため、ストリーミング推論中も生成品質を維持できる。近年のいくつかのアーキテクチャでは、局所的なスライディングウィンドウアテンション層と完全な大域的因果アテンション層を交互に配置したり（Gemma 3 では大域層 1 層につき局所層 5 層、gpt-oss では 1 層おき）、アテンション層と線形時間の状態空間層・リカレント層を交互に配置したりしている（第 12 章）。
- **投機的デコーディング：**候補となる m 個のトークンを並列に検証する処理は、単一トークンをデコードする場合とほぼ同じメモリ帯域幅コストで実行できる（11.3.2.1 節）。そのた

め、**投機的デコーディング**（speculative decoding）(Leviathan et al. 2023; C. Chen et al. 2023) は高速なドラフトモデル（またはターゲットモデルに付加した追加のマルチトークン予測ヘッド）を用いて m 個のトークンを提案し、ターゲット分布を厳密に保つ棄却サンプリング（rejection sampling）規則によって、ターゲットモデルの 1 回の順伝播でそれらを検証する。

11.8.4 エンコーダ、クロスアテンション、ビジョンバックボーン

- **双方向エンコーダと検索：**オープンエンドな生成では因果デコーダが主流を占める一方、双方向エンコーダ（11.4 節）は依然として密なテキスト埋め込み、リランキング、およびトークンレベルの分類における主力である。RoBERTa (Liu et al. 2019) と DeBERTa (He et al. 2021) は BERT のマスク言語モデリング（masked language modeling）による事前学習レシピを洗練させ、ModernBERT (Warner et al. 2024) は現代的な Pre-Norm、RoPE、SwiGLU、およびパディングなしの FlashAttention スタックを双方向エンコーダへと導入している。
- **Vision Transformer と階層型ハイブリッド：**中程度のデータ規模で ViT を効果的に訓練するには、強力なデータ拡張と正則化（DeiT (Touvron, Cord, Douze, et al. 2021)）、自己教師ありのマスクパッチ予測（MAE (He et al. 2022)）または自己蒸留（self-distillation、DINOv2 (Oquab et al. 2024)）、および 2D 回転位置埋め込み（rotary position embedding）が鍵となる。マルチスケールの特徴マップを必要とする密な予測タスク（検出、セグメンテーション）に対しては、Swin Transformer (Liu et al. 2021) が階層型ピラミッドにわたってシフトされた局所 2D ウィンドウ内に自己アテンションを制限する一方、ConvNeXt (Z. Liu et al. 2022) は Transformer ブロックの設計上の選択を純粋なデプスワイズ分離可能畳み込みネットワークへと還元している。

11.8.5 スケーリング則と混合エキスパートの設計

- **計算量最適スケーリングと推論考慮型スケーリング：**@kaplan2020scaling は、言語モデルの交差エントロピー損失がモデルサイズ N、データセットサイズ D、および計算量 C に対してべき乗則に従ってスケールすることを初めて確立した。続いて Hoffmann et al. (2022)（Chinchilla）は IsoFLOP 曲線を用い、一定の事前学習計算量のもとで訓練損失が最小化されるのは、N と D を同じ比率でスケールさせたとき（$D \approx 20N$）であることを示した。現代のオープンウェイトモデル（15 兆トークンで訓練された Llama 3 8B や Qwen3 など）は、小さなモデルを Chinchilla 比（11.6 節）をはるかに超えてあえて過剰に訓練することで、運用期間全体の総推論コストに対して訓練計算量を償却している (Sardana et al. 2024)。
- **スパース混合エキスパート：**Top-K ノイジーゲーティング (Shazeer et al. 2017)、GShard (Lepikhin et al. 2021)、および Switch Transformer (Fedus et al. 2022) を基盤とする現代の MoE モデルは、主に粒度において異なっている（11.7 節）。Mixtral 8x7B (Jiang et al. 2024) が全幅のエキスパート 8 個のうち 2 個へ各トークンをルーティングするのに対し、Qwen3 (A. Yang et al. 2025) や DeepSeek-V3 (DeepSeek-AI, Liu, et al. 2024) は 100 個を超える幅の狭いエキスパートのうち 8 個を選択する。さらに DeepSeek-V3 は常時活性な共有エキスパート (Dai et al. 2024) を追加し、ごく小さな補助ペナルティだけを残して、主に動的なエキスパートごとのルータバイアス (L. Wang et al. 2024) によって負荷を分散させている。また、**スパースアップサイクリング**（sparse upcycling）(Komatsuzaki et al. 2023) によって密なチェックポイントを MoE モデルへ変換することも可能であり、訓練済みの密な FFN から各エキスパートを初期化したうえで事前学習を継続する。

Chapter 12

状態空間モデル

第 8 章のリカレントネットワークは、テキストを 1 トークンずつ順番に読み込み、それまでに読み込んだ内容を固定サイズの**状態**（state）に保持して引き継ぐ。したがって、100 万番目のトークンを生成するコストは、10 番目のトークンを生成するコストと変わらない。これに対し、第 11 章の Transformer はすべてのトークンを保持する。長いレポートに関する質問に答える際には、レポートのすべてのトークンについてキーと値を保持するため（11.3 節）、メモリもトークンあたりの計算量もコンテキスト長（context length）に応じて増大する。ところが 11.2 節では、そこで構築した GPT と同規模の単純なリカレントネットワークは、コンテキストの最初の数バイトを過ぎたところで遅れをとり、最後まで追いつくことはなかった。より優れたリカレントネットワークはその差を埋められるのだろうか。そして、増大するアテンションのコストはどの程度の長さから問題になり始めるのだろうか。

そこには 2 つの障壁が立ちはだかっている。古典的なリカレントネットワークは状態を非線形に更新するため、訓練を 1 ステップずつ進めざるを得ず、GPU の大部分をアイドル状態のまま遊ばせてしまう。また、固定サイズの状態は忘却を避けられないため、何を保持するかを選択しなければならない。その解決策は 3 つの方向から得られる。すなわち、1990 年代のゲート機構 (Hochreiter and Schmidhuber 1997; Gers et al. 2000)、制御理論における線形力学系（linear dynamical system）(Kalman 1960) を訓練可能な層として深層学習に導入した手法 (Gu et al. 2020; Gu, Goel, et al. 2022)、そしてソフトマックスを除いたアテンション (Katharopoulos et al. 2020) である。これら 3 つの流れは、並列に訓練でき、一種の線形アテンション（linear attention）を計算するリカレント層において合流する。本章で扱うすべてのリカレント層は、状態、各トークンを取り込む**更新**（update）、および出力を計算する**読み出し**（readout）を備えている。制御工学ではこのようなシステムを**状態空間モデル**（state space model）と呼ぶ。本章のタイトルではこの用語をこの広義の意味で用い、12.4 節では連続時間の線形システムという狭義の意味で用いる。

まず、何を保持するかをネットワークが学習できるようにするゲート機構と、単一の状態ベクトルが計測可能なボトルネックを生む 2014 年のエンコーダ・デコーダから始める。続いて 3 つの改良により、状態を高速かつ選択的（selective）なものにする。すなわち、すべての位置にわたって並列に評価できアテンションと実行時間を比較できる線形更新（12.3.4 節）、連続時間から離散化された複数時間スケールの減衰、そして各トークンがどれだけ書き込みどれだけ忘却するかを自ら決められるようにする入力依存のステップサイズである。さらに状態をベクトルから行列へと拡張すると、状態はキー・値ペアの連想メモリ（associative memory）となり、その記憶容量を計算できるほか、デルタ則（delta rule）によって書き込み規則を先鋭化できる。最後に、少数のフルアテンション層を挟み込むことで、いかなる固定サイズの状態も持ち得ない正確な想起（exact recall）を取り戻す。これは複数の実用言語モデルで採用されているハイブリッドな構成

パターンである (Lieber et al. 2024; Qwen Team 2025)。

初読の際には、12.2 節、12.4 節における連続時間の導出、および 12.6 節と 12.7 節のチャンク化アルゴリズムは後回しにしてよい。特定の記憶能力を 1 つずつ切り分けるよう設計された小規模な合成タスクによって各メカニズムを検証し、11.2 節のバイトレベルのテキストベンチマークによって各ミキサが言語上でどのような性能を示すかを確認する。

12.1 ゲート付きリカレントネットワーク

8.4.2 節の素のリカレントネットワークは、単語や句読点としては認識できる文字列を生成するものの、数語を超えると文法が崩れてしまう。「**the keys to the cabinet in the hall**」に続く動詞を予測する場面を考えよう。ここで「**is**」ではなく「**are**」を選ぶには、30 文字前にある「**keys**」が複数形であることを覚えている必要がある。素の RNN は毎ステップ、同じリカレント行列と tanh 非線形性を通して状態全体を上書きするため、順方向の信号も逆方向の勾配も数十ステップで急速に減衰する。リカレントネットワークは、読み取った情報を数百ステップにわたってどのように保持できるだろうか。

1990 年代後半に導入された解決策は、現在の入力と前の状態から計算される $(0, 1)$ の値をもつベクトルである**ゲート**を介して、状態のどの程度の割合を保持するかをユニットごと・ステップごとにネットワーク自身に決定させることである。ゲートが 1 の近くにとどまるユニットは、保持している内容と逆方向の勾配の両方をほぼ変化させずに次のステップへと伝える。以下では、古典的な 2 つのゲート付きアーキテクチャである長・短期記憶（long short-term memory、LSTM）とゲート付きリカレントユニット（gated recurrent unit、GRU）を構築し、合成データの遅延想起（delayed recall）および第 11 章のバイトレベル WikiText-103 ベンチマークで評価する。ゲートは長距離の記憶保持を解決する一方で、前の状態に依存するためステップごとの逐次的な訓練が依然として避けられない。このボトルネックは 12.3 節で取り除く。

本章を通じて、数式は列ベクトルを用いて単一の系列に対して記述し、コードでは行ベクトルを用いて系列のミニバッチを処理する。

12.1.1 ゲート

12.1.1.1 勾配消失

8.5.1 節で見たように、隠れ状態（hidden state）$\mathbf{h}_t \in \mathbb{R}^d$ をもつ素の RNN を通じて勾配がどのように伝播するかを振り返ろう。

$$\mathbf{h}_t = \tanh(\mathbf{W}_{\text{hh}}\mathbf{h}_{t-1} + \mathbf{W}_{\text{xh}}\mathbf{x}_t + \mathbf{b}_{\text{h}}). \tag{12.1}$$

$\mathbf{h}_{t-k}$ に対する摂動は、$\mathbf{h}_t$ に到達するまでに k 個のステップごとのヤコビ行列 $\mathbf{J}_s = \operatorname{diag}(1 - \mathbf{h}_s^2)\,\mathbf{W}_{\text{hh}}$ の積が掛かる。

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-k}} = \mathbf{J}_t \mathbf{J}_{t-1} \cdots \mathbf{J}_{t-k+1}. \tag{12.2}$$

単一のスカラーユニットでは、各因子は $w(1-h_s^2)$ であり、その大きさは高々 $|w|$ である。$|w| < 1$ のとき、この積は少なくとも $|w|^k$ の速さで縮小する（$w = 0.9$ では、30 ステップで約 4%、100 ステップでは $\le 0.003\%$ となる）。

行列 $\mathbf{W}_{\text{hh}}$ については、その最大特異値が 1 未満であれば、すべての因子があらゆるベクトルを縮小させるため、積は必ずゼロに収束する。多くのステップにわたると、行列の累乗は ρ^k のオー

ダーで変化する。ここで、**スペクトル半径**（spectral radius）ρ（固有値の絶対値の最大値）は $\rho \le \|\mathbf{W}_{\mathrm{hh}}\|_2$ を満たす。$\rho \approx 1$ であり、かつ状態が小さく保たれて tanh がほぼ線形であれば、主固有ベクトル方向の成分は多くのステップにわたって生き残り得る。この境界領域の挙動は、以下の実験で実際に観察する。$\rho \approx 1$ から離れると、勾配は爆発するか（これは勾配クリッピングで上限を抑えられる）、あるいは消失する（これは事後的な再スケーリングでは回復できない）。

12.1.1.2 加法的な記憶

ヤコビ行列が単位行列となる最も単純な漸化式は純粋な累積和 $c_t = c_{t-1} + x_t$ であり、このとき $\partial c_t / \partial c_{t-k} = 1$ があらゆるラグ k に対して成り立つ。しかし、重み付けのない和は決して忘却しない。過去のあらゆる入力が等しい重みで重ね合わされたまま残り、状態の分散は系列長とともに増大する。

12.1.1.3 学習される減衰としてのゲート

関連する入力を保持しつつ無関係な入力を捨てるため、ネットワークが各ステップで保持率 $f_t \in (0,1)$、および入力率 $i_t \in (0,1)$（新たな候補内容 $\tilde{c}_t$ に乗じる割合）を選択できるようにする。

$$c_t = f_t \, c_{t-1} + i_t \, \tilde{c}_t. \tag{12.3}$$

直接的な記憶経路に沿った k ステップの導関数は保持率の積 $\partial c_t / \partial c_{t-k} = \prod_{s=t-k+1}^{t} f_s$ となる。保持率が一定の f であるとき、記憶された値は f^k に従って減衰し、その**半減期**（half-life）は $\ln 2 / \ln(1/f)$ ステップとなる。具体的には、$f = 0.9$ で約 7 ステップ、$f = 0.99$ で 69 ステップ、$f = 0.999$ で 693 ステップである。$f_t \to 1$ とすれば値とその勾配は無期限に保持され、$f_t \to 0$ とすればセルは 1 ステップで消去される。

幅 d の層では、記憶はベクトル $\mathbf{c}_t \in \mathbb{R}^d$ となり、外部に公開される隠れ状態 $\mathbf{h}_t \in \mathbb{R}^d$ を伴う。各**ゲート**は $(0,1)^d$ のベクトルを出力し、これは現在の入力 $\mathbf{x}_t$ と前の出力 $\mathbf{h}_{t-1}$ を読み取るシグモイド層によって次のように計算される。

$$\mathbf{f}_t = \sigma\!\left(\mathbf{W}_{\mathrm{xf}} \mathbf{x}_t + \mathbf{W}_{\mathrm{hf}} \mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{f}}\right). \tag{12.4}$$

毎ステップ状態全体に密行列 $\mathbf{W}_{\mathrm{hh}}$ を掛けて tanh を適用する素の RNN とは異なり、ゲート付き記憶は各座標に固有の学習された係数 $f_{t,j} \in (0,1)$ を要素ごとに乗じて更新する。$f_{t,j} \le 1$ であるため、直接的な記憶経路に沿った勾配は爆発し得ず、さらに $f_{t,j}$ を 1 の近くに保てば消失も防げる。

実用上の細部が 2 つ残っている。第 1 に、$f_t \approx 1$ の間も書き込みが繰り返されると記憶された値 c_t が増大し得るため、LSTM は tanh を適用して $\mathbf{c}_t$ を $\mathbf{h}_t$ へと読み出す。第 2 に、ゼロバイアス初期化のもとではシグモイドゲートは $\sigma(0) = 1/2$（半減期 1 ステップ）の近くから始まるため、ゲート付きネットワークは長期の記憶を表現できるものの、デフォルトの初期状態では長期の記憶をもたない。

12.1.2 長・短期記憶

12.1.2.1 記憶セル

Hochreiter and Schmidhuber (1997) による**長・短期記憶**（LSTM）に Gers et al. (2000) の忘却ゲート（forget gate）を加えたアーキテクチャは、Equation 12.3 を中心にリカレント層を構成

する。この層は幅 d のベクトルを 2 つ保持する。1 つは要素ごとのゲート処理と加算のみによって時間発展する内部の**記憶セル**（memory cell）$\mathbf{c}_t$ であり、もう 1 つは外部に公開される**隠れ状態** $\mathbf{h}_t$ であり、後者は $\mathbf{c}_t$ を圧縮してゲートに通した表現を後続の層や次ステップのゲートに提供する。

セルへのアクセスは、**入力ゲート**（input gate）$\mathbf{i}_t$、**忘却ゲート** $\mathbf{f}_t$、**出力ゲート**（output gate）$\mathbf{o}_t$ という 3 つのシグモイドゲートが制御する。

$$\begin{aligned}\mathbf{i}_t &= \sigma(\mathbf{W}_{\text{xi}}\mathbf{x}_t + \mathbf{W}_{\text{hi}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{i}}),\\ \mathbf{f}_t &= \sigma(\mathbf{W}_{\text{xf}}\mathbf{x}_t + \mathbf{W}_{\text{hf}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{f}}),\\ \mathbf{o}_t &= \sigma(\mathbf{W}_{\text{xo}}\mathbf{x}_t + \mathbf{W}_{\text{ho}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{o}}).\end{aligned} \tag{12.5}$$

tanh 活性化関数をもつ 4 つ目の射影は、候補となる**入力ノード**（input node）$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{\text{xc}}\mathbf{x}_t + \mathbf{W}_{\text{hc}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{c}})$ を提案する。続いて、セルと隠れ状態は要素ごとの積（$\odot$）を用いて次のように更新される。

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \qquad \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \tag{12.6}$$

$\mathbf{c}_t$ を $\mathbf{h}_t$ から分離することで、ユニットは必要になるまで情報を外部に出さずに保持できる。たとえば、$c_{t-1} = 0.8$ に対して $f_t = 0.95$、$i_t = 0.1$、$\tilde{c}_t = -0.5$、$o_t = 0.2$ であれば、セルは $c_t = 0.71$ に保たれる一方、外部には $h_t = 0.2\tanh(0.71) = 0.12$ しか出力されない（Figure 12.1）。

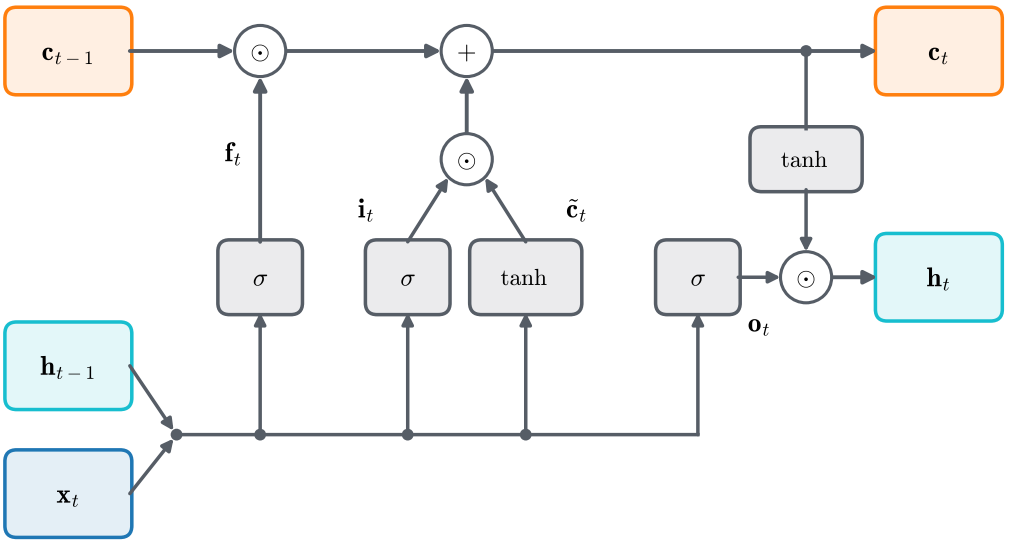


Figure 12.1: LSTM の記憶セル。前の出力 $\mathbf{h}_{t-1}$ と入力 $\mathbf{x}_t$ が、3 つのシグモイドゲートと 1 つの tanh 入力ノードからなる 4 つの層に入力される。セル $\mathbf{c}_t$ は忘却ゲートでスケールされつつ加法的に更新され、ネットワークの残りの部分が何を受け取るかは出力ゲートが制御する。

ゲートを固定したものとして扱うと、セルからセルへの直接的なヤコビ行列 $\partial\mathbf{c}_t/\partial\mathbf{c}_{t-1}$ は単に $\text{diag}(\mathbf{f}_t)$ となる（Figure 12.1 の上部の経路）。完全な導関数には $\mathbf{h}_{t-1}$ とゲートを介した間接項も含まれるが、セルの経路は $\mathbf{f}_t$ が 1 の近くにとどまる限り減衰のないハイウェイを提供する。

12.1.2.2 実装

$\mathbf{i}_t$、$\mathbf{f}_t$、$\tilde{\mathbf{c}}_t$、$\mathbf{o}_t$ はいずれも $(\mathbf{x}_t, \mathbf{h}_{t-1})$ を入力とするため、それぞれの重みを 2 つの行列 $\mathbf{W}_{\text{x}} \in \mathbb{R}^{d_{\text{in}} \times 4d}$ と $\mathbf{W}_{\text{h}} \in \mathbb{R}^{d \times 4d}$ に束ね、入力ごとに 1 回の行列積を計算してから結果を 4 つに分割する。

```
class LSTMScratch(nnx.Module):
    """An LSTM layer computed one time step at a time."""
    def __init__(self, num_inputs, num_hiddens, *, rngs):
        self.num_hiddens = num_hiddens
        # Weights of i, f, c_tilde, and o, stacked along the last axis, with
        # variance 1 / fan-in
        self.W_x = nnx.Param(rngs.params.normal((num_inputs, 4 * num_hiddens))
                             / math.sqrt(num_inputs))
        self.W_h = nnx.Param(rngs.params.normal((num_hiddens, 4 * num_hiddens))
                             / math.sqrt(num_hiddens))
        self.b = nnx.Param(jnp.zeros(4 * num_hiddens))

    def __call__(self, X):  # X: (batch_size, num_steps, num_inputs)
        def step(state, x):
            h, c = state
            i, f, c_tilde, o = jnp.split(
                x @ self.W_x[...] + h @ self.W_h[...] + self.b[...], 4, -1)
            c = jax.nn.sigmoid(f) * c + jax.nn.sigmoid(i) * jnp.tanh(c_tilde)
            h = jax.nn.sigmoid(o) * jnp.tanh(c)
            return (h, c), h
        h = c = jnp.zeros((X.shape[0], self.num_hiddens))
        # lax.scan steps through the leading axis, so time goes first
        _, outputs = jax.lax.scan(step, (h, c), jnp.swapaxes(X, 0, 1))
        return jnp.swapaxes(outputs, 0, 1)  # (batch_size, num_steps, ...)
```

LSTMScratch がフレームワークの LSTM 層とまったく同じ漸化式を計算することを検証するため、ライブラリの重みを自作クラスにコピーし、ランダムなバッチに対する出力を比較する。

ライブラリの層は nnx.LSTMCell である。

```
lstm_cell = nnx.LSTMCell(16, 32, rngs=nnx.Rngs(0))  # A layer per gate
lstm = nnx.RNN(lstm_cell)
cell = LSTMScratch(16, 32, rngs=nnx.Rngs(1))
gates = [(lstm_cell.ii, lstm_cell.hi), (lstm_cell.if_, lstm_cell.hf),
         (lstm_cell.ig, lstm_cell.hg), (lstm_cell.io, lstm_cell.ho)]
cell.W_x[...] = jnp.concatenate([x.kernel[...] for x, _ in gates], 1)
cell.W_h[...] = jnp.concatenate([h.kernel[...] for _, h in gates], 1)
cell.b[...] = jnp.concatenate([h.bias[...] for _, h in gates])
X = jax.random.normal(jax.random.key(0), (4, 9, 16))
print(f'max difference: {jnp.abs(cell(X) - lstm(X)).max():.1e}')
```

```
max difference: 0.0e+00
```

出力は float32 の丸め誤差（$\sim 10^{-7}$）の範囲で一致するため、以降はより高速な融合済みのライブラリ層を使用する。

12.1.3 ゲート付きリカレントユニット

ゲート付き層には、本当に 2 つの状態ベクトルと 3 つのゲートが必要なのだろうか。Cho, Van Merriënboer, Gulcehre, et al. (2014) の**ゲート付きリカレントユニット**（GRU）は、セルと隠れ状態を単一のベクトル $\mathbf{h}_t$ に統合し、わずか 2 つのゲート（**リセットゲート** $\mathbf{r}_t$ と**更新ゲート**（update gate）$\mathbf{z}_t$）で制御しながら、多くの系列タスクで LSTM に匹敵する性能を達成する (Chung et al. 2014)。

$$\mathbf{r}_t = \sigma(\mathbf{W}_{\text{xr}}\mathbf{x}_t + \mathbf{W}_{\text{hr}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{r}}), \qquad \mathbf{z}_t = \sigma(\mathbf{W}_{\text{xz}}\mathbf{x}_t + \mathbf{W}_{\text{hz}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{z}}). \tag{12.7}$$

リセットゲートは、$\mathbf{h}_{t-1}$ のうちどれだけを新しい**候補状態**（candidate state）の計算に通すかをフィルタリングする。

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}_{\text{xh}}\mathbf{x}_t + \mathbf{W}_{\text{hh}}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_{\text{h}}\right). \tag{12.8}$$

続いて、更新ゲート $\mathbf{z}_t$ が古い状態と候補状態の間を凸補間する。

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t. \tag{12.9}$$

Equation 12.9 は、忘却ゲートと入力ゲートを連動させた（$f_t = z_t, i_t = 1 - z_t$）ゲート付き和 Equation 12.3 にほかならない。各座標は $[-1, 1]$ 内の 2 つの値を凸結合するため、$\mathbf{h}_t$ は $[-1, 1]$ 内に有界に保たれ、追加の出力 tanh を必要としない（Figure 12.2）。また、GRU 層が必要とするリカレント・入力重みは $6d^2$ 個であり、LSTM の $8d^2$ 個よりも少ない。（ライブラリの融合版 GRU では、すべてのリカレント行列積を 1 回の呼び出しにバッチ化できるよう、$\mathbf{r}_t$ を $\mathbf{W}_{\text{hh}}$ との積の**後**に適用する。演習問題を参照のこと。）

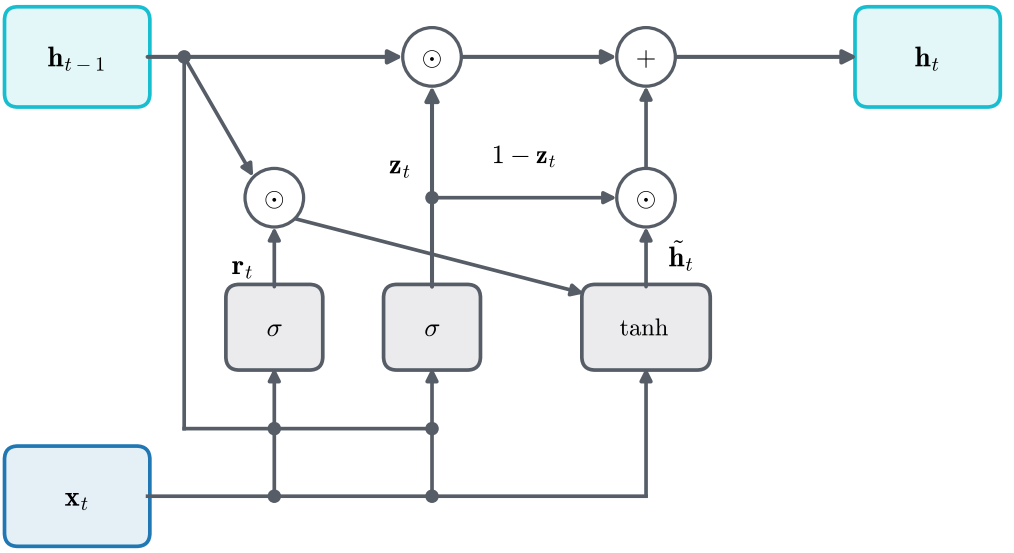


Figure 12.2: GRU セル。リセットゲートは候補 $\tilde{\mathbf{h}}_t$ の内部で前の状態をフィルタリングし、更新ゲートは古い状態と候補を混合する。

12.1.4 ゲート付きネットワークの訓練

12.1.4.1 遅延想起

ゲート付きの数式は数百ステップ持続する記憶を**可能にする**が、勾配降下法は実際にそれを見つけ出せるのだろうか。記憶の保持能力を合成データの**遅延想起**タスクで直接検証しよう。系列は8種類のターゲット記号（0〜7）のいずれか1つから始まり、その後に別の8種類の記号（8〜15）からランダムに選ばれた $T-1$ 個のフィラートークンが続き、ステップ T においてモデルは最初の記号を分類しなければならない（ランダム予測の正解率は $1/8 = 0.125$）。

```
def delayed_recall(rng, batch_size, num_steps, num_symbols=8,
                   blank_fillers=False):
    """A symbol, then num_steps - 1 fillers; the target is the symbol."""
    Y = rng.integers(0, num_symbols, (batch_size, 1))  # One query, at the end
    X = rng.integers(num_symbols, 2 * num_symbols, (batch_size, num_steps))
    if blank_fillers:  # The same filler token at every step
        X[:] = num_symbols
    X[:, :1] = Y
    return X, Y

X, Y = delayed_recall(np.random.default_rng(0), 2, 6)
print(X, Y)
```

```
[[ 6 10 10  8  8  8]
 [ 5 14 13 15 12 12]] [[6]
 [5]]
```

評価モデル（`RecallModel`）はトークンを32次元に埋め込み、幅64の単一のリカレント層に通して、最終ステップで線形分類器を適用する。本章を通じて、線形層・畳み込み層・リカレント層のすべての重み行列はLeCun正規初期化（分散 $1/n_{\text{in}}$）とゼロバイアスで初期化する（4.4節）。

Flaxの線形層とリカレント入力射影はデフォルトでLeCun正規初期化を用いる一方、リカレントカーネルはデフォルトで直交行列を用いるため、そこでもLeCun正規初期化を使うよう`recurrent_kernel_init`を渡す。

ここでは、6つのリカレント構成を3組のペアとして比較する。

1. **素の RNN**：LeCun正規初期化のリカレント重み（$\sigma = 1/\sqrt{64} = 1/8$）を用いるものと、対照群として $[-1/8, 1/8]$ 上の一様重み（分散はLeCun正規初期化の $1/3$）を用いるものである。
2. **デフォルトの LSTM と GRU**：ゲートバイアスをゼロとするため、すべてのゲートが $\sigma(0) = 1/2$（半減期1ステップ）の近くから始まる。
3. **保持ゲート付き LSTM と GRU（`init_keep_gate`）**：LSTMの忘却ゲートバイアスとGRUの更新ゲートバイアスを $+5$（$\sigma(5) = 0.993$、半減期 ≈ 100 ステップ）に初期化し、LSTMの入力ゲートバイアスを -5（$\sigma(-5) = 1 - \sigma(5) = 0.007$）に設定する。$b_{\text{i}} = -b_{\text{f}}$ と対にすることで、長いフィラー列にわたってLSTMセルの大きさが $\sigma(-5)/(1-\sigma(5)) = 1$ 以下に抑えられ、出力のtanhが決して飽和しなくなる。これはGRUに組み込まれた $1 - \mathbf{z}_t$ の連動に対応する（Jozefowicz et al. (2015) および Tallec and Ollivier (2018) の**クロノ初期化**（chrono initialization）を参照）。

```
class RecallModel(nnx.Module):
    """Embedding, a sequence layer, and a classifier at every position."""
    def __init__(self, layer, num_hiddens, vocab_size=16, num_inputs=32,
```

```
                num_outputs=8, *, rngs):
        # d2l.Embed computes its gradient by a matrix product: no atomic adds
        # that wait on one another when a few filler tokens fill every row
        self.embedding = d2l.Embed(vocab_size, num_inputs, rngs=rngs,
            embedding_init=nnx.initializers.normal(num_inputs ** -0.5))
        self.layer = layer
        self.head = nnx.Linear(num_hiddens, num_outputs, rngs=rngs)

    def __call__(self, X):  # Logits, (batch_size, num_steps, num_outputs)
        return self.head(self.layer(self.embedding(X)))

def init_keep_gate(cell, bias=5.0):
    """Start an LSTM's forget gate, or a GRU's update gate, near one."""
    if isinstance(cell, nnx.LSTMCell):
        cell.hf.bias[...] = jnp.full_like(cell.hf.bias[...], bias)
        cell.hi.bias[...] = jnp.full_like(cell.hi.bias[...], -bias)  # Input
    else:  # GRUCell: input biases ordered r, z, n
        h = cell.dense_i.bias.shape[0] // 3
        cell.dense_i.bias[...] = cell.dense_i.bias[...].at[h:2 * h].set(bias)
    return cell
```

デフォルトのゼロバイアスをもつゲート付きネットワークについて、単純な算術が何を予測するかに注目しよう。すべてのゲートが $1/2$ から始まるとき、ステップ 1 とステップ T を結ぶ直接経路は順逆両方向で $2^{-(T-1)}$ 倍に減衰する。これは約 10^{-6}（$T = 20$ のときで、float32 でもまだ解像可能）であるが、10^{-15}（$T = 50$ のとき）にまで落ち込む。ステップ T の状態は、直近のステップからの $O(1)$ の寄与に最初の記号の寄与を足し合わせるため、float32 の相対精度である $\sim 10^{-7}$ を下回る成分は読み出し層に届く前に丸め誤差で消えてしまい、記号の埋め込みへ戻る勾配も同様に小さくなる。したがって、デフォルトの LSTM と GRU は $T = 20$ では成功し $T = 50$ までには失敗する一方、保持ゲート版は $T = 200$ までのすべての遅延で成功すると予想される。

訓練用のヘルパー関数である `train_queries`、`query_accuracy`、`train_delayed_recall` を定義して保存し、本章のすべての合成想起タスクで再利用する。

```
@nnx.jit
def query_step(model, optimizer, X, Y):
    """One optimizer step on the cross-entropy at the last Y.shape[1] steps."""
    def loss_fn(model):
        logits = model(X)[:, -Y.shape[1]:]  # Predictions at the queries
        return optax.softmax_cross_entropy_with_integer_labels(
            logits, Y).mean()
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    optimizer.update(model, grads)
    return loss

def train_queries(model, batch, num_steps, optimizer):
    """Train on fresh batches; the queries are the last Y.shape[1] steps."""
    step = nnx.cached_partial(query_step, model, optimizer)  # Compile once
    for i in range(num_steps):
        step(*batch(i))
```

```
@nnx.jit
def query_accuracy(model, X, Y):
    """Fraction of queries answered correctly."""
    return (model(X)[:, -Y.shape[1]:].argmax(-1) == Y).mean()

def train_delayed_recall(model, num_steps, blank_fillers=False):
    """Train with Adam at rate 0.003 on 1,500 fresh batches of 128 sequences
    of delayed recall; return the accuracy on 2,048 new sequences."""
    task = lambda rng, n: delayed_recall(rng, n, num_steps,
                                         blank_fillers=blank_fillers)
    rng = np.random.default_rng(0)
    optimizer = nnx.Optimizer(model, optax.chain(
        optax.clip_by_global_norm(1.0), optax.adam(3e-3)), wrt=nnx.Param)
    train_queries(model, lambda step: task(rng, 128), 1500, optimizer)
    return float(query_accuracy(model, *task(np.random.default_rng(1), 2048)))
```

訓練の前に、64×64 の素の RNN における 2 通りの初期化について、スペクトル半径と最大特異値も確認しておこう。

```
rng, n = np.random.default_rng(0), 64  # n hidden units
for name, W in (('LeCun normal', rng.normal(0, n ** -0.5, (n, n))),
                ('default scale', rng.uniform(-n ** -0.5, n ** -0.5, (n, n)))):
    print(f'{name:>13}: spectral radius {np.abs(np.linalg.eigvals(W)).max():.2f},'
          f' largest singular value {np.linalg.norm(W, 2):.2f}')
```

```
 LeCun normal: spectral radius 1.02, largest singular value 2.01
default scale: spectral radius 0.62, largest singular value 1.10
```

LeCun 正規初期化の行列はスペクトル半径が 1.02 であり（状態が小さいときに主固有モードが消失も爆発もしない $\rho \approx 1$ の境界線上にちょうど位置する）、一方で一様分布によるデフォルトスケールの行列はスペクトル半径が 0.62 である（1 ステップあたり ~ 0.6 倍に状態を縮小させる）。それでは、全 6 モデルのそれぞれについて 2 つの乱数シードを用い、遅延 $T \in \{20, 50, 100, 200\}$ にわたって訓練する。

```
lecun = {'recurrent_kernel_init': nnx.initializers.lecun_normal()}
default = {'recurrent_kernel_init': nnx.initializers.variance_scaling(
    1 / 3, 'fan_in', 'uniform')}  # Uniform on [-1/8, 1/8]
cells = {'RNN': lambda rngs: nnx.SimpleCell(32, 64, rngs=rngs, **lecun),
         'RNN, default scale':
             lambda rngs: nnx.SimpleCell(32, 64, rngs=rngs, **default),
         'LSTM': lambda rngs: nnx.LSTMCell(32, 64, rngs=rngs, **lecun),
         'GRU': lambda rngs: nnx.GRUCell(32, 64, rngs=rngs, **lecun),
         'LSTM, keep gate': lambda rngs: init_keep_gate(
             nnx.LSTMCell(32, 64, rngs=rngs, **lecun)),
         'GRU, keep gate': lambda rngs: init_keep_gate(
             nnx.GRUCell(32, 64, rngs=rngs, **lecun))}

def recall_model(cell):  # Fresh random weights for every model
    rngs = nnx.Rngs(d2l.get_key())
    return RecallModel(nnx.RNN(cell(rngs), unroll=8), 64, rngs=rngs)
```

```
delays = [20, 50, 100, 200]
print(' ' * 18, ' '.join(f'{T:>9d}' for T in delays))
for name, cell in cells.items():
    runs = [[train_delayed_recall(recall_model(cell), T) for _ in range(2)]
            for T in delays]
    print(f'{name:18s}', ' '.join(f'{a:.2f}/{b:.2f}' for a, b in runs))
```

```
                        20        50       100       200
RNN                1.00/1.00 0.25/1.00 0.52/1.00 0.12/0.23
RNN, default scale 0.24/1.00 0.12/0.72 0.12/0.12 0.11/0.12
LSTM               0.49/1.00 0.12/0.12 0.13/0.13 0.12/0.12
GRU                1.00/1.00 0.12/0.12 0.13/0.13 0.13/0.12
LSTM, keep gate    1.00/1.00 1.00/1.00 1.00/1.00 1.00/1.00
GRU, keep gate     1.00/1.00 1.00/1.00 1.00/1.00 1.00/1.00
```

この結果は、GRU および $T \geq 50$ のすべての遅延については上の解析を裏付ける一方、LSTM については解析を修正するものとなっている。

- **デフォルト対保持ゲート付きの LSTM と GRU**：ゲートバイアスがゼロのとき、GRU は $T = 20$ で成功し、両セルともランダム予測の水準（0.12）へと $T \geq 50$ で崩壊する。しかしデフォルトの LSTM は、算術的な見積もりでは成功するはずだった $T = 20$ の時点ですでに不安定である。あの見積もりは忘却ゲートしか考慮していなかった。原因を完全には切り分けていないが、おそらく LSTM では記号が半分だけ開いた入力ゲートを通ってセルに入り、さらに半分だけ開いた出力ゲートと tanh を通って外へ出る必要があるため、$T = 20$ が訓練で回復できる限界付近になってしまうのだろう。`init_keep_gate` を用いれば、両者とも 1.00 の正解率を $T = 200$ までのすべての遅延・すべての試行で達成する。
- **境界線上と縮小領域の素の RNN**：LeCun 正規初期化の RNN はスペクトル半径 $\rho \approx 1$ から始まるため、シード間のばらつきは大きいものの一部のシードでは驚くほど遠く（$T = 100$ まで）記号を運ぶ。これに対し、$\rho \approx 0.62$ のデフォルトスケール RNN は $T = 50$ でも成否が安定せず、$T = 100$ 以降はすべてのシードで失敗する。適切にバイアスを初期化したゲートは、運任せのスペクトル保持を信頼性の高いユニットごとの記憶制御へと変える。

12.1.4.2 テキストにおけるゲート付きネットワーク

11.2.3.2 節では、幅の広い単層の素の RNN はコンテキストの最初の数バイトを過ぎると複数ブロックの GPT に遅れをとり、そのまま追いつけなかった。再帰構造そのものの違いを残差接続の深さや位置ごとの MLP と混同せずに系列ミキサを対等な条件で比較するため、本章ではすべてのミキサを 11.1 節に合わせた同一の Pre-Norm 残差バックボーン（`SeqLM`）の内部に配置する。具体的には、トークン埋め込みに続き、`H = H + mixer(norm(H))` とその後の 2 層 GELU MLP `H = H + mlp(H)` を各ブロックで適用する積層構造をとり、最後に層正規化と線形出力ヘッドを置く（Figure 12.3）。

```
class SeqLM(nnx.Module):
    """A language model of residual blocks: a sequence mixer, then an MLP."""
    def __init__(self, vocab_size, num_hiddens, mixers, *, rngs):
        d = num_hiddens
        self.embedding = d2l.Embed(vocab_size, d, rngs=rngs,  # Unit length
            embedding_init=nnx.initializers.normal(d ** -0.5))
        self.mixers = nnx.List(mixers)
        norm = lambda: nnx.LayerNorm(d, epsilon=1e-5, rngs=rngs)
        self.norms = nnx.List([norm() for _ in mixers])
```

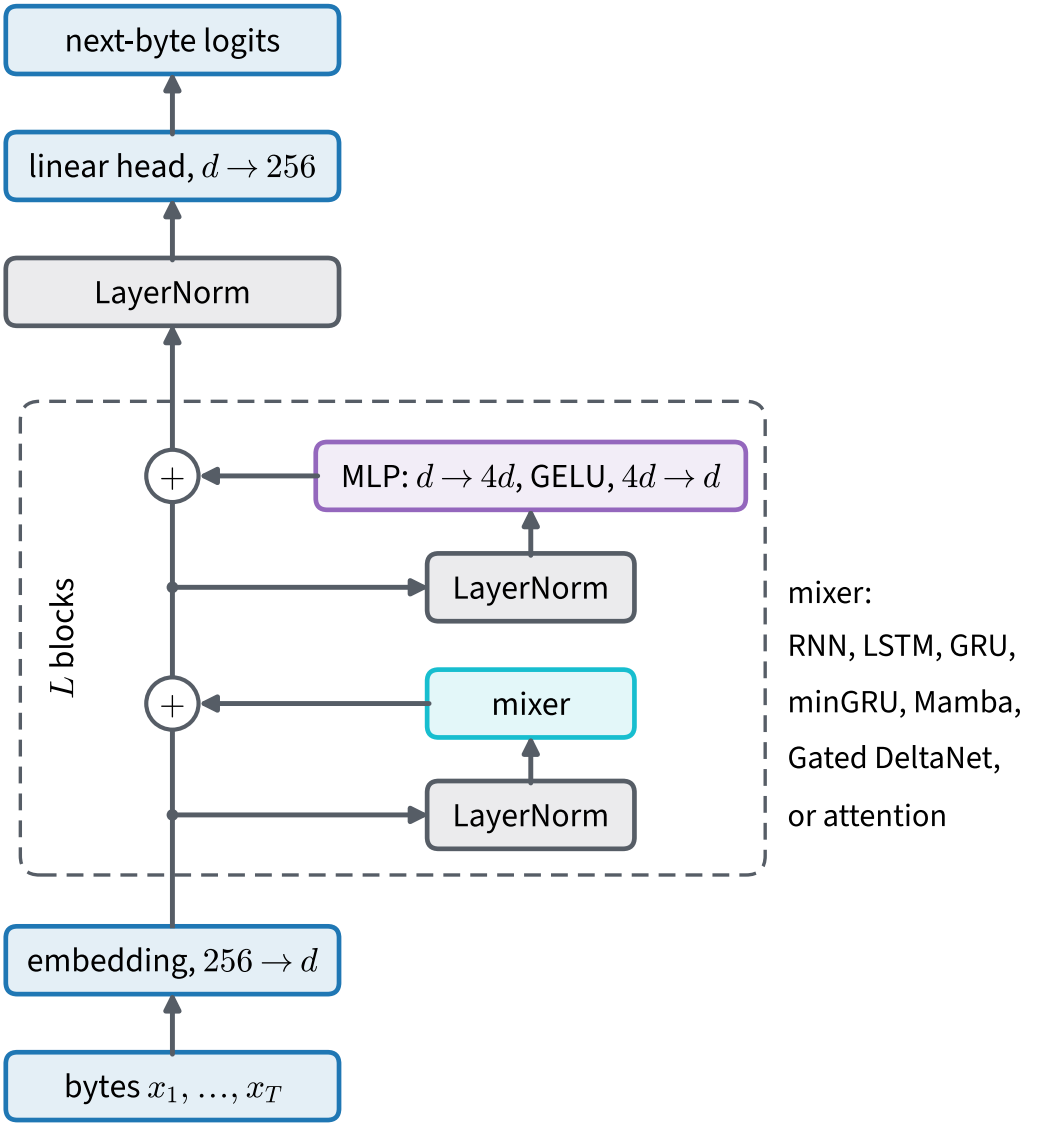


Figure 12.3: 共通バックボーン `SeqLM`。各ブロックは、それぞれ独自の層正規化を経たミキサの出力、続いて MLP の出力を残差ストリームに加算する。本章のモデル間で変わるのはミキサのみであり、言語モデルは $L = 4$ 個のブロックを積み重ねる。

```
        self.mlps = nnx.List([nnx.Sequential(
            norm(), nnx.Linear(d, 4 * d, rngs=rngs),
            lambda x: nnx.gelu(x, approximate=False),  # Exact GELU
            nnx.Linear(4 * d, d, rngs=rngs)) for _ in mixers])
        self.norm = norm()
        self.head = nnx.Linear(d, vocab_size, rngs=rngs)

    def __call__(self, X):
        H = self.embedding(X)  # (batch_size, num_steps, num_hiddens)
        for norm, mixer, mlp in zip(self.norms, self.mixers, self.mlps):
            H = H + mixer(norm(H))
            H = H + mlp(H)
        return self.head(self.norm(H))  # (batch_size, num_steps, vocab_size)
```

SeqLM 内のアテンションベースラインは AttentionMixer であり、これは RoPE 付きの因果マルチヘッドアテンション（11.2 節、10.4 節）、LeCun 正規初期化、およびオプションとして $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ に対するデプスワイズ因果畳み込み（後の 12.8 節で使用する）をまとめたものである。

```
class AttentionMixer(d2l.CausalAttention):
    """Causal multi-head attention, with rotary positions unless rope is
    False, as a sequence mixer; with conv_width, a causal convolution per
    channel first mixes each query, key, and value with those of the
    previous steps."""
    def __init__(self, num_hiddens, num_heads, conv_width=None, rope=True,
                 *, rngs):
        super().__init__(num_hiddens, num_heads, rope=rope, rngs=rngs)
        lecun = nnx.initializers.lecun_normal()
        for layer in (self.W_qkv, self.W_o):  # Like every mixer of the chapter
            layer.kernel[...] = lecun(rngs.params(), layer.kernel.shape)
        d = 3 * num_hiddens  # Channels of q, k, and v; one filter each
        self.conv = None if conv_width is None else nnx.Conv(
            d, d, (conv_width,), feature_group_count=d,
            padding=((conv_width - 1, 0),), rngs=rngs)

    def qkv(self, X, offset=0, tables=None):
        B, T, _ = X.shape
        QKV = self.W_qkv(X)
        if self.conv is not None:  # Causal: padded on the left only
            QKV = self.conv(QKV)
        q, k, v = (u.reshape(B, T, self.num_heads, -1).swapaxes(1, 2)
                   for u in jnp.split(QKV, 3, -1))
        if self.rope:
            q, k = (d2l.rope(u, offset, tables=tables) for u in (q, k))
        return q, k, v  # Each (batch_size, num_heads, num_steps, head width)
```

また、本章全体で使用する 3 つの評価用ヘルパー関数も保存しておく。まず、loss_by_position はコンテキストウィンドウの各位置における検証データの平均交差エントロピー（単位は nat）を計算する。

```
@nnx.jit
def position_loss_sum(model, X, Y):
```

```
    """Loss in nats at each position, summed over the batch."""
    return optax.softmax_cross_entropy_with_integer_labels(model(X), Y).sum(0)

def loss_by_position(model, data):
    """Mean validation loss in nats at each position of the window."""
    total, count = 0, 0
    for X, Y in data.val_dataloader():
        total += position_loss_sum(model, jnp.asarray(X), jnp.asarray(Y))
        count += len(X)
    return (total / count).tolist()
```

次に、`train_text` は AdamW でモデルを 1,000 ステップ訓練し（100 ステップかけて 2×10^{-3} までウォームアップしたのちコサイン減衰を適用）、`text_table` は各構成につき 2 つのシードを実行して、パラメータ数、全体および位置帯域別（`0-15`、`16-63`、`64-127`）の検証損失、ならびに 1 ステップあたりのミリ秒数を出力する。

```
def train_text(model, data, lr=2e-3, num_steps=1000):
    """Train a language model with LMOptimizer; return its validation loss
    in nats at each position and its milliseconds per training step."""
    optimizer = d2l.LMOptimizer(model, lr=lr, num_steps=num_steps, warmup=100)
    # One untimed call of 50 steps warms up (JAX compiles), the rest is timed
    bench = d2l.Benchmark(lambda: d2l.train_steps(model, data, optimizer, 50),
                          warmup=1, repeats=num_steps // 50 - 1)
    return loss_by_position(model, data), 1000 * bench.time / 50

def text_table(make_model, arms, data, num_runs=2, num_steps=1000):
    """Train num_runs fresh models make_model(*args) for each arm (a name
    and its args) for num_steps steps; print the parameters, the validation loss of every run,
    its losses at early, middle, and late positions of the window, and the
    milliseconds per step. Return the trained models of each arm."""
    T = data.num_steps
    bands = [(0, T // 8), (T // 8, T // 2), (T // 2, T)]
    width = max(len(name) for name in arms)
    header = ['validation'] + [f'{a}-{b - 1}' for a, b in bands]
    print(' ' * (width + 12), *(f'{h:>11}' for h in header), '  ms/step')
    trained = {}
    for name, args in arms.items():
        models = [make_model(*args) for _ in range(num_runs)]
        runs = [train_text(model, data, num_steps=num_steps)  # (losses, ms)
                for model in models]
        columns = [[np.mean(loss) for loss, _ in runs]] + [
            [np.mean(loss[a:b]) for loss, _ in runs] for a, b in bands]
        print(f'{name:{width}s} {d2l.num_params(models[0]):11,d}',
              *('/'.join(f'{v:.3f}' for v in column) for column in columns),
              '/'.join(f'{ms:.0f}' for _, ms in runs))
        trained[name] = models
    return trained
```

ここで、リカレントミキサとアテンションミキサをバイトレベルの WikiText-103（`d2l.WikiText103`、ウィンドウ長 128 バイト、バッチサイズ 64）でベンチマークする。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
```

各 `text_model` は幅 $d = 256$ の残差ブロックを 4 つ積み重ね、素の `RNN`、`LSTM`、`GRU`、8 ヘッドの因果 `attention` に加え、パラメータ数を揃えるために幅を $d = 302$ に広げた素の RNN（パラメータ数 3.81M で、GRU の積層モデルと同数）を比較する。

```
def text_model(mixer, d):  # 256 byte values; four blocks of width d
    rngs = nnx.Rngs(d2l.get_key())
    return SeqLM(256, d, [mixer(d, rngs) for _ in range(4)], rngs=rngs)

rnn = lambda d, rngs: nnx.RNN(nnx.SimpleCell(d, d, rngs=rngs, **lecun),
                              unroll=8)
lstm = lambda d, rngs: nnx.RNN(nnx.LSTMCell(d, d, rngs=rngs, **lecun),
                               unroll=8)
gru = lambda d, rngs: nnx.RNN(nnx.GRUCell(d, d, rngs=rngs, **lecun),
                              unroll=8)
attention = lambda d, rngs: AttentionMixer(d, num_heads=8, rngs=rngs)
arms = {'RNN': (rnn, 256), 'LSTM': (lstm, 256), 'GRU': (gru, 256),
        'attention': (attention, 256),
        'RNN, d=302': (rnn, 302)}  # As many parameters as the GRU
```

`text_table` を呼び出すと、各アームにつき 2 回ずつ訓練を実行し、横並びの比較表を出力する。

```
trained = text_table(text_model, arms, data)
```

```
                      validation        0-15        16-63       64-127   ms/step
RNN          2,763,520 1.347/1.346 1.511/1.511 1.323/1.323 1.323/1.323 11/12
LSTM         4,339,456 1.296/1.296 1.492/1.490 1.270/1.268 1.267/1.267 25/26
GRU          3,814,144 1.298/1.300 1.490/1.491 1.273/1.274 1.269/1.271 16/16
attention    3,286,784 1.387/1.386 1.562/1.562 1.363/1.363 1.362/1.360 9/9
RNN, d=302   3,815,724 1.332/1.337 1.499/1.504 1.308/1.313 1.308/1.313 16/14
```

この表からは 3 つの知見が浮かび上がる。

1. **残差積層＋ゲート機構：**幅の狭い 4 つの tanh RNN 層を Pre-Norm 残差＋ MLP バックボーンの内部に配置するだけで ~ 1.35 nat/byte に到達し、11.2 節の 6 ブロック SwiGLU GPT にすでに肩を並べる。素の RNN ミキサを LSTM や GRU に置き換えると検証損失はさらに 0.05 nat/byte 下がって ~ 1.30 になるのに対し、素の RNN の幅を広げて GRU とパラメータ数を揃えても（`RNN, d=302`）、その差の 1/3 未満しか縮まらない（約 1.33）。
2. **短いコンテキストの活用：**この 1,000 ステップ・128 バイトの設定では、どのモデルでも損失は帯域 `0-15` から `16-63` にかけて急減し、16 バイト目以降の変化は 0.005 未満にとどまる。ここでゲートが優位に立っているのは、100 ステップ先の記憶を運んでいるからではなく、短距離のバイト文脈をより効果的に管理しているからである。この小さな訓練予算では、アテンションの積層モデル（約 1.38）は最初の 16 バイトの時点ですでにリカレントミキサに遅れをとっている。
3. **ステップあたりのコスト：**融合されたライブラリのループを用いているにもかかわらず、LSTM と GRU は訓練 1 ステップあたり素の RNN より $1.4\times$〜$3\times$ 長い時間を要する。原因は各ステップ内部の計算量にあり、素の RNN が 1 回の積で済むところで 3〜4 回のゲート積を計算するためである。素の RNN も同じように前ステップの隠れ状態 $\mathbf{h}_{t-1}$ を待つため、逐次評価そのものが原因ではない。128 ステップではその逐次依存はまだ支配的なコストになっておらず、この表からは読み取れない。そのコストを切り分けるのが 12.3.4 節の

系列長スイープであり、そこでは 512 トークンから 16,384 トークンの間に GRU のステップ時間が 15 倍以上に増大する。

12.1.5 まとめ

素のリカレントネットワークでは、信号と勾配は単一の共有行列 $\mathbf{W}_{\text{hh}}$ に支配されるステップごとのヤコビ行列の積を通過するため、そのスペクトル半径が 1 の近くにあるときに限り長いホライズンにわたって生き残る。ゲート付きリカレントネットワークは、学習されるシグモイドゲートで制御される要素ごとの加法的な記憶経路 $c_t = f_t c_{t-1} + i_t \tilde{c}_t$ を導入する。LSTM は（入力ゲート、忘却ゲート、出力ゲートに守られた）内部の記憶セル $\mathbf{c}_t$ を公開状態 $\mathbf{h}_t$ から分離する一方、GRU は単一の状態ベクトル上で忘却ゲートと入力ゲートを連動させ（$1 - \mathbf{z}_t$）、リセットゲート $\mathbf{r}_t$ によって再帰状態をフィルタリングする。合成データの遅延想起では、忘却ゲート・更新ゲートを 1 の近くに初期化すること（`init_keep_gate`）が決定的に重要であり、$T \geq 50$ でのランダム予測水準の正解率を 100% の想起正解率（$T = 200$）へと変える。残差バックボーン（`SeqLM`）内部でのバイトレベル WikiText-103 では、LSTM ミキサと GRU ミキサは同一幅およびパラメータ数を揃えた素の RNN の両方を 0.03〜0.05 nat/byte 上回るが、ゲート内部で $\mathbf{h}_{t-1}$ に依存するため、訓練時には依然として T 回の逐次ステップを要する。

12.1.6 演習問題

1. **値の保持**。ステップ s の文字がずっと後のステップ t における予測を決定しており、1 つの LSTM ユニットがその情報を s から t まで運ぶものとする。
   1. Equation 12.5 におけるそのユニットの各ゲートが、ステップ s、s と t の真に間の各ステップ、およびステップ t においてとるべき値を答えよ。次に、Equation 12.9 における GRU ユニットの更新ゲートについても同様に答えよ。
   2. シグモイドゲートは決して 1 に到達しない。忘却ゲートの活性化前の値が定数 b に保たれ、毎ステップその値が $\sigma(b)$ であるとする。直接経路に沿って k ステップ後に生き残る記憶値の割合はいくらか。また、1,000 ステップ後に値の 90% が生き残るためには、b はどの程度大きくなければならないか。
   3. 重みがゼロのとき、忘却ゲートバイアスを $\log(T-1)$ とすると初期の忘却ゲートが $1 - 1/T$ となり、したがって大きい T に対して記憶された値が約 e 分の 1 に T ステップで減衰することを示せ。Tallec and Ollivier (2018) の**クロノ初期化**は、各ユニットの忘却ゲートバイアスを $\log u$（ただし u は $[1, T_{\max} - 1]$ 上の一様分布に従う）として抽出し、$\mathbf{b}_{\text{i}} = -\mathbf{b}_{\text{f}}$ と設定する。重みがゼロのとき、これにより $\mathbf{f}_t + \mathbf{i}_t = 1$ となることを示し、得られるセル更新式を Equation 12.9 と比較せよ。
2. **出力の圧縮**。入力ノード $\tilde{\mathbf{c}}_t$ はすでに tanh で圧縮されているにもかかわらず、Equation 12.6 は出力ゲートの前で tanh を $\mathbf{c}_t$ に再び適用している。
   1. $\mathbf{c}_0 = \mathbf{0}$ から始めたとき、$\mathbf{c}_t$ のすべての要素の絶対値が t より小さいことを示せ。さらに、すべての忘却ゲートの値が高々 $f_{\max} < 1$ であり、すべての入力ゲートの値が高々 $i_{\max}$ であるならば、すべての要素の絶対値が $i_{\max}/(1 - f_{\max})$ 未満にとどまる（t がいくらであっても成り立つ）ことを示せ。
   2. 仮に出力が $\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t$ であったとしよう。多数のステップにわたって値が蓄積された後、次ステップにおける各ゲートの活性化前の値はどうなるか。また、それはゲートを通過する勾配にどのような影響を与えるか。
3. **リセットの前後**。深層学習フレームワークの融合版 GRU 層は、リカレント行列積の後にリセットゲートを適用する。バイアス項の配置を除けば、候補を次の式で計算する。

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{\text{xh}}\mathbf{x}_t + \mathbf{r}_t \odot (\mathbf{W}_{\text{hh}}\mathbf{h}_{t-1}) + \mathbf{b}_{\text{h}})$$

これを Equation 12.8 の代わりに用いる。

1. 2 つの候補が任意の $\mathbf{r}_t$ と $\mathbf{h}_{t-1}$ に対して一致するのは、$\mathbf{W}_{\text{hh}}$ が対角行列であるとき、かつそのときに限ることを示せ。一般の $\mathbf{W}_{\text{hh}}$ に対して、それでも両者を一致させるリセットゲートの値は何か。
2. 全ステップの入力積 $\mathbf{W}_{\text{x}}.\mathbf{x}_t$ は、再帰が始まる前に 1 つの大きな積としてまとめて計算できる。それぞれの候補の形式のもとで、前の状態を含む行列積を各ステップで逐次的に何回実行しなければならないか。なぜ融合 GRU カーネルは上の形式を採用するのか。

4. [code] **スクラッチからの GRU**。クラス `GRUScratch` を `LSTMScratch` の編集によって実装せよ。具体的には、Equation 12.7 と Equation 12.8 に従う 3 つのヘッド、更新式 Equation 12.9、および $\mathbf{h}_t$ のみからなる状態をもたせよ。
   1. LSTM で行ったのと同様に、ライブラリの GRU 層の重みを自作の層にコピーして出力を比較せよ。両者は一致しないはずである。前の演習問題を利用して自作の層を 1 行変更し、出力が一致するようにせよ。
   2. $\mathbf{r}_t = \mathbf{1}$ に固定してリセットゲートを除去した層と、別の層として $\mathbf{z}_t = \mathbf{0}$ に固定して更新ゲートを除去した層を作成せよ。訓練の前に、どちらの除去が遅延想起をより大きく損なうかを Equation 12.9 から予測せよ。続いて 12.1.4.1 節の実験を実行し、予測を確かめよ。
5. [code] **セルの他の経路**。●入力が 1 つ、隠れユニットが 1 つの LSTM を考え、Equation 12.5 と Equation 12.6 のすべての量がスカラーであるとする。忘却ゲート、入力ゲート、入力ノードのリカレント重みをそれぞれ w_{hf}、w_{hi}、w_{hc} と書く。
   1. 出力ゲート o_{t-1} は h_{t-2} に依存するが、c_{t-1} には依存しない。$h_{t-1} = o_{t-1}\tanh(c_{t-1})$ を用いて、全微分 $\mathrm{d}c_t/\mathrm{d}c_{t-1}$ を導出せよ。これを直接経路の値 f_t と補正項の和として書き、さらに補正項をステップ t 由来の項と導関数 $\mathrm{d}h_{t-1}/\mathrm{d}c_{t-1}$ の積に因数分解せよ。
   2. ゲート、$\tilde{c}_t$、および c_{t-1} がどのような値のときにこの補正項はゼロになるか。セルが値を保持している最中に、ネットワークがゲートを通じて設定できるのはこれらの条件のうちどれか。
   3. LSTM が成功する遅延長で遅延想起タスクを訓練せよ。1 つのテスト系列について、セル $\mathbf{c}_{T-k}$ に関する損失の勾配のノルムをすべての k に対して自動微分により計算し、それを直接経路のみが運ぶ勾配（$\mathbf{c}_T$ における勾配に、直近 k ステップの忘却ゲート値の積を要素ごとに掛けたもの）と比較せよ。2 つの曲線はどこで乖離し、どちらが大きくなるか。
6. [code] **双方向のリーク**。●リカレント層は系列を逆向きに読み取ることもできる。**双方向**ネットワーク (Schuster and Paliwal 1997) は、最後のステップから最初のステップへ向かって 2 つ目の層を走らせ、2 つの層の出力を連結することで、各位置の表現が系列全体に依存するようにする。これは、文中の単語へのタグ付けや翻訳のための文の符号化のように、入力全体があらかじめ与えられている場合には常に有効である。しかし、言語モデルとなると話は別である。
   1. 反転させた系列に対して 2 つ目の LSTM を走らせ、その出力を再反転して順方向 LSTM の出力と連結し、結果を幅 d に射影して戻す双方向ミキサを構築せよ。12.1.4.2 節の LSTM をこのミキサに置き換えた `SeqLM` を `train_text` で訓練し、その検証損失を順方向モデルの検証損失と比較せよ。得られた数値を説明せよ。逆方向層は位置 t でどの入力を読み取っているか。また、後続のバイトがまだ存在しない生成時には、このモデルは何ができるだろうか。
   2. バイト x_t のリークのない双方向予測器は、ステップ $t-1$ 後の順方向状態とステップ $t+1$ 後の逆方向状態を結合することで、ウィンドウ内の他のすべてのバイトが与えられたもとでの x_t の確率をモデル化する。2 つの 2 値トークンからなる系列を用いて、これらの条件付き確率の t にわたる積を全系列について足し合わせても 1 になるとは限らないことを示せ。これは、その積を用いて系列全体にスコアを付けたり、パープレキシティを報告したりすることに対して何を意味するか。

12.2 系列対系列学習

前章までの言語モデルは、テキストを読み取って次のトークンを予測することで単一の系列を継続するものであった。しかし多くのタスクでは、ある系列を長さの異なる別の系列へと写像し、両者の位置は 1 対 1 に対応しない。たとえば、翻訳器が英語の文を読み取ってフランス語の文を書き出したり、音声認識器が音声クリップを単語に書き起こしたりする場合がそうである。翻訳は、2014 年にニューラルネットワークが初めて統計的フレーズベースシステムに肩を並べたベンチマークであった (Sutskever et al. 2014; Cho, Van Merriënboer, Gulcehre, et al. 2014)。そのアーキテクチャは単純であり、1 つ目のリカレントネットワークがソースを読み取り、2 つ目がターゲットを書き出し、両者の間では単一の固定幅の状態ベクトルが受け渡される。

この設計の後継にあたるクロスアテンション（第 10 章）や Transformer エンコーダ・デコーダ（11.4 節）はすでに登場したが、これらはその単一ベクトルのボトルネックを回避するために考案されたものである。では、なぜここで 2014 年の設計に立ち返るのか。そのボトルネックこそが本章の中心テーマだからである。次節以降で扱うすべてのリカレントモデルは、過去の情報を固定サイズの状態へと圧縮する。本節では、エンコーダ・デコーダの抽象化と標準的な語彙ベースの翻訳評価指標（chrF および BLEU）を整備したうえで、文字列反転ベンチマークを用い、入力が長くなるにつれて単一の固定状態ベクトルがどれだけの情報を保持できるかを直接測定する。

```
import collections
```

12.2.1 エンコーダ・デコーダの抽象化

ある系列を可変長の別の系列へと写像する**系列対系列学習**（sequence-to-sequence learning）、すなわち**系列変換**（sequence transduction）は、自然に 2 つのモジュールへと分かれる（Figure 12.4）。ソース系列を受け取って 1 つの状態に要約する**エンコーダ**と、その状態を受け取ってターゲットを 1 トークンずつ生成する**デコーダ**である。デコーダがソースについて得る情報はすべて、その状態を通過しなければならない。

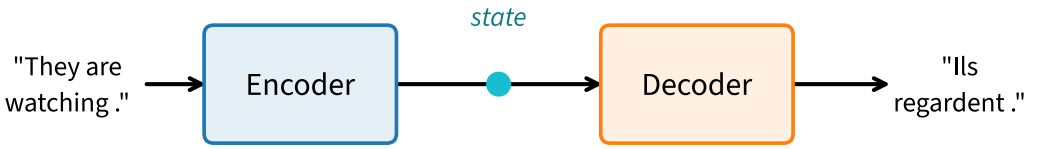


Figure 12.4: エンコーダ・デコーダアーキテクチャ。エンコーダがソース系列を読み取って状態に変換し、デコーダがその状態からターゲット系列を生成する。

ソース $x_1, \ldots, x_T$ とターゲット $y_1, \ldots, y_{T'}$（ただし T' は T と等しくなくてよい）が与えられたとき、デコーダは言語モデル（8.3 節）と同様に条件付き確率を自己回帰的に分解するが、すべての因子がエンコーダの状態 $\mathbf{c}$ で条件付けられる点が異なる。

$$P(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t'=1}^{T'} P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c}). \tag{12.10}$$

右辺からはソース系列が消え、$\mathbf{c}$ だけが残っている点に注目しよう。この等式が成り立つのは、$\mathbf{c}$ がデコーダに必要となるソースの情報をすべて保持している場合に限られる。訓練時には教師強制（8.4.1.2 節）のもとで正解の直前ターゲットトークンを入力し、次トークンの交差エントロピ

ーを最小化する。一方、推論時には、デコーダ自身が直前に出力したトークンを次の入力としてフィードバックし、系列終了トークン（end-of-sequence token）を生成するまでこれを繰り返す。

2014 年に 2 つの研究グループがこのリカレント設計を導入した。Cho, Van Merriënboer, Gulcehre, et al. (2014) は統計的翻訳システムの内部でフレーズ対のスコアを計算するために GRU エンコーダ・デコーダ（12.1.3 節）を用い、一方 Sutskever et al. (2014) は単一のベクトルで接続した深層 LSTM によって英語からフランス語への翻訳をエンドツーエンドで行った。特筆すべき工夫として、Sutskever らはソース文を**逆順**でエンコーダに入力し、翻訳の最初の数単語が対応するソース単語からわずか数リカレントステップの距離に置かれるようにした。これにより、状態が保持すべき総情報量は変わらないまま、勾配経路が短縮された。

リカレントなエンコーダとデコーダは後に Transformer（音声認識の Whisper (Radford et al. 2023) や視覚言語キャプション生成モデルなど）に取って代わられたが、エンコーダ・デコーダという抽象化そのものは生き残った。11.4 節で変わったのは、デコーダに渡される状態がソース全体に対する単一のベクトルではなく、ソーストークンごとに 1 つのベクトルを並べた系列になった点である。本節ではあえて 2014 年の単一ベクトルによる状態を維持する。その容量の限界こそが、ここで測定したい量だからである。

12.2.2 翻訳のスコアリング

人手による評価は遅すぎて訓練ループの内部では実行できないため、機械翻訳モデルは参照訳との語彙的な n グラムの重複によってスコア付けされる。2014 年の論文では BLEU が報告されていたが、現代の評価ベンチマークではその代わりに文字レベルの chrF が推奨されている (Popović 2015; Kocmi et al. 2021)。ここでは両方を実装し、18.4 節（英語版） でも再利用する。

chrF は、空白を除去した文字 n グラムを次数 $n = 1, \dots, 6$ にわたって比較するため、トークン化への依存を避けつつ、共通の語幹や惜しい語形変化にも部分的な評価を与えられる。各次数 n について、適合率（precision）は候補訳の n グラムのうち参照訳に含まれるものの割合であり、再現率（recall）は参照訳の n グラムのうち候補訳に含まれるものの割合である（多重集合の共通部分 `p & r` を用いることで、繰り返される n グラムは参照訳での出現回数でクリップされる）。6 つの次数にわたって適合率と再現率を平均して P と R を求め、chrF は次の F_β スコアを計算する。

$$F_\beta = \frac{(1+\beta^2)\,PR}{\beta^2 P + R}. \tag{12.11}$$

$\beta = 2$ とすると、適合率よりも再現率に大きな重みが置かれる。

```
def chrf(pred, label, n=6, beta=2):
    """chrF (Popovic, 2015): character n-gram F-score."""
    def ngrams(s, k):
        s = s.replace(' ', '')
        return collections.Counter(s[i:i + k] for i in range(len(s) - k + 1))
    prec = rec = 0.0
    for k in range(1, n + 1):
        p, r = ngrams(pred, k), ngrams(label, k)
        overlap = sum((p & r).values())
        if sum(p.values()) and sum(r.values()):
            prec += overlap / sum(p.values())
            rec += overlap / sum(r.values())
    prec, rec = prec / n, rec / n
```

```
    if prec + rec == 0:
        return 0.0
    return (1 + beta**2) * prec * rec / (beta**2 * prec + rec)
```

BLEU (Papineni et al. 2002) は、クリップされた単語 n グラム適合率 p_n（$n = 1, \ldots, k$）を測定し、短く適合率の高い断片がスコアを不正に稼げないよう、それらの重み付き幾何平均に**短文ペナルティ**（brevity penalty）を掛ける。

$$\text{BLEU} = \exp\left(\min\left(0, 1 - \frac{\text{len}_{\text{label}}}{\text{len}_{\text{pred}}}\right)\right) \prod_{n=1}^{k} p_n^{1/2^n}. \tag{12.12}$$

この式は積の形をとるため、適合率 $p_n = 0$ となる次数 $n \leq k$ が 1 つでもあると、文レベルの BLEU は 0 になる。（標準的なコーパスレベルの BLEU では、次数 $n \leq 4$ に $1/4$ ずつ均等な重みを付け、テストセット全体でカウントを集計するため、カウントがゼロになることはまれである。）

```
def bleu(pred_seq, label_seq, k):
    """Compute the BLEU."""
    pred_tokens, label_tokens = pred_seq.split(' '), label_seq.split(' ')
    len_pred, len_label = len(pred_tokens), len(label_tokens)
    score = math.exp(min(0, 1 - len_label / len_pred))
    for n in range(1, min(k, len_pred) + 1):
        num_matches, label_subs = 0, collections.defaultdict(int)
        for i in range(len_label - n + 1):
            label_subs[' '.join(label_tokens[i: i + n])] += 1
        for i in range(len_pred - n + 1):
            if label_subs[' '.join(pred_tokens[i: i + n])] > 0:
                num_matches += 1
                label_subs[' '.join(pred_tokens[i: i + n])] -= 1
        score *= math.pow(num_matches / (len_pred - n + 1), math.pow(0.5, n))
    return score
```

単一の文でこれら 2 つの評価指標がどのように食い違うかを見るため、完全なコピー、6 単語中 3 番目の単語の語形変化を 1 つだけ誤ったニアミス、および妥当な言い換えのスコアを計算してみよう。6 単語の文に含まれるすべての 4 グラムは 3 番目の単語を含むため、ニアミスでは $p_4 = 0$ となり、$k = 4$ の BLEU はゼロに落ち込む。一方、chrF が失うのは余分な **s** に重なる文字 n グラムだけである。

```
reference = 'we will visit our grandparents tomorrow'
candidates = {'copy': reference,
              'near miss': 'we will visits our grandparents tomorrow',
              'paraphrase': 'we are going to see our grandparents tomorrow'}
for name, candidate in candidates.items():
    print(f'{name:10s}  chrF {chrf(candidate, reference):.2f}  '
          f'BLEU {bleu(candidate, reference, k=4):.2f}')
```

```
copy        chrF 1.00  BLEU 1.00
near miss   chrF 0.91  BLEU 0.00
paraphrase  chrF 0.67  BLEU 0.00
```

ニアミスのスコアは chrF で 0.91、BLEU で 0.00 となる一方、意味を保った言い換えのスコアは chrF で 0.67、BLEU で 0.00 となる。どちらの評価指標も意味ではなく表層的な重複を数えるため、現代の評価では chrF（または過去の研究との比較にはコーパスレベルの sacreBLEU (Post 2018)）を、COMET (Rei et al. 2020) のような学習済みのニューラル評価器や LLM ジャッジと組み合わせて用いる。

12.2.3 固定状態のボトルネック

単一の状態ベクトルには、入力系列のどれだけの情報が収まるだろうか。自然言語では文の長さと構文的な希少性が交絡するため、ここでは 11.4.3.1 節の人工的な文字列反転ベンチマークを使って状態容量を切り出す。すなわち、16 個の記号からなるアルファベットから一様かつ独立に抽出された T 文字のランダム文字列を反転させるタスクである。各文字は独立であるため、すでにデコードされた文字は次の文字について何の情報も与えず、すべての文字がエンコーダの最終状態 $\mathbf{c}$ を経由しなければならない。また反転タスクでは、エンコーダが最後に読んだトークンをデコーダが最初に出力することになるため、Sutskever らの逆順入力の順序とも自然に一致する。

各ミニバッチでは、毎回新たなランダム文字列を生成する。`Reversal` は、ソース文字列と、先頭に開始トークン `V` を付加して右にシフトした反転ターゲットを、1 つのテンソル `X` にまとめて `d2l.train_steps` に渡す。

```
class Reversal:
    """Fresh batches of random strings and their reversals."""
    def __init__(self, length, batch_size=128, num_letters=16):
        self.length, self.batch_size = length, batch_size
        self.num_letters = num_letters

    def batch(self, batch_size, key):
        V = self.num_letters  # Letters 0 to V - 1; V is the start token
        src = jax.random.randint(key, (batch_size, self.length), 0, V)
        tgt = src[:, ::-1]
        dec_in = jnp.concatenate([jnp.full_like(tgt[:, :1], V),
                                  tgt[:, :-1]], 1)
        return jnp.concatenate([src, dec_in], 1), tgt

    def train_dataloader(self):
        key = d2l.get_key()
        while True:
            key, subkey = jax.random.split(key)
            yield self.batch(self.batch_size, subkey)
```

生のビット数で数えると、等確率な 16 種類の各文字は $\log_2 16 = 4$ ビットの情報を持つため、長さ T の文字列は $4T$ ビット（$T = 8$ で 32 ビット、$T = 64$ で 256 ビット）の情報を持つ。原理的には、64 個の単精度浮動小数点数は 2,048 ビットを占めるため、512 文字分の余裕がある。では、訓練された GRU は実際には 64 個または 256 個の浮動小数点数を通じて何文字を詰め込み、取り出せるのだろうか。

Figure 12.5 および Cho, Van Merriënboer, Gulcehre, et al. (2014) に従い、GRU エンコーダの最終状態 $\mathbf{c}$ は GRU デコーダを初期化する（$\mathbf{s}_0 = \mathbf{c}$）とともに、各デコーディングステップで入力埋め込みに連結される。

$$\mathbf{s}_{t'} = g(y_{t'-1}, \mathbf{c}, \mathbf{s}_{t'-1}). \tag{12.13}$$

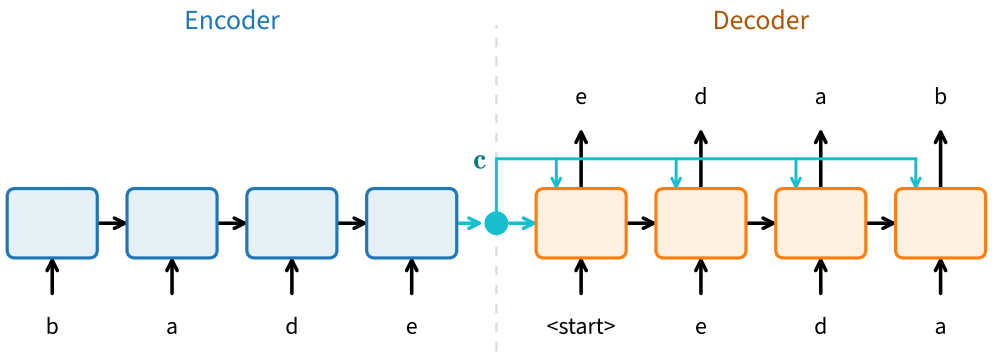


Figure 12.5: 反転タスクにおける GRU エンコーダ・デコーダを時間方向に展開した図。エンコーダはソースを読み取り、その最終状態 **c** がデコーダを初期化するとともにデコーダの各ステップに入力される。デコーダは開始トークンの後ろに右シフトされた反転文字列を読み取り、反転文字列を予測する。

```
class Reverser(nnx.Module):
    """A GRU encoder whose final state starts and conditions a GRU decoder."""
    def __init__(self, num_letters, num_hiddens, num_inputs=32, *, rngs):
        lecun = {'recurrent_kernel_init': nnx.initializers.lecun_normal()}
        self.embedding = nnx.Embed(num_letters + 1, num_inputs, rngs=rngs,
            embedding_init=nnx.initializers.normal(num_inputs ** -0.5))
        self.encoder = nnx.RNN(nnx.GRUCell(
            num_inputs, num_hiddens, rngs=rngs, **lecun), return_carry=True)
        self.decoder = nnx.RNN(nnx.GRUCell(
            num_inputs + num_hiddens, num_hiddens, rngs=rngs, **lecun))
        self.head = nnx.Linear(num_hiddens, num_letters, rngs=rngs)

    def __call__(self, X):  # X: the source, then the decoder input
        src, dec_in = jnp.split(X, 2, axis=1)
        c, _ = self.encoder(self.embedding(src))  # (batch, num_hiddens)
        context = jnp.broadcast_to(c[:, None], (*dec_in.shape, c.shape[-1]))
        dec_X = jnp.concatenate([self.embedding(dec_in), context], -1)
        return self.head(self.decoder(dec_X, initial_carry=c))
```

新規に生成した 128 個の文字列からなるミニバッチを用い、3×10^{-3} からゼロへのコサイン減衰を伴う AdamW（`d2l.LMOptimizer` および `d2l.train_steps`）によって、11.4 節と同じ設定で各モデルを 1,000 ステップ訓練する。評価では、新規の 1,000 文字列に対する文字ごとの正解率を教師強制のもとで測定する。各文字は独立であるため、正解の直前トークンを入力しても次のターゲット文字に関する情報は一切漏洩せず、同時に初期のデコーディング誤りが状態容量の測定に交絡するのを防げるからである。

```
def reversal_accuracy(num_hiddens, length):
    """Train a fresh model for 1,000 steps; return its per-letter accuracy."""
    data = Reversal(length)
    model = Reverser(data.num_letters, num_hiddens,
                     rngs=nnx.Rngs(d2l.get_key()))
```

```
    optimizer = d2l.LMOptimizer(model, lr=3e-3, num_steps=1000, warmup=0,
                                weight_decay=0.0)
    d2l.train_steps(model, data, optimizer, 1000)
    X, Y = data.batch(1000, d2l.get_key())
    return float((model(X).argmax(-1) == Y).mean())
```

状態幅 64 と 256 について、長さ $T \in \{8, 16, 32, 64\}$ にわたり 2 つの乱数シードで実験を実行する（ランダム予測の正解率は $1/16 \approx 0.06$ である）。

```
lengths, widths = (8, 16, 32, 64), (64, 256)
print('width' + ''.join(f'{f"T = {T}":>12}' for T in lengths))
for num_hiddens in widths:
    row = [[reversal_accuracy(num_hiddens, T) for _ in range(2)]
           for T in lengths]
    print(f'{num_hiddens:5d}' + ''.join(f'   {a:.2f}/{b:.2f}' for a, b in row))
```

```
width       T = 8       T = 16       T = 32       T = 64
   64   0.99/0.99   0.72/0.73   0.40/0.41   0.26/0.26
  256   1.00/1.00   0.99/0.99   0.79/0.81   0.50/0.51
```

幅 64 では、正解率は $T = 8$ での 0.99 から、$T = 16$ で約 0.73、$T = 64$ で 0.26〜0.27 へと低下する。状態幅を 4 倍の 256 に増やすと、$T = 16$ まで正解率 0.99 を維持し、$T = 64$ でも約半数の文字を復元できる。それでもなお、実効容量は生の浮動小数点のビット数に比べればごく一部にすぎない。$T = 16$ の文字列が持つ情報量はわずか 64 ビット（幅 64 では状態の浮動小数点数 1 個あたり 1 ビット）であるにもかかわらず、文字の 4 分の 1 が失われている。さらに、系列長に比例して状態幅を拡大しても性能は低下する（幅 64、$T = 8$ では 0.99 に達するのに対し、幅 256、$T = 32$ では約 0.80 にとどまる）。これは、ネットワークが序盤の文字をより長い非線形なリカレント更新の連鎖にわたって伝播させなければならないためである。

ニューラル機械翻訳も 2014 年にこれとまったく同じボトルネックに直面し (Cho, Van Merriënboer, Bahdanau, et al. 2014)、それを受けて Bahdanau et al. (2015) はデコーダがすべてのエンコーダ状態を振り返って参照できるようクロスアテンションを導入した（10.2 節、11.4 節）。これにより、入力長に対して線形に増大するキャッシュのコストと引き換えに、長さ 12 の反転タスクは 300 ステップ以内に完全に解けるようになった。本章の残りの節では、これと相補的な道筋を探る。すなわち、固定サイズの状態を保ったまま訓練を並列化し、状態をベクトルからキー・値行列メモリへと拡張し、最後に少数のアテンション層と組み合わせるアプローチである。

12.2.4 まとめ

エンコーダ・デコーダアーキテクチャは、ソースをエンコーダ状態 $\mathbf{c}$ に要約し、自己回帰デコーダを $\mathbf{c}$ で条件付けることによって、可変長のソース系列を可変長のターゲット系列へと写像する。翻訳品質は一般に、文字 n グラムの重複（chrF）や短文ペナルティを伴う単語 n グラム適合率（BLEU）で測定され、学習済みの意味的評価指標によって補完される。人工的な文字列反転タスクでは、単一の固定幅 GRU 状態ベクトルは系列長が増すにつれて急速に劣化し、状態の座標あたり 1 実効ビットを大きく下回る情報しか保持できない。この限界が、アテンション機構と、本章の残りの節で展開する構造化された高容量な再帰状態の双方への動機付けとなる。

12.2.5 演習問題

1. **BLEU 対 chrF**。ニアミスと言い換えについて $k = 1, 2, 3, 4$ で `bleu` のスコアを計算し、それぞれの値を n グラムの出現回数から説明せよ。次に、どちらの方向でもよいので chrF と

BLEU が可能な限り大きく食い違う候補訳を 1 つ構成し、その候補訳が各評価指標のどの性質を突いているかを説明せよ。

2. **状態を通過するビット数**。本節の数え上げの議論によれば、N 個の数値（各 b ビット）からなる状態は、高々 2^{Nb} 通りの値しかとり得ない。
   1. 16 種類の文字からなる長さ T のあらゆる文字列を誤りなく反転させるには、$Nb \geq 4T$ が必要であることを示せ。
   2. 本節と同様に正しい直前文字を入力として測定したデコーダの文字ごとの正解率を p とする。ファノの不等式（37.6.2.3 節）と相互情報量の連鎖律を用いて、文字列に関して少なくとも $T\,(4 - H_{\mathrm{b}}(1-p) - (1-p)\log_2 15)$ ビットの情報が $\mathbf{c}$ を通過したことを示せ。ただし、H_{b} はビット単位の二値エントロピーである。
   3. 表の各欄についてこの下界を計算し、$32N$ と比較せよ。訓練されたエンコーダは数値 1 個あたり何ビットを使用しているか。また、その値は T に依存するか。
   4. 幅 64 のモデルを各長さで 4 倍長く訓練せよ。どの長さで最も大きく改善し、長さに応じた低下は解消されるか。この結果から、限界を定めているのが状態なのか訓練なのかについて何が言えるか。
3. **Sutskever の工夫の解除**。本節の反転タスクは、最初に書き出す文字が最後に読み取った文字であるという点で、Sutskever らがソースを逆順にすることで得た配置を最初から備えている。ソースを逆順でエンコーダに入力することでこの配置を打ち消し、エンコーダが読み取った順序のままデコーダが文字列をコピーしなければならないようにせよ。
   1. 反転タスクとコピータスクのそれぞれについて、各文字を読み取ってから書き出すまでのステップ数を数えよ。平均して依存距離が短いのはどちらのタスクか。また、最も短い単一の依存距離を持つのはどちらか。
   2. 幅 64、$T = 16$ で両方のタスクを訓練し、全体および各出力位置での正解率を測定せよ。どちらのタスクの方が学習しやすいか。その結果は、ソースを逆順にする理由として Sutskever et al. (2014) が挙げた議論と一致するか。
4. **毎ステップでのコンテキスト入力**。Sutskever らが行ったように、コンテキスト $\mathbf{c}$ を毎ステップの入力としては渡さず、初期状態としてのみデコーダに渡せ。それによって各長さでの正解率はどのように変化するか。また、コンテキストを繰り返し与えることが小さなモデルの助けになり得るのはなぜか。

12.3 線形漸化式と並列スキャン

12.1 節のゲート付きネットワークは数百ステップにわたって情報を保持できるが、各ステップのゲートと候補が前の状態 $\mathbf{h}_{t-1}$ に依存するため、依然として系列を 1 トークンずつ順番に処理する。$4{,}096$ トークンの系列では、GRU は $4{,}096$ 回の逐次更新を実行することになり、Transformer が大規模な行列積によって全位置を並列に処理する一方で、訓練中に GPU の大部分を遊休状態にしてしまう。リカレントネットワークにおいて、$O(1)$ 状態の自己回帰生成（autoregressive generation）を維持しつつ、全位置にわたって並列に訓練することはできるだろうか。

答えはイエスである。ゲートと候補が現在の入力 $\mathbf{x}_t$ のみに依存するようになれば、漸化式は $\mathbf{h}_{t-1}$ に関して**線形**（より正確には**アフィン**）になるからである。アフィン更新はプレフィックス和（prefix sum）と同様に結合的に合成でき (Hillis and Steele 1986)、全 T 個の状態を $\lceil \log_2 T \rceil$ 回の並列ラウンドで評価できる。以下では、スカラーの減衰から、最小 GRU（minimal GRU、`MinGRU`）として Feng et al. (2024) が導入した構造へと進み、結合的な並列スキャン（parallel scan）を実装したうえで、バイトレベルの WikiText-103 で `MinGRU` を評価し、最大 $16{,}384$ トークンまでの実時間のスケーリングを逐次型の GRU および因果アテンション（causal attention）と比較する。

12.3.1 線形漸化式

12.3.1.1 減衰する単一の数値

まず、スカラー状態 h_t が直前の値の一定割合 $a \in (0,1)$ を保持し、そこに入力 x_t を加える場合から始めよう。

$$h_t = a\, h_{t-1} + x_t, \qquad h_0 = 0. \tag{12.14}$$

Equation 12.14 を展開すると、h_t は過去の入力によって直接次のように表される。

$$h_t = \sum_{s=1}^{t} a^{t-s} x_s. \tag{12.15}$$

すべての h_t は半減期（half-life）$\ln 2 / \ln(1/a)$ ステップ（$a = 0.9$ では ≈ 7 ステップ、$a = 0.999$ では ≈ 693 ステップ）をもつ過去の入力の指数重み付き移動平均（exponentially weighted moving average）であり、勾配 $\partial h_t / \partial x_s = a^{t-s}$ もまったく同じ率で減衰する。

12.3.1.2 入力依存の減衰

ベクトル状態 $\mathbf{h}_t \in \mathbb{R}^d$ へと一般化し、入力依存（input-dependent）の要素ごとの減衰 $\mathbf{a}_t \in (0,1)^d$ と**書き込み**（write）ベクトル $\mathbf{b}_t \in \mathbb{R}^d$ を導入すると、次のアフィン漸化式（affine recurrence）が得られる。

$$\mathbf{h}_t = \mathbf{a}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_t, \qquad \mathbf{a}_t = \mathbf{a}(\mathbf{x}_t), \quad \mathbf{b}_t = \mathbf{b}(\mathbf{x}_t). \tag{12.16}$$

Equation 12.16 を展開すると、べき乗 a^{t-s} は途中の減衰の累積積（cumulative product）に置き換わる。

$$\mathbf{h}_t = \sum_{s=1}^{t} \Big(\prod_{r=s+1}^{t} \mathbf{a}_r \Big) \odot \mathbf{b}_s. \tag{12.17}$$

$\mathbf{a}_t$ と $\mathbf{b}_t$ は $\mathbf{x}_t$ のみに依存するため、漸化式を実行する前に、全 T 個の減衰ベクトルと書き込みベクトルを系列全体にわたって並列に計算できる。また、各 $\mathbf{a}_r \in (0,1)^d$ であるから、過去の書き込みを発散させることなく選択的に保持したり（$\mathbf{a}_r \approx \mathbf{1}$）、消去したり（$\mathbf{a}_r \approx \mathbf{0}$）できる。

12.3.1.3 最小 GRU

前の状態 $\mathbf{h}_{t-1}$ が 12.1.3 節の標準的な GRU に入り込む 3 つの箇所を見てみよう。

$$\begin{aligned}
\mathbf{z}_t &= \sigma(\mathbf{W}_{\mathrm{xz}}\mathbf{x}_t + \underbrace{\mathbf{W}_{\mathrm{hz}}\mathbf{h}_{t-1}}_{(1)} + \mathbf{b}_{\mathrm{z}}), \\
\mathbf{r}_t &= \sigma(\mathbf{W}_{\mathrm{xr}}\mathbf{x}_t + \underbrace{\mathbf{W}_{\mathrm{hr}}\mathbf{h}_{t-1}}_{(2)} + \mathbf{b}_{\mathrm{r}}), \\
\tilde{\mathbf{h}}_t &= \tanh\big(\mathbf{W}_{\mathrm{xh}}\mathbf{x}_t + \underbrace{\mathbf{W}_{\mathrm{hh}}(\mathbf{r}_t \odot \mathbf{h}_{t-1})}_{(3)} + \mathbf{b}_{\mathrm{h}}\big),
\end{aligned}$$

これに続いて $\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t$ が計算される。印を付けた 3 つのリカレントな依存関係を除去すると（これによりリセットゲート $\mathbf{r}_t$ は完全に消える）、Feng et al. (2024) の**最小 GRU**（**minGRU**）が得られる。

$$\begin{aligned} \mathbf{z}_t &= \sigma(\mathbf{W}_{\mathrm{z}}\mathbf{x}_t + \mathbf{b}_{\mathrm{z}}), \qquad \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{\mathrm{h}}\mathbf{x}_t + \mathbf{b}_{\mathrm{h}}), \\ \mathbf{h}_t &= \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t. \end{aligned} \tag{12.18}$$

これは Equation 12.16 において減衰を $\mathbf{a}_t = \mathbf{z}_t$、書き込みを $\mathbf{b}_t = (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t$ としたアフィン漸化式であり、層のパラメータ数を $6d^2$ から $2d^2$ へと削減する。単一の minGRU 層はすでに $\mathbf{h}_{t-1}$ に入っている情報でゲートを条件付けることができないが、複数の minGRU ブロックを位置ごとの MLP とともに積み重ねれば、時間方向の並列評価を損なうことなくその能力の一部を回復できる。各層のゲートは、1 つ下の層の状態を参照できるからである。これにより、GRU が毎ステップ 1 回の状態依存ゲーティングを行うのに対し、1 層あたり 1 ラウンドの状態依存ゲーティングが得られる。固定回数のラウンドでは実現できない処理については、12.7.3.5 節で検証する。

12.3.2 並列スキャン

12.3.2.1 アフィンステップの合成

展開された和 Equation 12.17 をすべての $t = 1, \dots, T$ に対して素朴に評価すると、$O(T^2)$ 回の演算が必要になる。そこで、関数合成の結合律（associativity）を利用する。ステップ t のアフィン写像 $\mathbf{h} \mapsto \mathbf{a}_t \odot \mathbf{h} + \mathbf{b}_t$ を組 $(\mathbf{a}_t, \mathbf{b}_t)$ で表そう。ステップ 1 に続けてステップ 2 を適用すると $\mathbf{a}_2 \odot (\mathbf{a}_1 \odot \mathbf{h} + \mathbf{b}_1) + \mathbf{b}_2$ となり、これは次の係数をもつ別のアフィン写像になる。

$$(\mathbf{a}_1, \mathbf{b}_1) \bullet (\mathbf{a}_2, \mathbf{b}_2) = (\mathbf{a}_1 \odot \mathbf{a}_2,\ \mathbf{a}_2 \odot \mathbf{b}_1 + \mathbf{b}_2). \tag{12.19}$$

関数の合成は結合的であり、任意のステップ列の合成もやはり同じサイズの単一の組 $(\mathbf{a}, \mathbf{b})$ になるため、$\mathbf{h}_1, \dots, \mathbf{h}_T$ を初期状態 $\mathbf{h}_0 = \mathbf{0}$ から計算する処理は、演算 $\bullet$ のもとでの結合的なプレフィックス**スキャン**に帰着する（$\mathbf{h}_t$ は t 番目のプレフィックスの第 2 成分から読み出せる）。

12.3.2.2 倍増スケジュール

Figure 12.6 は、Hillis and Steele (1986) による並列倍増スケジュール（parallel doubling schedule）を $T = 8$ ステップの場合について示している。ラウンド $r = 0, 1, \dots, \lceil \log_2 T \rceil - 1$ において、すべての位置はストライド 2^r だけ手前の位置と結合する。$\lceil \log_2 T \rceil$ ラウンド後には、すべての位置が完全なプレフィックスを保持する。

Hillis–Steele の倍増スケジュールは $O(T \log_2 T)$ 回の結合演算を $\lceil \log_2 T \rceil$ 回の並列ラウンドにわたって実行するのに対し、Blelloch (1990) による**演算量効率的**（work-efficient）なアップスイープ・ダウンスイープ木スケジュールは、$\sim 2T$ 回の結合演算をサイズが縮小していく $2\lceil \log_2 T \rceil$ 回のラウンドにわたって実行する。

XLA に組み込まれた演算量効率的な `jax.lax.associative_scan` を呼び出す。

12.3.2.3 実装

`linear_scan` は、時間軸（軸 1）に沿ってすべての状態 $\mathbf{h}_1, \dots, \mathbf{h}_T$（Equation 12.16）を計算し、任意の初期状態 $\mathbf{h}_0$ が与えられた場合は最初の書き込み $\mathbf{b}_1 \leftarrow \mathbf{a}_1 \odot \mathbf{h}_0 + \mathbf{b}_1$ に折り込む。

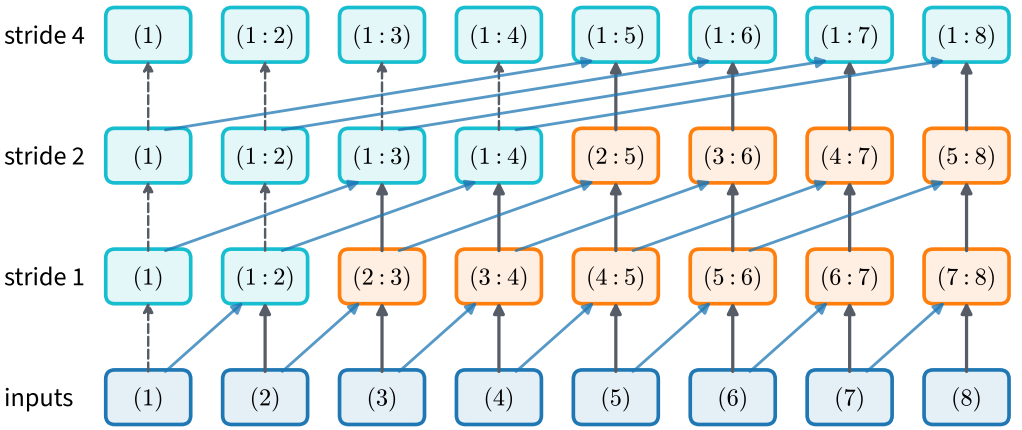


Figure 12.6: 8 ステップに対する並列スキャン。各ボックスはその値がどのステップを結合したものかを表し、(1:4) はステップ 1 から 4 を意味する。各ラウンドでは、それぞれの位置が一定のストライドだけ手前の位置と結合し（青色の矢印）、ストライドはラウンドごとに倍増する。破線の矢印は値をそのままコピーする。青緑色のボックスは完全なプレフィックスを保持している。$\log_2 8 = 3$ ラウンド後には、すべての位置が完全なプレフィックスを保持する。

```
def linear_scan(a, b, h0=None):
    """All states of h_t = a_t * h_{t-1} + b_t; time is the second axis."""
    a = jnp.broadcast_to(a, b.shape)  # Decays may omit axes; broadcast them
    if h0 is not None:  # Fold the initial state into the first step
        b = b.at[:, 0].add(a[:, 0] * h0)
    def combine(earlier, later):
        (a1, b1), (a2, b2) = earlier, later
        return a1 * a2, a2 * b1 + b2
    # Autograd keeps the intermediate results of the scan for the backward pass
    return jax.lax.associative_scan(combine, (a, b), axis=1)[1]
```

長さ $T = 100$（2 のべき乗でない長さ）において、`linear_scan` を 1 ステップずつ進める逐次ループと照合して検証してみよう。

```
def sequential_scan(a, b):
    def step(h, ab):
        h = ab[0] * h + ab[1]
        return h, h
    # lax.scan steps through the leading axis, so time goes first
    ab = (jnp.swapaxes(a, 0, 1), jnp.swapaxes(b, 0, 1))
    return jnp.swapaxes(jax.lax.scan(step, jnp.zeros_like(b[:, 0]), ab)[1], 0, 1)

key_a, key_b = jax.random.split(d2l.get_key())
a = jax.random.uniform(key_a, (4, 100, 8))
b = jax.random.normal(key_b, (4, 100, 8))
print(f'max deviation: '
      f'{jnp.abs(linear_scan(a, b) - sequential_scan(a, b)).max():.1e}')
```

```
max deviation: 4.8e-07
```

並列スキャンと逐次ループは、float32 の丸め誤差の許容範囲（$< 10^{-6}$）内で一致する。

12.3.3 線形リカレント言語モデル

12.3.3.1 テキスト上の minGRU

`linear_scan` を 2 つの線形射影（$\mathbf{W}_z$ と $\mathbf{W}_h$）と組み合わせることで、`MinGRU`（Equation 12.18）を `d2l.SeqLM` にそのまま差し替え可能な系列ミキサとして実装できる。

```
class MinGRU(nnx.Module):
    """The minimal GRU: its gate and candidate read only the input."""
    def __init__(self, num_inputs, num_hiddens, *, rngs):
        self.W_z = nnx.Linear(num_inputs, num_hiddens, rngs=rngs)  # Decay
        self.W_h = nnx.Linear(num_inputs, num_hiddens, rngs=rngs)

    def coefficients(self, X):
        """Decays a_t and writes b_t of the linear recurrence."""
        Z = jax.nn.sigmoid(self.W_z(X))
        return Z, (1 - Z) * jnp.tanh(self.W_h(X))

    def __call__(self, X):  # X: (batch_size, num_steps, num_inputs)
        return linear_scan(*self.coefficients(X))
```

ゲートから $\mathbf{h}_{t-1}$ を除去すると、バイトレベルの WikiText-103 におけるモデル化品質はどの程度損なわれるだろうか。12.1.4.2 節とまったく同じ設定（`d2l.SeqLM` の 4 ブロック、128 バイトのウィンドウ、1,000 ステップ）を用い、まず `text_model` と 3 つの比較アーム `arms` を定義する。すなわち、`GRU`（$d = 256$、パラメータ数 3.81M）、同じ幅の `minGRU`（$d = 256$、パラメータ数 2.76M）、およびパラメータ数が等しくなるよう幅を広げた `minGRU`（`minGRU, d=302`、パラメータ数 3.81M）である。

```
def text_model(mixer, d):  # 256 byte values; four blocks of width d
    rngs = nnx.Rngs(d2l.get_key())
    return d2l.SeqLM(256, d, [mixer(d, rngs) for _ in range(4)], rngs=rngs)

lecun = nnx.initializers.lecun_normal()  # For the recurrent matrices
gru = lambda d, rngs: nnx.RNN(
    nnx.GRUCell(d, d, recurrent_kernel_init=lecun, rngs=rngs), unroll=8)
min_gru = lambda d, rngs: MinGRU(d, d, rngs=rngs)
arms = {'GRU': (gru, 256), 'minGRU': (min_gru, 256),
        'minGRU, d=302': (min_gru, 302)}  # As many parameters as the GRU
```

128 バイトの `WikiText103` データセットを読み込み、3 つのアームすべてにわたって `d2l.text_table` を実行し、検証用に最初の訓練済み `minGRU` モデルを保持する。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
trained = d2l.text_table(text_model, arms, data)
model = trained['minGRU'][0]  # The model we look inside below
```

```
                        validation        0-15       16-63      64-127   ms/step
GRU             3,814,144 1.300/1.299 1.491/1.491 1.274/1.273 1.271/1.271 16/16
```

```
minGRU          2,764,544 1.346/1.345 1.520/1.521 1.320/1.320 1.321/1.320 10/10
minGRU, d=302   3,816,932 1.330/1.330 1.510/1.508 1.305/1.306 1.305/1.304 10/10
```

同じ幅（$d = 256$）では、minGRU は ~ 0.05 nat/byte の差で GRU の後塵を拝するが（1.35 対 1.30）、同じパラメータ数（minGRU, d=302）ではその差の約 5 分の 2 を縮め（~ 1.33）、12.1.4.2 節の素の tanh RNN とほぼ並ぶとともに、アテンションのベースライン（1.38）を上回り、しかも逐次ループではなく並列スキャンによって訓練できる。

12.3.3.2 訓練済みモデルの内部

minGRU が再帰状態（recurrent state）にどれだけ依存しているかを調べるため、訓練済みの $d = 256$ モデルを、毎ステップの前に状態をゼロにリセットした設定（StateReset、これにより $\mathbf{h}_t = \mathbf{b}_t$ は現在のバイトのみに依存する）で評価する。

```
class StateReset(nnx.Module):
    """A minGRU layer whose state is reset to zero before every step."""
    def __init__(self, layer):
        self.layer = layer

    def __call__(self, X):
        return self.layer.coefficients(X)[1]

layers = model.mixers
loss = np.mean(d2l.loss_by_position(model, data))
model.mixers = nnx.List([StateReset(layer) for layer in layers])
reset = np.mean(d2l.loss_by_position(model, data))
model.mixers = layers
print(f'validation loss: {loss:.3f} nats per byte, '
      f'with the state reset at every step: {reset:.3f}')
```

```
validation loss: 1.346 nats per byte, with the state reset at every step: 2.482
```

状態をゼロにすると、検証損失は 1.35 から 2.48 nat/byte へと悪化し（実質的に 2.45 nat/byte という 11.2 節の経験的バイトバイグラムのベースラインと同等）、1.1 nat/byte 以上の改善が状態によって運ばれる文脈に由来していることが確認できる。

また、第 1 ブロックにおける更新ゲート（update gate）の減衰 $\mathbf{z}_t$ の分布を訓練前後で調べることもできる。first_mixer_input を d2l.SeqLM に追加すると、最初のミキサに入力される正規化後のトークン埋め込みを取り出せる。

```
@d2l.add_to_class(d2l.SeqLM)
def first_mixer_input(self, X):
    """What the first mixer reads: the normalized token embeddings."""
    return self.norms[0](self.embedding(X))
```

検証ミニバッチを first_gates に通し、未訓練の minGRU と訓練済みの model の双方について、更新ゲートが 0.5、0.9、0.99 を超える割合を集計する。

```
def first_gates(model, X):  # Gates of the first layer
    return model.mixers[0].coefficients(model.first_mixer_input(X))[0]

X = jnp.asarray(next(iter(data.val_dataloader()))[0])  # Validation text
```

```
Z_init = first_gates(text_model(min_gru, 256), X)  # Untrained
Z = first_gates(model, X)
print('gate above  half-life in steps  share at initialization  trained')
for g in (0.5, 0.9, 0.99):
    print(f'{g:10}  {math.log(2) / -math.log(g):17.0f}  '
          f'{(Z_init > g).mean():23.1%}  {(Z > g).mean():7.1%}')
```

```
gate above  half-life in steps  share at initialization  trained
       0.5                  1                    49.1%    52.3%
       0.9                  7                     1.7%    17.8%
      0.99                 69                     0.0%     0.0%
```

訓練によって、半減期が ≥ 7 バイトとなるゲート（$g > 0.9$）の割合は 10 倍以上に増加し（$\sim 1\%$ から約 18% へ）、128 バイトのウィンドウ上で必要とされる単語レベルのタイムスケールと一致するようになる。一方、ゼロバイアス初期化からは、$g > 0.99$（半減期 69 バイト）に達するゲートはほとんど現れない。

12.3.4 アテンションとのコスト比較

12.3.4.1 演算量、深さ、メモリ

Table 12.1 は、幅 d の 1 層で T トークンを処理する際の漸近的な訓練時の演算量、逐次的な深さ、および自己回帰生成時の状態を比較したものである。

Table 12.1: 幅 d の 1 層における T トークンに対するコスト（定数倍を除く）。

層	訓練時の演算量	連続するステップ数	生成時のメモリ
GRU、1 ステップずつ	Td^2	T	d
minGRU、倍増スキャン	$Td^2 + Td\log_2 T$	$\log_2 T$	d
minGRU、演算量効率的スキャン	$Td^2 + Td$	$2\log_2 T$	d
因果アテンション	$Td^2 + T^2 d$	数回の行列積	$2Td$

`GRU` と `minGRU` はどちらも $O(Td^2)$ の射影の演算量を実行し、デコーディング中には $O(d)$ の状態ベクトルを保持するが、`minGRU` は訓練時の逐次的な深さを T 回の行列・ベクトル積ステップから $O(\log_2 T)$ 回の要素ごとのスキャンラウンドへと削減する。因果アテンションの訓練時の逐次的な深さは $O(1)$ であるが、アテンションスコアと混合の演算量は $O(T^2 d)$ として二次関数的に増大し（射影の $8Td^2$ FLOPs を $T \approx 4d$ 付近で上回る）、推論時の KV キャッシュも $2Td$ として線形に増大する。

12.3.4.2 実測コスト

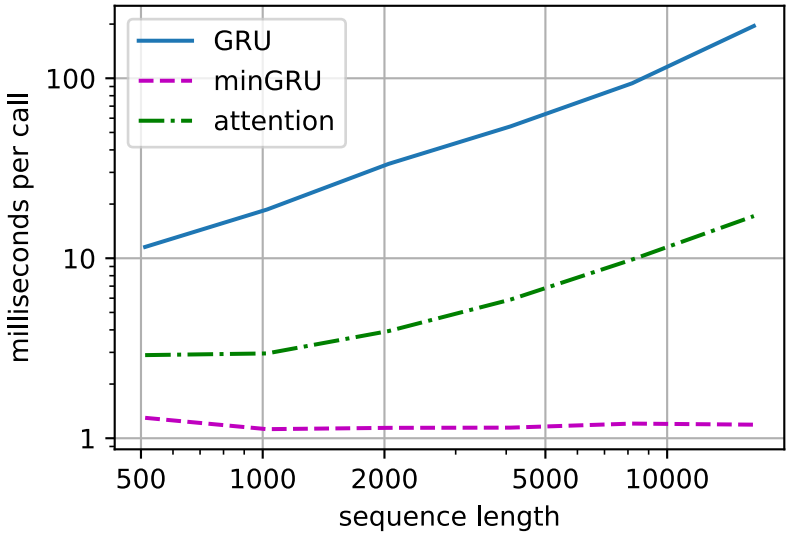
これらのスケーリング則（scaling law）が実ハードウェア上でどこで交差するかを見るため、float16 で幅 $d = 256$ とし、1 バッチあたりの総トークン数を $B \times T = 65{,}536$ に固定したまま、系列長（sequence length）T を 512 から 16,384 までスイープさせながら順伝播と逆伝播の合計時間を計測する。総トークン数を固定しているため、コストが T に対して厳密に線形な層の曲線は平坦になる一方、$O(T)$ 回の逐次ステップまたは $O(T^2)$ のペアワイズ演算量が支配的な層では、T が 2 倍になるたびに実行時間も 2 倍になる。

```
class FusedAttention(d2l.CausalAttention):
    """Causal attention through cuDNN's fused kernel."""
    @staticmethod
    def attend(q, k, v):
        return jax.nn.dot_product_attention(q, k, v, is_causal=True,
                                            implementation='cudnn')

d, tokens = 256, 65536  # Width; tokens per call, in sequences of length T
half, rngs = jnp.float16, nnx.Rngs(d2l.get_key())  # 16-bit floats
timed = {'GRU': nnx.RNN(nnx.GRUCell(d, d, recurrent_kernel_init=lecun,
                                    param_dtype=half, rngs=rngs), unroll=8),
         'minGRU': MinGRU(d, d, rngs=rngs),
         'attention': FusedAttention(d, num_heads=4, rngs=rngs)}
for layer in timed.values():  # Every weight in 16 bits
    params = nnx.state(layer, nnx.Param)
    nnx.update(layer, jax.tree.map(lambda p: p.astype(half), params))

@nnx.jit
def forward_backward(layer, X):
    loss = lambda layer, X: layer(X).astype(jnp.float32).sum()
    return nnx.grad(loss, argnums=(0, 1))(layer, X)

def time_call(layer, T):
    """Milliseconds for a forward and backward pass over tokens / T
    sequences of length T."""
    X = jax.random.normal(d2l.get_key(), (tokens // T, T, d), half)
    return 1000 * d2l.Benchmark(lambda: forward_backward(layer, X),
                                warmup=1, repeats=3).time
```

系列長 $T \in \{512, 1024, 2048, 4096, 8192, 16384\}$ にわたって `time_call` をスイープし、`GRU`、`minGRU`、`attention` の実実行時間を両対数軸上で比較する。

```
lengths = [512, 1024, 2048, 4096, 8192, 16384]
times = {name: [time_call(layer, T) for T in lengths]
         for name, layer in timed.items()}
print('T        ', *(f'{T:>7d}' for T in lengths))
for name, ms in times.items():
    print(f'{name:9s}', *(f'{t:7.1f}' for t in ms))
d2l.plot(lengths, list(times.values()), 'sequence length',
         'milliseconds per call', legend=list(times), xscale='log',
         yscale='log', figsize=(4.5, 3))
```

```
T             512    1024    2048    4096    8192   16384
GRU          11.6    18.6    33.5    54.0    93.8   195.3
minGRU        1.3     1.1     1.1     1.1     1.2     1.2
attention     2.9     3.0     3.9     5.9     9.8    17.2
```



実測されたスケーリング曲線は Table 12.1 と一致する。

- **GRU**：あらゆる長さで最も遅く、15× 以上に増大する（$T = 512$ から 16,384 まで）。T 回の逐次ステップを時間方向に並列化できないためである。
- **minGRU**：系列長が 32 倍に増加する全範囲にわたって、ほぼ平坦である。
- **attention**：$T \approx 2{,}048$ までは $O(Td^2)$ の射影とメモリトラフィックが支配的であるためほぼ平坦であるが、その後は系列長が倍増するごとに 2× へ漸近するように上向きに曲がる。$O(T^2 d)$ のアテンション項が支配的になるためである。

`minGRU` の演算量効率的スキャンは平坦なままであり、あらゆる長さで `attention` より高速で、$T = 16{,}384$ では約 14× 高速である。

`minGRU` にまだ欠けているのは、多重タイムスケールの記憶を構造化された方法で初期化・制御する仕組みである。この問いには、12.4 節で連続時間状態空間モデル（continuous-time state space model）を通じて取り組む。

12.3.5 まとめ

リカレント層のゲートから前の状態 $\mathbf{h}_{t-1}$ を除去すると、アフィン漸化式 $\mathbf{h}_t = \mathbf{a}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_t$ が得られ、その各ステップは $(\mathbf{a}_1, \mathbf{b}_1) \bullet (\mathbf{a}_2, \mathbf{b}_2) = (\mathbf{a}_1 \odot \mathbf{a}_2, \mathbf{a}_2 \odot \mathbf{b}_1 + \mathbf{b}_2)$ によって結合的に合成できる。結合的なスキャンを用いることで、訓練時には全 T 個の状態を $O(\log_2 T)$ 回の並列ラウンドで評価しつつ、デコーディング時には $O(1)$ 状態の逐次更新を維持できる。この簡略化を GRU に適用して得られる `MinGRU` は、バイトレベルの WikiText-103 において同じパラメータ数の素の RNN に匹敵する性能を示し（完全な GRU との差は約 0.03 nat/byte）、しかも最大 16,384 トークンまでトークンあたりほぼ一定の時間で訓練できる。これは逐次型の GRU より 1 桁以上高速であり、長文脈においては二次計算量のアテンションよりも数倍高速である。

12.3.6 演習問題

1. **合成に関する閉包性**。並列スキャンには結合律だけでなく、任意の連続するステップの合成が単一ステップと同じ個数の数値で記述でき、すべての結合演算が同じコストで実行できることが求められる。

    1. Equation 12.19 を、漸化式 $\mathbf{h}_t = \mathbf{A}_t\mathbf{h}_{t-1} + \mathbf{b}_t$（ただし $\mathbf{A}_t \in \mathbb{R}^{d\times d}$ は密行列）へと、各積の因子の順序を正しく保って一般化せよ。1 回の結合演算と 1 回の逐次ステップの積和演算数を数えよ。倍増スキャンの総演算量（約 $T\log_2 T$ 回の結合演算）とループの総演算量を $T = 4096$、$d = 64$ において比較せよ。この比較は密な遷移行列について何を示唆するか。
    2. スカラーステップ $h \mapsto \tanh(w_t h + c_t)$ に対し、$w_1, w_2 \neq 0$ である 2 ステップの合成は、どのような $h \mapsto \tanh(wh + c)$ の形（任意の w と c）にも表せないことを示せ。k ステップの合成を表現するには、スキャンは何を保持しなければならないか。
2. [code] **直接計算の経路**。スカラーの減衰に対しては、展開された和 Equation 12.17 はスキャンを使わずに、累積積 $P_t = a_1 a_2 \cdots a_t$ と 1 回の累積和から評価できる。
    1. $h_t = P_t \sum_{s=1}^{t} b_s / P_s$ が成り立つことを示せ。ライブラリの `cumprod` と `cumsum` を用いてこれを実装し、0.9 付近から抽出した減衰に対して 32 ビット浮動小数点数で `linear_scan` と比較せよ。どの長さで破綻するか、またなぜスキャンはその長さで破綻しないのか。
    2. Feng et al. (2024) は減衰と書き込みの対数を扱い、和を累積 log-sum-exp に置き換えることでこの式を修正している。この修正を実装せよ。書き込みの符号にはどのような条件が必要となり、著者らはそれをどのように実現しているか。
3. [code] **候補の非線形性**。`MinGRU` は tanh を候補 $\tilde{\mathbf{h}}_t$ に残しているのに対し、Feng et al. (2024) は線形な候補を用いている。
    1. $\mathbf{h}_0 = \mathbf{0}$ から出発し、$\mathbf{h}_t$ の各座標を候補 $\tilde{\mathbf{h}}_1, \ldots, \tilde{\mathbf{h}}_t$ の重み付き和として表せ。重みが非負でその和が高々 1 になることを示し、あらゆる長さで成り立つ $|\mathbf{h}_t|$ の上界を tanh がある場合とない場合のそれぞれについて導出せよ。
    2. tanh を取り除き、本節の言語モデルを `d2l.train_text` で 2 回訓練して、その検証損失を `MinGRU` の 2 回の実行結果と比較せよ。その差は実行間のばらつきより大きいか。観察された結果を導出した上界と関連付けて説明せよ。
4. [code] **状態の引き継ぎ**。`linear_scan` は初期状態を受け取るため、長い系列を分割して処理できる。
    1. 長さ T の系列を長さ τ のチャンクに分割し、直前のチャンクの最終状態から開始して各チャンクに対してスキャンを実行し、連結した状態が系列全体に対する 1 回の呼び出し結果と一致することを確かめよ。
    2. 次に、各チャンク境界で引き継がれる状態に対して勾配停止を適用せよ。状態自体は変化しない。ステップ t における損失の勾配への寄与のうちどれが消え、訓練済みモデルがテキストを生成する際にはどの依存関係を依然として利用できるか。打ち切り型通時的誤差逆伝播法（8.5 節）と比較せよ。
5. [code] **演算量効率的なスキャン**。●再帰的なスケジュールは倍増スケジュールよりも演算量が少ない。隣接するペアを結合して系列長を半分にし、その半分の長さの系列を再帰的にスキャンして 1 つおきのプレフィックスを得たあと、残りの各プレフィックスをもう 1 回の結合演算で求める。
    1. $T = 2^k$ に対して、各スケジュールの結合演算の回数と結合ラウンド数を数え、その両方をループの $T - 1$ ステップと $T = 16384$ において比較せよ。
    2. スライスを用いて両方のスケジュールを実装し、どちらも `sequential_scan` と照合して確認したうえで、12.3.4 節の計測と同じ系列長で実行時間を計測せよ。実行時間の比は結合演算の回数の比に従うか。短い系列長では従わない場合がある理由を説明せよ。

12.4 状態空間モデル

記憶はどのくらいの期間持続すべきだろうか。文の主語は数語の間、段落の話題は数百語の間、そして小説の登場人物の正体は 1 冊の本全体にわたって重要となる。原理的には、12.3 節の

minGRU はこれらの時間スケールのいずれも学習できるはずだが、そのゲートは $\sigma(0) = 1/2$（半減期 1 ステップ）の付近から出発する。ある記号を 1,000 ステップにわたって保持するには 0.999 付近の減衰（半減期 693 ステップ）が必要であり、初期減衰 $1/2$ を通じて 1,000 ステップさかのぼる勾配は 2^{-1000} 倍に縮小し、単精度では 0 にアンダーフローしてしまう。また、一様なゲート初期化では、最初から各チャネルに多様な時間スケールのバンクを与えることもできない。

連続時間の力学系（dynamical system）は、多重時間スケールの減衰に対する自然なパラメータ化を与える。**状態空間モデル**（state space model、SSM）は、各チャネルの記憶を、異なる減衰率を持ち離散的なステップサイズ Δ でサンプリングされる連続時間線形微分方程式のバンクとして扱う。この系を離散化すると、ステップごとの減衰が状態レートとチャネルのステップサイズの積で決まる線形漸化式が得られる。これにより、訓練開始前に幅広い半減期のスペクトルを初期化し、各チャネルに一度に N 個の時間スケールを持たせ、さらに時不変なダイナミクスを FFT による因果畳み込み（causal convolution）として並列に評価できる。以下では、厳密なゼロ次ホールド（zero-order hold）離散化を導出し、Gu, Gupta, et al. (2022) の対角 S4D 層を実装したうえで、1,000 ステップの遅延想起（delayed recall）において多重時間スケール初期化が長距離記憶の鍵となる要素かどうかを検証する。

12.4.1 連続時間から漸化式へ

12.4.1.1 連続時間ダイナミクス

まず、スカラー状態 $h(t)$ がレート λ で減衰しつつ、入力信号 $x(t)$ によって駆動される系から始めよう。

$$h'(t) = \lambda\, h(t) + x(t). \tag{12.20}$$

$x(t) = 0$ のとき、解は $h(t) = e^{\lambda t} h(0)$ である。安定性のためには $\lambda < 0$ が必要であり、このとき $h(t)$ は指数関数的に減衰する（$\lambda = -1$ では、割合 $e^{-1} \approx 0.37$ が連続時間の 1 単位後に残る）。一方、$\lambda > 0$ では際限なく増大する。重要な点として、λ はトークンステップあたりではなく、**単位時間あたり**の減衰率である。

状態空間モデル（SSM）は Equation 12.20 を一般化し、N 次元の状態ベクトル $\mathbf{h}(t) \in \mathbb{R}^N$ をスカラー入力 $x(t)$ で駆動してスカラー出力 $y(t)$ へと射影する。

$$\begin{aligned} \mathbf{h}'(t) &= \mathbf{A}\,\mathbf{h}(t) + \mathbf{B}\,x(t), \\ y(t) &= \mathbf{C}\,\mathbf{h}(t) + D\,x(t), \end{aligned} \tag{12.21}$$

ここで、$\mathbf{B} \in \mathbb{R}^N$ は入力を状態へと射影し、**状態行列**（state matrix）$\mathbf{A} \in \mathbb{R}^{N \times N}$ は状態を時間発展させ、$\mathbf{C} \in \mathbb{R}^{1 \times N}$ は状態を読み出し、$D \in \mathbb{R}$ は直接のスキップ接続である。

$\mathbf{A} = \operatorname{diag}(\lambda_1, \dots, \lambda_N)$ を対角行列に制限すると、Equation 12.21 は N 個の独立な Equation 12.20 のコピーへと帰着し、それぞれがレート $\lambda_1, \dots, \lambda_N$ で時間発展して $\mathbf{C}$ により線形結合される。密行列 $\mathbf{A}$ ではステップあたり $O(N^2)$、結合的なスキャンの結合演算あたり $O(N^3)$ のコストがかかるのに対し、対角行列 $\mathbf{A}$ のコストは $O(N)$ 回のスカラー演算で済む。複素数体上ではほとんどすべての行列が対角化可能であり、$\mathbf{B}, \mathbf{C}$ が固有ベクトルによる基底変換を吸収するため、対角構造にしても表現力はほとんど損なわれない (Gu, Gupta, et al. 2022)。実レート $\lambda_n < 0$ を用いると、純粋な実数演算で計算できる代わりに、振動モード（演習問題で扱う $\lambda = -\alpha + i\omega$）は失われる。

SSM 層は d 次元の入力系列を処理する際、このような N 状態の系を座標（**チャネル**）ごとに 1 つずつ独立に走らせるため、層全体の状態は合計 $d \times N$ 個の数値となる。これは状態数が N 倍（幅 d の minGRU との比較）であることを意味する。

12.4.1.2 離散化

Equation 12.21 を離散的なトークン系列に適用するため、ステップサイズ Δ の間隔で系をサンプリングし、ステップ t が連続時間 $t\Delta$ に対応するようにする。入力を x_t で区間 $((t-1)\Delta, t\Delta]$ にわたり一定に保つと（**ゼロ次ホールド**、略して ZOH の仮定）、Equation 12.20 を厳密に積分できる。$\frac{\mathrm{d}}{\mathrm{d}t}(h + x_t/\lambda) = \lambda(h + x_t/\lambda)$ が成り立つため、シフトした状態 $h + x_t/\lambda$ は 1 ステップの間に $e^{\Delta\lambda}$ 倍に減衰し、$h_t + x_t/\lambda = e^{\Delta\lambda}(h_{t-1} + x_t/\lambda)$ となる。これを整理すると、厳密な離散線形漸化式が得られる。

$$
\begin{aligned}
h_t &= \bar{a}\, h_{t-1} + \bar{b}\, x_t, \\
\bar{a} &= e^{\Delta\lambda}, \qquad \bar{b} = \frac{e^{\Delta\lambda} - 1}{\lambda}.
\end{aligned}
\tag{12.22}
$$

$\lambda = -1$ のとき $\bar{b} = 1 - \bar{a}$ となるため、Equation 12.22 は凸結合（convex combination）$h_t = \bar{a}\, h_{t-1} + (1 - \bar{a})\, x_t$ となり、最小 GRU の更新式（12.3.1.3 節）において $\bar{a} = e^{-\Delta}$ を更新ゲートとしたものと一致する。ステップサイズ Δ は記憶の長さを直接制御する。すなわち、$\lambda = -1$ のとき、$\Delta = 0.01$ では $\bar{a} = 0.990$（半減期 69 ステップ）、$\Delta = 0.1$ では $\bar{a} = 0.905$（半減期 7 ステップ）、$\Delta = 1$ では $\bar{a} = 0.37$ となる。

$\lambda = -e^{\theta} < 0$ および $\Delta > 0$ とパラメータ化すれば、シグモイドゲートと同様に $\bar{a} = e^{\Delta\lambda} \in (0, 1)$ が保証される。ここで新しいのは、半減期 $\ln 2/(\Delta|\lambda|)$ が積の形に分解される構造である（Figure 12.7）。N 個のレート $\lambda_1, \ldots, \lambda_N$ がチャネル内の状態を異なる相対的時間スケールへと分散させる一方、チャネルごとの単一のステップサイズ Δ が N 個すべての時間スケールを一括して引き伸ばしたり圧縮したりする。

対角行列 $\mathbf{A}$ に対しては、ZOH は要素ごとに適用され、$\bar{a}_n = e^{\Delta\lambda_n}$ および $\bar{b}_n = (e^{\Delta\lambda_n} - 1) b_n/\lambda_n$ となる（`expm1` を用いて計算することで $\Delta\lambda_n$ が小さいときの桁落ちを防ぐ。また、密な行列指数関数 $\bar{\mathbf{A}} = e^{\Delta\mathbf{A}}$ については 38.1.2.3 節を参照）。

実数値版の S4D は、$\lambda_n = -n$（$n = 1, \ldots, N$、HiPPO Legendre 射影行列の対角成分 (Gu et al. 2020)）と初期化し、各チャネルのステップサイズ Δ を $[0.001, 0.1]$ から対数一様に抽出する。$N = 16$ のとき、初期の半減期は半ステップ未満（$\Delta = 0.1, \lambda = -16$）から 693 ステップ（$\Delta = 0.001, \lambda = -1$）にまで及び、勾配ステップを 1 回も踏む前に Figure 12.7 の網掛けの帯をカバーする。

12.4.1.3 畳み込み形式

$\bar{a}_n$、$\bar{b}_n$、$\mathbf{C}$ がトークン位置 t によって変化しないとき、その系は**線形時不変**（linear time-invariant、LTI）である。Equation 12.22 を $\mathbf{h}_0 = \mathbf{0}$ から展開すると、状態 n は $\sum_{k \geq 0} \bar{a}_n^k \bar{b}_n x_{t-k}$ を累積し、読み出し $\mathbf{C}$ が N 個の状態を足し合わせて単一の**因果畳み込み**を形作ることがわかる。

$$
\begin{aligned}
y_t &= \sum_{k=0}^{t-1} K_k\, x_{t-k} + D x_t, \\
K_k &= \sum_{n=1}^{N} c_n \bar{a}_n^k \bar{b}_n.
\end{aligned}
\tag{12.23}
$$

各チャネルのインパルス応答（impulse response）$\mathbf{K} = (K_0, K_1, \ldots)$ は、N 個の減衰する指数関数の線形結合である。したがって、時不変 SSM には 3 つの等価な見方が存在する（Figure 12.8）。

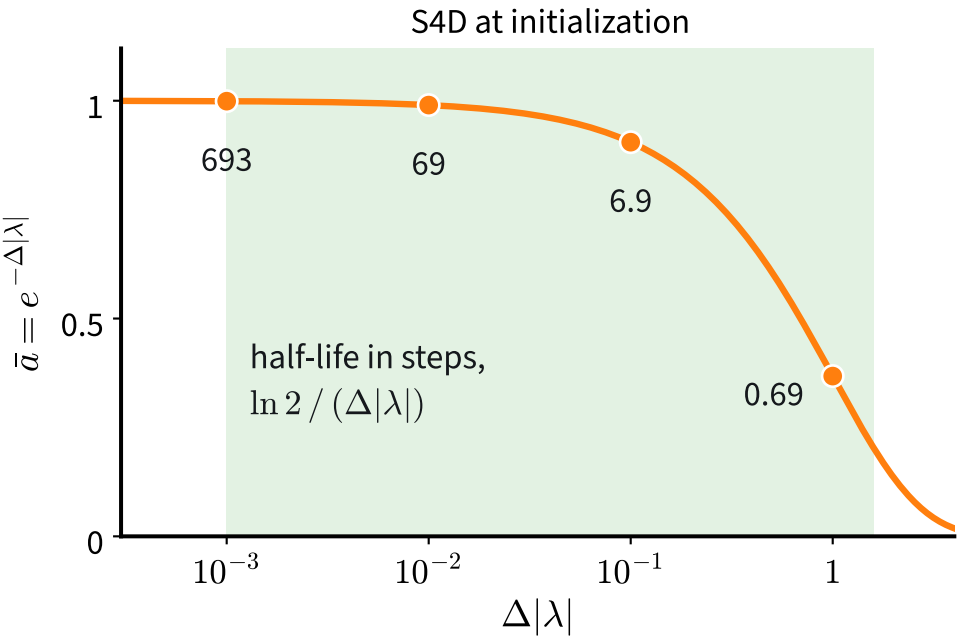


Figure 12.7: ゼロ次ホールドは実レート $\lambda < 0$ をステップあたりの減衰 $\bar{a} = e^{-\Delta|\lambda|}$ へと変換し、その値は積 $\Delta|\lambda|$ のみに依存する。図中のラベルはステップ単位の半減期 $\ln 2/(\Delta|\lambda|)$ を表す。網掛けの帯は S4D の初期化がカバーする範囲であり、Δ が 0.001 から 0.1 の間、$|\lambda|$ が 1 から 16 の間に対応する。

1. **連続時間常微分方程式**（continuous-time ordinary differential equation、ODE）：パラメータがレートとステップサイズを指定する。
2. **離散漸化式**：自己回帰生成の際に用いられ、トークンあたり $O(1)$ の計算時間と状態で動作する。
3. **因果畳み込み**：訓練時の全 T 位置にわたって、$O(T \log T)$ の計算量で高速フーリエ変換（fast Fourier transform、FFT）により評価される。S4 (Gu, Goel, et al. 2022) はこの畳み込みとしての見方を用いて、16,384 ステップの Path-X ベンチマークを解いた。なお、畳み込みとしての見方には時不変なダイナミクスが必要であるのに対し、12.3 節の並列スキャンはトークンごとに減衰が変化する場合でも機能する。

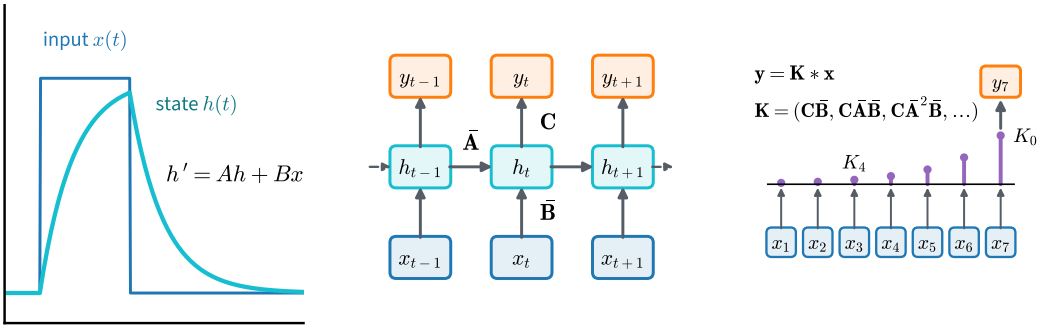


Figure 12.8: 1 つの状態空間モデルが持つ 3 つの形式。左：連続時間の状態が入力パルスに応答する。中央：離散化された系はトークンあたり 1 ステップの線形漸化式となる。右：展開すると、同じ系は単一のパルスに対する応答をカーネルとする因果畳み込みとなる。

12.4.1.4 時間スケールと学習されるカーネル

minGRU の初期ゲートバイアスを S4D の減衰に合わせて設定する以上に、複数状態を持つ対角 SSM は何をもたらすのだろうか。第 1 に、チャネルごとに単一のステップサイズ Δ が N 個すべての時間スケールを一括してスケーリングする点であり、12.5 節ではこれを入力依存にする。第 2 に、N 個の状態を読み出し $\mathbf{C}$ で結合することにより、各チャネルは畳み込みカーネル $K_k = \sum_{n=1}^{N} c_n \bar{a}_n^k \bar{b}_n$ を単一の単調減衰ではなく N 個の指数関数の和として形作ることができる。

N 個の実指数関数は、非単調な遅延カーネルをどこまで精密に近似できるだろうか。あるチャネルが 20 ステップ前の入力を読み取る必要があるとし、鋭いスパイク（$K_{20} = 1$ で他はゼロ）またはラグ 20 付近の滑らかなガウス型のバンプを考える。単一の状態（$N = 1$）は常にラグ 0 でピークをとる。以下では、読み出し重み $\mathbf{c}$ を $N \in \{1, 4, 16\}$ 個の状態（S4D のレート $\lambda_n = -n$ と固定の $\Delta = 0.05$）に対して最小二乗法で当てはめ、相対残差誤差 $\|\mathbf{K} - \text{target}\| / \|\text{target}\|$ と必要となる読み出し重みの大きさ $|\mathbf{c}|_\infty$ の両方を調べる。

```
def fit_kernel(N, target, dt=0.05):
    """Fit the readout c of N states at S4D's rates to a target kernel."""
    rate = -np.arange(1., N + 1)
    a_bar, b_bar = np.exp(dt * rate), np.expm1(dt * rate) / rate
    basis = a_bar ** np.arange(len(target))[:, None] * b_bar  # K = basis @ c
    c = np.linalg.lstsq(basis, target, rcond=None)[0]
    K = basis @ c
    return K, c, np.linalg.norm(K - target) / np.linalg.norm(target)
```

```
k = np.arange(100)
spike = (k == 20).astype(float)      # Exactly the input 20 steps back
bump = np.exp(-(k - 20) ** 2 / 32)   # About 20 steps back, give or take 4
print(' N  peak  spike |c|  spike residual  bump |c|  bump residual')
for N in [1, 4, 16]:
    K, c, spike_error = fit_kernel(N, spike)
    _, c_bump, bump_error = fit_kernel(N, bump)
    print(f'{N:2d}  {K.argmax():4d}  {np.abs(c).max():9.0e}  '
          f'{spike_error:14.2f}  {np.abs(c_bump).max():8.0e}  '
          f'{bump_error:13.2f}')
```

```
 N  peak  spike |c|  spike residual  bump |c|  bump residual
 1     0      7e-01            0.99     7e+00           0.90
 4    17      1e+02            0.97     1e+03           0.55
16    20      2e+11            0.89     8e+10           0.03
```

$N = 1$ では、カーネルはラグ 0 でピークをとり、どちらの目標も捉えられない。$N = 16$ では、ピークはラグ 20 へと移動し、滑らかなバンプには相対誤差 0.03 で適合する一方、鋭いスパイクの近似精度は依然として低い（誤差 0.89）。しかし、|c| に注目してほしい。実指数関数でこの適合を実現するには、符号が交互に入れ替わりほぼ相殺し合う 10^{11} 程度の読み出し重みが必要となる。これは実指数関数 $e^{-n\Delta k}$ が基底として極端に悪条件であるためである。単位スケール程度の重みから出発する勾配降下法ではこのような相殺には到達できないため、実数値 SSM のカーネルは時間ウィンドウにわたる広い平均にとどまる（対照的に、振動を伴う複素レート $\lambda = -\alpha + i\omega$ はフーリエ基底に似た直交基底を与える。演習問題 3 を参照）。

12.4.2 対角状態空間層

12.4.2.1 S4D 層

S4D 層は、N 個の実状態を d 個の各チャネルごとに保持する。レートを $\lambda_n = -e^{\theta_n} < 0$（log_neg_rate）、ステップサイズを log_dt によってパラメータ化し、$\mathbf{B} = \mathbf{1}$ に固定したうえで（積 $b_n c_n$ のみが Equation 12.23 に現れ、$\mathbf{C}$ が学習されるため）、$(\bar{\mathbf{a}}, \bar{\mathbf{b}})$ を discretize 内で Equation 12.22 により計算する。

```
class S4D(nnx.Module):
    """A diagonal state space layer: N states per channel, fixed dynamics."""
    def __init__(self, num_hiddens, num_states=16, dt_min=1e-3, dt_max=1e-1,
                 *, rngs):
        d, N = num_hiddens, num_states
        # lambda_n = -n for n = 1..N, stored as log(-lambda) to stay negative
        self.log_neg_rate = nnx.Param(
            jnp.tile(jnp.log(jnp.arange(1., N + 1)), (d, 1)))
        self.log_dt = nnx.Param(  # Log-uniform in [dt_min, dt_max]
            rngs.params.uniform((d, 1)) * math.log(dt_max / dt_min)
            + math.log(dt_min))
        self.C = nnx.Param(rngs.params.normal((d, N)) / math.sqrt(N))
        self.D = nnx.Param(jnp.ones(d))

    def discretize(self):
        """Zero-order hold with B = 1: decays a_bar and input weights b_bar."""
```

```
        rate = -jnp.exp(self.log_neg_rate[...])  # (d, N)
        dt_rate = jnp.exp(self.log_dt[...]) * rate
        return jnp.exp(dt_rate), jnp.expm1(dt_rate) / rate
```

訓練時の順伝播では、すべてのチャネルについて長さ T のカーネル（Equation 12.23）を構成し、ゼロパディングを施した長さ $2T$ の FFT によって因果畳み込みを実行する（これにより巡回的な回り込みを防ぐ）。

```
@d2l.add_to_class(S4D)
def __call__(self, X):  # X: (batch_size, num_steps, d)
    T = X.shape[1]
    a_bar, b_bar = self.discretize()
    # All powers a_bar^k for k < T: d * N * T numbers held in memory
    powers = a_bar[..., None] ** jnp.arange(T)
    K = ((self.C[...] * b_bar)[..., None] * powers).sum(1)  # Kernel, (d, T)
    Y = jnp.fft.irfft(jnp.fft.rfft(X, n=2 * T, axis=1) *
                      jnp.fft.rfft(K.T, n=2 * T, axis=0), n=2 * T, axis=1)
    return Y[:, :T] + self.D[...] * X
```

自己回帰生成の際には、`step` が $d \times N$ の再帰状態を 1 トークン分だけ、$O(dN)$ の計算時間と定数メモリで進める。

```
@d2l.add_to_class(S4D)
def step(self, x, h):
    """One step: x is (batch_size, d); the state h is (batch_size, d, N)."""
    a_bar, b_bar = self.discretize()
    h = a_bar * h + b_bar * x[..., None]
    return (h * self.C[...]).sum(-1) + self.D[...] * x, h
```

FFT 畳み込み、並列の `linear_scan`、および逐次的な `step` による評価が、単精度浮動小数点の許容誤差の範囲で一致することを確かめてみよう。

```
layer = S4D(8, num_states=4, rngs=nnx.Rngs(0))
X = jax.random.normal(jax.random.key(1), (2, 50, 8))  # (batch, T, d)
a_bar, b_bar = layer.discretize()
H = d2l.linear_scan(a_bar, b_bar * X[..., None])  # (batch, T, d, N)
Y_scan = (H * layer.C[...]).sum(-1) + layer.D[...] * X
h, Y_step = jnp.zeros((2, 8, 4)), []
for x in jnp.unstack(X, axis=1):
    y, h = layer.step(x, h)
    Y_step.append(y)
Y_conv = layer(X)
Y_step = jnp.stack(Y_step, 1)
print(f'convolution vs scan: {jnp.abs(Y_conv - Y_scan).max():.1e}, '
      f'convolution vs steps: {jnp.abs(Y_conv - Y_step).max():.1e}')
```

```
convolution vs scan: 2.4e-07, convolution vs steps: 2.4e-07
```

3 つの見方はすべて $\sim 10^{-7}$ の精度で一致する。ここで注目すべき点として、LTI 層では FFT 畳み込みを用いることで、バッチ全体にわたる 4 次元の (B, T, d, N) 中間状態テンソルを実体化せずに済み、(d, N, T) のカーネルのべき乗のみを保持すればよい。

12.4.2.2 長距離想起

初期の時間スケールが長距離記憶を左右するかどうかを検証するため、12.1 節の遅延想起ベンチマークを再び取り上げ、遅延 $T = 100$ および $T = 1{,}000$ ステップで実験する。ここでは同一の空白フィラー（`blank_fillers=True`）を用い、内容に基づくフィルタリング（12.5 節）ではなく純粋な保持期間をテストできるようにする。

層の種類と初期の時間スケールを掛け合わせた 4 つの設定を比較する（Figure 12.9）。

1. **デフォルトの `minGRU`**：ゲートが $1/2$ 付近から出発する。
2. **デフォルトの `S4D`**：$d = 64, N = 16$、$\Delta \in [0.001, 0.1]$ であり、半減期は最大 693 ステップに達する。
3. **`S4D, short timescales`**：$\Delta \in [0.1, 1.0]$ で初期化し、初期の半減期がいずれも 7 ステップを超えないようにする。
4. **`minGRU, S4D timescales`**：更新ゲートのバイアスを `init_timescales` によって初期化し、$\sigma(b) = e^{-\Delta}$ が $\Delta \in [0.001, 0.1]$ に対して成り立つようにすることで（$b = -\log(e^{\Delta} - 1)$）、S4D の各チャネルの最も遅い状態に合わせる。

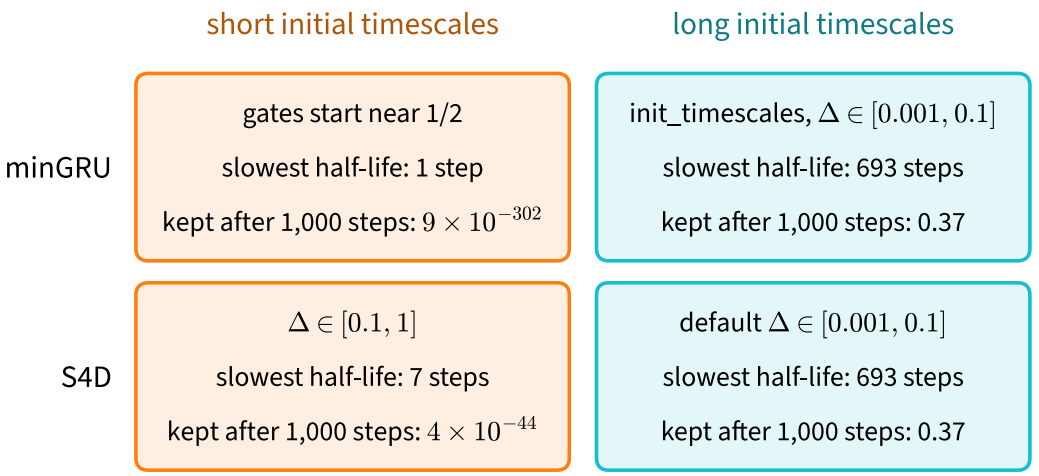


Figure 12.9: 長距離想起実験における 4 つの設定。2 種類の層（minGRU と S4D）のそれぞれに対し、短い初期時間スケールと長い初期時間スケールを組み合わせる。各セルには、初期化時の最も遅い半減期と、1,000 ステップの遅延後にその最も遅い状態が保持している格納値の割合を示す。時間スケールが結果を決めるなら列ごとに差が現れ、層の種類が決めるなら行ごとに差が現れる。

```
def init_timescales(layer, dt_min=1e-3, dt_max=1e-1, *, rngs):
    """Start a minGRU's gates at the decays exp(-dt) of S4D."""
    dt = dt_min * (dt_max / dt_min) ** rngs.params.uniform(
        layer.W_z.bias.shape)
    layer.W_z.bias[...] = -jnp.log(jnp.expm1(dt))  # sigmoid(b) = exp(-dt)
    return layer
```

各モデルにつき 2 つのシードを `d2l.train_delayed_recall`（1,500 ステップの Adam、学習率 3×10^{-3}）で訓練し、訓練後の最も遅い半減期 $\ln 2 / \min_{d,n}(\Delta|\lambda_n|)$ も 2 種類の S4D について出力する。

```
# (layer, embedding width): S4D keeps its width; the minGRUs map 32 to 64
layers = {
    'minGRU': lambda rngs: (d2l.MinGRU(32, 64, rngs=rngs), 32),
    'S4D': lambda rngs: (S4D(64, rngs=rngs), 64),
    'S4D, short timescales': lambda rngs: (
        S4D(64, dt_min=0.1, dt_max=1.0, rngs=rngs), 64),
    'minGRU, S4D timescales': lambda rngs: (init_timescales(
        d2l.MinGRU(32, 64, rngs=rngs), rngs=rngs), 32)}

def recall(make_layer, T):
    """Train a fresh model at delay T; return its test accuracy and layer."""
    rngs = nnx.Rngs(d2l.get_key())
    layer, num_inputs = make_layer(rngs)
    model = d2l.RecallModel(layer, 64, num_inputs=num_inputs, rngs=rngs)
    return d2l.train_delayed_recall(model, T, blank_fillers=True), layer

def slowest_half_life(layer):  # ln 2 / min(dt |lambda|), in steps
    rates = jnp.exp(layer.log_dt[...] + layer.log_neg_rate[...])
    return math.log(2) / rates.min().item()

runs = {name: {T: [recall(make_layer, T) for _ in range(2)]  # Two runs each
               for T in [100, 1000]} for name, make_layer in layers.items()}
print(f"{'accuracy':23s} {'T=100':^11s}  {'T=1000':^11s}")
for name, by_T in runs.items():
    print(f'{name:23s}', '  '.join(' '.join(f'{acc:5.2f}' for acc, _ in r)
                                    for r in by_T.values()))
print('slowest half-life after training, in steps')
for name in ['S4D', 'S4D, short timescales']:
    print(f'{name:23s}', '  '.join(' '.join(
        f'{slowest_half_life(layer):5.0f}' for _, layer in r)
        for r in runs[name].values()))
```

```
accuracy                  T=100       T=1000
minGRU                  0.12  0.12   0.12  0.13
S4D                     1.00  1.00   1.00  1.00
S4D, short timescales   1.00  1.00   0.12  0.12
minGRU, S4D timescales  1.00  1.00   1.00  1.00
slowest half-life after training, in steps
S4D                     1318  1336   2216  2595
S4D, short timescales    147   134      8     7
```

結果は層の種類ではなく初期の時間スケールに従っている。$T = 1{,}000$ では、`S4D` と `minGRU`，`S4D timescales` の両方がすべての試行で 100% の正解率を達成するのに対し、デフォルトの `minGRU` と `S4D, short timescales` はランダム予測の水準 (0.12) にとどまったままである。なお、`S4D, short timescales` が $T = 100$ では成功する理由に注目しよう。初期の最も遅い半減期は 7 ステップにすぎないものの、99 ステップにわたる減衰は $e^{-0.1\times 99} \approx 5 \times 10^{-5}$ であり、これは float32 の精度を上回るため、Adam が最も遅い半減期を 100〜200 ステップまで引き伸ばすのに十分な勾配信号が供給される。しかし、999 ステップにわたると $e^{-100} \approx 10^{-44}$ となって機械精度を下回って消失し、最も遅い半減期は 7 ステップ付近で凍りついたままとなる。

とはいえ、時不変なダイナミクスには根本的な死角がある。ラグ k にあるすべてのトークンは、それが決定的に重要な記号であれ情報を持たないフィラーであれ、まったく同じ重み K_k を受け取ってしまうのである。12.5 節では、各トークンが自身のステップサイズ Δ_t を選べるようにす

ることで、S4D の多重時間スケールの状態バンクと入力依存の選択機構を組み合わせる。

12.4.3 まとめ

連続時間の対角状態空間モデル $\mathbf{h}'(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}x(t)$（レート $\lambda_n < 0$）をステップサイズ Δ のゼロ次ホールドで離散化すると、減衰 $\bar{a}_n = e^{\Delta\lambda_n}$ と半減期 $\ln 2/(\Delta|\lambda_n|)$ を持つ線形漸化式が得られる。$\lambda_n = -n$ と初期化し、Δ を $[0.001, 0.1]$ にわたって対数一様に抽出することで、S4D の各チャネルは N 個の指数減衰状態を備え、その半減期は 1 ステップ未満からほぼ 700 ステップにまで及ぶ。パラメータが時不変であるため、S4D は推論時には $O(1)$ 状態の漸化式として、訓練時には FFT による因果畳み込みとして等価に評価できる。1,000 ステップの遅延想起では、長い時間スケールの減衰から出発すること（S4D であれ minGRU のゲートバイアスへ移し替えたものであれ）が学習を可能にする決定的な要因となるが、固定されたダイナミクスでは依然として無関係な入力を選択的に無視することはできない。

12.4.4 演習問題

1. [code] **離散化規則**。ゼロ次ホールドは、Equation 12.20 を漸化式に変換するいくつかの方法の 1 つである。前進オイラー法は $\bar{a} = 1 + \Delta\lambda$、後退オイラー法は $\bar{a} = 1/(1 - \Delta\lambda)$、双一次法は $\bar{a} = (1 + \Delta\lambda/2)/(1 - \Delta\lambda/2)$ を用いる（38.2.3 節では、これら 2 つのオイラー法を微分方程式のソルバとして扱う）。
   1. 各規則について、どのようなステップサイズ Δ であれば、すべての $\lambda < 0$ に対して $|\bar{a}| < 1$ となるかを求めよ。安定な連続系を不安定にし得る規則はどれか。
   2. $\lambda = -1$ の後退オイラー法が、更新式 $h_t = z h_{t-1} + (1 - z)x_t$（ただし $z = \sigma(-\log \Delta)$）を与えること、すなわちステップサイズの負の対数を更新ゲートの活性化前の値とする GRU の混合式になることを示せ。
   3. $\lambda = -1$ および $\Delta \in \{0.1, 1, 3\}$ に対して、単位ステップ入力に対する各規則の応答をシミュレーションし、厳密解 $1 - e^{-t}$ と比較せよ。
2. [code] **遅延とカーネル**。`fit_kernel` を用いて、N 個の実指数関数がどのような記憶を表現できるかを調べよ。
   1. 遅延 5、20、50 における単位スパイクに対し、$N = 16$ 個の状態と $\Delta = 0.05$ で当てはめよ。遅延に応じて誤差はどのように変化するか。最も遅い状態と最も速い状態の半減期に関連付けて答えよ。
   2. 遅延 20 のスパイクに対し、ステップサイズ Δ を 0.01 から 0.2 の間で変化させて当てはめよ。どの Δ が最もよく適合するか。この結果は、特定の長さの記憶を必要とするチャネルのステップサイズを、訓練された S4D がどのように設定すべきかについて何を示唆するか。
   3. $\mathbf{c}$ を最小二乗法の代わりにゼロからの勾配降下法（Adam、数千ステップ）で当てはめよ。バンプとスパイクのそれぞれについて、最小二乗法の誤差にどこまで近づくか。最小二乗法で当てはめた重みから何が予測されるか。
3. [code] **複素レート**。複素レート $\lambda = -\alpha + i\omega$（$\alpha > 0$）を持つ状態は、回転しながら減衰する。
   1. ゼロ次ホールドにより $\bar{a} = e^{-\alpha\Delta}e^{i\omega\Delta}$ となること、および共役なレート・入力重み・読み出し重みを持つ 2 つの状態が実カーネル $K_k = r\, e^{-\alpha\Delta k}\cos(\omega\Delta k + \varphi)$（ただし $r \geq 0$、φ はある定数）として寄与することを示せ。このようなペアは何個の実パラメータを持つか。また、実状態のペアは何個持つか。
   2. 実レートでは表現できず、このようなペアで表現できるカーネルはどのようなものか。一定の周期で繰り返す入力を考えてみよ。
   3. `fit_kernel` を変更し、複素レート $\lambda_m = -1/2 + i\pi m$（$m = 0, \ldots, N - 1$、Gu, Gupta, et al. (2022) の S4D-Lin 初期化）を用いてカーネルの実部を当てはめるようにせよ。遅延 20 のスパイクについて、誤差と重みの大きさを実レートによる当てはめ

と比較し、その違いを説明せよ。

4. [code] **FFT による畳み込み**。`S4D` の順伝播では、FFT の前に系列とカーネルを長さ $2T$ までパディングする。
   1. パディングを取り除いて FFT の長さを T にし、その出力を `step` の出力と比較せよ。どの出力が誤っているか、またそこに何が加算されているか。
   2. 厳密な因果畳み込みを与える最短のパディング長はいくつか。実装において、それを 2 のべき乗に切り上げることが多いのはなぜか。

12.5 選択的状態空間モデル

12.4 節の S4D 層は、ラグ k にあるすべてのトークンを同一に扱う。すなわち、k ステップ前の入力が段落の主要な名詞であろうとカンマであろうと、現在の出力には固定された畳み込み重み K_k で寄与する。しかし自然言語には**内容に基づく選択**（content-based selection）が不可欠である。大半のトークンはすぐに破棄してよい一方、少数の重要なトークンは強く書き込み、文ごとに異なる位置から数百ステップにわたって保持しなければならない。状態空間モデル（state space model、SSM）は、何をどのくらいの期間保持するかをトークンごとに決定できるだろうか。

前節までに登場した 2 つのモデルは、それぞれこの問いに対する答えの半分を備えている。

- `MinGRU`（12.3 節）は現在の入力トークンから減衰ゲートを計算するため、あるトークンを保持して次のトークンで上書きできるが、ゲートの初期値は 1 ステップの半減期であり、各チャネルは単一のスカラー状態しか持たない。
- `S4D`（12.4 節）は初期状態で 1 から ~ 700 ステップにわたる半減期をチャネルあたり N 個の状態に持つが、そのダイナミクスは時不変（time-invariant）である。

選択的状態空間モデル（selective state space model）は Gu and Dao (2023) が Mamba アーキテクチャとともに導入したものであり、これら 2 つを統合する。離散化された SSM のステップサイズ Δ_t は、書き込みベクトル $\mathbf{B}_t$ および読み出しベクトル $\mathbf{C}_t$ とともに、現在の入力 $\mathbf{x}_t$ の関数となる。$\Delta_t \to 0$ にすると状態は凍結され、情報を持たないフィラートークンは無視される。一方、Δ_t を大きくすると重要なトークンが書き込まれ、それまで状態が保持していた内容は減衰する。ダイナミクスがステップごとに変化するため FFT 畳み込みはもはや適用できないが、12.3 節の並列結合スキャン（parallel associative scan）を用いれば、依然として $O(\log_2 T)$ ラウンドで層を訓練できる。以下では、選択的コピーにおいて `S4D` と `MinGRU` をベンチマーク評価し、1 状態の選択的 SSM が `MinGRU` のゲートと数学的に等価であることを示す。続いて、`SelectiveSSM` と `Mamba` を省メモリなカスタム逆方向スキャン（`adjoint_scan`）によって構築し、訓練済みの Mamba 層がどのように状態を形成するかを調べ、最後にバイトレベルの WikiText-103 で評価する。

12.5.1 選択的コピー

Gu and Dao (2023) による合成ベンチマークの**選択的コピー**（selective copying）は、内容依存の保持能力を切り出して評価する（Figure 12.10）。ここでの設定では、系列は 1,024 個のフィラートークン（`0`）からなり、その中に 4 個のマーク付き記号（トークン `2..9` から一様に抽出）を**ランダム**な位置に挿入し、末尾に 4 個のクエリトークン（`1`）を続ける。j 番目のクエリトークンにおいて、モデルは j 番目のマーク付き記号を順番に出力しなければならない（ランダム予測の正解率は $1/8 = 0.125$）。記号の位置は系列ごとにランダムに変化するため、ラグ $T - s$ のみによって入力を重み付けする時不変フィルタは必ず失敗する。

```
def copy_batch(key, batch_size, num_fillers=1024, num_marked=4,
               num_symbols=8):
```

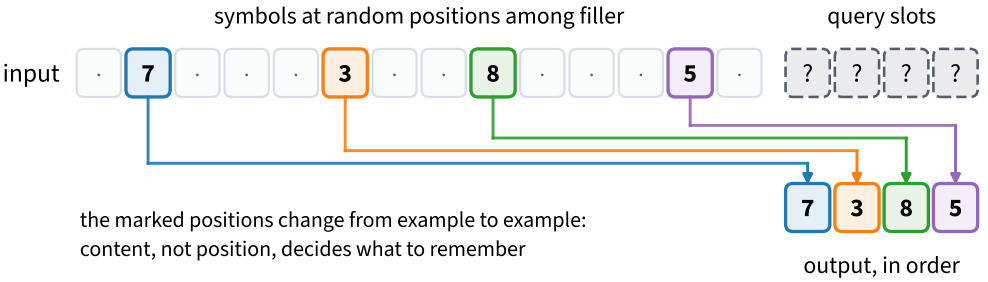


Figure 12.10: 選択的コピータスク。少数のマーク付き記号（色付き）が、系列ごとに変わる位置でフィラートークン（灰色）の中に散在している。クエリトークンの位置で、モデルはそれらの記号を順番に出力しなければならない。

```
    """Symbols at random positions among fillers, then query tokens."""
    key_y, key_p = jax.random.split(key)
    Y = jax.random.randint(key_y, (batch_size, num_marked), 2, 2 + num_symbols)
    positions = jnp.argsort(
        jax.random.uniform(key_p, (batch_size, num_fillers)), 1)
    X = jnp.zeros((batch_size, num_fillers + num_marked), dtype=jnp.int32)
    X = X.at[:, num_fillers:].set(1)  # Query tokens
    rows = jnp.arange(batch_size)[:, None]
    X = X.at[rows, jnp.sort(positions[:, :num_marked], 1)].set(Y)
    return X, Y

X, Y = copy_batch(d2l.get_key(), 2, num_fillers=12)
print(X, Y, sep='\n')
```

```
[[0 0 8 0 8 2 0 0 0 3 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 3 2 7 0 6 1 1 1 1]]
[[8 8 2 3]
 [3 2 7 6]]
```

選択的コピーでモデルを評価するため、`train_copy` は 1,500 ステップの Adam（`lr=3e-3`、バッチサイズ 64）にわたり `d2l.train_queries` を用いてネットワークを訓練し、新たに生成した 1,024 本のテスト系列に対するクエリ正解率を評価する。

```
def train_copy(model):
    """Train for 1,500 steps on fresh batches of selective copying; return
    the test accuracy."""
    num_steps = 1500
    optimizer = nnx.Optimizer(model, optax.chain(
        optax.clip_by_global_norm(1.0), optax.adam(3e-3)), wrt=nnx.Param)
    key = d2l.get_key()  # New data every run
    d2l.train_queries(model, lambda step: copy_batch(
        jax.random.fold_in(key, step), 64), num_steps, optimizer)
    X, Y = copy_batch(jax.random.fold_in(key, num_steps), 1024)
    return float(jnp.stack([d2l.query_accuracy(model, X[i:i + 128],  # In parts
                                               Y[i:i + 128])
```

```
                for i in range(0, 1024, 128)]).mean())
```

ラッパ関数 copy_runs は、1 つのアーキテクチャを独立したシードで num_runs=2 回訓練し、パラメータ数とテスト正解率を表示して、訓練済みモデルを返す。

```
def copy_runs(name, make_model, num_runs=2):
    """Train num_runs fresh models; print their accuracies; return them."""
    models = [make_model() for _ in range(num_runs)]
    accuracies = [train_copy(model) for model in models]
    print(f'{name:20s} {d2l.num_params(models[0]):6,} parameters, accuracy '
          + ', '.join(f'{acc:.2f}' for acc in accuracies))
    return models
```

まず、2 ブロックの d2l.SeqLM モデル（幅 $d = 48$）において、時不変な d2l.S4D と入力選択的な d2l.MinGRU を系列ミキサとして比較ベンチマークする。

```
rngs = nnx.Rngs(d2l.get_key())
models = {}
for name, mixer in (('S4D', lambda: d2l.S4D(48, rngs=rngs)),
                    ('minGRU', lambda: d2l.MinGRU(48, 48, rngs=rngs))):
    models[name] = copy_runs(name, lambda: d2l.SeqLM(
        10, 48, [mixer() for _ in range(2)], rngs=rngs))
```

```
S4D                  42,058 parameters, accuracy 0.17, 0.18
minGRU               48,202 parameters, accuracy 0.93, 0.78
```

予測どおり、時不変な S4D はランダム予測の水準付近（≤ 0.20）にとどまる一方、入力選択的な minGRU は実行ごとに約 0.75 から 0.95 の間に達する。ゲートの初期半減期が 1 ステップである（そのため 1,000 ステップの固定遅延想起では 12.4 節で失敗した）にもかかわらず、ここで minGRU が成功する理由に注目しよう。選択的コピーでは、マーク付き記号がたまたまクエリの近くに配置されることがあり、そこから短いホライズンの勾配が得られるため、より長いフィラー区間を越えて記号を保持するようゲートが段階的に学習していくのである（演習問題 6）。

12.5.2 選択的ダイナミクス

12.5.2.1 入力依存のステップサイズ

選択的 SSM を 2 段階で構築する。まず、単一の状態（$N = 1$）とレート $\lambda = -1$ を持つ単一チャネルを、ゼロ次ホールド（zero-order hold、ZOH、12.4.1.2 節）のもとで考えよう。その更新式は $h_t = e^{-\Delta} h_{t-1} + (1 - e^{-\Delta}) x_t$ である。ここで Δ を固定パラメータとして扱う代わりに、$\Delta_t = \mathrm{softplus}(s_t) = \log(1 + e^{s_t}) > 0$ を現在のトークンの線形射影 $s_t = \mathbf{w}^\top \mathbf{x}_t + b$ から計算する。$-\Delta_t$ の指数をとると、厳密な代数的恒等式が得られる。

$$e^{-\Delta_t} = e^{-\log(1+e^{s_t})} = \frac{1}{1+e^{s_t}} = \sigma(-s_t). \tag{12.24}$$

$z_t = \sigma(-s_t)$ とおくと、ZOH の更新式は $h_t = z_t h_{t-1} + (1 - z_t) x_t$ となり、MinGRU の厳密な凸結合と一致する（12.3.1.3 節、Gu and Dao (2023) の定理 1）。フィラートークンが $s_t = -4.6$ を生じるとき、$\Delta_t \approx 0.01$ かつ $z_t = 0.99$ となり（状態の 99% を保持し、半減期は 69 ステップ）、マーク付き記号が $s_t = 2$ を生じるとき、$\Delta_t \approx 2.1$ かつ $z_t \approx 0.12$ となる（新しい記号で

状態を上書きする)。また、$\Delta_t = \text{softplus}(s_t)$ を介してゲートをパラメータ化すると、S4D の多重時間スケールの初期化も直ちに適用できる。バイアス b を $\text{softplus}(b) \in [0.001, 0.1]$ となるように選べば、ゲートの初期半減期は 7 から 693 ステップとなり、さらに重要な点として、単一のチャネルステップサイズ $\Delta_{t,c}$ によって N 個の状態(それぞれ異なるレート $\lambda_1, \dots, \lambda_N$ を持つ)を同時に駆動できる。

12.5.2.2 書き込みベクトルと読み出しベクトル

次に、層の H 個のチャネル(Mamba では $H = 2d$)のそれぞれに、N 次元の状態ベクトル $\mathbf{h}_t^{(c)} \in \mathbb{R}^N$ を与え、レート $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_N)^\top$ を $(-1, \dots, -N)^\top$ に初期化する。現在の H 次元入力 $\mathbf{x}_t$ から、層はチャネルごとのステップサイズ $\boldsymbol{\Delta}_t \in \mathbb{R}_+^H$ と、全チャネルで共有される書き込み方向 $\mathbf{B}_t \in \mathbb{R}^N$ および読み出し方向 $\mathbf{C}_t \in \mathbb{R}^N$ を計算する。

$$\begin{aligned} \boldsymbol{\Delta}_t &= \text{softplus}(\mathbf{W}_\Delta \mathbf{x}_t + \mathbf{b}_\Delta), \\ \mathbf{B}_t &= \mathbf{W}_B \mathbf{x}_t, \qquad \mathbf{C}_t = \mathbf{W}_C \mathbf{x}_t. \end{aligned}$$

続いて各チャネル $c \in \{1, \dots, H\}$ は、自身の N 個の状態を更新し、次式によって出力を読み出す。

$$\begin{aligned} \mathbf{h}_t^{(c)} &= e^{\Delta_{t,c}\boldsymbol{\lambda}} \odot \mathbf{h}_{t-1}^{(c)} + \Delta_{t,c}\, x_{t,c}\, \mathbf{B}_t, \\ y_{t,c} &= \mathbf{C}_t^\top \mathbf{h}_t^{(c)} + D_c\, x_{t,c}. \end{aligned} \tag{12.25}$$

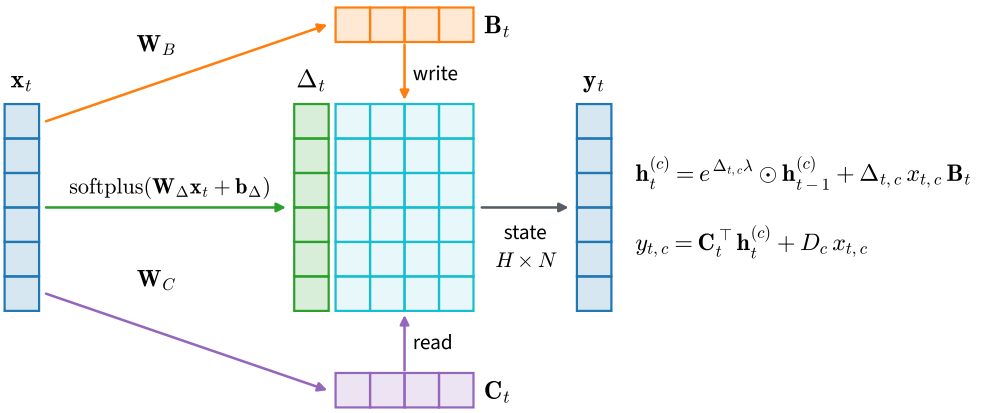


Figure 12.11: 選択的 SSM の 1 ステップ。入力 $\mathbf{x}_t$(H チャネル)から、3 つの射影によってステップサイズ $\boldsymbol{\Delta}_t$(チャネルごとに 1 つ)、書き込みベクトル $\mathbf{B}_t$、読み出しベクトル $\mathbf{C}_t$(いずれも長さ N)を計算する。第 c 行($H \times N$ 状態の行)は $e^{\Delta_{t,c}\lambda}$ だけ減衰して $\Delta_{t,c}\, x_{t,c}\, \mathbf{B}_t$ を受け取り、出力 $y_{t,c}$ はその行を $\mathbf{C}_t$ で読み出してスキップ項 $D_c\, x_{t,c}$ を加える。

Figure 12.11 は 3 つの射影がそれぞれどこに作用するかを示している。$\boldsymbol{\Delta}_t$ は状態の各行をチャネルごとにスケーリングする一方、$\mathbf{B}_t$ と $\mathbf{C}_t$ は列方向に作用し、すべてのチャネルで共有される。Gu and Dao (2023) に従い、Equation 12.25 では厳密な ZOH 減衰 $e^{\Delta_{t,c}\lambda}$ を維持する一方、ZOH の入力重み $(e^{\Delta\lambda} - 1)/\lambda$ を一次のオイラー重み $\Delta_{t,c}$ で近似している(これは $\Delta|\lambda| \ll 1$ のとき

に一致し、大きなステップ $\Delta_{t,c}$ において $1/|\lambda|$ を超える重みでの書き込みを可能にする。演習問題 1 を参照)。Equation 12.25 に現れるすべての量の形状を以下にまとめる。

量	形状	計算方法
入力 $\mathbf{x}_t$	H	ステップ t における層の入力
ステップサイズ Δ_t	H（チャネルごとに 1 つ）	$\mathbf{x}_t$ からランク R 分解された $\mathbf{W}_\Delta \in \mathbb{R}^{H\times H}$ を経由
書き込み・読み出しベクトル $\mathbf{B}_t$、$\mathbf{C}_t$	各 N（全チャネルで共有）	$\mathbf{x}_t$ から $\mathbf{W}_B, \mathbf{W}_C \in \mathbb{R}^{N\times H}$ を経由
レート λ	チャネルあたり N	$-1, \dots, -N$ に初期化された学習パラメータ
状態 $\mathbf{h}_t^{(1)}, \dots, \mathbf{h}_t^{(H)}$	$H \times N$	Equation 12.25 の漸化式
出力 $\mathbf{y}_t$	H	$\mathbf{C}_t$ による読み出しとスキップ項 $D_c x_{t,c}$ の和

12.5.3 Mamba 層

12.5.3.1 射影・畳み込み・ゲート

Mamba ブロック (Gu and Dao 2023) は、チャネル混合の射影、幅 4 のデプスワイズ因果畳み込み（depthwise causal convolution）、および SwiGLU 風の乗法的な出力ゲートの内側に選択的 SSM を組み込む（Figure 12.12）。

1. 入力射影 $\mathbf{W}_{\text{in}}$ が、d 次元の入力を幅 $H = 2d$ の 2 つのブランチに拡張する。
2. 最初のブランチは、選択的 SSM に入る前に幅 4 のデプスワイズ因果畳み込みと SiLU 活性化関数を通過する。これにより、$\Delta_t, \mathbf{B}_t, \mathbf{C}_t$ は孤立した単一トークンではなく 4 トークンの局所ウィンドウに条件付けられる。
3. SSM の出力は、もう一方のブランチの SiLU 活性化値と要素ごとに掛け合わされた後、幅 d へ出力射影 $\mathbf{W}_{\text{out}}$ によって戻される。
4. パラメータ数を節約するため、$\mathbf{W}_\Delta \in \mathbb{R}^{H\times H}$ はボトルネックランク $R = \lceil d/16 \rceil$ を経由して分解される。

12.5.3.2 スキャンによる訓練

Equation 12.25 は依然として各ステップにおけるアフィン漸化式 $\mathbf{h}_t = \mathbf{a}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_t$（$H \times N$ の状態上）であるため、`d2l.linear_scan` によって全 T ステップの状態を並列に計算できる。しかし、状態をトークンあたり $H \times N$ 個の数値に拡張すると、活性化メモリは $2N$ 倍に増大する。倍増スキャンの $\lceil \log_2 T \rceil$ ラウンドを素朴に自動微分すると、$\sim 2\lceil \log_2 T \rceil$ 個もの形状 (B, T, H, N) のテンソルが保持されてしまう（本節の小さなコピーモデルですら層あたり ~ 2 GB、LLM スケールでは数百 GB に達する）。

逆伝播される状態勾配 $\mathbf{g}_t = \partial \ell / \partial \mathbf{h}_t$ が次の逆時間アフィン漸化式を満たすことに注目すれば、中間スキャンラウンドの保存を回避できる。

$$\mathbf{g}_t = \left.\frac{\partial \ell}{\partial \mathbf{h}_t}\right|_{\text{direct}} + \mathbf{a}_{t+1} \odot \mathbf{g}_{t+1},$$

このときパラメータ勾配は $\partial \ell / \partial \mathbf{b}_t = \mathbf{g}_t$ および $\partial \ell / \partial \mathbf{a}_t = \mathbf{g}_t \odot \mathbf{h}_{t-1}$ となる。したがって逆伝播は、`linear_scan` を逆時間に実行する 2 回目の呼び出しにすぎず、保存が必要なのは順伝播の減衰 $\mathbf{a}$ と状態 $\mathbf{h}$ だけである（~ 20 個ではなく 2 個のテンソルで済む。なお Mamba の融合 CUDA

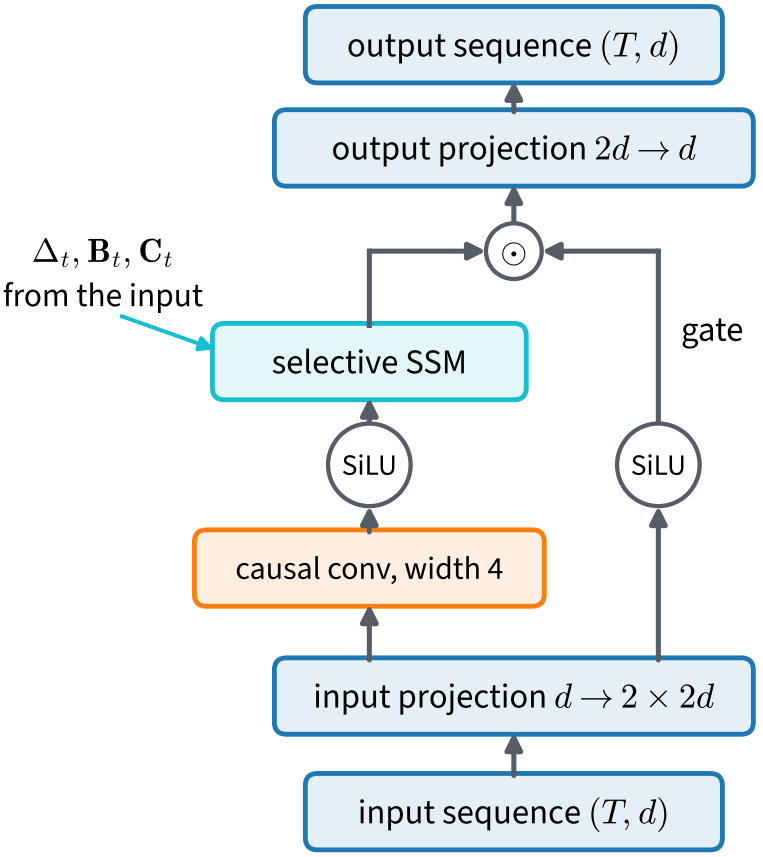


Figure 12.12: Mamba 層。入力は幅 $H = 2d$ の 2 つのブランチへ射影される。一方は短い因果畳み込み、SiLU、および選択的 SSM（ステップサイズと書き込み・読み出しベクトルをその入力から計算する）を通過し、もう一方は SiLU を通過してその結果をゲート制御する。最後の射影で幅 d に戻す。

カーネルはさらに一歩進め、**h** をオンチップ SRAM に保持し逆伝播中に再計算する。これについては 13.4 節で解説している)。カスタム逆伝播を備えた adjoint_scan を実装し、標準の自動微分と一致するか検証してみよう。

```
@jax.custom_vjp
def _adjoint_scan(a, b):
    return d2l.linear_scan(a, b)

def _adjoint_scan_forward(a, b):
    """Forward pass: the states, and what the backward pass needs."""
    h = d2l.linear_scan(a, b)
    return h, (a, h)

def _adjoint_scan_backward(saved, grad_h):
    """Backward pass: gradients of a and b by a scan in reverse time."""
    a, h = saved
    # g_t = grad_h_t + a_{t+1} g_{t+1}: a linear recurrence from the end
    a_next = jnp.concatenate([a[:, 1:], jnp.ones_like(a[:, :1])], 1)
    g = d2l.linear_scan(a_next[:, ::-1], grad_h[:, ::-1])[:, ::-1]
    h_prev = jnp.concatenate([jnp.zeros_like(h[:, :1]), h[:, :-1]], 1)
    return g * h_prev, g

_adjoint_scan.defvjp(_adjoint_scan_forward, _adjoint_scan_backward)

def adjoint_scan(a, b):
    """All states of h_t = a_t * h_{t-1} + b_t, differentiated by a scan."""
    return _adjoint_scan(jnp.broadcast_to(a, b.shape), b)
```

長さ $T = 100$ のランダム系列上で、adjoint_scan の微分が d2l.linear_scan を通した標準の自動微分と float32 の精度で一致することを検証する。

```
key_a, key_b, key_w = jax.random.split(d2l.get_key(), 3)
a = jax.random.uniform(key_a, (2, 100, 8))
b = jax.random.normal(key_b, (2, 100, 8))
w = jax.random.normal(key_w, (2, 100, 8))  # A loss that weights every state
grad = lambda scan: jax.grad(lambda a, b: (scan(a, b) * w).sum(), (0, 1))
deviation = max(float(jnp.abs(g1 - g2).max()) for g1, g2 in
                zip(grad(d2l.linear_scan)(a, b), grad(adjoint_scan)(a, b)))
print(f'max deviation: {deviation:.1e}')
```

```
max deviation: 9.5e-07
```

カスタムの逆時間スキャンによる勾配は、自動微分と $\sim 10^{-6}$ の精度で一致する。

12.5.3.3 実装

SelectiveSSM は Equation 12.25 を実装し、dt_bias を初期化して $\mathrm{softplus}(\mathrm{dt_bias})$ が $[\mathrm{dt_min}, \mathrm{dt_max}] = [0.001, 0.1]$ 上で対数一様に分布するようにする。selective=False を渡すと、$\Delta_t, \mathbf{B}_t, \mathbf{C}_t$ はチャネルごとの学習定数に置き換えられる(まったく同じ Mamba ブロック内での LTI 対照モデルとなる)。

```
class SelectiveSSM(nnx.Module):
    """A diagonal SSM with N states for each of num_channels channels. Its
    step sizes and write and read vectors depend on the input, or, with
    selective=False, are learned constants of each channel."""
    def __init__(self, num_channels, dt_rank, num_states=16, dt_min=1e-3,
                 dt_max=1e-1, selective=True, *, rngs):
        H, N, R = num_channels, num_states, dt_rank
        self.selective = selective
        self.log_neg_rate = nnx.Param(  # Rates -1, ..., -N in every channel
            jnp.tile(jnp.log(jnp.arange(1., N + 1)), (H, 1)))
        def dt_bias(key, shape, dtype=jnp.float32):  # softplus(bias) = dt
            dt = dt_min * (dt_max / dt_min) ** jax.random.uniform(key, shape)
            return (dt + jnp.log(-jnp.expm1(-dt))).astype(dtype)
        if selective:
            def small(key, shape, dtype=jnp.float32):  # U(-R^-1/2, R^-1/2)
                return jax.random.uniform(key, shape, dtype, -R ** -0.5,
                                          R ** -0.5)
            self.W_dt = nnx.Sequential(  # Step size per channel, rank R
                nnx.Linear(H, R, use_bias=False, rngs=rngs),
                nnx.Linear(R, H, kernel_init=small, bias_init=dt_bias,
                           rngs=rngs))
            self.W_B = nnx.Linear(H, N, use_bias=False, rngs=rngs)  # Write
            self.W_C = nnx.Linear(H, N, use_bias=False, rngs=rngs)  # Read
        else:  # Fixed dynamics: S4D with a learned write vector
            self.dt_bias = nnx.Param(dt_bias(rngs.params(), (H,)))
            self.B = nnx.Param(jnp.ones((H, N)))
            self.C = nnx.Param(rngs.params.normal((H, N)) / math.sqrt(N))
        self.D = nnx.Param(jnp.ones(H))
```

次に、`log_coefficients` を追加してステップごとの対数減衰 $\Delta_{t,c}\lambda$ と書き込み $\Delta_{t,c}u_{t,c}\mathbf{B}_t$（形状 `(batch_size, T, H, N)`）を計算し、さらに層の順伝播を追加して `adjoint_scan` を実行し状態を $\mathbf{C}_t$ と縮約する。

```
@d2l.add_to_class(SelectiveSSM)
def log_coefficients(self, U):
    """Log-decays and writes, each of shape (batch_size, T, H, N)."""
    if self.selective:  # Step sizes (batch_size, T, H), write vectors
        dt, B = jax.nn.softplus(self.W_dt(U)), self.W_B(U)[..., None, :]
    else:  # One step size and one write vector per channel
        dt, B = jax.nn.softplus(self.dt_bias[...]), self.B[...]
    b = (dt * U)[..., None] * B
    log_a = dt[..., None] * -jnp.exp(self.log_neg_rate[...])
    return jnp.broadcast_to(log_a, b.shape), b

@d2l.add_to_class(SelectiveSSM)
def __call__(self, U):  # U: (batch_size, num_steps, num_channels)
    log_a, b = self.log_coefficients(U)
    H = adjoint_scan(jnp.exp(log_a), b)  # States, (batch_size, T, H, N)
    C = self.W_C(U)[..., None, :] if self.selective else self.C[...]  # Read
    return jnp.einsum('...n,...n->...', H, C) + self.D[...] * U  # Sum over n
```

`Mamba` モジュールは `SelectiveSSM` を Figure 12.12 の 2 ブランチ構造で包み、幅 d を $2H = 4d$ へ

W_in によって拡張し、SSM ブランチに因果デプスワイズ畳み込み conv と SiLU を適用したうえで、silu(Z) でゲート制御し、W_out によって元の幅へ射影して戻す。

```
class Mamba(nnx.Module):
    """The Mamba layer: a gated selective SSM between two projections."""
    def __init__(self, num_hiddens, num_states=16, expand=2, conv_width=4,
                 *, rngs, **ssm_args):
        H = expand * num_hiddens
        self.W_in = nnx.Linear(num_hiddens, 2 * H, use_bias=False, rngs=rngs)
        self.conv = nnx.Conv(H, H, (conv_width,), feature_group_count=H,
                             padding=((conv_width - 1, 0),),  # Causal
                             rngs=rngs)
        self.ssm = SelectiveSSM(H, math.ceil(num_hiddens / 16), num_states,
                                rngs=rngs, **ssm_args)
        self.W_out = nnx.Linear(H, num_hiddens, use_bias=False, rngs=rngs)

    def ssm_input(self, X):
        """The input of the SSM and the gate branch."""
        U, Z = jnp.split(self.W_in(X), 2, -1)
        return jax.nn.silu(self.conv(U)), Z

    def __call__(self, X):  # X: (batch_size, num_steps, num_hiddens)
        U, Z = self.ssm_input(X)
        return self.W_out(self.ssm(U) * jax.nn.silu(Z))
```

12.5.4 訓練済み Mamba モデル

12.5.4.1 Mamba による選択的コピー

ここで、選択的コピーにおいて Mamba（チャネルあたり $N = 4$ 状態）と、その非選択的アブレーションである Mamba, not selective（selective=False とし、まったく同じ射影、幅 4 の畳み込み、SiLU ゲートを維持したもの）を並べて訓練する。

```
for name, selective in (('Mamba', True), ('Mamba, not selective', False)):
    models[name] = copy_runs(name, lambda: d2l.SeqLM(10, 48, [
        Mamba(48, num_states=4, selective=selective, rngs=rngs)
        for _ in range(2)], rngs=rngs))
```

```
Mamba                71,242 parameters, accuracy 1.00, 1.00
Mamba, not selective 70,090 parameters, accuracy 0.19, 0.49
```

Mamba はほぼすべての記号をコピーする。正解率は 95% 以上、通常は 100% 近くに達し、minGRU を上回る。一方、非選択的な対照モデルでは正解率が 30 パーセントポイント以上低下し、実行ごとのばらつきも大きくなる。これは Gu and Dao (2023) で報告されたアブレーション結果と一致する。射影、畳み込み、およびゲートは両モデルで同一であるため、この差は選択機構のみに起因する。演習問題 3 では、4,096 個のフィラーに対するゼロショットの長さ外挿（length extrapolation）を検証する。

12.5.4.2 状態への寄与

Mamba が**どのようにして**フィラーを無視しつつ 4 個のマーク付き記号を保持しているかを調べるため、最初のクエリトークンにおける最終状態を、それ以前の各位置 $s \leq T$ からの寄与へと分解

する。展開された漸化式 $\mathbf{h}_T = \sum_{s=1}^{T} (\prod_{r=s+1}^{T} \mathbf{a}_r) \odot \mathbf{b}_s$ において、位置 s の減衰後の項のユークリッドノルム $\| \exp(\sum_{r=s+1}^{T} \log \mathbf{a}_r) \odot \mathbf{b}_s \|_2$ は、位置 s での書き込みがステップ T の状態にどれだけ残存しているかを表す。

```
def state_contributions(log_a, b):
    """Size of each position's term in the final state of h = a h + b."""
    cum = log_a.cumsum(1)
    terms = jnp.exp(cum[:, -1:] - cum) * b  # Decayed to the last step
    return jnp.linalg.norm(terms.reshape(*terms.shape[:2], -1), axis=-1)
```

`log_coefficients` を `d2l.S4D` に追加し、ステップごとの対数減衰と書き込みを `(batch_size, T, d, N)` という `SelectiveSSM` と同じ形式で取り出せるようにする。

```
@d2l.add_to_class(d2l.S4D)
def log_coefficients(self, X):
    """Log-decays and writes of the recurrence, each (batch_size, T, d, N)."""
    rate = -jnp.exp(self.log_neg_rate[...])
    log_a = jnp.exp(self.log_dt[...]) * rate  # log a_bar = dt * lambda
    b_bar = jnp.expm1(log_a) / rate
    return (jnp.broadcast_to(log_a, (*X.shape, log_a.shape[-1])),
            b_bar * X[..., None])
```

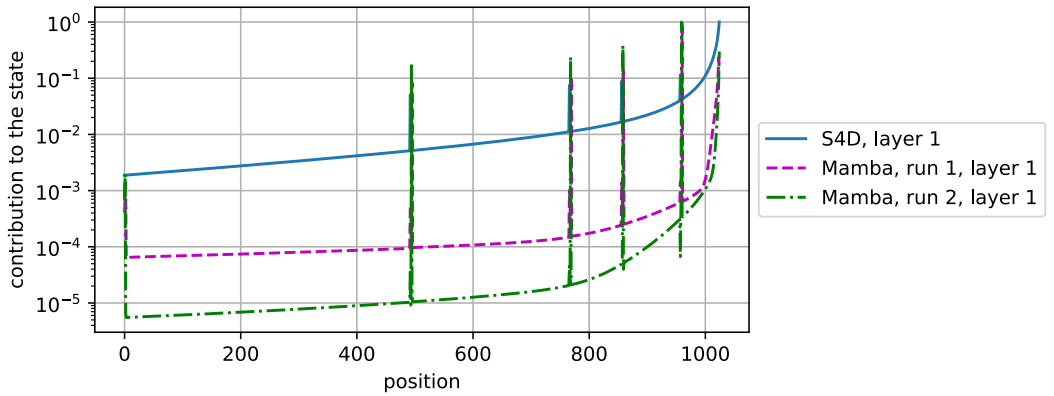
2 つの層のどちらを見るべきだろうか。訓練時にはどの層で選択を行うべきかは何も指定されないため、両方の層を調べる。`layer_contributions` は指定された層より下のブロックを実行し、最初のクエリトークンまでで打ち切ったテスト系列 `X`（`X[:, :-3]`）に対するその層の対数減衰と書き込みを取り出して、`state_contributions` を計算する。これを `S4D` および訓練済みの両方の `Mamba` 実行結果における両層に適用する。

```
def layer_contributions(model, X, layer):
    """Contributions to the state of a layer at the last step of X."""
    H = model.embedding(X)
    for i in range(layer):  # The blocks below it, as in the forward pass
        H = H + model.mixers[i](model.norms[i](H))
        H = H + model.mlps[i](H)
    mixer, X = model.mixers[layer], model.norms[layer](H)
    if isinstance(mixer, d2l.S4D):
        return state_contributions(*mixer.log_coefficients(X))
    U, _ = mixer.ssm_input(X)  # Mamba: what its selective SSM reads
    return state_contributions(*mixer.ssm.log_coefficients(U))

X, _ = copy_batch(d2l.get_key(), 1)
X = X[:, :-3]  # Up to the first query token
probed = {'S4D': models['S4D'][0], 'Mamba, run 1': models['Mamba'][0],
          'Mamba, run 2': models['Mamba'][1]}  # Both trained Mamba runs
contributions = {(name, layer): layer_contributions(model, X, layer)[0]
                 for name, model in probed.items() for layer in (0, 1)}
```

幅 4 の因果畳み込みは各マーク付き記号を最大 4 個の連続する位置に広げるため、`symbol_windows` で各記号から始まる 4 位置分のウィンドウを集める（系列全体の $\leq 1.6\%$）。各層について、最終状態のうちそれらのウィンドウに由来する割合を表示し、続いて各モデルについてその割合が大きい方の層における全 $1{,}025$ 位置の寄与をプロットする。

```
def symbol_windows(X, width=4):
    """Positions of each symbol and the width - 1 after it, without repeats."""
    starts = jnp.nonzero(X[0] > 1)[0]  # The four symbols
    windows = (starts[:, None] + jnp.arange(width)).flatten()
    windows = jnp.unique(windows)  # Windows of nearby symbols may overlap
    return windows[windows < X.shape[1]]  # A window may reach past the end

windows = symbol_windows(X)
print(f'symbol windows: {len(windows) / X.shape[1]:.1%} of the positions')
share = {key: float(c[windows].sum() / c.sum())
         for key, c in contributions.items()}
print('share of the state from them  layer 1  layer 2')
for name in probed:
    print(f'{name:28s}  {share[name, 0]:7.2f}  {share[name, 1]:7.2f}')
best = {name: max((0, 1), key=lambda layer: share[name, layer])
        for name in probed}  # The layer that selects more
curves = {f'{name}, layer {layer + 1}': contributions[name, layer]
          for name, layer in best.items()}
d2l.plot(jnp.arange(X.shape[1]), [c / c.max() for c in curves.values()],
         'position', 'contribution to the state',
         legend=list(curves), yscale='log', figsize=(6, 3),
         legend_loc='outside')  # The curves fill the axes
```

```
symbol windows: 1.6% of the positions
share of the state from them  layer 1  layer 2
S4D                             0.03     0.02
Mamba, run 1                    0.57     0.36
Mamba, run 2                    0.76     0.21
```



この割合は、1 本の系列内でたまたま記号がどの位置に落ちたかに依存する。局所的な指標となるのが**スパイク比**（spike ratio）、すなわち記号の 4 ステップウィンドウ内における最大寄与を、その直前のフィラートークンの寄与で割った値である。新たに生成した 64 本のテスト系列の記号全体にわたる中央値を、同様に両方の層で計算する。

```
X_test = copy_batch(d2l.get_key(), 64)[0][:, :-3]  # 64 new sequences
rows, starts = jnp.nonzero(X_test[:, 1:] > 1)  # The symbols,
starts = starts + 1                            # not at position 0
```

```
    window = jnp.minimum(starts[:, None] + jnp.arange(4),
                         X_test.shape[1] - 1)  # A window may reach past the end
    def spike_ratio(c):  # Median over the symbols
        return float(jnp.median(c[rows[:, None], window].max(1)
                                / c[rows, starts - 1]))

    print(f'spike ratio, median over {len(rows)} symbols   layer 1   layer 2')
    for name, model in probed.items():
        spikes = [spike_ratio(layer_contributions(model, X_test, layer))
                  for layer in (0, 1)]
        print(f'{name:36s} {spikes[0]:9,.1f} {spikes[1]:9,.1f}')
```

```
spike ratio, median over 256 symbols   layer 1   layer 2
S4D                                       10.5       5.3
Mamba, run 1                           1,418.1   1,291.5
Mamba, run 2                          31,145.3       2.2
```

S4D のプロファイルは直近性に従う。系列の末尾に向かって滑らかに上昇し、記号ウィンドウが状態に占める割合はわずか数パーセントと、位置数の割合を少し上回る程度にすぎない。固定フィルタであってもトークンによって強く応答することはできるため、記号はその直前のフィラーより目立ちはするものの、高々数十倍にとどまり、そのトークンをより長く保持することはできない。一方 Mamba では、選択を担う層が記号を隣接するフィラーの数百倍から数万倍にまで引き上げ、4 つのウィンドウが状態の大部分（多くの場合は過半）を占める。どちらの層がこの役割を担うかは、訓練前にあらかじめ決まっているわけではない。両方の層が選択を行う実行もあれば、片方の層だけが行う実行もあり（どちらの層になる場合もある）、そのときもう一方の層のスパイク比は S4D と同程度にとどまる。表に両方の層を載せ、プロットにはより選択性の高い層を表示しているのはそのためである。

層はどのようにしてこのようなスパイクを生み出すのだろうか。自然な推測は、記号のたびに大きなステップをとることで、その記号を書き込むと同時にそれ以前のフィラーを一掃している、というものであろう。しかし、訓練済みモデルのステップサイズはそれとは異なる事実を示している。$H = 96$ 個のチャネル全体で平均すると、記号でのステップは通常フィラーでのステップと変わらない。選択はごく少数のチャネルの仕事であり、各記号は 2〜3 個のチャネルによって運ばれる。フィラー上では、これらのチャネルはステップ $\Delta_{t,c}$ を約 10^{-5} にとるため、減衰も書き込みも起こらず、状態は凍結される。一方、記号上では同じチャネルが約 0.1 から 1.5 の間のステップをとり、書き込み $\Delta_{t,c}\, x_{t,c}\, \mathbf{B}_t$ はステップに比例するため、記号は十分な強さで状態に入る。このようなステップはチャネルがそれまで保持していた内容も減衰させるが、その減衰はわずかな倍率にすぎず、直前のフィラーを消去することがスパイクに寄与する度合いは小さい。ここでの選択とは、状態のフラッシュ（一掃）ではなく、凍結された少数のチャネルへの強い書き込みなのである。

12.5.4.3 言語モデリング

最後に、Mamba、minGRU、attention をバイトレベルの WikiText-103 ベンチマーク（d2l.SeqLM、幅 $d = 256$ の 4 ブロック、128 バイトのウィンドウ、1,000 ステップ）で比較する。まず 128 バイトの WikiText103 ストリームを読み込む。

```
    data = d2l.WikiText103(batch_size=64, num_steps=128)
```

次に text_model(mixer, d) を定義し、4 つの実験条件 arms を設定する。これには、パラメータ数を合わせた minGRU, d=309（パラメータ数 3.99M、Mamba のチャネル拡張後のパラメータ数と一

致）も含める。

```
def text_model(mixer, d):  # 256 byte values; four blocks of width d
    rngs = nnx.Rngs(d2l.get_key())
    return d2l.SeqLM(256, d, [mixer(d, rngs) for _ in range(4)], rngs=rngs)

mamba = lambda d, rngs: Mamba(d, rngs=rngs)
min_gru = lambda d, rngs: d2l.MinGRU(d, d, rngs=rngs)
attention = lambda d, rngs: d2l.AttentionMixer(d, num_heads=8, rngs=rngs)
arms = {'Mamba': (mamba, 256), 'minGRU': (min_gru, 256),
        'attention': (attention, 256),
        'minGRU, d=309': (min_gru, 309)}  # As many parameters as Mamba
```

`d2l.text_table` を呼び出すと、各アーキテクチャを 2 つのシードで $1{,}000$ ステップ訓練し、ウィンドウ内の位置帯ごとの検証損失とステップあたりのレイテンシを表にまとめる。

```
trained = d2l.text_table(text_model, arms, data)
```

```
                          validation        0-15       16-63      64-127   ms/step
Mamba           3,989,248 1.297/1.296 1.488/1.488 1.272/1.271 1.268/1.267 35/35
minGRU          2,764,544 1.344/1.343 1.521/1.522 1.319/1.319 1.318/1.316 10/10
attention       3,286,784 1.386/1.384 1.561/1.562 1.362/1.359 1.360/1.357 9/9
minGRU, d=309   3,991,918 1.328/1.328 1.508/1.507 1.302/1.303 1.302/1.301 10/10
```

Mamba は 1.30 nat/byte に達する。並列スキャンで訓練しながらも 12.1.4.2 節の完全な LSTM および GRU に匹敵し、同一幅の minGRU を 0.05 nat/byte、パラメータ数を合わせた minGRU, d=309 を 0.03 nat/byte 上回る。Mamba はトークンあたり $H \times N = 512 \times 16 = 8{,}192$ 個の数値を状態としてスキャンするため（サイズは $32\times$ であり、minGRU の状態は 256 個）、非融合カーネルでの訓練ステップは $2\times$〜$4\times$ ほど minGRU より時間がかかる。ただし推論時には、その $32d$ 個の状態（幅 4 の畳み込みバッファの $6d$ 個を加えても）は、マルチヘッドアテンションのわずか 19 トークン分の KV キャッシュに等しい。さらに、Mamba の $H \times N$ 状態を外積による行列メモリ $\mathbf{S}_t$（$(\Delta_t \odot \mathbf{x}_t)\mathbf{B}_t^\top$ で更新し $\mathbf{C}_t$ でクエリする）とみなす見方は、12.6 節の行列状態漸化式およびチャンク単位の行列積による訓練へと直結する。

12.5.5 まとめ

時不変の状態空間モデルは過去の入力をラグのみによって重み付けするため、選択的コピーのような内容選択的なタスクでは失敗する。選択的状態空間モデルは、各入力トークンからチャネルごとのステップサイズ $\Delta_t = \mathrm{softplus}(\mathbf{W}_\Delta \mathbf{x}_t + \mathbf{b}_\Delta)$ と共有の書き込み・読み出しベクトル $\mathbf{B}_t, \mathbf{C}_t$ を計算する。単一の状態（$N = 1$）では、$e^{-\mathrm{softplus}(s_t)} = \sigma(-s_t)$ によって MinGRU の更新ゲートが復元される。また、N 個の状態をレート $-1, \dots, -N$ に初期化し、対数一様な Δ バイアスと組み合わせることで、Mamba は内容選択と多重時間スケールの記憶を両立させる。並列スキャンを逆時間スキャン（adjoint_scan）によって微分すれば、$O(\log_2 T)$ 個の倍増ラウンド中間テンソルの保存を回避できる。選択的コピーにおいて Mamba は正解率 95% 以上に達し（選択性を無効にすると > 30 ポイント低下する）、これはフィラー上で凍結されたまま各記号を大きなステップで書き込む少数のチャネルによるものである。さらにバイトレベルの WikiText-103 では、完全な LSTM および GRU に匹敵しつつ、パラメータ数を合わせた MinGRU を 0.03 nat/byte 上回る。

12.5.6 演習問題

1. **書き込み重みの近似**。Mamba はゼロ次ホールドの減衰 $e^{\Delta\lambda}$ を維持する一方、書き込みには重み Δ を用い、$(e^{\Delta\lambda}-1)/\lambda$ は用いない。
   1. $\lambda = -1$ のとき、これら 2 つの重みの差が 10% 未満となる最大の Δ を求めよ。
   2. $\lambda = -1$、書き込みベクトル 1、および厳密な重みを持つ 1 状態のモデルにおいて、0 から出発した状態が、0 とそれまでに見た入力値によって張られる区間内にとどまることを示せ。一方、入力が $[-1, 1]$ の値をとる系列で、重み Δ を用いた状態がその区間からはみ出すものを 1 つ見つけよ。訓練済み Mamba モデルの値を適切な範囲内に保っているものは何か。

2. **スキャンのメモリ量**。H チャネルと N 個の状態を持つ Mamba 層において、B 本の系列（長さ T）からなるミニバッチと 32 ビット数値を考える。誤差逆伝播法がスキャンのために保存するバイト数を、(a) 倍増スキャンの各ラウンドを通す場合、(b) 随伴スキャンを用いる場合、(c) 層の入力のみを保存して状態を再計算するカーネルを用いる場合のそれぞれについて数えよ。$H = 4096$、$N = 16$、$B = 8$、$T = 8192$ に対してこれら 3 つを計算し、GPU のメモリ容量と比較せよ。

3. [code] **より長い系列**。クエリの前に 1,024 位置がある設定で訓練した Mamba モデルを、再訓練せずに 2,048 および 4,096（`num_fillers`）の系列で評価せよ。
   1. モデルは依然として記号をコピーできるか。minGRU と比較せよ。
   2. より選択性の高い層の各チャネルについて、最も遅い状態の 1,000 フィラートークンにわたる減衰を計算せよ。4,096 個のフィラーを越えて記号を運べるチャネルは何個あるか。

4. [code] **どの部分が選択的であるべきか**。●
   1. 本節で扱った 2 つの Mamba モデルの中間には、さらに 2 つの変種が存在する。入力依存のステップサイズと固定された書き込み・読み出しベクトル $\mathbf{B}$ および $\mathbf{C}$ を持つモデルと、固定されたステップサイズと入力依存の $\mathbf{B}_t$ および $\mathbf{C}_t$ を持つモデルである。両方をそれぞれ 2 回ずつ訓練せよ。どちらがタスクを解けるか。その結果を上で測定したスパイク比と関連付けて説明せよ。
   2. 訓練済み Mamba モデルにおいて、各記号から始まる 4 位置（上で測定したウィンドウ）における両層のステップサイズを、フィラー位置全体にわたる中央値に置き換え、どちらの層も大きなステップで記号を書き込めないようにせよ。正解率はどれだけ低下するか。次に、それ以外のすべての位置におけるステップサイズをウィンドウ全体にわたる中央値に置き換え、どちらの層もフィラーをスキップできないようにして同じ実験を繰り返せ。さらに、1 層ずつ個別にこの介入を行え。結果は上で測定した寄与割合と整合するか。

5. [code] **生成**。メソッド `step` を `SelectiveSSM` と `Mamba` に追加し、SSM の状態と畳み込みの直近 3 入力を受け取って 1 トークンを処理し、出力と新しい状態を返すようにせよ。系列を 1 ステップずつ進めることで順伝播の出力が再現されることを確認せよ。層はトークンからトークンへ何個の数値を引き継ぐか。

6. [code] **短い初期記憶**。minGRU は、ゲートの初期記憶が約 1 ステップしかないにもかかわらずコピーを学習した。クエリに最も近い記号がゲートを訓練するという説明を検証せよ。
   1. 訓練前後のフィラートークンにおける minGRU のゲート値を記録し、さらに訓練済みモデルについて、記号とそのクエリの間の距離でソートしたうえで各クエリごとの正解率を個別に記録せよ。どの記号を最もよくコピーできるか。
   2. Mamba モデルのステップサイズを 0.5 から 1 の間（`dt_min=0.5, dt_max=1.0`）で初期化し、すべての状態が高々 2 ステップの初期記憶しか持たないようにして訓練せよ。

訓練予算内で記号のコピーを学習できるか。もし学習できるなら、第 1 層のフィラートークンにおけるステップサイズを訓練前後でプロットせよ。

3. クエリ前の位置数を 4,096（クエリ付近に記号が現れる頻度がより低くなる設定）に変え、さらに 12.4 節の長距離想起実験と同様に初期化した minGRU も含めて、両方の実験を繰り返せ。

12.6 行列状態漸化式

12.5 節の選択的状態空間モデル（selective state space model）は、何をどれだけ長く保持するかをトークンごとに決定するが、固定サイズの状態がいくつの異なる対応関係を保存して検索できるかはまだ問うていない。言語モデリングの多くは**連想想起**（associative recall）、すなわちキーに値を束縛し（`rate = 0.05` や「**Mira は船の機関士である**」など）、数百トークン後にそのキーが再出現した際に値を検索する能力を必要とする。Transformer は、過去のすべてのキーと値を増大し続ける KV キャッシュに保持し、クエリをそれらすべてと比較することで連想想起を解いている。固定サイズの再帰状態（recurrent state）でも同様にキー・値ペアを保存して検索できるのだろうか。また、その容量は状態サイズに対してどのようにスケールするのだろうか。

答えはイエスである。ただしそれは、状態が**行列** $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$ であり、外積（outer product）$\mathbf{v}_t \mathbf{k}_t^\top$ の加算によって更新され、行列・ベクトル積 $\mathbf{S}_t \mathbf{q}_t$ によってクエリされる場合である。この構造の具体例にはすでに 2 つ出会っている。非正規化線形アテンション（10.6 節）と、チャネル × 状態の $H \times N$ 行列を持つ Mamba（12.5 節）である。以下では、外積行列メモリの厳密な検索誤差と容量のスケーリング則（scaling law）を導出し、古くなったペアを忘却するための入力依存（input-dependent）のスカラー減衰を導入し、減衰付き行列漸化式と因果マスク付き線形アテンションの間の**状態空間双対性**（state space duality）を確立する。さらに、ハードウェア効率の高いチャンク単位の訓練アルゴリズム（`chunked_linear_attention`）を実装し、訓練された `GatedLinearAttention` 層がマルチクエリ連想想起において理論的な容量則に従うかどうかを検証する。

12.6.1 連想メモリ

12.6.1.1 ペアの書き込みと読み出し

最も単純な連想メモリ（associative memory）は、外積の和をとることでペアを保存する。$\mathbb{R}^2$ における簡単な例として、直交する単位キー $\mathbf{k}_1 = (1, 0)^\top$ と $\mathbf{k}_2 = (0, 1)^\top$ をスカラー値 $v_1 = 3$ と $v_2 = 5$ に束縛し、行ベクトル $\mathbf{S} = v_1 \mathbf{k}_1^\top + v_2 \mathbf{k}_2^\top = (3, 5)$ を作ってみよう。どちらのキーを掛けても、$\mathbf{S}\mathbf{k}_1 = 3$、$\mathbf{S}\mathbf{k}_2 = 5$ のように誤差なくその値を検索できる。しかし、3 つ目のペアとして $\mathbf{k}_3 = (1, 1)^\top / \sqrt{2}$ と $v_3 = 7$ を追加すると、$\mathbf{k}_3$ は $\mathbf{k}_1$ および $\mathbf{k}_2$ と重なりを持つ。その結果、$\mathbf{S}$ は $(3 + 7/\sqrt{2}, 5 + 7/\sqrt{2}) \approx (7.95, 9.95)$ となり、$\mathbf{k}_1$ でクエリすると 7.95 が返り、3 ではなくなる。キーが 2 次元のとき、クロストークなしで保持できる直交キーは高々 2 個である。

一般に、キーベクトル $\mathbf{k}_t \in \mathbb{R}^{d_k}$、値ベクトル $\mathbf{v}_t \in \mathbb{R}^{d_v}$、クエリベクトル $\mathbf{q}_t \in \mathbb{R}^{d_k}$ に対し、状態は行列 $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$ であり、各ステップで次の式により更新および読み出しされる。

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top, \qquad \mathbf{y}_t = \mathbf{S}_t \mathbf{q}_t. \tag{12.26}$$

各書き込みでは、ランク 1 行列 $\mathbf{v}_t \mathbf{k}_t^\top$（方向 $\mathbf{k}_t$ を $\mathbf{v}_t$ に写す行列）を加算する。ここで、$(\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t)$ はトークン表現に対する学習された線形射影である。

12.6.1.2 線形アテンション

Equation 12.26 を $\mathbf{S}_0 = \mathbf{0}$ から展開すると、加法的な行列漸化式が非正規化の因果線形アテンション（linear attention）を計算していることがわかる（10.6 節、Katharopoulos et al. (2020)）。

$$\mathbf{y}_t = \Big(\sum_{s \le t} \mathbf{v}_s \mathbf{k}_s^\top\Big)\mathbf{q}_t = \sum_{s \le t} (\mathbf{k}_s^\top \mathbf{q}_t)\, \mathbf{v}_s .$$

右から結合して評価すると、各出力 $\mathbf{y}_t$ は過去のすべての値 $\mathbf{v}_s$ をドット積類似度 $\mathbf{k}_s^\top \mathbf{q}_t$ で重み付けする。一方、左から結合して評価すると、外積の和が固定サイズの再帰状態 $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$ として順方向に引き継がれていく。

12.6.1.3 干渉

クロストークが信号を圧倒するまでに、$\mathbf{S} = \sum_{i=1}^{n} \mathbf{v}_i \mathbf{k}_i^\top$ はいくつのペアを保存できるだろうか。保存済みの単位キー $\mathbf{k}_j$（$\|\mathbf{k}_j\| = 1$）でクエリすると、目的の値と他の $n-1$ 個のペアによる干渉とが分離される。

$$\mathbf{S}\mathbf{k}_j = \mathbf{v}_j + \sum_{i \ne j} (\mathbf{k}_i^\top \mathbf{k}_j)\, \mathbf{v}_i . \tag{12.27}$$

誤差ゼロの厳密な読み出しには互いに直交するキーが必要だが、$\mathbb{R}^{d_k}$ にはそのようなキーは高々 d_k 個しか存在しない。ランダムな単位キーについては、期待される干渉の大きさを厳密に計算できる。

命題： $\mathbf{k}_1, \dots, \mathbf{k}_n$ を $\mathbb{R}^{d_k}$ の単位球面上の一様分布に独立に従うベクトルとし、$\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^{d_v}$ をキーとは独立に選ばれた単位ベクトル（unit vector）とする。このとき、保存された任意のキーにおける期待二乗検索誤差は次の式で与えられる。

$$\mathbb{E}\,\big\|\mathbf{S}\mathbf{k}_j - \mathbf{v}_j\big\|^2 = \frac{n-1}{d_k}. \tag{12.28}$$

証明： $\sum_{i \ne j} (\mathbf{k}_i^\top \mathbf{k}_j)\mathbf{v}_i$ の二乗ノルムを展開すると、$i \ne l$ の交差項は期待値をとると消える。なぜなら、$\mathbf{k}_j$ を条件付けたとき、$\mathbf{k}_i^\top \mathbf{k}_j$ と $\mathbf{k}_l^\top \mathbf{k}_j$ は独立で平均がゼロだからである。$\|\mathbf{v}_i\|^2 = 1$ より、$n-1$ 個の対角項 $\mathbb{E}[(\mathbf{k}_i^\top \mathbf{k}_j)^2]$ が残る。$\mathbb{R}^{d_k}$ の単位球面の回転対称性より、$\mathbf{k}_i$ の各座標の二乗期待値は $1/d_k$ であるため、$\mathbb{E}[(\mathbf{k}_i^\top \mathbf{k}_j)^2] = 1/d_k$ となる。干渉する $n-1$ 個のペアについて和をとれば $(n-1)/d_k$ が得られる。■

なお、検索誤差は負荷率 $(n-1)/d_k$ にのみ依存し、ペアが**いつ**書き込まれたかには依存しない。つまり、加法的なメモリがペアを失う原因は、時間経過ではなく過密である。

Equation 12.28 を NumPy で実験的に検証するため、`matrix_memory` は単位キーとコードブック値のペア n 個を $\mathbf{S} = \sum_{i=1}^{n} \mathbf{v}_i \mathbf{k}_i^\top$ に書き込み（オプションで共通方向の相関 ρ を付与できる）、256 個の単位ベクトルからなるコードブック上での平均二乗検索誤差と最近傍想起率の両方を返す。

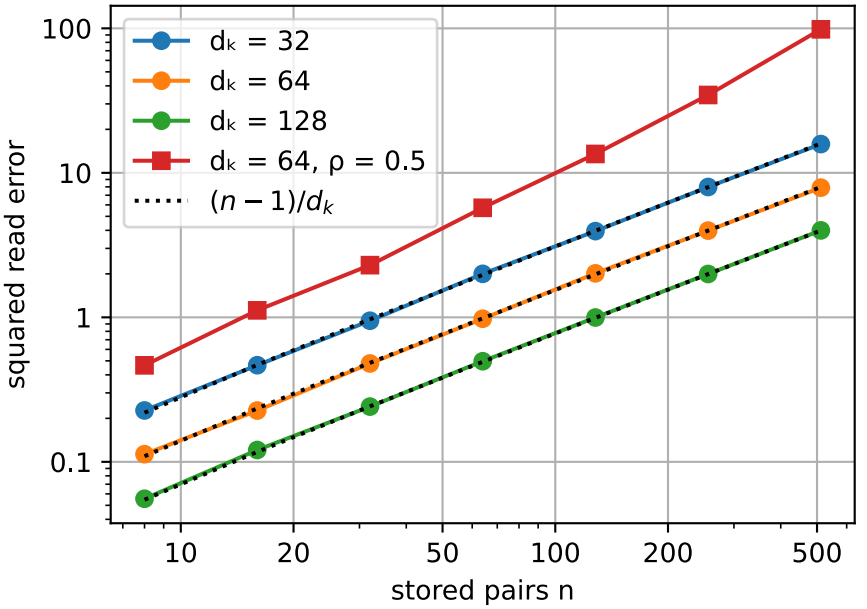
```
def matrix_memory(d_k, nums, rho=0, num_values=256, trials=20, seed=0):
    """Read error and recall of S = sum_i v_i k_i^T over random unit keys."""
    rng = np.random.default_rng(seed)
```

```
    unit = lambda X: X / np.linalg.norm(X, axis=-1, keepdims=True)
    codebook = unit(rng.standard_normal((num_values, d_k)))
    errors, recalls = [], []
    for n in nums:
        error = recall = 0
        for _ in range(trials):
            K = unit(rng.standard_normal((n, d_k)))  # Keys as rows
            shared = unit(rng.standard_normal(d_k))  # A common direction
            K = unit(np.sqrt(1 - rho**2) * K + rho * shared)
            ids = rng.integers(0, num_values, n)
            S = codebook[ids].T @ K  # Sum of v_i k_i^T
            reads = K @ S.T  # Row j is S k_j
            error += ((reads - codebook[ids]) ** 2).sum(1).mean() / trials
            recall += ((reads @ codebook.T).argmax(1) == ids).mean() / trials
        errors.append(error)
        recalls.append(recall)
    return errors, recalls
```

matrix_memory をペア数 $n \in \{8, 16, \ldots, 512\}$ にわたって、独立なランダムキー（$d_k \in \{32, 64, 128\}, \rho = 0$）および相関のあるキー（$d_k = 64, \rho = 0.5$）について評価し、二乗読み出し誤差を理論直線 $(n-1)/d_k$ と比較してプロットする。

```
nums = [8, 16, 32, 64, 128, 256, 512]
results = {f'd☒ = {d_k}': matrix_memory(d_k, nums) for d_k in (32, 64, 128)}
results['d☒ = 64, ρ = 0.5'] = matrix_memory(64, nums, rho=0.5)
predictions = [[(n - 1) / d_k for n in nums] for d_k in (32, 64, 128)]
d2l.plot(nums, [err for err, _ in results.values()] + predictions,
         'stored pairs n', 'squared read error',
         legend=list(results) + ['$(n-1)/d_k$'], xscale='log', yscale='log',
         fmts=('-o', '-o', '-o', '-s', 'k:', 'k:', 'k:'), figsize=(5, 3.5))
print('recall at n =    ', ' '.join(f'{n:4d}' for n in nums))
for name, (_, recalls) in results.items():
    print(f'{name:17s}', ' '.join(f'{r:4.2f}' for r in recalls))
```

```
recall at n =        8   16   32   64  128  256  512
d☒ = 32           1.00 0.99 0.96 0.74 0.39 0.16 0.07
d☒ = 64           1.00 1.00 1.00 1.00 0.96 0.72 0.37
d☒ = 128          1.00 1.00 1.00 1.00 1.00 1.00 0.94
d☒ = 64, ρ = 0.5  1.00 0.99 0.83 0.49 0.21 0.07 0.03
```



プロットと表から、2 つの性質が明確に読み取れる。

1. **想起率は状態の総要素数 $d_k d_v$ に応じてスケールする：**二乗誤差の曲線は $(n-1)/d_k$ にぴったり重なる。ところが $n = d_k = d_v = 64$ では、総二乗誤差が ≈ 1（単位信号 $\|\mathbf{v}_j\|^2 = 1$ と同じ大きさ）に達しているにもかかわらず、最近傍想起率は依然として 1.00 である（96% を維持するのは $n = 128$ のとき）。なぜか。干渉ベクトルは値の全 d_v 次元にわたって等方的に広がるため、任意の 1 つのコードブック候補ベクトルへの射影の分散が $\approx n/(d_k d_v)$ に抑えられるからである。$d_v = d_k$ のとき、d_k を 2 倍にすると状態の要素数 $d_k d_v$ は 4 倍になり、一定の正解率で想起できるペア数も 4 倍になる。実際、正解率 96% を達成するのは $n = 32$（$d_k = 32$）および $n = 128$（$d_k = 64$）であり、94% を達成するのは $n = 512$（$d_k = 128$）である（演習問題 5 では d_k と d_v を独立に変化させる）。

2. **キーの相関は削減不能な誤差の下限を生む：**キーが重み ρ で共通の方向を持つとき、ペアごとのキーの重なりは平均して $\approx \rho^2$ となり、期待二乗誤差は次のようになる。

$$\mathbb{E}\,\|\mathbf{S}\mathbf{k}_j - \mathbf{v}_j\|^2 \approx (n-1)\Big(\rho^4 + \frac{(1-\rho^2)^2}{d_k}\Big)$$

これは値の平均がゼロの場合の式である（有限のコードブックの標本平均に由来する n の二次項がさらに加わる。演習問題 4 を参照）。ρ^4 は d_k を大きくしても減少しないため、相関のあるキー（$\rho = 0.5$）では $n = d_k = 64$ における想起率が 1.00 から 0.49 へと低下する。

12.6.2 減衰

12.6.2.1 古いペアの忘却

減衰がなければ、これまでに書き込まれたすべてのペアが最大の重みのまま $\mathbf{S}_t$ に残り続けるため、干渉は系列長（sequence length）とともに際限なく増大する。各書き込みの前に状態へスカラー減衰 $a_t \in (0,1)$ を掛けることで、古いペアを徐々に薄れさせる。

$$\mathbf{S}_t = a_t\,\mathbf{S}_{t-1} + \mathbf{v}_t\mathbf{k}_t^\top, \qquad a_t \in (0,1). \tag{12.29}$$

定数減衰 a のもとでは、ℓ ステップ前に書き込まれたペアは重み a^ℓ を持つ。重みの総和は $1/(1-a)$（実効ウィンドウとして $1/(1-a)$ ペア分）となる一方、干渉の分散に寄与する二乗重みの総和は $1/(1-a^2)$ となり、全体の系列長にかかわらず定常状態の期待干渉は $1/((1-a^2)d_k)$ で上から抑えられる（演習問題 6）。入力トークンから a_t を計算するようにすれば、各ヘッドはある区間にわたって状態を保持し、話題や節の境界でそれを消去できる。（関連する変種として、ヘッドごとに固定減衰を用いる RetNet (Sun et al. 2023)、ヘッドごとに入力依存のスカラー減衰を用いる Mamba-2 (Dao and Gu 2024)、キーの座標ごとにベクトル減衰 $\mathbf{S}_{t-1}\mathrm{diag}(\alpha_t)$ を用いるゲート付き線形アテンション（Gated Linear Attention、GLA）(S. Yang, Wang, Shen, et al. 2024) がある。本節ではヘッドごとに 1 つの入力依存スカラー減衰を用いる。）

12.6.2.2 行列メモリとしての選択的状態空間モデル

Mamba の選択的 SSM における H 個のチャネル状態 $\mathbf{h}_t^{(c)} \in \mathbb{R}^N$（12.5 節、スキップ接続 $D_c x_{t,c}$ は省略）を行列 $\mathbf{S}_t \in \mathbb{R}^{H\times N}$ の行として積み重ねると、Mamba がすでに減衰付きの行列状態漸化式になっていることがわかる。

$$\begin{aligned}\mathbf{S}_t &= \mathbf{E}_t \odot \mathbf{S}_{t-1} + (\Delta_t \odot \mathbf{x}_t)\,\mathbf{B}_t^\top,\\ \mathbf{y}_t &= \mathbf{S}_t\mathbf{C}_t, \qquad \mathbf{E}_t = \exp(\mathrm{diag}(\Delta_t)\,\Lambda).\end{aligned}$$

Equation 12.29 と各項を対応させると、値ベクトルは $\mathbf{v}_t = \Delta_t \odot \mathbf{x}_t \in \mathbb{R}^H$、キーベクトルは $\mathbf{k}_t = \mathbf{B}_t \in \mathbb{R}^N$、クエリベクトルは $\mathbf{q}_t = \mathbf{C}_t \in \mathbb{R}^N$ である。唯一の違いは、Mamba の減衰行列 $\mathbf{E}_t \in \mathbb{R}^{H\times N}$ がチャネルと状態の両方にわたって変化する点である。$\mathbf{E}_t$ をヘッドあたり単一のスカラー a_t に制限すると Mamba-2 (Dao and Gu 2024) が得られ（演習問題 1）、以下に示す行列積による双対形式が導かれる。

12.6.3 双対形式

12.6.3.1 減衰を伴う展開

Equation 12.29 を $\mathbf{S}_0 = \mathbf{0}$ から展開すると、ステップ s で書き込まれたペアはステップ $t \ge s$ に到達するまでに累積減衰積 $L_{ts} = a_{s+1}a_{s+2}\cdots a_t$ によって減衰することがわかる。

$$\mathbf{y}_t = \sum_{s\le t} L_{ts}\,(\mathbf{k}_s^\top\mathbf{q}_t)\,\mathbf{v}_s, \qquad L_{ts} = a_{s+1}a_{s+2}\cdots a_t. \tag{12.30}$$

$\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t, \mathbf{y}_t$ をそれぞれ $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{Y}$ の行として積み重ね、$L_{ts} = 0$（$s > t$ のとき）とおくと、次の**双対形式**（dual form）が得られる。

$$\mathbf{Y} = (\mathbf{L} \odot \mathbf{Q}\mathbf{K}^\top)\,\mathbf{V}. \tag{12.31}$$

Equation 12.31 は**学習された入力依存の減衰マスク $\mathbf{L}$** を伴う因果線形アテンションであり、この対応関係を Dao and Gu (2024) は**状態空間双対性**（SSD）と呼んでいる。Equation 12.31 をステップごとの漸化式として評価すると計算量は $O(Td_vd_k)$、メモリは $O(d_vd_k)$ で済む。一方、完全な $T\times T$ 行列 $\mathbf{L}\odot\mathbf{Q}\mathbf{K}^\top$ を介して評価すると、並列な行列積として実行できる代わりに計算量は $O(T^2(d_k+d_v))$、メモリは $O(T^2)$ を要する。両方の形式を明示的に書いてみよう。まずは逐次的なステップごとの漸化式 `linear_attention_recurrent` から始める。

```
def linear_attention_recurrent(Q, K, V, a):
    """y_t = S_t q_t with S_t = a_t S_{t-1} + v_t k_t^T, step by step."""
    def step(S, inputs):
        q, k, v, a_t = inputs
        S = a_t * S + jnp.outer(v, k)
        return S, S @ q
    S0 = jnp.zeros((V.shape[-1], K.shape[-1]))
    return jax.lax.scan(step, S0, (Q, K, V, a))[1]
```

次に、`decay_mask(log_a)` は累積和を用いて下三角減衰行列 $L_{ts} = \exp(\sum_{r=s+1}^{t} \log a_r)$ を構築し、これによって `linear_attention_dual` がすべての位置にわたって一度に $\mathbf{Y} = (\mathbf{L} \odot \mathbf{Q}\mathbf{K}^\top)\mathbf{V}$ を評価できるようにする。

```
def decay_mask(log_a):
    """L[t, s] = a_{s+1} * ... * a_t for s <= t, and 0 for s > t."""
    cum = log_a.cumsum(-1)
    log_L = cum[..., :, None] - cum[..., None, :]  # Sum of log a over (s, t]
    causal = jnp.tril(jnp.ones(log_L.shape[-2:], dtype=bool))
    return jnp.exp(jnp.where(causal, log_L, -jnp.inf))

def linear_attention_dual(Q, K, V, a):
    """The same outputs as Y = (L * Q K^T) V, all positions at once."""
    return (decay_mask(jnp.log(a)) * (Q @ K.T)) @ V
```

12.6.3.2 チャンク単位の計算

どちらの純粋な形式も GPU での訓練には最適ではない。トークン単位の漸化式は小さな逐次ステップを T 回実行する必要があり、完全な双対形式はヘッドごとに $T \times T$ 行列を実体化してしまうからである。**チャンク化**（chunking）は、系列を T/m 個のチャンク（各長さ m）に分割することで両者の長所を兼ね備える（Figure 12.13）。チャンクをまたぐ任意の減衰 L_{ts} は、（キーが属するチャンクの終端までの減衰）×（中間の各チャンクの減衰）×（クエリが属するチャンクの始端からの減衰）へと因数分解されるため、$\mathbf{L}$ のすべての非対角ブロックのランクは 1 になる。また、$\mathbf{L} \odot \mathbf{Q}\mathbf{K}^\top$ のすべての非対角ブロックのランクは高々 d_k となり、チャンク境界を越えて引き継がれる $d_v \times d_k$ の状態行列によって完全に要約される（演習問題 2）。

各チャンク内の位置に 1 から m までのインデックスを振ると、`chunked_linear_attention` はバッチ化された次の 3 つのステップで進む。

1. **チャンク内（対角ブロック）：** 双対形式 Equation 12.31 を、長さ m の各チャンクの内部でそれぞれの $m \times m$ 減衰マスクを用いて並列に評価する。
2. **チャンク間（チャンク境界の状態）：** $\gamma^{(c)} = a_1 \cdots a_m$ をチャンク c 全体の減衰とすると、チャンク c の終端における状態は次のチャンク単位の線形漸化式（linear recurrence）を満たす。

$$\mathbf{S}^{(c)} = \gamma^{(c)}\, \mathbf{S}^{(c-1)} + \sum_{t=1}^{m} L_{mt}\, \mathbf{v}_t \mathbf{k}_t^\top, \tag{12.32}$$

この漸化式は、`d2l.linear_scan` によって全 T/m チャンクにわたり $\lceil \log_2(T/m) \rceil$ ラウンドで評価される。

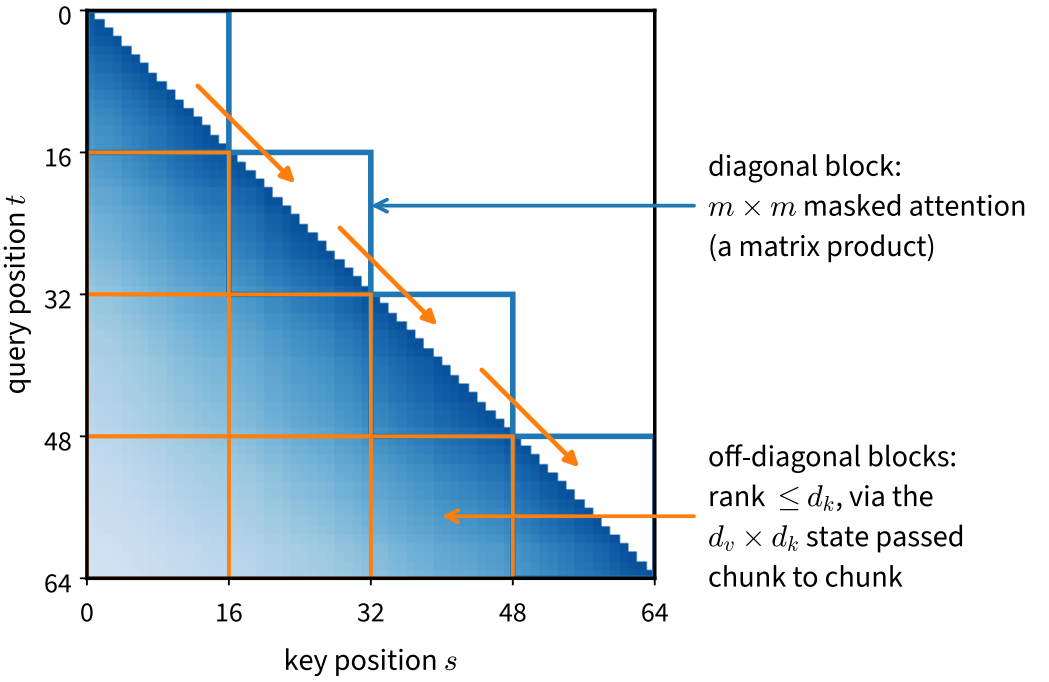


Figure 12.13: ランダムな減衰に対するマスク $\mathbf{L}$（Equation 12.31）をチャンクに分割した様子。対角ブロック（青枠）は並列に計算される小さなマスク付きアテンションの積である。対角より下の各ブロック（オレンジ枠）は $\mathbf{L}$ においてランク 1 であり、また高々ランク d_k を $\mathbf{L} \odot \mathbf{QK}^\top$ において持ち、チャンクからチャンクへ渡される状態（矢印）によって処理される。減衰が重なるにつれて陰影は薄くなっていく。

3. **過去状態の読み出し（非対角ブロック）：**$(a_1 \cdots a_t)\,\mathbf{S}^{(c-1)}\mathbf{q}_t$ を各位置 t（チャンク c 内）に加算する。

これで FLOPs の大部分は密な行列積となり、ピーク時の作業メモリは $O(T^2)$ から $O(Tm + (T/m)d_v d_k)$ へと減少する。

```
def chunked_linear_attention(Q, K, V, log_a, chunk_size):
    """y_t = S_t q_t with S_t = a_t S_{t-1} + v_t k_t^T, computed in chunks.

    Q, K: (batch_size, T, d_k); V: (batch_size, T, d_v); log_a: (batch_size, T)
    """
    B, T, m = Q.shape[0], Q.shape[1], chunk_size
    Q, K, V = (X.reshape(B, T // m, m, -1) for X in (Q, K, V))
    log_a = log_a.reshape(B, T // m, m)
    # 1. Within chunks: the dual form, one m x m mask per chunk
    Y = (decay_mask(log_a) * (Q @ K.mT)) @ V
    # 2. Across chunks: S^(c) = gamma^(c) S^(c-1) + sum_t L_mt v_t k_t^T
    cum = log_a.cumsum(-1)[..., None]  # Log decay from the chunk's start
    writes = V.mT @ (K * jnp.exp(cum[:, :, -1:] - cum))
    S = d2l.linear_scan(jnp.exp(cum[:, :, -1:]), writes)  # At chunk ends
    S = jnp.concatenate([jnp.zeros_like(S[:, :1]), S[:, :-1]], 1)  # Starts
    # 3. Every step reads the state at its chunk's start, decayed to it
    return (Y + (Q * jnp.exp(cum)) @ S.mT).reshape(B, T, -1)
```

逐次的な漸化式、完全な双対形式、およびチャンクサイズ $m \in \{16, 64, 256\}$ にわたるチャンク化アルゴリズムが、長さ $T = 512$ の系列上で float32 の精度で一致することを確かめてみよう。

```
T, d_k, d_v = 512, 64, 64
key_q, key_k, key_v, key_a = jax.random.split(jax.random.key(0), 4)
Q, K = jax.random.normal(key_q, (T, d_k)), jax.random.normal(key_k, (T, d_k))
V = jax.random.normal(key_v, (T, d_v))
a = 0.8 + 0.19 * jax.random.uniform(key_a, (T,))  # Decays in (0.8, 0.99)
with jax.default_matmul_precision('highest'):  # Full float32, not TF32
    Y = linear_attention_recurrent(Q, K, V, a)
    forms = {'dual form': linear_attention_dual(Q, K, V, a)}
    for m in (16, 64, 256):
        forms[f'chunks of {m}'] = chunked_linear_attention(
            Q[None], K[None], V[None], jnp.log(a)[None], m)[0]
for name, Y_form in forms.items():
    print(f'{name:13s} relative deviation from the recurrence: '
          f'{jnp.abs(Y_form - Y).max() / jnp.abs(Y).max():.1e}')
```

```
dual form     relative deviation from the recurrence: 2.4e-06
chunks of 16  relative deviation from the recurrence: 3.1e-07
chunks of 64  relative deviation from the recurrence: 3.9e-07
chunks of 256 relative deviation from the recurrence: 9.4e-07
```

すべての形式が相対誤差 $\sim 10^{-6}$ の範囲で一致する（最適なチャンクサイズ m については演習問題 3 で解析する）。

12.6.4 ゲート付き線形アテンション層

12.6.4.1 層の構成

GatedLinearAttention は、マルチヘッドのチャンク化線形アテンションを系列ミキサとしてまとめた層である。各ヘッド h に対して入力依存のスカラー減衰 $a_{t,h} = \sigma(\mathbf{w}_h^\top \mathbf{x}_t + b_h)$ を計算し、そのバイアスは init_decays によって初期の時間スケール τ_h（$a_{t,h} \approx e^{-1/\tau_h}$）がヘッド間で $[4, 256]$ ステップの範囲に対数一様に分布するよう初期化する。

```
class GatedLinearAttention(nnx.Module):
    """Multi-head linear attention with one input-dependent decay per head."""
    def __init__(self, num_hiddens, num_heads, chunk_size=16, conv_width=4,
                 timescales=(4, 256), *, rngs):
        d = num_hiddens
        self.num_heads, self.chunk_size = num_heads, chunk_size
        self.W_qkv = nnx.Linear(d, 3 * d, use_bias=False, rngs=rngs)
        self.conv = nnx.Conv(3 * d, 3 * d, (conv_width,),  # Causal, one
                             feature_group_count=3 * d,  # filter per channel
                             padding=((conv_width - 1, 0),), rngs=rngs)
        self.W_a = nnx.Linear(d, num_heads, rngs=rngs)  # Decay, per head
        self.W_g = nnx.Linear(d, d, rngs=rngs)  # Output gate
        self.W_o = nnx.Linear(d, d, use_bias=False, rngs=rngs)
        self.init_decays(*timescales)  # Memories of 4 to 256 steps by default

    def init_decays(self, tau_min, tau_max):
        """Start the decays at exp(-1/tau), tau log-uniform in range."""
        b = self.W_a.bias
        tau = jnp.logspace(math.log10(tau_min), math.log10(tau_max), len(b))
        b[...] = -jnp.log(jnp.expm1(1 / tau))  # log sigmoid(b) = -1 / tau
```

メモリの更新処理は memory メソッドに分離しておく。これにより、12.7 節ではブロックの残りの部分を重複して書くことなく GatedLinearAttention をサブクラス化し、デルタ則に差し替えることができる。heads_to_batch と batch_to_heads は、ヘッド軸をバッチ軸へ折り畳む操作と元に戻す操作を担う。

```
def heads_to_batch(Z, num_heads):
    """(batch_size, T, num_heads * e) -> (batch_size * num_heads, T, e)."""
    B, T, _ = Z.shape
    return Z.reshape(B, T, num_heads, -1).swapaxes(1, 2).reshape(
        B * num_heads, T, -1)

def batch_to_heads(Y, batch_size):
    """(batch_size * num_heads, T, e) -> (batch_size, T, num_heads * e)."""
    _, T, e = Y.shape
    return Y.reshape(batch_size, -1, T, e).swapaxes(1, 2).reshape(
        batch_size, T, -1)

@d2l.add_to_class(GatedLinearAttention)
def decays(self, X):
    """Log decays, (batch_size * num_heads, T)."""
    return heads_to_batch(jax.nn.log_sigmoid(self.W_a(X)),
                          self.num_heads)[..., 0]
```

```
@d2l.add_to_class(GatedLinearAttention)
def memory(self, X, Q, K, V, log_a):
    """Outputs of the decayed matrix memory of every head. X is unused here;
    the delta rule of the next section reads its write strengths from it."""
    return chunked_linear_attention(Q, K, V, log_a, self.chunk_size)
```

順伝播（Figure 12.14）は次の手順で進む。

1. 幅 4 のデプスワイズ因果畳み込みと SiLU によって隣接トークンを混合して $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ を作る。これにより、位置 t の値トークンが単一の層の内部で位置 $t-1$ のキートークンに直接束縛できるようになる（12.8 節ではこの畳み込みをアブレーションする）。
2. クエリとキーを単位長へ ℓ_2 正規化する（Equation 12.28 の設定と一致させる）。
3. 各ヘッドの読み出しに RMS 正規化を施し、$\mathrm{SiLU}(\mathbf{W}_g \mathbf{x}_t)$ でゲートを掛けたうえで元の幅 d へ出力射影（output projection）する (S. Yang, Wang, Shen, et al. 2024)。

```
@d2l.add_to_class(GatedLinearAttention)
def __call__(self, X):  # X: (batch_size, num_steps, num_hiddens)
    QKV = jax.nn.silu(self.conv(self.W_qkv(X)))
    Q, K, V = (heads_to_batch(Z, self.num_heads)
               for Z in jnp.split(QKV, 3, -1))
    Q, K = (Z / jnp.maximum(jnp.linalg.norm(Z, axis=-1, keepdims=True), 1e-12)
            for Z in (Q, K))  # Unit length
    Y = self.memory(X, Q, K, V, self.decays(X))  # (B * num_heads, T, d / h)
    Y = Y / jnp.sqrt((Y ** 2).mean(-1, keepdims=True) + 1e-6)  # Unit RMS
    return self.W_o(batch_to_heads(Y, X.shape[0]) * jax.nn.silu(self.W_g(X)))
```

12.6.4.2 連想想起

訓練されたニューラルネットワークは、ランダムキーに対して導出した $O(d_k d_v)$ の容量則に従って行列状態を活用するのだろうか。合成ベンチマークである**マルチクエリ連想想起**（multi-query associative recall）(Arora, Eyuboglu, Timalsina, et al. 2024) を用いてこれを検証する。各系列では、まず n 個のキー・値ペア（互いに異なるキートークンの直後に、4 クラスから選ばれた値トークンが続く組）を提示し、任意個のフィラートークンを挟んだあと、最後に全 n 個のキーをランダムな順序でクエリとして再度提示する。（オプションの `num_rebound` と `gap` は一部のキーを新しい値で上書きするためのもので、12.7 節で使用する。）

```
def associative_recall(rng, batch_size, num_pairs, num_keys=512,
                       num_values=4, num_steps=256, num_rebound=0, gap=None):
    """Key-value pairs, filler, then every key again as a query. The first
    num_rebound keys are written twice, the second time with a new value,
    at any later write or, given gap, gap writes after the first one.
    Returns the tokens (values, then keys, then the filler), the latest
    value of each query's key, and a mask of the queries for rebound keys."""
    n, r = num_pairs, num_rebound
    keys = np.argsort(rng.random((batch_size, num_keys)), 1)[:, :n]
    values = rng.integers(0, num_values, (batch_size, n + r))
    values[:, n:] = (values[:, :r] + rng.integers(  # A new value
        1, num_values, (batch_size, r))) % num_values
    if gap is None:  # Random write order, each second write after the first
```

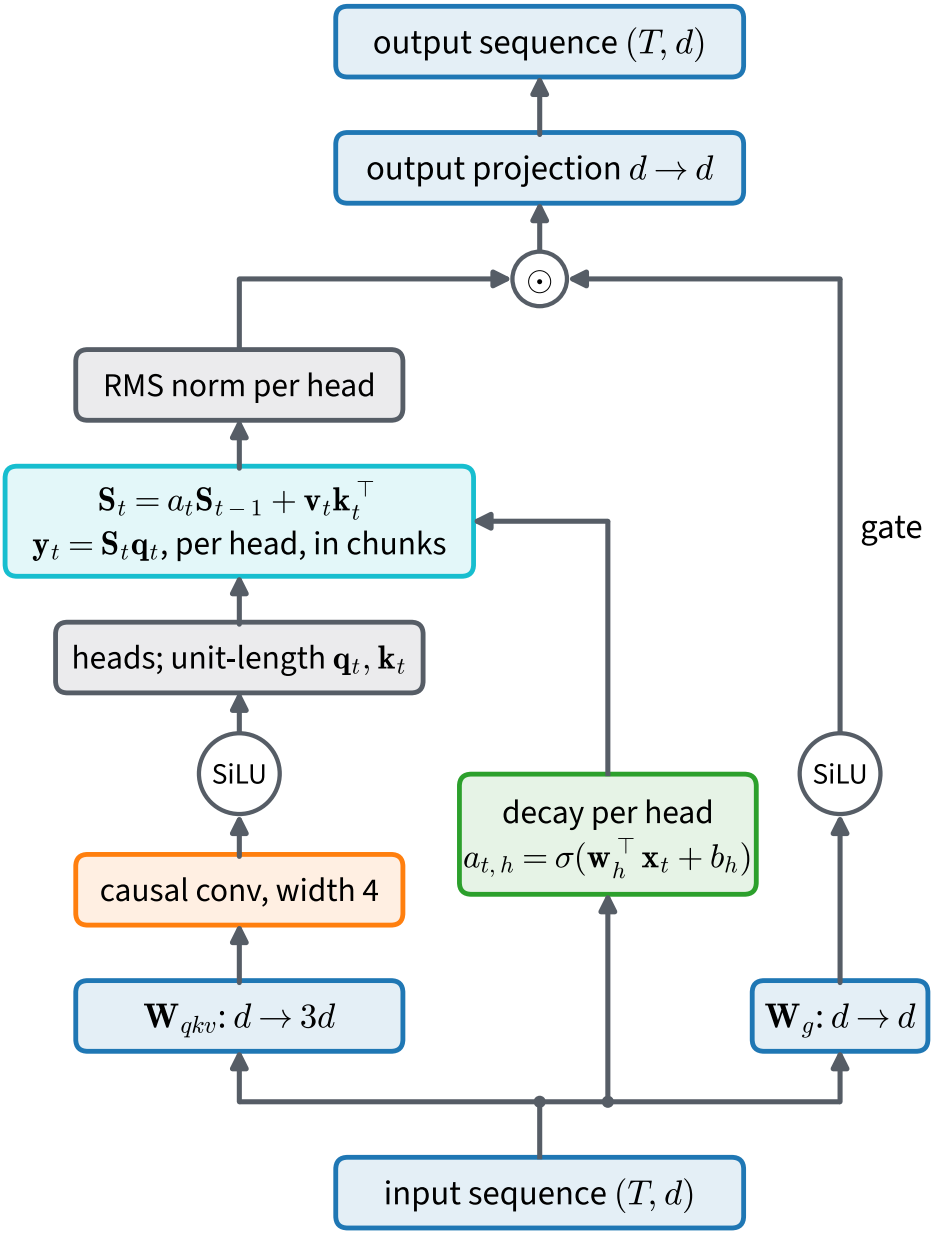


Figure 12.14: ゲート付き線形アテンション層。1 つ目の射影と短い因果畳み込みから全ヘッドのクエリ・キー・値が得られ、2 つ目の射影から各ヘッドの減衰 a_t が得られる。行列メモリはチャンク単位で計算される。その読み出しはヘッドごとに RMS 正規化され、入力から計算されたゲートを掛け合わせたのち、幅 d へ射影される。

```
        slots = np.argsort(rng.random((batch_size, n + r)), 1)
        slots[:, :r], slots[:, n:] = (np.minimum(slots[:, :r], slots[:, n:]),
                                      np.maximum(slots[:, :r], slots[:, n:]))
    else:  # First writes in random order, second ones gap first writes later
        first = np.argsort(rng.random((batch_size, n)), 1)
        times = np.concatenate([first, first[:, :r] + gap - 0.5], 1)
        slots = np.argsort(np.argsort(times, 1), 1)
    X = np.full((batch_size, num_steps), num_values + num_keys)  # Filler
    rows = np.arange(batch_size)[:, None]
    X[rows, 2 * slots] = num_values + np.concatenate([keys, keys[:, :r]], 1)
    X[rows, 2 * slots + 1] = values
    order = np.argsort(rng.random((batch_size, n)), 1)  # Queries
    X[:, -n:] = num_values + np.take_along_axis(keys, order, 1)
    latest = np.concatenate([values[:, n:], values[:, r:n]], 1)
    return X, np.take_along_axis(latest, order, 1), order < r
```

4 個のペアと 1 個の再束縛キーを含む短い系列を確認してみよう（トークン `0..3` が値、`4..515` がキー、`516` がフィラーである）。

```
X, Y, rebound = associative_recall(np.random.default_rng(0), 1, 4,
                                   num_steps=16, num_rebound=1)
print('tokens: ', X[0])
print('answers:', Y[0], ' written twice:', rebound[0])
```

```
tokens:  [273   3  15   3 200   2 407   0 273   0 516 516 200 407 273  15]
answers: [2 0 0 3]  written twice: [False False  True False]
```

訓練ハーネス `train_recall` を保存しておく。この関数は、2,000 ステップの AdamW（コサインウォームアップ付き、ピーク学習率 0.01）でペア数 $n \in \{4, 8, 16, 32, 64\}$ を巡回しながらモデルを訓練し、各負荷における想起率を 1,024 本の新規系列で評価する。

```
def train_recall(model, pairs=(4, 8, 16, 32, 64), num_updates=2000,
                 **task):
    """Train on associative recall (task: options of associative_recall)
    with a cycling number of pairs, 256 sequences per update, AdamW at peak
    rate 0.01; return the accuracy per number of pairs."""
    rng = np.random.default_rng()
    length = lambda n: 32 * math.ceil(
        (3 * n + 2 * task.get('num_rebound', 0)) / 32)
    data = lambda n, size: associative_recall(
        rng, size, n, num_steps=length(n), **task)[:2]
    schedule = lambda t: 1e-2 * d2l.cosine_warmup(t, num_updates,
                                                  num_updates // 10)
    optimizer = nnx.Optimizer(model, optax.chain(
        optax.clip_by_global_norm(1.0),
        optax.adamw(schedule, weight_decay=0.01)), wrt=nnx.Param)
    d2l.train_queries(model, lambda step: data(pairs[step % len(pairs)],
                      256), num_updates, optimizer)
    return {n: float(d2l.query_accuracy(model, *data(n, 1024)))
            for n in pairs}
```

モデル幅（model width）やパラメータ数（parameter count）を変え**ずに**行列状態の容量を変

化させるため、2 ブロックの SeqLM を幅 $d = 32$ で memory_model(num_heads) 内に固定し、その幅を $h \in \{8, 4, 2\}$ 個のヘッド（各次元 $e = d/h \in \{4, 8, 16\}$）に分割する。

```
def memory_model(num_heads, num_hiddens=32, vocab_size=4 + 128 + 1):
    """Two blocks of gated linear attention; tokens: values, keys, filler."""
    rngs = nnx.Rngs(d2l.get_key())  # A new initialization each call
    return d2l.SeqLM(vocab_size, num_hiddens, [GatedLinearAttention(
        num_hiddens, num_heads, timescales=(256, 2048), rngs=rngs)
        for _ in range(2)], rngs=rngs)
```

各ヘッドは $e \times e$ の状態行列を保持するため、層あたりの状態の総要素数は $h \cdot e^2 = d \cdot e \in \{128, 256, 512\}$ とスケールする一方、総パラメータ数はほぼ同一（約 37,000）に保たれる。各ヘッド構成について train_recall で 2 回ずつ訓練を実行する。

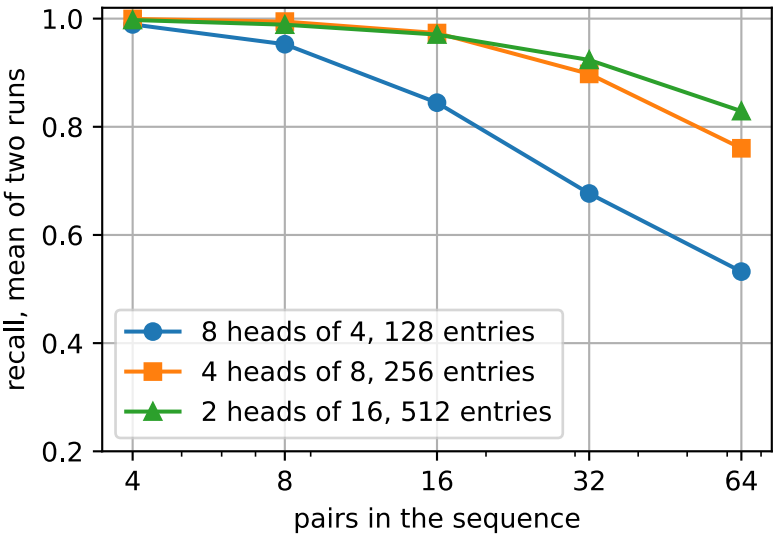
```
recall = {}
for num_heads in (8, 4, 2):
    d_h = 32 // num_heads
    name = f'{num_heads} heads of {d_h}, {num_heads * d_h ** 2} entries'
    models = [memory_model(num_heads) for _ in range(2)]
    recall[name] = [train_recall(model, num_keys=128) for model in models]
    print(f'{name:26s} {d2l.num_params(models[0]):,} parameters')
```

```
8 heads of 4, 128 entries  37,461 parameters
4 heads of 8, 256 entries  37,197 parameters
2 heads of 16, 512 entries 37,065 parameters
```

平均想起正解率をペア数 $n \in \{4, 8, 16, 32, 64\}$ に対してプロットおよび集計すると、検索容量が状態の総要素数に応じてどのようにスケールするかがわかる。

```
pairs = list(recall[name][0])
d2l.plot(pairs, [[np.mean([run[n] for run in runs]) for n in pairs]
                 for runs in recall.values()], 'pairs in the sequence',
         'recall, mean of two runs', legend=list(recall), xscale='log',
         ylim=(0.2, 1.02), fmts=('-o', '-s', '-^'), figsize=(4.5, 3))
d2l.plt.xticks(pairs, pairs)  # Ticks at the numbers of pairs
print(f'{"pairs":26s}', ''.join(f'{n:>11d}' for n in pairs))
for name, runs in recall.items():
    print(f'{name:26s}', ''.join(f'{runs[0][n]:6.2f}/{runs[1][n]:.2f}'
                                 for n in pairs))
```

```
pairs                              4          8         16         32         64
8 heads of 4, 128 entries    0.98/0.99  0.95/0.96  0.84/0.85  0.68/0.68  0.52/0.54
4 heads of 8, 256 entries    1.00/1.00  1.00/0.99  0.97/0.98  0.87/0.92  0.72/0.80
2 heads of 16, 512 entries   0.99/1.00  0.98/1.00  0.94/1.00  0.85/1.00  0.67/0.99
```



3 つの構成はいずれも 4 ペアの課題を解き（$\geq 98\%$）、ペア数が増えるにつれて想起率は状態の要素数に直結してスケールする。状態容量を 128 要素（8×4^2）から 256 要素（4×8^2）に倍増させると、同じ正解率で想起できるペア数もおよそ 2 倍になり（~ 0.50 から ~ 0.75 へ、64 ペアのとき）、512 要素（2×16^2）は 128 要素を ≥ 16 のすべての負荷で上回る。

ただし、すべての加法的な行列メモリには構造上の欠陥が 1 つ残っている。$\mathbf{v}_t \mathbf{k}_t^\top$ の書き込みは、$\mathbf{k}_t$ のもとにすでに保存されている値へ単純に**加算する**にすぎない点である。同じキーが 2 回目に新しい値へ束縛されると、そのキーをクエリした際には古い値と新しい値の重ね合わせが返ってしまう。キーの値を**上書きする**方法をメモリに獲得させることが、12.7 節の主題である。

12.6.5 まとめ

行列値の状態 $\mathbf{S}_t \in \mathbb{R}^{d_v \times d_k}$ は、外積の書き込み $\mathbf{v}_t \mathbf{k}_t^\top$ によってキー・値ペアを保存し、線形クエリ $\mathbf{S}_t \mathbf{q}_t$ によってそれらを検索する。この漸化式を展開すると因果線形アテンションになり、Mamba の選択的 SSM はキーを $\mathbf{B}_t$、クエリを $\mathbf{C}_t$ としたその一例である。ランダムな単位キーのもとでは、ペア間のクロストークによって期待二乗検索誤差 $(n-1)/d_k$（n 回の書き込み後）が生じる一方、最近傍想起の容量は状態の総要素数 $d_k d_v$ に応じてスケールする。ヘッドごとに入力依存のスカラー減衰 $a_t \in (0,1)$ を導入すると古いペアが薄れ、因果マスク付き線形アテンション $\mathbf{Y} = (\mathbf{L} \odot \mathbf{Q}\mathbf{K}^\top)\mathbf{V}$ との**状態空間双対性**が成り立つ。この双対形式をチャンク単位で評価すれば、チャンク内の並列な行列積とチャンク間のスキャンが組み合わさり、$O(T)$ のメモリで計算できる。パラメータ数を固定したマルチクエリ連想想起では、`GatedLinearAttention` 層の状態要素数を 2 倍にすると想起できるペア数もおよそ 2 倍になるが、純粋に加法的な書き込みだけでは依然として過去に束縛されたキーを上書きできない。

12.6.6 演習問題

1. **行列メモリとしての Mamba**。$\mathbf{S}_t = \mathbf{E}_t \odot \mathbf{S}_{t-1} + (\Delta_t \odot \mathbf{x}_t)\mathbf{B}_t^\top$（ただし $\mathbf{E}_t = \exp(\mathrm{diag}(\Delta_t)\Lambda)$）の形において、減衰の対数は行 c をステップサイズ $\Delta_{t,c}$ でスケールしたレートの行列となる。
    1. Δ_t およびレート Λ にどのような条件を課せば、$\mathbf{E}_t$ が全要素 1 の行列のスカラー倍となり、更新式が Equation 12.29 の形になるか。Mamba-2 はこれらの条件のうちどれを課し、どれを入力依存のまま残しているか。

   2. 本節で扱った各減衰形式（RetNet、Mamba-2、ゲート付き線形アテンション、Mamba）について、$d_v \times d_k$ 要素からなる 1 つのヘッドの減衰を決定するトークンあたりの入力依存の数値の個数を数えよ。
2. **ランク 1 ブロック**。$p_t = a_1 a_2 \cdots a_t$ を Equation 12.31 における減衰の積とする。
   1. L_{ts} を p_t を用いて表し、行インデックスがすべて列インデックス以上であるような $\mathbf{L}$ の任意の部分行列がランク 1 を持つことを示せ。
   2. あるチャンクをそれより前のチャンクと結合する $\mathbf{L} \odot \mathbf{Q}\mathbf{K}^\top$ のすべてのブロックが、高々 d_k のランクを持つことを導け。また、`chunked_linear_attention` がそのようなブロックのクエリ側の減衰とキー側の減衰をどこで適用しているかを特定せよ。
   3. キー座標ごとの減衰 $\mathbf{S}_t = \mathbf{S}_{t-1}\mathrm{diag}(\alpha_t) + \mathbf{v}_t \mathbf{k}_t^\top$ を用いる場合、単一のマスク $\mathbf{L}$ を $\mathbf{Q}\mathbf{K}^\top$ に掛ける形にはならない。クエリ t とキー s の減衰付きスコアを書き下し、2 つのチャンクを結合するブロックのランクが依然として高々 d_k であることを示せ。
3. **演算数のカウント**。1 回の積和演算を 2 回の浮動小数点演算と数え、$d_k = d_v = d$ において主要項のみを残せ。
   1. 系列長 T、チャンクサイズ m における漸化式、双対形式、チャンク形式の演算数を数えよ。チャンク形式の演算数はアルゴリズムの 3 つのパートに分けて求めること。
   2. $m \le d$ の範囲ではチャンク形式のコストが高々漸化式の小さな定数倍に収まることを示し、$T = 4096$、$d = 64$、$m = 256$ におけるチャンク形式と双対形式の演算数の比を計算せよ。
   3. あるスケジュールの実行時間を、その演算数をスループット P で割った値に、逐次ステップあたり一定の所要時間 τ（漸化式では T ステップ、チャンク形式では T/m ステップ）を加えたものとしてモデル化する。チャンク形式の実行時間を最小化するチャンクサイズを求め、$T = 4096$、$d = 64$、毎秒 $P = 10^{13}$ 演算、$\tau = 5\,\mu\mathrm{s}$ の場合について評価せよ。このモデルによれば、チャンク形式は漸化式より何倍高速になると予測されるか。
4. **相関のあるキー**。●`matrix_memory` における $\rho > 0$ のキーは、$\sqrt{1-\rho^2}\,\mathbf{g}_i + \rho\,\mathbf{c}$（ただし $\mathbf{g}_i$ は独立なランダム単位ベクトル、$\mathbf{c}$ は 1 つの共通単位ベクトル）を正規化したものである。本文の式では、相異なる 2 つのキーの重なりを $\rho^2 + \epsilon_{ij}$（ただし ϵ_{ij} は平均ゼロ・分散 $(1-\rho^2)^2/d_k$ で独立）とモデル化し、値の平均がゼロであると仮定している。
   1. `matrix_memory` の値は 256 個のランダムなベクトルからなるコードブックから選ばれており、その平均 $\bar{\mathbf{v}}$ は厳密にはゼロではない。これにより期待二乗誤差に $(n-1)(n-2)\,\|\bar{\mathbf{v}}\|^2$ に比例する項が加わることを示し、その完全な予測式を実測誤差と比較せよ。
   2. コードブックを中心化（平均を引いてから各要素を再度正規化）し、曲線の上方への曲がりが消えることを確認せよ。
   3. $\rho \in \{0, 0.25, 0.5, 0.75\}$ について、$d_k = 64$ と $d_k = 256$ における誤差と想起率を測定せよ。どの ρ までならキーの幅を広げることが依然として有効であり、どこから誤差が $(n-1)\rho^4$ に近づくか。
5. [code] **幅の異なるキーと値**。●`matrix_memory` に独立した値の幅 `d_v` を持たせ、コードブックが $\mathbb{R}^{d_v}$ 内の 256 個のランダムな単位ベクトルを保持するようにせよ。
   1. $n/(d_k d_v) = 1/32$ を一定に保ち、$(d_k, d_v) = (64, 64)$、$(256, 16)$、$(16, 256)$ について想起率を測定せよ。想起率は積 $d_k d_v$ のみに依存するか。
   2. 期待二乗読み出し誤差は d_v に依存しないため、測定された結果をそれだけでは説明できないことを示せ。
   3. 各コードブック要素はおよそ $n/256$ 個のキーのもとに保存されている。それらと読み出しキーとの重なりが、その要素の方向に分散およそ $n/(256\,d_k)$ の成分を加えることを示し、これを用いてどのような形状で積の法則が破綻するかを説明せよ。
6. [code] **スライディングウィンドウとしての減衰**。●`matrix_memory` に定数減衰 a を追加し、末尾から ℓ ステップ前に書き込まれたペアが重み a^ℓ で最終的なメモリに残るようにせよ。
   1. $d_k = 64$、$n = 256$ において、$a \in \{1, 0.99, 0.95, 0.9\}$ のそれぞれに対し、最も古

い 4 分の 1 のペアと最も新しい 4 分の 1 のペアの想起率を別々に測定せよ。減衰は最も新しいペアに何をもたらし、最も古いペアにどのような代償を強いるか。

2. 命題の証明にならい、任意のキーにおける期待二乗干渉が高々 $1/((1-a^2)\,d_k)$ であることを示せ。さらに、二乗信号 $a^{2\ell}$ がこの上界を上回るペアの個数は高々およそ d_k/e 個にとどまることを、任意の $a < 1$ に対して示せ。

12.7 デルタ則

12.6 節の加算型行列メモリは、新しいペアごとに $\mathbf{v}_t\mathbf{k}_t^\top$ を既存の $\mathbf{S}_{t-1}$ に足し合わせることで記憶する。この仕組みは、すべてのキーが 1 度しか束縛されない限り機能する。しかし実際のテキストやコードでは、キーの**再束縛**（rebind）が頻繁に起こる。たとえば、プログラムが `rate = 0.05` を代入した後に `rate = 0.07` へ更新したり、物語の中で登場人物がパリからローマへ移動したりする場合である。加算的書き込みのもとでは、2 度書き込まれたキーをクエリすると古い値と新しい値の和が返され、更新を繰り返すほど最新の値は古い値の下に埋もれてしまう。固定サイズの行列メモリにおいて、あるキーに格納された値を上書きするにはどうすればよいだろうか。

その答えは、**書き込み前に読み出す**ことである。すなわち、$\mathbf{k}_t$ に対する現在のメモリの予測（$\hat{\mathbf{v}}_t = \mathbf{S}_{t-1}\mathbf{k}_t$）を計算し、予測誤差 $\beta_t(\mathbf{v}_t - \hat{\mathbf{v}}_t)\mathbf{k}_t^\top$ のみを書き込む。この**デルタ則**（delta rule）は (Widrow and Hoff 1960; Schlag et al. 2021)、順伝播中に二乗想起損失 $\frac{1}{2}\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2$ に対する勾配降下法のオンラインステップを 1 回実行することに相当する。以下では、デルタ則の射影幾何学を導出し、ランダムなキーと相関のあるキーにおける再束縛の利点と信号浸食（signal erosion）のコストの両方を測定する。さらに、スカラー減衰と組み合わせた `GatedDeltaNet` を導入し (S. Yang et al. 2025)、チャンク単位の三角求解によって評価したうえで (S. Yang, Wang, Zhang, et al. 2024)、最後に系列メモリをオンライン回帰ソルバとして捉える統一的な視点で締めくくる。

12.7.1 加算的書き込み

12.7.1.1 2 度書き込まれたキー

まず、キーが再束縛されたときに加算的書き込みがどのように破綻するかを定量化しよう。単位キー $\mathbf{k}$ が R 回書き込まれ、それぞれの値が $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(R)}$ であるとき、加算型メモリはそれらの重みなし和を返す。

$$\mathbf{S}\mathbf{k} = \mathbf{v}^{(1)} + \mathbf{v}^{(2)} + \cdots + \mathbf{v}^{(R)}, \tag{12.33}$$

さらに、キー間干渉 $\sum_i (\mathbf{k}_i^\top \mathbf{k})\mathbf{v}_i$ がこれに加わる。スカラー減衰 $a_t \in (0,1)$ を導入すれば新しい値に比べて古い値を減衰させられるが、同じ経過時間の**すべて**のキーを無差別に減衰させてしまうため、1 度だけ書き込まれた事実も、再束縛されたキーの古い値と同じ速さで薄れてしまう（演習問題 2）。

キー間干渉から再束縛を切り離すため、`overwrite_recall` は $\mathbb{R}^{64}$ 内の互いに正規直交する 8 個のキーをテストする。具体的には、32 個の単位ベクトルからなるコードブックから値を取り出して各キーを $R \in \{1, 2, 4, 8\}$ 回書き込み、$\mathbf{S}\mathbf{k}$ に対するコードブック内の最近傍が最新の値 $\mathbf{v}^{(R)}$ と一致する頻度を測定する。

```
def overwrite_recall(write, num_writes, d=64, num_keys=8, num_values=32,
                     trials=200, seed=0):
    """Recall of the latest value after every key is written num_writes
```

```
    times."""
    rng = np.random.default_rng(seed)
    codebook = rng.standard_normal((num_values, d))
    codebook /= np.linalg.norm(codebook, axis=1, keepdims=True)
    recall = 0
    for _ in range(trials):
        keys = np.linalg.qr(rng.standard_normal((d, num_keys)))[0].T  # Rows
        S = np.zeros((d, d))
        for _ in range(num_writes):
            ids = rng.integers(0, num_values, num_keys)
            for k, v in zip(keys, codebook[ids]):
                S = write(S, k, v)
        reads = keys @ S.T  # Row j is S k_j
        recall += ((reads @ codebook.T).argmax(1) == ids).mean() / trials
    return recall

additive = lambda S, k, v: S + np.outer(v, k)
writes = [1, 2, 4, 8]
print('additive:', ' '.join(f'{overwrite_recall(additive, R):.2f}'
                            for R in writes))
```

```
additive: 1.00 0.51 0.27 0.21
```

$\mathbb{R}^{64}$ 内の正規直交する 8 個のキー間で干渉がゼロであっても、加算型メモリの想起率は 1.00（$R = 1$）から 0.51（$R = 2$、足し合わされた 2 つの値の間でのコイントス）、さらに 0.21（$R = 8$）へと低下する（演習問題 1）。

12.7.2 勾配ステップとしての書き込み

12.7.2.1 デルタ則

他の方向をできるだけ乱さずに $\mathbf{k}_t$ に束縛された値を置き換えるため、新しく入力されたペア $(\mathbf{k}_t, \mathbf{v}_t)$ に対するメモリの二乗予測誤差を測る。

$$\ell_t(\mathbf{S}) = \tfrac{1}{2}\,\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2, \tag{12.34}$$

そして、勾配降下法を 1 ステップ進める更新 $\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t \nabla_{\mathbf{S}} \ell_t(\mathbf{S}_{t-1})$ を、ステップサイズすなわち**書き込み強度**（write strength）を β_t として適用すると次のようになる。

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \beta_t(\mathbf{S}_{t-1}\mathbf{k}_t - \mathbf{v}_t)\mathbf{k}_t^\top. \tag{12.35}$$

これが**デルタ則**である (Widrow and Hoff 1960; Schlag et al. 2021)。現在の予測 $\hat{\mathbf{v}}_t = \mathbf{S}_{t-1}\mathbf{k}_t$ を読み出し、ランク 1 の補正 $\beta_t(\mathbf{v}_t - \hat{\mathbf{v}}_t)\mathbf{k}_t^\top$ を加える。なお、この勾配ステップは訓練時と推論時の両方において、各トークンの**順伝播の内部で**実行される。単位キー $\|\mathbf{k}_t\| = 1$ に対して Equation 12.35 の直後に $\mathbf{k}_t$ をクエリすると、$\mathbf{S}_t\mathbf{k}_t = (1-\beta_t)\hat{\mathbf{v}}_t + \beta_t\mathbf{v}_t$ が返される。これは $\beta_t = 1$ のとき厳密な置換となり、$\beta_t \in (0,1)$ のとき凸結合となる（対照的に、加算的書き込み $\mathbf{v}_t\mathbf{k}_t^\top$ は線形アライメント $\mathbf{v}_t^\top\mathbf{S}\mathbf{k}_t$ に対する勾配上昇法のステップであり、飽和点を持たない）。

12.7.2.2 遷移

$\mathbf{S}_{t-1}$ を Equation 12.35 からくくり出すと、既存のメモリに作用する行列遷移が明らかになる。

$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top . \tag{12.36}$$

単位キー $\mathbf{k}_t$ に対して、対称行列 $\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$ は $\mathbf{k}_t$ に直交するすべての部分空間を不変に保ち（固有値 1）、$\mathbf{k}_t$ の方向を固有値 $1 - \beta_t$ でスケーリングする（Figure 12.15）。

- $\beta_t = 0$：恒等変換（書き込みなし）。
- $\beta_t \in (0, 1)$：$\mathbf{k}_t$ 方向の部分縮小。
- $\beta_t = 1$：直交射影 $\mathbf{I} - \mathbf{k}_t \mathbf{k}_t^\top$。$\mathbf{k}_t$ 成分を完全に消去してから $\mathbf{v}_t \mathbf{k}_t^\top$ を書き込む。
- $\beta_t = 2$：ハウスホルダー鏡映（Householder reflection）$\mathbf{I} - 2\mathbf{k}_t \mathbf{k}_t^\top$（固有値 -1）。$\mathbf{k}_t$ 方向の符号を反転させる。

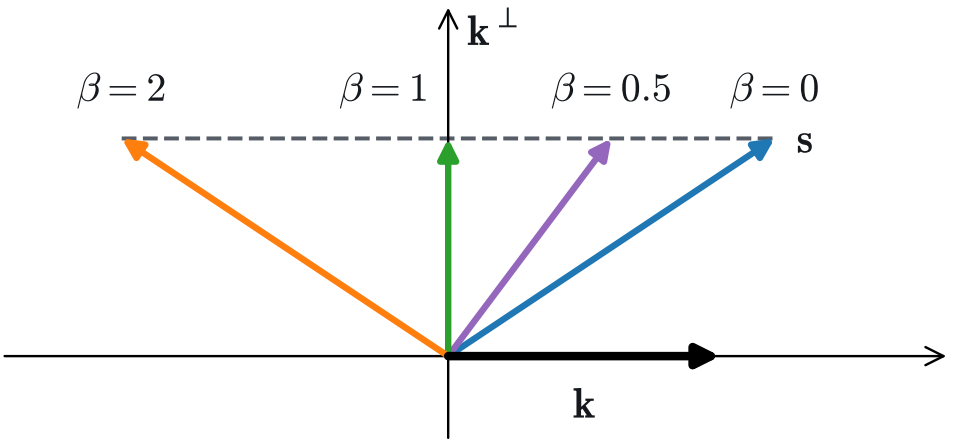


Figure 12.15: 遷移 $\mathbf{I} - \beta \mathbf{k}\mathbf{k}^\top$ が、ベクトル $\mathbf{s}$ に対して単位キー $\mathbf{k}$ とそれに直交する方向が張る平面内で作用する様子。$\mathbf{k}$ 方向の成分は $1 - \beta$ 倍され、$\beta = 0$ ではそのまま保たれ、$\beta = 0.5$ では半分になり、$\beta = 1$ では除去され、$\beta = 2$ では反転する。直交成分は変化しない。

固有値 $1 - \beta_t \|\mathbf{k}_t\|^2$ から、2 つの帰結が導かれる。

1. **単位ノルムキーが不安定性を防ぐ：**$\|\mathbf{k}_t\| = 1$ かつ $\beta_t \in [0, 2]$ のとき、遷移のすべての固有値は $[-1, 1]$ に収まるため、斉次なメモリ状態が幾何級数的に発散することはない。
2. **負の固有値が法 2 のカウント（パリティ）を可能にする：**$\beta_t \in [0, 1]$ のとき、遷移の固有値はすべて非負となり、有限精度では任意の系列長にわたってパリティを追跡できない（Grazzi et al. 2025）。β_t の範囲を 2 まで広げると、カウント対象のトークンごとに記憶状態を反転させる鏡映（固有値 -1）が可能になり、任意の長さで厳密なパリティを計算できる。12.7.3.5 節では、訓練された層がこの解を見つけられるかを検証する。

12.7.2.3 補正のコスト

正規直交するキーに対しては、$\beta_t = 1$ とすることで他のキーに一切触れずに再束縛されたキーの古い値を消去できる。では、$n > d_k$ の場合や、キーがランダムな単位ベクトルであって異なるキー $\mathbf{k}_i$ と $\mathbf{k}_t$ が非ゼロの重なり $\mathbf{k}_t^\top \mathbf{k}_i$ を持つ場合はどうなるだろうか。後続の各デルタステップは $\mathbf{k}_t$ に沿って進み、$\mathbf{k}_t$ 方向の成分を $\mathbf{v}_i \mathbf{k}_i^\top$ から射影除去するため、キー i に記憶された信号は 1 ステップあたり $1 - (\mathbf{k}_t^\top \mathbf{k}_i)^2 \approx 1 - 1/d_k$ 倍に縮小する。その結果、その後に m 回の書き込みが行われると、古いペアの信号は $(1 - 1/d_k)^m \approx e^{-m/d_k}$ しか残らない。

対象を絞った消去がいつ有効でいつ有害になるかを確認するため、delta_rule(beta) でオンラインのデルタ書き込みを実装し、correction_cost で複数の書き込みラウンドとキー間の相関にわたり幅 $d = 64$ での二乗読み出し誤差と最近傍想起率の両方を測定する。

```
def delta_rule(beta):
    """The memory after writing the pairs in order with the delta rule."""
    def fit(K, V):
        S = np.zeros((V.shape[1], K.shape[1]))
        for k, v in zip(K, V):
            S = S + beta * np.outer(v - S @ k, k)
        return S
    return fit

def correction_cost(memories, num_keys, writes=1, orthonormal=False, rho=0,
                    d=64, num_values=256, trials=20, seed=0):
    """Squared read error and recall of the latest value when each of
    num_keys keys is written `writes` times, for each memory."""
    rng = np.random.default_rng(seed)
    unit = lambda X: X / np.linalg.norm(X, axis=-1, keepdims=True)
    codebook = unit(rng.standard_normal((num_values, d)))
    error, recall = dict.fromkeys(memories, 0), dict.fromkeys(memories, 0)
    for _ in range(trials):
        if orthonormal:
            keys = np.linalg.qr(rng.standard_normal((d, num_keys)))[0].T
        else:  # Random unit keys that share one direction with weight rho
            keys = unit(np.sqrt(1 - rho**2) * unit(rng.standard_normal(
                (num_keys, d))) + rho * unit(rng.standard_normal(d)))
        ids = rng.integers(0, num_values, (writes, num_keys))  # Row = round
        K, V = np.tile(keys, (writes, 1)), codebook[ids.ravel()]
        for name, fit in memories.items():
            reads = keys @ fit(K, V).T  # Row j is S k_j
            error[name] += ((reads - codebook[ids[-1]]) ** 2).sum(1).mean()
            recall[name] += ((reads @ codebook.T).argmax(1) == ids[-1]).mean()
    return ({name: e / trials for name, e in error.items()},
            {name: r / trials for name, r in recall.items()})
```

ここでは 4 つのメモリ規則、すなわち additive、delta, β = 1、delta, β = 0.5、およびオフラインの least squares（np.linalg.lstsq により全ペアで同時に $\sum_s \|\mathbf{S}\mathbf{k}_s - \mathbf{v}_s\|^2$ を最小化する最小二乗法）を、次の 3 つの設定で比較する。(1) 各 4 回書き込まれる 8 個の正規直交キー、(2) 1 回書き込まれる $n \in \{32, 64, 128, 256\}$ 個の独立なランダム単位キー、(3) 1 回書き込まれる相関のある 64 個のキー（$\rho = 0.5$）である。

```
memories = {'additive': lambda K, V: V.T @ K,  # Sum of the v k^T
            'delta, β = 1': delta_rule(1), 'delta, β = 0.5': delta_rule(0.5),
            'least squares': lambda K, V: np.linalg.lstsq(K, V)[0].T}
cases = {'8 orthonormal keys, 4 writes': dict(num_keys=8, writes=4,
                                              orthonormal=True),
         **{f'{n} keys': dict(num_keys=n) for n in (32, 64, 128, 256)},
         '64 keys, ρ = 0.5': dict(num_keys=64, rho=0.5)}
results = {case: correction_cost(memories, **args)
           for case, args in cases.items()}
for i, measure in enumerate(['squared read error', 'recall']):
```

```
    print(f'{measure:28s}', ''.join(f'{name:>15s}' for name in memories))
    for case, result in results.items():
        print(f'{case:28s}', ''.join(f'{result[i][name]:15.2f}'
                                     for name in memories))
```

squared read error	additive	delta, β = 1	delta, β = 0.5	least squares
8 orthonormal keys, 4 writes	2.98	0.00	0.33	0.74
32 keys	0.49	0.29	0.37	0.00
64 keys	0.98	0.60	0.49	0.00
128 keys	1.99	1.06	0.67	0.50
256 keys	3.94	1.49	0.89	0.75
64 keys, ρ = 0.5	5.82	0.76	0.61	0.00
recall	additive	delta, β = 1	delta, β = 0.5	least squares
8 orthonormal keys, 4 writes	0.28	1.00	1.00	0.28
32 keys	1.00	1.00	1.00	1.00
64 keys	1.00	0.89	0.99	1.00
128 keys	0.96	0.53	0.82	1.00
256 keys	0.73	0.26	0.45	0.86
64 keys, ρ = 0.5	0.54	0.77	0.89	1.00

この表から、オンライン射影のトレードオフが浮き彫りになる。

- **再束縛（8 orthonormal keys, 4 writes）：**delta は想起率 1.00 を $\beta = 1$ と $\beta = 0.5$ の両方で達成する（二乗誤差はそれぞれ 0.00 と 0.33）のに対し、additive と重みなしの least squares は各キーに束縛された 4 つの値すべてを平均してしまい、スコアはわずか 0.28 にとどまる。
- **相関のあるキー（64 keys, ρ = 0.5）：**すべてのキーが共通の方向を持つため、additive はあらゆる値をその共有成分の上に積み重ねてしまう（二乗誤差 5.82、想起率 0.54）。一方、delta は書き込み前に共有方向の予測を差し引くため、想起率は 0.77（$\beta = 1$）および 0.89（$\beta = 0.5$）に達する。
- **過密な独立キー（128 keys、256 keys）：**ここでは、delta, β = 1 は additive よりも二乗誤差が小さい（1.06 対 1.99、$n = 128$ のとき）にもかかわらず、最近傍想起率は**低い**（0.53 対 0.96）。なぜだろうか。古い読み出し値を $\mathbf{0}$ に向かって縮小させると、二乗誤差は $\|\mathbf{0} - \mathbf{v}_j\|^2 = 1$ に向かって下がる一方で、初期のペアの信号が後続のペアに比べて浸食されるからである。より穏やかなステップ $\beta = 0.5$ を用いると、より低い二乗誤差（0.67）と大幅に高い想起率（0.82）の両方が回復する。このことから、β_t を入力依存のゲートとし、再束縛されたキーを上書きするときは $\beta_t \approx 1$ を使い、事実を蓄積するときはより小さな β_t を使えるようにする動機が得られる。

12.7.3 Gated DeltaNet

12.7.3.1 減衰と補正

デルタ則は $\mathbf{S}_{t-1}$ を現在のキー $\mathbf{k}_t$ の方向にしか変更しないため、直近のキーに直交する方向は単独では決して減衰しない。**Gated DeltaNet** は (S. Yang et al. 2025)、大域的な忘却を担う入力依存のスカラー減衰 $a_t \in (0, 1)$ と、対象キーの上書きを担うデルタ則のランク 1 方向別更新を組み合わせる。

$$\mathbf{S}_t = a_t \, \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top . \tag{12.37}$$

2 つのゲートは異なる役割を担い、両者の組み合わせによってこの層は新しいペアを 3 通りの方法で扱うことができる（Figure 12.16）。a_t が 0 に近いとき、層はそれまでに記憶した内容をすべ

て一掃して新しいペアだけを残す。これは話題の切り替わりに対する適切な応答である。$a_t \approx 1$ かつ $\beta_t = 1$ のときは、先ほどの再束縛テストと同様に、現在のキーに対応する値を置き換えつつ直交するキーには手を触れない。$a_t \approx 1$ で β_t が小さいときは、読み出し値を新しい値へ向かって途中までしか動かさないため、類似したキーに対する複数回の書き込みでは最後の 1 つが独り勝ちするのではなく平均化される。

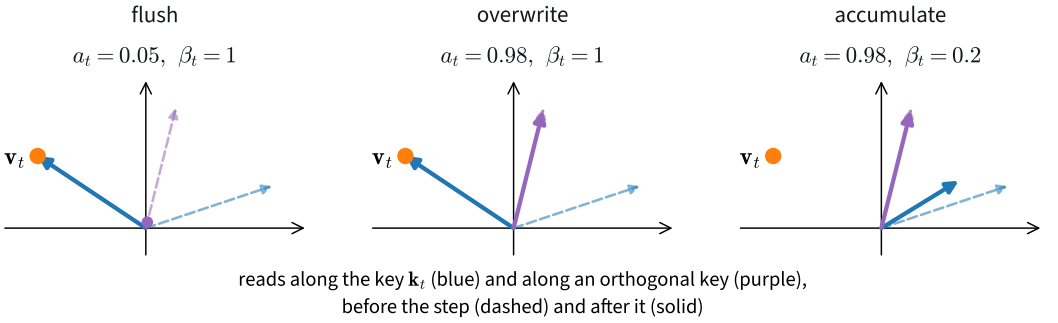


Figure 12.16: ゲートの 3 つの設定のもとでの、2×2 メモリに対する Gated DeltaNet の 1 ステップ。矢印は現在のキー $\mathbf{k}_t$ 方向（青）と直交するキー方向（紫）の読み出し値を表し、ステップ前が破線、ステップ後が実線である。減衰 a_t が 0 に近いとメモリが一掃され、新しいペアだけが残る。$a_t \approx 1$ のとき、書き込み強度 $\beta_t = 1$ では $\mathbf{k}_t$ に対応する値が $\mathbf{v}_t$ に置き換えられて他方のキーはそのまま保たれる一方、小さな β_t では読み出し値が $\mathbf{v}_t$ に向かって途中までしか移動しない。

12.7.3.2 チャンク単位の計算

遷移 $a_t(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)$ はスカラーではなく $d_k \times d_k$ 行列であるため、素朴なトークン単位のスキャンでは $d_k \times d_k$ 行列の乗算が必要となり、$O(T d_k^3)$ のコストがかかる。S. Yang, Wang, Zhang, et al. (2024) は、**WY 表現**（WY representation）を用いて Equation 12.37 をサイズ m のチャンク単位で評価する方法を示した。まず Equation 12.37 を、減衰付きの加算的更新 $\mathbf{S}_t = a_t \mathbf{S}_{t-1} + \mathbf{u}_t \mathbf{k}_t^\top$ として書き換える。ここで、**実効補正値**（effective corrected value）は $\mathbf{u}_t = \beta_t(\mathbf{v}_t - a_t \mathbf{S}_{t-1} \mathbf{k}_t)$ である。長さ m のチャンクの内部で初期状態を $\mathbf{S}_0$ とし、$a_t \mathbf{S}_{t-1} \mathbf{k}_t = \gamma_t \mathbf{S}_0 \mathbf{k}_t + \sum_{s<t} L_{ts} (\mathbf{k}_s^\top \mathbf{k}_t) \mathbf{u}_s$（ただし $L_{ts} = \prod_{r=s+1}^{t} a_r$、$\gamma_t = \prod_{r=1}^{t} a_r$）を展開すると、狭義下三角な $m \times m$ の連立一次方程式がチャンク内の補正値 $\mathbf{u}_1, \dots, \mathbf{u}_m$ について得られる。

$$\mathbf{u}_t + \beta_t \sum_{s<t} L_{ts} \, (\mathbf{k}_s^\top \mathbf{k}_t) \, \mathbf{u}_s = \beta_t (\mathbf{v}_t - \gamma_t \mathbf{S}_0 \mathbf{k}_t).$$

右辺は $\mathbf{S}_0$ に関してアフィンであるため、`delta_rule_writes` は単位下三角な $m \times m$ の系 $\mathbf{M} = \mathbf{I} + \mathrm{diag}(\beta)\,\mathrm{tril}_{-1}(\mathbf{L} \odot \mathbf{K}\mathbf{K}^\top)$ を、2 つの右辺に対してすべてのチャンクとヘッドにわたり並列に解き、$\mathbf{W} = \mathbf{M}^{-1}(\beta \odot \mathbf{V})$ と $\mathbf{Z} = \mathbf{M}^{-1}((\beta \odot \gamma) \odot \mathbf{K})$ を求める。これにより、$\mathbf{U} = \mathbf{W} - \mathbf{Z}\mathbf{S}_0^\top$ は $\mathbf{S}_0$ が確定した時点で直ちに得られる。

```
def delta_rule_writes(K, V, beta, log_a):
    """W and Z with corrected values U = W - Z S_0^T, for every chunk.

    K: (..., m, d_k); V: (..., m, d_v); beta, log_a: (..., m)
    """
    L = d2l.decay_mask(log_a)  # Products of decays within the chunk
```

```
    gamma = jnp.exp(log_a.cumsum(-1))[..., None]  # Decay since start
    beta = beta[..., None]
    M = jnp.eye(K.shape[-2]) + (
        beta * jnp.tril(L, -1) * (K @ K.mT))  # Lower triangular
    W = jax.scipy.linalg.solve_triangular(M, beta * V, lower=True)
    Z = jax.scipy.linalg.solve_triangular(M, beta * gamma * K, lower=True)
    return W, Z
```

$\mathbf{U} = \mathbf{W} - \mathbf{Z}\mathbf{S}_0^\top$ をチャンク末尾の状態 $\mathbf{S}_m = \gamma_m \mathbf{S}_0 + \mathbf{U}^\top \mathbf{D}\mathbf{K}$（ただし $\mathbf{D}$ はチャンク末尾までの減衰を並べた対角行列）に代入すると、チャンク遷移は T/m 個のチャンクにわたるアフィンな行列漸化式として表される。

$$\mathbf{S}_m = \mathbf{S}_0 \mathbf{A} + \mathbf{W}^\top \mathbf{D}\mathbf{K}, \qquad \mathbf{A} = \gamma_m \mathbf{I} - \mathbf{Z}^\top \mathbf{D}\mathbf{K}.$$

$(\mathbf{A}, \mathbf{W}^\top \mathbf{D}\mathbf{K})$ を T/m 個のチャンクにわたってスキャンするコストはわずか $(T/m)d_k^3$ 回の演算で済み（チャンク内の $O(Td_k^2)$ の行列積と $m \approx d_k$ のときに一致する）、その後は $\mathbf{U} = \mathbf{W} - \mathbf{Z}\mathbf{S}_0^\top$ およびすべての出力 $\mathbf{Y}$ が 1 回のバッチ行列積で求まる。

```
def chunked_delta_rule(Q, K, V, beta, log_a, chunk_size):
    """Outputs y_t = S_t q_t of the gated delta rule, computed in chunks.

    Q, K: (batch_size, T, d_k); V: (batch_size, T, d_v);
    beta, log_a: (batch_size, T)
    """
    B, T, m = Q.shape[0], Q.shape[1], chunk_size
    Q, K, V = (X.reshape(B, T // m, m, -1) for X in (Q, K, V))
    beta, log_a = beta.reshape(B, T // m, m), log_a.reshape(B, T // m, m)
    W, Z = delta_rule_writes(K, V, beta, log_a)
    L, cum = d2l.decay_mask(log_a), log_a.cumsum(-1)
    # Each chunk maps the memory S at its start to S A + W^T D K at its end
    to_end = jnp.exp(cum[..., -1:] - cum)[..., None]
    I = jnp.eye(K.shape[-1])
    A = jnp.exp(cum[..., -1, None, None]) * I - (Z * to_end).mT @ K
    def combine(earlier, later):  # A scan over chunks, with matrix products
        (A1, W1), (A2, W2) = earlier, later
        return A1 @ A2, W1 @ A2 + W2
    W_end = jax.lax.associative_scan(combine, (A, (W * to_end).mT @ K),
                                     axis=1)[1]
    S = jnp.concatenate([jnp.zeros_like(W_end[:, :1]), W_end[:, :-1]], 1)
    U = W - Z @ S.mT  # Corrected values, with the memory at each start
    Y = jnp.exp(cum)[..., None] * (Q @ S.mT) + (L * (Q @ K.mT)) @ U
    return Y.reshape(B, T, -1)
```

`chunked_delta_rule` を検証するため、まずトークンごとに逐次計算する参照実装 `delta_rule_recurrent` を用意する。

```
def delta_rule_recurrent(Q, K, V, beta, a):
    """The same outputs, one step at a time."""
    def step(S, inputs):
        q, k, v, b, a_t = inputs
```

```
        S = a_t * S
        S = S + b * jnp.outer(v - S @ k, k)
        return S, S @ q
    S = jnp.zeros((V.shape[-1], K.shape[-1]))
    return jax.lax.scan(step, S, (Q, K, V, beta, a))[1]
```

続いて、チャンクサイズ $m \in \{16, 64\}$ の `chunked_delta_rule` を、単位正規化されたキーを持つ長さ $T = 256$ のランダム系列上で `delta_rule_recurrent` と比較する。

```
T, d_k, d_v = 256, 32, 32
keys = jax.random.split(jax.random.key(0), 5)
Q = jax.random.normal(keys[0], (T, d_k))
V = jax.random.normal(keys[1], (T, d_v))
K = jax.random.normal(keys[2], (T, d_k))
K = K / jnp.linalg.norm(K, axis=-1, keepdims=True)
beta = jax.random.uniform(keys[3], (T,))
a = 0.8 + 0.19 * jax.random.uniform(keys[4], (T,))
with jax.default_matmul_precision('highest'):  # Full float32, not TF32
    Y = delta_rule_recurrent(Q, K, V, beta, a)
    for m in [16, 64]:
        Y_chunked = chunked_delta_rule(Q[None], K[None], V[None], beta[None],
                                       jnp.log(a)[None], m)[0]
        print(f'chunk size {m}: relative deviation '
              f'{jnp.abs(Y_chunked - Y).max() / jnp.abs(Y).max():.1e}')
```

```
chunk size 16: relative deviation 2.6e-07
chunk size 64: relative deviation 3.7e-07
```

チャンク単位のアルゴリズムは、トークン単位の漸化式と相対誤差 $\sim 10^{-7}$ の精度で一致する。

12.7.3.3 層

`GatedDeltaNet` は `d2l.GatedLinearAttention`（12.6 節）のサブクラスであり、書き込み強度の射影 `W_beta`（ヘッドごとに $\beta_{t,h} = \sigma(\mathbf{w}_{\beta,h}^\top \mathbf{x}_t + b_{\beta,h}) \in (0, 1)$）だけを追加し、`memory` をオーバーライドして `chunked_delta_rule` を呼び出す。それ以外のすべての構成要素（幅 4 の因果畳み込み、ℓ_2 正規化されたクエリとキー、ヘッドごとのスカラー減衰、ヘッドごとの RMS 正規化、および SiLU 出力ゲート）は同一である。

```
class GatedDeltaNet(d2l.GatedLinearAttention):
    """Gated linear attention whose memory is written by the delta rule."""
    def __init__(self, num_hiddens, num_heads, chunk_size=32, conv_width=4,
                 timescales=(4, 256), *, rngs):
        super().__init__(num_hiddens, num_heads, chunk_size, conv_width,
                         timescales, rngs=rngs)
        self.W_beta = nnx.Linear(num_hiddens, num_heads, rngs=rngs)

    def memory(self, X, Q, K, V, log_a):
        beta = d2l.heads_to_batch(jax.nn.sigmoid(self.W_beta(X)),
                                  self.num_heads)[..., 0]
        return chunked_delta_rule(Q, K, V, beta, log_a, self.chunk_size)
```

12.7.3.4 再束縛

訓練された GatedDeltaNet 層は、GatedLinearAttention が苦戦する再束縛されたキーの上書きを実行できるだろうか。1 ブロックの SeqLM（$d = 64$、次元 16 のヘッド 4 個、初期減衰時間スケール $[256, 2048]$）を d2l.associative_recall でテストする。ここでは 128 種類から選んだ 16 個のキーと 16 種類から選んだ値を用い、16 個のキーのうち 8 個を最初の束縛からわずか 2 回の書き込み後に新しい値へ再束縛する（num_rebound=8, gap=2）。この間隔を短く保つことで 2 つの束縛の経過時間がほぼ同じになるため、経過時間に基づく大域的減衰では、1 度しか書き込まれていないキーまで忘れることなしに新しい値と古い値を容易には切り分けられない（2 ブロックの加算型モデルは深さ方向を使って 1 回のデルタステップを近似できるため、ここでは単一ブロックを用いる。演習問題 6 を参照）。

```
def rebinding_model(layer, num_hiddens=64, num_heads=4,
                    vocab_size=16 + 128 + 1):
    """One block of `layer`; tokens: 16 values, 128 keys, the filler."""
    rngs = nnx.Rngs(d2l.get_key())  # A new initialization each call
    return d2l.SeqLM(vocab_size, num_hiddens, [layer(
        num_hiddens, num_heads, chunk_size=16, timescales=(256, 2048),
        rngs=rngs)], rngs=rngs)
```

recall_by_kind を用いて再束縛されたキー（rebound）と 1 度だけ書き込まれたキー（~rebound）の正解率を分けて集計し、GatedLinearAttention と GatedDeltaNet をそれぞれ 2 つの乱数シードで 16 ペアの再束縛タスク上で訓練する。

```
def recall_by_kind(model, X, Y, rebound):
    """Recall on the queries for keys written twice and for keys written
    once; a label of -1 matches no prediction."""
    accuracy = lambda mask: float(d2l.query_accuracy(
        model, X, np.where(mask, Y, -1))) / mask.mean()
    return accuracy(rebound), accuracy(~rebound)

task = dict(num_keys=128, num_values=16, num_rebound=8, gap=2)
test = d2l.associative_recall(np.random.default_rng(1), 2048, 16,
                              num_steps=64, **task)
print(f'{"":24s}{"parameters":>12s}{"written twice":>16s}'
      f'{"written once":>16s}')
for name, layer in (('gated linear attention', d2l.GatedLinearAttention),
                    ('Gated DeltaNet', d2l.GatedDeltaNet)):
    models = [rebinding_model(layer) for _ in range(2)]  # Two runs
    for model in models:
        d2l.train_recall(model, pairs=(16,), **task)
    twice, once = zip(*[recall_by_kind(model, *test) for model in models])
    print(f'{name:24s}{d2l.num_params(models[0]):12,d}'
          f'{twice[0]:11.3f} {twice[1]:.3f}{once[0]:11.3f} {once[1]:.3f}')
```

```
                        parameters   written twice    written once
gated linear attention      73,941      0.895 0.910      0.968 0.977
Gated DeltaNet              74,201      0.977 0.990      0.987 0.996
```

260 パラメータの W_beta ヘッドを追加するだけで、再束縛されたキーに対する想起率は約 0.9（gated linear attention）から 0.96 以上（Gated DeltaNet）へと上昇し、1 度だけ書き込まれたキーとの差は約 2 パーセントポイント以内に縮まる。

12.7.3.5 状態追跡

これまでの実験はすべて、何かをメモリに記憶させて後から返すことを求めていた。これとは異なる要求として、トークンごとに変化しうる状態を**追跡**するという課題がある。たとえば、一連の入れ替えの後にどのカップにボールが隠れているか、引用符が開いたままか、あるいは最も単純な例として、これまでに読み込んだ 1 の個数が奇数か偶数かといった問題である。パリティに必要な状態はわずか 1 ビットであり、1 が来れば反転し、0 が来ればそのまま保たれる。何かを長期間保持する必要はないが、1 つでも見落とすとそれ以降の答えがすべて誤りになるため、何ひとつ飛ばすことは許されない。

GRU はプログラムと同じやり方でこの課題を解く。そのゲートは直前の状態 $\mathbf{h}_{t-1}$ を参照できるため、1 ステップで「入力が 1 なら反転する」という計算を実行できる。本章の線形漸化式は、並列訓練と引き換えにこの仕組みを手放した。すなわち、減衰と書き込みが入力のみに依存する（12.3 節）。記憶されたビットを反転させるには、状態のある座標に負の数を掛けなければならないが、minGRU の減衰 $\mathbf{z}_t \in (0,1)$ は座標を縮小できても符号を変えることは決してできない。デルタ則も、書き込み強度がシグモイドである限り同じ制約を受ける。遷移 Equation 12.36 のキー方向の固有値が $1-\beta_t \in (0,1)$ となるからである。しかし $\beta_t \in (0,2)$ とすれば、固有値は -1 に達しうる。入力が 1 のたびに値 $\mathbf{v}$ を $\beta_t = 2$ で 1 つの固定キー $\mathbf{k}$ に書き込み、0 のときには何も書き込まない層は、1 が来るたびに読み出し値を $\mathbf{S}_t\mathbf{k} = 2\mathbf{v} - \mathbf{S}_{t-1}\mathbf{k}$ へと変化させる。空のメモリから始めれば、読み出し値は $2\mathbf{v}$ と $\mathbf{0}$ の間を交互に行き来し、これがそのままパリティとなる。

では、このような構成法が存在するとして、訓練によってそれを見つけられるだろうか。2 ブロック・幅 64 の `d2l.SeqLM`（この課題で Grazzi et al. (2025) が用いた深さ）を 64 ビットのランダムな列で訓練し、各ステップでそこまでのビットのパリティを出力させる。続いて 256 ビットの列でテストすることで、規則そのものを学習したモデルと、訓練時の長さに適合しただけのモデルとを切り分ける。比較するミキサは、GRU、minGRU、`GatedDeltaNet`、および書き込み強度のシグモイドを 2 倍にした点以外は同じ層である `GatedDeltaNetReflect` の 4 つである。上の議論が正しければ、GRU と $\beta_t \in (0,2)$ を持つ層は訓練長を超える位置を含めすべての位置で正解する一方、minGRU とシグモイドのデルタ則は失敗し、訓練時より長い列になるとランダム予測の水準である 0.5 まで低下するはずである。

```
class GatedDeltaNetReflect(d2l.GatedDeltaNet):
    """Gated DeltaNet with write strengths in (0, 2), so that the transition
    can reflect the memory along the key."""
    def memory(self, X, Q, K, V, log_a):
        beta = 2 * d2l.heads_to_batch(jax.nn.sigmoid(self.W_beta(X)),
                                      self.num_heads)[..., 0]
        return chunked_delta_rule(Q, K, V, beta, log_a, self.chunk_size)

def running_parity(rng, batch_size, num_steps):
    """Random bits and, at every step, the parity of the bits so far."""
    X = rng.integers(0, 2, (batch_size, num_steps))
    return X, np.cumsum(X, 1) % 2

def train_parity(model, num_updates=1000):
    """Train on strings of 64 bits; return the accuracy on positions 1-64
    and on positions 65-256 of new strings of 256 bits."""
    rng = np.random.default_rng()
    schedule = lambda t: 1e-2 * d2l.cosine_warmup(t, num_updates,
                                                  num_updates // 10)
    optimizer = nnx.Optimizer(model, optax.chain(
        optax.clip_by_global_norm(1.0),
```

```
            optax.adamw(schedule, weight_decay=0.01)), wrt=nnx.Param)
        X, Y = running_parity(np.random.default_rng(1), 1024, 256)
        with jax.default_matmul_precision('highest'):  # Full float32, not TF32
            d2l.train_queries(model, lambda step: running_parity(rng, 128, 64),
                              num_updates, optimizer)
            return (float(d2l.query_accuracy(model, X[:, :64], Y[:, :64])),
                    float(d2l.query_accuracy(model, X, Y[:, 64:])))

    d = 64  # Width; the delta rules have 4 heads of size 16
    lecun = nnx.initializers.lecun_normal()  # For the recurrent matrix
    parity_mixers = {
        'GRU': lambda rngs: nnx.RNN(nnx.GRUCell(
            d, d, recurrent_kernel_init=lecun, rngs=rngs), unroll=8),
        'minGRU': lambda rngs: d2l.MinGRU(d, d, rngs=rngs),
        'Gated DeltaNet': lambda rngs: d2l.GatedDeltaNet(
            d, 4, chunk_size=32, timescales=(256, 2048), rngs=rngs),
        'Gated DeltaNet, 2σ': lambda rngs: GatedDeltaNetReflect(
            d, 4, chunk_size=32, timescales=(256, 2048), rngs=rngs)}

    def parity_model(mixer):  # Tokens and outputs: the bits 0 and 1
        rngs = nnx.Rngs(d2l.get_key())  # A new initialization each call
        return d2l.SeqLM(2, d, [mixer(rngs), mixer(rngs)],  # Two blocks
                         rngs=rngs)
```

各ミキサについて 2 回ずつ訓練を実行し、訓練中に観測した長さである位置 1〜64 と、位置 65〜256 における正解率を報告する。

```
    print(f'{"":24s}{"positions 1-64":>16s}{"65-256":>12s}')
    for name, mixer in parity_mixers.items():
        (a, b), (c, e) = [train_parity(parity_model(mixer)) for _ in range(2)]
        print(f'{name:24s}{a:11.2f}/{c:.2f}{b:7.2f}/{e:.2f}')
```

```
                         positions 1-64      65-256
GRU                           1.00/1.00   1.00/1.00
minGRU                        0.58/0.55   0.50/0.50
Gated DeltaNet                0.74/0.81   0.50/0.50
Gated DeltaNet, 2σ            1.00/1.00   0.99/1.00
```

GRU は両方の実行において、訓練長でもそれを超える領域でもすべての位置で正解している。一方、minGRU とシグモイドのデルタ則はそうならない。最初の 64 位置ではランダム予測の水準を上回り、minGRU は 0.5 をわずかに超え、Gated DeltaNet は約 0.75 から 0.8 に達するものの、どちらも規則を学習してはいない。位置 64 を超えると、両者ともコイントスと同じ 0.50 の正解率に落ち込む。書き込み強度を 2 倍にすると、この結果が一変する。`GatedDeltaNetReflect` は訓練長で 1.00 に達し、その正解率を 256 ビットまで維持する。これは偶然うまくいった 2 回の実行ではなく典型的な結果である。さらに約 50 回実行したところ、1 回を除くすべてがこのように汎化し、その例外も位置 64 以降で約 0.9 のスコアを記録した。つまり、線形性そのものが障害なのではない。線形漸化式が手放したのは状態に依存する更新であり、それにもかかわらず行列メモリが状態を追跡できるかどうかは、遷移の固有値がとりうる範囲によって決まる。minGRU は負の固有値も、自身の状態を参照するゲートも持たない。2 ブロック目を重ねても欠けている符号は補えず、モデルは失敗する。

もっとも、この鏡映は繊細である。2 倍にしたシグモイドは $\beta_t = 2$ に漸近するだけで到達はしな

いため、反転のたびに記憶されたビットも少しずつ縮小し、書き込み強度が 2 の手前で止まった実行では長い列で正解率が低下する（演習問題 5）。複数のカップの入れ替えを追うような、より難しい状態追跡には、1 トークンあたり複数の鏡映が必要となる。

12.7.4 回帰としてのメモリ

一歩引いて眺めると、本章のすべての行列メモリ、さらにはソフトマックスアテンションも、系列を読み進めながらキー $\mathbf{K}_t \in \mathbb{R}^{t\times d_k}$ から値 $\mathbf{V}_t \in \mathbb{R}^{t\times d_v}$ への回帰予測器をその場で当てはめる操作として理解できる (K. A. Wang et al. 2025)。$\sum_{s\le t}\|\mathbf{S}\mathbf{k}_s-\mathbf{v}_s\|^2$ の厳密な線形最小二乗法による最小点は次の式で与えられる。

$$\mathbf{S}_t^{\star}=\mathbf{V}_t^{\top}\mathbf{K}_t\,(\mathbf{K}_t^{\top}\mathbf{K}_t)^{-1}, \tag{12.38}$$

ここで、$(\mathbf{K}_t^{\top}\mathbf{K}_t)^{-1}$ は $t<d_k$ のとき擬似逆行列に置き換えられる。Equation 12.38 の 2 つの因子に注目しよう（Table 12.2）。

- **線形アテンション**は最初の因子である非正規化交差モーメント $\mathbf{V}_t^{\top}\mathbf{K}_t=\sum_{s\le t}\mathbf{v}_s\mathbf{k}_s^{\top}$ のみを保持し、重なり合うキーを無相関化するグラム逆行列 $(\mathbf{K}_t^{\top}\mathbf{K}_t)^{-1}$ を省略する。
- **デルタ則**はトークンごとに Equation 12.38 に向かってオンラインの確率的勾配ステップを 1 回進めるのに対し、**Gated DeltaNet** は各ステップの前に重み減衰（weight decay、$a_t<1$）を加える。陽的な勾配ステップを近接陰的ステップ（proximal implicit step）$\arg\min_{\mathbf{S}}\frac{1}{2}\|\mathbf{S}-\mathbf{S}_{t-1}\|^2+\frac{\beta_t}{2}\|\mathbf{S}\mathbf{k}_t-\mathbf{v}_t\|^2$ に置き換えると、正規化されたステップサイズ $\beta_t/(1+\beta_t\|\mathbf{k}_t\|^2)$ を持つ Longhorn が得られ (B. Liu et al. 2024)（演習問題 3）、一方で $(\mathbf{K}_t^{\top}\mathbf{K}_t+\lambda\mathbf{I})^{-1}$ を陽に保持すれば逐次最小二乗法（recursive least squares）が得られる（演習問題 4）。
- **ソフトマックスアテンション**は、保存されたすべてのペアに対してクエリ時に**ノンパラメトリック**な Nadaraya–Watson カーネル回帰（10.1.2 節）を実行し、$O(t)$ の KV キャッシュというコストと引き換えにパラメトリックな圧縮を回避する。

厳密な当てはめも、1 ペアずつ最新の状態に保ち続けることができる。各ペアは $\mathbf{K}_t^{\top}\mathbf{K}_t$ をランク 1 の項 $\mathbf{k}_t\mathbf{k}_t^{\top}$ だけ変化させるため、Sherman–Morrison の公式（33.2.3.5 節）によって新たな逆行列計算を行うことなく逆行列を更新できる。これが逐次最小二乗法である。この手法は、メモリの隣にサイズ $d_k\times d_k$ の第 2 の状態を保持する（演習問題 4）。

Table 12.2: これまでに読み込んだペアに対する回帰としてのメモリ。

メモリ	当てはめる対象	方法	状態
ソフトマックスアテンション	すべての値のカーネル重み付き平均	クエリ時	全ペア
線形アテンション	$\mathbf{V}^{\top}\mathbf{K}$（キー補正なしの最小二乗法）	1 ペアにつき 1 回の加算	$d_v\times d_k$
ゲート付き線形アテンション	同上（直近のペアに重み付け）	1 ペアにつき 1 回の加算	$d_v\times d_k$
デルタ則	最小二乗法の近似	1 ペアにつき 1 回の勾配ステップ	$d_v\times d_k$
Gated DeltaNet	重み減衰付き最小二乗法の近似	1 ペアにつき 1 回の勾配ステップ	$d_v\times d_k$

これらすべてのアーキテクチャにおいて、学習は 2 つの時間スケールで進行する。状態 $\mathbf{S}_t$ の**高速重み**（fast weights）が順伝播中に現在のコンテキストにおけるキー・値の対応関係を当てはめる

一方、ネットワークの**低速重み**（slow weights、$\mathbf{W}_{qkv}, \mathbf{W}_a, \mathbf{W}_\beta, \ldots$）は、そのオンライン当てはめが有効に働くようなキー、値、クエリ、ステップサイズを生成するよう、系列をまたいで誤差逆伝播法により訓練される。しかし、どれほど最適な書き込み規則を用いても、$d_v \times d_k$ 行列のランクとビット容量は有限である。12.8 節ではその厳然たる限界を測定し、ごく一部のフルアテンション層を挟み込むことでその限界をどのように引き上げられるかを示す。

12.7.5 まとめ

加算型行列メモリは同じキーへのすべての書き込みを重ね合わせるため、再束縛されたキーをクエリすると古い値と新しい値の和が返される。デルタ則は書き込み前に現在の予測 $\mathbf{S}_{t-1}\mathbf{k}_t$ を読み出し、誤差補正 $\beta_t(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t)\mathbf{k}_t^\top$ のみを加える。これは $\frac{1}{2}\|\mathbf{S}\mathbf{k}_t - \mathbf{v}_t\|^2$ に対する勾配降下法の 1 回のオンラインステップであり、その遷移 $\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top$ はメモリを $\mathbf{k}_t$ 方向に選択的に縮小する。対象を絞った消去はキーが再束縛される場合や相関を持つ場合に加算的書き込みを上回る一方、多数のランダムなキーが重なり合う状況で強い消去（$\beta_t = 1$）を行うと初期の信号が浸食されうる。`GatedDeltaNet` は入力依存の書き込み強度 β_t とヘッドごとのスカラー減衰 a_t を組み合わせ、下三角求解（`delta_rule_writes`）とチャンク単位のスキャン（`chunked_delta_rule`）によって漸化式を並列チャンクで評価し、連想想起において再束縛されたキーを確実に復元する。

12.7.6 演習問題

1. **順序は結果に影響しない。**一連の書き込みを行った後の加算型メモリが、書き込みの順序に依存しないことを示せ。ここから、いかに巧妙な読み出し規則であっても、加算型メモリだけからは 1 つのキーに書き込まれた R 個の値のうち最新のものを復元できないことを結論づけ、さらに R 個の値がすべて異なるときに最良の規則が達成できる想起率を求めよ。コードブックのサイズを 32 個から 1,024 個の値へ増やしたとき、8 回の書き込み後に測定される想起率はどのように変化するか。また、それはなぜか。

2. [code] **補正の代わりに減衰を用いる。** `overwrite_recall` の加算的書き込みに定数減衰を与え、$\mathbf{S} \leftarrow a\mathbf{S} + \mathbf{v}\mathbf{k}^\top$ とせよ。

    1. $R = 2, 4, 8$ に対して、最新の値が少なくとも 95% の割合で想起される最大の減衰 a を求めよ。なぜその答えは同じキーに対する 2 回の書き込みの間に何回の書き込みが挟まるかに依存するのか。また、このテストにおいてより小さな減衰が決して悪影響を与えないのはなぜか。
    2. 次に、正規直交キーを幅 64 における 16 個のランダムな単位キーに置き換えよ。そのうち 8 個をこれまで通り R 回書き直し、残りの 8 個は最初に 1 度だけ書き込んで、最後に両方の種類のキーに対する想起率を測定せよ。どちらの種類のキーにも対応できる書き込み規則はどれか。

3. **陰的ステップ。** $\frac{1}{2}\|\mathbf{S} - \mathbf{S}_{t-1}\|^2 + \frac{\beta}{2}\|\mathbf{S}\mathbf{k} - \mathbf{v}\|^2$ の最小点が $\mathbf{S}_{t-1} - \frac{\beta}{1+\beta\|\mathbf{k}\|^2}(\mathbf{S}_{t-1}\mathbf{k} - \mathbf{v})\mathbf{k}^\top$ であることを示せ（ヒント：勾配を 0 と置き、その結果に $\mathbf{k}$ を掛けてまず $\mathbf{S}\mathbf{k}$ を求めるとよい）。さらにその遷移の固有値を求め、このステップが新しいペアを厳密に想起するのは極限 $\beta \to \infty$ の場合に限られることを示せ。

4. [code] **逐次最小二乗法。** Equation 12.38 に $\lambda \to 0$ で漸近する正則化最小二乗法の当てはめは、メモリが $\mathbf{P}_t = (\mathbf{K}_t^\top\mathbf{K}_t + \lambda\mathbf{I})^{-1}$ も併せて保持すれば 1 ペアずつ更新できる。

    1. 33.2.3.5 節の Sherman–Morrison の公式を用いて、新しいペアが 1 つ与えられた後の $\mathbf{P}_t$ と $\mathbf{S}_t$ の更新式を導出し、$\mathbf{S}_t$ の更新が $\mathbf{P}_t\mathbf{k}_t$ 方向に（$\mathbf{k}_t$ ではなく）書き込むデルタ則であることを示せ。デルタ則の $d_v d_k$ 個に対し、この状態はいくつの数値を保持するか。

2. これを実装し、上書きテストにおいてデルタ則と比較せよ。すべてのペアに対する最良の当てはめである逐次最小二乗法が、最新の値をうまく想起できないのはなぜか。また、どうすればこれを解決できるか。

5. [code] **パリティ**。●GatedDeltaNet の書き込み強度はシグモイドであるため、キー方向の遷移の固有値 $1-\beta_t$ は $(0,1)$ に収まる。$2\sigma(\cdot)$ を $\sigma(\cdot)$ の代わりに用いると、固有値は $(-1,1)$ に収まるようになる。

   1. それでもなお、訓練された層は $\beta_t = 2$ に漸近することしかできない。12.7.3.5 節のパリティの構成法において、カウント対象のすべてのトークンで $\beta = 2-\epsilon$ と置き、カウント対象のトークンが n 個続いた後の読み出し値が $\big(1-(\epsilon-1)^n\big)\mathbf{v}$ となることを示せ。奇数回カウントした後と偶数回カウントした後の読み出し値の差は、どれほどの速さで縮小するか。$\epsilon = 0.01$ のとき、この差が $\|\mathbf{v}\|$ の 1% 未満になるのはカウント対象のトークンが何個続いた後か。また、それは有限精度での読み出しにとって何を意味するか。
   2. 12.7.3.5 節では訓練長の 4 倍でモデルをテストした。Gated DeltaNet, 2σを改めて訓練し、1,024 ビットおよび 4,096 ビットの列でテストせよ。正解率が低下し始めるのはどの長さか。また、その長さは最初の小問の答えと整合しているか。

6. [code] **再束縛への別の経路**。再束縛の実験では、書き込み規則だけが値を補正できるように 3 つの設定を固定していた。これらを 1 つずつ緩和し、それぞれの結果を説明せよ。

   1. ゲート付き線形アテンションのメソッド decays（$\log a_t$ を返す）をオーバーライドして同じ形状のゼロを返すようにし、すべての $a_t = 1$ として減衰を無効化したうえで再度訓練せよ。書き直されたキーと 1 度だけ書き込まれたキーに対する想起率はそれぞれどのように変化するか。
   2. 減衰を元に戻し、各書き直しが後方の任意の位置で起こるようにせよ（gap=None を task に指定する）。2 つの層の間の差は縮まるか。2 つの値の経過時間を使って説明せよ。
   3. 両方の層を 1 ブロックではなく 2 ブロックで訓練せよ。2 つの加算層を組み合わせることで、どのようにして値を補正できるか。

12.8 ハイブリッドアーキテクチャ

本章で扱ってきたすべてのモデルは、ゲートから選択的ダイナミクス、さらには誤差修正型のデルタ則に至るまで、固定サイズの状態を**どのように**更新するかを改良してきた。しかし、どのモデルも状態のサイズが許す以上の情報を保持することはできない。ユーザーが 64 個の商品コードと価格の表をプロンプトに貼り付け、そのうち 20 個について尋ねたとき、Transformer は完全な KV キャッシュ全体を振り返って正確なトークンを取り出すのに対し、純粋なリカレントモデルには、どの項目が質問されるかを知る前に固定状態へ圧縮した情報しか残っていない。少数ショットプロンプティング（few-shot prompting）、検索拡張生成（retrieval-augmented generation）、コード編集などで必要となるコンテキスト内検索（in-context retrieval）こそが、固定状態の系列モデルが明確な上限に突き当たる場面である。

実運用されているリカレント言語モデルが採用する解決策は実務的なものであり、リカレントブロックの間にごく少数の完全なアテンション層を挟み込むことである。以下では、固定状態による想起に対する 2 つの基本的な限界（情報理論的なビット数え上げ限界と、キーの重なりに関する Welch の幾何学的下界）を導出し、純粋な GatedDeltaNet、純粋なアテンション、および GA ハイブリッドをマルチクエリ連想想起（multi-query associative recall）とバイトレベルの WikiText-103 上で比較する。さらに、隣接するキーと値のトークンを対にする短いデプスワイズ畳み込みをアブレーションにより検証し、256K コンテキスト長における推論時の KV キャッシュ削減量を定量化し、最後に本章で扱ったすべての系列メモリを総括する。

12.8.1 固定サイズの状態の限界

12.8.1.1 ビットの数え上げ

鳩の巣原理（pigeonhole principle）により、M 個の数値をそれぞれ b ビットで格納する状態は、高々 2^{Mb} 通りの異なる構成しかとり得ない。可能な全 V^k 通りの系列（長さ k トークン、語彙サイズ V）をすべて区別して再現するには、状態は次の条件を満たさなければならない (Jelassi et al. 2024)。

$$Mb \geq k \log_2 V. \tag{12.39}$$

したがって、固定状態が想起できる最大トークン数は状態サイズ M に比例して増加し、コンテキスト長 T にはまったく依存しない。これに対し、Transformer の KV キャッシュは T に比例して拡大し、飽和することがない。

実際には、線形行列メモリは生のビット数を使い切るはるか手前で幾何学的な限界に突き当たる。以下で用いる連想想起ベンチマークでは、1 つの系列に最大 64 個のペア（キーは 128 種類の記号、値は 4 種類の記号から選ばれ、テーブル情報量は高々 $128 \times 2 = 256$ ビット、64 個の値に限れば $64 \times 2 = 128$ ビット）を格納する一方、`GatedDeltaNet` 2 層（8 ヘッド、各ヘッドは 4×4 の float32 状態）は $256 \times 32 = 8{,}192$ ビット、すなわち必要量の $30\times$ を超えるビット数を保持する。なぜ行列メモリは、生のビット数を大きく下回る段階でペアを取りこぼすのだろうか。

12.8.1.2 キーの重なり

行列ヘッド $\mathbf{S} \in \mathbb{R}^{d_v \times d_k}$ を加法的に更新する場合（12.6 節）でも、最小二乗解へ向けてデルタ則で更新する場合（12.7 節）でも、任意の値に対して厳密な線形読み出し $\mathbf{S}\mathbf{k}_j = \mathbf{v}_j$ を実現するには、格納されたキーが線形独立でなければならない。ところが、$\mathbb{R}^{d_k}$ には線形独立なベクトルが高々 d_k 本しか存在しない。格納するキーの数 m がヘッド次元 d_k を超えると、訓練によってキーをどのように選んでも、キー同士は**必ず**重なり合うことになる。

命題（Welch の下界）。$\mathbf{k}_1, \dots, \mathbf{k}_m$ を $\mathbb{R}^{d_k}$ 内の単位ベクトルとし、$m > d_k$ とする。このとき、相異なるすべてのペア $i \neq j$ にわたる二乗内積の平均は、少なくとも次の値以上である (Welch 1974)。

$$\frac{m - d_k}{d_k\,(m-1)}. \tag{12.40}$$

証明。$\mathbf{G} = \sum_{i=1}^{m} \mathbf{k}_i \mathbf{k}_i^{\top} \in \mathbb{R}^{d_k \times d_k}$ とおく。そのトレースは $\sum_{i=1}^{m} \|\mathbf{k}_i\|^2 = m$ であるため、d_k 個の固有値の和は m となる。コーシー・シュワルツの不等式により、固有値の二乗和（$\|\mathbf{G}\|_F^2 = \sum_{i,j} (\mathbf{k}_i^{\top}\mathbf{k}_j)^2$ に等しい）は少なくとも m^2/d_k である。ここから m 個の対角項 $(\mathbf{k}_i^{\top}\mathbf{k}_i)^2 = 1$ を差し引くと、少なくとも $m^2/d_k - m = m(m-d_k)/d_k$ が $m(m-1)$ 個の非対角ペア全体の和として残る。■

次元 $d_k = 4$ のヘッドでは、$m = 64$ 個の単位キーの平均二乗重なりは少なくとも $60/(4\times63) \approx 0.24$ となり、ランダムな単位ベクトル（$1/d_k = 0.25$）とほぼ見分けがつかない。ヘッドを $d_k = 16$ に広げると、Welch の下限は $48/(16 \times 63) \approx 0.05$ まで下がる。

ここから、マルチクエリ連想想起において 2 つの明確な実験的特徴が予測される。

1. 純粋な `GatedDeltaNet` の積層モデルは、格納ペア数 n が状態容量を超えると性能が劣化する一方、層あたりの状態要素数を 4 倍（$8 \times 4^2 = 128$ から $2 \times 16^2 = 512$ へ、幅 $d = 32$ は固定）に増やすと、劣化が始まる位置が約 $4\times$ 高いペア負荷へとシフトする。

2. 2 つのリカレントブロックのうちわずか 1 つを完全な因果アテンションに置き換える（GA）だけで、容量不足による性能低下は完全に解消される。アテンション層がすべてのトークンをキャッシュに保持するためである。

12.8.2 1 つのアテンション層による想起

12.8.2.1 4 つのモデル

2 ブロック構成の d2l.SeqLM モデル（$d = 32$）4 種類を、マルチクエリ連想想起（num_keys=128、num_values=4、負荷 $n \in \{4, 8, 16, 32, 64\}$）で検証する。

- **Gated DeltaNet（GG）**：8 個のサイズ 4 ヘッド（層あたり 128 個の状態要素）。
- **Gated DeltaNet, 4x state（GG, num_heads=2）**：2 個のサイズ 16 ヘッド（層あたり 512 個の状態要素、パラメータ数は同じ）。
- **attention（AA）**：2 ヘッドの因果アテンション（d2l.AttentionMixer）。$(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ に対して同じ幅 4 のデプスワイズ因果畳み込みを適用し、位置エンコーディングは用いない（rope=False）。
- **hybrid（GA）**：ブロック 1 の GatedDeltaNet に続けて、ブロック 2 に AttentionMixer を配置した構成。

```
jax.config.update('jax_default_matmul_precision', 'float32')  # Not TF32

def recall_model(layout, num_heads=8, conv_width=4, num_hiddens=32):
    """SeqLM whose mixers are given by letters: G(ated DeltaNet) with
    num_heads heads, A(ttention) with two; tokens: 4 values, 128 keys, and
    the filler."""
    rngs = nnx.Rngs(d2l.get_key())  # A new initialization each call
    make = {'G': lambda: d2l.GatedDeltaNet(
                num_hiddens, num_heads, conv_width=conv_width,
                timescales=(256, 2048), rngs=rngs),
            'A': lambda: d2l.AttentionMixer(num_hiddens, num_heads=2,
                                            conv_width=4, rope=False,
                                            rngs=rngs)}
    return d2l.SeqLM(4 + 128 + 1, num_hiddens, [make[c]() for c in layout],
                     rngs=rngs)
```

12.8.2.2 想起スイープ

各アーキテクチャにつき 2 つのシードを d2l.train_recall で 2,000 回更新して訓練し、ペア負荷ごとの正解率を評価する。

```
arms = {'Gated DeltaNet': dict(layout='GG'), 'attention': dict(layout='AA'),
        'hybrid': dict(layout='GA'),
        'Gated DeltaNet, 4x state': dict(layout='GG', num_heads=2)}
recall = {}
for name, arm in arms.items():
    models = [recall_model(**arm) for _ in range(2)]
    recall[name] = [d2l.train_recall(model, num_keys=128) for model in models]
    print(f'{name:25s} {d2l.num_params(models[0]):,} parameters')
```

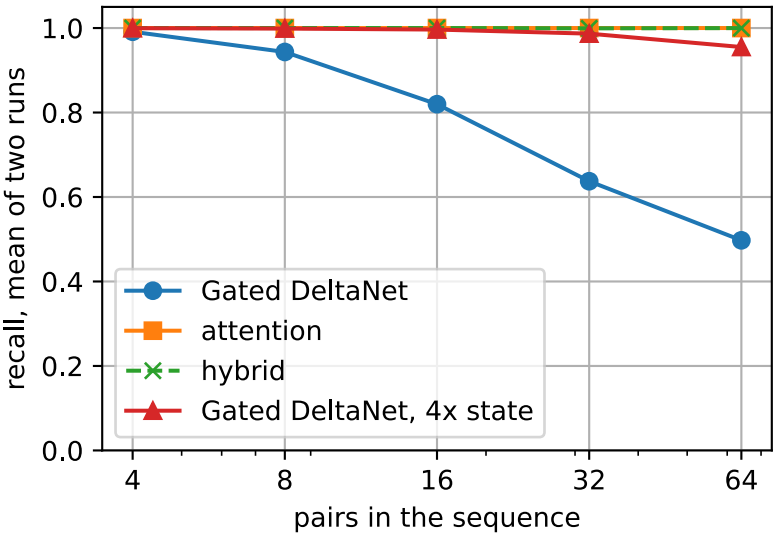
```
Gated DeltaNet            37,989 parameters
attention                 34,821 parameters
```

```
hybrid                   36,405 parameters
Gated DeltaNet, 4x state  37,197 parameters
```

続いて、各アームの 2 回の実行にわたって想起曲線を平均し、格納ペア数 $n \in \{4, 8, 16, 32, 64\}$ の関数として正解率をプロットする。

```
pairs = list(recall['hybrid'][0])
runs = {name: np.array([[run[n] for n in pairs] for run in two])
        for name, two in recall.items()}  # Two rows, one per run
d2l.plot(pairs, [r.mean(0) for r in runs.values()], 'pairs in the sequence',
         'recall, mean of two runs', legend=list(runs), xscale='log',
         ylim=(0, 1.05), fmts=('-o', '-s', '--x', '-^'), figsize=(4.5, 3),
         legend_loc='lower left')
d2l.plt.xticks(pairs, pairs)  # Ticks at the numbers of pairs
print(f'{"pairs":25s}', ''.join(f'{n:>11d}' for n in pairs))
for name, r in runs.items():
    print(f'{name:25s}', ''.join(f'{a:6.2f}/{b:.2f}' for a, b in r.T))
```

```
pairs                             4          8         16         32         64
Gated DeltaNet             0.99/0.99  0.94/0.95  0.80/0.84  0.62/0.66  0.50/0.50
attention                  1.00/1.00  1.00/1.00  1.00/1.00  1.00/1.00  1.00/1.00
hybrid                     1.00/1.00  1.00/1.00  1.00/1.00  1.00/1.00  1.00/1.00
Gated DeltaNet, 4x state   1.00/1.00  1.00/1.00  1.00/0.99  1.00/0.98  0.98/0.93
```



2 つの予測はいずれも成り立っている。

1. **GatedDeltaNet における状態容量のスケーリング：**層あたり 128 個の状態要素（GG）では、想起率は 4 ペアでの約 1.00 から、32 ペアでは 0.6〜0.7、64 ペアでは約 0.5 まで低下する。状態を層あたり 512 要素へ 4 倍に増やす（Gated DeltaNet, 4x state）と、32 ペアまで想起率は 1.00 付近を維持する。64 ペアでも大半のペアを想起でき、実行によって約 0.8〜1.0 の範囲となる。
2. **1 つのアテンション層が厳密な想起を回復する：**attention（AA）と hybrid（GA）はいずれも、すべての実行において最大 64 ペアまでのあらゆる負荷で 1.00 の想起率を達成しており、Waleffe et al. (2024) による大規模実験の知見と一致する。

12.8.2.3 短い畳み込み

なぜリカレントミキサやアテンションミキサの前に幅 4 のデプスワイズ因果畳み込みを入れたのだろうか。連想想起では、各値はトークン t において、トークン $t-1$ のキーの直**後**に到着する。局所的なトークン混合がない場合（conv_width=1）、ステップ t の単一の行列状態層は値トークンしか見ることができず、直前のキートークンと結び付けることができない。GG で conv_width=1 と設定し、この畳み込みのアブレーションを試してみよう。

```
no_conv = [d2l.train_recall(recall_model('GG', conv_width=1), num_keys=128)
           for _ in range(2)]  # Width 1: only scales and shifts each channel
print(f'{"pairs":25s}', ''.join(f'{n:>11d}' for n in pairs))
for name, (a, b) in [('Gated DeltaNet', recall['Gated DeltaNet']),
                     ('without the convolution', no_conv)]:
    print(f'{name:25s}', ''.join(f'{a[n]:6.2f}/{b[n]:.2f}' for n in pairs))
```

```
pairs                             4          8         16         32         64
Gated DeltaNet              0.99/0.99  0.94/0.95  0.80/0.84  0.62/0.66  0.50/0.50
without the convolution     0.82/0.88  0.67/0.74  0.53/0.59  0.43/0.49  0.37/0.41
```

幅 4 の因果畳み込みを取り除くと、すべての負荷で想起率が低下する（約 10〜35 パーセントポイント低下し、4 ペアの想起率でさえ 0.8〜0.9 まで落ち込む）。

12.8.3 同規模での言語モデリング

純粋な GatedDeltaNet（GGGG）、3 対 1 のハイブリッド（GGGA）、および純粋な因果 attention（AAAA）は、バイトレベルの WikiText-103 ベンチマーク（d2l.SeqLM、幅 $d = 256$ の 4 ブロック構成）においてどのように比較されるだろうか。この問いを 2 段階で検証する。最初は本章を通じて用いてきた 128 バイトのウィンドウで、次はより長いコンテキストによって固定状態と拡大するキャッシュに差がつくかを確かめるため 1,024 バイトのウィンドウで比較する。モデルが離れた位置のコンテキストを活用できるようになるのは訓練の後半であるため、いずれの実験でも前節までの 1,000 ステップではなく 8,000 ステップ訓練する。したがって、以下の損失はこれまでのテキスト実験の表よりも低く、それらと直接比較することはできない。

12.8.3.1 短いウィンドウ

まず、128 バイトの WikiText103 データセットを読み込む。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
```

公平な比較とするには、もう 1 つのアームが必要である。Mamba、ゲート付き線形アテンション、および Gated DeltaNet はいずれも再帰の前に幅 4 の因果畳み込みで隣接トークンを混合しているが、これまでのテキストベンチマークで用いたアテンションミキサにはそれがなかった。そのため、GatedDeltaNet が attention を上回ったとしても、その差はメモリではなく畳み込みに由来する可能性がある。そこで想起実験と同様に、4 つ目のアームとしてアテンションにも同じ畳み込みを与える。

そのために、text_model(layout, d) は 4 ブロックの d2l.SeqLM を構築する。各ブロックのミキサは 4 文字の文字列 layout に従い、'G'（GatedDeltaNet、4 ヘッド）、'A'（AttentionMixer、8 ヘッド）、'C'（幅 4 の因果畳み込みの後ろに配置した同じアテンション）、および 'R'（後述する長いウィンドウ用の GRU）に対応する。

```
jax.config.update('jax_default_matmul_precision',
                  'tensorfloat32')  # fp32 matmuls may use tf32
text_mixers = {
    'R': lambda d, rngs: nnx.RNN(nnx.GRUCell(
        d, d, recurrent_kernel_init=nnx.initializers.lecun_normal(),
        rngs=rngs), unroll=8),
    'G': lambda d, rngs: d2l.GatedDeltaNet(d, num_heads=4, rngs=rngs),
    'A': lambda d, rngs: d2l.AttentionMixer(d, num_heads=8, rngs=rngs),
    'C': lambda d, rngs: d2l.AttentionMixer(d, num_heads=8, conv_width=4,
                                            rngs=rngs)}

def text_model(layout, d):  # 256 byte values; one block per letter
    rngs = nnx.Rngs(d2l.get_key())
    return d2l.SeqLM(256, d, [text_mixers[c](d, rngs) for c in layout],
                     rngs=rngs)
```

'GGGG'、'GGGA'、'AAAA'、'CCCC' のそれぞれについて、幅 $d = 256$ で 8,000 ステップの訓練を d2l.text_table により 2 回ずつ実行する。

```
arms = {'Gated DeltaNet': ('GGGG', 256), 'hybrid': ('GGGA', 256),
        'attention': ('AAAA', 256),
        'attention, conv': ('CCCC', 256)}  # Width 256, four blocks each
trained = d2l.text_table(text_model, arms, data, num_steps=8000)
```

```
                          validation        0-15       16-63      64-127   ms/step
Gated DeltaNet    3,573,536 1.156/1.160 1.401/1.397 1.126/1.131 1.118/1.123 17/18
hybrid            3,501,848 1.151/1.155 1.399/1.398 1.122/1.125 1.112/1.117 15/15
attention         3,286,784 1.185/1.181 1.411/1.410 1.157/1.151 1.150/1.145 9/9
attention, conv   3,302,144 1.154/1.148 1.397/1.394 1.124/1.119 1.115/1.108 10/10
```

128 バイトのウィンドウでは、4 つのアームのうち 3 つがほぼ同等の結果に落ち着く。Gated DeltaNet はバイトあたり約 1.16 nat、GGGA ハイブリッドと畳み込み付きアテンションは約 1.15 である。畳み込みなしの attention だけが約 0.03 遅れて 1.18 と離れている。この長さでは損失は局所的な構文に支配されており、ウィンドウの先頭 16 バイト以降はほとんど変化しない（12.1.4.2 節）ため、この表は主にミキサが隣接バイトをどれだけうまく結合できるかを測っている。

したがって、素のアテンションの遅れはすべて畳み込みの有無で説明がつく。畳み込みを加えれば、アテンションはハイブリッドと並ぶ。どちらも Gated DeltaNet を高々 0.01 上回るにすぎず、これは同一アームの 2 回の実行間に生じる差と同程度であるため、そこにランキングは読み取らない。短いウィンドウでは、行列メモリにもキャッシュにも局所的な混合では説明できないほどの優位性はなく、ハイブリッドが持つ 1 つのアテンション層も測定可能な効果をもたらさない。この層が重要になるのは、厳密な長距離参照が必要とされる場面である（演習問題 5）。では、ウィンドウが十分に長くなれば、実際のテキストでもこの層は意味を持つのだろうか。

12.8.3.2 長いウィンドウ

128 バイトのウィンドウでは、拡大するキャッシュに対して固定状態が何を失うかも、行列メモリに対してベクトル状態が何を失うかも見えてこない。そこで、1,024 バイトのウィンドウ上で 4 種のミキサ、すなわち GRU の積層（'RRRR'）、Gated DeltaNet、GGGA ハイブリッド、および畳み込み付きアテンションを訓練する。バッチサイズを 64 から 8 に下げることで、各ステップが処理するバイト数と、8,000 ステップ全体で目にするテキスト量を同じに保つ。

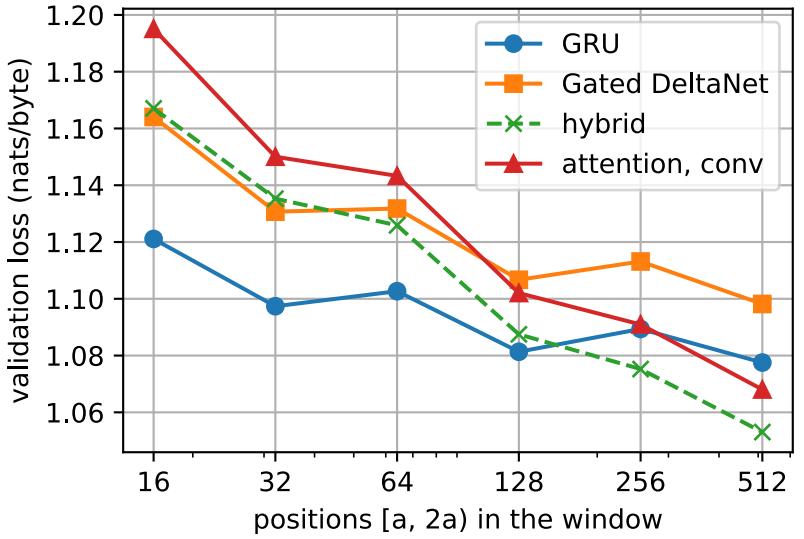
```
long_data = d2l.WikiText103(batch_size=8, num_steps=1024)  # Same bytes/step
long_arms = {'GRU': ('RRRR', 256), 'Gated DeltaNet': ('GGGG', 256),
             'hybrid': ('GGGA', 256), 'attention, conv': ('CCCC', 256)}
long_trained = d2l.text_table(text_model, long_arms, long_data,
                              num_steps=8000)  # Long context needs training
```

		validation	0-127	128-511	512-1023	ms/step
GRU	3,814,144	1.088/1.091	1.145/1.147	1.086/1.087	1.076/1.079	62/62
Gated DeltaNet	3,573,536	1.115/1.112	1.180/1.180	1.113/1.109	1.100/1.096	17/17
hybrid	3,501,848	1.074/1.084	1.180/1.184	1.075/1.084	1.047/1.059	15/15
attention, conv	3,302,144	1.123/1.066	1.207/1.192	1.118/1.071	1.105/1.031	23/23

ウィンドウ全体の平均をとると、ウィンドウ内のどの位置で各モデルが改善しているかが隠れてしまう。コンテキスト全体を活用するモデルは位置が進むにつれて予測精度を高め続けるのに対し、直近のバイトしか要約しないモデルは早い段階で改善が止まる。両者を見分けるため、長さが倍々に広がる位置の帯域 $[a, 2a)$（バイト位置 16〜31 から 512〜1,023 まで）ごとに、2 回の実行の検証損失を平均する。

```
losses = {name: np.mean([d2l.loss_by_position(model, long_data)
                         for model in models], 0)  # Mean of the two runs
          for name, models in long_trained.items()}
starts = [16, 32, 64, 128, 256, 512]  # Bands [a, 2a) of positions
bands = {name: [loss[a:2 * a].mean() for a in starts]
         for name, loss in losses.items()}
d2l.plot(starts, list(bands.values()), 'positions [a, 2a) in the window',
         'validation loss (nats/byte)', legend=list(bands), xscale='log',
         fmts=('-o', '-s', '--x', '-^'), figsize=(4.5, 3))
d2l.plt.xticks(starts, starts)  # Ticks at the first position of each band
print(f'{"positions":16s}', ''.join(f'{f"{a}-{2 * a - 1}":>10s}'
                                    for a in starts))
for name, band in bands.items():
    print(f'{name:16s}', ''.join(f'{v:10.3f}' for v in band))
```

positions	16-31	32-63	64-127	128-255	256-511	512-1023
GRU	1.121	1.097	1.103	1.081	1.089	1.078
Gated DeltaNet	1.164	1.131	1.132	1.107	1.113	1.098
hybrid	1.167	1.135	1.126	1.087	1.075	1.053
attention, conv	1.195	1.150	1.143	1.102	1.091	1.068



ウィンドウを長くすると、両方の長さで訓練したすべてのモデルで性能が向上する。Gated DeltaNet は 1.11〜1.12、ハイブリッドは 1.07〜1.08 に到達し、いずれも 128 バイトのウィンドウでの 1.16 と 1.15 から改善している。畳み込み付きアテンションは、大半の実行で約 1.07 に達する。ただし約 5 回に 1 回の実行では 1.12 付近にとどまるが、その理由は後述する。GRU は 1.09 に達するものの、ウィンドウを 1 ステップずつ逐次処理するのはこのモデルだけであるため、ステップあたりの計算時間が次に遅いモデルの約 2 倍以上かかり、大差で最も遅いアームとなっている。

短いウィンドウでは横並びだった 3 つのアームにも差が生じており、表の各列からどこで差が開くかがわかる。ウィンドウの最初の 128 バイトから最後の 512 バイトにかけて、GRU の損失は約 0.07、Gated DeltaNet の損失は約 0.08 下がる。これに対し、ハイブリッドは約 0.13、アテンションは大半の実行で約 0.16 改善する。プロットを見ると、改善が起きる位置がさらに詳しくわかる。バイト位置 127 までは固定状態モデルも互角以上に戦っており、すべての帯域で GRU が最も低く、序盤の帯域ではアテンションが最も高く、ハイブリッドは純粋な Gated DeltaNet と見分けがつかない。しかしバイト位置 128 を超えると、GRU と Gated DeltaNet は最後の帯域までに約 0.03 しか改善しなくなる。一方、ハイブリッドの損失は下がり続ける。バイト位置 128 で Gated DeltaNet を引き離し、バイト位置 256 以降は GRU も下回る。バイト位置 512〜1,023 では、Gated DeltaNet を約 0.05、GRU を約 0.02 リードする。このリードはすべて、第 4 ブロックに置かれた 1 つのアテンション層によるものである。

最も大きく損失が下がるのはアテンションだが、すべての実行でそうなるわけではない。大半の実行ではバイト位置 512〜1,023 で 1.03〜1.04 に達し、ハイブリッドを約 0.02 上回って 4 モデル中最高となる。しかし、約 5 回に 1 回の実行は遅れをとる。その場合、全体で約 1.12、最後の帯域で約 1.10 に終わり、Gated DeltaNet と変わらない。プロットは表中の 2 回の実行の平均を示しているため、その一方がこうした遅れた実行にあたると、アテンションの曲線はハイブリッドの下ではなく上で終わることになる。

この 2 種類の実行を分けるのは、たった 1 つの出来事である。最初の 1,000 ステップ前後までは、すべての実行が同じ訓練損失をたどる。その後、大半の実行では 1,000〜5,000 ステップのどこかで損失が約 0.05 急落する。一方、遅れをとった実行では損失がなだらかに下がるだけで、この急落が決して現れない。この急落こそ、アテンションがコピーを学習した瞬間である。テキストには繰り返しが現れる。人名や日付、言い回しは 1 キロバイト以内に何度も再登場することが多い。直前に読んだバイト列の過去の出現箇所を見つけ出し、そのとき後に続いたバイトを予測できる

モデルは、そうした繰り返しのたびに損失を減らせる。Olsson et al. (2022) はこの回路を**誘導ヘッド**（induction head）と呼んでおり、本書でも 10.7 節で既に扱った。この回路の有無は直接テストできる。同じ 512 バイトの文章を 2 回含むウィンドウを与えると、急落を経験した実行は 2 回目のコピーを 1 回目の 0.6〜0.8 倍の損失で予測する。遅れをとった実行は、8,000 ステップすべてを通じて両方のコピーをほぼ同じ損失で予測する。コピー元となる情報がほとんどないウィンドウ先頭の 128 バイトでは、この 2 種類の実行を見分けることはできない。

遅れをとった実行は不安定なわけではない。損失が発散することはなく、勾配の大きさも他の実行と変わらない。単に形成が遅いだけなのである。回路が形成されるステップはシードによって変動するが、コサインスケジュールでは 6,000 ステップ前後を過ぎると学習率がほとんど残らないため、それまでに回路を形成できなかった実行は最後まで形成できずに終わる。短い畳み込みは想起タスクと同様ここでも有効であり、これがないと大半の実行では 5,000 ステップまでに急落が起きない。他の学習率、重み減衰なし、異なるアテンションヘッド数も試したが、形成し損ねる頻度が下がる設定は 1 つあったものの、完全にゼロになる設定はなかった。単独ではほとんど役に立たない部品を組み合わせて回路を構築しなければならないとき、このような急落は典型的に生じる現象であり、2 つのアーキテクチャにランキングをつける前に複数のシードで訓練すべき十分な理由となる。

このように、長いコンテキストと十分な訓練があれば、テキスト上でも想起タスクで見られた傾向がより緩やかな形で現れる。拡大するキャッシュが離れたバイトから最も多くの恩恵を受け、3 つのリカレント層に 1 つのアテンション層を混ぜた構成がその改善の大部分を取り戻し、固定状態のみでは改善が最も小さい。128 バイトのウィンドウには利用できるような離れたバイトが存在しないため、短いウィンドウの表からはこの差が見えないのである。

12.8.4 推論時のメモリ

自己回帰生成（autoregressive generation）においては、系列あたりのメモリ量がサービング時の最大バッチサイズと、メモリ帯域幅律速となるデコーディング速度の両方を左右する。

- **グループ化クエリアテンション層**（16 ビット浮動小数点による $n_{\text{kv}} = 8$ 個のキー・値ヘッド、各ヘッドの次元 $d_{\text{head}} = 128$）：**トークンあたり** $2 \times n_{\text{kv}} \times d_{\text{head}} \times 2 = 4$ KB を保持し、層あたり 512 MB（$T = 128$K トークン時）に達する。
- **リカレント・行列状態層**（たとえば幅 4,096、チャネル数 8,192、状態次元 128 の Mamba-2 を 16 ビット浮動小数点で表現した場合）：**コンテキスト長によらず**一定の $8{,}192 \times 128 \times 2 = 2$ MB を保持する（わずか 512 トークンを超えた時点でアテンション層の KV キャッシュに追い抜かれる）。

ハイブリッドアーキテクチャでは、拡大していく KV キャッシュを蓄積するのはアテンション層だけである（Figure 12.17）。4 つのアテンション層を持つ 32 層モデル（Jamba (Lieber et al. 2024) のように 8 層中 1 層の割合）は、256K トークン時のキャッシュを 32 GB から 4 GB へと削減する。また、Qwen3-Next (Qwen Team 2025) や Kimi Linear (Kimi Team 2025b) のような 3 対 1 の Gated DeltaNet ハイブリッドは、KV キャッシュを $4\times$ 縮小する。

12.8.5 本章のメモリの比較

Table 12.3 に、本章で構築したすべての系列ミキサを因果アテンションおよびハイブリッドと並べて比較する（d はモデル幅、N は状態次元、$H = 2d$ は Mamba の拡張チャネル数、h はヘッド数、T は系列長を表す）。

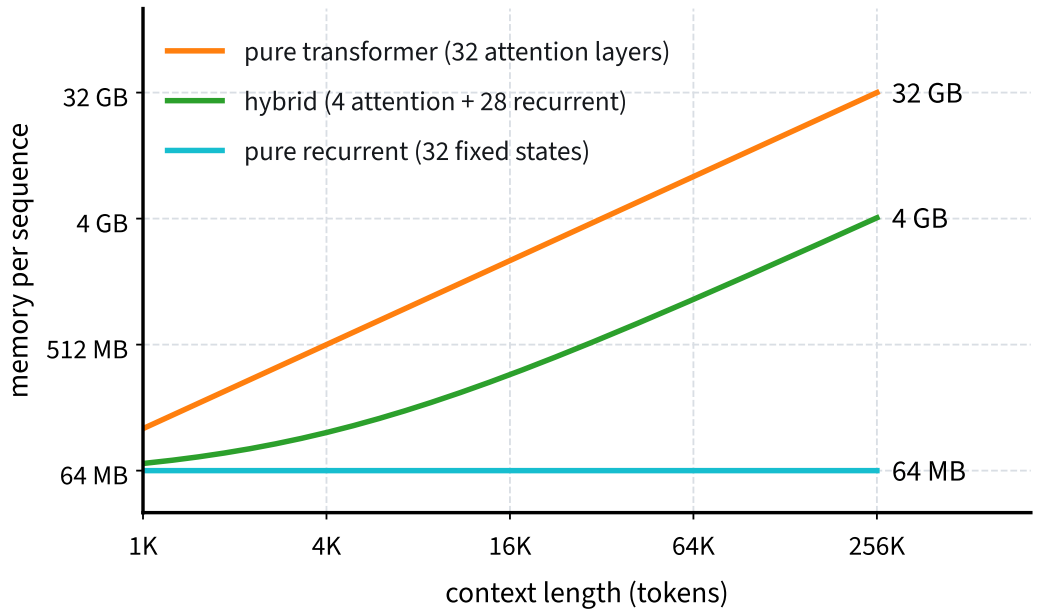


Figure 12.17: グループ化クエリアテンション（次元 128 のキー・値ヘッド 8 個）と Mamba-2 規模の再帰状態を備えた 32 層モデルにおいて、生成中に系列あたり保持されるメモリ量（16 ビット数値）とコンテキスト長の関係。Transformer のキャッシュは 256K トークンで 32 GB に達するが、純粋なリカレントモデルはあらゆる長さで 64 MB にとどまり、4 つのアテンション層を持つハイブリッドは Transformer の 8 分の 1 のキャッシュで済む。

Table 12.3: 本章のメモリ一覧。「テキスト」列はバイトレベル WikiText-103（128 バイトのウィンドウ、モデルごとに 2 回実行）におけるバイトあたり nat 単位の検証損失であり、括弧内はパラメータ数を示す。本節のモデルは 8,000 ステップ、それ以外は 1,000 ステップ訓練したため、同じステップ数の損失同士のみ比較できる。8,000 ステップ訓練後の 1,024 バイトのウィンドウでは、大半の実行で畳み込み付きアテンションが首位となり、ハイブリッド、GRU、Gated DeltaNet がそれに続く。

メモリ	生成時の層あたり状態	訓練	合成タスク	テキスト
LSTM、GRU	$2d$、d	1 ステップずつ逐次処理	1 付近で初期化したゲートは記号を 200 ステップ保持するが、0.5 で初期化すると 50 ステップまでに失う	1.29〜1.30（4.3 M、3.8 M）
minGRU	d	スキャン	100 ステップ後はランダム予測水準だが、S4D の時間スケールを用いれば 1,000 ステップ後も想起	1.34〜1.35（2.8 M）
S4D	dN	畳み込みまたはスキャン	1,000 ステップ後も想起、選択的コピーでは高々 5 回に 1 回	未訓練
Mamba	HN	スキャン	選択的コピーで 0.98 以上	1.30（4.0 M）
ゲート付き線形アテンション	d^2/h	チャンク処理	128 要素では 64 ペアの約半分、512 要素ではさらに多くを想起	未訓練
Gated DeltaNet	d^2/h	三角求解を伴うチャンク処理	書き換えられたキーの 0.95 以上を想起（ゲート付き線形アテンションは約 0.9）	8,000 ステップ後に 1.16（3.6 M）
アテンション	$2dT$	全位置にわたる融合アテンション	あらゆる負荷ですべてのペアを想起	1.38〜1.39、8,000 ステップ後は 1.18、幅 4 畳み込み付きで 1.15（3.3 M）
ハイブリッド	再帰状態およびアテンション層あたり $2dT$	両方	2 ブロック中 1 つをアテンションにしてすべてのペアを想起	8,000 ステップ後に 1.15（3.5 M）

各列を横断して読むと、設計上のトレードオフが浮かび上がる。

- 任意の過去トークンの厳密な参照が不要な**低消費電力ストリーミングや連続的なセンサー・音声処理**では、古典的なゲート付き再帰や選択的状態空間モデル（SSM）によってコンパクトな $O(1)$ 状態での処理が可能になる。
- **長文脈の基盤モデル**では、チャンク並列のデルタ則行列再帰（`GatedDeltaNet`）や選択的 SSM（`Mamba-2`）に少数の完全なアテンション層を（1:3〜1:7 の比率で）挟み込むことによ

り、線形時間の並列訓練、4×〜8× 小さい推論時 KV キャッシュ、および内容参照による厳密な想起を兼ね備えることができる。

12.8.6 まとめ

M 個の数値をそれぞれ b ビットで保持する固定サイズの状態は、高々 2^{Mb} 通りの履歴しか区別できない（$Mb \geq k \log_2 V$ が k 個のトークンを語彙 V から想起するための条件である）。行列状態層では、幾何学的な制約によってさらに早い段階で容量が制限される。Welch の下界により、$m > d_k$ 個の $\mathbb{R}^{d_k}$ 内の単位キーは二乗重なりの平均が少なくとも $(m - d_k)/(d_k(m-1))$ となり、加法的更新とデルタ則更新のいずれにおいてもキー間の干渉を引き起こす。マルチクエリ連想想起では、純粋な `GatedDeltaNet` モデルはペア数が状態要素数を超えると劣化するが、状態要素数を 4 倍にすると容量も比例して拡大し、リカレントブロックを 1 つだけ完全なアテンションに置き換える（`GA`）と全負荷で 100% の想起率が回復する。隣接するキートークンと値トークンを単一の層内で結び付けるには、ミキサの前に置く幅 4 の因果畳み込みが不可欠である。バイトレベルのテキストでは差はより小さく、ウィンドウ長に依存する。128 バイトのウィンドウでは、8,000 ステップの訓練後に `Gated DeltaNet`、ハイブリッド、および畳み込み付きアテンションが互いにバイトあたり約 0.01 nat 以内に収まり、素のアテンションに見られた約 0.03 の遅れも同じ畳み込みを加えれば解消する。1,024 バイトのウィンドウではバイト位置 128 以降で各モデルが分かれ、大半の実行でアテンションが離れたコンテキストから最大の改善を得て、ハイブリッドがそれに続き、純粋な `Gated DeltaNet` の改善が最も小さい。生成時にはハイブリッドのうちアテンション層のみが拡大する KV キャッシュを必要とするため、推論時のキャッシュメモリはアテンション層の割合に応じて削減される。

12.8.7 演習問題

1. **数え上げ限界**。モデルが長さ k トークンの系列（語彙サイズ V）を読み込み、それを再現しなければならないとする。
   1. Equation 12.39 を、読み込みと書き出しの間の状態が M 個の b ビット数値からなり、出力がその状態の決定論的な関数であるモデルについて証明せよ。
   2. 次に、系列が一様に選ばれるとき、モデルが高々確率 ϵ で誤った系列を出力してもよいとする。このとき $Mb \geq k \log_2 V + \log_2(1-\epsilon)$ が成り立つことを示せ。（ヒント：各状態が正しくデコードできる系列の個数を数える。）
   3. 語彙サイズ 50,000 のトークンを、48 層の Mamba-2 状態（各層は 16 ビット数値で 2 MB）を持つモデルは最大いくつコピーできるか。この規模のどのモデルも実際にはその上限に遠く及ばないのはなぜか。
2. **多数のユーザーのためのメモリ**。80 GB のメモリを持つ GPU が、Figure 12.17 のキャッシュサイズを持つ 32 層モデルをサービングしており、GPU メモリのうち 16 GB は重みに使われているとする。
   1. 長さ 128K トークンの系列を、この GPU は Transformer、4 つのアテンション層を持つハイブリッド、およびリカレントモデルのそれぞれで同時にいくつサービングできるか。
   2. キー・値ヘッド数を減らしたグループ化クエリアテンションと、キャッシュの 8 ビット数値化は、それぞれキャッシュを半分にする。両方を Transformer とハイブリッドに適用し、同じ計算を繰り返せ。2 つの変更のうちハイブリッドの再帰状態も縮小できるのはどちらか。また、両方の変更を適用した後、ハイブリッドのメモリに占める再帰状態の割合はいくらか。
3. **重なりと訓練**。
   1. 訓練によって、ヘッドのキーをランダムなキーよりどれだけ改善できるか。下界 Equation 12.40 をランダムな単位キーの平均二乗重なり $1/d_k$ で割り、64 個のキーについて次元 4、8、16、32 での比を求めよ。これらのヘッドのうち、訓練されたキーが

ランダムなキーより明確に優れ得るのはどれか。
2. [code] スイープ実験で用いた小規模な Gated DeltaNet モデルを 4,000 回および 8,000 回の更新（`num_updates` を `d2l.train_recall` に指定）で訓練せよ。32 ペアおよび 64 ペアの想起率はハイブリッドに近づくか。その結果から、この小規模モデルに不足しているのは訓練量と容量のどちらだといえるか。
3. [code] キーの候補数を 64 種類および 512 種類としてキーを抽出せよ（`num_keys` を変更し、それに合わせて `recall_model` の語彙も変更する）。1 つの系列が格納するペア数は依然として高々 64 個である。それにもかかわらず、キーの候補数によって想起率が変化し得るのはなぜか。

4. [code] **アテンションをどこに、どれだけ置くか**。
   1. 第 1 ブロックにアテンションを置いたハイブリッド `AG` を訓練し、その想起率を `GA` と比較せよ。アテンション層の位置は結果に影響するか。どの層が各キーとその後続の値を結び付けられるかを踏まえて考察せよ。
   2. 同じ幅を持つ 4 ブロックのモデル `GGGG`、`GGGA`、`GAGA` を訓練せよ。リカレント層を 2 つ増やすと、層あたりの状態を 4 倍にしたときと同じだけ想起曲線は押し上がるか。また、4 層中 1 層のアテンションでも依然としてすべてのペアを想起できるか。
5. [code] **テキストにおける想起**。バイトレベル WikiText-103 の検証ウィンドウにおいて、同じウィンドウ内のより前の位置に既に出現した少なくとも 8 バイトの文字列を完成させるバイトの位置に印を付けよ。
   1. 訓練済みの Gated DeltaNet、ハイブリッド、およびアテンションモデル（`trained`）の損失を、印を付けた位置とそれ以外のすべての位置とで比較せよ。想起におけるアテンションの優位性は、テキストが繰り返される箇所に現れるか。
   2. 512 バイトのウィンドウ（`num_steps=512` とし、ステップあたりのウィンドウ数を減らす）で同じ実験を繰り返せ。印を付けた位置の割合はどう変化し、それに伴ってモデル間の差はどう変わるか。
6. [code] **コピー回路の形成を観察する**。後半が前半の繰り返しになっている 1,024 バイトの検証ウィンドウを作成せよ。畳み込み付きアテンション（`'CCCC'`）を通常の 1,024 バイトのウィンドウ上で訓練し、250 ステップごとに訓練損失と、繰り返しウィンドウの前半の損失に対する後半の損失の比を記録せよ。
   1. この比が 1 を下回るのは何ステップ目か。また、同じステップで訓練損失も急落するか。
   2. 複数のシードで繰り返せ。急落が起きるステップはどの程度変動するか。また、8,000 ステップ以内に急落が起きない実行はどのような挙動を示すか。

12.9 関連文献

本章では、現代の長文脈モデルがアテンションと組み合わせるリカレント層を構築し、その起源を LSTM までさかのぼり、固定サイズの状態がどこで容量の限界に達するかを計測してきた。この分野の文献の大半は発表から 5 年未満のものであり、そのすべてが今後も残るとは限らない。本節では、本章で扱わなかった拡張手法や未解決問題とあわせて、参照実装およびチュートリアルを概観する。

12.9.1 再帰とアテンションの組み合わせ

12.9.1.1 実用におけるハイブリッドモデル

12.8 節の演習問題では、ハイブリッドモデルにどれだけのアテンションが必要か、またスタック内のどこに配置すべきかを問うた。3 つ目の問いは、スライディングウィンドウアテンションで大域アテンション（global attention）を代替できるかという点である。条件を揃えた Waleffe et al. (2024) の 8B パラメータ比較では、24 個の Mamba-2 層と 4 個のアテンション層（MLP を含

む全層の約 8%）を組み合わせ、位置埋め込み（positional embedding）が不要になるよう先頭に Mamba-2 層を置き、アテンション層を等間隔に配置している。この比率において、ハイブリッドモデルは 12 個の標準ベンチマークすべてで Transformer に匹敵するかそれを上回ったのに対し、純粋な Mamba モデルは 5-shot MMLU と電話帳ルックアップで遅れをとった。

実用モデルにおける具体的な混合比率は多様である。Qwen3-Next (Qwen Team 2025) と Kimi Linear (Kimi Team 2025b) はデルタ則の漸化式（12.7 節）の周囲で 4 層に 1 層の割合でアテンション層を用い、Jamba (Lieber et al. 2024; Jamba Team 2024) は 8 層に 1 層を用い、Nemotron-H (NVIDIA 2025) は Mamba-2 層を維持している。Griffin (De et al. 2024) は大域アテンションを完全に廃し、2 つのゲート付き対角漸化式層と 1 つのスライディングウィンドウアテンション層（10.6 節）を交互に配置している。ただし、ウィンドウは直近の w トークンしか保持しないため、コンテキストがウィンドウ幅を超えると厳密なルックアップ性能は劣化する。

12.9.1.2 Transformer からの蒸留

リカレントモデルやハイブリッドモデルをゼロから事前学習する代わりに、事前学習済みの Transformer のアテンション射影から行列状態層を初期化し（12.6 節）、蒸留（distillation）によって変換することもできる。Junxiong Wang et al. (2024) は一部のアテンション層を残してハイブリッドを構成する一方、Bick et al. (2024) は元の事前学習予算の 1% 未満にあたる 30〜50 億トークンを用いて、アテンションマップ、隠れ状態、および最終出力を段階的に整合させている。

12.9.1.3 固定状態と完全なキャッシュの間

固定サイズの状態と完全な KV キャッシュの間を補間する設計が 2 つある。Based (Arora, Eyuboglu, Zhang, et al. 2024) は、$\exp(\mathbf{q}^\top \mathbf{k})$ に対する 2 次テイラー近似と厳密なスライディングウィンドウアテンションを組み合わせ、状態サイズと想起性能を直接トレードオフさせる。対数線形アテンション（log-linear attention、(Guo et al. 2025)）は、過去へさかのぼるにつれて長さが 2 倍ずつ増える 2 進（dyadic）履歴ブロックごとに 1 つの行列状態を保持するため、メモリは $O(\log T)$ で増大し、Mamba-2 や Gated DeltaNet にそのまま適用できる。

12.9.2 行列メモリへの書き込み

12.9.2.1 固定状態モデルの表現力

系列タスクの中には、状態サイズをどれだけ大きくしても、対角または時不変な線形漸化式では解けないものがある。5 つのカップの下に隠されたボールを一連の入れ替えにわたって追跡する問題、すなわち対称群（symmetric group）S_5 における置換の合成を考えよう。Merrill et al. (2024) は、対角遷移または固定遷移を持つ状態空間モデル（S4 や Mamba を含む）が、Transformer と同様に、定数深さしきい値回路クラス（constant-depth threshold circuit class）TC^0 に属する関数しか計算できないことを証明している。5 要素以上の置換の合成は NC^1（対数深さ回路）完全であり、したがって TC^0 の外にある（ただし $\mathsf{TC}^0 = \mathsf{NC}^1$ でない限り）。（カップが 3 つまたは 4 つの場合は、対称群が可解群（solvable group）となるため TC^0 の内部にとどまる。）

デルタ則の遷移 $\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top$ は入力依存かつ非対角であるため、TC^0 の障壁によっては排除されない。しかし Grazzi et al. (2025) は、有限精度のもとでは、遷移の固有値が $[0, 1]$ 内に限定されるいかなる線形漸化式も、任意の長さにおいてパリティを計算できず、したがって入れ替えも追跡できないことを示している。標準の `GatedDeltaNet` はシグモイドによる $\beta_t \in (0, 1)$ を用いるため、この制約を受ける。$\beta_t \in (0, 2)$ を許容すれば、遷移は負の固有値をとれるようになる。実際、$\beta_t = 2$ かつキーが $(\mathbf{e}_i - \mathbf{e}_j)/\sqrt{2}$ のとき、$\mathbf{I} - 2\mathbf{k}_t \mathbf{k}_t^\top$ は座標 i と j を入れ替える鏡映（reflection）となる。12.7.3.5 節では、この派生モデルをパリティ課題で訓練している。

12.9.2.2 一般化デルタ則、指数ゲート、ネットワークメモリ

- **一般化デルタ則：**RWKV-7 (Peng et al. 2025) は、各座標を固有のレートで減衰させ、あるキーベクトルに沿って消去しつつそれとは異なるキーベクトルに沿って書き込むことでデルタ則のランク 1 射影の共有を解き、任意の正規言語（regular language）を認識可能な非対称遷移を実現している。DeltaProduct (Siems et al. 2025) はトークンごとに複数の鏡映 $\prod_i (\mathbf{I} - \beta_i \mathbf{k}_i \mathbf{k}_i^\top)$ を合成し、1 ステップでより高ランクの置換を表現できるようにしている。
- **指数ゲート：**シグモイド入力ゲートは、単一トークンの書き込み重みを最大 1 に制限する。xLSTM (Beck et al. 2024, 2025) の mLSTM は、指数入力ゲートを逐次正規化項の状態および 10.5 節のオンライン対数最大値（log-max）安定化と組み合わせて用いる。
- **ネットワークメモリ：**TTT (Y. Sun et al. 2024) と Titans (Behrouz et al. 2025) は、12.7 節におけるオンライン勾配の視点を一般化し、行列 $\mathbf{S}_t$ をトークンごとの勾配ステップで更新される線形層や小規模な MLP に置き換える（Titans ではモメンタムと重み減衰も併用する）。非線形な高速重みが、同じパラメータ数の行列よりも状態の 1 バイトあたり多くの対応関係を想起できるかどうかは、依然として未解決の問いである。

12.9.3 状態空間モデルと長い畳み込み

12.9.3.1 多項式射影による記憶

12.4 節では、S4D のレートを $\lambda_n = -n$ と初期化した。この選択は HiPPO (Gu et al. 2020) に由来する。HiPPO は、次数 $(N-1)$ の直交多項式による入力履歴 $x(s)$（$s \le t$）の最良近似の係数を保持する手法である。これらの係数は、固定された下三角の HiPPO 行列 $\mathbf{A}$ を持つ線形常微分方程式（ODE）を満たす。S4 (Gu, Goel, et al. 2022) はこの系を時不変にし（指数関数的に時間伸縮された履歴上の Legendre 多項式に対応する (Gu et al. 2023)）、高速なカーネル評価のために $\mathbf{A}$ を正規行列（normal matrix）と低ランク補正の和へと分解した。S4D (Gu, Gupta, et al. 2022) は、対角成分（複素固有値、または実対角成分 $-1, \dots, -N$）のみを残すだけで、はるかに単純な仕組みでほぼ同等の性能を回復できることを示した。S5 (Smith et al. 2023) はチャネルごとのスカラー SSM を、並列スキャン（12.3.2 節）で評価される単一の多入力・多出力対角 SSM に置き換えており、Gu の博士論文 (Gu 2023) がこの一連の研究を体系的にまとめている。

12.9.3.2 Mamba の後継モデル、直接的な線形漸化式、長い畳み込み

- **Mamba-2 と Mamba-3：**Mamba-2 (Dao and Gu 2024) は各ヘッド内でスカラー減衰を共有することで、訓練を 12.6 節の行列積に基づくチャンク形式へと帰着させる。Mamba-3 (Lahoti et al. 2026) は、台形離散化（trapezoidal discretization）、回転追跡のための複素状態、および自己回帰デコーディング時の演算密度（arithmetic intensity）を高める多入力更新を導入している。
- **Linear Recurrent Unit（LRU）：**@Orvieto.Smith.Gu.ea.2023 は、単位円板内の複素固有値と入力正規化 $\sqrt{1-|\lambda|^2}$ を用いて離散対角漸化式を直接パラメータ化し、Long Range Arena において連続時間 SSM に匹敵する性能を達成するとともに、Griffin で用いられている実数値版 RG-LRU の動機を与えた。
- **長い畳み込み：**H3 (Fu et al. 2023) はシフト SSM と乗法的連想 SSM を直列に連結し、Hyena (Poli et al. 2023) は SSM カーネルを完全に廃し、FFT によって $O(T \log T)$ で評価される陰的にパラメータ化された全長フィルタに置き換えた。一般の長い畳み込み（long convolution）は $O(1)$ デコーディングのためのコンパクトな再帰状態を持たないため、言語モデルでは主にリカレント SSM が選ばれてきたが、ゲノミクス分野では長い畳み込みが依然として広く用いられている。

12.9.4 言語を超えて

言語モデリング以外にも、画像パッチ上の双方向スキャンが Vision Mamba (Zhu et al. 2024) や VMamba (Y. Liu et al. 2024) を支えており、SaShiMi (Goel et al. 2022) ではマルチスケールの S4 スタックが生の音声波形をモデル化している。さらに、100 万ヌクレオチド規模のゲノムコンテキストは、HyenaDNA (Nguyen et al. 2023) では長い畳み込みにより、Caduceus (Schiff et al. 2024) では逆相補鎖同変（reverse-complement-equivariant）な双方向 Mamba 層によってモデル化されている。

12.9.5 参照実装とチュートリアル

- **実装：**The Annotated S4（Rush & Karamcheti, 2022）は、完全な S4 を 1 行ずつ構築する。state-spaces/mamba（Gu & Dao）は Mamba および Mamba-2 の `ssd_minimal` の参照実装を提供しており、mamba-minimal は単一ファイルの PyTorch 版 Mamba を与えている（12.5 節）。flash-linear-attention は、GLA、DeltaNet、および Gated DeltaNet 向けの Triton カーネルを提供している（12.6 節、12.7 節）。Gated DeltaNet for Linear Attention（Raschka, 2025）は Gated DeltaNet ハイブリッドをゼロから構築するチュートリアルを提供しており、zoology は 12.6 節および 12.8 節で使用した合成連想想起スイートを提供している。
- **解説とサーベイ：**On the Tradeoffs of SSMs and Transformers および State Space Duality (Mamba-2), parts I–IV（Gu & Dao, 2024〜2025）、A Visual Guide to Mamba and State Space Models（Grootendorst, 2024）、DeltaNet Explained, parts I–III（Yang, 2024）、ならびに Test-Time Regression（Wang, Shi & Fox, 2025）がある。最後の文献は、12.7 節の末尾で扱ったメモリの統一的な回帰としての見方をまとめている。

Chapter 13

計算性能

ここまでは、どのモデルを訓練するときも、ネットワークを書き、訓練ループを開始して完了を待つだけであり、ステップ時間とメモリは所与のものとして扱ってきた。しかし、実行に何日もかかる場合や、モデルがカードに収まらない場合、あるいは 2 基目の GPU を追加するとかえって訓練が遅くなる場合には、そのやり方は通用しなくなる。GPU により多くの処理を与えるため、11.2 節の GPT の幅を 2 倍に広げたものを考えてみよう。1 回の訓練ステップの算術演算量を GPU のピーク演算速度で割れば、そのステップが取り得る最短時間が求まるはずだが、実測したステップにはその数倍の時間がかかる。残りの時間はどこへ消えているのだろうか。

コンピュータアーキテクトは、深層学習よりはるか前からこの問いに直面していた。1990 年代までにはプロセッサの速度がメモリを大きく引き離し、多くのプログラムが実行時間の大部分をデータ待ちに費やすようになっていた。2009 年に Williams、Waterman、Patterson は、その帰結を**ルーフライン**モデルとして定式化した (Williams et al. 2009)。すなわち、プログラムは演算器が許す速度を超えて動作することはなく、また必要なバイト数をメモリが供給できる速度を超えて動作することもない。Python から駆動される GPU には、各演算を発行するホスト側のコストという 3 つ目の制約が加わる。これら 3 つのうちどれがステップを律速しているかによって、どの変更が高速化につながるかが決まる。この視点に立てば、大きなミニバッチや 16 ビット数値（5.5 節）といった前章までの選択の理由が説明できるとともに、次のような方法論が得られる。ステップの算術演算量とバイト数を数え、実行時間を予測し、実測して、その差を説明するのである。

すべての土台となる 13.1 節では、1 バイトの移動あたりに演算が実行する FLOPs である**演算密度**（arithmetic intensity）を導入し、この方法論を実際に適用する。ハードウェアの節（13.2 節）では、ピーク演算性能と帯域幅がどこから生じるのかを説明する。コンパイル（13.3 節）と 13.4 節のメモリ手法は、多くの実務者が日常的に使う単一 GPU 向けのツールである。続く 3 つの節では GPU を増やし、第一原理からのデータ並列（data parallelism、13.5 節）と実運用におけるデータ並列（13.6 節）を扱ったのち、重み、勾配、オプティマイザ状態を複数のカードにシャーディングする手法（13.7 節）を導入して、単一の GPU には収まらないほど大きなモデルを訓練できるようにする。最後に、ケーススタディ（13.8 節）では上述の GPT に立ち返り、1 基および 4 基の GPU 上でそのステップ時間の内訳をミリ秒単位で明らかにする。

本章では、GPU コードの実行時間とピークメモリを計測でき（5.7 節）、bf16 と tf32 を理解しており（5.5 節）、GPT とそのキー・値キャッシュに触れていること（11.2 節、11.3 節）を前提とする。すべての計測値は、4 枚の RTX 4090 カード、PyTorch 2.11、JAX 0.12 を備えた 1 台のホスト上で取得したものである。手元のマシンでは実行時間こそ異なるだろうが、FLOPs とバイト数のカウントは変わらず、それらを予測へと変換する方法論も変わらない。

13.1 性能モデル

これまで訓練してきたすべてのネットワークは算術演算の大半を行列積に費やしており、GPU はそれをピーク速度に近い速さで計算すると暗黙のうちに仮定してきた。しかし、その仮定は誤っていることが多い。本書のノートブックを実行している RTX 4090 では、2 つの 8192×8192 行列の bf16 行列積はハードウェアのピーク性能である 165 TFLOP/s から数パーセント以内で動作する一方、256×256 の行列積は適切に計測してもピークの 2% 未満にしか達しない。同じ演算、同じチップであるにもかかわらずである。訓練ステップは多数の演算の連なりであり、その中には小さな演算も含まれる。それらの大半がピークのごく一部でしか動作していないとき、より高速な GPU や 2 枚目のカードを導入してもほとんど改善しない。時間が計算ではなく待ちに費やされているからである。

GPU 上のあらゆる演算は、算術演算器、そこにデータを供給するメモリ帯域幅、あるいは Python から演算を発行する際の固定的なホスト側のコストという 3 つの資源のいずれかによって制限される。演算が実行する FLOPs と移動させるバイト数という机上で数えられる 2 つの量によって、最初の 2 つのどちらが制約になるかが決まる。そこに呼び出しを発行するためのホスト時間 h という 1 つの実測定数を加えれば、3 つ目の制約も予測できる。本章全体を通じたアプローチは、FLOPs とバイト数を数え、実行時間を予測し、実測し、その差を説明することである。

```
import collections
import re
import time
```

JAX は NVIDIA GPU 上ではデフォルトで fp32 の行列積を tf32 テンソルコア上で実行する。

13.1.1 演算密度とルーフライン

実際に時間を計測する前でも、演算の形状からどの資源がそれを制限するかを予測できる。この考え方は 11.3.2.1 節でも簡単に用い、プレフィルが自己回帰デコーディングよりもはるかに高速にトークンを処理できる理由を説明したが、ここではそれを本格的に展開する。

13.1.1.1 FLOPs とバイト数のカウント

あらゆるテンソル演算は GPU に 2 つの要求を課す。1 つは**浮動小数点演算**（floating-point operations、FLOPs。加算または乗算のそれぞれを 1 回と数える）で測られる算術演算の実行であり、もう 1 つは GPU のメインメモリとオンチッププロセッサの間でのバイト単位のデータ移動である。

GPU にはそのそれぞれに対して個別のバジェット（上限）がある。RTX 4090 のメインメモリが供給できるのは最大で $\beta = 1.008 \times 10^{12}$ bytes/s（約 1 TB/s）である。ピーク算術演算速度 P は数値形式と演算器に依存し、テンソルコア上の bf16 行列積は $P = 165 \times 10^{12}$ FLOP/s に達する一方、tf32 行列積と標準的な fp32 算術演算は定格の 2.52 GHz ブーストクロックにおいてその半分の速度（82.6 TFLOP/s）で動作する。定格クロックは公称値であり、実際のカードは温度や消費電力に応じて数パーセント速く、あるいは遅く動作する。したがって、実測速度が P の前後約 5% 以内に収まっていれば、ルーフに達しているものとみなす。

どちらのバジェットが先に尽きるかは、両者の比である**演算密度**（arithmetic intensity）によって決まる。

$$I = \frac{\text{FLOPs performed}}{\text{bytes moved}}.$$

行列積 **XW**（ただし $\mathbf{X} \in \mathbb{R}^{B \times D}$、$\mathbf{W} \in \mathbb{R}^{D \times F}$）、すなわち B 行のバッチに適用される線形層を考えよう。この演算は $2BDF$ FLOPs（内積の各項につき乗算 1 回と加算 1 回）を実行し、最良の場合でも **X** と **W** を 1 回ずつ読み出し、出力を 1 回書き込む（合計 $BD + DF + BF$ 個の数値）。数値 1 個あたり b バイトとすると、その演算密度は次のようになる。

$$\begin{aligned} \frac{2BDF}{b(BD + DF + BF)} &= \frac{2}{b}\left(\frac{1}{F} + \frac{1}{B} + \frac{1}{D}\right)^{-1} \\ &\approx \frac{2B}{b} \quad \text{for } B \ll D, F. \end{aligned} \tag{13.1}$$

中辺の式から、(B, D, F) のうち最小のものが演算密度を支配することがわかる。バッチサイズ B が層の次元よりはるかに小さいとき、演算密度は B に比例して増加する。メモリから読み出された各重みが、B 個すべての行にわたって再利用されるからである。

幅 $D = F = 2048$、bf16（$b = 2$）の 1B パラメータの GPT では、1,024 トークンの系列 8 本からなる訓練ミニバッチは $B = 8192$ 行となり、Equation 13.1 より $I \approx 910$ FLOP/byte となる（幅 D によって制限される）。単一系列の自己回帰生成（autoregressive generation）では $B = 1$ であり、$I \approx 1$ FLOP/byte となる。同じ層であっても、訓練とデコーディングの間で演算密度はほぼ 3 桁にわたって変化する。さらにその下限に位置するのが $y_i = x_i + 1$ のような要素ごとの演算であり、テンソルのサイズにかかわらず、移動する $2b$ バイトあたり 1 FLOP しか実行しない（bf16 では $I = 0.25$ FLOP/byte）。

13.1.1.2 ルーフラインモデル

ルーフラインモデル (Williams et al. 2009) は、演算密度をスループットの上界へと変換する。すなわち、いかなる演算も計算ルーフ P より速くは動作せず、メモリが速度 $I\beta$ でデータを供給するペースより速くも動作しない。

$$\text{performance} \;\le\; \min(P,\; I\beta). \tag{13.2}$$

両対数軸上では、この上界は傾きをもつ帯域幅の直線と水平な計算ルーフが**リッジポイント** $I^* = P/\beta$ で交わる形になる（Figure 13.1）。I^* 未満では演算は**帯域幅律速**（bandwidth-bound）となる。メモリが算術演算器に十分なデータを供給できないため、すでにオンチップにあるデータに対する追加の算術演算は実質的に無償で行える。I^* を超えると演算は**計算律速**（compute-bound）となる。算術演算器が飽和しているため、データ移動のバイト数を減らしても高速化しない。本章全体を通じて、β はメインメモリ（DRAM）の帯域幅を表す。72 MB のオンチップ L2 キャッシュに収まるほど小さいオペランドは、このルーフを超えることがある。

13.1.1.3 リッジポイント

RTX 4090 上の bf16 テンソルコア行列積では、リッジポイントは $I^* = 165.2/1.008 \approx 164$ FLOP/byte であり、tf32 行列積（$P = 82.6$ TFLOP/s）では $I^* \approx 82$ FLOP/byte である。現代のデータセンター向け GPU では、HBM の帯域幅よりもテンソルコアの FLOP/s の方が速く向上してきたため、リッジポイントはさらに高い（280〜300 FLOP/byte 前後）。これらのしきい値を超えるのは大規模な行列積だけであり、要素ごとの演算、正規化、単一トークンのデコーディングはいずれもリッジポイントを大きく下回る。

正方行列同士の bf16 行列積（$B = D = F = n$）では $I = n/3$ となるため、ルーフラインモデルからは $n \gtrsim 500$ を満たすすべての行列積が計算律速になると予測される。この予測を検証するには、GPU の実行時間を正確に計測する必要がある。

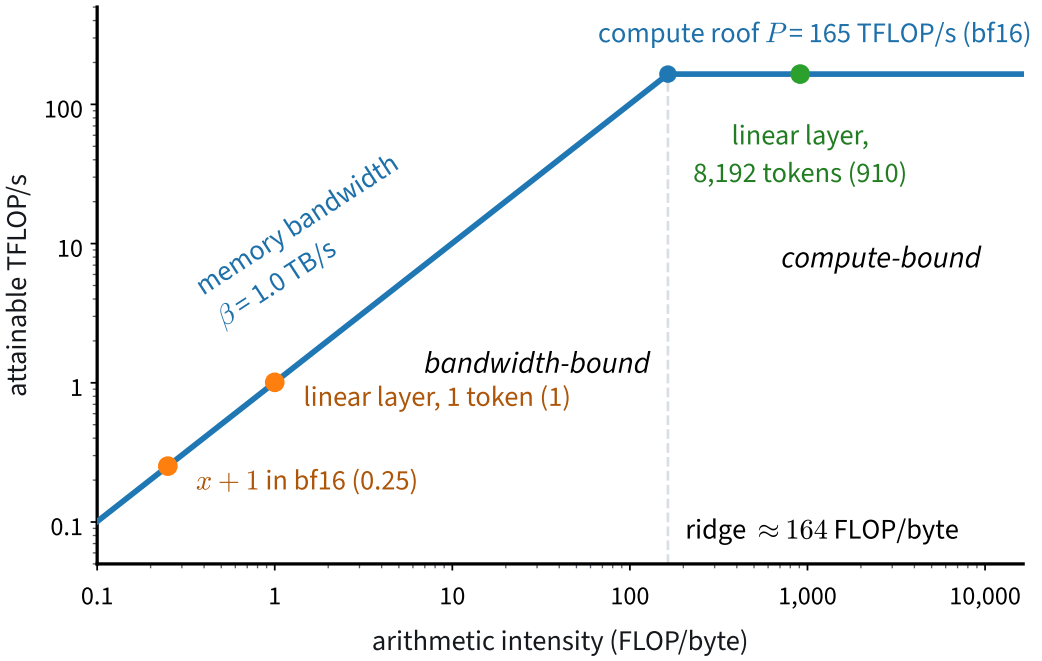


Figure 13.1: RTX 4090 のルーフライン。達成可能なスループットは、計算ルーフ P と、演算密度にメモリ帯域幅 β を掛けた値のうち小さい方となる。図中の点は本文の例（bf16 での $x + 1$、1 トークンに適用した幅 2048 の線形層、および 8,192 トークンのミニバッチに適用した同じ層）を示す。リッジポイントは 164 FLOP/byte 付近にある。

13.1.2 GPU コードの時間計測

GPU へのディスパッチは非同期であるため（5.7.2.1 節）、Python はカーネルをキューに入れると即座に次の処理へ進む。同期をとらずにカーネル呼び出しの前後で実経過時間を測ると、計測されるのは Python が処理をエンキューするのに要した時間であり、GPU がそれを実行するのに要した時間ではない。正確な時間計測では、`d2l.Benchmark` が行うように、まずデバイスをウォームアップし、計測の開始前と停止前に同期をとり、多数回の反復にわたって平均をとる。

13.1.2.1 同期ポイント

Python が GPU の完了を待ってブロックするのは、デバイス上にある値を要求したときだけである。たとえば、テンソルを Python のスカラーや NumPy 配列に変換するとき、それを表示するとき、その値で条件分岐するとき（`if loss < 0.1:`）、あるいは出力の形状がテンソルの値に依存する演算（ブールマスキング `x[x > 0]` など）を実行するときである。

ホストが 1 ステップを発行するのに時間 h を要し、GPU がそれを実行するのに t_{dev} を要するとしよう。同期のないループではホストディスパッチとデバイス実行がオーバーラップするのに対し、毎ステップでスカラーを読み出すと両者が直列化され、さらにデバイスからホストへの往復レイテンシ σ が加わる。

$$t_{\text{once}} \approx \max(h, t_{\text{dev}}), \qquad t_{\text{every}} \lesssim h + t_{\text{dev}} + \sigma.$$

1024×1024 行列の自身との積を計算し、その総和を GPU 上で累積する 2 つの 1,000 ステップのループを比較しよう。一方は累積和を毎ステップ Python の `float` に変換し、もう一方は最後に 1 回だけ読み出す。

```
x = jnp.ones((1024, 1024))
step = jax.jit(lambda s, x: s + (x @ x).sum())
s = step(jnp.zeros(()), x)
float(s)                          # Warm up: compile the step

t0 = time.perf_counter()
for _ in range(1000):
    s = step(s, x)
    total = float(s)              # Reads the value: a barrier every step
read_every = time.perf_counter() - t0

t0 = time.perf_counter()
for _ in range(1000):
    s = step(s, x)                # Stays on the device: no barrier
total = float(s)                  # One read at the end
read_once = time.perf_counter() - t0
print(f'read every step: {read_every:.3f} s   read once: {read_once:.3f} s')
```

```
read every step: 0.283 s   read once: 0.083 s
```

ステップごとの読み出しを行わなければ、JIT コンパイルされた各呼び出しにかかるホストディスパッチ時間は約 0.1 ms である。毎ステップ `float(s)` を読み出すと、反復ごとにデバイスからホストへの往復時間 σ が加わり、ループは 3 倍以上遅くなる。

したがって、訓練ループでは評価指標をデバイス上で累積し、ログ出力の間隔でのみ読み出すべきである。

13.1.2.2 ベンチマークヘルパ

これ以降、演算時間の計測には `d2l.Benchmark` を用いる。このヘルパはまずウォームアップ呼び出しを実行してコンパイルやカーネル読み込みのオーバーヘッドを吸収し、続いて 2 つのデバイスバリアの間で `repeats` 回の呼び出しを計測する。最初の確認として、4096×4096 の bf16 行列積は演算密度が $I \approx 1365$ FLOP/byte とリッジポイントを大きく上回るため、ルーフラインモデルからは $2 \cdot 4096^3 / P \approx 0.83$ ms と予測される。

```
matmul = jax.jit(jnp.dot)
a = jax.random.normal(d2l.get_key(), (4096, 4096), dtype=jnp.bfloat16)
d2l.Benchmark(lambda: matmul(a, a), desc='4096 x 4096 product')
```

```
4096 x 4096 product: 0.89 ms/call
```

実測時間は、予測値である 0.83 ms の約 10% 以内に収まっている。

13.1.3 3 つのレジーム

GPU プログラムの実経過時間は、計算律速なカーネル、帯域幅律速なカーネル、そしてデバイスがホストを待つアイドル時間の隙間に分けられる (He 2022)。3 つの実験によって各レジームを切り分けよう。

13.1.3.1 帯域幅ルーフと計算ルーフ

まず、活性化値からなる B 行の行列に、$d \times d$（$d = 16{,}384$、512 MiB）の bf16 重み行列を掛ける。Equation 13.1 より、その演算密度はおよそ B であり、$B = 164$ 付近でリッジポイントを横切る。リッジポイント未満では、どの呼び出しでも B にかかわらず 512 MiB の重みをストリーミングしなければならず（1.008 TB/s で約 0.53 ms）、リッジポイントを超えると $2Bd^2$ FLOPs が実行時間を支配して B に比例して増加する（$B = 256$ で 0.83 ms、$B = 1024$ で 3.32 ms）。

```
peak_tflops = 165.2    # RTX 4090: dense bf16 on the tensor cores
bandwidth_tbs = 1.008  # RTX 4090: GDDR6X memory
print(f'{jax.devices()[0].device_kind}: ridge point '
      f'{peak_tflops / bandwidth_tbs:.0f} FLOP/byte')
d, b = 16384, 2  # Width and bytes per bf16 number
W = jax.random.normal(d2l.get_key(), (d, d), dtype=jnp.bfloat16)  # 0.54 GB
for B in (1, 4, 16, 64, 128, 256, 1024):
    X = jax.random.normal(d2l.get_key(), (B, d), dtype=jnp.bfloat16)
    t = d2l.Benchmark(lambda: matmul(X, W)).time
    flops, nbytes = 2 * B * d * d, b * (B * d + d * d + B * d)
    t_mem, t_comp = nbytes / bandwidth_tbs / 1e12, flops / peak_tflops / 1e12
    roof = 'memory' if t_mem > t_comp else 'compute'
    print(f'B = {B:4d}: bound {1e3 * max(t_mem, t_comp):4.2f} ms ({roof:7s}), '
          f'measured {1e3 * t:4.2f} ms, {flops / t / 1e12:5.1f} TFLOP/s')
```

```
NVIDIA GeForce RTX 4090: ridge point 164 FLOP/byte
B =    1: bound 0.53 ms (memory ), measured 0.60 ms,   0.9 TFLOP/s
B =    4: bound 0.53 ms (memory ), measured 0.59 ms,   3.6 TFLOP/s
B =   16: bound 0.53 ms (memory ), measured 0.59 ms,  14.5 TFLOP/s
B =   64: bound 0.54 ms (memory ), measured 0.62 ms,  55.6 TFLOP/s
B =  128: bound 0.54 ms (memory ), measured 0.60 ms, 114.8 TFLOP/s
```

```
B =  256: bound 0.83 ms (compute), measured 0.82 ms, 166.9 TFLOP/s
B = 1024: bound 3.33 ms (compute), measured 3.27 ms, 168.0 TFLOP/s
```

1 つの重み行列で両方のルーフに到達している。$B = 1$ から $B = 128$ までは算術演算量が 128× に増えるにもかかわらず実行時間は 0.6 ms 付近でほぼ一定にとどまり、$B = 256$ 以上では計算ルーフ付近で B に比例して増加する。1,024 トークンを一度にサービングすると、1 トークンをサービングする場合の 1,024× の算術演算を、6× 未満の時間で実行できる。

13.1.3.2 要素ごとの演算

次に、リッジポイントを大きく下回る領域にとどまったまま、8192×8192 の fp32 テンソル（256 MiB）に対して要素あたりの算術演算を変えてみよう。試すのは加算、乗算、正弦（sin）、シグモイドである。いずれも 256 MiB を読み出して 256 MiB を書き込む（合計 512 MiB、1.008 TB/s で 0.53 ms）。

```
x = jax.random.normal(d2l.get_key(), (8192, 8192))  # 256 MiB in fp32
for desc, f in [('add', lambda x: x + 1), ('mul', lambda x: x * 1.5),
                ('sin', jnp.sin), ('sigmoid', jax.nn.sigmoid)]:
    f = jax.jit(f)
    print(d2l.Benchmark(lambda: f(x), desc=desc))
```

```
add: 0.62 ms/call
mul: 0.60 ms/call
sin: 0.60 ms/call
sigmoid: 0.59 ms/call
```

4 つすべてが約 0.6 ms で完了し、約 0.9 TB/s の実効メモリ帯域幅を達成する。メモリのデータ移動量が実行時間を決めるため、超越関数を含む演算（`sin`、`sigmoid`）であっても `x + 1` より余分な時間はかからない。

即時実行（eager execution）では、要素ごとの演算を連ねるとこのメモリ通信量が積み重なる。以下に示す GELU の tanh 近似（`gelu_ish`）は、8 個の要素ごとの演算を連鎖させている。6 個の単項演算がそれぞれ 1 個のテンソルを読み書きし（計 12 個のテンソル）、2 個の二項演算（`x + 0.04 * x**3` と最後の `0.5 * x` との乗算）がそれぞれ 2 個のテンソルを読み出して 1 個を書き込むため（計 6 個のテンソル）、合計で 256 MiB のテンソル 18 個分、すなわち単一の加算の 9× のバイト数を移動させることになる。

```
def gelu_ish(x):
    """The tanh approximation of GELU with rounded constants: a chain of
    eight elementwise operations."""
    return 0.5 * x * (1 + jnp.tanh(0.8 * (x + 0.04 * x**3)))

print(d2l.Benchmark(lambda: x + 1, desc='1 op'))
print(d2l.Benchmark(lambda: gelu_ish(x), desc='chain'))  # No jit: 8 kernels
```

```
1 op: 0.61 ms/call
chain: 5.30 ms/call
```

予測どおり、融合されていない演算連鎖には `x + 1` の 8× から 9× の時間がかかる。**カーネル融合**（kernel fusion、13.3 節）は、この連鎖を `x` を 1 回だけ読み出して出力を 1 回だけ書き込む単一のカーネルへと統合する。

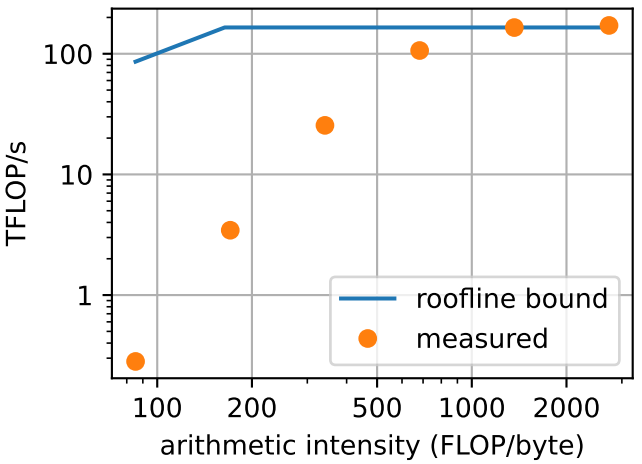
13.1.3.3 サイズ別の行列積

正方行列の bf16 行列積は、$I = n/3$ が $n = 500$ 付近で 164 を超えた途端に計算ルーフへ到達するのだろうか。n を 256 から 8,192 までスイープして確かめてみよう。

```
sizes = [256, 512, 1024, 2048, 4096, 8192]
times = []
for n in sizes:
    a = jax.random.normal(d2l.get_key(), (n, n), dtype=jnp.bfloat16)
    times.append(d2l.Benchmark(lambda: matmul(a, a), repeats=100).time)
```

達成された TFLOP/s を演算密度 $n/3$ に対して、ルーフライン上界 Equation 13.2 と並べてプロットする。

```
measured = [2 * n**3 / t / 1e12 for n, t in zip(sizes, times)]
intensity = [n / 3 for n in sizes]  # FLOP/byte of a square bf16 product
ridge = peak_tflops / bandwidth_tbs
roof_x = [intensity[0], ridge, intensity[-1]]
roof_y = [min(peak_tflops, bandwidth_tbs * i) for i in roof_x]
d2l.plot([roof_x, intensity], [roof_y, measured],
         'arithmetic intensity (FLOP/byte)', 'TFLOP/s',
         legend=['roofline bound', 'measured'],
         xscale='log', yscale='log', fmts=('-', 'o'))
```



呼び出し 1 回あたりの正確なレイテンシ `t`（マイクロ秒単位）とスループット `tf` を行列サイズ `n` ごとに出力すると、小さい n における一定時間の下限が明確になる。

```
for n, t, tf in zip(sizes, times, measured):
    print(f'n = {n:4d}: {1e6 * t:7.1f} us per call, {tf:5.1f} TFLOP/s')
```

```
n =  256:   119.1 us per call,   0.3 TFLOP/s
n =  512:    77.9 us per call,   3.4 TFLOP/s
n = 1024:    84.2 us per call,  25.5 TFLOP/s
n = 2048:   161.5 us per call, 106.4 TFLOP/s
```

```
n = 4096:   833.6 us per call, 164.9 TFLOP/s
n = 8192:  6404.3 us per call, 171.7 TFLOP/s
```

$n = 4096$ と 8192 では、行列積はクロック変動幅である 5% の範囲内で 165 TFLOP/s のルーフに達している。しかし小さいサイズでは、測定点はルーフラインを大きく下回り、両対数傾き 3 で上昇している。これは演算量が n^3 でスケールする一方で、呼び出し 1 回あたりの時間が一定であることの特徴である。$n = 256$ から 1024 にかけて演算量は $64\times$ に増えるにもかかわらず、呼び出しあたりの時間は $2\times$ 未満しか変化しない。

この一定時間がホストとデバイスのどちらで費やされているかを確かめるため、256×256 の行列積と同じ形状の `x + 1` の実行時間を計測し、さらにプロファイラ用ヘルパ `d2l.kernel_times`（後続の 13.1.4 節で構築する）を用いて呼び出し 1 回あたりに GPU 自体が稼働している時間を計測する。

```
add_one = jax.jit(lambda x: x + 1)
a = jax.random.normal(d2l.get_key(), (256, 256), dtype=jnp.bfloat16)
for desc, f in [('256 x 256 product', lambda: matmul(a, a)),
                ('x + 1 on 256 x 256', lambda: add_one(a))]:
    wall = d2l.Benchmark(f, repeats=100).time
    busy = sum(d2l.kernel_times(f, steps=100).values()) / 1e3  # Seconds
    print(f'{desc:18s}: {1e6 * wall:5.1f} us per call, '
          f'device busy {1e6 * busy:4.1f} us')
```

```
256 x 256 product :  58.7 us per call, device busy  4.2 us
x + 1 on 256 x 256:  61.2 us per call, device busy  1.1 us
```

いずれの場合も GPU が実際に稼働しているのはわずか数マイクロ秒であり、残りはホストディスパッチ時間 h（Python、フレームワーク、ドライバ）である。256×256 の行列積では、演算の中で最も高コストな部分はそれを要求する Python の 1 行なのである。呼び出しが立て続けにキューへ入れられると、ホストとデバイスはオーバーラップし、各呼び出しの所要時間は $\max(h, 2n^3/P)$ となる。小さい n ではホストが次の呼び出しを発行する前にデバイスが処理を終えてしまうため、プログラムは**オーバーヘッド律速**（overhead-bound）のレジームに入る。計算時間がディスパッチ時間を上回る屈曲点は $n \approx (hP/2)^{1/3}$ 付近にある。

単一演算のディスパッチには $h \approx 50$–100 µs かかるため（JIT コンパイルされた訓練ステップ全体に対して 1 回だけ支払われる）、屈曲点は $n = 1{,}600$–$2{,}000$ 付近にある。

これら 3 つの下限をすべてまとめると、ディスパッチされる単一演算に対する基本的な時間下界が得られる。

$$t \gtrsim \max\left(h,\ \frac{\text{FLOPs}}{P},\ \frac{\text{bytes}}{\beta}\right). \tag{13.3}$$

$h = 10$ µs とすると、演算量が $hP \approx 1.6 \times 10^9$ FLOPs 未満かつデータ移動量が $h\beta \approx 10$ MB 未満のあらゆる演算は、その起動時間だけで決まる。さらに、屈曲点 $n \approx (hP/2)^{1/3}$ は P とともに大きくなるため、より高速な GPU ほどオーバーヘッド律速を避けるためにより大きな演算を必要とする。

13.1.3.4 診断と対策

訓練ステップ全体では 3 つのレジームが混在し、最も大きな割合を占めるものがそのステップのレジームを決める（Figure 13.2）。Table 13.1 に、各レジームの症状と標準的な対策をまとめる。

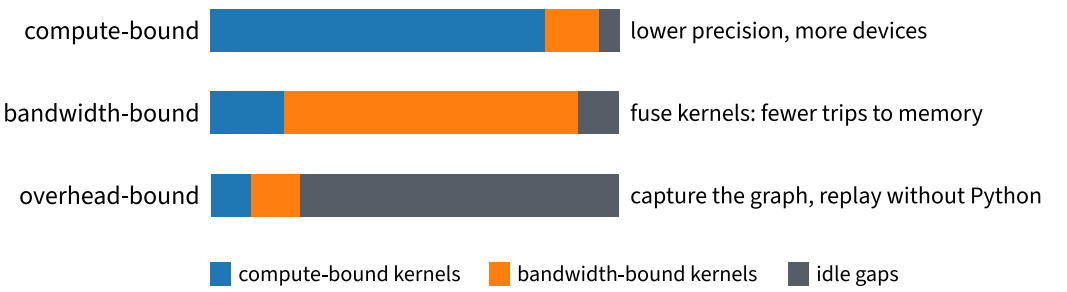


Figure 13.2: 3 つのレジームの模式図。1 ステップの時間が計算律速なカーネル、帯域幅律速なカーネル、アイドル時間の隙間のどこに費やされているかによって適用すべき対策が決まる。各バーには代表的な対策を 1 つずつ示してある。

Table 13.1: 3 つのレジームの症状と標準的な対策。

レジーム	症状	標準的な対策
計算律速	デバイスが稼働中、演算密度がリッジポイント超、時間が FLOPs に応じて増加	算術演算量の削減、より高速なテンソルコアを備えた数値形式、デバイス数の追加
帯域幅律速	デバイスが稼働中、演算密度がリッジポイント未満、時間が移動バイト数に応じて増加	行列積の演算密度を高めるバッチ拡大、より狭い数値形式、カーネル融合、保存の代わりの再計算、チップ上でデータを再利用するアルゴリズム
オーバーヘッド律速	カーネル間でデバイスがアイドル状態、呼び出しあたりの時間がサイズに非依存	例あたりの呼び出し回数を減らすバッチ拡大、プログラム全体のキャプチャと Python を介さないリプレイ

13.1.4 訓練ステップのプロファイリング

プロファイラはすべての GPU カーネルの所要時間を記録するため、デバイスの稼働時間を実経過時間と比較したり、デバイス時間をカーネルの種類ごとに内訳表示したりできる。プロファイリングを行う前に、1,024 個の例からなるミニバッチに対する MLP（$4096 \to 4096 \to 4096 \to 10$）の fp32（tf32 行列積を使用、$P = 82.6$ TFLOP/s）における SGD 1 ステップの実行時間を予測してみよう。

13.1.4.1 ステップ時間の予測

2 つの 4096×4096 層はそれぞれ、順伝播の行列積 1 回あたり $2 \cdot 1024 \cdot 4096^2 \approx 3.44 \times 10^{10}$ FLOPs を実行する。1 回の訓練ステップではこのような行列積が 5 回実行される（順伝播で 2 回、重み勾配で 2 回、2 番目の隠れ層に対する入力勾配で 1 回）。$I \approx 340$ FLOP/byte（tf32 のリッジポイントである 82 を上回る）であるため、各行列積には $3.44 \times 10^{10} / (82.6 \times 10^{12}) \approx 0.42$ ms かかる。SGD の更新は 3,360 万個の fp32 重みとその勾配をそれぞれ読み出して新しい重みを書き込むため（$12 \times 33.6 \times 10^6 \approx 0.40$ GB）、1 TB/s で約 0.40 ms かかる。小さな最終層と ReLU を無視すると、このステップの所要時間は次のようになるはずである。

$$5 \times 0.42\ \text{ms} + 0.40\ \text{ms} \;\approx\; 2.5\ \text{ms}.$$

```
def mlp(params, X):
    for W, b in params[:-1]:
        X = jax.nn.relu(X @ W + b)
    W, b = params[-1]
    return X @ W + b

@jax.jit
def step(params, X):
    grads = jax.grad(lambda p: mlp(p, X).mean())(params)
    return jax.tree.map(lambda p, g: p - 0.01 * g, params, grads)

widths = [4096, 4096, 4096, 10]
params = [(jax.random.normal(d2l.get_key(), (m, n)) / m**0.5, jnp.zeros(n))
          for m, n in zip(widths[:-1], widths[1:])]
X = jax.random.normal(d2l.get_key(), (1024, 4096))
wall = d2l.Benchmark(lambda: step(params, X)).time
```

kernel_times はフレームワークのプロファイラの下で関数を steps 回実行し、起動されたすべてのカーネルについて呼び出し 1 回あたりのデバイス上の平均ミリ秒を返す。

```
import glob
import tempfile

def kernel_times(f, steps=10):
    """Device milliseconds per call of f, for every kernel that f runs."""
    kernels = collections.Counter()
    with tempfile.TemporaryDirectory() as trace_dir:  # Removed afterwards
        with jax.profiler.trace(trace_dir):
            for _ in range(steps):
                out = f()
            jax.block_until_ready(out)
        xspace = glob.glob(f'{trace_dir}/**/*.xplane.pb', recursive=True)[0]
        gpu = jax.profiler.ProfileData.from_file(
            xspace).find_plane_with_name('/device:GPU:0')
        for line in gpu.lines:
            if line.name.startswith('Stream'):  # One line per CUDA stream
                for e in line.events:
                    kernels[e.name] += e.duration_ns / 1e6 / steps
    return kernels
```

フレームワークのプロファイラは、https://ui.perfetto.dev で閲覧可能なタイムライントレースをエクスポートすることもできる。

jax.profiler.trace は、トレースディレクトリにトレースを書き出す。

13.1.4.2 プロファイルの読み方

show_profile はデバイスの稼働時間をステップの実経過時間と比較し、行列積カーネルが占める割合を計算して、上位のカーネルを一覧表示する。

```
PRODUCT_KERNELS = 'gemm|xmma|cutlass|matmul|sdpa'
# Kernel names that mark a matrix product: cuBLAS and CUTLASS kernels (gemm,
# xmma, cutlass, which also names PyTorch's fused attention), XLA's fused
# products (gemm_fusion_dot), and matmul or sdpa kernels

def show_profile(kernels, step_ms, rows=6):
    """Busy device time against the wall-clock time step_ms of a step, the
    share of the matrix products, and the kernels with the most time."""
    busy = sum(kernels.values())
    products = sum(ms for name, ms in kernels.items()
                   if re.search(PRODUCT_KERNELS, name, re.I))
    print(f'wall clock {step_ms:.2f} ms per step, device busy {busy:.2f} ms '
          f'({busy / step_ms:.0%}), matrix products {products / busy:.0%}')
    for name, ms in sorted(kernels.items(), key=lambda kv: -kv[1])[:rows]:
        name = re.sub(r'^void |\(anonymous namespace\)::|\w+::|\(.*', '', name)
        print(f'{ms:6.2f} ms {ms / busy:4.0%}  {name[:52]}')
```

ここで show_profile を 3 層 MLP のステップのカーネル時間計測結果（kernel_times(step)）に適用し、デバイスの稼働時間を実測された実経過時間 1000 * wall と比較する。

```
show_profile(kernel_times(lambda: step(params, X)), 1000 * wall)
```

```
wall clock 2.41 ms per step, device busy 2.37 ms (99%), matrix products 97%
  0.51 ms  22%  gemm_fusion_dot_1
  0.50 ms  21%  gemm_fusion_dot
  0.44 ms  19%  gemm_fusion_dot_general_8
  0.42 ms  18%  Kernel2<cutlass_80_tensorop_s1688gemm_128x256_16x3_n
  0.42 ms  18%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_16x5_t
  0.06 ms   3%  input_reduce_select_fusion_1
```

実測されたステップ時間は予測値 2.5 ms の約 10% 以内に収まっており、ステップのほぼ全時間にわたって GPU が稼働している。行列積（**gemm** カーネル）がデバイス時間の 3/4 以上を占めている。

XLA は SGD の更新を重み勾配の行列積（gemm_fusion_dot）に直接融合し、更新後の重みを書き込む前にスケールされた勾配を減算するため、勾配が個別にメモリとの間を往復することはない。

13.1.5 まとめ

GPU 演算の所要時間は $t \gtrsim \max(h,\ \text{FLOPs}/P,\ \text{bytes}/\beta)$ によって下から抑えられる。ここで、h はホストディスパッチのオーバーヘッド、P はピーク算術演算スループット、β はメモリ帯域幅である。演算密度 $I = \text{FLOPs}/\text{bytes}$ は後半の 2 つを比較するものであり、リッジポイント $I^* = P/\beta$（RTX 4090 の bf16 では約 164 FLOP/byte）未満では演算は帯域幅律速となり、それを超えると計算律速となる。線形層では、演算密度はバッチサイズとともにスケールし、単一トークンのデコーディング時の約 1 FLOP/byte から訓練時の数百 FLOP/byte まで変化する。正確なベンチマークにはループ外でのデバイス同期とウォームアップ実行（d2l.Benchmark）が不可欠であり、d2l.kernel_times と d2l.show_profile を用いれば机上の予測をカーネルごとに検証できる。

13.1.6 演習問題

1. **最小の次元。** Equation 13.1 における積 **XW** の演算密度の厳密な式（$B \ll D, F$ の近似を行う前のもの）は、あらゆる形状に対して成り立つ。$m = \min(B, D, F)$ とおく。
   1. 演算密度が $2m/(3b)$ と $2m/b$ の間にあることを示せ。ここから、リッジポイントが 164 FLOP/byte の GPU 上で bf16 を用いる場合、164 未満の次元をもつすべての行列積はルーフラインモデルにおいて帯域幅律速となることを結論づけよ。
   2. この評価式を 3 つの bf16 行列積に適用せよ。すなわち、13.1.4 節の MLP の最終層（$4096 \to 10$、サイズ 4,096 のバッチ）、1 つのヘッドのアテンションスコア $\mathbf{QK}^\top$（ただし $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{T \times 64}$）、および 64 チャネルから 64 チャネルへの 1×1 畳み込み（6.4.3 節、画像 128 枚のバッチで各画像は 32×32 ピクセル）である。それぞれがリッジポイントに到達するためには、どの次元が大きくなる必要があるか。
   3. マルチヘッドアテンション層は、すべてのヘッドのスコアを 1 つのバッチ化されたカーネルで計算する。ヘッドをバッチ化するとスコアの演算密度は変化するか。
2. [code] **バッチサイズとリッジポイント。** bf16 における積 **XW**（ただし $\mathbf{X} \in \mathbb{R}^{B \times 8192}$、$\mathbf{W} \in \mathbb{R}^{8192 \times 8192}$）を考えよ。
   1. Equation 13.1 の厳密な式を用いて、この行列積が手元の GPU のリッジポイントに到達するバッチサイズを求め、小さい B の近似と比較せよ。
   2. Equation 13.3 から、同時にデコーディングする系列数が $B = 1$ と $B = 64$ の場合についてこの行列積の所要時間を予測し、両方を `d2l.Benchmark` で実測したうえで、ステップあたりの追加時間がほとんどかからずに系列数を増やせるのはどのバッチサイズまでか求めよ。
3. [code] **隠れたバリア。** 以下の各行が訓練ループのステップごとに 1 回実行されるとする。ただし、`x` と `g` は GPU 配列、`c` は Python の浮動小数点数である。対象とする行は `x[x > 0].sum()`、`print(x.shape)`、`jnp.argmax(x)`、`if jnp.linalg.norm(g) > c: g = g * c / jnp.linalg.norm(g)`、および `x * jnp.asarray(0.5)` である。
   1. どの行がホストにデバイスの完了を待たせるかを判定し、それぞれの理由を述べよ。
   2. 同期を発生させるすべての行を書き換え、ホストバリアなしに同じ結果を計算するようにせよ。
   3. `with jax.transfer_guard_device_to_host('disallow'):` の内側で実行して解答を確認せよ。
4. **計測時間からの診断。** RTX 4090（$P = 165$ TFLOP/s、$\beta = 1.008$ TB/s）上の 4 つの bf16 演算について、`d2l.Benchmark` が呼び出し 1 回あたり以下の時間を報告したとする。演算 A は 2 つの 8192×8192 行列の積を 7.4 ms で計算し、演算 B は 8192×8192 行列に行ごとのソフトマックスを 0.30 ms で適用し（要素あたり約 5 FLOPs、読み出し 1 回と書き込み 1 回）、演算 C は 2 つの 256×256 行列の積を 10 µs で計算し、演算 D は 2 つの 1536×1536 行列の積を 0.10 ms で計算する。
   1. 各演算について、演算密度ならびに達成している P と β に対する割合を計算せよ。リッジポイントに対する演算密度の大小と、近づいているルーフのそれぞれによって各演算を分類せよ。2 つの分類が食い違うのはどの演算か。
   2. どちらのルーフにも近くない各演算について、ホストディスパッチレイテンシ h が原因である場合と、すべてのストリーミングマルチプロセッサを占有するにはスレッドブロックのタイル数が不足している場合とを区別し、プロファイラのタイムラインがそれぞれどのように見えるかを述べよ。
5. [code] **ディスパッチのパイプラインモデル。●**1 回の GPU 呼び出しを、ホストエンキュー時間 h、デバイス実行時間 t_{dev}、および最後のバリアレイテンシ σ という 3 つの所要時間でモデル化せよ。
   1. 立て続けにエンキューされた R 回の呼び出しの後に 1 回のバリアを置くと、所要時間が $R \max(h, t_{\text{dev}}) + \min(h, t_{\text{dev}}) + \sigma$ になることを示せ。$t_{\text{dev}} < h$ のとき、`d2l.Benchmark` は何を報告するか。

   2. 13.1.2 節のループについて、毎ステップ読み出すループの所要時間と最後に 1 回だけ読み出すループの所要時間の比を、h、t_{dev}、σ を用いて導出せよ。
   3. 行列の辺の長さ $256, 512, 1024, 2048, 4096$ に対して両方のループを実行して h、σ、t_{dev} を推定し、ステップあたりのデバイス時間がいくらを超えれば毎ステップのスカラー読み出しのコストが 10% 未満になるかを求めよ。
6. [code] **タイルとウェーブ**。●GPU は n_{SM} 個のストリーミングマルチプロセッサ（SM）から構成される（RTX 4090 では $n_{\text{SM}} = 128$）。行列積カーネルが $n \times n$ の出力を 128×128 のタイルに分割し、各 SM が一度に 1 つのタイルを速度 P/n_{SM} で計算し、タイルは最大 n_{SM} 個ずつのウェーブ単位で実行されると仮定せよ。
   1. このモデルで達成可能なピークスループットに対する割合を n と n_{SM} の関数として導出し、13.1.3.3 節でのスイープ実測値と比較せよ。
   2. $n = 2048, 2176, 2304, 2432, 2560$（128 の連続する倍数）についてピークに対する割合を予測・実測し、実測値が単純なウェーブモデルから乖離する箇所を説明せよ。
   3. $n = 512$ の行列積をより多くのワークアイテムに分割してすべての SM を参加させる方法を 2 つ挙げ、それぞれが追加するオーバーヘッドを述べよ。

13.2 ハードウェア

前節では、GPU をピーク演算速度 P とメモリ帯域幅 β という 2 つの定数で特徴づけ、1 バイトあたり I FLOPs を実行する演算の速度が高々 $\min(P, I\beta)$ に抑えられることを示した。RTX 4090 では、bf16 の行列積に対して $P = 165$ TFLOP/s、$\beta \approx 1$ TB/s である。つまり、このカードがメモリから 1 バイトを読み出す間に、165 回の浮動小数点演算を実行できる計算になる。これら 2 つの数値はどこから来るのだろうか。また、なぜ 1 バイトの移動には 1 FLOP の演算よりはるかに長い時間がかかるのだろうか。

1B パラメータの GPT を用いて、テキストを 1 トークンずつ生成する場合を考えよう（11.3 節）。bf16 では重みが 2 GB を占め、トークンを 1 つ生成するたびにそのすべてを読み出す必要がある。1 トークン分の算術演算（2×10^9 FLOPs）はテンソルコアで 12 µs で終わる一方、実効帯域幅 0.9 TB/s で 2 GB の重みを読み出すには 2.2 ms かかる。したがって、単一ストリームでのデコード速度は高々約 450 tokens/s にとどまり、演算ユニットは 99% 以上の時間を遊休状態で過ごすことになる。

この隔たりは 3 つの物理的事実によって説明できる。第 1 にデータはチップ間の境界を越えなければならず、第 2 にオフチップ配線はオンチップのトランジスタよりも微細化に伴うスケーリングが遅く、第 3 に 1 バイトのデータ移動はそれを用いた演算よりも何桁も多くのエネルギーを消費する。これらが相まって、リッジポイント P/β が数百に達し、ハードウェアの世代を追うごとに上昇してきた理由が説明される。本節では具体例として、本書のノートブックを実行しているワークステーション（RTX 4090 GPU 4 基搭載、2026 年後半時点）を用いる。なお、コンピュータアーキテクチャの詳細については Hennessy and Patterson (2011) が深く扱っており、ハードウェアの選定については 39.4 節で議論する。

13.2.1 メモリ階層

プログラムが扱うすべてのバイトは、容量と速度をトレードオフするメモリ階層（memory hierarchy）のどこかに置かれている（Figure 13.3）。オンダイのレジスタと SRAM キャッシュは約 0.1 GB を保持し、数十 TB/s の帯域幅を提供する。GPU のオフダイ DRAM、すなわちデータセンタ向けカードの**広帯域メモリ**（high-bandwidth memory、HBM）やコンシューマ向けカードの GDDR は、数十〜数百 GB を保持し、帯域幅は 1〜8 TB/s である。ホスト DRAM は CPU ソケットあたり約 0.5 TB/s で数百 GB を保持し、NVMe フラッシュは約 14 GB/s で数 TB を保持し、そのさらに外側にはネットワークストレージが位置する。

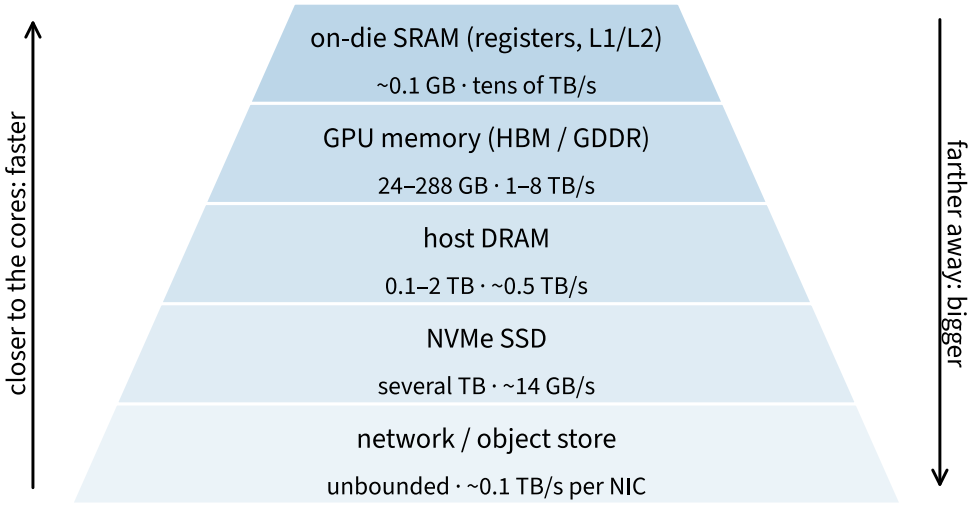


Figure 13.3: 2026 年後半時点のメモリ階層。下の階層へ行くほど保持できるバイト数は何桁も増えるが、毎秒転送できるバイト数は何桁も減る。

13.2.1.1 帯域幅とレイテンシ

Figure 13.4 は、訓練ジョブが利用するデータ経路を 14 GB/s（NVMe）から 8 TB/s（NVIDIA B200 の HBM）まで、ほぼ 3 桁にわたって並べたものである。どの段階の差も 1 つの法則で説明できる。すなわち、**帯域幅はデータ配線の本数と 1 本あたりのビットレートの積に等しい**。RTX 4090 は 384 本の配線を用いて各 21 Gb/s（合計約 1 TB/s）で GDDR6X メモリにアクセスするのに対し、PCIe 4.0 ×16 スロットは片方向あたり 16 レーン、各 16 Gb/s（合計 32 GB/s）しかもたない。配線 1 本あたりの速度はほぼ同じだが、PCIe 側は配線数が 24 分の 1（24× の差）しかないのである。

配線の本数を決めるのは距離である。HBM スタックは GPU パッケージ内部でダイから数ミリメートルの位置に置かれ、数千本の短い配線で接続されている。これに対し、基板配線で扱えるのは数百本、スロットやケーブルでは数十レーンにとどまる。**ダイから 1 段階離れるごとに配線の本数が、ひいては帯域幅が約 1 桁ずつ減少する**。B200 は HBM を 8 TB/s で読み出し、他の GPU には NVLink 5 経由の 0.9 TB/s（各 200 Gb/s の約 36 レーンを使用）で接続し、ホストには PCIe 5.0 経由の 64 GB/s で接続する。

帯域幅が定常状態での流量を支配するのに対し、**レイテンシ**は最初の 1 バイトが到着するまでの時間である（Figure 13.5、RTX 4090 のマイクロベンチマークは Luo et al. (2024) による）。Python からディスパッチされる 1 回の Eager 演算には 10〜100 µs のホスト時間がかかる（そのうちハードウェアのカーネル起動は 1〜5 µs である）。10 µs あれば RTX 4090 は 10 MB をストリーム転送できるため、小さな演算ではメモリや計算ではなくディスパッチのオーバーヘッドが支配的になる。

レイテンシと帯域幅は**アクセスパターン**を通じて相互作用する。DRAM が定格の帯域幅を発揮できるのは、連続したアドレスにアクセスする場合だけである。ここで 1 GiB の fp32 ベクトル `x[idx]` をギャザーする実験として、`idx` を恒等置換にした場合とランダム置換にした場合をそれぞれ試してみよう。GPU メモリは 32 バイトの**セクタ**単位でデータを転送するため、ランダムな 4 バイトの読み出しは毎回 32 バイトのセクタ全体を取得することになる一方、インデックスと出力のベクトルは連続的にストリーム転送される（全体として移動するバイト数は約 3× になる）。

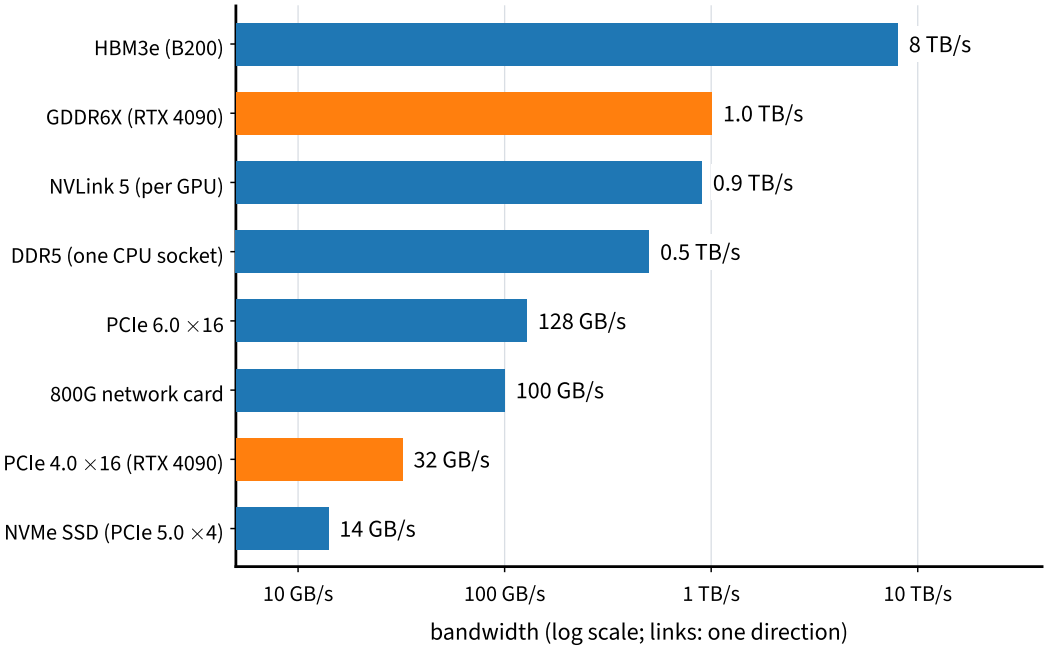


Figure 13.4: 訓練ジョブが使用するデータ経路の帯域幅（2026 年後半時点）。リンクは片方向の値である。本書のマシンのメモリとホストリンクをオレンジ色で示す。

```
n = 256 * 1024 * 1024
x = jax.random.normal(d2l.get_key(), (n,))  # 1 GiB fp32
gather = jax.jit(lambda x, idx: x[idx])
times = {}
for desc in ('identity', 'random'):
    idx = (jnp.arange(n) if desc == 'identity' else
           jax.random.permutation(d2l.get_key(), n))
    # Bytes per element: the index and the output stream; x is read as
    # 4 bytes in order, or as a whole 32-byte sector at a random address
    nbytes = idx.dtype.itemsize + 4 + (4 if desc == 'identity' else 32)
    times[desc] = d2l.Benchmark(lambda: gather(x, idx)).time
    print(f'{desc:8s}: {1e3 * times[desc]:5.1f} ms, '
          f'{nbytes * n / times[desc] / 1e12:.2f} TB/s at {nbytes} B/element')
print(f'random / identity: {times["random"] / times["identity"]:.1f}x')
```

```
identity:   3.5 ms, 0.92 TB/s at 12 B/element
random   :  36.6 ms, 0.29 TB/s at 40 B/element
random / identity: 10.5x
```

恒等置換によるギャザーが約 0.9 TB/s でストリーム処理されるのに対し、ランダムなギャザーは約 37 ms かかり（8–11× 遅い）、1 回の読み出しあたり 32 バイトのセクタ全体を転送量に算入してもピーク帯域幅の 3 分の 1 しか達成できない。32 バイトのセクタという粒度に加え、DRAM は数 KB 規模の開いた**行**（row）からデータを供給する仕組みになっている。連続するアドレスは開いた行を共有できるが、ランダムなアドレスは新しい行を開くレイテンシを繰り返し支払ううえ、アドレス変換キャッシュのミスも引き起こす。

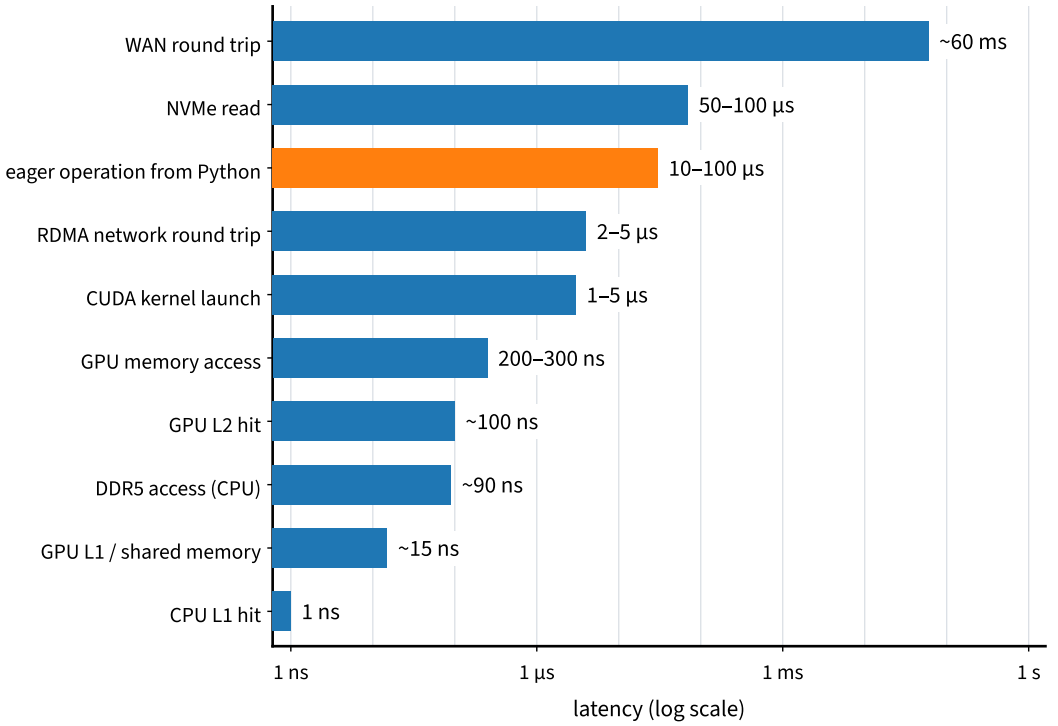


Figure 13.5: 対数スケールで示したレイテンシ。強調表示された項目は、Python からディスパッチされるフレームワーク演算 1 回あたりのホスト時間であり、カーネル起動はその一部を占める。

密行列の積や埋め込み参照（1 トークンあたり数千バイトの連続する行を読み出す）は効率よくストリーム転送できる。対照的に、細粒度のランダムアクセス（疎行列、グラフ探索、ハッシュテーブル）は β のごく一部しか出せない。より一般に、n バイトの連続ブロックを立ち上がりレイテンシ α で転送するには $t(n) = \alpha + n/\beta$ の時間がかかり、$n_{1/2} = \alpha\beta$ でピーク帯域幅の半分に達する (Hockney 1994)。NVMe ドライブ（$\alpha \approx 50$–$100\,\mu$s、$\beta = 14$ GB/s）では $n_{1/2} \approx 1$ MB となるため、ディスク上のデータセットは個々のデータ例ごとではなく MB 規模のシャード単位でシャッフルされる。

13.2.2 演算性能とメモリ帯域幅

13.2.2.1 面積と外周

演算ユニットがシリコンダイの内部を埋め尽くす（面積に比例し $O(n^2)$ でスケールする）のに対し、I/O パッドは外周、すなわち**ショアライン**に沿って並ぶ（辺の長さに比例し $O(n)$ でスケールする。Figure 13.6）。トランジスタの微細化も同じ効果をもたらす。最小加工寸法が半分になると論理回路の密度は 4 倍になるが、ダイ端におけるはんだバンプのピッチははるかに緩やかにしか縮小しない。

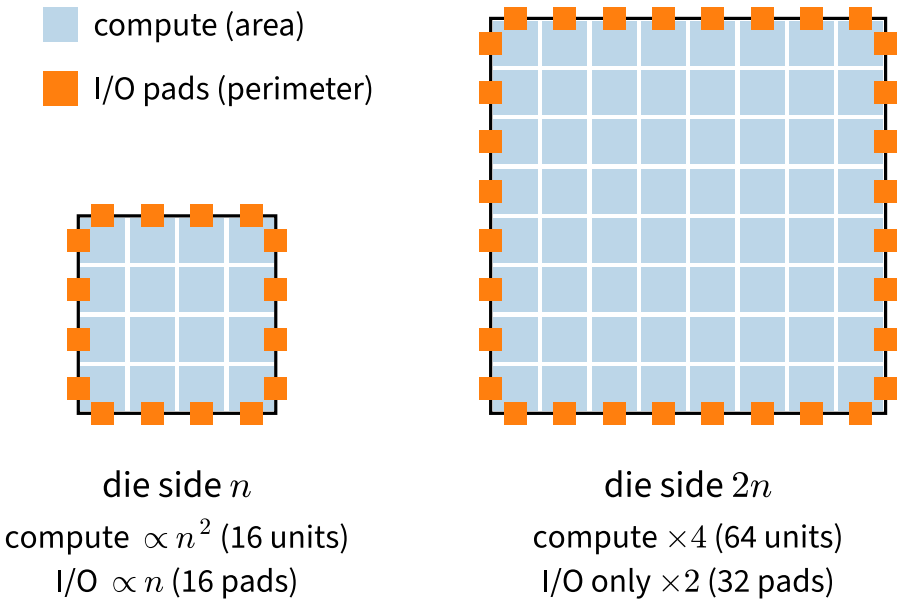


Figure 13.6: 演算は面積、I/O は外周で決まる。ダイの一辺を 2 倍にすると演算能力は 4 倍になるが、I/O のための余地は 2 倍にしかならない。

アーキテクトは、このショアラインのボトルネックを 2 つの方法で緩和している。

1. **シリコンインターポーザ上の HBM**：GPU ダイの隣に DRAM ダイを垂直に積層することで、数百本の基板配線を数千本の短く高密度なインターポーザ配線に置き換える（B200 では 8 TB/s に達するのに対し、CPU ソケットでは約 0.5 TB/s にとどまる）。
2. **マルチダイパッケージ**：リソグラフィの制約により単一ダイの面積は**レチクル限界**（reticle limit、約 $858\,\mathrm{mm}^2$）で頭打ちになるため、B200 のようなフラッグシップのアクセラレー

タはレチクル限界サイズのダイ 2 枚を 1 つのパッケージ上で結合し、外周を 2 倍にして 8 個の HBM スタックを収容している。

13.2.2.2 世代間のトレンド

V100（2017 年）から A100、H100 を経て B200（2024 年）に至る NVIDIA のデータセンタ向け世代を通じて、密な 16 ビット演算の FLOP/s は 1 世代あたり約 2.6×、メモリ帯域幅は約 2.0×、メモリ容量は約 1.8× のペースで拡大した (Gholami et al. 2024)。その結果、リッジポイントは V100 の 139 FLOP/byte から H100 の 295 FLOP/byte（B200 では 281 FLOP/byte）へと倍増した。したがって、演算密度（arithmetic intensity）が 200 FLOP/byte の演算は、V100 では計算律速であったが、H100 や B200 では帯域幅律速となる。

13.2.2.3 演算あたりのエネルギー

消費電力もまた、帯域幅に対する第 2 の厳格な上限を定める。45 nm CMOS プロセス（Figure 13.7、Horowitz (2014)）では、fp32 の乗算 1 回が 3.7 pJ、8 ビット整数の加算 1 回が 0.03 pJ であるのに対し、オフチップ DRAM から 64 ビットを読み出すには 1.3〜2.6 nJ、すなわちおよそ **500 回の fp32 乗算**に相当するエネルギーを要する。オンチップ SRAM の読み出しはその中間に位置する（距離に応じて 10〜100 pJ）。現代の微細化されたプロセスノードではすべての数値が縮小するものの、この階層関係自体は変わらない。データを移動させることは、データに対して演算するよりもはるかに多くのエネルギーを消費するのである。カーネル融合、狭い数値形式、および活性化値の再計算が電力を節約するのは、それらが時間を短縮するのとまったく同じ物理的理由による。

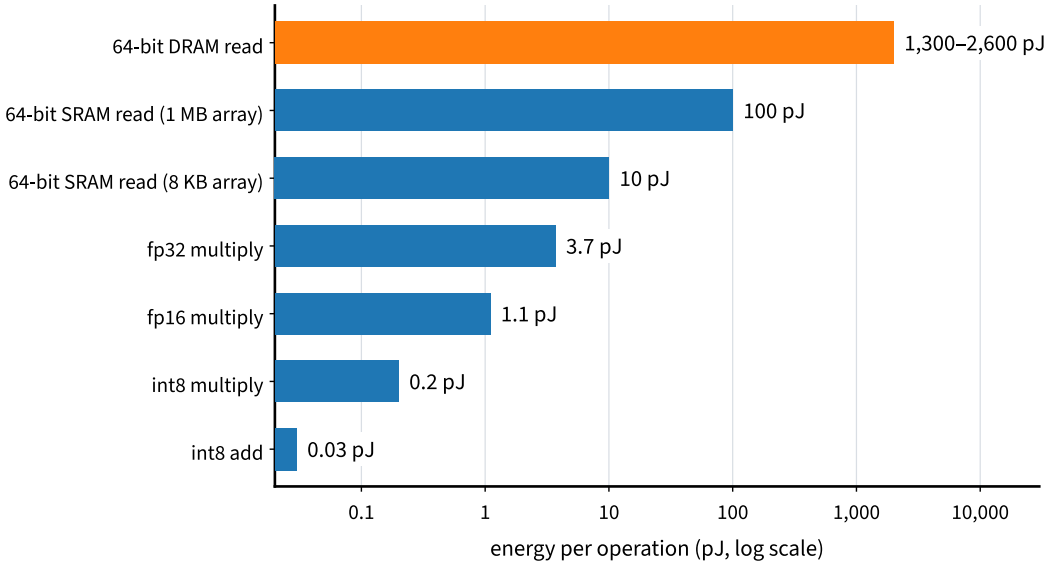


Figure 13.7: 45 nm プロセスにおける演算あたりのエネルギー。64 ビットの DRAM 読み出し 1 回は、fp32 の乗算約 500 回分に相当するエネルギーを消費する。

13.2.3 GPU

13.2.3.1 ストリーミングマルチプロセッサ

GPU はダイ面積の大半を数千個の演算ユニットに割り当てており、それらは**ストリーミングマルチプロセッサ**（SM、RTX 4090 では 128 基）へとまとめられている。各 SM は**ワープ**と呼ばれる

32 個のグループ単位でスレッドをロックステップ実行する。巨大なキャッシュに頼って DRAM のレイテンシを隠蔽するかわりに、SM は毎サイクル実行するワープを切り替え、メモリ待ちでストールしたワープがある間も別のワープが演算を続けられるようにする。

リトルの法則 (Little 1961) により、帯域幅 $\beta = 1.0$ TB/s のメモリパイプを 250 ns のレイテンシにわたって満たし続けるには、常時 $1.0\,\text{TB/s} \times 250\,\text{ns} \approx 250$ KB がインフライト（未完了の転送中）でなければならない（SM あたり約 2 KB、すなわち SM あたり 16 個のワープがそれぞれ 1 スレッドにつき 4 バイトのロードを待っている状態に相当する）。250 KB をインフライトに保てるだけの並列性を引き出せないカーネルは、帯域幅律速ではなくレイテンシ律速となる。

13.2.3.2 テンソルコアと数値形式

テンソルコアは小さな行列タイルの乗算と累積を単一命令で実行し、TPU のシストリックアレイと同様に、命令のフェッチとデコードをタイル全体にわたって償却する (Kung 1988; Jouppi et al. 2017)。H100 では、bf16 テンソルコアが 989 TFLOP/s を達成するのに対し、スカラー fp32 は 67 TFLOP/s であり（$15\times$ の差）、コンシューマ向けの RTX 4090 ではこの比が $2\times$ となる。

RTX 4090 のピークスループットは、仕様から直接導出できる。128 基の SM $\times$ 128 個の fp32 演算器 $= 16{,}384$ 個の演算器があり、それぞれが 2.52 GHz のブーストクロックで毎サイクル 1 回の融合積和演算（2 FLOPs）を実行するから、次のようになる。

$$P_{\text{fp32}} = 16{,}384 \times 2 \times 2.52 \times 10^{9}\ \text{s}^{-1} \approx 82.6\ \text{TFLOP/s}.$$

このカードのテンソルコアは、tf32 を同じ 82.6 TFLOP/s で実行し、bf16（fp32 累積あり）を $2\times = 165.2$ TFLOP/s で、fp8 を公称 $4\times = 330.4$ TFLOP/s で実行する。本章では tf32 と bf16 の速度を実測するが、fp8 の速度は本書のコードでは扱わない公称値である。

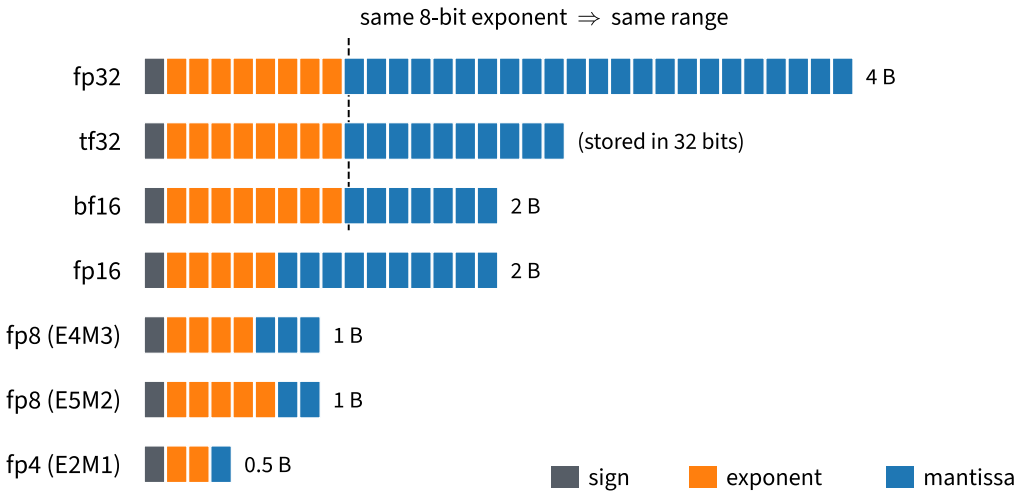


Figure 13.8: ビット単位で描いた浮動小数点形式。fp32、tf32、bf16 は 8 ビットの指数部を共有するため、表現範囲も同じである。tf32 はテンソルコアの演算モードでありメモリ上の値は 32 ビットで格納されるが、それ以外の形式では 1 段階狭まるごとに 1 値あたりのバイト数が半分になる。

Figure 13.8 が示すように、数値形式を狭めることにはルーフライン上で 2 つの効果がある。

1. **ビット幅を半分にすると移動バイト数が半減する**（帯域幅律速のカーネルが最大 $2\times$ 高速化され、重みのメモリ使用量も半減する）。ただし tf32 は例外であり、テンソルコア内部では 19 ビットの値（仮数部 10 ビット）で計算するものの、メモリ上には依然として 32 ビット浮動小数点数を格納する。
2. **ビット幅を半分にすると、通常はテンソルコアの計算ルーフ P が 2 倍になる。**

8 ビット以下の形式では明示的なスケール係数が必要になる。fp8 では、テンソル単位またはブロック単位のスケールを併用しつつ、順伝播の活性化値と重みに E4M3、勾配に E5M2 を用いる (Micikevicius et al. 2022)。4 ビット（fp4 / E2M1、-6 から 6 までの 15 通りの値しか符号化できない）になると、テンソル単位のスケーリングは外れ値に対して破綻するため、ハードウェアは 16 要素（NVFP4）または 32 要素（OCP MX）のブロックごとに 1 つの共有スケールを用いる**マイクロスケーリング**を適用する (Rouhani et al. 2023)。

13.2.3.3 アクセラレータの仕様

Table 13.2 に、2 つのデータセンタ向け GPU と 2 つのコンシューマ向けカードの比較を示す。

Table 13.2: 広く普及している 4 つの GPU。ブーストクロックにおける fp32 累積ありの密なテンソルコアスループットを TFLOP/s（TF）単位で示す。コンシューマ向けカード 2 機種の fp8 および fp4 の値は公称値であり、fp8 は bf16 の 2 倍、RTX 5090 の fp4 は実測値ではなくベンダの公称代表値に基づく。リンクは片方向の値である。

	H100 SXM	B200	RTX 4090（本機）	RTX 5090
メモリ容量 C	80 GB HBM3	180 GB HBM3e	24 GB GDDR6X	32 GB GDDR7
メモリ帯域幅 β	3.35 TB/s	8 TB/s	1.0 TB/s	1.8 TB/s
bf16	989 TF	2,250 TF	165 TF	210 TF
fp8	1,979 TF	4,500 TF	330 TF	419 TF
fp4	—	9,000 TF	—	1,676 TF
リッジポイント（bf16）	295 FLOP/B	281 FLOP/B	164 FLOP/B	117 FLOP/B
GPU 間リンク	NVLink 4、450 GB/s	NVLink 5、900 GB/s	PCIe 4.0 ×16（ホスト経由）、≤ 32 GB/s	PCIe 5.0 ×16（ホスト経由）、≤ 64 GB/s
消費電力	700 W	1,000 W	450 W	575 W

13.2.4 GPU 外のデータ経路

13.2.4.1 ホスト

算術演算を実行するのは GPU であるが、ホスト CPU も依然として訓練のボトルネックになり得る 3 つのタスクを担っている。すなわち、カーネルのディスパッチ（呼び出しあたり h）、データ読み込みと前処理パイプラインの実行、および PCIe を介したホストからデバイスへの転送の中継である。

GPU の DMA **コピーエンジン**が SM の演算と並行して PCIe 経由でデータを転送できるのは、転送元がページロック（**ピン留め**）ホストメモリにある場合だけである。通常の**ページャブル**メモリからの転送では、まず CPU が内部のピン留めステージングバッファへデータをコピーしなければならない。256 MiB の配列に対して、ページャブルメモリとピン留めメモリの転送速度を比較してみよう。

```
x_pageable = np.random.default_rng().standard_normal(
    64 * 1024 * 1024, dtype=np.float32)  # 256 MiB on the host
gpu = jax.devices()[0]
on_gpu = jax.sharding.SingleDeviceSharding(gpu, memory_kind='device')
x_pinned = jax.device_put(x_pageable, jax.sharding.SingleDeviceSharding(
    gpu, memory_kind='pinned_host'))
for desc, src in [('pageable', x_pageable), ('pinned', x_pinned)]:
    t = d2l.Benchmark(
        lambda: jax.device_put(src, on_gpu).block_until_ready()).time
    print(f'{desc}: {src.nbytes / t / 1e9:.1f} GB/s')
```

```
pageable: 8.3 GB/s
pinned: 23.3 GB/s
```

ピン留めメモリからの転送は約 23 GB/s に達する。これは PCIe 4.0 ×16 の理論上限である 32 GB/s の約 4 分の 3 であり、GPU メモリと比べると 40× 遅い。一方、ページャブルメモリからの転送はホストメモリ上での余分なコピーが加わるためさらに遅くなる。GPU メモリに収まるデータセットはデバイス上に常駐させるべきであり、ホストからストリーム供給するミニバッチにはピン留めホストメモリを用いて演算とオーバーラップさせるべきである。

`'pinned_host'` メモリ種別で配置された配列は、ピン留めメモリ上に置かれる。

13.2.4.2 GPU 間のリンク

データセンタ向け GPU は専用の NVLink ファブリック（片方向あたり H100 で 450 GB/s、B200 で 900 GB/s）を介して通信する。一方、コンシューマ向けの RTX 4090 カードは NVLink も PCIe のピアツーピア（P2P）アクセスも備えていない。Figure 13.9 に示すように、本書の 4 GPU ワークステーションでは GPU 間を行き来するすべてのバイトが一方の PCIe 4.0 リンクを上り、ホスト DRAM を経由して、もう一方のリンクを下らなければならないため、GPU 間転送はピン留めコピーの速度である約 23 GB/s（NVLink 5 より 40× 遅い）に制限される。

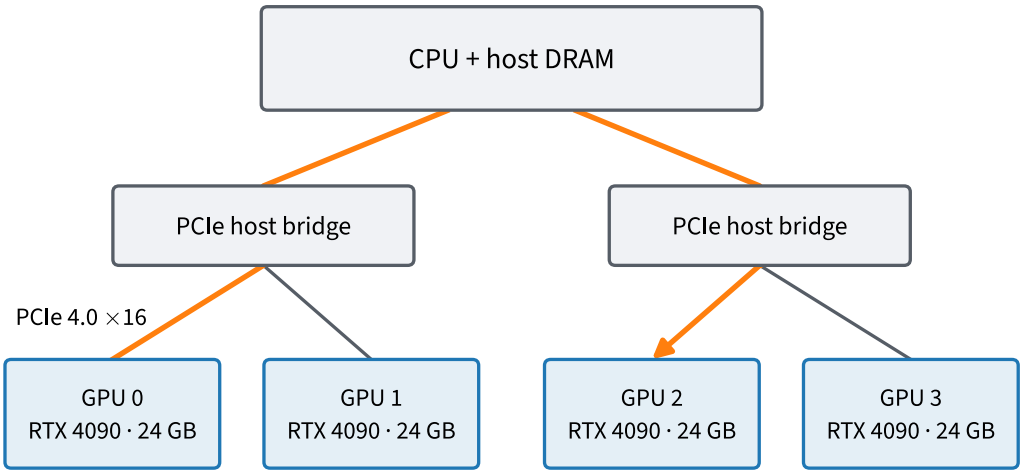


Figure 13.9: `nvidia-smi topo` が報告する本書のマシンの構成。2 つの PCIe ホストブリッジの配下に 4 基の RTX 4090 が 2 基ずつ接続されており、ピアツーピアアクセスも NVLink もない。強調表示された経路は、ある GPU から別の GPU へ向かうすべてのバイトがたどるルートである。

1B パラメータの GPT（fp32 勾配で $S = 4$ GB）の場合、勾配を 23 GB/s で 1 回移動させるだけで本書のワークステーションでは少なくとも 0.17 s かかる。NVLink 5 経由なら 5 ms 未満で済むのに対し、これは bf16 のピークスループットにおける 8,192 トークンの 1 ステップあたりの演算時間（0.3 s）の半分以上を占める。

13.2.5 ルーフライン上のワークロード

言語モデルの推論は 2 つの領域にまたがる（11.3.2.1 節）。**プレフィル**は n トークンのプロンプトを並列に処理し、（層の幅に達するまで）$I \approx n$ の演算密度を達成して計算ルーフ上で動作する。一方、**デコード**は 1 ステップにつき 1 トークンを生成し、bf16 の重み全 $2N$ バイトと KV キャッシュを読み出すため、演算密度は $I \approx 1$ FLOP/byte となる。

13.2.5.1 単一ストリームデコードとバッチデコード

単一ストリームのデコードは帯域幅律速であるため、そのトークン生成速度は次の式で上から抑えられる。

$$\text{tokens/s} \lesssim \frac{\beta}{\text{bytes per token}} = \frac{\beta}{2N + \text{KV bytes}}. \tag{13.4}$$

モデルの重みがカードのメモリ容量 C を埋め尽くすとき、単一ストリームでの最大デコード速度は**メモリスイープ時間**（memory sweep time）C/β、すなわち GPU DRAM の全バイトを 1 回読み出すのに要する時間の逆数になる。Table 13.2 の 4 つの GPU すべてにおいて、容量と帯域幅には 7–8× の開きがあるにもかかわらず、C/β はいずれも **20 ms** 付近に収まっている（H100 と RTX 4090 で 24 ms、B200 で 22.5 ms、RTX 5090 で 18 ms）。したがって、GPU メモリをいっぱいに使う任意の密なモデルは、コンシューマ向けカードであれフラッグシップのデータセンタ向け GPU であれ、単一ストリームをおよそ 40〜55 tokens/s でデコードすることになる。なお、この上限は各トークンがすべての重みを読み出すことを前提としている。混合エキスパート（mixture-of-experts）モデル（11.7 節）は、ルータがそのトークンのために選択したエキスパートのみを読み出すため、C/β が示唆するよりも高速にデコードできる。

同時並行する B 本の系列をバッチ化すると、重みの読み出しが B トークンにわたって償却され、重みに関する演算密度は $I \approx B$ FLOP/byte へと上昇する（B がリッジポイントに達するまで。13.1.3.1 節で実測したとおり）。ただし、各系列は依然として自身の KV キャッシュを個別に読み出す必要がある（11.3.2.2 節）。

13.2.5.2 経験則

Table 13.3 に、本章の残りの節を通じて用いる主要なハードウェア定数をまとめる。

Table 13.3: 経験則：Table 13.2 の広く普及している GPU における代表的なオーダー（最新のデータセンタ向け製品は上限を上回る）、本書のマシン（ピアツーピアアクセスなしの RTX 4090 4 基）における値、およびそれぞれが重要である理由。

物理量	代表的な大きさ	本書のマシン	重要である理由
呼び出しあたりのホスト時間 h	10 µs〜0.1 ms	13〜100 µs（13.1 節）	小さな演算では GPU を稼働させ続けられない
bf16 ピーク演算速度 P	0.17〜2.3 PFLOP/s	165 TFLOP/s	大規模な行列積の速度上限
GPU メモリ帯域幅 β	1〜8 TB/s	約 0.9 TB/s（定格 1.0）	要素ごとの演算とデコードの速度上限

物理量	代表的な大きさ	本書のマシン	重要である理由
リッジポイント P/β	120〜300 FLOP/byte	164 FLOP/byte	これ未満ではメモリ帯域幅が実行時間を決める
スイープ時間 C/β	約 20 ms	24 ms	メモリに収まる最大の密モデルは 1 ストリームを約 40〜55 tok/s でデコード
ホストからデバイス（ピン留め）	32〜64 GB/s（PCIe 4/5）	約 23 GB/s	訓練データはデバイス上またはピン留めメモリに置く
GPU 間	450〜900 GB/s（NVLink）	ホスト経由で $\leq$ 23 GB/s（13.5 節）	毎回の勾配交換にかかる時間を決める
パラメータあたりバイト数（Adam）	16（fp32）、18（混合精度）	16〜18	メモリに収まるモデルサイズを決める（13.4 節）
トークンあたりバイト数（デコード）	$2N$ + KV キャッシュ（bf16）		tokens/s $\lesssim \beta$/bytes per token

13.2.6 まとめ

帯域幅は配線の本数と 1 本あたりの速度の積で決まる。配線がダイの外周に沿って外へ出るのに対し演算ユニットは面積を埋め尽くすため、またオフチップのデータ移動は算術演算の数百倍ものエネルギーを要するため、リッジポイント P/β は数百 FLOP/byte に達し、ハードウェアの世代を追って上昇してきた。メモリは開いた DRAM 行から 32 バイトのセクタ単位でデータを転送するため、細粒度のランダム読み出しは β のごく一部しか達成できず、β を飽和させるには複数のワープにわたって $\beta \times$ latency バイトがインフライトである必要がある。より狭い数値形式はメモリ通信量を半減させ、通常は P を 2 倍にする。最後に、単一ストリームでのデコードにおけるトークン生成速度は $\beta/(2N + \text{KV bytes})$ で上から抑えられ、現代の GPU ではメモリスイープ時間 C/β がいずれも約 20 ms であることから、どのカードであってもメモリに収まる最大の密なモデルは単一ストリームを約 40〜55 tokens/s でデコードする。

13.2.7 演習問題

1. [code] **レイテンシと帯域幅の実測**。●ブロック転送モデル $t(n) = \alpha + n/\beta$（ただし $n_{1/2} = \alpha\beta$、13.2.1 節）は、1 回の同期呼び出しを記述する。
    1. 13.1.3 節の要素ごとの乗算 `x * 1.5` に対する $n_{1/2}$ を、$\alpha = h$ および β に 13.1 節での計測値を用いて予測せよ。
    2. 1 MB から 2 GB までのテンソルサイズに対して `x * 1.5` の実行時間を計測し、実効帯域幅をサイズに対して両対数軸でプロットしたうえで、領域の遷移を Figure 13.5 および Figure 13.3 のオンダイ L2 キャッシュ容量と関連づけよ。
    3. 4 KB から 256 MB までの転送に対して 13.2.4.1 節のホストからデバイスへの計測を繰り返し、α と漸近的な帯域幅をフィッティングにより求めよ。
2. **配線数と伝送速度**。13.2.1 節では、帯域幅を配線の本数と 1 本あたりの速度の積として表した。
    1. DDR5-6400 DIMM は 64 本のデータ配線をもち、各配線は 6.4 Gb/s で動作する。HBM3e スタックは 1,024 本のデータ配線をもち、B200 は 8 個のスタックを搭載している。DIMM 1 枚の帯域幅、0.5 TB/s の CPU ソケットに必要な DIMM の枚数、および Table 13.2 にある B200 の 8 TB/s から逆算される配線 1 本あたりのデータ速度を計算せよ。CPU ソケットと B200 の差を生み出している主な要因は何か。

2. B200 の 2 枚のダイはそれぞれ、8 個の HBM スタックのうち 4 個に隣接している。各ダイ上の SM がメモリの両半分を同じ頻度で読み出し、全体で 8 TB/s になるとき、パッケージ上のブリッジは片方向あたりどれだけの帯域幅を担う必要があるか。

3. **データ供給不足の入力パイプライン**。訓練ジョブの GPU 演算部が、デコード済み画像を 2 GB/s の速度で処理できるとする。各 CPU ワーカーは 2 つの逐次ステージを実行する。まず NVMe から圧縮された JPEG を 1 GB/s（圧縮時サイズ）で読み出し、次に 1 つの CPU コア上で 0.3 GB/s（デコード後出力サイズ、サイズが 8× に拡大）でデコードする。続いて訓練ループは、演算を始める前にデコード済みの各バッチをページャブルなホストメモリから GPU へ 12 GB/s で同期コピーする。

   1. ワーカー 1 個がバッチを供給する速度（デコード後の bytes/s 単位）はいくらか。また、どちらのステージが律速となるか。
   2. GPU にデータを供給し続けるには、何個の並列ワーカープロセスが必要か。
   3. ページャブルメモリからの同期コピーによって各ステップの時間はどの程度の割合だけ延びるか。また、24 GB/s のピン留めメモリを用いた非同期コピーに切り替えると、これはどう変わるか。

4. **リンクのリッジポイント**。リッジポイント P/β は帯域幅 β をもつ任意のデータ経路に適用でき、1 バイトがその経路を通過する間にデバイスが実行できる FLOPs 数を表す。

   1. bf16 における P/β を、RTX 4090 については自身の DRAM とホスト経由の GPU 間経路（23 GB/s）のそれぞれに対して、B200 については自身の HBM と NVLink 5 の片方向（900 GB/s）のそれぞれに対して計算せよ。
   2. データ並列訓練（13.5 節）では、各 GPU がパラメータ数 N のモデルを保持し、1 ステップあたり T トークンを 1 トークンにつき約 $6N$ FLOPs で処理したのち、GPU あたり約 $4N$ バイトを送信するオールリデュース（allreduce）によって bf16 勾配を合算する。通信と演算のオーバーラップがない場合、T がリンクのリッジポイントに比例する（N に依存しない）しきい値を超えるとき、かつそのときに限り、通信時間が演算時間以下になることを示せ。
   3. このトークン数のしきい値を、NVLink 5、23 GB/s のピン留めコピー速度、および本書のワークステーションにおけるデフォルトのホスト経由 NCCL の実効速度である 2.4 GB/s（13.6 節）のそれぞれについて評価せよ。

5. **起動律速のデコード**。Eager モードでの単一ストリームデコードでは、GPU が各ブロックの重みを読み出すのと並行して、CPU がすべてのブロックのすべてのカーネルを起動する。L 個ある幅 d の各ブロックが $12d^2$ 個の bf16 重みを保持し、1 トークンあたり $K = 12$ 個のカーネルを、カーネル 1 個につきホスト時間 $h = 8$ µs で起動するとする。

   1. ホストの起動時間と重みの読み出し時間のどちらが律速となるかが L に依存しないことを示し、RTX 4090 と B200 のそれぞれにおける境界幅 d^* を求めよ。
   2. RTX 4090 上の GPT-2 Small（$d = 768$、$L = 12$）について両方の上限を評価せよ。どちらの上限が律速となるか。
   3. RTX 4090 から B200 への d^* の変化は、より高速な GPU における CUDA グラフのキャプチャとリプレイ（13.3 節）の重要性について何を示唆するか。

6. [code] **ブロック単位でシャッフルしたギャザー**。●13.2.1 節のギャザーのベンチマークを繰り返せ。ただし、個々の要素ではなく連続する k 要素のブロックを $k \in \{1, 2, 4, \dots, 1024\}$ に対してランダムに置換せよ。実効帯域幅をバイト単位のブロックサイズに対してプロットし、遷移が生じる境界サイズを 32 バイトのセクタサイズおよび KB 規模の DRAM 行サイズと比較せよ。

7. **レイテンシ隠蔽**。●リトルの法則（13.2.3 節）をメモリレイテンシの隠蔽に適用せよ。

1. 250 ns のレイテンシの下で B200 の 8 TB/s の HBM を飽和させるのに必要なインフライトのバイト数を計算し、RTX 4090 の 250 KB と比較せよ。DRAM のレイテンシがほぼ一定のまま帯域幅が世代ごとに約 2× のペースで拡大することは、カーネルが引き出すべき並列性に対して何を意味するか。
2. ポインタチェイシング（ポインタ追跡）では、各ロードのアドレスが直前のロード結果に依存するため、1 本の連鎖につきインフライトにできるリクエストは 1 個に限られる。DDR5 のレイテンシが 90 ns、キャッシュラインが 64 バイトの CPU において、単一の連鎖で得られる帯域幅と、0.5 TB/s のソケットを飽和させるのに必要な独立した連鎖の本数を計算せよ。

13.3 計算グラフとコンパイル

13.1 節では、8 個の演算からなる活性化処理の連鎖 `gelu_ish` は、同じテンソルに対する単一の加算と比べて 8〜9 倍の時間がかかった。各演算が個別の GPU カーネルとして実行され、そのたびに入力を DRAM から読み出して出力を書き戻していたためである（配列転送が 18 回発生しており、本来の 2 回では済まなかった）。また、小さなテンソルでは、同じ演算列がもう 1 つの理由でも遅くなる。すべての演算が Python からディスパッチされ、GPU が各カーネルを実行する時間よりも、ホストがカーネルを起動する時間のほうが長くなるからである。

これら 2 つのコストはいずれも、Python が到達した順に演算を 1 つずつ実行する**即時実行**（eager execution）に起因する。実行前にプログラム全体をコンパイラへ渡せば、両方の問題に対処できる。第 1 に、**演算融合**（operation fusion）は要素ごとの演算列を単一のカーネルに統合して入力の読み出しと出力の書き込みを各 1 回に抑え、帯域幅律速（bandwidth-bound）のコストを解消する。第 2 に、**カーネルリプレイ**は一連のカーネル起動を記録してホストからの 1 回の呼び出しでリプレイし、オーバーヘッド律速（overhead-bound）のコストを解消する。計算律速（compute-bound）となる大規模な行列積そのものは、このどちらからも直接の恩恵を受けないが、その前後の要素ごとの演算をコンパイルすることには依然として十分な効果がある。

13.3.1 グラフキャプチャ

13.3.1.1 計算グラフ

ニューラルネットワークの演算は、基本演算をノードとし、テンソルを運ぶエッジで結んだ有向の**計算グラフ**（computation graph）をなす（Figure 13.10）。即時実行の自動微分（1.5 節）でも順伝播の最中に逆伝播用のテープを記録するが、その記録は順伝播自体を最適化するには遅すぎる。テープが出来上がった時点では、すべての演算がすでにホストのディスパッチ時間 h を支払い、GPU メモリとの往復を終えてしまっているからである。プログラムを最適化するには、演算が実行される前に計算グラフを取得する**グラフキャプチャ**が必要になる。

13.3.1.2 トレース

現代のフレームワークは**トレース**によってグラフをキャプチャする。すなわち、形状と dtype だけを持ち具体的な数値を持たないシンボリックな**プレースホルダ**に対して、Python 関数を 1 回実行するのである（Figure 13.11）。プレースホルダに対するすべてのテンソル演算はグラフのノードとして記録され、静的な形状やループ上限のみに依存する通常の Python の制御フローはトレース中に展開される。続いてコンパイラは、キャプチャしたグラフを融合 GPU カーネルへとローワリングし、入力の形状と dtype が一致する後続の呼び出しで再利用できるようキャッシュする。

JAX では、`jax.jit` が抽象トレーサオブジェクトを用いて関数を型付き中間表現（**jaxpr**、`jax.make_jaxpr` で確認可能）へとトレースし (Frostig et al. 2018)、XLA コンパイラがこれを入力の形状と

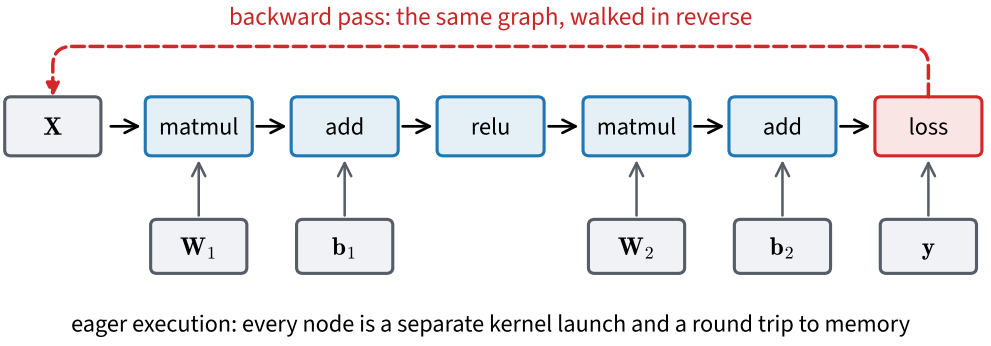


Figure 13.10: 損失を含む 2 層ネットワークの計算グラフ。自動微分はこのグラフを逆方向にたどる。順伝播の前にグラフ全体を持つコンパイラは、グラフを書き換えることができる。

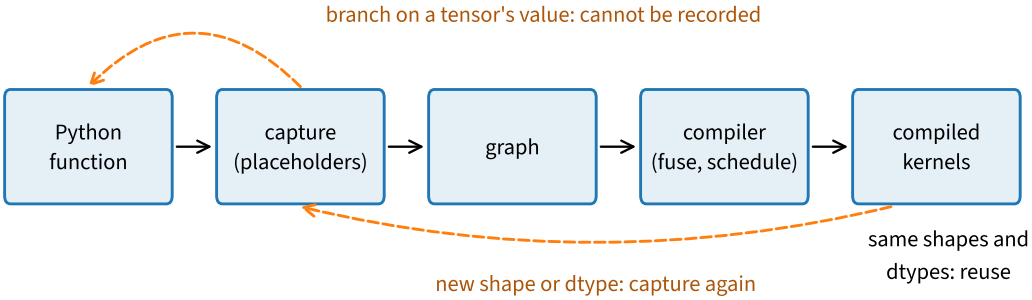


Figure 13.11: グラフキャプチャ。最初の呼び出しでプレースホルダを用いて関数をトレースし、コンパイラが記録されたグラフをカーネルへと変換する。同じ形状と dtype を持つ後続の呼び出しは、そのカーネルを再利用する。テンソルの値に依存する分岐は記録できず、新しい形状や dtype が現れると新たなキャプチャが始まる場合がある。

dtype ごとにキャッシュされる融合 GPU カーネルへとローワリングする。

13.3.1.3 値に依存する分岐

プレースホルダは具体的な数値を持たないため、トレーサは `if x.sum() > 0:` のようなデータ依存の Python の分岐を評価できない。グラフキャプチャの仕組みが取り得る対応は次の 3 通りである。

1. 具体例の入力が通った分岐で**凍結する**。高速だが、将来の入力がもう一方の分岐を通るべきときに、警告なしに誤った結果を返す。
2. エラーを出して**拒否する**。プログラマは、**選択演算**（select、両方の分岐を評価して一方を選ぶ）または**条件分岐演算**（conditional op）というグラフプリミティブで分岐を表現しなければならない。
3. **グラフブレーク**を挿入して Python に**フォールバックする**。条件式の箇所で最初のグラフを終了し、実体化された値に対して Python に分岐を評価させ、2 つ目のコンパイル済みグラフで処理を再開する。

静的なテンソル形状に基づく分岐（`if x.shape[0] > 4:`）はトレース時に確定するため、特別な扱いは不要である。フレームワークがデータ依存の分岐をどのように扱うか見てみよう。

```
def f(x):
    x = x * 2
    if x.sum() > 0:        # Python needs the value of this comparison
        return x + 1
    return x - 1

try:
    jax.jit(f)(jnp.ones(8))
except jax.errors.TracerBoolConversionError as e:
    print(type(e).__name__)

def f_select(x):  # Both alternatives in the graph; select keeps one
    x = x * 2
    return jax.lax.select(x.sum() > 0, x + 1, x - 1)

print(jax.make_jaxpr(f_select)(jnp.ones(8)))
```

```
TracerBoolConversionError
{ lambda ; a:f32[8]. let
    b:f32[8] = mul a 2.0:f32[]
    c:f32[] = reduce_sum[axes=(0,) out_sharding=None] b
    d:bool[] = gt c 0.0:f32[]
    e:f32[8] = add b 1.0:f32[]
    f:f32[8] = sub b 1.0:f32[]
    g:f32[8] = select_n d f e
  in (g,) }
```

`jax.jit` は `TracerBoolConversionError` を送出して拒否する。Python の `if` を `jax.lax.select`（一方の分岐のみを評価したい場合は `jax.lax.cond`）に置き換えれば、単一の jaxpr が得られる。

13.3.1.4 形状と再コンパイル

コンパイラはカーネル（タイルサイズ、ループ上限、メモリレイアウトなど）をテンソル形状に合わせて特化させるため、エポック末尾の短いミニバッチや可変の系列長のように入力形状が変化すると、コンパイラがシンボリック次元へ一般化しない限り再コンパイルが発生する。実務では、入力を少数のバケット化された形状にパディングするか、長さが変化する軸を動的軸として指定する。

```
@jax.jit
def g(x):
    print(f'tracing g for shape {x.shape}')  # Python runs only while tracing
    return jnp.sin(x) + 1

for n in (8, 8, 16, 32):
    g(jnp.ones(n))
```

```
tracing g for shape (8,)
tracing g for shape (16,)
tracing g for shape (32,)
```

`jax.jit` は静的な形状に対して厳密に特化する。Python の `print` は長さ 8、16、32 のトレース中に実行される一方（4 回の呼び出しで 3 回のトレース）、長さ 8 での 2 回目の呼び出しはキャッシュにヒットする。

13.3.2 演算融合

13.3.2.1 メモリトラフィックの算出

即時実行では、`gelu_ish(x)` は 8 個の要素ごとのカーネル（配列を 1 個読み出して 1 個書き出す単項演算 6 個と、配列を 2 個読み出して 1 個書き出す二項演算 2 個）を実行し、合計 $10 + 8 = 18$ 個のサイズ nb バイトの配列を転送する。これに対し、融合カーネルは `x` を 1 回読み出し、7 個の中間値をすべてレジスタに保持して、出力を 1 回だけ書き出す（Figure 13.12）。これにより、メモリトラフィックは次の倍率で削減される。

$$\frac{\text{bytes moved, eager}}{\text{bytes moved, fused}} = \frac{(10+8)\,nb}{(1+1)\,nb} = 9.$$

融合カーネルでさえ約 12 FLOPs / 8 バイト（fp32 の場合、$I \approx 1.5 \ll 82$ FLOP/byte）しか演算しないため、どちらの版も帯域幅律速であり、融合によって `gelu_ish` は約 $9\times$ 高速化するはずである。

```
x = jax.random.normal(d2l.get_key(), (4000, 4000))
compiled = jax.jit(d2l.gelu_ish)
assert jnp.allclose(d2l.gelu_ish(x), compiled(x), atol=1e-6)  # Same answer
arrays_moved = {'eager': 18, 'compiled': 2}  # In units of x.nbytes
for desc, fn in (('eager', d2l.gelu_ish), ('compiled', compiled)):
    b = d2l.Benchmark(lambda: fn(x), desc=desc)
    print(f'{b}, {arrays_moved[desc] * x.nbytes / b.time / 1e9:.0f} GB/s')
```

```
eager: 1.84 ms/call, 627 GB/s
compiled: 0.17 ms/call, 768 GB/s
```

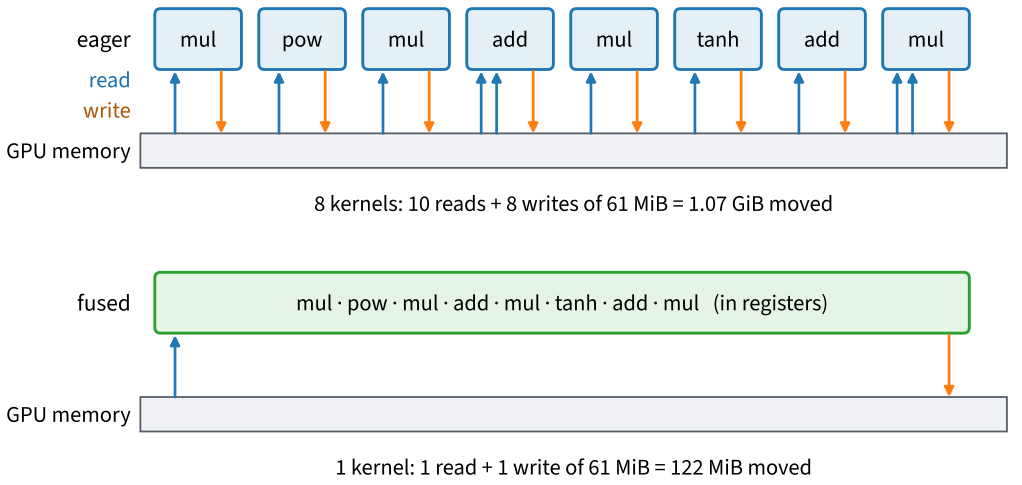


Figure 13.12: `gelu_ish` のメモリトラフィック。即時実行は 8 個のカーネルを実行して配列を 10 回読み出し 8 回書き出すのに対し、融合カーネルは入力を 1 回読み出し、7 個の中間結果をレジスタに保持して、結果を 1 回だけ書き出す。

どちらの版も 0.6〜0.9 TB/s でメモリをストリーミングし、コンパイル版は 18 回の DRAM 転送のうち 16 回を省くだけで $8\times$ から $11\times$ 高速に動作する。これはバイト数から予測した $9\times$ とよく一致している。

Transformer ブロックの内部では、活性化の融合はどれほど重要だろうか。幅 $d = 2048$、FFN の隠れ幅 $4d = 8192$ を持つ bf16 の 1B パラメータ GPT において、融合されていない 8 演算の活性化を M トークンに対して実行すると $18 \cdot 4dM \cdot 2 = 144\,dM$ バイトが転送される一方、その前後にある 2 つの FFN 行列積は $2 \cdot 2 \cdot M \cdot d \cdot 4d = 16\,d^2 M$ FLOPs を実行する。両者の所要時間の比はトークン数 M に依存しない。

$$\frac{t_{\text{activation}}}{t_{\text{products}}} = \frac{144\,dM/\beta}{16\,d^2 M/P} = \frac{9I^*}{d} \approx 0.72$$

ただし、上の数値は $I^* = 164$ FLOP/byte、$d = 2048$ とした場合である。融合しなければ、活性化には 2 つの行列積の 4 分の 3 近い時間がかかるが、融合すれば 10 分の 1 未満に収まる。

13.3.2.2 融合のパターン

コンパイラが日常的に融合するパターンは次の 2 つである。

- **要素ごとの演算列**（活性化関数、残差加算、ドロップアウトマスク、オプティマイザの更新）：各スレッドが出力の要素 i をレジスタ内で計算する。
- **行リダクション**（ソフトマックス、LayerNorm、RMSNorm）：各行がオンチップ SRAM に収まるため、カーネルは行を 1 回読み出し、リダクションと正規化をオンチップで計算して、出力を 1 回だけ書き出す。

コンパイラは新しいアルゴリズムを発明するわけではない。計算律速の行列積を高速化することはできないし、FlashAttention（10.5 節、(Dao et al. 2022)）のようなタイル化アルゴリズムを自動的に発見することもできない。FlashAttention はオンラインソフトマックスの再スケーリン

グによって $T \times T$ のアテンション行列を実体化せずに済ませる手法であり、CUDA や Triton で手書きされている (Tillet et al. 2019)。

13.3.3 訓練ステップのコンパイル

ここからは、幅 1,024、`gelu_ish` 活性化を持つ 4 層 MLP に対し、512 個の例からなるバッチ上での完全な訓練ステップ（順伝播、二乗損失、逆伝播、SGD 更新）をコンパイルしてみよう。

```
def init_mlp(key, width=1024, depth=4):
    """Weights and biases drawn uniformly from +-1/sqrt(width)."""
    bound = 1 / width**0.5
    keys = jax.random.split(key, (depth, 2))
    return [(jax.random.uniform(kW, (width, width), minval=-bound,
                                maxval=bound),
             jax.random.uniform(kb, (width,), minval=-bound, maxval=bound))
            for kW, kb in keys]

def mlp(params, X):  # Linear layers with the gelu_ish chain between them
    for W, b in params[:-1]:
        X = d2l.gelu_ish(X @ W + b)
    W, b = params[-1]
    return X @ W + b

def train_step(params, X, Y, lr=0.1):  # Forward, backward, SGD update
    loss_fn = lambda p: ((mlp(p, X) - Y) ** 2).mean()
    loss, grads = jax.value_and_grad(loss_fn)(params)
    return loss, jax.tree.map(lambda p, g: p - lr * g, params, grads)
```

まず 1 ステップの所要時間を予測してみよう。順伝播の 4 回の行列積と逆伝播の 7 回の行列積（最初の層は入力に関する勾配を必要としない）で約 $11 \times 2 \cdot 512 \cdot 1024^2 \approx 1.2 \times 10^{10}$ FLOPs を実行し（tf32 の 82.6 TFLOP/s で 0.14 ms）、16 MiB のパラメータの読み出しと更新で約 96 MiB を転送するため（1 TB/s で 0.10 ms）、実際のデバイス処理は合計**約 0.25 ms** となる。ところが即時実行では、100 個を超える個別の演算をそれぞれ $h \geq 10$ µs かけて発行するため（ホストの起動オーバーヘッドだけで 1 ms 超）、即時実行のステップはオーバーヘッド律速になる。

コンパイルには最初の呼び出し時に t_{compile} のコストがかかり、1 ステップあたり $t_{\text{eager}} - t_{\text{compiled}}$ だけ時間が短縮されるため、元が取れるまでのステップ数は次の式で与えられる。

$$n^* = \frac{t_{\text{compile}}}{t_{\text{eager}} - t_{\text{compiled}}} \tag{13.5}$$

このステップ数を実行した時点で損益分岐点に達する。

```
params = init_mlp(d2l.get_key())
X, Y = (jax.random.normal(d2l.get_key(), (512, 1024)) for _ in range(2))
compiled_step = jax.jit(train_step)
t_first = d2l.Benchmark(lambda: compiled_step(params, X, Y),
                        warmup=0, repeats=1).time  # Includes compilation
t_eager = d2l.Benchmark(lambda: train_step(params, X, Y)).time
t_compiled = d2l.Benchmark(lambda: compiled_step(params, X, Y)).time
saved = t_eager - t_compiled  # Per step; a difference of two short times
```

```
print(f'first call: {t_first:.1f} s\n'
      f'eager: {1000 * t_eager:.2f} ms, compiled: {1000 * t_compiled:.2f} ms\n'
      'break-even: ' + (f'about {float(f"{t_first / saved:.1g}"):,.0f} steps'
                        if saved > 0 else 'never, compiled is not faster'))
```

```
first call: 5.8 s
eager: 30.87 ms, compiled: 0.26 ms
break-even: about 200 steps
```

`jax.jit` は順伝播、逆伝播、パラメータ更新を 1 回の呼び出しで起動される単一の XLA プログラムへとコンパイルし、約 0.25 ms で実行する。これは机上の見積もりと一致し、JIT なしの JAX より約 100× 高速である。

13.3.4 カーネルリプレイ

融合によって行列積同士を統合することはできず、残った各カーネルの実行時間が短い場合には、カーネルごとのホストからの起動がステップ時間の大半を占めるようになる。幅 256 のバイアスなし線形層と tanh 活性化を 60 層重ねたネットワークに、64 個の例からなるバッチを入力する場合を考えよう（1 回の呼び出しあたり 120 演算）。各層は 8.4×10^6 FLOPs の演算と 384 KiB の転送しか行わないが（ルーフライン上では 0.5 µs 未満であり、ハードウェアのカーネル起動下限にカーネルあたり約 1〜3 µs が加わって実際のデバイス実行時間は約 4〜5 µs）、即時実行では 1 演算ごとに h（合計 $120\,h$）を支払うことになる。

カーネルごとのホストディスパッチをなくすため、NVIDIA の GPU は **CUDA グラフ**をサポートしている。ランタイムが一連のカーネル起動と固定されたバッファアドレスを 1 回記録し、その後はホストからの 1 回の起動で一連の処理全体をリプレイする仕組みである。記録時にバッファアドレスと形状が固定されるため、リプレイ前に入力を固定バッファへコピーし、次のリプレイ時には出力が上書きされる。

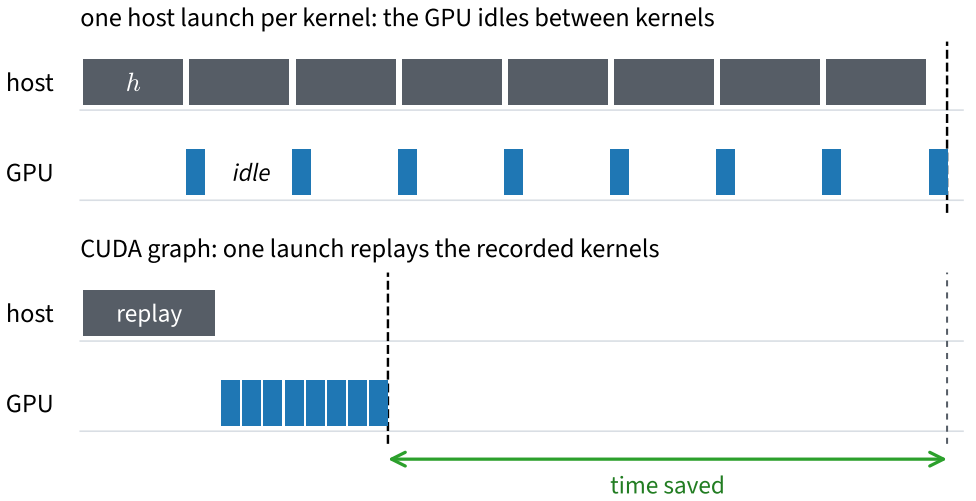


Figure 13.13: 短いカーネルを多数重ねた場合のホストとデバイスのタイムライン。カーネルごとに 1 回ずつ起動すると、GPU は毎回ホストを待つことになり、カーネル間でアイドル状態になる。CUDA グラフは 1 回の起動で一連の処理全体をリプレイするため、カーネルが隙間なく連続して実行される。

Figure 13.13 に 2 つのタイムラインの比較を示す。個々のカーネル自体の実行時間はどちらでも同じであり、リプレイが取り除くのはカーネル間のアイドル時間である。

```
def init_deep(key, depth=60, width=256):  # 120 operations per call
    return list(jax.random.uniform(key, (depth, width, width),
                                   minval=-width**-0.5, maxval=width**-0.5))

def deep(params, x):
    for W in params:
        x = jnp.tanh(x @ W)
    return x

params = init_deep(d2l.get_key())
x = jax.random.normal(d2l.get_key(), (64, 256))
jitted = jax.jit(deep)
jitted(params, x)  # First call compiles
times = {}
for desc, f in (('eager', deep), ('jit', jitted)):
    t = times[desc] = d2l.Benchmark(lambda: f(params, x), repeats=100).time
    print(f'{desc}: {1000 * t:.2f} ms/call, '
          f'{1e6 * t / 120:.1f} µs per operation')
# The kernels of the jitted stack, against the time of a call
d2l.show_profile(d2l.kernel_times(lambda: jitted(params, x)),
                 1000 * times['jit'], rows=3)
```

```
eager: 7.79 ms/call, 64.9 µs per operation
jit: 0.25 ms/call, 2.1 µs per operation
wall clock 0.25 ms per step, device busy 0.24 ms (95%), matrix products 74%
  0.17 ms  73%  gemm_fusion_dot_1
  0.06 ms  25%  loop_reduce_fusion_1
  0.00 ms   1%  gemm_fusion_dot
```

`jax.jit` は各 tanh を直後の行列積に直接融合し、コンパイルされた一連の処理を XLA コマンドバッファ（CUDA グラフ）としてリプレイするため、即時ディスパッチより約 30× 高速に動作し、約 0.25 ms の呼び出しのほぼ全期間にわたって GPU を稼働させ続ける。

13.3.5 コンパイルの限界

コンパイルが割に合うのは、プログラムが n^* ステップを超えて実行され、かつ帯域幅律速の演算列やホスト起動オーバーヘッドに多くの時間を費やしている場合に限られる。実務上、次の 3 つの注意点に留意しておこう。

- **テンソルの値に依存する分岐：**スカラーの損失のログ出力、`if loss < 0.1:` のような判定、データ依存のインデックス参照（`x[x > 0]`）は、グラフブレークやエラーを引き起こす。ホスト側での値の読み取りはコンパイル対象関数の外に出し、グラフ内の分岐は `where` / `select` や `cond` で表現する。
- **変化する形状：**可変の系列長やエポック末尾の短いミニバッチは、繰り返しの再コンパイルを引き起こすことがある。最後のミニバッチは切り捨てるかパディングし、可変長の系列はバケット化する。
- **デバッグ：**コンパイル済みカーネルは Python 上で 1 行ずつ実行されるわけではない。まず即時実行モードで動作検証とデバッグを済ませてからコンパイルする。

13.3.6 まとめ

即時実行は演算を 1 つずつ実行するため、カーネルごとにホストディスパッチのレイテンシ h を支払い、要素ごとの演算のたびに DRAM との往復が発生する。トレースは実行前にシンボリックなプレースホルダ上で計算グラフをキャプチャする。その際、データ依存の分岐は選択・条件分岐プリミティブで書き換えなければグラフブレークを招き、形状の変化は次元をシンボリックにしない限り再コンパイルを引き起こす。演算融合は要素ごとの演算列やオンチップの行リダクションを単一のカーネルに統合し（`gelu_ish` のメモリトラフィックと実行時間を 9 分の 1 に削減する）、CUDA グラフのリプレイは小さなモデルやバッチにおけるカーネルごとのホスト起動オーバーヘッドを解消する。

13.3.7 演習問題

1. [code] **値に依存する分岐**。トレースは引数の値をシンボリックなプレースホルダに置き換える（13.3.1 節）。
   1. `f` の分岐を `jnp.where` と `jax.lax.cond` でそれぞれ書き換え、`jax.make_jaxpr` で両方の jaxpr を確認せよ。実行時にはそれぞれどの演算が実行されるか。
   2. ベクトルのバッチに対して `jax.vmap` を `jax.lax.cond` 版に適用し、その jaxpr を確認せよ。条件分岐はどう変化したか。また、それはなぜか。
   3. `x.shape[0] > 4` で分岐する関数を JIT コンパイルし、長さ 3、5、6 のベクトルに対して呼び出せ。なぜエラーにならず成功するのか。また、何回トレースされるか。
2. [code] **形状の変化**。JIT コンパイルされた関数は引数の形状に特化する（13.3.1 節）。
   1. トレース中に Python のカウンタをインクリメントし、`jnp.tanh(x).mean()` を返す関数を JIT コンパイルせよ。長さ 100、200、300 のベクトルに対してそれぞれ 3 回ずつ呼び出し、初回呼び出しとキャッシュヒット時の呼び出し時間を比較せよ。
   2. すべての入力を長さ 512 にパディングし、マスクまたは長さを渡すことで、3 つの長さすべてについてパディング前の厳密な平均を単一のトレースで計算せよ。
   3. 長さをトレース対象の配列として渡す場合と、`static_argnums` で指定する場合とを比較せよ。
3. [code] **行列積に続く融合**。●コンパイルは大規模な行列積をそのままにしつつ、直後に続く要素ごとの演算列を融合する。`gelu_ish(a @ a)` を $n \times n$ の bf16 行列 `a` に対して実行する場合を考える（GPU のピーク性能を P、帯域幅を β、リッジポイントを $I^* = P/\beta$ とする）。行列積を計算律速（$2n^3/P$ 秒）、後続の演算列を帯域幅律速（即時実行モードでは $m = 18$ 個、融合時は 2 個の配列を転送し、各配列は n^2 個の $b = 2$ バイト要素からなる）としてモデル化せよ。
   1. 即時実行に対するコンパイル実行の高速化率の上界 $S(n)$ を n, m, b, I^* で表せ。また、RTX 4090（$I^* \approx 164$）において $n = 1024, 4096, 16384$ での値を計算し、さらに n のうち $S(n) = 1.1$ となる値を求めよ。
   2. `gelu_ish(a @ a)` の即時実行時間とコンパイル実行時間を $n = 256, 512, \ldots, 8192$ について計測し、実測された高速化率を $S(n)$ と重ねてプロットし、両者が乖離する領域を説明せよ。
4. **リダクションの融合**。●$R \times d$ の fp32 行列に対する数値的に安定な行ごとのソフトマックスは、行の最大値、減算、指数関数、行の和、除算という 5 つの即時実行演算として書ける。
   1. 5 つの即時実行カーネルと、各行を SRAM に保持する単一の融合カーネルのそれぞれについて、DRAM の読み出し回数と書き込み回数を数え（Rd に比例する項のみを残す）、帯域幅律速における高速化率を答えよ。
   2. 1 行が SRAM 容量を超える場合について、最大値と和を別々のパスで計算する融合カーネルと、$m_j = \max(m_{j-1}, x_j)$ および $s_j = s_{j-1} e^{m_{j-1} - m_j} + e^{x_j - m_j}$ を更新するオンラインの 1 パスリダクションを用いた融合カーネルのそれぞれに必要な最小パス数を求めよ。また、$s_d = \sum_{j=1}^{d} e^{x_j - m_d}$ が成り立つことを確かめよ。

5. [code] **トレース時の定数**。 JIT コンパイルされた関数の Python 本体はトレース中にのみ実行される（13.3.1 節）。
   1. `x + np.random.randn()` を返す関数を JIT コンパイルして 2 回呼び出し、その jaxpr を確認したうえで 2 回の出力結果を説明せよ。
   2. JIT コンパイルされたドロップアウト関数が Python のグローバル変数から PRNG キーを読み取るとする。ステップ間でそのグローバル変数に再代入しても何が起きるかを説明し、呼び出しごとに新しいマスクが生成されるよう関数を書き換えよ。
6. [code] **オーバーヘッドの交差点**。●13.3.4 節の 60 層スタックを用いて、カーネルリプレイの効果がなくなる境界を調べよ。
   1. 幅 256 のまま深さ $L \in [10, 200]$ にわたって即時実行とリプレイ実行の時間を計測し、$t(L) = c + sL$ を当てはめて傾きを比較せよ。
   2. $\max(\text{FLOPs}/P, \text{bytes}/\beta) = h$ から、1 層の行列積の実行時間と起動時間が等しくなる幅を予測せよ。次に深さ 60 で幅を 64 から 4,096 まで変えて即時実行とリプレイ実行の時間を計測し、両者が交差する幅を予測と比較せよ。
   3. 即時実行のスタックを `d2l.kernel_times` と `d2l.show_profile` でプロファイルし、デバイスの稼働割合および起動 1 回あたりの実時間をコンパイル済みプロファイルと比較せよ。

13.4 メモリと精度

これまでの節では訓練ステップがどれほど高速に実行されるかを問うてきたが、そもそも GPU メモリに収まらないステップは実行すらできない。fp32 の Adam で訓練する 1B パラメータの GPT は、パラメータあたり 4 つの fp32 値（重み、その勾配、および Adam の 2 つのモーメント推定値）を保持するため、トークンを 1 つも読み込む前に 24 GiB の RTX 4090 のうち $16N \approx 15.1$ GiB を消費する。残りの約 9 GiB に、順伝播が誤差逆伝播法（backpropagation）のために保存する**活性化値**（activation）を収めなければならず、これにはトークンあたり数メガバイトを要する。その結果、1,024 トークンの系列は 2〜3 本しかカードに収まらないのに対し、大規模言語モデル（large language model）は 50 万〜数百万トークンのグローバルバッチで訓練される (Brown et al. 2020)。

訓練ステップのメモリは、N に比例する固定のパラメータ項と、トークン数 n に比例する活性化項からなる。本節では、この GPT について両者を数え上げ、実測ピークメモリと照合して検証したうえで、活性化項またはマイクロバッチサイズに働きかける 3 つの手法を学ぶ。すなわち、16 ビットで計算する**混合精度**（mixed precision）、活性化値を保存する代わりに逆伝播中に再計算する**活性化チェックポイント**（activation checkpointing）、および大きなグローバルバッチを逐次的なマイクロバッチに分割する**勾配累積**（gradient accumulation）である。これら 3 つのいずれも、$16N$ バイトのパラメータ状態を縮小することはない。その状態を複数の GPU に分割する方法は 13.7 節の主題である。

```
import re
```

13.4.1 訓練ステップのメモリ

13.4.1.1 パラメータあたりのメモリ

パラメータメモリはオプティマイザに依存する。通常の fp32 SGD は重みと勾配を保持し（パラメータあたり 8 バイト）、モメンタム付き SGD はこれに速度を加え（12 バイト）、Adam は 2 つのモーメント推定値を保持する（**パラメータあたり 16 バイト**、5.7.3.2 節）。混合精度訓練では、これら 4 つすべてを fp32 で保持したうえで、ステップ中に 16 ビットの重みのコピーを保持する

（**パラメータあたり 18 バイト**）。

基準とするモデルは `d2l.GPT`（11.2 節）であり、幅 $d = 2048$、16 ヘッド、$L = 18$ 個の Pre-Norm SwiGLU ブロック、語彙サイズ $V = 50{,}257$、および学習可能な位置埋め込みを備える。各ブロックは約 $12d^2 \approx 50$M パラメータを保持する（18 ブロック全体で約 0.9B、さらに重み共有されたトークン埋め込みに約 0.1B）。メタデバイス（形状のみを扱うデバイス）上でモデルを検査すれば、4 GB の重みを確保することなく N を確認できる（5.2.2.2 節）。

```
config = dict(vocab_size=50257, num_hiddens=2048, num_heads=16,
              num_blks=18, pos='learned')
key = d2l.get_key()
model = nnx.eval_shape(lambda: d2l.GPT(**config, rngs=nnx.Rngs(key)))
N = d2l.num_params(model)
print(f'{N / 1e9:.2f} billion parameters, 16N = {16 * N / 2**30:.1f} GiB')
```

```
1.01 billion parameters, 16N = 15.1 GiB
```

13.4.1.2 トークンあたりのメモリ

順伝播の間、各層は逆伝播で必要となる中間活性化値を保存し、逆伝播の間にはパラメータ勾配が生成されるにつれてこれらの活性化値が層ごとに解放される（Figure 13.14）。したがって、メモリは逆伝播の開始時点でピークに達する。

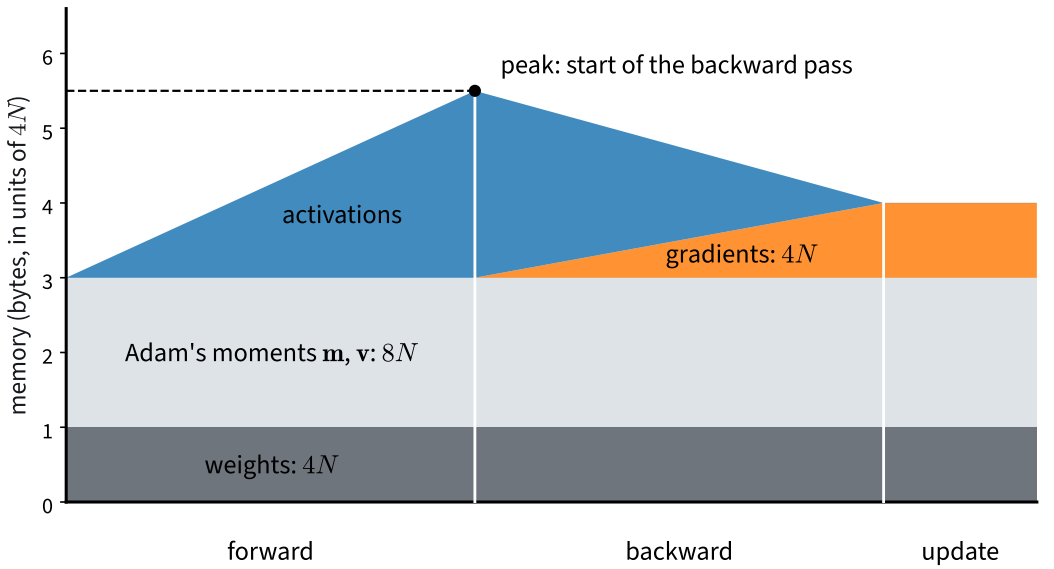


Figure 13.14: Adam を用いた訓練ステップのメモリ（$4N$ バイト単位）。重み（$4N$）と Adam のモーメント推定値（$8N$）はステップ全体を通して保持される。活性化値は順伝播中に蓄積され、逆伝播中に勾配（$4N$）が現れるにつれて解放される。メモリは逆伝播の開始時点でピークに達する。

Transformer ブロックが誤差逆伝播法のために何を保存するかは、次の 3 つの規則で決まる。

1. 線形層 $\mathbf{y} = \mathbf{W}\mathbf{x}$ は入力 $\mathbf{x}$ を保存する（$\nabla_{\mathbf{W}} = (\nabla_{\mathbf{y}})\mathbf{x}^\top$ であるため）。
2. 要素ごとの非線形性 f は入力または出力を保存する。

3. 要素ごとの積 $\mathbf{p} \odot \mathbf{q}$ は両方の因子を保存する。

幅 $d = 2048$ を単位としてトークンあたりで数え上げると、次のようになる。

- **アテンションサブ層：**アテンション前の RMSNorm が入力（d）を保存し、QKV 射影が正規化後の出力（d）を保存する。融合された因果アテンションは $\mathbf{Q}, \mathbf{K}, \mathbf{V}$（$3d$）とその出力（$d$、および $T \times T$ のスコア行列の代わりにヘッド・トークンあたりスカラーの対数正規化項のみ）を保存し、出力射影はアテンション出力を保存する（エイリアスであれば 0、ヘッド結合時にコピーされる場合は d）。合計は $6d$–$7d$ である。
- **SwiGLU FFN サブ層：**FFN 前の RMSNorm が入力（d）を保存し、ゲート射影とアップ射影が正規化後の出力（d）を共有する。さらに、$\mathrm{SiLU}(\mathbf{g}) \odot \mathbf{u}$ とダウン射影がゲート $\mathbf{g}$、活性化後のゲート $\mathrm{SiLU}(\mathbf{g})$、アップ射影 $\mathbf{u}$、およびそれらの積という 4 本のベクトル分（幅 $\frac{8}{3}d \approx 10.7d$）を保存する。合計は約 $12.7d$ である。

合計すると、融合された 1 ブロックはトークンあたり $18.7d$–$19.7d \approx$ **20d** 個の値を保存する（$d = 2048$ の fp32 で **160 KiB** であり、ゲートなしの GPT-3 ブロックに対する $16d$ という Korthikanti et al. (2022) の集計と比較できる）。融合アテンションは逆伝播中に $T \times T$ の重みをオンチップで再計算するため、この集計値の中に系列長 T に応じて増大するものは一切ない。最後に、交差エントロピー損失（cross-entropy loss）はトークンあたり $V = 50{,}257$ 個の fp32 対数確率（**196 KiB**）を保存し、これは Transformer ブロック 1 つ分全体よりも大きい。

この集計を実験的に検証するため、まず `saved_bytes` を定義し、`gpt_loss` の逆伝播のために保持されるテンソルの合計バイト数を求める。

```
def saved_bytes(f, *args):
    """Bytes of the arrays that the backward pass of f keeps."""
    _, f_vjp = jax.vjp(f, *args)  # The leaves of f_vjp are the saved arrays
    return sum(x.nbytes for x in jax.tree_util.tree_leaves(f_vjp))

def gpt_loss(model, X):
    """Cross-entropy of a GPT's predictions, with the inputs as targets."""
    logits = model(X).astype(jnp.float32)
    return optax.softmax_cross_entropy_with_integer_labels(logits, X).mean()
```

1 ブロックおよび 2 ブロックの GPT モデルに対してバッチサイズ $b \in \{1, 2\}$ で `saved_bytes` を評価することにより、`bytes_per_token` は単一の Transformer ブロックが要するトークンあたりの限界活性化メモリ（`block`）を、埋め込み・出力ヘッド・損失の分（`head`）から分離する。

```
def to_bf16(model):
    """Make the linear layers of `model` compute in bf16; parameters stay
    fp32."""
    for _, layer in nnx.iter_graph(model):
        if isinstance(layer, nnx.Linear):
            layer.dtype = jnp.bfloat16
    return model

def bytes_per_token(config, bf16=False, num_steps=1024):
    """Bytes stored per token by one block, and by the rest of a GPT."""
    per_token = []
    for L in (1, 2):  # GPTs with one and with two blocks
        model = d2l.GPT(**{**config, 'num_blks': L},
                        rngs=nnx.Rngs(d2l.get_key()))
```

```
        model = to_bf16(model) if bf16 else model
        # The structure, the weights, and the other state of the model;
        # nnx.merge puts them back together, so the loss is a function of
        # the weights
        graphdef, params, rest = nnx.split(model, nnx.Param, ...)
        loss = lambda params, X: gpt_loss(nnx.merge(graphdef, params, rest), X)
        X1, X2 = (jax.random.randint(d2l.get_key(), (b, num_steps), 0,
                                     config['vocab_size']) for b in (1, 2))
        per_token.append((saved_bytes(loss, params, X2)
                          - saved_bytes(loss, params, X1)) / num_steps)
    return per_token[1] - per_token[0], 2 * per_token[0] - per_token[1]

block, head = bytes_per_token(config)
print(f'one block: {block / 2**10:.0f} KiB per token '
      f'({block / (4 * config["num_hiddens"]):.1f} fp32 values of width d)')
print(f'embedding, output layer, and loss: {head / 2**10:.0f} KiB per token')
```

```
one block: 232 KiB per token (29.0 fp32 values of width d)
embedding, output layer, and loss: 228 KiB per token
```

XLA コンパイルの前段階では、`jax.vjp` はブロックあたり $29.0d$ 個の値（232 KiB）を記録する。これは `nnx.RMSNorm` と `jax.nn.silu` が要素ごとの複合式として記述されており、その中間配列が未加工の VJP 残差に残るためである。コンパイル後のプログラムがそれらをすべて保持し続けるかどうかは XLA のメモリ計画次第であり、これについては以下で確認する。

13.4.1.3 メモリバジェット

パラメータ状態（マイクロバッチ間で勾配が持続する場合は $16N$ バイト）に、1 ブロック・1 トークンあたり a バイトと、埋め込み・出力ヘッド・損失のトークンあたり a_{out} バイトを合わせると、n トークンに対するステップのメモリバジェットが得られる。

$$M(n) \approx 16N + n\,(La + a_{\text{out}}) \;\le\; M_{\text{GPU}}. \tag{13.6}$$

24 GiB のカード上の 18 ブロック GPT（$16N = 15.1$ GiB であり、活性化値には 8.9 GiB が残る）について、生の集計値は次のように予測する。

トークンあたり $La + a_{\text{out}} \approx 4.3$ MiB（約 2,100 トークン、すなわち長さ 1,024 の系列 2 本分）である。

13.4.1.4 実測ピークメモリ

保存された活性化値に加えて逆伝播が一時的に何を確保するかを調べるため、2 ブロック版の GPT において 1,024〜8,192 トークンにわたり、重みを除いたピークメモリを計測する。

```
key = d2l.get_key()
model = nnx.eval_shape(
    lambda: d2l.GPT(**{**config, 'num_blks': 2}, rngs=nnx.Rngs(key)))
graphdef, params, rest = nnx.split(model, nnx.Param, ...)
grad_fn = jax.jit(jax.grad(
    lambda params, rest, X: gpt_loss(nnx.merge(graphdef, params, rest), X)))
peaks = {}
```

```
for n in (1024, 2048, 4096, 8192):  # Tokens per step, in sequences of 1,024
    X = jax.ShapeDtypeStruct((n // 1024, 1024), jnp.int32)
    plan = grad_fn.lower(params, rest, X).compile().memory_analysis()
    peaks[n] = plan.output_size_in_bytes + plan.temp_size_in_bytes
    print(f'{n:5d} tokens: peak {peaks[n] / 2**30:.2f} GiB above the weights')
for n in (2048, 4096):  # Growth per token from n to 2n tokens
    slope = (peaks[2 * n] - peaks[n]) / n
    print(f'{n:5d} to {2 * n:5d} tokens: planned {slope / 2**10:.0f} KiB '
          f'per token; counted {(2 * block + head) / 2**10:.0f} KiB')
```

```
1024 tokens: peak 1.23 GiB above the weights
2048 tokens: peak 1.63 GiB above the weights
4096 tokens: peak 3.09 GiB above the weights
8192 tokens: peak 5.79 GiB above the weights
2048 to  4096 tokens: planned 749 KiB per token; counted 692 KiB
4096 to  8192 tokens: planned 691 KiB per token; counted 692 KiB
```

サイズが小さいうちは固定オーバーヘッドがまだ現れる。それが無視できるようになると、ピークはトークン数に比例して増加し、このトークンあたりの増加量こそが集計値と比較すべき数値となる。

4,096〜8,192 トークンにおいて、XLA が計画したピークはトークンあたり約 690 KiB ずつ増加し、未加工の VJP 集計値である 692 KiB と一致する。したがって、コンパイル後のプログラムはトークンあたり `jax.vjp` が記録したのと同量のメモリを必要とし、損失部分での計測可能な超過はなく、この集計値をそのままバジェットとして使える。

13.4.2 混合精度

混合精度訓練（5.5 節）では、マスター重み（master weights）、勾配、オプティマイザのモーメント、正規化、残差ストリーム（residual stream）、および損失を fp32 に保ちながら、行列積と中間活性化値を 16 ビット浮動小数点で実行する。これには、fp32 と同じ 8 ビットの指数部範囲を共有し損失スケーリング（loss scaling）を必要としない bf16、あるいは損失スケーリングを伴う fp16（5.5.3.3 節）を用いる。

ブロック内の正規化に入力される 2 つの残差ストリームは fp32（4 バイト）のままである一方、残りの 18 個分（ブロックあたりに保存される約 $20d$ 個の値のうち）は bf16（2 バイト）に下がるため、1 ブロックが保存する活性化値は fp32 のサイズの約 $(2 \cdot 4 + 18 \cdot 2)/(20 \cdot 4) = 0.55$ 倍に縮小する。

```
block16, head16 = bytes_per_token(config, bf16=True)
print(f'one block: {block16 / 2**10:.0f} KiB per token in bf16, '
      f'{block16 / block:.2f} of fp32')
print(f'embedding, output layer, and loss: {head16 / 2**10:.0f} KiB per token')
```

```
one block: 140 KiB per token in bf16, 0.60 of fp32
embedding, output layer, and loss: 228 KiB per token
```

1 ブロックあたりの活性化バイト数は fp32 の 0.58–0.60 倍に減少する。しかし、ステップ中に重みの bf16 コピーを保持することで、固定のパラメータメモリが $16N = 15.1$ GiB から $18N = 17.0$ GiB へと増加し、活性化値には 7.0 GiB しか残らなくなる。活性化値の節約分と追加の重みコピーの分は大部分が相殺し合うため、24 GiB カード上での最大トークン容量は 15〜27% しか拡大しない。

bf16 混合精度の主な利点は容量よりも速度にある。RTX 4090 では、bf16 の Tensor Core は 165 TFLOP/s で動作し、tf32 および fp32 の 82.6 TFLOP/s の 2 倍に達する。Tensor Core の効果（tf32 対スカラー fp32）と 16 ビットオペランドの効果（bf16 対 tf32）を切り分けるため、計算律速（compute-bound）となる幅 8,192 の 3 層 MLP において、2,048 個の例からなるバッチに対する SGD の 1 ステップをベンチマークする。このステップは 1.1×10^{12} FLOPs を要し、ルーフライン下限は 82.6 TFLOP/s で 13.3 ms、165 TFLOP/s で 6.7 ms であり、これに帯域幅律速の SGD 更新の約 1.3 ms が加わる。

```
def net_fn(params, X, dtype):
    h = X.astype(dtype)
    for W in params[:-1]:
        h = jax.nn.relu(h @ W.astype(dtype))
    return h @ params[-1].astype(dtype)

sizes = [(64 * 64, 8192), (8192, 8192), (8192, 10)]  # (inputs, outputs)
mlp_params = [jax.random.normal(k, s) * 0.02
              for k, s in zip(jax.random.split(d2l.get_key(), 3), sizes)]
X_mlp = jax.random.normal(d2l.get_key(), (2048, 64 * 64))
y_mlp = jax.random.randint(d2l.get_key(), (2048,), 0, 10)

def loss_fn(params, X, y, dtype):
    logits = net_fn(params, X, dtype).astype(jnp.float32)
    return optax.softmax_cross_entropy_with_integer_labels(logits, y).mean()

def sgd_step(params, X, y, dtype):
    grads = jax.grad(loss_fn)(params, X, y, dtype)
    return [p - 0.01 * g for p, g in zip(params, grads)]

step = jax.jit(sgd_step, static_argnums=3)
args = mlp_params, X_mlp, y_mlp
```

次に、MLP の順伝播および逆伝播の行列積について正確な積和演算数（`macs`）と合計 `flops` を数え、各精度形式をミリ秒と TFLOP/s でベンチマークする関数 `report` を定義する。

```
# Two FLOPs per multiply-add, in the forward pass, the weight gradients, and
# the input gradients of every layer but the first
macs = sum(i * o for i, o in sizes)  # Per example and pass
flops = 2 * len(X_mlp) * (3 * macs - sizes[0][0] * sizes[0][1])
print(f'{flops:.2e} FLOPs per step')

def report(desc, f):
    """Print the time per call of f and the arithmetic rate it achieves."""
    t = d2l.Benchmark(f).time
    print(f'{desc}: {1000 * t:5.2f} ms, {flops / t / 1e12:3.0f} TFLOP/s')
```

```
1.10e+12 FLOPs per step
```

それでは `report` を、厳密なスカラー演算の `'fp32'`、Tensor Core を用いる `'tf32'`、および混合精度の `'bf16'` にわたって実行しよう。

```
    with jax.default_matmul_precision('highest'):  # True fp32
        report('fp32', lambda: step(*args, jnp.float32))
    report('tf32', lambda: step(*args, jnp.float32))
    report('bf16', lambda: step(*args, jnp.bfloat16))
```

```
fp32: 21.72 ms,  51 TFLOP/s
tf32: 14.30 ms,  77 TFLOP/s
bf16:  8.28 ms, 133 TFLOP/s
```

スカラー演算器上の真の fp32 は、積和演算とアドレス計算のすべてが命令発行スロットを奪い合うため、ピークである 82.6 TFLOP/s の約 60% にとどまる。一方、Tensor Core 上の tf32 は同じピークの約 90% に達し、保存される型を変えることなくステップ時間を 3 分の 1 短縮する。さらに tf32 から bf16 へ移行すると、ステップはもう 1.5–1.75× 高速化する（fp32 の SGD 更新と重みのキャストは縮小しないため、下限と更新時間を合わせた $6.7 + 1.3 = 8.0$ ms に近づく）。

13.4.3 活性化チェックポイント

カードに収まるトークン数を何倍にも増やすには、1 ステップで保存する活性化値を減らさなければならない。**活性化チェックポイント**（5.7.3.3 節、(Chen et al. 2016)）は、順伝播中に各 Transformer ブロックの入力（トークンあたり d 個の fp32 値、すなわち $4d$ バイト）のみを保存し、逆伝播中にブロック内部の活性化値を 1 ブロックずつ再計算する。これにより、余分な順伝播 1 回分（約 33% の計算量増であり、トークンあたり $8N$ FLOPs と $6N$ から増加）と引き換えに、L ブロックにわたるトークンあたりのピークメモリを $La + a_{\text{out}} + \Delta$ から高々次の値まで削減する。

$$L \cdot 4d + a + a_{\text{out}} + \Delta,$$

ここで、a は現在再計算されている単一ブロックの分であり、Δ は集計値に対する実測ピークの超過分である（13.4.1.4 節）。

```
    M_gpu, d, L = 24 * 2**30, config['num_hiddens'], config['num_blks']
    slope = (peaks[8192] - peaks[4096]) / 4096  # Growth per token, measured
    excess = slope - (2 * block + head)  # Measured peak above the count
    budgets = {  # Configuration: (fixed bytes, bytes per token at the peak)
        'fp32, count': (16 * N, L * block + head),
        'fp32, measured': (16 * N, L * block + head + excess),
        'bf16, measured': (18 * N, L * block16 + head16 + excess),
        'fp32, checkpointing': (16 * N, L * 4 * d + block + head + excess),
        'bf16, checkpointing': (18 * N, L * 4 * d + block16 + head16 + excess)}
    for name, (fixed, per_token) in budgets.items():  # M(n) <= M_gpu
        print(f'{name:19s}: {per_token / 2**20:.2f} MiB per token, room for '
              f'{(M_gpu - fixed) / per_token:6,.0f} tokens')
```

```
fp32, count        : 4.30 MiB per token, room for  2,127 tokens
fp32, measured     : 4.30 MiB per token, room for  2,128 tokens
bf16, measured     : 2.68 MiB per token, room for  2,691 tokens
fp32, checkpointing: 0.59 MiB per token, room for 15,529 tokens
bf16, checkpointing: 0.50 MiB per token, room for 14,461 tokens
```

ブロックごとのチェックポイントを用いると、18 ブロック全体で保存されるのはトークンあたりわずか $18 \times 8 = 144$ KiB（出力層と損失だけの分よりも小さい）となり、トークンバジェット

は 3 倍以上（10,000〜15,000 トークン）に拡大する。なお、チェックポイント下では bf16 の方が fp32 よりも収まるトークン数が実際に**少なく**なることに注目したい。bf16 の重みコピーによる追加の $2N$ バイトが、（PyTorch では）より大きくなる bf16 の出力層および損失と相まって、再計算される単一ブロックでの活性化値の節約分を上回るためである。この段階では、出力ロジットと交差エントロピー損失がトークンあたりのメモリを支配するようになる。出力射影と損失を独自のチェックポイント内でトークンのチャンクごとに計算すれば、わずか $2dV/(6N) \approx 3\%$ の追加 FLOPs でそのボトルネックを解消できる（演習問題 6）。

長さ 1,024 トークンの系列 2 本を用いて、4 ブロックのスタック上でブロックごとのチェックポイントを計測してみよう（4 つのブロック入力と最終出力を合わせて、トークンあたり $5 \times 4d = 40$ KiB が保存されるはずである）。

```
blocks = [nnx.split(blk, nnx.Param, ...) for blk in d2l.GPT(
    **{**config, 'num_blks': 4}, rngs=nnx.Rngs(d2l.get_key())).blks]
blk_params = [p for _, p, _ in blocks]
def stack(params, H, remat):
    def block(graphdef, p, rest, H):  # A block as a function of its weights
        return nnx.merge(graphdef, p, rest)(H)
    if remat:
        # On a CPU, XLA's default scheduler runs the recomputations early,
        # so that they coexist and the saving vanishes; setting XLA_FLAGS to
        # --xla_cpu_enable_concurrency_optimized_scheduler=false before JAX
        # starts restores it there
        block = jax.checkpoint(block, static_argnums=0)
    for (graphdef, _, rest), p in zip(blocks, params):
        H = block(graphdef, p, rest, H)
    return (H ** 2).mean()

d = config['num_hiddens']
H1, H2 = (jax.random.normal(d2l.get_key(), (b, 1024, d)) for b in (1, 2))
for remat in (False, True):
    f = lambda params, H: stack(params, H, remat)
    stored = (saved_bytes(f, blk_params, H2)
              - saved_bytes(f, blk_params, H1)) / 1024
    stack_grad = jax.jit(jax.grad(f))
    peak = d2l.peak_memory(stack_grad, blk_params, H2)
    t = d2l.Benchmark(lambda: stack_grad(blk_params, H2)).time
    print(f'checkpointing={remat!s:5}: stores {stored / 2**10:4.0f} KiB per '
          f'token, peak {peak / 2**30:.2f} GiB, {1000 * t:.0f} ms per step')
```

```
checkpointing=False: stores  936 KiB per token, peak 0.78 GiB, 45 ms per step
checkpointing=True : stores   40 KiB per token, peak 0.45 GiB, 57 ms per step
```

チェックポイントにより、保存されるテンソルは予測どおりちょうどトークンあたり 40 KiB に減少し、ピークの活性化メモリは大幅に削減され、ステップ時間は約 25% 延びる（余分な順伝播 1 回分の上限である 33% 以内に収まっている）。ブロック全体を再計算する代わりに、**選択的再計算**（selective recomputation）(Korthikanti et al. 2022) は高コストな行列積の出力を保存し、SwiGLU のゲート積のような低コストな要素ごとの活性化値のみを再計算する（演習問題 5）。

13.4.4 勾配累積

より多くの活性化値を保存することなく数十万トークンのグローバルバッチ $B_{\text{global}} = m\, B_{\text{micro}}$ に到達するため、**勾配累積**はグローバルバッチを m 個の逐次的なマイクロバッチに分割し、それ

らの勾配を累積して（各マイクロバッチの平均損失を m で割る）、最後にオプティマイザを 1 回実行する。

勾配累積は単一の大きなバッチと同じ算術演算を実行するが、2 番目以降の各マイクロバッチは累積中の勾配アキュムレータと新しい勾配を読み出してその和を書き戻す（$12N$ バイトのメモリトラフィック）一方、マイクロバッチ自体は $6Nn$ FLOPs の演算を n トークンに対して実行する。ピーク計算時間に対する比として、このアキュムレータのトラフィックは次のコストを要する。

$$\frac{12N/\beta}{6Nn/P} = \frac{2I^*}{n},$$

ここで、$I^* = P/\beta$ はリッジポイント（ridge point、13.1 節）である。tf32（$I^* = 82$）でマイクロバッチが $n = 1{,}024$ トークンの場合、そのオーバーヘッドは累積を行う $m - 1$ 個の各マイクロバッチで約 16% となるが、$n = 4{,}096$ トークンでは 4% に低下する。

2 ブロックの GPT において、4,096 トークンを 1 バッチとして処理する場合と、1,024 トークンのマイクロバッチ $m = 4$ 個として処理する場合とで、数値的な等価性ならびにメモリと時間のオーバーヘッドの両方を検証する。

```
model = d2l.GPT(**{**config, 'num_blks': 2}, rngs=nnx.Rngs(d2l.get_key()))
graphdef, params, rest = nnx.split(model, nnx.Param, ...)
X = jax.random.randint(d2l.get_key(), (4, 1024), 0, config['vocab_size'])

def accumulate(params, rest, X, m):
    """Gradient of the loss on X, accumulated over m micro-batches."""
    loss = lambda params, Xb: gpt_loss(nnx.merge(graphdef, params, rest), Xb)
    def add(g, Xb):  # One micro-batch; / m because the loss averages
        g_b = jax.grad(loss)(params, Xb)
        return jax.tree.map(lambda a, b: a + b / m, g, g_b), None
    zeros = jax.tree.map(jnp.zeros_like, params)
    return jax.lax.scan(add, zeros, X.reshape(m, -1, X.shape[-1]))[0]

accumulate = jax.jit(accumulate, static_argnums=3)
full = accumulate(params, rest, X, 1)  # One-batch gradient
for m in (1, 4):
    plan = accumulate.lower(params, rest, X, m).compile().memory_analysis()
    peak = plan.output_size_in_bytes + plan.temp_size_in_bytes
    assert max(jax.tree.leaves(jax.tree.map(
        lambda a, b: jnp.linalg.norm(a - b) / jnp.linalg.norm(b),
        accumulate(params, rest, X, m), full))) < 1e-2
    f = lambda: accumulate(params, rest, X, m)
    times = 1000 * np.array([d2l.Benchmark(f).time for _ in range(5)])  # ms
    kernels = d2l.kernel_times(f)        # Device ms per step, per kernel
    busy = sum(kernels.values())
    products = sum(ms for name, ms in kernels.items()
                   if re.search(d2l.PRODUCT_KERNELS, name, re.I))
    print(f'{m} x {4096 // m:4d} tokens: peak {peak / 2**30:.2f} GiB above '
          f'the weights, {np.median(times):.0f} ms per step '
          f'({times.min():.0f} to {times.max():.0f}), device busy '
          f'{busy:.0f} ms\n{"":16s}matrix products {products:.0f} ms, '
          f'other kernels {busy - products:.0f} ms')
```

```
1 x 4096 tokens: peak 3.09 GiB above the weights, 80 ms per step (80 to 80), device busy 80 ms
                 matrix products 67 ms, other kernels 13 ms
4 x 1024 tokens: peak 1.43 GiB above the weights, 90 ms per step (89 to 90), device busy 89 ms
                 matrix products 69 ms, other kernels 21 ms
```

重みを除いたピークメモリは約半分に低下し（$4N = 0.77$ GiB の持続的な勾配がマイクロバッチ間で確保されたまま残る一方、活性化メモリが 4 分の 1 に減少するため）、アキュムレータの加算および小さなマイクロバッチでの Tensor Core 効率のわずかな低下により、ステップ時間は 10〜20% 増加する。したがって、最適なマイクロバッチは常にメモリに収まる最大のものとなる。

13.4.5 メモリ手法の選択

Table 13.4 は、それぞれの手法を必要とする症状別にメモリ手法をまとめたものである。

Table 13.4: 症状別のメモリ手法、ならびに Equation 13.6 への効果とコスト。

症状	手法	メモリへの効果	コスト
行列積がステップ時間を支配	bf16 混合精度	活性化値 $\times 0.6$、重みコピーにより固定状態 $+2N$	8 ビット仮数部の丸め（fp16 は損失スケーリングも必要）
有用なマイクロバッチが収まらない	活性化チェックポイント	ブロックあたりの保存活性化値をトークンあたり a から $4d$ バイトへ削減	余分な順伝播 1 回分（時間が約 25〜33% 増加）
出力層と損失が支配	チャンク化交差エントロピー損失	a_{out} と Δ を一度に 1 トークンチャンク分に限定	出力射影の余分なパス 1 回分（FLOPs が約 3% 増加）
グローバルバッチがマイクロバッチ容量を超過	勾配累積	なし（グローバルバッチが m 倍に拡大）	アキュムレータのトラフィック $2I^*/n$、マイクロバッチごとの重み再読み出し
$16N$ のパラメータ状態だけでカードが埋まる	重み・勾配・オプティマイザ状態のシャーディング（13.7 節）	各 GPU 上で $16N/k$（k 基に分割）	GPU 間の集団通信

単一の RTX 4090 上の 1B パラメータ GPT では、bf16 混合精度によりチェックポイントなしで 1,024 トークンの系列が 2〜3 本収まり、256 個のマイクロバッチを累積することで 524,288 トークンのグローバルバッチに到達する。しかし、トークンあたり $6N$ FLOPs では、1 回のグローバルステップに 3.2×10^{15} FLOPs を要し、カード 1 枚ではステップあたり少なくとも 19 s、100,000 ステップ（52B トークン）では 3 週間以上かかる。単一の GPU にこの GPT の 1 ステップを収めることはできても、訓練を完走するには複数の GPU が必要となる（13.5 節）。

13.4.6 まとめ

訓練ステップのメモリは、固定のパラメータ項（fp32 の Adam では $16N$ バイト、bf16 の重みコピーを伴う場合は $18N$）と、トークンあたりの活性化項に分かれる。3 つの微分規則（線形層は入力を保持し、要素ごとの関数は入力または出力を保持し、積は両方の因子を保持する）により、融合された SwiGLU Transformer ブロックはトークンあたり約 $20d$ 個の値を保存し、交差エントロピー損失は V 個の対数確率を保存する。実際のピークは損失部分での逆伝播の一時配列によってこの集計値を上回り得るため、実測が必要となる。混合精度は Tensor Core 上での行列積を高速化する一方で総メモリ容量をほとんど変化させず、活性化チェックポイントは約 25% の追加時間と引き換えにブロック入力（トークンあたり $4d$ バイト）のみを保存し、勾配累積はメモリに

収まるマイクロバッチから大きなグローバルバッチを構築し、$2I^*/n$ の割合でアキュムレータのトラフィックを追加する。

13.4.7 演習問題

1. **ピークが生じる位置**。@fig-memory-anatomy では、訓練ステップのピークを逆伝播の開始時点に置いている。n 個の同一な層からなるスタックを考え、各層のパラメータが w バイトを占有し、その順伝播が a バイトの活性化値を保存するものとする。逆伝播の開始時点では勾配は存在せず、各層の逆伝播はその a バイトの活性化値を解放して w バイトのパラメータ勾配を確保するものと仮定する。
   1. 逆伝播が j 個の層（全 n 層中）を処理した後に活性化値と勾配が保持しているバイト数を書き下せ。また、ピークが $a > w$ のときは逆伝播の開始時点にあり、$a < w$ のときはその終了時点にあることを示せ。
   2. 本節の GPT の 1 ブロックは約 $12d^2$ 個のパラメータを保持し、トークンあたり約 $20d$ 個の fp32 値を保存する。ステップあたりのトークン数がいくつを超えると、ピークが逆伝播の開始時点にくるか。
   3. m 個のマイクロバッチにわたる勾配累積のもとでは、最初のマイクロバッチの勾配が後続のマイクロバッチの実行中も確保されたまま残る。勾配累積を用いた場合のピークを求め、単一のフルバッチステップよりもピークを低くするような m の値を求めよ。
2. [code] **オプティマイザステップのメモリ**。13.4.1.4 節のコンパイル済み勾配計算を、`optax.adam` を用いた完全な Adam 更新へと拡張せよ。
   1. 512 トークンに対してステップをコンパイルし、`argument_size_in_bytes`、`output_size_in_bytes`、および `temp_size_in_bytes` を `memory_analysis()` から取得して $4N$ 単位で報告せよ。
   2. パラメータとオプティマイザ状態を対象に `donate_argnums` を指定して再コンパイルし、`alias_size_in_bytes` を報告したうえで、バッファ譲渡がピークメモリをどのように変化させるか説明せよ。
3. **勾配累積下でのクリッピング**。ノルム τ への勾配クリッピングは、$\mathbf{g}$ を $\mathbf{g}\min(1, \tau/\|\mathbf{g}\|)$ に置き換える。$\mathbf{g}_1, \dots, \mathbf{g}_m$ を $1/m$ 倍にスケールされたマイクロバッチ勾配とし、その和 $\mathbf{g}$ がグローバルバッチ勾配であるとする。
   1. ループ内で各 $\mathbf{g}_i$ をアキュムレータに加算する前にクリッピングする場合、累積された勾配が到達し得る最大ノルムはいくらか。また、マイクロバッチごとのクリッピングがグローバルなクリッピングと一致するのはどのようなときか。
   2. 係数 S の fp16 損失スケーリングのもとでは、最後のマイクロバッチの後にアキュムレータは $S\mathbf{g}$ を保持している。ループは S によるアンスケール、クリッピング、およびオプティマイザの更新ステップをどの順序で実行しなければならないか。また、マイクロバッチ間で S が変化すると何が問題になるか。
4. [code] **fp16 の勾配ウィンドウ**。●fp16 は 5 ビットの指数部を持つため、2^{-14}〜65,504 の正規化数と 2^{-24} までの非正規化数を表現できる。損失スケーリング (Micikevicius et al. 2018) は、微分の前に損失に S を掛け、その後で勾配を S で割る。真の fp32 による参照勾配とともに、`mlp_params`、`X_mlp`、`y_mlp`、`loss_fn`（13.4.2 節）を用いて以下を行え。
   1. 参照勾配のすべての非ゼロ要素にわたって $\log_2 |g|$ のヒストグラムをプロットし、スケール係数 S として、スケール後の要素 Sg の 99.9% を fp16 の正規化数の範囲内に収めるものを求めよ。
   2. $S = 2^0, 2^2, \dots, 2^{24}$ に対して、`jnp.float16` で勾配を計算し、S でアンスケールして相対誤差 $\|\hat{\mathbf{g}} - \mathbf{g}\|/\|\mathbf{g}\|$ をプロットせよ。`jnp.bfloat16` でも同様に行え。S のどの範囲で fp16 は bf16 よりも高精度になるか、またそれはなぜか。
   3. バッチサイズが 2,048 から 32,768 に増えると、有効な S の範囲はどのようにシフトするか。

5. [code] **選択的再計算**。●`jax.checkpoint` は `policy` 引数として `jax.checkpoint_policies.dots_with_no_batch_dims_saveable` などを受け取り、重み行列積の出力を保存して要素ごとの演算を再計算する。
   1. 本節の GPT の 1 ブロックについて、チェックポイントなし、デフォルトの `jax.checkpoint`、および `dots_with_no_batch_dims_saveable` のもとで、保存される残差と再計算される行列積を `jax.ad_checkpoint.print_saved_residuals` で確認しながら列挙せよ。
   2. このポリシーを 13.4.3 節の 4 ブロックベンチマークに追加し、トークンあたりの保存バイト数、計画ピークメモリ、およびステップ時間を比較せよ。
6. [extended] **チャンク単位の損失計算**。●●ブロックごとのチェックポイントのもとでは、本節の GPT の出力層と交差エントロピー損失は、損失がトークンあたり $V = 50{,}257$ 個の対数確率とその勾配を実体化するため、全 18 ブロックの合計よりもトークンあたり多くのメモリを消費する。
   1. トークンをサイズ $n_c = 1024$ のチャンクごとに処理するよう出力射影と交差エントロピー損失を記述し、ロジットと対数確率が一度に 1 チャンク分しか存在しないよう各チャンクをチェックポイントでラップせよ。損失と勾配がチャンク化しない版と一致することを検証せよ。
   2. ブロックごとのチェックポイントとチャンク化損失を用いた場合の、24 GiB カード上での 18 ブロック GPT の最大マイクロバッチサイズを予測せよ。また、2 ブロック GPT 上でトークンあたりのメモリ増加率とステップ時間を計測せよ。

13.5 第一原理からのマルチ GPU

これまでの訓練実行ではすべて 1 基の GPU しか使ってこなかったが、本書のビルド用ワークステーションには 4 基搭載されている。k 基すべての GPU を使う最も単純な方法は**データ並列**（data parallelism）である。すなわち、すべての GPU にモデルの完全なレプリカと各ミニバッチの $1/k$ を与え、得られた勾配を **allreduce** によって GPU 間で合計し、すべてのカードで同じパラメータ更新を適用する。

このワークステーションでは、GPU 間通信はホスト DRAM を経由して中継される PCIe 4.0 を通るため（13.2.4.2 節）、各リンクは公称 32 GB/s に制限される。これはカード上のオンボードメモリの 1 TB/s より約 $30\times$ 遅い。2 基目の GPU がステップを短縮できるのは、引き受けた計算にかかる時間が、新たに加わる勾配交換（およびホストディスパッチのオーバーヘッド）を上回る場合に限られる。本節では、データ並列を手作業で構築し、集団通信ライブラリで使われる**リング allreduce** アルゴリズムを導出し、個別に測定した定数からマルチ GPU のステップ時間を予測するコストモデルを構築する。このモデルの検証には Fashion-MNIST 上の LeNet を用いる。そのステップはごく小さいため、ホストや通信の固定オーバーヘッドが鮮明に現れる。

```
import time
from functools import partial
from jax.flatten_util import ravel_pytree
from jax.sharding import NamedSharding, PartitionSpec as P
```

13.5.1 3 つの分割方法

モデルとそのミニバッチを複数のデバイスに分割する基本的な方法は 3 つある（Figure 13.15）。

- **データ並列**は、すべてのデバイスにモデルを複製し、ミニバッチをデバイス間で分割して、ステップごとに 1 回だけ勾配を合計する。単一 GPU に収まる任意のモデルで動作するが、各デバイスは依然として fp32 Adam の訓練状態 $16N$ バイトすべてを保持する（13.4 節）。

- **パイプライン並列**（pipeline parallelism）は、層のスタックをデバイス間で分割し、ステージ境界で順方向に活性化値を、逆方向に勾配を受け渡す（演習問題 1）。
- **テンソル並列**（tensor parallelism）は、個々の重み行列をデバイス間で分割し（2 基の 3 GB GPU にまたがる 2012 年の AlexNet のように (Krizhevsky et al. 2012))、層ごとに複数回通信する（演習問題 3、13.7 節）。

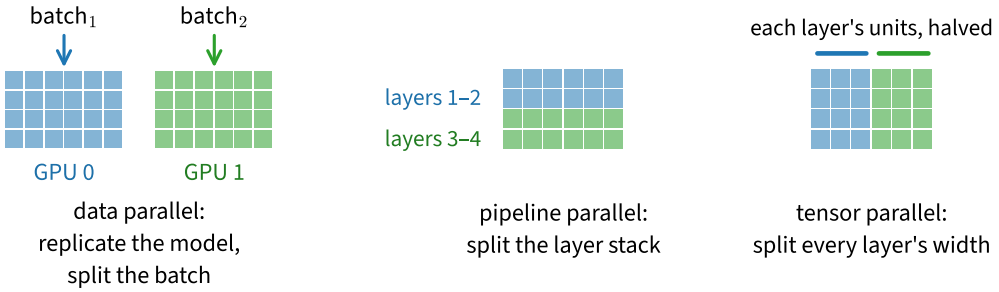


Figure 13.15: モデルを 2 つのデバイスに分割する 3 つの方法（各デバイスのセルを固有の色で示す）。モデルを複製してバッチを分割する（データ並列）、層のスタックを分割する（パイプライン並列）、各層の幅を分割する（テンソル並列）。

13.5.2 手作業によるデータ並列

k 個のデバイス上でのデータ並列の 1 ステップは 4 つのステージからなる（Figure 13.16）。ミニバッチを k 個の均等なシャードに分割し、各シャードの勾配をそれぞれのデバイス上で計算し、すべてのデバイスがその和を保持するよう allreduce で k 個の勾配を合計し、すべてのデバイスで同一のパラメータ更新を適用する。すべてのレプリカが同一のパラメータから開始して同一の更新を適用するため、重みを再コピーしなくても同期された状態が保たれる。

13.5.2.1 モデルとデータ

デバイス間のコピーがすべて見えるように、LeNet（6.6 節、ReLU 活性化関数と He 初期化を使用）をパラメータテンソルの明示的なリストを受け取る関数として記述する。その 61,706 個の fp32 パラメータは 247 KB、Fashion-MNIST の 60,000 枚の訓練画像は 188 MB に収まる。これはデータセットのコピーを各 GPU 上に保持できるほど小さいため、ホストからのデータ読み込みが計測時間に入り込むことはない。

```
def new_params(key):
    """LeNet's parameters: He-initialized weights, zero biases."""
    shapes = [(5, 5, 1, 6), (5, 5, 6, 16), (400, 120), (120, 84), (84, 10)]
    params = []
    for key, shape in zip(jax.random.split(key, len(shapes)), shapes):
        fan_in = math.prod(shape[:-1])
        params += [jax.random.normal(key, shape) * math.sqrt(2 / fan_in),
                   jnp.zeros(shape[-1])]
    return params

def lenet(params, X):
    W1, b1, W2, b2, W3, b3, W4, b4, W5, b5 = params
    conv = lambda X, W, padding: jax.lax.conv_general_dilated(
```

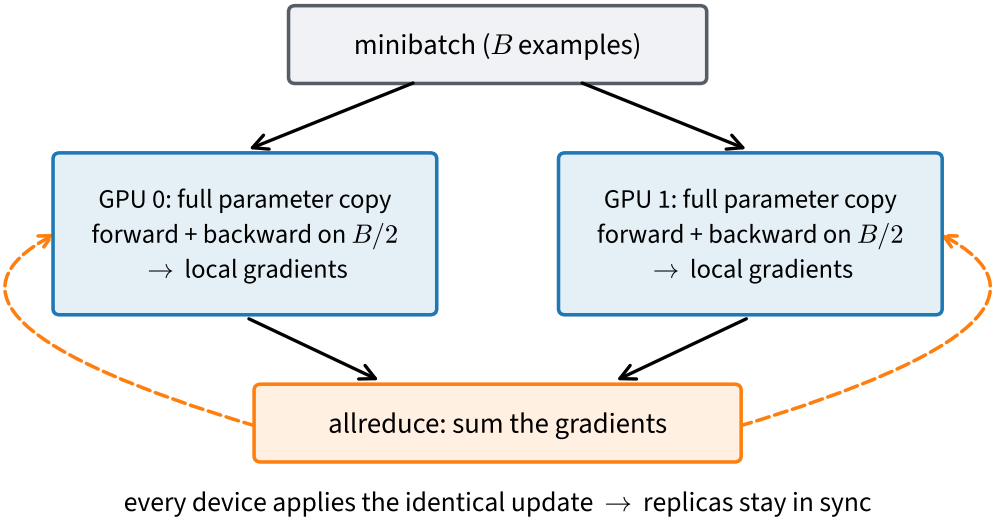


Figure 13.16: 2 基の GPU 上でのデータ並列の 1 ステップ。ミニバッチを分割し、各レプリカで同一の順伝播と逆伝播を実行し、allreduce で勾配を合計してから、すべてのデバイスで同一の更新を適用する。

```
        X, W, (1, 1), padding, dimension_numbers=('NHWC', 'HWIO', 'NHWC'))
    pool = lambda X: nnx.avg_pool(X, (2, 2), strides=(2, 2))
    H = pool(jax.nn.relu(conv(X, W1, ((2, 2), (2, 2))) + b1))
    H = pool(jax.nn.relu(conv(H, W2, 'VALID') + b2)).reshape(X.shape[0], -1)
    H = jax.nn.relu(H @ W3 + b3)
    H = jax.nn.relu(H @ W4 + b4)
    return H @ W5 + b5

def to_device(split, sharding):
    """A whole Fashion-MNIST split placed by `sharding`: images in [0, 1],
    labels."""
    X, y = split
    return jax.device_put((X[..., None] / np.float32(255),
                           y.astype(np.int32)), sharding)
```

少なくとも 2 基の GPU が利用可能であることを確認し、Fashion-MNIST の訓練分割を両方のデバイスに複製したうえで（train_data）、以降のすべての実験で共有する単一のパラメータ初期化 base を生成する。

```
assert jax.device_count() >= 2, 'This section needs two GPUs'
data = d2l.FashionMNIST()
meshes = [jax.make_mesh((k,), ('data',), devices=jax.devices()[:k])
          for k in (1, 2)]
train_data = [to_device(data.train, NamedSharding(mesh, P()))
              for mesh in meshes]  # A copy on every device of each mesh
test_data = (data.val[0][..., None] / np.float32(255),
```

```
                data.val[1].astype(np.int32))
base = new_params(d2l.get_key())  # One initialization for every run
print(f'{sum(p.size for p in base)} parameters in {len(base)} tensors; '
      f'images {train_data[1][0].shape} on each of 2 devices')
```

```
61706 parameters in 10 tensors; images (60000, 28, 28, 1) on each of 2 devices
```

JAX では、マルチデバイス配列は NamedSharding と PartitionSpec を介して名前付きデバイス**メッシュ**上に配置される（P('data') は先頭軸を 'data' メッシュ軸に沿って分割し、P() は配列全体をすべてのデバイスに複製する）。

13.5.2.2 ブロードキャストと allreduce

ブロードキャストは初期パラメータをすべてのデバイスに 1 度だけコピーし、**allreduce** はデバイスごとに 1 つのテンソルを合計して、その総和をすべてのデバイスに残す。

ここでは jax.shard_map と jax.lax.psum（内部で NCCL に委譲する）を使用する。

```
def broadcast(params, mesh):
    """A copy of every array in `params` on every device of `mesh`."""
    return jax.device_put(params, NamedSharding(mesh, P()))

@partial(jax.jit, static_argnames='mesh')
def allreduce(x, mesh):
    """Sum the rows of `x` (row j on device j); every device gets the sum."""
    return jax.shard_map(lambda x_j: jax.lax.psum(x_j, 'data'), mesh=mesh,
                         in_specs=P('data'), out_specs=P('data'))(x)
```

allreduce をテストするため、デバイス 0 に [1.0, 1.0]、デバイス 1 に [2.0, 2.0] を配置し、両方のデバイスが要素ごとの和 [3.0, 3.0] を受け取ることを確認しよう。

```
x = jax.device_put(jnp.array([[1.0, 1.0], [2.0, 2.0]]),
                   NamedSharding(meshes[1], P('data')))
print('before:', x.tolist())
print('after: ', allreduce(x, meshes[1]).tolist())
```

```
before: [[1.0, 1.0], [2.0, 2.0]]
after:  [[3.0, 3.0], [3.0, 3.0]]
```

13.5.2.3 データ並列ステップ

同じサイズの k 個のシャードがあるとき、シャードごとの平均損失の勾配 k 個の平均は、ミニバッチ全体の勾配に一致する（演習問題 2）。allreduce の呼び出しにはメッセージサイズによらず固定のレイテンシがかかるため、LeNet の 10 個の勾配テンソルを単一の平坦なベクトルに連結し、ステップごとに 1 回だけ allreduce を実行する。communicate=False を渡すと allreduce がスキップされ、純粋なデバイス計算の時間を計測できる。

```
def shard_grad(params, X, y, i):
    """Inside shard_map: the gradient of the mean loss on this device's
    share `i` of the minibatch, as one flat vector."""
```

```
    def loss(p):
        return optax.softmax_cross_entropy_with_integer_labels(
            lenet(p, X[i]), y[i]).mean()
    own = jax.lax.pcast(params, 'data', to='varying')  # This device's copy
    return ravel_pytree(jax.grad(loss)(own))[0]

@partial(jax.jit, static_argnames=('mesh', 'communicate'))
def train_step(params, data, idx, lr, mesh, communicate=True):
    """One data-parallel SGD step on the minibatch `idx`; without
    communication, every device steps on its own shard's gradient."""
    unravel = ravel_pytree(params)[1]
    def step(params, X, y, i):
        g = shard_grad(params, X, y, i)
        if communicate:
            g = jax.lax.psum(g, 'data')              # Allreduce
        k = jax.lax.axis_size('data')
        return jax.tree.map(lambda p, g_p: p - lr / k * g_p, params,
                            unravel(g))
    out = P() if communicate else P('data')  # Copies differ without the sum
    return jax.shard_map(step, mesh=mesh, in_specs=(P(), P(), P(), P('data')),
                         out_specs=out)(params, *data, idx)
```

13.5.2.4 1 ステップ検証

実行時間を計測する前に、2 基の GPU 上でのデータ並列の 1 ステップが、同じ 256 個の例に対する単一 GPU 上での 1 ステップと同じパラメータ更新を生成することを確認する（tf32 下での畳み込みアルゴリズムの違いによって k で割り忘れるなどのバグが隠れないよう、厳密な fp32 演算を用いる）。

```
idx = np.random.default_rng().permutation(len(data.train[1]))[:256]
updates = []
with jax.default_matmul_precision('highest'):  # True fp32
    for mesh, data_k in zip(meshes, train_data):
        i = jax.device_put(idx, NamedSharding(mesh, P('data')))
        params = train_step(broadcast(base, mesh), data_k, i, 0.2, mesh)
        updates.append([np.asarray(p) - np.asarray(b)
                        for p, b in zip(params, base)])
largest = max(np.abs(u).max() for u in updates[0])
diff = max(np.abs(u1 - u2).max() for u1, u2 in zip(*updates))
print(f'largest update {largest:.1e}, largest difference {diff:.1e}')
```

```
largest update 1.9e-02, largest difference 3.0e-08
```

単一 GPU と 2 基の GPU による更新は、fp32 の丸め誤差（約 10^{-7}）の範囲で一致する。

13.5.3 リング allreduce

スター allreduce はすべての通信トラフィックをデバイス 0 経由でルーティングするため、ハブとなるデバイスは $(k-1)S$ バイト（勾配のサイズを S とする）を送受信しなければならない。デバイスを追加するたびにハブの負荷は増大する一方、他の各デバイスが扱う通信量は双方向とも S バイトにとどまる（Figure 13.17 左）。

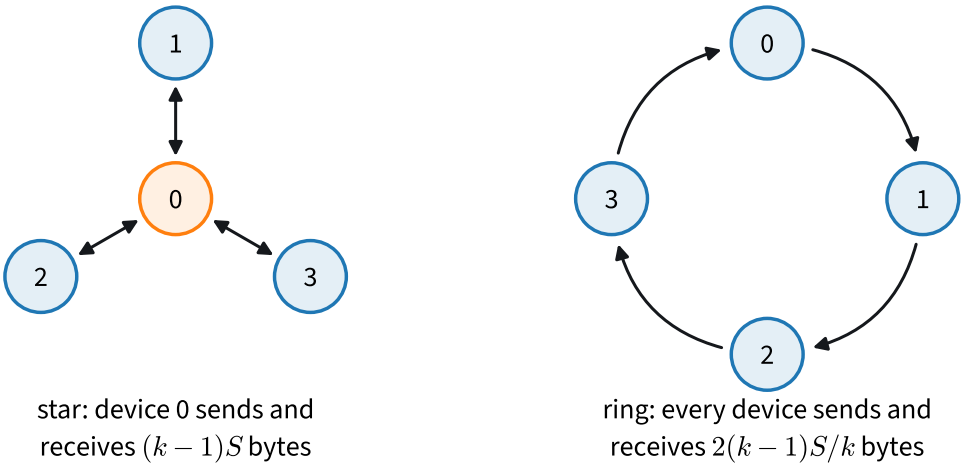


Figure 13.17: $k = 4$ におけるスター allreduce とリング allreduce の通信量。スターのハブは $(k-1)S$ バイトを送受信するが、リングではどのデバイスも $2(k-1)S/k$ バイトを送受信し、これは $2S$ 未満に収まる（任意の k に対して成立）。

リング allreduce (Patarasuk and Yuan 2009; Sergeev and Del Balso 2018) はこのハブをなくす。k 個のデバイスを論理的なリング状に並べてトラフィックを均等に分散させ、各デバイスの S バイトの勾配を k 個のチャンク（各 S/k バイト）に分割し、それぞれ $k-1$ ステップからなる 2 つのフェーズを実行する（Figure 13.18）。

1. **reduce-scatter**：$k-1$ ステップのそれぞれにおいて、すべてのデバイスは後続のデバイスに 1 つのチャンクを送り、先行のデバイスから受け取ったチャンクを自身のローカルコピーに加算する。$k-1$ ステップ後、各デバイスは互いに異なる 1 つのチャンクについて完全な総和を保持する。
2. **all-gather**：続く $k-1$ ステップにわたり、加算が完了したチャンクをリングに沿って巡回させ、すべてのデバイスが合計済みの k 個のチャンクすべてを保持する状態にする。

各デバイスは 1 フェーズあたり $(k-1)S/k$ バイトを送信するため、合計の送信量は次のようになる。

$$\frac{2(k-1)}{k}\,S \quad \text{bytes per device.} \tag{13.7}$$

デバイスあたりの通信量は S（$k = 2$ の場合）であり、リングに加わるデバイスがいくつになっても厳密に $2S$ 未満にとどまる。さらに、これら 2 つのフェーズは単独でも有用である。シャーディングを用いた訓練（13.7 節）では、合計された勾配を分割するのに reduce-scatter を使い、シャード化された重みを集めるのに all-gather を使う。

この 2 フェーズのリングを NumPy で実装し、各デバイス上で累積されるチャンクと、デバイスあたりの合計送信バイト数 `sent` の両方を追跡する。

```
def ring_allreduce(xs):
    """Ring allreduce of the vectors xs, one per device; returns every
    device's result and the bytes that each device sent."""
```

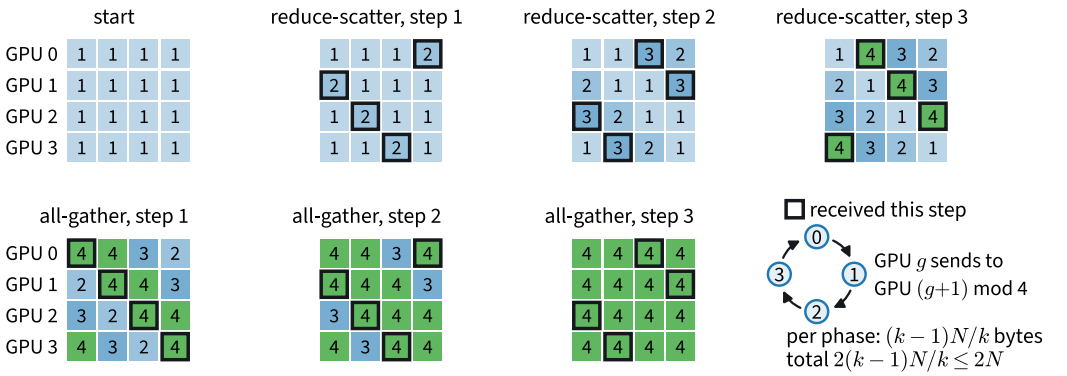


Figure 13.18: 4 基の GPU 上でのリング allreduce をステップごとに計算した様子。reduce-scatter はリングに沿ってデバイスごとに 1 つのチャンクを累積し、続いて all-gather が完成したチャンクを配布する。各セルの数字はそのチャンクが 4 つ分の寄与のうちいくつを保持しているかを表し、4 になったチャンクで加算が完了する。枠線付きのセルはそのステップで GPU が受信したチャンクであり、最後のパネルのリングはどの GPU がどの GPU へ送信するかを示す。

```
    k = len(xs)
    chunks = [np.array_split(x.copy(), k) for x in xs]  # chunks[j][c]
    sent = 0
    for s in range(k - 1):  # Reduce-scatter: add the received chunk
        msgs = [chunks[j][(j - s) % k].copy() for j in range(k)]
        for j, msg in enumerate(msgs):
            chunks[(j + 1) % k][(j - s) % k] += msg
        sent += msgs[0].nbytes
    for s in range(k - 1):  # All-gather: replace by the received chunk
        msgs = [chunks[j][(j + 1 - s) % k].copy() for j in range(k)]
        for j, msg in enumerate(msgs):
            chunks[(j + 1) % k][(j + 1 - s) % k] = msg
        sent += msgs[0].nbytes
    return [np.concatenate(c) for c in chunks], sent
```

ランダムな 1,024 要素の float ベクトルを用いて `ring_allreduce` を $k \in \{2, 4, 8, 16\}$ 個のデバイスでテストすると、すべてのデバイスが正確な総和を受け取りつつ、送信量が $2(k-1)S/k$ バイトになっていることが確認できる。

```
rng = np.random.default_rng()
for k in (2, 4, 8, 16):
    xs = [rng.standard_normal(1024, dtype=np.float32) for _ in range(k)]
    results, sent = ring_allreduce(xs)
    correct = all(np.allclose(r, sum(xs), atol=1e-5) for r in results)
    print(f'k = {k:2d}: every device holds the sum: {correct}; '
          f'sent per device: {sent / xs[0].nbytes:.2f} S')
```

```
k =  2: every device holds the sum: True; sent per device: 1.00 S
k =  4: every device holds the sum: True; sent per device: 1.50 S
k =  8: every device holds the sum: True; sent per device: 1.75 S
k = 16: every device holds the sum: True; sent per device: 1.88 S
```

デバイスあたりの通信バイト数は $2S$ 未満に抑えられるものの、リングでは $2(k-1)$ 回の逐次ホップを実行し、各ホップで起動レイテンシを支払うことになる。多数の GPU 上で小さなメッセージをやり取りする場合、集団通信ライブラリは深さ $\log_2 k$ の木アルゴリズムに切り替える（演習問題 5）。

13.5.4 通信コストとスケーリング

13.5.4.1 1 ステップのコストモデル

レイテンシ・帯域幅モデル (Hockney 1994; Thakur et al. 2005) を用いると、各 allreduce 呼び出しは固定の起動レイテンシ α に加え、各デバイスが送信する 1 バイトあたり β_{link} の時間（allreduce の**バス帯域幅**（bus bandwidth）の逆数）を要する。通信と計算のオーバーラップがない場合、k 個のデバイス上でグローバルバッチ B と c 回の allreduce 呼び出しを伴う 1 ステップは次の時間を要する。

$$t_{\text{step}}(k) \approx t_{\text{compute}}(B/k) + c\,\alpha + \frac{2(k-1)}{k}\,S\,\beta_{\text{link}}, \tag{13.8}$$

ここで、$c = 0$（$k = 1$ のとき）である。2 つ目のデバイスがステップを短縮できるのは、$t_{\text{compute}}(B) - t_{\text{compute}}(B/2) > c\alpha + S\,\beta_{\text{link}}$ が成り立つ場合に限られる。

13.5.4.2 allreduce のレイテンシと帯域幅

2 基の GPU 上で、α は 1 要素の float の allreduce によって測定し、β_{link} は 2^{26} 要素の float（268 MB）の allreduce によって測定する。

```
def allreduce_time(n, mesh, calls):
    """Seconds per allreduce of n floats per device."""
    x = jnp.zeros((mesh.size, n), device=NamedSharding(mesh, P('data')))
    return d2l.Benchmark(lambda: allreduce(x, mesh), repeats=calls).time

def program_time(m, mesh, calls=300):
    """Seconds per call of one compiled program that makes m allreduces of
    one float, each on the result of the one before."""
    def chain(x):
        for _ in range(m):              # Each allreduce waits for the last
            x = jax.lax.pmean(x, 'data')
        return x
    f = jax.jit(jax.shard_map(chain, mesh=mesh, in_specs=P('data'),
                              out_specs=P('data')))
    x = jnp.zeros((mesh.size, 1), device=NamedSharding(mesh, P('data')))
    return d2l.Benchmark(lambda: f(x), warmup=10, repeats=calls).time

# The training step's psum runs inside a program that is dispatched anyway,
# so alpha is what each further allreduce adds to one program
alpha = (program_time(10, meshes[1]) - program_time(1, meshes[1])) / 9
n = 2**26                               # 64M floats ...
S = 4 * n                               # ... 268 MB (256 MiB) per device
# Both calls dispatch a program of their own; their difference is the bytes
beta_link = (allreduce_time(n, meshes[1], calls=3)
             - allreduce_time(1, meshes[1], calls=100)) / S  # s per byte
```

```
print(f'alpha = {1e6 * alpha:.0f} us, beta_link = {1e9 * beta_link:.2f} '
      f'ns/byte (bus bandwidth {1e-9 / beta_link:.1f} GB/s)')
```

```
alpha = 25 us, beta_link = 0.41 ns/byte (bus bandwidth 2.4 GB/s)
```

Equation 13.8 が勾配融合（gradient fusion）の効果を予測できるか確かめるため、LeNet の 10 個の勾配テンソル（合計 $S = 247$ KB）を個別に allreduce する場合（$c = 10$）と、単一の平坦なベクトルに連結して allreduce する場合（$c = 1$）を比較する。両者の差は 9α になるはずである。

```
mesh = meshes[1]
sizes = [p.size for p in base]
flat = jnp.zeros((mesh.size, sum(sizes)), device=NamedSharding(mesh, P('data')))
n_bytes, c = flat.nbytes // mesh.size, len(base)  # Bytes per device

def separately(g):
    """One psum per gradient tensor, each after the one before; without the
    barrier, XLA would merge the ten into one allreduce."""
    out = []
    for g_i in jnp.split(g, np.cumsum(sizes)[:-1], axis=1):
        if out:
            g_i, _ = jax.lax.optimization_barrier((g_i, out[-1]))
        out.append(jax.lax.psum(g_i, 'data'))
    return jnp.concatenate(out, axis=1)

t = {}
for name, f in (('separately', separately),
                ('at once', lambda g: jax.lax.psum(g, 'data'))):
    f = jax.jit(jax.shard_map(f, mesh=mesh, in_specs=P('data'),
                              out_specs=P('data')))
    t[name] = d2l.Benchmark(lambda: f(flat), warmup=10, repeats=300).time
    print(f'{name:10s}: {1e6 * t[name]:4.0f} us measured')
# Both programs are dispatched once and move the same bytes
print(f'difference: {1e6 * (t["separately"] - t["at once"]):4.0f} us '
      f'measured, {1e6 * (c - 1) * alpha:4.0f} us predicted by {c - 1} alpha')
```

```
separately:  524 us measured
at once   :  287 us measured
difference:  236 us measured,  221 us predicted by 9 alpha
```

247 KB の勾配では、余分な 9 通のメッセージの封筒（9α）のほうが、その中身である全バイトの転送よりも多くのコストがかかる。実用的なデータ並列ライブラリが、通信前に小さな勾配を大きなバケットにまとめるのはこのためである（13.6 節）。

13.5.4.3 1 基および 2 基の GPU 上での LeNet

2 基の GPU で LeNet を実行する前に、256 個の例からなるミニバッチに対する 1 基の GPU 上でのローカルな計算をプロファイリングしてみよう。

```
mesh, params = meshes[0], broadcast(base, meshes[0])
i = jax.device_put(idx, NamedSharding(mesh, P('data')))
step = lambda: train_step(params, train_data[0], i, 0.2, mesh,
```

```
                            communicate=False)
    t = d2l.Benchmark(step, repeats=500).time
    d2l.show_profile(d2l.kernel_times(step, steps=100), 1e3 * t, rows=4)
```

```
wall clock 0.40 ms per step, device busy 0.17 ms (42%), matrix products 30%
  0.02 ms  12%  _5x_cudnn_ampere_scudnn_128x32_relu_small_nn_v1
  0.02 ms  11%  dgrad2d_alg1_1<float, 0, 6, 7, 5, 4, 5, false, true>
  0.02 ms  10%  implicit_convolve_sgemm<float, float, 128, 5, 5, 3,
  0.01 ms   8%  fft2d_r2c_32x32<float, false, 0u, false>
```

ステップの大半で GPU はアイドル状態にある。バッチサイズ 256 の LeNet はオーバーヘッド律速（overhead-bound）であり、ステップ時間はホストディスパッチ時間 h_{step} に等しい。そのため、1 基の GPU でバッチを 256 から 128 に半減させてもステップ時間はほとんど変わらない一方、1 つのホストスレッドから 2 基の GPU を駆動すると約 $2h_{\text{step}}$ かかる（Figure 13.19）。

```
    t_local = {}
    for k, b in ((1, 256), (1, 128), (2, 128)):
        mesh, data_k = meshes[k - 1], train_data[k - 1]
        params = broadcast(base, mesh)
        i = jax.device_put(idx[:k * b], NamedSharding(mesh, P('data')))
        t_local[k, b] = d2l.Benchmark(lambda: train_step(
            params, data_k, i, 0.2, mesh, communicate=False), repeats=500).time
        print(f'{k} GPU(s), {b} examples each: {1e3 * t_local[k, b]:.2f} ms')
```

```
1 GPU(s), 256 examples each: 0.37 ms
1 GPU(s), 128 examples each: 0.40 ms
2 GPU(s), 128 examples each: 0.76 ms
```

単一ホストのディスパッチ下限 $k\,h_{\text{step}}$ をデバイス計算時間 $t_{\text{dev}}(B/k)$ とあわせて組み込むと、Equation 13.8 は次のように精密化される。

$$t_{\text{step}}(k) \approx \max\bigl(k\,h_{\text{step}},\; t_{\text{dev}}(B/k)\bigr) + c\,\alpha + \frac{2(k-1)}{k}\,S\,\beta_{\text{link}}. \tag{13.9}$$

両方の予測をエンドツーエンドで検証するため、訓練ループ `train` を記述する。この関数は、`orders` 内の事前生成されたデータ例の順列ごとに 1 エポックを実行し、（最初のエポックを除いた）平均ステップ時間と検証正解率を返す。

```
    def train(params, train_data, test_data, mesh, orders, batch_size=256,
              lr=0.2):
        """Data-parallel SGD on the devices of `mesh`, one epoch per example
        order; returns seconds per step, test accuracy."""
        times = []
        for order in orders:
            starts = range(0, len(order), batch_size)
            batches = jax.device_put(  # The epoch's minibatches, in one call
                [order[s:s + batch_size] for s in starts],
                NamedSharding(mesh, P('data')))
            t0 = time.perf_counter()
            for idx in batches:
                params = train_step(params, train_data, idx, lr, mesh)
```

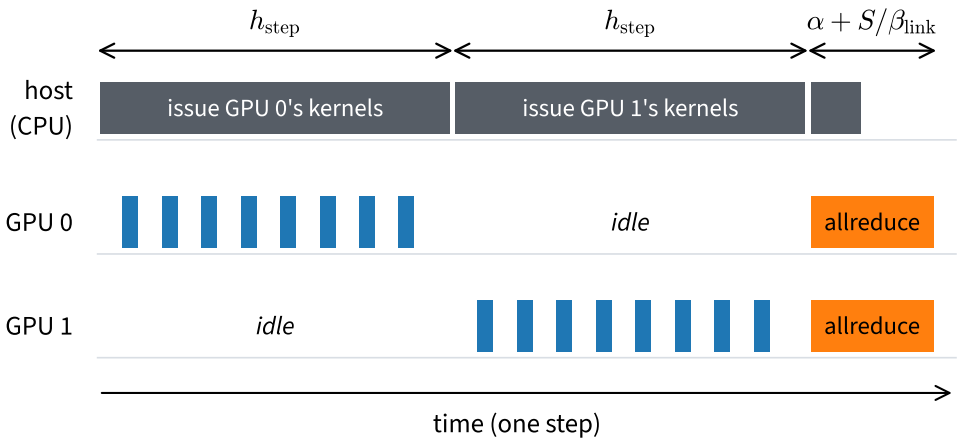


Figure 13.19: 2 基の GPU 上でのホスト律速ステップ。1 つのホストがまず GPU 0 の処理を発行し、次に GPU 1 の処理を発行する様子として描いている。ホストが一方の GPU を処理している間、もう一方の GPU はアイドル状態になるため、カーネルが短いステップではホスト時間の約 $2h_{\text{step}}$ に通信時間を加えた時間がかかる。

```
        jax.block_until_ready(params)
        times.append((time.perf_counter() - t0) / len(batches))
    X, y = test_data
    acc = float((jax.jit(lenet)(params, X).argmax(1) == y).mean())
    return sum(times[1:]) / (len(times) - 1), acc
```

それでは、グローバルバッチサイズ 256 で `train` を $k \in \{1, 2\}$ 基の GPU 上で 5 エポック実行し、測定された 2 GPU のステップ時間 `t_step[2]` を、素朴な予測 Equation 13.8 と単一ランチャモデル Equation 13.9 の両方と比較しよう。

```
rng = np.random.default_rng()
orders, t_step = [rng.permutation(len(data.train[1])) for _ in range(5)], {}
for k, mesh, data_k in zip((1, 2), meshes, train_data):
    t_step[k], acc = train(broadcast(base, mesh), data_k, test_data, mesh,
                           orders)
    print(f'{k} GPU(s): {1e3 * t_step[k]:.2f} ms per step, '
          f'test accuracy {acc:.2f}')
exchange = alpha + n_bytes * beta_link    # One allreduce of the gradient
print(f'2 GPUs predicted: {1e3 * (t_local[1, 128] + exchange):.2f} ms per '
      f'step by eq_dp_cost, {1e3 * (2 * t_step[1] + exchange):.2f} ms with '
      f'one launcher')
```

```
1 GPU(s): 0.37 ms per step, test accuracy 0.86
2 GPU(s): 0.89 ms per step, test accuracy 0.86
2 GPUs predicted: 0.52 ms per step by eq_dp_cost, 0.88 ms with one launcher
```

どちらの実行も同じテスト正解率に到達し、Equation 13.9 は 2 GPU での速度低下を正確に予測している。バッチ 256 では LeNet の勾配交換はごくわずか（約 0.1 ms）であり、ボトルネックは単一のホストランチャから 2 基の GPU に処理を発行する部分にある。

13.5.4.4 強スケーリングと弱スケーリング、およびより大きなミニバッチ

マルチ GPU のベンチマークは 2 つの流儀のいずれかに従う。**強スケーリング**（strong scaling）は、グローバルバッチ B を固定したまま k を増やし（同じステップをより速く終えることを目指す）、**弱スケーリング**（weak scaling）は、GPU あたりのバッチ b を固定してグローバルバッチを kb へと拡大する（例/秒で表したスループットを $k\times$ にすることを目指すが、グローバルバッチが大きくなると最適化のダイナミクスが変わる、9.8 節）。強スケーリングは**アムダールの法則** (Amdahl 1967) の領域である。ステップのうち並列化できない部分（ここではホストランチャと勾配交換）が、GPU をどれだけ増やしても高速化率の上限を決めてしまう。弱スケーリングは**グスタフソンの法則** (Gustafson 1988) の領域である。プロセッサ数に応じて計算量が増えると、逐次部分の相対的な割合は縮小し、スループットを伸ばし続けることができる。

Equation 13.9 より、2 基目の GPU が弱スケーリングのスループットを向上させる（$2\,t(1)/t(2) > 1$）のは、次の条件が成り立つときである。

$$\max(2h_{\text{step}},\ t_{\text{dev}}(b)) + \alpha + S\,\beta_{\text{link}} \ <\ 2\max(h_{\text{step}},\ t_{\text{dev}}(b)).$$

$t_{\text{dev}}(b) \le h_{\text{step}}$ のときは、2 基目の GPU を追加しても決して速くならない。$t_{\text{dev}}(b)$ が $2h_{\text{step}}$ と勾配交換時間を大きく上回るようになると、高速化率は 2 に近づく。これを GPU あたり $b \in \{256, 4096, 32768\}$ 個の例にわたって検証しよう。

```
rng = np.random.default_rng()
for b in (256, 4096, 32768):                # Examples per GPU
    batch = rng.integers(len(data.train[1]), size=2 * b)  # With replacement
    steps = []
    for k, mesh, data_k in zip((1, 2), meshes, train_data):
        params = broadcast(base, mesh)
        i = jax.device_put(batch[:k * b], NamedSharding(mesh, P('data')))
        steps.append(partial(train_step, params, data_k, i, 0.2, mesh))
    times = [[], []]                        # Three timed windows per k,
    for w in range(3):                      # taken in turn for k = 1 and 2
        for j, step in enumerate(steps):
            times[j].append(d2l.Benchmark(
                step, warmup=10 if w == 0 else 0,
                repeats=max(20, 2**18 // b)).time)
    t = np.median(times, axis=1)
    t_dev = sum(d2l.kernel_times(steps[0]).values()) / 1e3
    if b == 256:              # The host sets this step's time (profile above)
        h_step = t[0]
    predicted = 2 * t[0] / (max(2 * h_step, t_dev) + exchange)
    speedups = 2 * np.array(times[0]) / np.array(times[1])
    print(f'b = {b:5d}: 1 GPU {1e3 * t[0]:6.2f} ms (device busy '
          f'{1e3 * t_dev:5.2f} ms), 2 GPUs {1e3 * t[1]:6.2f} ms\n'
          f'           speedup {predicted:.2f} predicted, '
          f'{2 * t[0] / t[1]:.2f} measured '
          f'(windows {speedups.min():.2f} to {speedups.max():.2f})')
```

```
b =   256: 1 GPU   0.41 ms (device busy  0.16 ms), 2 GPUs   0.97 ms
           speedup 0.87 predicted, 0.84 measured (windows 0.83 to 0.87)
b =  4096: 1 GPU   1.75 ms (device busy  1.74 ms), 2 GPUs   1.99 ms
           speedup 1.89 predicted, 1.76 measured (windows 1.72 to 1.80)
```

```
b = 32768: 1 GPU  14.29 ms (device busy 14.26 ms), 2 GPUs  14.50 ms
           speedup 1.99 predicted, 1.97 measured (windows 1.96 to 1.97)
```

$b = 256$ では 2 基目の GPU による高速化は得られないが、$b = 32{,}768$ ではデバイスの計算時間が $2h_{\text{step}}$ と 247 KB の勾配交換を大きく上回り、予測どおりスループットがほぼ 2 倍になる。

13.5.5 まとめ

データ並列はモデルを k 個のデバイスに複製し、デバイスごとにミニバッチの $1/k$ に対する勾配を計算し、更新の前に allreduce で勾配を合計する。リング allreduce はこの総和を reduce-scatter とそれに続く all-gather に分解し、デバイスあたり $2(k-1)S/k < 2S$ バイトを $2(k-1)$ ホップにわたって送信する。本節のステップコストモデル Equation 13.9 は、デバイス計算時間、単一ホストのディスパッチオーバーヘッド $k\,h_{\text{step}}$、呼び出しごとのレイテンシ $c\alpha$、および帯域幅コスト $2(k-1)S\beta_{\text{link}}/k$ を組み合わせたものであり、小さな勾配テンソルを連結することでなぜ 9α を節約できるのか、そして GPU あたりのデバイス計算がホストディスパッチと通信の両方を上回って初めて 2 基目の GPU が訓練を高速化できるのはなぜかを説明する。

13.5.6 演習問題

1. **データ並列とパイプライン並列の比較**。バイアスなしの L 個の線形層（幅 d、fp32）のスタックを、2 基の GPU 上でバッチサイズ B で訓練する場合を考える。データ並列はすべての重み勾配を allreduce し、2 ステージのパイプライン並列（Figure 13.15）は各 GPU に $L/2$ 層を配置して、分割境界をまたいで順方向に $B \times d$ 個の活性化値を、逆方向に勾配を送信する。
   1. 各方式で 1 ステップあたりに転送されるバイト数を数え、$L = 32, d = 4096, B = 256$ に対して両者を評価し、データ並列のほうが転送バイト数が少なくなる (B, L, d) の条件を求めよ。
   2. バッチを m 個のパイプライン用マイクロバッチに分割したとき (Huang et al. 2019)、2 ステージのパイプラインにおける各 GPU のバブル（アイドル）割合を導出し、$m = 1$ および $m = 8$ に対して評価せよ。
2. **勾配の正規化**。k 個のシャード（合計 B 個の例からなるバッチを分割したもの）にわたる 2 つのデータ並列勾配レシピを比較せよ。レシピ A は各デバイスで例ごとの損失を合計し、デバイス間で勾配を合計してから B で割る。レシピ B は各デバイスで例ごとの損失を平均し、k 個のデバイス勾配を平均する。
   1. すべてのシャードが B/k 個の例を保持するとき、両者がともに厳密なフルバッチの平均勾配を与えることを示せ。
   2. トークン全体で損失を平均する言語モデルにおいて、2 つのシャードがそれぞれ 3,000 トークンと 1,000 トークンを保持しているとする。レシピ B が各シャード内の 1 トークンに割り当てる相対的な重みを求め、トークン数が不均等な場合でも厳密になるようレシピ B を修正せよ。
3. **強スケーリングの高速化率**。シャードの計算時間を $t_{\text{compute}}(s) = t_0 + s\,t_1$ とモデル化し、α を無視すると、Equation 13.8 より固定されたグローバルバッチ B において次式が成り立つ。

$$t_{\text{step}}(k) = t_0 + \frac{B\,t_1}{k} + \frac{2(k-1)}{k}\,S\,\beta_{\text{link}}. \tag{13.10}$$

   1. 強スケーリングの高速化率 $R(k) = t_{\text{step}}(1)/t_{\text{step}}(k) > 1$ が、ある 1 つの $k > 1$ で成り立つのは、すべての $k > 1$ で成り立つとき、かつそのときに限ることを示し、$\lim_{k\to\infty} R(k)$ を求めよ。
   2. $N = 11.2$M 個の fp32 パラメータ、$t_0 = 2$ ms、$t_1 = 0.2$ ms/例、$B = 256$ とするとき、$R(2)$、$R(4)$、および $\lim_{k\to\infty} R(k)$ をバス帯域幅 20 GB/s と 5 GB/s のそれぞ

れで評価し、少しでも高速化が得られる最小のバス帯域幅を求めよ。
3. 5 GB/s における $R(4)$ および $\lim_{k\to\infty} R(k)$ を、勾配を bf16 で通信する場合（S が半分になる）について再計算せよ。

4. [code] **手作業によるリング allreduce**。●
1. k 個のデバイス（それぞれ S バイトを保持）について、スター allreduce とリング allreduce で移動する総バイト数と、最も負荷の高いデバイスが送受信するバイト数を比較せよ。また、k 本すべての双方向リンクが同時に動作する場合と、すべての転送が単一のバスを逐次的に共有する場合のそれぞれについて、両者の所要時間の比を予測せよ。
2. `ring_allreduce` を `jax.shard_map` の内部で（`jax.lax.ppermute` と `jax.lax.axis_index` を用いて）実装し、`jax.lax.psum` と比較して $k \in \{2, 4\}$ で検証したうえで、デバイスあたり 64M 個の float に対してベンチマークせよ。

5. **allreduce におけるレイテンシ**。●n バイトの 1 ホップに $\alpha_{\text{hop}} + n\,\beta_{\text{link}}$ かかるとき、$2(k-1)$ ステップのリング allreduce の所要時間は

$$t_{\text{ring}} = 2(k-1)\left(\alpha_{\text{hop}} + \frac{S}{k}\,\beta_{\text{link}}\right), \tag{13.11}$$

であるのに対し、二分木 allreduce（$2\log_2 k$ ステップ、各ステップ S バイトで上向きに総和をとり下向きにブロードキャストする）の所要時間は

$$t_{\text{tree}} = 2\log_2 k\,\left(\alpha_{\text{hop}} + S\,\beta_{\text{link}}\right). \tag{13.12}$$

である。
1. 各アルゴリズムのレイテンシ項と帯域幅項が k に伴いどうスケールするかを比較せよ。$k = 8$、$\alpha_{\text{hop}} = 10$ µs、バス帯域幅 20 GB/s のとき、木のほうが高速になる境界のメッセージサイズ S を求めよ。
2. $k = 2$ において、半性能メッセージサイズ $S_{1/2}$（固定レイテンシが転送時間と等しくなるサイズ）を α と β_{link} を用いて表し、13.5.4 節での測定値から評価して、13.6 節の勾配バケットのサイズと比較せよ。

13.6 データ並列の実践

13.5 節で手書きしたデータ並列（data parallelism）ループでは、2 基の GPU で LeNet がかえって遅くなった。その理由は 2 つある。単一の Python スレッドが両方の GPU へカーネルを逐次的にディスパッチしていたこと、そして出力に近い層の勾配はずっと早い段階で揃っていたにもかかわらず、逆伝播全体が完了するまでオールリデュースが待機していたことである。

実用データ並列ツールは、これら両方のボトルネックを解消する。GPU ごとに 1 つのホストプロセスを実行するか、全 GPU にまたがる単一のシャーディングされたプログラムをコンパイルし、さらに勾配を**バケット**にまとめて逆伝播中に各バケットが埋まり次第そのオールリデュースを開始するのである。逆伝播の計算の背後に勾配交換をどれだけ隠蔽できるかを検証するため、ここでは LeNet から 64×64 の Fashion-MNIST 画像に対する ResNet-18 へと対象を移す（画像 1 枚あたり 22 億回の積和演算、45 MB の勾配 1 バイトあたり約 300 訓練 FLOPs を要し、画像 256 枚の 1 ステップあたり 1 基の RTX 4090 を約 100 ms 稼働させ続ける）。並列ステップが何を計算しているかを 1 ステップチェックで検証したうえで、1、2、4 基の GPU にわたる弱スケーリング（weak scaling）と強スケーリング（strong scaling）を計測し、コストモデルと比較する。

```
# JAX and NCCL read these settings once, when they start, so this cell runs
# before the imports. XLA overlaps a collective with computation only when
```

```
# its latency-hiding scheduler is on. The 'vmm' allocator reserves nothing
# up front and maps each array into one contiguous range, where JAX's
# default allocator reserves most of every GPU at start-up and carves every
# array out of that block: one step below needs about 10 GB in one piece
# after many smaller arrays have come and gone, and a later cell starts a
# second JAX process on the same GPUs.
# On GPUs without peer-to-peer access, such as ours, set
# NCCL_LOCAL_REGISTER=0 in the environment before starting: NCCL's buffer
# registration fails there and prints a warning at every collective.
import os
os.environ.setdefault('XLA_PYTHON_CLIENT_ALLOCATOR', 'vmm')
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_enable_latency_hiding_scheduler=true')
```

```
import json
import re
import subprocess
import sys
from jax.sharding import NamedSharding, PartitionSpec as P
```

13.6.1 実用データ並列

13.6.1.1 実行モデル

現代のあらゆるフレームワークは、次の 2 種類の**単一プログラム複数データ**（single program, multiple data、SPMD）設計のいずれかを用いて k 基の GPU を駆動する。

1. **マルチプロセス（k 個の複製＋集団通信ライブラリ）：**ランチャが k 個の独立した Python プロセス（GPU ごとに 1 つ、それぞれが $\{0, \dots, k-1\}$ の固有の**ランク**と専用の CPU スレッドを持つ）を起動し、NCCL を介して勾配を同期する。各プロセスの内部では任意の Python コードを実行できる。
2. **コンパイラ分割型の単一プログラム：**単一デバイス向けのステップを記述し、デバイスメッシュ上での入力のシャーディングを注釈として指定すると、コンパイラが計算を分割し、集団通信演算（collective operations）を自動的に挿入・スケジューリングする (Xu et al. 2021)。

JAX はコンパイラ分割型設計を採用しており、`nnx.jit` が単一デバイス向けの訓練ステップをコンパイルする。複製されたパラメータ（`P()`）とバッチ軸に沿ってシャーディングされた入力（`P('data')`）を並べて配置することで、Shardy/GSPMD パーティショナが勾配のオールリデュースを自動的に挿入する。

13.6.1.2 逆伝播と通信のオーバーラップ

実用フレームワークは、層の逆順（誤差逆伝播法が勾配を生成する順序）で勾配を数十メガバイト単位の**バケット**に詰め込み、バケット内の最後の勾配が揃い次第、各バケットの非同期オールリデュースを起動する (S. Li et al. 2020)（Figure 13.20）。

バケットサイズは、呼び出しごとのレイテンシ α とオーバーラップのトレードオフを決める。極小のバケットでは α を何度も支払うことになり、単一の巨大なバケットでは逆伝播が完了するまで通信を開始できない。理想的なオーバーラップの**バケットモデル**のもとでは（すなわち、次のバケットが埋まる前に各バケットの通信を終えられるほど GPU 間リンクが高速であり、かつ通

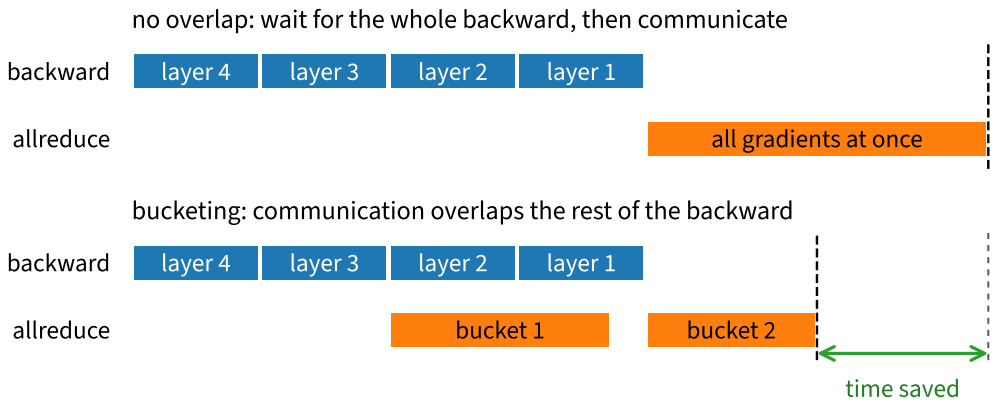


Figure 13.20: 上：オールリデュースが逆伝播全体の完了を待つ場合。下：勾配をバケットにまとめ、逆伝播が各バケットの勾配を生成し終え次第、そのオールリデュースを開始する場合。逆伝播の終了後に実行されるのは最後のバケットのみであり、それより前のバケットが隠蔽されるのは、逆伝播がまだ実行されている間に各オールリデュースが完了する場合に限られる。

信が同時実行中の逆伝播カーネルを減速させないと仮定すると)、最後のバケットを除くすべてのバケットが逆伝播の背後に隠蔽される。

XLA は 30 MiB のしきい値（`--xla_gpu_all_reduce_combine_threshold_bytes`）まで勾配のオールリデュースを結合し、ResNet-18 の 45 MB の勾配を、逆伝播計算と並行してスケジュールされる 2 組の非同期 `all-reduce-start` / `all-reduce-done` ペアにまとめる。

13.6.1.3 訓練ステップ

7.4 節の小画像向け ResNet-18（第 32 章（英語版） でも利用するため `#@save` しておく）を定義し、その勾配バイト数 S（fp32 で 45 MB）を数える。

```
def resnet18(num_classes, in_channels=1, *, rngs):
    """ResNet-18 for small images: a 3x3 stem without max-pooling, He
    initialization."""
    he = nnx.initializers.variance_scaling(2.0, 'fan_in', 'normal')
    def stage(num_channels, in_channels, first=False):
        return [d2l.Residual(num_channels, use_1x1conv=not first,
                             strides=1 if first else 2,
                             in_channels=in_channels, rngs=rngs),
                d2l.Residual(num_channels, in_channels=num_channels,
                             rngs=rngs)]
    return nnx.Sequential(
        # No stem bias: the batch normalization after the conv removes it.
        # Running averages decay 0.9, two-pass variance
        nnx.Conv(in_channels, 64, (3, 3), padding='same', use_bias=False,
                 kernel_init=he, rngs=rngs),
        nnx.BatchNorm(64, momentum=0.9, use_fast_variance=False, rngs=rngs),
        nnx.relu,
        *stage(64, 64, first=True), *stage(128, 64), *stage(256, 128),
        *stage(512, 256),
```

```
        lambda X: X.mean(axis=(1, 2)),  # Global average pooling
        nnx.Linear(512, num_classes, kernel_init=he, rngs=rngs))
```

次に、Fashion-MNIST を読み込み、ResNet-18 のパラメータ数 num_params と fp32 での勾配データ量 S を計算する。

```
data = d2l.FashionMNIST()
X_train = jax.image.resize(data.train[0][..., None] / np.float32(255),
                           (len(data.train[0]), 64, 64, 1), 'bilinear')
y_train = jnp.asarray(data.train[1], jnp.int32)
num_params = d2l.num_params(resnet18(10, rngs=nnx.Rngs(d2l.get_key())))
S = 4 * num_params                    # Gradient bytes in fp32
print(f'{num_params / 1e6:.1f}M parameters, S = {S / 1e6:.0f} MB')
```

```
11.2M parameters, S = 45 MB
```

続いて、64×64 の Fashion-MNIST 画像に対するデータ並列の訓練ステップを定義する。

setup はモデルとデータセットを 'data' メッシュ上に複製し、train_step はシャーディングされたミニバッチインデックス idx に対して JIT コンパイルされた SGD 更新を実行する。

```
def setup(k, key):
    """ResNet-18 with SGD and the training set, replicated on k GPUs."""
    mesh = jax.make_mesh((k,), ('data',), devices=jax.devices()[:k],
                         axis_types=(jax.sharding.AxisType.Auto,))
    replicated = NamedSharding(mesh, P())
    model = resnet18(10, rngs=nnx.Rngs(key))
    opt = nnx.Optimizer(model, optax.sgd(0.1), wrt=nnx.Param)
    for m in (model, opt):
        nnx.update(m, jax.device_put(nnx.state(m), replicated))
    return mesh, model, opt, jax.device_put((X_train, y_train), replicated)

def minibatches(mesh, batch_size):
    """Indices of random minibatches, split across the GPUs of the mesh."""
    rng, n = np.random.default_rng(), len(y_train)
    while True:
        perm = rng.permutation(n)
        for i in range(0, n - batch_size + 1, batch_size):
            yield jax.device_put(perm[i:i + batch_size],
                                 NamedSharding(mesh, P('data')))

@nnx.jit
def train_step(model, opt, X, y, idx):
    """One SGD step on the examples idx, written for a single device."""
    def loss_fn(model):
        return optax.softmax_cross_entropy_with_integer_labels(
            model(X[idx]), y[idx]).mean()
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    opt.update(model, grads)
    return loss
```

13.6.1.4 1 ステップチェック

LeNet とは異なり、ResNet-18 は各チャネルをミニバッチの平均と分散で正規化するバッチ正規化（7.3 節）を含んでいる。256 個の例からなるミニバッチを 2 基の GPU に分割したとき、各 GPU は自身の 128 例のシャード統計量で正規化するのだろうか（これは 1 基の GPU 上で 128 例のマイクロバッチ 2 つにわたって勾配累積した場合と一致する）。それとも、GPU 間で 256 例の大域的な統計量を同期するのだろうか。2 基の GPU による更新を、単一 GPU におけるこれら両方の参照結果と比較して確かめてみよう。

```
key = d2l.get_key()                  # One initialization for every step
idx = np.random.default_rng().permutation(len(y_train))[:256]

def update(k, idx):
    """Parameter change of one step on k GPUs on the examples idx."""
    mesh, model, opt, (X, y) = setup(k, key)
    before = jax.tree.map(np.asarray, nnx.state(model, nnx.Param))
    train_step(model, opt, X, y,
               jax.device_put(idx, NamedSharding(mesh, P('data'))))
    return jax.tree.leaves(jax.tree.map(lambda p, b: np.asarray(p) - b,
                                        nnx.state(model, nnx.Param), before))

with jax.default_matmul_precision('highest'):  # Full fp32 convolutions
    two = update(min(2, jax.device_count()), idx)
    full = update(1, idx)
    # For SGD, the mean of the steps on the two halves is the step on
    # their accumulated gradients
    halves = [(a + b) / 2 for a, b in zip(*(update(1, half)
                                            for half in np.split(idx, 2)))]
gap = lambda u: max(np.abs(a - b).max() for a, b in zip(two, u))
print(f'largest update {max(np.abs(u).max() for u in two):.1e}\n'
      f'largest difference from one GPU on all 256: {gap(full):.1e}\n'
      f'largest difference from one GPU on two accumulated halves: '
      f'{gap(halves):.1e}')
```

```
largest update 2.4e-02
largest difference from one GPU on all 256: 2.5e-05
largest difference from one GPU on two accumulated halves: 1.7e-02
```

バッチ軸が 'data' メッシュ軸に沿って Auto モードでシャーディングされるため、XLA のパーティショナはバッチ正規化の平均と分散をミニバッチ全体にわたって大域的に計算する。これにより、単一 GPU での 256 例のステップと一致し、2 つの大きな勾配バッファに加えてチャネルごとのバッチ正規化統計量に対する小さなオールリデュースが挿入される。

```
def allreduce_megabytes(compiled):
    """Megabytes of fp32 results of each allreduce in a compiled program,
    in the order in which the program runs them."""
    sizes = []
    for line in compiled.as_text().splitlines():
        # A synchronous all-reduce, or the -done half of an asynchronous
        # pair, whose result type lists the summed arrays only
        if re.search(r' all-reduce(-done)?\(', line):
            result = line.split(' = ')[1].split(' all-reduce')[0]
```

```
            sizes.append(sum(4 * math.prod(map(int, dims.split(','')))
                             for dims in re.findall(r'f32\[([\d,]+)\]',
                                                    result)) / 1e6)
    return sizes
```

マルチ GPU メッシュ上で train_step をコンパイルし、allreduce_megabytes を適用すると、2 つの大きな勾配オールリデュース（big、最後のバッファは last_bucket として記録）と並んで、チャネルごとのバッチ正規化に対する小さなオールリデュースが確認できる。

```
mesh, model, opt, (X, y) = setup(jax.device_count(), d2l.get_key())
idx = next(minibatches(mesh, 256 * mesh.size))
compiled = train_step.lower(model, opt, X, y, idx).compile()
sizes = allreduce_megabytes(compiled)  # One entry per allreduce
big = [m for m in sizes if m > 1]      # Gradient buffers, above 1 MB
last_bucket = 1e6 * big[-1]            # Bytes of the buffer sent last
print(f'{len(sizes)} allreduces: {len(big)} of '
      f'{" and ".join(f"{m:.1f}" for m in big)} MB, the other '
      f'{len(sizes) - len(big)} of {sum(sizes) - sum(big):.2f} MB together')
```

```
76 allreduces: 2 of 25.8 and 18.9 MB, the other 74 of 0.10 MB together
```

13.6.2 4 基の GPU でのスケーリング

ここでは、$k \in \{1, 2, 4\}$ 基の GPU にわたって弱スケーリング（GPU あたり 256 例）と強スケーリング（全体で 512 例）を計測する。

```
def stepper(k, b):
    """One data-parallel step on k GPUs with b examples per GPU, as a
    function without arguments."""
    mesh, model, opt, (X, y) = setup(k, d2l.get_key())
    batches = minibatches(mesh, k * b)
    return lambda: train_step(model, opt, X, y, next(batches))

def step_time(k, b, num_steps=50):
    """Seconds per data-parallel step on k GPUs with b examples per GPU."""
    return d2l.Benchmark(stepper(k, b), warmup=10, repeats=num_steps).time
```

まず、NCCL を介した単体の 256 MB オールリデュースで β_{link} を測定し、弱スケーリングのもとでの露出通信時間（exposed communication time）$t_{\text{step}}(k) - t_{\text{step}}(1)$ に対する次の 2 つの予測を比較する。

1. **オーバーラップなし：**$S = 45$ MB のリングオールリデュース全体が露出する（$\frac{2(k-1)}{k} S\beta_{\text{link}}$）。
2. **完全なバケットオーバーラップ：**逆伝播の終了後に最後のバケットまたはバッファ（$last_bucket$）のみが露出する。

```
ALLREDUCE = r'''
import json, jax
from jax import numpy as jnp
from jax.sharding import NamedSharding, PartitionSpec as P
```

```
from d2l import jax as d2l

mesh = jax.make_mesh((2,), ('data',), devices=jax.devices()[:2])
x = jnp.zeros((2, 2**26), device=NamedSharding(mesh, P('data')))  # 256 MB
psum = jax.jit(jax.shard_map(lambda x: jax.lax.psum(x, 'data'), mesh=mesh,
                             in_specs=P('data'), out_specs=P('data')))
t = d2l.Benchmark(lambda: psum(x), repeats=10).time
print(json.dumps({'beta_link': t / (x.nbytes / mesh.size)}))  # S per GPU
'''
def allreduce_beta(env=None):
    """beta_link of a bare 256 MB allreduce on two GPUs: the seconds
    per byte that each GPU sends."""
    run = subprocess.run([sys.executable, '-c', ALLREDUCE], check=True,
                         capture_output=True, text=True,
                         env={**os.environ, **(env or {})})
    return json.loads(run.stdout.splitlines()[-1])['beta_link']
```

allreduce_beta() を呼び出すと実測リンクコスト beta_link が得られる。これを用いて、$k \in \{2, 4\}$ 基の GPU に対する exposed[k] を、オーバーラップなしモデル（$2(k-1)S\beta_{\text{link}}/k$）と完全オーバーラップモデル（last_bucket）の両方について計算する。

```
if d2l.num_gpus() >= 2:
    beta_link = allreduce_beta()
    print(f'bus bandwidth = {1e-9 / beta_link:.2f} GB/s '
          f'(beta_link = {1e9 * beta_link:.2f} ns/byte)')
    exposed = {}                    # Predicted exposed seconds per step
    for k in (2, 4):
        ring = 2 * (k - 1) / k      # Bytes that each GPU sends, per byte of S
        exposed[k] = (ring * S * beta_link, ring * last_bucket * beta_link)
        print(f'k={k}: exposed {1e3 * exposed[k][0]:.0f} ms without overlap '
              f'({ring:.1f} S beta_link), {1e3 * exposed[k][1]:.0f} ms with '
              f'full overlap (last bucket)')
```

```
bus bandwidth = 2.43 GB/s (beta_link = 0.41 ns/byte)
k=2: exposed 18 ms without overlap (1.0 S beta_link), 8 ms with full overlap (last bucket)
k=4: exposed 28 ms without overlap (1.5 S beta_link), 12 ms with full overlap (last bucket)
```

それでは、GPU あたり 256 例（step_time(k, 256)）で $k \in \{1, 2, 4\}$ 基の GPU にわたる弱スケーリングをベンチマークし、実測されたスループット、スケーリング効率、および露出オーバーヘッド weak[k] - weak[1] を両方の予測と比較しよう。

```
ks = [k for k in (1, 2, 4) if k <= d2l.num_gpus()]
weak = {k: step_time(k, 256) for k in ks}
for k in ks:
    print(f'{k} GPU(s): {k * 256 / weak[k]:5.0f} examples/s, speedup '
          f'{k * weak[1] / weak[k]:.2f}, efficiency {weak[1] / weak[k]:.0%}')
    if k > 1:  # Predicted: the one-GPU step plus the exposed communication
        none, full = (weak[1] / (weak[1] + e) for e in exposed[k])
        print(f'{"":10s}predicted {none:.0%} without overlap, {full:.0%} '
              f'with full overlap; exposed {1e3 * (weak[k] - weak[1]):.0f} ms')
```

```
1 GPU(s):  2555 examples/s, speedup 1.00, efficiency 100%
```

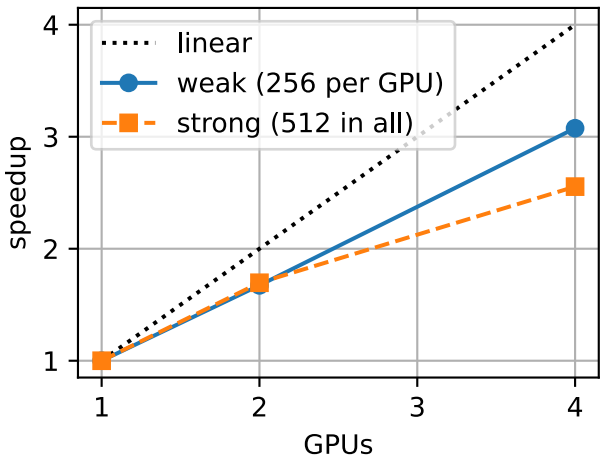
```
2 GPU(s):  4275 examples/s, speedup 1.67, efficiency 84%
          predicted 84% without overlap, 93% with full overlap; exposed 20 ms
4 GPU(s):  7859 examples/s, speedup 3.08, efficiency 77%
          predicted 78% without overlap, 90% with full overlap; exposed 30 ms
```

バス帯域幅が約 2.4 GB/s の場合、オーバーラップなしの 45 MB のオールリデュースには 2 基の GPU で約 18 ms、4 基で約 28 ms かかる。弱スケーリングの実測では、4 基の GPU は単一 GPU の 3.1–3.3× のスループット（スケーリング効率 77〜83%）を達成した。これは**オーバーラップなし**の予測とよく一致しており、完全オーバーラップのバケット予測を大きく下回っている。

強スケーリング（グローバルバッチを 512 例に固定するため、通信量が一定のまま k 基の各 GPU が $512/k$ 例を処理する設定）のもとでは、$k = 4$ で効率がさらに低下する。

```
strong = {k: step_time(k, 512 // k) for k in ks}
for k in ks:
    print(f'{k} GPU(s), {512 // k} examples each: speedup '
          f'{strong[1] / strong[k]:.2f}, efficiency '
          f'{strong[1] / strong[k] / k:.0%}')
d2l.plot(ks, [ks, [k * weak[1] / weak[k] for k in ks],
              [strong[1] / strong[k] for k in ks]], 'GPUs', 'speedup',
         legend=['linear', 'weak (256 per GPU)', 'strong (512 in all)'],
         fmts=('k:', '-o', '--s'))
```

```
1 GPU(s), 512 examples each: speedup 1.00, efficiency 100%
2 GPU(s), 256 examples each: speedup 1.70, efficiency 85%
4 GPU(s), 128 examples each: speedup 2.55, efficiency 64%
```



13.6.2.1 バケットモデルの仮定

なぜオーバーラップによって通信がほとんど隠蔽されなかったのだろうか。バケットモデルは、(1) 各バケットのオールリデュースが逆伝播の最中に開始すること、(2) NCCL が単体ベンチマークと同等の速度で動作すること、(3) NCCL が並行して動作しても逆伝播の計算カーネルが減速しないこと、の 3 つを仮定している。GPU 0 を 3 つの構成（1 基の GPU、通信なしの 2 基の GPU、通信ありの 2 基の GPU）にわたってプロファイリングすることで、これら 3 つの仮定すべてを検証できる。

```
def local_stepper(k, b):
    """The step of `stepper` without communication: inside shard_map every
    GPU sees only its shard, normalizes it with the shard's own batch
    statistics, and steps on its own shard's gradient."""
    mesh, model, opt, (X, y) = setup(k, d2l.get_key())
    batches = minibatches(mesh, k * b)
    graphdef, state = nnx.split((model, opt))
    def local(state, X, y, idx):
        model, opt = nnx.merge(graphdef, state)
        train_step(model, opt, X, y, idx)
        return nnx.state((model, opt))
    # With check_vma=False autodiff adds no sum over the GPUs, and the
    # copies, which then drift apart, are returned as if they were equal
    local = jax.jit(jax.shard_map(local, mesh=mesh, check_vma=False,
                                  in_specs=(P(), P(), P(), P('data')),
                                  out_specs=P()))
    def step():
        nonlocal state
        state = local(state, X, y, next(batches))
        return state
    return step

def profile_step(k, b, communicate=True):
    """Seconds per step on k GPUs with b examples per GPU, and device
    milliseconds per step on GPU 0 of computation and of NCCL's kernels:
    the two gradient allreduces and the small allreduces of the batch
    normalization statistics."""
    step = stepper(k, b) if communicate else local_stepper(k, b)
    t = d2l.Benchmark(step, warmup=10, repeats=50).time
    kernels = d2l.kernel_times(step)
    comm = sum(ms for name, ms in kernels.items() if 'nccl' in name.lower())
    return t, sum(kernels.values()) - comm, comm
```

GPU あたり $b = 256$ 例で、単一 GPU、勾配同期なしの 2 基の GPU、オールリデュースありの 2 基の GPU という 3 つの構成にわたって `profile_step` を実行し、実経過ステップ時間をデバイスの計算時間と NCCL カーネル時間に分離する。

```
if d2l.num_gpus() >= 2:
    for k, communicate in ((1, True), (2, False), (2, True)):
        t, compute, comm = profile_step(k, 256, communicate)
        name = f'{k} GPU' + ('s' if k > 1 else '') + (
            '' if communicate else ' without communication')
        print(f'{name + ":":30s}{1e3 * t:6.1f} ms per step; GPU 0 computes '
              f'{compute:6.1f} ms, communicates {comm:5.1f} ms')
```

```
1 GPU:                          100.3 ms per step; GPU 0 computes  100.1 ms, communicates   0.0 ms
2 GPUs without communication:   100.6 ms per step; GPU 0 computes  100.0 ms, communicates   0.0 ms
2 GPUs:                         119.1 ms per step; GPU 0 computes  107.8 ms, communicates  21.0 ms
```

通信なしの 2 基の GPU は単一 GPU と全く同じ速度でステップを実行し、NCCL のカーネルは 20〜22 ms 動作する（18 ms の単体ベンチマークと整合する）。破綻しているのは 3 つ目の仮定である。すなわち、**NCCL が逆伝播と並行して動作すると、計算カーネル自体が 8〜13 ms 遅くなる**のである。ピアツーピアアクセスを持たない GPU では、NCCL は独自の GPU SM カーネルを

使ってホストメモリ経由でデータをコピーするため、本来であれば逆伝播が利用するはずのストリーミングマルチプロセッサとメモリ帯域幅を占有してしまう。

非公式の NCCL フラグ（NCCL_SHM_USE_CUDA_MEMCPY=1、NCCL 2.28〜2.30 に存在）を用いると、単体のマイクロベンチマークにおいては、これらのホストメモリ経由のコピーを GPU の DMA コピーエンジンへオフロードできる。

```
if d2l.num_gpus() >= 2:
    # An undocumented parameter of NCCL's transport through host memory. It
    # exists in NCCL 2.28.9 (bundled with PyTorch 2.11) and 2.30.4 (with
    # jaxlib 0.12); the library of NCCL 2.31.2 no longer contains it
    beta_ce = allreduce_beta({'NCCL_SHM_USE_CUDA_MEMCPY': '1'})
    print(f'bus bandwidth {1e-9 / beta_link:.2f} GB/s by default, '
          f'{1e-9 / beta_ce:.2f} GB/s with the copy engines '
          f'({beta_link / beta_ce:.1f}x)')
```

```
bus bandwidth 2.43 GB/s by default, 10.98 GB/s with the copy engines (4.5x)
```

NVLink（片方向あたり 450〜900 GB/s、13.2.4.2 節）で接続されたデータセンタ向け GPU 上では、ResNet-18 の 45 MB のオールリデュースは 1 ミリ秒を優に切る時間で完了し、ほぼ線形のスケーリングを達成する。しかし理想的なスケーリングが得られたとしても、データ並列は依然としてパラメータとオプティマイザ状態の全 $16N$ バイトをすべてのカード上に複製する。13.7 節では、この状態を複数の GPU に分割する手法を扱う。

13.6.3 まとめ

実用データ並列は、NCCL を介して同期する k 個の独立したプロセス、あるいはコンパイラ分割型の単一プログラムのいずれかによって k 基の GPU を駆動し、勾配を 25〜30 MiB のバケットにまとめてオールリデュースを逆伝播とオーバーラップさせる。単一 GPU の参照結果に対する 1 ステップチェックを行えば、バッチ正規化がシャードごとの統計量を用いているのか、追加の集団通信を介して大域統計量を用いているのかが明らかになる。ホストメモリ経由で通信する 4 基の RTX 4090（バス帯域幅 2.4 GB/s）では、弱スケーリングのもとで単一 GPU の 3.1–3.3× のスループットが得られたが、カーネルプロファイリングの結果、ホスト中継の NCCL カーネルによる SM 競合が、通信と逆伝播計算のオーバーラップで節約された時間の大部分を帳消しにしていることがわかる。

13.6.4 演習問題

1. [code] **データの分割**。`minibatches` では、インデックスは `P('data')` で配置されている。
   1. 代わりに `P()` で配置し、すべての GPU が全インデックスを受け取るようにせよ。コンパイルされた `train_step` にどの集団通信が含まれるか、また `step_time(2, 256)` が `step_time(1, 256)` と比べてどうなるかを予測し、その両方を確認せよ。
   2. `P('data')` に戻したうえで、`train_step` の内部で `X[idx]` を `NamedSharding(mesh, P(None, 'data'))` に制約し、画像の高さを GPU 間で分割せよ。3×3 畳み込みのために隣接する GPU 間でどの境界の値を交換する必要があり、コンパイルされた HLO にはどの集団通信が現れるか。
2. **弱スケーリングの外挿**。k 基の GPU 上での 1 ステップに、単一 GPU 時間 t_1 と、リング通信量 $2(k-1)S/k$ に比例する露出通信時間がかかるとする。
   1. 弱スケーリング効率 $E(k) = t_1/t_{\text{step}}(k)$ を単一の定数を用いて表し、実測された $E(2)$ でその定数を較正したうえで、$E(4)$ に対する予測を 13.6.2 節の実測値と比較せよ。

2. a 個のマイクロバッチ（GPU あたり 256 例、最後のマイクロバッチでのみ通信する）にわたって勾配累積を行う場合、整数 a のうち $E(4)$ を 95% 以上に引き上げる最小の値を求め、グローバルバッチサイズがどのように変化するか述べよ。

3. [code] **バケットサイズ。**●バケットオーバーラップ（Figure 13.20）を次の仮定のもとでモデル化せよ。時間 t_b の逆伝播が等サイズの n 個のバケットを時刻 $i\,t_b/n$（$i = 1, \dots, n$）に順次放出し、各バケットのオールリデュースが逐次的に $\alpha + \tau/n$ を要するものとする（ただし $\tau = S\,\beta_{\text{link}}$）。
   1. 逆伝播終了後の露出通信時間が次の式で与えられることを示せ。
   $$t_{\text{exposed}}(n) = \max\Big(\alpha + \frac{\tau}{n},\; n\alpha + \tau - \frac{n-1}{n}\,t_b\Big), \tag{13.13}$$
   さらに、最適な n を $\tau < t_b$ の場合と $\tau > t_b$ の場合のそれぞれについて求めよ。
   2. 新規プロセス上で `--xla_gpu_all_reduce_combine_threshold_bytes` を 1 MB から 100 MB まで変化させ、生成される勾配オールリデュースの回数を数えて、実測された露出通信時間をこのモデルと比較せよ。

4. [code] **バッチ正規化の統計量。**●`nnx.jit` と `Auto` メッシュ軸のもとでは、バッチ正規化はすべての GPU にわたって大域統計量を計算する。
   1. `allreduce_megabytes(compiled)` に現れる 74 個の小さなオールリデュースの内訳を、`BatchNorm` 層の数（`resnet18` 内）ならびに順伝播・逆伝播の構成から説明せよ。
   2. コンパイル前に `model.eval()` を呼び出して `BatchNorm` がバッチ統計量の代わりに移動平均を用いるようにし、小さなオールリデュースが消えることを確認したうえで、$k = 1, 2, 4$ においてそれらがどれだけのステップ時間を要していたかを測定せよ。

13.7 訓練状態のシャーディング

13.4 節で見たように、fp32 の Adam を用いた訓練では、重み、その勾配、および Adam の 2 つのモーメント推定値にそれぞれ 4 バイトずつ、パラメータあたり合計 16 バイトを保持する。18 ブロック・1B パラメータの GPT では、この固定状態だけで 1 トークンも読み込む前に 24 GiB カードのうち 15.1 GiB を占め、さらにデータ並列（data parallelism、13.5 節、13.6 節）はこの 16 バイトすべてを各 GPU に複製する。GPT の深さを 2 倍の 36 ブロック（1.92B パラメータ、状態サイズ 28.6 GiB）にすると、1 基の GPU と同様、4 基の 24 GiB GPU によるデータ並列でも訓練できなくなる。

1 枚のカードに収まらないモデルを k 基の GPU で訓練できるだろうか。答えは可能である。なぜなら、GPU がブロックの重み全体を必要とするのはそのブロックの順伝播または逆伝播を計算している間だけであり、全勾配や完全なオプティマイザ状態（optimizer state）はそもそも一切必要としないからである。したがって、k 基の各 GPU は重み、勾配、およびオプティマイザのモーメントを $1/k$ ずつ保持し、各ブロックの実行直前にその重みをオンザフライで集め、各ブロックの勾配が生成され次第すぐに reduce-scatter すればよい。

```
# XLA reads its flags once, when JAX starts, so this cell runs before the
# imports. The latency-hiding scheduler orders the step with its memory in
# view; without it, the sharded step needs more memory than JAX may
# allocate by default (see the note on stage 2). On GPUs without
# peer-to-peer access, such as ours, set NCCL_LOCAL_REGISTER=0 in the
# environment before starting: NCCL's buffer registration fails there and
# prints a warning at every collective.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_enable_latency_hiding_scheduler=true')
```

```
import collections
import re
from jax.sharding import NamedSharding, PartitionSpec as P
```

まず、形状のみを保持するメタデバイス上で、36 ブロックの GPT が持つ $16N$ バイトの訓練状態（training state）が単一の RTX 4090 の容量 24 GiB を超えることを確認しよう。

```
config = dict(vocab_size=50257, num_hiddens=2048, num_heads=16,
              num_blks=36, pos='learned')
k = 2                       # GPUs, from the all-gather below on
key = d2l.get_key()
N = d2l.num_params(nnx.eval_shape(lambda: d2l.GPT(**config,
                                                  rngs=nnx.Rngs(key))))
print(f'{N / 1e9:.2f} billion parameters, 16N = {16 * N / 2**30:.1f} GiB')
if jax.default_backend() == 'gpu':
    limit = jax.devices()[0].memory_stats()['bytes_limit']
    print(f'JAX may allocate {limit / 2**30:.1f} GiB of one GPU')
```

```
1.92 billion parameters, 16N = 28.6 GiB
JAX may allocate 22.1 GiB of one GPU
```

13.7.1 オプティマイザ状態、勾配、重みの分割

標準的なデータ並列では、k 基すべての GPU が Adam のモーメント推定値の同一なコピーを保持し、同一のオプティマイザ更新を実行する。**ZeRO**（**Zero Redundancy Optimizer**）ファミリー (Rajbhandari et al. 2020) は、累積的な 3 つの段階によってこの冗長性を排除する（Table 13.5）。

- **ZeRO Stage 1** はオプティマイザ状態のみを分割する。各 GPU はパラメータの $1/k$ について Adam の 2 つのモーメントを保持する（パラメータあたり $8 + 8/k$ バイト）。allreduce（= reduce-scatter + all-gather、13.5.3 節）の代わりに、GPU 間で勾配を reduce-scatter することで、各 GPU が自身の $1/k$ シャードに対応する合算済み勾配を受け取り、自身のパラメータを更新したうえで、更新後の重みを all-gather する。移動するバイト数はデータ並列と完全に同じである。
- **ZeRO Stage 2** は勾配も分割する（パラメータあたり $4 + 12/k$ バイト）。逆伝播中に各勾配バケットが埋まり次第すぐに reduce-scatter され、自身が所有しない $1 - 1/k$ の勾配シャードは解放される。これもデータ並列に対する追加の通信バイトコストはゼロである。
- **ZeRO Stage 3**、すなわち**完全シャーディングデータ並列**（Fully Sharded Data Parallelism、FSDP）(Y. Zhao et al. 2023) は、重みもシャーディングする（パラメータあたり $16/k$ バイト）。順伝播または逆伝播でブロックを実行する直前に all-gather でそのブロックの重み全体を組み立て、ブロックが完了した直後に所有しない重みシャードを解放し、そのブロックの勾配は reduce-scatter する（Figure 13.21）。

Table 13.5: k 基の GPU 上で fp32 の Adam を用いた場合の逆伝播終了時点における GPU あたりの訓練状態、および $k = 2$ における 36 ブロック・19.2 億パラメータの GPT での値。

各 GPU の保持内容	重み	勾配	Adam 状態	パラメータあたりバイト数	GPT（$k = 2$）
データ並列	すべて	すべて	すべて	16	28.6 GiB
ZeRO Stage 1	すべて	すべて	$1/k$	$8 + 8/k$	21.4 GiB
ZeRO Stage 2	すべて	$1/k$	$1/k$	$4 + 12/k$	17.9 GiB

各 GPU の保持内容	重み	勾配	Adam 状態	パラメータあたりバイト数	GPT（$k=2$）
ZeRO Stage 3（FSDP）	$1/k$	$1/k$	$1/k$	$16/k$	14.3 GiB

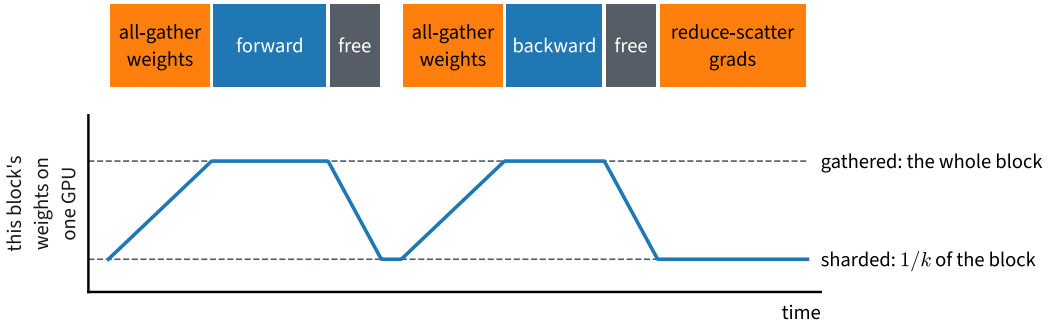


Figure 13.21: 1 基の GPU 上における FSDP 下での 1 ブロックの重みのライフサイクル。使用していない間、ブロックはシャーディングされており、各 GPU はその $1/k$ を保持する（他のブロックは常にシャーディングされたままである）。順伝播のために all-gather がブロック全体を組み立て、終了後に解放する。逆伝播のために 2 回目の all-gather が再びブロックを組み立てて終了後に解放し、reduce-scatter によって各 GPU には担当分の勾配が残る。

FSDP はステップごとに $S = 4N$ バイトに対する 2 回の all-gather（順伝播と逆伝播）と 1 回の reduce-scatter を実行するため、リング allreduce が 2 回の集団通信（collective）フェーズで済むところを 3 フェーズ要し、**データ並列の $1.5\times$ の通信量**を移動させる。Table 13.6 に並列訓練で使われる主要な 4 つの集団通信演算をまとめ、Figure 13.22 にそれぞれが 2 基の GPU 間で何を移動させるかを示す。

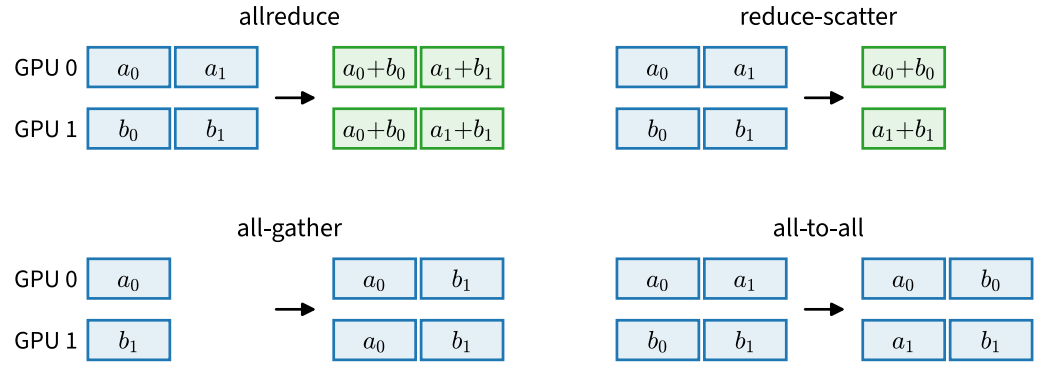


Figure 13.22: 2 基の GPU 間における実行前後のデータ移動として見た 4 つの集団通信。GPU 0 はチャンク a_0, a_1、GPU 1 はチャンク b_0, b_1 を初期状態とし、all-gather では GPU g は自身のシャードであるチャンク g のみを初期状態とする。元のチャンクを青、和を緑で示す。

Table 13.6: データ並列訓練およびシャーディング訓練を支える集団通信演算。

集団通信	各 GPU が最終的に得る結果	使用箇所
allreduce	全体の和	データ並列における勾配
reduce-scatter	和の $1/k$ シャード 1 つ	ZeRO および FSDP における勾配
all-gather	全シャードを連結したもの	ZeRO および FSDP における重み
all-to-all	各 GPU からの異なるスライス	混合エキスパート（11.7 節）におけるトークンルーティング

13.7.2 メモリと通信

$k = 2$ 基の GPU 上で、GPU あたり 1,024 トークンの系列 2 本とブロックごとの活性化チェックポイント（activation checkpointing、13.4.3 節）を用いて 36 ブロックの GPT を FSDP で実行したときのピークメモリとステップ時間を予測してみよう。

- **メモリ：**シャーディングされた訓練状態は GPU あたり $16N/2 = 14.3$ GiB を占める。チェックポイント保存された活性化値にトークンあたり約 1 MiB（2,048 トークンで約 2 GiB）を要し、さらにその時点で集められている 1〜2 個のブロック（各約 0.2 GB）を加えても、GPU あたりのピークメモリは高々**約 17 GiB** となり、24 GiB を十分に下回る。
- **通信：**3 つの集団通信フェーズはそれぞれ GPU あたり $(k-1)S/k$ バイトを送信するため、合計の通信時間は次式となる。

$$t_{\text{comm}} = \frac{3(k-1)}{k}\, S\, \beta_{\text{link}}. \tag{13.14}$$

  FSDP の重要なスケーリング特性に注目しよう。**GPU を増やすと GPU あたりの状態メモリは k で割られて減る一方、各 GPU がステップごとに送信するバイト数は $1.5S$（$k = 2$ のとき）から $3S$（k が大きいとき）へと増加する。**1 ステップに占める通信の割合を減らせる手段は、GPU あたりに処理するトークン数を増やすか、より高速なインターコネクトを使うことだけである。

まず、256 MiB の all-gather における β_{link} を $k = 2$ 基の GPU 間で計測する。

```
assert jax.device_count() >= k, 'The cells from here on need two GPUs'
mesh = jax.make_mesh((k,), ('data',), devices=jax.devices()[:k],
                     axis_types=(jax.sharding.AxisType.Auto,))
x = jnp.zeros(2**26, device=NamedSharding(mesh, P('data')))  # 256 MiB
# Asking for the result on every GPU makes the compiler gather the parts
gather = jax.jit(lambda x: x, out_shardings=NamedSharding(mesh, P()))
t = d2l.Benchmark(lambda: gather(x), repeats=10).time
beta_link = t / ((k - 1) / k * x.nbytes)  # Seconds per byte sent per GPU
del x                         # Frees its memory before the training step
print(f'bus bandwidth = {1e-9 / beta_link:.1f} GB/s in an all-gather on {k} '
      f'GPUs (beta_link = {1e9 * beta_link:.2f} ns/byte)')
```

```
bus bandwidth = 2.2 GB/s in an all-gather on 2 GPUs (beta_link = 0.46 ns/byte)
```

実測した β_{link} と RTX 4090 の tf32 ピーク性能である 82.6 TFLOP/s（`peak_flops`）を用いて、GPU あたりの状態メモリ（$16N/k$）、通信時間（$3(k-1)S\beta_{\text{link}}/k$）、およびチェックポイント保存時の計算時間（$8NbT/P$）に関する机上予測を評価しよう。

```
b, T = 2, 1024        # Sequences of T tokens per GPU and step
S = 4 * N             # Bytes of the fp32 weights, and of the gradients
peak_flops = 82.6e12  # tf32 FLOP/s of the RTX 4090
print(f'state per GPU, 16N/k: {16 * N / k / 2**30:.1f} GiB')
print(f'communication, 3(k-1)S beta_link/k: '
      f'{3 * (k - 1) / k * S * beta_link:.1f} s per step')
print(f'arithmetic at the peak, 8N bT/P: '
      f'{8 * N * b * T / peak_flops:.2f} s per step')
```

```
state per GPU, 16N/k: 14.3 GiB
communication, 3(k-1)S beta_link/k: 5.3 s per step
arithmetic at the peak, 8N bT/P: 0.38 s per step
```

ホストメモリ経由のバス帯域幅が約 2.2 GB/s の場合、ステップあたり $1.5S = 11.5$ GB のデータ移動には約 5 s かかるのに対し、2,048 トークン分の算術演算はわずか約 0.4 s で終わる。0.4 s の計算では 5 s の通信を隠蔽できないため、1 ステップには約 5 s かかるはずである。

13.7.3 2 基の GPU によるシャーディング訓練

それでは、シャーディングされた 36 ブロック GPT のステップを 2 基の GPU 上で実行する。その際、どの単一 GPU も 28.6 GiB の状態全体を一度に実体化しないようモデルを構築する。

JAX では、`fsdp_sharding` が各パラメータテンソルを k で割り切れる最初の軸に沿ってシャーディングし（`P('data')`）、`create` がこのシャーディング制約を適用することで、各 GPU は `model` と `opt` のうち自身のシャードのみを初期化する。

```
def fsdp_sharding(x):
    """Split the first axis of x whose length k divides across the GPUs;
    replicate x if there is none."""
    for i, n in enumerate(x.shape):
        if n % k == 0:
            return NamedSharding(mesh, P(*[None] * i, 'data'))
    return NamedSharding(mesh, P())

@nnx.jit
def create(key):
    # The key is an argument: XLA would fold a constant key into the program,
    # together with every weight drawn from it, as large constants
    model = d2l.GPT(**config, rngs=nnx.Rngs(key))
    opt = nnx.Optimizer(model, optax.adam(1e-4), wrt=nnx.Param)
    state = nnx.state((model, opt))
    nnx.update((model, opt), jax.lax.with_sharding_constraint(
        state, jax.tree.map(fsdp_sharding, state)))
    return model, opt

model, opt = create(d2l.get_key())  # Each GPU computes only its shards
```

次に、活性化チェックポイントのために各ブロックを `nnx.remat` でラップし、`loss_fn` における各ブロックの活性化値の出力がバッチ方向にシャーディングされた状態（`data`）を保つよう制約して XLA が FSDP の all-gather と reduce-scatter を自動発行するようにし、さらに `model` と `opt` を `train_step` でバッファ譲渡（donate）して更新後の状態が古いシャードをインプレースで上書きするようにする。

```
data = NamedSharding(mesh, P('data'))     # Split along the batch
block = nnx.remat(lambda blk, H: blk(H))  # Recomputed in the backward pass

def loss_fn(model, X):
    H = model.token_emb(X) + model.pos_emb(jnp.arange(X.shape[1]))
    for blk in model.blks:  # Every block's output stays split by batch
        H = jax.lax.with_sharding_constraint(block(blk, H), data)
    logits = model.token_emb.attend(model.norm(H))
    return optax.softmax_cross_entropy_with_integer_labels(logits, X).mean()

# Donated, the old weights and moment estimates make room for the new ones
@nnx.jit(donate_argnums=(0, 1))
def train_step(model, opt, X):
    loss, grads = nnx.value_and_grad(loss_fn)(model, X)
    opt.update(model, grads)
    return loss
```

ステップを実行する前に、train_step をバッチ方向にシャーディングされたトークンバッチ X（形状 (kb, T)）に対してコンパイルし、compiled.memory_analysis() を呼び出して譲渡された状態と一時バッファの GPU あたりのメモリ使用量を確認する。

```
X = jax.device_put(jax.random.randint(d2l.get_key(), (k * b, T), 0,
                                      config['vocab_size']), data)
compiled = train_step.lower(model, opt, X).compile()
mem = compiled.memory_analysis()  # Bytes per GPU
print(f'per GPU: weights and moments '
      f'{mem.argument_size_in_bytes / 2**30:.1f} GiB (donated: '
      f'{mem.alias_size_in_bytes / 2**30:.1f} GiB), temporaries '
      f'{mem.temp_size_in_bytes / 2**30:.1f} GiB')
```

```
per GPU: weights and moments 10.7 GiB (donated: 10.7 GiB), temporaries 5.0 GiB
```

XLA が期待どおり FSDP の集団通信を発行したことを検証するため、collective_megabytes でコンパイル済み HLO テキストを解析し、各集団通信演算の結果バイト数を GPU あたりで集計する。

```
def collective_megabytes(compiled):
    """Megabytes of the results of each kind of collective in a compiled
    program, on one GPU."""
    sizes = collections.Counter()
    for line in compiled.as_text().splitlines():
        # A synchronous collective, or the -done half of an asynchronous
        # pair, whose result type lists only the arrays that it returns
        op = re.search(r' = (.*?) (all-gather|reduce-scatter|all-reduce|'
                       r'all-to-all|collective-permute)(-done)?\(', line)
        for bits, dims in re.findall(r'[a-z]+(\d+)\[([\d,]*)\]',
                                     op[1] if op else ''):
            n = math.prod(int(d) for d in dims.split(',') if d)
            sizes[op[2]] += int(bits) // 8 * n / 1e6
    return dict(sizes)
```

compiled 内の集団通信を集計すると、all-gather（ステップあたり 2 回）と reduce-scatter（ステップあたり 1 回）が予測どおりのバイト量を移動させていることが確認できる。

```
    for kind, mb in sorted(collective_megabytes(compiled).items(),
                           key=lambda kv: -kv[1]):
        print(f'{kind}: {mb / 1e3:.2f} GB')
```

```
all-gather: 14.92 GB
reduce-scatter: 3.83 GB
all-to-all: 0.03 GB
all-reduce: 0.00 GB
```

ここで train_step をウォームアップしたうえで 3 回の反復実行を計測し、さらにステップをプロファイリングして NCCL 通信時間（comm）とデバイスのピークメモリ（peak_bytes）を記録する。

```
    train_step(model, opt, X).block_until_ready()  # Warm-up, finished
    times = [d2l.Benchmark(lambda: train_step(model, opt, X), warmup=0,
                           repeats=1).time for _ in range(3)]
    kernels = d2l.kernel_times(lambda: train_step(model, opt, X), steps=1)
    comm = sum(ms for name, ms in kernels.items() if 'nccl' in name.lower()) / 1e3
    peak_bytes = jax.devices()[0].memory_stats()['peak_bytes_in_use']
```

最後に、実測された GPU あたりのピークメモリ、ステップ所要時間の中央値、および GPU 0 が NCCL 通信カーネルに費やした時間を出力する。

```
    print(f'per GPU: peak memory {peak_bytes / 2**30:.1f} GiB; '
          f'{np.median(times):.2f} s per step ({min(times):.2f} to '
          f'{max(times):.2f})\nof each step, GPU 0 spends {comm:.2f} s in '
          f'communication kernels')
```

```
per GPU: peak memory 15.8 GiB; 5.29 s per step (5.28 to 5.50)
of each step, GPU 0 spends 5.01 s in communication kernels
```

両方の GPU はカードあたりピーク約 16 GiB（予測上界の 17 GiB 未満）で 36 ブロックのモデルを保持しており、各ステップの所要時間は約 5.5 s で、GPU 0 はステップの 90% 超を NCCL 通信カーネルに費やしている。これは机上予測である 5 s から 1/6 以内に収まっている。NVLink 5（片方向あたり 900 GB/s）を用いれば、同じ 11.5 GB の集団通信はわずか約 13 ms で完了し、0.4 s の計算時間の 1/30 未満となる。

この比較から、**いつ**シャーディングすべきかもわかる。訓練状態が妥当なサイズのマイクロバッチとともに単一 GPU に収まるモデル（18 ブロックの GPT など）では、データ並列と勾配累積（gradient accumulation、グローバルステップごとに 1 回だけ通信する）を組み合わせたほうが、PCIe 経由の FSDP（マイクロバッチごとに重みを 2 回再収集する）よりもはるかに高速に訓練できる。

13.7.4 まとめ

データ並列は fp32 Adam の状態 $16N$ バイトすべてを各 GPU に複製する。ZeRO はこの状態を k 基の GPU に分割する。Stage 1 はオプティマイザのモーメントをシャーディングし（パラメータあたり $8 + 8/k$ バイト）、Stage 2 は勾配もシャーディングするが（パラメータあたり $4 + 12/k$ バイト）、どちらもステップあたりに移動するデータ量はリング allreduce と完全に同じ $2(k-1)S/k$ バイトである。Stage 3（FSDP）は重みもシャーディングし（パラメータあたり $16/k$ バイト）、順伝播と逆伝播のたびに各ブロックの重みを収集し勾配を reduce-scatter することで、データ並列の $1.5\times$ の通信量（GPU あたり $3(k-1)S/k$ バイト）で動作する。シャーデ

ィングにより、単独のカードには収まらない 28.6 GiB の状態を持つ 36 ブロックの GPT を、2 基の 24 GiB GPU で訓練できるようになる。

13.7.5 演習問題

1. **シャーディングのコスト。**●GPU あたり・ステップあたりの通信量を、fp32 の重みまたは勾配 1 セット分のバイト数である $S = 4N$ を単位として数えよ。
   1. ZeRO Stage 3 のある変種では、収集した各ブロックを順伝播から逆伝播が完了するまでメモリ上に保持し続ける。これによりどの集団通信が不要になり、Table 13.5 のメモリ削減量のうちどれだけを手放すことになるか。
   2. 混合精度 FSDP は、各 GPU のマスター重みと Adam シャードを fp32 に保ったまま、重みの all-gather と勾配の reduce-scatter を bf16 で実行する。ステップあたりの転送バイト数を fp32 データ並列と比較し、パラメータあたりのメモリ量を Table 13.5 の最終行と比較せよ。
   3. $k = 2$ 基の GPU 上での 36 ブロック GPT において、実測した β_{link} のもとで、チェックポイント保存時の計算時間（83 TFLOP/s でトークンあたり $8N$ FLOPs）が通信時間 Equation 13.14 と釣り合うには、1 ステップで GPU あたり何トークン必要か。その活性化値はカードに収まるか。また、FSDP がマイクロバッチごとに重みを再収集する場合に勾配累積が役に立たないのはなぜか。
2. [extended] **オプティマイザ状態のシャーディング。**●ZeRO Stage 1 (Rajbhandari et al. 2020) を、`train_step`（13.6 節）と `optax.adam` を出発点として宣言的に実装せよ。モデルパラメータは複製された状態（`P()`）に保ち、`nnx.state(opt)` のリーフを `fsdp_sharding` で配置せよ。
   1. 必要に応じて `train_step` の末尾で `jax.sharding.reshard` を用いてパラメータが複製状態を保つようにし、複数ステップにわたるパラメータ更新が複製版の Adam と一致することを検証せよ。
   2. コンパイル済みステップを `collective_megabytes` で調べ、その集団通信および $k = 2$ と $k = 4$ における使用中メモリ・ピークメモリを Table 13.5 と比較せよ。
3. [code] **テンソル並列とデータ並列の比較。**●MLP ブロック $\mathbf{Y} = \phi(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2$（ただし $\mathbf{X} \in \mathbb{R}^{n\times d}$、$\mathbf{W}_1 \in \mathbb{R}^{d\times 4d}$、$\mathbf{W}_2 \in \mathbb{R}^{4d\times d}$）を k 基のデバイスに分散することを考える。テンソル並列（Figure 13.15、(Shoeybi et al. 2019)）では、$\mathbf{W}_1$ を列方向（`P(None, 'data')`）に、$\mathbf{W}_2$ を行方向（`P('data', None)`）に分割する。
   1. 各デバイスが通信なしに $n \times d$ の部分出力を計算でき、その k 個の部分出力を合算すると $\mathbf{Y}$ が得られることを示せ。順伝播にはどの集団通信が必要で、逆伝播で $\nabla_{\mathbf{X}}$ を求めるにはどの集団通信が必要か。
   2. デバイスあたり・ステップあたりに送信される値の個数をテンソル並列とデータ並列で比較し、テンソル並列のほうがデータ送信量が少なくなるトークン数 n の上限を求めよ。それにもかかわらず、テンソル並列が高速なノード内 NVLink ドメインに限定されるのはなぜか。
   3. $\mathbf{W}_1$ を `P(None, 'data')`、$\mathbf{W}_2$ を `P('data', None)` で配置し、$\mathbf{X}$ を `Auto` メッシュ上で複製したうえで、順伝播と逆伝播を JIT コンパイルして発行される集団通信を調べよ。
4. **次ブロックのプリフェッチ。** FSDP における順伝播で、1 ブロックの all-gather 時間を g、計算時間を c とする。オーバーラップなしでは L ブロックに $L(g + c)$ かかるが、プリフェッチではブロック $l + 1$ の all-gather をブロック l の計算中に発行する。
   1. プリフェッチによって順伝播の時間が $g + (L - 1)\max(g, c) + c$ に短縮されることを示し、どのようなスケジュールでも $L\max(g, c)$ より速くできない理由を説明せよ。
   2. g と c の値を 36 ブロック GPT の 1 ブロック（50M パラメータ、GPU あたり 2,048 トークン、83 TFLOP/s）について本節の PCIe リンクと NVLink 5（900 GB/s）のそれぞれで求め、各インターコネクトでプリフェッチがどれだけ時間を短縮するか述べよ。

13.8 ケーススタディ：Transformer の最適化

本章の各節では、訓練ステップを遅くする原因とそれを取り除く変更を 1 つずつ切り分けてきた。具体的には、ホストディスパッチのオーバーヘッド、小さなカーネルによるメモリトラフィック、ビット幅の広い数値形式、そして単一 GPU の限界である。実際の訓練ステップではこれら複数が同時に発生し、1 つを取り除くとボトルネックが別の箇所へと移る。本節では実務家と同じように 1 つのモデルを取り上げ、その訓練ステップをどこまで高速化できるか、またどの順序で変更を適用すべきかを検証する。

対象とするモデルは、11.2 節の GPT の幅を 2 倍の 512 に広げたもの（1900 万パラメータ）である。128 バイトのウィンドウ 64 個（8,192 トークン）に対する 1 回の訓練ステップは、約 $6 \times 1.9 \times 10^7 \times 8{,}192 \approx 9 \times 10^{11}$ FLOPs の演算を実行する。内訳は、1 パラメータ・1 トークンあたり順伝播で 2 FLOPs、逆伝播で 4 FLOPs である。RTX 4090（bf16 で 165 TFLOP/s）では、この算術演算に必要な時間は 6 ms 未満である。ところが、AdamW を用いた通常の fp32 ステップには約 25 ms かかる。数値形式だけで約 6 ms（その 19 ms の差のうち）を占めており、これは tf32 テンソルコアが bf16 の半分の速度で動作するため 11 ms を要するからである。残りの 13 ms ほどは、テンソルコアがピーク未満で動作しているかアイドル状態にある時間である。

ここでも 13.1 節の手順に従い、演算量と転送量を数え、予測し、計測し、その差を説明する。各変更を加える前に直前のステップのプロファイルから得られる高速化の上限を求め、適用後に実測値と比較する。各変更はそれまでの変更の上に累積的に積み重ねて計測する。本節の最後には、最速化したステップでもモデルが正しく訓練できることを確認し、残りの実行時間の内訳を明らかにする。

```
# Let XLA overlap collectives with computation. JAX reads XLA_FLAGS once,
# when it starts, so this cell runs before the imports. Our outputs come
# from a run with three more settings in the environment, which you set
# before starting: --xla_gpu_enable_triton_gemm=false in XLA_FLAGS, so that
# XLA calls cuBLAS for the matrix products (the Triton kernels it would
# generate run the products of the large minibatches more slowly on the
# RTX 4090); NCCL_LOCAL_REGISTER=0, since NCCL's buffer registration fails
# on GPUs without peer-to-peer access and would print a warning at every
# collective; and XLA_PYTHON_CLIENT_ALLOCATOR=vmm, which maps each array
# into one contiguous range: the fp32 step of the training check needs
# about 13 GiB in one piece after many other arrays have come and gone.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_enable_latency_hiding_scheduler=true')
```

```
import re
from jax.sharding import NamedSharding, PartitionSpec as P
```

13.8.1 実験設定

本節のすべての変更は同一の基準で評価するため、モデル、データ、およびスループットのベンチマークは一貫して固定する。

13.8.1.1 モデルとデータ

訓練するのは `d21.GPT`（11.2 節）であり、それぞれ 8 個のアテンションヘッドを持つ 6 個の Pre-Norm ブロック、回転位置埋め込み（rotary position embedding）、RMSNorm、SwiGLU、

および出力層と重み共有された入力埋め込みから構成される。データは、256 バイトの語彙上で WikiText-103 からランダムなオフセットで切り出した 128 バイトのウィンドウである（`d2l.WikiText103`、9.3 節）。

行列積が十分に計算律速（compute-bound）の領域に入るよう、幅を 256 から 512 に倍増させる（13.1 節）。$n \times d$ の活性化行列と $d \times d'$ の重み行列の積は、$2ndd'$ FLOPs の演算を実行し、$b(nd + dd' + nd')$ バイト（1 要素あたり b バイト）を転送する。$n = 8{,}192$ トークン・幅 $d = 512$ の場合、ブロック内の各射影は 125–185 FLOP/byte（tf32、$b = 4$）および 250–370 FLOP/byte（bf16、$b = 2$）に達し、RTX 4090 のリッジポイント（82 FLOP/byte：tf32、164 FLOP/byte：bf16）を大きく上回る。したがって、より高速な数値形式を使えば、これらの行列積の時間はそのまま短縮される。幅 256 ではリッジポイントをまたぐことになる（演習問題 6）。

270 MB の訓練コーパスは、フラットなバイト配列として GPU メモリに収まる。次のセルでは、このコーパスを GPU 上に配置し、ホスト間の転送なしにデバイス上で直接ミニバッチをサンプリングする `windows` を定義して、モデルを構築する。

```
data = d2l.WikiText103(batch_size=64, num_steps=128)
corpus = jnp.asarray(data.train)  # One byte per token

def windows(corpus, batch_size, num_steps=128, *, sharding=None):
    """Endless minibatches of windows at random offsets, each position with
    the next byte as its target; `sharding` spreads them over devices."""
    offsets = jnp.arange(num_steps + 1)
    while True:
        starts = jax.random.randint(d2l.get_key(), (batch_size, 1), 0,
                                    len(corpus) - num_steps)
        if sharding is not None:
            starts = jax.device_put(starts, sharding)
        W = corpus[starts + offsets].astype(jnp.int32)
        yield W[:, :-1], W[:, 1:]

model = d2l.GPT(len(data.vocab), num_hiddens=512,
                rngs=nnx.Rngs(d2l.get_key()))
num_params = d2l.num_params(model)
print(f'{num_params / 1e6:.1f}M parameters, '
      f'{len(corpus) / 1e6:.0f}M training bytes on the GPU')
```

```
19.0M parameters, 270M training bytes on the GPU
```

1 パラメータあたり 16 バイト（重み、勾配、および 2 つの Adam モーメント）では、19M パラメータの訓練状態が占めるメモリ使用量はわずか 0.3 GiB であり、勾配累積（gradient accumulation）もシャーディングも必要ない（13.4 節、13.7 節）。また、コーパスを GPU 上に保持することで、訓練ステップを入力パイプラインから切り離せる（13.2 節）。

13.8.1.2 スループットの計測

ミニバッチのサンプリング、順伝播と逆伝播、およびオプティマイザの更新を含むエンドツーエンドの訓練スループットを、毎秒トークン数で計測する。

設定項目	値
ハードウェア	RTX 4090 GPU 4 基のうち 1 基（データ並列では 2 基および 4 基）

設定項目	値
モデル	`d2l.GPT`：6 ブロック、幅 512、8 ヘッド、バイト語彙、コンテキスト長 128
データ	毎ステップ GPU 上でランダムに抽出する WikiText-103 の 128 バイトウィンドウ
オプティマイザ	AdamW（lr $= 10^{-3}$、重み減衰 0.01）
数値精度	fp32（行列積は tf32、bf16 の導入前まで）
時間計測	`d2l.Benchmark`：ウォームアップ 20 ステップの後、デバイス同期を挟む 3 個のウィンドウ（各 50 ステップ）
結果	3 ウィンドウの毎秒トークン数の中央値、直前の採用構成に対する比率、ウィンドウ間の相対的なばらつき

`tokens_per_second` は `d2l.Benchmark`（13.1 節）を通じてこれらの規則を適用する。`record` は各結果を保存し、直前の採用構成に対する高速化率と、3 つの計測ウィンドウ間の相対的なばらつき（range/median）を表示する。`kept=False` を渡した構成は分岐として保存され、後続の構成の土台にはならない。

```
def tokens_per_second(step, batches, warmup=20, repeats=50, num_windows=3):
    """Median training throughput of `step` on minibatches drawn from
    `batches`, over `num_windows` timed windows of `repeats` steps each, and
    the spread of the windows (largest minus smallest) relative to it."""
    X, _ = next(batches)
    speeds = [math.prod(X.shape) / d2l.Benchmark(
                  lambda: step(*next(batches)), warmup if i == 0 else 0,
                  repeats).time for i in range(num_windows)]
    return np.median(speeds), np.ptp(speeds) / np.median(speeds)

results, branches = {}, {}
def record(name, measured, kept=True):
    """Store a measured throughput and print it with its ratio to the last
    kept configuration and the spread of its windows; a configuration that
    is not kept is stored in `branches`."""
    speed, spread = measured
    previous = list(results.values())[-1] if results else speed
    (results if kept else branches)[name] = speed
    print(f'{name}: {speed:,.0f} tokens/s ({speed / previous:.2f}x), '
          f'windows within {spread:.1%}')
```

ホスト側の処理が多いステップはホストのバックグラウンド動作の影響を受けやすく、コンパイル済みやグラフキャプチャ済みのステップよりもウィンドウ間のばらつきが大きくなる。日々のホスト負荷の変動は 1 回の実行における連続したウィンドウ間の変動よりも大きいため、高速化率は小数第 1 位まで読み取り、数パーセント程度の小さな改善はプロファイルがそれを予測している場合にのみ信頼する。

13.8.2 単一 GPU

まず単一 GPU 上で、11.2 節のベースライン実装を出発点として、1 つずつ変更を加えながら訓練ステップを最適化する。

13.8.2.1 ベースライン

ベースラインは、AdamW を用いた 11.2 節の標準的な訓練ステップであり、順伝播、交差エントロピー損失、逆伝播、およびパラメータ更新からなる。

`train_step` は `nnx.jit` でコンパイルされる（`d2l.train_steps` と同様、9.3 節）。モデルとオプティマイザを引数にとり、fp32 のロジットから交差エントロピー損失を計算する。

```
def train_step(model, optimizer, X, Y):
    """Loss, gradients, and AdamW update for one minibatch."""
    def loss_fn(model):
        logits = model(X).astype(jnp.float32)
        return optax.softmax_cross_entropy_with_integer_labels(
            logits, Y).mean()
    loss, grads = nnx.value_and_grad(loss_fn)(model)
    optimizer.update(model, grads)
    return loss

adamw = optax.adamw(1e-3, weight_decay=0.01)
optimizer = nnx.Optimizer(model, adamw, wrt=nnx.Param)
jit_step = nnx.jit(train_step)
step = lambda X, Y: jit_step(model, optimizer, X, Y)
record('baseline', tokens_per_second(step, windows(corpus, 64)))
```

```
baseline: 340,062 tokens/s (1.00x), windows within 10.7%
```

予想どおり、ベースラインではステップごとのホスト処理の影響により、ウィンドウ間のばらつきが後の構成に比べて約 10 倍大きい。ステップの実行時間の内訳を調べるため、`profile` は `d2l.kernel_times` で 5 ステップ分を記録し、`d2l.show_profile`（13.1.4 節）によって内訳（ステップの実経過時間に対するデバイスの総カーネル時間、融合アテンションカーネルを含む行列積が占める割合、および上位カーネル）を表示し、最終的な時間集計のためにプロファイルを保存する。

```
profiles = {}
def profile(step, batches):
    """Kernel profile of `step`, the configuration recorded last, on
    minibatches drawn from `batches`: printed against the throughput that
    `record` measured, and stored in `profiles` under the same name."""
    name, speed = list(results.items())[-1]
    X, _ = next(batches)
    kernels = d2l.kernel_times(lambda: step(*next(batches)), steps=5)
    d2l.show_profile(kernels, 1e3 * math.prod(X.shape) / speed, rows=4)
    profiles[name] = kernels

profile(step, windows(corpus, 64))
```

```
wall clock 24.09 ms per step, device busy 23.61 ms (98%), matrix products 65%
  3.02 ms  13%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_32x3_n
  1.99 ms   8%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_32x3_n
  1.90 ms   8%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_32x3_t
  1.39 ms   6%  loop_add_multiply_fusion_5
```

デバイスはステップのほぼ全期間にわたってカーネルを実行しており、行列積がその時間の約 3 分の 2 を占めている。ホスト側の処理はおおむね追いついているため、このステップのボトルネ

ックはデバイス側の処理にある。

nnx.jit がステップ全体をコンパイルするため、XLA はすでに要素ごとの演算を前後のカーネルに融合している。行列積以外のデバイス時間の 3 分の 1 はこれらの融合カーネルに費やされており、依然としてデータ転送量によって律速されている。

13.8.2.2 ホストオーバーヘッドと小さなカーネル

行列積以外に費やされる時間には 2 つの改善策があり、プロファイルからそれぞれの改善幅の上限が求まる。この上限を与えるのが**アムダールの法則**（Amdahl's law）である (Amdahl 1967)。ある変更によってステップの割合 f が s 倍に高速化されるとすると、ステップ全体の高速化率は $1/(1-f+f/s)$ となり、s をどれほど大きくしても $1/(1-f)$ 未満にとどまる。これを 2 つの改善策に当てはめると次のようになる。

1. **ホストオーバーヘッドの削減**は、デバイスがホストを待つアイドル状態の隙間を対象とする（13.1 節）。その高速化率は、ステップに占めるアイドル時間の割合によって上から抑えられる。
2. **カーネル融合**（kernel fusion）は、帯域幅律速（bandwidth-bound）な要素ごとのカーネルの連鎖を統合し、中間配列をオンチップにとどめる。その高速化率は、行列積以外のデバイス時間によって上から抑えられる。

このステップはすでに XLA によって融合されており、GPU のホスト待ちはステップの 2% にすぎないため、現時点でホスト処理を削減しても得られる改善は高々 2% である。それでもここで取り除いておくのは、これがステップごとの固定コストであり、後の最適化でデバイス実行時間が短縮されると表面化するためである。nnx.jit は呼び出しのたびに Python のモデルおよびオプティマイザのオブジェクトを走査して配列の抽出と更新を行う。以下では、何もしない空の関数を用いて、通常の nnx.jit、nnx.jit(..., graph=False)（ツリーモード）、および nnx.jit_partial(..., graph=False)（モデルとオプティマイザ状態を一度だけ束縛し、各ステップではミニバッチのみを渡す方式）の走査オーバーヘッドをベンチマークする。

```
print(f'{len(jax.tree.leaves(nnx.state((model, optimizer))))} arrays in '
      'the model and the optimizer')
noop = lambda model, optimizer: None
walk, tree = nnx.jit(noop), nnx.jit(noop, graph=False)
bound = nnx.jit_partial(noop, model, optimizer, graph=False)
for name, call in (('nnx.jit', lambda: walk(model, optimizer)),
                   ('nnx.jit, graph=False', lambda: tree(model, optimizer)),
                   ('nnx.jit_partial', bound)):
    seconds = d2l.Benchmark(call, repeats=100).time
    print(f'{name}: {1e3 * seconds:.2f} ms per call that does nothing')
```

```
146 arrays in the model and the optimizer
nnx.jit: 8.69 ms per call that does nothing
nnx.jit, graph=False: 1.76 ms per call that does nothing
nnx.jit_partial: 0.23 ms per call that does nothing
```

nnx.jit を介した空の呼び出しにはホスト時間で 9 ms 近くかかっており、これはステップのデバイス時間の 3 分の 1 以上に相当する。このコストはホストがデバイスより先行している間だけ隠蔽されるが、ツリー走査に切り替えると大部分が消え、nnx.jit_partial で束縛すると残りもほぼ完全に解消される。次に、nnx.jit_partial が nnx.jit と同一の損失および重み更新を生成することを確認する。

```
twin = nnx.clone((model, optimizer))  # Equal states for the check
step = nnx.jit_partial(train_step, model, optimizer, graph=False)
X, Y = next(windows(corpus, 64))
loss, twin_loss = step(X, Y), jit_step(*twin, X, Y)
gaps = jax.tree.map(lambda a, b: jnp.abs(a - b).max(),
                    nnx.state(model, nnx.Param), nnx.state(twin[0], nnx.Param))
gap = max(float(g) for g in jax.tree.leaves(gaps))
print(f'losses {loss:.4f} and {twin_loss:.4f}, largest difference of the '
      f'updated weights {gap:.1e}')
assert gap < 1e-6
record('+ jit_partial', tokens_per_second(step, windows(corpus, 64)))
```

```
losses 2.3074 and 2.3074, largest difference of the updated weights 0.0e+00
+ jit_partial: 343,869 tokens/s (1.01x), windows within 0.2%
```

更新後の重みは完全に一致し、スループットは約 1% 向上する。これは上限の 2% およびベースラインのノイズ幅の範囲内である。

2 回目のプロファイルから、ステップがどのように変化したかがわかる。

```
profile(step, windows(corpus, 64))
```

```
wall clock 23.82 ms per step, device busy 23.61 ms (99%), matrix products 65%
  3.03 ms  13%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_32x3_n
  1.99 ms   8%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_32x3_n
  1.90 ms   8%  Kernel2<cutlass_80_tensorop_s1688gemm_128x128_32x3_t
  1.39 ms   6%  loop_add_multiply_fusion_5
```

デバイスは依然としてステップのほぼ全期間で稼働しており、行列積がデバイス時間の約 3 分の 2 以上を占めるようになった。

`nnx.jit_partial` はホスト側のディスパッチオーバーヘッドのみを削減したため、デバイスのカーネル時間は変化していない。

13.8.2.3 低精度化

GPU が常時稼働し、デバイス時間の大部分を行列積が占めるようになったため、次に取り組むべきは行列積の高速化である。tf32 から bf16 に切り替えると、RTX 4090 のテンソルコアのピークスループットは 2 倍の 165 TFLOP/s になり、bf16 の活性化値のメモリ使用量は半分になる（13.4 節）。行列積がステップの割合 $f \in [2/3, 0.7]$ を占め、行列積のみが 2 倍に高速化されるとすると、ステップ時間は元の $1 - f/2$ に短縮される。これは約 $1.5\times$ の高速化に相当する（行列積以外のカーネルのデータ転送量も半減すれば $2\times$ に近づく）。

ここでは 13.4.2 節の混合精度（mixed precision）のレシピを適用する。すなわち、行列積（アテンションを含む）と中間の要素ごとの演算は bf16 で実行する一方、マスター重み、勾配、Adam モーメント、正規化、損失、および残差ストリーム（residual stream）は fp32 のまま維持する（残差ストリームを fp32 に保つことで、小さなブロック更新が bf16 の 8 ビット仮数部で丸め消えるのを防ぐ）。bf16 は fp32 と同じ指数部の範囲を持つため、損失スケーリング（loss scaling）は必要ない。時間計測の前に、bf16 の順伝播が fp32 のロジットと bf16 の精度範囲内（約 1%）で一致することを確認する。

`d2l.to_bf16`（13.4.2 節）は線形層に `dtype=jnp.bfloat16` を設定する一方、`param_dtype` とその他のすべての層（埋め込み、正規化、および重み共有された出力射影）を fp32 のまま残す。

```
xla16 = d2l.GPT(len(data.vocab), num_hiddens=512,
                rngs=nnx.Rngs(d2l.get_key()))
nnx.update(xla16, nnx.state(model, nnx.Param))  # The same weights
d2l.to_bf16(xla16)
X, _ = next(windows(corpus, 64))
high = model(X)
error = float(jnp.linalg.norm(xla16(X) - high) / jnp.linalg.norm(high))
print(f'relative difference of the bf16 logits: {error:.1e}')
assert error < 0.01
step = nnx.jit_partial(train_step, xla16,
                       nnx.Optimizer(xla16, adamw, wrt=nnx.Param), graph=False)
record('+ bf16', tokens_per_second(step, windows(corpus, 64)))
```

```
relative difference of the bf16 logits: 2.0e-03
+ bf16: 499,090 tokens/s (1.45x), windows within 0.4%
```

bf16 のロジットは fp32 と 1% 以内で一致する。

スループットは 1.45× に向上したが、1.48× という予測値（65% を占める行列積の時間だけを半減させた場合）をわずかに下回っており、行列積以外のカーネルが高速化されなかったことがわかる。原因の 1 つはアテンションである。デフォルトでは、`jax.nn.dot_product_attention` はアテンション行列全体をデバイスメモリ上に実体化する XLA の実装を使用する（10.5 節）。活性化値が 16 ビット精度になったため、アテンションスコアをオンチップに保持する cuDNN の融合 FlashAttention カーネル（`implementation='cudnn'`）に切り替えられる。

```
class FusedGPT(d2l.GPT):
    """d2l.GPT with cuDNN's fused attention."""
    class CausalAttention(d2l.GPT.CausalAttention):
        @staticmethod
        def attend(q, k, v):  # cuDNN takes 16-bit inputs only
            return jax.nn.dot_product_attention(q, k, v, is_causal=True,
                                                implementation='cudnn')
```

同じパラメータを bf16 に変換して `model16` を `FusedGPT` からインスタンス化し、そのロジットが fp32 の参照値 `high` と 1% 以内で一致することを確認したうえで、訓練ステップをベンチマークする。

```
model16 = FusedGPT(len(data.vocab), num_hiddens=512,
                   rngs=nnx.Rngs(d2l.get_key()))
nnx.update(model16, nnx.state(model, nnx.Param))  # The same weights
d2l.to_bf16(model16)
error = float(jnp.linalg.norm(model16(X) - high) / jnp.linalg.norm(high))
print(f'relative difference of the bf16 logits: {error:.1e}')
assert error < 0.01
step = nnx.jit_partial(train_step, model16,
                       nnx.Optimizer(model16, adamw, wrt=nnx.Param),
                       graph=False)
record('+ fused attention', tokens_per_second(step, windows(corpus, 64)))
```

```
relative difference of the bf16 logits: 1.9e-03
+ fused attention: 675,436 tokens/s (1.35x), windows within 0.2%
```

cuDNN の融合アテンションに切り替えると、ロジットの 1% 以内の一致を保ったままスループットがさらに 3 分の 1 向上し、bf16 と融合アテンションを合わせた高速化率は 2× 近くに達する。

13.8.2.4 ミニバッチの拡大

ミニバッチサイズを大きくすると、ステップあたりの固定コストをより多くのトークンに償却できる。デバイスはすでにステップ全体を通して稼働しているため、償却できるのはトークン数ではなくパラメータ数に比例するデバイス処理のみである。

- AdamW は重み、勾配、2 つのモーメントを読み込み、重みと 2 つのモーメントを書き込む（fp32 で 28 bytes/param）。
- 混合精度では、順伝播で fp32 のマスター重みを bf16 にキャストし、逆伝播で bf16 の重み勾配を fp32 にキャストし直す（12 bytes/param）。

合計すると転送量は 40 bytes/param、すなわち 0.76 GB/step となる。メモリ帯域幅 1.0 TB/s では、この固定処理に約 0.75 ms かかり、これは $1/16$ 程度（12–13 ms かかるバッチサイズ 64 の bf16 ステップに対して）に相当する。ミニバッチを 8× に増やして 64 から 512 ウィンドウ（65,536 トークン）にすると、この固定コストの $7/8$ が償却され、約 5% のスループット向上が予測される。

単一 GPU での改善幅が小さくても、これには十分な価値がある。データ並列（data parallelism）がよくスケールするのは、GPU あたりのステップ時間が勾配のオールリデュースより十分に長い場合に限られるからである（13.8.3 節）。ただし、バッチサイズが臨界バッチサイズ（critical batch size）を超えるとサンプル効率が低下する点には注意が必要である（9.8 節、演習問題 5）。活性化メモリ（activation memory）はバッチサイズに比例して増加するため、重みとオプティマイザ状態を超えるピークメモリ使用量もあわせて記録する（13.4.1.4 節）。

```
batches = windows(corpus, 512)
plan = step.lower(*next(batches)).compile().memory_analysis()
record('+ batch 512', tokens_per_second(step, batches))
memory = plan.temp_size_in_bytes / 2**30
print(f'memory beyond weights and optimizer state: {memory:.1f} GiB')
```

```
+ batch 512: 718,714 tokens/s (1.06x), windows within 0.0%
memory beyond weights and optimizer state: 7.4 GiB
```

バッチサイズ 512 では、活性化メモリは GPU の 24 GiB のうち約 3 分の 1 を使用し、スループットは数パーセント向上する（演習問題 2 では、複数のバッチサイズにわたってステップあたりの固定コストとトークンあたりのコストを分離する）。3 回目のプロファイルは、バッチサイズ 512 における内訳を示す。

```
profile(step, windows(corpus, 512))
```

```
wall clock 91.19 ms per step, device busy 90.91 ms (100%), matrix products 59%
 14.54 ms  16%  ampere_bf16_s1688gemm_bf16_64x128_sliced1x2_ldg8_f2f
 11.90 ms  13%  ampere_bf16_s1688gemm_bf16_128x128_ldg8_f2f_nn
  8.36 ms   9%  ampere_bf16_s1688gemm_bf16_128x128_ldg8_f2f_tn
  6.72 ms   7%  ampere_bf16_s1688gemm_bf16_128x128_ldg8_f2f_stages_3
```

デバイスはステップ全体を通して稼働している。行列積（融合アテンションを含む）が時間の 60% 弱を占め、RMSNorm、SwiGLU、回転位置埋め込み、および fp32 と bf16 の間のキャストを行う帯域幅律速の融合カーネルが残りを占める。

13.8.2.5 活性化チェックポイント

活性化チェックポイント（activation checkpointing、13.4.3 節）は、順伝播では各ブロックの入力のみを保存し、逆伝播でブロック内部の活性化値を再計算することでメモリを節約する。バッチサイズ 512 で使用しているのは GPU メモリの 3 分の 1 にすぎないため、ここではチェックポイント化は不要であり、ステップを遅くするだけである。この速度低下は次の 2 通りの方法で予測できる。

1. **FLOP 数による予測**：順伝播の再計算により、トークンあたりのブロック演算量は 6 から 8 FLOPs/param に増えるため、チェックポイントなしのスループットの $6/8 = 0.75$ になると予測される。
2. **順伝播の時間計測による予測**：以下では、単体の順伝播時間 t_{fwd} をバッチサイズ 512 で計測し、$1/(1 + t_{\text{fwd}}/t_{\text{step}})$ を計算する。

```
X512, _ = next(windows(corpus, 512))  # One minibatch of the last step
forward = nnx.jit_partial(lambda model, X: model(X), model16, graph=False)
t_forward = d2l.Benchmark(lambda: forward(X512)).time
```

`t_forward` をステップ全体の所要時間 `t_step`（バッチサイズ 512）と比較することで、すべてのブロックの順伝播を再実行した際に維持されるスループットの割合 `1 / (1 + t_forward / t_step)` が予測できる。

```
t_step = 512 * 128 / results['+ batch 512']  # Seconds per step
print(f'forward pass {1e3 * t_forward:.1f} ms of a {1e3 * t_step:.1f} ms '
      f'step: repeating it keeps at least {1 / (1 + t_forward / t_step):.2f} '
      f'of the throughput')
```

```
forward pass 29.4 ms of a 91.2 ms step: repeating it keeps at least 0.76 of the throughput
```

2 つの推定は一致する。順伝播はステップの約 3 分の 1 を占めるため、これを繰り返しても少なくとも 4 分の 3 のスループットが維持されるはずである（埋め込みと出力射影はチェックポイント化しないため、実際にはもう少し高くなる）。ここで各ブロックを `Checkpointed` でラップし、ロジットの一致を確認したうえで、スループット（`kept=False`）と活性化メモリの両方を計測する。

`Checkpointed` は `nnx.remat` を呼び出す。

```
class Checkpointed(nnx.Module):
    """A module that keeps only its input for the backward pass."""
    def __init__(self, module):
        self.module = module

    def __call__(self, X, **kwargs):
        return nnx.remat(lambda module, X, kwargs: module(X, **kwargs))(
            self.module, X, kwargs)

remat = FusedGPT(len(data.vocab), num_hiddens=512,
                 rngs=nnx.Rngs(d2l.get_key()))
nnx.update(remat, nnx.state(model16, nnx.Param))
remat.blks = nnx.List(Checkpointed(blk) for blk in remat.blks)
d2l.to_bf16(remat)
assert jnp.allclose(remat(X), model16(X), atol=1e-6)
optimizer = nnx.Optimizer(remat, adamw, wrt=nnx.Param)
```

```
step = nnx.jit_partial(train_step, remat, optimizer, graph=False)
batches = windows(corpus, 512)
plan = step.lower(*next(batches)).compile().memory_analysis()
record('+ checkpointing', tokens_per_second(step, batches), kept=False)
memory = plan.temp_size_in_bytes / 2**30
print(f'memory beyond weights and optimizer state: {memory:.1f} GiB')
```

```
+ checkpointing: 565,754 tokens/s (0.79x), windows within 0.0%
memory beyond weights and optimizer state: 2.6 GiB
```

チェックポイント化により、活性化メモリは約 3 分の 2 削減される一方、単体の順伝播時間から予測された値よりも高いスループットが維持される。

XLA も同様に再計算される要素ごとの処理を逆伝播に融合するため、スループットは $0.79\times$ への低下にとどまり、下限の $0.76\times$ を上回る（演習問題 3）。

モデルやバッチサイズがデバイスメモリを超える場合にはチェックポイント化が不可欠であるが、ここではメモリに十分余裕があるため、以降の構成には含めない。

13.8.3 データ並列

最速化した単一 GPU のステップでも、マシンに搭載された 4 基の RTX 4090 のうち 3 基はアイドル状態のままである。複数 GPU で実行する前に、13.5.4 節のコストモデルを使ってデータ並列による高速化率を予測する。

13.8.3.1 高速化率の予測

データ並列（13.5 節）では、k 基の GPU のそれぞれがモデルのレプリカを保持し、自身のミニバッチで勾配を計算して、サイズ S バイトのオールリデュースを c 個のバケットに分けて実行することで勾配を合算する。そのステップ時間は次のとおりである。

$$t_{\text{compute}}(B/k) + c\,\alpha + \frac{2(k-1)}{k}\,S\,\beta_{\text{link}}.$$

ピアツーピアアクセスのないこの PCIe マシンでは、GPU 間の転送はホストメモリを経由して約 2.4 GB/s のバス帯域幅で行われ（13.6 節）、$\beta_{\text{link}} \approx 0.42$ ns/byte となる。弱スケーリング（weak scaling、GPU あたりのバッチサイズを固定）のもとでは、バケットあたりのレイテンシ $c\,\alpha$ は 1 ミリ秒未満であり、数十ミリ秒のステップ時間に比べれば無視できるため、オーバーラップなしのステップ時間は次のように簡略化される。

$$t_{\text{step}}(k) \approx t_{\text{step}}(1) + t_{\text{allreduce}}(k), \tag{13.15}$$

これより、スループットの倍率は $k\,t_{\text{step}}(1)/(t_{\text{step}}(1) + t_{\text{allreduce}}(k))$ となる。2 基目の GPU を追加してスループットが向上するのは、$t_{\text{allreduce}}(2) < t_{\text{step}}(1)$ の場合に限られる。

この 19M パラメータの GPT では、fp32 の勾配が占めるサイズは $S = 76$ MB であり、GPU 2 基では $t_{\text{allreduce}}(2) \approx S\beta_{\text{link}} \approx 31$ ms、GPU 4 基では $t_{\text{allreduce}}(4) \approx 1.5\,S\beta_{\text{link}} \approx 47$ ms となる。

- **GPU あたりバッチ 64 のコンパイル済み fp32**（$t_{\text{step}}(1) \approx 22$ ms）：オールリデュースにステップ自体より長い時間がかかる。Equation 13.15 からは、GPU 2 基では**減速**（$\sim 0.85\times$）し、GPU 4 基でも $\sim 1.3\times$ にとどまると予測される。

- **GPU あたりバッチ 512 のコンパイル済み bf16**（$t_{\text{step}}(1) \approx 90$–$100$ ms）：ステップがオールリデュースよりはるかに長いため、$\sim 1.5\times$（GPU 2 基）および 2.6–$2.7\times$（GPU 4 基）の高速化が予測される。

フレームワークは勾配オールリデュースのバケットを逆伝播とオーバーラップさせるため（Figure 13.20）、通常 Equation 13.15 は高速化率の下限を与え、それを上回るマージンが逆伝播の計算の裏にどれだけ通信を隠蔽できたかを表す。ただし、これは絶対的な下限として保証されているわけではない。NCCL のコピー用カーネルが同じ GPU をめぐって逆伝播と競合し（13.6.2 節）、この競合がマージンを食い潰すことがあるからである。

13.8.3.2 高速化率の実測

`data_parallel(k, bf16, batch_size)` は、k 基の GPU における訓練スループットとあわせて、同じ GPU 群で 76 MB の勾配バッファに対する単体のオールリデュース時間を計測する。

JAX は単一プロセスから k 基すべての GPU を駆動し（13.6 節）、モデル・オプティマイザ・コーパスを `Auto` メッシュ上に複製する一方、サイズ $k \times$ batch_size の各ミニバッチを `'data'` 軸に沿ってシャーディングする。XLA は勾配のオールリデュースを自動挿入し、最初のセルで有効化したレイテンシ隠蔽スケジューラによって逆伝播とオーバーラップさせる。

```
def data_parallel(k, bf16, batch_size):
    """Tokens per second on k GPUs, and seconds per allreduce of all
    gradients."""
    mesh = jax.make_mesh((k,), ('data',), devices=jax.devices()[:k],
                         # Auto: the compiler places intermediate results
                         axis_types=(jax.sharding.AxisType.Auto,))
    copy, split = NamedSharding(mesh, P()), NamedSharding(mesh, P('data'))
    net = (FusedGPT if bf16 else d2l.GPT)(len(data.vocab), num_hiddens=512,
                                          rngs=nnx.Rngs(d2l.get_key()))
    net = d2l.to_bf16(net) if bf16 else net
    optimizer = nnx.Optimizer(net, adamw, wrt=nnx.Param)
    nnx.update((net, optimizer),
               jax.device_put(nnx.state((net, optimizer)), copy))
    step = nnx.jit_partial(train_step, net, optimizer, graph=False)
    batches = windows(jax.device_put(corpus, copy), k * batch_size,
                      sharding=split)
    speed, _ = tokens_per_second(step, batches)
    grads = jnp.zeros((k, num_params), device=split)  # All gradients per GPU
    psum = jax.jit(jax.shard_map(lambda g: jax.lax.psum(g, 'data'),
                                 mesh=mesh, in_specs=P('data'),
                                 out_specs=P('data')))
    return speed, d2l.Benchmark(lambda: psum(grads), repeats=10).time
```

ここで `data_parallel` を、短い fp32 ステップ（`batch_size=64`）と長い bf16 ステップ（`batch_size=512`）の両方について $k \in \{1, 2, 4\}$ 基の GPU にわたって実行し、オーバーラップなしの予測値 Equation 13.15 と実測された高速化率を比較する。

```
for bf16, batch_size in ((False, 64), (True, 512)):
    tokens = batch_size * 128  # Per GPU and step
    one, _ = data_parallel(1, bf16, batch_size)
    print(f'{"bf16" if bf16 else "fp32"}, batch {batch_size} per GPU: '
          f'{one:,.0f} tokens/s on one GPU')
```

```
    for k in (2, 4):
        if k <= d2l.num_gpus():
            speed, allreduce = data_parallel(k, bf16, batch_size)
            predicted = k * tokens / (tokens / one + allreduce)
            print(f'  {k} GPUs: allreduce {1e3 * allreduce:3.0f} ms, '
                  f'speedup predicted {predicted / one:.2f}x, '
                  f'measured {speed / one:.2f}x')
```

```
fp32, batch 64 per GPU: 343,542 tokens/s on one GPU
  2 GPUs: allreduce  31 ms, speedup predicted 0.87x, measured 0.89x
  4 GPUs: allreduce  48 ms, speedup predicted 1.33x, measured 1.43x
bf16, batch 512 per GPU: 718,085 tokens/s on one GPU
  2 GPUs: allreduce  31 ms, speedup predicted 1.50x, measured 1.71x
  4 GPUs: allreduce  48 ms, speedup predicted 2.62x, measured 3.22x
```

単体のオールリデュースは GPU 2 基で 31 ms、GPU 4 基で 48 ms かかり、リンク速度からの見積もりと 1 ミリ秒以内で一致する。逆方向の 2 つの予測はいずれも的中した。バッチ 64 では、GPU 2 基は GPU 1 基より遅くなり、GPU 4 基でも高速化率は 1.5× 未満にとどまる。一方、bf16 のバッチ 512 では、GPU 4 基で単一 GPU の約 3× のスループット（本節の単一の変更として最大の高速化）に達し、通信のオーバーラップによって実測高速化率がオーバーラップなしの下限 Equation 13.15 を上回っている。（なお、片方向あたり 900 GB/s を持つ NVLink マシンであれば、76 MB のオールリデュースは 1 ミリ秒未満で完了し、バッチ 64 でもほぼ線形にスケールする。）

13.8.4 結果

ここで単一 GPU のすべての最適化を並べて総括し、最終的なステップでもモデルが適切に訓練されることを検証したうえで、残りの実行時間の内訳を整理する。

13.8.4.1 累積比較

下の表に単一 GPU の各構成を順に示す。各 `step` の比率は直上の行に対する相対値、`total` はベースラインに対する累積高速化率であり、チェックポイント化は採用しなかった分岐として区切り線の下に示す。`TFLOP/s` は 1 パラメータ・1 トークンあたり 6 FLOPs としてスループットを演算速度に換算した値であり、`MFU` はさらに RTX 4090 の bf16 ピーク性能である 165.2 TFLOP/s で割った**モデル FLOPs 利用率**（model FLOPs utilization）である Chowdhery et al. (2022)（演習問題 1 ではアテンションスコアの FLOPs も加える）。

```
flops = 6 * num_params  # Per token, without the attention scores
peak = {'tf32': 82.6e12, 'bf16': 165.2e12}  # FLOP/s of the RTX 4090
print(f'{"configuration":18}{"tokens/s":>10}{"step":>7}{"total":>7}'
      f'{"TFLOP/s":>9}{"MFU":>6}')
first = previous = next(iter(results.values()))
for name, speed in results.items():
    print(f'{name:18}{speed:10,.0f}{speed / previous:6.2f}x'
          f'{speed / first:6.2f}x{speed * flops / 1e12:9.1f}'
          f'{speed * flops / peak["bf16"]:6.0%}')
    previous = speed
print('-' * 57)
for name, speed in branches.items():  # Measured on top of the last row
    print(f'{name:18}{speed:10,.0f}{speed / previous:6.2f}x{"":7}'
```

```
            f'{speed * flops / 1e12:9.1f}{speed * flops / peak["bf16"]:6.0%}'
            '  (not kept)')
```

```
configuration       tokens/s   step  total  TFLOP/s   MFU
baseline             340,062  1.00x  1.00x     38.8   23%
+ jit_partial        343,869  1.01x  1.01x     39.2   24%
+ bf16               499,090  1.45x  1.47x     56.9   34%
+ fused attention    675,436  1.35x  1.99x     77.0   47%
+ batch 512          718,714  1.06x  2.11x     82.0   50%
-------------------------------------------------------------
+ checkpointing      565,754  0.79x             64.5   39%  (not kept)
```

単一 GPU の最適化を累積すると訓練スループットは 2 倍になり、RTX 4090 の bf16 ピーク性能の約 50% に達する。改善のほぼすべては行列積の高速化（bf16）とその前後のメモリトラフィックの削減（カーネル融合）によるものであり、ホストオーバーヘッドの除去とミニバッチの拡大による寄与はプロファイルの予測どおりそれぞれ数パーセントにとどまる。

なお、各高速化率は適用順序における位置（演習問題 7）とハードウェアのバランスに依存する。より高速な GPU（あるいはより小さなモデル）ではデバイス実行時間が短くなるため、ホストディスパッチの除去がはるかに大きな効果を持つようになる。

13.8.4.2 訓練の検証

どれほどスループットが高くても、最適化によって訓練が密かに壊れてしまっては意味がない。そこで正当性を検証するため、同一に初期化した 2 つのモデル（一方はベースラインのステップ、もう一方は最終的なコンパイル済み bf16 ステップを使用）を、512 ウィンドウからなる同じ 200 個のミニバッチ（〜 13M トークン）で訓練する。その際、AdamW における後半の損失スパイクを防ぐため、50 ステップの線形学習率ウォームアップを用いる（11.2 節）。両モデルが 0.15 nats/byte 以内で最後の 20 ステップの平均損失および検証損失（`d2l.lm_perplexity`、9.3 節）において一致すること、さらに両者とも 2.45 nats/byte という 11.2 節のバイグラムのベースラインを上回ることを確認する。

```
learner = d2l.to_bf16(FusedGPT(len(data.vocab), num_hiddens=512,
                               rngs=nnx.Rngs(d2l.get_key())))
reference = d2l.GPT(len(data.vocab), num_hiddens=512,
                    rngs=nnx.Rngs(d2l.get_key()))
nnx.update(reference, nnx.state(learner, nnx.Param))  # One initialization
# The learning rate rises linearly to 1e-3 over the first 50 steps
warm = optax.adamw(optax.linear_schedule(1e-3 / 50, 1e-3, 49),
                   weight_decay=0.01)
reference_optimizer = nnx.Optimizer(reference, warm, wrt=nnx.Param)
steps = {'baseline': lambda X, Y: jit_step(reference, reference_optimizer,
                                           X, Y),
         'final': nnx.jit_partial(train_step, learner, nnx.Optimizer(
             learner, warm, wrt=nnx.Param), graph=False)}
batches = windows(corpus, 512)
minibatches = [next(batches) for _ in range(200)]  # The same for both
losses = {name: jnp.stack([step(X, Y) for X, Y in minibatches]).tolist()
          for name, step in steps.items()}
validation = {name: math.log(d2l.lm_perplexity(net, data.val_dataloader()))
              for name, net in (('baseline', reference), ('final', learner))}
```

ステップ 1、50、100、200 における訓練損失の推移を、最後の 20 ステップの平均（late）および検証データの対数パープレキシティ（validation）と並べて比較し、両方の実行がバイグラムのしきい値である 2.45 nats/byte を下回り、かつ 0.15 nats/byte 以内で一致することを検証する。

```
checkpoints = (1, 50, 100, 200)
late = {name: np.mean(curve[-20:]) for name, curve in losses.items()}
print(f'{"loss at step":14}' + ''.join(f'{i:>7}' for i in checkpoints)
      + f'{"mean of last 20":>17}{"validation":>12}')
for name, curve in losses.items():
    print(f'{name:14}' + ''.join(f'{curve[i - 1]:7.2f}' for i in checkpoints)
          + f'{late[name]:17.2f}{validation[name]:12.2f}')
    assert validation[name] < 2.45  # Better than the bigram model
for values in (late, validation):  # The final step trains like the baseline
    assert abs(values['baseline'] - values['final']) < 0.15
```

```
loss at step        1     50    100    200  mean of last 20  validation
baseline         5.59   2.60   2.16   1.64             1.68        1.62
final            5.59   2.60   2.16   1.64             1.69        1.62
```

両曲線とも $\log 256 \approx 5.55$ から始まり、全 200 ステップにわたって互いによく追随し、バイグラムのしきい値を大きく下回るほぼ同一の検証損失に到達する。ただし、バッチサイズ 512 での 200 ステップ（13M トークン）の最終損失は、コサイン減衰を用いたバッチサイズ 64 での 1,000 ステップ（8M トークン、11.2 節）よりも高い点に注意したい。生のトークンスループットは、目標損失への到達時間と同一ではない（演習問題 5）。

13.8.4.3 最終ステップの時間内訳

最後に、account によってベースラインと最終構成のプロファイルを 8,192 トークンあたりのミリ秒数に正規化し、総実経過時間を理想的なピーク演算時間、実際の行列積カーネル時間、および行列積以外のカーネル時間と比較する。

```
def account(kernels, tokens, speed):
    """Milliseconds per 8,192 tokens of a step of `tokens` tokens that runs
    at `speed` tokens/s: the step, its matrix products, and all its other
    kernels."""
    per_step = 8192 / tokens  # Steps per 8,192 tokens
    products = per_step * sum(  # Kernel names as in d2l.show_profile
        ms for name, ms in kernels.items()
        if re.search(d2l.PRODUCT_KERNELS, name, re.I))
    other = per_step * sum(kernels.values()) - products
    return 1e3 * 8192 / speed, products, other

print(f'{"ms per 8,192 tokens":24}{"step":>6}{"at peak":>9}'
      f'{"products":>10}{"other":>7}')
for name, tokens, fmt in (('baseline', 64 * 128, 'tf32'),
                          ('+ batch 512', 512 * 128, 'bf16')):
    wall, products, other = account(profiles[name], tokens, results[name])
    arithmetic = 1e3 * 8192 * flops / peak[fmt]  # At the peak of the format
    print(f'{name + " (" + fmt + ")":24}{wall:6.1f}{arithmetic:9.1f}'
          f'{products:10.1f}{other:7.1f}')
```

```
ms per 8,192 tokens       step  at peak  products  other
baseline (tf32)           24.1     11.3      15.3    8.3
+ batch 512 (bf16)        11.4      5.7       6.7    4.7
```

8,192 トークンあたりで見ると、最終ステップは行列積に約 7 ms（80% 超の効率、bf16 の理論値 5.7 ms 比）、その合間の帯域幅律速カーネルに 5 ms を費やしている。これは縮約演算が全 FLOPs の 99% 超を実行しながら時間では約 60% を占めるという、典型的なルーフライン上の内訳である (Ivanov et al. 2021)。7 ms 対 5 ms に分かれたステップは、演算の高速化（たとえば fp8）単独でも、行列積以外のメモリトラフィックの除去単独でも半分にはできないため、さらに $2\times$ の高速化を達成するにはその両方が必要になる。

13.8.5 まとめ

カーネルプロファイルを用いれば、訓練ステップのボトルネックを分類し、候補となる各最適化を適用する前にその改善幅の上限を見積もることができる。具体的には、ホストディスパッチの削減で節約できるのは高々デバイスのアイドル時間、カーネル融合で節約できるのは高々行列積以外のデバイス時間であり、テンソルコアの数値精度スループットを倍増させたときの高速化率は $1/(1-f/2)$ から 2 の間（行列積がステップの割合 f を占める場合）となる。1 つのボトルネックを取り除くと残りの処理が占める割合が変化するため、各最適化は数値的等価性を検証しながら累積的に計測しなければならない。単一の RTX 4090 上では、19M パラメータの GPT のステップ速度は 2 倍に向上して 45〜50% MFU に達した。その改善のほぼすべては bf16 の行列積とカーネル融合によるものであり、最終的なステップはピークに近い行列縮約と帯域幅律速カーネルにおおむね 60 対 40 の割合で分かれた。最後に、データ並列のコストモデル $t_{\text{step}}(1) + t_{\text{allreduce}}(k)$ は、短い fp32 ステップにおける GPU 2 基での減速（22 ms のステップ対 31 ms の PCIe オールリデュース）と、バッチ 512 の bf16 ステップにおける GPU 4 基での約 $3\times$ の高速化（90–100 ms のステップ対 48 ms のオールリデュース）の両方を的確に予測した。

13.8.6 演習問題

1. [code] **モデル FLOPs 利用率**。13.8.1 節の GPT は、$L = 6$ 個のブロック（幅 $d = 512$、コンテキスト長 $T = 128$）を持つ。SwiGLU の丸めを除けば、各ブロックは行列積に $12d^2$ 個の重みを保持し、訓練には重み 1 個・1 トークンあたり約 6 FLOPs を要する。
   1. アテンションスコアの積 $\mathbf{QK}^\top$ と値の集約は、重みを持たない行列積である。1 トークン・1 ブロックあたりの訓練 FLOPs を数えよ（逆伝播は順伝播の 2 倍とし、マスクされた要素も含めること）。重みの FLOPs に対するその比率が $T/(6d)$ となることを示し、$T = 128$ および $T = 1{,}024$ に対してこの比率を求めよ。
   2. アテンションの FLOPs を含めて累積比較表の `TFLOP/s` 列と `MFU` 列を再計算し、あわせて各構成固有の数値形式のピーク性能（bf16 では 165 TFLOP/s、tf32 では 83 TFLOP/s）に対する割合も求めよ。
   3. 13.8.2 節の構成またはカーネルのうち、理論上のモデル FLOPs とハードウェアが実際に実行する FLOPs が異なるものを 2 つ挙げ、それぞれの差の向きを述べよ。
2. [code] **ステップの固定コスト**。バッチサイズ B ウィンドウにおける bf16 のステップ時間を $t(B) = a + bB$ とモデル化する（a はステップあたりの固定コスト、b はウィンドウあたりのコストである）。
   1. $B = 32, 64, 128, 256, 512$ でステップ時間を計測し、最小二乗法によって a と b を当てはめ、$B = 64$ のステップ時間のうち a が占める割合を報告せよ。
   2. 本文では、AdamW の更新と fp32・bf16 間の重みキャストを ~ 0.75 ms/step と見積もった。これは a のどれだけの割合を説明するか。B に依存しない他の処理を特定し、$B = 32$ におけるプロファイルと照合して確認せよ。
   3. 線形当てはめを用いて $B = 1{,}024$ および $B = 2{,}048$ におけるスループットを予測し、それらが 24 GiB の GPU メモリに収まるか確認したうえで、収まるものについて

実測せよ。

3. [code] **再計算のコスト**。13.8.2 節の活性化チェックポイントは、FLOP 数による見積もりと単体の順伝播時間による見積もりの両方を上回った。
   1. 単体の順伝播とチェックポイント化されたステップを `d2l.kernel_times` でプロファイリングせよ。チェックポイント化によって追加されたカーネルを単体の順伝播カーネルと突き合わせ、そのまま再出現するカーネルと逆伝播カーネルに融合されるカーネルはそれぞれどれか答えよ。
   2. 6 個のブロックのみの順伝播時間（埋め込みと出力層を除く）を計測し、スループットの下限を再計算せよ。
   3. 1 つおきのブロックのみをチェックポイント化する場合について、実行前に自身の計測値からスループットとメモリ使用量を予測し、その後に実測して比較せよ。
4. [code] **因果性ゲート**。欠陥のあるアテンションカーネルが因果マスクを密かに無視してしまうと仮定する。
   1. ベースラインと学習対象モデルの両方がこの欠陥カーネルを使用していた場合、訓練損失、検証ウィンドウ損失、および 2.45 nats/byte のバイグラムしきい値（13.8.4 節）がいずれもこの欠陥を検出できない理由を説明せよ。
   2. ミニバッチ内のすべての系列の最後の入力トークンに摂動を与え、それより前のすべての位置のロジットが数値誤差の範囲内で変化しないことを検証するテストを作成せよ。これを `learner` と、アテンションから因果マスクを省いたコピーの両方で実行せよ。
5. **スループット対目標到達時間**。9.8 節のモデルでは、バッチサイズ b で目標損失に到達するために必要な訓練例の数を $E(b) = E_{\min}(1 + b/b_{\text{noise}})$ と見積もる。ここで b_{noise} は勾配ノイズスケールである。
   1. バッチ 512 のバッチ 64 に対するスループット比を $\rho > 1$ とする。バッチ 512 がより短い実経過時間で目標損失に到達するための必要十分条件が

$$b_{\text{noise}} > \frac{512 - 64\rho}{\rho - 1}, \tag{13.16}$$

   であることを示し、$\rho = 1.2$、$\rho = 1.7$、および 13.8.4 節で実測された ρ に対してこのしきい値を求めよ。
   2. 13.8.2 節の各変更、弱スケーリングのデータ並列、および Muon（9.6 節）への切り替えを、それぞれ $E(b)$ を変化させるか、トークンスループットを変化させるか、あるいはその両方を変化させるかによって分類せよ。
6. [code] **幅 256**。`num_hiddens=256` とすると、`d2l.GPT` は 11.2 節で訓練したアーキテクチャと一致する。
   1. $8{,}192 \times 256$ 行列と 256×256 行列の出力射影 $\mathbf{W}_o$ について、4 バイトの tf32 と 2 バイトの bf16 における演算密度を計算し、両者を RTX 4090 のルーフライン（Figure 13.1）上に配置して、この積に対する bf16 の高速化率を予測せよ。
   2. `num_hiddens=256` でベースラインと 13.8.2 節の最初の 2 つの最適化を実行し、`profile` と `account` でプロファイリングして、実測された行列積の高速化率をルーフラインによる予測値と比較せよ。
7. [code] **変更の順序**。累積比較表の各高速化率は、それより前に適用された最適化に依存する。
   1. ベースラインのプロファイルと 13.8.2.3 節の上限式を用いて、未コンパイルのベースラインに直接 bf16 を適用した場合の高速化率を予測せよ。
   2. その高速化率を実測し、自身の予測値および累積比較表の `+ bf16` 行と比較せよ。さらに bf16 の上にコンパイル（表の 2 行目）を適用し、カーネルプロファイルを用いて両方の比率がどのように変化したかを説明せよ。

13.9 関連文献

本章の各節では、それぞれの考え方について代表的な例を 1 つずつ扱ってきた。すなわち、1 つのメモリ帯域幅を持つ 1 つのルーフライン、1 基の GPU とそのリンク、フレームワークごとに 1 つのコンパイラ、1 枚のカード上でメモリを節約する 3 つの方法、1 つの集合通信アルゴリズム（collective algorithm）、モデルの状態を GPU 間で分割する 1 つの方法、そして 1 台のマシン上で高速化した 1 つの Transformer である。オープンアクセスの 2 冊の書籍が、本章全体の格好の伴侶となる。『How to Scale Your Model』（Austin et al., 2025）は TPU と GPU におけるルーフライン、シャーディング記法、および集合通信のコストを導出しており、『The Ultra-Scale Playbook』（Hugging Face, 2025）はメモリバジェット、データ並列、ZeRO、およびマルチノード並列性を実測値とともに概説している。13.1 節の 3 つのレジームに関する手短な入門としては、Horace He のエッセイ (He 2022) が参考になる。

13.9.1 計測とモデル化

13.9.1.1 より精緻な性能モデル

13.1 節のルーフライン (Williams et al. 2009) は、上下どちらの方向にも誤差を生じる。一部のカーネルがその帯域幅ルーフを上回って動作するのは、72 MB の L2 キャッシュや共有メモリに収まるテンソルが β より速く到達するためである。キャッシュ考慮型ルーフライン（cache-aware roofline）(Ilic et al. 2014) はメモリ階層の各レベルごとに 1 本の帯域幅ルーフを引き、Yang et al. (2020) は深層学習カーネルについてこれらのレベルごとのバイト数をハードウェアカウンタから読み取っている。逆に、同時に移動中のバイト数（bytes in flight）が少なすぎるためにルーフを下回るカーネルもある（リトルの法則、13.2.3 節）。Volkov (2016) は、1 つのスレッド内の独立した命令が追加のワープの代わりになり得ること、そして従来の GPU 性能モデルがリトルの法則を誤って適用した結果、スループットを最大 1.7 倍誤って予測していたことを示している。

13.9.1.2 デバイスタイマとシステムプロファイラ

`d21.Benchmark`（13.1 節）はクロックを読み取る前にキューを空にするため、実際の訓練ループでは速度低下を招く。CUDA イベント（`torch.cuda.Event`）はキューを空にすることなく GPU ストリーム上でタイムスタンプを記録するが、その区間には GPU がホストを待機していた空白時間も含まれる。13.1.4 節のプロファイラは、1 つのプロセスについてビジー時間とアイドル時間を分離し、Perfetto 用のトレースを書き出す。NVIDIA の Nsight Systems は全プロセスにわたる OS スレッド、メモリコピー、NCCL 呼び出しを 1 つのタイムライン上に配置し、一方 Nsight Compute は単一のカーネルを両方のルーフに照らして計測する。

13.9.1.3 入力パイプライン

画像や音声で訓練する読者は、Python のオーバーヘッドよりも先にデータローダの供給不足（starvation）に直面することが多い。次のミニバッチを CPU がデコードしてデータ拡張するあいだ、GPU が待機してしまうからである。ホストの空きコア数が GPU の必要とする数より少ない場合（13.2.4.1 節）、デコード処理を GPU へ移すことができる（たとえば NVIDIA DALI を用いる）。Mohan et al. (2021) は複数の訓練ジョブにわたるデータストール（data stall）を分析し、訓練の特性を考慮したキャッシュ（training-aware cache）とハイパーパラメータスイープ間での前処理の共有によって、ストール時間の大部分を解消している。

13.9.1.4 利用率と目標損失までの時間

ケーススタディの締めくくりで行ったカーネルの内訳分析（13.8.4.3 節）は、Ivanov et al. (2021) が BERT に対して実施したものであり、行列積のあいだにある帯域幅律速のカーネルを融合する

ことでエンコーダ層を 1.3× 高速化した。ケーススタディの MFU 列は、大規模な訓練実行で報告される標準的な評価指標である (Chowdhery et al. 2022)。**ハードウェア FLOPs 利用率**（hardware FLOPs utilization、HFU）は再計算された FLOPs も数え上げるため、活性化チェックポイントのもとでは MFU を上回る (Korthikanti et al. 2022)。モデルが本来必要とする計算のみを評価対象とするため、システム間を公平に比較できるのは MFU だけである。

毎秒トークン数によるスループットは、たとえ収束により多くのステップを要する変更であっても、1 ステップを短縮するあらゆる変更を高く評価してしまう。そのため modded-nanoGPT スピードラン (Jordan and contributors 2024) では、8 基の GPU 上で GPT を一定の検証損失まで訓練するのにかかる実経過時間によって変更を順位付けしており、コンパイル済みカーネルや低精度形式を Muon オプティマイザ（9.6 節）と組み合わせている。

13.9.2 ハードウェアと数値形式

13.9.2.1 その他のアクセラレータ

ルーフライン解析は、Table 13.2 以外のアクセラレータにもそのまま適用できる。AMD の Instinct MI355X、Google の TPU7x、Intel の Gaudi 3、AWS の Trainium2 はいずれも、積層 HBM と行列演算ユニットを組み合わせている。これらの bf16 におけるリッジポイントは約 230 FLOP/byte（Trainium2）から 450 超（Gaudi 3）に及び、本章の RTX 4090 における 164 FLOP/byte の 1.4× から 3× に達するため、演算融合やビット幅の狭い形式がいっそう重要になる。

13.9.2.2 ユニファイドメモリ

重みがメモリに収まる限り、シングルストリームのデコーディングは、メモリ帯域幅を重みのバイト数で割った速度を超えて動作することはない（13.2.5 節）。Apple の M3 Ultra は、最大 512 GB のメモリを CPU と GPU のあいだで 819 GB/s の帯域幅で共有する。8 ビット表現の 70B モデル（約 70 GB）は 24 GB のカードには収まらないが、M3 Ultra 上では帯域幅の上限から毎秒約 12 トークンの生成が可能になる。このようなユニファイドメモリ搭載マシンは、計算ルーフの低さと引き換えに大容量を備えており、訓練よりもシングルストリームの推論に適している。

13.9.2.3 ブロックスケール形式の詳細

13.2.3 節のブロックスケールは、わずかなビット数のオーバーヘッドを伴う。OCP マイクロスケーリング形式 (Rouhani et al. 2023) では、32 個の値で共有される 8 ビットの 2 のべき乗スケールにより、値あたり 0.25 ビットが追加される。NVIDIA の NVFP4 は、16 個の値で共有する fp8 スケール（値あたり 0.5 ビット）に加えてテンソルごとに 1 つの fp32 スケールを持つことで、2 倍単位の範囲内にとどまるのではなく、各ブロックの最大値を 6（fp4 の範囲の上限）の近くに写像できる。いずれの方式でも、4 ビットの重みや活性化値に必要なのは多くても 4.5 ビットである。

13.9.3 コンパイラとカーネル

13.9.3.1 融合カーネルの記述

13.3 節のコンパイラは、要素ごとの演算の連鎖やリダクションを融合するが（PyTorch 2 については (Ansel et al. 2024)）、FlashAttention のようなタイル化を伴うアルゴリズム自体の書き換えまでは見つけ出せない。Triton (Tillet et al. 2019)（および JAX の Pallas）は、テンソルタイルをプログラミングの単位とする。すなわち、実装者がタイリングとオンチップバッファリングを指定し、コンパイラがスレッド、ベクトル命令、共有メモリを管理する。FlashAttention-2 (Dao 2024) はクエリタイルにわたって並列化し、FlashAttention-3 (Shah et al. 2024) は Hopper GPU

上でデータ移動をテンソルコアの積演算および指数計算とオーバーラップさせるとともに、fp8サポートを追加している。

13.9.3.2 大規模環境でのコンパイル時間

小さなステップのコンパイル（13.3.3 節）は数秒で済むが、大規模なモデルのコンパイルにはプロセスおよび形状ごとに数分かかることがある。領域コンパイル（regional compilation）は、繰り返されるブロック（Transformer 層など）をすべてのコピーに対して 1 回だけコンパイルし（`torch.compiler.nested_compile_region`）、永続キャッシュはコンパイル済みアーティファクトをプロセス間で共有する（`torch.compiler.save_cache_artifacts`、`jax_compilation_cache_dir`）。オートチューニング（PyTorch の `mode='max-autotune'`、XLA では標準搭載）は、コンパイル時に複数の行列積タイリングをベンチマークし、初回のコンパイル時間と引き換えに各ステップを高速化する（13.3.3 節）。

13.9.3.3 訓練におけるカーネルリプレイ

記録された CUDA グラフは、形状とメモリアドレスを固定する（13.3.4 節）。`mode='reduce-overhead'` の背後にある CUDA グラフツリーは、共有メモリプール内で形状ごとに 1 回ずつ順伝播と逆伝播を記録する。ステップをまたいで保持されるテンソル（ログに記録した損失など）は、クローンしておかない限り次のリプレイ時に上書きされる。また、ランタイムがステップ境界を自動検出できない場合には、`torch.compiler.cudagraph_mark_step_begin` で明示的に印を付ける。

13.9.3.4 エクスポートと初期のグラフ設計

キャプチャされたグラフは Python の外部でも実行できる。`torch.export` はグラフブレイクを一切許さずに厳密な全体グラフをキャプチャし、JIT コンパイルされた JAX 関数はモバイル、Web、およびサービング用ランタイムへのデプロイに向けて StableHLO へと変換（lowering）される。Theano (Bergstra et al. 2010) や TensorFlow 1 (Abadi et al. 2016) といった初期のフレームワークでは、ユーザーが静的なシンボリックグラフを明示的に構築する必要があったが、トレーシングコンパイラは通常の即時実行（eager）コードから同じプログラム全体のグラフを復元する (Frostig et al. 2018)。

13.9.4 メモリ

13.9.4.1 より小さなオプティマイザ状態

13.4.1 節におけるパラメータあたり 16 バイトのうち、Adam の 2 つのモーメント推定値は 8 バイトを占める。これらを GPU 間でシャーディングする（13.7 節）代わりに、8-bit Adam (Dettmers et al. 2022) は両方のモーメントをブロックごとに量子化して正解率を損なうことなくパラメータあたり 2 バイトに抑える。一方、Adafactor (Shazeer and Stern 2018) は一次モーメントを省略し、二次モーメントを行和と列和へと分解（factorize）する。

13.9.4.2 ホストメモリとディスクへのオフロード

ホスト DRAM は GPU メモリよりも容量が 1 桁大きく、かつ安価である。ZeRO-Offload (Ren et al. 2021) は、fp32 のマスター重みと Adam のモーメントをホストメモリに保持してオプティマイザの更新を CPU 上で実行し、GPU 上には 16 ビットの重みと勾配のみを残す。さらに ZeRO-Infinity (Rajbhandari et al. 2021) は、オフロード先を NVMe ドライブまで拡張する。

13.9.4.3 再計算対象の選択

ブロックの入力のみを保存する手法（13.4.3 節、(Chen et al. 2016)）は、トレードオフ曲線上の 1 点にすぎない。二項チェックポイント法（binomial checkpointing）(Griewank and Walther 2008) は連鎖構造において保存する状態数を固定したもとで再計算量を最小化し、Checkmate (Jain et al. 2020) は一般の計算グラフを整数線形計画法によって解く。現代のコンパイラもまた、選択的再計算を自動的に適用するか（PyTorch における AOTAutograd の最小カットパーティショナ）、明示的なポリシー（JAX の `jax.checkpoint_policies`）を通じて適用する。

13.9.4.4 断片化と事前割り当て

ステップごとにテンソルの形状が変化すると、PyTorch のキャッシングアロケータが空きメモリを非連続な断片へと分割してしまうことがある（予約済みだが未割り当てを意味する「reserved but unallocated」）。`PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True` を設定すると、セグメントをインプレースで拡張できるようになる。JAX では、デフォルトのアロケータが起動時に GPU メモリの 75% を事前割り当てする（`XLA_PYTHON_CLIENT_PREALLOCATE=false` で事前割り当てを無効化でき、`XLA_PYTHON_CLIENT_ALLOCATOR=vmm` で CUDA の仮想メモリ管理を使用する）。

13.9.5 通信とスケール

13.9.5.1 リングを超える集合通信アルゴリズム

リング allreduce（13.5.3 節）は移動する総バイト数を最小化するが、$2(k-1)$ 回の逐次ホップを要する。ツリーアルゴリズムはホップ数を $\log_2 k$ のオーダーに抑え、NCCL は相補的な 2 つの二分木を組み合わせることで、すべてのデバイスが一方の木では葉となり、もう一方の木では内部ノードとなるようにしている。k が 2 のべき乗のとき、再帰的半減・倍増法（recursive halving and doubling）はリングと同じバイト通信量を $2\log_2 k$ ステップで達成する (Thakur et al. 2005)。

13.9.5.2 パラメータサーバーと非同期更新

集合通信ライブラリが標準となる以前、分散訓練ではパラメータサーバー上で勾配を集約し (Dean et al. 2012; Li et al. 2014)、非同期更新やローカル SGD を支えていた (Stich 2019)。古い勾配（stale gradient）は最適化のダイナミクスを変化させてしまうため、現在では密なモデルに対して同期型の allreduce や ZeRO（(S. Li et al. 2020; Rajbhandari et al. 2020)）が標準となっている一方、巨大な疎の推薦テーブル（第 42 章（英語版））では依然としてパラメータサーバーが広く使われている。

13.9.5.3 並列化インターフェースと単一マシンを超える拡張

PyTorch は各並列化戦略をデバイスメッシュ上の組み合わせ可能な API（`DistributedDataParallel`、`fully_shard`、`torch.distributed.tensor.parallel`、`torch.distributed.pipelining`）として公開しているのに対し、JAX は GSPMD / Shardy (Xu et al. 2021) でコンパイルされる名前付きメッシュ上の配列配置によってそれらすべてを宣言的に表現する。マルチノードのテンソル並列・パイプライン並列・エキスパート並列 (Shoeybi et al. 2019; Narayanan et al. 2021; Lepikhin et al. 2021)（第 23 章（英語版））、fp8 訓練 (Micikevicius et al. 2022; DeepSeek-AI, Liu, et al. 2024)（21.3 節（英語版））、連続バッチングと PagedAttention (Yu et al. 2022; Kwon et al. 2023)（第 30 章（英語版））、および重みの量子化（第 28 章（英語版））については、「言語モデル」パートで詳しく扱う。

Chapter 14

強化学習

これまで本書で扱ってきたモデルはすべて、静止したデータセット上で訓練されていた。モデルが何を予測しても、次にどの例が現れるかは一切変わらなかったのである。ここで、数学の問題に取り組む言語モデルを考えてみよう。モデルは 1,000 トークンにわたる推論を書き連ねるが、答えが正しいかどうかを判定器が告げるのは最後だけである。どのトークンにも個別のラベルはなく、1 トークン出力するごとにモデルが次に書ける内容が変化する。動く台車の上で棒のバランスを取る制御器も同じ立場にある。台車を押すたびに次に補正すべき角度が変わり、得られる報酬（reward）は棒が倒れずに立っている時間だけである。Figure 14.1 は、意思決定主体である**エージェント**と、それが働きかける世界である**環境**（environment）の間のこのループを描いている。

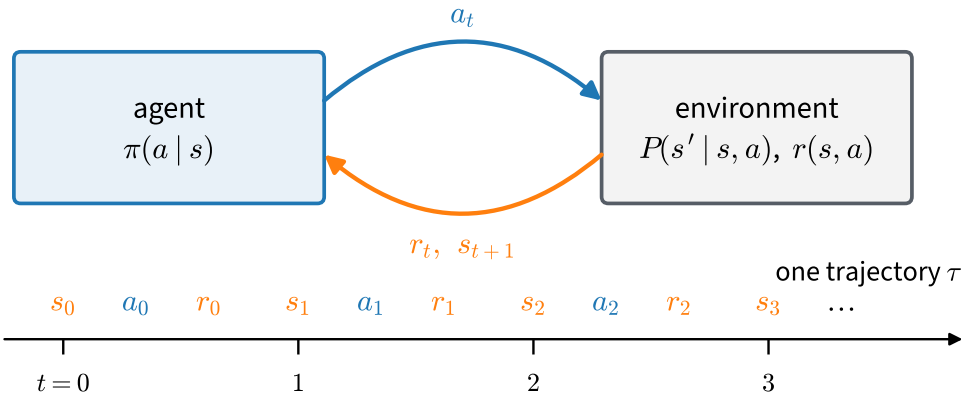


Figure 14.1: エージェントと環境の間のループ。ステップ t において、エージェントは方策 π に従って行動 a_t（青）を選択する。遷移確率 P と報酬 r に規定される環境は、報酬 r_t と次の状態 s_{t+1}（オレンジ）を返す。このやり取りを繰り返すことで、軌跡 $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ が生成される。

ここから 2 つの問いが生じる。遅れて届く報酬に対してどの意思決定が貢献したのか、そして何がよい意思決定かを誰にも示されないまま、学習者はどのようにしてそれを見つけ出せるのか。1 つ目の問いに対し、ルールが既知の場合の答えを 1950 年代に示したのが Richard Bellman である。各状態（state）にこれから得られる報酬によってスコアを付けることで、長いホライズンにわたる問題を 1 ステップの先読みの連鎖へと変換したのである (Bellman 1957b)。1980 年代

には Andrew Barto、Richard Sutton、Chris Watkins が試行錯誤によってそれらのスコアを学習し (Barto et al. 1983; Sutton 1988; Watkins 1989)、続いて Ronald Williams が意思決定の規則である**方策**（policy）を試行から得られた報酬によって直接改善する方法を示した (Williams 1992)。学習者は自らデータを収集するため、犯した失敗によって次に学ぶ対象そのものが変化する。モデルが自らの誤りから立て直す練習を一切しない場合、模倣だけでは不十分になる理由もここにある。

本章は 3 つの段階で進む。第 1 に、マルコフ決定過程（Markov decision process）を定式化し（14.1 節）、ルールが既知の場合の厳密な最適方策（optimal policy）を計算し（14.2 節）、サンプリングされた遷移から行動価値（action value）を学習し（14.3 節）、探索（exploration）のリグレットを定量化する（14.4 節）。第 2 に、長いホライズンにおいてエキスパートのデモンストレーションが見落とす問題を調べ（14.5 節）、既知の物理法則のもとで連続系の動的計画法（dynamic programming）がどのようになるかを、線形二次レギュレータ（linear-quadratic regulator、LQR）、反復 LQR（iterative LQR、iLQR）、およびリシーディングホライズン型のモデル予測制御（model predictive control、MPC）を通じて学ぶ（14.6 節）。最後に、スコア関数（score function）の勾配によって方策を直接最適化し（14.7 節）、ベースラインとグループ相対重みによって分散（variance）を抑え（14.8 節）、ルックアップテーブルをニューラルネットワークに置き換えたうえで、本章と次章の諸手法を体系的に整理する（14.9 節）。

すべての例は PyTorch と JAX で動作し、その大半はあらゆる推定値を厳密解と照らし合わせて検証できるほど小規模である。続く第 15 章ではブートストラップ型クリティック、経験再生（experience replay）、学習されたダイナミクスモデル、固定されたオフラインデータセットを扱い、「言語モデル」パートでは選好に基づく事後学習（post-training）をより深く掘り下げる。

参考資料と関連文献

以下に挙げる無料公開の資料では、理論的な解説、講義コース、実装、および実証的な指針が得られる。

教科書

- Sutton and Barto『Reinforcement Learning: An Introduction』(Sutton and Barto 2018) は、価値手法、方策手法、プランニング、関数近似を包括的に解説した入門書である。Sutton と Barto は、強化学習の基礎を築いた貢献により 2024 年のチューリング賞を受賞した。
- Szepesvári『Algorithms for Reinforcement Learning』(Szepesvári 2010) は、本章のアルゴリズムとその理論保証を 100 ページ未満でまとめている。
- Bertsekas『A Course in Reinforcement Learning』(Bertsekas 2025) は、最適制御と動的計画法の観点から強化学習を展開している。
- Agarwal、Jiang、Kakade、Sun『Reinforcement Learning: Theory and Algorithms』(Agarwal et al. 2019) は、本章で定性的に述べるサンプル計算量のレートを与えている。
- Lattimore、Szepesvári『Bandit Algorithms』(Lattimore and Szepesvári 2020) は、14.4 節の背後にある理論の全体像を扱っている。
- 14.6 節については、Kalman (1960) および Anderson and Moore (2007) が線形二次レギュレータ（LQR）を展開し、微分動的計画法 (Mayne 1966) と反復 LQR (Li and Todorov 2004) が後ろ向きのリカッチパスを非線形軌跡最適化へと拡張している。また、Rawlings et al. (2017) は制約付きのリシーディングホライズン型モデル予測制御を網羅し、Tedrake (2023) は劣駆動ロボティクスと軌跡最適化を扱っている。

講義コース

- Berkeley CS285: Deep Reinforcement Learning は、方策勾配、価値手法、モデルベース学習、オフライン強化学習を大学院レベルで扱っている。

- Stanford CS234: Reinforcement Learning は、基礎、探索、および理論保証を伴う方策勾配を扱い、対象範囲が本章に最も近い講義コースである。
- Stanford CS224R: Deep Reinforcement Learning は、模倣学習やオフライン手法を含め、実践面を扱っている。
- CMU 10-703: Deep Reinforcement Learning and Control は、アルゴリズムを制御および物理的な身体性と結びつけて扱っている。
- MIT 6.7920: Reinforcement Learning: Foundations and Methods は、14.2 節で導入する動的計画法の理論を展開している。

注釈付きの実装

- CleanRL (Huang, Rousslan F. J. Dossa, et al. 2022) は、14.9 節以降で扱う主要な深層強化学習アルゴリズムについて、ベンチマーク検証済みの単一ファイル実装を提供している。
- Stable-Baselines3 (Raffin et al. 2021) は同じアルゴリズム群のライブラリ実装を提供しており、付属の Zoo には調整済みのハイパーパラメータが用意されている。
- Gymnasium (Towers et al. 2024) は、これらの章全体で使用する環境インターフェースのドキュメントを提供している。

実証的な実践

- Spinning Up in Deep RL (Achiam 2018) は、スコア関数から SAC までの簡潔なカリキュラムであり、動作するコードと並べて読むことを想定して書かれている。
- The 37 Implementation Details of Proximal Policy Optimization (Huang, Rousslan Fernand Julien Dossa, et al. 2022) は、アルゴリズムの説明では省略されがちな実装上の選択が PPO の性能と再現性にどう影響するかを記録している。

14.1 マルコフ決定過程

凍った湖を渡るロボット、ボードゲームを対局するプログラム、あるいは 1 トークンずつ回答を書き進める言語モデルを想像してみよう。どの選択も次に起きることを変える。ロボットは別のマスへ滑り込み、ゲームは別の局面に至り、回答は別の展開をたどる。報酬は最後にしか得られないかもしれず、同じコマンドでも複数の結果が生じうる。うまく行動する方法を問う前に、まずこうした問題を書き下す枠組みが必要になる。すなわち、意思決定者が各ステップで何を知り、何ができ、その結果何が起き、何を達成しようとしているかである。

Bellman は 1957 年のマルコフ的決定過程に関する論文で、この記述に定式を与えた (Bellman 1957a)。その着想は、過去のうち将来に関わるすべてを**状態**（state）に集約し、行動の結果がその行動をとった状態のみに依存するようにすることであった。このただ 1 つの仮定によって、問題は扱いやすくなる。将来が現在の状態のみに依存するならば、状態ごとに 1 つの数値を与えるだけで、そこにいることの良さを表せる。こうして得られたモデルである**マルコフ決定過程**（Markov decision process、MDP）は、オペレーションズリサーチの標準的な道具となった (Puterman 1994)。動的計画法（dynamic programming）から報酬を用いた言語モデルの訓練に至るまで、本章と次章で扱うあらゆる手法はこのモデルの上で動作する。

本節では、すべての量を手計算で確かめられるほど小さい、16 マスの凍った湖を題材とする。それでもこの湖には、時間にわたる行動を難しくする性質が備わっている。滑りやすい氷の上では、エージェントが意図した方向とは別の場所へ滑ってしまうことが少なくないのである。まずシミュレータから湖の遷移表を読み出して各部に名前を付け、続いてエージェントが最大化する収益と、それを有界に保つ割引を定義する。さらに、状態が何を含まなければならないかを問い、一見有用に見えながら同じ場所を周回するだけでエージェントに報酬を与えてしまうボーナスを検証する。本節の締めくくりには、さらに小さな 2 つ目の例として、算数の問題に回答し判定器によってスコア付けされる言語モデルを扱う。

本章では Table 14.1 の記号を用いる。本節で使う記号は以下で定義し、残りの記号は後の各節において初出箇所でそれぞれ定義する。

Table 14.1: 強化学習の章で用いる記号。V^π 以降の行は後の各節に属し、λ 以降の行は第 15 章に属する。探索率 ϵ とステップごとの不一致率 ε は異なる記号であり、応答 y と価値ターゲット y_t も同様である。0 以外の添字または引数を持つ μ は平均（アームの平均、ガウス分布の平均など）を表す。

記号	意味	初出箇所
$\mathcal{S}, \mathcal{A}$	状態の集合と行動の集合	14.1 節
s_t, a_t, r_t	ステップ t における状態、行動、報酬	14.1 節
$P(s' \mid s, a)$	遷移カーネル：到達先が s' となる確率（状態 s、行動 a のもと）	14.1 節
$r(s, a)$	行動 a の状態 s における期待報酬	14.1 節
μ_0	初期状態分布（言語モデルではプロンプトの分布）	14.1 節
τ	軌跡 $(s_0, a_0, r_0, s_1, \ldots)$	14.1 節
γ	割引率	14.1 節
$R(\tau)$	軌跡の収益 $\sum_t \gamma^t r_t$	14.1 節
T	エピソードのステップ数、または時間制限	14.1 節
$\pi(a \mid s), \pi(s)$	方策：行動 a を状態 s でとる確率、および決定論的方策が返す行動	14.1 節, 14.2 節
$\ell(s)$	マス s からゴールまでのマンハッタン距離	14.1 節
$\Phi(s)$	シェイピングボーナスのポテンシャル	14.1 節
$x, y, r(x, y)$	プロンプト、応答、および応答に対する検証器のスコア	14.1 節
y_t	価値更新のターゲット：ステップ t からの収益の推定値	14.1 節, 14.9 節
V^π, Q^π	π のもとでの状態からの期待収益、状態行動対からの期待収益	14.2 節
V^*, Q^*	最適方策のもとでの同様の期待収益	14.2 節
$\mathcal{T}$	ベルマン作用素：価値のベクトルに適用する 1 ステップ分の先読み	14.2 節
δ, δ_t	1 回の遷移、およびステップ t の遷移における時間差分誤差	14.3 節
α	時間差分更新のステップサイズ	14.3 節
ϵ	ϵ 貪欲な行動選択における探索率	14.3 節
π_e	行動方策（手法が学習する間に行動を選択する方策）	14.3 節
κ	不確実性ボーナスの重み	14.4 節
K	プロンプトあたりにサンプリングされる応答数	14.4 節
π_θ, θ	パラメータ θ を持つ方策	14.1 節（検証器タスクの選好テーブル）、14.5 節
d_t^π, d_γ^π	s_t の π のもとでの分布、および t にわたるその割引平均	14.5 節, 14.7 節
ε	クローン方策がエキスパートと一致しないステップごとの確率	14.5 節

記号	意味	初出箇所
$\mathbf{x}_t$, $\mathbf{u}_t$, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{Q}$, $\mathbf{R}$	連続状態と連続制御、線形ダイナミクス行列と二次コスト行列	14.6 節
$\mathbf{P}_t$, $\mathbf{K}_t$	リッカチの残余コスト行列と線形状態フィードバックゲイン行列	14.6 節
$J(\theta)$	π_θ の期待収益	14.7 節
η	方策更新のステップサイズ	14.7 節
$\hat{G}_t$	残りの累積報酬：ステップ t 以降の報酬の割引和	14.8 節
b	収益から差し引くベースライン	14.8 節
η_V	価値ベースライン更新のステップサイズ	14.8 節
A^π	アドバンテージ $Q^\pi - V^\pi$	14.8 節
λ	λ 収益の重み	15.1 節
w, w^-	価値ネットワークのパラメータ、およびその固定コピーまたは Polyak 平均コピーのパラメータ	15.1 節, 15.4 節
n	n ステップターゲットまたは計画の深さ	15.1 節, 15.6 節
ρ_t	重要度比 $\pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$（ステップ t）	15.2 節
ϵ_{clip}	比のクリッピング帯の半幅	15.2 節
Δ_{KL}	1 回の更新あたりの KL バジェット	15.2 節
π_{ref}, β	参照方策、および KL ペナルティの重み・ソフトアクタークリティックの温度	15.3 節, 15.5 節
$\hat{r}$	当てはめた報酬	15.3 節, 15.6 節
ν	ターゲットネットワークの Polyak 更新率	15.5 節
$\hat{f}_\phi$	学習された遷移モデル	15.6 節

```
import gymnasium as gym
```

14.1.1 モデル

Figure 14.2 に湖を示す。これは Gymnasium スイート (Towers et al. 2024) に含まれる Frozen-Lake であり、本章の表形式を扱う各節で共通して用いる。エージェントは左上のマスから出発し、報酬 1 が得られる右下のゴールへの到達を目指す。4 つのマスは穴になっており、穴またはゴールに入るとその試行は終了する。**滑りやすい**（slippery）氷の上では、コマンドを出すと確率 $1/3$ で意図した方向へ進み、直交する 2 方向へもそれぞれ確率 $1/3$ で進む。グリッドの端から外へ出ようとする移動では、エージェントはその場にとどまる。

ゴールから 3 手の距離にあるマス 9 にいるエージェントを考えよう。4 つのコマンドのうち 3 つは、いずれも 3 回に 1 回の確率でマス 5 の穴に滑り落ちてしまう。**下**だけは穴に落ちないが、**下**も 3 回に 2 回は横へ滑ってマス 8 または 10 に移る。**下**の安全性には、横に滑って手数を失うだけの価値があるだろうか。コマンド同士を比較するには、各コマンドがどこへ至りうるか、そこからエージェントに何ができるか、そして最後に何を受け取るかを知る必要がある。

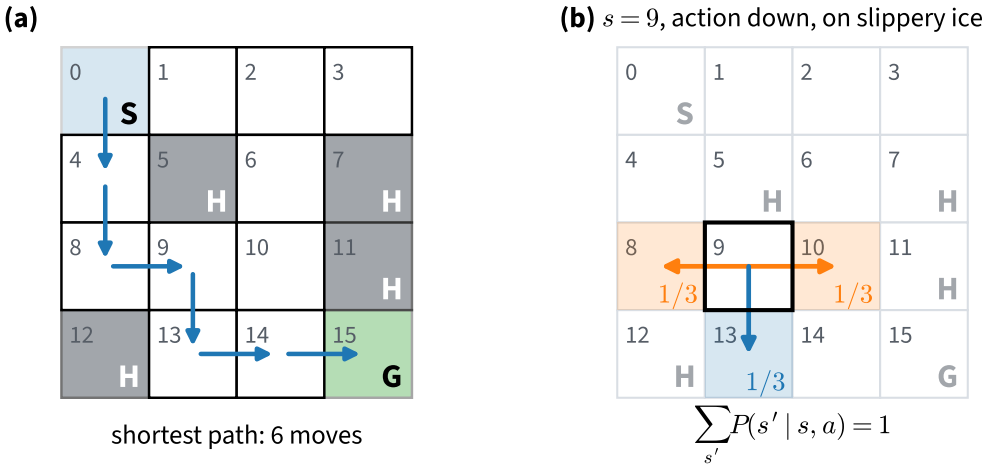


Figure 14.2: 湖。(a) S はスタート、G はゴールを表し、H と記された灰色のマスは穴である。各マスには状態のインデックスが付されている。滑らない氷の上では、最短経路は 6 回の移動でゴールに至る。(b) 滑りやすい氷の上での 1 回のコマンド：状態 $s = 9$ から行動**下**をとると、マス 13、8、10 のいずれかにそれぞれ確率 $1/3$ で到達する。意図した移動は等確率な 3 通りの結果の 1 つにすぎず、確率の和は 1 になる。

14.1.1.1 状態・行動・遷移カーネル

湖における**状態**はエージェントがいるマスである。状態の集合 $\mathcal{S} = \{0, 1, \ldots, 15\}$ は、行ごとに各マスへ番号を振る。**行動**（action）は**左**、**下**、**右**、**上**の 4 つのコマンドであり、0 から 3 として符号化され、集合 $\mathcal{A}$ をなす。滑りやすい氷の上では 1 つの行動が次の状態に関する分布を定め、Gymnasium はこの分布を直接読み出せる表として保持している。

```
env = gym.make('FrozenLake-v1', is_slippery=True)
LEFT, DOWN, RIGHT, UP = 0, 1, 2, 3
ARROWS = '<v>^'                          # one glyph per action, for printing
env.unwrapped.P[9][DOWN]
```

```
[(0.3333333333333333, 8, 0, False),
 (0.3333333333333333, 13, 0, False),
 (0.3333333333333333, 10, 0, False)]
```

出力には、（確率、次の状態、報酬、終端フラグ）という形式の結果が 3 つ並んでいる。マス 9 から**下**のコマンドを出すと、Figure 14.2 のとおり、エージェントはそれぞれ確率 $1/3$ で 8、13、10 のいずれかに到達する。記号で表せば、この表は**遷移カーネル**（transition kernel）$P(s' \mid s, a)$、すなわち状態 s' に状態 s から行動 a のもとで到達する確率である。各対 (s, a) に対して数値 $P(\cdot \mid s, a)$ は確率分布をなすため、$\sum_{s' \in \mathcal{S}} P(s' \mid s, a) = 1$ が成り立つ。

4 つ目の構成要素は**報酬**（reward）である。シミュレータはゴールに入るステップに対して 1 を与え、それ以外のすべてのステップには 0 を与えるため、1 ステップの報酬はそのステップの到達先に依存する。ここでは到達先のマスにわたって平均をとった量、すなわち**期待報酬**（expected reward）$r(s, a)$（行動 a の状態 s における平均報酬）を扱う。この平均化によって期待収益が一切変わらないことは、演習問題 3 で示す。以上 4 つの対象を合わせたものが MDP である。

$$\text{MDP}: \quad (\mathcal{S}, \mathcal{A}, P, r). \tag{14.1}$$

最初の 3 つは環境（environment）を記述する。一方、報酬はタスクを記述するものであり、タスクを設定する者がこれを選ぶ。この違いが、後の報酬設計の節の要となる。文献によっては、以下で定義する割引率をこの組に加えたり、報酬を確率変数としたり次の状態に依存させたりすることもある。その場合でも本節の $r(s,a)$ が期待報酬の役割を担うため、この簡潔な形式によって本章の議論が一般性を失うことはない。

14.1.2 収益・割引・ホライズン

エージェントは初期状態 s_0 で出発する。これは**初期状態分布**（start-state distribution）μ_0 から抽出され、湖の場合にはすべての確率質量をマス 0 に置く。各ステップ t において、エージェントは状態 s_t にあり、行動 a_t をとり、報酬 r_t を受け取って、次の状態 $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ へと遷移する。これらの量の系列を**軌跡**（trajectory）と呼び、次のように書く。

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots).$$

軌跡に現れる報酬 r_t はシミュレータから実際に与えられた実現値であり、期待報酬 $r(s_t, a_t)$ は s_t と a_t を与えたもとでのその平均である。

行動を選ぶ規則を**方策**（policy）と呼び、$\pi(a \mid s)$ で行動 a を状態 s において選ぶ確率を表す。最も単純な方策は、どの状態でも 4 つのコマンドをそれぞれ確率 $1/4$ で選ぶものである。これをマス 0 から 1 回実行し、各ステップを表示してみよう。

```
rng = np.random.default_rng(8)
s, _ = env.reset(seed=8)
terminated = truncated = False
total, t = 0.0, 0
while not (terminated or truncated):
    a = int(rng.integers(4))
    s_next, r, terminated, truncated, _ = env.step(a)
    print(f"t={t:>2}  s={s:>2}  a={ARROWS[a]}  s'={s_next:>2}  r={r:.0f}")
    total, s, t = total + r, s_next, t + 1
print(f'terminated={terminated}, truncated={truncated}, total reward={total:.0f}')
```

```
t= 0  s= 0  a=>  s'= 0  r=0
t= 1  s= 0  a=v  s'= 0  r=0
t= 2  s= 0  a=<  s'= 4  r=0
t= 3  s= 4  a=^  s'= 4  r=0
t= 4  s= 4  a=<  s'= 4  r=0
t= 5  s= 4  a=v  s'= 8  r=0
t= 6  s= 8  a=>  s'= 9  r=0
t= 7  s= 9  a=^  s'=10  r=0
t= 8  s=10  a=>  s'=11  r=0
terminated=True, truncated=False, total reward=0
```

この実行記録には、遷移カーネルの働きがそのまま現れている。$t=0$ ではエージェントは**右**のコマンドを出すが、氷の上で上側の壁に向かって滑ったためマス 0 にとどまる。$t=2$ では**左**のコマンドを出しながら、**下**へ滑ってマス 4 に移る。そして $t=8$ ではマス 10 から**右**のコマンドを出してマス 11 の穴に入り、`terminated=True` となって試行が終了する。一度もゴールに入らな

かったため、報酬はすべて 0 である。ゴールに到達した軌跡を評価するには、遅れて得られる報酬をどれだけ重く評価するかをまず決めなければならない。

14.1.2.1 割引と実効ホライズン

報酬を単純に足し合わせないのはなぜだろうか。第 1 に、軌跡は無限に長くなりうる。常に上のコマンドを出す方策は、湖の最上段の行から決して出ない。上は壁にぶつかり、横へ滑る 2 通りも同じ行に沿った移動になるからである。毎ステップ正の報酬があるなら、そのような軌跡に沿った和は際限なく増大してしまう。また一般に、いま得られる報酬は後になって得られる同じ報酬よりも価値が高い。この 2 つの理由から**割引率**（discount factor）$0 \le \gamma < 1$ を導入し、ステップ t の報酬に重み γ^t を掛ける。軌跡の**収益**（return）は、次の割引和で定義される。

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{14.2}$$

$\gamma = 0.95$ のとき、1 ステップ先の報酬の重みは 0.95 であり、20 ステップ先の報酬の重みは約 0.36 となる。目的は、カーネルからのサンプリング、初期状態、およびエージェント自身の選択にわたる平均の意味で、収益ができるだけ大きくなるように行動することである。この**期待収益**（expected return）を最大化するのは、独立な試行を数多く繰り返したときに平均して得られる量がこれにほかならないからである。$\gamma < 1$ の有限 MDP では、この問題に明確な解が存在する (Puterman 1994)。あらゆる初期状態から同時に最良となる単一の方策が存在し、しかもそれは決定論的かつ時刻に依存しないもの、すなわちマスごとに 1 つのコマンドを対応させるものとして選べる。したがって最良の方策はエージェントの出発点に依存せず、初期状態分布が 1 つのマスに集中していても何も損なわれない。

割引はまた、エージェントがどれだけ先まで見通すかも制御する。すべての報酬が $|r_t| \le r_{\max}$ を満たすならば、割引によって収益は有限に保たれ、ステップ k 以降の部分も次のように抑えられる。

$$|R(\tau)| \le \frac{r_{\max}}{1-\gamma},$$
$$\Big|\sum_{t \ge k} \gamma^t r_t\Big| \le \frac{\gamma^k\, r_{\max}}{1-\gamma}.$$

1 つ目の上界により目的関数は有限になる。2 つ目の上界は、ステップ k 以降の報酬が収益を動かしうる幅が、最大可能値のせいぜい γ^k 倍にとどまることを表している。このことから、割引が届く範囲をステップ数で測る尺度として**実効ホライズン**（effective horizon）$1/(1-\gamma)$ が導かれる。γ が 1 に近いとき $\gamma^{1/(1-\gamma)} \approx e^{-1}$ であるため、重みは 1 ホライズンあたり約 e 分の 1 に減衰する。一定の報酬が続く列に対しては、割引重み全体の 95% が $\log(0.05)/\log(\gamma)$ ステップ、すなわち約 3 ホライズン以内に収まる。次の表は、4 通りの割引率についてこれら両方のステップ数と、滑らない氷での 6 手の最短経路の最後でゴール報酬に掛かる重み γ^5 をまとめたものである。

```
print(f'{"gamma":>6} {"horizon 1/(1-gamma)":>20} {"t: gamma^t < 0.05":>18} '
      f'{"gamma^5":>8}')
for g in [0.5, 0.9, 0.95, 0.99]:
    t95 = int(np.ceil(np.log(0.05) / np.log(g)))
    print(f'{g:>6} {1 / (1 - g):>20.0f} {t95:>18} {g**5:>8.3f}')
```

gamma	horizon 1/(1-gamma)	t: gamma^t < 0.05	gamma^5
0.5	2	5	0.031
0.9	10	29	0.590
0.95	20	59	0.774
0.99	100	299	0.951

本章全体で用いる割引率 $\gamma = 0.95$ では、ホライズンは 20 ステップ、重みの 95% は 59 ステップ以内に収まり、6 手で到達するゴールの価値は $\gamma^5 \approx 0.774$ となる。一方、$\gamma = 0.5$ では同じゴールの価値は約 0.03 にすぎない。Figure 14.3 にこれらの量を示す。割引率はタスクの一部であり、設計者が選ぶものである。γ を小さくするとエージェントは近視眼的になるが、不正確なモデルを用いて計画を立てる場面ではかえって信頼性が高まる場合があることを演習問題 8 で確認する。

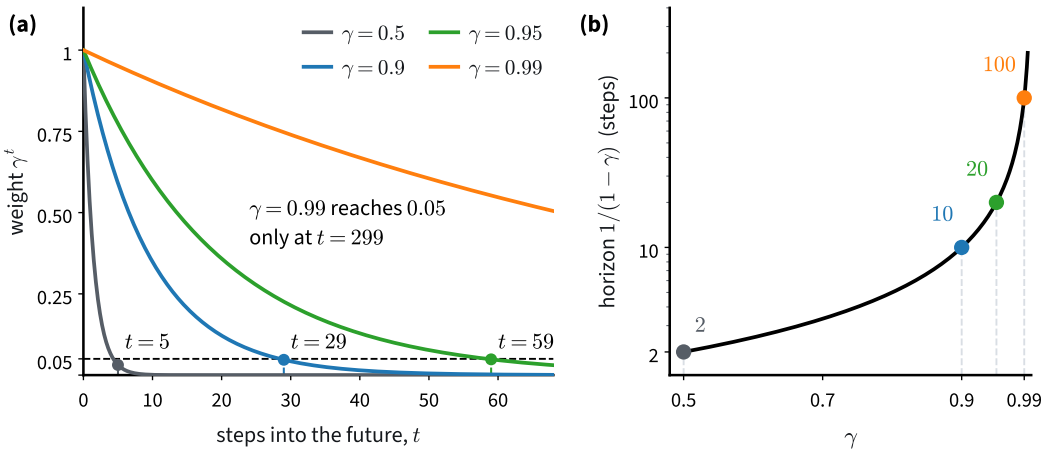


Figure 14.3: 割引は実効ホライズンを定める。(a) 重み γ^t（t ステップ先の報酬に対する重み）が 0.05 を下回るのは、$t = 5$ ($\gamma = 0.5$)、$t = 29$ ($\gamma = 0.9$)、$t = 59$ ($\gamma = 0.95$) であり、$t = 299$ ($\gamma = 0.99$) に至ってようやく下回る。(b) 対数軸で見たホライズン $1/(1-\gamma)$：$\gamma = 0.5$ で 2 ステップ、$\gamma = 0.95$ で 20 ステップ、$\gamma = 0.99$ で 100 ステップとなる。

14.1.2.2 配列としてのモデル

動的計画法はカーネル・報酬・割引率を何度も参照し、後の表形式を扱う各節でも学習器の結果をその厳密計算と照らし合わせて検証する。Gymnasium が公開しているタプルの辞書はこの用途には扱いにくい。次の状態にわたる平均を 1 回の行列積で計算できるよう、P を (s, a, s') で添字付けされた 1 つの配列として、r を (s, a) で添字付けされた 1 つの配列として持ちたい。`TabularMDP` はこれら 3 つの対象を保持するクラスである。その `from_gym` メソッドはシミュレータの表から配列を構築し、同じ次の状態に至る結果の確率を足し合わせるとともに、結果にわたって報酬を平均することで期待報酬 $r(s, a)$ を求める。本章の表形式の各節で使用するため、このクラスを `d2l` ライブラリに保存しておく。

```
class TabularMDP:
    """A finite MDP as dense arrays: P[s, a, s'] and r[s, a]."""
    def __init__(self, P, r, gamma):
        self.P, self.r, self.gamma = P, r, gamma
        self.num_states, self.num_actions = r.shape

    @classmethod
```

```
    def from_gym(cls, env, gamma):
        """Read the transition table Gymnasium exposes as env.unwrapped.P."""
        n_s, n_a = env.observation_space.n, env.action_space.n
        P, r = np.zeros((n_s, n_a, n_s)), np.zeros((n_s, n_a))
        for s, actions in env.unwrapped.P.items():
            for a, outcomes in actions.items():
                for p, s_next, reward, _ in outcomes:
                    P[s, a, s_next] += p      # several outcomes may share s'
                    r[s, a] += p * reward     # r(s,a) is the expected reward
        return cls(P, r, gamma)
```

$\gamma = 0.95$ として湖のモデルを構築し、先ほどシミュレータから読み出したコマンドに対応する行を表示してみよう。

```
gamma = 0.95
mdp = TabularMDP.from_gym(env, gamma)
print(mdp.P[9, DOWN])
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.33333333 0.         0.33333333 0.
 0.         0.33333333 0.         0.        ]
```

タプルのリストは 16 個の数値からなる密な行へと変わり、列 8、10、13 が $1/3$ で、それ以外は 0 になっている。配列ごとに 1 つずつ、計 2 つの確認で構築結果を検証しよう。P のすべての行の和が 1 であること、そしてゴールに入れる箇所でのみ報酬が非ゼロであることである。

```
print(f'largest |row sum - 1|: {np.abs(mdp.P.sum(axis=-1) - 1).max():.1e}')
for s, a in np.argwhere(mdp.r > 0):
    print(f'r(s={s}, a={ARROWS[a]}) = {mdp.r[s, a]:.3f}')
```

```
largest |row sum - 1|: 0.0e+00
r(s=14, a=v) = 0.333
r(s=14, a=>) = 0.333
r(s=14, a=^) = 0.333
```

P のどの行も和は 1 になっている。報酬配列は、ゴールに隣接する唯一の氷のマスであるマス 14 を除いてすべて 0 である。マス 14 では**下**、**右**、**上**がいずれも確率 $1/3$ でゴールに到達するため期待報酬 $1/3$ を持つ一方、**左**の 3 通りの結果はいずれもゴールにはならない。この湖では、ゴールに入るそのステップでのみ報酬が与えられるのである。

14.1.2.3 エピソード・終端・打ち切り

過程を終了させる状態を**終端**（terminal）状態と呼ぶ。湖では穴とゴールが終端状態である。出発から終端状態に至るまでの一連の実行を**エピソード**と呼ぶ。シミュレータはそこでエピソードを停止するため、リセットが必要になる。Gymnasium の表、ひいては `TabularMDP` では、P のすべての行が確率分布になるよう、各終端状態をどの行動でも報酬 0 で自分自身に戻る吸収状態（absorbing state）として格納している。吸収状態ではそれ以上何も得られないため、どちらの記述でも収益は同じになる。

すべてのエピソードが有限ステップで終わる場合、収益を有限に保つための割引は不要であり、$\gamma = 1$ が許容される。このとき収益は、T ステップからなるエピソードの報酬の単純和になる。湖では、常に**上**を選ぶ方策が最上段を永遠に回り続けるため、すべてのエピソードが終わるとは

限らないが、それでも報酬が非負で 1 本の軌跡が高々 1 しか得ないため $\gamma = 1$ としても問題はない。しかし一般にはそうはいかない。1 ステップあたり $+0.05$ の報酬があれば、同じ常に上を選ぶ方策の非割引収益は無限大に発散してしまう。したがって、建物の暖房制御のように終わりのないタスクでは $\gamma < 1$ を用いる。そのようなタスクに対するもう 1 つの一般的な選択肢は長期的なステップあたり平均報酬の最大化であるが、本章では扱わない。

Gymnasium はエピソードが停止する 2 つの場合を区別している。`terminated=True` は、過程が終端状態に入ったことを意味する。それ以上は何も起こりえず、継続することの価値は 0 である。一方 `truncated=True` は、本来なら過程が続くはずの状態にエージェントがいるまま、通常は時間制限によって記録が打ち切り（truncation）になったことを意味する。`gym.make` は FrozenLake にこの制限を設けており、100 ステップ経過すると、たとえエージェントがまだゴールへ到達できる氷の上に立っていても `truncated=True` でエピソードが終了する。Figure 14.4 に両者の違いを示す。したがって、打ち切られたエピソードにおける報酬の和は、打ち切り後の分がすべて欠けているため本来の収益ではない。最後の状態から継続する価値を推定する手法は、この 2 つを別々に扱う必要がある。終端後の価値は 0 であるが、打ち切り後の価値は最後の状態が持つ本来の**価値**（value）$V(s)$、すなわち s 以降の期待収益であり、その厳密な定義は 14.2 節で与える。本章で継続の価値を推定する手法はすべて、`terminated` のときに限ってそれを 0 とおく。

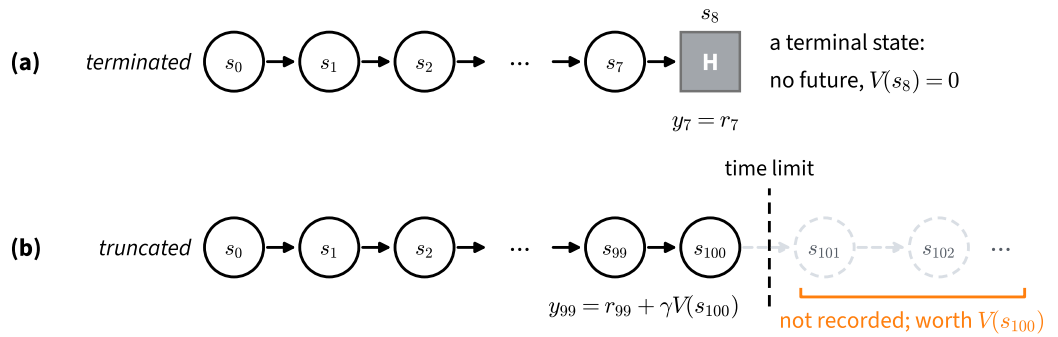


Figure 14.4: エピソードが停止する 2 つの場合。(a) 終端（Terminated）：エピソードが終端状態である穴 s_8 に入る。その先には何も続かないため、そこから継続する価値 $V(s_8)$ は 0 であり、最後のステップに対するターゲット y_7（そのステップから数えた収益）は最後の報酬 r_7 のみとなる。(b) 打ち切り（Truncated）：本来なら過程が続く氷の上にエージェントが立っている途中で、100 ステップの時間制限によって記録が打ち切られる。記録されなかった将来には最後に記録された状態から継続する価値 $V(s_{100})$ があり、これは 0 ではないため、最後のステップからの収益の推定値 $y_{99} = r_{99} + \gamma V(s_{100})$ にはこれを含めなければならない。

14.1.3 状態の選び方

遷移カーネル $P(s' \mid s, a)$ が条件付けるのは現在の状態と行動だけであり、それ以外のものは一切含まれない。これが**マルコフ性**（Markov property）である。すなわち、現在の状態がわかっていれば、それ以前の履歴は次の状態について新たな情報を何も付け加えない。この性質こそがモデルを小さく保つ鍵である。将来が現在の状態のみに依存するため、どの経路をたどってそこに到達したかにかかわらず、ある状態にいることの良さを状態ごとの 1 つの数値に要約できる。湖においては、この性質は構成上そのまま成り立つ。シミュレータは先ほど読み出した行 `P[s][a]` から次のマスをサンプリングし、滑りを毎回新たに抽選するからである。しかし一般のシステムでこの性質が成り立つかどうかは、何を状態と呼ぶかにかかっている。

14.1.3.1 状態拡張

記録された状態 s_t が車両の位置であり、行動 a_t がその加速度であるとする。次の位置は現在の速度にも依存するが、現在の位置だけでは速度はわからない。連続する 2 つの位置があれば速度を推定できるため、次の位置は過去 2 時点の位置に依存することになり、次式で表される。

$$s_{t+1} = f(s_t, a_t, s_{t-1}), \tag{14.3}$$

したがって、位置単独ではマルコフにならない。ここで状態を（位置、速度）の対として定義すれば問題は解消する。ニュートン力学により、現在の対と加えた加速度から次の位置と速度が一意に決まるからである。つまりマルコフ性とは、物理系そのものだけでなく、状態をどう表現するかという表現の性質なのである。

この構成法を**状態拡張**（state augmentation）と呼ぶ。将来を予測するために必要な情報が含まれるまで状態を拡大するのである。もっとも、拡大された状態は巨大になりうるし、必要な情報が未知であったり、保持するには大きすぎたり、そもそも観測不可能であったりすることもある。タスクの一部として時間制限が課されている場合は単純な例である。100 ステップしか許されていないとき、あるマスでの最良のコマンドは残りステップ数に依存しうるが、その残り回数を状態に加えればマルコフ性が回復する（演習問題 5）。

14.1.3.2 部分観測性

状態拡張を行うには、追加する変数が観測可能でなければならない。ポーカーのプレイヤーは相手の手札を見ることができない。エージェントが状態の一部しか明らかにしない観測（observation）o_t を受け取るとき、その問題は**部分観測可能**（partially observable）な MDP となる。厳密な解法は、それまでに観測したすべてを与えたもとでの隠れ状態（hidden state）の事後分布（posterior distribution）、すなわちエージェントの**信念**（belief）を追跡するものである（演習問題 7）。実際には、エージェントはこれをメモリで近似する。初期の Atari エージェントは、1 フレームだけでは位置しかわからず速度がわからないため、直近の数フレームを積み重ねて入力とした。現在のエージェントは、言語モデルがコンテキスト全体を参照するのと同様に、再帰状態（recurrent state）を保持するか履歴全体にアテンションを向ける。本章の残りの部分では、エージェントがマルコフ状態を直接観測できるものと仮定する。

14.1.4 報酬設計

状態・行動・遷移カーネルは環境を記述し、報酬はタスクを記述して設計者がこれを選ぶ。最適化手法は、書き下された報酬と設計者の本来の意図との間にいかなるずれがあろうと、書き下されたとおりの報酬を最大化する (Krakovna et al. 2020)。湖の例を使えば、わずか数行の算数でこの現象を確かめられる。

FrozenLake の報酬は**疎**（sparse）であり、ゴールに入るステップを除けばすべてのステップで 0 である。一度もゴールに到達したことのないエージェントは 0 しか目にしていないため、手助けをしたくなる。そこで $\ell(s)$ をマス s からゴールまでのマンハッタン距離、すなわち途中に穴がないとしたときに要する移動回数とし、ゴールに近づく移動のたびに 0.3 のボーナスを与えることにしよう。

$$\tilde{r}(s, a, s') = r(s, a) + 0.3\,\mathbf{1}\big(\ell(s') < \ell(s)\big). \tag{14.4}$$

このボーナスはエージェントをゴールへ導くように見えるが、一度もゴールせずに稼ぎ続けることができてしまう。最下段のゴールの隣にある 2 つのマス 13 と 14（$\ell = 2$ と $\ell = 1$）を取り上げ、ひとまず氷が滑らないものと考えよう。14 から 13 への移動は何も得ずコストもかからない

が、13 から 14 に戻ると 0.3 を得る。マス 14 からゴールに入れば、本来の報酬とボーナスを合わせて 1.3 を獲得してエピソードが終了する。ところがその代わりに 13 との往復を繰り返せば、2 ステップごとに 0.3 を永遠に獲得し続けられ、$\gamma = 0.95$ ではその価値は $0.3\gamma/(1-\gamma^2) \approx 2.9$ に達する。欠陥の原因は、近づく移動に報酬を与える一方で遠ざかる移動に何もペナルティを科さないため、どんなループを一周しても正の報酬が積み上がってしまう点にある。Randløv and Alstrøm (1998) も、ゴールへの前進に対して報酬を与えたシミュレーション上の自転車が、円を描いて走り続けることを学習してしまうという同じ欠陥に直面した。

ループを回って稼げないボーナスにするには、遠ざかる移動に対して、戻るときに与えるのと同じ分だけ差し引かなければならない。次の形をしたボーナスは、

$$\tilde{r}(s, a, s') = r(s, a) + \gamma\Phi(s') - \Phi(s), \tag{14.5}$$

ポテンシャルと呼ばれるある関数 $\Phi : \mathcal{S} \to \mathbb{R}$ に対してこの性質を持つ。出発点に戻るループを一周すると差分 $\Phi(s') - \Phi(s)$ が打ち消し合うため、割引の影響を除けばループからは何も得られない。この構成法を**ポテンシャルベースのシェイピング**と呼ぶ。$\Phi(s) = -0.3\,\ell(s)$ とおくと、13 から 14 への移動には 0.315 を与え、逆に戻る移動には 0.27 を科す。するとマス 14 から永遠に往復しても合計で 0.3 しか得られない一方、ゴールに入れば変わらず 1.3 を得られる。すなわちシェイピング後の報酬でも、元の報酬が与える 1 の分だけ正確にゴールがループを上回り続ける。Figure 14.5 は、このループ上で 2 つのボーナスを並べて比較したものである。それでもこのようなボーナスは有用である。ポテンシャルがゴールの報酬を経路全体に広げるため、学習器は初めてゴールに到達するずっと前から手がかりとなる信号を得られるからである。

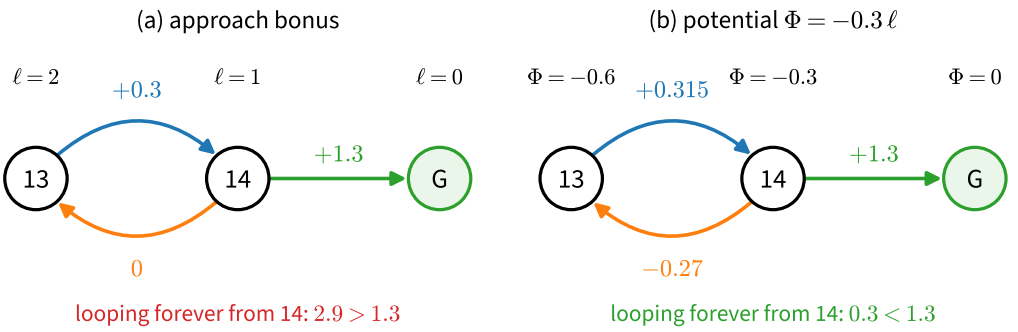


Figure 14.5: 滑らない氷の上で、ゴール G の隣にあるマス 13 と 14 に設けた 2 種類のボーナス。各矢印にはその移動に対するシェイピング後の報酬が添えられている。(a) 接近ボーナスはゴールへ向かう移動に 0.3 を与え、戻る移動には何も科さないため、マス 14 から永遠に往復する価値が 2.9 となり、ゴールに入る価値 1.3 を上回ってしまう。(b) ポテンシャル $\Phi = -0.3\,\ell$ はゴールへ向かう移動に 0.315 を与え、戻る移動に 0.27 を科す。これにより往復の価値はわずか 0.3 となり、ゴールに入るほうが良い選択になる。

残る条件が 1 つある。終端状態においてポテンシャルが 0 でなければならないことである。シミュレータは終端状態で停止するため、そこのポテンシャルが非ゼロであると、エピソードがどの終端状態でいつ終わったかに依存する最終項が残ってしまう。そのような項は、どの振る舞いが最良であるかを変えうる。先ほどの $\Phi = -0.3\,\ell$ はゴールでは 0 であるが穴では 0 ではないため、穴では明示的に 0 に設定しなければならない。滑りやすい氷の上で接近ボーナスが最良の方策に何をもたらすかは 14.2 節で計算し、あわせてポテンシャル形式が最良の方策を変化させないことも証明する。

14.1.5 2 つの特別な場合

本章では、湖が持つ難しさをそれぞれ 1 つずつ取り除いた 2 種類の MDP が繰り返し登場する。状態が 1 つしかない場合、各ラウンドは行動を 1 つ選んで報酬を観測するだけで完結し、どの行動も将来に影響を与えない。これが**多腕バンディット問題**（multi-armed bandit）であり、プレイヤーが実際に引きながら払い戻し率を探り当てなければならない一列のスロットマシンにちなんで名付けられた。この設定はどの行動を試すべきかという問題を純粋に取り出すものであり、14.4 節で詳しく調べる。2 つ目の場合は、将来への影響は残しつつ遷移のランダム性を取り除いたものであり、応答を書き進める言語モデルを記述する。

14.1.5.1 MDP としてのテキスト生成

1 トークンずつ応答を書き進める言語モデルは、単純なダイナミクスを持つ MDP に当てはまる。状態はプロンプトとそれまでに書かれたトークン列を合わせたもの、行動は次のトークンであり、遷移は確率 1 でそのトークンを状態に連結する。報酬は応答が完成したときに一度だけ届く。**検証器**（verifier）と呼ばれる判定器が応答を既知の正解と照合するか、あるいは学習されたモデルがそれにスコアを付けるのである。2 つの性質がこの MDP を湖と際立たせている。第 1 に遷移が決定論的であるため、ランダム性はすべて方策から生じる。第 2 に状態がプレフィックス全体であるため、同じ状態が二度訪れることはなく、状態全体はプロンプトを根とする木をなす。

遷移が決定論的で、最後に一度だけ報酬が与えられ、状態の再訪もないとき、期待収益は方策が各完成応答に与える確率のみに依存する。そのため、応答全体をプロンプトに対して選ばれる 1 つの行動として扱っても何も失われない。このとき MDP は 1 ステップに縮約される。プロンプト x（プロンプト上の分布 μ_0 から抽出）が状態、完全な応答 y が行動、そして検証器のスコア $r(x, y)$ が報酬となる。言語モデル向けの手法が応答全体にスコアを付けて更新できる理由はここにある。トークンレベルの見方には 15.8 節で立ち返る。

14.1.5.2 検証器タスク

本章の 2 つ目の共通例題として、厳密に計算できるほど小さな具体例でこの 1 ステップの見方を形にしよう。このタスクには 4 つの算数のプロンプトと、それらすべてで共有される 9 個の応答候補がある。検証器は、応答がプロンプトの正解と完全に一致したときに 1 を与え、それ以外では 0 を与える。プロンプトのうち `8*8` と `9*9` の 2 つには正解となる候補が存在しないため、どの方策もプロンプト全体の平均で 0.5 を超えるスコアは出せない。9 番目の候補である「one of 4, 12, 64 or 81」は、すべての答えを一度に並べたものである。完全一致の検証器はこの**両賭け**（hedge）を棄却するが、正解を含んでさえいればどんな応答でも受け入れる寛容な判定器であれば、すべてのプロンプトでこれに報酬を与えてしまうだろう。ここにも形を変えた報酬設計の問題が現れている。

方策は事前学習済み言語モデルの代役を務める。これはプロンプトごとに行、応答ごとに列を持つ選好のテーブル θ であり、ソフトマックス回帰（softmax regression）と同様に各行にわたるソフトマックスによって確率 $\pi(y \mid x) \propto \exp \theta_{xy}$ へと変換される。参照テーブル θ_{ref} は、-4 に設定された両賭けの列を除いてすべて 0 である。したがって参照方策からの応答は、通常の 8 候補がそれぞれ確率約 $1/8$ となり、両賭けが約 0.2% となる。クラス `VerifierTask` はこれらの要素をまとめたものである。メソッド `verify(x, y)` が判定器であり、属性 `R` は判定器の報酬をプロンプトごとの行・応答ごとの列からなるテーブルとして保持する。メソッド `policy(theta)` は選好テーブルの各行にソフトマックスを適用する。メソッド `success(pi)` はサンプリングされた 1 つの応答の期待スコアを 4 つのプロンプトにわたって平均した値、すなわち μ_0 をプロンプト上の一様分布としたこの 1 ステップ MDP の期待収益である。オプションの引数 `reward` を渡せば、別の判定器のもとで方策を評価できる。訓練はこのテーブルを書き換えるが、ここでは参照方策のスコアだけを確認しよう。

```
class VerifierTask:
    """Four prompts, nine candidate responses shared by all of them, an
    exact-match verifier, and a reference policy that rarely hedges."""
    prompts = ['2+2', '3*4', '8*8', '9*9']
    answers = ['4', '12', '64', '81']
    resp = ['1', '3', '4', '7', '12', '19', '23', '35',
            'one of 4, 12, 64 or 81']          # the last one is the hedge

    def __init__(self):
        self.theta_ref = np.zeros((len(self.prompts), len(self.resp)))
        self.theta_ref[:, -1] = -4.0           # the reference rarely hedges
        self.pi_ref = self.policy(self.theta_ref)
        self.R = self.reward_table(self.verify)   # R[x, y] = r(x, y)

    def verify(self, x, y):
        """The verifier: reward 1 iff response y is prompt x's answer."""
        return float(self.resp[y] == self.answers[x])

    def reward_table(self, reward):
        """reward(x, y) for every prompt x (rows) and response y (columns)."""
        return np.array([[reward(x, y) for y in range(len(self.resp))]
                         for x in range(len(self.prompts))])

    @staticmethod
    def policy(theta):
        """pi(y | x): a softmax over each prompt's row of preferences."""
        z = np.exp(theta - theta.max(1, keepdims=True))
        return z / z.sum(1, keepdims=True)

    def success(self, pi, reward=None):
        """E r(x, y), y ~ pi(. | x), averaged over the prompts; r = verify
        unless another reward function is given."""
        R = self.R if reward is None else self.reward_table(reward)
        return (pi * R).sum(1).mean()

task = VerifierTask()
print(f'success of a response sampled from the reference: '
      f'{task.success(task.pi_ref):.3f}')
```

```
success of a response sampled from the reference: 0.062
```

参照方策からサンプリングされた応答が検証器を通過する確率は 0.062 である。回答可能な 2 つのプロンプトではそれぞれ正解候補の確率が約 $1/8$ であり、残りの 2 つのプロンプトは何も寄与しないため、平均は約 $1/16$ になる。このタスクでの訓練は、0.062 と上限である 0.5 との差を埋めることになる。

これで意思決定問題を書き下せるようになった。湖は 2 つの配列と割引率として、検証器タスクはスコアのテーブルとして表され、それぞれについて軌跡が得る収益も定まった。この種のモデルは任意の状態と行動を指定して問い合わせることができ、これを利用するのが**モデルベース**（model-based）手法である。対照的に、**モデルフリー**（model-free）手法はエージェント自身の経験が生み出す遷移しか見ることができない。マス 9 で立てた問いはまだ未解決のままである。**下**と**右**を比較するには各コマンドがもたらしうる収益を知る必要があり、それはエージェントが次に到達しうるすべてのマスでどう行動するかに依存する。14.2 節では、すべてのマスに対する

これらの価値を一度に計算する。

14.1.6 まとめ

マルコフ決定過程は、現在の状態が過去のうち将来に関わるすべてを保持しているという仮定のもとで、時間にわたる行動の問題を状態・行動・遷移カーネル・報酬の組 $(\mathcal{S}, \mathcal{A}, P, r)$ として書き下す。エージェントは期待割引収益を最大化する。割引率 γ は収益を有限に保つとともに $1/(1-\gamma)$ ステップの実効ホライズンを定め、重みは 1 ホライズンあたり約 e 分の 1 に減衰する（γ が 1 に近い場合）。終端状態は過程そのものを終わらせるのに対し、打ち切りは記録を止めるだけであるため、継続の価値が 0 になるのは終端後に限られる。自明に見える状態がマルコフでない場合でも、欠けている情報が観測可能であれば状態拡張によってマルコフ性を回復できる。報酬は設計上の選択であり、ループを回って稼げるボーナスはエージェントの振る舞いを変えてしまうが、ポテンシャルに基づくボーナスはそのようには稼げない。FrozenLake と、トークンレベルの MDP を 1 ステップに縮約した 4 プロンプトの検証器タスクが、本章を通して用いる共通例題である。

14.1.7 演習問題

1. **状態が含むべき情報。**以下の各システムでは、自明な状態はマルコフではない。マルコフ性を回復するために最小限追加すべき情報を挙げ、エージェントがそれを観測できるかどうかを述べよ。

   1. 状態を現在位置とする配送ロボット。ただし、各移動で地形と積載量に応じた分だけバッテリーが消費され、バッテリーが空になると停止するものとする。
   2. 状態をユーザーのプロフィールとする推薦システム。ただし、あるトピックに対するユーザーの関心は、最近そのトピックが推薦された回数に応じて低下するものとする。
   3. 状態を盤上の駒の配置とするチェスプログラム。ただし、同一局面が 3 回現れた時点、または駒取りもポーンの移動もないまま双方 50 手が経過した時点で直ちに引き分けとなって対局が終了するものとする。

   これら 3 つのシステムのうち、追加すべき情報がエピソードの長さに伴って増大しうるものはどれか。また、そこでは何によってその情報が有界に保たれているか。

2. [code] **カーネルから予測するエピソード。**本節の軌跡の例 τ では、`rng.integers(4)` によって毎ステップの行動を一様ランダムに選んでいる。この方策 π のもとで、次の状態の分布は

$$P_\pi(s' \mid s) = \frac{1}{4} \sum_{a \in \mathcal{A}} P(s' \mid s, a), \tag{14.6}$$

   すなわち `mdp.P.mean(axis=1)` となる。

   1. `from_gym` が `+=` で確率を累積しているのは、1 つの状態行動対に対する Gymnasium の複数の結果が同じ次の状態に至りうるからである。この現象が起きるすべての状態行動対を見つけ、なぜそれらの対だけで他の対では起きないのかを Figure 14.2 から説明せよ。仮に `P` を埋める行で `=` による代入を使ったとすると、`mdp.P` のどの行の和が 1 でなくなり、それぞれの和はいくらになるか。
   2. d_t^π を s_t の分布（状態 0 上の点質量 $d_0^\pi = \mu_0$ から出発）とし、$d_{t+1}^\pi(s') = \sum_s d_t^\pi(s) P_\pi(s' \mid s)$ が成り立つとする。$t \le 30$ に対して、エージェントがゴールに到達している確率、穴に落ちている確率、およびエピソードがまだ継続している確率をプロットせよ。また、$t \to \infty$ におけるそれぞれの極限はいくらか。

3. env の時間制限を無視したとき、エピソードの平均長が $\sum_{t\geq 0} \Pr(s_t \text{ is not terminal})$ に等しいことを示せ。ランダム方策で env を 10,000 エピソード実行し、ゴールに到達したエピソードの割合と平均エピソード長を、d_t^π が予測する値と比較せよ。なお、env の 100 ステップ制限によってランダムなエピソードが終了する確率は 10^{-8} 未満であるため、比較には影響しない。

3. **期待報酬**。Gymnasium は報酬を遷移に付随させており、ゴールに入ると 1 が与えられる。from_gym はこの報酬 $\rho(s,a,s')$ をその期待値 $r(s,a) = \sum_{s'} P(s' \mid s,a)\,\rho(s,a,s')$ に置き換えている。

   1. 任意の方策と任意の初期状態に対し、ステップ t で $\rho(s_t,a_t,s_{t+1})$ が与えられる場合と $r(s_t,a_t)$ が与えられる場合とで、期待割引収益が等しくなることを示せ。さらに、(s,a) を訪れるたびに他のすべてと独立に平均 $r(s,a)$ の分布から報酬が抽出される場合へとこの議論を拡張せよ。
   2. $r(s,a)$ の規約のもとでは、FrozenLake のどのようなエピソードが正の収益を持つか。本節の一様ランダム方策について、10,000 エピソードから 2 つの規約のそれぞれにおける収益が正となる確率を推定せよ。したがって、モデル Equation 14.1 は収益のどの性質を決定し、どの性質を決定しないか。

4. **終端としての割引**。割引収益 Equation 14.2 はステップ t の報酬に重み γ^t を掛ける。

   1. 本節では、割引重みの 95% が $\log(0.05)/\log(\gamma)$ ステップ以内に収まり、これが約 3 実効ホライズンにあたると述べた。任意の割合 $p \in (0,1)$ に対して

$$\frac{\ln(1/p)}{\ln(1/\gamma)} \leq \frac{\ln(1/p)}{1-\gamma}, \tag{14.7}$$

      が成り立ち、かつ $\gamma \to 1$ のとき両辺の比が 1 に収束することを示せ。この不等式を用いて、Figure 14.3 における $\gamma = 0.5$、0.9、0.99 でのステップ数を説明せよ。

   2. MDP を変形し、毎回報酬を受け取った後に確率 $1-\gamma$ で報酬 0 の吸収状態へ遷移し、そうでなければ P に従って遷移するようにする。変形後の過程における任意の方策の期待非割引収益が、元の過程における期待割引収益に等しくなること、および受け取る報酬の個数の期待値が $1/(1-\gamma)$ であることを示せ。この解釈のもとで TabularMDP の配列を読み解いたとき、γP の各行の和は何を意味するか。

5. **時間制限**。gym.make は FrozenLake を時間制限でラップしており、100 ステップ後にエピソードは truncated=True で終了する（14.1.2.3 節）。いま、この制限がタスクの一部であり、目的関数が 100 ステップ以内にゴールへ入る確率であるとする。制限時間の満了を吸収状態への遷移としてモデル化せよ。このとき次の状態の分布が残りステップ数 h に依存するため、位置単独ではマルコフ状態にならないのに対し、対 (s,h) はマルコフ状態になることを示せ (Pardo et al. 2018)。ある位置における最良の行動が h に依存しうるのはなぜか。

6. **将来に影響しない行動**。多腕バンディット問題は状態を 1 つしか持たないため、どの行動も将来に影響を与えられない。

   1. より一般に、遷移カーネルが行動に依存せず、$P(s' \mid s,a) = P(s' \mid s)$ がすべての a に対して成り立つとする。このとき任意の γ に対し、各状態で $\arg\max_a r(s,a)$ に属する行動を選ぶ方策が、あらゆる初期状態からの期待割引収益を最大化することを示せ。
   2. 2 状態 2 行動の MDP であって、この貪欲方策が小さい γ では最適方策となるが大きい γ では最適にならないものを 1 つ構成し、最適な行動が切り替わる γ の値を求めよ。

7. **信念状態**。●部分観測可能な MDP において、エージェントは観測 o_t（確率 $O(o_t \mid s_t)$ で抽出）を状態 s_t の代わりに受け取る。エージェントの**信念** $\beta_t(s) = \Pr(s_t = s \mid o_0, a_0, \dots, a_{t-1}, o_t)$ とは、それまでに観測したすべてを与えたもとでの隠れ状態に関する事後分布である (Kaelbling et al. 1998)。

   1. 隠れ状態のマルコフ性を用いて、次の更新式を導出せよ。

$$\beta_{t+1}(s') \propto O(o_{t+1} \mid s') \sum_{s \in \mathcal{S}} P(s' \mid s, a_t)\, \beta_t(s). \tag{14.8}$$

   2. 信念がマルコフ状態であること、すなわち β_t と a_t が与えられたとき、β_{t+1} の分布と期待報酬がそれ以前の観測に依存しないことを示せ。得られる信念上の MDP の状態空間と遷移カーネルはどのようなものになるか。

   3. Bakker (2001) を改変した T 字迷路では、エージェントは出発時に確率 $1/2$ ずつで**左**または**右**の手がかりを観測し、続いてすべて同一の観測を生じる N マスの通路を 1 ステップに 1 マスずつ進んで分岐点に到達する。分岐点で手がかりの方向へ曲がれば報酬 1、逆方向へ曲がれば報酬 0 を得る。直近 k 個の観測のみに基づいて行動する方策は、成功確率が高々 $1/2$ にとどまることを示せ（ただし $k \geq N + 2$ の場合を除く）。分岐点における信念はどうなるか。また、最適に行動するにはどれだけのメモリがあれば十分か。数フレームを重ねる手法が Pong では機能するのに、この課題では機能しないのはなぜか。

8. [extended] **計画ホライズンとモデル誤差**。●●目的関数の割引率 γ はタスクによって固定されるが、プランナはそれより小さい**計画割引率** γ' を用いて方策を計算してもよい。Jiang et al. (2015) は、モデルをデータから推定する場合には $\gamma' < \gamma$ としたほうが $\gamma' = \gamma$ よりも優れた方策を与えうることを見出した。$V^{\pi}_{\gamma}(s)$ を、状態 s から方策 π および割引率 γ のもとで得られる期待割引収益とする。(b) では 14.2 節の価値反復法と厳密な方策評価（`d2l.value_iteration` および `TabularMDP.policy_value`）を用いる。

   1. 報酬が $[0, r_{\max}]$ に収まり、$\gamma' \leq \gamma$ であるとする。任意の方策 π と状態 s に対して

$$0 \leq V^{\pi}_{\gamma}(s) - V^{\pi}_{\gamma'}(s) \leq \frac{(\gamma - \gamma')\, r_{\max}}{(1-\gamma)(1-\gamma')}, \tag{14.9}$$

      が成り立つことを示し、真のモデル上では γ' に対して最適な方策が γ のもとでも高々この量しか失わないことを結論せよ。

   2. `TabularMDP` を用いてこの実験を小規模に再現せよ。10 状態 2 行動のランダムな MDP を 100 個生成する。その際、各状態行動対に対して重複なしで一様に 5 個の次の状態を選び、$[0, 1]$ から一様に抽出した重みを与えて正規化し、各 $r(s, a)$ も $[0, 1]$ から一様に抽出せよ。各 MDP と各 $n \in \{2, 5, 10, 20\}$ に対し、推定遷移カーネル $\hat{P}$ を 20 個ずつ、各対からサンプリングした n 個の次の状態から構築せよ（真の r はそのまま保つ）。各推定モデルの最適方策を $\gamma' \in \{0, 0.1, \dots, 0.9, 0.99\}$ について計算し、真のモデル上で $\gamma = 0.99$ において評価して、状態・推定カーネル・MDP にわたって損失 $V^{*}_{\gamma}(s) - V^{\pi}_{\gamma}(s)$ を平均せよ（ここで V^{*}_{γ} は真のモデルの最適方策の価値である）。価値反復法に必要なスイープ数は約 $\log(10^{-8}) / \log \gamma'$ 回であり、$\gamma' = 0.99$ では約 2,000 回、それ未満ではるかに少ない。このスケジュールを使えば実験は数分の CPU 時間で完了する。損失を γ' に対して各 n ごとにプロットせよ。

3. どの γ' が各 n に対して損失を最小化し、それは n に伴ってどのように移動するか。本問で上界を与えたホライズンのバイアスと $\hat{P}$ の誤差という 2 つの損失要因によって曲線の形状を説明し、どちらが γ' とともに増大するかを述べよ。

14.2 動的計画法

前節では、氷の湖を 2 つの配列と割引率として記述し、1 つの問いを未解決のまま残した。マス 9 では、隣の穴へエージェントを滑り込ませない唯一の指示は**下**だが、これを選ぶと 3 回中 2 回は横方向に滑ってしまう。この安全性は、手数を失うだけの価値があるのだろうか。滑らない氷の上なら迷う余地はない。ゴールへの最短経路は 6 手であり、それをたどるのが最善である。一方、滑る氷の上では最短経路が穴の脇を通るため、1 マス進む指示であっても、滑ればその場でエピソードが終了しかねない。どの指示が最善かは、次のマスを見ただけでは判断できない。着地しうるすべてのマスから、その後も適切に行動し続けたときにどれだけの報酬（reward）を期待できるかに依存するからである。

Richard Bellman は 1950 年代にこのような多段階の意思決定を研究し、状態（state）の**価値**（value）と呼ばれるこのスコアが局所的な方程式を満たすことを見いだした (Bellman 1957b)。あるマスの価値は、1 手の報酬にその移動先のマスの割引価値を加え、滑りによる結果について平均したものに等しい。価値は将来全体によって定義されるが、各マスの価値は隣接するマスの価値としか結びついていないため、1 ステップ先読みを繰り返すことですべての価値をまとめて計算できる。Bellman はこの手法を**動的計画法**（dynamic programming）と呼んだ。この手法が与える解は、後の節で扱う学習手法が近づこうとする厳密解そのものであり、小規模な問題では学習手法を検証する基準として使える。

本節では、固定した方策（policy）と最良の方策に対する価値を定義し、ベルマン方程式を導出したうえで、先読みの反復が収束することを停止規則とともに示す。滑る湖においてこの手法が見つける方策は、最短経路の約 16 倍の頻度でゴールに到達し、4 つのマスではゴールから遠ざかる向きに進むが、その理由も計算によって明らかにする。最後には前節の接近ボーナスに立ち返り、それが最良の方策にどのような影響を与えるかを計算する。本節で行う計算はすべて、既知の遷移表 P を参照する。

```
import gymnasium as gym
```

14.2.1 価値とベルマン方程式

14.2.1.1 方策と価値

14.1 節では、行動（action）を選ぶあらゆる規則を方策と呼んだ。一般に**方策** $\pi(a \mid s)$ は、行動 $a \in \mathcal{A}$ についての、状態 $s \in \mathcal{S}$ を与えた条件付き分布（conditional distribution）である。たとえば湖のあるマスにおいて、**左**、**下**、**右**、**上**の確率が $(0.4, 0.2, 0.1, 0.3)$ のようになる。$\pi(\cdot \mid s)$ で s における分布全体を表す。**決定論的**（deterministic）な方策は 1 つの行動にすべての確率を割り当てるものであり、これを $\pi(s)$ と書く。

方策を 1 つ固定しよう。s_0 から出発したエージェントは、$a_t \sim \pi(\cdot \mid s_t)$ を引き、$s_{t+1} \sim P(\cdot \mid s_t, a_t)$ へと遷移して、割引収益（discounted return）$\sum_t \gamma^t r(s_t, a_t)$（14.1 節）を受け取る。実行するたびに異なる軌跡（trajectory）が生じるため、初期状態を平均収益によって次のように評価する。

$$V^{\pi}(s_0) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t),\, s_{t+1} \sim P(\cdot|s_t,a_t)} \Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \Big]. \tag{14.10}$$

これは π の**価値関数**（value function）である。初期状態は任意であるため、V^{π} はすべての状態に 1 つの数値を割り当てる。すなわち、そこから π に従うエージェントが獲得を期待できる報酬である。湖の上ではゴールに入ったときにしか報酬が得られないため、$V^{\pi}(s)$ は s からゴールに到達する確率を、各成功に γ^t（ゴールに入るステップを t とする）の重みを付けて求めたものになる。

マス 9 における問いは各指示を比較するものであるのに対し、V^{π} はそれらについて平均をとってしまう。そこで**行動価値**（action value）は、最初の行動を固定し、それ以降は方策に従うものとして次のように定義される。

$$\begin{aligned} Q^{\pi}(s_0, a_0) &= r(s_0, a_0) \\ &\quad + \mathbb{E}_{a_t \sim \pi(\cdot|s_t),\, s_{t+1} \sim P(\cdot|s_t,a_t)} \Big[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \Big]. \end{aligned}$$

和が $t = 1$ から始まるのは、最初の報酬が a_0 によって確定しているからである。すなわち、最初の 1 ステップだけ指定された行動をとり、その後はずっと方策に従う。この行動価値を方策自身の最初の行動について平均すると、元の価値に戻る。

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, Q^{\pi}(s, a) \quad \text{for all } s \in \mathcal{S}. \tag{14.11}$$

したがって、$V^{\pi}(s)$ は数値 $Q^{\pi}(s, \cdot)$ の重み付き平均であり、その最大値を決して超えない。すなわち $\max_a Q^{\pi}(s, a) \geq V^{\pi}(s)$ が成り立ち、等号が成立するのは方策が最大値を達成する行動にすべての確率を割り当てる場合に限られる。本節におけるすべての改善ステップは、この不等式を出発点とする。

14.2.1.2 期待値形式

価値は無限和として定義されるが、マルコフ性（Markov property）によって各状態の価値は隣接する状態の価値と結びつけられる。軌跡を最初の遷移の直後で分割してみよう。前半には行動 a_0、報酬 $r(s_0, a_0)$、および次の状態 s_1 が含まれ、後半は s_1 から始まる軌跡となる。マルコフ性により、後半の期待収益は s_1 のみを通じて過去に依存し、その値は $V^{\pi}(s_1)$ に等しい。したがって、s_0 における価値は期待即時報酬に次の状態の割引期待価値を加えたものになる。

$$V^{\pi}(s_0) = \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} \Big[r(s_0, a_0) + \gamma\, \mathbb{E}_{s_1 \sim P(\cdot|s_0,a_0)} \Big[V^{\pi}(s_1) \Big] \Big]. \tag{14.12}$$

外側の期待値は方策による a_0 の選択について平均をとり、内側の期待値はその行動のもとで環境（environment）が遷移する次の状態 s_1 について平均をとる。状態空間と行動空間が有限であれば、これらの期待値は和で表される。

$$\begin{aligned} V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \Big[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^{\pi}(s') \Big], \\ \text{for all } s \in \mathcal{S}. \end{aligned} \tag{14.13}$$

最初の行動も条件として与えれば、行動価値に対する対応する方程式が得られる。

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s')\, Q^\pi(s',a'), \quad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}. \tag{14.14}$$

これらが**ベルマン期待方程式**（Bellman expectation equation）である。固定した方策に対して、Equation 14.13 は $|\mathcal{S}|$ 個の一次方程式からなる、$|\mathcal{S}|$ 個の未知数 $V^\pi(s)$ についての連立一次方程式（linear system）となる。無限ホライズンは再帰の中へと姿を消したのである。

湖の 1 つのマスを見れば、これがどのような連立方程式かがわかる。ゴールの隣にあるマス 14 で**下**を指示する方策を考えよう。**下**に進もうとすると下端の壁にぶつかってエージェントはマス 14 にとどまり、2 方向の滑りはエージェントを左のマス 13 か、報酬 1 を与える右のゴールへと運ぶ。どの結果も確率は $1/3$ であり、終端状態（terminal state）であるゴールの価値は 0 であるため、Equation 14.13 はマス 14 において次のようになる。

$$V^\pi(14) = \frac{1}{3} + \frac{\gamma}{3}\left[V^\pi(13) + V^\pi(14)\right].$$

滑った結果としてエージェントが元の場所にとどまることがあるため、未知数 $V^\pi(14)$ は両辺に現れる。さらに $V^\pi(13)$ もまた、マス 14 を含む自身の隣接マスに依存する。ゲーム木であれば、葉から根へ向かって価値を 1 つずつ順番に埋めていくことができる。しかし滑る湖にはループがあるため、方程式はまとめて解かなければならない。すなわち、1 つの連立一次方程式として解くか、初期推定値から始めて右辺を何度も繰り返し適用するかである。以下ではこの両方の方法を扱う。

これまでの式の右辺はいずれも、1 つの状態行動対（state-action pair）に対して同じ角括弧内の式、すなわち即時報酬に次の状態の割引期待価値を加えたものを計算している。この 1 ステップ先読みは、後続状態から元の状態へと価値を引き戻すことから**バックアップ**と呼ばれる。`TabularMDP`（14.1 節）の配列上では、すべての状態行動対のバックアップは `P[s, a, s']` の最後の軸と価値ベクトル `V` を縮約する 1 回の行列積になる。これをメソッド `backup` としてクラスに追加しよう。このメソッドは (s,a) で添字付けされた角括弧内の値の配列を返す。

```
@d2l.add_to_class(d2l.TabularMDP)
def backup(self, V):
    """Q(s, a) = r(s, a) + gamma * sum_{s'} P(s'|s, a) V(s') for every
    state-action pair: the bracket of the Bellman equations."""
    return self.r + self.gamma * self.P @ V
```

14.2.1.3 最適方程式

最適（optimal）な方策とは、価値を最大化する方策である。原理的には、最良の方策が初期状態に依存したり、エピソードの履歴全体を参照したり、行動をランダム化したりする可能性も考えられる。しかし割引付きの有限 MDP では、14.1 節で述べたように、そのいずれも必要ない。

命題（一様最適方策の存在）。割引率 $0 \le \gamma < 1$ の有限 MDP には、すべての状態で同時に最適となる決定論的な定常方策 π^* が存在する。すなわち、$V^{\pi^*}(s) \ge V^\pi(s)$ がすべての $s \in \mathcal{S}$ と、確率的方策や履歴に依存する方策を含むすべての方策 π に対して成り立つ (Puterman 1994)。

履歴に依存する方策の価値も、同じ期待収益によって定義される。$V^* \equiv V^{\pi^*}$ および $Q^* \equiv Q^{\pi^*}$ と書き、V^* がどの方程式を満たすかを考えよう。2 つの比較から答えが得られる。第 1 に、π^* の

期待方程式より、$V^*(s)$ は角括弧内の式 $r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s')$ に行動 $a = \pi^*(s)$ を代入したものに等しいため、$V^*(s)$ が角括弧内の式の最大値を超えることはない。第 2 に、ある行動 a を s で 1 回だけとり、その後は π^* に従う方策の価値は、a に対する角括弧内の式に等しい。上の命題により π^* を上回る方策は存在しないため、どの角括弧内の式も $V^*(s)$ を超えない。これら 2 つの比較を合わせると、$V^*(s)$ は角括弧内の式の最大値に等しいことがわかる。

$$V^*(s) = \max_{a \in \mathcal{A}} \Big[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a)\, V^*(s') \Big], \quad \text{for all } s \in \mathcal{S}. \tag{14.15}$$

これが**ベルマン最適方程式**（Bellman optimality equation）である (Bellman 1952, 1957b)。言葉で言えば、最適な行動選択のもとでのある状態の価値は、最良の初手を打ち、その移動先がどこであれそこから再び最適に行動し続けたときの価値に等しい。期待値形式とは異なり、この方程式には方策が一切現れない。未知の表 V^* に関する方程式であり、最大値演算を含むため非線形である。また第 1 の比較から、行動 $\pi^*(s)$ がこの最大値を達成することもわかる。したがって V^* が与えられれば、モデルを用いた 1 ステップ先読みによって最適な行動が求まる。

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \Big[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a)\, V^*(s') \Big]. \tag{14.16}$$

Equation 14.15 の行動価値版は、角括弧の中に $V^*(s') = \max_{a'} Q^*(s', a')$ を代入することで得られる。

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \max_{a' \in \mathcal{A}} Q^*(s', a'). \tag{14.17}$$

これら 2 つの形式は、方策を取り出す際に必要なものが異なる。Q^* からはモデルを使わずに $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$ として最適な行動が得られるのに対し、V^* から Equation 14.16 によって取り出すには P と r が必要になる。この違いは、モデルが未知になった途端に重要となる。一方、モデルが既知であれば、Equation 14.16 の 1 ステップ先読みを複数ステップへ拡張することもできる。ボードゲームをプレイするプログラムは、ゲームのルールをモデルとして用いるか (Silver et al. 2018)、あるいは自ら学習したモデルを用いて (Schrittwieser et al. 2020)、何手も先まで探索する。

14.2.1.4 バックアップ線図

これら 4 つの方程式は共通の構造をもっており、いずれも根における価値をその 1 ステップ下の価値から組み立てている。Figure 14.6 にそれらを並べて示す。最後の図は、モデルをもたない学習器がどうしなければならないかを表している。環境の次の状態について平均をとることができないため、実際に観測した 1 回の遷移を用いるのである。

14.2.2 価値反復法

14.1 節の滑る湖を `TabularMDP`（$\gamma = 0.95$）として読み込み、あわせて表示用に行動ごとの矢印記号を用意する。

```
gamma = 0.95
ARROWS = '<v>^'                  # left, down, right, up: actions 0 to 3
```

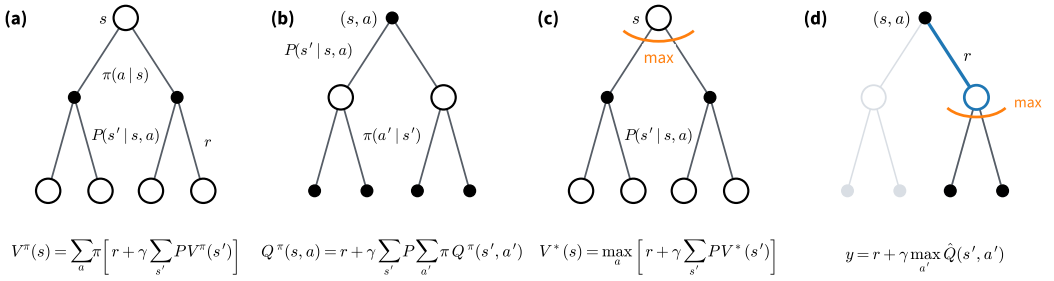


Figure 14.6: バックアップ線図。白丸は状態、黒丸は状態行動対を表し、各図は根における価値がその 1 ステップ下の価値からどのように組み立てられるかを示す。左から順に、(a) 価値関数は方策の行動と環境の次の状態について平均をとり、(b) 行動価値関数は最初の行動を固定してその後の平均をとり、(c) 最適作用素は行動についての平均を円弧で表された最大値に置き換え、(d) 標本バックアップは環境の平均をとれないため、青で示した観測された単一の遷移を用い、次の状態では最大値を維持する。

```
env = gym.make('FrozenLake-v1', is_slippery=True)
mdp = d2l.TabularMDP.from_gym(env, gamma)
```

14.2.2.1 アルゴリズム

最適方程式は V^* を陰に定義しているにすぎない。これを計算するため、Equation 14.15 の右辺を写像とみなそう。**ベルマン最適作用素**（Bellman optimality operator）$\mathcal{T}$ は、任意の表 $V : \mathcal{S} \to \mathbb{R}$ を受け取り、各状態で最大の角括弧項を並べた表を返す。

$$(\mathcal{T}V)(s) = \max_{a \in \mathcal{A}} \Big[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, V(s') \Big]. \tag{14.18}$$

最適方程式は $V^* = \mathcal{T}V^*$ と書ける。すなわち、最適価値関数は $\mathcal{T}$ の**不動点**（fixed point）である。不動点を探す最も単純な方法は、写像を繰り返し適用することである。$V_0 = 0$ から始め、次のようにスイープする。

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \Big[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, V_k(s') \Big], \quad \text{for all } s \in \mathcal{S}, \tag{14.19}$$

すなわち $V_{k+1} = \mathcal{T}V_k$ である。これが**価値反復法**（value iteration）である (Bellman 1957b)。各スイープでは状態行動対ごとに 1 回のバックアップを行い、状態ごとに最良のものを残す。backup がすべての角括弧項を一度に計算するため、コードでは 1 回のスイープが 1 行で書ける。価値が変化していく様子を観察できるよう、次の関数はすべての反復列を返す。

```
def value_iteration(mdp, num_iters):
    """Sweep V <- max_a backup(V); return the whole history of iterates."""
    V, history = np.zeros(mdp.num_states), []
    for _ in range(num_iters):
```

```
        V = mdp.backup(V).max(axis=1)
        history.append(V)
    return np.array(history)
```

これらの反復列は、単に V^* へ至る途中の通過点にとどまらない。それぞれの項がそれ自体で 1 つの問いに答えている。エージェントに残された行動ステップ数が k 回だけであり、それ以降は何も報酬が得られないとしよう。残りステップがなければ何も獲得できないため、$V_0 = 0$ である。残り 1 ステップであれば最善の結果は期待即時報酬の最大値 $\mathcal{T}V_0$ であり、帰納法により $V_k = \mathcal{T}V_{k-1}$ は k ステップにわたる最良の期待収益となる。これが**有限ホライズン**（finite-horizon）動的計画法である。H ステップ後に終了する問題は、終端から逆向きに解かれる。V_t でステップ t 以降の最良の収益を表せば、$V_H = 0$（終端報酬（terminal reward）がある場合はその値）であり、$V_t = \mathcal{T}V_{t+1}$ となる。定常の場合と比べて変わる点が 1 つある。価値が状態だけでなく時刻にも依存し、それに伴って最良の行動も時刻に依存することである。長期的には最適な安全な遠回りも、残りステップ数がわずかしかないのであれば選ぶ価値がないかもしれない。14.6 節では、制御問題をこの逆向きの形式で解く。

14.2.2.2 滑らない氷

最も単純なケースは滑りのない湖であり、そこではあらゆる指示が意図どおりに動作する。ゴールの報酬はゴールへ入る移動で支払われるため、ゴールから 1 手の距離にあるマスの価値は 1 である。2 手離れたマスの価値は γ となる。最良の移動によって 1 ステップ後に価値 1 のマスへ到達するからである。一般に、穴を避ける最短経路が j 手かかるマスの価値は $V^*(s) = \gamma^{j-1}$ である。$V_0 = 0$ から始める価値反復法は、これらの価値をゴールから同心円状に 1 層ずつ見つけていく。スイープ k の後には、ゴールから k 手以内のマスが非ゼロの価値をもち（その時点でそれぞれ確定値となっている）、残りのマスはまだ 0 のままである。Figure 14.7 に最初の 6 回のスイープを示す。

以下のコードセルでは、滑らない湖に対して 10 回のスイープを実行し、各スイープ後に非ゼロの価値をもつマスの個数を数え、貪欲方策（greedy policy）`shortest` を Equation 14.16 によって取り出す。

```
calm = d2l.TabularMDP.from_gym(
    gym.make('FrozenLake-v1', is_slippery=False), gamma)
history_calm = value_iteration(calm, num_iters=10)
V_calm = history_calm[-1]
shortest = calm.backup(V_calm).argmax(axis=1)  # ties go to the lower action
print(f'cells with nonzero value after each sweep: '
      f'{(history_calm > 0).sum(axis=1)}')
print(f'V*(s0) on calm ice: {V_calm[0]:.4f};  gamma^5 = {gamma ** 5:.4f}')
```

```
cells with nonzero value after each sweep: [ 1  3  5  7 10 11 11 11 11 11]
V*(s0) on calm ice: 0.7738;  gamma^5 = 0.7738
```

個数は 1、3、5、7、10、11 と増え、その後はゴール以外の氷のマスの総数である 11 のまま変わらない。6 回のスイープを終えるとすべての価値が確定し、スタートのマスは $\gamma^5 = 0.774$ を保持する。滑らない氷の上では、価値反復法は最短経路アルゴリズムそのものであり、1 回のスイープにつき 1 手ずつゴールから価値が逆向きに伝わっていく。その貪欲方策 `shortest` は、どのマスからも最短経路に沿って進む。マス 0 とマス 9 では 2 つの移動が同じ最短手数で並ぶため、`argmax` は先頭にある**下**を選ぶ。

滑りが入ると 1 つの点が変わる。滑ることによってエージェントが元のマスへ押し戻されたり、

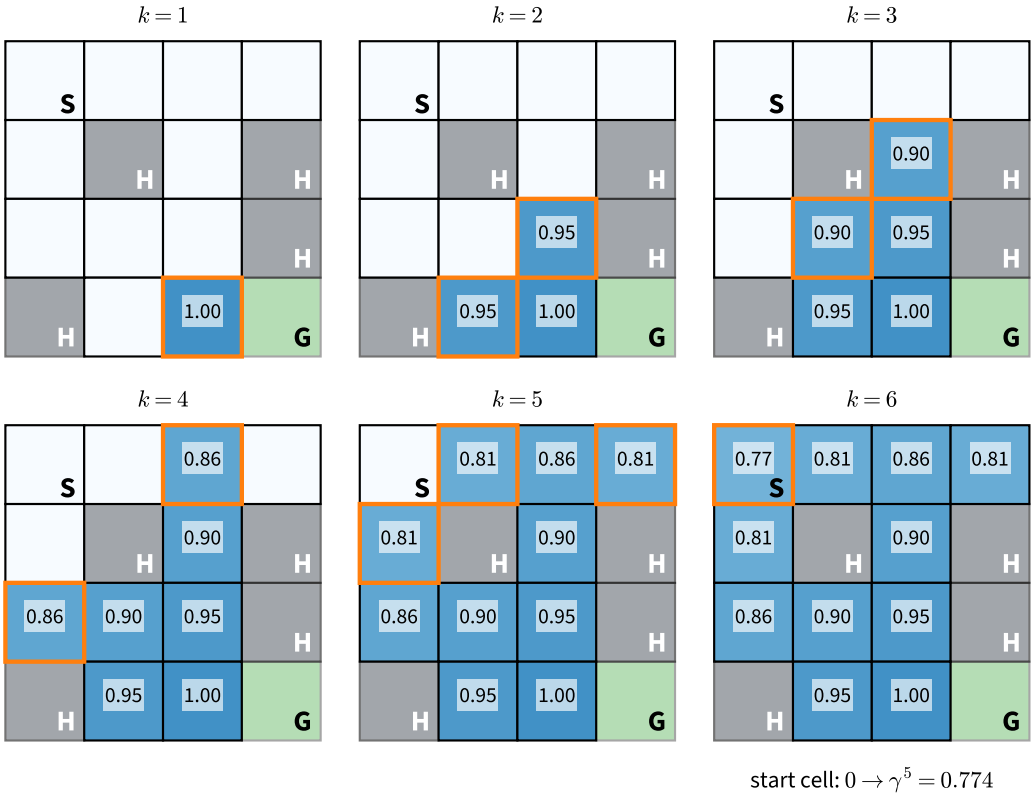


Figure 14.7: 価値が閉じた式で表される滑らない FrozenLake における価値反復法。スイープ k はゴールから k 手以内のマスに到達し（オレンジ色の枠線が新たに到達したマスを表す）、ゴールから j 手のマスは価値 γ^{j-1} をもつ。スタートのマスは 6 手離れているため、その推定値は 5 回のスイープの間 0 のままであり、6 回目以降は $\gamma^5 = 0.774$ に等しくなる。

先ほどのマス 14 のようにその場にとどまったりするため、価値同士が互いにフィードバックし合うのである。そうなると有限回のスイープでは厳密な価値に到達しないため、そもそも反復が収束するのか、どれほど速く収束するのか、そしていつ停止すべきかを問わなければならない。

14.2.2.3 縮小写像による収束

これら 3 つの問いへの答えはすべて、$\mathcal{T}$ がもつ 1 つの性質、すなわち任意の 2 つの表を互いに近づけるという性質に基づいている。表どうしの距離は、状態全体での最大の食い違いを表す上限ノルム（sup norm）$\|V\|_\infty = \max_s |V(s)|$ で測る。

命題（ベルマン作用素は γ 縮小写像である）。任意の 2 つの表 $V, V' : \mathcal{S} \to \mathbb{R}$ に対して、次が成り立つ。

$$\|\mathcal{T}V - \mathcal{T}V'\|_\infty \le \gamma \, \|V - V'\|_\infty .$$

証明。 s を固定し、$f(a)$ と $g(a)$ でそれぞれ $(\mathcal{T}V)(s)$ と $(\mathcal{T}V')(s)$ の角括弧内の項を表す。確率は非負で総和が 1 であるため、すべての a について $|f(a) - g(a)| = \gamma \left| \sum_{s'} P(s' \mid s,a)(V(s') - V'(s')) \right| \le \gamma \|V - V'\|_\infty$ が成り立つ。2 つの最大値の差は入力の差の最大値で抑えられる、すなわち $\left|\max_a f(a) - \max_a g(a)\right| \le \max_a |f(a) - g(a)|$ であるため、$|(\mathcal{T}V)(s) - (\mathcal{T}V')(s)| \le \gamma \|V - V'\|_\infty$ がすべての s について成り立つ。■

したがって、どのような 2 つの表であっても、1 回のスイープによって最大の食い違いは少なくとも γ 倍以下に縮小する（contraction）。これを V_k と、V^*（$\mathcal{T}$ によって変化しない不動点）に適用すれば、価値反復法の誤差はスイープごとに少なくとも γ 倍以下に縮むことがわかる。Figure 14.8 は、表が平面上の 1 点として表される 2 状態の MDP でこの様子を示したものである。

ここから 4 つの帰結が導かれる。第 1 に、不動点は一意である。もし V と V' がともに不動点であれば $\|V - V'\|_\infty = \|\mathcal{T}V - \mathcal{T}V'\|_\infty \le \gamma \|V - V'\|_\infty$ となり、$\gamma < 1$ のもとでは $V = V'$ でなければならない。したがって V^* は最適方程式の唯一の解である。

第 2 に、価値反復法はどのような初期値から始めても幾何級数的な速さで収束する。$e_k = \|V_k - V^*\|_\infty$ で k 回のスイープ後の誤差を表そう。このとき次が成り立つ。

$$e_k = \|\mathcal{T}V_{k-1} - \mathcal{T}V^*\|_\infty \le \gamma \, e_{k-1} \le \cdots \le \gamma^k e_0 , \tag{14.20}$$

したがって、誤差が 10^{-6} を下回るのに必要なスイープ数は高々 $\log(e_0/10^{-6}) / \log(1/\gamma)$ 回である。γ が 1 に近いとき分母はおよそ $1-\gamma$ となり、必要なスイープ数は実効ホライズン（effective horizon）$1/(1-\gamma)$（14.1 節）に比例して増加する。

第 3 に、反復列そのものが自身の精度を保証する。誤差 e_k は求めようとしている V^* を含むため直接観測できないが、連続する反復間の変化量は観測可能であり、次のように誤差の上界を与える（演習問題 2 でこの上界を導出する）。

$$\|V_k - V^*\|_\infty \le \frac{\gamma}{1-\gamma} \, \|V_k - V_{k-1}\|_\infty . \tag{14.21}$$

右辺は直近の 2 つの反復値のみに依存するため、この値が目標の許容誤差を下回った時点で直ちに価値反復法を停止できる。

第 4 の帰結は方策に関するものである。$\mathcal{T}$ の最大値を固定した方策 π のもとでの平均に置き換えると作用素 $\mathcal{T}^\pi$ が得られ、これは表 V を期待方程式 Equation 14.13 の右辺（ただし V で V^π を置き換えた式）へと写す。上の証明において最大値を平均に置き換えれば、$\mathcal{T}^\pi$ もまた γ 縮小写像であ

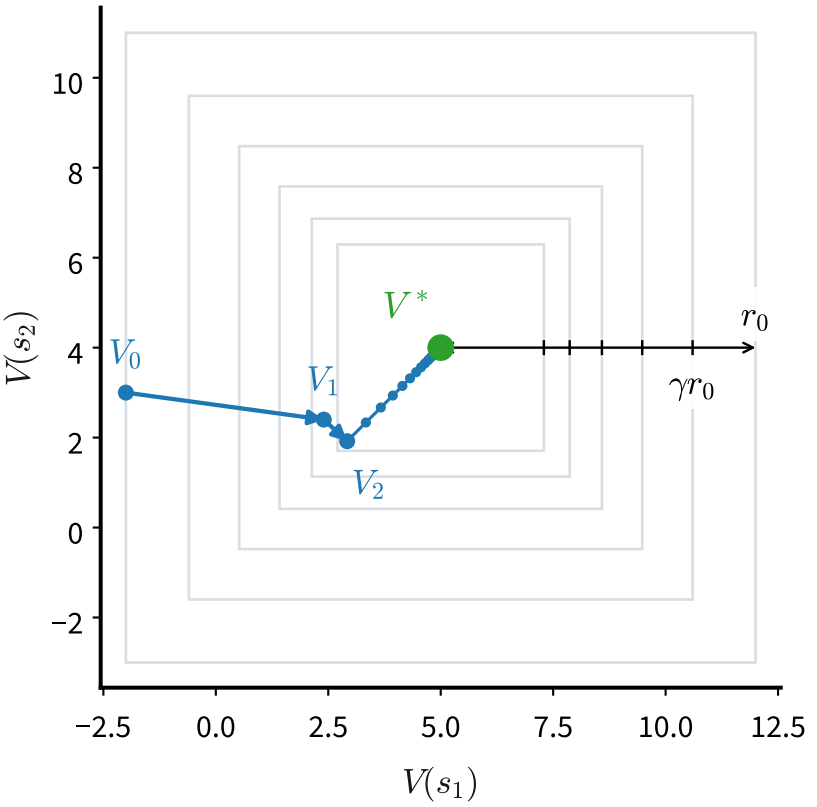


Figure 14.8: 価値反復法が収束するのは、ベルマン作用素 $\mathcal{T}$ が上限ノルムにおける縮小写像だからである。この MDP は 2 つの状態をもち、各状態でエージェントはとどまるか移動するかを選べる。s_1 にとどまると報酬 1 が得られ、それ以外の選択の報酬はすべて 0 であり、$\gamma = 0.8$ なので $V^* = (5, 4)$ となる。各正方形は半径 $\gamma^k r_0$（中心 V^*）の上限ノルム球を表し、ここで $r_0 = \|V_0 - V^*\|_\infty = 7$（初期点 $V_0 = (-2, 3)$ の場合）であり、スイープ k は k 番目の正方形の内側に入る。

ることがわかり、したがって V^π がその一意な不動点となる (Puterman 1994; Bertsekas 2025)。いま、π を V^* に関して貪欲な方策とする（Equation 14.16）。このとき $\mathcal{T}^\pi V^* = \mathcal{T} V^* = V^*$ が成り立つため、V^* は $\mathcal{T}^\pi$ の不動点、すなわち V^π に一致する。ゆえに、最大値を達成する行動が複数並んだときにどの行動を選んだとしても、V^* に対するあらゆる貪欲方策は最適であり、`argmax` によって安全に方策を取り出せる。

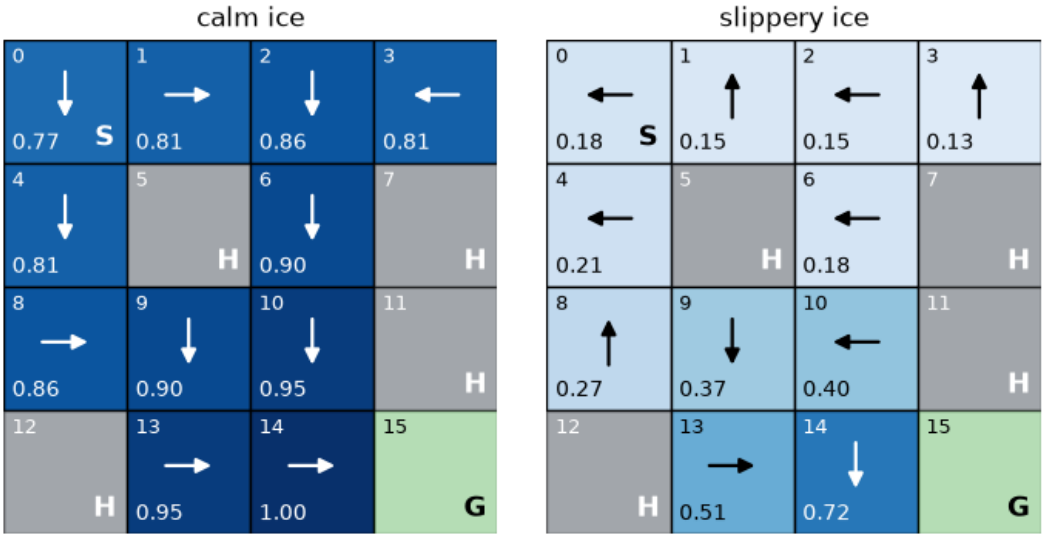
これら 4 つの帰結はすべて $\gamma < 1$ を必要とする。$\gamma = 1$ では、上の証明から言えるのは $\mathcal{T}$ が距離を増大させないことだけであり、一意性も収束性も成り立たなくなる場合がある（演習問題 3）。エピソードを決して終了させない 14.1 節の常に上を選ぶ方策がその例である。1 ステップあたり 0.05 の報酬があるとき、割引なしの価値はスイープのたびに 0.05 ずつ増え続け、決して収束しない。一方、すべての方策が確率 1 でエピソードを終了させる問題（確率的最短経路問題（stochastic shortest-path problem）と呼ばれる）では、終端状態の価値を 0 に固定すれば $\gamma = 1$ でも一意な解が保たれる (Bertsekas 2025)。

14.2.2.4 滑る氷

いよいよ滑る湖で価値反復法を実行しよう。1,000 回のスイープを行えば誤差上界 $\gamma^{1000} e_0$ は 10^{-22} 未満になるため、最後の反復値を V^* とみなし、その貪欲方策 π^* を Equation 14.16 によって取り出す。次のコードセルでは、計算した価値を用いてマス 14 の方程式を確かめ、マス 9 の問いに対する行動価値 $Q^*(9, \cdot)$ を出力し、両方の湖における価値と貪欲な指示を描画する。

```
history = value_iteration(mdp, num_iters=1000)
V_star = history[-1]
Q_star = mdp.backup(V_star)
pi_star = Q_star.argmax(axis=1)
print(f'V*(s0) = {V_star[0]:.4f}, V*(13) = {V_star[13]:.4f}')
print(f'state 14: V* = {V_star[14]:.4f};  1/3 + gamma (V*(13) + V*(14)) / 3 = '
      f'{1 / 3 + gamma * (V_star[13] + V_star[14]) / 3:.4f}')
print(f'cell 9: Q*(9, a) for a = {", ".join(ARROWS)}: {Q_star[9].round(3)}')
d2l.show_grid(env.unwrapped.desc, [V_calm, V_star], [shortest, pi_star],
              titles=['calm ice', 'slippery ice'])
```

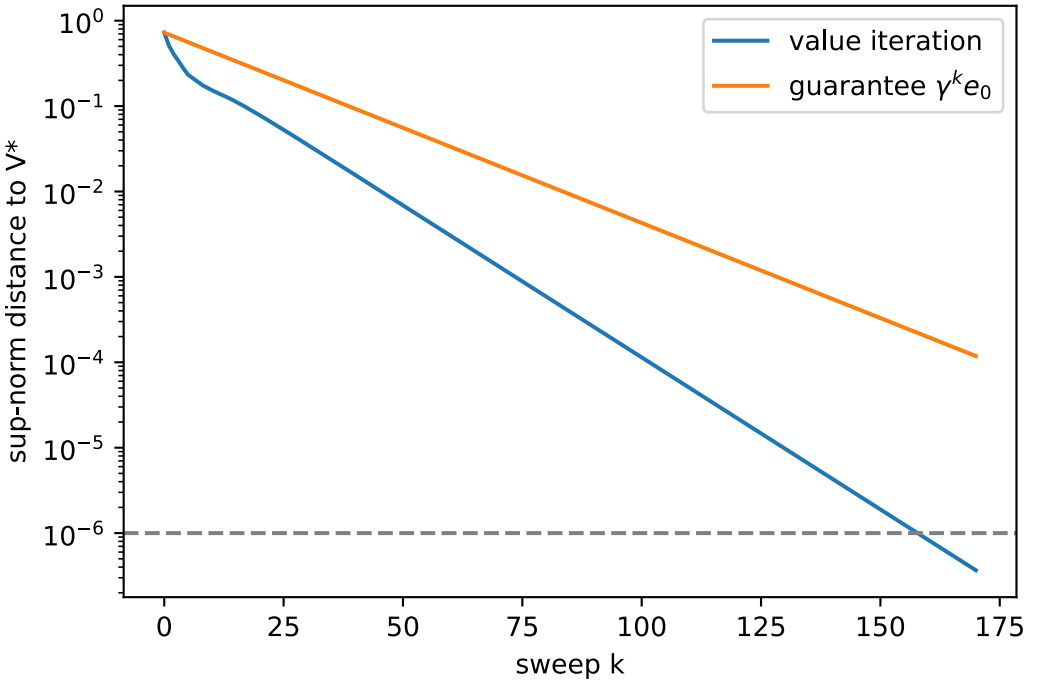
```
V*(s0) = 0.1805, V*(13) = 0.5090
state 14: V* = 0.7237;  1/3 + gamma (V*(13) + V*(14)) / 3 = 0.7237
cell 9: Q*(9, a) for a = <, v, >, ^: [0.247 0.375 0.289 0.213]
```



マス 14 の方程式は $V^*(13) = 0.509$ のもとで $\frac{1}{3} + 0.95\,(0.509 + 0.724)/3 = 0.724$ となり、両辺が同じ数値で一致する。マス 9 の問いにも答えが出た。**下**を指示する価値は 0.375 であるのに対し、穴に落ちる危険を伴う指示のうち良い方である**右**でも 0.289 にとどまる。すなわち、**下**の安全性には横に滑る不利を補って余りある価値がある。滑らない氷と比べると、滑りによってすべての価値が低下している。スタートのマスは 0.774 から 0.180 に下がり、ゴールの隣にあるマス 14 でさえ価値は 0.724 となって 1 には届かない。選ばれる指示も変化する。滑る氷での最適方策は、4 つのマスでゴールから遠ざかる向きを指し、別のいくつかのマスでは壁に向かって進む。それぞれの理由については 14.2.4 節で説明する。

スイープはどれほど速く収束したのだろうか。以下のコードセルでは、最初の 170 回のスイープにおける誤差 e_k を理論保証 $\gamma^k e_0$ と並べてプロットし（許容誤差 10^{-6} を破線で示す）、理論保証が決して破られないことをアサートで確かめる。

```
err = np.concatenate([[np.abs(V_star).max()],
                      np.abs(history - V_star).max(axis=1)])
k = np.arange(171)
assert (err[k] <= err[0] * gamma ** k + 1e-12).all()
d2l.plot_curves({'value iteration': err[k],
                 r'guarantee $\gamma^k e_0$': err[0] * gamma ** k},
                xlabel='sweep k', ylabel='sup-norm distance to V*',
                yscale='log', reference=1e-6)
```



対数スケール上では、幾何級数的な減衰は直線になる。理論保証は 1 スイープあたり $\gamma = 0.95$ の割合で減少し、$e_0 = V^*(14) = 0.724$ から出発してスイープ 264 で 10^{-6} に達する。実測された誤差はそれより速く、1 スイープあたり約 0.92 の割合で減少してスイープ 158 で破線を下回る。理論保証はこの割引率をもつあらゆる MDP に対して成り立つため、マスの数もゴールからの距離もその式には入らない（より速く収束する理由は演習問題 5 で解明する）。

14.2.2.5 停止規則

実際の実装では V^* が未知であるため、真の誤差を監視することはできず、観測可能な量に基づいて停止しなければならない。次のコードセルでは、観測可能な 2 つの規則を真の誤差と比較する。1 つはスイープ間の変化量 $\|V_k - V_{k-1}\|_\infty$ が 10^{-6} を下回ったときに停止する規則であり、もう 1 つは精度保証（certificate）Equation 14.21 がそれを下回ったときに停止する規則である。

```
gap = np.abs(np.diff(history, axis=0)).max(axis=1)
true_err = np.abs(history[1:] - V_star).max(axis=1)
certified = gamma / (1 - gamma) * gap
assert (true_err <= certified + 1e-12).all()
for name, e in [('sweep-to-sweep change', gap),
                ('certified error bound', certified),
                ('distance to V*', true_err)]:
    print(f'{name} first below 1e-6 at sweep {np.argmax(e <= 1e-6) + 2}')
print(f'distance to V* when the change first falls below 1e-6: '
      f'{true_err[np.argmax(gap <= 1e-6)]:.1e}')
k_cert = np.argmax(certified <= 1e-6) + 2
```

```
sweep-to-sweep change first below 1e-6 at sweep 128
certified error bound first below 1e-6 at sweep 164
```

```
distance to V* first below 1e-6 at sweep 158
distance to V* when the change first falls below 1e-6: 1.1e-05
```

変化量に基づく規則はスイープ 128 で停止する。その時点での真の誤差はまだ 1.1×10^{-5} と許容誤差の 11 倍以上も残っているが、この規則が観測できる量からはその事実を見抜けない。一方、精度保証に基づく規則はスイープ 164 で停止する。これは真の誤差が初めて 10^{-6} を下回るスイープ 158 からわずか 6 スイープ後であり、安全側に倒れているとはいえ、その差はごくわずかである。

14.2.3 方策評価と方策反復法

価値反復法は、スイープのたびに陰な形で方策を改善する。これに対し、第 2 の手法群は 2 つの作業を切り離す。すなわち、与えられた方策がどれほど良いかを測定し、そのうえで方策を改善するのである。この測定は、最適化して得たものではない任意の方策にスコアを付けられるため、それ単体でも有用である。

14.2.3.1 方策評価

スイープにおける最大値を方策自身の平均に置き換える、すなわち $\mathcal{T}^{\pi}$ を $\mathcal{T}$ の代わりに適用する。

$$V_{k+1}^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \Big[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\, V_k^{\pi}(s') \Big], \quad \text{for all } s \in \mathcal{S}. \tag{14.22}$$

これが**方策評価**（policy evaluation）である。V^{π} へと任意の初期値から収束するのは、$\mathcal{T}^{\pi}$ が γ 縮小写像だからである。コードは価値反復法のスイープにおいて最大値を `pi[s, a]` のもとでの平均に置き換えたものになり、ここでは一様ランダム方策に適用する。

```
def policy_evaluation(mdp, pi, num_iters):
    """Same sweep with the max replaced by an average under pi(a|s)."""
    V, history = np.zeros(mdp.num_states), []
    for _ in range(num_iters):
        V = (pi * mdp.backup(V)).sum(axis=1)
        history.append(V)
    return np.array(history)

uniform = np.ones((mdp.num_states, mdp.num_actions)) / mdp.num_actions
print(f'V(s0) for the uniformly random policy: '
      f'{policy_evaluation(mdp, uniform, 400)[-1][0]:.4f}')
```

```
V(s0) for the uniformly random policy: 0.0078
```

一様ランダム方策の初期状態価値は 0.0078 であり、最適値 0.180 の約 4% にすぎない。指示をランダムに選ぶ歩行者はめったにゴールへ到達せず、たまに着いたとしても多くの手数を要するからである。

最大値演算がなくなれば方程式は線形になり、表形式であれば直接解くことができる。$P_{\pi}(s' \mid s) = \sum_a \pi(a \mid s) P(s' \mid s, a)$ で π のもとでの遷移カーネル（s から s' へ、π から行動を引いたときに 1 ステップで移る確率）を表し、$r_{\pi}(s) = \sum_a \pi(a \mid s) r(s,a)$ でその期待報酬（expected

reward）を表す。すると Equation 14.13 は $V^\pi = r_\pi + \gamma P_\pi V^\pi$ と書け、その解は次のようになる。

$$V^\pi = (I - \gamma P_\pi)^{-1} r_\pi. \tag{14.23}$$

行列 $I - \gamma P_\pi$ は $\gamma < 1$ のとき正則である。γP_π があらゆるベクトルを上限ノルムにおいて縮小させるためである（演習問題 1 では、この逆行列を割引訪問回数として解釈する）。表形式であれば、連立一次方程式を 1 回解くだけで V^π が厳密に求まる。スイープはこれと同じ計算を 1 回の縮小ごとに進めるものであり、直接解くには大きすぎる問題へ拡張できる形式となっている。この直接解法を `TabularMDP` に追加しよう。

```
@d2l.add_to_class(d2l.TabularMDP)
def policy_value(self, pi):
    """V^pi = (I - gamma P_pi)^{-1} r_pi exactly, for pi[s, a] the
    probability of action a at state s (one-hot rows: deterministic).
    Needs gamma < 1 here: from_gym's terminal states loop on themselves."""
    P_pi = np.einsum('sa,sat->st', pi, self.P)
    r_pi = (pi * self.r).sum(axis=1)
    return np.linalg.solve(np.eye(self.num_states) - self.gamma * P_pi,
                           r_pi)

print(f'V(s0) for the uniformly random policy, solved exactly: '
      f'{mdp.policy_value(uniform)[0]:.4f}')
```

```
V(s0) for the uniformly random policy, solved exactly: 0.0078
```

どちらの方法でも 0.0078 が得られる。`einsum` は遷移カーネルの各行を行動について方策の確率で平均することで P_π を構成する。これらの配列上で直接解法を使うには $\gamma < 1$ が必要である。`from_gym` は各終端状態をどの行動でも自分自身へ戻る状態として格納するため、$\gamma = 1$ では行列 $I - P_\pi$ の対応する行がゼロ行となって特異になるからである。したがって、ゴール到達確率のような $\gamma = 1$ における量を求めるときはスイープを用いる。

14.2.3.2 方策反復法と一般化方策反復

方策評価と貪欲ステップによる方策改善（policy improvement）を組み合わせることで、Howard (1960) による別のアルゴリズムが導かれる。現在の方策を厳密に評価し、その価値に関して貪欲に行動し、これを繰り返すという手法である。貪欲ステップは、本節の冒頭で示した不等式 $\max_a Q^\pi(s,a) \geq V^\pi(s)$ をすべての状態に一斉に適用することに基づいている。

命題（方策改善）。 *π を任意の方策とし、π' を V^π に関して貪欲な方策、すなわち $\pi'(s) = \operatorname{argmax}_a \left[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^\pi(s') \right]$ とする。このときすべての状態で $V^{\pi'}(s) \geq V^\pi(s)$ が成り立ち、もしすべての状態で等号が成立するならば両方策とも最適である。*

証明。 貪欲性より、1 ステップだけ π' に従いその後を π の価値で評価したものは、少なくとも 1 ステップ π に従った場合以上に良い。すなわち $\mathcal{T}^{\pi'} V^\pi = \mathcal{T} V^\pi \geq \mathcal{T}^\pi V^\pi = V^\pi$ である。作用素 $\mathcal{T}^{\pi'}$ は係数が確率であるため単調であり、すべての状態で $V \leq U$ ならば $\mathcal{T}^{\pi'} V \leq \mathcal{T}^{\pi'} U$ が成り立つ。これを繰り返し適用すると $V^\pi \leq \mathcal{T}^{\pi'} V^\pi \leq (\mathcal{T}^{\pi'})^2 V^\pi \leq \cdots$ が得られ、この列は $\mathcal{T}^{\pi'}$ の一意な不動点である $V^{\pi'}$ に収束する。もし $V^{\pi'} = V^\pi$ であれば最初の式から $\mathcal{T} V^\pi = V^\pi$ となるため、V^π は最適方程式を満たす。その唯一の解は V^* であるから、π は最適である。■

したがって、**方策反復法**（policy iteration）の各ラウンドは、いずれかの状態で価値を引き上げるか、すべての状態の価値を変えないかのどちらかであり、後者の場合はその時点で方策がすで

に最適となっている。価値は決して減少しないため、最適解に到達する前に同じ方策が再び現れることはなく、有限 MDP における決定論的な方策は有限個しかないため、方策反復法は有限回のラウンドで終了する (Puterman 1994)。終了判定には上の命題の等号成立条件、すなわち貪欲ステップがいかなる状態でも価値を改善しないという条件を用いる。代わりに貪欲方策が変化しなくなったかどうかを調べると、滑る湖のように同値が生じるケースで破綻することがある。マス 6 では**左**と**右**が同じ行動価値をもち、丸め誤差（rounding error）によってラウンドごとに `argmax` の結果が両者の間で入れ替わりうるからである。以下のループは一様方策から始め、各方策を `policy_value` で評価し、どの行動も現在の価値を丸め誤差以上には上回らなくなった時点で停止する。

```
pi, num_outer = uniform, 0
while True:
    V = mdp.policy_value(pi)                          # evaluate pi exactly
    Q = mdp.backup(V)
    num_outer += 1
    if (Q.max(axis=1) <= V + 1e-12).all():            # greedy gains nothing:
        break                                         # pi is optimal
    pi = np.eye(mdp.num_actions)[Q.argmax(axis=1)]    # improve
assert np.abs(V - V_star).max() < 1e-12
```

上のアサートは、方策反復法と価値反復法が小数点以下 12 桁まで V^* で一致することを確かめている。次のコードセルでは、それぞれが要した計算量を数える。1 回のバックアップは 1 つの状態行動対に対する角括弧項の 1 回の評価にあたるため、価値反復法の 1 スイープには $|\mathcal{S}||\mathcal{A}| = 64$ 回のバックアップというコストがかかる。

```
ok = [(mdp.backup(V_k).argmax(axis=1) == pi_star).all() for V_k in history]
assert all(ok[np.argmax(ok):])     # once optimal, the greedy policy stays so
first = mdp.backup(mdp.policy_value(uniform)).argmax(axis=1)
per_sweep = mdp.num_states * mdp.num_actions          # backups in one sweep
print(f'policy iteration: {num_outer} exact solves; the first greedy step '
      f'gives pi*: {(first == pi_star).all()}')
print(f'value iteration: certified at sweep {k_cert} '
      f'({k_cert * per_sweep:,} backups)')
print(f'its greedy policy already equals pi* from sweep {np.argmax(ok) + 1} '
      f'on ({(np.argmax(ok) + 1) * per_sweep:,} backups)')
```

```
policy iteration: 2 exact solves; the first greedy step gives pi*: True
value iteration: certified at sweep 164 (10,496 backups)
its greedy policy already equals pi* from sweep 14 on (896 backups)
```

方策反復法が要した厳密解法は 2 回だけであった。一様方策の価値に対する貪欲方策はすでに π^* となっており、2 回目のラウンドは貪欲ステップで何も改善しないことを確認しているにすぎない。一方、価値反復法は価値の精度保証を得るまでに 10,496 回のバックアップを要したが、その貪欲方策は 14 スイープ（896 回のバックアップ）の時点ですでに π^* に一致していた。方策はどの行動価値が最大かだけに依存するため、価値そのものが収束するよりずっと早く確定しうるのである。方策反復法の各ラウンドは未知数 16 個の連立一次方程式を解くため、この問題では低コストで済むが、状態数が多い問題では直接解法にかかる 3 乗オーダーのコストゆえにスイープの方が好まれる（演習問題 8 で両者のコストを比較する）。

評価と改善は、それぞれ完了するまで交互に行う必要はない。価値を V^π に近づけるステップと、方策を貪欲方策に近づけるステップを織り交ぜるあらゆる枠組みを総称して**一般化方策反復**（generalized policy iteration）と呼ぶ (Sutton and Barto 2018)。方策反復法は各評価を完了さ

せてから改善を行い、価値反復法は評価を 1 スイープ行っただけで改善に移る。経験から学習する手法は、モデルの代わりにサンプリングされた遷移から計算した、同じ 2 種類のより小さくノイズを含んだステップを進める。Figure 14.9 にそのような 4 つの枠組みを示す。

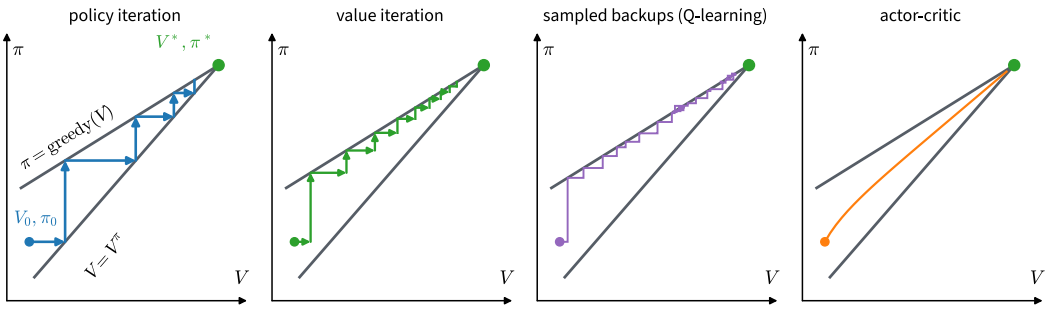


Figure 14.9: 一般化方策反復。直線 $V = V^\pi$ は方策に対して厳密な価値推定値の集合を、直線 $\pi = \text{greedy}(V)$ は価値推定値に対して貪欲な方策の集合を表し、両者が同時に成り立つのは (V^*, π^*) のみである。同じ初期点から出発して、方策反復法は改善のたびに最初の直線に到達するまで評価しきり、価値反復法はそこへ向けて 1 スイープだけ進み、標本バックアップは短くノイズを含んだステップをとり、方策とその価値推定値を同時に更新する手法は両方の座標を一度に少しずつ動かす。各軸は価値関数と方策の空間を表す模式図である。

14.2.4 滑る氷の上での最適方策

価値を見れば、π^* が最良の方策であることがわかる。では、明白な代替案である滑らない氷での最短経路と比べてどれほど優れているのか、またなぜ 4 つのマスでゴールから遠ざかる向きを指すのだろうか。この 2 つの問いにモデルから厳密に答え、さらにシミュレーションによってその結果を確認しよう。

14.2.4.1 モンテカルロ評価

モデルを使わない検証方法として、シミュレータ内で方策を実行し、得られた収益の平均をとるやり方がある。そのために、方策を観測（observation）から行動への関数 `policy(obs, rng)` として表す。方策が確率的であれば乱数生成器 `rng` から行動をサンプリングする。関数 `evaluate` は、指定したエピソード数にわたる平均割引収益を返す。

```
def evaluate(env, policy, num_episodes, gamma=1.0, rng=None):
    """Mean discounted return of an acting policy over Monte Carlo episodes.

    A policy is any function `policy(obs, rng) -> action`; a deterministic
    policy ignores `rng`, a stochastic one draws from it."""
    total = 0.0
    for _ in range(num_episodes):
        obs, done, discount = env.reset()[0], False, 1.0
        while not done:
            obs, reward, terminated, truncated, _ = env.step(policy(obs, rng))
            done = terminated or truncated
            total, discount = total + discount * reward, discount * gamma
    return total / num_episodes
```

変数 discount は現在のステップに対する γ^t を保持し、エピソードは terminated または truncated のいずれかで終了する。デフォルトの $\gamma = 1$ と、ゴールに入ったときにのみ与えられる湖の報酬のもとでは、平均収益はシミュレータの 100 ステップ制限内にゴールへ到達したエピソードの割合に一致する。

14.2.4.2 成功率

同じ確率はモデルからも計算できる。$\gamma = 1$ において、k 回のスイープによる方策評価（初期値 $V_0 = 0$）は、最初の k ステップの期待報酬を足し合わせる。報酬はゴールに入ったときにしか得られないため、k 番目の反復値の s_0 における成分は k ステップ以内にゴールへ到達する確率となり、$k = 100$ とすればシミュレータの制限と一致する。$\gamma = 1$ のモデルを lake1 として構築し、π^*、shortest、および一様方策のそれぞれについて到達確率を計算し、2,000 エピソードのモンテカルロ推定を並べて出力しよう。

```
lake1 = d2l.TabularMDP.from_gym(env, 1.0)    # gamma = 1: V(s0) = P(goal)
rng = np.random.default_rng(0)
env.reset(seed=0)
for name, p in [('slip-aware optimum', np.eye(4)[pi_star]),
                ('shortest-path policy', np.eye(4)[shortest]),
                ('uniformly random policy', uniform)]:
    exact = policy_evaluation(lake1, p, 100)[-1][0]   # within 100 steps
    mc = evaluate(env, lambda s, rng: int(rng.choice(4, p=p[s])), 2000,
                  rng=rng)
    print(f'{name:>23}: goal within 100 steps {exact:.3f} exactly, '
          f'{mc:.3f} in 2000 episodes')
```

```
     slip-aware optimum: goal within 100 steps 0.730 exactly, 0.736 in 2000 episodes
   shortest-path policy: goal within 100 steps 0.045 exactly, 0.046 in 2000 episodes
uniformly random policy: goal within 100 steps 0.014 exactly, 0.010 in 2000 episodes
```

最適方策（optimal policy）が 100 ステップ以内にゴールへ到達する確率は 0.730 であり、最短経路方策ではその約 1/16 にあたる 0.045、ランダムな歩行者では 0.014 となる。いずれのモンテカルロ推定も、厳密な値との差はサンプリングノイズで説明できる範囲に収まっている。2,000 エピソードにおける標準誤差（standard error）は、0.73 付近で約 1 パーセントポイント、0.014 付近で約 0.25 ポイントだからである。なお、制限時間によって π^* はわずかに不利を受けている。慎重な指示ゆえにエピソードが長引き、100 ステップを超えればゴールに到達できたはずの一部のエピソードが途中で打ち切られるためである。

14.2.4.3 ゴールに背を向ける指示

なぜ最短経路はこれほどひどく失敗するのだろうか。エージェントが行動する 11 個の氷のマスを 1 つずつ調べ、2 つの点を確認しよう。すなわち、π^* の指示がどちらを向いているか、そして各方策の指示によって次のステップでエージェントが穴に落ちる確率がどれほどあるかである。向きは滑らない氷での価値から読み取る。滑らない氷における遷移先のマスがより大きな価値 V_calm をもてばゴールに向かう移動、遷移先が元のマス自身であれば壁に向かう移動、それ以外はゴールから遠ざかる移動とする（穴もゴールも滑らない氷での価値が 0 であるため、そこへ入る移動も遠ざかる移動に分類される）。配列 risk[s, a] は 1 ステップで穴に落ちる確率 $\sum_{s' \text{ a hole}} P(s' \mid s, a)$ を保持する。

```
    desc = env.unwrapped.desc.flatten()
    hole, end = desc == b'H', np.isin(desc, [b'H', b'G'])
    risk = mdp.P[:, :, hole].sum(axis=-1)       # risk[s, a] = P(hole next | s, a)
    move = calm.P[np.arange(16), pi_star] @ V_calm - V_calm   # on calm ice
    for s in np.flatnonzero(~end):
        kind = ('toward the goal' if move[s] > 1e-9 else
                'into a wall' if move[s] > -1e-9 else 'away from the goal')
        print(f's={s:>2}: pi* {ARROWS[pi_star[s]]} {kind:<18}  hole risk: pi* '
              f'{risk[s, pi_star[s]]:.2f}, shortest {risk[s, shortest[s]]:.2f}, '
              f'least {risk[s].min():.2f}')
```

```
s= 0: pi* < into a wall         hole risk: pi* 0.00, shortest 0.00, least 0.00
s= 1: pi* ^ into a wall         hole risk: pi* 0.00, shortest 0.33, least 0.00
s= 2: pi* < away from the goal  hole risk: pi* 0.00, shortest 0.00, least 0.00
s= 3: pi* ^ into a wall         hole risk: pi* 0.00, shortest 0.33, least 0.00
s= 4: pi* < into a wall         hole risk: pi* 0.00, shortest 0.33, least 0.00
s= 6: pi* < away from the goal  hole risk: pi* 0.33, shortest 0.67, least 0.33
s= 8: pi* ^ away from the goal  hole risk: pi* 0.00, shortest 0.33, least 0.00
s= 9: pi* v toward the goal     hole risk: pi* 0.00, shortest 0.00, least 0.00
s=10: pi* < away from the goal  hole risk: pi* 0.00, shortest 0.33, least 0.00
s=13: pi* > toward the goal     hole risk: pi* 0.00, shortest 0.00, least 0.00
s=14: pi* v into a wall         hole risk: pi* 0.00, shortest 0.00, least 0.00
```

最適方策がゴールへ向かって進むのは、マス 9 とマス 13 の 2 マスだけである。マス 0、1、3、4、14 の 5 マスでは、壁に向かって進むよう指示する。このような指示を出すと、本来の移動はその場にとどまる結果となり、実際に移動が生じるのは 2 方向の滑りだけになる。たとえばマス 4 では、**左**を指示するとエージェントは壁に沿って上下に滑るだけで、右にあるマス 5 の穴へ入ることは決してない。マス 14 では、**下**も**右**も 3 回中 1 回はゴールに到達し、3 回中 1 回はマス 14 にとどまる。3 つ目の結果は、**下**ならマス 13、**右**ならマス 10 への移動であり、マス 10 の方が価値が低い（先ほどのグリッドで 0.40 対 0.51）。

マス 2、6、8、10 の 4 マスでは、最適方策はゴールから遠ざかる向きを指す。マス 8 とマス 10 では、ゴールへ向かう指示はいずれも穴（マス 8 の下にある 12 や、マス 10 の隣にある 11）へ落ちる可能性がある。これに対し、マス 8 での**上**とマス 10 での**左**は穴に落ちることがなく、しかもマス 10 での**左**は 3 回中 1 回の滑りでエージェントを下方のマス 14 へと運ぶ。マス 6 では両脇に穴があるため、どの指示にも穴に落ちる危険が伴う。**左**と**右**の危険度は 1/3、**上**と**下**の危険度は 2/3 であり、π^* は最も危険の少ない同値の 2 つの指示の一方を選ぶ。`argmax` は**左**を選ぶが（**右**ならゴールへ向かう向きになる）、これは 3 回中 1 回そのまま穴 5 へ入る移動をあえて指示している。**上**や**下**では 2 方向の滑りが両方とも穴に入るのに対し、**左**なら 2 方向の滑りがどちらも安全なマスに着地するからである。マス 2 ではどの指示にも穴の危険はないが、**左**が選ばれるのは、最上段で最も価値が低くどの経路も結局マス 2 を通り直すことになる角のマス 3 へ滑る可能性がないためである。

11 個すべてのマスにわたって、最適な指示の危険度は選べる指示の中で最小となっており、マス 6 以外のすべてのマスで 0 である。一方、最短経路方策はマス 1、3、4、8、10 で 1/3 の危険を冒し、マス 6 では 2/3 の危険を冒す。ただし、これらの 1 ステップの危険度だけではエピソードがどこで終わるかはまだわからない。各マスが何回訪問されるかにも依存するからである。決定論的な方策 π に対し、P_π を π のもとでの遷移カーネルのうち終端状態の行を 0 に置き換えてそこで過程が停止するようにしたものとする。ここで調べている両方策はいずれも確率 1 でエピソードを終了させるため $I - P_\pi$ は正則であり、$s_0 = 0$ から出発したときの各状態への期待訪問回数は $(I - P_\pi)^{-1}$ の s_0 に対応する行をなす。関数 `falls` は転置した連立一次方程式を解くことでこれを求める。各マスでの期待訪問回数にそのマスでの指示の穴への危険度を掛けると、そのマス

から穴に落ちてエピソードが終わる確率が得られる。

```
def falls(mdp, pi, risk, end):
    """P(the episode falls into a hole from s), for each s, without a time
    limit: expected visits to s before the end, times the hole risk at s."""
    S, I = np.arange(mdp.num_states), np.eye(mdp.num_states)
    P_pi = mdp.P[S, pi] * ~end[:, None]               # stop at the end
    visits = np.linalg.solve(I - P_pi.T, I[0])        # starting from s0 = 0
    return visits * risk[S, pi] * ~end

for name, p in [('pi*', pi_star), ('shortest', shortest)]:
    f = falls(mdp, p, risk, end)
    print(f'{name:>8}: P(falls into a hole) {f.sum():.2f}, from ' +
          ', '.join(f'{s}: {f[s]:.2f}' for s in np.argsort(-f) if f[s] > 0))
```

```
    pi*: P(falls into a hole) 0.22, from 6: 0.22
shortest: P(falls into a hole) 0.96, from 4: 0.31, 1: 0.31, 8: 0.13, 6: 0.08, 3: 0.06, 10: 0.05
```

時間制限がなければ、最適方策が穴に落ちる確率は 0.22（そのすべてがマス 6 で生じる）であり、残りの 78% のエピソードではゴールに到達する（100 ステップの制限によってこれが先ほどの 0.730 まで下がる）。一方、最短経路方策はエピソードの 96% で穴に落ちる。落下の約 1/3 は本来の経路上のマス 4 から生じるが、さらに別の 1/3 は経路上にまったくないマス 1 から生じる。マス 0 での最初の指示である**下**が 3 回中 1 回の割合でエージェントを右のマス 1 へ滑らせ、そこで `shortest` が指示する**右**が 3 回中 1 回の割合で下のマス 5 にある穴へと滑り落ちるのである。残りの落下はマス 8、6、3、10 から生じる。最短経路方策は、本来の経路そのものよりも、滑って飛ばされた先のマスでより多く失敗しており、対する最適方策はいずれかの指示で穴を避けられるすべてのマスにおいて確実に穴を避けている。

14.2.5 報酬シェイピングと最適方策

14.1 節では、報酬が疎な湖で学習器を助けるため、ゴールに近づく移動のたびに 0.3 のボーナス（マンハッタン距離 $\ell(s)$ で判定）を与える案を取り上げ、滑らない氷での計算によってそのボーナスが堂々巡りに報酬を与えてしまうことを示した。価値反復法と厳密な方策評価を手にした今、このボーナスが滑る氷の上での最良の方策に何をもたらすかを計算でき、さらに前節で提案したポテンシャル形式が最良の方策を変化させないことも証明できる。

14.2.5.1 接近ボーナス

次の状態について平均をとると、このボーナスは期待報酬に「ゴールへ近づく確率の 0.3 倍」を加える。

$$\tilde{r}(s,a) = r(s,a) + 0.3 \sum_{s'} P(s' \mid s,a)\, \mathbf{1}\big(\ell(s') < \ell(s)\big).$$

次のコードセルでは、ボーナス 0.3、0.1、0 に対してこの報酬を構成し、報酬シェイピング（reward shaping）を施した各 MDP の最適方策を価値反復法で求めたうえで、その方策を元の報酬のもとで `policy_value` により評価する。ボーナス 0 の行が元の問題に対応する。

```
row, col = np.divmod(np.arange(16), 4)
dist = abs(row - 3) + abs(col - 3)     # l(s): grid steps to the goal at (3, 3)
```

```
closer = dist < dist[:, None, None]    # closer[s, :, s'] = 1 iff l(s') < l(s)

for bonus in [0.3, 0.1, 0.0]:
    shaped = d2l.TabularMDP(mdp.P, mdp.r + bonus * (mdp.P * closer).sum(-1),
                            gamma)
    V_shaped = value_iteration(shaped, num_iters=500)[-1]
    pi_shaped = shaped.backup(V_shaped).argmax(axis=1)
    V_true = mdp.policy_value(np.eye(mdp.num_actions)[pi_shaped])
    print(f'bonus {bonus}: at s=14 goes {ARROWS[pi_shaped[14]]}, shaped V(0) '
          f'{V_shaped[0]:.2f}, V(13) {V_shaped[13]:.2f}, V(14) '
          f'{V_shaped[14]:.2f}; true V(0) {V_true[0]:.3f}')
```

```
bonus 0.3: at s=14 goes <, shaped V(0) 2.15, V(13) 1.71, V(14) 1.52; true V(0) 0.000
bonus 0.1: at s=14 goes v, shaped V(0) 0.79, V(13) 0.87, V(14) 0.94; true V(0) 0.115
bonus 0.0: at s=14 goes v, shaped V(0) 0.18, V(13) 0.51, V(14) 0.72; true V(0) 0.180
```

ボーナスが 0.3 のとき、シェイピング後の最適方策はマス 14 で**左**を指示する。その遷移先はマス 13、10、14 のいずれかであり、決してゴールには入らない。シェイピング後の報酬のもとでの初期状態価値は 2.15 と、元の最適方策の 0.180 を大きく上回るが、決してゴールに到達しないため元の報酬のもとでの価値は 0 である。シェイピング後の価値を見れば理由がわかる。シェイピング後の報酬では、マス 13 の価値がマス 14 の価値を上回っている（1.71 対 1.52）。13 から 14 へ向かう移動でボーナスが得られ、しかも 14 からはいつでも 13 に戻って再びボーナスを稼げるからである。これは 14.1 節で見たループに具体的な価値が付いた姿にほかならない。マス 13 とマス 14 の間を往復すれば 1 周ごとにボーナスが得られ、ゴールに入って 1 度きり得られる報酬とボーナスの合計よりも価値が高くなるのである。ボーナスが 0.1 であればループの価値はゴールを下回り、シェイピング後の最適方策は再びゴールを目指すようになるが、その真の価値は 0.115 となって元の 0.180 には届かず、元とは異なる劣った方策になってしまう。一見もっともらしいボーナスが最適方策を完全に変えてしまい、ボーナスを小さくしてもなお方策は変わってしまうのである。

14.2.5.2 ポテンシャルに基づくシェイピング

14.1 節で代わりに提案したボーナスは、次のようなポテンシャルの差である。

$$\tilde{r}(s, a, s') = r(s, a) + \gamma\Phi(s') - \Phi(s), \tag{14.24}$$

ここで Φ は状態上の有界な関数である。ループを一周すると各項は割引率の分を除いて打ち消し合い、次の命題が示すように、どの方策の価値も同じ量だけ変化する。

命題（ポテンシャルに基づくシェイピング）。$0 \leq \gamma < 1$ とし、`TabularMDP` と同様に終端状態は吸収状態であるとし、$\tilde{V}^\pi$ と $\tilde{Q}^\pi$ をシェイピング後の報酬のもとでの価値とする。このとき、任意の方策 π、状態 s、行動 a に対して次が成り立つ。

$$\tilde{V}^\pi(s) = V^\pi(s) - \Phi(s), \qquad \tilde{Q}^\pi(s, a) = Q^\pi(s, a) - \Phi(s). \tag{14.25}$$

特に $\tilde{V}^* = V^* - \Phi$ および $\tilde{Q}^*(s, a) = Q^*(s, a) - \Phi(s)$ が成り立ち、シェイピング後の MDP と元の MDP は同じ最適方策をもつ (Ng et al. 1999)。

証明。任意の軌跡に沿って、最初の T ステップのシェイピング項は望遠鏡和となって相殺し合う。

$$\sum_{t=0}^{T-1} \gamma^t \big(\gamma \Phi(s_{t+1}) - \Phi(s_t)\big) = \gamma^T \Phi(s_T) - \Phi(s_0). \tag{14.26}$$

$T \to \infty$ のとき、Φ が有界かつ $\gamma < 1$ であるため第 1 項は 0 に収束する。したがって、あらゆる軌跡のシェイピング後の収益は元の収益から $\Phi(s_0)$ を引いたものになる。π のもとで s から出発する期待値をとれば第 1 の等式が得られ、(s, a) から出発する期待値をとれば第 2 の等式が得られる。各状態におけるシフト量 $\Phi(s)$ はすべての方策とすべての行動に対して同一であるため、方策間の順序も $Q^*(s, \cdot)$ を最大化する行動も変わらない。■

このようにポテンシャルはすべての価値を変化させるが、どの方策も同じだけ変化を受けるため、最適化が用いる唯一の情報である方策間の大小関係は変わらない。演習問題 7c では、湖の上で $\tilde{V}^* = V^* - \Phi$ を確かめる。Ng et al. (1999) はその逆も証明している。追加する項 $F(s, a, s')$ がポテンシャルの差でないならば、同じ状態集合と行動集合の上に、シェイピング後の MDP のいかなる最適方策も元の MDP では最適とならないような遷移カーネルと報酬が存在する。接近ボーナスはそのような項の一例であり、滑る湖はそのような MDP の一例である。

ポテンシャルが最適方策を変えないのだとすれば、なぜそれを使うのだろうか。厳密なプランニングにおいては使う理由がない。価値反復法はポテンシャルの有無にかかわらず同じ方策を見つけるからである。その目的は、サンプリングされた遷移しか観測できない学習器を助けることにある。$\Phi = -0.3\,\ell$ のようにゴールへ向かって高くなるポテンシャルを導入すると、ゴールの報酬が経路全体に分散されるため、学習器は最後の 1 手だけでなく毎回の移動で信号を受け取れるようになる。

上の証明では `TabularMDP` の 1 つの性質を利用した。すなわち、終端状態が吸収状態であるため軌跡が永遠に続き、極限で項 $\gamma^T \Phi(s_T)$ が消えるという性質である。これに対し、シミュレータは最初の終端状態に到達した時点でエピソードを終了させる。もしそれが T ステップ後に起きたとすれば、そのエピソードのシェイピング後の収益は元の収益に $\gamma^T \Phi(s_T) - \Phi(s_0)$ を加えたものになる。項 $\gamma^T \Phi(s_T)$ はエピソードがどの終端状態でいつ終わったかに依存するため、すべての終端状態で Φ が 0 でない限り、どの方策が最良かを変えてしまうおそれがある。これが 14.1 節で言葉によって述べた条件である。ポテンシャル $\Phi = -0.3\,\ell$ はゴールでは 0 だが、穴では負の値をとる。そのためシミュレータ内では、穴に落ちることに対して（しかも遅く落ちるより早く落ちる方に大きく）元の課題には含まれないペナルティを課すことになってしまう。

本節の内容はすべて遷移表 P を参照していた。すなわち、どのバックアップも厳密な確率を用いて次の状態についての平均を計算していた。これで、既知の有限 MDP におけるすべての状態とすべての指示の価値を計算し、精度保証付きで最適方策を求め、報酬の変更がそれにどう影響するかを調べられるようになった。しかし、学習するエージェントに遷移表が与えられることはめったにない。エージェントが観測できるのは、自身の行動が生み出した 1 回ずつの遷移だけである。14.3 節では、そのようなサンプルだけから価値と最適方策を学習できるかどうかを問う。

14.2.6 まとめ

動的計画法は、既知の MDP における最良の方策をその状態の価値から計算する。価値 $V^\pi(s)$ は s から π のもとで得られる期待割引収益であり、ベルマン方程式は各状態の価値を後続状態の価値と結びつける（固定した方策では行動についての平均により、最適な方策では最大値による）。$\gamma < 1$ のときベルマン作用素 $\mathcal{T}$ は γ 縮小写像であるため、価値反復法は任意の初期値から一意な V^* へと幾何級数的に収束し、スイープ間の変化量がその誤差を保証する。また方策反復法は、1 回の連立一次方程式の求解による厳密な評価と貪欲な改善とを交互に行う。滑る湖において最適方策が最短経路の約 16 倍の頻度でゴールに到達するのは、たとえゴールから遠ざかる向きであっても、すべてのマスで穴に落ちる危険が最小の指示を選ぶからである。ゴールへ近づくことに

対して支払われるボーナスは最適方策を変えうるが、ポテンシャルの差として与えられるボーナスが最適方策を変えることはない。

14.2.7 演習問題

1. [code] **連立一次方程式としての方策評価**。@eq-policy-value-solve より、$V^\pi = M_\pi r_\pi$（ただし $M_\pi = (I - \gamma P_\pi)^{-1}$）である。
   1. $I - \gamma P_\pi$ が $0 \le \gamma < 1$ に対して正則であり、$M_\pi = \sum_{t \ge 0} \gamma^t P_\pi^t$ となることを示せ。
   2. 成分 $M_\pi(s, s')$ を、s' への訪問回数（問題 4b（14.1 節）の終了する過程において s から出発して π に従うエージェントによる訪問）の観点から解釈せよ。これより M_π の各行の和が $1/(1-\gamma)$ となり、したがって $(1-\gamma) M_\pi(s, \cdot)$ が状態上の分布、すなわち π の（s から出発した）**割引状態訪問分布**になることを導け。さらに $V^\pi(s)$ を、$1/(1-\gamma)$ とこの分布のもとでの r_π の期待値との積として表せ。
   3. 滑る湖における方策 `uniform` に対して M_π を計算し、$M_\pi r_\pi$ が初期状態の価値として 0.0078（`policy_evaluation` の出力）を再現することを確かめよ。この方策のエピソードは平均して 8 ステップ未満で終わるにもかかわらず、初期状態の行の和は 20 になる。残りの重みはどこに集中しているか。
2. **停止の精度保証**。精度保証 Equation 14.21 は、観測できない誤差 $\|V_k - V^*\|_\infty$ を観測可能な変化量 $\|V_k - V_{k-1}\|_\infty$ によって上から抑える。
   1. 14.2.2.3 節の縮小写像の命題から、望遠鏡和の裾 $\sum_{j \ge k} \|V_{j+1} - V_j\|_\infty$ を上界評価することでこの式を導出せよ。与えられた許容誤差未満の誤差を保証するには、スイープ間の変化量がどれだけ小さくなければならないか。
   2. 価値反復法を始める前にスイープ数を確定させる事前上界を、γ、許容誤差、および $\|V_1 - V_0\|_\infty$ のみを用いて導出せよ。滑る湖において $V_0 = 0$、許容誤差 10^{-6} としてその上界を計算し、この湖に対する価値反復法で精度保証が要した 164 回のスイープ（14.2.2 節）と比較せよ。
3. **割引なしの湖**。14.2.2.3 節の縮小写像の命題は $\gamma < 1$ を必要とするが、FrozenLake における価値反復法は $\gamma = 1$ でも収束する。FrozenLake では穴とゴールは報酬 0 の吸収状態であり、唯一の非ゼロ報酬である 1 はゴールへ入るときに得られる。
   1. $\gamma = 1$ として命題の証明をたどれ。それでもなお何が示されるか。続いて、Equation 14.18 の作用素 $\mathcal{T}$ が $\gamma = 1$ で不動点をもつならば、必ず無限個の不動点をもつことを示せ。
   2. $V_0 = 0$ から始める $\gamma = 1$ の価値反復法が、1 で上から有界な非減少列 $V_0 \le V_1 \le V_2 \le \cdots$ を生成し、したがって収束することを示せ。$V_k(s)$ は何を表しているか。
   3. 行動が 1 つだけの 2 状態 MDP であって、V_k が $\gamma = 1$ で限りなく増大するものを構成せよ。この MDP には、小問 b で用いた FrozenLake のどの性質が欠けているか。
4. [code] **インプレースなスイープ**。●`value_iteration` は V_{k+1} の全成分を V_k から計算する。これに対し、**インプレース**なスイープは固定した順序で各状態を巡回し、$V(s)$ をその場で直ちに上書きするため、同じスイープ内で後から訪れる状態はすでに新しい価値を利用する。
   1. 状態をどの順序で巡回しても、インプレースなスイープは上限ノルムにおける γ 縮小写像であり、Equation 14.18 の $\mathcal{T}$ と同じ不動点をもつことを示せ。さらに、その反復列に対しても精度保証 Equation 14.21 がそのまま成り立つことを導け。
   2. インプレースなスイープを実装し、精度保証が 10^{-6} に達するまでに要するスイープ数を、状態の昇順と降順のそれぞれについて、滑る氷と滑らない氷の 4×4 マップおよび 8×8 マップ上で数えよ。得られたスイープ数を `value_iteration` の場合と比較せよ。
   3. なぜ滑る氷よりも滑らない氷の方が順序の影響がはるかに大きいのか。滑らない氷でのスイープ数を Figure 14.7 を用いて説明せよ。

Sutton and Barto『Reinforcement Learning: An Introduction』（第 2 版）の 4.1 節お

よび 4.5 節より改変。

5. [code] **実測される縮小率。**●14.2.2 節では、価値反復法の誤差は 1 スイープあたり約 0.92 倍で縮小し、理論保証の $\gamma = 0.95$ よりも速く減少する。スイープ 14 以降は、すべての反復値に対する貪欲方策が `pi_star` に一致する。
   1. それ以降、誤差が線形漸化式 $V_{k+1} - V^* = \gamma P_{\pi^*}(V_k - V^*)$（ただし P_{π^*} は Equation 14.23 における π^* のもとでの遷移カーネル）に従い、穴とゴールでは誤差が 0 であることを示せ。これより漸近的な縮小率がスペクトル半径 ρ（非終端状態に制限した γP_{π^*} のスペクトル半径）に一致することを導き、ρ を計算せよ。
   2. ρ を $\gamma = 0.999$ において（その割引率での最適方策を用いて）計算し、価値反復法が誤差 10^{-6} に達するまでに要するスイープ数を予測せよ。理論保証 $\gamma^k \|V_0 - V^*\|_\infty$ では $13{,}000$ 回を超えるスイープが必要となる。`value_iteration` を用いて予測を確かめよ。FrozenLake のどの性質によって ρ は γ より小さくなるのか。
   3. 14.2.2 節の規則 $\|V_k - V_{k-1}\|_\infty \le 10^{-6}$ はスイープ 128 で停止した。そこでの真の誤差はスイープ間の変化量の 11.7 倍であり、精度保証の係数 $\gamma/(1-\gamma) = 19$ を大きく下回っていた。公比 ρ で幾何級数的に縮小する誤差が係数 $\rho/(1-\rho)$ を与えることを示せ。なぜ精度保証 Equation 14.21 は ρ を γ の代わりに用いることができないのか。
6. [code] **改善量の定量化。**●14.2.3.2 節の方策改善の命題は、貪欲ステップが価値を下げないことを述べている。性能差補題 (Kakade and Langford 2002) の行列表現である次の恒等式は、その改善量を測るものである。方策 π' に対し、$\mathcal{T}^{\pi'}$ を Equation 14.22 における π' の方策評価作用素とし、$M_{\pi'}$ を問題 1 の行列とする。
   1. 任意の 2 つの方策 π と π' に対して
   $$V^{\pi'} - V^{\pi} = M_{\pi'}(\mathcal{T}^{\pi'} V^{\pi} - V^{\pi}), \tag{14.27}$$
   および $(\mathcal{T}^{\pi'} V^{\pi} - V^{\pi})(s) = \sum_a \pi'(a \mid s)\,[Q^{\pi}(s,a) - V^{\pi}(s)]$ が成り立つことを示せ。
   2. この恒等式を用いて方策改善の命題を別証明せよ。さらに、状態 s が $Q^{\pi}(s, \pi'(s)) > V^{\pi}(s)$ を満たすならば、その状態において真に改善することを示せ。
   3. π^* を π の役割に置いてこの恒等式を適用し、$V^{\pi'} \le V^*$ が確率的方策を含むあらゆる定常方策 π' に対して成り立つことを示せ。
   4. π = `uniform` および $\pi' = \pi^*$ として、初期状態においてこの恒等式を数値的に確かめよ。
7. [code] **ポテンシャルに基づくシェイピング。**●14.2.5 節の命題は、`TabularMDP` のように終端状態が吸収状態であるときにはあらゆるポテンシャルに対して成り立つが、シミュレータのように終端状態でエピソードが停止するときには終端状態で 0 になるポテンシャルに対してのみ成り立つ。
   1. 吸収状態も含め、`mdp.r` のすべての要素に定数 c を加えよ。あらゆる軌跡の割引収益が同じ量だけ変化することを示し、その変化量を求めよ。どの定数ポテンシャル Φ が $\gamma\Phi(s') - \Phi(s) = c$ を満たすか。
   2. 次に、エピソードが進行している間だけ定数を課すこととし、非終端状態においてのみ c を $r(s,a)$ に加えよ。$c \in \{-0.02, -0.01, 0.01, 0.05\}$ のそれぞれについて `value_iteration` で最適方策を計算し、状態 0 における行動、`mdp.policy_value` による元の報酬のもとでの価値、および 14.2.4 節と同様に計算した 100 ステップ以内のゴール到達確率を求めよ。c の各符号が及ぼす影響を、定数ポテンシャルと終端条件を用いて説明せよ。
   3. `shaped` の接近ボーナスをポテンシャル $\Phi(s) = -0.3\,\ell(s)$（ただし ℓ はマンハッタン距離 `dist`）に置き換え、シェイピング後の報酬を次の状態について平均すると、すべての状態において次式が得られる。

$$\tilde{r}(s,a) = r(s,a) + \sum_{s'} P(s' \mid s,a)\,[\gamma\Phi(s') - \Phi(s)]. \tag{14.28}$$

value_iteration が返す価値 $\tilde{V}$ がすべての状態で $\tilde{V}(s) = V^*(s) - \Phi(s)$ を満たすこと、およびシェイピング後の最適方策が元の報酬のもとで pi_star と同じ価値をもつことを確かめよ（**左**と**右**が同値となるマス 6 では 2 つの方策は異なりうる）。続いて、終端状態で停止するシミュレータと同様に終端状態でのシェイピング項を取り除き、得られた最適方策の元の報酬のもとでの価値を再計算せよ。最後に、終端状態で Φ を 0 に設定して同じ計算を繰り返せ。

Sutton and Barto『Reinforcement Learning: An Introduction』の演習問題 3.15 および 3.16 より改変。

8. [extended] **バックアップ予算。●●**14.2.3.2 節の方策反復法が要したのはわずか 2 ラウンド（各ラウンドで 1 回の連立一次方程式の厳密解法）であったのに対し、価値反復法は価値の精度保証を得るまでに 164 回のスイープを要した。状態数が多い問題では直接解法は高コストすぎるため、代わりにスイープによって評価を行う。コストを**バックアップ**の回数で測ろう。1 回のバックアップは 1 つの状態行動対に対する Equation 14.19 の角括弧項の 1 回の評価であるため、価値反復法の 1 スイープには $|\mathcal{S}||\mathcal{A}|$ 回のバックアップ、決定論的な方策の評価の 1 スイープには $|\mathcal{S}|$ 回のバックアップというコストがかかる。**修正方策反復法** (Puterman 1994) は、これら 2 つのアルゴリズムの間を補間する手法である。初期値を $U_0 = 0$ とし、ラウンド k ではまず価値反復法のスイープを 1 回行い（同時に貪欲方策 π_k も U_k から得られる）、続いて現在の表から出発して m 回のスイープ（Equation 14.22 による π_k の評価）を行う。Equation 14.22 の作用素 $\mathcal{T}^{\pi_k}$ を用いれば、1 ラウンドは $U_{k+1} = (\mathcal{T}^{\pi_k})^{m+1} U_k$ と書ける。$m = 0$ では価値反復法に一致し、$m \to \infty$ の極限では方策反復法になる。
   1. $V_k = \mathcal{T}^k U_0$ を価値反復法の反復列とする。$\mathcal{T}U_0 \geq U_0$ ならば $V_k \leq U_k \leq U_{k+1} \leq V^*$ がすべての k とすべての m に対して成り立つことを示し、この前提が FrozenLake で成り立つことを確かめよ。これより、同じラウンド数の後では、修正方策反復法が V^* から価値反復法より遠くなることは決してないことを導け。
   2. 修正方策反復法を実装し、各スイープ後の誤差 $\|U - V^*\|_\infty$（現在の表 U の誤差）を累積バックアップ数に対してプロットせよ。ただし $m \in \{0, 1, 5, 20, 100\}$ とし、滑る氷の 4×4 マップおよび 8×8 マップ上で $\gamma = 0.95$ とする。
   3. 大きい m に対する曲線の形状を説明せよ。なぜ貪欲ステップと貪欲ステップの間で誤差の減少が停滞するのか。
   4. $\gamma = 0.99$ として小問 b を繰り返せ。最小のバックアップ数で到達させる m（目標誤差 10^{-6} に対して）は、$\gamma \to 1$ に伴ってどのように変化するか。

14.3 時間差分学習

前節では、遷移テーブルを読み取ることで滑る湖を渡る最良の方法を見つけた。すべてのセルとすべてのコマンドについて、エージェントが滑って行き着く可能性のある 3 つの場所すべてにわたって和をとったのである。しかし、現実の湖に降り立ったエージェントはそのようなテーブルを持たない。行動を起こし、どこに行き着いたかを確かめ、得られた報酬を受け取ることしかできない。各ステップで目にするのは環境が選んだ 1 つの遷移先だけであり、起こりうる遷移先とその確率の一覧が示されることは決してない。このような 1 回ずつの抽出結果からでも、エージェントは各コマンドの良し悪しを学習し、そこから最良の方策（policy）を導き出せるだろうか。

その答えを、価値の予測については 1988 年に Richard Sutton が、行動の選択については 1989 年に Chris Watkins が見出した (Sutton 1988; Watkins 1989; Watkins and Dayan 1992)。エージェントが観測した 1 つの遷移先を計算できない総和の 1 サンプル推定値とみなし、テーブルの

要素をその推定値に向けて少しだけ動かすのである。訪問を重ねるにつれてこの更新はサンプルを平均化し、その平均が動的計画法（dynamic programming）のバックアップと一致する。こうして得られる手法である **Q 学習**（Q-learning）は、環境のモデルを必要としない。テーブルをニューラルネットワークに置き換えれば、次章で扱う深層 Q 学習になる。

サンプルからの学習には 2 つの問いが生じる。第 1 に、サンプルにはノイズが含まれるため訪問が重なるにつれてステップを縮めなければならないが、どのくらいの速さで縮めるべきか。第 2 に、エージェントが一度も試さないコマンドの要素は決して更新されないため、テーブルでの評価が低いコマンドも試し続けなければならない。本節では 1 つ目の問いに答え、2 つ目の問いには小さな割合でランダムに行動するという最も単純な規則で対処する。その規則が伴うコストについては 14.4 節で詳しく調べる。最後に本節の締めくくりとして、同じ遷移データをもう 1 つの方法で利用し、遷移テーブルそのものを推定してその推定モデル上で計画を立てる。

```
import gymnasium as gym
```

14.3.1 サンプルによるバックアップ

価値反復法（value iteration）が解いた方程式から出発し、そのどの部分にモデルが必要なのかを確かめよう。

14.3.1.1 ベルマン作用素から 1 サンプルへ

14.2 節の価値反復法は、ベルマン最適バックアップという 1 つの操作を繰り返すものであった。状態価値（state value）ではなく行動価値（action value）に対して書き下すと、ベルマン作用素（Bellman operator）$\mathcal{T}$ はテーブル Q を次のテーブルへ写す。

$$(\mathcal{T}Q)(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \max_{a' \in \mathcal{A}} Q(s',a'), \qquad \text{for all } s \in \mathcal{S},\ a \in \mathcal{A}. \tag{14.29}$$

$\mathcal{T}Q$ の各要素は、即時報酬の期待値に、エージェントが行き着いた先から Q に関して貪欲に行動したときの割引価値を加えたものである。その不動点（fixed point）は Q^* であり、どのテーブルから始めても反復適用によって Q^* へ収束する。これは $\mathcal{T}$ が 2 つのテーブル間の最大差を係数 γ 倍に縮めるためであり、14.2 節の縮小議論を行動価値に適用した結果である。

モデルがバックアップに現れるのは、次状態 s' にわたる平均という 1 か所だけである。平均はサンプルから推定できる。エージェントが状態 s にいて a を指示し、報酬 r を受け取って s' に到達したとする。このとき $r + \gamma \max_{a'} Q(s',a')$ という値は確率的に変動する。湖の環境では、報酬もまたエージェントがどこに行き着くかに依存する。次状態と報酬にわたって平均をとれば、この値は $(\mathcal{T}Q)(s,a)$ に等しい。したがって、1 回の遷移はバックアップの 1 要素に対する不偏な 1 サンプル推定値となる。

モデルフリー制御が学習対象を Q とし、V としない理由はここにある。状態価値のバックアップ $V(s) \leftarrow \max_a [r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V(s')]$ では、最大値演算が s' にわたる平均の外側にある。1 回の遷移からわかるのは 1 つの行動がもたらした結果だけであり、他の行動の角括弧内の値については何もわからないため、それらの最大値を推定することはできない。一方、Q の形式では行動ごとに独立した要素があり、平均はその要素の内側に収まっているため、最大値演算はテーブルがすでに保持している数値に対してのみ作用する。さらにこの形式のおかげで、貪欲方策（greedy policy）$\operatorname{argmax}_a Q(s,a)$ もやはりモデルなしでテーブルの 1 行を参照するだけで求まる。

Q 学習は、訪問した要素を**ターゲット**と呼ばれるこのサンプルに向けて割合 α だけ動かす。

$$Q(s,a) \leftarrow (1-\alpha)\,Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a')\big). \tag{14.30}$$

ステップサイズ $\alpha \in (0,1]$ はどれだけ動かすかを表す。$\alpha = 1$ なら要素をターゲットでそのまま置き換え、小さな α なら多数のターゲットを平均化する。ターゲットが次状態におけるテーブル自身の推定値から作られるため、この更新は**ブートストラップ**すると呼ばれる。式を整理すると、変化量は次の 1 つの数値に比例する。

$$\delta = r + \gamma \max_{a'} Q(s',a') - Q(s,a), \qquad Q(s,a) \leftarrow Q(s,a) + \alpha\,\delta. \tag{14.31}$$

この数値 δ は**時間差分誤差**（temporal-difference error）である (Sutton 1988)。すなわち、1 ステップ後に判明したその対の価値の見積もりから、ステップ前にテーブルが示していた値を引いたものである。Figure 14.6 のパネル (d) にこの置き換えを示す。厳密なバックアップはすべての次状態にわたって和をとるが、サンプルによるバックアップは観測された単一の遷移をたどり、次状態での最大値演算はそのまま残す。

ブートストラップするかどうかは設計上の選択である。**モンテカルロ**推定はエピソードが終わるまで待ち、実際に得られた収益（return）を $Q(s_t,a_t)$ のターゲットとして用いる。このターゲットはエピソードを生成した方策の価値に対して不偏であるが、それ以降のすべての遷移に伴うランダム性を抱え込む。これに対し、**時間差分**（temporal-difference、TD）ターゲットは 1 ステップ先だけを見て、残りはテーブルの値を信用する。テーブルが不正確なうちはバイアスを持つものの、分散は通常はるかに小さい (Sutton 1988; Sutton and Barto 2018)。Q 学習はこの 1 ステップのターゲットに最大値演算を加えることで、データを収集する方策の価値ではなく Q^{*} を推定する。

14.3.1.2 1 遷移の手計算

滑る湖において、すべて 0 のテーブルから出発し、$\gamma = 0.95$、そして計算しやすいように $\alpha = 0.5$ とする。エージェントが初めてゴールに到達するまでは、ゴール以外の報酬がすべて 0 であり、ブートストラップ元となる要素もすべて 0 であるため、どのターゲットも 0 になる。最初の成功がセル 14 から生じたとしよう。ここでは**右**のコマンドを出すと、ゴールであるセル 15 へ確率 $1/3$ で滑り込む。ゴールは終端状態（terminal state）なのでターゲットは報酬のみとなり、更新は次のようになる。

$$Q(14,\text{right}) \leftarrow 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5.$$

テーブルの他の要素は何も変わらない。その後、エージェントがセル 13 に立ち、**右**を指示して 14 へ滑り込んだとする。このときのターゲットは $0 + 0.95 \cdot \max_{a'} Q(14,a') = 0.95 \cdot 0.5 = 0.475$ となるため、$Q(13,\text{right}) \leftarrow 0.5 \cdot 0.475 \approx 0.24$ と更新される。セル 14 での更新より前であれば、13 からの同じ遷移が起きてもエージェントは何も学べなかったはずである。このように価値は、エージェントがたまたまたどった遷移に沿って、訪問ごとに 1 セルずつゴールから逆方向へ伝播していく。価値反復法では、すべての遷移に沿って一度に、1 スイープにつき 1 セルずつ価値を伝播させていた。

ターゲットにはノイズがある。次にエージェントが 14 で**右**を指示し、まだ要素が 0 のままである上のセル 10 へ滑ったとすると、ターゲットは 0 となり $Q(14,\text{right}) \leftarrow 0.25$ へ下がる。この要素はターゲット 1 と 0 の間で揺れ動き、さらに後にはセル 10 とセル 14 からブートストラップされた値の間で揺れ動く。これらのターゲットの平均だけが本来のバックアップであり、その平

均を形成するためにステップをどれほどの速さで縮めるべきかが、後述するステップサイズの主題である。

14.3.1.3 最小二乗法と二重サンプリング

なぜこのようなノイズだらけのステップが正しい解へたどり着くのだろうか。テーブルを固定し、次状態と報酬にわたって時間差分誤差の平均をとってみよう。

$$\mathbb{E}[\delta \mid s, a] = (\mathcal{T}Q)(s,a) - Q(s,a). \tag{14.32}$$

期待更新式 $Q(s,a) \leftarrow Q(s,a) + \alpha\big((\mathcal{T}Q)(s,a) - Q(s,a)\big)$ は、その 1 要素に対する緩和された価値反復法のステップになっている。つまり Q 学習とは、サンプリングされた要素を 1 つずつ実行する価値反復法であり、各ステップに平均 0 のノイズによる摂動が加わったものである。

回帰モデルを当てはめた経験のある読者なら、テーブルをターゲットに当てはめる自然な方法は最小二乗法（least squares）ではないかと疑問に思うだろう。記録された N 個の遷移 (s_i, a_i, r_i, s_i') が与えられたとき、二乗時間差分誤差を勾配降下法（gradient descent）で最小化することも考えられる。

$$\ell(Q) = \frac{1}{N}\sum_{i=1}^{N}\Big(Q(s_i,a_i) - r_i - \gamma \max_{a'} Q(s_i', a')\Big)^2. \tag{14.33}$$

1 つの項の勾配は、$Q(s_i,a_i)$ を通る成分と、ターゲット内部の最大値を通る成分の 2 つからなる。更新式 Equation 14.31 はターゲットを固定することで前者を残し、後者を捨てる。そのためこの更新は**半勾配**（semi-gradient）ステップと呼ばれ、ℓ に対する勾配降下法ではない。

第 2 の成分を捨てることこそが、Q 学習を正しく機能させる鍵である。データには各対 (s,a) が頻度 $d(s,a)$（データ収集方策の訪問分布）で含まれており、サンプルが無限に多くあるとしよう。(s,a) で条件付けると、各二乗項の期待値は平均の二乗とターゲットの分散に分解される。

$$\begin{aligned} L(Q) = \mathbb{E}_{(s,a)\sim d}\Big[\big(Q(s,a) - (\mathcal{T}Q)(s,a)\big)^2\Big] \\ + \mathbb{E}_{(s,a)\sim d}\Big[\mathrm{Var}\big(r + \gamma \max_{a'} Q(s',a') \;\big|\; s,a\big)\Big]. \end{aligned} \tag{14.34}$$

第 1 項は二乗ベルマン残差であり、Q^* において 0 になる。第 2 項は次状態と報酬に関するターゲットの分散であり、遷移が確率的である限り Q に依存する。L の最小化は残差を小さくすることとターゲットの分散を抑えることのトレードオフになるため、その最小点（minimizer）は Q^* になるとは限らない。3 状態の例を見れば、どの方向に歪むかがわかる。状態 s から唯一の行動をとると、それぞれ確率 $1/2$ で u_1（報酬 1 を与えて終端）または u_2（報酬 0 を与えて終端）へ遷移するとする。このとき $Q^*(u_1) = 1$、$Q^*(u_2) = 0$、$Q^*(s) = \gamma/2$ である。一方、s におけるターゲットの分散は $\gamma^2(Q(u_1) - Q(u_2))^2/4$ となるため、L の最小点はこの分散を下げようとして 2 つの遷移先の価値を互いに引き寄せ、1 と 0 から遠ざけてしまう。どこまで引き寄せられるかは演習問題 4 で計算する。

残差勾配法（residual-gradient methods）は二乗残差の全勾配に沿って降下するが、同じ困難に直面する (Baird 1995)。その勾配の不偏推定量を作るには、同じ対からの独立な 2 つの遷移先（ターゲット用と導関数用）が必要になるが、1 本の軌跡（trajectory）を生きるエージェントが観測できるのは 1 つだけである。これが**二重サンプリング問題**（double-sampling problem）である。遷移と報酬が決定論的であれば分散の項は消え、2 つのアプローチは一致する。ターゲッ

トを固定するという考え方は、ニューラルネットワークを用いる設定では、ターゲット計算用にゆっくり更新される第 2 のネットワークのコピー、すなわち 15.4 節のターゲットネットワークとして再び登場する。

14.3.1.4 ステップサイズと収束

先ほどの手計算の例では、1 つの要素がノイズを含むターゲットの列 $y_1, y_2, \dots$ を次々に受け取っていた。これらを統合する最も素直な方法は平均をとることである。ステップサイズを $\alpha_n = 1/n$（その対への n 回目の訪問時）とおくと、更新式は $Q_n = Q_{n-1} + (y_n - Q_{n-1})/n = ((n-1)Q_{n-1} + y_n)/n$ となり、帰納法により次式を得る。

$$Q_n = \frac{1}{n}\sum_{k=1}^{n} y_k,$$

これは初期値が何であれ、ターゲットの標本平均（sample mean）にほかならない。もしターゲットが一定の平均を持つ独立な抽出値であれば、大数の法則（law of large numbers）によりこの要素はその平均へ収束する。Robbins と Monro は、減少するステップサイズが一般にこれと同じ効果を持つための条件を与えた (Robbins and Monro 1951)。

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty. \tag{14.35}$$

第 1 の条件はステップが任意の距離まで移動できるようにするものであり、これによって初期値は忘れ去られる。第 2 の条件は累積するノイズを有限に抑える。標本平均に対応する $\alpha_n = 1/n$ は両方を満たす。定数のステップサイズは第 2 の条件に反するため、要素は平均の周りで揺らぎ続け、α が小さいときその分散は α のオーダーになる。

命題（表形式の Q 学習の収束）。状態集合と行動集合が有限であり、報酬が有界かつ $\gamma < 1$ であるとする。すべての対 (s,a) が無限回更新され、そのステップサイズが Robbins–Monro 条件を満たすならば、Q は確率 1 で Q^* に収束する (Watkins and Dayan 1992; Jaakkola et al. 1994; Tsitsiklis 1994)。

平均化の議論だけではこの命題は証明できない。ターゲットが学習途中のテーブルから計算されるため、一定の平均を持たないからである。残差勾配降下法でも期待ステップはある不動点で 0 になるが、それは誤った不動点である。Q 学習を際立たせているのは縮小性である。現在の誤差がどのようなものであれ、期待ステップはその最大誤差を縮め、さらにステップサイズの条件によって各ステップ周りのノイズが平均化されて消えていく。

どのコマンドも指示どおりの方向へ進む滑らない氷の上では、各対の次状態は 1 つに定まり、ターゲットに遷移ノイズは生じない。このとき定数 α を用いた更新は緩和バックアップ $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(\mathcal{T}Q)(s,a)$ となり、全要素に適用すれば縮小率 $(1-\alpha)+\alpha\gamma < 1$ で縮小する。したがって $\alpha = 0.9$ のような大きなステップでも安全である。

ただし、この条件が扱うのは極限であり、有限の計算予算ではない。標本平均はすべてのターゲットに同じ重みを与えるため、テーブルの大部分がまだ 0 だった初期のターゲット（湖の環境では小さすぎる値になる）にも同じ重みがかかってしまう。$\alpha_n = 1/n$ を用いると、Q 学習が必要とする更新回数は $1/(1-\gamma)$（ここでは 20）に対して指数関数的に増大しうるが、$\alpha_n = 1/n^{\omega}$（ただし $1/2 < \omega < 1$）とすれば多項式オーダーの増大にとどまる (Even-Dar and Mansour 2003)。以下では、初期値 0 をもう 1 つのターゲットとして数えた標本平均にあたる $\alpha_n = 1/(1+n)$ を試す。本節でデフォルトとする $\alpha_n = 1/(1+0.1\,n)$ も両方の条件を満たし、そのステップサイ

ズは漸近的に標本平均の 10 倍になる。このスケジュールは後から得られるターゲットほど重く評価し、k 番目のターゲットにはおよそ k^9 に比例する重みを与えるため（演習問題 5）、推定値は改善されていくターゲットに追随できる。

14.3.1.5 終端のマスク処理

エピソード型タスクでは、ターゲットに 1 つ補正が必要になる。穴やゴールといった終端状態にはその先の未来がないため、継続価値は 0 であり、ターゲットには即時報酬のみが含まれる。

$$Q(s,a) \leftarrow Q(s,a) \\ + \alpha\Big(r + \gamma\,(1 - \mathbf{1}(s' \text{ terminal}))\max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)\Big).$$

このマスクには `terminated` を用い、`truncated` は用いない。14.1 節で説明したように、時間制限による打ち切りは本来なら過程が続くはずの状態で記録を止めるだけなので、その継続価値は 0 ではない。打ち切りを終端として扱ってしまうと、長いエピソードの最後のセルはどれも価値がないとテーブルに学習させてしまう。コード上では、この補正は `gamma * (1 - terminated) * Q[s_next].max()` という 1 つの因数で表される。

14.3.2 滑る湖における Q 学習

ここからは 14.1 節の滑る湖に対して $\gamma = 0.95$ でこの更新を実行し、14.2 節の厳密解と照らし合わせて結果を評価する。学習器は `reset` と `step` のみを通じて環境とやり取りする。モデル `mdp`、最適価値 `V_star`、最適行動価値 `Q_star`（V^* の 1 回のバックアップ）、および最適方策（optimal policy）`pi_star` は、結果の評価にのみ用いる。

```
gamma, num_episodes, num_seeds = 0.95, 4000, 5
env = gym.make('FrozenLake-v1', is_slippery=True)
mdp = d2l.TabularMDP.from_gym(env, gamma)
V_star = d2l.value_iteration(mdp, num_iters=1000)[-1]
Q_star = mdp.backup(V_star)
pi_star = Q_star.argmax(axis=1)
```

14.3.2.1 行動方策

Q 学習には、学習中に行動を選ぶための規則である**行動方策**（behavior policy）π_e が必要である。現在のテーブルに対して貪欲に行動するだけでは、2 つの理由で失敗する。第 1 に、すべて 0 のテーブルではどの行も 4 つすべてが同値になり、`np.argmax` は最初のインデックスを返すため、貪欲なエージェントは常に**左**を指示して決してゴールを見つけられない。第 2 に、現時点で要素の値が低い行動は一度も試されないため、その要素は更新されず、過小評価が永続してしまう。最も単純な打開策は、小さな確率 ϵ でランダムに行動することである。

$$\pi_e(a \mid s) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}| & a = \operatorname{argmax}_{a'} Q(s,a'), \\ \epsilon/|\mathcal{A}| & \text{otherwise.} \end{cases} \tag{14.36}$$

これが ϵ **貪欲**（epsilon-greedy）方策である。確率 ϵ で探索（exploration）し、それ以外のときは argmax における同値を一様ランダムに解消しつつ貪欲行動をとる。次の関数は一様乱数を 1 つ引いてどちらにするかを決め、ランダムな行動または最大点（maximizer）の集合からランダムに選んだ要素を返す。

```
def epsilon_greedy(q, epsilon, rng):
    """Explore with probability epsilon, else act greedily on the values q."""
    if rng.random() < epsilon:
        return int(rng.integers(len(q)))
    # Ties are coin flips: np.argmax would always pick action 0 on a zero
    # table, and an agent that only ever moves left never finds the goal.
    return int(rng.choice(np.flatnonzero(q == q.max())))
```

何も知らないテーブルは大いに探索すべきであり、ほぼ正しいテーブルは大半のステップで活用（exploitation）すべきである。そこで、訓練が進むにつれて ϵ を減少させる。`linear_schedule` は開始値、終了値、ステップ数を受け取り、ステップ数を引数として線形補間したのち終了値を維持する関数を返す。

```
def linear_schedule(start, end, num_steps):
    """step -> value, interpolated from start to end, then held at end."""
    return lambda step: end + (start - end) * max(0.0, 1.0 - step / num_steps)
```

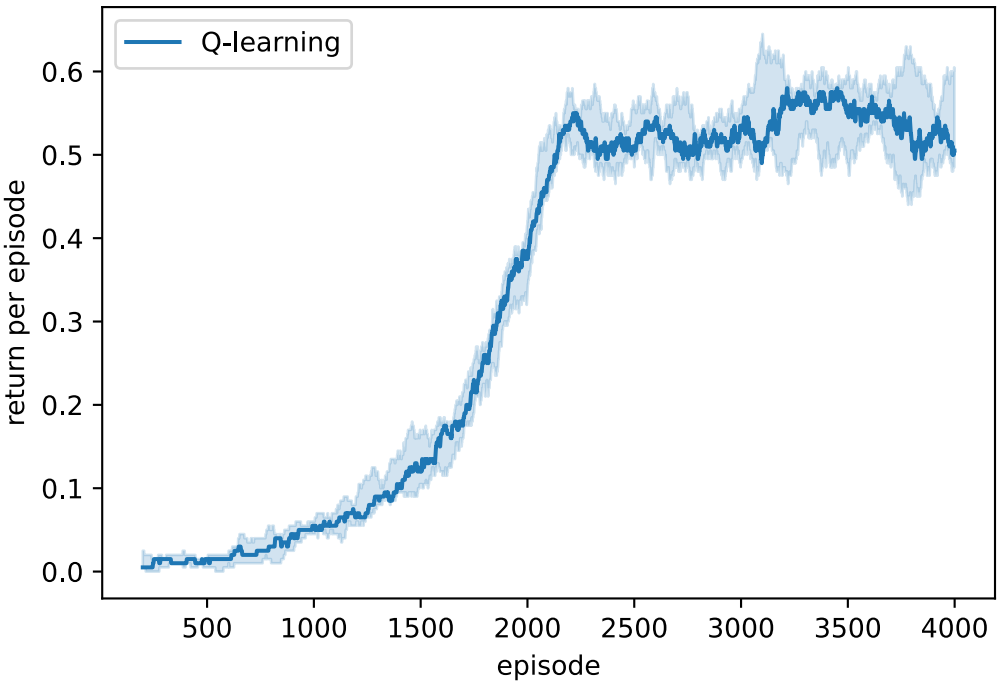
14.3.2.2 学習ループ

`q_learning` は終端マスク付きの更新式 Equation 14.30 を実行する。テーブル `Q` と訪問回数のテーブル `visits` を受け取り、両方をインプレースで書き換える。訪問回数は関数 `alpha(n)` を通じて各対のステップサイズを定め、実行後にはどの対が試されたかの記録にもなる。探索率（exploration rate）はエピソードを引数とする関数 `epsilon(k)` であり、デフォルトでは訓練の前半で 1 から 0.05 まで減少したのち 0.05 に保たれる。ループ本体は行動を選び、1 回の遷移を観測し、1 つの要素を $\alpha\delta$ だけ変化させ、各エピソードの収益を順に返す。

```
def q_learning(seed, Q, visits, env, num_episodes, gamma,
               alpha=lambda n: 1 / (1 + 0.1 * n), epsilon=None):
    """Tabular Q-learning; updates Q in place, yields each episode's return.
    `alpha(n)` is the step size at the n-th visit of a pair; `epsilon(k)` is
    the exploration rate in episode k (default: 1 to 0.05 over the first
    half of the episodes, then held)."""
    rng = np.random.default_rng(seed)
    if epsilon is None:
        epsilon = linear_schedule(1.0, 0.05, num_episodes // 2)
    env.reset(seed=seed)
    for episode in range(num_episodes):
        s, done, ret = env.reset()[0], False, 0.0
        while not done:
            a = epsilon_greedy(Q[s], epsilon(episode), rng)
            s_next, r, terminated, truncated, _ = env.step(a)
            visits[s, a] += 1
            delta = r + gamma * (1 - terminated) * Q[s_next].max() - Q[s, a]
            Q[s, a] += alpha(visits[s, a]) * delta
            s, done, ret = s_next, terminated or truncated, ret + r
        yield ret
```

テーブルに必要なのはインデックス演算だけなので、本節のコードセルはすべて素の NumPy で書かれている。シードごとに 1 つのテーブルを 4,000 エピソード訓練し、各エピソードの収益をプロットする。

```
Q = np.zeros((num_seeds, mdp.num_states, mdp.num_actions))
visits = np.zeros_like(Q)
returns = np.array([list(q_learning(k, Q[k], visits[k], env, num_episodes,
                                    gamma)) for k in range(num_seeds)])
d2l.plot_curves({'Q-learning': returns}, xlabel='episode',
                ylabel='return per episode', smooth=200)
```



曲線の各点はそこまでの直近 200 エピソードを平均しているため、曲線はエピソード 200 から始まる。帯はシード間の幅を表す。成功がまれなうちは収益は 0 付近にとどまり、ゴールから価値が逆伝播するにつれて上昇し、やがて 0.5 付近で横ばいになる。これはエピソード 2,000 で探索率が下限に達した後の領域である。

14.3.2.3 学習されたテーブルと厳密解

0.5 付近という収益は良い結果なのだろうか。14.2 節の最適方策は、100 ステップの制限時間内に約 73% の確率でゴールに到達する。この最適方策は速く到達するほど有利になる割引収益を最大化するため、割引を無視した方策のほうが成功率は 1 ポイントほど高くなる。さらに、訓練曲線が測っているのは 20 ステップに 1 回の割合でランダムに行動し続ける行動方策であり、テーブルそのものを評価しているわけではない。

ここではモデルが既知であるため、テーブルを厳密に評価できる。$\gamma = 1$ とし、ゴールでのみ報酬 1 が与えられるとすると、開始状態の価値はゴールへの到達確率に一致する。すべて 0 の状態から 100 スイープ実行すると、14.2 節の `policy_evaluation` は 100 ステップ以内のゴール到達確率を計算する。`success` は状態ごとに 1 行の行動確率で与えられた方策に対してこの方策評価（policy evaluation）をラップする関数である。マスク `live` はエージェントが行動するセルを表し、`greedy` は各テーブルの貪欲方策を保持する。次のコードセルでは、各テーブルの価値誤差と行動価値誤差、Q^* のもとで貪欲行動が取り逃す最大の価値、厳密な成功率、π^* が確率 0.05 でラ

ンダムに行動したときの成功率、および 1 回の実行が要したステップ数を表示する。

```
lake1 = d2l.TabularMDP.from_gym(env, 1.0)    # gamma = 1: V(s0) = P(goal)
success = lambda pi: d2l.policy_evaluation(lake1, pi, 100)[-1][0]
live = ~np.isin(env.unwrapped.desc.flatten(), [b'H', b'G'])  # agent acts
greedy = np.eye(mdp.num_actions)[Q.argmax(axis=-1)]    # (seed, s, a)
optimal = np.eye(mdp.num_actions)[pi_star]
taxed = 0.95 * optimal + 0.05 / mdp.num_actions        # pi* at epsilon = 0.05
v_err = np.abs(Q.max(axis=-1) - V_star).max(axis=-1)   # one entry per seed
q_err = np.abs(Q - Q_star)[:, live].max(axis=(1, 2))
regret = (V_star - (greedy * Q_star).sum(-1)).max(-1)  # what greedy gives up
wins = [success(g) for g in greedy]
for name, per_seed in [('max_s |V_Qhat(s) - V*(s)|', v_err),
                       ('max_(s,a) |Qhat - Q*|', q_err),
                       ('max_s V*(s) - Q*(s, greedy)', regret),
                       ('greedy success in 100 steps', wins)]:
    print(f'{name:>27}: {np.round(per_seed, 3)}')
steps = np.median(visits.sum(axis=(1, 2)))
print(f'success of pi* {success(optimal):.3f}, of pi* at epsilon = 0.05 '
      f'{success(taxed):.3f}; median environment steps {steps:,.0f}')
```

```
  max_s |V_Qhat(s) - V*(s)|: [0.019 0.012 0.021 0.008 0.006]
      max_(s,a) |Qhat - Q*|: [0.131 0.079 0.113 0.087 0.075]
max_s V*(s) - Q*(s, greedy): [0. 0. 0. 0. 0.]
greedy success in 100 steps: [0.73 0.73 0.73 0.73 0.73]
success of pi* 0.730, of pi* at epsilon = 0.05 0.537; median environment steps 95,569
```

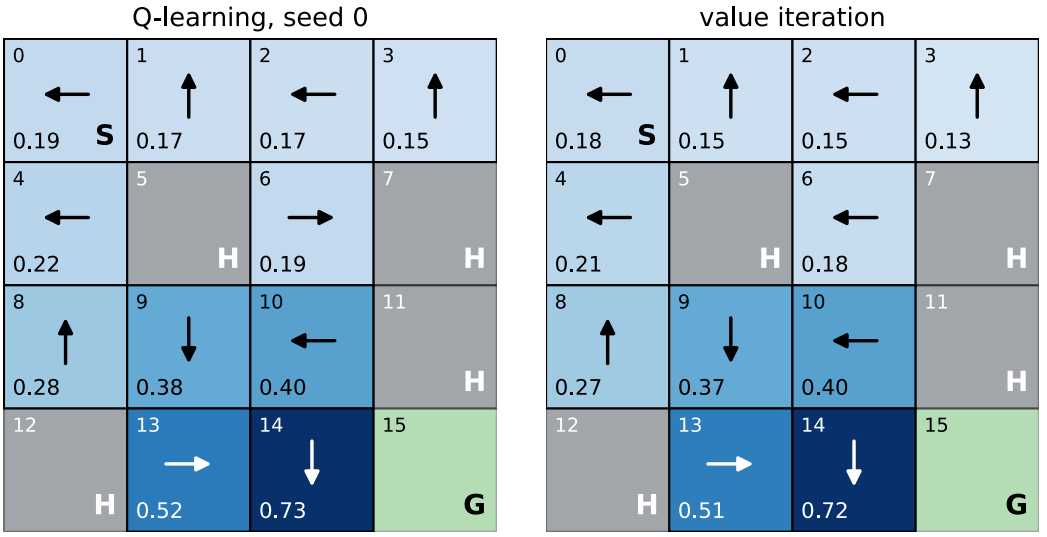
3 行目の値はすべてのシードで 0 になっている。すなわち、学習された貪欲行動が Q^* のもとで価値を損なうセルは 1 つもなく、どの貪欲方策も同値の選び方を除いて最適であり、厳密に 73%（π^* と同じ成功率）で成功している。ただしこれは典型的な実行結果であり、この予算での保証ではない。およそ 8 回に 1 回の実行では、$4{,}000$ エピソード後でも一部の貪欲行動が準最適のまま残るが、たいてい損失はごくわずかである（成功率は 71%〜74%）。まれに起こる最悪のケースでは、エージェントが開始状態で依然として**右**を選好し、成功率が半分未満にとどまる。開始状態の上位 2 つの行動は 0.01 未満しか Q^* の値が違わず、その実行では推定値が低く出たうえ、貪欲規則のもとでは探索のときにしかその行動が試されないため、低い推定値の修正が遅れるからである。価値誤差はシード全体にわたって約 0.03 以内に V^* との差が収まっている。一方、行動価値誤差はその数倍大きく、約 0.05〜0.17 である。貪欲方策が選ばない行動の要素は主にエージェントが探索するときにしか更新されないため、平均化されるターゲットの数が少ないからである。

20 ステップに 1 回ランダムに行動するよう強いられると、π^* でさえ成功率は約 54% にとどまる。訓練曲線が 0.5 付近で頭打ちになるプラトーは、この探索のコストによるものであり、テーブルの欠陥ではない。

1 回の実行は典型的に約 $96{,}000$ 環境ステップを使用する。価値反復法は V^* を、164 スイープ（各スイープ 64 回のバックアップ）、すなわち約 $10{,}500$ 回のバックアップで確定させたが、この 2 つの回数は異なるものを測っている。1 回のバックアップは遷移テーブルの 1 行全体を読み取るのに対し、1 ステップが明かすのはサンプリングされた 1 つの遷移先だけである。演習問題 6 では、必要な情報量の違いを明示したうえで両者を比較する。

最初のシードで学習された価値と矢印を、厳密解と並べて次に示す。

```
d2l.show_grid(env.unwrapped.desc, np.stack([Q[0].max(-1), V_star]),
              np.stack([Q[0].argmax(-1), pi_star]),
              titles=['Q-learning, seed 0', 'value iteration'])
```



学習された矢印は 14.2 節の滑りを考慮した戦略を再現しているが、例外はセル 6 であり、そこでは Q 学習が**右**を指示し、π^* が**左**を指示する。セル 6 は穴である 5 と 7 の間に位置し、この 2 つのコマンドは Q^* のもとで等しい価値を持つため、不一致は厳密な同値に対する解消の仕方の違いにすぎない。価値の精度はエージェントが頻繁に訪れるセルで最も高く、めったに訪れない右上の隅では粗くなっている。

14.3.2.4 ステップサイズスケジュールの比較

滑らない氷の上ではターゲットに遷移ノイズが含まれないため、前述の縮小議論により定数ステップ $\alpha = 0.9$ が許容される。このときエピソード予算の 1/8 でほぼ常に十分であり、ここでは最終テーブルの貪欲方策がすべてのエピソードでゴールに到達する。

```
calm = gym.make('FrozenLake-v1', is_slippery=False)
table = np.zeros((mdp.num_states, mdp.num_actions))
calm_visits = np.zeros_like(table)
list(q_learning(0, table, calm_visits, calm, 512, gamma,  # fills table
                alpha=lambda n: 0.9))
print(f'calm ice, 512 episodes ({calm_visits.sum():,.0f} steps), alpha = 0.9: '
      f'greedy policy succeeds '
      f'{d2l.evaluate(calm, lambda s, rng: int(table[s].argmax()), 100):.0%}')
```

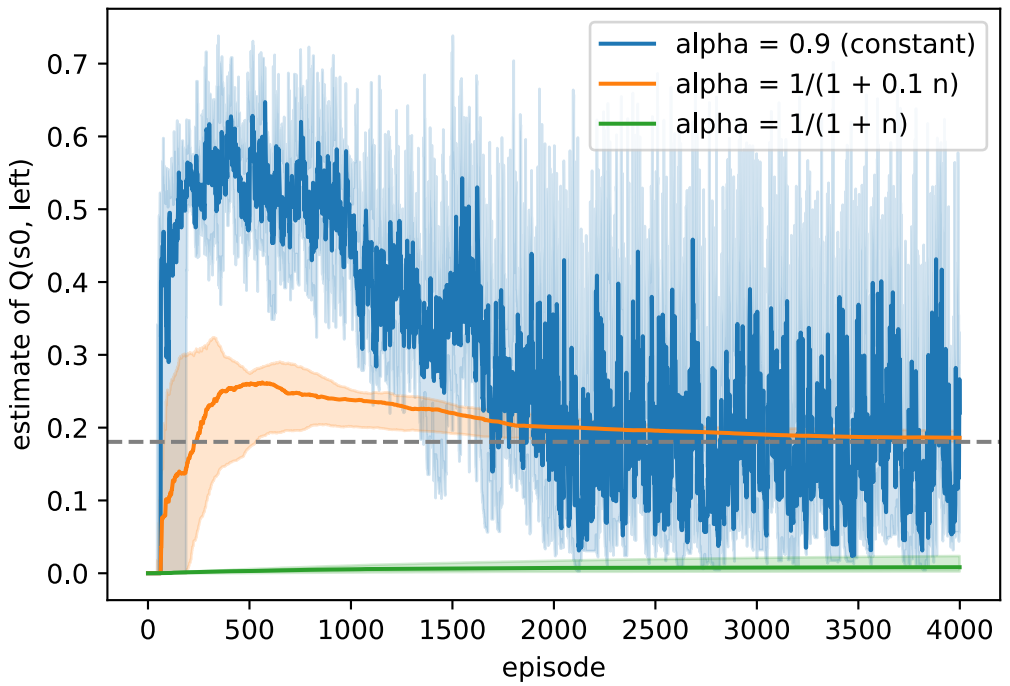
```
calm ice, 512 episodes (3,658 steps), alpha = 0.9: greedy policy succeeds 100%
```

滑らない氷での実行は約 3,500 環境ステップで済むのに対し、滑る氷での実行は約 96,000 ステップを要する。まれに失敗する実行では、エージェントがゴールに到達した回数が少なすぎて開始状態まで価値が逆伝播しておらず、これは 14.4 節で扱う探索の失敗にあたる。コストの大半を決めているのは学習規則ではなく遷移ノイズである。滑る氷の上ではステップサイズが重要になる。そこで同じシード上で 3 つのスケジュールを、1 つの要素 $Q(s_0, \text{left})$（その厳密値 0.180 は

π^* のもとでの開始状態の価値である）を追跡して比較する。プロットにはシード全体の中央値（median）と、最小値から最大値までの帯を示す。

```
a0 = int(pi_star[0])          # FrozenLake's actions: left, down, right, up
trace = {}
for name, alpha in [('0.9 (constant)', lambda n: 0.9),
                    ('1/(1 + 0.1 n)', lambda n: 1 / (1 + 0.1 * n)),
                    ('1/(1 + n)', lambda n: 1 / (1 + n))]:
    runs = []
    for seed in range(num_seeds):
        Qa = np.zeros_like(Q[0])  # updated in place: read after each episode
        runs.append([Qa[0, a0] for _ in q_learning(
            seed, Qa, np.zeros_like(Qa), env, num_episodes, gamma, alpha)])
    trace[f'alpha = {name}'] = np.array(runs)
    print(f'alpha = {name:>14}: final estimates {np.round(runs, 3)[:, -1]}')
print(f'peak of each 1/(1 + 0.1 n) run:  '
      f'{np.round(trace["alpha = 1/(1 + 0.1 n)"].max(axis=1), 3)}')
d2l.plot_curves(trace, xlabel='episode', ylabel='estimate of Q(s0, left)',
                reference=Q_star[0, a0])
```

```
alpha = 0.9 (constant): final estimates [0.056 0.221 0.206 0.265 0.236]
alpha =  1/(1 + 0.1 n): final estimates [0.19  0.186 0.195 0.184 0.18 ]
alpha =      1/(1 + n): final estimates [0.003 0.024 0.008 0.009 0.002]
peak of each 1/(1 + 0.1 n) run:  [0.228 0.291 0.262 0.324 0.205]
```



定数ステップは決して落ち着かない。各ターゲットが推定値を誤差の 90% 分だけ動かすため、4,000 エピソード後の最終推定値は 0.180 の周りに 0.05 付近から 0.25 超まで広く散らばる。デフォルトのスケジュールは、典型的な実行では厳密値から約 0.02 以内に収束して終わる。スケジ

ュール $1/(1+n)$ も Robbins–Monro 条件を満たすが、この予算ではどの実行も厳密値の 1/6 にあたる 0.03 未満にとどまる。すべて 0 のテーブルから計算された初期のターゲットが、そのままの重みを保ち続けるからである。これこそ Even-Dar と Mansour が予測した遅い収束速度である。条件が保証するのは極限であり、有限の予算に適したスケジュールを選んでくれるわけではない。

14.3.2.5 推定とデータ収集の間のフィードバック

行動方策は現在の推定値に依存するため、推定値に誤差があると次にどのデータが収集されるかが変わる。$Q(s,a)$ が大きすぎれば、ϵ 貪欲方策は a をより頻繁に選び、推定値を引き下げるようなターゲットを持つ遷移を集める。逆に $Q(s,a)$ が小さすぎれば、その行動はエージェントが探索するときにしか試されず、上の実行結果が示したように誤差の修正は遅くなる。したがって過大評価は自己修正されるが過小評価はそうならず、先ほど推定値が低かった実行で最初の行動が変わるまでに長い時間がかかったのもそのためである。

更新の不動点では、行動方策が訪問するすべての対において期待補正量が 0 になり、そこでは $Q(s,a) = (\mathcal{T}Q)(s,a)$ が成り立つ。しかしこれはまだ最適方程式ではない。$\mathcal{T}Q$ の内部にある最大値演算は、一度も訪問されず制約を受けない要素にも依然としてわたっているからである。テーブルが訪問された対の上で方程式を満たしていても全体としては誤っていることがありうるため、収束定理ではすべての対が試されることが必要になる。固定された遷移ログから学習するエージェントはこのフィードバックを完全に失うが、その場合については次章で改めて扱う。

14.3.3 オフポリシー学習

ターゲットに最大値演算が含まれることで、Q 学習はある方策に従いながら別の方策について学習する。以下の 2 つの項では、その性質とそれに伴う統計的バイアスを扱う。

14.3.3.1 オフポリシーターゲットと SARSA

Equation 14.31 の $\max_{a'}$ は、行動方策が s' で次にどの行動をとるかにかかわらず、貪欲な継続を評価する。したがって Q 学習は**オフポリシー**である。すなわち、データは探索を行う方策 π_e から得られる一方で、貪欲方策の価値を推定する。これに対するオンポリシーの代替手法は、行動方策が実際に選んだ行動 a'（状態 s' において）からブートストラップする。

$$\delta = r + \gamma\, Q(s', a') - Q(s, a). \tag{14.37}$$

この手法は、各更新が用いる組 (s, a, r, s', a') にちなんで **SARSA** と呼ばれる (Rummery and Niranjan 1994; Sutton and Barto 2018)。固定された行動方策に対して、SARSA の更新の期待値は π_e のベルマン期待方程式（Bellman expectation equation）を不動点に持つため、SARSA はランダムな行動も含めた探索方策の価値 Q^{π_e} を学習する。滑る湖において $\epsilon = 0.05$ を保ち続ける SARSA エージェントは、自身のランダムなステップでときおり穴に落ちるため、穴の隣のセルの価値を π^* がそこで達成する価値より低く見積もることになる。これに対し Q 学習は、粘り強く探索を続けながら、上で見たように π^* と一致する貪欲方策を導き出せる。同じ性質により、貪欲方策が必要とする対をデータが網羅している限り、過去の経験や別のエージェント、あるいは固定されたログからでも学習できる。ただしテーブルをニューラルネットワークに置き換えると、この柔軟性がかえって学習を不安定にしうることを次章で見る。

14.3.3.2 最大化バイアス

ノイズを含む推定値の最大値は、それぞれの平均の最大値を上回りやすい。最大値演算が、たまたま誤差が正になった要素を選び出すからである。最大値関数は凸関数であるため

$\mathbb{E}[\max_{a'} \hat{Q}(s', a')] \geq \max_{a'} \mathbb{E}[\hat{Q}(s', a')]$ が成り立ち、ノイズを含む行からブートストラップするすべてのターゲットはこの**最大化バイアス**（maximization bias）を受け継ぐ。先ほどのステップサイズごとの推移にもそれが現れている。デフォルトのスケジュールでは、$Q(s_0, \text{left})$ の推定値はいったん 0.180 を超えて上昇してから落ち着く。ノイズがなければこのような挙動は起こりえない。すべて 0 のテーブルから出発し、報酬が決して負にならない設定では、厳密なバックアップはいずれも価値を Q^* に向けて引き上げることしかできないため、価値反復法は下から 0.180 に近づく。それにもかかわらず、中央値の推移は約 0.22 以上の水準にエピソード 500 からエピソード $1{,}500$ までとどまっている。ノイズが最大値演算を通ることで推定値を押し上げたのである。表示された各実行のピークは約 0.2 から 0.33 の間にあり、12 個のシードにわたって大半の実行が同様に 0.01 の数倍から 0.1 を超える幅でオーバーシュートする。推定値のノイズがさらに大きい定数ステップではオーバーシュートもいっそう大きくなり、最初の 1,000 エピソードの大部分で約 0.5 に達する。

表形式の推定値が収束するにつれてこのバイアスは縮小するが、ノイズが消えない関数近似（function approximation）のもとでは残り続けることがある。ダブル Q 学習（Double Q-learning）は 2 つの推定器を保持し、一方で最大値を与える行動を選び、もう一方でその価値を評価することで上向きのバイアスを取り除く（ただし推定値が低めに出ることはある）(Hasselt 2010)。演習問題 7 では、このバイアスがエージェントの判断を誤らせるタスク上でダブル Q 学習を構築する。

14.3.4 学習されたモデルにおける計画

Q 学習は各遷移を 1 度しか使わない。1 つの要素を動かすとそのサンプルを捨ててしまうため、次にゴールの価値が変化しても、過去の遷移が残っていてその変化を逆方向へ伝えてくれることはない。これに対し、同じ遷移を数え上げておくこともできる。$N(s, a, s')$ を状態 s で行動 a をとって s' へ移った観測遷移回数とし、$N(s, a) = \sum_{s'} N(s, a, s')$ とおく。この回数から、次の推定モデルが得られる。

$$\hat{P}(s' \mid s, a) = \frac{N(s, a, s')}{N(s, a)},$$
$$\hat{r}(s, a) = \frac{1}{N(s, a)} \sum_{\text{visits of } (s,a)} r,$$

すなわち、各次状態の経験頻度と各対の平均報酬である。推定値をあたかも厳密な真のモデルであるかのように扱って $(\hat{P}, \hat{r})$ 上で価値反復法を行う手法は、**確実性等価**（certainty equivalence）と呼ばれる。これにより、モデルなしの問題は 14.2 節の問題へと還元される。

`Counts` は環境をラップし、すべてのステップをそのまま通しつつ $N(s, a, s')$ と報酬の合計を記録する。`q_learning` は `reset` と `step` のみを通じて環境に触れるため、このラッパ上でも変更なしにそのまま動作し、ラッパなしのときとまったく同じテーブルを学習する。`learned_model` はカウントを割り算して $\hat{P}$ と $\hat{r}$ を求め、`d2l.TabularMDP` として返す。一度も試されなかった対の行は 0 のまま残るため、プランナはそれらを価値のない行き止まりとして扱う。

```
class Counts(gym.Wrapper):
    """Pass every step through and count it: N[s, a, s'] and summed rewards."""
    def __init__(self, env):
        super().__init__(env)
        n_s, n_a = env.observation_space.n, env.action_space.n
        self.N, self.R = np.zeros((n_s, n_a, n_s)), np.zeros((n_s, n_a))
```

```
    def reset(self, **kwargs):
        self.s, info = self.env.reset(**kwargs)
        return self.s, info

    def step(self, a):
        s_next, r, terminated, truncated, info = self.env.step(a)
        self.N[self.s, a, s_next] += 1
        self.R[self.s, a] += r
        self.s = s_next
        return s_next, r, terminated, truncated, info

def learned_model(counts, gamma):
    """Certainty equivalence: the empirical kernel and mean rewards."""
    n = np.maximum(counts.N.sum(axis=-1), 1)   # unvisited pairs: zero rows
    return d2l.TabularMDP(counts.N / n[..., None], counts.R / n, gamma)
```

モデルはデータをどれほど有効に活用できるだろうか。Q 学習に 500 エピソードの短い予算と 2,000 エピソードの長い予算を与え、それぞれ独立した探索スケジュールで最後まで実行する。各実行で得られた遷移データ上で、Q 学習が学習したテーブルと、推定モデルの行動価値（価値反復法による解を 1 回バックアップしたもの）という 2 つのテーブルを比較する。`grade` はテーブルの `V_star` に対する上限ノルム（sup-norm）価値誤差と、`success` で計算した貪欲方策の厳密な成功率を返す。

```
def grade(Q, V_star, success):   # sup-norm error, success of greedy policy
    return (np.abs(Q.max(axis=-1) - V_star).max(),
            success(np.eye(Q.shape[-1])[Q.argmax(axis=-1)]))

for budget in [500, 2000]:
    rows = []
    for seed in range(num_seeds):
        counts, Qs = Counts(env), np.zeros((mdp.num_states, mdp.num_actions))
        list(q_learning(seed, Qs, np.zeros_like(Qs), counts, budget,
                        gamma))                    # fills Qs and the counts
        model = learned_model(counts, gamma)
        Q_model = model.backup(d2l.value_iteration(model, 500)[-1])
        rows.append([counts.N.sum(), *grade(Qs, V_star, success),
                     *grade(Q_model, V_star, success)])
    n, err_q, win_q, err_m, win_m = np.median(rows, axis=0)
    print(f'{budget} episodes, median {n:,.0f} steps '
          f'(medians over {num_seeds} seeds)')
    print(f'  Q-learning:        max |V - V*| {err_q:.3f}, success {win_q:.3f}')
    print(f'  planning in model: max |V - V*| {err_m:.3f}, success {win_m:.3f}')
```

```
500 episodes, median 9,984 steps (medians over 5 seeds)
  Q-learning:        max |V - V*| 0.084, success 0.486
  planning in model: max |V - V*| 0.096, success 0.639
2000 episodes, median 47,282 steps (medians over 5 seeds)
  Q-learning:        max |V - V*| 0.023, success 0.730
  planning in model: max |V - V*| 0.011, success 0.730
```

約 10,000 ステップという少ないデータ量では、計画によって得た貪欲方策のほうが優れている。Q 学習の貪欲方策がゴールに到達するのはおよそ 2 回に 1 回であるのに対し、計画に基づく貪欲

方策は典型的に 3 回に 2 回ほど到達し、10 回中 9 回ほどの実行で同等以上の成績を収めた。この予算での最悪ケースの価値誤差は、めったに訪問されない少数のセルによって決まる。この誤差もたいていはモデルのほうが小さいが常にそうとは限らず、ここに表示した 5 回の実行ではたまたま Q 学習のほうが小さくなっている。約 $47{,}000$ ステップになると、大半の実行で両方の貪欲方策が最適になり、計画による価値の精度は約 2 倍高くなる。ほぼすべてのシードで計画のほうが優れており、2 倍のエピソード数をかけた Q 学習のテーブルと同程度の精度に達していた。理由は先ほどの手計算の遷移が示したとおりである。Q 学習は訪問ごとに 1 遷移ずつ価値を逆伝播させるうえ、その遷移が得られた時点のまだ不正確なテーブルから計算したターゲットを使う。これに対しモデルはすべての遷移を保持しており、価値反復法の数百回のスイープそれぞれが、記録された全遷移を通してゴールの報酬を伝播させる。演習問題 1 では、2 状態の過程で同じ効果を確認する。

もっとも、モデルには固有のコストがある。モデルは $|\mathcal{S}|^2|\mathcal{A}|$ 個のカウントを保存する必要があり、ここでは約 1,000 個の数値で済むものの大きな状態空間では到底扱いきれず、さらに新しいデータのバッチが得られるたびにモデルを解き直さなければならない。またプランナは推定モデルを完全に信用するため、一度も試されなかった対は無価値に見え、1 度だけ試されて運よく良い結果が出た対は実態以上に良く見えてしまう。15.6 節では、カウントの代わりに回帰を用いて Pendulum 上でこのようなモデルを当てはめ、そのモデル内で計画を行う。

こうしてエージェントは、サンプルだけから最適行動価値を学習してそれに基づき行動することも、同じサンプルをモデルに変えて計画を立てることもできるようになった。どちらの方法もデータをどう収集するかに支えられているが、ここで用いた ϵ 貪欲規則は大雑把なものである。20 ステップに 1 回ランダムに行動するよう強いられると π^* でさえ成功率は約 54%（本来の 73% ではなく）に落ちるうえ、貪欲なテーブル参照には最大化の対象となる行動集合が有限である必要がある。14.4 節では、別の規則によって損失を抑えつつ同等の学習ができるかを問い、さらに本章の後半では方策そのものを学習対象とする。

14.3.5 まとめ

Q 学習は、ベルマンバックアップの期待値を観測された 1 回の遷移で置き換え、訪問した要素をステップサイズ α で時間差分誤差 δ に沿って動かす。期待ステップは緩和された価値反復法のステップであり、ベルマン作用素が縮小写像であるため、すべての対が無限回試されステップサイズが Robbins–Monro 条件を満たすとき、表形式の Q 学習は Q^* に収束する。これは半勾配法である。遷移が確率的な場合、二乗時間差分誤差に対する勾配降下法は別の点へ収束してしまい、これが二重サンプリング問題であった。滑る湖では典型的な実行で最適な貪欲テーブルが得られ、そこに到達する速さはステップサイズスケジュールが決める。またターゲット内の最大値演算は Q 学習をオフポリシーにする一方で、推定値にノイズがあるうちは上向きのバイアスを生じさせる。同じ遷移を数え上げて推定モデル上で計画を行えば、モデルの保存と求解のコストと引き換えに、同じデータからより正確な価値が得られる。

14.3.6 演習問題

1. **バッチ TD 対モンテカルロ**。マルコフ報酬過程とは行動を持たない MDP であり、各状態が報酬と次状態に関する固定された分布を持つ。そのような過程の 1 つとして、2 つの非終端状態 x と y を持ち、$\gamma = 1$ であるものを考える。あるバッチに 8 つのエピソードが記録されているとする。1 つは x から y へ報酬 0 で遷移したのち報酬 0 で終端し、6 つは y から始まって報酬 1 で終端し、1 つは y から始まって報酬 0 で終端する。TD(0) は Equation 14.31 から最大値演算を除いた状態価値の形式 $V(s) \leftarrow V(s) + \alpha\big(r + \gamma V(s') - V(s)\big)$ である。

    1. このバッチから $V(x)$ と $V(y)$ のモンテカルロ推定値を計算し、さらに小さなステップサイズでこのバッチを何度も繰り返し再生したときに TD(0) が収束する値を求め

よ。
2. 観測された 8 つの収益に対する二乗誤差が小さいのは、2 組の推定値のどちらか。
3. 回数を数えることでこの過程の遷移確率と期待報酬を推定し、そのベルマン方程式を厳密に解け。得られた解は 2 組の推定値のどちらを再現するか。
4. 今後 x から始まるエピソードの収益をより良く予測すると期待されるのはどちらの組か。また、それは過程に関するどのような仮定のもとでか。

Richard S. Sutton および Andrew G. Barto の『Reinforcement Learning: An Introduction』Example 6.4 を改変。

2. **終端として扱われた打ち切り。**状態が 1 つ、行動が 1 つで、毎ステップ報酬 1 を与え、決して終端しないタスクを考える。時間制限によって各エピソードは H ステップ後に打ち切られるものとする（`FrozenLake-v1` が 100 ステップ後に打ち切るのと同様である）。誤った実装が、ブートストラップのマスクに `terminated or truncated` を（`terminated` ではなく）用いたとする。

   1. 減衰するステップサイズを用いると、推定値は 1 エピソードの H 回の遷移にわたる期待更新量が 0 になる値へ収束する。この値を γ と H の関数として求め、正しい価値 $1/(1-\gamma)$ と $\gamma = 0.95$、$H = 100$ において比較せよ。この誤った実装は実質的にどの割引率を使っていることになるか。
   2. 今度は逆に時間制限がタスク自体の一部であり、収益が本当に H ステップ後に終わるものとする。単一の数値ではこの状態の正しい価値を表せないことを示せ。問題 5（14.1 節）と同様に、残りステップ数 h を状態に加えよ。各状態 (s, h) の価値を求め、この場合にはマスク `terminated or truncated` が正しいことを示せ。

3. **貪欲な SARSA と Q 学習。**SARSA は Equation 14.31 の最大値を、行動方策が選んだ行動 a'（状態 s' において）の価値で Equation 14.37 のように置き換える。ある実装では、a' を選んだ後に $Q(s, a)$ を更新し、次のステップでその a' を実行する。両手法とも現在のテーブルに対して貪欲に行動を選び、同値の解消も同じ固定された規則で行うとする。このとき SARSA と Q 学習は同じ行動選択と同じ更新を行う同一のアルゴリズムになるか。もし異なるなら、次のステップで両手法が異なる行動を選ぶような滑らない湖における遷移とテーブルの値を構成せよ。

Richard S. Sutton および Andrew G. Barto の『Reinforcement Learning: An Introduction』Exercise 6.12 を改変。

4. [code] **二重サンプリングの反例。**●各状態につき行動が 1 つだけの 3 状態 MDP を構築せよ。状態 s では、行動は報酬 0 を与え、u_1 または u_2 へそれぞれ確率 $1/2$ で遷移する。u_1 では報酬 1 を与えて終端し、u_2 では報酬 0 を与えて終端する。3 つの要素を $Q(s)$、$Q(u_1)$、$Q(u_2)$ と書く。

   1. サンプル目的関数 Equation 14.33 の 1 項をテーブルに関して微分せよ。非ゼロの導関数を持つのはどの要素であり、半勾配更新 Equation 14.31 が捨てるのはそのうちのどの導関数か。
   2. 3 つの要素に対する Q^* を書き下せ。
   3. d を 3 つの対の上の一様分布として母集団目的関数 Equation 14.34 を明示的に書き下し、γ の関数として閉形式で最小化せよ。最小点は Q^* に比べてどの要素をどの方向へ歪めるか。
   4. 3 つの対を同じ頻度で訪問し、減衰するステップサイズを用いて、半勾配更新と (a) の全導関数に対する確率的勾配降下法の両方を実行せよ。それぞれが (b) と (c) で予測した点へ収束することを確かめよ。
   5. 環境を状態 s にリセットでき、s を訪問するたびに独立な 2 つの次状態が得られるとする。一方をターゲットに使い、もう一方を (a) の導関数に使え。得られる確率的勾

配が二乗ベルマン残差の勾配に対する不偏推定量になることを示し、Q^* へ収束することをシミュレーションで確かめよ。

5. [code] **有限予算下でのステップサイズ**。●本節では、定数のステップサイズをスケジュール $\alpha = 1/(1 + c\,n)$（$c = 0.1$ および $c = 1$、ただし n は今回を含めたその要素への訪問回数）と比較した。解析のため、1 つの要素が受け取るターゲットは平均 m、分散 σ^2 の独立な抽出値として扱え。
   1. 定数 α について、多数回の更新後における推定値の分散を求めよ。それを $\alpha = 0.9$ において $Q(s_0, \text{left})$ に対して評価せよ。ただしターゲットは $r + \gamma V^*(s')$（s' は本節の `mdp.P[0, a0]` から抽出し、`V_star` も本節のものを使用）とし、得られた標準偏差を、本節で表示された厳密値 0.180 の周りでの定数ステップ最終推定値のばらつきと比較せよ。この要素では解析のどの仮定が破れており、残りのばらつきはどこから生じているか。
   2. $c = 1$ のとき、n 回の訪問後の推定値を初期値と n 個のターゲットの重み付き和として書き下せ。0 による初期化にはどの重みがかかるか。
   3. 一般の c について、k 番目のターゲットが n 回の訪問後に持つ重みを積の形で書き下し、その積の対数を積分と比較することで、重みが $k^{1/c-1}$ のオーダーで（k の関数として、大きな $k \le n$ に対して）増大することを示せ。$c = 0.1$ は初期の不正確なテーブルから計算されたターゲットをどのように扱うか。
   4. 滑る湖において $c \in \{0.01, 0.03, 0.1, 0.3, 1\}$ を各 5 シードでスイープし、横軸を c として、最終誤差 $|\widehat{Q}(s_0, \text{left}) - Q^*(s_0, \text{left})|$ と全要素にわたる上限ノルム誤差 $\|\widehat{Q} - Q^*\|_\infty$ の両方をプロットせよ。曲線の形状を (a) と (c) を用いて説明せよ。2 つの誤差指標は同じ c を選ぶか。

6. [code] **サンプル対バックアップ**。●`Q_star` を正解として用いよ。
   1. 滑る湖での `q_learning` の各エピソード後に $\|\widehat{Q} - Q^*\|_\infty$ を記録し、台帳 `visits` が数える環境ステップ数に対してプロットせよ。さらに同じ軸上に、Q_k を `mdp.backup`（`d2l.value_iteration` の k 回目の反復に対するもの）とおき、1 スイープあたり 64 回のベルマンバックアップ数に対して価値反復法の誤差をプロットせよ。Q 学習が 4000 エピソード後に到達する誤差に価値反復法が達するには、何回のバックアップが必要か。
   2. `q_learning` を `Counts(env)` 上で実行し、100、400、1000、4000 エピソード後に本節と同様に `learned_model` を `d2l.value_iteration` で解け。同じ環境ステップ数における推定モデルの行動価値の誤差をプロットに追加せよ。4000 エピソード後の Q 学習の誤差にモデルが到達するには何環境ステップ必要であり、モデルを 1 回解くごとに何回のバックアップを要するか。
   3. 価値反復法はモデルのバックアップを消費し、Q 学習は環境ステップを消費し、(b) の推定モデルは両方を消費する。1 環境ステップが b 回のバックアップと同じ計算コストを要するとする。測定結果を用いて、b のどの範囲で 3 つの手法のそれぞれが 4000 エピソード後の Q 学習の誤差に最小の総コストで到達するかを求めよ。遷移カーネルを読み取れないエージェントにも利用可能なまま残るのはどの手法か。

7. [code] **2 状態タスクにおける最大化バイアス**。●すべてのエピソードは状態 s_0 から始まり、そこには 2 つの行動がある。**右**は報酬 0 で終端し、**左**は状態 s_1 へ報酬 0 で遷移する。s_1 では m 個の行動のそれぞれが $\mathcal{N}(-0.1, 1)$ から抽出された報酬を与えて終端するため、**左**を選ぶのは誤りである。$\gamma = 1$、定数ステップサイズ $\alpha = 0.1$、初期値 0、および $\epsilon = 0.1$ の `epsilon_greedy` を用いよ。
   1. $m = 10$ に対して表形式の Q 学習を実装し、s_0 で**左**を選ぶ実行の割合（1000 回の実行・各 300 エピソードにわたる平均）をエピソード番号に対してプロットせよ。ϵ 貪

欲エージェントが Q^* に基づいて行動した場合、この割合はいくらになるか。

2. ダブル Q 学習 (Hasselt 2010) は 2 つのテーブル Q_1 と Q_2 を保持し、ϵ 貪欲に Q_1+Q_2 に基づいて行動し、各遷移で公平なコインを投げる。表が出れば次式で更新し、

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha(r + \gamma\, Q_2(s', \text{argmax}_{a'} Q_1(s',a')) - Q_1(s,a)), \tag{14.38}$$

裏が出れば 2 つのテーブルを入れ替えた同じ更新を適用する。この曲線をプロットに追加せよ。

3. 左を通じて到達しうる報酬はすべて負の平均を持つ。それにもかかわらず Q 学習による $Q(s_0, \text{left})$ の推定値が正になる仕組みを説明せよ。$m = 1$ と $m = 100$ に対して Q 学習の曲線がどう変化するかを予測し、実際に確認せよ。$m = 100$ における効果のうち、どの部分がノイズを含む推定値に由来し、どの部分が一度も試されていない行動に由来するか。

Richard S. Sutton および Andrew G. Barto の『Reinforcement Learning: An Introduction』Example 6.7 を改変。

8. [code] **崖歩きにおける SARSA と Q 学習。**●`CliffWalking-v1` では、エージェントは 4×12 のグリッドを左下の隅から右下の隅まで横断する。毎ステップ -1 のコスト（報酬）がかかり、下端に沿った崖へ踏み込むと -100 がかかって開始位置へ戻される。$\gamma = 1$、定数ステップサイズ $\alpha = 0.5$、定数 $\epsilon = 0.1$ を用いよ。

   1. `sarsa` を `q_learning` のコピーとして作成せよ。ただし、更新前に次の行動 a' を `d2l.epsilon_greedy` で選び、それをターゲット Equation 14.37 に使い、次のステップでその行動を実行するようにせよ。両手法を 10 シードで 500 エピソード訓練し、1 エピソードあたりの収益をプロットせよ。
   2. 各最終テーブルの貪欲方策を開始位置からたどれ。各手法はどの経路を学習し、その長さはいくらか。学習中により多くの報酬を集めるのはどちらの手法であり、2 つの問いの答えが異なるのはなぜか。
   3. 訓練中に ϵ を 0 まで減少させよ。学習される 2 つの経路はどうなり、極限で各手法はどの価値を学習するか。

Richard S. Sutton および Andrew G. Barto の『Reinforcement Learning: An Introduction』Example 6.6 を改変。

14.4 探索

お気に入りのレストランに足を運べば満足のいく食事ができるが、新しい店に入ればもっと良い食事に出会えるかもしれないし、期待外れに終わるかもしれない。しかし、お気に入りの店にばかり通い続けていては、それを確かめる機会は永遠に訪れない。選んでみて初めてその価値がわかる選択肢のなかから繰り返し選択する場面では、いつでも同じ問いが生じる。新しいものをどのくらいの頻度で試すべきか、そしていつ試すのをやめてよいのかという問いである。

William Thompson は 1933 年に医学の文脈でこの問題を提起した。これまでの患者の経過を踏まえたとき、次の患者には 2 つの治療法のどちらを施すべきかという問題である (Thompson 1933)。Herbert Robbins は 1952 年に、毎回どちらからサンプリングするかを選びながら 2 つの母集団から繰り返しサンプリングする問題としてこれを定式化し直した (Robbins 1952)。スロットマシンの俗称である片腕バンディットにちなんで、この問題はまもなく **2 腕バンディット**（two-armed bandit）と呼ばれるようになり、選択肢が 2 つより多い場合は**多腕バンディット問題**（multi-armed bandit）と呼ばれる。1985 年には Tze Leung Lai と Robbins が、学習のコストは試行回数とともに増大せざるを得ないものの、その試行回数の対数と同程度に小さく抑えられることを示した (Lai and Robbins 1985)。

14.3 節の Q 学習も同じ問いに直面し、素朴な方法でこれに答えていた。Q 学習が学習できるのは表のうち実際に試した要素だけであり、表で低く評価されている指示も試し続けるために、確率 ϵ でランダムに行動した。この規則には目に見えるコストがある。20 ステップに 1 回の割合でランダムな行動を強いられると、滑りやすい湖における最適方策（optimal policy）の成功率は 73% ではなく約 54% に落ちてしまう。これは試す量として適切なのだろうか。別の規則を使えば、取りこぼしをより少なく抑えつつ同程度のことを学習できるのではないだろうか。

この問いは、検証器（verifier）を用いて訓練される言語モデルにおいても再び重要になる。モデルはプロンプトに対する複数の回答をサンプリングし、判定器がそれらにスコアをつけ、モデルはより高いスコアを得た回答の方向へと移動する。もしすべての回答が同じスコアであれば向かうべき方向は存在せず、より良い回答を見つけ出す手段はモデル自身のランダム性だけになる。

まず、このトレードオフのコストを正確に測定できる単一状態の問題、すなわちバンディットにおいて検討する。続いて、未試行の対を試すのに長い行動の系列が必要になり得るマルコフ決定過程（MDP）に戻り、最後に 14.1 節の検証器タスクを扱う。

```
import gymnasium as gym
```

14.4.1 多腕バンディット問題

バンディットとは、状態を 1 つだけ持つ MDP である。各ラウンドでエージェントは複数の腕から 1 本を引いてランダムな報酬を受け取る。どの行動をとっても次に起こることは変化せず、報酬の遅延もないため、残る問いは「どの腕が最良か」という 1 点のみである。エージェントは実際に腕を引くことでしかこの問いに答えられず、劣った腕を 1 回引くことは、その 1 回分を最良の腕に費やせなかったことを意味する。

14.4.1.1 腕とリグレット

ここではベルヌーイ報酬を持つ 10 本の腕を用いる。腕 a は 1 を確率 μ_a で返し、それ以外のときは 0 を返す。最良の腕である 3 番目の腕は $\mu^* = \max_a \mu_a = 0.90$ であり、2 番目に良い腕は 0.60 である。エージェントは μ_a を知らない。$n(a)$ 回だけ腕 a を引いた時点で、エージェントは経験平均（empirical mean）$\hat{\mu}(a)$、すなわちその試行のうち 1 を返した割合を知る。**バンディット規則**（bandit rule）は、これらの集計値から次に引く腕を決める。関数 `bandit_run` は 1 つの規則を `num_pulls` ラウンド実行し、すべての乱数を乱数生成器 `rng` から引き出す。また、すべての腕について集計値 `wins` と `count` を保持し、それらを規則に渡す。

```
arms = np.array([0.50, 0.42, 0.90, 0.25, 0.55, 0.38, 0.60, 0.32, 0.50, 0.45])

def bandit_run(arms, rule, num_pulls, rng):
    """One state, one step: pull an arm, log the shortfall, update tallies.
    Returns the cumulative regret after each pull."""
    wins, count = np.zeros(len(arms)), np.zeros(len(arms))
    gap = np.empty(num_pulls)
    for t in range(num_pulls):
        a = rule(wins, count, t, rng)
        count[a] += 1
        wins[a] += float(rng.random() < arms[a])
        gap[t] = arms.max() - arms[a]
    return np.cumsum(gap)
```

この関数は、各試行の報酬ではなく不足分を記録する。成功率は方策（policy）の良さを表すが、学習器については学習にどれだけのコストがかかったかを知りたい。腕 a_u をラウンド u で引くと、期待値において最良の腕を $\mu^* - \mu_{a_u}$ だけ下回り、t 回引いた後の**リグレット**はこれらのギャップを足し合わせたものになる。

$$\mathrm{Regret}(t) = \sum_{u=1}^{t} \left(\mu^* - \mu_{a_u}\right) = \sum_{a} \Delta_a \, n_t(a), \qquad \Delta_a = \mu^* - \mu_a. \tag{14.39}$$

2 番目の式は試行を腕ごとにまとめたものであり、各腕のギャップ Δ_a にその腕を引いた回数 $n_t(a)$ を掛けている。リグレットとは、最初から最良の腕を知っていたエージェントと比べて失った量である。平均 0.50 の 1 本の腕をずっと引き続ける規則は、1 回あたり 0.40、合計では 800（2000 回引いた時点）を失い、そのリグレットは t に対して線形に増大する。規則のリグレットが t より緩やかに増大するならば、その規則が劣った腕を引く頻度は次第に減っていき、1 回あたりの平均リグレットは 0 に収束する。

14.4.1.2 貪欲規則とイプシロン貪欲規則

14.3 節の規則はここでもそのまま機能する。`d2l.epsilon_greedy(q, epsilon, rng)` は価値推定値の 1 行（ここでは 10 個の経験平均）を受け取る。確率 ϵ で一様ランダムに腕を 1 本選び、それ以外のときは推定値が最大の腕を選ぶ（同値の場合はランダムに選ぶ）。1 度も引かれていない腕の経験平均は 0 である。$\epsilon = 0$ のとき、この規則は**貪欲**（greedy）である。固定された ϵ は永遠に同じ割合で探索を続けるのに対し、**アニーリング**規則は時間とともに ϵ を下げていく。ここでは $\epsilon_t = \min(1, 20/t)$ をラウンド t で用いる。これは最初の 20 ラウンドではランダムに引き、その後は次第に頻度を下げていく。このスケジュールはラウンド数のみに依存し、エージェントが学習した内容には一切依存しない。

これらの規則はいずれも、価値推定値の 1 行を行動上の分布へと変換する。Figure 14.10 に、湖での 4 つの移動に対応する 4 つの行動価値の行に対する例を示す。貪欲規則は最大の推定値にすべての確率を割り当て、ϵ 貪欲規則は確率の割合 ϵ をすべての行動に一様に分配する。3 つ目の規則である**ボルツマン探索**（Boltzmann exploration）は、各行動の確率を $\exp(\hat{Q}(s,a)/\beta)$ に比例させる。これは 3.1 節のソフトマックスにソフトマックス温度 β を入れたものであり、β が大きければ一様分布に近づき、小さければ貪欲規則に近づく。ϵ 貪欲規則とは異なり、明らかに悪い行動よりも最良に近い行動をより頻繁に試す。

関数 `eps_rule` はスケジュール $t \mapsto \epsilon_t$ を、経験平均に `d2l.epsilon_greedy` を適用するバンディット規則へと変換する。関数 `regret_runs` は 20 回の独立した実行で規則を動かし、そのリグレット曲線を返す。異なる規則が互いに独立な実行になるよう、各実行はストリーム番号と実行インデックスから独自の乱数生成器のシードを設定する。関数 `report` は実行全体にわたる最終リグレットの中央値と四分位数を出力する。少数の実行が平均を大きく左右し得るため、ここでは中央値を報告する。

```
def eps_rule(schedule):
    """Epsilon-greedy on the empirical means, with epsilon = schedule(t)."""
    return lambda wins, count, t, rng: d2l.epsilon_greedy(
        wins / np.maximum(count, 1), schedule(t), rng)

def regret_runs(arms, rule, stream, num_runs=20, num_pulls=2000):
    """Regret curves of `num_runs` independent runs, shape [run, pull]."""
    return np.array([bandit_run(arms, rule, num_pulls,
                                np.random.default_rng([2026, stream, s]))
```

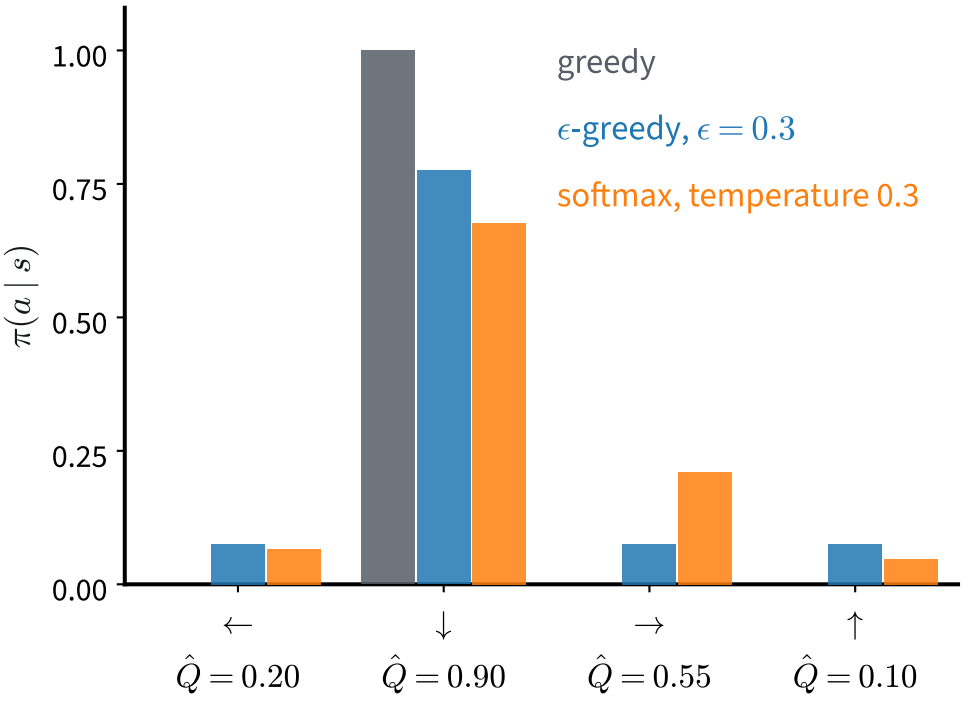


Figure 14.10: 湖での 4 つの移動に対する行動価値の 1 行 $\hat{Q}(s, \cdot) = (0.20, 0.90, 0.55, 0.10)$ に適用した 3 つの探索規則。貪欲規則は最大の値にすべての確率を置く。ϵ 貪欲規則は $\epsilon = 0.3$ のとき各行動に 0.075 を与え、残りを最大の値に割り当てる。温度 0.3 のソフトマックスは行動をその値によって順序づける。

```
                    for s in range(num_runs)])

def report(name, r):
    q25, q50, q75 = np.percentile(r[:, -1], [25, 50, 75])
    print(f'{name:>16}: regret after {r.shape[1]} pulls: median {q50:3.0f}'
          f' (quartiles {q25:.0f} to {q75:.0f})')

rules = {'greedy': eps_rule(lambda t: 0.0),
         'epsilon = 0.1': eps_rule(lambda t: 0.1),
         'epsilon = 20/t': eps_rule(lambda t: min(1.0, 20 / (t + 1)))}
regret = {name: regret_runs(arms, rule, i)
          for i, (name, rule) in enumerate(rules.items())}
for name, r in regret.items():
    report(name, r)
```

```
        greedy: regret after 2000 pulls: median 800 (quartiles 700 to 980)
  epsilon = 0.1: regret after 2000 pulls: median 106 (quartiles 92 to 132)
 epsilon = 20/t: regret after 2000 pulls: median  46 (quartiles 44 to 53)
```

貪欲規則の結果は最悪であり、その理由は正確に導き出せる。すべての経験平均は 0 から始まるため、最初の試行ではランダムな腕が引かれる。それが 0 を返せば、その腕の平均は 0 のままであり、10 本すべての腕が同値のまま並ぶ。もし 1 を返せば、その平均は正になり、その後も永遠に正のままである。1 度の成功によって成功割合は 0 より大きく保たれる一方、他のすべての腕は依然として 0 のままだからである。したがって貪欲規則は、どれか 1 本が報酬を返すまでランダムに腕を引き、それ以降は他の腕を決して引かない。これらのランダムな試行のそれぞれにおいて、腕 a は確率 $1/10$ で選ばれて 1 を確率 μ_a で返すため、最終的に落ち着く腕が a となる確率は $\mu_a / \sum_b \mu_b$ である。最良の腕がこのくじに当たるのは確率 $0.90/4.87$、すなわち約 5 回の実行に 1 回の割合である。腕 a に落ち着いた実行はほぼすべての試行で Δ_a を失い、合計で約 $2000\,\Delta_a$ を失うため、期待リグレットは次のようになる。

$$\mathbb{E}\big[\mathrm{Regret}(2000)\big] \approx 2000 \sum_a \frac{\mu_a}{\sum_b \mu_b} \Delta_a \approx 707.$$

1 回の実行で失う量は $2000\,\Delta_a$ のいずれかの値をとり、0（約 5 回の実行に 1 回）から 1300 までに分布する。800 前後となる中央値は、平均 0.50 の腕に落ち着いた実行に対応する。

$\epsilon = 0.1$ の固定規則はほぼすべての実行で最良の腕を見つけるが、探索を決してやめない。ランダムな試行は最良の腕を平均ギャップ $0.90 - 0.487 = 0.41$ だけ下回る（ここで 0.487 は 10 個の μ_a の平均である）。したがって探索だけで 1 回あたり $0.1 \times 0.41 \approx 0.04$、すなわち約 83（リグレットの中央値 106 のうち）のコストがかかり、このコストは縮小しない。固定された ϵ のリグレットは t に対して線形に増大する。

アニーリング規則がランダムに腕を引く回数は約 112 回（2000 ラウンド中）であり、最初に 20 回、その後に約 $20 \ln(2000/20) \approx 92$ 回である。1 回あたり 0.41 として、そのコストは 46 となり、リグレットの中央値である 46 と一致する。リグレットのほぼすべては探索に費やされており、その合間の貪欲な試行は最良の腕に向けられている。ラウンド t までのランダムな試行回数は $20 \ln t$ のように増大するため、リグレットも同様に増大する。すなわち、ϵ_t を $1/t$ に比例させると対数リグレットが得られる。ただし定数が重要である。定数が小さすぎると、最良の腕と 2 番目に良い腕を見分ける前に探索をやめてしまう。Auer et al. (2002) の解析によれば、m 本の腕を持ち最小ギャップが $\Delta_{\min}$ である任意のバンディットにおいて、$\epsilon_t = \min(1, c\,m/(\Delta_{\min}^2 t))$（任意の $c > 5$）とすれば対数リグレットが保証される。今回の腕では $\Delta_{\min} = 0.30$ であるため、

これはここで用いた $20/t$ のほぼ 30 倍にあたる。ここでの定数は今回の腕に合わせて選んだものであり、この理論保証の対象外である。

14.4.2 楽観主義と事後サンプリング

ϵ 貪欲規則は盲目的に探索する。ランダムな試行は 2 番目に良い腕と同じ確率で最悪の腕に向かってしまい、100 回引かれた腕も 2 回しか引かれていない腕と同じ確率で再び試される。より優れた規則は、平均がまだ不確実であり最良である可能性が残っている腕へと探索を振り向ける。意思決定を変え得るのはそのような試行だけだからである。

14.4.2.1 信頼上限

これを実現する 1 つの方法が楽観主義である。すなわち、各腕をデータが許容する範囲で最も良いものであるかのように扱う。未試行の腕はどのような値もあり得るため、まず試される。何度も引かれた腕は精度の高い推定値を持つため、その推定値が高い場合に限り再び引かれる。**信頼上限**（upper confidence bound、UCB）規則は、各経験平均にその不確実性を測るボーナスを加え、その和が最大となる腕を引く (Auer et al. 2002)。

$$a_t = \operatorname*{argmax}_a \left[\hat{\mu}(a) + \kappa \sqrt{\log t \,/\, n(a)} \right]. \tag{14.40}$$

ボーナスは $1/\sqrt{n(a)}$（$n(a)$ 個の報酬の平均における誤差が縮小する速さ）に従って縮小し、また t とともに緩やかに増大するため、腕 a が引かれていない間もどの腕も永久に見捨てられることはない。ある腕が引かれるのは、その平均が高いからか、あるいはその平均が不確実だからである。$n(a) = 0$ ではボーナスが定義されないため、規則を開始する前にすべての腕を 1 回ずつ引く。Figure 14.11 に、ボーナスと 1 回の実行におけるインデックスを示す。

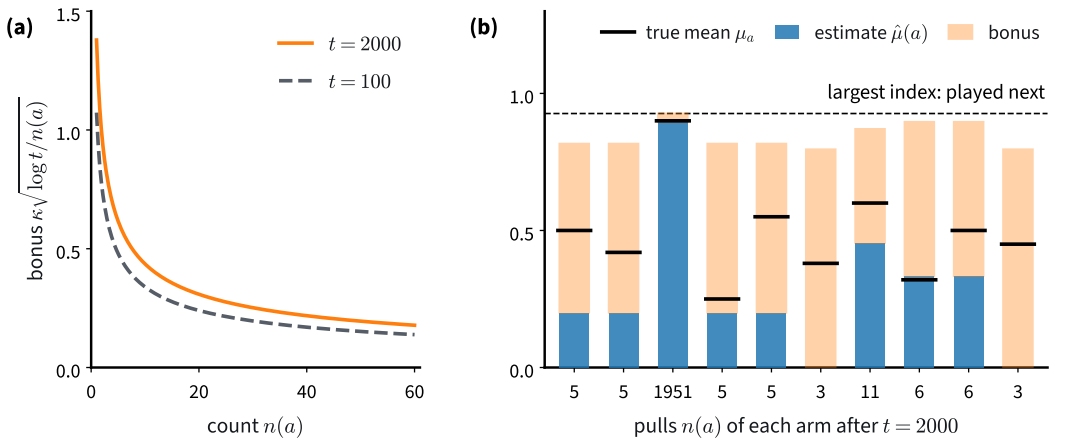


Figure 14.11: $\kappa = 0.5$ における不確実性のもとでの楽観主義。(a) ボーナス $\kappa\sqrt{\log t/n(a)}$ は腕の試行回数 $n(a)$ とともに縮小し、時刻 t とともに緩やかに増大する。(b) 10 本の腕に対して 2,000 回引いた後の UCB 規則の 1 回の実行。各棒は腕の推定値 $\hat{\mu}(a)$ の上にボーナスを積み上げたもので、その腕の試行回数が添えられており、黒い印は真の平均を表す。ほぼすべての試行はインデックスが最大の最良の腕に集まっているが、ほとんど試されていない腕もそれに近いインデックスを保っているため、$\log t$ が増大するにつれてそれぞれ再び試される。

ボーナスの大きさはヘフディングの不等式（Hoeffding's inequality）に由来する (Hoeffding 1963)。平均 $\hat{\mu}_n$（n 個の独立な $[0,1]$ 内の報酬の平均）が真の平均 μ を h より大きく上回る確率は高々 e^{-2nh^2} であり、下回る側についても同様である。$h = \kappa\sqrt{\log t/n}$ とおくと、真の平均がインデックスを上回る確率は高々 $t^{-2\kappa^2}$ となる（固定された回数 n に対して）。κ を大きくするとこのような失敗はまれになるが、悪い腕により多くの試行を費やすことになる。$\kappa = \sqrt{2}$ のとき、この規則は Auer et al. (2002) の UCB1 となり、そのリグレットは t に対して対数的に増大することが $[0,1]$ 内の報酬を持つあらゆるバンディットで成り立つ。同じことはすべての $\kappa > 1$ についても成り立ち (Bubeck and Cesa-Bianchi 2012)、演習問題 2 でこの上界を導出する。

14.4.2.2 トンプソンサンプリング

1933 年の Thompson の規則は、ボーナスを信念に置き換える。各腕についてその平均に関する事後分布（posterior distribution）を保持し、各事後分布からサンプルを 1 つずつ引き、サンプル値が最大となった腕を引く。報酬が $\{0,1\}$ の値をとり、$[0,1]$ 上の一様な事前分布（prior）を置く場合、w 回の成功（n 回の試行中）の後の事後分布はベータ分布 $\text{Beta}(1+w, 1+n-w)$ である。3 回の成功（4 回の試行中）を収めた腕の事後分布は $\text{Beta}(4,2)$ であり、平均は $2/3$ だが幅は広い。未試行の腕は一様な $\text{Beta}(1,1)$ を持ち、そのサンプル値が $\text{Beta}(4,2)$ からのサンプル値を上回る確率は $1/3$ である。Figure 14.12 にそのような 1 ラウンドを示す。このラウンドでは、未試行の腕が 18 回の成功（20 回の試行中）を収めた腕さえも上回っている。試行が積み重なるにつれて事後分布は狭まり、各腕はその腕が最良である事後確率（posterior probability）に従って引かれるようになる。トンプソンサンプリングもまた対数リグレットを達成し (Agrawal and Goyal 2012)、しかも選ぶべき定数を持たない。

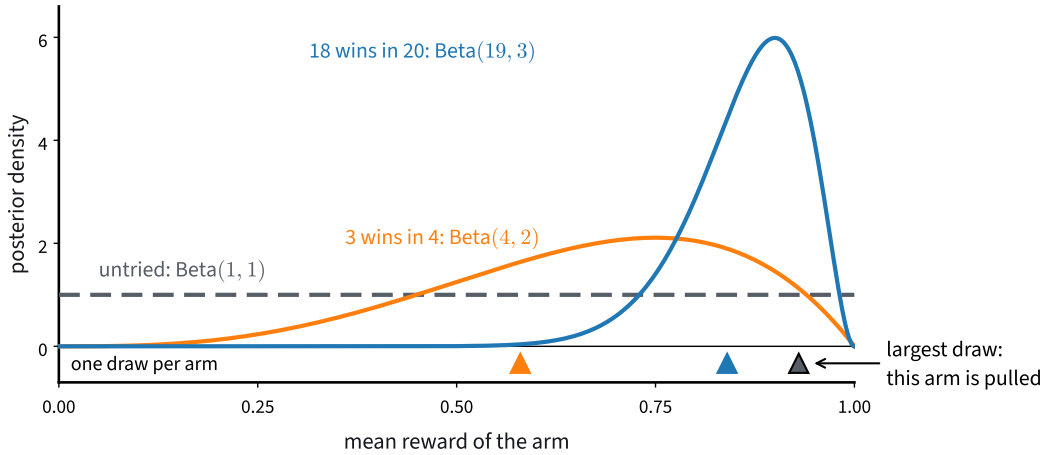


Figure 14.12: 3 本の腕（未試行の腕、3 回の成功を 4 回の試行で得た腕、18 回の成功を 20 回中で得た腕）に対するトンプソンサンプリングの 1 ラウンド。曲線は各腕の平均に対するベータ事後分布であり、腕が多く引かれるほど幅が狭くなる。規則は各事後分布から値を 1 つずつサンプリングし（三角形）、最大のサンプル値を持つ腕を引く。このラウンドでは未試行の腕が選ばれている。

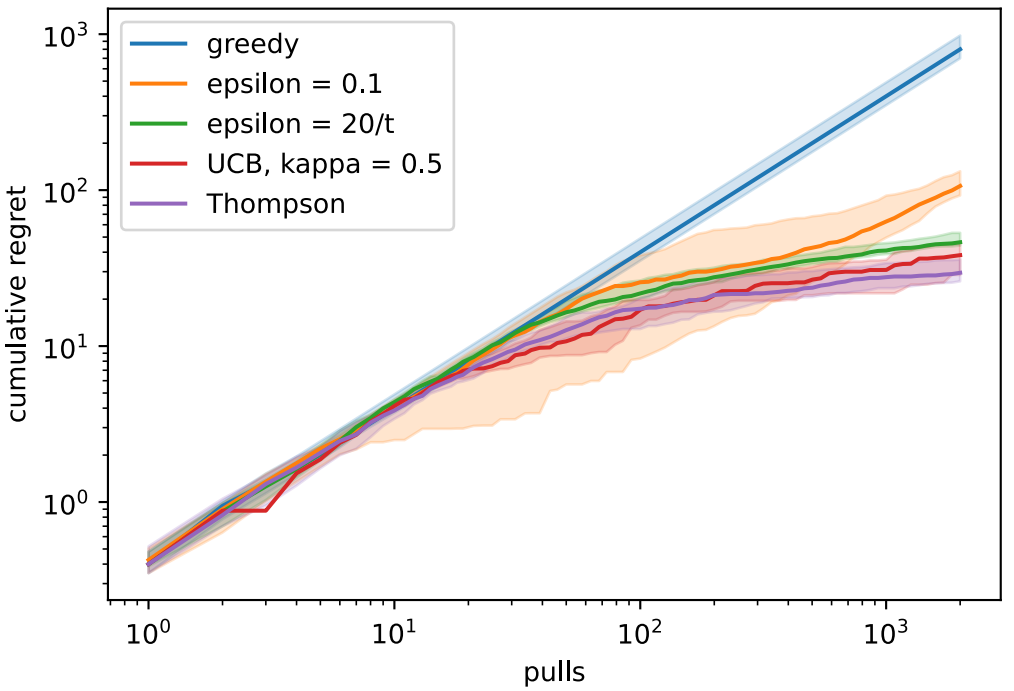
14.4.2.3 UCB とトンプソンサンプリングの比較

関数 `ucb(kappa)` は係数 κ を持つ UCB 規則を返し、`thompson` はベータ事後分布からサンプリングする。ここでは $\kappa = 0.5$（今回の腕に合わせて選んだ値であり、以下で検証する）の UCB とトンプソンサンプリングを実行し、5 つすべての規則をプロットする。`d2l.plot_curves` は実行全体に

わたる中央値と四分位数間の帯を描画する（ここでは対数軸上で等間隔にとった 100 個の試行時点において描画する）。

```
def ucb(kappa):
    """The UCB rule with coefficient kappa: each arm once, then the largest
    index mean + kappa * sqrt(log t / n)."""
    def rule(wins, count, t, rng):
        if (count == 0).any():          # play each arm once before pricing any
            return int(np.argmax(count == 0))
        bonus = kappa * np.sqrt(np.log(t + 1) / count)  # t + 1: round number
        return int(np.argmax(wins / count + bonus))
    return rule

def thompson(wins, count, t, rng):
    """Draw each arm's mean from its Beta posterior; play the largest draw."""
    return int(rng.beta(1 + wins, 1 + count - wins).argmax())

for i, (name, rule) in enumerate([('UCB, kappa = 0.5', ucb(0.5)),
                                  ('Thompson', thompson)], 3):
    regret[name] = regret_runs(arms, rule, i)
    report(name, regret[name])
pulls = np.unique(np.geomspace(1, 2000, 100).astype(int))  # even on a log axis
d2l.plot_curves({name: r[:, pulls - 1] for name, r in regret.items()},
                xlabel='pulls', ylabel='cumulative regret', x=pulls,
                xscale='log', yscale='log', band='iqr')
```

```
UCB, kappa = 0.5: regret after 2000 pulls: median  38 (quartiles 29 to 46)
        Thompson: regret after 2000 pulls: median  30 (quartiles 26 to 36)
```



どちらの規則も最終的なリグレットの中央値は約 30 から 40 となり、固定 ϵ の約 1/3 で、アニーリング規則よりも低い。両者は互いに近く、その差は実行間のばらつきよりも小さい。両対数軸上では、線形に増大するリグレットは傾き 1 の直線になる。貪欲規則と固定 ϵ はその傾きに近づく一方、UCB、トンプソンサンプリング、アニーリング規則はそこから折れ曲がって離れていく。2000 回の試行だけでは漸近的なレートを確定できないが、この曲がり方こそが対数的成長の現れである。

14.4.2.4 UCB の係数

$\kappa = 0.5$ という値は、理論が対象とする範囲 $\kappa > 1$ を下回っている。係数はどの程度影響するのだろうか。次のセルでは 3 つの係数のそれぞれについて 40 回の実行を試み、リグレットの中央値と最小値・最大値を出力する。

```
for j, kappa in enumerate([0.15, 0.5, np.sqrt(2)]):
    r = regret_runs(arms, ucb(kappa), 10 + j, num_runs=40)[:, -1]
    print(f'kappa = {kappa:.2f}: regret after 2000 pulls over 40 runs: median '
          f'{np.median(r):3.0f}, min {r.min():3.0f}, max {r.max():3.0f}')
```

```
kappa = 0.15: regret after 2000 pulls over 40 runs: median   8, min   4, max 709
kappa = 0.50: regret after 2000 pulls over 40 runs: median  38, min  21, max  59
kappa = 1.41: regret after 2000 pulls over 40 runs: median 204, min 173, max 241
```

2000 回の試行時点では、係数が結果を左右する。UCB1 ($\kappa = \sqrt{2}$) は 200 付近で終わり、$\kappa = 0.5$ のリグレットの 5〜6 倍になる。$\kappa = 0.15$ では中央値は約 10 と最小になるが、最大値が大きくなる。これほどボーナスが小さいと、最良の腕が最初の数回で不運にも失敗した実行では、劣った腕に落ち着いて 1 回あたり 0.3 以上を失い続けることがある。これは約 10 回の実行に 1 回の割合で起こり、中央値はこうした実行を隠してしまうが、平均は隠さない。

ホライズンが十分に長くなれば、理論値の κ が勝つと思うかもしれない。しかし今回の腕ではそうならない。10^5 回以上の試行（演習問題 4 で再現する）でも、$\kappa = \sqrt{2}$ のリグレットは $\kappa = 0.5$ の 6〜8 倍のままであり、どちらも $\log t$ のように増大する。この倍率はボーナスから説明できる。UCB は劣った腕 a を、そのボーナス $\kappa\sqrt{\log t / n(a)}$ がおよそ Δ_a に縮小するまで、すなわち約 $\kappa^2 \log t / \Delta_a^2$ 回引き続けるため、長い実行では 2 つのリグレットの比は $(\sqrt{2}/0.5)^2 = 8$ に近づく。理論が保証しているのは別のことである。$\kappa = \sqrt{2}$ は $[0, 1]$ 内の報酬を持つあらゆるバンディットで対数リグレットを達成するが、$\kappa = 0.5$ にはそのような保証がない。ヘフディングの不等式は最悪ケース、すなわち分散 $1/4$ の腕に対しても成り立たなければならない。ここでの最良の腕の分散は $0.9 \times 0.1 = 0.09$ であるため、その経験平均は不等式が想定するよりも正確であり、この問題ではより小さなボーナスで十分なのである。トンプソンサンプリングは、調整すべき定数なしに調整済みの UCB 規則に匹敵する。

14.4.2.5 リグレットの下界

リグレットはどこまで低く抑えられるのだろうか。Lai と Robbins は、ある族に属するすべてのバンディットでうまく機能する任意の規則は、t 回の試行のうちに各劣った腕 a を少なくともおよそ $\ln t / D(\mu_a \,\|\, \mu^*)$ 回は引かなければならないことを証明した (Lai and Robbins 1985)。ここで、次の式を考える。

$$D(p \,\|\, q) = p \ln \frac{p}{q} + (1 - p) \ln \frac{1 - p}{1 - q} \tag{14.41}$$

これは平均 p と q を持つベルヌーイ分布間の相対エントロピー（relative entropy）である。2 つの平均が近いほどこの値は小さくなるため、最良の腕に匹敵するほど良い腕は見分けにくく、より多くの回数引かなければならない。平均 0.60 の腕では $D(0.60 \| 0.90) = 0.31$ であり、約 $\ln 2000/0.31 \approx 24$ 回（ラウンド 2000 までに）引かなければならない。Equation 14.39 と合わせると、そのような規則のリグレットはいずれも少なくとも $\ln t$ のように増大する。UCB とトンプソンサンプリングはこのレートで増大するため、長い実行では定数倍を超えてこれらを上回る規則は存在しない。一方、リグレットが線形に増大する固定 ϵ はこのレートに到達しない。

14.4.3 MDP における探索

バンディットでは、どのラウンドでもすべての腕を引くことができる。しかし MDP では、状態行動対（state-action pair）はその状態からでなければ試すことができず、その状態に到達するには特定の行動の系列が必要になる場合がある。ランダムな行動が与えられた系列を生み出す確率は、その長さに対して幾何級数的に低下する。行動が 2 つの場合、長さ k の特定の行動系列が生じる確率は 2^{-k} となる。

14.4.3.1 状態のチェーン

チェーンを考えればこの現象を厳密に捉えられる。一列に並んだ N 個の状態があり、状態 0 から始まるものとする。行動 1 は 1 つ先の状態へ進み、行動 0 は開始状態へと戻り、最後の状態に入ると報酬 1 を得てエピソードが終了する（エピソードは $2N$ ステップで打ち切られる）。成功するには、エージェントは行動 1 を $N-1$ ステップ連続で選ばなければならない。1 度でも行動 0 を選ぶと開始状態に戻されてしまうからである（Figure 14.13）。

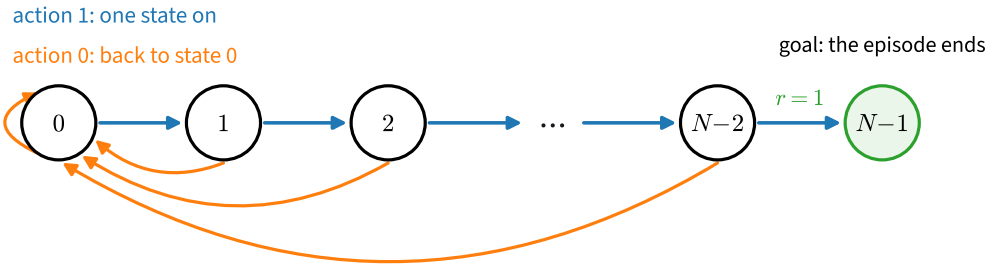


Figure 14.13: N 個の状態を持つチェーン。行動 1（青）は 1 つ先の状態へ進み、行動 0（オレンジ）はどこからでも状態 0 に戻り、唯一の報酬は最後の状態に入ったときの 1 であり、そこでエピソードが終了する。

関数 `chain` はこの MDP を `d2l.TabularMDP(P, r, gamma)`（14.1 節の配列による表現）として構築する。ここで `P[s, a]` は次の状態の分布であり、`r[s, a]` は期待報酬である。関数 `success_random` は、一様ランダムな行動が `horizon` ステップ以内に最後の状態へ到達する厳密な確率を計算する。この関数は 2 つの行動にわたって遷移カーネルを平均して $P_\pi(s' \mid s) = \frac{1}{2}\sum_a P(s' \mid s, a)$ とし、状態 0 からの状態分布を P_π によって 1 ステップずつ伝播させる。最後の状態は吸収状態（absorbing state）であるため、`horizon` ステップ後のその状態の確率が到達済みの確率となる。

```
def chain(N, gamma):
    """N states in a row: action 1 moves one state on, action 0 returns to
    the start; entering the last state pays 1 and ends the episode."""
    P, r = np.zeros((N, 2, N)), np.zeros((N, 2))
```

```
    P[:, 0, 0] = 1
    P[np.arange(N - 1), 1, np.arange(1, N)] = 1
    P[N - 1] = 0
    P[N - 1, :, N - 1] = 1              # the last state is absorbing
    r[N - 2, 1] = 1
    return d2l.TabularMDP(P, r, gamma)

def success_random(mdp, horizon):
    """Exact probability that uniformly random actions, started in state 0,
    reach the last state within `horizon` steps."""
    P_pi, d = mdp.P.mean(axis=1), np.eye(mdp.num_states)[0]
    for _ in range(horizon):
        d = d @ P_pi
    return d[-1]
```

`d2l.q_learning(seed, Q, visits, env, num_episodes, gamma, alpha, epsilon)`（14.3 節）は、表 `Q` を ϵ 貪欲な行動で収集した遷移からインプレースで更新し、`visits` に訪問回数を数え、各エピソードの収益（return）を生成する。この関数は `env.reset` と `env.step` を通じてのみ環境とやり取りし、クラス `ModelEnv` が `TabularMDP` にこれら 2 つのメソッドを与える。`reset` はエージェントを状態 0 に置き、`step` は `P[s, a]` から次の状態をサンプリングして、終端状態（terminal state）での終端と `max_steps` ステップ後の打ち切りを報告する。

```
class ModelEnv:
    """Gymnasium's reset/step on a TabularMDP: start in state 0, draw s'
    from P[s, a], end in a terminal state or after max_steps steps."""
    def __init__(self, mdp, terminal, max_steps):
        self.mdp, self.terminal, self.max_steps = mdp, terminal, max_steps
        self.rng = np.random.default_rng(0)

    def reset(self, seed=None):
        if seed is not None:
            self.rng = np.random.default_rng(seed)
        self.s, self.t = 0, 0
        return self.s, {}

    def step(self, a):
        s, self.t = self.s, self.t + 1
        self.s = int(self.rng.choice(self.mdp.num_states, p=self.mdp.P[s, a]))
        return (self.s, self.mdp.r[s, a], self.s in self.terminal,
                self.t >= self.max_steps, {})
```

ここでの学習器は 14.3 節の Q 学習に 1 つだけ変更を加えたものである。表のすべての要素をあらゆる収益の上界である 1 から開始し、エージェントは $\epsilon = 0$ で貪欲に行動する。これが**楽観的初期化**（optimistic initialization）であり、UCB の背後にある考え方と表を組み合わせたもの、すなわち未試行の対をあり得る最良の価値に見せる手法である。比較のため、次のセルではランダムな行動が最初に成功するまでの期待エピソード数（`success_random` で求めた確率の逆数）も出力する。ゼロの表から始める Q 学習は、どの行もすべて同値でランダムに選ばれるため、ϵ の値にかかわらず最初の成功まではこれとまったく同じように振る舞う（演習問題 1）。

```
def first_success(env, gamma, seed, max_episodes=100):
    """Episodes until greedy Q-learning from an optimistic table first
    reaches the goal (inf if it never does within max_episodes)."""
    # Optimistic: every entry starts at 1, an upper bound on every return
    Q = np.ones((env.mdp.num_states, env.mdp.num_actions))
    returns = d2l.q_learning(seed, Q, np.zeros_like(Q), env, max_episodes,
                             gamma, epsilon=lambda episode: 0.0)
    return next((k + 1 for k, ret in enumerate(returns) if ret > 0), np.inf)

for N in (4, 8, 12, 16, 20):
    mdp = chain(N, gamma=0.95)
    env = ModelEnv(mdp, terminal={N - 1}, max_steps=2 * N)
    first = [first_success(env, mdp.gamma, seed) for seed in range(20)]
    print(f'N = {N:2d}: episodes to the first success: random '
          f'{1 / success_random(mdp, 2 * N):5.0f} (mean), optimistic '
          f'{np.median(first):2.0f} (median, max {max(first):.0f})')
```

```
N =  4: episodes to the first success: random     2 (mean), optimistic  2 (median, max 2)
N =  8: episodes to the first success: random    23 (mean), optimistic  3 (median, max 4)
N = 12: episodes to the first success: random   273 (mean), optimistic  4 (median, max 5)
N = 16: episodes to the first success: random  3449 (mean), optimistic  6 (median, max 7)
N = 20: episodes to the first success: random 45590 (mean), optimistic  8 (median, max 10)
```

ランダムな行動では状態が 1 つ増えるごとに約 2 倍のエピソード数が必要になり、23（$N = 8$ のとき）、約 3,400（$N = 16$ のとき）、約 46,000（$N = 20$ のとき）となる。楽観的な学習器はわずかな回数で済み、約 8 回（$N = 20$ のとき）と、ほぼ N に比例して増える程度にとどまる。楽観主義が機能するのは、試した対に試行済みの印がつくからである。報酬を与えない対を 1 度試すと、その要素はターゲット $r + \gamma \max_{a'} Q(s', a')$（高々 $\gamma < 1$）に向かって移動するため、依然として 1 のままであるすべての未試行の要素を下回る。したがって貪欲規則はまだ試していない対を優先し、学習器は偶然に頼るのではなく体系的にチェーンをたどっていく。エピソード数は指数関数ではなく N の多項式オーダーで増大する。

14.4.3.2 カウントに基づくボーナス

楽観的初期化は、1 度目の試行で価値が下がるまでその対を探索する。確率的な MDP では 1 度の試行からわかることはほとんどなく、価値を知るには同じ対を繰り返し訪れる必要がある。**カウントに基づくボーナス**（count-based bonus）は、UCB 規則を MDP へと引き継ぐものである。各対の報酬に、訪問回数とともに $1/\sqrt{n(s,a)}$ のように縮小するボーナスを加え、価値バックアップ（value backup）によってそのボーナスをそこへ至る手前の状態へと伝播させることで、エージェントがあまり知らない対へ向かう経路をプランニングするようにする。モデルベース区間推定（model-based interval estimation）は学習されたモデルを用いてこれを実現し (Strehl and Littman 2008)、このようなボーナスを伴う Q 学習のリグレットは（MDP のサイズに関する多項式因子を除けば）ステップ数の平方根のように増大する (Jin et al. 2018)。表の代わりにネットワークを用いる場合、同じ状態を 2 度訪れることはまれであるため、実用的なエージェントでは訪問回数の代わりに学習された新規性の尺度（たとえば、状態の固定されたランダム関数を予測するよう訓練されたネットワークの誤差など）を用いる (Burda et al. 2019)。

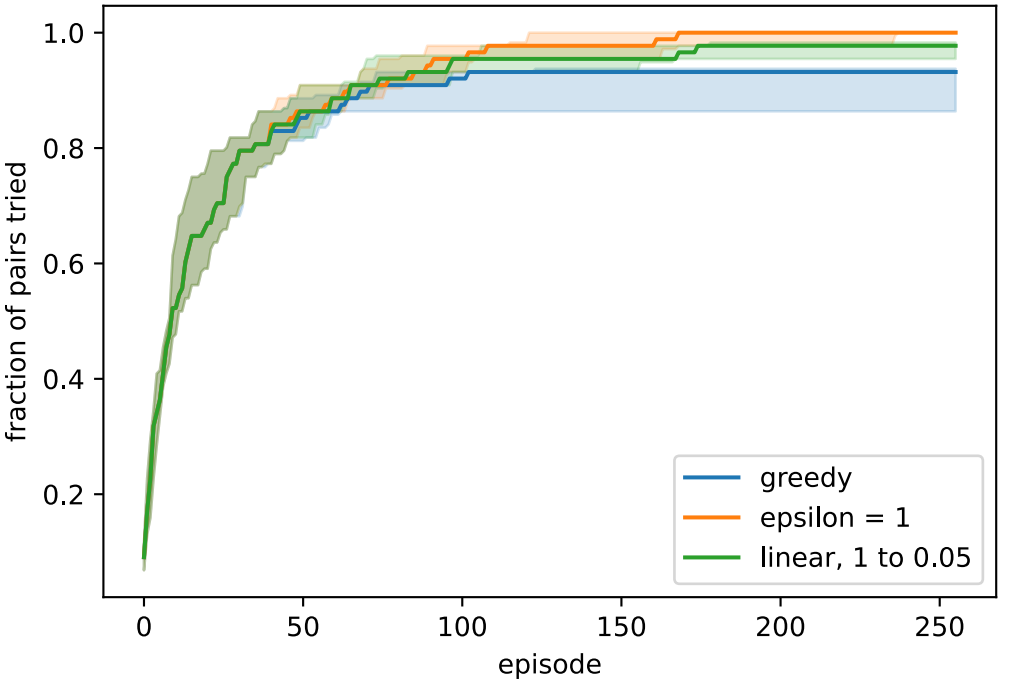
14.4.3.3 湖でのカバレッジ

湖のタスクにこうした手法は必要だろうか。14.3 節では、すべての移動が指示どおりに進む穏やかな湖での Q 学習において、ステップサイズ $\alpha = 0.9$ で 512 エピソードを実行し、得られた貪欲な表がこの湖を解いた。ここではその実験を 256 エピソードとし、3 つのスケジュールで繰

り返す。貪欲（$\epsilon = 0$）、常にランダム（$\epsilon = 1$）、そして `q_learning` のデフォルトである `d2l.linear_schedule(1.0, 0.05, 128)`（訓練の前半にわたって ϵ を 1 から 0.05 まで線形に下げる）の 3 つである。20 回の実行のそれぞれにおいて、次のセルは各エピソード後に 44 個の対（行動を選べる 11 マス、各 4 つの行動）のうち少なくとも 1 度試された割合を記録する。さらに、ゴールに到達した訓練エピソードの割合と、最終的な表の貪欲方策（greedy policy）がゴールに到達するかどうかも記録する（後者はこの決定論的な湖において `d2l.evaluate`（14.2 節）により 1 エピソードで検証する）。

```
def coverage_run(env, live, epsilon, seed, num_episodes=256, gamma=0.95):
    """One run of Q-learning with step size 0.9: the fraction of live pairs
    tried after each episode, the fraction of episodes won, and whether the
    final greedy policy reaches the goal."""
    Q = np.zeros((env.observation_space.n, env.action_space.n))
    visits, tried, returns = np.zeros_like(Q), [], []
    for ret in d2l.q_learning(seed, Q, visits, env, num_episodes, gamma,
                              alpha=lambda n: 0.9, epsilon=epsilon):
        tried.append((visits[live] > 0).mean())  # visits is updated in place
        returns.append(ret)
    solved = d2l.evaluate(env, lambda s, rng: int(Q[s].argmax()), 1)
    return tried, np.mean(returns), solved

calm = gym.make('FrozenLake-v1', is_slippery=False)
live = np.array([c not in b'HG' for c in calm.unwrapped.desc.flatten()])
schedules = {'greedy': lambda k: 0.0,
             'epsilon = 1': lambda k: 1.0,
             'linear, 1 to 0.05': d2l.linear_schedule(1.0, 0.05, 128)}
num_seeds, coverage = 20, {}
for name, epsilon in schedules.items():
    tried, won, solved = zip(*[coverage_run(calm, live, epsilon, seed)
                               for seed in range(num_seeds)])
    coverage[name] = np.array(tried)                  # (seed, episode)
    print(f'{name:>17}: pairs tried {np.median(coverage[name][:, -1]):.0%}, '
          f'episodes won {np.median(won):.0%} (medians); greedy policy '
          f'solves {np.sum(solved):.0f}/{num_seeds}')
d2l.plot_curves(coverage, xlabel='episode', ylabel='fraction of pairs tried',
                band='iqr')
```

```
          greedy: pairs tried 93%, episodes won 60% (medians); greedy policy solves 19/20
      epsilon = 1: pairs tried 100%, episodes won 2% (medians); greedy policy solves 19/20
linear, 1 to 0.05: pairs tried 98%, episodes won 58% (medians); greedy policy solves 19/20
```



穏やかな湖では、同値の場合のランダムな選択だけで貪欲規則に探索が生じる。10 個中約 9 個の対を試し、最終的な貪欲な表は他の 2 つのスケジュールの表と同様に、ほぼすべての実行で湖を解く。$\epsilon = 1$ では、エージェントはすべての対を試すものの、訓練エピソードのほとんどでゴールに到達しない。良い表を学習することと、学習中に報酬を集めることとは異なる目標であり、ϵ はこの 2 つの間でトレードオフをとる。3 つのスケジュールすべてを制約しているのは最初の成功である。それまではすべての要素が 0 であり、どのスケジュールもランダムに行動するため、同じシードであれば最初の成功はどのスケジュールでも同じエピソードで訪れる。ゴールは開始地点から 6 手の距離にあり、ランダムウォークでも数十エピソード以内に見つけられる。この湖は小さすぎてチェーンのような困難が現れないのである。一方、本節の冒頭で触れた滑りやすい氷の湖では、ある対を 1 度訪れただけではその対についてほとんど何もわからず、すべての対を 1 度ずつ試すだけでは不十分である（演習問題 1）。

14.4.4 検証器タスクにおける探索

14.1 節の検証器タスクには、4 つの算数のプロンプト x、それらすべてで共有される 9 個の候補応答 y、そして $r(x, y) = 1$（y がプロンプトの正解である場合）または 0（それ以外の場合）を返す検証器がある。参照方策（reference policy）π_{ref} はプロンプトごとに 1 行の選好度に対するソフトマックスであり、8 個の通常の候補に確率を均等に配分する。9 番目の候補（あらゆる答えを並べて保険をかけた回答であり、完全一致の検証器には決して受理されない）にはほとんど確率を与えない。$2 + 2$ と 3×4 の 2 つのプロンプトは候補のなかに正解を持つが、8×8 と 9×9 には一致する候補がない。サンプリングされた応答が正解となる確率は、プロンプト全体の平均で 0.062 である。

各ラウンドでプロンプトが到着し、1 つの応答にスコアがつけられ、エピソードが終了する。プロンプトはエージェントが観測できるものの影響を与えられない文脈であり、応答が腕にあたる。観測された文脈によって最良の腕が変わるバンディットは**文脈付きバンディット**（contextual bandit）と呼ばれる。エージェントはプロンプトごとに 1 つのバンディットに直面し、モデルを

用いることで学習した内容をプロンプト間で共有できる。

言語モデルは、このようなタスクをここまでの規則とは異なる方法で探索する。14.8 節で導出する一般的な訓練手法では、現在の方策からプロンプトに対する K 個の応答のグループをサンプリングしてスコアをつけ、グループ内の平均を上回るスコアを得た応答の確率を引き上げる一方、平均を下回る応答の確率を引き下げる。すべての応答が同じスコアになるグループでは、平均を上回るものも下回るものもないため、何も変化しない。方策があるプロンプトに確率 $p = \pi(\text{correct} \mid x)$ で正解するとき、K 個の独立な応答からなるグループですべてのスコアが同一になる確率は次のようになる。

$$(1-p)^K + p^K, \tag{14.42}$$

これは K 個すべてが不正解となる確率と、すべてが正解となる確率の和である。次のセルではプロンプトごとに 1,000 個のグループをサンプリングし、スコアがすべて同一になるグループの割合をこの式と比較する。比較は参照方策と、回答可能な 2 つのプロンプトをほぼ解いた方策（訓練された方策ではなく、各正解の選好度を 5 だけ引き上げて手作業で作ったもの）の双方で行う。配列 `R[x, y]` はすべての対に対する検証器のスコアを保持する。

```
task = d2l.VerifierTask()
R, rng = task.R, np.random.default_rng(0)     # R[x, y] = r(x, y)
for name, pi in [('reference', task.pi_ref),
                 ('solved', task.policy(task.theta_ref + 5 * R))]:
    p = (pi * R).sum(axis=1)            # chance that one response is right
    for K in (2, 8, 32):
        same = []                       # per prompt: all K rewards equal
        for x in range(len(task.prompts)):
            ys = rng.choice(len(task.resp), (1000, K), p=pi[x])
            same.append((R[x, ys] == R[x, ys[:, :1]]).all(axis=1).mean())
        print(f'{name:>9}, K = {K:2d}: identical {np.round(same, 2)}, '
              f'mean {np.mean(same):.2f}, '
              f'exact {np.mean((1 - p) ** K + p ** K):.2f}')
```

```
reference, K =  2: identical [0.78 0.78 1.   1.  ], mean 0.89, exact 0.89
reference, K =  8: identical [0.35 0.33 1.   1.  ], mean 0.67, exact 0.67
reference, K = 32: identical [0.01 0.02 1.   1.  ], mean 0.51, exact 0.51
   solved, K =  2: identical [0.89 0.93 1.   1.  ], mean 0.96, exact 0.96
   solved, K =  8: identical [0.67 0.7  1.   1.  ], mean 0.84, exact 0.85
   solved, K = 32: identical [0.21 0.21 1.   1.  ], mean 0.60, exact 0.61
```

サンプリングされた割合は式と一致する。どの K においても全グループの半分は同一になる。正解候補を持たない 2 つのプロンプトではすべての応答が失敗し、いくらサンプリングしても方策が生成できない答えは見つからないからである。回答可能な 2 つのプロンプトでは参照方策が確率 $p \approx 0.125$ で正解し、グループが大きいほど混在しやすくなる。サイズ 2 のグループでは約 78% が同一になり、サイズ 8 では 1/3、サイズ 32 ではほぼゼロになる。プロンプトがほぼ解けるようになるとグループは再び同一になりやすくなり、約 2/3 が $K = 8$ で同一となる。今度はほぼすべての応答が正解になるためであり、そのプロンプトでの学習は鈍化する。

この学習器は自身のサンプリングを通じてのみ探索する。グループが少なくとも 1 つの正解応答を含む確率 $1-(1-p)^K$ は、コードや推論のモデルを評価するのに用いられる **pass@K** スコアである (M. Chen et al. 2021)。方策がたいてい失敗するプロンプトにおいては、この値はグループから何かを学習できる確率にほぼ等しい。より高いソフトマックス温度でサンプリングすれば、より多くの応答に確率が広がり、方策がほぼ解いたプロンプトにおいてスコアが混在するグルー

プが増えるが、その代償として不正解も増える。正解の候補が存在しないプロンプトでは、これも助けにならない。

本節のすべての規則は、行動を実際に試すことでどの行動が良いかを学習するため、その試行には劣った腕を引くこと、ランダムウォークのエピソードを重ねること、あるいは何も教えてくれないグループを引くことといったコストが伴う。チェーンでは未試行の対に到達すること自体が難しく、検証器タスクでは方策が決して生成しない正解は永遠に見つからない。良い行動を手本として示すエキスパートがいればエージェントはこれらすべてを免れることができ、14.5 節ではそのような実演からの学習を扱う。

14.4.5 まとめ

自身の行動から学習するエージェントは、一見劣って見える行動も本当にそうであるかを確かめるために試さなければならず、リグレットは最初から最良の行動を知っていたエージェントと比べてこれにかかるコストを測る。10 本腕のバンディットでは、貪欲規則は最初に報酬を出した腕に落ち着いてしまい、固定された ϵ は永遠に 1 回あたり同じ量を失い続ける。ϵ を $1/t$ のように減衰させる規則、UCB、およびトンプソンサンプリングは、リグレットの増大を試行回数の対数程度に抑える。Lai と Robbins は、これが達成可能な最良のレートであることを示した。与えられた問題における UCB のリグレットはその係数によって決まり、理論上の係数はあらゆるバンディットに対して安全である反面、今回の問題では遅くなる。MDP では、ランダムな行動が必要な行動系列を見つけ出す確率はその長さに対して幾何級数的に低下するが、楽観的な価値を与えればエージェントは自ら未試行の対を探しに行く。検証器がサンプリングされた応答のグループにスコアをつける場合、すべて同じスコアになったグループは何の信号も持たず、方策自身のサンプリングが唯一の探索手段となる。

14.4.6 演習問題

1. [code] **探索スケジュール**。●●ゼロで初期化した表において、`epsilon_greedy` は同値の場合にすべて一様ランダムに選び、FrozenLake の報酬はすべて非負である。
    1. 最初の成功エピソードまでは表のすべての要素が厳密に 0 のままであり、それまではどの ϵ でも同じ振る舞いが生じることを示せ。このことから、ϵ を選ぶ基準としての最初の成功エピソードについて何が言えるか。
    2. 最初の成功の後、どの要素が正になるか。1 度正になった要素はその後も正のままであること、および貪欲規則（$\epsilon = 0$）は正の要素を含む行のゼロ要素を二度と試さないことを示せ。本節のカバレッジの実験では、この規則が穏やかな湖で 10 個中約 9 個の対を試し、そこで成功するよう学習することが示されている。滑りやすい氷の上でも同様に良い方策を学習するかどうか予測せよ。
    3. オプション引数 `epsilon`（`q_learning` の引数）はエピソードのインデックスから探索率への関数であり、`linear_schedule` が返す関数などがこれにあたる。滑りやすい湖において、4000 エピソード（14.3 節の設定）とし、ϵ を 0、0.1、0.5、1 に固定した場合、およびスケジュール `linear_schedule(1.0, 0.05, num_episodes // 2)`（`q_learning` が 14.3 節で用いているもの）の場合について、それぞれ 5 つのシードで実行せよ。各設定について、平均訓練収益、`d2l.evaluate` による 1000 エピソードにわたる最終的な貪欲方策の成功率、試された対の割合、および $\max_s \left| \max_a \hat{Q}(s,a) - V^*(s) \right|$ を報告せよ。予測は正しかったか。
    4. 各設定を訓練収益および貪欲方策の成功率によって順位づけせよ。2 つのランキングはどこで食い違い、その食い違いは何によって説明されるか。
2. **UCB の対数リグレット**。●●$\kappa = \sqrt{2}$ のとき、規則 Equation 14.40 は Auer et al. (2002) の UCB1 であり、その定理 1 は t 回引いた後の期待リグレット（$[0,1]$ 内の報酬を持つ腕に

対するもの）を次式で上から抑える。

$$\sum_{a:\,\Delta_a>0} \frac{8\ln t}{\Delta_a} + \left(1+\frac{\pi^2}{3}\right)\sum_a \Delta_a, \qquad \Delta_a = \mu^* - \mu_a. \tag{14.43}$$

この上界を導出せよ。用いる道具はヘフディングの不等式である。すなわち、平均 $\hat{\mu}_n$（n 個の独立な $[0,1]$ 内の報酬の平均であり、真の平均を μ とする）は次を満たし、

$$P(\hat{\mu}_n \geq \mu + h) \leq e^{-2nh^2}, \tag{14.44}$$

下側の裾についても同様の不等式が成り立つ。

1. 準最適な腕 a を固定し、$n_0 = \lceil 8\ln t/\Delta_a^2 \rceil$ とおく。腕 a がステップ $u \leq t$ において $n(a) \geq n_0$ のもとで引かれたならば、最良の腕のインデックスが μ^* 以下にあるか、あるいは腕 a の経験平均が μ_a をそのボーナスより大きく上回っているかのいずれかであることを示せ。
2. 両方の腕の試行回数を固定したとき、ステップ u におけるこれら 2 つの事象それぞれの確率を u^{-4} で上から抑えよ。回数に関するユニオンバウンドをとり、u について和をとることで、$\mathbb{E}[n(a)] \leq n_0 + \pi^2/3$ が t 回の試行後に成り立つことを示せ。
3. 小問 (b) とリグレットの分解 Equation 14.39 を組み合わせて定理を得よ。`arms` に対して $t = 2000$ でこの上界を評価し、本節の係数の比較実験で出力される中央値リグレット（200 付近、$\kappa = \sqrt{2}$ のとき）と比較せよ。証明のどのステップで最も大きく評価が緩んでいるか。
4. 一般の係数 κ について同じ議論を繰り返せ。どのような κ に対して小問 (b) のステップに関する和が収束するか。また、その答えは本節で選んだ $\kappa = 0.5$ について何を示唆するか。（回数に関するユニオンバウンドは緩い。ピーリング法による議論 (Bubeck and Cesa-Bianchi 2012) を用いればすべての $\kappa > 1$ に到達できる。）

3. **Thompson の事後分布。**●トンプソンサンプリングは、各腕の平均を $\text{Beta}(1+w, 1+n-w)$（w 回の成功が n 回の試行で得られた後）からサンプリングする。

1. これが $[0,1]$ 上の一様事前分布のもとでのベルヌーイ平均の事後分布であることを示せ。
2. 腕 A が 3 回の成功を 4 回の試行で収めており、腕 B は未試行であるとする。トンプソンサンプリングが B を引く確率は $1/3$ であることを示せ。より一般に、未試行の腕と比べるとき、未試行の腕を引く確率はもう一方の腕の事後平均を 1 から引いた値になることを示せ。
3. データが与えられたとき、トンプソンサンプリングは各腕を、その腕が最大の平均を持つ事後確率に等しい確率で引くことを示せ。
4. [code] 一様事前分布を、腕が悪いと予想する $\text{Beta}(1,9)$ と、腕が良いと予想する $\text{Beta}(9,1)$ にそれぞれ置き換えよ。2000 回引いた後のリグレットの中央値はどのように変化するか。2 つの事前分布のうち、楽観的初期化のように振る舞うのはどちらか。また、それはなぜか。

4. [code] **長いホライズン。**●●`bandit_run` は 1 度に 1 回の実行しか動かさない。

1. 多数の実行を同時に動かす版を作成せよ。すなわち、UCB とトンプソンサンプリングについて、`wins` と `count` を形状 (runs, arms) の配列として保持し、すべての実行の腕を 1 ステップで選ぶようにせよ。2000 回引いた時点でのその中央値が、実行間のばらつきの範囲内で本節の値と一致することを確かめよ。
2. 200 回の実行を、$\kappa \in \{0.15, 0.3, 0.5, \sqrt{2}\}$ の UCB およびトンプソンサンプリングについて 10^5 回の試行まで行い、リグレットの中央値を両対数軸上にプロットせよ。$\kappa = \sqrt{2}$ は $\kappa = 0.5$ に追いつくか。

3. $\kappa = 0.15$ について、中央値に加えてリグレットの平均もプロットせよ。なぜ両者は異なるレートで増大するのか。また、この規則を 1 度だけ実行するユーザーにとってのリスクを表すのは 2 つのうちどちらか。

5. [extended] **深い探索**。●●本節のチェーンを考える。状態は $0, \ldots, N-1$、開始状態は 0、行動 0 は状態 0 に戻り、行動 1 は状態を 1 つ進め、状態 $N-1$ に到達すると報酬 1 を得て終了し、エピソードは $2N$ ステップで打ち切られる。
   1. 一様ランダム方策が 1 エピソード内で成功する確率は高々 $2N \cdot 2^{-(N-1)}$ であり、したがって最初の成功までの期待エピソード数は少なくとも $2^{N-2}/N$ であることを示し、この下界をチェーンのセルが出力する厳密値と比較せよ。小問 (a)（演習問題 1）により、ゼロの表から始める ϵ 貪欲 Q 学習は最初の成功まで一様ランダムに振る舞うため、この下界は ϵ の値にかかわらず成り立つ。
   2. $N = 4$ のチェーンにおいて、楽観的な貪欲学習器の最初の数エピソードの推移をたどれ。各エピソードはどの要素を引き下げるか。そのパターンを用いて、必要となるエピソード数が N に対して高々多項式オーダーでしか増大しないことを論じよ。
   3. 両方の学習器を、100 ステップで打ち切られる `gym.make('FrozenLake-v1', map_name='8x8', is_slippery=True)` に移せ。楽観的な貪欲学習器と、スケジュール `linear_schedule(1.0, 0.05, num_episodes // 2)` のもとでのゼロ初期化学習器を、4000 エピソードおよび 20000 エピソードのそれぞれで各 5 シードずつ比較せよ。`d2l.evaluate` による貪欲方策の成功率、使用した環境ステップ数、および基準として価値反復法で計算した最適方策の成功率を報告せよ。確率的なマップでも楽観主義はその優位性を保つか。また、そのコストは何か。

   この課題には CPU 時間で数分かかる。

6. **何も教えてくれないグループ**。●方策が確率 p で正解するプロンプトにおいて、K 個の独立な応答からなるグループがすべて同一のスコアになる確率は $(1-p)^K + p^K$ である。
   1. 固定された $K \geq 2$ に対し、スコアが混在するグループが最も生じやすくなる p はいくらか。また、そのときの確率はいくらか。
   2. 小さい p に対し、グループに混在が生じる確率を少なくとも $1/2$ にする最小の K がおよそ $\ln 2/p$ であることを示せ。検証器タスクの参照方策（$p \approx 0.125$）および $p = 0.01$ についてこの値を計算せよ。
   3. ある訓練セットのプロンプトが持つ成功確率 p が $[0, 1]$ 上に一様に分布しているとする。サイズ $K = 8$ のグループのうちどれだけの割合が同一になり、どのプロンプトがそれらを生み出すか。
   4. `VerifierTask` の正解候補を持たない 2 つのプロンプトでは、どのような K やサンプリング温度に対してもすべてのグループが同一になる。これらのプロンプトで学習できるようにするための変更（タスクまたは学習器に対するもの）を 2 つ挙げ、それぞれが検証器単独では得られない何を必要とするかを述べよ。

14.5 デモンストレーションからの学習

「基礎」パートのすべてのモデルは、画像とそのクラス、文とその次の単語といったラベル付きの例から学習した。前節では、誰もラベルを与えてくれず、悪い行動を試しながら良い行動を見つけなければならないときに、エージェントがどれほど多くを失うかを見た。しかし多くの場合、誰かが手本を示すことができる。熟練した運転者が何を見て何をするか、すなわち各瞬間のカメラ画像と、それに応じて選ばれた操舵角を記録するとしよう。Fashion-MNIST で分類器を訓練したのと同じように、画像を入力、操舵角をラベルとして、これらの対から運転方策を訓練できるだろうか。

Dean Pomerleau は 1980 年代後半にカーネギーメロン大学でそのようなネットワークを構築した。1 つの隠れ層を持つネットワーク ALVINN は、道路の画像を操舵方向へ写像してバンを道路に沿って走らせた (Pomerleau 1989)。その後のバージョンでは、人間が運転する様子を観察することで、わずか数分で運転を学習した (Pomerleau 1991)。この手法は現在、**行動クローニング**（behavior cloning）と呼ばれている。環境のモデルも報酬も必要とせず、選択を記録できるエキスパートさえいればよい。これは今日でもエキスパートから学習する最も一般的な方法である。たとえば、厳選された応答で言語モデルをファインチューニングすることは、そこまでのテキストを状態、次のトークンを行動とする行動クローニングにほかならない。

Pomerleau はその弱点も見抜いていた。運転者がバンを車線の中央に保ち続けたため、記録には路肩の景色も、車線へ戻るための操舵の例も含まれていなかったのである。ひとたびネットワークが車線から逸れると、一度も見たことのない画像に直面することになったため、Pomerleau は視点をずらした道路画像を合成し、バンを元の車線へ戻す操舵角をラベルとして付与しなければならなかった。分類器は訓練時と同様の入力でテストされるが、方策は次に目にする入力を自ら選び出す。本節では、この違いが湖の環境でどれほどのコストをもたらすかを測定し、タスクが長くなるにつれてそのコストがどのように増大し得るかを示した上で、学習者が自力で到達した状態にエキスパートがラベルを付けるようデータの集め方を変える。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

14.5.1 行動クローニング

デモンストレーションとは、エキスパートから記録された軌跡である。行動クローニングは、エキスパートが訪問した状態とそこで取った行動だけを保持し、報酬も遷移モデルも使わない。湖の環境では、エキスパートは 14.2 節で価値反復法が計算する最適方策 π^* であり、したがって本節では π^* が最適方策とエキスパートの両方を表す。ここでは、どのコマンドもエージェントを意図した方向へ移動させる滑らない氷を用いる。このときエキスパートは、毎エピソードでスタートからゴールまで同じ 6 手の経路をたどり、その訪問状態は Pomerleau の記録における車線中央のように、湖を貫く細いチューブをなす。湖のモデルは既知であるため、あらゆる成功率を厳密に計算し、サンプリングノイズなしで学習者とエキスパートを比較できる。

14.5.1.1 分類への帰着

次のセルでは、滑らない湖を `TabularMDP`（配列 P と r、および 14.1 節の割引率 γ）として構築し、`d2l.value_iteration` で解いた上で、V^* にベルマン作用素を適用した `mdp.backup` から各状態の最良の行動を選ぶことでエキスパートを読み取る。関数 `demonstrations` は環境内でエキスパートをロールアウトし、観測した状態と取った行動だけを記録する。

```
gamma = 0.95
env = gym.make('FrozenLake-v1', is_slippery=False)
mdp = d2l.TabularMDP.from_gym(env, gamma)
V_star = d2l.value_iteration(mdp, num_iters=1000)[-1]
pi_star = mdp.backup(V_star).argmax(axis=1)

def demonstrations(env, expert, num_episodes):
    """Roll the expert; record only what it saw and what it did."""
    states, actions = [], []
    for _ in range(num_episodes):
        s, done = env.reset()[0], False
        while not done:
            states.append(s)
            actions.append(int(expert[s]))
            s, _, terminated, truncated, _ = env.step(actions[-1])
            done = terminated or truncated
    return np.array(states), np.array(actions)

demo_s, demo_a = demonstrations(env, pi_star, 3)
print(f'{demo_s.size} state-action pairs from 3 expert episodes, '
      f'covering states {np.unique(demo_s)}')
```

```
18 state-action pairs from 3 expert episodes, covering states [ 0  4  8  9 13 14]
```

滑らない氷の上ではどのエキスパートのエピソードも**下**、**下**、**右**、**下**、**右**、**右**とまったく同じになるため、3 エピソードからは同じ 6 個の対が 3 回繰り返された計 18 個のラベル付き対が得られる。状態 0 と状態 9 では**下**と**右**が同じ最短手数になり、同点の場合は**下**が選ばれる。エキスパートは氷のセル 1、2、3、6、10 を一度も訪問しない。

行動クローニングは、3.1 節が画像とラベルを扱ったのと同じようにこれらの対を扱う。すなわち、状態を入力、エキスパートの行動をクラスとし、条件付き分布 $\pi_\theta(a \mid s)$（パラメータ θ）を、対数尤度 $\sum_i \log \pi_\theta(a_i \mid s_i)$ の最大化、言い換えれば交差エントロピーの最小化によって当てはめる。必要なのは、パラメータを持つ方策と、それに対して勾配ステップを実行する仕組みだけである。

14.5.1.2 方策コンテナ

`Policy` は、状態を各行動につき 1 つのスコアへ写像するネットワーク `policy`（これにより $\pi_\theta(\cdot \mid s)$ は s におけるスコアのソフトマックスとなる）と、そのパラメータ用の Adam オプティマイザ `opt_pi` を保持する。`log_prob` は状態のバッチとそこで取られた行動に対して、行動クローニングが最大化する量である $\log \pi_\theta(a \mid s)$ を返す。

```
class Policy(nnx.Module):
    """A policy network and its optimizer; pi(.|s) is the softmax of the
    network's outputs at s."""
    def __init__(self, policy, lr=1e-2):
        self.policy = policy
        self.opt_pi = nnx.Optimizer(policy, d2l.shared_adam(lr),
                                    wrt=nnx.Param)

    def log_prob(self, obs, act, policy=None):
        """log pi(a|s). Gradients flow w.r.t. the module you differentiate;
```

```
        the update functions pass that module back in as `policy`."""
        policy = self.policy if policy is None else policy
        logp = jax.nn.log_softmax(policy(obs), axis=-1)
        return jnp.take_along_axis(logp, act[:, None], axis=-1).squeeze(-1)

    @classmethod
    def tabular(cls, num_states, num_actions, lr=0.1, seed=None):
        """One preference theta_{s,a} per state-action pair: an embedding."""
        rngs = nnx.Rngs(d2l.get_key() if seed is None else seed)
        return cls(d2l.Embed(num_states, num_actions, rngs=rngs,
                             embedding_init=nnx.initializers.zeros_init()),
                   lr)
```

コンストラクタ `tabular` はソフトマックス回帰モデルである。埋め込みテーブルが状態行動対ごとに 1 つの選好 $\theta_{s,a}$ を保持し、整数で表された状態がその行を指定し、ソフトマックスによって $\pi_\theta(\cdot \mid s)$ が得られる。ゼロ初期化により、初期方策は一様分布になる。

このテーブルは `d2l.Embed`（オンライン付録「ユーティリティ関数とクラス」）であり、その勾配をアクセラレータ上のアトミック加算ではなく 1 回の行列積として計算する `nnx.Embed` である。同じ行へのアトミック加算は到着順序が毎回変動するため、そのままでは 1 ステップの計算結果の下位ビットが変わり、大きなステップサイズで多数のステップを重ねると訓練実行全体の結果まで実行ごとに変わってしまう。

環境は NumPy の整数や配列をやり取りするため、3 つのメソッドがその境界で型変換を担う。`act` は呼び出し側から渡された NumPy の乱数生成器を用いて $\pi_\theta(\cdot \mid s)$ から行動をサンプリングするため、環境へ伝わる乱数性はすべて単一の明示的な発生源に集約される。このメソッドは、サンプリングしたエピソードの収益を平均する 14.2 節の `d2l.evaluate` の行動プロトコルに従う。`act_greedy` は最も確率の高い行動を返し、`log_prob_np` は対数確率を NumPy 配列として返す。

```
@d2l.add_to_class(Policy)
def act(self, obs, rng):
    """Sample an action; numpy in, int out, the acting protocol of evaluate."""
    logits = d2l.forward_np(self.policy, obs)
    probs = np.exp(logits - logits.max())
    return int(rng.choice(len(probs), p=probs / probs.sum()))

@d2l.add_to_class(Policy)
def act_greedy(self, obs, rng=None):
    """The most probable action; the same protocol as act."""
    return int(d2l.forward_np(self.policy, obs).argmax())

@d2l.add_to_class(Policy)
def log_prob_np(self, obs, act):
    """log pi(a|s) for numpy batches of states and actions, as numpy. The
    batch is padded to a bucket length, so that each operation compiles
    once per bucket rather than once per batch length."""
    obs_p, act_p, _ = d2l.padded(obs, act)
    return np.asarray(self.log_prob(jnp.asarray(obs_p),
                                    jnp.asarray(act_p)))[:len(act)]
```

`act` と `act_greedy` は `d2l.forward_np` を通してネットワークを実行する。`log_prob_np` と後述の `policy_step` は、異なるバッチサイズ間でコンパイル済みコードを再利用できるよう、`d2l.padded`

でバッチをバケット長までパディングする。また d2l.shared_adam は、同じ学習率を持つすべてのエージェントに単一のオプティマイザを割り当て、コンパイル済みの更新処理を共有する。これらの補助関数はオンライン付録「ユーティリティ関数とクラス」にある。log_prob が任意の引数 policy を受け取るのは、nnx.value_and_grad が与えられたモジュールに関して微分し、更新処理がそのモジュールを再び渡すからである。JAX はこのような変換の内部でのみ勾配を計算する。

状態集合が小さい方策に対しては、本章で行う厳密な計算のために、probs がテーブル全体 $\pi_\theta(a \mid s)$ を配列として学習者から読み出す。

```
@d2l.add_to_class(Policy)
def probs(self, num_states, num_actions):
    """pi(a|s) for every pair, as a (num_states, num_actions) numpy array."""
    s, a = np.divmod(np.arange(num_states * num_actions), num_actions)
    return np.exp(self.log_prob_np(s, a)).reshape(num_states, num_actions)
```

14.5.1.3 クローンの当てはめ

クローンの当てはめとは、$\frac{1}{n}\sum_i w_i \log \pi_\theta(a_i \mid s_i)$ においてすべての重み w_i を 1 とした勾配上昇法（gradient ascent）である。policy_step は、状態・行動・重みのバッチに対して、この目的関数に関するオプティマイザの更新を 1 ステップ実行する。

```
@nnx.jit
def _policy_pass(ac, obs, act, w, mask):
    """One step on -(1/n) sum_t w_t log pi(a_t|s_t) over the n real rows."""
    loss, grads = nnx.value_and_grad(lambda policy: -(
        mask * w * ac.log_prob(obs, act, policy)).sum() / mask.sum())(ac.policy)
    ac.opt_pi.update(ac.policy, grads)
    return loss

def policy_step(ac, obs, act, weight):
    """One ascent step on (1/n) sum_t w_t log pi(a_t|s_t) over a batch of
    n states and actions; the weights w arrive as numpy, that is, as data.
    The batch is padded to a bucket length, so the compiled pass is reused."""
    return float(_policy_pass(ac, *d2l.padded(obs, act, np.float32(weight))))
```

重みは NumPy 配列、すなわちデータとして渡されるため、そこに勾配は流れない。行動クローニングでは、重みはすべて 1 である。

当てはめはいつまで続けるべきだろうか。収束するまで訓練すれば、状態ごとに 1 行を持つテーブルはデモンストレーションされた 6 個の行動を完全にコピーし、滑らない氷の上ではクローンが毎回エキスパートの経路をたどることになる。しかし現実の分類器は厳密ではない。よく似た入力、限られた表現能力、ノイズを含むラベルといった要因により、訓練分布上にもある程度の誤差が残る。クローンにそのような誤差を持たせるため、clone は単位重みで policy_step を呼び出し、対全体で平均したときに π_θ がデモンストレーションされた行動と不一致になる確率が高々 $\varepsilon = 0.2$ になった時点で停止する。この停止規則が ε を固定するため、本節のすべての比較では当てはめ誤差が一定に保たれる。

```
def clone(mdp, states, actions, eps=0.2):
    """Behavior cloning: cross-entropy fit of pi(a|s) to the expert's
    choices (policy_step with every weight equal to one), stopped once pi
```

```
        disagrees with the demonstrated actions with probability at most eps."""
        learner = Policy.tabular(mdp.num_states, mdp.num_actions)
        weight = np.ones(len(states), np.float32)
        while 1 - np.exp(learner.log_prob_np(states, actions)).mean() > eps:
            policy_step(learner, states, actions, weight)
        return learner

    bc = clone(mdp, demo_s, demo_a)
    pi_bc = bc.probs(mdp.num_states, mdp.num_actions)
    for s in (9, 10):
        print(f'clone pi(.|s={s}): {np.round(pi_bc[s], 3)}')
```

```
clone pi(.|s=9): [0.061 0.818 0.061 0.061]
clone pi(.|s=10): [0.25 0.25 0.25 0.25]
```

Adam を 14 ステップ実行すると、不一致率は 0.182 になる。デモンストレーションされた各状態は同じ回数だけ現れて同じ勾配を受け取るため、状態 9 の行が示すように、6 つの行はいずれもエキスパートの行動に対する確率が 0.818、残り 3 つの行動に対する確率がそれぞれ 0.061 となって終了する。状態 10 は一度もデモンストレーションされていない。その行は勾配を受け取らず一様分布のままであるため、そこではクローンは 4 回に 3 回の割合でエキスパートと不一致になる。データを繰り返しても当てはめの結果は変わらない。6 個の対のコピー 3 つにわたる平均損失は 1 つのコピーにわたる平均と同じであるため、エキスパートの 1 エピソードから作ったクローンも 100 エピソードから作ったクローンも同じテーブルになる。

14.5.2 分布シフト

14.5.2.1 訓練と運用

分類器の性能保証は、訓練データの生成元である分布に対するものである。行動クローニングはエキスパートが訪問する状態で訓練される一方、運用時には自身の行動が生み出す状態に直面する。ここではクローンを 2 通りの方法で運用する。最も確率の高い行動を選ぶ貪欲な方法と、言語モデルが次のトークンをサンプリングするように π_θ から `act` でサンプリングする方法である。

湖のモデルは既知であるため、方策の状態分布はサンプリングではなく厳密に計算できる。$d_t^\pi(s)$ を、方策 π が状態 s に t ステップ後にいる確率とする。確率質量は最初すべてスタート状態に置かれ、1 ステップ進むごとに方策とカーネルを通して移動する。

$$d_{t+1}^\pi(s') = \sum_s d_t^\pi(s) \sum_a \pi(a \mid s)\, P(s' \mid s, a). \tag{14.45}$$

次のステップにおける s' の確率質量は、方策が選ぶ行動のもとで各状態からそこへ送り込まれる質量の和である。`visitation` は湖の 100 ステップという時間制限にわたってこの漸化式を実行する。ゴールと穴はエピソードを終了させて質量を保持するため、100 ステップ後にゴールにある質量が成功確率となる。サンプリングクローンについては、`d2l.evaluate` によるモンテカルロ実行でこの厳密値を検証する。

```
def visitation(mdp, pi, absorbing=None, T=100):
    """d_0, ..., d_T: the state distributions of the policy pi[s, a] from the
    start state, propagated exactly through P for FrozenLake's 100-step
    limit. Mass that enters an `absorbing` state stays there."""
    P_pi = np.einsum('sa,sat->st', pi, mdp.P)
```

```
        if absorbing is not None:
            P_pi[absorbing] = np.eye(mdp.num_states)[absorbing]
        d = [np.eye(mdp.num_states)[0]]
        for _ in range(T):
            d.append(d[-1] @ P_pi)
        return np.array(d)

    desc = env.unwrapped.desc.ravel()
    goal, holes = desc == b'G', desc == b'H'
    expert = np.eye(mdp.num_actions)[pi_star]               # one-hot rows
    greedy = np.eye(mdp.num_actions)[pi_bc.argmax(axis=1)]
    for name, pi in [('expert', expert), ('greedy clone', greedy),
                     ('sampling clone', pi_bc)]:
        d_T = visitation(mdp, pi)[-1]
        print(f'{name:>14}: reaches the goal {d_T[goal].sum():.3f}, '
              f'falls into a hole {d_T[holes].sum():.3f}')
    rate = d2l.evaluate(env, bc.act, num_episodes=1000,
                        rng=np.random.default_rng(0))
    print(f'sampling clone, 1000 episodes of bc.act: reaches the goal {rate:.3f}')
```

```
       expert: reaches the goal 1.000, falls into a hole 0.000
  greedy clone: reaches the goal 1.000, falls into a hole 0.000
sampling clone: reaches the goal 0.660, falls into a hole 0.340
sampling clone, 1000 episodes of bc.act: reaches the goal 0.656
```

エキスパートは毎回ゴールに到達し、貪欲クローンも同様である。デモンストレーションされた各行で最も確率の高い行動はエキスパートの行動と一致するため、貪欲クローンはエキスパートの経路をそのまま歩くからである。サンプリングクローンはエピソードの 66% でゴールに到達し、残りでは穴に落ちる。サンプリングした 1,000 エピソードの結果もノイズの範囲内でこれと一致する。滑らない氷の上ではクローンの選択以外にランダムな要素は存在しないため、経路から外れる一歩一歩はすべてクローン自身の誤った行動によるものである。

悪いのは分類器なのだろうか。次のセルでは、クローンがエキスパートとどれだけの頻度で不一致になるかを 3 つの状態集合で測定する。すなわち、訓練対、新たに生成したエキスパートの 100 エピソードの状態、そしてクローン自身が訪問する状態（訪問頻度で重み付けしたもの）である。続いて、エキスパートの 1、10、100 エピソードからクローンを作成する。

```
    disagree = 1 - pi_bc[np.arange(mdp.num_states), pi_star]   # per state
    fresh_s, _ = demonstrations(env, pi_star, 100)
    occupancy = (visitation(mdp, pi_bc)[:-1] * ~(goal | holes)).sum(axis=0)
    print('disagreement with the expert')
    print(f'  on the training pairs:        {disagree[demo_s].mean():.3f}')
    print(f'  on 100 fresh expert episodes: {disagree[fresh_s].mean():.3f}')
    print(f"  on the clone's own states:    "
          f'{occupancy @ disagree / occupancy.sum():.3f}')
    for n in [1, 10, 100]:
        pi_n = clone(mdp, *demonstrations(env, pi_star, n)).probs(
            mdp.num_states, mdp.num_actions)
        print(f'clone of {n:>3} expert episodes: reaches the goal '
              f'{visitation(mdp, pi_n)[-1][goal].sum():.3f}')
```

```
disagreement with the expert
  on the training pairs:        0.182
```

```
  on 100 fresh expert episodes: 0.182
  on the clone's own states:    0.216
clone of   1 expert episodes: reaches the goal 0.660
clone of  10 expert episodes: reaches the goal 0.660
clone of 100 expert episodes: reaches the goal 0.660
```

新たなエキスパートエピソード上でのクローンの誤り率は訓練対と同じ 18% であり、汎化ギャップは存在しない。デモンストレーションがエキスパートの分布を網羅しているからである。ここではエキスパートのデータを増やしても役に立たない。新しい状態は一切加わらず、同じ 6 個の対の繰り返しから作ったクローンはどれも同じテーブルになり、成功率も 66% のままである。一方、クローン自身の訪問状態では誤り率が 22% へと上昇する。データが存在せず確率 $3/4$ で不一致となるセルが含まれるためである。方策がテストされる状態の分布が移動しており、それを動かしたのは方策自身の行動にほかならない。これが**分布シフト**（distribution shift）である。6 ステップのタスクではこのシフトは小さく、それによって生じる損失の大きさは 14.5.3.2 節で測定する。

14.5.2.2 カバレッジの欠落

カバレッジの欠落（missing coverage）は、これとは異なる失敗である。滑りやすい氷の上ではエキスパート自身も滑るため、訪問する状態の集合はランダムになり、少数のエピソードでは正しい行動が重要となる状態を取りこぼすことがある。次のセルでは、滑りやすい湖とその最適方策を構築し、各 n についてエキスパートの n エピソードからなるデータセットを 200 個ずつ抽出し、各データセットの貪欲クローンを厳密に評価する。このクローンはデモンストレーションされたすべての状態でエキスパートの行動を取り、それ以外の状態では行が一様分布のままであるため同点時の規則により行動 0 を選ぶ。セルでは 600 個のクローンを実際に訓練する代わりに、このテーブルを直接書き下している。

```
ice = gym.make('FrozenLake-v1', is_slippery=True)
mdp_ice = d2l.TabularMDP.from_gym(ice, gamma)
V_ice = d2l.value_iteration(mdp_ice, num_iters=1000)[-1]
pi_ice = mdp_ice.backup(V_ice).argmax(axis=1)
one_hot = np.eye(mdp_ice.num_actions)
expert_ice = visitation(mdp_ice, one_hot[pi_ice])[-1][goal].sum()
ice.reset(seed=5)
print(f'share of 200 datasets whose greedy clone is below half the expert '
      f'({expert_ice:.3f}):')
for n in [1, 3, 10]:
    rates = []
    for _ in range(200):
        seen = np.isin(np.arange(mdp_ice.num_states),
                       demonstrations(ice, pi_ice, n)[0])
        # the greedy clone: the expert's action where it has data; elsewhere
        # its row is untouched and uniform, and the tie goes to action 0
        tab = np.where(seen, pi_ice, 0)
        rates.append(visitation(mdp_ice, one_hot[tab])[-1][goal].sum())
    failed = np.mean(np.array(rates) < expert_ice / 2)
    print(f'  expert episodes per dataset {n:>2}: {failed:.1%}')
```

```
share of 200 datasets whose greedy clone is below half the expert (0.730):
  expert episodes per dataset  1: 32.0%
  expert episodes per dataset  3: 1.5%
  expert episodes per dataset 10: 0.0%
```

1 エピソードの場合、約 3 つに 1 つのデータセットでクローンの成功率がエキスパートの半分を下回る。3 エピソードでは 100 個中 1〜2 個になり、10 エピソードでは 200 個中 0 個になる。カバレッジの欠落はエキスパートのデータが蓄積するにつれて消滅する。しかし分布シフトは消えない。クローンが自身のミスによって到達する状態は、エキスパートが一度も訪問しない状態だからである。

14.5.3 複合誤差

14.5.3.1 二次および線形の誤差増大

湖におけるエキスパートの経路は 6 手である。タスクがさらに長くなると、1 ステップあたりの誤りがもたらすコストはどのように増大するだろうか。教師あり学習が抑えるのは、1 回の意思決定における誤り確率である。逐次的なタスクでは、序盤の誤りが以降のすべての状態を変化させ、以降の報酬をすべて失わせかねない。デモンストレーションはエキスパートの状態分布 $d_t^{\pi^*}$ から状態をサンプリングするため、行動クローニングがクローンの誤り率を制御できるのはその分布上だけであり、他の場所では制御できない。

命題（複合誤差）（compounding error）：**T ステップからなり各ステップの報酬が $[0,1]$ に収まるエピソード型タスクを考え、方策 $\hat{\pi}$ は各ステップ t において、エキスパートと不一致になる確率が高々 ε であるとする（ただし状態はエキスパートの分布 $d_t^{\pi^*}$ から抽出されるものとする）。(i) $\hat{\pi}$ の期待収益がエキスパートの期待収益を下回る不足分は高々 $\varepsilon T(T+1)/2 \le \varepsilon T^2$ であり、$\varepsilon T \le 1$ である限り不足分がこのオーダーに達するタスクが存在する。(ii) これに対し、$\hat{\pi}$ がエキスパートと不一致になる確率が高々 ε であるのが自身の状態分布 $d_t^{\hat{\pi}}$ から状態を抽出した場合であり、かつ 1 回の誤った行動の後にエキスパートの振る舞いに従ったときのコストが高々 u であるならば、不足分は高々 $u\varepsilon T$ である** (Ross and Bagnell 2010)。

証明： (i) $\hat{\pi}$ とエキスパートを同じ初期状態・同じ環境の乱数性のもとで実行すると、$\hat{\pi}$ がエキスパートの取らない行動を最初に取るまで両者の軌跡は一致する。ステップ t におけるエキスパートの状態は $d_t^{\pi^*}$ から抽出され、最初の不一致がステップ t で生じるのは $\hat{\pi}$ がその状態で不一致になる場合に限られるため、その確率は高々 ε である。それ以降、$\hat{\pi}$ が失い得る報酬は高々ステップ $t, \dots, T$ の分、すなわち個数にして $T-t+1$ 個である。t について和をとると、不足分は高々 $\sum_{t=1}^{T} \varepsilon(T-t+1) = \varepsilon T(T+1)/2$ となる。(ii) π_k を、$\hat{\pi}$ に最初の k ステップだけ従い、その後はエキスパートに従う方策とする。すると π_0 はエキスパート、π_T は $\hat{\pi}$ となる。隣り合う 2 つのハイブリッド方策 π_{k-1} と π_k は、ステップ k における行動（$d_k^{\hat{\pi}}$ から抽出された状態で取られる行動）だけが異なる。その行動は高々確率 ε でエキスパートと不一致になり、不一致の後にエキスパートの振る舞いに従ったときのコストは高々 u である。エキスパートの収益から $\hat{\pi}$ の収益を引いた差は、隣り合うハイブリッド方策間の T 個の差の総和に等しく、したがって高々 $u\varepsilon T$ である。■

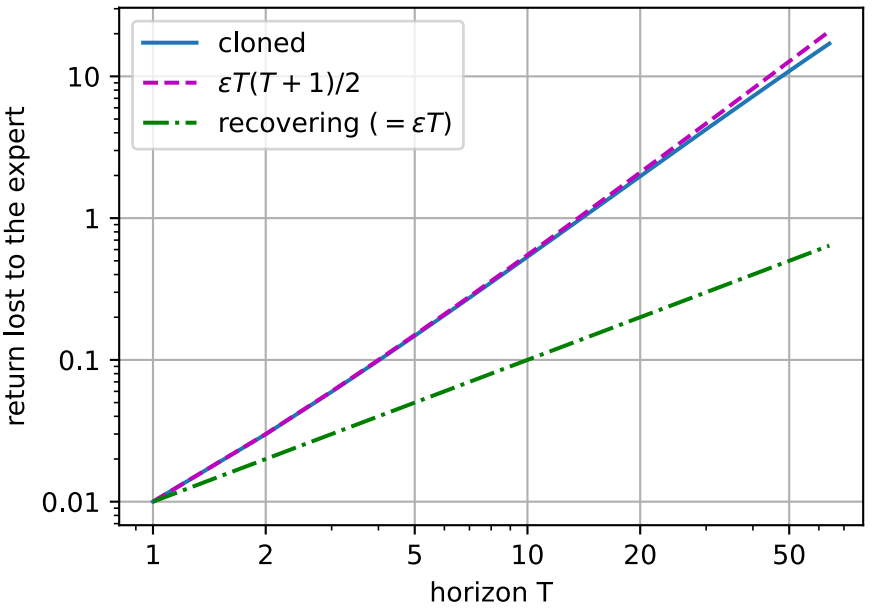
後半の (ii) は、Kakade and Langford (2002) による望遠鏡和（telescoping）の議論である。ε が T ステップにわたる 1 ステップあたり誤り率の最大値ではなく平均値であったとしても、(i) の和は依然として高々 εT^2 に収まる。(i) が最初のミスに対してそれ以降のすべてを損失として計上するのは、一度ミスをした後の方策は誤り率を何も制御できない状態に入るからである。(ii) は ε を測る場所を方策自身の訪問状態へと移すが、これが有用なのは u が小さいとき、すなわち 1 回の誤った行動からの回復をタスクが許す場合に限られる。1 回のミスが致命的であれば、u は残りのホライズンと同じ大きさになり、(ii) も再びオーダー εT^2 の評価を与える。

湖のホライズンは固定されているため、代わりに T に伴う増大を T ステップのチェーン上で観察しよう。エキスパートは毎ステップ右へ移動し、1 ステップごとに報酬 1 を獲得する。2 種類の模倣者は、毎ステップ独立に確率 ε でエキスパートと不一致になる。**クローン型**の模倣者にとってミスは永続的である。一度も示されたことのない状態へとチェーンから外れ、それ以降は何も得られないため、ステップ t の報酬を獲得する確率は $(1-\varepsilon)^t$ となる。一方、**回復型**の模倣者は

1 ステップ後にチェーンへ復帰するため、1 回のミスのコストはちょうど報酬 1 個分、すなわち $u = 1$ とした (ii) に相当し、ステップ t の報酬を確率 $1-\varepsilon$ で獲得する。次のセルでは、$\varepsilon = 0.01$ とし、T を 64 まで変化させて失われる収益を厳密に計算する。

```
eps, T_max = 0.01, 64
T = np.arange(1, T_max + 1)
# probability of earning the reward of step t: the cloned imitator only if
# it has agreed at every step up to t, the recovering one if it agrees at t
earn_bc = np.cumprod(np.full(T_max, 1 - eps))
earn_rec = np.full(T_max, 1 - eps)
gap_bc, gap_rec = T - earn_bc.cumsum(), T - earn_rec.cumsum()
d2l.plot([T] * 3, [gap_bc, eps * T * (T + 1) / 2, gap_rec],
         xlabel='horizon T', ylabel='return lost to the expert',
         legend=['cloned', '$\\varepsilon T(T+1)/2$',
                 'recovering ($=\\varepsilon T$)'],
         xscale='log', yscale='log', figsize=(5, 3.5))
print(f'lost return at T={T_max}: cloned {gap_bc[-1]:.2f} '
      f'(eps T(T+1)/2 = {eps * T_max * (T_max + 1) / 2:.2f}), '
      f'recovering {gap_rec[-1]:.2f} (eps T = {eps * T_max:.2f})')
```

```
lost return at T=64: cloned 17.03 (eps T(T+1)/2 = 20.80), recovering 0.64 (eps T = 0.64)
```



両対数軸で見ると、クローン型の模倣者は二次の基準線 $\varepsilon T(T+1)/2$ の近くに留まり（$T = 10$ で差は 3% 以内、$T = 30$ で 10% 以内）、εT が大きくなるにつれて基準線より緩やかに下へ離れていく。最初のミスは 1 度しか起こり得ないからである。最初の不一致がステップ t で生じる確率は $\varepsilon(1-\varepsilon)^{t-1}$ であり、上界が計上する ε より小さい。(i) が要請するとおり、この曲線が基準線を上回ることは決してない。回復型の模倣者が失う収益はちょうど εT である。$T = 64$ では、クローン型の模倣者が約 17 の報酬を失うのに対し、回復型は 0.64 にとどまり、損失は約 27 分の 1 に抑えられる。演習問題 1 では、両方の曲線を閉形式で導出する。

14.5.3.2 クローンの損失の分解

湖の環境に戻ろう。クローンの 34 ポイントの損失のうち、どれだけが分布シフトに起因するのだろうか。クローンが失敗する仕方は 2 通りある。1 つは、訓練された 6 つの状態のいずれかで誤った行動を取り、エキスパートの経路から直接穴に落ちる場合である。これはエキスパートの分布上での誤り ε であり、より良い当てはめによってしか取り除けない。もう 1 つは、経路を外れてデータのないセルへ入り、そこから失敗する場合である。こちらがシフトによる損失である。次のセルでは、経路外のセルを吸収状態にすることで visitation によりクローンが経路を外れる頻度を測り、経路上に留まるエピソードと経路を外れるエピソードのそれぞれがゴールに到達する頻度を比較する。最後の行では、経路上と経路外のすべてのセルでクローンと同じ 18% の誤り率を持つ学習者を評価する。

```
path = np.isin(np.arange(mdp.num_states), demo_s)    # the expert's cells
off_path = ~path & ~(goal | holes)
d_T = visitation(mdp, pi_bc, absorbing=off_path)[-1]
left = d_T[off_path].sum()
stay_goal = d_T[goal].sum()
clone_goal = visitation(mdp, pi_bc)[-1][goal].sum()
print(f'the clone leaves the expert path in {left:.3f} of its episodes, '
      f'entering cells {np.flatnonzero(d_T * off_path)}')
print(f'it reaches the goal in {stay_goal / (1 - left):.3f} of the episodes '
      f'that stay on the path, {(clone_goal - stay_goal) / left:.3f} of '
      f'those that leave it')
# the clone's demonstrated rows, with their error, at every cell
p_agree = pi_bc[demo_s[0], demo_a[0]]
labeled = np.where(expert == 1, p_agree, (1 - p_agree) / 3)
print(f'a learner with the same error at every cell reaches the goal '
      f'{visitation(mdp, labeled)[-1][goal].sum():.3f}')
```

```
the clone leaves the expert path in 0.172 of its episodes, entering cells [ 1 10]
it reaches the goal in 0.715 of the episodes that stay on the path, 0.394 of those that leave it
a learner with the same error at every cell reaches the goal 0.735
```

クローンはエピソードの 17% で経路を外れるが、その出口は常に状態 0 での誤った行動によるセル 1 への進入か、状態 9 または 14 からのセル 10 への進入である（Figure 14.14）。経路上に留まるエピソードは 72% の確率でゴールに到達し、経路を外れるエピソードは 39% の確率でゴールに到達する。損失もこれに応じて分解される。34 ポイントのうち、24 ポイントは経路そのものからの転落（0.828×0.285）であり、10 ポイントは経路を外れた後の失敗（0.172×0.606）である。すべてのセルでクローンと同じ誤り率を持つ学習者は 73.5% の確率でゴールに到達するため、経路外のラベルによって取り戻せるのは 34 ポイントのうち約 7.5 ポイントであり、10 ポイントには届かない。ラベル付きのセルでもなお 18% の確率で誤るからである。この湖では、分布シフトはクローンの損失の約 5 分の 1 を占める。残りは経路上および経路外における 18% の誤りそのものであり、データをどのように変えても取り除けない。経路がさらに長くなると、クローンははるかに頻繁に経路を外れるようになる。演習問題 4 では、8×8 の湖でそれが損失の 2 つの内訳にどう影響するかを調べる。

14.5.4 対話的なデータ収集

14.5.4.1 データセット集約

デモンストレーションを増やしてもシフトは解消できない。クローンが失敗するセルをエキスパートは一度も訪問しないからである。学習者が自らそこを訪問し、エキスパートがそこにラベル

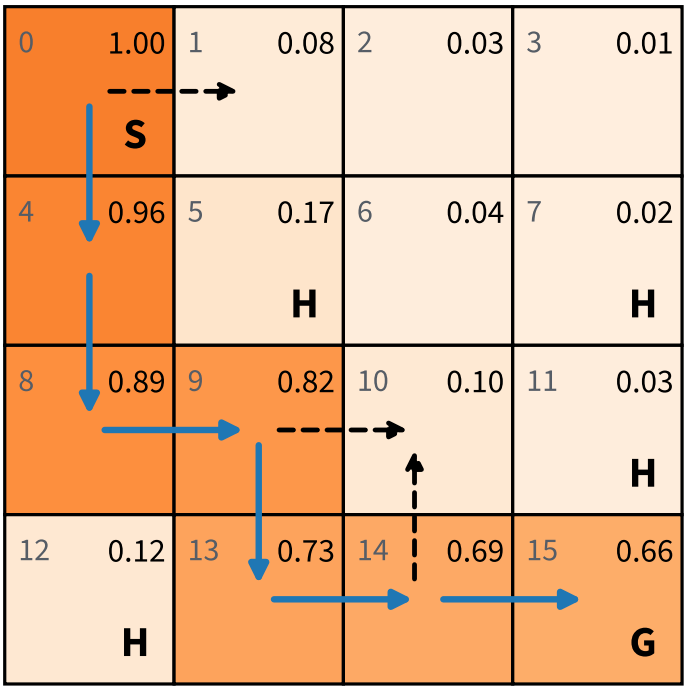


Figure 14.14: 滑らない湖における本節のクローンの移動先。青い矢印はエキスパートの 6 手の経路を示す。各セルの数値は、行動をサンプリングしエキスパートの状態で確率 $\varepsilon = 0.18$ でエキスパートと不一致になる運用時のクローンが、100 ステップの制限時間内にそのセルへ入る確率を表し、色が濃いセルほど頻繁に訪問される。破線の矢印は、クローンが経路を外れてデータがなく一様に行動するセルへと出る出口（0 から 1、および 9 または 14 から 10）である。クローンは確率 0.66 でゴールに到達し、残りの大半は経路脇の穴である 5 と 12 に吸い込まれる。

を付けなければならない。データセット集約（dataset aggregation）の略である **DAgger** は、まさにこれを行う手法である (Ross et al. 2011)。すなわち、現在の学習者を実行し、学習者が訪問した各状態でエキスパートならどの行動を取ったかを尋ね、それらの対をデータセットに追加して再び当てはめる（Figure 14.15）。モデル、損失、オプティマイザ、停止規則はすべてそのままであり、データだけが変わる。

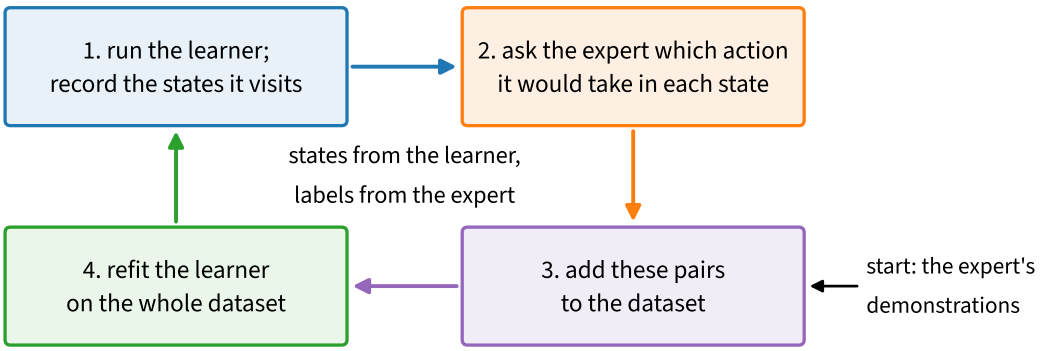


Figure 14.15: DAgger の 1 ラウンド。学習者が状態を選び、エキスパートがラベルを与える。すなわち、学習者を実行し、学習者が訪問した各状態でエキスパートが取るべき行動を指定し、それらの対をデータセットに加えて学習者を再び当てはめる。最初のデータセットはエキスパート自身のデモンストレーションである。

`relabel` は運用時と同じく行動をサンプリングする `act` で学習者をロールアウトし、訪問した状態にエキスパートの行動でラベルを付ける。

```
def relabel(env, expert, learner, num_episodes, rng):
    """Roll the learner, sampling its actions; label its states with the
    expert's actions."""
    visited = []
    for _ in range(num_episodes):
        s, done = env.reset()[0], False
        while not done:
            visited.append(s)
            s, _, terminated, truncated, _ = env.step(learner.act(s, rng))
            done = terminated or truncated
    visited = np.array(visited)
    return visited, expert[visited]
```

`dagger` は 3 エピソード分のデモンストレーションから出発し、各ラウンドで学習者の 10 エピソードを収集する処理を 4 ラウンド実行し、毎ラウンド後に `clone` で再び当てはめて各学習者を厳密に評価する。オンライン付録「ユーティリティ関数とクラス」の `d2l.run_seeds` でこれを 10 個のシードにわたって実行し、結果を積み重ねる。

```
def dagger(seed, env, mdp, expert, demo_s, demo_a, num_rounds=4,
           num_episodes=10):
    """Yield (labels, distinct states, success) for behavior cloning and
    each DAgger round after it."""
    rng = np.random.default_rng(seed)
```

```
        goal = env.unwrapped.desc.ravel() == b'G'
        S, A = demo_s, demo_a                       # the aggregated dataset
        for k in range(num_rounds + 1):
            learner = clone(mdp, S, A)
            pi = learner.probs(mdp.num_states, mdp.num_actions)
            yield S.size, np.unique(S).size, visitation(mdp, pi)[-1][goal].sum()
            if k < num_rounds:            # the learner visits, the expert labels
                new_s, new_a = relabel(env, expert, learner, num_episodes, rng)
                S, A = np.concatenate([S, new_s]), np.concatenate([A, new_a])

    runs = d2l.run_seeds(dagger, num_seeds=10, env=env, mdp=mdp, expert=pi_star,
                         demo_s=demo_s, demo_a=demo_a)   # (seed, round, quantity)
    for k, (n, m, rate) in enumerate(np.median(runs, axis=0)):
        print(f'round {k}: {n:>3.0f} labels at {m:>2.0f} states, success '
              f'{rate:.3f} (seeds: {runs[:, k, 2].min():.3f} to '
              f'{runs[:, k, 2].max():.3f}; medians over 10 seeds)')
```

```
round 0:  18 labels at  6 states, success 0.660 (seeds: 0.660 to 0.660; medians over 10 seeds)
round 1:  84 labels at  8 states, success 0.710 (seeds: 0.649 to 0.735; medians over 10 seeds)
round 2: 152 labels at  9 states, success 0.717 (seeds: 0.652 to 0.735; medians over 10 seeds)
round 3: 218 labels at 11 states, success 0.735 (seeds: 0.710 to 0.735; medians over 10 seeds)
round 4: 283 labels at 11 states, success 0.735 (seeds: 0.717 to 0.735; medians over 10 seeds)
```

ラウンド 0 は成功率 0.66 の行動クローニングである。各ラウンドでは約 65 個のラベルが追加され、その大半は経路上の 6 つの状態に対するものだが、ときおり経路外のセルも加わる。4 ラウンド（数百ラベル）のうちに、10 シードにわたる中央値はシードの選び方に応じて 0.70〜0.735 に達し、すべてのセルでクローンと同じ誤り率を持つ学習者の 0.735 に一致するかその近くまで到達する。また、どのシードでも第 4 ラウンドの学習者は行動クローニングを上回る。エキスパートのラベルだけをどれだけ増やしても、行動クローニングは 0.66 のままである。DAgger は経路外の状態に由来していた損失を取り戻す。エキスパートとの間に残る差は当てはめ自体の 18% の誤りであり、DAgger はこれには手を加えない。なお、学習者が 0.735 に到達するのは近似ではなく厳密な一致である。Adam は各パラメータの更新ステップをそのパラメータ自身の勾配の大きさで割るが、ラベル付きの行の勾配は「その行のラベル数 × ラベル付きの全行に共通する因子」となるため、何回ラベル付けされたかにかかわらず、ラベル付きのすべての行が同じ 0.818 で終了するからである。したがって学習者はどのセルにラベルがあるかだけで一意に決まり、到達し得るすべてのセルにラベルが付いた時点で厳密に 0.735 の学習者となる。

ラウンドごとに単調に改善するとは限らない。シードによってはあるラウンドで成功率がわずかに下がり、最悪で 0.649 になることがあり、その原因は演習問題 5 で突き止める。Ross et al. (2011) の理論保証は、学習者の系列のうち最良のもの、あるいはその系列からランダムに 1 つ選んだものに対する保証であり、最後の学習者に対するものではない。この保証は、多数ラウンドのデータ収集、蓄積されたデータセット上で訓練されるノーリグレット学習者（no-regret learner）、そして元の解析ではデータ収集時におけるエキスパートと学習者の行動の混合を仮定している。さらに、学習者がどのような状態に到達してもラベルを付けられるよう、随時呼び出せるエキスパートも必要とする。過去のすべてのラウンドのデータを保持し続けることで、当てはめた方策が直近の軌跡のみに依存してしまうのを防いでいる。

14.5.4.2 行動クローニングとしての教師ありファインチューニング

14.1 節のトークンマルコフ決定過程（MDP）のもとでは、言語モデルの教師ありファインチューニング（supervised fine-tuning）は行動クローニングの一例となる。状態はコンテキスト（プロンプトとそこまでに書かれたトークン列）、行動は次のトークン、遷移は確率 1 でそのトークン

を末尾に追加する操作であり、厳選された応答がエキスパートのデモンストレーションにあたる。ファインチューニングの交差エントロピーは、各応答のトークンにわたって和をとった行動クローニングの負の対数尤度である。ホライズン T は応答の長さであり、u は 1 トークン誤った後にモデルがうまく続けた場合のコストを表す。すなわち、風変わりだが無害な単語であれば小さく、後続のすべてのステップが依存する計算の序盤で数字を 1 つ誤れば大きくなる。運用時にはモデルは自身のトークンをサンプリングしてそれを条件とするため、応答が長くなるにつれてコンテキストは厳選された応答の分布からドリフトしていく。

言語モデルに対する DAgger には、モデル自身が生成したプレフィックスにラベルを付けるエキスパート、すなわち生徒モデルが生成した各コンテキストに対して次にどのトークンが来るべきかを指定する存在が必要である。人間が何百万ものプレフィックスに対してこれを行うことはできないが、より大規模なモデルなら可能である。オンポリシー蒸留（on-policy distillation）は、教師モデルの次トークン分布をラベルとして、生徒自身のサンプル上で生徒を訓練する手法であり (Agarwal et al. 2024)、教師を随時呼び出せるエキスパートとした DAgger にほかならない。また、行動クローニングは固定されたデータセットのみが利用可能なオフライン強化学習（offline reinforcement learning）における標準的な最初のベースラインでもあり、15.7 節ではオフライン手法を行動クローニングと比較する。

行動クローニングは単なる分類器だけでエキスパートの記録された選択を方策へと変え、DAgger はデータだけを変えることで、学習者自身の訪問状態に起因する損失を取り除く。ただし、どちらもエキスパートを必要とする。デモンストレーションを行うエキスパートであり、DAgger の場合はさらに学習者が何を行ってもその場でラベルを付けられる随時待機型のエキスパートである。エキスパートが利用できない場合にも、2 つの道が残されている。環境の規則が既知で微分可能であれば、そこから良い行動を計算でき、14.6 節では連続的な状態と行動に対してこれを行う。環境の規則すら未知であっても、重み $w_i = 1$ を「その行動がどれほどうまくいったか」を表す数値に置き換えれば、クローニングの更新則はそのまま適用できる。14.7 節では、その重みを報酬から導出する。

14.5.5 まとめ

行動クローニングはソフトマックス回帰と同様に交差エントロピーによってエキスパートの状態行動対に方策 $\pi_\theta(a \mid s)$ を当てはめる手法であり、遷移モデルも報酬も必要としない。各当てはめステップは、単位重みを用いた対数確率に対する勾配ステップである。この当てはめはエキスパートの状態分布上での方策の誤りを制御するが、運用時の方策は自身の行動が生み出す状態を訪問する。エキスパートの状態における 1 ステップあたりの誤り率 ε は、失われる収益が最大 $\varepsilon T(T+1)/2$（T ステップあたり）に達し得る一方、方策自身の状態における同じ誤り率であれば、損失は高々 $u\varepsilon T$（1 回のミスのコストが高々 u のとき）に収まる。滑らない湖の 6 手の経路では、エキスパートのデータを増やしてもクローンは変化せず、エキスパートが一度も訪問しない状態へのシフトが損失の約 5 分の 1 を占める。DAgger は学習者自身の訪問状態にエキスパートの行動でラベルを付け、データだけを変えることでその損失を回復する。

14.5.6 演習問題

1. **閉形式で表すチェーン**。14.5.3.1 節のチェーン実験では、模倣者は確率 ε で各ステップ $t = 1, \dots, T$ において独立にエキスパートと不一致になる。クローン型の模倣者がステップ t の報酬を獲得するのは、t までのすべてのステップで一致した場合に限られる。回復型の模倣者は、ステップ t で一致すれば常にその報酬を獲得する。
    1. クローン型の模倣者がエキスパートに対して失う期待収益 $G(T)$ を閉形式で導出し、回復型の模倣者が εT を失うことを示せ。両者を $\varepsilon = 0.01$ および $T = 64$ に対して出力された値と比較せよ。

   2. $G(T)$ を ε について 2 次まで展開せよ。1 次の項が命題の上界 $\varepsilon T(T+1)/2$ になることを示し、2 次の項を求めよ。その符号から、クローンの曲線が基準線に対してどちら側に位置するかについて何が言えるか。$\varepsilon = 0.01$ のとき、1 次の項が $G(T)$ の 10% 以内に収まるのはどのホライズンまでか。
   3. $G(T)$ は $\varepsilon T \gg 1$ になるとどのように増大するか。また、したがって二次の領域が成り立つには εT にどのような条件が必要か。プロットの右端にあたる $\varepsilon = 0.01$ および $T = 64$ における 2 つの損失の比を計算し、2 つの基準線の比 $(T+1)/2$ と比較せよ。

2. **行動クローニングとしてのファインチューニング**。14.5.4.2 節では、コンテキストを状態、次のトークンを行動、厳選された応答をデモンストレーションとみなすことで、言語モデルの教師ありファインチューニングを行動クローニングと同一視した。
   1. 14.5.3.1 節の命題は、すべてのステップで同じ誤り率 ε を仮定している。ファインチューニングされたモデルは、厳選された応答に沿っては小さな一定の率でしか誤らないにもかかわらず、自身が生成した応答に沿っては位置が進むにつれて増大する率で誤ることがある。その理由を説明し、命題のどちらの主張がこのモデルを記述するかを述べよ。
   2. 件数の問題を別にすれば、言語モデルに対する DAgger のラベル供給源として人間が不向きなのはなぜか。オンポリシー蒸留は、生徒の各プレフィックスに対して 1 つのトークンではなく教師の次トークン分布全体をラベルとして付与する。得られる損失を書き下し、教師が 1 つのトークンに全確率を置くときに行動クローニングの損失に帰着することを示せ。

3. [code] **カバレッジに必要なデモンストレーション**。●14.5.2 節のカバレッジのセルでは、滑りやすい湖 `mdp_ice` とその最適方策 `pi_ice` を構築している。決定論的方策 `pi` がゴールに到達する確率は、`visitation(mdp_ice, one_hot[pi])` の後にゴールにある確率質量である。
   1. 各非終端状態 s について、基本的には `pi_ice` に従うが s でのみ行動 0 を取る方策の成功確率を計算せよ。この変更によって成功確率が `pi_ice` の半分未満に下がるとき、s を重要状態と呼ぶ。どの状態が重要状態か。
   2. q_s を、エキスパートの 1 エピソードが s を訪問する確率とする。q_s を、引数 `absorbing`（`visitation` の引数）を用いて厳密に計算せよ。すなわち、s を吸収状態にし、100 ステップ後に s にある確率質量を読み取れ。$q_s = 0$ となる非終端状態はどれか。また、それは任意の量のエキスパートデータに対して何を意味するか。3 エピソードからなるデータセットが状態 13 を一度も訪問しない確率はいくらか。
   3. 重要状態への訪問を互いに独立とみなして、それらの q_s からエキスパートの $n \in \{1, 3, 10\}$ エピソードがいずれかの重要状態をデモンストレーションしないまま残す確率を予測し、その予測をセルが出力する割合と比較せよ。この予測は高すぎるか、それとも低すぎるか。また、それは異なる重要状態への訪問について何を物語っているか。

4. [code] **より長い経路での DAgger**。●4×4 の滑らない湖では、分布シフトがクローンの損失に占める割合は約 5 分の 1 である。最適経路に 14 手を要する 8×8 の滑らない湖 `gym.make('FrozenLake-v1', map_name='8x8', is_slippery=False)` で同じ実験を繰り返せ。すなわち、`env`、`mdp`、`pi_star`、`demo_s`、`demo_a` を再構築し、本節のセルを再実行せよ。
   1. サンプリングクローンの成功確率、エキスパートの経路を外れる確率、およびすべてのセルでクローンと同じ誤り率を持つ学習者の成功確率を計算せよ。クローンの損失のうち、経路を外れた後に生じる割合はどれだけか。また、経路外のラベルによって回復できる割合はどれだけか。前者が経路の長さとともに増大する一方で、後者がそうならないのはなぜか。
   2. 10 個のシードで DAgger を実行し、エキスパートへのクエリ数に対する成功率を、少なくとも同数のエキスパートラベルを用いた行動クローニングと並べてプロットせよ。
   3. 表形式の方策が必要とするラベルは、状態ごとに高々 1 つである。各ラウンド後に

DAgger がラベルを付けた異なる状態の個数を数え、ラベルのない状態でのみエキスパートに尋ねた場合に必要となるクエリ数を見積もれ。

5. **DAgger の理論保証**。●Ross et al. (2011) の解析では、データ収集方策 π_i（反復 i における方策）は各ステップで確率 β_i でエキスパートの行動を取り、それ以外では現在の学習者 $\hat{\pi}_i$ の行動を取る。次の学習者 $\hat{\pi}_{i+1}$ は、`clone` による再当てはめと同様に、反復 $1, \dots, i$ の集約データ上の損失を最小化する。$d_\pi = \frac{1}{T}\sum_{t=1}^{T} d_t^\pi$ を π の平均状態分布とし、$c(s,\pi) \in [0, c_{\max}]$ を π の状態 s における分類損失（$\mathbf{1}(\pi(s) \neq \pi^*(s))$ など）とし、$c_i(\pi) = \mathbb{E}_{s\sim d_{\pi_i}}[c(s,\pi)]$ とおく。
   1. $\|d_{\pi_i} - d_{\hat{\pi}_i}\|_1 \le 2\min(1, T\beta_i)$ を示せ。
   2. $\varepsilon_N = \min_\pi \frac{1}{N}\sum_{i=1}^{N} c_i(\pi)$ を事後的に最良な方策の損失とし、学習者の平均リグレットが高々 δ_N であると仮定する。

$$\frac{1}{N}\sum_{i=1}^{N} c_i(\hat{\pi}_i) - \varepsilon_N \le \delta_N. \tag{14.46}$$

   このとき、系列中のある学習者 $\hat{\pi}_j$ が次式を満たすことを示せ。

$$\mathbb{E}_{s\sim d_{\hat{\pi}_j}}[c(s,\hat{\pi}_j)] \le \varepsilon_N + \delta_N + \frac{2c_{\max}}{N}\sum_{i=1}^{N}\min(1, T\beta_i). \tag{14.47}$$

   同論文の定理 4.1 の証明では、最後の和を簡約する前にこの上界を導出している。
   3. 14.5.4.1 節のループでは $\beta_1 = 1$、かつ $\beta_i = 0$（$i \ge 2$）を用いている。この上界をこのスケジュールと 0-1 損失に特殊化し、14.5.3.1 節の命題の (ii) と組み合わせよ。N が T に対してどのように増大すれば、ε_N や δ_N を含まない項を有界に保てるか。
   4. 出力された実行結果の一部のシードでは、第 1 ラウンドの学習者が行動クローニングより悪化している。状態 1 のみにラベルを付けると滑らない湖のクローンの厳密な成功率が下がることを示し、状態 1 におけるエキスパートの行動と、ラベル付与前後における状態 1 および 2 の学習者の行からその低下を説明せよ。系列の中から最良の学習者を特定するには何が必要になるか。

6. **デモンストレーションの背後にある報酬**。●本文ではエキスパートの行動から学習した。これに対し、**逆強化学習**はデモンストレーションが最適となるような報酬を推定し、その報酬のもとでプランニングを行う。
   1. 14.2.5 節の望遠鏡和の議論により、ポテンシャルに基づく報酬シェイピング $\tilde{r}(s,a,s') = r(s,a) + \gamma\Phi(s') - \Phi(s)$（$r$ に対するもので、終端状態で $\Phi = 0$ を満たすもの）はいずれもエキスパートの最適性を保つ。ゼロ報酬のもとでもエキスパートが最適になることを示せ。これら 2 つの事実は、デモンストレーションから r を復元することについて何を意味するか。
   2. 最大エントロピー逆強化学習 (Ziebart et al. 2008) は、初期状態が固定された決定論的エピソード型タスクの軌跡を実行可能な軌跡全体の上で $P_\theta(\tau) \propto \exp\left(\sum_t r_\theta(s_t,a_t)\right)$ とモデル化し、デモンストレーションの尤度を最大化することで θ を当てはめる。ゴールに到達するエキスパートに対して、この基準がゼロ報酬を棄却するのはなぜか。実行可能なすべての軌跡が終端状態で終わると仮定する。このとき、この基準が r とその $\gamma = 1$ の報酬シェイピングを区別できないのはなぜか。また、時間制限によって打ち切られた軌跡に対しては、この議論のどの段階が成り立たなくなるか。

14.6 最適制御、LQR、モデル予測制御

動的計画法（14.2 節）は、16 個の状態価値からなる表を反復走査し、すべてのマスでベルマンの最適性方程式が成り立つまで更新することで滑りやすい湖の問題を解いた。それ以降に扱った Q 学習から行動クローニングに至るすべての手法も、状態ごとに表の 1 行を保持してきた。では、14.1 節の車両の例のように状態が位置・角度・速度からなる連続ベクトル $\mathbf{x}_t \in \mathbb{R}^d$ であり、行動が連続的な制御入力 $\mathbf{u}_t \in \mathbb{R}^m$ として既知のダイナミクス $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ を通じて作用する場合はどうなるだろうか。グリッドはスケールしない。d 個の座標それぞれを 100 個のビンに分割すると 100^d 個のセルが生じるが、これこそ Bellman の**次元の呪い**（curse of dimensionality）である (Bellman 1957b)。

制御理論（control theory）は、深層強化学習が登場するはるか以前に、Kalman (1960) による代数的観察を出発点として連続状態・連続行動空間におけるこの問題を解決していた。遷移ダイナミクスが状態と制御入力に関して線形であり、1 ステップのコストがその両方に関して二次であるとき、ベルマンの後ろ向き動的計画法はすべてのステップで価値関数の二次形式を保つ。100^d 個の要素をもつ表の代わりに、厳密な価値関数 $V_t(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_t \mathbf{x}$ は、$d \times d$ の対称行列 $\mathbf{P}_t$ によって $d(d+1)/2$ 個の独立な数値だけでパラメータ化される。ベルマンバックアップは**リッカチ漸化式**（Riccati recursion）に帰着し、最適方策は線形状態フィードバック則 $\mathbf{u}_t^* = -\mathbf{K}_t \mathbf{x}_t$ となる。これが**線形二次レギュレータ**（linear-quadratic regulator、LQR）である (Kalman 1960; Anderson and Moore 2007)。

現実の物理システムの多くは非線形であり、アクチュエータには飽和がある。2 つのアイデアが LQR を非線形連続制御へと拡張し、現代の軌跡最適化およびロボティクスの骨格をなしている (Tedrake 2023; Bertsekas 2025)。第 1 に、**反復 LQR**（iterative LQR、iLQR）(Li and Todorov 2004) は、候補となる軌跡の周りで非線形ダイナミクスの線形化とコストの二次近似を繰り返し、後ろ向きパスで時変 LQR 問題を解いて前向きパスで軌跡を改善する。第 2 に、**モデル予測制御**（model predictive control、MPC）(Rawlings et al. 2017) は、有限ホライズンの軌跡最適化を閉ループ方策へと変換する。すなわち、各ステップ t において現在の観測状態 $\mathbf{x}_t$ から移動する先読みホライズン（receding lookahead horizon）にわたって制御問題を解き直し、最初の制御入力 $\mathbf{u}_{t|t}^*$ のみを適用して、ステップ $t+1$ で再計画する。本節では LQR、iLQR、および移動ホライズン MPC を展開し、局所線形フィードバックで十分な領域と非線形なエネルギー注入（energy pumping）が必要となる領域を明らかにしたうえで、MPC が 15.6 節における学習済みニューラルダイナミクスを用いた計画の土台をいかに築くかを示す。

14.6.1 線形二次制御

14.6.1.1 状態空間ダイナミクスと二次コスト

制御理論の記法では、連続状態を $\mathbf{x}_t \in \mathbb{R}^d$、行動（または制御入力）を $\mathbf{u}_t \in \mathbb{R}^m$ と書く。強化学習が報酬 $r(\mathbf{x}_t, \mathbf{u}_t)$ を最大化するのに対し、最適制御（optimal control）はステージコスト $c(\mathbf{x}_t, \mathbf{u}_t) = -r(\mathbf{x}_t, \mathbf{u}_t)$ を最小化する。離散時間線形システムは次のように時間発展する。

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t, \tag{14.48}$$

ここで、$\mathbf{A} \in \mathbb{R}^{d \times d}$ は状態遷移行列、$\mathbf{B} \in \mathbb{R}^{d \times m}$ は制御入力行列である（38.1 節の線形常微分方程式の離散時間版に相当する）。目標状態は原点 $\mathbf{x} = \mathbf{0}$ である。H ステップの有限ホライズンにわたり、次の総二次コストを最小化する。

$$J(\mathbf{x}_0, \mathbf{u}_{0:H-1}) = \sum_{t=0}^{H-1} \left(\mathbf{x}_t^\top \mathbf{Q} \mathbf{x}_t + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t \right) + \mathbf{x}_H^\top \mathbf{Q}_H \mathbf{x}_H, \tag{14.49}$$

ここで、$\mathbf{Q} \succeq \mathbf{0}$ と $\mathbf{Q}_H \succeq \mathbf{0}$ は $d \times d$ の対称半正定値な状態コスト行列であり、$\mathbf{R} \succ \mathbf{0}$ は $m \times m$ の対称正定値な制御コスト行列である。$\mathbf{R} \succ \mathbf{0}$ を課すことで、大きな制御入力には常に正のペナルティが伴うため、$\mathbf{u}_t$ に関する 1 ステップ最小化は一意な有限解をもつ。

14.6.1.2 厳密な連続動的計画法とリッカチ漸化式

$V_t(\mathbf{x})$ を状態 $\mathbf{x}$、ステップ $t \in \{0, \dots, H\}$ からの最適残りコスト（cost-to-go）とする。終端ステップ $t = H$ では残りの制御入力はなく、$V_H(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_H \mathbf{x}$（ただし $\mathbf{P}_H = \mathbf{Q}_H$）となる。$t+1$ から t へと後ろ向きに進むと、有限ホライズン版のベルマンの最適性方程式（14.2 節）は次の式を与える。

$$V_t(\mathbf{x}) = \min_{\mathbf{u} \in \mathbb{R}^m} \Big\{ \mathbf{x}^\top \mathbf{Q} \mathbf{x} + \mathbf{u}^\top \mathbf{R} \mathbf{u} + V_{t+1}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) \Big\}. \tag{14.50}$$

後ろ向き帰納法により、ある対称半正定値行列について $V_{t+1}(\mathbf{x}') = \mathbf{x}'^\top \mathbf{P}_{t+1} \mathbf{x}'$（$\mathbf{P}_{t+1} \succeq \mathbf{0}$）が成り立つと仮定しよう。$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ を代入すると、括弧内の行動価値関数 $Q_t(\mathbf{x}, \mathbf{u})$ は $\mathbf{u}$ に関する狭義凸二次関数に展開される。

$$\begin{aligned} Q_t(\mathbf{x}, \mathbf{u}) &= \mathbf{x}^\top (\mathbf{Q} + \mathbf{A}^\top \mathbf{P}_{t+1} \mathbf{A}) \mathbf{x} + 2 \mathbf{u}^\top \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{A} \mathbf{x} \\ &\quad + \mathbf{u}^\top (\mathbf{R} + \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{B}) \mathbf{u}. \end{aligned}$$

$\mathbf{u}$ に関する勾配をゼロとおくと、次のようになる。

$$\nabla_{\mathbf{u}} Q_t(\mathbf{x}, \mathbf{u}) = 2(\mathbf{R} + \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{B}) \mathbf{u} + 2 \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{A} \mathbf{x} = \mathbf{0}, \tag{14.51}$$

これより、一意な最適制御が閉形式で得られる。

$$\mathbf{u}_t^\ast = -\mathbf{K}_t \mathbf{x}_t, \qquad \mathbf{K}_t = (\mathbf{R} + \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{A}. \tag{14.52}$$

$\mathbf{u} = -\mathbf{K}_t \mathbf{x}$ を $Q_t(\mathbf{x}, \mathbf{u})$ に戻して代入すると、$V_t(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_t \mathbf{x}$ が純粋な二次形式のままであることが確認でき、$\mathbf{P}_t$ は後ろ向きの**離散時間リッカチ漸化式**（discrete-time Riccati recursion）に従う。

$$\begin{aligned} \mathbf{P}_t &= \mathbf{Q} + \mathbf{A}^\top \mathbf{P}_{t+1} \mathbf{A} \\ &\quad - \mathbf{A}^\top \mathbf{P}_{t+1} \mathbf{B} (\mathbf{R} + \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{B})^{-1} \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{A}. \end{aligned} \tag{14.53}$$

この漸化式は $\mathbf{P}_H = \mathbf{Q}_H$ から後ろ向きに 1 回計算するだけでよく、その際に状態を知る必要はない。したがってシステムの実行コストは低い。各ステップで制御器は $\mathbf{x}_t$ を計測し、保存しておいたゲインを掛けるだけでよい（Figure 14.16）。

Equation 14.52 と Equation 14.53 には、特筆すべき 3 つの性質がある。

1. **次元の呪いがないこと：**各ベルマンバックアップは、無限の行動空間 $\mathbb{R}^m$ とすべての状態 $\mathbb{R}^d$ にわたる最小化を、$O(d^3 + m^3)$ 回の $\mathbf{P}_{t+1}$ に対する行列演算に置き換える。
2. **線形フィードバックゲイン：**最適方策 $\pi_t^\ast(\mathbf{x}) = -\mathbf{K}_t \mathbf{x}$ は状態に関して線形であり、ゲイン行列 $\mathbf{K}_t \in \mathbb{R}^{m \times d}$ は $(\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R})$ のみに依存し、初期状態 $\mathbf{x}_0$ には依存しない。

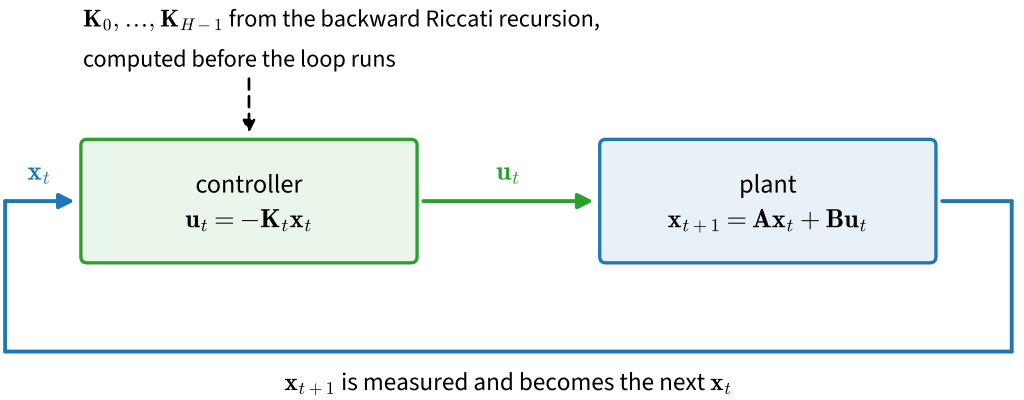


Figure 14.16: 閉ループでの LQR。制御器は計測された状態 $\mathbf{x}_t$ にゲイン $-\mathbf{K}_t$ を掛け、システム（制御用語でいう**プラント**）は次の状態で応答し、その状態が再び計測される。すべてのゲインはループ開始前に後ろ向きリッカチ漸化式によって計算される。

3. **確実性等価（certainty equivalence）**：平均ゼロの独立ノイズ $\mathbf{w}_t$（共分散 Σ）が遷移に加わっても（$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t$）、交差項 $\mathbb{E}[\mathbf{w}_t^\top \mathbf{P}_{t+1}(\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t)]$ は消え、価値関数に定数 $\operatorname{tr}(\mathbf{P}_{t+1}\Sigma)$ が加わるだけで最適フィードバック行列 $\mathbf{K}_t$ は変化しない（演習問題 2）。

ホライズン $H \to \infty$ の極限では（標準的な可制御性（controllability）および可観測性（observability）の条件のもとで）、$\mathbf{P}_t$ は**離散代数リッカチ方程式**（discrete algebraic Riccati equation、DARE）の定常解 $\mathbf{P}_\infty$ に収束し、時不変制御器 $\mathbf{u}_t = -\mathbf{K}_\infty \mathbf{x}_t$ は閉ループダイナミクス $\mathbf{x}_{t+1} = (\mathbf{A} - \mathbf{B}\mathbf{K}_\infty)\mathbf{x}_t$ を安定化して、$\mathbf{A} - \mathbf{B}\mathbf{K}_\infty$ のすべての固有値を単位円の厳密な内側、すなわち $\rho(\mathbf{A} - \mathbf{B}\mathbf{K}_\infty) < 1$ に配置する。

14.6.1.3 二重積分器

`riccati_lqr` を 14.1 節の状態拡張の例で扱った車両でテストしよう。これは直線上を動く単位質量の質点であり、状態は $\mathbf{x}_t = (p_t, v_t)^\top$（位置と速度）、制御入力は u_t（加速度）、タイムステップは $\Delta t = 0.1$ である。Newton の運動法則より、$p_{t+1} = p_t + v_t \Delta t + \tfrac{1}{2} u_t \Delta t^2$ および $v_{t+1} = v_t + u_t \Delta t$ が成り立つ。位置と速度のずれに $\mathbf{Q} = \operatorname{diag}(1.0, 0.1)$ でペナルティを課し、制御入力の重み $r_u \in \{0.01, 0.10, 1.00\}$ を $\mathbf{R} = [r_u]$ において変化させる。

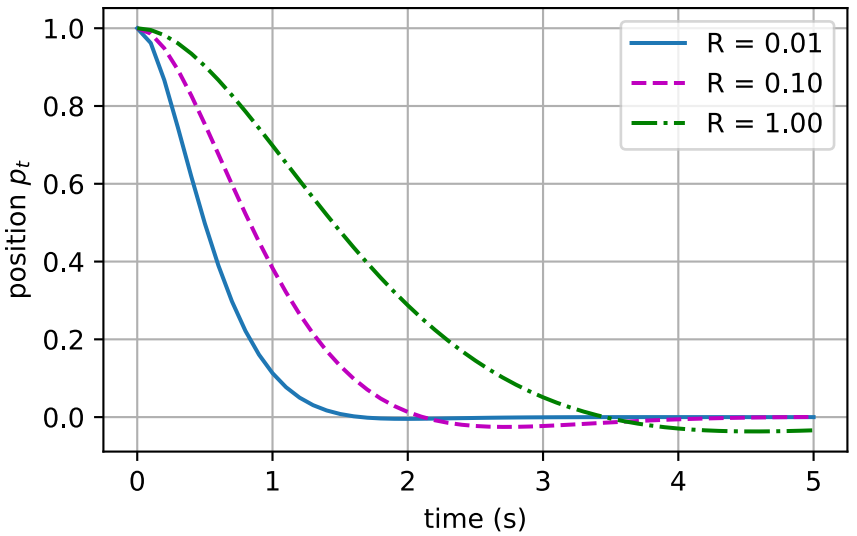
```
def riccati_lqr(A, B, Q, R, H=100):
    """Backward Riccati recursion for horizon H; returns (Ps, Ks) from t=0..H."""
    P = Q.copy()
    Ps, Ks = [P], []
    for _ in range(H):
        BtP = B.T @ P
        K = np.linalg.solve(R + BtP @ B, BtP @ A)
        P = Q + A.T @ P @ (A - B @ K)
        Ks.append(K)
        Ps.append(P)
    return Ps[::-1], Ks[::-1]

dt = 0.1
```

```
A_di = np.array([[1.0, dt], [0.0, 1.0]])
B_di = np.array([[0.5 * dt**2], [dt]])
Q_di = np.diag([1.0, 0.1])

for ru in (0.01, 0.10, 1.00):
    Ps, Ks = riccati_lqr(A_di, B_di, Q_di, np.array([[ru]]), H=100)
    K_inf, P_inf = Ks[0][0], Ps[0]
    rho = np.abs(np.linalg.eigvals(A_di - B_di @ Ks[0])).max()
    print(f'R={ru:4.2f}  K_inf=[{K_inf[0]:5.3f}, {K_inf[1]:5.3f}]  '
          f'P_00={P_inf[0, 0]:5.2f}  rho(A - B K)={rho:.3f}')
```

```
R=0.01  K_inf=[7.613, 4.585]  P_00= 6.02  rho(A - B K)=0.761
R=0.10  K_inf=[2.762, 2.508]  P_00= 9.08  rho(A - B K)=0.874
R=1.00  K_inf=[0.930, 1.395]  P_00=15.00  rho(A - B K)=0.930
```

また、閉ループの位置軌跡 p_t（初期状態 $\mathbf{x}_0 = (1, 0)^\top$）を 3 つの制御入力ペナルティのもとでプロットできる。

```
trajs, steps_di = [], np.arange(51) * dt
for ru in (0.01, 0.10, 1.00):
    _, Ks = riccati_lqr(A_di, B_di, Q_di, np.array([[ru]]), H=100)
    x, xs_di = np.array([1.0, 0.0]), [1.0]
    for _ in range(50):
        x = (A_di - B_di @ Ks[0]) @ x
        xs_di.append(x[0])
    trajs.append(xs_di)
d2l.plot(steps_di, trajs, 'time (s)', 'position $p_t$',
         legend=['R = 0.01', 'R = 0.10', 'R = 1.00'], figsize=(5, 3))
```



定常フィードバック則 $u_t^* = -k_p p_t - k_v v_t$ の 2 つの成分に注目しよう。動的計画法によって、位置誤差 p_t に対してゲイン k_p で押し戻し、速度 v_t をゲイン k_v で減衰させる**比例・微分**（proportional-derivative、PD）制御器が自動的に導出されている。制御入力が安価なとき

($R=0.01$)、LQR は積極的なゲイン $\mathbf{K}_\infty=[7.613, 4.585]$ を選び、状態ノルムを 1 ステップあたり $\rho(\mathbf{A}-\mathbf{B}\mathbf{K}_\infty)=0.761$ の比率で縮小させる。一方、制御入力が高価なとき ($R=1.00$) は、穏やかな加速度 $\mathbf{K}_\infty=[0.930, 1.395]$ を用い、縮小率は $\rho=0.930$、累積コスト $V_0(\mathbf{x}_0)=\mathbf{x}_0^\top \mathbf{P}_\infty \mathbf{x}_0$ はより大きくなる（$P_{00}=15.00$ 対 6.02）。

14.6.2 平衡点周りでの非線形システムの線形化

14.6.2.1 Pendulum-v1 の解析的線形化

ここで、本章の残りの部分および第 15 章を通じて用いる物理ベンチマークである倒立振子（inverted pendulum、Pendulum-v1）を考えよう。質量 $m=1$、長さ $l=1$ の剛体棒の一端が重力 $g=10$ のもとで軸支されており、有界なモータトルク $u_t \in [-2, 2]$ によって駆動される。連続状態は $\mathbf{x}_t=(\vartheta_t, \dot{\vartheta}_t)^\top$ であり、ϑ_t は鉛直上向きから測った角度（$\vartheta=0$ が直立、$\vartheta=\pi$ が真下への懸垂）、$\dot{\vartheta}_t \in [-8, 8]$ は角速度である。水平位置（$\vartheta=\pi/2$）における最大重力トルクは $mgl/2=5>2$ であるため、このモータは**劣駆動**（underactuated）であり、棒を最下点から頂上まで一息に持ち上げることはできない。シミュレータの 1 ステップ $\Delta t=0.05$ s にわたる半陰的オイラー法（semi-implicit Euler）の遷移 $\mathbf{x}_{t+1}=f(\mathbf{x}_t, u_t)$ は次の式で与えられる。

$$\begin{aligned} \dot{\vartheta}_{t+1} &= \dot{\vartheta}_t + \Big(\frac{3g}{2l}\sin\vartheta_t + \frac{3}{ml^2}u_t\Big)\Delta t, \\ \vartheta_{t+1} &= \vartheta_t + \dot{\vartheta}_{t+1}\Delta t, \end{aligned} \tag{14.54}$$

ただし、角速度は $[-8, 8]$ にクリッピングされる。各ステップでは二次ステージコスト $c(\vartheta_t, \dot{\vartheta}_t, u_t)=\vartheta_t^2+0.1\,\dot{\vartheta}_t^2+0.001\,u_t^2$ が発生し（ここで ϑ_t は $[-\pi, \pi]$ に折り返され、$\mathbf{Q}=\mathrm{diag}(1.0, 0.1)$ および $\mathbf{R}=[[0.001]]$ に対応する）、等価な表現として報酬 $r_t=-c(\vartheta_t, \dot{\vartheta}_t, u_t)$ が得られる。標準的なベンチマークのエピソードは 200 ステップ（10 s）実行される。無制御の振子のスコアが $-1{,}200$ から $-1{,}500$ 程度であるのに対し、棒を振り上げ（swing-up）て直立状態に保つ制御器は、ベンチマークのランダムな初期状態にわたる平均で約 -200 以上のスコアを達成する。本節では既知の運動方程式 Equation 14.54 を用いて直接計画を立てるが、14.9 節、15.5 節、および 15.6 節では、物理法則を知らないエージェントが行動を選択したりダイナミクスモデルを学習したりする場面でこの同じ振子ベンチマークに立ち返る。

直立姿勢 $\mathbf{x}^\ast=(0, 0)^\top$ とゼロトルク $u^\ast=0$ の組は不動点 $f(\mathbf{x}^\ast, u^\ast)=\mathbf{x}^\ast$ である。任意の基準対 $(\bar{\mathbf{x}}, \bar{u})$ の近傍では、1 次のテイラー展開によって非線形遷移を偏差 $\delta\mathbf{x}_t=\mathbf{x}_t-\bar{\mathbf{x}}$ と $\delta u_t=u_t-\bar{u}$ に関する局所線形モデルで近似できる。

$$\begin{aligned} f(\mathbf{x}_t, u_t) - f(\bar{\mathbf{x}}, \bar{u}) &\approx \mathbf{A}(\bar{\mathbf{x}}, \bar{u})\,\delta\mathbf{x}_t + \mathbf{B}(\bar{\mathbf{x}}, \bar{u})\,\delta u_t, \\ \mathbf{A}=\frac{\partial f}{\partial \mathbf{x}}(\bar{\mathbf{x}}, \bar{u}), &\qquad \mathbf{B}=\frac{\partial f}{\partial u}(\bar{\mathbf{x}}, \bar{u}). \end{aligned} \tag{14.55}$$

Equation 14.54 を $(\vartheta_t, \dot{\vartheta}_t)$ および u_t で微分すると、$\partial\dot{\vartheta}_{t+1}/\partial\vartheta_t = \frac{3g}{2l}\cos(\vartheta_t)\Delta t$ および $\partial\dot{\vartheta}_{t+1}/\partial u_t=\frac{3}{ml^2}\Delta t$ が得られる。

```
def pendulum_jacobians(th, dt=0.05, g=10.0):
    """Exact Jacobians (A, B) of Pendulum-v1 at angle th."""
    dthdot_dth = 1.5 * g * np.cos(th) * dt
    dthdot_du = 3.0 * dt
    A = np.array([[1.0 + dthdot_dth * dt, dt], [dthdot_dth, 1.0]])
```

```
    B = np.array([[dthdot_du * dt], [dthdot_du]])
    return A, B

A_p, B_p = pendulum_jacobians(0.0)
Q_p, R_p = np.diag([1.0, 0.1]), np.array([[0.001]])
Ps_p, Ks_p = riccati_lqr(A_p, B_p, Q_p, R_p, H=200)
K_lqr = Ks_p[0][0]
eigs_open = np.linalg.eigvals(A_p).real
eigs_closed = np.linalg.eigvals(A_p - B_p @ Ks_p[0]).real
print(f'A at upright:\n{A_p}')
print(f'open-loop eigenvalues:   {eigs_open[0]:.4f}, {eigs_open[1]:.4f}')
print(f'LQR gain K_inf:          [{K_lqr[0]:.3f}, {K_lqr[1]:.3f}]')
print(f'closed-loop eigenvalues: {eigs_closed[0]:.4f}, {eigs_closed[1]:.4f}')
```

```
A at upright:
[[1.0375 0.05  ]
 [0.75   1.    ]]
open-loop eigenvalues:   1.2133, 0.8242
LQR gain K_inf:          [19.693, 5.263]
closed-loop eigenvalues: 0.8537, 0.2467
```

直立不動点において、開ループのヤコビ行列 $\mathbf{A}$ は不安定な固有値 $\lambda_1 = 1.2133 > 1$ をもつ。そのため、わずかな傾きであっても 1 ステップあたり 21% 超の割合で指数関数的に増大する（38.1 節の相図と同様である）。Pendulum-v1 のコスト行列 $\mathbf{Q} = \mathrm{diag}(1, 0.1)$ と $\mathbf{R} = [[0.001]]$ を用いてリッカチ漸化式を解くと、フィードバックゲイン $\mathbf{K}_\infty = [19.693, 5.263]$ が得られ、閉ループの固有値は両方とも単位円の内側（$\lambda_1 = 0.8537, \lambda_2 = 0.2467$）に引き込まれる。

14.6.2.2 トルク飽和と吸引領域

モータトルクが $[-2, 2]$ にクリッピングされているとき、この線形制御器は頂上からどこまで離れた振子を捕捉できるだろうか。振子を引き下ろす重力トルクは $\frac{1}{2}mgl\sin\vartheta = 5\sin\vartheta$ であるのに対し、モータが供給できるトルクは高々 $|u| = 2$ である。$|\vartheta| > \arcsin(2/5) \approx 0.412\,\mathrm{rad}$（$23.6^\circ$）を超えると、重力がモータの最大保持トルクを上回る。クリッピング付き LQR 方策 $u_t = \mathrm{clip}(-\mathbf{K}_\infty \mathbf{x}_t, -2, 2)$ を、完全な非線形振子に対して静止状態の 5 つの初期角度 ϑ_0 からテストしてみよう。

```
def wrap(th):
    return ((th + np.pi) % (2 * np.pi)) - np.pi

def pendulum_step(x, u, dt=0.05, g=10.0):
    """One step of nonlinear Pendulum-v1 dynamics in (theta, thetadot)."""
    th, thdot = x
    u_c = float(np.clip(u, -2.0, 2.0))
    newthdot = float(np.clip(thdot + (1.5 * g * np.sin(th) + 3.0 * u_c) * dt,
                             -8.0, 8.0))
    newth = th + newthdot * dt
    cost = wrap(th)**2 + 0.1 * thdot**2 + 0.001 * u_c**2
    return np.array([newth, newthdot]), cost

for th0 in (0.20, 0.40, 0.50, 0.60, np.pi):
    x, total_cost = np.array([th0, 0.0]), 0.0
```

```
    for _ in range(200):
        u = -(K_lqr[0] * wrap(x[0]) + K_lqr[1] * x[1])
        x, c = pendulum_step(x, u)
        total_cost += c
    print(f'th0={th0:4.2f} rad: 200-step return={-total_cost:8.2f}, '
          f'final theta={wrap(x[0]):+6.3f} rad')
```

```
th0=0.20 rad: 200-step return=   -0.32, final theta=+0.000 rad
th0=0.40 rad: 200-step return=   -2.97, final theta=+0.000 rad
th0=0.50 rad: 200-step return= -770.94, final theta=+2.592 rad
th0=0.60 rad: 200-step return= -546.37, final theta=+0.001 rad
th0=3.14 rad: 200-step return=-1471.86, final theta=-2.249 rad
```

静的保持限界の内側（$\vartheta_0 = 0.20$ および 0.40 rad）では、線形 LQR はほぼゼロのコストで振子をそのまま直立状態（$\vartheta_{200} = 0.000$）へと導く（収益は -0.32 と -2.97）。このしきい値をわずかに超えると（$\vartheta_0 = 0.50$ rad）、最大の負トルク $u = -2$ でも落下を食い止められず、200 ステップ後も棒は最下点を通過して揺れ続けている（収益 -770.94）。しきい値の先では、結果はもはや ϑ_0 に対して単調ではない。0.60 rad から始めた場合も棒は同様に落下するが、さらに数回揺れたあと、たまたまクリッピング付き制御器が捕捉できるほど十分に遅い速度で上がってくる（収益 -546.37）。これは制御ではなく偶然である。最下点（$\vartheta_0 = \pi$）からは、線形制御器の収益は -1471.86 となり、一度も頂上に到達しない。モータが劣駆動（$u_{\max} = 2 < 5$）であるため、$\vartheta_0 = \pi$ からの振り上げには**エネルギー注入**が必要になる。すなわち、一方向に押し、振子が最下点を通って振り戻る際にトルクを反転させて運動エネルギーと位置エネルギーを蓄え、棒が頂上付近の捕捉領域まで登り切って初めて直立安定化へ切り替える必要がある。

14.6.3 反復 LQR による非線形軌跡最適化

14.6.3.1 軌跡に沿った線形化

$(\mathbf{0}, 0)$ での単一の線形化は、非線形な $\sin(\vartheta)$ の曲率（$\vartheta_0 = \pi$ からのスイングに沿ったもの）について何の情報ももたない。そこで、**反復線形二次制御**（iterative linear-quadratic regulation、iLQR）(Li and Todorov 2004) は、微分動的計画法（differential dynamic programming）(Mayne 1966) のガウス・ニュートン型変種であり、ノミナル軌跡全体の周りで線形化する。

ハードクリッピングによる折れ目を避けてトルク制約 $u_t \in [-2, 2]$ を滑らかに満たすため、$u_t = 2\tanh(v_t)$ とおき、無制約の制御変数 $v_t \in \mathbb{R}$ でパラメータ化する。任意のノミナル制御系列 $\bar{v}_{0:H-1}$ と、その前向きロールアウト $\bar{\mathbf{x}}_{0:H}$（真の非線形ダイナミクス $\mathbf{x}_{t+1} = f(\mathbf{x}_t, v_t)$ に従うもの。なお以下の振り上げでは速度上限に達しないため速度クリッピングは省く）が与えられたとき、iLQR の各反復は次の 2 つのパスを実行する。

1. **$(\bar{\mathbf{x}}_{0:H}, \bar{v}_{0:H-1})$ に沿った後ろ向きリッカチパス：**各ステップ $t = H-1, \dots, 0$ において局所ヤコビ行列 $\mathbf{A}_t = \frac{\partial f}{\partial \mathbf{x}}(\bar{\mathbf{x}}_t, \bar{v}_t)$ と $\mathbf{B}_t = \frac{\partial f}{\partial v}(\bar{\mathbf{x}}_t, \bar{v}_t)$ を計算し、行動価値の摂動 $\delta Q_t(\delta\mathbf{x}_t, \delta v_t)$ の 2 次テイラー近似を、ステージコストの勾配 $(\ell_{\mathbf{x},t}, \ell_{v,t})$、ヘッセ行列 $(\mathbf{L}_{\mathbf{xx},t}, \ell_{vv,t})$、および次ステップの価値勾配 $\mathbf{v}_{\mathbf{x},t+1}$ とヘッセ行列 $\mathbf{V}_{\mathbf{xx},t+1}$ から構成する。

$$
\begin{aligned}
\mathbf{q}_{\mathbf{x},t} &= \ell_{\mathbf{x},t} + \mathbf{A}_t^{\top}\mathbf{v}_{\mathbf{x},t+1}, \qquad q_{v,t} = \ell_{v,t} + \mathbf{B}_t^{\top}\mathbf{v}_{\mathbf{x},t+1},\\
\mathbf{Q}_{\mathbf{xx},t} &= \mathbf{L}_{\mathbf{xx},t} + \mathbf{A}_t^{\top}\mathbf{V}_{\mathbf{xx},t+1}\mathbf{A}_t, \qquad \mathbf{q}_{v\mathbf{x},t} = \mathbf{B}_t^{\top}\mathbf{V}_{\mathbf{xx},t+1}\mathbf{A}_t,\\
q_{vv,t} &= \ell_{vv,t} + \mu + \mathbf{B}_t^{\top}\mathbf{V}_{\mathbf{xx},t+1}\mathbf{B}_t,
\end{aligned}
$$

ここで、$\mu > 0$ はレーベンバーグ・マーカート減衰項である。δv_t に関して最小化すると、

開ループのフィードフォワード補正量 $k_t = -q_{v,t}/q_{vv,t}$ と局所線形フィードバックゲイン $\mathbf{K}_t = -\mathbf{q}_{v\mathbf{x},t}/q_{vv,t}$ が得られる。

2. **バックトラッキング直線探索を伴う前向きロールアウト：**線形モデルは $(\bar{\mathbf{x}}_t, \bar{v}_t)$ の近傍でのみ正確であるため、ステップサイズ $\alpha \in \{1, 1/2, 1/4, \dots\}$ を用いて**真の非線形ダイナミクス**をロールアウトする。

$$\begin{aligned} v_t^{\text{new}} &= \bar{v}_t + \alpha\, k_t + \mathbf{K}_t(\mathbf{x}_t^{\text{new}} - \bar{\mathbf{x}}_t), \\ \mathbf{x}_{t+1}^{\text{new}} &= f(\mathbf{x}_t^{\text{new}}, v_t^{\text{new}}), \end{aligned} \tag{14.56}$$

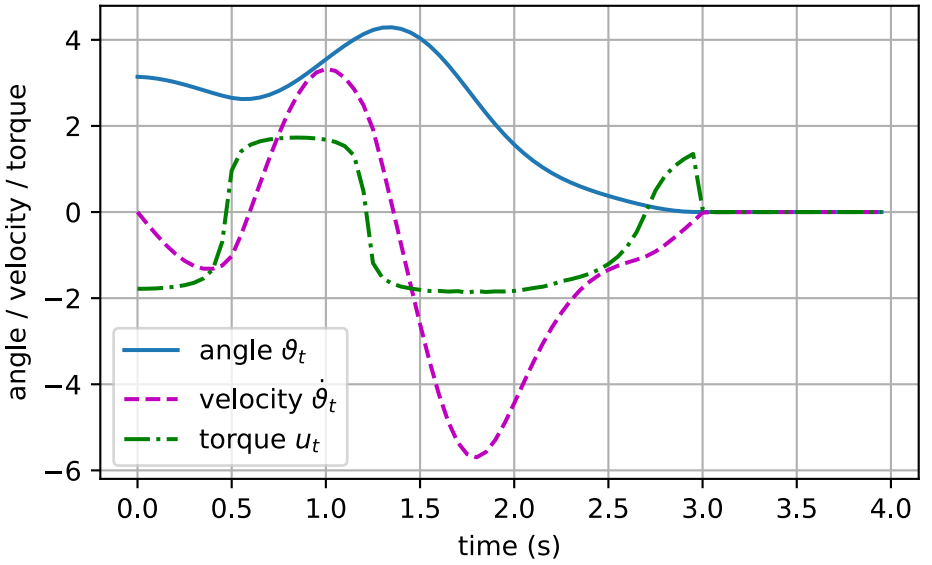
そして、真の非線形軌跡コストを減少させる最初の α を採用する。

軌跡コストの設計における 1 つの工夫は示唆に富む。$\vartheta_0 = \pi$ からエネルギーを注入するには、振子はまず反対側の $\vartheta > \pi$ へと振れてから、勢いよく $\vartheta = 0$ へ向かって振り戻る必要がある。もし重い二次ペナルティ ϑ_t^2 を序盤のすべてのステップ t に課すと、局所勾配法は 0 から離れる方向へ振ることを拒んでしまう。バックスイング中に ϑ_t^2 が増加するからである。そこで、スイング段階では状態コストの重みを軽くし（$q_{\text{run}} = 0.02$、$t < H - 20$）、終盤の捕捉区間では重くする（$q_T = 100$、$t \geq H - 20$）ことで、iLQR は序盤のステップを使ってエネルギーを注入し、終盤のステップで棒を $(\vartheta, \dot{\vartheta}) = (0, 0)$ に固定できる。

```
def step_tanh(x, v, dt=0.05, g=10.0):
    """Nonlinear step and analytical Jacobians (A_t, B_t) for u = 2 tanh(v)."""
    th, thdot = x[0], x[1]
    th_v = np.tanh(v)
    u = 2.0 * th_v
    du_dv = 2.0 * (1.0 - th_v**2)
    newthdot = thdot + (1.5 * g * np.sin(th) + 3.0 * u) * dt
    newth = th + newthdot * dt
    dthdot_dth = 1.5 * g * np.cos(th) * dt
    dthdot_dv = 3.0 * dt * du_dv
    A = np.array([[1.0 + dthdot_dth * dt, dt], [dthdot_dth, 1.0]])
    B = np.array([[dthdot_dv * dt], [dthdot_dv]])
    return np.array([newth, newthdot]), A, B, u

def ilqr_swingup(x0, H=80, iters=35, q_run=0.02, r_v=0.02, q_T=100.0,
                 vs_init=None):
    """iLQR trajectory optimization from x0 over horizon H."""
    vs = (np.array(vs_init, dtype=float) if vs_init is not None
          else np.array([0.2 if t < 15 else 0.0 for t in range(H)]))
    Q_base = np.diag([1.0, 0.1])
    for _ in range(iters):
        xs, As, Bs, us, cost = [np.array(x0, dtype=float)], [], [], [], 0.0
        for t in range(H):
            x_next, A, B, u = step_tanh(xs[-1], vs[t])
            As.append(A); Bs.append(B); us.append(u)
            w = q_T if t >= H - 20 else q_run
            cost += w * (xs[-1] @ Q_base @ xs[-1]) + r_v * vs[t]**2
            xs.append(x_next)
        cost += q_T * (xs[-1] @ Q_base @ xs[-1])

        Vx = 2.0 * q_T * (Q_base @ xs[-1])
        Vxx = 2.0 * q_T * Q_base
```

```
        ks, Ks = np.zeros(H), np.zeros((H, 2))
        for t in range(H - 1, -1, -1):
            w = q_T if t >= H - 20 else q_run
            A, B = As[t], Bs[t]
            Qx = 2.0 * w * (Q_base @ xs[t]) + A.T @ Vx
            Qv = 2.0 * r_v * vs[t] + (B.T @ Vx).item()
            Qxx = 2.0 * w * Q_base + A.T @ Vxx @ A
            Qvv = 2.0 * r_v + 1e-2 + (B.T @ Vxx @ B).item()
            Qvx = (B.T @ Vxx @ A).ravel()
            ks[t], Ks[t] = -Qv / Qvv, -Qvx / Qvv
            Vx = Qx + Qvx * ks[t]
            Vxx = Qxx + np.outer(Qvx, Ks[t])

        for alpha in (1.0, 0.5, 0.25, 0.1, 0.03):
            x_new, vs_new, c_new = np.array(x0, dtype=float), np.zeros(H), 0.0
            for t in range(H):
                v_t = vs[t] + alpha * ks[t] + Ks[t] @ (x_new - xs[t])
                vs_new[t] = v_t
                w = q_T if t >= H - 20 else q_run
                c_new += w * (x_new @ Q_base @ x_new) + r_v * v_t**2
                x_new, _, _, _ = step_tanh(x_new, v_t)
            c_new += q_T * (x_new @ Q_base @ x_new)
            if c_new < cost:
                vs = vs_new
                break
    return np.array(xs), np.array(us), vs, Ks, cost

xs_nom, us_nom, vs_nom, Ks_nom, _ = ilqr_swingup([np.pi, 0.0], H=80, iters=35)
for t in range(0, 81, 10):
    u_t = us_nom[t] if t < 80 else 0.0
    print(f't={t:2d}  theta={xs_nom[t, 0]:7.4f}  '
          f'thetadot={xs_nom[t, 1]:7.4f}  u={u_t:+6.3f}')
d2l.plot(np.arange(80) * 0.05, [xs_nom[:-1, 0], xs_nom[:-1, 1], us_nom],
         'time (s)', 'angle / velocity / torque',
         legend=[r'angle $\vartheta_t$', r'velocity $\dot{\vartheta}_t$',
                 r'torque $u_t$'], figsize=(5.5, 3.2))
```

```
t= 0  theta= 3.1416  thetadot= 0.0000  u=-1.782
t=10  theta= 2.6515  thetadot=-1.0260  u=+0.965
t=20  theta= 3.5453  thetadot= 3.3225  u=+1.681
t=30  theta= 4.0383  thetadot=-2.6013  u=-1.807
t=40  theta= 1.5668  thetadot=-4.4402  u=-1.832
t=50  theta= 0.3746  thetadot=-1.3417  u=-1.209
t=60  theta= 0.0009  thetadot=-0.0132  u=+0.066
t=70  theta= 0.0002  thetadot=-0.0006  u=-0.000
t=80  theta= 0.0001  thetadot=-0.0000  u=+0.000
```



iLQR が 35 回の反復によって最下点 $(\pi, 0)$ から発見した軌跡を見てみよう。

- **ステップ 0〜10（初期のバックスイング）：**左方向へ押し（$u_0 = -1.782$）、$\vartheta_{10} = 2.652$ rad まで振る。
- **ステップ 10〜30（エネルギー注入）：**トルクを反転させ（$u_{10} = +0.965 \to u_{20} = +1.681$）、最下点を通過して反対側の $\vartheta_{30} = 4.038$ rad まで振子を駆動し、重力位置エネルギーを蓄える。
- **ステップ 30〜60（振り抜きと捕捉）：**再びトルクを反転させ（$u_{30} = -1.807 \to u_{40} = -1.832$）、最大 5.7 rad/s で最下点を通過し、$\dot{\vartheta}_{40} = -4.440$ rad/s の速度を保ったまま水平位置（$\vartheta_{40} = 1.567$ rad）を越えて直立の捕捉領域（$\vartheta_{50} = 0.375$ rad）へと駆け上がり、滑らかに制動して頂上で静止する（$\vartheta_{60} = 0.0009$ rad、$\vartheta_{80} = 0.0001$ rad、$u_{80} = 0.000$）。

この計画は局所最適解であり、最良の振り上げだと誤解してはならない。iLQR の各反復は与えられた軌跡を改善するものであるため、結果は初期制御系列に依存する。`ilqr_swingup` は最初の 15 ステップにわたる小さな押し（$\bar{v}_t = 0.2$）とそれ以降のゼロ入力から出発し、固定された減衰係数 μ を用いている。この初期の押しは重要であり、$\bar{v} = 0$ から始めると、同じ 35 回の反復では棒を頂上で静止させることができない。初期推定値へのこの依存性は以下で再び現れ、制御器が衝撃から回復できるかどうかを左右することになる。

14.6.4 移動ホライズンモデル予測制御

14.6.4.1 開ループ計画、局所フィードバックチューブ、オンライン再計画

最適軌跡 $(\bar{\mathbf{x}}_{0:H}, \bar{u}_{0:H-1})$ は、それだけではまだ頑健な制御器ではない。この計画を実行するには次の 3 通りの方法がある。

1. **開ループ実行：**事前計算された系列 $\bar{u}_t$ を、実際の状態 $\mathbf{x}_t$ を見ずにそのまま適用する。直立姿勢は不安定な鞍点であるため、わずかな外乱や数値ドリフトでも軌跡はコースから外れてしまう。
2. **時変 LQR（time-varying LQR、TV-LQR）追従：**iLQR の最後の後ろ向きパスで計算された局所フィードバック行列 $\mathbf{K}_t$ を用い、$v_t = \bar{v}_t + \mathbf{K}_t(\mathbf{x}_t - \bar{\mathbf{x}}_t)$ を適用する。これによりオンライン計画コストなしでノミナル軌跡の周りに局所線形フィードバックチューブが形成

されるが、強い衝撃によって $\mathbf{x}_t$ がチューブの外へ押し出され、新たな振り上げのためのエネルギー注入が必要な領域に入ると破綻する。

3. **移動ホライズンモデル予測制御（MPC）**：各タイムステップ t において真の状態 $\mathbf{x}_t$ を観測し、$\mathbf{x}_t$ を始点とする先読みウィンドウ上で有限ホライズン制御問題を解き直し（前ステップの計画を 1 ステップずらしてウォームスタートする）、**最初の制御入力のみ** $u^*_{t|t}$ を適用して、これを $t+1$ で繰り返す (Rawlings et al. 2017)。

ここで実装する `mpc_action` では、3 番目の方式における 2 つの細部を定めている。第 1 はウィンドウの長さである。シフトされた計画は直前の計画より 1 ステップ短くなるため、ウィンドウはノミナル計画の 80 ステップから始まり、それとともに短縮していく。残り `H_min` $= 25$ ステップになった時点で計画の末尾をゼロで埋め、以降はウィンドウをその長さに保ったまま移動させる。第 2 はウォームスタートが失敗したときの対処である。iLQR は局所手法であるため、ウォームスタートされた計画は、元の軌跡がもはや頂上へ到達しなくなった場合でも古い軌跡の近くにとどまってしまう。再最適化した計画の終端が直立位置から 0.2 rad を超えて離れている場合、`mpc_action` はそれを破棄し、`H_mpc` $= 60$ ステップ（3 s）にわたる新たな振り上げを、`ilqr_swingup` がデフォルトで用いる初期の押しから計画し直す。

3 つの制御器すべてを、$T = 120$ ステップにわたり初期状態 $\mathbf{x}_0 = (\pi, 0)$ から比較しよう。振子がちょうど水平位置を越えて直立の捕捉領域へ登っていくステップにおいて、外部から角速度インパルス $\Delta\dot{\vartheta} \in \{0.0, +0.5, +2.2\}$ rad/s がステップ $t = 45$ で加わるものとする。

```
T = 120
xs_pad = np.vstack([xs_nom, np.zeros((T - 80, 2))])
vs_pad = np.concatenate([vs_nom, np.zeros(T - 80)])
us_pad = np.concatenate([us_nom, np.zeros(T - 80)])
Ks_pad = np.vstack([Ks_nom, np.tile(Ks_nom[-1], (T - 80, 1))])

def mpc_action(x, vs_warm, H_mpc=60, H_min=25):
    """One MPC step: re-optimize the shifted plan (at least H_min steps) from x;
    if it no longer ends at the top, plan afresh over H_mpc steps."""
    H = max(H_min, len(vs_warm))
    warm = np.concatenate([vs_warm, np.zeros(H)])[:H]
    xs_w, us_w, vs_w, _, _ = ilqr_swingup(x, H=H, iters=18, vs_init=warm)
    if abs(xs_w[-1, 0]) > 0.2:   # warm plan missed the top: re-seed
        _, us_s, vs_s, _, _ = ilqr_swingup(x, H=H_mpc, iters=25)
        return us_s[0], vs_s[0], vs_s[1:]
    return us_w[0], vs_w[0], vs_w[1:]

for impulse in (0.0, 0.5, 2.2):
    x_ol, x_tv, x_mpc = [np.array([np.pi, 0.0]) for _ in range(3)]
    ret_ol = ret_tv = ret_mpc = 0.0
    th_ol, th_tv, th_mpc = [], [], []
    vs_warm = vs_nom.copy()
    for t in range(T):
        if t == 45:
            for x in (x_ol, x_tv, x_mpc):
                x[1] += impulse
        th_ol.append(wrap(x_ol[0]))
        th_tv.append(wrap(x_tv[0]))
        th_mpc.append(wrap(x_mpc[0]))
        # 1. Open-loop
        ret_ol -= wrap(x_ol[0])**2 + 0.1 * x_ol[1]**2 + 0.001 * us_pad[t]**2
```

```
            x_ol, _, _, _ = step_tanh(x_ol, vs_pad[t])
            # 2. Time-varying LQR tracking
            v_tv = vs_pad[t] + Ks_pad[t] @ (x_tv - xs_pad[t])
            u_tv = 2.0 * np.tanh(v_tv)
            ret_tv -= wrap(x_tv[0])**2 + 0.1 * x_tv[1]**2 + 0.001 * u_tv**2
            x_tv, _, _, _ = step_tanh(x_tv, v_tv)
            # 3. Receding-horizon MPC
            u_m, v_m, vs_warm = mpc_action(x_mpc, vs_warm)
            ret_mpc -= wrap(x_mpc[0])**2 + 0.1 * x_mpc[1]**2 + 0.001 * u_m**2
            x_mpc, _, _, _ = step_tanh(x_mpc, v_m)
        print(f'impulse={impulse:+.1f} rad/s at t=45:')
        print(f'  Open-loop:  final theta={wrap(x_ol[0]):+6.3f}, return={ret_ol:6.1f}')
        print(f'  TV-LQR:     final theta={wrap(x_tv[0]):+6.3f}, return={ret_tv:6.1f}')
        print(f'  MPC (iLQR): final theta={wrap(x_mpc[0]):+6.3f}, return={ret_mpc:6.1f}')
    d2l.plot(np.arange(T) * 0.05, [th_ol, th_tv, th_mpc],
             'time (s)', r'wrapped angle $\vartheta_t$ (rad)',
             legend=['Open-loop', 'TV-LQR', 'MPC (iLQR)'], figsize=(5.5, 3.2))
```

```
impulse=+0.0 rad/s at t=45:
  Open-loop:  final theta=+0.075, return=-333.6
  TV-LQR:     final theta=+0.000, return=-333.7
  MPC (iLQR): final theta=+0.000, return=-331.6
impulse=+0.5 rad/s at t=45:
  Open-loop:  final theta=-2.939, return=-502.3
  TV-LQR:     final theta=-0.001, return=-333.9
  MPC (iLQR): final theta=+0.000, return=-331.7
impulse=+2.2 rad/s at t=45:
  Open-loop:  final theta=+2.941, return=-621.1
  TV-LQR:     final theta=+1.959, return=-797.9
  MPC (iLQR): final theta=-0.000, return=-558.2
```

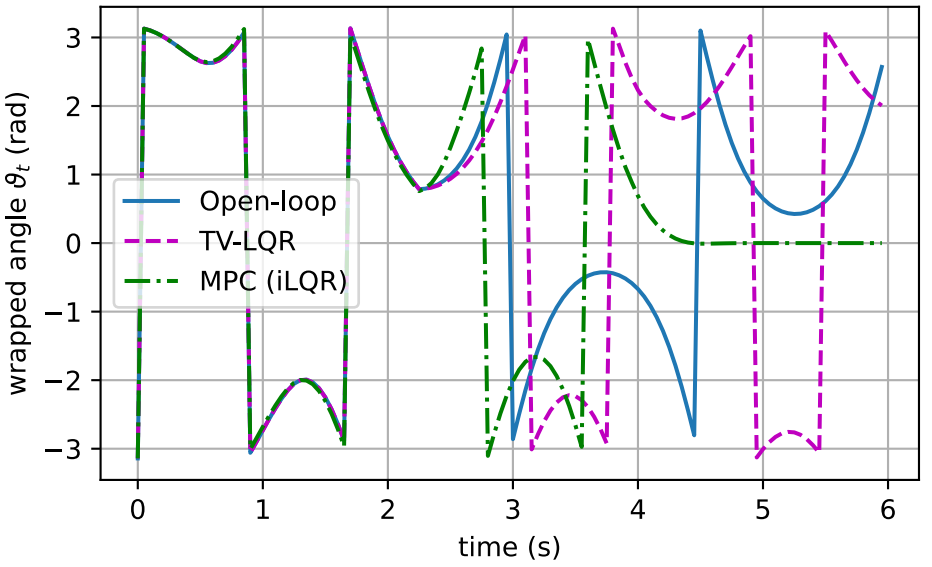


Figure 14.17 に本節の 3 つの段階を並べて示す。直立平衡点周りの局所 LQR の価値等高線とト

ルク飽和境界、$(\vartheta, \dot{\vartheta})$ 位相平面における iLQR のエネルギー注入軌跡、そして強いインパルス $\Delta\dot{\vartheta} = +2.2$ rad/s（$t = 45$）のもとでの外乱抑制（disturbance rejection）軌跡である。

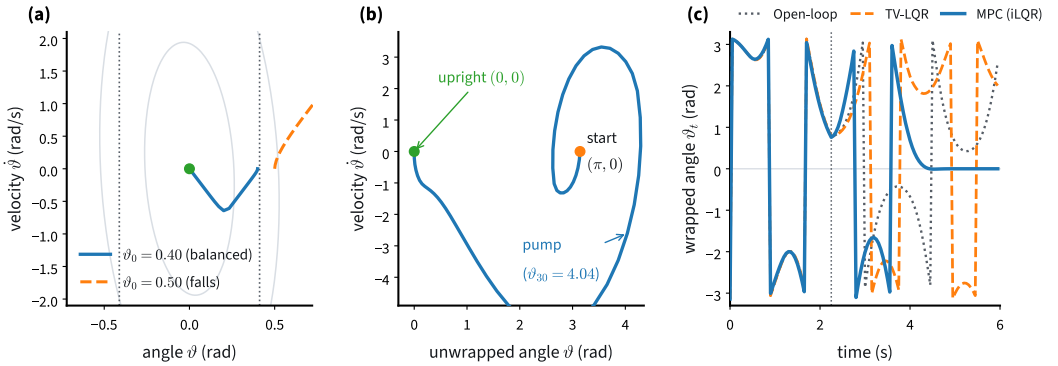


Figure 14.17: 局所 LQR から移動ホライズン MPC までの最適制御。(a) 直立 LQR の残りコスト等高線 $V_\infty(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_\infty \mathbf{x}$ と静的保持境界 $|\vartheta| \le \arcsin(2/5) \approx 0.412$ rad（点線）。$\vartheta_0 = 0.40$ rad からの出発では原点に収束する（青）のに対し、$\vartheta_0 = 0.50$ rad ではモータトルク $|u| \le 2$ が飽和して落下する（オレンジ破線）。(b) 80 ステップの iLQR 振り上げ軌跡（$(\vartheta, \dot{\vartheta})$ 位相平面）。$(\pi, 0)$ から $\vartheta_{30} = 4.04$ rad までエネルギーを注入したのち、最下点を振り抜けて直立平衡点 $(0, 0)$ に至る。(c) 折り返し角度 ϑ_t の時間変化（インパルス $\Delta\dot{\vartheta} = +2.2$ rad/s が $t = 45$ すなわち 2.25 s の縦点線で加わった場合）。開ループと TV-LQR は失敗するが、移動ホライズン MPC は新たな振り上げを計画して $\vartheta = 0$ に戻る。

この振子ベンチマークにおける 3 段階の外乱レベルは、3 つの制御器の特性を鮮明に描き分ける。

- **外乱なし（$\Delta\dot{\vartheta} = 0.0$）：**3 つとも振子を振り上げ、120 ステップにわたって -332 から -334 の収益を得る。これがベンチマークの目安として挙げた -200 を下回るのは 2 つの理由による。第 1 に、ベンチマークがランダムな初期状態にわたって平均するのに対し、ここでは最悪の状態である最下点から出発していること。第 2 に、iLQR は収益を採点する振子の報酬そのものではなく、スイング中の状態重みを軽くした独自の軌跡コストを最小化していることである。$t = 80$ 以降の最後の 40 ステップでは、開ループは不安定な鞍点で $\vartheta_{120} = +0.075$ rad へとドリフトし始めるが、TV-LQR と MPC は $\vartheta_{120} = 0.000$ rad を維持する。
- **中程度の外乱（$\Delta\dot{\vartheta} = +0.5$ rad/s）：**開ループ実行は事前計算されたスケジュールから弾き出されて再び落下し、最終的に $\vartheta_{120} = -2.939$ rad（収益 -502.3）に終わる。一方、TV-LQR（$\vartheta_{120} = -0.001$ rad、収益 -333.9）と MPC（$\vartheta_{120} = +0.000$ rad、収益 -331.7）はどちらも衝撃を吸収し、棒を直立状態に固定する。
- **強い外乱（$\Delta\dot{\vartheta} = +2.2$ rad/s）：**衝撃によって上昇が食い止められ、振子は最下点に向かって落下し始める。古いノミナル軌跡周りの TV-LQR の線形フィードバックはトルクが飽和して破綻する（$\vartheta_{120} = +1.959$ rad、収益 -797.9）。回復には古い軌跡の線形近傍には存在しない新たな振り上げのエネルギー注入が必要だからである。MPC は振子を直立平衡点へと復帰させる（$\vartheta_{120} = 0.000$ rad、収益 -558.2）が、これは再シード（re-seed）規則のおかげである。衝撃が加わったステップでウォームスタート計画が頂上を外すため、`mpc_action` は外乱後の状態から新たな振り上げを計画し、以降はそれに従う。ウォームスタート計画は古い軌跡の近くにとどまるため、ウォームスタートからの再計画だけでは回復できない。

14.6.4.2 古典的制御から強化学習へ

Table 14.2 は古典的最適制御と強化学習の対応関係をまとめた対訳表であり、本章の以降の節および第 15 章が LQR と iLQR の各仮定をどのように緩和していくかを示している。

Table 14.2: 古典的最適制御と強化学習の架け橋。

概念	古典的最適制御	強化学習
変数	状態 $\mathbf{x}_t \in \mathbb{R}^d$、制御入力 $\mathbf{u}_t \in \mathbb{R}^m$	状態 $s_t \in \mathcal{S}$、行動 $a_t \in \mathcal{A}$
目的関数	累積コスト $\sum_t c(\mathbf{x}_t, \mathbf{u}_t)$ の最小化	期待割引収益 $\mathbb{E}[\sum_t \gamma^t r(s_t, a_t)]$ の最大化
価値関数	残りコスト $V_t(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_t \mathbf{x}$（LQR）	状態価値 $V^\pi(s)$ または行動価値 $Q^\pi(s,a)$
ベルマンバックアップ	リッカチ漸化式 Equation 14.53	ベルマン期待値・最適作用素（14.2 節）
既知の非線形物理系 f	勾配ベースの軌跡最適化（iLQR・DDP）＋ MPC	木探索またはサンプリングベースの MPC（15.6 節）
未知ダイナミクス、学習モデル $\hat{f}_\phi$	システム同定＋確実性等価 MPC	ランダムシューティング・CEM によるモデルベース強化学習（15.6 節）
未知ダイナミクス、モデルなし	モデルフリー適応制御	方策勾配（14.7 節、14.9 節）、PPO（15.2 節）、DQN（15.4 節）、SAC（15.5 節）

ダイナミクス f が未知であり、しかも遷移モデルを当てはめずに行動しなければならない場合、後ろ向きリッカチパスに必要なヤコビ行列 $\mathbf{A}_t = \partial f / \partial \mathbf{x}$ と $\mathbf{B}_t = \partial f / \partial \mathbf{u}$ は計算できない。14.7 節では、確率的方策 $\pi_\theta(a \mid s)$ が自身のサンプリングした行動の対数確率を代わりに微分することで、これらの未知の導関数をいかに迂回するかを示す。

14.6.5 まとめ

線形ダイナミクス $\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t$ と二次コスト $\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u}$ に対して、ベルマンの後ろ向き動的計画法は連続空間における厳密な閉形式解をもつ。すなわち、残りコストは二次形式 $V_t(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_t \mathbf{x}$ となり、$\mathbf{P}_t$ は後ろ向きリッカチ漸化式に従い、最適方策は線形フィードバック制御器 $\mathbf{u}_t^* = -\mathbf{K}_t \mathbf{x}_t$（LQR）で与えられる。非線形システムを不安定不動点の周りで解析的ヤコビ行列によって線形化すると、アクチュエータ制限下の捕捉領域内にある状態は安定化できるが、その外側で劣駆動の振り上げによるエネルギー注入が必要になると破綻する。反復 LQR（iLQR）はノミナル軌跡の周りでダイナミクスの線形化とコストの二次近似を繰り返し、後ろ向きリッカチパスと前向き非線形ロールアウトによって多段階のエネルギー注入軌跡を発見する。開ループ計画の実行は外乱のもとで破綻し、固定軌跡周りの時変 LQR フィードバックは小さな摂動しか扱えないが、移動ホライズンモデル予測制御（MPC）は各ステップで現在の状態から有限ホライズン問題を解き直すため、大きな外乱によって全く新しい軌跡が必要になった場合でも回復できる。

14.6.6 演習問題

1. **リッカチバックアップにおける平方完成。**$\mathbf{S}_{t+1} = \mathbf{R} + \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{B}$ および $\mathbf{K}_t = \mathbf{S}_{t+1}^{-1} \mathbf{B}^\top \mathbf{P}_{t+1} \mathbf{A}$ とおく。
   1. 積を展開することで、任意の $\mathbf{x} \in \mathbb{R}^d$ と $\mathbf{u} \in \mathbb{R}^m$ に対して

$$Q_t(\mathbf{x}, \mathbf{u}) = \mathbf{x}^\top \mathbf{P}_t \mathbf{x} + (\mathbf{u} + \mathbf{K}_t \mathbf{x})^\top \mathbf{S}_{t+1} (\mathbf{u} + \mathbf{K}_t \mathbf{x}),$$

   が成り立つことを示せ（ここで $\mathbf{P}_t$ は Equation 14.53 で与えられる）。

   2. $\mathbf{R} \succ \mathbf{0}$ と $\mathbf{P}_{t+1} \succeq \mathbf{0}$ を用いて、$\mathbf{S}_{t+1} \succ \mathbf{0}$ であること、$\mathbf{u}_t^* = -\mathbf{K}_t\mathbf{x}$ が $Q_t(\mathbf{x},\mathbf{u})$ の一意な大域的最小点であること、および $\mathbf{P}_t \succeq \mathbf{0}$ であることを導け。
   3. Joseph の安定化形式
   $$\mathbf{P}_t = \mathbf{Q} + \mathbf{K}_t^\top \mathbf{R}\mathbf{K}_t + (\mathbf{A}-\mathbf{B}\mathbf{K}_t)^\top \mathbf{P}_{t+1}(\mathbf{A}-\mathbf{B}\mathbf{K}_t), \tag{14.57}$$
   が成り立つことを確かめ、$\mathbf{K}_t$ を任意のフィードバック行列 $\mathbf{K}$ に置き換えても右辺のすべての項が明らかに半正定値となる理由を説明せよ。
2. **加法的ガウスノイズのもとでの確実性等価**。線形ダイナミクス Equation 14.48 が、平均ゼロで独立なノイズベクトル $\mathbf{w}_0,\dots,\mathbf{w}_{H-1}$（共分散 $\mathbb{E}[\mathbf{w}_t\mathbf{w}_t^\top] = \Sigma$）によって摂動され、$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t$ となると仮定する。
   1. 後ろ向き帰納法により、最適残りコストが $V_t(\mathbf{x}) = \mathbf{x}^\top\mathbf{P}_t\mathbf{x} + c_t$ となることを証明せよ。ここで、$\mathbf{P}_t$ は決定論的な場合と全く同じリッカチ漸化式 Equation 14.53 に従い、$c_t = \sum_{k=t+1}^{H} \operatorname{tr}(\mathbf{P}_k\Sigma)$ である。
   2. 最適フィードバック方策 $\mathbf{u}_t^* = -\mathbf{K}_t\mathbf{x}_t$ がノイズの共分散 Σ に依存しないことを導け。もしノイズが乗法的であり、$\mathbf{x}_{t+1} = (\mathbf{A}+\tilde{\mathbf{A}}_t)\mathbf{x}_t + (\mathbf{B}+\tilde{\mathbf{B}}_t)\mathbf{u}_t$ である場合、この確実性等価の性質はどこで破綻するか。
3. [code] **可制御性と無限ホライズン極限**。線形システム $(\mathbf{A},\mathbf{B})$（状態数 d）は、可制御性行列 $\mathcal{C} = [\mathbf{B}, \mathbf{A}\mathbf{B}, \dots, \mathbf{A}^{d-1}\mathbf{B}]$ が行フルランク d をもつとき**可制御**であるという。
   1. $\mathbf{x}_0 = \mathbf{0}$ から出発したとき、d ステップ後に到達する状態が $\mathbf{x}_d = \mathcal{C}\,[\mathbf{u}_{d-1}^\top, \mathbf{u}_{d-2}^\top, \dots, \mathbf{u}_0^\top]^\top$ に等しくなることを示し、$\mathcal{C}$ が行フルランクをもつことが $\mathbb{R}^d$ 内の任意の目標状態に d ステップで到達するための必要十分条件であることを示せ。
   2. $\operatorname{rank}(\mathcal{C})$ と $\det(\mathcal{C})$ を、二重積分器 $(\mathbf{A}_{\text{di}},\mathbf{B}_{\text{di}})$ および線形化した直立振子 $(\mathbf{A}_p,\mathbf{B}_p)$ に対して計算せよ。
   3. 第 1 座標に制御が作用しない（$\mathbf{B}_{\text{un}} = [0,0]^\top$ に相当する）非結合な不安定システム $\mathbf{A} = \operatorname{diag}(1.2, 0.8)$, $\mathbf{B} = [0,1]^\top$ を設定し、`riccati_lqr` を $H \in \{10, 50, 100\}$ に対して実行せよ。$\mathbf{P}_0$ のどの成分が $H \to \infty$ で発散するか。また、それはなぜか。
4. [code] **エネルギー整形と軌跡最適化の比較**。古典的な非線形制御では、振子の全力学的エネルギー $E(\vartheta,\dot\vartheta) = \tfrac{1}{6}ml^2\dot\vartheta^2 + \tfrac{1}{2}mgl\cos\vartheta$ を制御して振り上げることが多い。この値は $E^* = \tfrac{1}{2}mgl = 5$（直立静止平衡点 $(\vartheta=0,\dot\vartheta=0)$）および -5（最下点 $(\vartheta=\pi,\dot\vartheta=0)$）に等しい (Tedrake 2023)。
   1. 連続時間の摩擦のない軌跡（$\ddot\vartheta = \frac{3g}{2l}\sin\vartheta + \frac{3}{ml^2}u$）に沿ったエネルギーの時間変化率が $\dot E = u\dot\vartheta$ となることを示し、トルク $u = -k_E(E-E^*)\dot\vartheta$（$k_E > 0$）を加えると $(E-E^*)^2$ が $\dot\vartheta \neq 0$ である限り単調に減少することを示せ。
   2. `pendulum_step` 上でハイブリッド制御器を実装せよ。すなわち、クリッピング付きエネルギー注入 $u = \operatorname{clip}(-k_E(E-E^*)\dot\vartheta, -2, 2)$（微小な初期キック $u_0 = 2$ を伴う）を $|\operatorname{wrap}(\vartheta)| > 0.35$ のときに適用し、直立 LQR 制御器 $u = \operatorname{clip}(-\mathbf{K}_\infty\mathbf{x}, -2, 2)$ へ $|\operatorname{wrap}(\vartheta)| \le 0.35$ となった時点で切り替えるものとする。200 ステップの $(\pi, 0)$ からの収益を iLQR の収益と比較せよ。
5. [code] **MPC におけるホライズン長と終端コスト**。`mpc_action` はシフトされた計画を少なくとも `H_min` $= 25$ ステップにわたって再最適化し、その計画が頂上を外したときには `H_mpc` $= 60$ ステップ（3 s）にわたる新たな振り上げを計画する。
   1. 本節の $\Delta\dot\vartheta = +2.2$ の実験を `H_mpc` $\in \{15, 25, 40, 60\}$ で繰り返せ。どのホライズンで MPC は振子を頂上へ戻せるか。それらをノミナルな振り上げが頂上に到達するのに要するステップ数と比較し、短いホライズンではプランナが最下点付近に閉じ込められる理由を説明せよ。
   2. 終端コスト行列 $q_T\mathbf{Q}_{\text{base}}$（`ilqr_swingup` 内）を、無限ホライズンの DARE 行列 $\mathbf{P}_\infty$（直立平衡点で `riccati_lqr` により計算したもの）に置き換えよ。すでに直立の捕捉領域

内（$|\vartheta_0| \le 0.35$）にある初期状態に対して、終端コスト $\mathbf{x}_H^\top \mathbf{P}_\infty \mathbf{x}_H$ をもつ移動ホライズン LQR 制御器は、**厳密な**無限ホライズン最適制御 $\mathbf{u}_t^* = -\mathbf{K}_\infty \mathbf{x}_t$ を、先読みがわずか 1 ステップ（$H = 1$）のときでさえ生成することを示せ。

6. **Pontryagin の共状態と価値勾配。** $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ を滑らかな決定論的システムとし、$J = \sum_{t=0}^{H-1} c(\mathbf{x}_t, \mathbf{u}_t) + c_H(\mathbf{x}_H)$ とおく。ラグランジュ乗数（**共状態**または**随伴変数**とよばれる）$\lambda_1, \dots, \lambda_H \in \mathbb{R}^d$ を、制約 $\mathbf{x}_{t+1} - f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{0}$ に対して導入する。
   1. ラグランジアン $\mathcal{L} = J + \sum_{t=0}^{H-1} \lambda_{t+1}^\top (f(\mathbf{x}_t, \mathbf{u}_t) - \mathbf{x}_{t+1})$ を $\mathbf{x}_t$ と $\mathbf{u}_t$ で微分し、後ろ向き随伴方程式

$$\lambda_H = \nabla_{\mathbf{x}} c_H(\mathbf{x}_H), \qquad \lambda_t = \nabla_{\mathbf{x}} c(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{A}_t^\top \lambda_{t+1}, \tag{14.58}$$

   および勾配 $\nabla_{\mathbf{u}_t} J = \nabla_{\mathbf{u}} c(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{B}_t^\top \lambda_{t+1}$ を導出せよ (Pontryagin et al. 1962)。
   2. この後ろ向き随伴漸化式を、1 次のベクトル $\mathbf{v}_{\mathbf{x},t}$ および $q_{v,t}$（`ilqr_swingup` の後ろ向きパスにおけるもの）、ならびに 8.5 節の通時的誤差逆伝播法と比較せよ。iLQR の 2 次のヘッセ行列の漸化式 $\mathbf{V}_{\mathbf{xx},t}$ は、1 次の通時的誤差逆伝播法に対して何を付け加えているか。

14.7 方策勾配

14.5 節では、デモンストレーションされた各行動の対数確率（log probability）に対して、すべての行動の重みを 1 とした上昇ステップを実行することで、方策（policy）がエキスパートから学習した。また 14.6 節では、プランナが既知の遷移方程式 $f(\mathbf{x}_t, \mathbf{u}_t)$ を微分して後ろ向きリッカチゲインを計算した。ではここで、エキスパートも環境の既知で微分可能なモデルも存在せず、エージェント自身の試行と各試行が得た報酬（reward）だけがある状況を考えてみよう。自然な推測は、クローニングの更新式をそのまま残して重みだけを変えることである。すなわち、エージェントがとったすべての行動の対数確率を、その行動が属するエピソードの収益（return）で重み付けして引き上げ、報われたエピソードの行動ほど起こりやすくなるようにする。この推測には理論的な根拠があるのだろうか。方策のパラメータ θ が動いたとき、期待収益はどのように変化するのだろうか。

Ronald Williams は 1992 年にこの問いに答えた (Williams 1992)。推測した更新の平均は期待収益の勾配（gradient）に一致するため、この推測は確率的勾配上昇法（stochastic gradient ascent）そのものになっており、Williams はこのアルゴリズムを REINFORCE と名付けた。エピソードの収益はエージェントが知らない環境の遷移確率に依存するにもかかわらず、この推定に必要なのはサンプリングされたエピソードと方策自身の対数確率だけである。方策を関数 $\pi_\theta(a \mid s)$ として書き下すことは、14.3 節の価値テーブルが抱えていた制約も取り除く。テーブル上で貪欲に行動するには有限個の行動リストにわたる最大値が必要になるが、パラメータ化された方策であればトルクを出力することも、トークンの語彙（vocabulary）上の分布を出力することもできる。同じ恒等式を言語モデル（language model）の次トークン分布に適用すれば、完成した回答に与えられる単一の報酬から推論モデルを訓練できる。

本節では、方策勾配（policy gradient）を持つ最小の問題からこの推定量を導出し、すべての勾配を厳密に計算できる 14.1 節の検証器タスク（verifier task）および湖の環境で実行する。この推定量は平均的には正しいが、更新のたびに新しいエピソードを必要とし、小さなバッチでは大きなノイズが残る。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
```

```
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

14.7.1 方策のパラメータ化

勾配上昇法には、パラメータに対して滑らかに変化する方策が必要である。決定論的な方策はargmax であり、区分的に定数となる。そのため、パラメータをわずかに動かしても何も変わらないか、行動が不連続に切り替わるかのどちらかになる。一方、確率的な方策の確率は任意の微小量だけ変化できる。

14.7.1.1 ソフトマックス選好

`Policy.tabular`（14.5 節）は、各状態行動対（state-action pair）に対して**選好**（preference）と呼ばれる 1 つのパラメータ $\theta_{s,a}$ を保持する。行動全体にわたるソフトマックスにより、状態 s における選好が確率へと変換される。

$$\pi_\theta(a \mid s) = \frac{e^{\theta_{s,a}}}{\sum_{a'} e^{\theta_{s,a'}}}. \tag{14.59}$$

これは状態のインデックスを入力とする 3.1 節のソフトマックス回帰（softmax regression）であり、行動クローニング（behavior cloning）が当てはめたものと同じ方策である。ゼロテーブルは一様方策（一様分布に従う方策）を与える。$Q(s,a)$ とは異なり、選好は収益の推定値ではない。ある状態におけるすべての選好に同じ定数を加えても、方策は変化しない。

14.7.1.2 スコア関数

ソフトマックスは容易に微分できる。$\log \pi_\theta(a \mid s) = \theta_{s,a} - \log \sum_{a''} e^{\theta_{s,a''}}$ であるから、同じ状態における選好 $\theta_{s,a'}$ に関する導関数（derivative）は、任意の行動 a' に対して次式で与えられる。

$$\frac{\partial \log \pi_\theta(a \mid s)}{\partial \theta_{s,a'}} = \mathbf{1}(a' = a) - \pi_\theta(a' \mid s), \tag{14.60}$$

また、他のすべての状態における選好に関する導関数は 0 である。ベクトル $\nabla_\theta \log \pi_\theta(a \mid s)$ は**スコア関数**（score function）と呼ばれる。言葉で言えば、a の対数確率を高めることは、その行動自身の選好を引き上げるとともに、s におけるすべての選好をそれぞれの確率に比例して引き下げることを意味する。e_a で行動 a の指示ベクトルを表すと、s におけるスコアの行は $e_a - \pi_\theta(\cdot \mid s)$ となる。その成分の和は 0 であるため、スコアは方策を変化させない唯一の方向、すなわち s におけるすべての選好に定数を加える方向と直交しており、その方向の成分を持たない。ランダムなテーブルにおいて、この公式を `Policy.log_prob` の自動微分（automatic differentiation）と照らし合わせて確認しよう。

```
rng = np.random.default_rng(0)
check = d2l.Policy.tabular(16, 4)
check.policy.embedding[...] = jnp.asarray(rng.standard_normal((16, 4)),
                                          jnp.float32)   # a generic table
score = nnx.grad(lambda policy: check.log_prob(
    jnp.array([6]), jnp.array([2]), policy).sum())(check.policy)
probs = np.exp(check.log_prob_np(np.repeat(6, 4), np.arange(4)))
hand = np.zeros((16, 4), dtype=np.float32)
hand[6], hand[6, 2] = -probs, 1 - probs[2]
print(bool(np.allclose(score.embedding[...], hand, atol=1e-6)))
```

```
True
```

JAX はモジュールをインプレースで微分するのではなく関数を微分するため、この確認では方策ネットワークを `log_prob` に渡し直し、それに関して `nnx.grad` をとる（`policy_step` と同様である）。

ライブラリの `log_prob` の状態 6・行動 2 における勾配は Equation 14.60 と一致し、テーブルの他のすべての行では 0 になっている。

14.7.2 方策勾配

期待収益の勾配はどのような形をしており、エージェント自身のエピソードから推定できるのだろうか。その答えは状態が 1 つの場合に最も明快に現れ、軌跡（trajectory）に対しても同じ形を保つ。

14.7.2.1 最小の事例

1 つの状態と、選好 θ_1 および θ_2 を持つ 2 つの行動を考えよう。行動 1 は報酬 1 を獲得し、行動 2 は 0 を獲得する。$p = \pi_\theta(1)$ とおく。期待報酬（expected reward）は $J(\theta) = p$ であり、Equation 14.60 より θ_1 に関するその導関数は $p(1-p)$ となる。ここで、p を直接計算することはできず、行動を試すことしかできないとしよう。それでもなお、この導関数は方策自身の行動にわたる平均として書き表せる。各項に $\pi_\theta(a)$ を掛けて割ると、次式が得られる。

$$\begin{aligned}\frac{\partial J}{\partial \theta_1} &= \sum_a r(a)\,\frac{\partial \pi_\theta(a)}{\partial \theta_1} = \sum_a \pi_\theta(a)\, r(a)\,\frac{\partial \log \pi_\theta(a)}{\partial \theta_1} \\ &= \mathbb{E}_{a\sim\pi_\theta}\left[r(a)\,\frac{\partial \log \pi_\theta(a)}{\partial \theta_1}\right].\end{aligned}$$

したがって、サンプリングされた 1 つの行動について報酬にそのスコアを掛けたものは、この導関数の推定値になる。確率 p でサンプルは行動 1 となり、その報酬は 1、スコアは $1-p$ である。確率 $1-p$ で報酬は 0 となり、推定値も 0 になる。平均は $p(1-p)$ であり、導関数そのものに一致する。二次モーメント（second moment）は $p(1-p)^2$ であるため、標準偏差は $(1-p)\sqrt{p(1-p)}$ となる。これと平均との比は次の値になる。

$$\sqrt{\frac{1-p}{p}} \tag{14.61}$$

すなわち、標準偏差は平均にこの比を掛けた大きさになる。報酬をもたらす行動がまれなとき、この推定値は平均としては正しいものの、ほぼすべての単一の抽出では役に立たない。たいていは 0 になり、ごくたまに約 $1/p$ 倍も大きすぎる値になるからである。独立な n 回の抽出を平均すると、この比は $\sqrt{n}$ で割られる。

湖の環境も、学習開始時点では同じ構造を持つ。一様方策がゴールに到達するのはエピソードの約 1.4% であり（14.2 節）、これは滑らない氷でも滑る氷でも変わらない。一様にランダムな指示は一様にランダムな移動になるからである。ゴールを逃したエピソードは何も獲得しない。$p = 0.014$ では上の比は 8.4 になり、16 エピソードからなるバッチに成功が 1 つも含まれない確率は $(1 - 0.014)^{16} \approx 0.80$ となる。つまり、5 バッチのうち 4 バッチは推定値 0 を返す。本節の残りでは、2 つの行動を軌跡に置き換えて同じ計算を実行する。

14.7.2.2 軌跡上の最適化問題

湖の環境におけるエピソードは軌跡 $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T)$ であり、最初の終端状態（terminal state）s_T で終わるため、その長さ T は確率的に決まる。エージェントが各行動を選び、環境が次の各状態を選ぶため、軌跡の確率は方策の因子、遷移カーネル（transition kernel）の因子、および初期分布 μ_0（14.1 節）の積となる。

$$P(\tau;\theta) = \mu_0(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t). \tag{14.62}$$

θ に依存するのは方策の因子だけである。最適化の目的関数は、$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t$ とおいたときのこれらの軌跡にわたる期待収益（期待値）である。

$$J(\theta) = \mathbb{E}_{\tau \sim P(\cdot;\theta)}\big[R(\tau)\big] = \sum_{\tau} R(\tau)\, P(\tau;\theta). \tag{14.63}$$

湖の環境は固定された初期状態を持つため、$J(\theta) = V^{\pi_\theta}(s_0)$ と 14.2 節の記法で書ける。検証器タスクでは、μ_0 は 4 つのプロンプト上の一様分布である。

14.7.2.3 対数微分トリック

最小の事例での手順がそのまま当てはまる。固定された軌跡の収益 $R(\tau)$ は θ に依存せず、その確率だけが依存する。勾配を和の中に入れ、$P(\tau;\theta)$ を掛けて割ると、次式が得られる。

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{\tau} R(\tau)\, \nabla_\theta P(\tau;\theta) = \sum_{\tau} P(\tau;\theta)\, R(\tau)\, \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\ &= \sum_{\tau} P(\tau;\theta)\, R(\tau)\, \nabla_\theta \log P(\tau;\theta), \end{aligned}$$

すなわち、次のようになる。

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\cdot;\theta)}\Big[R(\tau)\, \nabla_\theta \log P(\tau;\theta)\Big]. \tag{14.64}$$

恒等式 $\nabla \log P = \nabla P / P$ は**対数微分トリック**（log-derivative trick）と呼ばれる。これにより、勾配は現在の方策が生成する軌跡にわたる平均へと戻され、サンプリングによって求められるよ

うになる。エピソードは任意のステップ数だけ続きうるため、軌跡にわたる和は無限和である。それでも勾配を和の中に入れる操作は正当化される。ソフトマックス方策のもとではどの状態も数ステップ以内に終端状態へ到達する正の確率を持つため、長いエピソードは指数関数的にまれであり、各ステップが加えるスコアの長さも有界だからである。

14.7.2.4 遷移確率の相殺

勾配の式にはまだ $\log P(\tau;\theta)$ が含まれており、$P(\tau;\theta)$ にはエージェントが知らない遷移カーネルが含まれている。しかし、積 Equation 14.62 の対数は和になり、微分すると方策の項だけが残る。

$$\begin{aligned}\nabla_\theta \log P(\tau;\theta) &= \nabla_\theta \Big[\log \mu_0(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1} \log P(s_{t+1} \mid s_t, a_t) \Big] \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t).\end{aligned}$$

Figure 14.18 に因子の連鎖を示す。初期分布とカーネルは θ に依存しないため、それらの対数の勾配は 0 になる。どの軌跡がサンプリングされるかを決めるのは依然としてカーネルであるが、推定量がカーネルの値を評価することは一切ない。これがこの手法を**モデルフリー**にする理由であり、同時にこの導出が必要とする仮定、すなわちダイナミクスが方策のパラメータに依存してはならないという条件を明確にしている（演習問題 1 ではこの仮定を外す）。

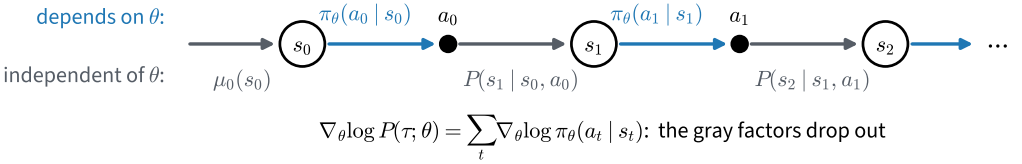


Figure 14.18: 因子の連鎖として表した軌跡の確率 $P(\tau;\theta) = \mu_0(s_0) \prod_t \pi_\theta(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t)$。方策の因子（青、連鎖の上側）だけが θ に依存する。初期分布と遷移カーネル（灰色、下側）は依存しないため、それらの対数の勾配は 0 となり、軌跡のスコアは方策のスコアの和となって P を知らずに計算できる。

14.7.2.5 REINFORCE 推定量

スコアの和を Equation 14.64 に代入し、期待値を現在の方策からサンプリングした n 本の軌跡 $\tau_1, \dots, \tau_n$ にわたる平均に置き換えると、次式が得られる。

$$\hat{u} = \frac{1}{n} \sum_{i=1}^{n} R(\tau_i) \sum_{t=0}^{T_i-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i). \tag{14.65}$$

各軌跡は、自らがとったすべての行動の対数確率を、その収益に比例する量だけ引き上げる。これは冒頭で述べた推測そのものであり、各項の期待値は $\nabla_\theta J(\theta)$ に等しいため、$\hat{u}$ は勾配の不偏推定値（統計的バイアスのない推定値）となる。REINFORCE (Williams 1992) は上昇ステップ

$\theta \leftarrow \theta + \eta\,\hat{u}$ をステップサイズ（step size）η で実行し、続いて更新後の方策から新しいバッチをサンプリングする。湖の環境ではすべての収益が非負であるため、サンプリングされた行動はすべて引き上げられる。サンプリングされなかった行動はスコアの $-\pi_\theta(\cdot \mid s)$ の部分を通じて確率を失い、収益 0 のエピソードは何も変化させない。

Figure 14.19 は、4 行動のソフトマックスでは曲線として描けない連続行動の 1 ステップ問題について、1 回の更新ステップを図示したものである。方策はガウス分布 $\mathcal{N}(a; \mu, 1)$ であり、平均に関するそのスコアは $a - \mu$ となる。1 エピソードは単一の行動 a であるから、その収益 $R(a)$ はその行動の報酬に一致し、更新ステップは $\mu \leftarrow \mu + \eta\,\frac{1}{n}\sum_i R(a_i)(a_i - \mu)$ となる。すなわち、平均は最も多くの報酬を得たサンプル行動の方向へ移動する。12 個のサンプルによって平均は 0.32 へ移動するが、厳密な勾配であれば 0.52 へ移動するところである。

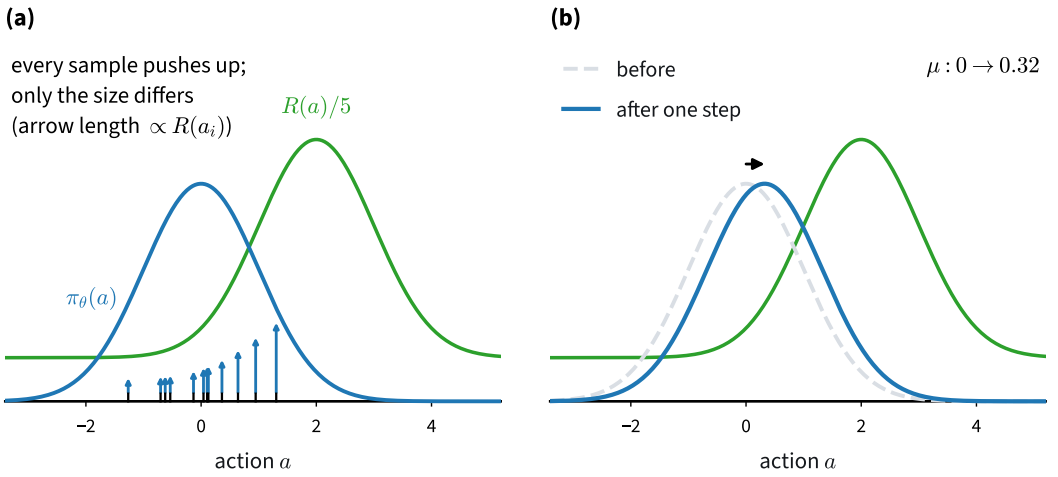


Figure 14.19: 1 ステップ問題における REINFORCE の 1 ステップ。方策は $\pi_\theta(a) = \mathcal{N}(a; \mu, 1)$（ただし $\mu = 0$）、報酬は $R(a) = 0.4 + 2e^{-(a-2)^2/2}$ であるため、期待報酬は $J = 0.92$、厳密な勾配は $\mathrm{d}J/\mathrm{d}\mu = 0.52$ である。(a) サンプリングされた 12 個の行動はそれぞれ、自身の報酬で重み付けされて対数確率を引き上げる。すべての報酬が正であるため、矢印はすべて上を向く。(b) ステップ $\mu \leftarrow \mu + \eta\,\frac{1}{12}\sum_i R(a_i)(a_i - \mu)$（$\eta = 1$）は平均を 0 から 0.32 へ、報酬の高い行動の方向へ移動させる。厳密な勾配であれば 0.52 へ移動していたはずである。

14.7.3 共通例題における REINFORCE

検証器タスクは 1 エピソードあたり 1 ステップしかなく状態遷移も持たないため、あらゆる量を厳密に計算できる。一方、湖の環境は複数ステップにわたるエピソードと、更新処理が決して参照しない遷移カーネルを持つ。

14.7.3.1 検証器タスク

14.1 節の検証器タスクには、4 つの算数のプロンプト x と、それらすべてで共有される 9 個の応答候補 y がある。方策はプロンプトごとに選好の行を 1 つ持つテーブル θ であり、$\pi_\theta(y \mid x)$ は第 x 行のソフトマックスで与えられ、検証器は正解に対して $r(x, y) = 1$、それ以外に対して 0 を与える。軌跡はプロンプトに対する単一の応答であり、その確率は $\mu_0(x)\,\pi_\theta(y \mid x)$ でカーネルは一切含まれない。第 x 行（θ の行）はプロンプト x の期待スコア $J_x(\theta) = \sum_y \pi_\theta(y \mid x)\,r(x, y)$

のみに影響し、状態をプロンプトに置き換えて Equation 14.60 を適用すれば、その厳密な勾配は 9 個の項の和となる。

$$\nabla_{\theta_x} J_x(\theta) = \sum_y \pi_\theta(y \mid x)\, r(x,y)\, (e_y - \pi_\theta(\cdot \mid x)). \tag{14.66}$$

REINFORCE は、この和をサンプリングした K 個の応答にわたる平均に置き換える。`verifier_step` は各プロンプトに対して K 個の応答をサンプリングし、関数 `weighting` でそれらの報酬を重みに変換し、θ の行を $\texttt{lr} = \eta$ に K 個のスコアの重み付き平均を掛けた分だけ動かす。`weighting = lambda r: r` とすれば重みは報酬そのものになり、更新ステップは Equation 14.65 と一致する（14.8 節では別の重み付けを渡す）。判定器は引数 `reward` として渡されるため、同じステップを用いて異なる検証器に対して訓練できる。

```
def verifier_step(task, theta, K, reward, rng, weighting, lr=2.0):
    """One REINFORCE step on a VerifierTask: K responses per prompt, each
    score e_y - pi(.|x) (eq_softmax_score) weighted by weighting(rewards of
    the prompt's K responses), averaged over the K, one ascent step."""
    pi, g = task.policy(theta), np.zeros_like(theta)
    eye = np.eye(len(task.resp))
    for x in range(len(task.prompts)):          # one group of K per prompt
        ys = rng.choice(len(task.resp), K, p=pi[x])
        w = weighting(np.array([reward(x, y) for y in ys]))
        g[x] = (eye[ys] - pi[x]).T @ w / K
    return theta + lr * g
```

完全一致の検証器は正解の応答 y^* にしか報酬を与えないため、プロンプト x に対する推定値は $\frac{c}{K}(e_{y^*} - \pi_\theta(\cdot \mid x))$ となる。ここで c は K 個のうち正解であった応答の個数である。個数 c は成功確率 $p = \pi_\theta(y^* \mid x)$ の二項分布に従うため、この推定値の平均は厳密な勾配である $p\,(e_{y^*} - \pi_\theta(\cdot \mid x))$ に一致し、相対標準偏差は $\sqrt{(1-p)/(Kp)}$、すなわち最小の事例における比を $\sqrt{K}$ で割った値になる。参照テーブルでは回答可能な 2 つのプロンプトにおいて $p \approx 1/8$ であるため、$K = 16$ のグループによる推定値は勾配の長さの約 0.66 倍だけずれると予測される。

次のコードセルでこの予測を確かめよう。参照テーブルにおいて 10,000 個の推定値を抽出し（`lr=1.0` とすると更新後の値から θ を引いた差が推定値になる）、さらに $\eta = 2$ で 1 ステップ進める。

```
task = d2l.VerifierTask()
theta0, pi0, R = task.theta_ref, task.policy(task.theta_ref), task.R
exact = pi0 * R - pi0 * (pi0 * R).sum(axis=1, keepdims=True)  # nine terms
rng = np.random.default_rng(0)
U = np.stack([verifier_step(task, theta0, 16, task.verify, rng, lambda r: r,
                            lr=1.0) - theta0 for _ in range(10_000)])
err = lambda u: np.linalg.norm(u - exact) / np.linalg.norm(exact)
print(f'one K=16 estimate is off by '
      f'{np.sqrt(np.mean([err(u) ** 2 for u in U])):.2f} of the gradient (RMS)')
print('the mean of M of them: ' + ', '.join(
    f'M={M}: {err(U[:M].mean(axis=0)):.3f}' for M in (10, 100, 1000, 10_000)))
print(f'groups without a success, per prompt: '
      f'{np.round((U == 0).all(axis=-1).mean(axis=0), 2)}')
```

```
    after = [task.success(task.policy(theta0 + 2.0 * u)) for u in U]
    print(f'success {task.success(pi0):.3f}; one step with lr = 2 gives '
          f'{task.success(task.policy(theta0 + 2.0 * exact)):.3f} (exact gradient),'
          f' median {np.median(after):.3f} (sampled)')
```

```
one K=16 estimate is off by 0.66 of the gradient (RMS)
the mean of M of them: M=10: 0.126, M=100: 0.061, M=1000: 0.047, M=10000: 0.009
groups without a success, per prompt: [0.12 0.12 1.   1.  ]
success 0.062; one step with lr = 2 gives 0.077 (exact gradient), median 0.077 (sampled)
```

予測どおり、1 回の推定値は勾配の長さの 0.66 倍だけずれている。M 個の推定値の平均の誤差は M が増えるにつれて平均的に $0.66/\sqrt{M}$ に従って下がり続け、下げ止まりは見られない。すなわち、この推定量は不偏である。各推定値は回答可能なプロンプトごとに 1 つの二項カウントに帰着するため、1 回の平均値はそのレートよりかなり小さく出ることもあれば、最大で約 2 倍大きく出ることもある。回答可能な各プロンプトでは、グループの約 12%（$(7/8)^{16} \approx 0.12$）が正解の応答を 1 つも含まず、何も寄与しない。どの候補も検証器を通過しない `8*8` と `9*9` では、すべてのグループが空になる。どの応答も解けないプロンプトに対しては、いくらサンプリングを重ねても勾配は一切生じない。$\eta = 2$ による厳密な 1 ステップは成功率を 0.062 から 0.077 へと引き上げ、サンプリングによるステップも中央値（median）では同程度の改善を示す。このタスクでは重みが付くスコアが正解の応答のものだけであるため、サンプリングによるステップが成功率を下げることは決してない。

14.7.3.2 湖

湖の環境では 1 エピソードが複数ステップからなり、遷移カーネルが関与する。ここではこれまでの節で用いた滑る氷ではなく、すべての指示が意図どおりにエージェントを移動させる滑らない湖を用いる。その理由は、1 つのエピソードが 1 つの行動についてどれだけの情報を持つかにある。一様な初期方策では、どちらのマップでも同じランダムウォークとなり、成功確率も同じ約 1.4% になる。しかし滑る氷の上では、ある指示はそれ自身の方向または直交する 2 方向のいずれかへそれぞれ確率 $1/3$ でエージェントを動かすため、指示が制御できるのは反対方向へは進まないという一点だけである。その結果、一様な初期方策において滑る氷での厳密な勾配の長さは滑らない氷の 3 分の 1 になる一方、1 エピソード推定値のノイズはほぼ変わらないため、同じ精度を得るには 9 倍大きなバッチが必要になる（演習問題 3 でこの係数を導出する）。

方策が改善するにつれて、この差はさらに広がる。滑る氷の上での成功エピソードは 6 ステップではなく数十ステップ続くため、1 回の成功がはるかに多くの行動の間で分け合われ、しかも γ^{T-1} で割り引かれるからである。それでも REINFORCE は滑る氷の上でもゆっくりとながら学習する（演習問題 6）。

このマップに対する Gymnasium のデフォルトの 100 ステップ上限は外しておく。目的関数 Equation 14.63 は完全なエピソード全体にわたって定義されており、途中で打ち切られたエピソードの報酬和はその収益ではないからである（14.1 節）。ソフトマックス方策はすべての行動に正の確率を与えるため、このマップではすべてのエピソードが確率 1 で終了し、10,000 ステップという上限はここでの実行が生み出すどのエピソードよりもはるかに大きい安全策にすぎない。

```
    gamma = 0.95
    env = gym.make('FrozenLake-v1', is_slippery=False, max_episode_steps=10_000)
```

`Batch` は複数エピソードの遷移をフラットな配列に格納し、あわせて各エピソードの最終ステップの 1 つ先のインデックスを保持する。`episode_returns` は各エピソードに対して $R(\tau)$ を計算し、`per_step` はエピソードごとに 1 つの値をそのエピソードの全ステップにわたって繰り返すこ

とで、収益を各行動の重みへと変換する。このコンテナは次の状態と、**終端**（terminated）のみを表し**打ち切り**（truncated）は決して含めないフラグ `term` も記録する。14.1 節の規則により、途中で打ち切られたエピソードは終端状態に到達していないからである。ここでの湖のエピソードはすべて終端に達するため、このフラグが意味を持つのはエピソードが打ち切られるようになってからである。

```
class Batch:
    """A flat batch of transitions, plus the episode boundaries."""
    def __init__(self, obs, act, rew, next_obs, term, ep_ends):
        self.obs, self.act, self.rew = obs, act, rew
        self.next_obs, self.term = next_obs, term
        self.ep_ends = ep_ends    # one past the last step of each episode

    def __len__(self):
        return len(self.rew)

    def episodes(self):
        """Yield one slice per episode."""
        start = 0
        for end in self.ep_ends:
            yield slice(start, end)
            start = end

    def episode_returns(self, gamma=1.0):
        """R(tau), the discounted return, one number per episode."""
        return np.array([(gamma ** np.arange(ep.stop - ep.start)
                          * self.rew[ep]).sum() for ep in self.episodes()])

    def per_step(self, values):
        """Repeat one value per episode over that episode's steps."""
        return np.repeat(values, np.diff(self.ep_ends, prepend=0))
```

`rollout` はバッチにデータを詰める関数である。関数 `policy(obs, rng) -> action`（14.2 節の `evaluate` と同じ行動プロトコル）として書かれた任意の方策で完全なエピソードを実行し、各ステップの状態、行動、報酬、次の状態、および終端フラグを追加していく。エージェントの乱数性はすべて NumPy の乱数生成器 `rng` に由来するため、シードを固定すればサンプリングされるすべてのエピソードが確定する。

```
def rollout(env, policy, num_episodes, rng):
    """Collect complete episodes from `policy(obs, rng) -> action` as a
    Batch. `term` records `terminated`, never `truncated`: a time-limit cut
    is not a terminal state. All sampling runs through the numpy `rng`."""
    cols, ep_ends = [[] for _ in range(5)], []
    for _ in range(num_episodes):
        obs, done = env.reset()[0], False
        while not done:
            act = policy(obs, rng)
            next_obs, reward, terminated, truncated, _ = env.step(act)
            done = terminated or truncated
            for col, val in zip(cols, (obs, act, reward, next_obs,
                                       float(terminated))):
                col.append(val)
```

```
            obs = next_obs
        ep_ends.append(len(cols[0]))
    obs, act, rew, next_obs, term = (np.asarray(c) for c in cols)
    return Batch(obs, act, rew.astype(np.float32), next_obs,
                 term.astype(np.float32), np.asarray(ep_ends))
```

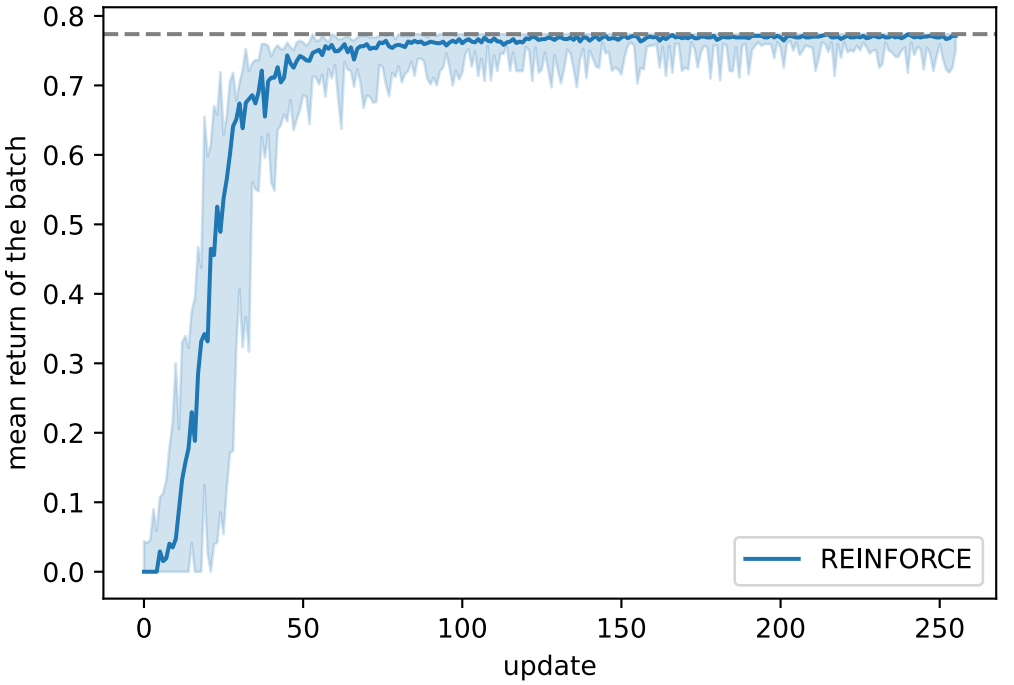
訓練ループでは 14.5 節の `policy_step` を再利用する。これは $\frac{1}{N}\sum_t w_t \log \pi_\theta(a_t \mid s_t)$ に対して、バッチの N ステップにわたりオプティマイザを 1 ステップ進める関数である。行動クローニングでは $w_t = 1$ を渡したが、REINFORCE ではステップ t を含むエピソードの収益を渡す。内部的には `policy_step` はこの和に負号を付けた**代理損失**（surrogate loss）を最小化するが、その損失の値自体は方策の性能について何も語らない。重要なのは現在の θ におけるその勾配、すなわち推定値だけである。

Equation 14.65 とは細部が 2 点異なる。第 1 に、`policy_step` が割るのはステップ数 N でありエピソード数 n ではないため、推定値はバッチの平均エピソード長（バッチごとに変動する係数）で割られることになる。第 2 に、オプティマイザは `Policy.tabular` の学習率（learning rate）0.1 を持つ Adam であり、素の更新ステップ $\theta \leftarrow \theta + \eta\hat{u}$ ではない。素の勾配上昇法は、ある行動の確率が小さいところでは極端に遅くなる。Equation 14.60 により、その選好の勾配がその確率を因数として持つからである。これに対し、Adam は各座標をほぼ学習率の大きさだけ動かす（演習問題 6 で両者を比較する）。以下のループは新しいバッチを収集し、各ステップをそのエピソードの収益で重み付けして 1 ステップ更新し、バッチの平均収益とステップ数を返す。

```
def train_reinforce_tabular(ac, env, gamma, seed, num_updates=256,
                            batch_episodes=16):
    """REINFORCE: a fresh batch from the current policy, every step of a
    trajectory weighted by that trajectory's return, one ascent step.
    Yields the batch's mean return and its number of steps."""
    rng = np.random.default_rng(seed)      # one stream for all sampling
    env.reset(seed=seed)
    for _ in range(num_updates):
        batch = rollout(env, ac.act, batch_episodes, rng)
        R = batch.episode_returns(gamma)
        d2l.policy_step(ac, batch.obs, batch.act, batch.per_step(R))
        yield float(R.mean()), len(batch)
```

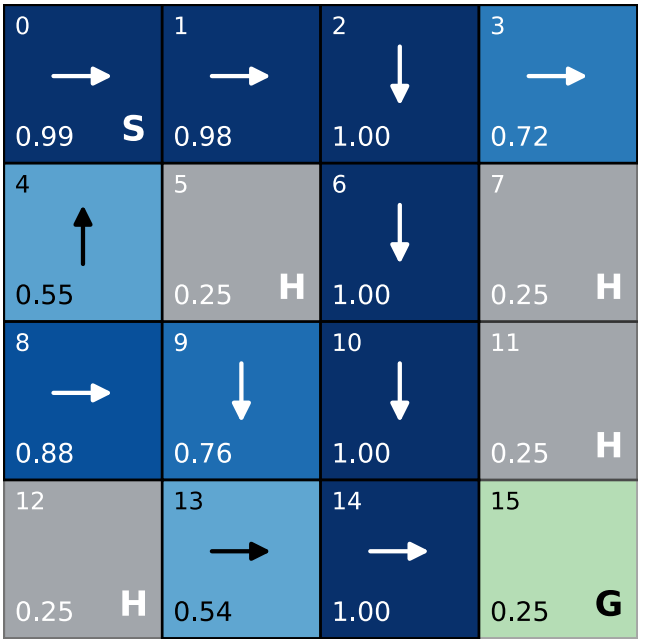
それぞれ異なるシードを持つ 12 個のエージェントを一様方策から訓練し、各訓練バッチの平均収益（$J(\theta)$ の推定値）を、6 ステップ経路の上限である $\gamma^5 \approx 0.774$ とともにプロットする。

```
agents = [d2l.Policy.tabular(16, 4) for _ in range(12)]  # uniform policies
runs = np.array([list(train_reinforce_tabular(ac, env, gamma, seed))
                 for seed, ac in enumerate(agents)])     # (seed, update, 2)
curves, steps = runs[..., 0], runs[..., 1].astype(int)
d2l.plot_curves({'REINFORCE': curves}, xlabel='update',
                ylabel='mean return of the batch', reference=gamma ** 5)
```



図は 12 回の実行の中央値と、最も遅い実行から最も速い実行までの帯を示している。最小の事例から予測されたとおり、初期のバッチの大半は成功を 1 つも含まず推定値が 0 になるため、中央値は平坦に始まる。ひとたびバッチ内で成功が出ると、そのときの行動が起こりやすくなり、成功の頻度が増して平均収益は急速に上昇する。すべての実行は数十回の更新のうちに破線の上限へ近づくが、方策が依然として確率的であり時折最短経路から外れるため、上限よりわずかに低い水準にとどまる。次のコードセルでは、1 番目のエージェントが学習した方策について、各マスで最も確率の高い行動とその確率を表示する。

```
probs = agents[0].probs(16, 4)
d2l.show_grid(env.unwrapped.desc, probs.max(axis=1), probs.argmax(axis=1))
```



方策はゴールへ至る 3 本の最短経路のうち、たまたま最初に成功したエピソードが通った 1 本に集中している。穴とゴールでは一度も行動がとられないため、一様分布のままである。選ばれた経路から外れたマスには、初期の成功エピソードがそこを通過した際に獲得した選好がほぼそのまま残っている。それらのマスが変化するのはエピソードがたまたま迷い込んだときだけだが、経路が固まるにつれてそれはまれになる。訪問と訪問の間は Adam のモメンタム（momentum）が直前の変化を数十回の更新にわたって引き継ぐが、その後は動かなくなる。このループのどこにも遷移カーネルを評価する処理はない。`rollout` はシミュレータがサンプリングした次の状態を記録しただけであり、更新処理が読み取ったのは状態・行動・収益のみである。これこそが 14.7.2.4 節の相殺が実際に働いている姿であり、カーネルを持たない検証器タスクでは見ることのできなかった現象である。

14.7.4 方策勾配定理

REINFORCE は、ある行動がとられるより前に届いた報酬も含め、軌跡全体の収益によってすべての行動のスコアを重み付けする。しかし、行動がすでに獲得された過去の報酬に影響を与えることはできないため、それらの報酬は平均的には何も寄与しないはずである。方策勾配定理（policy gradient theorem）(Sutton et al. 2000) はこの事実を厳密に定式化し、勾配を軌跡全体ではなく方策が訪問する状態にわたる平均として書き直す。この定理を述べるため、エピソード終了後もマルコフ連鎖は報酬 0 で終端状態にとどまり続けるものとし、s_t がすべての t に対して定義されるようにする。$d_t^{\pi_\theta}(s) = \Pr(s_t = s)$ を s_t が π_θ のもとで従う分布とし、さらに次式をおく。

$$d_\gamma^{\pi_\theta}(s) = (1-\gamma)\sum_{t \geq 0} \gamma^t \, d_t^{\pi_\theta}(s) \tag{14.67}$$

これは時間に関するその割引平均であり、**割引状態占有分布**（discounted state-occupancy distribution）と呼ばれる。係数 $1-\gamma$ によって重み γ^t の総和が 1 になるため、$d_\gamma^{\pi_\theta}$ は状態上の確率分布（probability distribution）となる。

14.7.4.1 主張と証明

命題（方策勾配定理）。 **目的関数** $J(\theta) = \mathbb{E}_{s_0 \sim \mu_0}[V^{\pi_\theta}(s_0)]$ **に対して、次式が成り立つ。**

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim P(\cdot;\theta)}\Big[\sum_{t \geq 0} \gamma^t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t) \, Q^{\pi_\theta}(s_t, a_t)\Big] \\ &= \frac{1}{1-\gamma} \, \mathbb{E}_{s \sim d_\gamma^{\pi_\theta},\, a \sim \pi_\theta(\cdot \mid s)}\Big[\nabla_\theta \log \pi_\theta(a \mid s) \, Q^{\pi_\theta}(s, a)\Big] \end{aligned} \tag{14.68}$$

証明。 Equation 14.64 と相殺により、$\nabla_\theta J = \mathbb{E}\big[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, R(\tau)\big]$ である。各 t において、収益を t より前の報酬と t 以降の報酬に分割する。t より前の報酬はエージェントが s_t に立った時点で確定している一方、a_t はこれから抽出される段階にあり、a_t のスコアは s_t が与えられたもとで平均が 0 になる。実際、$\sum_a \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) = \sum_a \nabla_\theta \pi_\theta(a \mid s) = \nabla_\theta 1 = 0$ である。したがって、過去の報酬の寄与は 0 となる。t 以降の報酬については (s_t, a_t) で条件付けると、マルコフ性（Markov property）によりそれらの割引和の期待値は $\gamma^t Q^{\pi_\theta}(s_t, a_t)$ となる。これより第 1 の形式が得られる。第 2 の形式を得るには、t にわたる和を訪問した状態ごとにまとめればよい。任意の関数 f に対して $\mathbb{E}\big[\sum_t \gamma^t f(s_t, a_t)\big] = \sum_s \sum_t \gamma^t d_t^{\pi_\theta}(s) \, \mathbb{E}_{a \sim \pi_\theta(\cdot \mid s)} f(s, a)$ が成り立つため、ここに $d_\gamma^{\pi_\theta}$ の定義を代入すればよい。終端状態では $Q^{\pi_\theta} = 0$ であるため、エピソード終了後のステップは何も寄与しない。■

14.7.4.2 帰結

この定理からは 3 つの帰結が導かれる。第 1 に、時刻 t におけるスコアが必要とするのは t 以降の報酬だけである。証明の中で過去の報酬は平均 0 のノイズとして取り除かれたため、それらを落とした推定量は同じ平均を持ち、通常はノイズも小さくなる。第 2 に、勾配は方策が訪問する状態にわたる平均であり、どれだけ頻繁に、かつどれだけ早い時点で訪問するかによって重み付けされる。そのため、方策がめったに到達しない状態はわずかな勾配しか受け取らず、一度も訪問しなかったバッチではその行の推定値が厳密に 0 になる。先ほど学習された経路から外れたマスがほとんど変化しなかった理由はここにある。第 3 に、多くの実装は第 1 の形式から因子 γ^t を落とす。これは $d_\gamma^{\pi_\theta}$ を割引なしの状態訪問に置き換え、遅い時点で訪問される状態の相対的な重みを変えるものであり、一般にはいかなる目的関数の勾配にもならない更新をもたらす (Nota and Thomas 2020)。演習問題 7 でその違いを詳しく調べる。

14.7.5 コストと限界

14.7.5.1 オンポリシーデータの要件

推定量 Equation 14.65 は、**現在の**方策から抽出された軌跡を平均する。1 回更新するとパラメータが変化するため、すでに収集した軌跡はもはや $P(\tau;\theta)$ に従わない。この要件を持つ手法は**オンポリシー**と呼ばれ、更新のたびに新しいバッチを収集する。これに対し、Q 学習は**オフポリシー**である。その更新は、任意の行動方策（behavior policy）が生成した遷移から貪欲方策（greedy policy）について学習する（14.3 節）。REINFORCE が使用するデータ量を環境のステップ数で数えてみよう。

```
hit = (curves >= 0.7).argmax(axis=1) + 1   # batches until the mean is 0.7
to_hit = np.array([s[:k].sum() for s, k in zip(steps, hit)])
for name, x in [('batches until a batch mean reaches 0.7', hit),
                ('environment steps until then', to_hit),
                ('environment steps of the full run', steps.sum(axis=1))]:
```

```
      print(f'{name:>38}: median {np.median(x):>6,.0f}, '
            f'range {x.min():,} to {x.max():,}')
```

```
batches until a batch mean reaches 0.7: median     32, range 25 to 49
          environment steps until then: median  4,313, range 3,210 to 6,191
     environment steps of the full run: median 27,049, range 26,058 to 28,383
```

バッチ平均が初めて 0.7（上限の約 90%）に達するのは 25〜50 回の更新後であり、環境のステップ数にして 3,000〜7,000 ステップに相当する。256 回の更新を行う実行全体では約 27,000 ステップを消費するが、これは方策の改善が止まった後も各更新が新しいバッチを収集し続けるためである。同じマップにおいて、$\alpha = 0.9$ の表形式の Q 学習はほぼすべての実行でゴールに到達する貪欲テーブルを発見し、実行全体に要したのは約 3,500 ステップであった（14.3 節）。1 つのバッチを複数回の更新に再利用するのが自然な対策に思えるが、補正なしでは理論的に破綻する（演習問題 4）。

14.7.5.2 不偏性と分散

湖の環境もモデルが既知であるため、検証器タスクと同様に厳密な勾配と照らし合わせて検証できる。14.2 節では、方策の価値は連立一次方程式（linear system）$V^\pi = r_\pi + \gamma P_\pi V^\pi$ の解として求まった。ここで $P_\pi(s' \mid s) = \sum_a \pi(a \mid s) P(s' \mid s, a)$ および $r_\pi(s) = \sum_a \pi(a \mid s)\, r(s, a)$ である。したがって、目的関数は次式で表される。

$$J(\theta) = V^{\pi_\theta}(s_0) = \left[\left(I - \gamma P_{\pi_\theta} \right)^{-1} r_{\pi_\theta} \right]_{s_0}. \tag{14.69}$$

P_{π_θ} と r_{π_θ} のすべての成分は θ の滑らかな関数であるため、連立一次方程式の求解を通した自動微分によって厳密な $\nabla_\theta J$ が得られる。`TabularMDP.from_gym` では穴とゴールが報酬 0 の吸収状態（absorbing state）になっているため、この求解にはエピソードの終端も織り込まれている。シード 3 の実行において上昇が始まっている 16 回更新後の方策を `probe` として固定しよう。

```
probe = d2l.Policy.tabular(16, 4)
list(train_reinforce_tabular(probe, env, gamma, 3, num_updates=16))  # train
mdp = d2l.TabularMDP.from_gym(env, gamma)
```

`policy_objective` は選好テーブル、カーネル `P[s, a, s']`、および報酬 `r[s, a]` から $J(\theta)$ を Equation 14.69 によって計算し、フレームワークの `grad` が連立一次方程式の求解を通してそれを微分する。このプローブにおける J とその厳密な勾配を計算しよう。

```
def policy_objective(theta, P, r, gamma):
    """J(theta) = V(s0) of the softmax policy, a differentiable linear solve."""
    pi = jax.nn.softmax(theta, -1)
    P_pi = jnp.einsum('sa,sat->st', pi, P)
    return jnp.linalg.solve(jnp.eye(len(P)) - gamma * P_pi,
                            (pi * r).sum(-1))[0]

P, r = jnp.asarray(mdp.P, jnp.float32), jnp.asarray(mdp.r, jnp.float32)
theta = probe.policy.embedding[...]
g_exact = np.asarray(jax.grad(policy_objective)(theta, P, r, gamma))
print(f'J(theta) after 16 updates = '
      f'{float(policy_objective(theta, P, r, gamma)):.3f}, '
```

```
        f'against the ceiling gamma^5 = {gamma ** 5:.3f}')
```

```
J(theta) after 16 updates = 0.313, against the ceiling gamma^5 = 0.774
```

プローブは一様方策と上限のちょうど中間あたりに位置している。ここで θ を固定し、推定量を何度も抽出してみよう。`tabular_score` はバッチの重み付きスコア $w_t\left(e_{a_t} - \pi_\theta(\cdot \mid s_t)\right)$ を Equation 14.60 に従ってそれぞれの状態の行へ足し合わせる。`reinforce_estimate` は `verifier_step` と同様に、これを用いて Equation 14.65 の 1 回の抽出を計算する。すなわち、各ステップをそのエピソードの収益で重み付けし、和を n で割る。各バッチサイズ（batch size）n について 50 個の推定値を抽出し、$\|\nabla_\theta J\|$ に対する相対値として 3 つの数値を出力する。1 回の抽出の二乗平均平方根誤差（root-mean-square error）、50 個の平均の誤差、そして不偏推定量（unbiased estimator）がその平均に残すはずの誤差（1 つ目の数値を $\sqrt{50}$ で割った値）である。

```
def tabular_score(obs, act, w, pi):
    """sum_t w_t (e_{a_t} - pi(.|s_t)), each term added into row s_t: the
    weighted softmax score (eq_softmax_score) of a batch, the shape of pi."""
    u = np.zeros(pi.shape)
    np.add.at(u, obs, w[:, None] * (np.eye(pi.shape[1])[act] - pi[obs]))
    return u

def reinforce_estimate(env, probe, gamma, n, rng):
    """One draw of eq_reinforce at the frozen tabular policy `probe`: each
    step adds its trajectory's return times its score to row s; the sum is
    divided by n."""
    pi = probe.probs(env.observation_space.n, env.action_space.n)
    batch = rollout(env, probe.act, n, rng)
    w = batch.per_step(batch.episode_returns(gamma))
    return tabular_score(batch.obs, batch.act, w, pi).ravel() / n

rng, g = np.random.default_rng(4), g_exact.ravel()
for n in [4, 16, 64]:
    U = np.stack([reinforce_estimate(env, probe, gamma, n, rng)
                  for _ in range(50)])
    zero = int((U == 0).all(axis=1).sum())
    one = np.sqrt(((U - g) ** 2).sum(axis=1).mean()) / np.linalg.norm(g)
    mean = np.linalg.norm(U.mean(axis=0) - g) / np.linalg.norm(g)
    print(f'n={n:>2}: {zero:>2}/50 zero; one draw off by {one:.2f}, the mean '
          f'of 50 by {mean:.2f} (if unbiased: {one / np.sqrt(50):.2f})')
```

```
n= 4:  3/50 zero; one draw off by 3.85, the mean of 50 by 0.48 (if unbiased: 0.54)
n=16:  0/50 zero; one draw off by 1.78, the mean of 50 by 0.24 (if unbiased: 0.25)
n=64:  0/50 zero; one draw off by 0.96, the mean of 50 by 0.10 (if unbiased: 0.14)
```

$n = 4$ では、単一の抽出は推定対象である勾配の長さの 3 倍以上も外れており、バッチ内のどのエピソードもゴールに到達しなかったために 0 になる抽出も現れうる。n を 4 倍にするたびに誤差は半分になり、最小の事例で見た $1/\sqrt{n}$ の法則に従っている。どの n においても 50 回の抽出の平均の誤差は不偏推定量から予測される値と同程度であり、このテストでは統計的バイアスは検出されない。演習問題 5 では抽出回数をさらに大幅に増やし、小さなバイアスでも現れるようにして検証する。

14.7.5.3 プラトーと収束

目的関数は θ に関して凹関数ではない。すでに最小の事例においても、$J = p$ は $\theta_1 - \theta_2$ のロジスティック関数であり、両端で平坦になる。非凹関数に対する勾配上昇法は、一般には性能の低い停留点（stationary point）で止まりうる。しかし、表形式のソフトマックス方策ではそうならない。準最適な停留点は存在せず、初期分布がすべての状態に正の確率を与えるならば、十分に小さいステップサイズによる厳密な勾配上昇法は最適方策（optimal policy）へと収束（convergence）する (A. Agarwal et al. 2021; Mei et al. 2020)。それでも収束が遅くなることはある。ある状態でより良い行動の確率が小さいと、その選好の勾配がその確率を因数として持つため、勾配上昇はプラトーの上を這うようにしか進まず、厳密なソフトマックス勾配上昇法が状態数に対して指数関数的なステップ数を要するマルコフ決定過程（MDP）も存在する (Li et al. 2021)。

固定ステップサイズによるサンプル更新は、2 つ目のリスクをもたらす。たまたま運の良かった少数のバッチによってある状態の方策が悪い行動にほぼ決定論的に偏ってしまうと、その選択を覆すための勾配が極端に小さくなり、実行が長いあいだその行動に固着してしまうのである (Mei et al. 2021)。滑らない湖では 12 回の実行すべてが上限から数パーセント以内の範囲に収まったが、滑る氷の上では同じ手順でもシードごとに異なる水準に落ち着き、最適値を大きく下回るものも現れる。演習問題 6 でその理由を突き止める。

これで、モデルもエキスパートも価値テーブルも使わず、方策自身のエピソードとその報酬だけから方策を改善できるようになった。しかも検証器タスクにおいては、同じ更新式がすでに言語モデルの事後学習（post-training）と同じ形状をしている。すなわち、応答が軌跡であり、報酬はその最後に届く。この推定値は平均的には正しいが、プローブ方策において 1 バッチあたり 4 エピソードとした場合、単一の抽出は勾配の長さの 3 倍以上も外れ、失敗したエピソードは何も寄与しない。続く 14.8 節では、平均を動かさずにそのノイズをどれだけ取り除けるかを問う。

14.7.6 まとめ

方策勾配法は、パラメータ化された方策 π_θ の期待収益 $J(\theta)$ に対する勾配上昇法によって、方策を直接改善する。対数微分トリックにより $\nabla_\theta J$ は方策自身の軌跡にわたる「収益と軌跡のスコアの積」の平均として書け、さらに遷移カーネルがスコアから消えるため、モデルなしでサンプリングされたエピソードから勾配を推定できる。REINFORCE はすべての行動のスコアをそのエピソードの収益で重み付けする手法であり、行動クローニングの更新における単位重みを収益に置き換えたものに相当する。検証器タスクと滑らない湖において、この推定値は平均的には厳密な勾配と一致するが、湖では小さなバッチの推定値は勾配の長さの数倍も外れ、誤差は $1/\sqrt{n}$ の速さでしか縮まらない。方策勾配定理は、行動より前に得られた報酬がノイズとしてしか寄与しないことを示している。また、各更新には現在の方策から収集した新しいバッチが必要である。

14.7.7 演習問題

1. **パラメータ依存のダイナミクス**。14.7.2.4 節の導出では、遷移確率が θ に依存しないと仮定している。しかし推薦システムにおいて、あるアイテムを頻繁に目にするユーザーほどそれをクリックしなくなる場合、この仮定は破られる。
    1. ダイナミクスが $P_\theta(s_{t+1} \mid s_t, a_t)$ であるとする。$\nabla_\theta \log P(\tau;\theta)$ の導出をやり直し、もはや消えなくなる項を示せ。その項を計算するには、エージェントは何を知っている必要があるか。
    2. 1 ステップのモデルにおいて、推薦システムはアイテム 1 を確率 $\pi_\theta(1) = \sigma(\theta)$ で提示し、それ以外ではアイテム 0 を提示するものとする（ここで σ はロジスティック関数である）。提示されたアイテム 1 は確率 $1 - \sigma(\theta)$ でクリックされ、アイテム 0 は決してクリックされず、クリックされると報酬 1 が得られる。$J(\theta)$ と $\mathrm{d}J/\mathrm{d}\theta$ を計算せよ。この導関数を、上の項を落とした REINFORCE 推定値 Equation 14.65 の期待値

と比較せよ。素の勾配上昇法は、これら 2 つのそれぞれを用いたときどこに収束するか。

2. [code] **疎な報酬**。本文の最小の事例では、1 つの状態と 2 つの行動があり、行動 1 の報酬が 1、行動 2 の報酬が 0、かつ $p = \pi_\theta(1)$ であり、1 サンプル推定値は平均の $\sqrt{(1-p)/p}$ 倍の標準偏差を持つ。湖の環境においてこれに相当する量は、一様方策のエピソードがゴールに到達する確率 $p_0 \approx 0.014$ であり、これは 14.2.4 節において $\gamma = 1$ での方策評価によって計算されている。

   1. 成功を 1 つも含まないバッチでは推定値が 0 となり、このとき `policy_step` はゼロテーブルを変化させないため、最初の成功が出るまで方策は一様なまま保たれる。最初の非ゼロ推定値が出るまでの更新回数（その回を含む）はどの分布に従い、16 エピソードのバッチではその平均はいくらか。12 回の訓練実行のそれぞれについてこの回数を記録し、理論予測および訓練曲線の平坦な立ち上がり部分と比較せよ。
   2. 独立な n 個の 1 サンプル推定値の平均が、その平均の $\sqrt{(1-p)/(np)}$ 倍の標準偏差を持つことを示せ。$p = p_0$ において標準偏差を平均と等しくするには、1 バッチあたり何エピソード必要か。訓練によって成功確率が上昇するにつれて、この答えはどのように変化するか。

3. [code] **テーブルにおける方策勾配定理**。●表形式のソフトマックス方策に対して、$\rho_\gamma(s) = \sum_{t \geq 0} \gamma^t d_t^{\pi_\theta}(s)$ で s への割引訪問回数を表し、$d_\gamma^{\pi_\theta} = (1-\gamma)\rho_\gamma$ が Equation 14.68 において成り立つようにする。

   1. Equation 14.68 の状態占有形式と Equation 14.60 を組み合わせ、次式が成り立つことを示せ。
   $$\frac{\partial J}{\partial \theta_{s,a'}} = \rho_\gamma(s)\, \pi_\theta(a' \mid s)\big(Q^{\pi_\theta}(s,a') - V^{\pi_\theta}(s)\big).$$
   2. 演習問題 1b（14.2 節）より、$\rho_\gamma^\top$ は $(I - \gamma P_{\pi_\theta})^{-1}$ のうち s_0 に対応する行である。ρ_γ、V^{π_θ}、Q^{π_θ} を `probe` のパラメータにおいて `mdp.P` と `mdp.r` を用いた連立一次方程式の求解により計算し、上の式を組み立てて結果を `g_exact` と比較せよ。
   3. 滑る氷の上では、指示 a はエージェントを方向 a またはそれに直交する 2 方向のいずれかへそれぞれ確率 $1/3$ で移動させる。一様方策のもとでは両方のマップが同じマルコフ連鎖を誘導し、したがって同じ ρ_γ と V を持つこと、さらに滑る氷での $Q(s,a)$ が $Q_{\text{slip}}(s,a) - V(s) = -\frac{1}{3}\big(Q_{\text{calm}}(s,\bar{a}) - V(s)\big)$（ただし $\bar{a}$ は a と反対向きの指示）を満たすことを示せ。これより、$\theta = 0$ における厳密な勾配の長さは滑る氷の上では 3 分の 1 になることを結論づけ、両方の遷移テーブル上で `policy_objective` を用いて確認せよ。

4. [code] **バッチの再利用**。●`train_reinforce_tabular` を修正し、何の補正も行わずに各バッチ上で `policy_step` を k 回（$k \in \{1, 2, 5, 20\}$）呼び出すようにせよ。そのバッチを収集したときのパラメータを θ_{old} と書く。

   1. 滑らない湖において、バッチの平均収益と、k 回目のステップ後のバッチ全体にわたる比 $\pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ の平均および最大値を記録せよ（両方策の評価には `log_prob_np` を用いる）。各 k において、バッチの平均収益が 0.7 に達するまでに環境のステップ数はどれだけ必要か。また、上の比はどれほど大きくなるか。
   2. k が大きくなるにつれて、更新ステップは固定されたバッチ上の代理目的関数 $\sum_t w_t \log \pi_\theta(a_t \mid s_t)$ の最大点に近づく（ここで w_t はステップ t を含むエピソードの収益である）。これを最大化する表形式方策を閉形式で求め、それを用いて小問 a の結果を説明せよ。

3. 滑るマップ（`is_slippery=True`）において、$k = 1$ と $k = 20$ を最終的な $J(\theta)$（滑る氷の遷移テーブルから Equation 14.69 で計算したもの）によって比較せよ。各実行は数秒で終わる。小問 b の閉形式を用いて、滑らない湖との違いを説明せよ。

5. [code] **不偏性の検証。**●本文の 50 回の抽出によるテストでは、50 回の抽出のノイズより小さい統計的バイアスを見抜くことはできない。より鋭敏なテストは M 個の推定値を平均することである。不偏推定量であれば M が増えるにつれて平均の誤差は縮み続けるが、バイアスのある推定量ではバイアスの大きさで下げ止まる。

   1. `probe` のパラメータにおいて、$M = 25{,}600$ 個の推定値を `reinforce_estimate`$(n = 1)$ で抽出せよ。$\|$`g_exact`$\|$ に対する累積平均の相対誤差を、両対数軸上で M に対してプロットせよ。曲線の傾きを説明せよ。
   2. `max_episode_steps` を 20 および 10 に下げて再構築した環境を `reinforce_estimate` に渡し、`rollout` が一部のエピソードを打ち切りによって終了させる状態で同じ測定を繰り返せ。それぞれの曲線はどの水準で下げ止まるか。打ち切られた推定値は、どの目的関数の勾配に対する不偏推定値になっているか。

6. **ソフトマックスのプラトー。**●1 つの状態と 2 つの行動があり、報酬が 1 と 0、初期選好が $\theta = (0, L)$（ただし $L > 0$）であって、より良い行動の初期確率が $1/(1 + e^{L})$ となる場合を考える。これは 14.7.5.3 節のプラトーを最小の形で取り出したものである。

   1. J が θ に $x = \theta_1 - \theta_2$ を通じてのみ依存し、$J = \sigma(x)$（σ はロジスティック関数）と書けることを示せ。これより、J が凹関数でないことを結論づけよ。
   2. ステップサイズ η による素の勾配上昇法の 1 ステップが、x を $2\eta\,\sigma(x)(1-\sigma(x))$ だけ増加させることを示せ。これを微分方程式として扱い、$\pi_\theta(1) = 1/2$ に $x = -L$ から到達するまでのステップ数が約 $e^{L}/(2\eta)$（ただし L は十分大きいとする）になることを示せ。
   3. 同じ厳密な勾配を Adam に入力すると、勾配の符号が一定であるあいだ、各座標は 1 ステップあたりほぼ学習率 η だけ動く。必要なステップ数を見積もり、小問 b の答えと $L = 10$、$\eta = 0.1$ において比較せよ。この比較は、`Policy.tabular` で Adam を用いることについて何を示唆しているか。
   4. [code] 滑るマップ上で複数のシードを用いて `train_reinforce_tabular` を 1,024 回更新し、最終的な $J(\theta)$ を Equation 14.69 で計算せよ。$V^{*}(s_0)$ を大きく下回って終了した実行の最終方策は、どの状態で、どの行動に対してほぼ決定論的になっているか。そこでの $\|\nabla_\theta J\|$ と一様な初期方策での値を計算せよ。最終的な θ は停留点か。その点および $\theta = 0$ から厳密な勾配上昇法を実行し、観察される挙動を小問 b および c と関連づけよ。

7. [code] **省略された割引因子。**●●多くの実装は、因子 γ^{t} を Equation 14.68 の軌跡形式から省略しており、これにより次の更新ベクトル場が得られる。

$$F(\theta) = \mathbb{E}_{\tau \sim P(\cdot;\theta)}\Big[\sum_{t \ge 0} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, Q^{\pi_\theta}(s_t, a_t)\Big]. \tag{14.70}$$

   Nota and Thomas (2020) は、F がいかなる関数の勾配にもならない場合があることを証明し、さらに F に沿った厳密な更新が方策を最悪の方策へと向かわせるマルコフ決定過程（MDP）を構成した。以下の各小問では、まず 1 つ目の事実を FrozenLake 上で確認し、続いて 2 つ目の出発点として F に沿ったステップが J を引き下げる小さな MDP を構成する。

   1. 演習問題 3a にならい、次式を示せ。

$$F_{s,a'}(\theta) = \rho_1(s)\,\pi_\theta(a' \mid s)\big(Q^{\pi_\theta}(s, a') - V^{\pi_\theta}(s)\big), \tag{14.71}$$

ここで $\rho_1(s) = \sum_{t \geq 0} \Pr(s_t = s,\, t < T)$ は、エピソードが終了するまでの s への期待訪問回数である。方策の遷移行列のうち終端状態に属する行を 0 に設定した連立一次方程式を解くことで、ρ_1 を計算せよ。

2. 連続微分可能な勾配場のヤコビ行列はヘッセ行列であるため対称になる。自動微分によってゼロテーブルおよび probe のパラメータにおけるヤコビ行列 DF を計算し、フロベニウスノルムによる $\|DF - DF^{\top}\| / \|DF\|$ を求めよ。対照実験として、$\nabla_\theta J$ に対しても同じ確認を実行せよ。

3. 同じ 2 点において、F と $\nabla_\theta J$ のなす角のコサインを測定せよ。それにもかかわらず実際の実装が F を受け入れている実用上の理由を 1 つ挙げよ。

4. 表形式のソフトマックス方策では $F(\theta)^{\top} \nabla_\theta J(\theta) \geq 0$ がすべての θ で成り立ち、したがって FrozenLake 上では F に沿った微小ステップが決して J を下げないことを示せ。次に、順次に訪問される系列上の 2 つの状態の方策を単一のパラメータが制御するような小さなエピソード型 MDP と、ある θ で $F(\theta)\, \mathrm{d}J/\mathrm{d}\theta < 0$ となる組を構成せよ。

14.8 ベースラインと分散低減

前節では、サンプリングされた各行動の対数確率を、その軌跡（trajectory）の収益（return）に比例して引き上げることで方策（policy）を訓練した。

$$\hat{u} = \frac{1}{n} \sum_{i=1}^{n} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i). \tag{14.72}$$

平均的には、この推定値は期待収益 $J(\theta)$ の勾配に一致する。しかし、1 回の抽出ではそうならない。1 回の更新あたり 4 エピソードを用いると、訓練途中の方策における単一の推定値は勾配の長さの 3 倍以上ずれるのが常であり、成功を 1 つも含まないバッチでは更新がまったく行われなかった。その原因の大部分は重みにある。穏やかな湖ではすべての収益が 0 か γ^{T-1} のいずれかであるため、各行動は無視されるか報酬を与えられるかのどちらかになる。成功したエピソードでの遠回りは確率が引き上げられ、失敗したエピソードで穴に踏み込む一歩は決して引き下げられない。

Williams は REINFORCE を導入した 1992 年の論文でこの問題を見抜き、そこで解決策を提案した。重み付けの前に、基準となる水準すなわち**ベースライン**を収益から差し引くというものである (Williams 1992)。統計学ではこれを制御変量（control variate）と呼ぶ。平均が既知の量を差し引けば、推定値の平均を保ったまま分散（variance）を下げることができる。本節では、方策自身の行動のスコアは平均が 0 になるという 1 つの事実を出発点として、どの量を差し引いてよいのか、そして厳密な勾配を基準に測ったときそれぞれがどれだけ分散を取り除くのかを明らかにする。次章のアクター・クリティック法や、言語モデルの事後学習（post-training）に使われるグループベースの更新は、この推定量に別のベースラインを組み合わせたものにほかならない。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
```

```
                        ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

穏やかな湖、$\gamma = 0.95$、および 14.7 節のソフトマックステーブルを引き続き用いる。穏やかな湖ではすべての行動が目に見えて収益に影響するが、滑りやすい氷の上では成功の大半が運に左右され、収益から個々の行動についてわかることはほとんどない（14.7 節で確認したように、一様な初期方策では同じ精度を得るために約 9 倍大きなバッチが必要になる）。各更新には分散の違いが現れるよう 4 エピソードだけを使い、各訓練の比較では 20 個のシード（最大のステップサイズでは 100 個）を実行する。エピソードは 200 ステップで打ち切る。それほど長く歩き続けるのはループに陥った方策だけであり、その価値（value）は 0 に近いため、この打ち切りによって重みが目立って変わることはない。打ち切りを行わなければ、以下で用いる大きなステップサイズでは数千ステップに及ぶエピソードが生じてしまう。

```
gamma, eta_v = 0.95, 0.1              # discount; value-table step size
num_updates, batch_episodes = 150, 4  # small batches, to expose variance
num_seeds = 20                        # twenty seeds per arm
num_seeds_large = 100                 # at the large step size, eta = 16
env = gym.make('FrozenLake-v1', is_slippery=False, max_episode_steps=200)
```

14.8.1 ゼロ平均恒等式

本節で扱う手法はいずれもスコアの重みを変更するが、平均は変えずに保たなければならない。14.7 節の方策勾配定理（policy gradient theorem）の証明ですでに用いた 1 つの恒等式がこれを保証する。ここでは、微分可能なあらゆる方策に対してこの恒等式を述べる。

14.8.1.1 ゼロ平均補題

ソフトマックステーブルでは、行動 a の状態 s におけるスコアは、状態 s の選好（preference）に対応する 1 つのブロックだけが非ゼロとなる。

$$\frac{\partial \log \pi_\theta(a \mid s)}{\partial \theta_{s,a'}} = \mathbf{1}(a' = a) - \pi_\theta(a' \mid s). \tag{14.73}$$

s における方策自身の行動について平均をとると、指示関数の平均は $\pi_\theta(a' \mid s)$ となり、第 2 項と打ち消し合う。この相殺は、微分可能なすべての方策で成り立つ。

補題：すべての状態 s に対して、$\sum_{a \in \mathcal{A}} \pi_\theta(a \mid s)\, \nabla_\theta \log \pi_\theta(a \mid s) = 0$ が成り立つ。

証明：$\sum_a \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) = \sum_a \nabla_\theta \pi_\theta(a \mid s) = \nabla_\theta \sum_a \pi_\theta(a \mid s) = \nabla_\theta 1 = 0.$ ■

言葉で言えば、どの状態においても、サンプリングされた行動のスコアの平均は 0 である。ある行動が得る確率質量は、他の行動が失うからである。

14.8.1.2 条件付き恒等式

実用上役立つ帰結はもう少し強い。c を、エージェントが状態 s_t に立った時点ですでに定まっている任意の量（軌跡の前半で得られた報酬、定数、あるいは s_t 自身の関数など）とする。軌跡の

プレフィックス $(s_0, a_0, r_0, \dots, s_t)$ で条件付けると、値 c は確定している一方で行動 a_t はまだ確率的であり、さらに方策は s_t のみを参照するため、内側の期待値は先ほどの補題が扱う形そのものになる。

$$\mathbb{E}\big[c\,\nabla_\theta \log \pi_\theta(a_t \mid s_t)\big] = \mathbb{E}\Big[c\,\underbrace{\mathbb{E}\big[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \mid s_0, a_0, \dots, s_t\big]}_{=\,0 \text{ by the lemma}}\Big] = 0.$$

したがって、REINFORCE 推定量に含まれる任意のスコアにそのような量を掛けても、あるいはその重みからそのような量を差し引いても、平均は変わらない。条件はプレフィックスに関するものである。すなわち、a_t やそれ以降に起きる事柄に依存する量は対象外である。

14.8.2 残りの累積報酬、ベースライン、制御変量

この恒等式により、ステップ t の重みに対する 2 つの操作が正当化される。a_t より前に確定している項を落とすことと、a_t より前に確定している量を差し引くことである。どちらも平均を動かさない。そのいずれかが分散を減らすかどうかは、また別の問題である。

14.8.2.1 残りの累積報酬と因果関係

REINFORCE 推定量では、ステップ t のスコアに軌跡全体の収益 $R(\tau_i)$ が掛けられており、その中には行動 a_t^i が抽出されるより前に得られた報酬も含まれている。それらの報酬はプレフィックスによって確定しているため、条件付き恒等式により、ステップ t のスコアとの積の平均は 0 になる。それらを取り除くと、次式が残る。

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\cdot;\theta)}\Big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'}\Big]. \tag{14.74}$$

これが**因果関係**（causality）である。行動は未来に影響を与えうるが、過去には影響しない。内側の和は $\gamma^t \widehat{G}_t$ に等しく、右の因子は次式で与えられる。

$$\widehat{G}_t = \sum_{t'=t}^{T-1} \gamma^{t'-t}\, r_{t'} \tag{14.75}$$

これはステップ t からの**残りの累積報酬**（reward-to-go）、すなわちエピソードの残り部分の割引収益である。素の推定量は T 個のスコアの和に T ステップにわたる和である収益を掛けるため、毎ステップ報酬が得られる場合にはエピソード長とともに分散が急速に増大する。これに対し、過去の報酬を取り除けば、各スコアに掛ける和が短くなる。もっとも、平均 0 の項を取り除くこと自体が必ずしも分散を下げるとは限らない。取り除いた項が残りの部分と相関（correlation）している場合があるからであり、以下の測定では両方の結果が現れる。

本書の実装も含め、実際の実装ではさらに因子 γ^t も落とし、ステップ t のスコアを $\widehat{G}_t$ のみで重み付けする。

$$\widehat{u} = \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1} \widehat{G}_t^i\, \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i). \tag{14.76}$$

γ^t を落とすことは、先ほどの恒等式からは正当化されない。状態への遅い時点での訪問を早い時点での訪問と同じ重みで数えることになるため、$\gamma < 1$ のとき Equation 14.76 の平均は $\nabla_\theta J$ から傾く。Thomas (2014) はアクター・クリティックアルゴリズムにおけるこの統計的バイアス（estimator bias）を指摘し、Nota and Thomas (2020) はこの更新がいかなる目的関数の勾配にもなっていないことを示した。以下で調べる方策では、この傾きは約 8 度（コサインで 0.991）であり、平均の長さは約 1.4 倍になる（演習問題 8）。本節では、この推定量を基礎とする諸手法と共通のこの慣例を踏襲しつつ、両方の版を測定する。

すべてのステップの残りの累積報酬は、最終ステップの後ろを 0 として始めるエピソードごとの 1 回の逆方向走査 $\widehat{G}_t = r_t + \gamma \widehat{G}_{t+1}$ から求まる。`backward_scan` は、任意のベクトルと係数に対してこの漸化式を 14.7 節の `Batch.episodes` が返す各エピソードにわたり実行する（後の節では別の量を別の係数で走査する）。そして `reward_to_go` は、これを係数 γ で報酬に適用する。

```
@d2l.add_to_class(d2l.Batch)
def backward_scan(self, x, factor):
    """y_t = x_t + factor * y_{t+1}, restarted at every episode boundary."""
    y = np.zeros_like(x)
    for ep in self.episodes():
        running = 0.0
        for t in reversed(range(ep.start, ep.stop)):
            running = x[t] + factor * running
            y[t] = running
    return y

@d2l.add_to_class(d2l.Batch)
def reward_to_go(self, gamma):
    """G_t: the discounted return of the rest of its episode, by one scan."""
    return self.backward_scan(self.rew, gamma)
```

14.8.2.2 ベースライン

2 つ目の操作は減算である。任意の量 b は、s_t までの軌跡のプレフィックスによって定まる限り（定数であっても現在の状態の関数 $b(s_t)$ であっても）、残りの累積報酬から差し引くことができる。

$$\widehat{u} = \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{T-1} \left(\widehat{G}_t^i - b(s_t^i)\right) \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i). \tag{14.77}$$

このような b を**ベースライン**と呼ぶ。追加される項は $b(s_t)$ とスコアの積であり、条件付き恒等式によってその平均は 0 になるため、Equation 14.77 と Equation 14.76 の平均は一致する。

この湖ではどの残りの累積報酬も負にならないため、ベースラインがなければ、サンプリングされた行動が引き下げられることは決してない。典型的な残りの累積報酬に近い値を差し引けば、通常より良い結果となった続きの展開には正の重みが、悪かった展開には負の重みが与えられる。では、平均が変わらないのに、なぜ学習が変化しうるのか。それは次項で計算する分散を通じてである。また、重みの最良の予測値である $Q(s_t, a_t)$ の推定値を差し引かないのはなぜか。それは a_t に依存するため恒等式が適用できなくなり、平均がずれてしまうからである（演習問題 1）。

14.8.2.3 制御変量

ベースラインの減算は、モンテカルロ推定における一般的な手法の 1 つの例である。$\mathbb{E}[X]$ を求めたいとし、各サンプルには期待値が既知で相関を持つ量 Y も含まれているとする。任意のスカラー c に対して、次式を定義する。

$$X_c = X - c\,(Y - \mathbb{E}[Y]). \tag{14.78}$$

$\mathbb{E}[Y - \mathbb{E}[Y]] = 0$ であるため、推定量 X_c の期待値は X の期待値とすべての c について一致する。その分散は次式で与えられる。

$$\operatorname{Var}(X_c) = \operatorname{Var}(X) - 2c\operatorname{Cov}(X,Y) + c^2\operatorname{Var}(Y), \tag{14.79}$$

これは c の二次関数であり、次式で最小となる。

$$c^{*} = \frac{\operatorname{Cov}(X,Y)}{\operatorname{Var}(Y)}. \tag{14.80}$$

c^{*} を元の式に代入すると、分散は $(1 - \operatorname{corr}^2)\operatorname{Var}(X)$ となる。ここで corr は X と Y の相関である。相関が強ければ大幅な分散低減（variance reduction）が可能になる。$\operatorname{corr} = 0.9$ では分散は約 5 分の 1 に減るが、$\operatorname{corr} = 0$ では変わらない。この量 Y を X に対する**制御変量**と呼ぶ。

方策勾配（policy gradient）では、$X = \hat{G}_t\,\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ および $Y = \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ とおく。補題により $\mathbb{E}[Y \mid s_t] = 0$ であり、$c = b(s_t)$ とすれば Equation 14.77 が復元される。すなわち、ベースラインとはスコアから構成された制御変量である。最良の係数は状態ごとに個別に計算した c^{*} であり、ベクトル値のスコアに対しては積を内積（inner product）に置き換えることで、演習問題 2 の最適ベースライン b^{*} が得られる。Greensmith et al. (2004) は、方策勾配に対してこの解析を展開している。

最小の事例を見ると、その効果の大きさがわかる。Figure 14.19 の 1 ステップ問題を取り上げよう。1 つの行動 a を正規分布 $\mathcal{N}(a;\mu,1)$(ただし $\mu = 0$)から抽出し、報酬を $R(a) = 0.4 + 2e^{-(a-2)^2/2}$ とする。期待報酬（expected reward）は $J = \mathbb{E}[R] = 0.92$、μ に関するスコアは $a - \mu = a$、厳密な勾配は $\mathrm{d}J/\mathrm{d}\mu = 0.52$ である。1 サンプルによる推定値 $R(a)\,a$ はこの平均を持つが、標準偏差は 1.34 と推定対象の量の 2.5 倍を超える。平均報酬を差し引いた $(R(a) - 0.92)\,a$ では、標準偏差が 0.69 に下がる。最良の定数は $b^{*} = \mathbb{E}[R(a)\,a^2]/\mathbb{E}[a^2] = 1.18$ であり、平均報酬よりも大きくなる。これは、0 から離れた行動ほど大きなスコアを持ち、さらにこの問題では大きな報酬も持つためである。このとき、ベースラインなしの分散のうち $1 - \operatorname{corr}^2 = 0.23$ だけが残る（Figure 14.20）。

14.8.2.4 アドバンテージと学習されたベースライン

この湖では、1 つの定数をすべての状態に適合させることはできない。ゴールの隣のマスは、スタート地点よりも期待される残りの累積報酬がはるかに高いからである。状態に依存する自然なベースラインは、その状態の価値である。14.2 節より、$V^{\pi}(s) = \sum_a \pi(a \mid s)\,Q^{\pi}(s,a)$ は行動の価値に関する方策の平均である。ある行動の**アドバンテージ**は、その行動がこの平均よりどれだけ優れているかを測る量であり、次式で定義される。

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s), \qquad \mathbb{E}_{a\sim\pi(\cdot\mid s)}\big[A^{\pi}(s,a)\big] = 0. \tag{14.81}$$

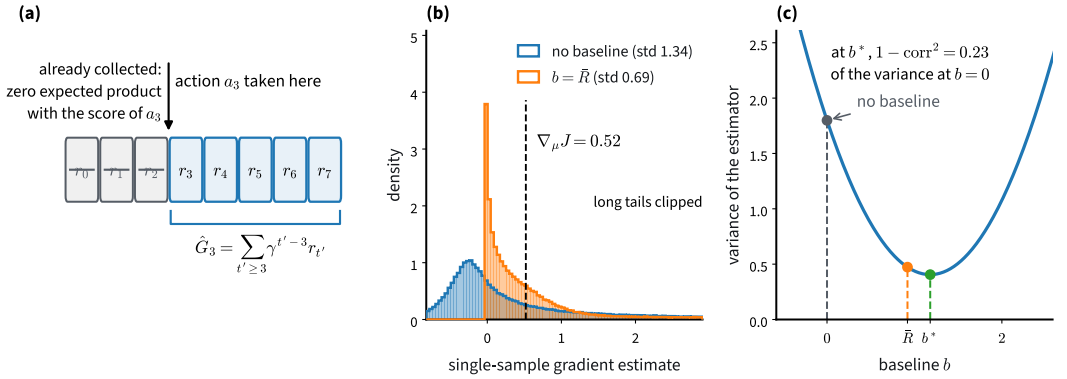


Figure 14.20: スコア関数推定量のノイズを抑える 2 つの方法。(a) 複数ステップのエピソードでは、ある行動より前に得られた報酬とその行動のスコアとの積の期待値は 0 になるため、残りの累積報酬 $\hat{G}_3$ だけが残る。(b) Figure 14.19 の 1 ステップ問題では、平均報酬を差し引くことで 1 サンプル推定値の標準偏差が 1.34 から 0.69 に下がる。両ヒストグラムとも 1 パーセンタイルと 95 パーセンタイルでクリップしている。(c) 定数ベースライン b の関数としての分散は放物線を描き、$b^* = 1.18$ で最小となり、元の分散の 0.23（$b = 0$ における値に対する比）だけが残る。

いま述べた関係から、その平均は 0 になる。正のアドバンテージは、状態 s における方策の平均を上回る行動を表す。$\mathbb{E}[\hat{G}_t \mid s_t, a_t] = Q^{\pi_\theta}(s_t, a_t)$ であるため、重み $\hat{G}_t - V^{\pi_\theta}(s_t)$ は選択された行動のアドバンテージのサンプルとなり、この更新はその状態の平均を上回る行動の確率を引き上げ、残りを引き下げる。

V^{π_θ} は未知だが、同じバッチから推定できる。テーブル $\hat{V}(s)$ を保持し、各バッチの後に訪問したすべての状態の項目を、そこで観測された残りの累積報酬に向けて次のように動かす。

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \eta_V(\hat{G}_t - \hat{V}(s_t)), \tag{14.82}$$

ここで η_V はステップサイズである。これがベースライン付き REINFORCE である (Williams 1992)。このテーブルは、完了したエピソードの残りの累積報酬、すなわちモンテカルロ収益（Monte Carlo return）に対する回帰によって当てはめられる。1 ステップ先の $\hat{V}$ 自身の予測値からターゲットを構成すると、第 15 章のアクター・クリティック法が得られる。そのアドバンテージ推定（advantage estimate）は、ブートストラップを一切行わない場合に $\hat{G}_t - \hat{V}(s_t)$ へと帰着する。

14.8.3 中心化、スケーリング、正規化

実務では、各バッチの重みを標準化したり、グループ内のサンプル同士を比較したりすることが多く、またどの実装でも損失の和を何で割るかを決めなければならない。それぞれの選択は、推定量そのものを変えるか、そのスケールだけを変えるかのいずれかであるが、この 2 つは混同しやすい。両者を見分けるには、厳密な勾配がわかっている固定された方策が必要になる。

まず、`probe` を 14.7 節とまったく同じように、`d2l.train_reinforce_tabular` と同じ 16 エピソードずつの 16 回の更新で再構築する。その価値 $J(\theta) = V^{\pi_\theta}(s_0)$ は、14.2 節の連立一次方程式（linear system）$V = r_\pi + \gamma P_\pi V$ の解の第 1 成分である。ここで $P_\pi(s' \mid s) = \sum_a \pi_\theta(a \mid s)\, P(s' \mid s, a)$ および $r_\pi(s) = \sum_a \pi_\theta(a \mid s)\, r(s, a)$ である。`probe.probs` はプローブから π_θ

を読み出し、mdp.policy_value(pi_np) はこの方程式系を NumPy で解いて、すべての状態における V^{π_θ} すなわち厳密な状態ベースラインを返す。

```
mdp = d2l.TabularMDP.from_gym(env, gamma)
probe = d2l.Policy.tabular(16, 4)        # the probe of sec_policygradient
list(d2l.train_reinforce_tabular(probe, env, gamma, 3, num_updates=16))
pi_np = probe.probs(16, 4)
V_pi = mdp.policy_value(pi_np)           # the exact state baseline
```

exact_gradient は、同じ方程式系をフレームワークの配列で記述した 14.7 節の d2l.policy_objective を、プローブの選好テーブルにおいて微分する。報酬テーブル r を引数にとるため、後ほど変更を加えた報酬に対してもそのまま使える。

```
theta_probe, P = probe.policy.embedding[...], jnp.asarray(mdp.P, jnp.float32)

def exact_gradient(r):
    """grad J at the probe for the rewards r[s, a], flattened."""
    return np.asarray(jax.grad(d2l.policy_objective)(
        theta_probe, P, jnp.asarray(r, jnp.float32), gamma)).ravel()

g_exact = exact_gradient(mdp.r)
print(f'J(theta) = {V_pi[0]:.3f}, |grad J| = {np.linalg.norm(g_exact):.3f}')
```

```
J(theta) = 0.313, |grad J| = 0.129
```

出力された $J(\theta) = 0.313$（達成可能な最大値は $\gamma^5 \approx 0.77$）は、14.7 節におけるプローブの価値である。その厳密な勾配が、以下のすべての測定の基準となる。

14.8.3.1 バッチ中心化と分散スケーリング

よく使われる変種として、各バッチ内で残りの累積報酬の値を標準化する手法がある。現在のバッチ内のすべての $\widehat{G}_t^i$ を集め、その平均 $\bar{R}$ と標準偏差 σ を計算して、次の量を用いる。

$$\tilde{G}_t^i = \frac{\widehat{G}_t^i - \bar{R}}{\sigma + 10^{-8}} \tag{14.83}$$

これを $\widehat{G}_t^i$ の代わりに使う。ここで定数 10^{-8} はゼロ除算を防ぐためのものである。$\bar{R}$ を差し引く操作はベースラインであり、サンプル間の相対的な重みを変え、推定値の方向も変えうる。ただし、$\bar{R}$ は同じバッチから計算されるため、サンプリングされた行動に弱く依存する。その結果生じる統計的バイアスについては、軌跡ごとの重みの場合を次項で、ここで用いたステップ単位の重みの場合を演習問題 3 で計算する。

$\sigma + 10^{-8}$ で割る操作は、バッチ全体の推定値に 1 つの正の数を掛けるだけなので、1 つのバッチ内ではベースラインとしてではなく、報酬のばらつきに応じて変化する適応的ステップサイズ（adaptive step size）として働く。一方、バッチをまたいで見るとこの係数は確率的でありバッチと相関するため、期待更新量も傾く。演習問題 4 では、プロンプトごとに異なる σ がもたらす関連する効果を導出する。この湖では $0 \le \widehat{G}_t \le 1$ から $\sigma \le 1/2$ が従うため、固定されたステップサイズのもとでは、正規化は中心化の少なくとも 2 倍の長さのステップをとる（以下の測定では約 3 倍となる）。standardize は Equation 14.83 を実装したものである。

```
def standardize(x, eps=1e-8):
    """Center a NumPy batch of weights and divide by its standard deviation
    (NumPy's std, which divides by the batch size)."""
    return (x - x.mean()) / (x.std() + eps)
```

14.8.3.2 1 つ抜きベースライン

バッチ平均には、差し引く対象であるサンプル自身が含まれている。1 つ抜き（leave-one-out）の中心化は、各軌跡に対して**他の**軌跡の平均をベースラインとして与えることで、この自己依存性を取り除く。

$$b_i = \frac{1}{n-1}\sum_{j \neq i} R(\tau_j).$$

各軌跡は独立に抽出されるため、b_i は軌跡 i と独立であり、条件付きゼロ平均恒等式が補正なしでそのまま成り立つ。すなわち、この推定量はどのバッチサイズでも不偏である。その重みは、$\bar{R}$ による中心化の重みを定数倍したものになる。

$$R_i - b_i = \frac{n}{n-1}(R_i - \bar{R}), \tag{14.84}$$

したがって、軌跡ごとの重みに対する単純な中心化は、推定量の平均を $(n-1)/n$ 倍（$n = 2$ では半分）にする。全列挙できるほど小さな問題（順に訪問する 2 つの状態、各状態で 2 つの行動、テーブルから与えられる報酬）で、これら両方の主張を確かめよう。`leave_one_out` は 1 バッチ分の重み $R_i - b_i$ を計算する。

```
def leave_one_out(R):
    """R_i - b_i, with b_i the mean of the other n - 1 returns: n/(n-1)
    times the centered returns."""
    return (R - R.mean()) * len(R) / (len(R) - 1)
```

次のセルでは、起こりうる 4 つの軌跡から厳密な勾配 `g` を計算し、さらにサイズ 2 の順序付きバッチ全 16 通りにわたる両推定量の厳密な平均を求める。

```
rng = np.random.default_rng(0)
th = rng.standard_normal((2, 2))              # a generic two-state table
pi2 = np.exp(th) / np.exp(th).sum(1, keepdims=True)
r2 = np.array([[0.3, 1.0], [0.6, 0.1]])       # r[s, a]; s0 -> s1 -> done
trajs = [(a0, a1) for a0 in range(2) for a1 in range(2)]
p = np.array([pi2[0, a0] * pi2[1, a1] for a0, a1 in trajs])
R = np.array([r2[0, a0] + r2[1, a1] for a0, a1 in trajs])
S = np.zeros((4, 2, 2))                       # the score of each trajectory
for i, (a0, a1) in enumerate(trajs):
    S[i] = -pi2
    S[i, 0, a0] += 1
    S[i, 1, a1] += 1
g = (p[:, None, None] * R[:, None, None] * S).sum(0)    # exact gradient
u_loo = u_cen = 0.0
for i in range(4):
```

```
    for j in range(4):                          # every batch of n = 2
        w, wc = leave_one_out(R[[i, j]]), R[[i, j]] - R[[i, j]].mean()
        u_loo += p[i] * p[j] * (w[0] * S[i] + w[1] * S[j]) / 2
        u_cen += p[i] * p[j] * (wc[0] * S[i] + wc[1] * S[j]) / 2
print(f'E[leave-one-out] equals the exact gradient: {np.allclose(u_loo, g)}')
print(f'E[centered] equals (n-1)/n of it:           {np.allclose(u_cen, g / 2)}')
```

```
E[leave-one-out] equals the exact gradient: True
E[centered] equals (n-1)/n of it:           True
```

どちらの恒等式も機械精度で成り立つ。Kool et al. (2019) は入力あたり複数のサンプルを用いる REINFORCE に対して 1 つ抜きベースラインを提案しており、これは RLOO という名称で言語モデルの事後学習に使われている (Ahmadian et al. 2024)。すなわち、1 つのプロンプトに対して n 個の応答をサンプリングし、それぞれの応答を、自身の報酬から残りの $n-1$ 個の平均報酬を引いた値で重み付けする。

14.8.3.3 バッチとしてのグループ

この湖ではすべてのエピソードが同じマスから始まるため、1 つのバッチ平均を全エピソードで共有できる。14.1 節の検証器タスク（verifier task）はそうではない。4 つの算術プロンプト、共通の 9 個の応答候補、そして正解に対して報酬 1 を与える検証器からなり、参照方策の成功確率は 0.062、また 2 つのプロンプトには正解候補が存在しないため、どの方策も 0.5 を超えることはできない。各プロンプトが固有の成功率を持つ独立した初期状態であるため、ベースラインはプロンプトごとに用意しなければならない。14.7 節の `verifier_step` は、各プロンプトに対して K 個の応答をサンプリングし、スコア $\mathbf{e}_y - \pi(\cdot \mid x)$（各応答 y に対するもの）を K 個の報酬の関数で重み付けする。1 つのプロンプトに対する K 個の応答、すなわち**グループ**がベースラインを与える。

報酬が 0 と 1 の場合、グループ平均による中心化は単純な形をとる。正解が k 個（全応答数 K のうち）であれば、グループ平均は k/K となるため、正解の応答は重み $1-k/K$ を、不正解の応答は重み $-k/K$ を受け取る。ベースラインなしでは不正解の応答の重みは 0 であったが、ここでは引き下げられるようになり、しかもそのプロンプトで方策がすでに高い頻度で成功しているほど強く引き下げられる。グループの標準偏差 $\sqrt{\hat{p}(1-\hat{p})}$（ただし $\hat{p} = k/K$）で割ると、2 つの重みは次のようになる。

$$+\sqrt{\frac{1-\hat{p}}{\hat{p}}} \quad \text{and} \quad -\sqrt{\frac{\hat{p}}{1-\hat{p}}}. \tag{14.85}$$

$K = 16$ で正解が 2 個のとき、各正解の応答は約 $+2.65$、各不正解の応答は約 -0.38 を受け取る。つまり、難しいプロンプトにおけるまれな成功は強く押し上げられる。全員不正解または全員正解のように、すべての応答が同じスコアをとるグループでは比較対象がなく、すべての応答の重みが 0 になる（これがどれほどの頻度で起きるかは 14.4 節で数えた）。

グループ内における応答 j の標準化された重みは次式で与えられる。

$$\hat{A}_j = \frac{r_j - \bar{R}}{\sigma + 10^{-8}}, \tag{14.86}$$

ここで、$\bar{R}$ と σ はグループ内の K 個の報酬の平均と標準偏差である。これはプロンプトを初期状態、グループをバッチとみなした Equation 14.83 の軌跡単位の形であり、ステップごとの

残りの累積報酬ではなく、応答ごとの 1 つの報酬を標準化する。これが Group Relative Policy Optimization（GRPO）の重みであり (Shao et al. 2024)、`group_step` は `verifier_step` の重み付けに `standardize` を用いたものである。

```
def group_step(task, theta, K, reward, rng, lr=2.0):
    """verifier_step with the group-normalized weight (GRPO): the rewards
    of each prompt's K responses, standardized within the group."""
    return d2l.verifier_step(task, theta, K, reward, rng, standardize, lr)
```

グループ平均にはその応答自身の報酬が含まれるため、1 つ抜きの計算により、グループ中心化された更新の期待値は勾配の $(K-1)/K$ 倍になる。σ による除算を除いたサイズ 2 のグループについて、これを厳密に確かめよう。次のセルでは、すべての応答のスコア（`Sv`）、9 項の和 $\sum_y \pi(y \mid x)\, r(x,y)\, (\mathbf{e}_y - \pi(\cdot \mid x))$ として求まる各プロンプトの成功率の厳密な勾配（`gv`）、および応答のすべての順序対にわたる中心化更新の厳密な平均（`u`）を計算する。

```
task = d2l.VerifierTask()                  # the instance of sec_mdp
K, pi, Rv = 2, task.policy(task.theta_ref), task.R
Sv = np.eye(len(task.resp))[None, :, :] - pi[:, None, :]  # eq_softmax_score
gv = np.einsum('xy,xy,xyv->xv', pi, Rv, Sv)          # the exact gradient
u = np.zeros_like(gv)
for y1, y2 in np.ndindex(len(task.resp), len(task.resp)):  # groups of K = 2
    w = (Rv[:, y1] - Rv[:, y2]) / 2                  # reward minus group mean
    u += (pi[:, y1] * pi[:, y2] * w)[:, None] * (Sv[:, y1] - Sv[:, y2]) / K
print(f'E of the group-centered update is (K-1)/K of the gradient: '
      f'{np.allclose(u, gv * (K - 1) / K)}')
print(f'rescaled by K/(K-1), leave-one-out, it is exact: '
      f'{np.allclose(u * K / (K - 1), gv)}')
```

```
E of the group-centered update is (K-1)/K of the gradient: True
rescaled by K/(K-1), leave-one-out, it is exact: True
```

この統計的バイアスはどれほど影響するだろうか。$K = 1$ では係数が 0 になる。グループ平均が応答自身の報酬と一致し、すべての重みが 0 になって、勾配が 0 でないにもかかわらず方策は一切動かない。$K \geq 2$ では、この係数はステップを縮小させるだけである。以下のスイープ実験では、予算をプロンプトあたり採点済み応答 6,400 個に固定するため、グループが小さいほど多くのステップを踏む（3,200 ステップが $K = 2$ の場合、200 ステップが $K = 32$ の場合である）。

```
budget = 6400                    # scored responses per prompt, matched
for K in (1, 2, 4, 8, 32):
    rng, theta = np.random.default_rng(1), task.theta_ref.copy()
    for _ in range(budget // K):
        theta = group_step(task, theta, K, task.verify, rng)
    print(f'K = {K:2d}: success of a sampled response '
          f'{task.success(task.policy(theta)):.3f}')
```

```
K =  1: success of a sampled response 0.062
K =  2: success of a sampled response 0.500
K =  4: success of a sampled response 0.500
K =  8: success of a sampled response 0.500
K = 32: success of a sampled response 0.500
```

予測どおり、$K = 1$ では方策は参照スコア 0.062 のままであり、$K \geq 2$ では試した 10 個のシ

ードすべてにおいて、同じ予算内で上限の 0.5 に到達する。本来なら学習された価値テーブルが担うはずのプロンプトごとの比較を、グループが提供しているのである。なお、プロンプトごとに σ で割ることには、この問題例には現れないさらなる効果がある。更新がどのプロンプトを優先するかが変わるのである（演習問題 4）。

14.8.3.4 損失の除数

Equation 14.76 の二重和はエピソードとステップにわたっており、実装ではこれを何らかの数で割る必要がある。エピソード数 n で割ればその式そのものになる。総ステップ数で割ると、各バッチがその平均エピソード長の逆数で再スケーリングされるが、この長さは方策の改善に伴って変化する。各エピソードをそれぞれの長さで割ると、エピソード間の重み付けが変わり、方向も変化しうる。固定された定数で割る場合はスケールだけが変わる。長さの異なる 4 エピソードからなる 1 つのバッチで、これら 4 通りすべてを計算してみよう。`agg` は、バッチ `b` の方策 `pi` のもとでのスコアを 14.7 節の `d2l.tabular_score` を用いて合計する際、各スコアに残りの累積報酬とステップごとの係数 `scale` の積を重みとして掛ける。

```
b = d2l.rollout(env, probe.act, 4, np.random.default_rng(6))
G, T = b.reward_to_go(gamma), np.diff(b.ep_ends, prepend=0)
print(f'episode lengths {T}, successes {int(b.rew.sum())}')

def agg(b, G, pi, scale):  # sum_t scale_t G_t score_t
    return d2l.tabular_score(b.obs, b.act, G * scale, pi).ravel()

grads = {'episodes': agg(b, G, pi_np, np.full(len(b), 1 / len(T))),
         'own length': agg(b, G, pi_np, np.repeat(1 / T, T) / len(T)),
         'total steps': agg(b, G, pi_np, np.full(len(b), 1 / len(b))),
         'a constant': agg(b, G, pi_np, np.full(len(b), 1 / 32))}
for k, u in grads.items():
    cos = u @ grads['episodes'] / (np.linalg.norm(u)
                                   * np.linalg.norm(grads['episodes']))
    print(f'{k:>12}: |grad| = {np.linalg.norm(u):.3f}, '
          f'cos to episodes = {cos:.3f}')
```

```
episode lengths [ 8 20  8 11], successes 3
    episodes: |grad| = 0.868, cos to episodes = 1.000
  own length: |grad| = 0.082, cos to episodes = 0.937
 total steps: |grad| = 0.074, cos to episodes = 1.000
  a constant: |grad| = 0.109, cos to episodes = 1.000
```

4 つの勾配のうち 3 つは互いに平行で、大きさには 1 桁の開きがある。エピソードごとの長さで割る版だけは、他から方向が傾いている。14.5 節の `policy_step` は 3 番目の選択肢である総ステップ数で割るため、以下の訓練ループでは 1 番目の選択肢を復元するよう重みを再スケーリングする。言語モデルの訓練でも、応答をエピソード、トークンをステップとみなせば同じ選択に直面する。GRPO は各応答をそれぞれの長さで割り、DAPO は総トークン数で割り (Q. Yu et al. 2025)、Dr. GRPO はすべてのバッチを同じ係数で再スケーリングする唯一の選択肢である定数で割る (Z. Liu et al. 2025)。

14.8.4 5 つの推定量の比較

これで、ステップ t におけるスコアに対する 5 通りの重み付けが出そろった。このうちどれが分散を減らし、その効果はどれほどだろうか。まず厳密な勾配がわかっている固定方策で答えを確かめ、続いて訓練の中で調べる。

14.8.4.1 固定方策における推定量

5 通りの重み付けは次のとおりである。

1. **軌跡の収益** $R(\tau)$：本節の冒頭に示した不偏な REINFORCE 推定量である。
2. **残りの累積報酬** $\hat{G}_t$：Equation 14.76 のとおり、因子 γ^t を落としたものである。
3. **中心化**した残りの累積報酬 $\hat{G}_t - \bar{R}$：$\bar{R}$ はバッチ内の全ステップにわたる平均である。
4. **正規化**した残りの累積報酬：Equation 14.83 である。
5. **価値ベースライン** $\hat{G}_t - V(s_t)$：固定方策では、ここでは計算可能な連立一次方程式の解から得られる厳密な V^{π_θ} を用い、訓練では Equation 14.82 の学習されたテーブル $\hat{V}$ を用いる。

6 行目の**因果的**は、ステップ t のスコアを因果関係の議論から得られた不偏推定量（unbiased estimator）である $\gamma^t \hat{G}_t$ で重み付けするものであり、過去の報酬を落とす操作と γ^t を落とす操作を切り分ける。

プローブを固定したまま、4 エピソードからなるバッチを 2,000 個抽出し、それぞれについてすべての推定量を計算する。`estimate` は `d2l.tabular_score` を用いてバッチの重み付きスコアを合計し、Equation 14.76 と同様にエピソード数で割る。`discounted` は各残りの累積報酬に γ^t（t はそのエピソードの先頭から数える）を掛ける。`weightings` は各名称を、バッチ・残りの累積報酬・価値テーブルの関数に対応付ける（`per_step` は各エピソードの収益をそのステップ全体に繰り返す）。`stats` は出力される 3 つの列を計算する。`|mean|/|grad J|` と `cos` は推定量の平均のスケールと方向、すなわち抽出された推定値の平均の $\nabla_\theta J$ に対する相対的な長さと、両者のなす角のコサインを表す。`noise` 列は相対分散であり、抽出値の共分散のトレースを、その推定量自身の平均の長さの二乗で割った値である。各推定量自身の平均を基準に測るのは、正規化された重みが中心化された重みを σ で割ったものであり、再スケーリングされた推定量がスケールの違いによって不利に評価されるべきではないからである。

```
def estimate(b, w, pi):
    """One draw of the estimator: weighted scores, averaged over episodes."""
    return d2l.tabular_score(b.obs, b.act, w, pi).ravel() / len(b.ep_ends)

def discounted(b, G, gamma):
    """gamma^t G_t: reward-to-go with the factor gamma^t of the gradient kept."""
    t = np.concatenate([np.arange(ep.stop - ep.start) for ep in b.episodes()])
    return gamma ** t * G

weightings = {                    # V: exact V^pi here, the learned table later
    'return': lambda b, G, V: b.per_step(b.episode_returns(gamma)),
    'causal': lambda b, G, V: discounted(b, G, gamma),
    'reward-to-go': lambda b, G, V: G,
    'centered': lambda b, G, V: G - G.mean(),
    'normalized': lambda b, G, V: standardize(G),
    'value baseline': lambda b, G, V: G - V[b.obs]}

def stats(u, g):
    """The scale and direction of the mean of the draws u against g, and
    the variance of the draws relative to their own mean."""
    m = u.mean(axis=0)
    size_m, size_g = np.linalg.norm(m), np.linalg.norm(g)
    return (size_m / size_g, m @ g / (size_m * size_g),
            ((u - m) ** 2).sum(axis=1).mean() / (m @ m))
```

この湖では報酬がゴールでの 1 回しかないため、落とすべき過去の報酬が存在しない。因果関係が働く様子を見るため、次のセルでは非終端マスからの移動ごとに 0.1 のコストを課して測定を繰り返す。記録されたすべてのステップはそのような移動であるため、バッチの各報酬から 0.1 を引けば新しい報酬のもとでの同じエピソードが得られる。`costly` は同じカーネルとコスト付き報酬を持つ MDP であり、厳密な勾配と V^{π} を与える。

```
step_cost = 0.1                 # charged for every move from a live cell
live = ~np.isin(env.unwrapped.desc.ravel(), [b'H', b'G'])
costly = d2l.TabularMDP(mdp.P, mdp.r - step_cost * live[:, None], gamma)
cases = {'lake': (0.0, g_exact, V_pi),
         'step cost': (step_cost, exact_gradient(costly.r),
                       costly.policy_value(pi_np))}
rng = np.random.default_rng(4)
batches = [d2l.rollout(env, probe.act, 4, rng) for _ in range(2000)]
for case, (c, g, V) in cases.items():
    bs = [d2l.Batch(b.obs, b.act, b.rew - c, b.next_obs, b.term, b.ep_ends)
          for b in batches]
    for k, fn in weightings.items():
        u = np.stack([estimate(b, fn(b, b.reward_to_go(gamma), V), pi_np)
                      for b in bs])
        scale, cos, noise = stats(u, g)
        print(f'{case:>9} {k:>14}: |mean|/|grad J| = {scale:.2f}, '
              f'cos = {cos:.3f}, noise = {noise:5.1f}')
```

```
     lake         return: |mean|/|grad J| = 0.99, cos = 0.998, noise =  13.6
     lake         causal: |mean|/|grad J| = 0.99, cos = 0.998, noise =  13.6
     lake   reward-to-go: |mean|/|grad J| = 1.38, cos = 0.987, noise =  12.3
     lake       centered: |mean|/|grad J| = 1.11, cos = 0.986, noise =   7.6
     lake     normalized: |mean|/|grad J| = 3.28, cos = 0.982, noise =   7.5
     lake value baseline: |mean|/|grad J| = 1.38, cos = 0.990, noise =   5.7
step cost         return: |mean|/|grad J| = 0.98, cos = 0.996, noise =  19.5
step cost         causal: |mean|/|grad J| = 1.00, cos = 0.999, noise =   5.8
step cost   reward-to-go: |mean|/|grad J| = 1.29, cos = 0.989, noise =   5.4
step cost       centered: |mean|/|grad J| = 1.07, cos = 0.985, noise =   5.6
step cost     normalized: |mean|/|grad J| = 2.12, cos = 0.980, noise =   5.8
step cost value baseline: |mean|/|grad J| = 1.28, cos = 0.989, noise =   3.0
```

通常の湖の設定では、軌跡の収益の行と因果的の行は抽出ごとに完全に一致する。成功エピソードの末尾に報酬が 1 つあるだけなので、どのステップでも $\gamma^t \hat{G}_t = \gamma^{T-1}$ が収益そのものになるからである。残りの累積報酬がこれらと異なるのは落とされた γ^t の分だけであり、平均は約 1.4 倍長くなり、コサインはわずかに低下する。不偏推定量を $2{,}000$ 回の抽出にわたって平均するとコサインは 0.997 付近になるため、この列に現れているのは実質的にこの傾きだけである。ノイズの列には、別のシードで抽出を繰り返しても保たれる順序が現れている。価値ベースラインが最もノイズが小さく、中心化と正規化が互いに一致してこれに続き、収益と残りの累積報酬が最もノイズが大きい。中心化は残りの累積報酬のノイズを約 40% 取り除き、価値ベースラインは約半分を取り除く。また中心化は平均の長さを残りの累積報酬の約 0.8 倍に縮めるが、これは軌跡ごとの中心化における係数 $(n-1)/n = 3/4$ に近い。σ で割ると平均の長さは約 3 倍に伸びる一方、ステップサイズとして当然であるように、相対ノイズは中心化と同じ値にとどまる。

移動ごとのコストがある設定では、過去の報酬を落とすことで収益のノイズがここでは約 3 分の 1 に減っている。しかし、常にそうなるとは限らない。落とされた項は平均こそ 0 だが残りの項と相関しており、$\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\,\mathrm{Cov}(X,Y)$ は $\mathrm{Var}(X)$ より小さくなりうる。実際、移動ごとのコストを 0.01 にすると、過去の報酬を落とすことでノイズは約 6 分の

1 増加する（演習問題 8）。他の行の順序もこの方策に固有のものであり、一様方策では中心化は役に立たない（演習問題 5）。

14.8.4.2 訓練におけるステップサイズと分散

訓練では多数のステップが積み重なり、ステップサイズが関わってくる。以下の各アームは同じシードを用い、`policy_step` に渡す重みだけが異なる。Adam は座標ごとに再スケーリングして調査対象のスケールの違いを部分的に打ち消してしまうため、ここではステップサイズ η の素の SGD で更新する。価値ベースラインのアームしか使わないものの、すべての実行において Equation 14.82 により $\hat{V}$ を更新する。`entropy` は、nat 単位のエントロピー $-\sum_a \pi(a \mid s) \log \pi(a \mid s)$ をバッチ内の状態にわたって平均するため、各状態はエージェントが訪問した回数だけ重み付けされる。

```
def entropy(theta, obs):
    """Entropy of pi(.|s) in nats, averaged over the states the batch visited."""
    logp = theta - np.logaddexp.reduce(theta, axis=1, keepdims=True)
    return float(-(np.exp(logp) * logp).sum(axis=1)[obs].mean())
```

`train` は、`weightings` のいずれか 1 つとステップサイズ `eta` を用いた素の SGD による REINFORCE である。残りの引数は環境および本節の冒頭で固定した設定である。`policy_step` はバッチの N ステップにわたって平均をとる一方、固定方策の表では n エピソードにわたって平均をとっていた。そこで、すべての重みに N/n を掛けて前者の平均を後者に変換し、各アームがその表で測定した推定量をそのまま実行するようにする。この係数がなければ実効的なステップは $\eta n/N$ となり、エピソードが短くなるにつれて大きくなってしまう。MDP を `exact` として与えると、`train` は学習されたテーブルを、これから更新される方策の厳密な V^{π_θ}（固定方策のときと同様に毎回の更新前に新たに解く）で置き換える。このアームを**厳密ベースライン**（exact baseline）と呼ぶ。各更新では、バッチの成功率、パラメータステップの長さ、およびバッチに沿ったエントロピーを記録する。

`train` はすべてのシードを一度に実行する。環境は Python 上で一度に 1 状態ずつステップを進めるため、各シードは独自の乱数生成器、環境、価値テーブルを保持し、ロールアウトはシードについてループする。一方、更新は単一の JAX 計算で行う。`sgd_step` は 1 つの選好テーブルに対する素の SGD による `policy_step` であり、`jax.vmap` がこれをシードの先頭軸にわたってマップするため、コンパイルされた 1 回の呼び出しですべてのテーブルが更新される。各シードは単独で実行した場合と同じ乱数を抽出する。

```
def sgd_step(theta, obs, act, w, mask, eta):
    """policy_step by plain SGD on one preference table: a step on
    -(1/N) sum_t w_t log pi(a_t|s_t) over the N real rows of the batch."""
    def loss(theta):
        logp = jax.nn.log_softmax(theta[obs])[jnp.arange(len(act)), act]
        return -(mask * w * logp).sum() / mask.sum()
    return theta - eta * jax.grad(loss)(theta)

# Many independent runs at once: map the step over a leading axis of seeds
sgd_steps = jax.jit(jax.vmap(sgd_step, in_axes=(0, 0, 0, 0, 0, None)))
```

これらの補助関数を用いて、`train` は素の SGD による REINFORCE ループを `num_updates` バッチにわたって実行する。各ステップの重みは `weighting(batch, G, V)` の出力に `len(batch) / batch_episodes` を掛けてスケーリングし、価値テーブル `V` を Equation 14.82 によって更新し（`exact` が与えられた場合は V^π を厳密に解く）、平均収益、パラメータステップのノルム、方策エントロピーを記録する。

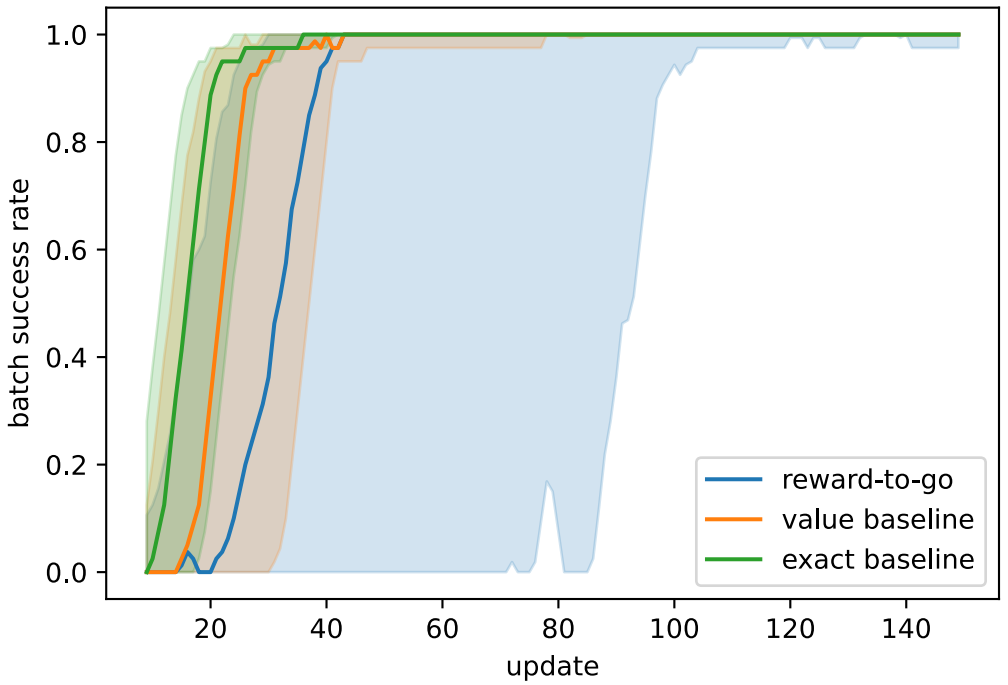
```
def train(seeds, weighting, eta, env, gamma, num_updates, batch_episodes,
          eta_v, exact=None):
    """REINFORCE by plain SGD for all seeds at once; the arms differ in one
    line, the weight. With an MDP as `exact`, V is the exact V^pi of the
    current policy. Returns the curves, of shape (seeds, updates, 3)."""
    S, A = env.observation_space.n, env.action_space.n
    n, size = len(seeds), batch_episodes * env.spec.max_episode_steps
    rngs = [np.random.default_rng(seed) for seed in seeds]
    envs = [gym.make(env.spec) for _ in seeds]
    for e, seed in zip(envs, seeds):
        e.reset(seed=seed)
    theta = np.zeros((n, S, A), np.float32)  # Policy.tabular starts at zero
    Vs = [np.zeros(S, np.float32) for _ in seeds]
    curves = np.zeros((n, num_updates, 3))
    for k in range(num_updates):
        pi = np.exp(theta - theta.max(-1, keepdims=True))
        pi /= pi.sum(-1, keepdims=True)
        obs, act = (np.zeros((n, size), np.int32) for _ in range(2))
        w, mask = (np.zeros((n, size), np.float32) for _ in range(2))
        batches = []
        for i in range(n):                  # padded batches, one row per seed
            batch = d2l.rollout(envs[i], lambda s, rng: int(rng.choice(
                A, p=pi[i, s])), batch_episodes, rngs[i])
            G, N = batch.reward_to_go(gamma), len(batch)
            if exact is not None:           # V^pi of the policy that acted
                Vs[i] = exact.policy_value(pi[i])
            obs[i, :N], act[i, :N], mask[i, :N] = batch.obs, batch.act, 1
            w[i, :N] = weighting(batch, G, Vs[i]) * N / batch_episodes
            batches.append((batch, G))
        new = np.asarray(sgd_steps(theta, obs, act, w, mask, eta))
        for i, (batch, G) in enumerate(batches):
            for s, g in zip(batch.obs, G):  # eq_value_baseline, every arm
                Vs[i][s] += eta_v * (g - Vs[i][s])
            curves[i, k] = (batch.episode_returns().mean(),
                            np.linalg.norm(new[i] - theta[i]),
                            entropy(theta[i], batch.obs))
        theta = new
    return curves
```

各重み付けはスケールが異なるため、共通の η を用いるとステップの長さが異なり、ステップの長さだけでも学習の速さは変わる。平均ステップ長をそろえても、この問題は解消されない。ステップの長さにはノイズが含まれるうえ、ベースラインを持つアームのステップは方策が成功し始めると縮小するのに対し、ベースラインを持たないアームはすでに選んでいる行動を引き上げ続けるため、平均でそろえたアーム同士でも学習中のステップ長は依然として異なるからである。純粋な比較を行うには、代わりに期待更新量を固定する。厳密な V^{π_θ} はバッチを抽出する前に確定しており、過去のバッチのみに当てはめられた学習済みの $\hat{V}$ も同様である。したがって条件付き恒等式により、残りの累積報酬、価値ベースライン、厳密ベースラインはあらゆる θ において同じ期待更新量を持ち、同一の η のもとではノイズのみが異なる。そこで、6 つのアームすべてを $\eta = 2$ で実行し（`shared`）、さらにこれら 3 つのアームを $\eta = 16$ で `num_seeds_large` 個のシードにわたって再実行する（`large`）。`setup` は全実行で共通の設定をまとめた辞書であり、各アームの実行結果は形状 (seeds, updates, 3) の配列に積み重ねられる。

```
variants = ['return', 'reward-to-go', 'centered', 'normalized',
            'value baseline']
arms = {v: dict(weighting=weightings[v]) for v in variants}
arms['exact baseline'] = dict(weighting=weightings['value baseline'],
                              exact=mdp)
setup = dict(env=env, gamma=gamma, num_updates=num_updates,
             batch_episodes=batch_episodes, eta_v=eta_v)
shared = {v: train(range(num_seeds), eta=2.0, **kw, **setup)
          for v, kw in arms.items()}
large = {v: train(range(num_seeds_large), eta=16.0, **arms[v], **setup)
         for v in ('reward-to-go', 'value baseline', 'exact baseline')}
```

共通の平均を持つ 3 つのアームの $\eta = 16$ における成功率を、10 回の更新ウィンドウで平滑化して次に示す。各帯はシードの中央値の周りの四分位範囲を表している。

```
d2l.plot_curves({v: r[:, :, 0] for v, r in large.items()}, xlabel='update',
                ylabel='batch success rate', smooth=10, band='iqr')
```



`to90` は、直近 10 回の更新にわたる成功率が 90% に達する最初の更新を求める（許容誤差は 10 回中 9 回の丸め誤差を吸収するためのものである）。次のセルでは、両方のステップサイズについて、シードにわたる中央値と最速・最遅のシード、150 回の更新以内に一度も 90% に達しない実行の数、および 90% に達する前の更新にわたる平均ステップ長を出力する。

```
def to90(curve):
    """Last update of the first 10-update window whose mean success is 0.9."""
    hit = np.convolve(curve, np.ones(10) / 10, 'valid') >= 0.9 - 1e-9
```

```
        return hit.argmax() + 9 if hit.any() else len(curve)

    for name, runs in (('eta = 2', shared), ('eta = 16', large)):
        print(name)
        for v, r in runs.items():
            reach = np.array([to90(c) for c in r[:, :, 0]])
            climb = np.mean([c[:t, 1].mean() for c, t in zip(r, reach)])
            print(f'{v:>14}: to 90%: median {np.median(reach):5.1f} '
                  f'[{reach.min():3d}, {reach.max():3d}], never '
                  f'{(reach == num_updates).sum():2d}, |step| {climb:.2f}')
```

```
eta = 2
        return: to 90%: median  63.0 [ 42, 108], never  0, |step| 0.43
  reward-to-go: to 90%: median  51.0 [ 29, 109], never  0, |step| 0.53
      centered: to 90%: median  58.5 [ 39, 119], never  0, |step| 0.32
    normalized: to 90%: median  32.0 [ 15,  85], never  0, |step| 0.93
value baseline: to 90%: median  52.0 [ 38,  97], never  0, |step| 0.37
exact baseline: to 90%: median  50.5 [ 33,  96], never  0, |step| 0.40
eta = 16
  reward-to-go: to 90%: median  35.5 [ 11, 150], never 15, |step| 1.44
value baseline: to 90%: median  26.0 [ 11, 150], never  5, |step| 1.30
exact baseline: to 90%: median  20.5 [ 10,  41], never  0, |step| 1.68
```

共通のステップサイズでは、順序はステップサイズ効果によるものとなる。正規化は最も少ない更新回数（50〜70 回に対して約 30 回）で済むが、90% に到達する前のステップは中心化の約 3 倍、残りの累積報酬の 1.5 倍以上の長さがある。中心化はステップを縮小させるため、素の収益とほぼ同じ速さにとどまり、自身が中心化する重みである残りの累積報酬よりも遅い。共通の平均を持つ 3 つのアームはこのステップサイズでは区別がつかず、いずれも約 50 回の更新を要する。

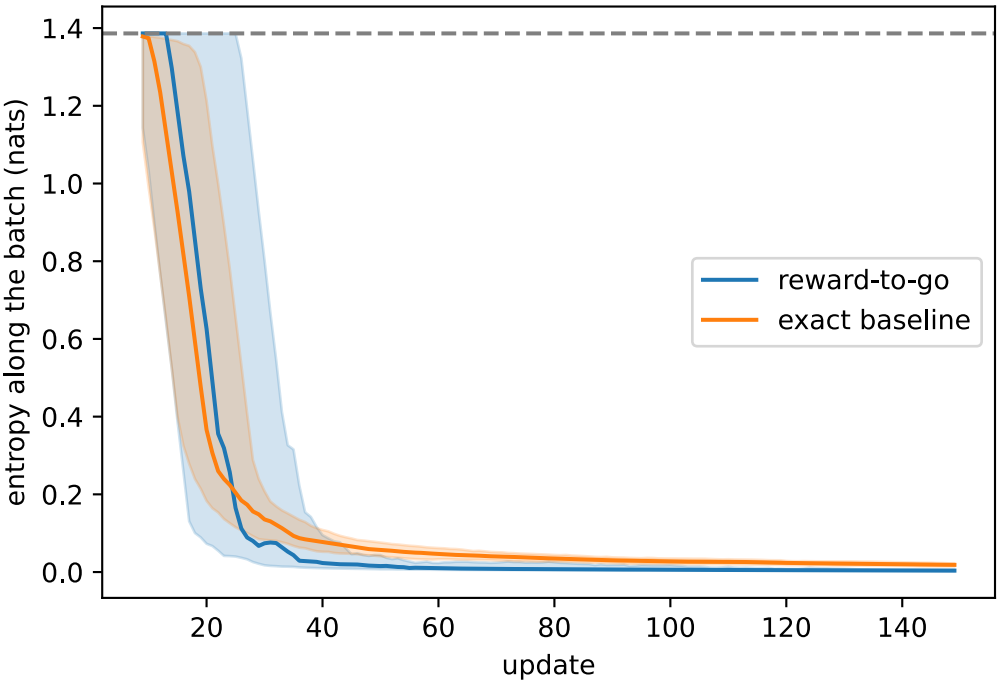
$\eta = 16$ では 3 つに差がつく。3 つとも $\eta = 2$ のときより速く学習するが、最も恩恵を受けるのは厳密ベースラインであり、その中央値は 50 回から約 20 回の更新へと下がる（残りの累積報酬は 51 回から約 35 回）。さらに大きな違いは失敗例に現れる。厳密ベースラインの実行では失敗が 1 つもないのに対し、残りの累積報酬では約 7 回に 1 回の実行が一度も 90% に到達しない。そうした実行のほぼすべてが停滞している。ノイズの乗った方向に長いステップを踏んだ結果、制限時間までループするか穴に落ちる経路上で方策がほぼ決定論的になり、以降のすべてのエピソードの収益が 0、すべての重みが 0 となって、方策が二度と動かなくなるのである。学習されたテーブルは両者の中間に位置する（中央値は 26 回の更新で、20 回に 1 回の実行が 90% に到達しない）。このステップサイズでは、$\hat{V}$ が追随できる以上に速く方策が変化するためである。ステップ長の列を見ると、この改善がステップの伸長によるものではないことがわかる。90% に到達する前、厳密ベースラインの 1 更新あたりの移動量は、残りの累積報酬より最大でも約 6 分の 1 大きいだけである。小さなステップを数多く踏めばノイズは平均化されて消えるため、分散が低くても小さなステップが良くなるわけではない。分散の低減は、より長いステップを使えるようにするのである（演習問題 6）。

$\eta = 2$ ではどのアームでも最遅のシードは最速のシードの 2〜6 倍の更新回数を要し、$\eta = 16$ では厳密ベースライン以外の実行のばらつきがさらに広がる。そのため、20 シードでは近接した中央値の順序が入れ替わりうる。アームあたり 200 シードを用いた別の実験で、そのうち 20 シードを無作為に抽出すると、$\eta = 2$ における残りの累積報酬と厳密ベースラインの中央値は約 7 回に 1 回の割合で 6 回超の差がつく。一方、$\eta = 16$ では 50 回の抽出のうち約 49 回で厳密ベースラインの方が低い中央値となるものの、残りの累積報酬のシードが 1 つ挙動を変えるだけで 20 シードの中央値が 15 回以上動きうる。そこで、$\eta = 16$ の各アームでは 100 シードを実行している。本節では中央値と範囲を報告し、主張に依拠する前により多くのシードで検証する (Henderson

et al. 2018; R. Agarwal et al. 2021)。

エージェントが訪問する状態に沿って、方策はどれほど決定論的になるだろうか。

```
d2l.plot_curves({v: large[v][:, :, 2] for v in ('reward-to-go',
                                                'exact baseline')},
                xlabel='update', ylabel='entropy along the batch (nats)',
                reference=np.log(4), smooth=10, band='iqr')
```



$\eta = 16$ では、両アームの中央値は最初の 20〜40 回の更新のうちに、一様方策の $\ln 4 \approx 1.39$ nat からバッチに沿って 0.1 nat 未満へと低下し、通常は残りの累積報酬の方が先に低下する。停滞した実行も成功した実行と同じくらい決定論的であり、繰り返す経路上では 0 nat 付近になる。なお、全 16 状態にわたって平均をとると 0.4 nat を上回ったままになる。4 つの穴とゴールでは行動が一切とられず、$\ln 4$ のまま残るためである。

以上で、どの重みが使用可能か、それぞれがどれだけ分散を取り除くか、そしてより優れた推定量とより長いステップとをどう見分けるかが明らかになった。本節の推定量はすべて、16 行の選好と 16 個の $\widehat{V}$ の要素からなるテーブルを参照していた。しかし、レール上の台車は 4 つの実数で記述され、同じ状態を二度と訪問することはない。14.9 節では、これら両方のテーブルをネットワークに置き換える。

14.8.5 まとめ

方策自身の行動のスコアは平均が 0 であり、その行動より前に確定している任意の量との積も同様に平均が 0 となる。これにより、期待勾配を変えることなく過去の報酬を落とし、ベースラインを差し引くことができる。ベースラインとは制御変量であり、重みをサンプリングされたアドバンテージへと変える状態価値 V^{π} は、測定した方策においてノイズの約半分を取り除いた。各軌跡の収益をバッチ平均やグループ平均で中心化すると期待勾配に $(n-1)/n$ が掛かるが、こ

れは 1 つ抜きによって解消される。ステップごとに中心化する場合は、別の係数で縮小する。ばらつきで割る操作はステップサイズを変え、二値報酬のもとでは GRPO のグループ正規化重みが失敗例を引き下げる（サイズ 1 のグループでは学習できない）。共通のステップサイズを用いると、最も長いステップをとる重み付けが有利になる。同一のステップサイズかつ共通の平均で比較すると、厳密な価値ベースラインと残りの累積報酬は小さなステップでは同じ速さで学習するが、大きなステップではベースラインの方が速く学習し、停滞もしない。

14.8.6 演習問題

1. **許容されるベースライン**。条件付きゼロ平均恒等式により、エージェントが s_t に立った時点ですでに定まっている任意の量を $\hat{G}_t$ から差し引くことができる。以下の各候補について、Equation 14.77 においてそれを差し引いたときに期待勾配が不変に保たれるかどうかを判定し、その条件を用いて理由を示せ。

   1. 時間インデックス t
   2. 和 $r_0 + \cdots + r_{t-1}$（ステップ t より前に得られた報酬の合計）
   3. エピソードに残っているステップ数 $T - t$
   4. 次の状態 s_{t+1} の関数
   5. 行動価値の推定値 $\hat{Q}(s_t, a_t)$

   最後の候補で $\hat{Q} = Q^{\pi_\theta}$ としたとき、減算後のステップ t の期待寄与を計算せよ。この結果は、行動に依存するベースラインについて何を示しているか。

2. **最適定数ベースライン**。状態 s を固定し、サンプリングされた行動のスコアを $\psi = \nabla_\theta \log \pi_\theta(a \mid s)$ と書き、1 ステップ推定量 $(\hat{G} - b)\,\psi$（b は定数）を考える。その平均は b に依存しないため、ノイズを共分散のトレースで測る。

$$\mathbb{E}\big[\|(\hat{G} - b)\,\psi\|^2\big] - \big\|\mathbb{E}[\hat{G}\,\psi]\big\|^2. \tag{14.87}$$

   1. このトレースを b について最小化し、最小点が次式で与えられることを示せ。

$$b^* = \frac{\mathbb{E}\big[\|\psi\|^2\,\hat{G}\big]}{\mathbb{E}\big[\|\psi\|^2\big]}. \tag{14.88}$$

      他の任意の b はトレースをちょうど $\mathbb{E}\big[\|\psi\|^2\big]\,(b - b^*)^2$ だけ増やすこと、および積を内積に置き換えたときに b^* が Equation 14.78 の係数 c^*（ただし $X = \hat{G}\,\psi$、$Y = \psi$）に一致することを示せ。

   2. Figure 14.20 の 1 ステップ問題では、行動は $a \sim \mathcal{N}(\mu, 1)$（$\mu = 0$）であるため、$\mu$ に関するスコアは a となる。b^* を $R(a) = 0.4 + 2e^{-(a-2)^2/2}$ に対して計算し、値 $b^* = 1.18$ を確認せよ。なお、標準正規分布の密度関数と $e^{-(a-2)^2/2}$ の積は別の正規分布の密度関数に比例する。なぜ b^* は $\mathbb{E}[R] = 0.92$ を上回るのか。

3. [code] **バッチ中心化の統計的バイアス**。●14.8.3.2 節では、軌跡ごとのバッチ中心化が期待勾配を縮小させる係数 $(n-1)/n$ を、1 つ抜きの恒等式 $R_i - b_i = \frac{n}{n-1}(R_i - \bar{R})$ から導いた。この議論は軌跡ごとの重みにしか適用できない。

   1. 1 つ抜きベースラインを使わずに、この係数を直接導出せよ。中心化された推定値を $\frac{1}{n}\sum_{i=1}^{n}\big(R(\tau_i) - \bar{R}\big)\,\mathbf{S}_i$ と書き（ここで $\mathbf{S}_i$ は軌跡 i のスコアの和、$\bar{R}$ はバッチの平

均収益である)、$\mathbb{E}[\bar{R}\,\mathbf{S}_i]$ を n 個の項に展開して、軌跡間の独立性を用いよ。非ゼロの期待値を持つのはどの項か。

2. 一方、`train` の中心化アームは、バッチ内のすべての残りの累積報酬の値の平均をステップごとに差し引く。この重み付けへと全列挙を拡張せよ。この全列挙では割引を用いないため、2 状態問題の各軌跡は $\hat{G}_0 = r_0 + r_1$ および $\hat{G}_1 = r_1$ を持ち、`S[i]` の第 s 行は状態 s でとられたステップのスコアとなる。$n \in \{2, 4, 8\}$ に対して推定値の期待値を計算し、ノルムの比とコサインを通じて `g` と比較せよ。
3. この比較結果を説明せよ。2 つのステップのそれぞれについて、そのステップのスコアとの積の期待値が非ゼロになるのは、バッチ平均に含まれるどの残りの累積報酬の値か。

4. **σ 正規化目的関数。**●GRPO は各グループの中心化された報酬をその標準偏差 σ で割るが、この選択については Z. Liu et al. (2025) が解析している。プロンプト x が二値報酬（正解の応答で $r = 1$、それ以外で $r = 0$）を持ち、現在の方策がこのプロンプトで成功する確率を p_x とする。グループが十分に大きく、$\bar{R}$ と σ がそれぞれの母集団の値 p_x と $\sqrt{p_x(1-p_x)}$ に等しいとする。

   1. σ を $p_x = 0.5$ および $p_x = 0.9$ のそれぞれについて計算せよ。この除算によって、2 つ目のプロンプトの更新は 1 つ目のプロンプトに対して何倍に増幅されるか。

   2. y をサンプリングされた応答とする。次式を示せ。

$$\mathbb{E}\big[(r - p_x)\,\nabla_\theta \log \pi_\theta(y \mid x)\big] = \nabla_\theta\, p_x. \tag{14.89}$$

   これより、σ 正規化更新が $\sum_x h(p_x)$（ただし関数 h は $h'(p) = 1/\sqrt{p(1-p)}$ を満たす）を上昇させることを結論づけ、h を求めよ。

   3. h を正規化しない更新の目的関数 $\sum_x p_x$ と比較せよ。それと比べて重みが増すのはどのプロンプトか。訓練セットの大半のプロンプトがほぼ常に解けるかほぼ決して解けないかのどちらかであるとき、これは実務者が望む重み付けといえるか。

   4. K 個の応答からなる有限のグループでは、K 個の報酬がすべて一致したときにプロンプトは勾配に寄与しない。$p_x = 0.9$ かつ $K = 8$ のときにこの事象が起きる確率を計算し、これが小さな σ を持つプロンプトの増幅をどのように制限するか説明せよ。

5. [code] **方策にわたる分散の順序。**●14.8.4.1 節の固定方策の比較では、16 回の更新後の 1 つの方策 `probe`（$J(\theta) = 0.313$）を調べた。プローブの更新回数 $k \in \{0, 8, 32, 64\}$ の後に到達する各方策でも同じ比較を繰り返せ。すなわち、`probe` を訓練するループの反復回数を変更し、`theta_probe`、`pi_np`、`g_exact`、`V_pi` を再計算してから 2,000 個のバッチを抽出せよ。

   1. 5 つの推定量のノイズを k に対して表にまとめよ。16 回の更新で見られた順序はすべての方策で保たれるか。中心化が役に立たなくなるのはどの方策であり、それはなぜか。
   2. $J(\theta)$ が最大値 $\gamma^5 \approx 0.77$ に近づくにつれて、ベースラインを持たない 2 つの推定量はどのように振る舞うか。また、それはなぜか。
   3. $k = 0$ において、成功を 1 つも含まないバッチの個数を数え、そのようなバッチが 5 つの推定量のそれぞれにどのように寄与するか説明せよ。

6. [code] **ステップサイズの交絡。**●共通のステップサイズでは、`train` の正規化アームが最も少ない更新回数で成功率 90% に到達したが、同時に最も長いステップもとっていた。$\sigma > 0$ であるバッチでは、正規化された重み Equation 14.83 は、中心化された重みをステップサイズ $\eta/(\sigma + 10^{-8})$ で SGD 更新したものと一致する。

1. 正規化アームにおいてバッチごとに σ を記録し、成功を含まないバッチを除いたうえで、$1/(\sigma + 10^{-8})$ のシードにわたる中央値を更新インデックスに対してプロットせよ。方策が改善するにつれて、このステップサイズの乗数はどのように変化するか。正規化アームを η を実行全体の $1/(\sigma + 10^{-8})$ の平均で割ったステップサイズで動かして両アームをそろえようとする試みに対し、このことは何を意味するか。
2. 残りの累積報酬、価値ベースライン、厳密ベースラインの比較を $\eta \in \{4, 8, 32\}$ で繰り返し、各アームについて η に対する 90% 到達までの更新回数の中央値と、一度も到達しない実行の割合をプロットせよ。各アームにとって最良のステップサイズはどれか。推定量のノイズが下がるにつれて最良のステップサイズが大きくなる理由を説明せよ。
3. 代わりにステップ長をそろえると、ノイズの大きいアームにはより小さい η が与えられる。`shared` の 5 つのアームのそれぞれについて、最初の 40 回の更新にわたる平均ステップ長が約 0.6 になるステップサイズを選び、各実行が 90% に到達する前と後とで平均ステップ長を別々に記録せよ。学習中により長いステップをとるのはどのアームか。また、平均ステップ長をそろえた比較でも依然として分散とステップサイズが混ざり合ってしまうのはなぜか。

7. [code] **最適状態ベースライン**。●●演習問題 2 では、1 つの状態における分散最小の定数ベースラインを導出した。これを状態ごとに適用すると、1 ステップ分の項の分散を最小化する Greensmith et al. (2004) の最適ベースライン関数が得られる。本問では、表形式のソフトマックス方策についてこれを具体的に求め、エピソード単位の推定値 `estimate` に対しても最適となる条件を調べる。

   1. ソフトマックスのスコアから次式を示せ。

$$\|\nabla_\theta \log \pi_\theta(a \mid s)\|^2 = 1 - 2\pi_\theta(a \mid s) + \sum_{a'} \pi_\theta(a' \mid s)^2. \tag{14.90}$$

      さらに $\mathbb{E}[\hat{G}_t \mid s_t = s, a_t = a] = Q^{\pi_\theta}(s, a)$ を用いて、最適ベースライン $b^*(s)$ を $\pi_\theta(\cdot \mid s)$ と $Q^{\pi_\theta}(s, \cdot)$ で表せ。

   2. 1 エピソードの推定値は、その全ステップの重み付きスコアの和である。表形式のパラメータ化では異なる状態における 2 つのステップのスコアが直交し、その結果、推定値の二乗ノルムがステップごとの項と同一状態への複数回の訪問間の交差項とに分解されることを示せ。エピソードに関するどのような条件のもとで、$b^*(s)$ は `estimate` の共分散のトレースを厳密に最小化するか。FrozenLake はその条件を満たすか。

   3. `probe` において $b^*(s)$ を計算し（Q^{π_θ} は `mdp.backup(V_pi)` から得る）、重み付け `G - b_star[b.obs]` を 14.8.4.1 節の `weightings` に `lambda b, G, V: G - b_star[b.obs]` として追加せよ。`probe` およびプローブの 64 回の更新後の方策において、そのノイズを厳密ベースラインのノイズと比較せよ。

   4. $b^*(s)$ が $V^{\pi_\theta}(s)$ と最も大きく異なるのはどの状態か。重み $\|\nabla_\theta \log \pi_\theta(a \mid s)\|^2$ を用いて、ほぼ決定論的な状態がなぜ $b^*(s)$ をまれにしか選ばれない行動の行動価値の側へ引き寄せるのかを説明し、2 つのベースラインの差を演習問題 2a で求めた過剰分散と関連づけよ。

8. [code] **残りの累積報酬の分解**。●Equation 14.76 の残りの累積報酬の重みは、軌跡の収益に対して 2 つの変更を加えている。各行動より前に得られた報酬を落とすこと（これは条件付き恒等式で許容される）と、因子 γ^t を落とすこと（これは許容されない）である。

   1. $\rho(s) = \mathbb{E}\big[\sum_t \mathbf{1}(s_t = s)\big]$ を 1 エピソードにおける s への期待訪問回数とする。演習問題 7a と同様に、Equation 14.76 の平均は $\sum_s \rho(s) \sum_a \nabla_\theta \pi_\theta(a \mid s)\, Q^{\pi_\theta}(s, a)$ で

ある一方、方策勾配は各状態を $\sum_t \gamma^t \Pr(s_t = s)$ で重み付けする。両者が一致するのはどの γ のときか。

2. `probe` において、穴でもゴールでもない 11 個のマス上で ρ を $\rho = \mathbf{e}_{s_0} + P_\pi^\top \rho$ を解くことで計算し、`mdp.backup(V_pi)` から Q^{π_θ} を得て、Equation 14.76 の平均を厳密に評価せよ。その `g_exact` とのコサインおよび `g_exact` に対する相対的な長さを、固定方策の表の残りの累積報酬の行と比較せよ。
3. 過去の報酬を落とすことは、平均 0 の項 Y を推定値 $X + Y$ から取り除くことに相当する。$\mathrm{Var}(X) < \mathrm{Var}(X + Y)$ となるのは $\mathrm{Var}(Y) + 2\,\mathrm{Cov}(X, Y) > 0$ のとき、かつそのときに限ることを示せ。移動ごとのコストを 0.01（0.1 の代わり）とした場合、およびプローブの 64 回の更新後に到達する方策において、収益の行と因果的の行の比較を繰り返せ。因果関係は常にノイズを減らすか。それぞれのケースを、過去の報酬と残りの累積報酬との共分散の符号によって説明せよ。

14.9 強化学習における関数近似

既知の運動方程式から計算した 14.6 節のプランナを除き、ここまで学習してきた量はすべてテーブルに格納されていた。湖のマスに対応する 16 行や、検証器タスクのプロンプトごとの 1 行である。ここで、上部にヒンジで取り付けられたポールのバランスをとる台車を思い描いてみよう。その状態は台車の位置と速度、およびポールの角度と角速度という 4 つの実数であり、1,000 エピソードを実行してもエージェントがまったく同じ 4 つの数値を 2 度目にすることはない。このような状態上のテーブルでは訪問のたびに新しい行が必要になり、1 回の訪問で学習したことは次の訪問について何も語らない。ポールがある速度で左に傾いているときにエージェントが学習したことを、わずかに異なる速度で左に傾いた瞬間にも引き継げるよう、似た状態に似た値を与える関数が求められる。

この問題は、この分野そのものと同じくらい古い。Bellman は状態変数の数に伴うテーブルの肥大化を**次元の呪い**（curse of dimensionality）と名付け (Bellman 1957b)、1959 年にはすでに Arthur Samuel のチェッカープログラムが盤面をテーブルから参照する代わりに特徴量の重み付き和でスコア付けしていた (Samuel 1959)。テーブルをパラメータ化された関数で置き換えることを**関数近似**（function approximation）と呼ぶ。この関数としてニューラルネットワークを用いることで、強化学習（reinforcement learning）はバックギャモン (Tesauro 1995)、Atari ゲーム (Mnih et al. 2015)、そして囲碁へと広がった (Silver et al. 2016)。

方策勾配（policy gradient）にとってこの一歩は小さい。14.7 節および 14.8 節の導出では、どこでもテーブルという形式を使っていなかったからである。REINFORCE が必要とするのはスコア $\nabla_\theta \log \pi_\theta(a \mid s)$ のみであり、ネットワークもテーブルの行とまったく同様にこれを供給できる。変わるのは、1 回の更新があらゆる状態における推定値を動かすようになる点である。この共有によって連続状態での学習が可能になる一方で、学習が破綻する新たな要因も生まれる。本節では、ポール付きの台車と連続トルクで駆動される振り子という 2 つの小規模な制御タスクでこの一歩を踏み出し、1 回の更新がどこまで広がるかを測定したうえで、こうして得られるエージェントになお何ができないのかを確認して締めくくる。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

14.9.1 テーブルに代わるネットワーク

前節までのテーブルは、方策（policy）と、ベースラインとして用いた価値推定の 2 か所でネットワークに置き換わる。いずれのネットワークを書く前にも、まずはネットワークがその自然な拡張となる形でテーブルを捉え直しておこう。

14.9.1.1 ワンホット状態上の線形ネットワークとしてのテーブル

`Policy.tabular`（14.5 節）は選好を 16×4 の埋め込みテーブル θ に格納しており、状態 s における選好はそのテーブルの第 s 行である。第 s 行は $\theta^\top \mathbf{e}_s$ に等しい。ここで $\mathbf{e}_s$ は状態 s のワンホットベクトルである。したがって、テーブルとは入力特徴量が固定されたワンホットベクトルである線形ネットワークにほかならない。連続状態に対する古典的な対処法も同じ形式をとる。状態空間を複数の箱に分割し、$\mathbf{e}_s$ を s が属する箱のワンホットベクトルとするのである。後述する台車とポールの課題に対する初期の学習型制御器では、人手で設計した分割による 162 個の箱が使われていた (Barto et al. 1983)。1 つの隠れ層を持つネットワークはこの形式を保ったまま、固定された特徴量を学習される特徴に置き換える。すなわち、選好は $\mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{s} + \mathbf{b}_1) + \mathbf{b}_2$ となり、ここで $\mathbf{s}$ は実数値の観測ベクトルそのものである。

これら 2 つの形式は、1 回の更新が何を引き起こすかという点で異なる。価値推定 $\hat{V}_w$（パラメータ w）に対し、サイズ η_V の勾配ステップを $\frac{1}{2}\big(y - \hat{V}_w(s_0)\big)^2$ 上で 1 回とって、単一の状態 s_0 における推定値をターゲット y に近づけるとする。η_V の一次までで、任意の状態 s における推定値の変化量は次式で与えられる。

$$\Delta \hat{V}(s) = \eta_V (y - \hat{V}(s_0))\, k(s, s_0), \qquad k(s, s_0) = \nabla_w \hat{V}(s)^\top \nabla_w \hat{V}(s_0). \tag{14.91}$$

したがって、s における変化量は、s_0 で求められた変化量に 2 つの状態における勾配の重なり具合を掛けたものになる。テーブルの場合、$\nabla_w \hat{V}(s) = \mathbf{e}_s$ であるため、$k(s, s_0)$ は 1（$s = s_0$ のとき）または 0（それ以外のとき）となる。すなわち、1 つの状態での勾配ステップは他のすべての要素を元のまま保つ。（`Policy` のデフォルトオプティマイザである Adam は、蓄積されたモメンタムを通じて現在のバッチで勾配が 0 である要素も動かし得るため、この記述はあらゆるオプティマイザのステップではなく勾配について成り立つ。）ネットワークの場合、k は 2 つの状態の滑らかな関数であり、そのネットワークの**接カーネル**（tangent kernel）と呼ばれる (Jacot et al. 2018)。このとき、s_0 でのステップは似た状態における推定値も一緒に動かす。演習問題 7 では Equation 14.91 を導出し、14.9.4.1 節では訓練するネットワークについて k の形状を測定する。

14.9.1.2 方策ネットワークと価値ネットワーク

`Policy` コンテナ（14.5 節）は、方策ネットワークとそのオプティマイザを保持する。14.8 節では、学習されたベースラインは別個の NumPy テーブルであり、各バッチの後に $\hat{V}(s_t)$ をステップサイズ η_V で、訪問したすべての状態で観測された残りの累積報酬（reward-to-go）$\hat{G}_t$ に向かって動かしていた。ここではベースラインもネットワークになり、コンテナの内部に移る。`ActorCritic` は `Policy` を継承し、価値モジュール `value` とそれ専用の Adam オプティマイザ `opt_v` を追加する。`V` は損失が微分する形式であるフレームワーク配列として価値を返し、`value_np` は `Policy` の境界メソッドと同様に NumPy 配列を受け取って返す。この名前は 2 つの構成要素に由来しており、方策はしばしば**アクター**、価値推定は**クリティック**と呼ばれる。`tabular` コンストラ

クタは両者をゼロ初期化されたテーブルとして構築するため、本節末尾の測定で行うように、どちらのネットワークの代わりにもテーブルを差し込める。

```
class ActorCritic(d2l.Policy):
    """A policy and a value function, each with its own optimizer."""
    def __init__(self, policy, value, lr=1e-2):
        super().__init__(policy, lr)
        self.value = value
        self.opt_v = nnx.Optimizer(value, d2l.shared_adam(lr), wrt=nnx.Param)

    def V(self, obs, value=None):
        """V(s) for a batch of states; `value` plays the part of `policy`
        in log_prob, for the update that differentiates it."""
        value = self.value if value is None else value
        return value(obs).squeeze(-1)

    def value_np(self, obs):
        """V(s) for a numpy batch of states, as numpy, through a forward
        pass on the batch padded to a bucket length."""
        obs_p, _ = d2l.padded(obs)
        return d2l.forward_np(self.value, obs_p)[:len(obs), 0]

    @classmethod
    def tabular(cls, num_states, num_actions, lr=0.1, seed=None):
        """Preference and value tables: two zero-initialized embeddings."""
        rngs = nnx.Rngs(d2l.get_key() if seed is None else seed)
        zeros = nnx.initializers.zeros_init()
        return cls(d2l.Embed(num_states, num_actions,
                             embedding_init=zeros, rngs=rngs),
                   d2l.Embed(num_states, 1, embedding_init=zeros, rngs=rngs),
                   lr)
```

コンストラクタ `mlp` はテーブルの代わりに、それぞれ `tanh_mlp` で構築したネットワークを与える。すなわち、`obs_dim` 個の入力と行動ごとに 1 つの選好を持つ方策ネットワークと、同じ形状で 1 つの出力を持つ価値ネットワークである。いずれも、この程度の小規模なタスクでよく使われる幅である 64 ユニットの隠れ層を 1 つ持ち、活性化関数にはオンポリシー手法の大規模な調査で方策ネットワークに最も有効であると報告された tanh を用いる (Andrychowicz et al. 2021)。14.7 節のソフトマックス $\pi_\theta(a \mid s) \propto \exp(\text{preference}_a)$ は、テーブルの行に対してと同様にネットワークの出力に作用する。スコアには隠れ層も含まれるようになり、自動微分が追加の連鎖律を適用する。

```
def tanh_mlp(in_dim, hidden, out_dim, *, rngs):
    """A network with one tanh hidden layer."""
    return nnx.Sequential(d2l.Linear(in_dim, hidden, rngs=rngs), jnp.tanh,
                          d2l.Linear(hidden, out_dim, rngs=rngs))

@d2l.add_to_class(ActorCritic)
@classmethod
def mlp(cls, obs_dim, num_actions, hidden=64, lr=1e-2, seed=None):
    """The same container with the tables replaced by one-hidden-layer nets."""
    rngs = nnx.Rngs(d2l.get_key() if seed is None else seed)
    return cls(tanh_mlp(obs_dim, hidden, num_actions, rngs=rngs),
```

```
                tanh_mlp(obs_dim, hidden, 1, rngs=rngs), lr)
```

各層には d2l.Linear（10.3 節）を用いる。これは nnx.Linear の一種であり、重みとバイアスを $[-1/\sqrt{n_{\text{in}}}, 1/\sqrt{n_{\text{in}}}]$（$n_{\text{in}}$ は入力数）から一様に抽出する。Flax のデフォルトは広がりが約 1.7 倍大きい lecun-normal の重みを抽出し、バイアスを 0 に設定する。本章と次章のすべてのネットワークでは、こうした小規模なネットワークをより安定して訓練できるため、この一様初期化を用いる。15.4 節の DQN を各初期化で 50 シードずつ実行したところ、一様初期化では実行の約 2/3 で CartPole を解いたのに対し、Flax のデフォルトでは約 1/4 にとどまった。

14.9.1.3 価値回帰

テーブルの場合、学習されたベースラインの更新は 1 つの要素を観測された 1 つのターゲットに向かって動かしていた。ネットワークの場合、それに対応するのは回帰（regression）、すなわち $\hat{V}(s_t)$ とバッチ全体のターゲットとの平均二乗誤差（mean squared error）に対するオプティマイザのステップである。fit_value は、固定されたターゲットベクトルに向かってこのようなステップを num_steps 回、価値ネットワーク ac.value（上記の ActorCritic が持つもの）上で実行する。

ベースラインにはバッチあたり何ステップ必要だろうか。以下の両タスクにおいてターゲットは大きさが 100 程度の割引収益であるのに対し、Adam の 1 ステップは各パラメータをおよそ学習率 0.01 だけ動かす。バッチあたり 1 ステップでは、ベースラインはターゲットに追いつかない。後述する振り子タスクでは、ターゲットの分散のうちベースラインが説明する割合 $1 - \mathrm{Var}(\hat{G} - \hat{V})/\mathrm{Var}(\hat{G})$ は、大半のシードで実行終了時までに 0.4 から 0.8 の間までしか上がらず、さらに低いままにとどまるシードもある。重み $\hat{G}_t - \hat{V}(s_t)$ の残りの部分は、本来なら良いベースラインが取り除くはずのノイズである。バッチあたり 40 ステップとすればこの割合は約 0.99 まで高まるため、以下の訓練ループでは 40 を用いる。

```
@nnx.jit
def _value_pass(ac, obs, target, mask):
    """One step on the mean squared error of V over the real rows."""
    loss, grads = nnx.value_and_grad(lambda value: (
        mask * (ac.V(obs, value) - target) ** 2).sum() / mask.sum())(ac.value)
    ac.opt_v.update(ac.value, grads)
    return loss

def fit_value(ac, obs, target, num_steps=1):
    """Regress the value head on a fixed target: eq_value_baseline for nets.
    The batch is padded to a bucket length, so the compiled pass is reused."""
    batch = d2l.padded(obs, np.float32(target))
    for _ in range(num_steps):
        loss = _value_pass(ac, *batch)
    return float(loss)
```

value_np は d2l.forward_np を通じて価値ネットワークを実行し、fit_value はコンパイル済みパス _value_pass を、d2l.padded でバケット長へパディングしたバッチ上で（policy_step と同様に）実行する。マスクによって損失は実際の行のみに制限される。これらの補助関数はオンライン付録「ユーティリティ関数とクラス」にある。

14.9.1.4 時間制限

本節の両タスクは時計によってエピソードを止める。CartPole は 500 ステップ後、Pendulum は 200 ステップ後である。14.1 節でその帰結を確認したとおり、時間制限で打ち切られたエピソードの報酬和はその収益ではない。過程は本来その後も続いていたはずだからである。残りの累積報酬はこの打ち切りを終端として扱ってしまう。上限までポールのバランスを保ち続けた CartPole のエピソードを考えてみよう。打ち切り付近では残りの累積報酬が 1 に向かって縮む一方、価値ネットワークには他の場面と同じくバランスのとれたポールが見えているため、エピソード序盤の直立状態とほぼ同じ価値を予測する。すると終盤ステップの重み $\hat{G}_t - \hat{V}(s_t)$ は強い負の値となり、更新はポールを立て続けていたまさにその行動の確率を引き下げてしまう。$\gamma = 0.99$、$T = 500$ とし、価値ネットワークが永遠にバランスを保ち続ける価値である $1/(1-\gamma) = 100$ を予測するとしよう。$t = 490$ における残りの累積報酬は $\hat{G}_t = (1 - 0.99^{10})/0.01 \approx 9.6$ であり、重みは -90 近くになる。演習問題 1 では、エピソード全体にわたってこの重みを計算する。

その補正は、打ち切りによって隠された将来の価値を足し戻すことである (Pardo et al. 2018)。これは時間制限をタスクの一部ではなく訓練上の便宜として扱うことを意味する。もし時間制限がタスクの一部であれば、残り時間を状態に含めるべきである（14.1 節）。あるエピソードがステップ $t = 0, \ldots, T-1$ を持ち、s_T を最後の行動が到達した状態とし、term $\in \{0, 1\}$ でその状態が終端状態かどうかを記録するとする。このとき、ステップ t のターゲットは次式となる。

$$y_t = \hat{G}_t + (1 - \text{term})\, \gamma^{T-t}\, \hat{V}(s_T). \tag{14.92}$$

終端したエピソードには何も加算されない。打ち切られたエピソードでは、記録が止まった状態からそのまま続いていたかのように、すべてのステップが記録停止時の状態の割引価値を受け取る。上の $t = 490$ の例では補正によって $0.99^{10} \cdot 100 \approx 90.4$ が加算されるため、$y_t \approx 100$ となって重みは約 0 に戻る。`Batch`（14.7 節）はこれに必要な情報をすでに記録している。すなわち、`next_obs` は各ステップに対する s_{t+1} を保持し、`term` は打ち切りではなく終端のみを記録する（14.1 節の規則）。`truncation_bootstrap` は、各エピソードの最終ステップでのみ $\gamma(1-\text{term})\hat{V}(s_T)$ を保持しそれ以外が 0 であるベクトルを構築し、逆方向スキャン $x_t + \gamma x_{t+1} + \gamma^2 x_{t+2} + \cdots$（14.8 節）をその上で実行することで、その価値に係数 γ^{T-t} を掛けて手前のすべてのステップへと伝播させる。

```
@d2l.add_to_class(d2l.Batch)
def truncation_bootstrap(self, gamma, V):
    """gamma^(T-t) (1 - term_T) V(s_T) for every step t of each episode:
    the discounted value of the future that a time-limit cut did not see.
    V maps a batch of numpy states to numpy values."""
    tail = np.zeros_like(self.rew)
    last = self.ep_ends - 1
    tail[last] = gamma * (1 - self.term[last]) * V(self.next_obs[last])
    return self.backward_scan(tail, gamma)
```

14.9.2 CartPole における REINFORCE

方策ネットワーク、価値ネットワーク、およびターゲットがそろったところで、訓練ループを組み立て、ポール付きの台車で実行してみよう。

14.9.2.1 訓練ループ

`train_reinforce` は、任意の方策オブジェクトに対して学習されたベースライン付きの REINFORCE を実行する関数であり、`make_agent(seed)` がエージェントを構築し、`env_name` が環境を指定する。各更新では `rollout` で 8 エピソードを収集し、ターゲット y_t を計算し、$y_t - \hat{V}(s_t)$ から計算した重みで方策ステップを 1 回とった後、40 回の回帰ステップで $\hat{V}$ をターゲットに当てはめる。ネットワークを用いる点以外にも、14.8 節の学習されたベースラインによる実行とは 4 つの違いがある。オプティマイザは素の SGD ではなく、各 `Policy` が保持する Adam である。重みは `d2l.standardize` を通してバッチ平均を引き、バッチ標準偏差で割る。ベースラインは要素ごとではなく回帰によって当てはめる。そして打ち切られたエピソードにはブートストラップを施す。

Adam のもとでは、すべての重みを 1 つの正の数で割ってもステップはほとんど変わらない。Adam は各座標をそのスケールの移動推定値で割るからである（演習問題 5 でこれを厳密に示す）。それでも正規化には 2 つの働きがある。中心化はバッチ自身から計算される第 2 のベースラインとして機能する。除算は、実行が進むにつれて収益が増大する間も重みのスケールを一定に保つ。このドリフトには、減衰の遅い Adam の移動平均では遅れてしか追従できないからである。

結果として得られる更新は、目的関数 $J(\theta)$ の勾配ではない。バッチ内のステップ数を N、正規化された重みを $\tilde{w}_t$ とすると、方策ステップは次の式に従う。

$$\hat{g} = \frac{1}{N} \sum_{t} \tilde{w}_t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t), \qquad \tilde{w} = \operatorname{standardize}\big(y - \hat{V}(s)\big). \tag{14.93}$$

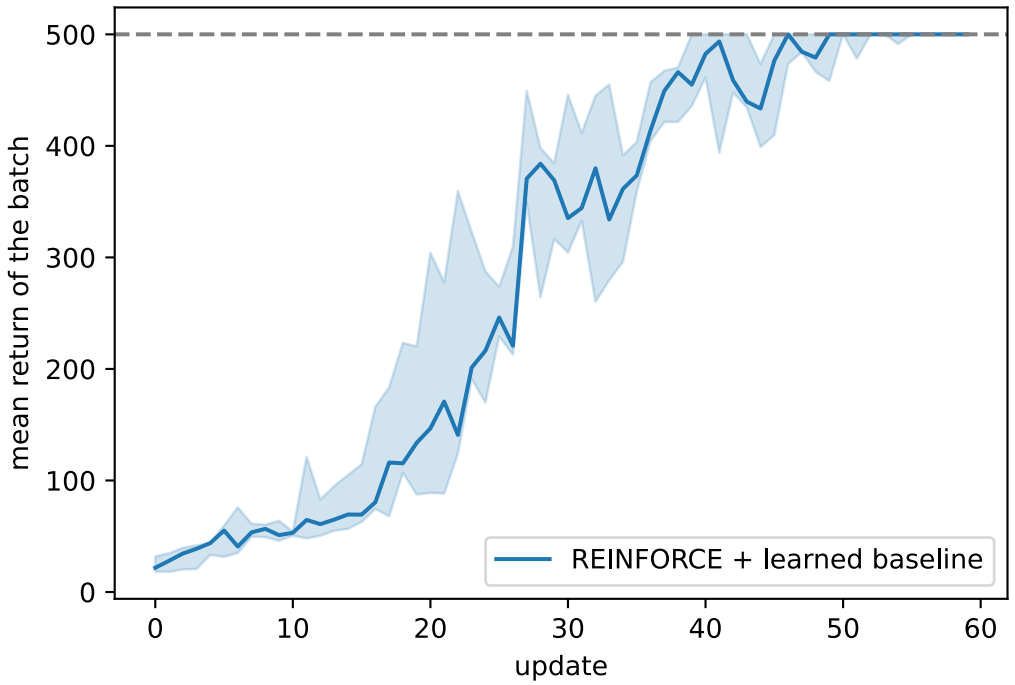
4 つの要因が $\hat{g}$ を $\nabla_\theta J$ から隔てている。因子 γ^t が落とされているため、終盤ステップのスコアも序盤ステップと同じ重みで扱われる。これは 14.8 節の残りの累積報酬推定量であり、一般にはいかなる目的関数の勾配でもない (Nota and Thomas 2020)。打ち切られたエピソードのターゲット y_t は $\hat{V}(s_T)$ を含むため、打ち切り時点におけるクリティックの誤差が統計的バイアスとして重みに入り込む。中心化にはバッチ自身の平均を用いるが、これはサンプリングされた行動にわずかに依存する。N およびバッチの標準偏差で割ることで各バッチはそのデータ自身によって再スケーリングされ、その後に Adam がほぼ一定の大きさのステップをとる。この組み合わせに直接適用できる確率近似（stochastic approximation）の定理は存在しないため、このループが機能するという根拠は収束証明ではなく実行結果から得られる。

```
def train_reinforce(seed, make_agent, env_name, gamma=0.99, num_updates=60,
                    batch_episodes=8, value_steps=40):
    """REINFORCE with a learned baseline for the agent `make_agent` builds."""
    rng, env = np.random.default_rng(seed), gym.make(env_name)
    ac = make_agent(seed)
    env.reset(seed=seed)
    for _ in range(num_updates):
        batch = d2l.rollout(env, ac.act, batch_episodes, rng)
        G = (batch.reward_to_go(gamma)
             + batch.truncation_bootstrap(gamma, ac.value_np))   # time limits
        w = d2l.standardize(G - ac.value_np(batch.obs))
        L = d2l.policy_step(ac, batch.obs, batch.act, w)
        fit_value(ac, batch.obs, G, num_steps=value_steps)  # fit V-hat well
        yield float(batch.episode_returns().mean()), L
```

14.9.2.2 CartPole

CartPole は古典的なポールバランシング問題である (Barto et al. 1983)。状態は冒頭で述べた 4 つの数値であり、2 つの行動は台車を左または右に押す。ポールが立っているステップごとに報酬 1 を得る。ポールが大きく傾きすぎるか台車がレールから外れるとエピソードは終端し、500 ステップに達すると打ち切られるため、達成可能な最大の収益は 500 である。湖での実行と比べると 3 つの設定が変わる。コンストラクタは `ActorCritic.mlp(4, 2)`（`Policy.tabular(16, 4)` の代わり）、環境は `'CartPole-v1'`、割引率は 0.99 であり、これによって実効ホライズン $1/(1-\gamma)$（14.1 節）は 20 ステップから 100 ステップに延びる。`d2l.run_seeds`（オンライン付録「ユーティリティ関数とクラス」）は訓練ジェネレータをシードごとに 1 回実行し、生成された値を積み重ねる。ここでは 3 つのシードを 60 回の更新にわたって訓練する。

```
def cartpole_agent(seed):
    return ActorCritic.mlp(4, 2, seed=seed)

runs = d2l.run_seeds(train_reinforce, 3, make_agent=cartpole_agent,
                     env_name='CartPole-v1')
d2l.plot_curves({'REINFORCE + learned baseline': runs[:, :, 0]},
                xlabel='update', ylabel='mean return of the batch',
                reference=500)
```



未訓練の方策はポールを約 20 ステップの間バランスさせる。どのシードでもバッチ平均は 50 回目の更新前後で 500 ステップの上限に到達し、個々のバッチがそれを下回ることはあるものの、おおむねその水準にとどまる。このタスクでは成功したエピソードのみが打ち切られるため、今回の実行では時間制限の補正を加えても平均収益はほとんど変わらなかったが、補正なしでは実行終盤に方策が上限から外れる頻度が増えた。

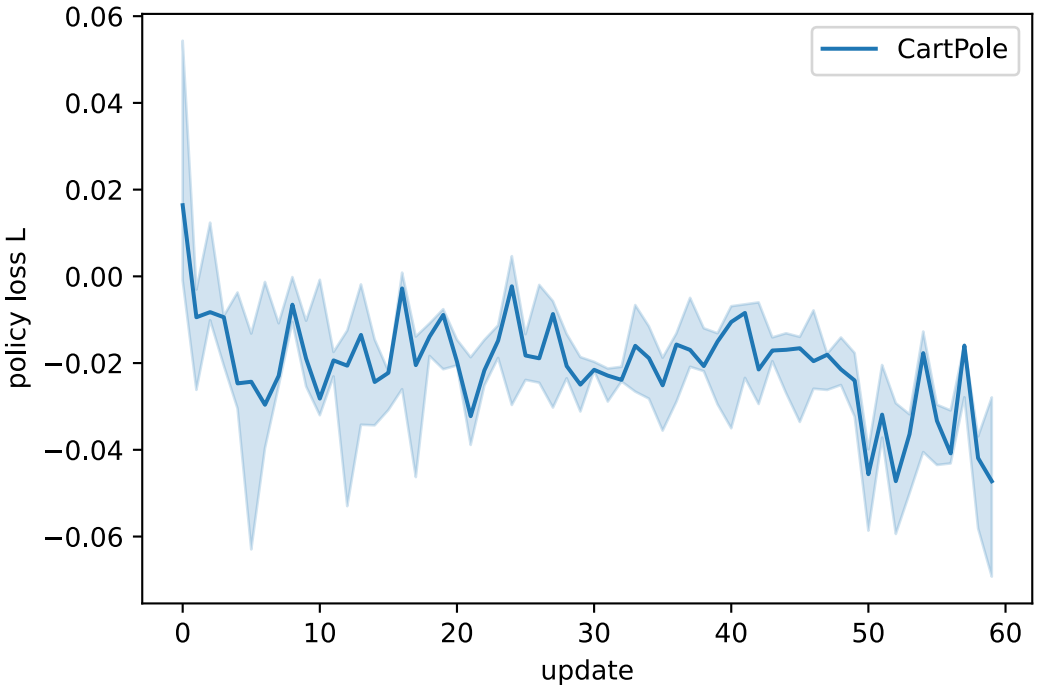
14.9.2.3 代理損失

`train_reinforce` は各更新でもう 1 つの数値、すなわち代理損失（surrogate loss）の値 `L` を返す。これは `policy_step` が最小化する量である（14.7 節）。深層学習フレームワークは更新をスカラー損失によって表現するため、$\hat{g}$ を明示的に組み立てる代わりに、`policy_step` は次式を最小化する。

$$L(\theta) = -\frac{1}{N}\sum_t \tilde{w}_t \log \pi_\theta(a_t \mid s_t), \tag{14.94}$$

ここで重み $\tilde{w}_t$ は固定されている。重みは NumPy 配列として渡され、計算グラフを持たない。L の勾配は $-\hat{g}$ であるため、L に対する降下ステップは $\hat{g}$ に沿ったステップとなる。しかし、L の値は方策の性能がどれほど良いかを推定するものではない。L は現在のパラメータにおいて正しい導関数を持つように作られているだけでそれ以上の意味はないため、L の減少が収益の増加を意味するとは限らない。CartPole の実行ではその両方を記録している。

```
d2l.plot_curves({'CartPole': runs[:, :, 1]}, xlabel='update',
                ylabel='policy loss L')
```



収益が 20 から 500 へと上昇する間も、L は大きさが 0.1 未満の小さな値にとどまり、初回以降のほぼすべての更新で負となる。正規化された重みは平均 0、標準偏差 1 を持つため、L はバッチ内の対数確率の標準偏差に重みとの相関を掛けて符号を反転させたものに等しい（演習問題 4）。正の重みを持つ行動がすでに平均を上回る確率を持っているときにこの値は負になり、その大きさは収益について何も語らない。訓練の進行を追うには、収益と、方策がどれだけ動いたかのような更新自体を表す量を観察すべきである。

14.9.3 連続行動

ソフトマックスには有限個の行動リストが必要である。しかし多くの制御問題では、トルクや操舵角のように実数値の行動を扱う。14.7 節におけるスコア関数（score function）の導出では、行動集合の有限性を一度も使っておらず、必要なのは $\log \pi_\theta(a \mid s)$ が θ について微分可能であることだけである。したがって、実数値の行動上の確率密度がこの性質を持つ方策であれば、まったく同じループで訓練できる。

14.9.3.1 ガウス方策

14.6 節の振り子（`Pendulum-v1`）に戻ろう。振り子はモーター付きの支点から吊り下げられており、行動はトルク $u \in [-2, 2]$ である。直立位置からの振り子の角度を ϑ と書き、θ は引き続き方策パラメータを表すものとする。14.6 節のプランナは運動方程式を知っており、それを微分していた。方策勾配エージェントは運動方程式を一切知らず、目にするのは観測 $(\cos\vartheta, \sin\vartheta, \dot\vartheta)$ と、各ステップにおける報酬 $-(\vartheta^2 + 0.1\,\dot\vartheta^2 + 0.001\,u^2)$（ただし ϑ は $[-\pi, \pi]$ に折り返される）のみである。したがって、振り子が真上で静止しているときにコストが最小となる。各エピソードはランダムな角度から始まり、200 ステップ後に打ち切られる。制御しない振り子が得る収益は -1200 付近であり、振り子を振り上げてバランスを保つ制御器は約 -200 以上の収益を得る。

ここでは、ネットワークが平均行動を出力するガウス方策（Gaussian policy）$\pi_\theta(a \mid s) = \mathcal{N}\big(a; \mu_\theta(s), \sigma^2\big)$ を用いる。`GaussianHead` がそのネットワークである。**トランク**が隠れ特徴を計算し、デフォルトでは 1 層の tanh 層であるが、`trunk` 引数によって `hidden` 個の出力を持つ任意のモジュールに置き換えられる。線形層がその特徴を平均 $\mu_\theta(s)$ に写像し、`zero_mean=True` を指定するとその重みとバイアスは 0 から始まる（以下の Pendulum の実行ではこれを用いる）。対数標準偏差 $\log\sigma$ はデフォルトでは 1 つの学習されるベクトルであり、したがって σ はどの状態でも同じになる。`state_std=True` を指定すると、特徴上の第 2 の線形層（$[-5, 2]$ にクランプされる）となり、σ が状態に依存するようになる。演習問題 2 ではこのオプションを用いる。$\log\sigma$ は方策モジュールに属するため、方策ステップは平均と合わせてこれも更新する。

```
class GaussianHead(nnx.Module):
    """A Gaussian over actions: a trunk of hidden features, a linear mean,
    and a log standard deviation that is one learned vector or, with
    `state_std=True`, a second linear head clamped to [-5, 2]. The default
    trunk is one tanh layer; any module with `hidden` outputs will do.
    `zero_mean=True` sets the mean layer to zero, so that mu = 0 at first."""
    def __init__(self, obs_dim, act_dim, hidden, state_std=False, trunk=None,
                 zero_mean=False, *, rngs):
        self.trunk = trunk if trunk is not None else nnx.Sequential(
            d2l.Linear(obs_dim, hidden, rngs=rngs), jnp.tanh)
        self.mean = d2l.Linear(hidden, act_dim, rngs=rngs)
        if zero_mean:
            self.mean.kernel[...] = 0.0
            self.mean.bias[...] = 0.0
        self.state_std = state_std
        self.log_std = (d2l.Linear(hidden, act_dim, rngs=rngs) if state_std
                        else nnx.Param(jnp.zeros(act_dim)))

    def __call__(self, obs):
        h = self.trunk(obs)
        if self.state_std:
            return self.mean(h), jnp.exp(jnp.clip(self.log_std(h), -5, 2))
```

```
        return self.mean(h), jnp.exp(self.log_std[...])
```

GaussianPolicy は ActorCritic を継承し、GaussianHead と tanh_mlp による価値ネットワーク（mlp と同様のもの）を構築したうえで、分布に依存する 3 つのメソッド log_prob、act、act_greedy をオーバーライドする。rollout、policy_step、fit_value、train_reinforce はこのインターフェースのみを用いるため、一切の変更を必要としない。

```
class GaussianPolicy(ActorCritic):
    """The same interface over a Normal instead of a softmax; nothing that
    consumes the interface changes. `head_args` go to GaussianHead."""
    def __init__(self, obs_dim, act_dim, hidden=64, lr=1e-2, seed=None,
                 **head_args):
        rngs = nnx.Rngs(d2l.get_key() if seed is None else seed)
        super().__init__(GaussianHead(obs_dim, act_dim, hidden, **head_args,
                                      rngs=rngs),
                         tanh_mlp(obs_dim, hidden, 1, rngs=rngs), lr)

    def log_prob(self, obs, act, policy=None):
        """log pi(a|s): the Normal density, summed over action dimensions."""
        mean, std = (self.policy if policy is None else policy)(obs)
        return jax.scipy.stats.norm.logpdf(act, mean, std).sum(-1)

    def act(self, obs, rng):
        """Sample mu(s) + sigma(s) z, z ~ N(0, I) from the numpy rng."""
        mean, std = d2l.forward_np(self.policy, obs)   # compiled, as in tabular
        return mean + std * rng.standard_normal(mean.shape, dtype=np.float32)

    def act_greedy(self, obs, rng=None):
        """The mean action mu(s)."""
        return d2l.forward_np(self.policy, obs)[0]
```

環境はサンプリングされた各トルクを $[-2, 2]$ にクリップするため、act は非有界なガウス分布からのサンプルをそのまま返してよい。log_prob はクリップ前のサンプルをスコア付けするが、演習問題 2 で推定量が不偏に保たれる理由を示す。更新は log_prob をその勾配（スコア）を通じてのみ用いる。$z = (a - \mu)/\sigma$ とおくと、ガウス分布のスコアは次式となる。

$$\frac{\partial \log \pi}{\partial \mu} = \frac{z}{\sigma}, \qquad \frac{\partial \log \pi}{\partial \log \sigma} = z^2 - 1. \tag{14.95}$$

正の重みのもとでは、平均を上回るサンプルは μ を引き上げ、どちらの方向であれ平均から離れたサンプルは σ を引き上げ、平均に近いサンプルは σ を引き下げる。log_prob をガウス密度の式と、またその勾配をこれら 2 つの式と、$\sigma = 0.5$ およびランダムな 5 個の状態と行動において照合してみよう。平均層のバイアスは μ に導関数 1 で入るため、その勾配は μ に関するスコアを 5 個のサンプルにわたって合計したものになる。

```
sigma = 0.5
gp = GaussianPolicy(3, 1, seed=1)
gp.policy.log_std[...] = jnp.full(1, np.log(sigma))
rng = np.random.default_rng(1)
obs = rng.standard_normal((5, 3)).astype(np.float32)
```

```
act = rng.standard_normal((5, 1)).astype(np.float32)
z = (act - np.stack([gp.act_greedy(o) for o in obs])) / sigma
hand = -0.5 * z ** 2 - np.log(sigma) - 0.5 * np.log(2 * np.pi)
print('density:', np.allclose(gp.log_prob_np(obs, act), hand.sum(-1), atol=1e-5))
g = nnx.grad(lambda p: gp.log_prob(obs, act, p).sum())(gp.policy)
g_mu, g_ls = np.asarray(g.mean.bias[...]), np.asarray(g.log_std[...])
print('score in mu:', np.allclose(g_mu, (z / sigma).sum(0), atol=1e-4),
      '| score in log sigma:', np.allclose(g_ls, (z ** 2 - 1).sum(0), atol=1e-4))
```

```
density: True
score in mu: True | score in log sigma: True
```

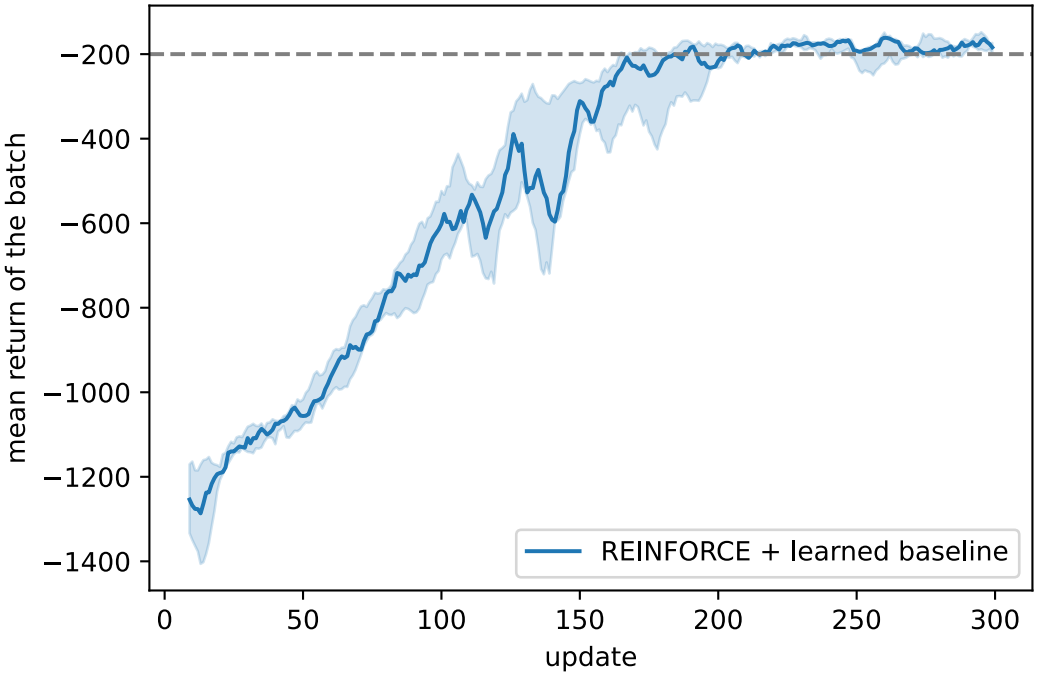
3 つの確認はいずれも True を出力する。すなわち、log_prob はガウス密度であり、その勾配は更新が用いるスコアとなっている。

14.9.3.2 Pendulum

曲線には割引なしの 200 ステップ収益を表示するが、更新では実効ホライズン 20 ステップに相当する $\gamma = 0.95$ で割引を行う。ここでの割引率はタスクの一部ではなく設計上の選択である。トルクは 100 ステップ後のコストよりも直後の数十ステップにはるかに強く影響し、ホライズンを短くすることで、報告される目的関数に対する多少の統計的バイアスと引き換えに分散の小さいターゲットが得られる。今回の実行では $\gamma = 0.9$ でもほぼ同等に訓練できた。

pendulum_agent は平均層の重みとバイアスを 0 に設定し（zero_mean=True）、他の層の乱数抽出にかかわらずすべてのシードが $\mu = 0$、$\sigma = 1$ から始まるようにする。平均が $[-2, 2]$ の外にあると学習は停滞してしまう。環境がほぼすべてのサンプルを同じ境界値にクリップするため、どのサンプルが引かれたかに報酬が依存しなくなり、行動に関する情報をまったく持たない重みがスコアに掛けられることになるからである。ここで用いる層の初期化では初期平均は境界の十分内側に収まるが、より大きな重みを持つ初期化では一部のシードで外に出てしまい、そうしたシードは停滞する。方策の最終層の重みを非常に小さくするか 0 にすることは、オンポリシー手法における標準的な推奨事項である (Andrychowicz et al. 2021)。方策はバランスを保つ前にまず振り子を振り上げることを学習しなければならないため、8 エピソードからなる更新を 300 回実行して訓練する。

```
def pendulum_agent(seed):   # every seed starts at mu = 0
    return GaussianPolicy(3, 1, seed=seed, zero_mean=True)

runs_p = d2l.run_seeds(train_reinforce, 3, make_agent=pendulum_agent,
                       env_name='Pendulum-v1', gamma=0.95, num_updates=300)
d2l.plot_curves({'REINFORCE + learned baseline': runs_p[:, :, 0]},
                xlabel='update', ylabel='mean return of the batch',
                reference=-200, smooth=10)
```



収益は -1200 付近から始まる。すべての実行が振り子を振り上げてバランスを保つことを学習する。10 回の更新にわたる平均は、約 150〜300 回の更新（すなわち数十万環境ステップ）の後に -200 を上回り、最良値はおおむね -150 から -200 の間に収まる。その後の更新もほとんどはその水準付近にとどまるが、数回の更新にわたって数百ほど後退することもある。実行全体を通じて σ は初期値の約半分まで低下するが、これは良い行動の近傍で演習問題 2 が予測する方向と一致する。ここでは時間制限の補正が重要になる。Pendulum のエピソードはすべて打ち切られるため、補正なしでは同じ予算内で -200 に到達したシードは今回の実行では約半数にとどまった。

これほど小さなタスクに対して、数十万ステップというのは膨大な経験量である。1,600 ステップからなる各バッチは 1 回の更新に使われただけで破棄され、重みは各行動の価値の学習された推定値ではなくエピソード全体の収益から計算されている。ソフトアクタークリティック（soft actor-critic、15.5 節）は、同じタスクにおいて約 5,000〜11,000 ステップ以内に -200 へ到達する。1 サンプルからなぜそれほど多くを学習できるのかという問いを、第 15 章で取り上げる。

14.9.3.3 スコア関数勾配とパスワイズ勾配

連続行動では、期待報酬（expected reward）を微分する第 2 の方法も可能になる。単一の状態、ガウス方策 $a \sim \mathcal{N}(\mu, \sigma^2)$、および既知の微分可能な報酬 $Q(a)$ を考えよう。$\mathbb{E}[Q(a)]$ の μ に関する勾配には、互いに等価な 2 つの形式がある。

$$\nabla_\mu \mathbb{E}[Q(a)] = \mathbb{E}\left[Q(a)\,\frac{a-\mu}{\sigma^2}\right] = \mathbb{E}[Q'(\mu+\sigma z)], \qquad z \sim \mathcal{N}(0,1). \tag{14.96}$$

1 つ目の等号は、本章を通じて用いてきたスコア関数の恒等式である。2 つ目の等号は $a = \mu + \sigma z$ を代入することで μ を分布から Q の内部へと移し、サンプルを通して微分する。これは**パスワイズ**勾配、すなわち再パラメータ化（reparameterization）勾配であり (Kingma and Welling

2014)、Figure 14.21 に各推定量の勾配がどこを流れるかを示す。既知の Q のもとでは両者とも不偏である。$Q(a) = -(a-1)^2$ において $\mu = 0$、$\sigma = 0.5$ としたとき（厳密な勾配は 2)、両者のノイズを比較しよう。

```
def Q(a):                       # a stand-in critic, differentiable in the action
    return -(a - 1.0) ** 2

sigma, N = 0.5, 100000
a = sigma * np.random.default_rng(2).standard_normal(N, dtype=np.float32)
g_score = Q(a) * a / sigma ** 2               # score-function samples, mu = 0
g_path = np.asarray(jax.vmap(jax.grad(Q))(jnp.asarray(a)))  # per-sample dQ/da
print(f'score:    mean {g_score.mean():.2f}, variance {g_score.var():.1f}')
print(f'pathwise: mean {g_path.mean():.2f}, variance {g_path.var():.2f}')
print(f'score variance if Q gains a constant +10: '
      f'{((Q(a) + 10) * a / sigma ** 2).var():.0f}; pathwise is unchanged')
```

```
score:    mean 1.98, variance 21.5
pathwise: mean 2.00, variance 1.00
score variance if Q gains a constant +10: 281; pathwise is unchanged
```

どちらの標本平均も 2 に近い。1 サンプルあたりの厳密な分散はスコア関数で 21.75、パスワイズ推定量で 1.0 と約 20 倍の開きがあり、さらに定数 10 を Q に加えるとスコア関数の分散は 281.75 に跳ね上がる。Q がスコアに掛け合わされるためである。パスワイズ推定量は Q' のみに依存するため、定数を加えても変化しない。演習問題 6 でこれらの分散を導出するが、その比は Q、σ、およびオフセットに依存する。低い分散の代償は、行動の微分可能な関数として Q にアクセスする必要があることである。実際には Q は学習されたクリティック Q_w であり、その行動勾配 $\partial Q_w/\partial a$ の誤差が方策勾配の統計的バイアスとなる。

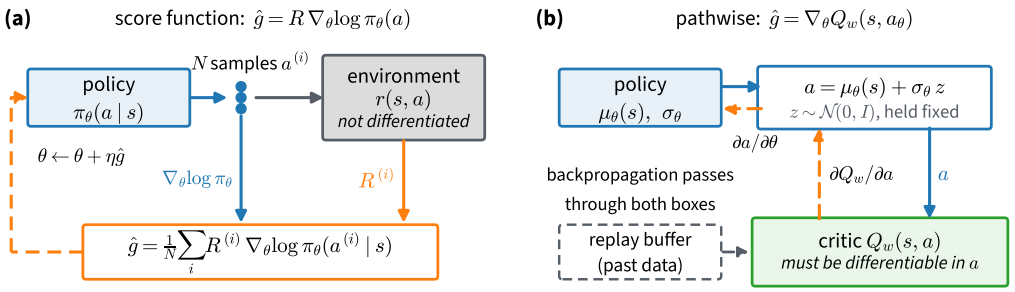


Figure 14.21: 同じ勾配に対する 2 つの推定量。(a) スコア関数推定量は返された報酬を通じてのみ環境を用いるため、環境は微分可能である必要がない。(b) パスワイズ推定量はサンプルを $a = \mu_\theta(s) + \sigma_\theta z$（ここでは単一の学習される σ_θ を持つ）と書き、$\partial Q_w/\partial a$ と $\partial a/\partial\theta$ を通して微分する。クリティック Q_w は行動について微分可能でなければならない。これはリプレイされたデータ上の回帰によって訓練でき、$\partial Q_w/\partial a$ における誤差は勾配の統計的バイアスとなる。

パスワイズ形式はまた、連続制御が行動に関する最大化を回避する手段でもある。価値ベースの手法は $\max_a Q(s,a)$ を計算して行動を選ぶが、これは湖の上では 4 つの行動からの参照で済むものの、$a \in \mathbb{R}^d$ に対しては毎ステップの最適化問題となる。決定論的方策勾配（deterministic policy gradient）は代わりに、学習されたクリティックを通じた $\partial Q_w/\partial a$ に従って、価値の高い行動を出力するようアクターネットワークを訓練する (Silver et al. 2014)。

14.9.4 状態間の汎化

冒頭では、似た状態に似た値を与える関数を求めた。Equation 14.91 によれば、1 回の勾配ステップはカーネル k に従って状態間に広がる。ここでは、CartPole の訓練に用いたものと同じ形状の、新規構築した価値ネットワークについてその広がりを測定する。

14.9.4.1 状態間の結合のプローブ

このプローブは、$\hat{V}$ を $[-1, 1]^4$ 内の 256 個のランダムな状態のうちの 1 つにおいて、現在の推定値より 1 単位高いターゲットへ向かう 1 回の勾配ステップによって引き上げ、全 256 個の状態における変化を記録する。続いて、16 要素のテーブルに対しても同じ操作を行う。各プローブでは Adam オプティマイザ `opt_v` を素の SGD（`d2l.plain_sgd`、オンライン付録「ユーティリティ関数とクラス」）に置き換え、この 1 ステップが Equation 14.91 に従うようにする。Adam のままでは、各パラメータがその移動スケールによって再スケーリングされてしまうからである。`fit_value` は二乗誤差の半分と比べて勾配が 2 倍になる平均二乗誤差を最小化するため、テーブルにおけるステップサイズ 0.25 はその要素を 0.5 だけ動かす。次のコードセルでは、残りの 255 個のプローブ状態を動かした状態からの距離順に並べ替える。

```
net_probe = ActorCritic.mlp(4, 2, seed=0)
tab_probe = ActorCritic.tabular(16, 4, seed=0)
for ac, lr in ((net_probe, 0.02), (tab_probe, 0.25)):  # one plain step
    ac.opt_v = d2l.plain_sgd(ac.value, lr)
S = np.random.default_rng(0).uniform(-1, 1, (256, 4)).astype(np.float32)
before = net_probe.value_np(S)
fit_value(net_probe, S[:1], before[:1] + 1.0)      # raise ONE state
dV = net_probe.value_np(S) - before
order = 1 + np.argsort(np.linalg.norm(S[1:] - S[0], axis=1))  # nearest first
before_t = tab_probe.value_np(np.arange(16))
fit_value(tab_probe, np.arange(1), before_t[:1] + 1.0)
dV_t = tab_probe.value_np(np.arange(16)) - before_t
print(f'network: nudged state {dV[0]:+.2f}; 10 nearest probes '
      f'{dV[order[:10]].mean():+.2f}, 10 farthest {dV[order[-10:]].mean():+.2f}; '
      f'{(dV[1:] < 0).sum()} of 255 moved down')
print(f'table:   nudged entry {dV_t[0]:+.2f}; '
      f'largest move among the other fifteen: {np.abs(dV_t[1:]).max():.6f}')
```

```
network: nudged state +0.55; 10 nearest probes +0.49, 10 farthest -0.01; 6 of 255 moved down
table:   nudged entry +0.50; largest move among the other fifteen: 0.000000
```

動かした状態は上昇し、それに最も近い 10 個のプローブもその変化量の 80% 超というほぼ同等の幅で上昇する。変化量は距離とともに減衰し、最も遠いプローブでは 0 付近かわずかに負となり、逆方向へ動くプローブも現れる。一方、テーブルは動かした要素を $+0.50$ 動かすだけで、他の要素は一切動かさない。最も遠いプローブがどれだけ下がるか、また何個のプローブが下がるかはネットワークのランダム初期化（random initialization）に依存するが、全体の形状は変わらない。Figure 14.22 は 1 次元の状態上で同じ対比を描いている。

正の隆起は汎化（generalization）である。1 つの状態での経験が似た状態における推定値に情報を伝えるため、ネットワークは状態空間を人手で分割しなくても CartPole を学習できる。負の遠方領域は干渉（interference）である。状態空間のある領域に当てはめると、バッチに含まれない領域の推定値まで変化してしまう。バランスをうまく保つ CartPole の方策はほぼ直立した状態のバッチばかりを収集するため、それに当てはめるとバッチが一度も訪問しない大きなポール角での推定値が動いてしまう（演習問題 7 でこれを詳しく追う）。こうした変化により、各更新がそのバッチ自体にはよく適合していても、訓練曲線が一時的に落ち込むことがある。

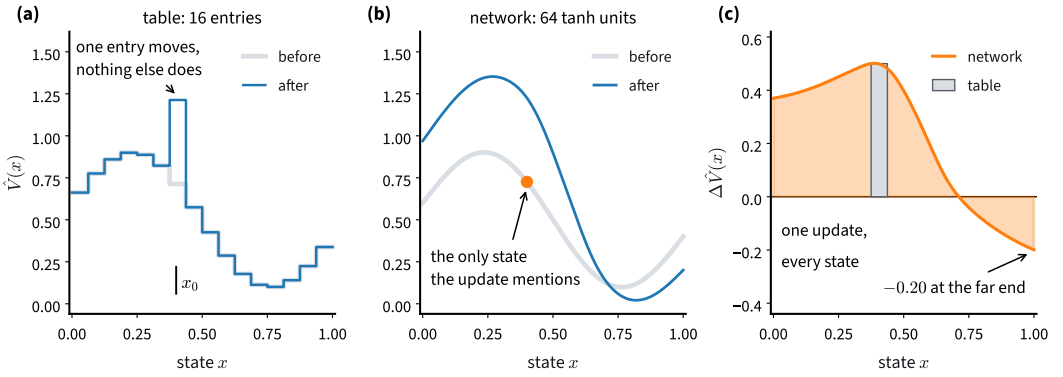


Figure 14.22: 1 回の価値更新が影響を及ぼす範囲。1 次元の状態上の価値推定を 16 要素のテーブルと 64 個の tanh ユニットを持つネットワークとして当てはめ、それぞれ印を付けた状態 x_0 において現在の推定値より 1 単位高いターゲットへ向けて 1 回の勾配ステップをとる（ステップサイズ η_V はいずれも x_0 が $+0.50$ だけ動くように選ぶ）。(a) テーブルは 1 つの要素のみを動かし、他は一切動かさない。(b) ネットワークはあらゆる状態の推定値が x_0 の推定値とパラメータを共有しているため、曲線全体を動かす。(c) 2 つの変化量を並べた比較。一次近似では、ネットワークの変化量は $\eta_V \left(y - \hat{V}(x_0)\right) k(x, x_0)$（$y$ はターゲット）であり、ここで k は x と x_0 におけるパラメータ勾配の内積である。すなわち、x_0 の周囲に正の隆起が生じ、$x \approx 0.7$ を超えると負に転じて遠端では -0.20 に達する。

14.9.4.2 モンテカルロエージェントの限界

本節の価値ネットワークは、ターゲットに自身の推定値 $\hat{V}(s_T)$ が含まれる時間制限の打ち切り時点を除き、観測された収益に当てはめられる。打ち切りから離れた箇所では、その当てはめは通常の回帰である。毎ステップで**ブートストラップ**し、次の状態における推定値をすべてのターゲットに組み込む手法では、この性質が失われる。ターゲットが更新のたびに動き、更新はもはや固定された目的関数の勾配ではなくなり、非線形近似器や別の方策からのデータを用いると反復が発散し得る (Tsitsiklis and Van Roy 1997)。

ここまでで、REINFORCE 推定量、当てはめたベースライン、および時間制限の補正を用いて、連続状態と連続行動の上で方策ネットワークを訓練できるようになった。それでもこのエージェントにはまだ 3 つの限界がある。ターゲットがエピソード全体の収益であるため、すべてのエピソードが終わるまで待たなければならない（15.1 節ではより早く学習するためにブートストラップを行う）。各バッチを 1 ステップだけ使って破棄してしまう（15.2 節では制御された数ステップにわたってバッチを再利用し、15.4 節では過去の遷移をリプレイする）。そして 1 回の更新が方策をどれだけ動かすかを制限していないが、大きく動かしすぎると現在の収益も次に収集されるデータも損なわれ得る。第 15 章ではこれら 3 つすべてを取り上げ、あわせてここでの数十万ステップとソフトアクタークリティックの 1 万ステップとの間にある隔たりを埋めていく。

14.9.5 手法の見取り図

本章と次章では多くの手法を扱う。それぞれの手法は、何を学習するのか、そしてどのデータで更新を駆動できるのかという 2 つの問いによって位置づけられる。Figure 14.23 はこの 2 つの問いに沿って各手法を配置したものである。

Table 14.3 に、両章のアルゴリズムを、それぞれが何を推定するか、どのデータを利用できるか、およびどこで導入されるかとともに一覧にする。

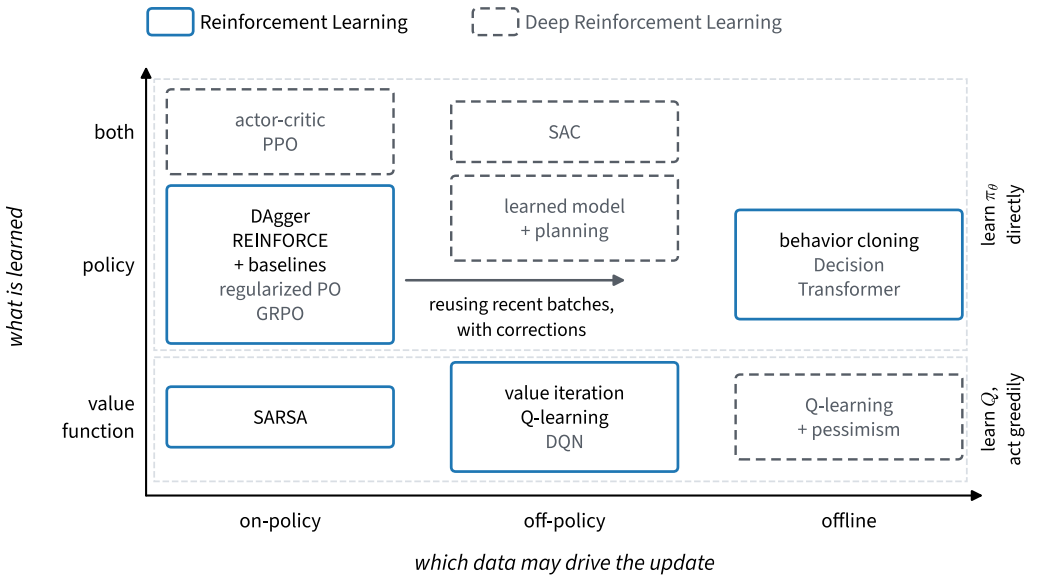


Figure 14.23: 何を学習するか、およびどのデータで更新を駆動できるかによって整理した、本章と第 15 章の各手法。実線の枠は本章の手法、破線の枠と灰色の名称は第 15 章の手法を表す。行動クローニングは固定されたデモンストレーションの集合から学習するのに対し、DAgger は学習者自身の状態にエキスパートがラベルを付け直したものから学習するため、オンポリシーの列に置かれる。価値反復法は特定の方策からのデータではなく既知のカーネルを更新に用いるため、オフポリシーの列に置かれる。オンポリシーの方策枠から出る矢印は直近バッチの再利用を表しており、収集時点から方策が変化しているため補正が必要になる。オフポリシーの列にある学習されたモデルはエージェント自身の蓄積された経験に当てはめられ、エージェントはその内部で計画を行う（15.6 節）。

Table 14.3: これら 2 章のアルゴリズムの見取り図。「〜で言及」は本書で説明するが実装はしない手法を示す。

アルゴリズム	推定対象	更新を駆動できるデータ	掲載箇所
価値反復法、方策反復法	ベルマン作用素のスイープによる厳密な V^*、方策評価による V^π	なし（カーネル P と報酬 r を使用）	14.2 節
行動クローニング	デモンストレーション上の交差エントロピーによる $\pi(a \mid s)$	固定されたエキスパートデータセット	14.5 節
DAgger	上と同じ当てはめ	必要に応じてエキスパートが再ラベル付けした学習者自身の状態	14.5 節
LQR、iLQR、移動ホライズン MPC	リッカチ漸化式による厳密な二次残りコスト $V_t(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_t \mathbf{x}$ と線形フィードバック $\mathbf{u}_t^* = -\mathbf{K}_t \mathbf{x}_t$	なし（既知のダイナミクス $f(\mathbf{x}, \mathbf{u})$ と二次コストを微分）	14.6 節
Q 学習（表形式）	サンプルによるバックアップからの Q^*	任意の行動方策の遷移（オフポリシー）	14.3 節
SARSA	探索を含めた行動方策自身の Q	自身の遷移（オンポリシー）	14.3 節
確実性等価計画	サンプルから集計したモデルの Q^*	任意の行動方策の遷移（カウント集計）	14.3 節
UCB、トンプソンサンプリング	腕の平均とその不確実性	自身の試行（バンディット）	14.4 節
REINFORCE	スコア関数による $\nabla_\theta J$	現在の方策からの新規の軌跡のみ	14.7 節
ベースライン付き REINFORCE、RLOO	より低分散な同じ勾配	現在の方策からの新規の軌跡	14.8 節
GRPO	グループ正規化されたアドバンテージ（価値ネットワークなし）	プロンプトごとの新規グループ（各グループを数回のクリップ付きステップで再利用）	重みは 14.8 節、機構は 15.8 節
ネットワーク上の学習されたベースライン付き REINFORCE	ネットワークとしての π_θ と $\widehat{V}$	現在の方策からの新規の軌跡	14.9 節
アクター・クリティック（A2C）	π_θ とブートストラップされたクリティック	新規の（現在に近い）軌跡	15.1 節
PPO	クリップされた確率比のもとでの π_θ	数ステップ再利用される 1 バッチ	15.2 節
KL 正則化方策最適化（RLHF の背後にある目的関数）	報酬によって参照方策から傾斜させた方策	学習された報酬でスコア付けされたロールアウト	15.3 節（大規模な RLHF はより詳しく扱う「言語モデル」パートに委ねる）
DQN、Double DQN	ネットワークとしての Q	過去の経験のリプレイバッファ	15.4 節
Rainbow	DQN とその検証済み構成要素	リプレイバッファ	15.9 節で言及
DDPG、TD3	クリティックとそれを最大化するよう訓練された決定論的アクター	リプレイバッファ（オフポリシー）	15.5 節で言及

アルゴリズム	推定対象	更新を駆動できるデータ	掲載箇所
SAC	ツインソフトクリティックと、報酬＋エントロピーを最大化する確率的スクワッシュアクター	リプレイバッファ（オフポリシー）	15.5 節
学習されたモデル内でのモデル予測制御、方策へ蒸留された探索	回帰で当てはめたダイナミクスと報酬のモデル、およびその内部で探索された計画	モデルへ再当てはめされるエージェント自身の蓄積経験	15.6 節
Dyna、MBPO	オフポリシー学習器のために追加の遷移を想像するのに用いる同じモデル	リプレイバッファ＋モデルロールアウト	15.6 節で言及
悲観主義を伴うオフライン Q 学習、CQL	データが薄い領域にペナルティを課した Q	固定されたログデータセット（相互作用は一切なし）	15.7 節
Decision Transformer	収益条件付き系列モデル	固定されたログデータセット	15.7 節で言及
MuZero、Dreamer	計画や想像を行うための学習された環境モデル	モデルを通じてリプレイされる自身の相互作用	15.9 節で言及
DPO	選好から直接求める正則化最適解	固定された選好データセット	15.8 節（大規模な選好最適化はより詳しく扱う「言語モデル」パートに委ねる）

14.9.6 まとめ

連続状態上のテーブルでは訪問のたびに行が必要になるため、方策と価値推定はネットワークへと置き換わる。テーブルとは、ワンホット状態上の線形ネットワークという特別な場合にほかならない。方策勾配の導出に必要なのはスコアのみであるため、学習されたベースライン付きの REINFORCE は 3 つの実践的な追加要素を伴ってそのままネットワークへ引き継がれる。すなわち、ベースラインがターゲットに追従できるよう十分な回数の回帰ステップで当てはめること、重みを正規化すること、そして時間制限で打ち切られたエピソードを最終状態の価値でブートストラップすることである。このループが実行する更新は J の不偏勾配ではなくステップごとの代理損失に基づくものであり、その損失の値は性能について何も語らない。ガウス方策はこのループを連続トルクへと拡張し、それによって Pendulum のすべての実行が数十万ステップの後に振り子を振り上げてバランスを保つようになる。1 回の更新はあらゆる状態における推定値を動かし、似た状態を引き上げる一方で離れた状態を引き下げる。このエージェントはエピソード全体の終了を待ち、各バッチを 1 度だけ使い、ステップ幅も制限しない。まさにそこから第 15 章が始まる。

14.9.7 演習問題

1. **時間制限における割引**。`train_reinforce` は各バッチ内のエピソードの割引なし平均収益を返す一方、更新では各ステップを $\gamma = 0.99$ での残りの累積報酬によって重み付けする。CartPole の各ステップは報酬 1 を獲得し、環境は 500 ステップに達したエピソードをすべて打ち切る。
    1. 長さ T のエピソードの割引収益が $(1-\gamma^T)/(1-\gamma)$ であることを示し、$T \in \{20, 100, 200, 500\}$ に対してその値を求めよ。これより、割引なし収益と割引収益がエピソードを同じ順序に順位付けすることを結論づけよ。エピソードが 200 ステップから 500 ステップに延びたとき、それぞれは何倍に増えるか。

2. 時間制限の補正なしの場合について、上限に達したエピソードの残りの累積報酬 $\hat{G}_t$ を $t \in \{0, 400, 450, 490\}$ において計算せよ。価値ネットワークは状態のみを入力とし、成功したエピソードの終盤の状態は序盤の状態とよく似ている。成功したエピソードの最終付近のステップの重みはどちらの符号をとりやすく、方策の更新はそれらのステップでとられた行動にどのような影響を与えるか。
3. `train_reinforce` はこれらのターゲットに `batch.truncation_bootstrap(gamma, ac.value_np)` を加算する。ポールが立っているすべての状態で $\hat{V}$ が $1/(1-\gamma)$ を予測すると仮定せよ。このとき、打ち切られたエピソードのすべてのステップにおけるターゲット y_t が、上限までの残りステップ数によらず $1/(1-\gamma)$ となることを示せ。ポールが倒れて終わるエピソードのターゲットが依然として終端に向かって縮むのはなぜか。また、それが正しいのはなぜか。
4. Pendulum のエピソードはすべて打ち切られるのに対し、CartPole では成功したエピソードのみが打ち切られる。この事実を用いて、補正を取り除くと CartPole よりも Pendulum の学習がはるかに遅くなる理由を説明せよ。

2. [code] **ガウス分布のスコア**。`GaussianHead` は 1 次元の行動を平均 μ と対数標準偏差 $\ell = \log\sigma$（コード上の `log_std`）によってパラメータ化する。$z = (a-\mu)/\sigma$（a はサンプリングされた行動）とおく。

   1. $\partial \log\pi/\partial\mu = z/\sigma$ および $\partial\log\pi/\partial\ell = z^2 - 1$ であること、さらにその両方が方策のもとで平均 0 を持つことを示せ。
   2. Q を行動の二次関数とし、その二階導関数を Q'' とする。$\mathbb{E}\big[Q(a)\,(z^2-1)\big] = \sigma^2 Q''$ となることを、$\mathbb{E}[z^4] = 3$ を用いて示せ。方策勾配の期待ステップは σ を、Q の最大値の近傍においてどちらの方向へ動かすか。このことは、平均が最良の行動へ近づいた方策の探索について何を意味するか。
   3. Pendulum はサンプリングされた各トルクを $[-2, 2]$ にクリップする一方、`log_prob` はクリップ前のサンプルをスコア付けする。クリップを伴う環境の収益に対して、スコア関数推定量が不偏に保たれる理由を説明せよ。
   4. `pendulum_agent` のエージェントを `GaussianPolicy(3, 1, state_std=True)` で構築して σ が状態に依存するようにし、Pendulum の実行と同様に `train_reinforce` で 3 シードを訓練せよ。25 回の更新ごとに、σ を垂下状態 $(\cos\vartheta, \sin\vartheta, \dot{\vartheta}) = (-1, 0, 0)$ と直立状態 $(1, 0, 0)$ で記録せよ。σ が先に縮小するのはどちらの状態であり、それは小問 b とどのように整合するか。収益曲線を `runs_p` と比較せよ。

   Richard S. Sutton および Andrew G. Barto 著 Reinforcement Learning: An Introduction の演習問題 13.4 を改変。

3. [code] **不連続な報酬**。@eq-score-vs-pathwise の 2 つ目の等号は Q を通して微分する。$\mu = 0$、$\sigma = 0.5$（`g_score` と `g_path` の計算と同じ設定）とし、しきい値報酬として $Q(a) = 1$（$a > 1$ のとき）、$Q(a) = 0$（それ以外のとき）をとる。

   1. 厳密な勾配 $\nabla_\mu \mathbb{E}[Q(a)]$ を計算せよ。パスワイズ推定量は各サンプルに対して何を返すか。2 つ目の等号の導出のどの段階が破綻するか。
   2. その計算における `Q` を滑らかな階段関数 $Q_k(a) = 1/\big(1 + e^{-k(a-1)}\big)$（$k \in \{1, 10, 100, 1000\}$）に置き換え、`g_score` と `g_path` の平均および分散を記録せよ。サンプルは単精度であるため、$k \geq 100$ では最小付近のサンプルで因子 $e^{-k(a-1)}$ がオーバーフローし、式をそのまま書くと導関数がそこで NaN になる。数値的に安定なシグモイドを用いて Q_k を実装せよ。両者の分散を k に対してプロットせよ。大きい k において分散が低いのはどちらの推定量か。`g_path` の分散が k に（Q_k' の高さと幅を通じて）どう依存するか、また `g_score` の分散が k に（Q_k を通じて）どう依存するか説明せよ。
   3. Pendulum はサンプリングされた各トルクを $[-2, 2]$ にクリップする。クリップされ

たトルクを通じたパスワイズ推定量は、この区間の外のサンプルに対して何を返すか。演習問題 2c のスコア関数推定量がこれらのサンプルを無駄にしないのはなぜか。

4. **代理損失の値**。`policy_step` は、現在の方策が収集したばかりのバッチ上で代理損失 Equation 14.94 を評価する。`d2l.standardize` による重みは、平均 0、標準偏差 1 を（バッチの N ステップにわたって）持つ。
   1. $L(\theta) = -s\,c$ であることを示せ。ここで s はバッチにわたる $\log \pi_\theta(a_t \mid s_t)$ の標準偏差であり、c は重みとこれらの対数確率との相関係数である。これより $|L(\theta)| \le s$ を結論づけよ。
   2. 行動次元が 1 である `GaussianPolicy` について、$s \approx 1/\sqrt{2}$ が任意の平均ネットワークと任意の σ に対して成り立つことを示せ。このことは、Pendulum の実行における方策損失について何を意味するか。
   3. CartPole において方策がほぼ決定論的になるにつれて、s はどのように振る舞うか。このことは、訓練終盤における L の値について何を意味するか。

5. [code] **Adam のもとでの正規化**。●素の SGD のもとでは、重みをその標準偏差で割ると実効学習率が変化する（演習問題 6、14.8 節）。`train_reinforce` は代わりに Adam で更新する。各座標に対するステップは $-\eta\,\hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon_0)$ であり、ここで $\hat{m}_t$ と $\hat{v}_t$ は勾配 g_t と g_t^2 のバイアス補正済み移動平均（減衰率は $\beta_1 = 0.9$ および $\beta_2 = 0.999$）である。
   1. $\epsilon_0 = 0$ のとき、すべての勾配に同じ定数 $c > 0$ を掛けても Adam の各ステップが不変であることを示せ。
   2. 次式を示せ。

$$\hat{v}_t = \frac{1-\beta_2}{1-\beta_2^t}\sum_{i=1}^{t}\beta_2^{t-i} g_i^2.$$

   Pendulum の実行長である $t = 300$ において、最も古い二乗勾配と最も新しい二乗勾配の重みは何倍異なるか。ある座標の勾配が一定の符号を保ったまま、300 回の更新の実行の途中でスケールが 10 分の 1 に低下したとする。実行終了時のステップは開始時のステップと比べてどの程度の倍率だけ小さくなるか。
   3. `d2l.standardize` の中心化は `train_reinforce` 内で維持したまま除算を取り除き、重みを `A - A.mean()`（ただし `A = G - ac.value_np(batch.obs)`）として、2 つの変種がスケールのみ異なるようにせよ。各バッチの重みの標準偏差を記録し、両変種をそれぞれ 3 シードずつ CartPole と Pendulum で訓練せよ。各タスクにおいて重みのスケールはどのように推移するか。小問 b の結果から、除算を取り除くことで収益曲線がより大きく変化すると予測されるのはどちらのタスクか。

6. **スコア関数推定量とパスワイズ推定量の分散**。●サンプル `g_score` と `g_path` は、Equation 14.96 の 2 つの形式を $Q(a) = -(a-1)^2$、$\mu = 0$ において推定する。$\mathbb{E}[z^4] = 3$ および $\mathbb{E}[z^6] = 15$（$z \sim \mathcal{N}(0,1)$ に対して）を用いよ。
   1. 1 サンプルあたりの分散が、スコア関数推定量では $15\sigma^2 + 14 + 1/\sigma^2$、パスワイズ推定量では $4\sigma^2$ となることを示せ。$\sigma = 0.5$ において両者を評価し、出力された分散と比較せよ。
   2. 定数 c を Q に加えることは、ベースライン $b = -c$ を差し引くことと同じである。スコア a/σ^2 を b^* の公式（演習問題 2、14.8 節）に代入して、$b^* = -(1 + 3\sigma^2)$ となること、およびスコア関数推定量の最小分散が $6\sigma^2 + 8$ となることを示せ。$c = 10$ に対して出力された分散を導出した式と照合せよ。

3. $\sigma \to 0$ とする。最適ベースラインがある場合とない場合とで、2 つの分散はそれぞれどのように振る舞うか。スコア関数の分散の極限 8 を、Q の μ の周りでのテイラー展開を通じて説明せよ。決定論的アクターに対して利用可能なまま保たれるのは 2 つの推定量のうちどちらか。

7. [code] **接カーネル。**●`dV` の測定では、$\widehat{V}$ を 1 つのプローブ状態において `fit_value` の素の SGD 1 ステップによって引き上げる。価値ネットワークのパラメータを w と書き、その接カーネルを

$$k(s, s') = \nabla_w \widehat{V}(s)^\top \nabla_w \widehat{V}(s') \tag{14.97}$$

と書く（Equation 14.91 と同様）。

1. サイズ η の勾配ステップを $\frac{1}{2}\big(y - \widehat{V}(s_0)\big)^2$ 上で 1 回とると、s における推定値が $\eta\,\big(y - \widehat{V}(s_0)\big)\,k(s_0, s) + O(\eta^2)$ だけ変化することを示せ。`fit_value` は代わりに $\big(y - \widehat{V}(s_0)\big)^2$ を最小化するが、この予測はどう変わるか。
2. 新規に構築した `net_probe` について、勾配 $\nabla_w \widehat{V}$ を 256 個のプローブ状態 `S` において計算し、カーネル値 $k(s_0, s)$ を s_0 = `S[0]` に対して計算せよ。Adam を使わずに $\eta = 10^{-3}$ で $y = \widehat{V}(s_0) + 1$ に向かう素の勾配ステップを 1 回とり、すべてのプローブにおける実測の変化量を予測と比較せよ。
3. $k(s_0, s)/k(s_0, s_0)$ を距離 $\|s - s_0\|$ に対してプロットせよ。負のカーネル値を持つプローブは何個あり、その個数はコードセルが下方向に動いたと報告したプローブの個数と比べてどうか。
4. バランスをうまく保つ CartPole の方策は、大半が垂直に近い状態からなるバッチを収集する。このカーネルを用いて、そのようなバッチへの当てはめが、バッチには一度も含まれない大きなポール角における価値推定と方策をどう変化させるか説明せよ。また、その変化によってそれらの角度における方策が悪化し得るのはなぜか。

Chapter 15

深層強化学習

第 14 章の最後のエージェントは、数十万ステップをかけて振り子を振り上げた。シミュレータならそれも安価に済む。しかし、ロボットアームであれば先に摩耗してしまうだろうし、自身の回答で訓練される言語モデルでは、回答の各バッチを 1 回の更新ごとに使い捨てるわけにはいかない。そのエージェントは毎回エピソードが終わるまで待ち、各バッチを 1 度しか使わず、1 回の更新で方策（policy）がどれだけ変化するかも一切制限していなかった。1 つのサンプルからどれだけのことを学べるのか、そしてそこからさらに多くを学ばせようとするとどのような問題が生じるのだろうか。

これには 4 つのアイデアが答えを与えるが、それぞれ固有の誤差を伴う。エージェントはエピソードの終了を待つ代わりに、残りの区間に対する学習された推定値を信頼できる (Samuel 1959; Sutton 1988)。ただし、学習が浅いうちはその推定値に統計的バイアスがある。過去の方策によるデータを重み付けし直すか再生して再利用することもできる (Lin 1992)。しかし、データは古くなりターゲットも移動する。方策の振る舞いの変化量によって各更新を抑えることもできる (Kakade and Langford 2002; Schulman et al. 2017)。もっとも、その上界が制限するのは各ステップの幅であり、ステップを重ねた到達先ではない。環境のモデルを学習してその内部でプランニングすることもできる (Sutton 1990)。その場合、モデルの誤差を引き継ぐうえ、ステップごとに計算が必要になる。さらに、学習されたターゲット、状態間で共有されるネットワーク、そして別の方策のデータが組み合わさると、表形式では決して起きない価値の発散が生じうる。制御器、ゲームプレイヤ、そして事後学習された言語モデルは、いずれもこれらのアイデアから構築されている。

本章では、これらのアイデアを 3 つの段階に分けて展開する。まずはオンポリシーおよび正則化付きの方策最適化から始める。15.1 節ではエピソード終了までの待機をブートストラップされたクリティックと多ステップのアドバンテージに置き換え、15.2 節では確率比クリッピング（probability-ratio clipping）によって保護された複数回の更新にわたり各ロールアウトを再利用し、15.3 節では報酬自体が比較データから当てはめられる場合に方策を参照分布へつなぎ留める。次に、オフポリシーの再生とプランニングに進む。15.4 節では凍結したターゲットのコピーと Double DQN によって、再生された遷移上でのニューラル Q 学習を安定化させ、15.5 節ではツインクリティックと最大エントロピー（maximum entropy）の連続行動アクターを組み合わせて数千ステップで振り子を振り上げ、15.6 節では数百ステップから物理法則を学習してモデル内部でプランニングする。最後に、オンライン探索（exploration）や遷移の不確実性がない設定を扱う。15.7 節では固定されたログから保守的に学習し、15.8 節では本章の手法群をトークンレベルの言語モデルの事後学習へと特化させる。

離散行動の各手法は環境ステップ数を共通軸として CartPole で一貫して評価し、ソフトアクタ

ークリティックと学習されたモデルは Pendulum で扱い、検証器タスクと 2 つの小さなグリッドワールドは厳密な閉形式の検証を与える。すべての例は PyTorch と JAX の双方で、CPU 上で数分以内に実行できる。Figure 14.23 と Table 14.3 は両章のすべての手法を位置づけ、15.9 節は本章の範囲を超える発展的トピックを概観する。なお、大規模な選好に基づく事後学習については「言語モデル」パートで扱う。

15.1 アクター・クリティック法と複数ステップ収益

14.9 節のエージェントはポールのバランスを取ることを学習するが、その学習は遅い。各更新では 8 エピソード全体、すなわち最大 4,000 ステップものバランス維持が終わるのを待ち、そのうえでようやく各行動に対し、エピソードの残りの部分が期待よりうまくいったかを問いかける。一方、14.3 節の Q 学習エージェントにはそのような忍耐は不要であった。遷移のたびに、エピソードの残りの代わりにその先の展開に対する自身の推定値を用いて、テーブルの 1 要素を書き換えていたからである。方策勾配エージェントもこれと同じ方法で学習できるだろうか。

結論から言えば可能であり、そのために必要な部品はすでに手元にある。14.9 節で学習されたベースライン $\hat{V}$ は、ある状態に続く報酬の割引和である残りの累積報酬（reward-to-go）を予測するように訓練されている。これがエピソードの残りを予測するのであれば、エージェントはその終了を待つ必要がない。1 ステップ分の報酬に次の状態での予測値を加えたものを、サンプリングされた末尾の代わりに使えるからである。このとき、方策は**アクター**となり、価値推定はその**クリティック**となる。Barto et al. (1983) はまさにこのカートポール問題に対して最初のこのような組を構築し、同論文における「適応的クリティック要素（adaptive critic element）」がクリティックの名前の由来となった。

この置き換えは、将来のランダム性とクリティックの誤差とを引き換えにするものである。本節ではこのトレードオフを詳しく調べる。すなわち、単一の遷移からアクターに何が伝わるのか、サンプリングされた末尾をどれだけ残すべきか、そしてクリティックが取り除く分散よりもクリティック自身の誤差のほうが大きく響く CartPole において、このトレードオフがどのような結果をもたらすのかを明らかにする。本章の残りで扱う手法の大半は何らかのクリティックを備えており、ここで構築する 1 ステップ目標と複数ステップ目標を幾何重みで混合した推定量は、PPO で用いられているものである。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

タスクとエージェントは 14.9 節のものを引き継ぐ。CartPole の状態はカートの位置・速度とポールの角度・角速度という 4 つの数値からなり、2 つの行動はカートを左または右に押す。ポールが倒れずに立っているステップごとに報酬 1 が与えられ、エピソードは 500 ステップで打ち切られるため、可能な最大の収益は 500 である。割引率は $\gamma = 0.99$ とする。エージェントは `ActorCritic.mlp` であり、それぞれ 64 ユニットの隠れ層を 1 つ持つ方策ネットワークと価値ネットワークから構成され、各ネットワークが独自の Adam オプティマイザを持つ。以下の主な訓練では、1 回あたり 8 エピソードからなる更新を 150 回、環境ステップ数にして最大 600,000 ステ

ップ実行する。新しいハイパーパラメータである critic_steps はバッチごとにクリティックが反復する回帰パスの回数を設定し、grad_clip はアクターの勾配のノルムに上限を設ける。これらはいずれも訓練ループとあわせて説明する。

```
gamma, num_updates, batch_episodes = 0.99, 150, 8
num_seeds, critic_steps, grad_clip = 3, 20, 0.5
def cartpole_agent(seed):
    return d2l.ActorCritic.mlp(4, 2, seed=seed)
```

15.1.1 時間差分アドバンテージ

15.1.1.1 残りの累積報酬のブートストラップ

クリティックが何を置き換えられるかを確認するため、残りの累積報酬を 1 ステップ分書き出してみよう。

$$\hat{G}_t = r_t + \gamma \hat{G}_{t+1}. \tag{15.1}$$

学習されたベースラインは残りの累積報酬に向かって回帰されるため、$\hat{V}(s_{t+1})$ はまさに $\hat{G}_{t+1}$ がサンプリングする量を予測している。そこでサンプルが得られるのを待つ代わりに、予測値を用いて $\hat{G}_t \approx r_t + \gamma\hat{V}(s_{t+1})$ と置き換える。ステップ t におけるスコアの重みは $\hat{G}_t - \hat{V}(s_t)$ であったが、この置き換えによって次のようになる。

$$\delta_t = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t), \tag{15.2}$$

ただし、$\hat{V}(s_{t+1}) = 0$（s_{t+1} が終端状態のとき）とする。この**時間差分誤差**（temporal-difference error、TD 誤差）は、同じ収益に対する 2 つの予測、すなわち s_t での予測と、実際の報酬を 1 つ得た 1 ステップ後の予測とを比較している。

14.3 節の TD 誤差も、1 点を除けばこれと同じ形をしていた。Q 学習は最適価値を推定するため、最良の継続価値 $\max_{a'} \hat{Q}(s', a')$ でブートストラップするが、ここでのブートストラップは同節の SARSA の目標と同様に、現在の方策自身の継続価値を用いる。終端マスクには terminated を用い、truncated は用いない。時間制限はタスク自体を終了させずに観測を打ち切るだけなので、打ち切り時点における状態の価値は 0 ではないからである（14.1 節）。td_target は、状態を価値に写す任意の関数から、目標 $r_t + \gamma\hat{V}(s_{t+1})$ を Batch（d2l.rollout が記録する遷移とエピソード境界の平坦なレコード、14.7 節）内のすべての遷移に対して計算する。戻り値は NumPy 配列であるため、クリティックの勾配がこの目標を通って流れることはない。

```
@d2l.add_to_class(d2l.Batch)
def td_target(self, bootstrap, gamma):
    """r_t + gamma (1 - terminated) V(s'), by a numpy bootstrap."""
    return self.rew + gamma * (1 - self.term) * bootstrap(self.next_obs)
```

14.8 節の残りの累積報酬推定量と同様に、アクターの更新では、エピソード後半のステップを前半に対して割り引く因子 γ^t を省略する。

15.1.1.2 1 遷移の手計算

3 つの数値を見れば、この新しい重みが何を意味するかがわかる。クリティックがすべての状態で 100、すなわちポールを永久に倒さず維持したときの価値 $1/(1-\gamma)$（$\gamma = 0.99$ の場合）を予測しているとしよう。ポールが立ったままのステップでは $\delta_t = 1 + 0.99 \cdot 100 - 100 = 0$ となり、そのステップは予測どおりに進んだため、とった行動は推奨も抑制もされない。一方、ポールが倒れるステップではエピソードが終了してブートストラップがマスクされ、$\delta_t = 1 - 100 = -99$ となる。エピソードを終わらせた行動の確率は強く押し下げられるのである。

初期化されたばかりのクリティックは、あらゆる状態で 0 付近の価値を予測する。すると、行動によってポールが立ったままでも倒れても、すべてのステップで $\delta_t \approx 1$ となる。とられた行動はどれも同じ大きさだけ押し上げられるため、クリティックが両者を区別できるようになるまでアクターはほとんど学習できない。

15.1.1.3 正確なクリティックが与えるアドバンテージ

完全なクリティックがあれば、TD 誤差はモンテカルロ重みと同じ量を推定する。いま仮に $\hat{V} = V^\pi$ であるとしよう。行動価値は $Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^\pi(s')$ を満たす。これは 14.2 節のベルマン方程式を V^π を用いて書き表したものである。したがって、Equation 15.2 を次の状態について平均すると次式が得られる。

$$\begin{aligned}\mathbb{E}\big[\delta_t \mid s_t, a_t\big] &= r(s_t,a_t) + \gamma \sum_{s'} P(s' \mid s_t,a_t)\, V^\pi(s') - V^\pi(s_t) \\ &= Q^\pi(s_t,a_t) - V^\pi(s_t).\end{aligned}$$

右辺はアドバンテージ $A^\pi(s_t,a_t)$ であり、とった行動がその状態における方策の平均よりもどれだけ優れているかを表す。14.8 節の学習されたベースラインは、これを $\hat{G}_t - \hat{V}(s_t)$ によって推定していた。モンテカルロ重みがエピソードの残り全体を必要としたのに対し、ここでは 1 つの報酬と 2 回のクリティック評価を伴う 1 回の遷移だけで、アドバンテージの不偏サンプルが得られる。

訓練中の $\hat{V}$ は V^π と等しくないため、δ_t はアドバンテージのバイアスを持つ推定値となる。14.8 節のゼロ平均恒等式によれば、行動の前に確定している任意の量は、勾配の平均を変えることなく重みから差し引くことができる。現在の状態のみに依存する $\hat{V}(s_t)$ にはこの性質が当てはまる。しかし、次の状態を通じて行動に依存する $\hat{V}(s_{t+1})$ には当てはまらないため、行動の遷移先の状態におけるクリティックの誤差が統計的バイアスとして更新に入り込む（演習問題 1）。このバイアスは、γ^t の省略によってすでに生じている傾きに上乗せされるものである（14.8 節）。

クリティック自身の更新は、サンプリングによる方策評価（policy evaluation）である。14.2 節における方策評価は、V^π を不動点に持つベルマン期待値バックアップを適用していた。$\hat{V}(s_t)$ を $r_t + \gamma \hat{V}(s_{t+1})$ に向かって回帰させることは、ちょうど Q 学習が最適性バックアップをサンプルで置き換えたのと同様に、その期待値を 1 回のサンプリングされた遷移で置き換えることにあたる。クリティックが現在の方策の価値を推定し、アクターがその推定値に対して方策を改善するため、アクター・クリティックは 14.2 節の一般化方策反復法（generalized policy iteration）の近似形となっている。回帰目標が $\hat{V}$ 自身を含んでいるため、勾配は予測 $\hat{V}(s_t)$ についてのみ計算し、目標を通しては計算しない。このような更新は**半勾配**（semi-gradient）ステップと呼ばれ、その期待更新量は固定されたいかなる目的関数の勾配とも一致しない。

15.1.1.4 チェーンの例

ブートストラップはどれだけの分散を取り除き、どれだけのバイアスをつけ加えるのだろうか。小さなチェーンを用いれば、その両方に閉じた形で答えられる。10 個の状態 $0, 1, \ldots, 9$ を考えよう。状態 s からエージェントは常に $s+1$ へ移動し、状態 9 からの遷移でエピソードが終了して報酬 1 が与えられる。各報酬には標準偏差 $\sigma = 0.15$ の独立なガウスノイズが加わり、割引率は $\gamma = 0.97$ とする。クリティックは $e(s) = 0.5\,(1 - s/9)^2$ だけの誤差を持ち、始点では大きく外れているが終点では誤差がない。ゼロ初期化からの TD 学習では、終端で得られる報酬が後半の状態にまず伝わるため、このような形状の誤差が生じやすい。

n 個の報酬をサンプリングしてから状態 n でクリティックからブートストラップする目標は、割り引かれたその状態でのクリティックの誤差と、n 個の報酬のノイズとを受け継ぐ。そのバイアスと分散は、次の式で与えられる。

$$\text{bias} = \gamma^n e(n), \qquad \text{variance} = \sigma^2 \frac{1 - \gamma^{2n}}{1 - \gamma^2}, \tag{15.3}$$

ただし上式は $n \le 9$ の場合であり、モンテカルロ目標である $n = 10$ にはバイアスがない。$n = 1$ ではバイアスが 0.38、分散が 0.02 であり、$n = 10$ ではバイアスが 0、分散が 0.17 となる。バイアスの二乗と分散の和である平均二乗誤差は、0.17（$n = 1$）から 0.10（$n = 4$）まで下がったのち、再び 0.17（$n = 10$）まで上昇する。

最良の深さが両端の間にあるのは、深い目標ほど価値がより正確にわかっている状態からブートストラップするためである。もしクリティックがすべての状態で一様に 0.5 だけ外れていれば、ブートストラップを行う深さの範囲で誤差は単調に変化するはずである（演習問題 2）。Figure 15.1 に、この目標の族と実測された曲線を示す。

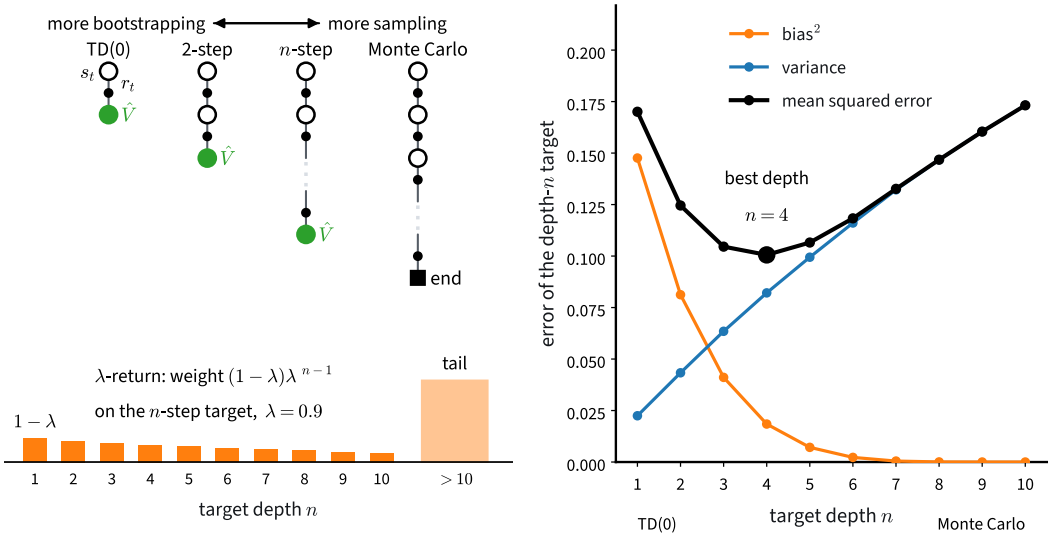


Figure 15.1: ブートストラップ目標の族。左：深さ 1、2、n、および終端までのバックアップ線図。緑のノードは目標がサンプリングを打ち切ってクリティックを信頼する状態を表し、重み $(1-\lambda)\lambda^{n-1}$（本節の λ 収益が $\lambda = 0.9$ のときに各目標へ割り当てる重み）を添えている。右：本文の 10 状態チェーンにおける深さ n の目標のバイアス、分散、平均二乗誤差（20,000 回のロールアウトで測定、最小誤差は $n = 4$）。

15.1.2 アクター・クリティックの更新

15.1.2.1 1 つの数値からの 2 つの更新

これで両方の学習器が同じスカラーから更新されるようになる。クリティックのパラメータを w、クリティックを $\hat{V}_w$ と書こう。遷移 (s_t, a_t, r_t, s_{t+1}) の後、2 つの更新式は次のようになる。

$$w \leftarrow w + \alpha\, \delta_t\, \nabla_w \hat{V}_w(s_t), \qquad \theta \leftarrow \theta + \eta\, \delta_t\, \nabla_\theta \log \pi_\theta(a_t \mid s_t), \tag{15.4}$$

ここで、クリティックのステップサイズは 14.3 節の TD 学習と同様に α とし、アクターのステップサイズは 14.7 節の方策勾配と同様に η とする。クリティックの更新は 1 ステップ目標に向かう半勾配ステップであり、アクターの更新は 14.8 節の方策勾配ステップにおいて重みを δ_t としたものである。たった 1 つの数値が両方の役割を果たす。すなわち、δ_t はクリティックにはその予測がどれだけ外れていたかを伝え、アクターには直前にとった行動がクリティックの予想より良い結果になったか悪い結果になったかを伝える（Figure 15.2）。δ_t の計算には 1 回の遷移しか要らないため、Q 学習がテーブルを更新したのと同じように、エージェントが行動しながら毎ステップ Equation 15.4 を適用でき、学習を始める前に何かが終了するのを待つ必要はない。

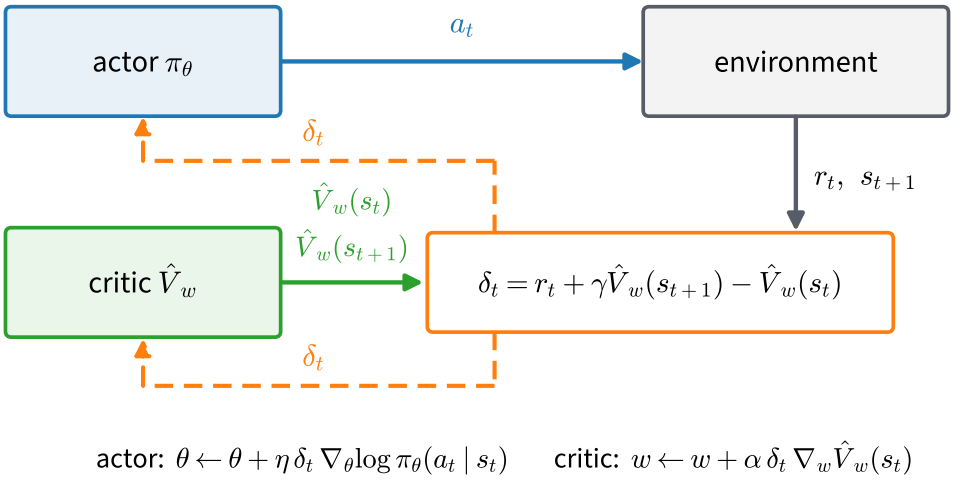


Figure 15.2: アクター・クリティックのループ。1 回の遷移から 1 つの TD 誤差が生じ、クリティックの予測を修正するとともにアクターのステップを重み付けする。

クリティックの目標は両方のネットワークに伴って移動する。アクターのステップはクリティックが予測すべき価値を変化させ、クリティックのステップは自身の目標を変化させる。もしアクターがクリティックの追従できる速さを超えて動けば、アクターは古くなったクリティックの誤差で重み付けされることになる。Konda and Tsitsiklis (2000) は、線形クリティックと減衰するステップサイズのもとで、クリティックのステップサイズがアクターよりもゆっくり縮小し、クリティックが現在の方策に追いつき続けられるときにアクター・クリティックが収束することを証明した。以下のループでは、バッチごとにクリティックへ複数回の回帰パスを与えることで同じ効果を得る。

15.1.2.2 バッチ化されたループ

以下のコードでは、遷移のたびに Equation 15.4 を適用するのではなく、14.9 節の 8 エピソードからなるバッチを維持したうえで各バッチに対する処理を変更する。まずクリティックが、バッチの 1 ステップ目標に向かって `critic_steps` 回の回帰パスを実行し、各パスの前に現在のクリティックから目標を再計算する。回帰器自身の予測を含む目標に向かって回帰することを**適合 TD**（fitted TD）と呼び、目標を再計算することでそれは移動目標となる。続いてアクターが、再適合されたクリティックの TD 誤差で重み付けされた 1 ステップを実行する。本節ではこのループをバッチ型 1 ステップ・アクター・クリティックと呼ぶ。これに最も近い著名な手法は Mnih et al. (2016) のアドバンテージ・アクター・クリティックであり、同じスコアを n ステップのアドバンテージで重み付けし、さらにエントロピーボーナスを加える。元の手法である A3C は多数のワーカーを非同期に実行していたが、現在の実装で用いられている同期版の **A2C** は、多数の環境から並列に収集したロールアウトごとに 1 回の同時勾配ステップを実行する。

アクターの重みには、14.9 節のモンテカルロ重みのようにバッチ内で正規化したものではなく、生の TD 誤差をそのまま用いる。これは Adam がステップをスケーリングする仕組みのためである。Adam は各勾配座標を過去の二乗の移動平均の平方根で割るが、この移動平均はデフォルトの減衰率では約 1,000 ステップ分を記憶し、今回の 150 回の更新よりも長い。そのため、最初のうちは重みのスケールによらずステップの大きさがほぼ同じになり、後のステップは勾配が初期の大きさより小さくなったときにのみ縮小する。クリティックが改善するにつれて、TD 誤差の標準偏差は訓練初期の約 3 から上限付近での 0.1 以下へと低下し、その時点では大半がクリティック自身のノイズとなるが、それに伴って Adam のステップも実行全体で約 8 分の 1 に縮小する。もし TD 誤差を正規化してしまうと、勾配が初期の大きさに保たれ、そのノイズが定める方向へ Adam が大きなステップを取り続けてしまう（演習問題 6）。

訓練の初期には生の重みが、ひいてはアクターの勾配が、正規化した重みを用いる場合よりも大きくなる。そこで一般的な値である 0.5 で勾配のノルムに上限を設ける。Adam のもとでは、この上限は 1 ステップの大きさを制限するのではなく、初期の大きなバッチが Adam の移動平均にどれだけ強く寄与するかを制限する。`policy_step_clip` は、バッチ内の行動の重み付き対数確率に対して 1 回の勾配ステップをとる 14.5 節の `policy_step` にこの上限を加えたものであり、上限適用前に測定したノルムを返す。

```
@nnx.jit
def actor_grads(ac, obs, act, w, mask):
    """Gradient of -(1/n) sum_t w_t log pi(a_t|s_t) over the n real rows,
    the loss of d2l.policy_step."""
    return nnx.grad(lambda policy: -(mask * w * ac.log_prob(
        obs, act, policy)).sum() / mask.sum())(ac.policy)

@nnx.jit
def _actor_pass(ac, obs, act, w, mask, clip):
    """One actor step with the gradient norm capped at `clip`; returns the
    norm before the cap."""
    grads = actor_grads(ac, obs, act, w, mask)
    gnorm = optax.global_norm(grads)
    grads = jax.tree.map(lambda g: g * jnp.minimum(1.0, clip / gnorm), grads)
    ac.opt_pi.update(ac.policy, grads)
    return gnorm

def policy_step_clip(ac, obs, act, weight, clip=grad_clip):
    """d2l.policy_step with the actor's gradient norm capped at `clip`;
    returns the norm measured before the cap."""
```

```
    return float(_actor_pass(ac, *d2l.padded(obs, act, np.float32(weight)),
                             clip))
```

policy_step_clip は d2l.padded を用いてバッチをバケット長までパディングする（d2l.policy_step と同様の仕組み）。これにより、コンパイル済みの _actor_pass が異なる長さのバッチ間で再利用され、マスクによってパディングされた行は損失から除外される。勾配のみを計算する関数が actor_grads であり、本節末尾のプローブで再利用する。

訓練ループを実装するのが train である。各更新では d2l.rollout でバッチを収集し、関数 weight(ac, b) を呼び出してバッチ b の各ステップにおけるアクターの重みを求め、その重みで上限付きのアクターステップを 1 回実行したうえで、これまでの環境ステップ数、バッチの平均収益、上限適用前のアクターの勾配ノルムを順に生成する。この重み関数の中でクリティックの適合も進めるため、本節のすべての推定量に対してループ自体は変わらず、重み関数だけを入れ替えればよい。アクター・クリティックの重みは td_weight である。d2l.fit_value（14.9 節）がクリティックの予測と目標ベクトルの二乗誤差に対して Adam の 1 ステップを実行し、ac.value_np が状態のバッチ上でクリティックを評価し、クリティックの反復パスが終わった後の重みは再適合されたクリティックの生の TD 誤差となる。

```
def train(seed, ac, weight, num_updates=num_updates,
          batch_episodes=batch_episodes):
    """Batched policy gradient with the per-step weights weight(ac, b);
    yields (env steps, mean return of the batch, actor grad norm)."""
    rng, env = np.random.default_rng(seed), gym.make('CartPole-v1')
    env.reset(seed=seed)
    steps = 0
    for _ in range(num_updates):
        b = d2l.rollout(env, ac.act, batch_episodes, rng)
        gnorm = policy_step_clip(ac, b.obs, b.act, weight(ac, b))
        steps += len(b)
        yield steps, float(b.episode_returns().mean()), gnorm

def td_weight(ac, b):
    """Actor-critic: refit the critic toward its one-step target, then
    weight each step by the raw TD error of the refitted critic."""
    for _ in range(critic_steps):   # fresh target, one pass, repeat
        d2l.fit_value(ac, b.obs, b.td_target(ac.value_np, gamma))
    return b.td_target(ac.value_np, gamma) - ac.value_np(b.obs)
```

アクターの重みを計算する前にクリティックを適合させるというのは 1 つの設計選択である。A2C や大半の GAE 実装では、バッチ収集時点のクリティックを用いて TD 誤差を計算し、その後で両方のネットワークを更新する。A2C のようにロールアウトあたりのクリティックの更新が 1 ステップであれば、この順序はほとんど影響しない。しかしバッチあたり 20 パスとなると話は別である。再適合されたクリティックはバッチ自身の状態において V^{π} にはるかに近づくため、クリティックに全面的に依存しているここでのアクターの重みに、クリティックの誤差が乗りにくくなるからである（演習問題 3 でこれら 2 つの順序を比較する）。

毎回新しいバッチを集めることは、もう 1 つの面でもクリティックを助けている。各バッチは現在の方策とそれが訪れる状態から得られるため、現在の振る舞いにとって問題となるクリティックの誤差があれば、次のバッチで関連する遷移に出会いやすい。もっとも、このフィードバックは収束を保証するものではなく、非線形クリティックを用いた半勾配 TD は自方策のデータ上であっても発散しうる (Tsitsiklis and Van Roy 1997)。さらに過去の方策から再生された遷移で訓練

すると、同じ更新が関数近似、ブートストラップ、オフポリシーデータという Sutton and Barto (2018) の**死の三要素**（deadly triad）を兼ね備えることになり、発散が頻繁に起こるようになる。この場合は 15.4 節で扱う。

15.1.2.3 CartPole におけるブートストラップ重みとモンテカルロ重み

ブートストラップされた重みは、置き換え元のモンテカルロ重みよりも速く学習するだろうか。比較基準とするのは 14.9 節のループである学習されたベースライン付き REINFORCE であり、重み以外のすべてのコードを両条件で共有できるよう、2 つ目の重み関数 `mc_weight` として記述する。重み以外の違いをなくすため、さらに 2 つの細部を揃える。すなわち、アクターのステップは `policy_step_clip` を通し、クリティックはバッチごとに同じ `critic_steps` 回の回帰パスを（ここでは固定されたモンテカルロ目標 $\widehat{G}_t$ に向かって）実行する。14.9 節と同様に、`truncation_bootstrap` が時間制限での打ち切り時点におけるクリティックの割引価値を残りの累積報酬に加え、`d2l.standardize` がバッチ内で重みを標準化し、重みはクリティックのパスより前に計算する。アクターとクリティックは別々のネットワークであるため、この順序によってアクターのステップが変わることはない。

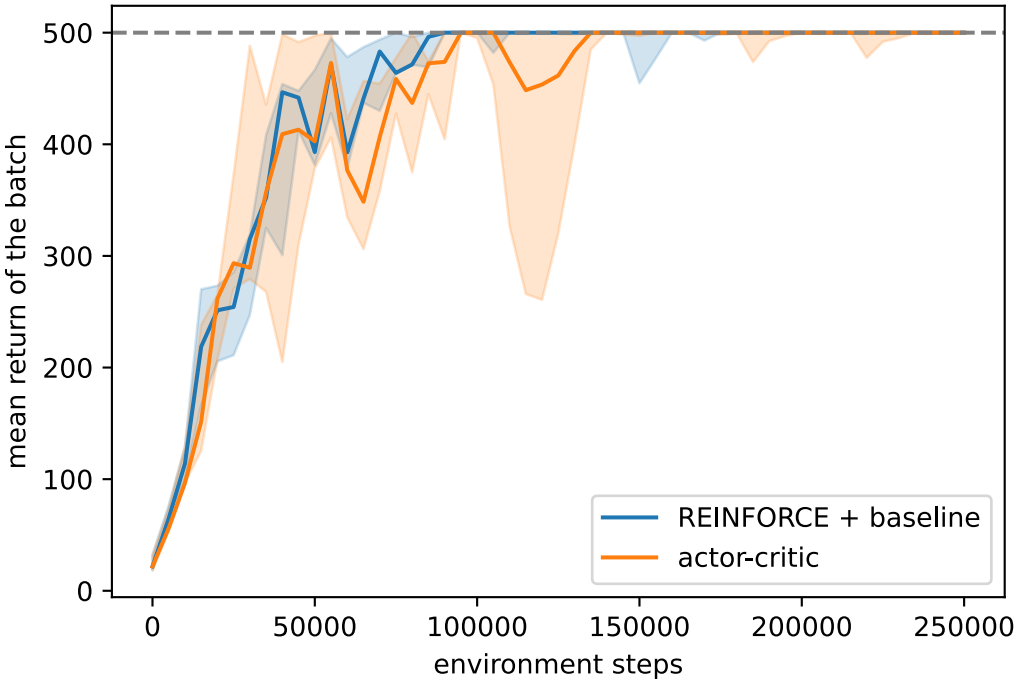
```
def mc_weight(ac, b):
    """REINFORCE with a learned baseline: the standardized Monte Carlo
    advantage G - V, after which the critic fits G."""
    G = (b.reward_to_go(gamma)
         + b.truncation_bootstrap(gamma, ac.value_np))   # time limits
    w = d2l.standardize(G - ac.value_np(b.obs))
    d2l.fit_value(ac, b.obs, G, num_steps=critic_steps)
    return w
```

それぞれの条件を 3 つのシードで訓練し、本章の以降でもコストの尺度として用いる環境ステップ数に対してバッチの平均収益をプロットする。`d2l.plot_curves`（オンライン付録「ユーティリティ関数とクラス」）はシード間の中央値を、最小シードから最大シードまでの帯とともに描画する。更新あたりのステップ数はシードごとに異なるため、まず `d2l.on_grid` で各実行結果を共通のステップ数のグリッド上に補間する。

```
runs = {'REINFORCE + baseline': np.array(
    [list(train(s, cartpole_agent(s), mc_weight)) for s in range(num_seeds)])}
```

次に、`'actor-critic'` 条件を `td_weight` を用いて同じ 3 つのシードで訓練し、両条件を共通のステップ数 `grid` 上へ `d2l.on_grid` で補間して、シード間でのバッチ収益の中央値と最小〜最大の帯をプロットする。

```
runs['actor-critic'] = np.array(
    [list(train(s, cartpole_agent(s), td_weight)) for s in range(num_seeds)])
grid = np.arange(0, 250_001, 5_000)
d2l.plot_curves({name: d2l.on_grid(r, grid, 1) for name, r in runs.items()},
                xlabel='environment steps', ylabel='mean return of the batch',
                x=grid, reference=500)
```



プロットをシードごとに読み解くため、直近 10 バッチの平均収益が初めて 450 に達するまでに各実行が要したステップ数と、アクターの勾配に対する上限が作動した頻度とを集計する。`steps_to` は実行結果の行から最初の該当ステップを見つける関数であり、15.5 節でも再び使用する。

```
def steps_to(run, level, window):
    """Env steps at which the mean return of the last `window` rows of
    run = [(env step, return, ...), ...] first reaches `level`; inf if it
    never does."""
    mean = np.convolve(run[:, 1], np.ones(window) / window, 'valid')
    hit = np.flatnonzero(mean >= level)
    return run[hit[0] + window - 1, 0] if len(hit) else np.inf
```

`steps_to(run, 450, 10)` を `runs` 内の各シードに適用し、10 バッチにわたって平均収益 450 を維持するまでに各条件が何千環境ステップを要したかを、アクターの勾配ノルムが `grad_clip` を超えた更新の割合とあわせて集計する。

```
for name, r in runs.items():
    k = [f'{steps_to(run, 450, 10) / 1000:.0f}' for run in r]
    print(f'{name:>20}: 450 after {", ".join(k)} thousand steps; '
          f'cap active {(r[:, :, 2] > grad_clip).mean():.0%}')
```

```
REINFORCE + baseline: 450 after 67, 92, 83 thousand steps; cap active 0%
        actor-critic: 450 after 98, 106, 58 thousand steps; cap active 28%
```

ここに示した 3 シードのうち少なくとも 2 シードでは、モンテカルロ重みのほうが先に上限付近へ到達する。3 シードは小さなサンプルであるが、同じバッチあたり 20 回のクリティックパスで 40 シードにわたって実験したところ、少なくとも 35 シードで REINFORCE が先に 450 へ到達し、必要な環境ステップ数も概して約 6 分の 1 から 3 分の 1 少なかった。また、上限を大きく下回っ

て終了した実行は REINFORCE には 1 つもなかったのに対し、アクター・クリティックでは 2 つあった。勾配の上限はアクター・クリティックの更新の 5 分の 1 から 5 分の 2 で作動するが、正規化された重みによって勾配が小さく保たれる REINFORCE では一度も作動しない。

CartPole では、ブートストラップによって実行が安定するわけでもない。報酬は毎ステップ決定論的な $+1$ であり、時間制限でのブートストラップが打ち切られた各エピソードの長い末尾をすでにクリティックの価値で置き換えているため、モンテカルロ重みのノイズは最悪の場合よりもはるかに小さい。残るのはクリティックのバイアスであり、それはクリティックがまだ未熟な段階、すなわちアクター・クリティックが遅れをとる段階で最も大きくなる。未訓練のクリティックが一様に $\delta_t \approx 1$ を出すのがその極端な例である。

15.1.2.4 共通バッチ上での 2 つの重み

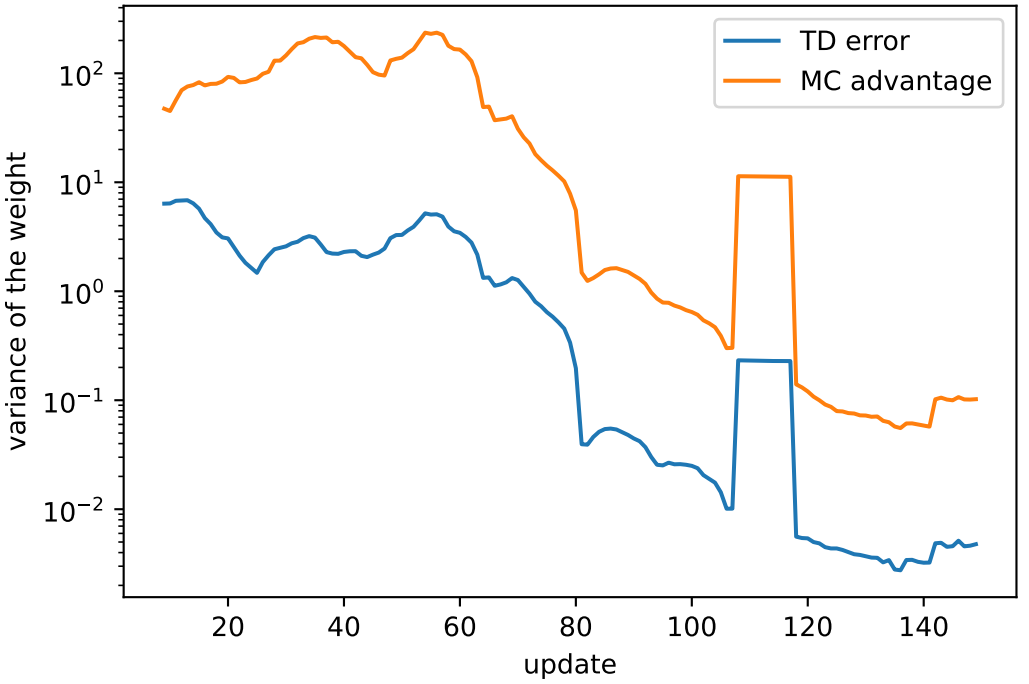
ここでブートストラップが学習を速めないのだとしたら、何を変えているのだろうか。同じバッチ上で両方の重みを測定してみよう。下のセルでは、1 つのシードについて `td_weight` を呼び出す重み関数でアクター・クリティック条件を再実行し、各更新において TD 誤差の標準偏差と、モンテカルロアドバンテージ `A_mc`（標準化前の `mc_weight` の重み）の標準偏差もあわせて記録する。

```
trace = []   # per update: std of the TD error, std of the MC advantage

def traced_td_weight(ac, b):
    """td_weight, recording the spread of both weights on its batch."""
    delta = td_weight(ac, b)
    A_mc = (b.reward_to_go(gamma)   # G of mc_weight, minus V
            + b.truncation_bootstrap(gamma, ac.value_np) - ac.value_np(b.obs))
    trace.append((delta.std(), A_mc.std()))
    return delta

for _ in train(0, cartpole_agent(0), traced_td_weight):
    pass
trace = np.array(trace)
```

`trace` から、最初の 10 回と最後の 50 回の更新における TD 誤差の標準偏差の中央値をモンテカルロアドバンテージのものと比較し、さらに分散比の中央値を計算して、バッチごとの両者の分散を対数スケールでプロットする。

```
for name, rows in (('first 10', trace[:10]), ('last 50', trace[-50:])):
    print(f'{name:>8} updates: std of the TD error {np.median(rows[:, 0]):.2f}'
          f', of the MC advantage {np.median(rows[:, 1]):.2f}')
print(f'median variance ratio, MC over TD: '
      f'{np.median((trace[:, 1] / trace[:, 0]) ** 2):.0f}x')
d2l.plot_curves({'TD error': trace[:, 0] ** 2, 'MC advantage': trace[:, 1] ** 2},
                xlabel='update', ylabel='variance of the weight', smooth=10,
                yscale='log')
```

```
first 10 updates: std of the TD error 2.51, of the MC advantage 6.37
 last 50 updates: std of the TD error 0.06, of the MC advantage 0.28
median variance ratio, MC over TD: 28x
```



同じ遷移上で比べると、最初の数回の更新においてモンテカルロアドバンテージは TD 誤差の 2〜3 倍の標準偏差、すなわち 5〜10 倍の分散を持ち、実行全体では TD 誤差の数十倍の分散を持つ（プロットの縦軸は対数スケールである）。TD 誤差は 1 つの報酬と 1 回の遷移のみに依存するが、モンテカルロアドバンテージはエピソードに残されたすべての報酬と遷移に依存する。ほぼすべてのエピソードが 500 ステップの打ち切りに達するようになると、両者の分散は急激に低下する。TD 誤差の分散が下がるのはその時点でクリティックが正確になっているためであり、モンテカルロアドバンテージの分散が下がるのは打ち切られた各エピソードの末尾をブートストラップがクリティックの価値で置き換えるためである。

分散の削減効果自体は確かにあるものの、CartPole において学習の律速となっているのは分散ではない。REINFORCE は更新ごとに 8 エピソードを平均して重みを正規化しており、十分に適合したベースラインがあればそれで事足りる。これに対し、クリティックのバイアスはクリティックが追いつくまでの間、毎回の更新を遅らせてしまう。

ブートストラップの本質的な強みは構造面にある。TD 誤差は 1 回の遷移しか必要としないため、エピソードが終わる前に更新できることにある。継続タスクではエピソード終了時の収益がそもそも訪れず、1 エピソードが数千ステップ続くタスクでは更新あたり 8 エピソードを集めるのに数万ステップを要する。そのような場面ではクリティックが唯一の選択肢となる（演習問題 5 で完全オンライン版を構築する）。これら両極端の間には調整ダイヤルが存在する。1 ステップではなく数ステップ後にブートストラップすれば、分散削減の大半を維持しつつ、クリティックのバイアスの影響を抑えられるのである。

15.1.3 複数ステップ収益と GAE

15.1.3.1 n ステップ収益

1 ステップ目標とモンテカルロ収益は、ある 1 つの族の両端にあたる。サンプリングされた n 個の報酬を含めてからクリティックでブートストラップすると、次のようになる。

$$\hat{G}_t^{(n)} = r_t + \gamma\, r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n\, \hat{V}(s_{t+n}), \tag{15.5}$$

そして $\hat{G}_t^{(n)} - \hat{V}(s_t)$ をアドバンテージ推定値として用いる。$n = 1$ のときこれは δ_t に一致し、n がエピソードの終端に達するときモンテカルロアドバンテージとなる。ステップを 1 つ増やすごとに、クリティックの誤差はさらに因子 γ だけ割り引かれてより先の状態で評価される一方、サンプリングされた報酬と遷移がもう 1 つ加わる。これはチェーンの例で見たトレードオフそのものであり、そこでは $n = 4$ が両者の均衡点であった。

15.1.3.2 λ 収益と逐次相殺の恒等式

1 つの深さだけを選ぶ代わりに、計算可能な形になる幾何重みを用いてすべての深さを平均することができる。

$$\hat{A}_t^{\text{GAE}} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} (\hat{G}_t^{(n)} - \hat{V}(s_t)), \qquad \lambda \in [0, 1]. \tag{15.6}$$

重み $(1-\lambda)\lambda^{n-1}$ の総和は 1 である。$\lambda = 0$ ではすべての重みが 1 ステップ推定値に置かれる。終了するエピソードでは、終端を超えるすべての深さがモンテカルロアドバンテージに等しくなるため、$\lambda \to 1$ とするとこの混合はモンテカルロアドバンテージに収束する。このように書くと、すべての深さを個別に計算する必要があるように見えるが、混合式が簡約されるためその必要はない。

命題。この混合は TD 誤差の割引和に等しい。

$$\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l\, \delta_{t+l}. \tag{15.7}$$

証明。各深さの式は逐次相殺によって TD 誤差の和となる。すなわち $\sum_{l=0}^{n-1} \gamma^l \delta_{t+l} = \sum_{l=0}^{n-1} \gamma^l r_{t+l} + \gamma^n \hat{V}(s_{t+n}) - \hat{V}(s_t) = \hat{G}_t^{(n)} - \hat{V}(s_t)$ であり、内側の $\hat{V}$ の項は対になって打ち消し合う。これを Equation 15.6 に代入して 2 つの和の順序を入れ替えると、組 (n, l)（ただし $l < n$）からの寄与は $(1-\lambda)\lambda^{n-1}\gamma^l\delta_{t+l}$ であるため、$\hat{A}_t^{\text{GAE}} = \sum_{l \geq 0} \gamma^l \delta_{t+l}\, (1-\lambda) \sum_{n \geq l+1} \lambda^{n-1}$ となる。内側の等比級数の和は λ^l である。■

この恒等式により、すべての深さにわたる平均がエピソード長に対して線形な 1 回の逆方向パスへと変換され、実際の実装でもそのように計算される (Schulman et al. 2016)。エピソード的タスクでは終端で $\hat{V} = 0$ となり後続の報酬もないため和は終端で止まり、終端より先の δ はすべて 0 となることから無限和は有限和と一致する。

15.1.3.3 逆方向スキャンとしての GAE

和 Equation 15.7 は、そのまま**一般化アドバンテージ推定**（generalized advantage estimation、GAE）のアルゴリズムになっている。すなわち、TD 誤差を計算し、エピソード境界ごとにリセットしながら因子 $\gamma\lambda$ で逆方向に累積すればよい。残りの累積報酬を漸化式 $\hat{G}_t = r_t + \gamma \hat{G}_{t+1}$ で計算する 14.8 節のメソッド `Batch.backward_scan` は、すでにこの漸化式を実装しており、変わるのは入力と割引因子だけである。

```
@d2l.add_to_class(d2l.Batch)
def gae(self, value_fn, gamma, lam):
    """GAE(gamma, lam): the reward-to-go scan, run on the TD errors."""
    delta = self.td_target(value_fn, gamma) - value_fn(self.obs)
    return self.backward_scan(delta, gamma * lam)
```

両端点はバッチ上で検証できる恒等式となっている。$\lambda = 0$ では、このスキャンは TD 誤差そのものを返す。$\lambda = 1$ では、終了するエピソードにおいて逐次相殺が終端まで進み、残りの累積報酬からベースラインを引いたもの、すなわち 14.8 節のモンテカルロアドバンテージを返す。

```
env = gym.make('CartPole-v1')
env.reset(seed=0)
fresh = cartpole_agent(0)
b = d2l.rollout(env, fresh.act, 8, np.random.default_rng(0))
V = fresh.value_np(b.obs)
delta = b.td_target(fresh.value_np, gamma) - V
assert np.allclose(b.gae(fresh.value_np, gamma, 0.0), delta)
assert np.allclose(b.gae(fresh.value_np, gamma, 1.0),
                   b.reward_to_go(gamma) - V, atol=1e-4)
print('lambda = 0 is the TD error; lambda = 1 is the Monte Carlo advantage')
```

```
lambda = 0 is the TD error; lambda = 1 is the Monte Carlo advantage
```

$\lambda = 1$ の恒等式が成り立つには、エピソードが終了する必要がある。時間制限で打ち切られたエピソードでは、`gae` が最後の TD 誤差を通じて欠けた将来をブートストラップするのに対し、`reward_to_go` は `truncation_bootstrap` で補われるまで将来を空として扱う。上の未訓練の方策は打ち切りよりずっと前にポールを倒すため、アサーションが成立している。この同じ逆方向の和にはオンライン版の TD(λ) があり、クリティックのパラメータごとに減衰する**適格度トレース**（eligibility trace）を保持することで、エピソードを保存せずにストリームから学習するエージェントに適している (Sutton 1988; Elsayed et al. 2024)。

15.1.4 λ 収益のバイアスと分散

15.1.4.1 λ のスイープ

どの λ が最もよく訓練できるだろうか。このダイヤルを動かすには、新しい重みと新しいクリティック目標が必要になる。アクターは GAE アドバンテージで重み付けし、クリティックは λ 収益、すなわちアドバンテージにその基準となった価値を加えたもの（`lambda_return` で計算する）に向かって適合させる。$\lambda = 0$ のとき、これは `td_weight` の 1 ステップ目標に一致する。15.2 節でも同じ方法でクリティックを適合させる。

```
@d2l.add_to_class(d2l.Batch)
def lambda_return(self, value_fn, gamma, lam):
    """The critic's target for GAE: the advantage plus the value it is
    measured from."""
    return self.gae(value_fn, gamma, lam) + value_fn(self.obs)
```

`gae_weight(lam)` はダイヤルの 1 つの設定に対する重み関数を返し、`gae_run` はそれを用いて 1 つのシードを訓練する。`td_weight` とは異なり、ここでは重みを標準化する。$\lambda = 1$ 付近では重みが数十の分散を持つモンテカルロアドバンテージのスケールになるため、生の重みのままだと大きな λ では上限に達し、小さな λ では達しないという不均衡が生じるからである。訓練予算は 4

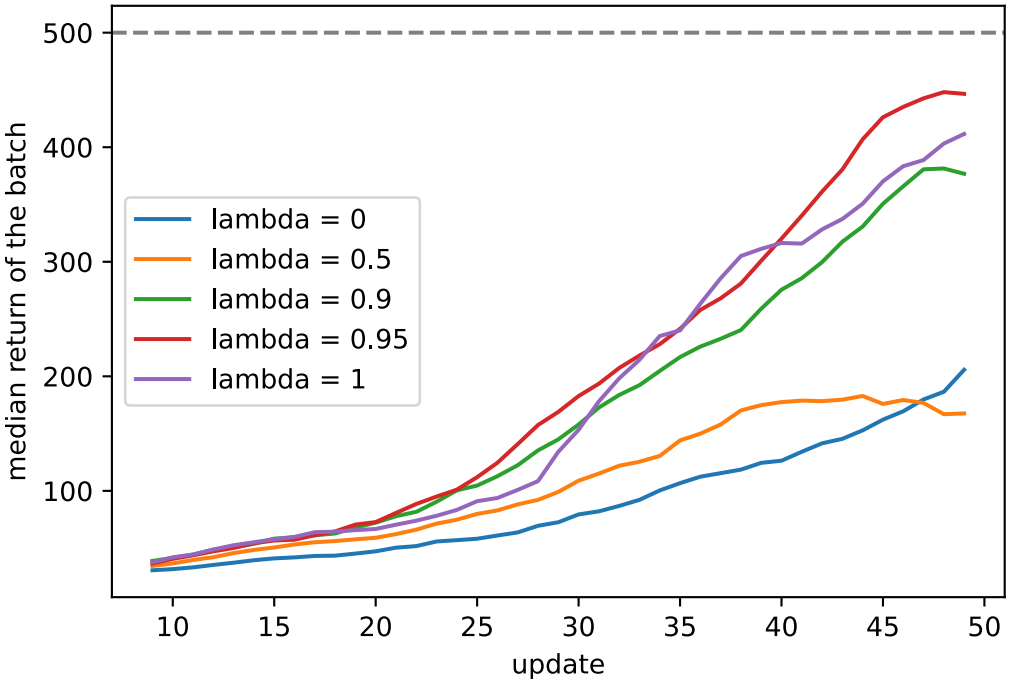
エピソードずつの更新 50 回と短く設定し、クリティックがまだ学習途上にあるエージェントをどの推定量が助けるかを比較する。

```
lams = [0.0, 0.5, 0.9, 0.95, 1.0]

def gae_weight(lam):
    """The weight of GAE(gamma, lam), standardized so that every lam steps
    alike; the critic first fits the lambda-return."""
    def weight(ac, b):
        for _ in range(critic_steps):
            d2l.fit_value(ac, b.obs, b.lambda_return(ac.value_np, gamma, lam))
        return d2l.standardize(b.gae(ac.value_np, gamma, lam))
    return weight

def gae_run(seed, lam):
    """The returns of one short run: 50 updates of four episodes."""
    return [ret for _, ret, _ in train(seed, cartpole_agent(seed),
                                       gae_weight(lam), num_updates=50,
                                       batch_episodes=4)]

sweep = {lam: d2l.run_seeds(gae_run, 5, lam=lam) for lam in lams}
```

sweep 内の各 λ について、20 更新移動平均による収益の最良値をシードごとに（シード間の中央値とともに）出力し、50 回の更新にわたるバッチ収益の中央値をプロットする。

```
for lam, r in sweep.items():
    best = np.array([np.convolve(c, np.ones(20) / 20, 'valid').max()
                     for c in r])
    print(f'lambda = {lam:4}: best 20-update window, '
          f'median {np.median(best):5.1f}, seeds {np.round(np.sort(best))}')
d2l.plot_curves({f'lambda = {lam:g}': np.median(r, axis=0)
                 for lam, r in sweep.items()},
                xlabel='update', ylabel='median return of the batch',
                smooth=10, reference=500)
```

```
lambda =  0.0: best 20-update window, median 163.7, seeds [141. 148. 164. 188. 281.]
lambda =  0.5: best 20-update window, median 167.4, seeds [133. 153. 167. 193. 351.]
lambda =  0.9: best 20-update window, median 336.9, seeds [180. 195. 337. 381. 422.]
lambda = 0.95: best 20-update window, median 362.1, seeds [213. 342. 362. 397. 443.]
lambda =  1.0: best 20-update window, median 351.7, seeds [267. 295. 352. 418. 455.]
```



シード間の中央値で見ると、$\lambda = 0$ と $\lambda = 0.5$ では 20 更新ウィンドウの最良値が約 150〜250 と上限を大きく下回るのに対し、$\lambda \geq 0.9$ ではその約 2 倍の高さに達する。個々のシードはこの境界をまたいで重なり合うため、ここでは中央値で比較している。短い訓練予算では訓練の大半の期間にわたってクリティックが不正確であり、クリティックへの依存が最も少ない推定量が最も速く学習する。これは先ほどの CartPole の比較を反対側から見たものにほかならない。すなわち CartPole では、未熟なクリティックが取り除く分散よりも、そのバイアスによる損失のほうが大きいのである。

15.1.4.2 固定方策プローブ

訓練曲線には、推定量の質だけでなく最適化や変化するデータの影響も混ざり合っている。推定量単独の性質を調べるため、途中まで訓練したエージェントを固定し、その固定されたパラメータのもとで多数のバッチを抽出して、各 λ による方策勾配の推定値を基準値と比較する。基準値には、全抽出にわたる $\lambda = 1$ の推定値の平均を用いる。エピソードが時間制限より前に終了する場合、これは環境のモデルなしに不偏なモンテカルロ勾配を近似する（打ち切られるエピソードについては演習問題 4 で扱う）。`grad_vec` は、1 バッチ分の重み付き対数確率の勾配を平坦なベクトルとして返す。

```
def grad_vec(ac, b, w):
    """The actor's gradient on batch b with weights w, as one flat vector."""
    g = actor_grads(ac, *d2l.padded(b.obs, b.act, np.float32(w)))
    return np.concatenate([np.asarray(x).ravel()
                           for x in jax.tree.leaves(g)])
```

次に、`freeze` は初期化したばかりのエージェントを `td_weight` で `warmup=40` 回のアクター・クリティック更新（通常の実行の約 4 分の 1）だけ訓練し、クリティックがまだ不正確な状態における方策勾配の推定値を調べられるようにする。

```
def freeze(seed, warmup=40):
    """An agent after `warmup` actor-critic updates: its critic still
    wrong."""
    ac = cartpole_agent(seed)
    for _ in train(seed, ac, td_weight, num_updates=warmup):
        pass
    return ac
```

probe は固定されたエージェントのもとで 4 エピソードからなるバッチを 200 個抽出し、各 λ について平均推定値のバイアス（基準値の長さに対する相対値）と単一推定値の分散（基準値の二乗長に対する相対値）を返す。バイアスの二乗と分散の和が、1 回の抽出における期待二乗誤差となる。

```
def probe(ac, seed, num_draws=200):
    """Relative bias and variance of the batch gradient for every lambda,
    against the mean of the lambda = 1 estimates."""
    rng, env = np.random.default_rng(seed), gym.make('CartPole-v1')
    env.reset(seed=seed)
    draws = {lam: [] for lam in lams}
    for _ in range(num_draws):
        b = d2l.rollout(env, ac.act, 4, rng)
        for lam in lams:
            draws[lam].append(grad_vec(ac, b, b.gae(ac.value_np, gamma, lam)))
    u = {lam: np.stack(d) for lam, d in draws.items()}
    u_ref = u[1.0].mean(axis=0)   # the many-rollout reference
    bias = [np.linalg.norm(u[l].mean(0) - u_ref) for l in lams]
    var = [((u[l] - u[l].mean(0)) ** 2).sum(1).mean() for l in lams]
    return (np.array(bias) / np.linalg.norm(u_ref),
            np.array(var) / (u_ref ** 2).sum())
```

1 つの固定エージェントは起こりうる多数の結果からの 1 サンプルにすぎず、あるエージェントにとって最良の λ はその個別のクリティックがどれだけ外れているかに依存する。そこで 8 個のエージェントを固定し、λ ごとにバイアス、分散、および同じエージェントの $\lambda = 1$ における誤差で割った誤差比のエージェント間中央値を出力する。

```
stats = np.array([probe(freeze(s), s) for s in range(8)])  # agent, stat, lam
err = stats[:, 0] ** 2 + stats[:, 1]
ratio = err / err[:, -1:]
for i, lam in enumerate(lams):
    print(f'lambda = {lam:4}: bias {np.median(stats[:, 0, i]):4.2f}, '
          f'variance {np.median(stats[:, 1, i]):4.2f}, '
          f'error / error at 1: {np.median(ratio[:, i]):4.2f}')
hits = np.isin(np.array(lams)[err.argmin(axis=1)], [0.9, 0.95]).sum()
print(f'{hits} of {len(stats)} agents have their smallest error at lambda = 0.9 '
      f'or 0.95')
```

```
lambda =  0.0: bias 0.99, variance 0.08, error / error at 1: 0.55
lambda =  0.5: bias 0.94, variance 0.07, error / error at 1: 0.52
lambda =  0.9: bias 0.81, variance 0.24, error / error at 1: 0.46
lambda = 0.95: bias 0.66, variance 0.53, error / error at 1: 0.48
lambda =  1.0: bias 0.00, variance 2.22, error / error at 1: 1.00
```

```
6 of 8 agents have their smallest error at lambda = 0.9 or 0.95
```

8 個の固定エージェント全体で見ると、誤差の中央値は $\lambda = 0.9$ または 0.95 で最小となり、$\lambda = 1$ における誤差のおよそ 2 分の 1 から 5 分の 3 になる。また $\lambda = 0$ における誤差よりも小さくなるが、そのマージンは実行ごとに大きくばらつく。$\lambda = 0$ から $\lambda = 1$ にかけてバイアスは 0 まで低下する一方、分散は 10 倍以上に増大し、その増大の大半は遠い将来の TD 誤差にかかる重み $(\gamma\lambda)^l$ が減衰しなくなる最後の 1 段階で生じる。個々のエージェントにはばらつきがあり、大半は 0.9 または 0.95 で最小誤差をとるが、残りのエージェントではすでに正確なクリティックに対して以下の命題が予測するとおり、より小さな λ が勝る。$\lambda = 1$ の行は自身の平均を基準値としているため、構成上バイアスが 0 となる。PPO の実装で広く使われるデフォルト値は $\lambda = 0.95$ である (Schulman et al. 2017)。

15.1.4.3 内部に最適点が生じる理由

なぜ最良の λ が両端の厳密に内側に位置するのだろうか。完全なクリティックを考えると、その対比から答えが明らかになる。

命題。クリティックが正確（$\hat{V} = V^\pi$）であり、エピソードに時間制限がないとする。このとき任意の λ は $A^\pi(s_t, a_t)$ の不偏推定値を与え、さらに次式が成り立つ。

$$\mathbb{E}\big[(\hat{A}_t^{\mathrm{GAE}})^2 \mid s_t, a_t\big] = \sum_{l \geq 0} (\gamma\lambda)^{2l}\, \mathbb{E}\big[\delta_{t+l}^2 \mid s_t, a_t\big],$$

したがって、推定値の誤差は $\lambda = 0$ で最小となる。

証明。クリティックが正確なとき、任意の状態における TD 誤差はその状態が与えられたもとで平均 0 となる。すなわち、前節の恒等式を行動について平均することにより $\mathbb{E}[\delta_{t+l} \mid s_{t+l}] = \sum_a \pi(a \mid s_{t+l})\, Q^\pi(s_{t+l}, a) - V^\pi(s_{t+l}) = 0$ が成り立つ。マルコフ性により、これは s_{t+l} までの全履歴（$l \geq 1$ のときは s_t、a_t、およびそれ以前のすべての TD 誤差を含む）が与えられたもとでも同様に成り立つ。ゆえに Equation 15.7 における $l \geq 1$ のすべての項は (s_t, a_t) が与えられたもとで平均 0 となり、推定値の平均は $\mathbb{E}[\delta_t \mid s_t, a_t] = A^\pi(s_t, a_t)$ となる。$k < l$ に対する交差項 $\mathbb{E}[\delta_{t+k}\, \delta_{t+l} \mid s_t, a_t]$ も同じ理由で消えるため、二乗和だけが残る。各二乗項は非負であり、その重み $(\gamma\lambda)^{2l}$ は λ とともに増大する一方、平均は変化しないため、平均二乗誤差は λ とともに増大する。■

正確なクリティックのもとでは、サンプリングされた末尾はノイズを加えるだけであり、1 ステップ TD 誤差が最良となる。したがって、内部の最適点はクリティックの誤差を映し出している。クリティックが不正確だからこそ内部に最適点が存在するのであり、クリティックが改善するにつれて最適点は $\lambda = 0$ へと移動するはずである（この点は演習問題 7 で固定方策のもとで検証する）。この意味で、$\lambda = 0.95$ というデフォルト値は、訓練中のクリティックが通常どの程度正確であるかを表す経験則といえる。

15.1.4.4 クリティックかグループベースラインか

学習されたクリティックは、アドバンテージを推定する 1 つの方法である。14.8 節ではもう 1 つの方法として、同じ初期状態からの複数のサンプルを互いに比較するグループベースラインを扱った。どちらを使うべきかは、本節で見たトレードオフから導かれる。報酬が毎ステップ得られ、エピソードが長く、状態の評価が安価な場面ではクリティックのほうが優れた推定量となる。学習に伴って誤差が縮小し、エピソードが終わる前に更新できるからである。一方、言語モデルがプロンプトに回答して検証器がその回答を採点する場合のように、各エピソードが単一の報酬で終わりサンプルが高価な場面では、グループベースラインのほうが優れた推定量となる。クリティックを使うなら各応答のあらゆるプレフィックスから最終評点を予測せねばならず、それ自体が

難しい回帰問題になるのに対し、同じプロンプトに対する他の応答を使えば 2 つ目のネットワークなしにベースラインが得られる。14.8 節で重みを導出した GRPO は、この選択をとっている。

実用されるアクター・クリティックの実装でも、エピソード全体を最後まで収集することはしない。多数の環境を並列に一定ステップ数だけ進め、エピソードが終了したかどうかにかかわらずそこで各セグメントを区切って更新する。この区切りは時間制限と同様に扱われる。すなわち、未終了セグメントの最後の TD 誤差は、`td_target` が打ち切りエピソードを処理するのと同じように区切り時点でのクリティックの価値からブートストラップし、`gae` の逆方向スキャンも各区切りでリセットされる。固定長のセグメントを用いることで、更新の間隔がエピソード長に依存しなくなり、エピソード単位のバッチでは活かされていなかったブートストラップの構造的な利点が発揮される。

以上で、単一の遷移から方策を更新し、サンプリングされた末尾をどれだけ残すかを選べるようになった。1 ステップなら TD 誤差、すべての深さの幾何重み付き混合なら λ 収益、そしてその混合を 1 回の逆方向パスで計算する GAE である。CartPole では、訓練の大部分においてクリティックが未熟であるため、末尾の大半を残す λ が 0.95 付近の選択が最良となる。もっとも、14.9 節からの制約がまだ 2 つ残っている。各バッチが 1 回のアクターステップに使われただけで破棄されることと、その 1 ステップが方策をどれだけ大きく動かすかに何の制限もないことである。15.2 節ではこの両方に取り組む。

15.1.5 まとめ

アクター・クリティック法は、サンプリングされた収益の代わりに、学習されたクリティックの TD 誤差によって方策勾配を重み付けする。正確なクリティックのもとでは TD 誤差は 1 遷移からなるアドバンテージの不偏サンプルとなるが、学習されたクリティックのもとではサンプリングされた末尾の分散をクリティックのバイアスと引き換えにし、クリティック自身の更新はサンプリングされた方策評価の半勾配ステップとなる。CartPole においてクリティックの計算予算を揃えると、未熟なクリティックが取り除く分散よりもそのバイアスの悪影響のほうが大きいため、モンテカルロ重みのほうが少ない環境ステップ数で上限に達する。ブートストラップの本質的な強みは、エピソードの終了を待たずに更新できる点にある。n ステップ収益と λ 収益はこれら 2 つの重みの間を補間し、GAE は λ 収益を TD 誤差の逆方向スキャンとして計算する。固定された方策のもとでは誤差は $\lambda = 0.9$〜0.95 付近で最小となるが、正確なクリティックがあれば $\lambda = 0$ で最小となるはずであり、内部の最適点はクリティックの誤差の大きさを表している。

15.1.6 演習問題

1. **ブートストラップされた重みのバイアス。** 14.8 節の条件付きゼロ平均恒等式によれば、現在の状態の任意の関数 $b(s_t)$ は、勾配にバイアスを生じさせることなくスコア $\psi_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ の重みから差し引くことができる。クリティックの誤差を $e(s) = \widehat{V}(s) - V^\pi(s)$ と書き、両方の価値が 0 である終端状態では $e = 0$ とする。
   1. Equation 15.2 の TD 誤差が $\delta_t = \delta_t^\pi + \gamma\, e(s_{t+1}) - e(s_t)$ と分解できることを示せ。ここで δ_t^π は正確なクリティック V^π の TD 誤差である。上の恒等式は 2 つの誤差項のうち一方を取り除く。もう一方の項については、恒等式の証明のどの段階が成り立たなくなるか。
   2. ステップ t の項 $\mathbb{E}[\delta_t\, \psi_t]$ が $\mathbb{E}[A^\pi(s_t, a_t)\, \psi_t]$ に対して持つバイアスは $\gamma\, \mathbb{E}[\bar{e}(s_t, a_t)\, \psi_t]$（ただし $\bar{e}(s, a) = \mathbb{E}[e(s_{t+1}) \mid s_t = s, a_t = a]$）であることを示せ。$e$ に関するどのような条件のもとでこのバイアスは消えるか。そのために $\widehat{V} = V^\pi$ は必要か。
   3. クリティックがすべての非終端状態を同じ定数 $c > 0$ だけ過大評価しているとする。一方の行動でエピソードが終了し、もう一方の行動では終了しない状態において、こ

のバイアスはどちらの行動を有利にするか。また $c < 0$ の場合はどう変わるか。毎ステップ 1 の報酬を与える CartPole では、初期化直後のクリティックは 0 付近の価値を予測する一方、V^{π} は正である。このモデルによれば、訓練初期のバイアスは方策をどの方向へ押しやると予測されるか。

2. [code] **最適なブートストラップ深さ**。本節のチェーンの例、すなわち $\sigma = 0.15$、$\gamma = 0.97$、クリティック誤差 $e(s) = 0.5\,(1 - s/9)^2$ を考える。
   1. バイアス $\gamma^n e(n)$ と分散 $\sigma^2(1-\gamma^{2n})/(1-\gamma^2)$ を、深さ n の目標（状態 0 から出発する場合）に対して導出せよ。さらにすべての深さについて 20,000 回のロールアウトをシミュレーションし、実測した平均二乗誤差を式と比較して最適点が $n = 4$ であることを確認せよ。
   2. クリティックの誤差をすべての非終端状態で定数 $e(s) = 0.5$ に置き換えよ。$n \le 9$ に対して平均二乗誤差が

$$\frac{\sigma^2}{1-\gamma^2} + \gamma^{2n}\Big(0.25 - \frac{\sigma^2}{1-\gamma^2}\Big), \tag{15.8}$$

   となり、任意の σ についてブートストラップする深さの範囲で単調であることを示せ。モンテカルロ目標 $n = 10$ を含めたときも単調なままであるか。このようなクリティックに対して最適となりうる深さはどれか、また $\sigma = 0.15$ で最適な深さはどれか。
   3. 先細りの誤差に戻し、$\sigma = 0.04$ および $\sigma = 0.3$ における最適な深さを求めよ。最適点が移動する方向を説明せよ。
3. [code] **クリティックのパス回数と更新の順序**。 `td_weight` はバッチごとに `critic_steps` 回の回帰パスをクリティックに与え、各パスの前に `td_target` を再計算する。
   1. `critic_steps` を $\{1, 5, 20\}$ としてそれぞれ 2 シードずつアクター・クリティック条件を再実行し、曲線をプロットせよ。パス回数は学習の速さと上限 500 付近での振る舞いをどのように変えるか。
   2. 代わりにバッチごとに 1 回だけ目標を計算し、`d2l.fit_value(ac, b.obs, target, num_steps=critic_steps)` で適合させよ。同じ 2 シードを 20 パスで訓練せよ。状態間の汎化を無視したとき、固定目標を用いた 1 バッチの間に、ある報酬が $\hat{V}$ に及ぼす影響はエピソードに沿って何ステップ後方まで伝わりうるか。また、各パスの前に目標を再計算する場合には高々何ステップまで伝わりうるか。これら 2 つの答えを用いて、20 パスにおける固定目標と再計算目標の曲線の違いを説明せよ。
   3. `delta` を計算する行をクリティックのパスより上に移動し、バッチ収集時点のクリティックの TD 誤差でアクターを重み付けするようにせよ。それぞれの順序で 10 シードずつ訓練し、最後の 10 バッチの平均が 300 を下回る実行の数を数えよ。1 パスの場合よりも 20 パスの場合でこの順序が重要になるのはなぜか。
4. λ **収益の漸化式**。$G_t^{\lambda} = \hat{A}_t^{\mathrm{GAE}} + \hat{V}(s_t)$ を、`gae_weight` がクリティックを適合させる目標とする。
   1. Equation 15.7 から、エピソード内で $\hat{A}_t^{\mathrm{GAE}} = \delta_t + \gamma\lambda\,\hat{A}_{t+1}^{\mathrm{GAE}}$ が成り立つことを示し、エピソードの最後ではない非終端ステップにおいて次式が成り立つことを導け。

$$G_t^{\lambda} = r_t + \gamma\big((1-\lambda)\,\hat{V}(s_{t+1}) + \lambda\,G_{t+1}^{\lambda}\big). \tag{15.9}$$

   終了するエピソードの最後のステップ、および時間制限で打ち切られるエピソードの最後のステップでは、この漸化式はそれぞれどのようになるか。
   2. エピソードが T ステップ後に打ち切られ、その最後の `next_obs` を s_T とする。このエピソードのステップ t において、`b.gae(value_fn, gamma, 1.0)` は `b.reward_to_go(gamma) - value_fn(b.obs)` をちょうど $\gamma^{T-t}\,\hat{V}(s_T)$ だけ上回ることを示せ。一般の λ について、G_t^{λ} の $\hat{V}(s_T)$ に関する導関数を計算せよ。$\lambda < 1$ のとき、打ち切り時点におけるクリティックの価値は打ち切られたエピソードのどのステップに影響するか。

3. δ_{t+l} が Equation 15.7 において持つ重みは $(\gamma\lambda)^l$ である。半減期 τ（この重みが $l = 0$ での値の半分まで低下するステップ数）を導出し、$\gamma = 0.99$ のもとで $\lambda \in \{0.5, 0.9, 0.95, 1\}$ に対してその値を求めよ。固定方策プローブが出力する分散は $\lambda = 0.95$ と $\lambda = 1$ の間で最も大きく増大する。これを半減期と関連づけて説明せよ。

Richard S. Sutton および Andrew G. Barto 著『Reinforcement Learning: An Introduction』の演習問題 12.1 および 12.2 を改変。

5. [extended] **完全オンライン更新。**●Equation 15.4 の更新は、バッチを一切使わずに遷移のたびに適用できる。すなわち、終了時にマスクされた単一の目標 $r_t + \gamma \hat{V}(s_{t+1})$ に向かうクリティックの 1 ステップと、`td_weight` と同様に δ_t そのもので重み付けされたアクターの 1 ステップを実行する。
   1. 長さ 1 の配列に `d2l.fit_value` と `policy_step_clip` を適用してこの変種を実装せよ。`train` によるアクター・クリティック 1 回分の環境ステップ数を数え、オンライン版にも同じステップ数を与えよ。`ActorCritic.mlp` のデフォルトの学習率と、その 10 分の 1 の学習率（`lr` として `cartpole_agent` 内に渡す）とでそれぞれ 3 シードずつ訓練し、バッチ化されたループの曲線と比較せよ。
   2. このとき連続する更新は 1 つのエピソードの連続する状態から生じる。何が問題になるか。リプレイバッファは過去の遷移を保存し、そこから一様に抽出したミニバッチで訓練する。これはそれらの失敗のうちどれに対処できるか。また、アクターのスコア関数による更新を再生された遷移上でそのまま訓練できないのはなぜか。
   3. 並列環境を用いれば、リプレイバッファなしにデータの相関を解消できる (Mnih et al. 2016)。8 個の CartPole 環境を同期して 1 ステップずつ進め、各環境から 1 つずつ集めた 8 遷移のスライスごとに更新せよ。これは先ほどの失敗のうちどれを解消し、どれを残すか。
6. [code] **勾配ノルムと上限。**●`policy_step_clip` が返すノルムは $\hat{\mathbf{g}} = \frac{1}{N}\sum_t \hat{A}_t \psi_t$ のノルムである。ここで $\hat{A}_t$ はアクターの重み（`mc_weight` の正規化されたモンテカルロアドバンテージ、および `td_weight` の生の TD 誤差）であり、$\psi_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ である。`runs` の最後の列には各更新におけるこのノルムが記録されている。
   1. `runs` 内の 2 条件についてノルムの中央値を比較せよ。N 個の項を独立同分布として扱い、次式を示せ。

$$\mathbb{E}[\|\hat{\mathbf{g}}\|^2] = \left\|\mathbb{E}[\hat{A}_t \psi_t]\right\|^2 + \frac{1}{N}\Big(\mathbb{E}[\|\hat{A}_t \psi_t\|^2] - \left\|\mathbb{E}[\hat{A}_t \psi_t]\right\|^2\Big). \tag{15.10}$$

   重みをシグナル $\mathbb{E}[\hat{A}_t \mid s_t, a_t] - \mathbb{E}[\hat{A}_t \mid s_t]$ と残余とに分解し、$\mathbb{E}[\hat{A}_t \psi_t]$ に寄与するのはシグナルのみであることを示せ。単位分散の重みにおいて残余が大きいと、ノルムはどうなるか。
   2. エージェント `freeze(0)` において、`d2l.rollout` で 8 エピソードからなるバッチを 50 個抽出せよ。各バッチについて、`grad_vec(ac, b, d2l.standardize(w))` のノルムを TD 誤差とモンテカルロアドバンテージ `b.reward_to_go(gamma) - ac.value_np(b.obs)` のそれぞれに対して計算し、さらにバッチ内で各重みベクトルをランダムに並べ替えた後にもう一度計算せよ。両方の重みを正規化することで、`td_weight` の生の TD 誤差が持つスケールの違いは取り除かれる。ランダムな並べ替えは何を取り除くか。各推定量のノルムは、並べ替え後のノルムと比べてどうなっているか。
   3. アクター・クリティック条件を変更し、バッチごとに `delta` の標準偏差（`td_weight` 内）と、1 ステップによるアクターのパラメータ変化量のノルム（`train` 内）を記録せよ。元のコードのまま、`clip=float('inf')` とした場合、重みを `d2l.standardize(delta)` とした場合の 3 つの規則のもとで 10 シードずつ訓練せよ。各規則のもとで最初の 10 回と最後の 50 回の更新におけるパラメータ変化量を比較し、移動平均が 475 に達した後に 300 を下回る実行の数を数えよ。記録した標準偏差と Adam の座標ごとのスケー

リングを用いて、クリティックのノイズへと縮小していく重みを正規化すると訓練終盤で悪影響が出うる理由、および Adam のもとで上限が何を変えるかを説明せよ。

7. [code] **GAE のバイアス。**●演習問題 1 では、重み δ_t のバイアスをクリティックの誤差 $e = \hat{V} - V^{\pi}$ を用いて求めた。本問ではこの結果を任意の λ へ拡張する。終了するエピソードを考え、終了時およびそれ以降では $e = 0$ とする。
   1. 次式が成り立つことを示せ。

$$\hat{A}_t^{\mathrm{GAE}} = \sum_{l \geq 0} (\gamma\lambda)^l \, \delta_{t+l}^{\pi} - e(s_t) + (1-\lambda) \sum_{k \geq 1} \gamma^k \lambda^{k-1} \, e(s_{t+k}), \tag{15.11}$$

   ここで δ^{π} は V^{π} の TD 誤差である。内部の最適点に関する命題とあわせて、勾配推定値のステップ t の項のバイアスが $\mathbf{b}(\lambda) = (1-\lambda) \sum_{k \geq 1} \gamma^k \lambda^{k-1} \, \mathbf{c}_k$（ただし $\mathbf{c}_k = \mathbb{E}[e(s_{t+k}) \, \psi_t]$）となることを導け。
   2. $\mathbf{c}_k = q^{k-1} \mathbf{c}_1$（固定ベクトル $\mathbf{c}_1$ およびある $0 \leq q \leq 1$ に対して）と仮定する。すなわち、後の状態におけるクリティック誤差に対する行動の影響が幾何級数的に減衰するものとする。$\|\mathbf{b}(\lambda)\| / \|\mathbf{b}(0)\| = (1-\lambda)/(1-\gamma\lambda q)$ を示し、`lams` 内の各値に対して $\gamma = 0.99$ のもとで $q = 1$ と $q = 0.9$ の場合にこの比を計算せよ。固定方策プローブの 8 個のエージェントそれぞれについて、各 λ におけるバイアスを $\lambda = 0$ におけるバイアスで割り、予測された比と比較せよ。どちらの q がよく当てはまるか。もし比が 1 を超えるなら、それはベクトル列 $\mathbf{c}_k$ について何を意味するか。
   3. `freeze(0)` の方策を固定したままクリティックのみを適合させ続けよ。すなわち、アクターのステップを行わずにクリティックのパスのみをさらに 50 バッチ実行し、プローブを繰り返せ。固定された方策のもとでクリティックをより長く適合させるにつれて、相対バイアスおよび 1 回抽出誤差を最小にする λ はどのように移動するか。実測された最適点は、正確なクリティックに対して命題が予測する $\lambda = 0$ に向かって移動するか。

15.2 信頼領域と近接方策最適化

14.9 節と 15.1 節のエージェントは、8 エピソードを収集して 1 回の勾配ステップを実行すると、そのバッチを破棄していた。エージェントがポールをうまくバランスさせられるようになると、この 8 エピソードは 4,000 ステップ分の経験に相当するが、そのすべてがたった 1 回の更新にしか使われない。しかも、そのステップには歯止めがない。大きな更新によって方策が大きく変化してしまうのを防ぐ仕組みはなく、次のバッチはその更新が生み出した方策でそのまま収集されることになる。1 つのバッチに対して複数回のステップを実行することはできるだろうか。また、どこまで更新を進めてよいのだろうか。

その答えは、パラメータに関するある事実から始まる。パラメータに同じだけの変化を加えても、出発点がどこにあるかによって、方策の振る舞いがほとんど変わらないこともあれば、一変してしまうこともある。つまり、パラメータ空間におけるステップサイズは、本来重要な量である振る舞いの変化を制御しない。2015 年、John Schulman らは、まさにその振る舞いの変化を制御対象に据えた。信頼領域方策最適化（trust region policy optimization、TRPO）は、そのようなステップがどれだけの改善を保証するかを示す定理に導かれ、訪問した状態における新旧の行動分布間のダイバージェンスを抑える (Schulman et al. 2015)。その 2 年後、近接方策最適化（proximal policy optimization、PPO）は、この制約を任意の勾配オプティマイザで実行できるクリップ付き目的関数に置き換えた (Schulman et al. 2017)。PPO は、シミュレーション上のロボットから人間のフィードバックに基づく言語モデルのファインチューニングに至るまで、標準的な方策勾配法となった (Ouyang et al. 2022)。

本節ではこの両方を構築する。まず 2 行動の方策を用いて、なぜパラメータ間の距離が不適切な

量なのかを示す。次に測度変換（change of measure）によって、古い方策で収集したバッチから新しい方策を評価できるようにする。この評価は厳密であるものの、エピソード長に伴って分散が増大するため、ステップごとに 1 つの比だけを残すことで再利用を現実的なものにする。続いて短い補題により、その近似が何を落としているかを明らかにする。TRPO はその誤差を抑え、PPO はそれをクリップする。最後に CartPole 上で、バッチの再利用によって環境ステップ数をどれだけ節約できるか、クリップがないと何が起きるか、そして方策の訓練中にどの数値を監視すべきかを確かめる。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

15.2.1 パラメータ空間と方策空間

更新するのは方策のパラメータであるから、パラメータ更新が方策そのものに何をもたらすかを問うべきである。単一のパラメータ θ と 2 つの行動を持つ方策族がその答えを与えてくれる。この例は、方策最適化に関する Joshua Achiam の講義に由来する (Achiam 2017)。次の方策を考えよう。

$$\pi_\theta(a) = \begin{cases} \sigma(\theta) & a = 1 \\ 1 - \sigma(\theta) & a = 2, \end{cases} \qquad \text{where } \sigma(\theta) = \frac{1}{1 + e^{-\theta}}. \tag{15.12}$$

異なる 2 つの出発点から、同じ大きさ $\Delta\theta = 2$ のパラメータ更新を 2 つ考えてみよう。

$$\pi_{\theta=0}(a{=}1) = \sigma(0) = 0.50 \xrightarrow{\Delta\theta=2} \sigma(2) = 0.88,$$
$$\pi_{\theta=6}(a{=}1) = \sigma(6) = 0.9975 \xrightarrow{\Delta\theta=2} \sigma(8) = 0.9997.$$

最初の更新は、エージェントを 2 つの行動に対して無差別な状態から、一方を強く選好する状態へと変化させる。2 つ目の更新では、エージェントの観察者が検出できるような変化は何も生じない。パラメータから方策への写像の傾きを見ればその理由は明らかである。$\sigma'(\theta) = \sigma(\theta)(1 - \sigma(\theta))$ は、$\theta = 0$ では 0.25 であるのに対し、$\theta = 6$ では約 0.0025 にすぎない。Figure 15.3 に両方の更新を示す。

この違いにより、オンポリシー訓練は不安定になる。$\theta = 0$ 付近でノイズの大きい大幅な更新が生じると、方策は飽和領域へと押し込まれうる。そこでは頻繁に選ばれる行動のスコアがほぼゼロになり（$\nabla_\theta \log \pi_\theta(a{=}1) = 1 - \sigma(\theta) \approx 0.0025$、$\theta = 6$ のとき）、もう一方の行動は -1 近くのスコアを持つものの、400 ステップに 1 回しかサンプリングされない。その結果、以降のオンポリシーのバッチには、この更新を覆すだけの材料がほとんど含まれなくなる。パラメータステップのユークリッド長を制限しても、これを防ぐことはできない。同じ長さであっても、$\theta = 6$ では無害である一方、$\theta = 0$ では劇的な変化をもたらすからである。

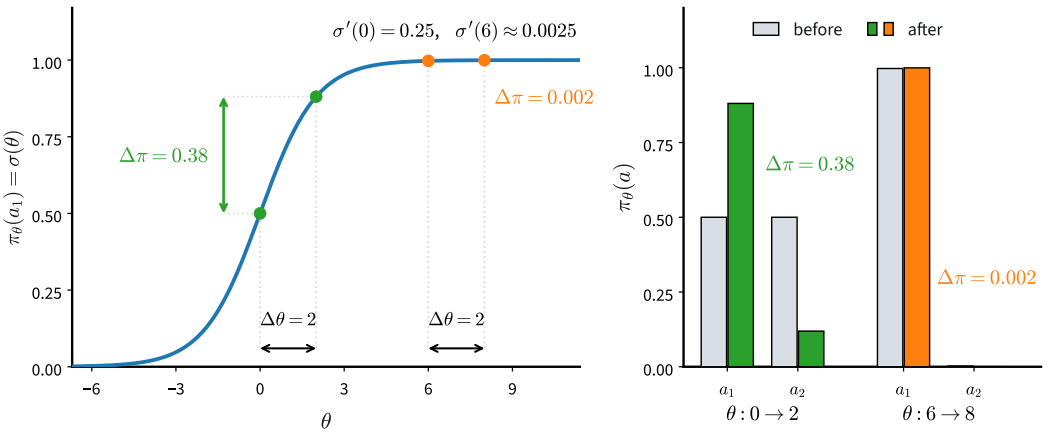


Figure 15.3: 同じ大きさ $\Delta\theta = 2$ の 2 つのパラメータ更新。左：両方の更新および各出発点における傾きを添えた写像 $\pi_\theta(a{=}1) = \sigma(\theta)$。右：各更新の前後における 2 つの行動分布。

したがって、更新ではパラメータ間の距離ではなく方策間の距離を制限すべきである。そのような制限のもとでは、オプティマイザは方策がほとんど変化しない領域で長いパラメータステップを取ることができ、さらに以下で見るように、同じバッチに対して複数回のステップを実行できるようになる。

15.2.2 重要度サンプリング

15.2.2.1 測度変換

14.7 節以降に登場したすべての推定量は現在の方策からサンプリングされた軌跡を用いていたため、バッチは 1 回の勾配ステップの後には使えなくなっていた。では代わりに、バッチが以前の方策 $\pi_{\theta_{\text{old}}}$ によって収集されたものだとしたらどうだろうか。そのバッチからでも、更新後の方策 π_θ がどれだけうまく振る舞うかを知ることができるだろうか。

重要度サンプリング（importance sampling）がその厳密な答えを与える。軌跡の任意の関数 f に対して、次が成り立つ。

$$\begin{aligned}\mathbb{E}_{\tau\sim P(\cdot;\,\theta)}\big[f(\tau)\big] &= \sum_{\tau} P(\tau;\theta)\, f(\tau) = \sum_{\tau} P(\tau;\theta_{\text{old}})\, \frac{P(\tau;\theta)}{P(\tau;\theta_{\text{old}})}\, f(\tau) \\ &= \mathbb{E}_{\tau\sim P(\cdot;\,\theta_{\text{old}})}\Big[\frac{P(\tau;\theta)}{P(\tau;\theta_{\text{old}})}\, f(\tau)\Big],\end{aligned} \tag{15.13}$$

この等式は、新しい方策が生成しうるすべての軌跡が古い方策のもとで正の確率を持つ限り成り立つ。ソフトマックス方策はどの行動にも確率ゼロを割り当てないため、この条件を満たす。収益を表す $f = R$ に対してこの恒等式を適用すると、目的関数は**古い**方策のもとでの期待値として書き直せる。

$$J(\theta) = \mathbb{E}_{\tau\sim P(\cdot;\,\theta)}\big[R(\tau)\big] = \mathbb{E}_{\tau\sim P(\cdot;\,\theta_{\text{old}})}\Big[\frac{P(\tau;\theta)}{P(\tau;\theta_{\text{old}})}\, R(\tau)\Big]. \tag{15.14}$$

このように、古い方策から得られたデータの各軌跡にその尤度比（likelihood ratio）で重み付けすれば、新しい方策の収益を統計的バイアスなしに推定できる。14.7 節で見たとおり、軌跡の確率は開始状態の確率、方策の行動確率、および環境の遷移確率の積である。この比では方策以外の因子がすべて約分されるため、重みの計算に環境のモデルは不要である。

$$\frac{P(\tau;\theta)}{P(\tau;\theta_{\text{old}})} = \prod_{t=0}^{T-1} \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{15.15}$$

15.2.2.2 爆発する積

この推定量は不偏であるが、その分散はエピソードの長さに伴って増大する。状態を無視し、$\pi_{\theta_{\text{old}}} = (0.5, 0.5)$ でデータを収集して $\pi_\theta = (0.6, 0.4)$ で評価される方策を考えよう。Equation 15.15 の各因子は等確率で 1.2 または 0.8 を取るため、その平均は 1、二乗平均は $(1.44 + 0.64)/2 = 1.04$ である。異なるステップの因子は互いに独立であるため、軌跡の重み W_T（ステップ数 T のエピソードに対する重み）は平均 1 を持ち、その分散は次のようになる。

$$\operatorname{Var}[W_T] = 1.04^T - 1,$$

この値は、$T = 10$ では約 0.5、$T = 100$ では約 50、CartPole の 500 ステップでは約 3×10^8 に達する。したがって、1 回の意思決定あたりデータ収集時の方策と 10 パーセントポイント異なる方策を、CartPole のエピソード全体から評価することはできない。大きな重みを持つごく少数の軌跡が推定値を決めてしまうからである。

15.2.2.3 ステップごとの代理目的関数

ステップ t における報酬はステップ t までに取った行動のみに依存するため、その報酬に対する厳密な補正に必要なのは比の積 $\rho_0 \cdots \rho_t$ だけである。ここで、各比は次で定義される。

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{15.16}$$

それ以前の比の積 $\rho_0 \cdots \rho_{t-1}$ は状態に対する補正であり、s_t の分布を古い方策のものから新しい方策のものへと変換する。これを落とし、s_t における行動に対する ρ_t だけを残せば、分散が t に伴って増大しない重みが得られる。このたった 1 つの近似により、古い方策の状態において新しい方策の行動を評価することになる。軌跡 i のステップ t におけるアドバンテージ推定値 $\hat{A}_t^i$ を用いると、得られる**代理目的関数**（surrogate objective）は、n 本の軌跡からなるバッチの全 $N = \sum_i T_i$ ステップにわたる平均となる。

$$\hat{L}(\theta) = \frac{1}{N} \sum_{i=1}^{n} \sum_{t=0}^{T_i - 1} \rho_t^i(\theta)\, \hat{A}_t^i. \tag{15.17}$$

コードで実際に計算するのは軌跡ごとではなくステップごとの平均であり、これによって以下で導入するダイバージェンス制約も状態にわたる平均となる。$\theta = \theta_{\text{old}}$ ではすべての比が 1 に等しく、$\nabla_\theta \rho_t = \nabla_\theta \log \pi_\theta(a_t \mid s_t)$ が成り立つため、そこでの $\hat{L}$ の勾配は 14.8 節の推定量と同じ方向を向く。すなわち、それ以降のすべての実装と同様に因子 γ^t を落としたうえで、各スコアをそのアドバンテージで重み付けしたものである。アドバンテージを平均 0 に標準化すれば、厳密に

$\widehat{L}(\theta_{\text{old}}) = 0$ となる。θ_{old} から離れるにつれて代理目的関数の精度は低下するため、次の問いはそれをどこまで信頼できるかである。

15.2.3 信頼領域

15.2.3.1 性能差補題

代理目的関数は、古い方策の状態において新しい方策の行動を評価する。この置き換えによって何が変わるかを確かめるため、2 つの収益を厳密に比較しよう。古い方策のアドバンテージを $A^{\text{old}}(s,a) = Q^{\pi_{\theta_{\text{old}}}}(s,a) - V^{\pi_{\theta_{\text{old}}}}(s)$ と書く。これは、その後に古い方策に従って行動するとき、行動 a が状態 s における古い方策の平均よりどれだけ優れているかを表す。次の補題は、終端状態（terminal state）で終了するエピソード、および $\gamma < 1$ の継続タスクに対して成り立つ。本節の CartPole の実行では、14.9 節と同様に 500 ステップの時間制限でブートストラップするため、最適化する目的関数は継続タスクのものとなる。

命題（性能差補題、Kakade and Langford (2002)）。

$$J(\theta) - J(\theta_{\text{old}}) = \mathbb{E}_{\tau \sim P(\cdot;\,\theta)}\Big[\sum_{t \ge 0} \gamma^t \, A^{\text{old}}(s_t, a_t)\Big]. \tag{15.18}$$

証明。V で $V^{\pi_{\theta_{\text{old}}}}$ を表すことにする。任意の軌跡に沿った和 $\sum_t \gamma^t (r_t + \gamma V(s_{t+1}) - V(s_t))$ は望遠鏡和となる。途中の V の項は対になって打ち消し合い、最後の項は終端状態で 0 になるか、$t \to \infty$（$\gamma < 1$ のとき）で消失する。残るのは $R(\tau) - V(s_0)$ である。ここで $\tau \sim P(\cdot;\,\theta)$ のもとで期待値を取る。(s_t, a_t) が与えられたとき、次の状態に関する $r_t + \gamma V(s_{t+1})$ の期待値は、古い方策のベルマン方程式により $Q^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ となるため、ステップ t の項は $\gamma^t \, \mathbb{E}\big[A^{\text{old}}(s_t, a_t)\big]$ となる。一方、$\mathbb{E}[R(\tau)] = J(\theta)$ であり、さらに $\mathbb{E}[V(s_0)] = J(\theta_{\text{old}})$ も成り立つ。両方策とも s_0 を同じ初期分布から引くからである。■

この補題は、新しい方策による改善量が、新しい方策自身の軌跡に沿って古い方策のアドバンテージを足し合わせたものに等しいことを示している。この和の項を状態ごとにまとめると、代理目的関数との比較が厳密になる。$d^{\pi}_{\gamma}(s) = (1-\gamma)\sum_{t \ge 0} \gamma^t \, \Pr(s_t = s)$ を 14.7 節の割引状態占有分布（discounted state-occupancy distribution）、すなわちステップ t に γ^t の重みを付けたもとで方策が訪問する状態の分布とする。このとき、次が成り立つ。

$$J(\theta) - J(\theta_{\text{old}}) = \frac{1}{1-\gamma} \sum_{s} d^{\pi_\theta}_{\gamma}(s) \sum_{a} \pi_\theta(a \mid s) \, A^{\text{old}}(s,a).$$

代理目的関数は、この和を新しい方策の状態占有分布ではなく古い方策の状態占有分布 $d^{\pi_{\theta_{\text{old}}}}_{\gamma}$ を用いて計算しており、この置き換えが唯一の誤差である。この誤差は状態分布がどれだけ動くかに依存し、それはパラメータがどれだけ動くかではなく、方策がどれだけ変化するかに依存する。

15.2.3.2 単調改善下界

直前の式の状態占有分布を置き換えることで、母集団版の代理目的関数が次のように定義される。

$$\bar{L}(\theta) = \frac{1}{1-\gamma} \sum_{s} d^{\pi_{\theta_{\text{old}}}}_{\gamma}(s) \sum_{a} \pi_\theta(a \mid s) \, A^{\text{old}}(s,a), \tag{15.19}$$

$\widehat{L}$ は、この式から因子 γ^t を除いた標本推定量にあたる。古い方策のアドバンテージはどの状態でも平均すると 0 になるため、$\bar{L}(\theta_{\text{old}}) = 0$ である。では、θ がどこまで動くと $\bar{L}$ は改善量を予測できなくなるのだろうか。その答えは、あらゆる状態における行動分布の最大の変化量を用いた下界として与えられる。

命題（Kakade and Langford (2002); Schulman et al. (2015)）。$\gamma < 1$ **とし、**$A_{\max} = \max_{s,a} |A^{\textbf{old}}(s,a)|$ **とおくと、次が成り立つ。**

$$\begin{aligned} J(\theta) \;\geq\; & J(\theta_{\text{old}}) + \bar{L}(\theta) \\ & - \frac{4\gamma A_{\max}}{(1-\gamma)^2} \max_{s} D_{\text{KL}}\big(\pi_{\theta_{\text{old}}}(\cdot \mid s) \,\|\, \pi_{\theta}(\cdot \mid s)\big). \end{aligned} \tag{15.20}$$

代理目的関数は、あらゆる状態における新旧の行動分布間のカルバック・ライブラー・ダイバージェンスの最大値に比例するペナルティの範囲内で正しい。演習問題 6 では、行動をカップリングして 2 つの方策を並走させることでこの下界を導出する。$\theta = \theta_{\text{old}}$ において下界の両辺はともに $J(\theta_{\text{old}})$ に等しいため、右辺を引き上げるステップは真の収益も必ず引き上げる。すなわち、この下界を最大化すれば単調改善が得られる。ただし、この定数はそのまま使うには大きすぎる。$\gamma = 0.99$ では $A_{\max}$ の $4 \cdot 0.99/0.01^2 \approx 40{,}000$ 倍にもなるため、この下界が保証できるステップはごく小さなものにとどまる。

TRPO はこの下界の形状を保ちつつ、その構成要素をバッチから推定できる量に置き換える。すなわち、バッチ内の状態にわたる平均ダイバージェンスを予算 Δ_{KL} 以下に抑えるという制約のもとで、標本代理目的関数を最大化する。

$$\begin{aligned} & \max_{\theta} \; \widehat{L}(\theta) \\ & \text{subject to } \frac{1}{N}\sum_{i,t} D_{\text{KL}}\big(\pi_{\theta_{\text{old}}}(\cdot \mid s_t^i) \,\|\, \pi_{\theta}(\cdot \mid s_t^i)\big) \leq \Delta_{\text{KL}}, \end{aligned} \tag{15.21}$$

ここでは、全状態にわたる最大値の代わりに平均を用い、保守的な係数の代わりに選んだ予算（典型的には $\Delta_{\text{KL}} = 0.01$）を用いている。TRPO はこの問題を二次手法で解くが、これについては演習問題 7 で本節の構成要素から構築する。Figure 15.4 に、なぜこの制約を方策空間に置く必要があるかを示す。

15.2.3.3 クリップ付き目的関数

PPO は、制約付きの二次最適化問題を、上限が組み込まれた一次の目的関数に置き換える。PPO は比そのものをクリップするのではなく、比が 1 の周りの半幅 ϵ_{clip} のバンドを超えた時点で、そのサンプルからそれ以上の利得を与えないようにする。

$$\begin{aligned} L^{\text{CLIP}}(\theta) = \frac{1}{N}\sum_{i,t} \min\Big(& \rho_t^i(\theta)\,\widehat{A}_t^i, \\ & \text{clip}\big(\rho_t^i(\theta),\, 1-\epsilon_{\text{clip}},\, 1+\epsilon_{\text{clip}}\big)\,\widehat{A}_t^i \Big). \end{aligned} \tag{15.22}$$

1 つのサンプルについて考えよう。$\widehat{A}_t > 0$ の場合、その寄与は ρ_t とともに増加するが、それは $\rho_t = 1 + \epsilon_{\text{clip}}$ までであり、そこを超えるとクリップされた項のほうが小さくなるため勾配は 0 になる。$\widehat{A}_t < 0$ の場合のしきい値は $1 - \epsilon_{\text{clip}}$ である。通常の $\epsilon_{\text{clip}} = 0.2$ を用い、古い方策が確率

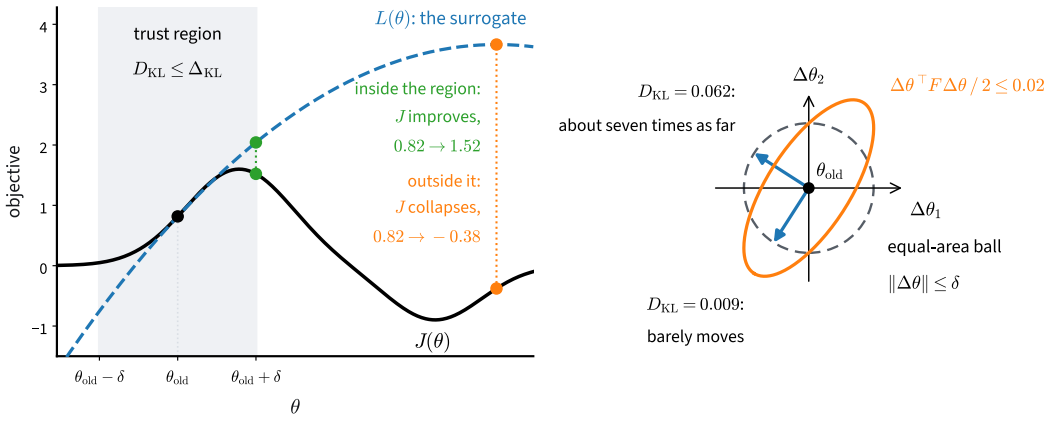


Figure 15.4: 方策更新の制限。左：代理目的関数 L は真の目的関数 J に θ_{old} で接しているが、方策が大きく変化すると不正確になる。L の制約なし最大点は J を 0.82 から -0.38 へと下げてしまうのに対し、網掛けの信頼領域 $D_{\mathrm{KL}} \le \Delta_{\mathrm{KL}}$（1 パラメータの場合は区間 $\theta_{\mathrm{old}} \pm \delta$）内の最良点は 1.52 へと引き上げる。右：3 行動のソフトマックス方策において、ダイバージェンス制約の二次近似 $\frac{1}{2}\Delta\theta^\top F \Delta\theta \le 0.02$（$F$ は θ_{old} におけるダイバージェンスのヘッセ行列）はパラメータ空間内の楕円となる。ユークリッド長が等しい 2 つのパラメータステップが、それぞれ 0.009 と 0.062 のダイバージェンスだけ方策を変化させており、同じパラメータ距離でも一方の方向では他方の約 7 倍大きく方策が動く。

0.5 で選んだ行動を考えると、正のアドバンテージを持つサンプルは新しい方策がその行動に確率 0.6 を与えた時点で更新への上積みをやめ、負のアドバンテージを持つサンプルは確率が 0.4 まで下がった時点で上積みをやめる。最小値を取ることで、この制限は片側のみに働く。すなわち、目的関数を悪化させる方向の変化は常に反映される一方、望ましい方向へのそれ以上の移動は何も加えない。Figure 15.5 に、アドバンテージの正負両方について 1 サンプルの寄与を示す。

比そのものがクリップされるわけではないため、2 つの理由から比は依然としてバンドの外へ出ることがある。第 1 に、すべてのサンプルがネットワークのパラメータを共有しているため、あるサンプルのためのステップが他のサンプルの比も動かしてしまうこと、第 2 に、あるサンプルの勾配が 0 になった後も Adam のモメンタムが数ステップにわたってパラメータを動かし続けることである。Engstrom et al. (2020) は、PPO の比が実際にバンドの外へ出ること、また更新あたりのダイバージェンスがいかなる一定の予算以下にも抑えられていないことを実測した。したがって、クリップは制約というより誘因の設計であり、Equation 15.20 とは異なり改善を保証するものではない。それでも、個々の行動確率の際限ない変化を促すことなく、1 つのバッチに対して複数パスの最適化を実行することを可能にする。

15.2.3.4 エントロピーボーナス

多くの実装では、バッチ内の状態における行動分布の平均エントロピーに重み `entropy_coef` を掛けた小さな**エントロピーボーナス**を加える。このボーナスは、2 行動の例で見たように頻繁に選ばれる行動のスコアが小さくなって回復が遅れる 0 や 1 の付近から、確率を遠ざける働きをする。K 個の行動上の分布のエントロピーは $\log K$ から一様分布とのダイバージェンスを引いたものに等しいため、このボーナスは一様方策からの距離に対するペナルティとなっている。

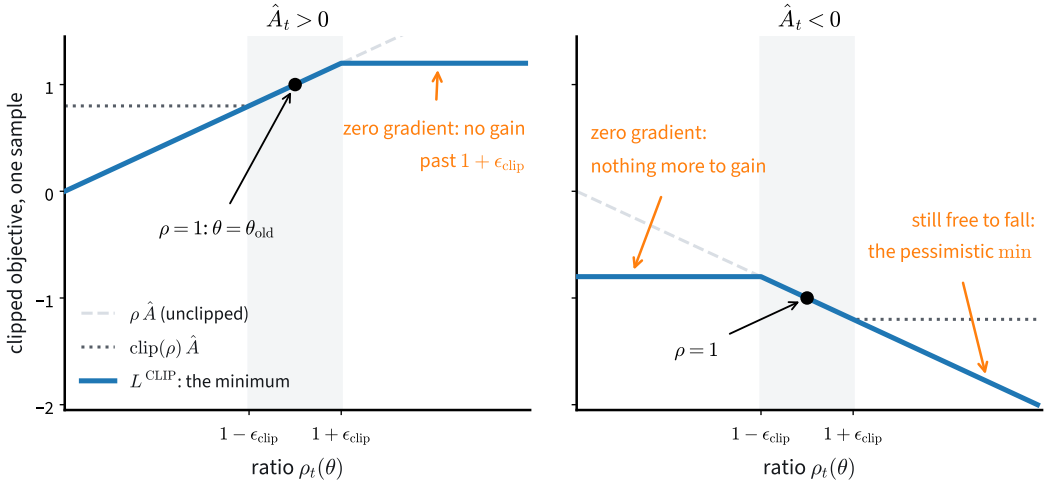


Figure 15.5: クリップ付き目的関数 Equation 15.22 に対する 1 サンプルの寄与を比 ρ_t に対してプロットしたもの。正のアドバンテージでは $1+\epsilon_{\text{clip}}$ を超えると増加が止まり、負のアドバンテージでは $1-\epsilon_{\text{clip}}$ を下回ると増加が止まる。1 を挟んだ反対側では最小値演算によりクリップなしの項が残るため、目的関数を下げる変化が隠されることはない。

15.2.4 PPO の実践

15.2.4.1 アドバンテージ推定

Equation 15.22 の $\hat{A}$ には、どのようなアドバンテージ推定を用いてもよい。ここでは 15.1 節の一般化アドバンテージ推定（generalized advantage estimation、GAE）を用いる。`Batch.gae` は、クリティックの TD 誤差 $\delta_t = r_t + \gamma\hat{V}(s_{t+1}) - \hat{V}(s_t)$ を重み $(\gamma\lambda)^k$ で各エピソードを逆向きに足し合わせるものであり、λ が各アドバンテージのどれだけ先まで見通すかを定める。同節では、この推定の誤差は $\lambda = 0.9$ から 0.95 付近で最小になることが示されたが、0.95 は PPO の原論文の設定でもある。アドバンテージは、クリティックをそのバッチに当てはめる前の時点のままで計算する。これは GAE の原論文の順序に従ったものである (Schulman et al. 2016)。15.1 節の `td_weight` では逆の順序を用いたが、それはアクターの重みがクリティックに全面的に依存する 1 ステップ TD 誤差だったからである。$\lambda = 0.95$ では、アドバンテージの大半はバッチ自身のサンプリングされた報酬からなり、クリティックへの依存は小さいため、この順序によって本節の結果が変わることはなく、原論文に従うことにする。

15.2.4.2 実装

CartPole の実験設定は 15.1 節のものを引き継ぐ。エージェントは 14.9 節の `ActorCritic.mlp(4, 2)` であり、それぞれ 64 個の tanh ユニットからなる 1 つの隠れ層と、学習率 10^{-2} の独自の Adam オプティマイザを持つ方策ネットワークと価値ネットワークで構成される。各更新では 8 エピソードからなるバッチを収集し、クリティックはその上で `critic_steps = 20` 回の回帰パスを実行する。新しく加わるハイパーパラメータは、各バッチに対するアクターのパス数 `num_epochs = 20`、バンドの半幅 `epsilon_clip = 0.2`、およびエントロピー重み `entropy_coef = 0.01` の 3 つである。20 パスという設定は RL Baselines3 Zoo における CartPole の設定でもあるが (Raffin et al. 2021)、そちらの学習率は本節の 10 分の 1 にあたる 10^{-3} である。本節の学習率では 1 パスごとに方策が大きく動くため、訓練が高速になる一方でクリップが不可欠となる設定になっており、クリップを外すとどうなるかは後のアブレーション実験で確かめる。

```
gamma, lam, num_updates, batch_episodes = 0.99, 0.95, 60, 8
num_seeds, num_epochs, critic_steps = 8, 20, 20
epsilon_clip, entropy_coef = 0.2, 0.01
def cartpole_agent(seed):
    return d2l.ActorCritic.mlp(4, 2, seed=seed)
```

サンプルごとの目的関数は、更新処理に関数として渡す。clipped_surrogate は Equation 15.22 であり、項 $\rho_t \hat{A}_t$ のみからなる plain_surrogate は後のアブレーション実験で用いる。いずれも引数としてバンドの半幅 ϵ_{clip} を受け取るが、実際にそれを使うのは前者のみである。

```
def clipped_surrogate(rho, adv, epsilon):
    """PPO's per-sample objective eq_ppo_clip: min(rho A, clip(rho) A)."""
    return jnp.minimum(rho * adv,
                       jnp.clip(rho, 1 - epsilon, 1 + epsilon) * adv)

def plain_surrogate(rho, adv, epsilon):
    """The unclipped per-sample objective rho A."""
    return rho * adv
```

ppo_epochs は、バッチの状態と行動に加え、再利用の前に固定した 2 つの量、すなわちアドバンテージと収集方策の対数確率 logp_old を受け取る。そして、符号を反転した平均代理目的関数からエントロピーボーナスを引いた損失に対して、num_epochs ステップだけアクターのオプティマイザ opt_pi を進める。エントロピーの計算には各状態における分布全体が必要になるため、更新処理では全行動にわたる対数ソフトマックスを一度計算し、そこから $\log \pi_\theta(a_t \mid s_t)$ を読み取る。各パスの前には診断指標を 1 行記録する。その内容は、バンド外に出た比の割合、$\log \pi_{\theta_{\text{old}}} - \log \pi_\theta$ の標本平均、および平均エントロピーである。2 つ目の数値は**近似 KL**（approximate KL）と呼ばれ、古い方策のもとでの期待値は訪問した状態におけるダイバージェンス $D_{\text{KL}}(\pi_{\theta_{\text{old}}} \| \pi_\theta)$ に一致する（もっとも、有限のバッチ上では負の値になることもある）。この行はパスの前に記録されるため、最初の行は常に 0、0、そして古い方策のエントロピーとなる。

```
@nnx.jit(static_argnames='surrogate')
def _ppo_step(policy, opt, obs, act, adv, logp_old, mask, epsilon,
              entropy_coef, surrogate):
    """One step on the mean surrogate plus an entropy bonus over the real
    rows of a padded batch; returns, measured before the step, the fraction
    of ratios outside the band, the approximate KL and the mean entropy."""
    mean = lambda x: (mask * x).sum() / mask.sum()
    def loss_fn(policy):
        logp_all = jax.nn.log_softmax(policy(obs), axis=-1)
        logp = jnp.take_along_axis(logp_all, act[:, None], -1).squeeze(-1)
        rho = jnp.exp(logp - logp_old)
        entropy = -(jnp.exp(logp_all) * logp_all).sum(-1)
        loss = -mean(surrogate(rho, adv, epsilon) + entropy_coef * entropy)
        return loss, (rho, logp, entropy)
    (_, (rho, logp, entropy)), grads = nnx.value_and_grad(
        loss_fn, has_aux=True)(policy)
    opt.update(policy, grads)
    return jnp.stack([mean(jnp.abs(rho - 1) > epsilon),
                      mean(logp_old - logp), mean(entropy)])
```

```
def ppo_epochs(ac, obs, act, adv, logp_old, epsilon, num_epochs,
               entropy_coef=0.01, surrogate=clipped_surrogate):
    """num_epochs passes on the mean surrogate plus an entropy bonus, on one
    frozen batch of states and actions; returns [num_epochs, 3] numpy
    diagnostics, recorded before each pass: fraction of ratios outside the
    band, approximate KL, mean policy entropy."""
    batch = d2l.padded(obs, act, adv, logp_old)
    step = nnx.cached_partial(_ppo_step, ac.policy, ac.opt_pi)
    return np.asarray(jnp.stack([
        step(*batch, epsilon, entropy_coef, surrogate=surrogate)
        for _ in range(num_epochs)]))
```

ppo_epochs は d2l.padded（オンライン付録「ユーティリティ関数とクラス」）を用いてバッチをバケット長までパディングし、同時に実際のデータ行を示すマスクも返す。_ppo_step はあらゆる平均計算からパディング行を除外するため、コンパイル済みのステップを長さの異なるバッチ間で再利用できる。代理目的関数はコンパイル済みステップの静的引数となっており、2 つの代理目的関数はそれぞれ 1 回ずつコンパイルされる。診断指標はすべてのパスが完了するまでデバイス上に保持される。

訓練ループは 15.1 節のアクター・クリティックのループにおいて、単一のアクターステップを ppo_epochs に置き換え、クリティックのパスの前にアドバンテージを計算するようにしたものである。まず rollout が 8 エピソードを実行して Batch に格納する。続いてループは Batch.gae と現在のクリティックを用いてアドバンテージを計算し、standardize によってバッチ全体で標準化する。また、logp_old を log_prob_np で読み取る。この関数は与えられた状態と行動に対する $\log \pi(a \mid s)$ を NumPy 配列として返す。次にクリティックが、15.1 節の Batch.lambda_return と 14.9 節の fit_value を用いて λ 収益のターゲットに向けた回帰パスを実行し、最後にアクターが自身のパスを実行する。各更新は、そこまでに費やした環境ステップ数とバッチの平均収益を順に返す。各バッチの診断指標は呼び出し元から渡されたリスト diag に追加され、CLIPPED、KL、ENTROPY がその 3 つの列を表す。

```
CLIPPED, KL, ENTROPY = 0, 1, 2   # the columns of ppo_epochs' diagnostics

def train_ppo(seed, ac, diag, surrogate=clipped_surrogate,
              num_epochs=num_epochs):
    """PPO on CartPole: freeze the advantages and the collecting policy's
    log-probs, then take num_epochs surrogate passes on each batch. Appends
    each batch's diagnostics to diag; yields (env steps, mean return)."""
    rng, env = np.random.default_rng(seed), gym.make('CartPole-v1')
    env.reset(seed=seed)
    steps = 0
    for _ in range(num_updates):
        b = d2l.rollout(env, ac.act, batch_episodes, rng)
        adv = d2l.standardize(b.gae(ac.value_np, gamma, lam))  # pre-fit critic
        logp_old = ac.log_prob_np(b.obs, b.act)
        for _ in range(critic_steps):   # fresh lambda-return target, per pass
            d2l.fit_value(ac, b.obs, b.lambda_return(ac.value_np, gamma, lam))
        diag.append(ppo_epochs(ac, b.obs, b.act, adv, logp_old, epsilon_clip,
                               num_epochs, entropy_coef, surrogate))
        steps += len(b)
        yield steps, float(b.episode_returns().mean())
```

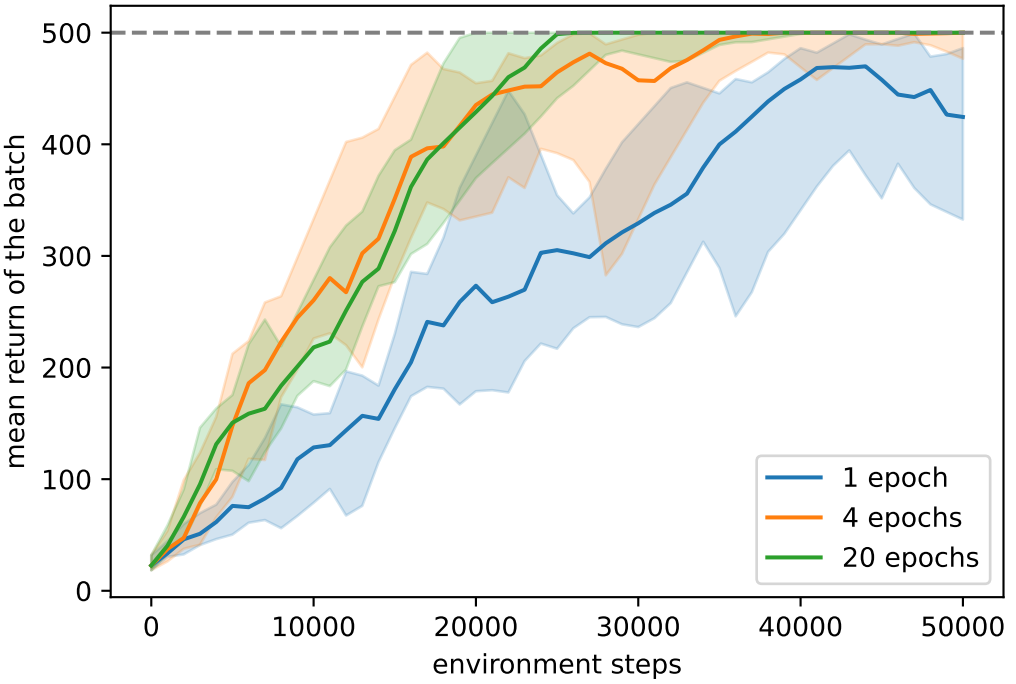
15.2.4.3 バッチの再利用と環境ステップ数

バッチの再利用によって環境ステップ数はどれだけ節約できるだろうか。バッチあたり 1、4、20 パスのクリップ付き PPO をそれぞれ 8 個のシードで訓練し、環境ステップ数が 50,000 に達した時点で各実行を打ち切って、そこまでに費やしたステップ数に対してバッチの平均収益をプロットする。1 パスの場合はすべての比が 1 に等しくクリップは一度も働かないため、この条件は 15.1 節の λ スイープに近く、エントロピーボーナスがあり、勾配の上限がなく、クリティックのパスの前にアドバンテージを計算する点だけが異なる。同節と同様に、`d2l.on_grid` を用いて各実行を共通のステップグリッド上に揃える。これらの実行では診断指標は不要なため、それぞれに空のリストを渡す。

```
budget = 50_000
grid = np.arange(0, budget + 1, 1000)
reuse = {}
for e in (1, 4, 20):
    runs_e = []
    for s in range(num_seeds):
        run = []
        for row in train_ppo(s, cartpole_agent(s), [], num_epochs=e):
            run.append(row)                  # (env steps, mean return)
            if row[0] >= budget:
                break
        runs_e.append(np.array(run))
    reuse[f'{e} epoch' + 's' * (e > 1)] = d2l.on_grid(runs_e, grid, 1)
```

`reuse` に格納された補間済みの収益曲線を `grid` に対してプロットすると、バッチの再利用によって追加の勾配パスと引き換えに環境との相互作用をどれだけ減らせるかがわかる。

```
d2l.plot_curves(reuse, xlabel='environment steps',
                ylabel='mean return of the batch', x=grid, reference=500)
```



各バッチを 4 エポック再利用すると、1 エポックの場合に必要な環境ステップ数（50,000 以上）の約半分にあたる約 25,000 ステップ、更新回数にしても約半分で上限に到達する（50,000 ステップの予算内では、1 エポックの中央値は上限のわずかに手前にとどまる）。さらに 20 エポックまで増やしてもそれに比べれば変化は小さく、クリップ付きの実行が崩壊することもない。この改善はバッチそのものに由来する。数千ステップからなるバッチには 1 回の勾配ステップでは抽出しきれない情報が含まれており、クリップがあるおかげでオプティマイザはどの行動確率も動かしすぎることなく複数回のステップを実行できるのである。

15.2.4.4 クリッピングのアブレーション

改善の大半を得るには 4 パスで十分であるが、本節の学習率で 20 パスを実行すると、1 つのバッチ内で方策は大きく動く。この領域でクリップは重要だろうか。同じ 8 個のシード上でクリップを有効にした 20 エポックの PPO を訓練し、後の検証用に訓練済みエージェントを、次の項のために全診断指標を保存しておく。

```
arms = {'clipped (PPO)': clipped_surrogate, 'no clip': plain_surrogate}
agents = {arm: [cartpole_agent(s) for s in range(num_seeds)] for arm in arms}
runs, diag = {}, {arm: [] for arm in arms}
runs['clipped (PPO)'] = np.array(
    [list(train_ppo(s, agents['clipped (PPO)'][s], diag['clipped (PPO)']))
     for s in range(num_seeds)])
```

続いて対照実験として、代理目的関数をクリップなしにする以外はすべての行が同一の実行を走らせる。その後、各条件の診断指標を（シード、更新、エポック、列）の形状に変形する。

```
runs['no clip'] = np.array(
    [list(train_ppo(s, agents['no clip'][s], diag['no clip'],
                    surrogate=plain_surrogate))
     for s in range(num_seeds)])
diag = {arm: np.reshape(d, (num_seeds, num_updates, num_epochs, 3))
        for arm, d in diag.items()}
```

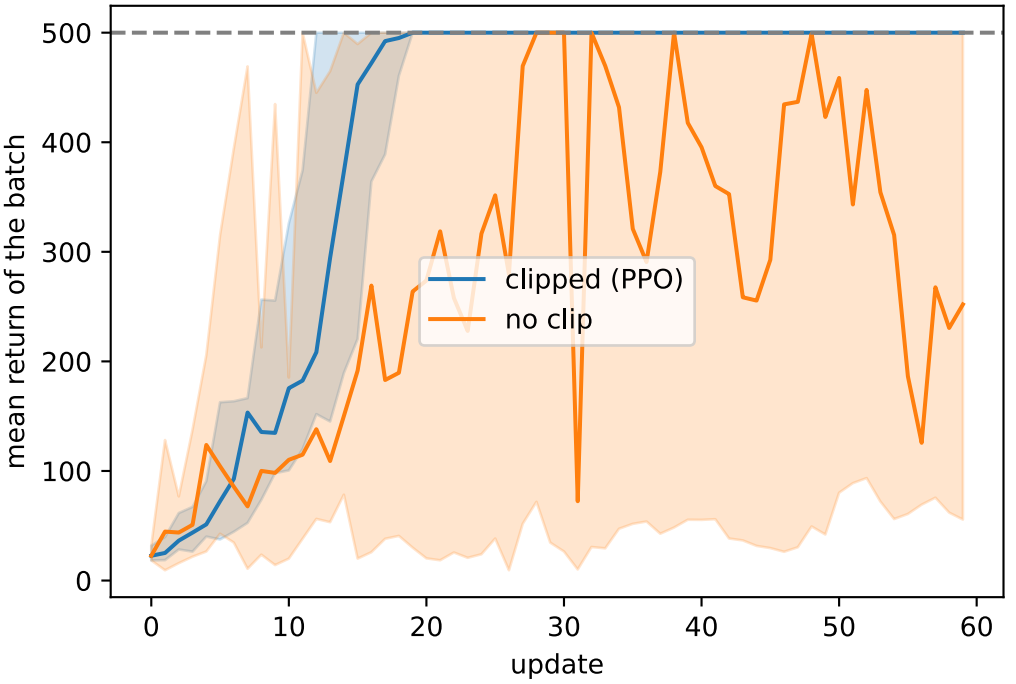
各実行を 2 つの基準で要約しよう。バッチ収益が最初に 475 に到達した後、最大値の半分である 250 を一度も下回らなければ、その実行はタスクを**維持**したとみなす。一方、最後の 10 回の更新における収益の平均が 100 を下回った場合、その実行は**破綻**に終わったとみなす。

```
def holds(ret):   # reaches 475, then never falls below 250 again
    hit = np.flatnonzero(ret >= 475)
    return len(hit) > 0 and ret[hit[0]:].min() >= 250

for name, r in runs.items():
    held = sum(holds(ret) for ret in r[:, :, 1])
    dead = r[:, -10:, 1].mean(axis=1) < 100
    print(f'{name:>13}: {held} of {num_seeds} seeds hold the task, '
          f'{int(dead.sum())} end dead')
```

```
clipped (PPO): 8 of 8 seeds hold the task, 0 end dead
      no clip: 1 of 8 seeds hold the task, 1 end dead
```

全更新にわたるバッチ収益の軌跡 `r[:, :, 1]` をプロットすると、`'clipped (PPO)'` の安定したプラトーと `'no clip'` の崩壊が鮮明に対比される。

```
d2l.plot_curves({name: r[:, :, 1] for name, r in runs.items()},
                xlabel='update', ylabel='mean return of the batch',
                reference=500)
```



20 エポック、学習率 10^{-2} の設定では、クリップ付きのシードはすべて上限に到達してそのままタスクを維持する。一方、クリップなしではタスクを維持できるシードはほとんどなく、一部は破綻に終わる。ノートブック外で 64 個のシードにわたって検証すると、タスクを維持できる実行は 20 回に 1 回程度にすぎず、約半数は破綻に終わる。しかもその大半は、毎ステップ同じ方向にカートを押し続けたときのエピソード長にあたる約 9 の収益に張り付く。8 個のシードではこれらの割合を安定して示すには少なすぎるため、上で出力される破綻した実行の数が半数を大きく下回ることもある。クリップなしのパスを繰り返すと方策は飽和領域へと押し込まれ、2 行動の例で見たように選ばれた行動の対数確率の勾配が小さくなる。すると次のバッチはその飽和した方策によって収集されるため、それを覆すような行動がほとんど含まれなくなるのである。PPO の原論文におけるアブレーション実験でも同じ結果が得られており、クリップもペナルティも持たない変種は検証されたすべての手法の中で最悪であった (Schulman et al. 2017)。クリップを必要としている原因はこの大きな学習率にある。アクターの学習率を 10 分の 1 の 10^{-3} に下げれば、クリップなし PPO でも 10 個のシードすべてで崩壊しなくなり、クリップ付き PPO と同じ速さで上限に到達する。さらに 3×10^{-4} ではクリップはほとんど働かない。ただし、どちらの学習率も本節の設定より遅く、本節の約 24,000 ステップに対して、上限到達までにそれぞれ約 35,000〜40,000 ステップおよび約 85,000〜90,000 ステップを要する。大きな学習率が訓練を高速にし、クリップがその大きな学習率を安全にしているのである。

クリップはどのくらいの頻度で働くのだろうか。各更新ではエポックごとにすべてのサンプルを 1 回ずつ検査し、最初のエポックはすべてのサンプルで $\rho_t = 1$ から始まる。エポックごとの全検査のうちバンド外に出た割合と、各バッチの最終エポックにおける同じ割合を出力してみよう。

```
for name, d in diag.items():
    print(f'{name:>13}: ratio checks outside the band: '
          f'{d[..., CLIPPED].mean():.1%} across all epochs')
    print(f'{"":>13}  {d[:, :, -1, CLIPPED].mean():.1%} at the last epoch '
          f'of each batch')
```

```
clipped (PPO): ratio checks outside the band: 4.9% across all epochs
               4.9% at the last epoch of each batch
      no clip: ratio checks outside the band: 24.9% across all epochs
               37.6% at the last epoch of each batch
```

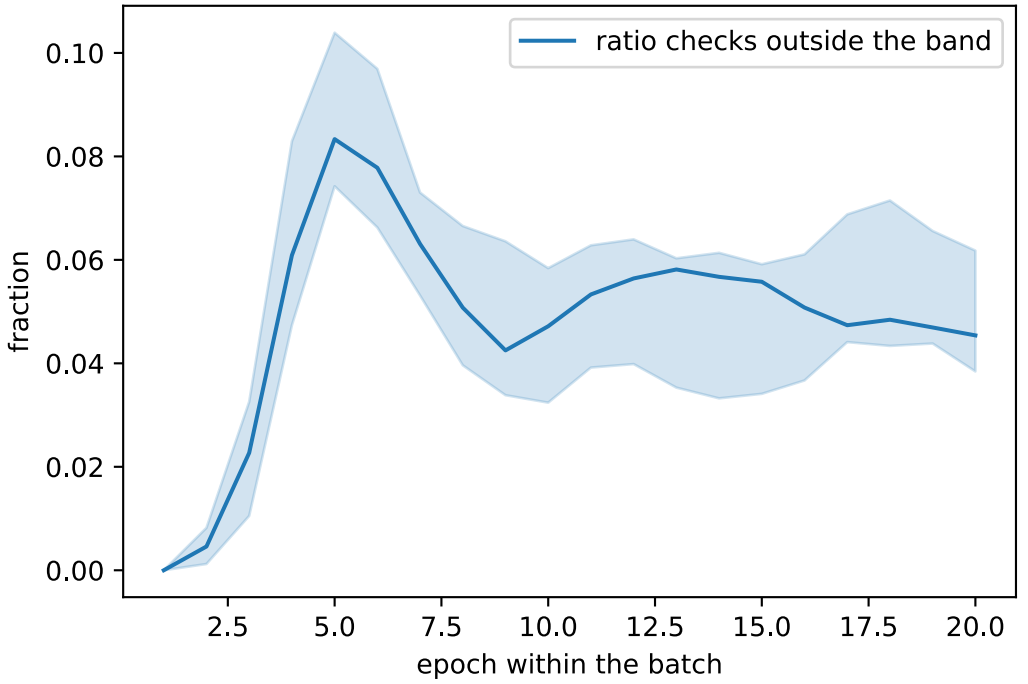
クリップ付きの実行では、どちらの割合も 20 回の検査につき 1 回前後である。クリップなしの対照実験では、比がバンド外に出る頻度はその数倍にのぼり、しかも平均よりバッチの最終エポックでのほうが高くなる。パスを重ねるたびに比がさらに離れていくのを止める仕組みがないからである。

15.2.4.5 訓練の診断指標

14.9 節で見たように、方策勾配の損失の値は学習の進捗について何も語らない。収益が 20 から 500 へと上昇するあいだも、損失は 0 付近にとどまり続ける。監視すべきは更新そのものの測定値であり、`diag` にはすでにすべての実行・シード・更新・エポックにわたる測定値が格納されている。まずは更新方向に平均を取り、1 バッチ内の 20 エポックの様子を見てみよう。

```
clipped = diag['clipped (PPO)']
d2l.plot_curves({'ratio checks outside the band':
                 clipped[..., CLIPPED].mean(1)},
                xlabel='epoch within the batch', ylabel='fraction',
                x=np.arange(1, num_epochs + 1))
kl = np.median(clipped[..., KL].mean(1), axis=0)
print(f'approximate KL: peak {kl.max():.4f} at epoch {kl.argmax() + 1}, '
      f'{kl[-1]:.4f} at epoch {num_epochs}')
for name, d in diag.items():
    print(f'{name:>13}: approximate KL at the last epoch, median '
          f'{np.median(d[:, :, -1, KL]):.3f}')
```

```
approximate KL: peak 0.0071 at epoch 6, 0.0056 at epoch 20
clipped (PPO): approximate KL at the last epoch, median 0.004
      no clip: approximate KL at the last epoch, median 0.386
```

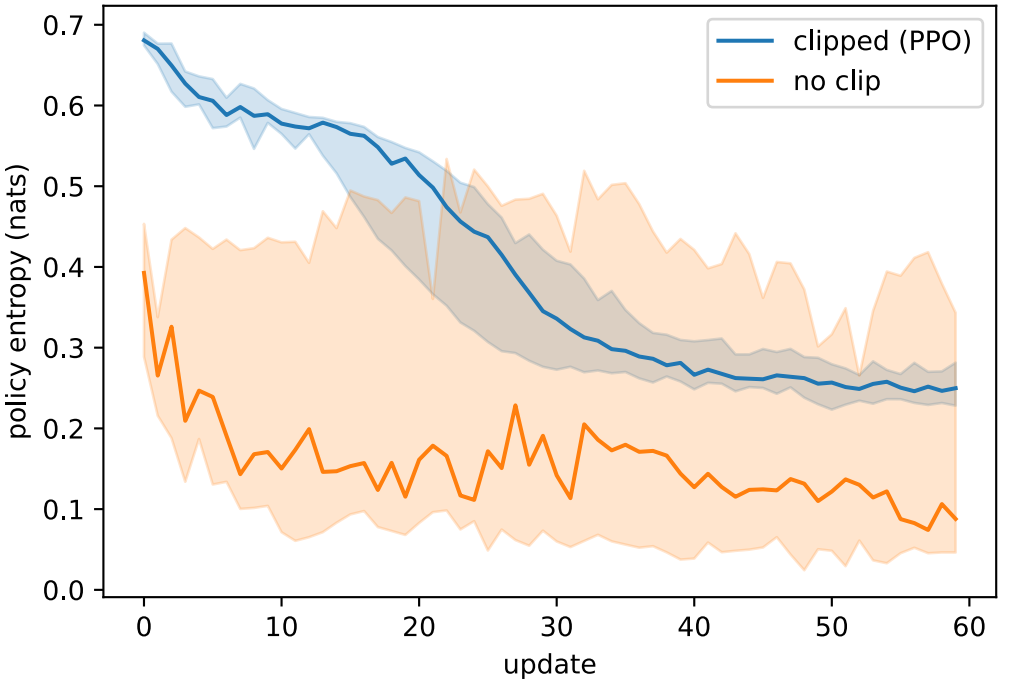


バッチ内では、すべての比が 1 に等しい最初のエポックにおいて、バンド外の割合は厳密に 0 である。その後 5〜6 エポック目あたりでピークに達し、20 エポック目までにはピークの約半分へと低下する。この低下は片側のみの最小値演算に由来する。悪化する側へバンドを出たサンプルはクリップなしの項の勾配を保つため比がバンド内へ引き戻されるのに対し、改善する側へバンドを超えたサンプルは勾配が 0 になってその場にとどまるからである。近似 KL も同じ形状を描き、値は約 10 分の 1 になる（上のセルがそのピークを出力する）。クリップがない場合、ダイバージェンスはパスごとに増大し続け、バッチの最終エポックではクリップ付きの実行の数十倍に達するのが普通である。

更新全体にわたってクリップ付き方策のエントロピーを追うと、方策がどれだけ決定論的に近づいたかがわかる。

```
d2l.plot_curves({name: d[..., ENTROPY].mean(-1) for name, d in diag.items()},
                xlabel='update', ylabel='policy entropy (nats)')
print(f'clipped: entropy {clipped[:, :5, :, ENTROPY].mean():.2f} over the '
      f'first five updates, {clipped[:, -5:, :, ENTROPY].mean():.2f} over the '
      f'last five')
```

```
clipped: entropy 0.65 over the first five updates, 0.25 over the last five
```



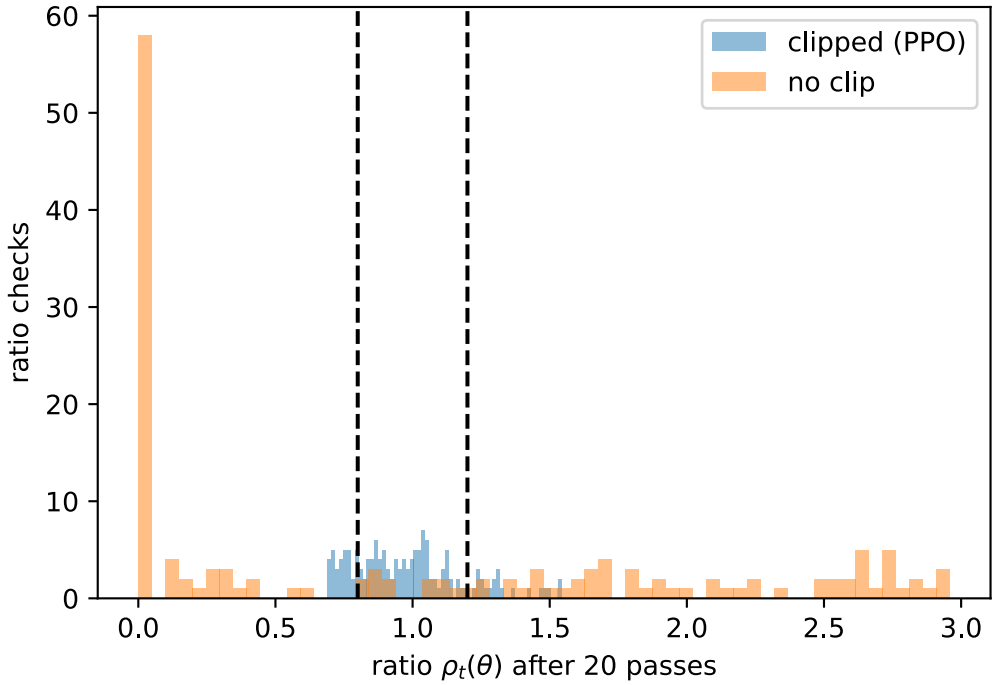
クリップ付きの実行のエントロピーは約 0.65 nats から約 0.25 nats へと低下する。学習が進むにつれて方策の選択はより確定的になり、エントロピーボーナスはそれを止めずに緩やかにする。一方、クリップなしの実行ではエントロピーがはるかに急速に失われており、これこそが破綻する実行を終焉させる飽和の正体である。

要約された割合だけでは、比がどのように分布しているかは見えない。そこで、バッチ内のドリフトが最大となる訓練開始時点の同一な初期エージェントを 2 つ用意し、固定した同一のアドバンテージと対数確率を持つ同じバッチを両方に与えて、一方ではクリップあり、もう一方ではクリップなしで 20 パスを実行し、各パスの後にすべての比を読み取る。3 つ目の同一なエージェント `collector` はバッチを収集した後はそのまま保持し、すべての比における古い方策とする。

```
probe = {arm: cartpole_agent(2) for arm in arms}   # twins
collector = cartpole_agent(2)                      # a third twin, kept fresh
env = gym.make('CartPole-v1')
env.reset(seed=2)
batch = d2l.rollout(env, collector.act, batch_episodes,
                    np.random.default_rng(2))
adv = d2l.standardize(batch.gae(collector.value_np, gamma, lam))
logp_old = collector.log_prob_np(batch.obs, batch.act)
rhos = {}
for arm, surrogate in arms.items():
    R = []
    for _ in range(num_epochs):   # one pass at a time, ratios read after
        ppo_epochs(probe[arm], batch.obs, batch.act, adv, logp_old,
                   epsilon_clip, 1, entropy_coef, surrogate)
        R.append(np.exp(probe[arm].log_prob_np(batch.obs, batch.act)
                        - logp_old))
    rhos[arm] = np.array(R)
```

`d2l.plot_hist`（オンライン付録「ユーティリティ関数とクラス」）を用いて最終パス後の 2 組の比のヒストグラムを重ねて描画し、バンドの両端を破線で示す。

```
d2l.plot_hist({arm: r[-1] for arm, r in rhos.items()},
              xlabel=rf'ratio $\rho_t(\theta)$ after {num_epochs} passes',
              ylabel='ratio checks',
              vlines=(1 - epsilon_clip, 1 + epsilon_clip))
```

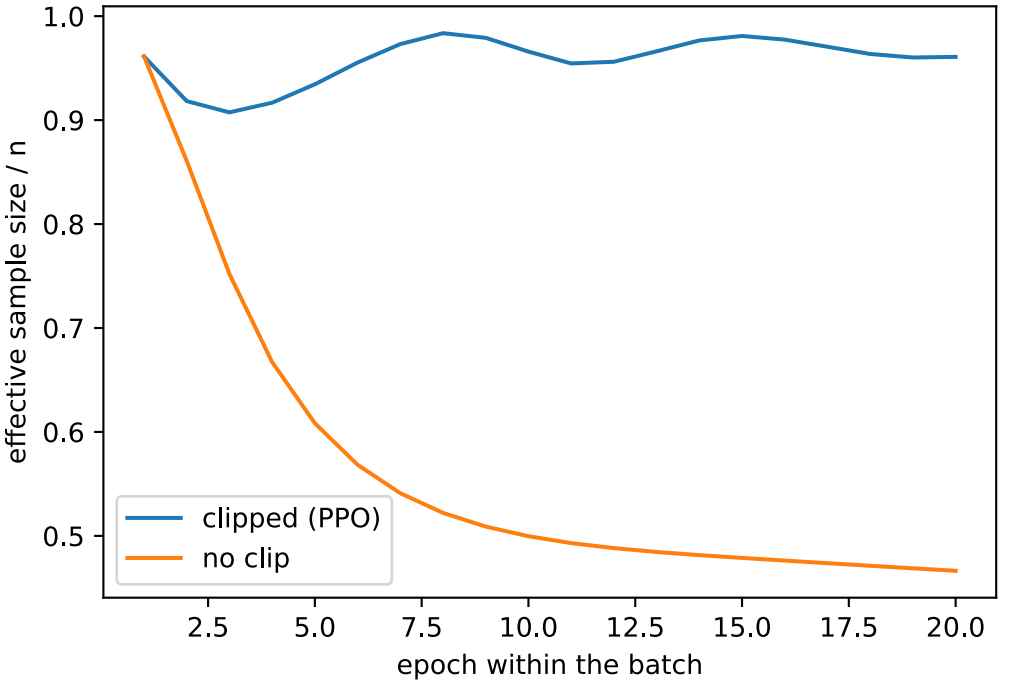


20 パスの後でも、クリップ付きエージェントの比は 1 の近くにまとまっており、破線で示されたバンドの端を超えるものは少数にとどまる。一方、クリップなしの比は 0 の近くに積み上がり、残りの大半はバンドを大きく超えて 2〜3 付近まで広がっている。2 や 3 という比は、初期方策が確率 2 分の 1 や 3 分の 1 で選んだ行動を新しい方策がほぼ常に選ぶようになった場合の $1/\pi_{\theta_{\text{old}}}(a)$ に相当し、0 付近の比はもう一方の行動に対応する。このように重み付けし直されたバッチには、どれだけの価値が残っているのだろうか。n 個の重要度重みの**有効標本サイズ**（effective sample size）$N_{\text{eff}} = 1/\sum_j \bar{w}_j^2$ がこの問いに答える。ここで $\bar{w}_j$ は、ステップ j の比をバッチ内のすべての比の和で割った値である。この量はすべての重みが等しいときに n に等しくなり、単一の重みが支配するときに 1 となる。

```
ess = {}
for arm, r in rhos.items():
    w = r / r.sum(axis=1, keepdims=True)   # normalized weights per epoch
    ess[arm] = 1 / (w ** 2).sum(axis=1) / r.shape[1]
d2l.plot_curves(ess, xlabel='epoch within the batch',
                ylabel='effective sample size / n',
                x=np.arange(1, num_epochs + 1))
print(f'after {num_epochs} epochs the batch is worth '
```

```
        + ' vs '.join(f'{v[-1]:.0%} ({k})' for k, v in ess.items())
        + f' of its {len(batch)} steps')
```

```
after 20 epochs the batch is worth 96% (clipped (PPO)) vs 47% (no clip) of its 147 steps
```



クリップありでは、20 エポック目まで重みはほぼ均等に保たれる。クリップなしでは、有効標本サイズはクリップありの条件を大きく下回り、バッチの約半分まで落ち込む。この値は飽和によって説明できる。新しい方策がある行動 $a^*(s)$ を確実に選ぶようになったとしよう。このとき s におけるサンプルは、重み $1/\pi_{\theta_{\text{old}}}(a^* \mid s)$ を持ち（そのサンプルの行動が a^* であった場合であり、これは確率 $\pi_{\theta_{\text{old}}}(a^* \mid s)$ で起きる）、それ以外では重み 0 となる。重みの平均は 1、重みの二乗平均は $1/\pi_{\theta_{\text{old}}}(a^* \mid s)$ であるため、バッチが大きくなるにつれて N_{eff}/n は訪問した状態にわたる $\pi_{\theta_{\text{old}}}(a^* \mid s)$ の調和平均に近づく。2 つの行動に対してほぼ一様な初期方策では、その値は約 2 分の 1 である。これを確かめるには、サンプリングされた行動だけでなくすべての行動の確率が必要になる。`action_probs` はそれらを NumPy 配列として返し、未訓練の双子エージェント `collector` が収集方策にあたる。

```
def action_probs(ac, obs):
    """pi(.|s) for a numpy batch of states, as numpy."""
    return np.asarray(jax.nn.softmax(d2l.forward_np(ac.policy, obs), -1))
```

`action_probs` は `d2l.forward_np`（オンライン付録「ユーティリティ関数とクラス」、すなわち `act` も用いるコンパイル済みの順伝播）を通して方策を実行する。

```
p_old = action_probs(collector, batch.obs)     # the collecting policy
p_new = action_probs(probe['no clip'], batch.obs)
```

```
a_star = p_new.argmax(axis=1)                  # the unclipped choice
p_star = p_old[np.arange(len(batch)), a_star]
print(f'unclipped policy: median probability of its choice '
      f'{np.median(p_new.max(axis=1)):.3f}')
print(f'harmonic mean of the old probability of that choice: '
      f'{1 / (1 / p_star).mean():.2f}')
```

```
unclipped policy: median probability of its choice 0.981
harmonic mean of the old probability of that choice: 0.41
```

クリップなしの方策はほぼ決定論的になっており、調和平均はクリップなし条件の有効標本サイズとの差が約 0.1 以内に収まっている（方策が完全には決定論的でないため、調和平均のほうがわずかに小さい）。有効標本サイズは重みがどれだけ均等に散らばっているかを測るものであり、アドバンテージや新しい方策が訪問するはずの状態は無視するため、ダイバージェンスやクリップ率を置き換えるのではなく補完する指標となる。

最後に、サンプリングのノイズを取り除くため最も確率の高い行動を貪欲に選ぶようにすると、訓練済みのクリップ付きエージェントは時間制限までポールをバランスさせ続けるはずである。

```
ac, env = agents['clipped (PPO)'][0], gym.make('CartPole-v1')
env.reset(seed=2)
score = d2l.evaluate(env, ac.act_greedy, num_episodes=100)
print(f'greedy mean return over 100 episodes: {score:.0f}')
```

```
greedy mean return over 100 episodes: 500
```

15.2.4.6 ベクトル化された収集とミニバッチ

ここまでのループでは、1 つの環境でエピソードを最後まで実行し、フルバッチのパスを取っていた。`cleanrl/ppo.py` (Huang, Rousslan F. J. Dossa, et al. 2022) のような実用実装では、環境の N 個のコピーを並列に固定ホライズン T だけ進め、$N \times T$ の長方形配列をなす遷移データを生成して、その端でまだ続いているすべてのエピソードを打ち切る。打ち切り箇所では、14.9 節が CartPole の時間制限で用いたのと同じブートストラップによって、記録されていない将来の価値をクリティックが補う。訓練済みエージェントを用い、8 個の環境を 32 ステップ進めてそのような長方形データを 1 つ収集してみよう。

```
envs = gym.vector.SyncVectorEnv([lambda: gym.make('CartPole-v1')] * 8)
obs, _ = envs.reset(seed=0)
rng, steps = np.random.default_rng(3), []
for _ in range(32):                        # the fixed 32 x 8 rectangle
    u = rng.random((len(obs), 1))          # inverse-CDF sampling, per env
    act = (u > action_probs(ac, obs).cumsum(1)[:, :-1]).sum(1)
    obs, rew, term, trunc, _ = envs.step(act)
    steps.append((rew, term | trunc))
rew, done = map(np.array, zip(*steps))
print(f'a fixed {rew.shape} rectangle of steps, '
      f'{int(done.sum())} episode boundaries inside it')
print(f'V at the cut, the value of the unrecorded future: '
      f'{ac.value_np(obs).round(0)}')
```

```
a fixed (32, 8) rectangle of steps, 0 episode boundaries inside it
V at the cut, the value of the unrecorded future: [100. 100. 100. 100. 100. 100. 100. 100.]
```

訓練済みエージェントは 32 ステップ以内には倒れないため、8 本の軌跡はすべて打ち切り箇所で終わり、その価値は永続的にバランスを保った場合の割引価値である $1/(1-\gamma) = 100$ に近い値となる。ブートストラップがなければ、GAE は各打ち切りを転倒として扱い、その直前の行動に大きな負のアドバンテージを与えてしまうだろう。また、Gymnasium のベクトル環境は終了したエピソードを自動的に再開する。エピソード終了直後のステップは報酬 0 とともに新しいエピソードの最初の観測を返し、渡された行動を無視するため、実装ではそのステップをマスクしてアドバンテージと損失の計算から除外する。最後に、実用コードはこの長方形データをシャッフルしてミニバッチに分割し、その上で数エポック（典型的には 4 エポック）を実行する。これにより各比が検査される回数は 20 回ではなく 4 回になり、ステップが小さくなることで必要なメモリが減り、勾配ノイズも加わる。演習問題 5 でこのループを構築する。

ベクトル化された収集は、1 サンプルあたりのコストの意味も変える。本章で環境ステップ数を数えているのは、ロボットアームの実機ではすべてのステップが現実の動作だからである。しかし GPU 上で動作する物理シミュレータは、一度に数千体のロボットのコピーを進めることができるため (Makoviychuk et al. 2021)、数千環境分の幅を持つ長方形データが 1 秒未満で得られ、重要になるコストは実経過時間へと変わる。非常に大きなバッチによって少数回の各更新が正確になるため、PPO はこの領域に適している。方策が実機へ転移するようにシミュレータのパラメータをランダム化する**ドメインランダマイゼーション** (Tobin et al. 2017) と組み合わせることで、脚式ロボットは数分の訓練で歩行を学習する (Rudin et al. 2022)。次節以降の手法のほうがはるかに少ないステップ数で済むにもかかわらず、この分野で今なお PPO が標準であり続けている理由はここにある。次節以降の手法が優位に立つのは、ステップが実機上の現実の動作である場合である。

PPO によってエージェントは各バッチに対して複数回のステップを実行できるようになり、CartPole では 4 エポックで 1 エポックの半分の環境ステップ数で上限に到達する。そのクリップは各ステップが行動確率をどれだけ動かすかを制限するが、制限しているのは経路であって到達点ではない。クリップ付きの実行でもエントロピーが依然として 0.65 nats から 0.25 nats へと低下するのは、目的関数があくまで小さなエントロピーボーナスを伴う収益であり、その最適解がほぼ決定論的だからである。15.3 節では代わりに目的関数そのものを変更し、方策を固定された参照方策の近くに保つことで、経路だけでなく最適解自体を動かす。

15.2.5 まとめ

数千環境ステップからなるオンポリシーのバッチは 1 回の勾配ステップ以上に活用されるべきであるが、パラメータ空間における同じ大きさのステップは方策をまったく異なる大きさで変化させうる。重要度サンプリングは古いデータ上で新しい方策を厳密に評価するものの、その分散はエピソード長に対して指数関数的に増大する。そこでステップごとに 1 つの比だけを残すと、その増大が取り除かれる代わりに、古い方策の状態において新しい方策の行動を評価することになる。性能差補題はこの置き換えこそが代理目的関数の唯一の誤差であることを示しており、TRPO は新旧の行動分布間のダイバージェンスに制約を課すことでこの誤差を抑える。これに対し PPO は、比が 1 の周りのバンドを出た時点で目的関数への各サンプルの寄与をクリップすることで、数エポックにわたる各バッチの再利用を可能にする。CartPole では 4 エポックで 1 エポックの約半分の環境ステップ数で上限に到達し、大きな学習率でもクリップが方策の崩壊を防ぐ。クリップ率、近似 KL、エントロピー、および比の有効標本サイズは、各更新が方策をどれだけ動かしたかを示す指標となる。

15.2.6 演習問題

1. **1 つのロジットに対するステップ制限。**●Figure 15.3 の 2 行動方策 $\pi_\theta(a{=}1) = \sigma(\theta)$ と、更新 $\theta \to \theta + \Delta$ を制限する 3 つの方法（ダイバージェンスに対する信頼領域、PPO のクリップ、および $|\Delta|$ の上限）を考える。

1. $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta+\Delta})$ を Δ の 2 次まで展開し、$\frac{1}{2}\sigma'(\theta)\,\Delta^2$ に等しいことを示せ。図に描かれた $\Delta = 2$ の 2 つの更新（$\theta = 0$ からの更新と $\theta = 6$ からの更新）について厳密なダイバージェンスを計算せよ。$\Delta_{\mathrm{KL}} = 0.02$ の信頼領域内に収まるのはどちらか。
2. この二次近似式を用いて、この信頼領域が許容する最大の $|\Delta|$ を $\theta = 0$ および $\theta = 6$ において求めよ。$\theta = 6$ において、その大きさを持つ正負両方向の 2 つのステップに対する厳密なダイバージェンスを計算せよ。そこで二次近似式の精度が $\theta = 0$ のときよりはるかに低くなるのはなぜか。
3. 次に、正のアドバンテージを持つ 1 つのサンプルに対し、$\epsilon_{\text{clip}} = 0.2$ とした Equation 15.22 のクリップによってステップを制限する。更新はサンプリングされた行動の確率を高めるため、$\Delta > 0$（サンプルが $a = 1$ のとき）または $\Delta < 0$（サンプルが $a = 2$ のとき）となる。クリップ付き目的関数が増加し続ける最大の $|\Delta|$ を、$\theta = 0$ と $\theta = 6$ のそれぞれで両方の行動について求めよ。
4. 方策が θ に対して鈍感な領域では大きなパラメータステップを許容し、敏感な領域では小さなステップのみを許容するのは、3 つの制限のうちどれか。$\theta = 6$ において、サンプリングされた行動がどちらのときにクリップは信頼領域より緩くなり、どちらのときにより厳しくなるか。

2. **爆発する比の積**。状態を無視する方策が $\pi_{\theta_{\text{old}}}$ でエピソードを収集し、そのデータを用いて π_θ を評価するとする。軌跡の比 Equation 15.15 を W_T（ステップ数 T のエピソードに対する比）と書く。

1. 古い方策のもとで $\mathbb{E}[W_T] = 1$ および $\mathrm{Var}[W_T] = (1+\chi^2)^T - 1$ が成り立つことを示せ。ここで $\chi^2 = \sum_a \big(\pi_\theta(a) - \pi_{\theta_{\text{old}}}(a)\big)^2 / \pi_{\theta_{\text{old}}}(a)$ である。本文の値 $1.04^T - 1$ が $\pi_{\theta_{\text{old}}} = (0.5, 0.5)$ と $\pi_\theta = (0.6, 0.4)$ に対して成り立つことを確かめよ。
2. エピソード数 n が大きくなるにつれて、重み W_T の有効標本サイズは $n/\mathbb{E}[W_T^2]$ に近づく。(a) の 2 つの方策について、この極限が 1 に等しくなる n（$T = 200$ のとき）はいくらか。その本数のエピソードからなるバッチを 1 つシミュレーションし（1 エピソード内で $a = 1$ となるステップ数は二項分布に従う）、その有効標本サイズを計算せよ。バッチから算出される有効標本サイズがその極限よりはるかに大きくなるのはなぜか。重要度重み付き推定量の分散を表しているのはこの 2 つのうちどちらか。
3. ステップごとの代理目的関数 Equation 15.17 は、各ステップを単一の比 ρ_t で重み付けする。ここでのその重みの分散はいくらか。また、Equation 15.17 におけるどの近似によってそれは T に依存しなくなっているか。

3. **診断指標が推定している量**。●状態 s を固定し、$a \sim \pi_{\theta_{\text{old}}}(\cdot \mid s)$ をサンプリングして、$\rho = \pi_\theta(a \mid s)/\pi_{\theta_{\text{old}}}(a \mid s)$ とおく。

1. ppo_epochs の近似 KL はバッチ全体で $k_1 = -\log\rho$ を平均する。$\mathbb{E}[k_1] = D_{\mathrm{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \| \pi_\theta(\cdot \mid s))$ が成り立つことを示せ。続いて $k_3 = (\rho - 1) - \log\rho$ も同じ期待値を持ち、すべてのサンプルに対して非負であることを示せ。両推定量を $\rho - 1$ の 2 次まで展開せよ。方策の変化が小さいとき、k_3 の分散が k_1 よりはるかに小さくなるのはなぜか。なお、`cleanrl/ppo.py` は両方を記録している。
2. 問題 2b では、有効標本サイズの極限 $n/\mathbb{E}[w^2]$（独立な n 個の重み w が $\mathbb{E}[w] = 1$ を満たす場合）を用いた。大数の法則を用いてこの極限を導出せよ。続いて $\mathbb{E}[\rho^2] = 1 + \chi^2$ を示し、重み ρ の有効標本サイズを n で割った値が $1/(1+\chi^2)$ に近づくことを示せ。ここで、次とおく。

$$\chi^2 = \sum_a \frac{\big(\pi_\theta(a \mid s) - \pi_{\theta_{\text{old}}}(a \mid s)\big)^2}{\pi_{\theta_{\text{old}}}(a \mid s)}.$$

方策の変化について 2 次までで $\chi^2 \approx 2D_{\mathrm{KL}}$ が成り立つことを示せ。

3. バッチから状態が引かれるとき、χ^2 は訪問した状態にわたるその平均となる。χ^2 のどの値を、15.2.4.5 節の 2 つの割合 `ess[name][-1]` は意味するか。(b) の二次の関係式は、それぞれに対してどの近似 KL を予測するか。各予測を、同じ比に対する k_1 推定値（`-np.log(rhos[arm][-1])).mean()`（各 `arm` について）と比較せよ。どちらのエージェントで二次の関係式が破綻するか。方策の変化の大きさ以外に、方策の訓練に用いたまさにそのバッチ上で測定された比は、(b) のどの仮定に反しているか。

John Schulman による Approximating KL Divergence の推定量 k_1 および k_3 に基づく。

4. [code] **KL による早期停止。**●クリッピングに加えて、方策が十分に動いた時点でバッチの再利用を打ち切る実装も可能である。Spinning Up の PPO は毎パスの前に近似 KL を計算し、それが 1.5 倍の `target_kl`（デフォルト値は 0.01）を超えた時点でそのバッチに対するパスを終了する。

   1. `ppo_epochs` の呼び出し（`train_ppo` 内）を、単一パスのループ `ppo_epochs(..., 1, ...)` に置き換えよ（`rhos` を 15.2.4.5 節で収集しているループと同様にする）。各パスの前に `log_prob_np` から近似 KL を計算し、停止規則を適用せよ。各バッチに費やされたパス数を記録せよ。
   2. アブレーション実験の 8 個のシード上で、早期停止を伴うクリップなしの変種を実行せよ。この停止規則だけで崩壊を防げるか。バッチあたりのパス数は訓練を通じてどのように変化するか。
   3. クリップ付きの変種も同様に早期停止付きで実行せよ。この実行において、1 バッチでの方策の変化を制限しているのは 2 つの仕組みのうちどちらか。その答えから、クリップを信頼領域と表現することについて何が言えるか。

Joshua Achiam による Spinning Up in Deep RL の PPO 実装（`target_kl` による早期停止）に基づく。

5. [code] **標準的なレシピ。**●●15.2.4.6 節で説明したループを構築せよ。すなわち、`gym.vector.SyncVectorEnv` 内の 8 個の環境を更新ごとに 32 ステップ進め、各打ち切り箇所でのクリティックの価値を用い自動リセットのステップをマスクして除外したうえで 32×8 の長方形データに対して GAE を計算し、続いてサイズ 64 のシャッフルされたミニバッチに分割して長方形データ上で 4 エポックを実行する。各ミニバッチでは、そのミニバッチ内で標準化した独自のアドバンテージを用いて `ppo_epochs` を 1 パス実行し、エポック開始前に計算した λ 収益のターゲットに向けて `fit_value` を 1 ステップ実行する。両ネットワークの学習率は 10^{-3} とせよ。

   1. 3 個のシードで訓練し、完了した直近 8 エピソードの平均が少なくとも 475 に達するまでの環境ステップ数を記録せよ。正しい実装では約 45,000 ステップ（大半のシードで 35,000〜55,000 ステップ）を要し、約 25,000 ステップで済む 15.2.4.3 節の再利用実験における 4 エポック条件より多くなる。
   2. クリティックの学習率を本節のループと同じ 10^{-2} に引き上げ、アクターの学習率は 10^{-3} のまま保て。正しい実装では約 15,000〜20,000 ステップで 475 に到達し、再利用実験のどの条件よりも少なくなる。(a) の実行の律速となっていたのはどちらのネットワークか。ベクトル化もミニバッチも用いない本節のループを同じ 2 つの学習率で動かすと、約 35,000〜40,000 ステップを要する。短い長方形データにすることで、1 環境ステップあたりに方策が更新される頻度はどう変わるか。
   3. 自動リセットのステップに対するマスクを外せ。それでも実行は学習するか。マスクを外した版はクリティックのターゲットにどのような遷移をつけ加えるか。また、CartPole がその影響をほとんど受けないのはなぜか。

6. **単調改善下界。**●●この問題では、Equation 15.20 を Schulman et al. (2015) のカップリン

グ論法によって導出する。$\gamma < 1$ の無限ホライズンタスクを考え、$\pi = \pi_{\theta_{\text{old}}}$、$\tilde{\pi} = \pi_{\theta}$、$A = A^{\text{old}}$、および $\bar{A}(s) = \sum_a \tilde{\pi}(a \mid s)\, A(s,a)$ と書く。$\Delta_{\text{TV}} = \max_s D_{\text{TV}}(\pi(\cdot \mid s), \tilde{\pi}(\cdot \mid s))$ とおき、ここで全変動距離は $D_{\text{TV}}(p,q) = \tfrac{1}{2}\sum_a |p(a) - q(a)|$ で定義される。

1. Equation 15.18 から $J(\theta) - J(\theta_{\text{old}}) = \sum_t \gamma^t \mathbb{E}_{s_t \sim \tilde{\pi}}\big[\bar{A}(s_t)\big]$ となること、および母集団代理目的関数 Equation 15.19 が $\bar{L}(\theta) = \sum_t \gamma^t \mathbb{E}_{s_t \sim \pi}\big[\bar{A}(s_t)\big]$ と書けることを示せ。ここで $s_t \sim \pi$ は時刻 t における方策 π のもとでの状態分布を表す。

2. 全変動距離が Δ_{TV} である 2 つの分布はカップリングできる。すなわち、$(a, \tilde{a})$ の同時分布であって、周辺分布が $\pi(\cdot \mid s)$ と $\tilde{\pi}(\cdot \mid s)$ であり、すべての状態で $P(a \neq \tilde{a}) \le \Delta_{\text{TV}}$ を満たすものが存在する。$\mathbb{E}_{a \sim \pi}[A(s,a)] = 0$ を用いて $|\bar{A}(s)| \le 2\Delta_{\text{TV}} A_{\max}$ を示せ。

3. 同じ開始状態から毎ステップでカップリングした行動を用いて両方策を実行し、m_t を時刻 t より前に 2 つの行動が食い違ったステップ数とする。事象 $m_t = 0$ の上では、2 本の軌跡は時刻 t まで一致する。次を示せ。

$$\big|\mathbb{E}_{s_t \sim \tilde{\pi}}[\bar{A}(s_t)] - \mathbb{E}_{s_t \sim \pi}[\bar{A}(s_t)]\big| \le 4\Delta_{\text{TV}}\big(1 - (1 - \Delta_{\text{TV}})^t\big) A_{\max}. \tag{15.23}$$

4. t について和を取ることで次を得て、

$$\big|J(\theta) - J(\theta_{\text{old}}) - \bar{L}(\theta)\big| \le \frac{4\Delta_{\text{TV}}^2 \gamma A_{\max}}{(1-\gamma)^2}, \tag{15.24}$$

   $D_{\text{TV}}^2 \le D_{\text{KL}}$ を用いて証明を完成せよ。

5. 係数 $4\gamma/(1-\gamma)^2$ の値を $\gamma = 0.99$ において計算せよ。(b) を用いて $\bar{L}(\theta) \le 2\Delta_{\text{TV}} A_{\max}/(1-\gamma)$ を示せ。ピンスカーの不等式 $D_{\text{KL}} \ge 2D_{\text{TV}}^2$ を用い、この下界が改善を保証するのは $\Delta_{\text{TV}} < (1-\gamma)/(4\gamma)$ の場合に限られることを示せ。このとき、改善が保証されるステップの $\bar{L}(\theta)$ は $A_{\max}$ を単位として最大どれだけ大きくなれるか。Δ_{TV} に対するこの上限を、ピンスカーの不等式が許容する全変動 $\sqrt{\Delta_{\text{KL}}/2}$（ダイバージェンスが予算 $\Delta_{\text{KL}} = 0.01$ に等しい状態における値であり、これは Schulman et al. (2015) が全実験で用いた予算である）と比較せよ。

7. [extended] **構成要素から作る TRPO。**●●`ppo_epochs` の呼び出し（`train_ppo` 内）を、1 回の TRPO 更新 (Schulman et al. 2015)（制約付き問題 Equation 15.21 で $\Delta_{\text{KL}} = 0.01$ としたもの）に置き換え、クリティックとアドバンテージはそのまま残せ。`ac.policy` のパラメータを 1 本の平坦なベクトルとして扱い、収集方策の行動確率 `action_probs(ac, b.obs)` を `logp_old` とともに固定せよ。

   1. 代理目的関数 Equation 15.17 の勾配 $\mathbf{g}$（θ_{old} における値）を計算せよ。続いて、バッチ内の状態にわたる平均ダイバージェンスの勾配をもう一度微分することで、フィッシャー・ベクトル積 $F\mathbf{v}$（θ_{old} におけるそのダイバージェンスのヘッセ行列を $\mathbf{v}$ に作用させたもの）を返す関数を作成せよ。勾配の有限差分と比較して検証せよ。
   2. 共役勾配法の 10 回の反復によって $F\mathbf{x} = \mathbf{g}$ を近似的に解き、ステップを $\xi\mathbf{x}$（ただし $\xi = \sqrt{2\Delta_{\text{KL}}/\mathbf{x}^\top F\mathbf{x}}$）にスケーリングせよ。$\xi$ を、実測される平均ダイバージェンスが高々 Δ_{KL} になり、かつ代理目的関数が増加するまで半分にし続けよ。
   3. 3 個のシードで訓練し、15.2.4.4 節のクリップ付きの実行と、学習曲線、各更新の実測ダイバージェンス、および半減ステップの回数について比較せよ。各手法は 1 バッチあたり何回の勾配計算とフィッシャー・ベクトル積の計算を費やすか。

4. Engstrom et al. (2020) は、コードレベルの設計選択を揃えると PPO と TRPO が同等の性能を示すことを見出した。ここでは両方とも方策更新以外のすべての選択を共有している。比較結果はこの知見と一致するか。

15.3 KL 正則化方策最適化

第 14 章および本章でこれまでに扱った手法はすべて、誰かが報酬を書き下していることを前提としていた。凍った湖やポールを載せたカートであれば、「ゴールに到達する」「ポールを直立に保つ」といったように報酬の設計は簡単である。しかし、人間が関心を寄せる大半のタスクではそうはいかない。有益な回答、安全な車線変更、あるいは読みやすい証明にスコアをつける数式を書き下せる人などいないからである。とはいえ、人間は 2 つの試行を見比べてどちらを好むかを答えることはでき、そのような比較データに報酬を当てはめることができる。広く使われた初期のチャットモデルに有益な応答を学習させたのもこの方法であり、人間の比較データに報酬モデル（reward model）を当てはめ、それに対して方策を訓練したのである (Christiano et al. 2017; Ouyang et al. 2022)。

データに当てはめた報酬は、人手で書き下した報酬にはなかった問題を突きつける。モデルは比較データが情報を提供する領域でのみ正確であり、それ以外の領域では推測に頼る一方、オプティマイザはあらゆる領域を探索する。比較データが決してカバーしなかった領域で、たまたま当てはめた報酬が高くなる場所をオプティマイザが見つけ出すと、方策はそこへ進み、真の品質は低下してしまう。実務で定着した対策は、比較データを生成した方策からの乖離に対してコストを課すこと、すなわちその参照からの KL ダイバージェンスにペナルティを科すことである。15.2 節の信頼領域（trust region）は、各更新が直前の方策からどれだけ動けるかを制限していた。これに対し、このペナルティは目的関数の一部であるため、解に至る経路だけでなく解そのものを変化させ、しかもその解は参照分布を報酬の指数で再重み付けした閉形式を持つ。

まず真の報酬がわかっているグリッドワールドを構築し、比較データから報酬を当てはめ、オプティマイザがその隙を突く様子を観察する。次にペナルティ付きの問題を 1 回の意思決定について解き、その解を多段階へと拡張した上で、方策が参照から離れるにつれて真の品質が上昇した後に低下していく様子をたどる。

15.3.1 比較からの報酬の学習

何がうまくいかなくなるのかを見るには、真の報酬が既知であるタスクを用意し、それを学習器には隠したまま結果を採点できるようにする必要がある。小さなグリッドワールドがその好例となる。

15.3.1.1 グリッドワールドとその参照方策

この環境は決定論的な移動を持つ 4×4 のグリッドであり、Figure 15.6 に示すとおりである。エピソードは左上のセルから始まり、右下の角にあるゴールに入ると終了して $+1$ の報酬を得る。それ以外の移動には 0.04 のコストがかかり、2 行目に隣り合って並ぶ 2 つの危険セル、すなわち**危険レーン**（hazard lane）のいずれかに入る移動には 1.5 のコストがかかる。真の報酬 $r^*(s, a)$ は、行動 a によって状態 s から入るセルに対する支払いである。セルには行ごとに番号が振られており、開始位置は状態 0、危険セルは状態 5 と 6、ゴールは状態 15 である。行動 0 から 3 はそれぞれ左、下、右、上への移動を表し、壁に向かって移動するとエージェントはその場にとどまる。

比較データは、1 つの固定された方策である**参照方策**（reference policy）π_{ref} の軌跡から収集する。これを有能だが慎重な方策とするため、温度 0.25 における最適行動価値のソフトマックス $\pi_{\text{ref}}(a \mid s) \propto e^{Q^*(s,a)/0.25}$ として定義する。開始位置では、ゴールへ向かう 2 つの移動は $Q^* = 0.59$、壁に向かう 2 つの移動は 0.52 であるため、参照方策は良い移動をそれぞれ確率

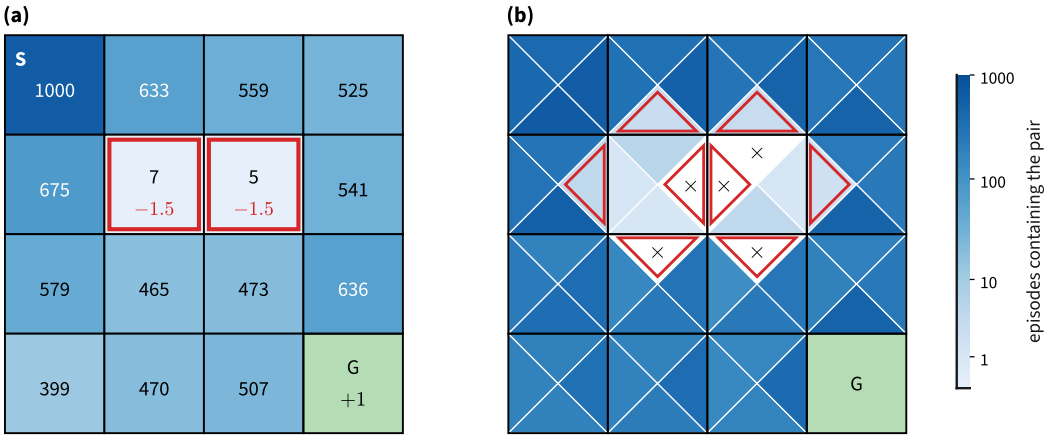


Figure 15.6: 本節のグリッドワールド。(a) エピソードは状態 0（S）から始まり、ゴールである状態 15（G、+1）で終了する。危険セルである 5 または 6 への移動には 1.5、それ以外のすべての移動には 0.04 のコストがかかる。各セルの数値は、1,000 本の参照軌跡のうち何本でそのセルが訪問されたかを示す。(b) 各状態行動対の台。各セルは左・下・右・上の移動に対応する 4 つの三角形に分割され、その対を含む軌跡の本数（対数スケール）で濃淡が付けられている。バツ印は一度も生じない対を表し、危険レーンへ入る 8 つの移動は太枠で囲まれている。

0.28 で、無駄な移動をそれぞれ確率 0.22 で選択する。危険セルへの移動は他の選択肢より価値が約 1.5 低く、ソフトマックスによって確率は約 e^{-6} になるため、参照方策がそこへ入ることはめったにない。次のコードセルでは、配列 $P[s, a, s']$ と $r^*[s, a]$ を `d2l.TabularMDP`（14.1 節）として構築し、Q^* を `d2l.value_iteration` の実行とさらに 1 回の `backup`（14.2 節）の適用によって計算して、参照方策を構成する。

```
gamma = 0.95                                  # a 4 x 4 gridworld
P = np.zeros((16, 4, 16))
for s, a in np.ndindex(16, 4):
    x, y = s % 4, s // 4
    dx, dy = [(-1, 0), (0, 1), (1, 0), (0, -1)][a]   # left, down, right, up
    P[s, a, min(3, max(0, y + dy)) * 4 + min(3, max(0, x + dx))] = 1.0
P[15], P[15, :, 15] = 0.0, 1.0                # the goal absorbs
bonus = np.full(16, -0.04)                    # every step costs a little
bonus[[5, 6]], bonus[15] = -1.5, 1.0          # a hazard lane; the goal pays
r_true = np.einsum('sat,t->sa', P, bonus)     # r*(s, a): the entered bonus
r_true[15] = 0.0
mdp = d2l.TabularMDP(P, r_true, gamma)
Q_true = mdp.backup(d2l.value_iteration(mdp, 200)[-1])
pi_ref = np.exp(Q_true / 0.25)                # competent but hedging
pi_ref /= pi_ref.sum(1, keepdims=True)
```

15.3.1.2 選好とブラッドリー・テリーモデル

1 回の比較からわかるのは、評価者が 2 つの軌跡のどちらを選好（preference）したかだけであり、どれほどの差があったかはわからない。ブラッドリー・テリーモデル（Bradley-Terry model）は、このような回答を統計モデルへと変換する (Bradley and Terry 1952)。このモデルは各候補にスカラースコアとして軌跡の収益 $r(\tau)$（報酬 r のもとでの値）を割り当て、比較をスコア差の

ノイズ付き観測として扱う。

$$P(\tau \succ \tau') = \sigma\big(r(\tau) - r(\tau')\big), \qquad \sigma(u) = \frac{1}{1+e^{-u}}. \tag{15.25}$$

したがって、収益の等しい 2 つの軌跡はそれぞれ確率 1/2 で選好され、収益に 1 単位の差があると優れた方が 4 回中約 3 回勝つことになる。

グリッドワールドでは、収益は報酬に関して線形である。軌跡の割引訪問回数を $\mathbf{c}(\tau)$ とし、状態行動対ごとに 1 つの要素を持たせる。すなわち、要素 (s,a) は γ^t の和であり、軌跡がステップ t において行動 a を状態 s でとるたびにこの項が加わる。すると $r(\tau) = \mathbf{c}(\tau)^\top r$（ここで報酬は 64 要素に平坦化されている）となり、Equation 15.25 の当てはめは特徴量を $\mathbf{c}(\tau) - \mathbf{c}(\tau')$、重みを報酬とするロジスティック回帰となる。`sample_traj` は $\mathbf{c}(\tau)$ を π_{ref} の 1 本の軌跡（ゴールに到達するか 40 ステップ後に終了する）に対して返し、`X` はそれを 1000 本積み重ねる。ランダムな 6000 組のペアに対し、Equation 15.25 の確率（r^* のもとでの値）に従うコイントスでラベルを付け、対数尤度に対する勾配上昇法（gradient ascent）によって $\hat{r}$ を当てはめる。このとき、データが少ない対でも当てはめが定まるよう、小さな重み減衰（weight decay）0.02 を加える。`episodes` は各対が出現するエピソード数を数え、`core` は少なくとも 50 エピソードに出現する対を表し、`into` は危険レーンへ上および左右から入る 4 つの移動を並べる。

```
rng = np.random.default_rng(0)
sigmoid = lambda u: 1 / (1 + np.exp(-u))

def sample_traj(mdp, pi, rng, T=40):
    """Discounted visit counts c(tau) of one episode from state 0."""
    s, c = 0, np.zeros((16, 4))
    for t in range(T):
        a = rng.choice(4, p=pi[s])
        c[s, a] += mdp.gamma ** t             # the return is c(tau) . r
        s = mdp.P[s, a].argmax()
        if s == 15:
            break
    return c.ravel()

X = np.stack([sample_traj(mdp, pi_ref, rng) for _ in range(1000)])
R = X @ r_true.ravel()                        # true returns; the fit never sees them
i, j = rng.integers(0, 1000, (2, 6000))
y = (rng.random(6000) < sigmoid(R[i] - R[j])).astype(float)
D = X[i] - X[j]
lr, wd, num_steps = 0.5, 0.02, 4000           # ascent step, weight decay
w = np.zeros(64)
for _ in range(num_steps):                    # logistic regression on (D, y)
    w += lr * (D.T @ (y - sigmoid(D @ w)) / len(y) - wd * w)
r_hat = w.reshape(16, 4)
episodes = (X > 0).sum(0).reshape(16, 4)      # episodes containing each pair
core = episodes >= 50
into = ([1, 4, 2, 7], [1, 2, 1, 0])           # into cells 5, 6 from above, sides
print(f'never visited: {(episodes == 0).sum()} pairs; '
      f'in at least 50 episodes: {core.sum()} pairs')
print(f'rms fit error on those {core.sum()}: '
      f'{np.sqrt(((r_hat - r_true)[core] ** 2).mean()):.3f}')
print(f'moves into the hazards: in {episodes[into]} episodes, '
```

```
        f'fitted {np.round(r_hat[into], 2)}')
```

```
never visited: 9 pairs; in at least 50 episodes: 46 pairs
rms fit error on those 46: 0.193
moves into the hazards: in [3 4 3 2] episodes, fitted [-0.03 -0.05 -0.02 -0.02]
```

この当てはめは、core に含まれる 46 個の対において二乗平均平方根（RMS）誤差 0.19 を持つ。9 個の対は一度も出現しない。すべてのエピソードがすでに終了しているゴールでの 4 つの行動と、Figure 15.6 でバツ印が付いた 5 つの対である。これらの当てはめ報酬は初期値の 0 のままである。危険レーンへ上および左右から入る移動は、1,000 エピソード中それぞれ 2〜4 回しか生じず、下から入る 2 つの移動は一度も生じない。これでは少なすぎて、重みの初期値であり重み減衰が引き戻す先でもある 0 から、当てはめ報酬を大きく動かすことはできない（真のコストはいずれも 1.5 である）。これに対し、ニューラル報酬モデルであれば、良くも悪くもその帰納バイアス（inductive bias）に従って外挿することになる（演習問題 4）。

15.3.1.3 シェイピングに対する不変性

RMS 誤差 0.19 は、ステップコスト 0.04 と比べると大きく見えるが、その大部分はいかなる比較からも観測できないものである。14.1 節で導入したポテンシャルに基づくシェイピング（potential-based shaping）は、$\gamma\Phi(s') - \Phi(s)$ を s から s' への各移動の報酬に加える（ここで Φ は状態上の関数であり、ゴールにおいて $\Phi = 0$ を満たす）。そして 14.2 節では、これが最適方策を変化させないことを証明した。軌跡に沿うとこれらの項は望遠鏡和となって相殺されるため、状態 0 からゴールに到達するすべての軌跡は、収益が同じ量 $-\Phi(0)$ だけ変化する。すべての比較は状態 0 からの 2 本の軌跡を対象とするため、このシフトは $r(\tau) - r(\tau')$ において打ち消し合い、Equation 15.25 では検出できない。（10 本に 1 本の割合で 40 ステップ後に打ち切られる軌跡には末尾の項 $\gamma^{40}\Phi(s_{40})$ が残るが、小さすぎて Φ を特定するには至らない。）したがって、比較データはシェイピングの自由度を除いてしか報酬を決定できず、区別できない報酬の中では重み減衰によって重みが最小のものが選ばれる。

次のコードセルでは、最小二乗法（least squares）の意味で core 上の当てはめ誤差を最もよく説明するシェイピング項のポテンシャルを求める。F の (s, a) 行に Φ を掛けると $\gamma \sum_{s'} P(s' \mid s, a)\,\Phi(s') - \Phi(s)$ が得られ、最後の列を除くと $\Phi(15) = 0$ と固定される。

```
err = (r_hat - r_true)[core]
F = (gamma * P - np.eye(16)[:, None, :])[core][:, :15]  # shaping per unit of Phi
Phi = np.linalg.lstsq(F, err, rcond=None)[0]
print(f'rms error on core {np.sqrt((err ** 2).mean()):.3f}, after removing '
      f'the best shaping term {np.sqrt(((err - F @ Phi) ** 2).mean()):.3f}')
print(f'Phi(0) = {Phi[0]:.2f}; moves 13 -> 14 -> goal: fitted '
      f'{r_hat[13, 2]:+.2f}, {r_hat[14, 2]:+.2f}, true '
      f'{r_true[13, 2]:+.2f}, {r_true[14, 2]:+.2f}')
```

```
rms error on core 0.193, after removing the best shaping term 0.032
Phi(0) = 0.78; moves 13 -> 14 -> goal: fitted -0.08, +0.09, true -0.04, +1.00
```

シェイピング項を取り除くと RMS 誤差は 0.03 になる。つまり、シェイピングを除けば、データが存在する領域での当てはめは良好である。ポテンシャルは開始位置での約 0.8 からゴール手前での約 0.9 まで上昇し、core のどの対も触れない危険セルでは不定のまま残る。このようなポテンシャルは、ゴールへ入る移動から約 0.9 を引き、それ以外の各移動から約 $(1-\gamma) \times 0.8 = 0.04$ を引く。出力された報酬が示すのもまさにこの値であり、最後の移動の報酬は 0.09（本来は 1）となり、通常の移動のコストは 0.08（本来は 0.04）となる。どちらの報酬も状態 0 からのあらゆ

る軌跡のペアに同じ順序を付けるが、当てはめ報酬の方が重みは小さい。そしてこの当てはめ報酬のもとでは、0 付近と推定された危険セルへの移動の方が、通常の移動よりも低コストになってしまう。

言語モデルも同様の不変性に直面する。そこでの比較は、同一のプロンプトに対する 2 つの応答 y と y'（プロンプトを x とする）を対象とし、報酬 $r(x,y)+f(x)$（ここで f はプロンプトの任意の関数）はすべての比較を $r(x,y)$ とまったく同じように予測するため、学習された報酬はプロンプトごとの相対的な値としてしか識別されない（演習問題 1）。方策勾配法（policy gradient）はこれに影響されない。報酬からプロンプトの関数を引く操作はベースラインに相当し、期待勾配を変化させないからである（14.8 節）。GRPO のグループ平均もそのようなベースラインの一種だが、応答自身の報酬を含んでいる点が異なり、その節で示したとおり、期待更新量が $(K-1)/K$ 倍（K 個の応答からなるグループの場合）にスケールされる。

15.3.2 当てはめ報酬の最適化

ここで、エージェントが当てはめ報酬に対してとるのと同じ手順、すなわちそれを最大化する方策を求めることを行う。ダイナミクスが既知であれば、価値反復法（value iteration）によってその方策が求まる。`plan` は与えられた報酬に対する最適な決定論的方策をワンホットな行を持つ行列として返す。`ret` は、状態 0 からの厳密な方策の収益を 14.2 節の `policy_value` を通じて返す（これは線形ベルマン方程式 $V^\pi=(I-\gamma P_\pi)^{-1}r_\pi$ を 1 ステップで解く）。真の報酬と当てはめ報酬のそれぞれについて計画を立て、両方の方策を両方の報酬のもとで採点してみよう。

```
def plan(r):
    """Optimal deterministic policy (one-hot rows) for reward r."""
    m = d2l.TabularMDP(P, r, gamma)
    return np.eye(4)[m.backup(d2l.value_iteration(m, 200)[-1]).argmax(1)]

def ret(pi, r):
    """Exact return of policy pi from state 0 under reward r."""
    return d2l.TabularMDP(P, r, gamma).policy_value(pi)[0]

for name, r in (('true reward  ', r_true), ('fitted reward', r_hat)):
    pi = plan(r)
    print(f'plan on {name}: fitted return {ret(pi, r_hat):+.2f}, '
          f'true return {ret(pi, r_true):+.2f}')
```

```
plan on true reward  : fitted return -0.24, true return +0.59
plan on fitted reward: fitted return -0.03, true return -2.11
```

当てはめ報酬のもとでは、$\hat{r}$ 上の計画のスコアは -0.03 となり、真の最適方策の -0.24 を上回る。真の最適方策の当てはめスコアは真のスコアより 0.83 低く、シェイピングのオフセット $\Phi(0)=0.78$ に近いため、当てはめスコアは差にしか意味がない。一方、真の報酬のもとでは、2 つの計画のスコアはそれぞれ -2.11 と $+0.59$ となる。$\hat{r}$ 上の計画は $0\to1\to5\to6\to7\to11\to15$ と進み、2 つの危険セルをまっすぐ突っ切る。これは外周に沿う安全な経路と同じ最短ステップ数でありながら、$\hat{r}$ のもとではより低コストなのである。価値反復法は与えられた報酬を忠実に最適化したのであり、失敗の原因は参照方策のデータ外における報酬にある。異なる乱数シードで抽出した 24 個の選好データセットにわたって調べると、$\hat{r}$ 上の計画は 19 個で危険セルを横断し、さらに 4 個でゴールに一度も到達しない。真の最適方策が得られたのは 1 個のデータセットだけであった。

この失敗には名前が付いている。グッドハートの法則（Goodhart's law）は、指標が目標になった途端に指標としての価値を失うと述べるものである。強化学習（reinforcement learning）で

は、指定された報酬や当てはめ報酬のもとで高いスコアを出しながら本来の目的を損なう方策は、報酬を**ハック**する（報酬ハッキング）と言い、その多数の事例がまとめられている (Krakovna et al. 2020)。14.2 節で扱った接近ボーナス（最適方策を永遠に周回させてしまった例）は、人手で書いた報酬の誤りによって同じ仕組みが働いたものであった。ここでの誤りは統計的なものであり、データが手薄な場所に潜んでいる。

15.3.3 KL 正則化目的関数

当てはめ報酬が信頼に値するのは、参照方策が訪れた領域だけである。参照方策の近くにとどまる方策はデータが存在する領域にとどまるため、そこから離れることに対してコストを課す。ここで自然な尺度となるのが KL ダイバージェンスである。これは方策自身のもとでの期待値であるため、その方策自身のサンプルから推定でき、参照方策が決してとらない行動をとる方策に対しては無限大となる。有限個の行動集合上の報酬 $r(a)$、参照方策 π_{ref}、および係数 $\beta > 0$ を持つ 1 回の意思決定に対して、**KL 正則化目的関数**（KL-regularized objective）は次の式で定義される。

$$\max_{\pi} \; \mathbb{E}_{a\sim\pi}\big[r(a)\big] - \beta\, D_{\mathrm{KL}}(\pi \| \pi_{\text{ref}}). \tag{15.26}$$

第 1 項は期待報酬（expected reward）であり、第 2 項は参照方策からのダイバージェンス 1 nat ごとに報酬を β 単位だけ差し引く。係数 β はペナルティの重みであり、参照方策は固定されたままである。

15.3.3.1 1 回の意思決定に対する閉形式の最適解

命題。$\beta > 0$ に対して、目的関数 Equation 15.26 を最大化する唯一の方策は次で与えられる。

$$\pi^{\star}(a) = \frac{1}{Z}\,\pi_{\text{ref}}(a)\, e^{r(a)/\beta}, \qquad Z = \sum_{a} \pi_{\text{ref}}(a)\, e^{r(a)/\beta}, \tag{15.27}$$

また、その最大値は $\beta \log Z$ である。

証明。構成より $r(a) = \beta \log\big(\pi^{\star}(a) Z / \pi_{\text{ref}}(a)\big)$ が成り立つ。これを目的関数に代入すると、任意の方策 π に対して次を得る。

$$\begin{aligned} \mathbb{E}_{\pi}[r] - \beta D_{\mathrm{KL}}(\pi\|\pi_{\text{ref}}) &= \sum_{a} \pi(a)\, \beta \log \frac{\pi^{\star}(a)\, Z}{\pi(a)} \\ &= \beta \log Z - \beta\, D_{\mathrm{KL}}(\pi\|\pi^{\star}). \end{aligned}$$

第 37 章で証明されているギブスの不等式（Gibbs' inequality）より、残ったダイバージェンスは非負であり、$\pi = \pi^{\star}$ のとき、かつそのときに限り 0 となる。■

このように、最適解は参照方策に指数ティルト（exponential tilt）$e^{r/\beta}$ を掛けて再正規化したものとなり、その達成値は対数分配関数（log-partition function）となる。Figure 15.7 に 5 つの行動に対するこの積を示す。そこでの参照方策は意図的に報酬に関して単調でないように選ばれており、2 番目の行動における落ち込みは穏やかなティルトのもとでも残存する。すなわち、最適解は行動を報酬だけでランキングするのではなく、参照方策を再重み付けするのである。

2 つの境界ケースを補足しておく。第 1 に、最適解は参照方策の台（support）を受け継ぐ。$\pi_{\text{ref}}(a) = 0$ であれば、a に正の確率質量を置くいかなる方策もダイバージェンスが無限大とな

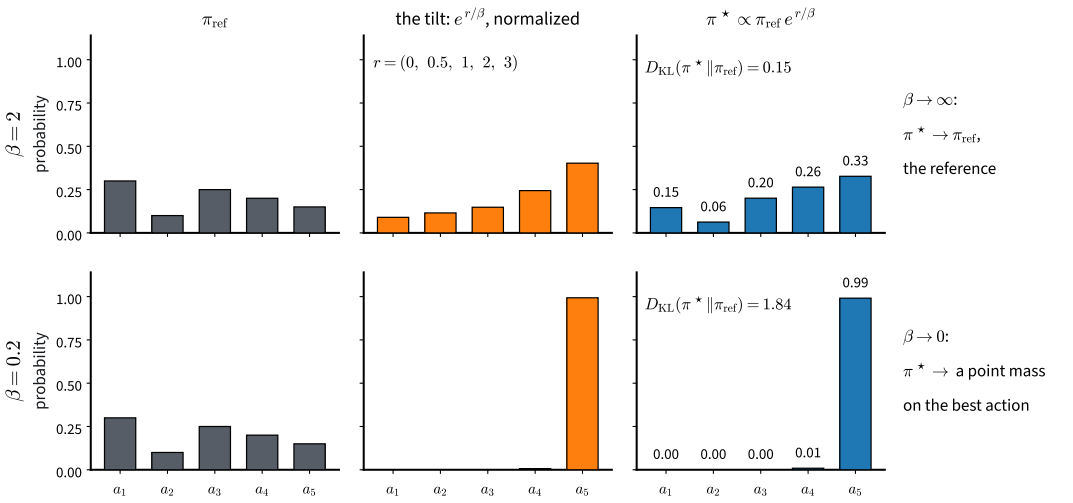


Figure 15.7: 積として表されるペナルティ付き目的関数の最適解。各行は参照方策 $\pi_{\text{ref}} = (0.30,\ 0.10,\ 0.25,\ 0.20,\ 0.15)$ にティルト $e^{r/\beta}$（報酬は $r = (0,\ 0.5,\ 1,\ 2,\ 3)$）を掛けて再正規化し、$\pi^\star$ を得る。上段は $\beta = 2$、下段は $\beta = 0.2$ を用いている。参照方策における a_2 での落ち込みは、上段では $\pi^\star$ にも残るが、下段では押し流される。余白の注記は 2 つの極限を示しており、$\beta \to \infty$ では参照方策に戻り、$\beta \to 0$ では最良の行動上の点質量となる。

り、Equation 15.27 は報酬がいくらであっても a に確率 0 を割り当てる。第 2 に、$\beta \to 0$ とすると、最適解は π_{ref} の台の中で最良の行動に集中する。最良の行動が複数同点で並ぶ場合、極限はその同点集合上で参照方策を再正規化した分布となる。

閉形式を一切知らない反復法で Equation 15.26 を解くことにより、この命題を数値的に確認しよう。`solve_kl` は指数勾配上昇法（exponentiated gradient ascent）を実行する。すなわち、現在の方策にステップサイズと勾配 $r - \beta(\log(\pi/\pi_{\text{ref}}) + 1)$ の積の指数を掛け、単体（simplex）上に再正規化する。$+1$ はすべての行動で共通であり正規化で相殺されるため、コードでは省略している。

```
p_ref = np.array([0.30, 0.10, 0.25, 0.20, 0.15])
r = np.array([0.0, 0.5, 1.0, 2.0, 3.0])

def solve_kl(r, p, beta, steps=3000, eta=0.3):
    pi = np.full_like(p, 1 / len(p))
    for _ in range(steps):                    # exponentiated gradient ascent
        pi = pi * np.exp(eta * (r - beta * np.log(pi / p)))
        pi /= pi.sum()
    return pi

for beta in (5.0, 2.0, 0.5, 0.2):
    pi = solve_kl(r, p_ref, beta)
    print(f'beta = {beta:3}: pi_beta = {np.round(pi, 3)}, '
          f'reward {pi @ r:.2f}, KL {(pi * np.log(pi / p_ref)).sum():.2f}')
```

```
beta = 5.0: pi_beta = [0.233 0.086 0.237 0.232 0.212], reward 1.38, KL 0.02
beta = 2.0: pi_beta = [0.146 0.062 0.2   0.264 0.327], reward 1.74, KL 0.15
beta = 0.5: pi_beta = [0.004 0.004 0.025 0.148 0.819], reward 2.78, KL 1.26
```

```
beta = 0.2: pi_beta = [0.    0.    0.    0.009 0.991], reward 2.99, KL 1.84
```

β が 5 から 0.2 へ下がるにつれて、期待報酬は 1.38 から 2.99 へ上昇し、ダイバージェンスは 0.02 から 1.84 nat へ増加する。Figure 15.7 の 2 つの行は、$\beta = 2$（ダイバージェンス 0.15 nat）および $\beta = 0.2$ における最適解である。ここで、収束した各反復解を Equation 15.27 と比較し、達成された目的関数値を $\beta \log Z$ と比較する。`pi_star` は閉形式を評価する際、オーバーフローを防ぐために指数の内部で最大報酬を差し引く（このシフトは正規化で打ち消し合う）。

```
def pi_star(r, p, beta):
    star = p * np.exp((r - r.max()) / beta)    # eq_kl_optimum, stabilized
    return star / star.sum()

gap = max(np.abs(solve_kl(r, p_ref, b) - pi_star(r, p_ref, b)).max()
          for b in (5.0, 2.0, 0.5, 0.2))
pi2 = solve_kl(r, p_ref, 2.0)
print(f'largest gap to the closed form over the ladder: {gap:.1e}')
print(f'objective at beta = 2: numerical '
      f'{pi2 @ r - 2.0 * (pi2 * np.log(pi2 / p_ref)).sum():.6f}, '
      f'beta log Z = {2.0 * np.log(p_ref @ np.exp(r / 2.0)):.6f}')
```

```
largest gap to the closed form over the ladder: 1.4e-16
objective at beta = 2: numerical 1.442004, beta log Z = 1.442004
```

反復解はすべての β において浮動小数点の精度まで閉形式と一致し、目的関数値も小数点以下 6 桁まで $\beta \log Z$ と一致する。

15.3.3.2 閉形式から導かれる 4 つの帰結

ペナルティがなければ、最適解は点質量となる。 Equation 15.27 において $\beta \to 0$ とすると、全確率質量が最高報酬の行動（同点がある場合は同点集合上での参照方策の分布）へと集中する。これは正則化なしの問題であり、その解は点質量（point mass）、すなわち argmax である。この主張は最適解に関するものであり、特定の訓練実行の経過について述べたものではない。15.2 節における PPO 方策のエントロピーが訓練中に低下したのは、軽くエントロピー正則化された PPO の目的関数が持つほぼ決定論的な最適解へと向かう動きと整合している。より大きな重みを持つペナルティを課せば、確率的方策が最適となる。

係数は参照方策と argmax の間を補間する。 β が非常に大きい場合と非常に小さい場合における閉形式を確認してみよう。

```
for beta in (100.0, 0.01):
    print(f'beta = {beta:6}: pi_star = {np.round(pi_star(r, p_ref, beta), 3)}')
```

```
beta =  100.0: pi_star = [0.297 0.099 0.25  0.202 0.153]
beta =   0.01: pi_star = [0. 0. 0. 0. 1.]
```

大きい β は参照方策を再現し、小さい β は貪欲（greedy）になり、その中間のあらゆる方策はある β の値に対する厳密な最適解となる。

一様な参照方策はペナルティをエントロピーボーナスに変える。 一様な参照方策に対しては $D_{\mathrm{KL}}(\pi \| \text{uniform}) = \log|\mathcal{A}| - H(\pi)$ となるため、ペナルティはエントロピーに対するボーナス $\beta H(\pi)$ と定数の和になり、Equation 15.26 は**最大エントロピー**（maximum-entropy）目的関数となる。閉形式は報酬のソフトマックスへと特殊化される。

```
boltzmann = np.exp(r / 0.5) / np.exp(r / 0.5).sum()
print(np.allclose(pi_star(r, np.full(5, 0.2), 0.5), boltzmann))
```

```
True
```

これは 14.7 節のソフトマックス方策と同じ形である。したがって、14.4 節のボルツマン探索（Boltzmann exploration）規則は、β を温度とみなせば、エントロピーボーナスを持つ 1 ステップ目的関数の厳密な最適解にほかならない。

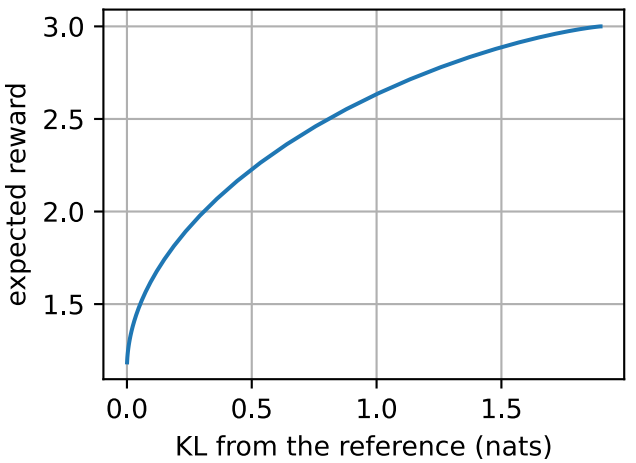
最適解は事後分布である。閉形式 Equation 15.27 はベイズの規則と同じ形をしている。すなわち、π_{ref} が事前分布（prior）、$e^{r/\beta}$ が非正規化尤度、$\pi^\star$ が事後分布（posterior）、Z が正規化定数である。推論としての制御（control-as-inference）の諸手法はこの恒等式を土台とし、報酬を「最適性」と呼ばれる事象の対数尤度、正則化方策最適化を変分推論（variational inference）として解釈する (Levine 2018; Korbak et al. 2022)。

15.3.3.3 1 回の意思決定におけるフロンティア

各 β は期待報酬と参照方策からのダイバージェンスを持つ 1 つの最適解を与える。β を掃引すると、各ダイバージェンスで達成可能な最良の報酬、すなわち**報酬・ダイバージェンスフロンティア**（reward-divergence frontier）が描かれる。

```
front = np.stack([pi_star(r, p_ref, b) for b in np.logspace(1.5, -1.5, 61)])
kl_f = (front * np.log(front / p_ref)).sum(1)
print(f'KL 0 pays {p_ref @ r:.2f}; the frontier ends at KL '
      f'{np.log(1 / p_ref[r.argmax()]):.2f}, reward {r.max():.0f}')
d2l.plot(kl_f, front @ r, 'KL from the reference (nats)', 'expected reward')
```

```
KL 0 pays 1.15; the frontier ends at KL 1.90, reward 3
```



このフロンティアは参照方策の平均報酬 1.15 から始まり、最良の行動上の点質量（ダイバージェンスは $\log(1/0.15) = 1.90$ nat）で終わる。この曲線は単調増加かつ凹であり、傾きは β（β に対する最適解での値）となるため（演習問題 5）、ダイバージェンスを 1 nat 増やすごとに得られる報酬の増分は逓減していく。Figure 15.7 の 2 つの行はこの曲線上の 2 点である。Best-of-N サンプリング（参照方策から N 個の回答を抽出し、最も報酬の高いものを残す手法）は、一切の訓練なしにこれと同じ軸に沿って移動する。その参照方策からのダイバージェンスは高々

$\log N - (N-1)/N$ であり、約 1.8 nat（$N = 16$ のとき）である (Gao et al. 2023)。グリッドワールドでも同様の曲線をたどるが、そこでは最適化対象の報酬ではなく真の報酬で採点することになる。

15.3.3.4 信頼領域とペナルティの比較

信頼領域と固定参照ペナルティはいずれも KL ダイバージェンスを用いるが、その役割は異なる。

> **KL ダイバージェンスの 2 つの用途。** **信頼領域**（15.2 節）は**直前の反復解**からのダイバージェンスを測る。各更新を制限して局所近似を保護するものであり、その参照先は更新のたびに移動する。一方、**固定参照ペナルティ**（本節）は目的関数の一部であり、最適解を Equation 15.27 へと移動させる。言語モデルの PPO 流のファインチューニングではこの両方が使われる。すなわち、安定した更新のためにサンプリング方策に対するクリップ付き代理目的関数を用い、最終的な方策を制御するために凍結した初期モデルに対する報酬ペナルティを用いる (Ouyang et al. 2022)。

閉形式を見れば、信頼領域だけを用いた場合にどこへ行き着くかがわかる。各更新が、直前の反復解からのダイバージェンスに重み $\tilde{\beta}$ を置いた信頼領域問題 $\pi_{k+1} = \operatorname{argmax}_{\pi} \mathbb{E}_{\pi}[r] - \tilde{\beta} D_{\mathrm{KL}}(\pi \| \pi_k)$ を、$\pi_0 = \pi_{\mathrm{ref}}$ から出発して厳密に解くものとする。命題より、各ステップは方策に $e^{r/\tilde{\beta}}$ を掛けるため、k ステップ後には次のようになる。

$$\pi_k(a) = \frac{\pi_{\mathrm{ref}}(a)\, e^{k\, r(a)/\tilde{\beta}}}{\sum_{a'} \pi_{\mathrm{ref}}(a')\, e^{k\, r(a')/\tilde{\beta}}}, \tag{15.28}$$

これは係数 $\tilde{\beta}/k$ を持つ閉形式にほかならない。k が大きくなるにつれてこの係数は 0 に近づき、反復解は最良の行動上の点質量へと収束する。信頼領域が定めるのは経路であって到達点ではなく、固定された参照方策だけが反復解の収束先を変えるのである（演習問題 6）。

15.3.3.5 ダイバージェンスの向き

ペナルティ $D_{\mathrm{KL}}(\pi \| \pi_{\mathrm{ref}})$ は 37.3.2 節の分類における**逆方向**（reverse）のダイバージェンスであり、π のもとでの期待値であるため方策自身のサンプルから推定できる。閉形式を見ると、ある行動 a の報酬が最良の行動より $\Delta(a) > 0$ だけ劣るときにどう扱われるかがわかる。最良の行動に対する相対比で見ると、$\pi^\star$ のもとでのその確率は参照確率の $e^{-\Delta(a)/\beta}$ 倍となる。一方、順方向（forward）のペナルティ $D_{\mathrm{KL}}(\pi_{\mathrm{ref}} \| \pi)$ は、参照方策がとる行動を 1 つでも切り捨てる方策に対して無限大となり、その最適解はすべての行動に対して参照確率の $\beta/\Delta(a)$ のオーダーの下限を保つ（演習問題 7）。どちらの向きでも、任意の $\beta > 0$ に対して、参照方策がとるすべての行動に正の確率が保たれる。両者の違いは、β が小さくなるときに劣った行動の確率がどれだけ速く減衰するかにあり、逆方向のペナルティでは $1/\beta$ に関して指数関数的に減衰するのに対し、順方向のペナルティでは β に関して線形にしか減衰しない。

15.3.4 ソフトバックアップ

グリッドワールドは 1 回きりの意思決定ではない。マルコフ決定過程（Markov decision process、MDP）では毎ステップでペナルティが課され、状態 0 からの目的関数は次のようになる。

$$\mathbb{E}_{\pi}\Big[\sum_{t \geq 0} \gamma^t \big(r(s_t, a_t) - \beta\, D_{\mathrm{KL}}\big(\pi(\cdot \mid s_t) \| \pi_{\mathrm{ref}}(\cdot \mid s_t)\big)\big)\Big]. \tag{15.29}$$

各状態において、方策は先ほど解いたような 1 回の意思決定問題に直面する。問題は、その意思決定をどの報酬によってティルトすべきかである。

15.3.4.1 終端の 1 ステップ前

エピソードの最後の意思決定では、行動の価値はその報酬そのものであり、先の命題がそのまま当てはまる。すなわち、そこでの最適解は参照方策を $e^{r/\beta}$ でティルトしたものであり、その状態の価値は $\beta \log Z$ である。その 1 ステップ前では、ある行動の価値はその行動の報酬と遷移先の状態の価値との和になる。その遷移先の価値とは、**そこで**の最適方策がペナルティのもとで達成する値 $\beta \log Z$ であり、そこで利用可能な最良の報酬よりも小さい。最終ステップにおける最適方策は、その最良の行動をある確率でしかとらない上、ペナルティも課されるからである。

2 ステップの例を考えると、この違いが具体的になる。Figure 15.7 の 5 行動の意思決定（$\beta = 2$）をエピソードの最終ステップとしよう。その価値は $\beta \log Z = 1.44$ であり、最良の報酬 3 よりも小さい。その 1 ステップ前で、エージェントは即座には報酬を生まないこの意思決定へ進むか、それとも確実に報酬 2 を得て終了するかのいずれかを選ぶものとし、参照方策はそれぞれを確率 1/2 で選び、割引はないものとする。次のコードセルでは、最初の意思決定をソフト価値（soft value）1.44 でティルトした場合と最良の報酬 3 でティルトした場合のそれぞれについて、2 ステップ方策のペナルティ付き目的関数を評価する。

```
beta, stop = 2.0, 2.0                          # stopping pays 2 for sure
V_next = beta * np.log(p_ref @ np.exp(r / beta))   # value of the last step
for name, q in (('soft value', V_next), ('best reward', r.max())):
    pi0 = pi_star(np.array([q, stop]), np.array([0.5, 0.5]), beta)
    J = pi0 @ [V_next, stop] - beta * (pi0 * np.log(pi0 / 0.5)).sum()
    print(f'tilt by the {name} {q:.2f}: pi(enter) = {pi0[0]:.2f}, '
          f'objective {J:.3f}')
Z0 = 0.5 * np.exp(V_next / beta) + 0.5 * np.exp(stop / beta)
print(f'beta log Z of the first decision: {beta * np.log(Z0):.3f}')
```

```
tilt by the soft value 1.44: pi(enter) = 0.43, objective 1.740
tilt by the best reward 3.00: pi(enter) = 0.62, objective 1.592
beta log Z of the first decision: 1.740
```

ソフト価値でティルトすると、最初の意思決定における最適値 $\beta \log Z$ が達成される。一方、最良の報酬でティルトすると、2 番目の意思決定を 3 と評価してしまうが、そこでのペナルティ付き方策は実際には 1.44 しか得られないため、進入しすぎて約 0.15 を失う。ティルトには、後続のあらゆる報酬**と**後続のあらゆるペナルティの両方を織り込んだ価値を用いなければならない。

15.3.4.2 ソフト価値反復法

この価値をすべての状態で書き下すと一般形が得られる。ペナルティ付き目的関数のもとでの行動価値を $Q_\beta(s,a)$（行動 a を状態 s でとったときの価値）、状態価値を $V_\beta(s)$（状態 s の価値）とする。すべての状態において、報酬の役割に $Q_\beta(s,\cdot)$ を置いて先の命題を適用すると、次式が得られる。

$$
\begin{aligned}
V_\beta(s) &= \beta \log \sum_a \pi_{\text{ref}}(a \mid s)\, e^{Q_\beta(s,a)/\beta}, \\
Q_\beta(s,a) &= r(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V_\beta(s'),
\end{aligned}
\tag{15.30}
$$

そして最適方策は $\pi_\beta(a \mid s) \propto \pi_{\text{ref}}(a \mid s)\, e^{Q_\beta(s,a)/\beta}$ となる。これは 14.2 節のベルマン最適方程式において、行動に関する最大値を参照方策で重み付けした log-sum-exp、すなわち $\beta \to 0$ で参照方策の台上のハード最大値へと収束する**ソフト最大値**（soft maximum）に置き換えたものである。14.3 節で示したように、行動価値がノイズを含む推定値であるときハード最大値は上方に統計的バイアスを持つが、log-sum-exp も凸関数であるため、程度は穏やかながら同様に上方へのバイアスを持つ。参照方策が一様である場合、これは最大エントロピー強化学習におけるソフト価値反復法（soft value iteration）に一致する。

価値反復法は、ハードな方程式を $V \leftarrow \max_a \text{backup}(V)$ という掃引によって $V = 0$ から解いた。ソフト最大値を用いて同じ掃引を行えば Equation 15.30 が解ける。ハードな掃引が縮小写像であることの証明（14.2.2.3 節）で使ったのは、最大値が持つ 2 つの性質だけであった。単調であることと、すべての要素に定数を加えると最大値にもその定数が加わることである。ソフト最大値はこの両方を満たすため、ソフトな掃引もまた γ 縮小写像となる（任意の $\beta > 0$ および任意の参照方策に対して成立する。演習問題 3）。`soft_max` は行動価値の配列の各行について、数値安定性のために行の最大値を差し引いてからソフト最大値を計算し、ティルトされた方策とあわせて返す。`soft_v` は `backup` に続けて `soft_max` を適用する掃引を繰り返す。

```
def soft_max(Q, p, beta):
    """Row-wise beta log sum_a p(a) exp(Q(a) / beta), and the tilted policy."""
    m = Q.max(1, keepdims=True)                # subtracted for stability
    tilt = p * np.exp((Q - m) / beta)
    V = m[:, 0] + beta * np.log(tilt.sum(1))
    return V, tilt / tilt.sum(1, keepdims=True)

def soft_v(mdp, p, beta, num_iters=400):
    """Soft value iteration V <- soft_max(backup(V)); returns V and pi_beta."""
    V = np.zeros(mdp.num_states)
    for _ in range(num_iters):
        V, pi = soft_max(mdp.backup(V), p, beta)
    return V, pi
```

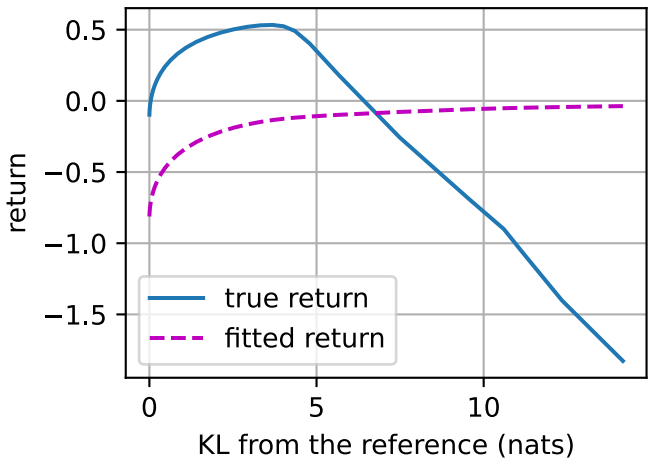
15.3.4.3 KL 予算に対する真の収益

ここでグリッドワールド上のフロンティアをたどるが、1 つ変更を加える。オプティマイザには当てはめ報酬 $\hat{r}$ を見せ、採点は真の報酬で行うのである。36 個の β の値（10 から $10^{-2.5}$ まで）のそれぞれについて、`soft_v` で厳密なペナルティ付き最適方策 π_β を $\hat{r}$ に対して計算し、その真の収益、当てはめ収益、および参照方策からのダイバージェンスを記録する。ダイバージェンスとしては、MDP の目的関数のペナルティ項である $\mathbb{E}_{\pi_\beta}\big[\sum_t \gamma^t D_{\text{KL}}(\pi_\beta(\cdot \mid s_t) \| \pi_{\text{ref}}(\cdot \mid s_t))\big]$（開始状態は状態 0）を用いる。これは、$\pi_\beta$ と π_{ref} が生成する軌跡の分布間の KL ダイバージェンス（各ステップ後に確率 $1-\gamma$ でもエピソードが終了するとした場合）に等しい（演習問題 2）。コードでは、状態ごとのダイバージェンス `kl` に、状態 0 からの各状態への期待割引訪問回数 `visits`（π_β の遷移行列を用いた 1 回の線形方程式の求解で得られる）を重みとして掛ける。あわせて、掃引全体にわたる Equation 15.30 の最大残差も記録する。

```
m_hat = d2l.TabularMDP(P, r_hat, gamma)
kls, true_ret, fit_ret, resid = [], [], [], 0.0
for beta in np.logspace(1, -2.5, 36):
    V, pi = soft_v(m_hat, pi_ref, beta)
    V_new, _ = soft_max(m_hat.backup(V), pi_ref, beta)
    resid = max(resid, np.abs(V_new - V).max())
```

```
        kl = (pi * np.log(np.where(pi > 0, pi / pi_ref, 1.0))).sum(1)
        P_pi = np.einsum('sa,sat->st', pi, P)
        visits = np.linalg.solve(np.eye(16) - gamma * P_pi.T, np.eye(16)[0])
        kls.append(visits @ kl)                    # discounted KL from state 0
        true_ret.append(ret(pi, r_true))
        fit_ret.append(ret(pi, r_hat))
    k = int(np.argmax(true_ret))
    print(f'largest soft Bellman residual: {resid:.0e}')
    print(f'reference: true return {ret(pi_ref, r_true):+.2f}; beta = 10: '
          f'KL {kls[0]:.3f}, true return {true_ret[0]:+.2f}')
    print(f'peak: true return {true_ret[k]:+.2f} at KL {kls[k]:.1f}; '
          f'smallest beta: {true_ret[-1]:+.2f} at KL {kls[-1]:.1f}')
    d2l.plot(kls, [true_ret, fit_ret], 'KL from the reference (nats)', 'return',
             legend=['true return', 'fitted return'])
```

```
largest soft Bellman residual: 0e+00
reference: true return -0.12; beta = 10: KL 0.001, true return -0.10
peak: true return +0.53 at KL 3.7; smallest beta: -1.83 at KL 14.2
```



プロット上の各点はペナルティ付き目的関数の厳密な最適解であり、最大のベルマン残差は 10^{-10} 未満である。当てはめ収益は一貫して上昇し、厳密な最適解である限り必ずそうなる。実際、$\beta_1 < \beta_2$ とし、それぞれの最適解が当てはめ収益 J_1, J_2 とダイバージェンス D_1, D_2 を持つとすると、最適性より $J_1 - \beta_1 D_1 \geq J_2 - \beta_1 D_2$ および $J_2 - \beta_2 D_2 \geq J_1 - \beta_2 D_1$ が成り立つ。この 2 式を足し合わせると $(\beta_2 - \beta_1)(D_1 - D_2) \geq 0$ となるため $D_1 \geq D_2$ が従い、これを第 1 の不等式に用いれば $J_1 \geq J_2$ が得られる。

真の収益は異なる振る舞いを見せる。参照方策の -0.12 から上昇して $+0.53$（3.7 nat の地点）に達し、真の最適値 $+0.59$ に迫るが、その後は方策が危険レーンに集中するにつれて -1.83 へと低下する。当てはめ報酬への適度な最適化は真の収益を改善するが、強く押しすぎるとデータが一度も計測しなかった移動を悪用してしまう。24 個の選好データセットにわたって調べると、ピークは数 nat の地点で約 $+0.5$ から最適値 $+0.59$ の間に位置し、そのうち 21 個ではピークの後に真の収益が低下する。残る 3 個では掃引範囲内で低下しない。1 個では当てはめ最適方策が真の最適方策と一致しており、2 個では安全な経路と危険な経路の当てはめ価値がほぼ同点となるため、ペナルティ付き最適方策が $\beta = 10^{-2.5}$ まで安全な経路を維持するからである。ピークが参照方策からどれだけ離れた位置に現れ、その後の低下がどれほど深くなるかは、比較データ

が危険セルをどれだけカバーしているかと、そのコストの大きさとに依存する。Gao et al. (2023) は、学習された報酬モデルに対して最適化しホールドアウトされた報酬モデルで採点した言語モデルにおいて、これと同じ上昇と低下を測定している。

オプティマイザに見えるのは当てはめ曲線だけであり、その曲線は単調に上昇するため、真の収益のピークはオプティマイザからは見えない。したがって実務では、訓練に使った報酬モデル以外の何か（ホールドアウトされた比較データ、人間の判定、あるいはチェッカー）に対して方策を評価することで β を選ぶ。参照方策が一様であれば同じペナルティはエントロピーボーナスとなり、15.5 節ではこれを最大化する連続行動向けのオフポリシー（off-policy）アクター・クリティックを構築する。そのクリティックは log-sum-exp バックアップを反復するのではなく、エントロピーボーナス付きの報酬のもとで自身の方策を評価するが、先の命題の証明にある恒等式により、方策がティルトされたものであれば両者は一致する。また 15.8 節では、閉形式 Equation 15.27 を第 14 章の検証器（verifier）タスクに適用し、保険をかけた回答（hedged answer）が割に合わなくなる β を予測する。ここまでの手法はすべて、現在の方策から新たに得たサンプルか既知のモデルから学習してきた。続く 15.4 節では、14.3 節の Q テーブルが行ったように、任意の方策から蓄積された遷移データからネットワークが行動価値を学習できるかを問う。

15.3.5 まとめ

比較データに当てはめた報酬は、比較データが存在する領域でのみ正確である。共通の開始状態から出発する軌跡間の比較では報酬のポテンシャルシェイピングを見分けることができず、データが一度もカバーしない状態行動対は当てはめの初期値のままである。オプティマイザはそうした隙を突く。グリッドワールドでは、当てはめ報酬上の計画は参照方策がめったに入らなかった危険セルを通り抜ける。参照方策からの KL ダイバージェンスにペナルティを科すと目的関数が変わり、その厳密な最適解は参照方策を $e^{r/\beta}$ でティルトしたものとなって、最適値は $\beta \log Z$ となる。参照方策が一様であれば、このペナルティはエントロピーボーナスになる。MDP においてティルトはソフト行動価値を用い、これは最大値の代わりに log-sum-exp を持つベルマン方程式を満たす。ペナルティの重みを掃引すると、当てはめ収益は単調に上昇する一方、真の収益は適度なダイバージェンスでピークに達した後に低下するため、重みはオプティマイザが参照しない評価によって選ばなければならない。

15.3.6 演習問題

1. **識別可能性とスケール**。ブラッドリー・テリーモデル Equation 15.25 を、同一のプロンプトに対する 2 つの応答 y と y'（プロンプトは x）の比較データに当てはめる。
   1. $r(x, y)$ を $r(x, y) + f(x)$（f はプロンプトの任意の関数）に置き換えても、このようなあらゆるデータセットの尤度が変化しないことを示せ。逆に、x に対する 2 つの応答がデータセット内で比較されているときに限り、両者を辺で結ぶグラフを考える。比較されたすべての対に同じ選好確率を割り当てる 2 つの報酬は、このグラフの各連結成分上で定数差しか持たないことを示せ。1 つのプロンプトに対するグラフが複数の連結成分を持つとき、何が識別されるか。
   2. アノテータがある一貫性パラメータのもとで $P(y \succ y') = \sigma\big(\omega\,(r^*(x, y) - r^*(x, y'))\big)$（ただし $\omega > 0$）に従ってラベル付けし、無限のデータに対して最尤法により Equation 15.25 を当てはめるものとする。この当てはめはどの報酬を復元するか。
   3. 当てはめ報酬に対する最適解 Equation 15.27（係数は β）が、r^* に対する係数 β/ω での最適解と一致することを示せ。あるアノテータ集団のラベルに当てはめた報酬モデルに対して β を調整したとする。同じ β を、より一貫性の低い集団（すなわち ω がより小さい集団）のラベルに当てはめた報酬モデルに用いた場合、どれほど強いペナルティとして働くか。

2. [code] **軌跡のダイバージェンス**。$p_\pi(\tau)$ を、方策 π が状態 0 から出発して固定ホライズンの軌跡 $\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$（長さ T）を生成する確率とする。
   1. 次式を示せ。
$$D_{\mathrm{KL}}(p_\pi \| p_{\pi_{\mathrm{ref}}}) = \mathbb{E}_\pi\Big[\sum_{t=0}^{T-1} D_{\mathrm{KL}}(\pi(\cdot \mid s_t) \| \pi_{\mathrm{ref}}(\cdot \mid s_t))\Big].$$
   $p_\pi(\tau)$ のどの因子が相殺されるか。また、それゆえに遷移のモデルを持たなくても π の軌跡からこのダイバージェンスを推定できるのはなぜか。
   2. 代わりに、軌跡が行動とは独立に各ステップの後で確率 $1-\gamma$ で終了するものとする。2 つの軌跡分布間のダイバージェンスが、15.3.4.3 節の掃引で `kls` に `visits @ kl` として追加した状態ごとのダイバージェンスの割引和になることを示せ。
   3. その掃引において真の収益がピークとなる方策 π_β をとれ。$\mathbf{c}(\tau)^\top \log(\pi_\beta/\pi_{\mathrm{ref}})$（対数は要素ごとにとり $\mathbf{c}(\tau)$ と同様に平坦化する）を、`sample_traj(mdp, pi, rng, T=400)` から生成した 2000 本の軌跡にわたって平均し、ピーク時点で出力されたダイバージェンス 3.7 nat と比較せよ。軌跡ごとの値の中には負になるものがある。これがギブスの不等式と矛盾しないのはなぜか。
3. [code] **湖の上のソフトバックアップ**。14.2.2.3 節の証明では、最大値の 1 つの性質 $|\max_a f(a) - \max_a g(a)| \le \max_a |f(a) - g(a)|$ を用いた。
   1. $m_\beta(f) = \beta \log \sum_a \pi_{\mathrm{ref}}(a)\, e^{f(a)/\beta}$ とおく。m_β が単調であり、かつ $m_\beta(f + c) = m_\beta(f) + c$ が任意の定数 c に対して成り立つことを示し、そこから m_β が最大値と同じ性質を持つことを導け。以上より、バックアップ Equation 15.30 が γ 縮小写像であることを結論せよ（任意の $\beta > 0$ および任意の参照方策に対して成立する）。
   2. `soft_v` を 14.2 節の滑りやすい湖（`d2l.TabularMDP` として構築したもの）に一様な参照方策のもとで適用せよ。$V_\beta(s_0)$ を β に対して 10^{-4} から 10^{2} までの対数グリッド上でプロットせよ。この曲線は両端において、14.2 節で計算したどの 2 つの値に近づくか。正則化目的関数を用いて、この曲線が単調になる理由を説明せよ。
   3. およそどの β において、$V_\beta(s_0)$ は 2 つの極限の中間に位置するか。この係数を、湖における最適行動価値 $Q^*(s_0, a)$ どうしの差と比較せよ。
4. [extended] **ニューラル報酬モデル**。●`r_hat` の線形当てはめではデータのない対が 0 付近にとどまるのに対し、ニューラル報酬モデルはその帰納バイアスに従って外挿する。64 個の重み `w` を、64 個の ReLU ユニットからなる隠れ層を 2 層持ち状態行動対ごとに 1 つの出力を出す多層パーセプトロンの出力に置き換え、ゴールの行は 0 に固定せよ。これを同じ `D` と `y` 上で、ブラッドリー・テリーモデルの負の対数尤度を用い、学習率 0.01 の Adam によってフルバッチで 2000 ステップ訓練せよ。
   1. 入力を状態と行動のワンホット符号としたモデルと、入力を $[0, 1]$ にスケーリングした状態の 2 つのグリッド座標および行動のワンホット符号としたモデルを、それぞれ 4 つのシードで訓練せよ。各モデルについて、`core` 上の RMS 誤差、危険セルへ入る移動 `into` の当てはめ報酬、および `plan` の真の収益を報告せよ。
   2. 各当てはめ報酬に対して 15.3.4.3 節の掃引を繰り返せ。ペナルティを必要とするのはどちらの表現か。入力のどの性質がその違いを説明するか。
5. **フロンティアの傾き**。●$u = 1/\beta$ とおき、$R(u)$ と $D(u)$ をそれぞれ期待報酬とダイバージェンス $D_{\mathrm{KL}}(\pi^\star \| \pi_{\mathrm{ref}})$（最適解 Equation 15.27 における値）とする。
   1. $R(u) = \frac{d}{du} \log Z$ および $R'(u) = \mathrm{Var}_{\pi^\star}(r)$ を示せ。次に、命題の最適値 $\beta \log Z$ を用いて $D'(u) = u\, \mathrm{Var}_{\pi^\star}(r)$ を示せ。
   2. `front` と `kl_f` が描くフロンティアは、傾きが β（点 $\pi^\star$ において）であり、r が π_{ref} の台上で定数でない限り単調増加かつ凹であることを結論せよ。
   3. $\pi^\star$ が制約付き問題 $\max_\pi \mathbb{E}_\pi[r]$（制約条件は $D_{\mathrm{KL}}(\pi \| \pi_{\mathrm{ref}}) \le D(u)$）の解でもあることを示せ。$\beta$ はこの制約付き問題におけるどの量に対応するか。

4. Best-of-N サンプリングは、N 個の独立標本を π_{ref} から抽出し、最も報酬の高いものを残す。参照方策のもとで報酬が連続分布を持つとき、Best-of-N 分布の参照方策からのダイバージェンスが $\log N - (N-1)/N$ であることを示せ。報酬が互いに異なり単調増加である 5 行動バンディット `p_ref`、`r` に対しては、Best-of-N は行動 a を確率 $F(a)^N - F(a-1)^N$ で選ぶ（ここで $F(a)$ は a 以下の行動に対する参照確率の和）。$N = 1, \dots, 16$ に対してその期待報酬をダイバージェンスに対して `front` および `kl_f` とともにプロットせよ。フロンティアにどれだけ近づくか。また、そのダイバージェンスは $\log N - (N-1)/N$ 以下にとどまるか。

6. **移動する参照方策。**●15.2 節の信頼領域は直前の反復解からのダイバージェンスを測る。本節では、理想化された更新 $\pi_{k+1} = \operatorname{argmax}_\pi \ \mathbb{E}_\pi[r] - \tilde{\beta}\, D_{\text{KL}}(\pi \| \pi_k)$ を $\pi_0 = \pi_{\text{ref}}$ から開始すると argmax に収束することを示した。
   1. ここで PPO 流のファインチューニングと同様に Equation 15.26 の固定参照ペナルティを加え、π_{k+1} が次式を最大化するものとする。

$$\mathbb{E}_\pi[r] - \beta\, D_{\text{KL}}(\pi \| \pi_{\text{ref}}) - \tilde{\beta}\, D_{\text{KL}}(\pi \| \pi_k). \tag{15.31}$$

   定数差を除いて次式が成り立つことを示せ。

$$\log \pi_{k+1} = \frac{\beta \log \pi_{\text{ref}} + \tilde{\beta} \log \pi_k + r}{\beta + \tilde{\beta}}, \tag{15.32}$$

   さらに π_k の極限を求めよ。極限を決定するのはどちらの係数か。また、どのくらいの速さで極限に近づくか。
   2. 小問 a の漸化式が、適切なステップサイズを用いた `solve_kl` の更新と一致することを示し、そのステップサイズを β と $\tilde{\beta}$ で表せ。`solve_kl` が収束するステップサイズのうち、信頼領域としての解釈を持たないものはどれか。

7. [code] **順方向 KL ペナルティ。**●15.3.3.5 節では Equation 15.26 の逆方向ダイバージェンスを順方向ダイバージェンスと対比した。ペナルティを順方向ダイバージェンスに置き換え、次の問題を考える。

$$\max_\pi \ \mathbb{E}_{a\sim\pi}\big[r(a)\big] - \beta\, D_{\text{KL}}(\pi_{\text{ref}} \| \pi). \tag{15.33}$$

   1. 乗数 m を制約 $\sum_a \pi(a) = 1$ に対して導入し、最適解が $\pi(a) = \beta\, \pi_{\text{ref}}(a)/(m - r(a))$ となることを示せ（ここで $m > \max_a r(a)$ はこれらの確率の和を 1 にする唯一の値である）。
   2. 準最適な行動の報酬ギャップを $\Delta(a) = \max_{a'} r(a') - r(a) > 0$（行動を a とする）とおく。$\beta \to 0$ のとき、$\pi(a)/\pi_{\text{ref}}(a)$ は順方向ペナルティのもとでどのくらいの速さで 0 に収束するか。また、逆方向ペナルティのもとではどうか。
   3. 5 行動バンディット `p_ref`、`r` に対し、m に関する二分法によって順方向ペナルティの最適解を計算せよ。それぞれのペナルティについて、β のうち最適解の期待報酬が 2.9 となる値を求め、2 つの最適解を行動ごとに比較せよ。順方向ペナルティのもとでより大きな確率が割り当てられるのはどの行動か。また、これは 15.3.3.5 節の内容とどのように整合するか。

15.4 深層 Q ネットワーク

14.3 節では、エージェントはいかなる振る舞いから生成された遷移からでも、1 遷移ずつ行動価値（action value）のテーブルを学習した。14.9 節では、方策（policy）のテーブルをネットワークに置き換え、CartPole 上で訓練した。Q 学習（Q-learning）でも同じことをするのは、一見す

ると 1 行の変更で済むように思える。ネットワークに 2 つの行動価値を出力させ、更新則をそのまま保てばよい。方策勾配（policy gradient）では、導出のどこにもテーブルという形式を使っていなかったため、この置き換えに代償は伴わなかった。しかし Q 学習では、テーブルがターゲット $r + \gamma \max_{a'} Q(s', a')$ の内部に入っている。テーブルがネットワークになると、更新のたびに予測とともにターゲットも動き、1 回の更新が一度も訪れていない状態の価値まで変化させ、表形式の手法を収束させていた縮小写像の議論はもはや通用しなくなる。ネットワークになっても、任意の遷移から学習できるというテーブルの特権を保てるだろうか。

2013 年、DeepMind の研究グループがこの置き換えを機能させた。過去の遷移の蓄積を用いて訓練することで、ネットワークが画面のピクセルから 7 種類の Atari ゲームをプレイできるようになったのである (Mnih et al. 2013)。研究グループはこれを深層 Q ネットワーク（deep Q-network、DQN）と名付けた。2015 年までには、同じネットワークとハイパーパラメータにターゲット計算用の凍結コピーを加えることで、49 種類のゲームをプレイし、その半数以上で人間のテスターと同程度の性能を達成した (Mnih et al. 2015)。学習を安定させた 2 つの仕組み、すなわち遷移の蓄積と凍結コピーは、それ以降のほぼすべてのオフポリシー深層強化学習手法に組み込まれている。またこの 2 つは、前節までの手法では得られなかった利点もエージェントにもたらす。アクター・クリティックは各バッチを 1 回しか使わず、PPO でも数エポックしか使わなかったのに対し、DQN は保存したすべての遷移を使って何度も繰り返し訓練する。

本節ではまず、サンプリングノイズが一切ないにもかかわらず重みが際限なく増大する 7 状態の例を用いて、素朴な置き換えが失敗し得る理由を示す。次に CartPole 上で DQN を構築し、凍結コピーを取り除いてそれが何を防いでいるかを確認したうえで、最後にターゲット内の最大値が生み出すバイアスと、それを軽減する小さな変更を扱う。15.3 節ではこの最大値をソフト最大値に置き換えたが、本節ではハード最大値のままとする。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

ここでも割引率 0.99 の CartPole を用いるが、方策の改善に伴ってエピソード長が変化するため、訓練予算は 50,000 環境ステップとして指定する。エージェントは 2 環境ステップごとに、保存された 128 個の遷移からなるミニバッチ上で 1 回の勾配ステップを実行する。Q ネットワークは状態を 128 ユニットの 2 つの隠れ層に通して行動ごとに 1 つの価値へ写像し、貪欲行動はその argmax とする。探索（exploration）は ϵ 貪欲とし、最初の 10,000 ステップにわたって ϵ を 1 から 0.05 まで線形に減少させる。これには `linear_schedule`（14.3 節）を用いる。`make_env` は環境を構築し、そのオプション引数 `copies` は各行動を複数回提示する（本節の最後の実験で用いるものであり、それまでは 1 である）。

```
gamma, num_env_steps, num_seeds = 0.99, 50_000, 3
buffer_size, batch_size, lr = 200_000, 128, 1e-3
train_freq, sync_every, warmup, grad_clip = 2, 250, 1_000, 10.0
epsilon = d2l.linear_schedule(1.0, 0.05, 10_000)
```

```
def make_env(copies=1):
    """CartPole; with copies > 1 each of its two actions is offered `copies`
    times (action a moves the cart as action a % 2), which leaves Q* as is."""
    env = gym.make('CartPole-v1')
    return env if copies == 1 else gym.wrappers.TransformAction(
        env, lambda a: a % 2, gym.spaces.Discrete(2 * copies))
```

次に、`make_qnet(seed, copies=1)` はシードを固定した 2 つの隠れ層を持つ多層パーセプトロン（multilayer perceptron、MLP）を構築し、4 次元の CartPole の状態を $2 \times$ copies 個の行動価値へ写像する。

```
def make_qnet(seed, copies=1):
    """A seeded Q-network with one output per action."""
    rngs = nnx.Rngs(seed)
    return nnx.Sequential(d2l.Linear(4, 128, rngs=rngs), jax.nn.relu,
                          d2l.Linear(128, 128, rngs=rngs), jax.nn.relu,
                          d2l.Linear(128, 2 * copies, rngs=rngs))
```

Q ネットワークの各層には `d2l.Linear` を用いる。その一様初期化については 14.9 節で説明したとおりである。今回の実行では、Flax のデフォルトの初期化を用いると CartPole を解けるシードが減少した。

15.4.1 不安定性の要因

15.4.1.1 半勾配更新

14.3 節のテーブルをネットワーク Q_w（パラメータは w）に置き換えても、更新則自体は変わらず、ブートストラップされたターゲットに向かう半勾配（semi-gradient）ステップのままである。遷移 (s, a, r, s') をサンプリングし、次のターゲットを作る。

$$y = r + \gamma \left(1 - \mathbf{1}(s' \text{ terminal})\right) \max_{a'} Q_w(s', a'), \tag{15.34}$$

そのうえで y を固定し、$Q_w(s, a)$ を y に近づけるオプティマイザのステップを 1 回実行する。指示関数は、s' でエピソードが終了するときにブートストラップを取り除く。各ステップは、固定された数値に向かう通常の回帰（regression）の 1 ステップである。しかし、この数値が w とともに動くため、ステップの列はいかなる固定された目的関数に対する降下にもなっていない。これは 14.9 節で、あらゆるブートストラップターゲットについて確認したとおりである。したがって、損失が下がっていることはネットワークが現在のターゲットに適合していることを示すにすぎず、ターゲット自体が正しいことを示すわけではない。

テーブルの場合、期待更新は価値反復法（value iteration）の緩和ステップであり、ベルマンバックアップは縮小写像となる（14.2 節）。すなわち、全状態にわたる最大誤差を係数 γ 倍に縮小する。一方、ネットワークはバックアップ後のあらゆる関数を表現できるわけではないため、各回帰は関数クラスの中でターゲットに最も近い関数を返す。この「近さ」は訓練データ上の最小二乗法（least squares）で測られ、各状態での誤差はデータがその状態を訪れる頻度に比例して重み付けされる。線形特徴量の場合、この当てはめは射影（projection）になる。バックアップが最大誤差を縮小するのに対し、当てはめは訪問頻度で重み付けされた平均誤差を縮小する。これら 2 つの尺度は一致しないため、重み付き誤差を減らす当てはめが、めったに訪れない状態での誤差をかえって拡大してしまうことがあり、次のバックアップがその誤差をそこへ至る状態のターゲットへと伝播させる。データが評価対象の方策自身から得られている場合は、2 つの尺度

が十分に整合するため線形特徴量では収束する (Tsitsiklis and Van Roy 1997)。しかし別の方策からのデータを用いると、線形特徴量であっても発散し得る。

15.4.1.2 相関したデータと移動するターゲット

実用上、2 つの結合（coupling）がネットワークの場合をテーブルの場合よりも難しくする。1 つ目はデータにおける結合である。カートとポールは 1 ステップごとにわずかしか動かないため、1 つのエピソード内の連続する遷移はほぼ同一であり、到着するストリームをそのまま使って訓練すると状態空間（state space）の狭い領域が何度も連続して更新される。テーブルは訪れた要素しか変更しないため、相関（correlation）があっても学習が遅くなるだけで済む。しかしネットワークはステップごとに多くの状態での予測を変化させるため（Figure 14.22）、似通った更新が続くと他の領域で学習した内容を上書きしてしまうことがある。

2 つ目の結合はターゲットにおける結合である。ターゲット y は Q_w を含むため、オプティマイザのステップごとに予測と回帰先の数値の両方が変化する。15.1 節のクリティックにも同じ結合があったが、更新のたびに現在の方策による新しい遷移で再当てはめしていたため、方策が訪れる状態での誤差はすぐに表面化して修正された。

15.4.1.3 死の三要素と Baird の反例

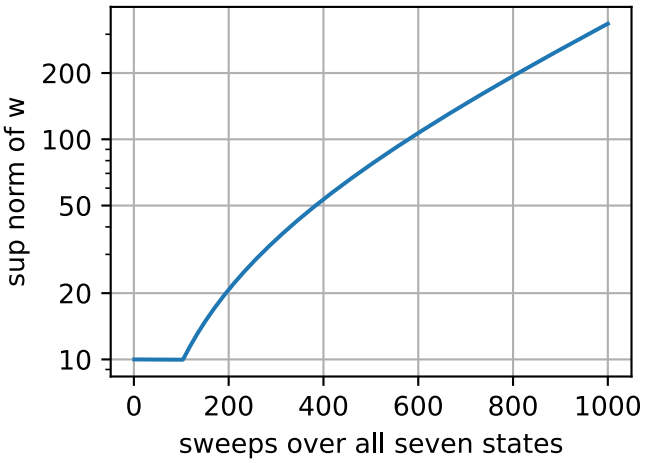
ネットワーク版の Q 学習では、**関数近似**（function approximation）、**ブートストラップ**、そして**オフポリシーデータ**（価値を学習したい方策以外の方策によって収集されたデータ）という 3 つの要素が同時に重なる。Figure 15.8 は、本章と前章のアルゴリズムをそれらが含む要素ごとに配置したものである。2 つの要素が重なる各領域には、それぞれの条件のもとで収束する手法が存在する。表形式の Q 学習はオフポリシーデータからブートストラップし、14.3 節のステップサイズ条件のもとで収束する。最も単純な形のアクター・クリティックにおけるクリティックである線形時間差分学習は、関数近似を用いてブートストラップし、データが評価対象の方策から得られる場合に線形特徴量のもとで収束する。行動クローニング（behavior cloning）はブートストラップを行わずにオフポリシーデータへネットワークを当てはめる手法であり、固定されたラベルへの教師あり回帰にほかならない。なお、これらは十分条件（sufficient condition）であって、2 要素の組み合わせ一般に関する定理ではない。非線形な時間差分学習はオンポリシーであっても発散し得る (Tsitsiklis and Van Roy 1997)。これら 3 つの要素がすべて揃う状況は**死の三要素**（deadly triad）と呼ばれる (Sutton and Barto 2018; Hasselt et al. 2018)。

Baird の反例は、7 状態の線形問題においてこの三要素だけを取り出して示す (Baird 1995)。すべての報酬は 0 であるため、真の価値関数（value function）は 0 であり、$w = 0$ によって厳密に表現される。計算には期待時間差分更新を用いるため、サンプリングノイズは存在しない。常に状態 7 へ遷移する方策を評価しながら、7 つの状態をオフポリシー分布として一様に重み付けする。

```
w = np.ones(8)                          # eight weights for seven states
w[6] = 10.0                             # the classic starting point
Phi = np.zeros((7, 8))                  # linear values v = Phi w
Phi[:6, :6], Phi[:6, 7] = 2 * np.eye(6), 1.0   # v(i) = 2 w_i + w_8
Phi[6, 6], Phi[6, 7] = 1.0, 2.0                # v(7) = w_7 + 2 w_8
sup = []
for _ in range(1000):
    v = Phi @ w                         # every reward is 0
    delta = gamma * v[6] - v            # the next state is always state 7
    w += 0.01 * (delta[:, None] * Phi).mean(0)   # uniform state weighting
    sup.append(np.abs(w).max())
print(f'sup norm of w after 0, 500, 1000 sweeps: 10, {sup[499]:.0f}, '
```

```
        f'{sup[-1]:.0f}')
print(f'the value the weights claim for state 7: {(Phi @ w)[6]:.0f}, '
      f'true value 0')
d2l.plot(np.arange(1, 1001), sup, 'sweeps over all seven states',
         'sup norm of w', yscale='log')
```

```
sup norm of w after 0, 500, 1000 sweeps: 10, 77, 335
the value the weights claim for state 7: 677, true value 0
```



過渡期を過ぎると、重みは 1 スイープあたり一定の倍率に収束するペースで増大する。1,000 スイープ後には重みの上限ノルム（sup norm）は 10 から 335 へと増大し、真の価値が 0 であるにもかかわらず、重みが主張する状態 7 の価値は 677 になる。ここには標本誤差（sampling error）は一切関与しておらず、ステップサイズを小さくしても発散が遅くなるだけで解消はされない。真の価値関数は表現可能であるため、原因は近似誤差でもない。オフポリシーの重み付けのもとでは、期待更新に重みが増大していく方向が存在するのである (Sutton and Barto 2018)。代わりに評価対象の方策自身の訪問頻度で状態を重み付けすると発散は止まるが、その重み付けが一度も訪れない状態では到達する価値が誤った値になる（演習問題 1）。

経験再生（experience replay）とターゲットネットワークは、三要素のどれかを取り除くわけではない。時間的相関を低減しターゲットの変化を遅らせることで実用上の訓練を大幅に安定させるが、一般的な収束保証（convergence guarantee）を回復させるものではない。

15.4.2 経験再生とターゲットネットワーク

DQN はこれら 2 つの結合に対し、それぞれ 1 つずつの仕組みで対処する。Figure 15.9 に両者の組み合わせを示す。

15.4.2.1 経験再生

相関したデータへの処方箋は**経験再生バッファ**（experience replay buffer）である (Lin 1992)。すべての遷移を大きな配列に保存し、訓練用の各ミニバッチをその配列から一様に抽出する。するとミニバッチには異なる時刻に収集された遷移が混ざり合い、各遷移が多くの更新で再利用される。(s, a, r, s') に対するターゲットが a を選んだ方策に依存しないため、Q 学習ではこの再利用が許される。一方で経験再生には欠点もある。保存された遷移の大半は古い方策によって収集

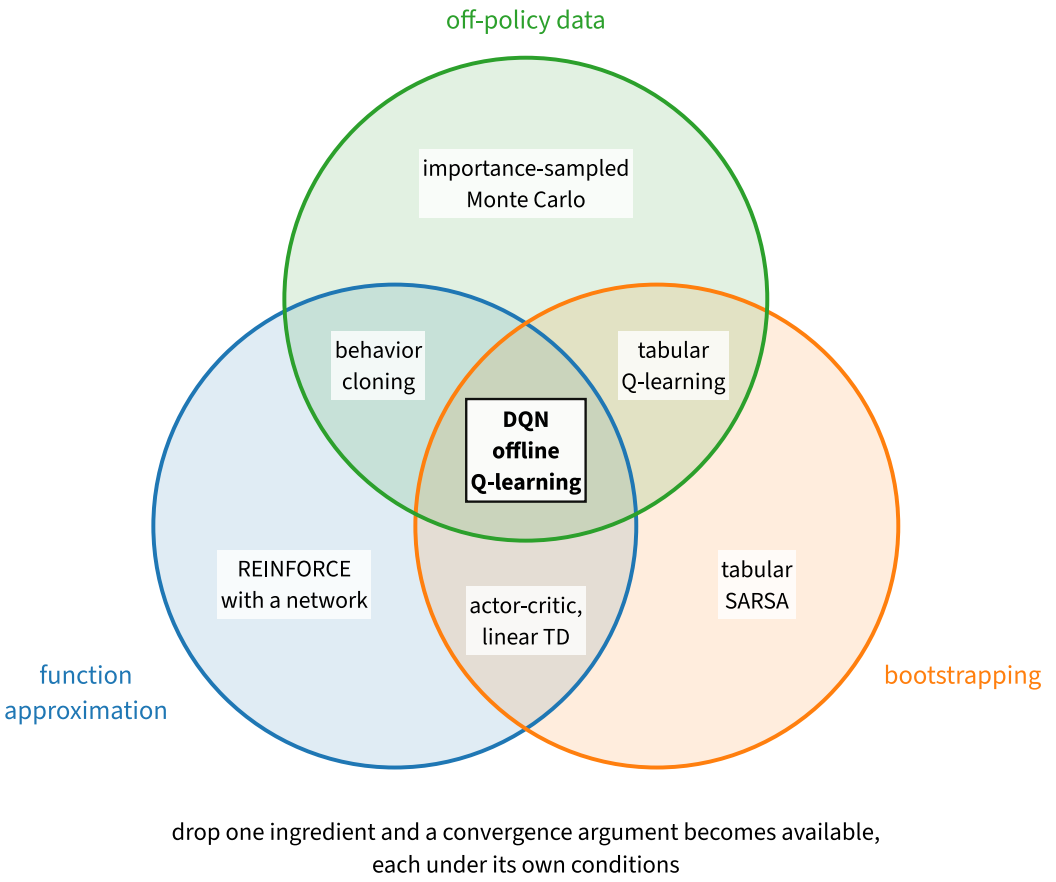


Figure 15.8: 死の三要素。各円が 1 つの要素を表し、すべての領域にこれら 2 つの章で扱ったアルゴリズムが入る。3 つの要素すべてが重なる中央に DQN とオフライン Q 学習が位置し、Baird の反例はそこで発散が起こり得ることを示す。いずれか 1 つの要素を取り除けば、それぞれのステップサイズ、カバレッジ、表現の条件のもとで収束の議論が成り立つようになる。死の三要素は警戒すべきパターンであり、必要十分条件の定理ではない。

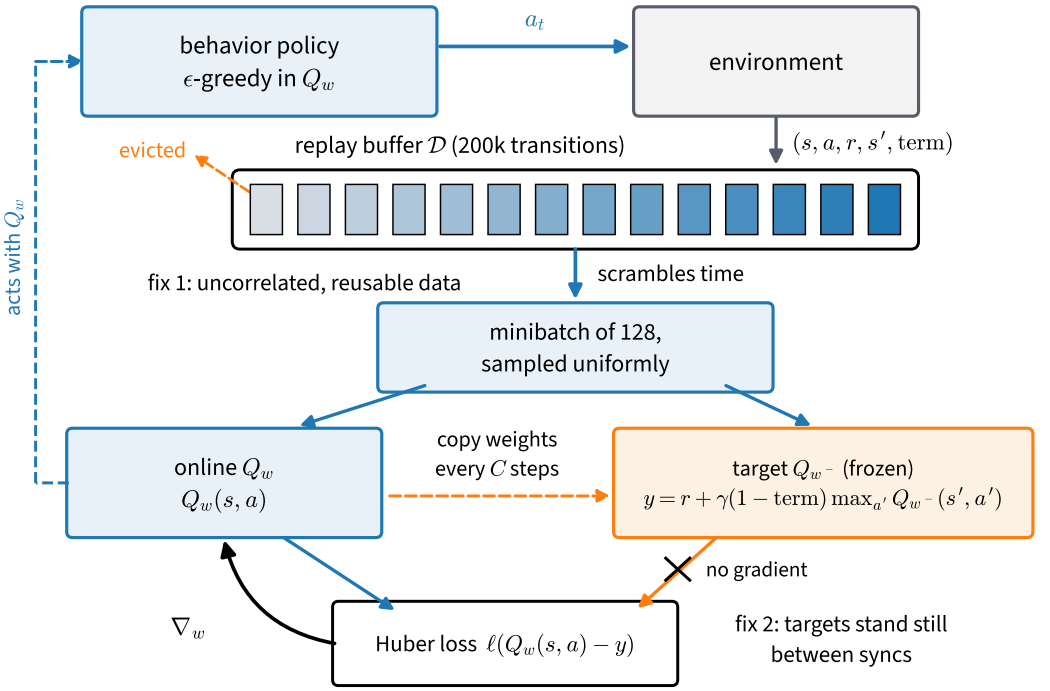


Figure 15.9: データフローとして見た深層 Q 学習。行動方策はオンラインネットワークを用いて行動し、遷移はリプレイバッファに入る。一様にサンプリングされたミニバッチが時間順をかき混ぜることで、データはほぼ独立かつ再利用可能になる。回帰目標は C ステップごとに更新されるネットワークの凍結コピーによって計算されるため、同期と同期の間、ターゲットは静止する。勾配はただ 1 つの経路を通ってオンラインネットワークへ流れ、凍結コピーからの印付きの辺には勾配が流れない。保存された各遷移はフラグ term を持ち、ターゲット内のマスク $1 - \text{term}$ は真の終端でのみブートストラップを止める。図のバッファは満杯で最も古い要素を追い出している長期実行の定常状態を描いているが、本節の実験では 50,000 ステップの予算が 200,000 スロットのリングを満たすことはないため、追い出しは生じない。

されたものであるため、現在の行動方策が該当する状態を再び訪れるまで、ターゲット内の誤差が多くの更新にわたって残り続ける可能性がある。

`ReplayBuffer` は、遷移の 5 つのフィールド（状態、行動、報酬、次状態、および終端を記録するフラグ `term`）を事前確保した配列に保持する。`add` は次の空きスロットに遷移を書き込み（配列が満杯になると最も古いものを上書きする）、`sample` は `batch_size` 個のスロットを一様に抽出して `Batch`（14.7 節）として返す。

```
class ReplayBuffer:
    """A ring of transitions in preallocated numpy; sample() returns a Batch."""
    def __init__(self, capacity, obs_dim):
        self.obs = np.zeros((capacity, obs_dim), np.float32)
        self.act = np.zeros(capacity, np.int64)
        self.rew = np.zeros(capacity, np.float32)
        self.next_obs = np.zeros((capacity, obs_dim), np.float32)
        self.term = np.zeros(capacity, np.float32)
        self.capacity, self.size, self.ptr = capacity, 0, 0

    def add(self, obs, act, rew, next_obs, term):
        i = self.ptr
        self.obs[i], self.act[i], self.rew[i] = obs, act, rew
        self.next_obs[i], self.term[i] = next_obs, term
        self.ptr, self.size = (i + 1) % self.capacity, min(self.size + 1,
                                                           self.capacity)

    def __len__(self):
        return self.size

    def sample(self, batch_size, rng):
        """batch_size transitions drawn uniformly, each a one-step
        fragment: ep_ends marks every transition as its own episode."""
        i = rng.integers(self.size, size=batch_size)
        return d2l.Batch(self.obs[i], self.act[i], self.rew[i],
                         self.next_obs[i], self.term[i],
                         np.arange(1, batch_size + 1))
```

本節での容量は 200,000 遷移であり、50,000 ステップの予算より大きいため、遷移の追い出しは生じない（より小さなバッファについては演習問題 4 で調べる）。一様抽出にはエピソードの順序がないため、`sample` はすべての遷移を長さ 1 のエピソード断片として扱う。このようなバッチ上では `reward_to_go` は報酬そのものになり、また 14.9 節の `truncation_bootstrap` は $\gamma(1-\text{term})\,\hat{V}(s')$（与えられた価値推定 $\hat{V}$ に対する項）を加えるため、両者の和は 15.1 節の `td_target` が計算する 1 ステップターゲットになる。経験再生では 1 ステップターゲットしか残らないため、1 ステップのブートストラップと組み合わせて用いられる。n ステップターゲットには連続した遷移が必要になるため、バッファ側で順序を保持しなければならない（演習問題 6）。

15.4.2.2 ターゲットネットワーク

移動するターゲットへの処方箋は**ターゲットネットワーク**である (Mnih et al. 2015)。オンラインネットワークの凍結コピー Q_{w^-} がすべてのブートストラップを次式のように計算する。

$$y = r + \gamma\,(1 - \mathbf{1}(s' \text{ terminal}))\max_{a'} Q_{w^-}(s', a'), \tag{15.35}$$

そしてこのコピーは、$w^- \leftarrow w$ として C 環境ステップごと（ここでは 250 ステップごと）にオンライン重みと同期（synchronization）される。同期と同期の間はターゲットが静止するため、オンラインネットワークは通常の回帰問題を解くことになる。予測がターゲットへフィードバックされるのは、勾配ステップごとではなく同期ごとに 1 回だけになる。15.1 節のクリティックは各バッチが新鮮だったため最新の重みからブートストラップしていたが、経験再生はその新鮮さを失わせるため、DQN は変化の緩やかな回帰問題と引き換えに数百ステップ古いターゲットを受け入れる。

さらに 2 つの制御によって 1 ステップの大きさを抑える。フーバー損失（Huber loss）は残差（residual）が 1 未満では二次、それ以上では線形であるため、予測に関する導関数（derivative）の大きさは決して 1 を超えない。ただし全勾配（full gradient）には $\nabla_w Q_w(s,a)$ も含まれ、これはフーバー損失では抑えられないため、更新時にはパラメータ勾配のノルムを 10 でクリップする（元論文ではなく Stable-Baselines3 のデフォルト値である (Raffin et al. 2021)）。コードでは、`q_values` が勾配を記録せずに NumPy の状態上でネットワークを順伝播させ、`fit_q` が $Q_w(s,a)$ を固定されたターゲット y に近づけるクリップ付きフーバー回帰ステップを 1 回実行し、`q_step` が DQN の更新を担う。そのターゲットは、凍結コピーのブートストラップ $\max_{a'} Q_{w^-}(s',a')$ を用いた `batch.td_target` である。

```
def fit_q(qnet, opt, obs, act, y):
    """One Huber regression step of Q_w(s, a) toward the fixed y; traced
    inside the jitted steps, with the norm clip in the optimizer chain."""
    _, grads = nnx.value_and_grad(lambda q: optax.huber_loss(
        jnp.take_along_axis(q(obs), act[:, None], -1).squeeze(-1),
        y).mean())(qnet)
    opt.update(qnet, grads)

@nnx.jit
def q_step(qnet, target, opt, obs, act, rew, next_obs, term):
    """The DQN update: the critic's own td_target, bootstrapped by the copy."""
    batch = d2l.Batch(obs, act, rew, next_obs, term, None)
    fit_q(qnet, opt, obs, act,
          batch.td_target(lambda s: target(s).max(-1), gamma))

q_values = d2l.forward_np   # the acting forward: numpy in, numpy out
```

`q_values` の実体は `d2l.forward_np` であり、ネットワーク構造ごとに 1 回の順伝播をコンパイルする（オンライン付録「ユーティリティ関数とクラス」）。ノルムによるクリッピングはオプティマイザのチェーンに組み込まれており、`q_step` は内部に `td_target` の計算を含めてコンパイルされる。コンパイルされた関数は `d2l.Batch` ではなく配列を受け取るため、`q_step` はバッチの各列を受け取り、内部で `Batch` を再構築して `td_target` の呼び出しにのみ用いる。

15.4.2.3 訓練ループ

フレームワーク間で異なる学習器の構成要素は `make_learner` から得られる。具体的には、凍結コピー、オプティマイザ、`update`（サンプリングされたバッチ上で更新則 `step` を 1 ステップ実行する）、および `sync`（オンライン重みを凍結コピーへコピーする）である。

```
def make_learner(qnet, step):
    """DQN's moving parts: the frozen copy, Adam with the norm clip, the
    update and the sync; the jitted step is bound to them once per run."""
    target = nnx.clone(qnet)
```

```
    opt = nnx.Optimizer(qnet, optax.chain(
        optax.clip_by_global_norm(grad_clip), optax.adam(lr)), wrt=nnx.Param)
    step_fn = nnx.cached_partial(step, qnet, target, opt)
    update = lambda b: step_fn(*map(jnp.asarray, (b.obs, b.act, b.rew,
                                                  b.next_obs, b.term)))
    sync = lambda: nnx.update(target, nnx.state(qnet, nnx.Param))
    return update, sync
```

nnx.cached_partial はネットワーク、そのコピー、およびオプティマイザをコンパイル済みステップへ一度だけ束縛するため、ループは呼び出しのたびにモジュールを走査せずに済む。

このループは 14.3 節の epsilon_greedy を用いて ϵ 貪欲に行動し（確率 ϵ でランダムな行動をとり、それ以外は argmax をとる）、すべての遷移を保存し、1,000 ステップのウォームアップ後に 2 ステップごと、サンプリングされたミニバッチ上で update を呼び出す。1 つのスイッチで本節の実験を切り替える。use_target=True では、コピーは sync_every ステップごとに更新される。use_target=False では毎ステップ後に更新されるため、ターゲットはオンラインネットワーク自身によって計算されることになり、ターゲットネットワークなしの素朴なレシピになる。引数 step は更新則を選択し、copies は各行動の複製数を選択する。各エピソードの終わりに、ジェネレータはステップ数、そのエピソードの収益、およびネットワークによる推定値 $\max_a Q_w(s_0, a)$（中心化された開始状態 $s_0 = \mathbf{0}$ における値）を返す。

```
def train_dqn(seed, qnet, use_target=True, step=q_step, copies=1):
    """DQN on CartPole; yields (env step, episode return, max_a Q(s0, a))."""
    rng, env = np.random.default_rng(seed), make_env(copies)
    update, sync = make_learner(qnet, step)
    buffer, s0 = ReplayBuffer(buffer_size, 4), np.zeros(4, np.float32)
    obs, ep_return = env.reset(seed=seed)[0], 0.0
    for t in range(1, num_env_steps + 1):
        a = d2l.epsilon_greedy(q_values(qnet, obs), epsilon(t), rng)
        next_obs, rew, terminated, truncated, _ = env.step(a)
        buffer.add(obs, a, rew, next_obs, float(terminated))
        obs, ep_return = next_obs, ep_return + rew
        if terminated or truncated:
            yield t, ep_return, q_values(qnet, s0).max()
            obs, ep_return = env.reset()[0], 0.0
        if len(buffer) >= warmup and t % train_freq == 0:
            update(buffer.sample(batch_size, rng))
        if t % (sync_every if use_target else 1) == 0:
            sync()
```

バッファが保存するのは terminated であり、truncated は決して保存しない。500 ステップで打ち切られた CartPole のエピソードは将来のある状態で終わっており、そこでブートストラップをマスクすると生き延びることに価値がないとネットワークに学習させてしまうことは 14.3 節で示したとおりである。誤ってマスクした場合に何が起きるかは演習問題 2 でたどる。

15.4.2.4 ターゲットネットワークのアブレーション

凍結コピーは本当に必要なのだろうか。そのスイッチ以外は同一の条件で、各アームにつき 3 つのシードを訓練する。シード間で事象がどの程度の頻度で起きるかに関する以下の記述は、各アーム 20 シードで確認したものである。訓練後のエージェントを後のセルで評価できるよう、ネットワークは訓練ループの外で構築する。train_checked は train_dqn をラップする関数である。エピソードごとの記録を収集するとともに、5,000 環境ステップの倍数を超えた最初のエピソード

終了時、および予算終了時にもう 1 回、別コピーの環境上で常に $\operatorname{argmax}_a Q_w(s,a)$ をとる**貪欲方策**（greedy policy）を 10 エピソード実行する。`d2l.evaluate` は複数エピソードにわたる方策の収益を平均するモンテカルロ評価（14.2 節）であり、`greedy` は Q ネットワークをそのような方策へ変換する。これらのチェックは訓練自体には影響しない。まずは価値推定の挙動を理解したうえで、その結果を読むことにする。

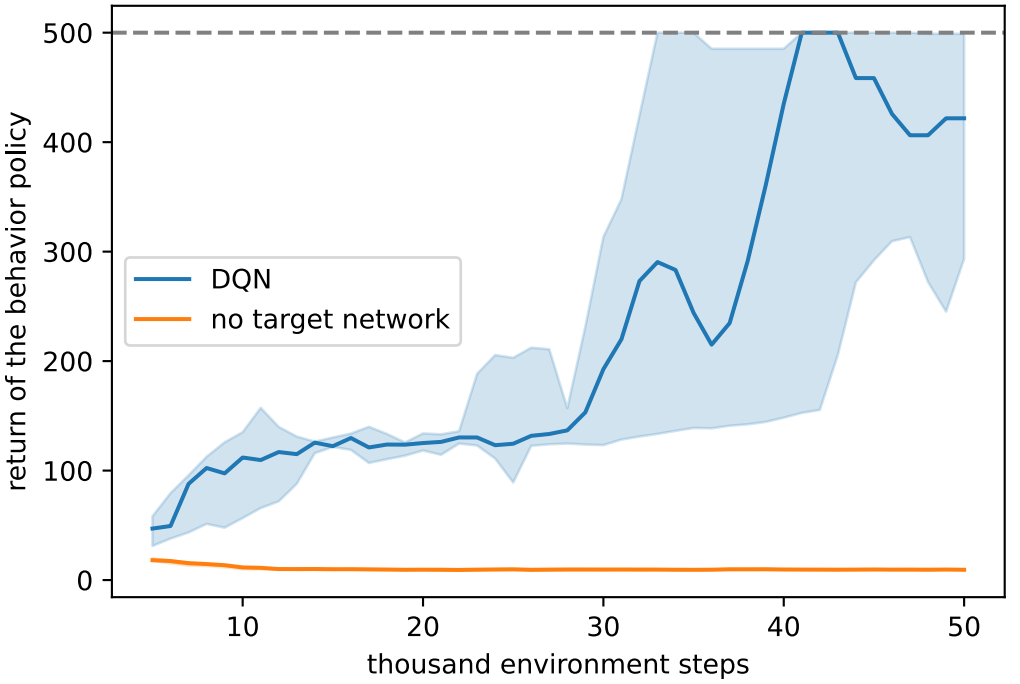
```
def greedy(qnet):
    """The greedy policy of a Q-network, (obs, rng) -> argmax_a Q(obs, a)."""
    return lambda s, rng: int(q_values(qnet, s).argmax())

def train_checked(seed, qnet, every=5_000, episodes=10, **kwargs):
    """train_dqn, plus a greedy evaluation over a few episodes at the first
    episode end past every multiple of `every` steps, and one at the end."""
    env, run, checks = make_env(kwargs.get('copies', 1)), [], []
    env.reset(seed=10_000 + seed)
    for t, ret, v0 in train_dqn(seed, qnet, **kwargs):
        run.append((t, ret, v0))
        if t >= every * (len(checks) + 1):
            checks.append(d2l.evaluate(env, greedy(qnet), episodes))
    checks.append(d2l.evaluate(env, greedy(qnet), episodes))
    return run, checks
```

`train_checked` を用いて、`arms = ('DQN', 'no target network')` の各アームにつき 3 つのシードを訓練し、エピソードごとの訓練記録を `runs` に、定期的な貪欲評価のスコアを `checks` に保存する。

```
arms = ('DQN', 'no target network')
qnets = {arm: [make_qnet(seed) for seed in range(num_seeds)] for arm in arms}
runs, checks = {}, {}
for arm in arms:
    out = [train_checked(seed, qnets[arm][seed], use_target=arm == 'DQN')
           for seed in range(num_seeds)]
    runs[arm], checks[arm] = [r for r, _ in out], [c for _, c in out]
```

シードごとに完了するエピソード数は異なるため、予算の単位である環境ステップ数を横軸にとってプロットする。`d2l.on_grid` は各実行のエピソードごとの記録を環境ステップの共通グリッド上に再サンプリングし、`d2l.plot_curves` はシード間の中央値（median）と最小シードから最大シードまでの帯を描画する。

```
grid = np.arange(1, num_env_steps // 1000 + 1) * 1000
d2l.plot_curves({arm: d2l.on_grid(runs[arm], grid, 1) for arm in runs},
                x=grid / 1000, xlabel='thousand environment steps',
                ylabel='return of the behavior policy', smooth=5,
                reference=500)
```

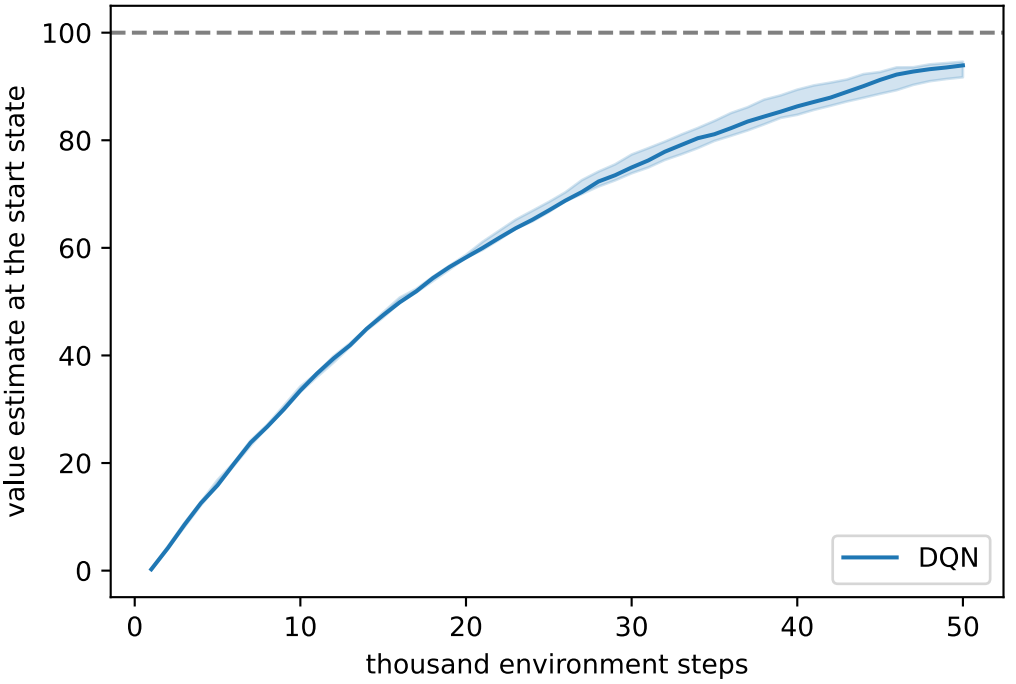


ターゲットネットワークありの場合、曲線はギザギザしたままであるものの、すべてのシードでランダム方策の収益である約 20 から数百の収益へと改善する。最終的な探索率（exploration rate）$\epsilon = 0.05$ のもとでも 20 ステップに 1 回はランダムな行動が強制されるため、行動方策（behavior policy）の収益は方策本来の性能を控えめに表している。ターゲットネットワークなしの場合、収益はランダム方策の水準を決して超えず、やがて約 9（毎ステップ同じ向きにカートを押し続けたときのエピソード長）まで落ち込む。

価値推定を見れば、何が起きたかがわかる。訓練ループは中心化された開始状態における $\max_a Q_w(s_0, a)$ を記録しており、その値がどこに落ち着くべきかはあらかじめ計算できる。バッファは `terminated` のときだけブートストラップをマスクするため、更新は 500 ステップの上限をまたいでブートストラップを続け、かつ状態には時計が含まれない。したがってネットワークは、時間上限がデータ収集だけを止めて過程自体は止めない継続タスク（continuing task）の価値を学習する。ポールを永久に倒さず保ち続ける方策は開始時点から $\sum_t \gamma^t = 1/(1-\gamma) = 100$ を獲得し、毎ステップの報酬が 1 である以上、いかなる方策もこれ以上は獲得できない。破線はこの上限を表す。

```
ceiling = 1 / (1 - gamma)
d2l.plot_curves({'DQN': d2l.on_grid(runs['DQN'], grid, 2)},
                x=grid / 1000, xlabel='thousand environment steps',
                ylabel='value estimate at the start state',
                reference=ceiling)
print(f'continuing-task ceiling: {ceiling:.0f}; the no-target arm ends at ' +
      ', '.join(f'{np.asarray(r)[-1, 2]:.0e}' for r in
                runs['no target network']))
```

```
continuing-task ceiling: 100; the no-target arm ends at 1e+08, 2e+08, 1e+08
```



DQN の推定値は上限に向かって上昇していくのに対し、ターゲットネットワークなしの推定値は 10^8 を超えて突き抜ける。暴走が任意の方向ではなく上向きに起きるのは、CartPole が毎ステップ 1 の報酬を与えるからである。エピソードを終了させないあらゆる遷移のターゲットは $1 + \gamma \max_{a'} Q_w(s', a')$、すなわちネットワーク自身の割引出力より 1 だけ大きい値になり、さらに $Q_w(s, a)$ をそこへ向かって押し上げる共有パラメータは $Q_w(s', \cdot)$ も一緒に押し上げる（s' が s に近いためである）。毎ステップでターゲットが予測の先を走り続け、この上昇を止めるものが何もない。Figure 15.10 はこのサイクルと、凍結コピーが断ち切る 1 本の辺を示している。

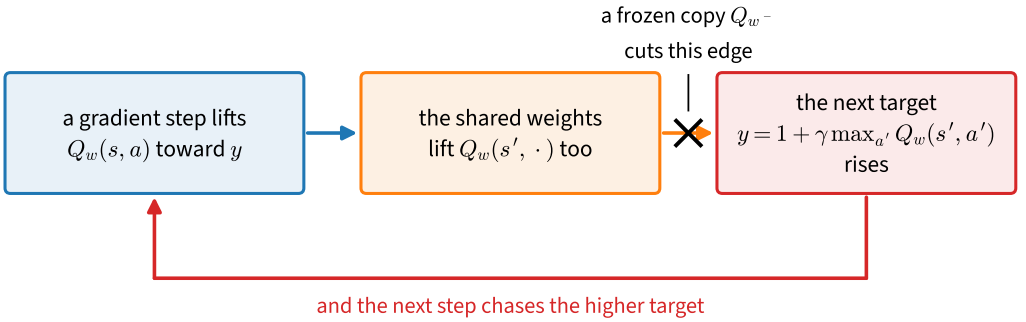


Figure 15.10: CartPole におけるターゲットネットワークなしの Q 学習。勾配ステップが $Q_w(s, a)$ をそのターゲット y に向かって押し上げると、共有重みによって $Q_w(s', \cdot)$ も押し上げられる。次のターゲットは $Q_w(s', \cdot)$ から計算されるためそれに伴って上昇し、次のステップはそのさらに高くなったターゲットを追いかける。代わりに凍結コピー Q_{w^-} がターゲットを計算することで、次の同期まで印付きの辺が断ち切られる。

凍結コピーは 125 勾配ステップの間ターゲットを静止させるため、価値は同期 1 回につき 1 段階

ずつしか上昇できない。すべての価値が 0 から始まり、オンラインネットワークが 2 回の同期の間でターゲットに厳密に適合すると仮定しよう。このとき、n 回の同期の後にいかなる価値も次の値を超えることはない。

$$\sum_{k=0}^{n-1} \gamma^k = \frac{1-\gamma^n}{1-\gamma}, \tag{15.36}$$

これは報酬 1 が n ステップ続くときの割引収益である。最初の勾配ステップの後、予算内には約 196 回の同期が含まれるため、開始状態の価値は約 86 で上から抑えられる。プロットされた推定値はこの上界によく沿って推移しつつ、約 10,000 ステップ以降は数ポイント上回り、すべてのシードで 86 を超えて終わる。これはネットワークが汎化（generalization）するためである。ある状態の価値を押し上げるステップはその近傍の価値も押し上げるため、同期 1 回につき 1 バックアップよりも速く価値が伝播する（演習問題 5）。

ターゲットネットワークはこのフィードバックに対する修復法の 1 つであり、唯一の方法ではない。PQN (Gallici et al. 2025) は凍結コピーとリプレイバッファの両方を廃止し、すべての隠れ層の活性化値（activation）を正規化したうえで、多数の環境から並列にデータを収集する。演習問題 8 では CartPole 上でこの正規化を試す。

15.4.2.5 貪欲方策の評価

訓練中の収益曲線は、実行結果の要約としては不十分である。探索的な行動が含まれているうえ、最終値は実行がたまたまいつ止まったかに左右され、さらに曲線全体を見た後で最良値を選ぶのはどの手法をも実際以上に良く見せてしまう。そこで、あらかじめ固定した予算ごとに貪欲方策を評価する。これが `train_checked` に記録しておいたチェック、すなわち 5,000 環境ステップごとおよび予算終了時点における 10 エピソードの貪欲収益である。

```
print('greedy return (10 episodes) every 5,000 environment steps')
for arm in arms:
    for seed, c in enumerate(checks[arm]):
        print(f'{arm:>17}, seed {seed}: ' + ' '.join(f'{x:3.0f}' for x in c))
```

```
greedy return (10 episodes) every 5,000 environment steps
              DQN, seed 0: 244 128 125 120 129 124 139 196 200 500
              DQN, seed 1: 154  78 121 118  95 500 500 500 500 500
              DQN, seed 2: 229 162 113 300 150 500 118 500 500 500
no target network, seed 0:   9   9  10   9   9   9   9   9   9  10
no target network, seed 1:   9   9  10   9   9   9  10   9   9   9
no target network, seed 2:   9   9  10  10   9   9   9   9   9   9
```

このように確認すると、DQN の貪欲方策は最初の 30,000 ステップではおおむね約 50〜250 の収益にとどまり（まれに早い段階で 500 に達することもある）、その後 35,000〜45,000 ステップの間で大半のシードが 500 に到達する。一度良い結果が出てもそれで確定するわけではない。出力結果の 1 シードがそうであるように、後から数百ポイント後退するシードもあり、20 シード中約半数でそうした低下が見られた。ターゲットネットワークなしでは、どのチェック時点でも貪欲方策が約 10 ステップを超えて持続することはなく、訓練を長く続けても救われない。また、DQN は保存された各遷移で何度も訓練するにもかかわらず、エピソード終了までポールを倒さず保つのに、バッチあたり 4 エポックの PPO が 15.2 節で要したよりも多くの環境ステップを必要とする。

貪欲収益が示すのは方策がどれだけ優れているかであり、ネットワークがその良さをどれだけ正確に把握しているかではない。Van Hasselt et al. (2016) は、価値推定を貪欲方策が実際に獲得する割引収益（まさにその推定値が予測しようとしている量）と照らし合わせて検証している。`audit` は訓練済みネットワークに対して 3 つの数値を計算する。100 エピソードにわたる貪欲収益、20 エピソードにわたる貪欲方策の割引収益（`d2l.evaluate` に `gamma` を指定）、および推定値 $\max_a Q_w(s_0, a)$ である。エピソードは中心化された状態の近くから始まるため、この割引収益は s_0 の価値を測るものになる。500 ステップの上限のもとでは、完璧な方策であっても獲得できる割引収益は $(1-\gamma^{500})/(1-\gamma) \approx 99.3$ にとどまり、継続タスクの上限よりわずかに小さい。

```
def audit(qnet, seed, copies=1):
    """Greedy return over 100 episodes, its discounted return over 20, and
    the network's own estimate max_a Q(s0, a) of the latter."""
    env = make_env(copies)
    env.reset(seed=100 + seed)
    score = d2l.evaluate(env, greedy(qnet), 100)
    env.reset(seed=100 + seed)
    disc = d2l.evaluate(env, greedy(qnet), 20, gamma=gamma)
    return score, disc, q_values(qnet, np.zeros(4, np.float32)).max()

audits = {arm: [audit(qnets[arm][seed], seed) for seed in range(num_seeds)]
          for arm in arms}
for arm in arms:
    for seed, (score, disc, v0) in enumerate(audits[arm]):
        print(f'{arm:>17}, seed {seed}: return {score:3.0f}, discounted '
              f'{disc:5.1f}, max_a Q(s0) {v0:8.3g}')
```

```
            DQN, seed 0: return 500, discounted  99.3, max_a Q(s0)     94.6
            DQN, seed 1: return 500, discounted  99.3, max_a Q(s0)     93.9
            DQN, seed 2: return 500, discounted  99.3, max_a Q(s0)     91.7
no target network, seed 0: return  10, discounted   9.0, max_a Q(s0) 1.24e+08
no target network, seed 1: return   9, discounted   9.0, max_a Q(s0) 1.53e+08
no target network, seed 2: return   9, discounted   8.9, max_a Q(s0) 1.39e+08
```

貪欲方策がポールを倒さず保てる場合、推定値は方策が獲得する割引収益から約 8 ポイント以内に収まり、たいていはそれを下回る。予算終了時点でも価値がまだ上昇途中であり、上の開始状態の価値プロットが頭打ちになっていないからである。一方、貪欲方策が失敗する場合（5 シード中約 2 シードで発生）には、推定値は方策が獲得する収益を上回り、そのような実行の約半数で 20 ポイント以上も高くなる。行動が 2 つしかない設定では、ネットワークが実際の方策の達成値以上の価値を約束してしまう兆候はこうした失敗時の実行にしか現れず、件数も少ないうえ他の誤差と混ざり合っているため効果を測りにくい。より鮮明な検証には、さらに多くの行動にわたる最大値が必要になる。

15.4.3 過大評価

15.4.3.1 数値で見る最大化バイアス

ターゲット Equation 15.35 は推定された行動価値の最大値をとる。ノイズを含む推定値に対しては $\mathbb{E}[\max_{a'} \hat{Q}(s', a')] \geq \max_{a'} \mathbb{E}[\hat{Q}(s', a')]$ が成り立ち、どの行動が最大になるかをノイズが変え得るときには狭義の不等号になる。14.3 節ではテーブル上でこの最大化バイアス（maximization bias）を実演し、15.3 節ではハード最大値が一様な参照方策を持つソフトバックアップの $\beta \to 0$ 極限であることを示した。関数近似を用いる場合、ある状態でのバイアスはそこへ至る状態群のターゲットへと流れ込む (Thrun and Schwartz 1993)。

次のシミュレーションでは、4 つの行動すべてに同じ真の価値 0 を与え、各推定値に独立な単位分散ノイズを加える。単一推定量（single estimator）は同じノイズ付き推定値を用いて行動の選択と評価の両方を行う。二重推定量（double estimator）は 1 組目のノイズ付き推定値で選択し、独立な 2 組目の推定値で評価する (Hasselt 2010)。

```
rng = np.random.default_rng(0)
est = rng.standard_normal((100_000, 4))     # four values, all truly zero
second = rng.standard_normal((100_000, 4))  # an independent second estimate
sel = est.argmax(1)
print(f'single estimator, E[max of the estimates]: {est.max(1).mean():.3f}')
print(f'select with one, evaluate with the other: '
      f'{second[np.arange(100_000), sel].mean():+.3f}')
```

```
single estimator, E[max of the estimates]: 1.031
select with one, evaluate with the other: -0.006
```

単位分散ノイズを持つ 4 つの推定値の最大値には、約 1 単位の上向きバイアスが生じる。選択と評価に独立な推定値を用いれば、どの行動が選ばれたとしても評価側のノイズの平均は 0 であるため、測定されるバイアスは 0 付近になる。Figure 15.11 に示すように、単一推定量のバイアスは行動数とともに増大するのに対し、独立な二重推定量のバイアスは 0 のままである。10^5 個の行動の中から次のトークンを選ぶ言語モデルでは、ノイズを含む推定値にわたる最大値から生じるバイアスははるかに大きくなり、これが言語モデルの事後学習で Q ネットワークではなく方策勾配が用いられる理由の 1 つとなっている。固定されたデータセットでは、新たな相互作用によって過大評価された行動が修正されないため、このバイアスはさらに深刻になる（15.7 節）。

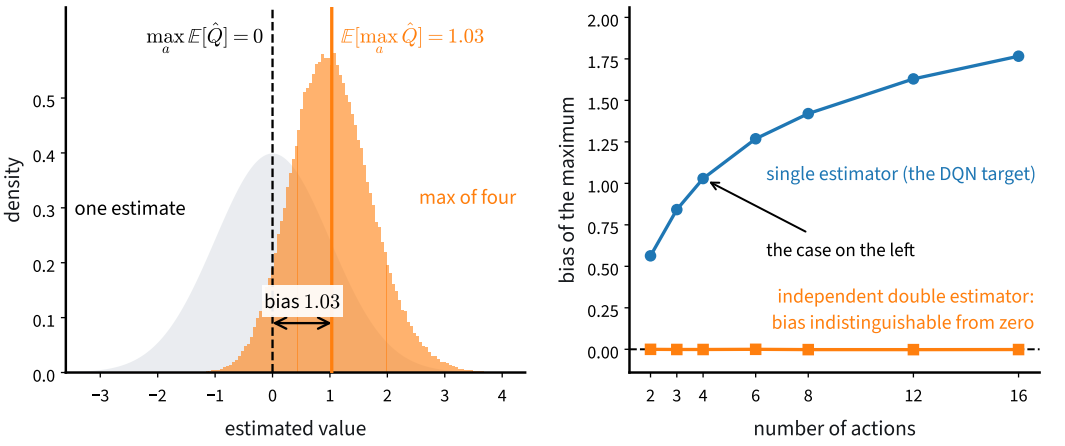


Figure 15.11: ノイズを含む推定値にわたる最大値は上向きにバイアスを持つ。左：同じ真の価値 0 に対する 4 つの独立な推定値の最大値の分布と、単一の推定値の分布の比較。最大値は測定値で 1.03 のバイアスを持つ。右：単一推定量のバイアスは行動数とともに増大する一方、1 組の推定値で行動を選択し独立な別の組で評価する独立な二重推定量のバイアスは、全体を通じて 0 と区別できない。

15.4.3.2 Double DQN

二重推定量には 2 つの価値推定が必要だが、DQN はすでに 2 つのネットワークを持っている。Double DQN (Van Hasselt et al. 2016) はオンラインネットワークで行動を選択し、ターゲット

ネットワークでその行動を評価する。

$$y = r + \gamma\,(1 - \mathbf{1}(s' \text{ terminal}))\, Q_{w^-}\big(s',\ \operatorname*{argmax}_{a'} Q_w(s', a')\big). \tag{15.37}$$

これにより、オンラインネットワークが偶然過大評価した行動は別のネットワークによって採点される。この変更に新たなネットワークは不要であり、s' に対するオンラインネットワークの順伝播が 1 回増えるだけで済む。ただし、コピーは高々 125 勾配ステップ前のオンラインネットワークであるため、2 つのネットワークは独立ではなく、Double DQN が取り除けるのはバイアスの一部である。評価側の推定値が選択側の推定値と相関係数 ρ で相関している場合、残るバイアスは単一推定量の ρ 倍になる（演習問題 3）。`q_step` への変更は 3 行だけである。

```
@nnx.jit
def q_step_double(qnet, target, opt, obs, act, rew, next_obs, term):
    """q_step with selection split from evaluation, eq_double_dqn."""
    sel = qnet(next_obs).argmax(-1)
    batch = d2l.Batch(obs, act, rew, next_obs, term, None)
    fit_q(qnet, opt, obs, act, batch.td_target(lambda s: jnp.take_along_axis(
        target(s), sel[:, None], -1).squeeze(-1), gamma))
```

15.4.3.3 複製された行動

本来の CartPole では最大値をとる選択肢がほとんどない。単位分散ノイズを持つ 2 つの独立な推定値の大きい方が持つバイアスは $1/\sqrt{\pi} \approx 0.56$ と 4 行動の場合の約半分であり、アブレーションにおける 2 行動の推定値は同期の遅れによって逆方向へ引っ張られている。タスク自体を変えずに訓練済みネットワーク上でこのバイアスを観察するため、CartPole の各行動を 9 回ずつ提示する。`make_env(copies=9)` は Atari の全行動セットと同じサイズである 18 個の行動を公開し、行動 a は行動 $a \bmod 2$ と同じ向きにカートを押す。また `make_qnet(seed, copies=9)` は 18 個の出力を持つ。複製を増やしても最適価値も上限の 100 も変わらない。18 個の出力は 2 つの価値それぞれに対する 9 個のノイズ付き推定値となり、ターゲット内の最大値は 9 個の中で最も運良く高くなったものを選ぶ。すなわち、Figure 15.11 の右パネルが実際のネットワーク上で再現されるのである。次のセルでは 18 行動の DQN と Double DQN、および 2 行動の Double DQN を訓練し、訓練済みのすべてのネットワークを監査する（2 行動の DQN はアブレーションのアームをそのまま用いる）。

```
arms2 = {'Double DQN, 2 actions': (q_step_double, 1),   # (update, copies)
         'DQN, 18 actions': (q_step, 9),
         'Double DQN, 18 actions': (q_step_double, 9)}
names = ('DQN, 2 actions', *arms2)
qnets[names[0]], runs[names[0]] = qnets['DQN'], runs['DQN']
audits[names[0]] = audits['DQN']
for name, (step, copies) in arms2.items():
    qnets[name] = [make_qnet(seed, copies) for seed in range(num_seeds)]
    runs[name] = [list(train_dqn(seed, qnets[name][seed], step=step,
                                 copies=copies)) for seed in range(num_seeds)]
    audits[name] = [audit(qnets[name][seed], seed, copies)
                    for seed in range(num_seeds)]
```

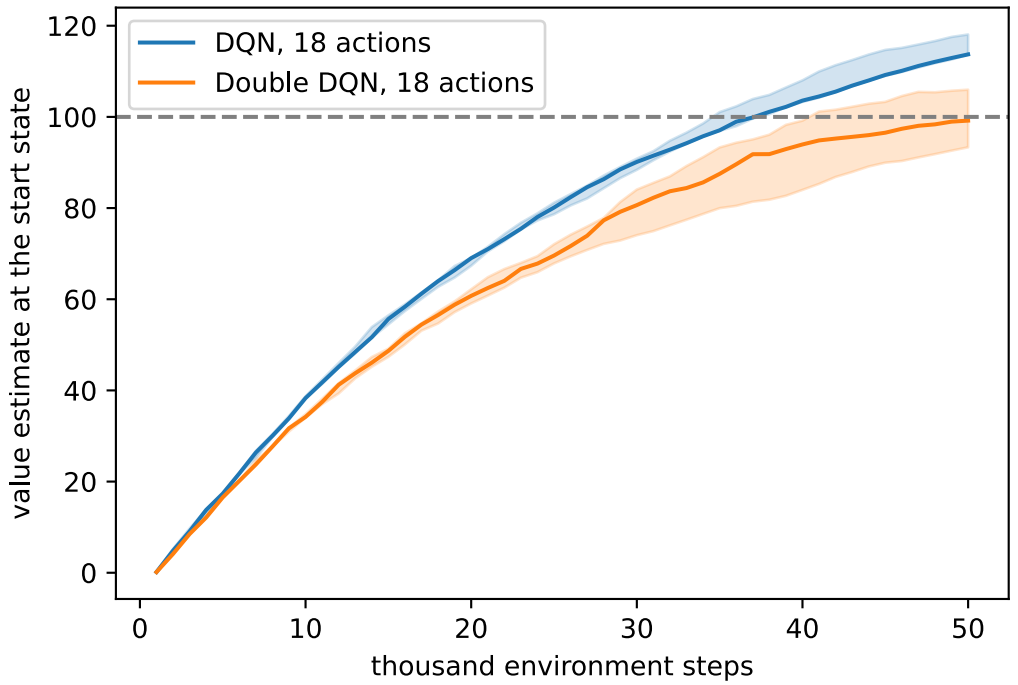
各アームについて、シードごとに貪欲収益、貪欲方策の割引収益、および開始状態での推定値を出力する。

```
print(f'{"per seed:":>22} {"greedy return":>17}   {"discounted":>17}'
      f'   {"max_a Q(s0)":>17}')
for name in names:
    res = np.array(audits[name]).T
    print(f'{name:>22} ' + '   '.join(' '.join(f'{x:5.1f}' for x in row)
                                       for row in res))
```

```
            per seed:     greedy return          discounted         max_a Q(s0)
       DQN, 2 actions 500.0 500.0 500.0    99.3  99.3  99.3    94.6  93.9  91.7
 Double DQN, 2 actions 500.0 500.0 491.5    99.3  99.3  99.3    90.3  90.4  96.6
      DQN, 18 actions 500.0 500.0 230.7    99.3  99.3  90.1   113.6 113.3 118.2
Double DQN, 18 actions 500.0 500.0  94.8    99.3  99.3  61.0    99.0  93.1 106.0
```

訓練を通じた 18 行動の 2 手法における開始状態の推定値と上限の比較を次にプロットする。

```
d2l.plot_curves({name: d2l.on_grid(runs[name], grid, 2) for name in names[2:]},
                x=grid / 1000, xlabel='thousand environment steps',
                ylabel='value estimate at the start state',
                reference=ceiling)
```



各行動を 9 回ずつ提示すると、DQN の開始状態の推定値はすべてのシードで上限を超えて終わり、典型的には上限より 10〜30 ポイント高くなる。貪欲方策がポールを倒さず保って約 99 を獲得しているシードでも、推定値はその収益を同程度超過する。このタスクで許されるいかなる価値も上限を超えることはないため、それ以上の評価を行わなくても過大評価が生じていることは確実である。プロットでは、DQN の推定値は 40,000 ステップより前に上限を突き抜けて上昇し続ける一方、Double DQN の推定値は上限付近かそれ以下にとどまる。Double DQN の推定値はほぼすべてのシードで 10〜25 ポイント低く、上限付近で終わる（ただし方策が失敗したシードで

は依然として上振れし得る)。20 シード実行した 18 行動 DQN のうち数本では、ブートストラップを通じてバイアスがフィードバックされ、50,000 ステップ時点でもまだ上昇が続いている。2 行動の場合には両者のターゲットは数ポイント以内で一致し、この予算では貪欲収益にもシード間のばらつきを超える差は生じない。CartPole において、二重推定量は貪欲方策がポールを保つ性能を変えることなく、価値推定をより正確にする。

これで、任意の方策から得られた遷移を何度も再生しながら、ターゲットを静止させる凍結コピーと最大値による水増しを抑える 2 つ目の推定値を用いて価値ネットワークを訓練できるようになった。ただし、これら 2 つの修復法はいずれも行動にわたる最大値が計算できることを前提としている。CartPole の 2 通りの押し方や Atari の 18 通りのジョイスティック操作は有限のリストをなしている。しかしモーターのトルクは区間内の連続値であり、その上での argmax はステップごとにそれ自体が 1 つの最適化問題になる。次節では経験再生と凍結コピーを保ったまま、最大値演算を最良の行動を提案するよう訓練された 2 つ目のネットワークに置き換え、Figure 14.23 のオフポリシーの列を完成させる。

15.4.4 まとめ

Q テーブルをネットワークに置き換えると、関数近似、ブートストラップ、オフポリシーデータの 3 つが組み合わさる。Baird の 7 状態の例は、真の価値が表現可能でサンプリングノイズが存在しない場合でさえ、この組み合わせが発散し得ることを示している。DQN はこれら 3 つの要素すべてを維持したまま、2 つの結合を修復する。リプレイバッファが連続する遷移の相関を断ち切って各遷移を何度も再利用し、凍結ターゲットネットワークが同期と同期の間で回帰目標を静止させる。CartPole では、凍結コピーを取り除くと開始状態の価値が 10^8 を超えて暴走するが、凍結コピーを用いれば価値は同期ごとに 1 段階ずつ上限の 100 に向かって上昇し、35,000〜45,000 環境ステップ後には大半のシードで貪欲方策が 500 ステップ最後までポールを保てるようになる。また、ターゲット内の最大値は価値を上向きにバイアスさせ、選択肢となる行動が多いほどそのバイアスは大きくなる。各行動を 9 回ずつ提示すると、DQN の推定値はすべてのシードで上限を超えて終わる。Double DQN はオンラインネットワークで行動を選択し凍結コピーでそれを評価することで、順伝播 1 回分の追加コストと引き換えにそのバイアスの大部分を取り除く。

15.4.5 演習問題

1. [code] **オンポリシー重み付けのもとでの Baird の反例**。15.4.1.3 節における Baird の反例の計算では、`.mean(0)` によって 7 つの状態を等しく重み付けしている。行列形式では、1 スイープは $\mathbf{w} \leftarrow \mathbf{w} + \alpha A \mathbf{w}$(ただし $\alpha = 0.01$)および

$$A = \Phi^\top D(\gamma P - I)\Phi, \tag{15.38}$$

と書ける。ここで D は状態重みの対角行列であり、P はすべての行が状態 7 に全確率を置く評価対象方策の遷移行列である。

   1. A の固有値を $D = I/7$ の場合に計算せよ。次に $1 + \alpha\lambda$(実部が最大の固有値 λ に対する値)を、$\mathbf{w}$ の上限ノルムが 20,000 スイープの実行終盤で示す 1 スイープあたりの成長率と比較せよ。
   2. 評価対象の方策は状態 7 へ移動してそこにとどまるため、その訪問分布は状態 7 にすべての重みを置く。この重み付けで反復を再実行せよ。どの重みおよびどの状態価値へ収束するか。また、それらの値は真の価値とどのように比較されるか。
   3. 2 つの重み付けを $\mathbf{d} = (1-p)\,\mathbf{1}/7 + p\,\mathbf{e}_7$ として混合し、最小の p(A のどの固有値も正の実部を持たなくなる値)を $\gamma \in \{0.9, 0.95, 0.99\}$ のそれぞれについて二分法で求めよ。反復が許容するオフポリシー重みの量は割引率にどのように依存するか。

2. **done ではなく terminated**。`train_dqn` は `float(terminated)` を保存するため、CartPole の 500 ステップの時間上限に至る遷移でもブートストラップは継続される。代わりに `float(terminated or truncated)` を保存する実装を考えよ。
    1. 貪欲方策がまだ数百ステップ以内に失敗している段階では、この変更は保存されたどの遷移に影響するか。また、すべてのエピソードが時間上限に到達するようになった後はどの遷移に影響するか。
    2. 観測には時計が含まれないため、ネットワークはステップ 499 の状態とステップ 50 のよく似たバランス状態を区別できない。バランス状態におけるネットワークの推定値を単一の数値 q、すなわち毎ステップ報酬 1 を与え H ステップ後に打ち切られる 1 状態タスクの価値としてモデル化せよ。誤ったマスクのもとでは、1 エピソードの遷移のうち $H-1$ 個（全 H 個中）がターゲット $1+\gamma q$ を持ち、最後の 1 個がターゲット 1 を持つ。平均化された更新の不動点が $q = H/\big(H-\gamma(H-1)\big)$ であることを示し、$H=500$、$\gamma=0.99$ における値を計算せよ。15.4.2.4 節の開始状態の価値プロットは $1/(1-\gamma)$ を上限として示している。q を、残り時間が既知である 500 ステップエピソードの開始状態の価値 $(1-\gamma^H)/(1-\gamma)$ と比較せよ。時計を持たないネットワークに対して、誤った実装の上限となるのはこれら 2 つのうちどちらか。最終的な推定値が 94 であったとしたら、それは何を示唆するか。

3. [code] **閉形式での最大化バイアス**。推定値 $X_1,\dots,X_m$（m 個の行動すべてが同じ真の価値 0 を持つ）が、それぞれ分散 σ^2 の同時正規分布に従うとする。
    1. $m=2$ かつ推定値が互いに独立であるとき、$\mathbb{E}[\max(X_1,X_2)] = \sigma/\sqrt{\pi}$ となることを、$\max(x,y) = (x+y)/2 + |x-y|/2$ を用いて示せ。
    2. 1 つのネットワークの複数の出力は隠れ層を共有するため、その誤差には相関が生じる。$m=2$ で相関係数が c（X_1 と X_2 の間）であるとき、バイアスが $\sigma\sqrt{(1-c)/\pi}$ になることを示せ。
    3. Double DQN では、オンラインネットワークが選択した行動をターゲットネットワークが評価するため、2 つのネットワークの誤差は相関する。行動 a の評価側の推定値を $Y_a = \rho X_a + \sqrt{1-\rho^2}\,Z_a$ とモデル化する。ここで $Z_1,\dots,Z_m$ はそれぞれ平均 0・分散 σ^2 で $X_1,\dots,X_m$ と独立であり、したがって Y_a は分散 σ^2 を持ち、相関係数 ρ で X_a と相関する。このとき、$Y_{\mathrm{argmax}_a X_a}$ のバイアスが $\rho\,\mathbb{E}[\max_a X_a]$ であることを示せ。
    4. (a) と (c) を、15.4.3.1 節のシミュレーションを変更して確認せよ。2 つの行動を用い、2 つ目の推定値を `rho * est + np.sqrt(1 - rho**2) * noise`（`noise` は新たに生成した標準正規分布に従う乱数）として、$\rho \in \{0, 0.5, 0.9\}$ について実行せよ。

4. [code] **経験再生の再利用回数とバッファ容量**。●バッファに `warmup` 個の遷移が貯まると、`train_dqn` は一様にサンプリングされた `batch_size` $=128$ 個の遷移上で、`train_freq` $=2$ 環境ステップごとに 1 回の勾配ステップを実行する。デフォルトの容量では、遷移の追い出しは生じない。
    1. ステップ $t_0 \geq 1000$ で保存された遷移が、実行全体を通じて期待値で約 $64\ln(50{,}000/t_0)$ 回サンプリングされることを示し、これを $t_0 \in \{1000, 25{,}000, 49{,}000\}$ に対して計算せよ。訓練のどの段階のデータが更新の大半を占めるか。
    2. 容量が 500 遷移のとき、ある遷移が満杯のバッファ内にとどまっている間に期待値で何回サンプリングされるか。これを (a) と比較せよ。
    3. `buffer_size` を 500 と 5000 に設定し、小さい方の容量では `warmup` を 500 に下げよ（`train_dqn` はバッファが `warmup` 個の遷移を保持するまで待機するためである）。各容量につき 1 シードを訓練し、収益曲線と開始状態の価値トレースをデフォルトの実行と比較せよ。悪影響は訓練のどの段階で現れるか。

4. 小さなバッファは、ミニバッチ内の遷移を相関させることと古い状態を捨てることの両方を引き起こす。これら 2 つの効果を切り分けるため、デフォルトの容量を保ったまま `ReplayBuffer.sample` を変更し、一様に抽出した開始インデックスから連続する `batch_size` 個の遷移を返すようにせよ。この実行を容量 500 での実行と比較せよ。(c) の悪影響を説明するのは 2 つの効果のうちどちらか。

5. [code] **同期周期**。●`train_dqn` は `sync_every` $= 250$ 環境ステップごと、すなわち 125 勾配ステップごとにオンライン重みをターゲットネットワークへコピーする。

   1. 15.4.2.4 節では、n 回の同期後のあらゆる価値を $(1-\gamma^n)/(1-\gamma)$ で上から抑えた。最初の勾配ステップに続く同期回数を数え、`sync_every` $\in \{10, 50, 250, 1000, 5000\}$ に対して予算終了時点でのこの上界を計算せよ。
   2. これら 5 つの値それぞれについて 2 シードずつ DQN アームを再実行し、予算終了後の貪欲評価と開始状態の価値トレースを記録せよ。スイープのどちらの端がターゲットネットワークなしの実行と同様に振る舞うか。また、(a) の上界が到達可能な値を制限するのはどちらの端か。
   3. この上界は、すべての価値が 0 から始まり、同期と同期の間でオンラインネットワークがターゲットに厳密に適合することを仮定している。`sync_every` $= 250$ とし、1 シードの最初の 40 回の同期それぞれの直前における $\max_a Q_w(s_0, a)$ を記録して上界とともにプロットせよ。実行結果が最初に上界を超えるのはどこか。また、その超過は 2 つの仮定のうちどちらに反しているか。

6. **経験再生からの n ステップターゲット**。●DQN の後継手法のいくつかは、1 ステップターゲット Equation 15.35 を n ステップターゲット、すなわち末尾に最大値を持つ n ステップ収益（15.1 節）に置き換える。

$$y_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \max_{a'} Q_{w^-}(s_{t+n}, a'), \tag{15.39}$$

   ここではエピソードがステップ $t+n$ を超えて続く場合について書いている。

   1. `ReplayBuffer.sample` は前後の切り離された遷移を返す。バッファが何を保存し、`sample` がどのようにインデックスを選べば、$y_t^{(n)}$ を構成できるようになるか。
   2. $y_t^{(n)}$ を、エピソードがステップ $t+j$（ただし $j < n$）で終端する場合について書き下せ。代わりにステップ $t+j$ で打ち切られる場合にはどのように変わるか。
   3. 行動 $a_{t+1}, \dots, a_{t+n-1}$ は、古いネットワークの ϵ 貪欲な行動方策によって選ばれたものである。$n = 2$ かつ $Q_{w^-} = Q^*$ のとき、$\mathbb{E}[y_t^{(2)} \mid s_t, a_t]$ を書き下し、それが Q^* の 1 ステップターゲットと一致するのは a_{t+1} が確率 1 で Q^* に関して貪欲である場合に限ることを示せ。n ステップターゲットはどの方策の価値を推定しているか。
   4. n ステップ収益（15.1 節）では (c) の問いは生じなかった。それはなぜか。

7. [code] **ターゲットネットワークを用いた Baird の反例**。●●DQN は重みの凍結コピーからブートストラップを計算する。15.4.1.3 節における Baird の反例の計算で、`delta` をコピー `w_target` から `gamma * (Phi @ w_target)[6] - Phi @ w` として計算し、さらに `w_target = w.copy()` を k スイープごとに実行するようにせよ。

   1. 100,000 スイープを $k \in \{1, 10, 30, 100, 300, 1000\}$ について実行し、$\mathbf{w}$ の上限ノルムと最大価値誤差 $\max_i |(\Phi \mathbf{w})_i|$ の両方をプロットせよ。どの k に対して反復は収束し、その速さは k にどう依存するか。価値が 0 に近づく場合でも $\mathbf{w}$ の上限ノルムが 0 に近づかないのはなぜか。

2. `Phi` の 7 つの行が線形独立であり、したがってターゲット価値の任意のベクトルがある **w** によって厳密に表現されることを示せ。ここから、2 回の同期の間の回帰が厳密に解かれるならば、正のあらゆる状態重み付けのもとで各同期が価値 **v** を $\gamma P\mathbf{v}$ へ写すことを導け。これを用いて (a) の両端の挙動を説明せよ。
3. Tsitsiklis と Van Roy の反例では、2 つの非終端状態が価値 w と $2w$（単一の重み w に対する値）を持ち、すべての報酬は 0 である。第 1 の状態は第 2 の状態へ遷移し、第 2 の状態は確率 $1-\xi$ でその場にとどまり、それ以外は終端する。両状態を等しく重み付けせよ。凍結された重み w_k のターゲットに対する厳密な最小二乗回帰が $w_{k+1} = \gamma(6-4\xi)\,w_k/5$ を与えること、そしてターゲットネットワークなしの半勾配反復が γ と ξ に関する同じ条件のもとで発散することを示せ。
4. どちらの例も三要素すべてを含んでいる。ターゲットネットワークで修復できる例と修復できない例を分けるのは、関数クラスのどの性質か。

Richard S. Sutton および Andrew G. Barto 著『Reinforcement Learning: An Introduction』Example 11.1 に基づく。

8. [extended] **ターゲットネットワークに代わる LayerNorm。●●**Gallici et al. (2025) は、十分に幅の広い LayerNorm 付き Q ネットワークを最終層の重みに対する ℓ_2 ペナルティとともに訓練すると、オフポリシーデータ上であってもターゲットネットワークなしの時間差分学習に対する安定性条件を満たすことを証明している。同論文の実験では、Baird の反例を除くすべての環境で LayerNorm 単独で十分であった。

   1. `make_qnet` の各隠れ線形層の直後（ReLU の直前）に LayerNorm を挿入し、ターゲットネットワークなしのアームを 3 シードで再実行せよ。開始状態の価値トレースと貪欲評価を、15.4.2.4 節の両アームと比較せよ。
   2. ターゲットネットワークを保持する DQN アームにも同じ層を追加せよ。そこでは LayerNorm によって何が変わるか。
   3. (a) で依然として発散するシードがある場合、最後の線形層の重みに ℓ_2 ペナルティを追加し、発散を防ぐ最小の係数を求めよ。係数を大きくすると、学習される価値はどうなるか。
   4. この定理は十分に幅の広いネットワークと正則化された更新を対象としている。今回の実験はその仮定のうちどれを満たしているか。(a)〜(c) の結果のうち、論文の主張と矛盾するのはどのような結果であり、単に定理の適用範囲外にとどまるのはどのような結果か。

15.5 ソフトアクタークリティック

14.9 節の最後では、方策ネットワークが振り子を振り上げて静止させることを学習したが、それには数十万もの環境ステップを要した。1,600 ステップからなる各バッチは 1 回の更新に使われたあと、そのまま捨てられていた。シミュレータであればこうしたステップは安価に実行できる。しかしロボットアームではそうはいかない。1 ステップごとに実時間がかかり、ハードウェアが摩耗し、何かを壊す危険も伴う。本節では、エージェントが同じ制御器を数千ステップから学習するにはどうすればよいかを問う。

すでに見てきた 2 つのアイデアが、この改善の大半を担っている。1 つ目は 15.4 節が訓練に用いた過去の遷移のバッファであるリプレイであり、古い方策が選んだ行動からクリティックが学習できる限り、各遷移を 1 回ではなく何回もの更新に使えるようにする。2 つ目は 14.9 節のパスワイズ勾配であり、そのクリティックを行動そのものを通して微分することでアクターを動かす。DDPG (Lillicrap et al. 2016) はこの 2 つを決定論的なアクターと組み合わせたが、クリティックの値を上げる方向へ進むアクターはクリティックの誤差も同時に追いかけてしまうため、不安定であることが判明した。TD3 (Fujimoto et al. 2018) は、この問題の多くが 15.4.3.1 節の最大化バ

イアス（maximization bias）に起因することを突き止め、2 つのクリティックを訓練して小さい方の値を採用するようにした。ソフトアクタークリティック（SAC）(Haarnoja, Zhou, Abbeel, et al. 2018; Haarnoja, Zhou, Hartikainen, et al. 2018) は、これら両方の改善策を維持しつつ、3 つ目のアイデアとして 15.3 節のエントロピー報酬を加える。ランダムさを保つことに対して方策に報酬が支払われるため、自身のクリティックに対して訓練されるアクターは、クリティックがその時点で好む単一の行動へと崩壊することなく探索を続ける。

一言でまとめれば、SAC は方策のエントロピーを報酬に加え、その結果得られる価値をリプレイされた遷移からオフポリシーで学習し、自身がサンプリングした行動を通してクリティックを微分することでアクターを更新する。ただし、2 つの細部に注意を要する。1 つ目に、$[-2, 2]$ に制限されたトルクを扱うには、勾配が行動を通って流れるため、正しい密度をもつ有界な方策が必要になる。2 つ目に、エントロピーにかける重みは報酬のスケールに見合っていなければならないため、SAC はこれを固定せずに学習する。本節では、エントロピー正則化目的関数の評価ステップと改善ステップを導出し、有界な方策を構築して検証したうえで更新則を組み立て、最後に振り子の学習に何ステップ必要か、またクリティックの予測がどこまで信頼できるかを測定する。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

本節を通して用いる例題は、14.9 節の Pendulum である。モーター付きの支点から振り子が吊り下げられている。エージェントは $(\cos\vartheta, \sin\vartheta, \dot\vartheta)$ を観測し（ϑ は直立位置からの角度）、トルク $u \in [-2, 2]$ を加える。各ステップでは $\vartheta^2 + 0.1\,\dot\vartheta^2 + 0.001\,u^2$ のコストがかかるため、真下（$\vartheta = \pi$）で静止して垂れ下がった振り子は 1 ステップあたり $\pi^2 \approx 9.9$ のコストを要し、他のどの静止位置よりも大きくなる。1 エピソードは 200 ステップ続く。一様ランダムなトルクでは約 $-1{,}200$ の収益となり、振り上げて静止させる制御器では約 -200 以上の収益が得られる。以下のすべての実行は高々 20,000 環境ステップとし、各設定は 3 つのシードで訓練する。$c = 2$ は行動のスケールである。ステップ数の範囲や、実行間で事象がどの程度の頻度で起きるかに関する記述は、各設定あたり 20 回の実行にわたって確認したものである。

```
num_env_steps, num_seeds, c = 20_000, 3, 2.0
```

15.5.1 最大エントロピー目的関数

15.5.1.1 報酬としてのエントロピー

15.3 節では、固定された参照方策 π_{ref} の近くに方策を保つため、期待報酬から $\beta\, D_{\text{KL}}(\pi \| \pi_{\text{ref}})$ を差し引いた。ここで $\beta > 0$ は報酬に対するダイバージェンスの重みを表す。参照方策が $|\mathcal{A}|$ 個の行動上の一様分布であるとき、$D_{\text{KL}}(\pi \| \text{uniform}) = \log|\mathcal{A}| - H(\pi)$ となる。ただし $H(\pi) = -\mathbb{E}_{a\sim\pi}[\log \pi(a)]$ は方策のエントロピーである。このときペナルティは、エントロピーボーナスと、どの方策も変えられない定数の和になる。同じことが Pendulum のトルク区間にも成り立つ。$[-2, 2]$ 上の一様密度は、その区間上のあらゆる密度の中で最大のエントロピー

$\log 4 \approx 1.39$ nat をもち、$\log 4$ が $\log |\mathcal{A}|$ の役割を果たす。SAC はこのボーナスを毎ステップ適用する。

$$J(\pi) = \mathbb{E}_{\pi}\Big[\sum_{t} \gamma^{t}\big(r_t + \beta\, H(\pi(\cdot \mid s_t))\big)\Big]. \tag{15.40}$$

エージェントは訪れるすべての状態においてエントロピー 1 nat ごとに β 単位の報酬を受け取るため、**温度**（temperature）β は 1 nat あたりの報酬という単位をもつ。本節でも 15.3 節の文字 β をそのまま用いる。なお SAC の原論文では α と表記されているが、本書ではこれを時間差分学習（temporal-difference learning）のステップサイズに使っている。

15.2 節のアクター損失における任意追加のエントロピーボーナスとは異なり、この項は目的関数の一部である。そのため最適解そのものが変化し、複数の行動がほぼ同等に優れている場所では最良の方策もランダムなまま保たれる。連続行動の場合、H は密度にわたる $-\pi \log \pi$ の積分である微分エントロピー（differential entropy）となり、長さが 1 未満の区間に集中した密度に対しては負の値をとる。最大エントロピー強化学習（maximum-entropy reinforcement learning）は深層学習よりも古くからあり (Ziebart et al. 2008)、SAC はそのオフポリシー型アクタークリティックにあたる。

本章の他の手法と同様、SAC も一般化方策反復法（generalized policy iteration、Figure 14.9）と同じく 2 種類のステップを交互に繰り返す。方策評価では価値推定値を現在の方策の価値へと近づけ、方策改善ではその推定値が好む行動へと方策を動かす。エントロピーはこの両方のステップを変化させる。ネットワークを導入する前に、2 つの行動をもつ 1 つの状態でその仕組みを確認しよう。

15.5.1.2 手計算で見る 2 つの行動

価値が $Q = (1, 0)$ である 2 つの行動をもつ 1 つの状態を取り上げ、$\beta = 0.5$ とする。15.3 節で示したように、KL 正則化の最適解は参照方策に $e^{Q/\beta}$ を掛けて再正規化したものになり、その価値は $\beta \log Z$ となる。一様な参照方策のもとでは、次のようになる。

$$\pi^{\star}(a) = \frac{e^{Q(a)/\beta}}{Z}, \qquad Z = \sum_{a} e^{Q(a)/\beta},$$
$$\max_{\pi}\big(\mathbb{E}_{\pi}[Q] + \beta H(\pi)\big) = \beta \log Z.$$

ここで $\pi^{\star} = (e^2, 1)/(e^2+1) = (0.88, 0.12)$ である。そのエントロピーは 0.37 nat であり、$\mathbb{E}_{\pi^{\star}}[Q] = 0.88$ なので、ソフト価値（soft value）は $0.88 + 0.5 \cdot 0.37 = 1.06$ となって $0.5 \log(e^2+1)$ に一致する。このソフト価値は通常の最大値 1 を上回っている。すなわち、傾斜方策（tilted policy）は期待価値を 0.12 手放す代わりに、エントロピーボーナスで 0.18 を得ている。以降の議論ではもう 1 つの事実が重要になる。括弧内の項 $Q(a) - \beta \log \pi^{\star}(a)$ は両方の行動に対して同じ値をとり、$1 + 0.5 \cdot 0.13$ と $0 + 0.5 \cdot 2.13$ はいずれも 1.06 になる。最適解においては、サンプリングされた 1 つの行動から誤差なしでソフト価値が得られるのである。次のセルでは、$\pi^{\star}$ を一様方策およびほぼ貪欲な方策と比較する。

```
Q, beta = np.array([1.0, 0.0]), 0.5
pi_star = np.exp(Q / beta) / np.exp(Q / beta).sum()   # the tilted optimum
for name, pi in (('tilted', pi_star), ('uniform', np.array([0.5, 0.5])),
                 ('nearly greedy', np.array([0.99, 0.01]))):
    bracket = Q - beta * np.log(pi)        # one sampled action's soft value
```

```
        print(f'{name:>13}: pi = {pi.round(2)}, E[Q] + beta H = '
              f'{pi @ bracket:.3f}, bracket = {bracket.round(2)}')
    print(f'beta log Z = {beta * np.log(np.exp(Q / beta).sum()):.3f}')
```

```
       tilted: pi = [0.88 0.12], E[Q] + beta H = 1.063, bracket = [1.06 1.06]
      uniform: pi = [0.5 0.5], E[Q] + beta H = 0.847, bracket = [1.35 0.35]
nearly greedy: pi = [0.99 0.01], E[Q] + beta H = 1.018, bracket = [1.01 2.3 ]
beta log Z = 1.063
```

傾斜方策は $\beta \log Z = 1.063$ を達成し、他の 2 つの方策はそれに届かず、ほぼ貪欲な方策では 1.018、一様方策では 0.847 にとどまる。これらの方策のもとでは括弧内の項が 2 つの行動の間で異なり（一様方策では 1 単位の差がある）、サンプリングされた単一の行動からはソフト価値のノイズを含んだ推定値しか得られない。この推定値が厳密になるのは最適解においてのみである。

15.5.1.3 ソフト方策評価

ある状態のソフト価値は、将来の報酬とともにその状態で得られるエントロピーも織り込み、$V^\pi(s) = \mathbb{E}_{a\sim\pi}\big[Q^\pi(s,a) - \beta \log \pi(a \mid s)\big]$ と表される。ここで $Q^\pi(s,a)$ はそれ以降のすべての状態のエントロピーボーナスを含むが、s 自身のボーナスは含まない。15.1 節のクリティックは 1 ステップのターゲット $r + \gamma V(s')$（終端状態では r に打ち切る）に対して回帰した。SAC のクリティックは Q を学習するため、次の状態の価値を Q を用いて表す必要があり、上の恒等式からサンプリングされた 1 つの行動を使ってそれを得る。Q_{w^-} をクリティック Q_w のゆっくり変化するコピー、すなわち 15.4 節のターゲットネットワークとする。これにより、回帰目標が更新ごとに動くのを防ぐ。ターゲットは次の式で与えられる。

$$
\begin{aligned}
y &= r + \gamma\,(1 - \mathbf{1}(s' \text{ terminal}))\Big(Q_{w^-}(s', \tilde{a}') - \beta \log \pi_\theta(\tilde{a}' \mid s')\Big), \\
\tilde{a}' &\sim \pi_\theta(\cdot \mid s').
\end{aligned}
\tag{15.41}
$$

唯一の新規項である $-\beta \log \pi_\theta(\tilde{a}' \mid s')$ は、いまサンプリングした行動を通じて推定された s' におけるエントロピーボーナスである。遷移 (s, a, r, s') はリプレイバッファから取り出されるが、$\tilde{a}'$ は現在の方策から新たにサンプリングされる。V^π は評価対象の方策のもとでの期待値だからである。バッファは過去に何が起きたかを与え、方策は次に何をするかを与える。

1 回のサンプリングによって現在の方策のソフト価値が推定されるため、このステップはソフト方策評価（soft policy evaluation）となる。これは傾斜最適方策の価値を計算する 15.3 節の log-sum-exp バックアップではない。両者が一致するのは現在の方策がその最適解にあるときだけであり、その場合は 2 行動の例で見たように括弧内の項がすべての行動に対して同じ値になる。ソフト評価はエントロピーボーナスで拡張された報酬に対する通常の方策評価にほかならないため、14.2 節の議論がそのまま成り立つ。すなわち、評価作用素を 1 回適用するごとに最大誤差は γ 倍に縮小し、繰り返し適用すると Q^π に収束する。この議論には拡張報酬が有界であることが必要であり、報酬だけでなくエントロピーも有界でなければならない。これは行動が有限個の場合や、密度が有界に保たれる連続方策の場合に成り立つため、本節ではこれを仮定する。

15.5.1.4 ソフト方策改善

方策改善ステップもまた 15.3 節から導かれる。同節では、任意の方策の正則化目的関数が最適解の価値から KL ダイバージェンスの分だけ下回ることを示した。

$$
\mathbb{E}_\pi[r] - \beta\, D_{\mathrm{KL}}(\pi \| \pi_{\mathrm{ref}}) = \beta \log Z - \beta\, D_{\mathrm{KL}}(\pi \| \pi^\star),
\tag{15.42}
$$

ここで $\pi^\star \propto \pi_{\text{ref}}\, e^{r/\beta}$ は傾斜最適方策である。r を $Q(s,\cdot)$ に置き換え、参照方策を一様分布とすると、ある状態で $\mathbb{E}_{\pi}[Q] + \beta H(\pi)$ を最大化することは $D_{\mathrm{KL}}(\pi \,\|\, e^{Q/\beta}/Z)$ を最小化することと等価になる。両者の差である $\beta \log Z$ は π に依存しないからである。通常、ネットワークによる方策はこの傾斜分布を厳密には表現できないため、SAC は方策族の中でこのダイバージェンスの意味で最も近い要素へと方策を動かす。これが**逆 KL 射影**（reverse-KL projection）である。この向きのダイバージェンスは、$e^{Q/\beta}$ が小さい領域に方策が確率質量を置くことにペナルティを科すため、傾斜分布が離れた 2 つの峰をもつとき、ガウス分布は両方に広がるのではなく一方の峰に落ち着く（15.3.3.5 節）。行動をノイズの微分可能な関数として書くと、次のアクター損失が得られる。

$$
\begin{aligned}
L_\pi(\theta) &= \mathbb{E}_{s\sim\mathcal{D},\, z}\Big[\beta \log \pi_\theta(\tilde{a}_\theta(s,z) \mid s) - Q_w(s, \tilde{a}_\theta(s,z))\Big], \\
\tilde{a}_\theta(s,z) &= c\tanh\big(\mu_\theta(s) + \sigma_\theta(s)\, z\big),
\end{aligned}
\tag{15.43}
$$

ここで $z \sim \mathcal{N}(0, I)$ であり、状態はリプレイバッファ $\mathcal{D}$ からサンプリングされる。この損失は新たにサンプリングされた行動の負のソフト価値であるため、これを下げるとクリティックの価値とエントロピーがともに高まる。なお、行動を範囲内に収めるスクワッシュ $c\tanh$ については次項で扱う。この損失の勾配は 14.9 節のパスワイズ勾配である。ガウス行動 $a = \mu + \sigma z$ の場合、その勾配は $\nabla_\mu \mathbb{E}[Q(a)] = \mathbb{E}[Q'(\mu + \sigma z)]$ となる。すなわちスコアに重みを掛けるのではなく行動を通して微分するものであり、同節の例における 1 サンプルあたりの分散は 1.0 であった（スコア関数では 21.75）。この計算には、行動に関して微分可能なクリティックが必要になる。離散行動の場合は、代わりに行動全体にわたって和をとることで期待値を計算する。

もし各状態における最大化が厳密であれば、方策改善ステップはいたるところでソフト価値を引き上げる。

命題。報酬とエントロピーが有界であり、方策族 Π が π_{old} を含み、表形式と同様に各状態における方策を独立に選べるとする。$\pi_{\text{new}}(\cdot \mid s)$ が $\mathbb{E}_{a\sim\pi}\big[Q^{\pi_{\text{old}}}(s,a)\big] + \beta H(\pi(\cdot \mid s))$ を Π 上ですべての s において最大化するならば、いたるところで $V^{\pi_{\text{new}}} \geq V^{\pi_{\text{old}}}$ が成り立つ。

証明。H_t を π_{new} の s_t におけるエントロピーとする。$\pi_{\text{old}} \in \Pi$ であるから、$s_0 = s$ における最適性と $Q^{\pi_{\text{old}}}(s,a) = r(s,a) + \gamma\, \mathbb{E}[V^{\pi_{\text{old}}}(s_1)]$ を順に用いると、次の不等式が得られる。

$$
\begin{aligned}
V^{\pi_{\text{old}}}(s) &\leq \mathbb{E}_{a\sim\pi_{\text{new}}}\big[Q^{\pi_{\text{old}}}(s,a) - \beta \log \pi_{\text{new}}(a \mid s)\big] \\
&= \mathbb{E}_{\pi_{\text{new}}}\big[r_0 + \beta H_0 + \gamma\, V^{\pi_{\text{old}}}(s_1)\big].
\end{aligned}
$$

同じ不等式を $V^{\pi_{\text{old}}}(s_1)$、次いで $V^{\pi_{\text{old}}}(s_2)$ というように n 回繰り返し適用すると、$V^{\pi_{\text{old}}}(s) \leq \mathbb{E}_{\pi_{\text{new}}}\big[\sum_{t<n} \gamma^t (r_t + \beta H_t)\big] + \gamma^n\, \mathbb{E}\big[V^{\pi_{\text{old}}}(s_n)\big]$ が得られる。価値は有界であるため、最後の項は $n \to \infty$ で消え、和は $V^{\pi_{\text{new}}}(s)$ に収束する。■

SAC はクリティックがまだ学習している最中に各損失に対して 1 回の勾配ステップをとるため、この命題は実際に走らせるループではなく厳密なアルゴリズムを記述したものである。Haarnoja, Zhou, Abbeel, et al. (2018) はその付録 B で完全な議論を与えている。

15.5.2 スクワッシュされたガウス方策

15.5.2.1 クリッピングとパスワイズ勾配

14.9 節のガウス方策は非有界なトルクをサンプリングし、環境側で $[-2, 2]$ にクリッピングしていた。行動を通して微分することのないスコア関数推定量にとっては、それで何の問題もなかっ

た。しかし SAC のパスワイズ推定量は行動を通して微分するため、クリッピングは 2 つの問題を引き起こす。第 1 に、区間の外側ではクリップ操作の導関数が 0 になるため、境界のトルクが最良の行動である状態でさえ、境界を超えたサンプルからはアクターに勾配が一切伝わらない。第 2 に、クリッピングは各境界の外側にある確率をすべて境界そのものの上に移動させるため、クリッピングされた分布はそこで密度をもたず、Equation 15.41 と Equation 15.43 における $\log \pi$ が未定義になる。ガウス分布からの滑らかで可逆な写像による有界な分布を用いれば、これら両方の問題が解消される。Figure 15.12 に、クリッピングと以下で用いる写像 $2 \tanh u$ を並べて示す。

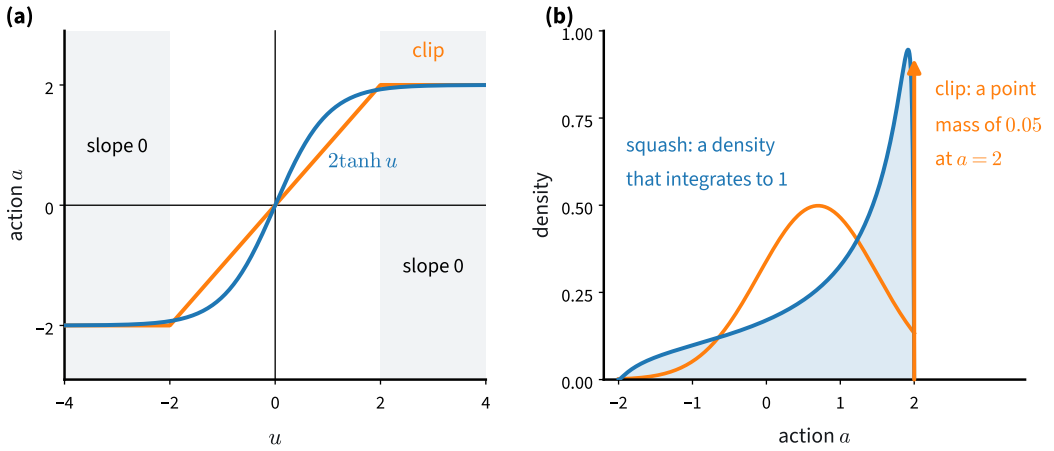


Figure 15.12: ガウスサンプル u の $[-2, 2]$ へのクリッピングとスクワッシュ。(a) クリッピングは境界の外側で傾きが 0 になるため、そこでのサンプルはアクターに勾配を与えないが、$2 \tanh u$ はいたるところで正の傾きをもつ。(b) $u \sim \mathcal{N}(0.7, 0.8^2)$ の場合、クリッピングは上界を超える約 0.05 の確率を単一の行動 $a = 2$ の上に集めるため、クリッピングされた分布はそこで密度をもたない。一方、スクワッシュは区間全体にわたって密度を与える。

15.5.2.2 tanh による変数変換

$u \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2)$ をサンプリングし、$a = c \tanh u$ によってスクワッシュする。この写像は滑らかな単調増加関数であるため、u の周りの短い区間と a の周りのその像は同じ確率をもち、像の長さは因子 $da/du = c\,(1 - \tanh^2 u)$ だけ短くなる。したがって a の密度は、座標ごとにガウス密度をこの因子で割ったものになる。

$$\log \pi(a \mid s) = \sum_i \Big[\log \mathcal{N}(u_i;\, \mu_i, \sigma_i) - \log\left(1 - \tanh^2 u_i\right) - \log c \Big]. \tag{15.44}$$

和の各項は、1 つの座標の圧縮に対してガウス分布の対数密度を補正している。Figure 15.13 に、ある平均と広がりに対するこの効果を示す。$a = 2$ の近くでは、スクワッシュによって u の上側の裾全体が行動の短い区間に詰め込まれるため、そこで密度が上昇する。この補正がないと、式はこの圧縮を見落とし、積分値が 1.13 となって 1 にならない。

$\log c$ の項はアクターの勾配を変えないが、すべての対数密度をシフトさせ、それに伴って報告されるエントロピーやターゲット内の項 $\beta \log \pi$ も $\log 2 \approx 0.69$ nat だけシフトさせる。報告されるエントロピーが実際のトルク分布のエントロピーになるよう、この項は残しておく。

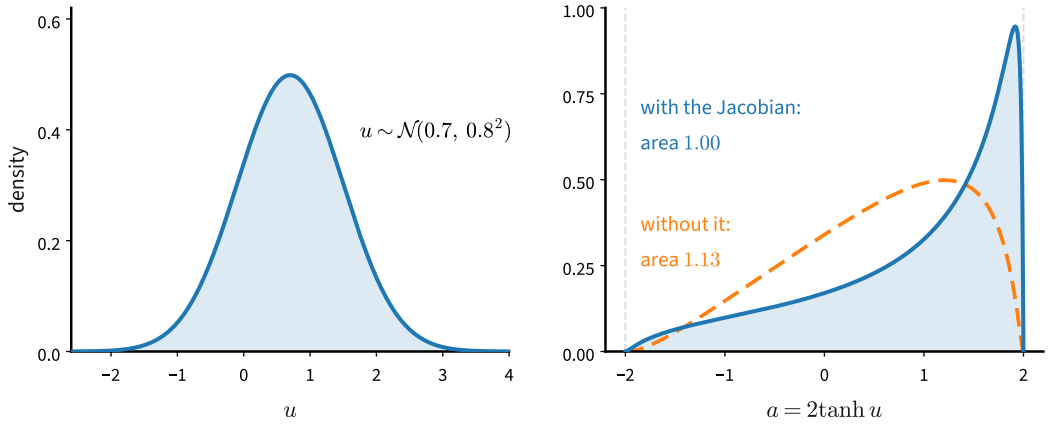


Figure 15.13: スクワッシュされたガウス分布。(a) $u \sim \mathcal{N}(0.7, 0.8^2)$ の密度。(b) 行動 $a = 2\tanh u$ の密度。Equation 15.44 のヤコビ行列の項を含めると、密度の積分値は 1 となり、$a = 2$ 付近（スクワッシュによって u の上側の裾が短い区間に圧縮される場所）で密度が上昇する。この項を省くと、式はガウス密度を単に $\operatorname{arctanh}(a/2)$ で評価したものになり、積分値は 1.13 となってその集中を見落とす。

方策ネットワークには 14.9 節の `GaussianHead` を用いる。**トランク**が特徴量を計算し、線形層がそれを平均 $\mu_\theta(s)$ に写像し、`state_std=True` の設定によって 2 つ目の線形層が $\log \sigma_\theta(s)$ を $[-5, 2]$ にクランプして出力する。SAC ではエントロピーボーナスがすべての状態で支払われるため、状態に依存する広がりが必要になる。真上で静止している振り子には狭い方策が必要である一方、振り上げの最中には広い方策を使えるが、単一の共有された σ ではその両方を実現できない。トランクには、後述するクリティックと同じ形状である 64 ユニットの ReLU 層 2 つを、14.9 節の単一の tanh 層の代わりに用いる。公開されている SAC の実装では、同じ形状で 256 ユニットが使われる。本節の実行では $\log \sigma$ が約 -3.5 より上に保たれるため、このクランプが制約として働くことはなく、取り除いても結果は変わらない。それでも保護策として残しておくのは、σ がゼロに近づくのを防ぎ、広がりの崩壊によってターゲット内の $\log \pi$ が発散しないようにするためである。ただし、平均がスクワッシュの飽和領域のずっと先まで行けばエントロピーはやはり下がるため、このクランプがスクワッシュ後の行動のエントロピーを下から抑えるわけではない。一般的な実装では $[-20, 2]$ にクランプする。

`SquashedGaussianPolicy` はこのヘッドをラップする。`sample` は $u = \mu + \sigma z$ をサンプリングし、損失の計算向けにスクワッシュ後の行動とその対数確率をいずれも微分可能な形で返す。`log_prob_u` はサンプラが保持しているスクワッシュ前の u から Equation 15.44 を評価するため、微分可能な経路上に逆双曲線正接は現れない。`act` は環境向けに NumPy 配列の行動をサンプリングし、`act_logp` は本節末尾の測定向けにその行動と対数確率を返す。これらはいずれもサンプル u を `_draw` から受け取る。また `act_greedy` はスクワッシュ後の平均 $c \tanh \mu_\theta(s)$ を返す。次節でこのクラスを再利用するため、ライブラリに保存しておく。

```
class SquashedGaussianPolicy(nnx.Module):
    """d2l.GaussianHead with SAC's network (two ReLU layers and a
    state-dependent standard deviation), squashed through
    a = scale * tanh(u)."""
    def __init__(self, obs_dim, act_dim, hidden=64, scale=2.0, seed=None):
        rngs = nnx.Rngs(d2l.get_key() if seed is None else seed)
```

```
        trunk = nnx.Sequential(
            d2l.Linear(obs_dim, hidden, rngs=rngs), jax.nn.relu,
            d2l.Linear(hidden, hidden, rngs=rngs), jax.nn.relu)
        self.policy = d2l.GaussianHead(obs_dim, act_dim, hidden,
                                       state_std=True, trunk=trunk, rngs=rngs)
        self.scale = scale

    def log_prob_u(self, u, mean, std):
        """log pi at a = scale tanh(u), from the pre-squash u the sampler
        keeps."""
        logdet = 2 * (jnp.log(2.0) - u - jax.nn.softplus(-2 * u))
        return (jax.scipy.stats.norm.logpdf(u, mean, std)
                - logdet - jnp.log(self.scale)).sum(-1)

    def sample(self, obs, key):
        """A reparameterized action and its log-probability, differentiable."""
        mean, std = self.policy(obs)
        u = mean + std * jax.random.normal(key, std.shape)
        return self.scale * jnp.tanh(u), self.log_prob_u(u, mean, std)

    def _draw(self, obs, rng):
        """A pre-squash sample u with the head's mean and std, from numpy
        states and the noise of rng."""
        mean, std = d2l.forward_np(self.policy, obs)   # compiled forward
        noise = rng.standard_normal(mean.shape, dtype=np.float32)
        return mean + std * noise, mean, std

    def act(self, obs, rng):
        """A sampled action, numpy in and out."""
        return self.scale * np.tanh(self._draw(obs, rng)[0])

    def act_logp(self, obs, rng):
        """A sampled action and its log-probability, numpy in and out."""
        u, mean, std = self._draw(obs, rng)
        return (self.scale * np.tanh(u),
                np.asarray(self.log_prob_u(u, mean, std)))

    def act_greedy(self, obs, rng=None):
        """The squashed mean scale * tanh(mean), numpy in and out."""
        return self.scale * np.tanh(d2l.forward_np(self.policy, obs)[0])
```

`_draw`（`act` と `act_logp` が共有する）および `act_greedy` は `d2l.forward_np` を通じてヘッドを実行する。この関数はネットワークごとに 1 回の順伝播をコンパイルし、呼び出しのたびに現在のパラメータを読み取るため（オンライン付録「ユーティリティ関数とクラス」）、1 状態ずつ行動を選択する際に更新後の再コンパイルを必要としない。

`log_prob_u` の内部では、バッチにわたる平均をとる前に、ガウス項とヤコビ行列の項を行動の各座標にわたって足し合わせる。一方の和をとり他方の平均をとってしまうと、座標が 1 つだけの Pendulum では正しい値になるものの、より大きなあらゆる行動空間で誤った値になる。

15.5.2.3 数値的に安定な対数行列式

直接的な形である `log(1 - tanh(u)**2)` は、$\tanh u$ が 1 に近づくとほぼ等しい 2 つの数の引き算になり、float32 が $\tanh u$ をちょうど 1 に丸めると破綻する。$1-\tanh^2 u = 4e^{-2u}/(1+e^{-2u})^2$ であるから、対数をとると次式が得られる。

$$\log\left(1-\tanh^2 u\right) = 2\left(\log 2 - u - \mathrm{softplus}(-2u)\right), \tag{15.45}$$

これはすべての u に対して成り立ち、ほぼ等しい数どうしの引き算を含まない恒等式であり、`log_prob_u` ではこれを用いている。次の float32 による計算では、直接的な形、よく使われる 10^{-6} を加える保護付きの形、および安定な形を比較する。

```
u = np.float32([0, 3, 8, 10, 20])
t = np.tanh(u)
with np.errstate(divide='ignore'):
    naive = np.log(1 - t ** 2)
guarded = np.log(1 - t ** 2 + np.float32(1e-6))
stable = 2 * (np.log(np.float32(2)) - u - np.logaddexp(np.float32(0), -2 * u))
print(f'{"u":>4} {"naive":>10} {"guarded":>10} {"stable":>10}')
for row in zip(u, naive, guarded, stable):
    print(f'{row[0]:4.0f} {row[1]:10.4f} {row[2]:10.4f} {row[3]:10.4f}')
```

```
   u      naive    guarded     stable
   0     0.0000     0.0000     0.0000
   3    -4.6187    -4.6186    -4.6187
   8   -14.5561   -13.4256   -14.6137
  10       -inf   -13.8155   -18.6137
  20       -inf   -13.8155   -38.6137
```

直接的な形は $u = 8$ ですでにわずかにずれており（float32 では $1-\tanh^2 u$ に残る有効桁がほとんどない）、さらには $-\infty$ が $u = 10$ 以降で生じる。保護項を加えれば無限大は避けられるものの、$u = 8$ で 1 nat 以上高くなりすぎ、$\log 10^{-6} = -13.8155$ の値に $u = 10$ 以降は留まるため、それ以上の飽和を検知できなくなる。その結果、収益曲線は正常に見えても、報告されるエントロピーと Equation 15.43 のエントロピー項が誤った値になる。

正規化条件から、変数変換（change of variables）に対する 2 つ目の独立した検証が得られる。行動の細かいグリッド上で密度を評価し、そのグリッド上でスクワッシュの逆変換を計算して、ヤコビ行列の項がある場合とない場合のそれぞれについて Equation 15.44 を台形公式で数値積分してみよう。

```
def squashed_logp(a, mu, sigma, logdet=True):
    up = np.arctanh(a / c)          # only here: the training path keeps u
    lp = (-0.5 * ((up - mu) / sigma) ** 2 - np.log(sigma)
          - 0.5 * np.log(2 * np.pi))
    if logdet:
        lp -= 2 * (np.log(2) - up - np.logaddexp(0, -2 * up)) + np.log(c)
    return lp

a_grid = np.linspace(-c + 1e-6, c - 1e-6, 200_001)
for m, s in ((0.0, 0.5), (0.7, 0.8)):
    w = np.trapezoid(np.exp(squashed_logp(a_grid, m, s)), a_grid)
```

```
    wo = np.trapezoid(np.exp(squashed_logp(a_grid, m, s, False)), a_grid)
    print(f'mu = {m}, sigma = {s}: integrates to {w:.6f} with the '
          f'log-det, {wo:.3f} without')
```

```
mu = 0.0, sigma = 0.5: integrates to 1.000000 with the log-det, 1.653 without
mu = 0.7, sigma = 0.8: integrates to 1.000000 with the log-det, 1.132 without
```

補正を含めれば、どちらの設定でも密度の積分値は 1.000000 になるが、補正なしでは積分値が 1.65 と 1.13 になる。この誤りがあっても Pendulum の収益はほとんど変わらない一方で、報告されるエントロピーはすべて誤った値になってしまうが、この検証なら一切訓練せずに誤りを見つけられる。なお、グリッドは 10^{-6} だけ $\pm c$ の手前で打ち切っている（境界では arctanh が無限大になるため）。この検証で適度な大きさの σ を使っているのはそのためである。

15.5.3 アルゴリズム

15.5.3.1 ツインクリティックと Polyak ターゲット

ノイズを含む推定値の最大値には上向きのバイアスが生じる（15.4.3.1 節）。たとえば同じ値に対する 2 つの独立な推定値が、それぞれ平均 0・標準偏差 σ のノイズでずれているとき、その最大値は真の値を平均して約 0.56σ 上回る。クリティックの勾配を登るアクターは、これと同じことを連続的に引き起こす。クリティックがたまたま過大評価した行動へとアクターが動き、ブートストラップによってその誤差が次のターゲットへと持ち越されるのである。TD3 に倣い、SAC は異なる初期化から 2 つのクリティック Q_{w_1} と Q_{w_2} を訓練し、それらの各点ごとの最小値を次のターゲットで用いる。

$$y = r + \gamma\,(1 - \mathbf{1}(s' \text{ terminal}))\Big(\min_{j=1,2} Q_{w_j^-}(s', \tilde{a}') - \beta \log \pi_\theta(\tilde{a}' \mid s')\Big), \tag{15.46}$$

同様に、アクター損失 Equation 15.43 においても $\min_j Q_{w_j}$ で Q_w を置き換える。両方のクリティックは同じターゲット y に対して回帰する。2 つの予測が食い違う場所では最小値が低い方を採用するため、これは意図的に悲観的な推定値となっている。本節の最後で、この悲観性がもたらすコストと、それが防ぐ現象を測定する。

DQN と同様に各クリティックはターゲット用のコピー w_j^- をもつが、15.4 節のように 250 環境ステップごとに上書きするのではない。毎回の更新のあと、オンラインの重みに向かってわずかな割合 ν だけ近づける。

$$w_j^- \leftarrow \nu\, w_j + (1 - \nu)\, w_j^-, \qquad \nu = 0.005.$$

この **Polyak 平均**（Polyak averaging）は半減期が $\ln 2 / \nu \approx 139$ 回の更新である指数移動平均にあたり、回帰目標は静止したまま突然ジャンプするのではなくゆっくりとドリフトする。コピーをもつのはクリティックだけである。ターゲット内の行動 $\tilde{a}'$ は現在の方策からサンプリングされ、確率的方策は自身の広がりにわたってターゲットを平均化するため、行動に対する追加の平滑化は必要ない。Figure 15.14 に、1 回の更新を構成する各要素のつながりを示す。

15.5.3.2 自動温度調整

固定された β は報酬とエントロピーの交換比率を固定するが、適切な比率は報酬の単位に依存する。すべての報酬を 10 倍することは、最適方策に対して β を 10 で割ることと同じ効果をもつため（演習問題 2）、あるタスク向けに調整した温度は、別の単位で測られた同じタスクには適さな

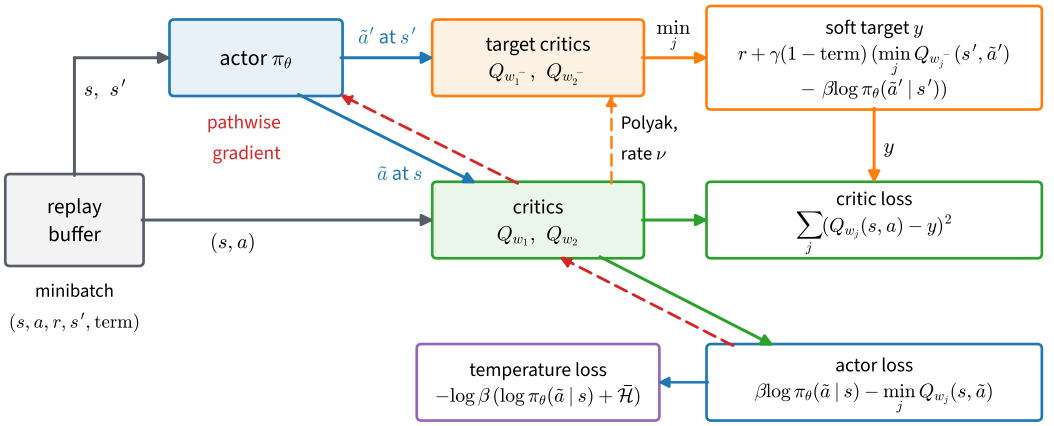


Figure 15.14: 1 回の SAC 更新。上段：ソフトターゲット y であり、新たな行動 $\tilde{a}'$（現在のアクターから s' でサンプリングしたもの）においてターゲットクリティックから計算される（2 つの最小値から $\beta \log \pi_\theta(\tilde{a}' \mid s')$ を引いたもの）。中段：クリティックが $Q_{w_j}(s,a)$ を y に回帰させる。下段：アクター損失はクリティックと新たな行動 $\tilde{a}$ を通ってアクターに届き（赤色、パスワイズ勾配）、温度は同じ対数確率に対する独自の損失をもつ。オレンジ色の破線矢印は Polyak 更新であり、ターゲットクリティックへ書き込む唯一のエッジである。

くなる。SAC の第 2 論文 (Haarnoja, Zhou, Hartikainen, et al. 2018) では、代わりにエントロピーを固定する。具体的には、訪問状態におけるエントロピーが目標値 $\bar{H}$ 以上となる最良の方策を次の制約付き問題として求める。

$$\max_{\pi} \; \mathbb{E}_{\pi}\Big[\sum_{t} \gamma^t r_t\Big] \quad \text{subject to} \quad \mathbb{E}_{s}\big[H(\pi(\cdot \mid s))\big] \geq \bar{H}, \tag{15.47}$$

そして β をこの制約のラグランジュ乗数として扱う。一般的な選択は $\bar{H} = -\dim \mathcal{A}$ であり、ここでは -1 nat となる。この乗数は、次の関数に対する勾配降下法によって学習する。

$$J(\beta) = -\log \beta \; \mathbb{E}_{s \sim \mathcal{D},\, a \sim \pi_\theta}\big[\log \pi_\theta(a \mid s) + \bar{H}\big], \tag{15.48}$$

この関数の $\log \beta$ に関する導関数は $\hat{H} - \bar{H}$、すなわちバッチのエントロピー推定値 $\hat{H} = -\mathbb{E}[\log \pi_\theta]$ が目標値をどれだけ上回っているかを表す量になる。したがって 1 回のステップは、方策が目標よりランダムな間は β を下げてランダムさを手放しやすくし、目標よりランダムでない間は β を引き上げる。$\log \beta$ を学習することで β を正に保ち、一般的な実装の既定値である $\beta = 1$ から開始して、ネットワークと同じ学習率 3×10^{-4} の Adam を用いる。勾配の符号が変わらない間、Adam は 1 ステップあたりほぼ学習率の分だけパラメータを動かすため、エントロピーが目標を上回り続ける限り、$\log \beta$ は 1 回の更新あたり約 3×10^{-4} ずつ減少する。6,000 回の更新後には $\beta \approx e^{-1.8} \approx 0.17$ になると予測されるが、これを後ほど実測された推移と比較する。

SAC の第 1 論文でもすでに 2 つのクリティックを訓練していたが、状態価値のための独立したネットワークを併用していた。本節で構築するアルゴリズムは、第 2 論文と同様に状態価値ネットワークをもたず温度を学習するものであり、一般的な実装で使われている版である。

15.5.3.3 更新ステップと訓練ループ

15.4 節のリプレイバッファである `d2l.ReplayBuffer` は、事前に確保した配列に遷移 $(s, a, r, s', \text{term})$ を保持する。`add` は 1 つの遷移を保存し、バッファが満杯になると最も古いものを上書きする。`sample` はバッチを一様にサンプリングして `d2l.Batch` として返す。その行動の列は整数を保持している。`ReplayBufferC` はその列を浮動小数点数のベクトルに広げるだけで、他は何も変更しない。方策と同様に、次節で再利用するためこれも保存しておく。

```
class ReplayBufferC(d2l.ReplayBuffer):
    """d2l.ReplayBuffer with the action column widened to float vectors."""
    def __init__(self, capacity, obs_dim, act_dim):
        super().__init__(capacity, obs_dim)
        self.act = np.zeros((capacity, act_dim), np.float32)
```

SAC は重点サンプリング比なしでオフポリシー学習を行う。次の行動を新たにサンプリングするため、クリティックのターゲット Equation 15.46 は遷移を収集した方策に依存せず、アクター損失も独自の行動をサンプリングするからである。もっとも、バッファ内の状態は古い方策が訪問したものであるため、リプレイによって損失の平均をとる状態の分布は変化しており、行動確率の比ではそのずれは補正されない。

方策と 1 つの状態価値ネットワークおよびそのオプティマイザからなる 14.9 節の `ActorCritic` コンテナは、行動価値クリティックとそのターゲットコピー、さらに温度をもつ SAC には合わない。そこで `SoftActorCritic` に、アクター、クリティックのリストとターゲットコピーのリスト、$\log \beta$、そしてアクター、クリティック、$\log \beta$ のそれぞれに対する Adam オプティマイザを保持させる。`beta=None` とすると温度は $\beta = 1$ から学習され、数値を渡すとその値に固定される。`beta` は温度をスカラー配列として返す。`min_q` はクリティックのリスト（既定ではオンラインのもの）にわたる最小値を評価し、`q_np` は本節末尾の測定向けに NumPy 入力に対して同じ計算を行う。

```
class SoftActorCritic(nnx.Module):
    """One actor, a list of action-value critics, their frozen copies, and
    the temperature beta = exp(log_beta), learned if beta is None."""
    def __init__(self, actor, qs, targets, lr, beta=None):
        self.actor = actor
        self.qs, self.targets = nnx.List(qs), nnx.List(targets)
        self.opt_pi = nnx.Optimizer(actor, optax.adam(lr), wrt=nnx.Param)
        self.opt_q = nnx.Optimizer(self.qs, optax.adam(lr), wrt=nnx.Param)
        self.learn_beta = beta is None
        start = 1.0 if beta is None else beta   # a learned beta starts at 1
        self.temp = nnx.Dict(log_beta=nnx.Param(jnp.log(start)))
        self.opt_beta = nnx.Optimizer(self.temp, optax.adam(lr),
                                      wrt=nnx.Param)

    @property
    def beta(self):
        """The temperature exp(log_beta), a scalar array."""
        return jnp.exp(self.temp.log_beta[...])

    def min_q(self, obs, act, nets=None):
        """min_j Q_j(s, a) over the online critics, or over `nets`."""
        nets = self.qs if nets is None else nets
        x = jnp.concatenate([obs, act], -1)
        return jnp.min(jnp.stack([q(x).squeeze(-1) for q in nets]), 0)
```

```
    def q_np(self, obs, act):
        """min_j Q_j(s, a), numpy in and out, through compiled forwards."""
        x = np.concatenate([obs, act], -1)
        return np.min([d2l.forward_np(q, x) for q in self.qs], 0)[..., 0]
```

make_critic は、連結された状態と行動から 1 つの数値を出力する、64 ユニットの ReLU 層 2 つを備えたクリティックを 1 つ構築する。make_sac は、ターゲットコピーがクリティックと同じ重みから始まるエージェントを構築する。

```
def make_critic(in_dim, *, rngs):
    """Q(s, a) from the concatenated state and action."""
    return nnx.Sequential(d2l.Linear(in_dim, 64, rngs=rngs), jax.nn.relu,
                          d2l.Linear(64, 64, rngs=rngs), jax.nn.relu,
                          d2l.Linear(64, 1, rngs=rngs))

def make_sac(seed, obs_dim, act_dim, num_critics=2, lr=3e-4, beta=None):
    """A seeded agent whose target critics start as copies of the critics."""
    rngs = nnx.Rngs(seed)          # one stream: the actor's key, then critics
    actor = SquashedGaussianPolicy(obs_dim, act_dim, seed=rngs())
    qs = [make_critic(obs_dim + act_dim, rngs=rngs)
          for _ in range(num_critics)]
    return SoftActorCritic(actor, qs, [nnx.clone(q) for q in qs], lr, beta)
```

1 回の更新は Figure 15.14 に従い、3 つの関数がその損失を担う。soft_target は現在の方策から新たにサンプリングした行動でターゲット Equation 15.46 を計算する。これは回帰のためのデータであるため、そこへ勾配は流れない。critic_loss はすべてのクリティックをその単一のターゲットに回帰させ、actor_loss は新たに再パラメータ化された行動におけるクリティックの最小値を用いた Equation 15.43 であり、温度の損失が再利用する対数確率もあわせて返す。

```
def soft_target(agent, rew, next_obs, term, beta, gamma, key):
    """eq_sac_target_twin at a fresh action from the live policy."""
    a2, logp2 = agent.actor.sample(next_obs, key)
    return rew + gamma * (1 - term) * (
        agent.min_q(next_obs, a2, agent.targets) - beta * logp2)

def critic_loss(qs, obs, act, y):
    """Squared error of every critic against the one shared target y."""
    x = jnp.concatenate([obs, act], -1)
    return sum(((q(x).squeeze(-1) - y) ** 2).mean() for q in qs)

def actor_loss(actor, min_q, obs, beta, key):
    """beta log pi - min_j Q_j at a fresh reparameterized action; also
    returns log pi."""
    a, logp = actor.sample(obs, key)
    return (beta * logp - min_q(obs, a)).mean(), logp
```

sac_step がこの更新を実行する。まずクリティックが critic_loss に対して 1 ステップ進み、次にアクターが actor_loss に対して 1 ステップ進む。エージェントが β を学習する場合、温度もアクターの対数確率を用いて $J(\beta)$ に対して 1 ステップ進む。最後にターゲットクリティックが ν（コード内では polyak）の割合だけ移動する。このステップは $\log \pi_\theta$ のバッチ平均（その負の値

が方策のエントロピーの推定値になる）を返すが、訓練ループが更新ごとに待機しなくて済むようデバイス上に置いたままにする。

```
@nnx.jit
def sac_step(agent, obs, act, rew, next_obs, term, key, gamma, polyak,
             target_entropy):
    """One SAC update: soft critic regression, pathwise actor step, Polyak
    averaging, and, if the agent learns beta, one temperature step toward
    the target entropy. Returns the batch mean of log pi."""
    k1, k2 = jax.random.split(key)
    beta = agent.beta
    y = soft_target(agent, rew, next_obs, term, beta, gamma, k1)
    agent.opt_q.update(agent.qs, nnx.grad(critic_loss)(agent.qs, obs, act, y))
    grads, logp = nnx.grad(actor_loss, has_aux=True)(agent.actor,
                                                     agent.min_q, obs, beta, k2)
    agent.opt_pi.update(agent.actor, grads)
    if agent.learn_beta:          # entropy above target: beta falls
        grads = nnx.grad(lambda temp: -temp.log_beta[...] * (
            logp.mean() + target_entropy))(agent.temp)
        agent.opt_beta.update(agent.temp, grads)
    for q, tnet in zip(agent.qs, agent.targets):   # Polyak: the drifting copy
        nnx.update(tnet, optax.incremental_update(
            nnx.state(q, nnx.Param), nnx.state(tnet, nnx.Param), polyak))
    return logp.mean()
```

`sac_step` は `nnx.jit` でコンパイルされるため、`d2l.Batch` ではなくバッチの各配列と乱数キーを受け取る。`nnx.grad` は各損失を第 1 引数（クリティックまたはアクター）に関して微分し、ターゲット `y` はデータとしてクリティックの損失に入る。バッチの形状は変化しないため、このステップのコンパイルは 1 回で済む。`optax.incremental_update` は各クリティックのパラメータとそのコピーのパラメータの Polyak 平均を計算する。

`train_sac` は Pendulum 上で 1 つのエージェントを実行する。最初の `warmup` ステップでは一様ランダムなトルクをとり、その後はアクターが行動を選択し、各環境ステップのあとに `utd` 回の更新を行う。この**更新対データ比**（update-to-data ratio）は、1 環境ステップあたりに行う勾配ステップの数である。ループは `terminated` を保存し、`truncated` は保存しない。Pendulum には終端状態がなく、エピソードは 200 ステップの時間制限でのみ終了するため、保存されるフラグは常に 0 となり、時間制限で求められる通りクリティックは打ち切りをまたいでブートストラップする（14.9 節）。各エピソードの終わりに、ループはステップ数、そのエピソードの収益、エントロピーの推定値としてエピソード内の更新にわたる $-\log \pi_\theta$ の平均、および現在の β を生成する。以降のすべての実験でこの単一のループ（15.4 節の `train_dqn` のオフポリシーループを連続行動にしたもの）を使えるよう、更新比とバッファは引数として渡す。`make_update` は `sac_step` をエージェントに束縛し、サンプリングされた `d2l.Batch` を配列に変換する。また実行時のサンプリングノイズにもシードを設定するため、単独で実行しても他の実行のあとに続けて実行しても同じ結果が得られる。

```
def make_update(agent, seed, **hp):
    """sac_step bound to the agent, on a d2l.Batch; every update folds its
    count into this run's key, for fresh sampling noise."""
    step_fn = nnx.cached_partial(sac_step, agent)
    key, n = jax.random.key(seed), 0
    def update(b):
```

```
        nonlocal n
        n += 1
        return step_fn(*map(jnp.asarray, (b.obs, b.act, b.rew, b.next_obs,
                                          b.term)),
                       jax.random.fold_in(key, n), **hp)
    return update
```

nnx.cached_partial はコンパイル済みのステップにエージェントを 1 度だけ束縛するため、ループは呼び出しのたびにエージェントのモジュールを走査せずに済む。

```
def train_sac(seed, agent, buffer, num_env_steps, utd=1, warmup=1_000,
              batch_size=256, gamma=0.99, polyak=0.005, target_entropy=-1.0):
    """SAC on Pendulum: `warmup` uniformly random steps, then `utd` updates
    per environment step. Yields (env step, episode return, policy
    entropy, beta) at the end of every episode."""
    update = make_update(agent, seed, gamma=gamma, polyak=polyak,
                         target_entropy=target_entropy)
    rng, env = np.random.default_rng(seed), gym.make('Pendulum-v1')
    scale, act_dim = agent.actor.scale, buffer.act.shape[1]
    obs, ep_return = env.reset(seed=seed)[0], 0.0
    logp_sum, n_upd = 0.0, 0
    for t in range(1, num_env_steps + 1):
        a = (rng.uniform(-scale, scale, act_dim).astype(np.float32)
             if t <= warmup else agent.actor.act(obs, rng))
        next_obs, rew, terminated, truncated, _ = env.step(a)
        buffer.add(obs, a, rew, next_obs, float(terminated))
        obs, ep_return = next_obs, ep_return + rew
        for _ in range(utd if t >= warmup else 0):
            logp_sum = logp_sum + update(buffer.sample(batch_size, rng))
            n_upd += 1
        if terminated or truncated:   # read the device sum once per episode
            yield (t, ep_return, float(-logp_sum / n_upd) if n_upd else np.nan,
                   float(agent.beta))
            obs, ep_return = env.reset()[0], 0.0
            logp_sum, n_upd = 0.0, 0
```

run_arm は 1 つの設定を複数のシードで訓練し、後の測定のためにエージェントを保持する。そのバッファは 1 回の実行のすべての遷移を保持する。まずはツインクリティックと学習される温度を備えた 2 つの設定、すなわち 1 環境ステップあたり 1 回の更新で 20,000 ステップ行う設定と、1 ステップあたり 10 回の更新で 4,000 ステップ行う設定から始める。

```
def run_arm(seeds, num_env_steps, utd=1, num_critics=2, beta=None):
    """Train one SAC configuration per seed. Returns the agents and, per
    seed, an array of per-episode rows (step, return, entropy, beta)."""
    agents = [make_sac(seed, 3, 1, num_critics, beta=beta) for seed in seeds]
    runs = [np.array(list(train_sac(seed, agent,
                                    ReplayBufferC(num_env_steps, 3, 1),
                                    num_env_steps, utd=utd)))
            for seed, agent in zip(seeds, agents)]
    return agents, runs
```

```
seeds, agents, runs = range(num_seeds), {}, {}
agents['UTD 1'], runs['UTD 1'] = run_arm(seeds, num_env_steps)
agents['UTD 10'], runs['UTD 10'] = run_arm(seeds, 4_000, utd=10)
```

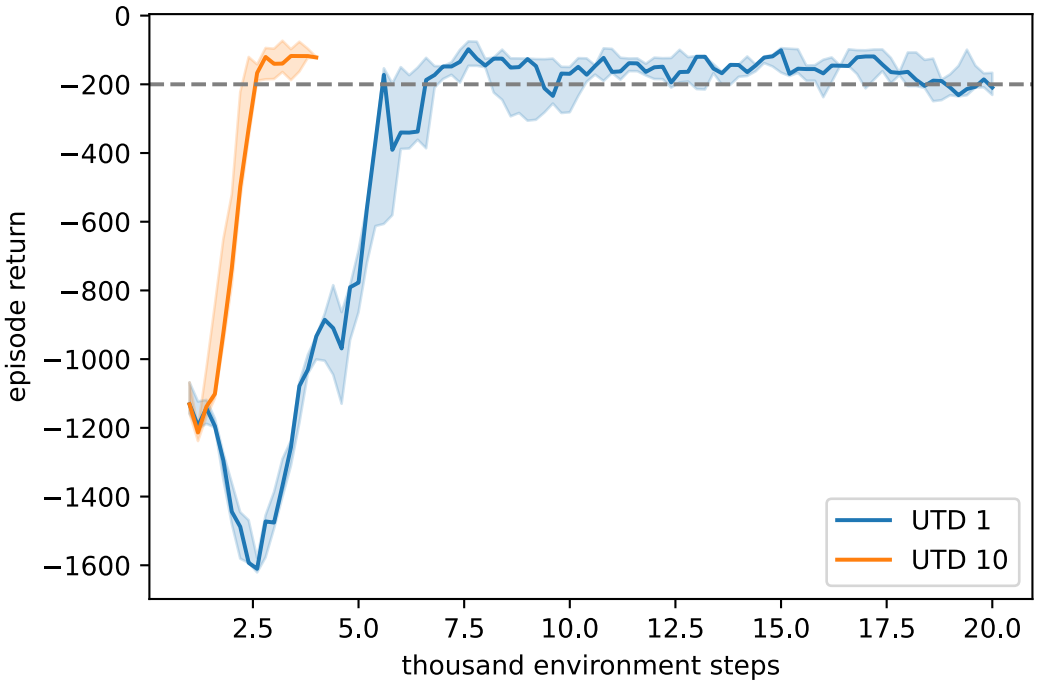
15.5.4 サンプル効率と較正

15.5.4.1 環境ステップに対する収益

次のプロットは、直近 5 エピソードにわたって平均した各エピソードの収益を環境ステップに対して描いたものであり、振り上げて静止させる制御器の水準として -200 に印を付けている。`crossings` は各条件のシードごとに、5 エピソード平均が初めてその水準に達するステップ数を `d2l.steps_to`（15.1 節）で求めて並べる（一度も到達しない実行があれば `inf` となる）。

```
def crossings(arm):
    """Per seed of an arm, the env step at which the five-episode average
    return first reaches -200."""
    return ', '.join(f'{d2l.steps_to(r, -200, 5):.0f}' for r in runs[arm])

grid = np.arange(1_000, num_env_steps + 1, 200)
curves = {arm: d2l.on_grid(runs[arm], grid, 1, smooth=5) for arm in runs}
curves['UTD 10'][:, grid > 4_000] = np.nan      # those runs end at 4,000
d2l.plot_curves(curves, x=grid / 1000, xlabel='thousand environment steps',
                ylabel='episode return', reference=-200)
for arm in runs:
    print(f'{arm}: steps to a five-episode average of -200, per seed '
          f'{crossings(arm)}')
```

```
UTD 1: steps to a five-episode average of -200, per seed 6600, 5600, 5600
UTD 10: steps to a five-episode average of -200, per seed 2600, 2400, 2600
```



1 環境ステップあたり 1 回の更新では、すべてのシードが約 5,000〜11,000 環境ステップ後に -200 を超える。学習されたベースライン付きの REINFORCE が同じタスクで同じ水準を超えたのは数十万ステップ後であり（14.9 節）、1〜2 桁多いステップを要していた。この差は 3 つの変更によって説明できる。第 1 に、REINFORCE が 1 回の更新に 1,600 ステップを費やしていたのに対し、リプレイによって各環境ステップごとに勾配ステップを実行できる。第 2 に、ブートストラップされたターゲットによって、クリティックはエピソード全体の収益ではなく単一の遷移から学習できる。第 3 に、パスワイズ勾配によって、トルクに応じて価値がどう変化するかというクリティックの知識がそのままアクターへ伝わる。

曲線は上昇する前にいったん下降する。ランダムなウォームアップのトルクによるエピソードの収益は約 $-1{,}200$ であるが、アクターが引き継ぐと、1,500〜3,000 ステップの間で収益はランダムより悪い約 $-1{,}600$ まで落ち込んでから上昇に転じる。これほど大きなコストを生む振る舞いには 2 つの可能性がある。アクターが振り子を回転させ続けて速度コストを被っているか、振り子を真下に保って最大の角度コストを被っているかである。両者を見分けるため、方策が振り子に何をしているかを測定する。`behavior` は 10 エピソードを実行し、平均収益、$\cos\vartheta$ の平均（真下で -1、真上で 1 となる）、および平均速さ $|\dot{\vartheta}|$ を報告する。比較基準として、トルク 0、運動方向に沿って常にトルク $2\,\mathrm{sign}(\dot{\vartheta})$ で押し続ける回転制御器、およびあらゆる運動に逆らう減衰制御器 $-2\tanh(2\dot{\vartheta})$ の 3 つの固定制御器を用いる。

```
def behavior(policy, num_episodes=10, seed=0):
    """Mean return, cos(theta) and |theta dot| over a policy's episodes."""
    env, rng = gym.make('Pendulum-v1'), np.random.default_rng(seed)
    env.reset(seed=seed)
    ret, states = 0.0, []
    for _ in range(num_episodes):
        obs, done = env.reset()[0], False
        while not done:
```

```
            obs, rew, terminated, truncated, _ = env.step(policy(obs, rng))
            done, ret = terminated or truncated, ret + rew
            states.append(obs)
        s = np.array(states)
        return ret / num_episodes, s[:, 0].mean(), np.abs(s[:, 2]).mean()

    policies = {'zero torque': lambda obs, rng: np.zeros(1, np.float32),
                'spin': lambda obs, rng: 2 * np.sign(obs[2:]),
                'damp': lambda obs, rng: -2 * np.tanh(2 * obs[2:])}
```

これらを、落ち込みの最中にある 2,000 ステップ後の各シードのアクター、および訓練済みエージェントと比較する。各実行は独自のシードを設定するため、同じシードを再び 2,000 ステップ訓練すれば上の実行の序盤がそのまま再現される。ここではアクターの平均行動 $c \tanh \mu_\theta(s)$ を測定する。

```
    for seed in seeds:
        agent = make_sac(seed, 3, 1)
        for _ in train_sac(seed, agent, ReplayBufferC(2_000, 3, 1), 2_000):
            pass
        policies[f'SAC at 2k, seed {seed}'] = agent.actor.act_greedy
    policies['SAC at 20k, seed 0'] = agents['UTD 1'][0].actor.act_greedy
    print(f'{"policy":>18} {"return":>8} {"cos theta":>10} {"|theta dot|":>12}')
    for name, policy in policies.items():
        ret, cos, speed = behavior(policy)
        print(f'{name:>18} {ret:8.0f} {cos:10.2f} {speed:12.2f}')
```

```
            policy   return  cos theta  |theta dot|
       zero torque    -1290      -0.49         2.50
              spin    -1610       0.04         6.83
              damp    -1886      -0.96         0.22
 SAC at 2k, seed 0    -1788      -0.93         0.62
 SAC at 2k, seed 1    -1786      -0.93         0.60
 SAC at 2k, seed 2    -1740      -0.93         0.79
SAC at 20k, seed 0     -167       0.83         0.67
```

2,000 ステップ後の時点では、どのアクターも振り子を真下付近（$\cos\vartheta$ が約 -0.9）に保ち、速さも毎秒約 1 ラジアン以下でほぼ静止させている。訓練済みエージェントが振り子を真上付近に保つのに対し、序盤のアクターは 3 つの比較基準の中で最もスコアの悪い減衰制御器のように振る舞っている。回転制御器も同程度の収益になるが、高速で $\cos\vartheta$ が 0 付近になるというまったく異なる運動をしており、そのような動きを示すアクターは 1 つもない。つまりアクターはまず揺れを減衰させることを学習し、その結果として最もコストの高い位置に振り子を垂れ下げたままにし、あとになってようやくエネルギーを注入して振り上げることを学習するのである。全 20 回の実行すべてにおいて、2,000 ステップ後のアクターの平均行動は同様に振り子を真下に保っていた。個別に検証してはいないが、考えられる理由は次のとおりである。初期のクリティックはまだ数ステップ先までしかコストを伝播できておらず、数ステップの間に行動が変えられるのは速度コストとトルクコストであるため、減衰させるのが最良に見えるのである。

2 本目の曲線は、1 環境ステップあたり 10 回の更新を行う設定である。1 ステップあたり 1 回の更新での 5,400〜11,200 ステップに対し、2,400〜4,000 環境ステップ後に -200 を超えており、ステップ数は約 2.5 分の 1 に減っている。収益の落ち込みも 1,200 ステップ付近と早く訪れ、1,600 ステップまでには解消される。環境ステップの節約には計算コストがかかる。水準を超えるまでに高い更新比は約 3 倍の勾配ステップを要し、4,000 ステップの実行全体では 30,000 回の勾配ス

テップとなり、これは 20,000 ステップの実行における 19,000 回の約 1.6 倍にあたる。高い更新比は計算量と環境との相互作用を交換するものであり、まさにロボットアームが求めるトレードオフである。最初の 1,000 ステップはランダムであり、5 エピソード平均は 2,000 ステップより前には -200 に到達できないため、この方法で 2,400 ステップよりさらに短縮できる余地はほとんどない。より難しいタスクでは、この交換は 2 つの限界に突き当たる。1 つは以下で調べる過大評価である。もう 1 つは**初期バイアス**（primacy bias）であり、最初の少数の遷移に対して何度も更新されたネットワークがそれらに過剰適合し、後から得られるデータからうまく学習できなくなる現象である。これに対しては、バッファを保持したままネットワークを定期的にリセットするのが標準的な対処法となっている (Nikishin et al. 2022)。

15.5.4.2 エントロピーと温度

学習される温度の働きを見るため、多くの実装で使われる固定値 $\beta = 0.2$ と、1 環境ステップあたり 1 回の更新による追加の 3 回の実行で比較する。

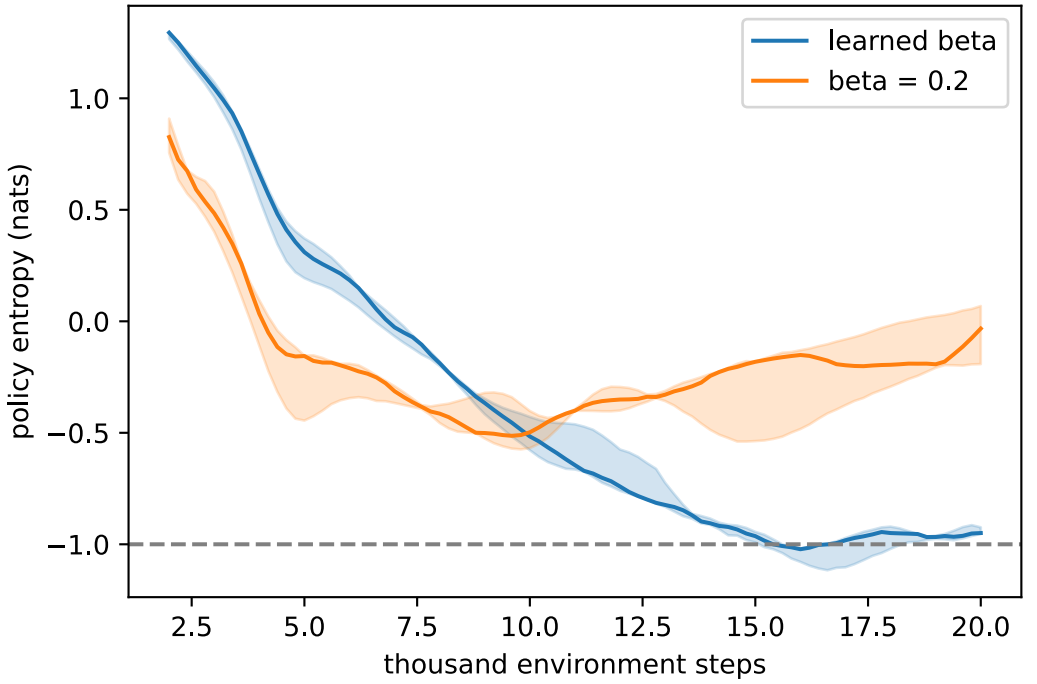
```
agents['beta = 0.2'], runs['beta = 0.2'] = run_arm(seeds, num_env_steps,
                                                   beta=0.2)
print(f'beta = 0.2: steps to -200, per seed {crossings("beta = 0.2")}')
```

```
beta = 0.2: steps to -200, per seed 6600, 8000, 6200
```

固定温度でも学習される温度と同じ範囲のステップ数で -200 を超え、どちらが一貫して早いということはなかった。目標値 -1 nat に対する両者のエントロピー推定値は次のようになる。

```
pair = {'learned beta': runs['UTD 1'], 'beta = 0.2': runs['beta = 0.2']}
grid = np.arange(2_000, num_env_steps + 1, 200)
d2l.plot_curves({arm: d2l.on_grid(r, grid, 2, smooth=5)
                 for arm, r in pair.items()},
                x=grid / 1000, xlabel='thousand environment steps',
                ylabel='policy entropy (nats)', reference=-1)
for arm, r in pair.items():
    print(f'{arm}: entropy over the last 20 episodes, per seed '
          f'{np.round([np.nanmean(x[-20:, 2]) for x in r], 2)}')
```

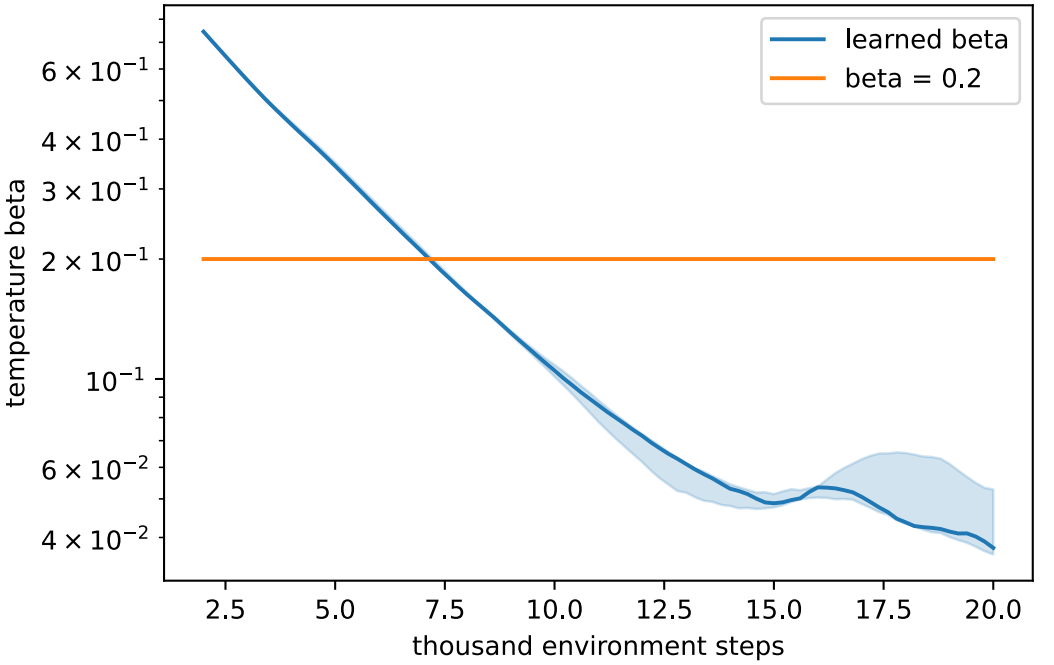
```
learned beta: entropy over the last 20 episodes, per seed [-1.   -0.95 -0.96]
beta = 0.2: entropy over the last 20 episodes, per seed [-0.   -0.23 -0.19]
```



温度を学習する場合、エントロピーは 1 nat 超から目標値へと着実に下がり、約 15,000 ステップ以降は目標値の近くに留まる。固定の $\beta = 0.2$ では、振り上げを学習している間にゼロを数十分の 1 nat 下回るまで落ち込み、その後再び上昇する。振り子が静止できるようになると、最良のトルクの周りでクリティックがほぼ平坦になるため、方策を広げても価値はほとんど損なわれない。すなわち、行動に関して曲率 h の二次関数となっているクリティックに対しては最良のガウス分布が分散 β/h をもち（演習問題 2）、クリティックが平坦なほど広い方策が許容される。20,000 ステップの時点でもこの上昇はまだ続いており、固定温度における最終的なエントロピーは実行をいつ止めるかに依存してしまう。対数スケールで描いた両設定の温度は次のとおりである。

```
d2l.plot_curves({arm: d2l.on_grid(r, grid, 3) for arm, r in pair.items()},
                x=grid / 1000, xlabel='thousand environment steps',
                ylabel='temperature beta', yscale='log')
print('learned beta, median over seeds at 2k, 7k and 20k steps:',
      np.median(d2l.on_grid(pair['learned beta'], [2_000, 7_000, 20_000], 3),
                axis=0).round(3))
```

```
learned beta, median over seeds at 2k, 7k and 20k steps: [0.745 0.207 0.038]
```



約 10,000 ステップまで、学習される β は 1 から対数スケール上の直線に沿って減少する。これはエントロピーが目標値を上回っている間、Adam が 1 回の更新あたりほぼ学習率の分だけステップを進めるためである。その値は 0.2 付近（実行が -200 を超える時点）にあり、6,000 回の更新後に 0.17 になるという予測に近い。エントロピーが目標値に達すると減少は緩やかになり、最終的に β は数百分の 1 程度に落ち着く。クリティックが平坦になるにつれて β を下げることで、学習される温度はエントロピーの落ち込みと反発を取り除くが、振り子の学習速度自体は変えない。

訓練中の収益は確率的方策から得られる。エントロピー自体がその方策の最大化する対象の一部だからである。これに加え、すべての訓練済みエージェントの平均行動 $c \tanh \mu_\theta(s)$ も、それぞれ 20 エピソードにわたって評価してみよう。

```
env = gym.make('Pendulum-v1')
for arm in runs:
    greedy_return, cost = [], []
    for seed, agent in zip(seeds, agents[arm]):
        env.reset(seed=100 + seed)
        g = d2l.evaluate(env, agent.actor.act_greedy, num_episodes=20)
        env.reset(seed=100 + seed)
        s = d2l.evaluate(env, agent.actor.act, num_episodes=20,
                         rng=np.random.default_rng(1000 + seed))
        greedy_return.append(g)
        cost.append(g - s)
    print(f'{arm:>10}: mean action {np.round(greedy_return)}, '
          f'cost of sampling {np.round(cost, 1)}')
```

```
     UTD 1: mean action [-181. -139. -184.], cost of sampling [-0.1  0.3  0.3]
    UTD 10: mean action [-182. -141. -187.], cost of sampling [-0.2  0.2 -0.2]
beta = 0.2: mean action [-184. -146. -183.], cost of sampling [6.5 0.5 0. ]
```

どのエージェントの平均行動も約 -120〜-190 の収益を達成しており、代わりに方策からサンプリングしても収益の変化は高々約 10 ポイントにとどまる。学習された状態依存の広がりによって、収益をほとんど損なうことなく方策のランダムさが保たれている。

15.5.4.3 クリティックの較正

アクターはクリティックを登るため、クリティックの誤差はそのままアクターの誤差になる。2 つのクリティックの最小値が本来の目的を果たしているかを確かめるため、エピソードの最初の状態における予測と、その後に方策が実際に集める値とを比較する。各エージェントについて確率的なエピソードを 20 回実行し、$\min_j Q_{w_j}(s_0, a_0)$ とともに、ソフトクリティックが予測対象とする実現割引ソフト収益 $r_0 + \sum_{t \geq 1} \gamma^t (r_t - \beta \log \pi_\theta(a_t \mid s_t))$ を記録する。なお、コード中の実現収益には $-\beta \log \pi_\theta(a_0 \mid s_0)$ も含めているが、その平均は β と s_0 でのエントロピーの積であり、これらのエージェントでは大きさが 0.1 未満にとどまる。また、実現収益はステップ 200 で打ち切られるのに対し、クリティックは時間制限をまたいでブートストラップし無限期間の収益を予測する。打ち切られた裾は $\gamma^{200} \approx 0.13$ にステップ 200 での価値を掛けたものだが、静止した振り子ではその価値がほぼゼロになる。

最小値をとらないクリティックを調べるため、上のツインクリティックが最も速く学習した設定である 1 環境ステップあたり 10 回の更新で、単一クリティックのエージェントを訓練する。

```
agents['single'], runs['single'] = run_arm(seeds, 4_000, utd=10,
                                           num_critics=1)
print(f'single critic, UTD 10: steps to -200, per seed '
      f'{crossings("single")}')
```

```
single critic, UTD 10: steps to -200, per seed 2400, 2400, 2600
```

この更新比では、単一クリティックもツインクリティックと同じ範囲のステップ数で -200 に達するため、収益だけでは両者を区別できない。しかし較正（calibration）を調べると違いが現れる。

```
def calibration(agent, seed, gamma=0.99, num_episodes=20):
    """Mean predicted min_j Q(s0, a0) and mean realized discounted soft
    return, at the agent's temperature beta."""
    rng, env = np.random.default_rng(seed), gym.make('Pendulum-v1')
    env.reset(seed=seed)
    beta, preds, softs = float(agent.beta), [], []
    for _ in range(num_episodes):
        obs, done, disc, soft = env.reset()[0], False, 1.0, 0.0
        a, logp = agent.actor.act_logp(obs, rng)
        preds.append(float(agent.q_np(obs, a)))
        while not done:
            obs, rew, terminated, truncated, _ = env.step(a)
            done = terminated or truncated
            soft += disc * (rew - beta * logp)
            disc *= gamma
            if not done:
                a, logp = agent.actor.act_logp(obs, rng)
        softs.append(soft)
    return np.mean(preds), np.mean(softs)
```

```
    for name, arm in (('twin, UTD 1', 'UTD 1'), ('twin, UTD 10', 'UTD 10'),
                      ('single, UTD 10', 'single')):
        for seed, agent in zip(seeds, agents[arm]):
            pred, soft = calibration(agent, 200 + seed)
            print(f'{name:>14}, seed {seed}: predicted {pred:7.1f}, '
                  f'realized {soft:7.1f}, gap {pred - soft:+6.1f}')
```

```
   twin, UTD 1, seed 0: predicted  -136.6, realized  -114.6, gap  -22.0
   twin, UTD 1, seed 1: predicted  -107.0, realized   -92.7, gap  -14.3
   twin, UTD 1, seed 2: predicted  -164.2, realized  -138.8, gap  -25.3
  twin, UTD 10, seed 0: predicted  -142.2, realized  -116.7, gap  -25.5
  twin, UTD 10, seed 1: predicted   -94.4, realized   -95.0, gap   +0.6
  twin, UTD 10, seed 2: predicted  -152.8, realized  -141.3, gap  -11.5
single, UTD 10, seed 0: predicted   -95.8, realized  -114.9, gap  +19.1
single, UTD 10, seed 1: predicted   -61.9, realized   -94.0, gap  +32.0
single, UTD 10, seed 2: predicted  -118.6, realized  -139.7, gap  +21.1
```

ツインクリティックの最小値は、ほぼすべての実行で実現収益を下回り、残りの実行でも上回るのは高々数ポイントにとどまる。一方、単一クリティックはすべての実行でツインの最小値より高く予測し、大半の実行で実現収益をも上回る。各設定あたり 20 回の実行にわたってこれらの符号の傾向はそのとおりに保たれた一方、乖離の大きさは実行ごとに数十ポイント単位でばらついた。したがって最小値をとることは、ここで単一クリティックが見せた過大評価を防ぐ一方で、最小値自体は過小な予測を行う。なお Pendulum では、どちらであっても収益は変わらない。より高い更新比によって単一クリティックの過大評価がさらに強まるかどうかは、演習問題 6 で扱う。

こうして振り子は数千環境ステップから学習できるようになり、さらに更新比を高めれば計算量と引き換えに環境ステップをもう一段減らせるようになった。それでも各遷移がクリティックに教えるのは 1 回の回帰のターゲットという 1 つの数値だけであり、1 ステップあたり 10 回の更新を行っても振り上げには数千個の遷移を要した。ブランコを数回押したことのある人間なら、次にブランコがどこへ動くかを予測できる。15.6 節では、遷移から振り子の物理法則を学習するエージェントが、数百ステップから振り上げを計画できるかどうかを問う。

15.5.5 まとめ

SAC はオフポリシー型アクタークリティックによって報酬とエントロピーの和を最大化する。ソフト方策評価はクリティックのブートストラップターゲットに $-\beta \log \pi$ を加え、ソフト方策改善はクリティックを通るパスワイズ勾配によって $e^{Q/\beta}$ の逆 KL 射影へとアクターを動かす。tanh スクワッシュによって行動を有界に保ち、その対数密度には安定な形で計算された変数変換の項が必要となる。Polyak 平均によるコピーを備えたツインクリティックがクリティックの過大評価を防ぎ、学習される温度は報酬とエントロピーの交換比率を固定する代わりにエントロピーを目標値に保つ。Pendulum において、SAC は 1 ステップあたり 1 回の更新で約 5,000〜11,000 環境ステップ以内、10 回の更新で約 3,000 環境ステップ以内に振り上げを達成し、これは REINFORCE より 1〜2 桁少ないステップ数である。

15.5.6 演習問題

1. **ターゲット内のエントロピー**。クリティックは、項 $-\beta \log \pi_\theta$（Equation 15.41）によって後続の状態で得られるエントロピーを価値に含めるが、15.2 節のエントロピーボーナスはアクター損失にしか作用しない。開始状態 s_0 に L と R の 2 つの行動があり、どちらの報酬も 0 であるとする。行動 L は状態 s_L（選択可能な行動は 1 つだけ）に遷移し、行動 R は状

態 s_R（選択可能な行動は n 個）に遷移する。s_L と s_R におけるすべての行動は報酬が 0 であり、エピソードを終了させる。

1. Equation 15.40 のもとで、s_L と s_R の最適ソフト価値、および s_0 における R の最適確率を求めよ。$n = 10$、$\gamma = 0.99$ のときにその確率を評価せよ。この確率は β にどう依存するか。
2. クリティックのターゲットから $-\beta \log \pi_\theta$ を省く一方、アクター損失には $\beta \log \pi_\theta$ を残すとする。アクターは R のどの確率へ収束し、この変種はどの目的関数を最適化するか。
3. 各状態で選択可能な行動上の一様分布を参照方策とする 15.3 節の KL ペナルティは、定数の差を除いてエントロピーボーナスに等しい。このペナルティを用いて小問 a を繰り返せ。ここではその定数が最適方策を変えるのに、Pendulum では最適方策を変えないのはなぜか。

2. [code] **二次クリティック**。最良の行動 $a^\star$ の近傍で、クリティックが 1 次元の行動に関する二次関数 $Q(s,a) = Q(s,a^\star) - \frac{h}{2}(a-a^\star)^2$（曲率 $h > 0$）であり、方策がスクワッシュなしのガウス分布 $\mathcal{N}(\mu, \sigma^2)$ であるとする。

1. $\mathbb{E}_{a\sim\pi}[Q(s,a)] + \beta H(\pi)$ を μ と σ に関して最大化せよ。最大点が傾斜分布 $e^{Q(s,\cdot)/\beta}/Z$ となり、したがって 15.5.1.4 節の逆 KL 射影がこの分布族に対して厳密になることを示せ。
2. 最適方策による価値の減少量が $Q(s,a^\star) - \mathbb{E}_{a\sim\pi}[Q(s,a)] = \beta/2$ となり、任意の h に対して成り立つことを示せ。
3. 演習問題 1c（15.3 節）では、単一の意思決定において報酬を定数倍することが KL 係数を同じ因子で割ることと同じ効果をもつことを示した。任意のマルコフ決定過程において、すべての報酬に $k > 0$ を掛けることが、Equation 15.40 のもとで β を k で割ることと同じ最適方策を導くことを示せ。報酬の単位が変わったとき、固定された β を再調整しなければならないのはなぜか。
4. $\beta \to 0$ とする。σ はどうなるか。$\beta = 0$ かつ $\sigma_\theta = 0$ のとき、損失 Equation 15.43 は $-Q_w\big(s, c\tanh\mu_\theta(s)\big)$、すなわち DDPG (Lillicrap et al. 2016) の決定論的アクターの損失になる。このようなエージェントの探索はどこから得なければならないか。また、この極限は問題 2b（14.9 節）とどう関係するか。
5. バッファに入れる前にすべての報酬を 10 倍し、報告するエピソード収益はスケーリングしないままとしたうえで、固定の $\beta = 0.2$ で 1 シード、学習される温度で 1 シードを訓練せよ。小問 c を用いて、1 つ目の実行のエントロピーと 2 つ目の実行の最終的な β が 15.5.4.2 節の実行と比べてどうなるかを予測し、実際に確認せよ。Haarnoja, Zhou, Abbeel, et al. (2018) はその図 3(b) において、固定温度を用いた SAC の報酬スケールに対する感度を示している。

3. [code] **スクワッシュされた方策のエントロピー**。$u \sim \mathcal{N}(\mu, \sigma^2)$、$a = c\tanh u$（$c = 2$）とし、`SquashedGaussianPolicy` と同じ設定を考える。

1. Equation 15.44 を用いて次式を示せ。

$$H(\pi) = \tfrac{1}{2}\log(2\pi e\sigma^2) + \mathbb{E}_u\big[\log(1 - \tanh^2 u)\big] + \log c. \tag{15.49}$$

$\mu = 0$ において、σ の増大とともに大きくなるのはどの項であり、小さくなるのはどの項か。

2. H を $\mu = 0$、$\sigma \in \{0.1, 0.3, 1, 3\}$ において、u に関する期待値として（`log_prob_u` を用いたモンテカルロ法、または u に関する数値積分によって）評価せよ。どの σ の

近くで H はピークをとるか。このピークを $\log 2c$（区間 $[-c, c]$ 上の一様分布のエントロピーであり、この区間上のどの密度もこれを超えない）と比較せよ。

3. Pendulum 向けに新しく初期化した `SquashedGaussianPolicy` を構築せよ。そのエントロピーを `sample` を用いたモンテカルロ法により、一様ランダムなトルクのもとで訪問した 1,000 個の状態（`train_sac` のウォームアップと同様）にわたって平均して推定せよ。その推定値は、小問 b における $\mu = 0$、$\sigma = 1$ のときの値にどれほど近くなるか。これらの状態に対する初期化直後のネットワークのどの出力が、その差を説明するか。もしアクター損失がその項 $\beta \log \pi_\theta$ のみを残していたとしたら、15.5.4.2 節のエントロピーの推移はどの値に近づくか。

Berkeley CS 285 の Assignment 3 第 3.3 節を改変。

4. **重点サンプリング比なしのオフポリシー学習。** `train_sac` のバッファは、一様ランダムなウォームアップの行動 1,000 個を含め、実行中のすべての遷移を保持する一方、Equation 15.41 と Equation 15.43 は現在の方策から新たな行動をサンプリングする。

   1. `sac_step` の変種として、$\tilde{a}'$ をデータ収集方策が s' で次にとった行動に置き換えるとする。このときクリティックはどの方策の価値を推定し、そのような行動における項 $-\beta \log \pi_\theta(a' \mid s')$ は何を測っているか。どの比を用いればターゲットを補正できるか。また、このバッファからはその比を計算できないのはなぜか。
   2. どちらの損失も、現在の方策からではなくバッファからサンプリングされた状態にわたって平均をとる。15.5.1.4 節の命題はすべての状態で方策を改善する。状態ごとの厳密な最大化においては状態の重み付けが問題にならないのはなぜか。一方、表現力に限りのあるアクターネットワークにおいてはそれが問題になるのはなぜか。

5. [code] **ヤコビ行列の復元力。●** アクター損失 Equation 15.43 において、項 $\beta \log \pi_\theta$ は行動の各座標について $-\beta \log(1 - \tanh^2 u)$ を含む。ここで $u = \mu_\theta(s) + \sigma_\theta(s) z$ である。

   1. Equation 15.45 を用いて、この項が損失の勾配に $2\beta \tanh(u)\, \partial u / \partial \theta$ だけ寄与することを示せ。これは u をどの方向へ動かし、その押し戻す力は最大でどれほど強くなるか。
   2. float32 の自動微分を用いて、u に関する導関数を安定な形および保護付きの式 `log(1 - tanh(u)**2 + 1e-6)` について $u \in \{3, 5, 7, 10, 20\}$ で評価し、両者を $-2 \tanh u$ と比較せよ。どの $|u|$ を超えると、保護付きの式で訓練されたアクターは小問 a の押し戻す力を失うか。
   3. `log_prob_u` から `logdet` を削除せよ。残ったエントロピー項が損失を変化させる割合は $-\beta/\sigma$（σ の 1 単位あたり）であり、μ に依存しないことを示せ。損失のクリティック項が σ の増大を抑えられなくなるのは、いったん u が飽和したあとではなぜか。σ がクランプ上限の e^2 に達したとき、エントロピーの推移はどの値を報告するか。またその値は問題 3b の上界と比べてどうなるか。
   4. この項を削除した状態で 1 シードを再訓練せよ。収益曲線、エントロピーの推移、および変更後の密度に適用した `squashed_logp` の数値積分による検証結果を記録せよ。3 つの測定結果のうち、どれがこの誤りを暴くか。

6. [code] **更新比。** 15.5.4.1 節の実行では、1 環境ステップあたり 1 回および 10 回の更新を行った。

   1. ツインクリティックと単一クリティックのそれぞれについて、`utd=20` で 3 つのシードを 3,000 環境ステップ訓練せよ。水準への到達は 1 ステップあたり 10 回の更新より早くなるか。`warmup=200` でも同様に繰り返せ。
   2. 小問 a のエージェントを `calibration` で測定せよ。単一クリティックの乖離は、1 ステップあたり 10 回から 20 回の更新に増やすと大きくなるか。ツインクリティックの最小値は実現収益以下に留まるか。

3. 1 ステップあたり 1 回、10 回、20 回の更新のそれぞれについて、各実行が -200 を超えるまでに要する勾配ステップ数を数えよ。高い更新比は環境ステップと引き換えに何を差し出しており、その交換がどこまで通用するかを制限するものは何か。

7. [extended] **離散ソフトアクタークリティック**。●有限の行動集合では、方策ネットワークはソフトマックスを通じて確率 $\pi_\theta(\cdot \mid s)$ を出力し、各クリティックは 15.4 節の Q ネットワークと同様に行動ごとに 1 つの価値を出力する (Christodoulou 2019)。

   1. $\tilde{a}'$ と $\tilde{a}$ に関する期待値を行動全体にわたる和で厳密に計算する形で、クリティックのターゲットとアクター損失を書き下せ。`SquashedGaussianPolicy` のどの部分が不要になるか。また、アクター勾配に行動に関して微分可能なクリティックが不要になるのはなぜか。
   2. 15.5.3.2 節の自動温度調整における目標値 $\bar{H} = -\dim \mathcal{A}$ が離散方策に対して意味をなさないのはなぜか。また、$|\mathcal{A}|$ 個の行動に対する目標値はどの区間になければならないか。
   3. ツインクリティックと Polyak ターゲットを用いて CartPole 上にこの手法を実装し、同じステップ予算とシードで環境ステップに対する収益を 15.4 節の `train_dqn` と比較せよ。

15.6 学習されたモデルと計画

ソフトアクタークリティックは数千回の遷移を経て振り子を振り上げたが、どの遷移も同じように扱っていた。すなわち、報酬に次の状態の割引価値を足したものを、クリティックの 1 回の回帰におけるターゲットとして使うだけである。しかし、1 回の遷移にはそれ以上の情報が記録されている。ある特定のトルクのもとで振り子がどう動いたかが示されており、その運動の背後にある物理法則は何度訪れても同じである。ブランコで数回背中を押してもらった子どもなら、少し後にブランコがどこへ行くか、もっと高く漕ぐにはどれくらい強く押せばよいかがわかるだろう。では、エージェントはクリティックも方策も持たずに、数百ステップのデータから振り子の物理法則を学習し、計画を立てて振り子を振り上げられるだろうか。

14.3 節では、湖の環境でこれを実行した。そこでは観測した遷移の回数を数えて推定モデルを作り、その推定値を正確なものとみなして価値反復法（value iteration）を適用した。14.6 節では、Pendulum の連続状態上で、既知の物理法則を iLQR で微分し、移動ホライズン（receding-horizon）方式のモデル予測制御（model predictive control、MPC）によって毎ステップ再計画した。物理法則が未知の場合、Pendulum の連続状態は数え上げられないため、数え上げの代わりに回帰を用いる。すなわち、現在の状態とトルクから次の状態と報酬を予測するようネットワークを学習させる。そして価値反復法や iLQR の代わりに移動ホライズン探索を用いる。エージェントは学習されたモデルの中で多数のトルク系列を試し、最良の系列の最初のトルクを実行する。Sutton の Dyna アーキテクチャは 1990 年に学習されたモデルと計画を組み合わせ (Sutton 1990)、この方法で探索されるニューラルネットワークモデルはわずか数分の経験からシミュレーション上のロボットを制御している (Nagabandi et al. 2018)。

モデルがあれば、エージェントは経験の代わりに計算を使えるようになる。1 回の実ステップの後に、何千回もの想像上のステップを続けられるからである。また、方策が提案する行動から探索を始め、その結果を方策へと教え戻すこともできる。AlphaZero と MuZero はこの方法で人間を超える強さのボードゲームプレイを学習し (Silver et al. 2018; Schrittwieser et al. 2020)、学習された方策に導かれる探索は、言語モデルがテスト時に追加の計算量を投入する一手法でもある。本節では、1 回のランダムなエピソードから Pendulum のモデルを当てはめ、計画に沿ってその誤差がどう増大するかを測定し、そのモデルで計画を立て、さらに探索をより良い方策へと変換する。本節全体を通じて常に問われるのは、データからどれだけ離れたところまでモデルを信頼できるかという点である。

```
# Deterministic XLA kernels. On an accelerator XLA otherwise picks each
# matrix product's kernel by timing candidates, and kernels that add in
# different orders differ in the last bits; training runs at large step
# sizes can turn such differences into different outcomes. JAX reads
# XLA_FLAGS once, when it starts, so this cell runs before the imports.
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
import gymnasium as gym
```

15.6.1 データからのダイナミクスモデル

15.6.1.1 数え上げに代わる回帰

Pendulum（`Pendulum-v1`）は、14.6 節で物理法則が既知の問題として導入され、14.9 節と 15.5 節でモデルフリーに扱った連続制御ベンチマークである。モーター付きの支点から振り子が吊り下げられており、状態は $s = (\cos\vartheta, \sin\vartheta, \dot{\vartheta})$（$\vartheta$ は直立位置からの角度）、行動はトルク $a \in [-2, 2]$ である。各ステップでは報酬 $-(\vartheta^2 + 0.1\,\dot{\vartheta}^2 + 0.001\,a^2)$ が与えられ（ただし ϑ は $[-\pi, \pi]$ に折り返される）、1 エピソードは 200 ステップ続く。真下で静止したままの振り子は 1 ステップあたり $\pi^2 \approx 9.9$、1 エピソードあたり約 $2{,}000$ を失う。ランダムなトルクでは約 900〜$1{,}700$ を失う。一方、振り子を振り上げてバランスを保つコントローラは、-200 という 14.6 節および 14.9 節の基準線を上回り、開始状態に応じて失点を 0 から数百の間に抑える。

環境の**モデル**とは、環境がどのように振る舞うかを予測するものである。状態と行動から予測される次の状態と報酬への写像を表すネットワークを $\hat{f}_\phi$（パラメータは ϕ）と書くことにすると、次式となる。

$$(\hat{s}', \hat{r}) = \hat{f}_\phi(s, a). \tag{15.50}$$

湖の環境ではモデルは頻度の表であり、各要素はその状態行動対（state-action pair）への訪問回数のみから推定されていた。これに対し、ネットワークは状態間でパラメータを共有するため、ある角度での遷移から近傍の角度についても学習できる。保存された遷移 (s_i, a_i, r_i, s_i') に対して、最小二乗法でこのネットワークを次のように当てはめる。

$$\min_{\phi} \sum_{i} \left\| \hat{f}_\phi(s_i, a_i) - (s_i', r_i) \right\|^2. \tag{15.51}$$

この当てはめは、保存されたすべての遷移について、記録された次の状態と報酬を再現するようモデルに求めるものである。もしネットワークの代わりに表を使えば、各状態行動対における最小点はそこで観測された結果の平均、すなわち 14.3 節の数え上げモデルをその平均へと縮約したものになる。Pendulum は決定論的であるため、結果の平均がそのまま結果そのものとなる。確率的な環境であれば、湖の環境で訪問回数が与えていたように、次の状態の分布全体のモデルが必要になるだろう。

2 つの工夫によってこの回帰は容易になる。第 1 に、ネットワークは状態の変化量 $s' - s$ を、s' そのものではなく予測対象とする。0.05 秒の 1 ステップでは振り子はわずかしか動かず、変化量こそが学習に値する部分だからである。第 2 に、入力とターゲットのすべての座標を、保存データ上で平均 0・分散 1 に標準化する。余弦と正弦は $[-1, 1]$、角速度は $[-8, 8]$、報酬はおおよそ

$[-16, 0]$ の範囲にあり、座標ごとにスケールが異なるためである。標準化しなければ、最小二乗法はもっぱら報酬ばかりに当てはめようとしてしまう。

まず本節全体で用いる設定を定義する。トルクの上限 $c = 2$、計画の深さ $n = 25$、探索で試す候補計画の数、モデルの 1 回の当てはめあたりの Adam ステップ数、シード数、そしてテストの初期状態を固定するシードである。

```
c, horizon, num_cand = 2.0, 25, 500   # torque bound, depth n, candidates
fit_steps, num_seeds, test_seed = 1_000, 5, 1_000
```

データはエピソード単位で収集する。`run_episode` は、Pendulum の複数のコピーを同期して進める gymnasium のコンテナである**ベクトル環境**（vector environment）内で行動するため、1 回の呼び出しで 10 個の初期状態から同時にコントローラを実行できる。コントローラは観測のバッチから行動のバッチへの任意の関数であり、`random_controller` は一様ランダムなトルクを抽出するコントローラを返す。`score` は環境シード 1000〜1009 に対応する 10 個の固定された初期状態からコントローラを 1 エピソード実行し、平均収益を返す。これが本節のすべてのコントローラに対するテストとなる。遷移は `d2l.ReplayBufferC`、すなわち 15.5 節のリプレイバッファで行動列を浮動小数点にしたものに格納する。ここでは単に配列の保管場所として使うだけであり、そこからバッチをサンプリングすることはない。

```
def run_episode(envs, controller, buffer=None, seed=None, num_steps=200):
    """One episode in every environment of a vector env, acting by
    controller(obs) -> actions. Returns the returns; records transitions."""
    obs, total = envs.reset(seed=seed)[0], 0.0
    for _ in range(num_steps):
        act = controller(obs)
        next_obs, rew, term, _, _ = envs.step(act)
        for i in range(envs.num_envs if buffer is not None else 0):
            buffer.add(obs[i], act[i], rew[i], next_obs[i], float(term[i]))
        obs, total = next_obs, total + rew
    return total

def random_controller(rng, scale=c):
    """Uniformly random torques: the data of the first episode."""
    return lambda obs: rng.uniform(-scale, scale, (len(obs), 1)).astype(
        np.float32)

def score(controller, num_starts=10, seed=test_seed):
    """Mean return of a controller over ten fixed Pendulum starts."""
    envs = gym.make_vec('Pendulum-v1', num_envs=num_starts,
                        vectorization_mode='sync')
    return run_episode(envs, controller, seed=seed).mean()
```

`model_data` は、保存された遷移を回帰の入力 (s, a) とターゲット $(s' - s, r)$ に変換し、あわせてそれらの平均と標準偏差を返す。

```
def model_data(buffer):
    """Inputs (s, a) and targets (s' - s, r) of the stored transitions,
    with their means and standard deviations."""
    n = buffer.size
    x = np.concatenate([buffer.obs[:n], buffer.act[:n]], -1)
```

```
    y = np.concatenate([buffer.next_obs[:n] - buffer.obs[:n],
                        buffer.rew[:n, None]], -1)
    return x, y, (x.mean(0), x.std(0) + 1e-6, y.mean(0), y.std(0) + 1e-6)
```

DynamicsModel は、64 ユニットの隠れ層を 2 つ持ち、滑らかな SiLU 活性化関数を用いた多層パーセプトロン（multilayer perceptron、MLP）である（4.1 節）。このモデルはパラメータと並んで標準化統計量を自身の状態の一部として保持しており、順伝播では入力を標準化し、出力の標準化を元に戻したうえで、予測された変化量を状態に加算する。fit_model は保存された全データから統計量を再計算し、256 個の遷移からなるミニバッチ上で Adam ステップを実行する。そのオプティマイザは make_model 内でモデルと同時に 1 度だけ生成されるため、以降の当てはめは最初からやり直すのではなく前回のパラメータから継続される。

```
class DynamicsModel(nnx.Module):
    """(s', r) from (s, a): an MLP that maps standardized inputs to the
    standardized change of state and reward."""
    def __init__(self, obs_dim=3, act_dim=1, hidden=64, rngs=None):
        d_in, d_out = obs_dim + act_dim, obs_dim + 1
        self.net = nnx.Sequential(
            d2l.Linear(d_in, hidden, rngs=rngs), jax.nn.silu,
            d2l.Linear(hidden, hidden, rngs=rngs), jax.nn.silu,
            d2l.Linear(hidden, d_out, rngs=rngs))
        self.stats = nnx.Variable((jnp.zeros(d_in), jnp.ones(d_in),
                                   jnp.zeros(d_out), jnp.ones(d_out)))

    def __call__(self, obs, act):
        mu_x, sd_x, mu_y, sd_y = self.stats.get_value()
        x = (jnp.concatenate([obs, act], -1) - mu_x) / sd_x
        y = self.net(x) * sd_y + mu_y          # (s' - s, r) in data units
        return obs + y[..., :-1], y[..., -1]

def make_model(seed, lr=1e-3):
    model = DynamicsModel(rngs=nnx.Rngs(seed))
    return model, nnx.Optimizer(model, optax.adam(lr), wrt=nnx.Param)

@nnx.jit
def model_step(model, opt, x, y):
    loss = lambda m: ((m.net(x) - y) ** 2).mean()
    value, grads = nnx.value_and_grad(loss)(model)
    opt.update(model, grads)
    return value

def fit_model(model, opt, buffer, rng, num_steps=fit_steps, batch_size=256):
    """Least squares on all stored transitions, in standardized units."""
    x, y, stats = model_data(buffer)
    model.stats.set_value(tuple(jnp.asarray(v) for v in stats))
    mu_x, sd_x, mu_y, sd_y = stats
    x, y = (x - mu_x) / sd_x, (y - mu_y) / sd_y
    step = nnx.cached_partial(model_step, model, opt)
    for _ in range(num_steps):
        i = rng.integers(len(x), size=batch_size)
        loss = step(x[i], y[i])
    return float(loss)
```

各層は d2l.Linear であり、これら 2 つの章のすべての JAX ネットワークで用いられる初期化を備えている（14.9 節）。統計量は nnx.Variable 内に保持されるため、以下のコンパイル済み関数は再当てはめのたびに最新の値を読み取る。model_step はコンパイル済みの 1 回の Adam ステップであり、nnx.cached_partial が当てはめごとに 1 度だけモデルとオプティマイザをそこに束縛する。

シード 0 からランダムなトルクによる 1 エピソードを収集し、その 200 個の遷移にモデルを当てはめる。

```
env = gym.make_vec('Pendulum-v1', num_envs=1, vectorization_mode='sync')
rng = np.random.default_rng(0)
buffer = d2l.ReplayBufferC(200, 3, 1)
print(f'return of the random episode: '
      f'{run_episode(env, random_controller(rng), buffer, seed=0)[0]:.0f}')
model, opt = make_model(0)
print(f'final minibatch loss (standardized units): '
      f'{fit_model(model, opt, buffer, rng):.4f}')
```

```
return of the random episode: -1072
final minibatch loss (standardized units): 0.0013
```

ランダムなエピソードは約 1,000 を失い、当てはめ終了時の損失はターゲットの分散（標準化単位では 1）のごく一部に収まっている。

15.6.1.2 1 ステップの手計算

訓練損失が小さいというだけでは、モデルが何を予測しているかはわからない。そこで、エピソード中の 1 つの遷移を元の問題の単位で見てみよう。rollout_model は状態と行動の系列をモデルに入力し、予測された各状態を次のステップの入力として順に送り込む。行動が 1 つだけなら、1 ステップ分の予測になる。

```
@nnx.jit
def _rollout_model(model, obs, actions):
    def step(s, a):
        s, r = model(s, a)
        return s, (s, r)
    _, (states, rewards) = jax.lax.scan(step, obs,
                                        jnp.moveaxis(actions, -2, 0))
    return jnp.moveaxis(states, 0, -2), jnp.moveaxis(rewards, 0, -1)

def rollout_model(model, obs, actions):
    """Open-loop rollout in the model: obs (..., 3) and actions (..., n, 1)
    give the predicted states (..., n, 3) and rewards (..., n), numpy in
    and out."""
    return jax.tree.map(np.asarray, _rollout_model(model, obs, actions))
```

rollout_model は jax.lax.scan を用いてコンパイル済みの _rollout_model 内で各ステップを実行する。この関数は入力の形状ごとに 1 度だけコンパイルされ、本節ではごく少数の形状でしか呼び出されない。他のタブと同様に、結果は NumPy 配列として返される。

```
def angle(s):
    """Angle from upright in degrees, from states (cos, sin, velocity)."""
```

```
    return np.degrees(np.arctan2(s[..., 1], s[..., 0]))

i = 100
pred, r_hat = rollout_model(model, buffer.obs[i], buffer.act[i][None])
print('s       ', buffer.obs[i].round(4), f'a = {buffer.act[i, 0]:.4f}')
print("true s' ", buffer.next_obs[i].round(4), f'r = {buffer.rew[i]:.4f}')
print("model s'", pred[0].round(4), f'r = {r_hat[0]:.4f}')
print(f'angle error {abs(angle(pred[0]) - angle(buffer.next_obs[i])):.3f} '
      f'degrees')
```

```
s        [-0.9407 -0.3394  5.6896] a = -0.0800
true s'  [-0.8154 -0.5789  5.4231] r = -11.0511
model s' [-0.8115 -0.5825  5.4327] r = -11.1913
angle error 0.295 degrees
```

ステップ 100 では、振り子は $\vartheta = \text{atan2}(-0.3394, -0.9407) = -2.795$ rad、すなわち直立位置から約 160 度のところにあり、トルクがほぼ 0 の状態で 5.69 rad/s で振れている。報酬は出力の通り $-(2.795^2 + 0.1 \cdot 5.690^2 + 0.001 \cdot 0.08^2) = -11.05$ である。1 ステップの 0.05 秒の間に角度は $\text{atan2}(-0.5789, -0.8154) = -144.6$ 度へと、約 $5.6 \times 0.05 = 0.28$ rad すなわち 16 度動き、モデルはこの動きを 1 度未満の誤差で予測している。報酬のずれも 0.3 未満である。1 ステップの予測は簡単である。直前に提示された状態に対する小さな補正にすぎないからである。

15.6.1.3 誤差の累積

計画には複数ステップが必要である。トルクの系列にスコアを付けるには、モデルは 25 ステップ先まで予測しなければならず、2 ステップ目以降の入力はモデル自身の直前の予測になる。モデルは真の状態のみで訓練されており、わずかにずれた自身の予測状態では一度も訓練されていないため、あるステップでの誤差が次のステップの入力を変化させる。角度の小さな誤差が重力の及ぼすトルクを変え、それが角速度を変え、さらに角度を変化させるのである。この影響を**開ループ**（open-loop）ロールアウトで測定する。すなわち、エピソードに記録された行動をたどりながらモデルに状態を予測させ、k ステップ後の予測角度を記録された実際の角度と比較する。

`kstep_error` は、エピソード終了までに 25 ステップ以上残っているすべてのステップからロールアウトを開始し、$k = 1, 5, 10, 25$ ステップ後の角度の二乗平均平方根誤差を報告する。この際、$+179$ 度と -179 度の差が 2 度となるように角度を折り返して計算する。

```
def kstep_error(model, buffer, first=0, ks=(1, 5, 10, 25), T=200):
    """RMS angle error (degrees) of open-loop model rollouts along the
    recorded actions, from every step of the stored T-step episodes (rows
    first onward) that leaves max(ks) steps before the episode ends."""
    n = max(ks)
    ep = lambda x: x[first:buffer.size].reshape(-1, T, *x.shape[1:])
    obs, act, nxt = ep(buffer.obs), ep(buffer.act), ep(buffer.next_obs)
    windows = lambda x: np.stack([x[:, t:t + n]
                                  for t in range(T - n + 1)], 1)
    pred = rollout_model(model, obs[:, :T - n + 1], windows(act))[0]
    d = np.radians(angle(pred) - angle(windows(nxt)))
    d = np.arctan2(np.sin(d), np.cos(d))          # wrapped to (-pi, pi]
    return [np.degrees(np.sqrt(np.mean(d[..., k - 1] ** 2))) for k in ks]
```

5 つのシードのそれぞれについて、1 つのランダムなエピソードにモデルを当てはめ、同じシードから生成した 2 つ目のランダムなエピソードでテストする。

```
models = []
print('seed   k=1    k=5   k=10   k=25   (degrees, held-out random episode)')
for seed in range(num_seeds):
    rng = np.random.default_rng(seed)
    train, test = d2l.ReplayBufferC(200, 3, 1), d2l.ReplayBufferC(200, 3, 1)
    run_episode(env, random_controller(rng), train, seed=seed)
    run_episode(env, random_controller(rng), test)
    m, o = make_model(seed)
    fit_model(m, o, train, rng)
    models.append((m, test))
    errors = kstep_error(m, test)
    print(f'{seed:4d} ' + ' '.join(f'{e:6.2f}' for e in errors))
```

```
seed   k=1    k=5   k=10   k=25   (degrees, held-out random episode)
   0   0.22   0.82   1.15   3.33
   1   1.40   4.83   6.87  15.09
   2   0.61   2.75   4.12   4.02
   3   0.44   1.61   2.56   6.04
   4   0.03   0.11   0.19   0.30
```

どのシードでも、10 ステップ後の誤差は 1 ステップ後の誤差の数倍になっており、1 ステップ後の誤差はほとんどのシードで 1 度未満である。25 ステップ後の誤差はシードによって異なるが、これらのホールドアウトされたエピソードでは約 20 度未満に収まっている。ランダムなトルクのもとでは、振り子は訓練エピソードが訪れたのと同じ真下付近で揺れ続けるからである。この但し書きには、プランナが振り子を別の場所へ動かした後で改めて立ち返る。

計画の深さは、n（n ステップ収益（15.1 節）のステップ数）と同様の調整ダイヤルである。計画を深くすればトルクのもたらすより先の結果まで見通せる。振り上げには数十ステップを要するため、振り子が持ち上がる前に終わる計画では、うまく勢いをつける系列とそうでない系列を区別できない。しかし、ステップを先へ延ばすたびにモデルの誤差が予測に加わっていく。本節では $n = 25$ ステップ（1.25 秒）を用い、演習問題 1 でこのダイヤルを動かして調べる。

15.6.2 モデル内での計画

15.6.2.1 ランダムシューティング

モデルの中であれば、トルクの系列を試すコストは数回の行列積だけであり、実ステップは一切かからない。最も単純な探索はこれをそのまま利用する。振り子が真下で静止しているとする。25 個のトルクからなるランダムな系列を 5 本引き、先ほど当てはめたモデル内でそれぞれをロールアウトし、予測される報酬を足し合わせる。

```
rng = np.random.default_rng(0)
s = np.array([[-1.0, 0.0, 0.0]], np.float32)      # hanging down, at rest
A = rng.uniform(-c, c, (1, 5, horizon, 1)).astype(np.float32)
_, r_hat = rollout_model(model, np.repeat(s[:, None], 5, axis=1), A)
real = gym.make('Pendulum-v1').unwrapped          # the same sequences, for real
for a, ret in zip(A[0], r_hat[0].sum(-1)):
    real.reset()
    real.state = np.array([np.pi, 0.0])           # angle, velocity
    true = sum(real.step(a_t)[1] for a_t in a)
    print(f'first actions {a[:3, 0].round(2)}  predicted {ret:7.1f}  '
          f'true {true:7.1f}')
```

```
first actions [ 0.55 -0.92 -1.84]  predicted  -185.7  true  -237.6
first actions [-0.47  1.99  1.92]  predicted  -184.5  true  -232.7
first actions [ 1.15 -1.04  1.51]  predicted  -183.7  true  -229.4
first actions [-0.3   0.48  1.98]  predicted  -182.1  true  -223.2
first actions [-0.08 -1.07  1.21]  predicted  -185.5  true  -242.4
```

25 ステップの間静止したままでいると約 $25 \times 9.9 \approx 247$ を失うが、5 つの予測はいずれもそれより大幅に高く、互いに数ポイント以内の差に収まっている。最後の列は、同じ状態から実環境で同じ 5 本の系列を再生した結果である。失点は 223〜242 の間にとどまっている。振り子を真下から少し揺り動かすことで静止状態に比べて節約できるのは 5〜24 ポイントであり、モデルが予測するような 60〜80 ポイントではない。モデルはどの系列に対しても 40〜70 ポイント楽観的すぎる。原因として考えられるのはカバレッジである。ランダムなエピソードでは振り子が真下を勢いよく通過していたため、モデルは真下でほぼ静止している状態を一度も観測しておらず、その状態に対する報酬を高く見積もりすぎているのである。それでもなお、予測収益が最も高い系列は実環境でも 5 本の中で最良であり、候補の中から選ぶのに必要なのはそれらの値そのものではなくランキングだけである。5 本の中からならその系列を選び、最初のトルクを適用することになる。モデルをどこまで信頼できるかという問いは、この最小の例にすでに現れている。ここでは予測の値は誤っているが、選択は正しい。

これが**ランダムシューティング**である。多数の候補系列 $(a_0^{(j)}, \dots, a_{n-1}^{(j)})$ を引き、それぞれの収益を次式で予測する。

$$\hat{G}^{(j)} = \sum_{t=0}^{n-1} \hat{r}_t^{(j)},$$

ここで $\hat{r}_t^{(j)}$ はステップ t における候補 j に対するモデルの予測報酬であり、その中から最良の系列を残す。25 ステップで $\gamma = 0.99$ としても割引によって重みが変わるのは高々約 1/5 にすぎないため、この和には割引を入れない。実行するのは最良の系列の最初のトルクだけである。次のステップでは、エージェントは実際の状態を観測し、そこから再び計画を立てる。これは 14.6 節の移動ホライズン方式の**モデル予測制御**（MPC）と同じ枠組みである。そこでは振り子の既知の運動方程式を用いて計画を立てたが、ここでは学習されたモデル $\hat{f}_\phi$ の中で探索を実行する。毎ステップ再計画することで、実行される軌跡に沿ってモデルの誤差が蓄積するのを防げる。誤差が累積する前に、次の実際の観測が予測を置き換えるからである。ただし、計画の後半ステップにおける誤差は、どの最初のトルクが選ばれるかに依然として影響する。

`plan` は B 個の状態のバッチに対してこれを一括で実行する。各状態に対する与えられた計画にガウスノイズを加えて 500 個の候補へと摂動させ（与えられた計画そのものも候補の 1 つとして残す）、トルクを $[-c, c]$ にクリップしたうえで、全候補を `rollout_model` でロールアウトし、各状態の最良の系列とその予測収益を返す。元の計画自身が候補に含まれているため、返される計画の予測収益が開始時の計画を下回ることは決してない。1 回の行動につき、モデルの評価が $500 \times 25 = 12{,}500$ 回必要になる。

```
def plan(model, obs, init, rng, noise=c, num_cand=num_cand, scale=c):
    """Random shooting around a plan for each of B states: obs (B, 3), init
    (B, n, 1). Each plan is perturbed into num_cand candidates (the plan
    itself is one), all are rolled out in the model, and the candidate with
    the highest predicted return is returned with that return."""
    B = len(obs)
    A = init[:, None] + noise * rng.standard_normal(
        (B, num_cand) + init.shape[1:], dtype=np.float32)
```

```
        A[:, 0] = init
        A = np.clip(A, -scale, scale)
        s = np.repeat(obs[:, None], num_cand, axis=1)          # (B, num_cand, 3)
        ret = rollout_model(model, s, A)[1].sum(-1)            # (B, num_cand)
        best = ret.argmax(1)
        return A[np.arange(B), best], ret[np.arange(B), best]
```

15.6.2.2 ウォームスタート

探索はどの計画から始めるべきだろうか。事前知識がまったくなければ、毎ステップ新たにランダムな系列を引くことになる。しかしそれでは、1 ステップ前に見つけた（おそらく良質な）計画を捨ててしまい、毎ステップ 500 本のランダムな系列の中から振り上げの計画を再び見つけ出さなければならなくなる。結果として、連続するトルクが互いに無関係な計画から選ばれることになる。**ウォームスタート**は前回の計画を再利用する。その最初のトルクは直前に実行済みであるため、残りの 24 個を 1 ステップ前にシフトし、末尾に 0 のトルクを追加して、このシフトした計画を探索で摂動させる。こうすれば、良い計画を一度見つけた後はステップごとにそれを洗練していける。Figure 15.15 にこの探索の 1 ステップを示す。

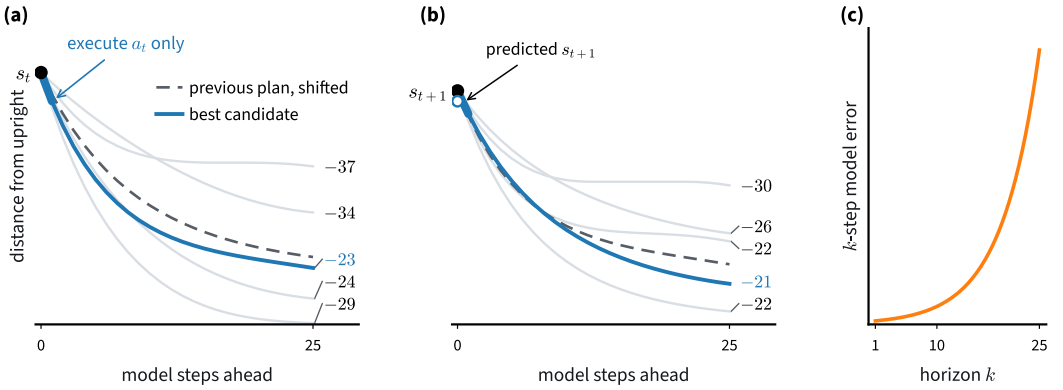


Figure 15.15: ウォームスタート付きランダムシューティングによる学習されたモデル内での計画。(a) 状態 s_t において、1 ステップ分シフトした前回の計画（破線）の周囲から引いた候補行動系列を $\hat{f}_\phi$ で 25 ステップロールアウトし、予測報酬の和（右側の数値）によってスコア付けする。最良の系列（青）の最初の行動のみを実行する。(b) 実際の次の状態は予測された状態（白丸）からわずかにずれ、プランナはそこから (a) の最良の計画を 1 ステップ分シフトしたものの周囲で再び計画を始める。(c) モデルの開ループ誤差はホライズンとともに増大するため、これが毎ステップ再計画する理由の 1 つとなる。縦軸とスコアは Pendulum ではなく模式的な 1 次元モデルのものである。スコアは速度やトルクのコストを含む全ステップの報酬を足し合わせるため、直立位置に最も近いところで終わる系列のスコアが最高になるとは限らない。

`mpc` はモデルからこのようなコントローラを構築する。計画は環境ごとに 1 つずつコントローラのクロージャ内に保持され、最初の呼び出し時に生成される。摂動の標準偏差はトルクの上限 $c = 2$ に等しい。別の 10 個の初期状態セットでスコアを評価する 5 シードの独立した実験で、毎ステップ 500 本の新たな一様ランダム系列を引くプランナと比較した。600 実ステップ後、ウォームスタート付きプランナのスコアはすべての実行で約 -115〜-155 の間に入ったのに対し、毎回新たに引くプランナでは約 -165〜-555 であった。1,000 実ステップ後でも、毎回新たに引くプランナはすべてのシードで依然として 40〜110 ポイント劣っていた。

```
def mpc(model, rng, horizon=horizon, noise=c):
    """Model-predictive control: at every step plan in the model, starting
    from the previous plan shifted by one step, and execute its first
    action."""
    plans = None
    def controller(obs):
        nonlocal plans
        if plans is None:
            plans = np.zeros((len(obs), horizon, 1), np.float32)
        shifted = np.concatenate([plans[:, 1:], np.zeros_like(plans[:, :1])],
                                 1)
        plans = plan(model, obs, shifted, rng, noise)[0]
        return plans[:, 0]
    return controller
```

15.6.2.3 数エピソードからの振り上げ

プランナには何回の実ステップが必要だろうか。mpc_run は 1 回のランダムなエピソードから始める。続いて、それまでの全データでモデルを再当てはめし、10 個のテスト初期状態でプランナのスコアを評価し、プランナで 1 エピソードを実行してデータを追加収集するという手順を 3 回繰り返した後、最後の再当てはめをしてスコアを評価する。したがって、得られるスコアは 200、400、600、800 実ステップから構築されたプランナを測定したものになる。

```
def mpc_run(seed, num_rounds=3):
    """One random episode, then rounds of: refit the model on all data,
    score the planner on the ten test starts, collect one MPC episode."""
    env = gym.make_vec('Pendulum-v1', num_envs=1, vectorization_mode='sync')
    rng, (model, opt) = np.random.default_rng(seed), make_model(seed)
    buffer = d2l.ReplayBufferC(200 * (num_rounds + 1), 3, 1)
    run_episode(env, random_controller(rng), buffer, seed=seed)
    scores = []
    for r in range(num_rounds + 1):
        fit_model(model, opt, buffer, rng)
        scores.append(score(mpc(model, rng)))
        if r < num_rounds:
            run_episode(env, mpc(model, rng), buffer)
    return scores, buffer
```

5 つのシードについて、収集した 4 エピソードの収益と 4 回のテストスコアの出力は次の通りである。

```
runs = [mpc_run(seed) for seed in range(num_seeds)]
print('seed  episode returns: random, MPC x 3 |'
      '  test score at 200/400/600/800 steps')
for seed, (scores, buf) in enumerate(runs):
    rets = buf.rew[:buf.size].reshape(-1, 200).sum(1)
    print(f'{seed:4d} ' + ''.join(f'{r:7.0f}' for r in rets) + '  | '
          + ''.join(f'{s:7.0f}' for s in scores))
```

```
seed  episode returns: random, MPC x 3 |  test score at 200/400/600/800 steps
   0   -1072   -236   -119   -125  |    -174   -169   -166   -170
```

```
   1    -876   -267   -120   -123  |    -215   -192   -214   -179
   2    -966   -124   -125   -116  |    -179   -182   -205   -169
   3   -1501  -1541  -1273     -2  |   -1550  -1393   -178   -170
   4   -1703  -1471   -936   -125  |   -1512   -649   -171   -175
```

どの程度であれば良いスコアといえるだろうか。10 個のテスト初期状態のうち、振り上げてバランスを保つコントローラは、真上に最も近い初期状態からはほとんど失点せず、真下付近の 5 つの初期状態からはそれぞれ約 250〜320、平均すると約 170 を失う。15.5 節のソフトアクタークリティックについて、各エピソード後にその貪欲アクターのスコアを同じ方法で評価すると、-170 付近に落ち着き、1 ステップあたり 1 回の更新では約 5,000〜7,500 ステップ後に初めて -200 に到達する。

3 エピソードにあたる 600 実ステップ後には、プランナのスコアはすべての実行で約 -165〜-215 とその水準に近づき、800 ステップ後には約 -170〜-195 となる。最初に -200 に到達するのは 200〜800 実ステップ後であり、大半の実行では 600 ステップまでに到達する。5 つのシードのうち 3 つでは、最初のランダムエピソードのモデルだけでも約 -175〜-340 のスコアを出すプランナが得られており、ランダム方策の $-1{,}000$ 以下とは大きく異なる。残る 2 つであるシード 3 とシード 4 では、最初のプランナはスコアが $-1{,}500$ 付近となって失敗し、最初の 1〜2 エピソードも失敗するが、600 実ステップ後には同じプランナが成功する。

15.6.2.4 カバレッジ

立ち上がりの遅い 2 つのシードは、ランダムエピソードでの失点が約 1,500 と 1,700 で最も大きかった 2 つであり、ランダムなトルクでは振り子が一度も水平より上まで持ち上がらなかったケースである。先ほどの誤差の表と同様に、各シードについてその 1 回のランダムエピソードに当てはめたモデルを取り出し、2 つ目のランダムエピソードではなくプランナ自身のエピソードに沿ってテストしてみよう。

```
print('seed   k=1    k=5   k=10   k=25   (degrees, the planner\'s episodes)')
for seed, ((m, _), (_, buf)) in enumerate(zip(models, runs)):
    errors = kstep_error(m, buf, first=200)      # skip the random episode
    print(f'{seed:4d} ' + ' '.join(f'{e:6.1f}' for e in errors))
```

```
seed   k=1    k=5   k=10   k=25   (degrees, the planner's episodes)
   0    0.2    1.0    3.2   56.4
   1    0.3    0.9    1.8   27.7
   2    0.2    0.9    1.8   29.3
   3    7.9   48.5   81.8  120.3
   4   10.8   56.4   78.5   87.5
```

シード 0〜2 では、プランナのエピソードに沿った誤差はランダムデータ上と同様に小さく始まり、1 ステップ後では 1 度未満、25 ステップ後には数十度へと増大する。これらのエピソードは大半の時間を直立位置の近くで過ごすが、そこでは振り子が不安定であり、予測角度のわずかな誤差がどちら側へ倒れるかを左右する。一方、シード 3 とシード 4 では、わずか 1 ステップ後で数度、5 ステップ後には数十度の誤差が生じている。モデルはデータが訪れた領域では正確だが、それ以外の領域では誤っているのである。湖の環境では、一度も試していない状態行動対は数え上げモデルには無価値に見えたが、ここではネットワークが一度も見たことのない上半円へと外挿し、その外挿が外れている。これを修復するのがプランナの失敗した試行である。振り上げようとする試行は、たとえ失敗してもランダムなトルクより高くまで振り子を運び、そこで何が起きるかを記録する。次の再当てはめではモデルがそれらの状態のデータを持つようになり、その上に構築された計画が機能する。モデルが報酬を見込む場所へとプランナが向かい、モデルが誤っていた場所はどこでも修正されるため、プランナは明示的な探索規則なしに探索を進めるので

ある。

15.6.2.5 実ステップのコスト

プランナは 200〜800 実ステップ後に初めて -200 に到達する。10 個のテスト初期状態にわたる平均という同じスコアで見ると、ソフトアクタークリティックが初めて -200 に到達するのは、環境 1 ステップあたり勾配ステップ 1 回では約 5,000〜7,500 実ステップ後、環境 1 ステップあたり勾配ステップ 10 回では約 1,600〜4,000 実ステップ後である。これらは学習される温度を用いた 15.5 節のコードを各設定 5 シードで実行した結果であり、ステップ数には最初の 1,000 ステップのランダムなウォームアップも含まれている。環境ステップ数という軸で見れば、プランナに必要なステップ数は前者の約 1/10、後者の数分の 1 で済む。

もう 1 つの軸は計算量である。プランナの行動は毎回が探索であり、モデルを 12,500 回評価する（この小さなネットワークなら CPU 1 コアで数 ms である）。さらにエピソードごとに 1,000 回の Adam ステップによる再当てはめが加わる。ソフトアクタークリティックは 1 実ステップあたり 3 つの小さなネットワークの勾配ステップを 1 回または 10 回実行し、行動選択にはアクターの順伝播が 1 回必要なだけである。実機のロボットでは、制御周期（ここでは 0.05 秒）以内に探索を終えなければならないという制約もある。モデルは学習のコストを無料にするわけではない。ロボットにとって希少な「経験」から、ロボットのコントローラにとって希少な「計算量」へとコストを移し替えるのである。

15.6.3 方策改善としての探索

15.6.3.1 アクターを初期値とする探索

これまでのプランナは方策を持たない。行動のたびに完全な探索が必要であり、見つけた結果はモデルと直前のステップからシフトした計画を介する以外には何も記憶されない。方策ネットワークがあれば 1 回の順伝播で行動でき、状態間で汎化（generalization）できるが、それには訓練が必要であり、15.5 節ではクリティックを通じて方策ネットワークを訓練するのに数千ステップを要した。では、代わりに探索を使って方策を訓練できないだろうか。

探索を方策自身の計画から始めることにしよう。アクターを、状態をトルクへ写すネットワーク $\bar{a}_\theta(s)$ とする。モデル内において、状態 s からのアクターの計画とは、アクター自身が予測される軌跡に沿って選ぶトルクの系列である。すなわち、$a_0 = \bar{a}_\theta(s)$、$\hat{s}_1$（$\hat{f}_\phi(s, a_0)$ による予測）、$a_1 = \bar{a}_\theta(\hat{s}_1)$ のように n ステップ分続ける。`actor_plan` は状態のバッチに対してこの計画を計算する。`improve` はシフトした前回の計画の代わりにこの計画を `plan` に渡し、同じ 500 個の候補と同じ摂動を用いて探索し、最良の候補の最初のトルクを返す。

```
def actor_plan(model, actor, obs, horizon=horizon):
    """The actor's own deterministic plan for each state, imagined in the
    model: (B, n, 1)."""
    acts = []
    for _ in range(horizon):
        acts.append(actor.act_greedy(obs))
        obs = rollout_model(model, obs, acts[-1][:, None])[0][:, 0]
    return np.stack(acts, 1)

def improve(model, actor, obs, rng, noise=c):
    """Search seeded by the actor: the best first action for each state."""
    return plan(model, obs, actor_plan(model, actor, obs), rng, noise)[0][:, 0]
```

アクター自身の計画も候補の 1 つに含まれているため、探索が返す行動の計画は、少なくともアクターの計画と同じかそれ以上の予測収益を持つ。モデルのもとで n ステップにわたって評価する限り、方策を初期値とする探索がその方策を下回ることは決してない。これは 14.2.3.2 節における方策反復法（policy iteration）の貪欲ステップにあたる。そこでは各状態において、現在の方策自身の価値を基準として現在の方策以上の行動を選んでいたが、ここで基準となるのは次の 25 ステップにわたるモデルの予測である。この保証が成り立つのはモデルの中だけであり、かつホライズンの範囲内に限られる。誤ったモデルはより悪い行動にお墨付きを与えかねず、25 ステップで終わる計画はそれ以降に起きることを評価できない。

15.6.3.2 アクターへの蒸留

探索がアクターを改善するのであれば、探索を模倣するようアクターを訓練できる。これは探索をエキスパートとみなした 14.5 節の行動クローニング（behavior cloning）である。すなわち、`improve` で状態にラベルを付け、トルクは連続値なので二乗誤差を用いてアクターのトルクをそのラベルへと回帰させる。アクターには `d2l.SquashedGaussianPolicy`（15.5 節）を用いる。これはガウス分布の平均を $c\tanh$ でトルクの範囲に押し込める 2 層のネットワークである。ここでは押し込められた平均 $\bar{a}_\theta(s) = c\tanh\mu_\theta(s)$（同節と同じく μ_θ はガウス平均であり、`act_greedy` が返す値）のみを訓練する。`make_actor` はアクターとその Adam オプティマイザを構築し、`distill` はこの回帰の Adam ステップを 1,000 回実行する。

```
def make_actor(seed, lr=1e-3):
    actor = d2l.SquashedGaussianPolicy(3, 1, scale=c, seed=seed)
    return actor, nnx.Optimizer(actor, optax.adam(lr), wrt=nnx.Param)

@nnx.jit
def distill_step(actor, opt, obs, labels):
    loss = lambda m: ((m.scale * jnp.tanh(m.policy(obs)[0]) - labels)
                      ** 2).mean()
    value, grads = nnx.value_and_grad(loss)(actor)
    opt.update(actor, grads)
    return value

def distill(actor, opt, obs, labels, rng, num_steps=1_000, batch_size=256):
    """Regress the actor's mean action, scale * tanh(mean), onto labels."""
    step = nnx.cached_partial(distill_step, actor, opt)
    for _ in range(num_steps):
        i = rng.integers(len(obs), size=batch_size)
        loss = step(obs[i], labels[i])
    return float(loss)
```

`distill_step` はこの回帰のコンパイル済み Adam ステップ 1 回分であり、`distill` の呼び出しごとに 1 度だけ `nnx.cached_partial` によってアクターとそのオプティマイザに束縛される（`model_step` がモデルに束縛されたのと同様である）。

`search_and_distill`（Figure 15.16）の各ラウンドでは、モデルを再当てはめし、保存された状態からランダムに抽出した 200 個に探索でラベルを付け、それらのラベルをアクターへと蒸留（distillation）し、探索を用いて行動しながら実エピソードを 1 つ収集する。アクターの訓練には現在のラウンドのラベルのみを使い、前のラウンドのラベルは保持しない。探索を用いて行動することで、各エピソードは現在のモデルとアクターが許す限り良質なものとなり、データは優れたコントローラが訪れる状態へと到達する。各状態の価値も予測するネットワーク、離散的な着手上の木探索、そして多数の対局にわたって探索のターゲットを保持するバッファを組み合わせれば、探索で行動し、ラベルを付け、蒸留するというこのループはそのまま AlphaZero のループ

になる。

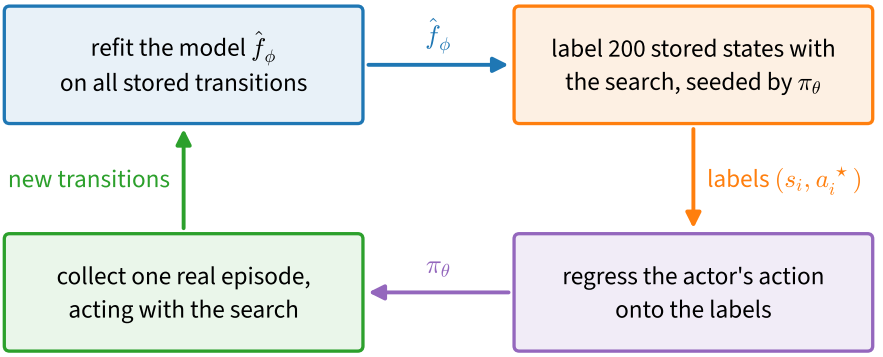


Figure 15.16: 探索と蒸留の 1 ラウンド。保存された全遷移でモデルを再当てはめし、アクターを初期値とする探索で保存された 200 個の状態にラベルを付け、それらのラベルへとアクターを回帰させ、探索を用いて行動して収集した 1 つの実エピソードが次のラウンドに新たな遷移を供給する。

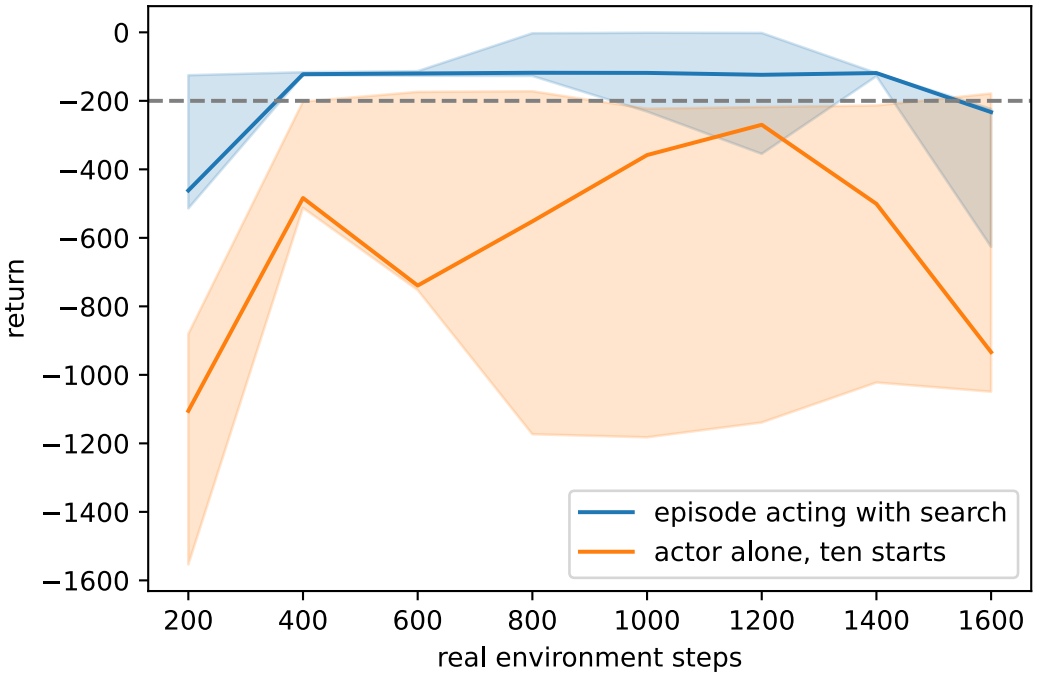
このループはラウンドごとに、探索を用いて収集したエピソードの収益と、探索を一切使わずにアクター単独で行動したときのスコアを記録し、最後に 10 個の初期状態で探索を用いて行動したときのスコアを評価する。2 本の曲線は異なるものを測っている。1 本目はラウンドごとの 1 つの訓練エピソードであり、たまたま始まった初期状態からの結果であるため、エージェントの性能と同じくらい初期状態にも左右され、難しい初期状態に当たると一時的な落ち込みとして現れる。2 本目は 10 個のテスト初期状態にわたる平均であり、本節の他のすべての箇所で用いているスコアである。

```
def search_and_distill(seed, num_rounds=8, num_labels=200):
    """One random episode, then rounds of: refit the model, label stored
    states with the search, regress the actor onto the labels, collect one
    episode acting with search. Returns per round the collected return and
    the actor's score alone, and the final score acting with search."""
    env = gym.make_vec('Pendulum-v1', num_envs=1, vectorization_mode='sync')
    rng, (model, opt) = np.random.default_rng(seed), make_model(seed)
    actor, actor_opt = make_actor(seed)
    buffer = d2l.ReplayBufferC(200 * (num_rounds + 1), 3, 1)
    run_episode(env, random_controller(rng), buffer, seed=seed)
    search = lambda obs: improve(model, actor, obs, rng)
    rounds = []
    for _ in range(num_rounds):
        fit_model(model, opt, buffer, rng)
        obs = buffer.obs[rng.choice(buffer.size, num_labels, replace=False)]
        distill(actor, actor_opt, obs, search(obs), rng)
        rounds.append([run_episode(env, search, buffer)[0],
                       score(actor.act_greedy)])
    return np.array(rounds), score(search)
```

1 回のランダムエピソードから 8 ラウンドを始めるため、ラウンド r のアクターは $200r$ 回の実ステップを経験している。3 つのシードを実行し、シード間の中央値と範囲をプロットする。

```
results = [search_and_distill(seed) for seed in range(3)]
rounds = np.array([r for r, _ in results])
d2l.plot_curves({'episode acting with search': rounds[:, :, 0],
                 'actor alone, ten starts': rounds[:, :, 1]},
                'real environment steps', 'return',
                x=200 * np.arange(1, 9), reference=-200)
print('final score on the ten starts, acting with search:',
      ' '.join(f'{s:.0f}' for _, s in results))
```

```
final score on the ten starts, acting with search: -223 -188 -241
```



15.6.3.3 探索と蒸留のラウンド

探索を用いて行動すると、エージェントは図示した 3 回の実行のいずれでも 2〜3 ラウンド以内に振り子を振り上げ、8 ラウンド後には 10 個の初期状態で約 −185〜−245 のスコアを出しており、大半の実行で先ほどのウォームスタート付きプランナに近い性能を示す。アクター単独でもすべての実行で改善し、第 1 ラウンド後の約 −900〜−1,600 から第 8 ラウンド後の約 −190〜−1,100 へと上がるが、あるラウンドから次のラウンドにかけてスコアが数百ポイント落ち込むこともある。ネットワークは探索が知っていることの一部を学習したものの、単独で行動すると力及ばない。

これらの実行だけではなぜ力及ばないのかまではわからず、3 つの原因が候補として考えられる。第 1 は、14.5 節が湖の環境で測定した分布シフト（distribution shift）である。ラベルは保存された状態から抽出されており、最初のランダムエピソードを除けば、それらは探索が訪れた状態である。探索を不完全に模倣したアクターは、単独で行動するとラベルの乏しい状態へと逸れていき、そこで誤差が累積する。DAgger（14.5.4.1 節）は湖の環境において、学習者自身の状態にラベルを付けることでこれを修復した。第 2 は訓練セットのサイズである。DAgger が全ラウンドにわたってラベルを集約し、AlphaZero が探索のターゲットをバッファに保持し続けるのに対

し、ここでは各ラウンドで新たな 200 個のラベルのみでアクターを訓練し、以前のラベルを破棄している。第 3 はラベルそのものの性質である。各ラベルは 500 個のランダム候補のうち最良のものの最初のトルクであり、ノイズを含んだターゲットとなっている。演習問題 5 では最初の 2 つの原因を切り分ける。なお AlphaZero はテスト時にも探索するため、ネットワークは探索の出発点を良くするだけでよい。そうすれば探索とネットワークが互いを高め合う。より良いアクターがより良い探索の初期値を与え、より良い探索がアクターを鍛えるのである。

15.6.3.4 データとしての想像上の遷移

モデルは、プランナを持たない学習器にデータを供給することもできる。Dyna は実ステップとモデルからサンプリングした遷移による更新とを交互に織り交ぜ (Sutton 1990)、モデルベース方策最適化（model-based policy optimization、MBPO）はこれをソフトアクタークリティックに対して実行する (Janner et al. 2019)。すなわち、数百ステップごとにモデルを再当てはめし、リプレイバッファ内の状態から現在のアクターをモデル内で数ステップだけロールアウトして、各訓練バッチの大半をこれらの想像上の遷移で埋めるのである（Figure 15.17）。想像上のデータは計算量が許す限りいくらでも生成できる。しかし、モデルの誤差がクリティックのターゲットに入り込むため、MBPO は短いロールアウトにとどめることで先ほど測定した累積誤差を抑えている。

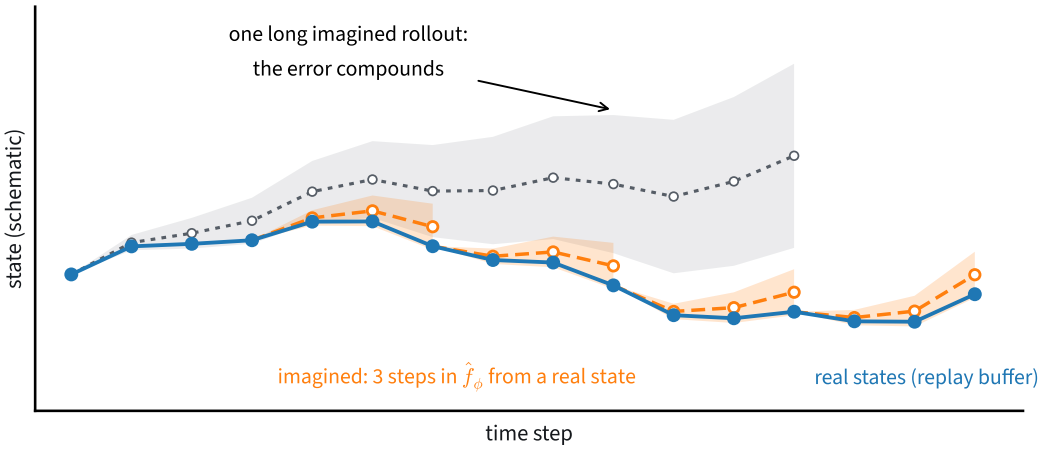


Figure 15.17: 分岐ロールアウトの模式図。短い想像上のロールアウト（オレンジ、破線、ここでは各 3 モデルステップ）はリプレイバッファ内の実際の状態（青）から始まるため、モデルの誤差（網掛けの帯）は数ステップ分しか累積しない。最初の状態からの 1 本の長い想像上のロールアウト（灰色、点線）はデータから大きく逸れていく。

これによって Pendulum で必要な実ステップ数は減るだろうか。温度を 0.2 に固定し、環境 1 ステップあたり勾配ステップを 10 回とし、3 ステップのロールアウトによる想像上の遷移 95% と実際の遷移 5% からなるバッチを用い、250 ステップごとにモデルを再当てはめする設定でソフトアクタークリティックを実行した。5 シードにわたってその貪欲アクターが初めて −200 以上のスコアを出したのは 1,500〜3,000 ステップ後であったが、モデルを使わずに同じく 1 ステップあたり 10 回更新した場合でも 1,500〜2,500 ステップであった。なお、これらの実行では別の固定された 10 個の初期状態セットを用いており、優れたコントローラの失点が本節のセットより少ないため、先ほど引用した実行よりも早く −200 に到達している。更新比率を揃えて比較すると、想像上の遷移を加えても実ステップ数は減らなかったのである。1 ステップあたり 1 回更新のソフトアクタークリティックに対して MBPO が得る改善の多くは、1 実ステップあたりの更新回数が増えたことに由来しうるものであり、それはモデルなしでもリプレイだけでまかなえる。実際、高い更新比率を用い、そのような比率が引き起こす過大評価を抑えるためにクリティック

のアンサンブルを備えたモデルフリー学習器は、MBPO に匹敵するサンプル効率を達成している (X. Chen et al. 2021)。演習問題 6 でこの比較を再現する。モデルが最も力を発揮するのは、まさに本節で使った場面、すなわち行動する前に先読みし、探索を方策へと変換するところである。不確実な領域を把握できるモデルのアンサンブル、学習された潜在状態のモデル、学習された価値で終端する探索はこれらのアイデアをさらに発展させるものであり、15.9 節で解説する。

ここまでで、数百ステップのデータからダイナミクスのモデルを当てはめ、それを通して計画を立て、探索が見つけた行動を方策に教えられるようになった。どの段階でも、自身のエピソードが訪れた範囲かつ 25 ステップ先までに限ってモデルを信頼し、実際に行動して再当てはめすることでその信頼を検証できた。続く 15.7 節では、この「行動する」ことが取り除かれる。データは固定されたログとなり、そのログの外側でモデルや価値関数が約束することは一切検証できなくなる。

15.6.4 まとめ

遷移には環境がどのように動くかが記録されており、最小二乗法で遷移に当てはめたネットワーク $\hat{f}_\phi(s,a) \to (\hat{s}', \hat{r})$ は、湖の環境における数え上げモデルを連続状態へと拡張する。その 1 ステップ誤差は小さいものの、ロールアウトに沿って累積し、データが一度も訪れていない領域では大きくなる。モデル予測制御はモデル内でトルク系列を探索し、最初のトルクを実行して次のステップで再び計画を立てる。ウォームスタートを用いれば、1〜3 エピソードから構築したプランナは、ソフトアクタークリティックが数千ステップを要した振り子の振り上げを数百実ステップで達成するが、各行動には数千回のモデル評価が必要となる。方策を初期値とする探索はその方策より低い収益を決して予測しないため方策を改善し、探索の行動へとアクターを回帰させることでその改善の一部が転移されるが、アクター単独で行動すると探索には遅れをとる。そのラベルは大半が探索の訪れた状態上にあり、ラウンドあたりの数が少なく、ノイズを含んでいるが、本節の実験だけではこの 3 つのうちどれが最も重要かまでは決まらない。また、想像上の遷移を訓練データとして用いても、1 ステップあたりの更新回数を揃えた条件ではソフトアクタークリティックの実ステップ数は減らなかった。

15.6.5 演習問題

1. [code] **計画の深さ**。プランナは $n = 25$ ステップ先まで見通す。
   1. コードを実行する前に、候補数を 500 に保ったまま $n = 10$ とした場合および $n = 50$ とした場合にテストスコアがどうなるかを、本節の 2 つの誤差の表を使って予測せよ。
   2. 両方の深さで 3 つのシードについて `mpc_run` を実行し、600 実ステップ後のスコアを $n = 25$ の場合と比較せよ。
   3. 真下からの振り上げには数十ステップにわたって勢いをつける必要がある。$n = 10$ の結果をその予測収益が区別できることとできないことから説明し、$n = 50$ の結果を誤差の表から説明せよ。1 回の行動にかかるコストは n に応じてどう変化するか。
2. [code] **ランダムデータの必要量**。シード 3 とシード 4 では、モデルが上半円をカバーするまでにプランナ自身の失敗エピソードが必要であった。
   1. 両方のシードについて、1、2、5 回のランダムエピソードにモデルを当てはめよ。データ中で振り子が到達した最高点（最大の $\cos\vartheta$）と、カバレッジの表と同様のプランナのエピソードに沿った各モデルの 25 ステップ誤差を報告せよ。
   2. ランダムデータのみから構築したプランナが、大半のテスト初期状態から振り上げるには何回のランダムエピソードが必要か。プランナ自身のエピソードが必要とした 600 ステップと比較し、その違いを説明せよ。
3. [code] **交差エントロピー法**。行動ごとにランダムシューティングを複数回繰り返し、サンプリング分布を上位の候補に再当てはめする手法を**交差エントロピー法**という (Rubinstein and Kroese 2004)。

    1. 5 ラウンドを実行するように `plan` を変更せよ。各ラウンドの後、計画の各トルクの平均と標準偏差を 500 個の候補のうち上位 50 個の平均と標準偏差に設定し、このガウス分布から次の 500 個を引くようにせよ。
    2. 600 実ステップ後のテストスコアおよび 1 行動あたりのモデル評価回数を、本節のウォームスタート付きランダムシューティングと比較せよ。
    3. ウォームスタートを使わず、毎ステップ平均 0 のトルク・標準偏差 c から始める設定でこの手法を実行せよ。1 ステップ内での洗練は、ステップをまたぐ洗練の代わりになるか。
4. [code] **信頼のシグナルとしての不一致**。●モデルのアンサンブルを使えば、どこが未知であるかを知ることができる (Chua et al. 2018)。
    1. シード 3 とシード 4 について、異なるシードを持つ 3 つのモデルをランダムエピソードに当てはめよ。プランナのエピソードに沿って、1、5、25 ステップ後における 3 つのモデルの不一致（予測角度の標準偏差）と平均予測の誤差とを比較せよ。大きな不一致は大きな誤差を予測するか。
    2. `plan` 内の各候補に対し、3 つのモデルの平均予測収益から、全ステップにわたる不一致の和の κ 倍を引いた値によってスコアを付けよ（ただし $\kappa \in \{0, 1, 10\}$ とする）。シード 3 とシード 4 の最初の MPC エピソードはどう変化するか。
    3. 不一致に対するボーナスとなる $-\kappa$ を用いて小問 b を繰り返せ。まだデータを収集できるエージェントにはどちらの符号が適しており、もはや収集できないエージェントにはどちらの符号が適しているか。
5. [code] **アクター自身の状態に対するラベル**。●本節のループは探索を用いて行動しながらエピソードを収集し、各ラウンドではそのラウンドの 200 個のラベルのみでアクターを訓練している。
    1. 探索による収集はそのまま維持しつつ、ラベルを集約せよ。すなわち、各ラウンドでラベル付けした状態とトルクを 1 つの増大する集合に追加していき、集合全体でアクターへと蒸留せよ。5 つのシードを実行し、ラウンドごとのアクターのスコアを本節の結果と比較せよ。
    2. 次に、各ラウンドのエピソードをアクター単独（`actor.act_greedy`）で収集し、アクターのエピソードの状態に `improve` でラベルを付け、(a) と同様にラベルを集約せよ。これは探索をエキスパートとした 14.5 節の DAgger である。3 つのバージョンのうち最も安定したアクターを与えるのはどれか。また、この比較から本文で挙げた原因について何が言えるか。
    3. シードによっては、DAgger 版は一度も振り子を振り上げられないことがある。そのような実行のデータを調べ、本節のカバレッジの表を用いて失敗の理由を説明せよ。2 つのデータ収集方法はどうすれば組み合わせられるか。
6. [extended] **ソフトアクタークリティックのための想像上の遷移**。●●モデルベース方策最適化は、モデルが想像した遷移上でソフトアクタークリティックを訓練する (Janner et al. 2019)。
    1. `train_sac` を 15.5 節からノートブックにコピーせよ。ウォームアップ後、250 環境ステップごとにリプレイバッファ上で `DynamicsModel` を再当てはめし、バッファから抽出した 2,000 個の状態から現在のアクターをモデル内で 3 ステップロールアウトせよ。直近 4 回の再当てはめで生成した想像上の遷移を 2 つ目の `d2l.ReplayBufferC` に保持し、各バッチの 95% をそこから抽出せよ。
    2. 環境 1 ステップあたり 10 回の更新で 5 つのシードを 4,000 ステップ実行し、250 ステップごとに 10 個のテスト初期状態で貪欲アクターのスコアを評価せよ。モデルを使わない同じループである対照群も同じシードで実行せよ。各実行が初めて -200 以上のスコアを出すステップ数を報告せよ。
    3. なぜ対照群でも環境 1 ステップあたりの更新回数を同じにしなければならないのか。両群を 1 ステップあたり 1 回の更新で繰り返し、その違いを解釈せよ。
    4. ロールアウト長を 1 ステップおよび 10 ステップに変更せよ。本節の誤差の表からは

どのような結果が予測され、実際にはどのような結果が観測されるか。

7. [code] **価値を用いた先読み**。本節のプランナは固定されたホライズンにわたって報酬を足し合わせ、それ以降のことは無視している。価値関数があれば、ホライズンを 1 ステップにできる。
   1. CartPole において、ランダムな行動による 5,000 個の遷移に 4 次元状態の `DynamicsModel` を当てはめよ。終了判定は CartPole の既知のルールに従い、ポールの角度とカートの位置から決定せよ。
   2. 15.4 節の DQN を 10,000 ステップ訓練し、$\text{argmax}_a\, Q(s,a)$ による方法と、1 ステップ先読み $\text{argmax}_a \left[\hat{r}(s,a) + \gamma \max_{a'} Q(\hat{s}'(s,a), a')\right]$ による方法の 2 通りで貪欲に行動させよ。5 つのシードについて 20 エピソードにわたる収益を比較せよ。
   3. 40,000 ステップの訓練後に同じ実験を繰り返せ。先読みが有効に働くのはどのようなときであり、モデルの誤差が先読みの利点を上回るのはどのようなときか。

15.7 オフライン強化学習

本章でここまで扱ってきた手法はすべて、実際に行動を試すことができた。Q 学習がある行動を過大評価しても、エージェントはいずれその行動を取り、結果を見て自身のテーブルを修正した。また、15.6 節のプランナは、自身のエピソードで訪れた状態の近傍でのみ学習済みモデルを信頼し、必要に応じてさらなるエピソードを収集した。ここで、新たなエピソードを一切得られない状況を考えてみよう。病院には何年分もの治療記録があるが、患者を相手に実験するわけにはいかない。企業には昨年運用していた推薦システムのログがある。ロボット工学の研究室には何時間分もの遠隔操作による実演データがある。このような固定された記録だけから方策を学習できるのだろうか。また、その方策はどこまで良いものになり得るのだろうか。

この記録は何らかの**行動方策**（behavior policy）π_e によって収集されたものだが、Q 学習のターゲットは誰がその行動を選んだかに依存しないため、記録された遷移に対して Q 学習を実行できる。難しさはその先にある。学習された方策が行動方策を上回るには、行動方策がまれにしか取らなかった行動を優先しなければならないが、そうした行動の価値こそ最も少ないサンプルに基づいている。ターゲットに含まれる最大値演算は最大の推定値を選び出すが、最大の推定値はたまたま運が良かっただけのものが多く、それを検証する新たな試行は二度と行われない。**悲観主義**（pessimism）と呼ばれる対策では、根拠となるデータが少ない推定値ほど割り引いて疑う。これは 14.4 節で探索の原動力となった楽観主義とちょうど鏡像の関係にある。

本節では、厳密な最適解が既知である 14.2 節の滑りやすい湖を題材とし、学習されたテーブルが予測するすべての価値を真の値と照合できるようにする。まず、他の方策のデータを利用できる更新則はそもそもどれかを問う。次に、2 種類の固定ログから学習し、どの方策が獲得し得る値よりも予測値がどれだけ上回るかを測定した上で、悲観主義が何を修復し、いつ不要になるのかを確認する。

```
import gymnasium as gym
```

15.7.1 固定された記録からの学習

15.7.1.1 他の方策のデータを利用できるターゲット

ここまでの 2 章で扱った更新則は、2 つの系統に分けられる。方策勾配（policy gradient）は現在の方策自身の軌跡に関する期待値であるため、別の方策からのサンプルでは異なる量を推定することになる。したがって REINFORCE とアクター・クリティックは**オンポリシー**であり、PPO が 1 つのバッチを数エポックにわたって再利用できるのも、各行動の確率を比率で補正しているからにほかならない。一方、Q 学習のターゲット $r + \gamma \max_{a'} \widehat{Q}(s', a')$ は、(s,a) に対して環

境がどう応答したかには依存するが、a を選んだ方策には依存しない。そのため、以前の方策や別のエージェント、あるいは固定ログによって収集された遷移であっても、同じバックアップの有効なサンプルとなり、Q 学習は**オフポリシー**である。Figure 15.18 に、データの 3 つの利用形態を並べて示す。

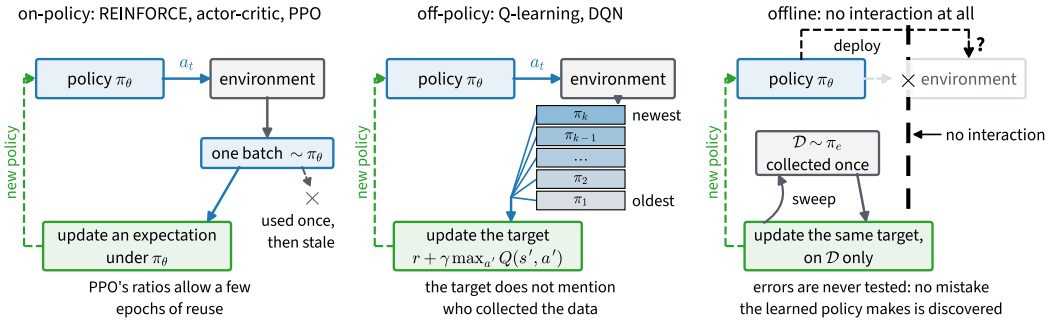


Figure 15.18: 3 つのデータ利用形態。(a) オンポリシー：各バッチは現在の方策から得られ、更新後には古くなる。PPO の比率補正により数エポックの再利用が可能になる。(b) オフポリシー：ベルマンターゲットは収集元の方策に依存しないため、バッファに過去の遷移を保持できる。(c) オフライン：固定されたデータセットを繰り返し走査し、運用する方策が選ぶ行動を新たな遷移で検証することはない。

14.3 節で扱った Q 学習のオンポリシー版である SARSA は、最良の行動ではなく、行動 a'（行動方策が s' で実際に取ったもの）を用いてブートストラップする。14.3 節からその時間差分誤差（temporal-difference error）を再掲する。

$$\delta_{\text{SARSA}} = r + \gamma\, Q(s', a') - Q(s, a). \tag{15.52}$$

$a' \sim \pi_e(\cdot \mid s')$ について平均を取ると、このターゲットは π_e のベルマン期待バックアップとなるため、SARSA はデータを収集した方策の価値 Q^{π_e} を学習する。他者のログから SARSA が学べるのは、その他者自身のことだけである。Q 学習はどのようなログからでも Q^* を学習し、演習問題 8（14.3 節）では崖の上でこの 2 つを並べて比較している。

オフポリシーだからといって、どのようなデータでもよいわけではない。Q 学習のテーブルの各要素の精度は背後にある遷移データ次第であるため、貪欲方策（greedy policy）が到達するすべての状態において選ぶ対は、実際に試されており、かつ推定に十分な回数だけ試されていなければならない。この要件を**被覆条件**（coverage condition）と呼ぶ。オンラインであれば、エージェントは推定が不確かな場所へ自ら向かうことで被覆の不足を補える。しかし**オフライン強化学習**（offline reinforcement learning）では、エージェントは固定されたデータセットしか受け取らないため (Levine et al. 2020)、ログが持つ被覆がすべてとなる。

15.7.1.2 滑りやすい湖の 2 種類のログ

滑りやすい湖には 16 個のセルと 4 つの行動がある。11 個のセルは穴でもゴールでもないため、エージェントが行動を選択できる状態行動対（state-action pair）は 44 個あり、あらゆる移動は 3 回に 2 回の割合で横方向に滑る。実験者は湖の遷移テーブルを知っており、それを `d2l.TabularMDP`、すなわち 14.1 節の配列 $P(s' \mid s, a)$ と $r(s, a)$ に読み込むが、以下の学習器がそれを目にすることは一切ない。各種類につき 40 個のログを生成する。

```
gamma, num_datasets = 0.95, 40      # discount; logs of each kind
env = gym.make('FrozenLake-v1', is_slippery=True)
mdp = d2l.TabularMDP.from_gym(env, gamma)
```

真のモデルに対する 14.2 節の価値反復法（value iteration）が基準を与える。いかなる方策も、開始セルから $V^*(s_0)$ を超える価値を得ることはできない。`greedy` は行動価値のテーブルを行動確率のテーブルとして表された貪欲方策に変換し、`d2l.epsilon_greedy` と同様に同値を均等に分割する。もし同値をすべて最初の行動に割り当てる規則にすると、すべて 0 の行からは常に左へ歩いてしまうだろう。`mdp.policy_value` は 14.2 節の線形ベルマン方程式 $V^\pi = (I - \gamma P_\pi)^{-1} r_\pi$ を解くことで、そのような方策テーブルを厳密に評価する。以下で方策が**獲得する**価値はすべて、ロールアウトを行わずにこの方法で計算したものである。

```
V_star = d2l.value_iteration(mdp, num_iters=1000)[-1]
Q_star = mdp.backup(V_star)

def greedy(Q):
    """The greedy policy of Q as a table of probabilities, ties split
    evenly, the rule of d2l.epsilon_greedy."""
    best = Q == Q.max(axis=1, keepdims=True)
    return best / best.sum(axis=1, keepdims=True)

print(f'V*(s0) = {V_star[0]:.3f}')
```

```
V*(s0) = 0.180
```

1 種類目のログは、能力をまったく持たないエージェントによるもので、一様ランダムな行動からなる 500 エピソードである。あらゆる場所ですべての行動を試すが、ゴールにはめったに到達しない。2 種類目は、有能だが不完全なオペレータによるもので、ϵ 貪欲な行動を Q^* のもとで $\epsilon = 0.2$ として 100 エピソード実行した記録である。したがってオペレータは確率 $0.8 + 0.2/4 = 0.85$ で最良の行動を取り、残りの確率でランダムな行動を取る。サンプルがオペレータの選択に集中するため、これを**狭い**（narrow）ログと呼ぶ。`make_log` は `d2l.rollout` を使ってログを収集する。この関数は方策をエピソード完了まで実行し、`d2l.Batch`（フィールドは `obs`、`act`、`rew`、`next_obs`、`term`）として遷移を返すものであり、シードによってログが固定される。`count` はログが各対を試した回数 $n(s, a)$ を集計する。また、行動方策自体を評価できるように、`logs` の各エントリにはその行動確率テーブルも保持しておく。

```
def make_log(env, behavior, num_episodes, seed):
    """A fixed log: complete episodes of behavior(obs, rng), seeded."""
    env.reset(seed=seed)
    rng = np.random.default_rng(seed)
    return d2l.rollout(env, behavior, num_episodes, rng)

def count(batch, shape=(16, 4)):
    """n(s, a): how often the log tried each state-action pair."""
    n = np.zeros(shape)
    np.add.at(n, (batch.obs, batch.act), 1)
    return n

def uniform(s, rng):
    return int(rng.integers(4))
```

```
def expert_eps(s, rng):          # the operator: epsilon-greedy on Q*, 0.2
    return d2l.epsilon_greedy(Q_star[s], 0.2, rng)

logs = {'uniform': (uniform, np.full((16, 4), 1 / 4), 500),
        'narrow': (expert_eps, 0.8 * greedy(Q_star) + 0.2 / 4, 100)}
live = np.isin(env.unwrapped.desc.flatten(), [b'S', b'F'])  # 11 cells
for name, (behavior, pi_e, num_episodes) in logs.items():
    batch = make_log(env, behavior, num_episodes, seed=0)
    n = count(batch)[live]
    print(f'{name:>7}: {len(batch):,} transitions, {(n > 0).sum()}/{n.size} '
          f'pairs tried, median support {np.median(n):.0f}; '
          f'behavior earns {mdp.policy_value(pi_e)[0]:.3f}')
```

```
uniform: 3,997 transitions, 44/44 pairs tried, median support 42; behavior earns 0.008
 narrow: 1,897 transitions, 42/44 pairs tried, median support 8; behavior earns 0.087
```

ランダムなエージェントが開始セルから獲得する価値は 0.008 と最適値 0.180 の約 1/20 にすぎないが、その 500 エピソードはすべての有効な対を試し、典型的な対を約 40 回試している。一方、オペレータは半分の長さのログから最適値の約半分にあたる 0.087 を獲得する。しかしサンプルが自身の選択に集中するため、2 つの対が未試行のまま残り、典型的な対には 8 サンプルしか与えられない。被覆と能力は逆方向に引き合い、これら 2 種類のログはその両端に位置している。

15.7.1.3 厳密な最適解に対する分布シフト

Q 学習は、環境に対して実行するのと同じようにログに対しても実行できる。`offline_q_sweeps` は記録された遷移をシャッフルし、14.3 節の更新を減衰ステップサイズ $1/(1+0.1\,n)$（`d2l.q_learning` と同じ形式で、n はその対のこれまでの更新回数）で適用し、この処理を 200 スイープ繰り返す。

```
def offline_q_sweeps(batch, gamma, num_sweeps=200, shape=(16, 4), seed=0):
    """The Q-learning loop of d2l.q_learning, step 1 / (1 + 0.1 n(s, a)),
    swept over the fixed log in a fresh random order each time."""
    Q, visits = np.zeros(shape), np.zeros(shape)
    rng = np.random.default_rng(seed)
    for _ in range(num_sweeps):
        for i in rng.permutation(len(batch)):
            s, a, s2 = batch.obs[i], batch.act[i], batch.next_obs[i]
            visits[s, a] += 1
            Q[s, a] += (batch.rew[i] + gamma * (1 - batch.term[i])
                        * Q[s2].max() - Q[s, a]) / (1 + 0.1 * visits[s, a])
    return Q
```

各スイープは同じ遷移を再訪するため、テーブルはログのみに依存する不動点（fixed point）に収束する。ログを集計して推定モデルを作ろう。$\widehat{P}(s' \mid s,a)$ は (s,a) の試行のうち s' に到達した割合であり、$\widehat{r}(s,a)$ はその平均報酬である。この不動点は、この**経験 MDP**（empirical MDP）における最適行動価値のテーブル、すなわち 14.3 節の確実性等価（certainty-equivalent）なテーブルである。`offline_q` は集計されたモデルに対する価値反復法によってこれを直接計算する。引数 `kappa` は各バックアップの内部で差し引かれる対ごとのペナルティであり、後ほど悲観主義を扱うまでは 0 のままとする。未試行の対に対応する行はすべて 0 であるため、どこにも遷移せず価値も 0 となる。この湖では報酬が決して負にならないため、どの方策も取り得る最小の価値

は 0 であり、0 でのクリッピングによってペナルティが価値をそれ未満に押し下げるのを防ぐ。正の kappa を与えると、この関数は未試行の対に無限大のペナルティも科すが、クリッピングによって同じ下限である 0 に戻されるため、悲観主義を導入しても未試行の対は素朴なテーブルが置いたのと同じ位置にとどまる。

```
def offline_q(batch, gamma, kappa=0.0, num_iters=500, shape=(16, 4)):
    """Where offline Q-learning with decaying steps converges: value
    iteration on the log's empirical MDP. kappa > 0 subtracts
    kappa / sqrt(n(s, a)) inside every backup, distrusts untried pairs
    entirely, and clips at zero, the floor this environment guarantees."""
    n = count(batch, shape)
    P, r = np.zeros(shape + shape[:1]), np.zeros(shape)
    np.add.at(P, (batch.obs, batch.act, batch.next_obs), 1)
    np.add.at(r, (batch.obs, batch.act), batch.rew)
    m = np.maximum(n, 1)                          # untried: rows of zeros
    mdp_hat = d2l.TabularMDP(P / m[..., None], r / m, gamma)
    penalty = np.where(n > 0, kappa / np.sqrt(m),
                       np.inf if kappa > 0 else 0.0)
    Q = np.zeros(shape)
    for _ in range(num_iters):
        Q = np.maximum(mdp_hat.backup(Q.max(axis=1)) - penalty, 0.0)
    return Q
```

このループと直接解法は一致するだろうか。最初の一様ログに対して両方を実行してみよう。

```
batch = make_log(env, uniform, 500, seed=0)
Q_loop, Q_solve = offline_q_sweeps(batch, gamma), offline_q(batch, gamma)
print(f'largest difference between the two tables: '
      f'{np.abs(Q_loop - Q_solve).max():.3f}')
print(f'their greedy policies earn {mdp.policy_value(greedy(Q_loop))[0]:.3f}'
      f' and {mdp.policy_value(greedy(Q_solve))[0]:.3f}')
```

```
largest difference between the two tables: 0.006
their greedy policies earn 0.137 and 0.137
```

2 つのテーブルの差はすべての要素で 1/100 未満であり、両者の貪欲方策が獲得する価値も一致する。ログ 1 個あたり、ループは数秒かかるのに対し直接解法は数ミリ秒で済むため、以下の実験では `offline_q` を呼び出す。テーブルに残る誤差がログのサイズとともにどう縮小するかは、演習問題 3 で問う。

2 つ目の学習器は 14.5 節の行動クローニング（behavior cloning）であり、ログを収集した方策に適合させる手法である。テーブル形式では、交差エントロピーによる訓練は各状態におけるログの行動頻度を復元するため、`clone_policy` はその頻度を集計し、ログが一度も訪れなかった状態では一様に行動する。クローンが予測するのはログが実際に達成した値、すなわちエピソードの割引収益の平均である。

```
def clone_policy(batch, shape=(16, 4)):
    """Behavior cloning on a table: the log's empirical action
    distribution, uniform wherever the log is silent."""
    n = count(batch, shape)
    total = n.sum(axis=1, keepdims=True)
```

```
    return np.where(total > 0, n / np.maximum(total, 1), 1 / shape[1])
```

各種類 40 個のログそれぞれに対して、ペナルティなしの素朴なテーブルとクローンを適合させ、開始セルにおいてそれぞれが予測する値と、その方策が獲得する価値を記録する。fit_and_score は任意のペナルティ kappa のもとでテーブルに対してこの処理を実行し、Batch のメソッド episode_returns は各記録エピソードの割引収益（クローンの予測のもとになる値）を与える。

```
def fit_and_score(batches, kappa=0.0):
    """Per log: the table's own prediction max_a Q(s0, a) and the exact
    value of its greedy policy."""
    fits = [offline_q(batch, gamma, kappa) for batch in batches]
    return (np.array([Q[0].max() for Q in fits]),
            np.array([mdp.policy_value(greedy(Q))[0] for Q in fits]))

datasets, pred, value = {}, {}, {}   # predicted and earned values
for name, (behavior, _, num_episodes) in logs.items():
    datasets[name] = [make_log(env, behavior, num_episodes, seed)
                      for seed in range(num_datasets)]
    pred[name, 'naive'], value[name, 'naive'] = fit_and_score(datasets[name])
    pred[name, 'clone'] = np.array(
        [batch.episode_returns(gamma).mean() for batch in datasets[name]])
    value[name, 'clone'] = np.array(
        [mdp.policy_value(clone_policy(batch))[0]
         for batch in datasets[name]])
```

summary は 40 個のログにわたる中央値と、予測値が最適値およびその方策自身の厳密な価値を上回ったログの個数を出力する。tally は、一方の学習器の方策が獲得する価値が他方より高いログ、低いログ、等しいログの個数をそれぞれ数える。

```
def summary(pred, value, v_star):
    """Medians; on how many logs the prediction exceeds V*(s0) and the
    exact value of the learner's own policy."""
    return (f'predicts {np.median(pred):.2f}, earns '
            f'{np.median(value):.2f}; above V* on {(pred > v_star).sum():2d}'
            f', above own value on {(pred > value + 1e-9).sum():2d}')

def tally(value, other):
    """On how many logs value is higher, lower, equal to other."""
    diff = value - other
    return (diff > 1e-9).sum(), (diff < -1e-9).sum(), (abs(diff) <= 1e-9).sum()

for name in logs:
    for arm in ['clone', 'naive']:
        print(f'{name:>7} {arm}: ' + summary(pred[name, arm],
                                              value[name, arm], V_star[0]))
    wins = tally(value[name, 'naive'], value[name, 'clone'])[0]
    print(f'{"":>7} the naive table beats the clone on {wins} of '
          f'{num_datasets} logs')
```

```
uniform clone: predicts 0.01, earns 0.01; above V* on  0, above own value on 17
uniform naive: predicts 0.20, earns 0.14; above V* on 28, above own value on 36
        the naive table beats the clone on 40 of 40 logs
```

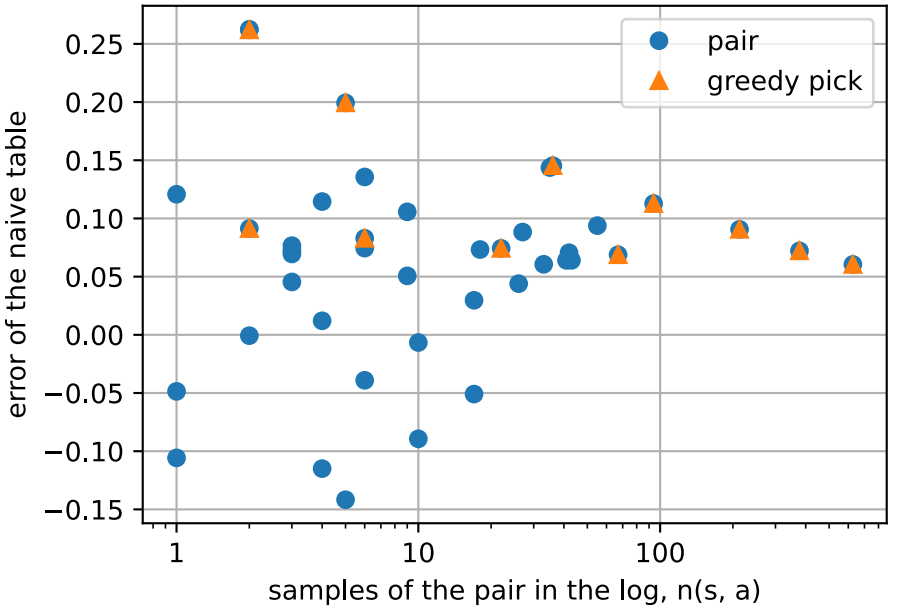
```
narrow clone: predicts 0.09, earns 0.09; above V* on  0, above own value on 18
narrow naive: predicts 0.25, earns 0.11; above V* on 39, above own value on 39
       the naive table beats the clone on 27 of 40 logs
```

一様ログでは、素朴なテーブルの予測値は 40 個中 36 個のログで自身の方策が獲得する価値を上回り、28 個のログではどの方策が獲得し得る価値よりも高くなる。予測値の中央値は約 0.20 であり、最適値 0.18 を上回る。狭いログではこの乖離がさらに広がる。予測値は 40 個中 39 個のログで V^* を上回り、中央値は 0.25 に達する一方、方策が実際に獲得する価値は約 0.1 にとどまる。学習器が予測する値とその方策が獲得する価値を比較することは、オフライン強化学習における標準的な診断手法であり、深層ネットワークを用いるとこの乖離ははるかに大きくなり得る (Fujimoto et al. 2019)。

この膨れ上がった予測値はどこから生じるのだろうか。最初の狭いログを取り上げ、試行された各有効対について、そのサポート $n(s, a)$ に対する素朴なテーブルの誤差 $\hat{Q}(s, a) - Q^*(s, a)$ をプロットし、各セルで貪欲方策が選ぶ対に印を付ける（2 つの行動が同値の場合、`argmax` は最初の行動に印を付ける一方、`greedy` は同値を均等に分割するが、これらのログでは選ばれた行動の間で同値が生じることはまれである）。

```
batch = datasets['narrow'][0]
n, Q = count(batch), offline_q(batch, gamma)
err, tried = Q - Q_star, live[:, None] & (n > 0)
cells = np.flatnonzero(live)
picks = Q[cells].argmax(axis=1)
print('picks with n <= 5: ' + ', '.join(
    f'cell {s} (n = {n[s, a]:.0f}, {err[s, a]:+.2f})'
    for s, a in zip(cells, picks) if n[s, a] <= 5))
d2l.plot([n[tried], n[cells, picks]], [err[tried], err[cells, picks]],
         xlabel='samples of the pair in the log, n(s, a)',
         ylabel='error of the naive table', legend=['pair', 'greedy pick'],
         xscale='log', fmts=('C0o', 'C1^'), figsize=(5, 3.5))
```

```
picks with n <= 5: cell 2 (n = 2, +0.09), cell 10 (n = 5, +0.20), cell 14 (n = 2, +0.26)
```



20 個を超えるサンプルを持つ対はすべて、約 0.05 から 0.15 だけ上向きに誤差を生じている。それら自身の遷移は精度よく推定されているものの、ターゲットが後続セルの価値からブートストラップしており、その後続セルがさらに先のセルの誤差を引き継ぎ、各後続バックアップの最大値演算がその誤差を上向きに偏らせるためである。サンプル数が約 10 個を下回ると、誤差は -0.15 から $+0.26$ まで両方向に広がり、貪欲方策が選んだサポートの薄い対にはプロット中で最大の 2 つの誤差が含まれている。具体的には、選ばれた対のうち 2 つは 2 サンプル、1 つは 5 サンプルにしか基づいておらず、そのうち 2 つは 0.26 および 0.20 と、最適値そのものを超える幅で過大評価されている。1 つのログだけでは偶然かもしれないので、40 個すべての狭いログにわたって選ばれた対を集約し、1〜5 サンプルにしか基づかない選択を、同じくサポートが薄いすべての対と比較してみよう。

```
pick_n, pick_err, all_n, all_err = [], [], [], []
for batch in datasets['narrow']:
    n, Q = count(batch), offline_q(batch, gamma)
    picks = Q[cells].argmax(axis=1)
    pick_n.append(n[cells, picks])
    pick_err.append((Q - Q_star)[cells, picks])
    all_n.append(n[live].ravel())
    all_err.append((Q - Q_star)[live].ravel())
pick_n, pick_err, all_n, all_err = map(
    np.concatenate, (pick_n, pick_err, all_n, all_err))
thin, thin_all = (pick_n >= 1) & (pick_n <= 5), (all_n >= 1) & (all_n <= 5)
print(f'greedy picks resting on 1 to 5 samples: {thin.mean():.0%} of picks')
print(f'their median error {np.median(pick_err[thin]):+.2f}; all pairs '
      f'with 1 to 5 samples {np.median(all_err[thin_all]):+.2f}')
```

```
greedy picks resting on 1 to 5 samples: 23% of picks
their median error +0.22; all pairs with 1 to 5 samples +0.08
```

貪欲方策の選択の約 1/4 は 5 個以下のサンプルに基づいており、これらの選択は約 0.2、すなわち同程度にサポートが薄い対の典型的な誤差の 2〜3 倍も過大評価されている。最大値演算はサポー

トの薄い対を無作為に選んでいるわけではない。たまたま誤差が正に振れた薄い対を選び出しているのであり、これは 15.4.3.1 節で見た最大化バイアス（maximization bias）にほかならない。さらに次のバックアップが、その膨らんだ価値を手前のセルへと伝播させる。その結果として、ログに含まれる行動と学習された方策が取る行動の間に生じる不一致を**分布シフト**（distribution shift）と呼ぶ。オンラインであれば、これは自然に修正される。エージェントが過大評価された行動を実際に取り、その価値を引き下げる遷移を収集して次へ進むからである。しかしオフラインではログが決して変化しないため、誤差はそのまま残る。

15.7.1.4 クローンとスティッチング

クローンはほぼ較正（calibration）されている。その予測値はログ自身の平均収益であり、行動方策の価値の不偏推定量となっていて、クローン自身の価値もそれに近く近似される。実際、予測値は約半数のログでクローンの価値を上回り、残りのログで下回る。しかし一様ログにおいては、まさにその性質が弱点となる。ランダムウォークを再現してしまい、獲得する価値は 0.01 にとどまるのである。

素朴なテーブルは 40 個すべての一様ログでクローンを上回り、最適値の 3/4 にあたる約 0.14 を獲得する（データを収集した方策の獲得価値はわずか 0.008 である）。その理由は遷移ごとのバックアップにある。あるランダムなエピソードがセル 10 に到達した後に穴へ落ち、別のエピソードが別の経路でセル 10 に到達してから滑ってゴールへ進んだとしよう（Figure 15.19）。バックアップはセル 10 を 2 番目のエピソードの後続経路によって評価し、その価値を 1 番目のエピソードの経路に沿って逆向きに伝播させることで、単一のエピソードでは一度もたどられなかった経路を合成する。この合成は**スティッチング**と呼ばれ、これによってオフライン Q 学習はデータを収集した方策を上回ることができる。一方、狭いログではクローンが強力なベースラインとなる。0.087 を獲得するオペレータを模倣して自身も約 0.09 を獲得するのに対し、素朴なテーブルはサポートの薄い選択に惑わされ、40 個中 27 個のログでしかクローンに勝てない。

15.7.2 悲観主義

15.7.2.1 手計算で見る 2 本腕バンディット

湖の学習器を変更する前に、最小の例を取り上げよう。2 本腕バンディットのオフラインログに、常に $1/2$ の報酬を与える腕 1 を 100 回引いた記録と、報酬 1 を確率 $1/4$ で与えそれ以外は 0 を与える腕 2 を 1 回だけ引いた記録があるとする。4 回に 1 回起きることだが、その 1 回の試行が 1 を出したとしよう。すると経験平均が最大の腕は 1 対 0.5 で腕 2 となり、それを選ぶと平均で 0.25 しか獲得できない。

信頼下限（lower confidence bound、LCB）は、各経験平均 $\hat{r}(a)$ から腕の試行回数 $n(a)$ に応じて縮小する半径を差し引く。報酬が $[0,1]$ に収まるとき、ヘフディングの不等式は次の半径を与える。

$$b(a) = \sqrt{\frac{\log(2K/q)}{2\,n(a)}} \tag{15.53}$$

ここで K は腕の本数、q は失敗確率である。少なくとも確率 $1-q$ で、すべての腕の真の平均はその経験平均から $b(a)$ 以内に収まる。$K=2$、$q=0.1$ とすると、$b(1)=\sqrt{\log 40/200}\approx 0.136$、$b(2)=\sqrt{\log 40/2}\approx 1.358$ である。下限は腕 1 が $0.5-0.136=0.364$、腕 2 が $1-1.358=-0.358$ となるため、この規則は腕 1 を選ぶ (Rashidinejad et al. 2021)。その 1 回の試行が出し得た報酬について平均すると、経験平均が最良の腕を選ぶ規則の獲得価値は $\frac{3}{4}\cdot 0.5+\frac{1}{4}\cdot 0.25\approx 0.44$ であるのに対し、下限規則は毎回 0.5 を獲得する。

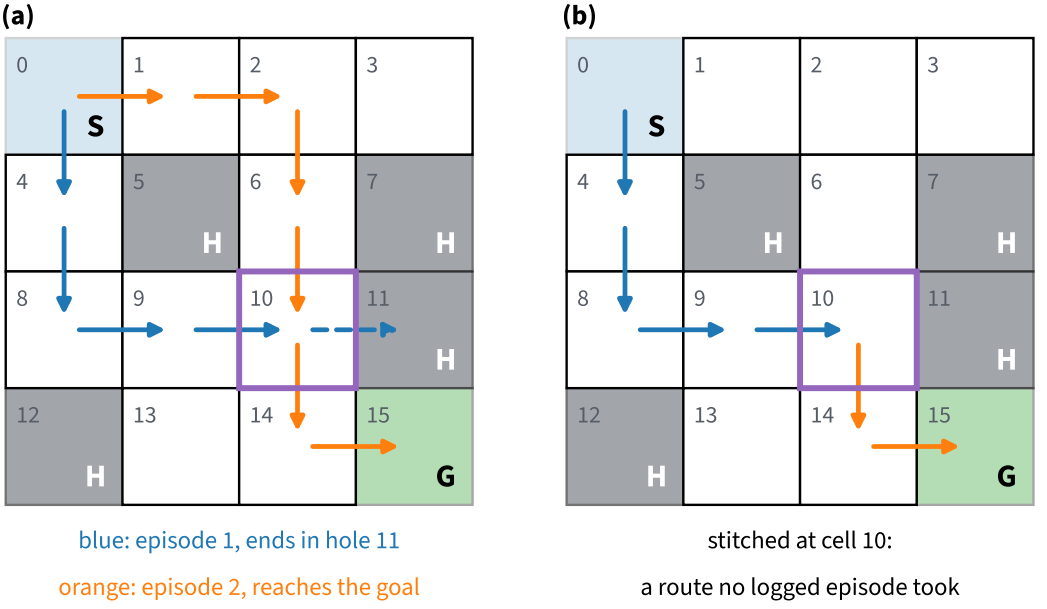


Figure 15.19: 湖におけるスティッチング。(a) 記録された 2 つのエピソードがセル 10（枠線付き）を通過する。1 つ目（青）は左端を下って穴 11 に落ち、2 つ目（オレンジ）は最上段を進んでゴールへ到達する。(b) 遷移ごとのバックアップが 1 つ目のエピソードの経路と 2 つ目のエピソードの後続経路をつなぎ合わせ、どの記録エピソードもたどらなかった経路を作る。

悲観主義には代償もある。もし腕 2 が平均して 0.7 を与え、経験平均も 0.7 だったとしても、$0.7 - \sqrt{\log 40/(2n_2)}$ が 0.364 を上回るまでこの規則は腕 2 を拒み続ける。それが初めて起きるのは $n_2 = 17$ 回引いた時点である。悲観的な学習器は、データが裏付けを与えるまで良い行動を見送るのである。

15.7.2.2 1 行で表されるペナルティ

湖の学習器にも同じ処置を施そう。素朴なテーブルの各要素はノイズを含む $n(s,a)$ 個のターゲットを平均しているため、その標本誤差（sampling error）は $1/\sqrt{n(s,a)}$ のオーダーで縮小する。$\kappa > 0$ のとき、`offline_q` はすべてのバックアップの内部で $\kappa/\sqrt{n(s,a)}$ を差し引き、結果を 0 でクリッピングする。

$$Q(s,a) \leftarrow \max\left(\hat{r}(s,a) + \gamma \sum_{s'} \hat{P}(s' \mid s,a) \max_{a'} Q(s',a') - \frac{\kappa}{\sqrt{n(s,a)}},\ 0\right). \tag{15.54}$$

更新式自体は素朴なものと同じであり、唯一の違いはログが最も手薄な箇所で最大となるペナルティである。一度も試されなかった対は、ペナルティの有無にかかわらず下限の 0 にとどまる。この形式は、悲観的価値反復法（pessimistic value iteration）の表形式版である (Jin et al. 2021)。ここで κ がヘフディングの不等式の定数ではなく自由なスケールパラメータとなっているのは、ある対のターゲット群が同じ推定後続価値からブートストラップしており、互いに独立ではないためである。

このペナルティは 14.4 節の信頼上限（upper confidence bound）規則、すなわち $\hat{\mu}(a) + \kappa\sqrt{\log t / n(a)}$ が最大の腕を選ぶ規則のボーナスと鏡像関係にある。どちらの項も $1/\sqrt{n}$ に比例して縮小するが、フィードバックの有無が異なるため符号が逆になる。オンラインでは、楽観的な推定値がエージェントにその行動を試させ、その試行が推定値を修正する。オフラインでは試行が不可能なため、同じ不確実性を差し引き、データがサポートするときにのみその行動を選ぶのである。

15.7.2.3 狭いデータと一様なデータ

それでは、同じ 80 個のログに対して `offline_q`、すなわち Equation 15.54 のバックアップを $\kappa = 0.1$ として再実行してみよう。

```
kappa = 0.1
for name in logs:
    pred[name, kappa], value[name, kappa] = fit_and_score(datasets[name],
                                                          kappa)
    print(f'{name:>7} kappa {kappa}: ' + summary(
        pred[name, kappa], value[name, kappa], V_star[0]))
    better, worse, tied = tally(value[name, kappa], value[name, 'naive'])
    print(f'{"":>7} against naive: better on {better}, worse on {worse}, '
          f'tied on {tied}; beats the clone on '
          f'{tally(value[name, kappa], value[name, "clone"])[0]}')
```

```
uniform kappa 0.1: predicts 0.05, earns 0.10; above V* on  0, above own value on  3
        against naive: better on 4, worse on 28, tied on 8; beats the clone on 40
 narrow kappa 0.1: predicts 0.13, earns 0.14; above V* on  4, above own value on 15
        against naive: better on 29, worse on 1, tied on 10; beats the clone on 39
```

狭いログでは、悲観主義はバンディットの例で見たとおりの効果を発揮する。予測値は 0.25 から約 0.13 まで下がり、最適値を上回るのはわずか 4 個のログになる。方策が獲得する価値は 40 個中 29 個のログで素朴なテーブルを上回り、下回るのは 1 個のみで、中央値は約 0.14 となり、40 個中 39 個のログでクローンを上回る。

一方、一様ログでは同じペナルティによって過大評価は解消され、最適値を上回る予測はなくなるものの、40 個中 28 個のログで収益が下がり、上がるのは 4 個にとどまる。すべての行動が何十回も試されている状況では、警戒すべき幸運な推定値はほとんどなく、対ごとに異なるペナルティは十分にサポートされた行動のうちどれが最良に見えるかを入れ替えるだけに終わる。悲観的オフライン学習の理論は、いつどちらの結果になるかを示している。信頼下限を用いる学習器の保証に必要なのは、ログがすべての対をカバーすることではなく、**最適**方策（optimal policy）の対をカバーすることである (Rashidinejad et al. 2021)。狭いログは最適方策の選択を厚く、それ以外を薄くカバーしているため悲観主義が有効に働くが、一様ログはすべてをカバーしているため素朴なテーブルに保護は不要なのである。

15.7.2.4 ペナルティの選択

κ はどの程度の大きさにすべきだろうか。値を大きくするとテーブルはより慎重になる。同じログに対して $\kappa = 0.3$ を試してみよう。

```
kappa = 0.3
for name in logs:
    p, v = fit_and_score(datasets[name], kappa)
    better, worse, _ = tally(v, value[name, 'naive'])
    print(f'{name:>7} kappa {kappa}: predicts {np.median(p):.2f}, earns '
          f'{np.median(v):.2f}; against naive better on {better}, '
          f'worse on {worse}')
```

```
uniform kappa 0.3: predicts 0.00, earns 0.02; against naive better on 0, worse on 40
 narrow kappa 0.3: predicts 0.00, earns 0.05; against naive better on 4, worse on 36
```

$\kappa = 0.3$ では、ほぼすべてのログで開始価値が 0 まで押し下げられ、方策は 40 個の狭いログのうち 36 個、また 40 個すべての一様ログで素朴なテーブルに敗れる。理由は、ペナルティがバックアップのたびに課され、価値が経路に沿った割引和だからである。各対が約 n サンプルを持つ経路に沿うと、課されるペナルティの合計は高々約 $\kappa/(\sqrt{n}\,(1-\gamma))$、すなわち $\gamma = 0.95$ では 1 ステップあたりの 20 倍に達する。もっとも、エピソードはゴールまたは穴で終了するため、経路に沿って実際に累積するペナルティはこれより少ない。経路上のすべての対に 1,000 個を超えるサンプルがあれば、この上界ですら $V^*(s_0) = 0.18$ を（$\kappa = 0.3$ であっても）下回るが、100 エピソードのログでは開始セルに数百サンプルあるだけで、その先ははるかに少ない。同じオペレータから 5 倍のデータを得れば、1 ステップあたりのペナルティは $\sqrt{5}$ 分の 1 に縮小する。すると最適経路上の大半の対は数百サンプルを持つようになり、上界が求める 1,000 サンプルには届かないものの、演習問題 1 で示すように、そこでは $\kappa = 0.3$ でもほとんど悪影響はない。実際に累積するペナルティがすでに十分小さいからである。ペナルティはデータ量に見合っていなければならず、したがって κ の選択はモデル選択（model selection）の問題となる。

15.7.2.5 較正とモデル選択

ここまでの比較はすべて実験者が持つ湖の厳密なモデルを用いたが、実際のオフライン運用ではそのようなモデルは存在しない。運用時に手元にあるのはログだけであり、κ の各値や学習された各テーブルについて、候補となる方策がどれほどうまく機能するかをそのログのみから推定しなければならない。1 つ目の推定量は、候補方策と行動方策のもとでの各行動の確率比によって記録されたエピソードに重み付けし直す手法である。その重みはエピソード全体にわたる確率比

の積、すなわち 15.2 節で見たホライズンとともに分散が増大する積であり、貪欲な候補方策に対しては大半のエピソードの重みが 0 になる。もう 1 つの推定量は、ログのうちホールドアウトした部分から作った経験 MDP で候補方策を評価するものだが、その部分が候補方策の選択をカバーしている範囲でしか信頼できない。較正については部分的な確認が可能であり、自身のログ内の最良の割引収益を上回る値を予測するテーブルは疑わしい。ただし、この湖ではこの検査が引っかかることは一度もない。どのログでも最良のエピソードは約 0.5 以上の収益を上げており、本節のあらゆる予測値を上回るからである。演習問題 5 では、一様ログに対してこれら両方の推定量を試す。

15.7.3 表形式を超えて

15.7.3.1 方策または価値の制約

連続的な状態と行動では出現回数を数えられないが、本節で見た 2 つの対処法はそのまま残る。ログがサポートする行動の近くに方策を保つか、サポートしない行動の価値を押し下げるかである。1 つ目は、行動方策を参照方策とする 15.3 節の KL 正則化目的関数である。各状態において、方策は次の式を最大化する。

$$\mathbb{E}_{a\sim\pi(\cdot|s)}\big[\hat{Q}(s,a)\big] - \beta\, D_{\mathrm{KL}}\big(\pi(\cdot \mid s)\,\|\,\pi_e(\cdot \mid s)\big), \tag{15.55}$$

このため、ログが一度も取らなかった行動には無限大のペナルティが課される。実際には π_e は未知であり、ログで代用する。たとえば TD3+BC は、方策の行動と記録された行動の二乗距離をアクターの目的関数に加える (Fujimoto and Gu 2021)。定数項を除けば、この二乗距離は方策の行動を中心とするガウス分布のもとでの記録行動の負の対数尤度に等しいため、この項は逆向きのダイバージェンス $D_{\mathrm{KL}}(\pi_e\|\pi)$ のサンプル推定量となっている。次節の結びで扱う直接選好最適化（direct preference optimization、DPO）は、同じ制約のもとで参照方策を π_e の役割に据え、応答対の選好からなる固定データセットから学習する。

2 つ目の対処法は、カウントを必要とする $\kappa/\sqrt{n}$ を、ネットワークの出力だけで計算できるペナルティに置き換える。保守的 Q 学習（conservative Q-learning、CQL）は、記録された状態におけるすべての行動の価値を引き下げつつ、記録された行動の価値を引き上げる項を時間差分損失に加える。離散行動の場合、この項は $\alpha_{\mathrm{c}}\big(\log\sum_{a'} e^{Q(s,a')} - Q(s,a)\big)$ である (Kumar et al. 2020)。この項の前半は行動にわたるソフト最大値（soft maximum）であるため、その勾配はソフトマックスの重みに比例してすべての価値を引き下げる。すなわち、ログがサポートしているかどうかにかかわらず、最良に見える行動ほど大きく引き下げられ、記録された行動だけが引き上げられる（Figure 15.20）。Table 15.1 に、これら 2 つの系統と本節でカウントを用いて行った処理との対応をまとめる。

Table 15.1: オフライン手法の 2 つの系統と、湖の実験における対応物。

系統	制約の対象	本節での対応物	代表的手法
方策制約	方策を記録された行動の近くに保つ	極端な場合としてのクローン	TD3+BC
価値ペナルティ	サポートされない行動の価値を押し下げる	`offline_q` の $\kappa/\sqrt{n}$	CQL

最大値演算の対象を記録された行動のみに限定する厳格な規則を設けても、湖の実験では何も変わらない。未試行の対の価値はすでに下限の 0 であり、最大値に選ばれることは決してないからである。そこでの難しさはサポートの欠如ではなく、サポートの薄さにあった。本章の関連文献（15.9 節）では、これら 2 つの考え方に基づく深層手法を比較している。

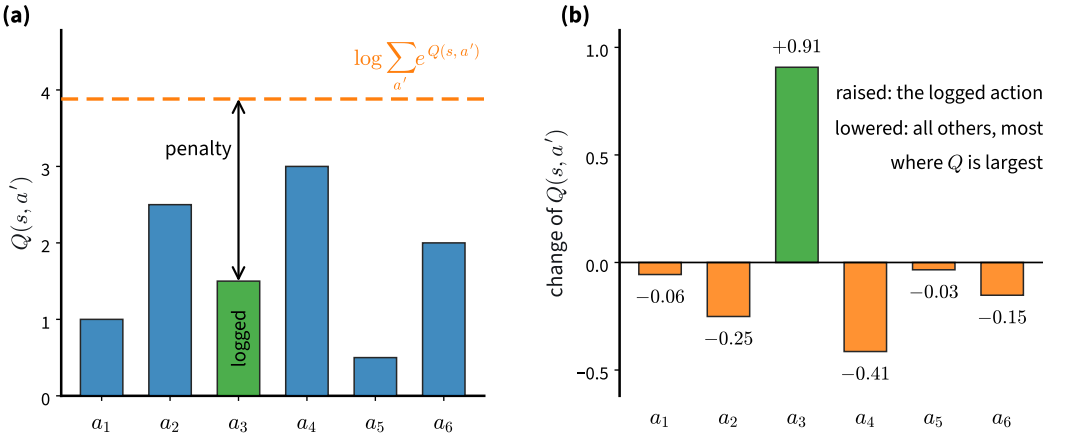


Figure 15.20: 記録された 1 つの状態における保守的ペナルティ（6 つの行動のうちログが保持するのは a_3）。(a) ペナルティは価値のソフト最大値 $\log \sum_{a'} e^{Q(s,a')}$（破線）と記録された行動の価値との差である。(b) ペナルティに対してステップサイズ 1 の勾配ステップを 1 回適用すると、すべての価値がそのソフトマックスの重みだけ引き下げられ（最大の価値ほど大きく下がり）、記録された行動の価値にのみ 1 が加算されて唯一上昇する。変化量の和は 0 になる。

15.7.3.2 収益条件付け

3 つ目のアプローチは、最大値演算そのものを回避する。Decision Transformer は記録された各軌跡を残余収益（return-to-go）・状態・行動の系列として扱い、各行動を予測するよう自己回帰モデルを訓練し、テスト時には高い目標収益で条件付ける (L. Chen, Lu, Rajeswaran, Lee, Grover, Michael Laskin, et al. 2021)。通常の教師あり学習によって収益で条件付けた小さなネットワークでもその性能の大部分が再現されるため、改善の多くは系列アーキテクチャではなく収益による条件付け自体に由来している (Emmons et al. 2022)。収益条件付き手法は、ログ内に良質な軌跡が存在するベンチマークでは価値ベースの手法と同程度の性能を示す。しかし、異なるエピソードの断片を継ぎ合わせて良い方策を構成しなければならない場面では後れを取り、また確率的な環境ではログ内の高い収益が単なる幸運の産物である場合があるため失敗し得る (Paster et al. 2022; Brandfonbrener et al. 2022)。

最小の例を見れば、この失敗がよくわかる。1 ステップの課題で、行動 A は報酬 1 を確率 $1/2$ で与え（それ以外は 0）、行動 B は常に 0.6 を与えるとし、一様ランダムな行動方策が両方を何度も記録したとする。収益 1 を記録したエピソードはすべて行動 A を取っているため、ログ内の最大収益で条件付けた方策は常に A を取り、獲得する価値は 0.5 となって B の 0.6 を下回る。収益条件付き方策が幸運なエピソードを模倣してしまうのに対し、Q 学習器は各行動の結果を平均して B を選ぶ。

以上で、固定されたログから方策を学習し、その予測値と獲得価値を比較し、データにサポートされた推定値とたまたま運が良かっただけの推定値を見分けられるようになった。悲観主義は、ログが狭いときには素朴なテーブルを改善するが、ログがすべてをカバーしているときには省くべきである。カウントを見ればログが各対をどれだけ均等にカバーしているかはわかるが、最適方策の対をカバーしているかどうかはわからず、ログのどこを見ても κ の値は決まらない。実務でよく取られる解決策は、純粋なオフラインにとどまるのをやめることである。すなわち、まずログから学習し、その後に少しだけ新しいデータを収集してファインチューニングする。また、本節で扱った考え方の 1 つは 15.8 節へとつながる。そこでは DPO が、上記の KL 制約のもとで選好の固定データセットから言語モデルを学習する。

15.7.4 まとめ

オフライン強化学習は固定されたログから方策を学習するが、利用できるのは Q 学習のようなオフポリシーのターゲットだけであり、しかも学習された方策が選ぶ対をログがカバーしている範囲に限られる。環境との相互作用がない場合、バックアップ内の最大値演算は推定値がたまたま過大になったサポートの薄い行動を選び出し、新たな試行がそれを修正することもない。実際、滑りやすい湖では素朴なテーブルは大半のログで最適値を上回る予測を出しながら、予測をはるかに下回る価値しか獲得できない。悲観主義は探索ボーナス（exploration bonus）と鏡像関係にある不確実性ペナルティ $\kappa/\sqrt{n}$ をバックアップの内部で差し引く。これはログが狭いときには方策を改善する一方、ログがすでにすべての行動をカバーしているときには収益を損ない、しかもそのスケールは本節の実験で用いたような厳密な評価なしに選ばなければならない。深層オフライン手法は方策を記録された行動の近くに制約するかサポートされない価値にペナルティを科し、収益条件付き手法は最大値演算を回避する代わりにスティッチングを犠牲にし、幸運なエピソードを鵜呑みにするリスクを負う。

15.7.5 演習問題

1. [code] **悲観主義の適切な強さ**。`offline_q` を $\kappa \in \{0, 0.02, 0.05, 0.1, 0.2, 0.3, 1\}$ と変化させて、一様ログおよび狭いログに加え、さらに 2 種類（各 40 個）のログ、すなわち 500 エピソードの $\epsilon = 0.2$ オペレータと 100 エピソードの $\epsilon = 0.1$ オペレータに対して再実行せよ。
    1. 各種類のログについて、貪欲方策の厳密な価値の中央値を κ に対してプロットし、クローンの価値を水平線で描き加えよ。ペナルティが有効に働く領域、無害な領域、逆効果になる領域はそれぞれどこか。
    2. $\kappa = 1$ のとき貪欲方策はどのように振る舞い、その予測値はどの程度よく較正されているか。この事例から、κ を選ぶ基準としての較正について何が言えるか。
    3. 同じオペレータのログが 100 エピソードから 500 エピソードへ増えるにつれて、最良の κ はどのように移動するか。累積ペナルティの上界 $\kappa/(\sqrt{n}\,(1-\gamma))$ と関連付けて答え、実際に累積するペナルティがそれより小さくなる理由を説明せよ。
    4. 最良の κ を見つけるために用いた情報のうち、実際のオフライン運用では利用できないものは何か。

2. **幸運による条件付け**。Decision Transformer の最も単純な表形式版である収益条件付き教師あり学習は、状態 s および目標収益 g に対して、g を達成したエピソード群におけるログの行動分布 $\pi(a \mid s, g) = P_{\mathcal{D}}(a \mid s, G = g)$ に従って行動する (Brandfonbrener et al. 2022)。本文の 1 ステップ課題において、行動 A は報酬 1 を確率 p で与えるものとする。
    1. ログ内の最大収益で条件付けたとき、最適な行動が選ばれるのはどのような p の範囲か。
    2. 同じデータからオフライン Q 学習は何を学習するか。また、幸運なエピソードに惑わされないのはなぜか。
    3. [code] 滑りやすい湖において、各一様ログからゴールに到達したエピソードのみを残し、`clone_policy` でクローンを作成して結果を厳密に評価せよ。ログ全体のクローンや素朴なテーブルと比べるとどうなるか。この比較は、本文で挙げた収益条件付けの 2 つの限界のうちどちらを示しているか。

3. [code] **データセットサイズと誤差の下限**。素朴なテーブルの各要素は $n(s,a)$ 個のターゲットを平均しているため、誤差は $1/\sqrt{n}$ に比例して減少するように思える。100、500、2,000 エピソードの一様ログをそれぞれ 5 個ずつ生成せよ。有効な対のサポート n と絶対誤差 $e = |\hat{Q} - Q^{*}|$ を集約し、$e\sqrt{n}$ の中央値を n の区間 $[1, 10)$、$[10, 30)$、$[30, 100)$、$[100, \infty)$ ごとに計算せよ。$e\sqrt{n}$ は一定に保たれるか。もし n とともに上昇するなら、誤差は $1/\sqrt{n}$ より遅くしか減少していないことになる。サンプルが十分にある対のターゲッ

トに着目してこの残差を説明せよ。すなわち、そのターゲットにはその対自身のサンプル以外に何が含まれており、ログが大きくなるにつれてその誤差はどう縮小するか。

4. **2 本腕バンディットにおける悲観主義。**●2 本腕バンディットのオフラインログに、腕 1 の n_1 回の試行（常に $1/2$ を与える）と、腕 2 の n_2 回の試行（報酬 1 を確率 $1/4$ で与え、それ以外は 0 を与える）が記録されているとする。$r(a)$ を腕 a の平均報酬、$\hat{r}(a)$ をその経験平均、a^* を最良の腕とする。同値の場合は腕 1 を選ぶものとする。
   1. 経験最良腕は $\hat{r}(a)$ を最大化する。それが腕 2 となる確率と、期待不足量 $r(a^*) - r(\hat{a})$ を $n_2 \in \{1, 5, 20\}$ について計算せよ。答えは n_1 にどう依存するか。
   2. 本文の信頼下限規則は $\hat{a} = \operatorname{argmax}_a \hat{r}(a) - b(a)$（ただし $b(a) = \sqrt{\log(2K/q)/(2n(a))}$）を選ぶ。すべての腕が少なくとも 1 回は引かれていると仮定する。演習問題 2（14.4 節）で述べられたヘフディングの不等式とユニオンバウンドを用いて、$r(a^*) - r(\hat{a}) \le 2b(a^*)$ が少なくとも確率 $1 - q$ で成り立つことを示せ。
   3. 楽観的な規則 $\operatorname{argmax}_a \hat{r}(a) + b(a)$ は信頼上限規則の形をしている。これについても同様の上界を導出せよ。今度の上界はどの腕の試行回数に依存するか。また、この依存性がオンラインでは無害でありながらオフラインでは有害になるのはなぜか。
   4. 代わりに腕 2 が報酬 1 を確率 0.7 で与えるものとし、$n_1 = 100$、$q = 0.1$ とする。$n_2 \in \{5, 10, 17, 30\}$ について、経験最良腕と下限規則のそれぞれが腕 2 を選ぶ厳密な確率と、各規則の期待報酬を計算せよ。腕 2 が劣った腕であった小問 (a) と比較し、悲観主義が何から守り、何を代償とするかを述べよ。

5. [code] **シミュレータなしでの評価。**●本節のすべての比較では、学習された方策を真のモデルで厳密に評価したが、実際のオフライン運用ではそれは不可能である。通常の重要度サンプリングは、ターゲット方策 π の価値を、m 個のエピソード（行動方策 π_e による記録）から $\frac{1}{m}\sum_i w_i G_i$ によって推定し、重み付き重要度サンプリングは $\sum_i w_i G_i / \sum_i w_i$ によって推定する。ここで G_i はエピソード i の割引収益であり、$w_i = \prod_t \pi(a_t \mid s_t)/\pi_e(a_t \mid s_t)$ はそのステップにわたる積である。
   1. 最初の一様ログの素朴なテーブルに対する貪欲方策について、ログ内の 500 エピソードのうち 0 でない重みを受け取るものはいくつあるか。エピソードの長さを用いてその個数を説明せよ。
   2. 代わりに各一様ログの素朴なテーブルに対する ϵ 貪欲版（$\epsilon = 0.3$）を、それを訓練したログ上で評価せよ。両方の推定値と重みの有効標本サイズ $(\sum_i w_i)^2 / \sum_i w_i^2$ を計算し、実験者しか計算できない厳密な価値 `mdp.policy_value(pi)[0]` と比較せよ。固定されたターゲット方策に対して、通常の重要度サンプリングは不偏である。重みのばらつきは、2 つの推定値の誤差をどのように説明するか。
   3. 演習問題 1 における κ を、いずれかの推定量によって信頼性をもって選択できただろうか。

   Richard S. Sutton および Andrew G. Barto『Reinforcement Learning: An Introduction』5.5 節より改作。

6. **表形式における保守的 Q 学習。**●●保守的 Q 学習 (Kumar et al. 2020) は、選択した行動分布 $\mu(a \mid s)$ のもとで価値を引き下げ、ログの経験行動分布 $\hat{\pi}_e(a \mid s) = n(s,a)/n(s)$（ただし $n(s) = \sum_a n(s,a)$）のもとで価値を引き上げる項を二乗ベルマン誤差に加える。表形式において、1 回の反復は次を選ぶ。

$$\hat{Q}_{k+1} = \operatorname*{argmin}_Q \; \alpha_{\mathrm{c}}\Big(\mathbb{E}_{s\sim\mathcal{D},\, a\sim\mu(\cdot|s)}[Q(s,a)] - \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q(s,a)]\Big) + \frac{1}{2}\mathbb{E}_{(s,a)\sim\mathcal{D}}\Big[\big(Q(s,a) - \hat{\mathcal{T}}\hat{Q}_k(s,a)\big)^2\Big],$$

ここで $\mathbb{E}_{\mathcal{D}}$ はログの遷移にわたる平均、$\alpha_{\mathrm{c}} > 0$ はペナルティの強さを定める係数であり、$\hat{\mathcal{T}}\hat{Q}_k(s,a)$ は Q 学習のターゲット $r + \gamma \max_{a'} \hat{Q}_k(s',a')$ をログ内の (s,a) からの遷移にわたって平均したものである。$\mu(a \mid s) > 0$ となるのは $n(s,a) > 0$ を満たす箇所に限られると仮定する（さもなければ目的関数は下に有界でなくなる）。

1. $n(s,a) > 0$ を満たすすべての対が次式を満たすことを示せ。

$$\hat{Q}_{k+1}(s,a) = \hat{\mathcal{T}}\hat{Q}_k(s,a) - \alpha_{\mathrm{c}}\Big(\frac{\mu(a \mid s)}{\hat{\pi}_e(a \mid s)} - 1\Big). \tag{15.56}$$

   ペナルティが引き下げるのはどの対であり、引き上げるのはどの対か。
2. $\mu = \pi$（π はある方策）とする。1 回の反復によって $\sum_a \pi(a \mid s)\hat{Q}_{k+1}(s,a)$ がバックアップの同じ平均と比べて $\alpha_{\mathrm{c}}(\sum_a \pi(a \mid s)^2/\hat{\pi}_e(a \mid s) - 1)$ だけ引き下げられること、およびこの量が非負であることを示せ。
3. μ として最初の狭いログに対する素朴なテーブルの貪欲方策をとる。その有効な対について、小問 (a) の CQL ペナルティと `offline_q` のペナルティ $\kappa/\sqrt{n(s,a)}$ を $n(s,a)$ に対してプロットせよ。それぞれのペナルティはデータのどの性質に反応しているか。また、両者が異なる扱いをするのはどのような対か。

15.8 系列生成のための強化学習

本書でここまで扱ってきた言語モデルはすべて、テキストを模倣するように訓練されていた。すなわち、コーパスの書き手に対する行動クローニング（behavior cloning、14.5 節）である。今日実際に使われているモデルは、コーパスには含まれない信号を用いてさらに訓練される。解答の最終的な答えが正しいかを判定器が判定したり、2 つの応答のどちらが優れているかを読み手が評価したりする信号である。そのスコアは、モデル自身が書き上げた応答全体の後に一度だけ与えられ、模倣すべき正解トークンは存在しない。14.7 節と 14.8 節では、4 つの算術プロンプトと判定器を用いて、この設定で方策（policy）を訓練した。

この着想自体は、こうしたモデルよりも古い。Christiano et al. (2017) は人間の比較データに報酬を当てはめ、方策勾配（policy gradient）によって最適化した。Ouyang et al. (2022) は大規模言語モデル（large language model）に対してこれを実施し、教師ありモデルからの乖離に対するペナルティを加えた。さらに Shao et al. (2024) は価値ネットワークをサンプルのグループに置き換え、DeepSeek-AI (2025) は学習された報酬を判定器に置き換えた。

報酬が応答ごとに 1 つの数値で与えられるとき、トークンごとの視点は何をもたらすのだろうか。また、抜け穴を見つけた方策を 15.3 節のペナルティはどこまで抑止できるのだろうか。これらの問いへの答えから、14.8 節のグループ重み、15.2 節の確率比とクリッピング、そしてこのペナルティを組み合わせた事後学習（post-training）の更新則が導かれる。

15.8.1 軌跡としての応答

15.8.1.1 1 つの勾配と 2 つの視点

応答 $y = (a_1, \dots, a_T)$ は、プロンプト x に対してトークンを行動として生成される軌跡（trajectory）である（Table 15.2）。トークン t の直前の状態はプレフィックス $s_t = (x, a_1, \dots, a_{t-1})$ であり、そこに a_t を追加すると確率 1 で s_{t+1} へ遷移する。したがって、応答の確率はトークン確率の積 $\pi_\theta(y \mid x) = \prod_t \pi_\theta(a_t \mid s_t)$ となり、次式が成り立つ。

$$\nabla_\theta \log \pi_\theta(y \mid x) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t). \tag{15.57}$$

Table 15.2: マルコフ決定過程としての自己回帰テキスト生成。

言語モデリング	強化学習
プロンプト $x \sim \mu_0$	初期状態 s_1
トークン a_t	行動 a_t
プレフィックス $(x, a_1, \dots, a_{t-1})$	状態 s_t
完全な応答 $y = (a_1, \dots, a_T)$	軌跡 τ
EOS で終わるプレフィックス (x, y)	終端状態
次トークン分布	方策 $\pi_\theta(a_t \mid s_t)$
検証器または報酬モデルのスコア $r(x, y)$	終端報酬

Equation 15.57 の左辺は応答を 1 つの行動として扱い、右辺は T 個のトークン行動として扱う。最後に 1 つの報酬 $r(x, y)$ のみが与えられ割引がない場合、すべてのトークンの残りの累積報酬（reward-to-go）は $r(x, y)$ になる。そのため、どちらの視点でも各トークンのスコアには同じ**応答重み**（response weight）$r(x, y) - b(x)$ が掛けられる。ここで $b(x)$ はプロンプトのみに依存する任意のベースラインである。

15.8.1.2 応答内の信頼割り当て

応答重みは不偏であるが、すべてのトークンを同等に扱う。しかし、誤答の中にも正しいトークンが含まれていることがある。プロンプト `3*4` に対して、十の位 a_1 と一の位 a_2 がそれぞれ $0, \dots, 9$ から一様に選ばれて応答されるとする。正解は `12` のみであるため、$V(x) = 1/100$、$V(x, 1) = 1/10$ であり、他のすべての十の位に対しては $V(x, a_1) = 0$ である。応答 `13` はスコア 0 となり、両方のトークンに重み $r - V(x) = -0.01$ を受け取るため、正しい十の位の確率まで押し下げてしまう。これに対し、14.8 節のトークンアドバンテージはプレフィックスで条件付けるため、十の位は $A(s_1, a_1) = V(x, a_1) - V(x) = +0.09$、一の位は $A(s_2, a_2) = r - V(x, a_1) = -0.1$ を受け取る。関数 `token_credit` は、$\pi(y \mid x)$、応答重み、および両方のアドバンテージを 100 通りすべての応答 $y = 10a_1 + a_2$ について計算する。

```
def token_credit(p1, p2, r):
    """For two-digit responses y = 10 a1 + a2: pi(y | x), the response
    weight r - V(x) and the token advantages A(s_1, a1), A(s_2, a2)."""
    a1, a2 = np.divmod(np.arange(100), 10)
    prob = p1[a1] * p2[a2]
    V_x, V_xa = prob @ r, r.reshape(10, 10) @ p2   # V(x), V(x, a1)
    return prob, r - V_x, V_xa[a1] - V_x, r - V_xa[a1]
```

3 つの応答に対する 3 種類の重みを表示し、応答重みと十の位のアドバンテージが十の位（そのスコアは $\mathbf{e}_{a_1} - \mathbf{p}_1$）に対して同じ期待勾配を与えることを確認したうえで、いずれも平均が 0 である各重みの分散を比較する。

```
y = np.arange(100)                    # the response y = 10 a1 + a2
r = (y == 12).astype(float)           # the verifier of 3*4
uniform = np.full(10, 0.1)
prob, w, A1, A2 = token_credit(uniform, uniform, r)
print('response   r - V(x)   A(s_1, a_1)   A(s_2, a_2)')
for k in (12, 13, 52):
    print(f'{k:8d} {w[k]:10.2f} {A1[k]:13.2f} {A2[k]:13.2f}')
S1 = np.eye(10)[y // 10] - uniform     # the tens digit's score
```

```
    print('same expected gradient for the tens digit:',
          np.allclose(prob * w @ S1, prob * A1 @ S1))
    print(f'variance: r - V(x) {prob @ w**2:.4f}, A(s_1, a_1) {prob @ A1**2:.4f}'
          f', A(s_2, a_2) {prob @ A2**2:.4f}')
```

```
response   r - V(x)   A(s_1, a_1)   A(s_2, a_2)
      12       0.99          0.09          0.90
      13      -0.01          0.09         -0.10
      52      -0.01         -0.01          0.00
same expected gradient for the tens digit: True
variance: r - V(x) 0.0099, A(s_1, a_1) 0.0009, A(s_2, a_2) 0.0090
```

応答重みの分散は、十の位のアドバンテージの 11 倍に達する。応答重みは $r - V(x) = (V(x, a_1) - V(x)) + (r - V(x, a_1)) = A(s_1, a_1) + A(s_2, a_2)$ のようにテレスコーピング分解でき、第 2 項はどのプレフィックスを条件としても平均が 0 になるため 2 つの項は無相関であり、分散は $0.0099 = 0.0009 + 0.0090$ と単純に加算される。各トークンのスコアは他のすべてのトークンのノイズを背負い込み、その数は T 個のトークンに対して T 項にのぼる。

15.8.1.3 必要な構成要素と任意の構成要素

既知の遷移と単一の終端報酬（terminal reward）という性質により、本章の手法の多くは不要になる。応答は有限長であるため $\gamma = 1$ でよく、遷移モデル（transition model）を学習する必要はなく、生成は任意のプレフィックスから再開できる。また、オフポリシーでブートストラップすることもないため、15.4 節の死の三要素（deadly triad）も生じない。応答重みは不偏であり、14.8 節のグループ平均が RLOO (Ahmadian et al. 2024) や GRPO のようにネットワーク不要のベースラインとなるため、クリティックも任意である。一方、各プレフィックスに価値ヘッドを設けるか (Ouyang et al. 2022)、VinePPO のようにプレフィックスから続きをサンプリングすれば (Kazemnejad et al. 2025)、トークンごとの信頼割り当て（credit assignment）が得られる。結局のところ必要として残るのは、ベースライン付きのスコア、各更新ステップの上限、そして報酬の誤りに対する防御である（Figure 15.21）。

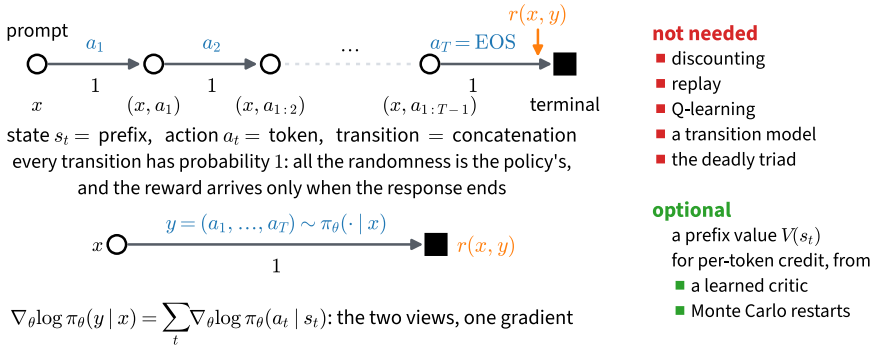


Figure 15.21: トークン MDP。上段：状態はプレフィックス、行動はトークンであり、すべての遷移は選択されたトークンを確率 1 で追加し、報酬は EOS により終端状態へ入ったときに一度だけ与えられる。下段：1 回のサンプル $y \sim \pi_\theta(\cdot \mid x)$ を $r(x, y)$ で評価した図であり、Equation 15.57 により同じ勾配を持つ。右：これによって不要になる構成要素と、任意の要素であるプレフィックス価値。

15.8.2 検証器・報酬モデルとその抜け穴

報酬は、人間の比較データに当てはめたモデル（15.3 節；比較データが存在しない領域で最も信頼性が低くなる）から得られるか、テストハーネスや完全一致採点器のような判定から得られる。こうした判定に対して訓練する手法は**検証可能な報酬からの強化学習**（reinforcement learning from verifiable rewards、RLVR）と呼ばれ、近年の推論モデルの訓練に用いられている (DeepSeek-AI 2025)。判定器は検査対象の性質については厳密であるが、それが本来意図した性質であるとは限らない。

15.8.2.1 KL ペナルティ付きグループステップ

14.1 節の検証器タスクでは、通常の候補の中に `8*8` や `9*9` に答えられるものはなく、9 番目の候補である**ヘッジ**「4、12、64、81 のいずれか」は、通常の各候補に対して参照確率が e^{-4} 倍となっている。`sloppy_checker` は正解文字列を含むあらゆる応答を受理するため、すべてのプロンプトでこのヘッジに報酬を与えてしまう。

```
def sloppy_checker(task):
    """A lenient checker: 1 if the answer appears anywhere in response y."""
    return lambda x, y: float(task.answers[x] in task.resp[y])

task = d2l.VerifierTask()
sloppy = sloppy_checker(task)
print(f'reference: exact score {task.success(task.pi_ref):.3f}, '
      f'lenient score {task.success(task.pi_ref, sloppy):.3f}')
```

```
reference: exact score 0.062, lenient score 0.065
```

この寛容な判定器に対して訓練すると、方策はこのヘッジを見つけ出す。15.3 節のペナルティ $\beta\, D_{\mathrm{KL}}(\pi(\cdot \mid x) \| \pi_{\mathrm{ref}}(\cdot \mid x))$ は報酬に折り込むことができ、サンプリングされた各応答を $r(x,y) - \beta \log\left(\pi_\theta(y \mid x)/\pi_{\mathrm{ref}}(y \mid x)\right)$ で評価すればよい。内部の $\log \pi_\theta$ を微分しても $-\beta\, \mathbb{E}[\nabla_\theta \log \pi_\theta] = 0$ が加わるだけであるため、REINFORCE 推定量は不偏のままである。$T > 1$ トークンの場合、対数確率比はトークンごとの対数確率比の和になり、終端報酬が各トークンでの報酬へと変換されるため、各トークンの重みにはそのトークン以降の対数確率比のみを残せばよい（演習問題 5c）。`kl_group_step` は、このペナルティ付き報酬を 14.8 節のグループステップ `d2l.group_step` に渡す。

```
def kl_group_step(task, theta, K, reward, rng, beta, lr=2.0):
    """d2l.group_step with the reward r - beta log(pi(y|x) / pi_ref(y|x))."""
    pi = task.policy(theta)                    # the policy that samples
    def penalized(x, y):
        return reward(x, y) - beta * np.log(pi[x, y] / task.pi_ref[x, y])
    return d2l.group_step(task, theta, K, penalized, rng, lr)
```

β はどれほど大きくすべきだろうか。最適解は報酬によって参照方策を傾けた分布 $\pi^\star(y \mid x) = \pi_{\mathrm{ref}}(y \mid x)\, e^{r(x,y)/\beta}/Z(x)$ である（15.3 節）。`8*8` では寛容な判定器がヘッジにのみ報酬を与えるため、通常の候補に対するオッズは e^{-4} から $e^{-4}e^{1/\beta}$ へと上昇する。したがって、ヘッジの方が高確率になるのはちょうど $\beta < 1/4$ のときであり、$\beta = 0.2$ では $e \approx 2.7$ 倍の確率になる。一方、`2+2` と `3*4` では正解にも報酬が与えられるため、オッズは e^{-4} のままであり、しきい値は存在しない。

15.8.2.2 閉形式解とのスイープ比較

$K = 32$ として 400 回の更新にわたる訓練を β の 5 つの値で実行し、最後の 100 個の方策を平均して、寛容スコア・厳密スコア・ヘッジの平均確率を表示する。あわせて、傾斜最適解（tilted optimum；選好値で表せば参照方策に報酬を β で割った値を加えたもの）に対する同じ数値も並べて示す。

```
R = task.reward_table(sloppy)            # sloppy(x, y) for every pair
def scores(task, pi, reward):
    """Expected reward, exact score, mean probability of the hedge."""
    return task.success(pi, reward), task.success(pi), pi[:, -1].mean()

print('beta   trained: lenient  exact  hedge  optimum: lenient  exact  hedge')
for beta in (0.0, 0.1, 0.2, 0.3, 0.5):
    rng, theta, avg = np.random.default_rng(0), task.theta_ref.copy(), 0
    for step in range(400):
        theta = kl_group_step(task, theta, 32, sloppy, rng, beta)
        if step >= 300:                  # average the last 100 policies
            avg = avg + task.policy(theta) / 100
    row = '{:4.1f} {:18.2f} {:6.2f} {:6.2f}'.format(
        beta, *scores(task, avg, sloppy))
    if beta > 0:                         # the tilted optimum in preferences
        pi_star = task.policy(task.theta_ref + R / beta)
        row += '{:18.2f} {:6.2f} {:6.2f}'.format(*scores(task, pi_star, sloppy))
    print(row)
```

```
beta   trained: lenient  exact  hedge  optimum: lenient  exact  hedge
 0.0              1.00   0.50   0.50
 0.1              0.99   0.50   0.49              0.99   0.49   0.50
 0.2              0.60   0.48   0.12              0.60   0.47   0.14
 0.3              0.44   0.40   0.04              0.43   0.39   0.04
 0.5              0.25   0.23   0.01              0.27   0.25   0.01
```

$\beta = 0$ では、方策は満点の寛容スコアを獲得するものの、厳密な評価では 0.50 にとどまり、`8*8` と `9*9` では確率のすべてがヘッジに集中する。$\beta = 0.1$ や 0.2 でもヘッジが依然として優位である。しかし $1/4$ を超えると逆転し、$\beta = 0.3$ では `8*8` におけるヘッジの確率が通常の各候補の約半分になり、$e^{-4}e^{1/0.3} \approx 0.5$ の予測と一致する。その代償として有用な学習も阻害され、$\beta = 0.5$ では厳密スコアが約 1/4 まで低下する。訓練結果の各行は最適解から約 0.02 以内に収まっているが、わずかな系統的ずれが残る。これは $\pi^\star$ の近傍ではグループ内のペナルティ付き報酬がほぼ等しくなり、反復が収束すべきまさにその場所で標準化がステップを拡大してしまうためである。ペナルティはどの β においても抜け穴に価格を付けるだけであり、抜け穴そのものを塞ぐことは決してない。

15.8.3 事後学習の更新則

15.8.3.1 古いサンプルに対する確率比

ここまでの更新はすべて、新しく生成したグループを一度だけ使っていた。しかし大規模な事後学習ではそうはいかない。訓練器とはカーネルや演算精度が異なる推論エンジンが応答を生成し、複数の勾配ステップが各バッチを共有するからである。サンプルは古い方策 $\pi_{\theta_{\text{old}}}$ から得られるため、トークン t には確率比 $\rho_t = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ で重みを付け、15.2 節と同様にクリップする。応答全体に対する確率比 W とその幾何平均 $\bar{\rho}$ は次式で与えられる。

$$W = \prod_{t=1}^{T} \rho_t, \qquad \bar{\rho} = W^{1/T} = \exp\Big(\frac{1}{T}\sum_{t=1}^{T} \log \rho_t\Big). \tag{15.58}$$

積 W は厳密な補正を与えるが、その対数はノイズを含む T 個の項の和になるため、長い応答では何桁にもわたって変動する。グループ系列方策最適化（Group Sequence Policy Optimization、GSPO）は代わりに $\bar{\rho}$、すなわち系列長とともにスケールが増大しない応答あたり 1 つの数値をクリップする (C. Zheng et al. 2025)。

15.8.3.2 非対称なクリッピング許容帯

確率比空間における固定幅の許容帯は、許容する確率の変化量が元の確率によって異なる。$\epsilon_{\text{clip}} = 0.2$ のとき、確率 0.01 のトークンは 0.012 を超えて上昇しても報酬上の利得がなく、確率 0.6 のトークンは 0.72 を超えても利得がない。大半が低頻度トークンからなる数万語の語彙においては、有用な低頻度トークンの確率がゆっくりとしか伸びずエントロピーが低下してしまう。そこで DAPO は上側の上限のみを $1 + \epsilon_{\text{high}}$（ただし $\epsilon_{\text{high}} = 0.28$）へと引き上げる (Q. Yu et al. 2025)（演習問題 6）。

15.8.3.3 グループ相対方策最適化

グループ相対方策最適化（Group Relative Policy Optimization、GRPO）(Shao et al. 2024) は、これらの要素を組み合わせた手法である。プロンプトごとに K 個の応答を $\pi_{\theta_{\text{old}}}$ からサンプリングし、応答 j のすべてのトークンに 14.8 節の標準化された重み $\hat{A}_j = (r_j - \bar{r})/(\sigma_r + 10^{-8})$ を与える。ここで、$\bar{r}$ と σ_r はグループ内の報酬の平均と標準偏差である。そして同じグループ上で複数エポックにわたり、次の目的関数を最大化する。

$$\frac{1}{K}\sum_{j=1}^{K}\frac{1}{T_j}\sum_{t=1}^{T_j}\Big[\min\big(\rho_{j,t}\,\hat{A}_j,\ \text{clip}(\rho_{j,t}, 1-\epsilon_{\text{clip}}, 1+\epsilon_{\text{clip}})\,\hat{A}_j\big) - \beta\, k_3(j,t)\Big],$$

ここで、T_j は応答 j の長さであり、$k_3 = \pi_{\text{ref}}/\pi_\theta - \log(\pi_{\text{ref}}/\pi_\theta) - 1$ はサンプリングされたトークンにおいて評価される。最小値演算は、確率比が有利な側で許容帯を超えた時点でそのサンプルの勾配を切り捨てる。クリッピングはサンプリング方策と比較して各更新を制限し、ペナルティは固定された参照方策と比較して最適解そのものを変化させる。

`grpo_step` は、すべての応答が 1 トークンである検証器タスク上でこの目的関数を実装したものである。`num_epochs` 回の各パスでは、サンプリング方策に対する確率比を計算し、クリップされた項が依然として θ に依存するサンプル（`live`）を残して勾配上昇ステップを 1 回実行する。ここで、$\rho\,\hat{A}$ と $-\beta k_3$ の勾配は、それぞれ $\rho\,\hat{A}$ と $-\beta(1 - \pi_{\text{ref}}/\pi_\theta)$ にスコア $\mathbf{e}_y - \pi_\theta(\cdot \mid x)$ を掛けたものになる。

```
def grpo_step(task, theta, K, reward, rng, beta=0.04, epsilon=0.2,
              num_epochs=4, lr=2.0):
    """One group of K responses per prompt, reused for num_epochs clipped
    ascent steps with the k3 penalty; returns theta and each epoch's ratios."""
    pi_old, n = task.policy(theta), len(task.resp)
    ys = np.stack([rng.choice(n, K, p=p) for p in pi_old])      # (prompts, K)
```

```
    A = np.stack([d2l.standardize(np.array([reward(x, y) for y in ys[x]]))
                  for x in range(len(ys))])
    rows, onehot, ratios = np.arange(len(ys))[:, None], np.eye(n)[ys], []
    for _ in range(num_epochs):
        pi = task.policy(theta)
        rho = pi[rows, ys] / pi_old[rows, ys]
        live = np.where(A > 0, rho < 1 + epsilon, rho > 1 - epsilon)
        u = live * A * rho - beta * (1 - task.pi_ref[rows, ys] / pi[rows, ys])
        theta = theta + lr * np.einsum('xk,xkn->xn', u, onehot - pi[:, None]) / K
        ratios.append(rho)
    return theta, ratios
```

プロンプトあたり 32 個の応答からなる 1 つのグループを 4 エポックにわたって再利用し、各エポックにおける確率比の範囲と許容帯 $[0.8, 1.2]$ の外に出た割合を表示したうえで、同じグループに対する 1 ステップのみの更新と比較する。

```
epsilon = 0.2                           # grpo_step's half-width of the band
theta, ratios = grpo_step(task, task.theta_ref, 32, task.verify,
                          np.random.default_rng(0), epsilon=epsilon)
for epoch, rho in enumerate(ratios, 1):
    print(f'epoch {epoch}: ratios from {rho.min():.2f} to {rho.max():.2f}, '
          f'{np.mean(abs(rho - 1) > epsilon):.0%} outside the band')
theta_1, _ = grpo_step(task, task.theta_ref, 32, task.verify,
                       np.random.default_rng(0), num_epochs=1)
print(f'exact score: reference {task.success(task.pi_ref):.3f}, one epoch '
      f'{task.success(task.policy(theta_1)):.3f}, four epochs '
      f'{task.success(task.policy(theta)):.3f}')
```

```
epoch 1: ratios from 1.00 to 1.00, 0% outside the band
epoch 2: ratios from 0.81 to 1.97, 5% outside the band
epoch 3: ratios from 0.74 to 2.23, 14% outside the band
epoch 4: ratios from 0.75 to 2.39, 30% outside the band
exact score: reference 0.062, one epoch 0.111, four epochs 0.137
```

第 1 エポックでは方策がまだ動いていないため、すべての確率比は 1 である。1 ステップ後には正解の確率比はすでに 1.5 を超えている。クリッピングは 1 回のステップ幅そのものを抑えるのではなく、許容帯を超えた増加にそれ以上の報酬を与えないようにするだけだからである。その後、許容帯の外に出る割合は増えていき、ほぼすべてのグループで第 2 エポックより第 4 エポックの方が大きくなる。40 個のグループで試したところ、4 エポックの更新は常に 1 エポックよりも高い厳密スコアで終了した。

k_3 項は、報酬に折り込んだペナルティとは異なる仕方で方策を抑制する。π_θ のもとでの期待値は $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}})$ であるが、サンプルを固定したまま微分すると、順方向ダイバージェンス $D_{\mathrm{KL}}(\pi_{\mathrm{ref}} \| \pi_\theta)$ の期待勾配を持つようになる (Tang and Munos 2025)（演習問題 5）。15.3.3.5 節で見た 2 つの向きは、抜け穴に対して大きく異なる振る舞いを示す。8*8 において $\beta = 0.5$ としたとき、逆方向の最適解はヘッジの確率を 0.017 に抑えるのに対し、順方向の最適解では 0.50 に達する。順方向ダイバージェンスは、参照方策の通常の候補を無視することにはペナルティを課すが、参照方策がめったに選ばなかった候補へ確率を移すことには課さないためである。

広く使われている 2 つの派生手法は、これらの設計選択を調整している。DAPO はペナルティを省き、上側のクリップを引き上げ、報酬がすべて等しいグループを破棄する (Q. Yu et al. 2025)。Dr. GRPO は σ_r および応答長による除算（14.8.3.4 節）を取り除く (Z. Liu et al. 2025)。

15.8.3.4 直接選好最適化

傾斜最適解の式を逆に解くと、報酬は次のように表される。

$$r(x, y) = \beta \log \frac{\pi^\star(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x), \tag{15.59}$$

したがって、あらゆる方策は（比較からは観測できないプロンプトのみの項を除いて）自らを最適とする報酬を定義する。これを Bradley-Terry 尤度に代入すると、報酬の当てはめはサンプリングも報酬モデルも不要な方策自体の分類損失へと変わる。これが**直接選好最適化**（direct preference optimization、DPO）である (Rafailov et al. 2023)。そのデータは固定された比較データの集合であるため、15.7 節の論点をそのまま引き継ぐことになる。

本節の議論はすべて、1 ターンかつ単一の終端報酬という前提に立っており、それゆえに応答全体に対する文脈付きバンディット（contextual bandit、14.4 節）となっている。ツール呼び出し、環境からのフィードバック、複数ターンの対話が加わると遷移カーネル（transition kernel）が再び現れ、それに伴ってクリティックと信頼割り当て問題も復活する。「言語モデル」パートでは大規模な事後学習を扱い、15.9 節では派生手法とプロジェクトをまとめる。

本章は、数十万ステップを要した振り子の課題から始まった。CartPole では 1 ステップのクリティックはステップ数を節約できなかったが、バッチあたり 4 エポックの PPO はステップ数を半減させ、DQN は 35,000〜45,000 ステップを要した。Pendulum ではソフトアクタークリティックが数千ステップ、学習されたモデルが計算量と引き換えに数百ステップで済んだ。さらにオフライン学習器は新たなステップを一切必要とせず、グループ上での 4 エポックの反復は同じサンプルから方策をより遠くまで進めた。こうした節約はいずれも、誤りうる何らかの推定に依拠している。すなわち、クリティック、古いバッチ、モデル、ログ、あるいは判定器である。

15.8.4 まとめ

応答はトークンの軌跡であり、応答単位とトークン単位の両視点は同一の勾配を与える。単一の終端スコアはすべてのトークンに重み $r - b(x)$ を与え、その分散は各トークンアドバンテージの分散の和になる。プレフィックス価値はクリティックや再生成のコストと引き換えに各トークン固有の信頼割り当てを与えるが、応答重みが不偏であるためクリティックは任意である。KL ペナルティは判定器の抜け穴に対し、しきい値も含めて傾斜最適解が予測するとおりの価格を付ける。GRPO はグループ重み、クリップされた確率比、および k_3 ペナルティを組み合わせるが、その勾配は順方向ダイバージェンスの勾配であるため、ヘッジをほとんど抑制しない。

15.8.5 演習問題

1. **軌跡内のツール出力**。ツールを呼び出すモデルがトークン列 $y^{(1)}$ を出力し、ツールの出力 $o \sim P_{\text{tool}}(\cdot \mid x, y^{(1)})$（これは θ に依存しない）を受け取った後、トークン列 $y^{(2)}$ を出力するとする。検証器は $\tau = (x, y^{(1)}, o, y^{(2)})$ に対して一度だけスコアを付け、報酬 $r(\tau)$ を与える。
   1. τ の確率とその対数の勾配を書き下せ。どのトークンがスコア項に寄与するか。
   2. 系列全体に対して対数確率を評価する訓練ループは、ツールの出力に対するモデル自身の確率を用いて $\big(r(\tau) - b(x)\big)\, \nabla_\theta \log \pi_\theta(o \mid x, y^{(1)})$ という項を加えてしまう。重みが定数であってもこの項の期待値は一般に 0 にならないことを示し、スコアの平均が 0 になるという証明のどの段階が破綻するかを述べよ。この項はモデルに何を学習させてしまうか。
2. **2 桁の応答**。15.8.1.2 節の 2 つの桁が、それぞれ独自の選好値のソフトマックスである分布 $\mathbf{p}_1$ と $\mathbf{p}_2$ から抽出され、検証器は正解に対してのみ 1 を支払うとする。

   1. 応答重み $r - V(x)$ とトークンアドバンテージが、各位置の選好値に対して同じ期待勾配を与えることを示せ。どの条件付き期待値によって両者は一致するか。
   2. $A(s_1, a_1)$ と $A(s_2, a_2)$ が無相関であることを示し、T トークンの応答に対する同様の結果を述べよ。
   3. 確率 0.9 を第 1 位置の数字 1 と第 2 位置の数字 2 に与え、他のすべての数字に $1/90$ を与える方策において `token_credit` を呼び出せ。このとき応答重みを $A(s_1, a_1)$ に置き換えると十の位の更新の分散はどれだけ減少するか。また、一様方策のときより減少幅が小さいのはなぜか。
3. **ヘッジに対する 2 つのしきい値**。$\pi^\star$ を、報酬 `sloppy` と参照方策 `task.pi_ref` に対する傾斜最適解とする。
   1. `8*8` と `9*9` では、ヘッジが `sloppy` によって報酬を与えられる唯一の応答である。$\pi^\star$ のもとでのヘッジの確率を β の関数として書き下せ。どの β で他の 8 つの各応答と同じ確率になり、どの時点で確率全体の半分を占めるようになるか。この 2 つのうち、本文で述べたしきい値はどちらか。
   2. `2+2` と `3*4` では、正解の応答とヘッジの両方に報酬が与えられる。$\pi^\star$ は $\beta \to 0$ のときヘッジにどれだけの確率を与えるか。
   3. $\beta = 0$ での訓練は、`2+2` と `3*4` においてヘッジにほとんど確率質量を置かなかったが、報酬のある 2 つの応答へのどのような確率配分であっても正則化なしの目的関数を最大化する。`d2l.group_step` の更新のどの性質が、訓練の到達した配分を選択したのか。
4. [code] **有限インスタンス上の DPO**。●同じプロンプトに対してともに `task.pi_ref` から抽出された 2 つの応答間の選好が、報酬 `sloppy` を持つ Bradley-Terry モデルに従うとする。選好された応答と棄却された応答をそれぞれ y_w、y_l と書き、$h_\theta(y) = \log\big(\pi_\theta(y \mid x)/\pi_{\text{ref}}(y \mid x)\big)$ とおく。
   1. Equation 15.59 を Bradley-Terry モデルに代入し、1 ペアあたりの負の対数尤度が $-\log \sigma\big(\beta\, h_\theta(y_w) - \beta\, h_\theta(y_l)\big)$ になることを示せ。ここで σ はロジスティック関数である。なぜ $Z(x)$ は現れないのか。
   2. 無限個のペアがあるとき、期待損失が傾斜最適解によって最小化されることを示せ。このことは演習問題 3a のしきい値に対して何を意味するか。
   3. プロンプトあたり 1000 個のラベル付きペアを抽出し、$\beta = 0.5$ における平均損失を、ステップサイズ 10 の勾配降下法で `task.theta_ref` から出発して選好テーブル上で最小化せよ。5 つのシードにわたり、200、2000、20000 ステップ後の各プロンプトにおけるヘッジの確率を報告せよ。ヘッジが比較で一度も負けないプロンプトでは何が起きるか。また、なぜ β はそれを防げないのか。
5. **損失としての KL 項**。●●GRPO はサンプリングされた各トークンで βk_3 を引き、トークンを固定したまま微分する (Shao et al. 2024)。Tang and Munos (2025) は、このような損失が何を最適化するかを解析している。
   1. 1 トークンの応答に対して $h = \log\big(\pi_\theta(y \mid x)/\pi_{\text{ref}}(y \mid x)\big)$ と書き、損失 $k_1 = h$、$k_2 = h^2/2$、$k_3 = e^{-h} + h - 1$ を比較せよ。$y \sim \pi_\theta$ のとき、k_1 の期待勾配は 0 であり、k_2 の期待勾配は $\nabla_\theta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})$、$k_3$ の期待勾配は $\nabla_\theta D_{\text{KL}}(\pi_{\text{ref}} \| \pi_\theta)$ であることを示せ。
   2. 順方向ペナルティ $\beta\, D_{\text{KL}}(\pi_{\text{ref}} \| \pi)$ のもとでは、最大点は $\pi(y) = \beta\, \pi_{\text{ref}}(y)/\big(m - r(y)\big)$ となる。ここで乗数 $m > \max_y r(y)$ は確率の総和を 1 にする定数である（演習問題 7a）。`8*8` で報酬 `sloppy` を用いた場合について、m を $\beta \in \{0.1, 0.25, 0.5\}$ で数値的に求め、ヘッジの確率を傾斜最適解のもとでの確率と比較せよ。両者の差は β にどう依存するか。
   3. T トークンの応答に対し、h_t をトークン t の対数確率比と書き、Equation 15.57 を用いて次式を示せ。

$$\nabla_\theta D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid x) \| \pi_{\mathrm{ref}}(\cdot \mid x)\big) = \mathbb{E}\Big[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{T} h_{t'}\Big]. \tag{15.60}$$

トークンごとの損失 k_2 の和はどの項を見落とすか。また、`kl_group_step` のようにペナルティを報酬に折り込むと、なぜこの見落としが回避されるのか。

6. **非対称な許容帯**。あるトークンがサンプリング方策のもとで確率 0.01 を持つとする。各バッチがその確率を、クリッピングがそれ以上の増加への報酬を打ち切る倍率 $1+\epsilon_{\text{high}}$ だけちょうど引き上げ（15.8.3.2 節）、次のバッチは更新後の方策からサンプリングされるとする。
   1. 確率を 0.01 から 0.5 まで引き上げるのに、$\epsilon_{\text{high}} = 0.2$ では何バッチかかり、$\epsilon_{\text{high}} = 0.28$ では何バッチかかるか。
   2. 2 行動の問題において、もう一方の行動の初期確率は 0.99 である。このような 1 バッチの更新後における、その行動の確率比を計算せよ。まれにしかサンプリングされない行動が確率を増やせる速さを制限しているのは、どちらの行動に適用された許容帯のどちら側か。
   3. 2 行動の場合、上側のクリップが高確率側の行動（確率 p）の増加を制限するのは $(1+\epsilon_{\text{high}})\,p < 1$ の間だけである。どのような p に対して ϵ_{high} を 0.2 から 0.28 へ引き上げると高確率側の行動まで速く増加するようになり、それはエントロピーにどのような影響を与えるか。また、大半が低頻度トークンである数万語の語彙を持つ言語モデルでは、同じ変更によってなぜエントロピーが高く保たれるのか。
7. **長さ正規化された確率比**。●GSPO (C. Zheng et al. 2025) は、重みとしてもクリップの内部でも、トークンごとの確率比を Equation 15.58 の幾何平均 $\bar\rho = W^{1/T}$ に置き換える。サンプリング方策のもとでのトークンごとの対数確率比 $\log \rho_t$ を、平均 $-d$、分散 v の独立同分布に従う確率変数としてモデル化せよ。ここで d はトークンあたりのダイバージェンス $D_{\mathrm{KL}}(\pi_{\theta_{\text{old}}} \| \pi_\theta)$ である。
   1. $\log W$ が平均 $-dT$、分散 vT を持つ一方、$\log \bar\rho$ は平均 $-d$、分散 v/T を持つことを示せ。
   2. $\mathbb{E}[\bar\rho] \le 1$ であること、および $\mathbb{E}[\bar\rho] \to e^{-d}$（$T \to \infty$ のとき）を示せ。重要度重み W は $\mathbb{E}_{\pi_{\theta_{\text{old}}}}[W f] = \mathbb{E}_{\pi_\theta}[f]$ を応答のあらゆる関数 f に対して満たす。$\bar\rho$ もこれを満たすか。
   3. GSPO は $\bar\rho$ を $[1-3\cdot 10^{-4}, 1+4\cdot 10^{-4}]$ にクリップし、比較対象の GRPO ベースラインはトークンごとの確率比を $[1-0.2, 1+0.27]$ にクリップする。(a) を用いて、系列レベルの許容帯が有効に働くためにはるかに狭くなければならない理由を説明せよ。

15.9 関連文献

本章の各節では、それぞれ 1 つの手法を取り上げ、小規模なタスクでその性能を測定した。近年の論文を開けばこうした手法の変種に出会うが、その大半が変更しているのは、更新の細部、更新が利用できるデータ、あるいは課す制約のうち 1 点だけである。本ページでは、読者が最も目にする機会の多い変種を本章の構成順に取り上げ、それぞれが扱う問題と変更点を示す。最後に参考資料を挙げ、15.8 節の事後学習（post-training）ループを完全に構築するプロジェクトで締めくくる。

15.9.1 実践における方策最適化

15.2 節の PPO は CartPole の上限に達するが、より難しいタスクで論文の報告結果を再現する実装は、論文にはほとんど明記されない多数の細かな選択においてこれと異なっている。Huang, Rousslan Fernand Julien Dossa, et al. (2022) は、そうした選択のうち 37 個を参照コードの各行と対応づけた。更新を最も大きく変えるものは次のとおりである。

細部	変更内容
観測と報酬の正規化	移動統計量によりネットワークの入力とクリティックのターゲットを単位スケール付近に維持
ミニバッチごとのアドバンテージ正規化	14.8 節のバッチごとの標準化を各ミニバッチに適用
0 までアニーリングする学習率	訓練終盤に向けたステップの縮小
直交初期化と小さな方策ヘッド	初期方策がほぼ一様になり、初期の比を 1 付近に維持
大域的な勾配ノルムのクリッピング	パラメータ空間における各ステップの大きさも制限
アクターとクリティックの単一損失	15.2 節のようにクリティックを先に適合させるのではなく、方策・価値・エントロピーの各項で各ミニバッチのパスを共有
価値損失のクリッピング	一部の実装のみが備える、クリティックの変化に対するクリップ幅
近似 KL による早期停止	測定された KL がしきい値を超えた時点でのバッチ内エポックの打ち切り

これらの中でどれが重要かを測定した研究が 2 つある。Engstrom et al. (2020) は、これらの工夫がなければ PPO の比がクリップ幅から自由に逸脱すること、また同じ細部を備えた TRPO は PPO と同程度の性能を示すことを明らかにした。したがって、PPO の優位性の多くは目的関数の周囲にあるコードに由来する。Andrychowicz et al. (2021) は、こうした選択肢のグリッド上で数十万個のエージェントを訓練し、その結果を推奨事項としてまとめた。観測の正規化は大半のタスクで有効であった一方、価値損失のクリッピングが有効だとする結果はどちらの研究でも見られなかった。保守されている単一ファイルの実装を 15.2 節と並べて読めば、この表がどのようにコードへ落とし込まれるかがわかる。

PPO の原論文では、ステップを制限する 2 つ目の方法も提示されている (Schulman et al. 2017)。旧方策からの変化に対する KL ペナルティであり、その重みは目標とする KL を超えたバッチの後では 2 倍にし、下回ったバッチの後では半分にする。同論文の実験ではクリッピングより劣っていたが、これは 15.3 節における固定された参照方策へのペナルティと密接な関係にある。

分散訓練では、多数のアクターが学習器より数回の更新分だけ遅れた方策のコピーを用いてデータを収集する。IMPALA の V-trace ターゲット (Espeholt et al. 2018) は、1 で打ち切った重点比を用いて 15.1 節の n ステップターゲットを補正する。この打ち切りにより分散は有界に保たれ、その代償としてターゲットはアクターの方策と学習器の方策の中間にある方策の価値を推定することになる。

シミュレータ自体が GPU 上で動作する場合、数千個の環境が並列にステップを進めるため、サンプル数ではなく実経過時間がコストとなる (Makoviychuk et al. 2021)。このとき、非常に大きなバッチとドメインランダム化（domain randomization）を組み合わせた PPO が脚式ロボットや器用なロボットにおける標準的なレシピとなる (Rudin et al. 2022)。一方、実機上でステップを実行する場合には、以下で述べるオフポリシー手法やモデルベース手法が好まれる。

15.9.2 DQN 以降の価値学習

15.4 節の DQN は CartPole で数万ステップを要し、ノイズを含む推定値の最大値をとるため過大評価が生じる。後続の手法はそれぞれ 1 つの箇所を変更している。マルチステップターゲット（multi-step target）は 15.1 節の n ステップ収益を Q 学習のターゲットに組み込むことで、より少ない更新回数で報酬が前の状態へ伝わるようにする。優先度付き経験再生（prioritized replay）は TD 誤差に比例して遷移をサンプリングし、それによって生じる統計的バイアスを打ち消すよう重み付けし直す (Schaul et al. 2016)。デュエリングネットワークは出力を状態価値と行動ごとのアドバンテージに分割し、共有された価値があらゆる行動の遷移から学習できるようにする (Wang et al. 2016)。

分布型手法（distributional methods）は、分布に対するベルマン方程式を用いて、収益の平均ではなく分布全体を学習する (Bellemare et al. 2017)。Lyle et al. (2019) は、表形式および線形の設定ではこれらの手法が学習する期待値が通常の更新によるものと一致することを示し、その改善がより情報量の多いターゲットによる深層ネットワークの訓練の仕方に由来することを明らかにした。Rainbow (Hessel et al. 2018) は、15.4 節の二重推定量（double estimator）を含むこれら 6 つの変更を組み合わせた手法である。Atari におけるアブレーションでは、優先度付き経験再生とマルチステップターゲットが最も重要な要素、二重推定量とデュエリングヘッドが最も重要度の低い要素と位置づけられた。その後の知見も Lyle らと同じ方向を示している。スカラーのターゲットを離散化した収益の隣接ビンへ分散させ、交差エントロピー損失で訓練したクリティックは、二乗損失ではスケールしない大規模なネットワークにもスケールする (Farebrother et al. 2024)。後述する Dreamer と TD-MPC2 は、このような損失を用いて価値関数を訓練している。

Hasselt et al. (2018) は、多様な設定で訓練された深層 Q ネットワークにおいて、15.4 節の死の三要素（deadly triad）がどれほどの頻度で問題を引き起こすかを測定した。際限のない発散はまれであったが、達成可能なあらゆる収益を一時的に上回るような推定値の上昇は頻繁に見られ、マルチステップターゲットとターゲットネットワークはそのようなエピソードを減らした。近年の 2 つの設計は、15.4 節の修復策を完全に排している。PQN (Gallici et al. 2025) はネットワーク内に層正規化を備え、多数の並列環境から得られるバッチで訓練し、リプレイバッファもターゲットネットワークも用いない。ストリーミング手法 (Elsayed et al. 2024) は、観測と報酬を正規化し各更新の大きさを制限したうえで、到着する各遷移から 1 度だけ学習する。したがって、経験再生と凍結されたコピーは、相関したデータと移動するターゲットに対する修復策の 1 つではあっても、唯一のものではない。

15.9.3 2018 年以降の連続制御

TD3 (Fujimoto et al. 2018) は、決定論的なアクターがクリティックの誤差を登ってしまう DDPG (Lillicrap et al. 2016) に対して 3 つの変更を加えた。すなわち、15.5 節も採用したツインクリティック、クリティックより低い頻度で更新されるアクター、そしてアクターがクリティックの鋭いピークを悪用できないようにするためのターゲット行動へのノイズ付加である。

それ以降の進展の大半は、クリティックの推定値を暴走させずに環境 1 ステップあたりの勾配ステップ数、すなわち 15.5 節の更新比（update ratio）を増やすことから生まれている。REDQ (X. Chen et al. 2021) はクリティックの大規模なアンサンブルを保持し、ランダムに選んだ 2 個の最小値をとることで、1 ステップあたり多数の更新を維持する。DroQ (Hiraoka et al. 2022) はこのアンサンブルを、ドロップアウトと層正規化を備えた 2 個のクリティックで置き換える。CrossQ (Bhatt et al. 2024) は例外であり、ターゲットネットワークを持たず 1 ステップあたり 1 回の更新で REDQ に匹敵する。そのクリティックはバッチ正規化を用い、現在の状態行動対と次の状態行動対を 1 つのバッチにまとめてネットワークへ通すことで、両者が同じ統計量を参照するようにしている。BRO (Nauman et al. 2024) はクリティックを層正規化付きの幅の広い残差ネットワークへと大規模化し、楽観的探索（optimistic exploration）を加える。PQN との共通点は、クリティック内部の正規化である。

更新比を制限する要因は過大評価だけではない。Nikishin et al. (2022) は、高い更新比で訓練されたエージェントが最初期のデータに過学習して後続のデータからうまく学習できなくなる初頭バイアス（primacy bias）を見出し、リプレイバッファを保持したままネットワークを周期的にリセットすることで学習能力が回復することを示した。このようなリセットを行えば、収益の改善が止まるまで更新比をさらに大きく引き上げられる (D'Oro et al. 2023)。

15.9.4 モデル、想像、探索

15.6 節のプランナは実ステップごとに数千回のモデル評価を費やし、データ近傍でのみモデルを信頼するが、以下の手法は学習されたモデルを異なる形で利用する。Dyna (Sutton 1990) はモデルを用いてモデルフリー学習器のための追加の遷移を想像する手法であり、14.3 節のカウントによるモデルが実ステップの合間に Q 学習の更新を駆動する。MBPO (Janner et al. 2019) は、リプレイバッファ内の状態から数ステップだけ分岐する短い想像上のロールアウトにとどめ、それをソフトアクタークリティックのバッチに混ぜることで、モデルの累積誤差（compounding error）を小さく抑える。前項で述べた REDQ はモデルなしで MBPO に匹敵するサンプル効率を達成しており、MBPO の改善の多くが実ステップあたりの更新回数の増加に由来することを示唆している。

PETS (Chua et al. 2018) は確率的ネットワークのアンサンブルを適合させ、そこからのサンプルを用いて交差エントロピー法（cross-entropy method）により計画を立てる。アンサンブルの各メンバーはデータが密な領域では一致し、それ以外では食い違うため、その不一致が計画を信頼すべきでない領域を示す目印となる。Dreamer (Hafner et al. 2025) は画像から学習した潜在空間上でダイナミクスのリカレントモデルを学習し、多数のドメインにわたり同一のハイパーパラメータ設定のまま、想像上の潜在軌道のみを用いてアクターとクリティックを訓練する。

学習された価値を用いた探索は、15.6 節のループを完結させる。AlphaZero (Silver et al. 2017, 2018) はゲームのルールをモデルとし、葉に価値ネットワークを置いて探索を行い、探索による選択を方策へと蒸留する。MuZero (Schrittwieser et al. 2020) は観測ではなく報酬・価値・方策のみを予測するモデルを学習し、そのモデル内で探索する。TD-MPC2 (Hansen et al. 2024) は潜在モデル内で短いホライズンにわたる計画を立て、各計画を学習された価値で締めくくる。

新たなデータを収集できない場合でも、データが希薄な領域でオフライン学習器がモデルを過信しない限り、学習されたモデルは役に立つ。MOPO (Yu et al. 2020) と MOReL (Kidambi et al. 2020) は、モデルのアンサンブルが食い違う領域で想像上の報酬にペナルティを課す。これは 15.7 節の悲観主義（pessimism）をモデルに適用したものである。

15.9.5 大規模なオフライン手法

15.7 節のペナルティは、ログ内の状態行動対の出現回数の平方根に応じて縮小する。連続行動では、学習された方策が提案するほぼすべての行動の出現回数が 0 になるため、深層オフライン手法はこの出現回数をネットワークが推定できる量で置き換える。各手法の違いは、何を制約するかにある。

手法	制約対象	方法
BCQ (Fujimoto et al. 2019)	方策がとり得る行動	ログ内の行動の生成モデルが候補を提案し、方策がその中から選択
CQL (Kumar et al. 2020)	価値	方策の行動での Q を引き下げ、ログ内の行動での値を引き上げる損失項

手法	制約対象	方法
IQL (Kostrikov et al. 2022)	バックアップが参照する行動	V をログ内の行動のみにわたる Q の上位エクスペクタイルに適合させ、アドバンテージ重み付き回帰により方策を抽出
TD3+BC (Fujimoto and Gu 2021)	データに対する方策の距離	TD3 のアクター損失へのクローニング項の追加

もう 1 つの系統である収益条件付き系列モデル（return-conditioned sequence models）については、その限界とともに 15.7 節で論じている。

これらの手法の大半はガウス方策または決定論的方策を適合させるが、デモンストレーションはそのような形状をしていないことが多い。人間の操作者は同じタスクを複数のやり方で解くため、2 通りのやり方に 1 つのガウス分布を適合させると、両者を平均して誰もとらなかった行動を出してしまう。拡散方策（diffusion policies）はデモンストレーションを模倣する手法であり (C. Chi et al. 2023)、複数のモードを同時に表現できる第 17 章の生成モデルを用いることで、ロボティクスにおける行動クローニング（behavior cloning）の標準的な形態となっている。さらにオフライン手法は、このようなモデルを方策として採用し、上述のようにログへ向けて制約を課す (Z. Wang et al. 2023)。その後には、次に述べるオンラインのファインチューニングが続けられることも多い。

多くの応用では、既存のログに加えて少量の新しいデータも収集できる。Cal-QL (Nakamoto et al. 2023) は、悲観的な価値がロギング方策などの参照方策の価値を下回らないように較正することで、価値が過度に低い場合にオンラインのファインチューニング開始時に生じる収益の落ち込みを防ぐ。RLPD (Ball et al. 2023) はオフライン訓練を省略し、各バッチの半分をログから抽出しつつ前項までの正規化されたクリティックを用いて、最初からオフポリシー学習器をオンラインで走らせる。

15.9.6 事後学習の変種

15.8 節では、14.8 節のグループ重み、PPO のクリップされた比、および 15.3 節の KL ペナルティから事後学習の更新を組み立て、DAPO、Dr. GRPO、GSPO を挙げた。推論モデル（reasoning models）の規模で訓練すると (Shao et al. 2024; DeepSeek-AI 2025)、この更新は 1 トークンのタスクでは現れない問題に直面し、各変種がそれに対処している。DAPO (Q. Yu et al. 2025) はさらに、まず各応答内で平均するのではなくバッチ内の全トークンにわたって損失を平均することで、長い応答がその長さに比例して寄与するようにし、長さ上限で打ち切られた応答の報酬にシェイピングを施す。Dr. GRPO (Z. Liu et al. 2025) は、取り除く 2 つの除算に 2 つのバイアスの原因を帰着させている。応答長で割ると長い誤答のトークンあたりペナルティが縮むため訓練によって誤答が長くなり、グループの標準偏差で割るとグループ内がほぼ全問正解または全問不正解であるプロンプトに最大の重みが与えられてしまう。GSPO (C. Zheng et al. 2025) は、トークンごとの比がとりわけノイジーになる大規模な混合エキスパート（mixture-of-experts）モデルの安定性のために導入された。

プロセス報酬（process rewards）は、報酬の出所を変える。Lightman et al. (2024) は個々の推論ステップに対する人間のラベルで報酬モデルを訓練し、最終回答のみで訓練されたモデルよりも適切に解法を判定できることを示した。学習されたプロセス報酬は、15.8 節のプレフィックス価値と同様に各ステップへ個別のクレジットを与える一方で、15.3 節で測定したような、学習された報酬が持つ悪用され得る誤差も伴う。

15.9.7 本章の先の話題

これら 2 つの章で置いた 4 つの仮定は成り立たないことがある。複数のエージェントが同時に学習する場合、各エージェントの環境には他のエージェントが含まれるため、定常なものは何もなくなる。自己対戦（self-play）と探索を組み合わせた手法は、ヘッズアップ・ノーリミット・ポーカーにおいて人間を超えるプレイに到達した (Brown and Sandholm 2017)。エージェントが状態ではなく観測しか得られない場合、14.1 節のマルコフ性は崩れ、方策にはメモリ、リカレントネットワーク、あるいは状態上の信念（belief）が必要となる (Kaelbling et al. 1998)。タスクが変化する場合、エージェントは少数のエピソードから新しいタスクへ適応することを学習したり（メタ学習、meta-learning）、多数のステップにわたって行動するサブ方策とそれらを切り替える方策を合わせて学習したりできる（階層型学習、hierarchical learning）。報酬が極めて疎でありランダムな行動ではほとんど到達できない場合、ϵ グリーディやエントロピーボーナスでは探索が遅すぎるため、エージェントは未見の状態を自ら探しに行かなければならない。ネットワークは 14.4 節のボーナスに変換できる訪問回数を保持しないため、深層手法はネットワークが供給できる量で代用する。すなわち、密度モデルの擬似カウント（pseudo-count）(Bellemare et al. 2016)、次の状態を予測するネットワークの誤差 (Pathak et al. 2017)、あるいは固定されたランダムネットワークの出力を予測するネットワークの誤差 (Burda et al. 2019) であり、いずれもエージェントの経験が希薄な領域で報酬に加算される。あるいは、価値関数のアンサンブルからランダムに 1 つ選んだメンバーに 1 エピソード全体を通して従う方法もあり、データが希薄な領域ではメンバー間の予測が食い違うことを利用する (Osband et al. 2016)。Murphy の概説 (Murphy 2025) はこれら 4 つすべてを概観している。

15.9.8 参考書、ベンチマーク、コードベース

第 14 章の基礎資料（CleanRL の単一ファイル実装 (Huang, Rousslan F. J. Dossa, et al. 2022) や PPO の実装上の細部をまとめたカタログ (Huang, Rousslan Fernand Julien Dossa, et al. 2022) を含む）に加え、以下の資料が本章の各節に対応している。

- Sutton と Barto (Sutton and Barto 2018) の第 11〜13 章は、15.1 節の λ 収益と適格度トレース（eligibility traces）、それらに基づくアクタークリティック法、および 15.4 節の死の三要素と、その 3 要素のうち 1 つを外したときに何が証明できるかを扱っている。また第 8 章は、Dyna と学習されたモデルを用いた計画（15.6 節）を展開している。
- Engstrom et al. (2020) による統制実験は 15.2 節に対応している。コードレベルの選択を揃えれば PPO と TRPO はほぼ同等の性能を示すため、実用上の PPO の優位性の多くは目的関数が言及しない細部に由来する。
- RL Baselines3 Zoo (Raffin et al. 2021) は、PPO（15.2 節）、DQN（15.4 節）、SAC（15.5 節）について環境ごとに調整済みのハイパーパラメータを提供している。
- rliable (R. Agarwal et al. 2021) は、少数のシードにわたる深層強化学習の実行結果を比較するための層化ブートストラップ信頼区間と性能プロファイルを提供している。
- Levine et al. (2020) のサーベイ論文は、15.7 節の内容を表形式での診断から深層オフライン手法へと引き継いでいる。
- The RLHF Book (Lambert 2026) は、15.3 節と 15.8 節で組み立てた更新を用いる事後学習手法を、本格的な言語モデルの規模で展開している。

15.9.9 プロジェクト

15.8 節では事後学習の更新を組み立て、それを単一のグループに適用した。このプロジェクトでは、その周囲に訓練ループを構築し、本章で厳密に計算できる結果と照合して検証する。

1. **タスク**。検証器タスクの 4 つのプロンプトを用い、15.8 節と同様に 2 つの数字トークンで回答させて、`04`、`12`、`64`、`81` が正解の応答となるようにせよ。方策は、プロンプトごとの十

の位の数字に対する選好度（preference）と、プロンプトおよび十の位の数字ごとの一の位の数字に対する選好度の表とし、参照方策は一様分布とする。

2. **更新**。応答が 2 トークンの場合へ `grpo_step` を拡張せよ。各プロンプトについて 16 個の応答からなるグループをサンプリングし、トークン対数確率を保存して、グループ内で報酬を標準化せよ。続いて、ステップサイズ 1 の勾配上昇法を E エポック実行せよ。目的関数は、$\epsilon_{\text{clip}} = 0.2$ でクリップしたトークンごとの代理目的関数をグループ内の全トークンにわたって平均したものから、β と 15.8 節の k_3 項の積を引いたものとし、各エポックのクリップ率（clip fraction）を記録せよ。
3. **確認事項**。最初のエポックではすべての比が 1 に等しくクリップ率は 0 であるため、$E = 1$ のときこの更新はグループ標準化された重みを用いた REINFORCE ステップと一致する。
4. **再利用の基準**。同じステップサイズかつ $\beta = 0$ のもとで、グループあたり 4 エポックを実行すると、試すどのシードでも、1 エポックの場合に必要なサンプル応答数の約半分で成功率 0.9 に達し、同時にクリップ率は第 1 エポックから第 4 エポックにかけて上昇するはずである。
5. **ペナルティの基準**。1 トークンの検証器タスクにおいて 15.8 節の寛容な判定器を用い、14.8 節の 1 つ抜き（leave-one-out）重みと $\beta = 1/2$ のもとで方策が落ち着くまで訓練せよ。k_3 損失のもとでは、ヘッジにより確率の約半分が逆向きの KL を持つ目的関数の最大点である `8*8` に保たれるはずである（演習問題 5）。一方、ペナルティを報酬に折り込んだ場合には、傾斜最適解（tilted optimum）である 2% 未満にとどまるはずである。これら 2 つの数値を得るために 1 つ抜き重みが必要となる理由と、グループ標準化を用いると何が変わるかを説明せよ。

Chapter 16

敵対的生成ネットワーク

100 次元の確率ベクトル $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ を抽出し、それをニューラルネットワーク $G(\mathbf{z})$ に通して画像を生成するとする。このような生成器ネットワークを記述するだけなら数行のコードで済むが、難しいのはその訓練である。観測されたラベルやトークンの対数尤度（log-likelihood）を最大化する分類器や自己回帰言語モデルとは異なり、画像を直接出力する生成器は、最大化すべき明示的な密度 $q(\mathbf{x})$ を持たない画像上の陰的な分布（implicit distribution）を定義する。サンプリングはできるが尤度を評価できないとき、生成モデル（generative model）をどのように訓練すればよいのだろうか。

分類器を用いれば、サンプルだけから 2 つの分布を比較できることはすでに見てきた。3.6.2 節で共変量シフト（covariate shift）を補正した際には、訓練入力とテスト入力を識別する二値分類器を訓練し、その最適ロジットから 2 つの分布間の対数密度比（log density ratio）を復元した。Goodfellow et al. (2014) は、この 2 サンプル比較を生成モデルの訓練原理へと発展させた。すなわち、**敵対的生成ネットワーク**（generative adversarial network、GAN）では、実データと生成サンプルを識別するように**識別器**（discriminator、または**クリティック**）D を訓練し、同時に、識別器が実データと見分けられないようなサンプルを生成するように**生成器**（generator）G を訓練する。識別器の勾配を通じて、生成器はどちらの密度も明示的に書き下すことなく、自身のサンプルがデータ分布に対して過剰または過少になっている領域を学習する。

汎用的な画像生成においては、拡散モデルやフローマッチングモデル（第 17 章）が単体の GAN にほぼ取って代わったものの、敵対的目的関数（adversarial objective）は依然として現代の生成システムの中核をなす構成要素であり、潜在拡散モデル（latent diffusion model）を支える潜在オートエンコーダや、ニューラル音声ボコーダ、複数ステップの拡散教師モデルから蒸留された 1 ステップ生成器などの訓練に用いられている。

本章では、敵対的モデリングを 3 段階で展開する。まずは目的関数とダイバージェンスから始める。16.1 節では対数損失における最適識別器、そのイェンセン・シャノン・ダイバージェンス（Jensen–Shannon divergence）、および非飽和生成器損失（non-saturating generator loss）を導出する。続く 16.2 節ではこの枠組みを f ダイバージェンスと、積分確率指標（integral probability metric）である最大平均値差異（maximum mean discrepancy、MMD）および 1 次ワッサースタイン距離（Wasserstein-1 distance）へ一般化し、16.3 節ではブラッドリー・テリー（Bradley–Terry）ランキングに基づく相対論的ペア型クリティック（RpGAN）を導入する。次に、16.4 節では Dirac-GAN 上でミニマックス訓練ダイナミクス（training dynamics）を調べ、正則化なしの更新が回転したり外側へ渦巻状に発散したりする理由と、ゼロ中心勾配ペナルティ（R_1 と R_2）が R3GAN のレシピをいかに安定化するかを示す (Huang et al. 2024)。最後に、16.5 節と 16.6 節では、畳み込みアーキテクチャと射影条件付きアーキテクチャを、特徴空間の評

価指標（FID、KID、k-NN の適合率と再現率）とあわせて画像生成へとスケールさせ、16.7 節では拡散蒸留、画像トークナイザ、音声ボコーダにおいて敵対的クリティックが条件付き平均によるぼけをいかに防ぐかを解説する。

記法

Table 16.1 に、本章全体で用いる記法をまとめる。ダイバージェンスとエントロピーは nat を単位として測り、第 37 章に従う。すなわち、D_{KL} はカルバック・ライブラー・ダイバージェンス、JS はイェンセン・シャノン・ダイバージェンス、H は微分エントロピーまたは離散エントロピー、$H_b(t)$ はパラメータ t のベルヌーイ分布の二値エントロピーを表す。

Table 16.1: 本章で用いる記法。

記号	意味	初出	よく使われる別記法
p、q、$m = \frac{1}{2}(p+q)$	データ密度、生成器密度、均衡混合密度	16.1 節	p_{data}、p_g
$x \sim p$、$x' \sim q$、$y \in \{0,1\}$	実サンプル、生成サンプル、出所ラベル（1 = 実データ）	16.1 節	
$z \in \mathbb{R}^{d_z}$、G	潜在ベクトルと生成器ネットワーク（$x' = G(z)$）	16.1 節	
D、$V(D)$	クリティック・識別器の真実性ロジット、ミニマックス価値関数	16.1 節	
$\rho = p/q$、$\lambda = \log(p/q)$	密度比と対数密度比	16.1 節	
w_{sat}、w_{ns}、w_{Rp}	サンプルごとの生成器勾配重み（飽和型、非飽和型、相対論的）	16.1 節、16.3 節	
$\eta = p/(p+q)$	ベイズクラス事後確率 $P(y=1 \mid x)$	16.2 節	
ℓ、$L(\eta)$、$\Delta_\ell(p,q)$	マージン利得、条件付きベイズリスク、ベイズリスクギャップ	16.2 節	
$D_f(p \,\|\, q)$、$k(x,x')$	凸関数 f による f ダイバージェンス、正定値カーネル	16.2 節	
$\Phi(D)$、$d_{\text{Rp}}(p,q)$	相対論的ペアリング目的関数とペアリングダイバージェンス	16.3 節	
R_1、R_2、γ	実サンプルおよび生成サンプルにおけるゼロ中心勾配ペナルティ、ペナルティ重み	16.4 節	
c、e_c、$\phi(x)$	条件付け変数、条件埋め込み、特徴表現	16.5 節、16.6 節	

16.1 敵対的生成ネットワーク

Goodfellow et al. (2014) は、生成器（generator）ネットワークに、実データと生成サンプルを見分けるよう学習する**識別器**（discriminator）または**クリティック**と呼ばれる二値分類器を組み

合わせることで、**敵対的生成ネットワーク**（generative adversarial network、GAN）を導入した。この二値分類器は、3.6.2 節で共変量シフト（covariate shift）の補正に用いたドメイン分類器とよく似ている。

ここで直ちに、2 つの根本的な疑問が生じる。

1. 生成器が識別器を騙せたとき、生成器は実際にデータ分布 p と一致するよう学習したといえるのか。
2. 識別器が生成サンプルと実データを容易に分離できるときでも、その勾配は生成器を p へと導くのか。

固定された生成器に対する最適識別器は閉形式で導出できるため、これら両方の疑問に解析的に答えることができる。その後、厳密な密度と KL ダイバージェンスが閉形式で得られる 2 次元ガウス分布の問題でこの解析を検証し、最後に Fashion-MNIST 画像上で多層パーセプトロン（multilayer perceptron、MLP）の生成器を訓練する。

16.1.1 尤度から 2 サンプル比較へ

16.1.1.1 陰的生成器

最尤推定（maximum likelihood estimation）では、モデルの対数密度（log density）$\log p_\theta(x)$ を各観測値において評価する必要がある。これに対し、本章で扱う生成器は**陰的**（implicit）である。すなわち、潜在ベクトル $z \sim \mathcal{N}(0, I_{d_z})$ をサンプリングし、決定論的なネットワーク $x' = G(z)$ に通す。潜在次元 d_z がデータ次元 d_x より小さいとき、生成サンプルは低次元多様体上に乗り、$\mathbb{R}^{d_x}$ 上のルベーグ測度に関する密度を持たない。たとえ $d_z = d_x$ であっても、変数変換（change of variables）の公式によって誘導密度を計算するには、G が可逆であり、かつ計算可能なヤコビ行列式（Jacobian determinant）を持つ必要がある。

さらに、テストデータに対する対数尤度（log-likelihood）が高くても、写実的なサンプルが得られるとは限らない (Theis et al. 2016)。真のデータ分布と一様ノイズまたはガウスノイズの混合分布を考えてみよう。

$$\tilde{p} \;=\; 0.01\, p_{\text{data}} + 0.99\, p_{\text{noise}}. \tag{16.1}$$

いたるところで $\tilde{p}(x) \geq 0.01\, p_{\text{data}}(x)$ が成り立つため、その期待対数尤度は真のデータ分布より高々 $\log 100 \approx 4.6$ nat しか低くならない。対数尤度が数千 nat に及ぶ画像にとって、この差は無視できるほど小さい。それにもかかわらず、$\tilde{p}$ から抽出されるサンプルの 99% は純粋なノイズである。したがって、サンプル生成器を評価するには、生成されたサンプルを直接調べる必要がある。

16.1.1.2 実サンプルと生成サンプルの識別

2 サンプル検定（two-sample test）は 2 組のサンプルが同じ分布から得られたものかを調べる手法であり、それを実装する自然な方法は、両者を分離する二値分類器を訓練することである。GAN はこの分類器を用いて生成器に勾配を与える。$y \sim \text{Bernoulli}(\frac{1}{2})$ を均衡のとれた出所ラベルとし、$y = 1$ ならばデータ分布から $x \sim p$ を抽出し、$y = 0$ ならば生成器分布から $x \sim q$ を抽出する。このとき x の周辺分布（marginal distribution）は均衡混合分布 $m = (p + q)/2$ となる。

クリティック（または**識別器**）は**真実性ロジット**（realness logit）を出力する関数 $D : \mathcal{X} \to \mathbb{R}$ であり、$\sigma(D(x)) = 1/(1 + e^{-D(x)})$ が事後確率（posterior probability）$P(y = 1 \mid x)$ の推定値となる。Goodfellow et al. (2014) に倣い、コード中では識別器ネットワークに `net_D` と

update_D を用いるが、本文中では本章後半の非確率的なスコアリングネットワークも包含するため**クリティック**という用語も併用する（なお、GAN のクリティックは、15.1 節で扱うアクター・クリティック強化学習における価値関数（value function）のクリティックとは別物である）。

Figure 16.1 に示すように、訓練は 2 つのネットワークの間で交互に進める。識別器は実データのミニバッチと生成データのミニバッチを分離するように更新し、生成器は自身のサンプルが偽物と分類されにくくなるよう、識別器を通じて逆伝播された勾配によって更新する。

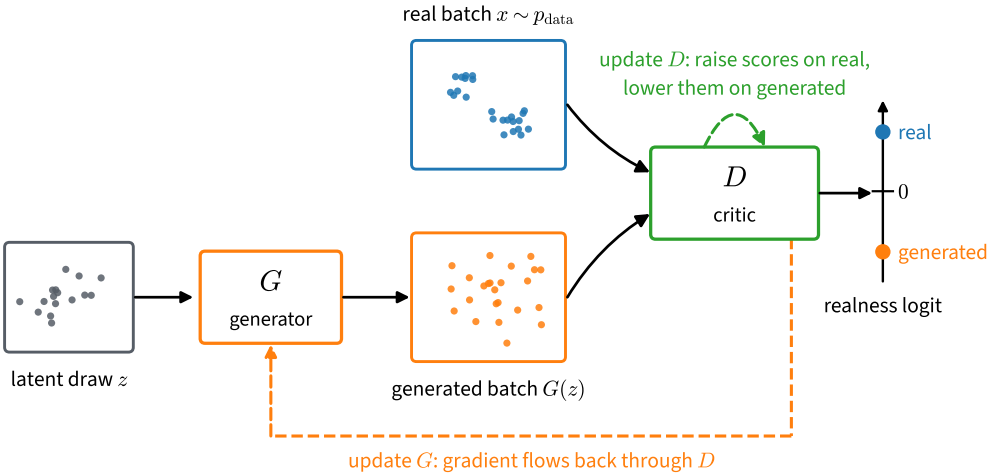


Figure 16.1: 敵対的訓練。生成器 G は潜在変数 z を生成サンプルへ写像し、クリティック D は生成サンプルと実サンプルの両方にスコアを付ける。クリティックは実データに対するスコアを最大化し生成サンプルに対するスコアを最小化する一方、生成器は D を通じた勾配を用いて 2 つの分布を見分けがつかないようにする。

16.1.2 対数損失の目的関数

均衡のとれた実サンプルと生成サンプルに対して二値交差エントロピー（binary cross-entropy、対数損失）で識別器を訓練すると、Goodfellow et al. (2014) のミニマックス価値関数が得られる。

$$V(D) = \mathbb{E}_{x\sim p}\big[\log \sigma(D(x))\big] + \mathbb{E}_{x'\sim q}\big[\log \sigma(-D(x'))\big]. \tag{16.2}$$

識別器は $V(D)$ を最大化する一方、生成器は第 2 項のみに影響し $V(D)$ を最小化しようとするため、ミニマックス問題 $\min_G \max_D V(D)$ が得られる。

16.1.2.1 最適クリティック

固定された生成器分布 q に対して、Equation 16.2 の最大化は各点 x ごとに独立な問題へ分解される。各点 x における被積分関数は $p(x)\log\sigma(D(x)) + q(x)\log\sigma(-D(x))$ である。$u = \sigma(D(x)) \in (0,1)$ とおくと、狭義凹関数 $p\log u + q\log(1-u)$ を一階条件（first-order condition）$p/u = q/(1-u)$ によって最大化することで、次の唯一の最適解が得られる。

$$\sigma(D^\star(x)) = \frac{p(x)}{p(x)+q(x)} = P(y=1 \mid x), \qquad D^\star(x) = \log\frac{p(x)}{q(x)} = \lambda(x). \tag{16.3}$$

したがって、最適識別器はベイズ事後確率 $P(y=1 \mid x)$ を出力し、そのロジット $D^\star(x)$ は**対数密度比**（log density ratio）$\lambda(x) = \log \rho(x)$（ただし $\rho(x) = p(x)/q(x)$）に等しい。$q(x)$ が閉形式の密度を持たなくても、二値分類によってサンプルから直接 $\log(p/q)$ が推定される。さらに、対数損失の目的関数は $D^\star$ の加法定数を一意に定める。すなわち、$D^\star$ を任意の非ゼロ定数 b だけシフトすると V は狭義に減少するため（演習問題 1）、$D^\star(x) > 0$ となるのはまさに $p(x) > q(x)$ の領域に限られる。

たとえば、分散 1 の 2 つのガウス分布 $p = \mathcal{N}(0,1)$ と $q = \mathcal{N}(\mu,1)$ に対して、最適ロジットは x に関するアフィン関数となる。

$$\lambda(x) = -\frac{x^2}{2} + \frac{(x-\mu)^2}{2} = \mu\Big(\frac{\mu}{2} - x\Big),$$

この関数は中点 $\mu/2$ で符号が反転する。p と q が異なる共分散行列を持つ場合、$\lambda(x)$ は x の二次関数となるため、非線形な識別器が必要になる。

16.1.2.2 イェンセン・シャノン・ダイバージェンスとの関係

最適識別器 $\sigma(D^\star) = p/(2m)$ と $\sigma(-D^\star) = q/(2m)$ を Equation 16.2 に代入すると、最適識別器に対して生成器が何を最小化しているかが明らかになる。

$$\begin{aligned} V(D^\star) &= \int p \log\frac{p}{2m} + \int q \log\frac{q}{2m} \\ &= D_{\mathrm{KL}}(p \,\|\, m) + D_{\mathrm{KL}}(q \,\|\, m) - 2\log 2. \end{aligned}$$

イェンセン・シャノン・ダイバージェンス（Jensen–Shannon divergence）の定義 $\mathrm{JS}(p,q) = \tfrac{1}{2}D_{\mathrm{KL}}(p \,\|\, m) + \tfrac{1}{2}D_{\mathrm{KL}}(q \,\|\, m)$ を用いると、次式が得られる。

$$\max_{D} V(D) = 2\,\mathrm{JS}(p,q) - 2\log 2. \tag{16.4}$$

$\mathrm{JS}(p,q) \geq 0$ であり、等号成立は $q = p$ のとき、かつそのときに限るため、$\max_D V(D)$ を最小化するとデータ分布 $q = p$ が大域的最小値（global minimum）$-2\log 2$ において一意に復元される（このときいたるところで $D^\star(x) = 0$ となり、識別器はランダムな当て推量に帰着する）。

$D_{\mathrm{KL}}(p \,\|\, m)$ と $D_{\mathrm{KL}}(q \,\|\, m)$ を微分エントロピー（differential entropy）によって展開すると、情報理論的な解釈が得られる。

$$\mathrm{JS}(p,q) \;=\; H(m) - \frac{H(p)+H(q)}{2} \;=\; I(x;y). \tag{16.5}$$

すなわち、$\mathrm{JS}(p,q)$ はサンプル $x \sim m$ とその二値の出所ラベル y の間の相互情報量（mutual information）$I(x;y)$ に等しい。二値ラベルが持つ情報量は高々 $\log 2$ nat であるため、$0 \leq$

$\mathrm{JS}(p, q) \leq \log 2$ が成り立ち、p と q が互いに素な台（support）を持つときは常に $\log 2$ に達する。

16.1.2.3 互いに素な台と勾配消失

$\mathrm{JS}(p, q)$ は $\log 2$ で飽和するため、p と q の台が重ならないときは常にミニマックス目的関数が定数になる。たとえば、$p = \delta_0$ と $q_\theta = \delta_\theta$ が $\mathbb{R}$ 上の点質量である場合、$\mathrm{JS}(p, q_\theta) = \log 2$ がすべての $\theta \neq 0$ で成り立つため、θ に関する勾配は、θ がどれほど 0 に近くてもゼロになる。裾で台が重なっている場合でさえ、たとえば $p = \mathcal{N}(0, 1)$ と $q = \mathcal{N}(\mu, 1)$ では、$\mathrm{JS}(p, q)$ は 0.11、0.34、0.63 nat（それぞれ $\mu = 1, 2, 4$ のとき）となり、$\mu = 8$ では 10^{-4} 未満の差まで $\log 2 \approx 0.6931$ に近づく。Figure 16.2 に曲線全体を示す。2 つのガウス分布が厳密な意味で重ならなくなるはるか手前で、曲線はすでに平坦になっている。

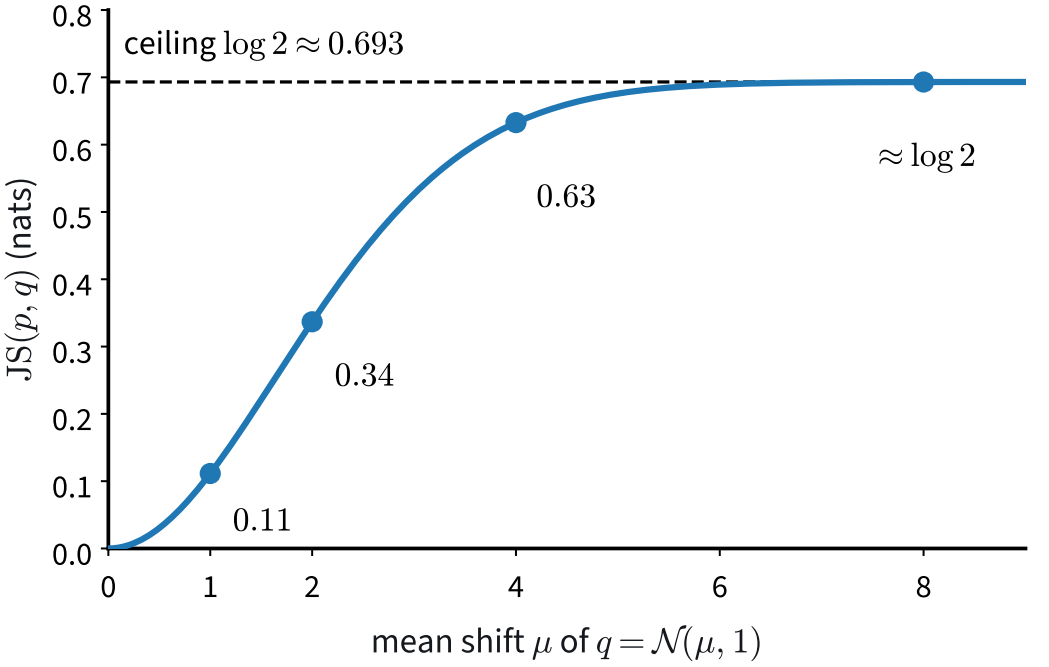


Figure 16.2: $p = \mathcal{N}(0, 1)$ と $q = \mathcal{N}(\mu, 1)$ の間のイェンセン・シャノン・ダイバージェンス（シフト μ の関数）。ダイバージェンスは標準偏差の数倍以内で上限 $\log 2$ に達する。それを超えると曲線は平坦になり、平坦な曲線からは生成器に勾配が与えられない。

高次元空間では、互いに素な台は例外ではなくむしろ常態である。自然画像も、$G : \mathbb{R}^{d_z} \to \mathbb{R}^{d_x}$（$d_z \ll d_x$）なる生成器の出力もともに低次元多様体上に集中し、それらの共通部分は通常、測度 0 となる (Arjovsky and Bottou 2017)。識別器が両者を完全に分離すると、ミニマックス値は飽和する。

最も直接的な解決策は、台を重なり合わせることである。識別器に入力する前に、すべての実サンプルと生成サンプルにスケール s の独立なガウスノイズを加えるとする。すると識別器は、いたるところで正の値を持つ 2 つの平滑化された密度を比較することになる。先ほどの点質量の場合、それらは $\mathcal{N}(0, s^2)$ と $\mathcal{N}(\theta, s^2)$ になり、両者のダイバージェンスは Figure 16.2 の曲線を $\mu = \theta/s$ で読み取った値に一致する。したがって、θ が s の数倍以内にある限り、生成器は勾配を受け取る。これが**インスタンスノイズ**である (Sønderby et al. 2017; Arjovsky and Bottou

2017)。ただし、このままでは生成器が元のデータではなくぼけたデータに一致してしまうため、訓練中に s を 0 に向けてアニーリングする必要がある。このアイデアは 16.7 節で再び登場し、そこでは拡散モデル（diffusion model）によってサンプルにノイズが加えられる。

16.1.3 生成器の勾配

実際の訓練では、生成器は固定された識別器 D に対して勾配ステップを踏むのであり、$\max_D V(D)$ を直接最小化するわけではない。生成器の目的関数の勾配を詳しく見てみよう。

16.1.3.1 飽和型と非飽和型の重み

文字どおりのミニマックス定式化では、生成器は $\mathbb{E}_z[\log \sigma(-D(G(z)))]$ を自身のパラメータ θ に関して最小化する。$\frac{d}{du}\log\sigma(-u) = -\sigma(u)$ と $x' = G(z)$ を用いると、その勾配は次のようになる。

$$\begin{aligned}&\nabla_\theta \, \mathbb{E}_z\big[\log \sigma(-D(G(z)))\big] \\ &\quad = -\,\mathbb{E}_z\big[\sigma(D(x'))\,\nabla_\theta D(G(z))\big].\end{aligned}$$

各生成サンプル x' は $\nabla_\theta D(G(z))$ に沿って上り方向へ押し上げられ、その重みは $\sigma(D(x'))$（識別器が推定する、x' が実データである確率）で与えられる。識別器が高い確信をもってサンプルを偽物と棄却するとき（$D(x') \ll 0$）、この重み $\sigma(D(x')) \approx e^{D(x')}$ は指数関数的に小さくなり、生成器が最も改善を必要としているまさにその局面で生成器の勾配が**飽和**（saturate）してしまう。

この飽和を避けるため、Goodfellow et al. (2014) は生成サンプルが実データと分類される対数確率（log probability）を最大化するよう、すなわち $-\mathbb{E}_z[\log \sigma(D(G(z)))]$ を最小化するよう生成器を訓練した。$\frac{d}{du}\log\sigma(u) = \sigma(-u)$ より、その勾配は次のようになる。

$$\begin{aligned}&\nabla_\theta\Big(-\mathbb{E}_z\big[\log \sigma(D(G(z)))\big]\Big) \\ &\quad = -\,\mathbb{E}_z\big[\sigma(-D(x'))\,\nabla_\theta D(G(z))\big].\end{aligned}$$

どちらの目的関数も各サンプルを同じ方向 $\nabla_\theta D(G(z))$ へ押し上げるが、サンプルごとのスカラー重みだけが異なる。

$$\begin{aligned}w_{\text{sat}}(x') &= \sigma(D(x')), \\ w_{\text{ns}}(x') &= \sigma(-D(x')) = 1 - \sigma(D(x')).\end{aligned} \tag{16.6}$$

Figure 16.3 に示すように、確信をもって棄却されるサンプル（$D(x') \ll 0$）に対して**非飽和型**（non-saturating）の重み $w_{\text{ns}}(x')$ は 1 に近づくのに対し、$w_{\text{sat}}(x')$ は $e^{D(x')}$ として減衰する（比は $w_{\text{ns}}/w_{\text{sat}} = e^{-D(x')}$ であり、$D = -3$ で $20\times$、$D = -5$ で $148\times$ となる）。

16.1.3.2 非飽和損失の実効ダイバージェンス

識別器が最適応答 $D = \lambda = \log(p/q)$ に保たれているとき、非飽和生成器損失はどのダイバージェンスを下るのだろうか。$\log\sigma(-D) - \log\sigma(D) = -D$ を用いると、非飽和型の目的関数は次のように分解できる。

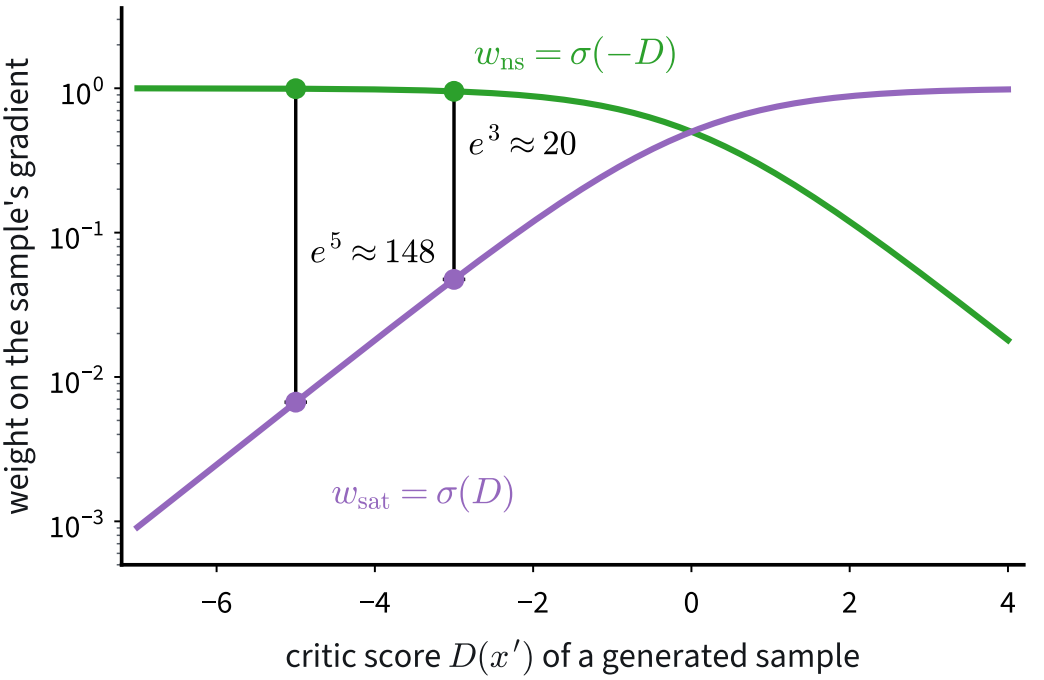


Figure 16.3: 識別器ロジット $D(x')$ の関数として対数スケールでプロットした、サンプルごとの生成器勾配の重み。確信をもって棄却されるサンプル（$D(x') \ll 0$）では、飽和型の重み $\sigma(D)$ が指数関数的に減衰するのに対し、非飽和型の重み $\sigma(-D)$ は 1 付近にとどまる。

$$
\begin{aligned}
&- \mathbb{E}_{x'}\big[\log \sigma(D(x'))\big] \\
&\quad = \mathbb{E}_{x'}\big[-D(x')\big] - \mathbb{E}_{x'}\big[\log \sigma(-D(x'))\big].
\end{aligned}
$$

D を λ に固定すると、右辺の第 1 項は q に関して逆方向 KL ダイバージェンス $D_{\mathrm{KL}}(q \,\|\, p) = \mathbb{E}_q[\log(q/p)]$ と同じ勾配を持ち、第 2 項はミニマックス損失にマイナスを付けたものであり、包絡線定理（envelope theorem）によりその勾配は $2\,\mathrm{JS}(p,q) = D_{\mathrm{KL}}(p \,\|\, m) + D_{\mathrm{KL}}(q \,\|\, m)$ の勾配と一致する (Arjovsky and Bottou 2017)（演習問題 5）。$\log(q/p) - \log(q/m) = \log(m/p)$ と $p + q = 2m$ を用いて積分をまとめると、次式が得られる。

$$
\begin{aligned}
&D_{\mathrm{KL}}(q \,\|\, p) - D_{\mathrm{KL}}(p \,\|\, m) - D_{\mathrm{KL}}(q \,\|\, m) \\
&\quad = \int q \log \frac{m}{p} + \int p \log \frac{m}{p} \\
&\quad = \int (p+q) \log \frac{m}{p} \;=\; 2\, D_{\mathrm{KL}}(m \,\|\, p).
\end{aligned}
\tag{16.7}
$$

したがって、最適クリティックに対して非飽和型の生成器は $2\,D_{\mathrm{KL}}(m \,\|\, p)$、すなわち**軟化された逆方向 KL ダイバージェンス**（softened reverse KL divergence）を下ることになり (Shannon et al. 2020)、これも唯一の大域的最小値 $q = p$ を共有する。

$2\,D_{\mathrm{KL}}(m \,\|\, p)$ を詳しく見ると、誤差へのペナルティの課し方に非対称性があることがわかる。

- $q(x) > 0$ かつ $p(x) = 0$ となる領域（データの台の外側にある偽サンプル）では、$m(x) > 0 = p(x)$ より $D_{\mathrm{KL}}(m \,\|\, p) = \infty$ となり、非現実的なサンプルに厳しいペナルティが課される。
- $p(x) > 0$ かつ $q(x) = 0$ となる領域（データの欠落したモード）では、$m(x) = p(x)/2$ より被積分関数の寄与は $p(x) \log \frac{1}{2}$ という小さな有限のペナルティにとどまる。

16.1.3.3 モード欠落

生成器が多峰性分布 p の一部のモードしかカバーしない現象を、**モード欠落**（mode dropping）、または**モード崩壊**（mode collapse）と呼ぶ。いま、p が互いに素な 2 つのモードの等重み混合分布であり、q が一方のモードには完全に一致し、もう一方を完全に落としているとする。カバーされたモード上では $p = q/2$ であるため、$D_{\mathrm{KL}}(q \,\|\, p) = \log 2 \approx 0.69$ nat、$\mathrm{JS}(p,q) \approx 0.22$ nat、$2\,D_{\mathrm{KL}}(m \,\|\, p) \approx 0.26$ nat となる。データ分布の半分を落としても、わずか 1/4 nat のコストしか生じないのである。これに対し、最尤法（maximum likelihood）は順方向 KL ダイバージェンス $D_{\mathrm{KL}}(p \,\|\, q)$ を最小化するため、p のいかなる部分にせよ密度 0 が割り当てられれば無限大となる。

さらに悪いことに、カバーされたモード上での識別器の最適応答 $D^\star(x) = \log(p(x)/q(x)) = -\log 2$ においては、$D^\star$ が q の台全体にわたって定数となるため、$\nabla_\theta D^\star(G(z)) = 0$ となる。すなわち、カバーされたモード上の局所勾配は消失し、欠落したモードへ向かう信号をまったく与えない。仮に不完全な識別器が、カバーされたモード上の生成サンプルに $D = 3$、カバーされていないモード上の実サンプルに $D = 5$ を割り当てたとしても、各生成サンプルは固定されたゼロのしきい値に対してのみ $\sigma(-3) \approx 0.05$ で重み付けされ、カバーされていない実モード上のより高いスコアが生成器の更新に入り込むことは決してない。16.3 節では、生成サンプルのスコアを実サンプルのスコアと直接比較することでこの問題に対処できるかを検証する。

16.1.4 2 次元ガウス分布での理論の検証

理論的予測を定量的に検証するため、まず p、q、$\lambda(x) = \log(p(x)/q(x))$、および $D_{\mathrm{KL}}(q \,\|\, p)$ がすべて閉形式で計算できる 2 次元のガウス分布データ上で線形生成器を訓練する。

16.1.4.1 交互更新則

本章全体で再利用するため、`update_D`、`update_G`、および交互訓練ループ `train_gan` をここで一度定義し、`d2l` に保存しておく。まず `update_D` は、Equation 16.2 を最大化する識別器の更新を 1 ステップ実行する（実サンプルにはラベル 1、切り離した生成サンプルにはラベル 0 を与えて二値交差エントロピーを最小化する）。合計された損失をバッチサイズで割ると $-V(D)$ の推定値となり、その値は $2\log 2 \approx 1.386$（$D = 0$ のとき）に等しい。

```
@nnx.jit
def update_D(X, Z, net_D, net_G, optimizer_D):
    """Update the critic; return its log loss summed over both batches
    (per sample, the sum of two means: 2 log 2 at equilibrium)."""
    batch_size = X.shape[0]
    fake_X = net_G(Z)  # computed outside the loss: no gradient to net_G
    def loss_D_fn(model_D):
        real_Y = model_D(X).squeeze()
        fake_Y = model_D(fake_X).squeeze()
        return (jnp.sum(optax.sigmoid_binary_cross_entropy(
                    real_Y, jnp.ones(batch_size))) +
                jnp.sum(optax.sigmoid_binary_cross_entropy(
                    fake_Y, jnp.zeros(batch_size))))
    loss_D, grads_D = nnx.value_and_grad(loss_D_fn)(net_D)
    optimizer_D.update(net_D, grads_D)
    return loss_D
```

次に `update_G` は、非飽和損失 $-\mathbb{E}[\log \sigma(D(G(z)))]$ を最小化する生成器の更新を 1 ステップ実行する。これは再計算した識別器スコア `net_D(net_G(Z))` に対する正解ラベル 1 の二値交差エントロピーとして実装され、サンプルあたりの値は $\log 2 \approx 0.693$（$D = 0$ のとき）に等しい。

```
@nnx.jit
def update_G(Z, net_D, net_G, optimizer_G):
    """Update the generator on the non-saturating loss; return its sum."""
    # Both networks are arguments, so critics with mutable state (batch
    # norm) work too; argnums=0 differentiates the generator only
    def loss_G_fn(model_G, model_D):
        fake_Y = model_D(model_G(Z)).squeeze()
        return jnp.sum(optax.sigmoid_binary_cross_entropy(
            fake_Y, jnp.ones(Z.shape[0])))
    loss_G, grads_G = nnx.value_and_grad(loss_G_fn, argnums=0)(net_G, net_D)
    optimizer_G.update(net_G, grads_G)
    return loss_G
```

`train_gan` は、変化する目的関数にモメンタムが素早く追従できるよう $\beta_1 = 0.5$ とした Adam を用い (Radford et al. 2015)、ミニバッチごとに `update_D` と `step_G` を交互に実行する。また、オプションとしてエポックごとのコールバック `on_epoch` と指数移動平均（exponential moving average）`ema` も受け付ける。

```
def train_gan(net_D, net_G, data_iter, num_epochs, latent_dim, lr_D, lr_G,
              step_G=update_G, seed=None, on_epoch=None, ema=None):
    """Alternate update_D and step_G over the minibatches with Adam
    (beta_1 = 0.5), calling ema.update(net_G) after every generator step
    and on_epoch(epoch) after each epoch if given; return each epoch's
    per-sample losses (loss_D, the sum of the critic's two batch means,
    and loss_G)."""
    key = d2l.get_key() if seed is None else jax.random.key(seed)
    optimizer_D = nnx.Optimizer(net_D, optax.adam(lr_D, b1=0.5),
                                wrt=nnx.Param)
    optimizer_G = nnx.Optimizer(net_G, optax.adam(lr_G, b1=0.5),
                                wrt=nnx.Param)
    history = []
    for epoch in range(1, num_epochs + 1):
        loss_D, loss_G, n = 0.0, 0.0, 0  # summed on the device, read once
        for X, *_ in data_iter:  # labels, if any, are ignored
            key, subkey = jax.random.split(key)
            Z = jax.random.normal(subkey, (X.shape[0], latent_dim))
            loss_D += update_D(X, Z, net_D, net_G, optimizer_D)
            loss_G += step_G(Z, net_D, net_G, optimizer_G)
            if ema is not None:
                ema.update(net_G)
            n += X.shape[0]
        history.append((loss_D / n, loss_G / n))
        if on_epoch is not None:
            on_epoch(epoch)
    return np.array(history)
```

16.1.4.2 既知の密度を持つ線形生成器

1,000 個の訓練点を $x = zA + b$（ただし $z \sim \mathcal{N}(0, I_2)$）から生成すると、真のデータ分布は傾いた楕円形のガウス分布 $p = \mathcal{N}(b, A^\top A)$ となる。

```
Z = jax.random.normal(jax.random.key(0), (1000, 2))
A = jnp.array([[1.0, 2.0], [-0.1, 0.5]])
b = jnp.array([1.0, 2.0])
data = Z @ A + b
print(f'covariance of the data distribution:\n{A.T @ A}')
```

```
covariance of the data distribution:
[[1.01 1.95]
 [1.95 4.25]]
```

生成器は単一の線形層 $G(z) = zA_q + \mu_q$ であり、ガウス分布 $q = \mathcal{N}(\mu_q, A_q^\top A_q)$ を誘導する。異なる共分散を持つ 2 つのガウス分布間の対数密度比は二次関数になるため、識別器には 16 個の Leaky ReLU ユニット（負の傾き 0.2）からなる 2 つの隠れ層を持つ 3 層の MLP を用いる（代わりに飽和する tanh 活性化関数を用いた場合に何が起きるかは、演習問題 3 で調べる）。

```
def make_nets(seed=7):
    """The linear generator and the critic, all weights and offsets drawn
    from N(0, 0.02^2) with keys from seed."""
```

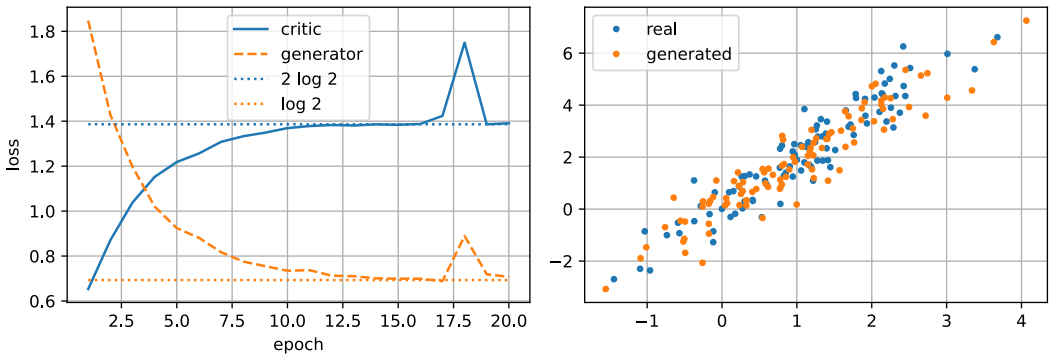
```
    normal = nnx.initializers.normal(0.02)
    dense = lambda m, n, rngs: nnx.Linear(m, n, kernel_init=normal,
                                          bias_init=normal, rngs=rngs)
    leaky = lambda x: nnx.leaky_relu(x, 0.2)
    rngs_G, rngs_D = nnx.Rngs(seed), nnx.Rngs(seed + 1)
    net_G = nnx.Sequential(dense(2, 2, rngs_G))
    net_D = nnx.Sequential(dense(2, 16, rngs_D), leaky,
                           dense(16, 16, rngs_D), leaky,
                           dense(16, 1, rngs_D))
    return net_G, net_D
```

生成器の移動に伴って識別器が最適応答に追従できるよう `lr_D=0.05`、`lr_G=0.005` と設定し、バッチサイズ 8 で 20 エポック訓練する。

```
data_iter = d2l.ArrayLoader((data,), batch_size=8, shuffle=True,
                            key=jax.random.key(6))
net_G, net_D = make_nets()
history = train_gan(net_D, net_G, data_iter, num_epochs=20, latent_dim=2,
                    lr_D=0.05, lr_G=0.005, seed=1)
fake = net_G(jax.random.normal(jax.random.key(4), (200, 2)))
```

ここで、`history` に記録されたエポックごとのクリティック損失と生成器損失をそれぞれの理論的均衡値（$2\log 2$ と $\log 2$）に対してプロットし、あわせて実サンプル（`data`）と生成サンプル（`fake`）の散布図を描く。

```
fig, axes = d2l.plt.subplots(1, 2, figsize=(9, 3.2))
epochs = list(range(1, len(history) + 1))
d2l.plot(epochs, [history[:, 0], history[:, 1], [2 * math.log(2)] * len(epochs),
                  [math.log(2)] * len(epochs)],
         'epoch', 'loss',
         legend=['critic', 'generator', '2 log 2', 'log 2'],
         fmts=('C0-', 'C1--', 'C0:', 'C1:'), axes=axes[0])
d2l.plot([data[:100, 0], fake[:100, 0]], [data[:100, 1], fake[:100, 1]],
         legend=['real', 'generated'], fmts=('C0.', 'C1.'), axes=axes[1])
fig.tight_layout()
```



10 エポック以内に、識別器損失は均衡値の $2\log 2$ まで上昇し、生成器損失は $\log 2$ に落ち着き、生成された点群は目標の楕円と一致する。

16.1.4.3 当てはまりの nat 単位での定量化

$G(z) = zA_q + \mu_q$ はガウス分布 $q = \mathcal{N}(\mu_q, \Sigma_q)$（ただし $\Sigma_q = A_q^\top A_q$）を定めるため、厳密な KL ダイバージェンス $D_{\mathrm{KL}}(q \,\|\, p)$ を d_x 次元で評価できる (Kullback 1959)。

$$\begin{aligned} &D_{\mathrm{KL}}(\mathcal{N}(\mu_0, \Sigma_0) \,\|\, \mathcal{N}(\mu_1, \Sigma_1)) \\ &\quad = \tfrac{1}{2}\Big[\operatorname{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^\top \Sigma_1^{-1}(\mu_1 - \mu_0) \\ &\qquad - d_x + \log \tfrac{\det \Sigma_1}{\det \Sigma_0}\Big]. \end{aligned} \tag{16.8}$$

```
def generator_gaussian(net_G):
    A_q, b_q = net_G.layers[0].kernel[...], net_G.layers[0].bias[...]
    return b_q, A_q.T @ A_q

def kl_gaussians(mu0, S0, mu1, S1):
    diff = mu1 - mu0
    return 0.5 * (jnp.trace(jnp.linalg.solve(S1, S0))
                  + diff @ jnp.linalg.solve(S1, diff) - len(mu0)
                  + jnp.linalg.slogdet(S1)[1] - jnp.linalg.slogdet(S0)[1])

mu_p, Sigma_p = b, A.T @ A
for name, net in [('at initialization', make_nets()[0]),
                  ('after training   ', net_G)]:
    mu_q, Sigma_q = generator_gaussian(net)
    print(f'KL(q || p) {name}: '
          f'{kl_gaussians(mu_q, Sigma_q, mu_p, Sigma_p):.3f} nats')
```

```
KL(q || p) at initialization: 7.856 nats
KL(q || p) after training   : 0.059 nats
```

q の密度を一度も評価することなく、敵対的訓練（adversarial training）によって $D_{\mathrm{KL}}(q \,\|\, p)$ は初期化時の ~ 8 nat（初期重み $\mathcal{N}(0, 0.02^2)$ の低いエントロピーが支配的）から 100 分の数 nat まで減少する。

16.1.4.4 学習された対数密度比の検証

訓練された識別器が $\lambda(x) = \log(p(x)/q(x))$（Equation 16.3）を正確に復元できるかを検証するため、生成器 q を $A_q = [[1, 2], [-0.5, 0.5]]$、$\mu_q = (1.5, 1.5)$（$D_{\mathrm{KL}}(q \,\|\, p) \approx 3.4$ nat）に固定し、この凍結した q に対して新たな識別器を 2,000 ステップ訓練する。

```
net_G_q, net_D_q = make_nets()  # a fresh critic
net_G_q.layers[0].kernel[...] = jnp.array([[1.0, 2.0], [-0.5, 0.5]])  # A_q
net_G_q.layers[0].bias[...] = jnp.array([1.5, 1.5])  # b_q
optimizer_D = nnx.Optimizer(net_D_q, optax.adam(0.01), wrt=nnx.Param)
key = jax.random.key(2)
for _ in range(2000):  # critic-only steps against the frozen generator
    key, k1, k2 = jax.random.split(key, 3)
    X = data[jax.random.randint(k1, (256,), 0, len(data))]
    Zb = jax.random.normal(k2, (256, 2))
    update_D(X, Zb, net_D_q, net_G_q, optimizer_D)
```

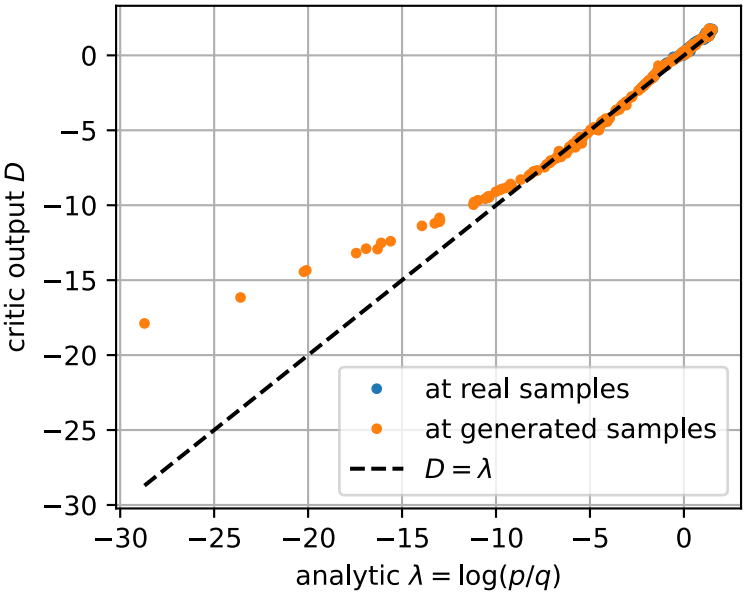
200 個の実データ点と 200 個の生成点にわたって識別器ロジット $D(x)$ と厳密な対数密度比 $\lambda(x)$ の両方を評価し、両者の相関と説明分散（explained variance）R^2 を求める。

```
def gaussian_logpdf(x, mu, Sigma):
    diff = x - mu
    sol = jnp.linalg.solve(Sigma, diff.T).T
    return -0.5 * ((diff * sol).sum(1)
                   + jnp.linalg.slogdet(2 * jnp.pi * Sigma)[1])

mu_q, Sigma_q = generator_gaussian(net_G_q)
key, subkey = jax.random.split(key)
xs = jnp.concatenate([data[:200],
                      net_G_q(jax.random.normal(subkey, (200, 2)))])
lam = (gaussian_logpdf(xs, b, A.T @ A)
       - gaussian_logpdf(xs, mu_q, Sigma_q))
D_out = net_D_q(xs).squeeze()
r2 = 1 - ((D_out - lam) ** 2).mean() / ((lam - lam.mean()) ** 2).mean()
print(f'correlation {jnp.corrcoef(jnp.stack([lam, D_out]))[0, 1]:.3f}, '
      f'R^2 {r2:.3f}')
```

```
correlation 0.986, R^2 0.953
```

クリティックのロジット `D_out` を厳密な対数密度比 `lam` に対して 200 個の実サンプルと 200 個の生成サンプルにわたりプロットすると、$D(x)$ が恒等直線 $D = \lambda$ にどれほど密接に追従しているかが視覚的にわかる。

```
lims = [float(lam.min()), float(lam.max())]
d2l.plot([lam[:200], lam[200:], lims], [D_out[:200], D_out[200:], lims],
         r'analytic $\lambda = \log(p/q)$', 'critic output $D$',
         legend=['at real samples', 'at generated samples', r'$D = \lambda$'],
         fmts=('C0.', 'C1.', 'k--'), figsize=(4.2, 3.4))
```



学習されたロジットは真の対数密度比によく追従する（correlation ≈ 0.98、$R^2 > 0.90$）。ずれが生じるのは λ が極端に負の値をとる領域（$p(x) \approx 0$ となる生成サンプル）のみである。これは、区分線形な Leaky ReLU ネットワークが裾に向かって二次関数的ではなく線形に外挿することに加え、$\frac{d}{dD}[-\log\sigma(-D)] = \sigma(D) \ll 1$（$D \ll 0$ のとき）となるため、識別器がいったん十分な確信を持つと、対数損失が識別器をさらに急峻にする勾配をほとんど生み出さなくなるからである。

16.1.4.5 飽和型と非飽和型の生成器損失

生成器がデータから遠く離れた位置から出発するときの飽和型と非飽和型の重み（Equation 16.6）を比較するため、生成器の初期バイアスを $(-6, 6)$ だけシフトさせ（データの平均から ~ 8 単位離れ、$D_{\mathrm{KL}}(q \,\|\, p) \approx 345$ nat）、`update_G` と、$\mathbb{E}_z[\log\sigma(-D(G(z)))]$ を直接最小化する `update_G_saturating` を比較する。

```
@nnx.jit
def update_G_saturating(Z, net_D, net_G, optimizer_G):
    def loss_G_fn(model_G, model_D):
        fake_Y = model_D(model_G(Z)).squeeze()
        # log sigma(-D) is minus the cross-entropy under the label zero
        return -jnp.sum(optax.sigmoid_binary_cross_entropy(
            fake_Y, jnp.zeros(Z.shape[0])))
    loss_G, grads_G = nnx.value_and_grad(loss_G_fn, argnums=0)(net_G, net_D)
    optimizer_G.update(net_G, grads_G)
    return loss_G
```

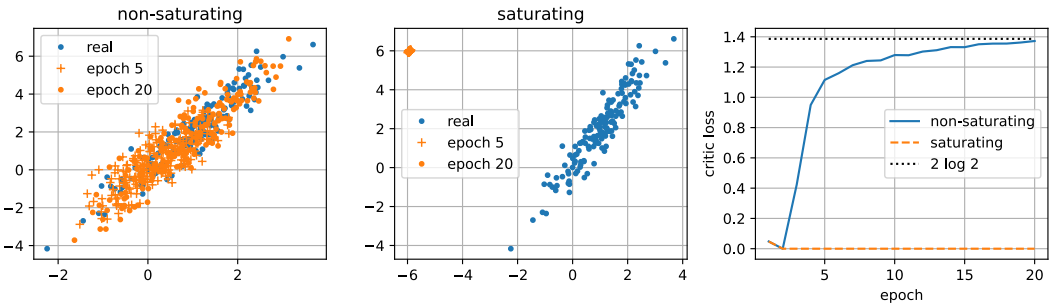
各生成器の目的関数について、同一の初期化とミニバッチ（`lr_G=0.01`、`lr_D=0.05`）のもとで 3 つのシードを実行する。

```
def far_start():
    """make_nets with the generator's offset moved by (-6, 6)."""
    net_G, net_D = make_nets()
    net_G.layers[0].bias[...] += jnp.array([-6.0, 6.0])
    return net_G, net_D

mu_q, Sigma_q = generator_gaussian(far_start()[0])
print(f'start of every run: KL(q || p) = '
      f'{kl_gaussians(mu_q, Sigma_q, mu_p, Sigma_p):.2f} nats')
Z_show = jax.random.normal(jax.random.key(5), (200, 2))  # for every snapshot
runs = {}  # name: (losses, snapshots) of the first run, for the plot
for name, step in [('non-saturating', update_G),
                   ('saturating', update_G_saturating)]:
    kls = []
    for seed in (1, 2, 3):  # both arms see the same minibatches and draws
        net_G_ab, net_D_ab = far_start()
        data_iter_ab = d2l.ArrayLoader((data,), batch_size=8, shuffle=True,
                                       key=jax.random.key(seed - 1))
        snaps = {}  # epoch: generated samples
        hist = train_gan(net_D_ab, net_G_ab, data_iter_ab, num_epochs=20,
                         latent_dim=2, lr_D=0.05, lr_G=0.01, step_G=step,
                         seed=seed, on_epoch=lambda epoch: snaps.update(
                             {epoch: net_G_ab(Z_show)}))
        runs.setdefault(name, (hist, snaps))
        mu_q, Sigma_q = generator_gaussian(net_G_ab)
        kls.append(f'{kl_gaussians(mu_q, Sigma_q, mu_p, Sigma_p):.2f}')
    print(f'{name}: final KL(q || p) = {", ".join(kls)} nats')
```

```
start of every run: KL(q || p) = 347.77 nats
non-saturating: final KL(q || p) = 0.05, 0.04, 0.03 nats
saturating: final KL(q || p) = 340.51, 341.12, 340.84 nats
```

最終的な KL ダイバージェンスがこれほど大きく異なる理由を確認するため、各目的関数における 5 エポック目と 20 エポック目の生成点群 snaps を、クリティック損失の推移と並べてプロットする。

```
fig, axes = d2l.plt.subplots(1, 3, figsize=(10.5, 3.2))
for ax, (name, (hist, snaps)) in zip(axes[:2], runs.items()):
    d2l.plot([data[:150, 0], snaps[5][:, 0], snaps[20][:, 0]],
             [data[:150, 1], snaps[5][:, 1], snaps[20][:, 1]],
             legend=['real', 'epoch 5', 'epoch 20'],
             fmts=('C0.', 'C1+', 'C1.'), axes=ax)
    ax.set_title(name)
epochs = list(range(1, 21))
d2l.plot(epochs, [runs['non-saturating'][0][:, 0],
                  runs['saturating'][0][:, 0], [2 * math.log(2)] * 20],
         'epoch', 'critic loss', legend=['non-saturating', 'saturating',
                                         '2 log 2'],
         fmts=('-', '--', 'k:'), axes=axes[2])
fig.tight_layout()
```



最初の 2 エポックで、どちらの識別器も遠く離れた生成点群をデータから容易に分離し、識別器損失は 0 付近まで下がり $D(x') \ll 0$ となる。$w_{\text{sat}}(x') = \sigma(D(x')) \approx 0$ であるため、飽和型の生成器は完全に停滞する（20 エポック後もデータから依然として 335–340 nat 離れている）。これに対し、非飽和型の重み $w_{\text{ns}}(x') = \sigma(-D(x')) \approx 1$ は識別器の空間勾配 $\nabla_\theta D(G(z))$ を維持し、20 エポック以内に生成器をデータ分布上へと引き寄せる（$D_{\text{KL}}(q \,\|\, p)$ は 100 分の数 nat）。なお、w_{ns} は $\nabla_\theta D$ を再スケーリングするだけであり、Leaky ReLU 識別器が x' において非ゼロの空間的な傾きを持つことに依存している。

16.1.5 Fashion-MNIST での MLP GAN の訓練

画像データでは $p(x)$ も $q(x)$ も閉形式ではわからないため、サンプル品質と訓練損失を監視するほかない。最初の画像実験として、Fashion-MNIST（3.2 節）の 60,000 枚の 28×28 グレースケール画像を $[-1, 1]$ にスケーリングし、2 つの Leaky ReLU MLP を訓練する。生成器は 100 次元の潜在ベクトルを 256 ユニットと 512 ユニットの隠れ層に通し、tanh 出力活性化関数によって 784 ピクセルへ写像する。識別器は 784 ピクセルを 512 ユニットと 256 ユニットの層に通して単一のロジットへ写像する。後の節でも単純な MLP を構築できるよう、補助関数 `make_mlp` を保存しておく。

```
def make_mlp(sizes, act, *, rngs):
    """Linear layers of the given widths with the activation function act
    between them; weights uniform in +-1/sqrt(fan_in), zero biases."""
    init = nnx.initializers.variance_scaling(1 / 3, 'fan_in', 'uniform')
    layers = []
    for d_in, d_out in zip(sizes[:-1], sizes[1:]):
        layers += [nnx.Linear(d_in, d_out, kernel_init=init, rngs=rngs), act]
    return nnx.Sequential(*layers[:-1])
```

`make_mlp` を用い、60,000 枚の Fashion-MNIST 訓練画像を 784 次元のベクトル（値域 $[-1, 1]$）に再スケーリングし、ミニバッチローダ `image_iter` を構築したうえで、生成器 `net_G_img`（末尾は `tanh`）と識別器 `net_D_img` をインスタンス化する。

```
fmnist = d2l.FashionMNIST()
# All 60,000 training images as rows of 784 pixels in [-1, 1]
images = fmnist.train[0].reshape(-1, 784).astype(np.float32) / 127.5 - 1
image_iter = d2l.ArrayLoader((images,), batch_size=64, shuffle=True,
                             key=jax.random.key(3))
rngs = nnx.Rngs(0)
leaky = lambda x: nnx.leaky_relu(x, 0.2)
net_G_img = nnx.Sequential(make_mlp([100, 256, 512, 784], leaky, rngs=rngs),
```

```
                            jnp.tanh)
net_D_img = make_mlp([784, 512, 256, 1], leaky, rngs=rngs)
```

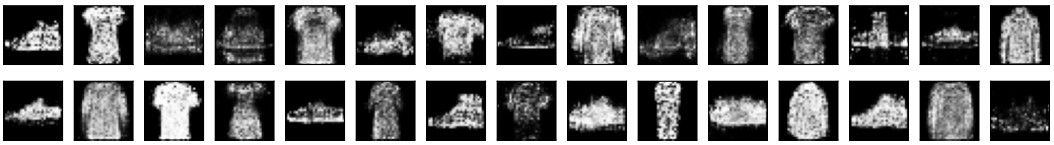
両ネットワークを学習率 2×10^{-4}、バッチサイズ 64 で 20 エポック（$\sim$ 18,750 ステップ）訓練する。

```
history_img = train_gan(net_D_img, net_G_img, image_iter, num_epochs=20,
                        latent_dim=100, lr_D=2e-4, lr_G=2e-4, seed=1)
print(f'last epoch: critic loss {history_img[-1, 0]:.2f}, '
      f'generator loss {history_img[-1, 1]:.2f}')
```

```
last epoch: critic loss 1.04, generator loss 1.37
```

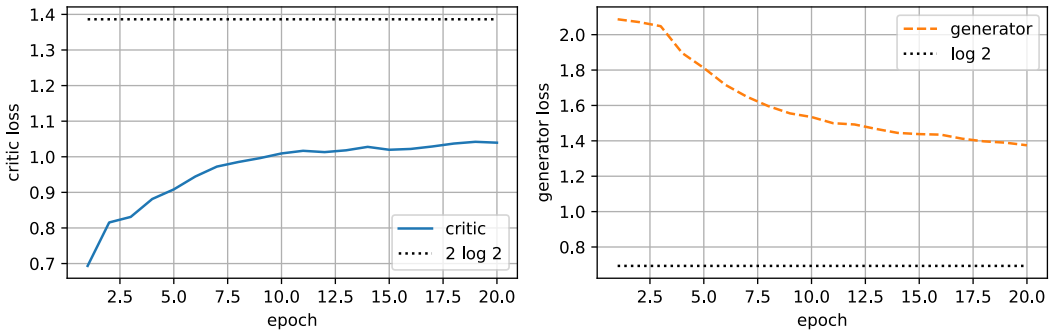
訓練後、30 個の潜在ベクトルを $\mathcal{N}(0, I_{100})$ から抽出し、`net_G_img` に通して出力を 28×28 の画像に変形し、生成された衣類を確認する。

```
fake_img = net_G_img(jax.random.normal(jax.random.key(2), (30, 100)))
d2l.show_images(fake_img.reshape(30, 28, 28), 2, 15, scale=0.9, cmap='gray');
```



また、`history_img` に記録されたエポックごとのクリティック損失と生成器損失を、基準水準である $2\log 2$ と $\log 2$ に対してプロットする。

```
epochs = list(range(1, len(history_img) + 1))
fig, axes = d2l.plt.subplots(1, 2, figsize=(9, 3), sharex=True)
for ax, loss, ref, name, ref_name, fmt in zip(
        axes, history_img.T, (2 * math.log(2), math.log(2)),
        ('critic', 'generator'), ('2 log 2', 'log 2'), ('C0-', 'C1--')):
    d2l.plot(epochs, [loss, [ref] * len(epochs)], 'epoch', f'{name} loss',
             legend=[name, ref_name], fmts=(fmt, 'k:'), axes=ax)
fig.tight_layout()
```



生成された画像は、MLP には畳み込みのような空間的帰納バイアス（inductive bias）がない

ためざらついたピクセルノイズを伴うものの、識別可能な衣類のシルエット（シャツ、ズボン、靴、バッグ）を捉えている。なお、20 エポック経過した後も、識別器損失（~ 1.05）は均衡値 $2\log 2 \approx 1.39$ を下回り続け、生成器損失（~ 1.4）は $\log 2 \approx 0.69$ を上回り続けている。2 次元ガウス分布の設定とは異なり、画像における損失曲線だけでは、不完全な生成と識別器の過学習（overfitting）を区別することも（演習問題 6）、すべての衣類カテゴリが均等にカバーされているかを明らかにすることもできない（演習問題 4）。

16.1.6 まとめ

敵対的生成ネットワークは、陰的生成器 $x' = G(z)$ に識別器（またはクリティック）D を組み合わせ、実データ $x \sim p$ と生成サンプル $x' \sim q$ を分離させることで、明示的な尤度を用いずに生成器を訓練する。二値対数損失のもとでは、最適識別器のロジットは対数密度比 $D^\star(x) = \lambda(x) = \log(p(x)/q(x))$ を復元し、最適価値関数は $\max_D V(D) = 2\,\mathrm{JS}(p,q) - 2\log 2$ に帰着する。ここで $\mathrm{JS}(p,q) = I(x;y) \in [0, \log 2]$ は、$m = (p+q)/2$ からのサンプルとその出所ラベルの間の相互情報量である。生成サンプルが確信をもって棄却されるとき（$D(x') \ll 0$）、文字どおりのミニマックス生成器の重み $w_{\mathrm{sat}}(x') = \sigma(D(x'))$ は指数関数的に消失するのに対し、非飽和生成器損失 $-\mathbb{E}[\log \sigma(D(G(z)))]$ は 1 に近い重み $w_{\mathrm{ns}}(x') = \sigma(-D(x')) \approx 1$ を維持する。最適識別器に対して非飽和損失は $2\,D_{\mathrm{KL}}(m \,\|\, p)$ を下り、データの台から外れたサンプルには厳しいペナルティを課す一方、欠落したモードには有界なペナルティしか課さない。

16.1.7 演習問題

1. **過剰損失と加法定数。** $\eta = p/(p+q)$ とし、$D_{\mathrm{b}}(a \,\|\, u)$ をパラメータ a と u を持つベルヌーイ分布間の KL ダイバージェンスとする。
   1. すべてのクリティックが $V(D^\star) - V(D) = 2\,\mathbb{E}_m\big[D_{\mathrm{b}}(\eta \,\|\, \sigma(D))\big]$ を満たし、したがって $D^\star$ が Equation 16.2 の m 零集合を除いて唯一の最大点であることを示せ。
   2. これより、$V(D^\star + b) < V(D^\star)$ が任意の定数 $b \neq 0$ に対して成り立つことを導け。
   3. b について 2 次まで展開し、
   $$V(D^\star + b) - V(D^\star) = -\frac{b^2}{2}\int \frac{pq}{p+q}\,dx + O(b^3),$$
   であり、その係数が $[0, \tfrac{1}{2}]$ に収まることを示せ。p と q がほぼ互いに素であるとき、有限個のサンプルは D の加法定数をどの程度強く拘束するか。
2. **測度 0 の台。**
   1. 生成器 $G(z) = zv + \mu_q$（1 次元の潜在変数 $z \in \mathbb{R}$）は、そのサンプルを $\mathbb{R}^2$ 内の直線上に配置する。$\mathrm{JS}(p,q) = \log 2$ が、2 次元ガウスデータ分布 $p = \mathcal{N}(\mu_p, A^\top A)$ に対して任意の v と μ_q のもとで成り立つことを示せ。
   2. $\mathrm{JS}(\hat{p}, q) = \log 2$ が、1,000 個の訓練点の経験分布 $\hat{p}$ と任意の連続密度 q（$q = p$ を含む）に対して成り立つことを示せ。
   3. κ リプシッツなクリティック D に対して、カントロヴィッチ・ルビンシュタイン双対性（37.4.2 節）を用い、Equation 16.2 において p を $\hat{p}$ に置き換えても $V(D)$ の変化は高々 $\kappa\, W_1(\hat{p}, p)$ であることを示せ。リプシッツ定数を制限すると、なぜ経験サンプル上での訓練が有益な情報をもたらすようになるのか。
3. [code] **飽和するクリティック活性化関数。**●`make_nets` 内の Leaky ReLU 活性化関数を tanh ユニットに置き換え、隠れ層の幅を 5 と 3 にせよ。

1. 16.1.4.5 節の非飽和型のアームを 5 つのシードで再実行し、最終的な $D_{\mathrm{KL}}(q \,\|\, p)$ とクリティック損失を記録せよ。
2. 停滞した実行について、実サンプルおよび生成サンプルにおける隠れ層の活性化値と $\|\nabla_x D(x)\|$ を調べよ。クリティックの隠れユニットが飽和すると、なぜ非飽和生成器損失でも回復できなくなるのか。
3. 収束した生成器（$q = p$）と定数クリティック（$D(x) = 0$）はいずれもクリティック損失 $2\log 2$ を与える。本節のどの診断手法によって両者を区別できるか。

4. [code] **Fashion-MNIST 生成器のクラス被覆。●**

   1. Fashion-MNIST 上で LeNet（6.6 節）を訓練し、`net_G_img` からの 1,000 個のサンプルを分類して、予測クラスのヒストグラムを一様なデータ分布と比較せよ。
   2. 5 エポックおよび 40 エポック訓練した生成器に対して同様の実験を繰り返せ。訓練を長くすると、クラス分布はより一様になるか。
   3. 生成画像の一部がぼやけていたり曖昧だったりするとき、10 クラスへの強制的な分類はヒストグラムにどのようなバイアスを生じさせるか。得られた結果をモード欠落（16.1.3.3 節）と関連付けて論じよ。

5. **生成器の重みと対応するダイバージェンス。●●**生成器損失 $\mathbb{E}_z[g(D(G(z)))]$（クリティック D は固定）は、勾配 $-\mathbb{E}_z[w(D)\,\nabla_\theta D(G(z))]$（ただし $w = -g'$）を持つ。D が最適対数密度比 $\lambda = \log(p/q_{\theta_0})$（現在の生成器 q_{θ_0} に対するもの）に保たれていると仮定し、$\rho = e^D = p/q_{\theta_0}$ とおく。

   1. $\int \nabla_\theta q_\theta = 0$ を用い、重みなし損失 $\mathbb{E}_z[-D(G(z))]$（$w = 1$）の勾配が $\nabla_\theta D_{\mathrm{KL}}(q_\theta \,\|\, p)$（$\theta = \theta_0$ において）となることを示せ。
   2. f ダイバージェンス $D_f(p \,\|\, q_\theta) = \mathbb{E}_{q_\theta}[f(p/q_\theta)]$（37.3.1.2 節）に対して $\nabla_\theta D_f\big|_{\theta_0} = -\mathbb{E}_z\big[\rho^2 f''(\rho)\,\nabla_\theta D(G(z))\big]$ を示し、したがって重み関数 w が、f ダイバージェンスのうち $\rho^2 f''(\rho) = w(\log\rho)$ を満たすものに対応することを示せ。
   3. w_{sat}、$w = 1$、$w(t) = e^t$ (Goodfellow 2016)、および w_{ns} に対応するダイバージェンスをそれぞれ特定せよ（Arjovsky and Bottou (2017) の定理 2.5）。
   4. Equation 16.7 を用いて、$2\,D_{\mathrm{KL}}(m \,\|\, p)$ の値（p が互いに素で等重みな K 個のモードを持ち、q がそのうち 1 つだけをカバーする場合）を求め、K に対する増大の仕方を $D_{\mathrm{KL}}(q \,\|\, p) = \log K$ と比較せよ。

   Stefano Ermon と Aditya Grover による CS236: Deep Generative Models を改変。

6. [extended] **有限サンプルとクリティックの容量。●●**標本サイズがクリティックのパラメータ数を $\tilde{O}(1/\epsilon^2)$ 倍上回るとき、経験クリティック値は母集団での値から ϵ 以内に集中する (Arora et al. 2017)。生成器を真のデータ分布に固定する（$G(z) = zA + \mu_p$、すなわち $q = p$）。

   1. 幅 4、16、128、1,024 を持つ隠れ層 2 層のクリティックを、$n \in \{100, 1000, 10000\}$ 個の p からの固定訓練点上で訓練せよ（学習率 10^{-3} の Adam、バッチサイズ 256 で 4,000 ステップ）。訓練損失と新規サンプル上の母集団損失との間の汎化ギャップを、パラメータ数に対してプロットせよ。
   2. 16.1.5 節の Fashion-MNIST クリティックは $\sim$ 530,000 個のパラメータを持ち、60,000 枚の画像で訓練されている。Fashion-MNIST のテストセットを用いた検証を設計・実行し、その $2\log 2$ を下回る損失が訓練セットの暗記を反映しているのか、それとも p と q の真の乖離を反映しているのかを確かめよ。

16.2 敵対的目的関数とダイバージェンス

多くの GAN 実装には、差し替え可能な複数の訓練目的関数が用意されている。非飽和ロジスティック損失、最小二乗法（LSGAN）、ヒンジ損失、ワッサースタイン損失などである。16.1 節ではロジスティック目的関数を解析し、最適な識別器が対数密度比 $\lambda(x) = \log(p(x)/q(x))$ を復元すること、そしてそのミニマックス値がイェンセン・シャノン・ダイバージェンス（p と q のサポートが重ならなくなると飽和する量）になることを示した。では、他の目的関数はどのようなダイバージェンスを測っているのだろうか。また、生成サンプルがデータから遠く離れているとき、勾配消失の原因は損失関数、クリティックのアーキテクチャ、背後にあるダイバージェンスのどれにあるのだろうか。

この族に属する目的関数はすべて、クリティックの出力に適用されるスコア関数か、クリティックが最適化の対象とする関数クラス $\mathcal{T}$ のいずれかを変更したものである。制約なしのクリティックを分類損失と組み合わせると f ダイバージェンスを評価することになり、これは各点での密度比 $p(x)/q(x)$ のみに依存する。これに対し、線形の利得を用いてクリティックを滑らかな関数クラスに制約すると、最大平均値差異（maximum mean discrepancy、MMD）や 1 次ワッサースタイン距離（Wasserstein-1 distance、W_1）などの**積分確率指標**（integral probability metric、IPM）が得られる。これらはサポートが互いに素であっても空間的な距離に対する感度を保つ。以下では、これら両方の族を導出し、離れた 1 次元ガウス分布と 2 次元混合ガウス分布における挙動を比較する。

```
# Deterministic GPU kernels; JAX reads XLA_FLAGS once, before the imports
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

```
from jax.nn import relu, softplus
```

16.2.1 一般的なテンプレート

16.1 節では、クリティックは $\log\sigma(D(x))$（実サンプル $x \sim p$ に対する値）と $\log\sigma(-D(x'))$（生成サンプル $x' \sim q$ に対する値）を受け取っていた。これらの対数尤度項を一般的な利得関数 ℓ_p と ℓ_q に置き換え、クリティックのクラス $\mathcal{T}$ 上で考えることで、一般的な敵対的乖離度が得られる。

$$d(p, q) = \sup_{T \in \mathcal{T}} \Big\{ \mathbb{E}_{x \sim p}\big[\ell_p(T(x))\big] + \mathbb{E}_{x' \sim q}\big[\ell_q(T(x'))\big] \Big\}. \tag{16.9}$$

$\ell_p(t) = \log\sigma(t)$、$\ell_q(t) = \log\sigma(-t)$ とおき、$\mathcal{T}$ に制約を課さない場合、$2\,\mathrm{JS}(p, q) - 2\log 2$ が復元される。本節を通じて、クリティックは期待利得の和を最大化する（同値な表現として、負の利得を最小化するものとし、これをクリティック損失と呼ぶ）。

Table 16.2 は、本章で扱う目的関数を 2 つの軸に沿って整理したものである。すなわち、クリティックが個々のサンプルにスコアを付けるか実サンプルと生成サンプルのペアにスコアを付けるかという軸と、クリティックのクラス $\mathcal{T}$ が制約なしか正則化されているかという軸である。

Table 16.2: クリティックの入力と関数クラスの制約による敵対的目的関数の分類。

	制約なしのクリティック	制約付き・ペナルティ付きのクリティック
単一サンプルクリティック	分類損失（ロジスティック、最小二乗法、ヒンジ）$\to f$ ダイバージェンス（比 p/q に基づく）	RKHS 球（MMD）またはリプシッツ球（W_1）上の線形利得、スペクトル正規化付きヒンジ損失
ペアワイズクリティック	相対論的ペアリング損失（RpGAN、16.3 節）	ゼロ中心勾配ペナルティ $R_1 + R_2$ 付きペアリング損失（R3GAN、16.4 節、16.5 節）

なお、任意の固定されたクリティック T に対し、Equation 16.9 内の期待値は分布 q に関して線形である。線形汎関数の族の上限は凸であるため、$d(p, q)$ は確率測度 q に関して（p を固定したとき）常に凸となる。しかし、分布の集合 $\{q_\theta\}$（ニューラルネットワーク G_θ によってパラメータ化されるもの）は凸集合ではないため、分布空間における凸性から生成器パラメータ θ に関する凸性は導かれない（演習問題 1）。

16.2.2 分類損失と f ダイバージェンス

16.2.2.1 2 値結果の例

表と裏の 2 通りの結果からなる空間を考え、データ分布 $p = (\frac{3}{4}, \frac{1}{4})$ と生成器分布 $q = (\frac{1}{4}, \frac{3}{4})$ から等しい事前確率 $\frac{1}{2}$ でサンプリングするものとする。結果が与えられたとき、それが p から生じたベイズ事後確率は $\eta = p/(p+q)$ であり、表なら $\frac{3}{4}$、裏なら $\frac{1}{4}$ となる。結果を見ない当て推量の予測（$D = 0$）は、結果にかかわらず $\frac{1}{2}$ を予測する。結果を観測することによる期待損失の減少量は、分類損失に応じて次のように異なる。

- **対数損失**：最適な予測 η は二値エントロピー $H_b(\frac{3}{4}) = 0.562$ nat を達成し、当て推量の $\log 2 = 0.693$ nat と比べて 0.131 nats $= \mathrm{JS}(p, q)$ だけ損失を減らす。
- 確率に対する二**乗誤差**：η を予測すると期待損失は $\frac{3}{4} \cdot \frac{1}{4} = \frac{3}{16}$ となり、損失 $\frac{1}{4}$（$\frac{1}{2}$ を予測する場合）と比べて $\frac{1}{16}$ だけ減少する。
- スコア t に対する**ヒンジ損失**：最適なスコア $t = +1$（表）と $t = -1$（裏）は期待損失 $2 \cdot \frac{1}{4} = \frac{1}{2}$ を達成し、損失 1（$t = 0$ の場合）と比べて $\frac{1}{2} = \mathrm{TV}(p, q) = \frac{1}{2}\sum |p - q|$ だけ減少する。

いずれの損失も、同じ組 (p, q) に対して同じ事後確率 η に基づきながら、異なる非負のダイバージェンスを割り当てている。以下では、マージンに基づくすべての分類損失が f ダイバージェンスを定めることを示そう。

16.2.2.2 ベイズリスクギャップ

対称な分類損失は、単一のマージン利得 ℓ を実サンプル上の $D(x)$ と生成サンプル上の $-D(x')$ に適用する（$\ell_p(t) = \ell(t)$、$\ell_q(t) = \ell(-t)$）。

$$V_\ell(D) \;=\; \mathbb{E}_{x\sim p}\big[\ell(D(x))\big] + \mathbb{E}_{x'\sim q}\big[\ell(-D(x'))\big]. \tag{16.10}$$

ここで符号付きスコア t（実データでは $+D(x)$、生成データでは $-D(x')$）は分類の**マージン**である。各点 x において、$p(x)\ell(t) + q(x)\ell(-t) = 2m(x)\big[\eta(x)\ell(t) + (1-\eta(x))\ell(-t)\big]$ が成

り立つ。ただし、$m = (p+q)/2$ および $\eta(x) = p(x)/(p(x)+q(x))$ である。事後確率 η における各点での最小期待損失を**条件付きベイズリスク**（conditional Bayes risk）という。

$$L(\eta) \;=\; \inf_{t} \Big\{ \eta\,(-\ell(t)) + (1-\eta)\,(-\ell(-t)) \Big\}. \tag{16.11}$$

η のアフィン関数の下限として定義されるため、$L(\eta)$ は $[0,1]$ 上で凹関数となる。定数クリティック（$t=0$）はリスク $L(\tfrac{1}{2})$ を達成する。x を観測することによるリスクの減少量が**ベイズリスクギャップ**である。

$$\Delta_{\ell}(p,q) \;=\; L(\tfrac{1}{2}) - \mathbb{E}_{x\sim m}\big[L(\eta(x))\big], \tag{16.12}$$

これにより最適値 $\sup_{D} V_{\ell}(D) \;=\; 2\Delta_{\ell}(p,q) - 2L(\tfrac{1}{2})$ が決まる。イェンセンの不等式と $\mathbb{E}_{m}[\eta] = \tfrac{1}{2}$ より、$\Delta_{\ell}(p,q) \ge 0$ が成り立つ。

16.2.2.3 f ダイバージェンスとしてのベイズリスクギャップ

37.3.1.2 節で見たように、f ダイバージェンスは凸関数 f（ただし $f(1)=0$）を密度比 $\rho(x) = p(x)/q(x)$ に適用し、q のもとで平均した量 $D_f(p\,\|\,q) = \mathbb{E}_{x\sim q}[f(\rho(x))]$ であった。

命題。 条件付きベイズリスク L が $[0,1]$ 上で有限ならば、次が成り立つ。

$$\begin{aligned} &\Delta_{\ell}(p,q) = D_{f}(p\,\|\,q) \\ &\text{with}\quad f(u) = L(\tfrac{1}{2}) - \frac{u+1}{2}\,L\!\left(\frac{u}{u+1}\right), \end{aligned}$$

ここで、f は $[0,\infty)$ 上で凸関数であり、$f(1)=0$ を満たす。

証明。 $u = \rho = p/q$ とおくと、$m = q\,(u+1)/2$ および $\eta = u/(u+1)$ となるので、次を得る。

$$\mathbb{E}_{x\sim m}\big[L(\eta(x))\big] = \int q\,\frac{u+1}{2}\,L\!\left(\frac{u}{u+1}\right),$$

さらに $L(\tfrac{1}{2}) = \int q\,L(\tfrac{1}{2})$ を積分の内側に入れると、$\Delta_{\ell}(p,q) = \mathbb{E}_{q}[f(\rho)]$ が得られる。凹関数 $L(v) = \inf_{j}\{\alpha_j v + \beta_j\}$ をその接線の下側包絡線として表せば、$(u+1)L(u/(u+1)) = \inf_{j}\{\alpha_j u + \beta_j (u+1)\}$ が u に関して凹であることがわかる。したがって $f(u)$ は凸であり、$f(1) = L(\tfrac{1}{2}) - L(\tfrac{1}{2}) = 0$ となる。$\blacksquare$

以上より、制約なしのクリティックと対称な分類損失の組み合わせはすべて f ダイバージェンスを評価し、その最適クリティック $D^{\star}(x)$ は対数密度比 $\lambda(x) = \log(p(x)/q(x))$ の単調な**リンク関数**（link function）となる。

16.2.2.4 代表的な分類損失のダイバージェンス

Table 16.3 に、代表的な 4 つの分類損失と、それぞれの条件付きベイズリスク $L(\eta)$、ベイズリスクギャップ $\Delta_{\ell}(p,q)$、最適クリティックのリンク $D^{\star}(\lambda)$、および実用上それらと組み合わせて用いられる生成器損失をまとめる。

Table 16.3: 制約なしのクリティックにおける分類損失。マージン損失 $-\ell(t)$、ベイズリスク $L(\eta)$、ベイズリスクギャップ $\Delta_\ell(p,q)$、最適クリティックのリンク $D^\star(\lambda)$、および $D = D(x')$ に対する実用上の生成器損失。

損失	マージン損失 $-\ell(t)$	ベイズリスク $L(\eta)$	ギャップ $\Delta_\ell(p,q)$	最適クリティックのリンク	生成器損失
ロジスティック	$\log(1+e^{-t})$	$H_b(\eta)$	$\mathrm{JS}(p,q)$	λ	$-\log\sigma(D)$
最小二乗法（LSGAN）	$\frac{1}{4}(1-t)^2$、$t = 2D-1$	$\eta(1-\eta)$	$\frac{1}{8}\int \frac{(p-q)^2}{p+q}$	$\sigma(\lambda)$	$\frac{1}{2}(D-1)^2$
二乗ヒンジ	$\max(0,1-t)^2$	$4\eta(1-\eta)$	$\frac{1}{2}\int \frac{(p-q)^2}{p+q}$	$\tanh(\lambda/2)$	$-D$
ヒンジ	$\max(0,1-t)$	$2\min(\eta,1-\eta)$	$\mathrm{TV}(p,q)$	$\operatorname{sign}\lambda$	$-D$

LSGAN では、クリティックはターゲット 1（本物）と 0（偽物）に回帰する (Mao et al. 2017)。これは $t = 2D-1$ のもとで二乗マージン損失 $\frac{1}{4}(1-t)^2$ に対応し、Vincze–Le Cam ダイバージェンス $\chi^2(p\,\|\,m) = \frac{1}{2}\int (p-q)^2/(p+q)$ の 4 分の 1 を評価する。ヒンジ損失は全変動 $\mathrm{TV}(p,q) = \frac{1}{2}\int |p-q|$ を評価し (Lim and Ye 2017)、その最適クリティックは $\operatorname{sign}\lambda$ を出力する（$[-1,1]$ 内の任意の値が $\eta = \frac{1}{2}$ において最適となる）。二乗ヒンジ損失はヒンジマージンおよび生成器目的関数 $-D(x')$ を共有するが、滑らかなリンク $\tanh(\lambda/2)$ によって λ の大きさを保持する。実用上は、いずれの損失も平衡点 $q = p$ を共有する非飽和型の生成器目的関数（最後の列）を用いる。

16.2.2.5 フェンシェル・ヤング変分下界（f-GAN）

Equation 16.10 は D と $-D$ に同じ利得を適用するため、そのベイズリスクは $L(\eta) = L(1-\eta)$ を満たす。したがって、すべての分類ギャップ $\Delta_\ell(p,q)$ は p と q に関して対称であり、上から $L(\frac{1}{2})$ で抑えられる（演習問題 5）。非対称または非有界な f ダイバージェンス（順方向 KL ダイバージェンス $D_{\mathrm{KL}}(p\,\|\,q)$ など）を扱うには、非対称な利得 $\ell_p \neq \ell_q$ が必要になる。

$f^*(t) = \sup_u\{ut - f(u)\}$ を凸ダイバージェンス関数 f の**凸共役**（convex conjugate）とする。フェンシェル・ヤングの不等式より、$f(u) \geq ut - f^*(t)$ がすべての u, t について成り立ち、等号は $t = f'(u)$ のときに成立する。$u = \rho(x) = p(x)/q(x)$、$t = T(x)$ とおいて q のもとで期待値をとると（$\mathbb{E}_q[\rho\, T] = \mathbb{E}_p[T]$ を用いる）、f-GAN の変分下界が得られる (Nowozin et al. 2016)（37.3.4 節）。

$$D_f(p\,\|\,q) \;\geq\; \mathbb{E}_{x\sim p}[T(x)] - \mathbb{E}_{x'\sim q}[f^*(T(x'))]. \tag{16.13}$$

これはテンプレート Equation 16.9 において $\ell_p(t) = t$、$\ell_q(t) = -f^*(t)$ とした場合に一致し、最適クリティック $T^\star(x) = f'(\rho(x))$ で等号を達成する（演習問題 2）。順方向 KL ダイバージェンスでは、$f(u) = u\log u$ の共役は $f^*(t) = e^{t-1}$、最適クリティックは $T^\star(x) = 1 + \log\rho(x) = 1 + \lambda(x)$ となる。

16.2.2.6 クリティックのリンク関数の実験的検証

ここで、ロジスティック、最小二乗、ヒンジ、順方向 KL の各目的関数のもとで訓練したクリティックが、$\lambda(x)$ に関するそれぞれの理論的リンク関数（λ、$\sigma(\lambda)$、$\operatorname{sign}\lambda$、$1+\lambda$）を学習することを検証しよう。$p$ を 3 つの 2 次元ガウス分布の一様混合として定義し、摂動を加えた 3 成分の混合分布 q を構成することで、いたるところで $\lambda(x)$ が解析的に求まるようにする。

```
mu_p = np.array([[0.0, 2.4], [-2.1, -1.2], [2.1, -1.2]])     # data
A_p = np.stack([0.7 * np.eye(2)] * 3)

mu_q = np.array([[0.2, 2.6], [-2.3, -1.0], [2.0, -1.3]])     # hand-set q
A_q = np.stack([np.diag([0.8, 0.55]), np.diag([0.5, 0.6]),
                np.array([[0.65, 0.2], [-0.1, 0.7]])])


def mixture_logpdf(x, mus, As):
    """Log density of the uniform mixture of Gaussians N(mu_k, A_k^T A_k)."""
    comps = []
    for mu, A in zip(mus, As):
        Sigma, diff = A.T @ A, x - mu
        sol = np.linalg.solve(Sigma, diff.T).T
        comps.append(-0.5 * ((diff * sol).sum(1)
                             + np.linalg.slogdet(2 * np.pi * Sigma)[1]))
    return np.logaddexp.reduce(np.stack(comps) - np.log(len(mus)), axis=0)


def sample_mixture(n, mus, As, rng):
    """n draws from the uniform mixture of Gaussians mu_k + z A_k."""
    k = rng.integers(0, len(mus), n)
    z = rng.standard_normal((n, 2))
    return np.einsum('ni,nij->nj', z, As[k]) + mus[k]
```

固定グリッド上の数値積分により、$\mathrm{JS}(p, q)$ がこの形式の任意の混合分布 q に対して求まる。本節の後半では、同じ関数を用いて訓練中の生成器をステップごとにスコア付けする。

```
grid_1d = np.linspace(-7.0, 7.0, 281)
grid_2d = np.stack([g.ravel() for g in np.meshgrid(grid_1d, grid_1d)], axis=1)
cell_area = (grid_1d[1] - grid_1d[0]) ** 2


def js_mixtures(mus_q, As_q, mus_p=mu_p, As_p=A_p):
    """JS(p, q) between two known mixtures, p the data by default, by
    quadrature on grid_2d."""
    logp = mixture_logpdf(grid_2d, mus_p, As_p)
    logq = mixture_logpdf(grid_2d, mus_q, As_q)
    p, q = np.exp(logp), np.exp(logq)
    logm = np.log(np.maximum((p + q) / 2, 1e-300))
    return 0.5 * (p * (logp - logm) + q * (logq - logm)).sum() * cell_area

print(f'JS(p, q) for the hand-set q: {js_mixtures(mu_q, A_q):.3f} nats')
```

```
JS(p, q) for the hand-set q: 0.033 nats
```

手動で設定したこの q はデータから 100 分の数 nat だけ離れており、これは生成器が訓練の終盤で直面する程度のずれである。クリティックには、ユニット数 64 の隠れ層を 2 つもつ小規模な Leaky ReLU ネットワークを用いる。

```
def make_critic(rngs):
    """Leaky-ReLU MLP; nnx.Linear gives LeCun-normal weights, zero biases."""
    leaky = lambda x: nnx.leaky_relu(x, 0.2)
    return nnx.Sequential(nnx.Linear(2, 64, rngs=rngs), leaky,
                          nnx.Linear(64, 64, rngs=rngs), leaky,
                          nnx.Linear(64, 1, rngs=rngs))
```

各損失は、Table 16.3 に示したクリティックの損失と生成器の損失のペアとして格納し、いずれも実バッチに対するクリティックのスコア `r` と生成バッチに対するスコア `f` の関数として記述する。

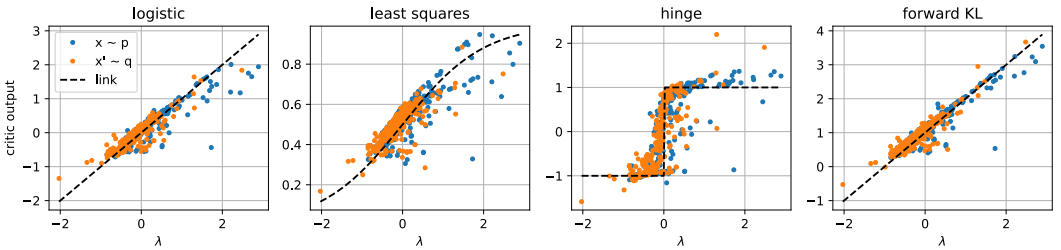
```
losses = {  # name: (critic loss, generator loss), both on the scores r, f
    'logistic': (lambda r, f: (softplus(-r) + softplus(f)).mean(),
                 lambda r, f: softplus(-f).mean()),
    'least squares': (lambda r, f: ((r - 1) ** 2 + f ** 2).mean() / 2,
                      lambda r, f: ((f - 1) ** 2).mean() / 2),
    'hinge': (lambda r, f: (relu(1 - r) + relu(1 + f)).mean(),
              lambda r, f: -f.mean()),
    'squared hinge': (lambda r, f: (relu(1 - r) ** 2 + relu(1 + f) ** 2).mean(),
                      lambda r, f: -f.mean()),
}
```

リンク関数は固定された q に対する最適応答を表すため、手動設定した q に対して 3 つの損失のもとでそれぞれ新しいクリティックを訓練し、4 つ目のクリティックは順方向 KL 下界 $\mathbb{E}_p[T] - \mathbb{E}_q[e^{T-1}]$ を最大化するように訓練する。

```
@nnx.jit(static_argnums=0)
def best_response_step(critic_loss, D, opt, X, Y):
    opt.update(D, nnx.grad(lambda D_: critic_loss(D_(X), D_(Y)))(D))

def best_response(critic_loss, seed=0, steps=3000, batch=512, lr=1e-3):
    """Fresh critic trained against the fixed q, as a NumPy function."""
    rng = np.random.default_rng(seed)
    D = make_critic(nnx.Rngs(seed))
    opt = nnx.Optimizer(D, optax.adam(lr), wrt=nnx.Param)
    for _ in range(steps):
        X = jnp.asarray(sample_mixture(batch, mu_p, A_p, rng), jnp.float32)
        Y = jnp.asarray(sample_mixture(batch, mu_q, A_q, rng), jnp.float32)
        best_response_step(critic_loss, D, opt, X, Y)
    return lambda x: np.asarray(D(jnp.asarray(x, jnp.float32))).squeeze(-1)

kl_dual = lambda r, f: -(r.mean() - jnp.exp(f - 1).mean())
critics = {name: best_response(losses[name][0])
           for name in ('logistic', 'least squares', 'hinge')}
critics['forward KL'] = best_response(kl_dual)
```

訓練した各クリティックを p からの 300 点と q からの 300 点で評価し、真の対数密度比 $\lambda(x)$ に対してその出力を理論的リンク関数（破線）とともにプロットする。

```
rng = np.random.default_rng(7)
pts = np.concatenate([sample_mixture(300, mu_p, A_p, rng),
                      sample_mixture(300, mu_q, A_q, rng)])
lam = mixture_logpdf(pts, mu_p, A_p) - mixture_logpdf(pts, mu_q, A_q)
lam_line = np.linspace(lam.min(), lam.max(), 200)
links = {'logistic': lambda l: l, 'least squares': lambda l: 1 / (1 + np.exp(-l)),
         'hinge': np.sign, 'forward KL': lambda l: 1 + l}
fig, axes = d2l.plt.subplots(1, 4, figsize=(12, 3))
for ax, (name, link) in zip(axes, links.items()):
    out = critics[name](pts)
    d2l.plot([lam[:300], lam[300:], lam_line],
             [out[:300], out[300:], link(lam_line)], xlabel=r'$\lambda$',
             fmts=('.', '.', 'k--'), axes=ax,
             legend=['x ~ p', "x' ~ q", 'link'] if name == 'logistic' else None)
    ax.set_title(name)
axes[0].set_ylabel('critic output')
fig.tight_layout()
```



4 つのネットワークはいずれも $\lambda(x)$ の理論的リンク関数、すなわち恒等関数 λ、シグモイド $\sigma(\lambda)$、階段関数 sign λ、平行移動した恒等関数 $1+\lambda$ によく追従している。なお、ヒンジクリティックでは $\lambda = 0$ 付近でばらつきが大きくなっているが、これは $[-1, 1]$ 内のどの出力も最小条件付きリスクを達成するためである。

16.2.3 積分確率指標

f ダイバージェンスが制約なしのクリティックと非線形な利得を組み合わせるのに対し、**積分確率指標**（IPM）は線形の利得 $\ell_p(t) = t$ と $\ell_q(t) = -t$ を用い、代わりにクリティックを対称な関数クラス $\mathcal{F}$ に制限する（37.4.1 節）。

$$d_{\mathcal{F}}(p, q) \;=\; \sup_{T \in \mathcal{F}} \Big\{ \mathbb{E}_{x \sim p}[T(x)] - \mathbb{E}_{x' \sim q}[T(x')] \Big\}. \tag{16.14}$$

$\mathcal{F}$ に制約がなければ、T を任意の正定数でスケールすることで、期待値の差は $+\infty$ へ発散してしまう（$p \neq q$ である限り）。$\mathcal{F}$ が滑らかな関数からなるとき、$d_{\mathcal{F}}(p, q)$ は $\mathcal{X}$ の空間的な幾何構造に依存し、単なる各点での比 $p(x)/q(x)$ だけでは決まらない。

16.2.3.1 最大平均値差異（MMD）

正定値カーネル $k(x, x')$（ガウス RBF カーネル $k(x, x') = \exp(-\|x - x'\|^2/(2s^2))$、バンド幅 s など）が与えられたとき、$\mathcal{F}$ として k に対応する再生核ヒルベルト空間（reproducing kernel Hilbert space、RKHS）の単位球を選ぶと、**最大平均値差異**が得られる (Gretton et al. 2012)（37.4.1 節）。その二乗値は、ペアごとのカーネル期待値を用いて閉形式で表せる。

$$\mathrm{MMD}^2(p,q) = \mathbb{E}_{x,\tilde{x}\sim p}[k(x,\tilde{x})] + \mathbb{E}_{x',\tilde{x}'\sim q}[k(x',\tilde{x}')] - 2\,\mathbb{E}_{x\sim p,\,x'\sim q}[k(x,x')]. \tag{16.15}$$

（定数倍を除いて）最適なテスト関数は**証人関数**（witness function）$w(x) = \mathbb{E}_{\tilde{x}\sim p}[k(x,\tilde{x})] - \mathbb{E}_{\tilde{x}'\sim q}[k(x,\tilde{x}')]$ であり、ある点と p との平均類似度から q との平均類似度を引いた差を測る。Equation 16.15 はクリティックのネットワークを必要としないため、生成器は有限標本の不偏推定量 `mmd2_unbiased` を直接最小化できる（この関数は 16.5 節で KID の計算にも用いる）。

```
def mmd2_unbiased(K_pp, K_qq, K_pq):
    """Unbiased MMD^2 from the kernel matrices within p, within q, and
    across; the diagonals (a point with itself) are left out."""
    m, n = len(K_pp), len(K_qq)
    return ((K_pp.sum() - K_pp.diagonal().sum()) / (m * (m - 1))
            + (K_qq.sum() - K_qq.diagonal().sum()) / (n * (n - 1))
            - 2 * K_pq.mean())
```

16.2.3.2 1 次ワッサースタイン距離（W_1）

$\mathcal{F}$ として **1 リプシッツ**関数のクラス（$|T(x) - T(y)| \le \|x - y\|$ がすべての x, y について成り立つもの）を選ぶと、カントロヴィチ・ルビンシュタイン双対性（37.4.2 節）により **1 次ワッサースタイン**距離（または**アースムーバー**距離）が得られる。

$$W_1(p,q) \;=\; \inf_{\pi\in\Pi(p,q)} \mathbb{E}_{(x,x')\sim\pi}[\|x - x'\|],$$

ここで、$\Pi(p,q)$ は周辺分布が p と q である同時輸送計画の集合である。最適な 1 リプシッツクリティック（**カントロヴィチポテンシャル**）は、q から p へ向かう最適輸送の半直線に沿って単位勾配ノルムをもつ。1 次元では $W_1(p,q) = \int_{-\infty}^{\infty} |F_p(t) - F_q(t)|\,dt$ となるため、分布を μ だけ平行移動すると $W_1(p,q) = |\mu|$ となり、どれだけ離れていても一定の傾き 1 をもつ。

より高い次元では、Wasserstein GAN（WGAN）はニューラルネットワークのクリティック T に対する 1 リプシッツ制約を主に次の 3 つの仕組みで近似する。

1. **重みクリッピング**（weight clipping）は、重みを $[-c, c]$ にクランプし、表現容量を制限する引き換えとして κ リプシッツ性（κ はアーキテクチャに依存する定数）を強制する (Arjovsky et al. 2017)。
2. **勾配ペナルティ**（gradient penalty、WGAN-GP）は、$\gamma\,\mathbb{E}[(\|\nabla_x T(\tilde{x})\| - 1)^2]$ を実サンプルと生成サンプルのランダムな内分点 $\tilde{x} = ux + (1-u)x'$ において加算する (Gulrajani et al. 2017)。
3. **スペクトル正規化**（spectral normalization）は、各重み行列をその最大特異値 $\sigma_{\max}(W)$ で正規化し、すべての線形層のリプシッツ定数を 1 以下に抑える (Miyato et al. 2018)。

なお、ヒンジ損失はこれら 2 つの族を橋渡ししている。$|D(x)| \le 1$ の範囲では利得が線形であるため、制約なしのヒンジクリティックは $|D(x)| \le 1$ で有界な関数クラス上の IPM に対応する（評価値は $2\,\mathrm{TV}(p,q) - 2$）。ヒンジ損失をスペクトル正規化と組み合わせると（SAGAN や BigGAN のように、D を有界かつ κ リプシッツにすると）、その値が κW_1 のように分離距離 $2/\kappa$ まで線形に増大し、その先で飽和するクリティックが得られる。

16.2.3.3 高次元におけるサンプル複雑度

固定カーネルの MMD と経験 W_1 はサンプル複雑度が大きく異なる。MMD^2 の不偏推定量は次元に依存しないレート $O(n^{-1/2})$ で収束する (Gretton et al. 2012)（ただし推定対象である固定カーネル下での所与のシフトに対する MMD^2 自体が、次元の増大に伴って縮小しうる (Ramdas et al. 2015))。これに対し、プラグイン経験 W_1 距離は次元 $d_x > 2$ において次元の呪いを受け、レート $O(n^{-1/d_x})$ でしか収束しない (Weed and Bach 2019)。一方、変分 f ダイバージェンス推定量は $\rho = p/q$ が大きいときに大きな分散に悩まされることがある。たとえば順方向 KL 双対下界では、$e^{T^\star(x')-1} = \rho(x')$ の q のもとでの分散は $\chi^2(p \,\|\, q)$ に等しくなり、$D_{\mathrm{KL}}(p \,\|\, q) < \infty$ であっても発散しうる（演習問題 4)。

16.2.4 データから遠い領域と近い領域における勾配

16.2.4.1 ダイバージェンスの傾きと分離距離

2 つの分布が離れるにつれて母集団ダイバージェンスがどう振る舞うかを見るため、$p = \mathcal{N}(0, 1)$ と $q = \mathcal{N}(\mu, 1)$ を分離距離 $\mu > 0$ の関数として考えよう。このとき $W_1(p, q) = \mu$ は定数の導関数 1 をもち、$\mathrm{JS}(p, q)$ は数値積分でき、$\mathrm{MMD}^2(p, q)$ はバンド幅 s のガウス RBF カーネルのもとで次の厳密なガウス積分をもつ。

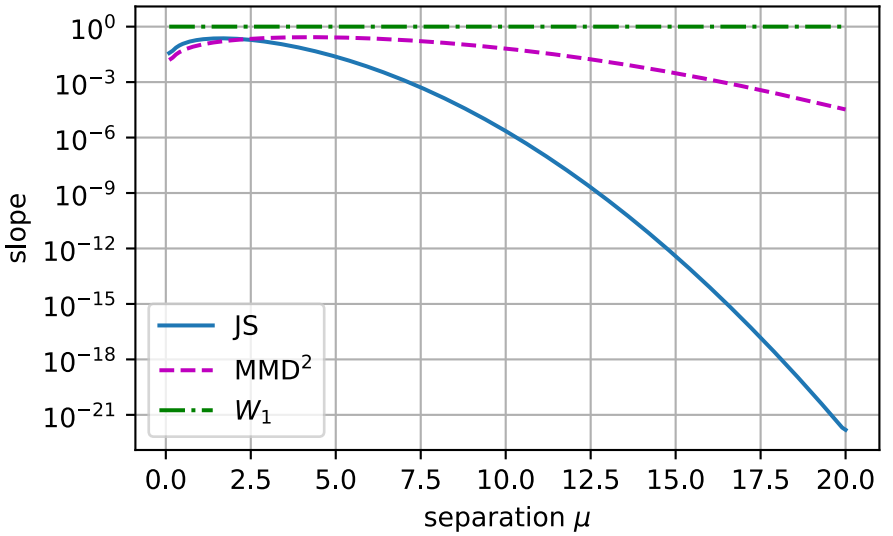
$$\mathrm{MMD}^2(p, q) \;=\; \frac{2s}{\sqrt{s^2 + 2}} \left(1 - \exp\left(-\frac{\mu^2}{2(s^2 + 2)}\right)\right), \tag{16.16}$$

この式の μ に関する導関数は $\mu = \sqrt{s^2 + 2}$（≈ 4.2、ただし $s = 4$ の場合）でピークに達し、その後 $e^{-\mu^2/(2(s^2+2))}$ のオーダーで減衰する。

```
xs = np.linspace(-10.0, 30.0, 8001)           # fixed quadrature grid

def normal_pdf(x, mu, sigma=1.0):
    return np.exp(-(x - mu) ** 2 / (2 * sigma ** 2)) / np.sqrt(
        2 * np.pi * sigma ** 2)

def js_gap(p, q):
    """log 2 - JS(p, q) on xs, accurate even when tiny: the integral of
    (p log(1 + q/p) + q log(1 + p/q)) / 2, with log(1 + e^a) computed as
    logaddexp(0, a)."""
    a = np.log(q) - np.log(p)
    return 0.5 * np.trapezoid(p * np.logaddexp(0, a) + q * np.logaddexp(0, -a),
                              xs)

def js_1d(p, q):
    return np.log(2) - js_gap(p, q)

s = 4.0                                        # RBF kernel bandwidth
mus = np.linspace(0.1, 20.0, 200)              # separations mu
gaps = np.array([js_gap(normal_pdf(xs, 0.0), normal_pdf(xs, mu))
                 for mu in mus])
mmd2 = 2 * s / np.sqrt(s ** 2 + 2) * (1 - np.exp(-mus ** 2
                                                 / (2 * (s ** 2 + 2))))
slopes = [-np.gradient(gaps, mus), np.gradient(mmd2, mus),
```

```
            np.ones_like(mus)]                    # W1 = |mu| has slope one
    for mu in (1.0, 7.0, 20.0):
        js = js_1d(normal_pdf(xs, 0.0), normal_pdf(xs, mu))
        print(f'mu = {mu:2.0f}: JS slope {np.interp(mu, mus, slopes[0]):.4g}, '
              f'JS / log 2 = {js / np.log(2):.4f}, '
              f'MMD^2 slope {np.interp(mu, mus, slopes[1]):.4g}')
    peak = slopes[1].argmax()
    print(f'MMD^2 slope peaks at mu = {mus[peak]:.1f} with {slopes[1][peak]:.3f}')
    d2l.plot(mus, slopes, xlabel=r'separation $\mu$', ylabel='slope',
             legend=['JS', 'MMD$^2$', '$W_1$'], yscale='log', figsize=(5, 3))
```

```
mu =  1: JS slope 0.1987, JS / log 2 = 0.1607, MMD^2 slope 0.1019
mu =  7: JS slope 0.001264, JS / log 2 = 0.9990, MMD^2 slope 0.188
mu = 20: JS slope 1.549e-22, JS / log 2 = 1.0000, MMD^2 slope 3.303e-05
MMD^2 slope peaks at mu = 4.2 with 0.269
```



上のプロットおよび Figure 16.4 に示すように、JS(p, q) の導関数は $\mu = 7$ までに 2 桁以上低下し（0.0013 対 0.20、後者は $\mu = 1$ の値）、1.5×10^{-22}（$\mu = 20$ の値）に達する。対照的に、W_1 はあらゆる距離で傾き 1 を保ち、MMD^2 はカーネルバンド幅 s 程度の分離距離まで強い勾配を保つ。

16.2.4.2 分離した初期状態から訓練したクリティック

理論上のダイバージェンス曲線は、すべての可測関数にわたる最適なクリティックを仮定している。しかし実際には、クリティックは生成器の更新 1 回あたり数ステップだけ訓練される有限のニューラルネットワークである。離れた初期化から訓練されたクリティックがどう振る舞うかを調べるため、1 次元の位置生成器 $q_\theta = \mathcal{N}(\theta, 1)$（$x' = \theta + z$）を $\theta_0 = 6$ から出発させて $p = \mathcal{N}(0, 1)$ に当てはめる。ここでは θ を素の確率的勾配降下法（SGD）で更新し（ステップサイズが勾配の大きさを直接反映するようにするため）、2 隠れ層の ReLU クリティックを生成器の 1 ステップにつき 5 回更新する。

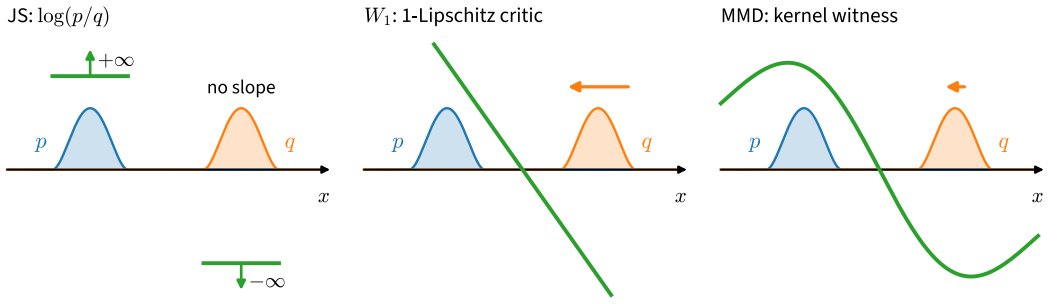


Figure 16.4: p（青）と q（オレンジ）が互いに素なサポートをもつときの最適クリティック。左：JS の最適クリティック $\log(p/q)$ は値 $+\infty$（$\mathrm{supp}(p)$ 上）と $-\infty$（$\mathrm{supp}(q)$ 上）をとり、両方の領域上で傾きが 0 になる。中央：W_1 のカントロヴィチポテンシャルは傾き 1 の線形ランプである。右：MMD の証人関数 $\mathbb{E}_p[k(x,\cdot)] - \mathbb{E}_q[k(x,\cdot)]$ はカーネルの山同士の滑らかな差であり、その裾は p に向かって傾いている。

```
def mmd2_loss(x, y, bandwidths=(0.5, 1.0, 2.0)):
    """Unbiased MMD^2 under the sum of RBF kernels of the given bandwidths."""
    def kernel(a, b):
        sq_dist = ((a[:, None] - b[None]) ** 2).sum(-1)
        return sum(jnp.exp(-sq_dist / (2 * s ** 2)) for s in bandwidths)
    return mmd2_unbiased(kernel(x, x), kernel(y, y), kernel(x, y))

def mmd2_median(x, y):
    """MMD^2 with one RBF kernel, its bandwidth the median pairwise distance
    of the pooled batch, held constant in the gradient."""
    xy = jnp.concatenate([x, y])
    dist = jnp.sqrt(((xy[:, None] - xy[None]) ** 2).sum(-1))
    return mmd2_loss(x, y, (jax.lax.stop_gradient(jnp.median(dist)),))


def gradient_penalty(D, X, fake, u, one_sided=False):
    """Mean (|D'(x)| - 1)^2 at the points u X + (1 - u) fake (inputs 1-D);
    one-sided, only slopes steeper than one pay: max(0, |D'(x)| - 1)^2."""
    grad = jax.grad(lambda x: D(x).sum())(u * X + (1 - u) * fake)
    excess = jnp.abs(grad) - 1
    return ((jnp.maximum(excess, 0) if one_sided else excess) ** 2).mean()
```

`train_shift` は生成器の 1 ステップごとにクリティックを Adam で 5 ステップ更新し、クリティックを最適応答の近くに保つ。生成器には学習率 0.01 の素の勾配ステップをとらせる。もし Adam を使うと、各ステップがほぼ学習率の大きさにリスケールされてしまい、ここで観察したい勾配の大きさが隠れてしまうからである。この関数は各ステップにおける θ と生成器の勾配を記録する。正の `gp` を指定するとクリティック損失に勾配ペナルティが加わり、`one_sided=True` にすると傾きが 1 より急な部分のみにペナルティ $\max(0, |T'| - 1)^2$ を課す。また、`critic_loss` を `None` にすると、生成器は 2 つのバッチに対する `gen_loss` を直接最小化する。

```
class Shift(nnx.Module):
    """The generator x' = theta + z."""
```

```
    def __init__(self, theta0):
        self.theta = nnx.Param(jnp.array(theta0))

    def __call__(self, z):
        return self.theta + z


@nnx.jit(static_argnums=(0, 1, 2, 3))
def shift_step(critic_loss, gen_loss, gp, one_sided, G, D, opt_G, opt_D, X, Z,
               U):
    """One critic step per row of U, then one generator step; returns the
    generator's gradient."""
    for i in range(len(U)):
        fake = G(Z[i])
        def d_loss(D_):
            loss = critic_loss(D_(X[i]), D_(fake))
            if gp:
                loss = loss + gp * gradient_penalty(D_, X[i], fake, U[i],
                                                    one_sided)
            return loss
        opt_D.update(D, nnx.grad(d_loss)(D))
    def g_loss(G_):
        fake = G_(Z[-1])
        return (gen_loss(X[-1], fake) if critic_loss is None
                else gen_loss(D(X[-1]), D(fake)))
    grad = nnx.grad(g_loss)(G)
    opt_G.update(G, grad)
    return grad.theta[...]


def train_shift(critic_loss, gen_loss, seed, gp=0.0, one_sided=False,
                theta0=6.0, steps=1500, n_critic=5, batch=256, lr_D=1e-2,
                lr_G=1e-2):
    """Fit x' = theta + z to N(0, 1) by plain gradient steps on theta; return
    theta0 and theta after every step, the gradient of every step, and the
    critic."""
    rng = np.random.default_rng(seed)
    G = Shift(theta0)
    D = d2l.make_mlp([1, 32, 32, 1], nnx.relu, rngs=nnx.Rngs(seed))
    opt_G = nnx.Optimizer(G, optax.sgd(lr_G), wrt=nnx.Param)
    opt_D = nnx.Optimizer(D, optax.adam(lr_D, 0.5, 0.9), wrt=nnx.Param)
    n = n_critic if critic_loss else 0
    thetas, grads = [theta0], []
    for _ in range(steps):
        X, Z = (jnp.asarray(rng.standard_normal((n + 1, batch, 1)),
                            jnp.float32) for _ in range(2))
        U = jnp.asarray(rng.uniform(size=(n, batch, 1)), jnp.float32)
        grad = shift_step(critic_loss, gen_loss, gp, one_sided, G, D, opt_G,
                          opt_D, X, Z, U)
        grads.append(grad)                      # device scalars,
        thetas.append(G.theta[...])             # converted once below
    return np.asarray(jnp.stack(thetas)), np.asarray(jnp.stack(grads)), D
```

6 つの設定をそれぞれ 3 つのシードで比較する。飽和型ロジスティック、非飽和型ロジスティック、ヒンジ、WGAN-GP ($\gamma = 10$、両側ペナルティ $(|T'| - 1)^2$)、片側ペナルティ $\max(0, |T'| - 1)^2$ 付き WGAN-GP、および中央値バンド幅 MMD である。各実行について、最初の 100 ステップにおける生成器の勾配の大きさの中央値、θ の軌跡、**到達**ステップ（$|\theta| < 0.5$ に初めて達するステップ）、および到達後の**最遠**逸脱量 $\max |\theta|$ を報告する。

```
w1 = (lambda r, f: f.mean() - r.mean(), lambda r, f: -f.mean())
arms = {'logistic, sat.': (losses['logistic'][0],
                           lambda r, f: -softplus(f).mean(), {}),
        'logistic, NS': (*losses['logistic'], {}),
        'hinge': (*losses['hinge'], {}),
        'W1 + GP': (*w1, dict(gp=10.0)),
        'W1 + 1-sided GP': (*w1, dict(gp=10.0, one_sided=True)),
        'MMD, median': (None, mmd2_median, {})}
steps, w1_runs = (0, 100, 300, 1000, 1500), []  # (arm, seed, theta, critic)
print(f'{"":17s}|grad|   theta at steps {", ".join(map(str, steps))}'
      '  arrival  farthest')
for name, (critic_loss, gen_loss, kwargs) in arms.items():
    for seed in range(3):
        thetas, grads, D = train_shift(critic_loss, gen_loss, seed, **kwargs)
        if name.startswith('W1'):
            w1_runs.append((name, seed, thetas[-1], D))
        hit = np.flatnonzero(np.abs(thetas) < 0.5)  # arrival: hit[0]
        print(f'{name:16s}{seed}{np.median(np.abs(grads[:100])):7.3f} '
              + ''.join(f'{thetas[t]:7.2f}' for t in steps)
              + (f'{hit[0]:9d}{np.abs(thetas[hit[0]:]).max():10.2f}'
                 if len(hit) else '  not reached'))
```

```
                |grad|   theta at steps 0, 100, 300, 1000, 1500  arrival  farthest
logistic, sat.  0  0.006    6.00   5.99   5.96   5.85   5.76  not reached
logistic, sat.  1  0.009    6.00   5.98   5.95   5.85   5.75  not reached
logistic, sat.  2  0.013    6.00   5.98   5.96   5.85   5.77  not reached
logistic, NS    0  2.819    6.00   3.27   0.78   0.03   0.00      381      0.50
logistic, NS    1  2.724    6.00   3.42   0.81   0.02   0.00      387      0.50
logistic, NS    2  2.896    6.00   3.14   0.76   0.02   0.01      378      0.50
hinge           0  1.633    6.00   4.48   3.22   0.00   0.04      748      0.50
hinge           1  1.071    6.00   4.98   3.59   0.06   0.04      938      0.49
hinge           2  1.485    6.00   4.57   3.21   0.01  -0.02      743      0.50
W1 + GP         0  1.254    6.00   5.13   2.74  -1.13  -1.13      508      1.13
W1 + GP         1  1.256    6.00   4.96   2.59  -4.08  -7.30      494      7.47
W1 + GP         2  1.259    6.00   4.74   2.39  -1.22  -1.22      477      1.22
W1 + 1-sided GP 0  1.257    6.00   4.75   2.40  -0.06  -0.06      479      0.49
W1 + 1-sided GP 1  1.262    6.00   4.74   2.39  -0.00   0.01      476      0.50
W1 + 1-sided GP 2  1.265    6.00   4.74   2.39  -0.00  -0.02      476      0.50
MMD, median     0  0.210    6.00   5.79   5.34   3.34   1.32  not reached
MMD, median     1  0.210    6.00   5.79   5.34   3.34   1.32  not reached
MMD, median     2  0.210    6.00   5.79   5.34   3.35   1.32  not reached
```

この結果から、実際のニューラルクリティックが制約なしの理論的最適解とどう異なるかが明確になる。

- **飽和型と非飽和型のロジスティック損失：** 16.1 節と同様、飽和型の生成器勾配はごく小さく（~ 0.01）、θ は 6.0 から 5.76 までしかほとんど動かない。一方、非飽和型の生成器は強い勾配（~ 3）を受け取り、ステップ 400 より前に目標へ到達する。

- **ヒンジクリティック**：理論上の最適ヒンジクリティック sign λ は両方の分布の塊の内部で傾きが 0 になるが、有限の ReLU MLP はギャップ間を傾き 1.1–1.7 で滑らかに補間し、ステップ 700–950 までに生成器を目標へ到達させる。
- **中央値バンド幅 MMD**：飽和型アームを除けば、ステップ 1,500 までに到達しなかった唯一の設定であり、その時点で $\theta \approx 1.3$ である。ここでの生成器は遅いが、停滞しているわけではない。中央値ヒューリスティックによってバンド幅が分離距離と同程度に保たれるため、勾配が消失することはない。ただし勾配は ~ 0.2 と、次項の W_1 クリティックの傾きより約 6 倍小さく、ステップサイズ 0.01 の素の勾配ステップではその分だけ多くのステップを要する。
- **両側と片側の WGAN-GP**：両方の W_1 アームとも、傾き ≈ 1.26（$-ad + 10(a-1)^2$ の最小化により $a = 1 + d/20$）で生成器を内側へ引き寄せ、ステップ 500 付近で到達する。ところが到達後、両側の `W1 + GP` はすべての実行で大きくオーバーシュートする（`farthest` が > 1、最大で > 7）のに対し、片側では到達点より外側へ離れる実行が 1 つもなく（`farthest` が ≤ 0.5）、ステップ 1,000 までにすべて原点から 0.06 以内に収まる。なぜか。θ が 0 を横切ると、クリティックは傾きを負から正へ反転させなければならない。しかし両側ペナルティ $(|T'| - 1)^2$ は平坦なクリティック（$T' = 0$）に γ のペナルティを課すため、傾き 0 にポテンシャル障壁が生じ、クリティックが元の傾きの符号（$1 - d/20$）に閉じ込められるか、すべての ReLU ユニットが死んでしまうからである。これは以下でステップ 1,500 におけるクリティックの傾きと活性 ReLU 数を調べることで確認できる。

```
def slope_and_units(D, x):
    """D'(x) and, per hidden layer, the number of ReLU units active at x."""
    def forward(x):
        h, active = x, []
        for layer in D.layers:
            h = layer(h)
            if not isinstance(layer, nnx.Linear):  # an activation
                active.append((h > 0).sum())
        return h.sum(), active
    slope, active = jax.grad(forward, has_aux=True)(jnp.array([[x]]))
    return float(slope.squeeze()), [int(a) for a in active]

print(f"{'':19s}theta D'(theta)  active    D'(0)  active"
      '  (of 32 units per layer)')
for name, seed, theta, D in w1_runs:
    (s_t, a_t), (s_0, a_0) = (slope_and_units(D, x) for x in (theta, 0.0))
    print(f'{name:16s}{seed}{theta:7.2f}{s_t:10.2f}{a_t[0]:5d}{a_t[1]:3d}'
          f'{s_0:9.2f}{a_0[0]:5d}{a_0[1]:3d}')
```

```
                   theta D'(theta)  active    D'(0)  active  (of 32 units per layer)
W1 + GP          0  -1.13      0.00    6  0     0.00    9  0
W1 + GP          1  -7.30      1.33   16  2     1.44   17  2
W1 + GP          2  -1.22      0.00   26  0     0.00   29  0
W1 + 1-sided GP 0  -0.06      0.00   32  0     0.00   32  0
W1 + 1-sided GP 1   0.01      0.56   30  4     0.56   30  4
W1 + 1-sided GP 2  -0.02      0.47   32  7     0.47   32  7
```

片側ペナルティ $\max(0, |T'| - 1)^2$ は $|T'| \leq 1$ のときにペナルティを課さないため、θ がデータに到達する際にクリティックが傾き 0 を自由に通過できる。16.4 節では、平衡点においてクリティックの傾きを積極的に 0 へ引き寄せるゼロ中心ペナルティ（R_1 と R_2）を解析する。

16.2.4.3 2 次元混合ガウス分布における 5 つの目的関数の比較

次に、生成器とデータのサポートが重なっている状況で 5 つの目的関数を比較する。非飽和型ロジスティック、最小二乗法、ヒンジ、二乗ヒンジ、およびマルチバンド幅 MMD（$s \in \{0.5, 1.0, 2.0\}$）である。生成器 `MixtureGenerator` は 3 成分の混合ガウス分布をパラメータ化し、ずらした平均 `mu_0` とデータより狭いスケール `A_0 = 0.4 * I` で初期化する（$\mathrm{JS}(p, q_0) \approx 0.53$ nat）。

```
mu_0 = np.array([[1.0, -1.0], [-4.3, -2.0], [-1.0, 3.8]])  # off the modes
A_0 = np.stack([0.4 * np.eye(2)] * 3)      # below the data's scale of 0.7

print(f'JS(p, q) at the start: {js_mixtures(mu_0, A_0):.3f} nats')
```

```
JS(p, q) at the start: 0.529 nats
```

初期状態はデータから約 0.5 nat 離れている。MMD アームは、不偏ミニバッチ推定量 MMD^2 をバンド幅 0.5、1、2 の 3 つの RBF カーネルの和のもとで最小化する。これは複数のスケールに感度をもつカーネルである。

```
class MixtureGenerator(nnx.Module):
    """x = mu_k + z A_k for a uniformly drawn component k."""
    def __init__(self, mu, A):
        self.mu = nnx.Param(jnp.asarray(mu, jnp.float32))
        self.A = nnx.Param(jnp.asarray(A, jnp.float32))

    def __call__(self, z, k):
        return jnp.einsum('ni,nij->nj', z, self.A[k]) + self.mu[k]

    def params_np(self):
        return np.asarray(self.mu[...]), np.asarray(self.A[...])

losses['MMD'] = (None, mmd2_loss)   # no critic: MMD^2 to the real batch
```

`train_toy` は 2 つの損失を引数にとり、クリティックの 1 ステップと生成器の 1 ステップを交互に繰り返す。クリティック損失を `None` として格納した MMD アームでは生成器のステップのみを実行し、クリティックのスコアではなく `mmd2_loss(X, fake)` に対して更新する。200 ステップごとに厳密な $\mathrm{JS}(p, q_t)$ を記録する。

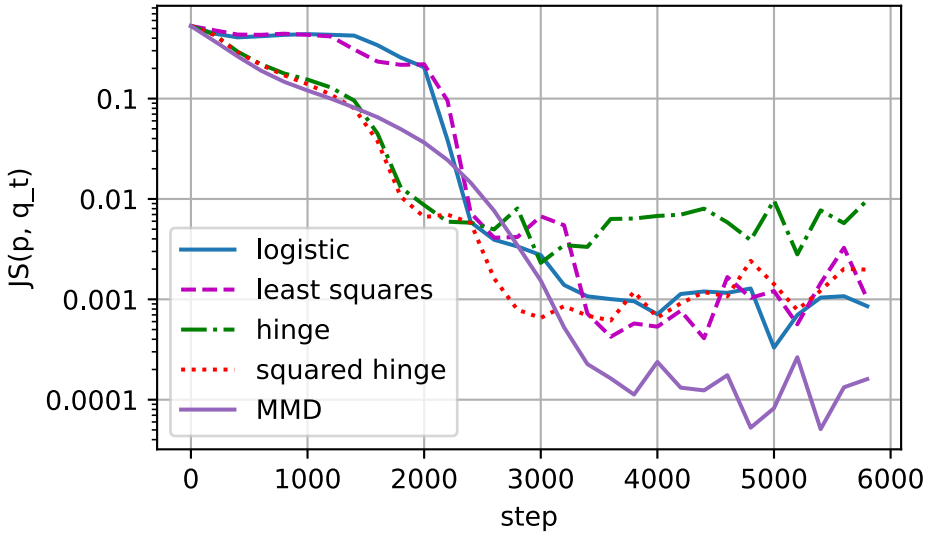
```
@nnx.jit(static_argnums=(0, 1))
def train_toy_step(critic_loss, gen_loss, G, D, opt_G, opt_D, X, z, k):
    """A critic step (unless critic_loss is None), then a generator step."""
    if critic_loss is not None:            # the generator is fixed here
        fake = G(z, k)
        opt_D.update(D, nnx.grad(lambda D_: critic_loss(D_(X), D_(fake)))(D))
    def g_loss(G_):
        fake = G_(z, k)
        return (gen_loss(X, fake) if critic_loss is None
                else gen_loss(D(X), D(fake)))
    opt_G.update(G, nnx.grad(g_loss)(G))


def train_toy(critic_loss, gen_loss, seed=0, steps=6000, batch=256, lr=1e-3):
```

```
    """Fit a MixtureGenerator from (mu_0, A_0) to the data mixture, one
    critic step per generator step. The losses act on the critic's scores of
    the real and generated batch; with critic_loss None, gen_loss acts on the
    two batches. Returns the exact JS(p, q_t) every 200 steps."""
    rng = np.random.default_rng(seed)
    G, D = MixtureGenerator(mu_0, A_0), make_critic(nnx.Rngs(seed))
    opt_G = nnx.Optimizer(G, optax.adam(lr), wrt=nnx.Param)
    opt_D = nnx.Optimizer(D, optax.adam(2 * lr), wrt=nnx.Param)
    trace = []
    for t in range(steps):
        X = jnp.asarray(sample_mixture(batch, mu_p, A_p, rng), jnp.float32)
        z = jnp.asarray(rng.standard_normal((batch, 2)), jnp.float32)
        k = jnp.asarray(rng.integers(0, 3, batch))
        train_toy_step(critic_loss, gen_loss, G, D, opt_G, opt_D, X, z, k)
        if t % 200 == 0:
            trace.append(js_mixtures(*G.params_np()))
    return np.array(trace)
```

各目的関数につき 3 つのシードそれぞれで、最後の 2,000 ステップにわたる $\mathrm{JS}(p, q_t)$ の中央値、ならびに $\mathrm{JS}(p, q_t) < 0.1$ nat および < 0.01 nat に初めて到達するステップを報告する。

```
def first_below(trace, level):
    """First recorded step (one record per 200 steps) with JS below level."""
    hits = np.flatnonzero(trace < level)
    return f'{200 * hits[0]:6d}' if len(hits) else '   n/a'

runs = {name: [train_toy(*pair, seed=seed) for seed in range(3)]
        for name, pair in losses.items()}
print(f'{"three runs:":14s}{"median JS, last 2,000 steps":>27s}'
      f'{"first JS < 0.1":>21s}{"first JS < 0.01":>21s}')
for name, traces in runs.items():
    medians = [np.median(trace[-2000 // 200:]) for trace in traces]
    print(f'{name:14s}   ' + ''.join(f'{m:8.4f}' for m in medians)
          + '   ' + ''.join(first_below(trace, 0.1) for trace in traces)
          + '   ' + ''.join(first_below(trace, 0.01) for trace in traces))
d2l.plot(np.arange(len(runs['logistic'][0])) * 200,
         [traces[0] for traces in runs.values()], xlabel='step',
         ylabel='JS(p, q_t)', yscale='log', legend=list(runs), figsize=(5, 3))
```

```
three runs:   median JS, last 2,000 steps       first JS < 0.1      first JS < 0.01
logistic           0.0011  0.0008  0.0019     2200  2200  1800     2400  2400  2400
least squares      0.0010  0.0007  0.0006     2200  2000  1800     2400  2200  2000
hinge              0.0069  0.0036  0.0064     1400  1400  1800     2000  2200  2000
squared hinge      0.0012  0.0007  0.0009     1400  1400  1200     2000  2000  2400
MMD                0.0001  0.0001  0.0001     1200  1200  1200     2600  2600  2600
```



5 つの目的関数すべてが $\mathrm{JS}(p, q_t)$ を 0.53 nat から 1,000 分の数 nat 以下まで減少させる。変化し続けるクリティックを追跡しない直接的な MMD 最小化が最も精密な最終当てはまり（0.0001 nat）を達成する一方、ヒンジ損失はやや高めの値（0.003–0.007 nat）に落ち着き、滑らかなリンクをもつ目的関数（ロジスティック、最小二乗法、二乗ヒンジはいずれも ≤ 0.0024 nat。演習問題 3）を上回る。続く 16.3 節では、各サンプルを個別に評価するのではなく、実サンプルと生成サンプルをペアで比較するクリティックを検討する。

16.2.5 まとめ

単一サンプルを扱うすべての敵対的目的関数は、クリティックのクラス $\mathcal{T}$ 上での期待利得の上限をとる形をしており（Equation 16.9）、分布 q に関しては凸になる（ただし生成器パラメータ θ に関しては非凸のままである）。クリティックに制約がない場合、対称なマージン分類損失は f ダイバージェンス（それぞれのベイズリスクギャップ $\Delta_\ell(p, q)$ によって定まるもの、Table 16.3）を評価し、最適クリティックは $\lambda = \log(p/q)$ の単調リンクを出力する。また、フェンシェル・ヤング変分下界（Equation 16.13）はこの双対性を非対称な f ダイバージェンスへと拡張する。これに対し、非線形な利得を制約付き関数クラス $\mathcal{F}$ 上の線形な期待値に置き換えると積分確率指標が定義され、その代表例が RKHS の単位球上の最大平均値差異と 1 リプシッツ関数上の 1 次ワッサースタイン距離である。サポートが離れると、f ダイバージェンスは飽和する一方、W_1 は傾き 1 を維持し、MMD^2 はカーネルバンド幅までの距離で有益な勾配を与える。実用上は、有限のニューラルクリティックを非飽和型の生成器損失と組み合わせれば、ギャップ間を滑らかに補間できる。

16.2.6 演習問題

1. [code] **分布空間における凸性とパラメータ空間の対比**。@eq-gan-template の形をしたすべての目的関数は q に関して凸であるが、パラメトリックな生成器は θ を変化させるのであり、q を直接動かすわけではない。`normal_pdf`、`js_1d`、`xs` を用いよ。
    1. $p = \mathcal{N}(0, 1)$、$q_0 = \mathcal{N}(8, 1)$ とおく。二階差分を用いて、$\mathrm{JS}(p, q_t)$ が $t \in [0, 1]$ に関して線形混合経路 $q_t = (1 - t)\, q_0 + t\, p$ に沿って凸であることを確認せよ。
    2. 次に $p = 0.7\,\mathcal{N}(-4, 1) + 0.3\,\mathcal{N}(4, 1)$、$q_\theta = \mathcal{N}(\theta, 1)$ とおく。$\mathrm{JS}(p, q_\theta)$ を $\theta \in [-6, 6]$ の範囲でプロットし、その極小点と極大点を特定せよ。

3. θ に関するこれらの非凸な性質が、q に関する凸性と矛盾しないのはなぜか。

2. **イェンセン・シャノン・ダイバージェンスの凸共役**。イェンセン・シャノン・ダイバージェンスの 2 倍は、ダイバージェンス関数 $f(u) = u\log u - (u+1)\log\frac{u+1}{2}$ に対応する。
   1. その凸共役 $f^*(t) = -\log(2 - e^t)$ と定義域 $t < \log 2$ を導出せよ。
   2. 再パラメータ化 $T(x) = \log(2\sigma(D(x)))$ のもとで、最適クリティック $T^\star = f'(\rho)$ が $D^\star = \lambda$ に対応し、変分下界 Equation 16.13 が $2\log 2 + V(D)$（ただし $V(D)$ は 16.1 節の古典的な対数損失 GAN の目的関数）になることを示せ。

3. [code] **混合ガウス分布における生成器の重みとクリティックの遅れ**。●`train_toy` と 16.2.4.3 節の 5 つの目的関数を用いよ。
   1. ロジスティックのクリティック損失を線形の生成器損失 $-D(x')$（重み $w = 1$）と組み合わせ、非飽和型損失 $-\log\sigma(D(x'))$ と 3 つのシードにわたり、$\mathrm{JS}(p, q_t) < 0.1$ nat に初めて到達するステップ数で比較せよ。サンプルごとの重みを用いてその違いを説明せよ。
   2. ヒンジ損失の最終的な JS がやや高めになる原因がクリティックの追従遅れにあるのかミニバッチノイズにあるのかを検証するため、3 つのシードで 2 つの対照実験を行え。(i) 生成器の 1 回の更新につきクリティックを 5 回更新する設定、(ii) クリティックのバッチサイズを $4\times$ 大きくする設定である。

4. [code] **サンプル複雑度と次元**。●$p = \mathcal{N}(\mathbf{0}, \mathbf{I}_{d_x})$、$q = \mathcal{N}(\mu, \mathbf{I}_{d_x})$（ただし $\|\mu\| = 1$）とおくと、$W_1(p, q) = 1$ が任意の次元 d_x で成り立つ。
   1. W_1 をサイズ n の 2 つの標本から最適線形割当（`scipy.optimize.linear_sum_assignment`）によって $d_x \in \{1, 2, 10, 50\}$ および $n \in \{100, 400, 1600\}$ について推定し、相対誤差を n に対して両対数軸でプロットせよ。また、MMD^2 の不偏推定量（バンド幅 $s = \sqrt{d_x}$ の RBF カーネルを使用）と比較せよ。
   2. 経験 W_1 距離を、サイズ $n = 256$ の 2 つの独立標本（**同**一の分布 p から次元 $d_x = 50$ で抽出したもの）の間で測定せよ。
   3. 順方向 KL 変分下界 Equation 16.13 を `critics['forward KL']` に対して `grid_2d` 上の数値積分で評価し、真の値 $D_{\mathrm{KL}}(p\,\|\,q) \approx 0.16$ nat およびサイズ 512 のミニバッチ推定値と比較せよ。なぜ q からの有限ミニバッチは、$T(x')$ が $1 + \lambda(x')$ を上回る（$q(x')$ が小さい領域において）ときにこの下界を過大評価しうるのか。

5. **分類損失から得られる対称な f ダイバージェンス**。●●すべてのマージン分類損失 Equation 16.10 は対称なベイズリスク $L(\eta) = L(1-\eta)$ をもち、したがって対称な f ダイバージェンスを評価する。逆に（Nguyen et al. (2009)、系 3）、f を $(0, \infty)$ 上の凸関数で $f(1) = 0$、$f(u) = uf(1/u)$、および $f(0) = \lim_{u\to 0} f(u) < \infty$ を満たすものとし、次のように定義する。

$$L(\eta) = 2f(0) - 2(1-\eta)\, f\!\left(\frac{\eta}{1-\eta}\right), \qquad 0 < \eta < 1.$$

   1. $L(\eta)$ が凹であり、$\frac{1}{2}$ に関して対称で、$\eta \to 0$ または 1 のとき 0 に収束し、ベイズリスクギャップ $\Delta_\ell(p, q) = D_f(p\,\|\,q)$ をもつことを示せ。
   2. 微分可能な L と $\hat{\eta} = \sigma(t)$ に対し、実サンプル損失 $-\ell(t) = L(\hat{\eta}) + (1-\hat{\eta})L'(\hat{\eta})$ を定義する。このとき $-\ell(-t) = L(\hat{\eta}) - \hat{\eta}L'(\hat{\eta})$ が成り立つこと、条件付きリスクが $\hat{\eta} = \eta$ で最小化されてその値が $L(\eta)$ になること、および $-\ell(t) \geq 0$ を示せ。
   3. この構成をイェンセン・シャノン・ダイバージェンスに適用するとロジスティック損失が復元されることを確認せよ。

16.3 相対論的目的関数

標準的な分類器型のクリティックは、固定されたゼロのしきい値に対して各サンプルにスコアを付ける（$D(x') > 0$）。しかし多くの場面では、2 つの候補のスコア差を勝率へと変換するブラッドリー・テリー選好モデル（15.3 節）と同様に、絶対的なスコアよりもペアワイズ比較の方が自然である。Jolicoeur-Martineau (2019) は、実サンプル $x \sim p$ と生成サンプル $x' \sim q$ を受け取り、それらのスコア差 $D(x) - D(x')$ からどちらが実データであるかを予測する**相対論的**（relativistic）GAN クリティックを導入した。

16.4 節で構築し、本章の残りの部分でも使用する R3GAN のレシピはこのペアワイズ目的関数に依拠しているため、本節では次の 3 つの問いを解析する。

1. 最適なペアワイズランキングのクリティックは、依然として対数密度比（log density ratio）$\lambda(x) = \log(p(x)/q(x))$ を復元するか。
2. ペアワイズ目的関数はその最適点においてどのダイバージェンスを測り、それは JS(p, q) と比べてどのような関係にあるか。
3. ペアワイズランキングは、どのような条件のもとでモード欠落（mode dropping）の極小値を解消するか。

```
import itertools
```

16.3.1 ペアのスコア付け

16.3.1.1 ブラッドリー・テリーのペアリングクリティック

独立なペア (x, x')（ただし $x \sim p$、$x' \sim q$）が与えられたとき、**相対論的ペアリング**（relativistic pairing）目的関数（RpGAN、Jolicoeur-Martineau (2019); Jolicoeur-Martineau (2020)）は、x が x' よりも本物らしい確率をブラッドリー・テリー尤度 $\sigma(D(x) - D(x'))$ によってモデル化する (Bradley and Terry 1952)。

$$\Phi(D) \;=\; \mathbb{E}_{x\sim p,\, x'\sim q}\big[\log \sigma\big(D(x) - D(x')\big)\big]. \tag{16.17}$$

クリティックは $\Phi(D)$ を最大化する（$-\Phi(D)$ を最小化する）。ミニバッチ上では、`rpgan_loss_D` は $-\log \sigma(D(x_i) - D(x'_i)) = \mathrm{softplus}(D(x'_i) - D(x_i))$ を対応するペア (x_i, x'_i) にわたって平均し、一方の `rpgan_loss_G` は引数を入れ替えることで非飽和生成器損失（non-saturating generator loss）を実装する。

```
def rpgan_loss_D(critic, real, fake):
    """Relativistic pairing critic loss -E[log sigma(D(x) - D(x'))]."""
    return jax.nn.softplus(critic(fake) - critic(real)).mean()


def rpgan_loss_G(critic, real, fake):
    """Non-saturating pairing loss for the generator."""
    return jax.nn.softplus(critic(real) - critic(fake)).mean()
```

16.3.1.2 シフト不変性と最適クリティック

対数損失によってゼロのしきい値が固定される単一サンプル分類器とは異なり、ペアワイズランキング目的関数は差分 $D(x) - D(x')$ のみに依存する（Figure 16.5）。任意の定数 b を D に加え

ても、$\Phi(D+b)=\Phi(D)$ は変わらない。したがって、個々のスコア $D(x')$ の符号は任意であり、自然な正解率の評価指標は代わりにペアワイズランキング正解率 $P(D(x)>D(x'))$ となる。

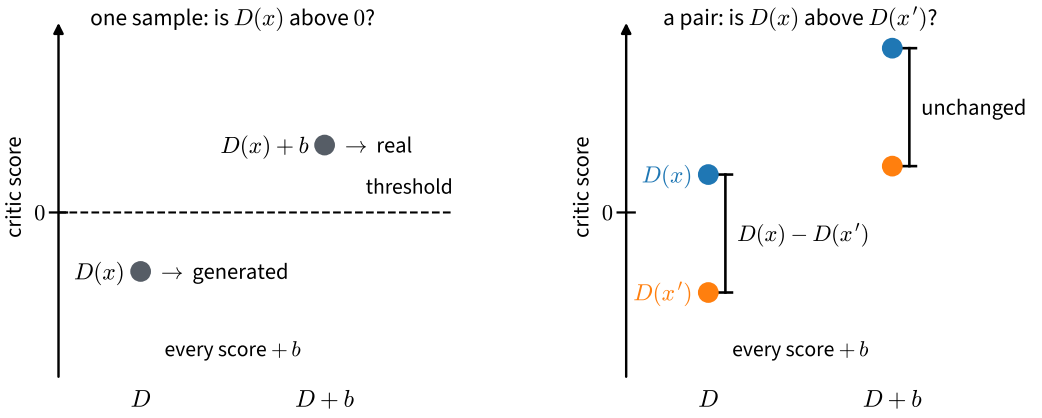


Figure 16.5: 単一サンプルのしきい値判定（左）とペアワイズ差分のスコア付け（右）。すべてのクリティックのスコアを定数 b だけシフトするとゼロに対する単一サンプル分類は変化するが、ペアワイズ差分 $D(x)-D(x')$ は不変に保たれる。

単一サンプルの対数損失目的関数では、$V(D)$ の最大化は各点 x ごとに独立に分解できた。Equation 16.17 ではすべての点が差分 $D(x)-D(x')$ を介して結合しているが、凹性（concavity）により依然として閉形式の最適解が得られる。

命題。 $\Phi(D)$ は D に関して凹関数であり、p と q が同じ台（support）を共有するとき、加法定数の違いを除いて一意に $D^\star(x)=\lambda(x)=\log(p(x)/q(x))$ で最大化される。

証明。 $D\mapsto D(x)-D(x')$ は線形であり、$\log\sigma$ は狭義凹関数であるため、$\Phi(D)$ は D に関して凹、スコア差に関しては狭義凹となる。$\Phi(D)$ の $D(t)$ に関する汎関数微分（ここで t は実サンプル x と生成サンプル x' の両方として現れる）をとり、$\frac{d}{du}\log\sigma(u)=\sigma(-u)$ を用いると次式を得る。

$$\begin{aligned}\frac{\delta\Phi}{\delta D(t)} &= p(t)\,\mathbb{E}_{x'\sim q}\big[\sigma(D(x')-D(t))\big] \\ &\quad - q(t)\,\mathbb{E}_{x\sim p}\big[\sigma(D(t)-D(x))\big].\end{aligned} \tag{16.18}$$

$D=\lambda=\log\rho$（ただし $\rho=p/q$）で評価すると、$\sigma(\lambda(x')-\lambda(t))=\rho(x')/(\rho(x')+\rho(t))$ となる。$q\rho=p$ を用いると、Equation 16.18 の両項はいずれも $p(t)\int\frac{p(x)}{\rho(x)+\rho(t)}\,dx$ に等しくなり、至るところで $\frac{\delta\Phi}{\delta D(t)}=0$ が成り立つ。スコア差に関する $\log\sigma$ の狭義凹性により、$D^\star=\lambda+c$ が定数シフト c の違いを除いて唯一の最大点となる。■

16.3.2 ペアリングダイバージェンス

16.3.2.1 2 状態の例

$\Phi(D)$ の最適値がイェンセン・シャノン・ダイバージェンス（Jensen–Shannon divergence）とどのように関係するかを確かめるため、2 状態空間 $\{0,1\}$ 上で $p=(0.8,0.2)$ および $q=(0.2,0.8)$ を考える。実サンプルを 1 番目に置いた順序対 (a_1,a_2) は積分布（product distribution）

$P(a_1,a_2)=p(a_1)q(a_2)$ に従い、実サンプルを 2 番目に置いたペアは $Q(a_1,a_2)=q(a_1)p(a_2)$ に従う。

ペア (a_1,a_2)	$(0,0)$	$(1,1)$	$(0,1)$	$(1,0)$
$P=p\otimes q$（実サンプルが先）	0.16	0.16	0.64	0.04
$Q=q\otimes p$（実サンプルが後）	0.16	0.16	0.04	0.64

同点ペア $(0,0)$ と $(1,1)$ はどちらの要素が実データであるかについて何の情報も持たないが、混合ペア $(0,1)$ と $(1,0)$ は事後確率（posterior probability）$0.64/(0.64+0.04)=16/17\approx 0.94$ で実データの要素を特定する（単一サンプルを観測した場合の 0.80 との比較）。最適なスコア差 $d=D^\star(0)-D^\star(1)=\lambda(0)-\lambda(1)=\log 16$ において、ランダムな当て推量 ($\Phi(0)=-\log 2$) に対する利得は $\sup_D \Phi(D)+\log 2 = 0.68(\log 2 - H_b(\frac{1}{17}))\approx 0.319$ nat となる。これは $1.66\times$ の倍率で $\mathrm{JS}(p,q)=\log 2-H_b(0.8)\approx 0.193$ nat を上回り、さらに $\mathrm{JS}(P,Q)$（2 つのペア分布 P と Q の間のダイバージェンス）に厳密に一致する。

```
def entropy(v):
    v = v[v > 0]                       # the convention 0 log 0 = 0
    return -(v * np.log(v)).sum()

def js(a, b):
    return entropy((a + b) / 2) - (entropy(a) + entropy(b)) / 2

p, q = np.array([0.8, 0.2]), np.array([0.2, 0.8])
P, Q = np.outer(p, q), np.outer(q, p)  # P[i, j]: real sample at i, generated at j
lam = np.log(p / q)                    # the optimal critic: lam[0] - lam[1] = log 16
U = lam[:, None] - lam[None, :]        # U[i, j] = D(i) - D(j)
phi = (P * -np.logaddexp(0, -U)).sum() # Phi = E_P[log sigma(D(a_1) - D(a_2))]
print(f'JS(p, q) = {js(p, q):.3f}   JS(P, Q) = {js(P.ravel(), Q.ravel()):.3f}   '
      f'sup Phi + log 2 = {phi + np.log(2):.3f}')
```

```
JS(p, q) = 0.193   JS(P, Q) = 0.319   sup Phi + log 2 = 0.319
```

16.3.2.2 積測度への持ち上げ

この等価性は一般に証明できる。そのために、順序対 $(a_1,a_2)\in\mathcal{X}\times\mathcal{X}$ を $P=p\otimes q$（実サンプルが先）または $Q=q\otimes p$（実サンプルが後）のいずれかから抽出された単一の観測とみなし、16.1 節の標準的な対数損失目的関数を $\mathcal{X}\times\mathcal{X}$ 上でペアクリティック $\mathcal{D}(a_1,a_2)$ に対して考える。

$$V_{P,Q}(\mathcal{D}) = \mathbb{E}_{(a_1,a_2)\sim P}\big[\log\sigma(\mathcal{D}(a_1,a_2))\big] + \mathbb{E}_{(a_1,a_2)\sim Q}\big[\log\sigma(-\mathcal{D}(a_1,a_2))\big].$$

補題（積空間への持ち上げ）。 $P=p\otimes q$ および $Q=q\otimes p$ とする。このとき、次が成り立つ。

1. **積空間（product space）上の対数密度比は加法的に分離する。すなわち、**$\log\frac{dP}{dQ}(a_1,a_2)=\lambda(a_1)-\lambda(a_2)$ **である。**
2. **任意の反対称な差分クリティック** $\mathcal{D}(a_1,a_2)=D(a_1)-D(a_2)$ **は** $V_{P,Q}(\mathcal{D})=2\,\Phi(D)$ **を満たす。**

3. **$\mathcal{D}$ を差分クリティック $D(a_1) - D(a_2)$ に制限しても上限（supremum）は減少しない。すなわち、$\sup_{\mathcal{D}} V_{P,Q}(\mathcal{D}) = \sup_{D} V_{P,Q}(D(a_1) - D(a_2))$ が成り立つ。**

証明。（1）は $\frac{dP}{dQ}(a_1, a_2) = \frac{p(a_1)q(a_2)}{q(a_1)p(a_2)} = \frac{\rho(a_1)}{\rho(a_2)}$ から直ちに従う。（2）については、Q のもとで $(a_1, a_2) \sim q \otimes p$ かつ $-\mathcal{D}(a_1, a_2) = D(a_2) - D(a_1)$ となるため、$V_{P,Q}(\mathcal{D})$ に含まれる 2 つの期待値はいずれも $\Phi(D)$ に等しい。（3）については、$V_{P,Q}(\mathcal{D})$ の制約なし最大点は $\mathcal{D}^\star(a_1, a_2) = \log \frac{dP}{dQ}(a_1, a_2)$ であり、これは（1）よりすでに差分の形 $\lambda(a_1) - \lambda(a_2)$ をしている。■

（なお、性質 3 はロジスティック利得が持つ対数リンク関数に固有の性質である。最小二乗法などの他の利得では、最適なペアクリティックは $\lambda(a_1) - \lambda(a_2)$ の非線形関数となり、加法的に分離可能ではなくなる。演習問題 4 を参照。）

定理。較正されたペアリングダイバージェンス $d_{\mathrm{Rp}}(p, q) := \sup_D \Phi(D) + \log 2$ は次を満たす。

$$\begin{aligned} d_{\mathrm{Rp}}(p, q) &= \mathrm{JS}(p \otimes q,\ q \otimes p) \\ &= H\Big(\tfrac{1}{2}(p \otimes q + q \otimes p)\Big) - H(p) - H(q). \end{aligned} \tag{16.19}$$

証明。持ち上げ補題の 2 と 3 を、イェンセン・シャノンのミニマックス恒等式 $\sup_{\mathcal{D}} V_{P,Q}(\mathcal{D}) = 2\,\mathrm{JS}(P, Q) - 2\log 2$（16.1 節）と組み合わせると、$2\sup_D \Phi(D) = \sup_{\mathcal{D}} V_{P,Q}(\mathcal{D}) = 2\,\mathrm{JS}(P, Q) - 2\log 2$ が得られ、第 1 の等号が示される。さらにエントロピーギャップの恒等式 $\mathrm{JS}(P, Q) = H(\frac{1}{2}(P + Q)) - \frac{1}{2}(H(P) + H(Q))$（16.1 節）を適用し、$H(p \otimes q) = H(q \otimes p) = H(p) + H(q)$ を用いれば、第 2 の等号が得られる。■

$D^\star = \lambda$ を Equation 16.17 に直接代入すると、明示的な期待値表示も得られる。

$$d_{\mathrm{Rp}}(p, q) = \log 2 - \mathbb{E}_{x \sim p,\, x' \sim q}\Big[\log\Big(1 + \frac{q(x)\,p(x')}{p(x)\,q(x')}\Big)\Big], \tag{16.20}$$

また、ペア空間上の相互情報量（mutual information）の恒等式 $\mathrm{JS}(P, Q) = I((a_1, a_2); y)$（16.1 節）から、ランダムに順序付けられたペア (a_1, a_2) とその順序ラベル $y \in \{0, 1\}$ に対して次式が成り立つ。

$$d_{\mathrm{Rp}}(p, q) = I((a_1, a_2);\, y) = \log 2 - H(y \mid a_1, a_2). \tag{16.21}$$

16.3.2.3 情報量境界

Equation 16.21 を用いると、$d_{\mathrm{Rp}}(p, q)$ を $\mathrm{JS}(p, q)$ と直接結びつけることができる。ランダムに順序付けられたペアの各周辺要素 a_k は分布 $m = (p + q)/2$ に従い、$I(a_k; y) = \mathrm{JS}(p, q)$ nat の情報量を順序ラベル y について持つ。一方、a_1 と a_2 は y が与えられたもとで条件付き独立である（$H(a_2 \mid a_1, y) = H(a_2 \mid y)$）。相互情報量の連鎖律（chain rule）より、次式を得る。

$$\begin{aligned} d_{\mathrm{Rp}}(p, q) &= I(a_1; y) + I(a_2; y \mid a_1) \\ &= I(a_1; y) + I(a_2; y) - I(a_1; a_2) \\ &= 2\,\mathrm{JS}(p, q) - I(a_1; a_2). \end{aligned}$$

$I(a_2; y \mid a_1) \geq 0$、$I(a_1; a_2) \geq 0$、および $d_{\mathrm{Rp}}(p, q) \leq \log 2$ より、タイトな上下界（挟み撃ちの評価式）が得られる。

$$\mathrm{JS}(p, q) \leq d_{\mathrm{Rp}}(p, q) \leq \min\big(2\,\mathrm{JS}(p, q),\, \log 2\big).$$

均衡点の近傍（$q \to p$）では、$I(a_1; a_2)$ は $p - q$ に関して 4 次のオーダーであるのに対し、$\mathrm{JS}(p, q)$ は 2 次のオーダーであるため、$d_{\mathrm{Rp}}(p, q) \approx 2\,\mathrm{JS}(p, q)$ となる（演習問題 2）。また、p と q が互いに素な台を持つとき、$\mathrm{JS}(p, q)$ と $d_{\mathrm{Rp}}(p, q)$ はどちらも $\log 2$ で飽和する。

16.3.3 ランキングとモード被覆

16.3.3.1 相対論的ランク重み

$\Phi(D)$ を生成器パラメータ θ に関して $x' = G(z)$ を介して微分すると、次式が得られる。

$$\begin{aligned} \nabla_\theta \Phi &= -\,\mathbb{E}_z\big[w_{\mathrm{Rp}}(G(z))\, \nabla_\theta D(G(z))\big], \\ w_{\mathrm{Rp}}(x') &= \mathbb{E}_{x \sim p}\big[\sigma\big(D(x') - D(x)\big)\big]. \end{aligned} \tag{16.22}$$

飽和型のゼロサム定式化（Φ の最小化）では、各生成サンプルは $\nabla_\theta D(G(z))$ に沿って重み $w_{\mathrm{Rp}}(x')$ で押し出される。この重みは、x' がランダムに抽出された実サンプル $x \sim p$ を上回るソフト化された確率である。非飽和型の定式化（`rpgan_loss_G`、すなわち $-\mathbb{E}[\log \sigma(D(x') - D(x))]$ の最小化）では、重みは $1 - w_{\mathrm{Rp}}(x') = \mathbb{E}_{x \sim p}[\sigma(D(x) - D(x'))]$ となる。単一サンプル重み $\sigma(\pm D(x'))$（$D(x')$ をゼロとのみ比較する）とは異なり、相対論的重みは $D(x')$ を実スコア $D(x)$ の分布と比較するため、D の定数シフトに対して不変である。

16.3.3.2 固定ペアリングと全ペアのランドスケープ

ペアワイズランキングはモード欠落の極小値を解消するだろうか。Sun et al. (2020) は、n 個の相異なる実データ点 $x_1, \ldots, x_n$ と n 個の自由に移動できる生成点 $x'_1, \ldots, x'_n$ を、厳密な最適応答（best-response）クリティック D のもとで評価する離散設定を解析した。

$$\begin{aligned} \mathcal{E}_{\mathrm{GAN}} &= \sup_D \tfrac{1}{2n} \textstyle\sum_i \big[\log \sigma(D(x_i)) + \log \sigma(-D(x'_i))\big], \\ \mathcal{E}_{\mathrm{diag}} &= \sup_D \tfrac{1}{n} \textstyle\sum_i \log \sigma\big(D(x_i) - D(x'_i)\big), \\ \mathcal{E}_{\mathrm{all}} &= \sup_D \tfrac{1}{n^2} \textstyle\sum_{i,j} \log \sigma\big(D(x_i) - D(x'_j)\big). \end{aligned} \tag{16.23}$$

ここで、$\mathcal{E}_{\mathrm{GAN}}$ は古典的な単一サンプル目的関数、$\mathcal{E}_{\mathrm{diag}}$ は**固定ペアリング**（fixed-pairing）目的関数（実データ点 x_i が生成点 x'_i と恒久的にペアを組む設定）であり、$\mathcal{E}_{\mathrm{all}}$ は実データと生成点の全 n^2 ペアにわたって平均をとる。すべての x'_i が実データ点上に位置するものの、少なくとも 2 点が一致する（結果として被覆されない実データ点が残る）配置は**衝突**（collision）と呼ばれる。Sun et al. (2020) は、すべての衝突（全部で $n^n - n!$ 個）が $\mathcal{E}_{\mathrm{GAN}}$ の狭義の極小値となる一方、$\mathcal{E}_{\mathrm{diag}}$ にはあらゆる衝突から大域的最小値 $-\log 2$ へと至る非増加な脱出経路が存在することを証明した。

この事実は $n = 2$ と $n = 3$ について直接検証できる。経験分布（empirical distribution）を $\hat{p}$ と $\hat{q}$ とすると、$\mathcal{E}_{\mathrm{GAN}} = \mathrm{JS}(\hat{p}, \hat{q}) - \log 2$ および $\mathcal{E}_{\mathrm{all}} = \mathrm{JS}(\hat{p} \otimes \hat{q}, \hat{q} \otimes \hat{p}) - \log 2$ となる一方、$\mathcal{E}_{\mathrm{diag}}$ は $-\log 2$ に閉路（cycle）上の有向辺 $i \to \mathrm{gen}[i]$ の割合を掛けた値に等しい（演習問題 5）。

```
    def on_cycle(i, gen):
        """Does the arrow from real point i to its partner's position close a cycle?"""
        j = gen[i]
        for _ in range(len(gen)):
            if j == i:
                return True
            if j not in range(len(gen)):  # a position that holds no data point
                return False
            j = gen[j]                    # follow the arrow of the real point at j
        return False

    def losses(gen):
        """Best-response classical, all-pairs and paired losses of a configuration."""
        n = len(gen)
        support = sorted(set(range(n)) | set(gen))
        p_hat = np.isin(support, range(n)) / n
        q_hat = np.array([gen.count(s) for s in support]) / n
        P, Q = np.outer(p_hat, q_hat).ravel(), np.outer(q_hat, p_hat).ravel()
        paired = -np.log(2) * np.mean([on_cycle(i, gen) for i in range(n)])
        return np.array([js(p_hat, q_hat) - np.log(2), js(P, Q) - np.log(2), paired])

    print('gen         classical, all pairs, paired: nats, multiples of log 2')
    for gen in ([0, 0], [0, 0.1]):   # both on x_1; then x'_2 steps off it
        value = losses(gen)
        print(f'{str(gen):12s}{np.round(value, 3)}  {np.round(value / np.log(2), 2)}')
```

```
gen         classical, all pairs, paired: nats, multiples of log 2
[0, 0]      [-0.477 -0.347 -0.347]  [-0.69 -0.5  -0.5 ]
[0, 0.1]    [-0.347 -0.173 -0.347]  [-0.5  -0.25 -0.5 ]
```

x_2' を 0 から 0.1 へ動かすと、古典的損失と全ペア損失は増加する（衝突 `[0, 0]` が両者にとって狭義の極小値であることが確認できる）一方、固定ペアリング損失は $-\frac{1}{2}\log 2$ のまま変化しない。全 n^n 通りの配置を体系的に調べるため、`count_bad_minima(n)` はすべての割り当てを `small_moves(gen)` による局所摂動に対してテストする。

```
    def small_moves(gen):
        """Each generated point stays or steps to one of n fresh positions next to it."""
        n = len(gen)
        for steps in itertools.product(range(n + 1), repeat=n):
            if any(steps):
                yield [g + k / 10 if k else g for g, k in zip(gen, steps)]

    def count_bad_minima(n):
        """Suboptimal strict local minima among the assignments, per loss."""
        best, count = losses(list(range(n))), np.zeros(3, dtype=int)
        for gen in itertools.product(range(n), repeat=n):
            value = losses(list(gen))
            rises = np.all([losses(m) > value + 1e-9 for m in small_moves(list(gen))],
                           axis=0)
            count += rises & (value > best + 1e-9)
        return count
```

`count_bad_minima` を $n \in \{2, 3\}$ に対して実行し、各目的関数における準最適な狭義の極小値の

個数を理論上の衝突数 $n^n - n!$ と比較する。

```
for n in (2, 3):
    c = count_bad_minima(n)
    print(f'n = {n}: classical {c[0]}, all pairs {c[1]}, paired {c[2]};  '
          f'n^n - n! = {n**n - math.factorial(n)}')
```

```
n = 2: classical 2, all pairs 2, paired 0;  n^n - n! = 2
n = 3: classical 21, all pairs 21, paired 0;  n^n - n! = 21
```

重要な点として、この全列挙から、固定結合損失 $\mathcal{E}_{\text{diag}}$ では衝突による極小値が 0 個であるのに対し、全ペア損失 $\mathcal{E}_{\text{all}}$ は $n^n - n!$ 個すべての衝突を極小値として持つ（$\mathcal{E}_{\text{GAN}}$ とまったく同様である）ことがわかる。標準的なミニバッチ訓練ではステップごとに実データと生成データのミニバッチを独立に再サンプリングするため、ランダムなペアリングにわたる目的関数の期待値は $\mathcal{E}_{\text{all}}$ となり、$\mathcal{E}_{\text{diag}}$ にはならない（演習問題 5）。したがって、Sun et al. (2020) のランドスケープ定理は固定された 1 対 1 結合にのみ適用され、ミニバッチ RpGAN によるモード被覆（mode coverage）の改善効果は実験的に検証する必要がある（16.4 節）。

16.3.3.3 訓練における飽和型と非飽和型のペアリング

飽和型の生成器損失（Φ）と非飽和型の生成器損失（`rpgan_loss_G`）は、どちらも大域的最小値 $q = p$ を共有する。p と q がすでに重なり合っている場合（たとえば 16.2.4.3 節の 3 モード混合ガウス分布ベンチマーク）、両方の変種とも数千分の 1 nat 以内に収束する。

```
scores = lambda s: s   # train_toy passes scores, so the critic is the identity
arms = {'non-saturating': (lambda r, f: d2l.rpgan_loss_D(scores, r, f),
                           lambda r, f: d2l.rpgan_loss_G(scores, r, f)),
        'saturating': (lambda r, f: d2l.rpgan_loss_D(scores, r, f),
                       lambda r, f: -d2l.rpgan_loss_D(scores, r, f))}

def first_step_below(trace, level=0.01, every=200):
    below = np.flatnonzero(trace < level)
    return f'{every * below[0]:5d}' if len(below) else ' none'

print('pairing critic; generator loss, runs 0, 1, 2')
for name, (critic_loss, gen_loss) in arms.items():
    traces = [d2l.train_toy(critic_loss, gen_loss, seed) for seed in range(3)]
    medians = ' '.join(f'{np.median(t[-2000 // 200:]):.4f}' for t in traces)
    firsts = ' '.join(first_step_below(t) for t in traces)
    print(f'{name:15s} median JS {medians}   first JS < 0.01 {firsts}')
```

```
pairing critic; generator loss, runs 0, 1, 2
non-saturating  median JS 0.0008 0.0009 0.0006   first JS < 0.01  2200  2400  2200
saturating      median JS 0.0018 0.0024 0.0010   first JS < 0.01  1800  1800  1800
```

どちらのアームも数千分の 1 nat 以下の精度で適合する。非飽和型の実行はステップ 2,200〜2,600 の間で 0.01 nat を下回る。これは 16.2 節のロジスティック損失（logistic loss）と同様であり、ペアリングによって適合精度が損なわれることはない。さらに飽和型は、データの近傍で重みが小さくならないため、すべての実行でステップ 1,800 において先にこの水準を下回る。

しかし、生成器がデータから遠く離れた位置から出発する場合（$\theta_0 = 6$ とした `d2l.train_shift`、16.2.4.2 節）、すべての生成サンプルがすべての実サンプルよりはるかに低くランク付けされ

($D(x') \ll D(x)$)、飽和型のランク重み $w_{\text{Rp}}(x')$ はゼロへと押し下げられる。

```
steps = (0, 100, 300, 1000, 1500)
print(f'{"":20s}|gradient|   theta at steps {steps}   first |theta| < 0.5')
for name, (critic_loss, gen_loss) in arms.items():
    for seed in range(3):
        thetas, grads, _ = d2l.train_shift(critic_loss, gen_loss, seed)
        hit = np.flatnonzero(np.abs(thetas) < 0.5)
        print(f'{name:15s} {seed}   {np.median(np.abs(grads[:100])):8.3f}   '
              + ''.join(f'{thetas[t]:7.2f}' for t in steps) + '   '
              + (f'{hit[0]:5d}' if len(hit) else 'not reached'))
```

```
                    |gradient|   theta at steps (0, 100, 300, 1000, 1500)   first |theta| < 0.5
non-saturating  0      3.186      6.00   2.43   0.58   0.02   0.00     324
non-saturating  1      3.058      6.00   2.24   0.54   0.02   0.00     314
non-saturating  2      3.006      6.00   2.56   0.59   0.02   0.00     331
saturating      0      0.000      6.00   6.00   6.00   6.00   6.00   not reached
saturating      1      0.000      6.00   6.00   6.00   6.00   5.99   not reached
saturating      2      0.000      6.00   6.00   6.00   6.00   5.99   not reached
```

$\theta_0 = 6$ から開始すると、飽和型ペアリング生成器の勾配の中央値は 5×10^{-4} 未満となり、1,500 ステップ後も $\theta \approx 6.00$ に停滞したままである。これに対し、非飽和型ペアリング生成器（`rpgan_loss_G`）は ~ 3 の勾配を受け取り、すべてのシードでステップ 330 前後には $|\theta| < 0.5$ に到達する。したがって、本章の残りの部分では一貫して非飽和型ペアリング損失を使用する。

16.3.4 まとめ

相対論的ペアリング目的関数（RpGAN）は、実サンプルを生成サンプルより上位にランク付けするブラッドリー・テリー対数尤度 $\Phi(D) = \mathbb{E}_{x\sim p, x'\sim q}[\log \sigma(D(x) - D(x'))]$ を最大化し、加法定数の違いを除いて対数密度比 $D^\star(x) = \lambda(x) = \log(p(x)/q(x))$ を復元する。(a_1, a_2) を積空間 $\mathcal{X} \times \mathcal{X}$ へ持ち上げることで、較正された最適値 $d_{\text{Rp}}(p, q) = \sup_D \Phi(D) + \log 2$ が $\text{JS}(p \otimes q, q \otimes p) = 2\,\text{JS}(p, q) - I(a_1; a_2)$ に等しくなり、$\text{JS}(p, q) \le d_{\text{Rp}}(p, q) \le \min(2\,\text{JS}(p, q), \log 2)$ を満たすことが示される。恒久的な 1 対 1 結合 $\mathcal{E}_{\text{diag}}$ は最適応答クリティックのもとで離散的なモード欠落の衝突極小値を解消するものの、独立なミニバッチの再サンプリングは平均化によって全ペア目的関数 $\mathcal{E}_{\text{all}}$ へと戻り、それらの極小値を残してしまう。最後に、生成サンプルがデータから遠く離れているとき、飽和型のランク重み $w_{\text{Rp}}(x') = \mathbb{E}_{x\sim p}[\sigma(D(x') - D(x))]$ は消失するが、非飽和型の重み $1 - w_{\text{Rp}}(x')$ は 1 付近にとどまり、生成器の勾配を維持する。

16.3.5 演習問題

1. [code] **飽和型と非飽和型のランク重み**。データ分布 $p = (0.35, 0.25, 0.20, 0.12, 0.08)$ と生成器分布 $q = (0.02, 0.05, 0.13, 0.30, 0.50)$ を持つ 5 状態の離散空間を考える。
   1. 2 状態のセルで定義した `js` を用いて $\text{JS}(p, q)$ と $d_{\text{Rp}}(p, q)$ を計算し、25 個のペア状態上の $\text{JS}(P, Q)$ が $\sup_D \Phi(D) + \log 2$（$D = \lambda$ における値）と一致することを検証せよ（それぞれ ≈ 0.234 および 0.377）。
   2. 飽和型の重み w_{Rp}（Equation 16.22）と非飽和型の重み $1 - w_{\text{Rp}}$ を $D = \lambda$ のもとで各状態について評価し、それらの q に関する期待値を比較せよ。
   3. $D = \lambda$ において、$\mathbb{E}_{x'\sim q}[w_{\text{Rp}}(x')] = \iint \frac{PQ}{P+Q}\, da_1\, da_2 \le \frac{1}{2}$ が成り立ち、等号成立は $q = p$ のとき、かつそのときに限ること、さらに p と q の台が離れるにつれてこの値が消失することを示せ。

4. `d2l.train_shift` をペアリングクリティックと両方の生成器損失に対して初期オフセット `theta0` $\in \{1, 2, 3, 4\}$（各 3 シード）から実行し、飽和型の生成器が停滞し始める距離を求めよ。

2. **d_{Rp} と $2\,\text{JS}(p, q)$ の差。**●ランダムに順序付けられたペア (a_1, a_2) とその順序ラベル y（Equation 16.21）を考える。
   1. a_1 と a_2 が y のもとで条件付き独立であること、および $I(a_k; y) = \text{JS}(p, q)$ が $k \in \{1, 2\}$ に対して成り立つことを示せ。
   2. p と q が互いに素な台を持つとき、$I(a_1; a_2)$ をペアの混合分布 $\frac{1}{2}(P + Q)$ から直接計算し、$I(a_1; a_2) = \text{JS}(p, q) = \log 2$ を検証せよ。
   3. 局所摂動 $q = p\,(1 + \epsilon v)$（ただし $\mathbb{E}_p[v] = 0$）に対して、$\text{JS}(p, q) = \Theta(\epsilon^2)$ である一方 $I(a_1; a_2) = \Theta(\epsilon^4)$ となり、したがって $d_{\text{Rp}}(p, q)/\text{JS}(p, q) \to 2$（$\epsilon \to 0$ のとき）が成り立つことを示せ。
   4. $d_{\text{Rp}}(p, q) = 2\,\text{JS}(p, q) - I(a_1; a_2)$ を、5 状態の分布（演習問題 1）上で数値的に検証せよ。

3. **2 つのガウス分布。**$p = \mathcal{N}(0, I_d)$ および $q = \mathcal{N}(\mu, I_d)$ を $\mathbb{R}^d$ 上の分布とする。
   1. $\lambda(x)$ が x に関してアフィンであり、$u = \lambda(x') - \lambda(x)$（Equation 16.20）がガウス分布に従うことを示し、その平均と分散を求めよ。
   2. $\text{JS}(p, q)$ を同じ形で表し、スケーリング定数 c として $\text{JS}(p, \mathcal{N}(c\mu, I_d)) = d_{\text{Rp}}(p, q)$ がすべての μ に対して成り立つものを求めよ。
   3. $d_{\text{Rp}}(p, q)$ と $\text{JS}(p, q)$ を 1D 数値積分によって $\|\mu\| \in \{0.1, 1, 2, 4\}$ に対して評価し、その比を局所極限および大分離極限と比較せよ。

4. **ペアクリティックの分離可能性。**●$\log \sigma$（Equation 16.17）を一般の凹利得関数 ℓ に置き換え、$\Phi_\ell(D) = \mathbb{E}_{x \sim p, x' \sim q}[\ell(D(x) - D(x'))]$ とする。
   1. 最小二乗法の利得 $\ell(t) = -(1-t)^2$ に対し、$\mathcal{X} \times \mathcal{X}$ 上の制約なし最適ペアクリティックが $2\eta - 1 = \tanh\left(\frac{\lambda(a_1) - \lambda(a_2)}{2}\right)$（ただし $\eta = P/(P+Q)$）となり、$D(a_1) - D(a_2)$ のように加法的に分離可能ではないことを示せ。
   2. 5 状態の例（演習問題 1）において、制約なしのペア最適値と $\sup_{D \in \mathbb{R}^5} 2\,\Phi_\ell(D)$ とのギャップを計算せよ。
   3. 不均衡な順序事前確率 $\alpha \neq \frac{1}{2}$ を持つロジスティック利得のもとで、ベイズ最適なペアクリティックを導出し、それを達成するには差分クラス $D(a_1) - D(a_2)$ をどのように拡張すべきかを示せ。

5. [code] **固定結合とランダムな再ペアリング。**●●
   1. `losses` で $\mathcal{E}_{\text{diag}}$ に対して用いた閉路則を証明せよ。すなわち、閉路に含まれない任意の有向辺上では、クリティックは $D(x_i) - D(x'_i) \to +\infty$ とすることでその損失をゼロにできる一方、長さ k の閉路に沿って差分の和をとるとゼロになり、$\log \sigma$ の凹性により等スコアのときに最大値 $-k \log 2$ をとることを示せ。
   2. `count_bad_minima` を $n = 4$（256 通りの配置）に拡張し、$\mathcal{E}_{\text{GAN}}$、$\mathcal{E}_{\text{all}}$、および $\mathcal{E}_{\text{diag}}$ における個数を検証せよ。
   3. 置換平均した最適応答目的関数 $\frac{1}{n!}\sum_\pi \sup_D \frac{1}{n}\sum_i \log \sigma(D(x_i) - D(x'_{\pi(i)}))$ を定義し、$n = 3$ に対する準最適な狭義の極小値の個数を数えよ。このレジームを、ランダムにペアリングされた多数のミニバッチにわたって緩やかに適応するクリティック（$\mathcal{E}_{\text{all}}$ に直面する）と比較せよ。

16.4 勾配ペナルティと収束

16.1 節の Fashion-MNIST 実験では、識別器損失も生成器損失も理論上の平衡値には落ち着かなかった。交互勾配降下法は収束が遅いだけなのか、それとも平衡点に到達することなくその周囲

を際限なく周回しうるのだろうか。16.1 節から 16.3 節までの母集団ダイバージェンスの解析では、クリティックが厳密な最適解 $D^\star$ にあるときのミニマックス目的関数を特徴づけた。しかし実際には、両方のネットワークが同時または交互に小さな勾配ステップを踏み、一方の更新がもう一方の目的関数を絶えず変化させる。

勾配訓練のダイナミクスを直接解析するため、Mescheder et al. (2018) は点質量のデータ分布、点質量の生成器、および線形クリティックからなる最小限の 2 パラメータ問題である **Dirac-GAN** を導入した。このような 2 パラメータ系においてさえ、正則化なしの勾配降下法は収束しない。一方で、クリティックの入力勾配の二乗ノルムにペナルティ（R_1 と R_2）を課すと、平衡点は局所漸近安定（locally asymptotically stable）になる。これらのゼロ中心勾配ペナルティ（zero-centered gradient penalty）を 16.3 節の相対論的ペアリング目的関数と組み合わせることで、本章の残りの節で用いる R3GAN レシピが得られる (Huang et al. 2024)。

16.4.1 Dirac-GAN

16.4.1.1 2 パラメータのミニマックス系

データ分布を原点における点質量 $p = \delta_0$ とし、生成器は $\theta \in \mathbb{R}$ に点質量を置く分布 $q_\theta = \delta_\theta$ とする（したがって $q_\theta = p$ となるのは $\theta = 0$ のとき、かつそのときに限る）。また、クリティックをバイアスなしの線形関数 $D_\psi(x) = \psi x$ とする。対称マージン目的関数 $V_\ell(D) = \mathbb{E}_{x\sim p}[\ell(D(x))] + \mathbb{E}_{x'\sim q}[\ell(-D(x'))]$（16.2 節）のもとでは、両方の期待値は単一の評価に帰着する。

$$\begin{aligned} V(\theta,\psi) &= \ell(D_\psi(0)) + \ell(-D_\psi(\theta)) \\ &= \ell(0) + \ell(-\psi\theta). \end{aligned} \tag{16.24}$$

ここではロジスティック利得 $\ell(t) = \log\sigma(t)$ を用いる。この関数は $\ell'(t) = \sigma(-t) > 0$ および $\ell''(t) = -\sigma(t)\sigma(-t) < 0$ を満たし、$\ell'(0) = \tfrac{1}{2}$、$\ell''(0) = -\tfrac{1}{4}$ である。

任意の $\theta \neq 0$ に対して、p と q_θ の台（support）は互いに素である。そのため、$V(\theta,\psi)$ を ψ について最大化すると $\psi\theta \to -\infty$ となり、定数 $\sup_\psi V(\theta,\psi) = \ell(0)$ がすべての $\theta \neq 0$ に対して得られる結果、θ に関する勾配はゼロになる。

16.4.1.2 連続時間における非減衰振動

勾配訓練では、$\sup_\psi V(\theta,\psi)$ を直接解く代わりに、現在の値から θ と ψ を少しずつ更新する。Equation 16.24 の唯一の停留点（stationary point）は平衡点 $(\theta,\psi) = (0,0)$ である。連続時間の同時勾配フロー（$\dot\theta = -\partial_\theta V$、$\dot\psi = +\partial_\psi V$）のもとでは、次式が成り立つ。

$$\begin{aligned} \dot\theta &= -\partial_\theta V = \psi\,\ell'(-\psi\theta), \\ \dot\psi &= +\partial_\psi V = -\theta\,\ell'(-\psi\theta). \end{aligned} \tag{16.25}$$

速度ベクトル $\mathbf{u}(\theta,\psi) = \ell'(-\psi\theta)\,(\psi,-\theta)$ は至るところで状態ベクトル (θ,ψ) に直交するため、原点までの二乗距離は保存される。

$$\frac{d}{dt}(\theta^2+\psi^2) = 2\theta\psi\,\ell'(-\psi\theta) - 2\psi\theta\,\ell'(-\psi\theta) = 0.$$

あらゆる軌跡は、$(0,0)$ を中心とする一定半径の閉じた円を描く（これは任意の狭義単調増加な利得 ℓ に対して成り立つ）。すなわち、生成器のパラメータ θ は決して収束することなく、0 をまたいで前後に振動し続ける。

Equation 16.25 を $(\theta,\psi)=(0,0)$ の周りで線形化すると（38.1.2.5 節）、ヤコビ行列と固有値は次のようになる。

$$J=\begin{pmatrix} 0 & \ell'(0) \\ -\ell'(0) & 0 \end{pmatrix}, \qquad \nu_{1,2}=\pm\, i\,\ell'(0). \tag{16.26}$$

純虚数の固有値は線形の**中心点**（center）に対応する（双線形鞍点問題 $\min_\theta \max_\psi \theta\psi$、Daskalakis et al. (2018)）。なお、スカラー因子 $\ell'(-\psi\theta)=\sigma(D_\psi(\theta))=w_{\text{sat}}$ は両成分に同時に掛かるため、円周上の角速度をリスケールするにすぎない。

16.4.1.3 離散時間における外向きスパイラルとリミットサイクル

離散時間では、ステップサイズ $h>0$ の同時勾配降下法は $(\theta,\psi)\mapsto(\theta,\psi)+h\,\mathbf{u}(\theta,\psi)$ と更新する。$\mathbf{u}(\theta,\psi)\perp(\theta,\psi)$ であるため、ピタゴラスの定理より次式が得られる。

$$\big\|(\theta,\psi)+h\,\mathbf{u}\big\|^2=\big\|(\theta,\psi)\big\|^2+h^2\big\|\mathbf{u}\big\|^2.$$

したがって、任意の $h>0$ に対して、離散時間の同時ステップは毎回、平衡点からの距離を厳密に**増加させ**、反復点は外向きに渦を巻いて発散する（演習問題 1）。

交互勾配更新（まず ψ を更新し、続いて θ を更新後の ψ を用いて更新する手法）は、これよりわずかに良い挙動を示す。$(0,0)$ の近傍では、各半ステップが面積保存の線形せん断となるため、合成されたヤコビ行列の行列式は 1 である。$h\,\ell'(0)\le 2$ のとき、両方の固有値は単位円上に乗るため (Mescheder et al. 2018)、交互更新の反復点は収束することなく原点の周りの有界な帯状領域内を周回する（演習問題 4）。非飽和生成器損失（原点で同じ線形化 J を共有する）も、相対論的ペアリング損失（$\ell(D_\psi(0)-D_\psi(\theta))=\ell(-\psi\theta)$ となり同一の勾配場を持つ）も、これらの局所固有値を変えない。

16.4.2 ゼロ中心勾配ペナルティ

平衡点 $(\theta,\psi)=(0,0)$ を局所漸近安定にするには、ヤコビ行列の固有値を虚軸上から開左半平面 $\mathrm{Re}(\nu)<0$ へと移動させなければならない。平衡時のクリティックは平坦でなければならないため（$\nabla_x D^\star(x)=0$ が $q=p$ のときに成り立つ）、クリティックの入力勾配の二乗ノルムにペナルティを課せば、平衡 $q=p$ をずらすことなくクリティックに減衰を加えられる。

Mescheder et al. (2018) は、2 つの**ゼロ中心勾配ペナルティ**として R_1（実データ $x\sim p$ 上で評価）と R_2（生成サンプル $x'\sim q$ 上で評価）を次のように定義している。

$$\begin{aligned} R_1&=\frac{\gamma}{2}\,\mathbb{E}_{x\sim p}\Big[\big\|\nabla_x D(x)\big\|^2\Big],\\ R_2&=\frac{\gamma}{2}\,\mathbb{E}_{x'\sim q}\Big[\big\|\nabla_{x'} D(x')\big\|^2\Big], \end{aligned} \tag{16.27}$$

ここで $\gamma>0$ はペナルティの重みである。クリティックは $V_\ell(D)-R_1-R_2$ を最大化し、生成器の目的関数は変わらない。$m=(p+q)/2$ とおくと、両者の和は混合分布のもとでの二乗勾配ノルムの期待値となる。

$$R_1 + R_2 = \gamma\, \mathbb{E}_{x\sim m}\Big[\big\|\nabla_x D(x)\big\|^2\Big], \qquad m = \frac{p+q}{2}. \tag{16.28}$$

16.4.2.1 Dirac-GAN の減衰

線形クリティック $D_\psi(x) = \psi x$ に対しては、至るところで $\nabla_x D_\psi(x) = \psi$ であるため、$R_1 = R_2 = \frac{\gamma}{2}\psi^2$ となる。クリティックの目的関数から $\frac{\gamma}{2}\psi^2$ を引くと、線形減衰項 $-\gamma\psi$ が $\dot\psi$ に加わる（$-2\gamma\psi$ となるのは R_1 と R_2 の両方を含めた場合である）。

$$\dot\theta = \psi\, \ell'(-\psi\theta), \qquad \dot\psi = -\theta\, \ell'(-\psi\theta) - \gamma\psi.$$

$(0,0)$ におけるヤコビ行列とその固有値は次のようになる。

$$J_\gamma = \begin{pmatrix} 0 & \ell'(0) \\ -\ell'(0) & -\gamma \end{pmatrix}, \qquad \nu_{1,2} \;=\; -\frac{\gamma}{2} \pm \sqrt{\frac{\gamma^2}{4} - \ell'(0)^2}. \tag{16.29}$$

すべての $\gamma > 0$ に対して、両方の固有値は厳密に負の実部（$\mathrm{Re}(\nu_{1,2}) < 0$）を持つため、平衡点 $(0,0)$ は局所指数安定（locally exponentially stable）である。特性多項式 $\nu^2 + \gamma\nu + \ell'(0)^2 = 0$ は減衰調和振動子（38.1.2.5 節）のものと同一であり、次の 3 つのレジームを持つ。

1. **不足減衰**（underdamped、$\gamma < 2\ell'(0)$）：実部 $-\gamma/2$ を持つ複素共役固有値となり、軌跡は内向きに渦を巻く（同時オイラーステップのもとでは $h < \gamma/\ell'(0)^2$ のときに収束する。演習問題 1）。
2. **臨界減衰**（critically damped、$\gamma = 2\ell'(0)$、すなわちロジスティック損失では $\gamma = 1$）：重複実固有値 $\nu = -\ell'(0)$ を持ち、最も速い非振動的減衰をもたらす。
3. **過減衰**（overdamped、$\gamma > 2\ell'(0)$）：2 つの負の実固有値を持ち、遅いモード $\nu \approx -\ell'(0)^2/\gamma$ は γ が大きくなるほどゆっくりと減衰する。

16.4.2.2 相図

Equation 16.25 を $(\theta_0, \psi_0) = (1, 1)$（初期半径 $\sqrt{2} \approx 1.41$）から、4 次のルンゲ・クッタ法（`rk4_step`）、同時前進オイラー法（`euler_step`）、および交互勾配更新（`alt_step`）を用いて数値積分し、これら 4 つのレジームを検証する。

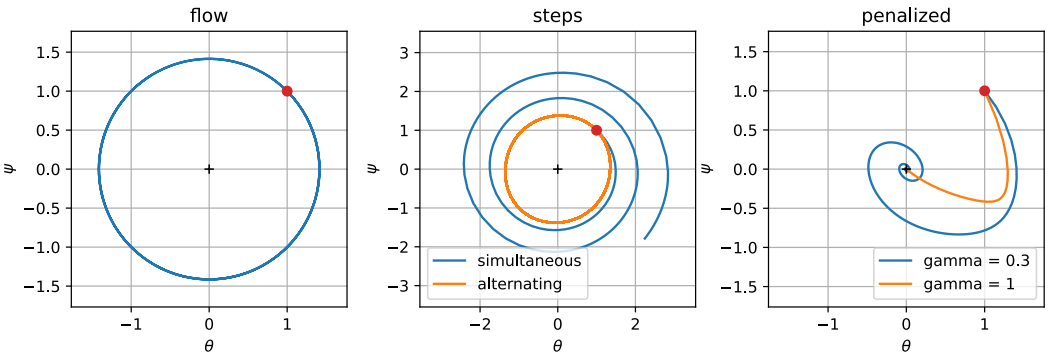
```
def dirac_field(theta, psi, gamma=0.0):
    """Simultaneous gradient field of the Dirac-GAN with the logistic loss."""
    w = 1.0 / (1.0 + np.exp(-psi * theta))      # l'(-psi theta) = w_sat
    return np.array([psi * w, -theta * w - gamma * psi])

def euler_step(x, h, gamma):                    # one simultaneous step
    return x + h * dirac_field(*x, gamma)

def alt_step(x, h, gamma):                      # critic first, then generator
    theta, psi = x
    psi = psi + h * dirac_field(theta, psi, gamma)[1]
    return np.array([theta + h * dirac_field(theta, psi, gamma)[0], psi])

def rk4_step(x, h, gamma):                      # one RK4 step of the ODE
    k1 = dirac_field(*x, gamma)
```

```
    k2 = dirac_field(*(x + h / 2 * k1), gamma)
    k3 = dirac_field(*(x + h / 2 * k2), gamma)
    k4 = dirac_field(*(x + h * k3), gamma)
    return x + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

def trajectory(step, x0, h, steps, gamma=0.0):
    xs = [np.array(x0)]
    for _ in range(steps):
        xs.append(step(xs[-1], h, gamma))
    return np.stack(xs)
```

すべての軌道は赤い点 $(\theta, \psi) = (1, 1)$（黒い十字で示す平衡点からの距離 $\sqrt{2} \approx 1.41$）から出発する。左のパネルは連続時間フローを示す。中央のパネルは、ステップサイズ $h = 0.2$ による 300 回の同時ステップと 2,000 回の交互ステップを示す。右のパネルは、ペナルティを加えた同じステップサイズによる 300 回の同時ステップを、不足減衰レジームの $\gamma = 0.3$ と臨界減衰の $\gamma = 1$ について示す。

```
x0 = np.array([1.0, 1.0])
panels = {'flow': {'RK4': trajectory(rk4_step, x0, 0.05, 800)},
          'steps': {'simultaneous': trajectory(euler_step, x0, 0.2, 300),
                    'alternating': trajectory(alt_step, x0, 0.2, 2000)},
          'penalized': {
              'gamma = 0.3': trajectory(euler_step, x0, 0.2, 300, gamma=0.3),
              'gamma = 1': trajectory(euler_step, x0, 0.2, 300, gamma=1.0)}}
fig, axes = d2l.plt.subplots(1, 3, figsize=(9.5, 3.4))
for ax, (title, paths) in zip(axes, panels.items()):
    for name, P in paths.items():
        r = np.linalg.norm(P, axis=1)           # distance to the equilibrium
        late = r[len(r) // 2:]
        print(f'{title}, {name}: {r[0]:.3g} -> {r[-1]:.3g} in {len(r) - 1} '
              f'steps; second half in [{late.min():.3g}, {late.max():.3g}]')
    lim = 1.25 * max(np.abs(P).max() for P in paths.values())
    d2l.plot([P[:, 0] for P in paths.values()] + [x0[:1], [0.0]],
             [P[:, 1] for P in paths.values()] + [x0[1:], [0.0]],
             r'$\theta$', r'$\psi$', list(paths) if len(paths) > 1 else None,
             xlim=(-lim, lim), ylim=(-lim, lim), axes=ax,
             fmts=('C0-', 'C1-')[:len(paths)] + ('C3o', 'k+'),
             legend_loc='lower right' if title == 'penalized' else None)
    ax.set_title(title)
    ax.set_aspect('equal')
fig.tight_layout()
```

```
flow, RK4: 1.41 -> 1.41 in 800 steps; second half in [1.41, 1.41]
steps, simultaneous: 1.41 -> 2.87 in 300 steps; second half in [2.44, 2.87]
steps, alternating: 1.41 -> 1.36 in 2000 steps; second half in [1.34, 1.42]
penalized, gamma = 0.3: 1.41 -> 0.000652 in 300 steps; second half in [0.000652, 0.0396]
penalized, gamma = 1: 1.41 -> 6.97e-12 in 300 steps; second half in [6.97e-12, 2.33e-05]
```



相図から 4 つの予測すべてが確認できる。連続時間フロー（RK4）は半径 1.41 の円上を周回し、同時オイラーステップは 300 ステップ後に半径 2.87 まで外向きに渦を巻いて広がり、交互ステップは $[1.34, 1.42]$ の内部を周回する。そしてゼロ中心ペナルティを加えると、同時オイラーステップは原点へと直接収束する（半径 6.5×10^{-4} に不足減衰 $\gamma = 0.3$ で到達し、7×10^{-12} に臨界減衰 $\gamma = 1$ で到達する）。

16.4.2.3 一般のネットワークに対する局所収束とペナルティの比較

Mescheder et al. (2018) は、この局所安定性解析を 2 パラメータの Dirac-GAN から一般のニューラルネットワークへと拡張した。すなわち、生成器がデータ分布 p を表し、クリティックが $\mathrm{supp}(p)$ 上で平坦であるような平衡の近傍において、クリティックが生成器の一次摂動を検出するのに十分な局所表現能力を備えていれば、R_1 または R_2（および $\ell'(0) \neq 0, \ell''(0) < 0$ を満たす狭義凹損失）を用いた同時および交互勾配更新は、平衡多様体へ指数関数的に収束する（Huang et al. (2024) は相対論的ペアリング目的関数に対しても同様の局所定理を確立している）。

この解析から、実践上の 2 つの設計選択の理由も説明できる。

- **ゼロ中心（R_1, R_2）対 1 中心（WGAN-GP）ペナルティ：** WGAN-GP は $(\|\nabla_x D\| - 1)^2$ にペナルティを課すため、$q = p$ のときでさえクリティックに傾き 1 を維持することを強制する。線形利得 $\ell(t) = t$ を持つ Dirac-GAN では、1 中心ペナルティによって $\theta = 0$ への収束が妨げられる。$\gamma \geq 2$ のときはワッサースタイン勾配とペナルティが釣り合う $|\theta| = \gamma$ に生成器を停留させ、それより小さい γ では生成器がこの点をオーバーシュートして周期軌道を回り続ける（演習問題 6）。
- **R_1 と R_2 の併用対 R_1 単独の使用：** $q = p$ の近傍では、R_1 と R_2 は等価である。しかし平衡から離れた領域では、R_1 は実データ点 $x \sim p$ におけるクリティックの傾きにしかペナルティを課さず、生成点 $x' \sim q$ における傾きを制約しないままである（たとえば $D_c(x) = -cx^2$ は $p = \delta_0$ において傾きがゼロであるため $R_1 = 0$ となり、$c \to \infty$ が許容されて $x' = 1$ における生成器の勾配が爆発する。演習問題 3）。R_2 を加えることで、生成サンプルにおけるクリティックの勾配も有界に抑えられる。

16.4.3 勾配ペナルティが価値関数に及ぼす影響

ゼロ中心ペナルティ $R_1 + R_2 = \gamma\, \mathbb{E}_m[\|\nabla_x D\|^2]$ は、平衡の近傍で生成器が最小化するダイバージェンスをどのように変化させるのだろうか。

16.4.3.1 局所二次近似とガウス分布のシフト

$q = p$ の近傍では、最適なクリティック D は小さい。$\ell(t) \approx \ell(0) + \ell'(0)t - \frac{1}{2}|\ell''(0)|t^2$ により D の 2 次まで $V_\ell(D)$ の内部を展開すると、次式が得られる。

$$V_\ell(D) \approx 2\ell(0) + \ell'(0)\,\langle p - q,\, D\rangle - |\ell''(0)| \int m\, D^2,$$

ここで $\langle p - q, D\rangle = \mathbb{E}_p[D] - \mathbb{E}_q[D]$ である。2 つの単位分散ガウス分布 $p = \mathcal{N}(0,1)$ と $q = \mathcal{N}(\mu,1)$（小さなシフト μ だけ離れている）および線形クリティック $D(x) = cx$ を考える。μ の主要項までとると、$\langle p - q, D\rangle = -c\mu$、$\int m\, D^2 \approx c^2$ であり、ペナルティは $\gamma \int m\, \|\nabla_x D\|^2 = \gamma c^2$ となる。凹二次関数 $-\ell'(0)\mu c - (|\ell''(0)| + \gamma)c^2$ を c について最大化すると、最適な傾き $c^\star = -\frac{\ell'(0)\mu}{2(|\ell''(0)|+\gamma)}$ と次の値が得られる。

$$\frac{\ell'(0)^2\mu^2}{4\,(|\ell''(0)| + \gamma)} = \frac{\mu^2}{4 + 16\gamma}$$

for the logistic loss, at slope $c^\star = -\dfrac{\mu}{1 + 4\gamma}$.

このように、ペナルティによってクリティックの傾きと局所的な値はともに $1 + 4\gamma$ 分の 1 に縮小する。$\gamma = 0$ では $\mu^2/4 \approx 2\,\mathrm{JS}(p,q)$ が復元され、大きい γ では値は $\mu^2/(16\gamma) = \frac{\ell'(0)^2}{4\gamma} W_2(p,q)^2$ のようにスケールする。以下では、1 次元グリッド上でニュートン法により計算した厳密なペナルティ付き最適値と比較し、この閉形式近似を検証する。

```
def penalized_value(p, q, gamma, x, newton_steps=30):
    """Value of the logistic game minus gamma E_m[D'^2], on a grid."""
    dx, m = x[1] - x[0], (p + q) / 2
    diff = np.diff(np.eye(len(x)), axis=0) / dx       # D -> D'
    K = 2 * gamma * dx * diff.T @ (((m[1:] + m[:-1]) / 2)[:, None] * diff)
    D = np.zeros_like(x)
    for _ in range(newton_steps):                     # Newton ascent on D
        s = 1 / (1 + np.exp(-D))                      # sigma(D)
        grad = (p * (1 - s) - q * s) * dx - K @ D     # gradient of the value
        neg_hess = np.diag(2 * m * s * (1 - s) * dx) + K  # minus its Hessian
        D = D + np.linalg.solve(neg_hess, grad)
    value = -(p * np.logaddexp(0, -D) + q * np.logaddexp(0, D)).sum() * dx
    return value - D @ K @ D / 2 + 2 * np.log(2)       # minus 2 l(0)

x = np.linspace(-8, 8, 401)
for mu in (0.2, 1, 3):
    p, q = [np.exp(-(x - c) ** 2 / 2) / np.sqrt(2 * np.pi) for c in (0, mu)]
    for gamma in (0, 0.01, 1, 10):
        exact = penalized_value(p, q, gamma, x)
        local = mu ** 2 / (4 + 16 * gamma)
        large = mu ** 2 / (16 * gamma) if gamma else np.inf
        print(f'mu {mu:>3}, gamma {gamma:>4}: exact {exact:.3e}, local '
              f'{local:.3e} ({local / exact:.2f}x), large-gamma {large:.3e} '
              f'({large / exact:.2f}x)')
```

```
mu 0.2, gamma    0: exact 9.950e-03, local 1.000e-02 (1.00x), large-gamma inf (infx)
mu 0.2, gamma 0.01: exact 9.566e-03, local 9.615e-03 (1.01x), large-gamma 2.500e-01 (26.13x)
mu 0.2, gamma    1: exact 1.996e-03, local 2.000e-03 (1.00x), large-gamma 2.500e-03 (1.25x)
mu 0.2, gamma   10: exact 2.438e-04, local 2.439e-04 (1.00x), large-gamma 2.500e-04 (1.03x)
mu   1, gamma    0: exact 2.228e-01, local 2.500e-01 (1.12x), large-gamma inf (infx)
```

```
mu   1, gamma 0.01: exact 2.135e-01, local 2.404e-01 (1.13x), large-gamma 6.250e+00 (29.27x)
mu   1, gamma    1: exact 4.818e-02, local 5.000e-02 (1.04x), large-gamma 6.250e-02 (1.30x)
mu   1, gamma   10: exact 6.120e-03, local 6.098e-03 (1.00x), large-gamma 6.250e-03 (1.02x)
mu   3, gamma    0: exact 1.054e+00, local 2.250e+00 (2.14x), large-gamma inf (infx)
mu   3, gamma 0.01: exact 1.012e+00, local 2.163e+00 (2.14x), large-gamma 5.625e+01 (55.57x)
mu   3, gamma    1: exact 3.909e-01, local 4.500e-01 (1.15x), large-gamma 5.625e-01 (1.44x)
mu   3, gamma   10: exact 6.862e-02, local 5.488e-02 (0.80x), large-gamma 5.625e-02 (0.82x)
```

シフトが小さい場合（$\mu = 0.2$）、二次近似式 $\mu^2/(4+16\gamma)$ はすべての γ にわたって厳密な数値解と 1% 以内の誤差で一致する。

16.4.3.2 双対ソボレフノルムとローパスフィルタリング

平衡近傍にある一般の分布 p と q に対しては、部分積分を用いることで勾配ペナルティを $\int m\,\|\nabla_x D\|^2 = \langle D, \mathcal{L}_m D\rangle$ と書き換えられる。ここで $\mathcal{L}_m D = -\nabla\cdot(m\nabla D)$ は m 重み付きラプラシアンである。局所二次近似をすべての関数 D にわたって最大化すると、次式が得られる（演習問題 5）。

$$\begin{aligned}\sup_D \Big\{\ell'(0)\,\langle p-q,\,D\rangle - |\ell''(0)|\int m\,D^2 - \gamma\int m\,\|\nabla_x D\|^2\Big\}\\ = \frac{\ell'(0)^2}{4}\,\big\langle p-q,\,(|\ell''(0)|\,m+\gamma\mathcal{L}_m)^{-1}(p-q)\big\rangle.\end{aligned} \tag{16.30}$$

$\gamma\to 0$ のとき、この式は $\frac{\ell'(0)^2}{4|\ell''(0)|}\int (p-q)^2/m \approx 2\,\mathrm{JS}(p,q)$ に帰着する。一方、$\gamma\to\infty$ のときは、**双対ソボレフノルム**（dual Sobolev norm）$\dot{H}^{-1}(m)$ の二乗の定数倍に漸近する。

$$\begin{aligned}&\frac{\ell'(0)^2}{4\gamma}\,\|p-q\|^2_{\dot{H}^{-1}(m)},\\ &\|v\|_{\dot{H}^{-1}(m)} = \sup\Big\{\langle v, D\rangle : \int m\|\nabla_x D\|^2 \le 1\Big\}.\end{aligned} \tag{16.31}$$

m がほぼ一様な領域では、$\mathcal{L}_m$ は波数 ω のフーリエモードに $m\omega^2$ を掛けるため、そのモードの重み付けは $(|\ell''(0)|+\gamma\omega^2)^{-1}$ となる（ペナルティなしの $|\ell''(0)|^{-1}$ に代わる）。したがって、R_1+R_2 は $p-q$ に対する空間的な**ローパスフィルタ**として働き、そのカットオフ長さスケールは $\sqrt{\gamma/|\ell''(0)|}$（ロジスティック損失では $2\sqrt{\gamma}$）となる。$\nabla_x D$ は長さの逆数の単位を持つため、γ は長さの二乗の単位を持ち、データの解像度やノルムに応じてスケールする。なお、R_1+R_2 が $\nabla_x D$ にペナルティを課すのは $m=(p+q)/2$ の台の上のみである。互いに素な台の間に広がる空白領域をまたぐ勾配は、依然としてニューラルネットワークであるクリティックの滑らかさに依存する（演習問題 2）。

16.4.4 R3GAN レシピ

R3GAN (Huang et al. 2024) は、相対論的ペアリング損失（`d2l.rpgan_loss_D` および `d2l.rpgan_loss_G`、16.3 節）と両方のゼロ中心勾配ペナルティ R_1、R_2 を組み合わせた手法である。

16.4.4.1 損失とペナルティの実装

比較のため、`gan_loss_D` と `gan_loss_G`（16.1 節の古典的なロジスティック識別器損失と非飽和生成器損失）も同じ（`critic, real, fake`）シグネチャで実装し、保存しておく。

```
def gan_loss_D(critic, real, fake):
    """Logistic critic loss -E[log sigma(D(x))] - E[log sigma(-D(x'))]: the
    sum of two batch means, 2 log 2 at equilibrium, as update_D."""
    return (jax.nn.softplus(-critic(real)).mean()
            + jax.nn.softplus(critic(fake)).mean())


def gan_loss_G(critic, real, fake):
    """Non-saturating generator loss -E[log sigma(D(x'))]; real is unused."""
    return jax.nn.softplus(-critic(fake)).mean()
```

ペナルティ関数は、係数 $\gamma/2$ を掛ける前の、両バッチにおけるサンプルごとの二乗入力勾配を返す。ここではクリティックのスコアの**和**を入力バッチに関して微分する。各スコアはそれぞれの行のみに依存するため、1 回の逆伝播ですべてのサンプルの入力勾配が求まる（行同士を混合するバッチ正規化を用いると、この性質は成り立たなくなる）。入力勾配は自動微分の計算グラフ内に保持されるため、オプティマイザはクリティックのパラメータに関してペナルティをさらに微分できる。このときサンプル自体は定数として固定する。2 回目の微分を伴うため、ペナルティ付きクリティックステップの計算コストはおよそ 2 回分の逆伝播に相当する。

スコアの和に対する `jax.grad` によって入力勾配が得られ、クリティックに関してペナルティを微分するとその外側に 2 回目の勾配計算が入れ子になる。また、`jax.lax.stop_gradient` によってサンプルを定数として扱う。

```
def r1_r2_penalty(critic, real, fake):
    """Per-sample squared critic input gradients on real (R1) and fake (R2),
    before the gamma/2 scale."""
    def sq_grad_norm(x):
        # Rows are independent, so the gradient of the sum is per-sample
        grad = jax.grad(lambda x: critic(x).sum())(jax.lax.stop_gradient(x))
        return (grad.reshape(x.shape[0], -1) ** 2).sum(axis=1)
    return sq_grad_norm(real), sq_grad_norm(fake)
```

16.4.4.2 訓練ステップ

訓練ループでは、各プレイヤーのステップを 1 回ずつ交互に実行する。`critic_step` は生成バッチを固定した状態で、クリティック損失に両ペナルティの $\gamma/2$ 倍を加えた目的関数に対してオプティマイザを 1 ステップ進め、損失ならびに実サンプルと生成サンプルにおける二乗入力勾配のバッチ平均（R_1 と R_2 の $\gamma/2$ を掛ける前の値）を返す。これにより、訓練ループ内でクリティックの急峻さを記録できる。続いて `generator_step` が、更新後のクリティックに対する生成器損失について 1 ステップ進める。オプションのラベル `y` は、16.6 節の条件付きクリティックで用いる。

```
@nnx.jit(static_argnums=2)
def critic_step(net_D, opt_D, loss_D, real, fake, gamma, y=None):
    """One critic step on loss_D + gamma/2 (R1 + R2); return the loss and
    the batch means of R1 and R2."""
    def objective(net_D):
        critic = net_D if y is None else lambda x: net_D(x, y)
        r1, r2 = r1_r2_penalty(critic, real, fake)
        loss = loss_D(critic, real, fake)
        return (loss + gamma / 2 * (r1 + r2).mean(),
```

```
                (loss, r1.mean(), r2.mean()))
    (_, logs), grads = nnx.value_and_grad(objective, has_aux=True)(net_D)
    opt_D.update(net_D, grads)
    return logs


@nnx.jit(static_argnums=2)
def generator_step(net_G, opt_G, loss_G, net_D, real, z, y=None):
    """One generator step on loss_G against the critic; return the loss."""
    def objective(net_G):
        critic = net_D if y is None else lambda x: net_D(x, y)
        return loss_G(critic, real, net_G(z) if y is None else net_G(z, y))
    loss, grads = nnx.value_and_grad(objective)(net_G)
    opt_G.update(net_G, grads)
    return loss
```

損失関数はコンパイルされるステップ関数の静的引数であるため、損失のペアごとに 1 回だけコンパイルされる。一方、γ はトレースされるため、再コンパイルなしに変更できる。

16.4.4.3 25 ガウスグリッドにおけるモード被覆

ニューラルネットワークにおけるモード被覆（mode coverage）に $R_1 + R_2$ とペアワイズランキングがどう影響するかを検証するため、2 次元ガウス分布の 5×5 グリッド（間隔 2.0、標準偏差 $\sigma = 0.05$）上で、3 層の MLP 生成器（64 次元の潜在変数、256 個の ReLU ユニットからなる 2 つの隠れ層）と 3 層の Leaky ReLU MLP クリティックを訓練する。生成された 10,000 個のサンプルを次の 4 つの評価指標で評価する。

1. **被覆モード数**（covered modes）：モード上サンプルのうち、少なくとも $1/5$ 倍（均等配分 $1/25$ に対する比率）を受け取ったモードの数。
2. **モード上サンプル比率**（on-mode fraction）：最近傍のモード中心から $3\sigma = 0.15$ 以内に入るサンプルの割合（真の分布では 98.9%）。
3. 最近傍の中心までの**平均距離**（真の分布では ≈ 0.06）。
4. 生成サンプルにおける**クリティックの急峻さ**（critic steepness）$\mathbb{E}_q[\|\nabla_x D(x')\|^2]$。

```
def grid_centers(side=5, spacing=2.0):
    coords = (jnp.arange(side) - (side - 1) / 2) * spacing
    gx, gy = jnp.meshgrid(coords, coords, indexing='ij')
    return jnp.stack([gx.reshape(-1), gy.reshape(-1)], axis=1)

def sample_modes(key, centers, n, width=0.05):
    k1, k2 = jax.random.split(key)
    idx = jax.random.randint(k1, (n,), 0, centers.shape[0])
    return centers[idx] + width * jax.random.normal(k2, (n, 2))
```

生成された点群をグリッドの `centers` に照らして評価するため、`mode_stats` は各サンプルを最近傍のモード中心に割り当て、被覆モード数、3σ 以内のサンプル比率、および最近傍中心までの平均距離を返す。

```
def mode_stats(x, centers, width=0.05):
    """Modes covered (each holds a fifth of an even share of the on-mode
```

```
    samples), fraction within three widths, mean distance."""
    dist = np.linalg.norm(x[:, None] - centers[None], axis=-1)
    near, idx = dist.min(axis=1), dist.argmin(axis=1)
    on = near <= 3 * width
    counts = np.bincount(idx[on], minlength=len(centers))
    covered = (counts >= on.sum() / len(centers) / 5).sum()
    return covered, on.mean(), near.mean()
```

`train_grid` は `critic_step` と `generator_step` を 10,000 ステップ（バッチサイズ 1,024）交互に実行する。オプティマイザには $\beta_1 = 0$ の Adam（回転ベクトル場において正の一次モーメントのモメンタムは安定領域を狭めうるため、R3GAN に倣う）を用い、2×10^{-3} から 2×10^{-4} へのコサイン学習率減衰を適用する。

```
def train_grid(centers, loss_D, loss_G, gamma, steps=10000, lr=2e-3,
               batch=1024, latent=64, seed=0, checkpoints=8):
    """Alternate critic and generator steps; at each checkpoint record the
    on-mode fraction of 2,000 samples, and the critic loss and E_q|grad D|^2
    averaged over the steps since the previous checkpoint."""
    rngs = nnx.Rngs(seed)
    net_G = d2l.make_mlp([latent, 256, 256, 2], nnx.relu, rngs=rngs)
    net_D = d2l.make_mlp([2, 256, 256, 1], lambda x: nnx.leaky_relu(x, 0.2),
                         rngs=rngs)
    adam = optax.adam(optax.cosine_decay_schedule(lr, steps, alpha=0.1),
                      b1=0.0, b2=0.99)
    opt_G, opt_D = (nnx.Optimizer(net, adam, wrt=nnx.Param)
                    for net in (net_G, net_D))

    @nnx.jit
    def step(net_G, net_D, opt_G, opt_D, key, logs):
        k1, k2 = jax.random.split(key)
        z = jax.random.normal(k1, (2, batch, latent))    # critic, generator
        real = sample_modes(k2, centers, 2 * batch).reshape(2, batch, 2)
        fake = net_G(z[0])
        loss, _, grad_q = critic_step(net_D, opt_D, loss_D, real[0], fake,
                                      gamma)
        generator_step(net_G, opt_G, loss_G, net_D, real[1], z[1])
        return logs + jnp.stack([loss, grad_q])          # summed on the device

    step = nnx.cached_partial(step, net_G, net_D, opt_G, opt_D)
    trace, every, grid = [], steps // checkpoints, np.asarray(centers)
    keys = jax.random.split(rngs(), steps).reshape(checkpoints, every)
    for chunk in keys:                                   # one checkpoint each
        logs = jnp.zeros(2)
        for key in chunk:
            logs = step(key, logs)
        x = np.asarray(net_G(jax.random.normal(rngs(), (2000, latent))))
        trace.append((mode_stats(x, grid)[1], *(logs / every).tolist()))
    return net_G, trace
```

ここでは 2×2 の要因実験として、2 つの目的関数（古典的な非飽和型 `NS` と `pairing`）に $\gamma = 0$（ペナルティなし）および $\gamma = 1$（$R_1 + R_2$）を掛け合わせた各条件を 3 シードずつ実行し、さらに過減衰の実行（$\gamma = 10$）を 1 つ加える。

```
centers, latent = grid_centers(), 64
ns, pairing = (gan_loss_D, gan_loss_G), (d2l.rpgan_loss_D, d2l.rpgan_loss_G)
arms = {'NS': (*ns, 0.0, 3), 'NS + R1 + R2': (*ns, 1.0, 3),
        'pairing': (*pairing, 0.0, 3), 'pairing + R1 + R2': (*pairing, 1.0, 3),
        'pairing + R1 + R2, gamma=10': (*pairing, 10.0, 1)}
runs, key = {name: [] for name in arms}, jax.random.key(1)  # samples, trace
for name, (loss_D, loss_G, gamma, n_runs) in arms.items():
    for seed in range(n_runs):
        net_G, trace = train_grid(centers, loss_D, loss_G, gamma,
                                  latent=latent, seed=seed)
        key, subkey = jax.random.split(key)
        z = jax.random.normal(subkey, (10000, latent))
        runs[name].append((np.asarray(net_G(z)), trace))
grid = np.asarray(centers)
```

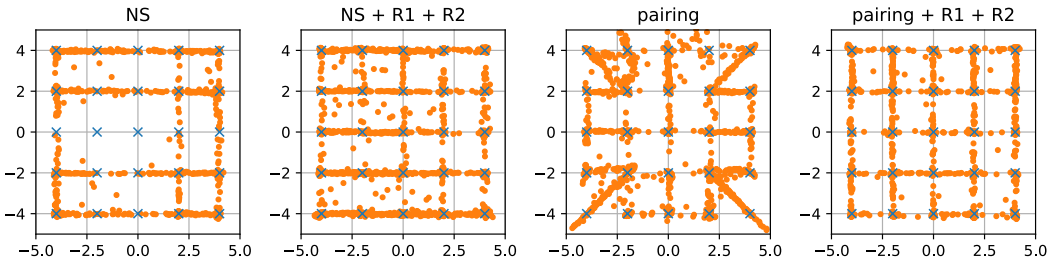
各実行から生成された 10,000 個のサンプルに対する `mode_stats` の結果を、クリティックの最終的な二乗勾配ノルム $\mathbb{E}_q[\|\nabla_x D(x')\|^2]$ と並べて表にまとめる。

```
print(f'{"arm":28} run  modes  on-mode  distance  E_q|grad D|^2')
for name, results in runs.items():
    for seed, (x, trace) in enumerate(results):
        modes, on, dist = mode_stats(x, grid)
        print(f'{name:28} {seed:3}  {modes:2}/25  {on:7.2f}  {dist:8.2f}'
              f'  {trace[-1][2]:13.1e}')
```

```
arm                          run  modes  on-mode  distance  E_q|grad D|^2
NS                             0  12/25     0.81      0.15        1.0e+01
NS                             1  15/25     0.94      0.08        7.2e-01
NS                             2  24/25     0.74      0.17        5.9e+00
NS + R1 + R2                   0  25/25     0.67      0.21        2.4e-03
NS + R1 + R2                   1  25/25     0.87      0.11        4.1e-04
NS + R1 + R2                   2  25/25     0.79      0.14        8.2e-04
pairing                        0  20/25     0.64      0.24        3.1e+00
pairing                        1  17/25     0.78      0.15        1.3e+01
pairing                        2  23/25     0.77      0.16        6.3e+00
pairing + R1 + R2              0  25/25     0.79      0.15        8.2e-04
pairing + R1 + R2              1  25/25     0.70      0.19        1.4e-03
pairing + R1 + R2              2  25/25     0.79      0.15        7.4e-04
pairing + R1 + R2, gamma=10    0  25/25     0.28      0.39        9.0e-05
```

主要な 4 つの実験アームそれぞれの最初の実行から生成された 3,000 個のサンプルを、25 個のモード中心（`grid`）と重ねてプロットすると、モード被覆の違いが直接見て取れる。

```
fig, axes = d2l.plt.subplots(1, 4, figsize=(10, 2.9))
for ax, name in zip(axes, list(runs)[:4]):
    x = runs[name][0][0][:3000]              # run 0; centers drawn last
    d2l.plot([x[:, 0], grid[:, 0]], [x[:, 1], grid[:, 1]],
             fmts=('C1.', 'C0x'), xlim=(-5, 5), ylim=(-5, 5), axes=ax)
    ax.set_title(name)
    ax.set_aspect('equal')
fig.tight_layout()
```



この結果から、勾配ペナルティとペアリング損失それぞれの役割が明確に分離される。

- **勾配ペナルティがモード被覆をもたらす：**ペナルティなし（$\gamma = 0$）では、クリティックの勾配は急峻なままであり（$\mathbb{E}_q\|\nabla_x D\|^2 \in [0.7, 17]$）、モード被覆は不安定になる（シードによって 12〜25 モードに落ち込む）。$R_1 + R_2$ を $\gamma = 1$ で課すと、クリティックの勾配は 3 桁減衰し（$\sim 10^{-3}$）、NS と pairing の双方ですべての実行が全 25 モードを被覆する。
- **大きい γ における過減衰：**γ を 10 に増やすとクリティックが過度に平坦化され（$\sim 10^{-4}$）、収束が遅くなって 10,000 ステップ後でも 20〜30% のサンプルしかモードの 3σ 以内に収まらない。

最後に、チェックポイントのトレースを調べると、落ち着いたクリティック損失曲線の解釈になぜ注意が必要かがわかる。

```
traces = {name: np.array(results[0][1]) for name, results in runs.items()}
for col, stat in enumerate(['on-mode fraction', 'critic loss']):
    print(f'{stat} of run 0 at every 1,250 steps')
    for name, trace in traces.items():          # trace: (checkpoint, stat)
        print(f'  {name:28}', ' '.join(f'{v:4.2f}' for v in trace[:, col]))
```

```
on-mode fraction of run 0 at every 1,250 steps
  NS                           0.01 0.02 0.06 0.15 0.52 0.68 0.78 0.81
  NS + R1 + R2                 0.03 0.03 0.06 0.11 0.27 0.36 0.59 0.68
  pairing                      0.01 0.07 0.15 0.25 0.27 0.46 0.59 0.62
  pairing + R1 + R2            0.03 0.03 0.04 0.07 0.51 0.61 0.71 0.79
  pairing + R1 + R2, gamma=10  0.02 0.02 0.03 0.05 0.06 0.09 0.20 0.27
critic loss of run 0 at every 1,250 steps
  NS                           1.34 1.15 0.95 0.93 0.94 0.95 0.96 0.96
  NS + R1 + R2                 1.38 1.37 1.36 1.37 1.37 1.38 1.38 1.38
  pairing                      0.68 0.67 0.64 0.61 0.59 0.58 0.58 0.58
  pairing + R1 + R2            0.68 0.68 0.67 0.68 0.68 0.69 0.69 0.69
  pairing + R1 + R2, gamma=10  0.69 0.70 0.69 0.69 0.69 0.69 0.69 0.69
```

ペナルティ付きの実行では、クリティック損失は最初のチェックポイントから平衡値（NS では $2\log 2 \approx 1.39$、pairing では $\log 2 \approx 0.69$）の付近に留まる。これは生成サンプルのわずか 1〜3% しかモード上に乗っていない段階でさえ生じる現象であり、勾配ペナルティが至るところで $D(x)$ を小さく保つからである。クリティック損失が平衡値にあることはクリティックが平坦であることを示すにすぎず、必ずしも生成器が収束したことを意味しない。

16.4.5 まとめ

2 パラメータの Dirac-GAN（$p = \delta_0, q_\theta = \delta_\theta, D_\psi(x) = \psi x$）では、連続時間勾配フローは一定半径の円周上を周回し、離散時間の同時ステップは外向きに渦を巻いて発散し、離散時間の交互ステップは有界な帯状領域に閉じ込められる。ゼロ中心勾配ペナルティ $R_1 = \frac{\gamma}{2}\mathbb{E}_p[\|\nabla_x D(x)\|^2]$ お

よび $R_2 = \frac{\gamma}{2}\mathbb{E}_{q}[\|\nabla_{x'} D(x')\|^2]$ を差し引くと、ヤコビ行列の固有値が開左半平面 $(\mathrm{Re}(\nu_{1,2}) < 0)$ へとすべての $\gamma > 0$ に対して移動し、平衡点は局所指数安定となる。平衡の近傍では、$R_1 + R_2$ は $p - q$ に対する空間ローパスフィルタ（ロジスティック損失のもとでのカットオフ長さスケールは $2\sqrt{\gamma}$）として働き、大きなスケールでの $2\,\mathrm{JS}(p, q)$ と細かなスケールでの二乗双対ソボレフノルム $\|p - q\|^2_{\dot{H}^{-1}(m)}$ の間を補間する。実験的にも、25 ガウスグリッドにおいて $R_1 + R_2$（$\gamma = 1$）はクリティックの急峻さを 3 桁減衰させ、古典的なロジスティック目的関数と相対論的ペアリング目的関数の双方で完全なモード被覆を達成した。

16.4.6 演習問題

1. [code] **Dirac-GAN の離散時間安定性。** ステップサイズ h の同時勾配降下法は $(\theta, \psi) \mapsto (\theta, \psi) + h\,\mathbf{u}(\theta, \psi)$ と写像する。
   1. $\gamma = 0$ のとき、$(0, 0)$ における離散更新のヤコビ行列は固有値 $1 \pm i\,h\,\ell'(0)$ を持ち、その絶対値が $\sqrt{1 + h^2 \ell'(0)^2} > 1$（任意の $h > 0$ に対して）となることを示せ。
   2. $\gamma > 0$ のとき、固有値は $1 + h\nu_{1,2}$（ただし $\nu_{1,2}$ は Equation 16.29 のもの）となる。不足減衰レジーム $\gamma < 2\ell'(0)$ において、離散更新が局所安定（$|1 + h\nu_{1,2}| < 1$）となるのは $h < \gamma / \ell'(0)^2$ のとき、かつそのときに限ることを示せ。
   3. $\gamma = 0.3$（$\ell'(0) = \frac{1}{2}$、しきい値 $h = 1.2$）におけるこのしきい値を、`trajectory(euler_step, (0.1, 0.1), h, 3000, gamma=0.3)` を $h = 1.1$ と $h = 1.3$ で実行して検証せよ。
2. **Dirac-GAN におけるペナルティ付き最適応答。** Dirac-GAN において $V_\gamma(\theta) = \sup_{\psi}\{V(\theta, \psi) - \gamma\psi^2\}$（$\ell = \log\sigma$）を考える。
   1. 小さい θ に対して、最大点を与える傾きは $\psi^\star \approx -\ell'(0)\theta/(2\gamma)$ かつ $V_\gamma(\theta) = 2\ell(0) + \frac{\ell'(0)^2}{4\gamma}\theta^2 + O(\theta^4)$ となり、Equation 16.31 において $W_2(\delta_0, \delta_\theta) = |\theta|$ とした結果に一致することを示せ。
   2. 次に、D を $\gamma \int m \|\nabla_x D\|^2$ でペナルティを課されたすべての微分可能関数にわたって動かすとする。点質量 $p = \delta_0$ と $q_\theta = \delta_\theta$（$\theta \neq 0$）に対して、$\sup_D\{V_\ell(D) - R_1 - R_2\} = 2\sup_t \ell(t)$ が θ に依存しないことを示せ。有限サイズのニューラルネットワークのクリティックが、それでもこの隔たりをまたいで非ゼロの勾配を与えるのはなぜか。
3. **R_1 単独のペナルティ対 $R_1 + R_2$。** $p = \delta_0$、$q = \delta_1$、$\ell = \log\sigma$ とし、二次クリティック $D_c(x) = -cx^2$（$c \geq 0$）を考える。
   1. R_1 と R_2 を D_c に対して計算し、$V_\ell(D_c) - R_1$ が c に関して狭義単調増加であるため有限な最大点を持たないことを示せ。
   2. 非飽和生成器損失 $-\log\sigma(D_c(x'))$ のもとでは、$x' = 1$ における生成器勾配の大きさは c とともに際限なく増大する一方、$R_1 + R_2$ を差し引くと有限な最大点 $c^\star < 1/(8\gamma)$ が得られることを示せ。
4. [code] **Dirac-GAN における交互更新。** ●交互更新では、$\psi_{k+1} = \psi_k + h\,u_\psi(\theta_k, \psi_k)$ に続けて $\theta_{k+1} = \theta_k + h\,u_\theta(\theta_k, \psi_{k+1})$ を適用する。
   1. 合成された更新を $(0, 0)$ の周りで（$\gamma = 0$ として）線形化し、そのヤコビ行列の行列式が 1 であり、固有値が単位円上に乗るのは $h\,\ell'(0) \leq 2$ のとき、かつそのときに限ることを示せ。
   2. `trajectory(alt_step, (0.1, 0.1), h, 10**5)` を $\gamma = 0$ のもとで $h \in \{0.2, 1, 3, 5\}$ に対して実行し、最後の 10^4 ステップにおける最小半径と最大半径を記録して、$\gamma = 0.3$（$h = 0.2$）の場合と比較せよ。
5. [code] **局所二次近似値とローパスフィルタリング。** ●$\mathcal{A} = |\ell''(0)|\,m + \gamma\mathcal{L}_m$（ただし $\mathcal{L}_m D = -\nabla\cdot(m\nabla D)$）とおく。
   1. 部分積分を用いて

$$|\ell''(0)| \int m\, D^2 + \gamma \int m\, \|\nabla_x D\|^2 = \langle D, \mathcal{A} D \rangle$$

を示し、半直線 tD_0 に沿って最大化してコーシー・シュワルツの不等式を適用することで Equation 16.30 を導出せよ。

2. $p = \mathcal{N}(0,1)$、$q(x) = p(x)(1 + 0.1\sin(\omega x))$ とする。`penalized_value` を用いて、$\gamma \in \{0.01, 0.1, 1\}$ におけるペナルティ付きの値と $\gamma = 0$ におけるペナルティなしの値との比を $\omega \in \{0.5, 2, 8\}$ について計算し、一様な m を仮定したフーリエ減衰因子 $(1 + 4\gamma\omega^2)^{-1}$ と比較せよ。

6. [code] **Dirac-GAN における 1 中心ペナルティのダイナミクス。**●WGAN の線形利得 $\ell(t) = t(\ell'(0) = 1)$ において、クリティックの目的関数から 1 中心ペナルティ $\frac{\gamma}{2}(|\psi| - 1)^2$ を差し引いた系を考える。
   1. 同時勾配場が $\psi \neq 0$ では決してゼロにならないことを示し、生成器のパラメータ位置 θ のうち、$\psi \to 0^{\pm}$ としたときの片側極限がゼロになるものを求めよ。
   2. 1 中心ペナルティを用いるように `dirac_field` を変更し、`trajectory` を $(1, 1)$ から $h = 0.01$ で 40,000 ステップ（$\gamma = 1$ および $\gamma = 3$）実行して、$\gamma = 3$ では生成器が $\theta = 3$ で停滞することを検証せよ。

16.5 敵対的画像生成

16.1 節の多層パーセプトロン（multilayer perceptron）は、識別可能な 28×28 の Fashion-MNIST のシルエットを生成できたものの、全結合層が空間的局所性（spatial locality）を無視するため、テクスチャはざらついていた。64×64 の RGB 画像を生成するには、100 次元の潜在ベクトルから 12,288 個のピクセル値を生成する必要がある。ここで 2 つの疑問が生じる。

1. 敵対的画像生成において、どのような畳み込みアーキテクチャであれば安定して訓練できるのか。
2. 尤度が利用できない状況で、画像生成器をどのように定量的に評価・比較できるのか。

畳み込みを用いた生成器と識別器は、Radford et al. (2015) の Deep Convolutional GAN（DCGAN）によって初めて GAN を自然画像へとスケールさせた。その後、汎用的な画像生成では拡散モデル（diffusion model、第 17 章）が主流となったが、畳み込み識別器と特徴空間での評価指標は、現代の画像トークナイザや 1 ステップの蒸留生成器（16.7 節）において依然として中心的な役割を担っている。

本節では、40,597 枚の 64×64 Pokémon スプライトからなるデータセット上でモデルを訓練する。まず 2015 年の古典的な DCGAN アーキテクチャを実装し、続いて R3GAN (Huang et al. 2024) に着想を得た最小構成の正規化なし畳み込みバックボーンを構築する。この正規化なしバックボーンを 3 つの目的関数（ペナルティなしの古典的ロジスティック損失、$R_1 + R_2$ ペナルティ付きの古典的ロジスティック損失、$R_1 + R_2$ ペナルティ付きの相対論的ペアリング損失）のもとで訓練することで、ゼロ中心勾配ペナルティ（zero-centered gradient penalty、16.4 節）がバッチ正規化（batch normalization）やその他のアーキテクチャ上の経験則を代替できるかを検証する。その後、特徴空間における標準的な評価指標であるフレシェ距離（Fréchet Distance、FID）、Kernel Inception Distance（KID）、および k-NN の適合率（precision）と再現率（recall）を実装し、潜在空間での補間と識別器の過学習（overfitting）を調べる。

```
# Deterministic GPU kernels; JAX reads XLA_FLAGS once, before the imports
import os
os.environ['XLA_FLAGS'] = (os.environ.get('XLA_FLAGS', '') +
                           ' --xla_gpu_deterministic_ops=true').strip()
```

16.5.1 DCGAN アーキテクチャ

DCGAN (Radford et al. 2015) は、16.1 節の古典的な非飽和ロジスティック目的関数を維持しつつ、以下の一連のアーキテクチャ規約によって畳み込みネットワークの訓練を安定化させる。

- 生成器における空間的なアップサンプリングには**転置畳み込み**（transposed convolution、分数ストライド畳み込み）を用い、識別器におけるダウンサンプリングにはプーリング層の代わりにストライド付き畳み込み（strided convolution）を用いる。
- 生成器の出力層と識別器の入力層を除くすべての畳み込みの後に、**バッチ正規化**を適用する。
- 生成器は隠れ層の活性化関数に ReLU、出力の活性化関数に tanh（$[-1, 1]$ のピクセル値に対応）を用いるのに対し、識別器は勾配が 0 になるデッドゾーンを防ぐため、全体にわたって Leaky ReLU（slope $= 0.2$）を用いる。
- 重みは $\mathcal{N}(0, 0.02^2)$ から初期化し、Adam では通常より小さいモメンタム $\beta_1 = 0.5$ を用いる。

16.5.1.1 Pokémon スプライトデータセット

ここで用いるデータセットは、pokemondb.net から取得した 721 種の Pokémon の RGB スプライト 40,597 枚で構成されており、64×64 にリサイズしたうえで $[-1, 1]$ にスケーリングしてある。データセット全体がメモリに収まるため、`load_pokemon` で単一のテンソルへとデコードする。

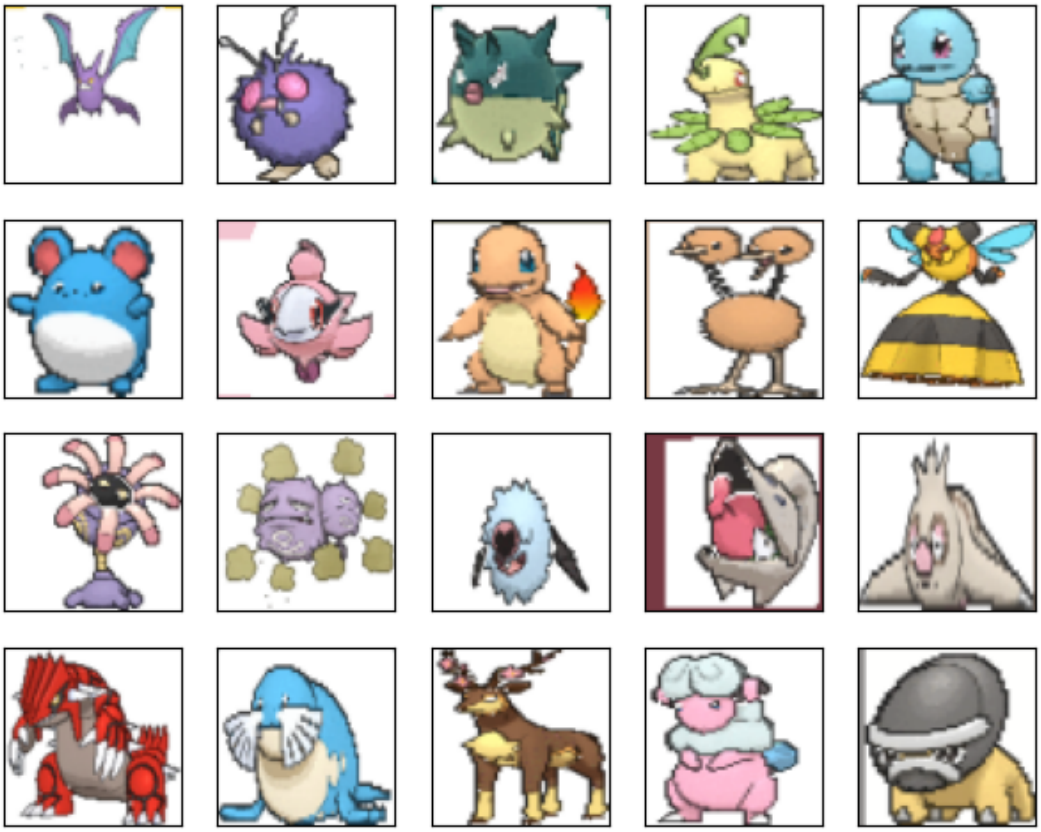
```
d2l.DATA_HUB['pokemon'] = (d2l.DATA_URL + 'pokemon.zip',
                           'c065c0e2593b8b161a2d7873e42418bf6a21106c')


def load_pokemon(resolution=64):
    """All Pokemon sprites as one (n, h, w, 3) array in [-1, 1]."""
    import warnings
    from PIL import Image
    data_dir = d2l.download_extract('pokemon')
    files = sorted(os.path.join(root, f)
                   for root, _, names in os.walk(data_dir) for f in names
                   if f.lower().endswith(('.png', '.jpg', '.jpeg')))
    def load(path):
        img = Image.open(path).convert('RGB')
        return np.asarray(img.resize((resolution, resolution), Image.BILINEAR),
                          dtype=np.float32)
    with warnings.catch_warnings():  # palette sprites with transparency
        warnings.filterwarnings('ignore', message='Palette images')
        return np.stack([load(f) for f in files]) / 127.5 - 1

images = load_pokemon()
images.shape
```

```
(40597, 64, 64, 3)
```

固定された順列によって画像の 10 分の 1 を取り分けておき、これらは訓練には一切使用しない。この画像は、生成画像をスコアリングする際の比較対象となる実サンプルであり、またクリティックが未知の画像よりも訓練画像に高いスコアを付けていないかを確かめる指標にもなる。分割は種ごとではなく画像ごとに行うため、同じ種の別バリエーションが訓練側とホールドアウト側の両方に含まれることがあり、訓練スプライトを丸写しした生成器がホールドアウト側の別バリエーションの近くに着地する可能性はある。

```
perm = np.random.default_rng(12345).permutation(len(images))
n_holdout = len(images) // 10
train_imgs = jnp.asarray(images[perm[n_holdout:]])
holdout_imgs = jnp.asarray(images[perm[:n_holdout]])
print(f'{len(train_imgs)} training and {len(holdout_imgs)} held-out images')
d2l.show_images(np.asarray(train_imgs[:20]) / 2 + 0.5,
                num_rows=4, num_cols=5);
```

```
36538 training and 4059 held-out images
```



16.5.1.2 転置畳み込みとバッチ正規化

100 チャネルの 1×1 特徴マップを 64×64 の画像へとアップサンプリングするため、DCGAN は**転置畳み込み**（32.10 節（英語版））を用いる。カーネルサイズ k、ストライド s、パディング p の転置畳み込みは、$n \times n$ の空間グリッドを次の辺長を持つ出力へと写像する。

$$s(n-1) + k - 2p. \tag{16.32}$$

$k = 4, s = 2, p = 1$ とすると、各層は空間解像度を 2 倍にする（$n \to 2n$）。最初の層（$k = 4, s = 1, p = 0$）では、1×1 の入力が 4×4 へと拡大される。逆に、$k = 4, s = 2, p = 1$ のストライド付き畳み込みは空間解像度を半分にする。

Figure 16.6 に示すように、DCGAN の生成器は 5 つの転置畳み込みを連ねて解像度を $1 \to 4 \to 8 \to 16 \to 32 \to 64$ と拡大し、各ステージでチャネル幅を半分にしながら最終的に 3 つの RGB チャネルを出力する。識別器はこの階層構造を反転させ、ストライド付き畳み込みによって単一の 1×1 ロジットへと集約する。

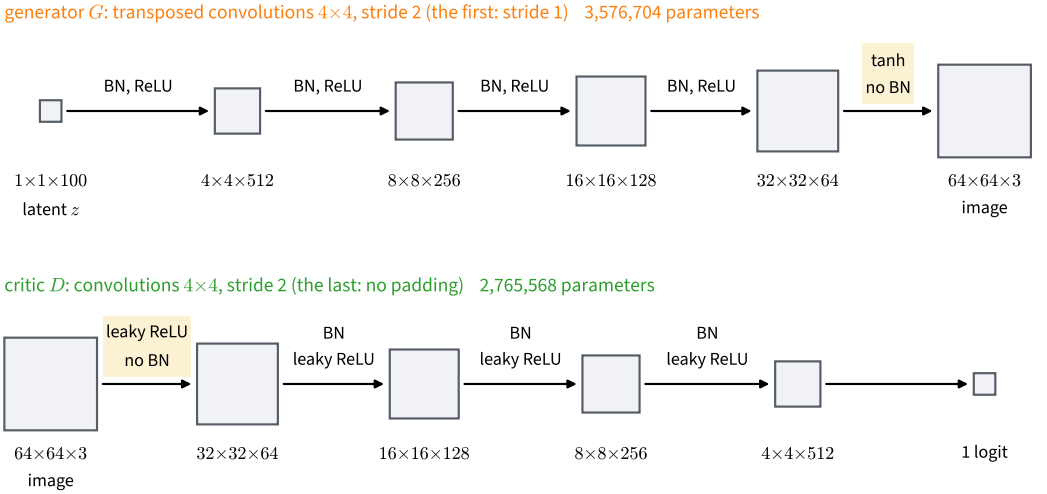


Figure 16.6: DCGAN アーキテクチャ (Radford et al. 2015)（基本チャネル幅 512、4×4 カーネルの場合）。生成器は 100 次元の潜在ベクトルを転置畳み込みによって 64×64 の RGB 画像へと写像し、クリティックはストライド付き畳み込みによってこの階層を逆にたどる。生成器の出力層とクリティックの入力ブロックを除くすべてのブロックでバッチ正規化が用いられる。

```
conv_init = nnx.initializers.normal(0.02)  # DCGAN: kernels from N(0, 0.02^2)

def bn_scale_init(key, shape, dtype=jnp.float32):
    return 1 + 0.02 * jax.random.normal(key, shape, dtype)  # N(1, 0.02^2)

def batch_norm(c, rngs):
    # Running averages decay 0.9; the variance is computed in two passes
    return nnx.BatchNorm(c, momentum=0.9, use_fast_variance=False,
                         scale_init=bn_scale_init, rngs=rngs)

def G_block(c_in, c_out, rngs, stride=2, padding='SAME'):
    """Transposed convolution (kernel 4), batch norm, ReLU."""
    return nnx.Sequential(
        nnx.ConvTranspose(c_in, c_out, (4, 4), strides=(stride, stride),
                          padding=padding, use_bias=False,
                          kernel_init=conv_init, rngs=rngs),
        batch_norm(c_out, rngs), nnx.relu)

def D_block(c_in, c_out, rngs, norm=True):
    """Strided convolution (kernel 4), batch norm, leaky ReLU."""
    return nnx.Sequential(
        nnx.Conv(c_in, c_out, (4, 4), strides=(2, 2), padding='SAME',
                 use_bias=False, kernel_init=conv_init, rngs=rngs),
```

```
        batch_norm(c_out, rngs) if norm else (lambda x: x),
        lambda x: nnx.leaky_relu(x, 0.2))
```

4 段の G_block に最終段の転置畳み込みと tanh を重ねて net_G を構成し、4 段の D_block（最初のブロックではバッチ正規化を省く）と 4×4 畳み込みによって net_D を構成する。2 個の 100 次元潜在ベクトルに対する順伝播で、それぞれの出力形状を確認しよう。

```
rngs = nnx.Rngs(0)
net_G = nnx.Sequential(                                    # (100,)
    lambda z: z.reshape(-1, 1, 1, 100),                    # (1, 1, 100)
    G_block(100, 512, rngs, stride=1, padding='VALID'),    # (4, 4, 512)
    G_block(512, 256, rngs), G_block(256, 128, rngs),
    G_block(128, 64, rngs),                                # (32, 32, 64)
    nnx.ConvTranspose(64, 3, (4, 4), strides=(2, 2), padding='SAME',
                      use_bias=False, kernel_init=conv_init, rngs=rngs),
    nnx.tanh)                                              # (64, 64, 3)
net_D = nnx.Sequential(                                    # (64, 64, 3)
    D_block(3, 64, rngs, norm=False),                      # (32, 32, 64)
    D_block(64, 128, rngs), D_block(128, 256, rngs),
    D_block(256, 512, rngs),                               # (4, 4, 512)
    nnx.Conv(512, 1, (4, 4), padding='VALID', use_bias=False,
             kernel_init=conv_init, rngs=rngs))            # (1, 1, 1)
X = net_G(jnp.zeros((2, 100)))
print(X.shape, net_D(X).shape)
```

```
(2, 64, 64, 3) (2, 1, 1, 1)
```

16.5.1.3 古典的損失による訓練

クリティックのロジスティック回帰ステップと生成器の非飽和ステップを交互に繰り返す d2l.train_gan（16.1 節）のループは、画像にもそのまま適用できる。ここでは DCGAN の設定に従いつつ、ミニバッチサイズを元論文の 128 枚より大きい 256 枚とし、学習率 $2 \cdot 10^{-4}$、$\beta_1 = 0.5$、畳み込みカーネルの初期値を $\mathcal{N}(0, 0.02^2)$ からの抽出とする。また、著者らの公開コードと同様に、バッチ正規化のスケールは $\mathcal{N}(1, 0.02^2)$ から、オフセットは 0 から開始する。

```
# The training images from the host array: the loader keeps them there
data_iter = d2l.ArrayLoader((images[perm[n_holdout:]],), 256, shuffle=True,
                            drop_last=True, key=jax.random.key(3))
```

サンプリングには、生成器の最終ステップの重みではなく直近の重みの平均を用いる。EMA はパラメータの指数移動平均（exponential moving average）を保持し、各ステップの後に次の式で更新する。

$$\bar{\theta} \leftarrow \beta \bar{\theta} + (1 - \beta)\, \theta, \qquad \beta = 2^{-1/n_{1/2}}, \tag{16.33}$$

これにより、過去のステップの重みは**半減期**（half-life）である $n_{1/2}$ ステップごとに半減する。ゲームの反復解は均衡点の周りを回転するため（16.4 節）、個々の反復解よりもその平均のほうが均衡点の近くに位置する。なお、バッチ正規化の統計量はパラメータではなく、移動平均モデルは訓練中のネットワークから統計量をそのままコピーする。しかし、コピーされた統計量は平均化された重みではなく訓練中の重みに対応しているため、訓練終了後に再推定する。

```
class EMA:
    """Exponential moving average of a model's parameters, given as a
    half-life in steps. Other state such as batch-norm statistics is copied;
    the average is kept in evaluation mode."""
    def __init__(self, model, half_life):
        self.model = nnx.clone(model)
        self.model.eval()
        self.decay = 0.5 ** (1 / half_life)

    def update(self, model):
        self.average(self.model, model, self.decay)

    @staticmethod
    @nnx.jit
    def average(avg, model, decay):
        """Move the parameters of avg toward those of model, in place."""
        params = jax.tree.map(lambda a, p: decay * a + (1 - decay) * p,
                              nnx.state(avg, nnx.Param),
                              nnx.state(model, nnx.Param))
        nnx.update(avg, params, nnx.state(model, nnx.Not(nnx.Param)))
```

本節の後半の実験では 32,000 枚の画像にわたって平均をとり、これはバッチサイズ 64 では 500 ステップの半減期に相当する。バッチサイズ 256 では、同じ 32,000 枚の画像は 125 ステップに相当し、今回の訓練全体の 2,840 ステップ（20 エポック、各 142 ステップ）の 20 分の 1 未満である。

出力されるクリティック損失は 16.1 節における 2 つのバッチ平均の和であり、均衡点では $2\log 2 \approx 1.39$ に等しく、そこでの生成器の損失は $\log 2 \approx 0.69$ に等しくなる。

```
ema = EMA(net_G, half_life=125)  # 32,000 images, as in the later runs
history = d2l.train_gan(net_D, net_G, data_iter, num_epochs=20,
                        latent_dim=100, lr_D=0.0002, lr_G=0.0002, seed=1,
                        ema=ema)
loss_D, loss_G = history[-1]
print(f'last epoch: loss_D {loss_D:.3f}, loss_G {loss_G:.3f}')
```

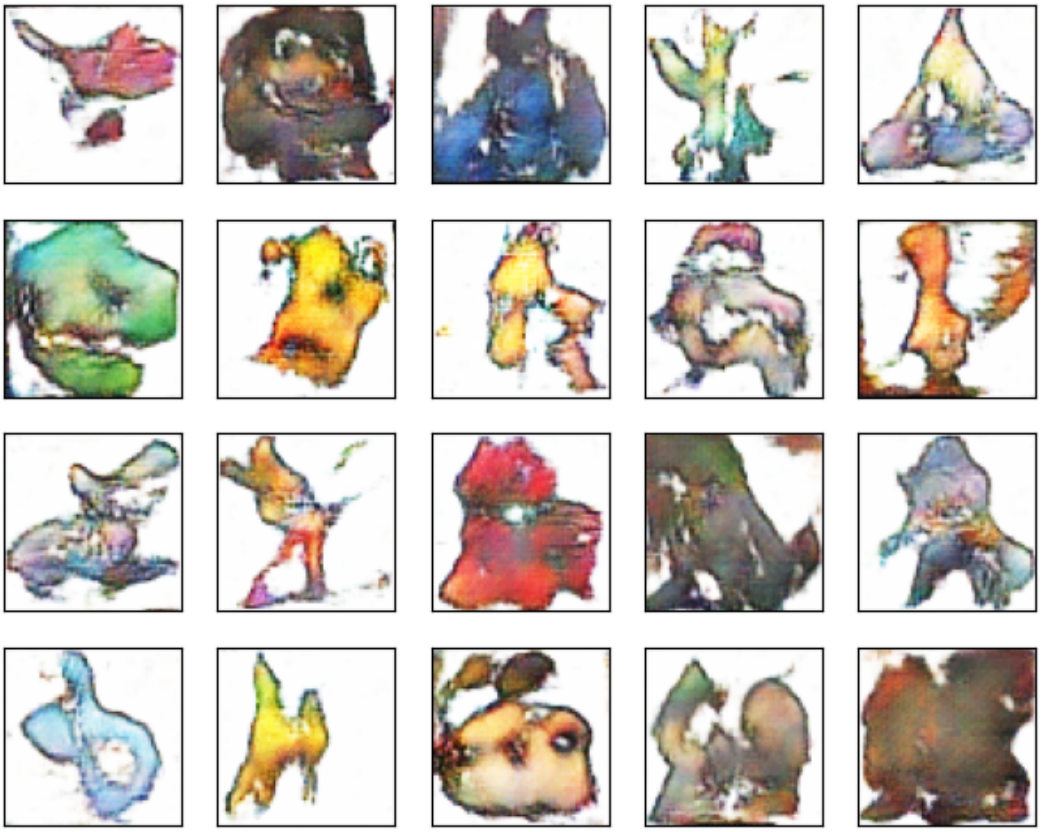
```
last epoch: loss_D 0.427, loss_G 3.027
```

次のセルでは、平均化された生成器を訓練モードにして新規の潜在コード 100 バッチを入力し、移動平均モデルのバッチ正規化統計量を再推定する。移動平均の統計量は 1 バッチあたり 0.1 の割合で過去の値を忘却するため、コピーされた値に残る重みは $0.9^{100} \approx 3 \cdot 10^{-5}$、すなわち実質的に 0 となる。

```
ema.model.train()  # batch-norm statistics for the averaged weights
# 100 batches; running averages forget at rate 0.1
for key in jax.random.split(jax.random.key(4), 100):
    ema.model(jax.random.normal(key, (256, 100)))
ema.model.eval()
```

各画像が自身の潜在コードのみに依存するよう、サンプリングは評価モードで行う。

```
fake = ema.model(jax.random.normal(jax.random.key(2), (20, 100)))
d2l.show_images(np.asarray(fake) / 2 + 0.5, num_rows=4, num_cols=5);
```



生成されたスプライトは、やや斑状のテクスチャが見られるものの、白背景の上に多様なシルエットと配色を示している。なお、識別器の損失（~ 0.25–0.45）は $2\log 2 \approx 1.39$ を大きく下回ったままであるが、モード崩壊（mode collapse）は引き起こしていない。

16.5.2 正規化なしバックボーン

DCGAN 以降の数年間で、大規模な画像 GAN には数多くのアーキテクチャ上の安定化手法が積み重ねられていった（StyleGAN2 (Karras, Laine, et al. 2020) は 10 種類の個別技術を組み合わせていた）。これに対し R3GAN (Huang et al. 2024) は、目的関数を $R_1 + R_2$ 勾配ペナルティと相対論的ペアリング損失で正則化すれば、そうした場当たり的なアーキテクチャ上の工夫を取り除けることを実証した。とりわけ、R_1 と R_2 を用いる際には識別器からバッチ正規化を省くことが不可欠である。バッチ正規化はミニバッチ内のサンプル同士を結合してしまうため、サンプルごとの入力勾配の計算が成り立たなくなるからである。

16.5.2.1 アーキテクチャとファンイン初期化

最小構成の畳み込みバックボーン（Figure 16.7）を、正規化層も出力の tanh も用いずに構築する。

- 生成器は、学習される $4\times4\times128$ の定数テンソルと z の線形射影（$4\times4\times100$、さらに 16.6 節のクラス条件付けで用いる任意の追加チャネル）を連結し、4 回（$4\to8\to16\to32\to64$）にわたって双線形補間（bilinear interpolation）とそれに続く 3×3 畳み込みおよび Leaky ReLU（slope $=0.2$）でアップサンプリングする。双線形アップサンプリングの後に 3×3 畳み込みを適用することで、転置畳み込みの不均一な重なりによって生じるチェッカーボードアーティファクト（Figure 16.8）を回避できる (Odena et al. 2016)。
- クリティックはこの構造を反転させ、3×3 畳み込み、Leaky ReLU、および双線形ダウンサンプリングによって 4×4 の特徴マップ $\phi(x)$ まで縮小し、最後に線形ヘッドを通す。
- バッチ正規化なしで各層にわたって活性化値の分散を保つため、すべての重みには He のファンイン初期化 (He et al. 2015) を傾き 0.2 の Leaky ReLU 向けにスケーリングして用いる（分散は $2/((1+0.2^2)\,n_{\text{in}})$）。

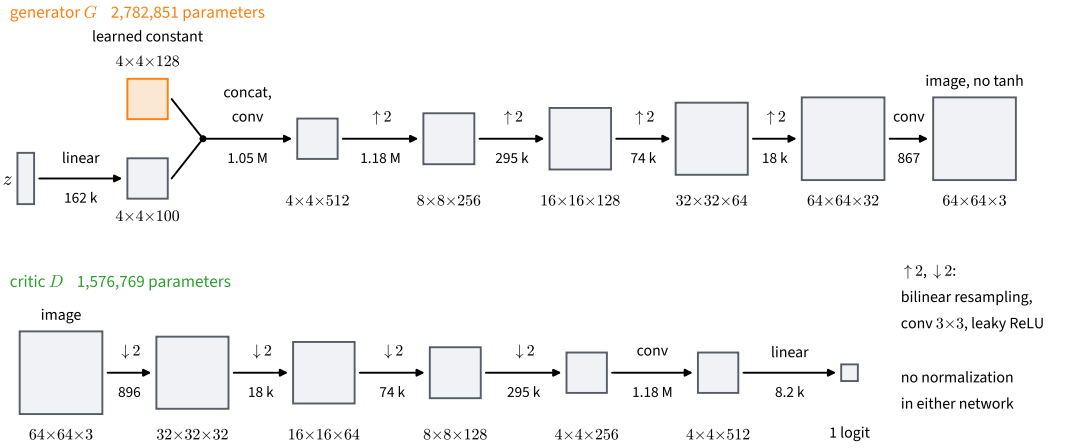


Figure 16.7: 正規化なし畳み込みバックボーン。生成器は学習される $4\times4\times128$ の定数と射影された潜在ベクトルを組み合わせ、双線形アップサンプリング、3×3 畳み込み、Leaky ReLU からなる 4 つのステージを適用する。クリティックは双線形ダウンサンプリングによってこの階層構造を逆にたどり、線形ヘッドへとつなぐ。

```
# Fan-in initialization with the leaky-ReLU gain (normal, not truncated)
fan_in_init = nnx.initializers.variance_scaling(2 / (1 + 0.2 ** 2),
                                                'fan_in', 'normal')


class Generator(nnx.Module):
    """Learned 4x4 constant + projected latent (+ extra 4x4 input maps),
    bilinearly upsampled to resolution x resolution; no normalization."""
    def __init__(self, latent_dim, resolution, base_ch, *, extra_ch=0,
                 const_ch=128, rngs):
        self.latent_dim = latent_dim
        self.const = nnx.Param(
            jax.random.normal(rngs.params(), (1, 4, 4, const_ch)))
        self.z_proj = nnx.Linear(latent_dim, 16 * latent_dim,
                                 kernel_init=fan_in_init, rngs=rngs)
        self.mix = nnx.Conv(const_ch + latent_dim + extra_ch, base_ch,
                            (3, 3), padding='SAME', kernel_init=fan_in_init,
```

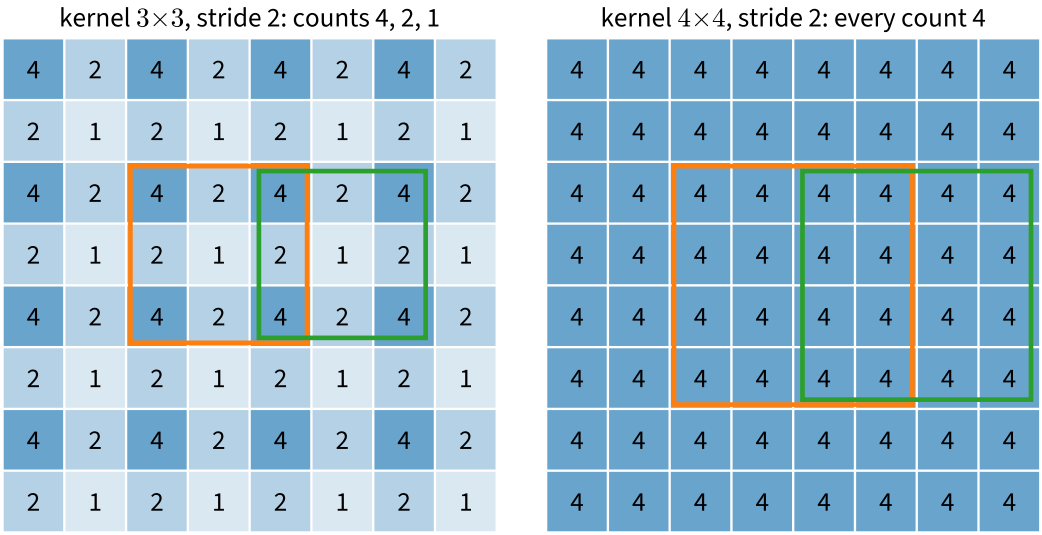


Figure 16.8: ストライド 2 の転置畳み込みにおける重なり回数。各入力ピクセルはカーネルのコピーを 1 つ出力上に押し当て、各数値は（境界から離れた位置で）その出力ピクセルを覆うカーネルの個数を表す。隣接する 2 つのカーネルの範囲を枠線で示している。3×3 カーネル（左）では重なり回数が 4、2、1 の間で交互に変わるため、定数の重みであってもチェッカーボード模様が描かれる。一方、サイズがストライドの倍数である 4×4 カーネル（右）では、すべてのピクセルが 4 回ずつ覆われる。

```
                        rngs=rngs)
        num_stages = int(np.log2(resolution / 4))       # 64: four doublings
        chans = [base_ch // 2 ** i for i in range(num_stages + 1)]
        self.stages = nnx.List(
            [nnx.Conv(c_in, c_out, (3, 3), padding='SAME',
                      kernel_init=fan_in_init, rngs=rngs)
             for c_in, c_out in zip(chans[:-1], chans[1:])])
        self.to_rgb = nnx.Conv(chans[-1], 3, (3, 3), padding='SAME',
                               kernel_init=fan_in_init, rngs=rngs)

    def __call__(self, z, extra=None):
        zc = self.z_proj(z).reshape(-1, 4, 4, self.latent_dim)
        const = jnp.broadcast_to(self.const[...],
                                 (z.shape[0],) + self.const.shape[1:])
        maps = [const, zc] + ([] if extra is None else [extra])
        x = nnx.leaky_relu(self.mix(jnp.concatenate(maps, axis=-1)), 0.2)
        for conv in self.stages:                        # 4 -> 8 -> ... -> 64
            b, h, w, c = x.shape
            x = jax.image.resize(x, (b, 2 * h, 2 * w, c), method='bilinear')
            x = nnx.leaky_relu(conv(x), 0.2)
        return self.to_rgb(x)                           # no tanh: raw output
```

クリティックは生成器と対称な構造を持つ。畳み込み、Leaky ReLU、双線形ダウンサンプリングからなる 4 つのステージで 64×64 の画像を 4×4 へと縮小し、混合用の畳み込みと線形ヘッドによって 1 つのスコアを返す。

```
class Discriminator(nnx.Module):
    """Critic that mirrors the generator: conv + leaky ReLU + bilinear
    downsampling to a 4x4 feature map phi(x), then a linear head on the
    flattened map."""
    def __init__(self, resolution, base_ch, *, rngs):
        num_stages = int(np.log2(resolution / 4))
        chans = [3] + [base_ch * 2 ** i for i in range(num_stages)]
        self.stages = nnx.List(
            [nnx.Conv(c_in, c_out, (3, 3), padding='SAME',
                      kernel_init=fan_in_init, rngs=rngs)
             for c_in, c_out in zip(chans[:-1], chans[1:])])
        self.mix = nnx.Conv(chans[-1], 2 * chans[-1], (3, 3),
                            padding='SAME', kernel_init=fan_in_init,
                            rngs=rngs)
        self.head = nnx.Linear(2 * chans[-1] * 4 * 4, 1,
                               kernel_init=fan_in_init, rngs=rngs)

    def phi(self, x):
        for conv in self.stages:                    # 64 -> 32 -> ... -> 4
            x = nnx.leaky_relu(conv(x), 0.2)
            b, h, w, c = x.shape
            x = jax.image.resize(x, (b, h // 2, w // 2, c),
                                 method='bilinear')
        return nnx.leaky_relu(self.mix(x), 0.2)

    def __call__(self, x):
        return self.head(self.phi(x).reshape(x.shape[0], -1))

for net in (Generator(100, 64, 512, rngs=nnx.Rngs(0)),
            Discriminator(64, 32, rngs=nnx.Rngs(0))):
    print(f'{type(net).__name__}: {d2l.num_params(net)} parameters')
```

```
Generator: 2782851 parameters
Discriminator: 1576769 parameters
```

パラメータ数は、生成器が ~ 2.8M、クリティックが ~ 1.6M である。

16.5.3 統制された損失の比較

16.5.3.1 3 つの実験群

正規化なしバックボーンを同一の初期重みから 3 回訓練する。Adam（$\beta_1 = 0, \beta_2 = 0.99, \mathrm{lr} = 2 \times 10^{-4}$）、バッチサイズ 64、ランダム水平反転、および EMA（half-life $= 500$ ステップ）を共通に用い、次の訓練目的関数のみを変化させる。

$$L(D) + \frac{\gamma}{2}\Big(\mathbb{E}_x\big[\|\nabla_x D(x)\|^2\big] + \mathbb{E}_{x'}\big[\|\nabla_{x'} D(x')\|^2\big]\Big), \tag{16.34}$$

ここで、$L(D)$ は古典的ロジスティック損失（`d2l.gan_loss_D`）または相対論的ペアリング損失（`d2l.rpgan_loss_D`）のいずれかであり、生成器はそれに対応する非飽和損失を用いる。各実験群の設定は次のとおりである。

実験群	クリティック損失	勾配ペナルティ	ステップ数
NS	古典的ロジスティック	$\gamma = 0$（なし）	6,000
NS + R1 + R2	古典的ロジスティック	$\gamma = 10$	12,000
RpGAN + R1 + R2	相対論的ペアリング	$\gamma = 10$	12,000

最初の 2 つの実験群を比較することで $R_1 + R_2$ 勾配ペナルティの効果が切り分けられ（ペナルティなしの実験群は 6,000 ステップよりかなり前に崩壊するため、6,000 ステップで打ち切る）、2 番目と 3 番目の実験群を比較することでペアワイズランキングの効果が切り分けられる（演習問題 1 では、画像解像度に対する γ のスケーリングを解析する）。訓練ループは保存用の 2 つの補助関数で支える。すなわち、`sample_real` は水平反転を伴うランダムなミニバッチを抽出し、`pair_stats` はペアワイズランキング正解率 $\widehat{P}(D(x) > D(x'))$ と平均の相対論的ランク重み $w_{\mathrm{Rp}} = \mathbb{E}[\sigma(D(x') - D(x))]$ を計算する。

```
def sample_real(key, images, n, labels=None):
    """Draw a training batch with random horizontal flips; return it with
    its labels (None without labels)."""
    k1, k2 = jax.random.split(key)
    idx = jax.random.randint(k1, (n,), 0, len(images))
    batch = images[idx]
    flip = jax.random.bernoulli(k2, 0.5, (n,))
    batch = jnp.where(flip[:, None, None, None], batch[:, :, ::-1, :], batch)
    return batch, (None if labels is None else labels[idx])


def pair_stats(net_D, real, fake, y=None):
    """Ranking accuracy P(D(x) > D(x')) and mean rank weight
    E[sigma(D(x') - D(x))] over aligned (real, generated) pairs; given
    labels y, the critic is D(., y)."""
    critic = net_D if y is None else lambda x: net_D(x, y)
    diff = critic(fake) - critic(real)
    return float((diff < 0).mean()), float(jax.nn.sigmoid(diff).mean())
```

JAX では、まず `r3gan_step` をコンパイルし、`d2l.critic_step` と `d2l.generator_step` を訓練反復あたり 1 回の JIT コンパイル済み呼び出しへとまとめる。

```
@nnx.jit(static_argnums=(4, 5))
def r3gan_step(net_G, net_D, opt_G, opt_D, loss_D, loss_G, gamma,
               X, z, y, X2, z2, y2):
    """One critic step on X against G(z), one generator step on (X2, z2);
    generated images take the labels y, y2 (None: unconditional). Return
    the critic loss, R1, R2 and the generator loss."""
    fake = net_G(z) if y is None else net_G(z, y)
    loss_d, r1, r2 = d2l.critic_step(net_D, opt_D, loss_D, X, fake, gamma, y)
    loss_g = d2l.generator_step(net_G, opt_G, loss_G, net_D, X2, z2, y2)
    return loss_d, r1, r2, loss_g
```

保存されるループ `train_r3gan` は、反転された実画像バッチ上で `d2l.critic_step` と `d2l.generator_step`（16.4 節）を交互に実行し、各ステップの後に EMA 生成器を更新する。クリティックのステップは R_1 と R_2 のバッチ平均を返すため、実画像に対するクリティックの急峻さ $\mathbb{E}_x\|\nabla_x D(x)\|^2$ はすべての実験群で記録される（最初の実験群では重み $\gamma = 0$）。さらに

250 ステップごとに、新規の 256 ペアに対して pair_stats を適用する。

```
def train_r3gan(net_G, net_D, loss_D, loss_G, gamma, num_steps, images,
                labels=None, *, eval_images=None, eval_labels=None,
                eval_size=256, batch_size=64, lr=2e-4, half_life=500,
                log_every=250, seed=None):
    """Train with Adam (beta_1 = 0), alternating critic_step and
    generator_step on flipped real batches; a generated batch takes the
    labels of its real batch, if any. Every log_every steps, log the losses,
    R1, R2 and pair_stats on fresh pairs from eval_images (default: images).
    Return the EMA of the generator and the history, a dict of arrays."""
    key = d2l.get_key() if seed is None else jax.random.key(seed)
    if eval_images is None:
        eval_images, eval_labels = images, labels
    opt_G, opt_D = [nnx.Optimizer(net, optax.adam(lr, b1=0.0, b2=0.99),
                                  wrt=nnx.Param) for net in (net_G, net_D)]
    noise = lambda key, n: jax.random.normal(key, (n, net_G.latent_dim))
    generate = lambda z, y: net_G(z) if y is None else net_G(z, y)
    ema = EMA(net_G, half_life)
    history = {k: [] for k in ('step', 'loss_D', 'loss_G', 'R1', 'R2',
                               'accuracy', 'rank_weight')}
    for step in range(1, num_steps + 1):
        key, kx, kz, kx2, kz2 = jax.random.split(key, 5)
        X, y = sample_real(kx, images, batch_size, labels)
        X2, y2 = sample_real(kx2, images, batch_size, labels)
        loss_d, r1, r2, loss_g = r3gan_step(
            net_G, net_D, opt_G, opt_D, loss_D, loss_G, gamma,
            X, noise(kz, batch_size), y, X2, noise(kz2, batch_size), y2)
        ema.update(net_G)
        if step % log_every == 0:  # fresh pairs of real and generated images
            key, kx, kz = jax.random.split(key, 3)
            X, y = sample_real(kx, eval_images, eval_size, eval_labels)
            fake = generate(noise(kz, eval_size), y)
            acc, weight = pair_stats(net_D, X, fake, y)
            logs = (step, loss_d, loss_g, r1, r2, acc, weight)
            for k, v in zip(history, logs):
                history[k].append(float(v))
    return ema.model, {k: np.array(v) for k, v in history.items()}
```

train_arm は、すべての実験群で同じシードから 2 つのネットワークを構築して訓練し、クリティックの訓練・ホールドアウトギャップ（1,024 枚の訓練画像に対する平均スコアと 1,024 枚のホールドアウト画像に対する平均スコアの差）を返す。

```
def train_arm(loss_D, loss_G, gamma, num_steps):
    rngs = nnx.Rngs(0)  # the same initialization in every arm
    net_G = Generator(100, 64, 512, rngs=rngs)
    net_D = Discriminator(64, 32, rngs=rngs)
    ema_G, history = train_r3gan(net_G, net_D, loss_D, loss_G, gamma,
                                 num_steps, train_imgs, seed=1)
    gap = (net_D(train_imgs[:1024]).mean()
           - net_D(holdout_imgs[:1024]).mean())
    return ema_G, history, float(gap)
```

ペナルティ付きのステップは、クリティックの入力勾配をもう一度微分するため、ペナルティなしのステップよりも計算コストがかかる。各実験の終了後に、最後の 2,000 ステップにおける 8 回の記録にわたって平均したクリティックの損失とペナルティ、および訓練・ホールドアウトギャップを表示する。

```
arms = {'NS': (d2l.gan_loss_D, d2l.gan_loss_G, 0, 6000),
        'NS + R1 + R2': (d2l.gan_loss_D, d2l.gan_loss_G, 10, 12000),
        'RpGAN + R1 + R2': (d2l.rpgan_loss_D, d2l.rpgan_loss_G, 10, 12000)}
runs = {}
for name, (loss_D, loss_G, gamma, num_steps) in arms.items():
    ema_G, hist, gap = train_arm(loss_D, loss_G, gamma, num_steps)
    runs[name] = (ema_G, hist)
    loss, r1, r2 = (hist[k][-2000 // 250:].mean()  # the last 2,000 steps
                    for k in ('loss_D', 'R1', 'R2'))
    print(f'{name:16s} critic loss {loss:.2f}, '
          f'penalty {gamma / 2 * (r1 + r2):.3f}, train-holdout gap {gap:+.2f}')
```

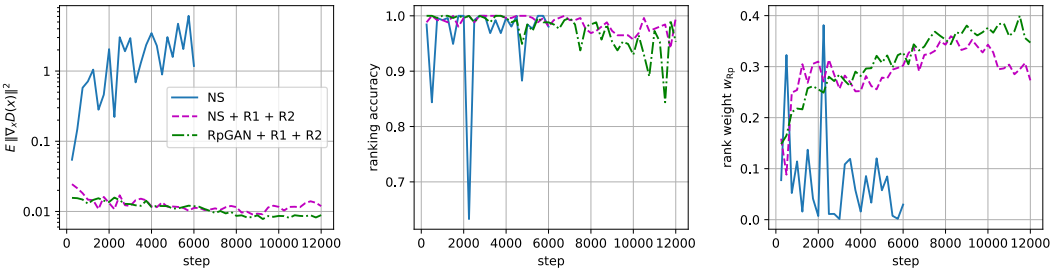
```
NS               critic loss 0.58, penalty 0.000, train-holdout gap +0.29
NS + R1 + R2     critic loss 1.04, penalty 0.122, train-holdout gap +0.05
RpGAN + R1 + R2  critic loss 0.47, penalty 0.092, train-holdout gap +0.03
```

ペナルティ付きの 2 つのクリティックはいずれも均衡値をやや下回る水準に落ち着くのに対し（古典的ロジスティックでは 1.04 対 $2\log 2 \approx 1.39$、ペアリングでは 0.47–0.51 対 $\log 2 \approx 0.69$）、ペナルティなしのクリティックは 0.57–0.58 まで低下する。

16.5.3.2 訓練の推移と生成サンプル

クリティックの二乗入力勾配 $\mathbb{E}_x[\|\nabla_x D(x)\|^2]$、ペアワイズランキング正解率 $\hat{P}(D(x) > D(x'))$、および相対論的ランク重み $w_{\mathrm{Rp}} = \mathbb{E}[\sigma(D(x') - D(x))]$ を 250 ステップごとにプロットすると、勾配ペナルティがどのように訓練を安定化させるかがわかる。

```
fig, axes = d2l.plt.subplots(1, 3, figsize=(12, 3.2))
hists = [hist for _, hist in runs.values()]
for ax, key, ylabel, yscale in zip(
        axes, ('R1', 'accuracy', 'rank_weight'),
        (r'$E\,\|\nabla_x D(x)\|^2$', 'ranking accuracy',
         r'rank weight $w_{\mathrm{Rp}}$'),
        ('log', 'linear', 'linear')):
    d2l.plot([h['step'] for h in hists], [h[key] for h in hists], 'step',
             ylabel, yscale=yscale, axes=ax)
axes[0].legend(list(runs), loc='center right')
fig.tight_layout()
```



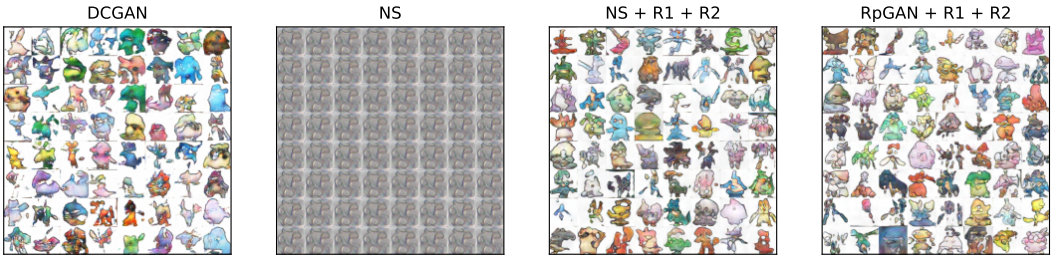
$R_1 + R_2$ がない場合、クリティックの二乗勾配ノルムは 2 桁上昇する（1–8 に達するのに対し、ペナルティ付きクリティックでは ~ 0.01）。3 つのクリティックはいずれもペアの $> 90\%$ を正しく順序付けるが、ペナルティ付きクリティックは小さなスコアマージンを維持するのに対し（$w_{\text{Rp}} \in [0.25, 0.40]$、ロジット差の中央値で 0.7 前後に相当）、ペナルティなしクリティックは w_{Rp} を 0.15 未満へと押し下げる。

サンプル品質を視覚的に比較するため、まず $[-1, 1]$ の画像バッチを単一の $[0, 1]$ グリッドに並べる補助関数 `tile_images` を定義する。

```
def tile_images(imgs, num_rows):
    """Tile images (n, h, w, 3) in [-1, 1] into one (rows * h, cols * w, 3)
    array in [0, 1]."""
    n, h, w, c = imgs.shape
    grid = (jnp.clip(imgs, -1, 1) + 1) / 2
    grid = grid.reshape(num_rows, n // num_rows, h, w, c)
    grid = grid.transpose(0, 2, 1, 3, 4)  # (rows, h, cols, w, c)
    return np.asarray(grid.reshape(num_rows * h, -1, c))
```

それでは、4 つの EMA 生成器すべて（DCGAN と正規化なしの 3 つの実験群）を同じ 64 個の潜在ベクトル `z_show` 上で評価し、その出力を 8×8 のグリッドに並べてみよう。

```
z_show = jax.random.normal(jax.random.key(42), (64, 100))
gens = {'DCGAN': ema.model,
        **{name: ema_G for name, (ema_G, _) in runs.items()}}
grids = [tile_images(G(z_show), 8) for G in gens.values()]
d2l.show_images(grids, 1, 4, titles=list(gens), scale=3.4);
```



バッチ正規化も勾配ペナルティもない場合、`NS` 実験群は完全なモード崩壊を起こす（64 個すべての潜在コードにわたって一様な灰色のテクスチャを生成する。演習問題 4 を参照）。一方、$R_1 + R_2$ を加えることで（`NS + R1 + R2` および `RpGAN + R1 + R2`）崩壊が防がれ、きれいな白背景の中央に多様なスプライトのシルエットが生成される。

16.5.4 定量的評価指標

壊滅的な崩壊は目視でも容易に見抜けるが、正常に機能している 2 つの生成器を比較するには定量的な指標が必要である。標準的な GAN の評価では、n 枚の実画像と n 枚の生成画像を事前学習済みの特徴抽出器 $\phi(x) \in \mathbb{R}^d$ で埋め込み、得られた 2 つの経験特徴量分布の間の乖離を 16.2 節の閉形式の距離によって測定する。

16.5.4.1 FID と KID

1. **Fréchet Inception Distance（FID）**（Heusel et al. 2017）は、実画像と生成画像の特徴量の点群に多変量ガウス分布 $\mathcal{N}(\mu_p, \Sigma_p)$ と $\mathcal{N}(\mu_q, \Sigma_q)$ を当てはめ、両者の間の二乗 2 次ワッサースタイン距離（Wasserstein-2 distance）を計算する（Dowson and Landau 1982; Givens and Shortt 1984）。

$$\begin{aligned}\mathrm{FID} = &\left\|\mu_p - \mu_q\right\|^2 \\ &+ \operatorname{tr}\Big(\Sigma_p + \Sigma_q - 2\big(\Sigma_p^{1/2}\,\Sigma_q\,\Sigma_p^{1/2}\big)^{1/2}\Big).\end{aligned} \tag{16.35}$$

   FID は特徴量分布の最初の 2 つのモーメントのみに依存し、そのプラグイン標本共分散推定量は有限サンプルサイズにおいて正の $O(1/n)$ の統計的バイアスを持つ（演習問題 3）。

2. **Kernel Inception Distance（KID）**（Bińkowski et al. 2018）は、3 次多項式カーネル $k(u, v) = (u^\top v / d + 1)^3$ を用いて二乗最大平均値差異（maximum mean discrepancy、MMD）の不偏推定量（`d2l.mmd2_unbiased`、16.2 節）を計算するものであり、特徴量の 3 次までのモーメントを比較する（演習問題 5）。

$$\begin{aligned}\widehat{\mathrm{MMD}}^2 = &\frac{1}{n_p(n_p-1)}\sum_{i\neq i'} k(\phi(x_i), \phi(x_{i'})) \\ &+ \frac{1}{n_q(n_q-1)}\sum_{j\neq j'} k(\phi(x'_j), \phi(x'_{j'})) \\ &- \frac{2}{n_p n_q}\sum_{i,j} k(\phi(x_i), \phi(x'_j)).\end{aligned} \tag{16.36}$$

```python
def fid_score(feat_p, feat_q):
    """Frechet distance between Gaussians fitted to two feature sets."""
    feat_p = np.asarray(feat_p, np.float64)
    feat_q = np.asarray(feat_q, np.float64)
    mu_p, mu_q = feat_p.mean(0), feat_q.mean(0)
    C_p = np.cov(feat_p, rowvar=False)
    C_q = np.cov(feat_q, rowvar=False)
    lam, U = np.linalg.eigh(C_p)
    sqrt_C_p = (U * np.sqrt(np.maximum(lam, 0))) @ U.T
    M = sqrt_C_p @ C_q @ sqrt_C_p
    cross = np.linalg.eigvalsh((M + M.T) / 2)   # symmetric up to rounding
    return (((mu_p - mu_q) ** 2).sum() + np.trace(C_p) + np.trace(C_q)
            - 2 * np.sqrt(np.maximum(cross, 0)).sum())
```

```
def kid_score(feat_p, feat_q):
    """Unbiased MMD^2 between two feature sets, cubic polynomial kernel."""
    d = feat_p.shape[1]
    kernel = lambda a, b: (a @ b.T / d + 1) ** 3
    return d2l.mmd2_unbiased(kernel(feat_p, feat_p), kernel(feat_q, feat_q),
                             kernel(feat_p, feat_q))
```

16.5.4.2 特徴ネットワーク

標準的なベンチマークでは、ImageNet で事前学習された Inception-v3 ネットワークの 2,048 次元の特徴量、あるいは自己教師ありの DINOv2 特徴量を用いて FID と KID を計算する (Stein et al. 2023; Oquab et al. 2024)。このノートブックを自己完結させるため、ここでは軽量な 3 ブロックの CIFAR-10 分類器 `d2l.FeatureCNN`（オンライン付録「ユーティリティ関数とクラス」）を 32×32 の解像度で 15 エポック訓練し、その出力直前の 128 次元層を ϕ として用いる（64×64 のスプライトは 32×32 へと 2×2 の平均プーリングでダウンサンプリングする）。

```
def batched(f, X, batch_size=1000):
    """Apply f to X in batches and concatenate."""
    return jnp.concatenate([f(X[i:i + batch_size])
                            for i in range(0, len(X), batch_size)])
```

メモリを圧迫せずに評価できるよう `batched` を用い、CIFAR-10 を読み込んで、ランダムな 4 ピクセルシフトを伴いながら `d2l.FeatureCNN` を 32×32 の解像度で訓練し、テストセットにおける分類正解率を確認する。

```
cifar_X, cifar_y, test_X, test_y = d2l.load_cifar10_arrays()
cnn = d2l.FeatureCNN(resolution=32, rngs=nnx.Rngs(1))
d2l.train_feature_cnn(cnn, cifar_X, cifar_y, shift=4)
pred = batched(cnn, test_X).argmax(1)
print(f'CIFAR-10 test accuracy: {float((pred == test_y).mean()):.3f}')
```

```
CIFAR-10 test accuracy: 0.781
```

16.5.4.3 適合率と再現率

FID や KID のような単一の距離指標は、p の台の外側に非現実的な画像を生成する**低い忠実度**（poor fidelity）と、モード欠落（mode dropping）によって p の一部を覆い損ねる**低い被覆**（poor coverage）という 2 つの異なる失敗モードを混同してしまう。Kynkäänniemi et al. (2019) は、特徴量空間における K 最近傍球を用いてこれらを分離した。各実特徴量 $\phi(x_i)$ の周りで、$B(\phi(x_i), r_i)$ をその K 番目に近い実特徴量まで届くユークリッド球とし、生成特徴量の間でも $B(\phi(x'_j), r'_j)$ を同様に定義する（ここでは $K = 3$ を用いる）。すると、次の式が得られる。

$$\text{precision} = \frac{1}{n_q} \sum_{j=1}^{n_q} \mathbb{1}\Big[\phi(x'_j) \in \bigcup\nolimits_i B(\phi(x_i), r_i)\Big],$$

$$\text{recall} = \frac{1}{n_p} \sum_{i=1}^{n_p} \mathbb{1}\Big[\phi(x_i) \in \bigcup\nolimits_j B(\phi(x'_j), r'_j)\Big].$$

適合率は推定された実特徴量の台に含まれる生成サンプルの割合であり、**再現率**は推定された生成特徴量の台によって覆われる実サンプルの割合である。生成器が単一の出力へと崩壊すると、その K-NN 半径 r'_j は 0 に縮小し、再現率は 0 になる。

```
def precision_recall(feat_real, feat_gen, k=3, chunk=1000):
    """k-NN precision and recall of two feature sets: the fraction of each
    set inside the union of the other set's balls, each ball reaching to its
    center's k-th nearest neighbor; return (precision, recall)."""
    P = np.asarray(feat_real, np.float64)
    Q = np.asarray(feat_gen, np.float64)
    def dists(A, B):  # (len(A), len(B)) Euclidean distances
        sq = (A ** 2).sum(1)[:, None] - 2 * A @ B.T + (B ** 2).sum(1)
        return np.sqrt(np.maximum(sq, 0))
    def chunks(A):
        return (A[i:i + chunk] for i in range(0, len(A), chunk))
    def radii(A):  # sorted distances start with the point itself, at 0
        return np.concatenate([np.partition(dists(a, A), k, axis=1)[:, k]
                               for a in chunks(A)])
    def covered(A, B, r_B):  # is each row of A inside some ball of B?
        return np.concatenate([(dists(a, B) <= r_B).any(1)
                               for a in chunks(A)])
    return (float(covered(Q, P, radii(P)).mean()),
            float(covered(P, Q, radii(Q)).mean()))
```

16.5.4.4 各実行のスコアリング

各生成器の移動平均モデルから抽出した $n = 4{,}059$ 個のサンプルを、4,059 枚のホールドアウトスプライトすべてに対してスコアリングする。各生成器は潜在コードの 2 回の抽出それぞれで評価するため、2 行目はサンプルのみが変わったときにスコアがどれだけ変動するかを表す。下限の目安（フロア）として、互いに素な 4,059 枚ずつの訓練画像の 2 セット同士もスコアリングする。両者の間の距離はすべて推定量のノイズであり、FD においてはさらに有限サンプルの統計的バイアスが含まれる。特徴量ネットワークは独自のものであるため、ここではスコアを FID や KID ではなく FD および MMD^2 と呼ぶ。これらは 12 倍のサンプル数を用いる既存の公表値とは直接比較できない。また、このバックボーンは出力に tanh を持たないため、スコアリング前に出力をクランプし、最後の 2 列には生のピクセル値のうち $+1$ を超える割合と -1 を下回る割合を示す。

```
features = lambda imgs: np.asarray(batched(  # 2 x 2 averages: 64 -> 32
    cnn.features, nnx.avg_pool(imgs, (2, 2), strides=(2, 2))))
n = len(holdout_imgs)
zs = [jax.random.normal(jax.random.key(seed), (n, 100))
      for seed in (7, 8)]  # a second latent draw gives the sampling range
z_score = zs[0]
feat_gen, out_of_range = {}, {}  # out_of_range: fractions above +1, below -1
for name, G in gens.items():
    for row, z in zip((name, '  second draw'), zs):
        x = batched(G, z)
        out_of_range[(name, row)] = (float((x > 1).mean()),
                                     float((x < -1).mean()))
        feat_gen[(name, row)] = features(jnp.clip(x, -1, 1))
feat_real = features(holdout_imgs)
feat_a, feat_b = features(train_imgs[:n]), features(train_imgs[n:2 * n])
```

```
white = float((holdout_imgs == 1).mean())
```

各特徴量セットをホールドアウトスプライトの特徴量セットと比較する。最初の行は下限の目安（フロア）である。

```
print(f'{"run":17s}{"FD":>7s}{"MMD^2":>8s}{"precision":>11s}{"recall":>8s}'
      f'{"> +1":>7s}{"< -1":>7s}')
scored = {(None, 'real vs. real'): (feat_a, feat_b),
          **{key: (feat_real, feat) for key, feat in feat_gen.items()}}
for key, (ref, feat) in scored.items():
    precision, recall = precision_recall(ref, feat)
    oor = ''.join(f'{v:7.3f}' for v in out_of_range.get(key, ()))
    print(f'{key[1]:17s}{fid_score(ref, feat):7.2f}{kid_score(ref, feat):8.3f}'
          f'{precision:11.2f}{recall:8.2f}{oor}')
print(f'real pixel values equal to +1: {white:.0%}')
```

```
run                   FD   MMD^2  precision  recall   > +1   < -1
real vs. real       0.17  -0.001       0.95    0.95
DCGAN               1.80   0.083       0.39    0.48  0.000  0.000
  second draw       1.84   0.090       0.37    0.50  0.000  0.000
NS                 71.97   7.502       0.00    0.00  0.001  0.000
  second draw      71.95   7.501       0.00    0.00  0.001  0.000
NS + R1 + R2        3.96   0.663       0.40    0.41  0.146  0.001
  second draw       4.06   0.687       0.41    0.42  0.144  0.001
RpGAN + R1 + R2     3.71   0.603       0.41    0.42  0.149  0.001
  second draw       3.78   0.608       0.41    0.43  0.149  0.001
real pixel values equal to +1: 45%
```

この表から、いくつかの重要な教訓が読み取れる。

1. **実画像対実画像のベースラインフロア：**互いに素な 4,059 枚ずつの実スプライトの 2 セットでは、FD ≈ 0.2（有限サンプルの共分散推定による統計的バイアスのため）、$\widehat{\mathrm{MMD}}^2 \approx 0.000$、適合率・再現率は 0.95 となる。潜在コードの 2 回目の抽出で評価しても、FD の変化は ≤ 0.2 に収まる。
2. **崩壊対ペナルティ付き訓練：**崩壊したペナルティなしの NS 実験群では、FD $\approx$ 64–72、recall $= 0.00$、precision ≤ 0.07 となる。ここに $R_1 + R_2$ を加えると、FD は 1 桁減少し（$\sim$ 3.5–5.5）、適合率は $\sim$ 0.35–0.4、再現率は ~ 0.4 へと向上する。
3. **特徴空間への感度（DCGAN 対ペナルティ付きバックボーン）：**ペナルティ付きバックボーンのほうが 64×64 では輪郭が鮮明で白背景もきれいであるにもかかわらず、2015 年の DCGAN のほうが低い FD（~ 2）および $\widehat{\mathrm{MMD}}^2$ をこの 32×32 CIFAR-10 分類器の特徴量において達成する。これは 2×2 のダウンサンプリングによって DCGAN の高周波の斑模様が平滑化されるためである。特徴空間の評価指標には、基盤となる特徴ネットワークの不変性と解像度が常に反映される。

16.5.5 潜在補間とクリティックの過学習

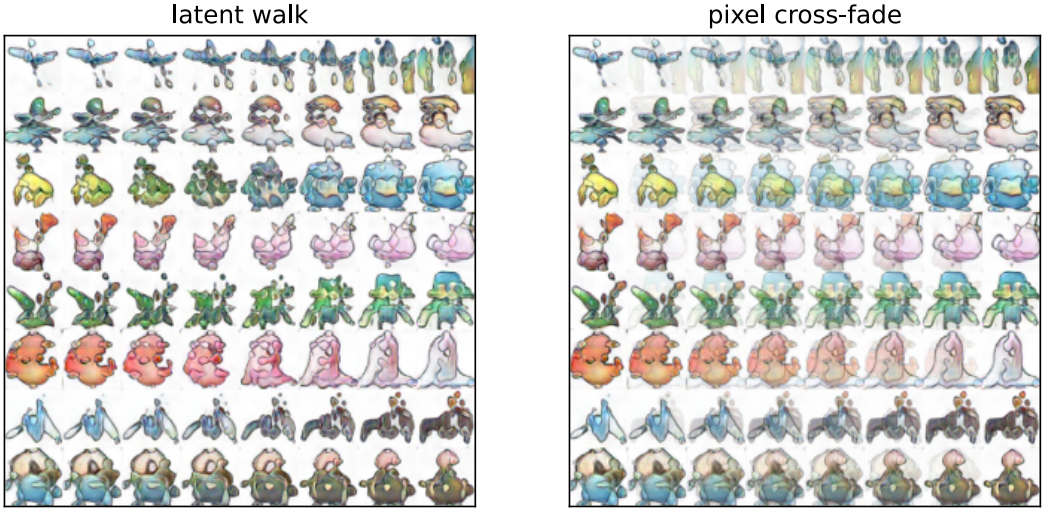
16.5.5.1 潜在空間での補間

生成器が離散的なテンプレートを暗記しているのではなく、潜在コードから画像への滑らかな写像を学習していることを確かめるため、64 組の潜在ベクトルのペアの間を $z_t = (1-t)z_0 + tz_1 (t \in [0,1])$ と線形補間し、生成画像 $G(z_t)$ をピクセルごとのクロスフェード $(1-t)G(z_0) + tG(z_1)$

と比較する。

```
G = runs['RpGAN + R1 + R2'][0]
z0, z1 = jax.random.normal(jax.random.key(0), (2, 64, 1, 100))  # 64 pairs
t = jnp.linspace(0, 1, 8).reshape(1, 8, 1)
path = jnp.clip(batched(G, ((1 - t) * z0 + t * z1).reshape(-1, 100)), -1, 1)
path = path.reshape(64, 8, 64, 64, 3)               # (pair, step, image)
t = t.reshape(1, 8, 1, 1, 1)
fade = (1 - t) * path[:, :1] + t * path[:, -1:]     # pixel cross-fades
inner = lambda x: features(x[:, 1:-1].reshape(-1, 64, 64, 3))
print(f'precision of the inner steps: latent walk '
      f'{precision_recall(feat_real, inner(path))[0]:.2f}, '
      f'cross-fade {precision_recall(feat_real, inner(fade))[0]:.2f}')
d2l.show_images([tile_images(x[:8].reshape(-1, 64, 64, 3), 8)
                 for x in (path, fade)],
                1, 2, titles=['latent walk', 'pixel cross-fade'], scale=4.5);
```

```
precision of the inner steps: latent walk 0.38, cross-fade 0.18
```



潜在空間における中間補間点は、生成器からのランダムな抽出と同等の特徴量適合率（~ 0.35–0.38）を維持しながら形状や色が滑らかに遷移するのに対し、ピクセルごとのクロスフェードでは残像を伴う二重露光のような画像が生じ、適合率は半分（~ 0.18–0.21）に低下する。

16.5.5.2 クリティックの過学習とリークしないデータ拡張

1.6M 個のパラメータと 36,538 枚の訓練スプライトのもとで、クリティックは 12,000 ステップの間に各訓練画像を約 20 回参照する。ペナルティ付きクリティックにおける訓練・ホールドアウト間のスコア差はごくわずかである（$+0.02 \sim +0.05$ であり、実画像対生成画像のマージン ~ 0.7 と比べて小さい。演習問題 2 を参照）。しかし、より小規模なデータセットやより長時間の訓練では、識別器による暗記が FID を悪化させることがある (Karras, Aittala, et al. 2020)。これに対する標準的な対策である Adaptive Discriminator Augmentation（ADA、Karras, Aittala, et al. (2020)）および DiffAugment (Zhao et al. 2020) は、勾配経路の内部で実画像と生成画像の双方に可逆かつ**リークしない**（non-leaking）データ拡張を適用する（これに対し、本節の水平反

転のように実画像のみを拡張すると、生成器は拡張後の分布に一致するように訓練される。演習問題 6 を参照)。

16.5.6 まとめ

DCGAN は、転置畳み込みとストライド付き畳み込み、バッチ正規化、小さな標準偏差による重みの初期化（$\sigma = 0.02$）、および $\beta_1 = 0.5$ の Adam によって、古典的ロジスティック損失のもとでの畳み込み敵対的訓練を安定化させる。バッチ正規化とアーキテクチャ上の安定化手法を取り除くと、ペナルティなしのロジスティック損失は完全に崩壊するが、ゼロ中心勾配ペナルティ（$R_1 + R_2$）を加えればクリティックの勾配が 2 桁減衰し、最小構成の双線形リサンプリングバックボーンを安定して訓練できる。生成器の重みの指数移動平均はミニマックス振動をさらに減衰させるが、バッチ正規化を用いる場合には移動統計量の再較正が必要である。最後に、定量的な画像評価では、フレシェ距離（Equation 16.35）、Kernel Inception Distance（Equation 16.36）、および K-NN の適合率と再現率を用いて実画像と生成画像の特徴量の点群を比較し、常に実画像対実画像の有限サンプルフロアを基準として較正する。

16.5.7 演習問題

1. **勾配ペナルティ重みと画像解像度**。$\nabla_x D(x)$ は長さの逆数の単位を持つため、ペナルティ重み γ は画像解像度とピクセル値のスケーリングに依存する。
   1. U を 64×64 の画像 x を 128×128 へアップサンプリングする作用素（各ピクセルを 2×2 ブロックに複製）とし、A を各 2×2 ブロックの平均をとる作用素として $D' = D \circ A$ が $D'(Ux) = D(x)$ を満たすとする。$\|\nabla D'(Ux)\|^2$ を $\|\nabla D(x)\|^2$ を用いて表し、ペナルティ重み γ' として 128×128 のペナルティを 64×64 のペナルティと等しくするものを求めよ。
   2. γ は、ピクセル値を $[0, 1]$ へと正規化した場合（$[-1, 1]$ の代わりに）、どのように再スケーリングしなければならないか。
   3. (a) の結果を、StyleGAN2-ADA で用いられている経験則 $\gamma \propto r^2/B$（解像度 r、バッチサイズ B）と関連づけて説明せよ (Karras, Aittala, et al. 2020)。
2. [code] **スケールに依存しないクリティックの過学習統計量**。スコアスケールの異なる目的関数の間でクリティックの暗記を比較するため、Karras, Aittala, et al. (2020) は次の量を定義している。

$$r_v = \frac{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{holdout}}]}{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{generated}}]},$$
$$r_t = \mathbb{E}\big[\operatorname{sign}(D_{\text{train}})\big].$$

   1. r_v は正のアフィン変換 $D \mapsto aD + c$（$a > 0$）に対して不変であるのに対し、生のギャップ $\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{holdout}}]$ は a 倍にスケーリングされ、r_t は平行移動 c に依存することを示せ。
   2. シフト不変なペアリングクリティックについて、D のどのような変換のもとで $\mathbb{E}[\operatorname{sign}(D_{\text{train}} - D_{\text{holdout}})]$ が不変になるか。また、訓練スコアとホールドアウトスコアが同一の分布に従うとき、この量はどの値をとるか。
   3. `train_arm` を変更して `net_D` を返すようにし、3 つの実験群それぞれについて 1,024 枚の画像にわたる r_v と $\mathbb{E}[\operatorname{sign}(D_{\text{train}} - D_{\text{holdout}})]$ を計算せよ。
3. [code] **FID と KID における有限サンプルの統計的バイアス**。

1. 互いに素な実画像対実画像の FD と $\widehat{\mathrm{MMD}}^2$ を、n 枚ずつの訓練スプライトの 2 セット間で $n \in \{250, 500, 1000, 2000, 4059\}$ の各値についてランダムな 5 分割にわたって計算せよ。
2. FD を $1/n$ に対して線形回帰で当てはめ、$1/n \to 0$ へと外挿せよ（Chong and Forsyth (2020)）。$n = 500$ および $n = 4{,}059$ における有限サンプルバイアスはどの程度の大きさか。
3. ホールドアウトスプライトに対する `RpGAN + R1 + R2` 生成器についても $1/n$ による線形当てはめを繰り返せ。$1/n$ の傾きは生成器の特徴量共分散に依存するため、小さな n における有限サンプルバイアスが 2 つの生成器のランキングを逆転させ得る理由を説明せよ。

4. [code] **重みの初期化とモード崩壊**。●サンプル多様性を、`z_show` 上の生成画像におけるペアごとの平均特徴量距離を 64 枚のホールドアウトスプライトにおけるペアごとの平均特徴量距離で割ったものとして定義する。

1. サンプル多様性と $\mathbb{E}_x[\|\nabla_x D(x)\|^2]$ を 5,000 ステップにわたり 500 ステップごとに、ペナルティなしの `NS` 実験群について、He のファンイン初期化のもとで、および $\mathcal{N}(0, \sigma^2)$（$\sigma \in \{0.01, 0.02, 0.05\}$）による初期化のもとでそれぞれ記録せよ。クリティックの勾配ノルムの急上昇は、多様性の低下に先行するか。
2. ファンイン初期化での実験を 3 つのシードにわたって繰り返し、崩壊が起きるステップ数がシードごとにどの程度変動するかを測定せよ。

5. **FID と KID が捉えるモーメント**。●

1. 3 次多項式の KID カーネル $k(u, v) = (u^\top v/d + 1)^3$ を二項定理によって展開し、母集団の $\mathrm{MMD}^2(p, q)$ が

$$\sum_{j=1}^{3} \binom{3}{j} d^{-j} \left\| \mathbb{E}_p[u^{\otimes j}] - \mathbb{E}_q[u^{\otimes j}] \right\|_F^2, \tag{16.37}$$

に等しく、したがって特徴量分布の最初の 3 つのモーメントを比較することを示せ。

2. 1 次元（$d = 1$）において、母集団の FID と 3 次カーネルの MMD^2 を、$p = \mathcal{N}(0, 1)$ と、(i) 平行移動した指数分布 $X - 1$（ただし $X \sim \mathrm{Exp}(1)$ であり、平均 0、分散 1、歪度 2）、および (ii) 対称な混合分布 $\frac{1}{2}\mathcal{N}(-a, 1 - a^2) + \frac{1}{2}\mathcal{N}(a, 1 - a^2)$（$0 < a < 1$）のそれぞれとの間で計算せよ。

6. **リークしないデータ拡張**。●F を水平反転作用素とし、$T_\rho r = (1 - \rho) r + \rho\, F_\# r$ を r からのサンプルを確率 ρ で反転させて得られる分布とする。

1. `sample_real` のように実画像のみを $\rho = \frac{1}{2}$ で反転させた場合、生成器は $T_{1/2} p$ に向かって訓練される。真のスプライトのうち割合 α が左向きで対応する右向き個体を持たないとき、最適な生成器からのサンプルのうちどれだけの割合が右向きになるか。
2. クリティックの内部で実画像と生成画像の双方に T_ρ を適用するとき、$F^2 = I$ を用いて、$T_\rho q = T_\rho p \implies q = p$ が $\rho \neq \frac{1}{2}$ のときに常に成り立つこと（**リークしない**性質、Karras, Aittala, et al. (2020)）、一方で $\rho = \frac{1}{2}$ のときには p と同じ反転対称化密度を持つ任意の分布が p と識別不能になることを示せ。

16.6 条件付き生成

16.1 節と 16.5 節の生成器は**無条件**（unconditional）である。すなわち、周辺データ分布 $p(x)$ からランダムにサンプルを抽出するだけで、目的のクラスやテキストプロンプト、入力画像を指定する手段を持たない。最初の GAN 論文の発表から間もなく、Mirza and Osindero (2014) は条件付け変数（conditioning variable）c を生成器 $G(z, c)$ と識別器 $D(x, c)$ の両方に入力する**条件付き GAN**（conditional GAN、cGAN）を導入した。

本節では、次の 3 つの問いに取り組む。

1. 条件付き定式化はどのような目的関数を最適化し、その最適クリティックは何を推定するのか。
2. 条件付け変数 c を生成器と識別器のアーキテクチャにどのように組み込むべきか。
3. 条件付き生成器を、条件アライメント（condition alignment）、サンプル忠実度（sample fidelity）、クラス内多様性（within-class diversity）のそれぞれについて個別に評価するにはどうすればよいか。

これらの概念を具体的に示すため、CIFAR-10 の 10 クラス上でクラス条件付き生成器を訓練し、評価する。

16.6.1 条件付き目的関数

16.6.1.1 最適条件付きクリティック

データと条件のペア $(x, c) \sim p(x, c)$ が同時データ分布から抽出されるものとする。ここで、c は離散的なクラスラベル、テキスト埋め込み、あるいは画像である。条件の周辺分布 $p(c)$ を一致させるため、実データから $c \sim p(c)$ をサンプリングしたうえで $x' = G(z, c) \sim q(\cdot \mid c)$ を生成すると、生成側の同時分布 $q(x, c) = p(c)\, q(x \mid c)$ が定まる。条件付き対数損失の目的関数は、古典的な二値対数損失の定式化（16.1 節）をペア (x, c) に適用したものである。

$$
\begin{aligned}
V(D) = {} & \mathbb{E}_{(x,c)\sim p}\big[\log \sigma(D(x, c))\big] \\
& + \mathbb{E}_{c\sim p(c),\, x'\sim q(\cdot|c)}\big[\log \sigma(-D(x', c))\big].
\end{aligned}
\tag{16.38}
$$

$D(x, c)$ に関する各点ごとの最適化は、直積空間 $\mathcal{X} \times \mathcal{C}$ 上での無条件の場合の導出と同一であるため、最適条件付きクリティックは同時密度の対数比となり、共通の周辺分布 $p(c)$ が約分されて消える。

$$
D^\star(x, c) = \log \frac{p(x, c)}{q(x, c)} = \log \frac{p(c)\, p(x \mid c)}{p(c)\, q(x \mid c)} = \log \frac{p(x \mid c)}{q(x \mid c)}. \tag{16.39}
$$

したがって、各条件 c に対して最適クリティックは、条件付き分布 $p(\cdot \mid c)$ と $q(\cdot \mid c)$ の対数密度比を計算する。$D^\star(x, c)$ を Equation 16.38 に代入すると、期待条件付きイェンセン・シャノン・ダイバージェンス（Jensen–Shannon divergence）が得られる。

$$
\max_D V(D) = \mathbb{E}_{c\sim p(c)}\Big[\, 2\,\mathrm{JS}\big(p(\cdot \mid c),\, q(\cdot \mid c)\big)\Big] - 2\log 2. \tag{16.40}
$$

この式が大域的最小値 $-2\log 2$ に到達するのは、$q(\cdot \mid c) = p(\cdot \mid c)$ がすべての条件 c（ただし $p(c) > 0$）に対して成り立つとき、かつそのときに限る。

16.6.1.2 条件付き分布の一致と周辺分布の一致

すべての条件付きスライスで $q(\cdot \mid c) = p(\cdot \mid c)$ を一致させることは、画像の周辺分布で $q(x) = p(x)$ を一致させることよりも真に強い条件である。たとえば、CIFAR-10 の生成器がリアルな画像を生成するものの、すべてのクラスラベルを巡回的にずらしてしまう場合 $(q(\cdot \mid c) = p(\cdot \mid c+1))$ を考えよう。10 個のクラスはすべて等しい事前確率 $p(c) = 1/10$ を持つため、画像の周辺分布 $q(x)$ は $p(x)$ と厳密に一致し、無条件ダイバージェンスは最小値 $\mathrm{JS}(p(x), q(x)) = 0$ を達成する。しかし条件付き目的関数 Equation 16.40 では、すべてのスライスが異なるクラスと比較されるため（$\mathrm{JS}(p(\cdot \mid c), p(\cdot \mid c+1)) \approx \log 2$）、条件付きダイバージェンスはほぼ最大値となる（演習問題 1）。

逆に、各条件 c が事前頻度 $p(c)$ によって Equation 16.40 の中で重み付けされていることにも注意が必要である。確率 $p(c) = \epsilon$ の希少な条件が目的関数に寄与する量は高々 $2\epsilon \log 2$ であるため、不均衡なデータセットでは、生成器が希少クラスを無視するのを防ぐためにクラス均衡サンプリングが必要になることが多い。

16.6.2 条件付けのアーキテクチャ

16.6.2.1 3 つの条件付け機構

Figure 16.9 に、条件 c を生成器や識別器に入力する主な 3 つの方法を示す。

1. **連結**（concatenation）(Mirza and Osindero 2014) は、c をベクトル e_c に埋め込み、潜在ベクトル z と連結する（あるいは空間特徴マップにブロードキャストして連結する）。
2. **特徴変調**（feature modulation）は、中間特徴マップ h の各チャネルを、学習された c の関数によってスケーリングおよびシフトする（$a(c) \odot h + b(c)$）。先に h を正規化する場合、これは**条件付きバッチ正規化**（conditional batch normalization）(Vries et al. 2017; Brock et al. 2019)（あるいは AdaIN や FiLM）となり、生成器のあらゆる解像度で条件を注入する。
3. **射影識別器ヘッド**（projection discriminator head）(Miyato and Koyama 2018) は、クラス埋め込み e_c と識別器のプーリング済み特徴ベクトル $\phi(x)$ の内積をとり、無条件スコア $\psi_0(x)$ を加算する。

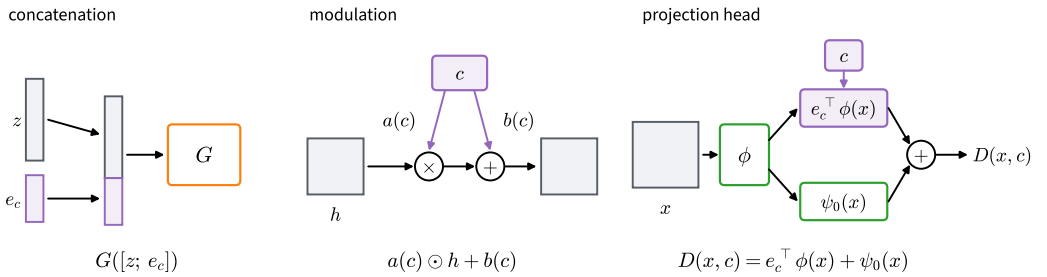


Figure 16.9: GAN における 3 つの条件付け機構。左：連結は、埋め込み e_c を入力または特徴マップに付加する。中央：特徴変調は、条件に依存するチャネルごとのスケーリングとシフト $a(c) \odot h + b(c)$ を適用する。右：射影識別器ヘッドは $e_c^\top \phi(x) + \psi_0(x)$ を計算する。

以下では、`Modulate` をクラスごとのスケーリングとシフトの埋め込みテーブルによって実装し、$a(c) = 1$ および $b(c) = 0$（恒等写像）で初期化したうえで、クラス 3 に対する勾配ステップが埋め込みテーブルの第 3 行のみを更新することを確認する。

```
class Modulate(nnx.Module):
    """Scale and shift each channel of h by a(c) and b(c), one row per class."""
    def __init__(self, num_classes, channels, *, rngs):
        # a(c) = 1, b(c) = 0: the identity
        self.a = nnx.Embed(num_classes, channels, rngs=rngs,
                           embedding_init=nnx.initializers.ones)
        self.b = nnx.Embed(num_classes, channels, rngs=rngs,
                           embedding_init=nnx.initializers.zeros)

    def __call__(self, h, c):
        return self.a(c)[:, None, None, :] * h + self.b(c)[:, None, None, :]
```

この層を実行する前に、2 つのことが予測できる。初期化時点では、クラスによらず出力は入力と一致するはずである。また、クラス 3（以下で用いる CIFAR-10 の並び順では猫）のみを指定するバッチで勾配ステップを実行すると、各テーブルの第 3 行だけが変化し、他の行は一切変化しないはずである。埋め込みの参照は、読み出した行にしか勾配を伝播しないからである。以下の確認コードでは、トイ目標値 $2h + 1$ に向けてそのようなステップを 1 回実行する。

```
mod = Modulate(10, 8, rngs=nnx.Rngs(0))
h = jax.random.normal(jax.random.key(0), (10, 4, 4, 8))  # one map per class
print('at initialization: max |out - h| =',
      float(jnp.abs(mod(h, jnp.arange(10)) - h).max()))
a0, b0 = mod.a.embedding[...], mod.b.embedding[...]
c3 = jnp.full(10, 3)  # a batch that requests class 3 only
grads = nnx.grad(lambda m: ((m(h, c3) - (2 * h + 1)) ** 2).mean())(mod)
nnx.Optimizer(mod, optax.sgd(0.1), wrt=nnx.Param).update(mod, grads)
rows = lambda w, w0: np.nonzero(np.any(w != w0, axis=1))[0].tolist()
print('after one step on class 3: rows changed in a',
      rows(mod.a.embedding[...], a0), 'and in b',
      rows(mod.b.embedding[...], b0))
```

```
at initialization: max |out - h| = 0.0
after one step on class 3: rows changed in a [3] and in b [3]
```

16.6.2.2 ベイズの定理による射影ヘッドの導出

ベイズの定理 $p(x \mid c) = p(c \mid x)p(x)/p(c)$ を Equation 16.39 の分子と分母（共通の事前分布 $p(c)$ を持つ）に適用すると、最適条件付きクリティックは次の 2 つの項に分解される。

$$\log \frac{p(x \mid c)}{q(x \mid c)} \;=\; \log \frac{p(c \mid x)}{q(c \mid x)} + \log \frac{p(x)}{q(x)}. \tag{16.41}$$

第 1 項は**クラスアライメント**（class alignment、x における実データと生成データのクラス事後分布の対数比）を測り、第 2 項は**無条件のサンプルリアリズム**（unconditional sample realism）を測る。両方のクラス事後分布を、共有の特徴表現 $\phi(x)$ 上のソフトマックス分類器、すなわち $p(c \mid x) \propto \exp(u_c^\top \phi(x))$ および $q(c \mid x) \propto \exp(w_c^\top \phi(x))$ としてモデル化すると、それらの対数比は $(u_c - w_c)^\top \phi(x)$ に c に依存しない対数分配関数の差を加えたものになる。この対数分配関数の差と $\log(p(x)/q(x))$ を単一のスカラーヘッド $\psi_0(x)$ に吸収し、$e_c = u_c - w_c$ とおけば、Miyato and Koyama (2018) の**射影識別器**（projection discriminator）が得られる（演習問題 2。演習問題 6 の補助分類器 GAN（auxiliary classifier GAN）と比較せよ）。

$$D(x, c) \;=\; e_c^\top \phi(x) + \psi_0(x). \tag{16.42}$$

16.6.3 CIFAR-10 におけるクラス条件付き生成

ここで、ラベル付きの訓練画像 50,000 枚（解像度 32×32）からなる CIFAR-10（クラスあたり 5,000 枚、`d2l.load_cifar10_arrays` で読み込む）を用いてクラス条件付き生成器を訓練する。

```
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']
train_X, train_y, test_X, test_y = d2l.load_cifar10_arrays()
first = [int(np.nonzero(np.asarray(train_y) == c)[0][0]) for c in range(10)]
d2l.show_images(np.asarray(train_X[np.array(first)]) / 2 + 0.5,
                num_rows=2, num_cols=5, titles=classes);
```

16.6.3.1 条件付き生成器と射影クリティック

`d2l.Generator` と `d2l.Discriminator`（16.5 節）を解像度 32×32 に拡張する。

- `CondGenerator` は、64 次元の単位分散クラス埋め込みを $4 \times 4 \times 64$ にブロードキャストし、生成器の `extra` 入力スロットに渡す（演習問題 4 では、この入力連結をブロックごとの `Modulate` 層と比較する）。
- `ProjectionCritic` は、4×4 の特徴マップ `self.phi(x)` を空間方向に足し合わせて $\phi(x)$ を求め、$e_c^\top \phi(x) + \psi_0(\phi(x))$（Equation 16.42）を計算する。このとき $e_c = 0$ で初期化することで、クリティックは無条件識別器として訓練を開始する。

```
class CondGenerator(d2l.Generator):
    """The backbone generator with a class embedding as extra 4x4 maps."""
    def __init__(self, latent_dim, resolution, base_ch, embed_dim=64,
                 num_classes=10, *, rngs):
        super().__init__(latent_dim, resolution, base_ch, extra_ch=embed_dim,
                         rngs=rngs)
        self.embed = nnx.Embed(num_classes, embed_dim, rngs=rngs,
                               embedding_init=nnx.initializers.normal(1.0))
```

```
    def __call__(self, z, y):
        e = self.embed(y)[:, None, None, :]
        ec = jnp.broadcast_to(e, (z.shape[0], 4, 4, e.shape[-1]))
        return super().__call__(z, ec)
```

nnx.Embed のデフォルトの初期化器は、埋め込み幅の平方根の逆数（ここでは 8 分の 1）を標準偏差として要素を抽出するため、コードでは単位分散を明示的に指定している。

ProjectionCritic は Equation 16.42 を実装したものである。基底クラスの phi は混合畳み込みの後に 4×4 の特徴マップを返し、それを空間位置にわたって足し合わせることで $\phi(x)$ が得られる。$\phi(x)$ 上の線形ヘッドが $\psi_0(x)$ を出力して、平坦化したマップを読み取る基底クラスのヘッドを置き換え、さらにテーブルがクラスごとに 1 つの埋め込み e_c を保持する。このテーブルはゼロから始まるため、クリティックは無条件クリティックとして出発し、データからラベル項を学習していく。

```
class ProjectionCritic(d2l.Discriminator):
    """The backbone critic with a projection head: e_c . phi(x) + psi_0(x)."""
    def __init__(self, resolution, base_ch, num_classes=10, *, rngs):
        super().__init__(resolution, base_ch, rngs=rngs)
        width = self.mix.out_features
        self.head = nnx.Linear(width, 1, kernel_init=d2l.fan_in_init,
                               rngs=rngs)  # psi_0 on the pooled features
        self.embed = nnx.Embed(num_classes, width, rngs=rngs,
                               embedding_init=nnx.initializers.zeros)

    def __call__(self, x, y):
        phi = self.phi(x).sum(axis=(1, 2))  # sum over the 4x4 positions
        return (self.embed(y) * phi).sum(axis=1) + self.head(phi).squeeze(-1)

net_G = CondGenerator(100, 32, 256, rngs=nnx.Rngs(0))
net_D = ProjectionCritic(32, 64, rngs=nnx.Rngs(0))
print(net_G(jnp.zeros((2, 100)), jnp.zeros(2, dtype=jnp.int32)).shape)
print(net_D(jnp.zeros((2, 32, 32, 3)), jnp.zeros(2, dtype=jnp.int32)).shape)
```

```
(2, 32, 32, 3)
(2,)
```

16.6.3.2 同一クラスペアによる訓練

d2l.train_r3gan（16.5 節）を用い、15,000 ステップにわたってバッチサイズ 128、勾配ペナルティの重み $\gamma = 0.5$ で訓練する。各ミニバッチでは、生成画像が実データのミニバッチとまったく同じクラスラベル y で条件付けされるため、相対論的ペアリング損失 $\log \sigma(D(x, c) - D(x', c))$ は常に**同**一クラス c の実画像と生成画像を比較する。なお、比較は純粋にクラス内でのみ行われるため、$D(x, c)$ はクラスごとに独立な加法定数 ζ_c の違いを除いてシフト不変である（演習問題 3）。記録されるホールドアウトのペアワイズランキング正解率 $\hat{P}(D(x, c) > D(x', c))$（512 組のテストペアで評価）も、このクラスごとのシフトに対して不変である。

```
rngs = nnx.Rngs(0)  # the initialization
net_G = CondGenerator(100, 32, 256, rngs=rngs)
```

```
net_D = ProjectionCritic(32, 64, rngs=rngs)
gamma = 0.5
ema_G, history = d2l.train_r3gan(
    net_G, net_D, d2l.rpgan_loss_D, d2l.rpgan_loss_G, gamma, 15000, train_X,
    train_y, eval_images=test_X, eval_labels=test_y, eval_size=512,
    batch_size=128, seed=1)
```

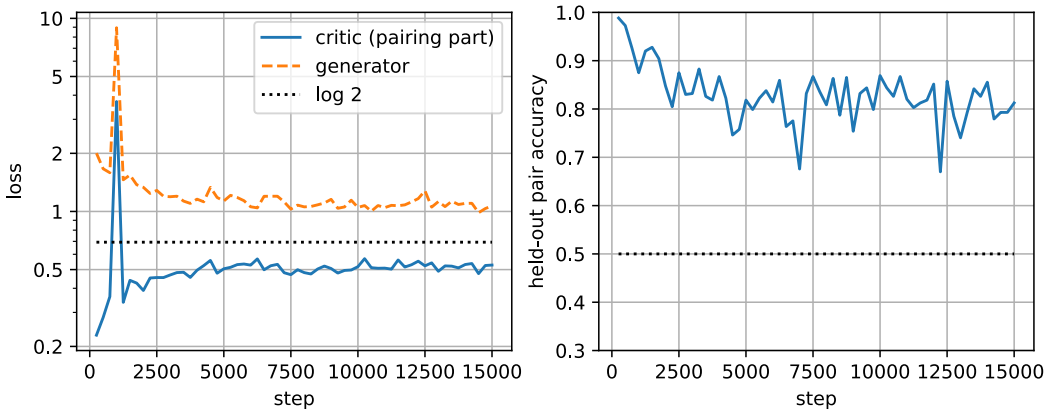
最後の 5,000 ステップにわたってログの平均をとり、ペア正解率については最初の 2,000 ステップでの平均も求める。

```
loss_D, loss_G, pen, acc = (np.mean(v[-5000 // 250:]) for v in (
    history['loss_D'], history['loss_G'], history['R1'] + history['R2'],
    history['accuracy']))
acc_early = history['accuracy'][history['step'] <= 2000].mean()
print(f'last 5,000 steps: critic loss {loss_D:.2f} (+ penalty '
      f'{gamma / 2 * pen:.3f}), generator loss {loss_G:.2f}, '
      f'pair accuracy {acc:.2f}')
print(f'first 2,000 steps: pair accuracy {acc_early:.2f}')
```

```
last 5,000 steps: critic loss 0.52 (+ penalty 0.064), generator loss 1.09, pair accuracy 0.81
first 2,000 steps: pair accuracy 0.92
```

loss_D、loss_G、およびホールドアウトの accuracy について全 15,000 ステップにわたる軌跡をプロットすると、射影クリティックと条件付き生成器がいかに速やかに安定したゲームへと落ち着くかがわかる。

```
steps = history['step']
fig, axes = d2l.plt.subplots(1, 2, figsize=(9, 3.2))
d2l.plot(steps, [history['loss_D'], history['loss_G'],
                 np.full_like(steps, np.log(2))], 'step', 'loss',
         legend=['critic (pairing part)', 'generator', 'log 2'],
         fmts=('-', '--', 'k:'), yscale='log', axes=axes[0])
d2l.plot(steps, [history['accuracy'], np.full_like(steps, 0.5)], 'step',
         'held-out pair accuracy', ylim=[0.3, 1], fmts=('-', 'k:'),
         axes=axes[1]);
```



初期の過渡期を過ぎると、クリティックのペアリング損失は 0.52 付近（$\log 2 \approx 0.69$ 未満）で安定し、ホールドアウトされた同一クラスペアのランキング正解率は 0.80 前後に落ち着く。

16.6.3.3 クラス条件付き生成サンプル

以下では、CIFAR-10 の 10 クラスそれぞれについて 9 枚のサンプルを `ema_G` から生成する。

```
z = jax.random.normal(jax.random.key(42), (90, 100))
y = jnp.repeat(jnp.arange(10), 9)  # nine per class
imgs = ema_G(z, y)
panels = [d2l.tile_images(imgs[9 * c:9 * (c + 1)], 3) for c in range(10)]
d2l.show_images(panels, 2, 5, titles=classes, scale=2);
```

3×3 の各パネルには、指定されたクラスの視覚的特徴（たとえば、飛行機や船なら青い空や水面、カエルや鹿なら緑の草木、自動車やトラックなら道路の背景）が反映されると同時に、9 個の潜在サンプルごとにポーズや色彩が変化している。

16.6.4 条件付き生成器の評価

条件付き生成器は、次の 3 つの異なる軸で失敗しうる。

1. **誤ったクラス**（低い条件アライメント）
2. **非現実的な画像**（低いサンプル忠実度）
3. **クラス内多様性の欠如**（各クラス内でのモード欠落）

Table 16.4 にまとめるように、単一の指標でこれら 3 つの失敗モードすべてを捉えることはできない。そこで、クラスアライメントを測る**レフェリー分類器**（referee classifier）に、周辺特徴距離（FD と MMD^2）および**クラスごとの** K-NN 適合率（precision）と再現率（recall）を組み合わせる。

Table 16.4: 条件付き生成の失敗モードに対する評価指標の感度。

失敗モード	クラスアライメント	周辺 FD、MMD^2	クラスごとの適合率	クラスごとの再現率
誤ったクラス	検出可	検出不可	検出可	検出可
低い画像忠実度	部分的	検出可	検出可	部分的

失敗モード	クラスアライメント	周辺 FD、MMD^2	クラスごとの適合率	クラスごとの再現率
低いクラス内多様性	検出不可（高く評価しうる）	検出可	検出不可（高く評価しうる）	検出可

16.6.4.1 レフェリー分類器とクラスアライメント

3 ブロックの CIFAR-10 分類器 `d2l.FeatureCNN`（16.5 節）を独立したレフェリーとして 15 エポック訓練し（テスト正解率は ～ 78% に達する）、そのクラス予測と最終層の 1 つ手前にある 128 次元の特徴量の層の両方を用いる。

```
cnn = d2l.FeatureCNN(resolution=32, rngs=nnx.Rngs(1))
d2l.train_feature_cnn(cnn, train_X, train_y, shift=4)
pred = d2l.batched(cnn, test_X).argmax(1)
cls_acc = [float((pred[test_y == c] == c).mean()) for c in range(10)]
print(f'CIFAR-10 test accuracy: {float((pred == test_y).mean()):.3f}')
```

```
CIFAR-10 test accuracy: 0.780
```

クラスあたり 500 枚の画像を `ema_G` から生成し、生成画像に対するレフェリーの分類正解率（**アライメント**）を、各クラスの実テスト画像に対する正解率と比較する。

```
def generate(seed, n=500):
    """n images of each class from the EMA generator, in class order."""
    z = jax.random.normal(jax.random.key(seed), (10 * n, 100))
    y = jnp.repeat(jnp.arange(10), n)
    return jnp.concatenate([ema_G(z[i:i + 1000], y[i:i + 1000])
                            for i in range(0, 10 * n, 1000)])

raw = generate(7)
gen = jnp.clip(raw, -1, 1)
yg = np.repeat(np.arange(10), 500)  # the requested classes
pred_gen = np.asarray(d2l.batched(cnn, gen).argmax(1))
align = [float((pred_gen[yg == c] == c).mean()) for c in range(10)]
```

`align` を実テスト画像に対するレフェリーの正解率 `cls_acc` と並べて 10 クラスそれぞれについて表にまとめると、`ema_G` が指定されたクラスラベルにどれほど忠実に従っているかがわかる。

```
print(f'{"class":12s}{"alignment":>11s}{"real":>7s}{"gap":>7s}')
for name, a, r in zip(classes + ['mean'], align + [np.mean(align)],
                      cls_acc + [np.mean(cls_acc)]):
    print(f'{name:12s}{a:11.2f}{r:7.2f}{a - r:+7.2f}')
```

```
class         alignment   real    gap
airplane           0.76   0.79  -0.02
automobile         0.80   0.92  -0.12
bird               0.66   0.64  +0.01
cat                0.51   0.58  -0.07
deer               0.66   0.72  -0.06
dog                0.62   0.75  -0.13
frog               0.81   0.85  -0.05
```

```
horse              0.71   0.80  -0.10
ship               0.89   0.89  +0.00
truck              0.80   0.86  -0.06
mean               0.72   0.78  -0.06
```

10 クラスにわたる平均クラスアライメントは 0.72–0.74 と、わずか 4–6 パーセントポイントだけ CIFAR-10 の実テスト画像に対するレフェリーの正解率（0.78）を下回るにすぎない。

16.6.4.2 忠実度とクラス内多様性

次に、クラスあたり 1,000 枚のテスト画像を 500 枚の参照セット（`f_real`）と 500 枚の実データ下限セット（real floor、`f_real2`）に分割し、周辺 FD と MMD^2（`d2l.fid_score`、`d2l.kid_score`）、およびクラスごとの K-NN 適合率と再現率（`d2l.precision_recall`、$K = 3$）を計算する。

```
features = lambda imgs: np.asarray(d2l.batched(cnn.features, imgs))
out_of_range = float((jnp.abs(raw) > 1).mean())
idx = jnp.argsort(test_y, stable=True).reshape(10, 1000)  # row c: class c
f_real = features(test_X[idx[:, :500].flatten()])    # the reference
f_real2 = features(test_X[idx[:, 500:].flatten()])   # the floor
f_gen = features(gen)
f_gen2 = features(jnp.clip(generate(8), -1, 1))      # a second draw
idx, pred_real = np.asarray(idx), np.asarray(pred)
```

それでは、周辺特徴距離（`d2l.fid_score`）と二乗 MMD（`d2l.kid_score`）を参照セット `f_real` との間で計算して、実データのテスト下限 `f_real2` と 2 回の生成サンプル群（`f_gen`、`f_gen2`）を比較し、あわせて範囲外となった生のピクセル値の割合 `out_of_range` も確認しよう。

```
draws = {'real (the floor)': f_real2, 'generated': f_gen,
         'generated, second draw': f_gen2}
print(f'{"against the reference":24s}{"FD (CIFAR-CNN)":>16s}'
      f'{"MMD^2 (CIFAR-CNN)":>19s}')
for name, f in draws.items():
    print(f'{name:24s}{d2l.fid_score(f, f_real):16.2f}'
          f'{d2l.kid_score(f, f_real):19.3f}')
print(f'generated values outside [-1, 1]: {out_of_range:.3f}')
```

```
against the reference     FD (CIFAR-CNN)  MMD^2 (CIFAR-CNN)
real (the floor)                    0.14             -0.001
generated                           1.74              0.114
generated, second draw              1.66              0.109
generated values outside [-1, 1]: 0.008
```

FD で見ると、生成セットは参照セットから下限セットの 10〜12 倍離れており、2 回目の抽出による FD の変動は 0.1 未満であるため、この距離は抽出の偶然ではなく生成器自体に起因するものである。MMD^2 では、下限セットがノイズの範囲内で 0 付近にあるのに対し、生成セットは約 0.11 となる。クランプ前の生のピクセル値のうち $[-1, 1]$ の外に出るのは 100 個に 1 個程度であり、クランプによって変化するピクセルはわずかで、問題が隠蔽されることもほとんどない。

これらの距離指標は、あらゆる違いを 1 つの数値に畳み込んでしまう。これに対し、クラスごとの適合率と再現率は 2 種類の違いを切り分ける。すなわち、そのクラスのどの実画像にも似ていない生成画像と、どの生成画像にも似ていない実画像である。各特徴ベクトルの周囲に、同じ集合内で 3 番目に近い近傍点まで届く球を描く（16.5.4.3 節）。**適合率**は、クラス c の生成画像のうちクラス c のいずれかの実データ球の内部に入るものの割合であり、**再現率**は、クラス c の実画

像のうちいずれかの生成データ球の内部に入るものの割合である。次のセルでは、参照セットに対してクラスごとに両者を計算し、クラス間の平均と最悪クラスを表示したうえで、最後にクラス内特徴分散（各クラス内の特徴量の分散を特徴量とクラスにわたって平均した値）を出力する。

```
by_class = lambda f: np.asarray(f).reshape(10, -1, f.shape[-1])

def pr_by_class(f):
    """Precision and recall of each class of f against the reference."""
    return np.array([d2l.precision_recall(r, g) for r, g in
                     zip(by_class(f_real), by_class(f))])

print(f'{"per class, k = 3":24s}{"precision":>10s}{"worst class":>17s}'
      f'{"recall":>8s}{"worst class":>17s}')
for name, f in draws.items():
    pr = pr_by_class(f)
    lo = pr.argmin(0)
    print(f'{name:24s}{pr[:, 0].mean():10.2f}{classes[lo[0]]:>12s}'
          f'{pr[lo[0], 0]:5.2f}{pr[:, 1].mean():8.2f}'
          f'{classes[lo[1]]:>12s}{pr[lo[1], 1]:5.2f}')
# Variance within each class, averaged over features and classes
cvar = lambda f: float(by_class(f).astype(np.float64).var(1).mean())
print(f'within-class feature variance: generated {cvar(f_gen):.3f}, '
      f'real {cvar(f_real):.3f}, the floor {cvar(f_real2):.3f}')
```

```
per class, k = 3         precision      worst class  recall      worst class
real (the floor)              0.85       truck 0.80    0.82    airplane 0.80
generated                     0.90       truck 0.87    0.57  automobile 0.46
generated, second draw        0.91       truck 0.86    0.57  automobile 0.46
within-class feature variance: generated 0.170, real 0.213, the floor 0.207
```

クラスごとの適合率は、生成画像（0.90）の方が実データ下限（0.85）よりもむしろ**高く**、生成特徴量が実データのクラスクラスタの十分内側に位置していることがわかる。一方で、クラスごとの再現率は大幅に低く（0.58 対 0.83、さらにクラス内特徴分散も約 20% 低い）、生成器が各クラスの多様性全体を網羅するのではなく、モード付近に集中していることを示している。

16.6.4.3 重心プロトタイプベースライン

高いアライメントと高い適合率がなぜ多様性の欠如を覆い隠してしまうのかを確認するため、実データ下限セットの中から特徴量がクラスの重心（centroid）に最も近い画像をクラスあたり 100 枚選んで**プロトタイプベースライン**（prototype baseline）を構築し、ランダムに選んだ実データ下限セットの画像 100 枚および生成画像 100 枚（いずれもクラスあたり）と比較する。

```
floor = by_class(f_real2)  # test images 500-999 of each class
dist = ((floor - floor.mean(1, keepdims=True)) ** 2).sum(-1)
near = np.argsort(dist, axis=1)[:, :100]  # 100 distinct images per class
rows = np.arange(10)[:, None]
sets = {  # 100 images per class: features and the referee's predictions
    'real, random': (floor[:, :100], pred_real[idx[:, 500:600]]),
    'real, nearest centroid': (floor[rows, near],
                               pred_real[idx[:, 500:][rows, near]]),
    'generated': (by_class(f_gen)[:, :100],
                  pred_gen.reshape(10, 500)[:, :100])}
```

```
print(f'{"100 per class":24s}{"alignment":>10s}{"FD":>7s}{"MMD^2":>8s}'
      f'{"precision":>10s}{"recall":>8s}')
for name, (f, p) in sets.items():
    f = f.reshape(1000, -1)
    prec, rec = pr_by_class(f).mean(0)
    print(f'{name:24s}{(p == rows).mean():10.2f}'
          f'{d2l.fid_score(f, f_real):7.2f}{d2l.kid_score(f, f_real):8.3f}'
          f'{prec:10.2f}{rec:8.2f}')
```

```
100 per class            alignment     FD   MMD^2 precision  recall
real, random                  0.79   0.43  -0.002      0.86    0.83
real, nearest centroid        0.90   2.92   0.116      0.97    0.36
generated                     0.72   2.02   0.115      0.89    0.65
```

Table 16.4 の予測どおり、最近傍重心のプロトタイプはほぼ完璧なアライメント（約 0.9 で、ランダムな実画像よりも高い）と適合率（約 0.97）を達成する一方、特徴量の共分散が人為的に収縮しているため、再現率の著しい低下（0.4 未満対約 0.85）と FD の悪化（約 3 対 0.43）を招く。BigGAN の**切断トリック**（truncation trick）は、潜在分布 z をゼロに向かって切断することで、意図的に再現率を犠牲にしてより高い適合率とアライメントを得る手法である（演習問題 5）。

16.6.5 まとめ

生成器 $G(z, c)$ とクリティック $D(x, c)$ の両方を $c \sim p(c)$ で条件付けると、スライスごとの最適クリティック $D^\star(x, c) = \log(p(x \mid c)/q(x \mid c))$ が得られ、期待条件付きダイバージェンス $\mathbb{E}_{c\sim p(c)}[2\,\mathrm{JS}(p(\cdot \mid c), q(\cdot \mid c))] - 2\log 2$ が最小化される。ベイズの定理を適用すると $D^\star(x, c)$ はクラスアライメント項 $\log(p(c \mid x)/q(c \mid x))$ と無条件リアリズム項 $\log(p(x)/q(x))$ に分解され、これが射影識別器ヘッド $D(x, c) = e_c^\top \phi(x) + \psi_0(x)$ の動機となる (Miyato and Koyama 2018)。条件付き生成器の評価では、条件アライメント（独立したレフェリー分類器による評価）、サンプル忠実度（クラスごとの適合率）、およびクラス内多様性（クラスごとの再現率と特徴距離）を切り分ける必要がある。再現率の指標がなければ、典型的なクラスプロトタイプのみを出力する生成器であっても、クラス内多様性を崩壊させたまま高い分類器アライメントと高い適合率を達成できてしまう。

16.6.6 演習問題

1. [code] **既知の密度を持つ 2 つのガウス分布スライス**。$c \in \{1, 2\}$ が等確率であり、$p(\cdot \mid 1) = \mathcal{N}(-2, 1)$、$p(\cdot \mid 2) = \mathcal{N}(2, 1)$ であるとし、$q(\cdot \mid 1) = \mathcal{N}(0, 1)$、$q(\cdot \mid 2) = \mathcal{N}(2, 1)$ と仮定する。
    1. 理論値 Equation 16.40 を 1 次元の数値求積法によって計算せよ。
    2. 小規模な MLP クリティックを $(x, \text{one_hot}(c))$ 上で訓練して Equation 16.38 を最大化し、収束後の目的関数値が (a) と一致することを確認したうえで、両方のスライスについて $D(x, c)$ を厳密な対数密度比 Equation 16.39 と重ねてプロットせよ。
    3. 次に、生成器が 2 つのクラスを入れ替えた場合、すなわち $q(\cdot \mid 1) = \mathcal{N}(2, 1)$ かつ $q(\cdot \mid 2) = \mathcal{N}(-2, 1)$ の場合を考える。条件付き目的関数値 Equation 16.40 と周辺分布に対する無条件目的関数値（c を無視した場合）の両方を計算し、訓練によって確認せよ。
2. **射影識別器ヘッドの表現力**。@eq-gan-cond-projection の導出では、$\phi(x)$ という共有の特徴マップを $p(c \mid x)$ と $q(c \mid x)$ の両方に仮定した。

   1. $p(c \mid x) \propto \exp(u_c^\top \phi_p(x))$ と $q(c \mid x) \propto \exp(w_c^\top \phi_q(x))$ が異なる特徴マップを用いる場合でも、得られる最適クリティックは連結した特徴ベクトル $(\phi_p(x), \phi_q(x))$ 上で依然として Equation 16.42 の射影形式をとることを示せ。
   2. K 個の離散クラスに対し、**任意の**関数 $D(x, c)$ が $e_c^\top \phi(x) + \psi_0(x)$ の形式で表せることを、$\dim(\phi(x)) \geq K$ が成り立つ場合について示せ。

3. [short-code] **分類器としての射影クリティックの利用**。訓練済みの射影クリティック `net_D` は、10 個のスコア $S_{ic} = D(x_i, c)$ を各画像 x_i に対して割り当てる。

   1. Equation 16.41 を用いて、$D^\star(x, c) - D^\star(x, c')$ を $p(c \mid x)$ と $q(c \mid x)$ によって表せ。なぜ $\arg\max_c D^\star(x, c)$ はベイズ分類器 $\arg\max_c p(c \mid x)$ と一致しないのか。
   2. 同一クラスペアリングではクラスごとに任意のシフト ζ_c が $D(x, c)$ に残るため、$\hat{\zeta}_c$ をクラス c の生成画像に対するクリティックスコアの平均として推定し、`test_X` 上での分類正解率を、補正なしの $\arg\max_c S_{ic}$ とシフト補正後の $\arg\max_c (S_{ic} - \hat{\zeta}_c)$ の間で比較せよ。
   3. シフト補正後のクリティックの正解率がランダム予測($1/10$)へと、$q(\cdot \mid c) \to p(\cdot \mid c)$ となるにつれて近づかなければならないのはなぜか。

4. [code] **特徴変調型生成器と入力連結の比較**。●クラス情報が入力の連結ではなく、すべての畳み込みブロックの後に置かれた `Modulate` 層を通じて入力される条件付き生成器を構築せよ。

   1. 初期化時点（$a(c) = 1, b(c) = 0$）では、変調型生成器の出力は c に依存しない。最初の生成器ステップにおいて、$a(c)$ と $b(c)$ に関する非ゼロの勾配が射影クリティックからどのように生じるかをたどれ。
   2. 変調型生成器と `CondGenerator` の両方をそれぞれ 2 つのシードで 4,000 ステップ訓練し、平均クラスアライメントとクラスごとの再現率を比較せよ。

5. [code] **切断トリックと適合率・再現率のトレードオフ**。切断トリック (Brock et al. 2019) は、各潜在座標を $[-\tau, \tau]$ に切断された標準正規分布からサンプリングする。

   1. 100 次元の潜在ベクトルと $\tau \in \{0.5, 1, 2\}$ に対し、切断サンプリングのもとでの二乗ノルムの期待値 $\mathbb{E}[\|z\|^2]$ を計算し、訓練時の $\mathbb{E}[\|z\|^2] = 100$ と比較せよ。
   2. `ema_G` を $\tau \in \{0.5, 1, 2, \infty\}$ の切断サンプリングのもとで評価し、クラスアライメント、クラスごとの適合率と再現率、および `f_real` に対する FD を報告せよ。また、τ を小さくすると再現率と引き換えにアライメントと適合率が向上し、生成器が重心プロトタイプの領域へ近づくことを確認せよ。

   Simon Prince『Understanding Deep Learning』問題 15.6 より改作。

6. **補助分類器 GAN（AC-GAN）のバイアス**。●AC-GAN (Odena et al. 2017) は、補助分類器 $C(c \mid x)$ を訓練し、交差エントロピー $-\mathbb{E}_{c \sim p(c), x \sim q(\cdot \mid c)}[\log C(c \mid x)]$ を生成器の損失に加える。

   1. この補助的な生成器損失が次の式に等しいことを示せ。

$$\mathbb{E}_{x \sim q}\Big[H(q(\cdot \mid x)) + D_{\mathrm{KL}}(q(\cdot \mid x) \,\|\, C(\cdot \mid x)) \Big].$$

   2. $C(c \mid x) = p(c \mid x)$ であり、かつクラス間に重なりがある場合（$H(p(\cdot \mid x)) > 0$）、この項を最小化すると生成器がデータ分布の重なり合う領域を避けるようになり、射影識別器（Equation 16.41）と比べてクラス内多様性が損なわれる理由を説明せよ。

16.7 GAN を超える敵対的損失

住宅価格のような単一のスカラーを予測する場合、平均二乗誤差（mean squared error、L_2 損失）を最小化するのが標準的な手法である。しかし、超解像（低解像度のサムネイル c から高解像度画像 x への写像）やスケッチから写真への変換（pix2pix、Isola et al. (2017)）といった条件付き画像生成問題では、ピクセルごとの L_2 および L_1 回帰損失はぼやけた出力を生む。同じ低解像度入力に対して多数の異なる高解像度テクスチャが対応するため、ピクセルごとの二乗誤差を最小化すると、ネットワークはそれらの条件付き平均 $\mathbb{E}[x \mid c]$ を出力せざるを得なくなる。これは、もっともらしい鮮明な画像のいずれにも属さないぼやけた像である。そこで回帰損失に識別器（discriminator）$D(x, c)$ を組み合わせ、出力が条件 c に対して現実的な画像かどうかをスコア付けすると、鮮明な高周波の細部が復元される。

一般的な画像生成では拡散モデルやフローマッチングモデル（第 17 章）が単体の GAN に取って代わったにもかかわらず、敵対的損失（adversarial loss）が今なお広く使われ続けている理由は、まさにこの仕組みにある。学習された識別器は現在でも、複数ステップの拡散モデルから蒸留された 1 ステップ生成器、潜在拡散モデルが動作する基盤となる画像オートエンコーダ（トークナイザ）、そしてスペクトログラムから音声波形を再構成するニューラル音声ボコーダの訓練を担っている。以下では、最小限の二峰性回帰問題を用いて条件付き平均による平均化効果を解析し、高ノイズのノイズ除去教師からの 1 ステップ蒸留が識別器なしでは条件付き平均へ崩壊する理由を示したうえで、現代の生成パイプラインにおいて敵対的損失がどこで使われているかを整理する。

16.7.1 条件付き平均と多峰性ターゲット

16.7.1.1 条件付き平均の分解

$g(c)$ を、x を条件 c から予測する決定論的予測器とし、$\mu(c) = \mathbb{E}[x \mid c]$ を条件付き平均（conditional mean）とする。$\mu(c)$ を $\mathbb{E}[\|x - g(c)\|^2]$ の内部で足して引くと、交差項 $2\,\mathbb{E}[\langle x - \mu(c), \mu(c) - g(c)\rangle] = 0$ が（c で条件付けた繰り返し期待値の法則により）消去され、次の直交分解が得られる。

$$\mathbb{E}\big[(x - g(c))^2\big] = \mathbb{E}\big[(x - \mu(c))^2\big] + \mathbb{E}\big[(\mu(c) - g(c))^2\big]. \tag{16.43}$$

第 1 項は削減不能な条件付き分散（conditional variance）$\mathrm{Var}(x \mid c)$ であり、第 2 項は $g(c)$ と $\mu(c)$ の二乗距離である。したがって、二乗誤差は条件付き平均 $g^\star(c) = \mu(c)$ によって一意に最小化される。$p(x \mid c)$ が単峰性で集中しているとき、$\mu(c)$ は代表的なターゲットとなる。一方、$p(x \mid c)$ が多峰性のとき、$\mu(c)$ はモード間の低密度領域に落ち込む。L_2 を L_1 損失 $\mathbb{E}[|x - g(c)|]$ に置き換えても、条件付き平均が座標ごとの条件付き中央値（conditional median）に置き換わるだけであり（演習問題 2）、対称な二峰性分布ではこれも同様に分布の台（support）の外側に位置する。

16.7.1.2 二峰性の 2 バンドベンチマーク

$c \sim \mathrm{Uniform}[-1, 1]$ とし、空白の回廊を挟んで平行に並ぶ 2 本のデータバンドをもつ 1 次元の条件付き分布を考えよう。

$$\begin{aligned} &x = c + b + 0.05\,\xi, \\ &b \in \{-1, +1\} \text{ with probability } \tfrac{1}{2} \text{ each}, \quad \xi \sim \mathcal{N}(0, 1). \end{aligned} \tag{16.44}$$

条件付き平均 $\mu(c) = c$ は 2 本のバンド $x = c \pm 1$ の間にある空白の回廊の真ん中を通り、$\mathbb{E}[(x - \mu(c))^2] = 1 + 0.05^2 = 1.0025$ を達成する。対照的に、上側のバンドに絞り込む予測器

$g(c) = c + 1$ の二乗誤差は $\frac{1}{2}(0) + \frac{1}{2}(2^2) + 0.05^2 = 2.0025$ と 2 倍の大きさになる。したがって二乗誤差は、正当なデータモードに絞り込むことに対してペナルティを課してしまう。

しかし、条件付き識別器 $D(x, c)$ は対数密度比（log density ratio）を推定する（16.6 節）。

$$D^\star(x, c) = \log \frac{p(x, c)}{q(x, c)}.$$

$p(x, c) \approx 0$ となる空白の回廊では $D^\star(x, c)$ が大きな負の値をとるため、非飽和型の敵対的損失 $-\log \sigma(D(\hat{x}, c))$ を最小化すると、予測器 $\hat{x} = g(c)$ は高密度なデータバンドのいずれかへと押し出される。

同一構造をもつ隠れ層 2 層の tanh 多層パーセプトロン（MLP、`g_mse` と `g_adv`）を同じ初期重みから 2 つ訓練してこれを検証しよう。`g_mse` は二乗誤差を最小化し、一方 `g_adv` は Leaky ReLU 識別器 $D(x, c)$ を相手に `d2l.gan_loss_D` と `d2l.gan_loss_G`（16.4 節）を用いて訓練する。

```
def make_bands(key, n):
    """Draw n pairs (c, x) from the two bands x = c + b + 0.05 xi."""
    k_c, k_b, k_xi = jax.random.split(key, 3)
    c = jax.random.uniform(k_c, (n, 1), minval=-1, maxval=1)
    b = jax.random.rademacher(k_b, (n, 1), dtype=jnp.float32)
    return c, c + b + 0.05 * jax.random.normal(k_xi, (n, 1))

rngs = nnx.Rngs(0)
g_mse = d2l.make_mlp([1, 32, 32, 1], nnx.tanh, rngs=rngs)
g_adv = nnx.clone(g_mse)  # both students start alike
D = d2l.make_mlp([2, 32, 32, 1], lambda h: nnx.leaky_relu(h, 0.2), rngs=rngs)
key_mse, key_adv, key_data = jax.random.split(rngs(), 3)
c_data, x_data = make_bands(key_data, 200)  # a data sample for the plot
```

1 つ目の生徒モデルは Adam で二乗誤差を最小化し、その学習率はコサイン曲線に沿って 0.01 から 0 まで 2,000 ステップかけて減衰させる。どこに収束するかを知るのに実験は必要ない。Equation 16.43 により、各バンドから 1 単位離れた直線 $x = c$ に沿うはずである。これは 2 つ目の生徒モデルに対する対照群となる。

```
@nnx.jit(static_argnums=3)
def mse_step(g, opt, key, batch_size):
    c, x = make_bands(key, batch_size)  # a fresh batch at every step
    opt.update(g, nnx.grad(lambda g_: ((g_(c) - x) ** 2).mean())(g))

def train_mse(g, key, num_steps=2000, batch_size=128, lr=0.01):
    schedule = optax.cosine_decay_schedule(lr, num_steps)
    opt = nnx.Optimizer(g, optax.adam(schedule), wrt=nnx.Param)
    for k in jax.random.split(key, num_steps):
        mse_step(g, opt, k, batch_size)

train_mse(g_mse, key_mse)
```

2 つ目の生徒モデルは、`d2l.gan_loss_D` と `d2l.gan_loss_G`（16.4 節）を損失として対数損失ゲームを行う。各ステップで、クリティックは実データペア (x, c) と予測ペア $(\hat{x}, c)$ に対してロジスティック回帰を 1 ステップ進め、続いて生徒モデルが更新後のクリティックに対して $-\log \sigma(D(\hat{x}, c))$ を 1 ステップ進める。予測値にはデータの観測ノイズを加え、$\hat{x} = g(c) + 0.05\,\xi'$（毎回新たに

引いた $\xi' \sim \mathcal{N}(0,1)$ を使用）としてゲームに入力する。こうすることで、2 つの分布の違いは滑らかさではなく確率質量の配置場所のみになる。ゲームは回帰よりも進行が遅いため、この生徒モデルには 8,000 ステップを与える。一方、二乗誤差の生徒モデルは 2,000 ステップ後には最適解に達しており、ステップ数を増やしてもそれ以上は動かない。

```
@nnx.jit(static_argnums=5)
def adversarial_step(g, D, opt_G, opt_D, key, batch_size):
    k_data, k_xi = jax.random.split(key)
    c, x = make_bands(k_data, batch_size)
    xi = jax.random.normal(k_xi, (batch_size, 1))  # the data's noise
    real = jnp.concatenate([c, x], 1)
    fake = lambda g_: jnp.concatenate([c, g_(c) + 0.05 * xi], 1)
    loss_D = lambda D_: d2l.gan_loss_D(D_, real, fake(g))
    opt_D.update(D, nnx.grad(loss_D)(D))
    loss_G = lambda g_: d2l.gan_loss_G(D, real, fake(g_))
    opt_G.update(g, nnx.grad(loss_G)(g))

def train_adversarial(g, D, key, num_steps=8000, batch_size=128):
    opt_G = nnx.Optimizer(g, optax.adam(1e-3), wrt=nnx.Param)
    opt_D = nnx.Optimizer(D, optax.adam(2e-3), wrt=nnx.Param)
    for k in jax.random.split(key, num_steps):
        adversarial_step(g, D, opt_G, opt_D, k, batch_size)

train_adversarial(g_adv, D, key_adv)
```

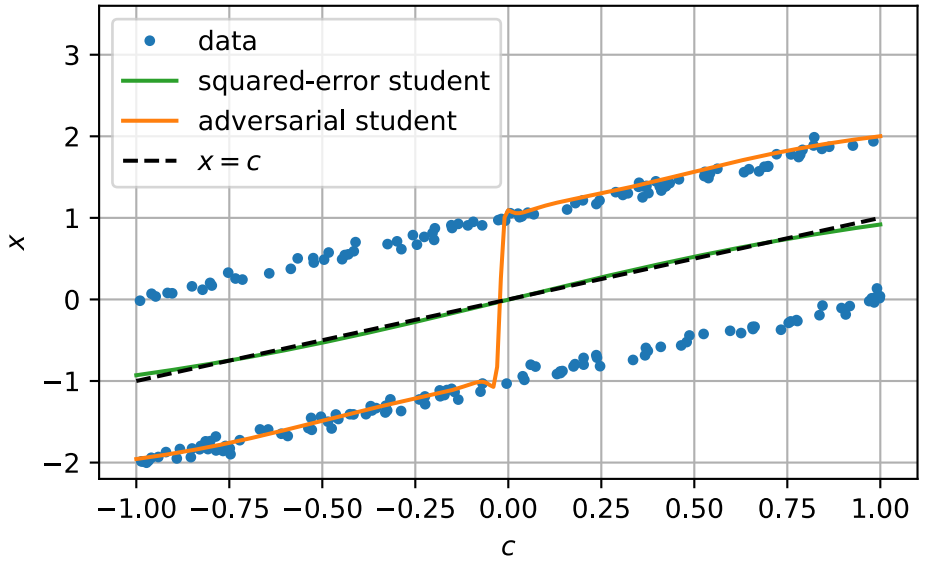
201 個の入力グリッド上で、各生徒モデルを 3 つの数値で要約する。曲線から近い方のバンドまでの平均距離、条件付き平均 $x = c$ までの平均距離、そして曲線が両方のバンドから 0.25 より大きく離れている入力の割合である。

```
cs = jnp.linspace(-1, 1, 201).reshape(-1, 1)
curves = {'squared error': g_mse(cs), 'adversarial': g_adv(cs)}
for name, x_hat in curves.items():
    to_band = jnp.minimum(jnp.abs(x_hat - cs - 1), jnp.abs(x_hat - cs + 1))
    print(f'{name}: {to_band.mean():.2f} from the nearer band, '
          f'{jnp.abs(x_hat - cs).mean():.2f} from x = c, '
          f'{(to_band > 0.25).mean():.0%} off the bands')
```

```
squared error: 0.98 from the nearer band, 0.02 from x = c, 100% off the bands
adversarial: 0.04 from the nearer band, 1.01 from x = c, 0% off the bands
```

curves に格納した学習済みの 2 本の曲線を、訓練サンプル（c_data, x_data）の上に破線の条件付き平均 $x = c$ と並べてプロットすると、回帰と敵対的訓練（adversarial training）の違いが一目でわかる。

```
d2l.plot([c_data, cs, cs, cs],
         [x_data, curves['squared error'], curves['adversarial'], cs],
         '$c$', '$x$', ['data', 'squared-error student',
                        'adversarial student', '$x = c$'],
         ylim=(-2.2, 3.6), fmts=('C0.', 'C2-', 'C1-', 'k--'),
         figsize=(5.5, 3.2))
```



予測どおり、二乗誤差モデル `g_mse` は空白の回廊に沿って条件付き平均 $\mu(c) = c$ をたどる（どちらのバンドからも約 1 単位離れており、予測の 100% がバンドから外れている）。一方、敵対的モデル `g_adv` は正当なデータバンドに絞り込む（最も近いバンドからの距離は平均で百分の数単位であり、回廊を横切る点は高々 2% にとどまる）。

ここで、根本的な知覚・歪みトレードオフ（perception–distortion trade-off）に注目しよう (Blau and Michaeli 2018)。理想的な確率的サンプラ $\hat{x} \sim p(x \mid c)$ でさえ、期待二乗誤差は $2\,\mathrm{Var}(x \mid c)$ となり、MSE で 2 倍（PSNR では 3 dB 低下）と、ぼやけた条件付き平均 $\mu(c)$ より悪化する（演習問題 1）。さらに、決定論的関数 $g(c)$ は入力 c ごとに 1 つの値しか出力できないため、**両方**のバンドを同時にカバーするには、確率的な潜在変数 z を c と併せて渡し、$\hat{x} = g(c, z)$ とする必要がある（演習問題 3）。

16.7.2 拡散蒸留における識別器

16.7.2.1 条件付き平均としてのノイズ除去器

拡散モデルとフローマッチングモデル（第 17 章）では、クリーンなデータ x_0 をノイズ付き入力 $x_\tau = x_0 + \tau\xi$（$\xi \sim \mathcal{N}(0, I)$）から予測するよう、ノイズ除去器（denoiser）ネットワークを二乗誤差で訓練する。Equation 16.43 により、最適なノイズ除去器は事後条件付き平均 $\mathbb{E}[x_0 \mid x_\tau]$ を計算する（17.1 節）。ノイズレベル τ が大きい領域では、多数のクリーンな画像が x_τ と整合するため、$\mathbb{E}[x_0 \mid x_\tau]$ はぼやけた平均になる。複数ステップのサンプラは、高ノイズから $\tau \to 0$ へと小さなステップを数多く刻むことで、この平均をそのまま返すのを避けている。

複数ステップの拡散教師モデルを高速な 1 ステップの生徒生成器へと**蒸留**（distillation）する際（17.8 節）、よく使われる手法の 1 つは、ノイズ $\tau\xi$ を生徒の出力 $\hat{x}$ に加えたうえで、$\hat{x}$ を凍結した教師のノイズ除去推定値 $\mathbb{E}[x_0 \mid \hat{x} + \tau\xi]$ へ回帰させることである。しかし高ノイズ τ では、教師のノイズ除去ターゲット自体が条件付き平均であるため、敵対的クリティックを加えない限り、1 ステップの生徒はぼやけた平均へと引き戻されてしまう。

16.7.2.2 2 バンドベンチマークにおける閉形式の蒸留

2 バンド問題 Equation 16.44 では、ノイズレベル τ における厳密なベイズ最適ノイズ除去教師は閉形式で表せる。ノイズ付き観測 $\tilde{x} = x + \tau\xi$（バンド $b \in \{-1, +1\}$ 上の点に対する観測）が与えられたとき、$\tilde{x} \mid c, b \sim \mathcal{N}(c + b, \tau^2 + 0.05^2)$ となる。ベイズの定理により、$b = +1$ 対 $b = -1$ の事後対数オッズ（log odds）は $2(\tilde{x} - c)/(\tau^2 + 0.05^2)$ となり、バンド中心の事後平均（posterior mean）は次式で与えられる。

$$\mathbb{E}[c + b \mid \tilde{x}, c] \;=\; c + \tanh\left(\frac{\tilde{x} - c}{\tau^2 + 0.05^2}\right). \tag{16.45}$$

ここで各生徒モデル $g(c, z)$ に 1 次元のガウス潜在変数 $z \sim \mathcal{N}(0, 1)$ を与えて両方のバンドをカバーできるモデル容量をもたせ、教師のノイズレベル $\tau = 2$ において 3 つの訓練目的関数を比較する。

1. **回帰のみ**：$\|\hat{x} - \text{stop_grad}(\text{teacher}(\hat{x} + \tau\xi, c, \tau))\|^2$ を最小化する。Equation 16.45 の $\tilde{x}$ に関する傾きは高々 $(\tau^2 + 0.05^2)^{-1} \approx \frac{1}{4} < 1$（$\tau = 2$ のとき）であるため、不動点方程式 $u = \mathbb{E}_\xi[\tanh((u + \sqrt{\tau^2 + 0.05^2}\,\xi)/(\tau^2 + 0.05^2))]$（オフセット $u = g(c, z) - c$ に関する式）は、縮小写像による唯一の不動点（fixed point）$u = 0$（空白の回廊）をもつ。
2. **クリティックのみ**：$g(c, z)$ を条件付き識別器 $D(x, c)$ を相手に `d2l.gan_loss_D` と `d2l.gan_loss_G` で訓練する。
3. **クリティック + 回帰**：敵対的損失と教師回帰損失を等しい重みで足し合わせる（敵対的拡散蒸留、すなわち Adversarial Diffusion Distillation、ADD、Sauer, Lorenz, et al. (2023) と同様）。

```
def teacher(x_tilde, c, tau):
    """Posterior mean of the band center c + b given x_tilde = x + tau xi."""
    return c + jnp.tanh((x_tilde - c) / (tau ** 2 + 0.05 ** 2))

@nnx.jit(static_argnums=7)
def student_step(g, D, opt_G, opt_D, key, lam, tau, batch_size):
    k_data, k_z, k_xi, k_tau = jax.random.split(key, 4)
    c, x = make_bands(k_data, batch_size)
    z = jax.random.normal(k_z, (batch_size, 1))  # the student's latent
    xi = jax.random.normal(k_xi, (batch_size, 1))  # the data's noise
    sample = lambda g_: g_(jnp.concatenate([c, z], 1)) + 0.05 * xi
    real = jnp.concatenate([c, x], 1)
    fake = lambda g_: jnp.concatenate([c, sample(g_)], 1)
    if D is not None:
        loss_D = lambda D_: d2l.gan_loss_D(D_, real, fake(g))
        opt_D.update(D, nnx.grad(loss_D)(D))
    noise = tau * jax.random.normal(k_tau, (batch_size, 1))
    def loss_G(g_):
        x_hat = sample(g_)  # the teacher's estimate is a fixed target
        target = jax.lax.stop_gradient(teacher(x_hat + noise, c, tau))
        loss = lam * ((x_hat - target) ** 2).mean()
        if D is not None:
            loss += d2l.gan_loss_G(D, real, fake(g_))
        return loss
    opt_G.update(g, nnx.grad(loss_G)(g))
```

```
def train_student(g, D, lam, key, tau=2.0, num_steps=4000, batch_size=128):
    """Train g(c, z) on the critic's loss if D is given, plus lam times the
    squared error toward the teacher's estimate at noise level tau."""
    opt_G = nnx.Optimizer(g, optax.adam(1e-3), wrt=nnx.Param)
    opt_D = (None if D is None
             else nnx.Optimizer(D, optax.adam(2e-3), wrt=nnx.Param))
    for k in jax.random.split(key, num_steps):
        student_step(g, D, opt_G, opt_D, k, lam, tau, batch_size)
```

これで生徒モデルは、各 c における出力の分布を表すようになった。関数 band_shares は、最初の実験と同じ 201 個の入力および潜在変数の等間隔な 64 個の分位点（quantile）で生徒モデルを評価し、近い方のバンドまでの平均距離と、上側バンドおよび下側バンドから 0.25 以内に収まる出力の割合を返す。

```
def band_shares(g, cs):
    """Mean distance of g(c, z) to the nearer band, and its shares on the
    upper and the lower band, over the inputs cs and 64 quantiles z."""
    zs = jax.scipy.special.ndtri((jnp.arange(64) + 0.5) / 64)
    c, z = (v.reshape(-1, 1) for v in jnp.meshgrid(
        cs.flatten(), zs, indexing='ij'))
    u = g(jnp.concatenate([c, z], 1)) - c  # offset from the corridor
    to_band = jnp.minimum(jnp.abs(u - 1), jnp.abs(u + 1))
    return (to_band.mean(), (jnp.abs(u - 1) < 0.25).mean(),
            (jnp.abs(u + 1) < 0.25).mean())
```

各生徒モデルは 3 つの乱数シードで訓練する。同じシード内では、3 つの生徒モデルは同一の重みから出発し、同一のデータを観測する。

```
students = {'regression': (False, 1.0), 'critic': (True, 0.0),
            'critic + regression': (True, 1.0)}
for name, (has_critic, lam) in students.items():
    for seed in range(3):
        rngs = nnx.Rngs(seed)
        g = d2l.make_mlp([2, 32, 32, 1], nnx.tanh, rngs=rngs)
        D = d2l.make_mlp([2, 32, 32, 1], lambda h: nnx.leaky_relu(h, 0.2),
                         rngs=rngs)
        train_student(g, D if has_critic else None, lam, rngs())
        dist, upper, lower = band_shares(g, cs)
        print(f'{name}, seed {seed}: {dist:.2f} from the nearer band, '
              f'{upper:.0%} upper, {lower:.0%} lower, '
              f'{1 - upper - lower:.0%} off the bands')
```

```
regression, seed 0: 0.96 from the nearer band, 0% upper, 0% lower, 100% off the bands
regression, seed 1: 0.97 from the nearer band, 0% upper, 0% lower, 100% off the bands
regression, seed 2: 0.97 from the nearer band, 0% upper, 0% lower, 100% off the bands
critic, seed 0: 0.03 from the nearer band, 52% upper, 47% lower, 1% off the bands
critic, seed 1: 0.06 from the nearer band, 50% upper, 49% lower, 1% off the bands
critic, seed 2: 0.03 from the nearer band, 52% upper, 47% lower, 2% off the bands
critic + regression, seed 0: 0.04 from the nearer band, 48% upper, 50% lower, 1% off the bands
critic + regression, seed 1: 0.04 from the nearer band, 49% upper, 50% lower, 1% off the bands
critic + regression, seed 2: 0.04 from the nearer band, 49% upper, 49% lower, 1% off the bands
```

3 シードの結果から、不動点解析の妥当性が確認できる。

- **高ノイズ教師への回帰**では、出力の 100% が空白の回廊（どちらのバンドからも約 1 単位の距離）へ崩壊し、潜在変数 z は無視される。
- **クリティックのみ**と**クリティック + 回帰**はいずれも、出力の 98–99% を 2 本のデータバンド上に直接配置し（平均距離は百分の数単位で常に 0.1 未満）、確率質量を上側と下側のバンドに均等に（$\sim 49\%/49\%$）潜在変数 z を通じて振り分ける。バンドの幅が $\sigma = 0.05$ と狭いため、最適クリティックの復元傾き（$\delta/0.05^2 = 400\delta$）が高ノイズ教師による内側への緩やかな引力を容易に上回るのである。

16.7.2.3 大規模な蒸留、オートエンコーダ、ボコーダ

回帰損失や知覚損失によるアンカーと敵対的識別器のこの組み合わせは、現代の生成システムにおける主要な 3 つのクラスに共通して見られる。

1. **敵対的拡散蒸留（ADD と DMD2）：**ADD (Sauer, Lorenz, et al. 2023) は、ノイズ除去教師への回帰損失と凍結された DINOv2 特徴に基づく識別器を組み合わせて 1 ステップの画像生成器を訓練し、上の実験と同じ優劣関係を確認している（教師回帰のみではぼやけた画像になるが、識別器を加えると鮮明な 1 ステップサンプルが得られる）。分布マッチング蒸留（distribution matching distillation、DMD、Yin, Gharbi, Zhang, et al. (2024)）は、凍結された教師のスコア $\nabla_x \log p_\tau$ と偽データ分布の拡散モデルのスコア $\nabla_x \log q_\tau$ の差を用いて $D_{\mathrm{KL}}(q_\tau \,\|\, p_\tau)$ を降下する。この差はすでに登場している。すなわち、$\nabla_x \log p_\tau - \nabla_x \log q_\tau = \nabla_x \lambda_\tau$ は、ノイズを加えた両分布間における最適クリティック $\lambda_\tau = \log(p_\tau / q_\tau)$ の入力勾配である。したがって DMD の生成器更新は、重みなし（$w = 1$）の逆方向 KL 更新（演習問題 5）において、クリティックの傾きを識別器の代わりに 2 つのスコアネットワークから供給したものにほかならない。両方の分布にノイズが加えられているため、あらゆるノイズレベル $\tau > 0$ で両者の台は重なり合う。これは 16.1.2.3 節で述べたインスタンスノイズと同じ仕組みである。DMD2 (Yin, Gharbi, Park, et al. 2024) はさらに実画像に対する敵対的識別器を追加し、1 ステップの生徒が教師の品質を超えられるようにしている。
2. **潜在画像オートエンコーダ（トークナイザ）：**潜在拡散モデル（17.7 節）は、オートエンコーダ（VQGAN や Stable Diffusion VAE など、Esser et al. (2021); Rombach et al. (2022)）を用いて画像を低次元の潜在空間へと圧縮する。圧縮された潜在コードだけでは微細な局所テクスチャが一意に定まらないため、デコーダの訓練ではピクセルごとの L_1/L_2 損失および知覚損失（LPIPS）にパッチレベルの識別器（PatchGAN）を加え、くっきりとした高周波テクスチャを保証している。
3. **ニューラル音声ボコーダ：**テキスト音声合成パイプラインは通常、メルスペクトログラム（フレームレベルの周波数振幅を指定するが位相は捨てる）を予測し、HiFi-GAN (Kong et al. 2020) などのボコーダを用いてスペクトログラムを音声波形へと逆変換する。正弦波 $\sin(\omega n + \vartheta)$ の未知の位相 $\vartheta \sim \mathrm{Uniform}[-a, a]$ に関する期待値は、振幅が $\sin(a)/a$ 倍に縮小するため（$\vartheta \sim \mathrm{Uniform}[-\pi, \pi]$ では完全に消失する。演習問題 5）、波形の回帰は無音へと崩壊してしまう。そこで HiFi-GAN は、メルスペクトログラムの L_1 損失および識別器の活性化値に対する特徴マッチング損失を、複数周期・マルチスケールの波形識別器と組み合わせている。

16.7.3 本章の敵対的目的関数の比較

Table 16.5 に、本章で学んだ 6 つの敵対的定式化を、最適クリティックが何を推定するか、入力をどのようにスコア付けするか、そしてクリティックの関数クラスをどのように正則化するかという 3 つの軸に沿ってまとめる。

Table 16.5: 本章で解析した敵対的目的関数のまとめ。

目的関数	最適クリティックの推定対象	スコアリング規則	クリティックの制約・正則化	台が重ならない場合の挙動（最適クリティック下）
対数損失 / f-GAN（16.1 節、16.2 節）	$\log(p/q)$、または $f'(p/q)$	単一サンプル分類	制約なし	飽和する
ヒンジ（16.2 節）	$\operatorname{sign}\log(p/q)$	マージン分類	制約なし（またはスペクトルノルム）	飽和する（$2/\kappa$ まで線形、κ リプシッツなクリティックの場合）
MMD（16.2 節）	カーネル平滑化された $p-q$（証人関数）	平均の差	RKHS 単位球	カーネルバンド幅 s 以内で非ゼロ勾配
1 次ワッサースタイン（16.2 節）	カントロビッチポテンシャル	平均の差	1 リプシッツ球（例：WGAN-GP）	任意の距離で単位傾き
RpGAN（16.3 節）	$\log(p/q)+c$	ペアワイズなブラッドリー・テリーランキング	制約なし	飽和する
R3GAN：RpGAN + R_1+R_2（16.4 節）	$\log(p/q)$ をスケール $2\sqrt{\gamma}$ 未満でローパスフィルタ処理したもの	ペアワイズなブラッドリー・テリーランキング	p と q 上のゼロ中心入力勾配ペナルティ	ペナルティ付きネットワークによるギャップ間の滑らかな補間

反復型の拡散モデルやフローモデル（第 17 章）と比べると、敵対的生成器は 1 回の順伝播で推論でき、劣決定な逆問題に対しても鮮明な出力を生成できるという利点がある。その一方で、次の 4 つの本質的な課題を抱えている。(i) 明示的な尤度評価ができないこと、(ii) 調整された勾配ペナルティを必要とする 2 ネットワーク結合のミニマックスダイナミクス、(iii) モード欠落に対する内在的なペナルティが弱いこと（16.1.3.3 節）、(iv) 大規模化に伴う訓練不安定性の懸念である (Brock et al. 2019)。次章（第 17 章）では、ミニマックス訓練を複数ステップの反復的なノイズ除去に置き換える拡散モデルとフローマッチングモデルへと進む。

16.7.4 まとめ

二乗誤差 $\mathbb{E}[\|x-g(c)\|^2]$ を最小化すると、予測器は条件付き平均 $\mu(c)=\mathbb{E}[x \mid c]$（Equation 16.43）へと向かうため、ターゲット x が条件 c のもとで曖昧な場合には常に異なるモード間が平均化されてしまう。予測器に条件付き識別器 $D(x,c)$（$\log(p(x,c)/q(x,c))$ を推定するモデル）を組み合わせると、モード間の低密度な回廊にある予測にペナルティが課されて出力はデータ多様体上へと押し出され、MSE が最大 2 倍増加する（PSNR で 3 dB 低下する）ことと引き換えに知覚的な現実性が得られる (Blau and Michaeli 2018)。拡散モデルのノイズ除去器も同様に事後平均 $\mathbb{E}[x_0 \mid x_\tau]$ を計算するため、高ノイズのノイズ除去器へ直接回帰させて 1 ステップ生成器を蒸留すると、敵対的クリティックを加えない限り予測はぼやけた平均へと縮小してしまう。同じ理由から、敵対的識別器は潜在画像オートエンコーダ、ニューラル音声ボコーダ、および 1 ステップ蒸留拡散生成器において今なお不可欠な構成要素となっている。

16.7.5 演習問題

1. **理想的なサンプラにおける知覚・歪みトレードオフ。** $\hat{x}\sim p(x \mid c)$ を、x とは独立に（c を与えたもとで）サンプリングしたものとし、予測器が真の条件付き分布 $p(x \mid c)$ に厳密に一致するとする (Blau and Michaeli 2018)。

1. Equation 16.43 を用いて $\mathbb{E}\big[(x-\hat{x})^2\big] = 2\,\mathbb{E}\big[(x-\mu(c))^2\big]$ を証明せよ。厳密なサンプラは、PSNR（$10\log_{10}(x_{\max}^2/\text{MSE})$）において条件付き平均 $\mu(c)$ と比べて何デシベル失うか。
2. 2 バンド分布 Equation 16.44 において、厳密なサンプラの期待二乗誤差を、$\mu(c)=c$（1.0025）および単一バンド予測器 $g(c)=c+1$（2.0025）と比較せよ。この 3 つのうち、最適クリティック $D(x,c)$ によって真のデータと区別できないものはどれか。
3. $x\sim\mathcal{N}(0,1)$ かつ $c=x+\tau\xi$（ただし $\xi\sim\mathcal{N}(0,1)$ は独立）とする。決定論的な線形予測器 $\hat{x}=c/\sqrt{1+\tau^2}$ が周辺分布 $\mathcal{N}(0,1)$（x の分布）に一致することを示し、その MSE と条件付き平均 $\mathbb{E}[x\mid c]$ の MSE との比を $\tau\in\{0.1,1,10\}$ に対して求めよ。

2. [code] **条件付き平均と条件付き中央値（L_2 対 L_1）**。絶対誤差 $\mathbb{E}[|x-g(c)|]$ を最小化すると、条件付き中央値 $x_{\text{med}}(c)$（すなわち $P(x\le x_{\text{med}}(c)\mid c)=\frac{1}{2}$ を満たす値）が得られる。
   1. オフセット b（Equation 16.44）が値 $(-1,0,+1)$ を確率 $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$ または $(\frac{1}{4},\frac{1}{4},\frac{1}{2})$ でとるとき、それぞれの場合について条件付き平均と条件付き中央値を求め、各々がデータバンド上に位置するかどうかを述べよ。
   2. 非対称な事前分布 $P(b=+1)=r$ をもつ 2 バンド問題において、条件付き平均を r の関数として計算し、$r=\frac{3}{4}$ のとき条件付き中央値が上側バンド上に位置することを示せ。さらに、L_1 リスクが回廊 $g(c)-c\in(-1,1)$ 全体でどのようになるかを $r=\frac{1}{2}$ の場合について特徴づけよ。
   3. `g_mse` のコピーを L_1 損失を用いて (a) と (b) の設定で訓練し、学習された曲線を解析的な予測と比較せよ。
3. [code] **二値の潜在変数の追加**。●16.7.1.2 節の生徒モデルに独立な一様二値潜在変数 $z\in\{-1,+1\}$ を追加して $\hat{x}=g(c,z)$ とする。ただし、クリティックは引き続き (x,c) のペアをスコア付けするものとする。
   1. 1 つの生徒モデルを二乗誤差で、もう 1 つを敵対的に訓練し、$z=-1$ と $z=+1$ がそれぞれどちらのバンドに写像されるかを確認せよ。なぜ二乗誤差の生徒モデルは z を無視するのか。
   2. 陰的最尤推定（IMLE、Li and Malik (2018)）は、クリティックを用いずに $\mathbb{E}_{(c,x)}[\min_{z\in\{-1,+1\}}(g(c,z)-x)^2]$ を最小化する。この損失で生徒モデルを訓練し、各データ点について z に関する最小値をとることで両方の潜在状態が異なるバンドをカバーするようになる理由を説明せよ。
4. **直接識別最適化（DDO）**。DDO (K. Zheng, Chen, Chen, et al. 2025) は、尤度モデル q_θ を凍結された参照コピー q_{ref} に対してファインチューニングする手法であり、クリティックを対数尤度比 $D_\theta(x)=\log(q_\theta(x)/q_{\text{ref}}(x))$ として直接パラメータ化し、ロジスティック識別器損失 $J(\theta)=-\mathbb{E}_{x\sim p}[\log\sigma(D_\theta(x))]-\mathbb{E}_{x\sim q_{\text{ref}}}[\log\sigma(-D_\theta(x))]$ を $\theta=\theta_{\text{ref}}$ から出発して最小化する。
   1. $q_\theta=p$ ならば、$D_\theta(x)$ は最適クリティック $D^\star(x)$（p と q_{ref} の間のクリティック）に等しいことを示せ。
   2. $\mathbb{E}_{x\sim q_\theta}[\nabla_\theta\log q_\theta(x)]=0$ を用いて、初期化時点（$\theta=\theta_{\text{ref}}$、すなわち $D_\theta(x)=0$）において、勾配 $\nabla_\theta J(\theta)$ が $\frac{1}{2}$ 倍の負の対数尤度勾配 $-\nabla_\theta\mathbb{E}_{x\sim p}[\log q_\theta(x)]$ に等しいことを示せ。
5. **音声ボコーダにおける位相の曖昧さと波形平均化**。長さ N の単一の音声フレームを、正弦波 $x_n=\sin(\omega n+\vartheta)$（$n=0,\dots,N-1$）としてモデル化する。ここで周波数 ω はスペクトログラムから既知であり、位相 $\vartheta\sim\text{Uniform}[-a,a]$（$0<a\le\pi$）は未観測とする。
   1. 条件付き平均波形 $\mathbb{E}_\vartheta[x_n]=\frac{\sin a}{a}\sin(\omega n)$ の振幅が $\frac{\sin a}{a}$ 倍に減衰する（$a=\pi$ のとき無音へ消失する）ことを示せ。
   2. $\omega=2\pi k/N$（$0<k<N/2$）のとき、離散フーリエ変換の振幅 $|\hat{x}_m|$ が ϑ に依存しないことを示せ。

3. 隣接する 2 つのフレームが独立な位相 $\vartheta_1 \neq \vartheta_2$ で生成された場合、フレームレベルの DFT 振幅損失は各フレーム単体では 0 になる。このときフレーム境界でどのようなアーティファクトが生じるか。また、時間領域の波形識別器はなぜそれを検出できるのか。

16.8 関連文献

本節では、本章の中核となる概念（密度比推定、ダイバージェンスによる定式化、相対論的目的関数、勾配ペナルティ、敵対的事後学習）を、より広範な文献およびオープンソース実装へと結びつける。

16.8.1 推奨される入門文献とコードベース

1. **局所収束と勾配ペナルティ：**@Mescheder.Geiger.Nowozin.2018 は 16.4 節で引用した局所収束定理を証明し、重みクリッピングや 1 中心ペナルティ（WGAN-GP）が破綻する均衡点近傍において、ゼロ中心勾配ペナルティ（R_1 と R_2）が交互勾配更新を安定化することを示した。著者らの GAN stability リポジトリには、Dirac-GAN および R_1 と R_2 の正則化項の実装が含まれている。
2. **現代化された GAN ベースライン（R3GAN）：**@Huang.Gokaslan.Kuleshov.ea.2024 は、相対論的ペアリング損失を $R_1 + R_2$ ペナルティおよび現代的な畳み込みブロックと組み合わせ、適切に正則化された目的関数を用いれば初期の GAN で使われていた場当たり的なアーキテクチャ上の安定化手法の多くが不要になることを示した（R3GAN 公式リポジトリ）。
3. **評価指標（適合率、再現率、FID、KID）：**@Kynkaanniemi.Karras.Laine.ea.2019 は、サンプルの忠実度（fidelity）とモード被覆（mode coverage）を切り分ける k-NN の適合率・再現率指標（16.5.4.3 節）を導入した。StyleGAN3 リポジトリ (Karras et al. 2021) は、フルスケールの FID、KID、適合率、再現率のリファレンス実装を提供している。
4. **敵対的拡散蒸留：**DMD2 (Yin, Gharbi, Park, et al. 2024) は、分布整合スコア蒸留（distribution-matching score distillation）と実画像に対して訓練された識別器を組み合わせることで、1 ステップの生徒生成器が複数ステップの拡散教師モデルを上回る性能を達成できるようにしている。
5. **対話的な可視化と歴史的サーベイ：**NIPS 2016 チュートリアル (Goodfellow 2016) と ganhacks チェックリストには初期の GAN 訓練におけるヒューリスティクスがまとめられており、GAN Lab (Kahng et al. 2018) では 2D の生成器と識別器のダイナミクスをブラウザ上で可視化できる。また、大規模ベンチマークにおいて Lucic et al. (2018) は、同等のハイパーパラメータ調整予算を与えれば、多くの敵対的目的関数（16.2 節）が同程度の FID を達成することを実証している。

16.8.2 目的関数とミニマックスダイナミクス

- **最小二乗 GAN（LSGAN）の符号化スキーム：**16.2 節では、0-1 最小二乗識別器が $\frac{1}{4}\chi^2(p \,\|\, m)$ を評価することを示した。Mao et al. (2017) は一般の正解ラベル b（実データ）、a（偽データ）、および c（生成器の目標ラベル）を解析し、最適クリティックのもとで生成器の目的関数が $\frac{1}{4}(a+b-2c)^2 + \frac{1}{4}(b-a)^2\chi^2(p \,\|\, m)$ に帰着することを示した。これは同じ χ^2 ダイバージェンスの正の定数倍である（任意の $b \neq a$ に対して成り立つ）。
- **f-GAN の出力活性化関数：**クリティックの出力 t が凸共役（convex conjugate）$f^*(t)$ の有限となる定義域に収まるようにするため（16.2 節）、Nowozin et al. (2016) は定義域に応じた出力活性化関数 $t(v)$ をネットワークの生の出力 v に適用している。具体的には、順方向 KL には恒等写像、逆方向 KL には $-e^{-v}$（$t < 0$）、イェンセン・シャノンには $\log 2 - \mathrm{softplus}(-v)$（$t < \log 2$）を用いる。

- **一般の相対論的ダイバージェンス：**@Jolicoeur-Martineau.2020 は、相対論的ペアリング目的関数（16.3 節）の較正された最適値が、任意の凹利得関数 ℓ（ただし $\ell(0) = 0$、$\ell'(0) \neq 0$、$\sup_{t>0} \ell(t) > 0$ を満たすもの）に対して正当なダイバージェンスを定義することを証明している。
- **双線形ミニマックス問題に対する最適化アルゴリズム：**双線形問題 $\min_\theta \max_\psi \theta\psi$ において、ステップサイズ h の同時勾配降下法はステップごとに原点からの距離を $\sqrt{1+h^2}$ 倍に増幅させる。**外挿勾配**法（extragradient method）(Korpelevich 1976) は先読みした試行点で勾配を評価し、距離を $\sqrt{1-h^2+h^4} < 1$ 倍（$0 < h < 1$ のとき）へと縮小させる。**楽観的勾配降下法**（optimistic gradient descent）(Daskalakis et al. 2018) は直前ステップの勾配を用いてこの先読みを近似し、負のモメンタム（negative momentum）(Gidel et al. 2019) は交互更新を安定化する。これらの手法がオプティマイザを変更するのに対し、大規模な画像 GAN では代わりに R_1/R_2 勾配ペナルティによって目的関数を変更する（16.4 節）。

16.8.3 固定された差異と学習される差異

クリティックネットワークを用いずに、固定された 2 サンプル間距離に対して生成器を直接訓練することはできるだろうか。Generative Moment Matching Networks (Li et al. 2015) は固定カーネルの MMD（16.2 節）を最小化し、陰的最尤推定（Implicit Maximum Likelihood Estimation、IMLE）(Li and Malik 2018) はモード欠落を防ぐために各実データ例を最近傍の生成サンプルに対応させ、Sliced Wasserstein 生成器 (Deshpande et al. 2018) はランダム射影にわたって 1D の最適輸送距離を平均する。しかし自然画像においては、ピクセル空間の固定カーネルやランダム射影では知覚的な差異を捉えることが難しく、これが MMD-GAN (C.-L. Li et al. 2017; Bińkowski et al. 2018) における敵対的カーネル学習の動機となった。

16.8.4 アーキテクチャ、特徴空間、データ効率

- **アーキテクチャの進化（2015〜2021 年）：**DCGAN (Radford et al. 2015) から R3GAN (Huang et al. 2024) に至るまで、訓練の安定性と高解像度生成に向けたアーキテクチャ上の進歩が重ねられた。代表的なものとして、漸進的成長（progressive growing）(Karras et al. 2018)、SAGAN や BigGAN におけるスペクトル正規化（spectral normalization）と自己アテンション (Miyato et al. 2018; Han Zhang et al. 2019; Brock et al. 2019)、そして StyleGAN の潜在マッピングネットワーク、スタイル変調・復調、およびエイリアスフリーなリサンプリング (Karras et al. 2019, 2021; Karras, Laine, et al. 2020) が挙げられる。
- **特徴空間と評価指標の落とし穴：**@Kynkaanniemi.Karras.Aittala.ea.2023 は、Inception-v3 の特徴量が ImageNet のクラス境界に強く影響されているのに対し、自己教師あり学習による DINOv2 の特徴量は人間の知覚的判断とより良く整合することを示した (Stein et al. 2023; Karras, Aittala, Kynkäänniemi, et al. 2024)。さらに、エイリアシングを伴う画像リサイズは FID を歪めることがあり (Parmar et al. 2022)、有限サンプルにおける FID の統計的バイアスはモデル依存の傾きをもって $O(1/n)$ でスケールする (Chong and Forsyth 2020)。
- **クリティックにおける事前学習済み特徴：**Projected GAN (Sauer et al. 2021) と StyleGAN-T (Sauer, Karras, et al. 2023) は、凍結した事前学習済みの視覚バックボーンの上に軽量な識別器ヘッドを適用することで、収束を加速している。この手法を用いる場合、評価には必ず独立した特徴抽出器を用いるべきである。
- **勾配ペナルティの償却：**R_1 と R_2 はクリティックを通じた 2 次の誤差逆伝播法を必要とするため、StyleGAN2 は**遅延正則化**（lazy regularization、Karras, Laine, et al. (2020)）として 16 ミニバッチごとに 1 回だけペナルティを評価する。一方、大規模な動画拡散モデルの事後学習では、微小なガウス入力摂動のもとでの有限差分（finite difference）によって R_1 を近似している (Lin et al. 2025)。

- **非リーク型データ拡張：**小規模なデータセットでは、クリティックは訓練画像を急速に暗記してしまう（16.5 節）。Adaptive Discriminator Augmentation (ADA) (Karras, Aittala, et al. 2020) および DiffAugment (Zhao et al. 2020) は、勾配経路の内部で実画像と生成画像の両方に可逆かつ**非リーク**（non-leaking）な変換を適用することで、目標分布を損なうことなくクリティックの過学習を防ぐ。

16.8.5 条件付け、画像変換、現代のハイブリッドシステム

- **特徴変調層：**チャネルごとのアフィン変調 $a(c) \odot h + b(c)$（16.6 節）は、Conditional Batch Normalization (Vries et al. 2017)、FiLM (Perez et al. 2018)、スタイル変換のための Adaptive Instance Normalization (AdaIN) (Huang and Belongie 2017)、およびセマンティック画像生成のための空間的に変化する SPADE (Park et al. 2019) として現れる。
- **ペアあり・ペアなしの画像から画像への変換：**Pix2pix (Isola et al. 2017) は、U-Net 生成器とパッチレベルの識別器（PatchGAN）を組み合わせることで L_1 再構成を鮮鋭化する。CycleGAN (Zhu et al. 2017) はサイクル一貫性（cycle consistency、$F(G(x)) \approx x$）を課すことでペアなしドメインへと変換を拡張するが、ドメイン間でエントロピーが異なるとき、サイクル一貫性を持つ写像は入力元の情報を高周波のステガノグラフィ的摂動の中に隠蔽してしまうことがある (Chu et al. 2017)。
- **トークナイザ、少数ステップ蒸留、ボコーダ：**視覚トークナイザ（VQGAN、ViTok）では、デコーダの敵対的損失によってピクセルごとの PSNR と引き換えに知覚的な再構成品質を向上させる (Esser et al. 2021; Hansen-Estruch et al. 2025, 2026)。拡散蒸留においては、LADD (Sauer et al. 2024)（FLUX.1-schnell で採用、Black Forest Labs (2024)）および SDXL-Lightning (Lin, Wang, et al. 2024) が潜在空間またはパッチ空間で直接識別器を適用する一方、MeanFlow (Geng et al. 2025) や連続時間一貫性モデル（continuous-time consistency model）(Lu and Song 2025) はクリティック不要の少数ステップ回帰目的関数を探求している。音声合成では、RAF (Lee and Choi 2026) が BigVGAN (Lee et al. 2023) を 16.3 節の相対論的ペアリング目的関数で改良している。
- **陰的クリティックとしての尤度モデル（DDO）：**Direct Discriminative Optimization (K. Zheng, Chen, Chen, et al. 2025) は、尤度モデルまたは拡散モデル q_θ を凍結した参照コピー q_{ref} に対してファインチューニングする際、クリティック自体を対数尤度比 $D_\theta(x) = \beta \log(q_\theta(x)/q_{\text{ref}}(x))$ としてパラメータ化することで、別個の識別器ネットワークを不要にしている（演習問題 4）。

Chapter 17

拡散モデル

第 16 章の敵対的な生成器（generator）は、ノイズをデータへ写像する生成器に直接的な回帰目標（regression target）がないため、識別器（discriminator）またはクリティックという 2 つ目のネットワークを必要とした。拡散モデル（diffusion model）とフローマッチングモデルは、各訓練例をガウスノイズで段階的に破壊することによって回帰目標を構築する。小さなガウスノイズのステップを逆転させるには、ノイズを含む観測が与えられたもとでのクリーンなデータの条件付き期待値（conditional expectation）を推定する必要がある。

この視点は、非平衡熱力学とノイズ除去拡散確率モデル（denoising diffusion probabilistic model、DDPM）(Sohl-Dickstein et al. 2015; Ho et al. 2020)、スコアベース生成モデリング（score-based generative modeling）(Song and Ermon 2019; Y. Song, Sohl-Dickstein, et al. 2021)、およびフローマッチングや確率的補間（stochastic interpolant）(Lipman et al. 2023; Liu et al. 2023; Albergo et al. 2023) という 3 つの研究系譜を統合するものである。これら 3 つの定式化のいずれにおいても、単一のニューラルネットワークがクリーンなデータ（または等価的に、付加されたノイズ、スコア関数 $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$、あるいは経路に沿った速度）をあらゆるノイズレベル $t \in [0, 1]$ で予測するように学習し、数値サンプラがその予測を $t = 1$ の純粋なノイズから $t = 0$ のデータへと逆向きに積分する。

敵対的なミニマックス目的関数を回帰に置き換えることで、GAN の 3 つの限界が解消される。すなわち、訓練に必要なのは二乗誤差で最適化される単一のネットワークのみであり、重み付き回帰損失は対数尤度（log-likelihood）の変分境界（さらに確率フロー ODE を通じた厳密な対数尤度）を与え、尤度目的関数は欠落したモードにペナルティを科す。本章全体を通じて、デフォルトの定式化では DDPM における分散保持（variance-preserving）経路上のノイズ予測器と並行して、線形経路 $\mathbf{x}_t = (1 - t)\mathbf{x}_0 + t\epsilon$ に沿った速度を回帰する (Esser et al. 2024)。

最初の 3 節では、真の密度、スコア関数、事後平均（posterior mean）、速度場（velocity field）が閉形式で得られる 3 つのガウス分布の 2 次元混合分布を用いて、数学的・アルゴリズム的な中核を完全に構築する。17.4 節では Fashion-MNIST 画像へと進み、時間条件付き U-Net、フレシェ距離（Fréchet distance）による評価、および分類器なしガイダンス（classifier-free guidance）を導入する。続く 4 節では、サンプラ（17.5 節）、尤度境界と設計空間（17.6 節）、潜在拡散（latent diffusion）と Transformer バックボーン（17.7 節）、およびフロー写像（flow map）による少数ステップ蒸留（17.8 節）を検討する。最後に 17.9 節では、ガウスノイズをトークンマスキングに置き換えることで、拡散を離散トークンへと拡張する。より長い数学的証明は第 38 章に掲載されており、外部のチュートリアル、講義、参照実装は 17.10 節にまとめられている。

17.1 ノイズ除去とスコア

二乗誤差で訓練されたネットワークは、入力が与えられたときの回帰目標の条件付き期待値（conditional expectation）を学習する（16.7 節）。したがって、クリーンなデータ点 $\mathbf{x}_0 \sim p$ に等方的なガウスノイズ $\sigma\epsilon$ を加えて $\tilde{\mathbf{x}} = \mathbf{x}_0 + \sigma\epsilon$ を作るとき、最小二乗法の意味で最適なノイズ除去器（denoiser）は事後平均（posterior mean）$\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}]$ となる。**Tweedie の公式** (Robbins 1956; Miyasawa 1961; Efron 2011) によれば、この事後平均はノイズを加えたデータの対数密度（log-density）の勾配 $\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$、すなわち**スコア**関数（score function）によって直接定まる。さらに Vincent (2011) は、ガウス摂動カーネルのスコアへネットワークを回帰させることで、$\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ を元のデータ密度 p にアクセスすることなく近似できることを示した。

密度そのものではなくスコアを学習すれば、エネルギーベースモデルで必要となる計算困難な正規化定数（normalizing constant）を回避しつつ、ランジュバン動力学（Langevin dynamics）などの勾配ベース MCMC によってサンプルを生成するのに必要なベクトル場（vector field）が得られる。本節ではまず、あらゆるノイズレベルにおいて密度、スコア、事後平均が厳密に求まる 2 次元の混合ガウス分布（Gaussian mixture）を導入する。このベンチマークを用いて Tweedie の公式を検証し、単一のノイズレベル σ で訓練されたスコアモデルが局所的なモード構造と大域的なモード重みを同時に捉えられない理由を調べたうえで、最後に単一レベルのノイズ除去を Fashion-MNIST 画像に適用する。

17.1.1 通し例

学習された各ベクトル場やサンプラを真の値と照合できるよう、本章で扱うすべての量があらゆるノイズレベルで厳密に求まる分布を 1 つ用意したい。それには平面上の混合ガウス分布で十分である。ここでは、平均が $(-2.5, -1.5)$、$(2.5, -1.5)$、$(0, 2.5)$、混合重みが $\pi_k = 0.5, 0.3, 0.2$、座標ごとの共通の標準偏差が $s = 0.5$ である 3 つの成分をもつ分布を用いる。各モードはその幅に比べて互いに遠く離れているため、モード間の密度には深い谷があり、さらに不均等な重みによってサンプラが誤りやすい条件が揃っている。

密度 p の**スコア**とはその対数の勾配 $\nabla \log p(\mathbf{x})$ であり、密度が最も急激に増加する方向を指すベクトル場である。混合分布 $p(\mathbf{x}) = \sum_k \pi_k \, \mathcal{N}(\mathbf{x}; \mu_k, s^2\mathbf{I})$ に対して、スコアは次のように書ける。

$$\begin{aligned} \nabla \log p(\mathbf{x}) &= \sum_k r_k(\mathbf{x}) \, \frac{\mu_k - \mathbf{x}}{s^2}, \\ r_k(\mathbf{x}) &= \frac{\pi_k \, \mathcal{N}(\mathbf{x}; \mu_k, s^2\mathbf{I})}{p(\mathbf{x})}. \end{aligned} \tag{17.1}$$

各成分は点をそれぞれの平均へと引き寄せ、その引力は**負担率**（responsibility）$r_k(\mathbf{x})$、すなわち $\mathbf{x}$ が成分 k から抽出された事後確率によって重み付け平均される（Figure 17.1）。重み π_k はこの負担率を通じてのみ式に現れる。モードの近くでは重みの値にかかわらず 1 つの負担率がほぼ 1 になるため、重みがスコアを変化させるのはデータ点がほとんど存在しない谷の部分だけである。一見すると何気ない性質だが、これがスコアモデルに学習できることを決定づけることになる。

ノイズを加えても、混合ガウス分布は混合ガウス分布のままである。抽出されたサンプルに標準偏差 σ のガウスノイズを加えると各成分の分散に σ^2 が加算され、s^2 は $s^2 + \sigma^2$ に置き換わるため、あらゆるノイズレベルにおいて密度、スコア、事後平均が閉じた式で求まる。以下のクラスはこれら 3 つをすべて計算する。クリーンなデータ点に対するスケールを表す引数 `alpha` は、次節まで 1 に固定しておく。

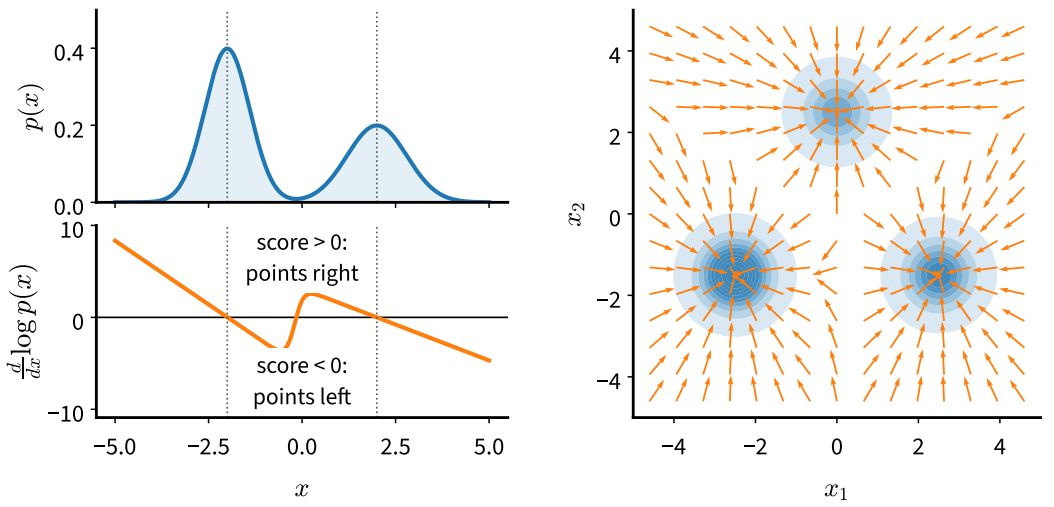


Figure 17.1: スコアは対数密度の勾配である。左：1 次元の二峰性密度（上）とそのスコア（下）。両方のモードと、その間の谷で 0 になる。右：通し例。陰影付きの等高線が密度を、モードへ向かう矢印がスコア場を表す（矢印の長さはクリップしてある）。

```
class GaussianMixture:
    """A mixture of isotropic Gaussians, exact at every noise level."""
    def __init__(self, means, weights, std):
        self.means = jnp.array(means, dtype=jnp.float32)
        self.weights = jnp.array(weights, dtype=jnp.float32)
        self.std = std

    @staticmethod
    def three_modes():
        """The chapter's running example, with mode weights 0.5/0.3/0.2."""
        return GaussianMixture([[-2.5, -1.5], [2.5, -1.5], [0.0, 2.5]],
                               [0.5, 0.3, 0.2], 0.5)

    def sample(self, key, n):
        """n draws from the mixture."""
        k1, k2 = jax.random.split(key)
        k = jax.random.choice(k1, len(self.weights), (n,), p=self.weights)
        return self.means[k] + self.std * jax.random.normal(
            k2, (n, self.means.shape[1]))

    def _modes(self, x, alpha, sigma):
        """Per-mode log densities, offsets x - alpha mu_k, variances, alpha."""
        a, s = (jnp.asarray(v, x.dtype).reshape(-1, 1, 1)
                for v in (alpha, sigma))
        var = (a * self.std) ** 2 + s ** 2
        r = x[:, None] - a * self.means
        log_p = (jnp.log(self.weights) - (r ** 2).sum(-1) / (2 * var[..., 0])
                 - x.shape[1] / 2 * jnp.log(2 * jnp.pi * var[..., 0]))
        return log_p, r, var, a
```

```
    def log_prob(self, x, alpha=1.0, sigma=0.0):
        """log p(x) for x = alpha x0 + sigma eps."""
        return jax.nn.logsumexp(self._modes(x, alpha, sigma)[0], 1)

    def score(self, x, alpha=1.0, sigma=0.0):
        """The gradient of log p(x) for x = alpha x0 + sigma eps."""
        log_p, r, var, _ = self._modes(x, alpha, sigma)
        return -(jax.nn.softmax(log_p, 1)[..., None] * r / var).sum(1)

    def posterior_mean(self, x, alpha, sigma):
        """E[x0 | x]; finite at alpha = 0 and at sigma = 0."""
        log_p, r, var, a = self._modes(x, alpha, sigma)
        return (jax.nn.softmax(log_p, 1)[..., None]
                * (self.means + a * self.std ** 2 / var * r)).sum(1)
```

本章を通して、この混合分布に対するサンプラは同じ基準で評価する。すなわち、真の密度のもとでの生成点の平均負の対数尤度と、`summary` が返す 3 つの数値である。**モード重み**（mode weights）は、各点を最も近い平均に割り当てて集計した割合である。**外れ点**（strays）は、どの平均からも $3s = 1.5$ を超えて離れた点の割合を表す。**広がり**（spread）は、残りの点から最も近い平均までの平均二乗距離である。外れ点を除外するため、この値が $(3s)^2 = 2.25$ を超えることはなく、極端に広がりすぎたサンプラは主に外れ点の増加として現れる。

```
@d2l.add_to_class(GaussianMixture)
def nll(self, x):
    """Mean negative log-likelihood of clean points, in nats."""
    return -self.log_prob(x).mean()

@d2l.add_to_class(GaussianMixture)
def summary(self, x):
    """Mode weights by nearest mean, stray fraction, spread of the rest."""
    d2 = ((x[:, None] - self.means) ** 2).sum(-1)
    near, k = d2.min(1), d2.argmin(1)
    inside = near < (3 * self.std) ** 2
    weights = jnp.bincount(k, length=len(self.weights)) / len(x)
    return dict(weights=[round(float(w), 3) for w in weights],
                stray=round(1 - float(inside.mean()), 4),
                spread=round(float(near[inside].mean()), 3))
```

これらの数値要約とあわせて 2D 点群を視覚的に比較するため、`scatter_panels` は共通の座標軸上に 1 つ以上のサンプル集合を並べてプロットする（任意で参照データセット `data` を薄い灰色で重ねて表示できる）。

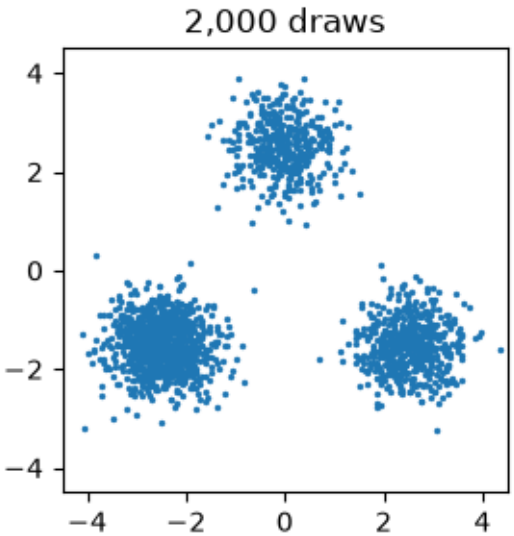
```
def scatter_panels(samples, titles=None, data=None, lim=4.5,
                   size=3.0):
    """Scatter 2-D point sets side by side on equal, shared axes."""
    _, axes = d2l.plt.subplots(1, len(samples), squeeze=False,
                               figsize=(size * len(samples), size))
    for i, (ax, pts) in enumerate(zip(axes[0], samples)):
        if data is not None:
            ax.scatter(*d2l.numpy(data).T, s=2, c='lightgray')
```

```
        ax.scatter(*d2l.numpy(pts).T, s=2)
        ax.set(xlim=(-lim, lim), ylim=(-lim, lim), aspect='equal')
        if titles:
            ax.set_title(titles[i])
    return axes[0]
```

完璧なサンプラはどのような値を出力するだろうか。サンプリング誤差を除けば、重みは 0.5、0.3、0.2 になる。2 次元では、ガウス分布からのサンプルと平均との二乗距離は平均 $2s^2 = 0.5$ の指数分布に従うため、$e^{-(3s)^2/(2s^2)} = e^{-4.5} \approx 1\%$ の割合の点が外れ点となり、残りの点の広がりは 0.5 をわずかに下回る。本節の 2 次元実験の訓練セットとなる 2,000 個のサンプルでこれを確認しよう。

```
mix = d2l.GaussianMixture.three_modes()
data = mix.sample(jax.random.key(0), 2000)
print(mix.summary(data), f'NLL {mix.nll(data):.3f}')
d2l.scatter_panels([data], ['2,000 draws']);
```

```
{'weights': [0.495, 0.293, 0.212], 'stray': 0.015, 'spread': 0.491} NLL 2.545
```



出力された値は予測どおりである。これが、この混合分布に対するすべてのサンプラが目指すべき基準値となる。負の対数尤度については一言注意が必要である。1 点あたり約 2.5 nat という値はこの混合分布のエントロピーであり、いかなる密度モデルも未知データに対して平均でこれより低い値を出すことはできず、正しいサンプラが生成した点も p のもとで同程度の値になる。サンプラにとってこの値は一致させるべき目標であって、下回るほどよいという限界値ではない。モードの中心付近に密集した点はこれより小さな値になり、谷にある点は大きな値になるからである。

17.1.2 最適ノイズ除去器

冒頭で述べた回帰問題には、クリーンなデータ点そのものと、そこへ向かうベクトルという 2 つの自然な回帰目標がある。それぞれの最小点（minimizer）を結びつける恒等式こそ、本節で最

も重要な結果である。

17.1.2.1 ノイズと回帰目標

ノイズレベル $\sigma > 0$ を選び、クリーンな点 $\mathbf{x}_0 \sim p$ と標準ガウスノイズ $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ を抽出して、ノイズ付きの点 $\tilde{\mathbf{x}} = \mathbf{x}_0 + \sigma\epsilon$ を作る。その密度 p_σ はデータ密度を幅 σ のガウス分布でぼかしたものであり、たとえ p がそうでなくても至るところ正かつ滑らかになる。ノイズ除去器 D は $\mathbb{E}\|D(\tilde{\mathbf{x}}) - \mathbf{x}_0\|^2$ を最小化するよう訓練する。本節の冒頭で確認したとおり（16.7 節）、二乗誤差は入力が与えられたときの目標の条件付き平均によって最小化されるため、最適なノイズ除去器は**事後平均** $\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}]$ となる。すなわち、$\tilde{\mathbf{x}}$ を生成し得たクリーンな点を、それぞれがそれを生成した尤もらしさで重み付けした平均である。

同じ回帰問題で、別の目標を予測させることもできる。クリーンな点が与えられたとき、ノイズ付きの点はその周囲のガウス分布に従い、この条件付き密度のスコアは次のようになる。

$$\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_0) = \frac{\mathbf{x}_0 - \tilde{\mathbf{x}}}{\sigma^2} = -\frac{\epsilon}{\sigma}, \tag{17.2}$$

これはノイズ付きの点から元のクリーンな点へと向かうベクトルである。**ノイズ除去スコアマッチング**（denoising score matching）では、ベクトル場 $\mathbf{s}_\theta$ をこの目標へ回帰させる (Vincent 2011)。

$$J_{\text{DSM}}(\theta) = \tfrac{1}{2}\, \mathbb{E}_{\mathbf{x}_0, \epsilon} \left\| \mathbf{s}_\theta(\mathbf{x}_0 + \sigma\epsilon) + \frac{\epsilon}{\sigma} \right\|^2. \tag{17.3}$$

その最小点もやはり条件付き平均（今度は条件付きスコアの条件付き平均）であり、Vincent の定理はその正体を次のように特定する。すなわち、**ノイズを加えたデータのスコアは、条件付きスコアの事後平均に等しい**。

$$\mathbb{E}\Big[-\frac{\epsilon}{\sigma} \,\Big|\, \tilde{\mathbf{x}} \Big] = \nabla \log p_\sigma(\tilde{\mathbf{x}}). \tag{17.4}$$

証明はベイズの定理を 2 行使うだけで済む。ノイズ付きの密度はクリーンな密度をノイズでぼかした $p_\sigma(\tilde{\mathbf{x}}) = \int p(\mathbf{x}_0)\, p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_0)\, d\mathbf{x}_0$ である。積分記号の下で微分し、カーネルの勾配を「カーネルとその対数の勾配の積」に書き換えて、両辺を $p_\sigma(\tilde{\mathbf{x}})$ で割ると次の式を得る。

$$\nabla \log p_\sigma(\tilde{\mathbf{x}}) = \int \frac{p(\mathbf{x}_0)\, p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_0)}{p_\sigma(\tilde{\mathbf{x}})}\, \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}_0)\, d\mathbf{x}_0.$$

被積分関数中の分数は事後分布 $p(\mathbf{x}_0 \mid \tilde{\mathbf{x}})$ に対するベイズの定理そのものであるから、右辺は条件付きスコアの事後平均となり、Equation 17.4 が得られる。この 2 行の導出ではノイズがガウス分布であることは一切使っていない。勾配を積分記号の中へ入れられる条件については 38.6.1.2 節で述べる。

したがって、ペアごとの目標に対する回帰によって、明示的に書き下すことすらなかった p_σ のスコアを学習できる。これこそが、本章で生成器を回帰によって訓練できる理由である。データのスコアは直接評価できなくても、自分で加えたノイズのスコアなら評価できるからである。

17.1.2.2 Tweedie の公式

クリーンな点への回帰と $-\epsilon/\sigma$ への回帰という 2 つの回帰問題では、それぞれの目標が写像 $\mathbf{x}_0 \mapsto (\mathbf{x}_0 - \tilde{\mathbf{x}})/\sigma^2$ で結ばれている。この写像は $\tilde{\mathbf{x}}$ を固定すればアフィン写像であるため、両者の最小点である 2 つの条件付き平均も同じ写像で結ばれる。これを事後平均について解くと、**Tweedie の公式** (Efron 2011) が得られる。

$$\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla \log p_\sigma(\tilde{\mathbf{x}}). \tag{17.5}$$

最良のノイズ除去器は、ノイズ付き密度のスコアの方向にステップサイズ σ^2 で 1 ステップ進む（Figure 38.23）。したがって、ノイズ除去器、スコア、ノイズ予測器（noise predictor）は、1 つの事後平均を表す 3 通りの書き方にほかならない。

$$\begin{aligned} \hat{\mathbf{x}}_0(\tilde{\mathbf{x}}) &= \tilde{\mathbf{x}} + \sigma^2\, \hat{\mathbf{s}}(\tilde{\mathbf{x}}), \\ \hat{\epsilon}(\tilde{\mathbf{x}}) &= \frac{\tilde{\mathbf{x}} - \hat{\mathbf{x}}_0(\tilde{\mathbf{x}})}{\sigma} = -\sigma\, \hat{\mathbf{s}}(\tilde{\mathbf{x}}). \end{aligned} \tag{17.6}$$

これら 3 つのうちどれか 1 つを出力するよう訓練されたネットワークがあれば、残りの 2 つも直ちに得られる。

最も単純な例を見れば、このステップの働きがわかる。データがガウス分布 $\mathcal{N}(\mu, v\mathbf{I})$ に従うとする。このとき $p_\sigma = \mathcal{N}(\mu, (v+\sigma^2)\mathbf{I})$ のスコアは $-(\tilde{\mathbf{x}} - \mu)/(v+\sigma^2)$ であり、Tweedie の公式から次式が得られる。

$$\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}] = \mu + \frac{v}{v+\sigma^2}\,(\tilde{\mathbf{x}} - \mu).$$

このノイズ除去器は、平均からのノイズを含んだずれに係数 $v/(v+\sigma^2)$ を掛けて縮小（shrinkage）し、座標ごとの事後分散（posterior variance）は $v\sigma^2/(v+\sigma^2)$ となる。$v = s^2 = 0.25$、$\sigma = 0.3$ のとき縮小係数は 0.74 であり、座標ごとの二乗誤差は $\sigma^2 = 0.09$ から 0.066 へと減少する。通し例においても、谷から離れた場所での事後平均は最も近い平均に向けたこの縮小に一致するため、点の 2 座標合計では、ノイズ除去によって二乗誤差が 0.18 から約 0.13 に下がるはずである。

17.1.2.3 訓練されたスコア

学習されたスコアは、この最良のノイズ除去器に到達できるだろうか。それを確かめるにはネットワークが必要であり、本章ではすべての 2 次元実験に 1 つの小さな MLP を用いる。このネットワークは点の座標に加え、ノイズレベルの正弦と余弦（次節以降では、データからノイズへの経路に沿った時刻 $t \in [0,1]$ の正弦と余弦）を入力として受け取る。本節ではノイズレベルが固定されているため、これらの特徴量は何の働きもしない。次節以降では 1 つのネットワークがあらゆるレベルを担当し、近接するレベルを識別しなければならないため、周波数は t の単位あたり 1 から 64 ラジアンまで変化し、最も速い特徴量は $[0,1]$ 上で約 10 周期分振動する。数千規模の整数トークン位置向けに設計された 10.4 節の位置エンコーディングでは、この区間上でほぼ定数または線形になってしまう。なお、17.8 節の少数ステップ生成器では再び周波数を下げるが、その理由は同節で述べる。

```
def time_embedding(t, dim, freqs=(1.0, 64.0)):
    """Sines and cosines of t at dim / 2 frequencies, geometric from freqs[0]
    to freqs[1] radians per unit of t (t in [0, 1])."""
    f = jnp.exp(jnp.linspace(math.log(freqs[0]), math.log(freqs[1]), dim // 2))
```

```
    args = jnp.asarray(t, jnp.float32)[:, None] * f
    return jnp.concatenate([jnp.sin(args), jnp.cos(args)], 1)

class TimeMLP(nnx.Module):
    """An MLP on a point x and the time features of one or more times;
    time_freqs: see time_embedding."""
    def __init__(self, hidden=256, depth=3, *, dim=2, num_times=1,
                 time_dim=64, time_freqs=(1.0, 64.0), rngs):
        self.time_dim, self.time_freqs = time_dim, time_freqs
        self.embed = d2l.make_mlp([time_dim * num_times, hidden, hidden],
                                  nnx.silu, rngs=rngs)
        self.inp = d2l.Linear(dim, hidden, rngs=rngs)
        self.body = nnx.Sequential(  # depth layers, each after a SiLU
            nnx.silu,
            d2l.make_mlp([hidden] * depth + [dim], nnx.silu, rngs=rngs))

    def __call__(self, x, *ts):
        feats = [time_embedding(t, self.time_dim, self.time_freqs) for t in ts]
        return self.body(self.inp(x) + self.embed(jnp.concatenate(feats, 1)))
```

訓練ループも本章全体で共通のものを用いる。各ステップで新たなミニバッチと新たなノイズを生成し、Adam で更新する。さらに重みの指数移動平均（exponential moving average、EMA）も保持するが、これが重要になるのは 17.4 節以降である。その他のオプションについては docstring に記載してある。

```
def train_regression(model, loss_fn, batch_fn, num_steps, key, *,
                     lr=1e-3, cosine=False, ema_half_life=1000, chunk=500):
    """Adam on loss_fn(model, batch_fn(k1), k2) -> model, EMA model, losses.

    The learning rate is constant or, with cosine=True, decays to zero along
    a cosine. The EMA model averages the weights (d2l.EMA): the weights of
    ema_half_life updates ago count half as much as the current ones. Each
    chunk of updates is one jitted scan, a single dispatch to the device:
    the networks of this chapter are so small that dispatching their
    updates one at a time would cost more time than running them."""
    schedule = optax.cosine_decay_schedule(lr, num_steps) if cosine else lr
    opt = nnx.Optimizer(model, optax.adam(schedule), wrt=nnx.Param)
    ema = d2l.EMA(model, ema_half_life)

    @nnx.jit
    def run(model, opt, avg, batches, keys):
        @nnx.scan(in_axes=(nnx.Carry, 0, 0), out_axes=(nnx.Carry, 0))
        def step(carry, batch, key):
            model, opt, avg = carry
            loss, grads = nnx.value_and_grad(loss_fn)(model, batch, key)
            opt.update(model, grads)
            ema.average(avg, model, ema.decay)    # on the carried copy
            return (model, opt, avg), loss
        return step((model, opt, avg), batches, keys)[1]

    chunks = [min(chunk, num_steps - i)           # steps per jitted dispatch
              for i in range(0, num_steps, chunk)]
```

```
    losses = []
    for key, n in zip(jax.random.split(key, len(chunks)), chunks):
        k1, k2 = jax.random.split(key)            # batches drawn outside jit
        batches = jax.vmap(batch_fn)(jax.random.split(k1, n))
        losses.append(run(model, opt, ema.model, batches,
                          jax.random.split(k2, n)))
    return model, ema.model, jnp.concatenate(losses)
```

混合分布に対しては、隠れ層 2 層（各 128 ユニット）のネットワークを用い、2,000 個の訓練点から毎回 256 個を選んで新たなノイズを加えながら Equation 17.3 を 2,000 ステップ最小化し、訓練されたスコアを点のみの関数として保持する。

```
def dsm_loss(net, x, key, sigma):
    """Denoising score matching at one noise level: regress onto -eps/sigma."""
    eps = jax.random.normal(key, x.shape)
    s = net(x + sigma * eps, jnp.full((len(x),), sigma))
    return 0.5 * ((s + eps / sigma) ** 2).sum(1).mean()

def fit_score(data, sigma, seed=1, num_steps=2000):
    """Train a score network at one noise level and return it as a function."""
    batch_fn = lambda key: data[jax.random.randint(key, (256,), 0, len(data))]
    net, _, _ = d2l.train_regression(
        d2l.TimeMLP(128, 2, rngs=nnx.Rngs(seed)),
        lambda net, x, key: dsm_loss(net, x, key, sigma), batch_fn,
        num_steps, jax.random.key(seed))
    return lambda x: net(x, jnp.full((len(x),), sigma))
```

$\sigma = 0.3$ において新たに生成した 20,000 個の点にノイズを加え、クリーンな点に対する 3 通りの推定値、すなわちノイズ付きの点そのもの、厳密な事後平均、そして訓練されたスコアを用いた Tweedie のステップを比較する。

```
sigma = 0.3
score = fit_score(data, sigma)
k1, k2 = jax.random.split(jax.random.key(1))
x0 = mix.sample(k1, 20000)
xt = x0 + sigma * jax.random.normal(k2, x0.shape)
exact = mix.posterior_mean(xt, 1.0, sigma)
for name, x0_hat in (('noisy point', xt), ('posterior mean', exact),
                     ('Tweedie, trained score', xt + sigma ** 2 * score(xt))):
    print(f'{name}: squared error {((x0_hat - x0) ** 2).sum(1).mean():.4f}')
```

```
noisy point: squared error 0.1806
posterior mean: squared error 0.1331
Tweedie, trained score: squared error 0.1340
```

ガウス分布の場合の予測どおり、厳密な事後平均は二乗誤差を 0.18 から約 0.13 へと引き下げ、訓練されたスコアによる Tweedie のステップもその約 1% 以内に迫っている。2,000 個の点に対する 2,000 ステップの訓練を経て、ネットワークはほぼ最適なノイズ除去器となり、したがって実用的なスコアとなっている。

もっとも、スコアとして見れば見かけほど優秀ではない。スコアは $\sigma^2 = 0.09$ 倍されてノイズ除去器に入るため、Tweedie のステップが事後平均に対して生じる過剰誤差の期待値は、スコアの

平均二乗誤差のちょうど σ^4 倍になる（$\tilde{\mathbf{x}}$ が与えられたとき事後平均の誤差の期待値は 0 になるため、交差項は消える）。したがって、約 0.001 というわずかな差は、スコア誤差 $\frac{1}{2}\mathbb{E}\|\hat{\mathbf{s}} - \nabla \log p_\sigma\|^2$ に換算すると約 0.06 に相当する。ノイズ除去器としてはほぼ完璧に見えても、そのスコアは目に見えてずれていることがあり、そしてサンプラが従うのはスコアのほうなのである。

17.1.2.4 スコアと正規化定数

なぜ密度ではなくスコアを学習するのか。**エネルギー**（energy）と呼ばれるスカラー出力 E_θ をもつネットワークは、**エネルギーベースモデル**（energy-based model）を通じて次のように密度を定める。

$$p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z_\theta}, \qquad Z_\theta = \int \exp(-E_\theta(\mathbf{x}))\, d\mathbf{x}. \tag{17.7}$$

正規化定数があるからこそ学習に意味が生じる。全体の確率質量が固定されているため、訓練点における密度を高めれば他の場所での密度が下がるからである。ところが残念なことに、Z_θ は入力空間全体にわたる積分であってパラメータ更新のたびに変化し、対数尤度の勾配には $\nabla_\theta \log Z_\theta = -\mathbb{E}_{p_\theta}[\nabla_\theta E_\theta]$ が含まれる。この期待値はモデルからのサンプリング、実際にはマルコフ連鎖を用いて推定しなければならない。一方、スコアには Z_θ が一切現れない。Z_θ は $\mathbf{x}$ に依存しないため、$\nabla_{\mathbf{x}} \log p_\theta = -\nabla_{\mathbf{x}} E_\theta$ となるからである。

ベクトル場 $\mathbf{s}$ とデータとの比較には、データが存在する場所における 2 つの場の平均二乗距離である**フィッシャーダイバージェンス**（Fisher divergence）を用いる。

$$D_{\mathrm{F}}(p, \mathbf{s}) = \tfrac{1}{2}\, \mathbb{E}_{\mathbf{x}\sim p}\big\|\mathbf{s}(\mathbf{x}) - \nabla \log p(\mathbf{x})\big\|^2. \tag{17.8}$$

この式には未知のデータスコアが含まれるが、**Hyvärinen の恒等式** (Hyvärinen 2005) は部分積分（integration by parts）によってこれを取り除く。すなわち定数項を除けば、D_{F} は正規化定数もマルコフ連鎖も不要なデータ上の平均 $\mathbb{E}_{p}\big[\tfrac{1}{2}\|\mathbf{s}\|^2 + \nabla \cdot \mathbf{s}\big]$ に等しい（38.6.1 節）。しかし第 2 項の $\nabla \cdot \mathbf{s}$（$\mathbf{s}$ のヤコビ行列のトレース）には入力次元ごとに 1 回の逆伝播が必要であり、画像に対してはコストが大きすぎる。ノイズ除去はこのコストも解消する。Vincent の定理により、Equation 17.3 は定数項を除いて p_σ に対するフィッシャーダイバージェンスに一致するからである。

17.1.3 単一ノイズレベルの限界

1 つのノイズレベルで訓練されたノイズ除去器から得られるのは、1 つのぼかされた分布 p_σ のスコアである。p からサンプリングするには p のスコアが必要であるため、最初の問いはどの σ を選ぶべきかということになる。

17.1.3.1 ノイズレベルの選択

ノイズレベルが小さい場合には 2 つの問題が生じる。1 つ目は統計的な問題である。通し例のように滑らかな密度をもつデータでは、回帰目標 $-\epsilon/\sigma$ の大きさは $1/\sigma$ のオーダーで増大する一方、その平均であるスコアは有界にとどまるため、小さな σ では目標の大部分がノイズとなってネットワークの当てはまりが悪くなる。たとえ厳密なスコアを用いても、条件付き平均の周りでの目標の分散が損失に残り、d 次元ではこの損失の下限は d/σ^2 のオーダーで σ の減少とともに増大する（38.6.1.2 節）。したがって、ノイズ除去損失が 0 よりかなり高い値で頭打ちになっても、ネットワークの性能が悪いことを意味するわけではない。

2 つ目の問題は、ピクセル空間内の低次元集合の近くに分布していると考えられる画像においてとりわけ重要になる (Bengio, Courville, et al. 2013)。極端な例として、平面上の 1 本の直線上に乗っているデータを考えよう。ノイズによって直線は帯状に広がり、帯を横切る方向の p_σ は分散 σ^2 のガウス分布になるため、帯を横切る方向のスコアは $-\tilde{x}_2/\sigma^2$（直線からの距離を $\tilde{x}_2$ とする）となる（演習問題 3)。その傾き $-1/\sigma^2$ は $\sigma \to 0$ で際限なく増大する。帯を横切る方向では目標 $-\epsilon_2/\sigma$ がこのスコアに厳密に一致するため、そこでの困難はノイズではない。ネットワークは、データ集合を横切る方向には断崖絶壁へと化す場を表現しつつ、データ集合に沿う方向ではほとんどノイズばかりの目標を扱わなければならないのである。

ノイズレベルが大きい場合には逆の問題が生じる。回帰問題の条件は良くなるが、得られる答えは p_σ のスコアであって p のスコアではない。ガウス成分 $\mathcal{N}(\mu, s^2\mathbf{I})$（$d$ 次元）が 1 つの場合、2 つのスコアは $-(\mathbf{x}-\mu)/(s^2+\sigma^2)$ と $-(\mathbf{x}-\mu)/s^2$ であり、その差の二乗ノルムを p で平均すると次のバイアスが得られる。

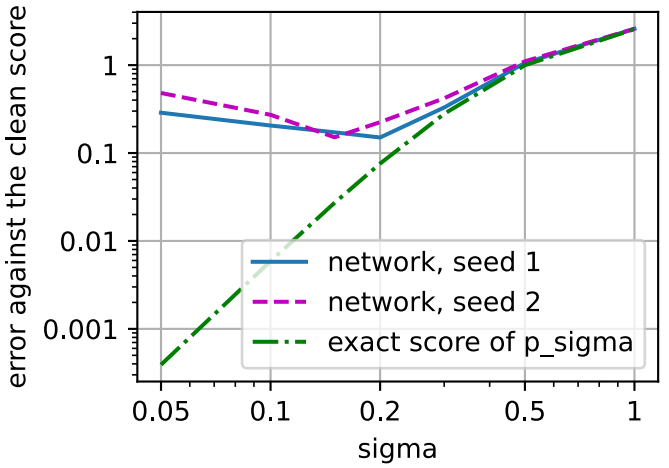
$$\tfrac{1}{2}\,\mathbb{E}_p\big\|\nabla \log p_\sigma - \nabla \log p\big\|^2 = \frac{d\,\sigma^4}{2s^2(s^2+\sigma^2)^2}.$$

このバイアスは最初 σ^4 のオーダーで増大する。$s = 0.5$、$d = 2$ のとき、その値は 0.0004 ($\sigma = 0.05$)、0.28 ($\sigma = 0.3$)、2.6 ($\sigma = 1$) となり、モードがよく分離した混合分布でもほぼ同じ値になる。クリーンなスコアの大きさ $\frac{1}{2}\mathbb{E}_p\|\nabla \log p\|^2 = d/(2s^2)$ で割ると $(\sigma^2/(s^2+\sigma^2))^2$ となり、これは σ/s のみに依存する。平滑化されたスコアが二乗平均平方根 (root mean square) の意味でクリーンなスコアの 10% 以内に収まるのは $\sigma < s/3 \approx 0.17$ の範囲であるため、このあたりからバイアスが無視できなくなると予想される。

これら 2 つの問題の間には最良のノイズレベルが存在するはずである。そこで 0.05 から 1 までの 7 つのレベルそれぞれにおいて 2 つのシードでネットワークを訓練し、20,000 個のテスト点を用いてクリーンなスコアに対するフィッシャーダイバージェンス Equation 17.8 を推定し、厳密な平滑化スコアのバイアスと並べて比較する。

```
sigmas, seeds = [0.05, 0.1, 0.15, 0.2, 0.3, 0.5, 1.0], (1, 2)
scores = {(sg, seed): fit_score(data, sg, seed)
          for seed in seeds for sg in sigmas}
x_test = mix.sample(jax.random.key(2), 20000)
target = mix.score(x_test)                    # the clean score
def score_error(score, x, target):
    return float(0.5 * ((score(x) - target) ** 2).sum(1).mean())
learned = {seed: [score_error(scores[sg, seed], x_test, target)
                  for sg in sigmas] for seed in seeds}
bias = [score_error(lambda x: mix.score(x, 1.0, sg), x_test, target)
        for sg in sigmas]
for seed in seeds:
    print(f'trained, seed {seed}:', [round(e, 3) for e in learned[seed]])
print('exact smoothed: ', [round(e, 4) for e in bias])
d2l.plot(sigmas, [*learned.values(), bias], 'sigma',
         'error against the clean score', xscale='log', yscale='log',
         legend=[f'network, seed {seed}' for seed in seeds]
         + ['exact score of p_sigma'])
```

```
trained, seed 1: [0.288, 0.206, 0.172, 0.15, 0.33, 1.058, 2.594]
trained, seed 2: [0.482, 0.272, 0.151, 0.224, 0.42, 1.105, 2.589]
exact smoothed:  [0.0004, 0.0059, 0.0272, 0.0759, 0.2794, 0.9972, 2.5655]
```



学習されたスコアの誤差は U 字型の曲線を描く。右側ではバイアスの曲線に沿い、その差は約 0.1 以内（$\sigma = 0.5$ 以上）に収まっている。ネットワークはぼかされた目標によく当てはまっているが、その目標自体が本来のスコアからずれているのである。左側ではバイアスが消えるにもかかわらず、目標の大部分がノイズとなるために誤差は再び上昇し、2 つ目のシードでは誤差が半分以上変化する箇所もある。最良のレベルは $\sigma = 0.1$ から 0.2 の間、すなわち成分の幅の 5 分の 1 から 5 分の 2 であり、先ほどのバイアスの目安である $s/3$ 付近に位置し、そのときの誤差は約 0.15〜0.25 となる。これが単一ノイズレベルにおける指針である。すなわち、σ をデータのスケールを単位として測り、ネットワークが当てはめられる限りにおいてなるべく小さく取ればよい。

17.1.3.2 モード重みに対する不感性

フィッシャーダイバージェンスにおける 0.2 という誤差は小さく聞こえるが、何と比べて小さいのだろうか。通し例を、その重みだけを均等にした分布と比較してみよう。この誤った重みをもつモデルは、1 点あたり 0.07 nat の尤度を失う（これが両者のカルバック・ライブラー・ダイバージェンスである）。ところがそのスコアが真のスコアと異なるのは Equation 17.1 の負担率を通じてのみであり、負担率が異なるのは p がほぼ 0 となる谷の部分に限られるため、p のもとでの平均であるフィッシャーダイバージェンスはこの違いをほとんど検知できない。

フィッシャーダイバージェンスの被積分関数はモンテカルロサンプルがほとんど落ちない谷に集中しているため、サンプリングではなく 721×721 グリッド上の数値積分（quadrature）によって両方のダイバージェンスを計算し、さらに両方の混合分布を同じノイズでぼかした場合についても同様の比較を繰り返す。

```
mix_eq = d2l.GaussianMixture(mix.means.tolist(), [1 / 3] * 3, mix.std)
g = jnp.linspace(-9, 9, 721)
grid = jnp.stack(jnp.meshgrid(g, g, indexing='ij'), -1).reshape(-1, 2)
area = (g[1] - g[0]) ** 2                    # of one grid cell
for sigma in (0.0, 0.5, 1.0):
    log_p = mix.log_prob(grid, 1.0, sigma)
    diff = mix.score(grid, 1.0, sigma) - mix_eq.score(grid, 1.0, sigma)
    fisher = area * (jnp.exp(log_p) * 0.5 * (diff ** 2).sum(1)).sum()
    kl = area * (jnp.exp(log_p)
                 * (log_p - mix_eq.log_prob(grid, 1.0, sigma))).sum()
    print(f'sigma = {sigma}: Fisher divergence {fisher:.1e}, KL {kl:.3f} nats')
```

```
sigma = 0.0: Fisher divergence 1.5e-05, KL 0.069 nats
sigma = 0.5: Fisher divergence 1.4e-03, KL 0.069 nats
sigma = 1.0: Fisher divergence 9.5e-03, KL 0.064 nats
```

クリーンな混合分布どうしでは、フィッシャーダイバージェンスは約 10^{-5} と、訓練されたネットワークの誤差（いずれも 0.1 を下回らなかった）より 4 桁も小さい一方、KL ダイバージェンスは 0.07 nat ある。したがって、クリーンなデータの近くで訓練されたスコアモデルは重みを学習できない。誤った重みをもつ場でも、正しい重みをもつ場と同程度によく当てはまってしまうからである（37.5.1 節、および 37.5 節の演習問題 2）。スコアはデータの近くでどれほど優れていても、各モードがどの頻度で生じるかについては何も知らないままになり得るのである。

ノイズを加えると重みが見えるようになる。σ が大きくなるにつれて各成分が谷へと広がり（Figure 17.2）、p_σ が確率質量をもつ領域で負担率に差が生じるため、KL ダイバージェンスがほとんど変化しない一方で、フィッシャーダイバージェンスは約 10^{-3}（$\sigma = 0.5$）および 10^{-2}（$\sigma = 1$）へと上昇する。それでも $\sigma = 1$ においてすら、このフィッシャーダイバージェンスは先ほどのスイープ実験におけるどの誤差よりも 10 倍以上小さいため、ノイズを加えただけで重みの学習が容易になるわけではない。ノイズがまず変えるのは、マルコフ連鎖が谷を越えられるようになるという点である。

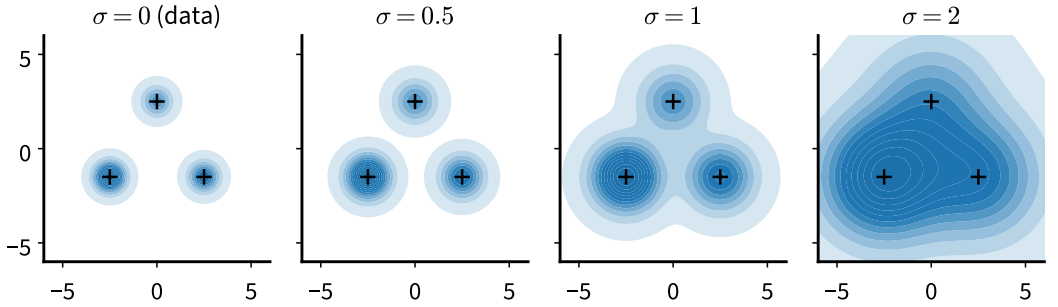


Figure 17.2: 標準偏差 σ を大きくしながらガウスノイズでぼかした通し例。十字は各成分の平均を表す。σ が大きくなるにつれてモード間の谷が埋まり、$\sigma = 2$ では 1 つの広い山だけが残る。

17.1.3.3 スコアによるサンプリング

サンプルを抽出するにはスコアがあれば十分であり、その最も単純な方法は 1 世紀以上前から知られている。1908 年、Paul Langevin は流体中で力に押されつつ周囲の分子からランダムな衝突を受ける粒子に対して、この種の方程式を書き下した。離散形式の**ランジュバン動力学**は任意の点から出発し、次の更新を繰り返す。

$$\mathbf{x} \leftarrow \mathbf{x} + \frac{h}{2}\,\mathbf{s}(\mathbf{x}) + \sqrt{h}\,\xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{17.9}$$

ここで h は小さなステップサイズである。ドリフト（drift）が点を対数密度の高い方向へ動かし、ノイズがモードへの一点崩壊を防ぐ。連続時間極限では、定常分布（stationary distribution）は p となり（$\mathbf{s} = \nabla \log p$ のとき。38.6.2 節）、十分に長く走らせた連鎖は初期点を忘れる。離散更新にはわずかなずれがあり、分散 v のガウス分布に対する定常分散は $v/(1 - h/(4v))$ となって、本節の成分では $h = 0.05$ において 5% の過剰が生じる。この更新式には密度そのものが一切現れないため、何らかの関数の勾配になっていない訓練済みの場であっても、任意のベクトル場で駆動できる。

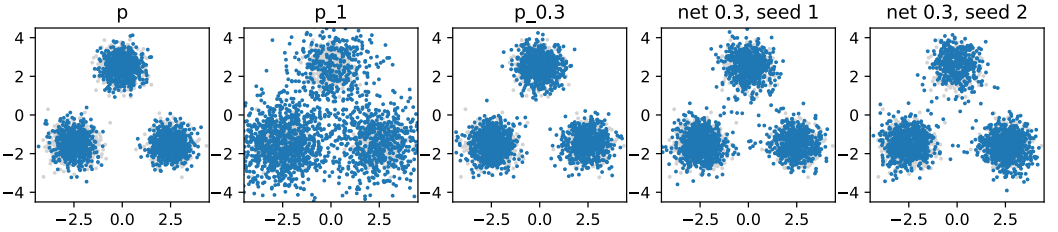
```
def langevin(score_fn, x, step, num_steps, key):
    """Langevin steps x <- x + step / 2 * score + sqrt(step) * noise."""
    def move(x, key):
        noise = jax.random.normal(key, x.shape)
        return x + step / 2 * score_fn(x) + step ** 0.5 * noise, None
    return jax.lax.scan(move, x, jax.random.split(key, num_steps))[0]
```

各モードに約 3 分の 1 ずつ割り振られるよう $\mathcal{N}(\mathbf{0}, 9\mathbf{I})$ から 2,000 本の連鎖を出発させ、$h = 0.05$ で 5 つの場を用いて 2,000 ステップ走らせる。用いる場は、p の厳密なスコア、p_σ の $\sigma = 1$ および $\sigma = 0.3$ における厳密なスコア（それぞれ `p_1`、`p_0.3` と表示）、そして先ほどのスイープ実験で $\sigma = 0.3$ において訓練した 2 つのネットワーク（これらを p のスコアであるかのように用いる）である。0.3 における厳密な場はネットワークが模倣するよう訓練された対象そのものであるため、ネットワークの振る舞いのうちどれがノイズレベルに由来し、どれがネットワーク自身の近似に由来するかを見分けられる。

p の厳密なスコアを用いれば、連鎖の最終的な重みは 0.5、0.3、0.2 になるだろうか。2,000 ステップが初期点を忘れるのに十分な長さであれば、そうなるはずである。$\sigma = 1$ における連鎖は p_σ からサンプリングするはずであり、その各成分の座標ごとの分散は $s^2 + \sigma^2 = 1.25$ と、p の 5 倍になる。完璧なサンプラに対する先ほどの計算で $2 \cdot 1.25$ を $2s^2$ の代わりに用いると、外れ点は $e^{-2.25/2.5} \approx 41\%$、広がりは 1 付近（データの約 2 倍にとどまる）と予測される。

```
fields = {'p': mix.score,                    # exact scores of p, p_1, p_0.3
          'p_1': lambda x: mix.score(x, 1, 1),
          'p_0.3': lambda x: mix.score(x, 1, 0.3),
          'net 0.3, seed 1': scores[0.3, 1],   # trained at sigma = 0.3
          'net 0.3, seed 2': scores[0.3, 2]}
key, *keys = jax.random.split(jax.random.key(2), len(fields) + 1)
x_init = 3 * jax.random.normal(key, (2000, 2))
samples = [d2l.langevin(f, x_init, 0.05, 2000, k)
           for f, k in zip(fields.values(), keys)]
print(f'{"start":>15}:', mix.summary(x_init))
print(f'{"data":>15}:', mix.summary(data))
for name, x in zip(fields, samples):
    print(f'{name:>15}:', mix.summary(x))
d2l.scatter_panels(samples, list(fields), data=data, size=2.3);
```

```
          start: {'weights': [0.324, 0.313, 0.363], 'stray': 0.763, 'spread': 1.131}
           data: {'weights': [0.495, 0.293, 0.212], 'stray': 0.015, 'spread': 0.491}
              p: {'weights': [0.335, 0.314, 0.352], 'stray': 0.02, 'spread': 0.492}
            p_1: {'weights': [0.479, 0.31, 0.212], 'stray': 0.4115, 'spread': 0.958}
          p_0.3: {'weights': [0.353, 0.312, 0.336], 'stray': 0.043, 'spread': 0.614}
net 0.3, seed 1: {'weights': [0.407, 0.297, 0.296], 'stray': 0.0595, 'spread': 0.601}
net 0.3, seed 2: {'weights': [0.393, 0.395, 0.213], 'stray': 0.0645, 'spread': 0.623}
```



p の厳密なスコアはすべての連鎖を正しい局所的な広がりをもつモードへと導くが、モード重みは初期値（それぞれ $\approx 1/3$）のまま動かない。モード間の谷は $-\log p$ においてモードより 10〜12 nat 高いため、期待される Kramers の脱出時間は 10^6 ステップのオーダー（$h = 0.05$ のとき。演習問題 6）となり、2,000 ステップではモード間を混合できないのである。逆に $\sigma = 1$ ではノイズが谷を埋めるため、連鎖が混合して真の重み $(0.5, 0.3, 0.2)$ を回復できるが、サンプリングされるのは裾の広い分布 p_1（成分の分散が $s^2 + \sigma^2 = 1.25$ で、$\approx 41\%$ の外れ点と 1 付近の広がりを生じる）になってしまう。

$\sigma = 0.3$ で訓練されたネットワークは $p_{0.3}$ のスコア（成分の分散 0.34、外れ点 $\approx 4\%$、広がり ≈ 0.6）を近似する。$p_{0.3}$ の厳密なスコアは 2,000 ステップでは 7〜9 nat の障壁を越えてほとんど混合しない（最大重みのモードの変化は高々 0.03）が、学習されたネットワークはモード間でより多くの質量を移動させる。データが疎な谷における近似誤差が、人工的なドリフトの通り道を作り出すためである。

このように、単一のノイズレベルは根本的なトレードオフに直面する。小さな σ は鋭いモードを保つが連鎖を閉じ込め、フィッシャーダイバージェンスもモード重みに対して不感なままになる。一方、大きな σ はモード間を速やかに混合させるが、強く平滑化された密度からしかサンプリングできない。17.2 節では、粗いレベルから細かいレベルまですべてのノイズレベルにわたってアニーリングすることでこの問題を解決する。

17.1.4 画像のノイズ除去

これまでの議論は平面の外でも成り立つだろうか。ノイズ除去損失にはデータの次元に依存する箇所が一切ないため、画像に対してもそのまま適用できる。失われるのは真の基準値である。画像の事後平均は計算できないため、計算可能な最良のノイズ除去器である線形ノイズ除去器とネットワークを比較する。ここではピクセル値を $[-1, 1]$ にスケーリングした Fashion-MNIST（3.2 節）を 2 つの小さな補助関数で読み込み・表示し、標準偏差 $\sigma = 0.5$ のノイズを用いる。ネットワークは 32 チャネル・SiLU 活性化関数をもつ 4 層の 3×3 畳み込みで構成し、スコアネットワークと同じ訓練ループを用いて、毎回 128 枚の画像のミニバッチに新たなノイズを加えながら 600 ステップ訓練する。得られた結果に再現性の裏付けを添えるため、2 つのシードから 2 回訓練する。このネットワークはノイズ ϵ を予測するため、Equation 17.6 によりその出力を $-\sigma$ で割ったものは、密度を一度も書き下すことなく得られた 784 次元のノイズ付き画像のスコアとなり、$\tilde{\mathbf{x}} - \sigma\hat{\epsilon}$ がノイズ除去後の画像となる。

```
def load_fashion_mnist_pm1(binary=False):
    """Fashion-MNIST (X, y, X_test, y_test), images in [-1, 1] or binary."""
    data = d2l.FashionMNIST()
    def prep(X):  # (n, 28, 28, 1) floats or (n, 28, 28) binary pixels
        return (jnp.asarray(X > 127, jnp.int32) if binary
                else jnp.asarray(X[..., None], jnp.float32) / 127.5 - 1)
    return (prep(data.train[0]), jnp.asarray(data.train[1], jnp.int32),
            prep(data.val[0]), jnp.asarray(data.val[1], jnp.int32))

def show_gray(imgs, num_rows, num_cols, titles=None, scale=1.0):
    """Show images (n, h, w, 1) in [-1, 1], or binary (n, h, w), in gray."""
    imgs = jnp.asarray(imgs)
    imgs = jnp.clip((imgs[..., 0] + 1) / 2, 0, 1) \
        if jnp.issubdtype(imgs.dtype, jnp.floating) else imgs.astype(float)
    return d2l.show_images(jnp.repeat(imgs[..., None], 3, -1), num_rows,
                           num_cols, titles, scale)
```

これらの補助関数を用意したうえで、$[-1, 1]$ にスケーリングした Fashion-MNIST 画像を読み込

み、ノイズ予測（noise prediction）の目的関数 `noise_loss` を $\sigma = 0.5$ で定義して、4 層の畳み込みノイズ除去器を 2 つのシードで 600 ステップずつ `d2l.train_regression` により訓練する。

```
X, _, X_test, _ = d2l.load_fashion_mnist_pm1()
sigma = 0.5
def noise_loss(net, x, key, sigma):
    """Squared error of the predicted noise; the score is -net / sigma."""
    eps = jax.random.normal(key, x.shape)
    return ((net(x + sigma * eps) - eps) ** 2).mean()

batch_fn = lambda key: X[jax.random.randint(key, (128,), 0, len(X))]
nets = {}
for seed in (0, 1):
    rngs = nnx.Rngs(seed)
    net = nnx.Sequential(nnx.Conv(1, 32, (3, 3), rngs=rngs), nnx.silu,
                         nnx.Conv(32, 32, (3, 3), rngs=rngs), nnx.silu,
                         nnx.Conv(32, 32, (3, 3), rngs=rngs), nnx.silu,
                         nnx.Conv(32, 1, (3, 3), rngs=rngs))
    nets[seed], _, losses = d2l.train_regression(
        net, lambda net, x, key: noise_loss(net, x, key, sigma), batch_fn,
        600, jax.random.key(seed))
    print(f'seed {seed}: last training loss {losses[-50:].mean():.3f}')
```

```
seed 0: last training loss 0.125
seed 1: last training loss 0.123
```

ピクセルあたりの予測ノイズの二乗誤差である損失は、どちらのシードでも 0 よりはるかに大きい 0.12 付近で頭打ちになる。これに $\sigma^2 = 0.25$ を掛けたものがノイズ除去画像の二乗誤差であり、ピクセルあたり約 0.03 となる。以下のテストでは、ネットワークが訓練時に見ていない画像でこれを確認する。

次に比較基準となる最良の**線形**ノイズ除去器を導入する。仮に画像が平均 μ、共分散 Σ のガウス分布に従うなら、Tweedie の公式は先ほどのガウス分布における縮小を Σ の各固有ベクトルに沿ってそれぞれの係数 $\lambda/(\lambda + \sigma^2)$ で適用する形になる。

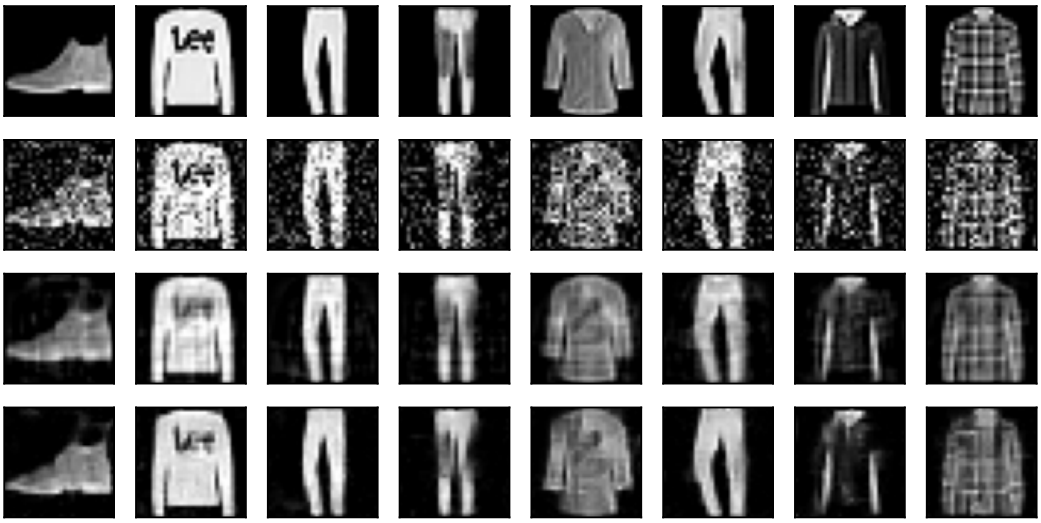
$$D_{\text{lin}}(\tilde{\mathbf{x}}) = \mu + \Sigma(\Sigma + \sigma^2\mathbf{I})^{-1}(\tilde{\mathbf{x}} - \mu). \tag{17.10}$$

この**ウィーナーフィルタ**（Wiener filter）は、画像の変動がノイズより大きい方向を残し、それ以外の方向を抑制するものであり、同じ平均と共分散をもつあらゆる分布に対して最良のアフィンノイズ除去器となる（演習問題 5）。60,000 枚の訓練画像から μ と Σ を推定し、1,000 枚のテスト画像上でノイズ付き入力、ウィーナーフィルタ、および 2 つのネットワークのピクセルあたり二乗誤差を比較する。各平均値の横には、画像間にわたる標準誤差もあわせて出力する。図の各行は上から順に、クリーンな画像、ノイズを加えた画像、フィルタの出力、そして 1 つ目のネットワークの出力を示している。

```
Xf = X.reshape(len(X), -1)
mu = Xf.mean(0)
Sigma = (Xf - mu).T @ (Xf - mu) / len(Xf)      # (784, 784) pixel covariance
A = jnp.linalg.solve(Sigma + sigma ** 2 * jnp.eye(784),
                     Sigma)                    # (Sigma + sigma^2 I)^-1 Sigma
x = X_test[:1000]
```

```
xt = x + sigma * jax.random.normal(jax.random.key(1), x.shape)
x_lin = (mu + (xt.reshape(1000, -1) - mu) @ A).reshape(x.shape)
x_net = {seed: xt - sigma * net(xt)            # Tweedie: xt + sigma^2 * score
         for seed, net in nets.items()}
outputs = {'noisy': xt, 'Wiener': x_lin,
           **{f'network, seed {seed}': xh for seed, xh in x_net.items()}}
for name, xh in outputs.items():               # error of each of 1,000 images
    err = ((xh - x) ** 2).reshape(len(x), -1).mean(1)
    print(f'{name:>16}: per-pixel squared error {err.mean():.4f} '
          f'(standard error {err.std() / math.sqrt(len(err)):.4f})')
d2l.show_gray(jnp.concatenate([x[:8], xt[:8], x_lin[:8], x_net[0][:8]]), 4, 8);
```

```
         noisy: per-pixel squared error 0.2500 (standard error 0.0004)
        Wiener: per-pixel squared error 0.0444 (standard error 0.0005)
network, seed 0: per-pixel squared error 0.0308 (standard error 0.0004)
network, seed 1: per-pixel squared error 0.0308 (standard error 0.0004)
```



ノイズ付き入力のピクセルあたり二乗誤差は $\sigma^2 = 0.25$ であり、最適な線形（ウィーナー）フィルタは ≈ 0.045 を達成し、4 層の畳み込みネットワークはいずれも ≈ 0.031 に達する（誤差が 30% 低く、20 標準誤差以上離れている）。ウィーナーフィルタがすべての画像を単一の大域的な線形部分空間（subspace）へと縮小させ、ぼやけた輪郭や背景のざらつきを残すのに対し、畳み込みネットワークはノイズ付きの点を衣類画像の非線形多様体（nonlinear manifold）へと射影する（演習問題 3 参照）。ただし、事後平均は $\tilde{\mathbf{x}}$ と整合するすべてのクリーンな画像にわたって平均を取るため、ノイズで失われた微細な高周波テクスチャは依然として滑らかな平均に置き換えられてしまう。ノイズ除去器を鋭さを保つ生成器へと変えるには、複数のノイズレベルにわたる積分が必要になる（17.2 節および 17.3 節）。

17.1.5 まとめ

二乗ノイズ除去誤差 $\mathbb{E}\|D(\tilde{\mathbf{x}}) - \mathbf{x}_0\|^2$（ただし $\tilde{\mathbf{x}} = \mathbf{x}_0 + \sigma\epsilon$）を最小化すると、事後平均 $\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}]$ が復元される。Vincent の定理と Tweedie の公式により $\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ が成り立つため、クリーンな点 $\hat{\mathbf{x}}_0$、ノイズ $\hat{\epsilon}$、およびスコア $\hat{\mathbf{s}} = -\hat{\epsilon}/\sigma$ の予測は、いずれも単一の条件付き期待値のアフィンな再パラメータ化にほかならない。スコアを学習すれば、エネルギー

ベースモデルにおける計算困難な正規化定数 Z_θ も、明示的スコアマッチング（explicit score matching）におけるヤコビ行列のトレース $\nabla \cdot \mathbf{s}$ も回避できる。しかし、単一のノイズレベル σ には厳しいトレードオフが伴う。小さな σ では目標の分散が大きくなり、低次元データ多様体の近くで勾配が急峻化し、フィッシャーダイバージェンスがモード重みに対して不感になり、さらにランジュバン連鎖が混合しなくなる。一方、大きな σ では速やかに混合してモード重みを回復できるものの、強く平滑化された密度 p_σ からしかサンプリングできない。

17.1.6 演習問題

1. **正規化なしの学習**。1 次元モデル $\tilde{p}(x) = \exp(-E(x))$（ただし $E(x) = (x-\mu)^2/(2\tau^2)$）を、少なくとも 2 つの異なる値をとるデータ $x_1, \dots, x_n$ に当てはめることを考える。
   1. 正規化を行わずに $\sum_i \log \tilde{p}(x_i)$ を μ と τ について最大化せよ。目的関数がその上限に近づくとき、データから離れた場所で $\tilde{p}$ はどのようになるか。
   2. 正規化したモデル $\tilde{p}/Z$ で同じことを行え。目的関数のどの項が最初の設問での破綻を防ぎ、そのときの最大点は何になるか。
   3. ネットワークによるエネルギーは通常、出力バイアス c をもつ線形層で終わる。正規化された密度 Equation 17.7 が c に依存しないことを示せ。$-\sum_i E_\theta(\mathbf{x}_i)$ のみに対して勾配上昇法を適用すると、c はどうなるか。

2. [code] **正規化定数の推定**。エネルギーの正規化定数 $Z = \int e^{-E(\mathbf{x})}\, d\mathbf{x}$ を計算する直接的な方法は重要度サンプリング（36.8.2 節）である。すなわち、$\mathbf{x}_1, \dots, \mathbf{x}_n$ を提案分布 q から抽出し、重み $w_i = e^{-E(\mathbf{x}_i)}/q(\mathbf{x}_i)$（その期待値は Z である）を平均する。$E(\mathbf{x}) = \|\mathbf{x}-\mathbf{m}\|^2/2$ とおくと、目標分布は正規化を除いて $\mathcal{N}(\mathbf{m}, \mathbf{I})$ となり、提案分布を $q = \mathcal{N}(\mathbf{0}, \mathbf{I})$ とする。
   1. $w/Z = \exp(\mathbf{m}^\top \mathbf{x} - \|\mathbf{m}\|^2/2)$ を導出し、ガウス分布の積率母関数を用いて $\mathbb{E}_q[w/Z] = 1$ および $\mathbb{E}_q[(w/Z)^2] = \exp(\|\mathbf{m}\|^2)$ を示せ。さらに、n 個のサンプルによる推定量の相対標準誤差が $\sqrt{(e^{\|\mathbf{m}\|^2} - 1)/n}$ であることを導け。相対誤差を 0.1 にするには（$\|\mathbf{m}\|^2 = 20$ のとき）、何個のサンプルが必要か。
   2. $\mathbf{m}$ を d 次元の全要素が 1 のベクトルとし、提案分布が各座標で標準偏差 1 つ分だけずれている状況を考える。$n = 10^5$ として Z を $d \in \{2, 4, 8, 16, 32\}$ について推定し、真の $Z = (2\pi)^{d/2}$ に対する推定値の比と**有効標本サイズ** $(\sum_i w_i)^2 / \sum_i w_i^2$ を報告して、両者を最初の設問の予測値と比較せよ（有効標本サイズは $n\, e^{-\|\mathbf{m}\|^2}$ に近くなるはずである）。d が大きくなると有効標本サイズが予測値を上回るのはなぜか。また、このことは診断指標としての有効標本サイズについて何を意味するか。
   3. 1 枚の画像は数十万個の座標をもつ。得られた結果から、画像のエネルギーベースモデルの正規化定数を推定すること、および最尤法でそれを訓練することについて何が言えるか。

3. **直線上のデータ**。データが $\mathbb{R}^2$ の第 1 座標軸上にあり、$\mathbf{x}_0 = (u, 0)$（ただし $u \sim q$ であり、q は $\mathbb{R}$ 上の滑らかな正の密度）であるとする。このとき p は $\mathbb{R}^2$ 上の密度をもたない。
   1. $p_\sigma(\tilde{\mathbf{x}}) = q_\sigma(\tilde{x}_1)\, \mathcal{N}(\tilde{x}_2; 0, \sigma^2)$（ただし q_σ は q を $\mathcal{N}(0, \sigma^2)$ でぼかしたもの）となることを示し、スコア $\nabla \log p_\sigma$ を書き下せ。
   2. スコアの第 2 成分は二乗平均が $1/\sigma^2$（p_σ のもとで）であり、傾きが $-1/\sigma^2$（$\tilde{x}_2$ に関する）であることを示せ。$\sigma \to 0$ のときこの両者はどうなるか。また、小さなノイズレベルでスコアを表現しなければならないネットワークにとって、これは何を意味するか。
   3. Tweedie の公式 Equation 17.5 を適用せよ。最適なノイズ除去器は $\tilde{\mathbf{x}}$ の各座標に何を行い、どのような幾何学的操作を実行するか。

4. 代わりにデータが k 次元の線形部分空間（$\mathbb{R}^d$ 内の）上にあるとする。部分空間に直交するスコア成分の二乗平均を計算せよ。

4. **2 点データと損失の下限。** 1 次元において $p = \tfrac{1}{2}\delta_{-1} + \tfrac{1}{2}\delta_{+1}$ とし、これにレベル σ のノイズを加える。

   1. ベイズの定理によって事後平均 $\mathbb{E}[x_0 \mid \tilde{x}]$ を計算し、Tweedie の公式 Equation 17.5 が要請するとおり $\tilde{x} + \sigma^2 \frac{d}{d\tilde{x}} \log p_\sigma(\tilde{x})$ に等しいことを示せ。
   2. $\tilde{x} = 0$ に対して最適なノイズ除去器は何を返すか。それが取り得るクリーンな値ではないにもかかわらず、二乗誤差のもとで最良の答えとなるのはなぜか。
   3. Equation 17.3 の最小値を $\frac{1}{2\sigma^4}\,\mathbb{E}\big[\operatorname{tr}\operatorname{Cov}(\mathbf{x}_0 \mid \tilde{\mathbf{x}})\big]$ と表し、1 点に集中していないあらゆるデータ分布に対してこれが正であることを示せ。
   4. 2 点データに対してこの最小値を $\tilde{x}$ に関する 1 次元積分として書き下し、$\sigma \in \{0.25, 0.5, 1, 2\}$ について数値積分で評価せよ。各値に $2\sigma^2$ を掛けてノイズ予測損失の単位で表せ。スケールし直したこの下限が $\sigma \to 0$ で 0 に収束する理由と、$\sigma \to \infty$ でも 0 に収束する理由を説明せよ。

5. [code] **最良の線形ノイズ除去器。**

   1. ウィーナーフィルタ Equation 17.10 が $\mathbb{E}\|D(\tilde{\mathbf{x}}) - \mathbf{x}_0\|^2$ をすべてのアフィン写像 D の中で最小化すること（平均 μ と共分散 Σ をもつあらゆるデータ分布に対して）、およびそのピクセルあたり二乗誤差が $\frac{1}{d}\sum_i \sigma^2\lambda_i/(\lambda_i + \sigma^2)$（ただし $\lambda_1, \dots, \lambda_d$ は Σ の固有値）であることを示せ。
   2. $\sigma = 0.5$ におけるこの予測値を `Sigma` から計算し、出力された `x_lin` の誤差と比較せよ。
   3. `Sigma` の固有値のうち σ^2 を上回るものの個数を $\sigma \in \{0.25, 0.5, 1\}$ について数え、対応する固有ベクトルをいくつか画像としてプロットせよ。各ノイズレベルにおいてフィルタは何を残しているか。

6. [code] **モードからの脱出。●** 通し例の下側 2 つのモード間にある谷を、1 次元における $\mathcal{N}(-2.5, 0.25)$ と $\mathcal{N}(2.5, 0.25)$ の等重み混合分布でモデル化する。ポテンシャルを $U = -\log p$ とすると、井戸の底は $a = \pm 2.5$、障壁は $b = 0$ にある。障壁の高さ $\Delta U = U(b) - U(a)$ が 1 に比べて大きいとき、Kramers の公式 (Kramers 1940) は、拡散過程 $dX = -\tfrac{1}{2}U'(X)\,dt + dW$（Equation 17.9 の連続時間極限）が一方の井戸から他方の井戸へ移るまでの期待時間を次式で与える。

$$\tau \approx \frac{4\pi}{\sqrt{U''(a)\,|U''(b)|}}\, e^{\Delta U}.$$

   1. ΔU、$U''(a)$、$U''(b)$ を計算し、τ を $h = 0.05$ におけるランジュバンステップ数に換算せよ。
   2. 通し例のモード 3 はモード 1（およびモード 2）から約 4.7 の距離にあり、下側 2 つのモード間の距離 5 よりも近い。2 つの平均を結ぶ線分に沿って通し例の $U = -\log p$ を評価し、障壁の高さを読み取って、同様に横断時間を推定せよ。2,000 本の連鎖のうち、モード 1 からモード 3 へ 2,000 ステップ以内に移動するのは何本になると予測されるか。
   3. p_σ に対して $\sigma = 0.3$ と $\sigma = 1$ で最初の設問を繰り返し（ΔU が小さい場合は桁数の見積もりとして扱え）、その結果を用いて $\sigma = 1$ の連鎖の重みを説明せよ。2,000 本の連鎖のうち 2,000 ステップ以内に横断するのは $\sigma = 0.3$ では何本になるはずか。また、出力された `p_0.3` の行はこれと一致しているか。$\sigma = 0.3$ で訓練されたネットワークはその少なくとも 2 倍の質量を移動させている。このことは、モード間における学習された場について何を物語っているか。

17.2 ガウス経路：拡散とフローマッチング

17.1 節で見たように、単一のノイズレベル σ では混合と鮮明な生成を両立できない。σ が小さいとランジュバン連鎖は初期モードに閉じ込められ、逆に σ が大きいとモード間を混合するものの強くぼやけた密度からサンプリングしてしまうからである。解決策は、$t=0$ におけるデータ分布と $t=1$ における単位ガウス分布を連続的な**ガウス経路**（Gaussian path）$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$ で結び、高ノイズ側からノイズゼロへとたどっていくことである。

ノイズ除去拡散確率モデル（denoising diffusion probabilistic model、DDPM）(Ho et al. 2020)、スコアベースモデル（score-based model）(Song and Ermon 2019; Y. Song, Sohl-Dickstein, et al. 2021)、フローマッチング (Lipman et al. 2023; Liu et al. 2023; Albergo et al. 2023) はいずれも、最小二乗回帰（least-squares regression）によって $t \in [0,1]$ の全レベルにわたる単一の時間条件付きネットワークを訓練する。一般的な訓練目標（クリーンな点 $\mathbf{x}_0$、ノイズ ϵ、スコア $\nabla \log p_t(\mathbf{x}_t)$、速度 $\dot\alpha_t \mathbf{x}_0 + \dot\sigma_t \epsilon$、角速度 $\mathbf{v} = \alpha_t \epsilon - \sigma_t \mathbf{x}_0$）はすべて、$\mathbf{x}_t$ と事後平均（posterior mean）$\hat{\mathbf{x}}_0(\mathbf{x}_t) = \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]$ のアフィン関数であるため、いずれか 1 つの目標で訓練したネットワークから残りの目標もすべて定まる。本節では、標準的なガウス経路とそれらが共有する対数信号対雑音比（log signal-to-noise ratio、log-SNR）の時計を定義し、厳密なスコア上でアニーリング付きランジュバンサンプリングを検証する。さらに、条件付きフローマッチングと目標の変換辞書を導出し、3 モード混合分布における厳密な損失の下限と訓練済みネットワークを比較して検証する。

17.2.1 データからノイズへの経路

すべてのノイズレベルをたどるには、各レベルに時間のラベルを付けて一列に並べる必要がある。

17.2.1.1 信号、ノイズ、対数信号対雑音比

前節で扱ったノイズ付きの点 $\tilde{\mathbf{x}} = \mathbf{x}_0 + \sigma\epsilon$ は、σ を時間とともに増大させれば 1 つの経路になる。この経路はデータの大きさを保ったまま、データが無視できるほどになるまでノイズを大きくしていく。その分散が際限なく増大するため、**分散爆発**（variance-exploding、VE）経路と呼ばれる。この経路上のネットワークにはノイズとともに大きくなる入力が入るため、1 組の重みで大きさ 1 の入力も、CIFAR-10 画像に対する Song と Ermon のモデルではその 50 倍の大きさの入力も処理しなければならない。そのため大半のモデルは、ノイズが増大するにつれてデータを縮小し、すべての入力を単位サイズ付近に保つ。**ガウス経路**とは、α_t と σ_t という $t \in [0,1]$ 上の関数の組であり、ノイズ付きの点を次の式で定義する。

$$\mathbf{x}_t = \alpha_t\, \mathbf{x}_0 + \sigma_t\, \epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{17.11}$$

この経路は $(\alpha_0, \sigma_0) = (1, 0)$ から出発する。VE 経路とは異なり、よく使われる 2 つの経路は純粋なノイズ $(\alpha_1, \sigma_1) = (0, 1)$、あるいはそれにほぼ等しい状態で終わる。

DDPM (Ho et al. 2020) の**分散保持**（variance-preserving、VP）経路は $\alpha_t^2 + \sigma_t^2 = 1$ を保つため、単位分散のデータは単位分散のまま保たれる。レート $\beta(t) = \beta_{\min} + (\beta_{\max} - \beta_{\min})\, t$ は $\beta_{\min} = 0.1$ から $\beta_{\max} = 20$ まで線形に増加し、データを忘却する速さを定める。微小時間 dt ごとに信号はその分散の割合 $\beta(t)\, dt$ を失うため、残存する割合は積分したレートに負号をつけたものの指数関数となる。

$$\bar\alpha_t := \alpha_t^2 = \exp\Big(- \int_0^t \beta(r)\, dr \Big),$$
$$\log \bar\alpha_t = -(0.1\, t + 9.95\, t^2).$$

0.1 と 20 という数値は恣意的なものではない。これらは DDPM の最初と最後のステップ分散である 10^{-4} と 0.02 にステップ数 1,000 を掛けたものであり、ステップがどのようにレートになるかは次項で示す。経路の終端では $\bar{\alpha}_1 = e^{-10.05} \approx 4 \cdot 10^{-5}$ となり、実用上は純粋なノイズとみなして差し支えない。時間に沿って追跡すると、VP 経路上の 1 点は原点へと引き戻されるランダムウォークであるオルンシュタイン・ウーレンベック過程（38.4.3 節）を、進みが加速する時計のもとで描く。引き戻す力とノイズの分散はいずれも $\beta(t)$ に比例し、終端では始端の 200 倍になる。

フローマッチングとレクティファイドフロー (Lipman et al. 2023; Liu et al. 2023; Albergo et al. 2023) の**線形**経路は、各クリーンな点からそのノイズへと直線を引く。

$$\alpha_t = 1 - t, \qquad \sigma_t = t. \tag{17.12}$$

これは厳密にノイズで終わる最も単純な経路であり、各データ例はその直線上を一定の速さで進む。この単純さゆえに、本章のデフォルトモデルの経路として、また 2024 年以降の多くの画像生成器の経路として採用されている (Esser et al. 2024)。

2 つの経路が異なる時刻においてノイズ除去器（denoiser）に同じ情報を与えるのはいつだろうか。ノイズ除去器が $\mathbf{x}_t$ から推論できる内容は、**対数信号対雑音比**（log-SNR）$\lambda_t = \log(\alpha_t^2/\sigma_t^2)$ として測られる信号対雑音比のみに依存する。その理由を見るために、ノイズ付きの点に $c = 1/(\alpha_t + \sigma_t)$ を掛けてみよう。$c\alpha_t \mathbf{x}_0 + c\sigma_t \epsilon$ の係数の和は 1 になるため、結果は同じ信号対雑音比を持つ時刻 $\sigma_t/(\alpha_t + \sigma_t)$ における線形経路上の点となる。既知の数を掛けても情報は失われないため、再スケーリングされた点は $\mathbf{x}_t$ と同じ情報を持つ。

したがって、λ が等しい時刻で読み取れば 2 つの経路はクリーンな点について同じ情報を保持しており、対数信号対雑音比はすべての経路が共有する時計となる。この時計はデータにおける $+\infty$ から純粋なノイズにおける $-\infty$ まで進む。線形経路はその全域を網羅し、VP 経路は $\lambda = -10$ 付近で止まる。ゆえに経路を選ぶことは、この時計がどれだけ速く進むかを選ぶことにほかならず、それによって訓練とサンプリングがどの領域に時間を費やすかが決まる。

`GaussianPath` は α_t と σ_t を保持し、対数信号対雑音比を計算して $\mathbf{x}_t$ をサンプリングする。そのコンストラクタは VP 経路、線形経路、および以下で登場する 3 つ目の経路であるコサイン経路を構築する。

```
class GaussianPath:
    """x_t = alpha(t) x0 + sigma(t) eps, data at t = 0 and noise at t = 1."""
    def __init__(self, alpha, sigma):
        self.alpha, self.sigma = alpha, sigma

    @staticmethod
    def vp(beta_min=0.1, beta_max=20.0):
        """Variance preserving, Ho et al.'s linear beta in continuous time."""
        log_abar = lambda t: -(beta_min * t + (beta_max - beta_min) * t**2 / 2)
        return GaussianPath(lambda t: jnp.exp(log_abar(t) / 2),
                            lambda t: jnp.sqrt(-jnp.expm1(log_abar(t))))

    @staticmethod
    def linear():
        """The straight line of flow matching and rectified flow."""
        return GaussianPath(lambda t: 1 - t, lambda t: t)

    @staticmethod
```

```
    def cosine(s=0.008):
        """Variance preserving, Nichol and Dhariwal's cosine schedule."""
        c = lambda t: jnp.cos(math.pi / 2 * (t + s) / (1 + s))
        g = lambda t: c(t) / c(jnp.zeros_like(t))     # exactly 1 at t = 0
        # and exactly 0 at t = 1 (cos(pi/2) is not), keeping the slope
        f = lambda t: jnp.where(t < 1, g(t),
                                g(t) - jax.lax.stop_gradient(g(t)))
        return GaussianPath(f, lambda t: jnp.sqrt(1 - f(t) ** 2))

    def log_snr(self, t):
        """lambda_t = log(alpha_t^2 / sigma_t^2), falling from +inf at 0."""
        return 2 * (jnp.log(self.alpha(t)) - jnp.log(self.sigma(t)))

    def sample(self, x0, t, eps):
        """x_t for per-example times t of shape (n,)."""
        t = t.reshape(-1, *[1] * (x0.ndim - 1))
        return self.alpha(t) * x0 + self.sigma(t) * eps
```

$t = 0.5$ において線形経路は信号とノイズを等しい割合で含み、$\lambda = 0$ となる。では VP 経路はどこにあるだろうか。

```
vp, linear = GaussianPath.vp(), GaussianPath.linear()
t = jnp.array([0.1, 0.5, 0.9])
r2 = lambda v: [round(float(u), 2) for u in v]
for name, path in (('vp', vp), ('linear', linear)):
    print(f'{name:>6}: alpha {r2(path.alpha(t))} sigma {r2(path.sigma(t))} '
          f'log-SNR {r2(path.log_snr(t))}')
```

```
    vp: alpha [0.95, 0.28, 0.02] sigma [0.32, 0.96, 1.0] log-SNR [2.16, -2.46, -8.15]
linear: alpha [0.9, 0.5, 0.1] sigma [0.1, 0.5, 0.9] log-SNR [4.39, 0.0, -4.39]
```

時間の中間点において VP 経路はすでに大半がノイズになっており、$\alpha = 0.28$、$\lambda \approx -2.5$ である。また $t = 0.9$ で残っている信号は、線形経路の 10% に対して 2% にすぎない。VP 経路は $\lambda = 0$ を $t = 0.26$ 付近で横切るため、時間の 4 分の 3 をノイズが支配的な領域で過ごす（Figure 17.3）。

小さな画像ではこれは無駄が多い。Nichol and Dhariwal (2021) は、一辺が 32 ピクセルや 64 ピクセルの画像では $t = 1$ よりずっと前に画像が消えてしまうため、DDPM のスケジュールの終盤がほとんど寄与しないことを観察した。同論文の**コサイン経路**は、データ付近での小さなオフセットを除けば $\alpha_t = \cos \frac{\pi t}{2}$ および $\sigma_t = \sin \frac{\pi t}{2}$ と置く。これもやはり VP 経路の一種だが、$\bar{\alpha}_t$ が中間領域でほぼ線形に減少し両端では緩やかに変化するため、画像がすでに消えた領域に費やす時間が少なくなる。

> **!** 時計の規約
>
> 本書全体を通してデータは $t = 0$、ノイズは $t = 1$ に置かれ、すべてのサンプラは $t = 1$ から 0 へと逆向きに走る。フローマッチングの論文では時間の向きが逆になっているため（Figure 38.26）、$1 - t$ をそれらの論文の t に代入し、ノイズ $\mathbf{x}_0$ を本書の ϵ、データ $\mathbf{x}_1$ を本書の $\mathbf{x}_0$ と読み替え、すべての速度の符号を反転させる必要がある。

拡散モデルに関する論文では、同じ量がいくつもの異なる記法で書かれる。Table 17.1 は参照用であり、本章でこれ以降用いる記号と論文での表記をまとめてある。初読時には読み飛ばし、論

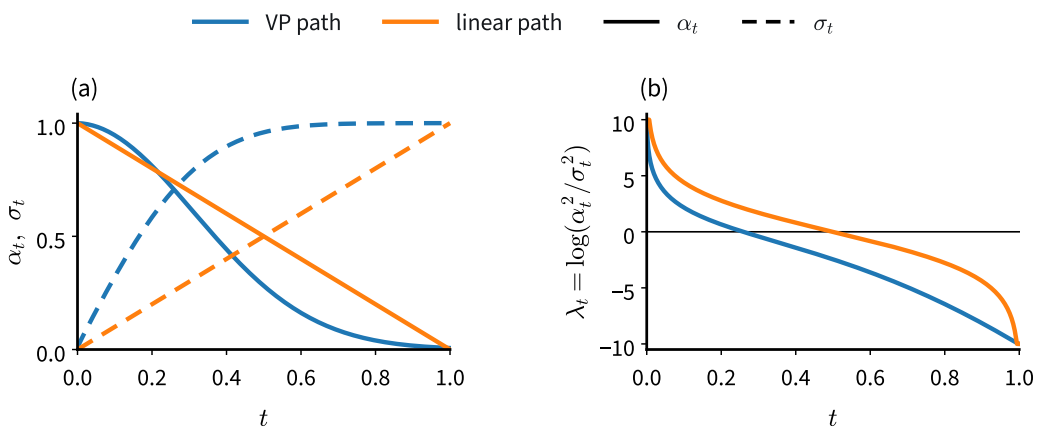


Figure 17.3: 1 つの時計の上の 2 つのガウス経路。(a) 分散保持経路の信号係数 α_t（実線）とノイズ係数 σ_t（破線）（ただし $\beta(t)$ は 0.1 から 20 まで増加）、および線形経路の両係数。(b) 両経路の対数信号対雑音比 λ_t（$[-10, 10]$ にクリップ）。分散保持経路の方がはるかに早くノイズ優勢の領域に達する。

文の記法に戸惑ったときに参照するとよい。

Table 17.1: 拡散モデルの章における記法。後の節では必要に応じていくつかの記号を追加する。

記号	意味	論文での表記
$t \in [0, 1]$	経路に沿った時刻（0 がデータ、1 がノイズ）	フローマッチングでは $t = 1$ がデータ、DDPM では整数ステップ
$\mathbf{x}_0$、ϵ、$\mathbf{x}_t$	クリーンな点、標準ガウスノイズ、ノイズ付きの点	フローマッチングではデータが $\mathbf{x}_1$、ノイズが $\mathbf{x}_0$
α_t、σ_t	経路の信号係数とノイズ係数	DDPM では $\sqrt{\bar{\alpha}_t}$、$\sqrt{1-\bar{\alpha}_t}$（DDPM の α_t は $1-\beta_t$）
$\bar{\alpha}_t = \alpha_t^2$、$\beta(t)$	VP 経路の二乗信号係数、そのレート	
k、T、β_k	DDPM 連鎖のステップ、ステップ数、ステップ k で加える分散	t（k の代わり）
λ_t	対数信号対雑音比 $\log(\alpha_t^2/\sigma_t^2)$	VDM では $-\gamma(t)$
σ、$\tilde{\sigma}_i$	VE 経路のノイズレベル、ラダーのレベル i	σ_i（$\tilde{\sigma}_i$ の代わり）
p_t、$\nabla \log p_t$	$\mathbf{x}_t$ の密度、そのスコア	
$\hat{\mathbf{x}}_0$、$\hat{\epsilon}$	$\mathbf{x}_0$ と ϵ の $\mathbf{x}_t$ が与えられた下での事後平均（添字 θ 付きはネットワークによる予測）	D_θ（ノイズ除去器）、ϵ_θ
$\mathbf{u}_t$、$\mathbf{u}_\theta$	経路の周辺速度、それを予測するネットワーク	フローマッチングでは逆符号の u_t、v_t
$\mathbf{v}$	$\alpha_t\epsilon - \sigma_t\mathbf{x}_0$（$\mathbf{v}$ 予測の目標）	
π_k、μ_k、s	混合分布の各成分の重み、平均、標準偏差（ここでの k は成分の添字）	

17.2.1.2 DDPM 連鎖

VP 経路は最初から連続な経路として生まれたわけではない。Ho et al. (2020) は、点を少しずつ縮小して少しずつノイズを加える T 個の微小ステップからなるマルコフ連鎖を次のように定義した。

$$\mathbf{x}_k = \sqrt{1-\beta_k}\,\mathbf{x}_{k-1} + \sqrt{\beta_k}\,\epsilon_k, \qquad k = 1, \dots, T, \tag{17.13}$$

ここで ϵ_k は独立な標準ガウスノイズ、$\beta_k \in (0,1)$ はステップ分散である。各ステップは単位分散を単位分散に写し、ノイズが互いに独立なガウス分布に従うため、k ステップの合成は 1 回のサンプリングにまとめられる。

$$\mathbf{x}_k = \sqrt{\bar{\alpha}_k}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_k}\,\epsilon, \qquad \bar{\alpha}_k = \prod_{j=1}^{k}(1-\beta_j), \tag{17.14}$$

この式において、残存する信号はそこまでの全ステップの因子の積になる（帰納法による証明は 38.6.3.2 節にある）。訓練で連鎖を逐次シミュレーションする必要がないのは、この閉形式があるからである。どの $\mathbf{x}_k$ も、$\mathbf{x}_0$ からガウスノイズを 1 回引くだけで得られる。

各ステップを時刻 $t_k = k/T$ に配置し、$\beta_k = 1 - \bar{\alpha}_{t_k}/\bar{\alpha}_{t_{k-1}}$ と選ぶと、積は相殺されて $\bar{\alpha}_{t_k}$ になり、この連鎖は VP 経路をグリッド上で読み取ったものになる。細かいグリッドでは $\beta_k \approx \beta(t_k)/T$ が成り立つため、ステップ分散とレートは同じ量をそれぞれ 1 ステップあたりと単位時間あたりで数えたものにすぎない。Figure 17.4 に順方向の連鎖と生成時に実行する逆方向の連鎖を示す。

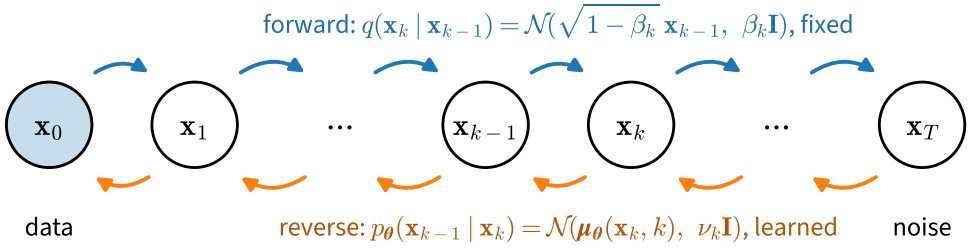


Figure 17.4: 拡散モデルのグラフィカルモデル。順過程（上）はデータ $\mathbf{x}_0$ からノイズ $\mathbf{x}_T$ へのガウス遷移からなる固定されたマルコフ連鎖であり、逆過程（下）はネットワークが平均を予測するガウス遷移からなる学習された連鎖である。

`grid` はノイズからデータまで等間隔に並べた時刻における α と σ を返し、本章のサンプラのデフォルトグリッドとなる。

```
@d2l.add_to_class(GaussianPath)
def grid(self, steps, t_start=1.0, t_end=0.0):
    """Arrays of alpha and sigma at steps + 1 even times, t_start to t_end."""
    with jax.ensure_compile_time_eval():          # constants, even under jit
        t = jnp.linspace(t_start, t_end, steps + 1)
        return self.alpha(t), self.sigma(t)
```

$T = 1000$ のとき、この経路はどのようなステップ分散を与えるだろうか。$\beta_k \approx \beta(t_k)/T$ から、最初のステップでは約 10^{-4}、最後のステップでは約 0.02 になると予想される。このグリッドはノイズ側の端から時刻を並べるため、先頭の要素が最後のステップに対応する。

```
a, _ = vp.grid(1000)
print(f'beta_1 = {1 - (a[-2] / a[-1]) ** 2:.1e}, '
      f'beta_T = {1 - (a[0] / a[1]) ** 2:.4f}, alpha_bar_T = {a[0] ** 2:.1e}')
```

```
beta_1 = 1.1e-04, beta_T = 0.0198, alpha_bar_T = 4.3e-05
```

ステップ分散は $1.1 \cdot 10^{-4}$ から 0.0198 まで増加し、Ho らの線形スケジュール（10^{-4} から 0.02）に近い値となる。1,000 ステップ後にはデータの $\sqrt{\bar{\alpha}_T} \approx 0.007$ 倍が残存する。

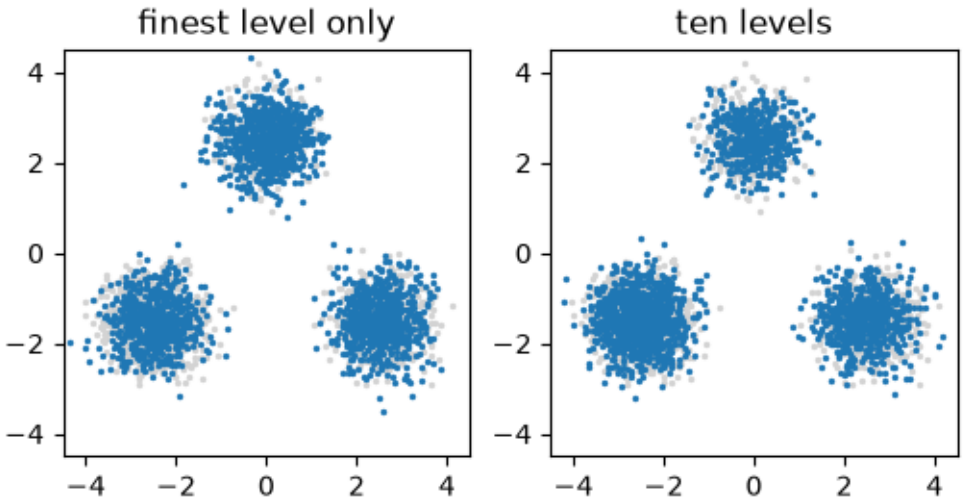
17.2.1.3 ノイズレベルにわたるアニーリング

多数のノイズレベルをたどれば、単一レベルではできなかったことが実現できるだろうか。ぼかした混合分布のスコアは厳密にわかっているため、ネットワークを一切訓練する前に、前節の VE 経路を下ることでこれを検証できる。ランジュバン更新 $\mathbf{x} \leftarrow \mathbf{x} + \frac{h}{2}\nabla \log p(\mathbf{x}) + \sqrt{h}\,\xi$（17.1 節）を、毎回新たなガウスノイズ ξ を用いながら、10 個のノイズレベル $\tilde{\sigma}_i$（対数スケールで 3 から 0.05 まで等間隔に配置）からなる**ラダー**に沿って下っていく。各レベルでは前のレベルが終了した点から出発し、厳密なスコア上で 100 ステップ実行する。ステップサイズ $h_i = 0.1\,\tilde{\sigma}_i^2$ はノイズレベルの二乗に比例して縮小するため、各ステップのノイズ $\sqrt{h_i} \approx 0.3\,\tilde{\sigma}_i$ は現在のぼかし幅に対して一定の割合に保たれる。

対照実験では同じ 1,000 ステップを最も細かいレベルだけに費やし、前節と同じステップサイズ $h = 0.05$ を用いる。このステップサイズは同じレベルにおけるラダーのステップより 200 倍大きく、谷を越える機会を対照実験に最大限与えている。どちらも同じ 2,000 個の点（$\mathcal{N}(\mathbf{0}, 9\mathbf{I})$ からのサンプル）から出発する。この考え方が正しければ、ラダーは重み 0.5、0.3、0.2 の近くに到達し、対照実験はそれぞれ約 3 分の 1 になるはずである。

```
key = jax.random.key(0)
mix = d2l.GaussianMixture.three_modes()
key, k_init, k_ctrl, k_data = jax.random.split(key, 4)
x_init = 3 * jax.random.normal(k_init, (2000, 2))
x = x_init
for sig in jnp.logspace(math.log10(3), math.log10(0.05), 10).tolist():
    key, k = jax.random.split(key)
    x = d2l.langevin(lambda z: mix.score(z, 1.0, sig), x, 0.1 * sig ** 2, 100,
                     k)
control = d2l.langevin(lambda z: mix.score(z, 1.0, 0.05), x_init, 0.05, 1000,
                       k_ctrl)
print('finest level only:', mix.summary(control))
print('ten levels:       ', mix.summary(x))
d2l.scatter_panels([control, x], ['finest level only', 'ten levels'],
                   data=mix.sample(k_data, 2000));
```

```
finest level only: {'weights': [0.312, 0.346, 0.342], 'stray': 0.017, 'spread': 0.511}
ten levels:        {'weights': [0.478, 0.305, 0.218], 'stray': 0.0165, 'spread': 0.508}
```



ラダーは真の重みから約 0.03 以内に収まり、どのモードからも遠く離れたはぐれ点はデータ自体の約 1% に対して 1〜2% にとどまる。一方、対照実験は各モードの周囲で適切な広がりを持つものの、初期点の重みである約 3 分の 1 ずつを保ったままである。モードの比率は平滑化された密度に谷がない粗いレベルで決まり、細かいレベルは各モードを先鋭化するだけである。拡散モデルも経路に沿って同じように役割を分担する。この実験にはネットワークが一切関与していないため、アイデアそのものだけを検証しており、この後で何か問題が生じるとすればそれはネットワークの責任である。

この**アニーリング付きランジュバン動力学**（annealed Langevin dynamics）は、この種の最初の生成器である Song and Ermon (2019) のノイズ条件付きスコアネットワーク（noise-conditional score network）のサンプリングに用いられた。単一のネットワークがノイズレベルを入力として受け取り、ラダーのすべてのレベルにおいて $\tilde{\sigma}_i^2$ で重み付けされたノイズ除去スコアマッチング（denoising score matching）によって訓練される（38.6.3 節）。そのレベル、ステップサイズ、ステップ数は手作業で設定されており、高次元ではその調整が微妙になる（演習問題 4）。経路はこれら 10 個のレベルを連続体に置き換えるものであり、単一のネットワークがそのすべてを学習しなければならない。

17.2.2 周辺場への回帰

単一のネットワークで経路上のすべての点を扱うことを目指す。まずは各点でネットワークが何を学習すべきかを定め、次にその正解を知らないまま学習させる回帰を導出しよう。

17.2.2.1 条件付き目標と周辺場

前節のネットワークは単一のノイズレベルにおいてスコアを学習し、それと同時に Tweedie の公式を通じてクリーンな点の事後平均 $\hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]$ を学習した。どの時刻においても、この 1 つの量がスコアを与え、ノイズ付きの点をどちらへ動かすべきかを教えてくれる。しかしこの量はデータ分布全体に依存するため未知である。そこでクリーンな点でも条件付けてみると、問題は自明になる。$\mathbf{x}_0$ が与えられれば、$\mathbf{x}_0$ の最良の推定値は $\mathbf{x}_0$ そのものだからである。

サンプラが知りたいのはノイズ付きの点をどの方向へ動かすべきかであるため、それをネットワークに直接尋ねてみよう。1 つのクリーンな点と 1 回のノイズのサンプルは、**条件付き速度**（conditional velocity）$\dot{\mathbf{x}}_t = \dot{\alpha}_t \mathbf{x}_0 + \dot{\sigma}_t \epsilon$（線形経路上では定数 $\epsilon - \mathbf{x}_0$）で移動する。与えられ

た $\mathbf{x}$ を時刻 t に通過する点は数多くあり、それぞれが固有の方向へ向かうが、点群全体としてはその地点でそれらの平均速度に従って移動する。

$$\mathbf{u}_t(\mathbf{x}) = \mathbb{E}\big[\dot{\alpha}_t\,\mathbf{x}_0 + \dot{\sigma}_t\,\epsilon \,\big|\, \mathbf{x}_t = \mathbf{x}\big], \tag{17.15}$$

これが**周辺速度**（marginal velocity）である。ただ 1 つのクリーンな点しか $\mathbf{x}$ を生成し得ない場所ではその点の速度に一致し、複数の点が生成し得る場所ではそれぞれの起こりやすさで重み付けした平均になる。

混合分布に対しては、この場を厳密に求められる。モードが与えられれば $\mathbf{x}_0$ と ϵ は $\mathbf{x}_t$ と結合ガウス分布に従うため、それぞれの条件付き平均は $\mathbf{x}_t$ の線形関数となり、`velocity` はそれらを各モードの事後確率で重み付けして平均する。

```
@d2l.add_to_class(d2l.GaussianMixture)
def velocity(self, x, t):
    """E[eps - x0 | x_t] on the linear path x_t = (1 - t) x0 + t eps."""
    log_p, r, var, a = self._modes(x, 1 - t, t)
    eps = (1 - a) * r / var                          # E[eps | x_t, mode k]
    x0 = self.means + a * self.std ** 2 / var * r  # E[x0 | x_t, mode k]
    return (jax.nn.softmax(log_p, 1)[..., None] * (eps - x0)).sum(1)
```

実データではこの場を計算できないが、学習することはできる。最小二乗法は条件付き平均を復元するため、条件付き期待値（conditional expectation）はその内部の量へ回帰することで学習できる。ノイズ除去スコアマッチングがスコアを復元したのもこの仕組みによる。速度に対してこの回帰を適用すると**条件付きフローマッチング**（conditional flow matching）損失が得られ、未知の周辺場への回帰とはネットワークに依存しない定数 C だけしか違わない。

$$\begin{aligned}&\mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left\|\mathbf{u}_\theta(\mathbf{x}_t,t) - (\dot{\alpha}_t\mathbf{x}_0 + \dot{\sigma}_t\epsilon)\right\|^2 \\ &\quad= \mathbb{E}_{t,\mathbf{x}_t}\left\|\mathbf{u}_\theta(\mathbf{x}_t,t) - \mathbf{u}_t(\mathbf{x}_t)\right\|^2 + C.\end{aligned} \tag{17.16}$$

両辺の最小点は一致するため、データ例ごとの速度で訓練したネットワークは周辺速度を学習する（証明は 38.7.1 節にある）。ノイズ除去器がスコア自体を必要としなかったのと同様に、周辺速度を学習するのに $\mathbf{u}_t$ を知る必要は一切ない。Figure 17.5 に混合分布における両方の見方を示す。$\mathbf{x}_t = \mathbf{x}$ を固定すると、線形経路の条件付き速度 $\epsilon - \mathbf{x}_0$ は $(\mathbf{x} - \mathbf{x}_0)/t$ に等しくなり、図ではサンプラが進む方向であるその負号のベクトルを描いている。

`target` は任意の経路上でのデータ例ごとの回帰目標を計算するメソッドであり、以下の訓練ではこれから速度を取得する。`rates` はフォワードモード微分によって導関数 $\dot{\alpha}_t$ と $\dot{\sigma}_t$ を供給するため、経路ごとに導関数を手書きする必要はない。残りの種類の目標と、予測値を $(\hat{\mathbf{x}}_0, \hat{\epsilon})$ へ戻す `x0_eps` については次項で扱う。

```
@d2l.add_to_class(GaussianPath)
def rates(self, t):
    """The time derivatives (alpha'(t), sigma'(t)), by forward-mode AD."""
    ones = jnp.ones_like(t)
    return (jax.jvp(self.alpha, (t,), (ones,))[1],
            jax.jvp(self.sigma, (t,), (ones,))[1])

@d2l.add_to_class(GaussianPath)
```

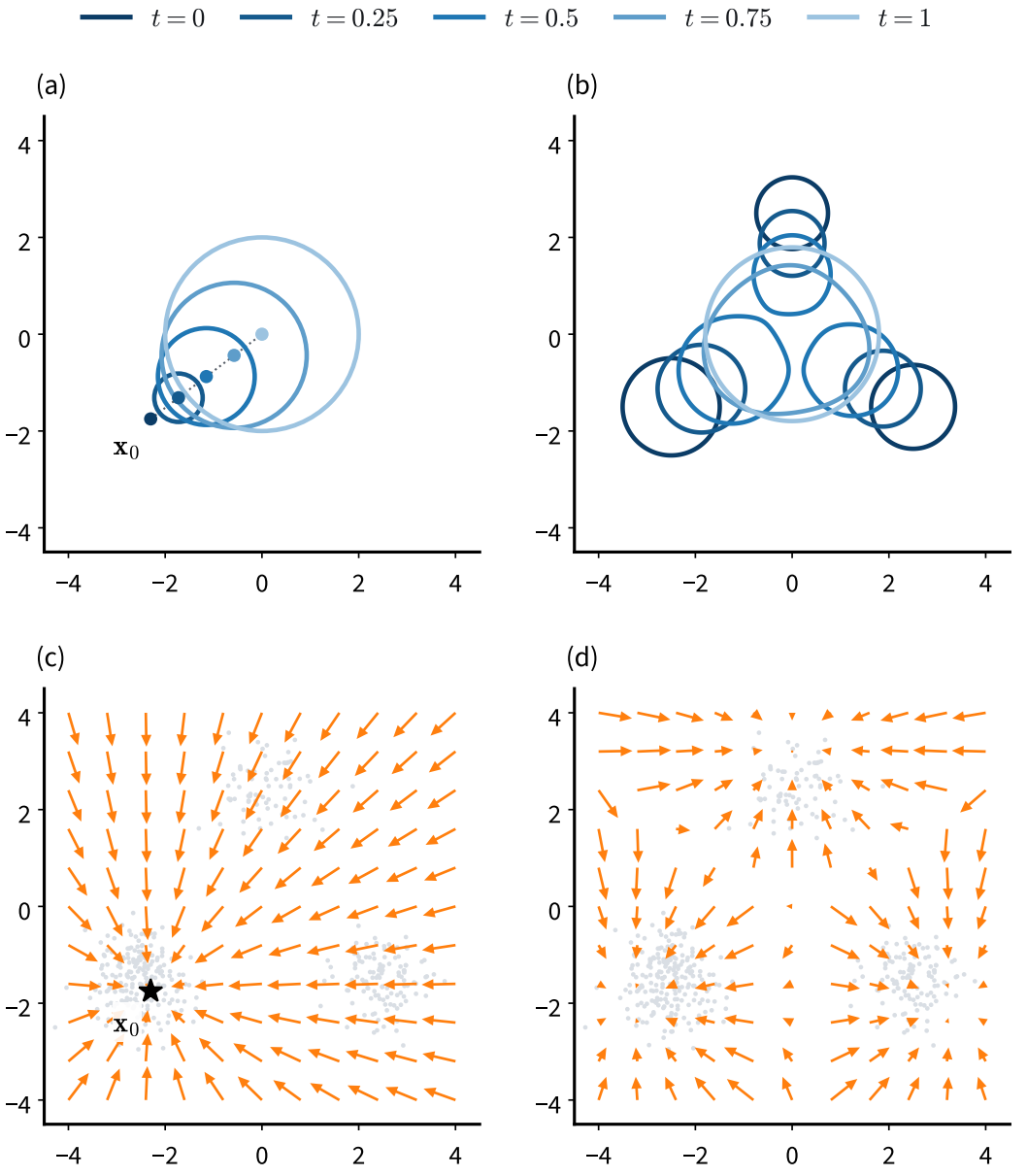


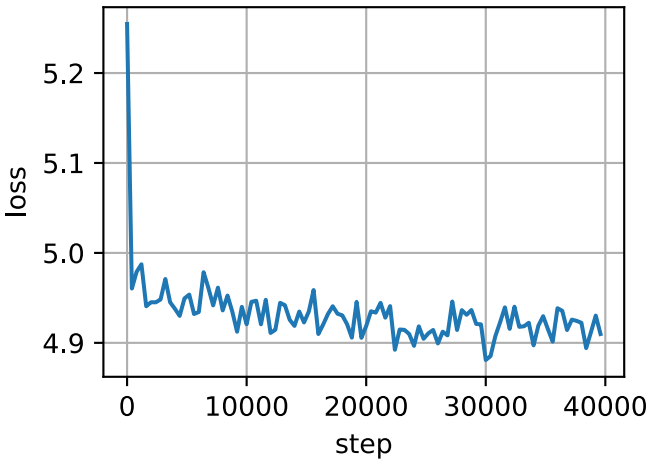
Figure 17.5: 混合分布上の線形経路における条件付きの見方と周辺的な見方（厳密計算）。(a) 最も重みの大きいモードから取った 1 つのクリーンな点 $\mathbf{x}_0$ の条件付き経路（半径 $2\sigma_t$、中心 $\alpha_t \mathbf{x}_0$ の円を 0 から 1 までの 5 つの時刻で表示）。(b) 周辺経路（同じ時刻における p_t の等高線）。(c) 条件付き場 $-(\mathbf{x} - \mathbf{x}_0)/t$（$t = 0.5$ においてその 1 点へ向かう）。(d) 周辺場 $-\mathbf{u}_{0.5}(\mathbf{x})$（条件付き場の事後平均）。

```
def target(self, kind, x0, eps, t):
    """The regression target of a kind: eps, x0, v, score or velocity."""
    t = t.reshape(-1, *[1] * (x0.ndim - 1))
    a, s = self.alpha(t), self.sigma(t)
    if kind == 'eps': return eps
    if kind == 'x0': return x0
    if kind == 'v': return a * eps - s * x0
    if kind == 'score': return -eps / s
    if kind != 'velocity': raise ValueError(f'unknown kind {kind!r}')
    da, ds = self.rates(t)                    # velocity = d x_t / dt
    return da * x0 + ds * eps

@d2l.add_to_class(GaussianPath)
def x0_eps(self, kind, out, x, t):
    """The (x0, eps) that a prediction `out` of a kind implies at (x, t)."""
    t = t.reshape(-1, *[1] * (x.ndim - 1))
    a, s = self.alpha(t), self.sigma(t)
    if kind == 'eps': return (x - s * out) / a, out
    if kind == 'x0': return out, (x - a * out) / s
    if kind == 'v': return ((a * x - s * out) / (a**2 + s**2),
                            (s * x + a * out) / (a**2 + s**2))
    if kind == 'score': return (x + s ** 2 * out) / a, -s * out
    if kind != 'velocity': raise ValueError(f'unknown kind {kind!r}')
    da, ds = self.rates(t)
    det = a * ds - s * da
    return (ds * x - s * out) / det, (a * out - da * x) / det
```

これで手法全体が 1 つのセルに収まる。各ステップでは混合分布から新たに 512 個の点を引き、それぞれに時刻とノイズを割り当てて、点と時間の正弦波特徴量を入力とする前節の小さなネットワーク `TimeMLP` を線形経路の条件付き速度へ回帰させる。訓練ではコサイン減衰する学習率を用いて Adam を 40,000 ステップ実行し、半減期 700 ステップで重みの移動平均を保持する。平均化された重みは終盤のステップのノイズを平滑化するものであり、以下ではこれを評価に用いる。前節の固定された 2,000 個の点からなる訓練セットとは異なり、バッチが毎回新しく生成されるため暗記すべき訓練セットが存在しない（訓練セットが固定されている場合に何が起きるかは演習問題 5 で扱う）。各ネットワークは 1 基の GPU 上で 1〜2 分で訓練できる。

```
def regression_loss(path, kind):
    """Squared error of net(x_t, t) against the target of a kind, t ~ U(0, 1)."""
    def loss_fn(net, x0, key):
        k_t, k_eps = jax.random.split(key)
        t = jax.random.uniform(k_t, (len(x0),))
        eps = jax.random.normal(k_eps, x0.shape)
        error = net(path.sample(x0, t, eps), t) - path.target(kind, x0, eps, t)
        return (error ** 2).reshape(len(x0), -1).sum(1).mean()  # per example
    return loss_fn

num_steps = 40_000
net_v, ema_v, losses_v = d2l.train_regression(
    d2l.TimeMLP(256, 3, rngs=nnx.Rngs(1)), regression_loss(linear, 'velocity'),
    lambda k: mix.sample(k, 512), num_steps, jax.random.key(1), cosine=True,
    ema_half_life=700)
```

```
d2l.plot(jnp.arange(100) * num_steps // 100,
         losses_v.reshape(100, -1).mean(1), 'step', 'loss')
```



曲線の各点は 400 ステップの平均である。損失は数千ステップのうちにゼロよりはるかに高い 5 付近で頭打ちになる。ネットワークは行き詰まってしまったのだろうか。実は、どのようなネットワークであってもこれより大幅に良くすることはできないことがすぐにわかる。

17.2.2.2 等価な目標

DDPM のネットワークはノイズ予測を行い、他のモデルにはクリーンな点を予測するものもある。この選択によってネットワークが学習するものは変わるのだろうか。固定された $\mathbf{x}_t$ においては、これらは異なる量ではない。Equation 17.11 の両辺の事後平均をとると $\mathbf{x}_t = \alpha_t \hat{\mathbf{x}}_0 + \sigma_t \hat{\epsilon}$（ただし $\hat{\epsilon} = \mathbb{E}[\epsilon \mid \mathbf{x}_t]$）が得られるため、一方が決まれば他方も定まる。

スコアは同じ対象の 3 つ目の見方である。前節の Tweedie の公式によれば、$\mathbb{E}[\mathbf{x}_0 \mid \tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla \log p_\sigma(\tilde{\mathbf{x}})$ が $\tilde{\mathbf{x}} = \mathbf{x}_0 + \sigma\epsilon$ に対して成り立つ。クリーンな点の役割に $\alpha_t \mathbf{x}_0$ を、ノイズレベルに σ_t を当てはめると $\alpha_t \hat{\mathbf{x}}_0 = \mathbf{x}_t + \sigma_t^2 \nabla \log p_t(\mathbf{x}_t)$ となり、スコアは次の式で表される。

$$\nabla \log p_t(\mathbf{x}_t) = \frac{\alpha_t \hat{\mathbf{x}}_0 - \mathbf{x}_t}{\sigma_t^2} = -\frac{\hat{\epsilon}}{\sigma_t}. \tag{17.17}$$

スコアは $\mathbf{x}_t$ からスケーリングされた事後平均へ向かうベクトルであり、ノイズ予測を $-\sigma_t$ で割ったものに等しい。

周辺速度 Equation 17.15 は $\dot{\alpha}_t \hat{\mathbf{x}}_0 + \dot{\sigma}_t \hat{\epsilon}$ であり、これとスコアの式から $\hat{\mathbf{x}}_0$ を消去すると、次のスコア・速度恒等式が得られる。

$$\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\alpha}_t}{\alpha_t}\mathbf{x} - \left(\sigma_t \dot{\sigma}_t - \sigma_t^2 \frac{\dot{\alpha}_t}{\alpha_t}\right) \nabla \log p_t(\mathbf{x}). \tag{17.18}$$

$\dot{\alpha}_t = -1$ かつ $\dot{\sigma}_t = 1$ である線形経路上では、この式は次のようになる。

$$\mathbf{u}_t(\mathbf{x}) = \hat{\epsilon} - \hat{\mathbf{x}}_0 = -\frac{\mathbf{x} + t \nabla \log p_t(\mathbf{x})}{1 - t}. \tag{17.19}$$

速度は点の既知の定数倍とスコアの既知の定数倍の和であり、どちらの係数も経路によって定まる。すなわち、速度モデルは姿を変えたスコアモデルにほかならない。

多くの画像モデルでは 5 つ目の目標として **v 予測** (Salimans and Ho 2022)（$\mathbf{v} = \alpha_t \epsilon - \sigma_t \mathbf{x}_0$）が用いられる。VP 経路上では $\alpha_t = \cos \phi_t$、$\sigma_t = \sin \phi_t$ と書けるため、$\mathbf{v} = d\mathbf{x}_t / d\phi_t$ は単位時間あたりではなく単位角度あたりの速度となる。小さなオフセットを除いたコサイン経路上では $\phi_t = \pi t / 2$ であり、$\mathbf{v}$ は速度の $2/\pi$ 倍になる（演習問題 1）。Salimans と Ho は、主に低い信号対雑音比でネットワークにクエリを送る極少ステップのサンプラ向けにこれを導入した。そのような領域では、ノイズ予測からクリーンな点を得るには小さな α_t で割る必要があるのに対し、$\mathbf{v}$ からは除算なしで求められる。

5 つの目標はすべて $\mathbf{x}_t$ と 1 つの事後平均のアフィン関数であり、その係数は経路と時刻のみに依存する。この厳密な事後平均を、目標が何であれ完璧なネットワークが学習するはずの予測器という意味で**オラクル**と呼び、目標間の変換を**辞書**（dictionary）と呼ぶ。Table 38.2 にこの辞書の全体を示す。`target` は $(\mathbf{x}_0, \epsilon)$ から任意の種類の目標へと順方向に辞書を引き、そのそばで定義した `x0_eps` は任意の種類の予測値から $(\hat{\mathbf{x}}_0, \hat{\epsilon})$ へと逆方向に辞書を引く。したがって、1 つの目標で訓練されたネットワークは残りの目標をすべて定め、別の経路上で訓練されたネットワークも対数信号対雑音比が等しい時刻で読み取れば同様にすべてを定める。

1 つの点でこの辞書の働きを確かめてみよう。混合分布に対して `oracle` は厳密な $\hat{\mathbf{x}}_0$ と $\hat{\epsilon}$ を返すため、どの損失やサンプラでもネットワークの代わりに利用できる。VP 経路上の点 $(0.5, -0.5)$、時刻 $t = 0.5$（$\alpha = 0.28$、$\sigma = 0.96$）において、すべての目標を表示する。最後の行では点を $1/(\alpha + \sigma)$ 倍に再スケーリングして対数信号対雑音比が等しい線形経路上の時刻 $t = 0.773$ に移し、異なる点における異なる公式である厳密な場 `velocity` を評価する。この結果は、最初の 2 行からあらかじめ計算できる $\hat{\epsilon} - \hat{\mathbf{x}}_0$ と一致するはずである。

```
    def oracle(x, alpha, sigma):
        """The exact posterior means (x0_hat, eps_hat) of the mixture at x."""
        x0_hat = mix.posterior_mean(x, alpha, sigma)
        a, s = (jnp.reshape(jnp.asarray(v), (-1, 1)) for v in (alpha, sigma))
        return x0_hat, (x - a * x0_hat) / s

    x = jnp.array([[0.5, -0.5]])                    # one noisy point
    t = jnp.array([0.5])                            # VP: alpha 0.28, sigma 0.96
    x0_hat, eps_hat = oracle(x, vp.alpha(t), vp.sigma(t))
    for kind in ('x0', 'eps', 'score', 'v', 'velocity'):
        print(f'{kind:>8} on vp:', r2(vp.target(kind, x0_hat, eps_hat, t)[0]))
    c = 1 / (vp.alpha(t) + vp.sigma(t))     # linear path, same alpha / sigma
    print(f'velocity on linear at t = {float((c * vp.sigma(t))[0]):.3f}:',
          r2(mix.velocity(c * x, c * vp.sigma(t))[0]),
          ' eps_hat - x0_hat:', r2((eps_hat - x0_hat)[0]))
```

```
      x0 on vp: [0.29, -0.98]
     eps on vp: [0.44, -0.23]
   score on vp: [-0.46, 0.24]
       v on vp: [-0.15, 0.88]
velocity on vp: [-0.22, 1.29]
velocity on linear at t = 0.773: [0.15, 0.75]  eps_hat - x0_hat: [0.15, 0.75]
```

クリーンな点の予測値は $(0.29, -0.98)$ となり、3 つの平均 $(-2.5, -1.5)$、$(2.5, -1.5)$、$(0, 2.5)$ がなす三角形の内部にあっていずれの平均からも遠い。これは $\alpha = 0.28$ では事後分布がまだ 3 つのモードすべてに広がっているためである。スコアはノイズ予測を -0.96 で割った値になっており、線形経路の場は $(0.15, 0.75) = \hat{\epsilon} - \hat{\mathbf{x}}_0$ を返している。2 つの経路上の 5 つの目標が、ただ

1 つの事後平均から導かれている。

では、どの目標を訓練すべきだろうか。オラクルに到達できるネットワークであればどれを選んでも変わらないが、実際のネットワークはオラクルに到達しないため、2 つの実務上の事実が決め手になる。第 1 に、純粋なノイズにおいては $\hat{\mathbf{x}}_0 = (\mathbf{x}_t - \sigma_t\hat{\epsilon})/\alpha_t$ がゼロ除算になるため、ノイズ予測器はクリーンな点について何も語れない。ゆえにノイズ予測器は、VP 経路のように α_t が正に保たれる経路と組み合わせるべきである。第 2 に、目標によって時刻ごとの重み付けが異なる。線形経路上では、速度損失はノイズ損失に $1/(1-t)^2$ の重みを掛けたものに等しく、またクリーンな点の損失に $1/t^2$ の重みを掛けたものに等しい（演習問題 2）。そのため、速度の二乗誤差を一様に小さくすることは、ノイズ端の近くではノイズの誤差を小さくし、データの近くではクリーンな点の誤差を小さくすることを意味する。実用的には、線形経路上の速度、コサイン経路上の $\mathbf{v}$、VP 経路上のノイズという 3 つの組み合わせが主流である。いずれも誤りではなく、辞書によって相互に変換できる。

DDPM の**単純損失**（simple loss）(Ho et al. 2020) はノイズへ回帰する。

$$L_{\text{simple}}(\theta) = \mathbb{E}_{t,\,\mathbf{x}_0,\,\epsilon}\left\|\epsilon - \epsilon_\theta(\alpha_t\mathbf{x}_0 + \sigma_t\epsilon,\, t)\right\|^2. \tag{17.20}$$

データ例ごとのスコア、すなわち条件付き密度 $\mathcal{N}(\alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I})$（$\mathbf{x}_t$ の $\mathbf{x}_0$ が与えられた下での分布）のスコアは $-\epsilon/\sigma_t$ であるため、単純損失はすべてのレベルにおいて σ_t^2 で重み付けされたノイズ除去スコアマッチングに等しい（38.6.3.2 節）。Ho らは尤度の変分下界からこの損失を導出したのち、t にわたる下界の重みを取り除いた方がサンプル品質が高くなることを見いだして重みを落とした。2 つ目のモデルでは DDPM の選択、すなわち VP 経路上のノイズ目標を採用し、それ以外は一切変更しない。

```
net_eps, ema_eps, losses_eps = d2l.train_regression(
    d2l.TimeMLP(256, 3, rngs=nnx.Rngs(2)), regression_loss(vp, 'eps'),
    lambda k: mix.sample(k, 512), num_steps, jax.random.key(2), cosine=True,
    ema_half_life=700)
```

両方のネットワークが十分に訓練できていれば、経路や目標が異なっていても同じ事後平均を保持しているはずだと辞書からわかる。これを損失の下限によって確かめよう。

17.2.2.3 損失の下限

なぜ速度の損失は 5 付近で頭打ちになったのだろうか。各目標にはそのデータ例固有のノイズが含まれており、これは $\mathbf{x}_t$ からはどのネットワークにも予測できない。オラクルでさえ、経路全体で平均した目標の条件付き分散（conditional variance）を残してしまう。前節で単一のノイズレベルにおいて登場したこの**損失の下限**（loss floor）こそ、定数 C（Equation 17.16）の正体である。$\mathbf{x}_t$ を固定するとノイズは $(\mathbf{x}_t - \alpha_t\mathbf{x}_0)/\sigma_t$ となるため、どの目標も $c_t\mathbf{x}_0$ と $\mathbf{x}_t$ の関数の和になり、時刻 t におけるその下限は次の式で与えられる。

$$c_t^2\ \mathbb{E}\,\operatorname{tr}\operatorname{Cov}[\mathbf{x}_0 \mid \mathbf{x}_t].$$

クリーンな点では $c_t = 1$、ノイズでは $c_t = -\alpha_t/\sigma_t$、線形経路の速度では $c_t = -1/t$ である。各目標の違いは、同一の事後分散（posterior variance）をどれだけ拡大するかという点にしかない。

データの近くではこの数値を解析的に見積もることができる。$t = 0.1$ ではノイズに比べてモード間の距離がまだ十分に離れているため、事後分布は単一のガウスモードの事後分布となり、1 座

標あたりの分散は $s^2\sigma_t^2/(\alpha_t^2 s^2+\sigma_t^2)$ となる。これに c_t^2 を掛けて 2 つの座標について足し合わせると、ノイズの下限 $2\alpha_t^2 s^2/(\alpha_t^2 s^2+\sigma_t^2)$ と、線形経路における速度の下限 $2s^2/(\alpha_t^2 s^2+\sigma_t^2)$（ただし $\sigma_t=t$）が得られる。$s=0.5$ のとき、VP 経路（$\alpha=0.95$、$\sigma=0.32$）のノイズの下限は約 1.37、線形経路（$\alpha=0.9$、$\sigma=0.1$）の速度の下限は 2.35 となる。

次のセルでは 100 万個のサンプルを用いて、3 つの時刻および t 全体の平均における各下限とその標準誤差を測定する。オラクルの誤差は $\mathbf{x}_t$ のあらゆる関数と無相関であるため、任意のネットワークの損失は下限と、ネットワークの予測およびオラクルの予測の平均二乗誤差の和に分解される。そこでこのセルでは、平均化ネットワークの下限に対する過剰損失をその距離として直接測定し（`excess` 列）、最後の $1{,}000$ ステップの訓練損失（`train loss` 列）と並べて表示する。最後の行では速度ネットワークを VP 経路上のノイズ予測器として読み取る。すなわち、`eps_from_velocity` が各点を対数信号対雑音比の等しい線形経路上の点へ移し、そこで速度ネットワークにクエリを送り、その出力を $\hat{\epsilon}$ へと `x0_eps` で変換する。

```
def floor_and_excess(path, kind, net, x0, eps, t):
    """The floor E||target - E[target | x_t]||^2 with its standard error, and
    the network's loss minus the floor, E||net(x_t, t) - E[target | x_t]||^2."""
    x = path.sample(x0, t, eps)
    best = path.target(kind, *oracle(x, path.alpha(t), path.sigma(t)), t)
    err = ((path.target(kind, x0, eps, t) - best) ** 2).sum(1)
    out = jnp.concatenate([net(x[i:i + 100_000], t[i:i + 100_000])  # chunks
                           for i in range(0, len(x), 100_000)])
    return (float(err.mean()), float(err.std()) / math.sqrt(len(err)),
            float(((out - best) ** 2).sum(1).mean()))

def eps_from_velocity(x, t):
    """ema_v as a noise predictor on the VP path, read at the linear-path
    time with the same alpha / sigma (the same posterior mean)."""
    c = 1 / (vp.alpha(t) + vp.sigma(t))
    t_lin, x_lin = c * vp.sigma(t), c[:, None] * x
    return linear.x0_eps('velocity', ema_v(x_lin, t_lin), x_lin, t_lin)[1]

n = 1_000_000
key, k_x0, k_eps, k_t = jax.random.split(key, 4)
x0, eps = mix.sample(k_x0, n), jax.random.normal(k_eps, (n, 2))
times = [jnp.full((n,), t) for t in (0.1, 0.5, 0.9)]
times.append(jax.random.uniform(k_t, (n,)))            # t ~ U(0, 1)
print('target, path      t=0.1  t=0.5  t=0.9  t~U(0,1)    s.e.'
      '  train loss   excess')
for name, path, kind, net, losses in (
        ('noise, vp', vp, 'eps', ema_eps, losses_eps),
        ('velocity, linear', linear, 'velocity', ema_v, losses_v)):
    rows = [floor_and_excess(path, kind, net, x0, eps, t) for t in times]
    print(f'{name:16}' + ''.join(f'{r[0]:7.3f}' for r in rows[:3])
          + f'{rows[3][0]:10.3f}{rows[3][1]:8.4f}'
          + f'{float(losses[-1000:].mean()):12.3f}{rows[3][2]:9.4f}')
floor, _, gap = floor_and_excess(vp, 'eps', eps_from_velocity, x0, eps,
                                 times[3])
print(f'velocity net read as a noise model on vp: loss {floor + gap:.3f},'
      f' excess {gap:.4f}')
```

```
target, path      t=0.1  t=0.5  t=0.9  t~U(0,1)    s.e.  train loss   excess
noise, vp         1.371  0.478  0.002     0.539  0.0011       0.541   0.0002
```

```
velocity, linear  2.355  2.812  9.156     4.920  0.0070       4.916   0.0010
velocity net read as a noise model on vp: loss 0.540, excess 0.0002
```

$t = 0.1$ における下限は、単一モードによる解析的な見積もりと 0.01 以内で一致している。経路に沿って 2 つの下限は逆方向に動く。ノイズの下限は $t = 0$ 付近（小さなノイズがモード内分散と区別できない領域）で最大となり、0.002（$t = 0.9$）まで下がるのに対し、速度の下限は ≈ 9（$t = 0.9$）まで上昇してから $\mathrm{tr}\,\mathrm{Cov}[\mathbf{x}_0]$（$t = 1$）へと落ち着く（演習問題 3）。$t \sim U(0,1)$ で平均すると、下限には 1 桁の差（ノイズの 0.54 に対して速度は 4.9）が生じる。これは目標のスケーリング c_t^2 と 2 つの経路の時計の違いによるものである。

訓練された両方のネットワークはそれぞれの理論的な下限に到達している。厳密なオラクルに対する平均二乗誤差（`excess`）はノイズネットワークで 0.0002、速度ネットワークで 0.001 であり、いずれも下限自体のモンテカルロ標準誤差を十分に下回る。さらに、線形経路の速度ネットワークに `eps_from_velocity` を介して VP ノイズ予測器としてクエリを送ると、VP ノイズ目標で直接訓練したネットワークとまったく同じ損失（0.540）および過剰損失（0.0002）を達成する。両ネットワークは背後にある同一の事後平均を学習して収束しているのである（生のスコア目標 $-\epsilon/\sigma_t$ はノイズの下限を σ_t^2 で割るため $t \to 0$ で発散する。スコアネットワークがノイズ除去スコアマッチングの目的関数に σ_t^2 の重みを掛けてノイズ予測に帰着させるのはこのためである）。こうして学習した場からサンプルを生成する方法が 17.3 節の主題となる。

17.2.3 まとめ

ガウス経路 $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$ は、$t = 0$ におけるクリーンなデータと $t = 1$ におけるガウスノイズの間を補間し、そのような経路はすべて信号対雑音比の対数である対数信号対雑音比 $\lambda_t = \log(\alpha_t^2/\sigma_t^2)$ によって与えられる標準的な時計を共有する。ノイズレベルにわたるアニーリングは単一レベルのジレンマを解消する。高ノイズレベルが平滑化された谷を越えて大域的なモード重みを確立し、低ノイズレベルが各モードを先鋭化するからである。データ例ごとの条件付き目標（$\mathbf{x}_0$、ϵ、$-\epsilon/\sigma_t$、$\mathbf{v}$、または $\dot\alpha_t \mathbf{x}_0 + \dot\sigma_t \epsilon$）へ回帰することで、$\mathbf{x}_t$ が与えられた下で対応する条件付き期待値を出力するようにネットワークを訓練できる。5 つの目標はすべて $\mathbf{x}_t$ と事後平均 $\hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]$ のアフィン関数であるため、単一の訓練済み予測器から任意のガウス経路上の 5 つの目標がすべて定まる。最後に、データ例ごとの回帰目的関数はいずれも観測できないノイズの実現値に起因する正の損失の下限 $c_t^2\, \mathbb{E}\,\mathrm{tr}\,\mathrm{Cov}[\mathbf{x}_0 \mid \mathbf{x}_t]$ を持つため、予測器の精度は生の訓練損失ではなく、この下限を上回る過剰損失によって測られる。

17.2.4 演習問題

1. **$\mathbf{v}$ から速度へ。** VP 経路上の点 $\mathbf{x} = (0.5, -0.5)$、時刻 $t = 0.5$（$\alpha_t = 0.28$、$\sigma_t = 0.96$）において、本文では厳密な予測値 $\hat{\mathbf{v}} = (-0.15, 0.88)$ が表示された。
   1. VP 経路上では $\alpha_t^2 + \sigma_t^2 = 1$ が成り立つ。$\hat{\mathbf{x}}_0 = \alpha_t \mathbf{x} - \sigma_t \hat{\mathbf{v}}$ および $\hat\epsilon = \sigma_t \mathbf{x} + \alpha_t \hat{\mathbf{v}}$ を示し、手計算によって本文で表示された $\hat{\mathbf{x}}_0$ と $\hat\epsilon$ を復元せよ。入力を 2 桁に丸めているため約 0.01 の誤差が生じる。
   2. $\alpha_t = \cos\phi_t$、$\sigma_t = \sin\phi_t$ と書くとき、周辺速度が $\mathbf{u}_t = \dot\phi_t\, \mathbf{v}$ となり、したがって $\mathbf{v}$ が単位角度あたりの速度であることを示し、コサイン経路における係数 $2/\pi$ を導出せよ。
   3. 本節の VP 経路に対して $\dot\phi_t = \beta(t)\,\alpha_t/(2\sigma_t)$ を示せ。これを $t = 0.5$ で評価し、最初の小問の丸め誤差の範囲内で、表示された速度 $(-0.22, 1.29)$ を $\hat{\mathbf{v}}$ から復元せよ。
2. **速度・ノイズ・クリーンな点の回帰目標。** 線形経路上で $\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\epsilon$ の事後平均をとる。
   1. 周辺速度が $(\mathbf{x} - \hat{\mathbf{x}}_0)/t$ と $(\hat\epsilon - \mathbf{x})/(1-t)$ の両方に等しいことを示せ。
   2. 最初の小問より、$\hat{\mathbf{x}}_0$ の誤差は t で割られて速度に伝わり、$\hat\epsilon$ の誤差は $1-t$ で割られて伝わる。それぞれの予測が速度の算出元として不適切になるのは経路のどちらの端

の近くか。また、このことからノイズモデルが $\alpha = 0$ で決してクエリされない理由をどのように説明できるか（コサイン経路上におけるすべての目標間の誤差伝播については 6（38.7 節）で扱う）。

3. $\mathbf{u}_\theta = (\mathbf{x} - \hat{\mathbf{x}}_{0,\theta})/t$ の速度損失がクリーンな点の損失 $\mathbb{E}\|\hat{\mathbf{x}}_{0,\theta} - \mathbf{x}_0\|^2$ に重み $1/t^2$ を掛けたものに等しく、$\mathbf{u}_\theta = (\hat{\epsilon}_\theta - \mathbf{x})/(1-t)$ の速度損失がノイズ損失に重み $1/(1-t)^2$ を掛けたものに等しいことを示せ。重みなしの 2 つの損失と比べたとき、フローマッチングはそれぞれどの時刻を重視しているか。

3. [code] **損失の下限**。`oracle` と `floor_and_excess` を用いよ。
   1. 線形経路上の速度目標の下限と VP 経路上のノイズ目標の下限を、t に対して区間 $(0, 1)$ 上のグリッドでプロットせよ。それぞれどこで最大になるか。表示された訓練損失は 1,000 個のミニバッチ（各 512 点）の平均である。その標準誤差を `losses_v` から推定し、表示された訓練損失と下限の差と比較せよ。
   2. Σ を混合分布の共分散行列とし、その固有値を κ_1, κ_2 とする。この共分散を持つガウスデータに対して、線形経路上の時刻 t における速度の下限が $\sum_i \kappa_i / \big((1-t)^2\kappa_i + t^2\big)$ になることを示せ。この曲線をプロットに追加し、混合分布の下限を下回り得ない理由を説明せよ。2 つの下限が経路の両端で一致するのはなぜか。
   3. スコア目標 $-\epsilon/\sigma_t$ の下限は、ノイズ目標の下限を σ_t^2 で割ったものになる。これを VP 経路上の $t = 0.01$、0.1、0.5 で評価し、t を $(0, 1)$ 上の一様分布としてこの目標でネットワークを訓練しない理由を説明せよ。
   4. ノイズモデルの訓練損失は速度モデルの約 10 分の 1 である。これはノイズモデルの方が優れていることを意味するか。VP 経路の時刻 $t \in \{0.1, 0.5, 0.9\}$ において、オラクルに対する各ネットワークの誤差を共通の単位 $\mathbb{E}\|\hat{\mathbf{x}}_0 - \hat{\mathbf{x}}_0^\star\|^2$（ただし $\hat{\mathbf{x}}_0^\star$ は `oracle` の出力）で測定せよ。`ema_eps` は `vp.x0_eps` で変換し、`ema_v` は `eps_from_velocity` に続けて $\hat{\mathbf{x}}_0 = (\mathbf{x}_t - \sigma_t\hat{\epsilon})/\alpha_t$ を適用して変換せよ。本文の係数 c_t を用いて、生の損失がこれほど大きく異なる理由を説明せよ。2 つのモデルを正しくランク付けできるのは、生の損失、下限に対する過剰損失、共通単位での誤差のうちどれか。

小問 (d) は CMU 10-799 の Homework 2（2026 年春学期）問 3(a) を改変したものである。

4. **高次元におけるラダーの間隔**。●データが $\mathbb{R}^d$ の原点にある単一の点であり、したがって $p_{\tilde{\sigma}} = \mathcal{N}(\mathbf{0}, \tilde{\sigma}^2\mathbf{I})$ であるとし、ラダーが公比 $q = \tilde{\sigma}_{i-1}/\tilde{\sigma}_i > 1$ の等比列であるとする。
   1. $r = \|\mathbf{x}\|$（$\mathbf{x} \sim p_{\tilde{\sigma}}$）が $\mathbb{E}[r^2] = d\tilde{\sigma}^2$ および $\mathrm{Var}[r^2] = 2d\tilde{\sigma}^4$ を満たすことを示し、r が近似的に $\mathcal{N}(\tilde{\sigma}\sqrt{d}, \tilde{\sigma}^2/2)$（$d$ が大きいとき）に従うことを導け。
   2. レベル $i-1$ を $p_{\tilde{\sigma}_{i-1}}$ のサンプルとともに離れた連鎖は、レベル i に入るとき高い確率で区間 $\tilde{\sigma}_{i-1}(\sqrt{d} \pm 3/\sqrt{2})$ 内の半径にある。$p_{\tilde{\sigma}_i}$ がこの区間に割り当てる確率質量が $\Phi\big(\sqrt{2d}(q-1) + 3q\big) - \Phi\big(\sqrt{2d}(q-1) - 3q\big)$ であることを示せ（ただし Φ は標準正規分布の累積分布関数である）。この確率質量がゼロに近いと、ネットワークが訓練入力をほとんど見たことのない領域から連鎖がそのレベルを開始することになるのはなぜか。
   3. Y. Song and Ermon (2020) はこの確率質量が約 2 分の 1 になるように q を選んでいる。q を CIFAR-10 画像の次元 $d = 3072$ において求め、$\tilde{\sigma} = 50$ から 0.01 までのレベル数を数えよ。これを同論文の CIFAR-10 モデルにおける 232 レベルと比較せよ。
   4. 最上位のレベルはモードを融合させるのに十分な粗さも備えていなければならない。$\pm\mu$ に位置し分散 v を持つ 2 つの等しいガウス成分が単一のモードを持つための必要十分条件は $\mu^2 \le v$ である（4（38.5 節））。この基準を混合分布の下側 2 つのモードに適用して両者が融合するノイズレベルを見積もり、ラダーの最上位レベルである 3 と比較せよ。

5. [code] **厳密な最適解は暗記する**。有限個の訓練セットはそれ自体が 1 つの分布、すなわち点質量の混合分布であり、その厳密な速度場は既知である。Bertrand et al. (2025) は、ノイズからこの場をたどると訓練点がそのまま再現されることを出発点とし、それにもかかわらず訓練されたネットワークがなぜ汎化するのかを問うている。

1. 混合分布から 64 個の点を引き、それらの点のみから抽出したバッチ上で `TimeMLP(256, 3)` を `regression_loss(linear, 'velocity')` により訓練せよ。
2. `d2l.GaussianMixture(points.tolist(), [1/64] * 64, 0.0)` は経験分布を表し、その `velocity` メソッドは $t > 0$ における厳密な場を与える。標準ガウスノイズから引いた同じ 2,000 個の初期点から、この厳密な場と訓練済みネットワークの双方を、100 回の直線ステップ $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{u}_t(\mathbf{x})/100$（$t = 1, 0.99, \dots, 0.01$）でたどれ。両方のサンプル集合について、最近傍の訓練点までの距離のヒストグラムをプロットせよ。
3. 訓練セットをそのまま再現するのはどちらの場で、新しい点を生成するのはどちらか。Bertrand らは、ネットワークを汎化させているのは回帰目標に含まれるノイズではないと論じている。訓練点を 1,024 個に増やして最初の 2 つの小問を繰り返し、2 つのヒストグラムがどのように変化するか述べよ。

17.3 積分によるサンプリング

ガウス経路（Gaussian path）に沿って訓練されたネットワークは、すべてのノイズレベル $t \in [0, 1]$ において周辺速度（marginal velocity）$\mathbf{u}_t(\mathbf{x})$（あるいは同値なスコア $\nabla \log p_t(\mathbf{x})$）を予測する。この決定論的なベクトル場を $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$（$t = 1$）から $t = 0$ まで逆向きに積分すると、なぜ p_0 に従うサンプルが、固定レベルのランジュバン連鎖が見逃した正確なモード重み $0.5, 0.3, 0.2$ を伴って生成されるのだろうか。

その保証は、確率輸送の**連続の式**（continuity equation）からもたらされる。各条件付きガウス分布の雲 $p_t(\mathbf{x} \mid \mathbf{x}_0)$ はそれぞれの条件付き速度（conditional velocity）のもとで質量保存則を満たすため、$p(\mathbf{x}_0)$ について平均をとれば、周辺密度 p_t が事後平均速度 $\mathbf{u}_t(\mathbf{x})$ によって輸送されることがわかる。Y. Song, Sohl-Dickstein, et al. (2021) はこの**確率フロー ODE**（probability-flow ODE）による決定論的サンプリングを定式化し、一方で Ho et al. (2020) は確率的な逆マルコフ連鎖によってサンプリングした。後に示すように、DDPM の各祖先ステップは、確率フロー ODE のオイラーステップと現在のノイズレベルにおけるランジュバンステップの和へと一次近似で分解される。本節では、これら両方のサンプラを導出し、経路に依存しない $(\bar{\mathbf{x}}, \bar{\sigma})$ 座標系で ODE を表したうえで、3 モード混合分布において訓練済みネットワークを厳密なオラクルと比較する。

各節は独立したノートブックとして動作するため、下のセルでは 17.2 節から必要な要素、すなわち 3 モード混合分布とその 2 つの経路である分散保持（variance-preserving、VP）経路および線形経路を再構築する。また、前節と同様に、任意の点とノイズレベルにおける混合分布の厳密な事後平均 $(\hat{\mathbf{x}}_0, \hat{\epsilon})$ を `oracle` という関数名で定義する。さらに、本節で扱う 2 つのサンプラに対応して 2 つのネットワークを再訓練する。1 つはフロー用の線形経路上の速度モデルであり、もう 1 つは DDPM の連鎖用の VP 経路上のノイズモデルである。訓練レシピは前節のステップ数を半分にしたもので、2 万ステップで十分である。これを確認するため、直近 1,000 ステップの平均訓練損失を表示し、前節で求めた下限（速度では約 4.9、ノイズでは 0.54）と比較する。2 つの訓練を合わせても、1 基の GPU で約 1 分で完了する。

```
key = jax.random.key(0)
mix = d2l.GaussianMixture.three_modes()
vp, linear = d2l.GaussianPath.vp(), d2l.GaussianPath.linear()

def oracle(x, alpha, sigma):
    """The exact posterior means (x0_hat, eps_hat) of the mixture at x."""
    x0_hat = mix.posterior_mean(x, alpha, sigma)
    a, s = (jnp.reshape(jnp.asarray(v), (-1, 1)) for v in (alpha, sigma))
    return x0_hat, (x - a * x0_hat) / s
```

```
num_steps = 20_000
_, ema_v, losses_v = d2l.train_regression(
    d2l.TimeMLP(256, 3, rngs=nnx.Rngs(1)),
    d2l.regression_loss(linear, 'velocity'), lambda k: mix.sample(k, 512),
    num_steps, jax.random.key(1), cosine=True, ema_half_life=700)
_, ema_eps, losses_eps = d2l.train_regression(
    d2l.TimeMLP(256, 3, rngs=nnx.Rngs(2)), d2l.regression_loss(vp, 'eps'),
    lambda k: mix.sample(k, 512), num_steps, jax.random.key(2), cosine=True,
    ema_half_life=700)
print(f'loss over the last 1,000 steps: velocity '
      f'{float(losses_v[-1000:].mean()):.3f}, '
      f'noise {float(losses_eps[-1000:].mean()):.3f}')
```

```
loss over the last 1,000 steps: velocity 4.913, noise 0.536
```

どちらの損失も下限に達しており、いずれのネットワークにも回帰からこれ以上学習すべきことはほとんど残っていない。肝心のサンプルにおいてこれらのネットワークがどれだけオラクルに迫れるかは、本節の後半で検証する。

17.3.1 確率質量の輸送

まず、速度場が密度に何をもたらすかから始めよう。p_t に従って分布する点の雲を考え、各点をその位置における速度 $\mathbf{u}_t(\mathbf{x})$ で微小時間だけ動かしてみる。点は生成も消滅もしないため、$\mathbf{x}$ における密度の変化は、流入する質量から流出する質量を引いた差によってのみ生じる。単位時間あたりに小さな窓を通過する質量は、密度と速度の積 $p_t\mathbf{u}_t$ である。この流れが $\mathbf{x}$ の周囲の微小な箱から単位体積あたりに正味で運び出す量は、その発散 $\nabla \cdot (p_t\mathbf{u}_t) = \sum_i \partial (p_t u_{t,i})/\partial x_i$ であり、その場所の密度はまさにその割合で減少する。これが流体力学における**連続の式**である。

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla \cdot \big(p_t(\mathbf{x})\,\mathbf{u}_t(\mathbf{x})\big).$$

密度と速度場がこの式を満たすとき、場に従って動く点は時間を順方向に進めても逆方向に進めても、すべての時刻で経路の密度を保つ。では、17.2 節の周辺速度は p_t とともにこの式を満たすだろうか。

まず 1 つのクリーンな点 $\mathbf{x}_0$ を考えよう。これにノイズを加えた点はガウス分布の雲 $\mathcal{N}(\alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I})$ をなし、雲の各点は条件付き速度 $\dot\alpha_t\mathbf{x}_0 + \dot\sigma_t\epsilon$ で動く。ここで $\epsilon = (\mathbf{x} - \alpha_t\mathbf{x}_0)/\sigma_t$ は、その点を位置 $\mathbf{x}$ に置いたノイズである。この場を $\mathbf{u}_t(\mathbf{x} \mid \mathbf{x}_0)$ と呼ぶことにすると、これはこの 1 つの雲を正しく動かす。すなわち、同じノイズ ϵ を保ったまま動く点は各時刻で $\alpha_t\mathbf{x}_0 + \sigma_t\epsilon$ に位置し、常にその雲からのサンプルとなっているため、この雲とその速度場は連続の式を満たす。周辺密度 p_t はデータ全体にわたる雲の平均であり、連続の式は密度に関して線形であるため、各雲の式を平均すると、右辺に平均質量流を持つ p_t の式が得られる。ベイズの定理 $p(\mathbf{x}_0)\,p_t(\mathbf{x} \mid \mathbf{x}_0) = p_t(\mathbf{x})\,p(\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x})$ により、この流れは次のようになる。

$$\begin{aligned}\mathbb{E}_{\mathbf{x}_0}\big[p_t(\mathbf{x} \mid \mathbf{x}_0)\,\mathbf{u}_t(\mathbf{x} \mid \mathbf{x}_0)\big] &= p_t(\mathbf{x})\,\mathbb{E}\big[\dot\alpha_t\mathbf{x}_0 + \dot\sigma_t\epsilon \,\big|\, \mathbf{x}_t = \mathbf{x}\big] \\ &= p_t(\mathbf{x})\,\mathbf{u}_t(\mathbf{x}).\end{aligned}$$

途中に現れた事後分布による平均は、周辺速度の定義そのものである。したがって、すべての雲が合わせて $\mathbf{x}$ を横切って運ぶ質量は、その位置の密度と周辺速度の積に等しく、p_t と $\mathbf{u}_t$ は連続

の式を満たす。次節以降のサンプラはこの結果に基づいている。必要な条件を含めた厳密な証明は 38.7 節で与える。

速度を平均しても軌跡の平均にはならないではないか、と疑問に思う読者もいるだろう。その指摘は正しい。条件付き経路（線形経路では直線）はあらゆる時刻であらゆる点を通過し、いたるところで互いに交差する。これに対し、1 つのベクトル場の 2 本の軌跡が同じ時刻に交わることは決してない。ある時刻のある点における速度はただ 1 つしか存在しないからである。$\mathbf{u}_t$ に従うサンプルは、いかなる条件付き経路もたどらない。Figure 17.6 に、1 次元の 2 成分混合ガウス分布に対する両方の曲線を示す。オレンジ色で描かれたノイズは右側のモードにあるクリーンな点から生じたものだが、その軌跡は左側のモードに行き着く。上の議論は個々の軌跡については何も主張していない。主張しているのは、この速度場があらゆる時刻で**密度**を再現するということであり、サンプラに必要なのはその密度だけである。

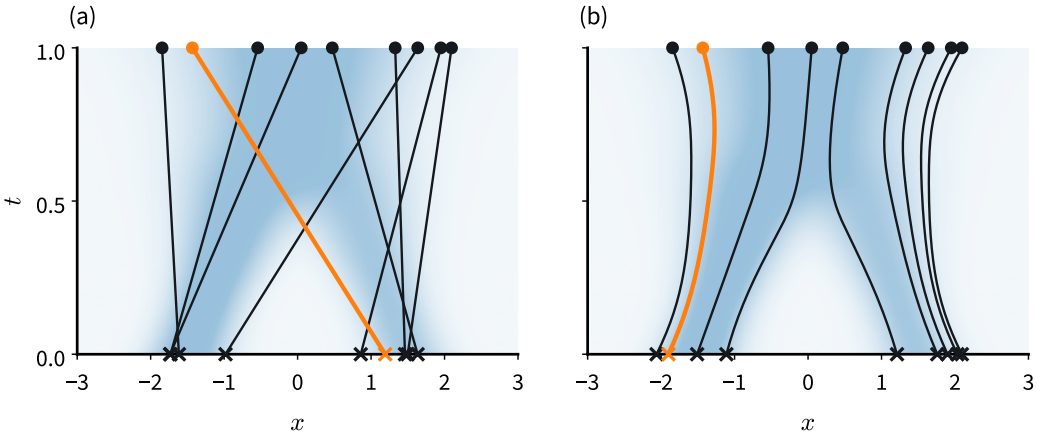


Figure 17.6: 1 次元の 2 成分混合ガウス分布における、線形経路上の条件付き経路と軌跡。時間は $t = 0$ のデータ（下）から $t = 1$ のノイズ（上）へと進み、陰影は密度 p_t を表す。(a) クリーンな点 $\mathbf{x}_0$（バツ印）とノイズ ϵ（点）の 9 組のペアが作る条件付き経路は直線であり、互いに交差する。(b) 同じ 9 個のノイズから出発する周辺速度場 $\mathbf{u}_t$ の軌跡は決して交わらない。オレンジ色のノイズは、条件付き経路に沿えば右側のモードに行き着くが、軌跡に沿えば左側のモードに行き着く。

これで冒頭の問いに答えが出た。厳密に解けば、フローはあらゆる瞬間で質量を保存するため、モード重みを誤ることはあり得ない。ノイズの半分は必ず重み 1/2 のモードに行き着く。ランジュバン連鎖は、一度データ分布に到達すればそれを定常に保てるものの、谷を越えてそこへ運ぶ仕組みを持たなかった。アニーリングラダーは、ノイズが谷を埋める粗いレベルで谷を越えたが、必要なレベル数やステップ数は手探りで決めるしかなかった。これに対し、フローは既知の密度である標準ガウス分布から出発し、理論的な保証をもってデータへと輸送する。それでも問題が生じ得る箇所は 2 つ残っている。離散ステップは解を近似するにすぎず、ネットワークも速度場を近似するにすぎない点である。本節の残りでは、この両方の誤差を測定する。

以上より、このサンプラは初期点 $\mathbf{x}$ を $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ から抽出し、次の常微分方程式（ordinary differential equation、ODE）を解く。

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}_t(\mathbf{x}), \qquad \text{from } t = 1 \text{ down to } t = 0. \tag{17.21}$$

すべての時刻においてその解は p_t からのサンプルであり、$t = 0$ ではデータからのサンプルとな

る。Y. Song, Sohl-Dickstein, et al. (2021) はこれを**確率フロー ODE**（38.5.4.3 節）と呼んだ。出発後にランダムな要素は一切入らず、サンプルのランダム性はすべてその初期点に集約されている。

17.3.2 確率フロー ODE に対するオイラー法

ODE は、小さな直線ステップを数多く積み重ねることで数値的に解かれる。**オイラー法**（Euler's method、38.2 節）は、時刻 1 から 0 までを K 個の等間隔なステップに分割し、各ステップで点を速度と逆向きに $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{u}_t(\mathbf{x})/K$（$t = 1, 1-1/K, \dots, 1/K$）と動かす。では、フローには何ステップ必要なのだろうか。一般のデータについて直接考えるのは難しいため、Nakkiran et al. (2025) にならい、答えが自明なデータから始めて段階的に議論を組み立てていこう。

17.3.2.1 1 点のデータからデータ平均へ

データが 1 つの点 $\mathbf{x}_0$ のみからなる場合を考える。線形経路（$\alpha_t = 1-t$、$\sigma_t = t$）上では、ある点 $\mathbf{x}$ は時刻 t においてノイズ $\epsilon = (\mathbf{x} - (1-t)\mathbf{x}_0)/t$ を持ち、速度 $\epsilon - \mathbf{x}_0 = (\mathbf{x} - \mathbf{x}_0)/t$ で動く。すべての軌跡は一定速度で $\mathbf{x}_0$ へと向かう直線となり、長さ 1 のオイラーステップ 1 回、すなわち $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{u}_1(\mathbf{x}) = \mathbf{x}_0$ で厳密な解が得られる。データ点が 2 つになると、速度場はそのような 2 つの直線的な速度場を事後確率で重み付け平均したものとなり、事後分布が一方の点から他方の点へと切り替わる場所で軌跡が曲がる（38.7.1.4 節）。

任意の分布は点質量の混合分布とみなせるため、事後分布がいずれの点とも決まらない場所では必ず軌跡が曲がる。$t = 1$ においては、事後分布は完全に未決定である。純粋なノイズの点からは元のクリーンな点について何の手がかりも得られないため、事後分布は事前分布に一致し、次の関係が成り立つ。

$$\mathbf{u}_1(\mathbf{x}) = \mathbb{E}\left[\epsilon - \mathbf{x}_0 \,\middle|\, \mathbf{x}_1 = \mathbf{x}\right] = \mathbf{x} - \mathbb{E}[\mathbf{x}_0].$$

したがって、長さ 1 のオイラーステップを 1 回行うと、どの初期点もデータの平均へと送られる。本節の混合分布については、コードを実行する前から結果を予測できる。すなわち $0.5\,(-2.5, -1.5) + 0.3\,(2.5, -1.5) + 0.2\,(0, 2.5) = (-0.5, -0.7)$ であり、これは各モードの間の、最も近いモードからも標準偏差の 4 倍以上離れた点である。ステップ数を増やすとどうなるかを見るため、線形経路上の混合分布の厳密な速度場 `mix.velocity` に対し、原点および原点から距離 1.5 にある 4 点の計 5 個の固定初期点から、1、10、100 ステップでオイラー法を実行してみよう。

```
def euler(net, x, steps):
    """Euler's method on dx/dt = net(x, t), from t = 1 down to t = 0."""
    for k in range(steps, 0, -1):
        t = jnp.full((len(x),), k / steps)
        x = x - net(x, t) / steps
    return x

starts = jnp.array([[0.0, 0.0], [1.5, 0.0], [-1.5, 0.0], [0.0, 1.5],
                    [0.0, -1.5]])
ends = jnp.stack([starts] + [euler(mix.velocity, starts, steps)
                             for steps in (1, 10, 100)], 1)
print('start          1 step          10 steps        100 steps')
for row in ends.tolist():
    print('  '.join(f'({x:5.2f}, {y:5.2f})' for x, y in row))
```

```
start          1 step          10 steps        100 steps
( 0.00,  0.00) (-0.50, -0.70) (-1.91, -1.27) (-1.98, -1.30)
( 1.50,  0.00) (-0.50, -0.70) ( 2.73, -1.34) ( 2.90, -1.36)
(-1.50,  0.00) (-0.50, -0.70) (-2.92, -1.38) (-3.06, -1.40)
( 0.00,  1.50) (-0.50, -0.70) (-0.11,  2.59) (-0.09,  2.77)
( 0.00, -1.50) (-0.50, -0.70) (-1.98, -2.06) (-2.05, -2.17)
```

1 ステップの列は予測どおりであり、5 つの初期点すべてがデータの存在しない $(-0.50, -0.70)$ に着地している。10 ステップになると、すべての初期点がすでにいずれかのモードの内部に入り、100 ステップの終点から約 0.2 以内の位置に達する。ただし、いずれもデータ平均に近い手前側にとどまっており、粗いステップには 1 ステップの誤差がわずかに残っている。次に、各初期点が**どの**モードに到達するかに注目しよう。原点、その下の初期点、および左の初期点はすべて最も重みの大きいモードに行き着き、右の初期点は右側のモード、上の初期点は上側のモードに行き着く。ここに質量保存則が働いている。標準ガウス分布の半分は重み 1/2 のモードに行き着かなければならないため、そのモードが最も広い初期点の領域（ここではガウス分布の中心を含む）を集めるのである。また、終点がモードの中心に集中するわけでもない。原点は $(-1.98, -1.30)$ に行き着くが、これはモードの中心からモードの標準偏差で約 1 つ分離れた位置である。フローはノイズからデータへの写像であり、それぞれの初期点が固有のサンプルに対応している。

17.3.2.2 すべての経路に共通する座標系

時刻 t に関するオイラーステップは経路に依存する。VP 経路上では、ノイズレベルは $\sigma_t = 0.96$ $(t = 0.5)$ にとどまり、そこから 0.32 $(t = 0.1)$ まで低下する（17.2 節）。つまり、前半の時間ではノイズがほとんど除去されず、その 3 分の 2 が次の 5 分の 2 の時間で除去される。t に関するステップは始点における変化率のまま進むため、このように曲がった経路上では大きなステップをとると大きく外れてしまう（演習問題 1）。そこで本書のライブラリのサンプラでは、あらゆるガウス経路が同じ形に見える座標系でステップを進める。経路を α_t で割ってみよう。

$$\bar{\mathbf{x}} = \frac{\mathbf{x}_t}{\alpha_t} = \mathbf{x}_0 + \bar{\sigma}\,\epsilon, \qquad \bar{\sigma} = \frac{\sigma_t}{\alpha_t}.$$

この座標系ではデータのスケールは一定に保たれ、ノイズだけがデータの $\bar{\sigma} = 0$ から純粋なノイズの $\bar{\sigma} = \infty$ まで増大する。あらゆる経路が 17.2 節の分散爆発（variance-exploding、VE）経路（アニーリングラダーが下っていった経路）になり、$\bar{\sigma}$ がそのノイズレベルとなる。Karras et al. (2022) はすべての経路をこの方法でサンプリングしている。ここでの上線（バー）は α_t による除算を表しており、DDPM の $\bar{\alpha}_t = \alpha_t^2$ における二乗の意味ではない（Table 17.1）。1 つの条件付き経路に沿うと、$\bar{\mathbf{x}}$ は $\bar{\sigma}$ に対して傾き ϵ の直線上を動く。周辺速度場はこの傾きの事後平均、すなわちノイズ予測（noise prediction）である。

$$\frac{d\bar{\mathbf{x}}}{d\bar{\sigma}} = \hat{\epsilon}(\bar{\mathbf{x}}).$$

最初の式の事後平均をとると $\bar{\mathbf{x}}_t = \hat{\mathbf{x}}_0 + \bar{\sigma}_t\hat{\epsilon}$ となるため、$\bar{\sigma}_t$ から $\bar{\sigma}_s$ へのオイラーステップは $\bar{\mathbf{x}}_s = \bar{\mathbf{x}}_t + (\bar{\sigma}_s - \bar{\sigma}_t)\hat{\epsilon} = \hat{\mathbf{x}}_0 + \bar{\sigma}_s\hat{\epsilon}$ となり、これに α_s を掛けると次式を得る。

$$\mathbf{x}_s = \alpha_s\,\hat{\mathbf{x}}_0 + \sigma_s\,\hat{\epsilon}.$$

このように、1 回のステップはクリーンな点とノイズを予測し、それらを次のノイズレベルに配置する。この更新式には 17.5 節で DDIM という名前のもとで再び出会うことになる。線形経路上ではこれは t に関するオイラーステップそのものであり、それは $\mathbf{x}_t = (1-t)\hat{\mathbf{x}}_0 + t\hat{\epsilon}$ を

$\mathbf{x}_t - (t-s)(\hat{\epsilon} - \hat{\mathbf{x}}_0)$ に代入すれば確かめられる。他の経路上では異なる手法となり、1 点のみからなるデータに対しては任意のステップ幅で厳密になるのに対し、上で見た VP 経路のストライドにおける t のオイラーステップは大きく外れる（演習問題 1）。

`ode_sample` は、レベル $\bar{\sigma}$ のリスト（`sigma_bars` が経路の t グリッドから計算する）に沿ってこのステップを進める。純粋なノイズでは $\bar{\sigma}$ は無限大となり $\bar{\mathbf{x}}$ は定義されないが、ステップに必要なのは $\hat{\mathbf{x}}_0$ と $\hat{\epsilon}$ だけであり、予測器はこれらをノイズそのものから計算できるため、最初のステップは $\mathbf{x}$ から開始する。

```
@d2l.add_to_class(d2l.GaussianPath)
def sigma_bars(self, steps):
    """sigma_bar = sigma / alpha on the grid of steps; inf where alpha = 0."""
    a, s = self.grid(steps)
    with jax.ensure_compile_time_eval():         # constants, even under jit
        return s / a

def pred_bar(pred, xb, r):
    """pred at x = alpha x_bar, sigma_bar = r; at r = inf, x_bar is noise."""
    finite = jnp.isfinite(r)
    a = jnp.where(finite, 1 / jnp.sqrt(1 + r ** 2), 0.0)
    return pred(xb * jnp.where(finite, a, 1.0), a,
                jnp.where(finite, r * a, 1.0))

def ode_sample(pred, x, sigma_bars):
    """Euler (= DDIM) on d x_bar / d sigma_bar = eps, from x to data; x is the
    noisy point at the first level, unit-variance scale (noise if inf)."""
    with jax.ensure_compile_time_eval():         # the grid is a constant
        r = jnp.asarray(sigma_bars)
        from_inf = bool(jnp.isinf(r[0]))
    def step(xb, r):
        r_t, r_s = r
        x0, eps = pred_bar(pred, xb, r_t)
        return x0 + r_s * eps, None              # the Euler (= DDIM) step
    xb = x if from_inf else x * jnp.sqrt(1 + r[0] ** 2)  # x / alpha
    return jax.lax.scan(step, xb, (r[:-1], r[1:]))[0]
```

サンプラは $\hat{\mathbf{x}}_0$ と $\hat{\epsilon}$ を任意の (α, σ) で必要とし、`make_pred` はどの種類の訓練済みネットワークであってもそのような予測器に変換する。本節では各ネットワークをそれ自身の経路上で動かすため、`make_pred` は予測値をこのペアに変換するだけである。後の節では別の経路のグリッド上でネットワークを動かすが、その場合 `make_pred` はまずネットワーク自身の経路上で同じ対数信号対雑音比（log-SNR）を持つ時刻を（`time_at` 内の二分法によって）見つけ、点をその経路に合わせて再スケーリングする。なお、ノイズモデルには 1 つ死角がある。純粋なノイズ $\alpha = 0$ ではクリーンな点 $\hat{\mathbf{x}}_0 = (\mathbf{x} - \sigma\hat{\epsilon})/\alpha$ が定義されないため、ノイズモデルは線形経路上でサンプラを開始できない。一方、VP 経路上では α は常に正に保たれ、約 0.007（$t = 1$）となるため、ノイズモデルはそちらで用いる。

```
@d2l.add_to_class(d2l.GaussianPath)
def time_at(self, log_snr):
    """The time with a given log-SNR, by bisection (lambda_t falls in t)."""
    def halve(_, interval):
        lo, hi = interval
```

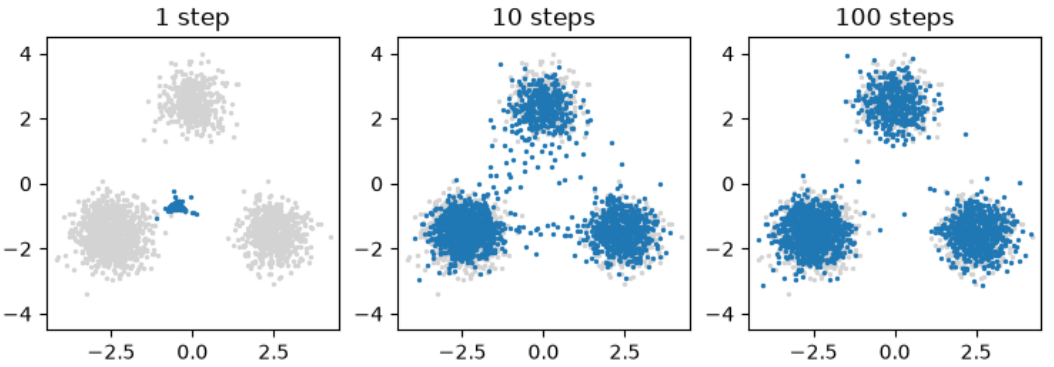
```
            mid = (lo + hi) / 2
            later = self.log_snr(mid) > log_snr      # the answer lies after mid
            return jnp.where(later, mid, lo), jnp.where(later, hi, mid)
        lo, hi = jax.lax.fori_loop(0, 50, halve, (jnp.zeros_like(log_snr),
                                                  jnp.ones_like(log_snr)))
        return (lo + hi) / 2

    def make_pred(net, path, kind):
        """pred(x, alpha, sigma) -> (x0, eps) from net(x, t) of a kind on path."""
        def pred(x, alpha, sigma):
            log_snr = jnp.where(alpha > 0, 2 * jnp.log(alpha / sigma), -jnp.inf)
            t = path.time_at(jnp.asarray(log_snr, jnp.float32))
            x = x * (path.sigma(t) / sigma)         # the same point on our path
            t = jnp.full((len(x),), t)
            return path.x0_eps(kind, net(x, t), x, t)
        return pred
```

17.3.2.3 学習された速度場

訓練した速度ネットワークは厳密な速度場と同じように振る舞い、1 ステップ後にデータ平均に着地し、10 ステップ後に各モードへ収まるだろうか。標準ガウス分布から抽出した 2,000 個の初期点から、`ode_sample` を 1、10、100 ステップ実行して確かめよう。数値の意味を読み取りやすくするため、さらに 2 行追加する。データそのものから抽出した 2,000 点（プロットの灰色の点）と、同じ初期点からオラクルで同じ 10 ステップを実行した結果である。最後に、オラクルを用いた `ode_sample` と、`mix.velocity` を用いた単純なオイラー法のループを比較する。線形経路上では両者は同一のステップを踏むため、丸め誤差の範囲で一致するはずである。

```
key, k_T = jax.random.split(key)
x_T = jax.random.normal(k_T, (2000, 2))
pred_v = make_pred(ema_v, linear, 'velocity')
samples = [ode_sample(pred_v, x_T, linear.sigma_bars(steps))
           for steps in (1, 10, 100)]
key, k_data = jax.random.split(key)
data = mix.sample(k_data, 2000)
print('one step: mean', [round(v, 3) for v in samples[0].mean(0).tolist()],
      'std', [round(v, 3) for v in samples[0].std(0).tolist()])
print(f'{"data":>18}:', mix.summary(data))
for steps, x in zip((10, 100), samples[1:]):
    print(f'{steps:>3} steps, network:', mix.summary(x))
print('  10 steps, oracle:',
      mix.summary(ode_sample(oracle, x_T, linear.sigma_bars(10))))
exact = float(jnp.abs(ode_sample(oracle, x_T, linear.sigma_bars(100))
                      - euler(mix.velocity, x_T, 100)).max())
print(f'ode_sample against the loop on the exact field: {exact:.0e}')
d2l.scatter_panels(samples, ['1 step', '10 steps', '100 steps'], data=data);
```

```
one step: mean [-0.495, -0.706] std [0.053, 0.033]
              data: {'weights': [0.512, 0.291, 0.197], 'stray': 0.0115, 'spread': 0.481}
 10 steps, network: {'weights': [0.496, 0.307, 0.197], 'stray': 0.0315, 'spread': 0.452}
100 steps, network: {'weights': [0.485, 0.313, 0.203], 'stray': 0.0135, 'spread': 0.477}
  10 steps, oracle: {'weights': [0.495, 0.31, 0.195], 'stray': 0.0285, 'spread': 0.456}
ode_sample against the loop on the exact field: 5e-05
```



1 ステップでは、すべての初期点がデータ平均 $\approx (-0.5, -0.7)$ へと崩壊する（$t = 1$ における小数第 2 位程度の小さなネットワーク近似誤差を除く）。10 回のオイラーステップを実行すると、3 つのモードが $0.5, 0.3, 0.2$ に近い重みで分離されるものの、$\approx 3\%$ の逸脱点が生じ（真のデータでは 1.1%〜1.5%）、モード内の広がりもわずかに狭くなる（0.45 対 0.48〜0.49）。厳密な `oracle` を用いて 10 回のオイラーステップを実行すると、同じモード重み（差は 0.003 以内）と 0.46 の広がりが得られる。厳密な速度場でもネットワークとほぼ同じだけモードが狭まることから、この縮小はソルバに起因している。一方、逸脱点には 2 つの要因がある。オラクルでも 2%〜3% の逸脱点が残り、データよりは多いがネットワークよりは少ないため、ソルバとネットワークの双方が寄与している。100 回のオイラーステップを実行すれば、学習されたネットワークは真のデータの逸脱点の割合（$\approx 1.3\%$）、広がり、およびモード重みにモンテカルロ誤差の範囲内で一致し、厳密なオラクルに対する `ode_sample` も陽的オイラー法のループと単精度浮動小数点の精度（10^{-4}）まで一致する。

17.3.3 祖先連鎖と逆 SDE

DDPM のサンプリング方法はこれとは異なる。Figure 17.4 の逆連鎖を、ノイズを伴うステップを 1 回ずつ踏みながら $t = 1$ から $t = 0$ まで実行するのである。そもそもノイズを含むステップの連鎖でなぜうまくサンプリングできるのか、そしてそれはフローとどのような関係にあるのだろうか。

17.3.3.1 祖先ステップ

小さな順方向ステップの逆過程はほぼガウス分布に従い (Sohl-Dickstein et al. 2015)、クリーンな点が与えられれば厳密にガウス分布となる。$s < t$ に対し、ベイズの定理により $q(\mathbf{x}_s \mid \mathbf{x}_t, \mathbf{x}_0)$ は $\mathbf{x}_s$ に関する 2 つのガウス分布、すなわち順方向ステップ $q(\mathbf{x}_t \mid \mathbf{x}_s)$ とノイズ付加 $q(\mathbf{x}_s \mid \mathbf{x}_0)$ の積に比例するため、$\mathbf{x}_t$ と $\mathbf{x}_0$ の線形結合を平均に持つガウス分布になる。Ho らのステップは、その平均において $\hat{\mathbf{x}}_0$ を $\mathbf{x}_0$ の代わりに用い、分散 $\beta = 1 - \alpha_t^2/\alpha_s^2$（$s$ から t への順方向ステップが加える分散）の新たなノイズを加える。これが**祖先**（ancestral）ステップと呼ばれるのは、グラフィカルモデルの祖先サンプリング（ancestral sampling）と同様に、各 $\mathbf{x}_s$ がその親 $\mathbf{x}_t$ のもとで抽出されるからである。VP 経路上において、$\hat{\mathbf{x}}_0 = (\mathbf{x}_t - \sigma_t \hat{\epsilon})/\alpha_t$ とスコア $\nabla \log p_t = -\hat{\epsilon}/\sigma_t$ を平均の式に代入して整理すると、ステップは次のように書ける。

$$\mathbf{x}_s = \frac{\mathbf{x}_t + \beta \nabla \log p_t(\mathbf{x}_t)}{\sqrt{1-\beta}} + \sqrt{\beta}\,\mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{17.22}$$

これは新しいサンプラなのだろうか、それとも先ほどのサンプラにノイズを加えたものなのだろうか。これを見るには、$1/\sqrt{1-\beta} \approx 1 + \beta/2$ と展開し、β に関する一次の項とノイズ項 $\sqrt{\beta}\,\mathbf{z}$

を残すとわかりやすい。するとステップは次の 2 つに分解される。

$$\mathbf{x}_s \approx \Big[\mathbf{x}_t + \tfrac{\beta}{2}\nabla \log p_t(\mathbf{x}_t) + \sqrt{\beta}\,\mathbf{z}\Big] + \tfrac{\beta}{2}(\mathbf{x}_t + \nabla \log p_t(\mathbf{x}_t)).$$

角括弧の部分は、ランジュバンステップ $\mathbf{x} \leftarrow \mathbf{x} + \frac{h}{2}\nabla \log p + \sqrt{h}\,\xi$（17.1 節）をステップサイズ $h = \beta$ で密度 p_t に適用したものであり、p_t を定常に保ちつつ逸脱した点をそこへ引き戻す。第 2 項は、時刻 t から s へ戻る確率フロー ODE の 1 オイラーステップである。VP 経路上での周辺速度は $\mathbf{u}_t = -\frac{1}{2}\beta(t)\,(\mathbf{x} + \nabla \log p_t)$ となるが、これは 17.2 節のスコア・速度の恒等式において $\dot{\alpha}_t/\alpha_t = -\beta(t)/2$ および $\alpha_t^2 + \sigma_t^2 = 1$ としたものにほかならない。ここで $\beta(t) = 0.1 + 19.9\,t$ は VP スケジュールのレートであり、微小ステップでは $\beta \approx \beta(t)\,(t - s)$ が成り立つ。したがって、t から s へ戻るオイラーステップは点を $-(t-s)\,\mathbf{u}_t = \frac{1}{2}\beta(t)\,(t-s)\,(\mathbf{x}_t + \nabla \log p_t) \approx \frac{\beta}{2}(\mathbf{x}_t + \nabla \log p_t)$ だけ動かし、これが第 2 項に一致する。すなわち、祖先ステップとは、フローのステップに現在のノイズレベルにおけるランジュバン補正を加えたものである。この連鎖は ODE と同じ輸送を行いつつ、ランジュバン項の働きによって点を p_t へと引き戻し、それ以前のステップで生じた誤差の一部を打ち消すことができる。実際にその効果が現れるかどうかは 17.5 節で測定し、そこでは Figure 17.9 において同じ初期点から出発する決定論的軌跡とノイズを含む軌跡を描画する。

17.3.3.2 逆 SDE

連続時間では、この時間反転は厳密になる。本節のコードではこの極限を必要としないが、論文ではサンプラがこの記法で書かれるため、記法を 1 つだけ導入しておく価値がある。**確率微分方程式**（stochastic differential equation、SDE）$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\,dt + g(t)\,d\mathbf{W}$ とは、$\Delta t \to 0$ の極限における次のステップの略記である。

$$\mathbf{x} \leftarrow \mathbf{x} + \mathbf{f}\,\Delta t + g\sqrt{\Delta t}\,\mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\,:$$

すなわち、決定論的な**ドリフト** $\mathbf{f}$ に、タイムステップに比例して分散が増大するガウスノイズを加えたものである。DDPM の順方向連鎖は $\Delta t = 1/T$、$\mathbf{f} = -\frac{1}{2}\beta(t)\,\mathbf{x}$、$g^2 = \beta(t)$ としたこの形のステップになっている（$\sqrt{1-\beta_k} \approx 1 - \beta_k/2$ かつ $\beta_k \approx \beta(t)\,\Delta t$ であるため）。Anderson (1982) は、このような過程を $t = 1$ から $t = 0$ へと逆向きにたどる過程が、ステップ $\mathbf{x} \leftarrow \mathbf{x} - [\mathbf{f} - g^2\nabla \log p_t(\mathbf{x})]\,\Delta t + g\sqrt{\Delta t}\,\mathbf{z}$ の $\Delta t \to 0$ における極限によって与えられることを示した。この極限において、逆過程は順過程と同じ周辺分布 p_t を持つ。論文では、この反転を時間が逆向きに進む次の SDE として記述する。

$$\text{with the same marginals as} \quad \begin{aligned} d\mathbf{x} &= \big[\mathbf{f} - g^2\nabla \log p_t(\mathbf{x})\big]\,dt + g\,d\bar{\mathbf{W}}, \\ \frac{d\mathbf{x}}{dt} &= \mathbf{f} - \tfrac{1}{2}g^2\nabla \log p_t(\mathbf{x}), \end{aligned} \tag{17.23}$$

ここで、$d\bar{\mathbf{W}}$ は各逆方向ステップで加わる新たなノイズを表す。なぜこれら 2 つの式は同じ周辺分布を共有するのだろうか。両者の逆方向ステップの差は $\frac{1}{2}g^2\nabla \log p_t\,\Delta t + g\sqrt{\Delta t}\,\mathbf{z}$ であり、これはステップサイズ $h = g^2\Delta t$ による p_t 上のランジュバンステップにほかならない。ランジュバンステップは p_t を不変に保つため、先ほどの祖先ステップの分解が連続時間でそのまま成り立っているのである。VP 経路の場合、右側の ODE は $\mathbf{u}_t = -\frac{1}{2}\beta(t)\,(\mathbf{x} + \nabla \log p_t)$ となり、その経路の確率フロー ODE Equation 17.21 と一致する。一方、逆 SDE のステップは

$\mathbf{x} + \frac{1}{2}\beta(t)\Delta t\,\mathbf{x} + \beta(t)\Delta t\,\nabla \log p_t + \sqrt{\beta(t)\Delta t}\,\mathbf{z}$ となり、$\beta = \beta(t)\,\Delta t$ とおいた祖先ステップの一次展開に一致する。すなわち、祖先連鎖は一次近似において、逆 SDE の最も単純な離散化であるオイラー・丸山法（Euler–Maruyama method、38.4.4.1 節）に対応している。ノイズを一切加えない ODE とこの SDE の間には、同じ周辺分布を持つ一連のサンプラ族が存在する。サンプラが注入するノイズが大きいほど、余分な広がりを引き戻すドリフト内のスコア項も強くなる（38.5.5 節、Figure 38.22）。

`ancestral_sample` は任意の経路と任意の予測器 `pred(x, alpha, sigma) -> (x0, eps)` に対し、一次近似ではなく厳密なステップを実行する。具体的には、$\mathbf{x}_s$ の条件付き平均（$\mathbf{x}_t$ と $\hat{\mathbf{x}}_0$ が与えられたもとでの平均）へと移動し、分散 $\sigma_t^2 - (\alpha_t/\alpha_s)^2\sigma_s^2$（$s$ から t への順方向ステップの分散であり、VP 経路上では β に等しい）のノイズを加える。

```
def ancestral_sample(pred, x, path, steps, key, *, t_start=1.0):
    """Ho et al.'s sampler from x at t_start (1 is noise) to data at t = 0."""
    a, s = path.grid(steps, t_start)
    a_t, s_t, a_s, s_s = a[:-1], s[:-1], a[1:], s[1:]
    ratio = s_s * a_t / (a_s * s_t)
    var = jnp.maximum(s_t ** 2 - (a_t / a_s * s_s) ** 2, 0)  # forward step
    noise = jnp.where(s_s > 0, jnp.sqrt(var), 0)             # none at t = 0
    def step(x, inputs):
        a_t, s_t, a_s, c_eps, c_noise, key = inputs
        x0, eps = pred(x, a_t, s_t)
        x = a_s * x0 + c_eps * eps                       # the posterior mean
        return x + c_noise * jax.random.normal(key, x.shape), None
    return jax.lax.scan(step, x, (a_t, s_t, a_s, s_s * ratio, noise,
                                  jax.random.split(key, steps)))[0]
```

フローと同じ 2,000 個の初期点から、ノイズモデルを用いてこの連鎖を 200 ステップ実行する。Ho らは画像に対して 1,000 ステップを用いたが、2 次元平面上では 200 ステップで十分である。

```
pred_eps = make_pred(ema_eps, vp, 'eps')
key, k_anc = jax.random.split(key)
print(mix.summary(ancestral_sample(pred_eps, x_T, vp, 200, k_anc)))
```

```
{'weights': [0.502, 0.291, 0.207], 'stray': 0.005, 'spread': 0.489}
```

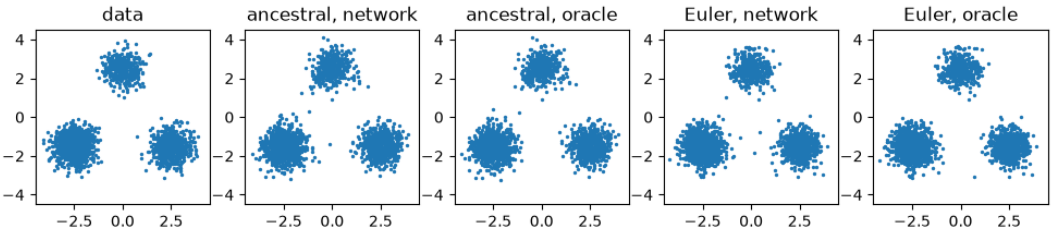
この連鎖は、2,000 点の標本誤差の範囲内で 3 つのモードを正しい重みで復元し、逸脱点の割合も 100 ステップのフローと同様に約 1% 以下に収まる。

17.3.4 両サンプラからのサンプル

ここまでは各サンプラを 2,000 点に対して 1 回ずつ実行しただけであり、ネットワークの誤差とサンプリングの偶然によるばらつきを見分けるには点数が少なすぎた。そこで、同一の条件のもとでネットワークとオラクルを比較する。20,000 個の初期点を用意し、ノイズモデルによる 200 回の祖先ステップと速度モデルによる 100 回のオイラーステップを実行したうえで、それぞれネットワークを `oracle` に置き換え、同じ初期点（連鎖については注入するノイズも同じ）から同じ計算を繰り返す。これにより、ネットワークの行と対応するオラクルの行との差はすべてネットワークに起因するものとなる。このサンプルサイズでは、モード重みの標準誤差は約 0.0035 である。さらに下のセルでは、ネットワークとオラクルが異なるモードへと送った初期点の個数も数える。

```
    key, k_T, k_data, k_noise = jax.random.split(key, 4)
    x_T = jax.random.normal(k_T, (20_000, 2))
    runs = {'data': mix.sample(k_data, 20_000)}
    for name, pred in (('network', pred_eps), ('oracle', oracle)):
        runs[f'ancestral, {name}'] = ancestral_sample(  # the same noise for both
            pred, x_T, vp, 200, k_noise)
    for name, pred in (('network', pred_v), ('oracle', oracle)):
        runs[f'Euler, {name}'] = ode_sample(pred, x_T, linear.sigma_bars(100))
    for name, x in runs.items():
        print(f'{name:>18}:', mix.summary(x))
    print('standard error of a weight:', [round(math.sqrt(w * (1 - w) / 20_000), 4)
                                          for w in mix.weights.tolist()])
    mode = lambda x: ((x[:, None] - mix.means) ** 2).sum(-1).argmin(1)
    for s in ('ancestral', 'Euler'):
        moved = mode(runs[s + ', network']) != mode(runs[s + ', oracle'])
        print(f'{s}: network and oracle end in different modes for '
              f'{float(moved.mean()):.2%} of the starts')
    d2l.scatter_panels([runs[k][:2000] for k in runs],
                       list(runs), lim=4.5, size=2.4);
```

```
              data: {'weights': [0.504, 0.299, 0.197], 'stray': 0.0111, 'spread': 0.478}
ancestral, network: {'weights': [0.504, 0.302, 0.194], 'stray': 0.012, 'spread': 0.485}
 ancestral, oracle: {'weights': [0.502, 0.303, 0.195], 'stray': 0.0114, 'spread': 0.485}
    Euler, network: {'weights': [0.498, 0.297, 0.205], 'stray': 0.0129, 'spread': 0.467}
     Euler, oracle: {'weights': [0.498, 0.299, 0.203], 'stray': 0.0106, 'spread': 0.472}
standard error of a weight: [0.0035, 0.0032, 0.0028]
ancestral: network and oracle end in different modes for 0.48% of the starts
Euler: network and oracle end in different modes for 0.31% of the starts
```



どちらのサンプラでも、訓練済みネットワークと厳密なオラクルが異なるモードへ送る初期点は 20,000 個のうちわずか 0.2%〜0.5% にすぎず、ネットワーク実行時の各モード重みも対応するオラクル実行時の値と 0.004 以内で一致する（二項分布の標準誤差 ≈ 0.0035 と同程度である）。また、両サンプラとも真のデータの広がり（誤差 0.01 以内）および逸脱点の割合（$\approx 1.1\%$、すなわち $3s$ を超えるガウス分布の裾）を再現している。

この混合分布において 100〜200 ステップを実行すれば、決定論的な ODE と確率的な祖先連鎖は実質的に同等の分布の精度を達成する。両者の違いはその構造的性質にある。確率フロー ODE はノイズ $\mathbf{x}_1$ とデータ $\mathbf{x}_0$ の間に決定論的で可逆な全単射を定めるため、潜在空間での補間や厳密な尤度評価が可能になる。一方、祖先連鎖はランジュバンノイズを注入することで、軌跡に沿った離散化誤差やスコア誤差を減衰させることができる（17.5 節）。Table 17.2 に、本章で扱う主要な生成モデルが経路・ターゲット・サンプラをどのように組み合わせているかをまとめる。

Table 17.2: 本章のモデル：ネットワークの訓練に用いるターゲットと経路、およびデータを生成するサンプラ。

モデル	ターゲットと経路	サンプラ	該当箇所
ノイズ除去スコアモデル	単一レベルでのスコア $-\epsilon/\sigma$	ランジュバン動力学	17.1 節
ノイズ条件付きスコアネットワーク	VE 経路上のラダーの各レベルにおけるスコア（重み $\tilde{\sigma}^2$）	アニーリング付きランジュバン動力学	17.2 節
DDPM	VP 経路上の ϵ	祖先連鎖または確率フロー ODE	17.2 節、本節
フローマッチング、Rectified Flow	線形経路上の $\epsilon - \mathbf{x}_0$	確率フロー ODE 上のオイラーステップ	17.2 節、本節
$\mathbf{v}$ 予測	コサイン経路上の $\mathbf{v}$	いずれも可	17.2 節、17.6 節
EDM	VE 経路上のクリーンな点（ノイズレベルごとに再スケーリング）	確率フロー ODE の 2 次ソルバ	17.5 節、17.6 節

17.4 節では 2 次元から 28×28 の画像へと進む。そこでは厳密なオラクルは利用できず、関数評価のたびに畳み込み U-Net の順伝播が必要になる。

17.3.5 まとめ

周辺速度 $\mathbf{u}_t(\mathbf{x}) = \mathbb{E}[\dot{\alpha}_t \mathbf{x}_0 + \dot{\sigma}_t \epsilon \mid \mathbf{x}_t = \mathbf{x}]$ は連続の式 $\partial_t p_t + \nabla \cdot (p_t \mathbf{u}_t) = 0$ を周辺密度 p_t とともに満たすため、確率フロー ODE $d\mathbf{x}/dt = \mathbf{u}_t(\mathbf{x})$ を $t = 1$ から $t = 0$ まで積分すれば、モード重みを保ったまま $\mathcal{N}(\mathbf{0}, \mathbf{I})$ が p_0 へと輸送される。$t = 1$ から $t = 0$ への 1 回のオイラーステップはすべての初期点をデータ平均 $\mathbb{E}[\mathbf{x}_0]$ へと崩壊させてしまうが、再スケーリングされた座標 $\bar{\mathbf{x}} = \mathbf{x}/\alpha_t$ と $\bar{\sigma} = \sigma_t/\alpha_t$ を用いれば、あらゆるガウス経路が共通の常微分方程式 $d\bar{\mathbf{x}}/d\bar{\sigma} = \hat{\epsilon}$ に帰着し、そのオイラーステップは $\mathbf{x}_s = \alpha_s \hat{\mathbf{x}}_0 + \sigma_s \hat{\epsilon}$ と書ける。ステップ分散 β の一次近似において、DDPM の確率的な祖先ステップはこの確率フローのオイラーステップと p_t 上のランジュバンステップの和へと分解され、これは Anderson の逆時間 SDE に対するオイラー・丸山法による離散化に対応している。

17.3.6 演習問題

1. **2 つの座標系における 1 ステップ。**
   1. 線形経路上において、ステップ $\mathbf{x}_s = \alpha_s \hat{\mathbf{x}}_0 + \sigma_s \hat{\epsilon}$ がオイラーステップ $\mathbf{x}_t - (t - s)\,\mathbf{u}_t(\mathbf{x}_t)$（$t$ に関するステップ）に等しいことを示せ。
   2. 任意のガウス経路上において、データが 1 点 $\mathbf{x}_0 = \mathbf{0}$ のみであるとする。$\hat{\epsilon} = \mathbf{x}/\sigma_t$ となること、$\mathbf{x}_t$ を通る厳密な軌跡が $\mathbf{x}_s = (\sigma_s/\sigma_t)\,\mathbf{x}_t$ であること、そして最初の小問のステップがすべての $s < t$ に対してこの厳密な軌跡を正確にたどることを示せ。
   3. VP 経路上では、このデータの周辺速度は $\mathbf{u}_t(\mathbf{x}) = (\dot{\sigma}_t/\sigma_t)\,\mathbf{x}$ であり、$d\sigma_t^2/dt = \beta(t)\,\alpha_t^2$ かつ $\beta(t) = 0.1 + 19.9\,t$ が成り立つ。$t = 0.5$（$\alpha_t \approx 0.28$、$\sigma_t \approx 0.96$）から $s = 0.1$（$\sigma_s \approx 0.32$）まで 1 ステップ進めよ。t に関するオイラーステップは $\mathbf{x}$ を何倍にするか。また本来は何倍にすべきか。この結果は、少数ステップで動作するサンプラが用いるべき座標系について何を示唆しているか。
2. **ガウスデータに対する祖先ステップの一次近似。**●データを $\mathcal{N}(\mathbf{0}, v_0\mathbf{I})$ とすると、VP 経路上では $p_t = \mathcal{N}(\mathbf{0}, v_t\mathbf{I})$（ただし $v_t = \alpha_t^2 v_0 + \sigma_t^2$）となる。
   1. Equation 17.22 の祖先ステップを t から s まで、厳密なスコアと $\beta = 1 - \alpha_t^2/\alpha_s^2$ を用いて 1 回行うと、p_t が $\mathcal{N}\big(\mathbf{0}, (v_s + \delta)\mathbf{I}\big)$（ただし $\delta = \beta^2(1/v_t - 1)/(1 - \beta)$）に

写されることを示せ。
   2. このステップが厳密になるのはどのようなデータのときか。また、混合分布の 1 成分（$v_0 = 0.25$）のように標準ガウス分布より幅の狭いデータに対しては、得られる分散は大きすぎるか、それとも小さすぎるか。
   3. グリッドを細分化してステップ数を 2 倍にし、各ステップの β を約半分にしたとき、連鎖全体で蓄積される誤差はどのように変化するか。
3. **ステップサイズとガウス反転。**●祖先連鎖はガウス遷移を用いるが、これはステップが小さい場合に正当化される。本問ではどれほど小さければよいかを調べる。$\sigma^2_{t|s} = \sigma_t^2 - (\alpha_t/\alpha_s)^2\sigma_s^2$ を、s から $t > s$ への順方向ステップの分散とする。
   1. 厳密な逆遷移が混合分布 $q(\mathbf{x}_s \mid \mathbf{x}_t) = \mathbb{E}_{\mathbf{x}_0|\mathbf{x}_t}\big[q(\mathbf{x}_s \mid \mathbf{x}_t, \mathbf{x}_0)\big]$ であり、その共分散が $\tau^2\mathbf{I} + c^2\,\mathrm{Cov}[\mathbf{x}_0 \mid \mathbf{x}_t]$ となることを示せ。ここで、$\tau^2 = \sigma^2_{t|s}\sigma_s^2/\sigma_t^2$ は $q(\mathbf{x}_s \mid \mathbf{x}_t, \mathbf{x}_0)$ の分散であり、$c = \alpha_s\sigma^2_{t|s}/\sigma_t^2$ はその平均における $\mathbf{x}_0$ の係数である。
   2. 1 次元においてデータが等確率で ± 1 をとるものとし、$x_t = 0$ とする。逆遷移が $\mathcal{N}(\pm c, \tau^2)$ の等重み混合分布となり、それが二峰性になるのは $\zeta^2 = c^2/\tau^2$ が 1 を超えるとき、かつそのときに限ること、そして $\zeta^2 = \alpha_s^2\sigma^2_{t|s}/(\sigma_t^2\sigma_s^2)$ となることを示せ。また、ζ はステップ幅 $t - s$ を細かくするときどのように縮小するか。
   3. VP 経路上において、ζ を $t = 0.5$ から $s = 0.495$、0.45、0.3 への各ステップ、および $t = 0.3$ から $s = 0$ への 1 ステップについて計算せよ。二峰性になるのはどの逆遷移か。最後のケースにおいて、最適な平均を持つガウス遷移は $x_t = 0$ で何を生成し、本来は何を生成すべきか。
4. [code] **コミットメント時刻。**●経路上のどの時点でサンプルは自身の属するモードを決定するのだろうか。Biroli et al. (2024) はこれをクローン作成によって測定している。すなわち、p_t の 1 点から出発し、独立なノイズを用いて逆連鎖のコピーを 2 本走らせ、両者が同じモードに行き着くかどうかを記録する。
   1. 2,000 個のクリーンな点を抽出し、VP 経路上で $\mathbf{x}_t$（$t = 0.9, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1$）へとノイズを加え、各点から `ancestral_sample(pred_eps, x_t, vp, steps, t_start=t)` のコピーを 2 本ずつ約 $200\,t$ ステップで実行せよ。終点の最近傍平均が一致するペアの割合 $\varphi(t)$ をプロットせよ。φ は $t \to 1$ のときどの値に近づくか、またそれはなぜか。
   2. Λ をデータの共分散行列の最大固有値とする。Biroli らは、第 1 主成分方向の信号がノイズを上回らなくなる時点 $\alpha_t^2\Lambda = 1$ によって種分化時刻を定義している。この混合分布に対してその時刻を計算し、得られた曲線およびモード重みを決定づけた 17.2 節のアニーリングラダーの粗いレベルと比較せよ。演習問題 2（38.7 節）および演習問題 4（38.5 節）では、1 次元の 2 モードに対して同じ問題を扱っている。
5. **同じ比率、同じフロー。**●2 つのガウス経路の係数が $\alpha_t^B = c_t\,\alpha_t^A$ および $\sigma_t^B = c_t\,\sigma_t^A$（$c_t$ は滑らかな正値関数）という関係にあり、すべての時刻で同じ対数信号対雑音比を持つとする。
   1. $p_t^B(\mathbf{y}) = c_t^{-d}\,p_t^A(\mathbf{y}/c_t)$ を示せ。さらに、17.2 節のスコア・速度の恒等式 $\mathbf{u}_t(\mathbf{x}) = (\dot\alpha_t/\alpha_t)\,\mathbf{x} - (\sigma_t\dot\sigma_t - \sigma_t^2\dot\alpha_t/\alpha_t)\nabla\log p_t(\mathbf{x})$ を用いて、$\mathbf{u}_t^B(\mathbf{y}) = (\dot c_t/c_t)\,\mathbf{y} + c_t\,\mathbf{u}_t^A(\mathbf{y}/c_t)$ となることを示せ。ここから、確率フロー ODE の軌跡が $\mathbf{y}_t = c_t\,\mathbf{x}_t$ で関係づけられることを結論づけよ。
   2. $\tau(t)$ を単調増加関数とする。経路 $p_{\tau(t)}$ が速度場 $\dot\tau(t)\,\mathbf{u}_{\tau(t)}$ によって生成され、$\mathbf{u}$ と同じ軌跡を持つことを示せ。
   3. 時間変換 $\tau(t)$ として、コサイン経路 $(\cos\frac{\pi\tau}{2}, \sin\frac{\pi\tau}{2})$ の時刻 $\tau(t)$ における対数信号対雑音比が線形経路の時刻 t における値と一致するものを求め、$c_t = 1/\sqrt{(1-t)^2 + t^2}$ となることを示せ。経路上のどこで再スケーリングが最大になるか。
   4. コサイン経路の確率フロー ODE は $\mathcal{N}(\mathbf{0}, \mathbf{I})$ に従うデータに対して何をするか、また

それはなぜか。同じデータに対して線形経路の確率フロー ODE は何をするか。

17.4 画像の拡散モデル

17.2 節および 17.3 節で扱った速度回帰、確率フロー ODE（probability-flow ODE）、祖先連鎖（ancestral chain）は、任意の次元 d にそのまま適用できる。平面（$d = 2$）から 28×28 の Fashion-MNIST 画像（$d = 784$）へスケールアップするにあたり、4 つの実践的な構成要素を導入する。

1. **アーキテクチャ：**多重解像度 U-Net (Ronneberger et al. 2015; Ho et al. 2020) を用いる。これは大域的な画像構造を捉える粗い 7×7 のボトルネックと、細かな空間構造を保つフル解像度のスキップ接続（skip connection）を組み合わせたものであり、すべての残差ブロックで時刻とクラスの埋め込みによる条件付けを行う。
2. **訓練レシピ：**モデルパラメータの指数移動平均（exponential moving average、EMA）と、訓練中のランダムな**ラベルドロップアウト**を導入し、単一のネットワークで条件付き速度場と無条件速度場の両方を学習させる。
3. **評価：**ドメイン分類器で抽出した実画像と生成画像の特徴量分布間のフレシェ距離（Fréchet distance、FD）(Heusel et al. 2017) を用い、有限標本における基準下限に照らして較正する。
4. **ガイダンス：**条件付き予測と無条件予測の差に沿って外挿し、サンプル多様性とクラスの典型性（typicality）をトレードオフする**分類器なしガイダンス**（classifier-free guidance、CFG）(Dhariwal and Nichol 2021; Ho and Salimans 2022) を導入する。

本節では、単一の GPU で訓練可能な 120 万パラメータの U-Net を用い、これら 4 つの構成要素すべてを Fashion-MNIST 上で実装・評価する。

17.4.1 ノイズ除去 U-Net

17.4.1.1 スキップ接続、解像度、時間埋め込み

17.2 節のネットワークである `TimeMLP` は、点と時刻を受け取り、同じサイズの点を返した。画像のネットワークもこの入出力の約束事を維持する。すなわち、ノイズの乗った画像 $\mathbf{x}_t$ と時刻 t を入力とし、同じ形状の予測（ここでは速度）を出力する。画像には畳み込みが適しているが、フル解像度のまま 3×3 畳み込みを積み重ねただけでは、各出力ピクセルは入力の小さな窓にしか依存しない。一方、高ノイズ領域では、ぼやけた塊が靴なのかバッグなのかを画像全体から判断しなければならない。

U-Net は複数の解像度で動作する。エンコーダは解像度を 28×28 から 14×14、さらに 7×7 へと 2 回半分にし、チャネル数を 64 から 128 へと 1 回倍増させることで、最下層の 3×3 畳み込みが画像の大部分を見渡せるようにする。続いてデコーダが 28×28 までアップサンプリングして戻す。ボトルネックだけでは袖の輪郭やスニーカーの縞模様といった細部が失われてしまうため、各解像度のデコーダは同じ解像度のエンコーダの特徴量も受け取り、自身の特徴量と連結する。これらの**スキップ接続**によって、ネットワークは U 字の形状をとる（Figure 17.7）。Ronneberger らは顕微鏡画像内の構造をセグメンテーションするためにこのアーキテクチャを設計した。セグメンテーションはピクセルごとに 1 つのラベルを求めるタスクであり、画像全体と各ピクセルの細部の両方に依存する入力と同じサイズの出力を必要とする。ノイズ除去器が求めるものもこれと同じである。

単一のネットワークがあらゆるノイズレベルでノイズ除去を担うが、最適な予測はレベルに応じて変化する。$t = 0$ 付近ではクリーンな画像はほぼ入力そのものであるのに対し、$t = 1$ では多数の画像の平均として推測するほかない。そのため、時刻はすべてのブロックに入力される。17.1 節の `d2l.time_embedding` は、単位時間あたり 1 から 64 ラジアンまで等比的に配置された 64 個の周

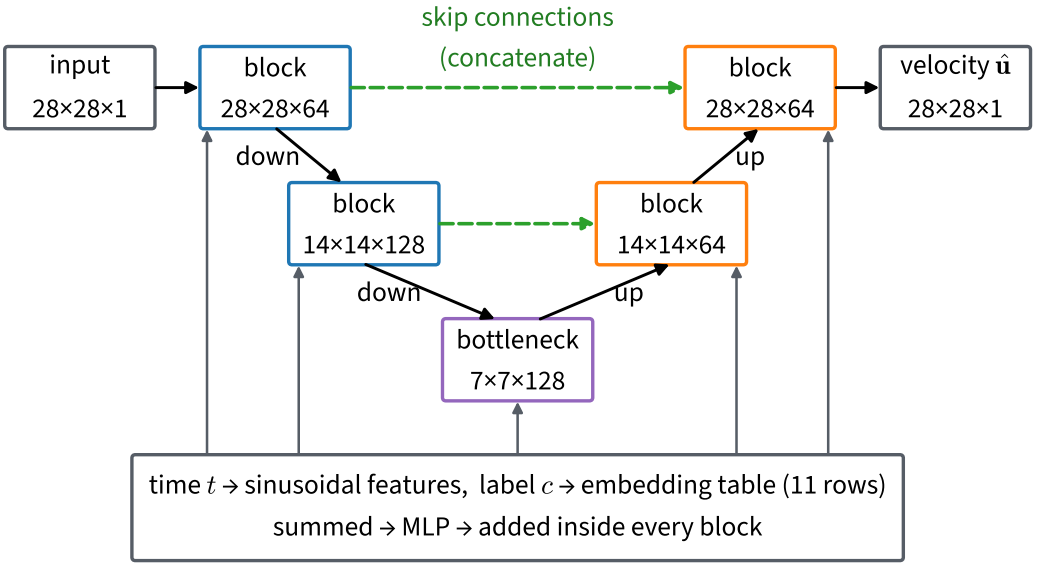


Figure 17.7: 本節の U-Net。2 段階のダウンサンプリング、7×7 のボトルネック、2 段階のアップサンプリングからなり、同じ解像度のエンコーダ特徴量をデコーダ特徴量に連結するスキップ接続を備える。時刻とクラスラベルは埋め込まれ、小さな MLP を経てすべてのブロック内部で加算される。

波数における正弦と余弦へ t を写像する。これは 10.4 節の位置エンコーディングと同じ構成であり、数千までの位置の代わりに 0 から 1 の時刻に合わせて周波数を選ぶことで、近接するノイズレベルにも明確に異なる特徴量を与える。2 層の MLP がこれらを 1 つのベクトルに変換し、各ブロックはそのベクトルの学習された射影をチャネルごとのバイアスとして特徴マップに加算する。

各ブロックはグループ正規化（group normalization）を伴う 2 回の 3×3 畳み込みを適用し、その間に条件付け埋め込みの射影を加算し、両者をまたぐ残差接続を設ける。グループ正規化は 1 枚の画像内のチャネルグループにわたって統計量を計算するため、バッチ正規化とは異なり、異なるノイズレベルにある画像間で統計量が混ざることがない。

`d2l.Conv` は `nnx.Conv` であり、重みとバイアスには Flax デフォルトの LeCun 正規分布の重みとゼロバイアスの代わりに、$U(-a, a)$（ただし $a = 1/\sqrt{\text{fan-in}}$）からのサンプルを用いる。これは全結合層に対する `d2l.Linear` と同じ初期化である。

```
class Conv(nnx.Conv):
    """nnx.Conv (kernel_size a tuple), all weights from U(+-1/sqrt(fan_in))."""
    def __init__(self, in_features, out_features, kernel_size, *, rngs,
                 **kwargs):
        a = (in_features * math.prod(kernel_size)) ** -0.5
        init = lambda key, shape, dtype=jnp.float32: jax.random.uniform(
            key, shape, dtype, -a, a)
        super().__init__(in_features, out_features, kernel_size,
                         kernel_init=init, bias_init=init, rngs=rngs,
                         **kwargs)
```

```
class UNetBlock(nnx.Module):
    """Two 3x3 convolutions with group norm, a residual, and an embedding."""
    def __init__(self, c_in, c_out, emb_dim=None, *, rngs):
        norm = lambda c: nnx.GroupNorm(c, num_groups=8, epsilon=1e-5,
                                       use_fast_variance=False, rngs=rngs)
        self.norm1, self.norm2 = norm(c_in), norm(c_out)
        self.conv1 = d2l.Conv(c_in, c_out, (3, 3), rngs=rngs)
        self.emb = d2l.Linear(emb_dim, c_out, rngs=rngs) if emb_dim else None
        self.conv2 = d2l.Conv(c_out, c_out, (3, 3), rngs=rngs)
        self.skip = (d2l.Conv(c_in, c_out, (1, 1), rngs=rngs)
                     if c_in != c_out else None)

    def __call__(self, x, emb=None):
        h = self.conv1(nnx.silu(self.norm1(x)))
        if self.emb is not None:
            h = h + self.emb(emb)[:, None, None, :]
        h = self.conv2(nnx.silu(self.norm2(h)))
        return h + (x if self.skip is None else self.skip(x))
```

この U-Net は、ストライド 2 の畳み込み 2 回によって 7×7 まで縮小し、最近傍アップサンプリングによって各スキップ接続のサイズまで拡大して戻す。

```
class UNet(nnx.Module):
    """A small U-Net on an image x (n, h, w, c), times t (n,) and labels y.

    y = num_classes is the null label. cond (n, emb_dim), if given, is added
    to the time features; the flow map of the few-step section passes the
    embedding of its step size there. time_freqs: see time_embedding."""
    def __init__(self, ch=64, num_classes=None, in_channels=1,
                 out_channels=1, emb_dim=128, time_freqs=(1.0, 64.0), *,
                 rngs):
        self.emb_dim, self.num_classes = emb_dim, num_classes
        self.time_freqs = time_freqs
        init = nnx.initializers.normal(1.0)  # N(0, 1), as in PyTorch
        self.label_emb = (nnx.Embed(num_classes + 1, emb_dim, rngs=rngs,
                                    embedding_init=init)
                          if num_classes else None)
        self.emb_mlp = d2l.make_mlp([emb_dim] * 3, nnx.silu, rngs=rngs)
        conv = lambda a, b, **kw: d2l.Conv(a, b, (3, 3), rngs=rngs, **kw)
        self.inc = conv(in_channels, ch)
        self.down1 = UNetBlock(ch, ch, emb_dim, rngs=rngs)        # 28 x 28
        self.pool1 = conv(ch, ch, strides=2, padding=1)           # 14 x 14
        self.down2 = UNetBlock(ch, 2 * ch, emb_dim, rngs=rngs)
        self.pool2 = conv(2 * ch, 2 * ch, strides=2, padding=1)   # 7 x 7
        self.mid = UNetBlock(2 * ch, 2 * ch, emb_dim, rngs=rngs)
        self.up1 = UNetBlock(4 * ch, ch, emb_dim, rngs=rngs)      # 14 x 14
        self.up0 = UNetBlock(2 * ch, ch, emb_dim, rngs=rngs)      # 28 x 28
        self.norm = nnx.GroupNorm(ch, num_groups=8, epsilon=1e-5,
                                  use_fast_variance=False, rngs=rngs)
        self.outc = conv(ch, out_channels)

    def __call__(self, x, t, y=None, cond=None):
```

```
        emb = d2l.time_embedding(t, self.emb_dim, self.time_freqs)
        if y is not None:
            emb = emb + self.label_emb(y)
        if cond is not None:
            emb = emb + cond
        emb = self.emb_mlp(emb)
        up = lambda h, skip: jnp.concatenate([jax.image.resize(
            h, h.shape[:1] + skip.shape[1:3] + h.shape[3:], 'nearest'),
            skip], -1)
        h1 = self.down1(self.inc(x), emb)
        h2 = self.down2(self.pool1(h1), emb)
        h = self.mid(self.pool2(h2), emb)
        h = self.up1(up(h, h2), emb)  # skip connection
        h = self.up0(up(h, h1), emb)  # skip connection
        return self.outc(nnx.silu(self.norm(h)))
```

このようなネットワークでは、パラメータはどこに保持され、計算はどこで費やされるのだろうか。3×3 畳み込みはどの解像度でも同じ数の重みを持つが、ピクセルごとに 1 回実行されるため、同じ層でも 28×28 では 7×7 の 16 倍のコストがかかる。フル解像度で 64 チャネル、10 クラスとしてネットワークを構築し、解像度ごとにパラメータ数と計算量の両方を数えてみよう。

```
net = UNet(ch=64, num_classes=10, rngs=nnx.Rngs(0))
levels = {28: ['inc', 'down1', 'up0', 'norm', 'outc'],
          14: ['pool1', 'down2', 'up1'], 7: ['pool2', 'mid']}
for side, names in levels.items():  # the blocks whose output has each size
    params = [p for n in names
              for p in jax.tree.leaves(nnx.state(getattr(net, n), nnx.Param))]
    kernels = sum(p.size for p in params if p.ndim == 4)  # convolutions
    print(f'{side} x {side}: {sum(p.size for p in params):,} parameters, '
          f'{kernels * side ** 2 / 1e6:.0f}M multiply-adds per image')
print(f'total: {d2l.num_params(net):,} parameters')
```

```
28 x 28: 211,329 parameters, 152M multiply-adds per image
14 x 14: 493,376 parameters, 92M multiply-adds per image
7 x 7: 459,776 parameters, 22M multiply-adds per image
total: 1,198,913 parameters
```

下位 2 つの解像度レベルがパラメータの 4/5 を保持する一方で演算量の 2/5 しか消費しないのに対し、フル解像度レベルはパラメータの 1/6 しか持たないにもかかわらず演算量の半分以上を占める。7×7 のボトルネックは重みの 38% を持ちながら、積和演算（multiply-add）は 8% にすぎない。低解像度では層を深くしても計算コストが安いため、U-Net は最も幅の広い層を最下部に置くのである。ネットワーク 1 回の評価には約 2.5 億回の積和演算を要する。この数値は、1 つのサンプルを生成するのに何回ネットワークを評価するかを数える際に念頭に置いておきたい。

これは意図的に簡素化した U-Net である。その 120 万個のパラメータは、Ho et al. (2020) の CIFAR-10 モデルが持つ 3570 万個の 1/30 にすぎない。同モデルは 4 つの解像度それぞれに 2 つの残差ブロックを備え、16×16 での自己アテンションとドロップアウトを追加している。さらに公開コードでは、初期化時にすべてのブロックの最後の畳み込みをゼロに設定し、各ブロックがスキップ経路そのものとして始まるようにしている。本節ではこれらを一切使わないが、それでも以下で生成するサンプルは衣類として十分に識別できる。2024 年以降、新しい大規模画像生成器の多くは U-Net を圧縮画像のパッチに対する Transformer へと置き換え、クラスではなくテキストプロンプトで条件付けを行っているが（17.7 節）、本節の残りの内容はそのまま通用す

る。

17.4.1.2 クラスによる条件付け

クラスラベルは追加の入力の 1 つであり、U-Net にはすでにそうした入力を受け取る場所、すなわちすべてのブロックが特徴量に加算する埋め込みが用意されている。クラスごとに 1 つの学習されたベクトルを持つ埋め込みテーブルからラベルのベクトルを取り出し、MLP に通す前に時間特徴量へ加算する。このテーブルには余分な行が 1 つあり、それは「クラス指定なし」を表す**ヌルラベル** $\emptyset$（ここではインデックス 10）である。

訓練中、各ラベルを確率 0.1 でヌルラベルに置き換える。この操作を**ラベルドロップアウト**と呼ぶ。これにより、1 つのネットワークが 2 つの予測を学習する。すなわち、$\mathbf{u}_\theta(\mathbf{x}_t, t, c)$（クラス c の画像の速度）と、$\mathbf{u}_\theta(\mathbf{x}_t, t, \emptyset)$（10 個に 1 個の訓練例で学習される Fashion-MNIST 全体の速度）である。なぜ与えられたラベルを無視するようネットワークに教えるのだろうか。本節の最後で扱うガイダンス（17.4.4.2 節）では、ラベルありの予測に対するベースラインとしてラベルなしの予測が必要になるが、ラベルドロップアウトを使えば同じネットワーク内でほぼ追加コストなしにそれを用意できるからである。Ho and Salimans (2022) は、ドロップアウト率 0.1 や 0.2 が 0.5 よりも良好に機能することを見出した。ベースラインが必要とするデータ量は、それが支える条件付き予測よりも少なくて済むのである。

サンプラは $(\mathbf{x}, t)$ のみのベクトル場を積分する。`conditional` はバッチのラベル c を固定し、ネットワークを場 $(\mathbf{x}, t) \mapsto \mathbf{u}_\theta(\mathbf{x}, t, c)$ へと変換する。

```
def conditional(net, c):
    """The field (x, t) -> net(x, t, c) with the labels c fixed."""
    return lambda x, t: net(x, t, c)
```

17.4.2 訓練とサンプリング

17.4.2.1 損失とコスト

損失は 17.2 節の損失において点を画像に置き換えたものである。クリーンな画像 $\mathbf{x}_0$（値域 $[-1, 1]$）とそのラベル、時刻 t（$[0, 1]$ 上の一様分布）、およびノイズ ϵ を抽出し、線形経路上の $\mathbf{x}_t = (1 - t)\,\mathbf{x}_0 + t\,\epsilon$ を構成する。ネットワークの出力は条件付き速度（conditional velocity）$\epsilon - \mathbf{x}_0$ へと回帰させ、二乗誤差をピクセル全体で平均する。唯一の追加要素はラベルドロップアウトであり、`batch_fn` が 128 枚の画像からなる各バッチに適用する。

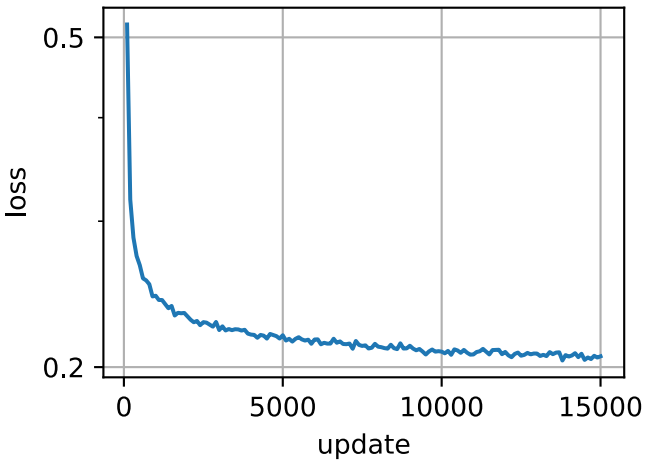
```
X, y, X_test, y_test = d2l.load_fashion_mnist_pm1()
path = d2l.GaussianPath.linear()

def batch_fn(key):
    k1, k2 = jax.random.split(key)
    i = jax.random.randint(k1, (128,), 0, len(X))
    drop = jax.random.uniform(k2, (128,)) < 0.1     # label dropout
    return X[i], jnp.where(drop, 10, y[i])          # 10 is the null label

def loss_fn(net, batch, key):
    x0, c = batch
    k1, k2 = jax.random.split(key)
    t, eps = jax.random.uniform(k1, (len(x0),)), jax.random.normal(k2, x0.shape)
    x_t = path.sample(x0, t, eps)
```

```
    return ((net(x_t, t, c) - path.target('velocity', x0, eps, t)) ** 2).mean()
```

17.1 節の d2l.train_regression はこの損失に対して Adam を実行し、訓練済みネットワーク、その重みの移動平均、および損失の履歴を返す。ここでは定数の学習率 10^{-3} で 15,000 回の更新を行う。なお Ho et al. (2020) は学習率 $2 \cdot 10^{-4}$ でその 50 倍以上にあたる 800,000 回の更新を行って訓練している。

本節において、訓練はむしろ計算コストの軽いパートである。訓練では延べ $15{,}000 \times 128 \approx 1.9$ 百万枚の画像に対してネットワークの順伝播と逆伝播を実行する。一方、以下で生成するサンプル（一部は 1 サンプルあたり 1,000 ステップを要する）には約 2000 万回のネットワーク評価が必要となり、1 回の逆伝播を順伝播 2 回分と数えても訓練の 3.5 倍の演算量に達する。拡散モデル（diffusion model）からサンプルを生成するコストは、生成方法をモデルに訓練するコストを上回り得るのであり、本章の残りの大部分はそのコストの削減を扱う。本節のコードは RTX 4090 上では 1 時間未満で完了するが、CPU 上では数時間を要する。GPU のない環境では、5,000 サンプルの代わりに 1,000 サンプルのスコアを計算し、後ほど算出する 1,000 枚時点の基準値と照らし合わせるとよい。

```
num_steps = 15_000
net, ema, losses = d2l.train_regression(net, loss_fn, batch_fn, num_steps,
                                        jax.random.key(0))
d2l.plot(jnp.arange(100, num_steps + 1, 100),
         losses.reshape(-1, 100).mean(1), 'update', 'loss', yscale='log')
```



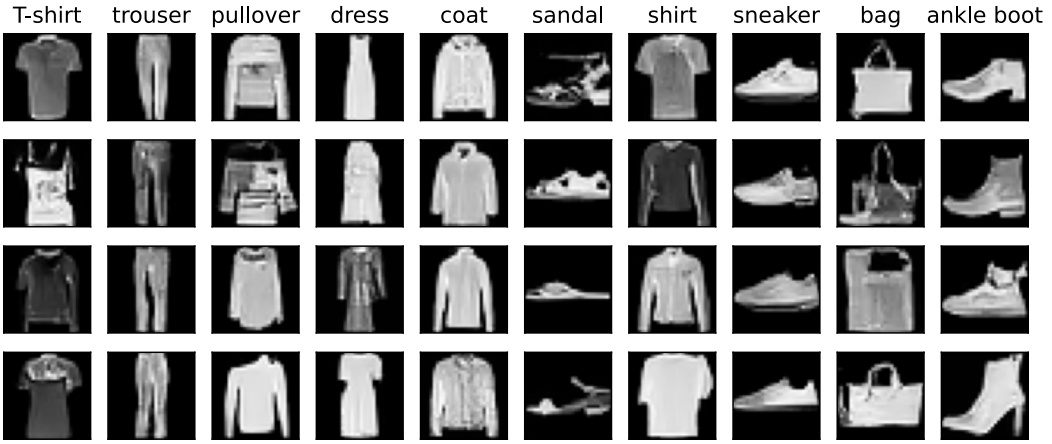
100 回の更新ごとに平均したピクセルあたりの損失は、最初の 1,000 回の更新のうちに約 0.5 から約 0.2 までほぼ下がりきる。損失がゼロよりかなり高い値で頭打ちになるのは、その下限が速度ターゲットの条件付き分散（conditional variance）だからである（17.2 節）。大きな t では、1 枚のノイズ画像と整合するクリーンな画像が多数存在する。

17.4.2.2 サンプルと予測

サンプリングにあたっては、17.3 節の d2l.make_pred でベクトル場をクリーンな画像とノイズの予測器へと変換し、d2l.ode_sample で確率フロー ODE を $t = 1$ から $t = 0$ まで（ここでは 50 回のオイラーステップで）積分する。generate はベクトル場を引数にとり、デフォルトは conditional だが、ガイダンスでは別の場を渡すことになる。この関数は n 枚の画像をチャンク単位で生成し、

i 番目の画像はクラス $i \bmod 10$ とする。初期ノイズはシード（特に断らない限り 0）から生成するため、同じシードを用いた 2 回の呼び出しは、両者の間で変更した条件のみが異なる。

```
euler50 = lambda pred, x, key: d2l.ode_sample(pred, x, path.sigma_bars(50))

@nnx.jit(static_argnums=(1, 5))  # Python functions: one trace for each
def sample_chunk(model, field, c, z, key, sampler):
    return sampler(d2l.make_pred(field(model, c), path, 'velocity'), z, key)

def generate(model, field=conditional, sampler=euler50, n=5000, chunk=500,
             seed=0):
    """n samples of field(model, c) from the noise of a seed; the i-th
    requests class i % 10."""
    kz, ks = jax.random.split(jax.random.key(seed))
    z, c = jax.random.normal(kz, (n, 28, 28, 1)), jnp.arange(n) % 10
    out = [sample_chunk(model, field, c[i:i + chunk], z[i:i + chunk],
                        jax.random.fold_in(ks, i), sampler)
           for i in range(0, n, chunk)]
    return jnp.concatenate(out).clip(-1, 1)

names = d2l.FashionMNIST().text_labels(range(10))
d2l.show_gray(generate(ema, n=40), 4, 10, titles=names + [''] * 30);
```



各列には 1 つのクラスのサンプルが 4 枚ずつ並んでいる。大半は指定したクラスだと認識でき、形状、柄、濃淡も多様である。一部には形の崩れたものもあり、また実画像と同様にシャツ、プルオーバー、コートは見分けにくい。

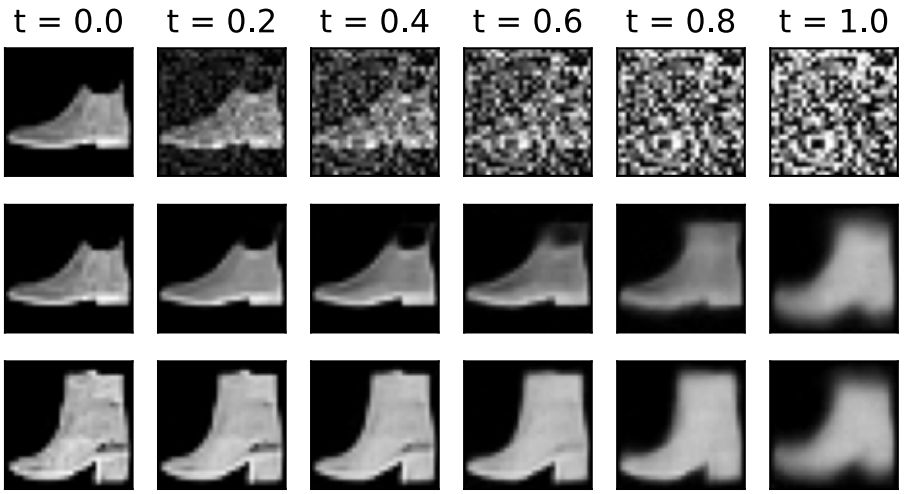
サンプラはあらゆるノイズレベルでネットワークに同じ問いを投げかける。このノイズ画像の背後にあるクリーンな画像はどれか、という問いである。各レベルでネットワークがどこまで答えられるかを見るため、以下の図の 1 行目では、アンクルブーツのテスト画像 1 枚に対し線形経路上の 6 つの時刻でノイズを加えている。2 行目は、ラベルを与えたときに各時刻でネットワークの予測が含意するクリーンな画像 $\hat{\mathbf{x}}_0 = \mathbf{x}_t - t\,\mathbf{u}_\theta(\mathbf{x}_t, t, c)$ を示す。2 行目の両端はあらかじめ予測できる。$t = 0$ では入力は元の画像そのものである。$t = 1$ では入力は純粋なノイズとなり、二乗誤差で訓練されたネットワークは残された情報であるラベルのもとでの事後平均（posterior mean）（17.1 節）、すなわちそのクラスの平均訓練画像を予測する。出力された最初の 2 行の数値は、各 $\hat{\mathbf{x}}_0$ と元の画像およびクラス平均画像とのピクセルあたりの二乗平均平方根（root mean

square、RMS）距離である。

3 行目はサンプラ自体の挙動を示す。新たなノイズから出発し、`euler50` の 50 回のオイラーステップでアンクルブーツへ向かって積分を進めながら、途中の $t = 1, 0.8, \ldots, 0.2$ においてネットワークが含意するクリーンな画像を記録している。$t = 0$ の列は完成したサンプルであり、出力の最後の行は完成したサンプルから各時点の予測までの距離を示す。

```
ts = jnp.linspace(0, 1, 6)
x0, c = jnp.repeat(X_test[:1], 6, 0), jnp.repeat(y_test[:1], 6)
eps = jax.random.normal(jax.random.key(1), x0[:1].shape)
x_t = path.sample(x0, ts, jnp.repeat(eps, 6, 0))
x0_hat = path.x0_eps('velocity', ema(x_t, ts, c), x_t, ts)[0]
rms = lambda a, b: jnp.sqrt(((a - b) ** 2).mean((1, 2, 3))).tolist()
for name, ref in (('image', x0), ('class mean', X[y == c[0]].mean(0))):
    print(f'RMS from the {name}:', ' '.join(f'{e:.2f}' for e in rms(x0_hat, ref)))
x = jax.random.normal(jax.random.key(2), x0[:1].shape)
grid, traj = jnp.linspace(1, 0, 51), []
for i in range(50):  # one sample of the class: 50 Euler steps from noise
    t = float(grid[i])
    u = ema(x, jnp.full((1,), t), c[:1])
    if i % 10 == 0:                          # t = 1.0, 0.8, ..., 0.2
        traj.append(x - t * u)               # its predicted clean image
    x = x + (float(grid[i + 1]) - t) * u
traj = jnp.concatenate([x] + traj[::-1])     # from t = 0 (the sample) to 1
print('RMS from the sample:', ' '.join(f'{e:.2f}' for e in rms(traj, x)))
d2l.show_gray(jnp.concatenate([x_t, x0_hat, traj]), 3, 6,
              titles=[f't = {t:.1f}' for t in ts.tolist()] + [''] * 12);
```

```
RMS from the image: 0.00 0.07 0.11 0.17 0.31 0.47
RMS from the class mean: 0.46 0.46 0.45 0.42 0.29 0.06
RMS from the sample: 0.00 0.04 0.08 0.14 0.23 0.42
```



両端は予測どおりの結果となる。$t = 1$ では、予測画像は平均的なアンクルブーツから 0.07 以内に位置し、元の画像からは 0.4 以上離れている。その中間では、ネットワークは私たち人間の目よりもよく見えている。$t = 0.6$ ではノイズの中にブーツを見つけるのは難しいが、それでも予

測画像はその輪郭をとどめており、平均的なブーツからの距離が 0.4 超であるのに対し、元のブーツからは 0.2 未満に収まっている。細部が最初に消え、輪郭は最後まで残るのである。

サンプラはノイズレベルを逆順にたどり、3 行目はその順序が何をもたらすかを示している。$t = 1$ での予測はぼやけた平均的なブーツであり、最終的に生成されるサンプルからは 0.3〜0.4 離れている。$t = 0.6$ までには最終サンプルから 0.12〜0.14 以内に入り、計算元の画像がまだノイズ 6 対ブーツ 4 の割合であるにもかかわらず、すでに最終的な輪郭を備えている。残りのステップでエッジを鮮鋭化し、陰影と質感を加えていく。つまりサンプルは早い段階で形状を確定させ、遅い段階で表面の細部を仕上げる。演習問題 3 ではこの性質を活用する。

17.4.2.3 移動平均

上のサンプリングは、最終更新時点の重みではなく、重みの移動平均である `ema` から行った。定数学習率のもとでは、Adam の反復点は収束して静止することがない。各更新がミニバッチのノイズを伴うため、反復点は良好な領域の周囲に雲のように漂い、直近の反復点の平均をとる方がその中心の近くに位置する (Polyak and Juditsky 1992)。16.5 節の `d2l.EMA` は指数移動平均を保持し、更新のたびに平均を現在の重みへ向かって少しだけ動かす。

$$\bar{\theta} \leftarrow \bar{\theta} + \left(1 - 2^{-1/n_{1/2}}\right)\left(\theta - \bar{\theta}\right), \tag{17.24}$$

これにより、$n_{1/2}$ 回前の更新における重みは現在の重みの半分の重みを持つようになる。この $n_{1/2}$ を**半減期**（half-life）と呼ぶ。ここでは $n_{1/2} = 1000$ を用い、各更新で差の 0.07% を縮める。論文やライブラリでは代わりに**減衰率**（decay）$2^{-1/n_{1/2}}$（ここでは 0.9993）を指定することが多いため、両者を換算できるようにしておくと便利である。

半減期は訓練の長さに合わせる必要がある。短すぎれば直近ステップのノイズが平均に残り、長すぎれば良好な領域からまだ遠い初期の重みまで引きずってしまう。Ho et al. (2020) は 800,000 回の更新からなる訓練において減衰率 0.9999、すなわち約 7,000 更新分の半減期を用いている。移動平均がどれほど有効であるかは、評価指標を導入した直後にまず測定する。

17.4.3 サンプル品質の測定

17.4.3.1 学習された特徴空間におけるフレシェ距離

40 枚のサンプルを並べたグリッドを見れば、サンプルがもっともらしいかどうかはわかる。しかし、5,000 枚のサンプルが適切な多様性を備えているか、バッグが多すぎてサンダルが少なすぎないか、あるいは見た目にはどちらも良好な 2 つのモデルのどちらが優れているかまでは判断できない。そのためには、サンプルの分布を実画像の分布と比較する数値指標が必要である。

このような比較を行う場として、ピクセル空間は不適切である。スニーカーを 2 ピクセル平行移動させただけで、描かれている内容は変わらないのにユークリッド距離では大きく離れてしまうからである。そこで、内容の似た画像同士が近くに配置される分類器の特徴量空間で画像を比較する。Heusel et al. (2017) の**フレシェ距離**（FD）は、実画像と生成画像の特徴量にそれぞれガウス分布 $\mathcal{N}(\mu_p, \Sigma_p)$ と $\mathcal{N}(\mu_q, \Sigma_q)$ を当てはめ、16.5 節で示した両者間の二乗 2 次ワッサースタイン距離を報告する。

$$\mathrm{FD} = \|\mu_p - \mu_q\|^2 + \mathrm{tr}\Big(\Sigma_p + \Sigma_q - 2\big(\Sigma_p^{1/2}\Sigma_q\Sigma_p^{1/2}\big)^{1/2}\Big). \tag{17.25}$$

第 1 項は、たとえば特定のクラスばかり過剰に生成されるなどして平均的なサンプルが平均的な実画像と異なるときに大きくなる。第 2 項は、たとえば生成されたコートがどれも似通っている

などして分布の広がりが異なるときに大きくなる。両方の項がゼロになるのは、2 つのガウス分布が一致する場合に限られる。

論文等で公表される FID と呼ばれるスコアは、ImageNet で訓練された Inception ネットワークの特徴量を用いる。ここでは 16.5 節と同様、代わりに独自の分類器を訓練する。具体的には、グレースケールのチャネルを 3 回繰り返した Fashion-MNIST 上で 15 エポック訓練した、128 次元の特徴量を持つ小規模な畳み込み分類器 `d2l.FeatureCNN`（オンライン付録「ユーティリティ関数とクラス」）を用いる。特徴量ネットワークは独自のものであるため、本節ではこのスコアを FD と呼び、既存の論文値とは決して比較しない。`fashion_feature_cnn` がこの分類器を訓練し、`fd_features` が 1,000 枚ずつ画像の特徴量を計算する。また、ネットワークのヘッドは各画像にクラスを割り当てる。

```
def fashion_feature_cnn(X, y):
    """The feature network that judges Fashion-MNIST samples in this chapter."""
    cnn = d2l.FeatureCNN(resolution=28, rngs=nnx.Rngs(0))
    d2l.train_feature_cnn(cnn, jnp.repeat(X, 3, -1), y)
    return cnn

@nnx.jit
def gray_features(cnn, x):
    """Features of gray images (n, 28, 28, 1), clipped to [-1, 1]."""
    return cnn.features(jnp.repeat(jnp.clip(x, -1, 1), 3, -1))

def fd_features(cnn, imgs):
    """Features of gray images (n, 28, 28, 1) in [-1, 1]."""
    return d2l.batched(lambda x: gray_features(cnn, x), imgs)

cnn = fashion_feature_cnn(X, y)
f_test = fd_features(cnn, X_test)
c_test = cnn.head(f_test).argmax(1)
print(f'test accuracy {(c_test == y_test).mean():.3f}')
```

```
test accuracy 0.926
```

この特徴量ネットワークはテスト画像の約 93% を正しく分類するため、その特徴量は各クラスを十分に分離している。`fd_and_agreement` は、基準となる特徴量に対するサンプル集合の FD とあわせて、指定したクラスに特徴量ネットワークが割り当てたサンプルの割合である**クラス一致率**（class agreement）を計算する。ここでは最初の 5,000 枚のテスト画像に対して 5,000 枚のサンプルのスコアを計算する。

```
def fd_and_agreement(cnn, imgs, f_ref):
    """FD to the reference features, and the fraction of samples that the
    feature network assigns to the requested class i % 10."""
    f = fd_features(cnn, imgs)
    agree = (cnn.head(f).argmax(1) == jnp.arange(len(imgs)) % 10).mean()
    return d2l.fid_score(f, f_ref), float(agree)

fd_ema, agree_ema = fd_and_agreement(cnn, generate(ema), f_test[:5000])
print(f'5,000 samples: FD {fd_ema:.2f}, class agreement {agree_ema:.3f}')
```

```
5,000 samples: FD 0.85, class agreement 0.892
```

サンプルの FD は約 0.8 であり、特徴量ネットワークはその 90% 近くを指定どおりのクラスに割

り当てている。では、0.8 という FD は良い値なのだろうか。この数値は特徴量ネットワークと画像の枚数に依存するため、単独では何も語らない。次項でその基準となる目盛りを与える。

17.4.3.2 フレシェ距離の基準値

実画像からなる 2 つの集合の間でも FD は正の値をとる。標本平均と標本共分散は偶然によって食い違い、Equation 17.25 のプラグイン推定量は（主にトレース項において）上向きの統計的バイアスを持つ。その大きさはおおよそ $1/n$（n は 1 集合あたりの画像枚数）に比例する。これがどれほどの大きさになるかを確かめるため、互いに素な 5 組の実訓練画像集合を、サンプルの基準集合である最初の n 枚のテスト画像に対して $n = 1{,}000$ と $n = 5{,}000$ でスコアリングする。この結果を、その枚数における FD の**基準値**（reference value）と呼ぶ。バイアスが $1/n$ に従うなら、5,000 枚での値は 1,000 枚での値の約 1/5 になるはずである。その横には Bińkowski et al. (2018) の Kernel Inception Distance（KID）も表示する。これは同じ特徴量上で 3 次カーネルを用いた二乗最大平均値差異（maximum mean discrepancy）（16.5 節）の不偏推定量であり、実画像の 2 集合に対する期待値は任意の n でゼロとなる。

```
f_real = fd_features(cnn, X[:25_000])  # five disjoint sets of 5,000
for n in (1000, 5000):
    sets = [f_real[i * n:(i + 1) * n] for i in range(5)]
    fd = np.array([float(d2l.fid_score(f, f_test[:n])) for f in sets])
    kid = np.array([float(d2l.kid_score(f, f_test[:n])) for f in sets])
    print(f'{n:,} real images, five sets: FD {fd.mean():.2f} '
          f'({fd.min():.2f} to {fd.max():.2f}), KID {kid.mean():.4f} '
          f'({kid.min():.4f} to {kid.max():.4f})')
```

```
1,000 real images, five sets: FD 0.87 (0.79 to 0.99), KID 0.0129 (-0.0036 to 0.0358)
5,000 real images, five sets: FD 0.16 (0.15 to 0.19), KID -0.0010 (-0.0042 to 0.0034)
```

予測は的中している。基準値は 1,000 枚で 0.9 弱、5,000 枚ではその 1/5 にあたる約 0.17 となる。KID にはこのようなバイアスはないが、ばらつきが大きい。1,000 枚では 5 つの値がゼロの両側に数百分の一の幅で散らばるため 1 つの値だけではほとんど判断できず、5,000 枚でも数千分の一の幅でばらつく。多くの論文が FD を報告しているため本章でもそれに倣い、数式による補正ではなく基準値との比較によってバイアスに対処する。

本章の残りの節では、必ず同じ画像枚数かつ同じ特徴量ネットワークのもとでのみ FD を比較する。5,000 枚の画像に対する 2 つの FD の差が、実画像 5 集合のばらつき幅である約 0.05 未満であれば両者は同等とみなす。1,000 枚ではそのばらつき幅は 0.15〜0.2 となる。以降の節では、完全な生成器が平均的に到達するスコアである**下限**（floor）として、各スコアの横にこの基準値を併記する。

生成サンプルの FD は約 0.8 であり、5,000 枚における基準値の約 5 倍高い。この差がモデルの誤差である。ところが、1,000 枚の実画像からなる 2 つの集合同士を比較しても、これと同程度のスコアが出る。画像枚数を伴わないフレシェ距離は無意味である。

基準となる目盛りが手に入ったところで、移動平均がどれほど効くのか、また 1,000 ステップのサンプラが 50 回のオイラーステップを上回るのかを確かめよう。1,000 ステップのサンプラは 17.3 節の祖先連鎖であり、Ho et al. (2020) の分散保持経路上で実行する。このネットワークは線形経路上で訓練されたものだが、問題はない。`make_pred` がネットワーク自身の経路上で同じ信号対雑音比（signal-to-noise ratio）を持つ時刻を見つけて再スケーリングするため、どの経路で訓練されたネットワークでも任意の経路に使えるからである。比較は対応のある形で行う。3 つのサンプラすべてを同じ初期ノイズから出発させることでノイズの引きによる差を排除し、これを 3 通りのノイズ抽出について繰り返す。

```
vp = d2l.GaussianPath.vp()
ancestral1000 = lambda pred, x, key: d2l.ancestral_sample(pred, x, vp, 1000,
                                                          key)
```

共通する 3 つの初期ノイズシードにわたり、EMA 重みと `euler50`、生の最終重み `net` と `euler50`、および EMA 重みと `ancestral1000` のそれぞれから 5,000 枚の画像を生成し、`f_test[:5000]` に対するフレシェ距離を比較する。

```
for seed in range(3):  # three draws of the initial noise
    fd = [fd_and_agreement(cnn, generate(model, sampler=sampler, seed=seed),
                           f_test[:5000])[0]
          for model, sampler in ((ema, euler50), (net, euler50),
                                 (ema, ancestral1000))]
    print(f'noise {seed}: Euler 50 average {fd[0]:.2f}, last weights '
          f'{fd[1]:.2f} ({fd[1] - fd[0]:+.2f}); chain with 1,000 steps '
          f'{fd[2]:.2f} ({fd[2] - fd[0]:+.2f})')
```

```
noise 0: Euler 50 average 0.85, last weights 1.35 (+0.50); chain with 1,000 steps 0.68 (-0.16)
noise 1: Euler 50 average 0.81, last weights 1.32 (+0.51); chain with 1,000 steps 0.62 (-0.20)
noise 2: Euler 50 average 0.84, last weights 1.25 (+0.41); chain with 1,000 steps 0.64 (-0.20)
```

このネットワークでは、いずれのノイズ抽出でも最終更新時の重みは移動平均より FD が 0.4〜0.6 悪化するのに対し、移動平均の FD はノイズ抽出間で数百分の一しか変動しない。どれだけ悪化するかは最後の更新が反復点の雲のどこにたまたま着地したかに左右され、別の訓練ランではまた別の場所に着地するだろう。直近の数千回の更新にわたる平均はそのような運にほとんど左右されないため、サンプリングには移動平均を用いるのである。また 1,000 ステップの連鎖は、やはりすべてのノイズ抽出で 50 回のオイラーステップより FD が 0.1〜0.2 低くなる。ただしこれは経路、サンプラ、ステップ数を同時に変更しているため、3 つの組み合わせの効果を測っているのであって個別の要因ではない。評価回数が 20 倍に増える割には控えめな改善であり、これ以降は 50 回のオイラーステップで評価を行う。

17.4.4 ガイダンス

17.4.4.1 スコア関数に対するベイズの定理

条件付きサンプルはすでに 90% 近くの割合で指定したクラスに属している。しかしそうでないものもあり、シャツ、プルオーバー、コートは見分けにくかった。本章の冒頭で触れたスライダが求めているのはそれ以上のもの、すなわち訓練済みネットワーク以上にラベルへ忠実に従うサンプラである。そのようなサンプラがどこから得られるかを知るため、ラベルがスコア関数にどのような影響を与えるかを見てみよう。時刻 t におけるベイズの定理は $p_t(\mathbf{x} \mid c) = p_t(\mathbf{x})\, p_t(c \mid \mathbf{x}) / p(c)$ と表される。ここで $p_t(c \mid \mathbf{x})$ は、ノイズの乗った画像 $\mathbf{x}$ が時刻 t においてクラス c のクリーンな画像から生じた確率であり、対数の勾配をとると $p(c)$ は消える。

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x} \mid c) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \log p_t(c \mid \mathbf{x}). \tag{17.26}$$

クラスのスコア関数は、データ全体のスコア関数に分類器の対数確率の勾配を足し合わせたものである。**分類器ガイダンス**（classifier guidance）(Dhariwal and Nichol 2021) は、第 2 項を別途用意した分類器 $p_\phi(c \mid \mathbf{x}, t)$ で与え、そこに**ガイダンス強度**（guidance scale）γ を掛ける。

$$\tilde{\mathbf{s}}(\mathbf{x}, t) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \gamma \nabla_{\mathbf{x}} \log p_\phi(c \mid \mathbf{x}, t). \tag{17.27}$$

$\gamma = 1$ のとき、これはベイズの定理そのものであるが、Dhariwal and Nichol (2021) は $\gamma > 1$ とすることでサンプルが指定クラスにはるかに良く一致することを見出した。サンプラは勾配を $\mathbf{x}_t$ で評価するが、クリーンな画像用の分類器はそこでは意味のある勾配を返さないため、分類器はノイズ画像上で訓練されていなければならない。付録では 1 次元混合分布上でこの規則を扱い（38.6.4 節）、演習問題 2 では本節の画像に対してこれを実行する。

17.4.4.2 分類器なしガイダンス

分類器ガイダンスにはノイズ画像上で訓練された 2 つ目のネットワークが必要であり、そのネットワーク固有の誤差も入り込む。そこで Ho and Salimans (2022) は、代わりに Equation 17.26 を右辺から左辺へと読んだ。

$$\nabla_{\mathbf{x}} \log p_t(c \mid \mathbf{x}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}). \tag{17.28}$$

分類器の勾配は条件付きスコア関数と無条件スコア関数の差にほかならず、ラベルドロップアウトを用いて訓練したネットワークはその両方を与える。この 2 つのスコア関数 $\mathbf{s}_c$ と $\mathbf{s}_\emptyset$ を Equation 17.27 に代入すると、ガイド付きスコア関数 $\tilde{\mathbf{s}} = \mathbf{s}_\emptyset + \gamma(\mathbf{s}_c - \mathbf{s}_\emptyset)$ が得られる。すなわち、分類器をネットワーク内部に暗黙的に組み込んだ分類器ガイダンスである。

本節のネットワークが予測するのはスコア関数ではなく速度であるが、固定した点 $(\mathbf{x}, t)$ において速度はスコア関数のアフィン関数である。線形経路上では、17.2 節の変換辞書から次式が成り立つ。

$$\mathbf{u}_t(\mathbf{x}) = -\frac{\mathbf{x} + t\,\nabla_{\mathbf{x}} \log p_t(\mathbf{x})}{1 - t}. \tag{17.29}$$

項 $-\mathbf{x}/(1-t)$ はラベルに依存せず、重み $1-\gamma$ と γ の和は 1 であるため、この項は線形結合をそのまま通過する。したがって、ガイド付き速度も 2 つの速度の同じ結合で表される。

$$\tilde{\mathbf{u}}(\mathbf{x}, t) = \mathbf{u}_\theta(\mathbf{x}, t, \emptyset) + \gamma\big(\mathbf{u}_\theta(\mathbf{x}, t, c) - \mathbf{u}_\theta(\mathbf{x}, t, \emptyset)\big). \tag{17.30}$$

`cfg` はこの規則を計算する。画像をラベル付きで 1 回、ヌルラベル付きで 1 回並べた 2 倍サイズのバッチ 1 つに対してネットワークを評価するため、ガイド付きサンプルは 1 ステップあたり 2 回の評価コストを要する。$\gamma = 1$ ではこの規則は条件付き速度場に帰着するため、`cfg` はバッチの後半を作らずに `conditional(net, c)` をそのまま返す。

```
def cfg(net, c, gamma):
    """Classifier-free guidance: (x, t) -> u + gamma (u_c - u), u unlabeled.

    gamma is a Python number; at gamma = 1 this is conditional(net, c)."""
    if gamma == 1:
        return d2l.conditional(net, c)
    null = jnp.full_like(c, net.num_classes)
    def guided(x, t):  # one doubled batch: labeled, then unlabeled
        out_c, out = jnp.split(net(jnp.concatenate([x, x]),
                                   jnp.concatenate([t, t]),
                                   jnp.concatenate([c, null])), 2)
        return out + gamma * (out_c - out)
    return guided
```

このアフィン性に基づく議論は、変換辞書のすべてのターゲットに対して成り立つ（演習問題 1）。なお Ho and Salimans (2022) はこの規則を $(1+w)\,\mathbf{u}_c - w\,\mathbf{u}_\emptyset$ と表記しているため、同論文の w は本節の $\gamma - 1$ にあたる。Figure 17.8 に本章を通じた例題上でのこの規則を示す。$\gamma > 1$ のとき、ガイド付き予測は無条件予測と条件付き予測を結ぶ直線上で、条件付き予測をさらに越えた先に位置する。

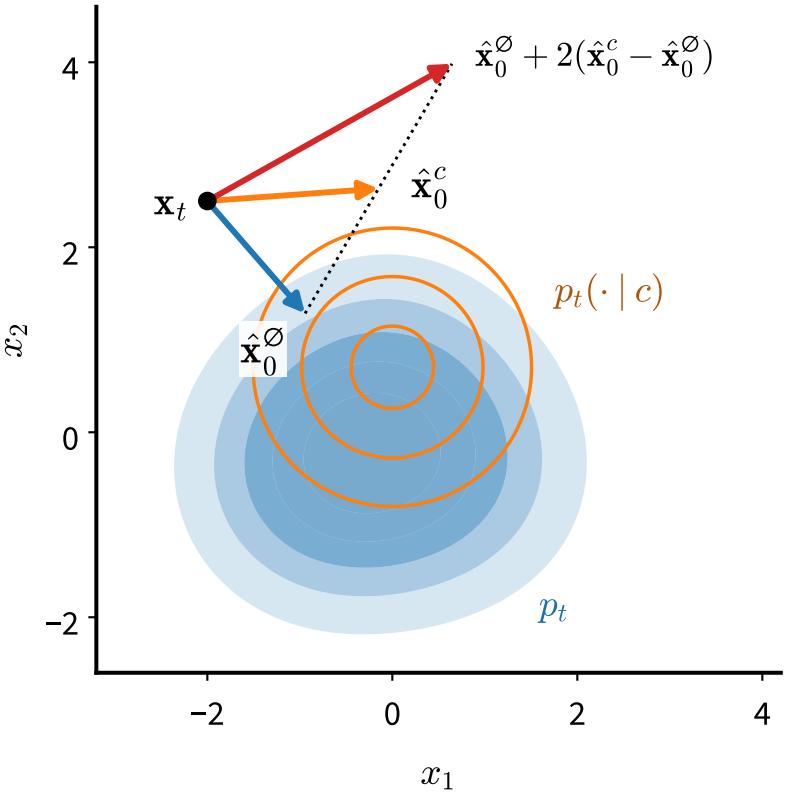


Figure 17.8: ある 1 つのノイズレベルにおける 3 モード混合分布上の分類器なしガイダンス。ノイズの乗った点 $\mathbf{x}_t$ から、データ全体の事後平均 $\hat{\mathbf{x}}_0^{\emptyset}$、上部のモードが与えられたときの事後平均 $\hat{\mathbf{x}}_0^{c}$、および $\hat{\mathbf{x}}_0^{\emptyset} + \gamma(\hat{\mathbf{x}}_0^{c} - \hat{\mathbf{x}}_0^{\emptyset})$（$\gamma = 2$ でのガイド付き予測であり、$\hat{\mathbf{x}}_0^{c}$ のさらに先に位置する）へと矢印が伸びている。等高線は全体のノイズ付き密度と上部モードのノイズ付き密度を表す。

γ がサンプルに与える影響を確かめるため、同じ初期ノイズから 7 通りの強度それぞれで 5,000 枚のサンプルを生成する。FD は実画像の分布をどれだけ満遍なく覆えているかを測り、クラス一致率はどれだけラベルに従っているかを測る。

```
guided = lambda gamma: lambda net, c: cfg(net, c, gamma)
samples = {gamma: generate(ema, guided(gamma))
           for gamma in (0.0, 1.0, 1.25, 1.5, 2.0, 3.0, 4.0)}
for gamma, imgs in samples.items():
    fd, agree = fd_and_agreement(cnn, imgs, f_test[:5000])
    print(f'gamma {gamma:.2f}: FD {fd:5.2f}, class agreement {agree:.3f}')
```

```
gamma 0.00: FD  2.80, class agreement 0.105
```

```
gamma 1.00: FD  0.85, class agreement 0.892
gamma 1.25: FD  0.84, class agreement 0.940
gamma 1.50: FD  1.14, class agreement 0.962
gamma 2.00: FD  2.05, class agreement 0.983
gamma 3.00: FD  3.97, class agreement 0.996
gamma 4.00: FD  5.64, class agreement 0.999
```

クラス条件付けを行わない場合（$\gamma = 0$）、クラス一致率はランダム予測の水準である 10% となり、FD は 2.7〜2.8 まで上昇する。これは $\gamma = 1$ における条件付き FD（≈ 0.8、クラス一致率は 90% 近く）の 3 倍を超える。$\gamma = 0$ での FD 上昇のうち、クラス比率の偏りに由来する分と各クラス内の画質劣化に由来する分を切り分けるため、次のコードセルでは Equation 17.25 の平均項 $\|\mu_p - \mu_q\|^2$ を比較し、生成画像のクラスヒストグラムを調べたうえで、5,000 枚の実訓練画像を $\gamma = 0$ のクラス比率に合わせて再サンプリングしたときの FD を計算する。

```
for gamma in (0.0, 1.0):
    f = fd_features(cnn, samples[gamma])
    mean_term = ((f.mean(0) - f_test[:5000].mean(0)) ** 2).sum()
    print(f'gamma {gamma:.1f}: mean term {mean_term:.2f} of FD '
          f'{d2l.fid_score(f, f_test[:5000]):.2f}')
share = lambda cls: jnp.bincount(cls, length=10) / len(cls)
cls = cnn.head(fd_features(cnn, samples[0.0])).argmax(1)
for name, s_test, s_0 in zip(names, share(c_test), share(cls)):
    print(f'{name:>11}: test images {s_test:.3f}, gamma 0 {s_0:.3f}')
# real training images, 5,000 in the class mix of the gamma = 0 samples
counts, y_real = jnp.bincount(cls, length=10).tolist(), np.asarray(y[:25_000])
mixed = np.concatenate([np.flatnonzero(y_real == k)[:n]
                        for k, n in enumerate(counts)])
for name, f in (('uniform', f_real[:5000]), ('gamma 0', f_real[mixed])):
    mean_term = ((f.mean(0) - f_test[:5000].mean(0)) ** 2).sum()
    print(f'real images, {name} class mix: mean term {mean_term:.2f} of FD '
          f'{d2l.fid_score(f, f_test[:5000]):.2f}')
```

```
gamma 0.0: mean term 0.89 of FD 2.80
gamma 1.0: mean term 0.33 of FD 0.85
    T-shirt: test images 0.101, gamma 0 0.099
    trouser: test images 0.099, gamma 0 0.089
   pullover: test images 0.099, gamma 0 0.133
      dress: test images 0.099, gamma 0 0.096
       coat: test images 0.102, gamma 0 0.113
     sandal: test images 0.099, gamma 0 0.062
      shirt: test images 0.097, gamma 0 0.114
    sneaker: test images 0.103, gamma 0 0.090
        bag: test images 0.101, gamma 0 0.101
 ankle boot: test images 0.098, gamma 0 0.104
real images, uniform class mix: mean term 0.04 of FD 0.19
real images, gamma 0 class mix: mean term 0.23 of FD 0.51
```

無条件サンプラはプルオーバーを過剰に生成し（13%〜14%）、サンダルを過少にしか生成しないが（6%〜7%）、これと同じクラス比率の偏りを実画像に与えても FD は 0.2〜0.3（0.19 から 0.4〜0.5 へ）しか上がらず、FD の差 1.9（$\gamma = 1$ と $\gamma = 0$ の間）の 1/6 未満しか説明しない。残りの差が生じるのは、無条件の速度場 $\mathbf{u}_\theta(\mathbf{x}_t, t, \emptyset)$ が全 10 クラスを同時に扱いながらミニバッチのわずか 10% でしか訓練されておらず、形の崩れた衣類や曖昧な衣類をより多く生み出すためである。Equation 17.30 は $\mathbf{u}_c + (\gamma - 1)(\mathbf{u}_c - \mathbf{u}_\emptyset)$ と書き換えられるため、無条件ベースライン

$\mathbf{u}_\emptyset$ の誤差は重み $1-\gamma$ を伴ってガイド付きサンプルに入り込む。

γ を 1 より大きくすると、多様性と典型性のトレードオフが現れる。穏やかなガイダンス（$\gamma = 1.25$）では、実行ごとのノイズの範囲を超えて FD を悪化させることなく、クラス一致率が約 5 パーセントポイント上昇する（93%〜94%）。$\gamma \geq 1.5$ では、クラス一致率は 96% を超えて上昇する一方（99% を $\gamma = 3$ および 4 で上回り、実テスト画像に対する分類器の正解率よりも高くなる）、クラス内の多様性が失われて高コントラストな典型的プロトタイプへと崩壊していくにつれ、FD は 5.5〜6 まで着実に上昇する。

```
for gamma in (0.0, 4.0):
    d2l.show_gray(generate(ema, guided(gamma), n=40), 4, 10,
                  titles=names + [''] * 30)
```

実際のテキストから画像を生成するシステムでは強いガイダンス強度が使われる。人間のユーザーは 5,000 サンプルからなるバッチの共分散を評価するのではなく、個々の画像をプロンプトへの忠実さと視覚的な明瞭さで評価するからである。

17.4.4.3 ガイド付きサンプラが従う分布

サンプラは $\gamma = 1$ では指定クラスから、$\gamma = 0$ では全画像からサンプリングを行うが、それ以外の強度では Equation 17.30 の規則はいかなる分布から導出されたものでもない。それでも、対応する分布を探してみよう。ガイド付きスコア関数 $\nabla \log p_t(\mathbf{x}) + \gamma \nabla \log p_t(c \mid \mathbf{x})$ は、次の**傾斜**（tilted）密度の対数の勾配になっている。

$$\tilde{p}_t(\mathbf{x}) \propto p_t(\mathbf{x})\, p_t(c \mid \mathbf{x})^{\gamma} \propto p_t(\mathbf{x})^{1-\gamma}\, p_t(\mathbf{x} \mid c)^{\gamma}. \tag{17.31}$$

$\gamma > 1$ のとき、分類器の確率がべき乗されることで、分類器が最も確信を持つ画像へと確率質量が移動する。同値な見方として、無条件密度の一部で割ることにより、そのクラスが他クラスと共有する領域からサンプルが押し出されるとも言える。

この効果はどれほど大きいのだろうか。最小の例を考えれば閉じた形で答えが出る。クラス c の画像が $\mathcal{N}(\mu_c, s^2\mathbf{I})$ に従い、全画像を合わせた分布をより広がりの大きい 1 つのガウス分布 $\mathcal{N}(\mathbf{0}, S^2\mathbf{I})$（$S^2 > s^2$）で戯画化して近似するとしよう。10 クラスの混合分布はガウス分布ではないため、これは本節で最も大胆な仮定であるが、すべての答えを厳密に求められる仮定でもある。ガウス分布の対数は二次関数であるため、クリーンな画像の傾斜対数密度（定数差を除いて $\gamma \log p(\mathbf{x} \mid c) + (1-\gamma)\log p(\mathbf{x})$）もやはりガウス分布の対数となる。二次の項と一次の項をまとめると、逆分散 $1/v$ と平均は次式で与えられる。

$$\frac{1}{v} = \frac{\gamma}{s^2} - \frac{\gamma - 1}{S^2}, \qquad \text{mean} = \frac{\gamma v}{s^2}\,\mu_c. \tag{17.32}$$

クラスの逆分散が γ 回分足され、全画像の逆分散が $\gamma - 1$ 回分引かれる。$\gamma > 1$ のとき、この結果は $1/s^2$ を $(\gamma-1)(1/s^2 - 1/S^2) > 0$ だけ上回り、μ_c の前に掛かる係数も 1 を超える。すなわち、ガイダンスはクラスの広がりを狭めるとともに、全画像の平均から遠ざける方向へ押し出す。1 次元において $\mu_c = 1$、$s = 0.5$、$S^2 = 2$ とすると、強度 $\gamma = 4$ ではクラスの標準偏差が 0.5 から 0.26 へと縮小し、平均は 1 から 1.10 へと移動する。これが多様性と典型性のトレードオフを数式で表した姿である。

では、ガイド付きサンプラは少なくともこの狭まったガウス分布に到達するのだろうか。答えは否であり、その差は大きい。1 次元ではこの例のガイド付き速度は x に関してすべての t でアフィン関数となるため、確率フロー ODE はガウス分布をガウス分布へと運び、その平均と分散を厳密に積分できる。すると、サンプラは傾斜分布そのものすらオーバーシュートすることがわかる。$\gamma = 4$ で純粋なノイズから出発すると、最終的な平均は 1.23、標準偏差は 0.02 となり、クラスの幅は 20 倍以上縮小して傾斜ガウス分布の幅の 1/10 未満にまで潰れてしまう。

$t = 0$ における傾斜分布はサンプラが実際に従う分布ではない。なぜなら、$t > 0$ における傾斜密度はその傾斜分布にノイズを加えたものではなく、他のいかなるクリーンな分布にノイズを加えたものでもないからである。分散 σ_t^2 のガウスノイズを加えて得られる密度は、ノイズ自体より鋭い曲率を持つことができない。すなわち、$-\nabla^2 \log p_t$ のいかなる固有値も $1/\sigma_t^2$ を超えない（証明は 38.6.1.2 節にある）。線形経路上では $\sigma_t = t$ であり、2 つのガウス分布は分散 $(1-t)^2 s^2 + t^2$ と $(1-t)^2 S^2 + t^2$ を持つガウス分布のままである。$t = 0$ のときと同じ計算を行うと、$t = 1$ 付近において $\gamma > S^2/(S^2 - s^2)$（この例では $8/7$）である限り必ず傾斜密度がこの上界を破ることがわかる。$\gamma = 1.5$ では 0.47 より大きいすべての t で、$\gamma = 4$ では 0.24 より大きいすべての t で上界を破る（演習問題 6）。経路の大部分にわたり、ガイド付きサンプラはいかなるノイズ付き分布も持ち得ないほど強く引き寄せるベクトル場に従っており、その引き寄せがステップごとに累積していくのである。付録では 1 次元の 2 クラス混合分布でも同じ現象が生じることを確認している（38.6.4 節）。

問題の原因は高ノイズ領域にあるため、Kynkäänniemi et al. (2024) は高ノイズ領域でガイダンスを切り、中間のノイズレベル範囲でのみガイダンスを適用することで、多様性の損失を抑えつつクラス一致率の向上の大半を維持している。演習問題 3 では本節のモデルでこれを試す。一方、Karras, Aittala, Kynkäänniemi, et al. (2024) は基準側を変更する手法を提案している。すなわち、無条件予測の代わりに、より小規模または訓練の浅い条件付きモデルを用いる。その誤差は誘導対象のモデルの誤差と似ているため、そこから遠ざかる方向に外挿することで多様性ではなく誤差を取り除くことができる。

これらはいずれも計算コストが高い。上記のガイド付きサンプルはそれぞれ 2 倍サイズのバッチによる 50 ステップ、すなわち 100 回のネットワーク評価を要しており、また 1,000 ステップの連鎖も 50 ステップをわずかに上回る程度であった。果たして何回まで評価回数を減らせば十分なのか、それを問うのが 17.5 節である。

17.4.5 まとめ

ノイズ除去 U-Net は、パラメータの大半を保持しつつわずかな演算量で大域的文脈を捉える低解像度のボトルネック層と、鋭い空間的エッジを保持するフル解像度のスキップ接続を組み合わせ、すべての残差ブロックに時刻とクラスの埋め込みを注入する。重みの指数移動平均（EMA）からサンプリングすることで、Adam の最終反復点を用いる場合と比べてフレシェ距離が大幅に低下する。ただし、フレシェ距離自体には正の $O(1/n)$ の有限標本による統計的バイアスがあるため、常に同じサイズ n の互いに素な実データ集合間の基準下限と照らし合わせて読む必要がある。最後に、ランダムなラベルドロップアウトによって単一のネットワーク内で条件付き速度場と無条件速度場を訓練することで、分類器なしガイダンス $\tilde{\mathbf{u}} = \mathbf{u}_\emptyset + \gamma(\mathbf{u}_c - \mathbf{u}_\emptyset)$ による生成の誘導が可能になる。穏やかなガイダンス強度（$\gamma \approx 1.25$）は FD を損なわずにクラス一致率を高める一方、より大きな強度（$\gamma > 1.5$）では高ノイズ領域の傾斜スコア関数がいかなるガウス畳み込み密度よりも鋭く分散を収縮させるため、サンプル多様性と引き換えに典型性が強まる。

17.4.6 演習問題

1. **同値なガイダンス規則**。ネットワークの 2 つの速度が厳密であり、したがって 17.2 節の変換辞書によりスコア関数 $\nabla \log p_t(\mathbf{x} \mid c)$ と $\nabla \log p_t(\mathbf{x})$ のアフィン関数になっているとし、さらに分類器もノイズ画像の厳密なクラス事後分布 $p_t(c \mid \mathbf{x})$ であるとする。
   1. 分類器ガイダンス Equation 17.27（強度 γ）と分類器なしガイダンス Equation 17.30（同じ γ）が同一のガイド付き速度を生成することを示せ。
   2. 変換辞書のすべてのターゲット（ノイズ、クリーン画像、スコア関数、$\mathbf{v}$、および速度）は、固定した $(\mathbf{x}, t)$ において $a(\mathbf{x}, t) + b(t)\,\hat{\mathbf{x}}_0$ という事後平均 $\hat{\mathbf{x}}_0$ のアフィン関数で表される。いずれか 1 つのターゲットにガイダンス規則を適用してから変換した結果が、他の任意のターゲットに適用した結果と一致することを示せ。この議論は重み $1-\gamma$ と γ のどの性質を用いているか。
   3. ネットワークが予測した $\hat{\mathbf{x}}_0$ を $[-1, 1]$ にクリップする場合でも、この規則はすべてのターゲットに対して同じ結果を与えるか。
2. [code] **クリーン画像分類器によるガイダンス**。●分類器ガイダンスにはノイズ画像の分類器が必要であるが、ネットワークによるクリーン画像の推定値に対してクリーン画像の分類器を適用することでも生成を誘導できる。
   1. 特徴量ネットワーク `cnn` のテスト画像に対する正解率を、線形経路上で $\mathbf{x}_t$（$t \in \{0.25, 0.5, 0.75\}$）までノイズを加えた場合について報告せよ。これらの正解率から、本文でノイズ画像上で訓練された分類器が必要だと述べた理由をどのように説明できるか。
   2. 代わりに無条件予測（`cfg(ema, c, 0.0)`）の事後平均 $\hat{\mathbf{x}}_0(\mathbf{x}_t)$ を分類器の入力として用い、無条件スコア関数に $\gamma \nabla_{\mathbf{x}_t} \log p_{\text{cnn}}(c \mid \hat{\mathbf{x}}_0(\mathbf{x}_t))$ を加えて変換辞書により速度へ

変換せよ。50 回のオイラーステップを用い、$\gamma \in \{1, 3, 10\}$ で 1,000 枚の画像をサンプリングせよ。

3. 生成したサンプルを `fd_and_agreement` でスコアリングせよ。このときクラス一致率はサンプラを誘導したのと同じネットワークで測定されている。別のシードで 2 つ目の `d2l.FeatureCNN` を訓練し、それを用いてもサンプルのスコアを計算せよ。両者の比較から、同じ分類器を誘導と評価の両方に使うことについて何が言えるか。

3. [code] **限定区間でのガイダンス**。 Kynkäänniemi et al. (2024) はノイズレベルのある区間内でのみガイダンスを適用し、それ以外では $\gamma = 1$ を用いる。同論文ではノイズレベルを $\bar{\sigma}_t = \sigma_t / \alpha_t$ で測り、ImageNet モデルの 1 つにおいて $0.28 < \bar{\sigma}_t \leq 2.9$ が最良であることを見出した。
   1. 線形経路において、時刻 t のうち $\bar{\sigma}_t$ が 0.28 および 2.9 に等しくなるものを求めよ。
   2. 強度 γ を指定された区間内の t に対してのみ適用する `cfg` を実装せよ。$\gamma = 4$ において、高ノイズ区間、中間区間、低ノイズ区間のそれぞれでガイダンスを行ったときの 5,000 サンプルの FD とクラス一致率を計算し、全ステップでのガイダンスおよび $\gamma = 1$ の場合と比較せよ。
   3. クラス一致率の向上の大半をもたらすのはどの区間であり、FD の上昇の大半をもたらすのはどの区間か。その答えを、サンプルがクラスを確定させる時刻に関する演習問題 4（17.3 節）および本節の予測画像の並びの 3 行目と関連づけて説明せよ。なお、演習問題 5（38.6 節）では 1 次元の 2 クラスについて同じ問いを扱っている。
4. [code] **スキップ接続**。`UNet` から 2 つのスキップ接続を取り除き、`up1` と `up0` がアップサンプリングされた特徴量のみを受け取るようにして入力チャネル幅をそれぞれ `2 * ch` と `ch` に縮小した変種を実装せよ。
   1. 訓練を行う前に、どの時刻 t でスキップ接続が最も重要になり、どの時刻ではほとんど影響しないかを予測せよ。
   2. 両方のネットワークを 5,000 回の更新で訓練せよ。1,000 枚のテスト画像に対し、各ネットワークの速度が含意するクリーン画像 $\hat{\mathbf{x}}_0 = \mathbf{x}_t - t\,\hat{\mathbf{u}}$ の二乗誤差を $t \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ で計算し、両方の曲線をプロットせよ。
   3. 各ネットワークの 5,000 サンプルに対する FD を比較せよ。(b) の誤差曲線からその差を予測できるか。
5. [code] **予測画像のしきい値処理**。●線形経路上では、ガイド付き速度からクリーン画像 $\hat{\mathbf{x}}_0 = \mathbf{x}_t - t\,\tilde{\mathbf{u}}$ が定まる。Saharia et al. (2022) は、大きなガイダンス強度で生じる飽和したサンプルの原因を、データの値域を外れた予測 $\hat{\mathbf{x}}_0$ に求めている。
   1. $\gamma \in \{1, 4, 16\}$ に対し、すべてのオイラーステップにおいて $\hat{\mathbf{x}}_0$ のピクセルのうち $[-1, 1]$ の外に出る割合を記録し、t に対してプロットせよ。経路上のどの領域で予測が値域を外れるか。また γ によってそれはどう変化するか。
   2. **静的しきい値処理**はステップの前に $\hat{\mathbf{x}}_0$ を $[-1, 1]$ にクリップする。一方、**動的しきい値処理**は画像ごとに m を 1 と $|\hat{\mathbf{x}}_0|$ の 99.5 パーセンタイルのうち大きい方に設定し、$[-m, m]$ にクリップしてから m で割る。しきい値処理後の $\hat{\mathbf{x}}_0$ から速度を構成するようにオイラーステップを書き換え、両方の手法を実装せよ。
   3. 画像集合の**飽和率**を、クランプ後に 1 に等しくなるピクセルの割合として定義する。$\gamma \in \{4, 8, 16\}$ において、3 つのサンプラの飽和率、FD、クラス一致率を比較せよ。
6. **ガウスクラスに対するガイダンス**。●本節のガウスクラスの設定、すなわちクラス c が $\mathcal{N}(\mu_c, s^2\mathbf{I})$ に従い、全画像が $\mathcal{N}(\mathbf{0}, S^2\mathbf{I})$（$S^2 > s^2$）に従い、線形経路上でノイズが加えられる場合を考える。
   1. 本文で示した $t = 0$ における傾斜ガウス分布の逆分散と平均を導出せよ。さらに、すべての t においてガイド付きスコア関数 $\gamma \nabla \log p_t(\mathbf{x} \mid c) + (1 - \gamma) \nabla \log p_t(\mathbf{x})$ がガウス分布のスコア関数になることを示してその分散を求め、その分散が t^2 を $t = 1$ 付近で下回る条件が $\gamma > S^2 / (S^2 - s^2)$ のとき、かつそのときに限ることを示せ。
   2. 1 次元で $\mu_c = 1$、$s^2 = 0.25$、$S^2 = 2$ とすると、ガイド付き速度は x に関してすべての t でアフィン関数になる。平均と分散に対する ODE を書き下し、$\mathcal{N}(0, 1)$

($t = 0.999$) から $t = 0$ まで $\gamma \in \{1, 2, 4\}$ について積分して、本文の $\gamma = 4$ における終端値（平均 1.23、標準偏差 0.02）を確認せよ。各終端分布を $t = 0$ における傾斜ガウス分布と比較し、本文の上界を用いてその違いを説明せよ。

3. 演習問題 5（16.6 節）の打ち切りトリックを生成器 $\mu_c + s\mathbf{z}$ に適用する手法や、$p(\mathbf{x} \mid c)^{1/T}$ から温度 $T < 1$ でサンプリングする手法もクラスの広がりを狭める。これら 3 つの手法のうち、クラスの平均も移動させるものはどれか。また、それはなぜか。

17.5 サンプラ、逆変換、編集

拡散サンプラの各ステップではスコアまたは速度のネットワークを 1 回評価するため、画像を生成する計算コストはステップ数に比例して増大する。17.4 節では、確率フロー ODE（probability-flow ODE）の 50 回のオイラーステップによって、1,000 ステップの祖先連鎖（ancestral chain）に近いフレシェ距離（Fréchet distance、FD）を 20 分の 1 のコストで達成した。17.2 節の訓練における回帰は、複数ステップにわたる同時分布ではなく、$\mathbf{x}_t$ の単一の時刻 t における周辺分布のみに依存するため、1 つの訓練済みネットワークで確率的および決定論的な逆過程の連続的な族をサポートできる (J. Song et al. 2021)。

本節では、**ノイズ除去拡散陰的モデル**（denoising diffusion implicit model、DDIM）の族を導出し、その決定論的なメンバー（$\eta = 0$）が $(\bar{\mathbf{x}}, \bar{\sigma})$ 座標系における確率フロー ODE 上のオイラー法と一致することを示す。さらに、1 次（オイラー法 / DDIM）、2 次（ホイン法）、および多段（アダムス・バッシュフォース法 / DPM-Solver++）のソルバを、さまざまなステップ予算と時間グリッドにわたって比較する。最後に、決定論的 ODE の可逆性を、画像からノイズへの逆変換（inversion）、潜在空間での補間、部分ノイズ付加による編集、およびゼロショットインペインティングに用いる。

17.5.1 決定論的サンプラと確率的サンプラ

17.3 節の 2 つのサンプラである祖先連鎖とオイラー法は、一方は各ステップで新たなノイズを加え、もう一方はまったく加えないという違いはあるものの、同じ方法でネットワークの出力を読み取る。ガウス経路（Gaussian path）上の点 $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$ において、ネットワークはクリーンな点の推定値 $\hat{\mathbf{x}}_0$ とノイズの推定値 $\hat{\epsilon}$ を与え、一方が決まればもう一方も次のように定まる。

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sigma_t \hat{\epsilon}}{\alpha_t}. \tag{17.33}$$

ネットワークが何を予測するように訓練されていようとも、`d2l.make_pred`（17.3 節）はそれを任意のノイズレベルでこの組を返す関数 `pred(x, alpha, sigma)` へと変換し、本節のサンプラはそれ以外の情報を一切参照しない。

17.5.1.1 DDIM 族とオイラー法

時刻 t とそれより前の時刻 $s < t$ をとる。t から s へのサンプラのステップは、$\mathbf{x}_t$ と同じ分布に従う点を $\mathbf{x}_s$ と同じ分布に従う点へと写す必要があり、回帰が課す制約はそれだけである。$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$ と表し、ノイズスケール $\tilde{\sigma}$ を 0 と σ_s の間で選んで、次のようにおく。

$$\mathbf{x}_s = \alpha_s \mathbf{x}_0 + \sqrt{\sigma_s^2 - \tilde{\sigma}^2}\, \epsilon + \tilde{\sigma}\, \xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{17.34}$$

ここで、ξ は ϵ と独立である。2 つのノイズ分散の和は σ_s^2 になるため、$\mathbf{x}_0$ が与えられたもとで点 $\mathbf{x}_s$ は任意の $\tilde{\sigma}$ に対して正しい分布を持つ。また、$\epsilon = (\mathbf{x}_t - \alpha_t \mathbf{x}_0)/\sigma_t$ であるから、各 $\tilde{\sigma}$ は $\mathbf{x}_t$ から $\mathbf{x}_s$ へのステップを定める。サンプラは、$\mathbf{x}_0$ と ϵ の代わりにネットワークの推定値を用いる。

$$\mathbf{x}_s = \alpha_s \hat{\mathbf{x}}_0 + \sqrt{\sigma_s^2 - \tilde{\sigma}^2}\,\hat{\epsilon} + \tilde{\sigma}\,\xi. \tag{17.35}$$

このステップは、予測されたクリーンな点を予測されたノイズとともにレベル s へ移し、そのノイズの一部を新たなノイズで置き換える。J. Song et al. (2021) は、この新たなノイズの割合を強度 $\eta \geq 0$ によって次のように表している。

$$\tilde{\sigma} = \eta\, \sigma_s \sqrt{1 - \kappa^2}, \qquad \kappa = \frac{\sigma_s \alpha_t}{\alpha_s \sigma_t}. \tag{17.36}$$

ここで $\kappa^2 = \mathrm{SNR}_t / \mathrm{SNR}_s < 1$ であり、$\mathrm{SNR} = \alpha^2/\sigma^2$ である。$\eta = 1$ のとき、$\tilde{\sigma}^2$ は、$\mathbf{x}_s$ が $\mathbf{x}_t$ と $\mathbf{x}_0$ を与えられたときに 17.2 節のノイズ付加連鎖のもとで持つ分散である（演習問題 1）。このときステップは、真の値の代わりに推定値を用いてその連鎖の事後分布からサンプリングする。$\eta = 0$ では新たなノイズは一切入らず、ステップは決定論的になる。これが**ノイズ除去拡散陰的モデル**（DDIM）であり、密度の連鎖ではなくノイズからの写像によってサンプルを定義することから陰的（implicit）と呼ばれる。

`ddim_sample` は、`path.grid` のグリッド上で任意の予測器 `pred(x, alpha, sigma) -> (x0, eps)` を用いて DDIM 族を実行し、レベル間でループがサンプルを操作できるよう `t_end`（デフォルトではデータ）で停止する。

```
def ddim_sample(pred, x, path, steps, eta=0.0, key=None, *,
                t_start=1.0, t_end=0.0):
    """DDIM from x at t_start (1 is noise) to t_end; eta > 0 needs a key.
    eta = 1 draws the posterior variance: not the ancestral sampler's."""
    if eta > 0 and key is None:
        raise ValueError('ddim_sample with eta > 0 draws noise: pass a key')
    a, s = path.grid(steps, t_start, t_end)
    a_t, s_t, a_s, s_s = a[:-1], s[:-1], a[1:], s[1:]
    ratio = s_s * a_t / (a_s * s_t)                    # kappa of the text
    var = (eta * s_s) ** 2 * (1 - ratio ** 2)          # fresh noise
    def step(x, inputs):
        a_t, s_t, a_s, c_eps, c_noise, key = inputs
        x0, eps = pred(x, a_t, s_t)
        x = a_s * x0 + c_eps * eps
        if eta > 0:
            x = x + c_noise * jax.random.normal(key, x.shape)
        return x, None
    keys = (jax.random.split(key, steps) if eta > 0
            else jnp.zeros(steps))                     # unused at eta = 0
    return jax.lax.scan(step, x, (a_t, s_t, a_s, jnp.sqrt(s_s ** 2 - var),
                                  jnp.sqrt(var), keys))[0]
```

長い決定論的ストライドはどれほど大きな誤差を生むだろうか。分散保持（variance-preserving）経路上で、データ自体が標準ガウス分布に従う場合を考えよう。このとき $\mathbf{x}_t$ はあらゆる時刻で標準ガウス分布に従うため、厳密なサンプラはどの点も動かさないはずである。厳密な推定値は $\hat{\mathbf{x}}_0 = \alpha_t \mathbf{x}_t$ および $\hat{\epsilon} = \sigma_t \mathbf{x}_t$ となり、$t = 1$ から $t = 0$ への 1 回の決定論的ストライドは $\hat{\mathbf{x}}_0 = \alpha_1 \mathbf{x}_1$ を返す。$\alpha_1 \approx 0.0066$ であるため、点群全体がその平均へと潰れてしまう。厳密な

軌跡に沿ってはノイズ推定値が変化するのに対し、このステップはストライド全体にわたってノイズ推定値を固定してしまうからである。

$\eta = 0$ のとき、このステップは $\mathbf{x}_s = \alpha_s \hat{\mathbf{x}}_0 + \sigma_s \hat{\epsilon}$ となり、`ode_sample` が 17.3 節でとるステップに一致する。そこで導出した座標 $\bar{\mathbf{x}} = \mathbf{x}/\alpha_t$ と $\bar{\sigma}_t = \sigma_t/\alpha_t$ では、これは次の式で表される。

$$\bar{\mathbf{x}}_s = \bar{\mathbf{x}}_t + (\bar{\sigma}_s - \bar{\sigma}_t)\,\hat{\epsilon}, \tag{17.37}$$

これは確率フロー ODE $d\bar{\mathbf{x}}/d\bar{\sigma} = \hat{\epsilon}$ の 1 回のオイラーステップであり、上の比 κ は単に $\bar{\sigma}_s/\bar{\sigma}_t$ となる。したがって、あらゆるガウス経路において DDIM はオイラー法であり、$\eta = 0$ のとき同じレベル列上の `ddim_sample` と `ode_sample` は 1 つの関数を 2 つの座標系で計算しているにすぎない。38.6.4 節ではこの等価性とともに、2 つの極端な場合を証明する。すなわち、単一のデータ点では各軌跡に沿って $\hat{\epsilon}$ が一定になるため任意の長さのストライドで誤差なく積分される一方、ガウスデータはストライドごとに縮小する（Figure 38.25）。

17.5.1.2 新たなノイズと長いステップ

1 つのネットワークがあらゆる η に対応できるなら、どの強度を使うべきだろうか。$f = \tilde{\sigma}^2/\sigma_s^2$ とおくと、Equation 17.35 のステップは次のようになる。

$$\mathbf{x}_s = \alpha_s \hat{\mathbf{x}}_0 + \sigma_s\Big(\sqrt{1-f}\,\hat{\epsilon} + \sqrt{f}\,\xi\Big), \qquad f = \eta^2\Big(1 - \frac{\mathrm{SNR}_t}{\mathrm{SNR}_s}\Big). \tag{17.38}$$

どのメンバーもクリーン推定値を同じようにレベル s に配置し、付随するノイズ分散のうち割合 f だけが新たなノイズとなる。細かいグリッド上では、f は $\eta = 1$ であっても小さいが、10 ステップのグリッド上では $\eta = 1$ のサンプラは各ステップでノイズの大半を入れ替える。

新たなノイズを加えることにはもっともな根拠がある。17.3 節の祖先ステップは、1 次近似において、現在の分布 p_t を不変に保ちつつ外れた点をそこへ引き戻すランジュバンステップと、確率フロー ODE のオイラーステップとに分解される。ノイズを注入して周辺分布を保つあらゆる逆過程も同様に分解される（38.5.5 節）。したがって、確率的サンプラは状態をモデルの分布へと引き戻し続けるため、あるステップで生じた誤差は次のステップで部分的に忘れ去られる可能性がある。一方、決定論的サンプラはすべての誤差を最後まで持ち越す。

同時に、新たなノイズに反対する同程度にもっともな根拠もある。ランジュバンによる引き戻しは、短いステップについての性質である。長いステップにわたる厳密な逆遷移はガウス分布からは程遠く（演習問題 3、17.3 節）、長い確率的ステップはそれ自体の誤差をつけ加えてしまう。少ない予算では決定論的サンプラが勝るはずであり、ステップが短くなったときに確率的サンプラがそれを追い抜くかどうかは、17.5.2.4 節で画像を用いて測定する。Figure 17.9 に、混合分布上でのこの族の両端を示す。

2 つの確率的サンプラは明確に区別しなければならない。祖先連鎖は順ステップの分散 $\sigma_t^2 - (\alpha_t/\alpha_s)^2\sigma_s^2$ を加えるのに対し、$\eta = 1$ はそれより係数 σ_s^2/σ_t^2 だけ小さい事後分散（posterior variance）を加える。どちらも同じ平均へ移動する。したがって、祖先連鎖はこの族のメンバーではない。保持するノイズと追加するノイズの分散の和が σ_s^2 を超えるからである。どちらの分散が正しいのだろうか。一般には、どちらも厳密ではない。データが単一の点であるときは $\mathbf{x}_t$ から $\mathbf{x}_0$ が一意に定まるため事後分散が厳密になり、分散保持経路上の標準ガウスデータに対しては順ステップの分散が厳密になる。座標あたりの分散が 1 未満でノイズより狭いデータでは、最良のガウス分散は両者の間に位置し、より広いデータでは順ステップの分散さえ超えることがある（演習問題 4）。1,000 ステップのグリッド上では、両方の分散がごく小さくなる最後の数ステップを除いてこの係数は 1 に近いため、どちらを選んでもほとんど差はないが、50 ステップでは大きな違いが生じる。

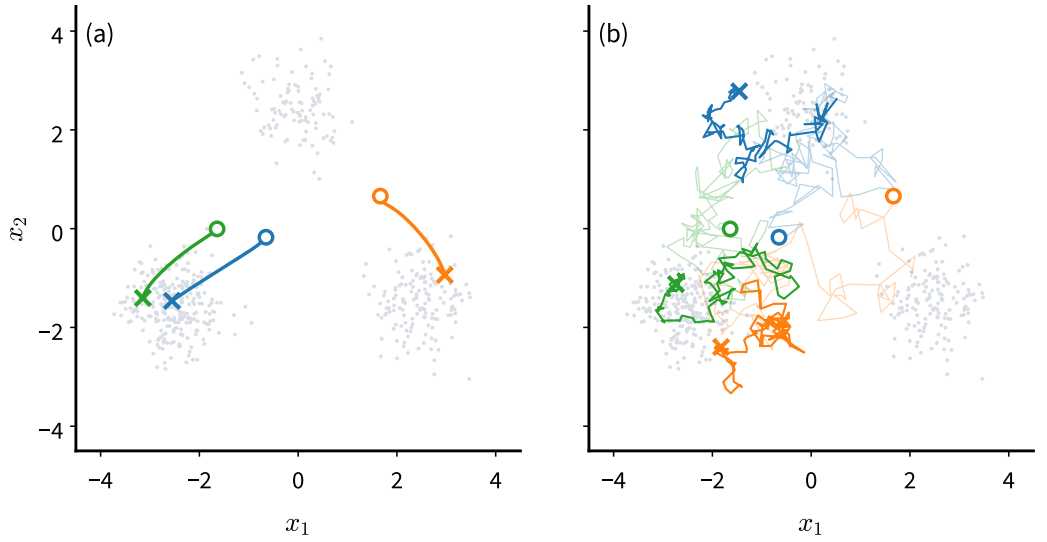


Figure 17.9: 厳密な予測器を用いた 3 モード混合分布上での DDIM 族の両端。分散保持経路上の 200 ステップで、$t = 1$ の 3 つの開始点（丸）から $t = 0$（バツ）まで進めた軌跡であり、訓練データを灰色で示す。(a) $\eta = 0$ では、各開始点は滑らかな曲線をたどってモードに入る。(b) $\eta = 1$ では経路がぎざぎざになり、ここでは 3 つのうち 2 つが (a) とは異なるモードに到達している。ノイズが支配的な各経路の前半は薄く描かれている。どちらも $t = 0$ では同じ分布を生成する。

17.5.2 ステップ数の削減

1 次手法はステップサイズに比例する誤差を生じるため、ステップを間引くと予測可能な形で精度が低下する。まずはその低下量を厳密に計算できる設定で測定し、続いてより高次の手法やより適切なステップ配置によってどれだけ回復できるかを調べる。あらかじめ断っておくと、画像上で得られる数値は 1 つの小規模なモデル固有のものであり、別のモデルにはそのまま当てはまらない。しかし、その数値を導き出す手順はそのまま通用する。

17.5.2.1 通し例におけるステップの間引き

ここで 3 モード混合分布 `mix`（17.1 節）に戻ろう。この分布ではあらゆるレベルで事後平均（posterior mean）、ひいては厳密な組 $(\hat{\mathbf{x}}_0, \hat{\epsilon})$ を計算でき、`pred_exact` はその組をサンプラのインターフェースで包んだ関数である。20,000 個の開始点と同数のデータ点を抽出し、データの**広がり**（spread）、すなわち各点から最も近いモード中心までの平均二乗距離（17.1 節）を出力する。各モードは座標あたり 0.25 の分散を持つため、広がりは 0.5 をわずかに下回る。

```
mix = d2l.GaussianMixture.three_modes()
vp, lin = d2l.GaussianPath.vp(), d2l.GaussianPath.linear()

def pred_exact(x, alpha, sigma):
    """The mixture's exact (x0, eps) at x = alpha x0 + sigma eps."""
    x0 = mix.posterior_mean(x, alpha, sigma)
    return x0, (x - alpha * x0) / sigma

k_z, k_data, k_eta = jax.random.split(jax.random.key(0), 3)  # k_eta: per n
```

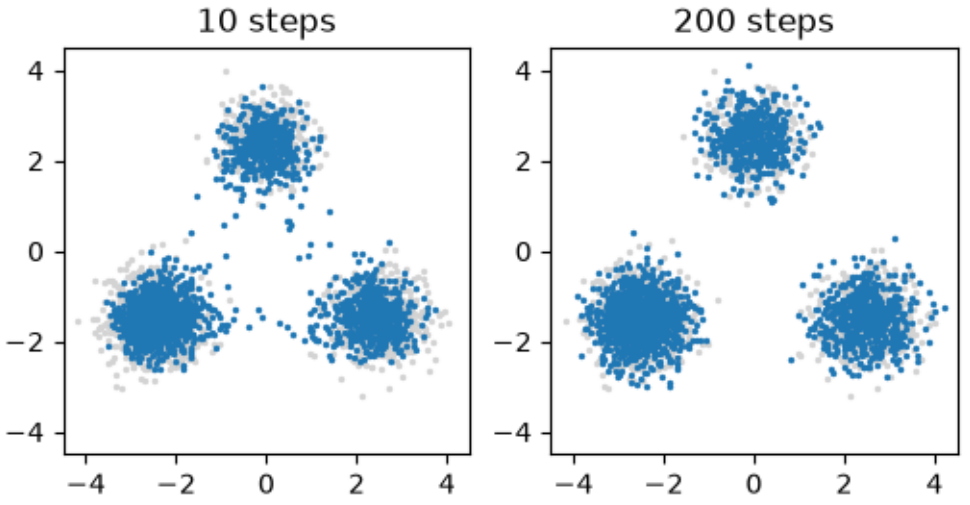
```
z, data = jax.random.normal(k_z, (20_000, 2)), mix.sample(k_data, 20_000)
print(f'data: spread {mix.summary(data)["spread"]:.3f}')
```

```
data: spread 0.473
```

厳密な予測器を用いる場合、同じ開始点からの 10 ステップのサンプルと 1,000 ステップのサンプルの差はすべてストライドの誤差であり、1 次手法ではステップ数を 2 倍にするごとに半減するはずである。分散保持グリッド上で 5〜1,000 ステップにわたって DDIM を実行し、1,000 ステップのサンプルまでの平均距離と広がりを出力する。$\eta = 0$ の広がりの横には、同じ開始点からの $\eta = 1$ および祖先連鎖の広がりも並べて示す。

```
samples = {n: d2l.ddim_sample(pred_exact, z, vp, n)
           for n in (5, 10, 20, 50, 200, 1000)}
spread = lambda pts: f'{mix.summary(pts)["spread"]:.3f}'
for n, x in samples.items():
    gap = jnp.linalg.norm(x - samples[1000], axis=1).mean()
    k1, k2 = jax.random.split(jax.random.fold_in(k_eta, n))
    eta1 = d2l.ddim_sample(pred_exact, z, vp, n, 1.0, k1)
    chain = d2l.ancestral_sample(pred_exact, z, vp, n, k2)
    print(f'{n:4d} steps: distance to 1,000 steps {gap:.3f}, spread '
          f'{spread(x)} (eta = 1: {spread(eta1)}, ancestral: {spread(chain)})')
d2l.scatter_panels([samples[10][:2000], samples[200][:2000]],
                   ['10 steps', '200 steps'], data=data[:2000]);
```

```
   5 steps: distance to 1,000 steps 0.412, spread 0.312 (eta = 1: 0.329, ancestral: 0.386)
  10 steps: distance to 1,000 steps 0.212, spread 0.383 (eta = 1: 0.318, ancestral: 0.523)
  20 steps: distance to 1,000 steps 0.109, spread 0.421 (eta = 1: 0.338, ancestral: 0.519)
  50 steps: distance to 1,000 steps 0.043, spread 0.456 (eta = 1: 0.395, ancestral: 0.498)
 200 steps: distance to 1,000 steps 0.009, spread 0.474 (eta = 1: 0.452, ancestral: 0.481)
1000 steps: distance to 1,000 steps 0.000, spread 0.478 (eta = 1: 0.475, ancestral: 0.474)
```



距離は予測どおりステップ数を 2 倍にするごとに半減し、モード間が約 5 単位離れたスケール上で、5 ステップでは約 0.4、10 ステップでは 0.2、50 ステップでは 0.04 となる。また、誤差には一定の向きがある。10 ステップにおける決定論的サンプルの広がりは約 0.38 で、データの 0.47〜

0.48 に対して小さく、左パネルの点群は背後にあるデータよりも目に見えて引き締まっている。1 回のストライドはノイズ推定値を開始時点の値に固定するため、ガウスデータに対しては常にノイズを取り除きすぎてしまう（演習問題 2）。そして、モードを識別できるほどノイズが小さくなると、混合分布の各モードは座標あたり分散 0.25 のガウスデータのように振る舞う。

確率的サンプラは両方向に外れる。10 ステップでは、$\eta = 1$ はモードをさらに収縮させて広がりを約 0.32 にする一方、祖先連鎖はデータ以上に広げて約 0.52 にする。ノイズを加える $\eta = 1$ の方が、ノイズを一切加えない $\eta = 0$ よりも狭く収まるのは一見すると意外である。DDIM 族のどのメンバーも、同じ予算 σ_s^2 を保持ノイズと新たなノイズに振り分ける。ガウスデータではクリーン推定値とノイズ推定値がどちらも $\mathbf{x}_t$ の正の定数倍になるため、保持ノイズはクリーン推定値と同じ向きに揃い、交差項を通じて広がりを押し上げる。新たなノイズは両者と独立であり、それ自体の分散を加えるだけでそれ以上の寄与はない。したがって、粗いグリッド上で $\eta = 1$ が行うように予算を保持ノイズから新たなノイズへ移すと、サンプルは狭くなる（演習問題 2 で証明する）。祖先連鎖は $\eta = 1$ と同じ量のノイズを保持しつつ予算を超える新たなノイズを加えるが、その超過分はノイズと同程度の幅を持つデータに適したものであり、ノイズより狭いモードには過剰である。細かいグリッド上では、3 つのサンプラはすべてデータの広がりに到達する。どちらの向きにもずれる統計量ではその大きさだけでサンプラを順位付けできないため、画像に対してはどちらの向きの誤差にもペナルティを課すスコアを用いる。

17.5.2.2 2 次手法とステップグリッド

オイラー法は 1 ステップの間ずっと傾きを固定するため、実行全体にわたる誤差はステップサイズに比例してしか減少しない。ホイン法（Heun's method、38.2 節）は、オイラーステップを試行ステップとして進み、その試行点で傾きをもう一度評価して、2 つの傾きの平均で移動する。

$$\begin{aligned}\bar{\mathbf{x}}' &= \bar{\mathbf{x}}_t + (\bar{\sigma}_s - \bar{\sigma}_t)\,\hat{\epsilon}(\bar{\mathbf{x}}_t, \bar{\sigma}_t),\\ \bar{\mathbf{x}}_s &= \bar{\mathbf{x}}_t + (\bar{\sigma}_s - \bar{\sigma}_t)\,\frac{\hat{\epsilon}(\bar{\mathbf{x}}_t, \bar{\sigma}_t) + \hat{\epsilon}(\bar{\mathbf{x}}', \bar{\sigma}_s)}{2}.\end{aligned}$$

その誤差はステップあたり 2 回の評価で、ステップサイズの二乗に比例して減少する。標準ガウスデータではノイズ推定値は $\hat{\epsilon} = \bar{\sigma}\bar{\mathbf{x}}/(1+\bar{\sigma}^2)$ であり、厳密なフローは $\bar{\mathbf{x}}$ に $\sqrt{(1+\bar{\sigma}_s^2)/(1+\bar{\sigma}_t^2)}$ を掛ける。$\bar{\sigma}_t = 1$ から $\bar{\sigma}_s = 0.5$ への厳密な倍率は 0.791 であるのに対し、オイラーステップは 0.75、ホイン法は 0.80 を与える。誤差はそれぞれ 0.041 と 0.009 であり、2 倍の評価回数で誤差は約 4 分の 1 になる。ただし、2 つのステップでは 2 回目の傾きをとることができない。$\bar{\sigma} = 0$ では推定すべきノイズが残っておらず、$\bar{\sigma} = \infty$ からのステップは無限に長いため $\hat{\mathbf{x}}_0 + \bar{\sigma}_s\hat{\epsilon}$ の形でしか書けない。したがって、$\bar{\sigma} = 0$ への最後のステップと $\bar{\sigma} = \infty$ からの最初のステップはオイラーステップとなり、`heun_sample` が使う評価回数は $2n - 1$ 回（n ステップあたり）、線形経路上で純粋なノイズから始める場合は $2n - 2$ 回となる。

2 回目の評価が有効かどうかは、ステップをどこに配置するかによって決まる。厳密な速度場を持つ 1 次元混合分布において、均等グリッド上のホイン法は約 40 回未満の評価回数ではオイラー法に劣る一方、データ側にステップを密に寄せたグリッド上ではあらゆる予算でオイラー法を上回り、同じ密な配置はオイラー法の誤差を 2 倍以上に悪化させる（38.7.4 節）。Karras et al. (2022) は、従来のサンプラが一括して決めていた手法の次数とステップ配置という 2 つの選択を切り離し、$\bar{\sigma}^{1/\rho}$ を等間隔に並べることでステップをデータ側に寄せた。

$$\bar{\sigma}_i = \left(\bar{\sigma}_{\max}^{1/\rho} + \tfrac{i}{n-1}\left(\bar{\sigma}_{\min}^{1/\rho} - \bar{\sigma}_{\max}^{1/\rho}\right)\right)^{\rho}, \qquad i = 0, \dots, n-1, \tag{17.39}$$

これに続けて最後に $\bar{\sigma} = 0$ へのステップをとり、$\bar{\sigma}_{\max} = 80$、$\bar{\sigma}_{\min} = 0.002$、$\rho = 7$ と設定する（画素値が $[-1, 1]$ の画像の場合）。`make_pred` が任意のレベルをネットワーク固有の時刻へ

と変換するため、グリッドは訓練経路とは独立したサンプラ側の選択肢となる。edm_grid はこれらのレベルを返し、heun_sample は任意のレベル列に沿ってホインステップを進める。ここでは 3 つの選択肢、すなわち分散保持経路上で時間に関して均等なグリッド、線形経路上で時間に関して均等なグリッド、および Karras らのグリッド（論文 **Elucidating the Design Space of Diffusion-Based Generative Models** にちなんで EDM グリッドと呼ぶ）について、それぞれの 10 ステップグリッドを出力する。

```
def edm_grid(steps, sigma_max=80.0, sigma_min=0.002, rho=7):
    """Karras et al.'s levels sigma_bar: even in sigma_bar^(1/rho), then 0."""
    hi, lo = sigma_max ** (1 / rho), sigma_min ** (1 / rho)
    return [(hi + i / (steps - 1) * (lo - hi)) ** rho
            for i in range(steps)] + [0.0]

def heun_sample(pred, x, sigma_bars):
    """Heun on d x_bar / d sigma_bar = eps: an Euler trial, then the average
    of the two slopes; from x at the first level to data, as ode_sample."""
    with jax.ensure_compile_time_eval():          # the grid is a constant
        r = jnp.asarray(sigma_bars)
        from_inf = bool(jnp.isinf(r[0]))
    def step(xb, r):
        r_t, r_s = r
        x0, eps = d2l.pred_bar(pred, xb, r_t)
        x_next = x0 + r_s * eps                   # the Euler step
        correct = lambda _: xb + (r_s - r_t) * (
            eps + d2l.pred_bar(pred, x_next, r_s)[1]) / 2
        heun = jnp.isfinite(r_t) & (r_s > 0)      # no slope at inf or 0
        return jax.lax.cond(heun, correct, lambda _: x_next, None), None
    xb = x if from_inf else x * jnp.sqrt(1 + r[0] ** 2)  # x / alpha
    return jax.lax.scan(step, xb, (r[:-1], r[1:]))[0]

for name, levels in (('VP', vp.sigma_bars(10)), ('linear', lin.sigma_bars(10)),
                     ('EDM', d2l.edm_grid(10))):
    print(f'{name:7s}', ' '.join(f'{float(r):7.3g}' for r in levels))
```

```
VP          152    58.8    25.1    11.8     6.1    3.41    2.03    1.23    0.72    0.34       0
linear      inf       9       4    2.33     1.5       1   0.667   0.429    0.25   0.111       0
EDM          80    42.4    21.1    9.72    4.07     1.5    0.47   0.117  0.0204   0.002       0
```

時間に関して均等なグリッドも、$\bar{\sigma}$ で見るとまったく異なる姿をしている。分散保持グリッドは 10 ステップのうち 4 ステップを 152 から 6.1 の間に費やし、0.34 からデータへ到達する。線形グリッドは 9 より上を無限大からの 1 ステップで通過し、4 から 0.11 の間に 7 ステップを費やす。EDM グリッドは 80 から始まり、同じく 4 より上に 4 ステップを費やしたあと、最後の 3 つのレベルを 0.12 未満に密集させる。どの配分がモデルに適しているかはレベルの数値だけからは読み取れないため、以下で画像を用いて測定する。

17.5.2.3 指数積分器と多段ソルバ

ホイン法で 2 次精度を得るには、ステップあたり 2 回目の評価が必要になる。より低コストな道筋はこれまで使ってきた座標系から出発するものであり、実はすでにその道筋の上にいたのである。$\mathbf{x}$ 座標では、確率フロー ODE には $\mathbf{x}$ に関して線形でネットワークに一切依存しない項が含まれている。たとえば分散保持経路では、ドリフトはスコア関数が何であれ $\mathbf{x}$ を速度 $\frac{1}{2}\beta(t)$ で縮小させる。ここで $\beta(t)$ はノイズスケジュールの変化率である。この線形部分の厳密解は指数減衰、

ここでは係数 α_t であり、α_t で割ることでこの項が消去される。残る $d\bar{\mathbf{x}}/d\bar{\sigma} = \hat{\epsilon}$ にはネットワークの項しか含まれない。線形部分を厳密に積分し、ネットワークの項だけを近似するのが**指数積分器**（exponential integrator）の考え方である (Zhang and Chen 2023; Lu, Zhou, et al. 2022)。ステップにわたって $\hat{\epsilon}$ を一定に保つ DDIM はその 1 次のメンバーであり、したがって DDIM をオイラー法に見せた座標変換こそがこれらのソルバの中心的なアイデアだったのである。より高次のメンバーは、ステップ内で $\hat{\epsilon}$ をどう変化させるかという点だけが異なる。DPM-Solver (Lu, Zhou, et al. 2022) は、$\log \bar{\sigma}$ におけるステップの中点で 2 回目の評価を行う（演習問題 3）。

この 2 回目の評価は省くことができる。直前のステップですでに通過したばかりのレベルにおける $\hat{\epsilon}$ が評価済みであるため、**多段**（multistep）法はそれを再利用し、直近 2 回の推定値から現在のステップの中点へと $\hat{\epsilon}$ を線形外挿する。現在のステップ幅を $\Delta = \bar{\sigma}_s - \bar{\sigma}_t$、直前のステップ幅を Δ'、それぞれの開始点における推定値を $\hat{\epsilon}_t$、$\hat{\epsilon}'$ とすると、2 次のアダムス・バッシュフォース（Adams–Bashforth）ステップは次の式で与えられる。

$$\bar{\mathbf{x}}_s = \bar{\mathbf{x}}_t + \Delta\left(\hat{\epsilon}_t + \frac{\Delta}{2\Delta'}\left(\hat{\epsilon}_t - \hat{\epsilon}'\right)\right). \tag{17.40}$$

ノイズ推定値がステップ間で変化しないとき、この更新は DDIM に一致し、変化するときはステップあたり 1 回の評価のままその変化に追随して補正を行う。最初のステップには直前の推定値がないためオイラーステップとなり、$\bar{\sigma} = \infty$ から始めた直後のステップでも Δ' が無限大になるため同様にオイラーステップとなる。DPM-Solver++(2M) (C. Lu et al. 2025) は同じ考え方をクリーン画像の推定値に適用し、$\log \bar{\sigma}$ 上で外挿する手法であり、10〜25 回の評価回数ではこれとその派生手法が標準的な選択肢となっている。`multistep_sample` は任意のレベル列に対して上のステップを実行する。

```
def multistep_sample(pred, x, sigma_bars):
    """Second-order Adams-Bashforth on d x_bar / d sigma_bar = eps: one
    evaluation per step, the slope extrapolated from the last two."""
    with jax.ensure_compile_time_eval():          # the grid is a constant
        r = [float(v) for v in sigma_bars]
    c = [0.0] + [(r[k + 1] - r[k]) ** 2 / (2 * (r[k] - r[k - 1]))
                 if math.isfinite(r[k - 1]) else 0.0     # 0: Euler
                 for k in range(1, len(r) - 1)]
    def step(carry, inputs):
        xb, eps_prev = carry
        r_t, r_s, c = inputs
        x0, eps = d2l.pred_bar(pred, xb, r_t)
        return (x0 + r_s * eps + c * (eps - eps_prev), eps), None
    xb = x if math.isinf(r[0]) else x * math.sqrt(1 + r[0] ** 2)  # x / alpha
    return jax.lax.scan(step, (xb, jnp.zeros_like(xb)),
                        (jnp.array(r[:-1]), jnp.array(r[1:]),
                         jnp.array(c)))[0][0]
```

17.5.2.4 画像における評価回数と品質

画像上では、速度場は学習によって求める。ここでは 17.4 節のクラス条件付き U-Net（約 120 万個のパラメータ）を、線形経路の速度に対して 10,000 回の更新で訓練する（GPU 1 基で数分程度）。ラベルの 10 分の 1 はヌルラベルに置き換え、サンプリングには重みの移動平均を用いる。本節の残りの実験はすべてこの 1 つのネットワークで行う。ガイダンスなしのサンプラ（以下の $\gamma = 1$）はすべてのクラスを均等な頻度で生成し（サンプル i に i を 10 で割った余りのクラスを

指定)、ガイダンス付きのサンプラ（$\gamma > 1$）はヌルラベルもあわせて評価し、逆変換と編集では各画像にそれぞれのラベルを与える。

```
X, y, X_test, y_test = d2l.load_fashion_mnist_pm1()
loss = d2l.regression_loss(lin, 'velocity')

def batch_fn(key):
    k1, k2 = jax.random.split(key)
    i = jax.random.randint(k1, (128,), 0, len(X))
    drop = jax.random.uniform(k2, (128,)) < 0.1     # label dropout
    return X[i], jnp.where(drop, 10, y[i])          # 10 is the null label

def loss_fn(net, batch, key):  # the regression loss of the labeled field
    x0, c = batch
    return loss(d2l.conditional(net, c), x0, key)

net = d2l.UNet(ch=64, num_classes=10, rngs=nnx.Rngs(0))
net, ema, _ = d2l.train_regression(net, loss_fn, batch_fn, 10_000,
                                   jax.random.key(0))
```

1,000 枚のサンプルを、1,000 枚のテスト画像に対する `d2l.FeatureCNN` の特徴量空間（17.4 節）におけるフレシェ距離（FD）で評価する。このサンプル数では、実画像の集合どうしでも互いに 1 をわずかに下回るスコアとなり、画像の抽出ごとにスコアは 0.15〜0.2 ほど変動する（17.4.3.2 節）。そこで各測定は 1,000 枚のノイズ画像の 3 回の抽出に対して繰り返し、すべての比較が対になるよう全サンプラで同じノイズを共有したうえで、平均値に範囲の半分（半範囲）をプラスマイナスした形で報告する。フロア（下限値）は、テスト画像を別の 3 組の実画像集合と比較して求める。半範囲内に収まる差は実質的な差ではない。また `fd_runs` は次項で使用するガイダンス強度 γ も引数にとり、$\gamma = 1$ は通常の条件付き速度場を表す。

```
cnn = d2l.fashion_feature_cnn(X, y)
features = lambda imgs: d2l.fd_features(cnn, imgs)
f_test = features(X_test[:1000])
floors = [d2l.fid_score(features(X_test[k:k + 1000]), f_test)
          for k in (1000, 2000, 3000)]
labels = jnp.arange(1000) % 10                      # sample i has class i % 10
k_noise, k_run = jax.random.split(jax.random.key(1))
noises = [jax.random.normal(k, (1000, 28, 28, 1))   # shared by every sampler
          for k in jax.random.split(k_noise, 3)]
run_keys = iter(jax.random.split(k_run, 1000))      # fresh noise of the samplers

@nnx.jit(static_argnums=(0, 1, 2))  # one trace per sampler, step count, gamma
def sample(sampler, n, gamma, model, c, z, key):
    pred = d2l.make_pred(d2l.cfg(model, c, gamma), lin, 'velocity')
    return sampler(pred, z, n, key).clip(-1, 1)

def fd_runs(sampler, n, gamma=1.0):
    """FD of sampler(pred, noise, n, key) on each noise draw, field cfg(gamma)."""
    keys = jax.random.split(next(run_keys), len(noises))
    return [d2l.fid_score(features(sample(sampler, n, gamma, ema, labels, z, k)),
                          f_test) for z, k in zip(noises, keys)]

def mean_range(fds):  # the mean, plus or minus half the range of the draws
```

```
        return f'{np.mean(fds):5.2f}±{(max(fds) - min(fds)) / 2:.2f}'

    print('floor, 1,000 real images against 1,000 others: FD', mean_range(floors))
```

```
floor, 1,000 real images against 1,000 others: FD  0.88±0.20
```

フロアは約 0.9 である。最初の表は、上で未解決のまま残した問いに決着をつけるため、時間に関して均等なステップを持つ分散保持経路上で比較を行う。この設定では、$\eta = 0$ の DDIM は先ほど出力した最初のグリッド上のオイラー法に一致する。もし決定論的サンプラが最後まで持ち越す誤差をノイズが補正するのであれば、ステップが短くなるにつれて $\eta = 1$ と祖先連鎖は $\eta = 0$ に迫り、1,000 ステップに達する前に追い抜くはずである。

```
    vp_samplers = {  # the DDIM family and the ancestral chain, even steps on VP
        'DDIM, eta = 0': lambda p, z, n, k: d2l.ddim_sample(p, z, vp, n),
        'DDIM, eta = 1': lambda p, z, n, k: d2l.ddim_sample(p, z, vp, n, 1.0, k),
        'ancestral': lambda p, z, n, k: d2l.ancestral_sample(p, z, vp, n, k)}
    steps = (10, 20, 50, 100, 200, 1000)
    fd_vp = {name: [fd_runs(f, n) for n in steps]
             for name, f in vp_samplers.items()}
    print('steps        ', ''.join(f'{n:>11d}' for n in steps))
    for name, row in fd_vp.items():
        print(f'{name:13s}', ''.join(f'{mean_range(v):>11s}' for v in row))
```

```
steps                 10         20         50        100        200       1000
DDIM, eta = 0   5.46±0.14  2.93±0.11  1.68±0.08  1.36±0.07  1.26±0.07  1.26±0.08
DDIM, eta = 1   8.06±0.02  4.89±0.15  3.17±0.16  2.29±0.10  1.86±0.06  1.39±0.04
ancestral      30.31±0.27 21.53±0.57 10.64±0.43  5.26±0.28  2.42±0.13  1.30±0.04
```

両者は差を縮めるものの、決して追い抜くことはない。200 ステップまでは、どの予算でも $\eta = 0$ のスコアが $\eta = 1$ より 3 分の 1 からほぼ半分低く、半範囲を大きく超えて優位である。1,000 ステップでは 3 手法とも約 1.3〜1.5 の間に収まり、互いの差は約 0.2 以内となる。このモデルでは新たなノイズが役立つことは一度もなかった。どの予算でも確率的サンプラが半範囲を超えて $\eta = 0$ を上回ることはなく、1,000 ステップでも祖先連鎖は並ぶにとどまる。100 ステップ以下では祖先連鎖が最も大きく劣り、50 ステップで約 8〜11 と、10 ステップの DDIM（5 強）よりも悪い。順ステップの分散はノイズと同程度の幅を持つデータには適しているが、衣類画像はノイズが小さくなればほぼ確定するため、過剰な分散は斑点状のノイズとして現れるはずである。次項末尾の画像列でそれを確認する。DDIM 自体は 100 ステップ以降の変化が約 10 分の 1 にとどまり、50 ステップの時点で 20 分の 1 の評価回数ながら 1,000 ステップの連鎖から約 3 分の 1 以内に迫る。これが J. Song et al. (2021) の中心的な結果である。

新たなノイズに関するこの結論はこのモデル固有のものであり、大規模なモデルにそのまま当てはめてはならない。64×64 ピクセルの ImageNet では、Karras et al. (2022) が各ホインステップの前に調整された少量のノイズを加えることで決定論的サンプラを改善できることを見いだしたほか、Y. Song, Sohl-Dickstein, et al. (2021) の予測子・修正子（predictor-corrector）サンプラはソルバのステップ間にランジュバンステップを挿入して同様の効果を得ており、Restart サンプリング (Xu et al. 2023) は大きな予算で最も効果を発揮する。その仕組みは、上で述べたランジュバンによる引き戻しである。複雑なデータを扱う大規模モデルは後続のノイズでなお補正可能な誤差を生じるため、十分なステップ数があればノイズによる補正効果がノイズ自体の加える分散を上回るのである。

2 つ目の表は、サンプルの実質的なコストであるネットワーク評価回数で比較する。オイラー法、ホイン法、および多段サンプラを、線形グリッド、$\rho = 7$ の Karras らの EDM グリッド、そして

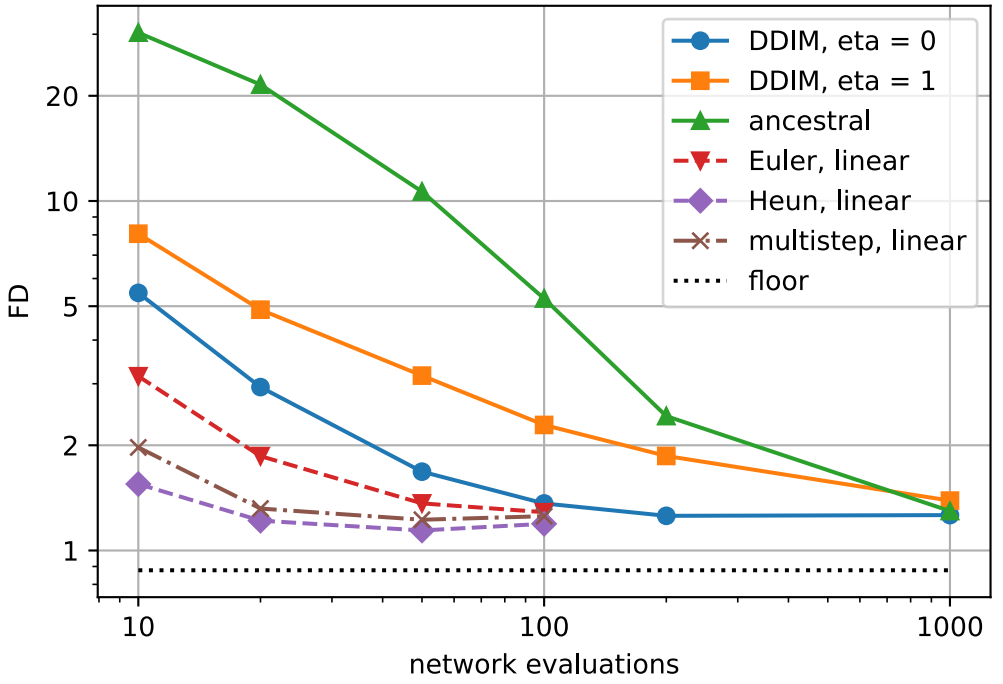
データ側から高ノイズ側へとレベルを移した $\rho = 3$ の EDM グリッドという 3 つのグリッド上で、約 10〜100 回の評価回数にわたって実行する。ホイン法は約半分のステップ数をとり、有限のレベルから始まる EDM グリッド上では予算より 1 回少ない評価回数となる。

```
grids = {'linear': lin.sigma_bars, 'EDM': d2l.edm_grid,
         'EDM, rho = 3': lambda n: d2l.edm_grid(n, rho=3)}
solvers = {'Euler': d2l.ode_sample, 'Heun': d2l.heun_sample,
           'multistep': multistep_sample}
budgets = (10, 20, 50, 100)  # network evaluations per sample

def on_grid(solver, grid):  # a sampler (pred, noise, steps, key)
    return lambda p, z, n, k: solver(p, z, grid(n))

def num_steps(s, g, b):  # Heun skips its second slope only at sigma = inf, 0:
    # b // 2 + 1 steps on the linear grid (b evaluations, the first level is
    # infinite), b // 2 on the EDM grids (b - 1)
    return (b // 2 + (g == 'linear')) if s == 'Heun' else b

fd_ode = {(s, g): [fd_runs(on_grid(solver, grid), num_steps(s, g, b))
                   for b in budgets]
          for s, solver in solvers.items() for g, grid in grids.items()}
print('evaluations            ', ''.join(f'{b:>11d}' for b in budgets))
for (s, g), row in fd_ode.items():
    print(f'{s + ", " + g:23s}', ''.join(f'{mean_range(v):>11s}' for v in row))
```

```
evaluations                       10          20          50         100
Euler, linear              3.15±0.09   1.86±0.09   1.36±0.08   1.29±0.07
Euler, EDM                 6.45±0.18   2.98±0.13   1.76±0.10   1.52±0.10
Euler, EDM, rho = 3        9.55±0.25   3.84±0.14   1.91±0.10   1.54±0.09
Heun, linear               1.55±0.04   1.22±0.03   1.14±0.05   1.19±0.06
Heun, EDM                 14.47±0.12   1.81±0.03   1.30±0.08   1.34±0.08
Heun, EDM, rho = 3        86.59±0.46 10.59±0.26   1.51±0.09   1.37±0.09
multistep, linear          1.97±0.07   1.32±0.06   1.22±0.07   1.25±0.07
multistep, EDM             4.23±0.15   1.87±0.10   1.41±0.09   1.37±0.09
multistep, EDM, rho = 3    6.46±0.20   2.26±0.10   1.38±0.08   1.35±0.08
```

2 つの表はどこで交わるだろうか。次のプロットは、最初の表のサンプラと線形グリッド上の 3 つのソルバを、ネットワーク評価回数という 1 つの横軸上にまとめたものである。

```
curves = {**{name: (steps, row) for name, row in fd_vp.items()},
          **{f'{s}, linear': (budgets, fd_ode[s, 'linear']) for s in solvers}}
d2l.plot([x for x, _ in curves.values()] + [[10, 1000]],
         [[np.mean(v) for v in row] for _, row in curves.values()]
         + [[np.mean(floors)] * 2], 'network evaluations', 'FD',
         legend=list(curves) + ['floor'], xscale='log', yscale='log',
         figsize=(6, 4), fmts=('o-', 's-', '^-', 'v--', 'D--', 'x-.', 'k:'))
```



2 次手法は、モデルの漸近的な FD プラトー（≈ 1.1〜1.3、有限サンプルのフロアより約 0.3 高い水準）に最も早く到達する。線形グリッド上のホイン法は 20 回の関数評価で ≈ 1.2〜1.3 に達し、2 次多段ソルバは 50 回までに到達するのに対し、オイラー法は 50〜100 回を要する。

ステップの配置はソルバの次数と同じくらい重要である。10 回の評価では、オイラー法のスコアは ≈ 3.2（線形グリッド）、≈ 5（VP グリッド）、≈ 6.5（$\rho = 7$ の EDM グリッド）である一方、ホイン法は線形グリッド上で ≈ 1.6 であるのに対し EDM 上では 12〜15 となる。ρ を 3 に下げてステップをデータ側から高ノイズ側へ移すと、10 回および 20 回評価での性能はさらに悪化する。$\bar{\sigma} \approx 4$（ピクセルの標準偏差 ≈ 0.7 の数倍）より上では、ノイズ画像はほぼ純粋なガウスノイズであり、軌跡もほぼ直線になる。線形グリッドは $\bar{\sigma} > 4$ の領域を 2 ステップ（$\infty \to 9 \to 4$）で通過するのに対し、10 ステップの EDM グリッドはそこに 4 ステップを費やし、衣類の構造が形成される中間領域に十分なステップを残せない。

17.5.2.5 積分誤差、ガイダンス、サンプラの選択

ここまでは FD のみでサンプラを評価してきたが、FD が評価するのはサンプル集合全体である。個々のサンプルが、同じノイズ画像から速度場を厳密に積分したときに到達する点からどれだけ離れているかは FD からはわからない。FD では捉えきれないソルバの誤差を見るため、すべてのノイズ画像を 200 回評価のホイン法で積分したものを参照解とし、それに対する各ソルバのピクセルあたりの二乗平均平方根（root mean square、RMS）距離を測定する。

ガイダンスは積分対象の速度場を変化させるため、この測定を 2 つの速度場に対して行う。分類器なしガイダンス（classifier-free guidance、17.4.4.2 節）はラベルなし予測からラベル付き予測へと $\tilde{\mathbf{u}} = \mathbf{u}_\emptyset + \gamma(\mathbf{u}_c - \mathbf{u}_\emptyset)$ のように外挿するため、$\gamma = 1$ が通常の条件付き速度場であり、ガイダンス付きの評価は 1 回につきネットワークの順伝播を 2 回要する。2 つの構成要素の外側へ外挿する速度場は軌跡に沿った変化も速くなるはずであり、$\gamma = 2$ ではどのソルバもそれに追随するためにより多くの評価回数を必要とすると予測される。

```
    def endpoints(solver, n, gamma):
        """Samples of a solver with n steps on the linear grid, per noise draw."""
        keys = jax.random.split(next(run_keys), len(noises))  # unused: no noise
        return [sample(on_grid(solver, lin.sigma_bars), n, gamma, ema, labels,
                       z, k) for z, k in zip(noises, keys)]

    for gamma in (1.0, 2.0):
        refs = endpoints(d2l.heun_sample, 101, gamma)    # 200 evaluations
        fds = [d2l.fid_score(features(x), f_test) for x in refs]
        print(f'gamma = {gamma:.0f}, reference: FD {mean_range(fds)}; '
              'FD and RMS distance to it')
        print('evaluations', ''.join(f'{b:>18d}' for b in (10, 20, 50)))
        for s, solver in solvers.items():
            cols = []
            for b in (10, 20, 50):
                xs = endpoints(solver, b // 2 + 1 if s == 'Heun' else b, gamma)
                fds = [d2l.fid_score(features(x), f_test) for x in xs]
                rms = np.mean([float(jnp.sqrt(((x - r) ** 2).mean()))
                               for x, r in zip(xs, refs)])
                cols.append(f'  {mean_range(fds)} {rms:.3f}')
            print(f'  {s:9s}', ''.join(cols))
```

```
gamma = 1, reference: FD  1.24±0.07; FD and RMS distance to it
evaluations                 10                  20                  50
  Euler        3.15±0.09 0.103   1.86±0.09 0.062   1.36±0.08 0.030
  Heun         1.55±0.04 0.117   1.22±0.03 0.037   1.14±0.05 0.013
  multistep    1.97±0.07 0.088   1.32±0.06 0.043   1.22±0.07 0.013
gamma = 2, reference: FD  2.12±0.12; FD and RMS distance to it
evaluations                 10                  20                  50
  Euler        4.01±0.17 0.099   3.09±0.17 0.059   2.47±0.14 0.028
  Heun         3.44±0.18 0.125   2.67±0.15 0.048   2.31±0.14 0.017
  multistep    3.32±0.17 0.086   2.64±0.16 0.042   2.23±0.13 0.014
```

2 つの指標は一致せず、その食い違いから FD の性質が見えてくる。ガイダンスなしの 10 回評価では、ホイン法のサンプルはオイラー法よりも参照解から離れている（約 0.1 に対して距離 0.12〜0.13）にもかかわらず、FD はオイラー法の半分（約 3.2 に対して約 1.6）である。FD は個々のサンプルの誤差ではなくサンプル集合の分布を評価するため、ソルバがあるノイズ画像に対して本来とは異なる衣類を生成しても、集合としては写実的なサンプル群になりうるのである。

予測どおり、ガイダンスを加えると速度場への追随は難しくなる。ガイダンスは写実性と引き換えにクラスとの整合性を高めるため、$\gamma = 2$ では参照解自体のスコアが約 2.1〜2.3 となる。20 回評価ではどのソルバもまだ参照解より 0.4〜1 高く、50 回までに多段サンプラは参照解の範囲内またはその端に達する一方、ホイン法は 0.15〜0.2、オイラー法は約 0.3 上回ったままである。参照解までの距離は、この予測を部分的にのみ裏付ける。20 回評価におけるホイン法の参照解からの距離はガイダンスなしより 3 分の 1 から半分大きくなるが、オイラー法と多段サンプラの変化は 6 分の 1 以下にとどまり、10 回および 20 回評価では 3 手法の中で多段サンプラが最も参照解に近い。したがってガイダンスのもとでは、50 回評価（ネットワーク順伝播 100 回）までにプラトーへ到達するのは多段サンプラだけであり、ホイン法とオイラー法はさらに多くの評価を必要とする（何回必要かはこの表からはわからない）。

積分誤差は視覚的にも確認できる。以下に示すのは、同じ 8 枚のノイズ画像とラベルから 4 つのサンプラ、すなわち 50 ステップの祖先連鎖、10 ステップおよび 50 ステップの DDIM、そして線形グリッド上で 20 回評価したホイン法によって生成した各 8 枚の画像である。

```
z8 = noises[0][:8]
pred8 = d2l.make_pred(d2l.conditional(ema, labels[:8]), lin, 'velocity')
rows = [d2l.ancestral_sample(pred8, z8, vp, 50, next(run_keys)),
        d2l.ddim_sample(pred8, z8, vp, 10), d2l.ddim_sample(pred8, z8, vp, 50),
        d2l.heun_sample(pred8, z8, lin.sigma_bars(11))]
d2l.show_gray(jnp.concatenate(rows).clip(-1, 1), 4, 8);
```

決定論的サンプラはノイズから画像への 1 つの写像を近似するため、決定論的な 3 つの行は列ごとに揃っている。10 ステップでは、50 ステップとホイン法で一致して描かれる柄や輪郭などの細部が平滑化されて消えている。祖先連鎖の行はざらついており、背景や生地に斑点状のノイズが見られる。これが最初の表で確認した過剰な分散を可視化したものである。

では、未測定のモデルに対してはどのようにサンプラを選べばよいだろうか。ノイズ画像の集合と評価スコアを固定し、モデル自身の訓練経路のグリッド上で決定論的ソルバから始めて、スコアの変化が実行間の変動幅以下に収まるまで評価回数を 2 倍ずつ増やしていけばよい。最後に残るのはモデル自体の誤差であり、サンプラにできる最善のことはサンプラ由来の誤差を上乗せしないことである。このモデルでは、プロットの曲線が平坦になる箇所、すなわちホイン法で約 20 回の評価で手順が停止し、新たなノイズが役立つことはない。ガイダンスのもとでは停止が遅くなり、多段サンプラの約 50 回評価となる。多段サンプラはステップあたり 1 回の評価で済み、積分誤差も最小であるため、ガイダンス下ではより優れたデフォルトとなる。論文から持ってきたグリッドで訓練グリッドを置き換える前にも、同じテストを通すべきである。10 回評価ではどのソルバもプラトーに達しておらず、1〜2 ステップでプラトーに到達するには別の種類のモデルが必要になる（17.8 節）。

17.5.3 逆変換と編集

決定論的サンプラは微分方程式を積分するものであり、微分方程式はどちらの向きにも実行できる。これを逆向きに実行すると、各画像に対してノイズ画像、すなわちその画像の**コード**（code）が得られ、サンプラはそのコードから元の画像を再現する。コードを使えば画像間の補間が可能になる。さらに同じサンプラを、部分的にノイズを加えた画像から開始したり各ステップで制約を課したりすることで、再訓練なしに編集やインペインティングを行える。

17.5.3.1 潜在コード

`ddim_invert` は決定論的サンプラを上向きに、すなわち $t = 0$ の画像からノイズへ向けて実行する。各ステップは次のオイラーステップによってレベル t から次のレベル $t' > t$ へと進む。

$$\bar{\mathbf{x}}_{t'} = \bar{\mathbf{x}}_t + (\bar{\sigma}_{t'} - \bar{\sigma}_t)\,\hat{\epsilon}(\bar{\mathbf{x}}_t, \bar{\sigma}_t), \tag{17.41}$$

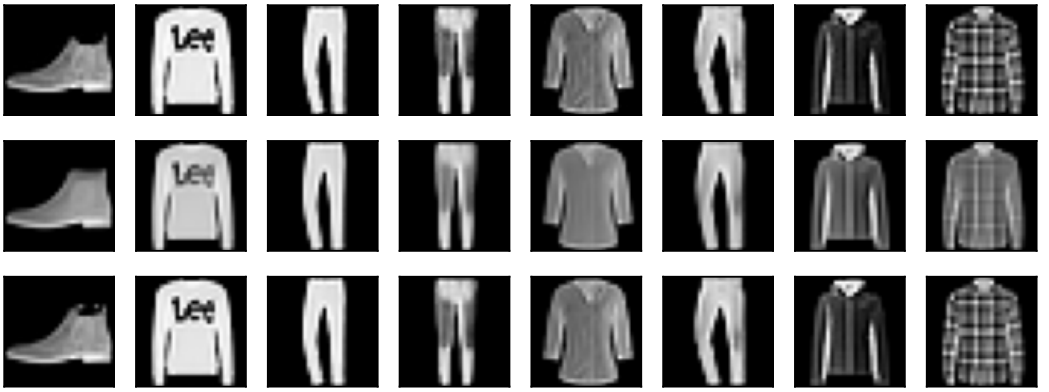
ここでノイズ推定値は下端、すなわちエンコーダが手にしている唯一の点で評価される。予測器が定義されない $\bar{\sigma} = 0$ からの最初のステップでは、代わりに次のレベルのノイズレベルを使って画像を評価する。

```
def ddim_invert(pred, x, path, steps, t_end=1.0):
    """Deterministic DDIM run backwards, from data x at t = 0 to t_end."""
    a, s = path.grid(steps, 0.0, t_end)
    j = np.maximum(np.arange(steps), 1)  # at t = 0 x is read at the next level
    def step(x, inputs):
        a_t, s_t, a_s, s_s, a_j, s_j = inputs
        eps = pred(x, a_j, s_j)[1]
        return a_s * (x - s_t * eps) / a_t + s_s * eps, None
    return jax.lax.scan(step, x, (a[:-1], s[:-1], a[1:], s[1:], a[j], s[j]))[0]
```

デコーダは上端で評価した $\hat{\epsilon}(\bar{\mathbf{x}}_{t'}, \bar{\sigma}_{t'})$ を使って再び下向きにステップを進める。2 つの推定値は 1 ステップにわたる $\hat{\epsilon}$ の変化分だけ異なるため、厳密な速度場を用いたとしてもこの往復は厳密には元に戻らず、その誤差は他の 1 次誤差と同様にステップサイズとともに縮小するはずである。往復では 1 つの速度場を両方向に実行するため、速度場自体が正しいかどうかにかかわらず、生じる誤差はその速度場上でのステップの誤差となる。ただし学習された速度場では、厳密な速度場よりも軌跡に沿って速く変化することがあり、その場合、往復誤差の収束は厳密な速度場を通す場合よりも遅くなる。そこで、それぞれ自身のラベルを持つ 100 枚のテスト画像を線形経路上の 10〜200 ステップでエンコード・デコードし、あわせて混合分布の 20,000 個の点も厳密な速度場による同じ往復に通してみる。

```
x0 = X_test[:100]
pred = d2l.make_pred(d2l.conditional(ema, y_test[:100]), lin, 'velocity')
codes = {n: d2l.ddim_invert(pred, x0, lin, n) for n in (10, 20, 50, 100, 200)}
recon = {n: d2l.ddim_sample(pred, code, lin, n) for n, code in codes.items()}
for n in codes:  # the mixture's points through its exact field, for contrast
    back = d2l.ddim_sample(pred_exact, d2l.ddim_invert(pred_exact, data, lin, n),
                           lin, n)
    print(f'{n:3d} steps: squared error per pixel '
          f'{((recon[n] - x0) ** 2).mean():.4f}, code std {codes[n].std():.2f}; '
          f'mixture {((back - data) ** 2).mean():.4f}')
d2l.show_gray(jnp.concatenate([x0[:8], recon[10][:8], recon[50][:8]]), 3, 8);
```

```
 10 steps: squared error per pixel 0.0349, code std 0.69; mixture 0.3113
 20 steps: squared error per pixel 0.0127, code std 0.74; mixture 0.0960
 50 steps: squared error per pixel 0.0028, code std 0.79; mixture 0.0189
100 steps: squared error per pixel 0.0008, code std 0.82; mixture 0.0067
200 steps: squared error per pixel 0.0002, code std 0.84; mixture 0.0026
```



どちらの往復も厳密には一致せず、どちらの誤差も着実に減少する。10 ステップから 100 ステップの間で二乗平均平方根誤差は両方とも約 6 分の 1 に縮小し（1 次誤差の 10 分の 1 には及ばない）、50 ステップ以降では画像の誤差は少なくとも混合分布と同程度以上の速さで減少する。したがって、学習された速度場は軌跡に沿って厳密な速度場より荒れているわけではなく、残っているのはステップ自体の誤差である。50 ステップにおける誤差はピクセルあたり 0.05〜0.06（-1 から 1 のスケール上）であり、3 行目の再構成画像は元画像に近い。10 ステップでは約 0.19 となり、2 行目では右端のシャツのチェック柄などの細部が失われている。

一方、得られたコードは満足のいくものではない。その標準偏差は 10 ステップで約 0.7、200 ステップでもなお 0.9 未満にとどまるため、画像を再現するコードはサンプラが本来開始点として想定しているノイズよりも幅が狭い。したがって、このコードは標準ガウスノイズからの典型的な抽出サンプルではなく、コード間を補間したりその近傍から新しいコードを抽出したりする際には注意が必要である。

実際の写真をそのコード経由で編集する手法は、これとは別の 2 つ目の困難に直面する。逆変換はある 1 つの速度場の軌跡をたどるのに対し、ガイダンス付きサンプラはそれとは異なる速度場を積分するため、コードからガイダンス付きサンプリングを行うとその軌跡から外れ、元の画像を再現できなくなるのである。ヌルテキスト逆変換（null-text inversion）(Mokady et al. 2023) は、ガイダンス付きサンプラがコードから元画像を再現するよう、各ステップでヌルラベルの埋め込みを最適化することでこの問題を修復する。

サンプラはコードの連続関数であるため、2 枚の画像のコード間を補間することで画像間を連続的に移動できる。しかし、直線は適切な経路ではない。等しいノルム r を持ち角度 ω をなす 2 つのコードの中点ではノルムが $r\cos(\omega/2)$ となり、独立なノイズ画像のようにコードどうしがほぼ直交する場合には約 $\sqrt{2}$ 分の 1 に縮んでしまうが、デコーダはそのように短いコードから開始されたことがない。そこで球面補間（spherical interpolation）では、代わりに 2 つのコードを結ぶ大円に沿って次のように移動する。

$$\mathbf{z}_u = \frac{\sin\left((1-u)\omega\right)\mathbf{z}_0 + \sin(u\omega)\,\mathbf{z}_1}{\sin\omega},$$
$$\cos\omega = \frac{\mathbf{z}_0^\top \mathbf{z}_1}{\|\mathbf{z}_0\|\|\mathbf{z}_1\|},$$

これにより両端点のノルムが保たれる。テスト画像の中からアンクルブーツ 2 枚の 50 ステップのコード間にある 8 点をデコードしてみよう。行全体が同一クラスであるため、デコーダには 1 つのラベルを与えられる。

```
i, j = jnp.nonzero(y_test[:100] == y_test[0])[0][:2]  # one class: one field
z0, z1 = codes[50][i], codes[50][j]
omega = jnp.arccos(jnp.clip(
    (z0 * z1).sum() / (jnp.linalg.norm(z0) * jnp.linalg.norm(z1)), -1, 1))
u = jnp.linspace(0, 1, 8).reshape(-1, 1, 1, 1)
zs = (jnp.sin((1 - u) * omega) * z0 + jnp.sin(u * omega) * z1
      ) / jnp.sin(omega)
pred_i = d2l.make_pred(d2l.conditional(ema, jnp.full(8, y_test[i])), lin,
                       'velocity')
d2l.show_gray(d2l.ddim_sample(pred_i, zs, lin, 50), 1, 8);
```

形状は一方のブーツからもう一方へと徐々に変化するが、中間のフレームは第 3 のブーツというより 2 つのブーツを重ね合わせたように見える。球面補間はコードを適切なノルムに保つものの、2 つのブーツの間にあるコードをデコーダが第 3 のブーツへと写す保証はどこにもないのである。

17.5.3.2 部分ノイズ付加による編集

逆変換は、画像を厳密に再現するノイズを求める操作であった。編集において欲しいのはその逆、すなわち入力の細部を忘れつつ大まかな構図だけを保つ開始点である。そこで C. Meng et al. (2022) は、スケッチや粗いコラージュのようなラフな入力 $\mathbf{y}$ に直接新たなノイズを加えて中間時刻 t_0 の状態 $\mathbf{x}_{t_0} = \alpha_{t_0}\mathbf{y} + \sigma_{t_0}\epsilon$ を作り、t_0 からデータ側へとサンプラを実行する。ノイズが実画像にはない入力の不自然な細部を覆い隠し、サンプラがそれを実画像らしい細部に置き換え、大まかな構図だけが生き残る。時刻 t_0 は入力への忠実度（fidelity）と写実性のバランスを決める。このバランスはどれほど良好になりうるだろうか。

ここではラフな入力として、実際のどの衣類にも似ていないぼやけたシルエットである 10 個のクラス平均画像を用い、それぞれに 100 通りずつ $t_0 \in \{0.3, 0.5, 0.7, 1\}$ までノイズを加えて、対応するクラスラベルのもとで 25 回の DDIM ステップにより逆向きに積分する。入力に対するピクセルあたりの二乗距離（忠実度）と 1,000 枚の出力の FD（写実性）を出力する。なお、$t_0 = 0$ の行は入力画像そのものである。

```
means = jnp.stack([X[y == c].mean(0) for c in range(10)])
inputs = jnp.tile(means, (100, 1, 1, 1))            # 1,000 inputs, class i % 10
pred = d2l.make_pred(d2l.conditional(ema, labels), lin, 'velocity')
eps = jax.random.normal(jax.random.key(2), inputs.shape)
edits = {0.0: inputs}
for t0 in (0.3, 0.5, 0.7, 1.0):
    x_t0 = lin.sample(inputs, jnp.full((1000,), t0), eps)
    edits[t0] = d2l.ddim_sample(pred, x_t0, lin, 25, t_start=t0).clip(-1, 1)
for t0, x in edits.items():
    print(f't0 = {t0:.1f}: squared distance to the input '
          f'{((x - inputs) ** 2).mean():.3f}, '
          f'FD {d2l.fid_score(features(x), f_test):.2f}')
d2l.show_gray(jnp.concatenate([x[:10] for x in edits.values()]), 5, 10);
```

```
t0 = 0.0: squared distance to the input 0.000, FD 87.03
```

```
t0 = 0.3: squared distance to the input 0.019, FD 29.37
t0 = 0.5: squared distance to the input 0.047, FD 11.31
t0 = 0.7: squared distance to the input 0.088, FD 3.64
t0 = 1.0: squared distance to the input 0.169, FD 1.67
```

入力の FD は 85〜87 であり、t_0 が大きくなるにつれて入力からの距離は増大し、FD は低下する。$t_0 = 0.3$ では出力は平均画像より鮮明になるもののむらがあり、約 30 という FD は 10 ステップの祖先連鎖と同水準で、どの決定論的サンプラにも遠く及ばない。0.5 では奇妙な質感を持つ衣類のように見える（FD は 11〜12）。0.7 になってようやく FD が約 4 となって 10 ステップの決定論的サンプラに近づくが、その時点では入力から生き残っているのは輪郭と姿勢だけである。$t_0 = 1$ では入力は完全に消え去り、各出力はそのクラスの新たなサンプルとなって FD は 2 を下回る。また、同じノイズ画像から開始しているため、0.7 と 1 の行はしばしば似た見た目になる。クラス平均のように粗い入力に対しては、入力がほとんど残らないほどの t_0 に達して初めて写実性が得られるのである。

17.5.3.3 インペインティング

画像の一部（たとえば上半分）が既知であり、残りの部分をモデルに埋めさせたいとしよう。既知のピクセルで 1 をとるマスクを $\mathbf{m}$、そのピクセル値を $\mathbf{y}$ と書く。どのレベルにおいても、$\mathbf{x}_s$ の既知ピクセルがどのような姿であるべきかはわかっており、それは $\alpha_s\mathbf{y}$ にスケール σ_s のノイズを加えたものである。そこで置換インペインティング（replacement inpainting）(Y. Song, Sohl-Dickstein, et al. 2021; Lugmayr et al. 2022) は、通常どおりサンプラの各ステップを進めたあと、既知ピクセルを新しいレベルで新たにノイズを加えた $\mathbf{y}$ のコピーによって次のように上書きする。

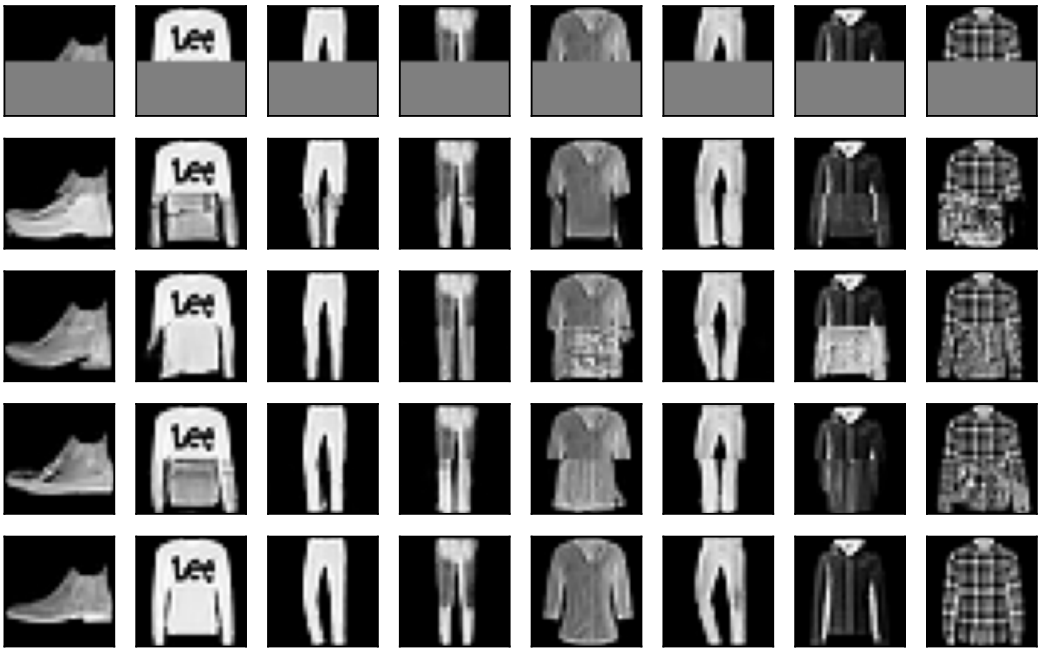
$$\mathbf{x}_s \leftarrow \mathbf{m} \odot \left(\alpha_s\mathbf{y} + \sigma_s\xi\right) + (\mathbf{1} - \mathbf{m}) \odot \mathbf{x}_s. \tag{17.42}$$

この置換は近似である。各ステップは既知ピクセルも自由であるかのように全ピクセルを動かし、そのあとステップがどこへ動かしたかとは無関係なノイズで既知ピクセルを上書きする。未知ピクセルは、ネットワークが次にノイズ画像を見るときに初めて既知ピクセルの情報を知るため、サンプラは既知ピクセルを条件とする正確な分布から未知ピクセルを抽出しているわけではない。`inpaint` はこの上書きを挟めるよう、サンプラを 1 ステップずつ実行する。

```
def inpaint(pred, known, mask, path, steps, key):
    """DDIM from noise that resets the known pixels (mask 1) at each level."""
    a, s = path.grid(steps)
    def step(x, inputs):
        a_t, s_t, a_s, s_s, key = inputs
        x0, eps = pred(x, a_t, s_t)
        x = a_s * x0 + s_s * eps                      # one DDIM step, eta = 0
        noised = a_s * known + s_s * jax.random.normal(key, x.shape)
        return mask * noised + (1 - mask) * x, None   # keep the known pixels
    keys = jax.random.split(key, steps + 1)
    x = jax.random.normal(keys[0], known.shape)
    return jax.lax.scan(step, x, (a[:-1], s[:-1], a[1:], s[1:], keys[1:]))[0]
```

8 枚のテスト画像の上半分を保持し、それぞれの画像のラベルのもとで異なるノイズから 3 通りずつ補完画像を生成する。各行は上から既知の上半分、3 通りの補完画像、そして元画像を示している。

```
mask = jnp.zeros((1, 28, 28, 1)).at[:, :14].set(1)  # the upper half is known
pred = d2l.make_pred(d2l.conditional(ema, jnp.tile(y_test[:8], 3)), lin,
                     'velocity')
fills = inpaint(pred, jnp.tile(X_test[:8], (3, 1, 1, 1)), mask, lin, 50,
                jax.random.key(3))
d2l.show_gray(jnp.concatenate([X_test[:8] * mask, fills, X_test[:8]]), 5, 8);
```



大半の補完画像は既知の上半分を自然に引き継いでそのクラスらしい衣類を形作っており、1 枚の画像に対する 3 通りの補完結果も互いに異なっている。一方、近似の影響は継ぎ目に現れている。いくつかの補完画像（特にシャツで顕著）では、下半分が上半分とは異なる質感や濃淡になっている（演習問題 5 でこれを測定し修復する）。

本節のすべてのサンプラは同じ訓練済み速度場を積分しており、違いはどのように積分するかという点だけであった。速度場そのものについてはここまで無条件に信頼してきたが、それは選んだターゲットに対する重みなし二乗誤差から得られたものにすぎない。次節では、なぜこの損失が適切なのか、そしてターゲット、経路、重み付けの選択が何を変えるのかを問う。

17.5.4 まとめ

訓練は $\mathbf{x}_t$ の各時刻 t における周辺分布のみに依存するため、1 つの訓練済みネットワークによって、確率性の強度 $\eta \geq 0$ でパラメータ化された DDIM サンプラ族の全体をサポートできる。$\eta = 0$ では DDIM は決定論的になり、確率フロー ODE $d\bar{\mathbf{x}}/d\bar{\sigma} = \hat{\epsilon}$（再スケーリングされた座標系 $(\bar{\mathbf{x}}, \bar{\sigma}) = (\mathbf{x}/\alpha_t, \sigma_t/\alpha_t)$）上のオイラー法と一致する。本節の Fashion-MNIST モデルでは、少ない予算および中程度の予算において決定論的 ODE ソルバが一貫して確率的サンプラ（$\eta = 1$ および祖先連鎖）を上回り、線形グリッド上の 2 次ホイン法は約 20 回の評価でモデルの FD プラトーに到達し、2 次のアダムス・バッシュフォース多段ソルバはステップあたり 1 回の評価のまま 50 回の評価までにガイダンス下のプラトーに到達する。決定論的 ODE を逆向きに実行する（`ddim_invert`）と任意の画像が潜在ノイズコードへと写され、再訓練なしに球面潜在補間、部分ノイズ付加による画像編集、および置換インペインティングを実現できる。

17.5.5 演習問題

1. **順ステップとマルコフ的なメンバー。**●$\mathbf{x}_0$ が与えられたもとで、$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$ と、$\mathbf{x}_s$ を Equation 17.34 から同じ ϵ を用いて抽出する。
   1. 同時ガウス分布を条件付けることで、$\mathbf{x}_t$ の（$\mathbf{x}_s$ と $\mathbf{x}_0$ が与えられたもとでの）平均と共分散を計算せよ。
   2. 平均が $\mathbf{x}_0$ に依存しないのは $\tilde{\sigma}$ が $\eta = 1$ における値をとるときに限ること、またそのとき条件付き分布が順ステップ $\mathbf{x}_t = (\alpha_t/\alpha_s)\mathbf{x}_s + \sqrt{\sigma_t^2 - (\alpha_t/\alpha_s)^2 \sigma_s^2}\, \nu$（17.2 節のノイズ付加連鎖）に一致することを示し、この連鎖が順過程がマルコフ的になる族の唯一のメンバーであると結論せよ。
   3. すべてのステップで $\tilde{\sigma} = 0$ とすると、順過程が $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$（1 つの ϵ をすべての時刻で共有）になることを示せ。なぜ回帰ではこれらの順過程を区別できないのか。
2. **ガウスデータ。**データを分布 $\mathcal{N}(0, v)$ に従う 1 次元データとする。
   1. 座標 $(\bar{x}, \bar{\sigma})$ において、厳密なノイズ推定値が $\hat{\epsilon} = \bar{\sigma}\bar{x}/(v + \bar{\sigma}^2)$ であること、$\bar{\sigma}$ から $\bar{\sigma}'$ への 1 回の決定論的ストライドが $\bar{x}$ に $(v + \bar{\sigma}\bar{\sigma}')/(v + \bar{\sigma}^2)$ を掛けること、および厳密なフローがそれに $\sqrt{(v + \bar{\sigma}'^2)/(v + \bar{\sigma}^2)}$ を掛けることを示せ。
   2. ストライドの倍率が厳密な倍率を決して超えず、等号成立は $\bar{\sigma}' = \bar{\sigma}$ のときに限ることを示せ。このことは、任意のグリッド上でのガウスデータの決定論的サンプルの分散について何を意味するか。
   3. 次に、Equation 17.35 の族から $0 \leq \eta \leq 1$ かつ厳密な推定値を持つ任意のメンバーをとる。x_s の分散（$x_t \sim \mathcal{N}(0, \alpha_t^2 v + \sigma_t^2)$ から 1 ステップ進んだ後）を $c = \sqrt{\sigma_s^2 - \tilde{\sigma}^2}$ の関数として表し、コーシー・シュワルツの不等式を用いてそれが厳密な $\alpha_s^2 v + \sigma_s^2$ を下回り、かつ η が大きくなるにつれて減少することを示せ。これは本節で出力した広がりのうち、どれを説明するものか。
   4. $v = 0.25$（`mix` の 1 つのモードの座標あたりの分散）とする。決定論的サンプルの標準偏差 s を、$x_1 \sim \mathcal{N}(0, 1)$ から始めて 200、50、20、10、5 ステップの分散保持グリッド上で計算せよ。座標あたりこの標準偏差を持つ 2 次元点群の広がりである $2s^2$ を、データの 0.5 および本節で出力した広がりと比較せよ。ガウスモードは、混合分布の収縮と同程度を予測するか、それより大きな収縮を予測するか、それとも小さな収縮を予測するか。
3. [extended] **対数信号対雑音比における 2 次サンプラ。**●●DPM-Solver-2 (Lu, Zhou, et al.

2022) は、$\bar{\sigma}$ から $\bar{\sigma}'$ へのステップを 2 回の評価で行い、2 回目の評価を中点 $\bar{\sigma}_m = \sqrt{\bar{\sigma}\bar{\sigma}'}$ ($\log \bar{\sigma}$ 上の中点) にとる。すなわち、$\bar{\mathbf{x}}_m = \bar{\mathbf{x}} + (\bar{\sigma}_m - \bar{\sigma})\,\hat{\epsilon}(\bar{\mathbf{x}}, \bar{\sigma})$ および $\bar{\mathbf{x}}' = \bar{\mathbf{x}} + (\bar{\sigma}' - \bar{\sigma})\,\hat{\epsilon}(\bar{\mathbf{x}}_m, \bar{\sigma}_m)$ である。

1. $h = \log(\bar{\sigma}/\bar{\sigma}')$ とおく。1 ステップにわたる $\bar{\mathbf{x}}$ の厳密な変化量を軌跡に沿った $\hat{\epsilon}$ の積分として表し、この更新の局所誤差が $O(\bar{\sigma}h^3)$ である一方、オイラーステップの局所誤差が $O(\bar{\sigma}h^2)$ であることを示せ。ただし、$\hat{\epsilon}$ は軌跡に沿って $\log \bar{\sigma}$ の滑らかな関数であり、$\mathbf{x} = \bar{\mathbf{x}}/\sqrt{1+\bar{\sigma}^2}$ に関してリプシッツ連続である (したがって大きな $\bar{\sigma}$ において $\bar{\mathbf{x}}$ に関するリプシッツ定数が $1/\bar{\sigma}$ のオーダーになる) ものと仮定せよ。
2. インターフェース `pred(x, alpha, sigma)` と関係式 $\alpha = 1/\sqrt{1+\bar{\sigma}^2}$、$\sigma = \bar{\sigma}\alpha$ を用いてこの更新を実装し、$\bar{\sigma} = 0$ への最後のステップはオイラー法で進めよ。これを `ode_sample`、`heun_sample`、`multistep_sample` と約 11、21、51 回の評価回数で比較せよ。比較には、`pred_exact` を用いた混合分布上での 1,000 ステップ参照解までの距離と、画像上での FD を用いよ。
3. $\log \bar{\sigma}$ 上で 80 から 0.002 の間に等間隔に配置したレベルを用いて同じ実験を繰り返せ。各予算において、手法とグリッドのどの組み合わせが最良か。

4. **最適な逆分散。**●t から s への確率的ステップは、事後平均 $\alpha_s \hat{\mathbf{x}}_0 + \sigma_s \kappa \hat{\epsilon}$ へ移動し、座標あたり分散 v のガウスノイズを加える。Ho et al. (2020) は順ステップ分散と事後分散を試し、Bao et al. (2022) は任意のデータに対する最良の v を導出した。厳密な推定値を仮定し、$\bar{e}_t = \mathbb{E}\|\epsilon - \hat{\epsilon}\|^2/d$ を時刻 t (d 次元空間) における厳密なノイズ推定値の座標あたり誤差とする。
   1. 演習問題 3a (17.3 節) より、真の逆遷移 $q(\mathbf{x}_s \mid \mathbf{x}_t)$ の共分散は $\tau^2\mathbf{I} + c^2\mathrm{Cov}(\mathbf{x}_0 \mid \mathbf{x}_t)$ となる。ここで c は $\mathbf{x}_0$ の係数 ($q(\mathbf{x}_s \mid \mathbf{x}_t, \mathbf{x}_0)$ の平均における係数) である。その平均を $\alpha_s \mathbf{x}_0 + \sigma_s \kappa \epsilon$ (ただし $\epsilon = (\mathbf{x}_t - \alpha_t \mathbf{x}_0)/\sigma_t$) と表すことで、$\tau^2 = \tilde{\sigma}^2$ (事後分散) および $c = k = \alpha_s - \sigma_s \kappa\, \alpha_t/\sigma_t$ となることを導け。真の逆遷移の平均は何か。
   2. 分散 v のうち、$q(\mathbf{x}_s \mid \mathbf{x}_t)$ からガウスステップへの期待 KL ダイバージェンスを最小化するものが $v^\star = \tilde{\sigma}^2 + k^2\bar{\sigma}_t^2\bar{e}_t$ であり、かつ $\bar{e}_t = 1 - \sigma_t^2\,\mathbb{E}\|\nabla \log p_t(\mathbf{x}_t)\|^2/d$ が成り立つため $v^\star$ がスコア関数のみから定まることを示せ。
   3. $v^\star$ が、データが 1 点のときは事後分散になり、分散保持経路上の標準ガウスデータのときは順ステップ分散になることを示せ。
   4. (任意・コード課題) $\bar{e}_t$ を、混合分布上で `pred_exact` を用いて 50 ステップの分散保持グリッド上で推定し、3 種類の分散それぞれを用いた確率的ステップの 20,000 サンプルの広がりを比較せよ。
5. [code] **調和化を伴うインペインティング。**●`inpaint` による補完画像は、境界に沿って既知の半分と食い違うことがある。
   1. 継ぎ目を測定せよ。すなわち、補完画像の 13 行目と 14 行目の平均絶対差を計算し、実際のテスト画像における同じ量と比較せよ。
   2. RePaint (Lugmayr et al. 2022) は上下両半分が情報を交換できるようにするため、t から s への各ステップの後に順ステップで結果を再び t までノイズ付加し、そのステップをレベルあたり r 回繰り返す。これを $r \in \{1, 5, 10\}$、50 レベルで実装し、継ぎ目を再度測定せよ。各反復にはどれだけのコストがかかるか。
   3. 各 r について、1 枚のテスト画像の補完結果を 5 通り生成せよ。それらはどの程度多様か。
6. [code] **別クラスへ向けた編集。**本節のクラス条件付きモデルを使用し、`d2l.make_pred(d2l.cfg(ema, c, gamma), lin, 'velocity')` によって誘導せよ (ここで `c` は画像ごとに 1 つのラベルを保持する)。
   1. ズボンのテスト画像 100 枚を $t_0 \in \{0.3, 0.5, 0.7, 0.9\}$ までノイズ付加し、ドレスのラベルへ向けて 25 回の DDIM ステップで逆向きに積分せよ ($\gamma = 1$ および $\gamma = 2$ の両方で実行せよ)。編集のコードセルと同様に、`cnn` がドレスと分類する出力の割合と、入力に対するピクセルあたりの二乗距離を報告せよ。
   2. 純粋なノイズから始め、ズボンのラベルかつ $\gamma = 1$ で $t = 1$ から t^* までサンプリ

ングし、続けてドレスのラベルかつ (a) の 2 つの強度で $t = 0$ までサンプリングせよ（`ddim_sample` は `t_start` と `t_end` を受け取る）。これを $t^* \in \{0.3, 0.5, 0.7, 0.9\}$ について行い、ドレスと分類される割合を t^* に対してプロットし、(a) の t_0 に対する曲線の横に並べて、混合分布における演習問題 4（17.3 節）のコミットメント時刻と両者を比較せよ。

3. 厳密なネットワークを用いた場合、同じ γ かつ $t_0 = t^*$ における 2 つの実験が同じ分布に従う出力を生成することを示せ。曲線間にはどのような違いが残り、それはどこから生じるのか。

小問 (a) は Berkeley CS180/280A の Project 5A（2025 年秋学期）1.7.3 節を改変したものである。

17.6 尤度、重み付け、設計空間

これまでの節では、ノイズや速度をターゲットとする重みなしの二乗誤差によって拡散モデル（diffusion model）やフローマッチングモデルを訓練し、生成サンプルによって評価してきた。第 16 章の敵対的モデルとは異なり、拡散モデルはデータ点に対して確率密度も割り当てる。すなわち、Sohl-Dickstein et al. (2015) と Ho et al. (2020) は、逆方向マルコフ連鎖が潜在変数モデル（latent-variable model）を定め、そのエビデンス下界（evidence lower bound、ELBO）が各ノイズレベルにおけるノイズ予測回帰の重み付き和に分解されることを示した。一方、確率フロー ODE（probability-flow ODE）は瞬間的な変数変換（instantaneous change of variables）の公式を通じて厳密な対数尤度を定める (Y. Song, Sohl-Dickstein, et al. 2021)。

さらに、Kingma et al. (2021) と Kingma and Gao (2023) は、対数信号対雑音比（log signal-to-noise ratio、log-SNR）$\lambda_t = \log(\alpha_t^2/\sigma_t^2)$ を用いて書き直すと、標準的な拡散モデルおよびフローマッチングの目的関数がすべて、関数 $w(\lambda)$ で重み付けされたノイズ予測誤差の同じ積分に帰着することを示した。この損失の重み付けと並んで、Karras et al. (2022)（EDM）は入出力の前処理（preconditioning）と訓練時のノイズレベル分布を独立した設計軸として特定した。本節では、離散時間および連続時間の ELBO を導出し、3 モード混合分布上で厳密なフロー尤度と境界を比較し、既存の目的関数が暗黙に定める対数信号対雑音比の重み付け $w(\lambda)$ によってそれらを統一したうえで、Fashion-MNIST 上で統制された 1 要因アブレーション実験を実施する。

17.6.1 尤度境界

まず 17.2 節の連鎖を生成モデルとして書き下し、その尤度の境界を求めよう。すると、ノイズ回帰が自然に現れる。

17.6.1.1 潜在変数モデルとしての連鎖

順方向連鎖のステップ k は、現在の点を縮小してノイズを加える $\mathbf{x}_k = \sqrt{1-\beta_k}\,\mathbf{x}_{k-1} + \sqrt{\beta_k}\,\epsilon_k$ であり、各ステップを合成すると 1 回のガウス抽出 $\mathbf{x}_k = \alpha_k \mathbf{x}_0 + \sigma_k \epsilon$（ただし $\alpha_k^2 = \prod_{j \le k}(1-\beta_j)$、$\sigma_k^2 = 1 - \alpha_k^2$）になる。これは時刻 $t_k = k/T$ における VP 経路にほかならない。この順方向連鎖の分布を q と書く。生成モデルは、学習されたガウスステップによってこれを逆向きにたどる。

$$
\begin{aligned}
p_\theta(\mathbf{x}_{0:T}) &= p(\mathbf{x}_T) \prod_{k=1}^{T} p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k), \\
p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k) &= \mathcal{N}\big(\mu_\theta(\mathbf{x}_k, k),\, \nu_k \mathbf{I}\big),
\end{aligned}
\tag{17.43}
$$

ここで、$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ であり、平均 μ_θ はネットワークで計算し、分散 ν_k は固定とする（σ_k は引き続き経路のノイズスケールを表す）。

$\mathbf{x}_0$ の密度は、この同時分布を潜在変数 $\mathbf{x}_{1:T}$ について積分したものであるが、これはデータ空間の T 個の直積上の積分であり直接計算できない。しかし、潜在変数上の任意の分布 $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ を用いれば、この積分は期待値 $p_\theta(\mathbf{x}_0) = \mathbb{E}_q\big[p_\theta(\mathbf{x}_{0:T})/q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\big]$ に書き換えられる。対数関数は凹関数であるため、平均の対数は対数の平均以上となる（イェンセンの不等式）。これより、負の対数尤度に対して次の上界が得られる。

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &= -\log \mathbb{E}_q\left[\frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] \\
&\le \mathbb{E}_{q(\mathbf{x}_{1:T}\mid\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] =: L(\mathbf{x}_0).
\end{aligned}
\tag{17.44}
$$

これは変分推論（variational inference、36.7 節）における**エビデンス下界**（ELBO）の符号を反転させたものである。多くの潜在変数モデルでは真の事後分布に近づくよう q を学習する必要があるが、ここでは順方向連鎖がそれを与えるため、学習すべきものは何もない。不等式のギャップは $D_{\mathrm{KL}}\big(q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{1:T} \mid \mathbf{x}_0)\big)$ であり、$\mathbf{x}_0$ が与えられたもとで学習された逆方向連鎖が順方向連鎖を正確に逆向きにたどるとき、境界はタイトになる。データ全体で平均をとると、この境界は少なくとも交差エントロピー $\mathbb{E}_p[-\log p_\theta(\mathbf{x}_0)] = H(p) + D_{\mathrm{KL}}(p \,\|\, p_\theta)$ 以上となるため、モード欠落を起こすモデルは、そのモードに属するすべてのデータ点で大きなペナルティを受ける。

2 つの連鎖を比較するため、それぞれの分解、すなわち p_θ に対する上記の積と $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_k q(\mathbf{x}_k \mid \mathbf{x}_{k-1})$ を代入すると、境界はステップごとの項に分解される。

$$
L(\mathbf{x}_0) = \mathbb{E}_q\Big[-\log p(\mathbf{x}_T) - \sum_{k=1}^{T} \log \frac{p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k)}{q(\mathbf{x}_k \mid \mathbf{x}_{k-1})}\Big]. \tag{17.45}
$$

第 k 項は、学習されたステップが $\mathbf{x}_k$ から $\mathbf{x}_{k-1}$ へ進むのに対し、順方向ステップは逆向きに進むため、このままではまだダイバージェンスとして読めない。そこでベイズの定理を使って順方向ステップの向きを反転させる。マルコフ性（Markov property）により、さらに $\mathbf{x}_0$ で条件付けてもこの分布は変わらないため、次式が成り立つ。

$$
\begin{aligned}
q(\mathbf{x}_k \mid \mathbf{x}_{k-1}) &= q(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{x}_0) \\
&= \frac{q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0)\, q(\mathbf{x}_k \mid \mathbf{x}_0)}{q(\mathbf{x}_{k-1} \mid \mathbf{x}_0)}.
\end{aligned}
$$

これを $k \ge 2$ に代入すると、周辺分布の比が $\prod_{k=2}^{T} q(\mathbf{x}_{k-1} \mid \mathbf{x}_0)/q(\mathbf{x}_k \mid \mathbf{x}_0) = q(\mathbf{x}_1 \mid \mathbf{x}_0)/q(\mathbf{x}_T \mid \mathbf{x}_0)$ と逐次相殺され、分子が $k = 1$ で残っていた順方向ステップを打ち消す。残った各対数は、期待値をとるとそれぞれダイバージェンスになる。

$$
\begin{aligned}
L(\mathbf{x}_0) = \underbrace{D_{\mathrm{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big)}_{L_T} \\
+ \sum_{k=2}^{T} \underbrace{\mathbb{E}_q \, D_{\mathrm{KL}}\big(q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k)\big)}_{L_{k-1}} \\
+ \underbrace{\mathbb{E}_q\big[-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\big]}_{L_0}.
\end{aligned}
\tag{17.46}
$$

事前項（prior term）L_T は連鎖の終端がサンプラの出発点であるガウス分布に一致するかを測り、***ノイズ除去項***（denoising term）L_{k-1} は学習された各ステップを既知のクリーンな点に対するステップ k の逆向き分布である**順事後分布**（forward posterior）と比較し、**再構成項**（reconstruction term）L_0 は最終ステップを評価する。$\mathbf{x}_0$ で条件付けることで順事後分布は解析的に計算できるようになる。すなわち、$\mathbf{x}_{k-1}$ に関する 2 つのガウス分布（順方向ステップと $q(\mathbf{x}_{k-1} \mid \mathbf{x}_0)$）の積であるため、これもガウス分布となる（演習問題 5 で平方完成により導出する）。

$$
\begin{aligned}
q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0) = \mathcal{N}(\tilde{\mu}_k, \tilde{\beta}_k \mathbf{I}), \qquad \tilde{\beta}_k = \frac{\beta_k \, \sigma_{k-1}^2}{\sigma_k^2}, \\
\tilde{\mu}_k = \frac{\alpha_{k-1}\beta_k}{\sigma_k^2} \, \mathbf{x}_0 + \frac{\sqrt{1-\beta_k}\, \sigma_{k-1}^2}{\sigma_k^2} \, \mathbf{x}_k.
\end{aligned}
\tag{17.47}
$$

この平均は、クリーンな点とノイズを含む点がそれぞれ示唆する $\mathbf{x}_{k-1}$ の位置の重み付き平均であり、分散は順方向ステップの β_k より小さい。両端を知ることで中間点が絞り込まれるからである。

事前項とノイズ除去項はいずれもガウス分布どうしを比較するため、1 つの公式で計算できる。d 次元の等方性ガウス分布に対して、次式が成り立つ。

$$
\begin{aligned}
&D_{\mathrm{KL}}\big(\mathcal{N}(\mathbf{m}_1, v_1\mathbf{I}) \,\|\, \mathcal{N}(\mathbf{m}_2, v_2\mathbf{I})\big) \\
&\quad = \frac{d}{2}\Big(\frac{v_1}{v_2} - 1 - \log\frac{v_1}{v_2}\Big) + \frac{\|\mathbf{m}_1 - \mathbf{m}_2\|^2}{2v_2},
\end{aligned}
\tag{17.48}
$$

これは 16.1.4.3 節の公式の特別な場合である。第 1 項は分散のずれに、第 2 項は第 2 の分布の分散を単位として測った平均のずれにペナルティを課す。

各項はどの程度の大きさになるだろうか。最小の例は手計算で確認できる。1 次元座標を考え、データは $\mathcal{N}(0,1)$ から抽出され、連鎖は $T = 2$ ステップで $\beta_1 = \beta_2 = \frac{1}{2}$（したがって $\alpha_1^2 = \frac{1}{2}$、$\alpha_2^2 = \frac{1}{4}$）とし、学習される平均は理論上最良のものとする。事前項は連鎖の終端 $\mathcal{N}(\alpha_2 x_0, \frac{3}{4})$ を $\mathcal{N}(0,1)$ と比較するもので、$x_0^2/8 + 0.019$、平均すると 0.144 に等しい。2 回の大きなステップでは x_0 の情報がかなり残るからである。第 2 ステップの順事後分布は平均 $\frac{\sqrt{2}}{3}(x_0 + x_2)$、分散 $\tilde{\beta}_2 = \frac{1}{3}$ をもち、最良の学習ステップはその平均に含まれる x_0 を x_2 からの最良の推定値（依然として分散 $\frac{3}{4}$ の不確実性をもつ）で置き換える。逆方向の分散を順方向ステップ自身の分散 $\nu_k = \beta_k$ とすると、境界の期待値は次のようになる。

$$\mathbb{E}\,L = \underbrace{0.144}_{L_2} + \underbrace{0.203}_{L_1} + \underbrace{1.072}_{L_0} = 1.419 = \tfrac{1}{2}\log(2\pi e),$$

これは $\mathcal{N}(0,1)$ の真の負の対数尤度に一致する。ガウスデータとこの分散の選択のもとでは、わずか 2 ステップですでに境界は厳密になる。再構成項が最も確認しやすい。x_1 が与えられたとき、クリーンな点にはまだ分散 $\frac{1}{2}$ の不確実性が残っており、これはちょうど ν_1 に等しいため、$L_0 = \frac{1}{2}\log(2\pi\cdot\frac{1}{2}) + \frac{1}{2} = 1.072$ となる。代わりに $\nu_2 = \tilde{\beta}_2$ を用いると、L_1 は 0.25 に増え、境界全体も 1.466 に増加する。この理由は記憶に留めておく価値がある。順事後分布は x_0 を知っているため分散が小さいのに対し、x_2 からの学習ステップはそれを知らないため、その不確実性も分散に含めなければならないのである。$T = 1000$ では事前項はほぼ 0 となり、各ステップは微小になって、ノイズ除去項が訓練信号を運ぶ。

17.6.1.2 境界から重み付き回帰へ

各ノイズ除去項はネットワークに依存しない分散をもつ 2 つのガウス分布を比較するため、Equation 17.48 より平均間の距離だけが θ を含む。

$$L_{k-1} = \mathbb{E}_q\Big[\frac{1}{2\nu_k}\big\|\tilde{\mu}_k(\mathbf{x}_k,\mathbf{x}_0) - \mu_\theta(\mathbf{x}_k,k)\big\|^2\Big] + \text{const}, \tag{17.49}$$

ここで定数項は $\nu_k = \tilde{\beta}_k$ のとき 0 になる。つまり各ステップは、順事後分布の平均を出力するようネットワークに要求している。

ネットワークはその平均を直接出力する必要はない。$\mathbf{x}_0 = (\mathbf{x}_k - \sigma_k\epsilon)/\alpha_k$ を Equation 17.47 に代入し、$\sigma_k^2 = \beta_k + (1-\beta_k)\sigma_{k-1}^2$ を用いると、$\mathbf{x}_k$ の係数はまとまって $1/\sqrt{1-\beta_k}$ となり、平均は次のように書ける。

$$\tilde{\mu}_k = \frac{1}{\sqrt{1-\beta_k}}\Big(\mathbf{x}_k - \frac{\beta_k}{\sigma_k}\,\epsilon\Big). \tag{17.50}$$

この式で ϵ 以外の量はすべて、$\mathbf{x}_k$ と k を受け取るネットワークにとって既知である。そこでネットワークにはノイズを予測させ、同じ形で平均を構成する。

$$\mu_\theta(\mathbf{x}_k,k) = \frac{1}{\sqrt{1-\beta_k}}\Big(\mathbf{x}_k - \frac{\beta_k}{\sigma_k}\,\hat{\epsilon}_\theta(\mathbf{x}_k,k)\Big), \tag{17.51}$$

すると、2 つの平均の差はノイズ誤差の定数倍となる。

$$L_{k-1} = \mathbb{E}_{\mathbf{x}_0,\epsilon}\Big[\frac{\beta_k^2}{2\nu_k(1-\beta_k)\sigma_k^2}\,\big\|\epsilon - \hat{\epsilon}_\theta(\alpha_k\mathbf{x}_0 + \sigma_k\epsilon, k)\big\|^2\Big] + \text{const}. \tag{17.52}$$

境界の各項は 17.2 節で見た 1 つのノイズレベルにおけるノイズ回帰に重みを掛けたものであり、$\nu_k = \tilde{\beta}_k$ のときその重みは $\beta_k/(2(1-\beta_k)\sigma_{k-1}^2)$ に等しい。ステップ k の間に信号対雑音比 α_k^2/σ_k^2 は $\sigma_k^2/((1-\beta_k)\sigma_{k-1}^2) = 1 + \beta_k/((1-\beta_k)\sigma_{k-1}^2)$ 倍だけ低下するため、$\lambda_k = \log(\alpha_k^2/\sigma_k^2)$ とおくと重みは次のようになる。

$$\frac{\beta_k}{2(1-\beta_k)\sigma_{k-1}^2} = \tfrac{1}{2}\left(e^{\lambda_{k-1}-\lambda_k}-1\right) \approx \tfrac{1}{2}(\lambda_{k-1}-\lambda_k).$$

すなわち、この境界は対数信号対雑音比をどれだけ下げたかに応じて各ステップに重みを課す。次のセルでは、VP 連鎖（$T = 1000$）に沿って 2 通りの重みを対数信号対雑音比のステップ幅の半分と並べて出力する。

```
vp = d2l.GaussianPath.vp()
a, s = vp.grid(1000)
a, s = [float(v) for v in a[::-1]], [float(v) for v in s[::-1]]  # k = 0..T
for k in (2, 10, 100, 500, 1000):
    beta = 1 - (a[k] / a[k - 1]) ** 2
    beta_tilde = beta * s[k - 1] ** 2 / s[k] ** 2
    weight = lambda nu: beta ** 2 / (2 * nu * (1 - beta) * s[k] ** 2)
    dlam = math.log((a[k - 1] * s[k]) ** 2 / (a[k] * s[k - 1]) ** 2)
    print(f'k = {k:4d}: weight {weight(beta):.4f} (nu = beta), '
          f'{weight(beta_tilde):.4f} (nu = beta_tilde), '
          f'half log-SNR step {dlam / 2:.4f}')
```

```
k =    2: weight 0.2705 (nu = beta), 0.5900 (nu = beta_tilde), half log-SNR step 0.3900
k =   10: weight 0.0725 (nu = beta), 0.0848 (nu = beta_tilde), half log-SNR step 0.0783
k =  100: weight 0.0100 (nu = beta), 0.0102 (nu = beta_tilde), half log-SNR step 0.0101
k =  500: weight 0.0055 (nu = beta), 0.0055 (nu = beta_tilde), half log-SNR step 0.0055
k = 1000: weight 0.0101 (nu = beta), 0.0101 (nu = beta_tilde), half log-SNR step 0.0100
```

ステップ幅が小さい領域では、$\nu_k = \tilde{\beta}_k$ に対する重みは対数信号対雑音比の半ステップ幅に一致し、$\nu_k = \beta_k$ に対する重みも先頭付近を除けばそれと変わらない。いずれも対数信号対雑音比が最も急激に低下する最初の数ステップで最大となり、出力された半ステップ幅を 2 倍すると 0.78（$k = 2$）および 0.157（$k = 10$）に達する。等間隔のタイムステップからなるどのようなグリッドもこの性質からは逃れられない。定数の β でさえ対数信号対雑音比を少なくとも $\log(k/(k-1))$（ステップ k の値）だけ低下させ、これは $\log 2 \approx 0.69$（$k = 2$）、$\log(10/9) \approx 0.105$（$k = 10$）になる。本節の β は連鎖に沿って増加するため、最初の数ステップでの低下幅はさらに大きい。最初の数十ステップを過ぎると、どちらの重みも 0.01 付近かそれ以下にとどまる。ここで 17.2 節の単純損失（simple loss）は次式で定義される。

$$L_{\text{simple}}(\theta) = \mathbb{E}_{k,\mathbf{x}_0,\epsilon}\left\|\epsilon - \hat{\epsilon}_\theta(\alpha_k \mathbf{x}_0 + \sigma_k \epsilon, k)\right\|^2,$$

この損失は k を $\{1, \dots, T\}$ 上の一様分布から抽出するため、すべての重みを 1 に設定することになり、境界に比べてノイズの最も小さいステップを軽視する。それにもかかわらず、Ho et al. (2020) はこの損失のほうが良質なサンプルを生成することを見出した。これらのステップは画像の最も細かいディテールを担っており（17.6.3.2 節）、尤度はそれを漏れなく勘定に入れるが、人間の目にはほとんど知覚されないからである。

ステップ幅を限りなく小さくすると、この重み付き和は対数信号対雑音比に関する積分へと収束する。

$$L(\mathbf{x}_0) \to \frac{1}{2}\int_{\lambda_{\min}}^{\lambda_{\max}} \mathbb{E}_\epsilon\left\|\epsilon - \hat{\epsilon}_\theta(\mathbf{x}_\lambda, \lambda)\right\|^2 d\lambda + L_T + L_0, \tag{17.53}$$

ここで $\mathbf{x}_\lambda$ は対数信号対雑音比が λ である経路上の点であり (Kingma et al. 2021)、積分範囲は最小ノイズレベルにおける $\lambda_{\max}$ から経路終端の $\lambda_{\min}$ までである。言葉で言えば、この境界はノイズ誤差を重み 1 で対数信号対雑音比にわたって積分したものに、両端の 2 項を加えたものである。この極限が成り立つには λ に関してステップ幅が小さい必要があるが、離散連鎖のグリッドはデータ側の端でこの条件を満たさない。第 1 ステップから第 2 ステップにかけて対数信号対雑音比は $\log 2$ より大きく低下し、これは T がいくらであっても変わらないからである。その部分で離散連鎖の境界は緩くなる（演習問題 1）。

17.6.1.3 混合分布に対する数値評価

訓練済みネットワークの尤度を測る指標は 2 つある。境界 Equation 17.53 は確率的な逆過程に対応するものである。一方、17.3 節の確率フロー ODE はノイズからデータへの決定論的な写像であり、その密度は変数変換によって求まる。厳密な場に対しては両者は一致するが、訓練されたネットワークでは一致するとは限らない。そこで 17.2 節の混合分布のレシピに従い、VP 経路上でノイズ予測器を 20,000 ステップ訓練する。

```
mix = d2l.GaussianMixture.three_modes()

_, ema, _ = d2l.train_regression(
    d2l.TimeMLP(256, 3, rngs=nnx.Rngs(0)), d2l.regression_loss(vp, 'eps'),
    lambda key: mix.sample(key, 512), 20_000, jax.random.key(0),
    cosine=True, ema_half_life=700)
```

$d\mathbf{x}/dt = \mathbf{u}_t(\mathbf{x})$ の軌跡に沿うと、対数密度は変化率 $-\nabla \cdot \mathbf{u}_t$ で変化する。フローが点を押し広げる場所では密度が薄くなるためである。これは 38.3.3 節の瞬間的な変数変換にほかならない。微小な時刻 $t_{\min}$ におけるデータ点から、密度が標準ガウス分布となる $t = 1$ まで積分すると、次式が得られる。

$$\log p(\mathbf{x}_{t_{\min}}) = \log \mathcal{N}(\mathbf{x}_1; \mathbf{0}, \mathbf{I}) + \int_{t_{\min}}^{1} \nabla \cdot \mathbf{u}_t(\mathbf{x}_t)\, dt.$$

`pf_nll` は、場が最も急激に変化するデータ側へ密に配置した 100 ステップ上で、点の位置と累積発散を同時に積分する。数値積分には古典的 4 次ルンゲ・クッタ法（38.2 節）を用いる。これは 17.5 節のホイン法の仲間であり、1 ステップあたり場を 4 回評価する。速度は 17.2 節の変換式を通じてノイズ予測から計算する。発散は 2×2 ヤコビ行列のトレースであり、座標ごとに 1 回のフォワードモード導関数で求まる。d 次元では 1 回の評価につき d 回の導関数計算が必要になるため、画像に対してはより低コストな推定法が必要となる（演習問題 3）。積分は $t_{\min} = 0.001$（ノイズが信号の約 1% となる時刻）から開始し、混合分布の厳密なノイズ予測を基準とする。

```
def eps_exact(x, t):
    """E[eps | x_t] of the mixture, from its posterior mean."""
    x0 = mix.posterior_mean(x, vp.alpha(t), vp.sigma(t))
    return vp.x0_eps('x0', x0, x, t)[1]

def pf_nll(eps_fn, path, x, t_min=1e-3, steps=100):
    """-log p(x) under the probability-flow ODE, RK4 from t_min to 1."""
    def f(x, t):                              # velocity and its divergence
        t = jnp.full((len(x),), t)
        def u(x):                             # the velocity, by x0_eps, target
```

```
            x0, eps = path.x0_eps('eps', eps_fn(x, t), x, t)
            return path.target('velocity', x0, eps, t)
        e = jnp.eye(x.shape[1])
        jvps = [jax.jvp(u, (x,), (jnp.broadcast_to(e_i, x.shape),))
                for e_i in e]
        return jvps[0][0], sum(j[1][:, i] for i, j in enumerate(jvps))
    ts = t_min + (1 - t_min) * jnp.linspace(0, 1, steps + 1) ** 2
    def step(i, carry):                          # from the data to the noise
        x, div = carry
        ta, tb = ts[i], ts[i + 1]
        h = tb - ta
        k1, d1 = f(x, ta)
        k2, d2 = f(x + h / 2 * k1, ta + h / 2)
        k3, d3 = f(x + h / 2 * k2, ta + h / 2)
        k4, d4 = f(x + h * k3, tb)
        return (x + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4),
                div + h / 6 * (d1 + 2 * d2 + 2 * d3 + d4))
    x, div = jax.lax.fori_loop(0, steps, step, (x, jnp.zeros(len(x))))
    return (x ** 2).sum(1) / 2 + x.shape[1] / 2 * math.log(2 * math.pi) - div
```

ここで `mix` から 20,000 個のテスト点を抽出し、最初の 4,000 個に対して `pf_nll` を厳密なノイズ予測器 `eps_exact` と訓練済みネットワーク `ema` の両方で実行し、得られた確率フローの負の対数尤度を真の値 `-mix.log_prob(test)` と比較する。

```
k_test, k_bound = jax.random.split(jax.random.key(1))
test = mix.sample(k_test, 20_000)
nll = -mix.log_prob(test)
mean_se = lambda v: f'{v.mean():.4f} +- {v.std() / math.sqrt(len(v)):.4f}'
pf_exact, pf_net = (pf_nll(f, vp, test[:4000]) for f in (eps_exact, ema))
print(f'true NLL {mean_se(nll[:4000])} nats')
print(f'flow, exact field - true: {mean_se(pf_exact - nll[:4000])}')
print(f'flow, network - exact field: {mean_se(pf_net - pf_exact)}')
```

```
true NLL 2.4432 +- 0.0162 nats
flow, exact field - true: 0.0001 +- 0.0001
flow, network - exact field: 0.0041 +- 0.0013
```

厳密な場のフローは 1 点あたり約 2.5 nat という真の値を小数点以下約 4 桁まで再現しており、数値積分器の正確さが確認できる。訓練済みネットワークの負の対数尤度はそこから 1000 分の数 nat 以内に収まっており、対応のある差の標準誤差は約 1000 分の 1 nat である。ネットワークは単にサンプリングの方法だけでなく、確率密度そのものを学習しているのである。

境界の計算には両端の 2 項も必要である。事前項は約 $\frac{1}{2}\alpha_1^2\|\mathbf{x}_0\|^2 \approx 2\cdot 10^{-4}$ nat である。最終ステップは $\hat{\mathbf{x}}_0$ の周りに最小ノイズレベルの分散 $e^{-\lambda_{\max}} = \sigma^2/\alpha^2$ をもつガウス分布を出力し、$\mathbf{x}_0 - \hat{\mathbf{x}}_0 = (\sigma/\alpha)(\hat{\epsilon} - \epsilon)$ であるから、次式が成り立つ。

$$L_0 = \frac{d}{2}\log\left(2\pi e^{-\lambda_{\max}}\right) + \frac{1}{2}\,\mathbb{E}\left\|\epsilon - \hat{\epsilon}_\theta(\mathbf{x}_{t_{\min}}, t_{\min})\right\|^2.$$

セルの出力を見る前に、この内訳を見積もっておこう。2 次元でノイズレベルが 0.01 のとき、第 1 項は非常に幅の狭いガウス分布の対数正規化項であり、約 -7.3 nat となる。第 2 項は、これほど小さなノイズをデータ自身の変動と区別できないネットワークの期待二乗誤差の半分であるか

ら約 1 となり、したがって $L_0 \approx -6.3$ となる。境界全体が 2.5 nat 付近にあるなら、積分項単体では約 $2.5 + 6.3 \approx 8.8$ nat でなければならない。どちらの部分も単独では尤度ではなく、その和だけが負の対数尤度の境界となる。

bound は、一様かつ層化した λ を用いてモンテカルロ法により積分を推定する。境界単体を 0.01 nat の精度で確定するには数百万回のネットワーク評価を要する。しかし 2 つの予測器の差であれば、両者に同じデータ点・ノイズレベル・ノイズを入力することで、モンテカルロノイズの大部分が共通となって相殺されるため、はるかに少ない評価回数で済む（36.5.2.6 節）。そのため bound は、共有された乱数抽出上で複数の予測器のリストを一括評価する。

```
def bound(eps_fns, path, x0, draws, key, t_min=1e-3):
    """Per-point bound on -log p(x0) for each eps(x, t), on shared draws,
    and its integral over the log-SNR alone."""
    n, d = x0.shape
    a1, s1 = (float(f(jnp.array(1.0))) for f in (path.alpha, path.sigma))
    lam_max = float(path.log_snr(jnp.array(t_min)))
    lam_min = 2 * math.log(a1 / s1)               # the end of the path
    scale = (lam_max - lam_min) / (2 * draws)
    prior = (((a1 * x0) ** 2).sum(1) / 2          # KL to N(0, I)
             + d / 2 * (s1 ** 2 - 1 - 2 * math.log(s1)))
    key, k_eps = jax.random.split(key)
    t, eps = jnp.full((n,), t_min), jax.random.normal(k_eps, x0.shape)
    x = path.sample(x0, t, eps)                # the reconstruction term
    ends = [prior + d / 2 * math.log(2 * math.pi / math.exp(lam_max))
            + ((f(x, t) - eps) ** 2).sum(1) / 2 for f in eps_fns]
    integrals = [jnp.zeros(n) for _ in eps_fns]
    for k in jax.random.split(key, draws):    # log-SNR uniform, stratified
        k_perm, k_u, k_eps = jax.random.split(k, 3)
        u = (jax.random.permutation(k_perm, n)
             + jax.random.uniform(k_u, (n,))) / n
        t = path.time_at(lam_max - u * (lam_max - lam_min))
        eps = jax.random.normal(k_eps, x0.shape)
        x = path.sample(x0, t, eps)
        integrals = [o + scale * ((f(x, t) - eps) ** 2).sum(1)
                     for o, f in zip(integrals, eps_fns)]
    return [e + i for e, i in zip(ends, integrals)], integrals

(b_exact, b_net), (integral, _) = bound([eps_exact, ema], vp, test, 64,
                                        k_bound)
print(f'exact field: integral {integral.mean():.2f} nats, '
      f'prior and reconstruction {(b_exact - integral).mean():.2f} nats')
print(f'bound, exact field - true: {mean_se(b_exact - nll)}')
print(f'bound, network - exact field: {mean_se(b_net - b_exact)}')
```

```
exact field: integral 8.74 nats, prior and reconstruction -6.28 nats
bound, exact field - true: -0.0033 +- 0.0148
bound, network - exact field: 0.0042 +- 0.0005
```

セルの実行結果は予測を裏付けている。積分項は約 8.75 nat、両端の項は約 -6.3 nat である。厳密な場については、全 20,000 個のテスト点にわたって点ごとに対応をとった境界と真の負の対数尤度の差が標準誤差 0.015 の範囲内で 0 となっており、最良の予測器のもとでは連続時間境界にこの精度でのギャップは見られない。ネットワークの境界は厳密な場の境界より約 1000 分の 5 nat だけ大きく、この対応のある差は 10,000 分の 5 nat の精度で分解されている。2 つの指標は

いずれも一致しており、この混合分布上でネットワークは最尤法の到達値から 1000 分の 5 nat 以内に迫っている。

17.6.1.4 単純損失と境界の比較

ここまでの結果だけでは、まだ冒頭の問いには答えられない。モデルの表現力に余裕があれば、すべてのノイズレベルを覆うどの重み付けも同じ最小点、すなわち厳密なノイズ予測をもつため、どれを使っても同程度に近づくからである。重み付けが違いを生むのは、ネットワークが小さすぎてあらゆる場所で厳密にはなれないときだけであり、そのとき重み付けは誤差をどこに割り振るかを決める。そこで、単純損失と境界自身の重み付け（λ が一様であり、`bound_loss` が `time_at` を通じて抽出する）の 2 通りで、元の幅と隠れユニット数 32 のそれぞれについて 2 つのシードからノイズ予測器を訓練する。実験の計算量を抑えるため、8 個のモデルはいずれも先ほどの半分の予算である 10,000 ステップ訓練する。結果を見る前に予想を立てておこう。元の幅では 2 つの重み付けはほぼ一致するはずである。一方、32 ユニットでは、境界の重み付けで訓練したモデルのほうが境界の値で優位に立つはずである。それを直接最小化するよう訓練されているからである。他方、単純損失はほぼクリーンな点への抽出を大幅に減らし、すべてのサンプルとフロー軌跡が通過するノイズの大きいレベルにより多くを割くため、そちらでは優位に立つ可能性がある。

```
def bound_loss(path, t_min=1e-3):
    """The noise regression with the log-SNR uniform: the bound's weighting."""
    lam_max, lam_min = (float(path.log_snr(jnp.array(t)))
                        for t in (t_min, 1.0))
    def loss_fn(net, x0, key):
        k_u, k_eps = jax.random.split(key)
        u = jax.random.uniform(k_u, (len(x0),))
        t = path.time_at(lam_max - u * (lam_max - lam_min))
        eps = jax.random.normal(k_eps, x0.shape)
        return ((net(path.sample(x0, t, eps), t) - eps) ** 2).sum(1).mean()
    return loss_fn

losses = {'simple': d2l.regression_loss(vp, 'eps'), 'bound': bound_loss(vp)}
nets = {}
for width in (256, 32):
    for name, loss_fn in losses.items():
        for seed in (0, 1):
            nets[width, name, seed] = d2l.train_regression(
                d2l.TimeMLP(width, 3, rngs=nnx.Rngs(seed)), loss_fn,
                lambda key: mix.sample(key, 512), 10_000,
                jax.random.key(seed), cosine=True, ema_half_life=700)[1]
```

各モデルは次の 3 通りで評価する。テスト点上で厳密な場との対応のある差として測った境界およびフロー尤度、そして共有ノイズから祖先連鎖（ancestral chain）で抽出し真の密度のもとで評価した生成サンプルである。サンプルについては、真の密度のもとでの平均負の対数尤度と、どのモードからも標準偏差の 3 倍以上離れた逸脱点（stray）の割合を報告する。データ自身の値は約 2.47 nat および 1% である。

```
b_all, _ = bound([eps_exact, *nets.values()], vp, test, 64, k_bound)
z = jax.random.normal(jax.random.key(2), (20_000, 2))  # shared by every model
rows = {}
for (width, name, _), net, b_net in zip(nets, nets.values(), b_all[1:]):
    flow = pf_nll(net, vp, test[:4000])
```

```
        x = d2l.ancestral_sample(d2l.make_pred(net, vp, 'eps'), z, vp, 1000,
                                 jax.random.key(3))
        rows.setdefault((width, name), []).append(
            ((b_net - b_all[0]).mean(), (flow - pf_exact).mean(), mix.nll(x),
             mix.summary(x)['stray']))
    print('width loss    bound - exact  flow - exact  samples: NLL  strays'
          '   (seeds 0, 1)')
    for (width, name), (r0, r1) in rows.items():
        print(f'{width:5} {name:7}' + ''.join(
            f'{v0:7.4f}{v1:7.4f}' if i < 2 else f'{v0:7.3f}{v1:7.3f}'
            for i, (v0, v1) in enumerate(zip(r0, r1))))
    print(f'data {"":36}{mix.nll(test):7.3f}{mix.summary(test)["stray"]:14.3f}')
```

```
width loss    bound - exact  flow - exact  samples: NLL  strays   (seeds 0, 1)
  256 simple  0.0093 0.0126 0.0080 0.0090  2.500  2.501  0.011  0.011
  256 bound   0.0079 0.0078 0.0146 0.0156  2.512  2.502  0.014  0.012
   32 simple  0.0378 0.0458 0.0244 0.0254  2.514  2.518  0.015  0.016
   32 bound   0.0267 0.0262 0.0434 0.0427  2.572  2.564  0.023  0.021
data                                       2.467         0.010
```

元の幅では重み付けの影響はほとんどない。4 つのモデルはいずれも両方の指標で厳密な場から 100 分の 2 nat 以内に収まり、生成サンプルも区別がつかず、逸脱点の割合もデータ自身に近い。もっとも、それぞれの重み付けは自身が対象とする指標ではすでにわずかに優位に立っており、境界では 1000 分の 1〜5 nat、フローでは 1000 分の 5〜10 nat の差がついている。32 ユニットになるとこの差が広がり、サンプル品質にも及ぶ。境界重み付けモデルのほうが境界は 100 分の 1〜2 nat 低い。一方、尤度の境界ではない単純損失のほうがフロー尤度は良好で、負の対数尤度が 100 分の 1〜2 nat 低く、サンプルも優れている。すなわち、サンプルの平均負の対数尤度は 100 分の 2〜6 nat 低く、逸脱点の割合も約 1.5〜1.8%（境界重み付けでは 2.0〜2.3%）と少ない。この混合分布では、境界で訓練しても小規模ネットワークに最良の尤度はもたらされなかったのである。その理由は予想どおり、訓練時の抽出がどこに向かうかにあると考えられる。λ を $[-10.05, 9.12]$ 上で一様に抽出すると、境界の抽出の約 $7/19$（3 分の 1 超）が、点がほぼクリーンな $\lambda = 2$ より上に費やされるのに対し、VP 経路上で一様な t を抽出する場合はそこには約 10 分の 1 しか割かれない。

したがって、冒頭の問いに対する答えは条件付きの「否」である。単純損失は尤度境界そのものではなく、境界の被積分関数に別の重みを掛けたものである。表現力に余裕があれば、重みの違いがどちらの尤度を動かす幅も高々 100 分の 1 nat 程度にとどまるが、表現力が限られるときには、より良い境界をとるか、それともより良いフロー尤度とサンプルをとるかを重みが左右する。

17.6.2 重み付けとしての目的関数

境界は対数信号対雑音比の単位量あたりに等しい重みを置き、単純損失は連鎖の各ステップに等しい重みを置き、フローマッチングは別の経路上で速度を回帰する。共通の変数で表せば、これらは 3 通りの重み付けをもつ 1 つの目的関数である。

17.6.2.1 対数信号対雑音比による形式

これまで扱ってきた各損失は、時刻 t を抽出して $\mathbf{x}_t$ を作り、ターゲットとの二乗誤差をとる。固定した $\mathbf{x}_t$ において、どのターゲットもノイズ予測のアフィン関数であり、そのスカラー傾きは t のみに依存する。したがって、各二乗誤差は $c(t)\,\|\hat{\epsilon} - \epsilon\|^2$（係数 $c(t)$ はターゲットと経路で決まる）と書ける。変数を t から λ に変換すると、Kingma and Gao (2023) の形式が得られる。

$$\mathcal{L}_w = \frac{1}{2}\int_{\lambda_{\min}}^{\lambda_{\max}} w(\lambda)\,\mathbb{E}\big\|\hat{\epsilon}_\theta(\mathbf{x}_\lambda,\lambda)-\epsilon\big\|^2\,d\lambda,$$
$$w(\lambda) = 2\,c(t)\,p(t)\left|\frac{dt}{d\lambda}\right|, \tag{17.54}$$

ここで $p(t)$ は訓練時刻の密度である。このように、損失は各対数信号対雑音比におけるノイズ誤差をどれだけ重視するかを表す 1 つの関数によって完全に記述される。境界 Equation 17.53 は重み付け $w \equiv 1$ に対応する。

積分内部の期待値は経路に依存しない。同じ λ であれば、2 つの経路は定数倍を除いて同じノイズ付きデータ点を与えるため（17.2 節）、ネットワークは同じ予測問題に直面するからである。したがって、スケジュールが $\mathcal{L}_w$ に関与するのは重み付けと両端 $\lambda_{\min}$、$\lambda_{\max}$ を通じてのみであり、境界については両端が同じスケジュールなら同じ損失を与える (Kingma et al. 2021)。

最小の具体例として、線形経路上で一様な t を用いるノイズ損失 ($c = p = 1$) を考えよう。このとき $\sigma_t/\alpha_t = t/(1-t) = e^{-\lambda/2}$ であるから、$t = 1/(1+e^{\lambda/2})$ となり、$|dt/d\lambda| = \frac{1}{8}\,\mathrm{sech}^2(\lambda/4)$ が得られる。重み付け $w(\lambda) = \frac{1}{4}\,\mathrm{sech}^2(\lambda/4)$ は、信号とノイズの強さが等しくなる $\lambda = 0$ を中心とする山型の関数であり、両裾では $e^{-|\lambda|/2}$ のように減衰する。

17.6.2.2 既存の目的関数における重み付け

他の目的関数については 2 つの変換が必要になる。VP 経路上では、Salimans and Ho (2022) の $\mathbf{v}$ ターゲットは $\hat{\mathbf{v}} - \mathbf{v} = (\hat{\epsilon}-\epsilon)/\alpha_t$ を満たすため、$c = 1/\alpha_t^2 = 1 + e^{-\lambda}$ となる。線形経路上の速度も同じ関係式を満たし、そこでは $1/\alpha_t = 1 + e^{-\lambda/2}$ である。一様な t のもとで、定数倍を除いて次の表の重み付けが得られる（演習問題 2 で導出する）。独自にノイズレベルを抽出する EDM の行には、後ほど EDM とともに定義するデータのスケール σ_{data} が現れる。

目的関数	経路とノイズレベル	$w(\lambda)$
境界（ELBO）	任意の経路	1
ノイズ、DDPM	VP、線形 β、t 一様	$\lvert dt/d\lambda\rvert$
ノイズ	線形、t 一様	$\mathrm{sech}^2(\lambda/4)$（$\lambda=0$ に山）
ノイズ	コサイン、t 一様	$\mathrm{sech}(\lambda/2)$
$\mathbf{v}$	コサイン、t 一様	$e^{-\lambda/2}$
速度（フローマッチング）	線形、t 一様	$e^{-\lambda/2}$
EDM	$\log(\sigma_t/\alpha_t) \sim \mathcal{N}(-1.2, 1.2^2)$	$\mathcal{N}(\lambda; 2.4, 2.4^2)\,(e^{-\lambda}+\sigma_{\text{data}}^2)$

この種の式変形は定数倍を誤りやすいため、表の 2 行と DDPM の重み付けのピークをコンピュータで確認しておこう。`weighting` は代数計算を行わずに $w(\lambda)$（一様な t の場合）を求める。係数 $c(t)$ は予測ノイズに対するターゲットの傾きの二乗であり、`x0_eps` と `target` を用いて固定された $\mathbf{x}_t$ 上で計算できる。また $d\lambda/dt$ は `rates` と同様にフォワードモード自動微分で求まる。

```
def weighting(path, kind, lam):
    """w(lambda) = 2 c(t) |dt / dlambda| for t uniform, where c is the squared
    slope of the target in the predicted noise at a fixed x_t."""
    t = path.time_at(lam)
    x, zero, one = (jnp.full((len(t), 1), v) for v in (0.0, 0.0, 1.0))
    def target(eps_hat):                        # the target that eps_hat implies
```

```
            return path.target(kind, *path.x0_eps('eps', eps_hat, x, t), t)
        c = ((target(one) - target(zero)) ** 2)[:, 0]
        dlam_dt = jax.jvp(path.log_snr, (t,), (jnp.ones_like(t),))[1]
        return 2 * c / jnp.abs(dlam_dt)

    lam = jnp.linspace(-8, 8, 9)
    w_u = weighting(d2l.GaussianPath.linear(), 'velocity', lam)
    w_v = weighting(d2l.GaussianPath.cosine(), 'v', lam)
    row = lambda name, w: print(f'{name:26}' + ''.join(f'{v:8.3f}' for v in w))
    print(f'{"lambda":26}' + ''.join(f'{v:8.0f}' for v in lam.tolist()))
    row('velocity, linear path', w_u.tolist())
    row('  times e^(lambda / 2)', (w_u * jnp.exp(lam / 2)).tolist())
    row('v, cosine path', w_v.tolist())
    row('  divided by the row above', (w_v / w_u).tolist())
    fine = jnp.linspace(-9, 9, 1801)
    peak = fine[weighting(vp, 'eps', fine).argmax()]
    print(f'noise, VP path: largest weight at lambda = {peak:.1f}')
```

```
lambda                          -8      -6      -4      -2       0       2       4       6       8
velocity, linear path       54.598  20.086   7.389   2.718   1.000   0.368   0.135   0.050   0.018
  times e^(lambda / 2)       1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000   1.000
v, cosine path              35.034  12.888   4.741   1.744   0.642   0.236   0.086   0.031   0.010
  divided by the row above   0.642   0.642   0.642   0.642   0.642   0.641   0.639   0.622   0.530
noise, VP path: largest weight at lambda = -0.9
```

速度の重み付けは表示されたすべての桁で $e^{-\lambda/2}$ に一致し、$\mathbf{v}$ の重み付けも同じ曲線を 0.64 倍したもの（$\lambda = -8$ から 4 の範囲）になっている。ノイズ側の端で 0.642 となるこの定数は $2 \cdot 1.008/\pi$ である。本節のコサイン経路は Nichol and Dhariwal (2021) によるもので、$\alpha_t \propto \cos\left(\frac{\pi}{2}(t + 0.008)/1.008\right)$ のように小さなオフセット 0.008 を入れて時刻 0 で 1 から始まるよう再スケールしているため、その位相角は $\cos\frac{\pi t}{2}$ よりわずかに遅く進む。このオフセットが効くのは点がほぼクリーンな領域だけであり、そこでは比が 0.62（$\lambda = 6$）、さらに 0.53（$\lambda = 8$）へと低下する。したがって、定数倍とそのクリーン側の端を除けば、コサイン経路上の $\mathbf{v}$ 予測と線形経路上のフローマッチングは同じ目的関数である (Gao et al. 2024)。VP 経路上のノイズ損失は、信号とノイズが等しくなる点よりわずかにノイズ寄りの $\lambda = -0.9$ でピークをとる。17.2 節の速度モデルとノイズモデルは、何を学習するかではなく、その重み付けが異なるのである（Figure 17.10）。

これらの重み付けのうち、どれが尤度境界になっているのだろうか。驚くべきことに、1 つだけではない。あるレベル μ 以下で 1、それより上で 0 となる重み付けは、対数信号対雑音比 μ までノイズを加えたデータに対する境界 Equation 17.53 である。そのようなデータは μ から経路を開始するため、境界には μ 以下のレベルだけが残るからである。λ に関して単調減少する重み付けは、こうした階段関数の正の混合として表せる。したがって、そのような $\mathcal{L}_w$ はランダムなレベルのガウスノイズでデータ拡張したデータに対する境界となる (Kingma and Gao 2023)。$w = e^{-\lambda/2}$ の場合、混合密度 $-w'(\mu) = \frac{1}{2}e^{-\mu/2}$ は低い対数信号対雑音比に向かって増大するため、$\mathbf{v}$ 予測とフローマッチングは、ランダムなレベル（主に強いノイズ）までノイズを加えたデータに対する尤度訓練に相当する。これら強いノイズに耐えて残る大まかな構造を重点的に訓練し、尤度を支配する最も細かいディテールにはほとんど重みを置かない。一方、DDPM のノイズ損失と EDM の重み付けはいずれもピークをもち、この種の境界にはまったく当てはまらない。

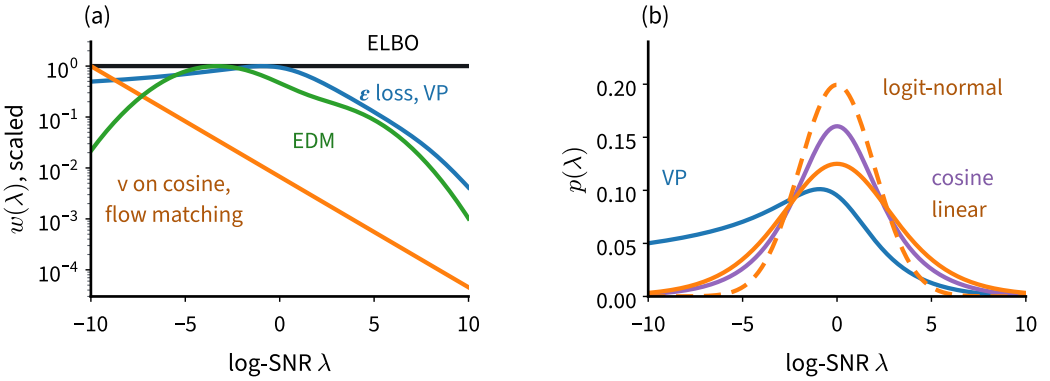


Figure 17.10: 対数信号対雑音比 λ にわたるノイズ誤差の重み付けとしての拡散モデルの目的関数。(a) 対数軸上の $w(\lambda)$。境界は平坦であり、線形 β の VP 経路上のノイズ損失は $\lambda \approx -1$ 付近でピークをとり、$\mathbf{v}$ 予測とフローマッチングは曲線 $e^{-\lambda/2}$ を共有し、EDM の重み付け（論文の値 $\sigma_{\text{data}} = 0.5$ を使用）は山型となる。(b) λ の密度（3 つの経路上で一様な t をとった場合、および線形経路上で Esser et al. (2024) のロジット正規時刻をとった場合）。

17.6.2.3 重要度サンプリングとしての時刻分布

訓練時刻は、積 $w(\lambda) = 2c\,p\,|dt/d\lambda|$ を通じてのみ Equation 17.54 に現れる。原理的には、訓練時刻の密度と重み付けは独立した選択であり、λ を範囲全体を覆う任意の密度 $p(\lambda)$ から抽出すれば、次式が成り立つ。

$$\mathcal{L}_w = \mathbb{E}_{\lambda\sim p}\Big[\frac{w(\lambda)}{2\,p(\lambda)}\,\mathbb{E}\big\|\hat{\epsilon}_\theta(\mathbf{x}_\lambda,\lambda)-\epsilon\big\|^2\Big],$$

これは 36.8.2 節の重要度サンプリング（importance sampling）の恒等式であり、$p(\lambda)$ は推定量の分散を変えるにすぎない。それでもこの選択は重要である。一様な t を線形 β の VP 経路上でとると、抽出のほぼ半分が $\lambda = -3$ 未満に費やされるが（Figure 17.10 のパネル (b)）、そこでは境界の被積分関数は小さい。その領域では $\mathbf{x}_t$ がほぼノイズそのものであり、そこからノイズを予測するのは容易だからである。この理由から Nichol and Dhariwal (2021) はコサイン経路へ移行した。さらに境界自体を推定する際には、重要度サンプリングによってステップを抽出した（演習問題 1）。

実際には $p(\lambda)$ で割ることは行われないため、時刻分布は形を変えた重み付けとして働く。よく使われるのが Esser et al. (2024) の**ロジット正規分布**（logit-normal）であり、$t = 1/(1+e^{-(m+sn)})$（ただし $n \sim \mathcal{N}(0,1)$）と定める。中心 m は経路に沿って時刻を移動させ、広がり s は集中の度合いを決め、$m = 0$ では $t = \frac{1}{2}$ の周りに時刻を集中させる。$m = 0$ で密度による除算を行わずに用いると、損失の重み付けに経路中央付近を中心とする山型の関数を掛けることになる。

17.6.3 設計空間

拡散モデルの論文はそれぞれ、経路、ターゲット、重み付け、訓練ノイズレベル、ネットワーク、サンプラを固定している。Karras et al. (2022) はこれらの選択を切り分け、その大半を互いに独立に選べることを示した。これにより、各要素を 1 つずつ比較できるようになる。

これらの選択の 1 つであるターゲットは、すべてのターゲットが同じノイズ誤差への重み付けに書き換えられた前項で消え去ったかのように見える。しかし、重み付けの観点が述べるのは損失

を最小化する関数が何かということだけであり、有限サイズのネットワークにとってその関数を出力することがどれだけ難しいかや、ある出力を別の出力へ変換するときに誤差がどれだけ拡大するかについては何も語らない。したがって、同じ重み付けをもつ 2 つの損失でも異なるネットワークが訓練されうるのであり（演習問題 2 の最後の小問でその違いを調べる）、ネットワークに何を出力させるかは依然として重要である。

17.6.3.1 EDM の比較表

自身の手法を EDM と呼ぶ Karras et al. の列に沿って整理すると、Table 17.2 の 3 つのモデルと EDM 自体は次のように表される。

手法	訓練：経路、ノイズレベル、重み付け	ネットワーク出力	サンプラ
DDPM	VP、線形 β、t 一様、単純損失	ノイズ	祖先連鎖、1,000 ステップ
NCSN	$\mathbf{x}_0 + \sigma\epsilon$、$\sigma$ の等比列、重み σ^2	スコア関数	アニーリング付きランジュバン動力学
フローマッチング	線形、t 一様	速度	オイラー法または適応的 ODE ソルバ
EDM	$\log\bar{\sigma} \sim \mathcal{N}(-1.2, 1.2^2)$、重み $1/c_{\text{out}}^2$	前処理付きクリーン画像	$\rho = 7$ グリッド上のホイン法

NCSN (Song and Ermon 2019)（38.6.3 節）はデータを一切スケールしない（$\alpha = 1$）。これは**分散爆発**（variance-exploding）の観点である。EDM は、どの経路に対しても $\bar{\mathbf{x}} = \mathbf{x}/\alpha$ とノイズレベル $\bar{\sigma} = \sigma/\alpha$ というこの座標系で、`ode_sample` と同様に動作する。

EDM が新たに加えた列はこのパラメータ化、すなわちネットワークが何を出力し、その入出力をどのようにスケールするかである。これを決める観点は 2 つある。第 1 はスケールである。ネットワーク F はすべてのノイズレベルで重みを共有するため、どのレベルでも単位分散の入力を受け取り、単位分散の出力を生成するのが望ましい。平均 0、標準偏差 σ_{data} のデータに対して、ノイズ除去器（denoiser）を次式で定める。

$$
\begin{aligned}
D(\bar{\mathbf{x}}; \bar{\sigma}) &= c_{\text{skip}}\,\bar{\mathbf{x}} + c_{\text{out}}\,F(c_{\text{in}}\,\bar{\mathbf{x}};\ \bar{\sigma}), \\
c_{\text{in}} &= \frac{1}{\sqrt{\bar{\sigma}^2 + \sigma_{\text{data}}^2}}, \quad c_{\text{skip}} = \frac{\sigma_{\text{data}}^2}{\bar{\sigma}^2 + \sigma_{\text{data}}^2}, \\
c_{\text{out}} &= \frac{\bar{\sigma}\,\sigma_{\text{data}}}{\sqrt{\bar{\sigma}^2 + \sigma_{\text{data}}^2}},
\end{aligned}
\tag{17.55}
$$

そして $\|D - \mathbf{x}_0\|^2$ に重み $1/c_{\text{out}}^2$ を掛けて訓練することで、F のターゲットも単位分散にする（各係数は演習問題 6 で導出する）。低ノイズ領域ではノイズ除去器は入力をそのまま通し（$c_{\text{skip}} \to 1$、$c_{\text{out}} \to 0$）、F はその補正のみを担う。高ノイズ領域では、出力は F の推定値をデータのスケールに合わせたものになる（$c_{\text{skip}} \to 0$、$c_{\text{out}} \to \sigma_{\text{data}}$）。中心化されていないデータに対しても、二乗平均平方根（root mean square）を σ_{data} の代わりに用いれば同じ導出がそのまま成り立つ。導出中のすべての分散が 0 周りの二次モーメント（second moment）に置き換わるためである。Karras et al. は自然画像に対して $\sigma_{\text{data}} = 0.5$ を用いているが、以下では本節のデータ上でその値を実際に計測する。

このスケールの議論からは、素のノイズ予測器で何が問題になるかもわかる。素のノイズ予測器は信号係数で割ることで出力をクリーン画像に変換するが（$\hat{\mathbf{x}}_0 = (\mathbf{x}_t - \sigma_t\hat{\epsilon})/\alpha_t$）、VP 経路上では $\alpha_1 \approx 0.007$ であるため、純粋なノイズ付近ではこの除算によってノイズ誤差が約 150 倍に拡大される。その領域では入力をそのままコピーするだけでもノイズ損失はほぼ満たされてしまうため、除算が必要とする 150 分の 1 未満まで誤差を押し下げる動機が損失には働かない。EDM は小さな数で割ることを一切しない。

第 2 の観点は、有限サイズのネットワークが何を表現できるかである。画像はピクセル数よりはるかに低次元の集合の近くに分布しているのに対し、ノイズはあらゆる次元を埋め尽くす。わずかな信号から画像全体を生成しなければならない高ノイズ領域では、クリーン画像のほうが出力しやすい対象であり、そこでは $c_{\text{skip}} \to 0$ となって F はスケールされたクリーン画像を予測する。一方、低ノイズ領域では F はノイズの定数倍、すなわち入力に対する小さな補正を予測する。Li and He (2026) はこの議論を極限まで推し進め、大きなピクセルパッチを扱う Transformer がノイズ予測や速度予測では失敗する一方、クリーン画像の予測では機能することを示した。

EDM は経路の座標系においてネットワーク $F(\mathbf{x}, t)$ をこの前処理で包み（$c_{\text{in}}\bar{\mathbf{x}} = \mathbf{x}/n$、ただし $n^2 = \sigma^2 + \alpha^2\sigma_{\text{data}}^2$ であり $\alpha = 0$ でも有限にとどまる）、クリーン画像を予測する。`edm_loss` は対数正規分布からノイズレベルを抽出し、重みを適用する。なお Karras et al. はノイズレベルを $\frac{1}{4}\log\bar{\sigma}$ としてネットワークに入力しているが、以下の比較で出力だけを変えられるよう、本節の EDM は本章の他のすべてのモデルと同様に経路の時刻 t を渡す。

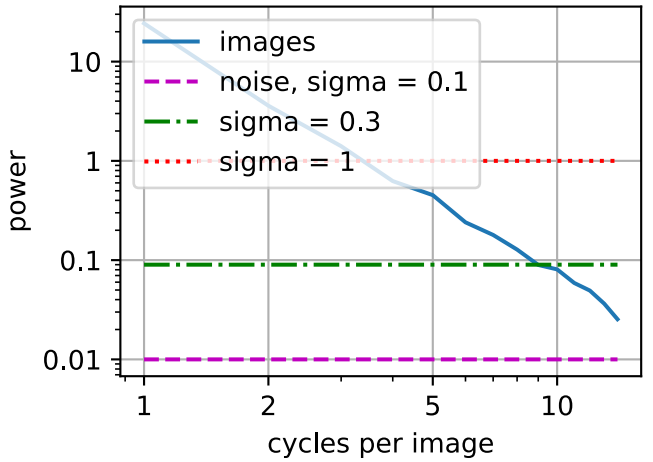
```
class EDM(nnx.Module):
    """Karras et al.'s preconditioned denoiser on a path; returns x0-hat."""
    def __init__(self, net, path, sigma_data=0.5):
        self.net, self.path, self.sigma_data = net, path, sigma_data

    def __call__(self, x, t):
        a = self.path.alpha(t).reshape(-1, 1, 1, 1)
        s = self.path.sigma(t).reshape(-1, 1, 1, 1)
        sd = self.sigma_data
        n = jnp.sqrt(s ** 2 + (a * sd) ** 2)
        return a * sd ** 2 / n ** 2 * x + s * sd / n * self.net(x / n, t)

def edm_loss(edm, x0, key):
    """Log-normal noise levels sigma_bar, loss weight 1 / c_out^2."""
    k_sigma, k_eps = jax.random.split(key)
    sigma_bar = jnp.exp(-1.2 + 1.2 * jax.random.normal(k_sigma, (len(x0),)))
    t = edm.path.time_at(-2 * jnp.log(sigma_bar))
    x = edm.path.sample(x0, t, jax.random.normal(k_eps, x0.shape))
    sd = edm.sigma_data
    weight = (sigma_bar ** 2 + sd ** 2) / (sigma_bar * sd) ** 2
    return (weight.reshape(-1, 1, 1, 1) * (edm(x, t) - x0) ** 2).mean()
```

17.6.3.2 ノイズレベルと画像のスケール

あるノイズレベルは画像から何を消し去るのだろうか。正規直交 2 次元フーリエ変換（Fourier transform）のもとでは、標準偏差 σ の白色ノイズはあらゆる周波数で一様なパワー σ^2 をもつのに対し、画像のパワーは低周波に集中している。そのため、ノイズのパワーが画像のパワーを上回った周波数成分はノイズに埋もれて見えなくなる。次のセルでは、画像あたり r サイクルの半径をもつリング上で平均したテスト画像のパワーを、3 つのノイズレベルと比較してプロットする。

```
X, y, X_test, _ = d2l.load_fashion_mnist_pm1()
spec = jnp.fft.fft2(X_test[..., 0], norm='ortho')  # noise: power sigma^2
k = jnp.fft.fftfreq(28, 1 / 28)
ring = jnp.round(jnp.sqrt(k[:, None] ** 2 + k ** 2)).astype(int).flatten()
power = (jnp.bincount(ring, (jnp.abs(spec) ** 2).mean(0).flatten())
         / jnp.bincount(ring))                    # the mean over each ring
r, power = jnp.arange(1, 15), power[1:15]
d2l.plot(r, [power] + [jnp.full((14,), s ** 2) for s in (0.1, 0.3, 1.0)],
         'cycles per image', 'power', xscale='log', yscale='log',
         legend=['images', 'noise, sigma = 0.1', 'sigma = 0.3', 'sigma = 1'])
for s in (0.1, 0.3, 0.5, 1.0, 2.0):
    hidden = jnp.nonzero(power < s ** 2)[0]
    print(f'sigma = {s}: noise dominates from', f'{int(r[hidden[0]])} cycles'
          if len(hidden) else 'no ring below 15')
```

```
sigma = 0.1: noise dominates from no ring below 15
sigma = 0.3: noise dominates from 9 cycles
sigma = 0.5: noise dominates from 6 cycles
sigma = 1.0: noise dominates from 4 cycles
sigma = 2.0: noise dominates from 2 cycles
```



画像のパワーは、画像あたり 1 サイクルから 14 サイクルにかけて 3 桁低下する。$\sigma = 0.1$ のノイズは全領域で画像のパワーを下回るが、$\sigma = 0.3$ では画像あたり約 9 サイクルより外側のリングが隠れ、$\sigma = 1$ では約 4 サイクルから外側が隠れる。したがって、ノイズを加える操作は σ が大きくなるにつれてカットオフ周波数が下がるローパスフィルタとして働き、サンプリングは粗い構造から細かい構造へと進む。高ノイズ側のステップが衣服のシルエットを決め、最後のステップがその質感を決めるのである (Dieleman 2024)。

同じ見方から、なぜスケジュールが画像解像度に依存するかも説明できる。ピクセルあたりのノイズが同じまま 56×56 にすると、各 2×2 ブロックの平均をとることで 28×28 の画像が復元される一方、4 つのノイズ値の平均のノイズ分散は $\sigma^2/4$ に下がる。つまり大まかな構造に対する信号対雑音比が 4 倍に増大するため、ある解像度で調整したスケジュールはより高い解像度では簡単になりすぎるのである。そこで Hoogeboom et al. (2023) は、ピクセル数の比の対数だけ λ_t を下方にシフトさせる（演習問題 4、Figure 17.11）。

ブロック平均はまた、スケジュールの終端がわずかに残した信号をも増幅する。本節の VP 経路

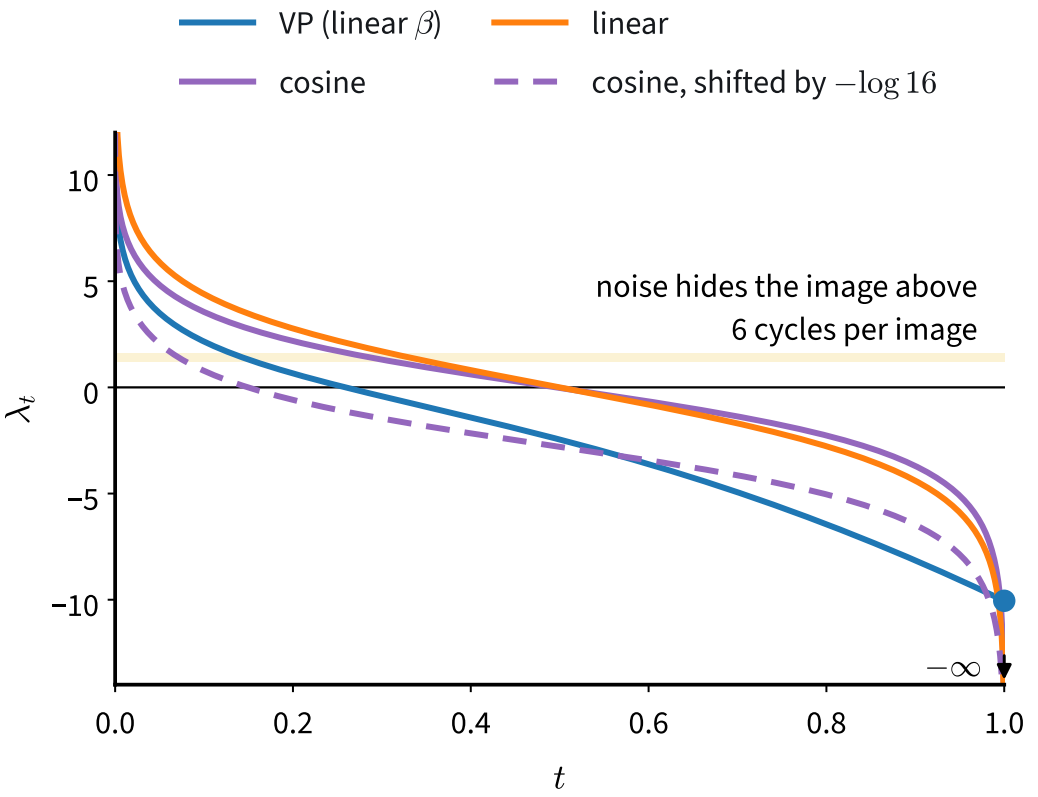


Figure 17.11: 対数信号対雑音比の時計で見たノイズスケジュール。λ_t を t に対してプロットしたもので、線形 β の VP 経路、コサイン経路、線形経路、および 1 辺の長さを 4 倍にした場合のシフト付きコサイン経路（$\lambda_t - \log 16$）を示す。VP 経路は $\lambda \approx -10$ で終わり、他は $-\infty$ で終わる。帯状の領域は、Fashion-MNIST において画像あたり 6 サイクル超の成分がノイズに隠れる対数信号対雑音比を表す。

は $\lambda \approx -10$ で終わり、そこでは全 784 ピクセルの平均の信号対雑音比は高々 $784\, e^{-10} \approx 0.04$ であり、読み取るには小さすぎる。しかし Stable Diffusion のスケジュールは、対数信号対雑音比が約 -5 のところで止まっており、チャネルあたり 64×64 個の値をもつ潜在表現上ではチャネル平均の信号対雑音比が $4096\, e^{-5} \approx 28$ に達する。モデルはこの生き残った明るさに依存するよう学習してしまい、サンプラは平均 0 のノイズから出発するため、非常に明るい画像や非常に暗い画像を生成できなくなった (Lin, Liu, et al. 2024)。

17.6.3.3 1 要因ずつの比較

いよいよ画像上でこれらの設計選択を 1 つずつ検証しよう。ベースとするのは 17.4 節のレシピ、すなわち線形経路上で t を一様とした速度予測であり、他の各行はそこから 1 つの要素だけを変更する。

行	ベースからの変更点
`t logit-normal`	$m = 0$、$s = 1$ のロジット正規時刻（Esser et al. (2024) の選択）
`t logit-normal, m = -0.5`	同上、中心をデータ寄りの $t \approx 0.38$ に設定
`cosine path, v`	コサイン経路上の $\mathbf{v}$ 予測（重み付けの表よりベースと同じ目的関数）
`t at EDM's levels`	EDM の対数正規ノイズレベル（次の行に対する対照群）
`EDM, clean image`	EDM の前処理付きクリーン画像（EDM のノイズレベルと重みを使用）

各モデルはクラス条件なしの 64 チャネル `d2l.UNet` であり、5,000 回の更新（17.4 節の予算の 3 分の 1）を 2 つのシードから訓練する（ただし比較の読み取りが最も微妙なベース、対照群、EDM には 4 シードを用いる）。すべてのモデルについて、同じ 5,000 枚のノイズ画像からホイン法でサンプリングし、5,000 枚のテスト画像に対するフレシェ距離（Fréchet distance、FD）で評価する。ここでサンプラの選択には公平性（fairness）の問題が生じる。EDM の原論文は独自の $\rho = 7$ グリッド上でサンプリングするが、一様な時刻で訓練したモデルに対しては、この計算予算ではそのグリッドが線形グリッドを上回らないことが 17.5 節でわかっている。それでも EDM のノイズレベルで訓練したモデルには適している可能性があり、それを対照群の行で検証できる。したがって単一のグリッド上での比較は特定のレシピに有利に働きかねないため、すべてのモデルを EDM のグリッド（49 回評価）と線形グリッド（48 回評価）の両方でサンプリングする。次のセルではまず、EDM の σ_{data} のためにピクセル値を計測し、あわせて 17.4.3.2 節の下限値、すなわち実画像 2 セット間の FD を求める。

```
sigma_data = float(jnp.sqrt((X ** 2).mean()))  # root mean square of a pixel
print(f'pixels: mean {X.mean():.2f}, standard deviation {X.std():.2f}, '
      f'root mean square {sigma_data:.2f}')
cnn = d2l.fashion_feature_cnn(X, y)
f_test = d2l.fd_features(cnn, X_test[:5000])
floor = d2l.fid_score(f_test, d2l.fd_features(cnn, X_test[5000:]))
print(f'FD, real against real: {floor:.2f}')
```

```
pixels: mean -0.43, standard deviation 0.71, root mean square 0.83
FD, real against real: 0.17
```

Fashion-MNIST は背景の大部分が黒であるため、ピクセルの平均は -0.43、二乗平均平方根は 0.83 と論文の値 0.5 を大きく上回っており、セルはこの実測値を `EDM` に渡す。5,000 枚の実画像 2 セット間の FD は 0.2 をわずかに下回る。

`logit_normal_loss` は、`regression_loss` の一様な時刻をロジット正規時刻に置き換えたものである。線形経路上では $\bar{\sigma} = t/(1-t)$ であるから $\log \bar{\sigma} = m + sn$ となり、$m = -1.2$、$s = 1.2$ とおけば速度モデルも EDM の対数正規ノイズレベルで訓練される。したがって、その名前の行と EDM の行との違いはパラメータ化と重みだけである。これはまた、$m = -0.5$ の行よりもさらにデータ寄りの $t \approx 0.23$ を中心とするロジット正規分布でもある。

```
def logit_normal_loss(path, kind, m=0.0, s=1.0):
    """regression_loss with logit-normal times t = sigmoid(m + s N(0, 1))."""
    def loss_fn(net, x0, key):
        k_t, k_eps = jax.random.split(key)
        t = jax.nn.sigmoid(m + s * jax.random.normal(k_t, (len(x0),)))
        eps = jax.random.normal(k_eps, x0.shape)
        error = net(path.sample(x0, t, eps), t) - path.target(kind, x0, eps, t)
        return (error ** 2).reshape(len(x0), -1).sum(1).mean()
    return loss_fn

linear, cosine = d2l.GaussianPath.linear(), d2l.GaussianPath.cosine()
unet = lambda seed: d2l.UNet(ch=64, rngs=nnx.Rngs(seed))
recipes = {  # name: (path, kind of prediction, network, loss)
    'base: velocity, t uniform': (linear, 'velocity', unet,
                                  d2l.regression_loss(linear, 'velocity')),
    't logit-normal': (linear, 'velocity', unet,
                       logit_normal_loss(linear, 'velocity')),
    't logit-normal, m = -0.5': (linear, 'velocity', unet,
                                 logit_normal_loss(linear, 'velocity', -0.5)),
    'cosine path, v': (cosine, 'v', unet, d2l.regression_loss(cosine, 'v')),
    "t at EDM's levels": (linear, 'velocity', unet,  # ln sigma_bar = logit t
                          logit_normal_loss(linear, 'velocity', -1.2, 1.2)),
    'EDM, clean image': (linear, 'x0',
                         lambda seed: EDM(unet(seed), linear, sigma_data),
                         edm_loss)}
grids = {'EDM grid': d2l.edm_grid(25),            # 49 network evaluations
         'linear grid': linear.sigma_bars(25)}    # 48 network evaluations
```

次のループは各レシピを訓練し、同じノイズから両方のグリッド上で全モデルをサンプリングして、各行・各グリッドについてシード間の平均 FD と括弧内に各シードの FD を出力する。実行前に 1 つ結果を予測できる。コサイン経路上の $\mathbf{v}$ 損失は別の座標系で表したベースの目的関数そのものであるから、ベースに劣ることはないはずである。

```
batch_fn = lambda key: X[jax.random.randint(key, (128,), 0, len(X))]
z = jax.random.normal(jax.random.key(1), (5000, 28, 28, 1))  # shared noise
four_seeds = ('base: velocity, t uniform', "t at EDM's levels",
              'EDM, clean image')         # to split EDM's gain
models, samples = {}, {}
print(f'{"FD: mean (each seed)":27} {"EDM grid":28} linear grid')
for name, (path, kind, make_net, loss_fn) in recipes.items():
    fd = {g: [] for g in grids}
    for seed in range(4 if name in four_seeds else 2):
        _, ema_img, _ = d2l.train_regression(make_net(seed), loss_fn,
                                             batch_fn, 5_000,
                                             jax.random.key(seed))
        pred = d2l.make_pred(ema_img, path, kind)
```

```
        for g, levels in grids.items():
            x = jnp.concatenate([d2l.heun_sample(pred, z[i:i + 1000], levels)
                                 for i in range(0, len(z), 1000)])
            fd[g].append(d2l.fid_score(d2l.fd_features(cnn, x), f_test))
            if seed == 0 and g == 'EDM grid':
                models[name], samples[name] = ema_img, x[:10]
    edm, lin = (f'{sum(fd[g]) / len(fd[g]):.2f} ('
                + ' '.join(f'{v:.2f}' for v in fd[g]) + ')' for g in grids)
    print(f'{name:27} {edm:28} {lin}')
```

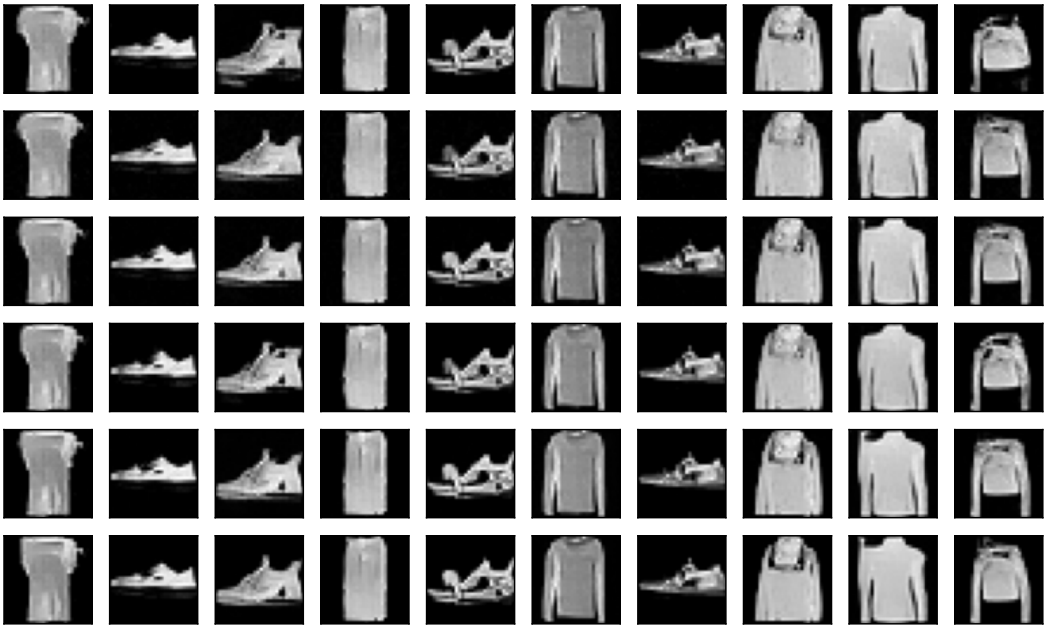
```
FD: mean (each seed)        EDM grid                    linear grid
base: velocity, t uniform   4.08 (3.94 4.11 4.19 4.07)  3.48 (3.39 3.50 3.51 3.51)
t logit-normal              5.25 (5.29 5.21)            3.95 (4.10 3.80)
t logit-normal, m = -0.5    3.51 (3.84 3.19)            2.74 (3.07 2.41)
cosine path, v              3.34 (3.43 3.25)            3.24 (3.34 3.15)
t at EDM's levels           3.56 (3.93 3.81 3.01 3.50)  3.06 (3.30 3.32 2.48 3.13)
EDM, clean image            3.11 (2.60 2.91 3.34 3.61)  3.02 (2.37 2.96 2.97 3.78)
```

5,000 回の更新（20,000 ステップという 17.7 節で潜在拡散（latent diffusion）に用いる予算の 4 分の 1）という訓練初期の段階において、この表は 2 つのサンプリンググリッドと複数のシードにわたってサンプル品質を測定している。

1. **サンプリンググリッドへの感度：**線形経路上で訓練されたすべての速度モデル（EDM のノイズレベルで訓練された対照群を含む）は、EDM グリッドより線形グリッドのほうで FD が 0.3〜1.3 低くなるのに対し、前処理を施した EDM モデルはグリッドの選択に左右されない。EDM のノイズレベルで訓練しただけでは、EDM グリッドが有利になるわけではない。
2. **訓練ノイズレベルの分布：**ロジット正規訓練時刻の中心を $m = 0$（$t = 0.5$）に置くと、EDM グリッド上で FD が 1.1〜1.3（線形グリッド上では 0.5〜0.75）だけ一様な t に比べて悪化する。ロジット正規の中心をデータ側へシフトさせるか（$m = -0.5$、$t \approx 0.38$）、EDM のノイズレベル（$m = -1.2$、$t \approx 0.23$）を用いれば、この劣化は解消する。
3. **ターゲットと前処理：**線形グリッド上では、ベースの速度レシピ、コサイン経路上の $\mathbf{v}$ 予測、および EDM がシード間で重なり合う FD の範囲を達成する。

同じ 10 枚のノイズ画像から生成した、EDM グリッド上における第 1 シードのサンプルを表の順に 1 レシピ 1 行で示す。

```
d2l.show_gray(jnp.concatenate(list(samples.values())), len(samples), 10);
```



ほとんどの列で、6 つのレシピすべてが同じノイズから細部のみ異なる同じ衣服を生成しており、衣服の種類自体が変わるのはごく一部の列に限られる。重み付けの異なる同一の事後平均（posterior mean）から期待されるとおり、各レシピはノイズから画像へのほぼ同じ写像を学習している。

ノイズレベルごとのテスト損失を調べると、中央に寄せたロジット正規モデルの弱点がどこにあるかが特定できる。次のセルでは、同じ 2,000 枚のテスト画像とノイズを用いて、このモデルとベースモデルを個別のノイズレベルで評価する。

```
x0 = X_test[:2000]
eps = jax.random.normal(jax.random.key(3), x0.shape)
for t in (0.01, 0.03, 0.1, 0.5, 0.9, 0.98):
    t = jnp.full((len(x0),), t)
    x, target = linear.sample(x0, t, eps), linear.target(
        'velocity', x0, eps, t)
    loss = [((models[name](x, t) - target) ** 2).reshape(len(x0), -1)
            .sum(1).mean()
            for name in ('base: velocity, t uniform', 't logit-normal')]
    print(f't = {t[0]:.2f}: loss {loss[0]:6.1f} (uniform), '
          f'{loss[1]:6.1f} (logit-normal)')
```

```
t = 0.01: loss  435.6 (uniform),  701.1 (logit-normal)
t = 0.03: loss  336.3 (uniform),  411.4 (logit-normal)
t = 0.10: loss  225.8 (uniform),  239.7 (logit-normal)
t = 0.50: loss  130.7 (uniform),  126.8 (logit-normal)
t = 0.90: loss  245.1 (uniform),  246.7 (logit-normal)
t = 0.98: loss  278.1 (uniform),  293.7 (logit-normal)
```

訓練時刻の大半が集まる $t = \frac{1}{2}$ では、ロジット正規モデルはベースをわずかに上回る（損失 127 対 131）が、$t = 0.01$ では大幅に劣る（約 670〜700 対 420〜440）。このモデルは $t = 0.03$ 未満の訓練時刻を 3,000 回に 1 回未満しか抽出しないのに対し、一様な時刻は 100 回に 3 回抽出し、

どのサンプラも最終ステップではこれらのレベルを使って細部を仕上げる。Esser et al. ははるかに大きな画像の潜在表現に合わせて中心を選んだが、先ほどのスケールの議論からわかるとおり、28×28 の画像はより少ないノイズで構造が決まるため、中心をシフトした行は訓練レベルをその領域へ移しているのである。

17.6.3.4 レシピの選択

新たなモデルを設計するにあたり、これまでの分析から 3 つの設計指針が導かれる。

1. **純粋なノイズ（$\alpha_1 = 0$）に到達する経路上で、条件の良いターゲットを選ぶ：**線形経路上の速度予測、コサイン経路上の $\mathbf{v}$ 予測、および EDM の前処理は $[0, 1]$ の両端で適切なスケールを保つのに対し、素のノイズ予測は高ノイズ領域で $\alpha_t \to 0$ による除算を行って $\hat{\mathbf{x}}_0$ を再構成しなければならない。
2. **空間構造が形成される領域に訓練ノイズレベルを配置する：**28×28 の画像では、一様な t または低ノイズ側へシフトしたロジット正規分布（$m < 0$）が、$t = 0.5$ を中心とするロジット正規分布を上回る。画像解像度が r 倍に増えるときは、対数信号対雑音比を $\log r^2$ だけ下方にシフトさせる（17.6.3.2 節）。
3. **境界ではなくサンプルのために損失の重み付けを選ぶ：**重み付け $w(\lambda) \equiv 1$ は尤度境界を訓練し、低ノイズの細部に多くの労力を費やす。実際に良質な画像サンプルをもたらしてきたのは、$w(\lambda) = e^{-\lambda/2}$（フローマッチングや $\mathbf{v}$ 予測）のように高い対数信号対雑音比を割り引く重み付けである (Kingma and Gao 2023)。本節の混合分布でも、境界の重み付けは小規模ネットワークに最良のフロー尤度さえもたらさなかった。

本章のデフォルト設定である、線形経路上で一様な t を用いた速度回帰と線形グリッド上の積分は、追加のハイパーパラメータなしでこれらの基準を満たしている。続く 17.7 節では、拡散過程を学習された潜在空間（latent space）へ移すことで、高周波のピクセル細部にネットワーク評価を費やすのを避ける方法を調べる。

17.6.4 まとめ

順方向のノイズ付加連鎖を反転させることで潜在変数モデルが定まり、その負の対数尤度は ELBO、すなわちステップごとの対数信号対雑音比の低下量の半分 $\frac{1}{2}(\lambda_{k-1} - \lambda_k)$ で重み付けされたノイズ予測誤差の和によって上から抑えられ、その連続時間極限は $\frac{1}{2}\int \mathbb{E}\|\epsilon - \hat{\epsilon}_\theta\|^2 \, d\lambda$ となる。より一般に、標準的な拡散モデルおよびフローマッチングの目的関数はすべて対数信号対雑音比の座標で $\mathcal{L}_w = \frac{1}{2}\int w(\lambda)\, \mathbb{E}\|\hat{\epsilon}_\theta - \epsilon\|^2 \, d\lambda$ と書け、単調減少する重み付け $w(\lambda)$ はノイズ付加データ上の尤度境界に対応する。特に、線形経路上のフローマッチングとコサイン経路上の $\mathbf{v}$ 予測は $w(\lambda) \propto e^{-\lambda/2}$ を共有する。3 モード混合分布上では、訓練済みネットワークは連続時間 ELBO と確率フロー ODE の変数変換公式の両方のもとで厳密なオラクルの負の対数尤度に 0.005 nat 以内で一致するが、表現力が制限された設定では $w(\lambda) \equiv 1$ がよりタイトな ELBO を与える一方、重みなしのノイズ回帰はより良いフロー尤度と少ない逸脱サンプルをもたらす。最後に、EDM の前処理はネットワークの入力・スキップ接続・ターゲットを全ノイズレベルにわたって単位分散にスケールし、また自然画像に対して白色ノイズはローパスフィルタとして働くため、画像解像度を r 倍にする際には対数信号対雑音比のスケジュールを $2\log r$ だけ下方にシフトさせる必要がある。

17.6.5 演習問題

1. [code] **境界の重み。**●ω_k を Equation 17.52 の重みとする。
    1. ω_k を $\nu_k = \beta_k$ と $\nu_k = \tilde{\beta}_k$ のそれぞれについて VP 連鎖のすべてのステップ $k = 2, \dots, 1000$ で計算し、両者を対数軸上にプロットせよ。それぞれの選択におい

て、$\sum_k \omega_k$ のうちステップ $k \le 20$ が占める割合はいくらか。単純損失がそこに割り当てる割合はいくらか。

2. 離散連鎖の境界 Equation 17.46 を、混合分布上で厳密なノイズ予測器、$\nu_k = \tilde{\beta}_k$、およびステップ 1 のノイズレベルにおける本文の再構成項を用いて評価せよ。これを真の負の対数尤度および本文の連続時間境界と比較せよ。本文の 2 ステップの例を踏まえると、過剰分の大部分はどのステップから生じると予想されるか、またそれはなぜか。
3. 離散連鎖の境界のミニバッチ推定では k を一様に抽出する。これに対し Nichol and Dhariwal (2021) は、k を確率 p_k（サンプルごとの L_{k-1} の値の二乗平均平方根に比例）で抽出し、抽出された項を p_k で割る。なぜこれにより推定量の分散が減少するのか。このようなサンプラは混合分布上でどのステップを最も頻繁に抽出するか。

2. **暗黙の重み付け**。@eq-diffusion-weighted-loss を用いよ。
   1. 線形経路上で t を一様とした速度損失が $w(\lambda) \propto e^{-\lambda/2}$ をもち、$\mathbf{v}$ 損失もコサイン経路 $\alpha_t = \cos\frac{\pi t}{2}$、$\sigma_t = \sin\frac{\pi t}{2}$ 上で同じ重み付けをもつことを示せ。
   2. コサイン経路上で t を一様としたノイズ損失が $w(\lambda) \propto \operatorname{sech}(\lambda/2)$ をもつことを示し、さらに w をノイズ損失と $\mathbf{v}$ 損失について線形 β の VP 経路 `vp` 上で計算せよ。
   3. これらの重み付けのうち、λ に関して単調なのはどれか。VP 経路上のノイズ損失は、同じ経路上の $\mathbf{v}$ 損失に比べてどのノイズレベルの重みを小さくしており、最終ステップでは何倍小さくなるか。
   4. （コード）第 1 小問より、コサイン経路上の $\mathbf{v}$ 損失と線形経路上の速度損失は同一の目的関数であるが、本文の比較実験ではこれらによって 2 つの異なるネットワークが訓練される。比較実験の最後のセルと同様に、両モデルの出力を同じ λ におけるノイズ予測へ `make_pred` で変換したうえで、比較実験の第 1 シードの 2 モデルを個別のノイズレベルで評価せよ。どのレベルで両者に差が生じるか。2 つの経路は同じデータ点を異なるスケールでネットワークに入力するが、$t = 1/2$ において各経路の入力はどのスケールをもつか、またその違いから観測結果をどのように説明できるか。

3. [code] **高次元における尤度**。●d 次元において、`pf_nll` の厳密な発散計算は 1 回の評価につき d 回のフォワードモード導関数を必要とする。これを 38.3 節の Hutchinson トレース推定値 $\mathbf{r}^\top(\partial \mathbf{u}/\partial \mathbf{x})\,\mathbf{r}$ に置き換え、ステップごとに 1 つのラデマッハベクトル $\mathbf{r}$ を用いるようにせよ（必要な導関数計算は 1 回になる (Grathwohl et al. 2019)）。
   1. 得られる $\log p$ の推定量が不偏であることを示せ。プローブ 1 個あたりの分散は演習問題 5（38.3 節）の第 1 小問で導出されている。混合分布上で 1 点あたりの推定値の標準偏差を測定し、訓練済みネットワークと厳密な場との間にある 1000 分の数 nat という差と比較せよ。その差を識別するには、1 ステップあたり何個のプローブ、あるいは何個のテスト点が必要か。
   2. 100 枚のテスト画像の各ピクセルに 1 階調分の幅 $2/255$ の一様ノイズを加えたうえで、比較実験のベースモデル（線形経路上の速度）にこの推定法を適用せよ。結果を次元あたりビット数 $\big(-\log p(\mathbf{x})/d + \log 127.5\big)/\log 2$（8 ビット画像のピクセルあたりビット数に相当）で報告せよ。256 階調上で一様な密度はちょうど 8 ビットとなる。正常に機能しているモデルはそれよりはるかに小さい数ビット以下の値となり、8 付近またはそれ以上の値は通常、$\log 127.5$ の付け忘れか符号の誤りを意味する。更新回数を 15,000 回にすると結果はどう変わるか。

4. **解像度とスケジュール**。ピクセルあたり単位分散をもつデータに対して $\mathrm{SNR}_t = \alpha_t^2/\sigma_t^2$ と書く。
   1. データを定数 κ 倍すると、$\mathbf{x}_t$ における信号分散とノイズ分散の比が κ^2 倍になる（すべての t において）ことを示せ。
   2. 各画像を $28r \times 28r$ ピクセルへ、各ピクセルを $r \times r$ ブロックに繰り返すことでアップサンプルし、その大きな画像に同じスケジュールでノイズを加える。ノイズ付き画像を各ブロック上で平均すると、比 $r^2\,\mathrm{SNR}_t$ をもつ元の画像が得られることを示せ。
   3. $r = 32$（896×896 画像）のとき、`vp` に対する $r^2\,\mathrm{SNR}_1$ を計算し、28×28 の経路

上で同じ比をもつ時刻を求めよ。純粋なノイズからサンプラを開始する場合、このスケジュールで訓練された 896×896 モデルにとってこの結果は何を意味するか。

4. λ_t をどのようにシフトさせれば、ブロック平均した大画像が 28×28 モデルと同じ比を通過するようになるか。これを Hoogeboom et al. (2023) のシフトと比較せよ。
5. （コード）ピクセルを繰り返して Fashion-MNIST を 56×56 にアップサンプルし、比較実験のベースレシピを線形経路そのままの場合と λ_t を $-\log 4$ だけシフトさせた場合の 2 通りで、それぞれ 2 シードから訓練せよ。第 4 小問から予想されるとおり、シード間のばらつきを上回る改善がシフトによって得られるか。

5. **順事後分布**。@eq-diffusion-posterior を導出せよ。
   1. 次式

$$q(\mathbf{x}_{k-1} \mid \mathbf{x}_k, \mathbf{x}_0) \propto q(\mathbf{x}_k \mid \mathbf{x}_{k-1})\, q(\mathbf{x}_{k-1} \mid \mathbf{x}_0)$$

   を $\mathbf{x}_{k-1}$ に関する 2 つのガウス分布の積として書き、それらの精度（逆分散）を足し合わせ、精度に比例して平均を結合せよ。
   2. $\tilde{\beta}_k \le \beta_k$ であり、等号成立は $\sigma_{k-1}^2 = 1$ の場合に限られることを示せ。連鎖のどの部分で $\tilde{\beta}_k$ は β_k よりはるかに小さくなるか。
   3. 本文の 2 ステップの例において、データを $\mathcal{N}(0, a^2)$ とする。境界を最小化する分散 ν_2 が $\mathrm{Var}(x_1 \mid x_2)$ であり、それが $\tilde{\beta}_2$（$a \to 0$ のとき）および β_2（$a = 1$ のとき）という Ho et al. (2020) の 2 つの選択に一致することを示せ。

6. **EDM の前処理**。●$\mathbf{x}_0$ はノイズと独立で、平均 0、座標ごとの分散 σ_{data}^2 をもつとする。
   1. c_{in} を選び、入力 $c_{\text{in}}(\mathbf{x}_0 + \bar{\sigma}\epsilon)$（$F$ への入力）が単位分散をもつようにせよ。F の訓練ターゲットを $(\mathbf{x}_0 - c_{\text{skip}}\,\bar{\mathbf{x}})/c_{\text{out}}$ と書いたとき、その分散を最小化するよう c_{skip} を選び、次にその分散を 1 にするよう c_{out} を選べ。結果を Equation 17.55 と照合せよ。さらに平均が 0 でないデータに対して分散を二次モーメントに置き換えて同じ導出を繰り返し、σ_{data} が二乗平均平方根になることを示せ。
   2. 損失 $\|D - \mathbf{x}_0\|^2 / c_{\text{out}}^2$ が $\|F - \text{target}\|^2$ に等しく、またノイズ損失として書き直すとノイズ誤差の $(e^{-\lambda} + \sigma_{\text{data}}^2)/\sigma_{\text{data}}^2$ 倍になることを示せ。
   3. $\log \bar{\sigma} \sim \mathcal{N}(-1.2, 1.2^2)$ のもとで、EDM の重み付けが $w(\lambda) \propto \mathcal{N}(\lambda; 2.4, 2.4^2)\,(e^{-\lambda} + \sigma_{\text{data}}^2)$ となることを示せ。そのピークはどこにあるか。Kingma and Gao (2023) は、ピークより小さいすべての λ で重みをピーク値に固定することでこれを単調関数にしている。本文の定理によれば、この変更によって EDM の目的関数は何になるか。

17.7 潜在拡散と Transformer

ピクセル空間で拡散モデルを直接実行すると、サンプリングの各ステップですべてのピクセルにわたってノイズ除去ネットワークを評価することになる。28×28 のグレースケール画像に含まれる値は 784 個にすぎないが、512×512 の RGB 写真には 786,432 個もの値が含まれ、しかも 17.6.3.2 節で示したように、こうした高周波次元の大部分は人間の目には知覚できない微細なテクスチャを担っている。**潜在拡散**（latent diffusion）は、知覚圧縮（perceptual compression）と意味的な生成を切り離す手法である (Rombach et al. 2022)。まず変分オートエンコーダ（variational autoencoder）が各画像を**潜在表現**（latents）の低次元空間グリッドへと圧縮し、拡散モデルはこの潜在空間においてステップあたりの FLOPs を大幅に抑えつつ訓練とサンプリングを行い、生成サンプルごとに 1 回だけデコーダを実行する。

第 2 のアーキテクチャ上の転換は、畳み込み U-Net を**拡散 Transformer**（diffusion transformer、DiT）に置き換えることである (Peebles and Xie 2023)。DiT は、適応的層正規化（adaptive layer normalization）の一種である `adaLN-Zero` によって変調されるパッチトークンの系列として、

画像や潜在グリッドを処理する。本節では、KL 正則化を施した畳み込みオートエンコーダを Fashion-MNIST 上で訓練し、その $7 \times 7 \times 4$ の潜在表現上で速度フローマッチングを実行して、デコーダの歪み（distortion）と潜在表現のモデル化のしやすさ（modelability）のトレードオフを診断するとともに (Dieleman 2025)、計算規模を変えながら U-Net と DiT のバックボーンを比較する。

17.7.1 知覚圧縮

17.7.1.1 ピクセルのコスト

サンプリングの各ステップでは、画像全体に対してネットワークを評価する。畳み込みはすべてのピクセルにカーネルを適用するため、U-Net のコストはピクセル数に比例して増大する。また、アテンションはあらゆる位置の組を比較するため、そのコストは位置数の二乗で増大する。512×512 のカラー画像は $512 \cdot 512 \cdot 3 = 786{,}432$ 個の数値を保持しており、50 ステップのサンプラはその一つひとつに 50 回アクセスする。これらの数値の大半は細部を記述している。画像をフーリエ基底で表してみよう。画像あたり k サイクルまでの周波数の個数は k^2 のオーダーで増えるため、座標の大半は高周波（細かい周波数）に属する。実際、17.6.3.2 節で見たように、Fashion-MNIST 画像のパワーは画像あたり 1 サイクルから 14 サイクルにかけて 3 桁減少していた。数多くの微細な座標は、画像のパワーをほとんど担っていないのである。

何を捨てるべきかを語る語彙を与えてくれるのが、レート・歪み理論（rate–distortion theory）である。各歪みレベルに対し、レート・歪み関数は符号が必要とする画像あたりの最小ビット数を表す（37.6.3.2 節）。どのビットを削っても見る人が気づかないかは二乗誤差ではなく人間の知覚に関する問いであり、Rombach et al. (2022) は観測によってこれに答えた。自然画像の尤度モデルは、曲線の低歪み側の領域において、誰にも知覚されない細部に大半のビットを費やしているのである。そこで彼らは、この**知覚圧縮**をオートエンコーダに任せ、画像が何を表しているかをモデル化する**意味圧縮**（semantic compression）を拡散モデルに任せた。Stable Diffusion のオートエンコーダは $512 \times 512 \times 3$ の画像を $64 \times 64 \times 4$ の潜在表現へと写像し、数値の個数を 48 分の 1 に減らす。拡散の各ステップはこの潜在表現上で動作し、デコーダはサンプルごとに 1 回だけ実行される。

良い潜在表現とはどのようなものだろうか。デコーダが画像を復元するのに十分な情報を保ちつつ、拡散モデルにとって学習しやすいものでなければならないが、この 2 つは相反する。すべてのピクセルをそのまま保存する潜在表現は完璧に再構成できるが、モデル化の難しさはピクセルそのものと変わらない。Dieleman (2025) は、このトレードオフを構成する 3 つの量に名前を与えている。**レート**は潜在表現が持つ数値の個数、**歪み**はデコーダが失う情報量、そして**モデル化のしやすさ**は潜在表現の生成モデルがどれだけうまく機能するかを表す。本節のオートエンコーダは、784 個の数値を 4 チャネルの 7×7 グリッド、すなわち 196 個の数値というレートへ写像する（Figure 17.12）。

17.7.1.2 オートエンコーダ

3 つの量のうち最も自明でないのがモデル化のしやすさであるため、拡散モデルが潜在表現に何を求めるかを整理しておこう。必要なのは、ガウス経路（Gaussian path）のノイズを較正する基準となる既知のスケール、孤立した外れ値がないこと、そしてわずかにずれた潜在表現からももっともらしい画像が得られるよう、近くの潜在表現が似た画像にデコードされることの 3 点である。再構成のみを目的として訓練されたオートエンコーダは、これらのいずれも保証しない。コードを 1,000 のスケールで保持することも、隣接する 2 つのコードを靴とシャツにデコードすることも妨げられないからである。

変分オートエンコーダは、これら 3 つすべてに対する解決策を与える (Kingma and Welling 2014)。エンコーダは平均 μ と標準偏差 $\mathbf{s}$ を 196 個の潜在値それぞれに対して出力し、潜在表現

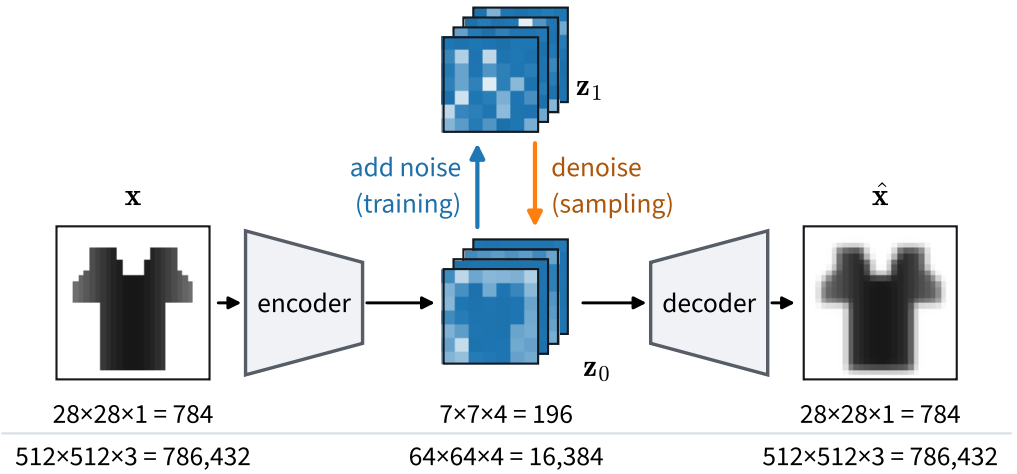


Figure 17.12: 潜在拡散。エンコーダは画像をより小さな潜在表現に写像し、拡散モデルは潜在表現上で訓練とサンプリングを行い、デコーダはサンプリングされた潜在表現を画像に戻す。上：本節のオートエンコーダであり、$28 \times 28 \times 1$（784 個の数値）から $7 \times 7 \times 4$（196 個）へ写像する。下：Stable Diffusion のオートエンコーダであり、$512 \times 512 \times 3$（786,432 個）から $64 \times 64 \times 4$（16,384 個）へ写像する。

は $\mathbf{z} = \mu + \mathbf{s} \odot \xi$（ただし ξ は標準正規分布に従う）としてサンプリングされ、デコーダは $\mathbf{z}$ を再構成 $\hat{\mathbf{x}}$ に写像する。損失は、再構成誤差に対して、各潜在値を標準正規分布へ引き寄せる次の項を加える。

$$\mathcal{L}(\mathbf{x}) = \frac{1}{784}\,\|\mathbf{x} - \hat{\mathbf{x}}\|^2 + \frac{\beta}{196}\sum_{j=1}^{196} \tfrac{1}{2}\left(\mu_j^2 + s_j^2 - 1 - \log s_j^2\right). \tag{17.56}$$

この和はカルバック・ライブラー・ダイバージェンス（Kullback–Leibler divergence）$D_{\mathrm{KL}}\big(\mathcal{N}(\mu, \mathrm{diag}(\mathbf{s}^2)) \,\|\, \mathcal{N}(\mathbf{0}, \mathbf{I})\big)$、すなわちエビデンス下界（evidence lower bound）に現れる項であり（36.7 節）、β はその重みである（スケジュールの β_k ではない）。先ほどの 3 つの要件に照らすと、この損失は 2 つの働きをする。各 μ_j を 0 に、各 s_j を 1 に引き寄せることで、スケールを固定し外れ値にペナルティを科す。また、訓練中に $\mathbf{z}$ を大きさ $\mathbf{s}$ のノイズ付きでサンプリングすることで、デコーダには約 $\mathbf{s}$ だけ揺らいだコードが提示されるため、それらを同様にデコードするよう学習する。これが 3 つ目の要件にあたる。

こうした作用がどの程度働くかは β に依存する。本節で選ぶ $\beta = 10^{-4}$ では、再構成が主導権を握ったままである。この重みでは、KL 項はスケールを固定し、エンコーダのノイズが 0 に崩壊するのを防ぐものの、隣接する潜在表現が同様にデコードされるようにするにはあまりに弱すぎる。それが問題になるかどうかは自明ではなく、以下で実際に測定する。

大規模システムのオートエンコーダには、さらなる工夫が加えられている。二乗誤差はもっともらしいテクスチャの候補を平均化してぼやけさせてしまうため（16.7 節）、通常は画像パッチにスコアを付けるクリティックと知覚損失（perceptual loss）を併用して訓練されてきた。ただし、近年の大規模オートエンコーダである ViTok-v2 は、スケールアップに際してクリティックを廃し、固定された知覚損失のみを用いている (Hansen-Estruch et al. 2026)。本節のモデルにはどちらも不要である。Stable Diffusion のオートエンコーダが写真の数値の 48 分の 1 しか保持しない

のに対し、本節のモデルは黒い背景上の小さなグレースケール画像の 4 分の 1 の数値を保持するからである。このほか、潜在表現を離散トークンへ量子化するオートエンコーダもあり (Esser et al. 2021; Chang et al. 2022)、その場合は較正すべきスケールのない有界なコードが得られ、マスク付き拡散（masked diffusion）（17.9 節）が扱う形式の入力となる。

本節のオートエンコーダは、埋め込みを持たない U-Net の残差ブロックを幅 64 と 128 で積み重ねたものである。エンコーダは解像度を 2 回半分にして、潜在チャネルごとに平均と対数分散を持つ 7 × 7 グリッドへと変換し、デコーダはその鏡像の構造をとる。

```
class Autoencoder(nnx.Module):
    """Residual convolutional autoencoder from 28x28x1 images to a 7x7
    Gaussian latent and back; returns the reconstruction and the KL term."""
    def __init__(self, latent_channels=4, ch=64, num_blocks=2, *, rngs):
        blocks = lambda a, b: [d2l.UNetBlock(a, b, rngs=rngs)] + [
            d2l.UNetBlock(b, b, rngs=rngs) for _ in range(num_blocks - 1)]
        conv = lambda a, b, **kw: d2l.Conv(a, b, (3, 3), rngs=rngs, **kw)
        down = lambda c: conv(c, c, strides=2, padding=1)
        up = lambda x: jnp.repeat(jnp.repeat(x, 2, axis=1), 2, axis=2)
        head = lambda a, b: [nnx.GroupNorm(a, num_groups=8, epsilon=1e-5,
                                           use_fast_variance=False,
                                           rngs=rngs), nnx.silu, conv(a, b)]
        self.encoder = nnx.Sequential(
            conv(1, ch), *blocks(ch, ch),
            down(ch), *blocks(ch, 2 * ch),                      # 14 x 14
            down(2 * ch), *blocks(2 * ch, 2 * ch),              # 7 x 7
            *head(2 * ch, 2 * latent_channels))  # mean and log variance
        self.decoder = nnx.Sequential(
            conv(latent_channels, 2 * ch), *blocks(2 * ch, 2 * ch),
            up, *blocks(2 * ch, ch),                            # 14 x 14
            up, *blocks(ch, ch),                                # 28 x 28
            *head(ch, 1))

    def encode(self, x):
        mean, log_var = jnp.split(self.encoder(x), 2, axis=-1)
        return mean, jnp.clip(log_var, -30, 20)

    def __call__(self, x, key):
        mean, log_var = self.encode(x)
        z = mean + jnp.exp(0.5 * log_var) * jax.random.normal(key, mean.shape)
        kl = 0.5 * (mean ** 2 + jnp.exp(log_var) - 1 - log_var).mean()
        return self.decoder(z), kl
```

`d2l.train_regression`（17.1 節）を用いて、20,000 回の更新にわたりこのモデルを訓練する。この関数は損失に対して Adam を実行し、訓練済みモデル、重みの移動平均、および損失の履歴を返す。ただし、ここではコサイン曲線に沿って学習率が 0 まで減衰するため、最終的な重みの平均化は不要である。実行前にコストについて一言述べておくと、本節では 2 つのオートエンコーダと 14 個の拡散モデルをそれぞれ 20,000 回更新するため、1 基の RTX 4090 上でノートブック全体の実行に約 1 時間半かかり、CPU 上ではそれよりはるかに長い時間を要する。

```
X, y, X_test, y_test = d2l.load_fashion_mnist_pm1()

def batch_fn(key):
```

```
    return X[jax.random.randint(key, (128,), 0, len(X))]

def ae_loss(ae, x, key, beta=1e-4):
    x_hat, kl = ae(x, key)
    return ((x_hat - x) ** 2).mean() + beta * kl

ae, _, _ = d2l.train_regression(d2l.Autoencoder(rngs=nnx.Rngs(1)), ae_loss,
                                batch_fn, 20_000, jax.random.key(1),
                                cosine=True)
print(f'{d2l.num_params(ae) / 1e6:.2f}M parameters')
```

```
2.40M parameters
```

1 行目は 8 枚のテスト画像、2 行目はそれらの潜在平均に対するデコーダの出力である。

```
encode = nnx.jit(lambda ae, x: ae.encode(x))      # mean and log variance
decoder = nnx.jit(lambda ae, z: ae.decoder(z))
x_hat = decoder(ae, encode(ae, X_test[:8])[0])
d2l.show_gray(jnp.concatenate([X_test[:8], x_hat]), 2, 8);
```

再構成画像は元の画像と見分けがつきにくい。これを定量化するため、17.4 節で導入したフレシェ距離（Fréchet distance、FD）を用いる。これは小規模な分類器 `d2l.FeatureCNN` が 2 組の画像集合から抽出した特徴にそれぞれガウス分布を当てはめ、その間の距離を測る指標である。実画像の 2 つの集合であっても偶然による差が生じるため、互いに素な 5,000 枚ずつのテスト画像 2 組の間の FD を**フロア**（下限の目安）として表示する。本節の FD はすべて 5,000 枚の画像を用いて計算する。1,000 枚ではフロアが約 5 倍高くなるからである（17.4.3.2 節）。

```
cnn = d2l.fashion_feature_cnn(X, y)
f_ref = d2l.fd_features(cnn, X_test[:5000])

def fd(imgs):
    """FD of images against the first 5,000 test images."""
    return d2l.fid_score(d2l.fd_features(cnn, imgs), f_ref)

print(f'floor, the other 5,000 test images: FD {fd(X_test[5000:]):.2f}')
```

```
floor, the other 5,000 test images: FD 0.17
```

`fd` を用い、残りの 5,000 枚のテスト画像 `X_test[5000:]` をバッチ単位でエンコードおよびデコードして、ピクセルあたりの二乗再構成誤差と再構成のフレシェ距離 `fd_rec` の両方を測定しよう。

```
x_hat = d2l.batched(lambda x: decoder(ae, encode(ae, x)[0]), X_test[5000:])
fd_rec = fd(x_hat)
print(f'reconstructions: squared error {((x_hat - X_test[5000:]) ** 2).mean():.4f}'
      f' per pixel, FD {fd_rec:.2f}')
```

```
reconstructions: squared error 0.0039 per pixel, FD 0.78
```

再構成画像の FD スコアは、フロアの 0.17 に対して 0.6〜0.8 であり、ピクセルあたりの二乗誤差は約 0.004 である。これがオートエンコーダの歪みであり、同時に理想的な潜在モデルのスコアでもある。もし拡散モデルがエンコーダの生み出す分布どおりに潜在表現をサンプリングできれば、エンコーダの小さなノイズを除いて、デコーダはそれらを再構成画像へと変換するはずだからである。以下のすべての潜在モデルは、この数値を基準として評価する。ただし、後の摂動テストで示す理由により、これは厳密な下界ではなく参照値である。

訓練セットを 1 回エンコードし、潜在表現の標準偏差、エンコーダの $\mathbf{s}$ の平均、および全潜在値の超過尖度（excess kurtosis）を表示する。超過尖度は正規分布では 0 になり、少数の値が大きく外れた位置にあると大きな値をとる。

```
encoded = d2l.batched(lambda x: jnp.concatenate(encode(ae, x), -1), X)
mu, log_var = jnp.split(encoded, 2, axis=-1)  # mean and log variance
s = jnp.exp(0.5 * log_var)
z = mu + s * jax.random.normal(jax.random.key(2), mu.shape)
scale = 1 / float(z.std())
kurtosis = (((z - z.mean()) * scale) ** 4).mean() - 3
print(f'latent std {z.std():.2f} (scale factor {scale:.2f}), encoder std '
      f'{s.mean():.3f}, excess kurtosis {kurtosis:.2f}')
```

```
latent std 0.95 (scale factor 1.05), encoder std 0.024, excess kurtosis -0.45
```

潜在表現の標準偏差は 1 をわずかに下回っているため、約 1.05 の係数 c を掛けて経路のノイズと同じ単位分散に揃える。また、超過尖度は 0 付近または負であり、裾の重い分布にはなっていない。これで 3 つの要件のうち 2 つが満たされた。一方、エンコーダの標準偏差は 0.02〜0.03 であるため、訓練中にデコーダが目にした各コードのぼかしは潜在表現の広がりの約 3% にとどまり、10 分の 1 を超えることはほとんどなかった。わずかにずれた潜在表現をもっともらしい画像にデコードするという 3 つ目の要件は、この弱い KL 項では満たされないのである。

17.7.2 潜在空間での拡散

17.7.2.1 潜在表現に対する同じ回帰

オートエンコーダを固定すれば、データ以外は何も変わらない。前節までに構築した仕組みがすべてそのまま引き継げることこそ、この方式の魅力である。各バッチでは、128 枚の訓練画像の潜在表現 $\mathbf{z}_0 = c\,(\mu + \mathbf{s} \odot \xi)$ を、Rombach et al. (2022) と同様に毎回新たにエンコーダの分布からサンプリングする。訓練は 17.2 節の線形経路上で行い、$\mathbf{z}_t = (1-t)\,\mathbf{z}_0 + t\,\epsilon$ を構成して、ネットワークの出力を速度 $\epsilon - \mathbf{z}_0$ へと回帰させる。

スケール係数が重要になるのは、この経路がデータを単位分散のノイズと混合するためである。標準偏差が 5 の潜在表現をそのまま使うと、あらゆる時刻で対数信号対雑音比（log-SNR）が $\log 25$ だけシフトしてしまう（演習問題 1）。Stable Diffusion が潜在表現に 0.18215 を掛けているのも同じ理由による。ネットワークには 4 つの入力チャネルと出力チャネルを持つ 17.4 節の U-Net を用い、画像と潜在表現の両方で同一の `loss_fn` を共用する。

```
path = d2l.GaussianPath.linear()
loss_fn = d2l.regression_loss(path, 'velocity')  # for images or latents

def latent_batch_fn(key):
    k1, k2 = jax.random.split(key)
    i = jax.random.randint(k1, (128,), 0, len(mu))
    return (mu[i] + s[i] * jax.random.normal(k2, mu[i].shape)) * scale

_, unet_latent, _ = d2l.train_regression(
    d2l.UNet(ch=64, in_channels=4, out_channels=4, rngs=nnx.Rngs(3)),
    loss_fn, latent_batch_fn, 20_000, jax.random.key(3))
```

比較基準として、同じ U-Net をピクセル上でも訓練する。どちらの実行も更新回数は 20,000 回と、17.5 節の予算の 2 倍に設定する。後述する Transformer にはより長い訓練が必要であり、このピクセルモデルはその比較基準も兼ねるからである。

```
_, unet_pixel, _ = d2l.train_regression(d2l.UNet(ch=64, rngs=nnx.Rngs(4)),
                                        loss_fn, batch_fn, 20_000,
                                        jax.random.key(4))
```

`sample` は固定されたノイズから `d2l.ode_sample`（17.3 節）のオイラー法 50 ステップで 5,000 個のサンプルを生成し、`decode` はスケール係数を元に戻して潜在表現をデコードする。下の図の 1 行目はピクセルモデルのサンプル、2 行目は潜在モデルのデコード済みサンプルである。

```
@nnx.jit
def euler50(net, noise):
    pred = d2l.make_pred(net, path, 'velocity')
    return d2l.ode_sample(pred, noise, path.sigma_bars(50))

def sample(net, shape, n=5000):
    """n samples by 50 Euler steps from fixed noise, 1,000 at a time."""
    noise = jax.random.normal(jax.random.key(0), (n, *shape))
    return d2l.batched(lambda x: euler50(net, x), noise)

def decode(z):
    return d2l.batched(lambda z_k: decoder(ae, z_k / scale), z)

latent_shape, image_shape = (7, 7, 4), (28, 28, 1)
x_pixel = sample(unet_pixel, image_shape)
x_latent = decode(sample(unet_latent, latent_shape))
fd_pixel, fd_latent = fd(x_pixel), fd(x_latent)
print(f'FD: pixel U-Net {fd_pixel:.2f}, latent U-Net {fd_latent:.2f}')
d2l.show_gray(jnp.concatenate([x_pixel[:8], x_latent[:8]]), 2, 8);
```

```
FD: pixel U-Net 1.52, latent U-Net 2.22
```

どちらの行にも、それとわかる衣類が並んでいる。FD スコアは潜在モデルが約 2（2.0〜2.2）、ピクセルモデルが約 1.5 である。本節のすべてのモデルと同様、これらはどちらも無条件モデルであるため、17.4 節のクラス条件付きモデルのスコアである 0.8 とは直接比較できない。

では、潜在モデルによって何が節約されたのだろうか。`train_step_gflops` は、128 個の入力に対する順伝播と逆伝播からなる 1 訓練ステップの浮動小数点演算数を、10 億単位（GFLOP）で計測する。

```
def train_step_gflops(net, loss_fn, shape, batch_size=128):
    """GFLOP of one training step, forward and backward, on a batch of zeros
    of the given shape: XLA's estimate for the step lowered for the CPU (for
    a GPU it leaves out the convolutions)."""
    graphdef, params, rest = nnx.split(net, nnx.Param, ...)
    grad = jax.grad(lambda p, r, x, key: loss_fn(nnx.merge(graphdef, p, r),
                                                 x, key))
    args = jax.device_put((params, rest, jnp.zeros((batch_size, *shape)),
                           jax.random.key(0)), jax.devices('cpu')[0])
    return jax.jit(grad).lower(*args).cost_analysis()['flops'] / 1e9
```

`train_step_gflops` を 4 チャネルの潜在 U-Net（`latent_shape`）と 1 チャネルのピクセル U-Net（`image_shape`）で評価し、両者のステップあたりの演算コストを直接比較しよう。

```
gflops_latent = d2l.train_step_gflops(
    d2l.UNet(ch=64, in_channels=4, out_channels=4, rngs=nnx.Rngs(0)), loss_fn,
    latent_shape)
gflops_pixel = d2l.train_step_gflops(d2l.UNet(ch=64, rngs=nnx.Rngs(0)),
                                     loss_fn, image_shape)
print(f'GFLOP per training step: latent U-Net {gflops_latent:.0f}, '
      f'pixel U-Net {gflops_pixel:.0f}')
```

```
GFLOP per training step: latent U-Net 11, pixel U-Net 192
```

潜在 U-Net がステップあたりに必要とする演算量は約 12 GFLOP であるのに対し、ピクセル U-Net が要する GFLOP 数はおよそ次の値に達する。200. 訓練時および各サンプリングステップのいずれにおいても、潜在ネットワークの評価 1 回あたりの演算量は 10 分の 1 未満で済む。

17.7.2.2 デコーダの耐性とモデル誤差

このように、潜在モデルは計算コストがはるかに低い一方で、品質はやや劣る。その FD はピクセルモデルの約 1.4 倍、再構成画像の約 3 倍である。Dieleman の用語で言えば、196 個の数値というレートには 0.6〜0.8 の歪みが伴い、そこから生成サンプルまでの隔たりがモデル化のしやすさを測る尺度となる。この差はどこから生じるのだろうか。考えられる原因は 2 つある。1 つは、

潜在モデルがエンコーダの出力とは異なる潜在表現をサンプリングしてしまうこと、すなわち潜在モデル自身の誤差である。もう 1 つは、潜在モデルがわずかにずれただけの潜在表現をサンプリングしているのに、広がりの約 3% しか揺らがないコードで訓練されたデコーダがそれを質の低い画像に変えてしまうことである。KL 項が弱いことを踏まえると、2 つ目の原因はいかにもありそうに思える。

この 2 つを切り分けるため、誤ったコードに対する耐性を持つデコーダを構築し、実際に耐性があることを確認したうえで、その潜在表現上で潜在モデルを訓練してみよう。耐性の低さが差の原因であれば、生成サンプルは再構成画像に近づくはずである。候補とするのは、10 倍の KL 重みで訓練した 2 つ目のオートエンコーダである。耐性の確認では、残りの 5,000 枚のテスト画像の潜在平均に対し、潜在表現の標準偏差の σ 倍にあたるガウスノイズを加えることで、意図的にずらした潜在表現をデコードする。先に結果を予測しておこう。2 つ目のエンコーダはコードをより強くぼかすため、そのデコーダはより大きな誤差に耐える一方、再構成の品質は悪化するはずである。

```
ae_10, _, _ = d2l.train_regression(
    d2l.Autoencoder(rngs=nnx.Rngs(9)),
    lambda ae, x, key: ae_loss(ae, x, key, beta=1e-3), batch_fn, 20_000,
    jax.random.key(9), cosine=True)
```

perturbed_decodes は、潜在表現の標準偏差を単位とするエンコーダの標準偏差、再構成誤差と FD、そして各 σ に対するデコード画像の平均二乗変化量と FD を表示する。

```
def perturbed_decodes(ae, x, sigmas=(0.1, 0.3, 0.5)):
    """Decode the latent means of x, and the means plus Gaussian noise of
    sigma times their standard deviation."""
    mu, log_var = jnp.split(
        d2l.batched(lambda x: jnp.concatenate(encode(ae, x), -1), x), 2,
        axis=-1)
    x_rec = d2l.batched(lambda z: decoder(ae, z), mu)
    print(f'  encoder std {jnp.exp(0.5 * log_var).mean() / mu.std():.3f}, '
          f'reconstructions: error {((x_rec - x) ** 2).mean():.4f}, '
          f'FD {fd(x_rec):.2f}')
    eps = jax.random.normal(jax.random.key(0), mu.shape)
    for sigma in sigmas:
        x_noisy = d2l.batched(lambda z: decoder(ae, z),
                              mu + sigma * mu.std() * eps)
        print(f'  sigma {sigma}: change {((x_noisy - x_rec) ** 2).mean():.4f} '
              f'per pixel, FD {fd(x_noisy):.2f}')

for beta, model in ((1e-4, ae), (1e-3, ae_10)):
    print(f'beta {beta:g}')
    perturbed_decodes(model, X_test[5000:])
```

```
beta 0.0001
  encoder std 0.025, reconstructions: error 0.0039, FD 0.78
  sigma 0.1: change 0.0022 per pixel, FD 0.63
  sigma 0.3: change 0.0198 per pixel, FD 0.73
  sigma 0.5: change 0.0528 per pixel, FD 3.20
beta 0.001
  encoder std 0.192, reconstructions: error 0.0041, FD 1.09
  sigma 0.1: change 0.0026 per pixel, FD 1.00
```

```
sigma 0.3: change 0.0228 per pixel, FD 0.66
sigma 0.5: change 0.0608 per pixel, FD 1.27
```

2 つの予測はどちらも的中している。重みを 10 倍にしたことで、エンコーダは各コードを潜在表現の広がりの約 5 分の 1（以前の 7 倍）だけぼかすようになり、再構成のスコアは約 0.3 悪化した。広がりの半分にあたるノイズを加えると、弱い重みのデコーダでは FD が 3 を超えて跳ね上がるのに対し、耐性の高いデコーダでは 1.3 付近にとどまる。それでいて、デコード画像の変化量はピクセルあたり 0.053〜0.055 対 0.061〜0.062 と、両者でほぼ変わらない。強い KL 項は、誤ったコードを無視するようデコーダに教えたのではない。誤ったコードを、元とは異なるがもっともらしい別の衣類へとデコードするよう教えたのであり、フレシェ距離が高く評価するのはまさにその点である。

出力結果には小さな驚きも含まれている。どちらのオートエンコーダでも、わずかなノイズを加えると、デコード画像は元の画像から遠ざかるにもかかわらず、FD が再構成画像の値を下回るのである（両者とも $\sigma = 0.1$ で低下し、耐性の高い方では $\sigma = 0.3$ でさらに低下する）。考えられる理由は、各コードの平均をデコードすると訓練中にデコーダが目にしたコード群の平均のような少し滑らかすぎる画像になり、そこにわずかなノイズが加わることで実画像の多様性が一部回復するからだというものである（本節のコードセルではこの仮説までは検証しない）。いずれにせよ、出力された数値から重要な結論が確定する。再構成 FD はあくまで参照値であり、いかなる潜在モデルも超えられない下界ではないということである。

では、耐性の高いデコーダを使えば、より良いサンプルが得られるだろうか。`latent_unet_fd` は、与えられたオートエンコーダの潜在表現に対して先ほどと同じ手順（潜在 U-Net を 20,000 回更新して訓練し、オイラー法 50 ステップでサンプリングする）を実行し、デコードされたサンプルの FD を返す。オートエンコーダごとに 2 つのシードから 2 モデルずつ訓練する。

```
def latent_unet_fd(ae, seed):
    """FD of a latent U-Net trained on the latents of ae as above."""
    mu, log_var = jnp.split(
        d2l.batched(lambda x: jnp.concatenate(encode(ae, x), -1), X), 2,
        axis=-1)
    s = jnp.exp(0.5 * log_var)
    key_z, key_train = jax.random.split(jax.random.key(seed))
    c = 1 / (mu + s * jax.random.normal(key_z, mu.shape)).std()  # scale

    def latent_batch(key):
        k1, k2 = jax.random.split(key)
        i = jax.random.randint(k1, (128,), 0, len(mu))
        return (mu[i] + s[i] * jax.random.normal(k2, mu[i].shape)) * c

    _, net, _ = d2l.train_regression(
        d2l.UNet(ch=64, in_channels=4, out_channels=4, rngs=nnx.Rngs(seed)),
        loss_fn, latent_batch, 20_000, key_train)
    z = sample(net, latent_shape)
    return fd(d2l.batched(lambda z_k: decoder(ae, z_k / c), z))

for beta, model in ((1e-4, ae), (1e-3, ae_10)):
    fds = [latent_unet_fd(model, seed) for seed in (10, 11)]
    print(f'beta {beta:g}: latent U-Net FD {fds[0]:.2f} and {fds[1]:.2f}')
```

```
beta 0.0001: latent U-Net FD 2.09 and 2.24
beta 0.001: latent U-Net FD 2.64 and 2.63
```

結果は否である。10 倍の重みを用いた潜在 U-Net はいずれも、弱い重みを用いたどのモデルより

も約 0.5 ポイント（0.4〜0.8）悪いスコアとなった。これはどちらの設定における 2 シード間のばらつきと比べても少なくとも 2.5 倍以上の差であり、しかも 10 倍の重みのオートエンコーダによるサンプルは、それ自身の再構成 FD からもさらに大きく離れている。デコーダの耐性の低さが原因であったなら、耐性の高いデコーダによってこの差は縮まっていたはずだが、どの実行でもそうはならなかった。摂動テストは 2 つ目の手がかりも与えてくれる。潜在表現の広がりの半分にあたるノイズを加えても耐性の高いデコーダの FD は約 1.3 にしかならないのに、その潜在表現上で訓練した潜在モデルのスコアは 2.6〜2.8 に達している。つまり、これらのモデルが犯している誤りが何であれ、それは正しい潜在表現に乗った小さなランダム誤差ではなく、それより大きな誤差か、あるいは別種の誤差なのである。本節の規模において、この差の正体は潜在モデル自身の誤差である。

だからといって、オートエンコーダに責任がないわけではない。同じ手順で訓練した同じ U-Net が、一方のエンコーダの潜在表現では他方よりも明確に悪い結果を出した以上、潜在モデルの仕事の難しさを決めているのはエンコーダである。これこそが Dieleman のいうモデル化のしやすさであり、訓練した拡散モデルによってのみ測定できるオートエンコーダの性質にほかならない。したがって KL 重みは実質的な設計上の選択であり、ここでは小さい値を選ぶのが正しい。強い KL 項は再構成と生成サンプルの両方を悪化させたからである。演習問題 3 では、この重みをさらに大小両方向へ振って調べる。

では、モデル化しやすい潜在表現とは何だろうか。2024 年以降の研究は、KL 重みよりも、潜在表現が何を表しているかに注目している。REPA (S. Yu et al. 2025) は、拡散モデルの隠れ状態を DINOv2 のような事前学習済み画像エンコーダの特徴と整合させる損失を加えることで、所定のサンプル品質に数倍速く到達する。表現オートエンコーダ（representation autoencoder）(Zheng et al. 2026) はさらに踏み込み、KL 正則化エンコーダ自体を廃して、凍結した事前学習済みエンコーダから潜在表現を供給し、それを逆変換するデコーダのみを訓練する。このような潜在表現は圧縮型のものよりはるかに多くのチャネルを持つため、先ほどの議論からすれば不利になるはずだが、実際には拡散モデルはこれをより容易に学習する。こうした潜在表現では、近くの点が似たピクセルではなく似た物体を表す。単に小さく滑らかであることよりも、意味によって整理された潜在表現であることの方が重要であるらしい。

28×28 の解像度では、このトレードオフに魅力はない。ピクセルモデルはもともと安価であるうえ、オートエンコーダには独自の訓練が必要になり、しかも潜在モデルのサンプルはスコアで劣るからである。

17.7.3 Transformer バックボーン

U-Net には、近くのピクセルが最も重要であり、同じ特徴が画像内のどこにでも現れうるという畳み込みネットワークの仮定が組み込まれている。11.5 節のビジョン Transformer はそうした仮定をより少なく置いており、画像分類では少量のデータで訓練すると畳み込みネットワークに後れを取るが、大量のデータで事前学習するとそれを追い抜く。Peebles and Xie (2023) は、拡散においても同じことが成り立つかを問うた。

17.7.3.1 パッチとトークン

DiT はビジョン Transformer と同じ方法で入力を読み取る。画像を $p \times p$ のパッチに切り分け、それぞれを幅 d のベクトルへと線形変換（カーネルサイズとストライドが p の畳み込み）で写像し、学習される位置埋め込み（position embedding）を加えて、Pre-Norm の Transformer ブロック（11.1 節）にトークンを通す。$p = 4$ のとき、28×28 の画像はそれぞれ 16 ピクセルを持つ 49 個のトークンになる。一方、本節の潜在表現に $p = 1$ を適用すると、49 個のグリッド位置のそれぞれが 4 個の数値からなるトークンとなる。

出力は分類器とは異なる。ノイズ除去器は入力と同じ形状の予測を返すため、DiT は最終の線形

層によって各トークンをそのパッチの値へと写像し、パッチを組み直して元の形に戻す。各トークンはアテンションを通じて他のすべてのトークンから情報を受け取りながら、自身のパッチの速度を予測する。ピクセルごとに 1 トークンを割り当てるとトークン数は 16 倍になり、アテンションのコストは 256 倍に膨れ上がる。

17.7.3.2 適応的層正規化

Transformer には時刻 t、さらに場合によってはラベルも入力する必要がある。17.4 節の U-Net では、埋め込まれた時刻とラベルの学習された射影を各ブロックの特徴マップにチャネルごとのシフトとして加算することでこれを解決していた。適応的層正規化はこのシフトを残したうえで、スケールとゲートを追加する。

通常の層正規化は各トークンを標準化し、学習されたスケールとシフトを適用する。**適応的層正規化**では、スケール $\mathbf{a}$ とシフト $\mathbf{b}$ を条件 $\mathbf{c}$（時刻埋め込みとラベル埋め込みの和）から計算し、さらにこの条件によって各サブ層の出力に対する**ゲート** $\mathbf{g}$ も設定する。1 つのブロックは次の計算を行う。

$$\mathbf{X} \leftarrow \mathbf{X} + \mathbf{g}_1 \odot \text{Attention}\big(\text{LN}(\mathbf{X}) \odot (1 + \mathbf{a}_1) + \mathbf{b}_1\big),$$
$$\mathbf{X} \leftarrow \mathbf{X} + \mathbf{g}_2 \odot \text{MLP}\big(\text{LN}(\mathbf{X}) \odot (1 + \mathbf{a}_2) + \mathbf{b}_2\big),$$

ここで、LN 自体は固有のスケールやシフトを持たず、6 つのベクトルは $\text{SiLU}(\mathbf{c})$ に適用される 1 つの線形層から得られ、1 枚の画像内の全トークンで共有される。したがって、画像あたり 1 回の小さな行列積によって、各サブ層の各特徴量がノイズレベルごとに異なる強さで増減される。

この線形層はゼロで初期化されるため、**adaLN-Zero** と呼ばれる。初期状態ではすべてのゲートが 0 になるため、各ブロックは恒等写像を計算し、同じくゼロで初期化された最終層は 0 を出力する。訓練はいたるところで速度 0 を予測するネットワークから始まり、ゲートが大きくなるにつれて各ブロックが働き始める（Figure 17.13）。このブロックのあらゆる要素のなかで最も記憶に留めておくべき細部は、ゼロから始まるゲートである。Peebles and Xie (2023) は、この設計を、条件を追加トークンとして末尾に加える方法、条件に対するクロスアテンションを用いる方法、およびゲートなしの適応的層正規化と比較し、4 つのなかで adaLN-Zero が最良のサンプルを与えることを示した。演習問題 4 では、ゼロからの開始が最初の 2,000 回の更新にどのような影響を与えるかを確認する。

時刻やラベルは画像あたり 1 つのベクトルであるため、適応的層正規化に適している。一方、キャプションは凍結されたテキストエンコーダが出力するトークンごとのベクトルの系列であり、これにはクロスアテンションが自然な選択肢となる。**クロスアテンション**では、m 個の画像トークンがクエリを供給し、n 個のキャプショントークンがキーと値を供給するため、出力は画像トークンごとに 1 行、すなわち画像と同じ形状になる (Rombach et al. 2022)（これに対し、Esser et al. (2024) のマルチモーダル DiT は画像トークンとキャプショントークンを 1 つの系列にまとめる）。訓練中にキャプションをランダムに脱落させれば、空のキャプションが分類器なしガイダンス（classifier-free guidance）（17.4.4.2 節）におけるヌルラベルの役割を果たす。

`DiTBlock` は 11.5 節の `ViTBlock` を基にこのブロックを構築し、線形変換として `ada` を持つ。

```
class DiTBlock(d2l.ViTBlock):
    """ViTBlock with adaLN-Zero: the condition c sets a shift, a scale and a
    gate for each sublayer; the gates start at zero."""
    def __init__(self, num_hiddens, mlp_num_hiddens, num_heads, *, rngs):
        super().__init__(num_hiddens, mlp_num_hiddens, num_heads, 0.0,
                         use_bias=True, rngs=rngs)
```

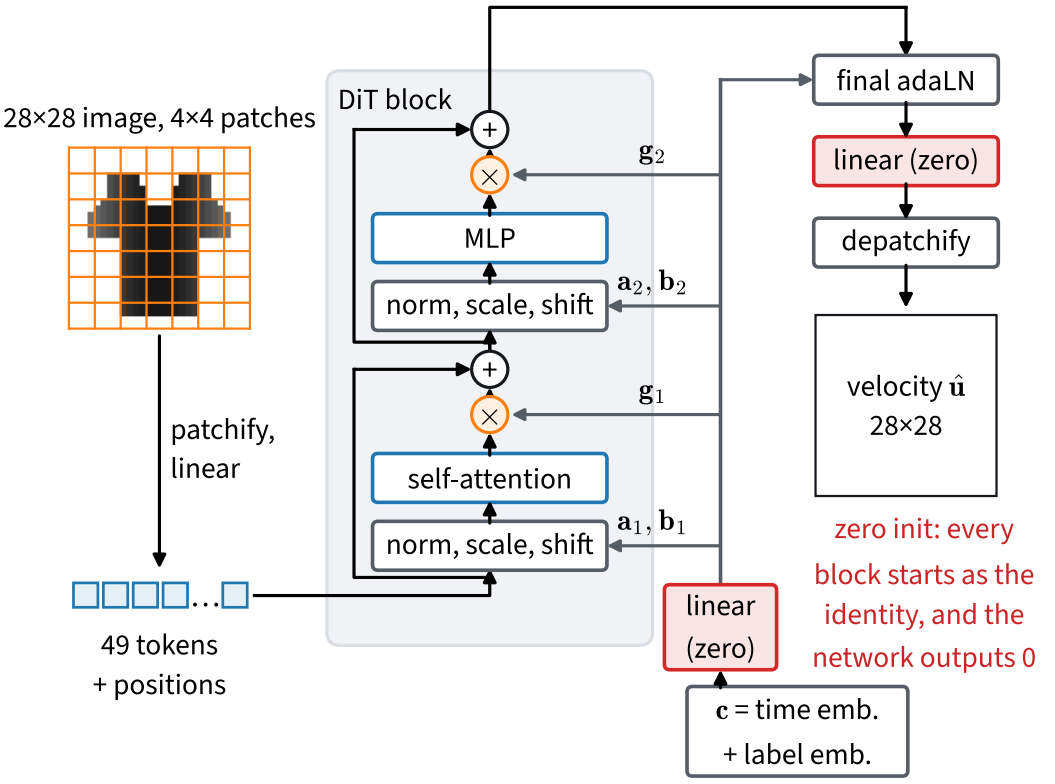


Figure 17.13: 拡散 Transformer。左：4×4 のパッチが、学習される位置を持つ 49 個のトークンになる。中央：DiT ブロック。時刻埋め込みとラベル埋め込みの和である条件 $\mathbf{c}$ が各サブ層のシフト、スケール、ゲートを設定し、ゲートはゼロから始まる。右：最終の適応的層正規化とゼロ初期化された線形層が、各トークンを対応するパッチへ写像する。

```
        ln = lambda: nnx.LayerNorm(num_hiddens, epsilon=1e-6, use_scale=False,
                                   use_bias=False, rngs=rngs)
        self.norm1, self.norm2 = ln(), ln()
        zeros = nnx.initializers.zeros
        self.ada = nnx.Linear(num_hiddens, 6 * num_hiddens, kernel_init=zeros,
                              bias_init=zeros, rngs=rngs)

    def __call__(self, X, c):
        shift1, scale1, gate1, shift2, scale2, gate2 = jnp.split(
            self.ada(nnx.silu(c))[:, None], 6, axis=-1)
        X_mod = self.norm1(X) * (1 + scale1) + shift1
        X = X + gate1 * self.attention(X_mod, X_mod, X_mod)
        return X + gate2 * self.ffn(self.norm2(X) * (1 + scale2) + shift2)
```

DiT はパッチ埋め込み（patch embedding）、学習される位置、時刻埋め込みとラベル埋め込み、および最終の適応的層正規化と線形層を追加する。

```
class DiT(nnx.Module):
    """Diffusion transformer (NHWC) conditioned on a time t in [0, 1] and,
    with num_classes, on a label y; label num_classes means no label."""
    def __init__(self, img_size=28, patch_size=4, num_channels=1, width=256,
                 depth=6, num_heads=4, num_classes=None, *, rngs):
        self.width, self.num_classes = width, num_classes
        self.p, self.g = patch_size, img_size // patch_size
        self.patch = d2l.PatchEmbedding(img_size, patch_size, width,
                                        num_channels, rngs=rngs)
        self.pos = nnx.Param(
            0.02 * rngs.params.normal((1, self.g ** 2, width)))
        self.time_mlp = d2l.make_mlp([width] * 3, nnx.silu, rngs=rngs)
        init = nnx.initializers.normal(1.0)  # N(0, 1), as in PyTorch
        self.label_emb = (nnx.Embed(num_classes + 1, width, rngs=rngs,
                                    embedding_init=init)
                          if num_classes else None)
        self.blks = nnx.List([DiTBlock(width, 4 * width, num_heads, rngs=rngs)
                              for _ in range(depth)])
        self.ln = nnx.LayerNorm(width, epsilon=1e-6, use_scale=False,
                                use_bias=False, rngs=rngs)
        zeros = nnx.initializers.zeros
        self.ada = nnx.Linear(width, 2 * width, kernel_init=zeros,
                              bias_init=zeros, rngs=rngs)
        self.out = nnx.Linear(width, patch_size ** 2 * num_channels,
                              kernel_init=zeros, bias_init=zeros, rngs=rngs)

    def __call__(self, x, t, y=None):
        c = self.time_mlp(d2l.time_embedding(t, self.width))
        if y is not None:
            c = c + self.label_emb(y)
        X = self.patch(x) + self.pos[...]
        for blk in self.blks:
            X = blk(X, c)
        shift, scale = jnp.split(self.ada(nnx.silu(c))[:, None], 2, axis=-1)
        X = self.out(self.ln(X) * (1 + scale) + shift)  # (n, g*g, p*p*c)
```

```
        n, g, p = x.shape[0], self.g, self.p  # tokens back to pixels
        X = X.reshape(n, g, g, p, p, -1).transpose(0, 1, 3, 2, 4, 5)
        return X.reshape(n, g * p, g * p, -1)
```

幅 256、深さ 6、ヘッド数 4 の DiT を構築し、8 枚の画像と時刻を入力してみよう。

```
dit = d2l.DiT(rngs=nnx.Rngs(0))  # width 256, depth 6, 4 heads, 4 x 4 patches
x, t = X[:8], jax.random.uniform(jax.random.key(5), (8,))
tokens, out = dit.patch(x), dit(x, t)
print('images', x.shape, '-> tokens', tokens.shape, '-> output', out.shape)
print(f'parameters: DiT {d2l.num_params(dit) / 1e6:.2f}M, U-Net '
      f'{d2l.num_params(d2l.UNet(ch=64, rngs=nnx.Rngs(0))) / 1e6:.2f}M')
print('output at initialization all zero:', bool((out == 0).all()))
```

```
images (8, 28, 28, 1) -> tokens (8, 49, 256) -> output (8, 28, 28, 1)
parameters: DiT 7.39M, U-Net 1.20M
output at initialization all zero: True
```

予告どおり、未訓練の出力は厳密に 0 である。この DiT は U-Net の 6 倍にあたる 739 万個のパラメータを持つが、以下で行うスケーリング実験では、ステップあたりの演算量は両者でほぼ同等であることがわかる。違いは重みの使われ方にある。畳み込みは少数の重みを多数の位置すべてに適用するのに対し、DiT は多数の重みをわずか 49 個のトークンに適用するのである。

17.7.3.3 計算量に対するスケーリング

Transformer は U-Net より優れているのだろうか。どちらの種類であっても大きいネットワークは小さいネットワークに勝るため、単一のサイズだけを見ても一般的な答えは出ない。重要なのは、規模を大きくしたときにそれぞれがどう改善するかである。Peebles and Xie (2023) は 4 種類のサイズの DiT を、それぞれ 3 種類のパッチサイズで訓練した。訓練ステップ数を固定すると、ネットワークを大きくした場合でもパッチを小さくした場合でも、順伝播あたりの演算量が増えるにつれて FID は着実に低下した。700 万ステップ訓練された最大のモデルは、256×256 のクラス条件付き ImageNet において FID の最良記録を更新した。

同じ測定を本節の規模で両方のバックボーンに対して行おう。2 つの DiT は原論文の大半のモデルと同様にヘッドあたり 64 次元を保ちながら幅と深さをともに拡大したもので、DiT-S（幅 128、深さ 4、ヘッド数 2）および先ほど構築した DiT-M である。2 つの U-Net はチャネル数を `ch=32` から先ほどのピクセルモデルである `ch=64` へと倍増させたものである。いずれもピクセル上で同じ損失と手順により 20,000 回の更新にわたり、異なるシードから 2 回ずつ訓練し、1 訓練ステップの FLOPs によってその規模を測る。

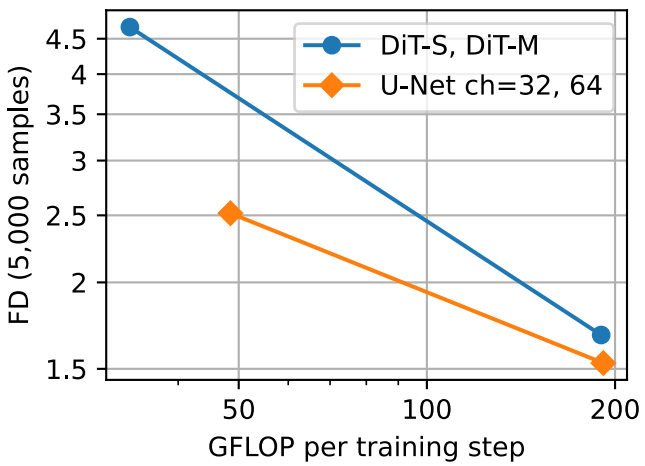
```
sizes = {'DiT-S': dict(width=128, depth=4, num_heads=2),
         'DiT-M': dict(width=256, depth=6, num_heads=4)}
nets = {'DiT-S': lambda rngs: d2l.DiT(**sizes['DiT-S'], rngs=rngs),
        'DiT-M': lambda rngs: d2l.DiT(**sizes['DiT-M'], rngs=rngs),
        'U-Net 32': lambda rngs: d2l.UNet(ch=32, rngs=rngs),
        'U-Net 64': lambda rngs: d2l.UNet(ch=64, rngs=rngs)}
seeds = {'DiT-S': (5, 12), 'DiT-M': (6, 13), 'U-Net 32': (7, 14),
         'U-Net 64': (15,)}  # the first U-Net 64 is the pixel model above
runs = {name: dict(fd=[]) for name in nets}
runs['U-Net 64'].update(fd=[fd_pixel], imgs=x_pixel[:8])
for name, make_net in nets.items():
```

```
        for seed in seeds[name]:
            net = make_net(nnx.Rngs(seed))
            _, ema, _ = d2l.train_regression(net, loss_fn, batch_fn, 20_000,
                                             jax.random.key(seed))
            imgs = sample(ema, image_shape)
            runs[name]['fd'].append(fd(imgs))
            runs[name].setdefault('imgs', imgs[:8])
        runs[name].update(params=d2l.num_params(net), gflops=d2l.train_step_gflops(
            net, loss_fn, image_shape))
    for name, run in runs.items():
        print(f'{name:8} {run["params"] / 1e6:5.2f}M parameters '
              f'{run["gflops"]:4.0f} GFLOP per step   FD '
              + ', '.join(f'{v:.2f}' for v in run['fd']))
```

```
DiT-S     1.26M parameters   33 GFLOP per step   FD 4.54, 4.81
DiT-M     7.39M parameters  190 GFLOP per step   FD 1.67, 1.68
U-Net 32  0.34M parameters   48 GFLOP per step   FD 2.55, 2.49
U-Net 64  1.20M parameters  192 GFLOP per step   FD 1.52, 1.54
```

各モデルの 2 回の実行における平均 FD を、ステップあたりの FLOPs に対して両対数軸でプロットする。

```
    curves = [[runs['DiT-S'], runs['DiT-M']], [runs['U-Net 32'], runs['U-Net 64']]]
    d2l.plot([[run['gflops'] for run in curve] for curve in curves],
             [[sum(run['fd']) / 2 for run in curve] for curve in curves],
             'GFLOP per training step', 'FD (5,000 samples)',
             legend=['DiT-S, DiT-M', 'U-Net ch=32, 64'], xscale='log',
             yscale='log', fmts=('o-', 'D-'))
```



以下で述べる差はいずれも、関係する各モデルの 2 回の実行間のばらつきの少なくとも 2 倍以上あるため、3 つの読み取りは確かなものである。1 つ目は意外な結果である。DiT-S は ch=64 の U-Net と同程度のパラメータ数（126 万個対 120 万個）を持ちながら FD はその 3 倍に達し、6 倍のパラメータを持つ DiT-M でさえまだ及ばない。28 × 28 の衣類画像においては、畳み込みに組み込まれた仮定の方が、パラメータ数を 6 倍に増やすことよりも価値が大きいのである。

計算量で測っても、差は縮まるものの、依然として両方の水準で U-Net が優位に立つ。小規模側では、ステップあたり約 50 GFLOP の ch=32 の U-Net が約 2.5 のスコアであるのに対し、演算量がそれより 3 分の 1 少ない DiT-S は 4.5〜4.9 である。200 GFLOP 付近では、U-Net が約 1.5、DiT-M が 1.7〜1.9 となる。それでも、規模の拡大に伴う改善幅は Transformer の方が大きい。計算量を 5.8 倍にすると DiT の FD は 2.5〜2.8 分の 1 に下がるのに対し、計算量を 4 倍にしたときの U-Net の FD は約 1.65 分の 1 にとどまる。プロット上でより急な傾きを示しているのは Transformer の直線である。

下の各行は、同じノイズから生成した DiT-S、DiT-M、および 2 つの U-Net の最初の 8 サンプルを示している。

```
d2l.show_gray(jnp.concatenate([run['imgs'] for run in runs.values()]), 4, 8);
```

4 つのモデルはいずれもノイズ画像を 1 枚の画像へと送る常微分方程式（ODE）を積分しているため、ほとんどの列で同じ衣類を描き出している（1〜2 列では衣類の種類自体が変化している）。この 8 列を眺めただけでは、最上段の FD が最下段の 3 倍も悪いことはほとんど読み取れないだろう。フレシェ距離は分類器の特徴を通じて何千ものサンプルを比較するが、私たちの目は 8 枚しか比較していないからである。

本節の 4 点だけでは、大規模領域で Transformer を主流のバックボーンへと押し上げた主張、すなわち幅と深さを増やすだけの単純な手順で何桁もの計算量にわたり改善し続けるという事実までは示せない。その傾向を示したのは Peebles と Xie の実験であり、本節の実験ではない。ここでのプロットにはより急な傾きは現れているものの、逆転までは至っていない。このスケーリング傾向に加えて Transformer を魅力的にしているのは、その演算が大規模な行列積として実行されるため、言語モデルを大規模に訓練するための手法がそのまま流用できる点である。

17.7.4 表現とバックボーンの選択

本節の前半と後半のテーマは、最後の 1 つの実験で交差する。潜在空間において Transformer は有効だろうか。サイズ 1 のパッチで潜在表現上に DiT-M を訓練してみよう。これにより、ピクセル上のときと同じ 49 個のトークンを扱うことになり、ステップあたりにも同じ演算量（潜在 U-Net の 12〜17 倍）を費やす。

```
net = d2l.DiT(img_size=7, patch_size=1, num_channels=4, **sizes['DiT-M'],
              rngs=nnx.Rngs(8))
_, dit_latent, _ = d2l.train_regression(net, loss_fn, latent_batch_fn, 20_000,
                                        jax.random.key(8))
gflops = d2l.train_step_gflops(net, loss_fn, latent_shape)
print(f'latent DiT-M, {gflops:.0f} GFLOP per step: FD '
      f'{fd(decode(sample(dit_latent, latent_shape))):.2f}; latent U-Net '
      f'{fd_latent:.2f}; reconstructions {fd_rec:.2f}')
```

```
latent DiT-M, 190 GFLOP per step: FD 2.22; latent U-Net 2.22; reconstructions 0.78
```

潜在 DiT-M の FD スコアは約 2（1.8〜2.2）と、潜在 U-Net（1.9〜2.2）とほぼ同じであり、それらと同様に再構成 FD の約 3 倍の位置に着地する。また、ここでも潜在表現が Transformer を助けることはない。同一の計算量で比べると潜在 DiT-M はピクセル DiT-M を上回らず、最良のピクセルモデルが本節のあらゆる潜在モデルに勝るからである。

では、オートエンコーダを導入してまで潜在表現を使う価値があるのはどのような場合だろうか。潜在表現による節約効果はピクセル数とともに増大するが、その代償であるオートエンコーダの歪みとモデル化の難化はそうではない。512×512 の解像度では、Stable Diffusion の潜在表現上で動作する U-Net が扱う位置数は 64 分の 1 になり、演算量もおよそその比率で減少する（本節でも、16 倍の位置数を扱うピクセル U-Net は潜在 U-Net の 13〜18 倍の演算量を要していた）。この節約は訓練の各更新と 50 回のサンプリングステップのすべてに効いてくるため、その画像サイズになればオートエンコーダの代償を上回る。

Transformer の場合、計算量の事情は異なる。$64 \times 64 \times 4$ の潜在表現上でサイズ 2 のパッチを使うと DiT が扱うトークン数は 1,024 個になるが、ピクセル上でサイズ 16 のパッチを使っても、ほぼ同じコストで同じ 1,024 個のトークンを扱うことになる（演習問題 2）。オートエンコーダは Transformer のトークン数を減らさない。その役割は、$16 \cdot 16 \cdot 3 = 768$ 個の生のピクセル値の線形埋め込みを、学習された非線形な埋め込みに置き換えることである。これほど多くの値の線形埋め込みはノイズへの対処が不得手である。画像がはるかに低次元の集合の近くに横たわっているのに対し、ノイズは 768 次元すべてを埋め尽くすからである。近年の 2 つの設計は、この問題を中心に据えてピクセル空間の Transformer を構築している。Li and He (2026) は大きなパッチを維持したままノイズや速度ではなくクリーンな画像を予測し（演習問題 6）、砂時計型 Transformer（hourglass transformer）(Crowson et al. 2024) は U-Net の多重解像度の形状を取り入れて細かい解像度では局所的にのみアテンションを行うことで、コストがピクセル数に対して線形にしか増えないようにしている。

以上をまとめよう。ピクセル空間での拡散が計算予算（compute budget）に収まる低解像度画像では、U-Net を用いてピクセルを直接拡散すればよい。高解像度画像では、オートエンコーダを再構成の歪みと潜在表現のモデル化のしやすさの両方を左右する生成パイプラインの中核部分として位置づけたうえで、軽く KL 正則化したオートエンコーダの潜在表現を拡散させ、大きな計算予算にわたってスケールさせる際には DiT バックボーンを用いるのがよい。

17.7.5 まとめ

潜在拡散は、KL 正則化オートエンコーダによって画像を低次元の空間グリッドへと圧縮し、潜在表現を単位標準偏差に再スケーリングしたうえで、潜在空間において標準的な拡散またはフローマッチングの回帰を実行する。オートエンコーダはレート（潜在次元数）、歪み（再構成 FD）、およびモデル化のしやすさ（拡散モデルが潜在分布をどれだけ容易に学習できるか）のバランスをとる。KL 重み β を大きくするとランダムな潜在摂動に対するデコーダの耐性は高まるものの、再構成品質と生成サンプルの FD はともに悪化する。ノイズ除去バックボーン自体については、拡散 Transformer（DiT）が画像や潜在グリッドをパッチへとトークン化し、ゼロ初期化された残

差ゲートを持つ適応的層正規化である adaLN-Zero を通じて条件付けを注入する。28×28 画像における小さな計算予算では、同一 FLOPs で比較すると畳み込み U-Net が DiT を上回る一方、ステップあたりの計算量を拡大すると DiT の方がより急な改善の傾きを示す。

17.7.6 演習問題

1. **潜在表現のスケール**。線形経路はデータを単位分散のノイズと混合する。
   1. 各潜在値の標準偏差が 1 ではなく ς であるとする。時刻 t における信号対雑音比が $\varsigma^2(1-t)^2/t^2$ となり、したがってあらゆる時刻で対数信号対雑音比が $\log \varsigma^2$ だけシフトすることを示せ。
   2. $\varsigma = 5$ の潜在表現が、$\varsigma = 1$ の潜在表現における $t = \frac{1}{2}$ での信号対雑音比と同じ値を持つのはどの時刻か。そのとき、一様にサンプリングされた訓練時刻のうち、信号が依然としてノイズを上回る領域に落ちる割合はどれだけか。
   3. [code] 潜在表現に $5c$ を掛けたもの（c の代わり）を用いて潜在 U-Net を訓練し、デコード後の FD を比較せよ。この結果を、解像度に伴うスケジュールのシフト（17.6.3.2 節）と関連づけて説明せよ。
2. **高解像度におけるトークン数**。Stable Diffusion のオートエンコーダは $512 \times 512 \times 3$ の画像を $64 \times 64 \times 4$ に写像する。
   1. 潜在表現上でサイズ 2 のパッチを用いた場合と、ピクセル上でサイズ 16 のパッチを用いた場合とで、DiT が扱うトークン数はそれぞれいくつか。また、この 2 つの場合において各トークンは何個の数値を保持するか。
   2. 幅 d の DiT ブロックは約 $12d^2$ 個の重みを持ち、1 訓練ステップには重みとトークンの組あたり約 6 FLOPs と、さらにアテンションのコストがかかる。サイズ 2 のパッチを用いた潜在表現上で、バッチサイズ 256 に対し幅 1,152・深さ 28 の DiT を 1 訓練ステップ実行するときの FLOPs を見積もれ。そのうちアテンションが占める部分はどの程度の大きさか。
   3. ピクセル上でサイズ 16 のパッチを用いた場合について同様の見積もりを行え。オートエンコーダは Transformer のステップあたりのコストをどれだけ節約するか。また、位置数に比例してコストが増大する U-Net の場合はどれだけ節約するか。
3. [code] **オートエンコーダの幅と重み**。レートと KL 重みはオートエンコーダの 2 つの設定項目である。
   1. 潜在チャネル数を 2、4、8 としたオートエンコーダをそれぞれ訓練し、各モデルの再構成 FD と、その潜在表現上で訓練した潜在 U-Net のデコード後の FD を報告せよ。2 つの数値のうちより大きく動くのはどちらか。また、潜在表現が大きくなるにつれて両者の差は縮まるか。この結果は、Stable Diffusion のオートエンコーダが 4 チャネルであることや、後続のモデルが 16 チャネル以上を用いていることについて何を示唆するか。
   2. 本節の摂動潜在表現テストを $\beta \in \{0, 10^{-2}\}$ に拡張し、潜在表現の標準偏差、エンコーダの標準偏差、および超過尖度もあわせて表示せよ。$\beta = 0$ では潜在表現のスケールを 1 に縛るものがなくなるが、それでもスケール係数 c によって利用可能になるのはなぜか。
   3. 新たな 2 つのオートエンコーダの両方について潜在 U-Net を訓練せよ。KL 重みが小さいほど良いサンプルが得られるという本文の順序関係は、$\beta = 0$ でも維持されるか。
4. [code] **なぜゼロから始めるのか**。ada 層のゼロ初期化（d2l.DiTBlock および d2l.DiT 内）を、フレームワークのデフォルトの初期化に置き換えよ。
   1. 両方の初期化について、未訓練ネットワークの出力の大きさを表示せよ。
   2. 両方の DiT-M モデルを 2,000 回の更新にわたり訓練し、損失をプロットせよ。初期化時点での恒等写像の性質を通じて、その違いを説明せよ。
5. **参照としてのクロスアテンション**。画像トークン $\mathbf{Y} \in \mathbb{R}^{m \times d}$ がキャプショントークン $\mathbf{X} \in \mathbb{R}^{n \times d}$ にクエリを送る状況を考える。クエリを $\mathbf{Q} = \mathbf{Y}\mathbf{W}_q$、キーを $\mathbf{K} = \mathbf{X}\mathbf{W}_k$、値を

$\mathbf{V} = \mathbf{X}\mathbf{W}_v$ とすると、この層は $\mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d})\,\mathbf{V}$ を出力する。

1. 各画像位置がキャプションの値ベクトルの凸結合を受け取ることを示せ。また、1 トークンからなる空のキャプションを用いた場合、すべての位置が同じベクトルを受け取り、したがってガイダンスにおける無条件予測にとってクロスアテンションが学習された定数シフトとして働くことを示せ。
2. 仮に値を画像側から $\mathbf{V} = \mathbf{Y}\mathbf{W}_v$ として計算したとする。$m \neq n$ のとき形状にどのような不整合が生じるか。また、$m = n$ のときそのような層は何を計算することになるか。
3. クロスアテンションではキャプションのキーと値は $\mathbf{x}_t$ に依存しないが、マルチモーダル DiT ではすべてのブロックでキャプショントークンが画像トークンによって更新される。77 トークンのキャプション、1,024 個の画像トークン、ガイダンス付き 50 ステップの条件において、クロスアテンションモデルがサンプルごとに 1 回だけ実行すれば済むキャプション側の計算はどれか。また、マルチモーダル DiT のトークン計算のうちキャプションが占める割合はどれだけか。

小問 (b) は CMU 10-423/623「Generative AI」Homework 4, Question 2.3（2026 年春学期）を改変したものである。

6. [code] **大きなパッチとクリーン画像のターゲット。**●`DiT` を `patch_size=7`（16 個の各トークンが 49 個のピクセル値を保持する設定）でピクセル上に訓練せよ。幅 32（ヘッド数 1）と幅 256 の 2 通りについて、それぞれ 10,000 回の更新を行うこと。さらに各幅に対し、同じ速度損失のもとで 2 通りの出力パラメータ化を与えよ。すなわち、ネットワークが速度を直接出力する場合と、$\hat{\mathbf{x}}_0$ を出力して速度を $(\mathbf{x}_t - \hat{\mathbf{x}}_0)/t$（ただし t の下限を 0.05 にクランプする）として構成する場合である。
   1. 4 つのモデルの FD を 2×2 の表にまとめて報告せよ。
   2. トークンが保持する値の個数がネットワークのチャネル数を上回る幅 32 において、クリーン画像の出力は有効か。また、幅 256 ではその差は縮まるか。
   3. 画像はピクセル数よりはるかに低次元の集合の近くに横たわる一方、ノイズはすべての次元を埋め尽くすという 17.6 節の議論を用いて結果を説明せよ。さらに Li and He (2026) の知見と比較せよ。

17.8 少数ステップ生成

拡散モデルは、ネットワークを何度も繰り返し呼び出すことでサンプルを生成する。17.5 節のサンプラは祖先サンプリング連鎖における 1,000 回の呼び出しを約 50 回に減らし、17.7 節の潜在モデルは 1 回あたりの呼び出しコストを下げたが、それでも 50 回は 50 回であり、GAN（第 16 章）の生成器に必要な呼び出しは 1 回だけであった。では、1 回の長いステップで済ませてはどうだろうか。これはうまくいかない。しかも、良いネットワークがないからではない。厳密な速度場（velocity field）を用いたとしても、確率フロー ODE（probability-flow ODE）の 1 回のオイラーステップはあらゆる出発点をデータの平均へと送り込んでしまうからである（17.3 節）。3 つの塊からなる混合分布では、その点は塊と塊の間のデータがまったく存在しない場所にある。

速度がサンプラに教えるのはどの方向へ進むかであり、1 ステップで生成するには最終的にどこへ到達するかを知る必要がある。そこで、常微分方程式（ordinary differential equation、ODE）の右辺ではなく解そのもの、すなわちある時刻の点からその軌跡がより早い時刻に位置する点への写像をネットワークに学習させる。Song et al. (2023) は、訓練済みの拡散モデルである**教師**（teacher）を新しいネットワークである**生徒**（student）へと蒸留することでデータへのジャンプを学習させ、その生徒を**一貫性モデル**（consistency model）と呼んだ。Geng et al. (2025) は、データのみから任意の区間にわたるジャンプを学習させ、この手法を **MeanFlow** と呼んだ。訓練は単一のネットワークによる回帰のままであり、同じノイズからは常に同じサンプルが得られるため、生徒と教師を画像ごとに比較できる。これに対し 16.7 節ではクリティックを用いて 1 ステップ生成器を訓練したが、その路線には本節の最後で立ち返る。まずは 1 回あるいは数回のオイ

ラーステップが何を誤るのかを確認し、続いて両手法に共通する 1 つの損失を導出して、これまでの例題と Fashion-MNIST で生徒を訓練する。

17.8.1 長いオイラーステップ

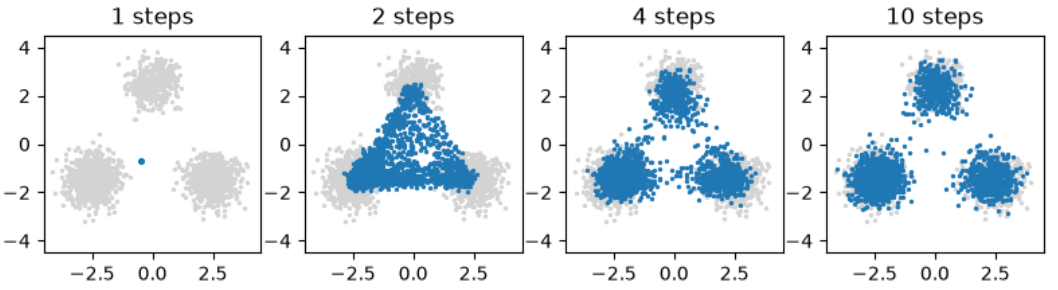
厳密な速度場において、長いオイラーステップが犯す誤差はただ 1 つ、軌跡が曲がっているのに接線に沿って進んでしまうことである。$t = 1$ ではこの誤差が最大になる。どの出発点からも速度はデータの平均を向いており、長さ 1 のステップはそこに着地する（17.3 節）。それ以降のステップはいずれかのモードへ向かい始めた点から出発するため狙いは定まってくるが、それでも各ステップは曲線を横切ってショートカットしてしまう。

17.8.1.1 厳密な速度場でのオイラーステップ

17.3 節では 1、10、100 ステップの結果を見たが、2 ステップと 4 ステップではどうなるだろうか。`mix.velocity` は混合分布の厳密な速度場を評価し、`d2l.make_pred` はそれを `d2l.ode_sample` が積分する予測器へと変換する。ここでは同じ 20,000 個のガウス初期点から、1、2、4、10 ステップでオイラー法を実行する。2 ステップと 4 ステップの行が、本節の生徒モデルが上回るべきベースラインとなる。

```
mix = d2l.GaussianMixture.three_modes()
lin = d2l.GaussianPath.linear()
pred_exact = d2l.make_pred(mix.velocity, lin, 'velocity')
z = jax.random.normal(jax.random.key(0), (20_000, 2))
euler = {n: d2l.ode_sample(pred_exact, z, lin.sigma_bars(n))
         for n in (1, 2, 4, 10)}
for n, x in euler.items():
    s = mix.summary(x)
    print(f'{n:2d} Euler steps: weights {s["weights"]}, '
          f'stray {s["stray"]:.1%}')
d2l.scatter_panels([x[:2000] for x in euler.values()],
                   [f'{n} steps' for n in euler],
                   data=mix.sample(jax.random.key(1), 2000), size=2.4);
```

```
 1 Euler steps: weights [1.0, 0.0, 0.0], stray 100.0%
 2 Euler steps: weights [0.644, 0.221, 0.135], stray 25.2%
 4 Euler steps: weights [0.54, 0.278, 0.182], stray 6.6%
10 Euler steps: weights [0.511, 0.293, 0.196], stray 2.3%
```



1 ステップではすべてのサンプルが平均 $(-0.5, -0.7)$ に集まり、`summary` はこれを逸脱サンプル（stray）として数えつつ、最も近い最大重みのモードに割り当てるため、重みは 1/0/0 となる。2 ステップでは各モードに到達するものの、サンプルの約 4 分の 1 がモード間に取り残され、重み

も本来の 0.5/0.3/0.2 に対して約 0.64/0.22/0.14 となる。最初の半ステップが依然として最大重みのモードに最も近い平均を狙うため、そのモードの割合が増えてしまうのである。4 ステップでも約 6.5%、10 ステップでも約 2% のサンプルが逸脱したままであるのに対し、データ自体の逸脱率は約 1% である。速度場が厳密であるため、この超過分はすべてステップ幅に起因する誤差である。

17.8.1.2 Reflow

そもそもなぜ軌跡は曲がるのだろうか。各訓練例はデータ点と独立に抽出されたノイズベクトルを直線分で結ぶが、異なる訓練例の線分どうしは交差する。交差する場所では速度場がそれらの方向を平均化するため、互いに交差できない軌跡は互いを避けるように曲がるのである（38.7.2 節）。この曲率は損失ではなく、ノイズとデータのペアリングから生じている。

Reflow (Liu et al. 2023) はこのペアリングを変更する。各ノイズベクトルを、訓練済みモデルがそこから到達するサンプルとペアにし、そのペアを用いてネットワークを再訓練するのである。新しい各線分は元のモデルの 1 本の軌跡に対する弦（chord）であり、元の軌跡どうしは決して交差しないため、それらの弦もめったに交差しない。38.7.2.3 節では、厳密な速度場を用いた Reflow がデータ分布を保ち、輸送コストを決して増加させないことを証明しており、2 つの三日月（two-moons）データでは Reflow 後のモデルの 1 回のオイラーステップが元のモデルの多数ステップと同等のスコアを達成する。

しかし、訓練された速度場ではこの保証が弱まる。新しいペアの終点は実データではなくモデルの生成サンプルであるため、Reflow 後のモデルは元のモデルの誤差をそのまま引き継ぐ。2 つの三日月データでは欠陥も含めて元のモデルのサンプルが再現され（38.7.2.3 節）、画像モデルでも同様に生成サンプルのあらゆる欠陥が受け継がれる。さらに 1 ラウンドごとに 1 回分の訓練コストがかかるうえ、Liu et al. (2023) における最良の 1 ステップ結果も結局は Reflow 後のモデルを蒸留して得られたものであった。そこで本節の残りでは、データのペアリング方法ではなく、ネットワークが何を出力するかの方を変更する。

17.8.2 フロー写像

速度モデルは軌跡がどちらを向いているかを表し、フロー写像（flow map）は軌跡がどこに到着するかを表す。

17.8.2.1 速度の積分

確率フロー ODE $d\mathbf{x}_\tau/d\tau = \mathbf{u}_\tau(\mathbf{x}_\tau)$ を、点 $\mathbf{x}$（時刻 t）からより早い時刻 $s \le t$ まで逆向きに走らせる。その到着点が**フロー写像** $\Phi_{t\to s}(\mathbf{x})$ である。速度場が $\mathbf{x}$ についてリプシッツ連続であれば、厳密なフロー写像は全単射（bijection）となり（38.1.1 節）、求めたいサンプラは $\Phi_{1\to 0}$ である。

軌跡 $\mathbf{x}_\tau$（$\mathbf{x}_t$ を通る解）に沿ってこの ODE を積分すると、フロー写像は次のように変位として表される。

$$\Phi_{t\to s}(\mathbf{x}_t) = \mathbf{x}_t - \int_s^t \mathbf{u}_\tau(\mathbf{x}_\tau)\, d\tau = \mathbf{x}_t - (t-s)\,\bar{\mathbf{u}}(\mathbf{x}_t, s, t), \tag{17.57}$$

ここで、$\bar{\mathbf{u}}(\mathbf{x}_t, s, t)$ は 2 つの時刻間における軌跡上の**平均速度**（average velocity）である。オイラー法はこれを始点での速度 $\mathbf{u}_t(\mathbf{x}_t)$ で置き換えるため、ステップサイズとともに増大する誤差が生じる（38.2 節）。長さ 0 の区間では $\bar{\mathbf{u}}(\mathbf{x}, t, t) = \mathbf{u}_t(\mathbf{x})$ となって両者は一致するため、$\bar{\mathbf{u}}$ を表すネットワークは、速度モデルに終点時刻というもう 1 つの入力を加えたものになる。

最小の例を見れば、瞬間速度にはわからず平均速度にはわかっていることが明らかになる。データを平均 m、標準偏差 ς の 1 次元ガウス分布とする。このとき x_t は平均 $(1-t)\,m$、標準偏差 $k_t = \sqrt{(1-t)^2\varsigma^2 + t^2}$ のガウス分布に従う。1 次元では軌跡どうしが追い越すことはできないため、フローは各点の分位点（quantile）を保ち、ガウス分布における分位点は標準化された位置 $(x-(1-t)m)/k_t$ によって決まる。したがって、フロー写像は次のアフィン写像になる。

$$\Phi_{t\to s}(x) = (1-s)\,m + \frac{k_s}{k_t}(x - (1-t)\,m). \tag{17.58}$$

$m=2$、$\varsigma=0.5$ とすると、全区間のジャンプ $\Phi_{1\to 0}(x) = 2 + 0.5\,x$ は標準ガウスノイズをデータ分布 $\mathcal{N}(2, 0.5^2)$ へと写し、その平均速度は $\bar{u}(x,0,1) = 0.5\,x - 2$ となる。始点での速度は $u_1(x) = x - 2$ であり、そのオイラーステップは定数 2 をあらゆる x に対して返す。データの広がりとしてノイズの広がりの半分を残す必要があるため、平均速度はノイズの半分しか取り除かないのである。

誰もその値を教えてくれないなかで、ネットワークはどうやって Φ を学習できるのだろうか。それは、真のフロー写像だけが満たす恒等式を損失へと変換することによって可能になる。軌跡に沿って 2 つの時刻を動かす方法ごとに 1 つの恒等式が存在する（Figure 17.14）。$\mathbf{x}_\tau$ を点 $\mathbf{x}$（時刻 t）を通る軌跡とし、$s \le r \le t$ とする。

合成（compositional）恒等式は、2 回のジャンプが 1 回のジャンプに等しいこと、すなわち $\Phi_{r\to s}(\Phi_{t\to r}(\mathbf{x})) = \Phi_{t\to s}(\mathbf{x})$ を表す。これに基づく損失は導関数を必要とせず、中間時刻におけるモデルの評価だけで済み、ターゲットの精度はそれを構成する短いジャンプの精度で決まる。漸進的蒸留（progressive distillation）の手法 (Salimans and Ho 2022) やショートカットモデル (Frans et al. 2025) がこれを用いている。

ラグランジュ型（Lagrangian）恒等式は、始点を固定して終点を動かす。すると写像された点は粒子のように速度場に従って動き、$\partial_s \Phi_{t\to s}(\mathbf{x}) = \mathbf{u}_s(\Phi_{t\to s}(\mathbf{x}))$ を満たす。これに基づく損失には、ジャンプの着地点における速度が必要になる。訓練例から条件付き速度（conditional velocity）が得られるのは自身の $\mathbf{x}_t$ においてのみであるため、着地点での速度は教師モデルか、あるいはフロー写像自身が供給しなければならない（演習問題 3）。

オイラー型（Eulerian）恒等式は、終点を固定して始点を軌跡に沿って滑らせる。1 本の軌跡上のどの点も同じ場所へと写像されるため、次式が成り立つ。

$$\frac{d}{dt}\Phi_{t\to s}(\mathbf{x}_t) = \partial_t \Phi_{t\to s}(\mathbf{x}_t) + \nabla_{\mathbf{x}} \Phi_{t\to s}(\mathbf{x}_t)\,\mathbf{u}_t(\mathbf{x}_t) = \mathbf{0}. \tag{17.59}$$

これに基づく損失が必要とするのは 1 回のヤコビ行列・ベクトル積（Jacobian-vector product）と始点での速度だけであり、始点には訓練例の速度が存在する。さらにターゲットに含まれるのは 1 組の時刻だけで、中間時刻もステップサイズも不要である。

ガウス分布の例では、3 つとも手計算で確かめられる。比 k_s/k_t は合成則を満たし、点 $\Phi_{t\to s}(x)$ を s を動かしながら追跡すると標準化された位置が保たれるためフローの点と同じように動き、さらにその標準化された位置は軌跡に沿って一定である。

写像が恒等写像となり $s = t$ において s に関する傾きが速度に一致するという条件と組み合わせれば、どの恒等式からもフロー写像が一意に定まる。違いは、損失が何を計算しなければならないかだけである。合成恒等式が最も単純であり、中間時刻もジャンプの着地点における速度も必要としないのはオイラー型恒等式だけであるが、その代わりに 1 回のヤコビ行列・ベクトル積を必要とする。一貫性モデルと MeanFlow はどちらもオイラー型恒等式を用いており、Boffi et al. (2025) は課す恒等式によって少数ステップ手法を分類している。

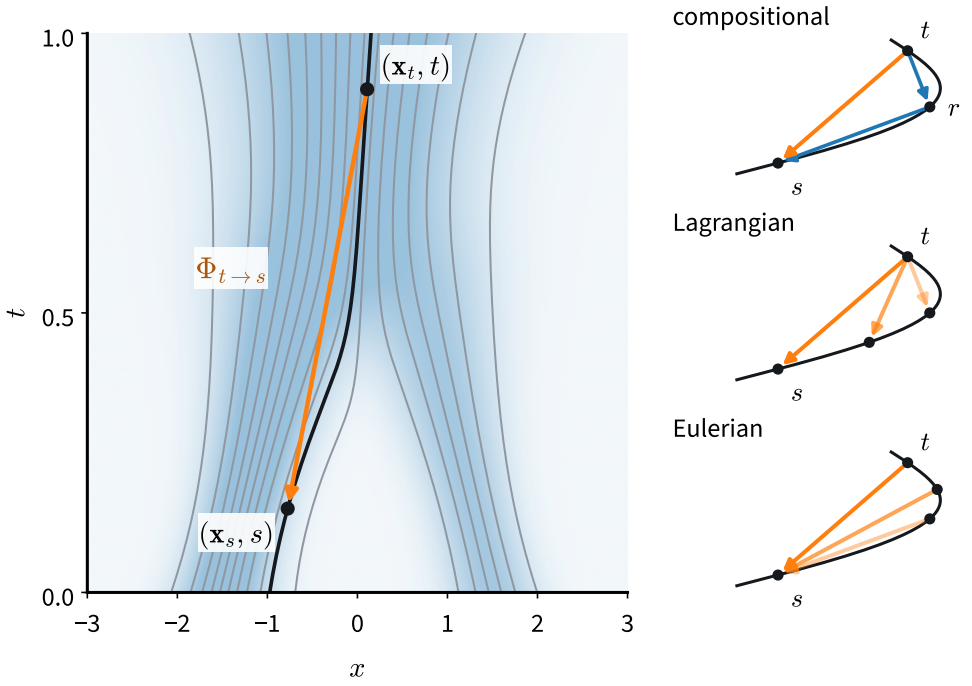


Figure 17.14: フロー写像。左：2 つのガウス分布からなる 1 次元混合分布に対する確率フロー ODE の軌跡であり、$t = 1$ のノイズ（上）から $t = 0$ のデータ（下）へと向かう。$(\mathbf{x}_t, t)$ から $(\mathbf{x}_s, s)$ への弦がフロー写像の 1 回のジャンプを表す。右（1 本の軌跡上）：r を経由する 2 回のジャンプが 1 回のジャンプになる（合成）、始点を t に固定して終点を動かす（ラグランジュ型）、終点を s に固定して始点を動かす（オイラー型）。

17.8.2.2 一貫性モデル

最初のフロー写像手法は、データへのジャンプのみを保持するものであった。**一貫性モデル** (Song et al. 2023) とは、ネットワーク $f(\mathbf{x}, t)$ を用いてあらゆる時刻で $\Phi_{t\to 0}(\mathbf{x})$ を出力するよう訓練したモデルである。その訓練条件は、入力が軌跡に沿って動いても出力が変化しないこと、すなわち $\frac{d}{dt} f(\mathbf{x}_t, t) = \mathbf{0}$ かつ $f(\mathbf{x}, 0) = \mathbf{x}$ であり、これはオイラー型恒等式 Equation 17.59 において $s = 0$ としたものにほかならない。

一貫性蒸留（consistency distillation）において、Song らは時間グリッドの隣接する時刻間で教師に 1 回の ODE ステップを踏ませ、その 2 点における生徒の出力が一致することを要請した。より早い時刻の点での出力がターゲットとなり、これは固定されたうえで生徒の重みの移動平均を用いて計算される。グリッドを細かくしていくと、この出力差をステップ幅で割ったものは上記の導関数に収束し、Lu and Song (2025) はこれを直接用いている。**一貫性訓練**（consistency training）は教師を必要とせず、教師の速度を各訓練例の直線分の速度 $\epsilon - \mathbf{x}_0$、すなわちフローマッチング（flow matching）の条件付き速度で置き換える。

一貫性モデルを 1 回評価すればサンプルが 1 つ得られる。しかし、このモデルは軌跡の終点しか知らないため、2 回目の評価を行う際に $t = 1$ から $t = 0.5$ へジャンプすることはできない。その代わりに**再ノイズ化**（re-noising）を行う。すなわち、新たなノイズを用いてサンプルを $t = 0.5$ へと戻し（$\mathbf{x}_{0.5} = 0.5\,\mathbf{x}_0 + 0.5\,\epsilon'$）、軌跡の曲がりがより小さいその点から再びデータへと写像するのである。始点と終点の両方の時刻を持つフロー写像であれば再ノイズ化は不要であり、2 回の評価は 1 本の軌跡に沿った 2 回のジャンプとなって、サンプルは初期ノイズのみに依存したまま保たれる。

17.8.2.3 平均速度と MeanFlow 恒等式

フロー写像を Equation 17.57 のように平均速度で表すと、オイラー型恒等式は回帰目標（regression target）へと変わる。平均速度に区間の長さを掛けて速度の積分とし、軌跡に沿って t で微分する（ただし s は固定する）。

$$\frac{d}{dt}\Big[(t-s)\,\bar{\mathbf{u}}(\mathbf{x}_t, s, t)\Big] = \frac{d}{dt}\int_s^t \mathbf{u}_\tau(\mathbf{x}_\tau)\,d\tau = \mathbf{u}_t(\mathbf{x}_t). \tag{17.60}$$

左辺に積の微分法則を適用すると、Geng et al. (2025) の **MeanFlow 恒等式**が得られる。

$$\bar{\mathbf{u}}(\mathbf{x}_t, s, t) = \mathbf{u}_t(\mathbf{x}_t) - (t-s)\,\frac{d}{dt}\bar{\mathbf{u}}(\mathbf{x}_t, s, t). \tag{17.61}$$

平均速度は、始点での速度に対して、始点が軌跡上を滑るときに平均速度が変化する速さで補正を加えたものになっている。Equation 17.57 を Equation 17.59 に代入すれば、2 つの恒等式が同一であることがわかる。全微分（total derivative）は $\partial_t \bar{\mathbf{u}} + (\nabla_{\mathbf{x}} \bar{\mathbf{u}})\,\mathbf{u}_t$ であり、これはネットワーク $(\mathbf{x}, t) \mapsto \bar{\mathbf{u}}(\mathbf{x}, s, t)$ と接ベクトル $(\mathbf{u}_t, 1)$ とのヤコビ行列・ベクトル積にほかならない。フォワードモード微分を用いれば、この量は出力と同時に逆伝播 1 回分程度のコストで計算できる。したがって、1 回の訓練ステップにかかるコストは、速度モデルのステップに比べて逆伝播約 1 回分多くなり、教師がある場合にはそこに教師の評価 1 回分が加わる。

ネットワーク自身から計算して固定した右辺へとネットワークを回帰させることで、この恒等式は損失になる。周辺速度（marginal velocity）$\mathbf{u}_t$ は直接利用できないが、2 つの代替手段がある。教師があれば、その推定値 $\hat{\mathbf{u}}$（$\mathbf{u}_t(\mathbf{x}_t)$ の近似）が得られる。教師がなければ、フローマッチングが機能したのと同じ理由により、訓練例の条件付き速度 $\epsilon - \mathbf{x}_0$ がその役割を果たす。ターゲットは直接的にもヤコビ行列・ベクトル積の接ベクトルを通じても速度に関してアフィンである。し

たがって、$\mathbf{x}_t$ を生じ得た訓練例全体で平均をとれば、条件付き速度を $\mathbf{u}_t(\mathbf{x}_t)$ に置き換えたターゲットに一致し、二乗誤差はその平均によって最小化される。用いる速度を $\mathbf{u}_{\text{ref}}$ と書き、これを**参照速度**（reference velocity）と呼ぶと、ある訓練点における損失は次のようになる。

$$\left\| \bar{\mathbf{u}}_\theta(\mathbf{x}_t, s, t) - \text{sg}\left[\mathbf{u}_{\text{ref}} - (t-s)\,\frac{d}{dt}\bar{\mathbf{u}}_\theta(\mathbf{x}_t, s, t)\right]\right\|^2, \tag{17.62}$$

ここで sg は**勾配停止**（stop-gradient）であり、損失を微分する際に定数として扱う量を表す。これは先ほどの平均化の議論に不可欠な操作である。その議論は、ネットワークが予測値のみを通じて損失に入る場合に限り成り立つからである。もしターゲット側まで微分を通せば、ネットワークは恒等式を満たす代わりにターゲット内部のヤコビ行列を変形させて訓練例間のターゲットのばらつきを変えることで、損失を下げられてしまう（演習問題 4）。勾配停止の呼び出し 1 つがあるかないかだけの違いしかないため、導出の全ステップのなかで再実装時に最も誤りやすい箇所がここである。

この呼び出しは `jax.lax.stop_gradient` である。

この損失には 2 つの選択肢が残されている。$\mathbf{u}_{\text{ref}}$ をどこから得るか、そして終点時刻 s を自由にするか 0 に固定するかである。主要な手法はこの 4 通りの組み合わせに分類される。ただし、この損失以外の方法で恒等式を課す手法もあり、それぞれに 17.10 節で述べる独自の工夫が加えられている。

Table 17.3: Diffusion Flow Map Methods

速度 $\mathbf{u}_{\text{ref}}$	1 時刻（$s = 0$）	2 時刻（$s \le t$）
教師の $\hat{\mathbf{u}}$	一貫性蒸留	フロー写像蒸留 (Kim et al. 2024)
訓練例の $\epsilon - \mathbf{x}_0$	一貫性訓練	MeanFlow

$s = 0$ では Equation 17.57 が $f(\mathbf{x}, t) = \mathbf{x} - t\,\bar{\mathbf{u}}(\mathbf{x}, 0, t)$ となるため、左上のセルは Lu and Song (2025) による連続時間形式の一貫性蒸留にあたる。`flow_map_error` は 4 手法すべてに対して線形経路上の訓練例ごとの誤差 Equation 17.62 を実装している。ネットワーク `u` が $\bar{\mathbf{u}}$ を計算し、`teacher=None` のときは条件付き速度を選び、呼び出し側が時刻の抽出と平均化を行う。

```
def flow_map_error(u, teacher, x0, eps, s, t):
    """Per-example squared error of the average velocity u(x_t, s, t) against
    the stop-gradient target v - (t - s) du/dt; v from teacher, or eps - x0."""
    x_t = d2l.GaussianPath.linear().sample(x0, t, eps)
    v = (eps - x0 if teacher is None
         else jax.lax.stop_gradient(teacher(x_t, t)))
    u_t, du_dt = jax.jvp(lambda x, r: u(x, s, r), (x_t, t),
                         (v, jnp.ones_like(t)))
    gap = (t - s).reshape(-1, *[1] * (x0.ndim - 1))
    target = jax.lax.stop_gradient(v - gap * du_dt)
    return ((u_t - target) ** 2).reshape(x0.shape[0], -1).sum(1)
```

t に関する微分はネットワークの時間特徴量を通過するため、フロー写像では本章の他のモデルよりも低周波の特徴量を用いる。特徴量 $\sin(\omega t)$ の導関数は $\omega\cos(\omega t)$ であるから、各特徴量はその周波数が掛かった状態でターゲットに入る。他のネットワークは t を最大 64 までの周波数で受け取るが（17.1 節）、フロー写像では `time_freqs=slow`、すなわち 10^{-4} から 1 までの周波数（下のコードで `slow` として定義）を渡す。高い周波数は導関数に大きな係数を生じさせ（たとえ

ば $1000\,t$ の埋め込みを用いるコードでは最大 1,000 倍される)、これは Lu and Song (2025) が一貫性モデルにおいて報告している不安定性の原因となる。

フロー写像は $\mathbf{x}_t$ を $\mathbf{x}_s$ へと 1 回のジャンプ $\mathbf{x}_s = \mathbf{x}_t - (t-s)\,\bar{\mathbf{u}}(\mathbf{x}_t, s, t)$ で写すため、少数ステップのサンプラは $[0,1]$ を等間隔の区間に分割し、各区間を順にジャンプする。

```
def flow_map_sample(u, x, steps, t_start=1.0):
    """Jump from x at t_start (1 is noise) to data at t = 0 in equal steps."""
    ts = jnp.linspace(t_start, 0, steps + 1)
    def jump(x, ts):
        t, s = ts
        n = x.shape[0]
        return x - (t - s) * u(x, jnp.full(n, s), jnp.full(n, t)), None
    return jax.lax.scan(jump, x, (ts[:-1], ts[1:]))[0]
```

速度 $\mathbf{u}_t$ を $\bar{\mathbf{u}}$ の代わりに渡せば同じ関数がそのままオイラー法になるため、1 つのサンプラを教師と生徒の両方に使える。ただし、ジャンプは一方向のみである。本節のすべての損失は $s \le t$ として時刻を引くため、17.5 節で ODE を逆向きに解いたようにサンプルを反転するには、逆方向にも写像を訓練する必要がある。

次節末尾の画像実験では、生徒を教師の U-Net のコピーから出発させる。時間幅 $t-s$ は専用の時間埋め込みとゼロ初期化した線形変換を通して U-Net の埋め込みに加算されるため、訓練開始時点では $\bar{\mathbf{u}}(\mathbf{x}, s, t) = \hat{\mathbf{u}}(\mathbf{x}, t)$ となって教師の速度と一致する。

```
class UNetFlowMap(nnx.Module):
    """u(x, s, t): a velocity U-Net, plus a zero-initialized input for the
    gap t - s. The caller passes a copy of the teacher's U-Net."""
    def __init__(self, unet, *, rngs):
        self.unet = unet
        zeros = nnx.initializers.zeros
        self.gap = nnx.Linear(unet.emb_dim, unet.emb_dim, kernel_init=zeros,
                              bias_init=zeros, rngs=rngs)

    def __call__(self, x, s, t):
        gap = self.gap(d2l.time_embedding(t - s, self.unet.emb_dim,
                                          self.unet.time_freqs))
        return self.unet(x, t, cond=gap)
```

17.8.3 フロー写像の訓練

17.8.3.1 3 モード混合分布

この損失は実際に機能するのだろうか。また、参照速度をどこから得るかは結果に影響するのだろうか。この混合分布ならその両方に答えられる。厳密な速度場が既知であるため、完璧なサンプラがどのような数値を出力するかがわかるからである。教師は 17.2 節と同様に訓練した速度モデル `TimeMLP` であり、入力点と t の正弦特徴量を幅 256 の層に通すネットワークを、ここでは 20,000 ステップ訓練する。

```
_, teacher, _ = d2l.train_regression(
    d2l.TimeMLP(256, 3, rngs=nnx.Rngs(1)),
    d2l.regression_loss(lin, 'velocity'), lambda k: mix.sample(k, 512),
```

```
    20_000, jax.random.key(1), cosine=True)
```

3 種類の生徒が Table 17.3 の 3 つのセルを埋める。いずれも `TimeMLP` として始点 t と区間幅 $t - s$ を受け取り、単一実行の偶然と手法固有の性質を見分けられるよう、教師と同じ設定でそれぞれ 6 回ずつ（乱数シード 2〜7）訓練する。一貫性モデルの生徒は t を一様に引き、$s = 0$ に固定して教師の速度を用いる。フロー写像蒸留（flow-map distillation）の生徒は一様乱数で 2 つの時刻を引いて小さい方を s とし、同様に教師の速度を用いる。MeanFlow の生徒も同じ方法で時刻を引くが、教師は用いない。4 つ目のセルである一貫性訓練には専用の生徒を用意する必要がない。その 2 つの選択（1 時刻と訓練例の速度）は、それぞれ他の生徒によって検証されるからである。

損失はどの程度の値になると予想されるだろうか。蒸留される生徒のターゲットは位置と時刻のみの関数であるため、その損失は 0 に近づき得る。一方、MeanFlow のターゲットには各訓練例固有の速度が $\mathbf{u}_{\text{ref}}$ として 1 回、接ベクトルの方向としてもう 1 回の計 2 回含まれるため、同じ位置と時刻における訓練例間のターゲットのばらつき（分散）より下に損失が下がることはない。

```
class MLPFlowMap(d2l.TimeMLP):
    """u(x, s, t): a TimeMLP on the start t and the interval t - s."""
    def __call__(self, x, s, t):
        return super().__call__(x, t, t - s)

slow = (1e-4, 1.0)  # time features of a flow map, whose loss has a d/dt

def consistency_loss(u, x0, key):  # s = 0, the teacher's velocity
    k1, k2 = jax.random.split(key)
    t = jax.random.uniform(k1, (len(x0),))
    return d2l.flow_map_error(u, teacher, x0, jax.random.normal(k2, x0.shape),
                              jnp.zeros_like(t), t).mean()

def distillation_loss(u, x0, key):  # s < t, the teacher's velocity
    k1, k2 = jax.random.split(key)
    a, b = jax.random.uniform(k1, (2, len(x0)))
    s, t = jnp.minimum(a, b), jnp.maximum(a, b)
    return d2l.flow_map_error(u, teacher, x0, jax.random.normal(k2, x0.shape),
                              s, t).mean()

def meanflow_loss(u, x0, key):  # s < t, no teacher
    k1, k2 = jax.random.split(key)
    a, b = jax.random.uniform(k1, (2, len(x0)))
    s, t = jnp.minimum(a, b), jnp.maximum(a, b)
    return d2l.flow_map_error(u, None, x0, jax.random.normal(k2, x0.shape),
                              s, t).mean()

students = {}
for name, loss in (('consistency', consistency_loss),
                   ('distillation', distillation_loss),
                   ('MeanFlow', meanflow_loss)):
    final = []
    for seed in range(2, 8):
        _, students[name, seed], losses = d2l.train_regression(
            MLPFlowMap(256, 3, num_times=2, time_freqs=slow,
                       rngs=nnx.Rngs(seed)),
            loss, lambda k: mix.sample(k, 512), 20_000, jax.random.key(seed),
```

```
            cosine=True)
        final.append(float(losses[-1000:].mean()))
    print(f'{name:>12}, seeds 2 to 7: mean loss of the last 1,000 updates '
          f'{min(final):.3f} to {max(final):.3f}')
```

```
 consistency, seeds 2 to 7: mean loss of the last 1,000 updates 0.063 to 0.079
distillation, seeds 2 to 7: mean loss of the last 1,000 updates 0.031 to 0.034
    MeanFlow, seeds 2 to 7: mean loss of the last 1,000 updates 78.772 to 85.188
```

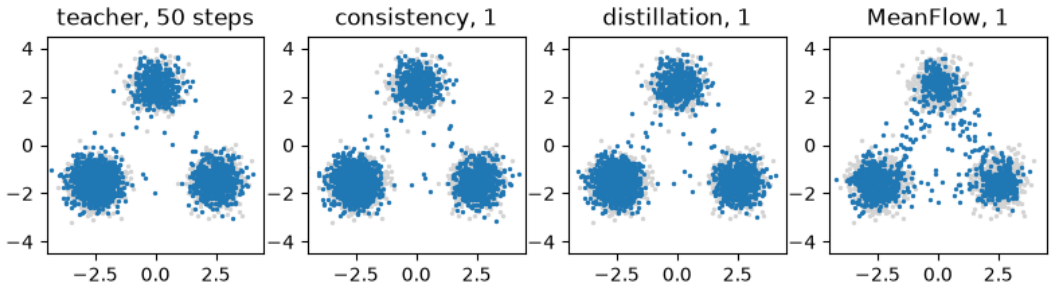
蒸留された生徒の損失は 0 付近で収束し、2 時刻の生徒で約 0.03、一貫性モデルの生徒で 0.06〜0.08 となる。これに対し MeanFlow は 6 回の実行で 76〜85 となり、一貫性モデルの生徒の約 1,000 倍の損失で終わる。その大部分は、いかなるネットワークも取り除けない訓練例間の不一致、すなわち損失の下限（フロア）である可能性がある。このコードセルではその下限を測定していないため、損失の数値自体から生徒の出来はほとんどわからない。MeanFlow の実行結果は損失ではなく生成サンプルによって評価する。

次に、すべてのモデルをオイラー法の実験と同じ 20,000 個の初期点から出発させる。教師は 50 オイラーステップ、一貫性モデルの生徒は 1 回ジャンプまたは $t = 0.5$ で再ノイズ化する 2 回ジャンプ、2 時刻の生徒は 1、2、4 回ジャンプで評価する。混合分布から新たに抽出したサンプルは、完璧なサンプラが出力する基準値を示す。表の各行には、まず各実行における 3 つのモード重みの最大誤差が並ぶ（生徒はシード順に 6 個、データと教師は 1 個）。続いて実行間の範囲として、すべてのモード中心から標準偏差の 3 倍を超えて離れたサンプルの割合である逸脱率と、逸脱しなかったサンプルから所属モードの中心までの平均二乗距離である広がりを示す（17.1 節）。

```
def renoised_sample(u, z, key):
    """Two evaluations of a one-time map: jump, re-noise to t = 0.5, jump."""
    x = d2l.flow_map_sample(u, z, 1)
    half = jnp.full((len(z),), 0.5)
    return d2l.flow_map_sample(u, lin.sample(x, half, jax.random.normal(
        key, x.shape)), 1, t_start=0.5)

def span(values, fmt):
    """'lo-hi' over the runs, or the one value if they print alike."""
    lo, hi = format(min(values), fmt), format(max(values), fmt)
    return lo if lo == hi else f'{lo}-{hi}'

def report(name, xs):
    """Each run's largest weight error; the range of strays and spreads."""
    stats, w_true = [mix.summary(x) for x in xs], mix.weights.tolist()
    err = ' '.join(
        f'{max(abs(w - w0) for w, w0 in zip(s["weights"], w_true)):.3f}'
        for s in stats)
    print(f'{name:>24}  {err:35}  {span([s["stray"] for s in stats], ".1%"):9}'
          f'  {span([s["spread"] for s in stats], ".3f")}')

samples = {'data': [mix.sample(jax.random.key(3), len(z))],
           'teacher, 50 steps': [d2l.flow_map_sample(
               lambda x, s, t: teacher(x, t), z, 50)]}
for (name, seed), u in students.items():
    if name == 'consistency':  # a one-time map re-noises for a second step
        key = jax.random.fold_in(jax.random.key(4), seed)
        new = {'1': d2l.flow_map_sample(u, z, 1),
```

```
                '2 re-noised': renoised_sample(u, z, key)}
    else:
        new = {str(k): d2l.flow_map_sample(u, z, k) for k in (1, 2, 4)}
    for k, x in new.items():
        samples.setdefault(f'{name}, {k}', []).append(x)
print(f'{"":24}  {"largest weight error, each run":35}  {"stray":9}  spread')
for name, xs in samples.items():
    report(name, xs)
names = ['teacher, 50 steps', 'consistency, 1', 'distillation, 1',
         'MeanFlow, 1']
d2l.scatter_panels([samples[k][0][:2000] for k in names], names,
                   data=mix.sample(jax.random.key(5), 2000), size=2.4);
```

```
                         largest weight error, each run       stray      spread
                   data  0.004                                1.1%       0.474
        teacher, 50 steps  0.003                                1.4%       0.464
          consistency, 1  0.005 0.004 0.004 0.004 0.004 0.004  2.0%       0.459-0.461
consistency, 2 re-noised  0.005 0.004 0.003 0.006 0.005 0.005  1.2%-1.4%  0.469-0.482
         distillation, 1  0.004 0.004 0.004 0.004 0.005 0.004  1.7%-1.9%  0.448-0.461
         distillation, 2  0.005 0.006 0.005 0.004 0.005 0.005  1.0%-1.1%  0.468-0.475
         distillation, 4  0.005 0.004 0.006 0.005 0.006 0.004  0.9%       0.460-0.470
             MeanFlow, 1  0.004 0.005 0.002 0.001 0.002 0.004  3.9%-4.2%  0.318-0.342
             MeanFlow, 2  0.004 0.012 0.004 0.002 0.005 0.006  1.1%-1.7%  0.361-0.399
             MeanFlow, 4  0.005 0.020 0.007 0.003 0.013 0.005  0.6%-1.1%  0.352-0.394
```



1 回の評価後、どの生徒のどの実行でも、サンプルは誤差 0.007 以内で 0.5/0.3/0.2 の重みをもつ 3 つのモード上に配置される。新たに抽出した 20,000 点でも 0.004〜0.005 のずれが生じるため、1 回のジャンプでサンプリングノイズと同程度の精度まで重みを再現できており、蒸留した 2 時刻の生徒は 2 回および 4 回のジャンプでもその精度を維持する。新規抽出データの逸脱率は各ガウス分布の標準偏差の 3 倍を超える裾に相当する約 1% であり、50 ステップの教師では約 1.5% である。蒸留した 2 種類の生徒はいずれも 1 回の評価後の逸脱率が約 2% で、広がりも 0.45〜0.46（データは 0.47）とデータの広がりを保っている。2 回目の評価を行うと、再ノイズ化する場合（1.2%〜1.5%）でも再ジャンプする場合（1.0%〜1.1%）でも、逸脱率は教師と同等以下まで下がる。この混合分布では、再ノイズ化による悪化はこれらの数値からは見られない。2 回目のジャンプが保持しているのは、各サンプルと初期ノイズとの対応関係である。一方、MeanFlow は 1 回ジャンプ後の逸脱率が約 4% であり、図のパネルからはもう 1 つの欠陥も見て取れる。モードが狭すぎるのである。広がりは 1 回ジャンプ後で 0.31〜0.34、2 回および 4 回ジャンプ後でも 0.35〜0.40 にとどまる。4 回ジャンプ時の逸脱率がデータ自体と同程度に低いのも、このモード収縮が理由である。

6 回の実行を比較すると、1 回だけの実行では見誤る点も浮かび上がる。蒸留された生徒の重み誤差は、すべての実行において 2 回および 4 回のジャンプで 0.006 以内に収まる。しかし MeanFlow

はそうではない。4 回ジャンプ後には 6 回中 1〜2 回の実行で誤差が 0.013〜0.023（新規抽出のずれの 2〜6 倍）に達する一方、残りの実行では 0.010 以内に収まる。このように、複数回ジャンプ後の MeanFlow の重みは訓練実行ごとにばらつくため、1 回の実行結果だけでは判断できないが、蒸留された生徒にはそのような依存性がない。

このタブで重みがずれているのはシード 3 と 6（表の 2 列目と 5 列目）の実行であり、4 回ジャンプ後にそれぞれ 0.020 と 0.013 ずれている。シード 3 の実行は 2 回ジャンプの時点ですでに 0.012 ずれている。

この比較から欠陥の原因が特定できる。2 時刻の 2 つの生徒はネットワーク、時刻のペア、更新回数をすべて共有しており、ターゲット内部の速度だけが異なる。それにもかかわらず、両者の広がりの範囲は一度も重ならない。すべての蒸留実行はすべての MeanFlow 実行よりも、1 回ジャンプ後で 0.1 以上、2 回および 4 回ジャンプ後でも少なくとも 0.06 上回っているのに対し、同一手法の 6 回の実行間の幅は高々 0.013（蒸留）および約 0.04（MeanFlow）にすぎない。したがってこの実験において、モードの収縮は訓練例の速度で訓練したことに起因している。ノイジーなターゲットで訓練した結果、ノイジーな写像ではなく狭すぎる写像に行き着く理由までは、この実験からは断定できない。Equation 17.61 の補正項を一部しか学習できなかった生徒は部分的にオイラーステップとして振る舞い、オイラー法のストライドはまさにこのようにモードを収縮させるが（17.5 節）、これはもっともらしい解釈であって証明された結論ではない。少なくともこの混合分布においては教師を用意する価値がある。データのみから訓練すれば教師の訓練は省けるが、生徒ははるかにノイジーなターゲットを平均化しなければならなくなる。

17.8.3.2 Fashion-MNIST

画像の教師モデルは、無条件の速度 U-Net である `UNet(ch=64)`（パラメータ数は約 120 万個）であり、生徒がそのコピーから出発するため、低周波の時間特徴量を用いて 10,000 回更新して訓練する。`train_regression` は訓練後のネットワークとともに重みの移動平均を返し、その平均化されたネットワークを教師とする。

```
X, y, X_test, y_test = d2l.load_fashion_mnist_pm1()

def batch_fn(key):
    return X[jax.random.randint(key, (128,), 0, len(X))]

_, teacher, _ = d2l.train_regression(  # the student copies these time features
    d2l.UNet(ch=64, time_freqs=slow, rngs=nnx.Rngs(0)),
    d2l.regression_loss(lin, 'velocity'), batch_fn, 10_000, jax.random.key(0))
```

1,000 枚の生成サンプルを、17.4 節のフレシェ距離（Fréchet distance、FD）を用いて 1,000 枚のテスト画像と比較しスコア化する。これは Fashion-MNIST 分類器が両画像集合から抽出した特徴量に当てはめたガウス分布間の距離であり、互いに素な 1,000 枚ずつのテスト画像 2 組の間で計算した下限を基準に読み取る。

```
cnn = d2l.fashion_feature_cnn(X, y)
features = lambda imgs: d2l.fd_features(cnn, imgs)
f_test = features(X_test[:1000])
floor = d2l.fid_score(features(X_test[1000:2000]), f_test)
print(f'floor, 1,000 real images against 1,000 others: FD {floor:.2f}')
```

```
floor, 1,000 real images against 1,000 others: FD 0.83
```

生徒は先ほど定義した `UNetFlowMap` に教師のコピーを渡したものであり、教師の速度を用い、2 つの時刻をともに自由に動かして訓練する。時刻ペアの半分では $s = t$ とし、ターゲットを教師の速度そのものにする。生徒は教師から出発するため、ジャンプを学習するあいだに瞬間速度を忘れないようにつなぎ留める狙いがある。残りの半分では混合分布のときと同様に $s < t$ となるよう時刻を引く。訓練済みの重みから出発し、学習率 $2 \cdot 10^{-4}$ で 5,000 回の更新（1 回あたり 128 枚、教師の 10,000 回の半分）にわたって学習させる。結果がどの程度実行ごとのばらつきによるものかを確かめるため、シード 2 と 3 を用いて同じ教師から生徒を 2 つ訓練する。

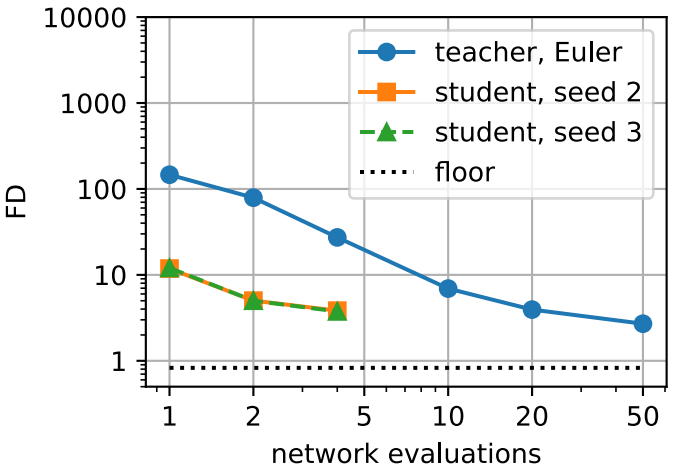
```
def student_loss(u, x0, key):
    k1, k2, k3 = jax.random.split(key, 3)
    a, b = jax.random.uniform(k1, (2, len(x0)))
    s, t = jnp.minimum(a, b), jnp.maximum(a, b)
    # half the pairs have s = t: the teacher's velocity as target
    s = jnp.where(jax.random.uniform(k2, (len(x0),)) < 0.5, t, s)
    return d2l.flow_map_error(u, teacher, x0, jax.random.normal(k3, x0.shape),
                              s, t).mean()

image_students = {}
for seed in (2, 3):
    _, image_students[seed], _ = d2l.train_regression(
        d2l.UNetFlowMap(nnx.clone(teacher), rngs=nnx.Rngs(seed)),
        student_loss, batch_fn, 5_000, jax.random.key(seed), lr=2e-4,
        ema_half_life=500)
```

同じ 1,000 枚のノイズ画像から、教師は 1〜50 オイラーステップ、各生徒は 1、2、4 回のジャンプを行い、ネットワーク評価回数に対して FD をプロットする。教師の 1 ステップは平均画像に着地するはずであり、これは混合分布での失敗のピクセル版にあたるため、そこでの FD は他のすべての点を圧倒的に上回るはずである。

```
noise = jax.random.normal(jax.random.key(3), (1000, 28, 28, 1))
velocity = lambda x, s, t: teacher(x, t)
ns_teacher, ns_student = (1, 2, 4, 10, 20, 50), (1, 2, 4)
images = {('teacher', n): d2l.flow_map_sample(velocity, noise, n)
          for n in ns_teacher}
images |= {(f'student {seed}', n): d2l.flow_map_sample(u, noise, n)
           for seed, u in image_students.items() for n in ns_student}
fd = {key: d2l.fid_score(features(x), f_test) for key, x in images.items()}
print('FD at evaluations' + ''.join(f'{n:8d}' for n in ns_teacher))
for name in ['teacher', *(f'student {seed}' for seed in image_students)]:
    print(f'{name:17}' + ''.join(f'{fd[name, n]:8.2f}' for n in ns_teacher
                                  if (name, n) in fd))
d2l.plot([ns_teacher, ns_student, ns_student, [1, 50]],
         [[fd['teacher', n] for n in ns_teacher],
          *[[fd[f'student {seed}', n] for n in ns_student]
            for seed in image_students], [floor, floor]],
         'network evaluations', 'FD', ylim=(0.5, 1e4), xscale='log',
         yscale='log', fmts=('o-', 's-', '^--', 'k:'),
         legend=['teacher, Euler', 'student, seed 2', 'student, seed 3',
                 'floor'], legend_loc='upper right')
```

```
FD at evaluations       1       2       4      10      20      50
teacher            146.34   79.09   27.29    6.93    3.94    2.70
```

```
student 2           11.84    5.01    3.83
student 3           12.04    5.00    3.75
```



教師の FD は、1 オイラーステップでの約 145〜170、2 ステップでの約 80 から、10 ステップで約 7、50 ステップで約 2.6 まで低下する（下限は約 0.8）。少ない評価予算では、生徒はこの曲線のずっと下に位置する。1 回ジャンプのスコアは 11〜12（教師の 50 ステップ FD の 4〜5 倍）、2 回ジャンプはほぼ 2 倍、4 回ジャンプは 1.4〜1.5 倍となり、教師が 20 ステップで到達する水準に迫る。プロットを横方向に読むと、生徒の 2 回ジャンプは教師の 10〜20 オイラーステップの間に、1 回ジャンプは 4〜10 ステップの間に相当する。2 つの生徒はどの評価予算でも FD の 10 分の 1 以内で一致しているため、これらの結果はどちらの生徒にも当てはまる。

蒸留された生徒にできるのは最良でも教師の ODE を再現することまでであるため、その到達目標は下限ではなく教師の厳密な ODE がもつ FD（50 オイラーステップが上から近づく値）である。残る差のうちどれだけが訓練量に起因するのかは、今回の実行からはわからない。各生徒が見た画像は延べ 640,000 枚であるが、公開されている論文の生徒モデルははるかに多くの画像で訓練されており、1 ステップで教師にずっと近い性能に達している (Song et al. 2023; Lu and Song 2025)。

下の各行は同じ 8 枚のノイズ画像から生成したものであり、上から 50 オイラーステップの教師、2 オイラーステップの教師、そしてシード 2 の生徒による 1 回ジャンプと 2 回ジャンプである。

```
rows = [('teacher', 50), ('teacher', 2), ('student 2', 1), ('student 2', 2)]
d2l.show_gray(jnp.concatenate([images[r][:8].clip(-1, 1) for r in rows]),
              4, 8);
```

各行は列ごとに対応している。教師の 2 オイラーステップは衣類の灰色の平均像を描き出すにとどまり（混合分布での失敗のピクセル版）、50 ステップが描く衣類の種類にすらなっていない列もある。これに対し生徒は、同じノイズからわずか 1 回の評価で、ほぼすべての列において教師と同じ衣類を描き出す。1 回のジャンプで失われるのは細部であり、柄や縞模様がにじみ、輪郭がところどころギザギザになり、わずかな染みが現れる。2 回目のジャンプを行うと、ほとんどの列が教師の 50 ステップに近い仕上がりになる。

17.8.4 分布マッチング

フロー写像の性能は、学習元となる軌跡によって上限が決まる。蒸留された生徒は教師の ODE によって制限され、データのみから訓練された生徒は厳密な速度場をどれだけうまく学習できるかによって制限される（混合分布では広がりの再現が不十分であった）。

もう 1 つの路線は、軌跡ではなく分布に一致させるように 1 ステップ生成器 $g_\theta(\mathbf{z})$ を訓練する方法である。このような生成器を二乗誤差で教師のサンプルに当てはめると、平均ぼけした画像を描いてしまう。教師の長い計算を 1 回の順伝播で再現できないネットワークにとっては複数のノイズベクトルが区別しづらく見え、二乗誤差のもとではそれらの画像の平均が最適になるからである（16.7 節）。同節では 2 つの解決策も示した。実画像に対するクリティックを用いる方法 (Sauer, Lorenz, et al. 2023) と、クリティックを必要としない分布マッチング蒸留（distribution matching distillation）である (Yin, Gharbi, Zhang, et al. 2024)。後者は生成器のサンプルにノイズを加えて $\mathbf{x}_t = (1-t)\, g_\theta(\mathbf{z}) + t\,\epsilon$ とし、その分布 $q_{\theta,t}$ とノイズ付きデータ分布 p_t とのカルバック・ライブラー・ダイバージェンスを下げる。その勾配は次のようになる。

$$\begin{aligned}&\nabla_\theta D_{\mathrm{KL}}(q_{\theta,t} \,\|\, p_t) \\ &\qquad = \mathbb{E}\Big[\big(\nabla \log q_{\theta,t}(\mathbf{x}_t) - \nabla \log p_t(\mathbf{x}_t)\big)^\top \frac{\partial \mathbf{x}_t}{\partial \theta}\Big].\end{aligned} \tag{17.63}$$

この勾配は生徒のスコアから教師のスコアを引いたものであるため、降下ステップはノイズを加えた各サンプルを、生徒が割り当てている以上にデータの確率質量が多い領域へと動かす。線形経路上ではスコアが速度のアフィン関数になるため教師が $\nabla \log p_t$ を供給でき（17.2 節）、生徒自身のサンプルを用いて並行して訓練する第 2 の拡散モデルが $\nabla \log q_{\theta,t}$ を供給する（Figure 17.15）。このような生徒の上限は教師の軌跡ではなく教師の分布によって決まる。敵対的蒸留におけるク

リティックや、分布マッチングにクリティックを追加する手法 (Yin, Gharbi, Park, et al. 2024) のように実画像を損失に含めた場合に限り、生徒は教師を上回ることができる（16.8 節）。どちらの変種も ODE の軌跡に沿って訓練されるわけではないため、あるノイズベクトルに対する生成器のサンプルは教師のサンプルと一致するとは限らず、ODE を介した反転や編集はもはや適用できない。

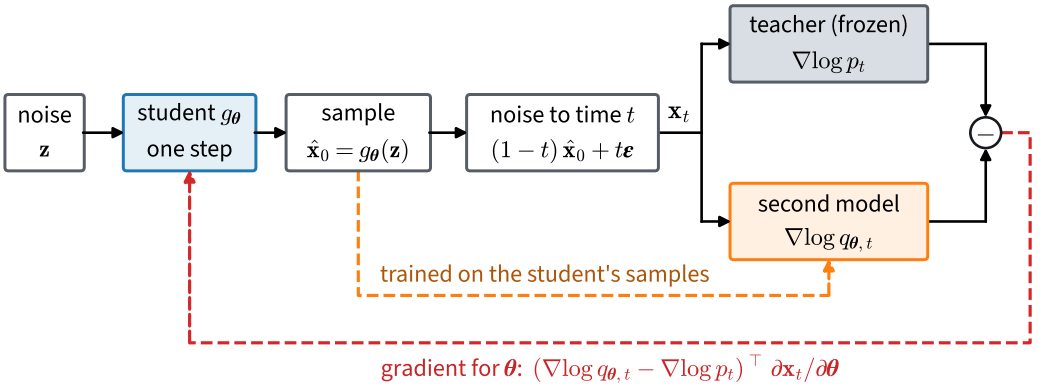


Figure 17.15: 分布マッチング蒸留。生徒はノイズ $\mathbf{z}$ を 1 ステップでサンプルへと写し、そのサンプルに時刻 t までノイズを加える。凍結された教師はノイズ付きデータのスコア $\nabla \log p_t$ を $\mathbf{x}_t$ において供給する。生徒のサンプル上で並行して訓練される第 2 の拡散モデル（オレンジ）は、ノイズを加えた生成サンプルのスコア $\nabla \log q_{\theta,t}$ を供給する。2 つのスコアの差が生徒に伝わる勾配となる（赤）。

どの手法を選ぶべきかは計算予算から決まる。50 回の評価が可能であれば、優れたソルバと組み合わせた教師モデルで十分であり、2 回目の訓練は不要である。レイテンシが重要な場合には、教師から 2 時刻のフロー写像を蒸留するとよい。Fashion-MNIST では 2 回のジャンプで教師の 10〜20 オイラーステップに相当する品質が得られた。教師がない場合、MeanFlow はデータのみからフロー写像を訓練できるが、混合分布において本節で用いた単純な二乗誤差ではモードが狭くなりすぎた（公開されたレシピでは損失に重み付けを導入している。演習問題 4）。クリティックの有無にかかわらず分布マッチングも 1 ステップ生成を実現するが、反転や編集に必要なノイズからサンプルへの対応写像は失われる。

いずれの路線も同じ対象、すなわち速度やスコアが定義される連続的な経路から出発している。しかし、$\mathbf{x}$ がトークンの系列である場合にはそのどちらも存在しない。これが 17.9 節の主題である。

17.8.5 まとめ

確率フロー ODE を $t = 1$ から $t = 0$ まで 1 回のオイラーステップで進めると、曲がった軌跡の初期接線に沿って進むため、すべての出発点がデータの平均 $\mathbb{E}[\mathbf{x}_0]$ へと潰れてしまう。これに対しフロー写像 $\Phi_{t\to s}(\mathbf{x}_t) = \mathbf{x}_t - (t-s)\,\bar{\mathbf{u}}(\mathbf{x}_t, s, t)$ は、平均速度 $\bar{\mathbf{u}}$ を介して ODE 軌跡に沿った積分変位を予測するものであり、これを t で微分するとオイラー型恒等式すなわち MeanFlow 恒等式 $\bar{\mathbf{u}} = \mathbf{u}_t - (t-s)\,\frac{d}{dt}\bar{\mathbf{u}}$ が得られ、1 回のフォワードモードによるヤコビ行列・ベクトル積と勾配停止ターゲットを用いて訓練できる。参照速度 $\mathbf{u}_{\text{ref}}$ の選択（事前学習済みの教師 $\hat{\mathbf{u}}$ か条件付き速度 $\boldsymbol{\epsilon} - \mathbf{x}_0$ か）と時間領域の選択（$s = 0$ か $s \le t$ か）を組み合わせることで、一貫性蒸留、一貫性訓練、フロー写像蒸留、および MeanFlow が統一的に理解できる。3 モード混合分布および Fashion-MNIST において、教師からのフロー写像蒸留はモード内の広がりを保ちつつ教師の

10〜20 オイラーステップに相当する品質を生徒の 2〜4 回のジャンプで達成する一方、ノイジーな訓練例ごとの速度に対して単純な二乗誤差で訓練した教師なし MeanFlow は混合分布のモードを収縮させる。

17.8.6 演習問題

1. **1 ステップと 2 ステップのオイラー法。**
   1. 純粋なノイズから $t = 0$ への決定論的サンプラの 1 ステップは、経路によらずどの出発点からも $\mathbb{E}[\mathbf{x}_0]$ を返す（17.3 節）。t に関する 1 回のオイラーステップ $\mathbf{x} - \mathbf{u}_1(\mathbf{x})$ をコサイン経路 $\alpha_t = \cos\frac{\pi t}{2}$、$\sigma_t = \sin\frac{\pi t}{2}$ 上でとった場合にも同じことが成り立つか。このステップは何を返すか。また、t に関するオイラーステップが経路に依存する一方で、$\bar{\sigma}$ に関するステップが経路に依存しないのはなぜか。
   2. 線形経路上の 1 次元データを、等確率の 2 点 $\pm a$ とする。$\mathbb{E}[x_0 \mid x_t = x] = a\tanh\left(a(1-t)x/t^2\right)$ となること、および長さ $\frac{1}{2}$ の 2 回のオイラーステップが初期点 x を $a\tanh(ax)$ へ写すことを示せ。この例の厳密なフローは 38.7.1.4 節にある。
   3. $a = 2$ のとき、2 ステップのサンプルのうちデータ点から距離 0.1 以内に入る割合はいくらか。また、厳密なフロー写像 $\Phi_{1\to 0}$ は x をどこへ送るか。
2. [code] **混合分布における Reflow。**●Reflow (Liu et al. 2023) は、ノイズベクトルと現在のモデルがそこから生成するサンプルのペアを用いて速度モデルを再訓練する。
   1. 20,000 個のノイズベクトルを引き、本節の教師を 50 オイラーステップで積分し、得られたペア上で速度損失と教師と同じ設定を用いて新しい `TimeMLP` を訓練せよ。
   2. 新しいモデルの 1、2、4、10 オイラーステップにおけるモード重みと逸脱率、および 50 ステップの軌跡とその弦との平均距離を、教師の同じ量と並べて報告せよ。
   3. Reflow は教師の誤差のうちどれを取り除き、どれを引き継ぎ、どれを新たに付け加えるか。2 ラウンド目を行うとそれぞれどうなるか。
3. **ラグランジュ型損失。**●Boffi et al. (2025) はラグランジュ型恒等式を用いてフロー写像を訓練している。
   1. Equation 17.57 を $\partial_s \Phi_{t\to s}(\mathbf{x}) = \mathbf{u}_s(\Phi_{t\to s}(\mathbf{x}))$ に代入し、ジャンプの出力点における速度を含む $\bar{\mathbf{u}}$ の恒等式を導け。
   2. この恒等式ではなぜ条件付き速度 $\epsilon - \mathbf{x}_0$ を用いることができず、速度モデルが必要になるのか。また、フロー写像自身の速度 $\bar{\mathbf{u}}(\cdot, s, s)$ は利用できるため、単一のネットワークで自己訓練できることを示せ。
   3. [code] このような生徒を混合分布上で訓練し、逸脱率と 1 ステップあたりの訓練コストについて MeanFlow の生徒と比較せよ。
4. [code] **勾配停止と重み付け。** Equation 17.62 の最小点が何になるかは、2 つの細部によって決まる。
   1. `flow_map_error` をセルにコピーし、ターゲットに対する勾配停止を取り除け。この損失で混合分布上に MeanFlow の生徒を訓練し、1 ステップでのモード重み、逸脱率、広がりを報告せよ。ターゲット側も微分した場合の損失の勾配を書き下し、勾配停止が取り除いている項を特定せよ。ネットワークは Equation 17.61 を満たさずにこの損失を下げるため、何を変えることができるか。
   2. Song らは代わりに生徒の重みの移動平均を用いてターゲットを計算している。これを `d2l.EMA` で実装し、比較せよ。
   3. MeanFlow の ImageNet レシピ (Geng et al. 2025) は、各訓練例の二乗誤差 e を固定した $e + c$（c は小さい定数）で割ることで、勾配が $\log(e + c)$ の勾配と一致するようにしている。二乗誤差はターゲットの条件付き平均によって最小化されるが、重み付き損失は最小点をどの値の近くに置くか。教師がない場合、混合分布のノイジーな点におけるターゲットは 3 つのモードに広がる。1 回ジャンプのサンプルのモード重みを予測したうえで、この重み付けと $c = 10^{-3}$ を用いて MeanFlow の生徒を訓練し、結果を確認せよ。

4. 28×28 の画像では、e は 784 個の二乗誤差の和となる。このとき同じ (s, t) における訓練例間のそのばらつきは 2 次元の場合よりはるかに小さくなり、重み付けは主として時刻間の重み付けとして働くことを論ぜよ。混合分布での実験から画像向けレシピについて何がわかり、何がわからないか。

5. [code] **ガイダンス付き生徒**。●17.4.4.2 節の分類器なしガイダンスは、1 ステップごとにネットワークを 2 回評価する。Meng et al. (2023) はガイダンス付き速度を 1 つのネットワークへと蒸留し、ステップ数を減らす前の段階ですべてのステップのコストを半減させている。本節の画像教師と同様に 17.4 節のクラス条件付き U-Net を `time_freqs=slow` で訓練し、参照速度 $\mathbf{u}_{\text{ref}}$ をガイダンス付き速度 $\tilde{\mathbf{u}}$ すなわち `d2l.cfg(teacher, c, 2.0)`（c はランダムなラベル）として、ラベルも入力に受け取る 2 時刻の生徒へと蒸留せよ。
   1. 50 回のガイダンス付きオイラーステップを用いた教師と、1、2、4 回のジャンプを用いた生徒について、FD および予測クラスと指定クラスの一致率を比較せよ。
   2. 各場合において、サンプル 1 枚あたり何回ネットワークを評価するか。
   3. Meng らはガイダンス強度も生徒への入力として与えている。訓練中にガイダンス強度をどのようにサンプリングすべきか。また、$\gamma = 2$ に固定して蒸留した生徒にはないどのような利点がその生徒から得られるか。

6. ***ノイズ付き分布のマッチング***。●分布マッチング蒸留 (Yin, Gharbi, Zhang, et al. 2024) は、1 ステップ生成器 $\mathbf{x} = g_\theta(\mathbf{z})$ を勾配 Equation 17.63 に沿って訓練する。ここで $q_{\theta,t}$ はノイズを加えた生成サンプル $\mathbf{x}_t$ の分布である。
   1. Equation 17.63 を導出せよ。$q_{\theta,t}$ 自身の θ への依存から生じる項が消えるのはなぜか。
   2. 1 次元において、生成器を $m + bz$（ただし $b > 0$）、データを $\mathcal{N}(\mu, v)$ とする。線形経路上の $D_{\text{KL}}(q_t \,\|\, p_t)$ を閉じた式で表し、$m = \mu$、$b^2 = v$ が任意の $t < 1$ における唯一の最小点であることを示せ。
   3. データを $\frac{1}{2}\mathcal{N}(-2, 0.25) + \frac{1}{2}\mathcal{N}(2, 0.25)$ とする。$D_{\text{KL}}(q_t \,\|\, p_t)$ を (m, b) のグリッド上で数値積分により $t \in \{0.1, 0.3, 0.5, 0.9\}$ のそれぞれについて評価し、各 t における最小点を求めよ。最小値が $\log 2$ 付近にとどまる（小さい t において）のはなぜか。この結果を GAN におけるモード欠落（16.1.3.3 節）およびノイズ下でのモードの合流（演習問題 4、38.5 節）と関連付けよ。大きい t に向けてノイズレベルに重みを付けるとガウス生成器にとって何が得られ、どのような重み付けによっても得られないものは何か。

17.9 離散拡散

前節までのモデルはすべて、$\mathbb{R}^2$ 上の点、ピクセル強度、あるいはオートエンコーダの潜在表現といった連続値ベクトルにガウスノイズを加えることに依拠していた。しかし、テキストトークン、ソースコード、量子化された画像トークンのような離散系列には、微小な実数を加算する摂動を与えられない。また、自己回帰モデル（11.2 節）が左から右へ L 回の逐次的な順伝播によって L 個のトークンを生成するのに対し、離散拡散モデル（discrete diffusion model）は双方向文脈（bidirectional context）を用い、1 回のネットワーク評価あたり複数のトークンを並列に生成する。

有限語彙上の拡散を定義するため、Hoogeboom et al. (2021) と Austin et al. (2021) はガウス摂動を、遷移行列 $\mathbf{Q}_t$ に従う離散状態マルコフ連鎖に置き換えた。テキストや離散符号において最も有効な破損過程は**吸収（マスキング）カーネル**（absorbing kernel）であり、各トークンを独立にマスク記号 $[\text{M}]$ へ置き換えていき、最終的に系列全体を消去する。この吸収連鎖のもとでは、17.6 節の変分下界（variational lower bound）は重み付きマスク交差エントロピー目的関数へと簡約される (Sahoo et al. 2024; Shi et al. 2024)。本節では、吸収マルコフ連鎖とその変分上界を導出し、3 モード混合分布を量子化した 2 トークン版および 2 値化 Fashion-MNIST 画像において複数トークンを並列にマスク解除する際に生じる独立性誤差（independence error）を解析したうえで、マスク拡散をマスク言語モデルおよび自己回帰モデル（11.4 節、11.2 節）と結びつけ

る。

```
from scipy.special import ndtr, xlogy
```

17.9.1 トークンの破損

トークンの順過程（forward process）に求める条件は、17.2 節のガウス連鎖が備えていたものと同じである。すなわち、周辺分布と事後分布が閉形式で得られ、終端状態がデータに関する情報を一切持たないことである。

17.9.1.1 遷移行列

まず記法を変更する。離散拡散の文献に合わせ、$t = 1, \dots, T$ はステップ数を数える整数とする。これはガウス連鎖において k が担っていた役割である。ネットワークへの入力は引き続き、本章の時計における時刻 t/T とする。

トークンは K 通りの値のいずれかをとるものとし、ワンホット行ベクトル $\mathbf{x} \in \{0,1\}^K$ で表す。トークンに対するノイズとはその値をランダムに変化させることであり、各行の和が 1 となる**遷移行列**（transition matrix）$\mathbf{Q}_t \in [0,1]^{K \times K}$ で与えられる。ここで $[\mathbf{Q}_t]_{ij}$ は、値 i から値 j へステップ t で遷移する確率である。これはマルコフ連鎖モンテカルロ法（36.8.3 節）における行確率行列であり、分布は行ベクトルとして右から $\mathbf{Q}_t$ を掛けることで更新される。したがって、次式が成り立つ。

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathrm{Cat}(\mathbf{x}_t;\, \mathbf{x}_{t-1}\mathbf{Q}_t), \tag{17.64}$$

ここで、$\mathrm{Cat}(\mathbf{x}; \mathbf{p})$ は確率ベクトル $\mathbf{p}$ を持つカテゴリカル分布であり、この場合は現在の値によって選ばれる $\mathbf{Q}_t$ の行がその確率ベクトルとなる。長さ L のトークン系列は、すべての位置で独立にこの連鎖を走らせることで破損させる。2 つの遷移の合成はそれぞれの行列の積になるため、t に関する帰納法により、t ステップ後の周辺分布は次のようになる。

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathrm{Cat}(\mathbf{x}_t;\, \mathbf{x}_0\bar{\mathbf{Q}}_t), \qquad \bar{\mathbf{Q}}_t = \mathbf{Q}_1\mathbf{Q}_2 \cdots \mathbf{Q}_t. \tag{17.65}$$

両端が与えられたときの 1 ステップの事後分布は、ベイズの定理とマルコフ性（Markov property）から次のように導かれる。

$$\begin{aligned} q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) &= \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})\, q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \\ &= \mathrm{Cat}\left(\mathbf{x}_{t-1};\, \frac{\mathbf{x}_t\mathbf{Q}_t^\top \odot \mathbf{x}_0\bar{\mathbf{Q}}_{t-1}}{\mathbf{x}_0\bar{\mathbf{Q}}_t\mathbf{x}_t^\top}\right), \end{aligned} \tag{17.66}$$

ここで、$\odot$ は要素ごとの積である。とり得る各直前状態は、初期状態のもとでの確率と、観測された $\mathbf{x}_t$ へ至る確率の両方で重み付けされる。

各ステップで確率 0.1 で反転する 2 値トークンを考えると、この事後分布の働きがよくわかる。ここでは $\mathbf{Q}_1 = \mathbf{Q}_2 = \left(\begin{smallmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{smallmatrix}\right)$ である。1 番目の値 $\mathbf{x}_0 = (1, 0)$ から出発し、2 ステップ後に 2 番目の値 $\mathbf{x}_2 = (0, 1)$ が観測されたとしよう。これが起こる確率は $0.9 \cdot 0.1 + 0.1 \cdot 0.9 = 0.18$ であり、これが分母となる。中間の状態については、$\mathbf{x}_2\mathbf{Q}_2^\top = (0.1, 0.9)$ かつ $\mathbf{x}_0\bar{\mathbf{Q}}_1 = (0.9, 0.1)$ である。これらの要素ごとの積は $(0.09, 0.09)$ となり、0.18 で割ると $(1/2, 1/2)$ が得られる。ト

ークンは 1 度反転しており、両端が与えられたもとでは、その反転が 1 ステップ目で起きた確率と 2 ステップ目で起きた確率は等しい。

17.9.1.2 吸収カーネル

変分上界には $\bar{\mathbf{Q}}_t$ の閉形式が必要であり、サンプラには既知の初期状態が必要である。Austin et al. (2021) の**吸収カーネル**はその両方を与える。このカーネルは語彙に**マスク** $[\mathrm{M}]$ という 1 つの値を追加し、各トークンを確率 β_t でステップ t においてそこへ遷移させる。一度マスクされたトークンはマスクされたままである。

$$\mathbf{Q}_t = (1-\beta_t)\,\mathbf{I} + \beta_t\,\mathbf{1}\mathbf{e}_{\mathrm{M}}^{\top}, \tag{17.67}$$

ここで、$\mathbf{1}$ は全要素が 1 の列ベクトル、$\mathbf{e}_{\mathrm{M}}$ はマスクに対応するワンホット列ベクトルであり、$\mathbf{1}\mathbf{e}_{\mathrm{M}}^{\top}$ のすべての行は確率質量をすべてマスクに置く。

このような行列の積は簡潔にまとまる。$\mathbf{P} = \mathbf{1}\mathbf{e}_{\mathrm{M}}^{\top}$ と書こう。$\mathbf{e}_{\mathrm{M}}^{\top}\mathbf{1} = 1$ より $\mathbf{P}^2 = \mathbf{P}$ が成り立ち、したがって $(a\mathbf{I} + b\mathbf{P})(c\mathbf{I} + d\mathbf{P}) = ac\,\mathbf{I} + (ad + bc + bd)\,\mathbf{P}$ となる。$a + b = c + d = 1$ のとき第 2 項の係数は $1 - ac$ に等しいため、$\mathbf{I}$ の係数は連鎖に沿ってそのまま掛け合わされる。そこで、$\bar{\alpha}_t = \prod_{s \le t}(1 - \beta_s)$ とおく。これは 17.2 節のガウス連鎖と同じ積である。ガウス連鎖ではクリーンな点の重みの二乗であったが、ここではより直接的な意味、すなわちトークンが t ステップの間マスクされずに残る生存確率を表す。すると、次式が得られる。

$$\bar{\mathbf{Q}}_t = \bar{\alpha}_t\,\mathbf{I} + (1 - \bar{\alpha}_t)\,\mathbf{1}\mathbf{e}_{\mathrm{M}}^{\top}. \tag{17.68}$$

この連鎖はあるトークンを別のトークンと取り違えることは決してなく、ただ消去するだけである。$\bar{\alpha}_T = 0$ となるスケジュールは、ガウス連鎖が純粋なノイズに到達するのと同様に、データに関する情報を一切持たない全マスク系列に到達する（Figure 17.16）。もう 1 つのよく使われる選択肢である**一様カーネル**（uniform kernel）は、語彙から一様にトークンを再サンプリングする (Hoogeboom et al. 2021)（演習問題 1）。

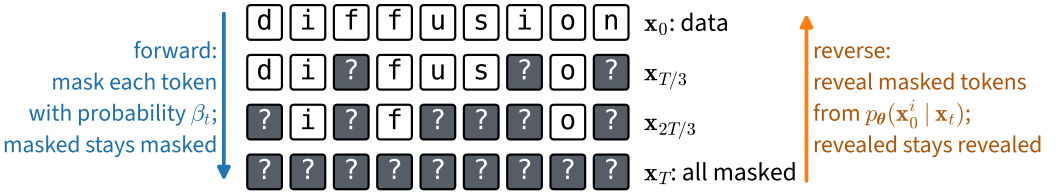


Figure 17.16: 9 個のトークンに対する吸収連鎖。順方向では、各トークンはステップごとに確率 β_t で独立にマスク（「?」として図示）へ置き換えられ、その後もマスクされたままとなり、T ステップ後にはすべてのトークンがマスクされる。逆方向では、モデルが部分的にマスクされた系列からマスクトークンを予測し、一度マスク解除されたトークンは決して変更されない。

以降は吸収カーネルを用いる。Austin らの比較実験においてテキスト上で最良の尤度を達成したうえ、事後分布がこれ以上ないほど単純になるからである。一方で失うものもある。逆方向では連鎖はマスクを解くことしかしないため、左から右へ生成するモデルと同様に、サンプラは一度書き込んだすべてのトークンをそのまま確定させることになる。一様カーネルのようにトークンを再サンプリングする連鎖や、ReMDM (G. Wang et al. 2025) のように一度マスク解除したトークンを再びマスクするサンプラであれば、早い段階での誤りを修正できる（17.10 節）。

17.9.2 マスク拡散の変分上界

逆連鎖を訓練するには損失が必要であり、17.6 節の変分上界がその損失を与える。

17.9.2.1 同じ上界、異なるターゲット

逆モデルは全マスク系列から出発するマルコフ連鎖 $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ である。17.6 節では、このような連鎖の負の対数尤度を次の式で上から抑えた。

$$\begin{aligned} -\log p_\theta(\mathbf{x}_0) \le L_{\text{VLB}} &= D_{\text{KL}}\big(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)\big) \\ &\quad + \sum_{t=2}^{T} \mathbb{E}_q\Big[D_{\text{KL}}\big(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\big)\Big] \\ &\quad + \mathbb{E}_q\big[-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)\big]. \end{aligned}$$

これは順連鎖を変分分布としたときの 36.7 節のエビデンス下界（evidence lower bound）に負号をつけたものである。その導出はマルコフ性とベイズの定理しか使っていないため、トークンに対してもそのまま成り立つ。

変わるのはターゲットである。ガウスノイズの場合、ネットワークはノイズ、速度、またはクリーンな点を予測しており、これらはいずれも 1 つの事後平均（posterior mean）$\mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t]$ のアフィン関数であった（17.1 節）。ワンホットトークンに対するこの事後平均は、クリーンなトークンがとり得る各値の確率ベクトルになるため、その分布を予測することが本章の出発点であったノイズ除去器（denoiser）の離散版にあたる。Austin et al. (2021) に従い、ネットワークは破損した系列全体が与えられたもとで、各位置 i についてクリーンな値の上の分布 $p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_t)$ を出力し、逆遷移（reverse transition）はこの予測に関して真の事後分布を平均化する。

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \sum_{\hat{\mathbf{x}}_0} q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0)\, p_\theta(\hat{\mathbf{x}}_0 \mid \mathbf{x}_t), \tag{17.69}$$

この計算は位置ごとに行い、マスクされていない位置での予測は観測されたトークンそのものとする。単一のトークンであれば、予測が完璧なときこれは真の逆遷移と一致する（演習問題 3a）。しかし系列に対しては 1 つの点で真の逆遷移に及ばない。真の逆遷移が同じステップで元に戻る複数の位置を同時分布からサンプリングするのに対し、この逆モデルはそれらを互いに独立にサンプリングするのである。本節の大部分は、この不足がもたらすコストを扱う。

17.9.2.2 吸収連鎖の事後分布

命題（吸収連鎖の事後分布） $\beta_1 > 0$（**したがって** $\bar{\alpha}_t < 1$ **がすべての** $t \ge 1$ **で成り立つ**）**である吸収カーネルのもとで、クリーンな値** $\mathbf{x}_0 \neq [\mathbf{M}]$ **を持つ 1 つの位置を考える。** $\mathbf{x}_t = \mathbf{x}_0$ **ならば、確率 1 で** $\mathbf{x}_{t-1} = \mathbf{x}_0$ **である。** $\mathbf{x}_t = [\mathbf{M}]$ **ならば、次式が成り立つ。**

$$\begin{aligned} q(\mathbf{x}_{t-1} = \mathbf{x}_0 \mid \mathbf{x}_t = [\mathrm{M}], \mathbf{x}_0) &= \frac{\bar{\alpha}_{t-1} - \bar{\alpha}_t}{1 - \bar{\alpha}_t}, \\ q(\mathbf{x}_{t-1} = [\mathrm{M}] \mid \mathbf{x}_t = [\mathrm{M}], \mathbf{x}_0) &= \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}, \end{aligned} \tag{17.70}$$

これ以外の値が正の確率を持つことはない。

証明 Equation 17.66 を Equation 17.67 および Equation 17.68 を用いて評価する。ベクトル $\mathbf{x}_0\bar{\mathbf{Q}}_{t-1}$ は確率質量 $\bar{\alpha}_{t-1}$ を $\mathbf{x}_0$ に置き、$1-\bar{\alpha}_{t-1}$ をマスクに置くため、とり得る直前状態はこの 2 つだけである。$\mathbf{x}_t = \mathbf{x}_0$ ならば、マスクがトークンに戻ることは決してないため、ベクトル $\mathbf{x}_t\mathbf{Q}_t^\top$ は成分 $1-\beta_t$ を $\mathbf{x}_0$ に持ち、成分 0 をマスクに持つので、事後分布は点質量となる。$\mathbf{x}_t = [\mathrm{M}]$ ならば、成分は β_t が $\mathbf{x}_0$ に、1 がマスクに対応するため、正規化前の事後分布は $\bar{\alpha}_{t-1}\beta_t$ を $\mathbf{x}_0$ に、$1-\bar{\alpha}_{t-1}$ をマスクに置く。それらの和 $1-\bar{\alpha}_{t-1}(1-\beta_t) = 1-\bar{\alpha}_t$ が分母 $q(\mathbf{x}_t = [\mathrm{M}] \mid \mathbf{x}_0)$ であり、$\bar{\alpha}_{t-1}\beta_t = \bar{\alpha}_{t-1} - \bar{\alpha}_t$ となる。■

1 ステップ遡ると、マスクトークンはマスクされたままであるか、クリーンな値に戻るかのいずれかである。したがって逆モデル Equation 17.69 は、$\mathbf{x}_t$ のマスクされていない位置をそのままコピーし、マスクされた各位置を互いに独立に、元に戻る確率で $p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_t)$ からのサンプルに置き換える。

17.9.2.3 重み付き交差エントロピー

逆モデルは見えているトークンをコピーしてマスクのみを埋めるため、変分上界においてコストを生じ得るのはマスクされた位置だけである。この形で Sahoo et al. (2024) と Shi et al. (2024) によって示された次の結果が、本節の中心となる命題である。すなわち、そのコストは交差エントロピーになる。

命題（マスク拡散の変分上界）$\bar{\alpha}_0 = 1$、$\bar{\alpha}_t < 1$ $(t \geq 1)$、$\bar{\alpha}_T = 0$ を満たす吸収カーネルと、上で述べた逆モデルに対して、次式が成り立つ。

$$L_{\mathrm{VLB}} = \sum_{t=1}^{T} \frac{\bar{\alpha}_{t-1} - \bar{\alpha}_t}{1-\bar{\alpha}_t}\, \mathbb{E}_{q(\mathbf{x}_t \mid \mathbf{x}_0)}\Big[\sum_{i\,:\,\mathbf{x}_t^i = [\mathrm{M}]} -\log p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_t) \Big]. \tag{17.71}$$

証明事前項（prior term）は 0 になる。$\bar{\alpha}_T = 0$ により $q(\mathbf{x}_T \mid \mathbf{x}_0)$ は全マスク系列上の点質量となり、これは $p(\mathbf{x}_T)$ と一致するからである。また、$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ と $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ はともに位置ごとに分解されるため、各ダイバージェンスは位置ごとのダイバージェンスの和となる。マスクされていない位置では両分布は同じ点質量となるため、ダイバージェンスは 0 である。マスクされた位置では $w_t = (\bar{\alpha}_{t-1} - \bar{\alpha}_t)/(1-\bar{\alpha}_t)$ とおく。真の事後分布は確率 $1-w_t$ をマスクに、w_t を $\mathbf{x}_0^i$ に割り当て、モデルは確率 $1-w_t$ をマスクに、$w_t\, p_\theta(\cdot \mid \mathbf{x}_t)$ を各トークンに割り当てる。したがって、次式が成り立つ。

$$\begin{aligned} D_{\mathrm{KL}} &= (1-w_t)\log\frac{1-w_t}{1-w_t} + w_t \log\frac{w_t}{w_t\, p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_t)} \\ &= -w_t \log p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_t). \end{aligned}$$

以上で $t = 2, \ldots, T$ の項が得られる。再構成項（reconstruction term）は $\mathbb{E}[-\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)]$ である。$w_1 = (1-\bar{\alpha}_1)/(1-\bar{\alpha}_1) = 1$ であるから、モデルはステップ 1 でマスクされているすべての位置を $p_\theta(\cdot \mid \mathbf{x}_1)$ からのサンプルで置き換え、残りをコピーする。ゆえにこの項はマスクされた位置にわたる $-\log p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_1)$ の和に等しく、総和の $t=1$ の項と一致する。■

これはマスク言語モデルの損失に 2 つの変更を加えたものである。第 1 に、マスク率が 1 つの値に固定されず、ゼロから全マスクまですべてのレベルにわたって変化する。第 2 に、各レベルが固有の重みを持つ。この上界を $\log 2$ とトークン数で割ると、トークンあたりビット数で表した符号長（code length）となる（37.2 節）。画像実験ではこの単位で値を読み取る。

線形スケジュール（linear schedule）$\bar{\alpha}_t = 1 - t/T$ のもとでは、トークンの割合 t/T がステップ t でマスクされ、重みは $w_t = (1/T)/(t/T) = 1/t$ となる。多くがマスクされた系列での予

測誤差は、わずかしかマスクされていない系列での同じ誤差よりも小さく評価される。マスクされる位置の個数は t に比例して増えるため、重み付きマスク位置数の期待値はどのステップでも同じであり、L/T（トークン数 L の場合）となる。ステップ間の違いは、マスクトークンの予測がどれだけ難しいかだけである。

この総和はリーマン和（Riemann sum）でもある。$\ell(\mathbf{x}_0, \mathbf{x})$ を $\mathbf{x}$ のマスク位置における交差エントロピー、すなわち Equation 17.71 の内側の和とし、$T \to \infty$ の極限を、$\bar{\alpha}_t = \bar{\alpha}(t/T)$（本章の時計における減少関数で、時刻 0 がデータ、時刻 1 が全マスク系列に対応する）のもとでとる。いま t はステップ数を表すため、時刻を $\tau = t/T$ と書く。すると $w_t \approx -\bar{\alpha}'(\tau)/(T(1-\bar{\alpha}(\tau)))$ となる。この因子 $1/T$ は 1 ステップの幅 $d\tau$ であるから、変分上界は次の積分へ収束する。

$$\int_0^1 \frac{-\bar{\alpha}'(\tau)}{1-\bar{\alpha}(\tau)}\, \mathbb{E}\big[\ell(\mathbf{x}_0, \mathbf{x}_\tau)\big]\, d\tau.$$

ここでマスク確率 $u = 1 - \bar{\alpha}(\tau)$ へ変数変換しよう（本節における u は速度ではなく確率である）。$du = -\bar{\alpha}'(\tau)\, d\tau$ より、積分は次の形になる。

$$\int_0^1 \frac{1}{u}\, \mathbb{E}_u\big[\ell(\mathbf{x}_0, \mathbf{x})\big]\, du,$$

ここで、$\mathbb{E}_u$ は各トークンを確率 u で独立にマスクする期待値である。時刻ではなくマスク確率を入力として受け取るネットワークにとって、この極限はスケジュールに依存しない。これはガウス拡散の上界が持つスケジュール不変性（17.6 節）の離散版である (Kingma et al. 2021)。

実は、最適なネットワークにはそのどちらも不要である。どの位置をマスクするかは、トークンの中身を一切見ないコイン投げによって決まる。入力によってどの位置がマスクされ、残りの位置にどのトークンがあるかが示された時点で、時刻は隠されたトークンについて何の追加情報ももたらさない。したがって、マスクされた位置における最良の予測、すなわち可視トークンが与えられたもとでのそのトークンの条件付き分布は、時刻に一切依存しない (Ou et al. 2025)（演習問題 4）。

近年の論文では、この極限は別の形で登場する。すなわち、残っている各トークンをレート $-\bar{\alpha}'(\tau)/\bar{\alpha}(\tau)$ でマスクする**連続時間マルコフ連鎖**（continuous-time Markov chain）(Campbell et al. 2022) を考え、1 トークンだけ異なる系列間の確率比を用いて時間反転する定式化である (Lou et al. 2024)。吸収カーネルでは、この確率比は既知の時間関数と本節のネットワークが予測するクリーンな条件付き分布との積になるため (Ou et al. 2025)、どちらの見方でも同じ対象を訓練することになる。

関数 `masked_loss` は、系列ごとに 1 つのステップ t を用い、スケジュール `alpha_bar`（値 $\bar{\alpha}_0, \dots, \bar{\alpha}_T$）とマスクのインデックス `mask_id` を受け取って変分上界を推定する。係数 T により、一様にサンプリングしたステップから全ステップの和の不偏推定量が得られる。総和は最初の軸以降のすべての軸にわたってとるため、トークンのペアでも画像でも同じ関数を共有できる。本章のインターフェースを統一するためネットワークには引き続き t/T を入力するが、訓練されたペアモデルがこの時刻入力をほとんど使っていないことは演習問題 4 で確認する。

```
def masked_loss(model, x0, t, alpha_bar, mask_id, key):
    """One-draw estimate of the masked bound per sequence, in nats."""
    T = len(alpha_bar) - 1
    shape = (-1,) + (1,) * (x0.ndim - 1)       # broadcast over positions
    masked = (jax.random.uniform(key, x0.shape)
```

```
            < (1 - alpha_bar[t]).reshape(shape))  # with prob. 1 - abar_t
    logits = model(jnp.where(masked, mask_id, x0), t / T)
    nll = -jnp.take_along_axis(jax.nn.log_softmax(logits), x0[..., None],
                               -1)[..., 0]
    w = (alpha_bar[t - 1] - alpha_bar[t]) / (1 - alpha_bar[t])  # 1/t if linear
    return (T * w.reshape(shape) * nll * masked).reshape(len(x0), -1).sum(1)
```

17.9.3 マスク解除によるサンプリング

訓練済みのネットワークは、マスクされたすべての位置で分布を与える。生成においては、どの位置をいつ埋めるかを決めなければならない。

17.9.3.1 並列マスク解除と独立性誤差

生成は全マスク系列から始まる。ステップ t では、1 回の順伝播によってマスクされたすべての位置で $p_\theta(\cdot \mid \mathbf{x}_t)$ が得られ、各マスク位置は候補トークンをサンプリングして確率 w_t で採択する。17.5 節のガウスサンプラと同様に、T 回ものネットワーク評価は避けたいため、レベル $T = t_0 > t_1 > \cdots > t_S = 0$ を選んでそれらの間をジャンプする。レベルを飛び越えることはカーネルの合成にあたり、上で示した簡約により吸収カーネルの積は再び吸収カーネルとなる。したがって粗視化した連鎖も各レベルで周辺分布 Equation 17.68 を持ち、先ほどの命題がそのまま適用できる。すなわち、関数 unmask は t_k から t_{k+1} へジャンプする際、各マスクトークンを確率 $(\bar\alpha_{t_{k+1}} - \bar\alpha_{t_k})/(1-\bar\alpha_{t_k})$ でマスク解除する。

マスク解除の判定はネットワークを一切参照しないため、各トークンがいつ現れるかは事前にわかる。マスクされたまま残る確率 $(1-\bar\alpha_{t_{k+1}})/(1-\bar\alpha_{t_k})$ はジャンプに沿って掛け合わされて逐次相殺されるため、レベル t_k でトークンがまだマスクされている確率は順連鎖と同じく $1-\bar\alpha_{t_k}$ である。したがって各トークンは、ジャンプ k ($k = 0, \dots, S-1$) において確率 $\Delta_k = \bar\alpha_{t_{k+1}} - \bar\alpha_{t_k}$ で、他の位置とは独立にマスク解除される。

```
@nnx.jit
def reveal(model, z, t, s, alpha_bar, mask_id, key):
    """One jump of the sampler, from level t down to level s."""
    T, (k1, k2) = len(alpha_bar) - 1, jax.random.split(key)
    draw = jax.random.categorical(k1, model(z, jnp.full(z.shape[:1], t / T)))
    p = (alpha_bar[s] - alpha_bar[t]) / (1 - alpha_bar[t])  # reveal prob.
    accept = (z == mask_id) & (jax.random.uniform(k2, z.shape) < p)
    return jnp.where(accept, draw, z)

def unmask(model, shape, steps, alpha_bar, mask_id, key):
    """Reveal tokens from the all-mask state in `steps` jumps."""
    T = len(alpha_bar) - 1
    levels = [round(l) for l in np.linspace(T, 0, steps + 1)]
    z = jnp.full(shape, mask_id)
    for t, s, k in zip(levels[:-1], levels[1:], jax.random.split(key, steps)):
        z = reveal(model, z, t, s, alpha_bar, mask_id, k)
    return z
```

ネットワークの予測は位置ごとの積になっている。そのため、同じジャンプでマスク解除されるトークンはそれぞれの予測分布から互いに独立にサンプリングされる一方、異なるジャンプでマスク解除されるトークンは、すでにマスク解除されたトークンで条件付けられながら順次サンプ

リングされる。この独立性にはどれだけのコストがかかるだろうか。コストが生じ得る最小の例である 2 トークンの場合から始めよう。

本章で一貫して用いている例からこの 2 トークンを作ることができる。それは平面上の 3 つのガウス分布の混合分布であり、平均は $(-2.5, -1.5)$、$(2.5, -1.5)$、$(0, 2.5)$、重みは 0.5、0.3、0.2、標準偏差は 0.5 であった（17.1 節）。平面を 32×32 のグリッド（範囲 $[-5, 5]^2$）へ量子化すると、各サンプルは 2 つの軸に沿ったセルのインデックスというトークンのペアになる。ペアの確率テーブル p には、各モードに対応する 3 つの塊（ブロブ）がある。周辺分布 p^1 と p^2 は、第 1 軸に沿っては 3 つの山を持つが、2 つのモードが同じ高さを共有しているため第 2 軸に沿っては 2 つの山しか持たない。周辺分布の積 $p^1 \otimes p^2$ は 2 つのトークンを独立にサンプリングするため、一方の山と他方の山が交差するすべての場所にブロブを置く。その結果 6 つのブロブが生じ、そのうち 3 つはデータが存在しない場所にある。

ここで、マスクされた各位置での予測が、マスクされていないトークンが与えられたもとでの真の条件付き分布と一致する**厳密な**ネットワークを考えよう。線形スケジュールのもとで S 回の等間隔ジャンプを行うと、すべてのジャンプで $\Delta_k = 1/S$ となるため、両方のトークンが同じジャンプでマスク解除される確率は $S \cdot (1/S)^2 = 1/S$ であり、この事象ではペアは $p^1 \otimes p^2$ からサンプリングされる。それ以外の場合は、一方のトークンが周辺分布から、もう一方が後から条件付き分布からサンプリングされるため、ペアは p から得られる。したがってサンプラは次の混合分布からサンプリングする。

$$\left(1 - \tfrac{1}{S}\right) p + \tfrac{1}{S}\, p^1 \otimes p^2.$$

2 つの確率テーブルの差の絶対値和の半分である全変動（total variation）で測ると、この混合分布は $\mathrm{TV}(p^1 \otimes p^2, p)/S$ だけ p から離れている。誤差はステップ数の逆数に比例して減少し、トークン間の依存性が強いほど大きくなる。

1 ステップでの誤差は計算する前から予測できる。積分布の 3 つの偽のブロブは確率質量のうち $0.5 \cdot 0.2 + 0.3 \cdot 0.2 + 0.2 \cdot 0.8 = 0.32$ を受け取り、もしブロブ同士に重なりがなければこれがそのまま全変動になる。各セルの確率は 1 次元のガウス積分に分解できるため、以下のコードセルでは p を厳密に計算し、そのエントロピー、2 トークン間の相互情報量、および 4 通りのステップ数に対する厳密なネットワークのサンプラ誤差を求める。

```
mix = d2l.GaussianMixture.three_modes()
means, weights = np.float64(mix.means), np.float64(mix.weights)
K, T = 32, 32                 # vocabulary of each token; number of steps
edges = np.linspace(-5, 5, K + 1)
cdf = ndtr((edges - means[:, :, None]) / mix.std)  # (modes, axes, K + 1)
cdf[:, :, 0], cdf[:, :, -1] = 0, 1      # edge cells extend to infinity
cell = np.diff(cdf, axis=2)             # mass of each cell, per mode, axis
table = np.einsum('m,mi,mj->ij', weights, cell[:, 0], cell[:, 1])
p1, p2 = table.sum(1), table.sum(0)     # marginals of the two tokens
product = np.outer(p1, p2)
H = lambda p: -xlogy(p, p).sum()        # entropy in nats
tv = lambda p, q: np.abs(p - q).sum() / 2
mutual = H(p1) + H(p2) - H(table)
print(f'entropy {H(table):.3f} nats, mutual information {mutual:.3f} nats')
for S in (1, 2, 8, 32):
    print(f'{S:>2} steps, exact network: TV {tv(product, table) / S:.3f}')
```

```
entropy 4.840 nats, mutual information 0.479 nats
 1 steps, exact network: TV 0.314
```

```
 2 steps, exact network: TV 0.157
 8 steps, exact network: TV 0.039
32 steps, exact network: TV 0.010
```

1 ステップのサンプラは確率質量の 0.314 を誤った場所に配置する。隣接するブロブがわずかに重なっているため予測値の 0.32 より少し小さい値となっており、32 ステップでは誤差は 0.010 まで下がる。

17.9.3.2 確信度順のマスク解除

実用上の多くのサンプラはランダムにマスク解除するわけではない。MaskGIT (Chang et al. 2022) や拡散言語モデル LLaDA (Nie, Zhu, You, et al. 2025) は、マスクされたすべての位置で候補トークンをサンプリングし、それぞれの予測分布のもとで最も確率の高い候補だけを残して、残りを再びマスクする。この**確信度再マスキング**（confidence remasking）は、サンプリングした値に基づいてどのトークンをマスク解除するかを選ぶ。これはランダムな順序より優れているのだろうか。2 トークンの確率テーブルを使えば、一切の訓練なしに答えが出る。2 トークンの場合、確信度再マスキングは両方を周辺分布 $A \sim p^1$ と $B \sim p^2$ からサンプリングし、自身の周辺分布のもとでより確率の高い方を残して、もう一方を残したトークンが与えられた条件付き分布から引き直す。その出力テーブルは次のようになる。

$$\begin{aligned} q(a,b) = {} & p^1(a)\, P\big(p^2(B) \le p^1(a)\big)\, p(b \mid a) \\ & + p^2(b)\, P\big(p^1(A) < p^2(b)\big)\, p(a \mid b), \end{aligned}$$

ここで第 1 項は第 1 トークンのサンプル a を残す場合、第 2 項は第 2 トークンのサンプル b を残す場合である。次のコードセルでは、このテーブルを 1 ステップ、2 回のランダムジャンプ、および固定順序（第 1 トークンを周辺分布から、第 2 トークンを条件付き分布からサンプリング）のテーブルと比較する。

```
c2 = table / p1[:, None]                # p(x2 | x1); each row sums to one
c1 = table / p2[None, :]                # p(x1 | x2); each column sums to one
keep1 = p1[:, None] >= p2[None, :]      # the draw of x1 is the likelier one
confidence = ((product * keep1).sum(1)[:, None] * c2
              + (product * ~keep1).sum(0)[None, :] * c1)
centers = (edges[:-1] + edges[1:]) / 2
grid = np.stack(np.meshgrid(centers, centers, indexing='ij'), -1)
mode = ((grid[:, :, None] - means) ** 2).sum(-1).argmin(-1)  # nearest mode
orders = {'one step': product, 'two random jumps': (product + table) / 2,
          'fixed order': p1[:, None] * c2, 'confidence': confidence}
print(f'{"order":>16} {"TV":>6} {"entropy":>8}  mass near the modes')
for name, q in orders.items():
    masses = ' '.join(f'{q[mode == k].sum():.3f}' for k in range(3))
    print(f'{name:>16} {tv(q, table):6.3f} {H(q):8.3f}  {masses}')
```

```
           order     TV  entropy  mass near the modes
        one step  0.314    5.319  0.482 0.321 0.197
two random jumps  0.157    5.216  0.491 0.310 0.199
     fixed order  0.000    4.840  0.500 0.300 0.200
      confidence  0.179    4.406  0.606 0.322 0.072
```

条件付き分布が厳密であれば、左から右であれランダムであれ、トークンをサンプリングする前に固定された順序で 1 ステップに 1 トークンずつマスク解除するサンプラは、固定順序の結果が示すとおり厳密になる。2 回のランダムジャンプでは半分の確率で両トークンが同時にマスク解

除され、1 ステップ誤差は半分になる。驚くべきことに、同じく 1 ステップに 1 トークンずつマスク解除しながらサンプリングした値から順序を選ぶ確信度再マスキングは、2 回のランダムジャンプよりも p から遠ざかり、0.179 対 0.157 となる。上の式からその理由がわかる。小さなモードからのサンプルは周辺確率が低いため、比較に負けて引き直される側になりやすく、勝った側で条件付けた分布から引き直しても小さなモードへ戻ることはめったにない。つまりこのサンプラはモード探索的であり、最小モードの質量は 0.2 から約 0.07 へ落ち込み、エントロピーも 4.84 から 4.41 nat へ低下する。テキストではエントロピーが低い方が流暢に読めることが多く、また不完全なネットワークでは簡単な位置から先にマスク解除することが有効な場合もある。しかし変分上界はこのサンプラを記述しておらず、そのバイアスを検出できない。変分上界はランダムマスクのもとで予測を評価するものであり、言い換えればランダム順のサンプラを記述しているからである。

17.9.4 訓練されたノイズ除去器

ここまでの数値はすべて厳密な確率テーブルから求めたものであった。ここからはネットワークを実際に訓練する。まずはあらゆる誤差の原因を切り分けられるトークンペアを扱い、続いて画像を扱う。

17.9.4.1 量子化された混合分布

ペア用のネットワークは、インデックス K にある 33 番目の値としてのマスクを含む両トークンのワンホットエンコーディングと時刻 t/T を受け取り、各位置について 32 通りのクリーンな値に対するロジットを出力する。スケジュールは $T = 32$ ステップの線形スケジュールとする。

```
class MaskedMLP(nnx.Module):
    """Logits of both clean tokens of a pair, given the masked pair and t."""
    def __init__(self, K, hidden=256, *, rngs):
        self.K = K
        self.net = d2l.make_mlp([2 * (K + 1) + 1, hidden, hidden, 2 * K],
                                nnx.silu, rngs=rngs)

    def __call__(self, z, t):  # z: (n, 2) in {0, ..., K}, where K is the mask
        h = jax.nn.one_hot(z, self.K + 1).reshape(len(z), -1)
        return self.net(jnp.concatenate([h, t[:, None]], 1)).reshape(
            -1, 2, self.K)

def quantize(x):  # points in the plane to pairs of tokens in {0, ..., K - 1}
    return jnp.clip(jnp.floor((x + 5) / 10 * K), 0, K - 1).astype(jnp.int32)

alpha_bar = 1 - jnp.arange(T + 1) / T       # the linear schedule
```

このネットワークを `d2l.train_regression`（17.1 節）により、混合分布から毎回新たにサンプリングした 1 バッチあたり 512 ペアと一様にサンプリングしたステップを用い、6,000 回の更新にわたってコサイン減衰する学習率で訓練する。

```
def batch_fn(key):
    k1, k2 = jax.random.split(key)
    return (quantize(mix.sample(k1, 512)),
            jax.random.randint(k2, (512,), 1, T + 1))
```

```
loss_fn = lambda model, batch, key: masked_loss(model, *batch, alpha_bar,
                                                mask_id=K, key=key).mean()
_, pair_model, _ = d2l.train_regression(
    MaskedMLP(K, rngs=nnx.Rngs(0)), loss_fn, batch_fn, 6000,
    jax.random.key(0), cosine=True)
```

位置が 2 つだけなので、変分上界の期待値は厳密に計算できる。各ステップにおいて、少なくとも 1 つのマスクを含むマスクパターンは 3 通りある。m 個のマスクを持つパターンの確率は $u^m(1-u)^{2-m}$（ただし $u = 1 - \bar{\alpha}_t$）であり、1,024 通りのペアを確率テーブルで重み付けする。この平均はテーブルのエントロピー、すなわちいかなるモデルも下回ることのできない平均負の対数尤度の最小値を下回らない（37.1 節）。ネットワークはどこまでこの値に近づけるだろうか。

テーブルからサンプリングされたペアの 2 トークンを X^1、X^2 と書き、厳密なネットワークについてこれを計算してみよう。変分上界は、逆連鎖が各トークンをマスク解除するステップにおいてそのトークンを評価する。実際、重み w_t と、確率 $1 - \bar{\alpha}_t$（ステップ t でトークンがマスクされている確率）の積は $\bar{\alpha}_{t-1} - \bar{\alpha}_t$ であり、これは逆連鎖がステップ t でそのトークンをマスク解除する確率に等しい。2 つのトークンが異なるステップでマスク解除されるとき、最初のトークンは周辺分布から、2 番目のトークンは最初のトークンが与えられた条件付き分布から予測されるため、合計のコストは $H(X^1) + H(X^2 \mid X^1) = H(X^1, X^2)$ となり、余分なコストは生じない。一方、同じステップでマスク解除される 2 つのトークンはそれぞれの周辺分布から独立に予測されるため、コストは $H(X^1) + H(X^2)$ となり、$H(X^1, X^2)$ より相互情報量 $I(X^1; X^2)$（37.6 節）だけ余分にかかる。線形スケジュールでは各トークンがあるステップでマスク解除される確率は $1/T$ なので、両方が同じステップでマスク解除される確率は $T \cdot (1/T)^2 = 1/T$ となり、厳密なネットワークは次の値を達成する。

$$H(X^1, X^2) + \frac{I(X^1; X^2)}{T}$$

（演習問題 5 では、この議論を任意のスケジュールに対する証明へと一般化する。）各位置を独立に予測するいかなるネットワークも、この下限 $4.840 + 0.479/32 \approx 4.855$ nat を下回ることはできない。$T \to \infty$ では 2 つのトークンが同じステップでマスク解除されることはなくなり、下限はエントロピーまで下がる。

```
def exact_bound(model, table, alpha_bar, mask_id):
    """The bound averaged over the table, summed exactly over t and masks."""
    T, K = len(alpha_bar) - 1, len(table)
    pairs = jnp.stack(jnp.meshgrid(jnp.arange(K), jnp.arange(K),
                                   indexing='ij'), -1).reshape(-1, 2)
    p, total = jnp.asarray(table.flatten(), jnp.float32), 0.0
    for t in range(1, T + 1):
        u = 1 - alpha_bar[t]
        w = (alpha_bar[t - 1] - alpha_bar[t]) / u
        for masked in jnp.array([[1, 0], [0, 1], [1, 1]], dtype=bool):
            z, m = jnp.where(masked, mask_id, pairs), masked.sum()
            logp = jax.nn.log_softmax(model(z, jnp.full(K * K, t / T)))
            nll = -(jnp.take_along_axis(logp, pairs[..., None], -1)[..., 0]
                    * masked).sum(1)
            total += u ** m * (1 - u) ** (2 - m) * w * (p @ nll)
    return float(total)
```

```
bound = exact_bound(pair_model, table, alpha_bar, mask_id=K)
print(f'exact bound {bound:.3f} nats; '
      f'entropy {H(table):.3f}, floor {H(table) + mutual / T:.3f}')
```

```
exact bound 4.856 nats; entropy 4.840, floor 4.855
```

上界はエントロピーを 0.02 nat 未満しか上回っておらず、モデルは各トークンの周辺分布と他方が与えられたときの条件付き分布を学習できている。差のほとんどすべては下限によるものであり、下限に対するネットワーク自身の超過分は 0.001 nat のオーダーにすぎない。この下限は単なる上界の緩さではない。厳密なネットワークの 32 ステップサンプラがモデル化するのは $(1-1/32)\,p + (1/32)\,p^1 \otimes p^2$ であって、p ではないからである。何が失われるかについて、変分上界とサンプラは一致している。

訓練されたサンプラも同じだけ失うのだろうか。2 トークンであれば、出力テーブルはサンプリングではなく総和によって厳密に計算できる。各トークンはジャンプ k において確率 Δ_k（コード中の `pi`）で、もう一方とは独立にマスク解除される。同時にマスク解除されるトークンはそのレベルにおけるネットワークの 2 つの周辺分布からサンプリングされ、後からマスク解除されるトークンは先に現れたトークンが与えられた条件付き分布からサンプリングされる。`conditionals` はペアモデルからこれらの予測を読み取り、`sampler_table` は全ジャンプにわたって和をとる。これに真の周辺分布と条件付き分布を渡せば、先に表示した厳密なネットワークの誤差が再現されるはずである。

```
def conditionals(model, tau):
    """A pair model's predictions at time tau: both marginals with both
    tokens masked, then c2[a, b] = q(x2 = b | x1 = a), c1[a, b] = q(x1 = a |
    x2 = b)."""
    K = model.K                                  # also the mask index
    a, m = jnp.arange(K), jnp.full(K, K)
    z = jnp.concatenate([jnp.array([[K, K]]),    # both masked
                         jnp.stack([a, m], 1),   # x1 = a revealed
                         jnp.stack([m, a], 1)])  # x2 = a revealed
    q = jax.nn.softmax(model(z, jnp.full(2 * K + 1, tau)))
    q = np.float64(q)                            # (2K + 1, 2, K)
    return q[0, 0], q[0, 1], q[1:K + 1, 1], q[K + 1:, 0].T
```

関数 `cond`（`conditionals` と同じシグネチャを持つ関数）を与えると、`sampler_table` はすべてのジャンプの組み合わせにわたって和をとり、S ステップのマスク解除サンプラの厳密な出力分布を構成する。これを真の同時分布 `table` と比較し、`pair_model` と `exact` の両方について $S \in \{32, 8, 2, 1\}$ にわたり評価する。

```
def sampler_table(cond, steps, alpha_bar):
    """Exact output table of `unmask` on a pair; cond(t / T) returns the
    predictions at level t in the order of `conditionals`."""
    T = len(alpha_bar) - 1
    levels = [round(l) for l in np.linspace(T, 0, steps + 1)]
    pi = np.float64([float(alpha_bar[s] - alpha_bar[t])  # P(revealed in jump)
                     for t, s in zip(levels[:-1], levels[1:])])[:, None]
    q1, q2, c2, c1 = map(np.stack, zip(*[cond(t / T) for t in levels[:-1]]))
    before1 = np.cumsum(pi * q1, 0) - pi * q1    # revealed in earlier jumps
    before2 = np.cumsum(pi * q2, 0) - pi * q2
    return (np.einsum('j,ja,jb->ab', pi[:, 0] ** 2, q1, q2)  # same jump
```

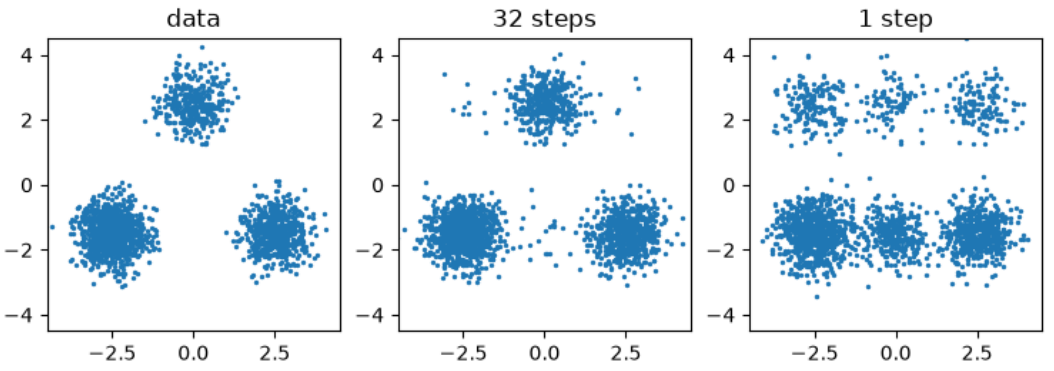
```
            + np.einsum('ja,jab->ab', pi * before1, c2)       # x1 first
            + np.einsum('jb,jab->ab', pi * before2, c1))      # x2 first

exact = lambda tau: (p1, p2, c2, c1)  # the true marginals and conditionals
print('TV to the table: trained and exact network; TV between the two')
for S in (32, 8, 2, 1):
    q = sampler_table(lambda tau: conditionals(pair_model, tau), S, alpha_bar)
    q_exact = sampler_table(exact, S, alpha_bar)
    print(f'{S:>2} steps: trained {tv(q, table):.3f}, exact '
          f'{tv(q_exact, table):.3f}, between {tv(q, q_exact):.3f}')
```

```
TV to the table: trained and exact network; TV between the two
32 steps: trained 0.020, exact 0.010, between 0.013
 8 steps: trained 0.045, exact 0.039, between 0.013
 2 steps: trained 0.159, exact 0.157, between 0.012
 1 steps: trained 0.316, exact 0.314, between 0.012
```

厳密なネットワークの列は $0.314/S$ を再現し、訓練されたサンプラもすべてのステップ数でそれに追随している。訓練されたテーブルと厳密なテーブルの距離であるネットワーク自身の誤差は、約 0.01 でどの S でもほぼ変わらない。この誤差は 32 ステップでは距離を 0.020 対 0.010 と 2 倍にするが、1 ステップや 2 ステップではほとんど問題にならない。ステップ数を増やしてもこの誤差は消えず、より良いネットワークによってしか減らせない。

以下のパネルは、2,000 個のデータペアと、各サンプラから 2,000 個ずつ unmask でサンプリングしたペアを、それぞれのセル内のランダムな位置に配置して示したものである。1 ステップサンプラは、あるモードの列と別のモードの行が交差する 3 つの偽のブロブを埋めてしまう。

```
k0, k32, k1, *kc = jax.random.split(jax.random.key(1), 6)
pairs = [quantize(mix.sample(k0, 2000))] + [
    unmask(pair_model, (2000, 2), S, alpha_bar, mask_id=K, key=k)
    for S, k in ((32, k32), (1, k1))]
decode = lambda z, k: (z + jax.random.uniform(k, z.shape)) / K * 10 - 5
d2l.scatter_panels([decode(z, k) for z, k in zip(pairs, kc)],
                   ['data', '32 steps', '1 step']);
```



17.9.4.2 2 値トークンとしての画像

2 トークンで仕組みがわかったので、画像でスケールさせた場合を見よう。画像に対するマスク拡散は、学習されたトークナイザの離散符号上で動作する (Chang et al. 2022)。ここではトークナイザを必要としない 2 値ピクセルをそのような符号の代わりとして用いる。Fashion-MNIST を半分の輝度でしきい値処理すると、すべてのピクセルが 2 値をとるトークンとなり、マスクされたピクセルが 3 番目の値を持つ。これにより画像はグリッド上の 784 個のトークン系列となるため、ネットワークには本章の U-Net（17.4 節）をそのまま使える。ワンホットエンコーディングを 3 つの入力チャネルとして受け取り、クリーンなピクセルのロジットを 2 つの出力チャネルとして返すのである。損失とサンプラはペアの場合と同じものを用い、$T = 100$ ステップの線形スケジュールとし、マスクのインデックスを 2 とする。

```
X, _, X_test, _ = d2l.load_fashion_mnist_pm1(binary=True)  # (n, 28, 28)
T_img = 100                                  # steps of the image chain
alpha_bar_img = 1 - jnp.arange(T_img + 1) / T_img

class MaskedUNet(nnx.Module):
    """d2l.UNet on one-hot pixels (black, white, mask); clean-pixel logits."""
    def __init__(self, *, rngs):
        self.unet = d2l.UNet(in_channels=3, out_channels=2, rngs=rngs)

    def __call__(self, z, t):  # z: (n, 28, 28) in {0, 1, 2}
        return self.unet(jax.nn.one_hot(z, 3), t)  # (n, 28, 28, 2)
```

訓練は 6,000 回の更新を 128 枚の画像バッチ上で行い、定数の学習率と、半減期 1,000 回更新の重みの移動平均を用いる。

```
def batch_fn(key):
    k1, k2 = jax.random.split(key)
    return (X[jax.random.randint(k1, (128,), 0, len(X))],
            jax.random.randint(k2, (128,), 1, T_img + 1))

loss_fn = lambda model, batch, key: masked_loss(  # per pixel
    model, *batch, alpha_bar_img, mask_id=2, key=key).mean() / (28 * 28)
_, unet, _ = d2l.train_regression(MaskedUNet(rngs=nnx.Rngs(2)), loss_fn,
                                  batch_fn, 6000, jax.random.key(2))
```

1,000 枚のホールドアウト画像に対し、各画像・各レベルにつき 1 つのランダムマスクを用いて変分上界を推定し、訓練セットにおける各位置の白ピクセルの出現頻度を用いてピクセルを独立とみなすモデルの交差エントロピーと比較する。

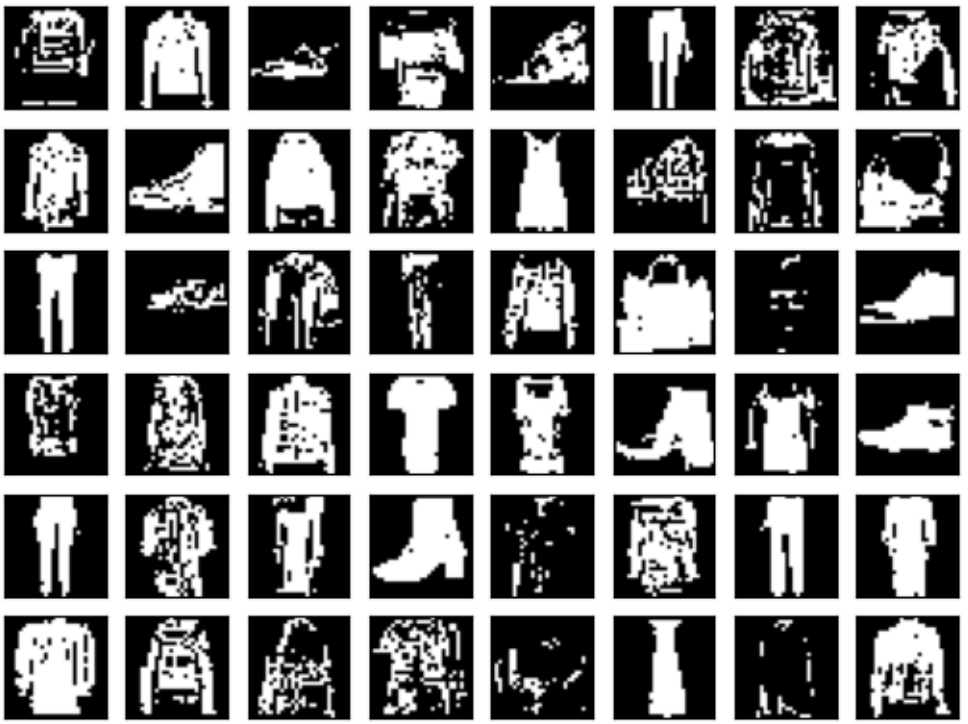
```
X_eval, keys = X_test[:1000], jax.random.split(jax.random.key(3), T_img)
term = nnx.jit(lambda model, x0, t, key: masked_loss(
    model, x0, jnp.full(len(x0), t), alpha_bar_img, mask_id=2,
    key=key).mean())
bound = sum(term(unet, X_eval, t, k)
            for t, k in zip(range(1, T_img + 1), keys)) / T_img
white = jnp.clip(X.mean(0), 1e-6, 1 - 1e-6)  # frequency of white
indep = -jnp.where(X_eval == 1, jnp.log(white), jnp.log(1 - white)).sum((1, 2))
bits = lambda nats: f'{nats / math.log(2) / (28 * 28):.3f} bits per pixel'
print(f'bound {bits(bound)}, independent pixels {bits(indep.mean())}')
```

```
bound 0.170 bits per pixel, independent pixels 0.702 bits per pixel
```

変分上界はピクセルあたり約 0.17 ビットであるのに対し、独立ピクセルモデルでは約 0.70 ビットである。符号長として読めば、ホールドアウトされた衣類画像 1 枚あたり約 135 ビットであり、独立ピクセルの約 550 ビットから大きく短縮されている。もっとも、隣接ピクセルを見るモデルならほぼどれでも大差で勝てるため、独立ピクセルが弱い基準であることは確かである。この実行が示しているのはその差の大きさである。ピクセル間の依存性をモデル化すれば独立ピクセルの符号長の少なくとも 4 分の 3 を節約でき、ネットワークはその節約を実現するのに十分な依存性を学習している。本来対等な比較相手となるのは、ラスタ順に 1 ピクセルずつマスク解除する同規模のモデルであろう。ここではそれを訓練しないが、テキスト上でのマスク拡散と左から右へのモデルの比較については 17.9.5.1 節で紹介する。

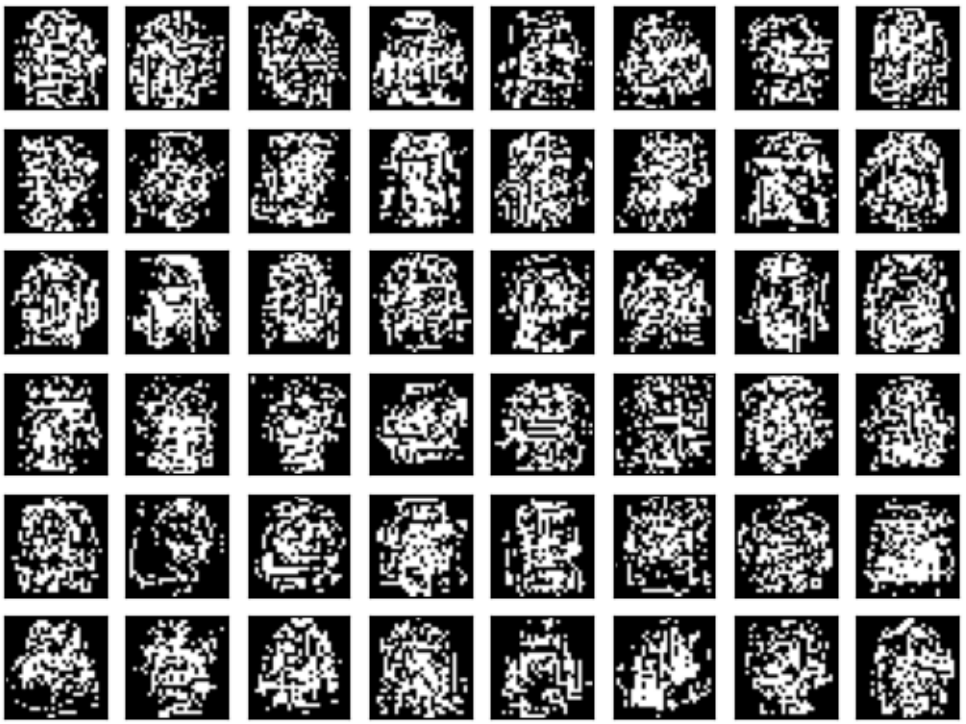
サンプラは 1,000 枚の画像を 100 ステップで生成し、グリッドには最初の 48 枚を示す。

```
z100 = unmask(unet, (1000, 28, 28), T_img, alpha_bar_img, mask_id=2,
              key=jax.random.key(4))
d2l.show_gray(z100[:48], 6, 8, scale=0.8);
```



サンプルの大半は識別可能な 2 値の衣類になっているが、前節までのグレースケール画像のサンプルに比べると粗く、5 枚に 1 枚ほどは断片的であるかほぼ空白である。6,000 回の更新では U-Net は訓練不足のままであるが、ここでは問題ない。この実験の目的は変分上界とステップ数の関係を調べることであり、最良の衣類画像を生成することではないからである。100 回の各ジャンプでは、それ以前のジャンプでマスク解除されたすべてのピクセルを参照する予測から約 8 ピクセルずつがマスク解除される。次のグリッドは、1 回あたり約 200 ピクセルをマスク解除する 4 回のジャンプを用いた結果である。

```
z4 = unmask(unet, (1000, 28, 28), 4, alpha_bar_img, mask_id=2,
            key=jax.random.key(5))
d2l.show_gray(z4[:48], 6, 8, scale=0.8);
```



衣類の形はまったく残らない。最初のジャンプが全マスク画像に対する予測（衣類がどのあたりに置かれやすいかしか知らない分布）から各ピクセルを独立にサンプリングし、後続のすべてのジャンプがこのノイズに条件付けられるためである。残るのは特定の衣類というより平均的な衣類の輪郭であり、その内部がテクスチャで埋め尽くされている。

4 ステップで何が失われるかを測るため、上下左右 4 つの隣接ピクセルすべてと値が異なるピクセルである**スペックル**を各種 1,000 枚の画像で数え、その割合と 1,000 枚にわたる標準誤差を表示する。さらに 100 ステップでもまだ何が失われているかを見るため、3 つ目のバッチでは平均して 1 ジャンプあたり 1 ピクセルとなる 784 回のジャンプを行う。元の連鎖の 100 レベルではこれだけのジャンプ数を担えない。`unmask` はレベルを連鎖の整数レベルへ丸めるため、同じレベル間のジャンプでは何もマスク解除されないからである。そこでこのジャンプには独自の 784 レベルの線形スケジュールを用いる。その時刻 $k/784$ の大半はネットワークが訓練時に見た 100 個の時刻の間に入り（最後の 7 個は最小時刻より下にある）、いずれにせよマスクされているピクセルの個数自体がネットワークに実質的な時刻を教えている。

```
def speckles(z):
    """Per image, the fraction of interior pixels unlike all four neighbors."""
    c = z[:, 1:-1, 1:-1]
    unlike = ((c != z[:, :-2, 1:-1]) & (c != z[:, 2:, 1:-1])
              & (c != z[:, 1:-1, :-2]) & (c != z[:, 1:-1, 2:]))
    return unlike.mean((1, 2))
```

```
alpha_bar_784 = 1 - jnp.arange(785) / 784          # 784 levels
z784 = unmask(unet, (1000, 28, 28), 784, alpha_bar_784, mask_id=2,
              key=jax.random.key(6))
for name, z in (('data', X[:1000]), ('784 steps', z784), ('100 steps', z100),
                ('4 steps', z4)):
    per_image = speckles(z)
    print(f'{name:>9}: speckles {per_image.mean():.4f}, standard error over '
          f'the 1,000 images {per_image.std(ddof=1) / len(z) ** 0.5:.5f}')
```

```
     data: speckles 0.0037, standard error over the 1,000 images 0.00016
784 steps: speckles 0.0047, standard error over the 1,000 images 0.00017
100 steps: speckles 0.0069, standard error over the 1,000 images 0.00020
  4 steps: speckles 0.0337, standard error over the 1,000 images 0.00027
```

データには内部ピクセル 1,000 個あたり約 4 個のスペックルがあり、4 ステップのサンプルにはその約 9 倍ある。これは 1 ステップのペアにおける偽のブロブの画像版である。100 ステップのサンプルはデータの約 2 倍であり、そこには 2 つの原因が混在している。1 ジャンプあたりまだ約 8 ピクセルが同時に現れることと、6,000 回の更新後ではネットワークが不完全なことである。784 ステップの行がこの 2 つを切り分ける。ペアの場合と同様、ジャンプ数を増やすと独立性誤差は縮まりネットワーク自身の誤差はそのまま残るが、1 ジャンプあたり 1 ピクセルにすることで 100 ステップの過剰分の 3 分の 2 から 4 分の 3 が解消される。この減少は画像のサンプルの偶然だけで生じ得るだろうか。両方の行は同じネットワークから生成されているため、比較における偶然の要素はそれぞれの 1,000 枚の画像だけである。各値は独立な画像にわたる平均であり、標準誤差は 0.00016〜0.0002、2 つの平均の差の標準誤差は約 0.00026 であるのに対し、実際の減少量は約 10 倍に達する。したがって、100 ステップで失われていたものの大半はネットワークではなく独立性誤差である。残りの部分（784 ステップのサンプルになおデータの約 1.3 倍、差の標準誤差にして約 4 倍分のスペックルを残している要因）は、ネットワーク自身の誤差と、ピクセルが隣接ピクセルとまれに同じジャンプを共有するときにわずかに残る独立性誤差の和である。演習問題 2e ではこの 2 つをさらに分離する。

17.9.5 隣接するモデルと振り返り

本節ではここまで、マスク拡散をそれ自身の枠組みで扱ってきた。本書においてこれに隣接するモデルは 11.2 節や 11.4 節の言語モデルであり、系譜上の祖先は本章の前節までのモデルである。

17.9.5.1 拡散と自己回帰

11.2 節の言語モデルを訓練した読者なら、そのモデルと本節のモデルがどう関係するのか気になるだろう。1 ステップあたりにマスク解除するトークン数によってサンプラを整理すれば、2 トークンの確率テーブルがその答えを与えてくれる。

左から右へ固定された順序で 1 ステップに 1 トークンずつマスク解除すると、サンプラは 11.2 節の言語モデルと同じ形になる。その条件付き分布は、マスクされていないプレフィックスとマスクされた残りからなる入力に対するネットワークの予測である。ペアの実験では、これは 4 つの順序を比較したテーブルの固定順序の行にあたり、距離 0.000 で p と一致する。条件付き分布が厳密であれば、1 トークンずつマスク解除する連鎖は厳密であり、その損失はエントロピーに等しい。

一様ランダムな順序で 1 ステップに 1 トークンずつマスク解除すると、サンプラは**順序非依存な自己回帰モデル**（order-agnostic autoregressive model）(Uria et al. 2014) となり、その損失はあらゆる順序にわたる負の対数尤度の平均になる。Hoogeboom et al. (2022) は、時刻を無視す

るネットワークにおいてこの損失が 17.9.2.3 節の連続時間上界と一致することを示し（演習問題 6 ではベータ積分を用いてこれを導出する）、Ou et al. (2025) は、マスクトークンに対する最良の予測がすでにマスク解除されたトークンのみに依存するため、時刻を無視しても何も失われないことを示した。したがってマスク拡散モデルは、すべての順序にわたる言語モデルとして同時に訓練されていることになる。厳密なネットワークではどの順序も同じ同時尤度、すなわちデータのエントロピーを与えるため、連続時間上界はタイトである。これはペアにおける下限を順序の側から見たものにほかならない。$T \to \infty$ では 2 つのトークンが同時にマスク解除されることはなくなり、超過分 I/T は消える。

1 ステップに複数のトークンをマスク解除すると、サンプラは独立性誤差を被る。ペアの場合、S 回のジャンプでは確率質量の $1/S$ が周辺分布の積からサンプリングされ、T ステップの上界は I/T だけ上昇する。その見返りとして、S 回のネットワーク評価で L 個のトークンを生成できるのに対し、左から右へのモデルは L 回の評価を必要とする。

もっとも、この 2 種類の評価の計算規模は同じではない。キーと値をキャッシュする左から右へのモデル（11.3 節）は、1 ステップにつき新しい 1 トークンだけを処理する。これに対しノイズ除去器には因果マスクがないため、1 つのトークンがマスク解除されるたびにすべての位置の活性化値が変化し、何もキャッシュできず、各評価は全 L 位置にわたる順伝播となる。トークン通過回数で数えれば、コストは約 L 対 SL である。したがって並列デコーディングは逐次ステップ数を減らし、それに伴ってレイテンシを削減する一方、総演算量は通常より多くなる。Fast-dLLM (Wu et al. 2026) はステップ間で近似キャッシュを再利用することでその一部を取り戻し、ブロック拡散（block diffusion）(Arriola et al. 2025) は完了したブロック上に厳密なキャッシュを持ちながらブロック単位で順次デコードすることで削減を図る。

対極に位置するのが BERT (Devlin et al. 2018) であり、テキスト中のトークンの 15% を隠して両側から予測する。11.4 節のマスク言語モデルも同様で、各バイトを確率 15% でマスクする。この単一のマスクレベルに制限すると、損失 Equation 17.71 は（BERT が一部のマスクをランダムなトークンや元のトークンに置き換える点を除けば）隠されたトークンの負の対数尤度の和というそれらのマスクトークン目的関数に帰着する。すなわちマスク言語モデルは 1 ステップの離散拡散モデルである (Austin et al. 2021)。ただし訓練時に 1 つのマスクレベルしか見ていないため、全マスク系列から出発するサンプラに入力すると、訓練時とはかけ離れた入力を見せることになる。

これら両極端の間に位置するのが今日の拡散言語モデルであり、そのノイズ除去器は因果マスクを持たない Transformer に、$[\mathrm{M}]$ 用の語彙項目を追加し、時刻入力を省いたものである (Nie, Zhu, You, et al. 2025)。これらは何をもたらすのだろうか。データが豊富にある状況での尤度ではない。常に新鮮なデータで訓練する場合、マスク拡散が自己回帰モデルと同じ損失に到達するには定数倍多くの計算量が必要になる (Nie, Zhu, Du, et al. 2025)。一方、データが限られていて何度も繰り返し学習する場合にはマスク拡散が上回ることがあり (Prabhudesai et al. 2025)、著者らはランダムマスクがデータ拡張として働くためだと説明している。これらのモデルの強みは、1 回のネットワーク評価で多数のトークンを書き込むことで逐次ステップ数を減らせる並列デコーディングと、空白の両側のテキストに条件付けられる穴埋め生成（infilling）にある。

17.9.5.2 拡散モデルの振り返り

関連文献の節を除けば、本章はここで終わり、「発展」パートもここで完結する。最後に一度全体を振り返っておこう。すべての節を貫いていたのは 1 つの着想である。あらゆるノイズレベルでノイズを含む点からクリーンな点を予測するネットワークはそれ自体が生成器であり、サンプラはその予測を積分することでデータへと変換する。トークンにおいては、ノイズを含む点は一部が消去された系列となり、予測は消去された値の上の分布となり、積分は段階的なマスク解除の列となったが、変分上界はそのまま引き継がれた。

この解答の各要素にはそれぞれ弱点があり、実務家はそれらすべてに直面する。密度は上界とし

てしか得られず、最良の上界を与える重み付けは最良のサンプルを与える重み付けとは一致しない（17.6 節）。第 16 章の敵対的ゲームは回帰に置き換わるものの、サンプリングではネットワークを何度も呼び出す必要がある。元の DDPM 連鎖では 1,000 回、優れたソルバを使っても数十回（17.5 節）かかり、数回で済むのは生徒モデルを蒸留する 2 回目の訓練を行った後だけである（17.8 節）。潜在空間を導入すれば大きな画像も扱えるようになるが、最初に訓練してから凍結するオートエンコーダという 2 つ目のモデルが増える（17.7 節）。そしてテキストでは、1 回の評価あたり多数のトークンを生成しようとすれば独立性誤差が生じ、しかもサンプラは一度書き込んだトークンを取り消せない。

未解決の問いもいくつか残っている。教師モデルの軌跡に沿って訓練された少数ステップの生徒モデルは、依然として教師に及ばない。教師に並ぶかそれを上回る生徒モデルは、分布マッチングにクリティックを加えており、ノイズからサンプルへの決定論的な写像を手放している（17.8 節）。また、最良の尤度と最良のサンプルの両方を同時に達成する単一の重み付けは知られていない。テキストにおけるマスク拡散と自己回帰の計算量ギャップが最大規模のスケールでも残るかどうかは、本書で扱ういかなる実験よりもはるかに大規模な実験によって決着がつきつつある。「言語モデル」パートでは、概観するアーキテクチャの 1 つとして再び拡散言語モデルを取り上げる（31.8 節（英語版））。

17.9.6 まとめ

離散拡散は行確率遷移行列 $\mathbf{Q}_t$ によってカテゴリトークンを破損させる。吸収カーネルのもとでは、各トークンは t ステップ後まで確率 $\bar{\alpha}_t = \prod_{s \le t}(1-\beta_s)$ で生存し、そうでなければ $[\mathrm{M}]$ に置き換えられる。負の対数尤度に対する変分上界は、重み $w_t = (\bar{\alpha}_{t-1} - \bar{\alpha}_t)/(1-\bar{\alpha}_t)$（線形スケジュールでは $1/t$）を持つマスクトークン交差エントロピーの重み付き和へと簡約され、連続時間極限は $\int_0^1 \frac{1}{u}\,\mathbb{E}_u[\ell]\,du$ となる。このとき独立なマスキングにより、マスク位置における最適なクリーントークン分布は時刻非依存になる。逆モデルが各ステップ内で位置ごとに分解されるため、複数のトークンを同時にマスク解除すると独立性誤差が生じる。すなわち 2 トークン分布 p に対して、S 回の等間隔線形ジャンプは $(1-1/S)p + (1/S)p^1 \otimes p^2$ からサンプリングし、$1/S$ に比例する全変動誤差を生じる。また T ステップの上界の下限は、エントロピー $H(X^1, X^2)$ を $I(X^1; X^2)/T$ だけ上回る。マスク解除スケジュールという観点から見ると、マスク拡散は古典的な系列モデルを結びつける。左から右へ 1 ステップに 1 トークンずつマスク解除すれば自己回帰モデル（11.2 節）が得られ、ランダムな順序で 1 ステップに 1 トークンずつマスク解除すれば順序非依存な自己回帰が得られ、単一のマスク率 $u = 0.15$ で訓練すれば BERT（11.4 節）が得られる。

17.9.7 演習問題

1. **一様カーネル**。順連鎖がすべてのステップで一様カーネル $\mathbf{Q}_t = (1-\beta_t)\,\mathbf{I} + \beta_t\,\frac{1}{K}\mathbf{1}\mathbf{1}^\top$（ただし $\beta_t > 0$）を用いるとする。
   1. $\bar{\mathbf{Q}}_t = \bar{\alpha}_t\,\mathbf{I} + (1-\bar{\alpha}_t)\,\frac{1}{K}\mathbf{1}\mathbf{1}^\top$ となること、および $\mathbf{x}_t = \mathbf{x}_0$ となる確率が $\bar{\alpha}_t + (1-\bar{\alpha}_t)/K$ であることを示せ。したがって、ステップ t である値を示しているトークンは再サンプリングされたものである可能性がある。吸収カーネルでは答える必要がないが、このカーネルのもとではすべての位置についてネットワークが答えなければならない問いは何か。
   2. 事後分布 $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ を Equation 17.66 から、$\mathbf{x}_t = \mathbf{x}_0$ の場合と $\mathbf{x}_t \neq \mathbf{x}_0$ の場合に分けて導出せよ。また、$t \ge 2$ では K 個の値すべてが正の確率を持つことを示し、2 点事後分布 Equation 17.70 との違いを確認せよ。
   3. 変分上界のダイバージェンス項は、すべての位置でこの事後分布と逆モデルを比較する。これらを評価するコストが Equation 17.71 の被加数よりも高くなるのはなぜか。

2. [code] **同時にマスク解除されるペア**。サンプラがレベル $T = t_0 > t_1 > \cdots > t_S = 0$ を、$\bar{\alpha}_0 = 1$ かつ $\bar{\alpha}_T = 0$ を満たすスケジュールのもとで用いるとし、$\Delta_k = \bar{\alpha}_{t_{k+1}} - \bar{\alpha}_{t_k}$ を $\bar{\alpha}$ のジャンプ k（$k = 0, \dots, S-1$）における増加量とする。
    1. レベル t_k 上の粗視化した連鎖では、トークンはジャンプ k で確率 Δ_k により、他の位置とは独立にマスク解除される。2 つのトークンが同じジャンプでマスク解除される確率が $\rho = \sum_k \Delta_k^2$ となること、および和が 1 になる S 個の増分のうち等間隔の増分 $\Delta_k = 1/S$ が ρ を最小化することを示せ。
    2. 厳密なネットワークを用いたとき、サンプラが量子化混合分布の 2 トークンを $(1-\rho)\,p + \rho\, p^1 \otimes p^2$ からサンプリングすることを示せ。
    3. 長さ L のトークン系列と等間隔増分に対して、同じジャンプでマスク解除されるトークンペア数の期待値を計算せよ。$L = 1024$ のとき、この期待値を 1 未満に抑えるには何ステップ必要か。また、テキストにおいてこの個数がサンプラの誤差を過大評価しているのはなぜか。
    4. 1 つもトークンをマスク解除しないジャンプでは、次のジャンプへの入力は変化しない。したがって、演習問題 4a により厳密なネットワークがそうであるように、ネットワークが時刻を無視するサンプラでは、先にマスク解除の判定をすべて引き、少なくとも 1 つのトークンをマスク解除するジャンプでのみネットワークを評価できる。線形スケジュールのもとで S 回の等間隔ジャンプを行うとき、評価回数の期待値が $S\,(1 - (1 - 1/S)^L)$（トークン数 L の場合）となることを示せ。これをペアの $S = 32$、2 値画像の $S = 100$ および $S = 784$、ならびに $L = S = 1024$ について計算せよ。
    5. 784 ステップによる 17.9.4.2 節のサンプルには、100 ステップにおける過剰スペックルの 4 分の 1 から 3 分の 1 が残っている。S 回の等間隔ジャンプにおいて、内部ピクセルが 4 つの隣接ピクセルの少なくとも 1 つと同じジャンプを共有する確率が $1 - (1 - 1/S)^4$ であることを示し、$S = 4$、100、784 についてその値を計算せよ。次に `unet` を 2 倍の更新回数だけ訓練し、100 回および 784 回のジャンプで再びスペックルを数えよ。どちらの個数が減少し、そこから残存していた過剰分のうちネットワークに起因する割合について何が言えるか。

3. **2 通りの逆パラメータ化**。●逆モデル Equation 17.69 はネットワークの予測に関して事後分布を平均化する。これに対し Austin et al. (2021) は次式のように置いた。

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \propto \sum_{\hat{\mathbf{x}}_0} q(\mathbf{x}_{t-1}, \mathbf{x}_t \mid \hat{\mathbf{x}}_0)\, \tilde{p}_\theta(\hat{\mathbf{x}}_0 \mid \mathbf{x}_t) \tag{17.72}$$

ここで $\tilde{p}_\theta$ はネットワーク出力であり、各候補を $q(\mathbf{x}_t \mid \hat{\mathbf{x}}_0)$ で再重み付けする形になっている。ほとんどのカーネルではこれら 2 つの形式は異なる。データから出発した順過程の条件付き分布を $q(\mathbf{x}_0 \mid \mathbf{x}_t)$ および $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ と書くことにする。

    1. 単一のトークンを考える。Equation 17.69 の予測が $q(\mathbf{x}_0 \mid \mathbf{x}_t)$ に等しいならば、あらゆるカーネルに対して逆モデルは真の逆遷移 $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ に一致することを示せ。
    2. 吸収カーネルのもとでは、マスクされた位置において重み $q(\mathbf{x}_t \mid \hat{\mathbf{x}}_0)$ がすべての候補に対して等しくなり、したがって 2 つの形式が一致することを示せ。

4. [code] **時刻に依存しない予測**。●本節では、吸収カーネルのもとでマスク位置における最良の予測が t に依存しないことを論じた (Ou et al. 2025)。

    1. この議論を短い証明として書き下せ。吸収順過程のどの性質を用いているか。また、一様カーネルでも同じことが成り立つか。
    2. 11.4 節のマスク言語モデルは各トークンを独立に確率 15% でマスクする。(a) により、その厳密な予測はそのマスク密度のパターンにおいてサンプラが必要とする予測と一

致する。長さ $L = 512$ トークンの系列に対して、そのようなモデルが訓練中に見るマスク割合の範囲（たとえば 15% の前後 3 標準偏差以内）を求めよ。線形スケジュールのもとで $S = 512$ 個の等間隔レベルを用いて全マスク系列から生成を行うとき、全ステップのうちこの範囲のマスク割合をネットワークに見せるステップの割合はいくらか。

3. `MaskedMLP` は時刻を入力として受け取る。第 1 トークンがマスクされ第 2 トークンが観測されているパターンを固定し、`pair_model` の予測を第 2 トークンの全 32 通りの値について $t \in \{1, 16, 31\}$（すなわち時刻入力 $t/32$）で計算せよ。次に、それぞれ第 2 トークンの周辺分布で平均した 2 種類のカルバック・ライブラー・ダイバージェンスを計算せよ。すなわち、$t = 16$ および $t = 31$ における予測の、$t = 1$ における予測からのダイバージェンスと、各 t における予測の、`table` の厳密な条件付き分布からのダイバージェンスである（確率は 10^{-12} でクランプせよ）。これら 2 種類の値の大きさから、訓練済みネットワークによる時刻入力の使われ方について何が言えるか。

5. [code] **厳密なネットワークの下限。**●$\bar{\alpha}_0 = 1$ かつ $\bar{\alpha}_T = 0$ を満たすスケジュールの増分を $\Delta_t = \bar{\alpha}_{t-1} - \bar{\alpha}_t$ とし、量子化混合分布の 2 トークンを考える。

   1. 真のテーブルにわたって平均した厳密なネットワークの上界 Equation 17.71 が次式となることを示せ。

$$H(X^1, X^2) + I(X^1; X^2) \sum_{t=1}^{T} \Delta_t^2. \tag{17.73}$$

   ステップ t の項を 3 つのマスクパターンとその確率を用いて書き下せば、$(1-\bar{\alpha}_t)^2$ の望遠鏡和によって t にわたる和を計算できる。線形スケジュールでは $H(X^1, X^2) + I(X^1; X^2)/T$ となることを確かめよ。

   2. スケジュール $\bar{\alpha}_t = 1 - (t/T)^2$ に対してこの和を閉形式および $T = 32$ で評価し、線形スケジュールの下限と比較せよ。$T \to \infty$ の極限でスケジュールの選択が結果に影響しなくなるのはなぜか。

   3. `MaskedMLP` と同じ呼び出しシグネチャを持ち、`table` から計算した厳密な条件付き分布（および両トークンがマスクされている場合は周辺分布）の対数を返す関数を書け（確率は 10^{-12} でクランプせよ）。これを `exact_bound` に、2 つのスケジュールをそれぞれ `alpha_bar` として渡せ。

   4. `MaskedMLP` を `masked_loss` により、(b) のスケジュールのもとで `pair_model` と同じ設定を用いて訓練し、その厳密な上界をその下限および `pair_model` の厳密な上界と比較せよ。

6. **順序非依存な上界。**●演習問題 4a により厳密なネットワークがそうであるように、ネットワークが時刻を無視するとし、$\ell(i, U) = -\log p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_0^U)$ を位置 i における損失（集合 U に属する位置だけがちょうどマスク解除されているとき）とする。このとき 17.9.2.3 節の上界の連続時間極限をマスク確率 u に関する積分として書くと、$\ell(i, U)$ の $i \notin U$ にわたる重み付き和になる。Hoogeboom et al. (2022) と Ou et al. (2025) は、これが順序非依存な自己回帰モデル (Uria et al. 2014) の期待負の対数尤度

$$\mathbb{E}_\sigma \Big[\sum_{k=1}^{L} -\log p_\theta(\mathbf{x}_0^{\sigma(k)} \mid \mathbf{x}_0^{\sigma(1)}, \dots, \mathbf{x}_0^{\sigma(k-1)}) \Big], \tag{17.74}$$

に等しいことを示した。ここで σ は L 個の位置の一様ランダムな置換である。

1. 積分における $\ell(i, U)$ の重みを $|U| = L - m$ のときに求めよ。評価にはベータ積分 $\int_0^1 u^{m-1}(1-u)^{L-m}\,du = (m-1)!\,(L-m)!/L!$ を用いることができる。
2. ランダムな順序が最初に U の位置をマスク解除し、次に位置 i をマスク解除する確率を求め、2 つの式が一致することを結論せよ。
3. 厳密なネットワークではどの順序も同じ値を与える。それはどの値か。演習問題 5a の $T \to \infty$ の場合と比較せよ。
4. イェンセンの不等式を用いて、この式が一様ランダムに順序を引いてからその順序でトークンをサンプリングするモデルの負の対数尤度の上界にもなることを示せ。

7. [extended] **マスク拡散言語モデル**。●●Prabhudesai et al. (2025) は、マスク拡散モデルが自己回帰モデルよりも繰り返しデータをうまく活用すると報告している。この問題では意図的にデータを繰り返す。文字レベルのウィンドウ `d2l.TimeMachine(batch_size=64, num_steps=128, tokenization='char', num_train=100000, num_val=3000)`（8.2 節）を用いよ。その 100,000 個の訓練ウィンドウは、最初の 100,000 文字位置（『The Time Machine』の冒頭）から始まるため、2,000 回の更新を 64 個の 128 文字ウィンドウからなるミニバッチで行うと、これらの位置を約 160 回通過することになる。

   1. `GPT`（11.2 節）をこれらのウィンドウ上で 2,000 回更新にわたりその訓練レシピ `LMOptimizer` で訓練し、訓練損失と検証損失を 200 回更新ごとに記録せよ。検証損失はどの更新回数で最小に達し、その後 2 つの損失はどう振る舞うか。
   2. ノイズ除去器を `d2l.GPT` のサブクラスとして構築せよ。トークン数を `vocab_size + 1`（最後のトークンがマスク）とし、本節の議論が許すとおり時刻を受け取って無視する順伝播を持たせること。（ヒント：因果マスクを外すには、入れ子クラス `CausalAttention` をオーバーライドし、フレームワークの融合アテンションをマスクなしで呼び出せばよい。）これを `masked_loss` と線形の `alpha_bar`（$T = 128$ ステップ）を用いて 2,000 回更新にわたり訓練せよ。
   3. 200 回更新ごとに、検証ウィンドウ上で 1 ウィンドウあたり複数のマスクを用いて文字あたりビット数での上界を推定せよ。これを文字あたりビット数で表した `GPT` の検証損失と並べてプロットし、各曲線の最良値とそれに到達する更新回数を比較せよ。ノイズ除去器の上界は、`GPT` の損失よりも、`GPT` が過学習を始める前のすべての更新回数において十分上に留まるはずである。この差はスケーリング研究が測定しているコストそのものであり、バグではない。
   4. ノイズ除去器の訓練を 8,000 回更新まで継続せよ（2,000 回更新の時点でまだ上界が下がり続けている場合）。どちらのモデルの方が遅く過学習を始めるか。また、ランダムにマスクされたウィンドウ上で訓練することは、何度も見たウィンドウからノイズ除去器が暗記できる内容をどのように変えるか。
   5. $S \in \{128, 32, 8\}$ ステップでウィンドウをサンプリングし、`GPT` のサンプルと比較せよ。少数ステップのサンプルはどこで破綻するか。その破綻を演習問題 2c と関連付けて説明せよ。

17.10 関連文献

本節では、本章で展開した手法を発展させる専門書、チュートリアル、参照実装、および一次文献をまとめる。

17.10.1 専門書、講義、コード

- **The Principles of Diffusion Models** (Lai et al. 2025) は、変分・スコアベース・フローベースの各経路に沿って拡散モデルを導出し、それらがどこで合流するかを示している。

- MIT の講義 6.S184 の講義ノート、講義、ラボ (Holderrieth and Erives 2025) は、フローマッチング、潜在拡散 Transformer、および離散拡散（discrete diffusion）を扱っている。
- **Diffusion Meets Flow Matching** (Gao et al. 2024) は、スケジュールの再スケーリングのもとで DDIM とオイラー法のフローマッチングサンプラが等価であることを、対話的な図を用いて解説している。
- **Flow Matching Guide and Code** (Lipman et al. 2024) とそのライブラリは、17.2 節に対する完全な証明と参照コードを提供している。
- **Step-by-Step Diffusion** (Nakkiran et al. 2025) は、確率解析（stochastic calculus）を用いず初等確率論のみから、決定論的サンプラと確率的サンプラ（17.3 節および 17.5 節）を導出している。
- Sander Dieleman のブログでは、周波数空間でのノイズ除去 (Dieleman 2024)、潜在表現 (Dieleman 2025)、フロー写像 (Dieleman 2026b)、および言語に対する連続拡散と離散拡散の比較 (Dieleman 2026a) が検討されている。
- 参照実装としては、DDPM 公式リポジトリ (Ho et al. 2020)、Score SDE リポジトリ (Y. Song, Sohl-Dickstein, et al. 2021)、および MDLM リポジトリ (Sahoo et al. 2024) がある。
- **Deep Generative Models** 講義 (Kuleshov 2023)、Yang Song のスコアベースモデルのチュートリアル (Song 2021)、Lilian Weng の **What Are Diffusion Models?** (Weng 2021)、および Luo (2022) は、スコアマッチングと変分拡散に関する平易な解説を提供している。

一次論文のなかでも、次の 5 編はとりわけ基礎をなすものである。Karras et al. (2022)（EDM）はサンプラ、前処理（preconditioning）、ノイズスケジュールを切り分け、Esser et al. (2024)（Stable Diffusion 3）はレクティファイドフローとマルチモーダル拡散 Transformer を組み合わせた。また、C. Lu et al. (2025)（DPM-Solver++）はガイダンス下での高次ソルバを解析し、Lu and Song (2025)（sCM）は大規模環境における連続時間一貫性モデル（consistency model）を安定化し、Sahoo et al. (2024)（MDLM）はマスク付き離散拡散の見通しのよい定式化を与えている。隣接するトピックは 16.8 節（敵対的蒸留および分布整合蒸留、FID 評価）ならびに 38.8 節（指数積分器、スコア誤差の上界、シュレーディンガーブリッジ）で扱う。

17.10.2 エネルギーベースモデルとスコアマッチング

エネルギーベースモデルの訓練：@Song.Kingma.2021 は、正規化されていない密度に対する最尤法による訓練を概観している。コントラスティブダイバージェンスはデータ点を初期値として短い MCMC 連鎖を開始するため (Hinton 2002)、離れたモード間での混合が妨げられるが（17.1 節）、持続的コントラスティブダイバージェンス（persistent contrastive divergence）はパラメータ更新をまたいで連鎖の状態を保持する (Tieleman 2008; Du and Mordatch 2019)。

ノイズ対照推定：微分によって正規化定数を消去する代わりに、Gutmann and Hyvärinen (2012) は正規化定数を学習可能なパラメータとして扱い、既知のノイズ分布に対するロジスティック回帰によってモデルを訓練する。

スコアマッチングの限界：低密度の谷を隔てたモードの重みに対してフィッシャーダイバージェンスが不感になる性質（17.1 節）は Wenliang and Kanagawa (2020) によって解析されており、Koehler et al. (2022) は等周定数（isoperimetric constant）を用いてスコアマッチングの統計的効率に境界を与えている。

ノイズラダーのヒューリスティクス：@Song.Ermon.2020 は、幾何ノイズラダー（17.2 節）に対して 4 つの実践的な規則を与えた。すなわち、最大ノイズスケールを訓練データ間の最大ペアワイズ距離に設定すること、高確率シェルが重なり合うほど隣接レベルを十分に近づけて選ぶこと（演習問題 4、17.2 節）、ランジュバン法のステップサイズを調整すること、および重みに指数移動平均を適用することである。

経験ベイズ：トゥイーディーの公式は経験ベイズ推定（empirical Bayes estimation）に端を発

しており (Robbins 1956)、Efron (2011) が高次元推定における選択バイアスの補正にこれを用いる手法を広めた。

17.10.3 画像ネットワークとガイダンス

大規模な U-Net アーキテクチャ：@Dhariwal.Nichol.2021（ADM）は、DDPM の U-Net に多重解像度アテンション、残差リサンプリング、および適応グループ正規化を追加した。Karras, Aittala, Lehtinen, et al. (2024)（EDM2）は、活性化値と重みの大きさを保つように U-Net を再設計し、周期的なチェックポイントからの事後的な EMA 再構成を導入した。

飽和のないガイダンス：大きなガイダンス強度はピクセル強度を飽和させることがある（演習問題 5、17.4 節）。Sadat et al. (2025) は、ガイダンスによる更新を条件付き予測に平行な成分と直交する成分へ分解し、平行成分を減衰させることで飽和を防いでいる。

空間的条件付け：ControlNet (Zhang et al. 2023) は、ゼロ初期化された層を介して接続したエンコーダの訓練可能なコピーを用いることで、事前学習済みのテキスト画像生成モデルを空間マップ（エッジ、深度、姿勢）で条件付ける。

17.10.4 評価、暗記、汎化

プロンプトアライメント：フレシェ距離はプロンプトへの忠実度ではなく分布の類似度を測るため、CLIPScore (Hessel et al. 2021) は共通埋め込みにおける画像とキャプションの類似度を測定し、一方 GenEval (Ghosh et al. 2023) は物体検出器を用いて物体の数、位置、属性を検証する。

暗記：@Carlini.Hayes.Nasr.ea.2023 は大規模なテキスト画像生成モデルから訓練画像とほぼ同一の重複画像を抽出しており、Biroli et al. (2024) は逆過程においてクラスの確定（種分化、speciation）と訓練データ点の暗記（崩壊、collapse）に対応する異なる力学的レジームを特定している。

ノイズ除去器の汎化：ノイズ除去目的関数の厳密な経験最小点は有限の訓練セットを暗記してしまうが、Bertrand et al. (2025) は目標ノイズだけでは汎化を説明できないことを示している。互いに素な部分集合で訓練されたノイズ除去器はほぼ同一のスコア場を学習し (Kadkhodaie et al. 2024)、それは主として畳み込みの局所性と同変性によって規定される (Kamb and Ganguli 2025)。サーベイについては Massias and Bertrand (2026) を参照のこと。

17.10.5 サンプラ、編集、逆問題

ガイダンス付きサンプリングのためのソルバ：大きなガイダンス強度は高次のノイズ予測ソルバを不安定にする（演習問題 3、17.5 節）。そこで DPM-Solver++ (C. Lu et al. 2025) は代わりにクリーン画像の推定値を用いて多段更新をパラメータ化し、UniPC (W. Zhao et al. 2023) は追加計算コストのない修正子（corrector）ステップを加えている。

ODE ソルバにおける確率性：@Karras.Aittala.Aila.ea.2022 はホイン法の各ステップの前に少量のノイズを注入し、一方 Restart サンプリング (Xu et al. 2023) は決定論的な ODE 積分とより大きな順方向ノイズのジャンプを交互に繰り返すことで、蓄積した離散化誤差を縮小させる。

厳密な反転：DDIM 反転（DDIM inversion、17.5 節）は分類器なしガイダンスのもとで離散化誤差を蓄積するため、ヌルテキスト反転（null-text inversion）(Mokady et al. 2023) はガイダンスなしの軌跡の周囲で無条件埋め込みを最適化し、一方 EDICT (Wallace et al. 2023) は結合されたアフィン変換を用いて厳密な可逆性を実現する。

再訓練なしの逆問題：ブラー除去や超解像などの線形逆問題に対し、Diffusion Posterior Sampling（DPS）(Chung et al. 2023) は事後平均 $\hat{\mathbf{x}}_0(\mathbf{x}_t)$ において測定尤度の勾配を近似し、DDRM (Kawar et al. 2022) は測定作用素のスペクトル領域で動作する。

17.10.6 スケジュール、重み付け、尤度

学習される逆過程の分散：コサインスケジュール（17.2 節）と並び、Nichol and Dhariwal (2021) は逆過程の分散を $\tilde{\beta}_k$ と β_k の間の学習される補間としてパラメータ化し、変分境界を用いて訓練している。

Min-SNR 重み付け：@Hang.Gu.Li.ea.2023 は、クリーン画像の二乗誤差に $\min(\mathrm{SNR}_t, \gamma)$ の重みを掛けることで（等価的にノイズ誤差へ $\min(1, \gamma e^{-\lambda})$ の重みを掛けることに相当）、低ノイズのステップが勾配更新を支配するのを防いでいる。

尤度最適化：変分拡散モデル（variational diffusion model、VDM）(Kingma et al. 2021) は ELBO 推定量の分散を最小化するようにノイズスケジュールを最適化する。また、Y. Song, Durkan, et al. (2021) は連続時間スコアマッチングに対する尤度重み付けスケジュールを導出し、Zheng et al. (2023) は確率フロー ODE に対する厳密な尤度訓練と評価を改善している。

17.10.7 潜在表現、テキスト、動画

テキスト条件付け：大規模なテキスト画像生成モデルは、凍結した CLIP (Radford et al. 2021) や T5 (Raffel et al. 2020) のテキストエンコーダを用い、クロスアテンション（17.7 節）または画像・テキスト結合自己アテンション (Esser et al. 2024) を介して条件付けを行う。Saharia et al. (2022) は、テキストエンコーダをスケールする方が、U-Net をスケールするよりもプロンプトアライメントを大きく向上させることを示した。

表現アライメントとオートエンコーダ：REPA (S. Yu et al. 2025) は、中間の隠れ状態を事前学習済みの視覚エンコーダと整合させることで潜在 DiT の訓練を高速化する。Deep Compression Autoencoders（DC-AE）(Chen et al. 2025) は空間方向のダウンサンプリングを $8\times$ 超へと拡大し、表現オートエンコーダ（representation autoencoder）(Zheng et al. 2026) は凍結したセマンティックエンコーダ（DINO など）と学習されるデコーダを組み合わせる。

動画生成：動画拡散モデルは、3D 時空間オートエンコーダと、空間および時間にわたる分解型または結合型のアテンションを適用する (Ho et al. 2022; Blattmann et al. 2023; Polyak et al. 2024)。

17.10.8 少数ステップ生成

漸進的蒸留とガイダンス蒸留：漸進的蒸留（progressive distillation）(Salimans and Ho 2022) は教師モデルのステップ数を反復的に半減させる手法であり、低 SNR における数値安定性のために $\mathbf{v}$ 予測を導入した。Meng et al. (2023) は、分類器なしガイダンスをガイダンス強度 γ で条件付けられた単一パスへと蒸留している。

大規模環境における一貫性とフロー写像：改良型一貫性訓練（improved consistency training、iCT）(Song and Dhariwal 2024) は離散時間において教師モデルを不要にし、一方 sCM (Lu and Song 2025) は三角関数経路と接線正規化（tangent normalization）を用いて ImageNet スケールでの連続時間一貫性訓練を安定化する。Improved MeanFlow (Geng et al. 2026) は瞬間速度へ直接回帰し、Boffi et al. (2025) はオイラー形式、ラグランジュ形式、および合成形式のフロー写像恒等式にわたる自己蒸留を解析している。

17.10.9 離散拡散

代替カーネルと再マスキング：離散フローマッチング (Gat et al. 2024) は、単体上の確率経路を通じて離散生成を定式化する。一様破壊カーネルを用いれば任意のステップでトークンを修正でき、Sahoo et al. (2025) は一様離散拡散をしきい値処理されたガウス拡散と結びつけている。

ReMDM (G. Wang et al. 2025) は吸収型拡散に再マスキングを導入し、Kim et al. (2025) は数独などの推論タスクにおいて確信度順のアンマスキングが高い効果を発揮することを実証している。

拡散言語モデル：LLaDA (Nie, Zhu, You, et al. 2025) は 8B のマスク付き拡散言語モデルをゼロから訓練し、一方 Dream (J. Ye et al. 2025) は自己回帰モデルのチェックポイントから初期化する。ブロック拡散（block diffusion）(Arriola et al. 2025) はブロック間の自己回帰生成とブロック内の並列拡散を組み合わせ、Fast-dLLM (Wu et al. 2026) は KV キャッシュの再利用（11.3 節）を追加している。なお K. Zheng, Chen, Mao, et al. (2025) は、32 ビットの Gumbel-max サンプリングがサンプリング温度を人為的に低下させ、サンプルエントロピーを犠牲にして報告される生成パープレキシティを実態より良く見せてしまうことに警鐘を鳴らしている。

17.10.10 事後学習と視覚・テキスト以外の応用

強化学習と選好最適化：DDPO (Black et al. 2024) と Flow-GRPO (Jie Liu et al. 2025) は、拡散モデルおよびフローモデルの確率的な逆遷移に対して方策勾配更新（15.8 節）を適用する。Diffusion-DPO (Wallace et al. 2024) は、計算困難なモデルの対数尤度を拡散モデルの ELBO で置き換えることにより、直接選好最適化を適合させている。

音声、ロボティクス、タンパク質：DiffWave (Kong et al. 2021) は生の音声波形に拡散を適用する。拡散方策（diffusion policy）(C. Chi et al. 2023) は視覚観測に条件付けられた多峰性のロボット行動軌跡をモデル化し、直接的な MSE 方策回帰におけるモード平均化の破綻を回避する。マルチモーダルなタンパク質モデル (Campbell et al. 2024) は、連続的な 3D 原子座標と離散的なアミノ酸配列を同時に拡散させる。

深層学習のための数学

本書の主要な各章では、数学的ツールを**必要になったその場で**導入しており、勢いを失うことなく実際に動作するモデルを構築・訓練できるよう、それぞれの結果を幾何学的・統計的・物理的な直観とあわせて提示している。しかし、既存のレシピを適用する段階から一歩進んで、微妙な訓練の失敗を診断したり、最前線の研究論文を批判的に読み解いたり、新たな手法を考案したりするには、いずれその背後にある数学というエンジンの内部を覗き込む必要がある。なぜ勾配降下法（gradient descent）の学習率には、ヘッセ行列（Hessian）の最大固有値によって定まる厳格な上限があるのか。なぜ交差エントロピー（cross-entropy）を最小化すると真の条件付き分布が推定でき、なぜ 2 つの分布に重なりがないとイェンセン・シャノン・ダイバージェンスは飽和してしまうのか。そして、画像に日々ガウスノイズを加えていくことが、なぜ純粋なノイズを写真へと戻す逆時間の微分方程式を定めることになるのだろうか。

全 6 章からなる本付録は、これらの問いに余すところなく答える。本付録は、互いに補完し合う次の 2 つの目的を果たすよう設計されている。

- **本編と対になるリファレンス**。「**基礎**」パートや「**発展**」パートの各章で数学の定理や恒等式を用い、その実践的な帰結に焦点を当てている箇所では、完全な導出・証明・境界条件（boundary condition）を扱う本付録の対応する節が参照されている。すでに馴染みのあるトピックについては、ここでの該当する章を読み飛ばし、相互参照を見て興味が湧いたときにだけ必要な箇所を拾い読みすればよい。
- **深層学習の数学を扱う講義のための自己完結した教科書**。最初から最後まで通読すれば、これら 6 つの章は深層学習に特化した一貫性のある現代的な数学カリキュラムとなる。病的な例の追求そのものに深入りすることなく、現代の機械学習で実際に使われる範囲まで、各主題を第一原理から展開している。

本書の他のパートと同様に、本付録のすべての節では数学的な導出に実行可能なコードと計算実験（PyTorch、JAX、TensorFlow、MXNet で実装）を組み合わせており、抽象的な定理を自ら調べ、プロットし、検証できる具体的な数値現象へと落とし込んでいる。

- **線形代数（第 33 章）**。深層ネットワークはデータとパラメータをベクトル、行列、テンソルとして表現し、各層が空間の幾何学構造を変換する。本章では長さ、角度、内積、射影、ベクトルおよび行列のノルムから出発し、部分空間を回転させ、引き伸ばし、潰す能動的な線形変換として行列を捉える。続いて、高次元のダイナミクスを支配するスペクトル分解を展開する。具体的には、漸化式や最適化で繰り返される行列積がいかに固有値に支配されるかを示す**固有値分解**（eigendecomposition）、ゲルシュゴリンの円や条件数からランダム行列理論およびマルチェンコ・パスツール則に至る**実践におけるスペクトル**、そしてパラメータ効率的ファインチューニング（LoRA）やスペクトル正規化の基礎を成す**特異値分解**（singular value decomposition、SVD）、ムーア・ペンローズ擬似逆行列、エッカート・ヤングの低ランク近似を扱う。
- **微分積分（第 34 章）**。数十億個のパラメータを持つニューラルネットワークの訓練は、すべてのパラメータに同時に摂動を与えたときスカラーの損失がどう変化するかを計算でき

るかに全面的に支えられている。本章では 1 変数の導関数、テイラー級数、数値微分の落とし穴から始め、多変数の勾配、方向微分、ヤコビ行列（Jacobian matrix）、ヘッセ行列による曲率へと進む。続いて計算グラフ上で多変数の連鎖律（chain rule）をたどり、**自動微分**（automatic differentiation）の仕組みを解き明かす。そこではフォワードモードとリバースモード（**誤差逆伝播法**、backpropagation）による累積を対比し、なぜリバースモードを使えば 1 回の順伝播のコストの小さな定数倍で数百万個のパラメータに関する厳密な勾配を計算できるのかを示す。

- **最適化（第 35 章）**。本章では、第 9 章で観察した経験的現象の背後にある形式的な証明を与える。凸集合と凸関数の幾何学（イェンセンの不等式、凸性の一次および二次特徴づけ）を確立し、L 平滑性と μ 強凸性が条件数 $\kappa = L/\mu$ を通じて勾配降下法の収束率とステップサイズの安定性をいかに支配するかを証明する。さらに、悪条件の二次モデルにおけるモメンタムやニュートン法・準ニュートン法による前処理（preconditioning）を解析し、ラグランジュ乗数、カルーシュ・キューン・タッカー（KKT）最適性条件、ラグランジュ双対性（Lagrangian duality）を用いた制約付き最適化（constrained optimization）を展開する。
- **確率と統計（第 36 章）**。機械学習は、ノイズを含むデータ生成や確率的なミニバッチサンプリングから標本外汎化に至るまで、あらゆる段階で不確実性のもとで推論する。本章では確率空間、条件付き分布、ベイズ則、期待値、主要なパラメトリック分布族（ベルヌーイ、カテゴリ、ガウス、ラプラス、指数）から出発し、経験平均のまれな逸脱を抑える古典的な**集中不等式**（concentration inequality）である Markov、Chebyshev、Hoeffding の各不等式へと積み上げる。続いて統計的推論と学習理論を展開し、最尤推定（maximum likelihood estimation、MLE）、フィッシャー情報量、クラメール・ラオの下界、ベイズ的な**最大事後確率推定**（maximum a posteriori、MAP）、十分統計量、仮説検定、信頼区間、そして**汎化**（generalization）の理論的基礎（経験リスク最小化、一様収束、ラデマッハ複雑度、VC 次元、バイアス・分散のトレードオフ）を扱う。
- **情報理論（第 37 章）**。深層学習における分類、言語モデリング、生成のほぼすべての目的関数は、情報理論に源流を持つ。本章では Claude Shannon による自己情報量（サプライザル）とエントロピーの公理から出発し、近似的なモデル分布 Q を真のデータ分布 P の代わりに用いることで生じる符号化の非効率性を**交差エントロピー**と**カルバック・ライブラー（KL）ダイバージェンス**がいかに測るかを示したうえで、微分エントロピーと最大エントロピー原理を通じてこれらの概念を連続確率変数へと拡張する。さらに視野を広げ、KL ダイバージェンスが機能しない場面で確率分布を比較する枠組みとして、f ダイバージェンス、GAN の背後にある変分双対表現、標本空間の基礎にある幾何学構造を反映する**ワッサーシュタイン最適輸送**（Wasserstein optimal transport）距離を扱う。章の締めくくりでは、**相互情報量**（mutual information）、データ処理不等式（data processing inequality）、変分情報ボトルネック、そして現代の対照表現学習を駆動する InfoNCE 下界を解説する。
- **ダイナミクス、フロー、拡散（第 38 章）**。連続正規化フローや状態空間モデルからノイズ除去拡散、フローマッチングに至るまで、現代の深層学習における最も強力なアイデアの多くは連続時間で最も自然に理解できる。最終章となる本章では、ダイナミクスのツールキットを基礎から構築する。具体的には、常微分方程式（ordinary differential equation、ODE）系、ベクトル場、相図（phase portrait）、線形安定性、数値積分器（オイラー法、ルンゲ・クッタ法、シンプレクティック・リープフロッグ法）、および連続随伴感度法によって訓練される**ニューラル ODE** を扱う。続いて、**連続の式**（continuity equation）と Hutchinson のトレース推定量を通じて速度場が確率密度全体をどのように輸送するかを追跡したのち、ランダム性を導入する。すなわち、ブラウン運動、**伊藤確率微分方程式**（Itô stochastic differential equation、SDE）、伊藤の補題、オルンシュタイン・ウーレンベック過程、**フォッカー・プランク方程式**、さらに Anderson の逆時間 SDE と**確率フロー ODE** を導き、スコア関数 $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ を学習することでニューラルネットワークが純粋なガウスノイズを任意のターゲット分布からのサンプルへと変換できる理由を明らかにする。

Chapter 33

線形代数

本書に登場するすべてのニューラルネットワークは、行列積から組み立てられている。線形層は入力に重み行列を掛け、アテンションはクエリにキーを掛け、リカレントネットワークはステップごとに隠れ状態に同じ行列を掛ける。1.3 節では、行列が作用するベクトルに対して実際に**何をするのか**を問うことなく、これらの積を計算した。選択章である本章では、証明も含めてこの幾何学的な問いに取り組む。本文の内容は本章に一切依存しないが、行列積の背後にある幾何学が見えてくると、本書に登場するほぼすべてのモデルの読み解き方が変わるだろう。

784 ピクセルの Fashion-MNIST 画像を 256 個の特徴量へ写像することを考えよう。2 枚の画像はどれほど似ているか、この層はどれだけの情報を捨てるか、そして同じ線形変換（linear map）を 50 回続けて適用すると何が起きるだろうか。784 次元の空間に図を描くことはできないため、その答えは平面幾何を任意の次元へと拡張する代数から得られる。

複雑な行列も、適切な方向に沿って見れば見通しがよくなる。1810 年頃、Gauss は未知数より多くの観測値に対して最小二乗法（least squares）で小惑星パラスの軌道を当てはめたが、これは本章の最初の節で解き、最後の節で再び取り上げる問題である。正規直交軸のもとでは長さと角度がドット積に帰着し、固有ベクトル（eigenvector）に沿えば正方行列は各方向を 1 つのスカラー倍にし、1 組の特異基底のもとでは**任意の**行列が独立な伸縮へと分解される。アテンションがなぜスコアを $\sqrt{d}$ で割るのか（10.2 節）、リカレント勾配がいつ爆発するのか（8.5 節）、そして低ランクアダプタが何を表現できるのかは、これらの考え方から説明できる。

本章は 2 つの段階で進む。第 1 段階では、行列を 1 回だけ適用する。33.1 節では長さ、角度、および部分空間（subspace）内で最も近い点を与える直交射影（orthogonal projection）を高次元へ拡張し、33.2 節ではどの方向が 0 に潰れるか、また写像がいつ可逆になるかを追跡する。第 2 段階では、行列に合わせた座標系を選ぶ。対称行列に対する固有ベクトルとスペクトル定理（33.3 節）、固有値（eigenvalue）だけでは捉えきれない非正規な振る舞い（33.4 節）、そして任意の形状に対する特異ベクトル（singular vector）と最適な低ランク打ち切り（33.5 節）を順に扱う。本章全体を通じて、安易な思い込みを戒める 2 つの小さな行列を一貫して用いる。複素固有値を持つ $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ と、固有ベクトルの張る空間（span）が平面全体に満たないせん断（shear）$\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ である。

初読の際は、n 次元の行列式（determinant）や、33.4 節における Gershgorin の円盤、PageRank、擬スペクトル（pseudospectrum）の議論は流し読みしてかまわない。数値的な固有値ソルバと線形ソルバについては 33.6 節で概観する。

33.1 ベクトル、角度、射影

「基礎」パートの各章を通して、あらゆる入力をベクトルとして表してきた。画像はピクセル値の並びになり、単語は埋め込みになった。機械学習モデルがこれらのベクトルに対して行う操作は、突き詰めれば 3 つの幾何学的な操作に集約される。2 つの埋め込みが似ているかを測るときのようにベクトルを**比較**し、目的変数の組に最もよく当てはまる特徴量の組み合わせを求めるときのようにベクトルを**当てはめ**、分類器が入力を境界のどちら側に振り分けるかを決めるときのようにベクトルを**分離**する。2 次元平面では、これらの操作はいずれも長さ、角度、影からなるおなじみの図として描ける。しかし実際の入力は、誰にも図示できない何百、何千もの次元の空間に存在する。そのような高次元空間でも、こうした幾何学的なイメージはなお意味を持つのだろうか。

T シャツとズボンの見分けが具体的なテストになる。28×28 の Fashion-MNIST 画像（3.2 節）はそれぞれ 784 個のピクセル輝度の並びであり、$\mathbb{R}^{784}$ 内の 1 点である。考えうる最も単純な規則を試してみよう。平均的な T シャツと平均的なズボンを計算し、新しい各画像をより近いほうの平均に割り当てるのである。平面上なら「より近い」かどうかは目で見て確かめられるが、$\mathbb{R}^{784}$ では画像 1 枚あたり 1 回のドット積で済む計算になる。これら 2 つのクラスのテスト画像 $2{,}000$ 枚に対して、この規則は約 92% の割合で正しい判定を下す。これほど単純な規則がこれほどうまく機能するという事実から、2 つの点群は十分に離れて位置しており、目に見えない空間でも距離のような幾何学的概念が意味を保っていることがわかる。

平面の幾何を任意の次元へと運ぶのはドット積である。長さはドット積の平方根であり、あるベクトルの別のベクトルへの影はドット積の比で表され、2 つのベクトルのなす角もドット積から直接定義される。本節では、この幾何学を 3 つの段階で展開する。第 1 に、1 組のベクトルを対象として、直線への射影、2 つのベクトルのなす角、およびコーシー・シュワルツの不等式を調べる。この不等式は 1.3 節で代数的に証明したが、ここでは直角三角形から改めて証明する。第 2 に、本章の残りの部分で基盤となる部分空間の言葉（張る空間、線形独立性、基底、次元）を導入し、QR 分解を計算ツールとして最小二乗法を射影として解く。第 3 に、これらの考え方を高次元の画像へと持ち込み、$\mathbb{R}^{d}$ において無関係なベクトル同士がどれほど似ているかを問い、線形分類器を超平面として読み解く。

なかでも特に重要な結果が 3 つある。射影の残差が射影先の直線や部分空間と直交し、それゆえに射影が最近接点になること、任意の次元で角度が矛盾なく定義されることを保証するコーシー・シュワルツの不等式、そして典型的なスケールが $1/\sqrt{d}$ となる $\mathbb{R}^{d}$ 内のランダムな方向の間のコサインである。データセットの読み込みを除けば、本節のコードはすべて素の NumPy で書かれており、どのフレームワークでも同一に動作する。

33.1.1 ベクトルとその幾何学

33.1.1.1 点と方向

ベクトルは、互いに補い合う 2 つの見方で捉えられる。**点**として見れば、$\mathbf{v} = (3, 2)$ は**原点**から右に 3 単位、上に 2 単位進んだ位置を表す。この見方に立つと、データセット全体が 1 つの点群になり、T シャツとズボンを見分けるような分類課題は 2 つの点群を分離する幾何学的な問題になる。一方、**方向**（または変位）として見れば、同じベクトルは右へ 3 歩、上へ 2 歩移動せよという指示になる。変位はどこを始点にしてもよいため、Figure 33.1 に描かれた平行な矢印はすべて同じベクトルを表している。

方向という見方をとれば、ベクトルの加算は直ちに理解できる。1 本目の矢印をたどり、その先端から 2 本目の矢印をたどればよい（Figure 33.2）。

減算も同様である。$\mathbf{u} = \mathbf{v} + (\mathbf{u} - \mathbf{v})$ が成り立つため、差 $\mathbf{u} - \mathbf{v}$ は点 $\mathbf{v}$ から点 $\mathbf{u}$ へと運ぶ変位の矢印にほかならない。

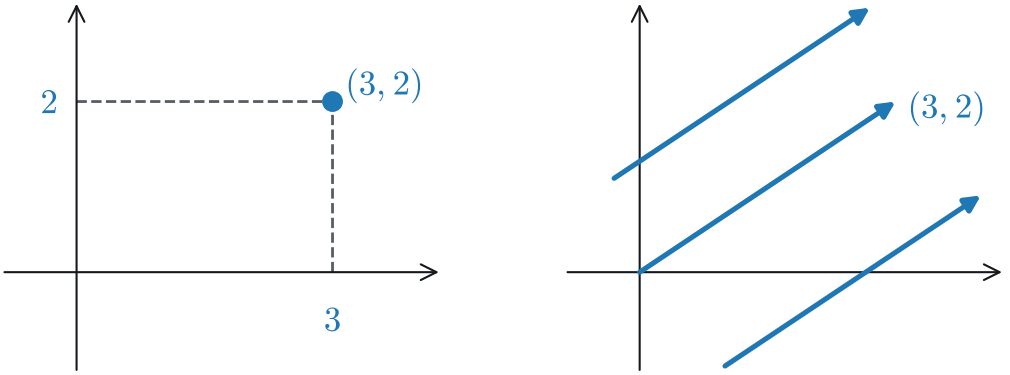


Figure 33.1: ベクトルの 2 通りの読み方：座標 (x, y) を持つ**点**として見るか（左）、どこからでも出発できる矢印としての**方向**と見るかであり、後者では図中のすべての矢印が同じベクトル $(3, 2)$ を表す（右）。

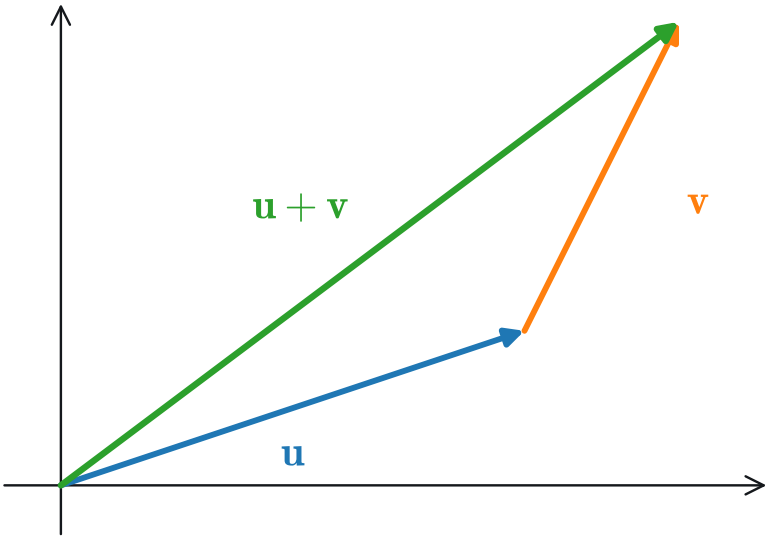


Figure 33.2: 先端と始点をつなぐベクトルの加算：1 本目の矢印をたどり、その終点から 2 本目の矢印をたどる。

本節のこれ以降の内容はすべて、1.3 節で導入した**ドット積**（dot product）というただ 1 つの演算から導かれる。2 つのベクトル $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ に対し、対応する成分同士を掛けてその和をとる演算である。

$$\mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{d} u_i v_i. \tag{33.1}$$

ドット積は $\mathbf{u}^\top \mathbf{v} = \mathbf{v}^\top \mathbf{u}$ と対称であるため、しばしば $\mathbf{u} \cdot \mathbf{v}$ と書く。2 本の矢印をイメージするときはドットを用い、行列の式の中に現れるときは転置を用いる。$\mathbf{u} = (1, 3)$ と $\mathbf{v} = (2, 1)$ に対しては $\mathbf{u} \cdot \mathbf{v} = 2 + 3 = 5$ であり、$\mathbf{u}$ の長さの二乗は $\mathbf{u} \cdot \mathbf{u} = 1 + 9 = 10$ となる。

33.1.1.2 直線への射影

ベクトル $\mathbf{v}$ のうち、別のベクトル $\mathbf{w}$ に沿った成分はどれだけあるだろうか。同じことだが、どのスカラー倍 $t\mathbf{w}$ が $\mathbf{v}$ に最も近くなるだろうか。これは線形代数における最も単純な当てはめ問題であり、単一の方向を使って $\mathbf{v}$ を近似し、最良の係数 t を求めることにあたる。

先ほどの組 $\mathbf{v} = (1, 3)$ と $\mathbf{w} = (2, 1)$ で、t にいくつか値を代入して試してみよう。$t = 1$ では、誤差 $\mathbf{v} - t\mathbf{w} = (-1, 2)$ の長さの二乗は 5 である。$t = 0.5$ では誤差は $(0, 2.5)$ で長さの二乗は 6.25 になり、$t = 1.5$ では $(-2, 1.5)$ でやはり長さの二乗は 6.25 になる。最小値は $t = 1$ で達成され、最近接点は $(2, 1)$ となる。このとき残る誤差を**残差**（residual）と呼ぶ。ここでは残差は $\mathbf{r} = (-1, 2)$ であり、$\mathbf{w}$ とのドット積を計算すると $\mathbf{r} \cdot \mathbf{w} = -2 + 2 = 0$ となることに注目しよう。最短の誤差ベクトルは直線と直角に交わるのである。さらに都合のよいことに、長さの二乗の間には直角三角形の関係が成り立つ。すなわち $\|\mathbf{v}\|^2 = 10$ に対し、最近接点の長さの二乗は 5、残差の長さの二乗も 5 であり、$10 = 5 + 5$ となっている。

この法則はすべての次元で成り立つ。

命題（直交射影）。 $\mathbf{w} \neq \mathbf{0}$ とする。$\mathbf{w}$ のスカラー倍のうち $\mathbf{v}$ に最も近いものは次で与えられる。

$$\operatorname{proj}_{\mathbf{w}} \mathbf{v} = \frac{\mathbf{v} \cdot \mathbf{w}}{\mathbf{w} \cdot \mathbf{w}} \, \mathbf{w}, \tag{33.2}$$

また、残差 $\mathbf{r} = \mathbf{v} - \operatorname{proj}_{\mathbf{w}} \mathbf{v}$ は $\mathbf{r} \cdot \mathbf{w} = 0$ を満たす。

証明。 二乗誤差を t の関数として最小化する。$\mathbf{v} - t\mathbf{w}$ 自身のドット積を展開すると、t の二次関数が得られる。

$$q(t) = \|\mathbf{v} - t\mathbf{w}\|^2 = \|\mathbf{w}\|^2 \, t^2 - 2 \, (\mathbf{v} \cdot \mathbf{w}) \, t + \|\mathbf{v}\|^2.$$

最高次の係数 $\|\mathbf{w}\|^2$ は正であるため、$q(t)$ は導関数 $q'(t) = 2\|\mathbf{w}\|^2 t - 2\,\mathbf{v} \cdot \mathbf{w}$ がゼロになる点、すなわち $t^\star = (\mathbf{v} \cdot \mathbf{w}) / (\mathbf{w} \cdot \mathbf{w})$ で唯一の最小値をとる。$t^\star$ を代入すれば Equation 33.2 が得られる。残差を確認すると、$\mathbf{r} \cdot \mathbf{w} = \mathbf{v} \cdot \mathbf{w} - t^\star \, \mathbf{w} \cdot \mathbf{w} = 0$ となる。■

先ほどの例では $q(t) = 5t^2 - 10t + 10$ となり、最小値 5 を $t^\star = 5/5 = 1$ でとるため、試行した値の結果が裏づけられる。$\operatorname{proj}_{\mathbf{w}} \mathbf{v}$ を、$\mathbf{v}$ の $\mathbf{w}$ への**射影**（projection）、あるいは**影**（shadow）と呼ぶ。射影は $\mathbf{v}$ のうち $\mathbf{w}$ に沿った成分を取り出し、残差はそれに直交する成分を取り出す。なぜ $\mathbf{w} \neq \mathbf{0}$ を仮定したかにも注意しよう。ゼロベクトルは方向を定めず、$\mathbf{w} = \mathbf{0}$ とおくとゼロ除算が生じるからである。

この直角のおかげで、長さの二乗がきれいに足し合わされる。2 つのベクトル $\mathbf{a}$ と $\mathbf{b}$ は、$\mathbf{a}\cdot\mathbf{b}=0$ のとき**直交**（orthogonal）するという。このとき、和の長さの二乗を展開すると**ピタゴラスの定理**（Pythagoras' theorem）が得られる。

$$\|\mathbf{a}+\mathbf{b}\|^2 = \|\mathbf{a}\|^2 + 2\,\mathbf{a}\cdot\mathbf{b} + \|\mathbf{b}\|^2 = \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2. \tag{33.3}$$

どの次元でも、直交する成分同士では交差項が消え、長さの二乗がそのまま足し合わされる。これを分解 $\mathbf{v}=\operatorname{proj}_{\mathbf{w}}\mathbf{v}+\mathbf{r}$ に適用すると $\|\mathbf{v}\|^2=\|\operatorname{proj}_{\mathbf{w}}\mathbf{v}\|^2+\|\mathbf{r}\|^2$ が得られ、先ほどの例ではこれが $10=5+5$ に対応する。

また、影が $\mathbf{w}$ と同じ向きか逆向きかを表す符号を含めた影の長さも必要になる。係数 $t^\star$ に $\|\mathbf{w}\|$ を掛けると、この**符号付き長さ**（signed length）が得られる。

$$\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{w}\|}. \tag{33.4}$$

先ほどの例では $5/\sqrt{5}=\sqrt{5}$ となり、$(2,1)$ の長さに一致する。この符号付き長さは、本節の終盤で点から超平面までの符号付き距離として再び登場する。Figure 33.3 にこの幾何学的な関係全体を示す。

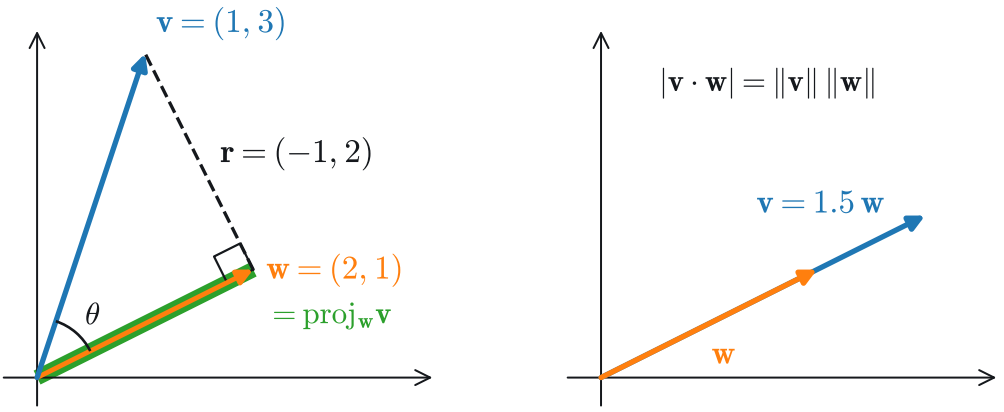


Figure 33.3: (a) $\mathbf{v}=(1,3)$ の $\mathbf{w}=(2,1)$ への射影（影）は $(2,1)$ であり、残差 $(-1,2)$ は $\mathbf{w}$ と直角に交わるため $\|\mathbf{v}\|^2=10=5+5$ が成り立つ。(b) コーシー・シュワルツの不等式の等号成立条件：$\mathbf{v}$ が $\mathbf{w}$ と共線であり、残差がゼロになる。

33.1.1.3 コーシー・シュワルツの不等式と角度

784 次元の空間に分度器はない。それでも「2 枚の画像のなす角」に意味を持たせられるだろうか。意外なことに、その答えははるかに単純な問いから得られる。ベクトルの影が元のベクトル自身より長くなることはありうるだろうか。Figure 33.3 のパネル (a) を見れば、答えは否である。影は $\mathbf{v}$ を斜辺とする直角三角形の 1 辺だからである。ピタゴラスの定理を使えば、この図をあらゆる次元で通用する証明へと変えられる。

命題（コーシー・シュワルツの不等式）。 すべてのベクトル $\mathbf{v},\mathbf{w}$ に対して、次の不等式が成り立つ。

$$|\mathbf{v}\cdot\mathbf{w}| \le \|\mathbf{v}\|\,\|\mathbf{w}\|, \tag{33.5}$$

等号は $\mathbf{v}$ と $\mathbf{w}$ が共線（一方が他方のスカラー倍）のとき、かつそのときに限り成り立つ。

証明。 $\mathbf{w}=\mathbf{0}$ のときは両辺ともゼロになり、2 つのベクトルは共線（collinear）である。そうでないとき、$\mathbf{v}$ を $\mathbf{w}$ への射影とそれに直交する残差 $\mathbf{r}$ に分解する。ピタゴラスの定理 Equation 33.3 より、次が成り立つ。

$$\|\mathbf{v}\|^2 = \|\operatorname{proj}_{\mathbf{w}}\mathbf{v}\|^2 + \|\mathbf{r}\|^2 \ge \|\operatorname{proj}_{\mathbf{w}}\mathbf{v}\|^2 = \frac{(\mathbf{v}\cdot\mathbf{w})^2}{\|\mathbf{w}\|^2}.$$

両辺に $\|\mathbf{w}\|^2$ を掛けて平方根をとれば、求める不等式が得られる。等号が成り立つのは $\mathbf{r}=\mathbf{0}$、すなわち $\mathbf{v}=t^{\star}\mathbf{w}$ のとき、かつそのときに限る。逆に $\mathbf{v}=t\mathbf{w}$ ならば、$|\mathbf{v}\cdot\mathbf{w}| = |t|\,\|\mathbf{w}\|^2 = \|\mathbf{v}\|\,\|\mathbf{w}\|$ となる。$\blacksquare$

平たく言えば、ベクトルの影が元のベクトル自身より長くなることは決してなく、長さが一致するのは残差に何も残らない場合だけである。1.3 節ではコーシー・シュワルツの不等式を代数的に証明したが、ここではパネル (a)（Figure 33.3）から直ちに導かれ、パネル (b) が等号成立の場合を示している。

これで任意の次元において角度を定義できる。平面上では、余弦定理により $\mathbf{v}\cdot\mathbf{w} = \|\mathbf{v}\|\|\mathbf{w}\|\cos\theta$ が成り立つ（1.3 節）。$\mathbb{R}^d$ には実際に測定できる三角形がないため、この恒等式を定義へと転用する。ゼロでないベクトル $\mathbf{v},\mathbf{w}\in\mathbb{R}^d$ に対し、そのなす**角度**を次で定義する。

$$\theta = \arccos\!\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|}\right) \in [0,\pi]. \tag{33.6}$$

arccos は $[-1,1]$ 上でのみ定義されるため、この式が意味を持つのは比がその区間内に収まる場合に限られるが、コーシー・シュワルツの不等式によってそれは常に保証される。したがって角度はすべての次元で矛盾なく定義され、平面におけるドット積の公式が直ちに任意の次元で成り立つ。

$$\mathbf{v}\cdot\mathbf{w} = \|\mathbf{v}\|\|\mathbf{w}\|\cos\theta. \tag{33.7}$$

コーシー・シュワルツの不等式における 2 つの等号成立条件は、$\theta=0$（同じ向きを指すベクトル）と $\theta=\pi$（反対向きを指すベクトル）に対応する。

ここで自然な疑問が生じる。この代数的な定義は、分度器で実際に測った角度と一致するのだろうか。結論から言えば一致する。任意の 2 つのベクトルは 1 つの平面内（共線であれば 1 つの直線上）に収まり、その平面内では通常の幾何学が成り立つ。射影を使えば、その平面に直交座標を入れられる。$\mathbf{q}_1 = \mathbf{w}/\|\mathbf{w}\|$ とおき、$\mathbf{r}\neq\mathbf{0}$ であれば $\mathbf{q}_2 = \mathbf{r}/\|\mathbf{r}\|$ とおく。これら 2 つのベクトルは単位長を持ち互いに直交するため、それらの線形結合のドット積は $\mathbb{R}^2$ におけるドット積とまったく同じように振る舞う。この座標系では $\mathbf{w} = (\|\mathbf{w}\|, 0)$、$\mathbf{v} = (\mathbf{v}\cdot\mathbf{w}/\|\mathbf{w}\|,\ \|\mathbf{r}\|)$ となる。平面上のこれら 2 本の矢印のなす角を分度器で測ると、そのコサインは $(\mathbf{v}\cdot\mathbf{w}/\|\mathbf{w}\|)/\|\mathbf{v}\|$ となり、Equation 33.6 と一致する。先ほどの例 $\mathbf{v}=(1,3)$ と $\mathbf{w}=(2,1)$ では $\cos\theta = 5/(\sqrt{10}\sqrt{5}) = 1/\sqrt{2}$ となり、$\theta = 45^{\circ}$ が得られる。

ゼロでないベクトルに対して、直交とは $\theta=\pi/2$ を意味する。直交性を $\mathbf{v}\cdot\mathbf{w}=0$ で定義しておくと、ゼロベクトルも含められるというちょっとした利点がある。ゼロベクトルに対して角度は定義されないが、すべてのベクトルと直交することになる。

Equation 33.6 をコードにするには 2 行で足りるが、実用上注目すべき安全策が 1 つある。先ほどの例と、意図的に扱いを難しくした共線な組 $\mathbf{v} = (10^8 + 1, 3, 7)$ と $3\mathbf{v}$（真のコサインは 1、真の角度は 0）で試してみよう。共線な組については、生のコサインから 1 を引いた値と、クリップ前の arccos も出力する。

```
def angle(v, w):
    cos = v @ w / (np.linalg.norm(v) * np.linalg.norm(w))
    return np.arccos(np.clip(cos, -1, 1))  # roundoff can push |cos| past 1

v = np.array([1e8 + 1, 3, 7])
cos = v @ (3 * v) / (np.linalg.norm(v) * np.linalg.norm(3 * v))
with np.errstate(invalid='ignore'):        # silence the warning for nan
    print(angle(np.array([1., 3]), np.array([2., 1])), cos - 1,
          np.arccos(cos), angle(v, 3 * v))
```

```
0.7853981633974484 2.220446049250313e-16 nan 0.0
```

最初の数値は $0.785 = \pi/4$ であり、先ほどの例の角度 $45°$ と一致する。残りの 3 つの値は、なぜ `np.clip` が不可欠かを示している。厳密な演算では、コーシー・シュワルツの不等式によって比は $[-1, 1]$ の内部に保たれる。しかし浮動小数点演算では、丸めによって最終桁の 1 単位分だけ境界の外へはみ出すことがある。ここでは共線な組に対して $1 + 2.2 \times 10^{-16}$ が生じ、`np.arccos` が `nan` を返している。$[-1, 1]$ にクリップすることで、正しい答えである 0 が復元される。

コーシー・シュワルツの不等式からは、和の長さが長さの和を決して超えないこともわかる。$\|\mathbf{v}+\mathbf{w}\|^2$ を展開し、交差項 $2\,\mathbf{v}\cdot\mathbf{w}$ を Equation 33.5 によって $2\,\|\mathbf{v}\|\|\mathbf{w}\|$ で上から抑えると、1.3 節と同様に三**角不等式**（triangle inequality）$\|\mathbf{v}+\mathbf{w}\| \le \|\mathbf{v}\| + \|\mathbf{w}\|$ が得られる。

33.1.2 部分空間と最小二乗法

33.1.2.1 張る空間、基底、部分空間

ある量を時刻 $t = 0, 1, 2$ で 3 回測定し、データに最もよく当てはまる直線 $C + Dt$ を求めたいとしよう。測定値を並べてベクトル $\mathbf{y} \in \mathbb{R}^3$ とし、時刻を $\mathbf{t} = (0, 1, 2)$ にまとめ、$\mathbf{1} = (1, 1, 1)$ とおく。3 つの時刻で直線の値を評価すると、ベクトル $C\mathbf{1} + D\mathbf{t}$ が得られる。したがって最良の直線を見つけるとは、$C\mathbf{1} + D\mathbf{t}$ の形をしたベクトルのうち $\mathbf{y}$ に最も近いものを見つけることにほかならない。C と D がすべての実数を動くとき、これらの候補ベクトルは $\mathbb{R}^3$ 内で原点を通る平面を描き出す。直線への射影では単一の方向を扱ったが、平面やその高次元版を扱うには、基本的な用語をいくつか用意する必要がある。張る空間、基底、部分空間にすでに慣れている読者は、直線の当てはめを完了させる次の項へそのまま進んでかまわない。

重み付き和 $a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$ を、$\mathbf{v}_1, \ldots, \mathbf{v}_k$ の**線形結合**（linear combination）と呼ぶ。そのような線形結合全体の集合が、それらの**張る空間**（span）である。

$$\operatorname{span}(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \{a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k : a_1, \ldots, a_k \in \mathbb{R}\}.$$

$\mathbb{R}^2$ では（Figure 33.4）、1 つのゼロでないベクトルが張る空間はその方向に沿って原点を通る直線であり、原点を通る同一の直線上にない 2 つのベクトルが張る空間は平面全体になる。先ほどの直線当てはめの問題では、候補となる当てはめ全体の集合は $\operatorname{span}(\mathbf{1}, \mathbf{t})$ である。

張る空間は、常に加算とスカラー倍について閉じていることに注目しよう。2 つの線形結合を足しても、あるいは 1 つの線形結合を実数倍しても、やはり線形結合になる。これら 2 つの閉包性を持つ空でないベクトルの集合を**部分空間**（subspace）と呼ぶ。すべての部分空間はゼロベクト

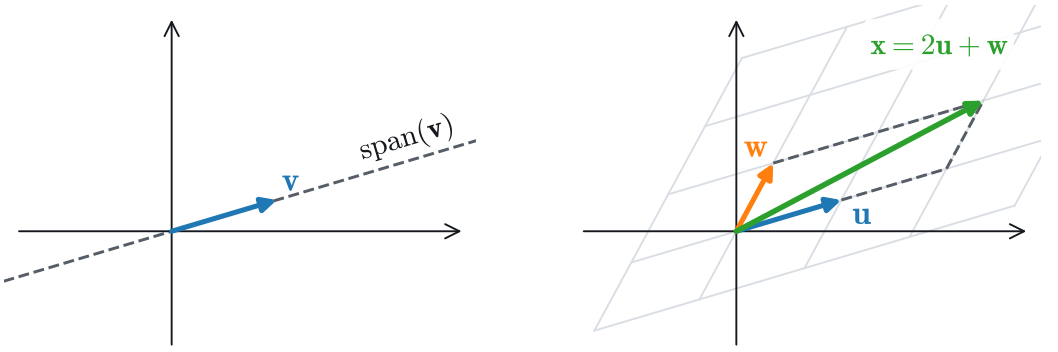


Figure 33.4: 左：1 つのゼロでないベクトル $\mathbf{v}$ が張る空間はそのスカラー倍全体の集合であり、原点を通る直線、すなわち 1 次元の部分空間をなす。右：原点を通る同一の直線上にない 2 つのベクトルは平面全体を張る。破線の平行四辺形は $\mathbf{x}$ を $2\mathbf{u} + \mathbf{w}$ と分解しており、$\mathbf{u}$ と $\mathbf{w}$ が線形独立であるため座標は一意に定まる。

ル $\mathbf{0}$ を含まなければならない（任意の要素に 0 を掛ければよい）。つまり、原点を通らない直線や平面は部分空間ではない。$\mathbb{R}^2$ の部分空間は、原点のみからなる集合、原点を通る直線、そして $\mathbb{R}^2$ 自身に限られる。$\mathbb{R}^3$ では、これに原点を通る平面が加わる。

部分空間を張るベクトルの組には、冗長なものが含まれうる。あるベクトルが他のベクトルの張る空間にすでに含まれているなら、そのベクトルを取り除いても張る空間は縮まない。そのような冗長性が一切ないとき、ベクトルの組は**線形独立**（linearly independent）であるという。これは、線形結合 $a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$ が $\mathbf{0}$ に等しくなるのが、すべての $a_i = 0$ とした自明な場合に限られることを意味する。先ほどのベクトル $\mathbf{1}$ と $\mathbf{t}$ は線形独立である。実際、$a_1\mathbf{1} + a_2\mathbf{t} = (a_1,\, a_1 + a_2,\, a_1 + 2a_2)$ がゼロになるのは $a_1 = a_2 = 0$ のときに限られる。

部分空間の**基底**（basis）とは、その部分空間を張る線形独立なベクトルの組である。おなじみのベクトル $\mathbf{e}_1 = (1, 0)$ と $\mathbf{e}_2 = (0, 1)$ は $\mathbb{R}^2$ の**標準基底**（standard basis）をなすが、Figure 33.4 の傾いた 2 本の組も同様に立派な基底である。線形代数の基本定理（ここでは証明なしに用いる）によれば、与えられた部分空間のどの基底も同じ個数のベクトルからなり、その共通の個数を部分空間の**次元**と呼ぶ。これにより、「d 次元空間」における d に明確な意味が与えられる。すなわち、$\mathbb{R}^d$ の次元が d であるのは、$\mathbf{e}_1, \ldots, \mathbf{e}_d$ が基底をなすからである。同様に、先ほどの直線当てはめの候補全体は $\mathbb{R}^3$ の 2 次元部分空間をなす。

部分空間内のすべての点に一意な座標を与えるのが線形独立性である。$\mathbf{v}_1, \ldots, \mathbf{v}_k$ が部分空間 S の基底であれば、すべての $\mathbf{x} \in S$ は $\mathbf{x} = a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$ とただ 1 通りに表せる。基底は S を張るため、少なくとも 1 つの表示が存在する。もし係数 a_i と b_i を持つ 2 通りの表示が存在したとすれば、両者の差をとることで $\sum_i (a_i - b_i)\,\mathbf{v}_i = \mathbf{0}$ が得られ、線形独立性により $a_i = b_i$ がすべての i に対して成り立つ。

いったん基底を固定すれば、k 次元の部分空間は $\mathbb{R}^k$ とまったく同じように振る舞い、各点は係数の組 $(a_1, \ldots, a_k)$ によって一意に指定される。直線の当てはめにおいては、その座標とは単に切片 C と傾き D のことである。

33.1.2.2 部分空間への射影

具体的な測定値 $\mathbf{y} = (6, 0, 0)$ を用いて直線の当てはめに戻ろう。$C\mathbf{1} + D\mathbf{t}$ という形の平面 $\operatorname{span}(\mathbf{1}, \mathbf{t})$ 上の点のうち、$\mathbf{y}$ に最も近いものを求めたい。1 本の直線へ射影したときは、最近接点の残差はその直線と直交していた。平面への射影でも、残差は平面全体と直交し、したがって $\mathbf{1}$

と $\mathbf{t}$ の両方と直交するはずである（この事実はすぐ後で証明する）。両方のドット積をゼロとおくと、2 変数に関する 2 本の線形方程式が得られる。

$$\mathbf{1}\cdot(\mathbf{y}-C\mathbf{1}-D\mathbf{t}) = 6-3C-3D = 0,$$
$$\mathbf{t}\cdot(\mathbf{y}-C\mathbf{1}-D\mathbf{t}) = 0-3C-5D = 0.$$

第 2 式から第 1 式を引くと $2D=-6$ となるので、$D=-3$、$C=5$ が得られる。最もよく当てはまる直線は $5-3t$ である。時刻 $t=0,1,2$ における予測値は $(5,2,-1)$ となり、残差は $\mathbf{r}=\mathbf{y}-(5,2,-1)=(1,-2,1)$ となる。直交性の確認も $\mathbf{r}\cdot\mathbf{1}=1-2+1=0$ および $\mathbf{r}\cdot\mathbf{t}=0-2+2=0$ と両方成り立っている。ピタゴラスの定理により、ここでも $\mathbf{y}$ の長さの二乗が $36=30+6$ ときれいに分解される。比較として、$\mathbf{y}$ を $\mathbf{1}$ のみに射影すると定数による当てはめ $\mathbf{y}\cdot\mathbf{1}/\mathbf{1}\cdot\mathbf{1}=2$（標本平均）が得られ、その残差 $(4,-2,-2)$ の長さの二乗は 24 になる。$\mathbf{t}$ に沿った傾きを許すことで、二乗誤差は 24 から 6 へと減少する。Figure 33.5 に $\mathbb{R}^3$ におけるこの射影を示す。

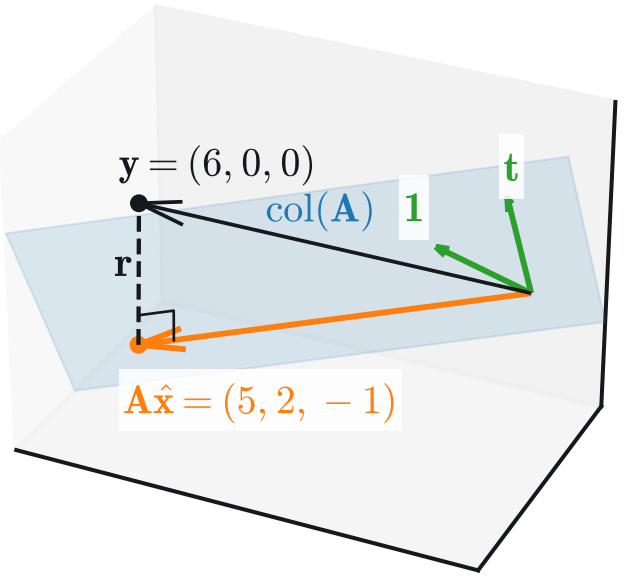


Figure 33.5: 射影としての最小二乗法：測定値 $\mathbf{y}=(6,0,0)$ は候補直線 $C\mathbf{1}+D\mathbf{t}$ のなす平面の外にあり、平面内の最近接点は $(5,2,-1)$（直線 $5-3t$）で、残差 $(1,-2,1)$ は $\mathbf{1}$ と $\mathbf{t}$ の両方に垂直である。

一般の結果を見通しよく述べ、証明するには、互いに直交し単位長を持つベクトルからなる基底を選ぶと都合がよい。ベクトル組 $\mathbf{q}_1,\dots,\mathbf{q}_k$ がなす部分空間 $S\subseteq\mathbb{R}^n$ の基底は、すべてのベクトルが $\|\mathbf{q}_i\|=1$ を満たし、異なるベクトル同士が $\mathbf{q}_i\cdot\mathbf{q}_j=0$ を満たすとき**正規直交**（orthonormal）であるという。有限個のベクトルからなる任意の生成系からは、このような正規直交基底を作ることができる。33.1.2.3 節では、直線への射影だけを使って実際に構成する。これらの基底ベクトルを $n\times k$ 行列 $\mathbf{Q}$ の列として並べると、正規直交性は $\mathbf{Q}^{\top}\mathbf{Q}=\mathbf{I}_k$ という簡潔な形で表せる。なぜなら、成分 (i,j)（$\mathbf{Q}^{\top}\mathbf{Q}$ の各成分）は $\mathbf{q}_i\cdot\mathbf{q}_j$ に等しいからである。本章全体を通じて、文字 $\mathbf{Q}$

は正規直交列を持つ行列を表すために用いる。次の命題は、本章の残りの部分で最も頻繁に用いられる中核的な結果である。

命題（部分空間への射影）。 $\mathbf{Q}$ が S を張る正規直交列を持つとし、$\mathbf{P} = \mathbf{Q}\mathbf{Q}^{\top}$ とおく。このとき、すべての $\mathbf{x} \in \mathbb{R}^n$ に対して次が成り立つ。 (i) $\mathbf{P}\mathbf{x} \in S$ であり、残差 $\mathbf{x} - \mathbf{P}\mathbf{x}$ は S のすべてのベクトルと直交する。 (ii) $\mathbf{P}\mathbf{x}$ は S 内で $\mathbf{x}$ に最も近い唯一の点である。 (iii) $\mathbf{P}^2 = \mathbf{P}$ かつ $\mathbf{P}^{\top} = \mathbf{P}$ である。

証明。 (i) について、$\mathbf{P}\mathbf{x} = \mathbf{Q}(\mathbf{Q}^{\top}\mathbf{x})$ は $\mathbf{Q}$ の列の線形結合であるため S に属する。残差に左から $\mathbf{Q}^{\top}$ を掛けると次が得られる。

$$\mathbf{Q}^{\top}(\mathbf{x} - \mathbf{P}\mathbf{x}) = \mathbf{Q}^{\top}\mathbf{x} - (\mathbf{Q}^{\top}\mathbf{Q})\,\mathbf{Q}^{\top}\mathbf{x} = \mathbf{0},$$

したがって残差はすべての列 $\mathbf{q}_i$ と直交し、ひいてはそれらの任意の線形結合とも直交する。(ii) について、任意の $\mathbf{s} \in S$ をとり、$\mathbf{x} - \mathbf{s} = (\mathbf{x} - \mathbf{P}\mathbf{x}) + (\mathbf{P}\mathbf{x} - \mathbf{s})$ と書く。$\mathbf{P}\mathbf{x} - \mathbf{s}$ は S に属する一方、$\mathbf{x} - \mathbf{P}\mathbf{x}$ は S と直交するため、ピタゴラスの定理 Equation 33.3 より次が成り立つ。

$$\|\mathbf{x} - \mathbf{s}\|^2 = \|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2 + \|\mathbf{P}\mathbf{x} - \mathbf{s}\|^2 \geq \|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2,$$

ここで等号が成り立つのは $\mathbf{s} = \mathbf{P}\mathbf{x}$ のときに限られる。(iii) については、$\mathbf{P}^2 = \mathbf{Q}(\mathbf{Q}^{\top}\mathbf{Q})\mathbf{Q}^{\top} = \mathbf{Q}\mathbf{Q}^{\top} = \mathbf{P}$、および $\mathbf{P}^{\top} = (\mathbf{Q}^{\top})^{\top}\mathbf{Q}^{\top} = \mathbf{P}$ である。$\blacksquare$

逆に、(i) の直交条件は最近接点を一意に特徴づける。ある $\mathbf{s} \in S$ に対して残差 $\mathbf{x} - \mathbf{s}$ が S と直交するならば、差 $\mathbf{P}\mathbf{x} - \mathbf{s} = (\mathbf{x} - \mathbf{s}) - (\mathbf{x} - \mathbf{P}\mathbf{x})$ は S に属すると同時に S と直交するため、自分自身と直交することになり、$\mathbf{s} = \mathbf{P}\mathbf{x}$ が導かれる。直線の当てはめで 2 本の直交方程式を解けば必ず最近接点が得られたのは、まさにこのためである。

性質 (iii) の 2 つの式には、それぞれ明確な幾何学的意味がある。恒等式 $\mathbf{P}^2 = \mathbf{P}$ は、1 回目の射影ですでに S 内に到達しているため、2 回目の射影を施しても何も変わらないことを表す。対称性 $\mathbf{P}^{\top} = \mathbf{P}$ こそが、この射影を**直交**射影たらしめている。対称性なしに $\mathbf{P}^2 = \mathbf{P}$ を満たす行列は**斜**射影行列（oblique projector）であり、ベクトルを斜めの方向に沿って列空間へ写すため、一般には最近接点から外れる（3）。1 次元の場合（$k = 1$）には、単一の単位列ベクトル $\mathbf{q} = \mathbf{w}/\|\mathbf{w}\|$ をとることで、$\mathbf{P} = \mathbf{q}\mathbf{q}^{\top}$ は先ほどの直線への射影 Equation 33.2 にそのまま帰着する。

先ほどの直線当てはめは、線形回帰の中核をなす最適化問題である**最小二乗法**（least squares）の 1 例である。行がデータ例、列が特徴量であるデータ行列を $\mathbf{A}$ とし、目的変数のベクトルを $\mathbf{y}$ とする。最小二乗法では、二乗誤差を最小化する係数ベクトル $\mathbf{x}$ を求める。

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2. \tag{33.8}$$

$\mathbf{x}$ がすべての係数ベクトルを動くとき、$\mathbf{A}\mathbf{x}$ は $\mathbf{A}$ の列が張る空間、すなわち**列空間**（column space）$\mathrm{col}(\mathbf{A})$ を描き出す。したがって最小二乗法とは、$\mathrm{col}(\mathbf{A})$ 内で $\mathbf{y}$ に最も近い点を求める問題である。先ほどの射影の命題により、その点は残差 $\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}$ が $\mathbf{A}$ のすべての列と直交する点として定まる。この直交条件を $\mathbf{A}^{\top}(\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}) = \mathbf{0}$ と書けば、次の**正規方程式**（normal equations）が得られる。

$$\mathbf{A}^{\top}\mathbf{A}\,\hat{\mathbf{x}} = \mathbf{A}^{\top}\mathbf{y}. \tag{33.9}$$

これは手計算で解いた 2×2 連立方程式の行列形式にほかならない。直線の当てはめでは、$\mathbf{A} = [\mathbf{1}\ \mathbf{t}]$ に対して $\mathbf{A}^{\top}\mathbf{A} = \left(\begin{smallmatrix} 3 & 3 \\ 3 & 5 \end{smallmatrix}\right)$ および $\mathbf{A}^{\top}\mathbf{y} = (6, 0)$ となる。$\mathbf{A}$ の列が線形独立であれば、常

に $\mathbf{A}^{\top}\mathbf{A}$ は可逆であり、最適な係数 $\hat{\mathbf{x}}$ は一意に定まる。列が線形従属である場合でも、当てはめベクトル $\mathbf{A}\hat{\mathbf{x}}$ は依然として $\mathrm{col}(\mathbf{A})$ 内の最近接点であるため一意のままだが、それを与える係数ベクトル $\hat{\mathbf{x}}$ は複数存在しうる。

したがってコード上では、当てはめ直線 $5 - 3t$ を計算する方法が 2 通りある。正規方程式を直接解くか、$\mathbf{P} = \mathbf{Q}\mathbf{Q}^{\top}$ を使って射影するかである。$\mathbf{Q}$ を得るには `np.linalg.qr` を呼び出す。この関数は、線形独立な列を持つ行列の列空間に対する正規直交基底を構築する（次の項ではこれを手計算で構成する）。

```
A = np.array([[1., 0], [1, 1], [1, 2]])     # columns 1 and t
y = np.array([6., 0, 0])
x_hat = np.linalg.solve(A.T @ A, A.T @ y)   # normal equations
Q, R = np.linalg.qr(A)                      # orthonormal basis of col(A)
P = Q @ Q.T
r = y - P @ y
print(x_hat.round(6), (P @ y).round(6), r.round(6))
print(np.linalg.norm(P @ P - P), np.linalg.norm(Q.T @ r))
```

```
[ 5. -3.] [ 5.  2. -1.] [ 1. -2.  1.]
1.962615573354719e-16 0.0
```

どちらの方法でも、手計算と同じく係数 $C = 5$ と $D = -3$、射影された値 $(5, 2, -1)$、および残差 $(1, -2, 1)$ が復元される。2 行目にある確認用の 2 つのノルムは浮動小数点の丸め誤差の範囲でゼロになり、$\mathbf{P}^2 = \mathbf{P}$ と $\mathbf{Q}^{\top}\mathbf{r} = \mathbf{0}$ が裏づけられる。

このように小さく性質のよい問題では、2 つの手法は完全に一致する。しかし $\mathbf{A}$ の列がほぼ線形従属である場合、$\mathbf{A}^{\top}\mathbf{A}$ を作ると問題の条件数（丸め誤差に対する感度）が二乗されてしまう（このことは 33.5.3.2 節で示す）。そのため、実用的な最小二乗ソルバは正規方程式を避け、代わりに列空間の正規直交基底を用いて計算する。

33.1.2.3 グラム・シュミット法と QR 分解

実際にはどのようにして正規直交基底を構築するのだろうか。答えは射影の繰り返しである。2 つの線形独立なベクトルが与えられたとき、1 つ目のベクトルの方向はそのまま保ち、2 つ目のベクトルから 1 つ目の方向に沿った影を引き去る。射影の命題により、残った成分は 1 つ目のベクトルと直交する。

これを $\mathbf{a}_1 = (3, 4)$ と $\mathbf{a}_2 = (1, 0)$ に対して実行してみよう。$\mathbf{a}_1$ を正規化すると $\mathbf{q}_1 = \mathbf{a}_1 / 5 = (0.6, 0.8)$ となる。$\mathbf{a}_2$ の $\mathbf{q}_1$ への影は $(\mathbf{q}_1 \cdot \mathbf{a}_2)\,\mathbf{q}_1 = 0.6\,(0.6, 0.8) = (0.36, 0.48)$ である。これを引くと $(1, 0) - (0.36, 0.48) = (0.64, -0.48)$ が残り、これは長さ 0.8 のベクトルで、確かに $\mathbf{q}_1$ と直交している（$0.384 - 0.384 = 0$）。この残りを正規化すると $\mathbf{q}_2 = (0.8, -0.6)$ が得られる。以上の手順を逆にたどれば、元の各ベクトルを新しい正規直交基底で表すことができる。

$$\mathbf{a}_1 = 5\,\mathbf{q}_1, \qquad \mathbf{a}_2 = 0.6\,\mathbf{q}_1 + 0.8\,\mathbf{q}_2 .$$

行列記法で書けば、これは $\mathbf{A} = \mathbf{Q}\mathbf{R}$ である。ここで $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2]$、$\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2]$ であり、$\mathbf{R} = \left(\begin{smallmatrix} 5 & 0.6 \\ 0 & 0.8 \end{smallmatrix}\right)$ はドット積 $\mathbf{q}_i \cdot \mathbf{a}_j$ を集めた行列である。なぜ $\mathbf{R}$ が上三角行列になるかにも注目しよう。構成法から、各 $\mathbf{a}_j$ は $\mathbf{q}_1, \dots, \mathbf{q}_j$ のみに依存するからである。

この考え方を任意の線形独立なベクトル組 $\mathbf{a}_1, \dots, \mathbf{a}_k$ へ拡張したものが**グラム・シュミットの直交化法**（Gram–Schmidt process）である。ベクトルを順に処理し、各 $\mathbf{a}_j$ から、すでに構成した単位ベクトル $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$ への射影 Equation 33.2 を引き去り、残りを正規化して $\mathbf{q}_j$ を得る。その

結果として、$\mathrm{span}(\mathbf{a}_1, \ldots, \mathbf{a}_k)$ の正規直交基底が得られると同時に、**QR 分解**（QR factorization）$\mathbf{A} = \mathbf{QR}$ も得られる。ここで $\mathbf{Q}$ は正規直交列を持ち、$\mathbf{R}$ は対角成分が正の上三角行列である。この分解には線形独立性が不可欠である。それ以前のベクトルが張る空間にすでに含まれているベクトルからは残りがゼロになり、正規化できないからである。張る空間の正規直交基底だけが必要なら、残りがゼロになったベクトルを単に飛ばして先へ進めばよい。したがって有限個のベクトルからなる任意の生成系から正規直交基底が得られ、先ほど用いた存在の主張が正当化される。

数値計算ライブラリは鏡映に基づく別のアルゴリズムで QR 分解を計算しており、浮動小数点演算においてはるかに良好に振る舞う。その出力は符号の違いを除いてグラム・シュミット法と一致する。$\mathbf{Q}$ のある列とそれに対応する $\mathbf{R}$ の行の符号を同時に反転させても、積 $\mathbf{QR}$ は変わらないからである。$\mathbf{A}$ の列が線形独立であれば、不定性はこの符号反転のみに限られる。2 ステップのグラム・シュミット計算を `np.linalg.qr` と比較し、符号を除いて同じ列が得られることを確かめてみよう。

```
a1, a2 = np.array([3., 4]), np.array([1., 0])
q1 = a1 / np.linalg.norm(a1)
v = a2 - (q1 @ a2) * q1                    # remove the shadow on q1
q2 = v / np.linalg.norm(v)
Q_qr, R_qr = np.linalg.qr(np.stack([a1, a2], axis=1))
print(np.stack([q1, q2], axis=1).round(6))
print(Q_qr.round(6))
print(R_qr.round(6))
```

```
[[ 0.6  0.8]
 [ 0.8 -0.6]]
[[-0.6 -0.8]
 [-0.8  0.6]]
[[-5.  -0.6]
 [ 0.  -0.8]]
```

グラム・シュミット法は $(0.6, 0.8)$ と $(0.8, -0.6)$ を生成する一方、`np.linalg.qr` は両方の列の符号を反転させたものと、対角成分が -5 と -0.8 である $\mathbf{R}$ を返す。これは同じ基底であり、符号を除けば手計算で求めたドット積 5、0.6、0.8 とも一致している。浮動小数点演算では、引き算を実行する順序も重要になる。ほぼ線形従属な列に対しては、**古典的**（classical）グラム・シュミット法と呼ばれる上記の定式化は徐々に直交性を失っていく（5）。

33.1.3 高次元における類似度

ここまでの結果には、すべて $\mathbb{R}^2$ や $\mathbb{R}^3$ の図が伴っていた。ここからは図を描くことのできない空間へと移り、実際の 2 枚の画像がどれほど似ているか、そして無関係なベクトル同士が純粋な偶然によってどの程度の類似度を持つかを考える。Fashion-MNIST データセット（3.2 節）を `d2l.FashionMNIST` 経由で NumPy 配列として読み込み、各画像を 784 ピクセルの行ベクトルへ平坦化し、$[0, 1]$ にスケーリングする。

```
data = d2l.FashionMNIST()
(X_train, y_train), (X_test, y_test) = data.train, data.val
X_train = X_train.reshape(-1, 784) / 255
X_test = X_test.reshape(-1, 784) / 255
X_train.shape, X_test.shape
```

```
((60000, 784), (10000, 784))
```

これら 70,000 枚の画像は、いまやそれぞれが $\mathbb{R}^{784}$ 内の 1 点である。

33.1.3.1 コサイン類似度

Equation 33.6 の内部にある比は、1.3 節で導入した**コサイン類似度**（cosine similarity）である。

$$\cos\theta = \frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\,\|\mathbf{w}\|} \;\in\; [-1,1]. \tag{33.10}$$

コサイン類似度は長さを無視して方向だけを比較する。ある画像と、その明るさを 10% に落としたコピーはユークリッド距離では大きく離れているが、コサイン類似度は依然として 1 である。

このスケール不変性により、コサイン類似度は埋め込みに対する最近傍検索や、SimCLR (Chen et al. 2020) のような対照学習（contrastive learning）の目的関数における標準的な指標となっている。SimCLR では、同一画像の 2 つのビューから生成された埋め込みの方向を揃える。一方、元の Transformer のアテンション層は異なる選択をしており、$1/\sqrt{d}$ でスケーリングした**正規化なしの**ドット積を用いるため（その理由はまもなく導出する）、そこではベクトルの長さも情報を持つ。後発の多くのモデルでは、先にクエリとキーを正規化する（11.1.2.5 節）ことで、スコアを再びスケール付きコサインへと戻している。

33.1.3.2 ランダムな方向の概直交性

無関係な 2 枚の画像、あるいは無関係な 2 つの埋め込みを選んだとしよう。純粋な偶然によって、それらの間にどの程度のコサイン類似度が期待されるだろうか。比較の基準がなければ、観測されたコサインが意味のある大きさかどうかを判断できない。最も明快な基準は、空間のどの領域にも偏りを持たせず、単位球面上から一様ランダムに選んだ方向である。

命題（概直交性）。 単位ベクトル $\mathbf{u}\in\mathbb{R}^d$ を固定し、$\mathbf{v}$ を $\mathbb{R}^d$ の単位球面上の一様分布から抽出する。このとき $\cos\theta = \mathbf{u}\cdot\mathbf{v}$ は次を満たす。

$$\mathbb{E}[\cos\theta] = 0, \qquad \mathrm{Var}(\cos\theta) = \frac{1}{d}, \qquad \Pr\!\left(|\cos\theta| \ge \varepsilon\right) \le \frac{1}{d\,\varepsilon^2} \quad \text{for every } \varepsilon > 0.$$

証明。 球面上の一様分布は回転に対して不変であるため、$\mathbf{u} = \mathbf{e}_1$ となるように座標系を回転させてよく、このとき $\cos\theta = v_1$ となる。この分布は符号反転 $\mathbf{v}\mapsto -\mathbf{v}$ によっても変わらないため、$\mathbb{E}[v_1] = 0$ である。対称性により、すべての座標は同じ二次モーメント $\mathbb{E}[v_i^2]$ を持つ。$\sum_{i=1}^{d} v_i^2 = \|\mathbf{v}\|^2 = 1$ であるから、期待値をとると $d\,\mathbb{E}[v_1^2] = \mathbb{E}[\sum_{i=1}^{d} v_i^2] = 1$ となり、したがって $\mathrm{Var}(\cos\theta) = \mathbb{E}[v_1^2] = 1/d$ が得られる。裾の評価はチェビシェフの不等式 $\Pr(|X-\mathbb{E}[X]|\ge\varepsilon)\le \mathrm{Var}(X)/\varepsilon^2$（1.6 節）から直ちに従う。■

したがって、ランダムな 2 つの方向のなす角のコサインは典型的に $1/\sqrt{d}$ のオーダーになる。$d = 784$ では標準偏差が $1/28 \approx 0.036$ となり、画像空間内のランダムな 2 つの方向はほとんど常に直交から数度以内に収まる。チェビシェフの不等式は $|\cos\theta| \ge 0.1$ となる確率を $1/(784\cdot 0.01) \approx 0.13$ 以下と評価するが、真の確率は 0.01 未満である。チェビシェフの不等式が緩いのは分散しか使っていないためであり、より鋭い指数型の上界は 36.9 節で扱う。

低次元での様相はまったく異なる（Figure 33.6）。平面（$d = 2$）では、固定した方向とランダムな方向のなす角は $[0,\pi]$ 上で一様であるため、直交から 0.1 ラジアン以内に収まる確率と、共線から 0.1 ラジアン以内に収まる確率は等しい（それぞれ約 6%）。ところが**コサイン**は ± 1 の近くに集中する。$\cos\theta$ が $\theta = 0$ と $\theta = \pi$ の近くで平坦になるためであり、$|\cos\theta| > 0.9$ となる確率が約 0.29 であるのに対し、わずか 0.06 にすぎない（$|\cos\theta| < 0.1$ の場合）。$d = 3$ では

コサインは $[-1, 1]$ 上で一様になり、これは Archimedes にも知られていた古典的な結果である。$d = 4$ からは密度が 0 でピークを持つようになり、$d = 10$ では幅 $1/\sqrt{10} \approx 0.32$ の釣鐘型をなし、$d = 1000$ に至ると幅が約 0.03 の鋭いスパイクへと収縮する。

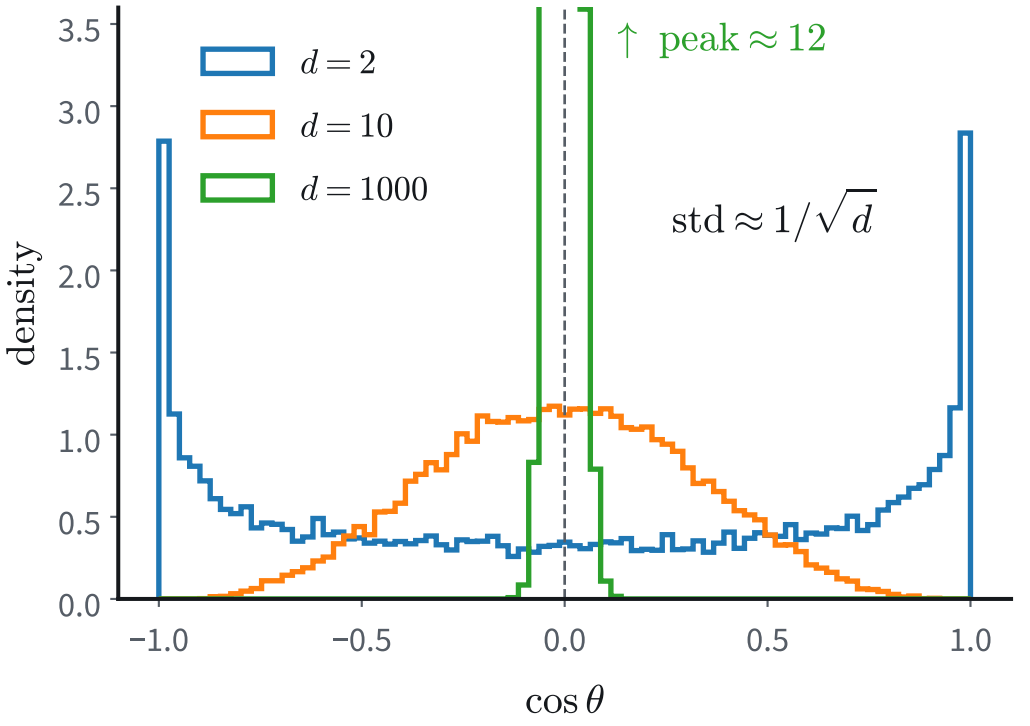


Figure 33.6: 次元 $d = 2$、10、1000 における独立なランダム単位ベクトル間のコサインのヒストグラム。$d = 2$ では、角度は一様であるにもかかわらずコサインが ± 1 付近に集中する。$d = 10$ では 0 の周りに釣鐘型をなし、$d = 1000$ では命題の通り幅 $\approx 1/\sqrt{d}$ の細いスパイクへと集中する。

$d = 784$ におけるこの等方的な基準をシミュレーションで確かめてみよう。標準ガウスベクトルの分布は回転不変であるため、それを正規化すると一様ランダムな単位ベクトルが得られる。同じコードセル内で、すぐ後のアテンションの議論で必要になる分散、すなわち $\mathbf{q} \cdot \mathbf{k}$ の分散（64 次元の独立な標準正規ベクトルの場合）もあわせて推定しておく。

```
rng = np.random.default_rng()
d = 784
U = rng.standard_normal((10000, d))
U /= np.linalg.norm(U, axis=1, keepdims=True)
V = rng.standard_normal((10000, d))
V /= np.linalg.norm(V, axis=1, keepdims=True)
q, k = rng.standard_normal((10000, 64)), rng.standard_normal((10000, 64))
np.std((U * V).sum(axis=1)), 1 / np.sqrt(d), np.var((q * k).sum(axis=1))
```

```
(np.float64(0.03539421752901573),
 np.float64(0.03571428571428571),
 np.float64(63.05852998792989))
```

経験標準偏差は $1/\sqrt{784} \approx 0.036$ と一致し、正規化なしのドット積の分散は次元である 64 に近い値となる。

しかし実データが等方的に見えることはめったにない。無関係な 2 枚の Fashion-MNIST 画像は、ほぼ直交するベクトルのように振る舞うだろうか。テスト画像からランダムなペアを 5,000 組サンプリングし、まず生のピクセルに対して、次に訓練画像の平均を引いた後でコサインを測ってみよう。

```
def cosines(A, B):
    """Cosine similarity between corresponding rows of A and B."""
    return (A * B).sum(axis=1) / (np.linalg.norm(A, axis=1)
                                  * np.linalg.norm(B, axis=1))

i, j = rng.integers(0, len(X_test), (2, 5000))
mean_image = X_train.mean(axis=0)
raw = cosines(X_test[i], X_test[j])
centered = cosines(X_test[i] - mean_image, X_test[j] - mean_image)
raw.mean(), raw.std(), centered.mean(), centered.std()
```

```
(np.float64(0.5922711564823272),
 np.float64(0.1780528067338411),
 np.float64(0.0011425036942642342),
 np.float64(0.34541199385170157))
```

結果は似ても似つかない。生の画像の平均コサインは約 0.6、標準偏差は約 0.18 であり、0 ± 0.036 とは程遠い。理由は単純で、ピクセル輝度は非負であるため、すべての画像が $\mathbb{R}^{784}$ の同じ正象限（positive orthant）に属しており、負のドット積が生じえないからである。平均画像を引くとコサインの中心は 0 に移るが、その標準偏差は約 0.34 と依然として等方的な値のほぼ 10 倍もある。先ほどの命題によれば、0.34 という広がりはわずか 8〜9 次元におけるランダムな方向に相当する（$1/0.34^2$ は 8 と 9 の間にある）。コサインから判断する限り、中心化された画像群は 784 次元ではなく 10 次元未満の空間に収まる点群のように振る舞っており、これは 33.5 節で学ぶ低ランク構造の片鱗である。実用上、あるコサインが大きいと言えるのは、**同じ**データセットからとった無関係なペアの広がりよりも際立っている場合に限られる。関連するペアをその背景分布の上へと押し上げることこそが、埋め込みモデルを訓練する役割である。

33.1.3.3 アテンションスコアのスケール

ドット積アテンション（10.2 節）は、クエリ $\mathbf{q} \in \mathbb{R}^d$ とキー $\mathbf{k}_1, \mathbf{k}_2, \ldots$ の適合度を $\mathbf{q} \cdot \mathbf{k}_j / \sqrt{d}$ によってスコア付けし、そのスコアをソフトマックスに通す (Vaswani et al. 2017)。簡単な分散の計算から、$\sqrt{d}$ の由来が説明できる。全 $2d$ 個の成分（$\mathbf{q}$ と $\mathbf{k}$ の成分）が互いに独立で、平均 0、分散 1 を持つと仮定しよう。このとき各積 $q_i k_i$ は平均 0、分散 $\mathbb{E}[q_i^2]\,\mathbb{E}[k_i^2] = 1$ を持ち、異なる項同士は $\mathbb{E}[(q_i k_i)(q_j k_j)] = \mathbb{E}[q_i]\,\mathbb{E}[k_i]\,\mathbb{E}[q_j]\,\mathbb{E}[k_j] = 0$（$i \neq j$）より無相関である。したがって d 個の項の分散はそのまま足し合わされる。

$$\operatorname{Var}(\mathbf{q} \cdot \mathbf{k}) = \sum_{i=1}^{d} \operatorname{Var}(q_i k_i) = d.$$

ゆえに生のドット積の標準偏差は $\sqrt{d}$ となってヘッド次元とともに増大するが、$\sqrt{d}$ で割れば単位分散に戻る。このスケーリングがないと、ソフトマックスは飽和してほぼワンホットに近い重みになってしまう。$d = 64$ の場合、生のスコアの標準偏差は 8 であり、8 だけ離れた 2 つのスコ

アにはソフトマックス重み 0.9997 と 0.0003 が割り当てられる。$\sqrt{d} = 8$ で割ることで、重みは健全な 0.73 と 0.27 に戻る。幾何学からもまったく同じスケーリングが導かれる。各ベクトルの長さは約 $\sqrt{d}$ である一方、無関係な方向のなす角のコサインは約 $1/\sqrt{d}$ であるため、そのドット積の典型的な大きさは $\sqrt{d} \cdot \sqrt{d} \cdot (1/\sqrt{d}) = \sqrt{d}$ となる。

もっとも、無関係なキーが多数ある場合、概直交性があるからといってそれらを無視できるわけではない。無関係な個々のコサインのスケールは $1/\sqrt{d}$ であっても、独立な n 個のキーの中での**最大の**コサインは主要項で $\sqrt{2 \ln n / d}$ のように増大する。大規模な検索インデックスでは、このような背景ノイズによる最悪の一致度が $1/\sqrt{d}$ の数倍に達するため、真に関連するキーははるかに高い水準を超えなければならず、訓練によってまさにそれが実現される（1 でこの効果を具体的な数値で確認する）。

33.1.4 超平面と決定境界

線形分類器は、入力 $\mathbf{x}$ に対して 2 つの幾何学的な問いを投げかける。決定境界（decision boundary）のどちら側にあるか、そしてその境界からどれだけ離れているかである。本節を通して、この境界を $\mathbf{w} \cdot \mathbf{x} + b = 0$ と書く。

33.1.4.1 超平面と符号付き距離

平面上ではこの方程式は直線を定め、簡単な例から $\mathbf{w}$ と b の役割がわかる。$\mathbf{w} = (2, 1)$、$b = -1$ としよう。$2x + y - 1 = 0$ を満たす点は直線 $y = 1 - 2x$ をなす。この直線上のすべての点 $\mathbf{x}$ は $\mathbf{w} \cdot \mathbf{x} = -b$ を満たすため、$\mathbf{w}$ への影はすべて同じ符号付き長さ Equation 33.4 を持つ。

$$\frac{\mathbf{w} \cdot \mathbf{x}}{\|\mathbf{w}\|} = \frac{-b}{\|\mathbf{w}\|} = \frac{1}{\sqrt{5}}.$$

幾何学的には、この直線は $\mathbf{w}$ に垂直に走り、符号付き位置 $-b/\|\mathbf{w}\|$、すなわち原点からの距離 $|b|/\|\mathbf{w}\| = 1/\sqrt{5} \approx 0.45$ の位置でその方向を横切る。

高次元でも事情はまったく変わらない。$\mathbb{R}^d$ における**超平面**（hyperplane）とは、集合 $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$（ただし $\mathbf{w} \neq \mathbf{0}$）のことであり、$\mathbf{w}$ をその**法線**（normal）ベクトル、b をその**オフセット**と呼ぶ。これは部分空間 $\mathbf{w} \cdot \mathbf{x} = 0$ を平行移動させたものにすぎず、その次元は $d - 1$ である（33.2.2.2 節）。すなわち、$\mathbb{R}^2$ では直線、$\mathbb{R}^3$ では平面、画像空間内では 783 次元の平坦な面となる。超平面は $\mathbb{R}^d$ を 2 つの**半空間**（half-space）$\mathbf{w} \cdot \mathbf{x} + b > 0$ と $\mathbf{w} \cdot \mathbf{x} + b < 0$ に分割する。b を変化させると、超平面は傾きを変えずに $\mathbf{w}$ に沿って平行移動する。

任意の点 $\mathbf{x}$ は超平面からどれだけ離れているだろうか。超平面上の任意の点 $\mathbf{x}_0$ をとると、$\mathbf{w} \cdot \mathbf{x}_0 = -b$ が成り立つ。$\mathbf{x}$ から超平面までの距離は、変位 $\mathbf{x} - \mathbf{x}_0$ の法線 $\mathbf{w}$ への影の符号付き長さであり、次の**符号付き距離**（signed distance）を与える。

$$\frac{\mathbf{w} \cdot (\mathbf{x} - \mathbf{x}_0)}{\|\mathbf{w}\|} = \frac{\mathbf{w} \cdot \mathbf{x} + b}{\|\mathbf{w}\|}. \tag{33.11}$$

$\mathbf{x}_0$ が完全に消えていることに注目しよう。符号付き距離は $\mathbf{x}$、$\mathbf{w}$、b のみに依存する。その値は、$\mathbf{w}$ が向く側で正、反対側で負、超平面上ではゼロになる。先ほどの平面の例では、原点は符号付き距離 $-1/\sqrt{5}$ の位置にあり、点 $(1, 1)$ はスコア $\mathbf{w} \cdot \mathbf{x} + b = 2$ を持つため正の側の符号付き距離 $+2/\sqrt{5} \approx 0.89$ の位置にある。直線上でこの点に最も近い点は $(1, 1) - (2/5)(2, 1) = (0.2, 0.6)$ である。点 $(0, 1)$ のスコアは 0 であり、ちょうど直線上に乗っている（Figure 33.7）。

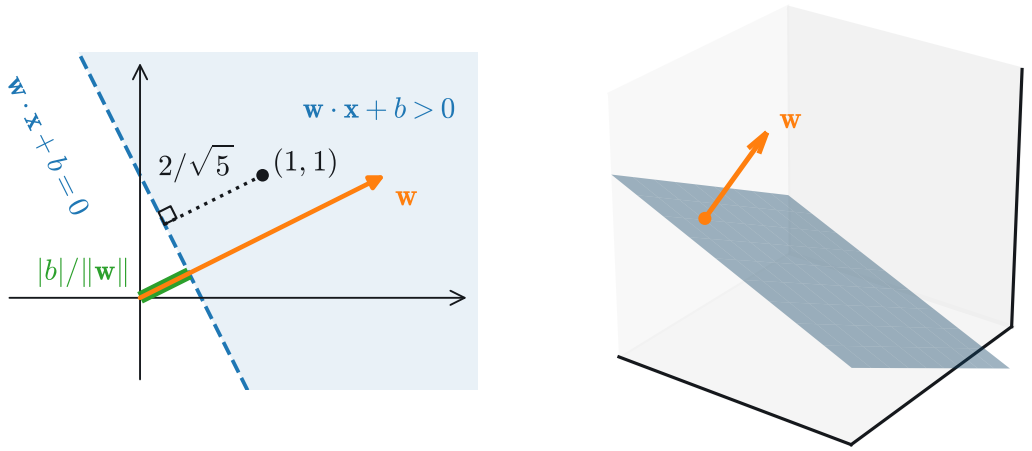


Figure 33.7: 超平面 $\mathbf{w}\cdot\mathbf{x}+b=0$（法線 $\mathbf{w}$）。左：$\mathbf{w}=(2,1)$、$b=-1$ のとき、超平面は直線 $2x+y=1$（原点からの距離 $1/\sqrt{5}$）であり、点 $(1,1)$ は符号付き距離 $2/\sqrt{5}$ の位置（$\mathbf{w}$ が向く側）にあり、色付きの半平面は $\mathbf{w}\cdot\mathbf{x}+b>0$ を表す。右：3 次元では同じ方程式が平面を定める。

ラベル付きデータ例 $(\mathbf{x}, y)$（ただし $y\in\{-1,+1\}$）に対しては、それが**正しい**側にあるかどうかも知りたい。符号付き距離にラベルを掛けたものは**マージン** $y\,(\mathbf{w}\cdot\mathbf{x}+b)/\|\mathbf{w}\|$ を定め、分類が正しいときに正となり、点が本来の半空間の奥深くにあるほど大きくなる。$(\mathbf{w}, b)\to(c\mathbf{w}, cb)$（任意の $c>0$）と再スケーリングしても超平面とマージンは変わらない一方、生のスコア $\mathbf{w}\cdot\mathbf{x}+b$ は c 倍になる。したがって、生のスコアを最大化するような学習目的関数は、いずれも $\mathbf{w}$ のスケールに制約を課さなければならない。

33.1.4.2 最近傍平均分類器

2 つのクラスを分離するには、どの超平面を選ぶべきだろうか。最も単純な規則には反復的な訓練すら不要で、平均がより近いほうのクラスへ $\mathbf{x}$ を割り当てればよい。驚くべきことに、この規則自体がただ 1 つの超平面になる。2 つのクラス平均を $\bar{\mathbf{x}}_0$ と $\bar{\mathbf{x}}_1$ としよう。両方の距離の二乗を展開すると、二次項 $\|\mathbf{x}\|^2$ が両辺で相殺される。

$$\begin{aligned}\|\mathbf{x}-\bar{\mathbf{x}}_1\|^2 < \|\mathbf{x}-\bar{\mathbf{x}}_0\|^2 &\iff -2\,\mathbf{x}\cdot\bar{\mathbf{x}}_1+\|\bar{\mathbf{x}}_1\|^2 < -2\,\mathbf{x}\cdot\bar{\mathbf{x}}_0+\|\bar{\mathbf{x}}_0\|^2\\ &\iff (\bar{\mathbf{x}}_1-\bar{\mathbf{x}}_0)\cdot\mathbf{x}-\tfrac{1}{2}\big(\|\bar{\mathbf{x}}_1\|^2-\|\bar{\mathbf{x}}_0\|^2\big)>0.\end{aligned}$$

恒等式 $\|\mathbf{a}\|^2-\|\mathbf{c}\|^2=(\mathbf{a}-\mathbf{c})\cdot(\mathbf{a}+\mathbf{c})$ を使うと、最後の不等式は次の定義のもとで $\mathbf{w}\cdot\mathbf{x}+b>0$ となる。

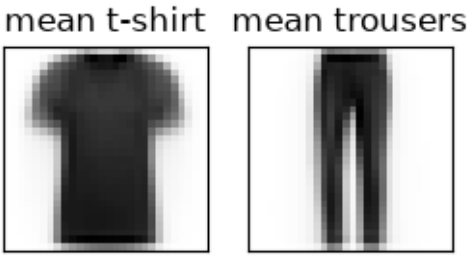
$$\mathbf{w}=\bar{\mathbf{x}}_1-\bar{\mathbf{x}}_0,\qquad b=-\mathbf{w}\cdot\tfrac{1}{2}\,(\bar{\mathbf{x}}_0+\bar{\mathbf{x}}_1)\,.$$

つまり「最近傍平均」とは垂直二等分超平面にほかならない。その法線は一方の平均から他方の平均へと向き、2 つの平均の中点をまっすぐ横切る。手動で調整すべきしきい値はなく、この規則はピクセルの一様な再スケーリングに対して不変である。すべてのピクセルに $c>0$ を掛けると $\mathbf{w}\to c\mathbf{w}$、$b\to c^2 b$ となり、すべてのスコアが c^2 倍されるだけで予測は 1 つも変わらない（2 ではこの不変性を回転と平行移動へ拡張し、逆に予測を**変えてしまう**ピクセルごとの再スケーリングを構成する）。

この分類器を T シャツとズボンで試す前に、2 つのクラス平均を観察し、そのなす角を計算してみよう。

```
X0, X1 = X_train[y_train == 0], X_train[y_train == 1]    # t-shirts, trousers
mean_0, mean_1 = X0.mean(axis=0), X1.mean(axis=0)
d2l.show_images([mean_0.reshape(28, 28), mean_1.reshape(28, 28)], 1, 2,
                titles=['mean t-shirt', 'mean trousers'], cmap='Greys')
np.degrees(angle(mean_0, mean_1))
```

```
np.float64(33.20917997376519)
```



どちらの平均画像もぼやけてはいるが、ひと目で判別できる。両者のなす角は約 33°（コサインにして 0.84）にすぎず、直交からは程遠い。どちらの衣類も画像中央の明るい領域を占めているからである。2 つの平均の角度がこれほど近いと悲観的になるかもしれないが、最近傍平均分類器（nearest-mean classifier）は平均同士の直交性を必要とせず、両者の差のみに依拠する。T シャツとズボンのテスト画像全 2,000 枚で評価してみよう。

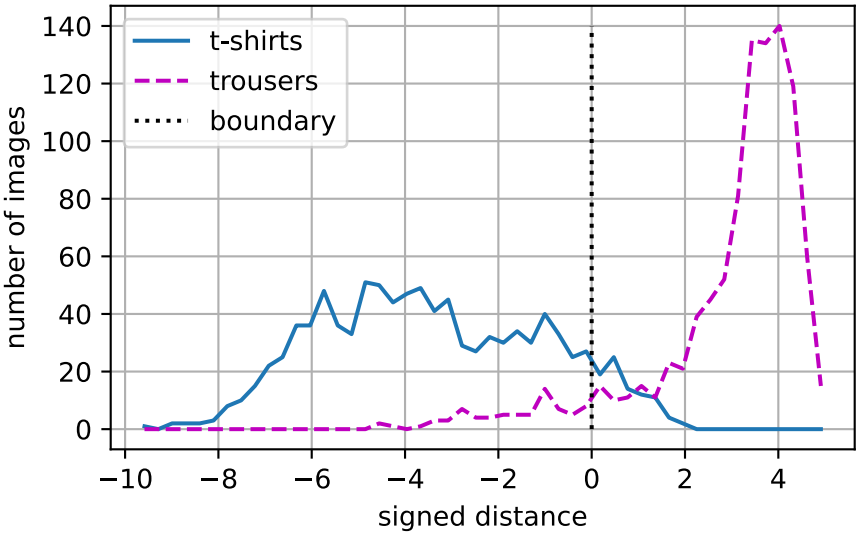
```
w = mean_1 - mean_0
b = -w @ (mean_0 + mean_1) / 2
mask = (y_test == 0) | (y_test == 1)
X2, y2 = X_test[mask], y_test[mask]
predictions = X2 @ w + b > 0                              # True: trousers
np.mean(predictions == (y2 == 1))
```

```
np.float64(0.9155)
```

この規則はテスト画像の約 92% を正しく分類し、誤った側に振り分けられたのはわずか 169 枚（全 2,000 枚中）である。各クラスの平均をとり、1 つの超平面のどちら側に各テスト画像が入るかを調べただけでこの結果が得られた。

残りの誤分類がどこから生じるかを理解するため、すべてのテスト画像を境界までの符号付き距離 Equation 33.11 という 1 つの数値に圧縮し、クラスごとにヒストグラムを描いてみよう。

```
dist = (X2 @ w + b) / np.linalg.norm(w)         # signed distance to boundary
bins = np.linspace(dist.min(), dist.max(), 51)
counts = [np.histogram(dist[y2 == c], bins)[0] for c in (0, 1)]
top = max(counts[0].max(), counts[1].max())
d2l.plot([bins[:-1]] * 2 + [[0, 0]], counts + [[0, top]], 'signed distance',
         'number of images', fmts=('-', 'm--', 'k:'),
         legend=['t-shirts', 'trousers', 'boundary'], figsize=(5, 3))
```



2 つのクラスは明確に分かれた山をなすが、それぞれの裾が境界をまたいでいる。T シャツは -4 付近を中心とする幅広い山をなす一方、ズボンは $+4$ 付近に鋭いピークを持ち、ゼロに向かって長い裾を引いている。平均的には、各クラスはその訓練平均の符号付き距離 $\mp\|\mathbf{w}\|/2 \approx \mp 3.2$ の近くに位置する。$\mathbf{w}$ に沿った各クラスの標準偏差を計算し、誤った側に入るデータ例の個数を数えて、この様子を定量化しよう。

```
(dist[y2 == 0].std(), dist[y2 == 1].std(),
 int((dist[y2 == 0] > 0).sum()), int((dist[y2 == 1] < 0).sum()))
```

```
(np.float64(2.356225995048113), np.float64(1.6627004538100882), 101, 68)
```

$\mathbf{w}$ に沿った標準偏差は T シャツが約 2.4、ズボンが約 1.7 であり、T シャツ 101 枚とズボン 68 枚が誤分類されている。各クラスの中心から境界までの距離が標準偏差の 2 倍未満であるため、数パーセントの誤り率が生じるのは当然の結果である。ここに中点規則の本質的な限界がある。2 つのクラス平均しか見ておらず、どちらの点群の大きさや形状も考慮する術を持たないのである。では、T シャツ対シャツやプルオーバー対コートのように、見た目がはるかに似通った難しいペアに適用するとどうなるだろうか。

```
# t-shirt/trouser, t-shirt/shirt, pullover/coat
for a, c in [(0, 1), (0, 6), (2, 4)]:
    m_a = X_train[y_train == a].mean(axis=0)
    m_c = X_train[y_train == c].mean(axis=0)
    w_ac, b_ac = m_c - m_a, -(m_c - m_a) @ (m_a + m_c) / 2
    pair = (y_test == a) | (y_test == c)
    acc = np.mean((X_test[pair] @ w_ac + b_ac > 0) == (y_test[pair] == c))
    dist_ac = (X_test[pair] @ w_ac + b_ac) / np.linalg.norm(w_ac)
    sd_a, sd_c = (dist_ac[y_test[pair] == k].std() for k in (a, c))
    names = data.text_labels([a, c])
    print(f'{names[0]:>8} vs {names[1]:<9}accuracy {acc:.3f}   '
          f'||w||/2 {np.linalg.norm(w_ac) / 2:.2f}   '
          f'std along w {sd_a:.2f}, {sd_c:.2f}')
```

```
 T-shirt vs trouser  accuracy 0.915   ||w||/2 3.23   std along w 2.36, 1.66
 T-shirt vs shirt    accuracy 0.786   ||w||/2 1.86   std along w 1.77, 2.38
pullover vs coat     accuracy 0.716   ||w||/2 1.24   std along w 1.88, 2.09
```

正解率は急落し、T シャツ対シャツでは約 79%、プルオーバー対コートでは約 72% まで落ち込む。残りの出力列がその理由を物語っている。各訓練平均は境界から距離 $\|\mathbf{w}\|/2$ の位置にあり、重なり具合を左右するのは、この距離を $\mathbf{w}$ に沿った各クラスの標準偏差を単位として測った値である。T シャツとズボンでは、その隔たりは標準偏差の 1.4 倍から 1.9 倍ある。ところが難しい 2 組のペアではわずか 0.6 倍から 1.0 倍に縮まるため、中点に置いた境界が両クラスの中心部をまっぷたつに横切ってしまう。その処方箋は、クラスのばらつきが大きい方向を避けるように $\mathbf{w}$ を傾けることであり、これがフィッシャーの線形判別（Fisher's linear discriminant、4）の考え方である。ソフトマックス回帰はさらに一歩進めて傾きとオフセットの両方をデータから直接学習し、深層ネットワークは入力そのものを変換して、単一の超平面でクラスがほぼ分離可能になる表現を獲得する。

33.1.4.3 超平面の積層としての層

ニューラルネットワークの線形層は、多数の超平面判定を並列に実行している。第 i 成分（行列・ベクトル積 $\mathbf{Ax}$ の成分）は、第 i 行（$\mathbf{A}$ の行）と $\mathbf{x}$ のドット積、すなわちスコア $\mathbf{a}_i \cdot \mathbf{x}$（法線 $\mathbf{a}_i$ に対するスコア）であり、バイアスベクトルを加えると各成分が超平面スコアになる。たとえば、3.1 節における Fashion-MNIST 上の 10 クラスソフトマックス層は、10 個の法線 $\mathbf{w}_1, \dots, \mathbf{w}_{10}$ と 10 個のオフセットを保持している。スコアが最大のクラスを予測するため、クラス i とクラス j を分ける決定境界は、両者のスコアが並ぶ超平面 $(\mathbf{w}_i - \mathbf{w}_j) \cdot \mathbf{x} + (b_i - b_j) = 0$ となる。

具体例として、本章で通しで用いる行列 $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ と入力 $\mathbf{x} = (2, -1)$ をとろう。第 1 行からは $1 \cdot 2 + 2 \cdot (-1) = 0$ が得られ、$\mathbf{x}$ は原点を通る法線 $(1, 2)$ の超平面上にちょうど乗っていることがわかる。第 2 行からは $-2 - 3 = -5$ が得られ、$\mathbf{x}$ は法線 $(-1, 3)$ の超平面の負の側に符号付き距離 $-5/\sqrt{10} \approx -1.58$ だけ離れて位置する。両方の結果を並べれば $\mathbf{Ax} = (0, -5)$ となる。

$\mathbf{Ax}$ を行ごとに眺めれば線形層が超平面判定の積層であることがわかるが、2 つの本質的な問いが未解決のまま残る。この変換はどの入力をゼロに潰すのか、そしてその作用はいつ逆転できるのかという問いである。33.2 節では、まったく同じ積を列ごとに読み解くことでこの両方の問いに答える。

33.1.5 まとめ

ドット積は、平面上の長さ、角度、射影をまったく同じ公式のまま任意の次元へと拡張する。直線や部分空間への射影は、射影先の空間と直交する残差によって特徴づけられる唯一の最近接点を与え、このただ 1 つの幾何学的事実からコーシー・シュワルツの不等式（高次元の角度を矛盾なく定義可能にする）と最小二乗法の正規方程式が導かれる。グラム・シュミットの直交化法は任意の生成系を正規直交基底へと変換し、QR 分解の因子 $\mathbf{Q}$ を与える。$\mathbb{R}^d$ におけるランダムな方向は典型的に $1/\sqrt{d}$ のオーダーのコサインを持ってほぼ直交する一方、実際の画像データセットは等方的とは程遠いため、観測されたコサインは同じ分布からとった無関係なペアと比較して初めて意味を持つ。最後に、最近傍平均分類器は 2 つのクラス平均を二等分する単一の超平面を定め、T シャツ対ズボンでは約 92% の正解率に達するが、境界の法線方向に沿ったクラスの広がりが効いてくる見た目の似たペアでは 72%〜79% に低下する。

33.1.6 演習問題

1. [code] **ランダムベクトル間のコサイン**。

1. 標準ガウスベクトルを正規化することで、独立なランダム単位ベクトルのペアを $1{,}000$ 組（$d = 10{,}000$ 次元）サンプリングし、各ペアのコサインを計算せよ。経験標準偏差を概直交性の命題による予測値 $1/\sqrt{d} = 0.01$ と比較し、観測された最大の $|\cos\theta|$ を報告せよ。
2. ある検索システムは 1 つのクエリを n 個のキーと比較し、最大のコサインを持つキーを返す。クエリとキーを $d = 256$ 次元の独立なランダム単位ベクトルとしてモデル化せよ。n 個のコサインの最大値を $n = 10$、10^3、10^5 に対して計算し、主要項の評価である $\sqrt{2\ln n/d}$（n 個の独立な分散 $1/d$ のガウス乱数の最大値に対する評価）と比較せよ。10^5 個のキーの中で最上位に一致した無関係なキーは、どの程度の大きさのコサインに達するか。2 つの埋め込みが関連していると判定するためのしきい値は、n と d に応じてどのようにスケーリングすべきか。

2. **最近傍平均と不変性。**最近傍平均規則は、$\mathbf{x}$ をクラス 1 に割り当てる条件を $\mathbf{w}\cdot\mathbf{x} + b > 0$（ただし $\mathbf{w} = \bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_0$、$b = -\mathbf{w}\cdot\frac{1}{2}(\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1)$）とする。
   1. すべての訓練画像とテスト画像に同じ変換 $\mathbf{x} \mapsto c\mathbf{Q}\mathbf{x} + \mathbf{z}$（ただし $c > 0$、$\mathbf{Q}$ は $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ を満たす正方行列、$\mathbf{z}$ は固定されたベクトル）を施し、クラス平均と b を再計算するとする。予測が一切変わらないことを示し、これを本文で議論した再スケーリング $\mathbf{w} \to c\mathbf{w}$、$b \to c^2 b$ と関連づけよ。
   2. 各ピクセルを単位分散に標準化すると、$\mathbf{x}$ は $\mathbf{D}\mathbf{x}$（$\mathbf{D}$ は正の対角成分を持つ対角行列）に置き換わる。$\mathbb{R}^2$ において、このような $\mathbf{D}$ のもとで予測クラスが反転する 2 つのクラス平均とテスト点を構成せよ。
   3. 最近傍平均規則のもとで q 個のクラスがあるとき、クラス i に割り当てられる領域が $q-1$ 個の半空間 $(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j)\cdot\mathbf{x} + b_{ij} > 0$ $(j \neq i)$ の共通部分になることを示し、b_{ij} を求めよ。これをソフトマックス層のペアごとの境界 $(\mathbf{w}_i - \mathbf{w}_j)\cdot\mathbf{x} + (b_i - b_j) = 0$ と比較せよ。重みとバイアスをどのように選べば、ソフトマックス層は最近傍平均分類器を再現するか。

3. **直交射影行列と斜射影行列。**
   1. $\mathbf{P}$ が $\mathbf{P}^2 = \mathbf{P}$ と $\mathbf{P}^\top = \mathbf{P}$ を満たすとする。すべての $\mathbf{x}$ に対して、$\mathbf{P}\mathbf{x}$ が $\mathbf{P}$ の列空間に属し、$\mathbf{x} - \mathbf{P}\mathbf{x}$ がその列空間と直交することを示せ。正規直交基底を用いることなく、ピタゴラスの定理から $\mathbf{P}\mathbf{x}$ が列空間内の最近接点であることを導け。
   2. 行列 $\mathbf{P}_{\mathrm{o}} = \left(\begin{smallmatrix}1 & 1\\ 0 & 0\end{smallmatrix}\right)$ は $\mathbf{P}_{\mathrm{o}}^2 = \mathbf{P}_{\mathrm{o}}$ を満たすが対称ではない。$\mathbf{x} = (1, 1)$ に対して、$\mathbf{P}_{\mathrm{o}}\mathbf{x}$ と、同じ列空間（第 1 座標軸）への $\mathbf{x}$ の直交射影、およびそれぞれと $\mathbf{x}$ との距離を計算せよ。$\mathbf{P}_{\mathrm{o}}$ はどの方向に沿って点を射影するか。
   3. 任意の行列 $\mathbf{P}$（$\mathbf{P}^2 = \mathbf{P}$）が、すべての $\mathbf{x}$ を $\mathbf{P}\mathbf{x} + (\mathbf{x} - \mathbf{P}\mathbf{x})$ と分解し、第 1 項が $\mathbf{P}$ の列空間に、第 2 項がその零空間 $\{\mathbf{z} : \mathbf{P}\mathbf{z} = \mathbf{0}\}$ に属することを示せ。これら 2 つの部分空間が直交するための必要十分条件が $\mathbf{P}^\top = \mathbf{P}$ であることを示せ。

4. [code] **境界を傾ける。**●平均の差をとった法線 $\Delta = \bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_0$ は、各クラスの画像がその平均の周りにどう広がっているかを無視している。訓練セット上での 2 つのクラス内共分散行列の平均を $\mathbf{C}$ としよう。フィッシャーの線形判別 (Fisher 1936) は、次式を最大化する法線 $\mathbf{w}$ を選ぶ。

$$J(\mathbf{w}) = \frac{(\mathbf{w}\cdot\Delta)^2}{\mathbf{w}^\top\mathbf{C}\mathbf{w}},$$

これは射影されたクラス平均間の距離の二乗を、射影のクラス内分散で割った量である。$\mathbf{C}$ が正定値（$\mathbf{v}^\top\mathbf{C}\mathbf{v} > 0$ がすべての $\mathbf{v} \neq \mathbf{0}$ に対して成り立つ）のとき、逆行列 $\mathbf{C}^{-1}$ が存在し、最大点は $\mathbf{w} \propto \mathbf{C}^{-1}\Delta$ となる。（本問では 33.2.2.3 節の逆行列、33.3.2.2 節の正定値性、および 33.5.3.2 節の条件数を用いる。）

1. $J(c\mathbf{w}) = J(\mathbf{w})$ がすべての $c \neq 0$ に対して成り立つことを示せ。$\mathbf{C} = s^2\mathbf{I}$ のとき、コーシー・シュワルツの不等式を適用して Δ が J を最大化することを示せ。
2. 本文中の T シャツとズボンの配列 `X0` と `X1` を使って $\mathbf{C}$ を計算し、平均ピクセル分散を $\bar{s}^2 = \mathrm{tr}(\mathbf{C})/784$ とおけ。これらの画像では端のピクセルがほぼ一定であるため、$\mathbf{C}$ の条件数は約 10^9 に達する。$\gamma \in \{10^{-4}, 10^{-2}, 1, 10^2, 10^4\}$ に対して正則化した法線 $\mathbf{w}_\gamma = (\mathbf{C} + \gamma \bar{s}^2 \mathbf{I})^{-1}\Delta$ を作り、境界を 2 つの平均の中点に置いて訓練正解率とテスト正解率の両方を評価せよ。次に、各クラスから訓練画像を 1,000 枚ずつホールドアウトし、残りの画像で $\mathbf{C}$ とクラス平均を再計算して、ホールドアウトセット上の検証正解率によって γ を選び、その $\mathbf{w}_\gamma$ を用いて符号付き距離のヒストグラムを再びプロットせよ。
3. なぜ $\mathbf{w}_\gamma$ は Δ の方向へ（$\gamma \to \infty$ の極限で）揃うのか。また、$\bar{s}^2$ で正則化項をスケーリングし、$\mathbf{I}$ の固定定数倍をそのまま足さないのはなぜか。

5. [code] **浮動小数点におけるグラム・シュミット法。**●グラム・シュミットの直交化法は、代数的に同値な 2 通りの順序で実行できる。古典的グラム・シュミット法（CGS）は、2 ベクトルの例と同様に、$\mathbf{q}_j$ を $\mathbf{a}_j - \sum_{i<j} (\mathbf{q}_i \cdot \mathbf{a}_j)\,\mathbf{q}_i$ の正規化によって計算する。修正グラム・シュミット法（MGS）は $\mathbf{v} = \mathbf{a}_j$ と初期化し、$\mathbf{v} \leftarrow \mathbf{v} - (\mathbf{q}_i \cdot \mathbf{v})\,\mathbf{q}_i$ を $i = 1, \dots, j-1$ について順に更新してから $\mathbf{v}$ を正規化する。

   1. 厳密な演算においては CGS と MGS が同一のベクトルを生成することを示せ。
   2. 両方のアルゴリズムを実装し、ほぼ線形従属な列を持つヒルベルト行列 $\mathbf{H}_n$（成分は $1/(i+j-1)$、$n = 4, 6, 8, 10, 12$）に対して実行せよ。フロベニウスノルム $\|\mathbf{Q}^\top\mathbf{Q} - \mathbf{I}\|_F$ を n に対して CGS、MGS、`np.linalg.qr` のそれぞれでプロットせよ。各手法について、どの n で $\mathbf{P} = \mathbf{Q}\mathbf{Q}^\top$ が射影行列として機能しなくなるか（$\|\mathbf{P}^2 - \mathbf{P}\|_F$ によって測れ）。
   3. 各列に対して CGS の引き算ステップを 2 回適用せよ（1 回目のパスで残ったベクトルに対して 2 回目のパスを実行する）。得られる列は、1 回のみの CGS や MGS と比べてどれほど正規直交に近いか。1 回目のパスに入るベクトルと 2 回目のパスに入るベクトルそれぞれに含まれる $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$ 方向の成分の大きさを比較して、精度が改善する理由を説明せよ。

6. [extended] **ほぼ直交する方向のパッキング。**●●$\mathbb{R}^d$ には互いに直交する方向は高々 d 個しか収まらないが、ペアごとのコサインを小さく保ったままなら、それよりはるかに多くの方向を詰め込むことができる。Elhage et al. (2022) はこの幾何学を用いて、幅 d の層が d 個を超える特徴量をほぼ直交する方向として**重ね合わせ**の形で表現できる仕組みを説明している。ここで扱われる特徴量は**疎**であり、各特徴量はごく一部の入力でしか活性化しないため、一度に読み出す必要があるのは少数にとどまる。$\mathbf{u}_1, \dots, \mathbf{u}_n$ を $\mathbb{R}^d$ 内の単位ベクトルとし、それらを行として並べた行列を $\mathbf{U} \in \mathbb{R}^{n \times d}$ とし、ペア間の最悪の干渉を $\mu = \max_{i \neq j} |\mathbf{u}_i \cdot \mathbf{u}_j|$ とおく。

   1. $\sum_{i,j} (\mathbf{u}_i \cdot \mathbf{u}_j)^2$ を座標で展開し、これが $\sum_{k,l} \big(\sum_i u_{ik} u_{il}\big)^2$、すなわち $d \times d$ 行列 $\mathbf{U}^\top\mathbf{U}$ の各成分の二乗和に等しいことを示せ。d 個の対角項と、$\sum_{k=1}^{d} a_k^2 \geq \big(\sum_k a_k\big)^2 / d$ の形のコーシー・シュワルツの不等式を用い、$\sum_k (\mathbf{U}^\top\mathbf{U})_{kk} = n$ に注意してこの和を下から評価せよ。さらに元の和を上から $n + n(n-1)\mu^2$ で抑えることで、次の **Welch の下界** (Welch 1974) を導出せよ。

$$\mu^2 \geq \frac{n - d}{d\,(n-1)}.$$

   $d = 32$ のもとで $n = 64$ と $n = 512$ に対してこの下界の値を求めよ。

2. すべてのペアにわたるユニオンバウンドにより、独立なランダム方向は $\mu \le \varepsilon$ を n が $e^{d\varepsilon^2/4}$ のオーダーになるまで保つことが示される（36.9 節）。$d = 32$、$n \in \{64, 128, 256, 512\}$ に対してランダム単位ベクトルの μ を計算し、Welch の下界と比較せよ。

3. 最適化によって直接 μ を小さくせよ。すなわち、$\sum_{i \neq j} (\mathbf{u}_i \cdot \mathbf{u}_j)^p$ を $p = 10$ のような偶数のべきに対して、$\mathbf{U}$ 上の勾配ステップ（各ステップ後にすべての行を再正規化する）によって最小化せよ。$d \in \{16, 32, 48, 64\}$ と $n \in \{d, 2d, 4d, 8d, 16d\}$ に対し、ランダムなベクトル、最適化したベクトル、および Welch の下界について μ を n に対してプロットせよ。

4. これらの曲線を使って、各 d に対して最大の n（最適化した方向が $\mu \le 0.2$ を保てる上限）を推定せよ。この容量は d の増加に伴って、ランダム方向の増加率 $e^{d\varepsilon^2/4}$ と比べどのように増大するか。もし一部の特徴量が他より重要であったり、多くの特徴量が同時に活性化したりする場合、結論はどのように変わるか。

33.2 線形変換としての行列

33.1 節では、行列とベクトルの積 $\mathbf{A}\mathbf{x}$ を 1 行ずつ読み解いた。第 i 成分は第 i 行と $\mathbf{x}$ のドット積であり、1 回の超平面判定にあたる。この行ごとの見方は、1 つの入力ベクトルに対して行列が何を計算するかは教えてくれるが、入力空間全体に一度に何をするかは見えにくい。とりわけ、その写像が情報を保つのか捨ててしまうのか、またその作用を元に戻せるのかどうかはわからない。

この問いは、ニューラルネットワークの層が入力より狭くなった瞬間に現れる。Fashion-MNIST 画像の 784 ピクセルを 256 個の隠れ特徴量へ写す全結合層（fully connected layer）は、256×784 行列である。出力が保持する数値の個数は入力より少ないため、空間は必然的に潰される。まもなく厳密に数えるように、少なくとも $784 - 256 = 528$ 個の独立な入力方向が 0 に送られるため、非線形性（nonlinearity）を通る前の段階で、異なる画像がまったく同じ特徴量を生み出しうる。

正方行列であっても情報を失うことがある。本章で通しで用いる行列 $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ を考えよう。この行列は $(2, -1)$ を $(0, -5)$ に送り、$(0, -5)$ をそのまま $(2, -1)$ に戻す別の行列が存在する。実際、その 2 つ目の行列はあらゆる入力に対して $\mathbf{A}$ の作用を打ち消すため、$\mathbf{A}$ は何も失わない。これを行列 $\mathbf{B} = \left(\begin{smallmatrix} 2 & 1 \\ 4 & 2 \end{smallmatrix}\right)$ と対比してみよう。こちらは $(1, 1)$ と $(0, 3)$ の両方を $(3, 6)$ に送る。出力 $(3, 6)$ だけを与えられても、どちらの入力から生じたのかを判別するすべはないため、どのような行列も $\mathbf{B}$ を逆向きに戻すことはできない。

線形変換（linear map）を元に戻せるかどうかによって、層が情報を捨てるか、連立一次方程式（linear system）が一意な解を持つか、そして変数変換（change of variables）のもとで確率密度がどのようにスケーリングされるか（36.2.3.1 節）が決まる。この問いに明快に答えるため、視点を行から列へと移そう。行列の第 j 列とは、第 j 座標軸の行き先にほかならず、それらの像だけで写像全体が定まる。線形独立（linearly independent）な列がいくつあるかを数えれば入力の何次元が写像で生き残るかがわかり、変換後の単位立方体の符号付き体積を測れば、次元が失われたかどうかを検出する 1 つの数値が得られる。この 2 つの考え方を、まずはすべての主張を図として描ける 2 次元で展開し、続いて n 次元へと拡張する。

本節の軸となる結果は 3 つある。生き残る次元と潰れる次元の釣り合い、正方行列が逆行列を持つための厳密な条件、そして 2 つの写像を合成したときに体積の拡大率が従う規則である。一般の n 次元における行列式（determinant）の構成（33.2.3.2 節）は最も技術的な部分であり、初読の際は流し読みしてかまわない。

```
import numpy as np
```

33.2.1 線形変換

33.2.1.1 座標軸の像としての列

行列は無限に多くの入力ベクトルに作用するが、有限個の数値の表として格納されている。たった 4 つの数値で、なぜ平面上のあらゆるベクトルに対する出力が定まるのだろうか。2×2 行列 $\mathbf{A}$（成分は p, q, r, s）をとり、$\mathbf{x} = (x, y)$ に適用してみよう。x が掛かる項と y が掛かる項をそれぞれまとめると、次の式が得られる。

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} p & q \\ r & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} px + qy \\ rx + sy \end{bmatrix} = x \begin{bmatrix} p \\ r \end{bmatrix} + y \begin{bmatrix} q \\ s \end{bmatrix} = x\,\mathbf{A}\mathbf{e}_1 + y\,\mathbf{A}\mathbf{e}_2.$$

したがって、$\mathbf{A}\mathbf{x}$ とは $\mathbf{A}$ の 2 つの列を、$\mathbf{x}$ の座標を重みとして重み付き和をとったものにすぎない。$\mathbf{A}\mathbf{e}_1 = (p, r)$ かつ $\mathbf{A}\mathbf{e}_2 = (q, s)$ であるから、各列は標準基底ベクトル（standard basis vector）$\mathbf{e}_1 = (1, 0)$ と $\mathbf{e}_2 = (0, 1)$ の像である。この 2 本の軸の行き先さえわかれば、平面上の他のすべてのベクトルの像は完全に定まる。

通しで用いる行列（$p = 1$、$q = 2$、$r = -1$、$s = 3$）では、2 本の軸は $\mathbf{A}\mathbf{e}_1 = (1, -1)$ と $\mathbf{A}\mathbf{e}_2 = (2, 3)$ に移る。入力 $\mathbf{v} = (2, -1)$ は $2\mathbf{e}_1 - \mathbf{e}_2$ と表せるため、その像は列のまったく同じ線形結合になる。

$$\mathbf{A}\mathbf{v} = 2\,(1, -1) - (2, 3) = (0, -5).$$

行ごとに検算しても $(1, 2) \cdot (2, -1) = 0$ および $(-1, 3) \cdot (2, -1) = -5$ となり、結果が一致することが確かめられる。どちらの見方でも同じ積が計算されるが、列ごとの見方に立てば $\mathbf{A}$ が平面全体に一度に何をするかが見て取れる。Figure 33.8 にこの幾何学的意味を示す。$\mathbf{e}_1$ と $\mathbf{e}_2$ が張る空間である単位正方形は 2 つの列が張る平行四辺形へと運ばれ、グリッド上の点 $\mathbf{v}$ は変換後のグリッドの対応する点 $\mathbf{A}\mathbf{v}$ に着地する。

同じ列による描像は任意の次元で成り立つ。$m \times n$ 行列は $\mathbb{R}^n$ のベクトルを $\mathbb{R}^m$ のベクトルへ写し、次の意味で**線形**（linear）である。

$$\mathbf{A}(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha\,\mathbf{A}\mathbf{x} + \beta\,\mathbf{A}\mathbf{y} \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \text{ and } \alpha, \beta \in \mathbb{R}.$$

言葉で言えば、重み付き和の像は、それぞれの像の同じ重み付き和に等しい。任意の入力を $\mathbf{x} = \sum_j x_j \mathbf{e}_j$ と書けば $\mathbf{A}\mathbf{x} = \sum_j x_j\,\mathbf{A}\mathbf{e}_j$ となり、ここで $\mathbf{e}_j$ を掛ける操作は第 j 列を取り出すことにほかならない。逆に、この線形性の規則を満たすすべての写像 $f : \mathbb{R}^n \to \mathbb{R}^m$ は、第 j 列を $f(\mathbf{e}_j)$ とする行列として書ける。なぜなら、$f(\mathbf{x}) = f(\sum_j x_j \mathbf{e}_j) = \sum_j x_j f(\mathbf{e}_j)$ が成り立つからである。したがって、$\mathbb{R}^n$ から $\mathbb{R}^m$ への線形変換と $m \times n$ 行列は、同じ対象を記述する 2 つの表現にほかならない。

Figure 33.8 のグリッドが変換後もグリッドの姿を保つのも、線形性のおかげである。あらゆる直線 $\mathbf{p} + t\mathbf{u}$ は $\mathbf{A}\mathbf{p} + t\,\mathbf{A}\mathbf{u}$ に写り、これもまた直線（あるいは $\mathbf{A}\mathbf{u} = \mathbf{0}$ のときは 1 点）になる。さらに、原点は決して動かない（$\mathbf{A}\mathbf{0} = \mathbf{0}$）。

ここまでの議論に行列が正方である必要はまったくない。2×3 行列は $\mathbb{R}^3$ を $\mathbb{R}^2$ 内の 3 つの列が張る空間（span）へと写し、少なくとも 2 つの列が線形独立である限り出力平面全体を埋め尽くす。逆に、3×2 行列は平面を $\mathbb{R}^3$ の中へと写し、その出力は 2 つの列に応じて原点を通る平面、直線、または原点のみをなす。ニューラルネットワークのすべての全結合層はこのような長方形の写像であり、その後にバイアスによるシフトと非線形性が続く。

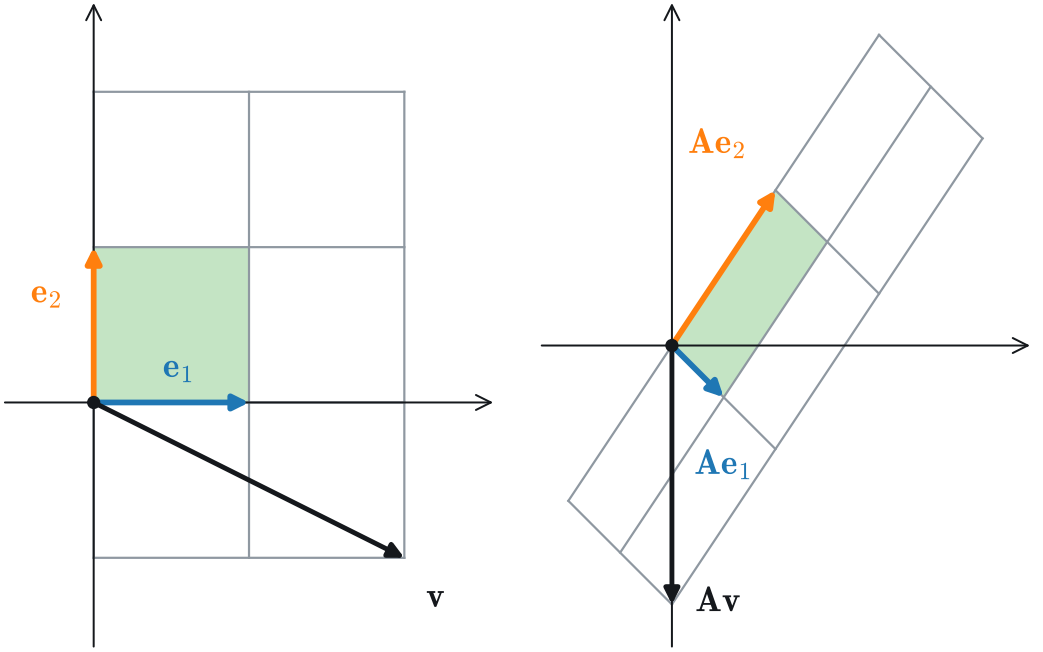


Figure 33.8: 平面に作用する通しで用いる行列 $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$。(a) 入力グリッド、網掛けの単位正方形、基底ベクトル、および入力 $\mathbf{v} = (2, -1)$。(b) その像。$\mathbf{e}_1$ は $(1, -1)$ に、$\mathbf{e}_2$ は $(2, 3)$ に、$\mathbf{v}$ は $(0, -5)$ に移る。直線は直線のまま保たれ、原点は動かず、等間隔の点は等間隔のまま保たれる。網掛けの正方形は網掛けの平行四辺形に写り、その面積から行列式が導かれる。パネル (b) は (a) の 3 分の 1 の縮尺で描かれている。

視覚的な直感を養うため、Figure 33.9 に 6 つの 2×2 行列を単位正方形への作用によって示す。反時計回りの 90° 回転は $\mathbf{e}_1$ を $\mathbf{e}_2$ に、$\mathbf{e}_2$ を $-\mathbf{e}_1$ に送るため、その列は $(0,1)$ と $(-1,0)$ になる。水平方向のせん断（shear）は $\mathbf{e}_1$ を動かさずに $\mathbf{e}_2$ を $(1,1)$ へと傾け、すべての水平線を高さに比例して横へ滑らせる。最後の 2 つのパネルは質的に異なり、次元を 1 つ失って正方形全体、ひいては平面全体を 1 本の直線へと潰してしまう。

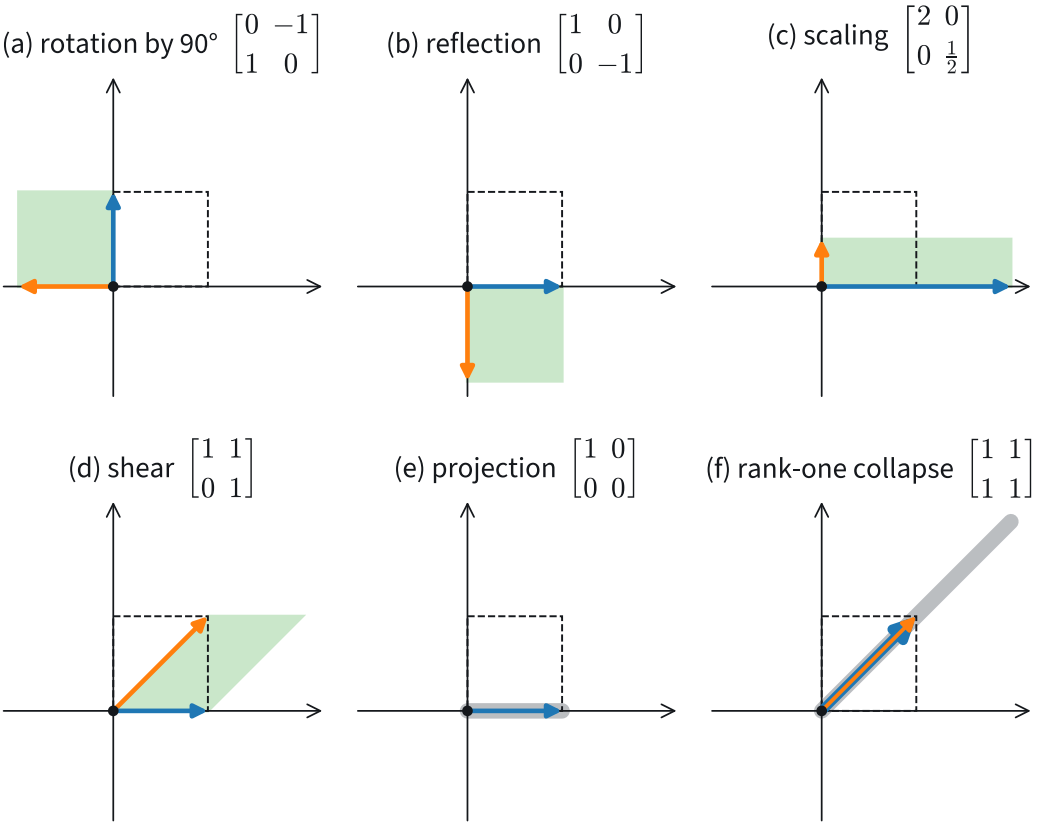


Figure 33.9: 2×2 行列が単位正方形（破線）を変換する 6 通りの様子。各パネルで、矢印は $\mathbf{e}_1$（青）と $\mathbf{e}_2$（オレンジ）の像である各列を表し、網掛け領域は正方形の像を表す。パネル (e) と (f) は平面を直線へと潰しており、(f) では両方の列が $(1,1)$ である。

このギャラリーには、おなじみの操作が 1 つ見当たらないことに気づくだろう。平行移動（translation）$\mathbf{x} \mapsto \mathbf{x} + \mathbf{c}$（ただし $\mathbf{c} \neq \mathbf{0}$）である。原点を動かしてしまうため、これは線形ではない。したがって、ニューラルネットワークにおけるアフィン層 $\mathbf{W}\mathbf{x} + \mathbf{b}$ は、線形変換の後に平行移動を施したものである。

33.2.1.2 合成

非線形性を挟まずに 2 つの線形層を直列に並べると、1 つの線形変換に縮約される。その合成された写像を表すのはどの行列だろうか。Figure 33.9 の 2 つの具体的な写像、すなわち 90° 回転 $\mathbf{R}_{90}$ と（縦軸を傾けることにちなんで名付けた）せん断 $\mathbf{T}$ で確かめてみよう。

$$\mathbf{R}_{90} = \begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}.$$

この 2 つの行列は本章全体を通して繰り返し登場する。それぞれの順序で標準基底ベクトルがどこへ移るかを追跡しよう。先にせん断を施すと、$\mathbf{e}_1$ は $(1,0)$ にとどまり、続く回転によって $(0,1)$ に送られる。一方、$\mathbf{e}_2$ はせん断で $(1,1)$ に移り、回転で $(-1,1)$ に移る。したがって、「せん断してから回転」の列は $(0,1)$ と $(-1,1)$ になる。逆の順序では、回転が $\mathbf{e}_1$ を $(0,1)$ に送り、せん断がそれを $(1,1)$ に動かす。また、回転が $\mathbf{e}_2$ を $(-1,0)$ に送り、せん断はそれを動かさない。ゆえに、2 つの合成写像は次のようになる。

$$\text{shear, then rotate: } \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}, \qquad \text{rotate, then shear: } \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}.$$

Figure 33.10 から明らかなように、この 2 つは同じ写像ではない。

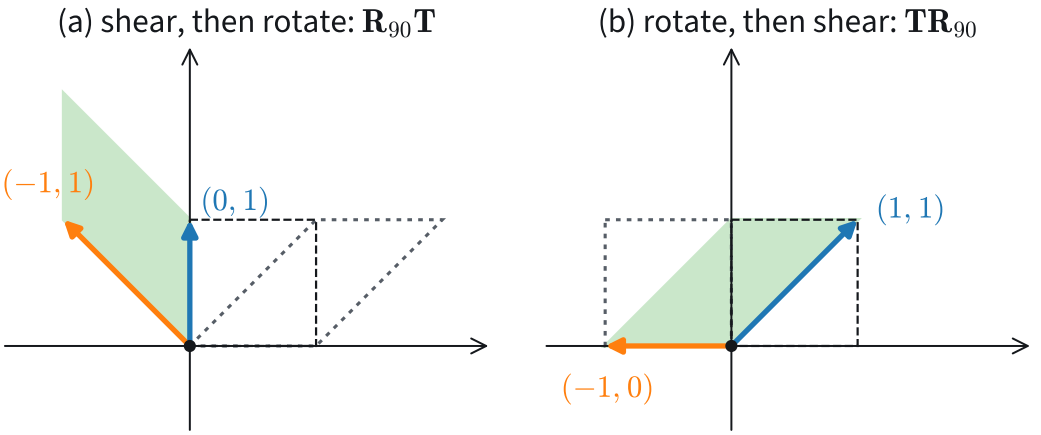


Figure 33.10: 合成の順序は結果を変える。(a) せん断 $\mathbf{T}$（点線の枠）に続けて回転 $\mathbf{R}_{90}$ を適用すると $\mathbf{R}_{90}\mathbf{T}$ になる。(b) 回転（点線の枠）に続けてせん断を適用すると $\mathbf{T}\mathbf{R}_{90}$ になる。ラベルは $\mathbf{e}_1$（青）と $\mathbf{e}_2$（オレンジ）の最終的な像を示す。行列積は常に右から左へと読む。

このような合成を計算する代数規則が行列積（matrix multiplication）である。次元が整合する任意の 2 つの行列 $\mathbf{A}$ と $\mathbf{B}$ について、まず $\mathbf{B}$ を $\mathbf{v}$ に適用し、次にその結果へ $\mathbf{A}$ を適用して、第 i 成分を調べてみよう。

$$\big(\mathbf{A}(\mathbf{B}\mathbf{v})\big)_i = \sum_j a_{ij}(\mathbf{B}\mathbf{v})_j = \sum_j \sum_k a_{ij} b_{jk} v_k = \sum_k \Big(\sum_j a_{ij} b_{jk}\Big) v_k = \big((\mathbf{A}\mathbf{B})\mathbf{v}\big)_i. \tag{33.12}$$

2 つの写像を順に適用することは、単一の行列 $\mathbf{AB}$ を適用することとまったく同じであり、その (i,k) 成分は $\mathbf{A}$ の第 i 行と $\mathbf{B}$ の第 k 列のドット積である。順序に注意しよう。積 $\mathbf{AB}$ は「先に $\mathbf{B}$ を適用し、次に $\mathbf{A}$ を適用する」ことを意味し、関数の合成 $f(g(x))$ と同じく右から左へと読む。また、$\mathbf{v} = \mathbf{e}_j$ を Equation 33.12 に代入すると有用な列の規則も得られる。すなわち、$\mathbf{AB}$ の第 j 列は $\mathbf{A}\mathbf{b}_j$ であり、これは $\mathbf{B}$ の第 j 列の $\mathbf{A}$ による像にほかならない。この事実は行列式を調べる際に活躍する。

積を写像の合成として捉えれば、一見奇妙に見えるその定義にも納得がいく。「行と列の積」という公式は恣意的な約束事ではなく、2 つの線形変換を合成した結果そのものなのである。代数的性質もここから直ちに従う。行列積が結合法則 $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$ を満たす（associative）のは、

関数の合成が結合法則を満たすからである。すなわち、両辺ともまず **C**、次に **B**、最後に **A** を適用する。また、一般に可換（commutative）でないのは、2 つの幾何学的操作を逆の順序で施せば通常は異なる結果になるからである。NumPy では、行列積に `@` 演算子を用いる。回転とせん断（以下のコードでは `R90` と `S` に格納）の 2 通りの合成が再現されることを確かめてみよう。

```
R90 = np.array([[0.0, -1.0], [1.0, 0.0]])  # rotation by 90 degrees
S = np.array([[1.0, 1.0], [0.0, 1.0]])     # shear
print(R90 @ S)
print(S @ R90)
```

```
[[ 0. -1.]
 [ 1.  1.]]
[[ 1. -1.]
 [ 1.  0.]]
```

もちろん、可換になる行列の組も存在する。平面上の 2 つの回転は可換である。角度 ϕ だけ回転してから ψ だけ回転すれば、どちらの順序でも $\phi+\psi$ の回転になるからである。2 つの対角行列も、それぞれ座標軸を独立にスケーリングするだけなので可換である。演習問題 3 では、2×2 行列のうち $\mathbf{R}_{90}$ と可換なものをすべて特徴づける。

33.2.1.3 直交行列

線形変換の中には、あらゆる長さとあらゆる角度を保ったまま空間を剛体的に動かすものがある。どの行列がこの性質を持つだろうか。平面における角度 ϕ の反時計回りの回転を考えよう。この回転は $\mathbf{e}_1$ を $(\cos\phi, \sin\phi)$ に、$\mathbf{e}_2$ を $(-\sin\phi, \cos\phi)$ に送るため、その行列は次のようになる。

$$\mathbf{R}_{\phi} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}.$$

その列は単位ベクトルとしての長さ $\sqrt{\cos^2\phi+\sin^2\phi}=1$ を持ち、ドット積は $-\cos\phi\sin\phi+\sin\phi\cos\phi=0$ である。すなわち列は正規直交（orthonormal）しており、これは 33.1 節で $\mathbf{R}_{\phi}^{\top}\mathbf{R}_{\phi}=\mathbf{I}$ と書いた条件である。$\phi=90^\circ$ とおけば $\mathbf{R}_{90}$ が復元され、これは $(3,2)$ を $(-2,3)$ に送るが、どちらのベクトルも長さは $\sqrt{13}$ である。鏡映（reflection）$\mathrm{diag}(1,-1)$（x 軸に関する折り返し）も同様に正規直交する列を持つ。

任意の次元において、正方行列 $\mathbf{Q}$ の列が正規直交しているとき、すなわち次を満たすとき、その行列を**直交行列**（orthogonal matrix）と呼ぶ。

$$\mathbf{Q}^{\top}\mathbf{Q}=\mathbf{I}.$$

この定義には注意すべき微妙な点が 2 つある。第 1 に、列が互いに垂直であるだけでは不十分である。$\mathrm{diag}(1,10)$ は長さ 1 と 10 の互いに垂直な列を持つが、90° をなす $(1,1)$ と $(1,-1)$ の間の角度を $\arccos(-99/101)\approx 169^\circ$ へと曲げてしまう（1）。第 2 に、「直交」という名称は正方行列に限って用いられる。縦長の $n\times k$ 行列（$k<n$）は、列が正規直交していれば長さを保つものの、$\mathbb{R}^k$ をより大きな空間 $\mathbb{R}^n$ に埋め込むため、両側逆行列を持たない（33.2.2.3 節）。

直交変換はすべてのドット積を保つ。

$$(\mathbf{Q}\mathbf{x})\cdot(\mathbf{Q}\mathbf{y}) = (\mathbf{Q}\mathbf{x})^{\top}(\mathbf{Q}\mathbf{y}) = \mathbf{x}^{\top}\mathbf{Q}^{\top}\mathbf{Q}\,\mathbf{y} = \mathbf{x}^{\top}\mathbf{y} = \mathbf{x}\cdot\mathbf{y}.$$

$\mathbf{y} = \mathbf{x}$ とおけば $\|\mathbf{Q}\mathbf{x}\| = \|\mathbf{x}\|$ となるため、長さは変わらない。さらに、角度はドット積と長さだけで決まるため、角度も変わらない。埋め込みベクトルの集合全体に同じ直交行列を掛けて回転させても、あらゆるペアのコサイン類似度はまったく同じままである。

定義式 $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ は、転置 $\mathbf{Q}^\top$ を $\mathbf{Q}$ の**後に**適用すれば元に戻ることを表している。正方行列 $\mathbf{Q}$ については、実は $\mathbf{Q}^\top$ を $\mathbf{Q}$ の**前に**適用しても打ち消されること、すなわち $\mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$ が成り立ち、したがって $\mathbf{Q}$ の行も正規直交であり $\mathbf{Q}^{-1} = \mathbf{Q}^\top$ となる。これは定義だけからは自明ではなく、33.2.2.3 節で証明する正方行列の一般定理から導かれる。

2 次元では、直交行列は回転と鏡映に限られる。第 1 列は単位ベクトル $(\cos\phi, \sin\phi)$ でなければならず、これに垂直な第 2 列の単位ベクトルは $\pm(-\sin\phi, \cos\phi)$ の 2 通りしか残らない。正の符号をとれば回転 $\mathbf{R}_\phi$ になる。負の符号をとれば $\mathbf{R}_\phi\,\mathrm{diag}(1,-1)$、すなわち x 軸に関する鏡映の後に回転を施した写像になり、これ自身も原点を通るある直線に関する鏡映である。後ほど見るように、行列式の符号を見れば回転と鏡映を一目で区別できる（33.2.3.3 節）。

33.2.1.4 テンソルとアインシュタインの総和規約

Transformer では、自己アテンションがバッチ内のすべての系列とすべてのアテンションヘッドにわたって、あらゆるクエリとあらゆるキーを比較する。バッチを b、ヘッドを h、クエリ位置を i、キー位置を t、ヘッド内の特徴量次元を j とラベル付けすると、アテンションスコアは次の式で与えられる。

$$s_{bhit} = \sum_{j} q_{bhij}\, k_{bhtj},$$

ここで、q_{bhij} と k_{bhtj} は 4 軸のクエリテンソルとキーテンソルの成分である（1.1 節）。各スコアは特徴量軸 j に沿ったドット積であり、残る 4 つの添字のあらゆる組み合わせに対して計算される。通常の行列記法にはこの 4 添字の縮約（contraction）をすっきり書く方法がなく、「最後の 2 軸を転置したバッチ処理版の行列積」と呼んでしまうと、実際にどの軸について和をとっているのかが見えにくくなる。

一歩引いて眺めると、ここまでに出会ったすべての積がまったく同じパターンに従っていることに気づく。ドット積、行列とベクトルの積、行列積、そして正方行列の対角成分の和である**トレース** $\mathrm{tr}(\mathbf{A})$ は、いずれも次のように書ける。

$$\mathbf{v}\cdot\mathbf{w} = \sum_i v_i w_i, \qquad (\mathbf{A}\mathbf{v})_i = \sum_j a_{ij} v_j, \qquad (\mathbf{A}\mathbf{B})_{ik} = \sum_j a_{ij} b_{jk}, \qquad \mathrm{tr}(\mathbf{A}) = \sum_i a_{ii}.$$

トレースはここで初めて登場したが、33.3 節で行列の固有値の和として再び現れる。これら 4 つの式はいずれも、成分どうしを掛け合わせ、2 回現れる添字について和をとっている。**アインシュタイン記法**（Einstein notation）はこのパターンを規約にしたものであり、総和記号を省略し、繰り返されるすべての添字について自動的に和をとる。この規則のもとでは、$\mathbf{v}\cdot\mathbf{w} = v_i w_i$ および $(\mathbf{A}\mathbf{B})_{ik} = a_{ij} b_{jk}$ と書ける。

アテンションスコアの例を見ると、古典的な規約だけでは不十分な場面がわかる。バッチとヘッドの添字 b と h は両方の因子に現れるが、和をとって消すのではなく出力に残したいからである。数値計算ライブラリの `einsum` は、残す添字を明示的に指定させることでこれを解決する。入力の添字を並べた文字列、矢印 `->`、出力の添字を渡すと、右辺で省略された添字について和がとられる。4 つの基本的な積をこの記法で表現し、通しで用いる行列で確かめてみよう。

```
A = np.array([[1.0, 2.0], [-1.0, 3.0]])    # the running matrix
v = np.array([2.0, -1.0])
(np.einsum('i,i->', v, v),         # dot product: v.v
 np.einsum('ij,j->i', A, v),       # matrix-vector product: Av
 np.einsum('ij,jk->ik', R90, S),   # matrix product: R90 S
 np.einsum('ii->', A))             # trace: tr(A)
```

```
(np.float64(5.0),
 array([ 0., -5.]),
 array([[ 0., -1.],
        [ 1.,  1.]]),
 np.float64(4.0))
```

$\operatorname{tr}\mathbf{A} = 1 + 3 = 4$ を含め、すべての出力が先ほどの手計算と一致する。同じ記法を使えば、アテンションスコアもわずか 1 行で書ける。以下では、これを `einsum` で計算し、転置したキーとのバッチ行列積と一致することを確かめ、積和演算の総数を数える。

```
rng = np.random.default_rng()
queries = rng.standard_normal((2, 3, 4, 8))   # (batch, heads, queries, width)
keys = rng.standard_normal((2, 3, 5, 8))      # (batch, heads, keys, width)
scores = np.einsum('bhij,bhtj->bhit', queries, keys)
(scores.shape, np.allclose(scores, queries @ keys.swapaxes(-1, -2)),
 scores.size * queries.shape[-1])             # multiply-adds
```

```
((2, 3, 4, 5), True, 960)
```

出力の形状は $(2, 3, 4, 5)$ であり、系列とヘッドの組ごとに 4×5 のスコア表が 1 つ得られる。2 つの式の計算結果は一致し、この縮約には $2 \cdot 3 \cdot 4 \cdot 5 \cdot 8 = 960$ 回の積和演算を要する。これは 5 つの添字すべてのサイズを掛け合わせた積にほかならない。

コードだけでなく、添字記法を使えば 1 行の簡潔な証明も難なく書ける。任意の $\mathbf{A} \in \mathbb{R}^{m \times n}$ と $\mathbf{B} \in \mathbb{R}^{n \times m}$ に対して、$\mathbf{AB}$ と $\mathbf{BA}$ はどちらも正方行列であり、それらのトレースは次を満たす。

$$\operatorname{tr}(\mathbf{AB}) = a_{ij} b_{ji} = b_{ji} a_{ij} = \operatorname{tr}(\mathbf{BA}). \tag{33.13}$$

中央の変形は、添字には一切触れずに 2 つのスカラー値の積の順序を入れ替えただけである。驚くべきことに、たとえ $\mathbf{AB} \neq \mathbf{BA}$ であっても、さらには 2 つの積のサイズが異なっていても、積のトレースは 2 つの因子の順序によらない。ここから直ちに導かれる帰結の 1 つは、固有値を学ぶ際に重要になる。$\mathbf{V}$ が逆行列 $\mathbf{V}^{-1}$ を持つとき（33.2.2.3 節）、$\operatorname{tr}(\mathbf{V}\mathbf{A}\mathbf{V}^{-1}) = \operatorname{tr}(\mathbf{A}\mathbf{V}^{-1}\mathbf{V}) = \operatorname{tr}(\mathbf{A})$ が成り立つ。$\mathbf{V}\mathbf{A}\mathbf{V}^{-1}$ は同じ線形変換 $\mathbf{A}$ を別の座標系で表したものであるから、トレースは選んだ座標系に依存する見かけの量ではなく、写像そのものに備わった固有の性質である。後で見るように、行列式もこの不変性を共有している（33.2.3.3 節）。

実用上の注意点を 1 つ挙げておく。`einsum` の式で 3 つ以上のテンソルを組み合わせるとき、文字列が指定するのは数学的な結果だけであり、2 つずつの縮約を実行する順序までは指定しないが、その順序によって計算速度に大きな差が生じることがある。たとえば $n \times n$ 行列では、$(\mathbf{AB})\mathbf{v}$ の計算には約 n^3 回の積和演算がかかるのに対し、$\mathbf{A}(\mathbf{B}\mathbf{v})$ ならわずか $2n^2$ 回で済む。

33.2.2 ランクと可逆性

33.2.2.1 列空間と零空間

行列は実際にどのような出力を生み出せ、どのような入力を消し去ってしまうのだろうか。その両方を一度に見るため、特異行列の例を調べてみよう。

$$\mathbf{B} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}.$$

その列は $\mathbf{b}_1 = (2, 4)$ と $\mathbf{b}_2 = (1, 2)$ であり、一方は他方のちょうど 2 倍になっている（$\mathbf{b}_1 = 2\mathbf{b}_2$）。$\mathbf{B}$ のすべての出力はその列の重み付き和であるから、あらゆる出力は $\mathbf{b}_2$ のスカラー倍へと潰れてしまう。

$$a_1\mathbf{b}_1 + a_2\mathbf{b}_2 = (2a_1 + a_2)\,\mathbf{b}_2.$$

したがって、$\mathbf{B}$ は 2 次元平面全体を、1 本の直線 $y = 2x$（点 $(1, 2)$ を通る直線）の上へと送り込む。必然的に、異なる入力がその直線上で衝突し、$(1, 1)$ と $(0, 3)$ はどちらも $(3, 6)$ に着地する。線形性により、両者の差 $(1, -2)$ はそれぞれの出力の差へと写るはずであり、$\mathbf{B}(1, -2) = \mathbf{0}$ となる。行ごとの見方からも同じ理由が確かめられる。$\mathbf{B}$ の 2 つの行 $(2, 1)$ と $(4, 2)$ はどちらも $(1, -2)$ に垂直であるため、どちらの超平面判定も 0 を返す。Figure 33.11 にこの 2 本の直線を並べて示す。

この例は、すべての $m \times n$ 行列 $\mathbf{A}$ に付随する 2 つの基本部分空間（subspace）を具体的に示している。**列空間**（column space）は $\operatorname{col}\mathbf{A}$ と書き、$\mathbf{A}$ の列が張る空間である。$\mathbf{Ax}$ は $x_1, \dots, x_n$ を重みとする列の重み付き結合であるから、列空間は到達可能なすべての出力の集合にぴったり一致する。**零空間**（null space）は**核**（kernel）とも呼ばれ、$\ker\mathbf{A}$ と書いて、$\mathbf{A}$ が 0 に送るすべての入力の集合を表す。

$$\ker\mathbf{A} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{0}\}.$$

これは $\mathbb{R}^n$ の部分空間である。実際、$\mathbf{Ax} = \mathbf{Ay} = \mathbf{0}$ ならば $\mathbf{A}(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha\mathbf{Ax} + \beta\mathbf{Ay} = \mathbf{0}$ が成り立つ。$\mathbf{B}$ においては、列空間は出力平面内の直線 $y = 2x$ であり、零空間は入力平面内の $(1, -2)$ を通る直線である。（どちらも $\mathbb{R}^2$ に属しているのは $\mathbf{B}$ が正方行列であることによる偶然にすぎず、一般には $\ker\mathbf{A} \subseteq \mathbb{R}^n$ かつ $\operatorname{col}\mathbf{A} \subseteq \mathbb{R}^m$ である。）

この例から、次の 3 つの一般的な事実がそのまま導かれる。

1. **零空間は、すべての行に垂直な入力全体からなる。** 行ごとの見方では、$(\mathbf{Ax})_i$ は第 i 行と $\mathbf{x}$ のドット積であるため、$\mathbf{Ax}$ のすべての成分が 0 になるのは、$\mathbf{x}$ がすべての行に直交するとき、かつそのときに限る。
2. **列が線形従属（linearly dependent）であることは、零空間が非ゼロベクトルを含むことと同値である。** 非自明な従属関係 $\sum_j x_j\mathbf{a}_j = \mathbf{0}$ が列 $\mathbf{a}_j$ の間に成り立つとは、$\mathbf{Ax} = \mathbf{0}$ を満たす非ゼロベクトル $\mathbf{x}$ が存在するということそのものであり、この等式を逆に読めば逆向きも従う。$\mathbf{B}$ では、関係式 $\mathbf{b}_1 - 2\mathbf{b}_2 = \mathbf{0}$ から零空間のベクトル $(1, -2)$ が得られる。
3. **2 つの入力が同じ出力を生むのは、それらの差が零空間のベクトルであるとき、かつそのときに限る。** 実際、$\mathbf{Ax} = \mathbf{Ay}$ は $\mathbf{A}(\mathbf{x} - \mathbf{y}) = \mathbf{0}$ と同値である。$\mathbf{B}$ が $(3, 6)$ に写すすべての入力の集合は、平行移動した直線 $(1, 1) + t(1, -2)$ であり、これは点 $(0, 3)$ を $t = -1$ で通る。$\ker\mathbf{A}$ が非ゼロベクトルを含む限り、異なる入力が衝突し、逆行列は存在しえない。

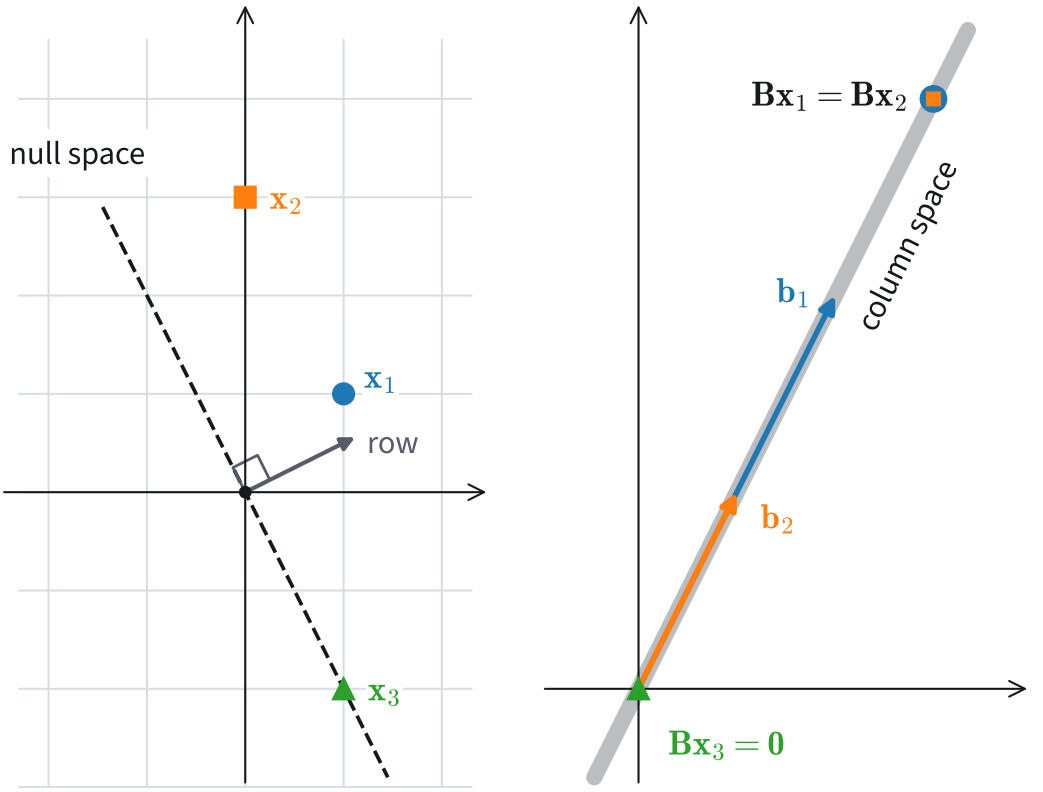


Figure 33.11: 平面を潰す行列 $\mathbf{B} = \left(\begin{smallmatrix} 2 & 1 \\ 4 & 2 \end{smallmatrix}\right)$。(a) 入力平面において、$(1, -2)$ を通る破線の零空間の直線は行の方向 $(2, 1)$ に垂直であり、3 つの入力 $\mathbf{x}_1 = (1, 1)$、$\mathbf{x}_2 = (0, 3)$、および零空間の直線上の $\mathbf{x}_3 = (1, -2)$ が印されている。(b) 出力平面（同じ縮尺）では、あらゆる出力は列空間である直線 $y = 2x$（$\mathbf{b}_1$ と $\mathbf{b}_2$ が張る空間）に着地する。$\mathbf{x}_1$ と $\mathbf{x}_2$ はともに $(3, 6)$ に写り、$\mathbf{x}_3$ は原点に潰れる。異なる入力が同じ像を共有するため、逆行列は存在しえない。

33.2.2.2 ランク

線形変換によって入力の何次元が生き残るだろうか。行列の**ランク**とは、その列空間の次元、言い換えれば線形独立な列の最大本数である。

$$\operatorname{rank}\mathbf{A} = \dim \operatorname{col}\mathbf{A}.$$

行列 $\mathbf{B}$ のランクは 1 である。2 つの列は線形従属であり、どちらもゼロではないからである。もう少し豊かな例として、次の 4×5 行列を考えよう。

$$\mathbf{C} = \begin{bmatrix} 1 & 3 & 0 & -1 & 0 \\ -1 & 0 & 1 & 1 & -1 \\ 0 & 3 & 1 & 0 & -1 \\ 2 & 3 & -1 & -2 & 1 \end{bmatrix},$$

この行列のランクは 2 である。最初の 2 つの列は線形独立であるが、$\binom{5}{3} = 10$ 通りある 3 列の組はどれも線形従属になっている。

注目すべきことに、写像で生き残る次元の数と 0 に潰される次元の数を足し合わせると、常に入力空間の次元に一致する。

命題（ランク・退化次数の定理） $m \times n$ **行列 A に対して、次が成り立つ。**

$$\operatorname{rank}\mathbf{A} + \dim \ker \mathbf{A} = n. \tag{33.14}$$

証明基底 $\mathbf{w}_1, \dots, \mathbf{w}_k$ を $\ker \mathbf{A}$ から選び、それを拡張して基底 $\mathbf{w}_1, \dots, \mathbf{w}_k, \mathbf{u}_1, \dots, \mathbf{u}_{n-k}$（$\mathbb{R}^n$ の基底）を作る。（線形独立なベクトルの組が常に基底へと拡張できること、および部分空間のすべての基底が同じ個数のベクトルを持つことは、標準的な事実として証明なしに用いる。）像 $\mathbf{A}\mathbf{u}_1, \dots, \mathbf{A}\mathbf{u}_{n-k}$ は $\operatorname{col}\mathbf{A}$ の張る空間をなす。なぜなら、任意の入力 $\mathbf{x} \in \mathbb{R}^n$ は基底全体の線形結合で表され、各 $\mathbf{w}_j$ は $\mathbf{A}$ によって 0 に潰れるからである。さらに、これらの像は線形独立でもある。もし $\sum_i c_i \mathbf{A}\mathbf{u}_i = \mathbf{0}$ ならば、$\sum_i c_i \mathbf{u}_i$ は $\ker \mathbf{A}$ に属するため、$\mathbf{w}_1, \dots, \mathbf{w}_k$ の線形結合として書ける。基底の内部では、すべての $c_i = 0$ でない限りそのようなことは起こりえない。ゆえに、$\operatorname{col}\mathbf{A}$ の次元は $n - k$ である。$\blacksquare$

Figure 33.12 に、この証明で構成した基底を示す。n 本のベクトルのそれぞれは、$\mathbf{A}$ によって 0 になるか列空間に 1 次元分寄与するかのいずれかであり、両方に当てはまるベクトルは存在しない。

この次元の保存を具体例で確かめてみよう。$\mathbf{B}$ では $1 + 1 = 2$ となる。冒頭で挙げた Fashion-MNIST の 256×784 層では、ランクは出力座標の個数である 256 を超えられないため、零空間の次元は少なくとも $784 - 256 = 528$ でなければならない。$\mathbf{C}$ では、零空間の次元は $5 - 2 = 3$ となるため、線形独立な 3 本の零空間のベクトルがあれば $\ker \mathbf{C}$ の基底をなす。以下では、$(3, -1, 3, 0, 0)$、$(1, 0, 0, 1, 0)$、$(0, 0, 1, 0, 1)$ がいずれも $\mathbf{C}$ によって 0 に写ることを確かめ、これらを列として並べた行列 `Z` のランクを調べる。

```
B = np.array([[2.0, 1.0], [4.0, 2.0]])
C = np.array([[1, 3, 0, -1, 0], [-1, 0, 1, 1, -1],
              [0, 3, 1, 0, -1], [2, 3, -1, -2, 1]], dtype=float)
Z = np.array([[3, -1, 3, 0, 0], [1, 0, 0, 1, 0],
              [0, 0, 1, 0, 1]], dtype=float).T   # candidate null vectors
rank_C, rank_Z = np.linalg.matrix_rank(C), np.linalg.matrix_rank(Z)
np.linalg.matrix_rank(B), rank_C, np.abs(C @ Z).max(), rank_Z, rank_C + rank_Z
```

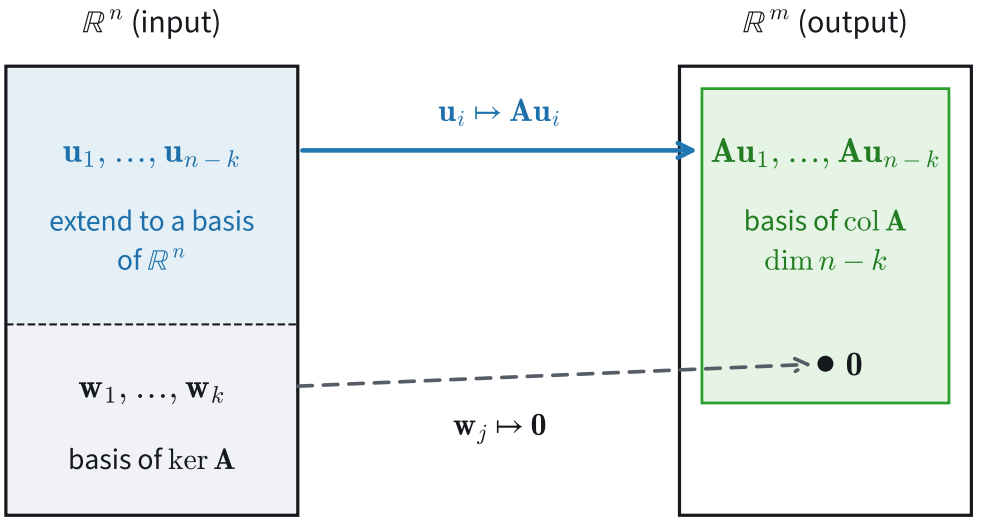


Figure 33.12: ランク・退化次数の定理の証明における基底。k 本のベクトル $\mathbf{w}_1, \ldots, \mathbf{w}_k$ は $\ker \mathbf{A}$ を張る空間をなし、$\mathbf{0}$ に送られる。残る $n-k$ 本のベクトル $\mathbf{u}_1, \ldots, \mathbf{u}_{n-k}$ はこれらを補って $\mathbb{R}^n$ の基底を完成させ、その像が $\operatorname{col} \mathbf{A}$ の基底をなす。

```
(np.int64(1), np.int64(2), np.float64(0.0), np.int64(3), np.int64(5))
```

$\mathbf{B}$ と $\mathbf{C}$ のランクはそれぞれ 1 と 2 になり、$|\mathbf{CZ}|$ の最大成分は 0、Z の 3 つの列のランクは 3 となって、$2 + 3 = 5$ が確認できる。

行列を列ではなく行ごとに読んだ場合はどうだろうか。線形独立な行の最大本数である**行ランク**（row rank）は、その行列が独立な超平面判定をいくつ実行するかを数えている。$\mathbf{C}$ では、第 3 行は第 1 行と第 2 行の和であり、第 4 行は第 1 行と第 2 行の差であるため、行ランクは 2 となって列ランクと一致する。この一致は常にあらゆる行列で成り立つ。

命題（行ランクと列ランクの一致） すべての行列 $\mathbf{A}$ に対して、線形独立な行の最大本数は、線形独立な列の最大本数に等しい。

証明 $r = \operatorname{rank} \mathbf{A}$ とおき、$\operatorname{col} \mathbf{A}$ の基底を $m \times r$ 行列 $\mathbf{G}$ の各列に並べる。$\mathbf{A}$ の各列は $\mathbf{G}$ の列の線形結合であるため、第 j 列（$\mathbf{A}$ の列）に対する係数を第 j 列（$r \times n$ 行列 $\mathbf{H}$ の列）に格納すれば、次の分解が得られる。

$$\mathbf{A} = \mathbf{G}\mathbf{H}. \tag{33.15}$$

今度は同じ積を行ごとに読もう。第 i 行（$\mathbf{A}$ の行）は $\sum_k g_{ik}\,(\text{row } k \text{ of } \mathbf{H})$ となるため、$\mathbf{A}$ のすべての行は r 本の $\mathbf{H}$ の行が張る空間に含まれる。したがって、$\mathbf{A}$ の行ランクは高々 r である。転置 $\mathbf{A}^{\top}$（その列は $\mathbf{A}$ の行である）に対して同じ議論を繰り返せば、列ランクも高々行ランクであることがわかり、両者は一致しなければならない。$\blacksquare$

分解 $\mathbf{A} = \mathbf{G}\mathbf{H}$ は単なる証明の道具にとどまらない。ここからは、ランク r のあらゆる行列が幅 r のボトルネックに入力を通していることが見えてくる。すなわち、$\mathbf{H}$ が $\mathbf{x} \in \mathbb{R}^n$ を r 個の数値へと圧縮し、$\mathbf{G}$ がその r 個の数値を再び $\mathbb{R}^m$ へと展開する。r が m や n よりはるかに小さいとき、

G と **H** の格納に必要なパラメータ数はわずか $r(m+n)$ 個で済み、mn 個を保持する必要はない。これが 33.5.4.1 節で扱う低ランク更新の中心的な着想となる。

実際には、列の部分集合を片端から調べてランクを求めようとすれば指数関数的な時間がかかってしまう。手計算ではガウスの消去法を用いる（本章では扱わない）。一方、`np.linalg.matrix_rank` のような数値計算ライブラリは、微小な許容誤差を超える**特異値**（singular value）（33.5 節）がいくつあるかを数える。ランクは行列の成分の連続関数ではないため、許容誤差の導入は避けられない。ごくわずかな摂動でも、ランク落ちの状態からフルランクへと不連続に跳ね上がってしまうからである（33.2.3.6 節）。

33.2.2.3 逆行列

出力 **Ax** が与えられたとき、いつ入力 **x** を確実に復元できるだろうか。正方行列 **A** の**逆行列**（inverse）とは、行列 $\mathbf{A}^{-1}$ であって両側から **A** の作用を打ち消すものである。

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I},$$

これにより、$\mathbf{x} = \mathbf{A}^{-1}(\mathbf{A}\mathbf{x})$ がすべての入力 **x** に対して成り立つ。2×2 行列（成分 p, q, r, s が $ps - qr \neq 0$ を満たすもの）には、明示的な公式がある。

$$\mathbf{A}^{-1} = \frac{1}{ps-qr}\begin{bmatrix} s & -q \\ -r & p \end{bmatrix}. \tag{33.16}$$

実際に掛けてみればすぐに確かめられる。$\left(\begin{smallmatrix} s & -q \\ -r & p \end{smallmatrix}\right)$ と **A** をどちらの順序で掛けても $(ps-qr)\,\mathbf{I}$ になり、$ps-qr$ で割れば **I** が残る。条件 $ps-qr \neq 0$ を外すことはできない。もし $ps-qr = 0$ ならば、この積はゼロ行列になるため、仮に逆行列 $\mathbf{A}^{-1}$ が存在して両辺に掛けられたとすると $\left(\begin{smallmatrix} s & -q \\ -r & p \end{smallmatrix}\right) = \mathbf{0}$、ひいては $\mathbf{A} = \mathbf{0}$ となってしまうが、ゼロ行列が逆行列を持たないことは明らかである。量 $ps-qr$ には、33.2.3.1 節で符号付き面積として再び出会う。通しで用いる行列では $ps - qr = 3 + 2 = 5$ であるから、次のようになる。

$$\mathbf{A}^{-1} = \frac{1}{5}\begin{bmatrix} 3 & -2 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{A}^{-1}(0,-5) = \tfrac{1}{5}\,(10,-5) = (2,-1).$$

確かに、$\mathbf{A}^{-1}$ は 33.2.1.1 節の計算例を逆向きに戻している。

逆行列が存在するとき、それは常に一意である。さらに都合のよいことに、同じ行列の左逆行列と右逆行列は必ず一致する。実際、$\mathbf{L}\mathbf{A} = \mathbf{I}$ かつ $\mathbf{A}\mathbf{X} = \mathbf{I}$ ならば、結合法則により次が成り立つ。

$$\mathbf{L} = \mathbf{L}(\mathbf{A}\mathbf{X}) = (\mathbf{L}\mathbf{A})\mathbf{X} = \mathbf{X}.$$

一般には、いつ逆行列が存在するのだろうか。正方行列の場合、その答えには零空間とランク・退化次数の定理しか必要なく、本章の残りの部分は本節の他のどの結果よりもこの定理に依拠する。

命題（可逆行列） $n \times n$ **行列 A に対して、次の条件はすべて同値である。(i) A は可逆である。(ii) A の列は線形独立である。(iii)** $\ker \mathbf{A} = \{\mathbf{0}\}$ **である。(iv)** $\operatorname{rank} \mathbf{A} = n$ **である。**

証明 条件 (ii) と (iii) は 33.2.2.1 節の事実 2 により同値であり、(iii) と (iv) はランク・退化次数の定理により同値である。**A** が可逆（invertible）であり $\mathbf{A}\mathbf{x} = \mathbf{0}$ を満たすとき、左から $\mathbf{A}^{-1}$ を掛ければ $\mathbf{x} = \mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{0}$ となるため、(i) から (iii) が従う。

逆向きを示すには、実際に逆行列を構成しなければならないためもう少し手間がかかる。(iv) より、列空間は n 次元の $\mathbb{R}^n$ の部分空間であり、これは $\mathbb{R}^n$ 全体に一致しなければならない。もしその外にあるベクトルを $\mathbf{col}\,\mathbf{A}$ の基底に付け加えれば、$n+1$ 本の線形独立なベクトルが $\mathbb{R}^n$ 内に生じることになり矛盾するからである。ゆえに、$\mathbb{R}^n$ 内のあらゆる目標ベクトルに到達できる。また (iii) と事実 3 より、異なる 2 つの入力が同じ出力に写ることはない。したがって、第 j 列（行列 $\mathbf{X}$ の列）を $\mathbf{A}\mathbf{x} = \mathbf{e}_j$ の一意な解として定義できる。構成法から直ちに $\mathbf{A}\mathbf{X} = \mathbf{I}$ が成り立つ。$\mathbf{X}$ が左からも逆行列として働くことを確かめるため、任意の $\mathbf{x} \in \mathbb{R}^n$ をとろう。$\mathbf{A}(\mathbf{X}\mathbf{A}\mathbf{x}) = (\mathbf{A}\mathbf{X})\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}$ であり、異なる 2 つの入力は衝突しないから、$\mathbf{X}\mathbf{A}\mathbf{x} = \mathbf{x}$ でなければならない。ゆえに $\mathbf{X}\mathbf{A} = \mathbf{I}$ も成り立ち、$\mathbf{X} = \mathbf{A}^{-1}$ となる。$\blacksquare$

これにより、33.2.1.3 節で証明を先送りにしていた主張が直ちに解決する。正方の直交行列 $\mathbf{Q}$（$\mathbf{Q}^{\top}\mathbf{Q} = \mathbf{I}$ を満たす）に対して、もし $\mathbf{Q}\mathbf{c} = \mathbf{0}$ ならば $\mathbf{c} = \mathbf{Q}^{\top}\mathbf{Q}\mathbf{c} = \mathbf{0}$ となるため、$\ker \mathbf{Q} = \{\mathbf{0}\}$ である。上の命題より $\mathbf{Q}$ は可逆であり、その左逆行列である転置 $\mathbf{Q}^{\top}$ は $\mathbf{Q}^{-1}$ に一致しなければならず、したがって $\mathbf{Q}\mathbf{Q}^{\top} = \mathbf{I}$ が成り立つ。

逆行列を持たない正方行列は**特異**（singular）であるといい、$\mathbf{B}$ がその典型例である（特異行列とも呼ぶ）。一方、長方形の行列はそもそも両側逆行列を決して持たない。$m \times n$ 行列のランクは高々 $\min(m, n)$ である。$m > n$ のとき、列空間の次元は高々 $n < m$ であるため写像は**全射**（onto）ではなく、$\mathbb{R}^m$ 内に決して到達できない目標が生じる。$n > m$ のとき、ランク・退化次数の定理により少なくとも次元 $n - m > 0$ の零空間が生じるため写像は**単射**（one-to-one）ではなく、異なる入力が衝突する。写像を逆向きにたどるには両方の性質が不可欠であり、上の命題の美しさは、正方行列においてはどちらか一方さえ成り立てばもう一方も保証される点にある。

33.2.2.4 連立一次方程式の実践的な解法

逆行列は代数計算や証明には欠かせない道具であるが、数値計算において逆行列を明示的に作ることはまずない。$\mathbf{A}\mathbf{x} = \mathbf{b}$ を解く必要があるときは、`np.linalg.solve(A, b)` を用い、`np.linalg.inv(A) @ b` は避けるべきである。

理由は大きく 2 つ、計算速度と計算精度にある。第 1 に、`solve` は $\mathbf{A}$ を三角行列の積へと 1 回だけ分解し（**LU 分解**（LU factorization）、コストは約 $n^3/3$ 回の積和演算）、高速に解ける 2 つの三角連立一次方程式を解く。これに対し、$\mathbf{A}^{-1}$ を明示的に求めると約 n^3 回の積和演算がかかり、$\mathbf{b}$ を掛ける前の段階ですでに 3 倍の計算量を要する。第 2 に、`solve` は**後退安定**（backward stable）である。すなわち、計算された解は、$\mathbf{A}$ と $\mathbf{b}$ の成分を浮動小数点の丸め誤差レベルだけわずかに摂動させた連立一次方程式の厳密解になっている。その結果、残差 $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|$ は計算機精度と $\|\mathbf{A}\|\,\|\mathbf{x}\|$（1.3.4 節のスペクトルノルムを使用）の積のオーダーに収まる。一方、`inv(A) @ b` の計算にはそのような保証がなく、残差は最大で**条件数**（condition number）$\kappa(\mathbf{A})$ 倍まで膨らみうる。条件数とは、$\mathbf{A}$ が特異行列にどれほど近いかを測る指標である（33.5.3.2 節）。

行列が特異性にどれほど近いかは問題そのものの性質であり、アルゴリズムの性質ではない。倍精度（10 進で約 16 桁）では、どちらの手法でも解 $\mathbf{x}$ の精度が最大で約 $\log_{10}\kappa(\mathbf{A})$ 桁失われうる。これは最悪ケースの上界であって個々の右辺に対する厳密な予測ではないため、悪名高い具体例がこの限界にどれほど迫るかを見てみよう。成分 $h_{ij} = 1/(i+j-1)$ を持つ**ヒルベルト行列**（Hilbert matrix）は、特異に極めて近い行列の古典的なテストケースである。$\mathbf{x}_{\text{true}} = \mathbf{1}$ とし、$\mathbf{b} = \mathbf{H}\mathbf{1}$ を 10×10 ヒルベルト行列 $\mathbf{H}$ に対して構成し、両方の方法で $\mathbf{x}$ の復元を試みる。

```
H = 1 / (np.arange(10)[:, None] + np.arange(10) + 1)   # Hilbert matrix
x_true = np.ones(10)
b = H @ x_true
for x in (np.linalg.solve(H, b), np.linalg.inv(H) @ b):
    print(f'residual {np.linalg.norm(H @ x - b):.1e}, '
```

```
            f'error {np.linalg.norm(x - x_true):.1e}')
np.linalg.cond(H)
```

```
residual 5.7e-16, error 2.7e-04
residual 4.8e-05, error 9.0e-03
```

```
np.float64(16024416987428.36)
```

条件数は $\kappa(\mathbf{H}) \approx 1.6 \cdot 10^{13}$ であり、最大で 13 桁が失われうることを警告している。予告どおり、`solve` は残差を計算機精度付近に保つのに対し、`inv` の残差は約 11 桁も大きくなり、因子 κ から予想される結果とよく一致する。ただし、残差が微小だからといって精度の高い解が保証されるわけではない点に注意しよう。どちらの方法でも 16 桁のうち 12 桁以上が失われて最悪ケースの上界に迫り、その中でも `inv` はさらに 1 桁以上悪い誤差を示す。

スパース行列は、決して逆行列を求めてはならない 3 つ目の決定的な理由をもたらす。非ゼロ成分が 500 万個しかない 100 万行 100 万列の行列はメモリに余裕で収まるが、その逆行列は通常完全に密になり、約 10^{12} 個もの成分を要してしまう。

33.2.3 行列式

33.2.3.1 平行四辺形の面積

行列は面積を何倍に拡大・縮小するだろうか。Figure 33.8 を振り返ってみよう。通しで用いる行列 $\mathbf{A}$ は、単位正方形（辺 $\mathbf{e}_1$ と $\mathbf{e}_2$、面積 1）を、辺 $\mathbf{a}_1 = (1, -1)$ と $\mathbf{a}_2 = (2, 3)$、頂点 $(0, 0)$、$(1, -1)$、$(3, 2)$、$(2, 3)$ を持つ平行四辺形へと送る。この平行四辺形は、バウンディングボックス $[0, 3] \times [-1, 3]$（面積 12）の内部にぴったり収まっている。ボックスの余った部分は、直角を挟む 2 辺がそれぞれ 1 と 1、2 と 3、1 と 1、2 と 3 である 4 つの直角三角形からなり、その面積の合計は $\frac{1}{2} + 3 + \frac{1}{2} + 3 = 7$ である。これを差し引くと面積 $12 - 7 = 5$ が残る（Figure 33.13 のパネル (a)）。すなわち、行列 $\mathbf{A}$ は単位正方形の面積を 5 倍にする。

三角形を切り出す方法は 1 つの図に対しては有効だが、頂点の配置が変わるたびに新しい図を描き直す必要がある。一方、「底辺と高さの積」を使えばあらゆる場合を一挙に扱える。平行四辺形の辺を $\mathbf{u} = (p, r)$ と $\mathbf{w} = (q, s)$、すなわち一般の 2×2 行列の 2 つの列としよう。底辺の長さは $\|\mathbf{u}\|$ である。$\mathbf{u}$ を反時計回りに 90° 回転させると $\mathbf{R}_{90}\mathbf{u} = (-r, p)$ になるため、次のベクトルを考える。

$$\mathbf{n} = \frac{(-r, p)}{\|\mathbf{u}\|}$$

これは底辺に垂直な単位ベクトルである。33.1 節で見たとおり、$\mathbf{w}$ の単位ベクトル $\mathbf{n}$ 方向の符号付き成分はドット積 $\mathbf{w} \cdot \mathbf{n}$ で与えられる。幾何学的には、これは底辺から測った平行四辺形の高さに符号を付けたものであり、$\mathbf{w}$ が $\mathbf{u}$ から見て反時計回りの側にあるときは正、時計回りの側にあるときは負になる。底辺に符号付き高さを掛けると、次の式が得られる。

$$\|\mathbf{u}\| \, (\mathbf{w} \cdot \mathbf{n}) = \|\mathbf{u}\| \, \frac{-qr + ps}{\|\mathbf{u}\|} = ps - qr.$$

都合のよいことに長さ $\|\mathbf{u}\|$ が約分されて消え、符号による場合分けを一切必要としない、行列の 4 成分に関する簡潔な多項式だけが残る。通しで用いる行列では $\|\mathbf{u}\| = \sqrt{2}$、$\mathbf{n} = (1, 1)/\sqrt{2}$、$\mathbf{w} \cdot \mathbf{n} = 5/\sqrt{2}$ となり、符号付き面積は 5 となる（パネル (b)）。これが正になるのは、$\mathbf{a}_2$ が $\mathbf{a}_1$ から見て反時計回りの側にあるからである。

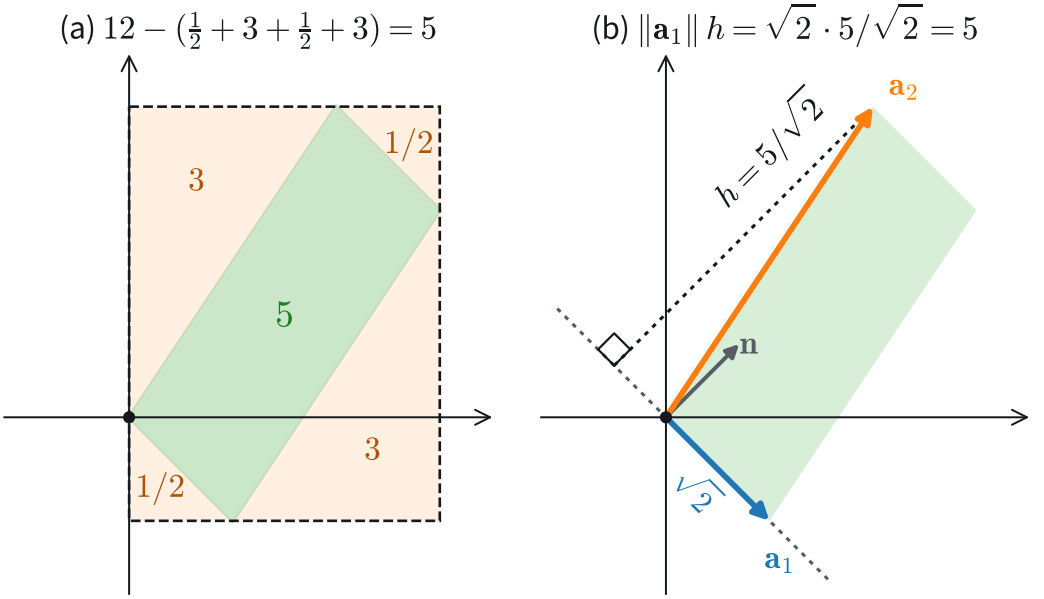


Figure 33.13: $\mathbf{A}$ による単位正方形の像の面積は 5 である。(a) バウンディングボックスによる計算。$[0,3] \times [-1,3]$ のボックス（面積 12）から、合計面積 7 の 4 つの直角三角形を引く。(b) 底辺と高さの積。底辺 $\mathbf{a}_1 = (1,-1)$ の長さは $\sqrt{2}$ であり、単位法線ベクトル $\mathbf{n} = (1,1)/\sqrt{2}$ はこれに垂直で、底辺を含む直線から測った $\mathbf{a}_2 = (2,3)$ の高さは $\mathbf{a}_2 \cdot \mathbf{n} = 5/\sqrt{2}$ である。

2×2 行列の**行列式**を、この単位正方形の像の符号付き面積として定義する。

$$\det \begin{bmatrix} p & q \\ r & s \end{bmatrix} = ps - qr. \tag{33.17}$$

その絶対値 $|\det \mathbf{A}|$ は写像が面積を何倍にするかを表し、その符号は $\mathbf{e}_1$ と $\mathbf{e}_2$ の像が反時計回りの向き（orientation）を保っているかどうかを記録する。$\mathbf{A}$ に対しては $1 \cdot 3 - 2 \cdot (-1) = 5$ となる。$\mathbf{A}$ の 2 つの列を入れ替えると $\left(\begin{smallmatrix} 2 & 1 \\ 3 & -1 \end{smallmatrix}\right)$ が得られ、辺を並べる順序が逆になっただけで張る空間としての平行四辺形はまったく同じだが、行列式の符号は反転して $-2 - 3 = -5$ になる。特異行列 $\mathbf{B}$ に対しては $2 \cdot 2 - 1 \cdot 4 = 0$ となり、単位正方形は $\mathbf{0}$ から $(3,6)$ までの面積 0 の線分へと平坦に潰れる。

Figure 33.14 はこれら 3 つの類型を対比している。正の行列式は向きを保ち、負の行列式は向きを反転させ、0 の行列式は正方形を潰す。

「底辺と高さの積」によって平面の場合は解決した。n 次元へ進むには、n 次元の箱の符号付き体積が必要になる。

33.2.3.2 一般の行列式

n 本の列が $\mathbb{R}^n$ 内で張る箱の符号付き体積は、どのように定義すればよいだろうか。図形を切り分けて高次元の体積を計算しようとする代わりに、ここではより見通しのよい道をとる。符号付き体積が満たすべき 3 つの幾何学的性質を書き下し、それらの性質から公式を一意に決定させるのである。列 $\mathbf{a}_1, \dots, \mathbf{a}_n$（$n \times n$ 行列の各列）の関数として見たとき、行列式は次の性質を持つ。

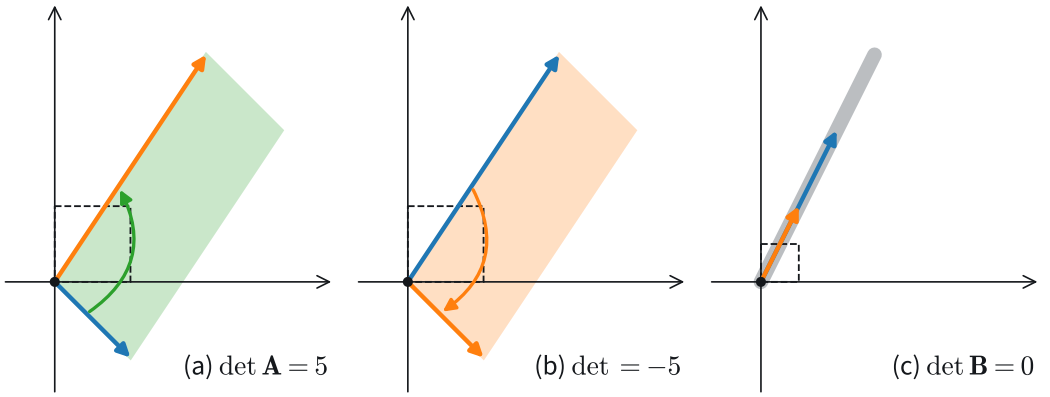


Figure 33.14: 3 つの 2×2 の例における符号付き面積としての行列式。単位正方形（破線）は各列が張る領域へと写る。(a) $\mathbf{A}$（行列式 5）。列は反時計回りの順序（曲線矢印）を保つ。(b) 列を入れ替えた $\mathbf{A}$（行列式 -5）。平行四辺形は同じだが向きが反転している。(c) $\mathbf{B}$（行列式 0）。正方形は直線 $y = 2x$ 上の線分へと潰れる。パネル (c) は (a) と (b) の半分の縮尺で描かれている。

1. **多重線形**（multilinear）：他の列を固定したとき、各列について個別に線形である。（箱の 1 つの辺をスカラー倍すると体積も同じ倍率で変化し、1 つの辺を和として表すと体積もそれに応じて分割される。）
2. **交代的**（alternating）：2 つの列が一致するときは常に 0 に等しい。（2 つの辺が重なると、箱は体積 0 の低次元の板へと潰れる。）
3. **正規化**（normalized）：$\det \mathbf{I} = 1$ である。（単位超立方体の体積は 1 である。）

2×2 の公式 $ps - qr$ がこれら 3 つをすべて満たすことを確かめよう。(p, r) と (q, s) のそれぞれについて線形であり、$(p, r) = (q, s)$ のときは常に 0 になり、さらに値 1 を $\mathbf{I}$ に対してとる。これら 3 つの公理だけから、2 つの強力な帰結が直ちに導かれる。列を入れ替えたときの規則と、3 つの公理をすべて満たす関数が高々 1 つしか存在しないことを示す明示公式である。

第 1 に、**任意の 2 つの列を入れ替えると行列式の符号が反転する**。理由を見るため、2 つの列の位置に $\mathbf{u} + \mathbf{w}$ を置いて多重線形性により展開しよう。

$$\begin{aligned} 0 &= \det(\dots, \mathbf{u} + \mathbf{w}, \dots, \mathbf{u} + \mathbf{w}, \dots) \\ &= \det(\dots, \mathbf{u}, \dots, \mathbf{u}, \dots) + \det(\dots, \mathbf{u}, \dots, \mathbf{w}, \dots) \\ &\quad + \det(\dots, \mathbf{w}, \dots, \mathbf{u}, \dots) + \det(\dots, \mathbf{w}, \dots, \mathbf{w}, \dots). \end{aligned}$$

左辺、第 1 項、および最後の項はいずれも一致する 2 つの列を持つため、交代性により 0 になる。残るのは次の式である。

$$\det(\dots, \mathbf{u}, \dots, \mathbf{w}, \dots) = -\det(\dots, \mathbf{w}, \dots, \mathbf{u}, \dots).$$

箱の 2 つの辺を入れ替えると、Figure 33.14 のパネル (b) で見たのと同様に、箱の向きが反転する。

第 2 に、これら 3 つの性質から公式が完全に定まる。各列を標準基底ベクトルで $\mathbf{a}_j = \sum_i a_{ij}\, \mathbf{e}_i$ と展開し、1 列ずつ多重線形性を適用しよう。すると $\det \mathbf{A}$ は、$a_{i_1 1} a_{i_2 2} \cdots a_{i_n n}\, \det(\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_n})$ という項の大きな和になり、行添字 i_j を各列 j で選ぶすべての組み合わせごとに 1 項ずつ現れ

る。選ばれた行添字のうち 2 つでも等しければ、交代性によりその項は 0 になる。n 個の行添字がすべて異なるとき、並び $\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_n}$ は $\mathbf{I}$ の列を並べ替えたものにすぎないため、列の入れ替えを繰り返すことで $\det(\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_n})$ は ± 1 に帰着する。

2×2 の場合、列 $\mathbf{a}_1 = p\,\mathbf{e}_1 + r\,\mathbf{e}_2$ と $\mathbf{a}_2 = q\,\mathbf{e}_1 + s\,\mathbf{e}_2$ に対して展開には 4 つの項が現れる。同じ基底ベクトルが重複する 2 項は消え、次の式が残る。

$$\det(\mathbf{a}_1, \mathbf{a}_2) = ps\,\det(\mathbf{e}_1, \mathbf{e}_2) + rq\,\det(\mathbf{e}_2, \mathbf{e}_1) = ps - qr,$$

こうして手計算で導いた符号付き面積が再現される。一般に、$\det \mathbf{A}$ はすべての行とすべての列から成分を 1 つずつ選ぶあらゆる方法にわたる符号付き和になる。$\sigma(j)$ で第 j 列において選ばれた行を表せば、次の**置換公式**（permutation formula）が得られる。

$$\det \mathbf{A} = \sum_{\sigma} \operatorname{sgn}(\sigma)\, a_{\sigma(1)1}\, a_{\sigma(2)2} \cdots a_{\sigma(n)n}, \tag{33.18}$$

ここで、σ は $n!$ 通りの $\{1, \dots, n\}$ のすべての置換（permutation）を動き、$\operatorname{sgn}(\sigma) = \det(\mathbf{e}_{\sigma(1)}, \dots, \mathbf{e}_{\sigma(n)})$ は $+1$（σ が偶数回の入れ替えを要するとき）または -1（奇数回の入れ替えを要するとき）をとる。3 つの公理を満たすいかなる関数もこの公式に一致しなければならないため、行列式は高々 1 つしか存在しえない。

ここでは存在の完全な証明には立ち入らない。微妙な点は $\operatorname{sgn}(\sigma)$ が一意に定まること（いかなる置換も偶数回の入れ替えと奇数回の入れ替えの両方では表せないこと）であるが、これは標準的な事実として受け入れる (Horn and Johnson 2012)。この偶奇性の事実さえ認めれば、上の公式は 3 つの公理をすべて満たす。すなわち、各項は各列から成分を 1 つずつ含むため多重線形性が成り立ち、$\mathbf{I}$ に対しては恒等置換のみが生き残るため正規化が成り立ち、交代性も同じ偶奇性の議論から従う（証明は省略する）。

正規化の条件が導出の一番最後にしか使われなかったことに注目しよう。この着眼点から、次の項で用いるより鋭い道具が得られる。もし f が列に関する**任意の**多重線形かつ交代的な関数であれば、同じ列の入れ替え規則と展開がそのまま適用でき、すべての因子 $f(\mathbf{e}_{\sigma(1)}, \dots, \mathbf{e}_{\sigma(n)})$ が $\operatorname{sgn}(\sigma)\, f(\mathbf{I})$ へと変わる。その結果、次の関係が成り立つ。

$$f(\mathbf{A}) = f(\mathbf{I})\,\det \mathbf{A}.$$

要するに、列に関するすべての多重線形かつ交代的な関数は、行列式のスカラー倍にすぎない。

小さな行列の手計算では、Equation 33.18 の各項を第 1 列で選ばれた行ごとにまとめることで**余因子展開**（cofactor expansion）が得られる。すなわち、各成分 a_{i1} に、$(n-1) \times (n-1)$ 小行列（第 i 行と第 1 列を除いて作られる行列）の行列式と交代符号 $(-1)^{i+1}$ を掛け合わせる。次の 3×3 行列に対して、

$$\mathbf{M} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix},$$

第 1 列に沿って展開すると次のようになる。

$$\begin{aligned}\det \mathbf{M} &= 2 \det \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} - 1 \det \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} + 0 \det \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \\ &= 2 \cdot 5 - 1 \cdot 2 + 0 = 8.\end{aligned} \tag{33.19}$$

したがって、$\mathbf{M}$ は $\mathbb{R}^3$ 内の単位立方体を体積 8 の箱へと写す。

三角行列では、行列式は劇的に簡単になり、単に対角成分の積になる。行列式を計算するあらゆる規則の中で、真っ先に覚えておくべきなのはこの規則である。$\mathbf{U}$ を上三角行列とし、$u_{ij} = 0$ が $i > j$ のとき常に成り立つとしよう。置換公式において非ゼロの項を得るには、すべての列から成分を 1 つずつ、すべて異なる行から選ばなければならない。第 1 列で非ゼロになりうる成分は第 1 行にしかない。第 2 列では第 1 行がすでに使われており、第 $3, \dots, n$ 行は 0 であるため、必然的に第 2 行を選ぶことになる。同様に列を順に進めていくと、対角成分を選ぶ恒等置換だけが生き残り、入れ替えの回数は 0 回であるため、次が成り立つ。

$$\det \mathbf{U} = u_{11} u_{22} \cdots u_{nn}.$$

第 n 列から逆向きに同じ議論をたどれば、下三角行列の場合も示せる。さらに嬉しいことに、この規則はブロック行列へも拡張できる。**ブロック上三角**（block upper triangular）行列 $\left(\begin{smallmatrix} \mathbf{D}_1 & * \\ \mathbf{0} & \mathbf{D}_2 \end{smallmatrix} \right)$ を考えよう。ただし、正方の対角ブロック $\mathbf{D}_1$ と $\mathbf{D}_2$ 自体は三角行列でなくてもよい。非ゼロとなる置換の項は、$\mathbf{D}_1$ の列における成分を $\mathbf{D}_1$ の行から選ばなければならない。その下にある成分はすべて 0 だからである。これにより $\mathbf{D}_1$ の行がすべて使い果たされるため、残りの列は $\mathbf{D}_2$ の行から成分を選ぶしかなく、非対角ブロック $*$ は一切寄与しない。生き残るすべての置換は 2 つのブロック上の独立な置換の組へと分解され、符号も積になるため、和はきれいに因数分解される。

$$\det \begin{bmatrix} \mathbf{D}_1 & * \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} = \det \mathbf{D}_1 \, \det \mathbf{D}_2.$$

Figure 33.15 に、それぞれの場合で生き残る成分を示す。

これら 2 つの規則は手計算だけでなく、現代の機械学習においても不可欠である。正規化フロー（normalizing flow）は単純な確率変数を可逆なニューラルネットワークで変換するモデルであり、各ステップでネットワークのヤコビ行列の対数行列式（log-determinant）を計算しなければならない。変数変換によって確率密度が局所的な体積拡大率で割られるからである（36.2.3.1 節）。そのためフローは、各層が三角またはブロック三角のヤコビ行列を持つように設計される。すると対数行列式は n 個のスカラーの対数の和に帰着し、一般の行列で必要となる約 $n^3/3$ 回の積和演算を回避できる。

もう 1 つの有用な対称性として、転置しても行列式は変わらない（$\det \mathbf{A}^\top = \det \mathbf{A}$）。$\mathbf{A}^\top$ に対する置換公式では、各項は $\operatorname{sgn}(\sigma) \prod_j a_{j\sigma(j)}$ である。ここで $j = \sigma^{-1}(i)$ とおくと積は $\prod_i a_{\sigma^{-1}(i)\, i}$ と書き直せ、これは $\mathbf{A}$ において逆置換 σ^{-1} に対応する項に一致する。一連の入れ替えを元に戻すにはまったく同じ回数の入れ替えを要するため、$\operatorname{sgn}(\sigma^{-1}) = \operatorname{sgn}(\sigma)$ が成り立つ。したがって 2 つの和はまったく同じ項を含み、行列式の列に関するあらゆる命題が行に対してもそのまま成り立つことがわかる。

$n!$ 個もの項を持つ置換公式はあくまで理論上の道具であり、ごく小さな行列を除いて数値計算に使われることは決してない。数値計算ライブラリは（`solve` と同様に）行列を三角行列の因子へと分解し、行の入れ替えに伴う符号を調整したうえで対角成分を掛け合わせることで、約 $n^3/3$ 回の積和演算というコストで行列式を計算する。

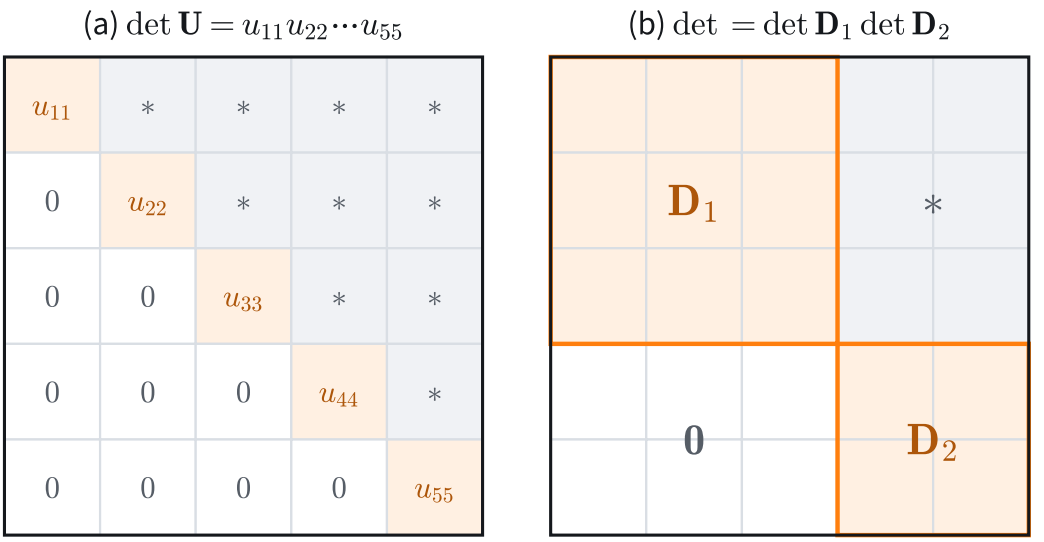


Figure 33.15: 行列式に寄与する成分。(a) 上三角行列では、対角線より下の 0 によって対角成分を選ぶ置換以外がすべて除外されるため、行列式は対角成分の積になる。(b) ブロック上三角行列では、ゼロブロックによって生き残るすべての置換が対角ブロック $\mathbf{D}_1$ と $\mathbf{D}_2$ に閉じ込められる。$*$ と記された成分は一切寄与しない。

33.2.3.3 乗法性

B が面積を $\det \mathbf{B}$ 倍にし、**A** が面積を $\det \mathbf{A}$ 倍にするならば、**B** に続けて **A** を適用すると面積は 2 つの倍率の積倍になるはずである。2 つの例で確かめてみよう。通しで用いる行列では $\mathbf{A}^2 = \left(\begin{smallmatrix} -1 & 8 \\ -4 & 7 \end{smallmatrix}\right)$ であり、$\det \mathbf{A}^2 = -7 + 32 = 25 = 5^2$ となる。また、33.2.1.2 節の合成写像では $\det(\mathbf{R}_{90}\mathbf{T}) = 0 \cdot 1 - (-1) \cdot 1 = 1$ となり、$\det \mathbf{R}_{90} \cdot \det \mathbf{T} = 1 \cdot 1$ と一致する。

なぜこれが一般に成り立つかは、簡単なタイル貼りの図から納得できる。任意の平面領域を微小な正方形の細かいグリッドで覆ってみよう。線形変換は空間全体に一様に作用するため、どの正方形も元の面積の $|\det \mathbf{A}|$ 倍の面積を持つ平行四辺形へと運ばれる。その結果、線形変換は**あらゆる**図形の面積を $|\det \mathbf{A}|$ 倍にする。**B** を面積 V の領域に適用すると面積は $|\det \mathbf{B}|\,V$ になり、続けて **A** を適用すると $|\det \mathbf{A}|\,|\det \mathbf{B}|\,V$ にスケーリングされる。合成 **AB** 自体も 1 つの線形変換であるから、その拡大率はこの積に一致しなければならない。33.2.3.2 節の特徴づけを用いれば、この直感を符号まで含めた 3 行の厳密な証明へと仕立てられる。

命題（行列式の乗法性） 同じサイズの正方行列 A と B に対して、次が成り立つ。

$$\det(\mathbf{AB}) = \det(\mathbf{A})\,\det(\mathbf{B}). \tag{33.20}$$

証明 **A** を固定し、$D(\mathbf{B}) = \det(\mathbf{AB})$ を行列 **B** の列の関数とみなす。33.2.1.2 節より、第 j 列（**AB** の列）は $\mathbf{Ab}_j$ である。**A** は線形であり det は多重線形であるから、D は **B** の列について多重線形である。また、**B** の 2 つの列が一致すれば **AB** の対応する列も一致するため、$D(\mathbf{B}) = 0$ となって D は交代的である。列に関するすべての多重線形かつ交代的な関数は行列式のスカラー倍であるから（33.2.3.2 節）、次が得られる。

$$\det(\mathbf{AB}) = D(\mathbf{B}) = D(\mathbf{I})\,\det(\mathbf{B}) = \det(\mathbf{A})\,\det(\mathbf{B}). \qquad \blacksquare$$

Equation 33.20 から導かれる次の 4 つの帰結は、本章の残りの部分で繰り返し現れる。

1. **行列が可換でなくとも行列式は可換である。** 行列式は通常のスカラーであるため、$\det(\mathbf{AB}) = \det\mathbf{A}\,\det\mathbf{B} = \det\mathbf{B}\,\det\mathbf{A} = \det(\mathbf{BA})$ が成り立つ。回転とせん断の例では $\mathbf{R}_{90}\mathbf{T} \neq \mathbf{T}\mathbf{R}_{90}$ であるが、どちらの積も行列式は 1 である。

2. **相似不変性**（similarity invariance）。任意の可逆行列 $\mathbf{V}$ に対し、帰結 1 を因子 $\mathbf{VA}$ と $\mathbf{V}^{-1}$ に適用すると、$\det(\mathbf{VAV}^{-1}) = \det(\mathbf{V}^{-1}\mathbf{VA}) = \det\mathbf{A}$ が得られる。行列 $\mathbf{VAV}^{-1}$ は、写像 $\mathbf{A}$ を $\mathbf{V}^{-1}$ の各列を座標軸とする座標系で表したものにすぎず、座標系を取り替えても体積の拡大率は変わらない。

3. **逆行列。** $\mathbf{A}$ が可逆ならば、$\mathbf{B} = \mathbf{A}^{-1}$ とおくことで $\det\mathbf{A}\,\det\mathbf{A}^{-1} = \det\mathbf{I} = 1$ が得られる。したがって、すべての可逆行列は $\det\mathbf{A} \neq 0$ かつ $\det\mathbf{A}^{-1} = 1/\det\mathbf{A}$ を満たす（通しで用いる行列では $1/5 = 0.2$）。

4. **直交行列。** 乗法性と $\det\mathbf{Q}^{\top} = \det\mathbf{Q}$ を $\mathbf{Q}^{\top}\mathbf{Q} = \mathbf{I}$ に組み合わせると次が得られる。

$$\det(\mathbf{Q})^2 = \det(\mathbf{Q}^{\top})\det(\mathbf{Q}) = \det(\mathbf{Q}^{\top}\mathbf{Q}) = \det(\mathbf{I}) = 1,$$

   したがって、$\det\mathbf{Q} = \pm 1$ となる。剛体運動は決して体積を変えず、平面においてはその符号によって向きを保つ回転（$\det = +1$）と向きを反転させる鏡映（$\det = -1$）が区別される（33.2.1.3 節）。

33.2.3.4 可逆性と行列式

行列式は、33.2.2.3 節の可逆性定理に 5 つ目の同値条件を加えるものであり、その中でも唯一、単一の数値で表される条件である。2 次元ではこの同値性は直ちにわかる。符号付き面積 $ps - qr$ が 0 になるのは、2 つの列が原点を通る同一の直線上にある（または一方が $\mathbf{0}$ である）とき、すなわち列が線形従属であるとき、かつそのときに限る。同じことがすべての次元で成り立つ。

命題（可逆性と行列式） $n \times n$ **行列 $\mathbf{A}$ が可逆であることは、$\det\mathbf{A} \neq 0$ であることと同値である。言い換えれば、$\det\mathbf{A} = 0$ であることは、$\mathbf{A}$ の列が線形従属であることと同値である。**

証明 $\mathbf{A}$ が可逆ならば、乗法性の帰結 3 より $\det\mathbf{A}\,\det\mathbf{A}^{-1} = 1$ となるため、$\det\mathbf{A} \neq 0$ である。逆に、$\mathbf{A}$ が可逆でないと仮定しよう。33.2.2.3 節の命題よりその列は線形従属であるため、ある列が他の列の線形結合として $\mathbf{a}_j = \sum_{i \neq j} c_i \mathbf{a}_i$ と書ける。第 j 列を多重線形性で展開すると $\det\mathbf{A} = \sum_{i \neq j} c_i \det(\dots, \mathbf{a}_i, \dots)$ となり、各項で $\mathbf{a}_i$ が第 i 列と第 j 列の両方の位置に現れる。したがってすべての項が一致する 2 つの列を持ち、交代性によって 0 になる。$\blacksquare$

通しで用いる行列では $\det\mathbf{A} = 5 \neq 0$ によって可逆性が保証され、$\mathbf{B}$ では $\det\mathbf{B} = 0$ によって特異性が確認される。ただし、この定理が約束**しない**事柄にも留意しておこう。非ゼロの行列式が保証するのは $\mathbf{A}^{-1}$ が数学的に存在することであり、それを数値的に安全に計算できることではない（33.2.3.6 節）。

33.2.3.5 ブロック行列と低ランク更新

行列の一部分（1 つのブロックや、小さな変更が加わる前の行列）の逆行列がすでにわかっている場面は少なくない。これを再利用できないだろうか。答えは、スカラーではなくブロックに対して消去を 1 ステップ施すことで得られる。正方行列 $\mathbf{M}$ を 4 つのブロック $\mathbf{M}_{11}$、$\mathbf{M}_{12}$、$\mathbf{M}_{21}$、$\mathbf{M}_{22}$ に分割し、$\mathbf{M}_{11}$ は正方な可逆行列とする。第 1 ブロック行に左から $\mathbf{M}_{21}\mathbf{M}_{11}^{-1}$ を掛けて第 2 ブロック行から引けば左下のブロックが消去でき、列に対して同じ操作を行えば右上のブロックも消去できる。これを行列の積として書くと次のようになる。

$$\begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{M}_{21}\mathbf{M}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{M}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{M}_{11}^{-1}\mathbf{M}_{12} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \qquad \mathbf{S} = \mathbf{M}_{22} - \mathbf{M}_{21}\mathbf{M}_{11}^{-1}\mathbf{M}_{12}, \tag{33.21}$$

これは 3 つの因子を実際に掛け合わせれば確かめられる。行列 $\mathbf{S}$ は $\mathbf{M}_{11}$ に関する**シューア補行列**（Schur complement）であり、最初のブロックを消去した後に $\mathbf{M}_{22}$ に残る部分にほかならない。通しで用いる行列を 1×1 ブロックに分割した場合、$\mathbf{S} = 3 - (-1) \cdot 1^{-1} \cdot 2 = 5$ となる。

外側の 2 つの因子はいずれも対角成分が 1 の三角行列であるため、それぞれの行列式は 1 である。乗法性とブロック三角行列の規則を適用すると $\det \mathbf{M} = \det \mathbf{M}_{11} \, \det \mathbf{S}$（上の例では $1 \cdot 5$）が得られ、$\mathbf{M}$ が可逆であることは $\mathbf{S}$ が可逆であることと同値になる。外側の因子の逆行列は、非対角ブロックの符号を反転させるだけで得られる。したがって、3 つの因子の逆行列を逆順に掛けると次の式が得られる。

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{I} & -\mathbf{M}_{11}^{-1}\mathbf{M}_{12} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{M}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{M}_{21}\mathbf{M}_{11}^{-1} & \mathbf{I} \end{bmatrix}. \tag{33.22}$$

$\mathbf{M}^{-1}$ の右下のブロックは単に $\mathbf{S}^{-1}$ であり、左上のブロックは $\mathbf{M}_{11}^{-1} + \mathbf{M}_{11}^{-1}\mathbf{M}_{12}\mathbf{S}^{-1}\mathbf{M}_{21}\mathbf{M}_{11}^{-1}$ となる。しかし、左上隅から消去を始めなければならない理由はどこにもない。先に $\mathbf{M}_{22}$ を消去すればブロックの役割が入れ替わり、$\mathbf{M}^{-1}$ の左上のブロックはもう一方のシューア補行列 $\mathbf{M}_{11} - \mathbf{M}_{12}\mathbf{M}_{22}^{-1}\mathbf{M}_{21}$ の逆行列として現れる。この行列は、36.3.3.3 節でガウス分布の条件付き共分散として再び登場する。

左上のブロックに対するこれら 2 つの表現は一致するはずであり、この事実が驚くほど役に立つ。いま $\mathbf{A}^{-1}$ が $n \times n$ 行列 $\mathbf{A}$ に対して既知であり、そこへ $\mathbf{A}$ にランク k の行列（$\mathbf{U}\mathbf{W}^{\top}$、ただし $\mathbf{U}, \mathbf{W} \in \mathbb{R}^{n \times k}$ と書く）だけの変更が加わったとしよう。ブロックを $\mathbf{M}_{11} = \mathbf{A}$、$\mathbf{M}_{12} = -\mathbf{U}$、$\mathbf{M}_{21} = \mathbf{W}^{\top}$、$\mathbf{M}_{22} = \mathbf{I}_k$ と選ぶと、一方のシューア補行列は $\mathbf{A} + \mathbf{U}\mathbf{W}^{\top}$、もう一方は $k \times k$ 行列 $\mathbf{I}_k + \mathbf{W}^{\top}\mathbf{A}^{-1}\mathbf{U}$ となる。左上のブロックの 2 つの表現を等置すれば、次の**ウッドベリーの恒等式**（Woodbury identity）が得られる。

$$(\mathbf{A} + \mathbf{U}\mathbf{W}^{\top})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I}_k + \mathbf{W}^{\top}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{W}^{\top}\mathbf{A}^{-1}. \tag{33.23}$$

行列に対するランク k の変更は、その逆行列にも高々ランク k の変更しか引き起こさず、その変化分を求めるのに必要なのは $k \times k$ の逆行列だけである。$\mathbf{A}^{-1}$ が手元にあれば、更新コストは $n^2 k$ のオーダーの積和演算で済み、逆行列を最初から求め直す n^3 はかからない。さらに $\mathbf{A}$ が対角行列であれば、$\mathbf{A} + \mathbf{U}\mathbf{W}^{\top}$ を係数とする方程式を解くコストはわずか nk^2 のオーダーで済む。また 2 通りの消去は $\det \mathbf{M}$ に対しても 2 つの表現を与えるため、$\det(\mathbf{A} + \mathbf{U}\mathbf{W}^{\top}) = \det \mathbf{A} \, \det(\mathbf{I}_k + \mathbf{W}^{\top}\mathbf{A}^{-1}\mathbf{U})$ が従う。$k = 1$ のときは内側の逆行列が単なる割り算になり、この恒等式は**シャーマン・モリソンの公式**（Sherman–Morrison formula）として知られる。

$$(\mathbf{A} + \mathbf{u}\mathbf{w}^{\top})^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\,\mathbf{w}^{\top}\mathbf{A}^{-1}}{1 + \mathbf{w}^{\top}\mathbf{A}^{-1}\mathbf{u}}. \tag{33.24}$$

最小二乗法による当てはめでデータ例を 1 つずつ追加・削除できるのは、この公式のおかげである。各データ例は $\mathbf{X}^{\top}\mathbf{X}$ をランク 1 の項だけ変化させるからである。演習問題 6 では、この行列式と逆行列の公式の両方を正規化フローに応用する。

33.2.3.6 行列式の限界

$\det \mathbf{A} \neq 0$ であるとわかれば行列が可逆であることは言えるが、その行列が空間を潰す状態にどれほど**近い**か、あるいはどの方向に沿って潰れかけているかについては何も語らない。両極端の 2 つの鮮烈な例がこの事実を浮き彫りにする。第 1 に、スカラー倍した単位行列 $0.1\,\mathbf{I}_{400}$ は逆行列 $10\,\mathbf{I}_{400}$ を持つこれ以上ないほど性質のよい行列であるが、その行列式 10^{-400} はあまりに小さく、倍精度浮動小数点数で表せる最小の正の数を下回ってアンダーフローしてしまう。第 2 に、極端なせん断 $\left(\begin{smallmatrix} 1 & 10^{8} \\ 0 & 1 \end{smallmatrix}\right)$ は行列式が 1 であるにもかかわらず、左下の成分をわずか 10^{-8} 変えるだけで特異行列になってしまう（5）。

以下では、（1 つ目の例には符号と絶対値の対数を安全に求める `slogdet` を用いて）両方の行列式を、せん断の特異値とともに計算する。セルを実行する前に、**変更を加えていない**元のせん断に対して NumPy がどのランクを返すか予想してみてほしい。

```
sign, logdet = np.linalg.slogdet(0.1 * np.eye(400))
print(np.linalg.det(0.1 * np.eye(400)), sign, logdet)
shear = np.array([[1.0, 1e8], [0.0, 1.0]])
print(np.linalg.det(shear), np.linalg.matrix_rank(shear),
      np.linalg.svd(shear, compute_uv=False))
shear[1, 0] = 1e-8
print(np.linalg.det(shear), np.linalg.matrix_rank(shear))
```

```
0.0 1.0 -921.0340371976151
1.0 1 [1.e+08 1.e-08]
0.0 1
```

予想どおり、`det` を $0.1\,\mathbf{I}_{400}$ に適用すると `0.0` へとアンダーフローするが、`slogdet` は $400 \ln 0.1 \approx -921.03$ を難なく計算する。変更前のせん断の行列式は 1 であり、摂動を加えたせん断の行列式は丸め誤差の範囲で 0 になるが、`matrix_rank` は**どちらに対しても**ランク 1 を返す。厳密な算術では変更前のせん断のランクは 2 であるが、その 2 つの特異値は約 10^{8} と 10^{-8} である。NumPy は、最大特異値に行列サイズと計算機精度を掛けた値（ここでは $10^{8} \cdot 2 \cdot 2.2 \cdot 10^{-16} \approx 4 \cdot 10^{-8}$）を下回る特異値をすべて 0 とみなす。せん断の成分自体は厳密に格納されていても、大きさ 10^{8} の数を含む浮動小数点演算では 10^{-8} 前後の丸め誤差が生じるため、数値計算上このせん断は特異行列と区別できないのである。

行列が異なる方向に沿って空間をどのように引き伸ばし押し縮めるかを、たった 1 つの数値で捉えることは到底できない。それらの方向を見るには、固有値と特異値が必要になる。通しで用いる行列 $\mathbf{A}$ では、固有値は $2 \pm i$ であってその積は $5 = \det \mathbf{A}$ になり（33.3 節）、特異値は約 3.618 と 1.382 であってその積は $5 = \lvert \det \mathbf{A} \rvert$ になる（33.5 節）。実はすべての正方行列において、行列式は（複素共役ペアを含め重複度を込めて数えた）固有値の積に等しく、$\lvert \det \mathbf{A} \rvert$ は特異値の積に等しい。行列が空間を潰す状態にどれほど近いかを測るのは行列式ではなく最大特異値と最小特異値の**比**であり、こうした方向ごとの倍率を求めることが次節以降の課題となる。

33.2.4 まとめ

行列とは、標準座標軸の行き先を記録した各列によって完全に定まる線形変換であり、行列積とは右から左へと読む線形変換の合成にほかならない。アインシュタイン記法と `einsum` は、ドット積、行列積、トレース、そして Transformer のアテンションのような多軸テンソルの縮約を、単一の添字規則のもとで統一する。

すべての $m \times n$ 行列は、到達可能な出力からなる列空間と 0 に送られる入力からなる零空間を定め、両者はランク・退化次数の定理 $\operatorname{rank} \mathbf{A} + \dim \ker \mathbf{A} = n$ によって結ばれている。したがって、256×784 層は少なくとも 528 個の入力次元を必然的に消去する。正方行列においては、

列が線形独立であること、零空間が自明であること、フルランクであること、行列式が非ゼロであること、そして逆行列を持つことはすべて同値である。行列式は写像が体積を拡大・縮小する符号付き倍率を測る量であり、合成のもとで乗法的となり、任意の三角行列に対しては対角成分の積に等しい。ブロック消去を 1 ステップ施すとシューア補行列が現れ、そこからブロック行列の行列式と逆行列が、さらにウッドベリーの恒等式を通じて低ランク変更後の逆行列が導かれる。とはいえ、非ゼロの行列式が保証するのは逆行列の存在だけであり、行列が特異性にどれほど近いかまでは教えてくれない。この問いは固有値と特異値へと直結する。

33.2.5 演習問題

1. **角度を保つ写像。** すべての埋め込みベクトルを同じ直交行列で回転させても、すべてのコサイン類似度は変わらない。本問では、この性質を持つすべての線形変換を求める。
   1. 正方行列 $\mathbf{A}$ の列に関する条件として、$\mathbf{A}\mathbf{e}_1$ と $\mathbf{A}\mathbf{e}_2$ が直交するための条件を求めよ。また、$\mathbf{A} = \mathrm{diag}(1, 10)$ はこの条件を満たすが、$(1,1)$ と $(1,-1)$ のなす角を変えてしまうことを示せ。
   2. 可逆な $\mathbf{A}$ が非ゼロベクトルのあらゆるペアのなす角を保つと仮定する。これをペア $\mathbf{e}_i, \mathbf{e}_j$ および $\mathbf{e}_i + \mathbf{e}_j, \mathbf{e}_i - \mathbf{e}_j$（ただし $i \neq j$）に適用し、$\mathbf{A}^\top \mathbf{A} = c^2 \mathbf{I}$ がある $c > 0$ に対して成り立つこと、すなわち $\mathbf{A} = c\mathbf{Q}$（$\mathbf{Q}$ は直交行列）と書けることを示せ。逆に、この形のすべての行列があらゆるコサイン類似度を保つことを示せ。
   3. 代わりに、$\|\mathbf{A}\mathbf{x}\| = \|\mathbf{x}\|$ がすべての $\mathbf{x}$ に対して成り立つと仮定する。$\|\mathbf{x}+\mathbf{y}\|^2$ と $\|\mathbf{x}-\mathbf{y}\|^2$ を展開することで $\mathbf{A}$ がすべてのドット積を保つことを示し、$\mathbf{A}$ が直交行列であると結論せよ。
2. **積のランク。** 線形オートエンコーダは、d 次元の入力を幅 $k < d$ の隠れ層に通し、$\mathbf{x} \mapsto \mathbf{W}_2 \mathbf{W}_1 \mathbf{x}$（ただし $\mathbf{W}_1 \in \mathbb{R}^{k \times d}$、$\mathbf{W}_2 \in \mathbb{R}^{d \times k}$）と変換する。
   1. $\mathbf{A}$ を $m \times n$ 行列、$\mathbf{B}$ を $n \times p$ 行列とする。$\mathbf{AB}$ の列空間が $\mathbf{A}$ の列空間に含まれ、$\mathbf{B}$ の零空間が $\mathbf{AB}$ の零空間に含まれることを示せ。1 つ目の事実、および 2 つ目の事実とランク・退化次数の定理を用いて、$\mathrm{rank}(\mathbf{AB}) \le \min(\mathrm{rank}\,\mathbf{A}, \mathrm{rank}\,\mathbf{B})$ を証明せよ。
   2. $\mathbf{W}_2\mathbf{W}_1$ のランクが高々 k であると結論せよ。その零空間の次元がとりうる最小値はいくらか。また、このオートエンコーダがすべての入力を厳密に再構成できる部分空間の最大次元はいくらか。
   3. ともにランク 2 である 2×3 行列と 3×2 行列であって、その 2×2 の積がランク 1 になるものを構成せよ。
3. **回転と可換な行列。** 行列積は一般に可換ではない。本問では $\mathbf{R}_{90}$ に対する例外をすべて求める。
   1. 2×2 行列 $\mathbf{X}$ のうち $\mathbf{X}\mathbf{R}_{90} = \mathbf{R}_{90}\mathbf{X}$ を満たすものをすべて求め、それらが $\left(\begin{smallmatrix} p & -q \\ q & p \end{smallmatrix}\right) = p\mathbf{I} + q\mathbf{R}_{90}$ であることを示せ。
   2. $\mathbf{R}_{90}^2 = -\mathbf{I}$ を用いて $(p\mathbf{I} + q\mathbf{R}_{90})(r\mathbf{I} + s\mathbf{R}_{90}) = (pr - qs)\,\mathbf{I} + (ps + qr)\,\mathbf{R}_{90}$ を示し、これらの行列が複素数 $p + qi$ と同じ規則で掛け合わされることを確かめよ。
   3. $(p, q) \neq (0, 0)$ のとき、$p\mathbf{I} + q\mathbf{R}_{90} = \sqrt{p^2+q^2}\,\mathbf{R}_{\phi}$（ただし角度 ϕ は $(\cos\phi, \sin\phi) = (p,q)/\sqrt{p^2+q^2}$ を満たす）と書け、スケーリングされた回転になることを示せ。さらにその行列式を計算し、$p = q = 0$ でない限り可逆であると結論せよ。
   4. 通しで用いる行列 $\mathbf{A}$ は $\mathbf{R}_{90}$ と可換か。両方の積を計算せよ。
4. **ハウスホルダー鏡映。** $\mathbf{u}$ を $\mathbb{R}^n$ 内の単位ベクトルとし、$\mathbf{H} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^\top$ とおく。
   1. $\mathbf{H}$ が対称かつ直交であること、$\mathbf{H}\mathbf{u} = -\mathbf{u}$ であること、および $\mathbf{H}\mathbf{x} = \mathbf{x}$ がすべての $\mathbf{x}$（$\mathbf{u}$ に垂直なベクトル）に対して成り立ち、したがって $\mathbf{H}$ が法線ベクトル $\mathbf{u}$ を持つ超平面に関する鏡映であることを示せ。
   2. $\mathbf{H} = \mathbf{I} - 2\mathbf{P}$（ただし $\mathbf{P} = \mathbf{u}\mathbf{u}^\top$）と書き、$\mathbf{u}$ への射影（33.1 節）を用いてこの式を説明せよ。$\mathbf{x} - 2\mathbf{P}\mathbf{x}$ は、$\mathbf{x}$ の $\mathbf{u}$ 方向の成分に何をするか。
   3. $\det \mathbf{H} = -1$ を証明せよ。その際、$\mathbf{u}$ を拡張して正規直交基底（直交行列 $\mathbf{Q} =$

$[\mathbf{u}\,\mathbf{q}_2 \cdots \mathbf{q}_n]$ の各列）を作り、$\mathbf{Q}^\top \mathbf{H}\mathbf{Q}$ を計算して乗法性を用いよ。

4. 単位ベクトル $\mathbf{u}$ であって、その鏡映が $(3,4)$ を $(5,0)$ に送るものを求め（ヒント：$\mathbf{u}$ は $(3,4)-(5,0)$ に平行である）、$\mathbf{H}$ を書き下して $\det \mathbf{H} = -1$ を確かめよ。この種の鏡映を 1 列ずつ適用していく手法こそ、数値計算ライブラリが行列を直交因子と三角因子に分解する仕組みである。

5. [code] **行列式と特異性への近さ**。正方行列が可逆であることは、その行列式が非ゼロであることと同値である。では、行列式の大きさは、行列が特異である状態にどれほど近いかをどの程度うまく測れるだろうか。
   1. 多重線形性を用いて $\det(c\mathbf{A}) = c^n \det \mathbf{A}$ を $n \times n$ 行列 $\mathbf{A}$ とスカラー c に対して証明し、行列 $\mathbf{M}$（Equation 33.19）と $c = 2$ に対してこの恒等式をコードで確かめよ。
   2. 対角線上およびそれより下に独立な標準ガウス分布に従う成分を持つ $n \times n$ 下三角行列を、$n = 20$、100、1000 に対して生成せよ。`np.linalg.slogdet` が返す対数と、三角行列の規則により丸め誤差を除いて厳密に一致するはずの対角成分の絶対値の対数和とを比較せよ。どの n から両者は食い違い始め、$n = 1000$ ではどれほどずれるか。さらに `np.linalg.cond` を $n = 20$ と $n = 100$ に対して計算し、この破綻の理由を説明せよ。（このルーチンは入力が三角行列であることを知らず、他の一般の行列と同様に行の入れ替えを伴う分解を実行する。）
   3. 本文では、左下の成分を 10^{-8} 変化させるとせん断 $\left(\begin{smallmatrix} 1 & 10^8 \\ 0 & 1 \end{smallmatrix}\right)$ が特異になると述べている。これを確かめ、さらに単一の成分に対するそれより小さな変更では特異にならないことを示せ。

6. **ランク 1 更新の行列式。●プレーナーフロー** (Rezende and Mohamed 2015) は、$\mathbf{z} \in \mathbb{R}^d$ を $f(\mathbf{z}) = \mathbf{z} + \mathbf{u}\, h(\mathbf{w}\cdot\mathbf{z} + b)$ によって変換する（ただし $\mathbf{u}, \mathbf{w}$ はベクトル、b はスカラー、h はスカラー非線形性である）。$f(\mathbf{z})$ の密度を計算するには、ヤコビ行列 $\mathbf{I} + h'(\mathbf{w}\cdot\mathbf{z}+b)\,\mathbf{u}\mathbf{w}^\top$ の行列式が必要になる。
   1. 多重線形性と交代性を用いて、$\det(\mathbf{u}\mathbf{w}^\top) = 0$ が $n \times n$ 行列（ただし $n \geq 2$）に対して成り立つことを示せ。
   2. 次の 2 つの分解を確かめよ。

$$\begin{aligned}\begin{bmatrix} \mathbf{I} & -\mathbf{u} \\ \mathbf{w}^\top & 1 \end{bmatrix} &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{w}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{u} \\ \mathbf{0}^\top & 1 + \mathbf{w}\cdot\mathbf{u} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I} & -\mathbf{u} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} + \mathbf{u}\mathbf{w}^\top & \mathbf{0} \\ \mathbf{w}^\top & 1 \end{bmatrix}.\end{aligned}$$

   置換公式 Equation 33.18 を用いて、最後の列が $\mathbf{e}_{n+1}$ である行列の行列式が左上の $n \times n$ ブロックの行列式に等しいことを示せ。さらに $\det(\mathbf{I} + \mathbf{u}\mathbf{w}^\top) = 1 + \mathbf{w}\cdot\mathbf{u}$ を Equation 33.20 と三角行列の規則から導け。
   3. プレーナーフローのヤコビ行列式の絶対値の対数を書き下し、その評価コストを一般の $d \times d$ 行列式のコストと比較せよ。また $h = \tanh$ のとき、$\mathbf{u}\cdot\mathbf{w}$ がどの範囲の値をとれば、すべての $\mathbf{z}$ においてヤコビ行列が可逆に保たれるか。
   4. シャーマン・モリソンの公式 Equation 33.24 を用いて、ヤコビ行列 $\mathbf{I} + c\,\mathbf{u}\mathbf{w}^\top$（ただし $c = h'(\mathbf{w}\cdot\mathbf{z}+b)$）の逆行列を書き下せ。この公式が破綻する c の値が行列式が 0 になる値と一致することを確かめ、その逆行列を 1 つのベクトルに適用するのに必要な積和演算の回数を数えよ。なお、ヤコビ行列の逆行列は f 自体の逆写像を与えるわけではない。$\mathbf{z}$ を $f(\mathbf{z})$ から復元する問題が、$\mathbf{w}\cdot\mathbf{z}$ に関する 1 つのスカラー方程式を解くことに帰着することを示せ。

7. [code] **アテンションに対するアインシュタインの総和規約**。クエリ、キー、値の各テンソルがそれぞれ形状 (n_b, n_h, n_q, d_h)、(n_b, n_h, n_k, d_h)、(n_b, n_h, n_k, d_h) を持つとする。
   1. `einsum` の文字列を、スコア $s_{bhit} = \sum_j q_{bhij} k_{bhtj}$ および値の重み付き和 $o_{bhij} = \sum_t p_{bhit}\, v_{bhtj}$（ただし p_{bhit} はアテンション重み）に対してそれぞれ書き下し、`@` と `swapaxes` を用いた計算と一致することを確かめよ。

2. 各縮約の積和演算の回数をサイズの積として数えよ。それらをすべてのクエリに 1 つの $d_h \times d_h$ 線形変換を適用するときの積和演算の回数と比較し、スコア計算のコストの方が大きくなるキー数 n_k の境界を求めよ。
3. $\mathrm{tr}(\mathbf{A}^\top \mathbf{A})$、$\mathrm{tr}(\mathbf{A}\mathbf{A}^\top)$、および $\sum_{ij} a_{ij}^2$ をそれぞれ 1 つの `einsum` としてランダムな 3×5 行列に対して書き、3 つが一致することを確かめよ。2 つの等号のうち、どちらが Equation 33.13 の具体例であり、もう一方はなぜ成り立つのか。

33.3 固有値分解

33.2 節では、行列を 1 回適用したときに空間に何が起きるかを問うた。しかし機械学習における多くの計算では、同じ行列を何度も繰り返し適用する。同一の層を持つ深層線形ネットワークは 50 層を通過すると $\mathbf{A}^{50}\mathbf{x}$ を計算し、二次目的関数に対する勾配降下法はステップごとに誤差へ同じ固定された行列を掛ける。いずれの場合も知りたいのは、信号が増大するか、減衰して消えるか、あるいは一定の値に落ち着くか、そしてそれがどれほど速く進むかである。行列のべき乗を具体的に掛け算すれば数値は得られるが、なぜそうなるのかの説明にはならない。

2 つのウェブページ間のランダムウォークは、この問いの最小の例である。ページ 1 にいるサーファーがそれぞれ確率 $1/2$ でページ 1 に留まるかページ 2 へクリックして移動し、ページ 2 からは確率 $3/4$ でそのまま留まり、確率 $1/4$ でページ 1 に戻るとしよう。$\mathbf{p}_k$ を k 回のクリック後に各ページにいる確率を並べたベクトルとする。1 回クリックするごとに、このベクトルには次の固定された行列が掛けられる。

$$\mathbf{p}_{k+1} = \mathbf{P}\mathbf{p}_k, \qquad \mathbf{P} = \begin{pmatrix} 1/2 & 1/4 \\ 1/2 & 3/4 \end{pmatrix},$$

この行列の各列はそれぞれのページからのクリック確率を記録しているため、列の和は 1 になる。ページ 1 から出発して $\mathbf{p}_0 = (1, 0)$ とすると、最初の数回の分布は $(0.5, 0.5)$、$(0.375, 0.625)$、$(0.344, 0.656)$、$(0.336, 0.664)$ と推移する。クリックを重ねるたびに変化量は小さくなり、ランダムウォークがある一定の分布へ落ち着いていくように見える。50 回クリックした後にサーファーはどこにおり、分布はどれほどの速さで収束するのだろうか。PageRank は、まさにこれと同じ問いをウェブ全体に対して投げかけている。

鍵となるのは、行列が単なる定数倍として作用する特別な方向、すなわち固有ベクトル（eigenvector）を座標軸として初期ベクトルを書き直すことである。その座標系では、各成分がスカラーのべき乗 λ^k として独立に時間発展する。

1.3 節では、行列が伸縮のみを施す方向としてすでに固有ベクトルに出会った。対称行列 $\mathbf{S} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ は $(1, 1)$ を 3 倍に、$(1, -1)$ を 1 倍に引き伸ばし、任意の対称行列に対して $\mathbf{A}^k\mathbf{x} = \sum_i c_i \lambda_i^k \mathbf{v}_i$ が成り立った。本節では対称性の仮定を外す。任意の正方行列の固有値（eigenvalue）を求める方法を学び、空間全体を張る固有ベクトルの組がいつ存在し、存在しないときには何がそれに代わるのかを確認したうえで、なぜ対称性があるとこれほど豊かな性質が保証されるのかを証明する。

なかでも特に重要な結果は 3 つある。固有ベクトル展開 Equation 33.28（$\mathbf{A}^k\mathbf{x}$ の展開）、スペクトル定理（spectral theorem）、そして二次形式（quadratic form）を重み付き二乗和として表す Equation 33.36 である。先を急ぐ読者は、33.3.1.3 節から直接 33.3.2.1 節へ進んでよい。行列が大きいときに固有値の位置をどう見積もって計算するか、また固有値だけでは判断を誤る状況とはどのようなものかという 2 つの実践的な問いは、33.4 節に譲る。本節全体を通して計算には NumPy を用いる。NumPy の `eig` 関数は、実行列が複素固有値を持つ場合には複素数をそのまま返す。

```
import numpy as np
```

33.3.1 固有値と対角化

33.3.1.1 固有ベクトルと固有値

特定の入力ベクトルに対しては、行列 $\mathbf{P}$ はあたかも 1 つの数であるかのようにスケール倍するだけに働く。$\mathbf{P}$ を $(1,2)$ に適用すると $\mathbf{P}(1,2) = (\frac{1}{2}+\frac{1}{2}, \frac{1}{2}+\frac{3}{2}) = (1,2)$ となってまったく変化しない一方、$\mathbf{P}$ を $(1,-1)$ に適用すると $(\frac{1}{2}-\frac{1}{4}, \frac{1}{2}-\frac{3}{4}) = \frac{1}{4}(1,-1)$ となる。第 1 の方向に沿っては $\mathbf{P}$ は 1 倍として作用し、第 2 の方向に沿っては $1/4$ 倍として作用する。

一般に、ゼロでないベクトル $\mathbf{v}$ は、正方行列 $\mathbf{A}$ と対応する**固有値** λ に対して次の関係を満たすとき、その行列の**固有ベクトル**と呼ばれる。

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}. \tag{33.25}$$

幾何学的には、$\mathbf{A}$ は固有ベクトルを倍率 λ で伸縮させるだけであり、$\lambda < 0$ のときはその向きを反転させる。サーファーの行列 $\mathbf{P}$ に対する固有対（eigenpair）$(\lambda, \mathbf{v})$ は、$(1, (1,2))$ と $(\frac{1}{4}, (1,-1))$ である。零ベクトルはすべての λ に対して Equation 33.25 を自明に満たしてしまうため、固有ベクトルからは除外する。しかし、$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ を満たすすべての解（λ は固定）は、$\mathbf{0}$ を含めると線形部分空間をなし、これを λ の**固有空間**（eigenspace）と呼ぶ。

一方、固有値 $\lambda = 0$ は完全に正当な固有値である。これは $\mathbf{A}\mathbf{v} = \mathbf{0}$ がゼロでないある $\mathbf{v}$ に対して成り立つことを意味するから、正方行列が特異（singular）であることと、0 がその固有値の 1 つであることは同値になる。また、実行列が実固有ベクトルを持つとは限らないことにも注意しよう。90° の回転は、平面上のゼロでないあらゆる方向を回転させてしまう。その固有値は複素数になるが、この場合は 33.3.1.6 節で詳しく扱う。

可逆な 2×2 行列が単位円に作用してできる楕円の主軸と、固有ベクトルとを混同しがちであるが、Figure 33.16 に示すとおり両者は別の概念である。対称行列 $\mathbf{S}$ では、確かに固有ベクトルは楕円の軸上に並ぶ。しかし上三角行列 $\left(\begin{smallmatrix}1 & 1\\ 0 & 2\end{smallmatrix}\right)$ ではそうなっていない。その固有ベクトルは $(1,0)$（$\lambda=1$）と $(1,1)$（$\lambda=2$）であるのに対し、楕円の軸は 58° と -32° の方向を向いている（その求め方は 33.5 節で示す）。楕円の軸が最大および最小の伸縮方向を表すのに対し、固有ベクトルはまったく回転しない方向を表す。対称行列ではこれら 2 組の方向が常に一致することが 33.3.2.1 節のスペクトル定理で証明されるが、一般の行列では一致するとは限らない。

33.3.1.2 特性多項式

$\mathbf{P}$ の固有ベクトルは観察によって見つけたが、より大きな行列ではその方法は通用しない。任意の正方行列の固有値を体系的に求めるには、$\lambda\mathbf{v}$ を Equation 33.25 の左辺へ移項し、$\mathbf{v}$ をくくり出す。

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = \mathbf{0}. \tag{33.26}$$

ゼロでない解 $\mathbf{v}$ が存在するのは、$\mathbf{A}-\lambda\mathbf{I}$ が非自明な零空間（null space）を持つとき、すなわち特異であって $\det(\mathbf{A}-\lambda\mathbf{I}) = 0$ を満たすとき、かつそのときに限る。この行列式を展開すると n 次の λ の多項式が得られ、これを**特性多項式**（characteristic polynomial）$p(\lambda) = \det(\mathbf{A}-\lambda\mathbf{I})$ と呼ぶ。その根こそが $\mathbf{A}$ の固有値にほかならない。根 λ が求まれば、斉次連立一次方程式 Equation 33.26 を解くことで対応する固有空間が得られる。

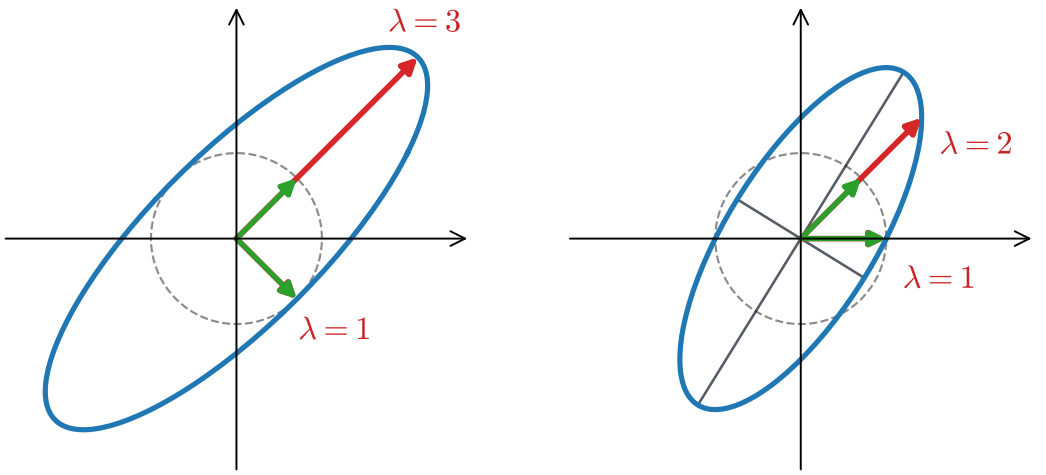


Figure 33.16: 単位円（破線）とその像（青）。左：対称行列 $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ では、単位固有ベクトル（緑）が楕円の軸上に乗り、それらの像（赤）はそれぞれ 3 倍と 1 倍にスケールされる（$\lambda = 1$ では 2 本の矢印が重なる）。右：上三角行列 $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 2 \end{smallmatrix}\right)$ では、固有ベクトル $(1, 0)$ と $(1, 1)$ は楕円の軸（灰色）と一致しない。

この手順をサーファーの行列に適用してみよう。

$$\det(\mathbf{P} - \lambda \mathbf{I}) = \left(\tfrac{1}{2} - \lambda\right)\left(\tfrac{3}{4} - \lambda\right) - \tfrac{1}{4} \cdot \tfrac{1}{2} = \lambda^2 - \tfrac{5}{4}\lambda + \tfrac{1}{4} = (\lambda - 1)\left(\lambda - \tfrac{1}{4}\right).$$

根は 1 と $1/4$ であり、先ほど見つけた 2 つの倍率と一致する。各根において $\mathbf{P} - \lambda \mathbf{I}$ は特異であるため、1 つの行を見るだけで固有ベクトルが求まる。$\lambda = 1$ のときは、$\mathbf{P} - \mathbf{I}$ の第 1 行から $-\frac{1}{2}v_1 + \frac{1}{4}v_2 = 0$、すなわち $v_2 = 2v_1$ となるので、$\mathbf{v} = (1, 2)$ と選べる。$\lambda = \frac{1}{4}$ のときは、$\mathbf{P} - \frac{1}{4}\mathbf{I}$ の第 1 行から $\frac{1}{4}v_1 + \frac{1}{4}v_2 = 0$ となり、$\mathbf{v} = (1, -1)$ が得られる。なお、すべての固有ベクトルはゼロでないスカラー倍の自由度を除いてのみ定まることに留意しておこう。$\mathbf{v}$ が固有ベクトルであれば、$c\mathbf{v}$ も任意の $c \neq 0$ に対して固有ベクトルである。Equation 33.25 の両辺がともに c 倍されるからである。

実際には、この代数計算は数値計算ライブラリに任せる。NumPy の `eig` は、固有値の配列と、対応する固有ベクトルを列に並べた行列を返す。以下のコードの最終行では、すべての固有対に対して Equation 33.25 が成り立つことを一括して確かめている。

```
P = np.array([[0.5, 0.25], [0.5, 0.75]])
lam, V = np.linalg.eig(P)
print('eigenvalues:', lam)
print('eigenvectors (columns):\n', V)
print('max |P V - V diag(lam)|:', np.abs(P @ V - V * lam).max())
```

```
eigenvalues: [0.25 1.  ]
eigenvectors (columns):
 [[-0.70710678 -0.4472136 ]
 [ 0.70710678 -0.89442719]]
max |P V - V diag(lam)|: 0.0
```

順序、スケール、符号はいずれもライブラリの実装次第である。ここでは `eig` は固有値を $1/4$ と 1 の順で返し、各列は $(1,-1)$ および $(1,2)$ に平行な単位長のベクトルとなっている。

数値計算ライブラリは、実際に特性多項式を作って解いているわけではない。$n \geq 5$ では多項式の根をべき根で表す一般の閉じた公式が存在せず、それほど大きくない n でさえ、多項式の係数に生じる微小な丸め誤差（rounding error）に対して根が極端に敏感だからである。その代わりに `eig` は、固有値を変えることなく直交基底変換の列を順次適用し、行列を段階的に三角行列へと近づけていく。三角行列になれば、固有値は対角線上にそのまま現れる。$\lambda\mathbf{I}$ を引いても三角行列のままであり、三角行列の行列式は単に対角成分の積になるからである（33.2.3.2 節）。分解すること自体が不可能なほど巨大な行列に対しては、べき乗法（power iteration）（33.4.1.2 節）が行列・ベクトル積のみを用いて絶対値の大きい固有値を抽出する。

33.3.1.3　対角化

これで冒頭の $\mathbf{P}^{50}\mathbf{p}_0$ に関する問いに答える準備が整った。$\mathbf{P}$ の 2 つの固有ベクトルを行列 $\mathbf{V}$ の列として並べ、対応する固有値を Λ の対角線上に配置する。

$$\mathbf{V} = \begin{pmatrix} 1 & 1 \\ 2 & -1 \end{pmatrix}, \qquad \Lambda = \begin{pmatrix} 1 & 0 \\ 0 & 1/4 \end{pmatrix}.$$

2 つの固有ベクトル方程式 $\mathbf{P}\mathbf{v}_i = \lambda_i \mathbf{v}_i$ を列として横に並べると、簡潔に $\mathbf{P}\mathbf{V} = \mathbf{V}\Lambda$ と書ける。$\det \mathbf{V} = -1 - 2 = -3 \neq 0$ より行列 $\mathbf{V}$ は可逆であり、$\mathbf{V}^{-1} = \frac{1}{3}\left(\begin{smallmatrix} 1 & 1 \\ 2 & -1 \end{smallmatrix}\right)$ となる。右から $\mathbf{V}^{-1}$ を掛けると次式を得る。

$$\mathbf{P} = \mathbf{V}\Lambda\mathbf{V}^{-1}. \tag{33.27}$$

この分解を右から左へ 3 つのステップとして読もう。まず固有ベクトル座標へ変換し（$\mathbf{V}^{-1}$）、次に各座標をそれぞれの固有値で独立にスケール倍し（Λ）、最後に元の座標へ戻す（$\mathbf{V}$）。正方行列 $\mathbf{A}$ が $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$（ただし $\mathbf{V}$ は可逆行列、Λ は対角行列）と書けるとき、その行列は**対角化可能**（diagonalizable）であるといい、この分解そのものを**固有値分解**（eigendecomposition）と呼ぶ。このような分解が存在するのは、$\mathbf{A}$ が n 個の線形独立な固有ベクトルを持ち、それらで $\mathbf{V}$ の列を埋められるとき、かつそのときに限る。

対角化可能な行列のべき乗は、これで実に簡単に計算できる。k 個の $\mathbf{V}\Lambda\mathbf{V}^{-1}$ の積では、隣り合うすべての $\mathbf{V}^{-1}\mathbf{V}$ の組が打ち消し合う。

$$\mathbf{A}^k = \overbrace{(\mathbf{V}\Lambda\mathbf{V}^{-1})\cdots(\mathbf{V}\Lambda\mathbf{V}^{-1})}^{k\text{ times}} = \mathbf{V}\overbrace{\Lambda\cdots\Lambda}^{k\text{ times}}\mathbf{V}^{-1} = \mathbf{V}\Lambda^k\mathbf{V}^{-1}.$$

したがって、対角化可能な行列のべき乗はスカラーのべき乗に帰着する。$\mathbf{A}^k$ をベクトル $\mathbf{x}$ に適用し、$\mathbf{c} = \mathbf{V}^{-1}\mathbf{x}$ を固有基底（eigenbasis）における $\mathbf{x}$ の座標とすれば（すなわち $\mathbf{x} = \sum_i c_i \mathbf{v}_i$）、次式が得られる。

$$\mathbf{A}^k\mathbf{x} = \mathbf{V}\Lambda^k\mathbf{c} = \sum_{i=1}^{n} c_i \lambda_i^k \mathbf{v}_i. \tag{33.28}$$

各固有座標は、等比数列 $c_i\lambda_i^k$ として独立に推移する。ページ 1 から出発するサーファーの場合、$\mathbf{c} = \mathbf{V}^{-1}\mathbf{p}_0 = \frac{1}{3}(1,2) = (\frac{1}{3}, \frac{2}{3})$ であるから、任意のクリック回数 k の後の分布が閉じた形で次のように求まる。

$$\mathbf{p}_k = \mathbf{P}^k \mathbf{p}_0 = \frac{1}{3}\begin{pmatrix}1\\2\end{pmatrix} + \frac{2}{3}\left(\frac{1}{4}\right)^k \begin{pmatrix}1\\-1\end{pmatrix}. \tag{33.29}$$

これで冒頭の問いは完全に解決した。k が大きくなるにつれて第 2 項は消え、分布は $\pi = (1/3, 2/3)$ に近づく。つまり長期的には、サーファーはページ 1 の 2 倍の時間をページ 2 で過ごすことになる。残る誤差は各成分で $\frac{2}{3}\,4^{-k}$ であり、常に固有ベクトル $(1,-1)$ の方向を向いている。1 回クリックするごとに誤差は 4 で割られるため、10 回のクリック後には 6×10^{-7} まで下がり、50 回のクリック後には 10^{-30} 未満となる。検算として $k = 3$ を代入すると $(\frac{1}{3} + \frac{1}{96},\ \frac{2}{3} - \frac{1}{96}) = (0.34375, 0.65625)$ となり、冒頭で計算した 3 回目の反復値と一致する。Figure 33.17 にその幾何学的な様子を示す。すべての $\mathbf{p}_k$ の成分の和は 1 であるため、反復列は直線 $x + y = 1$ 上を滑りながら、$\lambda = 1$ の固有ベクトルがこの直線と交わる点へ向かって進んでいく。

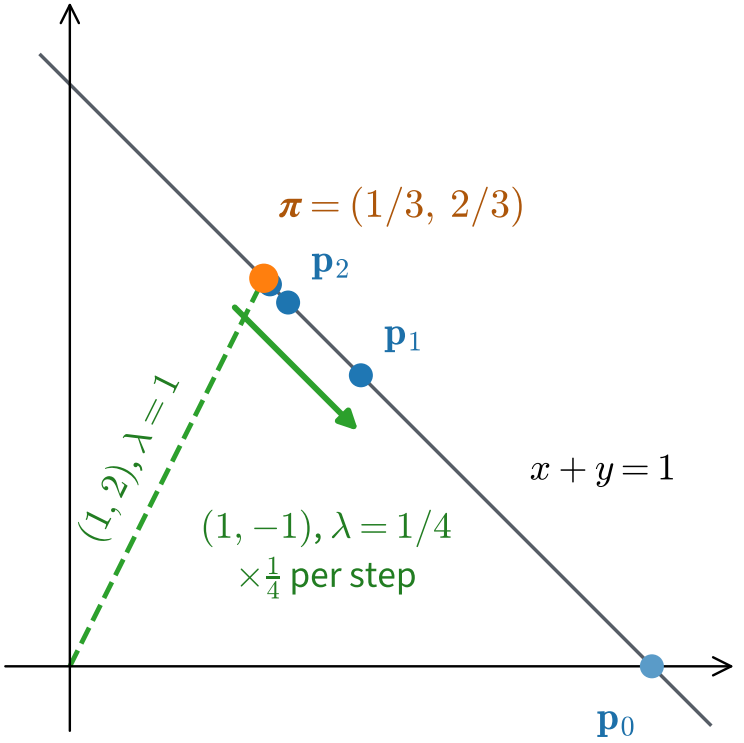


Figure 33.17: サーファーの分布 $\mathbf{p}_k$ は直線 $x + y = 1$ に沿って $\pi = (1/3, 2/3)$（固有ベクトル $(1,2)$（$\lambda = 1$）がこの直線と交わる点）へと移動していく。誤差は固有ベクトル $(1,-1)$（$\lambda = 1/4$）の方向を向き、クリックごとに 4 分の 1 に縮小する。

50 回の行列積と 2 項からなる固有展開が同じ $\mathbf{p}_{50}$ を与えることを検証し、連続する誤差の比も確認してみよう。

```
p0 = np.array([1.0, 0.0])
c = np.linalg.solve(V, p0)              # coordinates of p0 in the eigenbasis
print('P^50 p0            :', np.linalg.matrix_power(P, 50) @ p0)
```

```
    print('sum c_i lam_i^50 v_i:', V @ (c * lam ** 50))
    pi = np.array([1 / 3, 2 / 3])
    errs = np.array([np.abs(np.linalg.matrix_power(P, k) @ p0 - pi).max()
                     for k in range(8)])
    print('error ratios:', np.round(errs[1:] / errs[:-1], 4))
```

```
P^50 p0             : [0.33333333 0.66666667]
sum c_i lam_i^50 v_i: [0.33333333 0.66666667]
error ratios: [0.25 0.25 0.25 0.25 0.25 0.25 0.25]
```

2 つの計算結果は一致し、隣り合う誤差の比はすべて 0.25 となっている。固有値 1 が $\mathbf{p}_0$ のうち生き残る成分を支配し、固有値 $1/4$ が減衰して消える成分を支配している。

同じ手法は、対角化可能な行列に対する他の関数にもそのまま適用できる。どの固有値も 0 でなければ、$\mathbf{A}^{-1} = \mathbf{V}\Lambda^{-1}\mathbf{V}^{-1}$ となり、対角線上の各固有値の逆数をとるだけで逆行列が求まる。同様に、行列指数関数（matrix exponential）$e^{\mathbf{A}t} = \mathbf{V}e^{\Lambda t}\mathbf{V}^{-1}$（線形微分方程式 $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ の解を与える）も、各固有値の指数関数をとることで構成できる。これについては 38.1 節で詳しく展開する。

相似。固有値分解は、$\mathbf{P}$ と Λ が 2 つの異なる基底で表されたまったく同じ線形変換であることを示している。一般に、$\mathbf{A}' = \mathbf{B}\mathbf{A}\mathbf{B}^{-1}$ という関係（ただし $\mathbf{B}$ はある可逆行列）で結ばれる 2 つの行列は**相似**（similar）であるという。次の命題は、基底を取り替えても行列の固有値は変わらないことを保証する。証明には 33.2 節で学んだ行列式の乗法性を用いる。

$$\det(\mathbf{X}\mathbf{Y}) = \det\mathbf{X}\,\det\mathbf{Y},$$

特にこの性質から、$\det(\mathbf{B}^{-1}) = 1/\det\mathbf{B}$ が $\mathbf{B}\mathbf{B}^{-1} = \mathbf{I}$ より従う。

命題（相似な行列は特性多項式を共有する）。 *任意の正方行列 $\mathbf{A}$ と任意の可逆行列 $\mathbf{B}$ に対して、次が成り立つ。*

$$\det(\mathbf{B}\mathbf{A}\mathbf{B}^{-1} - \lambda\mathbf{I}) = \det(\mathbf{A} - \lambda\mathbf{I}). \tag{33.30}$$

証明。$\lambda\mathbf{I} = \mathbf{B}(\lambda\mathbf{I})\mathbf{B}^{-1}$ と書き換えて $\mathbf{B}$ と $\mathbf{B}^{-1}$ をくくり出すと、$\det(\mathbf{B}\mathbf{A}\mathbf{B}^{-1} - \lambda\mathbf{I}) = \det\big(\mathbf{B}(\mathbf{A} - \lambda\mathbf{I})\mathbf{B}^{-1}\big) = \det\mathbf{B}\,\det(\mathbf{A} - \lambda\mathbf{I})\,\det\mathbf{B}^{-1} = \det(\mathbf{A} - \lambda\mathbf{I})$ となる。$\blacksquare$

したがって、相似な行列は根としての重複度も含めて同じ固有値を持ち、特性多項式によって決まる量はすべて**相似不変量**（similarity invariant）となる。最も有用な 2 つの不変量は行列式とトレースである。任意の対角化可能な行列に対して、次が成り立つ。

$$\det(\mathbf{A}) = \det(\mathbf{V}\Lambda\mathbf{V}^{-1}) = \det(\mathbf{V})\det(\Lambda)\det(\mathbf{V}^{-1}) = \det(\Lambda) = \lambda_1\lambda_2\cdots\lambda_n. \tag{33.31}$$

すなわち、行列式は固有値の積に等しい。$\mathbf{V}$ がどのような体積の歪みをもたらしても $\mathbf{V}^{-1}$ がそれを元に戻すため、Λ による各軸のスケーリングだけが残る。トレースについては、その巡回性（33.2.1.4 節）を用いる。

$$\operatorname{tr}(\mathbf{X}\mathbf{Y}) = \operatorname{tr}(\mathbf{Y}\mathbf{X}),$$

これは積が正方行列になる任意の $\mathbf{X}$ と $\mathbf{Y}$ に対して成り立つ。$\mathbf{X} = \mathbf{V}$、$\mathbf{Y} = \Lambda\mathbf{V}^{-1}$ と置けば、次が得られる。

$$\operatorname{tr}(\mathbf{A}) = \operatorname{tr}(\mathbf{V}\Lambda\mathbf{V}^{-1}) = \operatorname{tr}(\Lambda\mathbf{V}^{-1}\mathbf{V}) = \operatorname{tr}(\Lambda) = \lambda_1 + \cdots + \lambda_n. \tag{33.32}$$

したがって、トレースは固有値の和に等しい。サーファーの行列では、$\operatorname{tr}\mathbf{P} = \frac{1}{2} + \frac{3}{4} = \frac{5}{4} = 1 + \frac{1}{4}$ および $\det\mathbf{P} = \frac{3}{8} - \frac{1}{8} = \frac{1}{4} = 1 \cdot \frac{1}{4}$ と、両方の恒等式を一目で確かめられる。任意の 2×2 行列では、これら 2 つの数から $\lambda^2 - (\operatorname{tr})\lambda + \det$ の根として 2 つの固有値が直接定まる。

実は、固有値を $\mathbb{C}$ 上で根としての重複度込みで数えれば、対角化可能かどうかにかかわらず**すべての**正方行列で両方の恒等式が成り立つ。**代数学の基本定理**（fundamental theorem of algebra）（ここでは既知とする）により、すべての n 次多項式は $\mathbb{C}$ 上で完全に因数分解できる。

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \prod_{i=1}^{n} (\lambda_i - \lambda).$$

$\lambda = 0$ を代入すれば、ただちに $\det\mathbf{A} = \prod_i \lambda_i$ が得られる。トレースについては、両辺の λ^{n-1} の係数を比較する。右辺の係数は $(-1)^{n-1}\sum_i \lambda_i$ である。一方、左辺で $\mathbf{A} - \lambda\mathbf{I}$ の展開項のうち次数 $n-1$ に達するのは対角成分同士の積だけであり、その係数は $(-1)^{n-1}\sum_i a_{ii}$ となる。ゆえに $\operatorname{tr}\mathbf{A} = \sum_i \lambda_i$ が成り立つ。実行列の実数でない固有値は必ず複素共役の対として現れるため、その和も積も実数のまま保たれる。

最後に、対角化可能な行列のランクは、重複度込みで数えた非ゼロ固有値の個数に等しい。可逆行列 $\mathbf{V}$ と $\mathbf{V}^{-1}$ を掛けても次元は保たれるため、$\operatorname{rank}\mathbf{A} = \operatorname{rank}\Lambda$ が成り立つからである。行列が対角化可能でない場合にどうなるかは、演習問題 2 で調べる。

33.3.1.4　代数的重複度と幾何学的重複度

サーファーの行列は 2 つの相異なる固有値を持ち、そこから 2 つの線形独立な固有ベクトルが得られた。これは偶然ではない。相異なる固有値からは、常に線形独立な固有ベクトルが生じる。

命題（相異なる固有値に属する固有ベクトルの線形独立性）。相異なる固有値に属する固有ベクトルは線形独立である。特に、任意の $n \times n$ 行列が n 個の相異なる固有値を持てば、その行列は対角化可能である。

証明。矛盾を導くため、固有ベクトル $\mathbf{v}_1, \dots, \mathbf{v}_m$（対応する固有値 $\lambda_1, \dots, \lambda_m$ は互いに異なる）が線形従属であると仮定しよう。非自明な線形従属関係のうち、非ゼロの係数が最も少ないもの $\sum_{i=1}^{m} c_i\mathbf{v}_i = \mathbf{0}$ を 1 つ選び、必要なら添字を付け替えて $c_m \neq 0$ とする。もし非ゼロの項が $c_m\mathbf{v}_m = \mathbf{0}$ だけなら $\mathbf{v}_m = \mathbf{0}$ となってしまうため、この関係式には少なくとも 2 つの非ゼロ係数が含まれる。ここでこの関係式に $\mathbf{A}$ を適用し、元の関係式の λ_m 倍を引く。

$$\mathbf{0} = \mathbf{A}\sum_i c_i\mathbf{v}_i - \lambda_m \sum_i c_i\mathbf{v}_i = \sum_{i=1}^{m-1} c_i(\lambda_i - \lambda_m)\,\mathbf{v}_i.$$

すると $\mathbf{v}_m$ の項が消え、より短い関係式が残る。しかもこれは依然として非自明である。他の任意の非ゼロ係数 c_i（$i < m$）は、固有値が互いに異なるため $c_i(\lambda_i - \lambda_m) \neq 0$ として生き残るからである。これは選んだ関係式の最小性に矛盾する。$\mathbf{A}$ が n 個の相異なる固有値を持つとき、その n 個の線形独立な固有ベクトルは $\mathbb{C}^n$（すべての固有値が実数なら $\mathbb{R}^n$）の基底をなすため、$\mathbf{V}$ は可逆となる。$\blacksquare$

固有値に重複がある場合は、2 種類の重複度を区別して追う必要がある。λ が特性多項式の根として現れる回数を**代数的重複度**（algebraic multiplicity）$a(\lambda)$ と呼ぶ。一方、その固有空間の次

元、すなわちその固有値がもたらす線形独立な固有ベクトルの個数を**幾何学的重複度**（geometric multiplicity）$g(\lambda)$ と呼ぶ。

命題。 正方行列のすべての固有値 λ に対して、$1 \le g(\lambda) \le a(\lambda)$ が成り立つ。

証明。 $\mathbf{A} - \lambda\mathbf{I}$ は特異であるから、その零空間は非自明であり、$g \ge 1$ が従う。上界を示すため、固有空間の基底 $\mathbf{v}_1, \dots, \mathbf{v}_g$ をとり、それを拡張して $\mathbb{C}^n$ の基底とし、これらの基底ベクトルを列として並べた行列を $\mathbf{B}$ とする。このとき、最初の g 列について $\mathbf{AB}$ の列は単に $\lambda\mathbf{v}_1, \dots, \lambda\mathbf{v}_g$ となるから、この新しい基底のもとで行列は次のブロック上三角行列の形をとる。

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{B} = \begin{pmatrix} \lambda\mathbf{I}_g & \mathbf{X} \\ \mathbf{0} & \mathbf{Y} \end{pmatrix}.$$

Equation 33.30 より、この行列は特性多項式 p を $\mathbf{A}$ と共有する。いま固定した固有値を表すのに用いている記号と区別するため、多項式の変数を t と書こう。λ に対して $t\mathbf{I}$ を引いてもブロック上三角構造は保たれ、対角ブロックは $(\lambda - t)\mathbf{I}_g$ と $\mathbf{Y} - t\mathbf{I}$ になる。ブロック上三角行列の行列式は対角ブロックの行列式の積に等しいから（33.2.3.2 節）、$p(t) = (\lambda - t)^g \det(\mathbf{Y} - t\mathbf{I})$ を得る。したがって $(\lambda - t)^g$ は特性多項式を割り切り、$a \ge g$ が成り立つ。$\blacksquare$

1 つの固有値から得られる線形独立な固有ベクトルの個数が、根としての重複度を超えることは決してないが、それより少なくなることはあり得る。いま各固有空間から基底を 1 組ずつ選び、選んだベクトルをすべて集めよう。これらは全体としても線形独立である。実際、それらの線形結合が零ベクトルに等しいとして固有値ごとに項をまとめれば、各グループは零ベクトルか固有ベクトルのいずれかになり、相異なる固有値に関する命題からすべてのグループが零ベクトルでなければならないからである。ゆえに行列が持つ線形独立な固有ベクトルは高々 $\sum_{\lambda} g(\lambda)$ 個である。$\mathbb{C}$ 上では代数的重複度の和は n に等しく、かつすべての固有値に対して $g(\lambda) \le a(\lambda)$（各 λ ごと）が成り立つため、明快な判定条件が得られる。すなわち、**$\mathbf{A}$ の固有ベクトルが張る空間が $\mathbb{C}^n$ 全体に一致するのは、$g(\lambda) = a(\lambda)$ がすべての複素固有値 λ に対して成り立つとき、かつそのときに限る。**

実行列が**実**固有ベクトルからなる基底を持つためには、さらにその固有値がすべて実数でなければならない。90° の回転行列 $\mathbf{R}_{90} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ は、なぜこの追加条件が不可欠なのかを示している。その固有値 $\pm i$ はそれぞれ $g = a = 1$ を満たすため完全な複素固有基底を持つが、実固有ベクトルは 1 つも存在しない。

固有値が重複する場合には、対角化可能性そのものが成り立たなくなることがある。最も単純な反例は、33.2.1.2 節で登場した**せん断**（shear）である。

$$\mathbf{T} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}. \tag{33.33}$$

この行列の特性多項式は $(1 - \lambda)^2$ であるから、$\lambda = 1$ の代数的重複度は 2 である。ところが $(\mathbf{T} - \mathbf{I})\mathbf{v} = \mathbf{0}$ は $\left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)\mathbf{v} = \mathbf{0}$ となり、これから $v_2 = 0$ が強制される。固有空間は $(1, 0)$ が張る直線だけであり、幾何学的重複度は 1 にとどまる。したがって固有基底は存在せず、このせん断は対角化可能でない。ある固有値に対して $g(\lambda) < a(\lambda)$ となる行列は**欠陥行列**（defective）と呼ばれる。Figure 33.18 を見れば、固有ベクトルが足りない理由が視覚的にわかる。せん断はすべての水平線をその高さに比例して横へ滑らせるため、原点を通る直線で向きが保たれるのは水平軸ただ 1 本だけなのである。（対照的に、重複固有値を持つ対角化可能な 2×2 行列は、その固有値を λ とすれば $\lambda\mathbf{I}$ にならざるを得ず、あらゆる方向を保つ。）

せん断を反復適用すると、欠陥性が行列のべき乗の振る舞いをどう変えるかがわかる。$\mathbf{T}^k = \left(\begin{smallmatrix} 1 & k \\ 0 & 1 \end{smallmatrix}\right)$ となるため、$\mathbf{T}^k\mathbf{e}_2 = (k, 1)$ は唯一の固有値が 1 であるにもかかわらず際限なく増大する。純粋

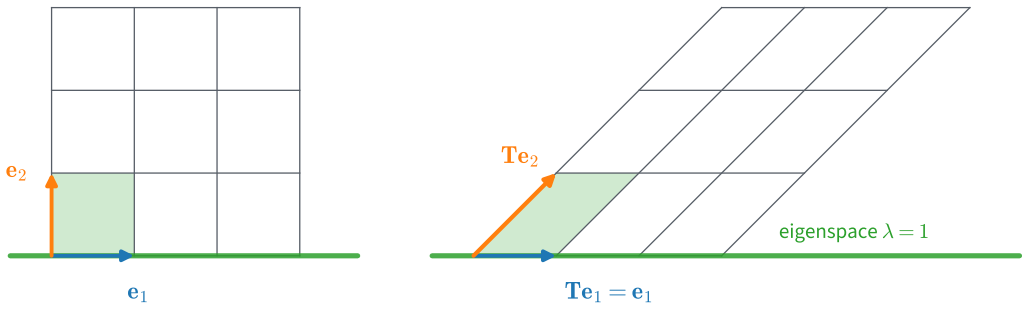


Figure 33.18: 平面に作用する欠陥を持つせん断。すべての水平線がその高さに比例した量だけ右へ滑ってグリッドを傾ける一方、x 軸（緑）は自分自身に写る。すなわち $\mathbf{T}\mathbf{e}_1 = \mathbf{e}_1$ が $\lambda = 1$ に対する唯一の固有方向である。もう一方の基底ベクトルは水平成分を得て $\mathbf{T}\mathbf{e}_2 = (1, 1)$ となるため、向きが保たれる第 2 の方向は存在せず、代数的重複度 2 に対して幾何学的重複度は 1 となる。

な固有展開からは、このような線形の因子 k は決して生じない。次項では、この因子がどこから現れるのかを説明する。

33.3.1.5 ジョルダン標準形

幾何学的重複度が代数的重複度に満たないとき、固有基底は存在しない。では、代わりにどのような構造が現れるのだろうか。本書の以降の議論において最も重要な帰結は 1 つ、すなわち欠陥固有値は λ^k に k の多項式を掛けるということである。複素数体上では、すべての正方行列は次の**ジョルダンブロック**（Jordan block）から構成されるブロック対角行列と相似になる。

$$\mathbf{J}_m(\lambda) = \begin{pmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{pmatrix},$$

これは $m \times m$ のブロックであり、対角線上に単一の固有値 λ が並び、そのすぐ上の優対角線（superdiagonal）に 1 が並ぶ。この分解が**ジョルダン標準形**（Jordan normal form）であり (Horn and Johnson 2012)、本書では証明なしに用いる。Equation 33.30 により、その固有値は元の行列と一致する。

各ジョルダンブロックからは、その最初の基底ベクトルという 1 つの固有ベクトルだけが得られる。したがって λ の幾何学的重複度は λ を持つブロックの個数に等しく、代数的重複度はそれらのブロックのサイズの和に等しい。特に、行列が対角化可能であることと、すべてのジョルダンブロックのサイズが 1×1 であることは同値である。せん断 $\mathbf{T}$（Equation 33.33）は、単一のブロック $\mathbf{J}_2(1)$ そのものである。

反復適用のもとで、ジョルダンブロックは単なるスケーリング以上の働きをする。$\mathbf{J}_m(\lambda) = \lambda\mathbf{I} + \mathbf{N}$ と書こう。ここで $\mathbf{N}$ は対角線のすぐ上に 1 が並び、それ以外の成分がすべて 0 の行列である。幾何学的には、$\mathbf{N}$ は各基底ベクトルを 1 つ前の基底ベクトルへ送るシフトとして作用するため、$\mathbf{N}^m = \mathbf{0}$ となる（べき乗が零行列になる行列を**べき零**（nilpotent）と呼ぶ）。$\mathbf{I}$ と $\mathbf{N}$ は可換であるため、二項定理によってべき乗を展開できる。$m = 2$ のときは最初の 2 項より後のすべての項が $\mathbf{N}^2 = \mathbf{0}$ を含むため、次の項だけが残る。

$$\mathbf{J}_2(\lambda)^k = (\lambda \mathbf{I} + \mathbf{N})^k = \lambda^k \mathbf{I} + k\lambda^{k-1}\mathbf{N}.$$

長期的にはやはり固有値が増大と減衰を分けるものの、そこに k の多項式（次数は $m-1$、すなわち $m \times m$ ブロックのサイズより 1 小さい）が掛かるようになる。せん断のように $|\lambda| = 1$ のときは、反復列は線形にドリフトする。$|\lambda|$ が 1 よりわずかに小さいときには、さらに驚くべき現象が起きる。右上の成分 $k\lambda^{k-1}$ は最初増大し、$k \approx -1/\log|\lambda|$ 付近でピークに達してから、ようやく 0 に向かって減衰するのである。$\lambda = 0.99$ の場合、ピークは $k \approx 100$ 付近に現れ、この成分は $100 \cdot 0.99^{99} \approx 37$ に達する。この予測を数値的に確かめてみよう。

```
J = np.array([[0.99, 1.], [0., 0.99]])   # the Jordan block J_2(0.99)
for k in [1, 50, 100, 200, 800]:
    Jk = np.linalg.matrix_power(J, k)
    pred = k * 0.99 ** (k - 1)   # predicted corner entry, k * lambda^(k-1)
    print(f'k = {k:3d}   ||J^k|| = {np.linalg.norm(Jk, 2):8.3f}   '
          f'k * 0.99^(k-1) = {pred:8.3f}')
```

```
k =   1   ||J^k|| =    1.609   k * 0.99^(k-1) =    1.000
k =  50   ||J^k|| =   30.568   k * 0.99^(k-1) =   30.556
k = 100   ||J^k|| =   36.977   k * 0.99^(k-1) =   36.973
k = 200   ||J^k|| =   27.067   k * 0.99^(k-1) =   27.067
k = 800   ||J^k|| =    0.260   k * 0.99^(k-1) =    0.260
```

$\mathbf{J}$ の固有値はすべて 0.99 であるにもかかわらず、そのべき乗のノルムは約 37（$k = 100$）まで上昇し、27（$k = 200$）に留まったのち、ようやく指数因子 0.99^k が支配的になって減衰へ転じる。固有値だけを見ていても、この過渡的増大（transient growth）を予見することはできない。この現象は 33.4.2.3 節で改めて取り上げる。

ジョルダンブロックは、実行列が実固有基底を持てない 1 つ目の要因、すなわち固有ベクトルの個数そのものが足りない場合を表している。2 つ目の要因はまったく性質が異なる。実行列がある平面を**回転**させるために、その平面内に実固有ベクトルを 1 つも持たない場合である。

33.3.1.6 複素固有値

角度 θ の平面回転を考えよう（33.2.1.3 節）。

$$\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

θ が π の倍数でない限り、$\mathbf{R}_\theta$ はゼロでないすべての実ベクトルの向きを変えてしまうため、実固有ベクトルは存在しない。その特性多項式 $\lambda^2 - 2\cos\theta\,\lambda + 1$ は、複素共役な根 $\lambda = \cos\theta \pm i\sin\theta = e^{\pm i\theta}$ を持つ。ここで絶対値 $|\lambda| = 1$ は $\mathbf{R}_\theta$ が長さを保つことを表し、偏角 $\pm\theta$ は回転角を記録している。

次に、成分を見ただけでは回転が隠れているとはわかりにくい、本章で通しで用いる 2×2 行列を見てみよう。

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}, \qquad \operatorname{tr}\mathbf{A} = 4, \quad \det\mathbf{A} = 5.$$

この行列の特性多項式は $\lambda^2 - 4\lambda + 5$ であり、その根は $\lambda = 2 \pm i = \sqrt{5}\,e^{\pm i\theta}$（ただし $\theta = \arctan\tfrac{1}{2} \approx 26.6^\circ$）となる。$2+i$ に対する固有ベクトルを求めるため、$\mathbf{A} - (2+i)\mathbf{I}$ の第 2 行を読み取ると $-w_1 + (1-i)w_2 = 0$ が得られる。$w_2 = 1$ と置けば次式を得る。

$$\mathbf{w} = \begin{pmatrix} 1 - i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{\mathbf{u}} + i\,\underbrace{\begin{pmatrix} -1 \\ 0 \end{pmatrix}}_{\mathbf{v}}.$$

回転はどこに隠れているのだろうか。複素固有対を持つ任意の実行列 $\mathbf{A}$ をとり、$\mathbf{A}\mathbf{w} = \lambda\mathbf{w}$（ただし $\lambda = a + ib$、$b \neq 0$）として、固有ベクトルを実部と虚部に分け $\mathbf{w} = \mathbf{u} + i\mathbf{v}$ と書く。$\mathbf{A}(\mathbf{u} + i\mathbf{v}) = (a + ib)(\mathbf{u} + i\mathbf{v})$ を展開して実部と虚部をそれぞれ比較すると、次の 2 式が得られる。

$$\mathbf{A}\mathbf{u} = a\mathbf{u} - b\mathbf{v}, \qquad \mathbf{A}\mathbf{v} = b\mathbf{u} + a\mathbf{v}.$$

どちらの像も $\operatorname{span}\{\mathbf{u}, \mathbf{v}\}$ の内部に留まるため、この部分空間は $\mathbf{A}$ のもとで不変である。しかもこれは正真正銘の 2 次元平面である。もし $\mathbf{u}$ と $\mathbf{v}$ が線形従属であれば、それらが張る空間は自分自身に写される 1 本の直線（$\mathbf{w} \neq \mathbf{0}$ より零空間ではない）となり、その直線上で $\mathbf{A}$ はある実スカラー μ 倍として作用することになる。すると $\mathbf{A}\mathbf{w} = \mu\mathbf{w}$ となって $\lambda = \mu$ が実数となり、$b \neq 0$ に矛盾する。そこで $\mathbf{v}$ と $\mathbf{u}$ を列に並べて $\mathbf{B} = [\mathbf{v}\ \ \mathbf{u}]$ と置くと、上の 2 つの等式は $\mathbf{A}\mathbf{B} = \mathbf{B}\left(\begin{smallmatrix} a & -b \\ b & a \end{smallmatrix}\right)$、すなわち同値な形として次式になる。

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{B} = \begin{pmatrix} a & -b \\ b & a \end{pmatrix} = \underbrace{\sqrt{a^2 + b^2}}_{r} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \qquad \theta = \arg(a + ib). \tag{33.34}$$

不変平面の基底 $(\mathbf{v}, \mathbf{u})$ のもとでは、$\mathbf{A}$ は回転・スケーリングブロック $r\mathbf{R}_\theta$ として作用する。すなわち絶対値 r 倍にスケールし、偏角 θ だけ回転させる。この導出では $\mathbf{A}\mathbf{w} = \lambda\mathbf{w}$ しか使っていないため、得られるブロックは $\mathbf{w}$ のスケールや位相の回転の選び方に依存しない。この対応は逆向きにも成り立つ。$(x, y) \in \mathbb{R}^2$ を複素数 $x + iy$ と同一視すれば、このブロックは (x, y) を $(ax - by,\ bx + ay)$ へ写すが、これは複素数 $a + ib$ の乗算にほかならない。

通しで用いる行列 $\mathbf{A}$ については、$\mathbf{B} = \left(\begin{smallmatrix} -1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ かつ $\mathbf{B}^{-1}\mathbf{A}\mathbf{B} = \left(\begin{smallmatrix} 2 & -1 \\ 1 & 2 \end{smallmatrix}\right)$ となり、$\sqrt{5}$ 倍のスケーリングと 26.6° の回転を表す。すると、1 ステップごとにベクトルの長さが一様に $\sqrt{5} \approx 2.24$ 倍になると思うかもしれない。`eig` が返す固有ベクトルからこのブロックを計算し、実際の反復列 $\mathbf{A}^k(1, 0)$ の長さを追ってみよう。

```
A = np.array([[1., 2.], [-1., 3.]])
lam, V = np.linalg.eig(A)
j = np.argmax(lam.imag)                  # the eigenvalue a + ib with b > 0
r, theta = np.abs(lam[j]), np.angle(lam[j])
u, v = V[:, j].real, V[:, j].imag
B = np.column_stack([v, u])              # basis (v, u) of the invariant plane
print(f'lambda = {lam[j]:.4f}   r = {r:.4f}   theta = {theta:.4f}')
print('B^-1 A B =\n', np.round(np.linalg.inv(B) @ A @ B, 4))
xs = [np.array([1., 0.])]
for _ in range(8):
    xs.append(A @ xs[-1])
norms = np.linalg.norm(xs, axis=1)
```

```
    print('norm ratios:', np.round(norms[1:] / norms[:-1], 2))
    x20 = np.linalg.matrix_power(A, 20) @ np.array([1., 0.])
    print(f'average rate ||A^20 x||^(1/20) = {np.linalg.norm(x20) ** (1 / 20):.3f}')
```

```
lambda = 2.0000+1.0000j   r = 2.2361   theta = 0.4636
B^-1 A B =
 [[ 2. -1.]
 [ 1.  2.]]
norm ratios: [1.41 2.92 3.45 2.76 2.27 1.88 1.5  1.54]
average rate ||A^20 x||^(1/20) = 2.252
```

eig がどの固有ベクトルを選んでも、得られるブロックは $\left(\begin{smallmatrix}2 & -1\\ 1 & 2\end{smallmatrix}\right)$ となる。ところがベクトルの長さは、ステップごとに一定の倍率 2.24 で増大するわけでは**ない**。軌跡 $(1,0)\to(1,-1)\to(-1,-4)\to(-9,-11)\to(-31,-24)$ に沿って見ると、1 ステップあたりのノルムの比は約 1.4 から 3.5 の間で大きく揺れ動く。理由は、スケーリングが一様になるのが正規直交ではない座標系 $(\mathbf{v},\mathbf{u})$ の内部に限られるからである。元の標準座標に戻すと、反復列は引き伸ばされた楕円に沿って外向きに渦を巻き、約 $2\pi/\theta\approx 13.6$ ステップで 1 周する。長期的な幾何平均 $\|\mathbf{A}^k\mathbf{x}\|^{1/k}$ だけが $\sqrt{5}$ に落ち着く（実際、2.25 に $k=20$ までに到達する）。標準座標において剛体回転と一様なスケーリングになるのは、行列が**正規**（normal）であるときに限られる。この性質は 33.4.2.2 節で定義する。

実基底変換を除けば、すべての実正方行列はここまでに見いだした 3 つの基本動作、すなわち直線に沿った伸縮、ジョルダンブロック内のせん断、そして平面内の回転を伴うスケーリングを単独または組み合わせて構成される。この分解は**実ジョルダン標準形**（real Jordan form）として知られている (Horn and Johnson 2012)。

33.3.2 対称行列と二次形式

33.3.2.1 スペクトル定理

実ベクトルからなる正規直交固有基底（orthonormal eigenbasis）の存在が保証されるのは、どのような行列だろうか。対称性は、前節で出会った 2 つの障害をどちらも取り除く。複素固有値も、固有ベクトルの不足も生じなくなるのである。1.3 節の対称行列 $\mathbf{S}$ がその好例である。その単位固有ベクトル $(1,1)/\sqrt{2}$ と $(1,-1)/\sqrt{2}$ は互いに直交しており、次の分解を与える。

$$\mathbf{S}=\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}=\underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}}_{\mathbf{V}}\begin{pmatrix}3 & 0\\ 0 & 1\end{pmatrix}\underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}}_{\mathbf{V}^{\top}}.$$

固有ベクトル行列 $\mathbf{V}$ は直交行列（$\mathbf{V}^{\top}\mathbf{V}=\mathbf{I}$）であるため、その逆行列は単なる転置になる。スペクトル定理は、これが**すべての**実対称行列に対して成り立つことを保証する。これは本章の中心定理であり、正定値性、レイリー商（Rayleigh quotient）、そして 33.5 節の内容すべてがこの定理の上に築かれる。

定理（スペクトル定理）。 すべての実対称行列 $\mathbf{A}$ は、実固有値と実固有ベクトルからなる正規直交基底を持つ。同値な表現として、

$$\mathbf{A}=\mathbf{V}\Lambda\mathbf{V}^{\top},\tag{33.35}$$

と書ける。ここで $\Lambda=\operatorname{diag}(\lambda_1,\dots,\lambda_n)$ は実行列であり、$\mathbf{V}$ は直交行列である。

これを 3 段階で証明する。第 1 にすべての固有値が実数であること、第 2 に相異なる固有値に属する固有ベクトルが直交すること、第 3 に空間全体を張る正規直交固有基底が常に存在することを示す。複素ベクトルに対してはエルミート内積（Hermitian inner product）$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^{*}\mathbf{y}$ を用いる。ここで $\mathbf{x}^{*}$ は共役転置（conjugate transpose）であり、実ベクトル上では通常のドット積に帰着する。外部から借りる事実は、特性多項式が少なくとも 1 つの複素根を持つこと（代数学の基本定理）、すなわち $\mathbb{C}$ 上に少なくとも 1 組の固有対が存在することだけである。(i) と (ii) はそれぞれわずか 2 行で済む。本質的な議論は (iii) にある。

証明 (i)：固有値は実数である。 いったん複素成分を許して、$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}$ かつ $\mathbf{v} \neq \mathbf{0}$ とする。$\mathbf{A}$ は実対称行列であるから $\mathbf{A}^{*} = \mathbf{A}^{\top} = \mathbf{A}$ が成り立ち、したがって次を得る。

$$\lambda\, \mathbf{v}^{*}\mathbf{v} = \mathbf{v}^{*}\mathbf{A}\mathbf{v} = (\mathbf{A}\mathbf{v})^{*}\mathbf{v} = \bar{\lambda}\, \mathbf{v}^{*}\mathbf{v}.$$

$\mathbf{v}^{*}\mathbf{v} = \|\mathbf{v}\|^{2} > 0$ であるから、両辺をこれで割れば $\lambda = \bar{\lambda}$ となり、固有値は実数でなければならない。するとその固有ベクトルは実係数の斉次連立一次方程式 $(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = \mathbf{0}$ の解となるため、やはり実ベクトルとして選べる。$\blacksquare$

証明 (ii)：相異なる固有値の固有ベクトルは直交する。 $\mathbf{A}\mathbf{u} = \lambda \mathbf{u}$ および $\mathbf{A}\mathbf{v} = \mu \mathbf{v}$ とし、$\lambda \neq \mu$ とする。(i) により両固有値および両固有ベクトルは実数としてよい。$\mathbf{A} = \mathbf{A}^{\top}$ を用いて内積の中で $\mathbf{A}$ をもう一方へ移すと、次式が成り立つ。

$$\lambda \langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{A}\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, \mathbf{A}\mathbf{v} \rangle = \mu \langle \mathbf{u}, \mathbf{v} \rangle.$$

よって $(\lambda - \mu)\langle \mathbf{u}, \mathbf{v} \rangle = 0$ となる。$\lambda \neq \mu$ であるから、$\langle \mathbf{u}, \mathbf{v} \rangle = 0$ でなければならない。$\blacksquare$

証明 (iii)：正規直交固有基底の存在。 n に関する帰納法で示す。$n = 1$ のとき主張は明らかである。$n > 1$ とすると、(i) より実固有対 $(\lambda_1, \mathbf{v}_1)$ で $\|\mathbf{v}_1\| = 1$ を満たすものが存在する。ここで $(n-1)$ 次元の直交補空間（orthogonal complement）$U = \mathbf{v}_1^{\perp}$ を考えよう。対称性により、$\mathbf{A}$ は U を再びそれ自身の中へ写す。実際、任意の $\mathbf{x} \in U$ に対して次が成り立つ。

$$\langle \mathbf{A}\mathbf{x}, \mathbf{v}_1 \rangle = \langle \mathbf{x}, \mathbf{A}\mathbf{v}_1 \rangle = \lambda_1 \langle \mathbf{x}, \mathbf{v}_1 \rangle = 0.$$

U の正規直交基底を 1 組選び、そのベクトルを列として並べた行列を $\mathbf{Q} \in \mathbb{R}^{n \times (n-1)}$ とする。$\mathbf{A}\mathbf{Q}$ の各列はすべて U に属するため、$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{C}$ と書ける。ここで $\mathbf{C} = \mathbf{Q}^{\top}\mathbf{A}\mathbf{Q}$ は $\mathbf{A}$ を U に制限した変換を表す。$\mathbf{C}$ が対称行列であり（$\mathbf{C}^{\top} = \mathbf{Q}^{\top}\mathbf{A}^{\top}\mathbf{Q} = \mathbf{C}$）、サイズが $(n-1) \times (n-1)$ であることに注目しよう。帰納法の仮定より、$\mathbf{C} = \mathbf{V}' \Lambda' \mathbf{V}'^{\top}$ が、ある直交行列 $\mathbf{V}'$ に対して成り立つ。そこで $\mathbf{Q}\mathbf{V}'$ の各列を調べると、$(\mathbf{Q}\mathbf{V}')^{\top}\mathbf{Q}\mathbf{V}' = \mathbf{V}'^{\top}\mathbf{V}' = \mathbf{I}$ よりこれらは正規直交であり、U に属するから $\mathbf{v}_1$ と直交し、さらに $\mathbf{A}$ の固有ベクトルにもなっている。なぜなら $\mathbf{A}\mathbf{Q}\mathbf{V}' = \mathbf{Q}\mathbf{C}\mathbf{V}' = \mathbf{Q}\mathbf{V}'\Lambda'$ が成り立つからである。これらを $\mathbf{v}_1$ の後ろに並べれば、直交行列 $\mathbf{V} = [\,\mathbf{v}_1 \;\; \mathbf{Q}\mathbf{V}'\,]$ が得られ、$\mathbf{A}\mathbf{V} = \mathbf{V}\Lambda$ を満たす。$\blacksquare$

幾何学的には、Equation 33.35 は任意の対称な線形変換を 3 つの明快な操作へと分解する。まず $\mathbf{V}^{\top}$ が正規直交固有ベクトルを座標軸に揃える。これは純粋な回転か、それに鏡映を組み合わせた変換である（上の $\mathbf{S}$ では $\det \mathbf{V} = -1$ であった）。次に Λ が各座標軸を対応する固有値倍に伸縮させる。最後に $\mathbf{V}$ が軸を元の向きへ戻す。これにより、Figure 33.16 の左パネルが $\mathbf{S}$ について図示した性質が、すべての対称行列に対して証明されたことになる。

長方形か正方かを問わず、あらゆる行列 $\mathbf{A}$ から対称行列 $\mathbf{A}^{\top}\mathbf{A}$ を作ることができる。その二次形式 $\mathbf{x}^{\top}\mathbf{A}^{\top}\mathbf{A}\mathbf{x} = \|\mathbf{A}\mathbf{x}\|^{2}$ は決して負にならない。次項ではこの性質に名前を付け、判定法を導き、活用していく。

33.3.2.2 正定値性

二次形式 $\mathbf{x}^\top \mathbf{A}\mathbf{x}$ が負の値をとることはあるだろうか。損失曲面の臨界点が最小点であるか、行列が正当な共分散行列であるか、そして最小二乗法の問題が一意解を持つかは、この 1 つの問いにかかっている。対称行列 $\mathbf{A}$ が次を満たすとき、**半正定値**（positive semidefinite、PSD）であるといい、$\mathbf{A} \succeq 0$ と書く。

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} \geq 0 \quad \text{for all } \mathbf{x},$$

さらに、ゼロでないすべてのベクトルに対して不等号が狭義に成り立つとき、**正定値**（positive definite、PD）であるといい、$\mathbf{A} \succ 0$（すべての非ゼロ $\mathbf{x}$ に対して厳密に正）と書く。たとえば $\mathbf{x}^\top \mathbf{S}\mathbf{x} = 2x_1^2 + 2x_1x_2 + 2x_2^2 = x_1^2 + x_2^2 + (x_1+x_2)^2$ は $\mathbf{x} \neq \mathbf{0}$ のとき常に正であるから $\mathbf{S} \succ 0$ であるが、$\left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ は $\mathbf{e}_2$ に対して -1 をとるため PD でも PSD でもない。手計算による平方完成は高次元になるとすぐに現実的でなくなるが、固有値を見れば任意の対称行列に対してこの問いに明快な決着がつく。

命題（固有値による定値性の判定）。 *対称行列 $\mathbf{A}$ に対して、次が成り立つ。*

$$\mathbf{A} \succeq 0 \iff \lambda_i \geq 0 \text{ for all } i, \qquad \mathbf{A} \succ 0 \iff \lambda_i > 0 \text{ for all } i.$$

証明。 スペクトル定理を適用して $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^\top$ と書き、長さを保つ直交基底変換によって固有ベクトル座標 $\mathbf{y} = \mathbf{V}^\top \mathbf{x}$ へ移ると、次式を得る。

$$\mathbf{x}^\top \mathbf{A}\mathbf{x} = \mathbf{x}^\top \mathbf{V}\Lambda\mathbf{V}^\top \mathbf{x} = \mathbf{y}^\top \Lambda \mathbf{y} = \sum_{i=1}^{n} \lambda_i \, y_i^2 = \sum_{i=1}^{n} \lambda_i \, (\mathbf{v}_i^\top \mathbf{x})^2. \tag{33.36}$$

固有ベクトル座標では、二次形式は固有値を重みとする単純な重み付き二乗和になる。すべての $\lambda_i \geq 0$ であれば、この和が非負であることは明らかである。逆に $\mathbf{x} = \mathbf{v}_j$ を代入すると $\mathbf{y} = \mathbf{e}_j$ かつ $\mathbf{v}_j^\top \mathbf{A}\mathbf{v}_j = \lambda_j$ となるため、$\lambda_j \geq 0$ でなければならない。正定値の場合も、$\mathbf{y} \neq \mathbf{0}$ が $\mathbf{x} \neq \mathbf{0}$ のとき常に成り立つことから、等号を除いたまったく同じ議論により従う。$\blacksquare$

Equation 33.36 を $\mathbf{S}$ と $\mathbf{x} = \mathbf{e}_1$ で確かめてみよう。固有座標は $\mathbf{y} = \mathbf{V}^\top \mathbf{e}_1 = (1/\sqrt{2}, 1/\sqrt{2})$ であり、重み付き和は $3 \cdot \tfrac{1}{2} + 1 \cdot \tfrac{1}{2} = 2$ となって $\mathbf{e}_1^\top \mathbf{S}\mathbf{e}_1 = s_{11}$ と一致する。深層学習で正定値性が登場する場面の大半は、次の 3 つの帰結で説明できる。

- **PD ならば可逆である。** $\mathbf{A} \succ 0$ かつ $\mathbf{A}\mathbf{x} = \mathbf{0}$ とすると $\mathbf{x}^\top \mathbf{A}\mathbf{x} = 0$ となり、これから $\mathbf{x} = \mathbf{0}$ が導かれる。
- **グラム行列と共分散行列は PSD である。** 任意の行列 $\mathbf{X}$ に対して $\mathbf{x}^\top \mathbf{X}^\top \mathbf{X}\mathbf{x} = \|\mathbf{X}\mathbf{x}\|^2 \geq 0$ であるから、$\mathbf{X}^\top \mathbf{X} \succeq 0$ が成り立つ。これが狭義に PD となるのは、$\mathbf{X}$ が列フルランク（full column rank）を持つとき（すなわち $\mathbf{X}\mathbf{x} = \mathbf{0}$ から $\mathbf{x} = \mathbf{0}$ が従うとき）、かつそのときに限る。中心化したデータの標本共分散は、グラム行列をサンプル数で割ったものであるため、常に PSD である。
- **PD のヘッセ行列は狭義の極小値を保証する。** f が 2 回連続微分可能で $\nabla f(\mathbf{x}^*) = \mathbf{0}$ を満たし、そのヘッセ行列 $\nabla^2 f(\mathbf{x}^*)$（2 階偏微分の対称行列）が PD であるとする。微小なステップ δ に対し、二次テイラー展開（34.2.3.2 節）は $f(\mathbf{x}^* + \delta) = f(\mathbf{x}^*) + \tfrac{1}{2}\delta^\top \nabla^2 f(\mathbf{x}^*)\delta + o(\|\delta\|^2)$ と書ける。Equation 33.36 より二次項は少なくとも $\tfrac{1}{2}\lambda_{\min}\|\delta\|^2 > 0$ となるため、この関数は $\mathbf{x}^*$ で狭義の極小値をとる。なお、この条件は十分条件であって必要条件ではない。実際、$f(x) = x^4$ は 0 において狭義の最小値をとるが、そこでの二階導関数は 0 である。

正定値性には、**コレスキー分解**（Cholesky factorization）による高速なアルゴリズム的判定法もある。すなわち、$\mathbf{A} \succ 0$ が成り立つのは、$\mathbf{A} = \mathbf{L}\mathbf{L}^\top$ を満たす対角成分が正の下三角行列 $\mathbf{L}$ が存在するとき、かつそのときに限る。一方の向きはただちに従う。そのような $\mathbf{L}$ は可逆であるから、$\mathbf{x}^\top \mathbf{A}\mathbf{x} = \|\mathbf{L}^\top \mathbf{x}\|^2 > 0$ がゼロでないすべての $\mathbf{x}$ に対して成り立つ。コレスキー分解を試みることは正定値性を判定する標準的な数値計算手法であり、すべての固有値を計算するよりはるかに低コストで済む。さらに副産物として、同じコレスキー因子 $\mathbf{L}$ を用いれば、独立な標準ガウス分布からのサンプル $\mathbf{z}$ を、サンプル $\mathbf{L}\mathbf{z}$（共分散は $\mathbf{L}\mathbf{L}^\top = \mathbf{A}$）へと変換できる。

幾何学的には、固有値の符号が曲面 $z = \mathbf{x}^\top \mathbf{A}\mathbf{x}$ の形状を決定する（Figure 33.19）。すべての固有値が正ならお椀型になり、固有値 0 の方向があればその方向に沿って平坦な谷樋型になり、符号が混在していれば鞍型になる。

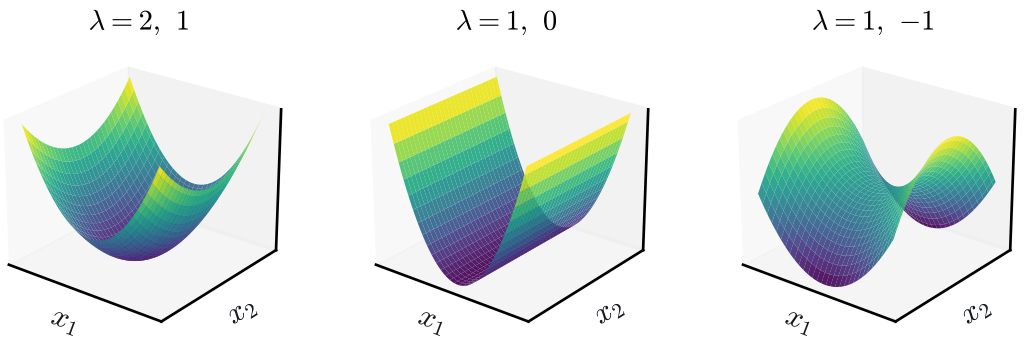


Figure 33.19: 3 つの対角行列に対する二次形式の曲面 $z = \mathbf{x}^\top \mathbf{A}\mathbf{x}$ は、固有値の符号に応じた形状をとる。すべての $\lambda_i > 0$ ならお椀型（正定値）、1 つの $\lambda_i = 0$ なら谷樋型（半正定値）、符号が混在していれば鞍型（不定値）になる。

グラム行列のランク条件は、その固有値にそのまま現れる。以下では、列が線形独立であるため $\mathbf{X}$ に対して $\mathbf{X}^\top \mathbf{X}$ が PD になるはずの縦長行列と、第 2 列が第 1 列の 2 倍であるため固有値 0 を持つ PSD 行列になるはずの行列を比較する。

```
X_full = np.array([[1., 0.], [1., 1.], [0., 1.], [2., 1.]])  # rank 2
X_dep = np.array([[1., 2.], [1., 2.], [0., 0.], [2., 4.]])   # col 2 = 2*col 1
for X, name in [(X_full, 'full column rank'), (X_dep, 'dependent columns')]:
    eigvals = np.linalg.eigvalsh(X.T @ X).round(4)
    eigvals[np.abs(eigvals) < 1e-9] = 0.0  # clean up tiny round-off / -0.0
    print(f'{name}: eigenvalues of X^T X = {eigvals}')
```

```
full column rank: eigenvalues of X^T X = [1.1459 7.8541]
dependent columns: eigenvalues of X^T X = [ 0. 30.]
```

フルランクの場合は両方の固有値が正になっている。一方、列が線形従属な場合は片方の固有値が 0 に落ちる。二次形式は $(2, -1)$、すなわち $\mathbf{X}$ が零ベクトルへ写すまさにその方向に沿って 0 になる。

33.3.2.3 レイリー商

恒等式 Equation 33.36 の役割は符号の判定にとどまらない。この式によって、最大固有値と最小固有値が最適化問題の解として特徴づけられる。対称行列 $\mathbf{A}$ に対し、**レイリー商**を次のように定義する。

$$R(\mathbf{x}) = \frac{\mathbf{x}^{\top}\mathbf{A}\mathbf{x}}{\mathbf{x}^{\top}\mathbf{x}}, \qquad \mathbf{x} \neq \mathbf{0}.$$

これは単位長さの二乗あたりの二次形式の値を測る量であり、$\mathbf{x}$ が固有ベクトルのときには $R(\mathbf{x})$ はその固有値に一致する。本項全体を通して、固有値は絶対値ではなく符号付きの値によって $\lambda_1 \geq \cdots \geq \lambda_n$ の順に並べる。

命題（Rayleigh）。 固有値（A は対称行列）を $\lambda_1 \geq \cdots \geq \lambda_n$ とするとき、

$$\lambda_1 = \max_{\mathbf{x}\neq\mathbf{0}} R(\mathbf{x}), \qquad \lambda_n = \min_{\mathbf{x}\neq\mathbf{0}} R(\mathbf{x}),$$

が成り立ち、最大値と最小値はそれぞれ対応する固有ベクトルにおいて達成される。

証明。 $\mathbf{x}$ を定数倍しても $R(\mathbf{x})$ は変わらないため、単位ベクトルに限って考えればよい。$\mathbf{y} = \mathbf{V}^{\top}\mathbf{x}$ と置くとノルムが保たれ（$\|\mathbf{y}\| = \|\mathbf{x}\| = 1$）、Equation 33.36 は $R(\mathbf{x}) = \sum_i \lambda_i y_i^2$ となる。これは非負の重み y_i^2（総和は 1）による固有値の重み付き平均である。いかなる平均も最小値と最大値の間に収まるため、$\lambda_n \leq R(\mathbf{x}) \leq \lambda_1$ が成り立つ。$\mathbf{x} = \mathbf{v}_1$ または $\mathbf{x} = \mathbf{v}_n$ としてすべての重みを単一の座標に集中させれば、上下の限界がそれぞれ達成される。$\blacksquare$

通しで用いている対称行列 $\mathbf{S}$ に当てはめると、$R(\mathbf{x})$ は $[1, 3]$ の範囲を動き、45° の固有ベクトル方向で最大となり、135° の固有ベクトル方向で最小となる。その中間のあらゆる方向では、固有値 1 と 3 が固有座標の二乗に比例して混ぜ合わされる。

部分空間全体にわたって最大化すれば、中間のすべての固有値も復元できる。**クーラン・フィッシャーの定理**（Courant–Fischer theorem）によれば、$\lambda_j = \max_{\dim U = j} \min_{\mathbf{0}\neq\mathbf{x}\in U} R(\mathbf{x})$ は、すべての j 次元部分空間 U にわたって保証されるレイリー商の最大値を与える（演習問題 4d で証明の手順を追う）。

最大・最小固有値の 2 つの応用は、本書全体を通して繰り返し登場する。第 1 に、任意の行列 $\mathbf{A}$ に対し、$\mathbf{A}^{\top}\mathbf{A}$ のレイリー商の最大値は $\max_{\mathbf{x}\neq\mathbf{0}} \|\mathbf{A}\mathbf{x}\|^2/\|\mathbf{x}\|^2$ であり、これは $\mathbf{A}$ がベクトルを引き伸ばし得る最大倍率の二乗に等しい。第 2 に、凸二次関数 $f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^{\top}\mathbf{A}\mathbf{x}$（$\mathbf{A} \succ 0$）に対する勾配降下法を考えよう。その更新式は $\mathbf{x} \leftarrow (\mathbf{I} - \eta\mathbf{A})\mathbf{x}$ である。Equation 33.28 により、各固有座標にはステップごとに $1 - \eta\lambda_i$ が掛かるため、任意の初期点から反復が収束するための必要十分条件は、$|1 - \eta\lambda_i| < 1$ がすべての i で成り立つこと、すなわち $0 < \eta < 2/\lambda_{\max}$ となる。このように最大固有値はステップサイズの上限を定め（これは滑らかさの定数 L として 35.1 節に登場する）、ステップサイズが $\eta \leq 1/\lambda_{\max}$ の範囲にあるとき、最小固有値が最も遅い縮小率 $1 - \eta\lambda_{\min}$ を決定する。両者の比 $\lambda_{\max}/\lambda_{\min}$ は $\mathbf{A}$ の条件数（condition number）であり、33.5.3.2 節で任意の行列へと一般化する。

33.3.2.4 標本共分散とマルチェンコ・パスツール則

高次元データの中に構造を探すときには、標本共分散行列の大きな固有値を調べる。しかし、どの固有値が本当に大きいのかを判断するには、まずデータにまったく構造がないときに固有値がどう見えるかという基準が必要である。いま n 個の純粋なノイズのサンプルを d 次元空間から抽出し、各座標は独立で平均 0、分散 1（したがって真の共分散は $\mathbf{I}$）であるとして、それらを行として並べた行列を $\mathbf{X} \in \mathbb{R}^{n\times d}$ としよう。標本共分散 $\mathbf{C} = \frac{1}{n}\mathbf{X}^{\top}\mathbf{X}$ の固有値は、1 の近くに集まるだろうか。任意の単位ベクトル $\mathbf{w}$ に対して、レイリー商 $\mathbf{w}^{\top}\mathbf{C}\mathbf{w}$ はデータを $\mathbf{w}$ 上へ射影したときの標本分散になる。したがって $\mathbf{C}$ の最大固有値と最小固有値は、分散が最も大きい方向と最も小さい方向に沿った標本分散そのものであり、主成分分析（principal component analysis）（33.5.2.2 節）がデータの構造として解釈するまさにその量である。

驚くべきことに、固有値は 1 の近くに集まらない。$d, n \to \infty$ の極限を、**アスペクト比** $c = d/n \le 1$ を固定したまま考えよう。すると $\mathbf{C}$ の固有値のヒストグラムは、次の区間を台とする決定論的な密度関数へ収束する。

$$[(1-\sqrt{c})^2,\ (1+\sqrt{c})^2],$$

この区間は**マルチェンコ・パスツールのバルク**（Marchenko–Pastur bulk）として知られており (Marchenko and Pastur 1967)、ノイズが有限の 4 次モーメントを持つ限り、最大・最小固有値はちょうどその両端へ収束する。$c > 1$ のときは行列 $\mathbf{C}$ のランクが高々 $n < d$ となるため、このバルクに加えて少なくとも $d - n$ 個の固有値が恒等的に 0 になる。1 次元あたり 4 サンプル ($c = 1/4$) ある場合でさえ、純粋なノイズの固有値は $[0.25, 2.25]$ 全体に大きく広がる。ランダムな揺らぎが最も大きく出た方向の標本分散は真の分散の 2 倍を超え、最も小さい方向では 4 分の 1 にしかならない。$n = 4000$、$d = 1000$ の規模になれば、この漸近極限はすでに驚くほど正確である。実際に確かめてみよう。

```
n, d = 4000, 1000                       # aspect ratio c = d / n = 1/4
X = np.random.randn(n, d)               # pure noise: true covariance I
ev = np.linalg.eigvalsh(X.T @ X / n)
c = d / n
print(f'eigenvalues of C: min {ev.min():.3f}, max {ev.max():.3f}')
print(f'Marchenko-Pastur edges: {(1 - c ** 0.5) ** 2:.3f}, '
      f'{(1 + c ** 0.5) ** 2:.3f}')
```

```
eigenvalues of C: min 0.254, max 2.240
Marchenko-Pastur edges: 0.250, 2.250
```

何度実行しても、最小固有値と最大固有値は予測された両端 0.25 と 2.25 から 100 分の数以内の範囲に収まる。このバルクの内部に落ちる固有値はどれも純粋なノイズと区別できず、上端を明確に超える固有値だけが真の構造の証拠となる。固有値がバルクを抜け出すにはどれほど強い信号が必要かについては、演習問題 5 で調べる。

本節全体を通して、2 つのことを当然の前提としてきた。行列の固有値を実際に計算できること、そして固有値を見れば、多数回のステップ後だけでなく数ステップ後のべき乗の振る舞いもわかるということである。大規模な行列や、対称からほど遠い行列に対しては、どちらの前提も安全ではない。33.4 節ではこの両方を取り上げる。

33.3.3 まとめ

固有ベクトル座標のもとでは、対角化可能な行列はスカラーの並びとして作用し、$\mathbf{A}^k\mathbf{x} = \sum_i c_i \lambda_i^k \mathbf{v}_i$ によってマルコフ連鎖、深層線形ネットワーク、および二次関数上の勾配降下法における反復適用を明快に説明する。$\mathbb{C}$ 上で空間全体を張る固有基底が存在するための必要十分条件は、すべての固有値について幾何学的重複度が代数的重複度と一致することである。一致しない場合にはジョルダンブロックが k の多項式因子をもたらして過渡的増大を引き起こし得るほか、複素共役対は不変平面内での回転とスケーリングに対応する。実対称行列はこの両方の落とし穴を免れており、常に実固有値と正規直交固有基底を持つ。その二次形式は重み付き二乗和となるため、固有値の符号が正定値性を決定し、最大・最小固有値はレイリー商の最大値・最小値と一致する。最後に、純粋なノイズから計算した標本共分散の固有値はマルチェンコ・パスツール区間全体にわたって広がるため、バルクの上端を超える固有値だけが真の構造を示す信号となる。

33.3.4 演習問題

1. **トレース、行列式、定値性。** 恒等式 Equation 33.31 と Equation 33.32 は、和と積を通じて 2×2 行列の 2 つの固有値を定める。

   1. 特性多項式を作らずにトレースと行列式だけから $\left(\begin{smallmatrix} 4 & 1 \\ 1 & 4 \end{smallmatrix}\right)$ の固有値を求め、続いて固有ベクトルを求めよ。サーファーの行列 $\mathbf{P}$（トレースは $5/4$、行列式は $1/4$）についても同様に行え。
   2. 2×2 の対称行列が正定値であるための必要十分条件は、そのトレースと行列式がともに正であることを示せ。半正定値性を特徴づけるのは、トレースと行列式に対するどのような符号条件か。
   3. 行列 $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$、$\left(\begin{smallmatrix} 1 & 2 \\ 2 & 1 \end{smallmatrix}\right)$、$\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ はそれぞれ、二次関数 $f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^{\top}\mathbf{A}\mathbf{x}$ の臨界点 $\mathbf{0}$ におけるヘッセ行列である。(b) を用いて、各 f について原点が狭義の最小点、狭義でない最小点、鞍点のいずれであるかを判定せよ。鞍点となるものについては、f が減少する方向を 1 つ挙げよ。
   4. コレスキー因子 $\mathbf{L}$（対象となる行列は $\left(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\right)$）の対角成分が $\sqrt{a}$ と $\sqrt{(ac - b^2)/a}$ になることを示せ。正の対角成分を持つ因子が存在する条件が (b) の判定条件と一致するのはなぜか。

2. **ランクと固有値 0。** 本節では、対角化可能な行列のランクが重複度込みの非ゼロ固有値の個数に等しいことを示した。

   1. $\mathbf{N} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ のランクと固有値を求めよ。対角化可能な行列に対する議論のどの段階が $\mathbf{N}$ では破綻するか。
   2. g_0 を固有値 0（対象は $n \times n$ 行列 $\mathbf{A}$）の幾何学的重複度とし、ただし $g_0 = 0$（0 が固有値でない場合）とする。任意の正方行列に対して $\operatorname{rank}\mathbf{A} = n - g_0$ が成り立つことを示せ。さらにこれから、ランクは代数的重複度込みで数えた非ゼロ固有値の個数以上であり、等号が成り立つための必要十分条件は固有値 0 が欠陥固有値でないことであると結論せよ。
   3. 行列の**特異値**は $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^{\top}\mathbf{A})}$、すなわち PSD 行列 $\mathbf{A}^{\top}\mathbf{A}$ の固有値の平方根で定義される。$\mathbf{N}$、および 33.3.1.5 節のジョルダンブロック $\mathbf{J}_m(0)$ の特異値を求めよ。非ゼロ特異値の個数を、ランクおよび非ゼロ固有値の個数と比較せよ。

3. **中心化行列。** $\mathbf{1} \in \mathbb{R}^n$ をすべての成分が 1 のベクトルとし、$\mathbf{H} = \mathbf{I} - \tfrac{1}{n}\mathbf{1}\mathbf{1}^{\top}$ と置く。

   1. $n\mathbf{I} - \mathbf{1}\mathbf{1}^{\top}$ に対し、$\mathbf{x}^{\top}(n\mathbf{I} - \mathbf{1}\mathbf{1}^{\top})\mathbf{x}$ を差 $x_i - x_j$ の二乗和として表すことで、この行列が半正定値であることを示せ。続いて $n\mathbf{I} - \mathbf{1}\mathbf{1}^{\top}$ のすべての固有値と対応する固有ベクトルを求めよ。
   2. $\mathbf{H}$ が対称行列であり $\mathbf{H}^2 = \mathbf{H}$ を満たすことを示し、(a) からその固有値とランクを導け。
   3. $\mathbf{X} \in \mathbb{R}^{n \times d}$ の各行を n 個のサンプルとする。すべての行から標本平均を引くと $\mathbf{H}\mathbf{X}$ となるため、標本共分散は $\mathbf{C} = \tfrac{1}{n}\mathbf{X}^{\top}\mathbf{H}\mathbf{X}$ と書ける。$\mathbf{C} \succeq 0$ かつ $\operatorname{rank}\mathbf{C} \le n - 1$ を示せ。512 次元の埋め込みについて、256 個の例からなるミニバッチから計算した標本共分散は、少なくとも何個の固有値 0 を持つか。これは 33.3.2.4 節における $c = d/n > 1$ の場合にあたる。この性質により、$\mathbf{C}$ に対するどのような演算が不可能になるか。

   Gilbert Strang の MIT 18.065 問題集（第 13 回講義の問題 4）を改変。

4. [code] **計算された固有値の精度。**●`np.linalg.eigvals` の背後にあるアルゴリズムは後退安定である。すなわち、ある摂動行列を伴う $\mathbf{A} + \mathbf{E}$（ただし $\|\mathbf{E}\|_2$ は $\epsilon_{\text{mach}}\|\mathbf{A}\|_2$ と同程度の大きさであり、倍精度では $\epsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$）の厳密な固有値を返す (Golub and Van Loan 1996)。微小な摂動によって、欠陥固有値は ε の分数べき乗だけ動き得る。

   1. $\mathbf{e}_1, \dots, \mathbf{e}_m$ を標準基底ベクトルとする。$\mathbf{J}_m(\lambda) + \varepsilon\,\mathbf{e}_m\mathbf{e}_1^{\top}$（ジョルダンブロックの左

下の隅に ε を置いた行列）の固有値が $\lambda + \varepsilon^{1/m}\omega$（ただし ω は 1 の m 乗根全体を動く）となることを示せ。

2. $m = 1, \dots, 4$ に対し、$\mathbf{Q}\mathbf{J}_m(2)\mathbf{Q}^{\top}$ を構成せよ（ただし直交行列 $\mathbf{Q}$ はガウスランダム行列の QR 分解からとる）。これを 20 回試行し、$\max_i |\hat{\lambda}_i - 2|$（$\hat{\lambda}_i$ は数値計算された固有値）の中央値を記録して $\epsilon_{\text{mach}}^{1/m}$ と比較せよ。対称行列 $\mathbf{Q}(2\mathbf{I})\mathbf{Q}^{\top}$ についても同じ実験を行え。
3. $\mathbf{A}$ と $\mathbf{E}$ を対称行列とする。33.3.2.3 節の Rayleigh の命題を用いて、$\lambda_{\max}(\mathbf{A}+\mathbf{E}) \le \lambda_{\max}(\mathbf{A}) + \|\mathbf{E}\|_2$ および $\lambda_{\min}(\mathbf{A}+\mathbf{E}) \ge \lambda_{\min}(\mathbf{A}) - \|\mathbf{E}\|_2$ を示せ。これによって (b) の対称行列の結果はどのように説明されるか。
4. クーラン・フィッシャーの定理 $\lambda_j = \max_{\dim U = j} \min_{\mathbf{0} \neq \mathbf{x} \in U} R(\mathbf{x})$ を、対称行列 $\mathbf{A}$（固有値は $\lambda_1 \ge \cdots \ge \lambda_n$）に対して証明せよ。上界については U と $\operatorname{span}(\mathbf{v}_j, \dots, \mathbf{v}_n)$（次元は $n - j + 1$ であるため、共通部分は少なくとも 1 本の直線を含む）の共通部分をとれ。下界については $U = \operatorname{span}(\mathbf{v}_1, \dots, \mathbf{v}_j)$ とせよ。さらにこれから、(c) をすべての固有値へ拡張するワイルの不等式 $|\lambda_j(\mathbf{A}+\mathbf{E}) - \lambda_j(\mathbf{A})| \le \|\mathbf{E}\|_2$（任意の j）を導け。

5. [code] **バルクを超えるスパイク。**●33.3.2.4 節では、マルチェンコ・パスツールのバルクを純粋なノイズの固有値が広がる範囲として扱った。いまデータに真の方向が 1 つ含まれているとしよう。すなわち n 個のサンプルは独立なガウス座標を持ち、第 1 座標を除くすべての座標の分散が 1 で、第 1 座標の分散だけが $1 + \beta$ であるとする。$n \to \infty$ とし $c = d/n$ を固定した極限において、Baik and Silverstein (2006) は $\mathbf{C} = \frac{1}{n}\mathbf{X}^{\top}\mathbf{X}$ の最大固有値が次へ収束することを示した。

$$\lambda_{\max} \to \begin{cases} (1 + \sqrt{c})^2, & \beta \le \sqrt{c}, \\ (1 + \beta)\left(1 + c/\beta\right), & \beta > \sqrt{c}. \end{cases}$$

1. $n = 2000$、$d = 500$ とし、$\mathbf{C}$ の最大固有値を $\beta \in \{0, 0.3, 0.5, 0.7, 1, 2\}$ のそれぞれについて 5 回の試行の平均として求め、理論予測と比較せよ。
2. $d = 500$、$\beta = 0.4$ に固定する。上の公式を用いて、最大固有値がバルクを抜け出すサンプル数 n の閾値を求めよ。次にサンプル数を $n \in \{500, 1000, 2000, 4000, 8000\}$ と増やしながら、最大固有値をバルクの上端、理論予測、および $\beta = 0$（同じ n）に対する最大固有値と比較せよ。$n = 4000$ では分離が見えにくいのはなぜか。
3. 実務者が、固有値がバルクの上端を超える主成分だけを残すとする。$c = 1/4$ においてある方向が残されるには、その方向の分散はどれほど大きくなければならないか。また、他の方向より分散が 10% 大きい方向を検出するには、1 次元あたり何サンプル必要か。

33.4 実践におけるスペクトル

33.3 節で扱った行列は手計算で分解できるほど小さく、固有値も期待どおりの働きをした。ランダムサーファーの誤差は、第 2 固有値が予測したとおりクリックごとに $1/4$ ずつ縮小したのである。しかし教科書の例を一歩出ると、この 2 つの好都合な前提は崩れ去る。ウェブのリンク行列は何十億もの行を持つため、特性多項式（characteristic polynomial）を書き下したり完全な固有値分解（eigendecomposition）を計算したりすることはない。また、たとえ固有値がわかっていたとしても、見た目ほど多くのことを保証してくれるわけではない。固有値がともに 0.99 であるジョルダンブロックは、減衰に転じる前にノルムが 37 まで増大したし（33.3.1.5 節）、リカレントネットワークにおける誤差逆伝播法（backpropagation）では、1 つの行列を 50 回掛けるのではなく、勾配に 50 個の**異なる**行列を次々と掛けることになる。

本節では、この 2 つの問題を順に取り上げる。第 1 に、分解するには大きすぎる行列の固有値について、何がわかるだろうか。実は驚くほど多くのことがわかる。行列の成分を見るだけで全固有値を複素平面上のいくつかの円盤内に絞り込めるほか、大規模になっても低コストな唯一の演算である行列・ベクトル積を繰り返せば、最大固有値とその固有ベクトルが求まる。PageRank はもともとこのようにして計算されていた。第 2 に、有限回のステップや、変化していく行列の積について、固有値は何を語り残しているだろうか。こちらの答えは厳しい現実を突きつける。固有値が支配するのは単一の行列のべき乗における長期的な振る舞いだけであり、それ以外を扱うには行列ノルム、そして最終的には 33.5 節の特異値（singular value）が必要になる。

以下で扱うすべての話題が深層学習において同じ重要度を持つわけではないため、どこに重点を置くべきかを知っておくとよい。本書全体を通じて繰り返し現れる考え方が 3 つある。べき乗法（power iteration、33.4.1.2 節）は層の最大の引き伸ばし率を推定する手法であり、スペクトル正規化（spectral normalization、33.5.4.3 節）の中核を担う。過渡的増大（transient growth）とヤコビ行列（Jacobian matrix）の積（33.4.2.3 節、33.4.2.5 節）は、8.5 節の勾配爆発と勾配消失（vanishing gradient）を裏づける数学を与える。そして 33.4.2.6 節のランダム行列は、4.4 節における初期化時の重み行列をモデル化する。Gershgorin の定理と PageRank は古典的な名結果だが深層学習に登場する頻度は低いため、急ぐ読者は流し読みしても構わない。擬スペクトル（pseudospectrum、33.4.2.4 節）も専門的な診断道具であり、初読時には同様に飛ばしてよい。以下の数値例はすべて NumPy を用いる。

```
import numpy as np
```

33.4.1 固有値の位置特定と計算

33.4.1.1 Gershgorin の円盤

固有値を一切計算することなく、行列の固有値が大まかにどこにあるかを知ることはできるだろうか。対角行列の場合、固有値は単に対角成分そのものである。非対角成分が小さければ、固有値も対角成分の近くにとどまるだろうと直観的には予想できる。次の定理はこの直観を厳密な形で定式化するものであり、その証明はわずか 3 行で完結する。

定理（Gershgorin (Gershgorin 1931)） **$\mathbf{A} = (a_{ij})$ を任意の $n \times n$ 行列とし、各行 i に対して非対角成分の絶対値の和を $r_i = \sum_{j \neq i} |a_{ij}|$ とおく。$\mathcal{D}_i$ を、複素平面上で中心 a_{ii}、半径 r_i を持つ閉円盤とする。このとき、$\mathbf{A}$ のすべての固有値は和集合 $\bigcup_i \mathcal{D}_i$ に含まれる。**

証明： $\mathbf{A}\mathbf{v} = \lambda \mathbf{v}$ かつ $\mathbf{v} \neq \mathbf{0}$ とし、添字 i として $\mathbf{v}$ の成分の絶対値が最大になるものを選ぶ。すると $|v_i| \geq |v_j|$ がすべての j について成り立ち、かつ $v_i \neq 0$ である。第 i 行を $\mathbf{A}\mathbf{v} = \lambda \mathbf{v}$ から書き出すと、次を得る。

$$\sum_{j} a_{ij} v_j = \lambda v_i \quad \Longrightarrow \quad (\lambda - a_{ii})\, v_i = \sum_{j \neq i} a_{ij} v_j .$$

両辺を v_i で割って絶対値をとり、$|v_j / v_i| \leq 1$ がすべての j について成り立つことを用いると、次のようになる。

$$|\lambda - a_{ii}| = \left| \sum_{j \neq i} a_{ij} \, \frac{v_j}{v_i} \right| \leq \sum_{j \neq i} |a_{ij}| = r_i ,$$

これより $\lambda \in \mathcal{D}_i$ が従う。■

この証明が実際に示していることに注目しよう。各固有値は、対応する固有ベクトルの成分が最大となる行の円盤に収まるのである。どの行がそうなるかは事前にはわからないため、定理が保証するのは、すべての固有値が n 個の円盤の和集合のどこかに含まれるということだけである。具体例で確認するため、次の非対称行列を考えよう。

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 0 \\ -1 & 4 & 0.5 \\ 0.5 & 0 & 1 \end{pmatrix}. \tag{33.37}$$

3 つの円盤の中心は 4、4、1 であり、半径はそれぞれ 1、1.5、0.5 である。中心が 4 の 2 つの円盤は重なり合っている（実際には一方が他方の内部に含まれている）のに対し、中心が 1 の円盤は $|4-1| = 3 > 1.5 + 0.5$ であるため、それらから完全に離れている。実際の固有値は $3.99 \pm 0.96i$ と 1.03 であり、複素共役対は中心 4 の 2 つの円盤の内部に、実固有値は中心 1 の孤立した円盤の内部に収まっている。Figure 33.20 に、これらの円盤と固有値を並べて示す。

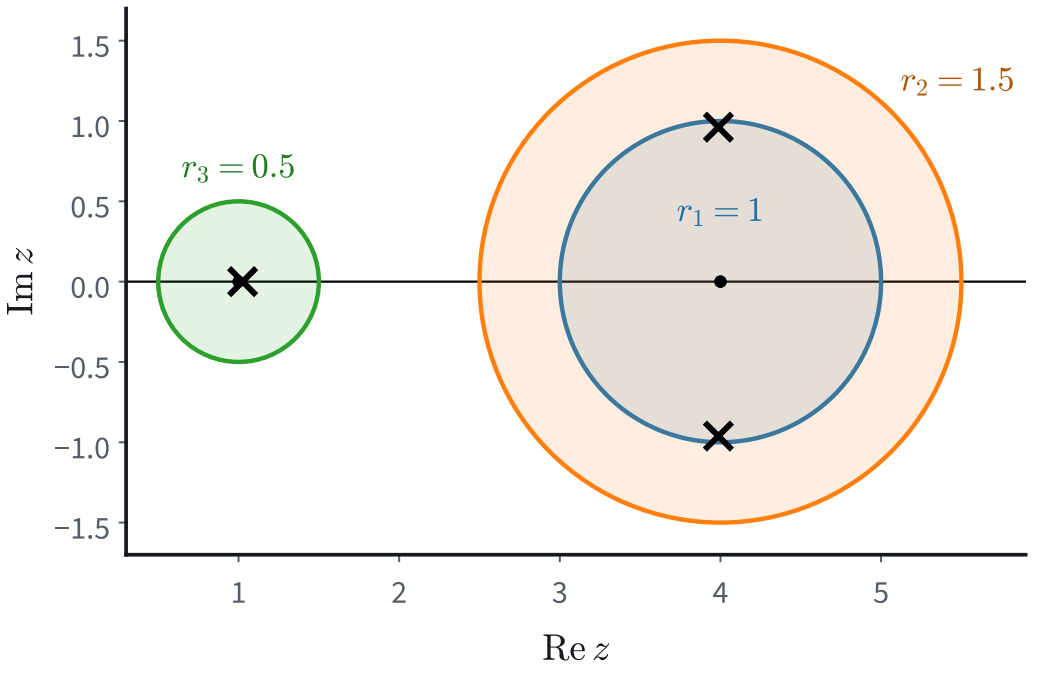


Figure 33.20: 本文中の 3×3 行列に対する Gershgorin の円盤（中心は対角成分、半径は各行の非対角成分の絶対値の和）とその固有値（十字印）。中心が 4 の重なり合う 2 つの円盤が複素共役対である 2 つの固有値を含み、中心が 1 の離れた円盤が 3 つ目の固有値を含む。

重なり合う 2 つの円盤に 2 個、孤立した円盤に 1 個の固有値が分かれて入ったのは偶然ではない。**m 個の円盤の和集合が連結であり、かつ残りのすべての円盤と交わらないならば、その領域には重複度込みでちょうど m 個の固有値が含まれる**。理由は連続性にある。非対角成分を 0 から実際の値まで連続的にスケールしていくと、固有値は対角成分から出発して連続的に移動するため、離れた円盤群のあいだにある空白を飛び越えることは決してできないからである (Horn and Johnson 2012)。中心、半径、固有値を数値計算で確かめてみよう。

```
A = np.array([[4.0, 1.0, 0.0],
              [-1.0, 4.0, 0.5],
              [0.5, 0.0, 1.0]])
centres = np.diag(A)
radii = np.abs(A).sum(axis=1) - np.abs(centres)
print('centres:', centres, '  radii:', radii)
for z in np.linalg.eigvals(A):
    print(f'eigenvalue {z:.3f}   distances to the centres: '
          f'{np.abs(z - centres).round(2)}')
```

```
centres: [4. 4. 1.]   radii: [1.  1.5 0.5]
eigenvalue 3.987+0.961j   distances to the centres: [0.96 0.96 3.14]
eigenvalue 3.987-0.961j   distances to the centres: [0.96 0.96 3.14]
eigenvalue 1.025+0.000j   distances to the centres: [2.97 2.97 0.03]
```

ここでは複素共役対が中心 4 から距離 0.96 の位置、すなわち半径 1 の内側の円盤の境界すぐ近くにあり、Gershgorin の評価がほぼタイトになりうることがわかる。

計算を一切必要としない直接の系が 1 つある。$\mathbf{A}$ の対角成分が正で**狭義対角優位**（strictly diagonally dominant）であるとき、すなわち $a_{ii} > r_i$ がすべての i について成り立つとき、すべての円盤は開右半平面の内部に完全に収まる。したがって 0 は固有値になりえず、$\mathbf{A}$ は自動的に正則（可逆）となる。さらに $\mathbf{A}$ が対称行列であれば、固有値はすべて実数かつ真に正となるため、正定値（positive definite）である。

33.4.1.2 べき乗法

Gershgorin の円盤が教えてくれるのは、固有値の大まかな位置にすぎない。では、10 億行の行列の正確な固有値を求めたい場合はどうすればよいだろうか。そのような行列を分解するのは問題外だが、疎行列であればベクトルとの積は依然として低コストに計算できる。初期ベクトル $\mathbf{v}_0$ から出発し、単に $\mathbf{A}$ を何度も繰り返し掛け続けるとどうなるだろうか。

答えを与えるのは、べき乗展開 $\mathbf{A}^k\mathbf{x} = \sum_i c_i \lambda_i^k \mathbf{v}_i$（33.3.1.3 節）である。$\mathbf{A}$ が対角化可能であり、固有ベクトル $\mathbf{v}_i$ が固有値の絶対値の大きい順に並んでいるとし、さらに最大固有値が**狭義優位**（strictly dominant）、すなわち $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ であるとする。（$\mathbf{A}$ が実行列のとき、狭義優位性から λ_1 も実数になる。複素固有値であれば、その複素共役と同じ絶対値を共有してしまうからである。）$\mathbf{v}_0 = \sum_i c_i \mathbf{v}_i$ と展開して λ_1^k をくくり出すと、次式が得られる。

$$\mathbf{A}^k \mathbf{v}_0 = \sum_{i=1}^{n} c_i \lambda_i^k \mathbf{v}_i = \lambda_1^k \left(c_1 \mathbf{v}_1 + \sum_{i \geq 2} c_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{v}_i \right). \tag{33.38}$$

比 $|\lambda_i / \lambda_1|$ はすべての $i \geq 2$ に対して 1 より真に小さいため、k が大きくなるにつれて後ろの項はすべて消えていく。$c_1 \neq 0$ である限り、括弧の中は $c_1 \mathbf{v}_1$ に収束する。したがって $\mathbf{A}^k \mathbf{v}_0$ の向きは主固有ベクトル $\mathbf{v}_1$ に近づき、その誤差は $|\lambda_2/\lambda_1|^k$ のオーダーで減衰する。一方、その長さはステップごとに $|\lambda_1|$ に漸近する倍率でスケールされる。成分のオーバーフローやアンダーフローを防ぐため、乗算のたびにベクトルを単位長に正規化し直す。この単純なループが**べき乗法**である (Golub and Van Loan 1996)。$\mathbf{A}$ が対称行列のときは、固有値そのものについてさらに速い収束が得られる。現在の反復ベクトルのレイリー商（Rayleigh quotient）は λ_1 に二乗のレート $|\lambda_2/\lambda_1|^{2k}$ で収束する。対称行列の固有ベクトルは互いに直交するため、大きさ δ の向きのずれが生じてもレイリー商は $O(\delta^2)$ しか変化しないからである（演習問題 2）。

具体的な 2×2 の例で、この両方の収束レートを確認できる。1.3 節では、$\mathbf{S} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ を $\mathbf{e}_1$ から反復したベクトル列を計算し、$(2,1)$、$(5,4)$、$(14,13)$、一般に $\left((3^k+1)/2,\ (3^k-1)/2\right)$ となることを見た。単位固有ベクトル $\mathbf{v}_1 = (1,1)/\sqrt{2}$ と $\mathbf{v}_2 = (1,-1)/\sqrt{2}$ を用いれば $\mathbf{e}_1 = (\mathbf{v}_1 + \mathbf{v}_2)/\sqrt{2}$ と書けるため、次のようになる。

$$\mathbf{S}^k \mathbf{e}_1 = \frac{3^k \mathbf{v}_1 + \mathbf{v}_2}{\sqrt{2}}, \qquad \tan\angle(\mathbf{S}^k\mathbf{e}_1, \mathbf{v}_1) = 3^{-k}, \qquad 3 - R(\mathbf{S}^k\mathbf{e}_1) = \frac{2}{9^k + 1}.$$

$|\lambda_2/\lambda_1| = 1/3$ であるため、向きの誤差はステップごとに 3 分の 1 に減少する。実際、$\mathbf{v}_1$ とのなす角の正弦は 0.316、0.110、0.037（$k = 1, 2, 3$）となる。一方、レイリー商の誤差はステップごとに 9 分の 1 に減少し、0.2、0.024、0.0027 となる。Figure 33.21 は両方の曲線を対数スケールでプロットしたものである。

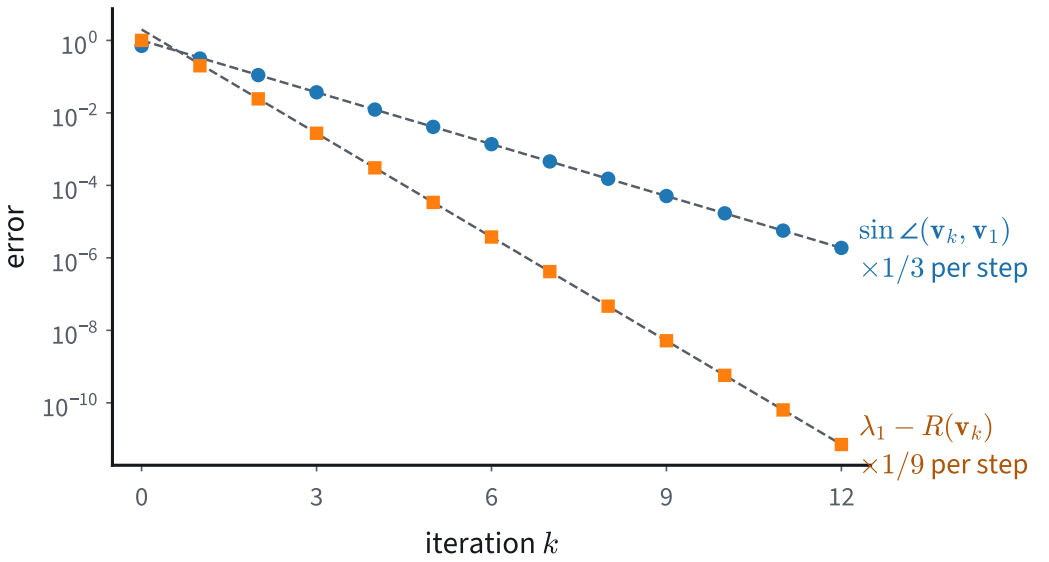


Figure 33.21: $\mathbf{S} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ に対する $\mathbf{e}_1$ からのべき乗法。向きの誤差はステップごとに 3 分の 1 に減少するのに対し、固有値の推定誤差は直交する固有ベクトルに由来する二乗のレートにより、ステップごとに 9 分の 1 に改善する。破線は 3^{-k} と $2\cdot 9^{-k}$ を示す。

べき乗法が失敗するのはどのような場合だろうか。Equation 33.38 における 2 つの仮定が鍵を握る。第 1 に、初期ベクトルは $\mathbf{v}_1$ 方向に非ゼロの成分を持つ必要がある（$c_1 \neq 0$）。$\mathbf{v}_0$ をランダムに選べば、これは確率 1 で保証される。第 2 に、最大固有値が狭義優位でなければならない。最大の絶対値を複素共役対が共有している場合、反復ベクトルの主成分は 1 本の直線に落ち着くことなくいつまでも回転し続ける。すると連続するノルムの比は、33.3.1.6 節で扱った例題の行列とまったく同様に永遠に振動し続ける。ただしその場合でも、幾何平均 $\|\mathbf{A}^k\mathbf{v}_0\|^{1/k}$ は最大の絶対値に収束する。

すべての成分が真に正である行列では、この問題は完全に回避される。33.4.1.3 節で見るとおり、ペロン・フロベニウスの定理によって、実数かつ単根で狭義優位な固有値の存在が保証されるからである。これを確かめるため、べき乗法を 200 ステップ実行してみよう。5×5 のランダム行列の成分を $[0, 1]$ から一様に抽出し、ランダムなベクトルから出発して、最終的なノルム比を真の最大固有値の絶対値と比較する。

```
A = np.random.rand(5, 5)                # positive entries
v = np.random.randn(5)
for _ in range(200):
    Av = A @ v
    ratio = np.linalg.norm(Av) / np.linalg.norm(v)
    v = Av / np.linalg.norm(Av)
mods = np.sort(np.abs(np.linalg.eigvals(A)))[::-1]
print(f'norm ratio after 200 steps = {ratio:.10f}')
print(f'largest |eigenvalue|       = {mods[0]:.10f}')
print(f'|lambda_2| / |lambda_1|    = {mods[1] / mods[0]:.3f}')
```

```
norm ratio after 200 steps = 2.5650940439
largest |eigenvalue|       = 2.5650940439
|lambda_2| / |lambda_1|    = 0.127
```

どの試行でも、ノルム比は表示されたすべての桁で最大固有値の絶対値と一致する。この速い収束がどれほど典型的なものかを見るため、ギャップ比 $|\lambda_2/\lambda_1|$ をこのような正行列 10,000 個にわたって計算してみよう。比較のため、正値性の保証がない標準ガウス分布の成分を持つ行列も 10,000 個生成し、主固有値がどのくらいの頻度で複素共役対をなすかを調べる。

```
mods = np.sort(np.abs(np.linalg.eigvals(np.random.rand(10000, 5, 5))), axis=1)
print('uniform entries, gap ratio |lambda_2| / |lambda_1|, 5% and 95% quantiles:',
      np.quantile(mods[:, -2] / mods[:, -1], [0.05, 0.95]).round(2))
lam = np.linalg.eigvals(np.random.randn(10000, 5, 5))
top = lam[np.arange(10000), np.abs(lam).argmax(axis=1)]   # a dominant eigenvalue
print('Gaussian entries, fraction with a complex dominant pair:',
      np.mean(top.imag != 0).round(2))
```

```
uniform entries, gap ratio |lambda_2| / |lambda_1|, 5% and 95% quantiles: [0.14 0.37]
Gaussian entries, fraction with a complex dominant pair: 0.43
```

一様な正の成分を持つ場合、10 回中 9 回の試行でギャップ比はおよそ 0.14 から 0.37 のあいだに収まるため、200 回の反復によって向きの誤差は浮動小数点精度（floating-point precision）をはるかに下回るまで小さくなる。対照的にガウス分布の成分を持つ場合は、10 回中約 4 回の試行で主固有値が複素共役対となり、そのような行列ではノルム比が決して収束しない。

33.4.1.3 PageRank

非負の成分を持つ行列では、主固有ベクトルこそが求めたい対象そのものである場合が多く、その計算にはべき乗法が用いられる。この固有ベクトルがいつ一意で扱いやすい性質を持つかを述べるため、有向辺 $j \to i$ を $a_{ij} > 0$ のときに張る有向グラフにおいて、すべての頂点から他のすべての頂点へ到達できるとき、その非負行列を**既約**（irreducible）と呼ぶことにする。**ペロン・フロベニウスの定理**（Perron–Frobenius theorem）(Horn and Johnson 2012) によれば、任意の既約な非負行列は全固有値の最大絶対値に等しい実数の正の固有値を持ち、この固有値は単根であり、対応する固有ベクトルはすべての成分が真に正となるように選べる。さらに行列のすべての成分が真に正であれば、この固有値は狭義優位でもあり、他のすべての固有値の絶対値はこれより真に小さくなる。

Google の初期の PageRank アルゴリズム (Brin and Page 1998) は、ウェブのランキングをまさにこのような固有ベクトル問題へと帰着させる。リンク $1 \to 2$、$2 \to 1$、$2 \to 3$ を持つ 3 ページからなる小規模なウェブを考えよう。ランダムサーファーは出力リンクの中から一様ランダムに 1 つを選んで辿る。ページ 3 には出力リンクがないため、サーファーがページ 3 から全ページの

いずれかへ等確率でジャンプするものとして、この行き止まりを補正する。補正後の遷移行列 $\mathbf{P}$ は列確率行列（column-stochastic matrix）となり、第 j 列がページ j からクリックして遷移する確率を表す。

$$\mathbf{P} = \begin{pmatrix} 0 & \tfrac{1}{2} & \tfrac{1}{3} \\ 1 & 0 & \tfrac{1}{3} \\ 0 & \tfrac{1}{2} & \tfrac{1}{3} \end{pmatrix}.$$

長期的に見てサーファーが各ページに滞在する時間の割合は、定常分布（stationary distribution）π、すなわち $\mathbf{P}\pi = \pi$ を満たすベクトルになる。これは手計算で解くことができる。第 1 行と第 3 行はいずれも $\pi_1 = \tfrac{1}{2}\pi_2 + \tfrac{1}{3}\pi_3 = \pi_3$ を与え、$\pi_1 = \pi_3$ を第 1 行に代入すると $\pi_2 = \tfrac{4}{3}\pi_3$ が得られる。成分の和が 1 になるよう正規化すると、$\pi = (0.3, 0.4, 0.3)$ となる。ページ 2 が最上位になるのは、ページ 1 がすべてのクリックをそこへ送るのに対し、ページ 2 はクリックをページ 1 とページ 3 に折半するからである。$\mathbf{P}$ の固有値は 1、$-\tfrac{2}{3}$、0 であるため、任意の初期分布からのべき乗法はステップあたり $2/3$ のレートで収束する。

実は、**すべての**列確率行列は最大絶対値を持つ固有値として 1 を持つ。第 1 に、すべての列の和が 1 であることから $\mathbf{1}^{\top}\mathbf{P} = \mathbf{1}^{\top}$ が成り立つ。これは 1 が $\mathbf{P}^{\top}$ の固有値であり、したがって $\mathbf{P}$ の固有値でもあることを意味する（$\det \mathbf{X}^{\top} = \det \mathbf{X}$ であるため、行列とその転置は同じ特性多項式を共有する）。第 2 に、Gershgorin の定理を $\mathbf{P}^{\top}$（その各行は $\mathbf{P}$ の各列である）に適用すると、すべての固有値は中心 $p_{ii} \geq 0$、半径 $1 - p_{ii}$ の円盤内に収まり、そのような円盤はすべて単位円盤の内部に含まれる。

しかし現実のウェブは 3 ページの例よりもはるかに複雑であり、リンクの巡回路や、サーファーが抜け出せない孤立したページ群を含んでいる。そのため、補正後の行列 $\mathbf{P}$ が既約でなくなったり、絶対値 1 の固有値を複数持ったりすることがある。たとえば 4 ページの単純な巡回路は固有値 ± 1 と $\pm i$ を持つため、減衰なしのべき乗法は決して収束しない（演習問題 4）。PageRank は、次の**減衰付き**（damped）行列を反復することで、これら両方の問題を一挙に解決する。

$$\mathbf{G} = \alpha \mathbf{P} + (1 - \alpha)\tfrac{1}{n}\mathbf{1}\mathbf{1}^{\top}, \qquad \alpha = 0.85,$$

この行列は、確率 0.85 でランダムなリンクを辿り、確率 0.15 で一様ランダムに選んだページへテレポートするサーファーを表す。$\mathbf{G}$ のすべての成分が真に正であるため、構成から直ちにペロン・フロベニウスの定理が適用できる。さらに都合のよいことに、Haveliwala and Kamvar (2003) は $|\lambda_2(\mathbf{G})| \leq \alpha$ を示した。つまりリンクグラフがどれほど入り組んでいても、べき乗法の誤差はステップごとに少なくとも 0.85 倍以下に縮小する。このたった 1 つの上界がウェブ規模の PageRank を実用的なものにした。グラフに何十億ページ含まれていようとも、50 回の反復で誤差は 3000 分の 1 未満にまで縮小するのである（$0.85^{50} \approx 3 \times 10^{-4}$）。

ウェブ規模では、$\mathbf{P}$ はリンクごとに 1 成分を持つ疎行列として保持され、密行列 $\mathbf{G}$ を陽に構築することは決してない。任意の確率ベクトル $\mathbf{p}$（$\mathbf{1}^{\top}\mathbf{p} = 1$）に対して、$\mathbf{G}\mathbf{p} = \alpha \mathbf{P}\mathbf{p} + (1-\alpha)\tfrac{1}{n}\mathbf{1}$ はわずか 1 回の疎行列・ベクトル積と定数シフトだけで計算できる。出力リンクを持たない行き止まりページの補正もランク 1 の項であり、まったく同様に処理される。

33.4.2 反復と安定性

33.4.2.1 スペクトル半径

べき乗法は 1 つの数、すなわち固有値の最大絶対値を浮き彫りにした。この数は**スペクトル半径**（spectral radius）と呼ばれ、$\rho(\mathbf{A}) = \max_i |\lambda_i|$ と表される。スペクトル半径は、単一の固定され

た行列のべき乗における長期的な増大率を支配する。ゲルファントの公式（Gelfand's formula）はこの事実を厳密に表しており、任意の正方行列 $\mathbf{A}$ と任意の行列ノルムに対して次が成り立つ。

$$\lim_{k \to \infty} \|\mathbf{A}^k\|^{1/k} = \rho(\mathbf{A}).$$

この主張のうち比較的容易な片側の不等式、すなわち $\rho(\mathbf{A}) \le \|\mathbf{A}^k\|^{1/k}$ がすべての $k \ge 1$ について成り立つことは、演習問題 1 で扱う。

ゲルファントの公式が教えて**くれない**のは、固定された有限のステップ k で何が起きるかであり、そのギャップが問題になる行列にはすでに 2 つ出会っている。せん断行列 $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ は $\rho(\mathbf{T}) = 1$ を満たすが、そのべき乗 $\mathbf{T}^k = \left(\begin{smallmatrix} 1 & k \\ 0 & 1 \end{smallmatrix}\right)$ は際限なく増大するため、$\rho \le 1$ であっても反復列が有界にとどまるとすら限らない。ジョルダンブロック $\mathbf{J}_2(0.99)$ は $\rho < 1$ でありいずれ 0 へ減衰するはずだが、そのべき乗は減少に転じる前にノルムが 37 まで上昇するため、$\rho < 1$ であっても大きな過渡的増大は排除できない。では、遠い極限だけでなく**すべての**ステップを固有値が制御するのは、どのような場合だろうか。

33.4.2.2 正規行列

せん断とジョルダンブロックという 2 つの反例はいずれも欠陥行列（defective matrix）であり、基底をなすのに十分な数の固有ベクトルを持たない。そこで一気に正反対の極端な場合へ進み、固有ベクトルからなる**正規直交**（orthonormal）基底を持つ行列を考えよう。そのような行列は互いに直交する方向に沿って独立に作用するため、すべてのステップにおいて固有値が振る舞いのすべてを語るはずである。

実行列は、自身の転置と可換であるとき、すなわち $\mathbf{A}^\top \mathbf{A} = \mathbf{A}\mathbf{A}^\top$ を満たすとき**正規**（normal）であるという（複素行列の場合は、転置を共役転置（conjugate transpose）$\mathbf{A}^*$ に置き換え、$\mathbf{A}^*\mathbf{A} = \mathbf{A}\mathbf{A}^*$ を課す）。対称行列、直交行列、そして回転・スケーリングブロック $r\mathbf{R}_\theta$（33.3.1.6 節）はすべて正規行列である。対照的に、これまで例として用いてきた $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ は正規ではない。実際、$\mathbf{A}\mathbf{A}^\top = \left(\begin{smallmatrix} 5 & 5 \\ 5 & 10 \end{smallmatrix}\right)$ であるのに対し、$\mathbf{A}^\top\mathbf{A} = \left(\begin{smallmatrix} 2 & -1 \\ -1 & 13 \end{smallmatrix}\right)$ となる。この非可換性こそが、反復点列が円ではなく楕円に沿って渦を巻いた理由である。

複素スペクトル定理（complex spectral theorem）(Horn and Johnson 2012) は、この行列のクラスを完全に特徴づける。すなわち、行列が正規であることと、$\mathbb{C}$ 上の正規直交固有基底を持つこと、言い換えれば $\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^*$ とユニタリ行列 $\mathbf{U}$（$\mathbf{U}^*\mathbf{U} = \mathbf{I}$）を用いて表せることは同値である。証明は 33.3.2.1 節の実スペクトル定理と同じ帰納法に従う。

1 ステップにおける最悪ケースの増幅率を測るため、$\|\mathbf{X}\|_2 = \max_{\|\mathbf{x}\|=1} \|\mathbf{X}\mathbf{x}\|$ と書いて $\mathbf{X}$ の**スペクトルノルム**（spectral norm）を表すことにする。これは $\mathbf{X}$ がベクトルを引き伸ばしうる最大の倍率である。ユニタリ行列による基底変換はユークリッド長を保つため、正規行列の任意のべき乗は次式を満たす。

$$\|\mathbf{A}^k\|_2 = \|\mathbf{U}\Lambda^k\mathbf{U}^*\|_2 = \max_i |\lambda_i|^k = \rho(\mathbf{A})^k. \tag{33.39}$$

正規行列では、漸近的にだけでなく、個々のすべてのステップ k において固有値が最悪ケースの増幅率を制御する。

$\mathbf{A}$ が対角化可能（$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$）ではあるが正規ではない場合はどうなるだろうか。このとき固有ベクトル行列 $\mathbf{V}$ はもはや直交行列ではなく、スペクトルノルムの劣乗法性（submultiplicativity）からは次の評価式（上界）しか得られない。

$$\|\mathbf{A}^k\|_2 \le \underbrace{\|\mathbf{V}\|_2\|\mathbf{V}^{-1}\|_2}_{\kappa_2(\mathbf{V})}\,\rho(\mathbf{A})^k. \tag{33.40}$$

前因子 $\kappa_2(\mathbf{V})$ は固有ベクトル行列の条件数（condition number、33.5.3.2 節）である。この値は固有ベクトルが直交するとき 1 に等しく、2 つの固有ベクトルがほぼ同じ向きを指すときいくらでも大きくなる。この前因子が許容する過渡的増大には、相殺という明快な幾何学的原因がある。固有ベクトルどうしがほぼ平行であると、通常の単位ベクトルであっても、互いにほぼ打ち消し合う巨大な固有成分の線形結合としてしか表せなくなる。そこに $\mathbf{A}$ を繰り返し作用させると、各成分が異なるレートで減衰するため微妙な打ち消し合いが崩れ、最終的に全体の指数減衰が支配的になる前に、反復ベクトルの長さが大きく膨れ上がるのである。

33.4.2.3 過渡的増大

この過渡的増大を示す最も単純な行列は、次の 2×2 上三角行列である。

$$\mathbf{M} = \begin{pmatrix} 0.9 & 4 \\ 0 & 0.8 \end{pmatrix}. \tag{33.41}$$

この行列の固有値は対角成分である 0.9 と 0.8 である。どちらも単位円の内側に十分収まっており、互いに異なるため $\mathbf{M}$ は対角化可能である。固有ベクトルを手計算で求め、単位ベクトル $\mathbf{e}_2 = (0,1)$ をそれらで展開してみよう。$\lambda = 0.9$ に対しては、$\mathbf{M} - 0.9\mathbf{I}$ の第 2 行が $-0.1 v_2 = 0$ となることから $\mathbf{v}_1 = (1,0)$ を得る。$\lambda = 0.8$ に対しては、$\mathbf{M} - 0.8\mathbf{I}$ の第 1 行が $0.1 v_1 + 4 v_2 = 0$ となることから $\mathbf{v}_2 = (-40, 1)$ を得る。これら 2 つの向きがいかに近いか注目してほしい。両者のなす角はわずか $\arctan(1/40) \approx 1.4^\circ$ しかない。このほぼ平行な基底で $\mathbf{e}_2$ を展開すると $\mathbf{e}_2 = 40\mathbf{v}_1 + \mathbf{v}_2$ となり、固有ベクトルによるべき乗展開（33.3.1.3 節）を適用すると次式が得られる。

$$\mathbf{M}^k\mathbf{e}_2 = 40\cdot 0.9^k\,\mathbf{v}_1 + 0.8^k\,\mathbf{v}_2 = \begin{pmatrix} 40\,(0.9^k - 0.8^k) \\ 0.8^k \end{pmatrix}. \tag{33.42}$$

$k = 0$ において、長さ 1 のベクトルが、長さおよそ 40 の 2 つの固有成分の差として表されている。両成分ともステップごとに減衰するが、0.8^k は 0.9^k よりも速く減少するため打ち消し合いが崩れ、和は最初かえって増大する。第 1 座標 $40(0.9^k - 0.8^k)$ が最大になるのは k に関する導関数が 0 になる点、すなわち $(9/8)^k = \ln 0.8 / \ln 0.9$ を満たす $k \approx 6.4$ であり、その高さは約 10.8 に達する。Figure 33.22 に、$k = 0$ と $k = 6$ における 2 つの成分を示す。比較として、単位固有ベクトルを用いた一般の上界 Equation 33.40 では $\kappa_2(\mathbf{V}) = 80$ となり、$\|\mathbf{M}^6\|_2 \le 80\cdot 0.9^6 \approx 42.5$ という予測を与える。この上界は確かに成り立っているものの、真のピーク値 10.8 と比べると約 4 倍控えめな評価になっている。

この計算を数値的に検証し、$\mathbf{M}$ をまったく同じ固有値を持つ正規な対角行列 $\Lambda = \mathrm{diag}(0.9, 0.8)$ と比較してみよう。

```
M = np.array([[0.9, 4.0], [0.0, 0.8]])
Lam = np.diag([0.9, 0.8])                  # same eigenvalues, normal
lam, V = np.linalg.eig(M)                  # unit eigenvectors as columns
print('eigenvalues:', lam)
print('coefficients of e_2 on the unit eigenvectors:',
      np.linalg.solve(V, [0., 1.]))
ks = np.arange(61)
```

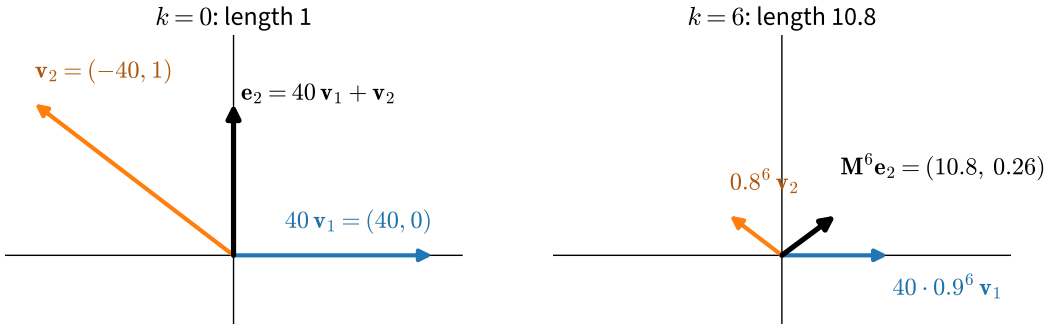


Figure 33.22: 減衰前の $\mathbf{M}^k\mathbf{e}_2$ の増大。左：$k = 0$ では、単位ベクトル $\mathbf{e}_2$ はほぼ打ち消し合う長さ約 40 の 2 つの固有成分の差として表される。右：6 ステップ後には 2 つの成分がそれぞれ 0.9^6 倍と 0.8^6 倍に減衰して相殺が崩れ、元の 10 倍を超える長さのベクトルが生じる。縦軸は約 30 倍に引き伸ばして描かれており、等倍で描けば $\mathbf{v}_1$ と $\mathbf{v}_2$ のなす角はわずか 1.4° しかない。

```
norm_M = [np.linalg.norm(np.linalg.matrix_power(M, k), 2) for k in ks]
norm_L = [np.linalg.norm(np.linalg.matrix_power(Lam, k), 2) for k in ks]
k_peak = int(np.argmax(norm_M))
print(f'max ||M^k|| = {norm_M[k_peak]:.3f} at k = {k_peak};  '
      f'max ||Lambda^k|| = {max(norm_L):.3f}')
print(f'kappa_2(V) = {np.linalg.cond(V):.1f}')
```

```
eigenvalues: [0.9 0.8]
coefficients of e_2 on the unit eigenvectors: [40.         40.01249805]
max ||M^k|| = 10.788 at k = 6;  max ||Lambda^k|| = 1.000
kappa_2(V) = 80.0
```

単位固有ベクトルで展開すると、$\mathbf{e}_2$ は絶対値が約 40 の 2 つの係数を持ち、これは $40\mathbf{v}_1$ と $\mathbf{v}_2$ の長さに一致する。スペクトルノルム $\|\mathbf{M}^k\|_2$ は最大 10.8（$k = 6$）まで上昇するのに対し、$\|\Lambda^k\|_2$ は初期値である 1 を決して超えない。両行列は同じ固有値を共有しているにもかかわらず、正規行列が最初のステップから縮小し続ける一方、非正規行列は減衰する前に 10 倍以上に増大する。Figure 33.23 の左パネルは両者の軌跡を対数スケールで描いたものであり、ピークを過ぎた後はゲルファントの公式が保証するとおり、どちらの曲線もやがて同じ傾き $\log 0.9$ で下降していく。

33.4.2.4 擬スペクトル

本項では新たな現象ではなく非正規性の診断手法を導入するため、初読時には飛ばしても構わない。過渡的増大と並ぶ非正規性のもう 1 つの兆候は、微小な摂動に対する固有値の極端な敏感さである。この敏感さを可視化するため、ε **擬スペクトル**は、$\mathbf{A} + \mathbf{E}$ の固有値となるすべての複素数（ただし $\mathbf{E}$ はスペクトルノルムが高々 ε のある摂動）を集めた集合として定義される。都合のよいことに、同じ集合を**レゾルベント** $(z\mathbf{I} - \mathbf{A})^{-1}$ から直接計算できる。

$$\begin{aligned}\Lambda_\varepsilon(\mathbf{A}) &= \{z \in \mathbb{C} : z \text{ is an eigenvalue of } \mathbf{A} + \mathbf{E} \text{ for some } \|\mathbf{E}\|_2 \le \varepsilon\} \\ &= \{z \in \mathbb{C} : \|(z\mathbf{I} - \mathbf{A})^{-1}\|_2 \ge 1/\varepsilon\},\end{aligned} \tag{33.43}$$

ここで、$\mathbf{A}$ 自身の固有値においてはレゾルベントノルムを無限大と約束する。これら 2 つの定義が同値であることの証明は、演習問題 3 とする。正規行列の場合、点 z におけるレゾルベントノ

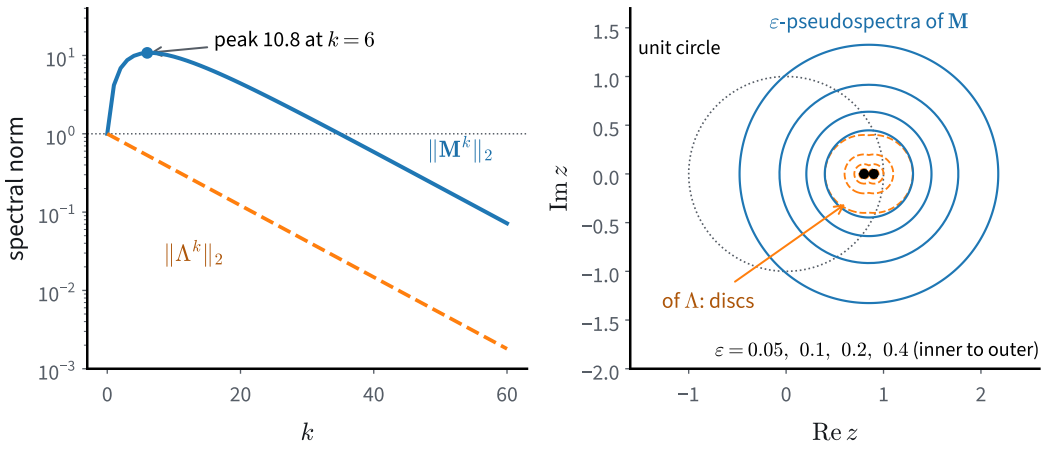


Figure 33.23: 左：同じ固有値 0.9 と 0.8 を持つ $\|\mathbf{M}^k\|_2$ と $\|\Lambda^k\|_2$。$\mathbf{M}$ は 10.8（$k = 6$）まで上昇した後、両曲線とも傾き $\log 0.9$ で減衰する。右：ε 擬スペクトル（$\varepsilon = 0.05, 0.1, 0.2, 0.4$）。正規行列 Λ の周りでは単に半径 ε の円盤になるのに対し、$\mathbf{M}$ の周りでは単位円を大きく越えて張り出す。

ルムは単に z から最も近い固有値までの距離の逆数になるため、$\Lambda_\varepsilon(\mathbf{A})$ は各固有値を中心とする半径 ε の円盤の和集合にすぎない。しかし $\mathbf{M}$ のような非正規行列では、単位円上の $z = 1$ で確かめられるように、レゾルベントノルムが桁違いに大きくなりうる。

```
I = np.eye(2)
print(f'||(I - M)^-1||      = {np.linalg.norm(np.linalg.inv(I - M), 2):.1f}')
print(f'||(I - Lambda)^-1|| = {np.linalg.norm(np.linalg.inv(I - Lam), 2):.1f}')
```

```
||(I - M)^-1||      = 200.3
||(I - Lambda)^-1|| = 10.0
```

Λ に対するレゾルベントノルムは 10 であり、これは距離 0.1（点 1 から固有値 0.9 までの距離）の逆数である。一方、$\mathbf{M}$ に対しては 200.3 まで跳ね上がる。つまり大きさ $1/200.3 \approx 0.005$ のごく小さな摂動だけで $\mathbf{M}$ の固有値は単位円上まで押し出されてしまうのに対し、Λ ではその 20 倍にあたる大きさ 0.1 の摂動が必要になる（Figure 33.23 右パネル）。浮動小数点の丸め誤差は行列成分への微小な摂動として働くため、非正規性の強い行列では数値計算された固有値が驚くほど不正確になることがある。33.6 節では、Kreiss の行列定理を通じてこの話題をさらに掘り下げる。

33.4.2.5 ヤコビ行列の積

ここまでは、ベクトルに**同じ**行列を繰り返し掛けてきた。しかしリカレントニューラルネットワーク（8.5 節）では、その前提さえ崩れる。リカレント層は隠れ状態を $\mathbf{h}_t = \phi(\mathbf{W}\mathbf{h}_{t-1} + \cdots)$ と更新し、各ステップで同じ重み行列 $\mathbf{W}$ を用いるが、通時的誤差逆伝播法（backpropagation through time、BPTT）は勾配にステップごとのヤコビ行列の系列 $\mathbf{D}_t = \partial \mathbf{h}_t / \partial \mathbf{h}_{t-1}$ を掛けていく。

$$\nabla_{\mathbf{h}_0} \mathcal{L} = \mathbf{D}_1^\top \mathbf{D}_2^\top \cdots \mathbf{D}_T^\top \nabla_{\mathbf{h}_T} \mathcal{L}.$$

非線形性 ϕ はタイムステップごとに異なる隠れ状態で評価されるため、ヤコビ行列 $\mathbf{D}_1, \ldots, \mathbf{D}_T$

は互いに異なる行列となり、単一の行列のべき乗に関する理論はもはや適用できない。

実際、個々の因子のスペクトル半径からは、それらの積について何ひとつ知ることはできない。次の 2 つの 2×2 行列を考えてみよう。

$$\mathbf{D}_1 = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{D}_2 = \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix}, \qquad \mathbf{D}_2\mathbf{D}_1 = \begin{pmatrix} 0 & 0 \\ 0 & 4 \end{pmatrix}.$$

$\mathbf{D}_1$ と $\mathbf{D}_2$ はどちらも $\rho = 0$ のべき零（nilpotent）行列であり、それぞれのべき乗は 2 ステップ目にはゼロ行列になる。ところが両者の積は $\rho = 4$ を持つ。$\mathbf{D}_1$ が第 2 座標を 2 倍して第 1 座標にコピーし、$\mathbf{D}_2$ がそれをさらに 2 倍して元の座標へコピーし戻すからである。

したがって、反復される線形変換を調べる際には、3 つの異なる量を明確に区別しておくとよい。**スペクトル半径**は、1 つの固定された行列のべき乗における漸近的な増大率を与える。**スペクトルノルム**は、劣乗法性により $\|\mathbf{D}_T \cdots \mathbf{D}_1\|_2 \le \prod_t \|\mathbf{D}_t\|_2$（上の例では $4 \le 2 \cdot 2$）が成り立つため、変化する行列の任意の有限積を上から抑える。最後に、**最大リアプノフ指数**（top Lyapunov exponent）は次の式で定義される。

$$\mu = \lim_{T \to \infty} \frac{1}{T} \log \|\mathbf{D}_T \cdots \mathbf{D}_1 \mathbf{x}_0\|$$

この量は、変化する行列の長い積に沿った 1 ステップあたりの平均対数増大率を測るものである。各因子が独立同分布のランダム行列であるとき、緩やかな条件のもとでこの極限は確率 1 で存在する (Furstenberg and Kesten 1960)。

勾配が $\mu > 0$ であるヤコビ行列の系列を逆方向に伝播すると指数関数的に増大して爆発し、$\mu < 0$ であれば指数関数的に縮小して消失する (Bengio et al. 1994; Pascanu et al. 2013)。深層学習における標準的な対策が、固有値ではなくノルムやヤコビ行列そのものを対象としていることに注目しよう。直交初期化（orthogonal initialization）(Saxe et al. 2014) は重み行列を直交行列に選ぶことで、線形漸化式がすべてのステップであらゆるベクトルのユークリッド長を保つようにする。これは $\rho = 1$ よりもはるかに強い保証である。勾配クリッピングは、誤差逆伝播法の直後に勾配のノルムを直接頭打ちにする。LSTM や GRU のようなゲート付きアーキテクチャはヤコビ行列自体を変化させ、選択された方向が 1 に近い倍率で通過できる経路を作り出す（8.5.1.1 節）。

33.4.2.6 初期化時のランダム行列

初期化したばかりのニューラルネットワークのランダムな重み行列において、スペクトル半径とスペクトルノルムはどのくらいの大きさになるだろうか。独立な標準ガウス分布に従う成分を持つ $n \times n$ 行列 $\mathbf{A}$ を考えよう。これは実 **Ginibre アンサンブル**として知られるものである (Ginibre 1965)。$n \to \infty$ のとき、$1/\sqrt{n}$ でリスケールした固有値は複素平面上の単位円盤全体に一様に広がる。これが**円則**（circular law）であり、Tao and Vu (2010) は平均 0、分散 1 の独立かつ同一分布に従う成分に対して一般の場合にこれを証明した。円則はほとんどすべての固有値がどこに分布するかを教えてくれるが、それ単体では少数の外れ値が飛び出す可能性までは排除しない。しかし別の定理によって外れ値も単位円の外へは出ないことが示されており、確率 1 で $\rho(\mathbf{A})/\sqrt{n} \to 1$ が成り立つ (Geman 1986)。

一方、スペクトルノルムはまったく異なる振る舞いを示し、$\|\mathbf{A}\|_2/\sqrt{n} \to 2$ となる (Yin et al. 1988)。これはマルチェンコ・パスツール則（33.3.2.4 節）においてアスペクト比を $c = 1$ とした特別な場合にあたる。すなわち、$\frac{1}{n}\mathbf{A}^\top\mathbf{A}$ の最大固有値は上端 $(1 + \sqrt{c})^2 = 4$ に収束し、$\|\mathbf{A}\|_2^2$ は単にその固有値の n 倍に等しい。$n = 1000$ 程度でも、1 回ランダムに生成しただけで両方の漸近極限のすぐ近くに収まる。

```
n = 1000
A = np.random.randn(n, n)
print(f'rho(A) / sqrt(n)       = {np.abs(np.linalg.eigvals(A)).max() / n ** 0.5:.3f}')
print(f'sigma_max(A) / sqrt(n) = {np.linalg.norm(A, 2) / n ** 0.5:.3f}')
```

```
rho(A) / sqrt(n)       = 1.021
sigma_max(A) / sqrt(n) = 1.998
```

n が大きいとき、スペクトル半径は $\sqrt{n}$ から数パーセント以内に収まる一方、最大の引き伸ばし率は $2\sqrt{n}$ の近くに位置する。つまりランダムなガウス行列は、固有値から予想される値の 2 倍も特定のベクトルを引き伸ばすのである。標準的な初期化手法がランダムな重みを $\sqrt{\text{fan-in}}$ で割るのは、これとはまったく別の理由による。成分の分散を $1/n$ にすると、$\mathbb{E}\|\mathbf{Ax}\|^2 = \|\mathbf{x}\|^2$ が任意の固定されたベクトル $\mathbf{x}$ に対して成り立ち、二乗長が平均的に保たれる。スペクトル半径が 1 の近くになるのはその副産物にすぎず、33.4.2.5 節で見たように、それによって異なる行列の積が制御されるわけではない。驚くべきことに、まさにこのスケーリングを持つ独立な行列を数多く掛け合わせると、個々の因子は平均的に二乗長を保っているにもかかわらず、典型的なベクトルは**縮小**して $\mu < 0$ となる。その理由は対数関数の凹性にある。単位ベクトル $\mathbf{x}$ に対して、$\log \|\mathbf{Ax}\|^2$ の期待値は $\|\mathbf{Ax}\|^2$ の期待値の対数（すなわち $\log 1 = 0$）より真に小さくなり、深い積に沿って足し合わされるのはこの対数のほうだからである（演習問題 5）。

非正規行列 $\mathbf{M}$ とランダムなガウス行列は、どちらも同じ結論を指し示している。有限ステップで行列がベクトルをどれだけ引き伸ばすかについて固有値が教えてくれることはほとんどなく、そもそも長方形行列には固有値が存在さえしない。任意の形状を持つあらゆる行列に対して引き伸ばしを直接測るため、次は 33.5 節の特異値へと進む。

33.4.3 まとめ

行列を分解しなくても、Gershgorin の円盤を用いれば成分から直接固有値の位置を絞り込むことができ、べき乗法を用いれば等比レート $|\lambda_2/\lambda_1|$ で主固有対を抽出できる。これが PageRank を支える仕組みである。スペクトル半径 $\rho(\mathbf{A})$ が支配するのは単一の固定された行列のべき乗における長期的なレートだけであり、それ以上のことは語らない。非正規行列は減衰に転じる前に劇的に増大することがあり、また変化する行列の積については個々の因子の固有値から何もわからない。変化する写像の積に対しては、スペクトルノルムが有限ステップを上から抑え、最大リアプノフ指数が長期的な平均レートを与える。初期化時の $n \times n$ ランダムガウス行列は、固有値と引き伸ばし率のあいだの乖離を具体的に示している。そのスペクトル半径は $\sqrt{n}$ に近い一方、スペクトルノルムは $2\sqrt{n}$ に近くなるのである。

33.4.4 演習問題

1. **スペクトル半径とノルム。**スペクトル半径は行列のすべてのべき乗を下から抑え、正規行列に対しては上からも抑える。
    1. $\rho(\mathbf{A}) \le \|\mathbf{A}^k\|_2^{1/k}$ が任意の正方行列 $\mathbf{A}$ とすべての $k \ge 1$ に対して成り立つことを示せ。
    2. 式変形 Equation 33.39 では 2 つの事実を証明なしに用いている。$\|\mathbf{UXU}^*\|_2 = \|\mathbf{X}\|_2$ が任意のユニタリ行列 $\mathbf{U}$ と任意の正方行列 $\mathbf{X}$ に対して成り立つこと、および対角行列のスペクトルノルムがその対角成分の最大絶対値に等しいことを示せ。
    3. Equation 33.39 の逆は成り立たないことを示せ。すなわち、正規ではない 3×3 行列であって、$\|\mathbf{A}^k\|_2 = \rho(\mathbf{A})^k$ をすべての $k \ge 1$ に対して満たすものを 1 つ見つけよ。
    4. 与えられた $t > 0$ に対し、対角化可能な 2×2 行列でスペクトル半径が $1/2$ かつ $\|\mathbf{A}\|_2 > t$ を満たすものを構成せよ。

2. [code] **べき乗法の収束レート。** $\mathbf{S}$ を Figure 33.21 の行列とする。
   1. Equation 33.38 より、反復ベクトルと $\mathbf{v}_1$ の向きのずれはレート $|\lambda_2/\lambda_1| = 1/3$ で減衰する。このずれを 100 分の 1 以下にするには何回の反復が必要か。
   2. $\mathbf{A}$ を対称行列とし、単位固有ベクトル $\mathbf{v}_1$（固有値 λ_1 に対応）をとる。また $\mathbf{x} = \mathbf{v}_1 + \delta\mathbf{u}$ とおく（ただし $\mathbf{u}$ は $\mathbf{v}_1$ に直交する単位ベクトル）。33.3.2.3 節のレイリー商が $R(\mathbf{x}) - \lambda_1 = O(\delta^2)$ を満たし、δ に関する一次の項を持たないことを示せ。したがって、べき乗法の反復ベクトルに対するレイリー商はどのレートで λ_1 に近づくか。また、この議論のどこで $\mathbf{A}$ が対称であることを用いているか。
   3. Figure 33.21 の実験を、ランダムな初期ベクトルから $\left(\begin{smallmatrix} 1 & 0.1 \\ 0.1 & 1 \end{smallmatrix}\right)$（$\mathbf{S}$ と同じ固有ベクトルを持ち、固有値が 1.1 と 0.9 である行列）に対して繰り返せ。$\mathbf{v}_1$ とのなす角の正弦とレイリー商の誤差を、30 ステップにわたって対数スケールでプロットせよ。両方の傾き、およびそれぞれの誤差が 100 分の 1 に縮小するまでに要するステップ数を予測し、なぜこの行列が $\mathbf{S}$ よりもはるかに遅く収束するのか説明せよ。

3. [code] **擬スペクトルと過渡的増大。** ●$\mathbf{M}$ を非正規行列 Equation 33.41 とし、$\Lambda = \mathrm{diag}(0.9, 0.8)$ を 33.4.2.3 節と同じく同一の固有値を持つ正規行列とする。
   1. Equation 33.43 の 2 つの集合が等しいことを証明せよ。まず $\|(z\mathbf{I} - \mathbf{A})^{-1}\|_2 = 1/\sigma_{\min}(z\mathbf{I} - \mathbf{A})$ を示せ（ただし $\sigma_{\min}(\mathbf{X}) = \min_{\|\mathbf{x}\|=1} \|\mathbf{X}\mathbf{x}\|$）。一方の包含関係については、単位ベクトル $\mathbf{u}$、$\mathbf{v}$ として $(z\mathbf{I} - \mathbf{A})\mathbf{v} = \sigma\mathbf{u}$（$\sigma = \sigma_{\min}(z\mathbf{I} - \mathbf{A})$）を満たすものをとり、$z$ が $\mathbf{A} + \sigma\mathbf{u}\mathbf{v}^*$ の固有値であることを示せ。逆の包含関係については、$(z\mathbf{I} - \mathbf{A})\mathbf{x} = \mathbf{E}\mathbf{x}$ を固有ベクトル $\mathbf{x}$（$\mathbf{A} + \mathbf{E}$ の固有ベクトル）に適用せよ。33.4.2.4 節で出力したレゾルベントノルムを用いると、$z = 1$ を摂動後の $\mathbf{M}$ の固有値にするにはどの程度の大きさの摂動が必要か。また Λ の場合はどの程度か。
   2. $\|\mathbf{A}^k\|_2 \le K_0$ がすべての $k \ge 0$ で成り立つとする。$|z| > 1$ に対して級数 $\sum_{k\ge 0} \mathbf{A}^k / z^{k+1}$ が $(z\mathbf{I} - \mathbf{A})^{-1}$ に収束することを示し、次を結論せよ。

$$\sup_{k\ge 0} \|\mathbf{A}^k\|_2 \;\ge\; K(\mathbf{A}) = \sup_{|z|>1} \left(|z| - 1\right) \|(z\mathbf{I} - \mathbf{A})^{-1}\|_2 .$$

      量 $K(\mathbf{A})$ は Kreiss 定数と呼ばれる。この下界を擬スペクトルの言葉で言い換えよ。すなわち、$\Lambda_\varepsilon(\mathbf{A})$ が単位円の外側に距離 δ 離れた点を含むとき、$\mathbf{A}$ のあるべき乗は少なくともどのくらいの大きさにならなければならないか。
   3. $K(\mathbf{M})$ を複素点 $z (|z| > 1)$ のグリッド上で数値的に評価し、最大の $\|\mathbf{M}^k\|_2$（33.4.2.3 節で出力した値）と比較せよ。また $K(\Lambda)$ を手計算で求めよ。

4. [code] **PageRank における減衰。** ●$\mathbf{P}$ を列確率的な $n \times n$ 行列とし、$\mathbf{G} = \alpha\mathbf{P} + (1-\alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^\top$（$0 < \alpha < 1$）を 33.4.1.3 節の減衰付き行列とする。
   1. $\mathbf{P}\mathbf{x} = \lambda\mathbf{x}$ とする。$\mathbf{1}^\top\mathbf{x} = 0$ が $\lambda \ne 1$ のときに必ず成り立つこと、および $\mathbf{G}\mathbf{x} = \alpha\lambda\mathbf{x}$ が $\mathbf{1}^\top\mathbf{x} = 0$ のときに必ず成り立つことを示せ。
   2. 固有値 1（行列 $\mathbf{P}$ の固有値）は、リンクが互いに非連結なページ群に分かれている場合のように重複しうる。ここでは、任意の列確率行列 $\mathbf{P}$ が固有値 1 に対する非負の固有ベクトル（定常分布）を持つという事実を用いてよい。この固有空間に属するベクトル $\mathbf{x}$ のうち $\mathbf{1}^\top\mathbf{x} = 0$ を満たすものが、元の固有空間より次元が 1 小さい部分空間をなすことを示せ。これより、対角化可能な $\mathbf{P}$ に対して $\mathbf{G}$ の固有値は 1 と $\alpha\lambda_2(\mathbf{P}), \dots, \alpha\lambda_n(\mathbf{P})$ になり、したがって上界 $|\lambda_2(\mathbf{G})| \le \alpha$（Haveliwala and Kamvar (2003)）が得られることを結論せよ。

3. 4 つのページが巡回路 $1 \to 2 \to 3 \to 4 \to 1$ をなしてリンクしているとする。このリンク行列 $\mathbf{P}$ の固有値を求めよ。ページ 1 から出発する減衰なしのべき乗法が決して収束しないのはなぜか。
4. $\alpha = 0.85$ と (c) の巡回路に対し、$\mathbf{G}$ 上のべき乗法をページ 1 から実行し、定常分布までの ℓ_1 距離が 10^{-8} を下回るまでの反復回数を数えよ。(b) による予測 $0.85^{k} \le 10^{-8}$ と比較せよ。
5. 33.4.1.3 節の 3 ページの例について、$\mathbf{G}$ の固有値が固有値 1 を除いて α 倍された $\mathbf{P}$ の固有値になっていること、および定常分布が $(0.3, 0.4, 0.3)$ から約 $(0.303, 0.394, 0.303)$ へ移動することを数値的に確認せよ。減衰によって定常分布が一様分布へ近づくのはなぜか。

5. [code] **ランダム行列積のリアプノフ指数。●●** $\mathbf{D}_1, \mathbf{D}_2, \dots$ を独立な $n \times n$ 行列とし、各成分は独立に $\mathcal{N}(0, 1/n)$ に従うものとする（ファンイン初期化における深層ネットワークのヤコビ行列の線形モデル）。また $\mathbf{x}_0$ を単位ベクトルとする。33.4.2.6 節より、各因子のスペクトル半径は 1 に近い（n が大きいとき）。33.4.2.5 節では、この積の増大率を次の最大リアプノフ指数で要約した。

$$\mu = \lim_{T \to \infty} \frac{1}{T} \log \left\| \mathbf{D}_T \cdots \mathbf{D}_1 \mathbf{x}_0 \right\|,$$

この極限は Furstenberg と Kesten の定理により確率 1 で存在する。

1. $\mathbb{E}\|\mathbf{D}\mathbf{x}\|^2 = \|\mathbf{x}\|^2$ が任意の固定された $\mathbf{x}$ に対して成り立ち、したがってこのスケーリングが二乗ノルムを平均的に保つことを示せ。
2. $\mathbf{u}_0 = \mathbf{x}_0$ および $\mathbf{u}_t = \mathbf{x}_t / \|\mathbf{x}_t\|$ とおく（ただし $\mathbf{x}_t = \mathbf{D}_t \cdots \mathbf{D}_1 \mathbf{x}_0$）。$\mathbf{D}_1, \dots, \mathbf{D}_{t-1}$ が与えられた条件のもとで、$n\|\mathbf{D}_t \mathbf{u}_{t-1}\|^2$ はどの分布に従うか。その結果を用いて、$\log \|\mathbf{D}_T \cdots \mathbf{D}_1 \mathbf{x}_0\|$ を独立な項の和として表せ。さらに $\mathbb{E} \log \chi^2_n = \psi(n/2) + \log 2$（$\psi$ はディガンマ関数）を用い、Cohen and Newman (1984) による次の閉じた式を導出せよ。

$$\mu = \tfrac{1}{2}\big(\psi(n/2) + \log 2 - \log n\big).$$

3. (a) が成り立つにもかかわらず、$\mu < 0$ がすべての n に対して成り立つことを示せ。μ を $n = 1$ の場合に閉じた形で求めよ（$\psi(1/2) = -\gamma_{\mathrm{E}} - 2\log 2$ とオイラー定数 $\gamma_{\mathrm{E}} \approx 0.5772$ を用いよ）。さらに $\psi(x) = \log x - \frac{1}{2x} + O(x^{-2})$ を用いて、$\mu \approx -1/(2n)$ が大きな n に対して成り立つことを示せ。
4. 再正規化を伴う $T = 10^4$ ステップのシミュレーションを $n \in \{1, 2, 5, 50\}$ に対して実行し、計測された指数を (b) の理論値、ならびに単一因子のスペクトル半径の対数および最大特異値の対数と比較せよ。この初期化を用いた幅 50、深さ 100 の線形ネットワークでは、典型的な入力は何倍に縮小するか。また、この比較は 33.4.2.5 節の 3 つの量について何を示唆するか。

33.5 特異値分解と低ランク近似

33.3 節では、固有値分解（eigendecomposition）が正方行列を「単に伸縮するだけの方向」によって記述し、行列のべき乗を単なる数値のべき乗へと帰着させることを見た。しかし、深層学習に現れる大半の行列はそのような性質を持たない。重み行列はある幅の入力を別の幅の出力へ写し、データ行列はデータ例ごとに 1 行、特徴量ごとに 1 列を持つ。どちらも自身の出力へ繰り返

し適用されるわけではなく、正方行列である必要もない。さらに行列が**正方行列であっても**、固有値だけでは不十分なことがある。せん断行列 $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ は**欠陥**（defective）を持ち、平面を張るには 2 方向が必要なところ固有ベクトルの方向を 1 つしか持たない。また 33.4 節では、2 つの固有値がともに 1 より小さいにもかかわらず、6 乗するとベクトルを 10 倍に引き伸ばす行列に出会った。ここで必要なのは、**あらゆる**行列に対して存在し、伸縮を直接測ることができ、大きな行列を精度よく近似するのにどれほど少ない数値で足りるかを教えてくれる分解である。

最後の要件のありがたみは、具体的な写真を考えると実感しやすい。400×500 ピクセルの猫のグレースケール画像は、単なる 200,000 個の数値からなる行列である。他のあらゆる行列と同様、この画像も 1 本の列と 1 本の行の外積（outer product）で作られるランク 1 の成分の和として書ける。驚くべきことに、うまく選んだわずか 20 個の成分（18,020 個の数値、すなわち元データの 9%）だけで、相対誤差わずか 8% で写真を再現できる（Figure 33.28）。どの 20 個の成分がこれを達成するのか、また別の 20 個を選べばさらに良くできるのだろうか。深層学習の至るところでこれと同じ問いが現れる。大規模モデルのファインチューニング用アダプタにどれだけのパラメータが必要か、そしてデータセットのどの方向が本当に残す価値があるのかを、この問いが教えてくれるのである。

鍵となる考え方は、入力と出力の両方を単一の基底で済ませようとするのをやめることである。入力空間に 1 つの正規直交基底（orthonormal basis）を、出力空間に別の正規直交基底を用意すれば、**あらゆる**行列は独立な非負の伸縮率の列 $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$ へと帰着する。このたった 1 つのリストから、行列のランク、最良の低ランク近似（low-rank approximation）、主成分分析（principal component analysis、PCA）、最小二乗法（least squares）の解、さらには連立一次方程式（linear system）が測定誤差に対してどれほど敏感かまで読み取れる。1.3 節では、行列の大きさを測る 2 つの方法として、フロベニウスノルム $\|\mathbf{A}\|_F$（全成分の二乗和の平方根）とスペクトルノルム $\|\mathbf{A}\|_2$（$\mathbf{A}$ が単位ベクトルを引き伸ばせる最大の倍率）を導入した。本節では、そのどちらも特異値（singular value）σ_i の単純な関数として表される。

以下の議論の柱となるのは 2 つの主要な結果である。1 つは任意の行列に対する特異値分解（singular value decomposition、SVD）の存在であり、もう 1 つは分解を途中で打ち切ることであらゆるランクにおける最良近似が得られると保証する Eckart–Young の定理である。そこから、擬似逆行列（pseudoinverse）と条件数（condition number）がわずか数行で導かれる。最後の小節ではこれらの道具を現代の深層学習と結びつけるが、初読時には飛ばしてもよい。以下の数値実験はすべて NumPy を用い、特に断らない限り、本節を通じて $\mathbf{T}$ はせん断行列、$\mathbf{A}$ は一般の行列を表す。

33.5.1 特異値分解

33.5.1.1 分解とその幾何学

1.3 節では、対称行列 $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ が単位円を楕円へ写し、その半軸が固有ベクトル $(1,1)$ と $(-1,1)$ の方向に沿って長さ 3 と 1 を持つことを見た。驚くべきことに、まもなく証明するように、**非対称**行列もまた単位円を楕円へと写す（行列が特異行列であれば線分に潰す）。違いは、この楕円の軸の上に着地する直交した入力方向が、もはやその軸自身の方向とは一致しない点にある。代わりに、入力空間用の正規直交フレームと出力空間用の正規直交フレームという 2 つの正規直交フレームが存在し、特異値分解はその両方に名前を与える。

定理（特異値分解）。 任意の実 $m \times n$ 行列 $\mathbf{A}$ は次のように書ける。

$$\mathbf{A} = \mathbf{U}\,\Sigma\,\mathbf{V}^{\top}, \tag{33.44}$$

ここで、$\mathbf{U} \in \mathbb{R}^{m \times m}$ と $\mathbf{V} \in \mathbb{R}^{n \times n}$ は直交行列（$\mathbf{U}^{\top}\mathbf{U} = \mathbf{I}$、$\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$）であり、$\Sigma \in \mathbb{R}^{m \times n}$ は主対角線上の非負成分 $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$ を除いてすべて 0 である。列ベクトル $\mathbf{v}_i$

（$\mathbf{V}$ の各列）を**右特異ベクトル**（right singular vector）、列ベクトル $\mathbf{u}_i$（$\mathbf{U}$ の各列）を**左特異ベクトル**（left singular vector）、対角成分 σ_i を**特異値**と呼ぶ。

幾何学的に言えば、形状にかかわらず**あらゆる**行列は、直交変換、対角方向の伸縮、そして 2 つ目の直交変換を順に施したものにほかならない。$\mathbf{AV} = \mathbf{U}\Sigma$ と書き直して第 i 列を読み取れば、1 方向ずつの作用が明示的にわかる。

$$\mathbf{A}\mathbf{v}_i = \sigma_i \, \mathbf{u}_i, \qquad i \le \min(m, n). \tag{33.45}$$

各入力方向 $\mathbf{v}_i$ は対応する出力方向 $\mathbf{u}_i$ へと写され、σ_i 倍に伸縮される。$n > m$ のときは、余分な入力方向 $\mathbf{v}_{m+1}, \dots, \mathbf{v}_n$ はゼロに潰される。Equation 33.44 を右から左へ読むと、$\mathbf{A}$ をベクトル $\mathbf{x}$ に作用させる操作は次の 3 段階で進む。

1. $\mathbf{V}^{\top}$ は入力を回転させ、各右特異ベクトル $\mathbf{v}_i$ を座標軸 $\mathbf{e}_i$ に揃える。
2. Σ は軸 i を σ_i 倍に伸縮し、$m \neq n$ のときは出力次元に合わせてゼロ座標を追加するか、余分な座標を切り捨てる。
3. $\mathbf{U}$ は伸縮された各軸を回転させ、軸 i が $\mathbf{u}_i$ の方向を向くようにする。

ここで「回転」とは直交変換の便宜的な略称であり、鏡映を含む場合もある。

Figure 33.24 に、せん断 $\mathbf{T}$ におけるこれら 3 段階を示す。特異値は $\sigma_1 = 1.618$ と $\sigma_2 = 0.618$ であり、第 1 右特異ベクトル $\mathbf{v}_1$ は 58.3° の角度で x 軸の上方を向く一方、その像の方向 $\mathbf{u}_1$ は 31.7° を向く。入力フレームと出力フレームが異なるため、$\mathbf{U} \neq \mathbf{V}$ となる。これらの正確な数値は 33.5.1.2 節で導出する。

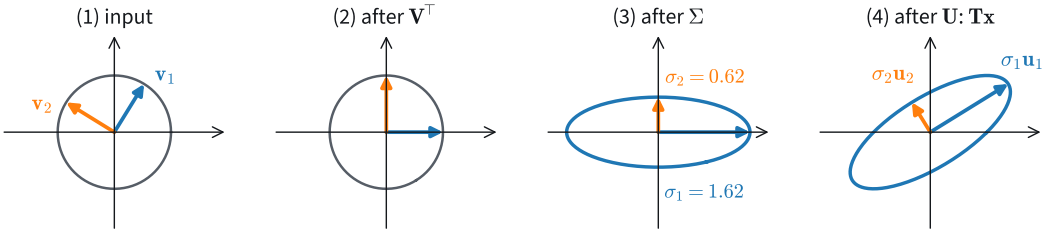


Figure 33.24: 単位円に作用するせん断 $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right) = \mathbf{U}\Sigma\mathbf{V}^{\top}$（左から右へ読む）。$\mathbf{V}^{\top}$ が -58.3° 回転させ、Σ が $\varphi = 1.618$ 倍に x 方向へ、$1/\varphi = 0.618$ 倍に y 方向へ伸縮し、$\mathbf{U}$ が 31.7° 回転させる。入力フレーム $\{\mathbf{v}_i\}$ と出力フレーム $\{\mathbf{u}_i\}$ は異なる。

n 次元でも同じ 3 段階によって、単位球面はベクトル $\sigma_i\mathbf{u}_i$ を半軸とする楕円体へと変形される。入力方向のうち正の特異値を持つものが $r < n$ 個だけであれば、残りの $n - r$ 方向はゼロに潰れ、像は平坦な r 次元楕円体となって $\mathbf{u}_1, \dots, \mathbf{u}_r$ が張る空間の内部に収まる。

コンパクト形式と薄い形式。正の特異値の個数を r とする（33.5.1.5 節で r がまさに $\mathbf{A}$ のランクに一致することを確かめる）。Σ の最初の r 個を超える行と列はすべて 0 であるため、それらと対になる特異ベクトルは積に一切寄与しない。これらのゼロ成分を取り除くと、**コンパクト SVD**（compact SVD）$\mathbf{A} = \mathbf{U}_r\Sigma_r\mathbf{V}_r^{\top}$ が得られる。ここで $\mathbf{U}_r \in \mathbb{R}^{m \times r}$ と $\mathbf{V}_r \in \mathbb{R}^{n \times r}$ は最初の r 本の特異ベクトルを並べた行列であり、$\Sigma_r = \mathrm{diag}(\sigma_1, \dots, \sigma_r)$ である。完全形式とコンパクト形式の中間に位置するのが**薄い SVD**（thin SVD）である。これは数値計算ライブラリで縮小形式を指定した際に返される既定の形式であり、$\min(m, n)$ 本の列を $\mathbf{U}$ と $\mathbf{V}$ に残しつつ、$r < \min(m, n)$ のときにはゼロ特異値もそのまま記録する。ランク 2 の 3×2 行列では、完全形式の因子の形状は 3×3、3×2、2×2 であるのに対し、薄い形式の因子の形状は 3×2、2×2、2×2 となる（この行列は列フルランクであるため、ここではコンパクト SVD と一致する）。完全形式と薄い形

式は行列が正方でないときに異なり、薄い形式とコンパクト形式は行列がランク落ちしているときに異なる。Figure 33.25 に、3 つの形式がすべて異なるランク 3 の 6×4 行列を示す。

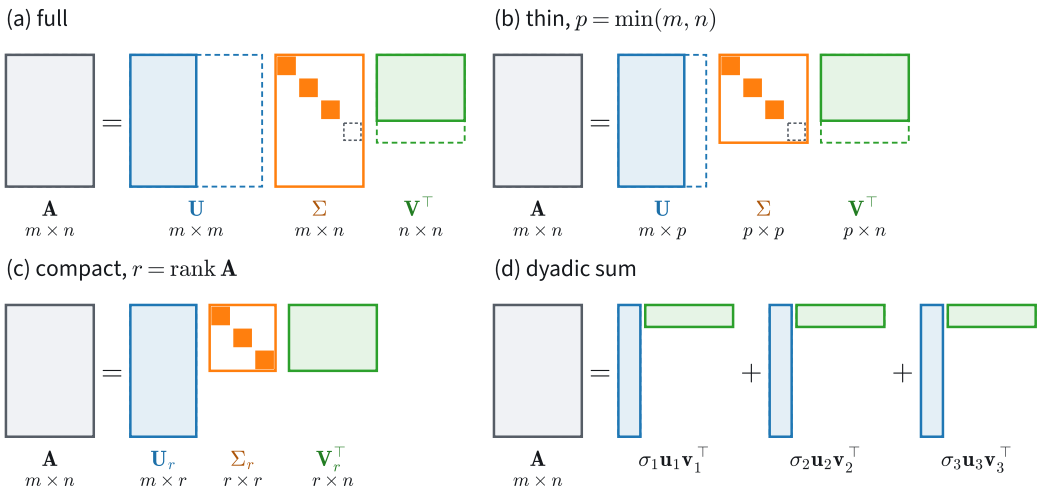


Figure 33.25: $m \times n$ 行列（ランク r）における SVD 因子の形状（$m = 6$、$n = 4$、$r = 3$ の縮尺で描画）。(a) 完全形式、(b) 薄い形式、(c) コンパクト形式、(d) r 個のランク 1 項からなるダイアド和。Σ の対角線上の塗りつぶされた正方形は正の特異値であり、破線の正方形はゼロ特異値である。$\mathbf{U}$ と $\mathbf{V}^\top$ の破線部分は Σ のゼロとしか掛からないため、より小さい各形式では積を変えずにそれらの一部を省ける。

ダイアド和。コンパクト SVD を列と行の積として展開すると、$\mathbf{A}$ はしばしば**ダイアド**と呼ばれるランク 1 行列の重み付き和（ダイアド和、dyadic sum）として表される。

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \, \mathbf{u}_i \mathbf{v}_i^\top . \tag{33.46}$$

各項 $\mathbf{u}_i \mathbf{v}_i^\top$ は出力方向と入力方向の外積に特異値の重みを掛けたものであり、重い項から軽い項へとあらかじめ並んでいる。せん断行列の場合は $\mathbf{T} = \varphi \, \mathbf{u}_1 \mathbf{v}_1^\top + \varphi^{-1} \, \mathbf{u}_2 \mathbf{v}_2^\top$ となり、重み 1.618 の成分と重み 0.618 の 2 つ目の成分に分解される。このうち重い成分だけを残すというのが、33.5.2.1 節で扱う低ランク近似の基本的な考え方である。

33.5.1.2 存在性

σ_i、$\mathbf{v}_i$、$\mathbf{u}_i$ はどこから現れ、なぜ出力方向まで自動的に正規直交になるのだろうか。33.3.2.1 節のスペクトル定理により、あらゆる対称行列は実固有値を持つ固有ベクトルの正規直交基底を備えている。これを**グラム行列**（Gram matrix）$\mathbf{A}^\top \mathbf{A}$、すなわちサイズ $n \times n$ の $\mathbf{A}$ の列同士のドット積からなる行列に適用できる。この行列は対称行列であり、さらに半正定値（positive semidefinite、33.3.2.2 節）でもある。なぜなら、$\mathbf{x}^\top \mathbf{A}^\top \mathbf{A} \mathbf{x} = \|\mathbf{A}\mathbf{x}\|^2 \geq 0$ がすべてのベクトル $\mathbf{x}$ に対して成り立つからである。

Equation 33.44 の証明（存在性）。スペクトル定理より、$\mathbf{A}^\top \mathbf{A}$ は固有ベクトルの正規直交基底 $\mathbf{v}_1, \ldots, \mathbf{v}_n$ と、対応する固有値 $\lambda_1 \geq \cdots \geq \lambda_n$ を持つ。$\lambda_i = \mathbf{v}_i^\top \mathbf{A}^\top \mathbf{A} \mathbf{v}_i = \|\mathbf{A}\mathbf{v}_i\|^2$ であるから、すべての固有値は非負である。そこで次のように定義する。

$$\sigma_i = \sqrt{\lambda_i}, \qquad r = \#\{i : \sigma_i > 0\}.$$

すなわち、特異値はグラム行列の固有値の平方根であり、r は正の特異値の個数を表す。各 $i \le r$ に対して $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i/\sigma_i$ とおく。ここで最も重要な確認事項は、こうして写されたベクトルが自動的に正規直交になることであり、これは $\mathbf{v}_j$ が $\mathbf{A}^{\top}\mathbf{A}$ の固有ベクトルであることと、$\mathbf{v}_i$ が正規直交であることから直ちに従う。

$$\mathbf{u}_i^{\top}\mathbf{u}_j = \frac{1}{\sigma_i\sigma_j}\,\mathbf{v}_i^{\top}\mathbf{A}^{\top}\mathbf{A}\,\mathbf{v}_j = \frac{\lambda_j}{\sigma_i\sigma_j}\,\mathbf{v}_i^{\top}\mathbf{v}_j = \frac{\lambda_j}{\sigma_i\sigma_j}\,\delta_{ij} = \delta_{ij}.$$

したがって、入力フレームの正規直交性は $\mathbf{A}^{\top}\mathbf{A}$ を介してそのまま出力フレームへと受け継がれる。次に次元を確認しよう。r 本の正規直交ベクトル $\mathbf{u}_1, \dots, \mathbf{u}_r$ は $\mathbb{R}^m$ に属するため $r \le m$ であり、固有値は全部で n 個あるため $r \le n$ である。ゆえに $r \le \min(m,n)$ となり、$\lambda_i = 0$ がすべての $i > r$ で成り立つ。$\mathbf{u}_1, \dots, \mathbf{u}_r$ を延長して正規直交基底 $\mathbf{u}_1, \dots, \mathbf{u}_m$（$\mathbb{R}^m$ の基底）を作る（たとえば任意の基底延長に Gram–Schmidt 法を適用すればよい）。$r < i \le n$ に対しては $\|\mathbf{A}\mathbf{v}_i\|^2 = \lambda_i = 0$ であるから、$\mathbf{A}\mathbf{v}_i = \mathbf{0}$ となる。ここで $\mathbf{AV}$ と $\mathbf{U}\Sigma$ を 1 列ずつ比較すると、第 i 列（$\mathbf{U}\Sigma$ の列）は、$\sigma_i\mathbf{u}_i$（$i \le \min(m,n)$ のとき。$i > r$ ではゼロになる）または $\mathbf{0}$（$i > m$ のとき。この範囲には Σ の対角成分がない）である。いずれの場合も、第 i 列（$\mathbf{AV}$）は第 i 列（$\mathbf{U}\Sigma$）と一致する。よって $\mathbf{AV} = \mathbf{U}\Sigma$ が成り立ち、右から $\mathbf{V}^{\top} = \mathbf{V}^{-1}$ を掛ければ $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\top}$ が得られる。$\blacksquare$

この証明はそのまま計算手順にもなっている。対称行列 $\mathbf{A}^{\top}\mathbf{A}$ を対角化し、固有値の平方根をとって特異値を求め、固有ベクトルを $\mathbf{A}$ で写して左特異ベクトルを得ればよい。

各ステップの働きを具体的に確かめるため、せん断行列を手計算で追ってみよう。$\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ に対し、グラム行列は $\mathbf{T}^{\top}\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ である。トレースは 3、行列式は 1 であるから、特性多項式は $\lambda^2 - 3\lambda + 1$ となり、固有値 $(3 \pm \sqrt{5})/2$ が得られる。$\varphi^2 = \varphi + 1$ を満たす黄金比 $\varphi = (1+\sqrt{5})/2$ を導入すると、これらの固有値は φ^2 と φ^{-2} へと綺麗に簡約される。したがって特異値は $\sigma_1 = \varphi \approx 1.618$ および $\sigma_2 = 1/\varphi \approx 0.618$ となり、その積は $1 = |\det \mathbf{T}|$ である。

続いて特異ベクトルを求めよう。第 1 右特異ベクトル (x, y) について、$(\mathbf{T}^{\top}\mathbf{T} - \varphi^2\mathbf{I})(x,y) = \mathbf{0}$ の第 1 行は $(1-\varphi^2)x + y = 0$ となり、$\varphi^2 - 1 = \varphi$ を用いると $y = \varphi x$、すなわち $\mathbf{v}_1 \propto (1, \varphi)$ が得られる。これに $\mathbf{T}$ を作用させると $\mathbf{T}(1,\varphi) = (1+\varphi, \varphi) = \varphi\,(\varphi, 1)$ となるため、$\mathbf{u}_1 \propto (\varphi, 1)$ である。両ベクトルとも長さは $\sqrt{1+\varphi^2} = \sqrt{\varphi+2}$ であり、正規化すると $\mathbf{v}_1 \approx (0.526, 0.851)$（角度 $\arctan\varphi = 58.3^{\circ}$）および $\mathbf{u}_1 \approx (0.851, 0.526)$（角度 31.7°）となる。第 2 の組は直交性から $\mathbf{v}_2 = (-\varphi, 1)/\sqrt{\varphi+2}$、$\mathbf{u}_2 = (-1, \varphi)/\sqrt{\varphi+2}$ と求まり、実際 $\mathbf{T}\mathbf{v}_2 = (1-\varphi, 1)/\sqrt{\varphi+2} = \varphi^{-1}\mathbf{u}_2$ が $(1-\varphi)\varphi = -1$ より成り立つ。$\mathbf{V}$ と $\mathbf{U}$ はどちらも回転行列 $\mathbf{R}_{\theta} = \left(\begin{smallmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{smallmatrix}\right)$ であるため、せん断行列は次のように分解される。

$$\mathbf{T} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \mathbf{R}_{31.7^{\circ}} \begin{bmatrix} \varphi & 0 \\ 0 & 1/\varphi \end{bmatrix} \mathbf{R}_{58.3^{\circ}}^{\top}. \tag{33.47}$$

幾何学的には、このせん断は平面を -58.3° 回転させ、1.618 倍に x 軸方向へ、0.618 倍に y 軸方向へ伸縮し、最後に 31.7° だけ回転させて戻す。2 つの角度の和が 90° になるのは、$\mathbf{u}_1$ が単に $\mathbf{v}_1$ の座標を入れ替えたものだからであり、これは対称性 $\mathbf{T}^{\top} = \mathbf{JTJ}$（入れ替え行列 $\mathbf{J} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ に関する関係）を反映している。

NumPy の `np.linalg.svd` はこの手計算と一致するだろうか。特異ベクトルは符号を除いてしか定まらないため（33.5.1.3 節）、以下のセルではせん断行列を `A` として格納し、まず $\mathbf{u}_1$ が第 1 象限に入り $\mathbf{U}$ が純粋な回転になるよう符号を揃えている。

```
A = np.array([[1., 1.], [0., 1.]])
U, s, Vt = np.linalg.svd(A)
if U[0, 0] < 0:                          # joint sign choice: u_1 in the first quadrant
    U[:, 0], Vt[0] = -U[:, 0], -Vt[0]
if np.linalg.det(U) < 0:                 # joint sign choice: U a rotation (then so is V)
    U[:, 1], Vt[1] = -U[:, 1], -Vt[1]
phi = (1 + np.sqrt(5)) / 2
print('singular values', s.round(4), '  phi, 1/phi', np.array([phi, 1 / phi]).round(4))
print('U =', U.round(3).tolist(), '  Vt =', Vt.round(3).tolist())
print('angles of u_1, v_1 (degrees):', np.degrees(np.arctan2(U[1, 0], U[0, 0])).round(1),
      np.degrees(np.arctan2(Vt[0, 1], Vt[0, 0])).round(1))
print('A v_1 = sigma_1 u_1:', np.allclose(A @ Vt[0], s[0] * U[:, 0]),
      '  sigma_1 sigma_2 =', s.prod().round(6))
```

```
singular values [1.618 0.618]   phi, 1/phi [1.618 0.618]
U = [[0.851, -0.526], [0.526, 0.851]]   Vt = [[0.526, 0.851], [-0.851, 0.526]]
angles of u_1, v_1 (degrees): 31.7 58.3
A v_1 = sigma_1 u_1: True   sigma_1 sigma_2 = 1.0
```

結果は一致する。ライブラリは φ と $1/\varphi$ に加え、Equation 33.47 の 2 つの回転を返す。

同じ構成法は長方形行列に対してもそのまま機能する。3×2 行列として、各行が $(3,1)$、$(1,3)$、$(0,2)$ であるものを考えよう。そのグラム行列は $\left(\begin{smallmatrix} 10 & 6 \\ 6 & 14 \end{smallmatrix}\right)$ であり、固有値は $12 \pm \sqrt{40}$ である。次のセルでは、証明の 3 ステップ（$\mathbf{A}^{\top}\mathbf{A}$ の固有ベクトル、平方根、および $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i/\sigma_i$）をそのまま実行し、結果を `np.linalg.svd` と比較する。

```
A = np.array([[3., 1.], [1., 3.], [0., 2.]])
lam, V = np.linalg.eigh(A.T @ A)
lam, V = lam[::-1], V[:, ::-1]                    # descending order
sigma = np.sqrt(lam)
U = A @ V / sigma                                  # u_i = A v_i / sigma_i
U_lib, s_lib, Vt_lib = np.linalg.svd(A, full_matrices=False)
print('sigma from eigh:', sigma.round(5), '  from svd:', s_lib.round(5))
print('U orthonormal:', np.allclose(U.T @ U, np.eye(2)), '  same U, V up to sign:',
      np.allclose(abs(U), abs(U_lib)), np.allclose(abs(V), abs(Vt_lib.T)))
print('rebuild error:', np.linalg.norm((U * sigma) @ V.T - A).round(12))
```

```
sigma from eigh: [4.28072 2.38232]   from svd: [4.28072 2.38232]
U orthonormal: True   same U, V up to sign: True True
rebuild error: 0.0
```

どちらの方法でも $\sigma_1 = \sqrt{12+\sqrt{40}} \approx 4.281$ と $\sigma_2 = \sqrt{12-\sqrt{40}} \approx 2.382$ が得られる。証明で保証されたとおり、写されたベクトル $\mathbf{u}_i$ は正規直交となり、$\mathbf{U}$ と $\mathbf{V}$ はともに符号を除いてライブラリの出力と一致する。

33.5.1.3 一意性と最大伸縮率

行列の SVD は一意だろうか。特異**値**は常に一意である。$\mathbf{A}^{\top}\mathbf{A}$ の固有値の平方根であり、$\mathbf{A}$ によって完全に定まるからである。これに対し、特異**ベクトル**には次の 3 種類の自由度がある。

- 単根（重複しない）の $\sigma_i > 0$ に対しては、自由度は同時符号反転 $(\mathbf{u}_i, \mathbf{v}_i) \to (-\mathbf{u}_i, -\mathbf{v}_i)$ のみである。片方だけの符号を反転させると $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$ が崩れるため、両方のベクトルを同時に反転させなければならない。

- 重複特異値 $\sigma > 0$（添字集合を $\mathcal{J}$ とする）に対しては、列集合 $\mathbf{V}_{\mathcal{J}}$ に同じサイズの任意の直交行列 $\mathbf{Q}$ を掛けて $\mathbf{V}_{\mathcal{J}}\mathbf{Q}$ と回転させてよい。ただし、$\mathbf{U}_{\mathcal{J}}$ にも**同じ** $\mathbf{Q}$ を掛けて $\mathbf{U}_{\mathcal{J}} = \mathbf{A}\mathbf{V}_{\mathcal{J}}/\sigma$ を保つ必要がある。
- $\sigma = 0$ に対しては、零空間のベクトル $\mathbf{v}_{r+1}, \dots, \mathbf{v}_n$ と左零空間のベクトル $\mathbf{u}_{r+1}, \dots, \mathbf{u}_m$ はダイアド和 Equation 33.46 に現れないため、互いに完全に独立に回転させることができる。

2 つの数値計算ライブラリ（あるいは同じライブラリでも異なる 2 台の計算機）が特異ベクトルに対して逆の符号を返すことがあるのも、また本節のコードセルでベクトルを表示する前に符号の規約を固定しているのも、この自由度のためである。

$\mathbf{A}$ を分解するだけでなく、最大特異値には直接的な幾何学的意味がある。それは $\mathbf{A}$ が任意の単位ベクトルを引き伸ばせる最大の倍率である。

$$\sigma_1 = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| = \|\mathbf{A}\|_2. \tag{33.48}$$

これは 33.3.2.3 節の Rayleigh の命題から直ちに従う。すなわち、単位ベクトル $\mathbf{x}$ 全体にわたって二次形式（quadratic form）$\mathbf{x}^{\top}\mathbf{S}\mathbf{x}$（対称行列 $\mathbf{S}$ に対するもの）を最大化すると、最大固有ベクトルにおいて最大固有値に等しい最大値をとる。$\mathbf{S} = \mathbf{A}^{\top}\mathbf{A}$ とおけば、$\|\mathbf{A}\mathbf{x}\|^2 = \mathbf{x}^{\top}\mathbf{A}^{\top}\mathbf{A}\mathbf{x}$ は $\mathbf{x} = \mathbf{v}_1$ で最大値 $\lambda_1 = \sigma_1^2$ をとることがわかる。したがって、σ_1 はまさにスペクトルノルム $\|\mathbf{A}\|_2$（1.3 節）に一致する。せん断行列では、いかなる単位ベクトルも $\varphi = 1.618$ 倍を超えて引き伸ばされることはなく、$\mathbf{v}_1$ がその最大伸縮率を達成する。残りの特異値も、先行する右特異ベクトルと直交する空間に制限して同じ最大化問題を解くことで順に得られる。この手続きを**デフレーション**と呼び、$\sigma_k = \max\{\|\mathbf{A}\mathbf{x}\| : \|\mathbf{x}\| = 1,\ \mathbf{x} \perp \mathbf{v}_1, \dots, \mathbf{v}_{k-1}\}$ と書ける。実際、演習問題 1a ではこの論理を逆転させ、スペクトル定理を使わずに最大伸縮率だけから SVD 全体を構成する。

33.5.1.4 特異値と固有値

特異値と固有値はどのような関係にあるのだろうか。Equation 33.44 を 2 つのグラム行列に代入し、$\mathbf{U}^{\top}\mathbf{U} = \mathbf{I}$ と $\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$ を用いると次式が得られる。

$$\mathbf{A}^{\top}\mathbf{A} = \mathbf{V}\Sigma^{\top}\Sigma\mathbf{V}^{\top}, \qquad \mathbf{A}\mathbf{A}^{\top} = \mathbf{U}\Sigma\Sigma^{\top}\mathbf{U}^{\top}. \tag{33.49}$$

これら両式はいずれも対称行列の固有値分解になっている。すなわち、右特異ベクトルは $\mathbf{A}^{\top}\mathbf{A}$ の固有ベクトル、左特異ベクトルは $\mathbf{A}\mathbf{A}^{\top}$ の固有ベクトルであり、二乗 σ_i^2 は 2 つのグラム行列が共有する非ゼロ固有値である。$\mathbf{A}$ が対称半正定値行列のときは、SVD と固有値分解は $\mathbf{U} = \mathbf{V}$ として一致する。

しかし、それ以外のあらゆる行列では 2 つの分解は異なり、特異値が固有値の絶対値に等しいと思い込むのはよくある誤りである。本章で繰り返し用いている行列 $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ を考えよう。その固有値は $2 \pm i$ であり、絶対値はともに $\sqrt{5} \approx 2.236$ である（33.3 節）。一方、グラム行列 $\mathbf{A}^{\top}\mathbf{A} = \left(\begin{smallmatrix} 2 & -1 \\ -1 & 13 \end{smallmatrix}\right)$ はトレース 15、行列式 25 を持つため、$\sigma_{1,2}^2 = (15 \pm 5\sqrt{5})/2$ となり、したがって $\sigma_{1,2} = (5 \pm \sqrt{5})/2 \approx 3.618,\ 1.382$ となって 2.236 とは大きく異なる。それでもなお、あらゆる正方行列において 2 つのリストを結ぶ明快な恒等式が 1 つ成り立つ。

$$|\det \mathbf{A}| = \prod_{i} \sigma_i = \prod_{i} |\lambda_i|.$$

第 1 の等号は直交行列の行列式が ± 1 であることから従い、第 2 の等号は行列式が全固有値の積に等しいことから従う。2×2 行列が共通の絶対値 $|\lambda|$ を持つ共役複素固有値の対を備えるとき、

ここから $|\lambda| = \sqrt{\sigma_1 \sigma_2}$ が強制される。実際、この例でも $\sqrt{3.618 \cdot 1.382} = \sqrt{5}$ が成り立っている。またこの恒等式は、33.2 節で見た行列式の体積としての解釈も復元する。回転は面積を保つため、面積変化のすべてを対角伸縮 Σ が担い、上の例題行列は単位正方形を面積 $3.618 \cdot 1.382 = 5$ の平行四辺形へと写す。

実用上の SVD 計算。 本節の存在証明では $\mathbf{A}^{\top}\mathbf{A}$ から SVD を構成したが、数値計算ライブラリがその積を直接作ることは決してない。$\mathbf{A}^{\top}\mathbf{A}$ を作ると条件数が二乗され（33.5.3.2 節）、浮動小数点演算において小さな特異値が失われてしまうからである。その代わりにライブラリは、$\mathbf{A}$ に直接直交変換を施して二重対角形へと簡約し（Golub–Kahan 二重対角化）、その二重対角行列を対角化する (Golub and Van Loan 1996)。

33.5.1.5 4 つの基本部分空間

$\mathbf{A}$ はどの入力をゼロに潰し、どの出力に到達できるのだろうか。33.2 節では、これらの集合をそれぞれ零空間（null space）と列空間（column space）と名付け、線形独立な列の最大本数としてランクを定義した。$\mathbf{U}$ と $\mathbf{V}$ は可逆であるため、これらを掛けても次元は一切変わらない。したがって $\operatorname{rank}\mathbf{A} = \operatorname{rank}\Sigma$ となり、ランクは単なる非ゼロ対角成分の個数に帰着する。

$$\operatorname{rank}\mathbf{A} = \#\{i : \sigma_i > 0\} = r. \tag{33.50}$$

言い換えれば、ランクとは潰れずに生き残る方向の数を数えたものである。後述する丸め許容誤差と組み合わせれば、特異値を数える方法はランクを判定する上で数値的に最も信頼性が高い。ガウスの消去法より計算コストは高いものの、はるかに安定している。浮動小数点の丸め誤差（rounding error）のような摂動 $\mathbf{E}$ が加わっても、いかなる特異値も $\|\mathbf{E}\|_2$ を超えてずれることはないからである（33.5.2.1 節の Weyl の不等式）。したがって、大きな特異値と微小な特異値の間に明確なギャップがあれば、その結果を信頼できる。

さらに都合のよいことに、特異ベクトルを添字 r で分割すれば、$\mathbf{A}$ の 4 つの基本部分空間（fundamental subspace）すべての明示的な正規直交基底がダイアド和 Equation 33.46 から直接得られる。

- $\mathbf{A}$ の**列空間**（値域）は $\operatorname{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$ である。あらゆる出力 $\mathbf{A}\mathbf{x} = \sum_{i \le r} \sigma_i (\mathbf{v}_i^{\top}\mathbf{x})\mathbf{u}_i$ はこれらのベクトルの線形結合であり、かつ各基底ベクトルが $\mathbf{u}_i = \mathbf{A}(\mathbf{v}_i/\sigma_i)$ によって実際に達成されるからである。
- $\mathbf{A}$ の**零空間**は $\operatorname{span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_n\}$ である。同じ式より、$\mathbf{A}\mathbf{x} = \mathbf{0}$ が成り立つのは $\mathbf{v}_i^{\top}\mathbf{x} = 0$ がすべての $i \le r$ で成り立つとき、かつそのときに限る（$\mathbf{u}_i$ は線形独立であり、σ_i は正だからである）。これは $\mathbf{x}$ が $\mathbf{v}_1, \dots, \mathbf{v}_r$ に直交することを意味する。
- **行空間**（row space、$\mathbf{A}^{\top}$ の列空間）は $\operatorname{span}\{\mathbf{v}_1, \dots, \mathbf{v}_r\}$ であり、**左零空間**（left null space、$\mathbf{A}^{\top}$ の零空間）は $\operatorname{span}\{\mathbf{u}_{r+1}, \dots, \mathbf{u}_m\}$ である。これは同じ 2 つの議論を $\mathbf{A}^{\top} = \mathbf{V}\Sigma^{\top}\mathbf{U}^{\top}$（単に $\mathbf{u}_i$ と $\mathbf{v}_i$ の役割を入れ替えたもの）に適用すれば導かれる。

これにより、あらゆる線形変換の見通しが驚くほど明快になる。入力空間は r 次元の行空間と $(n-r)$ 次元の零空間という直交する 2 つの部分空間に分解される。同様に、出力空間も r 次元の列空間と $(m-r)$ 次元の左零空間に分解される。行空間の上では、$\mathbf{A}$ は各 $\mathbf{v}_i$ を $\sigma_i \mathbf{u}_i$ へと写し、列空間への 1 対 1 の全単射を与える。一方、零空間の元はすべてゼロへと送られる。

最も単純な非自明の例でこれを見るため、33.2 節の特異行列 $\mathbf{B} = \left(\begin{smallmatrix} 2 & 1 \\ 4 & 2 \end{smallmatrix}\right)$ を考えよう。グラム行列 $\left(\begin{smallmatrix} 20 & 10 \\ 10 & 5 \end{smallmatrix}\right)$ の固有値は 25 と 0 であるため、$\sigma_1 = 5$、$\sigma_2 = 0$、$r = 1$ となる。4 つの部分空間は次の各ベクトルが張る直線である。

$$\mathbf{v}_1 = \tfrac{1}{\sqrt{5}}(2,1), \quad \mathbf{v}_2 = \tfrac{1}{\sqrt{5}}(-1,2), \quad \mathbf{u}_1 = \tfrac{1}{\sqrt{5}}(1,2), \quad \mathbf{u}_2 = \tfrac{1}{\sqrt{5}}(-2,1), \qquad \mathbf{B} = 5\,\mathbf{u}_1\mathbf{v}_1^{\top}.$$

ここで列空間は直線 $y = 2x$（$\mathbf{u}_1$ の方向）であり、左零空間はそれに直交する $\mathbf{u}_2$ 方向の直線である。入力平面においては、零空間は $(1, -2)$ 方向の直線であり、行空間を張る行ベクトル $(2, 1)$ に直交している。Figure 33.26 に一般の幾何学的関係を示す。33.5.3.1 節で見るように、擬似逆行列はこの行空間から列空間への全単射をちょうど逆向きにたどる写像にほかならない。

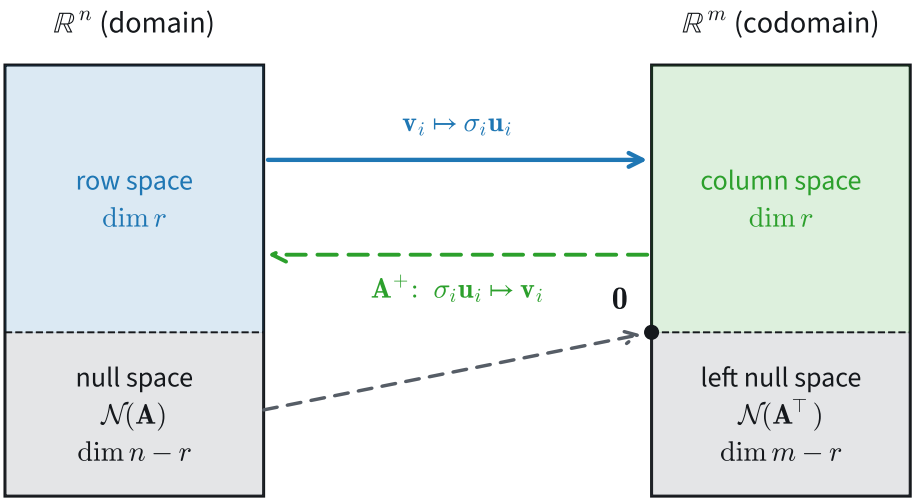


Figure 33.26: 4 つの基本部分空間。$\mathbf{A}$ は行空間 $\operatorname{span}\{\mathbf{v}_1, \dots, \mathbf{v}_r\}$ を列空間 $\operatorname{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$ の上へ $\mathbf{v}_i \mapsto \sigma_i \mathbf{u}_i$ によって 1 対 1 に写し、零空間をゼロへと写す。擬似逆行列 $\mathbf{A}^+$（破線）はこの写像を逆向きに $\sigma_i \mathbf{u}_i \mapsto \mathbf{v}_i$ とたどり、左零空間をゼロへと送る。

数値ランク。浮動小数点演算においては、数学的にランク落ちしている行列であっても、特異値が文字どおり厳密にゼロになるとは限らない。丸め誤差によって $\epsilon_{\text{mach}}\sigma_1$ 程度の微小な値が残るのが普通だからである。そのため実用上は、閾値を用いて数値ランク（numerical rank）を判定する。NumPy の `np.linalg.matrix_rank` は $\sigma_1 \max(m, n)\, \epsilon_{\text{mach}}$ を超える特異値の個数を数えており、この許容誤差は最大特異値と行列の次元の双方に応じて自然にスケールする。ランク 2 の 4×4 行列において、「ゼロ」であるはずの特異値が実際にどのような値になるか見てみよう。

```
A = np.random.standard_normal((4, 2)) @ np.random.standard_normal((2, 4))   # rank <= 2
s = np.linalg.svd(A, compute_uv=False)
tol = s[0] * max(A.shape) * np.finfo(float).eps
print('singular values:', np.array2string(s, formatter={'float_kind': '{:.1e}'.format}))
print(f'tolerance {tol:.1e};  exact zeros: {(s == 0).sum()};  '
      f'numerical rank: {np.linalg.matrix_rank(A)}')
```

```
singular values: [2.4e+00 1.4e+00 2.7e-16 6.3e-17]
tolerance 2.1e-15;  exact zeros: 0;  numerical rank: 2
```

2 つの特異値はオーダー 1 であるのに対し、残りの 2 つは純粋な丸めノイズであり、10^{-15} 以下と許容誤差を大きく下回っている。丸めの結果たまたま片方が厳密なゼロになることはあっても、両方がゼロになることはめったにない。したがって厳密なゼロを数えるとランクを誤判定してしまうが、閾値を用いれば真のランク 2 が正しく復元される。

33.5.2 低ランク近似

33.5.2.1 最良低ランク近似

ここで本節冒頭の猫の写真に戻ろう。どの 20 個のランク 1 成分を残すべきだろうか。ダイアド和 Equation 33.46 はすでに $\mathbf{A}$ の成分を重いものから軽いものへと並べているため、最初の k 項を残すのが自然な候補となる。こうして得られるのが**打ち切り SVD**（truncated SVD）である。

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \, \mathbf{u}_i \mathbf{v}_i^{\top}. \tag{33.51}$$

行列 $\mathbf{A}_k$ のランクは高々 k であり、これを保存するには k 個の特異値、k 本の長さ m の列、k 本の長さ n の列だけで足りるため、合計 $k(m+n+1)$ 個の数値で済む（元の mn 個の代わり）。400×500 の写真で $k = 20$ とすれば、200,000 個のうちわずか 18,020 個の数値である。すると直ちに 2 つの疑問が生じる。近似誤差はどれほど大きいか、そしてもっと巧妙なランク k 行列を作ればさらに良くできるのか、という問いである。

特異値で表される行列ノルム。近似誤差を測るには行列ノルムが必要である。1.3 節で導入したフロベニウスノルム $\|\mathbf{A}\|_F^2 = \sum_{ij} a_{ij}^2$ は、$\mathbf{A}$ の各列の長さの二乗和であり、それらは $\mathbf{A}^{\top}\mathbf{A}$ の対角成分に等しい。したがって $\|\mathbf{A}\|_F^2 = \operatorname{tr}(\mathbf{A}^{\top}\mathbf{A})$ となる。ここに SVD を代入し、トレースの巡回性（33.2 節）を使って $\mathbf{V}$ を反対側へ移すと次式が得られる。

$$\|\mathbf{A}\|_F^2 = \operatorname{tr}(\mathbf{V}\Sigma^{\top}\Sigma\mathbf{V}^{\top}) = \operatorname{tr}(\Sigma^{\top}\Sigma\,\mathbf{V}^{\top}\mathbf{V}) = \sum_{i} \sigma_i^2. \tag{33.52}$$

ゆえにフロベニウスノルムは特異値の二乗和の平方根であり、ここから左右どちらから直交行列を掛けてもフロベニウスノルムは決して変わらないことが直ちにわかる。せん断行列では $1+1+0+1 = 3 = \varphi^2 + \varphi^{-2}$ となる。一方、スペクトルノルムは単なる最大特異値 $\|\mathbf{A}\|_2 = \sigma_1$ である（Equation 33.48）。

両方のノルムが特異値で表されたところで、線形代数における中心的な近似定理を述べよう。

定理（Eckart–Young–Mirsky）。$\sigma_i = 0$ $(i > r)$ とおく。すべての $k \geq 0$ に対し、ランクが高々 k の行列で達成可能な最小のスペクトルノルム誤差は次式で与えられる。

$$\min_{\operatorname{rank}\mathbf{B} \leq k} \|\mathbf{A} - \mathbf{B}\|_2 = \sigma_{k+1}, \tag{33.53}$$

また、最小の二乗フロベニウス誤差は $\min_{\operatorname{rank}\mathbf{B}\leq k}\|\mathbf{A}-\mathbf{B}\|_F^2 = \sum_{i>k}\sigma_i^2$ である。いずれの最小値も、打ち切り SVD である $\mathbf{B} = \mathbf{A}_k$ において達成される。

要するに、どちらのノルムにおいても、いかなるランク k 行列も打ち切り SVD を上回ることはできない。最良のランク k 近似の誤差は、切り捨てた最初の特異値（スペクトルノルムの場合）、あるいは切り捨てたすべての特異値のエネルギーの総和（フロベニウスノルムの場合）に一致する。Eckart と Young がフロベニウスノルムの場合を証明し (Eckart and Young 1936)、後に Mirsky が左右両側からの直交行列の積に対して不変なすべてのノルムへと拡張した (Mirsky 1960)。まずスペクトルノルムの評価を証明しよう。フロベニウスノルムの評価は、同じ議論からそのまま導かれるもう 1 つの道具である Weyl の不等式を使えば明快に従う。

一般の証明を書く前に、$n = 3$、$k = 1$ の場合を思い描いてみよう。ランク 1 の任意の行列 $\mathbf{B}$ は入力のある平面全体をゼロに潰す一方、$\mathbf{A}$ の上位 2 本の右特異ベクトルは別の平面を張る。$\mathbb{R}^3$

において原点を通る 2 つの平面は、少なくとも 1 本の直線で交わらなければならない。その直線上では、$\mathbf{B}$ の出力はゼロであるのに対し、$\mathbf{A}$ はすべてのベクトルを少なくとも σ_2 倍に引き伸ばす。したがって $\mathbf{A} - \mathbf{B}$ は、ある単位ベクトルを少なくとも σ_2 倍に引き伸ばさざるを得ない（Figure 33.27）。以下の証明は、この次元の勘定を n 次元へそのまま持ち上げたものである。

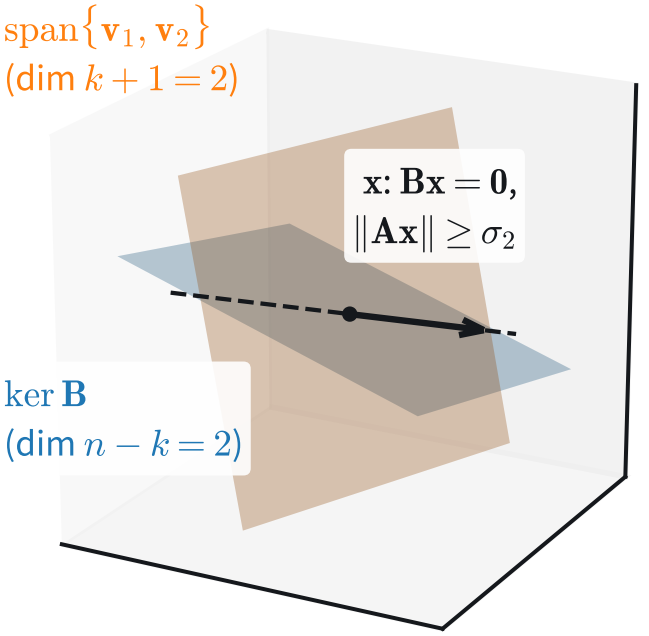


Figure 33.27: $n = 3$、$k = 1$ における Eckart–Young の背後にある次元の勘定。平面 $\ker \mathbf{B}$ と上位 2 本の右特異ベクトルが張る平面は直線で交わらなければならない。その直線上では $\mathbf{B}$ はゼロになる一方、$\mathbf{A}$ は少なくとも σ_2 倍に引き伸ばす。

証明（スペクトルノルム）。 第 1 に、打ち切り SVD は誤差 σ_{k+1} を達成する。残差 $\mathbf{A} - \mathbf{A}_k = \sum_{i>k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ はすでに最大特異値 σ_{k+1} を持つコンパクト SVD の形をしており、$\|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1}$ が Equation 33.48 より従う。同様に、Equation 33.52 から $\|\mathbf{A} - \mathbf{A}_k\|_F^2 = \sum_{i>k} \sigma_i^2$ も得られる。

第 2 に、ランクが高々 k のいかなる行列もこれを下回ることはできない。$k \geq r$ ならば誤差はゼロで証明すべきことはないため、$k < r$ と仮定し、$\operatorname{rank} \mathbf{B} \leq k$ とする。ランク・退化次数の定理（33.2 節）より、零空間 $\mathcal{X} = \ker \mathbf{B}$ の次元は少なくとも $n - k$ である。一方、張る空間 $\mathcal{V} = \operatorname{span}\{\mathbf{v}_1, \dots, \mathbf{v}_{k+1}\}$（上位 $k+1$ 本の右特異ベクトルによるもの）の次元は $k+1$ である。これら 2 つの次元の和は n を超えるため、2 つの部分空間は非ゼロベクトルを共有しなければならない。

$$\dim(\mathcal{X} \cap \mathcal{V}) = \dim \mathcal{X} + \dim \mathcal{V} - \dim(\mathcal{X} + \mathcal{V}) \geq (n-k) + (k+1) - n = 1,$$

ここで等号は 2 つの部分空間の次元公式であり、不等号は $\dim(\mathcal{X} + \mathcal{V}) \leq n$ を用いている（次元公式を確かめるには、$\mathcal{X} \cap \mathcal{V}$ の基底をそれぞれ $\mathcal{X}$ の基底と $\mathcal{V}$ の基底へ別々に延長すればよい。合わせたベクトル群は線形独立で $\mathcal{X} + \mathcal{V}$ を張り、その本数は $\dim \mathcal{X} + \dim \mathcal{V} - \dim(\mathcal{X} \cap \mathcal{V})$

となる)。そこで単位ベクトル $\mathbf{x} \in \mathcal{K} \cap \mathcal{V}$ を 1 つ選び、$\mathbf{x} = \sum_{i \le k+1} c_i \mathbf{v}_i$（$\sum_i c_i^2 = 1$）と展開する。すると $\mathbf{B}\mathbf{x} = \mathbf{0}$ であり、$\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$ と $\mathbf{u}_i$ の正規直交性を用いれば次式を得る。

$$\|\mathbf{A}\mathbf{x}\|^2 = \Big\| \sum_{i \le k+1} c_i \sigma_i \mathbf{u}_i \Big\|^2 = \sum_{i \le k+1} c_i^2 \sigma_i^2 \ge \sigma_{k+1}^2 \sum_{i \le k+1} c_i^2 = \sigma_{k+1}^2 .$$

ゆえに $\|\mathbf{A} - \mathbf{B}\|_2 \ge \|(\mathbf{A} - \mathbf{B})\mathbf{x}\| = \|\mathbf{A}\mathbf{x}\| \ge \sigma_{k+1}$ が成り立つ。$\blacksquare$

この幾何学的意味は一文で繰り返しておく価値がある。任意のランク k の競合行列 $\mathbf{B}$ は、上位 $k+1$ 本の右特異ベクトルが張る空間の中の少なくとも 1 方向をゼロに潰さなければならず、そして $\mathbf{A}$ はその空間内のあらゆる方向を少なくとも σ_{k+1} 倍に引き伸ばすのである。せん断行列 $\mathbf{T}$ で $k=1$ とすると、打ち切り $\mathbf{T}_1 = \varphi\, \mathbf{u}_1 \mathbf{v}_1^{\top} = \frac{1}{\sqrt{5}} \left(\begin{smallmatrix} \varphi & \varphi^2 \\ 1 & \varphi \end{smallmatrix} \right) \approx \left(\begin{smallmatrix} 0.724 & 1.171 \\ 0.447 & 0.724 \end{smallmatrix} \right)$ の誤差は両ノルムで $1/\varphi = 0.618$ となり（残差がランク 1 であるためスペクトルノルムとフロベニウスノルムが一致する）、いかなるランク 1 行列もこれより近づくことはできない。

フロベニウスノルムについて定理の証明を完結させるため、今証明したスペクトルノルムの評価から一般的な摂動不等式を先に導いておこう。

補題（特異値に対する Weyl の不等式）。同じ形状を持つ任意の 2 つの行列 $\mathbf{X}, \mathbf{Y}$ と、添字 $i, j \ge 1$ $(i + j - 1 \le \min(m,n))$ に対し、次式が成り立つ。

$$\sigma_{i+j-1}(\mathbf{X} + \mathbf{Y}) \le \sigma_i(\mathbf{X}) + \sigma_j(\mathbf{Y}). \tag{33.54}$$

証明。 $\mathbf{X}_{i-1}$ と $\mathbf{Y}_{j-1}$ を打ち切り SVD Equation 33.51 とする（$\mathbf{X}_0 = \mathbf{0}$ とおく）。その和 $\mathbf{X}_{i-1} + \mathbf{Y}_{j-1}$ のランクは高々 $i+j-2$ であるから、スペクトルノルムの下界を $\mathbf{X}+\mathbf{Y}$ に $k = i+j-2$ として適用すれば $\sigma_{i+j-1}(\mathbf{X}+\mathbf{Y}) \le \|\mathbf{X}+\mathbf{Y}-\mathbf{X}_{i-1}-\mathbf{Y}_{j-1}\|_2$ が得られる。三角不等式（triangle inequality）と Eckart–Young の達成可能性の議論より、右辺は高々 $\|\mathbf{X}-\mathbf{X}_{i-1}\|_2 + \|\mathbf{Y}-\mathbf{Y}_{j-1}\|_2 = \sigma_i(\mathbf{X}) + \sigma_j(\mathbf{Y})$ である。$\blacksquare$

言葉で言えば、$\mathbf{Y}$ を $\mathbf{X}$ に加えたとき、第 $(i+j-1)$ 特異値が第 i 特異値（$\mathbf{X}$ のもの）を上回る量は高々 $\sigma_j(\mathbf{Y})$ に抑えられる。$j = 1$ として不等式を両方向に適用すれば、行列に $\mathbf{Y}$ の摂動を加えてもいかなる特異値も $\|\mathbf{Y}\|_2$ を超えて動かないことがわかる (Horn and Johnson 2012)。詳細は演習問題 2 で扱う。

証明（フロベニウスノルム）。 Weyl の補題を $\mathbf{X} = \mathbf{A} - \mathbf{B}$、$\mathbf{Y} = \mathbf{B}$、$j = k+1$ として適用する。$\operatorname{rank} \mathbf{B} \le k$ より $\sigma_{k+1}(\mathbf{B}) = 0$ であるから、$\sigma_{k+i}(\mathbf{A}) \le \sigma_i(\mathbf{A} - \mathbf{B})$ がすべての $i \ge 1$ に対して成り立つ。両辺を二乗して i について和をとり、Equation 33.52 を適用すると次式を得る。

$$\|\mathbf{A} - \mathbf{B}\|_F^2 = \sum_{i \ge 1} \sigma_i(\mathbf{A} - \mathbf{B})^2 \ge \sum_{i \ge 1} \sigma_{k+i}(\mathbf{A})^2 = \sum_{j > k} \sigma_j(\mathbf{A})^2 ,$$

そしてこの下界を打ち切り $\mathbf{A}_k$ が達成する。$\blacksquare$

最小点の一意性。 2 つのノルムの間には 1 つ微妙な違いがある。スペクトルノルムでは、最良ランク k 近似は $k \ge 1$ かつ $\sigma_{k+1} > 0$ である限り一意になることは決してない。たとえば $\mathbf{A} = \operatorname{diag}(3,2,1)$、$k=1$ のとき、行列 $\mathbf{B} = \operatorname{diag}(3+t, 0, 0)$ $(|t| \le 1)$ はすべて $\|\mathbf{A} - \mathbf{B}\|_2 = \max(|t|, 2, 1) = 2 = \sigma_2$ を与える。対照的にフロベニウスノルムでは、$\mathbf{A}_k$ は $\sigma_k > \sigma_{k+1}$ または $\sigma_{k+1} = 0$（この場合は $\mathbf{A}_k = \mathbf{A}$）である限り**一意**な最小点（minimizer）となる (Golub and Van Loan 1996)。次に見る主成分分析がフロベニウスノルムで定式化される理由の 1 つが、この一意性にある。

それでは、400×500 の猫の写真でこの定理を検証してみよう。Eckart–Young は各打ち切りのスペクトル誤差を σ_{k+1} に定めるが、実際の画像の特異値がどれほど速く減衰するかや、素朴なランク k の競合手法がどれほど劣るかまでは教えてくれない。以下のセルでは `img/cat1.jpg` を読み込み、赤・緑・青に対する標準的な輝度重みを使ってグレースケールへ変換し、`np.einsum` を用いて打ち切り近似を構築する（この関数は入力を掛け合わせ、矢印の後に省かれたすべての添字について和をとる。第 i 行（`Vt` の行）は $\mathbf{v}_i^\top$ であるため、`'i,mi,in->mn'` でダイアド和が計算される）。

```
img = d2l.plt.imread('../img/cat1.jpg') @ np.array([0.299, 0.587, 0.114])  # 400 x 500
U, s, Vt = np.linalg.svd(img, full_matrices=False)
m, n = img.shape
fro = np.linalg.norm(img)
energy = np.cumsum(s**2) / (s**2).sum()                       # energy kept at each rank
slope = np.polyfit(np.log(np.arange(1, 101)), np.log(s[:100]), 1)[0]
print(f'{m} x {n} = {m * n} numbers;  slope of log sigma_i on log i: {slope:.2f}')
print(f'energy kept at rank 1: {energy[0]:.3f}, at rank 4: {energy[3]:.3f}')
for k in [1, 5, 20, 50]:
    A_k = np.einsum('i,mi,in->mn', s[:k], U[:, :k], Vt[:k])    # first k dyads
    print(f'k={k:2d}: ||A-A_k||_2={np.linalg.norm(img - A_k, 2):6.0f}  '
          f'sigma_{k + 1}={s[k]:6.0f}  rel. err {np.linalg.norm(img - A_k) / fro:.3f}  '
          f'stores {k * (m + n + 1) / (m * n):.1%}')
cols = np.argsort(-np.linalg.norm(img, axis=0))[:20]          # 20 largest columns
C = img[:, cols]
proj = C @ np.linalg.pinv(C) @ img
print(f'projection onto the 20 largest columns: rel. err {np.linalg.norm(img - proj) /
 ↪  fro:.3f}')
```

```
400 x 500 = 200000 numbers;  slope of log sigma_i on log i: -1.11
energy kept at rank 1: 0.886, at rank 4: 0.955
k= 1: ||A-A_k||_2=  9234  sigma_2=  9234  rel. err 0.337  stores 0.5%
k= 5: ||A-A_k||_2=  4048  sigma_6=  4048  rel. err 0.191  stores 2.3%
k=20: ||A-A_k||_2=   874  sigma_21=   874  rel. err 0.080  stores 9.0%
k=50: ||A-A_k||_2=   342  sigma_51=   342  rel. err 0.045  stores 22.5%
projection onto the 20 largest columns: rel. err 0.310
```

保証どおり、すべてのランクでスペクトル誤差は σ_{k+1} と一致し、ランク 20 ではわずか 9% の数値を保持するだけで相対誤差 8% に収まっている。Figure 33.28 を見ると、猫はランク 1 では明るいぼやけた塊にすぎないが、ランク 5 では 4 本の足を持つシルエットとして識別でき、ランク 20 になると仰向けに寝転ぶ猫だと紛れもなくわかる。左パネルを見れば、なぜこれほど少数の成分で足りるのかが明らかになる。最初の 100 個の添字にわたって、特異値はおよそべき乗則 $\sigma_i \propto i^{-1.1}$ に従って減衰しているのである。

エネルギー比。フロベニウス誤差の公式は、k を選ぶための実用的な指針を与える。二乗「エネルギー」の総和 $\sum_i \sigma_i^2$ のうち、ランク k の打ち切りによって保持される割合は次式で表される。

$$\frac{\sum_{i \le k} \sigma_i^2}{\sum_i \sigma_i^2} = 1 - \frac{\|\mathbf{A} - \mathbf{A}_k\|_F^2}{\|\mathbf{A}\|_F^2}, \tag{33.55}$$

すなわち、相対フロベニウス誤差は 1 からエネルギー比を引いた値の平方根になる。したがってエネルギーの 95% を保持することは相対誤差 $\sqrt{0.05} \approx 22\%$ に対応し、99% を保持すれば誤差は 10% まで下がる。ただし、この比率の解釈には注意が必要である。上の写真では、定数画像がランク 1 であり（すべての行が同一）、かつ写真の平均輝度が大きいため、最初のダイアドが主に

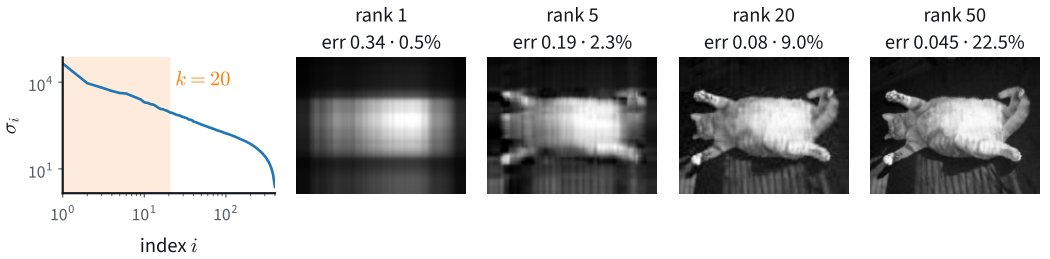


Figure 33.28: 400×500 の写真における Eckart–Young。左：特異値はおよそべき乗則に従って減衰する（両対数軸。網掛け帯はランク 20 のパネルで残した 20 個を示す）。右：各打ち切り $\mathbf{A}_k$ と、その相対フロベニウス誤差および元の数値数に対する保持割合 $k(m+n+1)/mn$。

その平均を捉えることになり、σ_1^2 だけでエネルギーの 89% を占めてしまう。そのままではランク 4 ですでにエネルギーの 95% を保持する（相対誤差 21%）が、ピクセル値の平均をあらかじめ差し引くと、95% に達するにはランク 8 が必要になる（4 ではなくなることは 33.5.4.3 節で確認する）。常に σ_1 が非ゼロの平均に支配されていないか確認するか、主成分分析と同様にまず平均を差し引くべきである。

素朴なランク 20 の手法と比べて Eckart–Young がどれほど優れているかを見るため、セルの末尾では写真からノルムが最大の実列を 20 本選び、すべての列をそれらが張る空間へ射影している（行列自身の行と列の部分集合で近似する CUR 分解の背後にある考え方である）。その相対誤差は 0.31 であり、打ち切り SVD の 0.08 と比べて 4 倍近く悪い。

ノイズ除去としての打ち切り。 SVD を打ち切る理由は圧縮だけではなく、ノイズ除去（denoising）も重要な用途である。観測された $m \times n$ 行列（$m \le n$）が低ランクの信号にランダムノイズが加わった $\mathbf{A} = \mathbf{B} + \mathbf{N}$ であり、$\mathbf{N}$ の各成分が平均 0、標準偏差 σ_{noise} で独立だとする。33.3.2.4 節のマルチェンコ・パスツール則を $\mathbf{N}\mathbf{N}^{\top}/n$ に適用すると、大きなノイズ行列の特異値は区間 $[(\sqrt{n}-\sqrt{m})\sigma_{\text{noise}}, (\sqrt{n}+\sqrt{m})\sigma_{\text{noise}}]$ の内部に集中する。特異値がこの上端を大きく超える信号成分はノイズの塊（バルク）からくっきりと浮かび上がる一方、弱い成分はその中に埋もれて消える。ノイズレベル（noise level）が既知の正方行列に対し、Gavish と Donoho は漸近的に最適なハード閾値が $\frac{4}{\sqrt{3}}\sqrt{n}\,\sigma_{\text{noise}} \approx 2.309\sqrt{n}\,\sigma_{\text{noise}}$ であり、ノイズの上端 $2\sqrt{n}\,\sigma_{\text{noise}}$ よりわずかに上に位置することを証明した (Gavish and Donoho 2014)。演習問題 8 で定数 $4/\sqrt{3}$ を導出する。

33.5.2.2 主成分分析

各データ例を行として並べたデータセット $\mathbf{X} \in \mathbb{R}^{n \times d}$（データ点数 n、特徴量数 d）が与えられたとしよう。データの変動の大半がごく少数の方向に集中していることは多い。ここで 2 つの自然な問いが生じる。データの最大の分散を捉える単位方向はどれか、そして二乗誤差を最小にする k 次元アフィン部分空間はどれか、という問いである。驚くべきことに、両者の答えは完全に一致し、中心化したデータの打ち切り SVD そのものになる。

中心化。 まず任意の k 次元部分空間とその直交射影行列 $\mathbf{P}$ を固定し、どのオフセット $\mathbf{c}$ が総二乗再構成誤差（reconstruction error）$\sum_i \|(\mathbf{I}-\mathbf{P})(\mathbf{x}_i - \mathbf{c})\|^2$ を最小化するかを考えよう。この目的関数は $\mathbf{c}$ に関する凸二次関数であり、その勾配 $-2(\mathbf{I}-\mathbf{P})\sum_i(\mathbf{x}_i - \mathbf{c})$ は標本平均 $\mathbf{c} = \bar{\mathbf{x}}$ において $\mathbf{P}$ によらずゼロになる。したがって最良のアフィン部分空間は常に平均を通るため、まずデータの**中心化**（centering）から始める。すなわち $\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^{\top}$ とおく。その経験共分散は対称半正定値行列 $\mathbf{C} = \frac{1}{n}\tilde{\mathbf{X}}^{\top}\tilde{\mathbf{X}}$ である（不偏分散の係数 $\frac{1}{n-1}$ を使ってもすべての固有値が同じ倍率でスケールされるだけであり、主方向もそれらが説明する分散の割合も変わらない）。

命題（Rayleigh による PCA）。 すべての単位ベクトル $\mathbf{w}$ にわたって、射影後の分散 $\frac{1}{n}\|\tilde{\mathbf{X}}\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{C}\mathbf{w}$ を最大化するのは $\mathbf{C}$ の最大固有ベクトル、すなわち最大右特異ベクトル $\mathbf{v}_1$（$\tilde{\mathbf{X}}$ のもの）であり、最大分散は σ_1^2/n である。

証明。 中心化した点を $\mathbf{w}$ の上へ射影するとスカラー座標 $\tilde{\mathbf{x}}^\top\mathbf{w}$ が得られ、その経験分散は $\frac{1}{n}\sum_i (\tilde{\mathbf{x}}_i^\top \mathbf{w})^2 = \frac{1}{n}\|\tilde{\mathbf{X}}\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{C}\mathbf{w}$ となる。33.3.2.3 節の Rayleigh の命題より、この二次形式は $\mathbf{C}$ の最大固有ベクトルで最大値 $\lambda_1(\mathbf{C})$ をとる。Equation 33.49 より $\mathbf{C} = \frac{1}{n}\mathbf{V}\Sigma^\top\Sigma\mathbf{V}^\top$ であるから、$\mathbf{C}$ の固有ベクトルは右特異ベクトル $\mathbf{v}_i$（$\tilde{\mathbf{X}}$ のもの）であり、固有値は σ_i^2/n である。■

$\mathbf{v}_1$ に直交する単位ベクトルの中では分散は $\mathbf{v}_2$ で最大化され、帰納的に続ければ上位 k 個の**主方向**（principal direction）$\mathbf{v}_1, \dots, \mathbf{v}_k$ が得られ、第 i 成分が分散 σ_i^2/n を説明する。ここで再構成誤差を確認しよう。中心化した各点を $\operatorname{span}\{\mathbf{v}_1,\dots,\mathbf{v}_k\}$ へ射影する操作は、$\tilde{\mathbf{X}}$ に右から $\mathbf{V}_k\mathbf{V}_k^\top$ を掛けることに対応する。$\mathbf{V}^\top\mathbf{V}_k$ は単位行列の最初の k 列を取り出すため、次式が成り立つ。

$$\tilde{\mathbf{X}}\mathbf{V}_k\mathbf{V}_k^\top = \mathbf{U}\Sigma\mathbf{V}^\top\mathbf{V}_k\mathbf{V}_k^\top = \mathbf{U}_k\Sigma_k\mathbf{V}_k^\top = \tilde{\mathbf{X}}_k.$$

したがって、上位 k 個の主方向への射影は、中心化データのランク k 打ち切り SVD をとることと完全に一致する。フロベニウスノルムにおける Eckart–Young の定理より、他のいかなる射影も、さらには他のいかなるランク k 行列も、より小さい二乗誤差で中心化データ点を再構成することはできない。切り捨てられた分散 $\sum_{i>k}\sigma_i^2/n$ は平均二乗再構成誤差に等しく、説明分散比（explained-variance ratio）はエネルギー比 Equation 33.55（$\tilde{\mathbf{X}}$ に対するもの）そのものである。

Figure 33.29 に、300 個の点からなる点群（標準偏差 3 と 0.8 を持ち、$\pi/5 = 36^\circ$ 回転した軸に沿う分布）と、その 2 本の主軸を示す。母集団においては、第 1 方向が分散の $9/9.64 \approx 93.4\%$ を占め、長軸に一致する。以下のセルでは、300 点の有限標本からこれらの値がどれほど正確に復元されるかを確認する。

```
theta = np.pi / 5
rot = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
X = np.random.standard_normal((300, 2)) * np.array([3.0, 0.8]) @ rot.T   # the cloud
Xc = X - X.mean(0)
U, s, Vt = np.linalg.svd(Xc, full_matrices=False)
X1 = np.outer(U[:, 0] * s[0], Vt[0])                                    # rank-1 reconstruction
print('explained-variance ratio:', (s**2 / (s**2).sum()).round(3), ' (population 0.934,
 ↪  0.066)')
print('angle of v_1 to the long axis:',
      np.degrees(np.arccos(abs(Vt[0] @ rot[:, 0]))).round(1), 'degrees')
print('mean squared rank-1 error:', (((Xc - X1)**2).sum() / 300).round(4),
      '  sigma_2^2 / n:', (s[1]**2 / 300).round(4))
```

```
explained-variance ratio: [0.932 0.068]  (population 0.934, 0.066)
angle of v_1 to the long axis: 0.1 degrees
mean squared rank-1 error: 0.6082   sigma_2^2 / n: 0.6082
```

乱数抽出を繰り返すと、第 1 主方向は分散のおよそ 91% から 95% を説明し、真の長軸から数度以内に収まる。しかしデータの中心化を忘れると、平均のオフセットが点群の広がりを超えた途端に、第 1 特異ベクトルは点群の平均の方向へと振れてしまい、点群が**どのように**変動しているかではなく**どこに**位置しているかを表すようになってしまう。演習問題 3 でこの影響を定量化する。

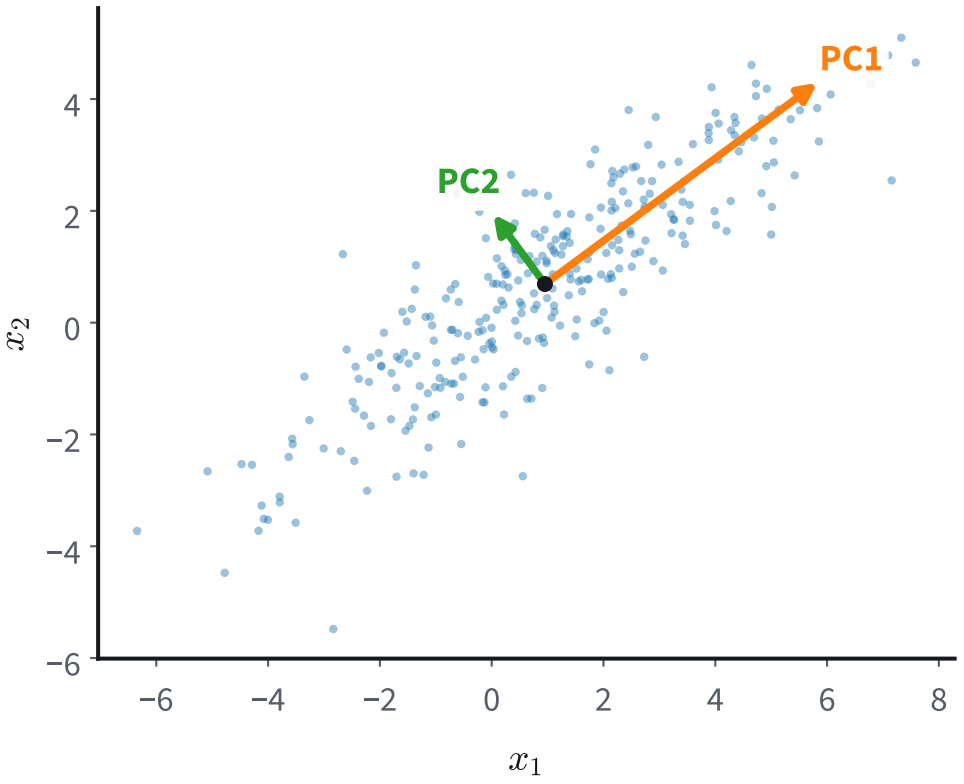


Figure 33.29: 300 個の点からなる点群（中心 $(1, 0.6)$、標準偏差 3 と 0.8、回転角 36° の軸に沿う分布）と、平均から描いた主軸（各軸の長さは標準偏差の 2 倍である $2\sigma_i/\sqrt{n}$）。第 1 主軸は最大分散の方向を指す。

33.5.3 SVD による連立一次方程式の解法

33.5.3.1 擬似逆行列と最小二乗法

未知数より方程式が多い連立一次方程式 $\mathbf{Ax} = \mathbf{b}$ は一般に厳密解を持たず、逆に未知数より方程式が少ない連立一次方程式は無数の解を持つ。どちらの状況でも、33.1 節で導入した最小二乗法の解を求める（ここでは目的ベクトルを $\mathbf{b}$ と書き、$\mathbf{y}$ の代わりに用いる）。

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2,$$

さらに、複数の $\mathbf{x}$ が同じ最小残差を達成するときは、その中でノルムが最小のものを選ぶ。

まず最も単純な劣決定（underdetermined）の例として、2 つの未知数に対する 1 本の方程式 $x_1 + x_2 = 2$ を考えよう。解は直線全体をなし、その上のどこでも残差は 0 である。直線上のどの点を選ぶべきだろうか。ここでは $\mathbf{A} = \begin{pmatrix} 1 & 1 \end{pmatrix}$ であり、非ゼロ特異値は $\sigma_1 = \sqrt{2}$ の 1 つだけで、右特異ベクトルは $\mathbf{v}_1 = (1,1)/\sqrt{2}$、左特異ベクトルは $u_1 = 1$ となる。この方程式は $\mathbf{x}$ の $\mathbf{v}_1$ 方向の座標を $\sqrt{2}$ に固定する一方、零空間方向 $\mathbf{v}_2 = (1,-1)/\sqrt{2}$ に沿う座標は残差に一切影響しない。その自由な座標を 0 に設定すると、原点から直線へ下ろした垂線の足 $(1,1)$（ノルム $\sqrt{2}$）が得られ、他のあらゆる解 $(1+t, 1-t)$ は真に大きい二乗ノルム $2+2t^2$ を持つ（Figure 33.30）。

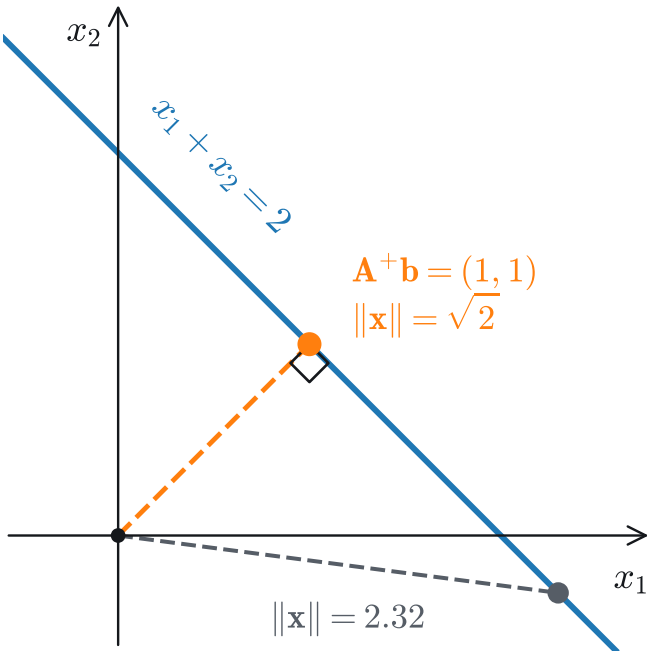


Figure 33.30: 方程式 1 本・未知数 2 個の最小二乗法。直線上のすべての点が $x_1 + x_2 = 2$ を満たし、擬似逆行列は原点に最も近い点、すなわち垂線の足を返す。

一般の規則は、単にすべての非ゼロ特異値の逆数をとり、直交因子を転置するだけである。$\mathbf{A}$ の**ムーア・ペンローズ擬似逆行列**（Moore–Penrose pseudoinverse）は次式で定義される。

$$\mathbf{A}^{+} = \mathbf{V}\Sigma^{+}\mathbf{U}^{\top}, \qquad \Sigma^{+}_{ii} = \begin{cases} 1/\sigma_i & \sigma_i > 0, \\ 0 & \sigma_i = 0, \end{cases} \tag{33.56}$$

ここで Σ^{+} の形状は $n \times m$ であり、Σ を転置した形状になっている。幾何学的には、$\mathbf{A}^{+}$ は元に戻せるすべての伸縮を逆転させ、左零空間をゼロへと送る。せん断行列に対しては $\mathbf{T}^{+} = \mathbf{T}^{-1} = \left(\begin{smallmatrix} 1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$ となり、$\mathbf{A} = \begin{pmatrix} 1 & 1 \end{pmatrix}$ に対しては $\mathbf{A}^{+} = \mathbf{v}_1 \sigma_1^{-1} u_1 = \left(\tfrac{1}{2}, \tfrac{1}{2}\right)^{\top}$ となるため、$\mathbf{A}^{+} \cdot 2 = (1, 1)$ が得られる。

命題（最小ノルム最小二乗法）。$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|$ のすべての最小点の中で、ノルムが最小となる一意の解は $\hat{\mathbf{x}} = \mathbf{A}^{+}\mathbf{b}$ である。

証明。 $\mathbf{y} = \mathbf{V}^{\top}\mathbf{x}$、$\mathbf{c} = \mathbf{U}^{\top}\mathbf{b}$ とおいて特異基底へ変数変換する。$\mathbf{U}$ は直交行列でありユークリッドノルムを保つため、次式が成り立つ。

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 = \|\mathbf{U}^{\top}(\mathbf{A}\mathbf{x} - \mathbf{b})\|^2 = \|\Sigma\mathbf{y} - \mathbf{c}\|^2 = \sum_{i \le r} (\sigma_i y_i - c_i)^2 + \sum_{i > r} c_i^2 .$$

特異基底へ回転することで、連立方程式は互いに独立なスカラー方程式へと分離される。第 1 の和は、$y_i = c_i/\sigma_i$（$i \le r$）とおくことで項ごとに最小化される。第 2 の和は $\mathbf{b}$ の列空間外の成分を表す固定定数であり、$\mathbf{x}$ をどう選んでも減らすことはできない。一方、座標 y_i（$i > r$）は残差に一切現れず、$\|\mathbf{x}\| = \|\mathbf{y}\|$ であるから、それらを 0 に設定することで最小ノルムの一意な最小点が得られる。この選択は $\mathbf{y} = \Sigma^{+}\mathbf{c}$ であり、元の変数へ戻せば $\hat{\mathbf{x}} = \mathbf{V}\Sigma^{+}\mathbf{U}^{\top}\mathbf{b} = \mathbf{A}^{+}\mathbf{b}$ となる。$\blacksquare$

特異基底で見ると、残差の最小化とノルムの最小化は完全に互いに素な座標群（$i \le r$ と $i > r$）を支配しており、両方の目的が同時に達成される。$\mathbf{A}$ が正方かつ可逆のときは、すべての σ_i が正となり、$\mathbf{A}^{+} = \mathbf{V}\Sigma^{-1}\mathbf{U}^{\top} = \mathbf{A}^{-1}$ となる。しかし同じ式は、最小二乗法の主な危険性もあらわにする。微小な σ_i で割ることにより、$\mathbf{b}$ に含まれる $\mathbf{u}_i$ 方向のノイズが $1/\sigma_i$ 倍に増幅されてしまうのである。

この不安定な方向を抑え込む標準的な処方箋が 2 つある。**リッジ回帰**（ridge regression）は $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda\|\mathbf{x}\|^2$（$\lambda > 0$）を最小化し、$\mathbf{x}_\lambda = (\mathbf{A}^{\top}\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^{\top}\mathbf{b}$ を与える。一方、**打ち切り擬似逆行列**（truncated pseudoinverse）による解 $\mathbf{x}_k$ は、最初の k 個の特異値項だけを $\mathbf{A}^{+}\mathbf{b}$ から残す。SVD を代入し、$\mathbf{A}^{\top}\mathbf{A} + \lambda\mathbf{I} = \mathbf{V}(\Sigma^{\top}\Sigma + \lambda\mathbf{I})\mathbf{V}^{\top}$ に注意すると、両方の解は見通しのよいフィルタ形式をとる。

$$\mathbf{x}_\lambda = \sum_{i \le r} \frac{\sigma_i^2}{\sigma_i^2 + \lambda} \, \frac{\mathbf{u}_i^{\top}\mathbf{b}}{\sigma_i} \, \mathbf{v}_i, \qquad \mathbf{x}_k = \sum_{i \le k} \frac{\mathbf{u}_i^{\top}\mathbf{b}}{\sigma_i} \, \mathbf{v}_i . \tag{33.57}$$

どちらも正則化なしの擬似逆行列の項 $\mathbf{u}_i^{\top}\mathbf{b}/\sigma_i$ から出発する。リッジ回帰は第 i 成分に滑らかなフィルタ係数 $\sigma_i^2/(\sigma_i^2 + \lambda)$ を掛け、この係数は $\sigma_i^2 \gg \lambda$ のとき 1 付近にとどまり、$\sigma_i^2 \ll \lambda$ のとき 0 へと縮む。これに対し、打ち切りは 1 か 0 かの急峻な階段関数を適用する。たとえば $\lambda = 1$ のとき、特異値 3、1、0.3 に対するフィルタ係数はそれぞれ 0.9、0.5、0.083 となる。二乗損失で訓練される線形モデルにおいて、重み減衰（weight decay、2.7 節）はまさにリッジ回帰そのものであり、これらのフィルタ係数はどの方向が減衰されるかを正確に示している。

正規方程式。 古典的な教科書における最小二乗法の導出では、正規方程式（normal equations）$\mathbf{A}^{\top}\mathbf{A}\mathbf{x} = \mathbf{A}^{\top}\mathbf{b}$ を解く。任意の $\mathbf{A}$ に対してその解は最小二乗法の最小点となり（$\mathbf{A}^{\top}(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{0}$ は残差が列空間に直交することを表すため）、$\mathbf{A}$ が列フルランクであれば解は一意に定まる。しかし数値計算上、$\mathbf{A}^{\top}\mathbf{A}$ を直接作るのは得策ではない。条件数が二乗されてしまうからである

(33.5.3.2 節)。そのため現代のライブラリは、代わりに SVD や QR 分解を用いて最小二乗問題を解く。

過剰決定（overdetermined）系と劣決定系を 1 つずつ使って擬似逆行列を試してみよう。1 つ目は直線 $b \approx x_1 + x_2 t$ を 4 つの観測値 $\mathbf{b} = (6, 5, 7, 10)$（$t = 1, 2, 3, 4$）に当てはめる問題であり、2 つ目は先ほどの 1 本の方程式 $x_1 + x_2 = 2$ である。

```
A = np.array([[1., 1.], [1., 2.], [1., 3.], [1., 4.]])     # 4 equations, 2 unknowns
b = np.array([6., 5., 7., 10.])
x_pinv = np.linalg.pinv(A) @ b
x_lstsq = np.linalg.lstsq(A, b, rcond=None)[0]
print('overdetermined: pinv', x_pinv.round(4), ' lstsq', x_lstsq.round(4),
      ' residual', np.linalg.norm(A @ x_pinv - b).round(4))
A1 = np.array([[1., 1.]])                                   # 1 equation: x1 + x2 = 2
x1 = np.linalg.pinv(A1) @ np.array([2.])
print('underdetermined: pinv gives', x1.round(4), ' norm', np.linalg.norm(x1).round(4))
```

```
overdetermined: pinv [3.5 1.4]  lstsq [3.5 1.4]  residual 2.0494
underdetermined: pinv gives [1. 1.]  norm 1.4142
```

擬似逆行列と `np.linalg.lstsq` は当てはめ直線 $3.5 + 1.4t$（残差 $\sqrt{4.2} \approx 2.049$）で一致し、1 本の方程式に対しては擬似逆行列は垂線の足 $(1, 1)$ を返す。

33.5.3.2 条件数

実際の計算では、連立一次方程式の各成分はノイズを含む測定値や有限精度の丸めから生じる。計算された解を信頼する前に、データのわずかな変化が答えをどれほど変え得るかを知っておく必要がある。単純な 2×2 の系を見るだけで、その増幅がいかに劇的になり得るかがわかる。$\mathbf{K}\mathbf{x} = \mathbf{b}$ において $\mathbf{K} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1.0001 \end{smallmatrix}\right)$、$\mathbf{b} = (2, 2.0001)$ としよう。厳密解は $\mathbf{x} = (1, 1)$ である。ここで b_2 をほんの 10^{-4} だけ動かして 2.0002 にすると、解は一気に $(0, 2)$ まで飛び跳ねる。データにおける 10^5 分の 3 程度の摂動が、答えをまったく別物に変えてしまったのである。その理由は $\mathbf{K}$ の特異値（およそ 2.00005 と 5×10^{-5}）を見ればわかる。

行列 $\mathbf{A}$（ランク r）の**条件数**とは、最大の特異値と最小の非ゼロ特異値の比である。

$$\kappa(\mathbf{A}) = \frac{\sigma_1}{\sigma_r}. \tag{33.58}$$

幾何学的には、$\kappa(\mathbf{A})$ は $\mathbf{A}$ が異なる方向をどれほど不均一に伸縮するかを測っている。$\mathbf{K}$ では約 4×10^4 に達するのに対し、せん断行列ではわずか $\varphi^2 \approx 2.618$ である。$\mathbf{A}$ が対称正定値（positive definite）行列のときは特異値が固有値と一致するため $\kappa = \lambda_{\max}/\lambda_{\min}$ となり、33.3.2.3 節で見た二次関数のボウルに対する条件数が復元される。次の命題は、連立一次方程式を解く際の相対誤差に対する厳密な最悪増幅率が $\kappa(\mathbf{A})$ であることを示している。

命題（誤差増幅）。 **$\mathbf{A}$ を正方かつ可逆とし、$\mathbf{A}\mathbf{x} = \mathbf{b}$（$\mathbf{b} \neq \mathbf{0}$）とする。摂動 $\delta\mathbf{b}$ によって解が $\mathbf{x} + \delta\mathbf{x}$ に変化したとき、次式が成り立つ。**

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A}) \, \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}, \tag{33.59}$$

さらに、任意の $\mathbf{A}$ に対して等号を達成する $\mathbf{b}$ と $\delta\mathbf{b}$ が存在する。

証明。 $\mathbf{A}\,\delta\mathbf{x} = \delta\mathbf{b}$ より $\delta\mathbf{x} = \mathbf{A}^{-1}\delta\mathbf{b}$ である。$\mathbf{A}^{-1} = \mathbf{V}\Sigma^{-1}\mathbf{U}^{\top}$ の最大特異値は $1/\sigma_n$ であるから、Equation 33.48 より $\|\delta\mathbf{x}\| \le \sigma_n^{-1}\|\delta\mathbf{b}\|$ が成り立つ。同様に、$\mathbf{b} = \mathbf{A}\mathbf{x}$ より $\|\mathbf{b}\| \le \sigma_1\|\mathbf{x}\|$ が得られる。これら 2 つの不等式を組み合わせると次式を得る。

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{\sigma_1}{\sigma_n}\,\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} = \kappa(\mathbf{A})\,\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

両方の評価で同時に等号が成り立つのは、$\mathbf{b} = \mathbf{u}_1$（単位長さの右辺に対して $\mathbf{x} = \sigma_1^{-1}\mathbf{v}_1$ を最小の長さにするとき）かつ $\delta\mathbf{b} = \varepsilon\mathbf{u}_n$（$\delta\mathbf{x} = \varepsilon\sigma_n^{-1}\mathbf{v}_n$ を大きさ ε の摂動に対して最大の長さにするとき）である。$\blacksquare$

したがって、データの相対誤差は高々 κ 倍に増幅される。$\kappa = 10^{4}$ の場合、$\mathbf{b}$ の相対誤差 10^{-8} が $\mathbf{x}$ では 10^{-4} まで膨らみ得るため、有効精度が 4 桁失われる。先ほどの 2×2 の例では、$\mathbf{b}$ の相対変化量は $10^{-4}/\|\mathbf{b}\| \approx 3.5\times 10^{-5}$ であるため、上界は $\mathbf{x}$ の相対変化として 1.41 まで許容し、実際の変化量は 1.0 となっている。これは摂動 $(0, 10^{-4})$ が長さの $1/\sqrt{2}$ を最も増幅される方向 $\mathbf{u}_2 \approx (1,-1)/\sqrt{2}$ に持つ一方、$\mathbf{b}$ がほぼ真っ直ぐ $\mathbf{u}_1$ の方向を向いているためである。

理想的な極限にある直交行列では、すべての特異値が 1 に等しいため $\kappa = 1$ となり、回転と鏡映は誤差を決して増幅しない。直交因子 $\mathbf{U}$ と $\mathbf{V}$ を用いた計算が数値的に安全である理由も、直交初期化された重み行列があらゆる入力ベクトルのノルムを保つ理由もここにある。その対極として、グラム行列を作ると条件数は二乗される。

$$\kappa(\mathbf{A}^{\top}\mathbf{A}) = \kappa(\mathbf{A})^{2},$$

これは Equation 33.49 がそのまま $\mathbf{A}^{\top}\mathbf{A}$ の SVD になっており、その特異値が σ_i^2 となるからである（任意の形状・任意のランクの行列 $\mathbf{A}$ で成り立つ）。せん断行列では、κ は $\varphi^2 \approx 2.62$ から $\varphi^4 \approx 6.85$ へと跳ね上がる。以下のセルでは、2×2 の例を検証するとともに、先ほどの直線当てはめ行列について $\kappa(\mathbf{A})$ と $\kappa(\mathbf{A}^{\top}\mathbf{A})$ を比較する。

```
K = np.array([[1., 1.], [1., 1.0001]])
b = np.array([2., 2.0001])
x = np.linalg.solve(K, b)
x2 = np.linalg.solve(K, b + np.array([0., 1e-4]))
print('x =', x.round(4), '  after changing b_2 by 1e-4:', x2.round(4) + 0.)
kappa = np.linalg.cond(K)
print(f'kappa(K) = {kappa:.0f};  bound {kappa * 1e-4 / np.linalg.norm(b):.2f};  '
      f'realized {np.linalg.norm(x2 - x) / np.linalg.norm(x):.2f}')
A = np.array([[1., 1.], [1., 2.], [1., 3.], [1., 4.]])
print(f'kappa(A) = {np.linalg.cond(A):.3f};  kappa(A^T A) = {np.linalg.cond(A.T @ A):.2f};  '
      f'kappa(A)^2 = {np.linalg.cond(A)**2:.2f}')
```

```
x = [1. 1.]   after changing b_2 by 1e-4: [0. 2.]
kappa(K) = 40002;  bound 1.41;  realized 1.00
kappa(A) = 7.469;  kappa(A^T A) = 55.78;  kappa(A)^2 = 55.78
```

出力から手計算の結果が確認できる。解は $(1,1)$ から $(0,2)$ へ飛び、$\kappa(\mathbf{K}) \approx 40{,}002$ である。控えめなサイズの 4×2 直線当てはめ行列でさえ $\kappa = 7.47$ であり、正規方程式を作ると条件数はすでに $\kappa^2 = 55.8$ まで膨れ上がってしまう。

条件性とボウルの形状。 大きな条件数は、最適化においても細長く伸びた等位集合として直接現れる。列フルランクの最小二乗目的関数 $\tfrac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2$ に対し、ヘッセ行列（Hessian）は $\mathbf{H} = \mathbf{A}^{\top}\mathbf{A}$

であり、その等位集合は半軸の長さが $1/\sigma_i$ に比例する $\mathbf{v}_i$ 方向の楕円体となる。したがって等高線の軸比は $\kappa(\mathbf{A})$ となる一方、ヘッセ行列自身の条件数は $\kappa(\mathbf{A})^2$ となる。このような細長い谷の上では、勾配降下法（gradient descent）は急峻な壁の間をジグザグに往復して谷底に沿う進行が遅くなり（Figure 33.31）、その収束率（convergence rate）は 35.1 節で導出するとおり $\kappa(\mathbf{H})$ によって決まる。

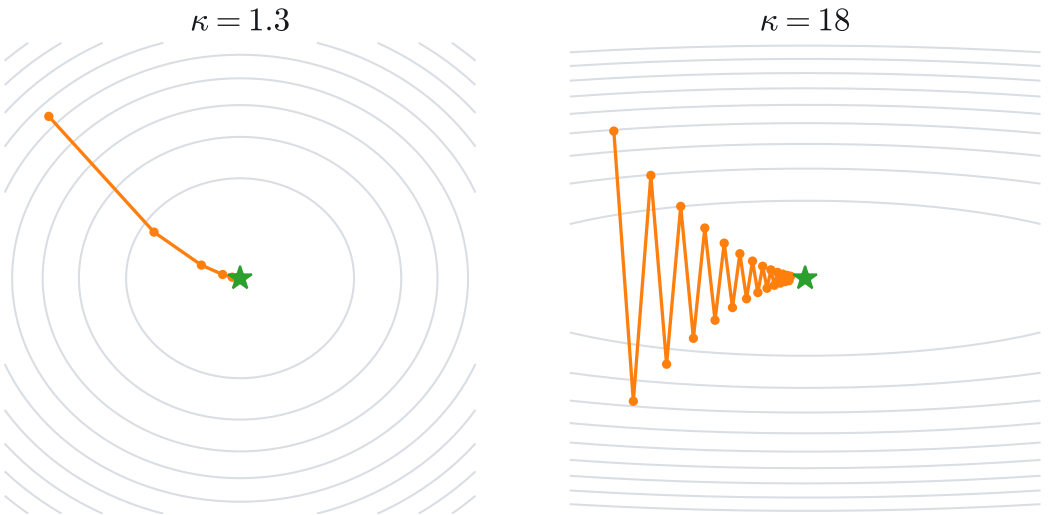


Figure 33.31: 良条件な二次関数のボウル（$\kappa(\mathbf{H}) = 1.3$、ほぼ円形の等高線、ほぼ直線的な経路）と悪条件なボウル（$\kappa(\mathbf{H}) = 18$、細長い谷を横切るジグザグの軌跡）における勾配降下法。パネルのタイトルはヘッセ行列の条件数 $\kappa(\mathbf{H})$ を示し、等高線はその平方根の倍率（第 2 パネルでは約 4.2 倍）だけ細長く伸びている。

33.5.4 深層学習における SVD

現代の大規模モデルの訓練で繰り返し現れる 3 つの問いは、重み行列の特異値へと直接帰着する。すなわち、ファインチューニング用アダプタにどれだけのパラメータが必要か、重み空間のあらゆる方向を対等に扱う更新をどう構成するか、そして単一の層が入力を最大どれだけ引き伸ばし得るか、という 3 つである。

33.5.4.1 低ランク更新

大規模な事前学習済みモデルの全パラメータをファインチューニング中に更新するのは、メモリと計算量の両面でコストが大きい。LoRA (Hu et al. 2021) はこれを回避するため、事前学習済みの各重み行列 $\mathbf{W} \in \mathbb{R}^{m \times n}$ を固定し、ランク k の補正項 $\Delta\mathbf{W} = \mathbf{B}\mathbf{A}$ のみを学習する。ここで $\mathbf{B} \in \mathbb{R}^{m \times k}$、$\mathbf{A} \in \mathbb{R}^{k \times n}$ である（ここでの $\mathbf{A}$ は原論文の記法に従い $k \times n$ の低次元射影を表しており、前節までの一般行列ではない）。このアダプタで訓練可能なパラメータ数は $k(m+n)$ 個となり、元の mn 個から大幅に減る。4096×4096 の層で $k = 8$ とすれば、パラメータ数は $16{,}777{,}216$ 個から $8 \cdot 8192 = 65{,}536$ 個、すなわちわずか 0.39% へと減少する。Figure 33.32 に示すように、固定された重みと低ランクバイパスは同じ入力に対して並列に作用し、それぞれの出力を足し合わせる。

ここで Eckart–Young の定理は何を教えてくれ、何を教えてくれ**ない**のだろうか。もしフルファインチューニングによって特異値 $\sigma_1 \geq \sigma_2 \geq \cdots$ を持つ更新が得られるとすれば、いかなるラン

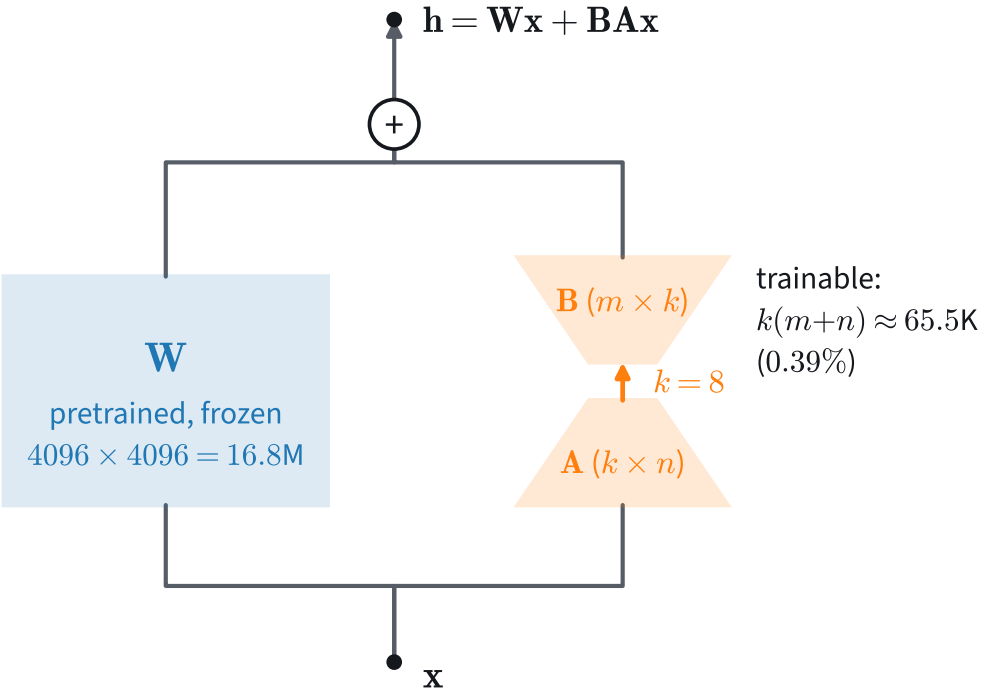


Figure 33.32: 1 層分の LoRA アーキテクチャ。事前学習済みの $\mathbf{W}$（全 $mn = 16.8\text{M}$ 個のパラメータ、$m = n = 4096$ の場合）は固定したままにし、並列バイパスがまず入力を $k = 8$ 次元へ $\mathbf{A}$ で圧縮してから $\mathbf{B}$ で元の次元へ戻すため、訓練されるのは $k(m + n) \approx 65.5\text{K}$ 個のパラメータ（約 0.39%）のみである。2 つの経路は $\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}$ と加算される。訓練後は $\mathbf{B}\mathbf{A}$ を $\mathbf{W}$ に統合できるため、デプロイ時の追加コストは一切かからない。

ク k アダプタもその特定の更新に対してスペクトルノルムで σ_{k+1} 未満、フロベニウスノルムで $(\sum_{i>k}\sigma_i^2)^{1/2}$ 未満には近づけないことが Eckart–Young から保証される。しかし、勾配降下法が実際にどのアダプタを学習するかまでは規定し**ない**。LoRA が成功するのは、たとえフルファインチューニングの更新に対する最良ランク k 近似でなくとも、**何らかの**低ランク更新が低い損失を達成するときである。多くの下流タスク（downstream task）においてそのような低ランク解が存在するという事実こそ、LoRA の背後にある経験的発見である。演習問題 2 では、ランク k アダプタが $\mathbf{W}$ の特異値を最大どれだけ動かし得るかを評価する。

33.5.4.2 直交化された更新

Muon オプティマイザ (Jordan et al. 2024) は、生のモメンタム行列 $\mathbf{M}$ の方向にそのまま重み行列を更新するのではなく、$\mathbf{M}$ の**極因子**（polar factor）、すなわち $\mathbf{M}$ の特異ベクトルを保ったまま非ゼロ特異値をすべて 1 に置き換えた行列の方向に更新する。オプティマイザ全体の動機と詳細は 9.6 節で扱うこととし、ここではそれが依拠する 2 つの線形代数の道具、すなわち極因子とは何か、そして行列積だけを使ってそれをどう計算するかを与える。

極分解。正方行列 $\mathbf{A}$ に対し、SVD の中に $\mathbf{V}^\top\mathbf{V}=\mathbf{I}$ を挿入すると、直交因子と対称な伸縮の積へと分解（極分解、polar decomposition）される。

$$\mathbf{A}=\mathbf{U}\Sigma\mathbf{V}^\top=(\mathbf{U}\mathbf{V}^\top)\,(\mathbf{V}\Sigma\mathbf{V}^\top)=\mathbf{Q}\,\mathbf{P}. \tag{33.60}$$

ここで $\mathbf{Q}=\mathbf{U}\mathbf{V}^\top$ は（直交行列の積として）直交行列であり、$\mathbf{P}=\mathbf{V}\Sigma\mathbf{V}^\top$ は固有値 σ_i を持つ対称半正定値行列である。このように、あらゆる正方行列は対称な伸縮に直交変換を続けたものへと分解される。これは複素数を極形式 $z=e^{i\theta}|z|$ と書くことの行列版にあたる。なお、$\mathbf{Q}$ は純粋な回転とは限らない。$\mathbf{A}$ が可逆のとき $\det\mathbf{Q}=\operatorname{sign}\det\mathbf{A}$ となるため、$\mathbf{Q}$ は $\det\mathbf{A}<0$ のとき常に鏡映を含む。せん断行列 $\mathbf{T}$ に対しては、Equation 33.47 より $\mathbf{Q}=\mathbf{R}_{31.7^\circ}\mathbf{R}_{58.3^\circ}^\top=\mathbf{R}_{-26.6^\circ}=\frac{1}{\sqrt{5}}\left(\begin{smallmatrix}2&1\\-1&2\end{smallmatrix}\right)$（$\tan 26.6^\circ=\frac{1}{2}$ より）および $\mathbf{P}=\frac{1}{\sqrt{5}}\left(\begin{smallmatrix}2&1\\1&3\end{smallmatrix}\right)$ が得られる。積を計算すると、確かに $\mathbf{QP}=\frac{1}{5}\left(\begin{smallmatrix}5&5\\0&5\end{smallmatrix}\right)=\mathbf{T}$ および $\mathbf{P}^2=\left(\begin{smallmatrix}1&1\\1&2\end{smallmatrix}\right)=\mathbf{T}^\top\mathbf{T}$ が成り立っている。

長方形行列 $\mathbf{M}$ に対しても、その薄い SVD を $\mathbf{M}=\mathbf{U}\Sigma\mathbf{V}^\top$ とすれば、極因子はやはり $\mathbf{U}\mathbf{V}^\top$ である。これは $\mathbf{M}$ と同じ特異ベクトルを持ち、すべての特異値が 1 に等しいため、その列（$m\ge n$ のとき）または行（$m\le n$ のとき）が正規直交になる。$\mathbf{M}$ がランク落ちしている場合（$r<\min(m,n)$）は極因子は一意ではないが、コンパクト SVD から得られる $\mathbf{U}_r\mathbf{V}_r^\top$ が自然な標準的選択となる。演習問題 6 では、正方行列において $\mathbf{U}\mathbf{V}^\top$ がフロベニウスノルムの意味で $\mathbf{A}$ に最も近い直交行列でもあることを示す。

行列の奇多項式。訓練ステップごとにすべての重み行列の完全 SVD を計算するのは、遅すぎて現実的ではない。幸いなことに、行列積（matrix multiplication）だけを使えば、特異ベクトルには一切触れずに特異値だけを変換できる。

命題（奇多項式は特異値に作用する）。$\mathbf{X}=\mathbf{U}\Sigma\mathbf{V}^\top$ を薄い SVD とし、$p(\sigma)=\sum_j c_j\sigma^{2j+1}$ を奇多項式とする。このとき次式が成り立つ。

$$p(\mathbf{X}):=\sum_j c_j\,\mathbf{X}(\mathbf{X}^\top\mathbf{X})^j=\mathbf{U}\,p(\Sigma)\,\mathbf{V}^\top. \tag{33.61}$$

証明。$\mathbf{X}^\top\mathbf{X}=\mathbf{V}\Sigma^2\mathbf{V}^\top$ であるから、べき乗をとると $(\mathbf{X}^\top\mathbf{X})^j=\mathbf{V}\Sigma^{2j}\mathbf{V}^\top$ となり、左から $\mathbf{X}=\mathbf{U}\Sigma\mathbf{V}^\top$ を掛けると $\mathbf{X}(\mathbf{X}^\top\mathbf{X})^j=\mathbf{U}\Sigma^{2j+1}\mathbf{V}^\top$ が得られる。これを j について足し合わせれば証明が完了する。■

したがって、任意の行列奇多項式は $\mathbf{U}$ と $\mathbf{V}$ をそのまま保ちながら、各特異値に対して独立にスカラー関数 p を適用する。極因子 $\mathbf{U}\mathbf{V}^\top$ に到達するには、繰り返し適用することですべての正の特異値を 1 へと向かわせる奇多項式 p を選べばよい。

Newton–Schulz。古典的な **Newton–Schulz 反復** $\mathbf{X} \leftarrow \frac{1}{2}(3\mathbf{X} - \mathbf{X}\mathbf{X}^\top\mathbf{X})$ は、すべての特異値に 3 次多項式 $p(\sigma) = \frac{1}{2}(3\sigma - \sigma^3)$ を適用する。その不動点（fixed point）は 0 と ± 1 である。区間 $(0, 1]$ 上では、p は 1 を行き過ぎることなくすべての値を上向きに押し上げる。なぜなら $p(\sigma) - \sigma = \frac{1}{2}\sigma(1 - \sigma^2) \geq 0$ であり、かつ p は最大値 $p(1) = 1$ をとるからである。実際、p は区間 $(0, \sqrt{3})$ 全体を $(0, 1]$ の中へ写すため、1 のこの**直接吸引領域**（immediate basin）に含まれるすべての初期値は 1 へと収束する一方、それより大きい初期値では不規則な挙動を示し得る（演習問題 5）。初期行列を $\mathbf{X}_0 = \mathbf{M}/\|\mathbf{M}\|_F$ とスケーリングすれば、すべての特異値が確実に $(0, 1]$ の内部から出発する。$\|\mathbf{M}\|_F \geq \sigma_1(\mathbf{M})$ だからである。またゼロ特異値はゼロのままとどまるため（$p(0) = 0$ より）、ランク落ちした $\mathbf{M}$ も綺麗に $\mathbf{U}_r\mathbf{V}_r^\top$ へと収束する。

これには何回の反復が必要だろうか。0 の近傍では $p(\sigma) \approx \frac{3}{2}\sigma$ であるから、小さな特異値 s は 1 ステップごとに $3/2$ 倍に成長し、オーダー 1 に達するまでにおよそ $\log_{3/2}(1/s)$ ステップを要する。1 の近傍では展開すると $p(1 - e) = 1 - \frac{3}{2}e^2 + \frac{1}{2}e^3$ となり、誤差が毎ステップおよそ二乗されるため、オーダー 1 から 10^{-6} まで縮むのに追加で約 4 ステップしかかからない。この 2 つの段階をつなぎ合わせると、およそ $\log_{3/2}(1/s_{\min}) + 4$ ステップと予測され、これは最小の正規化特異値 $s_{\min} = \sigma_{\min}(\mathbf{M})/\|\mathbf{M}\|_F$ によって決まる。モメンタムに見立てたランダムな 6×4 行列でこの大まかな見積もりを検証してみよう。

```
M = np.random.standard_normal((6, 4))                # a stand-in momentum matrix
X = M / np.linalg.norm(M)                           # Frobenius normalization: sigma in (0, 1]
s_min = np.linalg.svd(X, compute_uv=False)[-1]
for k in range(1, 41):
    X = 1.5 * X - 0.5 * X @ X.T @ X                  # p(sigma) = (3 sigma - sigma^3) / 2
    if np.abs(np.linalg.svd(X, compute_uv=False) - 1).max() < 1e-6:
        break
U, s, Vt = np.linalg.svd(M, full_matrices=False)
print(f'smallest scaled sigma {s_min:.3f};  predicted steps '
      f'{np.log(1 / s_min) / np.log(1.5) + 4:.1f}')
print(f'steps until max|sigma_i - 1| < 1e-6: {k};  distance to U V^T: '
      f'{np.linalg.norm(X - U @ Vt):.1e}')
```

```
smallest scaled sigma 0.190;  predicted steps 8.1
steps until max|sigma_i - 1| < 1e-6: 8;  distance to U V^T: 9.6e-09
```

実際のステップ数は予測と 1 反復以内の差で一致し、反復結果は完全 SVD から求めた厳密な極因子から 10^{-5} 以内に収まっている。$s_{\min}$ は乱数抽出ごとに変動するためステップ数も変動し、ほぼランク落ちに近い $\mathbf{M}$ ではより多くの反復を要する。実用上の Muon では、3 次多項式の代わりに調整された 5 次多項式を用い、1 への厳密な収束と引き換えに高速化を図っている（9.6.2 節）。

33.5.4.3 スペクトルの計測と制御

3 つ目の問いは線形層が入力を最大どれだけ引き伸ばし得るかというものであり、最大特異値がその答えを直接与える。より広く見れば、特異値スペクトル全体から行列が本当に低ランクに近いかどうかがわかる。そのため現代の訓練レシピ（training recipe）では、完全 SVD のコストを払うことなく特異値を抑え、調べ、近似する。

スペクトル正規化。@eq-mdl-sigma1-variational より、線形層 $\mathbf{x} \mapsto \mathbf{W}\mathbf{x}$ は $\|\mathbf{W}\mathbf{x} - \mathbf{W}\mathbf{x}'\| \leq$

$\sigma_1\|\mathbf{x}-\mathbf{x}'\|$ を満たし、$\mathbf{x}-\mathbf{x}'$ が $\mathbf{v}_1$ と同じ方向を向くときに等号が成り立つ。写像が入力の差を最大どれだけ増幅し得るかを抑える最小の定数を**リプシッツ定数**（Lipschitz constant）と呼ぶため、ここでのリプシッツ定数は単なる $\sigma_1 = \|\mathbf{W}\|_2$ である。$\mathbf{W}$ を σ_1 で割って層の伸縮率の上限を 1 に抑える手法がスペクトル正規化（spectral normalization）であり、GAN の識別器（discriminator）の訓練を安定化させる (Miyato et al. 2018)。その際、完全 SVD を計算する代わりに、σ_1 を $\mathbf{W}^\top\mathbf{W}$ に対するべき乗法（power iteration、33.4.1.2 節）で推定する。具体的には $\mathbf{u} \leftarrow \mathbf{W}\mathbf{v}$ と $\mathbf{v} \leftarrow \mathbf{W}^\top\mathbf{u}$ を半ステップごとに正規化しながら交互に繰り返す。この 1 往復で $\mathbf{W}^\top\mathbf{W}$ を 1 回掛けたことになり、$\|\mathbf{W}\mathbf{v}\|$ が σ_1 の推定値を与える。1 往復ごとに $\mathbf{v}$ は $\mathbf{v}_1$ へ引き寄せられ、誤差はおよそ $(\sigma_2/\sigma_1)^2$ 倍に縮小する。Miyato et al. (2018) は、前ステップのベクトルを初期値とするウォームスタートを行えば、訓練ステップあたり 1 往復で十分であることを実験的に見出した（なお畳み込み層では、カーネルを 2D 行列に変形することで畳み込みのスペクトルノルムの厳密値ではないものの近似値が得られる）。

スペクトルの計測。先ほどの LoRA における 0.39% という数値は、ランク 8 で十分だと仮定していた。ある行列が本当に低ランクに近いかどうかは特異値がどれほど速く減衰するかに完全に依存するため、打ち切る前に計測すべきである。以下のセルでは、2 つの 400×500 行列、すなわち構造をまったく持たないランダムなガウス行列と、ピクセル値の平均を差し引いた猫の写真を比較する。

```
G = np.random.standard_normal(img.shape)                        # no structure
for name, W in [('Gaussian', G), ('photograph', img - img.mean())]:
    s = np.linalg.svd(W, compute_uv=False)
    r95 = int(np.searchsorted(np.cumsum(s**2) / (s**2).sum(), 0.95)) + 1
    print(f'{name:10s}: sigma from {s[0]:.3g} to {s[-1]:.2g};  95% energy at rank {r95:3d};  '
          f'adapter {r95 * (m + n) / (m * n):.1%} of W')
```

```
Gaussian  : sigma from 42.6 to 2.5;  95% energy at rank 269;  adapter 121.0% of W
photograph: sigma from 2.65e+04 to 2.4;  95% energy at rank   8;  adapter 3.6% of W
```

対比は鮮明である。ガウス行列の特異値はマルチェンコ・パスツール区間 $[\sqrt{n}-\sqrt{m}, \sqrt{n}+\sqrt{m}] \approx [2.4, 42.4]$（33.5.2.1 節）全体に広がって急激な減衰を見せず、エネルギーの 95% を捉えるには 400 のうちおよそランク 270 が必要となるため、低ランクアダプタの方が元の行列よりかえって**多くの**パラメータを使ってしまう。対照的に、中心化した写真はランク 8 でエネルギーの 95% に達し、パラメータ数 4% 未満のアダプタに相当する。どちらも訓練されたニューラルネットワークの重みではないため、この比較はあくまで両極端を示す例にすぎない。LoRA において問題となるのはタスク固有の更新行列であり、そのスペクトルは仮定するのではなく実際に計測しなければならない。

ランダム化 SVD。行列が大きすぎて直接分解できない場合、**ランダム化 SVD**（randomized SVD）(Halko et al. 2011) は高精度なランク k 近似を、ごく少数の行列・ベクトル積（$\mathbf{A}$ との積）だけで計算する。その中核となる要素が**値域探索**（range finder）である。すなわち、$\mathbf{A}$ にランダムなガウス行列 Ω（k より少し多い列を持つ）を掛け、$\mathbf{A}\Omega$ の列を正規直交化して $\mathbf{Q}$ とし、はるかに小さい行列 $\mathbf{Q}^\top\mathbf{A}$ の SVD を計算する。特異値が減衰していれば、$\mathbf{A}$ の列のランダムな線形結合は常に上位の特異方向に支配されるため、$\mathbf{Q}$ は高い確率でそれらを捉える。演習問題 7 で、減衰が遅いスペクトルに対するべき乗法による改良も含めてこのアルゴリズムを実装し、33.6 節でランダム化スケッチングというより広い文脈の中に位置づける。

最後に、SVD の 3 つの限界を心に留めておこう。第 1 に、完全 SVD には $O(mn\min(m,n))$ 回の演算コストがかかる（$4096^3 \approx 7\times10^{10}$ 回にものぼり、単一の 4096×4096 重み行列でも膨大である）。Newton–Schulz 反復やランダム化値域探索のような実用的アルゴリズムが行列積のみに頼るのはこのためである。第 2 に、低ランク性はデータが持つ性質であって無条件に保証されるものではない。このことは直前のガウス行列が示したとおりである。第 3 に、SVD が捉

えるのは**線形**な構造のみである。平面上の円周上に並ぶ点群は、円が本質的に 1 次元であるにもかかわらずランク 2 のデータ行列をなす。曲がった関数や非線形な構造を理解するには導関数（derivative）が必要であり、次章ではそのテーマへと進む。

33.5.5 まとめ

形状にかかわらず任意の実行列 $\mathbf{A}$ には、入力の正規直交基底 $\{\mathbf{v}_i\}$ と出力の正規直交基底 $\{\mathbf{u}_i\}$ が存在し、その間でこの行列は非負の伸縮率の単純な列 $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$（$\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$）として作用する。この分解は、グラム行列 $\mathbf{A}^\top\mathbf{A}$ にスペクトル定理を適用することで $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^\top\mathbf{A})}$ として得られる。

順序付けられたこのたった 1 つの特異値のリストから、5 つの中心的な量が一度に導かれる。すなわち、ランク（正の σ_i の個数）、スペクトルノルムとフロベニウスノルム（$\|\mathbf{A}\|_2 = \sigma_1$ と $\|\mathbf{A}\|_F^2 = \sum_i \sigma_i^2$）、打ち切り SVD $\mathbf{A}_k$ が与える最良ランク k 近似（Eckart–Young の定理であり、中心化データに対しては主成分分析となる）、最小ノルム最小二乗解を返すムーア・ペンローズ擬似逆行列 $\mathbf{A}^+$、そして相対誤差の増幅率を抑える条件数 $\kappa = \sigma_1/\sigma_r$ である。深層学習においても、同じ特異値スペクトルがファインチューニング用の低ランクアダプタのサイズを決め、Muon が用いる極因子を定義し、スペクトル正規化が制御するリプシッツ定数を与える。

33.5.6 演習問題

1. ***ノルムと最大伸縮率***。本文ではスペクトル定理から Equation 33.48 を導いたが、最大伸縮率だけから逆に SVD を構成することもできる。
    1. $\sigma_1 = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|$ とおき、$\sigma_1 > 0$ と仮定して、スペクトル定理を使わずに SVD を構成せよ。(i) ある単位ベクトル $\mathbf{v}_1$ で最大値が達成されることを示し、$\mathbf{u}_1 = \mathbf{A}\mathbf{v}_1/\sigma_1$ とおけ。(ii) 単位ベクトル $\mathbf{w} \perp \mathbf{v}_1$ に対し、関数 $t \mapsto \|\mathbf{A}(\mathbf{v}_1 + t\mathbf{w})\|^2/(1+t^2)$ は $t=0$ で最大値をとるため、そこでの導関数は 0 になる。ここから $\mathbf{u}_1^\top \mathbf{A}\mathbf{w} = 0$、すなわち $\mathbf{A}$ が $\mathbf{v}_1^\perp$ を $\mathbf{u}_1^\perp$ の中へ写すことを導け。(iii) $\mathbf{A}$ の $\mathbf{v}_1^\perp$ への制限に対して帰納的に繰り返せ。単位球面のコンパクト性はどこで使われたか。
    2. $\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r}\,\|\mathbf{A}\|_2$（ただし $r = \operatorname{rank}\mathbf{A}$）を示せ。左右それぞれの不等号で等号を達成するのはどのような行列か。
    3. 比 $\|\mathbf{A}\|_F^2/\|\mathbf{A}\|_2^2$ は $\mathbf{A}$ の**安定ランク**と呼ばれ、(b) より 1 と $\operatorname{rank}\mathbf{A}$ の間にある。減衰スペクトル $\sigma_i = e^{-i/12}$（$i = 0, \ldots, 255$、幾何級数）に対して安定ランクを閉じた式で求め、このスペクトルのエネルギーの 95% を保持するにはランク 18 が必要であることを示せ。次に、400×500 のガウス行列（33.5.4.3 節）の安定ランクを $\|\mathbf{G}\|_F^2 \approx mn$ と $\sigma_1 \approx \sqrt{m} + \sqrt{n}$ から見積もり、エネルギーの 95% を保持するランク（約 270）と比較せよ。なぜ安定ランクは通常のランクよりはるかに小さくなり得るのか。
2. **Weyl 摂動限界**。更新 Δ、たとえばランク k の LoRA 更新 $\mathbf{B}\mathbf{A}$（原論文の記号）は、重み行列 $\mathbf{W}$ の特異値を変化させる。Weyl の不等式 Equation 33.54（33.5.2.1 節で証明）は、その変化幅の上限を与える。
    1. $j = 1$ を Equation 33.54 に代入し、不等式を両方向に適用することで、$|\sigma_i(\mathbf{W}+\Delta) - \sigma_i(\mathbf{W})| \leq \sigma_1(\Delta)$ がすべての i で成り立つことを導け。
    2. $\mathbf{W} = \operatorname{diag}(1, 2)$、$\Delta = 0.1\,\mathbf{1}\mathbf{1}^\top$ とする。$\mathbf{W}+\Delta$ の特異値を計算し、それぞれの変化量を $\sigma_1(\Delta)$ と比較せよ。
    3. 次に $\operatorname{rank}\Delta \leq k$ とする。33.5.2.1 節のフロベニウスノルムの証明では、Equation 33.54 を $j = k+1$ としてランクが高々 k の行列に適用した。同じ手順を Δ に適用して、交錯不等式 $\sigma_{i+k}(\mathbf{W}) \leq \sigma_i(\mathbf{W}+\Delta)$ および $\sigma_{i+k}(\mathbf{W}+\Delta) \leq \sigma_i(\mathbf{W})$ を導け。これらを用いて、$\mathbf{W}+\Delta$ と $\mathbf{W}$ のランクの差が高々 k であること、さらに高々 k

個の特異値（$\mathbf{W}+\Delta$ のもの）しか $\sigma_1(\mathbf{W})$ を超えないことを、$\sigma_1(\Delta)$ がどれほど大きくても成り立つ形で示せ。なぜ (a) の評価からはこれらの結論が得られないのか。

3. [code] **PCA における中心化と Fashion-MNIST**。$\mathbf{X}\in\mathbb{R}^{n\times d}$ の各行をデータ点とし、平均行ベクトルを $\bar{\mathbf{x}}$、共分散を $\mathbf{C}$ とする（33.5.2.2 節の主成分分析と同じ設定）。
   1. $\frac{1}{n}\mathbf{X}^\top\mathbf{X}=\mathbf{C}+\bar{\mathbf{x}}\bar{\mathbf{x}}^\top$ を示せ。
   2. 33.5.2.2 節と同様の点群（300 点、標準偏差 3 と 0.8、回転角 $\pi/5$ の軸に沿う分布）を生成し、すべての点を $c\,(1,-1)/\sqrt{2}$（$c\in\{0,1,3,10,30\}$）だけ平行移動させよ。各 c について、**中心化していない**データの最大右特異ベクトル、主軸 $(\cos\frac{\pi}{5},\sin\frac{\pi}{5})$ とのなす角、および比 $\sigma_1^2/\sum_i\sigma_i^2$ を計算せよ。
   3. (a) を用いて、最大特異ベクトルが主軸から平均の方向へと向きを変えるオフセットの大きさを $\mathbf{C}$ の固有値から予測せよ。
   4. 再び (a) を用いて比 $\sigma_1^2/\sum_i\sigma_i^2$ を大きな c に対して近似し、$c\to\infty$ における極限を求めよ。$c=10$ では、この比は中心化後の説明分散比に近い値となる。この一致はデータの性質を反映したものか、それとも選んだオフセットによるものか。
   5. `d2l.FashionMNIST` を用いて（33.1 節と同様に）Fashion-MNIST の T シャツとズボンのテスト画像を読み込み、長さ 784 のベクトルへ平坦化し、中心化した上で上位 2 本の右特異ベクトルを計算せよ。最初の 2 つの主成分が張る平面上に 2 クラスをプロットせよ。これら 2 方向は分散のどれだけの割合を占めるか。さらに $\mathbf{v}_1$ とクラス平均の差 $\bar{\mathbf{x}}_{\text{trouser}}-\bar{\mathbf{x}}_{\text{t-shirt}}$ のなす角を計算し、2 つの方向が近いものの厳密には一致しない理由を説明せよ。最後に、1 枚の T シャツ画像を最初の 5 個、20 個、50 個の主成分から再構成し、それぞれの相対誤差を報告せよ。

4. [code] **リッジなし極限と勾配降下法**。過剰パラメータ化された線形モデルは方程式より多くのパラメータを持つため、最小二乗法は無数の最小点を持つ。小問 (a) は任意の形状・ランクで成り立ち、(b) と (c) では $\mathbf{A}\in\mathbb{R}^{m\times n}$（$m<n$）を用いる。
   1. フィルタ形式 Equation 33.57 を用いて、$(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})^{-1}\mathbf{A}^\top\to\mathbf{A}^+$（$\lambda\to0^+$）が任意の形状・ランクの行列 $\mathbf{A}$ に対して成り立つことを示せ。なぜ $(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$ は $m<n$ のとき存在しないのか。
   2. $\frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2$ に対するステップサイズ $0<\eta<2/\sigma_1^2$ の勾配降下法は、$\mathbf{x}_{t+1}=\mathbf{x}_t-\eta\,\mathbf{A}^\top(\mathbf{A}\mathbf{x}_t-\mathbf{b})$ と更新する。$\mathbf{x}_t-\mathbf{x}_0$ が常に $\mathbf{A}$ の行空間内にとどまること、および $\mathbf{x}_t$ が $\mathbf{A}^+\mathbf{b}+(\mathbf{I}-\mathbf{A}^+\mathbf{A})\mathbf{x}_0$ へ収束することを示せ。初期化 $\mathbf{x}_0=\mathbf{0}$ はどの解を選び出すか。
   3. 20×50 のガウス行列 $\mathbf{A}$ とガウスベクトル $\mathbf{b}$ に対して (a) と (b) を検証せよ。$\mathbf{x}_0=\mathbf{0}$ とランダムな $\mathbf{x}_0$ の両方から勾配降下法を実行し、それぞれの収束先とそのノルムを $\mathbf{A}^+\mathbf{b}$ と比較せよ。

5. [code] **Newton–Schulz の吸引領域**。Newton–Schulz 反復 $\mathbf{X}\leftarrow\frac{1}{2}(3\mathbf{X}-\mathbf{X}\mathbf{X}^\top\mathbf{X})$（33.5.4.2 節）は $\mathbf{X}=\mathbf{M}/\|\mathbf{M}\|_F$ から出発する。
   1. $\mathbf{X}=\mathbf{U}\Sigma\mathbf{V}^\top$ を薄い SVD とする。Equation 33.61 を使わずに直接計算することで、1 ステップ後の結果が $\mathbf{U}\,p(\Sigma)\,\mathbf{V}^\top$（各対角成分に $p(\sigma)=\frac{1}{2}(3\sigma-\sigma^3)$ を適用したもの）になることを示せ。
   2. $(0,\sqrt{3})$ が 1 の直接吸引領域であることを示せ。すなわち、p がそれを $(0,1]$ の中へ写し、反復列 $p(p(\cdots p(\sigma_0)))$ がすべての $\sigma_0\in(0,\sqrt{3})$ に対して 1 へ収束することを示せ。また、$\sqrt{3}$ を超える初期値の中にも 1 に到達するもの（2.15 を試せ）と到達しないもの（2 を試せ）があることを示せ。さらに $p(1-e)$ を e のべき乗で展開せよ。1 までの距離が小さくなった後は、どれほどの速さで縮小するか。
   3. 6×4 のガウス行列 $\mathbf{M}$ に対し、$\|\mathbf{X}-\mathbf{U}\mathbf{V}^\top\|_F<10^{-6}$ になるまでのステップ数を、初期値 $\mathbf{M}/c$ で $c=\|\mathbf{M}\|_F$ とした場合と $c=\sigma_1(\mathbf{M})$ とした場合のそれぞれで数えよ。

両方のステップ数を $\log_{3/2}(1/s_{\min})+4$（$s_{\min}$ はスケーリング後の最小特異値）と比較せよ。1 ステップ節約できるにもかかわらず、Muon が $\|\mathbf{M}\|_F$ で割るのはなぜか（$\sigma_1(\mathbf{M})$ ではなく）。

4. $\mathbf{M}/c$ を初期値として、$c=\sigma_1(\mathbf{M})/2$ の場合と $c=\sigma_1(\mathbf{M})/2.5$ の場合で反復を開始せよ。それぞれ何に収束するか。(b) の多項式 p を用いて両方の結果を説明せよ。

6. [code] **極因子と Procrustes 問題**。●極分解 Equation 33.60 は正方行列を $\mathbf{A}=\mathbf{QP}$（$\mathbf{Q}$ は直交行列、$\mathbf{P}$ は対称半正定値行列）と表す。

1. $\mathbf{P}=(\mathbf{A}^\top\mathbf{A})^{1/2}$ が $\mathbf{A}$ によって一意に定まること、さらに $\mathbf{Q}$ も $\mathbf{A}$ によって（$\mathbf{A}$ が可逆のとき）一意に定まることを示せ。また $\operatorname{diag}(1,0)$ の異なる 2 つの極分解を挙げよ。
2. フロベニウスノルムにおいて $\mathbf{Q}=\mathbf{UV}^\top$ が $\mathbf{A}$ に最も近い直交行列であることを示せ。その際、問題を $\operatorname{tr}(\mathbf{R}^\top\mathbf{A})$ の直交行列 $\mathbf{R}$ に関する最大化へと帰着させよ。
3. 回転だけ異なる 2 つの埋め込み空間は、対応点の組 $\mathbf{X},\mathbf{Y}\in\mathbb{R}^{d\times N}$ から $\|\mathbf{RX}-\mathbf{Y}\|_F$ を直交行列 $\mathbf{R}$ 上で最小化することで位置合わせできる。(b) を用いて、最小点が $\mathbf{YX}^\top$ の極因子であることを示せ。$d=50$、$N=1000$ とし、ランダムな直交行列 $\mathbf{R}$（ガウス行列の QR 分解の $\mathbf{Q}$ 因子）、ガウス行列 $\mathbf{X}$、および $\mathbf{Y}=\mathbf{RX}+\mathbf{E}$（$\mathbf{E}$ の成分は独立同分布な $\mathcal{N}(0,s^2)$、$s\in\{0,0.1,1,3\}$）でこれを検証せよ。$\|\hat{\mathbf{R}}-\mathbf{R}\|_F/\sqrt{d}$ を、この推定量と制約なし最小二乗推定量 $\mathbf{YX}^{+}$ について比較せよ。

7. [code] **ランダム化値域探索**。●33.5.4.3 節のランダム化 SVD は、$\mathbf{A}$ にガウス行列 $\Omega\in\mathbb{R}^{n\times(k+p)}$ を掛け、$\mathbf{Y}=\mathbf{A}\Omega$ を正規直交化して $\mathbf{Q}$ とし、$\mathbf{Q}^\top\mathbf{A}$ の SVD をランク k で打ち切る。

1. これを実装せよ。$\mathbf{A}\in\mathbb{R}^{1000\times 500}$ を $\mathbf{U}\operatorname{diag}(\sigma_1,\dots,\sigma_{500})\mathbf{V}^\top$（$\mathbf{U}$ と $\mathbf{V}$ はランダムな列正規直交行列）として構成し、速い減衰 $\sigma_i=e^{-i/5}$ の場合と遅い減衰 $\sigma_i=1/i$ の場合の 2 通りを用意せよ。$k=20$、$p\in\{0,5,10\}$ に対し、ランク k の近似結果のスペクトルノルム誤差を σ_{k+1}（Equation 33.53 の最小誤差）で割り、10 回の Ω の抽出にわたって平均せよ。なぜ片方のスペクトルではオーバーサンプリングだけで十分なのに、もう片方では不十分なのか。
2. $\mathbf{Y}$ を $(\mathbf{AA}^\top)^q\mathbf{A}\Omega$（$q\in\{1,2\}$）に置き換えよ。各スペクトルで誤差比はどう変化するか。q の効果を $(\mathbf{AA}^\top)^q\mathbf{A}$ の特異値を用いて説明せよ。また、各変種には $\mathbf{A}$ または $\mathbf{A}^\top$ との積が何回必要か（計算コストを答えよ）。

8. [code] **最適ハード閾値**。●●閾値 $\frac{4}{\sqrt{3}}\sqrt{n}\,\sigma_{\text{noise}}$（Gavish and Donoho (2014)）は、単一の信号成分の振る舞いから導かれる。$\mathbf{A}=\theta\,\mathbf{uv}^\top+\mathbf{N}$ を $n\times n$ 行列とし、$\mathbf{u},\mathbf{v}$ を単位ベクトル、$\theta>0$ とし、$\mathbf{N}$ の各成分は独立同分布なガウス分布 $\mathcal{N}(0,\sigma_{\text{noise}}^2)$ に従うとする。$x=\theta/(\sqrt{n}\,\sigma_{\text{noise}})$ とおき、$(\sigma_1,\hat{\mathbf{u}},\hat{\mathbf{v}})$ を $\mathbf{A}$ の最大特異三つ組とする（ただし対の符号は $\mathbf{u}^\top\hat{\mathbf{u}}\ge 0$ となるように選ぶ）。$n\to\infty$ のとき、Benaych-Georges and Nadakuditi (2012) によれば、σ_1 はバルク上端 $2\sqrt{n}\,\sigma_{\text{noise}}$ へ収束して特異ベクトルは $\mathbf{u}$、$\mathbf{v}$ と直交するようになり（$x\le 1$ のとき）、また $x>1$ のときは次式が成り立つ。

$$\frac{\sigma_1}{\sqrt{n}\,\sigma_{\text{noise}}}\to x+\frac{1}{x},\qquad (\mathbf{u}^\top\hat{\mathbf{u}})^2\to 1-\frac{1}{x^2},\qquad (\mathbf{v}^\top\hat{\mathbf{v}})^2\to 1-\frac{1}{x^2}.$$

$x=1$ におけるこの相転移は、演習問題 5（33.3 節）における $\beta=\sqrt{c}$ での固有値の相転移の特異値版にあたる。

1. 信号 $\theta\,\mathbf{uv}^\top$ に対する 2 つの推定量として、ランク 1 打ち切り $\sigma_1\hat{\mathbf{u}}\hat{\mathbf{v}}^\top$ とゼロ行列を考える。それぞれの二乗フロベニウス誤差を θ、σ_1、$\mathbf{u}^\top\hat{\mathbf{u}}$、$\mathbf{v}^\top\hat{\mathbf{v}}$ を用いて表せ。
2. 上の極限を用いて、打ち切りの方が漸近誤差が小さくなるのはちょうど $x>\sqrt{3}$ のときであり、$x=\sqrt{3}$ における特異値が $\frac{4}{\sqrt{3}}\sqrt{n}\,\sigma_{\text{noise}}$ であることを示せ。ノイズの上端

を超える特異値を生じるにもかかわらず切り捨てるべき信号強度はどの範囲か。

3. $n = 500$、$\sigma_{\text{noise}} = 1/\sqrt{n}$ とし、$x \in \{0.5, 1.2, 1.5, \sqrt{3}, 2, 3\}$ のそれぞれで 10 回ずつ乱数抽出して (a) と (b) を検証せよ。σ_1、余弦の二乗、および両方の誤差をそれぞれの極限値と比較せよ。

33.6 関連文献

本節では、本章で展開した線形代数を数値解析、ランダム行列理論、および深層学習の応用へと結びつけ、続いて本章の中核的な内容を網羅する厳選された教科書と講義の一覧を示す。主な関心が深層学習にある場合は、まずランダム射影（random projection）、反復ソルバ、ランダム行列と訓練可能な深さ、およびスケッチングから読み始め、その後に低ランクアテンションとスペクトルノルムオプティマイザを扱う最後の 2 項目へ進むとよい。本書の他の箇所は本節の内容に一切依存しない。

33.6.1 幾何学と線形変換

33.6.1.1 他の内積：カーネル法

33.1.4.2 節の最近傍平均規則では、テスト画像を 1 回のドット積によって 2 つのクラス平均と比較して分類するため、決定境界は必然的に平坦な超平面になる。しかし、T シャツとシャツを見分ける場合のように、クラスが湾曲していたり複数のクラスタに分かれていたりすると、この単純な仕組みは破綻する。カーネル法（kernel method）は、まったく同じ幾何学的規則を保ったまま内積だけを変更する。すなわち、$\mathbf{x} \cdot \mathbf{x}_i$ を、別の（場合によっては無限次元の）特徴空間における真の内積に対応する類似度関数 $k(\mathbf{x}, \mathbf{x}_i)$ に置き換えるのである。するとクラス平均はその隠れた空間での平均となり、各訓練例が重み $k(\mathbf{x}, \mathbf{x}_i)$ で投票するようになり、結果として得られる境界は元の入力空間内で自在に曲がることができる。どの訓練点が実際に非ゼロの重みを必要とするかを問うと、そのままサポートベクターマシンへと至る。Schölkopf and Smola (2002) はまさにこの分類器から筆を起こし、そこからカーネル法の理論全体を築き上げている。

33.6.1.2 ランダム射影

33.1 節では、高次元空間 $\mathbb{R}^d$ におけるランダムな方向どうしが互いにほぼ直交することを見た。驚くべきことに、この幾何学的な性質を利用すると、データを一切見ることなくデータセットの次元を縮小できる。Johnson–Lindenstrauss の補題 (Johnson and Lindenstrauss 1984) によれば、任意の n 個の点からなる集合に対し、適切にスケーリングしたランダムガウス行列を掛けて k 次元（オーダーは $\varepsilon^{-2} \log n$）へ写すと、$\binom{n}{2}$ 個すべてのペア間距離が高い確率で $1 \pm \varepsilon$ 倍の範囲内に保たれる。主成分分析（33.5.2.2 節）とは異なり、ランダム射影はデータにまったく依存せず、目標次元 k は外囲次元（ambient dimension）d にすら依存しない。したがって、この手法が最も威力を発揮するのは、d が $\varepsilon^{-2} \log n$ よりはるかに大きいときである。Dasgupta and Gupta (2003) は明示的な定数を伴う驚くほど短い証明を与えており、一方で Vershynin (2018) は確率ベクトル（random vector）の集中に関するより広い理論の中にこの補題を位置づけている。

33.6.1.3 大規模な連立一次方程式の解法

33.2.2.4 節では、LU 分解によって連立一次方程式を解き、大規模な疎行列（sparse matrix）の逆行列は一般に密になることを確認した。病的な最悪ケースの行列が存在するにもかかわらず、部分ピボット選択付きガウスの消去法（Gaussian elimination with partial pivoting）が浮動小数点演算でこれほど高い信頼性をもって機能するのはなぜだろうか。Trefethen and Bau (1997) は最悪ケースと実用上の安定性の間にあるこの乖離を明快に説明しており、Higham (2002) は丸め誤差（rounding error）に関する決定版の解析を与えている。

大規模な問題では、疎行列ソルバは決して $\mathbf{A}^{-1}$ を作らず、密な LU 因子の構築すら避けることが多い。疎行列の直接法（direct method）は、消去の過程で新たに生じる非ゼロ要素ができるだけ少なくなるよう、未知数の順序を注意深く並べ替える。反復法（iterative method）はさらに一歩進め、べき乗法（33.4.1.2 節）とまったく同様に、行列・ベクトル積 $\mathbf{Ax}$ のみを通じて行列にアクセスする。$\mathbf{A}$ が対称正定値であるとき、Hestenes and Stiefel (1952) の共役勾配法（conjugate gradient method）は、k ステップ後の誤差を高々 $2\big((\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)\big)^{k}$ 倍（$\mathbf{A}$ が誘導するノルムのもとで）にまで縮小する。ここで、κ は 33.5.3.2 節で導入した条件数である。Golub and Van Loan (1996) は共役勾配法とその関連手法を詳細に扱っている。

33.6.1.4 縮約順序とテンソル分解

`einsum` 記法（33.2.1.4 節）は複数のテンソルを縮約した数学的結果を指定するが、ペアごとの縮約をどの順序で実行するかは定めない。この選択によって計算量は劇的に変わり得る。たとえば $n \times n$ 行列に対して、$(\mathbf{AB})\mathbf{x}$ の計算には約 n^3 回の乗算を要するが、$\mathbf{A}(\mathbf{Bx})$ ならばわずか $2n^2$ 回で済む。多くの因子が関わる場合、最も計算コストの低い順序を選ぶ問題は括弧の付け方の探索になり、`np.einsum(..., optimize=True)` や `opt_einsum` パッケージ (Smith and Gray 2018) がこれを自動化する。

低ランク近似（low-rank approximation）の考え方は行列を超えて高階テンソルへも広がるが、そこには厳しい落とし穴がある。3 元テンソルは、ランク 1 の外積 $\mathbf{a}_i \otimes \mathbf{b}_i \otimes \mathbf{c}_i$ の和（CP 分解）、小さなコアテンソルに各モードに沿って因子行列を掛けた形（Tucker 分解）、あるいは 3 元コアの連鎖（Oseledets (2011) のテンソルトレイン）として表せる。しかし残念ながら、テンソル一般に対しては Eckart–Young の定理ほど明快な結果は成り立たない。CP ランクの決定は NP 困難であり、与えられた CP ランクでの最良近似が存在しないことさえある。CP 分解と Tucker 分解およびそれらのアルゴリズムについては、Kolda and Bader (2009) のサーベイが標準的な案内書である。

33.6.2 固有値とスペクトル

33.6.2.1 実用上の固有値計算

5 次以上の一般の多項式にはべき根による解の公式が存在しないため、実用的な固有値ソルバはすべて反復計算に頼らざるを得ない。さらに悪いことに、多項式の根はその係数に含まれる微小な丸め誤差に対して極端に敏感になり得るため、実用ライブラリが 33.3.1.2 節の特性多項式（characteristic polynomial）を陽に構成することもない。その代わり、`eig` はまず直交相似変換を適用し、$O(n^3)$ 回の演算で行列を Hessenberg 形（第 1 劣対角線より下がすべて 0 の形）へと簡約する。続いてシフト付き QR 法（shifted QR algorithm）を実行する。すなわち、$\mathbf{H} - \mu\mathbf{I} = \mathbf{QR}$ と分解し、$\mathbf{RQ} + \mu\mathbf{I}$ を作り、これを繰り返す。各ステップは相似変換であるため固有値は一切変化しない一方、劣対角成分は 0 に向かって縮小し、対角線上にスペクトルが現れる（実行列の複素共役ペアは 2×2 の対角ブロックに落ち着く）。対称行列では Hessenberg 形が三重対角になるため、`eigh` はこの構造を利用して大幅な高速化を実現している。Trefethen and Bau (1997) は部分空間上のべき乗法から QR 法を明快に導出しており、Golub and Van Loan (1996) は実用実装における工学的な詳細を網羅している。

33.6.2.2 擬スペクトルと Kreiss の行列定理

33.4.2.4 節では、非正規行列においては固有値が単位円盤の内部にあっても $\|\mathbf{A}^k\|_2$ の大きな過渡的増大（transient growth）は防げないことを見た。Kreiss の行列定理は、擬スペクトル（pseudospectrum）から得られる視覚的な直感を定量的な評価式へと変える。Kreiss 定数を $K(\mathbf{A}) = \sup_{|z|>1} (|z|-1)\|(z\mathbf{I}-\mathbf{A})^{-1}\|_2$ と定義すると、任意の $n \times n$ 行列に対して累乗ノルムの最悪ケースは次の不等式で上下から挟まれる。すなわち、$K(\mathbf{A}) \le \sup_k \|\mathbf{A}^k\|_2 \le e\,n\,K(\mathbf{A})$

である。下界は 33.4 節の演習問題で証明できる。一方、上界からは、擬スペクトルが単位円盤に密着している行列では非有界な過渡的増大が起こり得ないことが保証される。Trefethen and Embree (2005) はこの理論を網羅的に展開しており、線形化した固有値がすべて厳密に安定であるにもかかわらず乱流へ遷移する流体流れへの古典的な応用も詳しく扱っている。

33.6.2.3 混合時間とスペクトルギャップ

33.3 節の 2 ページからなるマルコフ連鎖では、第 2 固有値が $1/4$ であるため、定常分布までの距離がクリックごとに 4 分の 1 に縮小した。では、一般のマルコフ連鎖が混合するまでには何ステップ必要だろうか。可逆な連鎖に対し、$\lambda_\star$ を 1 以外のすべての固有値における絶対値の最大値とする。定常分布との全変動距離（total variation distance）が ε 以内に入るまでに必要なステップ数は、上下ともに $1/(1-\lambda_\star)$ の定数倍で抑えられる（定数は ε と最小の定常確率に依存する）。少なくとも確率 $1/2$ で現在の状態にとどまる怠惰な連鎖（lazy chain）では、すべての固有値が非負になるため、$\lambda_\star$ は単に第 2 固有値 λ_2 となる。さらに Cheeger の不等式は、スペクトルギャップ $1-\lambda_2$ を連鎖の**コンダクタンス**（部分集合の定常質量に対する、そこから流出する確率フローの最小比率）によって上下両側から評価する。言い換えれば、グラフのボトルネックから収束率を直接読み取ることができる。Levin and Peres (2017) は、33.4.1.3 節が Horn and Johnson (2012) から引用している収束の結果とあわせて、この理論をきわめて明快に解説している。

33.6.2.4 ランダム行列と訓練可能な深さ

33.4.2.5 節では、個々の層のヤコビ行列のスペクトル半径だけでは、ヤコビ行列の長い積の振る舞いを制御できないことを確認した。では、非常に深いネットワークを初期化からそのまま訓練できるかどうかは何によって決まるのだろうか。Saxe et al. (2014) は深層線形ネットワークの学習ダイナミクス（learning dynamics）を厳密に解き、直交する初期重みを用いれば訓練時間が深さに依存しなくなる一方、スケーリングしたガウス重みでは深さが増すにつれて学習が遅くなることを示した。この解析を非線形ネットワークへ拡張した Pennington et al. (2017) は、自由確率論（free probability）を用いて入出力ヤコビ行列の特異値分布を完全に求めている。その結果、すべての特異値が 1 の近傍に密に集まる**動的等長性**（dynamical isometry）と呼ばれるレジームは、直交重みと tanh 活性化関数の組み合わせでは達成できるものの ReLU では達成できず、このレジームで初期化したネットワークは層が深くてもはるかに高速に訓練できることが示された。

33.6.2.5 訓練済み重みのスペクトル

33.3.2.4 節のマルチェンコ・パスツール則（Marchenko–Pastur law）は、純粋な独立ノイズからなる行列の特異値スペクトルを記述するものであり、初期化したばかりのネットワークの重み行列はまさにこのような姿をしている。Martin and Mahoney (2021) は、訓練済みニューラルネットワークの重みのスペクトルをこのランダム行列のベースラインと比較し、訓練によって鋭い外れ値と重いべき乗則の裾の両方が生じることを観察した。同研究はこの裾指数（tail exponent）を暗黙的な自己正則化（implicit self-regularization）の兆候と解釈している。マルチェンコ・パスツール則からのこうした乖離は多くのアーキテクチャで一貫して見られるものの、テストデータで評価せずに裾指数を用いてモデルの品質を順位付けできるかについては、証明された定理ではなく経験的な相関にとどまっている。

33.6.3 打ち切り SVD を超える低ランク構造

33.6.3.1 行列補完と核ノルム

映画の評価値行列では全成分のうち数パーセントしか観測されていないことがある一方、Eckart–Young の定理（33.5.2.1 節）を用いるには最初から行列全体がわかっている必要がある。行列補完（matrix completion）はこの隔たりを埋めるため、観測されたすべての成分に一致する

という制約のもとで**核ノルム**（nuclear norm）$\|\mathbf{Z}\|_* = \sum_i \sigma_i$ を最小化する。これは、ベクトルのスパース性に対して ℓ_1 ノルムが凸な代理となるのと同様に、行列のランクに対する凸な代理として働く。Candès and Recht (2009) は、背後にある低ランク行列の特異ベクトルが少数のスパイクに集中せず座標全体に広がっている限り、この凸最適化問題によって n^2 個よりはるかに少ないランダムな観測成分から元の行列が高い確率で厳密に復元できることを証明した。Soft-Impute (Mazumder et al. 2010) は、欠損部分の補完ステップと特異値のソフト閾値処理（soft-thresholding）を交互に繰り返すことで、この問題のペナルティ付き版を解く。

33.6.3.2 CUR 分解とスケッチング

打ち切り SVD は近似誤差の面では最適であるものの、その特異ベクトルはすべての行と列を混ぜ合わせてしまうため解釈しにくく、さらに $m \times n$ 行列の完全な SVD を計算するには $O(mn \min(m,n))$ 回の演算コストがかかる。これに対し CUR 分解は、データ行列の実際の列 **C** と実際の行 **R** を取り出し、それらを小さなコア行列 **U** で結ぶことで近似を構成する。33.5.2.1 節では、ノルムが最大の列を単純に選ぶ素朴な戦略が打ち切り SVD に大きく劣ることを見た。Mahoney and Drineas (2009) はその改善策を示しており、上位の特異ベクトルへの射影の長さの二乗である**レバレッジスコア**に応じて列と行をサンプリングすれば、高い確率で Eckart–Young の誤差の小さな定数倍以内に収まる。

より広く見れば、33.5.4.3 節のランダム化値域探索（randomized range finder）(Halko et al. 2011) は**スケッチング**の 1 つの例である。すなわち、大きな行列に細長いランダム行列を掛け、はるかに小さなスケッチ上で問題を解く手法である。Woodruff (2014) は、これと同じ発想によって最小二乗法や回帰がどのように高速化され、元のデータセットよりはるかに少ない行数しか持たないスケッチから最適解の $1+\varepsilon$ 倍以内の解を返せるかを概説している。

33.6.3.3 アダプタの派生手法

LoRA（33.5.4.1 節）は、低ランク更新 **BA** を **B** $= \mathbf{0}$ およびランダムな **A** で初期化するため、最初の勾配ステップでは事前学習済みの重みとは無関係なランダムな k 次元部分空間を通してしか入力を捉えられない。いくつかの派生手法は、この初期化やパラメータ化を単純な方法で変更している。PiSSA (Meng et al. 2024) は **BA** を事前学習済みの重みのランク k 打ち切り SVD で初期化して残差を凍結するため、ファインチューニングの開始直後から元の行列の主方向を適応させることができる。DoRA (S.-Y. Liu et al. 2024) は重みの各列をスカラーの大きさと単位方向に分解し、大きさは直接訓練しつつ、低ランク更新を方向のみに適用する。

33.6.3.4 アテンションにおける低ランク構造

n トークンの系列に対する自己アテンションは $n \times n$ のスコア行列を作るため、アテンションの各章で扱う効率的なアーキテクチャのいくつかは、この行列またはそのキー・値キャッシュを低ランク因子分解（low-rank factorization）に置き換えている。線形アテンション（10.6 節）はアテンションスコアを特徴量のドット積 $\phi(\mathbf{q}) \cdot \phi(\mathbf{k})$ として表し、スコア行列のランクを特徴量の次元以下に抑える。マルチヘッド潜在アテンション（multi-head latent attention、11.3.3.2 節）(DeepSeek-AI 2024) は、ヘッドごとに別々のキーと値を保持する代わりにトークンあたり 1 つの低次元潜在ベクトルを保存することでキャッシュを圧縮する。Linformer (Wang et al. 2020) は、訓練済みアテンション行列の特異値が急速に減衰するという経験的な観測から出発し、キーと値を系列次元に沿って固定長へと射影する。

33.6.3.5 他のノルムのもとでの最急降下法

33.5.4.2 節では、勾配行列の極因子（polar factor）$\mathbf{UV}^\top$ を計算した。幾何学的には、この行列はステップサイズをユークリッドノルムではなくスペクトルノルムで測ったときの最急降下方向

である。Bernstein and Newhouse (2024) はこの知見を深層学習のオプティマイザに対する統一的な視点へと発展させた。移動平均を無効にすると、Adam は符号降下法（sign descent、すなわち ℓ_∞ ノルムのもとでの最急降下法）に帰着し、Shampoo はスペクトルノルムのもとでの最急降下法に帰着する。この視点に立てば、各層に適切なノルムを選ぶことこそが中核となる設計上の判断であり、更新則はそこから自動的に導かれる。この考え方に基づいて構築されたスペクトルノルムオプティマイザは 9.6 節で展開する。

33.6.4 書籍、講義、講義ノート

書籍

- 『Mathematics for Machine Learning』（Deisenroth、Faisal、Ong 著）：第 2 章（線形代数）、第 3 章（解析幾何）、第 4 章（行列分解）で本章の内容を機械学習の視点から網羅している無料の教科書である。
- 『Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares』（Boyd、Vandenberghe 著）：最小二乗法とデータを軸に構成されたスライド・コード付きの無料の書籍であり、33.1 節および 33.5 節とよく対応している。
- 『Introduction to Linear Algebra, 6th ed.』および『Linear Algebra and Learning from Data』（Gilbert Strang 著）：列による描像（column picture）と 4 つの基本部分空間を軸に据えた古典的な入門書と、データ解析および最適化に向けて SVD を中心に据えたその姉妹編である。
- 『Linear Algebra Done Right, 4th ed.』（Sheldon Axler 著）：行列式を使わずに固有値、作用素、およびスペクトル定理（33.3 節）を展開する無料の教科書である。
- 『Numerical Linear Algebra』（Trefethen、Bau 著）：条件性（conditioning）、安定性、QR 法、ならびに `solve`、`eig`、`svd` の背後にある浮動小数点アルゴリズムをわかりやすく解説した書籍である。

講義とチュートリアル

- 「MIT 18.06 Linear Algebra」および「MIT 18.065 Matrix Methods in Data Analysis, Signal Processing, and Machine Learning」（Gilbert Strang 担当）：講義ノートと演習問題を備えた完結した講義コースである。
- 3Blue1Brown による「Essence of Linear Algebra」および Steve Brunton による「Singular Value Decomposition」：線形変換、行列式、固有ベクトル、および SVD を視覚的に解説する講義シリーズである。
- Kolter、Do による「Linear Algebra Review and Reference」（Stanford CS229）および Ström、Åström、Akenine-Möller による『Immersive Linear Algebra』。

Chapter 34

微分積分と自動微分

本書全体を通して、ニューラルネットワークの訓練は同じリズムをたどる。損失を評価し、フレームワークにその勾配を求めさせ、すべての重みを谷の方向へ小さなステップだけ動かすという手順である。これまではその仕組みに関する 2 つの自然な問いに立ち止まることなく、このループに頼ってきた。第 1 に、学習率を 2 倍にした途端、訓練が突然発散したとする。勾配そのものは依然として正しいが、局所的な傾きで損失を確実に予測できる小さな近傍を、そのステップが飛び越えてしまったのである。この線形近似（linear approximation）はどこまで信頼できるのだろうか。第 2 に、勾配がいかに驚くほど低コストであるかに注目しよう。1 億個の重みを 1 つずつ摂動させてすべての傾きを推定するには 1 億回の順伝播（forward pass）を要するが、1 回の逆伝播の呼び出しは、およそ数回の順伝播に相当するコストですべての傾きを返す。なぜそのようなことが可能なのだろうか。

いずれの問いも微分積分に属する。微分積分とは、連続的な変化をどう測るか、そして無限に多くの微小な断片をどう足し合わせるかを Newton と Leibniz が問うた 17 世紀に生まれた分野である。その 2 つの道具である微分と積分は、互いに逆の関係にあることがわかる。深層学習において、導関数は現在の重みの近傍における損失曲面（loss surface）の最良の線形モデルを与え、損失の値そのものを除けば、オプティマイザが目にする唯一の情報である。一方、データや重みが確率的になると直ちに積分が登場する。確率や期待損失は曲線の下の面積であり、方策勾配（policy gradient、14.7 節）のような手法では、サンプリング元の分布そのものに関してその積分を微分しなければならない。

本書の序盤では、微分の基本規則を述べ（1.4 節）、自動微分を既製のツールとして扱った（1.5 節）。本編の各章ではそれ以上の知識を必要としないため、この付録の章は、その仕組みがなぜ機能し、その保証がどこから得られるのかを知りたい読者のための任意の読み物である。議論の根幹を担うのは 3 つの結果である。すなわち、曲率（curvature）の上界を 9.1 節で引用した安全なステップサイズの規則へと変換する降下補題（descent lemma）、損失側から逆向きに掛けることではるかに低コストとなり誤差逆伝播法（backpropagation、4.3 節）を導くヤコビ行列（Jacobian）の積としての連鎖律（chain rule）、そして汎用性と分散の間でトレードオフをなす、期待値の勾配に対する相補的な 2 つの推定量である。

本章は 3 つのパートで展開する。まず 1 変数（34.1 節）、続いて多変数（34.2 節）において関数の線形近似と二次近似（quadratic approximation）を構築し、安全なステップサイズに上界を与え、極小点と鞍点（saddle point）を見分けられるようにする。次に、計算グラフ（computation graph）上の経路にわたる和（34.3 節）から各層のヤコビ行列の積（34.4 節）、さらにはそれらを自動的に乗算するアルゴリズム（34.5 節）へと連鎖律をたどる。最後のパートでは積分（34.6 節）を導入し、サンプリングと微分を組み合わせて期待値の勾配を推定する（34.7 節）。

初等的な微分積分にすでに慣れている場合は、いくつかの節の冒頭にあるウォームアップのページを流し読みしてもよい。本章では、勾配を超えるベクトル解析（発散やストークスの定理など）および測度論（measure theory）は割愛する。微分方程式については第 38 章を参照されたい。

34.1 1 変数の微分積分

「基礎」パートのすべてのモデルは、勾配降下法（gradient descent）によって訓練した。各ステップにおいて、オプティマイザは現在の重みにおける損失とその傾きを評価し、重みを下り方向へ少しだけ動かして、これを繰り返す。他のどの点でも損失を調べないため、すべてのステップは局所的な予測に依拠している。すなわち、現在の点における傾きが、少し離れた位置の損失も記述していると仮定するのである。その予測はどれほど正確であり、どの程度大きなステップまで安全に支えられるのだろうか。

重みが 1 つだけの場合でも、その答えがどのような形になるかはすでに見て取れる。1 つの訓練例、1 つのスカラー重み w、およびロジスティック損失（logistic loss）$\ell(w) = \log(1 + e^{-w})$ を考えよう。$w = 0$ において、損失は 0.693147 であり、その傾きは $-\frac{1}{2}$ である。重みを $w = 0.1$ に動かすと、$w = 0$ を通りその傾きを持つ直線は、損失を $0.693147 - 0.05 = 0.643147$ と予測する。実際の損失は 0.644397 であるため、線形予測のずれはわずか 0.00125 にすぎない。注目すべきことに、この小さな差は単純な構造を持っている。表示された桁の範囲で、この差は $\frac{1}{2} \cdot \frac{1}{4} \cdot 0.1^2$ に等しく、ここで因子 $\frac{1}{4}$ は ℓ のグラフが $w = 0$ でどれだけ急激に曲がっているかを表す。傾きが予測を与え、曲がり具合がステップを大きくしたときにその予測がどれだけ速くずれていくかを示すのである。微分の公式を手にしたのち、ほどなくこれら両方の定数を導出しよう。

本節では、一階導関数（first derivative）から局所的な直線モデルを導き、さらに二階導関数（second derivative）を用いて曲がり具合も捉える放物線へと拡張する。議論の軸となるのは 3 つの結果である。点における傾きを区間上の変化へと結びつける平均値の定理（mean value theorem）、直線と放物線の両方の誤差に上界を与えるラグランジュの剰余項（Lagrange remainder）、そして曲率（curvature）の上界を勾配降下法の保証された安全なステップサイズへと変換する降下補題（descent lemma）である。さらに、ニュートン法や、接線が存在しないにもかかわらず誤差逆伝播法（backpropagation）が数値を返さなければならない ReLU の角も扱う。ここではこれらすべての概念を 1 変数で学び、多変数の最適化理論は第 35 章に委ねる。

先立つ 1.4 節では導関数（derivative）を導入し、微分の公式を列挙して、自動微分（automatic differentiation）を実用的なツールとして用いた。ここではそれらの公式を手短に復習し、証明、幾何学的直観、および誤差の上界に焦点を当てる。微分の公式にすでに慣れている場合は、34.1.3 節へ直接進んでもよい。あらゆる多変数の損失には 1 変数の構造が含まれているため、1 変数から始めるのは自然である。すなわち、$\ell(w_1, \dots, w_n)$ の重みのうち w_i 以外をすべて固定すれば 1 次元スライスが得られ、1 変数の微分積分とはそのようなスライスを調べる学問にほかならない。

本節を通して用いる具体例は、**ソフトプラス**関数 $s(x) = \log(1 + e^x)$ である。これは本章全体で使われる log-sum-exp 関数の 1 変数スライス $\log(e^0 + e^x)$ であり、冒頭のロジスティック損失は単に $\ell(w) = s(-w)$ と表せる。

34.1.1 導関数

34.1.1.1 差分商

滑らかな関数 f のグラフを点 x の近傍で十分に拡大すると、曲線は直線と見分けがつかなくなる（Figure 34.1）。その直線の傾きこそが導関数である。

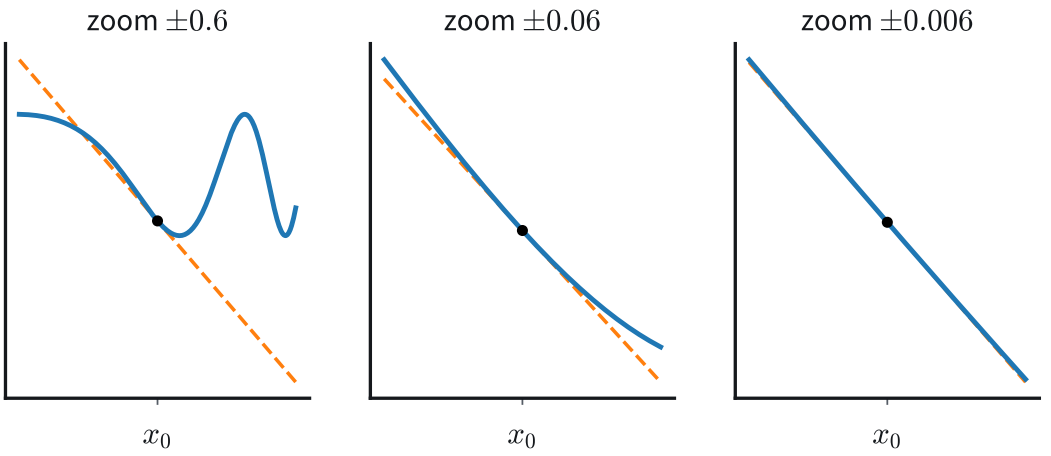


Figure 34.1: 曲線 $\sin(x^x)$ を $x_0 = 2$ の近傍で、半幅 0.6、0.06、0.006 の表示窓で示したもの。窓が狭まるにつれて、曲線（実線）はその接線（破線）と見分けがつかなくなる。

この傾きを代数的に捉えるため、1.4 節にならい、$(x, f(x))$ と $(x+\epsilon, f(x+\epsilon))$ を通る**割線**（secant line）から始める。その傾きは**差分商**（difference quotient）$\big(f(x+\epsilon) - f(x)\big)/\epsilon$ である。ϵ がゼロに向かって小さくなるにつれて、割線は接線に向かって回転する（Figure 34.2）。

割線の傾きが接線の傾きの信頼できる代用となるには、ϵ はどれほど小さくなければならないだろうか。残念ながら普遍的なしきい値は存在せず、関数に完全に依存する。多項式 $f(x) = x^2 + 1701(x-4)^3$（$x = 4$ における真の傾きは 8）を考え、ϵ が小さくなるにつれて割線の傾きがいかにゆっくりと落ち着くかを観察してみよう。

```
def f(x):
    return x**2 + 1701*(x-4)**3

# Slope of the secant through x=4, for shrinking epsilon
for epsilon in [0.1, 0.001, 0.0001, 0.00001]:
    print(f'epsilon = {epsilon:.5f} -> {(f(4+epsilon) - f(4)) / epsilon:.5f}')
```

```
epsilon = 0.10000 -> 25.11000
epsilon = 0.00100 -> 8.00270
epsilon = 0.00010 -> 8.00012
epsilon = 0.00001 -> 8.00001
```

$\epsilon = 0.1$ では、割線の傾きは 25.11 と真の極限の 3 倍以上にもなり、出力値が 8 の付近に落ち着くのは ϵ が 0.001 に達してからである。$f(4+\epsilon) - f(4) = 8\epsilon + \epsilon^2 + 1701\epsilon^3$ を展開すればその理由がわかる。差分商は $8 + \epsilon + 1701\epsilon^2$ に等しく、三次の項による寄与は 17（$\epsilon = 0.1$ のとき）に達する一方、1.7×10^{-7}（$\epsilon = 10^{-5}$ のとき）にすぎない。大きな係数は極限値そのものを変えることなく、極限への接近を遅らせる。自動微分はこの問題を完全に回避する。ϵ を選ぶことは一切せず、微分の公式を直接適用するからである（34.5 節）。

34.1.1.2 導関数と線形近似

関数 f の点 x における**導関数**は、割線の傾きの極限として次式で定義される。

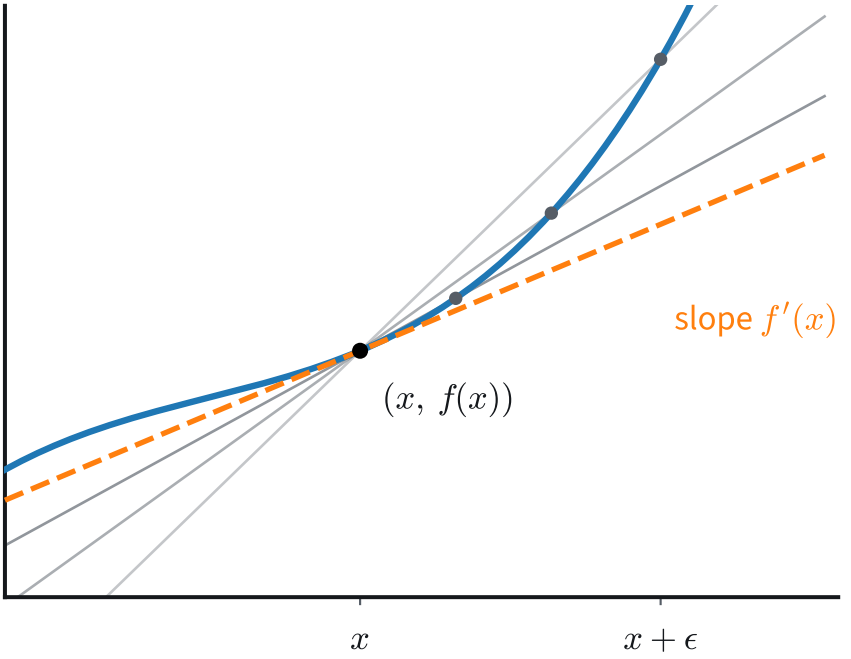


Figure 34.2: 差分商は $(x, f(x))$ と $(x+\epsilon, f(x+\epsilon))$ を通る割線の傾きである。$\epsilon \to 0$ につれて、この傾きは接線の傾き $f'(x)$ に近づく。

$$f'(x) = \frac{df}{dx}(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}, \tag{34.1}$$

ただし、これはこの極限が存在する場合に限られ、そのとき f は x で**微分可能**（differentiable）であるという。上の三次関数では $f'(4) = 8$ である。表記としては f' または $\frac{df}{dx}$ を用い、かさばる式を微分するときは $\frac{d}{dx}[...]$ と書く。また、$|x|$ の点 0 におけるように極限が存在しないこともあり、そのような角については 34.1.6 節で改めて扱う。

この定義を書き換えてみると示唆に富む。直線予測の誤差を $r(\epsilon) = f(x+\epsilon) - f(x) - \epsilon f'(x)$ と置くと、極限 Equation 34.1 は $r(\epsilon)/\epsilon \to 0$ を意味する。そこで、記法 $o(\epsilon)$ を、量 $r(\epsilon)$ のうち $r(\epsilon)/\epsilon \to 0$（$\epsilon \to 0$ のとき）を満たす任意のものを表すために用いると、次の**線形近似**（linear approximation）が得られる。

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + o(\epsilon). \tag{34.2}$$

平易な言葉で言えば、微小なステップ ϵ によって出力は $\epsilon f'(x)$ だけ変化し、その誤差は ϵ より速くゼロに収束する。逆も成り立つ。すなわち、$f(x+\epsilon) = f(x) + a\epsilon + o(\epsilon)$ がある定数 a に対して成り立つならば、両辺を ϵ で割って $\epsilon \to 0$ とすることで、f が x で微分可能であり $f'(x) = a$ となることが証明される。したがって、展開 Equation 34.2 と極限 Equation 34.1 は同じ情報を持っており、和の微分公式や連鎖律の証明はこの同値性に依拠する。

$x+\epsilon$ の関数としてプロットすると、直線 $f(x)+\epsilon f'(x)$ は点 x におけるグラフの**接線**（tangent）、すなわち回転する割線の極限となる。これが $w = 0.1$ におけるロジスティック損失を予測した直

線であり、0.00125 の差がその $o(\epsilon)$ の剰余項であった。それ単体では、$o(\epsilon)$ はほとんど何も保証しない。$o(\epsilon)$ のオーダーの誤差は、ϵ よりほんのわずかに速く縮むだけかもしれないからである。しかし今回の例では差は ϵ^2 に比例して縮んでおり、34.1.5.2 節で見るように、二次の誤差減衰は 2 回微分可能なすべての関数に共通する法則である。

34.1.2 導関数の計算

実際には、極限を毎回計算して導関数を求める人はいない。その代わりに、1.4 節と同様、初等関数の導関数の短い表と、和・積・合成に関する公式を組み合わせる。ここでは各公式を線形近似から直接証明する。表の項目のうち、指数関数、対数関数、べき乗の微分公式については導出を行い、正弦・余弦の項目、および傾きが 1（点 0 において）となる指数関数の底が存在することは、微分積分の初等講義で既知のものとする。

34.1.2.1 代表的な導関数の表

以下の公式は右辺が定義されるすべての点で成り立つ。すなわち、べき乗の微分公式は実数 n のすべて（$x > 0$ のとき）および整数 n（x^n が定義されるすべての点）に対して成り立ち、対数関数は $x > 0$ で成り立つ。

- **定数の微分：** $\frac{d}{dx} c = 0$。
- **一次関数の微分：** $\frac{d}{dx}(ax) = a$。
- **べき乗の微分公式：** $\frac{d}{dx} x^n = n x^{n-1}$。
- **指数関数の微分：** $\frac{d}{dx} e^x = e^x$。
- **対数関数の微分：** $\frac{d}{dx} \log(x) = \frac{1}{x}$。
- **正弦の微分：** $\frac{d}{dx} \sin(x) = \cos(x)$。
- **余弦の微分：** $\frac{d}{dx} \cos(x) = -\sin(x)$。

指数関数の項目は、なぜ底 e が特別なのかも明らかにする 1 行の考察に基づいている。任意の $a > 0$ に対し、a^x の差分商は次のようにきれいに因数分解できる。

$$\frac{a^{x+\epsilon} - a^x}{\epsilon} = a^x \, \frac{a^\epsilon - 1}{\epsilon},$$

ここで 2 番目の因数は、単に点 0 における差分商である。したがって、a^x が点 0 で微分可能でありその傾きが c_a ならば、$\frac{d}{dx} a^x = c_a \, a^x$ がすべての x に対して成り立つ。すなわち、指数関数の傾きは常にその高さに比例し、底ごとにただ 1 つの比例定数を持つ。数値的には、$\epsilon = 10^{-6}$ において $(2^\epsilon - 1)/\epsilon = 0.69315$（底 2）および $(10^\epsilon - 1)/\epsilon = 2.30259$（底 10）が得られる。数 e はこの定数が 1 となる唯一の底として定義され、直ちに $\frac{d}{dx} e^x = e^x$ が従う。対数関数の導関数が得られれば、連鎖律によって他のすべての底に対する c_a も特定できる。

34.1.2.2 導関数を組み合わせる公式

損失関数は、上の表をもとに和、積、商、合成を用いて構築される。g と h は x で微分可能であるとする（連鎖律については、h が x で微分可能であり、g が $h(x)$ で微分可能であるとする）。このとき、次が成り立つ。

- **和の微分公式：** $\frac{d}{dx}\big(g(x) + h(x)\big) = g'(x) + h'(x)$。
- **積の微分公式：** $\frac{d}{dx}\big(g(x)\,h(x)\big) = g(x)h'(x) + g'(x)h(x)$。
- **商の微分公式：** $\frac{d}{dx}\frac{g(x)}{h(x)} = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$（ただし $h(x) \neq 0$）。

- **連鎖律：** $\frac{d}{dx} g(h(x)) = g'(h(x))\, h'(x)$。

和と積の微分公式は型通りに示せるが、連鎖律だけは十分な注意を要する。素朴な議論では $h(x+\epsilon) - h(x)$ で割ることになるが、これは $\epsilon \neq 0$ であってもゼロになり得るからである。これらの証明はすべて 1 つの基本的な事実に依拠する。すなわち、**x で微分可能な関数は x で連続である**。なぜなら、$g(x+\epsilon) - g(x) = \epsilon \cdot \frac{g(x+\epsilon)-g(x)}{\epsilon}$ は $0 \cdot g'(x) = 0$ へと（$\epsilon \to 0$ のとき）収束するからである。

和の微分公式： g と h の線形近似を足し合わせると、次式が得られる。

$$g(x+\epsilon) + h(x+\epsilon) = g(x) + h(x) + \epsilon\big(g'(x) + h'(x)\big) + o(\epsilon),$$

これは、ϵ より速くゼロに収束する 2 つの量の和もまた ϵ より速くゼロに収束するためである。線形近似の逆命題により、ϵ の係数が導関数となる。すなわち、一次の変化は単に足し合わされる。■

積の微分公式： 分子において $g(x+\epsilon)h(x)$ を足して引くと、積の差分商は既知の 2 つの商へと分解される。

$$\frac{g(x+\epsilon)h(x+\epsilon) - g(x)h(x)}{\epsilon} = g(x+\epsilon)\,\frac{h(x+\epsilon) - h(x)}{\epsilon} + h(x)\,\frac{g(x+\epsilon) - g(x)}{\epsilon}.$$

$\epsilon \to 0$ のとき、連続性より $g(x+\epsilon) \to g(x)$ となるため、右辺は $g(x)h'(x) + h(x)g'(x)$ に収束する。■

連鎖律： $h(x+\epsilon) - h(x)$ で割る代わりに、内側の関数の変化を外側の関数の線形近似へ直接代入する。g は $h(x)$ で微分可能であるため、誤差比 $\rho(t) = \frac{g(h(x)+t) - g(h(x))}{t} - g'(h(x))$（$t \neq 0$ に対して定義し、$\rho(0) = 0$ と拡張する）は 0 へと（$t \to 0$ のとき）収束する。したがって、任意の t（$t = 0$ を含む）に対して次のように書ける。

$$g(h(x) + t) = g(h(x)) + \big(g'(h(x)) + \rho(t)\big)\, t.$$

ここで $t = h(x+\epsilon) - h(x) = \epsilon h'(x) + o(\epsilon)$ と置く。h の連続性により、$t \to 0$（$\epsilon \to 0$ のとき）となるため、$\rho(t) \to 0$ も成り立つ。この t を代入すると次式が得られる。

$$\begin{aligned} g(h(x+\epsilon)) - g(h(x)) &= \big(g'(h(x)) + \rho(t)\big)\big(\epsilon h'(x) + o(\epsilon)\big) \\ &= \epsilon\, g'(h(x))\, h'(x) + o(\epsilon), \end{aligned}$$

なぜなら、$g'(h(x))\, \epsilon h'(x)$ 以外のすべての項は、ϵ より速くゼロに収束する因数を含んでいるからである。線形近似の逆命題を適用すれば証明が完了する。■

合成関数に沿うと変化率は掛け合わされる。連鎖律をニューラルネットワークに層ごと適用すると局所的な傾きの積が生じ、その積を効率的に評価することこそが誤差逆伝播法の役割にほかならない。商の微分公式は、$g \cdot h^{-1}$ に積の微分公式と連鎖律を適用すれば直ちに従う。

34.1.2.3 逆関数

連鎖律は、別の関数の働きを打ち消す関数も微分できる。関数 g が $g(f(x)) = x$ をある点の近傍のすべての x に対して満たすとき、それをその点における f の**逆関数**（inverse）という。そのような g が存在し、$f(x)$ で微分可能であるとしよう。ここでは証明なしに用いる逆関数定理（inverse function theorem）によれば (Rudin 1976)、f' が x の近傍で連続かつ非ゼロであれば、この両方が成り立つ。$g(f(x)) = x$ の両辺を連鎖律で微分すると、次式が得られる。

$$g'(f(x))\,f'(x) = 1, \qquad \text{so} \qquad g'(y) = \frac{1}{f'(g(y))} \quad \text{at } y = f(x).$$

言い換えれば、逆関数の傾きは、対応する点で評価した f の傾きの逆数である。幾何学的には、g のグラフは f のグラフを対角線に関して折り返したものであり、直線を対角線に関して折り返すと縦の変化と横の変化の比が上下逆転する。

対数関数は e^x を打ち消すため、この公式から $\log'(y) = 1/e^{\log y} = 1/y$、すなわち先ほどの表の 5 番目の項目が得られる。なお、条件 $f'(x) \neq 0$ は不可欠である。$f(x) = x^3$ では $f'(0) = 0$ となるため、$y = 0$ でこの式は破綻する。実際、その逆関数 $x^{1/3}$ は点 0 で微分可能ではない。差分商 $\epsilon^{1/3}/\epsilon = \epsilon^{-2/3}$ が限りなく大きくなり、その点でグラフが垂直な接線を持つからである。

対数関数の微分が手に入ったことで、連鎖律によって表の残り 2 つの項目も解決する。$a^x = e^{x \log a}$ と書けば $\frac{d}{dx} a^x = (\log a)\, a^x$ が得られ、定数が $c_a = \log a$ であることがわかる。実際、$\log 2 = 0.69315$ と $\log 10 = 2.30259$ は先ほど数値的に測定した定数そのものである。同様に、$x^n = e^{n \log x}$（$x > 0$ のとき）と書けば、$\frac{d}{dx} x^n = e^{n \log x} \cdot \frac{n}{x} = n x^{n-1}$ がすべての実数 n に対して得られる。

34.1.2.4 ソフトプラス、シグモイド、ロジット

これで、本節の具体例である関数 $s(x) = \log(1 + e^x)$ にこれらの公式を適用できる。外側の関数を log、内側の関数を $1 + e^x$ として連鎖律を用いると、次式が得られる。

$$s'(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} = \sigma(x),$$

したがって、ソフトプラスの導関数は**シグモイド** σ である。$\sigma(x) = (1 + e^{-x})^{-1}$ に対して、外側の関数を u^{-1}、内側の関数を $1 + e^{-x}$（傾きは $-e^{-x}$）として 2 回目の連鎖律を適用すると、次式が得られる。

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x)),$$

これは $1 - \sigma(x) = e^{-x}/(1 + e^{-x})$ だからである。積 $p(1-p)$ は $p = \frac{1}{2}$ で最大となるため、シグモイドは原点で最も急になり、そこでは $\sigma(0) = \frac{1}{2}$ かつ $\sigma'(0) = \frac{1}{4}$ である。σ の逆関数は**ロジット**関数 $\operatorname{logit}(p) = \log \frac{p}{1-p}$ であり、逆関数の微分公式から直ちに $\operatorname{logit}'(p) = 1/\sigma'(\operatorname{logit} p) = \frac{1}{p(1-p)}$ が得られる。$p = \frac{1}{2}$ ではこの値は $4 = 1/\frac{1}{4}$ となり、シグモイドの傾きの逆数と一致する。

ここで、冒頭のロジスティック損失の例における 2 つの定数を確かめることができる。$\ell(w) = s(-w)$ であるから、連鎖律より $\ell'(w) = -\sigma(-w)$ となり、$\ell'(0) = -\frac{1}{2}$ が従う。もう一度微分すると $\ell''(w) = \sigma(-w)\big(1 - \sigma(-w)\big)$ となり、原点での曲率 $\ell''(0) = \frac{1}{4}$ が得られる。$\ell(\epsilon)$ とそ

の線形近似 $\ell(0) - \frac{1}{2}\epsilon$ の差が、実際に ϵ^2 に比例して先頭係数 $\frac{1}{2}\ell''(0) = \frac{1}{8}$ を伴い縮んでいくかを数値的に確認してみよう。

```
# The logistic loss l(w) = log(1 + e^{-w}) near w = 0, where l'(0) = -1/2
def ell(w):
    return np.log(1 + np.exp(-w))

for eps in [0.1, 0.01, 0.001]:
    line = ell(0) - 0.5 * eps          # the linear approximation
    gap = ell(eps) - line
    print(f'eps = {eps:<5}: l = {ell(eps):.6f}, line = {line:.6f}, '
          f'gap = {gap:.2e}, gap / eps^2 = {gap / eps**2:.4f}')
```

```
eps = 0.1  : l = 0.644397, line = 0.643147, gap = 1.25e-03, gap / eps^2 = 0.1249
eps = 0.01 : l = 0.688160, line = 0.688147, gap = 1.25e-05, gap / eps^2 = 0.1250
eps = 0.001: l = 0.692647, line = 0.692647, gap = 1.25e-07, gap / eps^2 = 0.1250
```

確かに、ϵ を 10 分の 1 にするたびに差は 100 分の 1 に減少し、gap/ϵ^2 は $0.125 = \frac{1}{2}\ell''(0)$ へと落ち着く。線形近似の誤差は確かに二次であり、その先頭係数は曲率によって決まるのである。また、1 行目の出力が 0.1249 であり 0.1250 ではないことにも注目しよう。差は厳密には $\epsilon^2/8$ ではなく、このわずかな違いを生む高次の項については 34.1.5.2 節で特定する。

これまでの計算はすべて、1 つの点の無限小の近傍におけるものであった。傾きから区間全体にわたる関数の振る舞いについて何がわかるかという問いは、また別の種類の問題である。

34.1.3 平均値の定理

最適化では日常的に 2 つの直観に従って行動する。傾きが正ならば関数は上り坂にあり、傾きがゼロになる点は最小点の候補であるという直観である。勾配降下法はこの両方に依拠している。しかし、いずれの主張も個々の点における導関数から区間上の f の振る舞いを推論するものであり、導関数の定義自体に組み込まれているわけではない。まずはより単純な 2 番目の主張から証明しよう。

補題（Fermat） 関数 f が区間の内点 ξ で極大値または極小値をとり、かつ ξ で微分可能であるならば、$f'(\xi) = 0$ である。

証明：極大値をとる点では、$f(\xi + \epsilon) - f(\xi) \le 0$ が正負いずれの符号でも十分に小さいすべての ϵ に対して成り立つ。したがって差分商は、≤ 0（$\epsilon > 0$ のとき）かつ ≥ 0（$\epsilon < 0$ のとき）となる。f は ξ で微分可能であるため、両方の片側極限はともに $f'(\xi)$ に等しくなければならず、同時に ≤ 0 かつ ≥ 0 である数はゼロしかない。極小値の場合も、不等号の向きを逆にすれば同じ議論から従う。■

1 番目の直観を示すには、より深い道具である平均値の定理が必要になる。本節の残りで導く誤差の上界はすべて、適切に選んだ補助関数にこの定理を適用することで得られる。幾何学的には（Figure 34.3）、区間 $[a, b]$ 上のグラフの両端を結ぶ弦（chord）が、ある内点における接線と平行になることを主張する定理である。

$f(x) = x^2$ の区間 $[0, 2]$ 上といった単純な例では、弦の傾きは $(4 - 0)/2 = 2$ であり、接線の傾き $f'(\xi) = 2\xi$ は中点 $\xi = 1$ でこれと一致する。ただし、この定理にはすべての内点での微分可能性が真に必要であることに注意しよう。絶対値 $|x|$ は $[-1, 1]$ 上で $f(-1) = f(1)$ を満たすためその弦は水平であるが、傾き 0 を持つ内点は存在しない。傾きは左側で -1、右側で $+1$ であり、点 0 では定義されないからである。

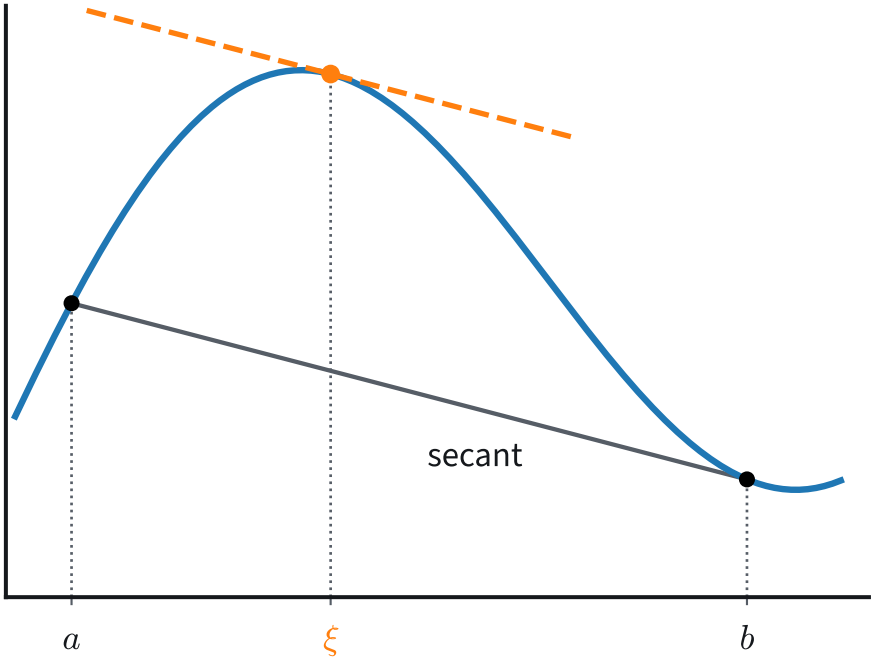


Figure 34.3: a から b までの区間において、両端を通る割線の弦の傾きは平均変化率に等しい。平均値の定理は、その弦と平行な同じ傾きの接線を持つ内点 ξ が存在することを保証する。

命題（平均値の定理） f **が** $[a,b]$ **上で連続かつ** (a,b) **上で微分可能であるならば、次式を満たす点** $\xi \in (a,b)$ **が存在する。**

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}. \tag{34.3}$$

証明： まず、ロルの定理（Rolle's theorem）として知られる両端の高さが等しい場合 $f(a) = f(b)$ を考える。ここで実解析の基礎的な結果である**最大値・最小値の定理**（extreme value theorem）、すなわち閉区間上の連続関数は最大値と最小値の両方をとるという事実を用いる (Rudin 1976)。もし両方の極値が端点で達成されるならば、f は定数であり内部のいたるところで $f' \equiv 0$ となる。そうでなければ、少なくとも 1 つの極値は内点 ξ で達成され、フェルマーの補題（Fermat's lemma）によりそこで $f'(\xi) = 0$ となる。

一般の場合には、弦を引いて両端の高さを揃える。傾きを補正した関数 $\tilde{f}(x) = f(x) - \frac{f(b)-f(a)}{b-a}(x-a)$ を定義しよう。$\tilde{f}(a) = \tilde{f}(b) = f(a)$ であるから、ロルの定理によりある内点 ξ で $\tilde{f}'(\xi) = 0$ となり、これを整理すれば直ちに Equation 34.3 が得られる。■

言葉で言い換えれば、区間上の平均変化率は必ずその内部のある点における瞬間変化率として実現される。ここから導かれる 2 つの帰結は、本章全体で繰り返し登場する。

1. **ある区間上で** $f' > 0$ **ならば、その区間上で** f **は狭義単調増加である。** 区間内の任意の $x_1 < x_2$ に対し、Equation 34.3 より $f(x_2) - f(x_1) = f'(\xi)(x_2 - x_1) > 0$ が両者の間にある何らかの ξ に対して成り立つ。同様に、$f' < 0$ ならば f は狭義単調減少であり、$f' = 0$ ならば f は定数である。

2. **$f' < 0$ が $(x_0 - \delta, x_0)$ 上で成り立ち、$f' > 0$ が $(x_0, x_0 + \delta)$ 上で成り立ち（ある $\delta > 0$ に対して）、さらに f が x_0 で連続であるならば、x_0 は狭義の極小点である。**それぞれの側に 1 つ目の帰結を適用すれば、f は左側から x_0 に向かって減少し、そこから右側へ向かって増加することがわかる。

通し例の関数に立ち返ると、いたるところで $s' = \sigma > 0$ であるから、ソフトプラスは $\mathbb{R}$ 全体で狭義単調増加である。

34.1.4 1 変数の勾配降下法

34.1.4.1 勾配降下法のステップ

線形近似は、関数 f が微小なステップ ϵ に対してどう応答するかを $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$ と予測する。f を減少させるには、ϵ の符号を傾きと逆向きに選べばよい。$\epsilon = -\eta f'(x)$ と正の**ステップサイズ** $\eta > 0$ を用いて置くと、線形モデルは次の変化を予測する。

$$f(x - \eta f'(x)) - f(x) \approx -\eta\, f'(x)^2,$$

この値は $f'(x) \neq 0$ である限り真に負である。傾きが正か負かにかかわらず、それと逆向きに進めば、線形モデルは傾きの二乗に比例する量だけ減少する（Figure 34.4）。

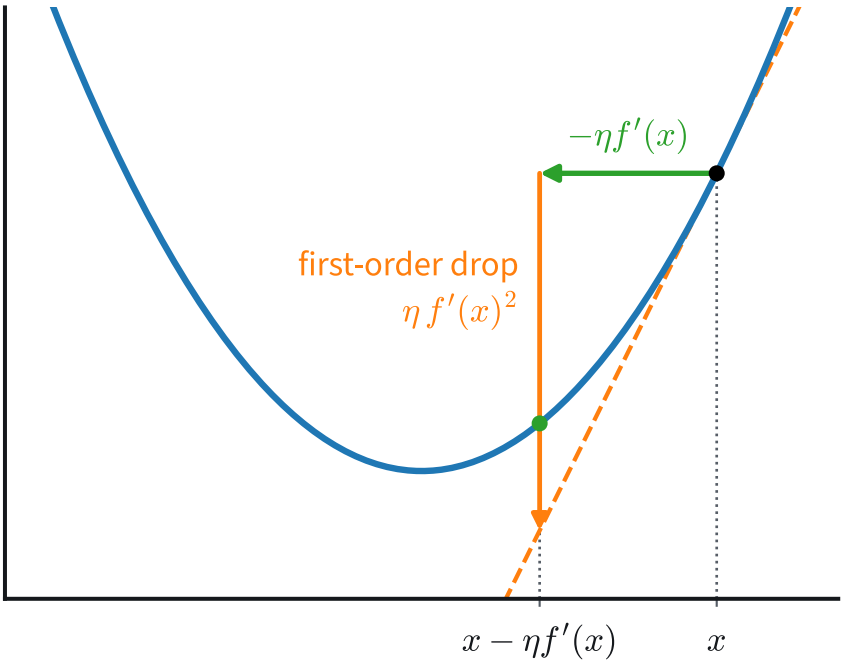


Figure 34.4: 1 次元の谷における勾配降下法の 1 ステップ。点 x における接線の傾きは $f'(x)$ であり、ステップ $-\eta f'(x)$ は傾きと逆向きに動いて曲線上のより低い点 $x - \eta f'(x)$ に到達し、一次近似での減少量は $\eta f'(x)^2$ に等しい。

この更新を反復すると、1 次元の**勾配降下法**が得られる。

$$x_{t+1} = x_t - \eta\, f'(x_t). \tag{34.4}$$

各ステップは局所的な傾きに逆らって進み、反復は**停留点**（stationary point）、すなわち $f'(x_t) = 0$ となる点で静止する。ニューラルネットワークでは、重みの更新式 $w_i \leftarrow w_i - \eta\, \partial \ell / \partial w_i$ の各座標がこの形をとり、すべての座標が同時に更新される。ロジスティック損失において $w = 0$ から出発すると、傾きは $\ell'(0) = -\tfrac{1}{2}$ であるから、大きさ η のステップは重みを $\epsilon = \eta/2$ だけ動かし、損失が $\eta/4$ 減少すると予測する。

ここで、線形モデルの明白な限界に注意しよう。予測される変化量 $-\eta f'(x)^2$ は、$\eta > 0$ がどれほど巨大であっても常に負になる。実際には、ステップが大きすぎると最小点を飛び越えてしまい、出発点よりも高い位置に着地することがある。接線は曲線がどう曲がっているかをまったく知らないため、この危険を警告できない。最も単純な谷である $f(x) = x^2$ を見るだけでも、線形予測がどこで破綻するかが明らかになる。

34.1.4.2 二次関数の場合

$f(x) = x^2$ の場合、傾きは $f'(x) = 2x$ であるから、勾配降下法の更新式は次のように簡約される。

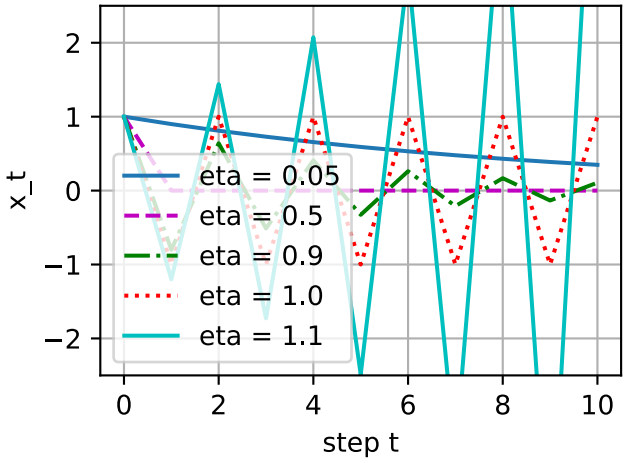
$$x_{t+1} = x_t - \eta\,(2x_t) = (1 - 2\eta)\, x_t.$$

各ステップは現在の反復点に同じ因子 $1 - 2\eta$ を掛けるため、閉じた式 $x_t = (1-2\eta)^t x_0$ が得られる。したがって、反復点列が最小点 $x = 0$ に収束するのは $|1 - 2\eta| < 1$、すなわち $0 < \eta < 1$ のとき、かつそのときに限る。この収束範囲のなかでも、小さい η では単調にじわじわと 0 へ近づき、$\eta = \tfrac{1}{2}$ では 1 ステップで最小点に到達し、$\tfrac{1}{2} < \eta < 1$ では最小点を飛び越えながら内側へ振動する。ちょうど境界の $\eta = 1$ では反復点は $+x_0$ と $-x_0$ の間を永久に往復し、$\eta > 1$ では発散する。

$x_0 = 1$ から各領域を代表する 5 通りのステップサイズについて 10 ステップ実行し、これらの軌跡を並べて見てみよう。

```
# Gradient descent on f(x) = x^2 from x0 = 1, for several step sizes
def gd(eta, steps=10, x0=1.0):
    xs = [x0]
    for _ in range(steps):
        xs.append(xs[-1] - eta * (2 * xs[-1]))  # x <- x - eta * f'(x)
    return xs

etas = [0.05, 0.5, 0.9, 1.0, 1.1]
trajectories = [gd(eta) for eta in etas]
for eta, xs in zip(etas, trajectories):
    print(f'eta = {eta:.2f} -> x_10 = {xs[-1]:+.5f}')
d2l.plot(list(range(11)), trajectories, 'step t', 'x_t', ylim=[-2.5, 2.5],
         legend=[f'eta = {eta}' for eta in etas],
         fmts=('-', 'm--', 'g-.', 'r:', 'c-'))
```

```
eta = 0.05 -> x_10 = +0.34868
eta = 0.50 -> x_10 = +0.00000
eta = 0.90 -> x_10 = +0.10737
eta = 1.00 -> x_10 = +1.00000
eta = 1.10 -> x_10 = +6.19174
```



出力された値は閉じた式のべき乗 $(1-2\eta)^{10}$ と一致している。慎重なステップ $\eta = 0.05$ はまだ 0.349 までしか下りておらず、$\eta = 0.9$ はジグザグに進んで 0.107 まで下がり、$\eta = 1$ はちょうど 1 に戻り、$\eta = 1.1$ はすでに 6.19 へと発散している。安定性のしきい値 $\eta = 1$ が $2/f''$ に等しいことにも注目しよう（$f'' \equiv 2$ が x^2 で成り立つためである）。34.1.5.4 節で見るように、これは偶然ではない。曲率の任意の上界 L から、すべての滑らかな関数に対して安全なステップサイズのしきい値 $2/L$ が導かれるのである。

34.1.5 曲率とテイラー級数

線形近似はどこまで信頼でき、どの程度大きなステップサイズなら安全なのだろうか。どちらの問いも、傾きそのものがどれだけ速く変化するかにかかっている。

34.1.5.1 二階導関数

f' それ自体が x の関数であるため、もう一度微分できる。その結果得られるのが**二階導関数** $f'' = \frac{d^2 f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right)$ であり、傾きの変化率を測る。この操作を n 回繰り返すと、次の n 階導関数が得られる。

$$f^{(n)}(x) = \frac{d^n f}{dx^n} = \left(\frac{d}{dx}\right)^n f.$$

幾何学的には、$f'' > 0$ は傾きが増加していることを意味し、グラフはお椀のように上向きに曲がる。一方、$f'' < 0$ は傾きが減少していることを意味し、グラフはドームのように下向きに曲がる。停留点において、この符号は極小値と極大値を見分ける手がかりとなる。

命題（二階微分判定法） $f'(x_0) = 0$ **とし、** $f''(x_0)$ **が存在すると仮定する。** $f''(x_0) > 0$ **ならば** x_0 **は狭義の極小点であり、** $f''(x_0) < 0$ **ならば狭義の極大点である。**

証明： $f'(x_0) = 0$ であるから、$f''(x_0)$ の定義は $f''(x_0) = \lim_{\epsilon \to 0} f'(x_0+\epsilon)/\epsilon$ に簡約される。この極限が正ならば、商 $f'(x_0+\epsilon)/\epsilon$ は十分に小さいすべての $\epsilon \neq 0$ に対して正でなければならない。ゆえに $f'(x_0+\epsilon)$ は ϵ と同じ符号を持ち、x_0 のすぐ左側では負、すぐ右側では正となる。平均値の定理の 2 番目の帰結により、x_0 は狭義の極小点である。$f''(x_0) < 0$ の場合も、すべての符号を反転させれば同様に示せる。■

この証明において本質的な役割を果たした箇所に注目しよう。f' の局所的な符号パターンを区間上の最小性へと結びつけるには、平均値の定理が不可欠なのである。また、この判定法が保証するのは x_0 の近傍のみであり、定義域全体ではないことにも留意されたい。多変数へ進むと、二階偏導関数のなす行列の固有値（eigenvalue）の符号がこれと同じ役割を果たすことになる。

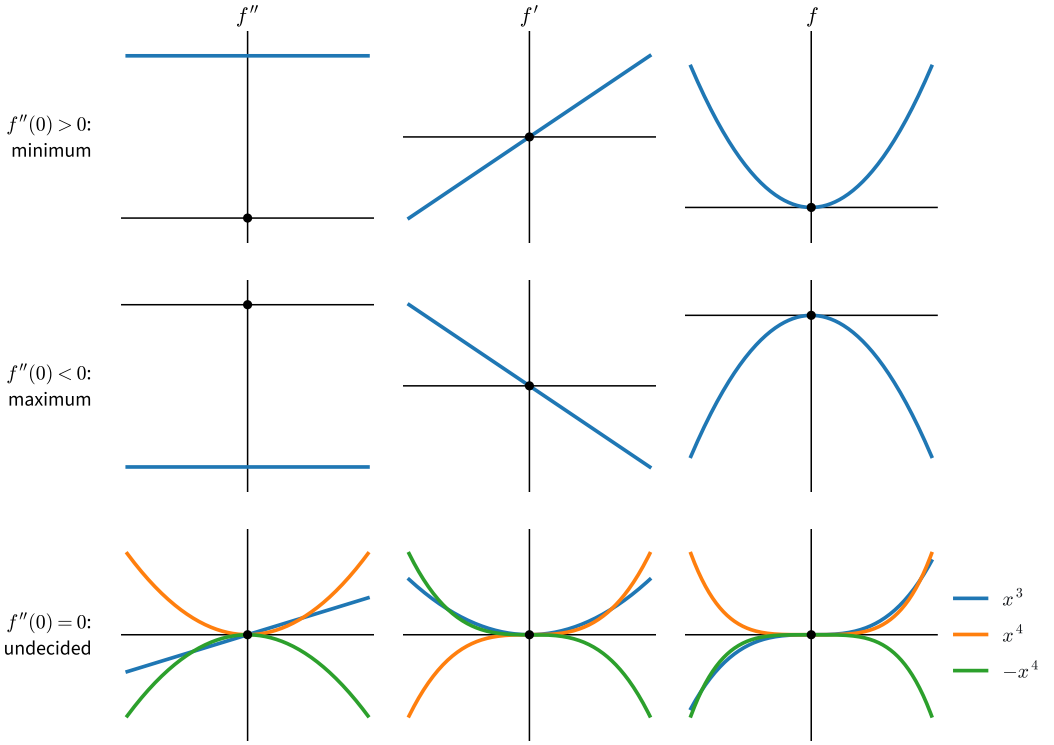


Figure 34.5: 二階微分判定法。各行は上から $f''(0) > 0$（お椀型、極小）、$f''(0) < 0$（ドーム型、極大）、および $f''(0) = 0$ を示し、最下行では x^3、x^4、$-x^4$ がいずれも $f'(0) = f''(0) = 0$ を満たしながら、点 0 でそれぞれ変曲点、極小、極大をとる。各列は左から f''、f'、f である。

停留点において $f''(x_0) = 0$ となるとき、この判定法は何も語らない（Figure 34.5 下段）。x^3、x^4、$-x^4$ はいずれも $f'(0) = f''(0) = 0$ を満たすが、点 0 は 1 番目の関数では変曲点（inflection point）、2 番目では極小点、3 番目では極大点である。このような平坦な停留点の判定には高階導関数が必要となる。

34.1.5.2 テイラー多項式

接線は、関数 f の基点 x_0 における値と傾きに一致する。曲がり具合まで捉えるには、放物線を当てはめればよい。一般の二次関数 $q(x) = ax^2 + bx + c$ の導関数は次の通りである。

$$q'(x) = 2ax + b, \qquad q''(x) = 2a,$$

したがって、その曲率は定数 $2a$ である。f の x_0 における曲率に一致させると $a = f''(x_0)/2$ と定まり、値と傾きに一致させると b と c が定まる。$f(x) = \sin(x)$ では $\sin' = \cos$ かつ $\sin'' = -\sin$ であるから、x_0 において最もよく当てはまる放物線は次式となる。

$$q(x) = \sin(x_0) + \cos(x_0)\,(x - x_0) - \tfrac{1}{2}\sin(x_0)\,(x - x_0)^2.$$

この放物線は正弦曲線の曲がりに追随するため、接線よりも目に見えて広い区間にわたって精度を保つ（Figure 34.6、放物線が非ゼロの曲率を持つよう $\sin x_0 \neq 0$ となる基点で描画している）。

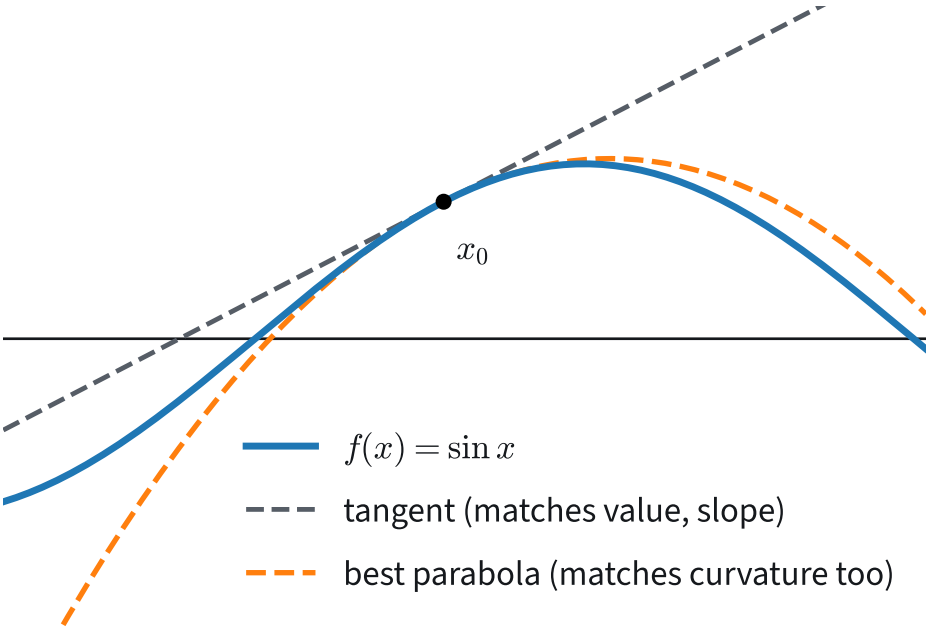


Figure 34.6: 正弦曲線と x_0 における 2 つの局所モデル。接線は値と傾きに一致し、最良の放物線はさらに曲率にも一致してより広い区間で正確さを保つ。

通し例の関数では、$x_0 = 0$ における値 $s(0) = \log 2$、$s'(0) = \frac{1}{2}$、$s''(0) = \frac{1}{4}$ から最良の放物線 $\log 2 + \frac{x}{2} + \frac{x^2}{8}$ が得られる。その二次の係数 $\frac{1}{8}$ は、ロジスティック損失において gap$/\epsilon^2$ が収束していった厳密な定数（$\ell(w) = s(-w)$ に対するもの）にほかならない。

同じ考え方は任意の次数 n へと拡張できる。次数 n の、関数 f の点 x_0 における**テイラー多項式**（Taylor polynomial）は次式で定義される。

$$P_n(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i,$$

これは高々 n 次の多項式のうち、値および最初の n 階までの導関数が f と点 x_0 において一致する唯一のものである。階乗が現れるのは、$(x - x_0)^i$ をちょうど i 回微分して x_0 で評価すると $i!$ が生じ、他のすべての項がゼロになるからである。この記法では、P_1 が接線、P_2 が最良の放物線にあたる。e^x の $x_0 = 0$ においてはすべての導関数が 1 に等しいため $P_n(x) = \sum_{i=0}^{n} x^i/i!$ が得られ、新しい項を加えるたびに多項式は原点からより遠くまで指数曲線に寄り添うようになる。P_n をどこまで信頼でき、その範囲を超えるとどれだけ速く誤差が増大するかを知るには、剰余項の明示的な公式が必要である。

34.1.5.3 ラグランジュの剰余項

平均値の定理を用いると、P_n の誤差を表す厳密な公式が、ただ 1 つの未知の内点を用いて得られる。これは本節の中心をなす結果であり、降下補題もニュートン法の二次収束率（convergence rate）も、その $n = 1$ の場合から直ちに導かれる。

命題（ラグランジュの剰余項） 関数 f が $(n+1)$ 回微分可能であるような開区間が x_0 と x（ただし $x \neq x_0$）を含むならば、次式が成り立つ。

$$f(x) = P_n(x) + R_n(x), \qquad R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,(x - x_0)^{n+1}, \tag{34.5}$$

ここで ξ は x_0 と x の真の間にある何らかの点である。なお、$n = 0$ の場合は平均値の定理 $f(x) = f(x_0) + f'(\xi)(x - x_0)$ を復元する。

証明： 唯一の巧みな工夫は適切な補助関数を選ぶことであり、それ以降は素直な微分計算にすぎない。x を固定し、基点 t を変数とみなす。すなわち、t（x_0 と x の間）に対して次式を定義する。

$$G(t) = f(x) - \sum_{k=0}^{n} \frac{f^{(k)}(t)}{k!}\,(x - t)^k - M\,\frac{(x - t)^{n+1}}{(n+1)!},$$

これは点 x における（t を中心とした）テイラー多項式の誤差から、その誤差がとると予想される形の項を引いたものである。定数 M を $G(x_0) = 0$ となるように選ぶ。これで証明は 3 つのステップで完結する。

第 1 に、G は両端点でゼロになる。M の選び方から $G(x_0) = 0$ が成り立ち、また $G(x) = 0$ も成り立つ（$(x-t)$ の正のべき乗はすべて $t = x$ でゼロになり、$k = 0$ の項からは $f(x) - f(x) = 0$ が残るためである）。したがってロルの定理により、ある点 ξ が x_0 と x の真の間に存在して $G'(\xi) = 0$ を満たす。

第 2 に、$G(t)$ を t に関して微分する。積の微分公式により、$k \geq 1$ の各項からは次式が生じる。

$$-\frac{f^{(k+1)}(t)}{k!}(x - t)^k + \frac{f^{(k)}(t)}{(k-1)!}(x - t)^{k-1}$$

これが $G'(t)$ への寄与となり、一方 $k = 0$ の項からは $-f'(t)$ が生じる。隣り合う項どうしが対になって打ち消し合い（望遠鏡和）、最高階の導関数と M の項の導関数だけが残る。

$$G'(t) = -\frac{f^{(n+1)}(t)}{n!}\,(x - t)^n + M\,\frac{(x - t)^n}{n!}.$$

第 3 に、$t = \xi$ で評価する。ξ は x_0 と x の真の間にあるため $(x - \xi)^n \neq 0$ であり、ゆえに $G'(\xi) = 0$ から $M = f^{(n+1)}(\xi)$ が定まる。これを $G(x_0) = 0$ に戻せば Equation 34.5 が得られる。■

この剰余項はテイラー多項式の次の項とまったく同じ形をしているが、ただ $(n+1)$ 階導関数が未知の内点 ξ で評価される点だけが x_0 での評価と異なっている。したがって、$f^{(n+1)}$ が x_0 の近傍で有界である限り、誤差は $|x - x_0|^{n+1}$ のオーダーで変化する。すなわち、一致させる導関数が 1 階増えるごとに近似精度の次数が 1 つ上がるのである。

これを通し例の関数で試してみよう。ソフトプラスでは、$s''' = \sigma(1-\sigma)(1-2\sigma)$ が点 0 でゼロになり、$s''''(0) = -\frac{1}{8}$ となるため、次の四次多項式が得られる。

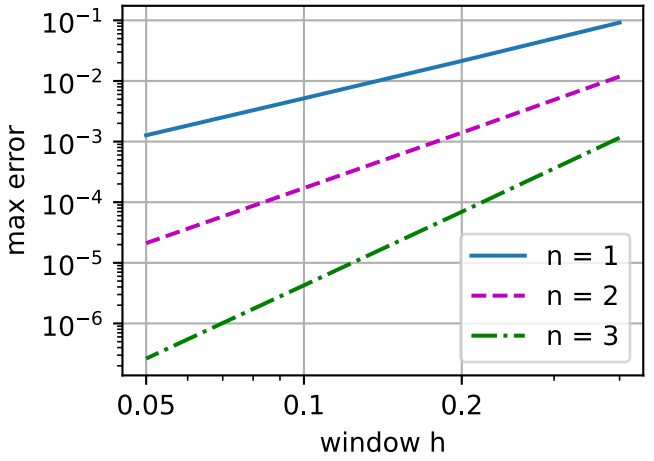
$$P_4(x) = \log 2 + \frac{x}{2} + \frac{x^2}{8} - \frac{x^4}{192}.$$

$x = 0.5$ で評価するとこの値は 0.974072 となり、真の値 $s(0.5) = 0.974077$ に対する誤差はわずか 5×10^{-6} である。実際、この誤差は x^5 による見積もりが示唆するものよりもさらに小さい。なぜなら $s(x) - \frac{x}{2} = \log(2\cosh\frac{x}{2})$ は偶関数であるため、より高次の奇数次の項はすべて消え、次の非ゼロの項は六次になるからである。この四次の項は、先ほどのスクリプトが 0.1249 と出力し 0.1250 にはならなかった理由（$\epsilon = 0.1$ のとき）も説明してくれる。そこで見えていたのは $\frac{1}{8} - \frac{\epsilon^2}{192} = 0.12495$ だったのである。

h^{n+1} のスケーリングはコードで直接確認できる。P_n の区間 $|x| \le h$ 上での最悪誤差が h^{n+1} のように振る舞うならば、h を半分にすると最大誤差は 2^{n+1} で割られる、すなわち接線では 4、放物線では 8、三次多項式では 16 で割られるはずである。この予測を e^x の $x_0 = 0$ において確かめてみよう。

```
# Max error of P_n for e^x on |x| <= h. Lagrange predicts error ~ h^(n+1),
# so halving h should divide it by 2^(n+1)
def taylor_error(n, h, m=1000):
    x = h * np.arange(-m, m + 1) / m
    P = sum(x**k / math.factorial(k) for k in range(n + 1))
    return np.max(np.abs(np.exp(x) - P))

hs = [0.4, 0.2, 0.1, 0.05]
errs = [[taylor_error(n, h) for h in hs] for n in (1, 2, 3)]
for n, e in zip((1, 2, 3), errs):
    ratios = ' '.join(f'{a / b:.1f}' for a, b in zip(e, e[1:]))
    print(f'n = {n}: ratios between successive windows {ratios}, '
          f'predicted {2**(n + 1)}')
d2l.plot(hs, errs, 'window h', 'max error', xscale='log', yscale='log',
         legend=['n = 1', 'n = 2', 'n = 3'])
```

```
n = 1: ratios between successive windows 4.3 4.1 4.1, predicted 4
n = 2: ratios between successive windows 8.4 8.2 8.1, predicted 8
n = 3: ratios between successive windows 16.7 16.3 16.2, predicted 16
```



実測された比は確かに 4、8、16 の近くに並び、両対数プロット上では 3 本の誤差曲線が傾き 2、3、4 の直線として現れ、$n+1$ と一致する。広い窓で比が理論値をわずかに上回るのは、剰余項に含まれる因子 e^{ξ} がそこではより大きくなるためである。

ラグランジュの剰余項は、無限テイラー級数がいつ元の関数自身を再現するかという問いにも答えてくれる。e^x ではすべての導関数が e^x に等しいため、Equation 34.5 を $x_0 = 0$ の周りで適用すると次式が得られる。

$$\left| e^x - P_n(x) \right| = \frac{e^{\xi}\,|x|^{n+1}}{(n+1)!} \le \frac{e^{|x|}\,|x|^{n+1}}{(n+1)!}.$$

任意の固定された x に対し、階乗はいずれべき乗よりも速く増大するため、$n \to \infty$ のとき右辺はゼロに収束する。ゆえに次式が成り立つ。

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$$

これは $\mathbb{R}$ 全体で成立する。ただし注意が必要である。無限回微分可能であっても、関数がそのテイラー級数と一致するとは限らない。$f(x) = e^{-1/x^2}$（$f(0) = 0$ と拡張したもの）では、点 0 においてすべての導関数がゼロになるため、原点におけるテイラー級数は恒等的にゼロとなり、f とは $x = 0$ でしか一致しない（Figure 34.7）。幸いなことに、以降の議論で無限級数に依存する箇所は 1 つもない。扱うのは有限次のテイラー多項式 P_n のみであり、その誤差は Equation 34.5 によって直接抑えられる。

34.1.5.4 降下補題

曲率に上界があれば、x^2 のときと同様に、勾配降下法が任意の滑らかな関数を減少させることは保証されるのだろうか。$n = 1$ のラグランジュの剰余項は、この問いに肯定的な答えを与える。本節のすべての結果のなかで、最適化理論が最も頻繁に用いるのがこの補題である。

命題（降下補題）関数 f が開区間 I（x と $x - \eta f'(x)$ を含む）上で 2 回微分可能であり、$|f''| \le L$ が I 上で成り立つとする。このとき、次式が成り立つ。

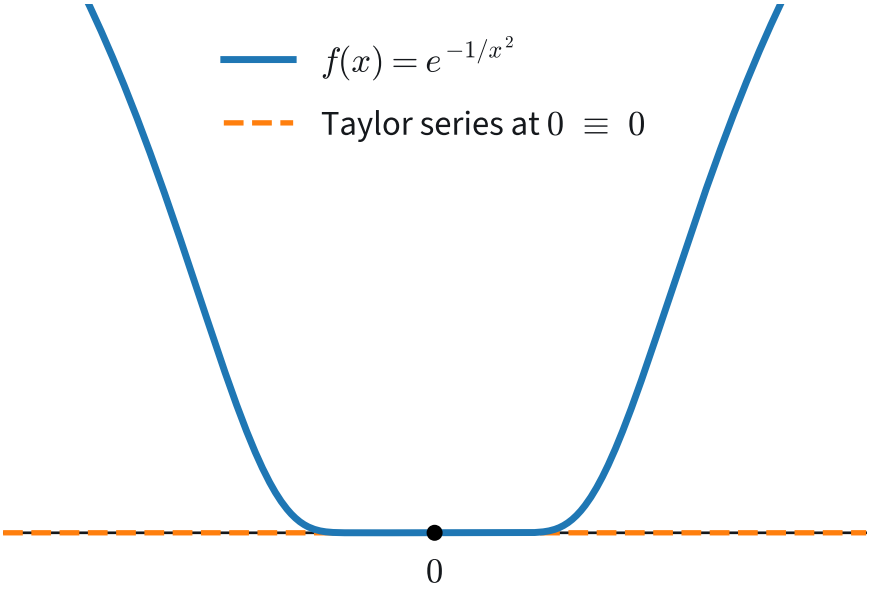


Figure 34.7: 滑らかな関数 $f(x) = e^{-1/x^2}$ は原点ですべての導関数がゼロに等しいため、そこでのテイラー級数は恒等的にゼロとなる。級数はいたるところで収束するが、f とは $x = 0$ でしか一致しない。すなわち、滑らかであることは解析的であることを意味しない。

$$f(x - \eta\, f'(x)) \;\le\; f(x) - \eta\Bigl(1 - \frac{L\eta}{2}\Bigr) f'(x)^2. \tag{34.6}$$

特に、f は $0 < \eta < 2/L$ かつ $f'(x) \neq 0$ である限り真に減少し、保証される減少量は $\eta = 1/L$ で最大となって $f'(x)^2/(2L)$ に等しくなる。

証明：任意のステップ u に対し、$n = 1$ のラグランジュの剰余項から $f(x+u) = f(x) + f'(x)u + \tfrac{1}{2}f''(\xi)u^2 \le f(x) + f'(x)u + \tfrac{L}{2}u^2$ が得られる。$u = -\eta f'(x)$ と置けば、右辺は $f(x) - \eta(1 - L\eta/2)f'(x)^2$ となる。因子 $1 - L\eta/2$ は $\eta < 2/L$ である限り正であり、二次関数 $\eta(1 - L\eta/2)$ は $\eta = 1/L$ で最大値をとる。$\blacksquare$

幾何学的には、この補題は未知の局所曲率 $f''(\xi)$ をその最悪の上界 L に置き換えている。得られる二次関数 $f(x) + f'(x)u + \tfrac{L}{2}u^2$ は、f に点 x で接し区間 I 全体でその上側に留まる上界の放物線として働く。ステップ $\eta = 1/L$ は、この天井の放物線の厳密な最小点（minimizer）へと一足飛びに移動するものである（Figure 34.8）。この上界は滑らかな最適化の主力となる道具である (Nesterov 2018)。実はこの結果は二階導関数さえ必要とせず、f' が L-**Lipschitz** である限り、すなわち任意の 2 点における傾きの差が高々その距離の L 倍（$|f'(u) - f'(v)| \le L|u - v|$）である限り成り立つ。多変数最適化（35.1 節）で用いられるのはこちらの形式であり、演習問題 1 でその 1 変数版を証明する。

よく知られた 2 つの関数でこの補題を確かめてみよう。x^2 では $L = 2$ であるから、$2/L = 1$ となって閉じた式から求めた厳密な安定性のしきい値が復元される。ロジスティック損失では、$\ell''(w) = \sigma(-w)\bigl(1 - \sigma(-w)\bigr) \le \tfrac{1}{4}$（等号成立は $w = 0$）であるから $L = \tfrac{1}{4}$ となり、任意のステップサイズ $\eta < 8$ に対して、どの初期重みからでも ℓ の減少が保証される。$w = 0$ から $\eta = 1$ で出発すると $w_1 = 0.5$ へ移動し、上界は損失が高々 $0.693147 - (1 - \tfrac{1}{8}) \cdot \tfrac{1}{4} = 0.4744$ になる

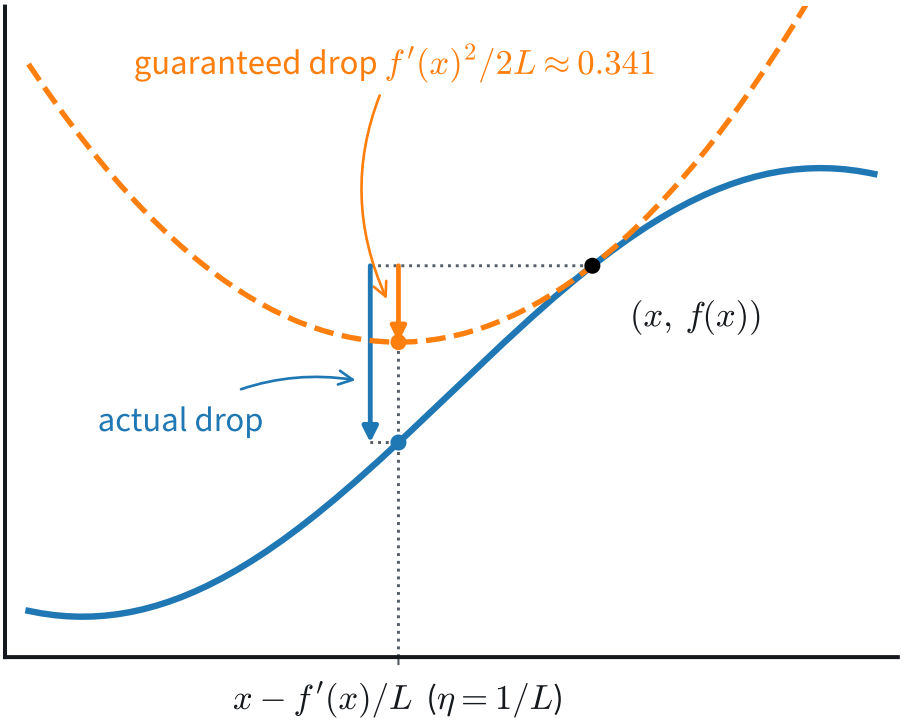


Figure 34.8: $|f''| \le L$ のとき、$f(x) + f'(x)u + \frac{L}{2}u^2$ はステップ u に関する二次の上界となり、基点で f と一致する。$\eta = 1/L$ とした勾配ステップはこの上界を最小化し、少なくとも $\frac{1}{2L}f'(x)^2$ の減少を保証する。ここでは $f(x) = \sin(2x)$（$L = 4$）であり、実際の減少量は最悪ケースの保証を上回っている。

ことを保証するが、実際の値は $\ell(0.5) = 0.4741$ である。この短いステップの間では曲率が最大値 L の近くに留まるため、上界はほぼタイトになっている。より大きなステップ $\eta = 4 = 1/L$ では、上界が約束するのは 0.1931 であるのに対し、実際の損失は $\ell(2) = 0.1269$ まで一気に下がる。原点から離れると曲率が平坦になるため、最悪ケースの天井が予測する以上に曲線が下っていくからである。

34.1.5.5 ニュートン法

降下補題は、大域的な最悪ケースの曲率 L を用いて二次モデルを構築していた。その代わりに現在の反復点における実際の局所曲率 $f''(x_t)$ を用いれば、任意の二次関数を一足飛びに最小化するステップが得られる。x_t における局所モデルは二次のテイラー多項式 P_2 である。

$$q(x) = f(x_t) + f'(x_t)\,(x - x_t) + \tfrac{1}{2} f''(x_t)\,(x - x_t)^2.$$

$f''(x_t) > 0$ である限り、この放物線は上向きに開き、唯一の最小点を持つ。$q'(x) = f'(x_t) + f''(x_t)(x - x_t) = 0$ と置いて x について解くと、**ニュートン法**の更新式が得られる。

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}. \tag{34.7}$$

これは適応的な局所ステップサイズ $1/f''(x_t)$ を持つ勾配ステップであり、上向きに開いた任意の放物線上では 1 ステップで厳密な最小点へ到達する。Figure 34.9 は、$f(x) = x^4/4 - x$（最小点は $x^\star = 1$）における 1 ステップを示している。$x_t = 2$ では $f(2) = 2$、$f'(2) = 7$、$f''(2) = 12$ であるから、放物線の最小点は $x_{t+1} = 2 - 7/12 = 1.4167$ に位置する。

$x^4/4 - x$ 上で $x_0 = 2$ からニュートン法がどれほど速く収束するかを見てみよう。あわせて 2 つ目の関数 $x \arctan x - \tfrac{1}{2}\log(1 + x^2)$ についても、$x_0 = 1.4$ と $x_0 = 1.3$ から出発したときに何が起きるかを試す。この 2 つ目の例については、数値を確認した直後に解析しよう。

```python
# Newton on f(x) = x^4/4 - x: solve f'(x) = x^3 - 1 = 0, root x* = 1
x = 2.0
for t in range(6):
    print(f't = {t}: x = {x:.12f}, error = {abs(x - 1):.1e}')
    x = x - (x**3 - 1) / (3 * x**2)  # x - f'(x) / f''(x)

# Newton on the convex f(x) = x arctan(x) - log(1 + x^2)/2,
# with f' = arctan(x) and f'' = 1/(1 + x^2)
for x0 in [1.4, 1.3]:
    x, xs = x0, []
    for t in range(6):
        x = x - math.atan(x) * (1 + x**2)
        xs.append(x)
    print(f'from x0 = {x0}:', ', '.join(f'{v:.2f}' for v in xs))
```

```
t = 0: x = 2.000000000000, error = 1.0e+00
t = 1: x = 1.416666666667, error = 4.2e-01
t = 2: x = 1.110534409842, error = 1.1e-01
t = 3: x = 1.010636768405, error = 1.1e-02
t = 4: x = 1.000111557304, error = 1.1e-04
t = 5: x = 1.000000012443, error = 1.2e-08
```

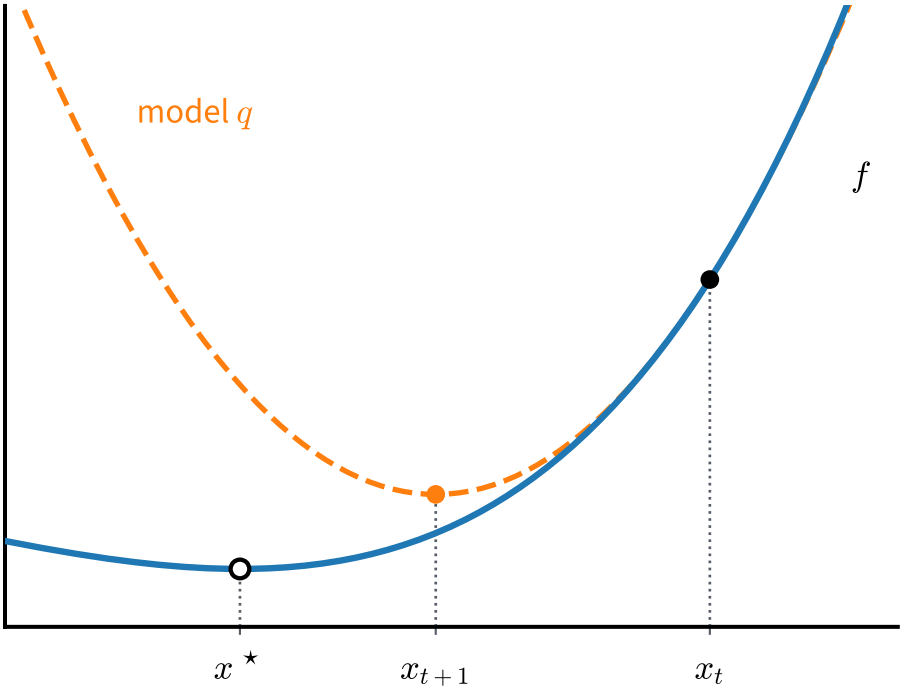


Figure 34.9: $f(x) = x^4/4 - x$ における $x_t = 2$ からの 1 回のニュートンステップ。放物線 q は f の x_t における値、傾き、曲率に一致し、その最小点が次の反復点 $x_{t+1} = x_t - f'(x_t)/f''(x_t) = 2 - 7/12$ となって最小点 $x^\star = 1$ に近づく。

```
from x0 = 1.4: -1.41, 1.45, -1.55, 1.85, -2.89, 8.71
from x0 = 1.3: -1.16, 0.86, -0.37, 0.03, -0.00, 0.00
```

$x^4/4 - x$ 上では、誤差は 1.0、0.42、0.11、1.1×10^{-2}、1.1×10^{-4}、1.2×10^{-8} と急減する。最小点に近づくと、各ステップは直前の誤差を二乗し、有効桁数を約 2 倍に増やしていくのである。対照的に、x^2 上の勾配降下法ではステップごとに定数因子 $|1 - 2\eta|$ 倍ずつしか誤差が縮まらず、反復ごとに一定の桁数しか稼げなかった。

なぜ誤差が二乗されるのかは、ラグランジュの剰余項によって説明できる。$x^\star$ を停留点とし、$e_t = x_t - x^\star$ をステップ t における誤差とする。Equation 34.5 を $n = 1$ として f' に基点 x_t の周りで適用し、$x^\star$ で評価すると、$0 = f'(x_t) - f''(x_t)\, e_t + \tfrac{1}{2} f'''(\xi_t)\, e_t^2$ が得られる。両辺を $f''(x_t) \neq 0$ で割り、ニュートン法の更新式 Equation 34.7 を代入すると、次式が残る。

$$e_{t+1} = \frac{f'''(\xi_t)}{2 f''(x_t)}\, e_t^2 \tag{34.8}$$

ここで ξ_t は x_t と $x^\star$ の間にある何らかの点である（ただし f は 3 回微分可能とする）。$x^4/4 - x$ では、前の係数は $f'''(1)/(2f''(1)) = 6/(2 \cdot 3) = 1$ に収束し、出力された誤差からも実際に $e_{t+1}/e_t^2 \approx 0.42$、$0.64$、$0.87$、$0.99$ が得られる。正の曲率を持つ最小点に十分近い任意の初期点からの二次収束（quadratic convergence）を与える一般の多変数定理は、35.2 節に登場する。

この「十分近い」という但し書きは、たとえ狭義凸関数であっても不可欠である。$f(x) = x \arctan x - \tfrac{1}{2} \log(1 + x^2)$ を考えよう。この関数は $f'(x) = \arctan x$ かつ $f''(x) = 1/(1 + x^2) > 0$ を満たすため狭義凸であり、点 0 に唯一の大域的最小値（global minimum）を持つ。ところが $x_0 = 1.4$ から出発すると、上の 2 つ目のループは $-1.41, 1.45, -1.55, 1.85, -2.89, 8.71$ と出力し、ニュートン法は激しく発散する。わずかに近い $x_0 = 1.3$ から出発すると滑らかに収束し（-1.16、0.86、-0.37、0.03、...）、収束と発散の境目は $x_0 \approx 1.39$ の付近にある。

何が問題だったのかは局所的な放物線が説明してくれる。原点から離れると曲率 $1/(1 + x^2)$ がごく小さくなるため、近似放物線 q はほぼ平坦になり、その最小点を $x^\star$ のはるか向こう側へ置いてしまうのである。標準的な対処法は、実際に f が減少することを保証するよう選んだ係数を掛けてステップを縮小する手法であり、これを**減衰**（damping）と呼ぶ。減衰付きニュートンステップ（Newton step）の構築と、高次元でそれを計算するコストについては 35.2.1 節で検討する。

34.1.6 微分不可能な点

あらゆる ReLU ネットワークは角だらけである。$\mathrm{ReLU}(x) = \max(0, x)$ は点 0 で導関数を持たない。それでも誤差逆伝播法はそこで数値を返す。いったいどの数値を選び、その数値はどのような意味で妥当な傾きなのだろうか。

34.1.6.1 片側導関数

差分商 Equation 34.1 を見れば、問題は直ちに明らかになる。$f(x) = |x|$ の $x = 0$ においては、次のようになる。

$$\frac{|0 + \epsilon| - |0|}{\epsilon} = \frac{|\epsilon|}{\epsilon} = \begin{cases} +1 & \epsilon > 0, \\ -1 & \epsilon < 0. \end{cases}$$

この商は 0 の両側でそれぞれ定数となるが、その 2 つの定数は一致しない。右側から近づくとグラフは傾き $+1$ で上り、左側から近づくと傾き -1 で下る。これらの極限が**片側導関数**（one-sided

derivatives）$f'_+(0) = +1$ および $f'_-(0) = -1$ であり、通常の導関数が存在するのはこの 2 つが一致するときに限られる。ReLU の点 0 でも同じ計算を行うと、平坦な左側では $f'_-(0) = 0$、右側では $f'_+(0) = 1$ となる。これらの角には接線の傾きが存在しないにもかかわらず、自動微分エンジンは何らかの数値を返さなければならない。実際に自動微分が何を返すか確認してみよう。

```
# What autograd returns at the corner, where no derivative exists
for name, fn in [('d|x|/dx', jnp.abs), ("ReLU'", jax.nn.relu)]:
    print(f"{name} at 0: autograd returns {jax.grad(fn)(0.0):.1f}")
```

```
d|x|/dx at 0: autograd returns 1.0
ReLU' at 0: autograd returns 0.0
```

ReLU$'(0)$ に対しては、自動微分は左側の傾きである 0 を返す。$\frac{d}{dx}|x|$ の点 0 における値としては、規約によって定められた固定値を返す。

JAX は右側の傾きである 1 を返す。

34.1.6.2 劣勾配と最適性

角にはただ 1 本の接線は存在しないが、その角でグラフに接し、かつ決してグラフの上側へ出ない直線の族全体が存在する。すべての弦がグラフ上またはその上側に位置する**凸**関数（convex function、35.3 節）においては、これらの支持直線（supporting line）が接線の役割を担う。実数 g が凸関数 f の点 x における**劣勾配**（subgradient）であるとは、次式が成り立つことをいう。

$$f(y) \geq f(x) + g\,(y - x) \quad \text{for all } y, \tag{34.9}$$

これは、$(x, f(x))$ を通り傾き g を持つ直線が、いたるところで f のグラフ上またはその下側にあることを意味する。点 x におけるすべての劣勾配からなる集合を**劣微分**（subdifferential）といい、$\partial f(x)$ と書く (Rockafellar 1970)。f が微分可能な任意の点では、$\partial f(x)$ はただ 1 つの数 $f'(x)$ のみを含む。一方、角においては両側の片側導関数を結ぶ閉区間へと広がり、Figure 34.10 に示す扇状の支持直線群に対応する。

$$\partial |x|(0) = [-1, 1], \qquad \partial\,\mathrm{ReLU}(0) = [0, 1].$$

先ほど自動微分が返した値はすべて、対応する区間のなかに収まっている。

通し例の関数を見れば、この区間がどこから現れるのかが直観的にわかる。鋭くしたソフトプラス $\mathrm{softplus}_\beta(x) = \frac{1}{\beta} s(\beta x)$ は $\mathrm{ReLU}(x)$ へと（$\beta \to \infty$ のとき）収束し、その導関数は $\sigma(\beta x)$ である（Figure 34.11）。$x = \pm 0.1$ における傾きは、0.40 と 0.60（$\beta = 4$ のとき）から、0.17 と 0.83（$\beta = 16$ のとき）へと急峻になる。劣微分 $[0, 1]$ は、$\beta \to \infty$ と $x \to 0$ を同時にとったときにこれらの滑らかな近似の傾きが近づいていく集合にほかならない。

劣勾配からは明快な最適性条件も得られる。$g = 0$ を Equation 34.9 に代入すると $f(y) \geq f(x)$ がすべての y に対して成り立つため、点 x が凸関数 f の大域的最小点となるための必要十分条件は次式で与えられる。

$$0 \in \partial f(x). \tag{34.10}$$

この同値性の両方向は、その 1 行の式から直ちに従う。$|x|$ の場合、その点に導関数が存在しなくても、$0 \in [-1, 1]$ であることから原点が唯一の大域的最小値を与える点だと保証される。対照的

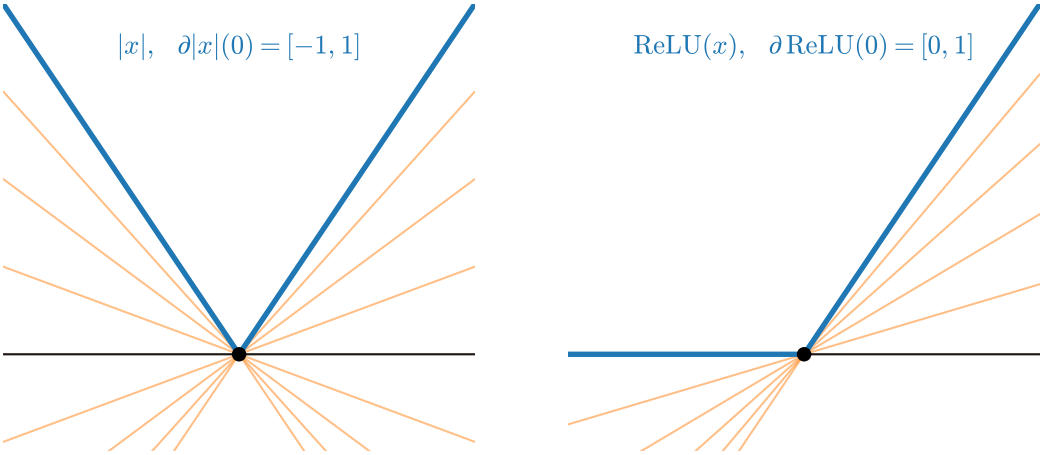


Figure 34.10: 2 つの角とその劣微分。$|x|$（左）と ReLU（右）の原点を通る扇状の支持直線群であり、その傾きはそれぞれ $\partial|x|(0)$ と $\partial\,\mathrm{ReLU}(0)$ にわたって動く。角以外の点では唯一の接線が存在し、劣微分は通常の導関数へと縮退する。

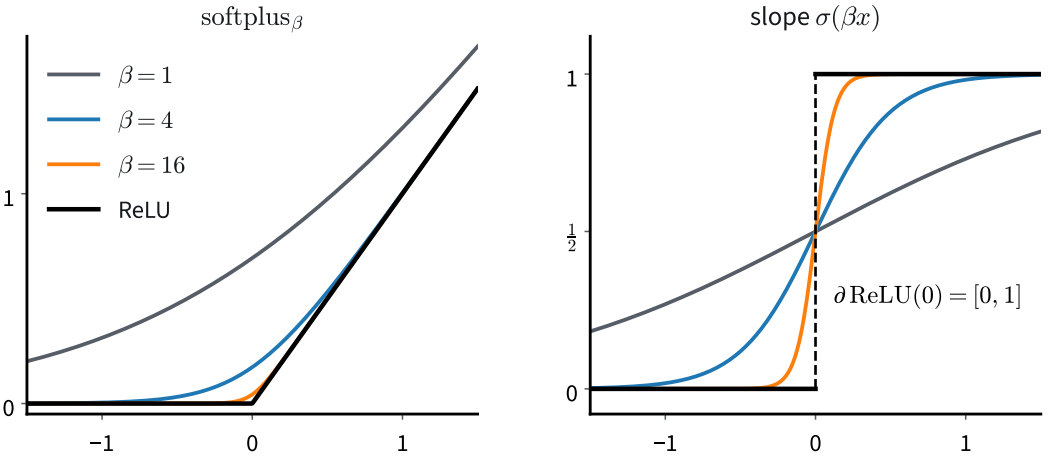


Figure 34.11: 角の平滑化。左：$\mathrm{softplus}_\beta(x) = \frac{1}{\beta}\log(1 + e^{\beta x})$ は ReLU へと、β が大きくなるにつれて近づく。右：その傾き $\sigma(\beta x)$ は 0 から 1 への階段関数へと急峻になり、$x = 0$ の近傍では滑らかな近傍点での傾きが区間 $[0, 1]$、すなわち角における ReLU の劣微分を埋め尽くす。

に、凸性（convexity）がなければ $f'(x) = 0$ は必要条件にすぎない（フェルマーの補題）。実際、x^3 は $f'(0) = 0$ を満たすが最小値を一切持たない。

劣勾配は 35.3 節で多変数へと拡張するが、ここで 1 つ注意を述べておこう。真の勾配ステップとは異なり、劣勾配ステップは関数値を減少させる保証がない。$|x|$ の最小点そのものにおいて、実数 $g = \frac{1}{2}$ は妥当な劣勾配であるが、$x = -\eta/2$ へ進むと実際には $|x|$ が増加してしまう。そのため劣勾配法（subgradient method）は、毎ステップの前進を保証するのではなく、ステップサイズを徐々に縮小させることで収束を達成する (Nesterov 2018)。

34.1.6.3 ネットワーク内の折れ点

ニューラルネットワーク内の各折れ点（kink）は、$|x|$、ReLU、ヒンジ損失（hinge loss）といった単純な凸の基本要素に由来する。折れ点において、自動微分はその要素の劣微分から固定された元を 1 つ選び（たとえば $\mathrm{ReLU}'(0) = 0$）、連鎖律を用いて周囲の合成関数へと伝播させる。残念ながら、個々の部品の妥当な劣勾配を連鎖律でつなぎ合わせると、合成関数自体の劣勾配ではまったくない数値が生じることがある。最も単純な反例は、恒等関数を回りくどく書き表した次式である。

$$\varphi(x) = \mathrm{ReLU}(x) - \mathrm{ReLU}(-x) = x \quad \text{for all } x.$$

いたるところで $\varphi(x) = x$ であるから、点 0 における真の傾きは 1 しかない。ところが連鎖律を機械的に適用すると $\varphi'(x) = \mathrm{ReLU}'(x) + \mathrm{ReLU}'(-x)$ となり、これはほぼいたるところで 1 に等しいものの、$x = 0$ だけは例外であり、規約 $\mathrm{ReLU}'(0) = 0$ によって $0 + 0 = 0$ となってしまう。自動微分が実際にこのように振る舞うことを確かめてみよう。

```
# phi(x) = relu(x) - relu(-x) is the identity, so the true slope at 0 is 1
def phi(x):
    return jax.nn.relu(x) - jax.nn.relu(-x)

print(f"autograd: phi'(0) = {jax.grad(phi)(0.0):.1f}  (true slope: 1.0)")
```

```
autograd: phi'(0) = 0.0  (true slope: 1.0)
```

予測通り、自動微分は恒等関数に対して傾き 0 を返す。

このような誤差が引き起こされるには、活性化前の値が厳密に 0 に着地する必要がある。厳密な実数演算では、データと初期重みが連続分布から抽出されるとき、任意の固定されたステップでこれが起きる確率はゼロである。しかし有限精度の浮動小数点（floating-point）演算では、厳密なゼロが実際に生じる。Bertoin et al. (2021) は、小さな全結合ネットワーク上でこれがどの程度の頻度で起きるかを測定した。ここでは、出版版の実験におけるバグを修正して結論の一部を覆した著者らの 2023 年の正誤表（論文の改訂 arXiv 版）の数値を引用する。$\mathrm{ReLU}'(0) = 0$ と $\mathrm{ReLU}'(0) = 1$ による誤差逆伝播法は、少なくとも 1 つのミニバッチで異なる勾配を生成した割合が、16 ビット精度ではすべての初期化（initialization）、32 ビット精度では全初期化の半分弱、64 ビット精度では皆無であった。

それらの不一致が訓練に影響を与えるかどうかは別の問題であり、その答えは安心できるものである。実用上はまったく影響しない。32 ビット演算では、影響を受けるのは 100 万個のミニバッチのうちわずか数個にすぎず、そのミニバッチにおいてさえ 2 つの勾配の相対差は約 10^{-3} にとどまる。修正後の実験では、$\mathrm{ReLU}'(0)$ を $[0, 1]$ 内のどの値に選んでも実質的に同一の訓練損失（training loss）とテスト正解率が得られている。訓練が劣化するのは $\mathrm{ReLU}'(0) = 10^4$ のような極端な選択をした場合だけであり、バッチ正規化（batch normalization）や Adam を用いれば

それさえも抑え込まれる。したがって、この規約は実行間のわずかな数値的差異を生む実際の要因ではあるものの、無害なものである。演習問題 5 では、この現象を小規模な設定で調べる。

本節で扱ったのはすべて単一のスカラー重みであった。ニューラルネットワークでは何百万もの重みが連動して動く。どの同時方向が損失を下げるのか、そして高次元における曲率とは何を意味するのかが、続く 34.2 節の主題である。

34.1.7 まとめ

導関数 $f'(x)$ は、線形近似 $f(x+\epsilon) = f(x) + \epsilon f'(x) + o(\epsilon)$ の傾きである。この近似から証明される和・積・連鎖律によって、初等関数の短い表から組み立てられる任意の合成関数（ソフトプラスにおける $s' = \sigma$ など）を微分できる。単一の点における傾きから区間上の保証へと進むために用いるのが平均値の定理とその高次への拡張であるラグランジュの剰余項であり、次数 n のテイラー多項式の誤差を $|x - x_0|^{n+1}$ の定数倍で抑える（$f^{(n+1)}$ が有界である限り）。

次数 $n = 1$ で適用すると、ラグランジュの剰余項から降下補題が導かれる。すなわち、ステップ区間上で $|f''| \le L$ かつ $f'(x) \neq 0$ であれば、$0 < \eta < 2/L$ を満たすすべての勾配ステップは f を真に減少させる。大域的な上界 L を局所曲率 $f''(x_t)$ に置き換えればニュートンステップ $-f'/f''$ が得られ、$f'' > 0$ を満たす最小点の近傍では誤差を二乗するが、最小点から遠いとたとえ狭義凸関数上であっても発散し得る。最後に、凸関数の微分不可能な角においては劣微分 $\partial f(x)$ が導関数に取って代わり、$0 \in \partial f(x)$ が大域的最小点を特徴づける。自動微分は折れ点ごとに 1 つの元を選ぶが、浮動小数点演算においてそれらの選択の連鎖が真の劣勾配を与え損ねることがあっても、訓練への影響は無視できるほど小さい。

34.1.8 演習問題

1. **ロジスティック損失の滑らかさ**。降下補題 Equation 34.6 には曲率の上界 L が必要である。
   1. 平均値の定理 Equation 34.3 を用いて、ある区間上で $|f''| \le L$ ならばその区間上で f' が L-Lipschitz であることを示せ。逆に、定義 Equation 34.1 から、2 回微分可能な f が L-Lipschitz な f' を持つならば $|f''| \le L$ を満たすことを示せ。
   2. 1 つの重み w、入力 $a_i \in \mathbb{R}$、ラベル $y_i \in \{-1, 1\}$ を持つロジスティック回帰は、次の関数を最小化する。

$$\ell(w) = \frac{1}{n} \sum_{i=1}^{n} \log\!\left(1 + e^{-y_i a_i w}\right). \tag{34.11}$$

      ℓ'' を σ を用いて表せ。最小の L（ℓ' が L-Lipschitz に $\mathbb{R}$ 全体でなるもの）、およびすべての初期点からの減少を Equation 34.6 が保証するステップサイズを求めよ。
   3. f が微分可能であることのみを仮定し、ステップの区間上で f' が L-Lipschitz である場合に降下補題を証明せよ。$u = -\eta f'(x)$ に対し $\psi(t) = f(x+tu) - f(x) - t f'(x) u - \tfrac{L}{2} t^2 u^2$ と置く。$\psi(0) = 0$ および $\psi'(t) = u\,(f'(x+tu) - f'(x)) - L t u^2 \le 0$ が $t \in [0,1]$ で成り立つことを示し、平均値の定理から $\psi(1) \le 0$ を導け。これは多変数関数の最適化で用いられる補題の形式である。
2. **エントロピーの極値**。表が確率 p で出るコインの二値エントロピーは次式で与えられる。

$$H(p) = -p \log p - (1-p) \log(1-p), \qquad 0 < p < 1. \tag{34.12}$$

1. $f(x) = x \log x$ の $x \geq 0$ における最小値を求めよ（ただし $f(0) = 0$ は極限値とする）。最小点が境界 $x = 0$ に位置しないのはなぜか。
2. $H'(p) = -\log \frac{p}{1-p}$、すなわち p の負のロジットとなることを示せ。H の停留点を求め、二階微分判定法によって分類せよ。
3. $H'(p)$ の $p \to 0$ における極限を求めよ。次にコインをそのロジット z でパラメータ化して $p = \sigma(z)$ と置き、$\frac{d}{dz} H(\sigma(z))$ を計算せよ。2 つのパラメータ化のうち、定義域全体で H の傾きが Lipschitz となり、Equation 34.6 によってすべての初期点に対して $-H$ 上の勾配降下法に共通のステップサイズが保証されるのはどちらか。

3. **退化した最小点におけるニュートン法。** $p > 1$ に対し、凸関数 $f_p(x) = |x|^p / p$ は $x = 0$ に唯一の最小点を持つ。

1. f_p が点 0 を含むいたるところで微分可能であって $f_p'(x) = \operatorname{sign}(x)\,|x|^{p-1}$ を満たすことを示し、$f_p''(x)$ を $x \neq 0$ に対して計算せよ。どの p に対して $f_p''(0)$ は存在し、その値は何か。
2. ニュートン法 Equation 34.7 がすべての $x_t \neq 0$ を $x_{t+1} = \frac{p-2}{p-1}\, x_t$ に写すことを示せ。どの p に対してこの反復は収束するか。また、$p = 4$ ではどれほどの速さで収束するか。
3. 34.1.5.5 節における局所収束の命題の仮定のうち、$p = 4/3$ で成り立たないものはどれか。また、$p = 4$ で成り立たないものはどれか。
4. $p = 4/3$ において、減衰付き更新式 $x_{t+1} = x_t - \alpha f_p'(x_t) / f_p''(x_t)$ はニュートンステップを係数 $\alpha > 0$ だけ短縮する。どの α に対してすべての x_0 から収束し、その収束率はどのようになるか。

MIT 18.01SC、Session 33 演習問題「Cube Root of x」より改題。

4. [code] **折れ点にある最小点。** 1 つの重み x、重み減衰 $\lambda > 0$、およびラベルと入力の積が 1 に等しい 1 つの訓練例を持つサポートベクトルマシンは、次の関数を最小化する。

$$F(x) = \max(0, 1 - x) + \frac{\lambda}{2} x^2. \tag{34.13}$$

1. ヒンジ損失 $\max(0, 1-x)$ のすべての x における劣微分を Equation 34.9 から計算せよ。
2. 微分可能な凸関数 q を凸関数 h に加えるとすべての劣勾配に q' が加わること、すなわち $\partial(q + h)(x) = q'(x) + \partial h(x)$ は既知としてよい。最適性条件 Equation 34.10 を用いて、関数 F の最小点をすべての $\lambda > 0$ に対して求めよ。最小点が折れ点に位置するのはどの λ のときか。
3. $\lambda = 1/2$ とし、劣勾配法 $x_{t+1} = x_t - \eta_t g_t$ を $x_0 = 0$ から 10^4 ステップ実行せよ。ただし g_t は、ヒンジ損失を $F'(x_t)$ の計算において $\mathrm{ReLU}(1 - x)$ と書いたときに自動微分が返す値とする。定数ステップ $\eta_t = 0.1$ および $\eta_t = 0.01$ を、減衰ステップ $\eta_t = 0.5/\sqrt{t+1}$ と比較せよ。最小点までの最終的な距離はスケジュールにどう依存するか。
4. 折れ点 $\{x = 1\}$ は単一の点であり、確率密度からの抽出がそこに当たる確率はゼロである。(b) と (c) の結果は、訓練が折れ点に出会うことはまれであるという 34.1.6.3 節の議論に対して何を示唆するか。

5. [code] **連鎖させた劣勾配。** 自動微分は $\mathrm{ReLU}'(0) = 0$ を返し、34.1.6.3 節で見たように、あらゆる式にわたってこの値を連鎖律で伝播させる。

1. f_1 と f_2 を凸関数とし、$g_1 \in \partial f_1(x)$、$g_2 \in \partial f_2(x)$、$c_1, c_2 \geq 0$ とする。Equation 34.9 から $c_1 g_1 + c_2 g_2 \in \partial(c_1 f_1 + c_2 f_2)(x)$ を示せ。この議論が $\mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$ には適用できないのはなぜか。

2. すべての実数 c に対して次式が成り立つことを示せ。

$$k_c(x) = \mathrm{ReLU}(x) + \mathrm{ReLU}(-x) + c\,(x - \mathrm{ReLU}(x) + \mathrm{ReLU}(-x)) = |x|. \tag{34.14}$$

自動微分が $k_c'(0)$ に対して返す値を予測し、$c = 0$、1、5 についてその予測を確かめよ。どの c に対して、返された値は $|x|$ の点 0 における劣勾配となるか。

3. 自動微分から得られる導関数を用いた勾配降下法を $\varphi(x) = \mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$ 上で 10 ステップ、$x_0 = 0.5$ から、$\eta = 0.125$ の場合と $\eta = 0.1$ の場合についてそれぞれ実行せよ。各実行では何が起きるか。2 つのステップサイズで振る舞いが異なるのはなぜか。また、この違いは浮動小数点演算における 34.1.6.3 節の確率ゼロの議論に対して何を示唆するか。

6. **四次関数上の降下法。**●$f(x) = x^4/4$ の最小点は $f''(0) = 0$ であるため平坦である。勾配降下法 Equation 34.4 を f に適用すると $x_{t+1} = x_t - \eta x_t^3$ となる。

1. 問題 1a を用いて、最小の L（f' が L-Lipschitz に $[-R, R]$ 上でなるもの）を求めよ。どの初期点 x_0 が、与えられた η に対して降下補題 Equation 34.6 の適用対象となるか。固定された η ではすべての初期点を網羅できないのはなぜか。
2. 反復点列は 0 に $\eta x_0^2 < 2$ のとき収束し、$\eta x_0^2 > 2$ のとき発散することを示せ。$\eta x_0^2 = 2$ では何が起きるか。このしきい値を (a) で対象となった初期点と比較せよ。
3. $1/x_{t+1}^2 = 1/x_t^2 + 2\eta + O(x_t^2)$ を示し、$x_t^2 \approx 1/(x_0^{-2} + 2\eta t)$ を導け。この予測を $\eta = 0.1$、$x_0 = 1$ について $t = 10$、100、1000 で確かめよ。この収束率は、x^2 上の勾配降下法や、$x^4/4$ 上のニュートン法（問題 3b）と比べるとどうか。

7. [code] **中心差分。** ●前進差分 $D_+ f(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}$ と中心差分 $D_0 f(x) = \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$ はいずれも $f'(x)$ を近似する。

1. ラグランジュの剰余項 Equation 34.5 を用いて、$D_+ f(x) - f'(x) = \frac{1}{2} f''(\xi)\,\epsilon$ および $D_0 f(x) - f'(x) = \frac{1}{6} f'''(\xi')\,\epsilon^2$ がある ξ、ξ'（x の近傍）に対して成り立つことを示せ。ただし f は 3 回連続微分可能とする（f''' の連続性により 2 つの剰余項の点が 1 つにまとめられる）。$f(x \pm \epsilon)$ を必要な次数まで展開して差をとれ。
2. $f(x) = x^2 + 1701(x-4)^3$ の $x = 4$ において、2 つの誤差がそれぞれ $\epsilon + 1701\epsilon^2$ と $1701\epsilon^2$ であることを示せ。$\epsilon = 0.1$ ではそれぞれ 17.11 と 17.01 となり、ほぼ等しい。中心差分は誤差の次数を引き上げるが、それが効力を発揮するのは ϵ が小さくなってからである。中心差分の誤差が前進差分の誤差の 100 分の 1 になるのはどの ϵ のときか。

浮動小数点演算では、丸め誤差によって ϵ に下限が生じる。34.3 節の勾配チェックの演習問題では、これと (a) の打ち切り誤差との釣り合いをとる。

8. [code] **$2/L$ を超える勾配降下法。**●●$f(x) = x^2$ 上では、ステップサイズ $\eta > 2/L$ はすべて発散する。しかし他の関数では、このしきい値を超えた勾配降下法が発散する代わりに振動へと落ち着くことがあり (Chen and Bruna 2023)、これは安定性の縁の 1 次元版にあたる (Cohen et al. 2021)。$x^\star$ を 4 回連続微分可能な関数 f の極小点（$f''(x^\star) > 0$）とし、$T(x) = x - \eta f'(x)$ を Equation 34.4 の 1 ステップとする。

1. Equation 34.3 を用いて、$x^\star$ に十分近い点から出発した反復点列は $\eta < 2/f''(x^\star)$ のときそれに収束することを示せ。また、$\eta > 2/f''(x^\star)$ のとき、$x^\star$ のある近傍内の $x^\star$ 以外のすべての反復点について、次の反復点が $x^\star$ からより遠ざかることを示せ。

2. 周期 2 の軌道とは、$a \neq b$ かつ $T(a) = b$、$T(b) = a$ を満たす対である。任意の周期 2 の軌道に対し、ある点 ξ が a と b の真の間に存在して $f''(\xi) = 2/\eta$ を満たすことを示せ。

3. $\eta = (1 + \delta)\, 2/f''(x^\star)$（ただし $\delta > 0$ は微小）と置く。$T(T(x))$ を $x^\star$ の周りで 3 次まで展開し、周期 2 の軌道が $x^\star$ の近傍に存在して近傍の反復点を引き寄せるための条件が次式であることを示せ。

$$3 f'''(x^\star)^2 - f''(x^\star)\, f''''(x^\star) > 0. \tag{34.15}$$

   軌道の半幅 $|a - b|/2$ を δ の主要項まで導出せよ。

4. 二重井戸関数 $f(x) = \tfrac{1}{4}(x^2 - 1)^2$ 上で $x_0 = 0.5$ から勾配降下法を 5000 ステップ、各ステップサイズ $\eta \in [0.9, 1.5]$ について実行し、最後の 100 個の反復点を η に対してプロットせよ。$\eta = 1$ 近傍における軌道の半幅を (c) と比較せよ。さらに、安定な周期 2 の軌道が存在する η の範囲を、$1 < \eta < \sqrt{4.5} - 1$（この範囲では Chen and Bruna (2023) がほとんどすべての $0 < x_0 < 1$ から軌道への収束を証明している）と比較せよ。

5. (c) を用いて、$\eta = 2.02$ の勾配降下法が $\log\cosh x$ および $x^2/2 + x^4/4$（いずれも $x^\star = 0$ かつ $f''(x^\star) = 1$ を持つ）上でどのように振る舞うかを予測せよ。両方について $x_0 = 0.01$ から 500 ステップ実行し、予測を確かめよ。

34.2 多変数の微分積分

前節では、他のすべての重みを固定したまま、単一の重みの関数として損失を調べた。しかし実際の訓練はそのようには進まない。勾配降下法の各ステップはすべての重みを一度に少しずつ動かすため、同時移動に対して損失がどう応答するか、そしてどの同時移動が損失を最も速く減少させるかを知る必要がある。重みごとに 1 つずつ、100 万個の別々の傾きを持っているだけでは、そのどちらの問いにも答えられない。

ところが、1 つの穏やかな仮定のもとでは、それらの傾きが両方の問いに答えてくれる。損失が微分可能であるとき、**勾配**（gradient）と呼ばれる傾きのベクトルは、あらゆる微小な同時移動の影響を 1 次の範囲で一括して予測し、最も急激に増加する方向を指し示す。さらに踏み込んで極小点（local minimum）と鞍点（saddle point）を見分けるには、曲面がどのように曲がっているかを測る二階導関数の対称な表、すなわち**ヘッセ行列**（Hessian）が必要になる。本節では、これら 2 つの対象が多変数関数について何を教えてくれるかを展開する。大規模なネットワークに対してプログラムが実際にどうやって勾配を計算するかは別の問題であり、続く 34.3 節で取り上げる。

1.4 節ではすでに微分公式を述べ、勾配を実際に活用した。本節では、34.1 節が 1 変数について行ったのと同様に、その数学的基礎を与える。すなわち、線形近似がいつ成り立つか、そして二階導関数がどのように臨界点を分類するかを証明する。本節全体の柱となるのは、多変数の線形モデルと二階微分判定法（second-derivative test）という 2 つの結果である。接平面に関する短い項と末尾のラグランジュ乗数に関する内容は、初読時には飛ばしても差し支えない。本節を通して、各概念を共通の例題で検証していく。2 変数の log-sum-exp 関数 $f(x, y) = \log(e^x + e^y)$ を点 $(0, \log 2)$ で評価したものである。その勾配は、おなじみのソフトマックスになることがわかるだろう。

34.2.1 偏微分と勾配

34.2.1.1 偏微分

w_1 以外のすべての重みを固定すると、損失は 1 変数の関数になる。その通常の導関数が、1.4 節で導入した**偏微分**（partial derivative）$\partial \ell / \partial w_1$ である。幾何学的には、これは ℓ のグラフにおいて w_1 のみを動かして得られる 1 次元スライスの傾きを表す。1 変数の微分積分と同様に、この傾きは微小な変化 ϵ_1 を w_1 単独に加えたときの影響を次のように予測する。

$$\ell(w_1 + \epsilon_1, w_2, \dots, w_N) \approx \ell(w_1, w_2, \dots, w_N) + \epsilon_1 \frac{\partial \ell}{\partial w_1}(w_1, w_2, \dots, w_N). \tag{34.16}$$

例題である $f(x, y) = \log(e^x + e^y)$ の偏微分を計算してみよう。y を固定して x に関して微分すると $\partial f / \partial x = e^x / (e^x + e^y)$ が得られ、対称性から $\partial f / \partial y = e^y / (e^x + e^y)$ となる。点 $(0, \log 2)$ では $e^x = 1$ および $e^y = 2$ であるから、$f(0, \log 2) = \log 3 \approx 1.0986$、$\partial f / \partial x = \frac{1}{3}$、$\partial f / \partial y = \frac{2}{3}$ となる。言い換えれば、y だけをわずかに動かすと、f は x だけを動かしたときの 2 倍の速さで増加する。

34.2.1.2 微分可能性と勾配

偏微分が教えてくれるのは、各座標軸に沿った 1 方向ずつの傾きである。すべての座標が同時に動くとき、たとえば $(0, \log 2)$ から $\epsilon = (0.01, -0.03)$ だけステップを進めるときに何が起きるかは、偏微分からは自動的にはわからない。座標ごとの変化を単純に足し合わせて $\frac{1}{3} \cdot 0.01 + \frac{2}{3} \cdot (-0.03)$ になると推測したくなるかもしれない。しかしその推測は、実はその点の近傍における f の振る舞いについての強い主張であり、成り立つにはある仮定が必要である。

その仮定とは、$\mathbf{w}$ の近傍において、関数 ℓ がステップサイズより速く消える誤差を除いて線形に見えるというものである。形式的には、ℓ が点 $\mathbf{w}$ で**微分可能**（differentiable）であるとは、次の式を満たす線形写像 Λ が存在することをいう。

$$\ell(\mathbf{w} + \epsilon) = \ell(\mathbf{w}) + \Lambda(\epsilon) + o(\|\epsilon\|), \tag{34.17}$$

ここで $o(\|\epsilon\|)$ は、$\|\epsilon\|$ で割ったときに $\epsilon \to \mathbf{0}$ の極限でゼロに収束する任意の剰余項を表す。

そのような線形写像 Λ が存在するとき、偏微分によってそれは完全に決定される。$\epsilon = h\mathbf{e}_i$ を Equation 34.17 に代入し、両辺を h で割って $h \to 0$ とすると、$\Lambda(\mathbf{e}_i) = \partial \ell / \partial w_i$ が導かれる。Λ は線形であるため、任意の同時ステップ ϵ に対する作用は次のように定まる。

$$\Lambda(\epsilon) = \Lambda\Big(\sum_{i=1}^{N} \epsilon_i \mathbf{e}_i\Big) = \sum_{i=1}^{N} \epsilon_i \frac{\partial \ell}{\partial w_i} = \epsilon \cdot \nabla_{\mathbf{w}} \ell, \qquad \nabla_{\mathbf{w}} \ell = \left[\frac{\partial \ell}{\partial w_1}, \dots, \frac{\partial \ell}{\partial w_N}\right]^{\top}.$$

偏微分を並べたこの列ベクトル $\nabla_{\mathbf{w}} \ell$ を ℓ の**勾配**と呼ぶ。これを Equation 34.17 に代入すると、多変数の線形近似が得られる。

$$\ell(\mathbf{w} + \epsilon) = \ell(\mathbf{w}) + \epsilon \cdot \nabla_{\mathbf{w}} \ell(\mathbf{w}) + o(\|\epsilon\|). \tag{34.18}$$

これを 1 変数の近似式 $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$ と比べてみよう。スカラーの傾きがベクトルに、通常の積がドット積に置き換わっただけで、式の形はまったく同じである。勾配とは、ステッ

プとのドット積が ℓ の 1 次変化を与える唯一のベクトルである。勾配の捉え方はいくつもあるが、最も心に留めておくべきなのはこの見方である。本書を通じて、勾配は列ベクトルとして扱う。

点 $(0, \log 2)$ における例題に戻ると、$\nabla f = [\frac{1}{3}, \frac{2}{3}]^\top$ であった。$\epsilon = (0.01, -0.03)$ を Equation 34.18 に代入すると、$f(0.01, \log 2 - 0.03) \approx \log 3 + \frac{1}{3} \cdot 0.01 - \frac{2}{3} \cdot 0.03 \approx 1.081946$ と予測される。なお、**あらゆる**点 (x, y) において、∇f の 2 つの成分は正であり、和が 1 になることに注目してほしい。すなわち log-sum-exp の勾配はソフトマックス $\mathbf{p} = \mathrm{softmax}(x, y)$ であり、その成分は $e^x/(e^x + e^y)$ と $e^y/(e^x + e^y)$ である。

ここで一度立ち止まり、なぜ微分可能性が単なる偏微分の存在以上のものを要求するのかを確認しておこう。$f(x, y) = 2xy/(x^2 + y^2)$（ただし $f(0, 0) = 0$）を考える。両方の座標軸上では f は恒等的にゼロであるため、原点における両偏微分はいずれも 0 となり、候補となり得る線形モデルは定数関数 0 しかない。ところが対角線 $x = y \neq 0$ に沿うと、原点にどれほど近づいても $f(x, x) = 1$ のままであるため、原点ではいかなる線形近似も成り立たない。この例では f は原点で連続ですらないが、演習問題 1 では、連続関数であってすべての偏微分が存在していてもなお微分可能でない場合があることを示す。

幸いなことに、Equation 34.17 を直接確かめなければならない場面はめったにない。使いやすい十分条件として、$\mathbf{w}$ のある近傍ですべての偏微分が存在し、かつ点 $\mathbf{w}$ で連続であれば微分可能となる (Rudin 1976)。log-sum-exp の偏微分はいたるところで連続であるため、log-sum-exp はいたるところで微分可能である。

微分可能性の定義が約束するのは、近似誤差が $o(\|\epsilon\|)$ であること、つまり $\|\epsilon\|$ より速く縮むことだけである。実際にはどれほど速く縮むのだろうか。次のセルを実行する前に、ステップ $\epsilon = (0.01, -0.03)$ を 10 分の 1 に縮めたとき、線形モデルの誤差が何分の 1 になるか予想してみてほしい。以下の実験では、通常の NumPy を使ってこれを確かめる。

```
import numpy as np
import jax
from jax import numpy as jnp
```

```
def lse(xy):  # f(x, y) = log(e^x + e^y), computed stably
    return np.logaddexp(xy[0], xy[1])

def grad_lse(xy):  # its gradient, the softmax of (x, y)
    return np.exp(xy - lse(xy))

xy0 = np.array([0.0, np.log(2)])
print('gradient:', grad_lse(xy0))
for scale in [1, 0.1, 0.01]:
    eps = scale * np.array([0.01, -0.03])
    linear = lse(xy0) + eps @ grad_lse(xy0)
    true = lse(xy0 + eps)
    print(f'|eps| {np.linalg.norm(eps):.1e}: linear {linear:.8f}, '
          f'true {true:.8f}, gap {true - linear:.3e}')
```

```
gradient: [0.33333333 0.66666667]
|eps| 3.2e-02: linear 1.08194562, true 1.08212418, gap 1.786e-04
|eps| 3.2e-03: linear 1.09694562, true 1.09694740, gap 1.779e-06
|eps| 3.2e-04: linear 1.09844562, true 1.09844564, gap 1.778e-08
```

$\|\epsilon\| \approx 0.03$ では、線形モデルと真の値とのずれは 1.79×10^{-4}（小数第 4 位の差）にすぎず、ス

テップを 10 分の 1 に縮めるたびにその誤差は 100 分の 1 に縮小する。つまり誤差はステップに関して 2 次であり、微分可能性単独で保証されるものよりはるかに精度が高い。まもなく見るように、ヘッセ行列を使えばこの誤差を正確に予測できる（34.2.3.2 節）。

34.2.1.3 方向微分

ここまでは微小ステップに対する変化を予測するために勾配を使ってきたが、勾配は座標軸方向だけでなく、任意の方向に沿った厳密な瞬間的傾きも与える。任意の単位ベクトル $\mathbf{u}$ をとり、$\epsilon = h\mathbf{u}$ を Equation 34.18 に代入して h で割り、$h \to 0$ とする。すると剰余項 $o(\|h\mathbf{u}\|)$ は消え、次式が残る。

$$\lim_{h \to 0} \frac{\ell(\mathbf{w} + h\mathbf{u}) - \ell(\mathbf{w})}{h} = \mathbf{u} \cdot \nabla_{\mathbf{w}} \ell(\mathbf{w}). \tag{34.19}$$

左辺の極限は ℓ の $\mathbf{u}$ に沿った**方向微分**（directional derivative）であり、右辺はそれが勾配の $\mathbf{u}$ への射影にほかならないことを示している。したがって、たった 1 つのベクトルがあらゆる方向の傾きを同時に符号化しているのである。

点 $(0, \log 2)$ における例題で 3 つの方向を調べると、その振る舞いの全貌が見えてくる。対角方向 $\mathbf{u} = (1,1)/\sqrt{2}$ に沿った傾きは $(\tfrac{1}{3} + \tfrac{2}{3})/\sqrt{2} = 1/\sqrt{2} \approx 0.707$ である。実はこの傾きは**あらゆる**点で同じ値をとる。というのも $f(x+t, y+t) = \log(e^t(e^x + e^y)) = f(x,y) + t$ が成り立ち、対角方向に距離 $\sqrt{2}\,t$ だけ進むと f は常に t だけ増加するからである。勾配と正反対の方向 $\mathbf{u} = -(1,2)/\sqrt{5}$ に向かうと、傾きは $-(\tfrac{1}{3} + \tfrac{4}{3})/\sqrt{5} = -\sqrt{5}/3 \approx -0.745$ となる。最後に、$\mathbf{u} = (2,-1)/\sqrt{5}$ に沿った傾きは $(\tfrac{2}{3} - \tfrac{2}{3})/\sqrt{5} = 0$ となり、この方向には 1 次の範囲で f はまったく変化しない。

34.2.2 勾配の幾何学

34.2.2.1 最急降下

勾配降下法（2.1 節）は、損失が最も速く減少する方向へパラメータを少しだけ動かす手法である。Equation 34.19 によれば、求めたいのは単位ベクトル $\mathbf{v}$ のうち、射影 $\mathbf{v} \cdot \nabla \ell$ が最小（負として絶対値が最大）になるものである。1.4 節では $\mathbf{v}$ と勾配のなす角の余弦からこの方向を導いたが、コーシー・シュワルツの不等式（Cauchy–Schwarz inequality）を使えばさらに簡潔に証明でき、しかもその方向が一意であることまで確認できる。

命題（最急上昇と最急降下） $\nabla \ell \neq \mathbf{0}$ **となる任意の点において、すべての単位ベクトル** $\mathbf{v}$ **の中で、方向微分** $\mathbf{v} \cdot \nabla \ell$ **は** $\mathbf{v} = \nabla \ell / \|\nabla \ell\|$ **のとき最大となり、** $\mathbf{v} = -\nabla \ell / \|\nabla \ell\|$ **のとき最小となる。したがって、** $+\nabla \ell$ **は最急上昇（steepest ascent）の方向であり、** $-\nabla \ell$ **は最急降下（steepest descent）の方向である。**

証明： コーシー・シュワルツの不等式（33.1 節）によれば、$|\mathbf{v} \cdot \mathbf{g}| \le \|\mathbf{v}\|\|\mathbf{g}\|$ が成り立ち、等号は $\mathbf{v}$ と $\mathbf{g}$ が共線であるとき、かつそのときに限り成り立つ。単位ベクトル $\mathbf{v}$ と $\mathbf{g} = \nabla \ell$ に対して、この不等式は次のようになる。

$$-\|\nabla \ell\| \;\le\; \mathbf{v} \cdot \nabla \ell \;\le\; \|\nabla \ell\|.$$

上下それぞれの限界は、$\nabla \ell$ と共線な単位ベクトルによって一意に達成される。すなわち、上界は $\mathbf{v} = \nabla \ell / \|\nabla \ell\|$ によって、下界は $\mathbf{v} = -\nabla \ell / \|\nabla \ell\|$ によって達成される。■

点 $(0, \log 2)$ における例題では、最急降下方向は $-(1,2)/\sqrt{5}$ であり、その傾きは $-\|\nabla f\| = -\sqrt{5}/3 \approx -0.745$ となって、先ほど計算した 3 つの傾きの中で最も小さい負の値になっている。$-\nabla_{\mathbf{w}}\ell$ の方向にステップを進めると、おなじみの勾配降下法の更新式が得られる。

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\, \nabla_{\mathbf{w}} \ell(\mathbf{w}).$$

この更新式では勾配を正規化していないことに注目してほしい。ステップの長さは $\eta\|\nabla\ell\|$ であり、最小点の近くで勾配が小さくなるにつれて自然に短くなる。

見落とされがちだが、上の命題における「単位」とは**ユークリッド**長が 1 という意味であり、これは自然法則ではなくモデル化における 1 つの選択にすぎない。代わりにステップの各座標の絶対値を高々 1 に制約すると、最急降下ステップは座標ごとに $-\operatorname{sign}(\nabla\ell)$ を適用したものになる。点 $(0, \log 2)$ ではそのステップは $(-1, -1)$ であり、傾きは $-(\frac{1}{3} + \frac{2}{3}) = -1$ となる。更新量の各座標をそのスケールの移動推定値で割る Adam のような適応的オプティマイザ（第 9 章）は、ユークリッドな最急降下よりもむしろこの符号ステップに近い振る舞いをする。演習問題 4 では、他のノルムのもとでの最急降下を調べる。

34.2.2.2 勾配と等位集合

ステップを一切持ち出さなくても、等高線図から直接勾配を読み取ることもできる。ℓ の**等位集合**（level set）とは、ℓ が一定の値 c をとるすべての点の集まりである。等位集合に沿って動いても ℓ は一定のままであるから、任意の接線方向に沿った方向微分は消えるはずである。

命題（勾配は等位集合に直交する） *ℓ は点 $\mathbf{x}$ で微分可能であるとする。このとき $\nabla\ell(\mathbf{x})$ は、点 $\mathbf{x}$ において等位集合 $\{\mathbf{x}' : \ell(\mathbf{x}') = c\}$（$\mathbf{x}$ を通るもの）に接するすべての方向と直交する。*

証明： ベクトル $\mathbf{v}$ が点 $\mathbf{x}$ において等位集合に**接する**とは、$\mathbf{v} = \gamma'(0)$ となる微分可能な曲線 $\gamma(t)$ が等位集合内に存在し、$\gamma(0) = \mathbf{x}$ を満たすことをいう。このような曲線に沿うと、$\ell(\gamma(t)) = c$ がすべての t で成り立つため、$t \mapsto \ell(\gamma(t))$ の $t = 0$ における微分係数はゼロでなければならない。1 次の範囲では $\gamma(t) = \mathbf{x} + t\mathbf{v} + o(t)$ であるから、Equation 34.18 よりこの微分係数は $\mathbf{v} \cdot \nabla\ell(\mathbf{x})$ に等しい。ゆえに $\mathbf{v} \cdot \nabla\ell(\mathbf{x}) = 0$ が成り立つ。$\blacksquare$

例題でこれを確かめてみよう。f の $(0, \log 2)$ を通る等位集合は曲線 $e^x + e^y = 3$ である。この曲線に沿って微分すると $e^x\,dx + e^y\,dy = 0$ となるので、$(0, \log 2)$ における接線方向は $(e^y, -e^x) = (2, -1)$ である。勾配とのドット積をとると $(2,-1)\cdot(\frac{1}{3}, \frac{2}{3}) = 0$ となり、先ほど見つけた傾きゼロの方向と確かに一致する。

したがって等高線図の上では、勾配はすべての等高線と直角に交わり、値が大きくなる方向を指す。さらに、隣り合う等高線の値の差が Δc で間隔が d であれば、等高線を横切る傾きはおおよそ $\Delta c/d$ になるため、等高線が最も密に詰まっている場所ほど勾配ベクトルは長くなる（Figure 34.12）。

34.2.2.3 接平面

線形近似 Equation 34.18 から誤差項を取り除き、予測される高さを z と書くと、f のグラフの点 $\mathbf{x}_0$ における**接平面**（tangent plane）が得られる。

$$z = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0). \tag{34.20}$$

点 $(0, \log 2)$ における例題では、この平面は $z = \log 3 + \frac{1}{3}x + \frac{2}{3}(y - \log 2)$ となる。Equation 34.20 を $\nabla f(\mathbf{x}_0)\cdot(\mathbf{x} - \mathbf{x}_0) - (z - f(\mathbf{x}_0)) = 0$ と書き直せば、ベクトル $[\nabla f(\mathbf{x}_0), -1]^{\top}$ が $(\mathbf{x}, z)$

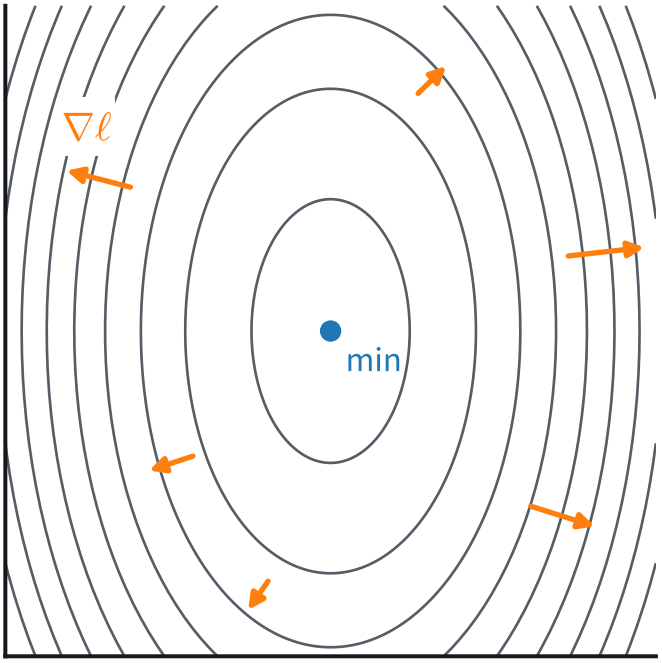


Figure 34.12: 損失 ℓ の等高線（灰色）と複数の点における $\nabla \ell$。各勾配は等高線に直交し、値が大きくなる方向を指し、等高線が最も密に詰まっている場所で最も長くなる。

空間における接平面の法線ベクトルであることが直ちにわかる。そこから高さ座標を落とすと $\nabla f(\mathbf{x}_0)$ が残り、これは底平面における等位曲線の法線ベクトルになっている（Figure 34.13）。

34.2.2.4 臨界点

深層学習の損失は閉じた形で最小化するにはあまりに複雑だが、それでも線形モデルはあらゆる最小点が満たすべき必要条件を与えてくれ、勾配降下法はまさにこの条件が成り立つ場所で停止する。以下では、f は $\mathbb{R}^n$ 全体で定義され、点 $\mathbf{x}_0$ で微分可能であるとする。

もし $\nabla f(\mathbf{x}_0) \neq \mathbf{0}$ ならば、$\epsilon = -\eta \nabla f(\mathbf{x}_0)$（$\eta > 0$）とおいて勾配と逆向きにステップを進めることができる。Equation 34.18 より、このときの f の変化量は $-\eta \|\nabla f(\mathbf{x}_0)\|^2 + o(\eta)$ であり、十分小さいすべての η に対して真に負となる。したがって $\mathbf{x}_0$ は極小点ではあり得ず、$+\eta$ を用いて同じ議論を繰り返せば極大点（local maximum）でもあり得ないことがわかる。ゆえに、**いかなる極小点・極大点においても $\nabla f(\mathbf{x}_0) = \mathbf{0}$ が成り立つ**。勾配がゼロになる点は**臨界点**（critical point）と呼ばれる。この一階条件（first-order condition）は必要条件であって十分条件ではない。大域的な極値を見つけたと言い切るためにあと何が必要かを、次の例で確認しよう。

$\psi(x, y) = x e^{-x^2 - y^2}$ を考えよう。これは $x > 0$ に正の山、$x < 0$ に対称な負のくぼみを持つ曲面である。その偏微分は次式で与えられる。

$$\frac{\partial \psi}{\partial x} = (1 - 2x^2)\, e^{-x^2 - y^2}, \qquad \frac{\partial \psi}{\partial y} = -2xy\, e^{-x^2 - y^2}.$$

第 2 の偏微分をゼロとおくと、$x = 0$ または $y = 0$ が導かれる。$x = 0$ とすると第 1 の偏微分が $e^{-y^2} \neq 0$ となってしまうため $y = 0$ でなければならず、すると $1 - 2x^2 = 0$ から $x = \pm 1/\sqrt{2}$

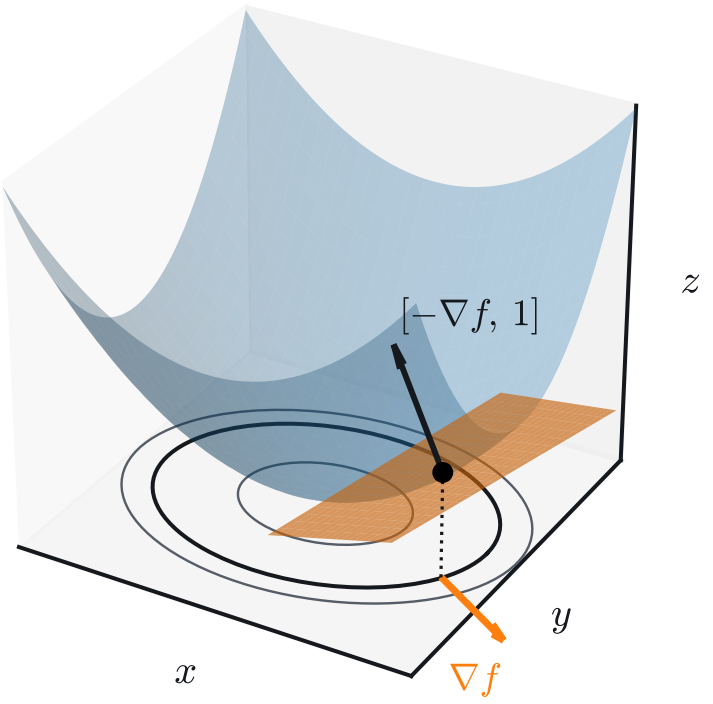


Figure 34.13: 曲面 $z = f(\mathbf{x})$（青）とある点における接平面（オレンジ）。ベクトル $[-\nabla f, 1]$（$[\nabla f, -1]$ の符号を反転したもの）は接平面に直交する（黒の矢印）。高さ座標を落とすと ∇f の定数倍が残り、底平面上（オレンジの矢印）ではその点を通る等位曲線（黒）に直交する。

が得られる。これら 2 つの臨界点における関数値は $\psi(\mp 1/\sqrt{2}, 0) = \mp e^{-1/2}/\sqrt{2} \approx \mp 0.4289$ である。

しかし、$(-1/\sqrt{2}, 0)$ が**大域的**最小点（global minimum）であると結論づけるには、そもそも大域的最小値が存在することも確かめなければならない。半径 $R = 2$ の円板の外側では、指数減衰（exponential decay）によって $|\psi| \le R e^{-R^2} < 0.04$ に抑えられるのに対し、円板の内部では $\psi(-1/\sqrt{2}, 0) \approx -0.43$ をとる。ψ は連続であるから閉円板上で最小値を達成するが、その最小値は -0.04 を下回るため境界上にはあり得ず、ψ の $\mathbb{R}^2$ 全体における大域的最小点、したがって臨界点でなければならない。2 つの候補を比較すれば $(-1/\sqrt{2}, 0)$ が大域的最小点であると特定でき、同じ議論から $(+1/\sqrt{2}, 0)$ が大域的最大点（global maximum）であるとわかる。存在確認を省くと論理が破綻する。たとえば x^3 は臨界点をただ 1 つ持つが、どこにも最小点を持たない。

一階の情報だけではできないことにも注意しておこう。最小点でも最大点でも勾配はゼロになるため、勾配だけでは両者を区別できない。それを見分けるために、二階導関数へと進む（34.2.3.3 節）。

34.2.3 ヘッセ行列

34.2.3.1 ヘッセ行列の対称性

線形モデルが捉えるのは 1 次の振る舞いだけである。臨界点が極小点か、極大点か、あるいはそのどちらでもないかは、曲面が接平面からどのように曲がって離れていくかに依存し、その曲率は二階導関数によって測られる。n 変数の関数には、次のように n^2 個の二階偏微分がある。

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right),$$

これらをまとめたものが**ヘッセ行列**である。

$$\mathbf{H}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}. \tag{34.21}$$

(i, j) 成分は、j 番目の傾きが座標 x_i に沿って動いたときにどれほど速く変化するかを測る。ヘッセ行列は $\nabla^2 f$ とも書かれ、対象となる関数と点が文脈から明らかなときは単に $\mathbf{H}$ と書く。

log-sum-exp について、$(x_1, x_2) = (x, y)$ と書いてソフトマックスの成分 $p_i = e^{x_i}/(e^{x_1} + e^{x_2})$ をもう 1 回微分してみよう。商の微分法則を適用すると $\partial p_i / \partial x_j = p_i \delta_{ij} - p_i p_j$ が得られる。ここで δ_{ij} は、1（$i = j$ のとき）または 0（それ以外のとき）をとる記号である。行列形式で書くと次のようになる。

$$\begin{aligned} \mathbf{H}_f &= \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^{\top}, \\ \mathbf{H}_f(0, \log 2) &= \begin{bmatrix} \frac{1}{3} - \frac{1}{9} & -\frac{2}{9} \\ -\frac{2}{9} & \frac{2}{3} - \frac{4}{9} \end{bmatrix} = \frac{2}{9}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \end{aligned} \tag{34.22}$$

この 2×2 行列は、固有値 0（$(1, 1)$ 方向）と固有値 $\frac{4}{9}$（$(1, -1)$ 方向）を持つ。ゼロ固有値はすでに見た事実と対応している。対角方向 $(1, 1)$ に沿うと $f(x + t, y + t) = f(x, y) + t$ は完全に

線形であり、曲率をまったく持たない。この半正定値（positive semidefinite）で特異なヘッセ行列には、のちほど二階微分判定法で再び出会うことになる。

ヘッセ行列の n^2 個の成分は互いに独立ではない。f が 2 回連続微分可能であれば微分の順序は結果に影響せず、ヘッセ行列は対称行列になる。これはクレロー・シュワルツの定理（Clairaut–Schwarz theorem）として知られる古典的な結果である (Rudin 1976)。この対称性はきわめて重要である。スペクトル定理（33.3 節）によって実固有値と互いに直交する固有ベクトルが保証され、それによって二階微分判定法が可能になるからである。

命題（ヘッセ行列の対称性：クレロー・シュワルツの定理） f **の一階偏微分、ならびに混合偏微分** $\dfrac{\partial^2 f}{\partial x_i \partial x_j}$ **と** $\dfrac{\partial^2 f}{\partial x_j \partial x_i}$ **が点** $\mathbf{x}$ **の近傍で存在し、さらにこれらの混合偏微分が点** $\mathbf{x}$ **で連続であるとする。このとき次式が成り立つ。**

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}) = \frac{\partial^2 f}{\partial x_j \partial x_i}(\mathbf{x}).$$

これがすべての i, j **について成り立つとき、ヘッセ行列は対称となる（**$\mathbf{H}_f = \mathbf{H}_f^{\top}$**）。**

証明：証明の着想は、両方の混合偏微分を**同一の**量、すなわち次の対称な二階差分の極限として表すことにある。

$$\Delta_h = \frac{f(\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} + h\mathbf{e}_j) + f(\mathbf{x})}{h^2}, \tag{34.23}$$

この式は、見た目から明らかなように i と j を入れ替えても変わらない。ここでの課題は、Δ_h を近くの点で評価した混合偏微分として表すことである。1 変数関数 $\varphi(t) = f(\mathbf{x} + t\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x} + t\mathbf{e}_i)$ を定めると、$h^2 \Delta_h = \varphi(h) - \varphi(0)$ となる。平均値の定理（34.1.3 節）を 2 回、まず φ に対して適用し、次に x_j に沿って φ' の内部にある一階偏微分の差に対して適用すると、ある数 τ と σ が 0 と h の間に存在して次式を満たす。

$$\begin{aligned} \varphi(h) - \varphi(0) &= h\,\varphi'(\tau) = h\Big[\frac{\partial f}{\partial x_i}(\mathbf{x} + \tau\mathbf{e}_i + h\mathbf{e}_j) - \frac{\partial f}{\partial x_i}(\mathbf{x} + \tau\mathbf{e}_i)\Big] \\ &= h^2\,\frac{\partial^2 f}{\partial x_j \partial x_i}(\mathbf{x} + \tau\mathbf{e}_i + \sigma\mathbf{e}_j). \end{aligned}$$

したがって、$\Delta_h = \dfrac{\partial^2 f}{\partial x_j \partial x_i}(\xi_h)$ が成り立つ。ここで点 ξ_h は、$\mathbf{x}$ と $\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j$ を対角の頂点とする正方形の内部にあり、高々 $\sqrt{2}\,|h|$ の距離に $\mathbf{x}$ から収まる点である。i と j の役割を入れ替えても Δ_h は不変であり、同じ正方形内の別の点において $\Delta_h = \dfrac{\partial^2 f}{\partial x_i \partial x_j}(\xi_h')$ が得られる。$h \to 0$ とすると、点 $\mathbf{x}$ における混合偏微分の連続性により、2 つの式はそれぞれ $\mathbf{x}$ における 2 つの混合偏微分へと収束する。すべての h に対して両者は一致するため、その極限も一致しなければならない。$\blacksquare$

連続性の仮定は省くことができない。実際、混合偏微分がいたるところで存在するにもかかわらず原点で連続でなくなり、そこで両者の値が食い違う関数が存在する。

34.2.3.2 2 次テイラー近似

勾配が f に対する最良の線形近似を与えるのとちょうど同じように、ヘッセ行列は最良の二**次**近似を与える。係数がどこから現れるかを見るには、まず厳密な二次関数から係数を読み取ってみるのがわかりやすい。$f(x_1,x_2)=a+b_1x_1+b_2x_2+c_{11}x_1^2+c_{12}x_1x_2+c_{22}x_2^2$ をとり、原点における関数値、勾配、およびヘッセ行列 Equation 34.21 を計算すると次のようになる。

$$f(0,0)=a,\qquad \nabla f(0,0)=\begin{bmatrix}b_1\\ b_2\end{bmatrix},\qquad \mathbf{H}_f(0,0)=\begin{bmatrix}2c_{11} & c_{12}\\ c_{12} & 2c_{22}\end{bmatrix},$$

これらの要素を組み合わせると、元の多項式 $f(\mathbf{x})=f(\mathbf{0})+\nabla f(\mathbf{0})\cdot\mathbf{x}+\tfrac12\mathbf{x}^{\top}\mathbf{H}_f(\mathbf{0})\,\mathbf{x}$ が厳密に復元される。係数 $\tfrac12$ は、二乗を微分したときに生じる係数 2 を打ち消しているにすぎない。

一般の関数 f に対してはこの二次関数はもはや厳密には一致しないが、f を線分上に制限することでその近似精度がわかる。f が $\mathbf{x}_0$ と $\mathbf{x}$ を結ぶ線分の近傍で 2 回連続微分可能であるとする。変位を $\delta=\mathbf{x}-\mathbf{x}_0$ と書き、1 変数スライス $\varphi(t)=f(\mathbf{x}_0+t\delta)$ を考える。線形モデル Equation 34.18 より、その一階導関数は $\varphi'(t)=\sum_i \frac{\partial f}{\partial x_i}(\mathbf{x}_0+t\delta)\,\delta_i$ である。この和に現れる各偏微分もまた微分可能であるから、同じ操作をもう 1 回適用すれば二階導関数 $\varphi''(t)=\sum_{i,j}\delta_j\frac{\partial^2 f}{\partial x_j\partial x_i}(\mathbf{x}_0+t\delta)\,\delta_i=\delta^{\top}\mathbf{H}_f(\mathbf{x}_0+t\delta)\,\delta$ が得られる。

ここで、ラグランジュの剰余項を持つ 1 変数の 1 次テイラーの定理（34.1.5.2 節）を φ の区間 0 から 1 に適用しよう。するとある $\tau\in(0,1)$ が存在して $\varphi(1)=\varphi(0)+\varphi'(0)+\tfrac12\varphi''(\tau)$ が成り立つ。これを f の言葉に戻すと次式が得られる。

$$f(\mathbf{x})=f(\mathbf{x}_0)+\nabla f(\mathbf{x}_0)\cdot\delta+\tfrac12\delta^{\top}\mathbf{H}_f(\xi)\,\delta,\qquad \xi=\mathbf{x}_0+\tau\delta.$$

この恒等式は厳密であるが、その代償としてヘッセ行列を線分上の未知の中間点 ξ で評価しており、基点 $\mathbf{x}_0$ での値にはなっていない。しかし二階偏微分は連続であるため、$\mathbf{H}_f(\xi)$ を $\mathbf{H}_f(\mathbf{x}_0)$ に置き換えても、二次の項の変化は高々 $\tfrac12\|\mathbf{H}_f(\xi)-\mathbf{H}_f(\mathbf{x}_0)\|\,\|\delta\|^2=o(\|\delta\|^2)$ に抑えられる。これにより **2 次テイラー近似**（second-order Taylor approximation）が得られる。

$$\begin{aligned} f(\mathbf{x}) &= f(\mathbf{x}_0)+\nabla f(\mathbf{x}_0)\cdot(\mathbf{x}-\mathbf{x}_0)\\ &\quad+\frac12(\mathbf{x}-\mathbf{x}_0)^{\top}\mathbf{H}_f(\mathbf{x}_0)(\mathbf{x}-\mathbf{x}_0)+o(\|\mathbf{x}-\mathbf{x}_0\|^2). \end{aligned} \tag{34.24}$$

この二次モデルと f とのずれはステップサイズの二乗より速く消える誤差だけであり、そのような性質を持つ二次関数はこれに限られる。f が 3 回連続微分可能であれば、誤差はステップに関して 3 次となる。

任意の単一方向 $\mathbf{v}$ に沿ってステップ $t\mathbf{v}$ をとると、二次の項 $\frac{t^2}{2}\mathbf{v}^{\top}\mathbf{H}\mathbf{v}$ が生じる。この値 $\mathbf{v}^{\top}\mathbf{H}\mathbf{v}=\frac{d^2}{dt^2}f(\mathbf{x}_0+t\mathbf{v})\big|_{t=0}$ は**二階方向微分**（second directional derivative）と呼ばれ、f の $\mathbf{v}$ に沿ったスライスの曲率を測る。

これで、最初のコードセルで線形モデルが残した誤差を説明できる。ステップ $\epsilon=(0.01,-0.03)$ を点 $(0,\log 2)$ から進めたとき、ヘッセ行列 Equation 34.22 が予測する 2 次の補正項は $\tfrac12\epsilon^{\top}\mathbf{H}\epsilon=\tfrac19(\epsilon_1-\epsilon_2)^2=\tfrac19(0.04)^2\approx 1.78\times10^{-4}$ となり、実測された誤差 1.786×10^{-4} とよく一致する。$\mathbf{H}$ の固有値と予測される誤差の両方をコードで確かめてみよう。

```
    p = grad_lse(xy0)                # the softmax at (0, log 2)
    H = np.diag(p) - np.outer(p, p)  # Hessian of log-sum-exp
    print('9 H =', (9 * H).round(6).tolist())
    print('eigenvalues:', np.linalg.eigvalsh(H).round(6) + 0.0)  # no '-0.'
    for scale in [1, 0.1]:
        eps = scale * np.array([0.01, -0.03])
        print(f'|eps| {np.linalg.norm(eps):.1e}: '
              f'predicted gap {0.5 * eps @ H @ eps:.4e}')
```

```
9 H = [[2.0, -2.0], [-2.0, 2.0]]
eigenvalues: [0.       0.444444]
|eps| 3.2e-02: predicted gap 1.7778e-04
|eps| 3.2e-03: predicted gap 1.7778e-06
```

行列そのものも、その固有値 0 と $\frac{4}{9}$ も Equation 34.22 と一致している。予測された誤差 1.7778×10^{-4} と 1.7778×10^{-6} は、実測値 1.786×10^{-4} と 1.779×10^{-6} に有効数字 2〜3 桁で一致する。最初の誤差に残っている約 10^{-6} のわずかなずれは 3 次の項に由来するものであり、ステップを 10 分の 1 に縮めるとその残差はおよそ 1,000 分の 1 に減少し、表示桁の下へと消えてしまう。

log-sum-exp の例では、1 つの方向が完全に平坦であった。今度は両方の方向に曲がる関数として、$\psi(x, y) = xe^{-x^2-y^2}$ を基点 $\mathbf{x}_0 = (-1, 0)$ のまわりで見てみよう。先ほど求めた偏微分から $\psi(-1, 0) = -e^{-1}$ および $\nabla\psi(-1, 0) = (-e^{-1}, 0)$ が得られ、もう 1 回微分すると $\mathbf{H}_\psi(-1, 0) = 2e^{-1}\mathbf{I}$ となる。これらを Equation 34.24 に代入すると、二次モデル $q(x, y) = e^{-1}(-1 - (x + 1) + (x + 1)^2 + y^2)$ が得られる。Figure 34.14 に示すように、曲面とその二次近似は $(-1, 0)$ の近くではほとんど見分けがつかず、遠くへ離れるにつれて乖離していく。

この二次モデルの誤差が確かに 3 次であるなら、ステップサイズを 2 倍にすると誤差は約 8 倍になるはずである。次のセルでは、ψ と q を各座標方向にステップ d だけ $(-1, 0)$ から動かした点で評価して確かめる。

```
    def psi(x, y):
        return x * np.exp(-x**2 - y**2)

    def quad(x, y):  # second-order Taylor model of psi at (-1, 0)
        return np.exp(-1) * (-1 - (x + 1) + (x + 1)**2 + y**2)

    for d in [0.0, 0.05, 0.1, 0.3]:  # step d in each coordinate from (-1, 0)
        x, y = -1 + d, d
        print(f'step {d:.2f}: psi = {psi(x, y):.6f}, quadratic = {quad(x, y):.6f}, '
              f'gap = {abs(psi(x, y) - quad(x, y)):.6f}')
```

```
step 0.00: psi = -0.367879, quadratic = -0.367879, gap = 0.000000
step 0.05: psi = -0.384315, quadratic = -0.384434, gap = 0.000119
step 0.10: psi = -0.396388, quadratic = -0.397310, gap = 0.000921
step 0.30: psi = -0.391929, quadratic = -0.412025, gap = 0.020096
```

予想どおり、基点では誤差がゼロになり、ステップを 0.05 から 0.1 へ 2 倍にすると誤差は 8 倍弱（0.000119 から 0.000921）に増えて、3 次の剰余項の振る舞いと一致する。各ステップでこの局所的な二次関数を当てはめ、**その二次関数の**最小点へ一気に飛ぶ手法がニュートン法であり、1 変数の場合は 34.1.5.5 節で導入し、多変数のオプティマイザとしては 9.1.5.1 節で展開する。

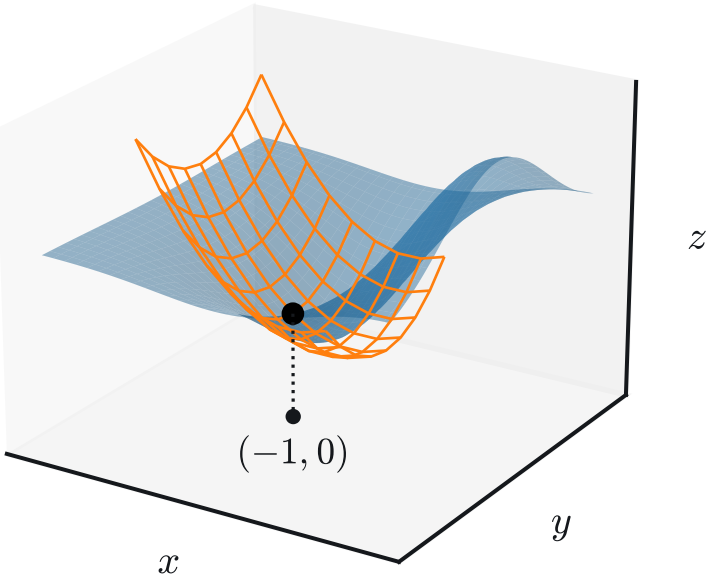


Figure 34.14: 曲面 $z = xe^{-x^2-y^2}$（青）と基点 $(-1, 0)$ における 2 次テイラー二次関数（オレンジ）。両者は基点で値・傾き・曲率が一致するため、基点の近くではほとんど見分けがつかず、外側へ離れると分かれていく。

34.2.3.3 二階微分判定法

臨界点 $\mathbf{x}_0$ では、Equation 34.24 の一次の項が消える。したがって $\mathbf{x}_0$ の近傍では、関数は $o(\|\mathbf{x}-\mathbf{x}_0\|^2)$ の誤差を除いて二次の項のように振る舞う。そのため、曲率がゼロでない限り、f が方向 $\mathbf{v}$ に離れたときに増加するか減少するかは $\mathbf{v}^{\top}\mathbf{H}\mathbf{v}$ の符号によって決まる。

$\mathbf{H}$ の固有値を使えば、これらの符号をすっきりと整理できる。$\mathbf{H}$ は対称行列であるため、スペクトル定理（33.3 節）により、固有ベクトル $\mathbf{q}_i$ からなる正規直交基底と実固有値 λ_i の存在が保証される。この基底で展開すると、二次形式（quadratic form）は二乗の単純な重み付き和になる。

$$\mathbf{v}^{\top}\mathbf{H}\mathbf{v} = \sum_{i=1}^{n} \lambda_i \, (\mathbf{q}_i^{\top}\mathbf{v})^2. \tag{34.25}$$

この式から局所的な幾何構造が一目で読み取れる。すべての $\lambda_i > 0$ が成り立てば、曲率はあらゆる方向で真に正となり、少なくとも $\lambda_{\min}\|\mathbf{v}\|^2$ 以上になる。一方、固有値に正と負の両方の符号が混在していれば、曲面はある固有ベクトルに沿っては上向きに曲がり、別の固有ベクトルに沿っては下向きに曲がる。

命題（二階条件） *f は臨界点 $\mathbf{x}_0$ の近傍で 2 回連続微分可能であるとし、$\mathbf{H} = \mathbf{H}_f(\mathbf{x}_0)$ と書く。*

1. *$\mathbf{x}_0$ が極小点ならば、$\mathbf{H} \succeq 0$（すべての固有値が ≥ 0）である。*
2. *$\mathbf{H} \succ 0$（すべての固有値が > 0）ならば、$\mathbf{x}_0$ は狭義の極小点である。*
3. *$\mathbf{H}$ が正負両方の符号の固有値を持つならば、ゼロ固有値の有無にかかわらず $\mathbf{x}_0$ は極小点でも極大点でもなく、鞍点である。*

極大点に関する対応する主張は、この命題を $-f$ に適用すれば得られる。$\mathbf{H}$ が特異かつ半定値であるときは 2 も 3 も適用できず、二階微分判定法では判定不能となる。

証明： (1) 背理法のため、$\mathbf{v}^{\top}\mathbf{H}\mathbf{v} < 0$ となる方向 $\mathbf{v}$ が存在すると仮定する。1 変数スライス $\varphi(t) = f(\mathbf{x}_0 + t\mathbf{v})$ は $\varphi'(0) = \nabla f(\mathbf{x}_0)\cdot\mathbf{v} = 0$ かつ $\varphi''(0) = \mathbf{v}^{\top}\mathbf{H}\mathbf{v} < 0$ を満たす。1 変数の二階微分判定法により、φ は $t=0$ で狭義の極大となるため、f は $f(\mathbf{x}_0)$ より真に小さい値を $\mathbf{x}_0$ のいくらでも近くでとることになり、$\mathbf{x}_0$ が極小点であるという仮定に矛盾する。

(2) すべての方向について同時に剰余項を一様に抑える必要があるため、このステップには注意を要する。$\nabla f(\mathbf{x}_0) = \mathbf{0}$ のもとで厳密なテイラーの恒等式を用いると、$f(\mathbf{x}_0+\delta) - f(\mathbf{x}_0) = \tfrac{1}{2}\delta^{\top}\mathbf{H}_f(\xi)\delta$ となる点 ξ が、$\mathbf{x}_0$ と $\mathbf{x}_0+\delta$ を結ぶ線分上に存在する。ヘッセ行列を $\mathbf{H}_f(\xi) = \mathbf{H} + \big(\mathbf{H}_f(\xi) - \mathbf{H}\big)$ と分解しよう。Equation 34.25 より、主要項は $\delta^{\top}\mathbf{H}\delta \geq \lambda_{\min}\|\delta\|^2$ を満たす。また、二階偏微分の連続性により、$\|\mathbf{H}_f(\xi)-\mathbf{H}\| \leq \tfrac{1}{2}\lambda_{\min}$ が十分小さいすべての δ に対して成り立つ。これら 2 つの評価を組み合わせると、$f(\mathbf{x}_0+\delta) - f(\mathbf{x}_0) \geq \tfrac{1}{2}(\lambda_{\min} - \tfrac{1}{2}\lambda_{\min})\|\delta\|^2 = \tfrac{1}{4}\lambda_{\min}\|\delta\|^2 > 0$ が十分小さいすべての $\delta \neq \mathbf{0}$ に対して成り立つ。

(3) $\lambda > 0$ である固有ベクトルに沿うと、スライス φ は $\varphi'(0)=0$ かつ $\varphi''(0) = \lambda > 0$ を満たすため、f は $f(\mathbf{x}_0)$ より真に大きい値を $\mathbf{x}_0$ のいくらでも近くでとる。同様に $\lambda < 0$ である固有ベクトルに沿うと、真に小さい値をとる。$\blacksquare$

3 つの主張のうち覚えておくべきは 2 番目であり、証明に実質的な手間がかかるのもこれだけである。また、判定不能となるケースは単なる理論上の注釈ではないことにも注意したい。ゼロ固有ベクトルに沿うと二次モデルは完全に平坦になり、高次の項が振る舞いを支配するからである。たとえば $x^2 + y^4$ と $x^2 - y^4$ は原点で同じヘッセ行列 $\operatorname{diag}(2,0)$ を共有するが、前者は原点で狭義の極小点を持つのに対し、後者は鞍点を持つ（演習問題 3 ではこの現象をさらに掘り下げる）。

Figure 34.15 は、これまでの例題の二次モデルを用いて 4 つのケースすべてを図示したものである。

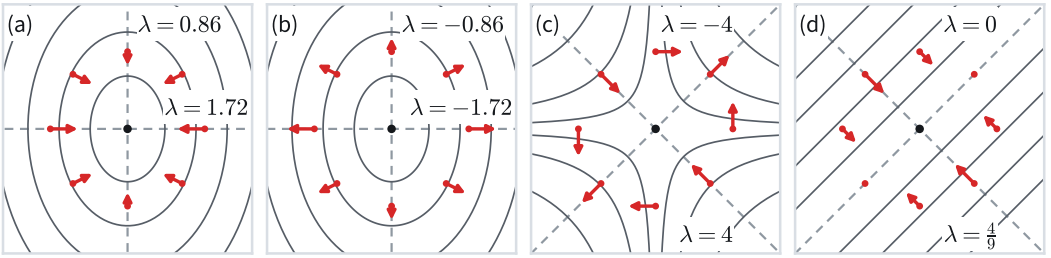


Figure 34.15: 二次モデル $\frac{1}{2}\mathbf{v}^\top \mathbf{H}\mathbf{v}$ の等高線（灰色）、固有値を添えた固有ベクトル方向（破線）、および降下方向の矢印 $-\mathbf{H}\mathbf{v}$（赤）。(a) $\mathbf{H} \succ 0$：極小点であり、すべての降下矢印が内側を向く。(b) $\mathbf{H} \prec 0$：極大点。(c) 不定値：鞍点であり、一方の固有ベクトルに沿って内側へ、他方に沿って外側へ流れる。(d) 特異かつ半定値：二次モデルはゼロ固有ベクトルに沿って平坦であり、2 次の情報では判定不能となる。

Figure 34.15 において、(a) は ψ の $(-1/\sqrt{2}, 0)$ におけるヘッセ行列 $e^{-1/2}\,\mathrm{diag}(2\sqrt{2}, \sqrt{2})$ を示し、(b) は $(+1/\sqrt{2}, 0)$ におけるその符号反転を示す。(c) は $x^4 + y^4 - 4xy$ の原点を示しており、そのヘッセ行列は $\left(\begin{smallmatrix} 0 & -4 \\ -4 & 0 \end{smallmatrix}\right)$ である。(d) は純粋に行列の例として log-sum-exp のヘッセ行列 Equation 34.22 を再利用したものである。log-sum-exp 自体は、その勾配であるソフトマックスが決してゼロにならないため臨界点を持たない。

二階微分判定法を数値的に実装し、平坦な方向を持つ点に対して実行すると何が起きるだろうか。有限差分（finite difference）から構成したヘッセ行列が厳密にゼロの固有値を返すことはまずないため、それ以下をゼロとみなす許容誤差を設定しなければならない。次のセルにおいて、中心混合差分 `fxy` は二階差分 Equation 34.23 をステップ $2h$、基点 $(x - h, y - h)$ で評価したものである。

```
def hessian(f, x, y, h=1e-4):  # 2x2 Hessian from second differences
    fxx = (f(x + h, y) - 2 * f(x, y) + f(x - h, y)) / h**2
    fyy = (f(x, y + h) - 2 * f(x, y) + f(x, y - h)) / h**2
    fxy = (f(x + h, y + h) - f(x + h, y - h)
           - f(x - h, y + h) + f(x - h, y - h)) / (4 * h**2)
    return np.array([[fxx, fxy], [fxy, fyy]])

def classify(lo, hi, tol=1e-6):  # from the smallest, largest eigenvalue
    if lo > tol:
        return 'minimum'
    if hi < -tol:
        return 'maximum'
    if lo < -tol < tol < hi:
        return 'saddle'
    return 'inconclusive'

quartic = lambda x, y: x**4 + y**4 - 4 * x * y
trough = lambda x, y: x**2 + y**4
for fn, x0 in [(psi, -1 / np.sqrt(2)), (psi, 1 / np.sqrt(2)),
               (quartic, 0.0), (trough, 0.0)]:
    lo, hi = np.linalg.eigvalsh(hessian(fn, x0, 0.0))
```

```
        print(f'({x0:+.4f}, 0): eigenvalues {lo:+.3f}, {hi:+.3f} '
              f'-> {classify(lo, hi)}')
```

```
(-0.7071, 0): eigenvalues +0.858, +1.716 -> minimum
(+0.7071, 0): eigenvalues -1.716, -0.858 -> maximum
(+0.0000, 0): eigenvalues -4.000, +4.000 -> saddle
(+0.0000, 0): eigenvalues +0.000, +2.000 -> inconclusive
```

点 $(-1/\sqrt{2}, 0)$ では固有値は 0.858 と 1.716（それぞれ $\sqrt{2}e^{-1/2}$ と $2\sqrt{2}e^{-1/2}$ に一致）となり、先ほど見つけた大域的最小点であることが裏づけられる。点 $(+1/\sqrt{2}, 0)$ では両方の符号が反転し、極大点であることが確認できる。$x^4 + y^4 - 4xy$ の原点では固有値は ± 4 となり、上昇方向と下降方向である $(1, -1)$ と $(1, 1)$ が座標軸から回転した鞍点を与える。示唆に富むのは 4 番目の点である。$x^2 + y^4$ の原点では、有限差分は平坦な方向に沿って $2h^2 = 2 \times 10^{-8}$ を返すが、これは許容誤差を下回るため、原点が実際には狭義の極小点であるにもかかわらず、プログラムは正しく「inconclusive」（判定不能）と報告する。2 次の情報からは 4 次の立ち上がりがそもそも見えず、許容誤差のおかげで数値判定が誤った結論を下さずに済んでいるのである。

34.2.3.4 高次元における鞍点

n 次元において、極小点であるためにはヘッセ行列の n 個すべての固有値が非負でなければならず、非退化な極小点であるためには n 個すべてが真に正でなければならない。もしランダムな臨界点における固有値の符号が独立なコイン投げのように振る舞うなら、n 個すべてが正になる確率は 2^{-n} であり、n が大きければそれだけでも無視できるほど小さくなる。

驚くべきことに、このコイン投げの経験則は実際には極小点が見つかる確率を**過大評価**している。対角成分が非対角成分の 2 倍の分散を持つランダム対称行列、すなわちガウス直交アンサンブル（Gaussian orthogonal ensemble、GOE）を考えよう。$n = 2$ ですら、両方の固有値が正になる厳密な確率は $(1 - 1/\sqrt{2})/2 \approx 0.146$ であり、0.25 より小さい（演習問題 4）。大きな n に対しては、この確率は指数関数よりはるかに速く $\exp\!\left(-\tfrac{\ln 3}{4} n^2\right)$ のオーダーで減衰する (Dean and Majumdar 2006)。ランダム対称行列の固有値は互いに反発し合い、ゼロの片側に密集することがほとんどないからである。

したがって高次元のランダムなランドスケープでは、典型的な（高い）関数値にある臨界点は圧倒的多数が鞍点である。関数値が低い領域に入って初めてヘッセ行列の固有値が正の側へシフトし、極小点が支配的になり始める (Bray and Dean 2007)。ニューラルネットワークも同じ振る舞いを示す。小規模なネットワークの損失曲面上でニュートン法により臨界点を探すと、損失が大きい臨界点は鞍点ばかりである (Dauphin et al. 2014)。実用上、オプティマイザが高い損失のまま停滞したとき、質の悪い極小点に陥ったという説明は最も可能性が低い。オプティマイザは鞍点の近くでもたついているか、あるいはさらに多いケースとして、勾配がまったく小さくない悪条件の領域を横切っている最中なのである。

34.2.4 制約付き最適化とラグランジュ乗数

ここまでは、変数はどの方向にも自由に動けるものとしてきた。しかし多くの機械学習の問題では、パラメータはある曲面上に制限される。単位長に正規化された重み、和が 1 になるよう制約された確率、あるいは曲線上で原点に最も近い点などがその例である。これらの制約付き最適化（constrained optimization）問題ではいずれも、$f(\mathbf{x})$ を制約（constraint）$g(\mathbf{x}) = c$ のもとで最小化したい。制約なしの条件 $\nabla f = \mathbf{0}$ はもはや成り立たない。最良の実行可能点（feasible point）において ∇f がゼロになる必要はなく、制約曲面上に留まる移動に対して 1 次の減少を生まないだけで十分だからである。注目すべきことに、$\nabla f = \mathbf{0}$ に取って代わる条件は、二階導関数を一切使わずに等位集合の幾何学から直接導かれる。

f と g は微分可能であるとする。実行可能点において許容される無限小移動は、方向 $\mathbf{v}$ のうち制約曲面 $\{g = c\}$ に接するものである。制約付きの局所最小点 $\mathbf{x}^\star$ では、いかなる許容移動も 1 次の範囲で f を減少させられないため、$\mathbf{v} \cdot \nabla f(\mathbf{x}^\star) \geq 0$ がすべての接線方向 $\mathbf{v}$ に対して成り立つ。$-\mathbf{v}$ もまた接線方向であるから、実は等号が成り立たなければならず、$\nabla f(\mathbf{x}^\star)$ はすべての接線方向と直交する。一方、等位集合の命題より $\nabla g(\mathbf{x}^\star)$ もすべての接線方向と直交するため、2 つの勾配は同じ直線上に並ばざるを得ない。

命題（ラグランジュ条件）$\mathbf{x}^\star$ を f の $\{\mathbf{x} : g(\mathbf{x}) = c\}$ 上における局所最小点とする。f は点 $\mathbf{x}^\star$ で微分可能であり、g は $\mathbf{x}^\star$ の近傍で連続微分可能であり、かつ $\nabla g(\mathbf{x}^\star) \neq \mathbf{0}$ であるとする。このとき、ラグランジュ乗数（Lagrange multiplier）と呼ばれるスカラー μ が存在して次式を満たす。

$$\nabla f(\mathbf{x}^\star) = \mu \, \nabla g(\mathbf{x}^\star). \tag{34.26}$$

証明： 証明は制約集合に関する 1 つの幾何学的事実に基づく。後半の 2 つの仮定のもとで、$\{g = c\}$ は $\mathbf{x}^\star$ の近傍で滑らかな曲面をなし、任意のベクトル $\mathbf{v}$ が $\nabla g(\mathbf{x}^\star) \cdot \mathbf{v} = 0$ を満たすならば、それは速度ベクトル $\gamma'(0)$ として、微分可能な曲線 γ で $\{g = c\}$ 内に含まれ $\gamma(0) = \mathbf{x}^\star$ を満たすものから得られる (Rudin 1976)。このような曲線に沿うと、1 変数関数 $t \mapsto f(\gamma(t))$ は $t = 0$ で極小となるため、そこでの微分係数はゼロでなければならない。等位集合の証明と同様、その微分係数は $\mathbf{v} \cdot \nabla f(\mathbf{x}^\star)$ に等しい。したがって、$\nabla f(\mathbf{x}^\star)$ は、ベクトル $\mathbf{v}$ のうち $\nabla g(\mathbf{x}^\star)$ に直交するもの全体がなす超平面と直交する。その超平面と直交するベクトルは $\nabla g(\mathbf{x}^\star)$ のスカラー倍に限られるため、$\nabla f(\mathbf{x}^\star) = \mu \nabla g(\mathbf{x}^\star)$ が成り立つ。$\blacksquare$

幾何学的には、制約付き最適点において f と g の勾配は平行であり、つまり f の $\mathbf{x}^\star$ を通る等位集合は制約曲面に接している。代数的には、n 個の式からなる Equation 34.26 と制約 $g(\mathbf{x}) = c$ を合わせることで、$n + 1$ 個の方程式が $n + 1$ 個の未知数 $\mathbf{x}$ と μ に対して得られる。

具体例として、双曲線 $xy = 1$ 上で原点に最も近い点を求めてみよう。すなわち、$f(x, y) = x^2 + y^2$ を制約 $g(x, y) = xy = 1$ のもとで最小化したい。勾配は $\nabla f = (2x, 2y)$ と $\nabla g = (y, x)$ であり、双曲線上のいたるところで $\nabla g \neq \mathbf{0}$ が成り立つ。ラグランジュ条件から $2x = \mu y$ と $2y = \mu x$ が得られる。第 2 式を第 1 式に代入すると $4x = \mu^2 x$、すなわち $x(4 - \mu^2) = 0$ となる。$x = 0$ は $xy = 1$ 上ではあり得ないため、$\mu = \pm 2$ でなければならない。$\mu = -2$ を選ぶと $y = -x$ となり、したがって $xy = -x^2 = 1$ となって実数解を持たない。残るのは $\mu = 2$ すなわち $x = y$ であり、2 つの候補 $(1, 1)$ と $(-1, -1)$ が得られ、いずれも $f = 2$ をとる。各分枝上で $f = x^2 + 1/x^2$ は $x \to 0$ または $|x| \to \infty$ のとき限りなく大きくなるため大域的最小値が必ず存在し、これら 2 点が確かに最も近い点である。点 $(1, 1)$ において円 $x^2 + y^2 = 2$ は双曲線にぴったりと接し、$\nabla f = (2, 2) = 2 \, \nabla g$ が成り立っている（Figure 34.16）。

双曲線は幾何学的イメージを描きやすいが、同じ道具を使えば本節の共通の例題へとまっすぐ戻ってくる。$\mathbf{z} \cdot \mathbf{p} - \sum_i p_i \log p_i$ を確率ベクトル $\mathbf{p}$ について制約 $\sum_i p_i = 1$ のもとで最大化することを考えよう。ラグランジュ条件を書き下すと $z_i - \log p_i - 1 = \mu$ が各座標 i について得られ、これから $p_i \propto e^{z_i}$ が従う。したがって最適な分布はソフトマックスであり、達成される最大値は log-sum-exp となる（35.7 節）。

正則性の仮定 $\nabla g(\mathbf{x}^\star) \neq \mathbf{0}$ は落とすことができない。これがないと、ラグランジュ乗数がそもそも存在しない場合がある（演習問題 6 で明快な反例を扱う）。のちほど 35.6 節では KKT 条件を通じてこの条件を不等式制約へと拡張し、35.7 節では双対問題（dual problem）を展開する。

本節のすべての結果は、勾配がすでにわかっていることを前提としてきた。黒板に書ける 2 変数の式であれば、手計算で勾配を求めるのは造作もない。しかし 100 万個の重みを持つニューラルネットワークでは、勾配を効率よく求めることこそが難しい問題となる。

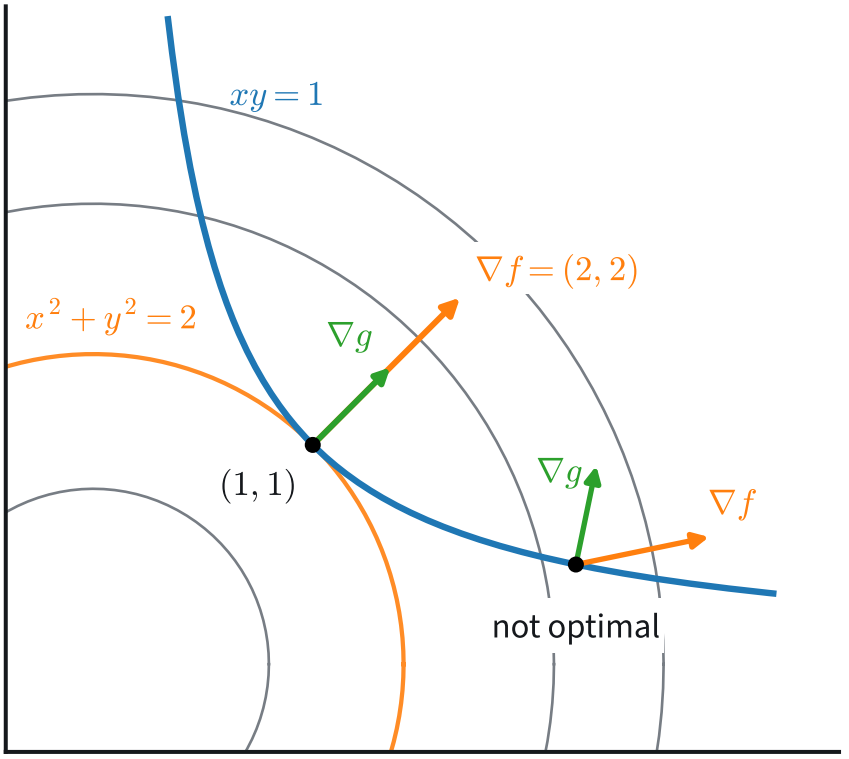


Figure 34.16: $xy = 1$ 上で原点に最も近い点。点 $(1, 1)$ では円 $x^2 + y^2 = 2$ が双曲線に接し、$\nabla f = (2, 2) = 2\nabla g$ となるため $\mu = 2$ である（鏡像解 $(-1, -1)$ は図の範囲外）。最適でない実行可能点では 2 つの勾配は異なる方向を指し、∇f は制約に沿った成分を残している。

34.2.5 まとめ

微分可能な関数は、1 次の範囲においてステップと勾配のドット積だけ変化する。単に偏微分が存在するだけでは、この性質は保証されない。幾何学的には、負の勾配は最急降下方向を指し、勾配はすべての等位集合に直交する。次数を 1 つ上げると、対称なヘッセ行列が局所的な二次近似を定める。臨界点においてはその固有値の符号が極小点・極大点・鞍点を判別し、ヘッセ行列が特異かつ半定値のときは判定不能となる。変数が制約曲面 $g(\mathbf{x}) = c$ 上に制限されるときには、等位集合の幾何学によって $\nabla f = \mathbf{0}$ がラグランジュ条件 $\nabla f = \mu \nabla g$ へと置き換えられる。

34.2.6 演習問題

1. **方向微分はあるが勾配を持たない関数。** $f(x,y) = x^2 y/(x^2+y^2)$（$(x,y) \neq (0,0)$ のとき）および $f(0,0) = 0$ とする。

   1. f が原点で連続であることを示せ。
   2. 原点における f の 2 つの偏微分と、任意の単位ベクトル $\mathbf{u} = [\cos\theta, \sin\theta]^\top$ に沿った原点での方向微分を計算せよ。
   3. いかなるベクトル $\mathbf{g}$ に対してもこれらの方向微分は $\mathbf{u}\cdot\mathbf{g}$ の形では表せず、したがって f は原点で微分可能でないことを示せ。34.2.1.2 節で述べた微分可能性の十分条件のうち、どの仮定が破れているか。また、同項の反例 $2xy/(x^2+y^2)$ では示せなかったどのような事実をこの関数は示しているか。

2. **引き伸ばされたボウルの勾配。** $f(x,y) = x^2 + \kappa y^2$（$\kappa \geq 1$）とする。その最小点は原点であり、$\kappa$ は f が y 方向に沿って x 方向よりどれだけ強く曲がっているかを測る。

   1. f の $(1,1)$ を通る等位曲線に接する方向を 1 つ求め、それが $\nabla f(1,1)$ と直交することを確かめよ。さらに、$-\nabla f(1,1)$ と、方向 $-[1,1]^\top$（$(1,1)$ から最小点へ向かう方向）のなす角を計算せよ。$(x,y) \neq (0,0)$ のうち、負の勾配が最小点をまっすぐ指すのはどの点か。
   2. すべての点 $(x,y) \neq (0,0)$ にわたるこの角の余弦の最小値と、それを達成する点を求めよ。（ヒント：$\rho = y/x$ とおき、ρ に関して最小化せよ。）$\kappa = 10$ のときの値を計算せよ。κ が大きくなるにつれて、このような損失に対する勾配降下法のステップに何が起きることをこの結果は示唆しているか。

3. **平坦な方向。**

   1. (x,y) の関数であって、原点に臨界点を持ち、そこでのヘッセ行列が $\mathrm{diag}(2,0)$ となるものを 3 つ挙げよ。ただし、1 つは狭義の極小点を持つもの、1 つは狭義でない極小点を持つもの、もう 1 つは極小点を持たないものとせよ。
   2. 関数 $f(x,y) = (x - y^2)(x - 2y^2)$ も原点でヘッセ行列 $\mathrm{diag}(2,0)$ を持つ。原点を通る任意の直線への制限が原点で狭義の極小点を持つことを示せ。平面上で $f < 0$ となる領域を図示し、それにもかかわらず原点は f の極小点ではないことを示せ。少数の方向に沿って損失を調べることで臨界点を判定する手法に対し、この事実は何を示唆するか。
   3. **猿の鞍** $f(x,y) = x^3 - 3xy^2$ は原点に臨界点を持つ。原点におけるヘッセ行列を計算し、f を極座標で表して、f が減少する方向をすべて求めよ。原点からの距離に対して $\|\nabla f\|$ は、鞍点 $x^2 - y^2$ と比べてどのようにスケールするか。これら 2 種類の鞍点の近くから出発した勾配降下法に対して、この違いは何を意味するか。

4. [code] **ランダムなヘッセ行列。**●極小点であるためにはヘッセ行列の n 個すべての固有値が非負でなければならず、非退化な極小点であるためにはすべて正でなければならない。本節のコイン投げの経験則では後者の確率は 2^{-n} となる。この経験則をランダム対称行列

$\mathbf{A} = (\mathbf{G} + \mathbf{G}^\top)/\sqrt{2}$ で検証せよ。ただし $n \times n$ 行列 $\mathbf{G}$ の各成分は独立な標準正規分布に従うとする。

1. $n = 2$ のとき、$s = (a_{11} + a_{22})/2$、$d = (a_{11} - a_{22})/2$、および a_{12} が独立な標準正規確率変数であること、そして $\mathbf{A} \succ 0$ となる必要十分条件が $s > \sqrt{d^2 + a_{12}^2}$ であることを示せ。3 次元の標準正規分布の回転対称性を用いて $P(\mathbf{A} \succ 0)$ を厳密に求め、2^{-2} と比較せよ。
2. モンテカルロ法により $P(\mathbf{A} \succ 0)$ を $n = 1, \dots, 6$ について推定せよ。ただしサンプル数は 10^6（$n \le 5$ のとき）およびバッチごとに生成した 10^7（$n = 6$ のとき）とせよ。$\log P$ を n に対して $\log 2^{-n}$ とともにプロットし、推定値に n の二次関数を当てはめよ。Dean and Majumdar (2006) は $\log P \approx -\frac{\ln 3}{4} n^2$ が大きな n で成り立つことを証明している。この係数と当てはめた二次の係数を比較せよ。コイン投げの経験則は極小点の確率を過大評価しているか過小評価しているか。また、固有値のどのような性質を無視しているか。
3. 大きな n に対して $\mathbf{A}$ の最小固有値は $-2\sqrt{n}$ の近くに位置する。$n = 5, 10, 20$ について、シフト β のうち行列 $\mathbf{A} + \beta \mathbf{I}$ のちょうど半分が正定値となるものを求め、$2\sqrt{n}$ と比較せよ。Bray and Dean (2007) のランダムなランドスケープでは、臨界点におけるヘッセ行列はそのようなシフトされた行列のように振る舞い、臨界点でのランダム関数の値が小さくなるほどシフトが大きくなる。この結果は、損失の小さい臨界点について何を示唆しているか。

5. [extended] **線形ネットワークの鞍点。●●** $\mathbf{A}$ を $n \times n$ 対称行列とし、固有値 $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$ と固有ベクトルからなる正規直交基底 $\mathbf{q}_1, \dots, \mathbf{q}_n$ を持つとする。隠れユニットを 1 つ持ち、入力重み $\mathbf{v}$ と出力重み $\mathbf{u}$ を備えた線形ネットワークは、$\mathbf{x}$ を $\mathbf{u}\mathbf{v}^\top \mathbf{x}$ へと写す。白色化された入力（$\mathbb{E}[\mathbf{x}\mathbf{x}^\top] = \mathbf{I}$）とターゲット $\mathbf{A}\mathbf{x}$ で訓練するとき、期待二乗誤差の半分は次式で与えられる。

$$\ell(\mathbf{u}, \mathbf{v}) = \tfrac{1}{2} \| \mathbf{A} - \mathbf{u}\mathbf{v}^\top \|_F^2 .$$

Baldi and Hornik (1989) は、このような線形ネットワークの極小点はすべて大域的最小点であることを示した。本問では臨界点の集合を完全に導出し、Dauphin et al. (2014) が非線形ネットワークについて計測した損失と指数の関係と比較する。なお、臨界点の**指数**とは、そのヘッセ行列の負の固有値の割合である。

1. $\nabla_{\mathbf{u}} \ell$ と $\nabla_{\mathbf{v}} \ell$ を計算せよ。臨界点が原点および $\mathbf{u} = \alpha \mathbf{q}_i$、$\mathbf{v} = (\lambda_i/\alpha)\mathbf{q}_i$（$\alpha \neq 0$）の形をした点であることを示し、それぞれの点における ℓ の値を計算せよ。
2. $2n \times 2n$ ヘッセ行列の固有値を、釣合いのとれた点 $\mathbf{u} = \mathbf{v} = \sqrt{\lambda_i}\,\mathbf{q}_i$ および原点において計算せよ。（固有ベクトル基底で展開すると、ヘッセ行列は 2×2 ブロックへと分解される。各ブロックは $\mathbf{q}_j$ に沿った $\mathbf{u}$ と $\mathbf{v}$ の座標対に対応する。）各臨界点で負の固有値は何個あるか。ゼロ固有値が生じる理由を ℓ の対称性から説明し、すべての極小点が確かに大域的最小点であると結論せよ。
3. $n = 4$、$(\lambda_1, \dots, \lambda_4) = (4, 3, 2, 1)$ とし、ランダムな正規直交固有ベクトル基底をとれ。スケールの異なる多数のランダムな初期点からニュートン法を実行し、到達したすべての臨界点について損失と指数を記録せよ。原点以外のすべての臨界点でヘッセ行列は特異になるため、各ニュートンステップは最小二乗法で計算せよ。Dauphin et al. (2014) の図 1 と同様に、損失を指数に対してプロットせよ。ニュートン法はどの種類の臨界点にどれくらいの頻度で到達するか。また、小さなランダム初期化から出発した勾配降下法はどこへ収束するか。
4. $\mathbf{u}\mathbf{v}^\top$ を隠れユニット 2 個の $\mathbf{U}\mathbf{V}^\top$ に置き換えよ（$\mathbf{U}$ と $\mathbf{V}$ は 4×2 行列となる）。ランク 1 の解析結果から臨界点での損失値を予測し、ニュートン法で確認せよ。臨界点に

おいてヘッセ行列はゼロ固有値を何個持つか。また、どの対称性がそれを説明するか。指数は損失の単調関数になっているか。

6. **ラグランジュ乗数とレイリー商。** $\mathbf{A}$ を $n \times n$ 対称行列とする。$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x}$ を制約 $g(\mathbf{x}) = \|\mathbf{x}\|^2 = 1$ のもとで最大化せよ。
    1. ラグランジュ条件を書き下し、それが $\mathbf{A}\mathbf{x} = \mu\mathbf{x}$ と表せること、すなわち制約付き臨界点はすべて単位固有ベクトルであり、その乗数が固有値に一致することを示せ。
    2. すべての制約付き臨界点で $f(\mathbf{x}) = \mu$ となることを示せ。なぜ制約付き最大値が存在するのか。以上から最大値は $\lambda_{\max}(\mathbf{A})$、最小値は $\lambda_{\min}(\mathbf{A})$ であると結論せよ。
    3. $\mathbf{A} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ に対して最大点・最小点とそれぞれの値を求め、ある最大点においてそこを通る f の等位集合が単位円に接していることを確かめよ。
    4. $g(\mathbf{x}) = \|\mathbf{x}\|^2$ は単位球面上のすべての点でラグランジュ条件の命題のどの仮定を満たしているか。次に、線形関数 $\mathbf{c}^\top \mathbf{x}$（$\mathbf{c} \neq \mathbf{0}$）を制約 $\|\mathbf{x}\|^2 = 0$ のもとで最小化せよ。最小点は何か。また、その点でいかなる乗数 μ も $\nabla f = \mu \nabla g$ を満たさないのはなぜか。

34.3 計算グラフ上の連鎖律

本書におけるすべての訓練ステップは、`backward()` または `grad` という 1 回の呼び出しから始まる。この呼び出しは、ネットワーク内のあらゆる重みに関する損失の傾きを返す。34.2 節では、それらの傾きが何を意味するかを見た。すなわち、それらをまとめたものが勾配（gradient）であり、パラメータを同時に微小変化させたときに損失がどう変化するかを予測するベクトルとなる。まだ説明していないのは、ネットワークを数回評価するのとほぼ同じ時間で、プログラムがどのようにして数百万個ものこうした傾きを計算しているのかという点である。重みを 1 つずつ微小に動かし、摂動のたびにネットワークを再実行する素朴な手法では、重みごとに別々のパスが必要となり、たちまち実行不可能になる。

連鎖律（chain rule）を使えばはるかに効率よく計算でき、その理由は数式を書き下す前にも容易に見て取れる。ニューラルネットワークは単純な関数を深く合成したものであるため、多数の重みに関する微分は計算の大部分を共有している。どの重みの影響も、後続の層を通過して損失へと到達しなければならないからである。入れ子になった 1 つの数式ではこの共有構造が隠れてしまうのに対し、中間値ごとに 1 つのノード、直接の依存関係ごとに 1 本の有向エッジを置く**計算グラフ**（computational graph）はその構造を明瞭に描き出す。グラフ上で見れば、連鎖律は経路にわたる和となり、共有された計算を活用することはその和をどうまとめるかを選ぶことに帰着する。出力から逆向きに経路をまとめれば、1 回のパスですべての入力に関する微分が得られる。このまとめ方こそが誤差逆伝播法（backpropagation）である。

すべてのステップを具体的で手計算でも確かめやすいものにするため、本節を通して整数値をとるスカラー変数からなる小さなグラフを扱う。ここでの議論全体は、34.2 節で学んだ 1 つの事実に基づいている。すなわち、関数が微分可能な任意の点の近傍において、関数の変化量は変数の変化ステップと勾配のドット積、つまり 34.2.1.2 節の線形モデルで近似されるということである。この考え方をベクトルからベクトルへ写す層へと拡張し、実際のネットワークにおける計算コストを測る議論は、続く 2 つの節で扱う。

34.3.1 経路にわたる和としての連鎖律

34.3.1.1 スカラーのグラフ

すべて手計算で追えるほど小さな具体例から始めよう。4 つのスカラー入力 w, x, y, z が中間変数の 2 つの層を経て、1 つのスカラー出力を生成する。

$$
\begin{aligned}
f(u,v) &= (u+v)^2, \\
u(a,b) &= (a+b)^2, \qquad v(a,b) = (a-b)^2, \\
a(w,x,y,z) &= (w+x+y+z)^2, \qquad b(w,x,y,z) = (w+x-y-z)^2.
\end{aligned}
\tag{34.27}
$$

これらの式は、Figure 34.17 に示す有向非巡回グラフ（directed acyclic graph）を定めており、各ノードが中間値を保持し、各矢印が直接の関数依存関係を表す。二乗演算自体に特別な意味はなく、中間値と導関数の値がすべて厳密な整数に保たれるという理由だけで選んでいる。本節を通して、対象とするグラフのスカラー出力を f で表す。

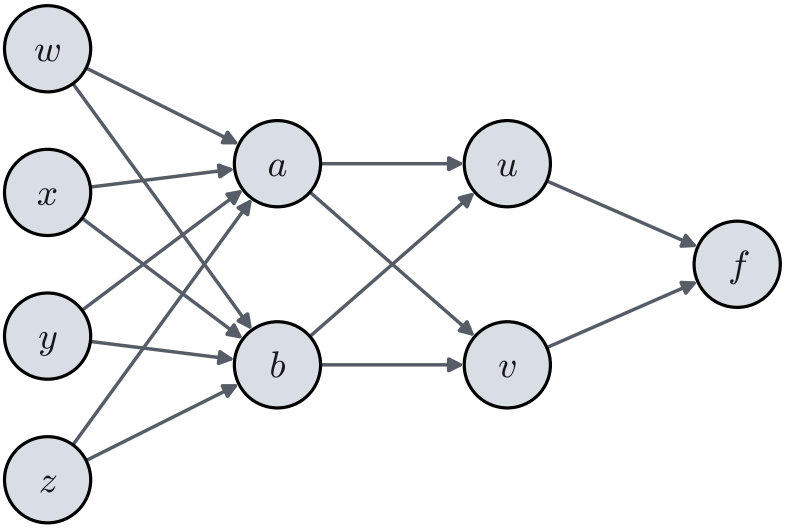


Figure 34.17: Equation 34.27 の依存関係グラフ。各ノードは変数を表し、各有向エッジは直接の関数依存関係を表す。

34.3.1.2 1 ステップを通る 2 つの経路

グラフ全体を扱う前に、最後の合成ステップ $f(u(a,b), v(a,b))$ だけを取り上げよう。f がどう応答するかを見るため、a を微小量 ϵ だけ動かす。2 つの中間ノードはそれぞれの偏微分に従ってこの摂動を受け、u は $u + \epsilon \frac{\partial u}{\partial a}$ へ、v は $v + \epsilon \frac{\partial v}{\partial a}$ へと移動する。これら両方の変化を、34.2.1.2 節で導入した f の一次線形近似に代入すると、次の式が得られる。

$$
f\big(u + \epsilon \tfrac{\partial u}{\partial a},\, v + \epsilon \tfrac{\partial v}{\partial a}\big) \approx f(u,v) + \epsilon \left[\frac{\partial f}{\partial u}\frac{\partial u}{\partial a} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial a} \right].
$$

ϵ の係数を読み取れば、a に関する合成関数の導関数、すなわち多変数の連鎖律が得られる。

$$
\frac{\mathrm{d}f}{\mathrm{d}a} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial a} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial a}.
$$

微分を表す 2 つの異なる記号に注目しよう。ここから先は、この 2 つを厳密に区別する。丸みを帯びた ∂ は、単一ノードを定義する関数について、他の引数を固定したまま直接の引数の 1 つに関してとった偏微分を表す。立体活字の d は**全**微分（total derivative）である $\mathrm{d}f/\mathrm{d}a$ を表し、a の下流にあるすべての変数が応答することを許す。（入力ノードについては、全微分は f を入力のみの関数とみなしたときの通常の偏微分と一致する。）幾何学的には、a が f に影響を及ぼす有向**経路**（path）は $a \to u \to f$ と $a \to v \to f$ の 2 つある。各経路はそのエッジに沿った微分の積として寄与し、全微分はすべての経路にわたってそれらの寄与を足し合わせる。

この導出は 1 変数の連鎖律の証明（34.1.2 節）と対応している。内側の関数の線形変化を、外側の関数の線形モデルに流し込んでいるからである。ただし、ここでは 2 つの変数が同時に動くため、f は (u, v) において単に偏微分を持つだけでなく、**微分可能**でなければならない。反例 $2xy/(x^2 + y^2)$（34.2.1.2 節）を振り返ろう。対角線 $x = y = t$ に沿って見ると、原点では両方の偏微分がゼロになるため、連鎖律は微分値 0 を $t = 0$ において予測するが、実際には関数はそこで 0 から 1 へと不連続に跳ぶ。

34.3.1.3 エッジ微分と全微分

同じ推論は、数学的帰納法によって任意の有向非巡回グラフへと拡張できる。あるノードの親の 1 つに関する**エッジ微分**（edge derivative）を、他のすべての親を固定したときの、その親に関するノードの関数の偏微分として定義しよう。このとき、出力 f の入力 x に関する全微分は次の式で与えられる。

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \sum_{\text{directed paths } x \to \cdots \to f} \; \prod_{\text{edges on the path}} (\text{edge derivative}). \tag{34.28}$$

言葉で言えば、x から f への各有向経路はそのエッジに沿った局所的な変化率を掛け合わせ、全微分はそれらの経路ごとの積を足し合わせたものになる。

スキップ接続（skip connection）を持つグラフを見ると、エッジ微分と全微分を決して混同してはならない理由がよくわかる。次の 5 ノードの系を考えよう。

$$\begin{aligned} p(x, y) &= x + y, \qquad q(x, y) = x - y, \\ r(p) &= p^2, \qquad s(q) = q^2, \\ f(r, p, s) &= r + p + s. \end{aligned} \tag{34.29}$$

ここでは、f は r と s に依存するだけでなく、スキップエッジを通じて p にも直接依存している（Figure 34.18）。その結果、入力 y は f に 3 つの異なる経路、すなわち $y \to p \to r \to f$、ショートカットである $y \to p \to f$、そして $y \to q \to s \to f$ に沿って到達する。局所的なエッジ微分は $\partial f/\partial r = \partial f/\partial p = \partial f/\partial s = 1$、$\partial r/\partial p = 2p$、$\partial s/\partial q = 2q$、$\partial p/\partial y = 1$、$\partial q/\partial y = -1$ である。3 つの経路の積を足し合わせると、次のようになる。

$$\begin{aligned} \frac{\mathrm{d}f}{\mathrm{d}y} &= \underbrace{1 \cdot 2p \cdot 1}_{y \to p \to r \to f} + \underbrace{1 \cdot 1}_{y \to p \to f} + \underbrace{1 \cdot 2q \cdot (-1)}_{y \to q \to s \to f} \\ &= 2p + 1 - 2q = 1 + 4y. \end{aligned}$$

すべてを f に代入すれば直接確認できる。すなわち $f = (x+y)^2 + (x+y) + (x-y)^2$ であり、その y に関する導関数は確かに $1 + 4y$ となる。$(x, y) = (1, 2)$ では $p = 3$、$q = -1$ であるから、3 つの経路の寄与は $6 + 1 + 2 = 9 = 1 + 4 \cdot 2$ となる。ノード p における明確な違いに注目

しよう。**エッジ**微分 $\partial f/\partial p$ は単なる 1（r を固定した場合）であるのに対し、**全**微分 $\mathrm{d}f/\mathrm{d}p$ はスキップエッジと r を経由する間接経路の両方を考慮するため、$2p + 1 = 7$ となる。この 2 つの量の混同は、多変数の連鎖律を適用するときに最もよく起こる誤りである。

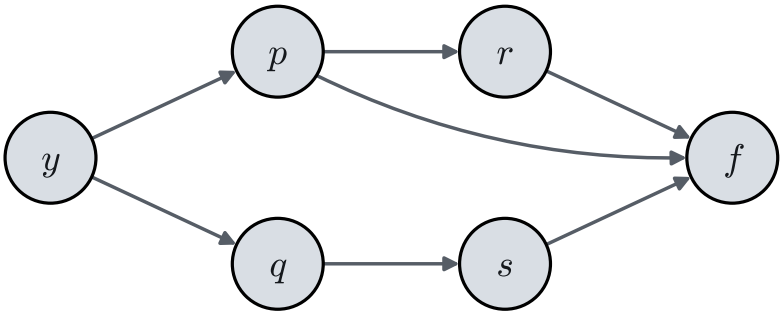


Figure 34.18: Equation 34.29 の依存関係グラフ。入力 y は f に 3 つの経路、すなわち $y \to p \to r \to f$、スキップエッジ $y \to p \to f$、および $y \to q \to s \to f$ によって到達する。

この経路和の視点は、深いアーキテクチャの内部を勾配がどのように伝わるかも説明してくれる。LSTM（12.1 節）のゲートや、7.4 節の残差ネットワーク（residual network）におけるスキップ接続のように経路を開閉する仕組みは、どの経路が初期の層へ勾配信号を運び戻すかを制御することで、学習を直接方向づける。

34.3.2 誤差逆伝播法

数学的な恒等式としては、経路にわたる和 Equation 34.28 は完全に一般的であり、あらゆる計算グラフのあらゆる導関数や微分値を与える。しかし、これを文字通りのアルゴリズムとして実行するのは絶望的である。深さとともに経路数が指数関数的に増大し得るからである。たとえば m 段の単純な連鎖であっても、各段に 2 個の並列ノードがあれば、始点から終点までにすでに 2^m 本の経路が含まれる。幸い、それらの経路は大きく重なり合っており、多数の経路が共有する部分積は 1 回だけ計算すればよい。こうした共有される部分積をどの順序でくくり出すかが、計算コストを決定する。

34.3.2.1 フォワードモード

4 入力の関数 Equation 34.27 を点 $(w, x, y, z) = (1, 1, -2, 1)$ で考え、和をくくり出す 2 つの方法を比較してみよう。**順伝播**（forward pass）では、トポロジカル順序に従って各ノードを評価し、$a = 1^2 = 1$、$b = 3^2 = 9$、$u = 10^2 = 100$、$v = (-8)^2 = 64$、そして $f = 164^2 = 26896$ を得る。これらのノード値から、局所的なエッジ微分がすべて次のように読み取れる。

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial v} = 2(u+v) = 328,$$
$$\frac{\partial u}{\partial a} = \frac{\partial u}{\partial b} = 2(a+b) = 20,$$
$$\frac{\partial v}{\partial a} = 2(a-b) = -16, \qquad \frac{\partial v}{\partial b} = -2(a-b) = 16,$$
$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial x} = \frac{\partial a}{\partial y} = \frac{\partial a}{\partial z} = 2(w+x+y+z) = 2,$$
$$\frac{\partial b}{\partial w} = \frac{\partial b}{\partial x} = 2(w+x-y-z) = 6, \qquad \frac{\partial b}{\partial y} = \frac{\partial b}{\partial z} = -6.$$

各エッジ微分は、ノードとその直接の親だけから計算される局所的な数値である。

$\mathrm{d}f/\mathrm{d}w$ を求めるには、w から f への 4 つの経路がすべて、まず a または b を通り、次に u または v を通ることに注目すればよい。入力 w からグラフに沿って順方向に経路をくくり出すと、次のようになる。

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}w} &= \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial a}\frac{\partial a}{\partial w} + \frac{\partial u}{\partial b}\frac{\partial b}{\partial w}\right) + \frac{\partial f}{\partial v}\left(\frac{\partial v}{\partial a}\frac{\partial a}{\partial w} + \frac{\partial v}{\partial b}\frac{\partial b}{\partial w}\right) \\ &= 328\,(20\cdot 2 + 20\cdot 6) + 328\,(-16\cdot 2 + 16\cdot 6) \\ &= 328\cdot 224 = 73472.\end{aligned}$$

括弧内の 2 つの和は、それぞれ $\mathrm{d}u/\mathrm{d}w = 160$ と $\mathrm{d}v/\mathrm{d}w = 64$、すなわち w に関する中間ノードの全微分にほかならない。このように 1 つの入力に対する感度をグラフに沿って順方向に伝播させる手法を**フォワードモード**と呼ぶ。この計算を素の Python で確かめてみよう。

```
w, x, y, z = 1, 1, -2, 1
# Forward pass: node values
a, b = (w + x + y + z)**2, (w + x - y - z)**2
u, v = (a + b)**2, (a - b)**2
f = (u + v)**2
print('a, b, u, v, f =', a, b, u, v, f)

# Edge derivatives
df_du, df_dv = 2*(u + v), 2*(u + v)
du_da, du_db, dv_da, dv_db = 2*(a + b), 2*(a + b), 2*(a - b), -2*(a - b)
da_dw, db_dw = 2*(w + x + y + z), 2*(w + x - y - z)

# Forward mode: push the dependence on w through the graph
du_dw, dv_dw = du_da*da_dw + du_db*db_dw, dv_da*da_dw + dv_db*db_dw
df_dw = df_du*du_dw + df_dv*dv_dw
print('df/dw =', df_dw)
```

```
a, b, u, v, f = 1 9 100 64 26896
df/dw = 73472
```

フォワードモードは $\mathrm{d}f/\mathrm{d}w = 73472$ を返すが、その中間微分は x、y、z のいずれに対しても再利用できない。$\mathrm{d}u/\mathrm{d}w$ も $\mathrm{d}v/\mathrm{d}w$ も、w だけに結びついているからである。ニューラルネットワ

ークの訓練では単一のスカラー損失をすべてのパラメータに関して微分する必要があるため、入力を 1 つずつ計算していたのでは共有構造の大部分が活かされないままになる。

34.3.2.2 随伴変数と逆伝播

解決策は視点を逆転させ、入力ではなく出力を固定することである。フォワードモードでは 1 つの入力を固定し、下流のすべてのノードがそれにどう応答するかを問うた。今度は単一の出力 f を固定し、上流のすべてのノードに対してそれがどう応答するかを問う。**随伴変数**（adjoint）$\bar{n}$ をノード n に対して定め、出力の全微分 $\mathrm{d}f/\mathrm{d}n$（すなわち n に関する全微分）を、n から f へのすべての有向経路にわたるエッジ微分の積の和とする。そのような経路はすべて、n からその直接の子 c のいずれかを通って出る必要があり、c から f への残りの経路の和は $\bar{c}$ になる。経路和 Equation 34.28 をこの最初のエッジごとにまとめると、単純な漸化式が得られる。

$$\bar{f} = 1, \qquad \bar{n} = \sum_{\text{children } c \text{ of } n} \bar{c}\,\frac{\partial c}{\partial n}. \tag{34.30}$$

右辺に現れるのは、局所的なエッジ微分と下流ノードの随伴変数だけである。$\bar{f} = 1$ から出発して逆トポロジカル順序でノードをたどれば、必要になる時点ですべての子ノードの随伴変数がすでに求まっている。たとえばスキップエッジを持つグラフ Equation 34.29 では、この漸化式から $\bar{r} = 1$ および $\bar{p} = \bar{r}\cdot 2p + \bar{f}\cdot 1 = 7$ が得られ、先に求めた全微分と一致する。同じスイープを 4 入力のグラフに適用すると、次のようになる。

- $\bar{u} = \bar{v} = 328$;
- $\bar{a} = \bar{u}\cdot 20 + \bar{v}\cdot(-16) = 1312$、$\bar{b} = \bar{u}\cdot 20 + \bar{v}\cdot 16 = 11808$
- $\bar{w} = \bar{x} = \bar{a}\cdot 2 + \bar{b}\cdot 6 = 73472$、$\bar{y} = \bar{z} = \bar{a}\cdot 2 - \bar{b}\cdot 6 = -68224$

このように、1 回の**逆伝播**（backward pass）によって、4 つの入力すべてに関する偏微分が一度に得られる（Figure 34.19）。前のセルで計算したエッジ微分を再利用して、この逆方向スイープを Python で実行してみよう。

```
da_dy, db_dy = 2*(w + x + y + z), -2*(w + x - y - z)  # also for x, z
# Backward pass: adjoints from the output toward the inputs
u_bar, v_bar = df_du, df_dv
a_bar = u_bar*du_da + v_bar*dv_da
b_bar = u_bar*du_db + v_bar*dv_db
print(f'adjoints: u {u_bar}, v {v_bar}, a {a_bar}, b {b_bar}')
w_bar = a_bar*da_dw + b_bar*db_dw  # x enters a, b like w
y_bar = a_bar*da_dy + b_bar*db_dy  # z enters a, b like y
print('gradient:', {'w': w_bar, 'x': w_bar, 'y': y_bar, 'z': y_bar})
```

```
adjoints: u 328, v 328, a 1312, b 11808
gradient: {'w': 73472, 'x': 73472, 'y': -68224, 'z': -68224}
```

w と x に関する微分が一致し、同様に y と z に関する微分も一致するのは、f が $w+x$ と $y+z$ という組み合わせを通じてのみ入力に依存しているからである。

演算回数を簡単に数えてみると、その利点が具体的になる。フォワードモードでは入力 1 つあたり 6 回の乗算（$\mathrm{d}u/\mathrm{d}w$、$\mathrm{d}v/\mathrm{d}w$、$\mathrm{d}f/\mathrm{d}w$ にそれぞれ 2 回ずつ）が必要であり、4 つの入力すべてでは合計 24 回となる。一方、逆伝播はグラフの共有された上側部分を 1 回しかたどらないため、$\bar{a}$ と $\bar{b}$ の計算に 4 回、その後は入力 1 つあたり 2 回、合計 12 回の乗算で済む。トイグラフでの 2 倍という差は控えめに見えるかもしれないが、入力数が増えるにつれてその差は急速に広が

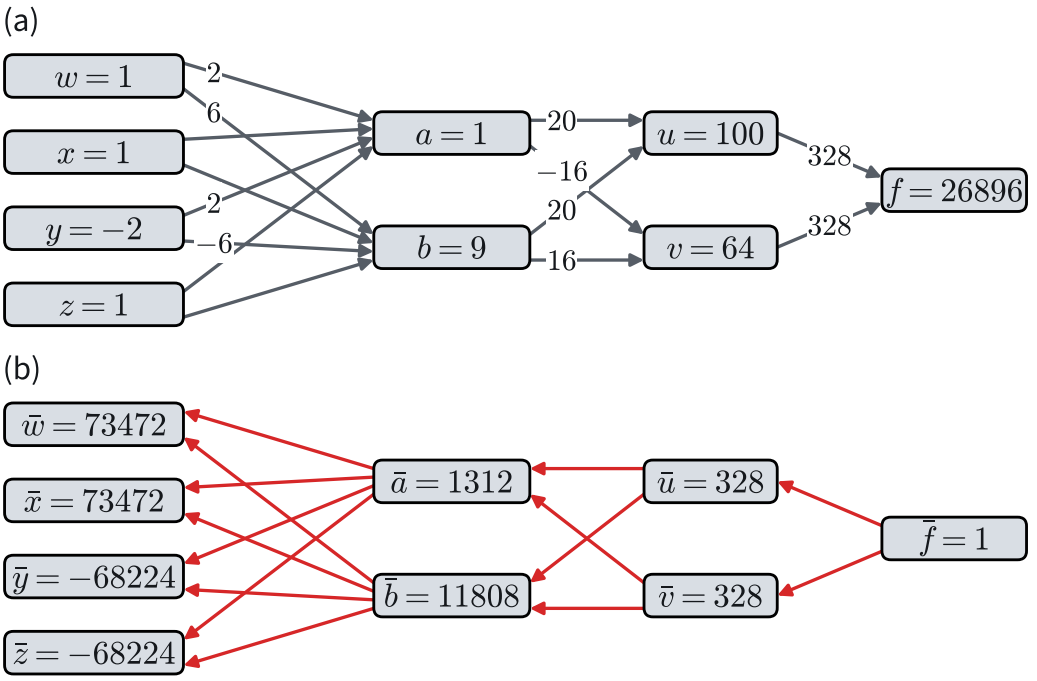


Figure 34.19: (a) 順伝播では、すべてのノード値と局所的なエッジ微分を記録する。(b) 逆伝播では、$\bar{f} = 1$ から始めて随伴変数を累積し、$\bar{a} = 328 \cdot 20 + 328 \cdot (-16) = 1312$ および $\bar{w} = 1312 \cdot 2 + 11808 \cdot 6 = 73472$ を得る。

る。入力が 1 つ増えるたびに、フォワードモードではグラフ全体のパスがまるごと 1 回余分にかかるのに対し、逆伝播では乗算がわずか 2 回増えるだけである。

34.3.2.3 フレームワークにおける誤差逆伝播法

出力から入力へと随伴変数を逆向きに伝播させて微分を計算する手法が**誤差逆伝播法**であり (Rumelhart et al. 1986)、4.3 節ではこれを多層パーセプトロン（multilayer perceptron）に適用している。現代のあらゆる自動微分（automatic differentiation）エンジンは、これと同じ 2 つのパスを実行する。すなわち、順伝播の間に中間値（またはエッジ微分を構成するのに必要な演算）を記録し、続いて逆伝播の間に随伴変数を累積するのである。同じ 4 つの入力に対して、手計算の結果をフレームワークの自動微分と照らし合わせてみよう。

```
def f_graph(wxyz):
    w, x, y, z = wxyz
    a, b = (w + x + y + z)**2, (w + x - y - z)**2
    u, v = (a + b)**2, (a - b)**2
    return (u + v)**2

print(jax.grad(f_graph)(jnp.array([1., 1., -2., 1.])))  # one backward pass
```

```
[ 73472.  73472. -68224. -68224.]
```

フレームワークは $(73472, 73472, -68224, -68224)$ を返し、手計算で求めた随伴変数と一致する。この一致が近似ではなく厳密なのは、すべての中間値が 2^{24} 未満の整数であり、単精度（single precision）で丸め誤差（rounding error）なしに表現できるからである。

ここまでのグラフではすべてのノードがスカラーであったため、乗算の回数も簡単な手計算で数えることができた。実際のニューラルネットワークでは各層がベクトルをベクトルへ写すため、エッジ微分は行列となり、それらの行列を掛ける順序はスカラー乗算の順序よりもはるかに重要になる。そのステップには 34.4 節で進む。

34.3.3 まとめ

計算グラフにおいて、入力に関する出力の全微分は、両者を結ぶすべての有向経路にわたって局所的なエッジ微分の積を足し合わせたものに等しい。ノードの随伴変数、すなわちそのノードに関する出力の全微分は、そこから出るすべての経路を集約したものである。それらの経路を最初に出るエッジごとにまとめると漸化式が得られ、これを出力から逆向きに実行すれば、1 回のパスですべての随伴変数を計算できる。この逆方向の累積が誤差逆伝播法である。これに対し、1 つの入力を順方向に伝播させるとその入力に関する微分しか求まらないため、スカラー損失を多数のパラメータに関して微分する計算は常に逆向きに行われる。

34.3.4 演習問題

1. [code] **2 つの tanh ユニットに対する逆伝播**。1 つの入力 x、2 つの隠れユニット、1 つの出力を持つネットワークが、ターゲット t に対して次を計算するとする。

$$h_i = \tanh(w_i x + b_i) \quad (i = 1, 2), \qquad f = v_1 h_1 + v_2 h_2, \qquad \ell = \tfrac{1}{2}(f - t)^2.$$

   1. ℓ の依存関係グラフを描け。素の Python で、すべての中間値を保存する順伝播と、ℓ の 6 個すべてのパラメータ $w_1, b_1, w_2, b_2, v_1, v_2$ に関する偏微分を返す逆伝播を実装

せよ。$x = 1.5$、$t = 0.5$、$(w_1, b_1, w_2, b_2, v_1, v_2) = (0.5, 0, -1, 0.2, 1, -0.5)$ において、それらの値を自動微分と照合せよ。6 個の偏微分すべてが再利用する中間微分はどれか。

2. 他の値はそのままにして、$w_1 = 5$、$x = 2$ とせよ。素の Python による逆伝播で $\partial \ell / \partial w_1$ を計算し、その経路に沿ったエッジ微分の積から値の大きさを説明せよ。単精度の自動微分が返す値と比較し、その違いを説明せよ。このことは、活性化前の値が大きいユニットの訓練に対して何を示唆するか。
3. $w_1 = w_2$、$b_1 = b_2$、$v_1 = v_2$ とせよ。逆伝播から両方のユニットが同一の勾配を受け取り、その結果、勾配降下法が毎ステップで両者を同一のまま保つことを示せ。このことは、隠れユニットの初期化に対して何を示唆するか。

2. **残差スタックを通る経路。**●7.4 節の残差ネットワークの 1 次元モデルにおいて、スカラーブロックは $x_{l+1} = x_l + g_l(x_l)$（$l = 0, \dots, m-1$、ただし g_l は微分可能な関数）を計算する。

1. 連鎖律を用いて、$\partial x_m / \partial x_0$ を m 個の因子の積として表せ。
2. その積を展開し、各項を依存関係グラフにおける x_0 から x_m への有向経路に対応させよ。経路は全部で何本あり、そのうちちょうど k 個の関数 g_l を通るものは何本あるか。経路の数と、逆伝播が $\partial x_m / \partial x_0$ を計算するのに要する乗算の回数を比較せよ。
3. 訓練済みの $m = 54$ 個の残差ブロックからなるネットワークにおいて、Veit et al. (2016) は、勾配のほぼすべてが 5 から 17 個のブロックを通る経路に沿って流れることを見いだした。これらの経路は全経路のどれだけの割合を占めるか。次に、すべてのブロックで $g_l'(x_l) = c > 0$ であると仮定する。どのような c の値のときに、約 11 個のブロックを通る経路が $\partial x_m / \partial x_0$ に最も大きく寄与するか。また、そのとき 5 から 17 個のブロックを通る経路は微分全体のどれだけの割合を担うか。

3. [code] **単精度における勾配チェック。**●勾配チェックは、逆伝播を中心差分 $\big(f(w + h, x, y, z) - f(w - h, x, y, z)\big)/(2h)$ と比較する手法である。f を Equation 34.27 の関数とし、$(w, x, y, z) = (1, 1, -2, 1)$ における厳密な偏微分は $\partial f / \partial w = 73472$ であるとする。

1. NumPy で f を実装し、この点における中心差分を $h = 10^{-1}, 10^{-2}, \dots, 10^{-12}$ に対して `float64` と `float32` のそれぞれで計算せよ。両方の精度について相対誤差を h に対して両対数軸でプロットし、それぞれの最適な h と最小の相対誤差を読み取れ。
2. 中心差分の誤差は、オーダー h^2 の打ち切り誤差の項とオーダー $\varepsilon_{\text{mach}} |f| / h$ の丸め誤差の項の和となる。ここで計算機イプシロン $\varepsilon_{\text{mach}}$ は、約 2.2×10^{-16}（`float64`）および約 1.2×10^{-7}（`float32`）である。最適な h と最小の相対誤差は、`float64` から `float32` に変えたとき何倍変化するはずか。予測される倍率をプロットと比較せよ。
3. `float32` での勾配チェックにおいて、正しい逆伝播を誤って棄却しないためには相対誤差の許容誤差をどの程度に設定できるか。勾配チェックが倍精度で実行されるのはなぜか。
4. `float32` の推定値が最小領域の h に対して厳密にゼロになるのはなぜか。

34.4 行列微分

直前の 2 節では、損失をスカラー変数の関数として扱った。34.2 節では偏微分を並べた列として勾配を構成し、34.3 節ではノードごとに 1 つの数値を持つグラフ上の逆伝播によってその列を計算した。しかし、ニューラルネットワークはスカラーでは記述されない。各層はベクトルをベクトルへ写し、その重みは行列をなす。ベクトルや行列に関する損失の導関数とは何であり、どうすれば層全体に対して一度に計算できるのだろうか。

手計算できるほど小さなネットワークを見れば、何が必要かがわかる。2 つの入力、2 つの ReLU 隠れユニット、1 つの出力を持つ次のネットワークを考えよう。

$$g(\mathbf{x}) = \mathbf{w}_2^\top \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}), \qquad \mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ 3 & -1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ -4 \end{pmatrix}, \quad \mathbf{w}_2 = \begin{pmatrix} 2 \\ -1 \end{pmatrix},$$

ここで、φ は各成分に $\max(0, \cdot)$ を適用する写像である。$\mathbf{x} = (1, 1)$ において活性化前の値は $\mathbf{W}_1 \mathbf{x} + \mathbf{b} = (3, -2)$ となるため、2 番目の隠れユニットは非活性となり、隠れベクトルは $(3, 0)$、出力は $g = 6$ となる。訓練ステップでは、この出力に関して 3 つの問いが生じる。第 1 に、入力が微小な δ だけ動くと、出力は $2\delta_1 + 4\delta_2$ だけ変化する。この係数は 2 つの数値からなる行であり、列である勾配の転置になっている。第 2 に、重み行列が $\Delta \mathbf{W}_1$ だけ動くと、出力は $2\,\Delta W_{11} + 2\,\Delta W_{12}$ だけ変化し、第 2 行からはまったく影響を受けない。したがって、行列に関する導関数はそれ自体が行列 $\left(\begin{smallmatrix} 2 & 2 \\ 0 & 0 \end{smallmatrix}\right)$ となる。第 3 に、隠れ層 $\mathbf{x} \mapsto \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b})$ はベクトルをベクトルへ写す。その導関数はスカラーの傾きでも勾配ベクトルでも表せないが、出力はまさにこの層を介してのみ $\mathbf{W}_1$ に依存している。

このような導関数は、手計算では驚くほど間違えやすい。正方の重み行列で転置を落としても、次元は一致したままでエラーも発生せず、ただ訓練が狂ってしまう。行列微分の公式集には何百もの項目が並んでいるが、公式を引くだけではこの種のうっかりミスはほとんど防げない。

本節では、これら 3 つの問いすべてに、単一の数学的対象と単一の原理で答える。ベクトル値写像の導関数は、その写像の最良線形近似として定義される行列、すなわち**ヤコビ行列**（Jacobian）である。写像の合成はヤコビ行列の積に対応するため、ネットワークの導関数は層ごとに 1 つずつの行列の積となる。さらに、層が必要とするあらゆる勾配（線形形式や二次形式、最小二乗法、行列積、ソフトマックス）は、1 つの手法から導かれる。関数の一次の変化を書き下し、その係数として勾配を読み取るのである。同じ手法から、4.3 節で用いた各層の逆伝播だけでなく、$\mathbf{p} - \mathbf{y}$ という 3.1.2.2 節の分類器の勾配も得られる。

特に覚えておくべき結果は 3 つある。ヤコビ行列の積 Equation 34.36、勾配は常にその変数と同じ形状を持つという規則、そして行列積の逆伝播則 Equation 34.39 である。初読時には連鎖律の証明を読み飛ばしても構わないが、全微分法（differential method、34.4.3.1 節）は読み飛ばせない。以降の議論はすべてこれに依拠するからである。

本節全体を通して、以下のインポートを使用する。

```
import numpy as np
import jax
from jax import numpy as jnp
```

34.4.1 ヤコビ行列

34.4.1.1 最良線形近似

$\mathbf{x} = (1, 1)$ の近傍では、冒頭のネットワークの隠れ層はアフィンであるため、その線形近似は厳密に一致し、調べるべき誤差が残らない。そこでまずは、本節と次節で共通の例として再利用する非線形な写像から始め、定義が整ったところで再びネットワークに戻ることにしよう。

$$\mathbf{f}(x, y) = \begin{pmatrix} x + \sin y \\ y + x^2/2 \end{pmatrix}, \qquad \mathbf{x}_0 = (0.5,\ 0.5).$$

$\mathbf{x}_0$ において、この写像の値は $\mathbf{f}(\mathbf{x}_0) = (0.9794,\, 0.6250)$ である。2 つの出力はそれぞれ 2 つの偏微分を持ち、$\partial f_1/\partial x = 1$、$\partial f_1/\partial y = \cos y = 0.8776$、$\partial f_2/\partial x = x = 0.5$、$\partial f_2/\partial y = 1$ となる。これら 4 つの数値を、出力ごとに 1 行、入力ごとに 1 列を持つ行列に並べる。

$$\mathbf{J}(\mathbf{x}_0) = \begin{pmatrix} 1 & \cos 0.5 \\ 0.5 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0.8776 \\ 0.5 & 1 \end{pmatrix}.$$

第 j 列は、入力 j が動いたときに両方の出力がどう応答するかを記録している。この行列は $\mathbf{f}$ の $\mathbf{x}_0$ における導関数である。すなわち、$\mathbf{f}(\mathbf{x}_0 + \delta) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J}\delta$ が微小な δ に対して成り立ち、他のどの行列もこれほど精密には $\mathbf{f}$ を近似しない。

1 変数の場合、導関数は最良の直線近似 $f(x+\delta) \approx f(x) + f'(x)\,\delta$ の傾きを与える。ベクトル値への一般化も同じ局所近似に基づく。写像 $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ が点 $\mathbf{x}$ で**微分可能**（differentiable）であるとは、ある行列 $\mathbf{J} \in \mathbb{R}^{m \times n}$ が存在して次式を満たすことをいう。

$$\mathbf{f}(\mathbf{x} + \delta) = \mathbf{f}(\mathbf{x}) + \mathbf{J}\,\delta + o(\|\delta\|), \tag{34.31}$$

これは、線形モデル $\mathbf{f}(\mathbf{x}) + \mathbf{J}\delta$ の誤差を $\|\delta\|$ で割ったものが、$\delta \to \mathbf{0}$ のとき 0 に収束することを意味する。このような行列が存在するとき、それは一意に定まる。これを**ヤコビ行列** $\mathbf{J}_{\mathbf{f}}(\mathbf{x})$ と呼び、その各成分は単なる偏微分となる。

$$[\mathbf{J}_{\mathbf{f}}]_{ij} = \frac{\partial f_i}{\partial x_j}. \tag{34.32}$$

その理由を見るには、摂動 $\delta = t\,\mathbf{e}_j$（座標軸 j に沿った微小ステップ）を Equation 34.31 に代入し、第 i 成分に注目すればよい。t で割って $t \to 0$ とすると、$\partial f_i/\partial x_j$ が (i,j) 成分として残り、これは出発点とした行列 $\mathbf{J}$ の選び方によらない。したがって、偏微分は $\mathbf{J}$ の各成分を与えるが、それを定義するのは線形近似である。

偏微分の存在だけでは、微分可能性は保証されない。$f(x,y) = 2xy/(x^2+y^2)$（ただし $f(0,0) = 0$）という 34.2.1.2 節の関数を考えよう。原点では両方の偏微分が 0 であるにもかかわらず、f は値 1 を対角線 $x = y$ 上のいたるところでとる。ヤコビ行列の唯一の候補はゼロ行ベクトルであるが、対角線上の近似誤差は 1 のままであり、δ をどれほど小さくしても変わらないため、f は原点でヤコビ行列を持たない。このような振る舞いを排除するには、偏微分の連続性が十分条件となる。すなわち、すべての偏微分 $\partial f_i/\partial x_j$ が $\mathbf{x}$ の近傍で存在し、かつ $\mathbf{x}$ で連続であれば、$\mathbf{f}$ は $\mathbf{x}$ で微分可能である (Rudin 1976)。

Figure 34.20 は、この定義を幾何学的に示している。$\mathbf{x}_0$ の近傍では、写像はそのヤコビ行列のように振る舞う。半径 r で $\mathbf{x}_0$ を中心とする円は、楕円 $\mathbf{f}(\mathbf{x}_0) + \mathbf{J}\delta$（$\|\delta\| = r$）へとほぼ写され、その半軸の長さは r と $\mathbf{J}$ の特異値の積になる（33.5 節で示したように、任意の行列は単位円を特異値を半軸とする楕円へ写す）。ここでは特異値が 1.706 と 0.329 であるため、半径 0.35 の円は半軸 0.60 と 0.12 の楕円になる。真の像と楕円との最大のずれは 0.061（$r = 0.35$ のとき）および 0.015（$r = 0.175$ のとき）であり、半径を半分にするとずれは 4 分の 1 に縮まる。

定義は線形モデルの誤差を $\|\delta\|$ で割った比が 0 に収束することを保証するが、その速さまでは述べていない。摂動を 10 分の 1 ずつ縮めながら、その比を調べてみよう。

```
def f(v):                                  # the running map, R^2 -> R^2
    x, y = v
    return np.array([x + np.sin(y), y + x**2 / 2])
```

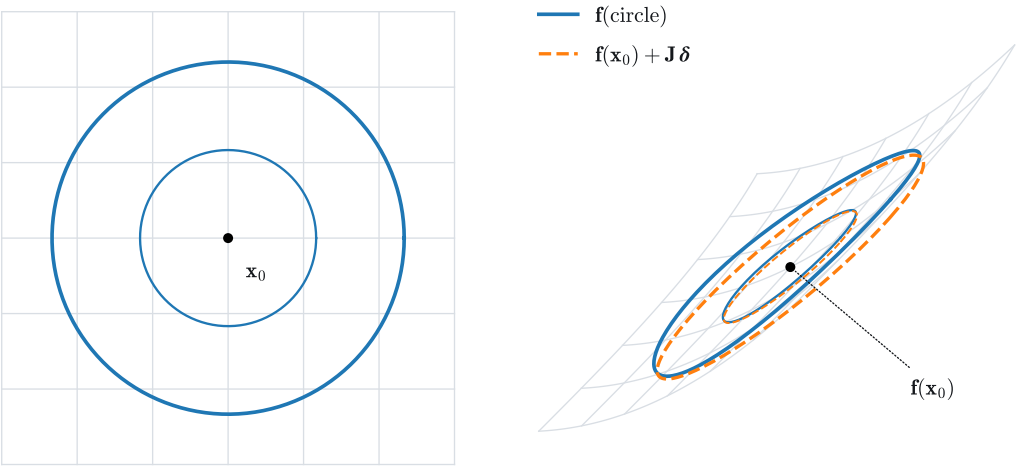


Figure 34.20: 微分可能な写像は局所的にそのヤコビ行列と一致する。半径 0.35 と 0.175 で $\mathbf{x}_0 = (0.5, 0.5)$ を中心とする 2 つの円（左）と、$\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ によるそれらの像（右、青）および楕円 $\mathbf{f}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0)\delta$（オレンジ、破線）の比較。各楕円の半軸は、半径に特異値 1.71、0.33（行列 $\mathbf{J}$ の特異値）を掛けた値である。半径を半分にすると、像と楕円の最大のずれは 0.061 から 0.015 へと縮まり、剰余項 $o(\|\delta\|)$ の振る舞いを示す。

```
def J(v):                                   # its Jacobian, entry by entry
    x, y = v
    return np.array([[1.0, np.cos(y)], [x, 1.0]])

x0 = np.array([0.5, 0.5])
for s in [1, 0.1, 0.01, 0.001]:
    delta = s * np.array([0.2, 0.1])
    err = np.linalg.norm(f(x0 + delta) - f(x0) - J(x0) @ delta)
    print(f'|delta| = {np.linalg.norm(delta):.1e}   '
          f'error / |delta| = {err / np.linalg.norm(delta):.1e}')
```

```
|delta| = 2.2e-01   error / |delta| = 9.0e-02
|delta| = 2.2e-02   error / |delta| = 9.0e-03
|delta| = 2.2e-03   error / |delta| = 9.0e-04
|delta| = 2.2e-04   error / |delta| = 9.0e-05
```

比は 0.09 から 0.009、0.0009、0.00009 へと減少する。0 に収束しており、定義の剰余項 $o(\|\delta\|)$ が確認できる。さらに、この比は 10 分の 1 に $\|\delta\|$ を縮めるごとに 10 分の 1 に減少していることにも注目したい。つまり誤差そのものは 100 分の 1 に減少し、$O(\|\delta\|^2)$ となっている。定義が保証するのは最初の性質だけであり、より速い二次の減衰が成り立つのは $\mathbf{f}$ が連続な二階導関数を持つからである。

34.4.1.2 勾配、ヘッセ行列、導関数の形状

34.2 節で扱った勾配とヘッセ行列（Hessian）は、どちらもヤコビ行列の特別な場合である。この観点に立つと、手計算のたびに生じる形状の疑問、すなわち導関数は行なのか列なのか、また

行列に関する導関数はどう並べるべきなのかという問題に決着がつく。

スカラー値関数 $f:\mathbb{R}^n\to\mathbb{R}$（$m=1$）に対して、ヤコビ行列は 1 行のベクトル $\partial f/\partial \mathbf{x} = [\partial f/\partial x_1,\dots,\partial f/\partial x_n]$ となり、Equation 34.31 は一次展開 $f(\mathbf{x}+\delta)\approx f(\mathbf{x})+(\partial f/\partial \mathbf{x})\,\delta$ となる。34.2 節では勾配を列ベクトルとして書いた。両者は単に互いの転置である。

$$\nabla_{\mathbf{x}} f = \Big(\frac{\partial f}{\partial \mathbf{x}}\Big)^{\top}, \tag{34.33}$$

本書では、それぞれの目的に応じて両方を使い分ける。ベクトル値写像のヤコビ行列は出力ごとに 1 行を持つ $m\times n$ 行列とし、後述する連鎖律が転置なしですっきりと合成できるようにする。一方、スカラー関数の勾配 $\nabla_{\mathbf{x}} f$ は $\mathbf{x}$ の形状に合わせた列ベクトルとし、オプティマイザがそのまま減算できるようにする。冒頭のネットワークでは、$\partial g/\partial \mathbf{x} = (2,\,4)$ が 1×2 のヤコビ行列であり、$\nabla_{\mathbf{x}} g = (2,4)^{\top}$ が列ベクトルとしての勾配である。

行列 $\mathbf{W}\in\mathbb{R}^{p\times q}$ のスカラー関数には、もう 1 つ規約が必要になる。その勾配 $\nabla_{\mathbf{W}} f$ を、$p\times q$ 行列で各成分が $\partial f/\partial W_{ij}$ であるものとして定義し、$\mathbf{W}$ の形状に一致させる。すると一次展開は次のように書ける。

$$f(\mathbf{W}+\mathbf{E}) = f(\mathbf{W}) + \langle \nabla_{\mathbf{W}} f,\,\mathbf{E}\rangle + o(\|\mathbf{E}\|), \qquad \langle \mathbf{A},\mathbf{B}\rangle = \operatorname{tr}(\mathbf{A}^{\top}\mathbf{B}) = \sum_{ij} A_{ij}B_{ij}, \tag{34.34}$$

ここで、内積 $\langle\cdot,\cdot\rangle$ は同じ形状の 2 つの行列を成分ごとに掛けて総和をとる演算であり、行列をベクトルに平坦化したときの通常のドット積と同じである。言葉で言えば、f の一次の変化はすべての成分にわたる「感度 × 微小変化」の和である。冒頭の例では、$\nabla_{\mathbf{W}_1} g$ は 2×2 行列 $\left(\begin{smallmatrix}2&2\\0&0\end{smallmatrix}\right)$ であり、$\langle \nabla_{\mathbf{W}_1} g, \Delta\mathbf{W}_1\rangle = 2\,\Delta W_{11} + 2\,\Delta W_{12}$ から先ほど求めた変化量が復元される。ここから、本節のあらゆる公式に対する 1 つの妥当性確認が得られる。すなわち、勾配は常にその変数と同じ形状を持つ。導出した勾配の形状が合わなければ、転置が抜けているのである。

もう一度微分すると、ヘッセ行列が現れる。勾配場 $\nabla f:\mathbb{R}^n\to\mathbb{R}^n$ はそれ自体がベクトル値写像であり、そのヤコビ行列は二階偏微分の行列、すなわちヘッセ行列 $\mathbf{H}_f$ であって、その成分は $[\mathbf{H}_f]_{ij} = \partial^2 f/\partial x_i\partial x_j$ で与えられる。厳密に言えば、$[\mathbf{J}_{\nabla f}]_{ij} = \partial^2 f/\partial x_j\partial x_i$ は 2 回の微分の順序が逆になるが、交差偏微分が連続である限り両者は一致し、$\mathbf{H}_f$ は対称になる（34.2 節のクレロー・シュワルツの定理）。

34.4.2 ヤコビ行列の連鎖律

ニューラルネットワークは層の合成である。スカラーのグラフ上では、連鎖律は入力から出力へ至るすべての経路にわたって導関数の積を足し合わせた（34.3.1 節）。各層がベクトル値写像になると、その経路上の和は行列積になる。

34.4.2.1 行列積としての合成

命題（連鎖律）$\mathbf{f}:\mathbb{R}^n\to\mathbb{R}^p$ が $\mathbf{x}$ で微分可能であり、$\mathbf{g}:\mathbb{R}^p\to\mathbb{R}^m$ が $\mathbf{f}(\mathbf{x})$ で微分可能であれば、合成写像 $\mathbf{g}\circ\mathbf{f}$ は $\mathbf{x}$ で微分可能であり、次式が成り立つ。

$$\mathbf{J}_{\mathbf{g}\circ\mathbf{f}}(\mathbf{x}) = \mathbf{J}_{\mathbf{g}}\big(\mathbf{f}(\mathbf{x})\big)\,\mathbf{J}_{\mathbf{f}}(\mathbf{x}). \tag{34.35}$$

証明 $\mathbf{y} = \mathbf{f}(\mathbf{x})$、$\mathbf{A} = \mathbf{J}_{\mathbf{f}}(\mathbf{x})$、$\mathbf{B} = \mathbf{J}_{\mathbf{g}}(\mathbf{y})$ とおく。基本的な考え方は、単に $\mathbf{f}$ の線形モデルを $\mathbf{g}$ の線形モデルに代入することである。Equation 34.31 より、$\mathbf{f}(\mathbf{x}+\delta) = \mathbf{y} + \mathbf{A}\delta + o(\|\delta\|)$ が成り立つ。この変位を $\eta = \mathbf{A}\delta + o(\|\delta\|)$ とおき、$\mathbf{g}$ に渡すと次式を得る。

$$\mathbf{g}\big(\mathbf{f}(\mathbf{x}+\delta)\big) = \mathbf{g}(\mathbf{y}) + \mathbf{B}\,\eta + o(\|\eta\|) = \mathbf{g}(\mathbf{y}) + \mathbf{B}\mathbf{A}\,\delta + o(\|\delta\|).$$

2 つ目の等号は 2 つの事実に基づいている。第 1 に、$o(\|\delta\|)$ の剰余項に定数行列 $\mathbf{B}$ を掛けても $o(\|\delta\|)$ のままである。第 2 に、$\|\eta\| \le \|\mathbf{A}\|\,\|\delta\| + o(\|\delta\|)$ は $\|\delta\|$ の定数倍で上から抑えられるため、剰余項 $o(\|\eta\|)$（$\mathbf{g}$ の展開に現れる項）もまた $o(\|\delta\|)$ となる。純粋に代数的ではないステップはここだけである。したがって δ の係数は Equation 34.31 を満たす行列となり、一意性により合成写像のヤコビ行列 $\mathbf{J}_{\mathbf{g}\circ\mathbf{f}} = \mathbf{B}\mathbf{A}$ に一致する。$\blacksquare$

この積の 1 つの成分に注目すれば、スカラーの場合の経路上の和が復元される。すなわち $[\mathbf{B}\mathbf{A}]_{ik} = \sum_{j} (\partial g_i/\partial y_j)(\partial f_j/\partial x_k)$ は、すべての中間変数 y_j にわたって、x_k から g_i への各経路に沿った導関数の積を足し合わせている。

34.4.2.2 ネットワークのヤコビ行列

冒頭のネットワークに戻ると、これは 3 つの写像の合成とみなせる。アフィン変換 $\mathbf{a} = \mathbf{W}_1\mathbf{x} + \mathbf{b}$、ReLU による活性化値 $\mathbf{h} = \varphi(\mathbf{a})$、そして線形出力 $g = \mathbf{w}_2^{\top}\mathbf{h}$ である。第 1 の写像はもともとアフィンであるため、そのヤコビ行列は $\mathbf{W}_1$ である。バイアスは傾きを変えずに値を平行移動させるにすぎない。ReLU は各成分に独立に作用し、傾きは 1（$a_i > 0$ のとき）および 0（$a_i < 0$ のとき）となるため、そのヤコビ行列は対角行列 $\operatorname{diag}(\mathbf{m})$（これらの傾きからなる**マスク** $\mathbf{m}$ を対角に並べた行列）となる。最後に、出力写像のヤコビ行列は $\mathbf{w}_2^{\top}$ である。$\mathbf{x} = (1, 1)$ においては $\mathbf{a} = (3, -2)$、$\mathbf{m} = (1, 0)$ であり、連鎖律により 3 つの因子の積をとると次のようになる。

$$\mathbf{J}_{g}(\mathbf{x}) = \mathbf{w}_2^{\top}\operatorname{diag}(\mathbf{m})\,\mathbf{W}_1 = \begin{pmatrix} 2 & -1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 4 \end{pmatrix}.$$

マスクによって $\mathbf{w}_2^{\top}$ の第 2 成分が 0 になるため、$\mathbf{W}_1$ の第 2 行は出力に一切届かない。すなわち、非活性なユニットの重みは導関数にまったく寄与しない（Figure 34.21）。得られた行 $(2, 4)$ は、冒頭で見た変化量 $2\delta_1 + 4\delta_2$ と一致している。入力が変わればマスクも変わり、ヤコビ行列も変化する。たとえば $\mathbf{x} = (2, -3)$ では活性化前の値が $(-4, 5)$ となり、マスクは $(0, 1)$ に反転して $\mathbf{J}_g = (0,\ -1)\,\mathbf{W}_1 = (-3,\ 1)$ となる。ReLU は区分線形であるため、マスクが変わらないほど小さなステップをとれば、各軸に沿った有限差分によってこれらの成分が厳密に再現される（演習問題 1）。

あらゆる多層パーセプトロン（multilayer perceptron、MLP）は、同じ 2 種類の因子から構成される。全結合層 $\mathbf{x} \mapsto \mathbf{W}\mathbf{x}$ のヤコビ行列は $\mathbf{W}$ である。要素ごとの活性化関数 $\varphi(\mathbf{x}) = (\varphi(x_1), \dots, \varphi(x_n))$ は異なる座標どうしを結合しないため、そのヤコビ行列は対角行列 $\mathbf{J}_{\varphi}(\mathbf{x}) = \operatorname{diag}(\varphi'(x_1), \dots, \varphi'(x_n))$ となる。ここでは φ がすべての x_i で微分可能であると仮定している。ReLU は 0 で導関数を持たないが、フレームワークでは慣例として固定値を割り当てる（34.1.6 節）。深さ K のネットワークは合成 $\mathbf{f} = \mathbf{f}_K \circ \cdots \circ \mathbf{f}_1$ であり、Equation 34.35 を繰り返し適用すると、その導関数は 1 つの積になる。

$$\mathbf{J} = \mathbf{J}_K\,\mathbf{J}_{K-1}\cdots\mathbf{J}_1, \tag{34.36}$$

ここで各因子は、それぞれの層への入力において評価される。全結合層と活性化関数が交互に並ぶ場合、これは $\mathbf{J} = \operatorname{diag}(\varphi_K')\,\mathbf{W}_K \cdots \operatorname{diag}(\varphi_1')\,\mathbf{W}_1$ となり、重み行列と対角行列が交互に現れ

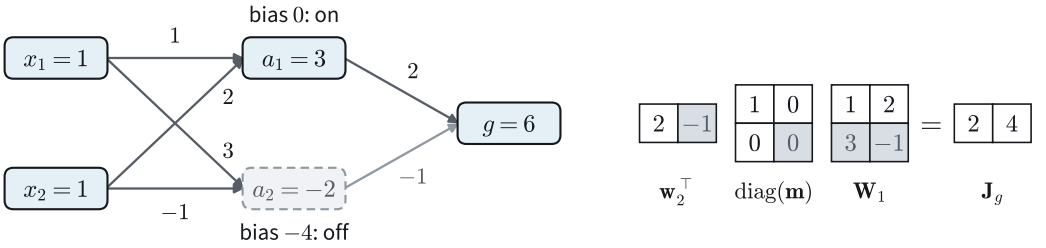


Figure 34.21: 2-2-1 ReLU ネットワークの $\mathbf{x} = (1, 1)$ におけるヤコビ行列は 3 つの行列の積である。第 2 ユニットの活性化前の値は -2 であるためマスク成分が 0 となり、$\mathbf{W}_1$ の第 2 行は出力に届かない。結果として $\mathbf{J}_g = \mathbf{w}_2^\top \mathrm{diag}(1, 0)\,\mathbf{W}_1 = (2,\ 4)$ となる。

る。ベクトルに対角行列を掛ける演算は各成分を定数倍するだけなので、各活性化因子の計算コストは $O(n)$ 回の演算で済み、$O(n^2)$ にはならない。

ここで、本節の後半でも再利用する 2-2-1 ネットワークをコードで定義し、積 $\mathbf{w}_2^\top \mathrm{diag}(\mathbf{m})\mathbf{W}_1$ を自動微分（automatic differentiation）が返す勾配と比較してみよう。

```
W1, b = jnp.array([[1.0, 2.0], [3.0, -1.0]]), jnp.array([0.0, -4.0])
w2, x = jnp.array([2.0, -1.0]), jnp.array([1.0, 1.0])

def g(x, W1, b, w2):                          # the 2-2-1 ReLU network
    return w2 @ jax.nn.relu(W1 @ x + b)

m = (W1 @ x + b > 0).astype(jnp.float32)      # the ReLU mask
print('by hand    :', np.asarray(w2 @ jnp.diag(m) @ W1))
print('by autograd:', np.asarray(jax.grad(g)(x, W1, b, w2)))
```

```
by hand    : [2. 4.]
by autograd: [2. 4.]
```

予想どおり、どちらの行も $(2, 4)$ を出力する。

積 Equation 34.36 からは、深いネットワークを通過するうちに導関数がどれほど大きく、あるいは小さくなりうるかもわかる。すべての因子が同じ行列 $\mathbf{W}$ である場合（活性化関数を無視して、毎ステップ同一の重み行列を適用するリカレントネットワークのような場合）、積は $\mathbf{W}^K$ に簡約される。対角化可能な行列 $\mathbf{W} = \mathbf{V}\Lambda\mathbf{V}^{-1}$ に対しては $\mathbf{W}^K = \mathbf{V}\Lambda^K\mathbf{V}^{-1}$ が成り立つため、大きな K に対してそのノルムは $|\lambda_{\max}|^K$ のように増大または縮小する。ここで $\lambda_{\max}$ は絶対値が最大の固有値であり、$0.9^{50} \approx 0.005$ であるのに対し、$1.1^{50} \approx 117$ となる。この指数関数的なスケーリングこそが、深層ネットワークやリカレントネットワークにおける勾配消失・勾配爆発問題である（4.4 節、8.5 節）。ただし、$|\lambda_{\max}|^K$ が表すのは長期的な変化率にすぎない。$\mathbf{V}$ の固有ベクトルが直交から程遠い場合、積は最終的に減衰する前に何ステップにもわたって増大しうるし（33.4.2.3 節）、異なる因子の積を支配するのは固有値ではなく特異値である（33.4.2.5 節）。

しかし、訓練という観点では Equation 34.36 に重要な限界がある。この式が与えるのは**入力**に関する導関数なのである。重み行列 $\mathbf{W}_k$ は変数ではなく固定された因子として現れているため、重みが変化したときに出力がどう変化するかはまだわからない。重み勾配を計算するには層ごとにあと 1 つの規則が必要であり、34.4.3.2 節でそれを導出して同じネットワークに適用する。

34.4.2.3 乗算の順序

行列積は結合法則を満たすため、積 Equation 34.36 はどのような順序で括弧を付けても厳密に同じ行列になる。しかし、括弧の付け方によって計算コストにはネットワークの幅に比例して拡大するほどの莫大な差が生じる。訓練で求めたいのはスカラー損失 ℓ の勾配であるため、連鎖の左端は行ベクトル、すなわち $1 \times h$ のヤコビ行列 $\mathbf{u}^\top = \partial \ell / \partial \mathbf{h}_K$（最終層の出力 $\mathbf{h}_K$ に関する損失のヤコビ行列）から始まる。左から順に掛けていけば、中間結果は常に行ベクトルのまま保たれ、すべてのステップが行列・ベクトル積になる。反対に右から掛けていくと、最後にようやく行ベクトルが掛けられるまで、高コストな行列・行列積をずっと作り続けることになる。低コストな順序とは、すべてのステップを行列・ベクトル積に保つ順序である。

演算回数を簡単に数えれば、この差は具体的になる。$\mathbf{u}^\top \mathbf{J}_3 \mathbf{J}_2 \mathbf{J}_1$ の各因子の形状を $1 \times h$、$h \times h$、$h \times h$、$h \times n$ とし、$h = n = 1000$ とおく。なお、$p \times q$ 行列と $q \times r$ 行列の積には pqr 回の乗算を要する。左から掛ける場合、3 回の行列・ベクトル積を行うため、乗算回数は合計 $h^2 + h^2 + hn = 3 \times 10^6$ 回である。右から掛ける場合、$\mathbf{J}_2 \mathbf{J}_1$ と続く $\mathbf{J}_3(\mathbf{J}_2 \mathbf{J}_1)$ の計算にそれぞれ $h^2 n = 10^9$ 回の乗算を要し、最後の $\mathbf{u}^\top$ との積で $hn = 10^6$ 回が加わって合計 2.001×10^9 回となる。300 万回対 20 億回であり、約 670 倍、すなわち $2h/3$ 倍の差がある（Figure 34.22）。

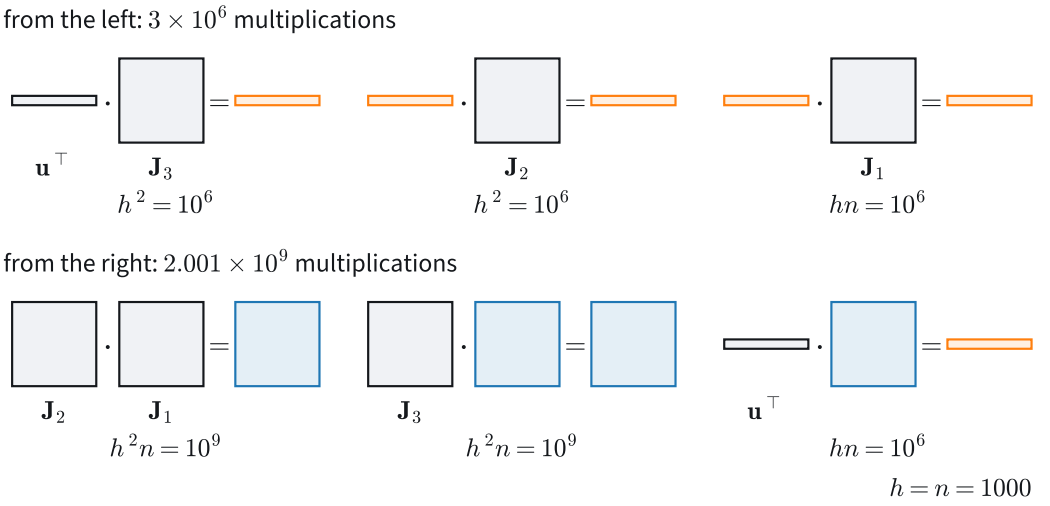


Figure 34.22: $\mathbf{u}^\top \mathbf{J}_3 \mathbf{J}_2 \mathbf{J}_1$（$h = n = 1000$）に対する 2 通りの計算順序。左から掛けると中間結果はすべて $1 \times h$ の行となり、各ステップが行列・ベクトル積となって乗算は 3×10^6 回で済む。右から掛けると最初の 2 ステップがそれぞれ 10^9 回の乗算を要する行列・行列積になる。コストを決めるのは深さではなく形状である。

これら 2 つの順序は、ネットワークをたどる 2 つの向きに対応している。左からの乗算は、層あたり 1 回の行列・ベクトル積のコストで、出力の感度を層に沿って逆方向へ伝播させる。一方、右からの乗算が低コストになるのは、連鎖の右端に列ベクトル $\mathbf{v}$ がある場合であり、単一の入力方向を順方向へ伝播させて $\mathbf{J}\mathbf{v}$ を求める。これら 2 つの順序を自動化するアルゴリズムが、それぞれリバースモードおよびフォワードモードの自動微分と呼ばれる（34.5 節）。

上の演算回数では、すべての層の幅が等しいと仮定した。各因子の形状が異なるとき、左端または右端から一方向に掛ける順序が最適とは限らない。連鎖状の積であれば、最小コストの括弧の付け方は動的計画法（dynamic programming）で求められるが（演習問題 2）、一般の計算グラフではこの問題は NP 完全である (Naumann 2008)。

34.4.3 ベクトルと行列に関する導関数

層を訓練するには、入力、重み、バイアスといったベクトルと行列の双方に関してスカラー損失の勾配を求める必要がある。本項では、必要となるすべての勾配を単一の手法で導出する。この手法は、それぞれの結果を自動的に変数の形状で返す。

34.4.3.1 全微分法

変数の微小な摂動を $d\mathbf{x}$ と書き、df をその結果生じる f の変化のうち $d\mathbf{x}$ に関して線形な部分を表すものとする。これは単に、項 $\mathbf{J}\delta$（Equation 34.31 において δ を $d\mathbf{x}$ に書き換えたもの）にほかならない。Equation 34.31 と Equation 34.34 より、この線形部分は勾配との内積となる。

$$df = \langle \nabla f,\, d\mathbf{x} \rangle, \tag{34.37}$$

これはベクトル変数（$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^{\top}\mathbf{b}$）と行列変数のどちらに対しても成り立つ。Equation 34.37 は 2 通りに読むことができる。勾配が既知なら、微小変化に対して f がどう応答するかを教えてくれる。一方、f が与えられたなら、勾配を直接求める手順になる。すなわち、通常の微分規則を使って df を計算し、$\langle \mathbf{M},\, d\mathbf{x} \rangle$ の形に整理して、$\nabla f = \mathbf{M}$ を読み取るのである。この読み取りに曖昧さはない。なぜなら、$d\mathbf{x}$ の線形関数を $\langle \mathbf{M}, d\mathbf{x} \rangle$ と表す行列 $\mathbf{M}$ はただ 1 つしかないからである。実際、$d\mathbf{x}$ を、ただ 1 つの 1 を (i, j) 成分に持ち他がすべて 0 である行列にとれば、M_{ij} が取り出せる。

df の計算には、おなじみの 4 つの規則があれば十分である。第 1 に、微分は線形である。第 2 に、積の微分法則 $d(\mathbf{AB}) = d\mathbf{A}\,\mathbf{B} + \mathbf{A}\,d\mathbf{B}$ に従う。なぜなら、$(\mathbf{A} + d\mathbf{A})(\mathbf{B} + d\mathbf{B})$ を展開した式と $\mathbf{AB} + d\mathbf{A}\,\mathbf{B} + \mathbf{A}\,d\mathbf{B}$ との差は、二次の項 $d\mathbf{A}\,d\mathbf{B}$ だけだからである。第 3 に、転置と可換であり、$d(\mathbf{A}^{\top}) = (d\mathbf{A})^{\top}$ が成り立つ。第 4 に、成分ごとに適用されるスカラー関数は各成分でスカラーの連鎖律に従い、$d\varphi(\mathbf{X}) = \varphi'(\mathbf{X}) \odot d\mathbf{X}$ となる。ここで $\odot$ は要素ごとの乗算を表す。

df を $\langle \mathbf{M}, d\mathbf{x} \rangle$ の形に整理するには、もう 1 つだけ道具があればよい。**移動則**（move rule）は、$d\mathbf{x}$ の隣に掛かっている因子を内積の反対側へと移す規則である。

$$\langle \mathbf{A},\, \mathbf{BC} \rangle = \langle \mathbf{B}^{\top}\mathbf{A},\, \mathbf{C} \rangle = \langle \mathbf{AC}^{\top},\, \mathbf{B} \rangle,$$

これはトレースの巡回性 $\operatorname{tr}(\mathbf{XY}) = \operatorname{tr}(\mathbf{YX})$ から直接導かれる。実際、$\operatorname{tr}(\mathbf{A}^{\top}\mathbf{BC}) = \operatorname{tr}\big((\mathbf{B}^{\top}\mathbf{A})^{\top}\mathbf{C}\big) = \operatorname{tr}(\mathbf{CA}^{\top}\mathbf{B}) = \operatorname{tr}\big((\mathbf{AC}^{\top})^{\top}\mathbf{B}\big)$ が成り立つ。言葉で言えば、変数の左にある因子は転置をとって相手側の左へ移り、右にある因子は転置をとって相手側の右へ移る。要素ごとの積の因子はそのまま移り、$\langle \mathbf{A}, \mathbf{B} \odot \mathbf{C} \rangle = \langle \mathbf{A} \odot \mathbf{B}, \mathbf{C} \rangle$ となる。両辺とも $\sum_{ij} A_{ij} B_{ij} C_{ij}$ に等しいからである。

最も単純な 2 つの例は、それぞれ 1 行で済む。線形形式では $d(\mathbf{a}^{\top}\mathbf{x}) = \mathbf{a}^{\top} d\mathbf{x} = \langle \mathbf{a}, d\mathbf{x} \rangle$ より $\nabla_{\mathbf{x}}\, \mathbf{a}^{\top}\mathbf{x} = \mathbf{a}$ である。二次形式では、積の微分法則が 2 つの $\mathbf{x}$ の両方に作用する。

$$\begin{aligned} d(\mathbf{x}^{\top}\mathbf{A}\mathbf{x}) &= d\mathbf{x}^{\top}\mathbf{A}\mathbf{x} + \mathbf{x}^{\top}\mathbf{A}\,d\mathbf{x} = \mathbf{x}^{\top}\mathbf{A}^{\top} d\mathbf{x} + \mathbf{x}^{\top}\mathbf{A}\,d\mathbf{x} \\ &= \big\langle (\mathbf{A} + \mathbf{A}^{\top})\mathbf{x},\, d\mathbf{x} \big\rangle. \end{aligned}$$

2 つ目の等号では 1×1 行列の転置をとった。$d\mathbf{x}^{\top}\mathbf{A}\mathbf{x}$ はスカラーなので、自身の転置 $\mathbf{x}^{\top}\mathbf{A}^{\top} d\mathbf{x}$ に等しい。係数を読み取れば $\nabla_{\mathbf{x}}\, \mathbf{x}^{\top}\mathbf{A}\mathbf{x} = (\mathbf{A} + \mathbf{A}^{\top})\mathbf{x}$ が得られ、これは $2\mathbf{A}\mathbf{x}$ に簡約される（$\mathbf{A}$ が対称な場合）。

添字を書き下すことでも同じ結果に至る。成分ごとに計算すると、同じ導関数は $\partial(\sum_{ij} x_i A_{ij} x_j)/\partial x_k = \sum_j A_{kj} x_j + \sum_i x_i A_{ik}$ となり、第 k 成分として $(\mathbf{A} + \mathbf{A}^{\top})\mathbf{x}$ の対

応する成分が復元される。添字による導出は長くなるが、転置に関して疑問の余地が残らない。4.3 節では添字を使って行列積の規則を導出している。

任意の行列恒等式に対する手早い妥当性確認は、すべての行列を 1×1 に置き換えることである。その場合、式はおなじみのスカラーの公式に帰着しなければならない。先ほどの二次形式の勾配は $\frac{d}{dx}(ax^2) = 2ax = (a+a)x$ となるため、この確認を通過する。この 1×1 **チェック**は係数の誤りや項の抜けを確実に検出するが、転置や因子の順序には無力である。1×1 行列は可換であり、自身の転置に等しいからである。形状規則はたいていの転置ミスを検出してくれるが、残念ながら因子が正方行列のときには機能しない。これこそ本節の冒頭で警告した落とし穴である。正方の因子に対しては、数値チェックでしか誤りを見つけられない。$\mathbf{A} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$、$\mathbf{x} = (1, 2)$ としよう。このとき $\mathbf{x}^{\top}\mathbf{A}\mathbf{x} = x_1 x_2$ であり、真の勾配 $(x_2, x_1) = (2, 1)$ は $(\mathbf{A} + \mathbf{A}^{\top})\mathbf{x}$ に一致する。一方、素朴な推測 $2\mathbf{A}\mathbf{x} = (4, 0)$（スカラーの公式 $2ax$ から連想される式）は、1×1 チェックを通過し、形状も正しいにもかかわらず、やはり誤りである。

全微分法は、行列変数も追加の手間なく扱える。最小二乗法の損失 $\ell = \|\mathbf{W}\mathbf{x} - \mathbf{y}\|^2$ において残差を $\mathbf{r} = \mathbf{W}\mathbf{x} - \mathbf{y}$ とおくと、$\mathbf{W}$ を変化させたとき $d\mathbf{r} = d\mathbf{W}\,\mathbf{x}$ となり、移動則によって $\mathbf{x}$ が反対側へ移る。すなわち $d\ell = 2\mathbf{r}^{\top} d\mathbf{r} = \langle 2\mathbf{r},\, d\mathbf{W}\,\mathbf{x}\rangle = \langle 2\mathbf{r}\mathbf{x}^{\top},\, d\mathbf{W}\rangle$ であり、したがって次式を得る。

$$\nabla_{\mathbf{W}} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|^2 = 2\,(\mathbf{W}\mathbf{x} - \mathbf{y})\,\mathbf{x}^{\top}, \tag{34.38}$$

これは残差と入力のランク 1 の外積であり、自動的に $\mathbf{W}$ の形状になっている。代わりに $\mathbf{x}$ を変化させると、$d\mathbf{r} = \mathbf{W}\,d\mathbf{x}$ より $\nabla_{\mathbf{x}}\ell = 2\mathbf{W}^{\top}\mathbf{r}$ となる。どちらの場合も、勾配は残差の 2 倍に、変数に掛かっている因子の転置を掛けたものになっている。

34.4.3.2 層の逆伝播則

ネットワークの内部では、層 $\mathbf{Y} = \mathbf{F}(\mathbf{X})$ はスカラー損失 ℓ に至る長い計算の 1 ステップである。その完全なヤコビ行列を作るのは現実的ではない。1000×1000 の重み行列を 1000 個の例からなるミニバッチに適用すると、10^6 個の出力が 10^6 個の重みから生成されるため、重みに関する出力のヤコビ行列は 10^{12} 個もの成分を持つことになる。幸い、誤差逆伝播法が必要とする量ははるかに少ない。層がすべきことは、自身の出力に関する損失の勾配を、入力と重みに関する勾配へと変換することだけである。34.3.2 節に従い、$\bar{\mathbf{Y}} = \nabla_{\mathbf{Y}}\ell$ と書いて $\mathbf{Y}$ の随伴変数（adjoint）を表す。これは $\mathbf{Y}$ と同じ形状の行列であり、$\mathbf{Y}$ の各成分に関する損失の全微分を保持する。

命題（層の逆伝播則） **$\mathbf{Y} = \mathbf{F}(\mathbf{X})$ が $\mathbf{X}$ で微分可能であり、ℓ が $\mathbf{Y}$ の微分可能なスカラー関数であるとする。このとき、$\bar{\mathbf{X}}$（合成関数 $\ell(\mathbf{F}(\mathbf{X}))$ の随伴変数）は、次式を満たす一意な行列である。**

$$\langle \bar{\mathbf{X}},\, d\mathbf{X}\rangle = \langle \bar{\mathbf{Y}},\, d\mathbf{Y}\rangle \quad \textit{for every } d\mathbf{X},$$

ここで、$d\mathbf{Y}$ は $\mathbf{F}$ において $d\mathbf{X}$ が引き起こす一次の変化である。

証明連鎖律 Equation 34.35 により（行列を平坦化したベクトルとみなせば）、合成関数の線形部分は、ℓ の線形部分を $\mathbf{F}$ の線形部分に適用したものとなり、$d\ell = \langle \bar{\mathbf{Y}}, d\mathbf{Y}\rangle$ が成り立つ。$d\mathbf{Y}$ は $d\mathbf{X}$ に関して線形であるため、$d\ell$ もそうであり、$\langle \mathbf{M}, d\mathbf{X}\rangle$ の形で一意な行列 $\mathbf{M}$ を用いて表せる。Equation 34.37 より、$\mathbf{M} = \bar{\mathbf{X}}$ である。■

実際には、この命題の適用は移動則の適用に帰着する。$d\mathbf{y} = \mathbf{J}\,d\mathbf{x}$ を満たすベクトル層に対しては、$\langle \bar{\mathbf{y}}, \mathbf{J}\,d\mathbf{x}\rangle = \langle \mathbf{J}^{\top}\bar{\mathbf{y}}, d\mathbf{x}\rangle$ となるため、$\bar{\mathbf{x}} = \mathbf{J}^{\top}\bar{\mathbf{y}}$ を得る。すなわち、層の逆伝播則はその線形化の転置にほかならない。線形変換の転置は随伴（adjoint）とも呼ばれ、随伴変数という名称はこれに由来する。$\bar{\mathbf{x}}^{\top} = \bar{\mathbf{y}}^{\top}\mathbf{J}$ と書き直せば、これは Equation 34.36 を左から右へ掛ける順序の 1 ステップそのものである。行ベクトルとヤコビ行列の積は**ベクトル・ヤコビ行列積**（vector–Jacobian

product）と呼ばれる。この命題により、行列変数に対してもヤコビ行列を一度も作ることなくこれを計算できる。

深層学習で最も頻繁に現れる導関数は、行列積の導関数である。ミニバッチを処理するか単一のベクトルを処理するかによって重みはデータの右側または左側に置かれるため、両方の配置をまとめて導出しよう。1 行に 1 つの例を並べたミニバッチ $\mathbf{X} \in \mathbb{R}^{n\times d}$（本文全体の規約）に重み行列 $\mathbf{W} \in \mathbb{R}^{d\times q}$ を掛けると、$\mathbf{Y} = \mathbf{X}\mathbf{W}$ が得られる。積の微分法則より $d\mathbf{Y} = d\mathbf{X}\,\mathbf{W} + \mathbf{X}\,d\mathbf{W}$ となり、移動則によって内積は変数ごとの項に分解される。すなわち $\langle \bar{\mathbf{Y}}, d\mathbf{X}\,\mathbf{W}\rangle + \langle \bar{\mathbf{Y}}, \mathbf{X}\,d\mathbf{W}\rangle = \langle \bar{\mathbf{Y}}\mathbf{W}^\top, d\mathbf{X}\rangle + \langle \mathbf{X}^\top\bar{\mathbf{Y}}, d\mathbf{W}\rangle$ であり、両方の係数を読み取ると次式を得る。

$$\bar{\mathbf{X}} = \bar{\mathbf{Y}}\,\mathbf{W}^\top, \qquad \bar{\mathbf{W}} = \mathbf{X}^\top\bar{\mathbf{Y}}. \tag{34.39}$$

単一の例を列ベクトルとして $\mathbf{y} = \mathbf{W}\mathbf{x}$ と書いた場合も、まったく同じ計算から次式が得られる。

$$\bar{\mathbf{W}} = \bar{\mathbf{y}}\,\mathbf{x}^\top, \qquad \bar{\mathbf{x}} = \mathbf{W}^\top\bar{\mathbf{y}},$$

これは Equation 34.38 の形と一致しており、重み勾配は出力の随伴変数と入力の外積になる。2 つの配置は単に互いの転置である。$\mathbf{x}_i^\top$ と $\bar{\mathbf{y}}_i^\top$ をそれぞれ $\mathbf{X}$ と $\bar{\mathbf{Y}}$ の行とすれば、ミニバッチの公式 $\mathbf{X}^\top\bar{\mathbf{Y}} = \sum_i \mathbf{x}_i\bar{\mathbf{y}}_i^\top$ は例ごとにこの外積を足し合わせたものになっている。

したがって、線形層の逆伝播は順伝播の積と同じサイズの 2 つの行列積からなり、順伝播は ndq 回の乗算、逆伝播は $2ndq$ 回の乗算を要する。形状を確認すれば両方の転置も確かめられる。$\bar{\mathbf{Y}}\mathbf{W}^\top$ は $n\times d$ で $\mathbf{X}$ と一致し、$\mathbf{X}^\top\bar{\mathbf{Y}}$ は $d\times q$ で $\mathbf{W}$ と一致する。より複雑な行列演算（逆行列、行列式、各種の行列分解）の逆伝播則も、まったく同じ手法から導かれる (Mike B. Giles 2008)。

あと 2 種類の層が揃えば、多層パーセプトロンが完成する。要素ごとの活性化値 $\mathbf{Y} = \varphi(\mathbf{X})$ は $d\mathbf{Y} = \varphi'(\mathbf{X}) \odot d\mathbf{X}$ を満たすため、$\varphi'(\mathbf{X})$ を内積の反対側へ移すと $\bar{\mathbf{X}} = \varphi'(\mathbf{X}) \odot \bar{\mathbf{Y}}$ が得られる。これは先ほどの対角ヤコビ行列をそのままマスクとして適用したものである。バイアスの加算 $\mathbf{y} = \mathbf{z} + \mathbf{b}$ では $d\mathbf{y} = d\mathbf{b}$ より $\bar{\mathbf{b}} = \bar{\mathbf{y}}$ となる。ミニバッチに対しては、バイアスは $\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{1}\mathbf{b}^\top$ によって各行にブロードキャストされるが、これも単なる行列積であり、すべての例にわたって随伴変数の和をとる $\bar{\mathbf{b}} = \bar{\mathbf{Z}}^\top\mathbf{1}$ を与える。これら 3 つの局所規則が基本要素となり、4.3 節では多層パーセプトロン全体の逆伝播が組み立てられる。

これらの規則を使えば、冒頭の重み勾配行列も導出できる。Figure 34.21 のネットワークを、$\bar{g} = 1$ から 1 層ずつ逆向きに $\mathbf{x} = (1, 1)$ においてたどると次のようになる。

1. 出力 $g = \mathbf{w}_2^\top\mathbf{h}$ から、$\bar{\mathbf{h}} = \mathbf{w}_2 = (2, -1)$ および $\bar{\mathbf{w}}_2 = \mathbf{h} = (3, 0)$ が得られる。
2. ReLU から $\bar{\mathbf{a}} = \mathbf{m} \odot \bar{\mathbf{h}} = (2, 0)$ が得られ、バイアスから $\bar{\mathbf{b}} = \bar{\mathbf{a}} = (2, 0)$ が得られる。
3. アフィン写像 $\mathbf{a} = \mathbf{W}_1\mathbf{x} + \mathbf{b}$ から、$\bar{\mathbf{W}}_1 = \bar{\mathbf{a}}\,\mathbf{x}^\top = \left(\begin{smallmatrix} 2 & 2 \\ 0 & 0 \end{smallmatrix}\right)$ および $\bar{\mathbf{x}} = \mathbf{W}_1^\top\bar{\mathbf{a}} = (2, 4)$ が得られる。

$\bar{\mathbf{x}}$ は先ほど求めたヤコビ行列の行 $(2, 4)$ の転置になっており、$\bar{\mathbf{W}}_1$ は冒頭で重みを微小変化させて得た行列そのものである。コード上で g を $\mathbf{W}_1$、$\mathbf{b}$、$\mathbf{w}_2$ に関して微分すると、3 つの配列すべてが一致することが確認できる。

```
gW1, gb, gw2 = jax.grad(g, argnums=(1, 2, 3))(x, W1, b, w2)
h = jax.nn.relu(W1 @ x + b)
show = lambda t: (np.asarray(t) + 0.0).tolist()  # + 0.0 turns -0.0 into 0.0
print('dg/dW1 = (w2*m) x^T:', show(jnp.outer(w2 * m, x)),
      '  autograd:', show(gW1))
```

```
print('dg/db  = w2*m      :', show(w2 * m), '  autograd:', show(gb))
print('dg/dw2 = h         :', show(h), '  autograd:', show(gw2))
```

```
dg/dW1 = (w2*m) x^T: [[2.0, 2.0], [0.0, 0.0]]   autograd: [[2.0, 2.0], [0.0, 0.0]]
dg/db  = w2*m      : [2.0, 0.0]   autograd: [2.0, 0.0]
dg/dw2 = h         : [3.0, 0.0]   autograd: [3.0, 0.0]
```

どこにゼロが現れるかにも注目しておきたい。非活性な隠れユニットは 3 か所にゼロを残す。$\bar{\mathbf{W}}_1$ のゼロ行、$\bar{\mathbf{b}}$ のゼロ成分、そして $\bar{\mathbf{w}}_2$ のゼロ成分である（その出力 $h_2 = 0$ が $w_{2,2}$ に掛かるためである）。この入力に対する訓練ステップでは、そのユニットに属する 4 つのパラメータはすべて更新されないまま残る。

正方の因子に対する転置の誤りを検出する数値テストがまだ残っていた。失敗しえないテストは何も証明しないため、ここでは誤りが露呈するようにテストを設計する。以下の各因子はすべてランダムな正方行列（したがって非対称行列）とし、正しい規則の隣に同じ形状のもっともらしい誤答候補を並べて出力する。二次形式と行列積の両方をテストしよう。行列積については、スカラー $\langle \bar{\mathbf{Y}}, \mathbf{X}\mathbf{W} \rangle$ をランダムな $\bar{\mathbf{Y}}$ に対して微分する。逆伝播則の命題により、$\mathbf{X}$ と $\mathbf{W}$ に関するその勾配はまさに $\bar{\mathbf{X}}$ と $\bar{\mathbf{W}}$ になる。

```
A, x = jnp.array(np.random.randn(3, 3)), jnp.array(np.random.randn(3))
X, W, Ybar = (jnp.array(np.random.randn(3, 3)) for _ in range(3))  # square
ok = lambda u, v: bool(jnp.allclose(u, v, atol=1e-5))
gx = jax.grad(lambda x: x @ A @ x)(x)
gX, gW = jax.grad(lambda X, W: (Ybar * (X @ W)).sum(), argnums=(0, 1))(X, W)
print('grad x^T A x: (A + A^T) x', ok(gx, (A + A.T) @ x),
      '| wrong guess 2 A x       ', ok(gx, 2 * A @ x))
print('Xbar        : Ybar W^T   ', ok(gX, Ybar @ W.T),
      '| wrong guess Ybar W      ', ok(gX, Ybar @ W))
print('Wbar        : X^T Ybar   ', ok(gW, X.T @ Ybar),
      '| wrong guess (X^T Ybar)^T', ok(gW, (X.T @ Ybar).T))
```

```
grad x^T A x: (A + A^T) x True | wrong guess 2 A x        False
Xbar        : Ybar W^T    True | wrong guess Ybar W       False
Wbar        : X^T Ybar    True | wrong guess (X^T Ybar)^T False
```

導出した規則は `True` を出力し、誤答候補は `False` を出力する。ここではテストの結果以上に、テストの設計が重要である。長方行列の因子、たとえば $\mathbf{X}$ を形状 3×4、$\mathbf{W}$ を形状 4×2 とした場合、積に対する 2 つの誤答候補は実行すらできない。転置の位置を誤ると数値を比較する前に形状エラーが発生するが、これは形状規則が役割を果たした結果である。同様に、対称な $\mathbf{A}$ を使うと、誤答候補 $2\mathbf{A}\mathbf{x}$ は真の勾配と一致してテストをすり抜けてしまう。厄介なのは正方かつ非対称な因子であり、数値チェックではまさにそれを生成しなければならない。

行列積の規則を使えば、積の両方の因子が未知である損失、たとえば行列因子分解（matrix factorization）の損失 $\|\mathbf{X} - \mathbf{U}\mathbf{V}\|_F^2$（演習問題 3）も同じように容易に扱える。いずれの場合も、逆伝播則は順伝播でその変数に掛かっていた因子の転置を受け取った随伴変数に掛ける。また、各規則が順伝播のどの値を必要とするかにも注目したい。行列積では $\mathbf{X}$ と $\mathbf{W}$、活性化では $\varphi'(\mathbf{X})$ が必要であり、バイアスでは何も必要としない。逆伝播ではこれらの中間値を順伝播から保持しておく必要があり、そのメモリコストは 34.5 節で直接計測する。

34.4.3.3 ソフトマックスと交差エントロピー

分類器が最も頻繁に評価する導関数は、ソフトマックスと交差エントロピー損失を合成した関数の導関数である。ソフトマックスはスコアのベクトル $\mathbf{z}$、すなわち**ロジット**を確率ベクトルへ変換する。$S = \sum_k e^{z_k}$ とおけば $p_i = e^{z_i}/S$ である。2 つのロジットに対して 34.2 節で log-sum-exp のヘッセ行列を導いたのと同じ商の微分法則から、任意のクラス数に対するソフトマックスのヤコビ行列が得られる。$\delta_{ij} = 1$（$i = j$ のとき）、それ以外では 0 と書けば、次式が得られる。

$$\frac{\partial p_i}{\partial z_j} = \frac{\delta_{ij} e^{z_i} S - e^{z_i} e^{z_j}}{S^2} = p_i(\delta_{ij} - p_j), \qquad \mathbf{J}_{\text{softmax}} = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^{\top}. \tag{34.40}$$

言葉で言えば、ロジット i を増やすと p_i は変化率 $p_i(1 - p_i)$ で増加し、他のすべての確率 p_j は変化率 $p_i p_j$ で減少する。簡単な検算として、確率の和は 1 であるため $\mathbf{J}\mathbf{1} = \mathbf{p} - \mathbf{p}(\mathbf{p}^{\top}\mathbf{1}) = \mathbf{0}$ となる。すべてのロジットに同じ定数を加えても $\mathbf{p}$ は変化しないため、これは当然成り立たなければならない。結果として、ソフトマックスのヤコビ行列は常に特異である。

また、$\mathbf{J}_{\text{softmax}}$ が対称であることにも注目したい。一般のヤコビ行列は対称とは限らないが、これが対称になるのは実際にはヘッセ行列だからである。すなわち、ソフトマックスは本章で繰り返し用いている例である log-sum-exp 関数 $\operatorname{LSE}(\mathbf{z}) = \log \sum_k e^{z_k}$ の勾配であり（$\partial \operatorname{LSE}/\partial z_i = e^{z_i}/S = p_i$）、勾配のヤコビ行列はその関数のヘッセ行列に一致する。2 ロジットの公式に Equation 34.40 が付け加えるのは、この行列の明快な統計的解釈である。これは確率 $\mathbf{p}$ に従って抽出されるワンホットベクトルの共分散行列になっており、したがって任意の $\mathbf{z}$ と任意のベクトル $\mathbf{v}$ に対して、二次形式 $\mathbf{v}^{\top}\mathbf{J}\mathbf{v} = \sum_i p_i v_i^2 - \left(\sum_i p_i v_i\right)^2$ は $\mathbf{v}$ の各成分の、分布 $\mathbf{p}$ の下での分散になる。分散は決して負にならないため、このヘッセ行列はいたるところで半正定値（positive semidefinite）となり、log-sum-exp が凸性を持つことが証明される。定数ベクトル $\mathbf{v}$ の分散は 0 であり、ここからも零ベクトルを与える方向 $\mathbf{1}$ が再び得られる。

ここで、ソフトマックスと交差エントロピー損失 $\ell = -\sum_i y_i \log p_i$（ターゲット分布 $\mathbf{y}$ に対する損失）を合成しよう。連鎖律により行ベクトル $\partial \ell / \partial \mathbf{p} = (-y_i/p_i)_i$ に Equation 34.40 を掛けると、式は劇的に簡約される。

$$\frac{\partial \ell}{\partial z_j} = -\sum_i \frac{y_i}{p_i} p_i(\delta_{ij} - p_j) = -y_j + p_j \sum_i y_i = p_j - y_j, \qquad \text{that is,} \qquad \nabla_{\mathbf{z}} \ell = \mathbf{p} - \mathbf{y}. \tag{34.41}$$

ロジットに関する勾配は、単に予測確率からターゲットを引いたものになり、これは $\hat{\mathbf{y}} - \mathbf{y}$ として 3.1.2.2 節で書いた公式そのものである。最後の変形では $\sum_i y_i = 1$ しか使っていないため、この結果はワンホットラベルだけでなく任意の確率ベクトル $\mathbf{y}$ に対して成り立つ。したがって、ラベル平滑化（label smoothing）や蒸留のターゲットもまったく同じ公式に従う。和が 1 にならないターゲットに対しては、勾配は代わりに $\left(\sum_i y_i\right)\mathbf{p} - \mathbf{y}$ となる。たとえば $\mathbf{z} = (1, -0.5, 2)$ では $\mathbf{p} = (0.254,\ 0.057,\ 0.690)$ であり、第 2 クラスをターゲットとするとき、勾配は $(0.254,\ -0.943,\ 0.690)$ となる。

実際のライブラリでは、数値安定性のためにソフトマックスと交差エントロピーを単一のプリミティブに融合している（3.3.5.2 節、35.8.2 節）。以下では、自動微分によって $\mathbf{z}$ におけるソフトマックスのヤコビ行列を計算し、その固有値を出力する。分散による議論から予測されるとおり、固有値は非負で 1 つが 0 になるはずである。続いて、融合された損失を、先ほどのワンホットタ

ーゲットと、成分の和が 1 にならないツーホットターゲット $\mathbf{y} = (1, 1, 0)$ の両方に対して微分する。

```
z = jnp.array([1.0, -0.5, 2.0])
p = jax.nn.softmax(z)
J = jax.jacobian(jax.nn.softmax)(z)              # the Jacobian, by autograd
print('eigenvalues of J:', np.linalg.eigvalsh(np.asarray(J)).round(3) + 0.0)
loss = lambda z, y: -(y * jax.nn.log_softmax(z)).sum()
for y in [jnp.array([0.0, 1.0, 0.0]), jnp.array([1.0, 1.0, 0.0])]:
    print(f'y = {np.asarray(y)}: d loss / d z = {np.asarray(jax.grad(loss)(z, y)).round(4)},
    ↪  '
          f'p - y = {np.asarray(p - y).round(4)}')
```

```
eigenvalues of J: [0.    0.079 0.378]
y = [0. 1. 0.]: d loss / d z = [ 0.2537 -0.9434  0.6897], p - y = [ 0.2537 -0.9434  0.6897]
y = [1. 1. 0.]: d loss / d z = [-0.4926 -0.8868  1.3793], p - y = [-0.7463 -0.9434  0.6897]
```

固有値は 0、0.079、0.378 となり、方向 $\mathbf{1}$ に対応する 1 つのゼロと 2 つの正の値が得られた。ワンホットターゲットに対しては、融合された損失の勾配は $\mathbf{p} - \mathbf{y} = (0.2537, -0.9434, 0.6897)$ となり、手計算と一致する。しかしツーホットターゲットに対しては $(-0.4926, -0.8868, 1.3793)$ を返し、これは $2\mathbf{p} - \mathbf{y}$ であって、隣に出力された $\mathbf{p} - \mathbf{y}$ ではない。「予測からターゲットを引いた形」になるのは和が 1 になるターゲットの性質であり、損失単独の性質ではない。

本節では、単純な 3 層の連鎖に対して $\mathbf{J}_K \cdots \mathbf{J}_1$ を手計算で掛け合わせた。しかし実際には、プログラムによって分岐や合流を含む任意の計算グラフを扱い、各逆伝播則が必要とする順伝播の値を保持し、パラメータがどれほど多くてもネットワークの評価の小さな定数倍にとどまるコストですべての勾配を計算しなければならない。そのプログラムこそが自動微分であり、34.5 節の主題である。

34.4.4 まとめ

写像 $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ のヤコビ行列とは、その最良線形近似を表す $m \times n$ 行列であり、各成分は $[\mathbf{J}_\mathbf{f}]_{ij} = \partial f_i / \partial x_j$ で与えられる。勾配（1 行のベクトルを列として書いたもの）もヘッセ行列（勾配のヤコビ行列）も、その特別な場合である。入力に関するネットワークの導関数は各層のヤコビ行列の積 $\mathbf{J}_K \cdots \mathbf{J}_1$ となり、乗算の順序がコストを左右する。幅 1000 の 3 層では、300 万回対 20 億回の乗算という差が生じる。全微分則 $df = \langle \nabla f, d\mathbf{x} \rangle$ を用いれば、行列積の逆伝播則 $\bar{\mathbf{X}} = \bar{\mathbf{Y}}\mathbf{W}^\top$、$\bar{\mathbf{W}} = \mathbf{X}^\top \bar{\mathbf{Y}}$ や、ソフトマックスと交差エントロピーの合成が $\mathbf{p} - \mathbf{y}$ に簡約される結果を含め、層が必要とするあらゆる勾配が自動的に変数の形状で導出される。手計算でのうっかりミスの大半は形状規則と 1×1 チェックで検出できるが、正方の因子における転置の誤りは両方をすり抜けるため、非対称行列を用いた数値テストが必要になる。

34.4.5 演習問題

1. **ReLU ネットワークのヤコビ行列。** $\mathbf{g}(\mathbf{x}) = \mathbf{W}_2\, \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b})$ とし、$\mathbf{W}_1 \in \mathbb{R}^{h \times n}$、$\mathbf{b} \in \mathbb{R}^h$、$\mathbf{W}_2 \in \mathbb{R}^{m \times h}$、さらに要素ごとの ReLU を φ とする。活性化前の値 $\mathbf{W}_1 \mathbf{x} + \mathbf{b}$ のどの成分も 0 でないと仮定する。Equation 34.35 より $\mathbf{J}_\mathbf{g}(\mathbf{x}) = \mathbf{W}_2 \mathbf{M}(\mathbf{x}) \mathbf{W}_1$ であり、ここで対角行列 $\mathbf{M}(\mathbf{x})$ は活性な各隠れユニットに対して成分 1 を、非活性な各隠れユニットに対して成分 0 を持つ。
    1. $\mathbf{g}(\mathbf{x} + \delta) = \mathbf{g}(\mathbf{x}) + \mathbf{J}_\mathbf{g}(\mathbf{x})\, \delta$ が十分小さな任意の δ に対して厳密に成り立ち、したがって Equation 34.31 の剰余項が恒等的に消えることを示せ。本節の 2-2-1 ネットワークの $\mathbf{x} = (1, 1)$ において、最大の r として、すべての δ $(\|\delta\|_\infty < r)$ に対してこの等式

が成り立つ値を求めよ。$n = m = 2$ のとき、小さな円の $\mathbf{g}$ による像は Figure 34.20 の像とどう異なるか。

2. k 個の隠れユニットが $\mathbf{x}$ において活性であるとする。$\operatorname{rank} \mathbf{J}_{\mathbf{g}}(\mathbf{x}) \le k$ を示せ。$n = m = 2$ かつ $k = 1$ のとき、$\mathbf{x}$ の周りの小さな円の像はどうなるか。
3. いずれかの活性化前の値が 0 になる入力の集合を記述せよ。$\mathbf{J}_{\mathbf{g}}$ は、$\mathbf{x}$ がこの集合を横切るときどう変化するか。

2. [code] **最小コストの連鎖順序**。●Equation 34.36 より、K 個の写像の連鎖のヤコビ行列は $\mathbf{J}_K \cdots \mathbf{J}_1$（ただし $\mathbf{J}_k \in \mathbb{R}^{d_k \times d_{k-1}}$）である。$p \times q$ 行列と $q \times r$ 行列の乗算には pqr 回の乗算を要する。
   1. 右から掛ける順方向順序のコストと、左から掛ける逆方向順序のコストを、$d_0, \dots, d_K$ の関数として表せ。広い層と狭い層が交互に並ぶ写像 $(d_0, \dots, d_4) = (1000, 10, 1000, 10, 1000)$ に対して両者を評価し、本節で求めた $(1 \times h)(h \times h)(h \times h)(h \times n)$ の演算回数と比較せよ。
   2. 部分連鎖 $\mathbf{J}_j \cdots \mathbf{J}_i$ 上の動的計画法によって最小コストの括弧の付け方を求めるプログラムを書き、(a) の次元に対して実行せよ。どの順序が選ばれ、(a) の 2 つの順序のうち優れた方と比べてどれだけ低コストになるか。
   3. スカラー出力 $d_K = 1$ の場合について、数千個のランダムな次元ベクトル（K は 2 から 6 の範囲とし、$d_0, \dots, d_{K-1}$ の各成分を $\{1, \dots, 10\}$ から一様に抽出する）上で最適順序と逆方向順序を比較せよ。逆方向以外の順序の方が低コストになる次元ベクトルを特徴づけよ。

3. [code] **行列積の逆伝播則**。
   1. $\mathbf{Y} = \mathbf{X}\mathbf{W}$ に対して Equation 34.39 を全微分法で導出し、さらに添字を用いて再度導出して転置を確認せよ。
   2. ベクトル・ヤコビ行列積 `jax.vjp` を用い、$\mathbf{X} \in \mathbb{R}^{3 \times 4}$、$\mathbf{W} \in \mathbb{R}^{4 \times 2}$ およびランダムな随伴変数 $\bar{\mathbf{Y}}$ に対して両方の規則を検証せよ。順伝播の積と逆伝播の 2 つの積それぞれの乗算回数を数えよ。
   3. 行列因子分解の損失 $\|\mathbf{X} - \mathbf{U}\mathbf{V}\|_F^2$（$\mathbf{R} = \mathbf{X} - \mathbf{U}\mathbf{V}$）について、$\nabla_{\mathbf{U}} = -2\mathbf{R}\mathbf{V}^\top$ および $\nabla_{\mathbf{V}} = -2\mathbf{U}^\top\mathbf{R}$ を数値的に確認せよ。次に、一方の因子を固定して他方の勾配を 0 とおく交互最小二乗法の更新ステップを書き下し、各ステップが 1 回の線形方程式の求解になることを示せ。

4. **ソフトマックスのベクトル・ヤコビ行列積**。逆伝播ではソフトマックスのヤコビ行列 Equation 34.40 を陽に作ることは決してなく、確率の随伴変数 $\mathbf{u} = \bar{\mathbf{p}}$ をロジットの随伴変数 $\bar{\mathbf{z}} = \mathbf{J}_{\text{softmax}}^\top \mathbf{u}$ へ直接写す。
   1. $\mathbf{J}_{\text{softmax}}^\top \mathbf{u} = \mathbf{p} \odot \big(\mathbf{u} - (\mathbf{p}^\top \mathbf{u})\,\mathbf{1}\big)$ を示せ（$\odot$ は要素ごとの積）。n クラスの場合について、この式の演算回数と、$\mathbf{J}_{\text{softmax}}$ を作って $\mathbf{u}$ を掛ける場合の演算回数をそれぞれ数えよ。
   2. 自己アテンションは $n \times n$ のスコア行列 $\mathbf{S}$ の各行にソフトマックスを適用して重み $\mathbf{P}$ を生成する。$\mathbf{P}$ の（$\mathbf{S}$ に関する）ヤコビ行列は何個の成分を持ち、そのうち非ゼロになりうるのは何個か。float32 で $n = 4096$ トークンの場合について、完全なヤコビ行列、その非ゼロ成分、および (a) の式を行ごとに適用して返す $\mathbf{S}$ の随伴変数が要するメモリを比較せよ。

5. [code] **LayerNorm のヤコビ行列**。層正規化 (Ba et al. 2016) は、$\mathbf{x} \in \mathbb{R}^n$ を $\hat{\mathbf{x}} = (\mathbf{x} - \mu\mathbf{1})/\sigma$ へ写す。ここで $\mu = \frac{1}{n}\sum_i x_i$、$\sigma^2 = \frac{1}{n}\sum_i (x_i - \mu)^2 > 0$ である（学習されるスケールも ϵ もないものとする）。
   1. 全微分法により $\mathbf{J} = \frac{1}{\sigma}\big(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top - \frac{1}{n}\hat{\mathbf{x}}\hat{\mathbf{x}}^\top\big)$ を示せ。**ヒント**：$d\mu = \frac{1}{n}\mathbf{1}^\top d\mathbf{x}$ および $d\sigma = \frac{1}{n}\hat{\mathbf{x}}^\top d\mathbf{x}$ である。
   2. $\mathbf{1}$ と $\hat{\mathbf{x}}$ が直交し、かつ $\|\hat{\mathbf{x}}\|^2 = n$ であることを示せ。これから、$\sigma\mathbf{J}$ が $\operatorname{span}\{\mathbf{1}, \hat{\mathbf{x}}\}$ の直交補空間への直交射影であり、$\operatorname{rank}\mathbf{J} = n - 2$（$n \ge 2$ のとき）となることを導け。入力のどのような 2 種類の変化を LayerNorm は感知できないか。また、それはなぜ

か。
3. 自動微分で計算したヤコビ行列（ランダムな $\mathbf{x} \in \mathbb{R}^5$ に対するもの）と照合して (a) と 2 つの零ベクトルを確認し、$\mathbf{J}$ の固有値を出力せよ。

6. **対数行列式の勾配。**●正規化フロー (Rezende and Mohamed 2015) は、確率ベクトル $\mathbf{X}$ を可逆な層に順に通すモデルである。1 つの層 $\mathbf{Y} = \mathbf{g}(\mathbf{X})$ に対して、$\mathbf{Y}$ の点 $\mathbf{g}(\mathbf{x})$ における対数密度は、密度の変数変換の公式より $\log p_{\mathbf{X}}(\mathbf{x}) - \log \left|\det \mathbf{J}_{\mathbf{g}}(\mathbf{x})\right|$ となる。線形層 $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ では補正項が $-\log \left|\det \mathbf{A}\right|$ となるため、最尤法による訓練では $\log \left|\det \mathbf{A}\right|$ を $\mathbf{A}$ に関して微分する。
   1. $\det(\mathbf{I} + t\mathbf{M})$ を t の一次まで展開し、$\det(\mathbf{A} + d\mathbf{A}) = \det \mathbf{A}\, \det(\mathbf{I} + \mathbf{A}^{-1} d\mathbf{A})$ を用いて、ヤコビの公式 $d \det \mathbf{A} = \det \mathbf{A}\, \operatorname{tr}(\mathbf{A}^{-1} d\mathbf{A})$ を可逆行列 $\mathbf{A}$ に対して導け。さらに Equation 34.37 を用いて $\nabla_{\mathbf{A}} \log \left|\det \mathbf{A}\right| = \mathbf{A}^{-\top}$ を導け。なぜ $\det \mathbf{A}$ の符号は無関係なのか。
   2. ランダムな 3×3 の可逆行列上で、`np.linalg.slogdet` の中心差分によってこの恒等式を確認せよ。
   3. $\mathbf{A}$ が特異行列に近づくとき、勾配のノルムはどう振る舞うか。これは、重み行列がほぼ特異になったフロー層に対して何を意味するか。

7. **スペクトルノルムの勾配。**●スペクトル正規化 (Miyato et al. 2018) は、ネットワークの各重み行列 $\mathbf{W}$ をその最大特異値 $\sigma_1(\mathbf{W})$ で割る手法であり、訓練では σ_1 を微分することになる。
   1. $\mathbf{S}$ を単純固有値 λ と単位固有ベクトル $\mathbf{q}$ を持つ実対称行列とし、これを $\mathbf{S} + t\mathbf{E}$（$\mathbf{E}$ は対称行列）へ摂動させる。$\lambda(t)$ と $\mathbf{q}(t)$（$(\mathbf{S} + t\mathbf{E})\mathbf{q} = \lambda \mathbf{q}$ と $\mathbf{q}^{\top}\mathbf{q} = 1$ で定まる関数）は $t = 0$ の近傍で微分可能であるとしてよい（λ が単純固有値であるため陰関数定理から従う）。両式を $t = 0$ で微分してヘルマン・ファインマンの定理 $\lambda'(0) = \mathbf{q}^{\top}\mathbf{E}\,\mathbf{q}$ を示し、$\nabla_{\mathbf{S}} \lambda$ を内積 $\operatorname{tr}(\mathbf{A}^{\top}\mathbf{B})$ の下で求めよ。
   2. (a) を $\sigma_1(\mathbf{W})^2 = \lambda_{\max}(\mathbf{W}^{\top}\mathbf{W})$ に適用し、$\nabla_{\mathbf{W}} \sigma_1 = \mathbf{u}_1 \mathbf{v}_1^{\top}$ が（$\sigma_1 > 0$ が単純であるとき）成り立つことを示せ。ここで $\mathbf{u}_1$ と $\mathbf{v}_1$ は上位の左特異ベクトルと右特異ベクトルである。なぜ $\sigma_1 > 0$ が必要なのか。$\mathbf{W} = \mathbf{w} \in \mathbb{R}^{n \times 1}$ を $\mathbf{w} = \mathbf{0}$ の近傍で考察せよ。NumPy でランダムな 5×3 行列上の中心差分によってこの公式を確認せよ。
   3. $\sigma_1(\mathbf{I} + t\,\operatorname{diag}(1, -1))$ を正負両符号の小さな t に対して計算せよ。σ_1 は 2×2 単位行列において微分可能か。また、そこでは (a) のどの仮定が破れているか。

Alan Edelman および Steven G. Johnson による Matrix Calculus for Machine Learning and Beyond, problem set 2（MIT 18.S096、IAP 2023、問題 3）を改変。

34.5 自動微分

本書の「基礎」パートを通じて、あらゆる勾配はフレームワークの自動微分エンジンを 1 回呼び出すだけで得られていた。前節では、その呼び出しが数学的に何を計算しているかを示した。すなわち、ネットワークの導関数は各層に 1 つずつ対応するヤコビ行列の積であり、それらの行列をどの順序で掛けるかによってコストが決まる。しかし、実際のプログラムは整然と並んだ行列のリストではなく、加算、乗算、関数呼び出しの系列である。フレームワークは通常のコードからどのようにそれらの因子を取り出し、低コストな順序で掛け合わせているのだろうか。

低コストな順序によってどれだけ計算を節約できるかは、現代のネットワークを考えれば一目瞭然である。10^8 個のパラメータを持ち、単一のスカラー損失 ℓ を出力するモデルを考えよう。各訓練ステップでは、ℓ の 10^8 個すべてのパラメータに関する導関数が必要になる。有限差分（finite difference）による直接的な手法では、パラメータを 1 つずつ微小に動かし、ネットワークを再実行して、ℓ の変化量をその微小変化で割る。これには 1 ステップあたり 10^8 回の実行を要する。1 回の実行に 10 ms かかるとすると、1 つの勾配を計算するだけで 10^6 s、すなわち 12 日近くかかり、しかもステップサイズの選び方によって打ち切り誤差や丸め誤差が入り込む。これに対し、

誤差逆伝播法（backpropagation）は 10^8 個すべての微分値を、浮動小数点の丸めを除いて厳密に、順伝播およそ 3 回分のコスト（約 30 ms）で返す。本節では、この 10^8 倍という差がどこから生まれるのかを解き明かす。

核心となるアイデアは驚くほど単純である。ヤコビ行列の積は、どちらの端からでも評価できる。入力側から始めると、1 回のパスで 1 つの入力方向が順方向に伝播するため、10^8 個の入力には 10^8 回のパスが必要になる。一方、損失側から始めると、1 回のパスで 1 つの出力方向が逆方向に伝播するが、スカラー損失の出力はただ 1 つしかない。

1.5 節では、深層学習フレームワークで両方のモードを呼び出す方法を見た。本節ではその内部の仕組みに踏み込む。数十行の Python コードでフォワードモードとリバースモードをゼロから構築し、それらが真の導関数を返すことを証明し、逆伝播を手計算でトレースし、各モードのコストを計測する。さらに、2 つのモードを合成してある方向に沿った二階の曲率を計算し、ソルバ自体を通して微分することなく方程式の解を微分する方法を示す。なかでも特に重要な要点は 2 つある。第 1 に、プログラムを**二重数**（dual numbers）上で評価すると自動的に微分値が得られること（本節で唯一の命題）、第 2 に、どのスイープもプログラムを 1 回評価する場合の小さな定数倍のコストしかかからず、フォワードモードは入力方向ごとに 1 回のスイープを、リバースモードは出力方向ごとに 1 回のスイープを必要とすることである。初めて読む読者は、34.5.3 節を 2 回目の通読時に回してもよい。

以下の議論では、3 つの小さな例を一貫して用いる。本章で扱っている勾配がソフトマックスになる log-sum-exp $\ell(\mathbf{x}) = \log \sum_j e^{x_j}$、前節で導入した平面写像 $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$、そして手計算で完全にトレースできるほど小さな多項式 $y = (uv + u)^2$ である。

本節全体を通して、以下のインポートを使用する。

```
import numpy as np
import jax
from jax import numpy as jnp
```

34.5.1 フォワードモードと二重数

34.5.1.1 二重数

1 回の実行からプログラムの値と導関数の両方を、ステップサイズの調整も、あらかじめ f' の数式を書き下すこともなしに求めたい。その鍵となる仕掛けは、実数のごく小さな拡張である。0 ではないが二乗すると消える新しい記号 ε を添加する。

$$\varepsilon^2 = 0, \qquad \varepsilon \neq 0. \tag{34.42}$$

このたった 1 つの規則のもとで、**二重数**と呼ばれる $a + b\varepsilon$ の形の式を通常の代数法則に従って計算する。34.1 節のあらゆる微分公式は二次の項を切り捨てることで成り立っていたが、$\varepsilon^2 = 0$ とおくことで、その切り捨てが算術そのものに直接組み込まれる。

この規則を $3 + \varepsilon$ に適用するとどうなるか見てみよう。二乗および三乗すると、次のようになる。

$$(3 + \varepsilon)^2 = 9 + 6\varepsilon + \varepsilon^2 = 9 + 6\varepsilon, \qquad (3 + \varepsilon)^3 = (9 + 6\varepsilon)(3 + \varepsilon) = 27 + 27\varepsilon.$$

ε の係数に注目しよう。6 と 27 は、x^2 と x^3 の $x = 3$ における導関数の値である。微分公式は何ひとつ使っていないにもかかわらず、乗算から直接微分値が現れた。同じことは任意の積について成り立つ。2 つの二重数を掛け合わせ、ε^2 の項を落とすと次式を得る。

$$(a + b\varepsilon)(c + d\varepsilon) = ac + (ad + bc)\,\varepsilon. \tag{34.43}$$

b と d が、それぞれ値 a と c をとる 2 つの量の導関数であるとき、$ad + bc$ はそれらの積の導関数となる。つまり、二重数の乗算は積の微分法則そのものである。

懐疑的な読者は、0 でないにもかかわらず二乗すると 0 になる数などあり得ないと反論するかもしれない。もちろん実数には存在しないが、行列なら容易に構成できる。$\varepsilon = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ とおこう。すると $\varepsilon^2 = \mathbf{0}$ となり、二重数 $a + b\varepsilon$ は 2×2 行列 $\left(\begin{smallmatrix} a & b \\ 0 & a \end{smallmatrix}\right)$ になり、通常の行列積 $\left(\begin{smallmatrix} a & b \\ 0 & a \end{smallmatrix}\right)\left(\begin{smallmatrix} c & d \\ 0 & c \end{smallmatrix}\right) = \left(\begin{smallmatrix} ac & ad+bc \\ 0 & ac \end{smallmatrix}\right)$ は Equation 34.43 を再現する。複素数が平面の回転と拡大縮小と同じくらい具体的な対象であるのと同様に、二重数も 2×2 行列と同じくらい具体的な対象なのである。

実際のプログラムは sin、exp、log などの初等関数も呼び出す。関数 φ が a の周りのテイラー級数で与えられるとき、その級数に $a + b\varepsilon$ を代入すると $(b\varepsilon)^2$ 以降のすべての項が消え、一次近似だけが残る。

$$\varphi(a + b\varepsilon) = \varphi(a) + \varphi'(a)\,b\,\varepsilon.$$

したがって、$e^{a+b\varepsilon} = e^a + e^a b\,\varepsilon$ や $\sin(a + b\varepsilon) = \sin a + (\cos a)\,b\,\varepsilon$ は代数構造そのものから必然的に導かれる。解析的ではなく、単に a で微分可能であるにすぎないプリミティブについては、上の式を二重数に対する作用の定義として採用する。いずれの場合も、ε 成分は連鎖律に従う。すなわち、引数から伝わってきた導関数の値 b に局所導関数 $\varphi'(a)$ が掛けられる。

命題（二重数は導関数を計算する） 式 f が、定数、恒等写像、和、積、およびプリミティブ φ（sin、exp、log など）から構成されているとする。f を x で評価したときに各プリミティブが受け取る引数において、そのプリミティブが微分可能であるとする（log の場合は正の引数）。このとき、同じ式を二重数 $x + \varepsilon$ 上で評価すると次式が成り立つ。

$$f(x + \varepsilon) = f(x) + f'(x)\,\varepsilon. \tag{34.44}$$

証明：式の構造に関する構造的帰納法（structural induction）で示す。定数 c は $c + 0\varepsilon$、恒等写像は $x + 1\varepsilon$ と表されるため、それぞれの ε 成分はその導関数である 0 と 1 になる。次に、2 つの部分式が $g(x + \varepsilon) = g(x) + g'(x)\varepsilon$ および $h(x + \varepsilon) = h(x) + h'(x)\varepsilon$ を満たすと仮定する。これらの和の ε 成分は $g' + h' = (g + h)'$ である。また Equation 34.43 より、積の ε 成分は $g'h + gh' = (gh)'$ となる。最後に、プリミティブ φ を g に適用すると、$a = g(x)$ および $b = g'(x)$ とした上の規則により、ε 成分は $\varphi'(g(x))\,g'(x) = (\varphi \circ g)'(x)$ となる。部分式を組み合わせるどの操作でもこの性質が保たれるため、式全体も Equation 34.44 を満たす。■

したがって、プログラムを微分するには、入力を $x + \varepsilon$ として実行し、出力の ε 成分を読み取ればよい。ループや条件分岐を含むプログラムも同様に自動で処理される。任意の固定された入力に対して、プログラムは基本演算からなる 1 本の具体的な系列を実行するが、これはまさに上の命題の形の式になっており、二重数による実行はその実行経路を微分する（フレームワークでの実例は 1.5.3.1 節を参照）。ただし、得られるのはあくまで実行された経路の導関数であり、これが $f'(x)$ と一致するのは、近傍のすべての入力が同じ分岐を通る場合に限られる。たとえば、値 `x` を `x != 0` のときに返し、それ以外では `0.0` を返すプログラムは恒等写像を計算するが、$x = 0$ では定数側の分岐を実行するため、二重数による実行は傾き 0 を報告する。これは、$\mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$ について 34.1.6.3 節で見た失敗と同じである。入力の ε 成分（ここでは 1）は**シード**と呼ばれる。f' の数式表現が作られることは一度もなく、ステップサイズを選ぶ必要もない。生じる誤差は浮動小数点の丸め誤差のみである。

34.5.1.2 フォワードモードのエンジン

演算子オーバーロードを使えば、この代数をそのままコードに落とし込める。以下の `Dual` クラスは、値 a とその微分パート b を組にして保持する。この微分パートを**タンジェント**と呼び、$\dot{v}$ と書く（元の量を v とする）。このクラスは Equation 34.43 を $\star$ に対して実装し、`sin`、`exp`、`log` にプリミティブの微分規則を備えている。クラスカウンタ `ops` は実行された基本演算の回数を記録するもので、34.5.2.4 節でコストを計測する際に用いる。

```
class Dual:
    """A dual number a + b*eps with eps^2 = 0; b carries the derivative."""
    ops = 0                                          # primitive operations run
    def __init__(self, a, b=0.0):
        self.a, self.b = a, b                        # value, tangent
    def __add__(self, o):
        o = o if isinstance(o, Dual) else Dual(o)
        Dual.ops += 1
        return Dual(self.a + o.a, self.b + o.b)
    __radd__ = __add__
    def __mul__(self, o):
        o = o if isinstance(o, Dual) else Dual(o)
        Dual.ops += 1
        return Dual(self.a * o.a, self.a * o.b + self.b * o.a)   # product rule
    __rmul__ = __mul__
    def sin(self):
        Dual.ops += 1
        return Dual(np.sin(self.a), np.cos(self.a) * self.b)     # chain rule
    def exp(self):
        Dual.ops += 1
        return Dual(np.exp(self.a), np.exp(self.a) * self.b)
    def log(self):
        Dual.ops += 1
        return Dual(np.log(self.a), self.b / self.a)
    def __repr__(self):
        return f'{np.round(self.a, 6)} + {np.round(self.b, 6)} eps'
```

`Dual` を $f(x) = \sin(x^2) + e^x$ の $x = 1.3$ においてテストしてみよう。引数に $1.3 + \varepsilon$ を渡し、出力の ε 成分を手計算で求めた導関数 $f'(x) = 2x\cos(x^2) + e^x$ と比較する。

```
def f(x):
    return (x * x).sin() + x.exp()                   # sin(x^2) + e^x, in dual algebra

x0 = 1.3
out = f(Dual(x0, 1.0))                               # seed the derivative with 1
exact = 2 * x0 * np.cos(x0**2) + np.exp(x0)          # f'(x) by hand
print('dual result :', out)
print('value f(x)  :', np.sin(x0**2) + np.exp(x0))
print("derivative  : dual %.6f  vs  exact %.6f" % (out.b, exact))
```

```
dual result : 4.6622 + 3.360101 eps
value f(x)  : 4.662200318713363
derivative  : dual 3.360101  vs  exact 3.360101
```

1 回の実行で関数値 4.6622 と導関数の値 3.360101 の両方が返され、表示されたすべての桁で解

析的な公式と一致している。

しかし、1 変数の例ではフォワードモードの最大の制約が隠れてしまう。3 次元入力 $\mathbf{x} = (1, -0.5, 2)$ における log-sum-exp を考えよう。その勾配はソフトマックス $\partial \ell / \partial x_j = e^{x_j} / \sum_k e^{x_k} = p_j$ である。この勾配全体を得るのに、フォワードモードは何回のパスを必要とするだろうか。入力 j にシードとして 1 を、他のすべての入力に 0 を与えると、単一の偏微分 $\partial \ell / \partial x_j$ が出力の ε 成分に現れ、それ以外は何も得られない。

```
def lse_dual(xs):                                   # log-sum-exp, in dual algebra
    total = 0.0
    for x in xs:
        total = total + x.exp()
    return total.log()

x_lse = [1.0, -0.5, 2.0]
p = np.exp(x_lse) / np.exp(x_lse).sum()             # softmax, the known gradient
for j in range(3):                                  # one pass per input direction
    out = lse_dual([Dual(x, 1.0 if k == j else 0.0) for k, x in enumerate(x_lse)])
    print(f'seed e{j + 1}: value {out.a:.6f}, partial {out.b:.6f}, '
          f'softmax {p[j]:.6f}')
```

```
seed e1: value 2.371539, partial 0.253716, softmax 0.253716
seed e2: value 2.371539, partial 0.056612, softmax 0.056612
seed e3: value 2.371539, partial 0.689672, softmax 0.689672
```

3 回のパスが必要になる。各パスは同じ値 2.371539 を再計算しながら、ソフトマックスの要素を 0.253716、0.056612、0.689672 と 1 つずつしか取り出せない。より一般に、n 個の入力に関する勾配を求めるには n 回のパスを要する。冒頭で挙げた 10^8 パラメータのネットワークでは、結局プログラムを 10^8 回実行することに逆戻りしてしまう。フォワードモードは有限差分のステップサイズこそ不要にするが、実行回数は減らせないのである。

34.5.1.3 ベクトルタンジェントとヤコビ行列の列

次に、ベクトル値写像 $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ と、その $m \times n$ ヤコビ行列 $\mathbf{J}$ を考えよう。1 回の順伝播は何を計算するだろうか。任意の方向 $\mathbf{v}$ を入力のシードとし、入力 j を $x_j + v_j \varepsilon$ とおくとしよう。先ほどの命題を 1 次元のスライス $t \mapsto f_i(\mathbf{x} + t\mathbf{v})$ に適用すると、出力 i は $f_i(\mathbf{x}) + (\nabla f_i \cdot \mathbf{v})\,\varepsilon$ として現れることがわかる。したがって、ε 成分を m 個すべての出力にわたって集めると、ベクトル $\mathbf{J}\mathbf{v}$ が得られる。これは**ヤコビ行列・ベクトル積**（Jacobian–vector product、JVP）と呼ばれる。$\mathbf{v}$ が j 番目の標準基底ベクトル $\mathbf{e}_j$ であるとき、JVP は第 j 列を $\mathbf{J}$ から取り出す。すなわち、1 回の順伝播は 1 つの入力方向を写像全体にわたって伝播させる。

これを平面写像 $\mathbf{f}(x, y) = (x + \sin y,\; y + x^2/2)$ の点 $(0.5, 0.5)$ において確かめてみよう。第 i 行が出力 i の偏微分を並べたものになっているヤコビ行列は、次のようになる。

$$\mathbf{J} = \begin{pmatrix} 1 & \cos y \\ x & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0.877583 \\ 0.5 & 1 \end{pmatrix}.$$

まず $\mathbf{e}_1$、次に $\mathbf{e}_2$ をシードとして 2 回の順伝播を実行すると、ヤコビ行列の 2 つの列が復元される。

```
def plane_dual(x, y):
    return [x + y.sin(), y + 0.5 * (x * x)]        # (x + sin y, y + x^2/2)

z0 = (0.5, 0.5)
cols = []
for seed in ((1.0, 0.0), (0.0, 1.0)):              # e1, then e2
    out = plane_dual(Dual(z0[0], seed[0]), Dual(z0[1], seed[1]))
    cols.append([out[0].b, out[1].b])              # tangent = J @ seed, a column
J_fwd = np.array(cols).T                           # columns side by side
J_exact = np.array([[1.0, np.cos(0.5)], [0.5, 1.0]])
print('forward-mode J:\n', J_fwd.round(6))
print('matches the analytic Jacobian:', np.allclose(J_fwd, J_exact))
```

```
forward-mode J:
 [[1.       0.877583]
 [0.5      1.      ]]
matches the analytic Jacobian: True
```

ここで、Dual の実装にはタンジェントをスカラーに限定する制約が何もないことに注目しよう。x にベクトルタンジェント $(1,0)$ を、y にベクトルタンジェント $(0,1)$ を与えれば、すべての演算が両方の入力方向を同時に伝播させ、出力 i のタンジェントは第 i 行を $\mathbf{J}$ から取り出したものになる。では、1 回のパスで行列全体を復元できるのだろうか。

```
out = plane_dual(Dual(z0[0], np.array([1.0, 0.0])),     # identity seed:
                 Dual(z0[1], np.array([0.0, 1.0])))     # both directions at once
J_one = np.array([out[0].b, out[1].b])                  # row i = tangent of f_i
print('one-pass J:\n', J_one.round(6))
print('matches the analytic Jacobian:', np.allclose(J_one, J_exact))
```

```
one-pass J:
 [[1.       0.877583]
 [0.5      1.      ]]
matches the analytic Jacobian: True
```

復元はできるが、計算量の節約は見かけ倒しである。タンジェントの各演算が長さ n の配列に対して作用するようになるため、その 1 回のパスはスカラーのパスに比べておよそ n 倍の算術演算を実行するからである。パス数を数えることと算術演算の回数を数えることは別問題であり、算術演算数は 34.5.2.4 節で直接計測する。10^8 個の入力を持つスカラー損失に対して、フォワードモードは 10^8 回のスカラーパスを実行するか、長さ 10^8 のタンジェントを伴う 1 回のパスを実行するかのいずれかを必要とする。どちらにしても、総計算量はプログラムの実行 n 回分に比例して増大する。

34.5.2 リバースモードとテープ

34.5.2.1 1 つの積に対する 2 つの計算順序

n 倍という計算量の増大をなくすには、導関数の連鎖を逆の順序で掛け合わせる必要がある。34.4 節で見たように、K ステップのプログラム $\mathbf{f} = \mathbf{f}_K \circ \cdots \circ \mathbf{f}_1$ のヤコビ行列は、各ステップのヤコビ行列の積で表される。

$$\mathbf{J} = \mathbf{J}_K \mathbf{J}_{K-1} \cdots \mathbf{J}_1, \tag{34.45}$$

そして、この積にどう括弧をつけるかによってコストが決まることも見た。たとえば幅 1000 の 3 つの層に左から行ベクトルを掛ける場合、左から順に結合すれば乗算は 3×10^6 回で済むのに対し、右から結合すると 2×10^9 回を要する（Figure 34.22）。自動微分エンジンは、これらのヤコビ行列を明示的に実体化することは決してない。その代わり、シードベクトルを 1 ステップずつ順に伝播させる。フォワードモードは列方向のベクトル $\mathbf{v}$ を右から掛け、リバースモードは行ベクトル $\mathbf{u}^\top$ を左から掛ける。

$$\mathbf{J}\mathbf{v} = \mathbf{J}_K\Big(\cdots(\mathbf{J}_2(\mathbf{J}_1\mathbf{v}))\Big), \qquad \mathbf{u}^\top\mathbf{J} = \Big(((\mathbf{u}^\top\mathbf{J}_K)\mathbf{J}_{K-1})\cdots\Big)\mathbf{J}_1. \tag{34.46}$$

最初からシードベクトルが存在するため、どのステップも行列・ベクトル積になり、行列・行列積になることは一度もない。左の式では、入力方向が順方向に伝わって JVP $\mathbf{J}\mathbf{v}$ を生成する。右の式では、出力方向が逆方向に伝わって**ベクトル・ヤコビ行列積**（vector–Jacobian product、VJP）$\mathbf{u}^\top\mathbf{J}$ を生成する。これは $\mathbf{J}$ の各行を $\mathbf{u}$ で重み付けした線形結合である。$\mathbf{u} = \mathbf{e}_i$ とおけば、第 i 行を $\mathbf{J}$ から取り出せる。

この対称性から、自動微分の基本的なコスト規則が導かれる。フォワードモードは入力方向ごとに 1 回のスイープを要し、リバースモードは出力方向ごとに 1 回のスイープを要するが、各スイープのコストはプログラムを 1 回評価する場合の小さな定数倍にすぎない。$+, -, \times, \div$ から構成されるプログラムについて、Baur と Strassen は、すべての偏微分をまとめて計算するのに必要な演算回数が、数える演算の種類に応じてプログラム自体の演算回数の高々一定倍（3〜5 倍）に収まることを証明した (Baur and Strassen 1983)。スカラー損失のヤコビ行列は $1 \times n$、すなわち 1 行だけであるため、1 をシードとする 1 回の逆方向スイープで勾配全体が一度に返される。Figure 34.23 は、この規則を行列の形状の観点から示したものである。

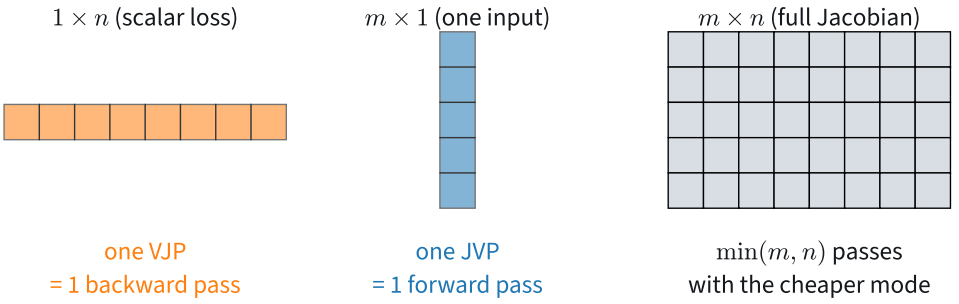


Figure 34.23: ヤコビ行列の形状によって低コストなモードが決まる。スカラー損失のヤコビ行列は $1 \times n$ であり、1 行、1 回の VJP、1 回の逆伝播で済む。1 入力の写像のヤコビ行列は $m \times 1$ であり、1 列、1 回の JVP、1 回の順伝播で済む。$m \times n$ ヤコビ行列全体を求めるコストは、低コストな方のモードを使えば $\min(m, n)$ 回のパスになる。

2 つのモードは、まったく同じ行列積を反対側の端から評価している。どちらが有利かは $\mathbf{J}$ の形状だけで決まり、シードの個数が短い方の辺に一致するモードの方が低コストになる。1 つのスカラー出力と数百万から数十億個のパラメータを持つニューラルネットワークの損失は、考えられる限り最も極端に縦横比が偏ったヤコビ行列を持っているのである。

34.5.2.2 テープ

フォワードモードとリバースモードが伝播させる量は、根本的に異なる。入力 x を出力 y に写すプログラムにおいて、タンジェント $\dot{v}$（中間変数 v に対するもの）は、v が入力の $\dot{x}$ 方向への移

動に伴ってどれだけ速く変化するかを測る。これに対し、**随伴変数**（adjoint）$\bar{v}$（34.3 節で導入）は、最終出力 y の v に関する全微分であり、v から y へのすべての経路にわたって和をとったものである。タンジェントは v より**前**のステップのみに依存するのに対し、随伴変数は v より**後**のステップに依存する。たとえば $y = v^2$（ただし $v = 3x$）では、下側から伝わる $\dot{v} = 3\dot{x}$ と、上側から伝わる $\bar{v} = 2v$ が得られる。

この非対称性が、各モードの実行方式を決定づける（Figure 34.24）。タンジェントは値と同じ方向に進むため、フォワードモードはプログラム自体の実行と並行して 1 回のパスで完了し、`Dual` が過去の中間値を記憶しておく必要はまったくない。しかし随伴変数は、実行の流れに逆らって進む。あるノードの随伴変数を計算するには、そのノードを利用するすべてのノードの随伴変数が必要であり、それらはプログラムが出力に到達した後にしか存在しないからである。したがって、リバースモードは 2 段階で動作する。まずプログラムを順方向に実行しながらすべての基本演算を記録し、次にその記録を逆向きに再生する。この記録は、Wengert (1964) がフォワードモードのために導入した基本演算のリストにちなんで、**テープ**または Wengert リストと呼ばれる。このリストを逆順にたどることで、Linnainmaa (1970) が導入したリバースモードが得られる。

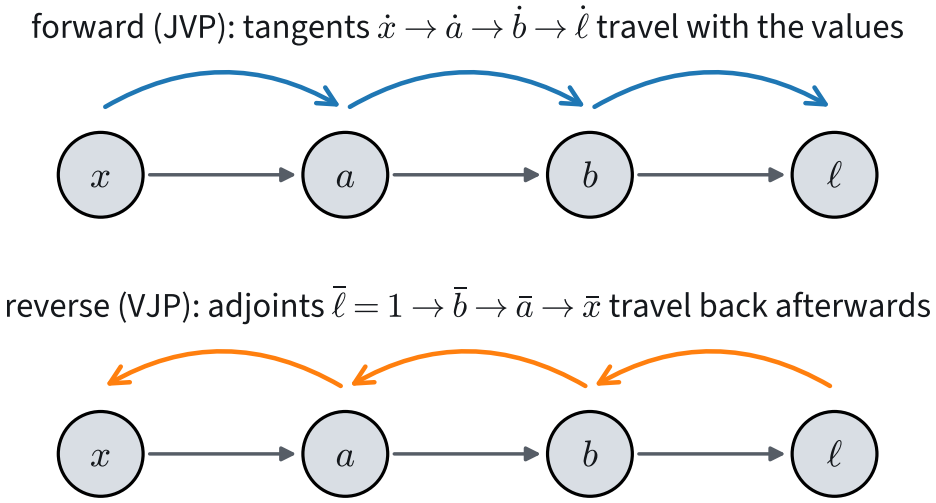


Figure 34.24: 連鎖 $x \to a \to b \to \ell$ におけるフォワードモードとリバースモード。どちらもまず左から右へ連鎖を評価する（灰色）。フォワードモード（上）は値とともにタンジェント $\dot{v}$ を運び、1 回のスイープで 1 つの入力方向を処理する。リバースモード（下）は先に連鎖を実行してから、随伴変数 $\bar{v}$（ℓ の v に関する全微分）をシード $\bar{\ell} = 1$ から逆向きに運び、1 回のスイープで 1 つの出力方向（スカラー損失の場合は勾配全体）を求める。

実用的に動くリバースモードのエンジンは、約 40 行の Python コードで構築できる。テープ上の各ノードは、順伝播の値、親ノード、および小さな逆伝播クロージャを保持する。そのノード自身の随伴変数 g が与えられると、クロージャは各親ノードの随伴変数にこのノードからの寄与を加算する。すなわち、$c = a + b$ に対しては両方の親に g を足し、$c = ab$ に対しては $b\,g$ を $\bar{a}$ に、$a\,g$ を $\bar{b}$ に足す。関数 `backprop` は、すべてのノードがその親より後に現れるようにグラフをトポロジカルソートし、出力の随伴変数にシードとして 1 を設定したうえで、クロージャを逆順に呼び出す。`Dual` と同様に、このクラスも順伝播と逆伝播の両ステップにわたって `ops` カウンタを管理する。

```
class Var:
    """A scalar node on the autodiff tape; .grad accumulates the adjoint."""
    ops = 0                                       # recorded ops + backward steps
    def __init__(self, value, parents=(), backward=lambda g: None):
        self.value = value
        self.grad = 0.0
        self._parents = parents                   # nodes this one depends on
        self._backward = backward                 # pushes adjoint to parents
    def __add__(self, o):
        o = o if isinstance(o, Var) else Var(o)
        Var.ops += 1
        out = Var(self.value + o.value, (self, o))
        def back(g):                              # d(a+b) = 1*g to each parent
            Var.ops += 1
            self.grad += g; o.grad += g
        out._backward = back
        return out
    __radd__ = __add__
    def __mul__(self, o):
        o = o if isinstance(o, Var) else Var(o)
        Var.ops += 1
        out = Var(self.value * o.value, (self, o))
        def back(g):                              # product rule, reversed
            Var.ops += 1
            self.grad += o.value * g; o.grad += self.value * g
        out._backward = back
        return out
    __rmul__ = __mul__

def backprop(node):
    """Topologically sort the tape and replay it backward from `node`."""
    order, seen = [], set()
    def visit(v):
        if v not in seen:
            seen.add(v)
            for p in v._parents:
                visit(p)
            order.append(v)
    visit(node)
    node.grad = 1.0                               # seed the output adjoint
    for v in reversed(order):
        v._backward(v.grad)
```

`Var` が実装しているのは `+` と `*` だけだが、3 つ目の例である $y = (uv + u)^2$ を扱うにはこれで十分である。コードを実行する前に、$u = 2$、$v = -3$ における逆方向スイープ全体を手計算でトレースしてみよう。順伝播は 3 つの基本演算、すなわち $t = uv = -6$、次に $r = t + u = -4$、最後に $y = r \cdot r = 16$ を記録する。逆方向スイープは y, r, t, u, v のような逆トポロジカル順序でノードを訪問し、各ノードがそれに依存するすべてのノードより後に処理されることを保証する（`backprop` はたまたま v を u より先に訪問するが、これも同様に正しい順序である）。各随伴変数は出力辺に沿って届く寄与の総和であり、それぞれの寄与は利用先ノードの随伴変数に局所導関数を掛けたものになる。

ノード	出力辺とその寄与	随伴変数
y	なし（シード）	$\bar{y} = 1$
r	$y = r \cdot r$ の両方の引数：$\bar{y}\,r + \bar{y}\,r$	$\bar{r} = -8$
t	$r = t + u$: $\bar{r} \cdot 1$	$\bar{t} = -8$
u	$r = t + u$ および $t = uv$：$\bar{r} \cdot 1 + \bar{t} \cdot v$	$\bar{u} = -8 + 24 = 16$
v	$t = uv$: $\bar{t} \cdot u$	$\bar{v} = -16$

$y = (uv + u)^2$ を直接微分すれば、$\partial y/\partial u = 2(uv + u)(v + 1) = 2(-4)(-2) = 16$ および $\partial y/\partial v = 2(uv + u)\,u = 2(-4)(2) = -16$ となり、両方の答えが正しいことが確かめられる。グラフ内の 2 箇所、すなわち r が最後の積の両方の引数に入るところと、u が t と r の両方に入るところで、加算による累積がいかに重要かに注目しよう（Figure 34.25）。もし加算ではなく上書きしてしまうと $\bar{r} = -4$ になり、仮に $\bar{r}$ を正しく直したとしても、どちらのクロージャが最後に実行されたかによって $\bar{u}$ が -8 または 24 になってしまう。随伴変数はすべての出力辺にわたって和をとる（連鎖律における全経路の和に対応する）ため、すべての逆伝播クロージャは `+=` で累積しなければならない。

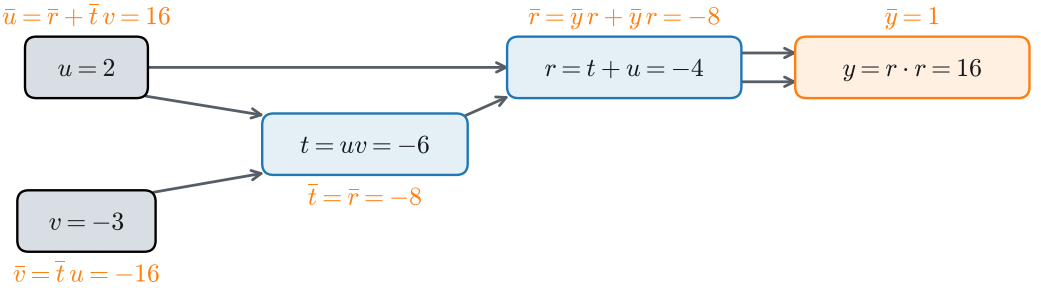


Figure 34.25: $y = (uv + u)^2$ の $u = 2$、$v = -3$ におけるテープ。各ボックスはそのノードの順伝播の値を示し、オレンジ色のラベルはその随伴変数、すなわちそのノードに関する y の導関数を示す。グラフは一直線の連鎖ではなくひし形になっている。u は $t = uv$ と $r = t + u$ の両方に入り、r は $y = r \cdot r$ の両方の引数に入るため、r と u の随伴変数はそれぞれ 2 つの寄与の和となる。

同じ式を `Var` で記録し、1 回の逆方向スイープを実行すると次の結果が得られる。

```
u, v = Var(2.0), Var(-3.0)
r = u * v + u                                   # t = uv, then r = t + u
y = r * r                                       # r feeds both arguments
backprop(y)
print('our tape  : dy/du = %.4f  dy/dv = %.4f' % (u.grad, v.grad))
```

```
our tape  : dy/du = 16.0000  dy/dv = -16.0000
```

これらは手計算のトレースで求めた $\bar{u}$ と $\bar{v}$ に一致する。実用レベルの深層学習フレームワークも、まったく同じアルゴリズムをテンソル規模で実行し、同じ 2 つの導関数の値を出力する。

```
def g(u, v):
    r = u * v + u
```

```
    return r * r
du, dv = jax.grad(g, argnums=(0, 1))(2.0, -3.0)     # framework autograd
print('jax       : dy/du = %.4f  dy/dv = %.4f' % (float(du), float(dv)))
```

```
jax       : dy/du = 16.0000  dy/dv = -16.0000
```

実用フレームワークは、ここで作った簡易的な Var エンジンと実務上の 3 つの点で異なるが、いずれも根底にあるアルゴリズムを変えるものではない。第 1 に、ノードはスカラーではなくテンソル全体を保持するため、各逆伝播クロージャは行列積に対する $\bar{\mathbf{X}} = \bar{\mathbf{Y}}\mathbf{W}^\top$ や $\bar{\mathbf{W}} = \mathbf{X}^\top\bar{\mathbf{Y}}$（34.4.3.2 節）のように、テンソル演算全体の VJP を実装する。第 2 に、プリミティブのライブラリがはるかに充実しており、それぞれに人手で最適化された VJP 規則が組み合わされている。第 3 に、フレームワークが計算グラフを構築する方式には 2 通りある。**即時テープ**（eager tape）は、Var と同じようにプログラムの実行に合わせて動的に演算を記録する。これに対し**トレーサ**は、抽象的なプレースホルダ入力に対してプログラムを 1 回実行して演算の系列を捕捉し、各ステップの線形化されたフォワードモード規則を構築したうえで、その系列を転置して逆順にする。どちらの方式でも、得られる逆方向スイープはまったく同じである。ニューラルネットワークに適用したとき、このリバースモードのスイープこそが深層学習で**誤差逆伝播法**と呼ばれるものである (Rumelhart et al. 1986)。

JAX はトレーサ方式をとる。jax.grad は関数をトレースし、転置された逆方向スイープを実行する新しい Python 関数を返す。

各フレームワークはまた、二重数の命題における仮定、すなわち各プリミティブが受け取る引数において微分可能でなければならないという条件も引き継いでいる。実務上、この仮定が最も頻繁に破られるのがマスク処理である。パディングされたベクトルの有効な要素にわたって $\log x_i$ の和をとり、where によってパディング要素を 0 に置き換える場合を考えよう。where はすでに計算された 2 つの値のうち一方を選ぶ演算であるため、パディング要素（ここでは $x_2 = 0$）に対しても log が実行されてしまう。順伝播では問題は生じない。$-\infty$ は捨てられ、損失は $\log 2 \approx 0.6931$ となる。しかし逆方向スイープでは、where が捨てられた側の分岐にも随伴変数 0 を送り込み、log の VJP がそこに局所導関数 $1/0 = \infty$ を掛ける。浮動小数点演算では、$0 \cdot \infty$ は 0 ではなく NaN になる。

```
x = jnp.array([2.0, 0.0])                           # x[1] is padding
mask = jnp.array([True, False])
def one(x):                                         # sum of log x, real entries
    return jnp.where(mask, jnp.log(x), 0.0).sum()
def two(x):
    safe = jnp.where(mask, x, 1.0)                  # padding never reaches log
    return jnp.where(mask, jnp.log(safe), 0.0).sum()
print('one where   : loss = %.4f, grad =' % one(x), jax.grad(one)(x))
print('double where: loss = %.4f, grad =' % two(x), jax.grad(two)(x))
```

```
one where   : loss = 0.6931, grad = [0.5 nan]
double where: loss = 0.6931, grad = [0.5 0. ]
```

1 行目は勾配 $(0.5, \mathrm{NaN})$ を報告しているが、損失はパディング要素にまったく依存しておらず、そこでの真の導関数の値は 0 である。たった 1 つの NaN でも局所的な問題では済まず、x_2 の上流にあるすべてのパラメータへと伝播してしまう。2 行目は、標準的な対処法である**ダブル where** を示している。内側の where が、プリミティブに渡る前にマスクされた要素を無害な値（ここでは 1）に置き換えるため、無限大の局所導関数が作られることはなくなり、勾配は $(0.5, 0)$ となる。引数を定義域の端から離すようにクランプする手法も同じ役割を果たす。同じ落とし穴は $\sqrt{x}$ の

0 における評価や、それに伴うノルム $\sqrt{\sum_i x_i^2}$ の $\mathbf{x} = \mathbf{0}$ における計算にも潜んでいる。

34.5.2.3 パラメータ勾配

ここまでヤコビ行列の積 Equation 34.45 はネットワークの入力に関する導関数を追跡してきたが、訓練で必要になるのは重みに関する導関数である。幸いなことに、重みの勾配は層の VJP 規則（34.4.3.2 節）を通じて、まったく同じ逆方向スイープから自然に得られる。線形層 $\mathbf{h}_k = \mathbf{W}_k \mathbf{h}_{k-1}$（重み行列 $\mathbf{W}_k$ の形状は $d_k \times d_{k-1}$）に対して、この規則は次式で与えられる。

$$\nabla_{\mathbf{W}_k} \ell = \bar{\mathbf{h}}_k \, \mathbf{h}_{k-1}^{\top}, \tag{34.47}$$

これは、上側から届く随伴変数と下側から保存しておいた活性化値との外積である。直感的には、ある重みの重要度は、その出力が損失にどれだけ強く影響するかという度合いと、入力の活性化値がどれだけ大きかったかという度合いの双方に比例する。コスト解析において重要なのは、この規則がどれだけの算術演算を要するかである。

外積 $\bar{\mathbf{h}}_k \, \mathbf{h}_{k-1}^{\top}$ を作るには $d_k d_{k-1}$ 回の乗算を要し、これは順伝播の行列・ベクトル積 $\mathbf{W}_k \mathbf{h}_{k-1}$ と同じ回数である。随伴変数を前の層へ引き戻す計算 $\bar{\mathbf{h}}_{k-1}^{\top} = \bar{\mathbf{h}}_k^{\top} \mathbf{W}_k$ にも、さらに $d_k d_{k-1}$ 回の乗算がかかる。したがって乗算回数で測ると、逆方向スイープのコストは順伝播のおよそ 2 倍となり、順伝播と逆方向スイープを合わせたコストは順伝播の約 3 回分となって、冒頭で述べた 30 ms という数値と一致する。対照的にフォワードモードでは、個々の重みをそれぞれ独立な入力方向として扱わなければならず、たった 1 つの層だけでも $d_k d_{k-1}$ 回のパスが必要になってしまう。どのフレームワークもフォワードモードでニューラルネットワークを訓練しない理由はここにある。

34.5.2.4 2 つのモードのコスト

両方のエンジンを自前で構築したので、漸近的な議論に頼らず演算回数を直接計測できる。$f(\mathbf{x}) = (\sum_j x_j^2)^2$ を、$n = 200$ 個の入力 $x_j = 0.01\,j$（$j = 1, \dots, 200$）に対して考えよう。$+$ と $\times$ だけから構成される多項式を選んだのは、`Var` に `exp` や `log` のクロージャ（`Dual` のメソッドと対になるもの）を追加しなくてもそのまま記録できるようにするためである。この関数の勾配は $\nabla f = 4\big(\sum_j x_j^2\big)\mathbf{x}$ であり、通常の評価 1 回あたり $2n + 1 = 401$ 回の基本演算（n 回の二乗、n 回の加算、そして最後の 1 回の二乗）を実行する。フォワードモードが 200 回のパスでそれぞれ 401 回の演算を必要とすることは明らかである。以下の出力を読む前に少し立ち止まって、リバースモードが何回の演算を要するか予想してみてほしい。

```
n = 200
def f(xs):                        # f: R^n -> R, from + and * alone
    total = 0.0
    for x in xs:
        total = total + x * x
    return total * total          # (sum of squares)^2

xs0 = [0.01 * (j + 1) for j in range(n)]
Dual.ops = 0
grad_fwd = [f([Dual(x, 1.0 if k == j else 0.0) for k, x in enumerate(xs0)]).b
            for j in range(n)]    # forward mode: one pass per input direction
Var.ops = 0
xs = [Var(x) for x in xs0]        # reverse mode: one recorded pass...
```

```
    backprop(f(xs))                 # ...and one backward sweep for all n partials
    grad_rev = [x.grad for x in xs]

    gap = max(abs(a - b) for a, b in zip(grad_fwd, grad_rev))
    print(f'forward mode: {Dual.ops:,} operations ({n} passes of {2 * n + 1})')
    print(f'reverse mode: {Var.ops:,} operations (one recorded pass, one sweep)')
    print(f'ratio {Dual.ops / Var.ops:.1f}, largest gradient difference {gap:.1e}')
```

```
forward mode: 80,200 operations (200 passes of 401)
reverse mode: 802 operations (one recorded pass, one sweep)
ratio 100.0, largest gradient difference 0.0e+00
```

フォワードモードが $80{,}200 = n(2n+1)$ 回の演算を実行したのに対し、リバースモードはわずか $802 = 2(2n+1)$ 回（順伝播で記録された 401 回と、逆方向スイープ中の逆伝播クロージャ 401 回）しか実行しなかった。その比は $100 = n/2$ であり、2 つの勾配は表示されたすべての桁で一致している。これらのエンジンでは、パス数は n 倍、演算回数は $n/2$ 倍異なり、$n = 10^8$ においては小さな定数倍を除いて冒頭の倍率と一致する。

一見すると、ここで計測された 2 倍という倍率は、先ほど挙げた 3 倍という数字と矛盾するように見えるかもしれない。しかし矛盾はない。乗算の逆伝播クロージャは内部で 2 回の乗算を行っているにもかかわらず、`Var.ops` は各クロージャを 1 回の演算として数えているからである。スカラー乗算の回数を直接数えれば、逆方向スイープの 402 回に対して順伝播は 201 回であり、線形層の場合とまったく同様に、値と勾配を合わせて計算するコストは値のみの場合の 3 倍になる。随伴変数を累積する加算まで含めると逆方向スイープは 1204 回の演算となり、全体の倍率は 4 倍になるが、それでもなお Baur–Strassen の上界の範囲内に十分収まっている。

リバースモードは、この速度と引き換えにメモリを消費する。逆方向スイープが使い終えるまでテープは記録された 401 個すべての中間値を保持し続けなければならないのに対し、フォワードモードは元のプログラムと同様、現在の値とそのタンジェントだけを保持すればよい。

34.5.2.5 勾配チェックポイント

記録されるステップ数が T の計算では、逆方向スイープが到達するまで T 個の中間値をメモリ上に保持し続ける。深層ニューラルネットワークにおいて、それらの値とはバッチ内の各データ例に対するすべての層の活性化値であり、バッチサイズやネットワークの深さの上限を決める要因は、算術演算量ではなく活性化メモリであることが多い。

最も単純な対処法は、テープを s ステップずつの区間に分割することである。順伝播の間は s ステップごとの値、すなわち T/s 個の**チェックポイント**だけを保存し、その間にある中間値はすべて破棄する。逆方向スイープがある区間に到達したら、保存しておいたチェックポイントからその区間の s ステップを一時的なテープ上に再計算し、その短いテープを逆方向にスイープした後、ただちに解放する。どの時点においても、メモリが保持するのは T/s 個のチェックポイントと、処理中の長さ s の区間 1 つ分だけである。これら 2 つの項が釣り合うように s を選べば、ピークメモリが最小化される。

$$M(s) = \frac{T}{s} + s, \qquad \min_{s} M(s) = M(\sqrt{T}) = 2\sqrt{T}. \tag{34.48}$$

この最小値は、$M'(s) = 1 - T/s^2 = 0$ とおくか、あるいは相加・相乗平均の不等式 $T/s + s \geq 2\sqrt{T}$ から直接得られる。これによりピークメモリは計算長に対して線形ではなく、その平方根のオーダーでしか増大しなくなる。各ステップは 1 回だけ再計算されるため、追加の計算量は順伝播わずか 1 回分である。$T = 400$ の場合、区間長 $s = 1, 5, 10, 20, 100, 400$ に対するピークメ

モリは $M = 401, 85, 50, 40, 104, 401$ となり、$s = 20 = \sqrt{400}$ で最小値をとる。$T = 10^4$ では、長さ $s = 100$ の区間を用いることで、保存する値はわずか 200 個で済み、10,000 個すべてを保持する必要がなくなる (Chen et al. 2016)。Figure 34.26 にこのスケジュールを示す。

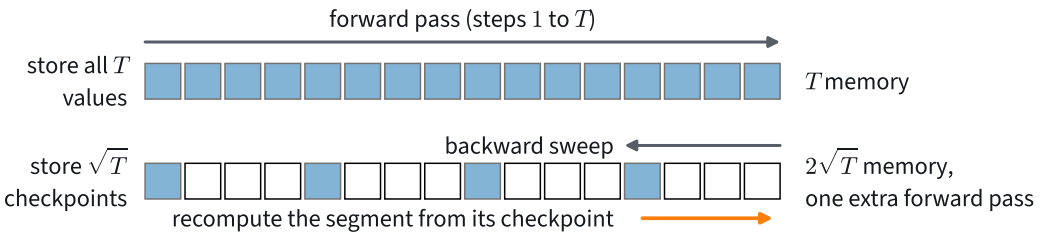


Figure 34.26: T ステップのテープにおける勾配チェックポイント。通常のリバースモード（上）は、逆方向スイープが使用し終えるまですべての T 個の値を保持する。チェックポイント保存（下）は s ステップごとの値だけを保持し、逆方向スイープがある区間に到達すると、その区間をチェックポイントから再計算してからスイープする。順伝播 1 回分の追加計算により、メモリ使用量は $T/s + s$、すなわち $2\sqrt{T}$（$s = \sqrt{T}$ のとき）となる。

重要な点として、再計算が決定論的である限り、チェックポイント保存は勾配を一切変えることなく計算時間とメモリ使用量だけを変化させる。再計算された中間値が元の値と完全に同一になるため、得られる導関数の値も通常のリバースモードとビット単位で一致するからである。深層学習フレームワークはこのトレードオフをネットワークのブロック単位で提供しており、チェックポイント対象ブロックの入力だけを保存し、誤差逆伝播法の実行中にそのブロックを再実行する。ただし実務では、この決定論性という前提条件に注意が要る。乱数生成器の状態を復元しなければ再実行されたブロックは新しいドロップアウトのマスクを引き直してしまい、バッチ正規化は移動統計量を 2 回更新してしまい、非決定論的な GPU カーネルは 1 回目の実行結果をビット単位で再現するとは限らない。同じアイデアを各区間の内部に再帰的に適用すれば、メモリ使用量を $O(\log T)$ まで削減できる（再計算のコストは $O(T \log T)$ となる）。

34.5.3 ヤコビ行列積の合成

自動微分は、ここまで簡易エンジンで構築してきた 2 つの線形作用素、すなわちフォワードモードの JVP $\mathbf{Jv}$ とリバースモードの VJP $\mathbf{u}^\top\mathbf{J}$ の両方を提供する。この 2 つを自在に合成できるプリミティブとして扱えるようになると、方向曲率を計算したり、数値ソルバを通して微分したり、必要に応じてヤコビ行列全体を組み立てたりできるようになる。

34.5.3.1 ヤコビ行列・ベクトル積とベクトル・ヤコビ行列積

1.5.4 節では、両方の積に対するフレームワークの API を紹介した。平面写像の点 $(0.5, 0.5)$ において、$\mathbf{v} = \mathbf{u} = (1, 2)$ としてこれらを評価してみよう。先ほど `Dual` で求めたヤコビ行列を用いると、$\mathbf{Jv} = (1 + 2 \cdot 0.8776,\ 0.5 + 2) = (2.7552,\ 2.5)$ および $\mathbf{u}^\top\mathbf{J} = (1 + 2 \cdot 0.5,\ 0.8776 + 2) = (2,\ 2.8776)$ になるはずであり、フレームワークの呼び出し結果もこれらの値と直接一致する。

```
def plane(z):                                       # (x + sin y, y + x^2/2)
    return jnp.stack([z[0] + jnp.sin(z[1]), z[1] + 0.5 * z[0]**2])

z0 = jnp.array([0.5, 0.5])
v, u = jnp.array([1.0, 2.0]), jnp.array([1.0, 2.0])
_, Jv = jax.jvp(plane, (z0,), (v,))                 # forward mode: J v
```

```
    _, vjp_fn = jax.vjp(plane, z0)                    # reverse mode: records once,
    uJ, = vjp_fn(u)                                   # then u^T J per seed
    J = jax.jacfwd(plane)(z0)                         # one JVP per column
    print('J v   =', np.asarray(Jv).round(4))
    print('u^T J =', np.asarray(uJ).round(4))
    print('J     =', np.asarray(J, dtype=float).round(4).tolist())
```

```
J v   = [2.7552 2.5   ]
u^T J = [2.     2.8776]
J     = [[1.0, 0.8776], [0.5, 1.0]]
```

JVP と VJP はどちらも 1 回のスイープ分のコストで済み、導関数に関する情報のベクトルだけを伝播させる（ただし VJP はテープ上に順伝播の中間値も保持する必要がある）。密な $m \times n$ ヤコビ行列を組み立てるには、低コストな方のモードで $\min(m, n)$ 回のスイープを実行するのに加え、数値 mn 個分の記憶領域が必要になる。行列全体を実体化するのが適切なのは、n が小さい場合（この 2×2 の例など）や、確率密度を $\left|\det \mathbf{J}\right|$ によって変換する正規化フロー（normalizing flow）のようにアルゴリズムが行列式を必要とする場合である。

34.5.3.2 ヘッセ行列・ベクトル積

$n = 10^6$ 個のパラメータを持つモデルでは、ヘッセ行列の要素数は 10^{12} 個に達し、float32 で保存すると 4 TB ものメモリを占有する。しかし、二次最適化手法が行列そのものを必要とすることはめったにない。ニュートン法がヘッセ行列を使うのは $\mathbf{H}\mathbf{d} = -\nabla \ell$ を更新ステップ $\mathbf{d}$ について解くためだけであり、ニュートン CG 法（または打ち切りニュートン法）はその連立一次方程式を共役勾配法（conjugate gradient method）で解く。共役勾配法は反復ソルバであり、$\mathbf{H}$ には行列・ベクトル積 $\mathbf{H}\mathbf{v}$ を通じてのみアクセスする (Nocedal and Wright 2006; Martens 2010)。では、$\mathbf{H}\mathbf{v}$ を、$\mathbf{H}$ を一度も作らずに計算できるだろうか。

リバースモードとフォワードモードを合成すれば計算できる。ヘッセ行列は勾配写像 $\nabla \ell$ のヤコビ行列であるため、積 $\mathbf{H}\mathbf{v}$ は単に勾配の JVP であり、方向 $\mathbf{v}$ に沿って勾配がどれだけ速く変化するかを測る量にほかならない。

$$\mathbf{H}\mathbf{v} = \mathbf{J}_{\nabla \ell}\,\mathbf{v} = \left.\frac{d}{dt}\right|_{t=0} \nabla \ell(\mathbf{x} + t\mathbf{v}). \tag{34.49}$$

逆方向スイープが $\nabla \ell$ を評価し、その勾配計算全体に対してタンジェント $\mathbf{v}$ をシードとするフォワードモードの順伝播を適用することで、勾配を $\mathbf{v}$ 方向に微分する。この合成は**フォワード・オーバー・リバース**と呼ばれる (Pearlmutter 1994)。次元 n によらず、そのコストは 1 回の勾配計算の小さな定数倍にとどまる。

log-sum-exp は閉じた形の解を持つ格好のテストケースとなる。その勾配はソフトマックス $\mathbf{p}$ であるため、ヘッセ行列はソフトマックスのヤコビ行列になる（34.4 節、同値な見方としてワンホットベクトル $\mathbf{e}_J$（ただし J は $\mathbf{p}$ からサンプリングされる）の共分散行列でもある）。したがって、次式が成り立つ。

$$\mathbf{H} = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^{\top}, \qquad \mathbf{H}\mathbf{v} = \mathbf{p} \odot \mathbf{v} - \mathbf{p}\,(\mathbf{p}^{\top}\mathbf{v}), \tag{34.50}$$

ここで、$\odot$ は要素ごとの積を表す。右辺の評価に必要な乗算はわずか $3n$ 回であり、$n \times n$ 行列を作ることは一度もない。$\mathbf{x} = (1, -0.5, 2)$ および $\mathbf{v} = (1, 0.5, -1)$ においては、$\mathbf{p} = (0.2537, 0.0566, 0.6897)$、$\mathbf{p}^{\top}\mathbf{v} = -0.4077$ となり、次の結果が得られる。

$$\mathbf{Hv} = (0.3571,\ 0.0514,\ -0.4085).$$

これは二次形式 $\frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ でテストするのとは異なり、本質的な検証になっていることに注目しよう。二次形式では $\mathbf{v}$ における勾配がたまたま $\mathbf{Hv}$ と一致するため、バグのある実装でもテストを通過してしまう恐れがある。一方 log-sum-exp では $\mathbf{H}$ が $\mathbf{x}$ に応じて変化し、$\mathbf{v}$ における勾配 $\mathrm{softmax}(\mathbf{v}) = (0.574, 0.348, 0.078)$ は $\mathbf{Hv}$ とは似ても似つかない値になる。

両方のモードが利用できれば、フォワード・オーバー・リバースはわずか 1 行のコードで書ける。

```
x = jnp.array([1.0, -0.5, 2.0])
v = jnp.array([1.0, 0.5, -1.0])
_, Hv = jax.jvp(jax.grad(jax.nn.logsumexp), (x,), (v,))  # forward over reverse
p = jax.nn.softmax(x)
print('forward over reverse :', np.asarray(Hv).round(4))
print('p*v - p (p.v)        :', np.asarray(p * v - p * (p @ v)).round(4))
```

```
forward over reverse : [ 0.3571  0.0514 -0.4085]
p*v - p (p.v)        : [ 0.3571  0.0514 -0.4085]
```

計算された積は Equation 34.50 と表示された全桁の精度で一致しており、その過程で 3×3 のヘッセ行列は一度も作られていない。演習問題 3 では、本節で実装した `Dual` クラスと `Var` クラスから同じヘッセ行列・ベクトル積を直接構築し、フォワード・オーバー・リバースとリバース・オーバー・リバースを比較する。

34.5.3.3 陰関数微分

興味の対象となるすべての量が陽な公式で与えられるとは限らない。漸化式の不動点、物理系の平衡状態、内側にある最適化問題の最小点など、多くの量は方程式の解として陰に定義され、反復ソルバによって計算される。20 回のニュートン反復をテープに記録し、そのすべてを通して逆伝播することも可能ではあるが、そうするとすべての中間反復点を保存しなければならず、しかもソルバがたまたまたどった特定の軌跡を微分することになる。定義方程式だけから直接導関数を求めることはできないだろうか。

考えられる最も単純な場合、すなわち未知数 1 つの方程式 1 本 $g(x, \theta) = x^2 - \theta = 0$（解は $x^\star = \sqrt{\theta}$）から始めよう。閉じた形の解を微分する代わりに、恒等式 $x^\star(\theta)^2 - \theta \equiv 0$ を θ に関して直接微分する。すると $2x^\star\, dx^\star/d\theta - 1 = 0$ が得られ、これを導関数について解くと次式になる。

$$\frac{dx^\star}{d\theta} = \frac{1}{2x^\star} = \frac{1}{2\sqrt{\theta}},$$

これは確かに $\sqrt{\theta}$ の導関数になっている（値は 0.25、ただし $\theta = 4$ のとき）。決定的に重要なのは、$x^\star$ の陽な公式を一切使わず、方程式 $g(x, \theta) = 0$ と解そのものの値だけを使ったという点である。

一般の結果は**陰関数定理**（implicit function theorem）(Rudin 1976) であり、ここでは証明なしで述べる。$\mathbf{g}(\mathbf{x}, \theta)$ を $\mathbb{R}^d \times \mathbb{R}^p$ から $\mathbb{R}^d$ への写像とし、$\mathbf{g}$ が解 $(\mathbf{x}^\star, \theta)$（$\mathbf{g} = \mathbf{0}$ の解）の近傍で連続微分可能であり、ヤコビ行列 $\partial \mathbf{g}/\partial \mathbf{x}$ がそこで正則であると仮定する。このとき、θ のある近傍において、解は微分可能な関数 $\mathbf{x}^\star(\theta)$ となる。恒等式 $\mathbf{g}(\mathbf{x}^\star(\theta), \theta) \equiv \mathbf{0}$ に連鎖律を適用すると $\frac{\partial \mathbf{g}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}^\star}{\partial \theta} + \frac{\partial \mathbf{g}}{\partial \theta} = \mathbf{0}$ が得られ、したがって次式が成り立つ。

$$\frac{\partial \mathbf{x}^\star}{\partial \theta} = -\Big(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Big)^{-1}\frac{\partial \mathbf{g}}{\partial \theta}. \tag{34.51}$$

右辺の 2 つのヤコビ行列は、どちらも解 $(\mathbf{x}^\star, \theta)$ においてのみ評価される。ソルバが何回の反復を要したかにかかわらず、導関数行列全体の計算コストはパラメータ 1 つあたり $\partial \mathbf{g}/\partial \mathbf{x}$ を係数行列とする連立一次方程式を 1 回解くだけで済む。さらに良いことに、VJP の形にすれば、すべてのパラメータを合わせても連立一次方程式をわずか 1 回解くだけで計算できる。すなわち $\mathbf{u}^\top \partial \mathbf{x}^\star/\partial\theta = -\mathbf{a}^\top \partial \mathbf{g}/\partial\theta$ となり、ここで随伴ベクトル $\mathbf{a}$ は $(\partial \mathbf{g}/\partial \mathbf{x})^\top \mathbf{a} = \mathbf{u}$ の解である。

これと同じ原理が、機械学習におけるいくつもの主要な手法を支えている。ODE ソルバを通して微分する随伴法（adjoint method）（38.3.2 節）、深層平衡モデル（deep equilibrium models）(Bai et al. 2019)（層全体を不動点 $\mathbf{z}^\star = \mathbf{f}(\mathbf{z}^\star, \theta)$ として定義し、Equation 34.51 によって訓練するモデル）、そして内側の最小化問題を通じた勾配ベースのハイパーパラメータ最適化などである。

Equation 34.51 を非線形な 2×2 の連立方程式 $g_1 = x_1^2 + \theta x_2 - 2 = 0$、$g_2 = x_1 x_2 + \theta - 1 = 0$（20 回のニュートンステップで解く）でテストし、陰関数微分による勾配をソルバ出力の中心差分と比較して確かめてみよう。

```
# Implicit gradient dx*/dtheta = -(dg/dx)^{-1} dg/dtheta on a 2x2 system,
# checked against a finite difference of the solver output
def g(x, th):                       # g(x, theta) = 0 defines x*(theta)
    return np.array([x[0]**2 + th * x[1] - 2, x[0] * x[1] + th - 1])

def dg_dx(x, th):                   # Jacobian in x, evaluated at (x, th)
    return np.array([[2 * x[0], th], [x[1], x[0]]])

def solve(th):                      # Newton iteration, from a fixed start
    x = np.array([1.0, 1.0])
    for _ in range(20):
        x = x - np.linalg.solve(dg_dx(x, th), g(x, th))
    return x

th, h = 0.3, 1e-6
xs = solve(th)
dg_dth = np.array([xs[1], 1.0])     # dg/dtheta at the solution
implicit = -np.linalg.solve(dg_dx(xs, th), dg_dth)
fd = (solve(th + h) - solve(th - h)) / (2 * h)
print('implicit-function gradient:', implicit.round(8))
print('finite-difference check   :', fd.round(8))
```

```
implicit-function gradient: [-0.11311451 -0.69322129]
finite-difference check   : [-0.11311451 -0.69322129]
```

2 つのベクトルは有限差分の精度の限界まで一致しており、しかも陰関数の公式には 20 回のニュートン反復が一切現れていない。導関数は $\mathbf{g}$ と解 $\mathbf{x}^\star$ のみに依存するため、同じ根に収束するソルバであればどれを使っても同じ勾配が得られる。

他のあらゆる定理と同様に、陰関数定理にも 2 つの明確な適用限界があり、どちらも実務で容易に観察できる。第 1 に、解において $\partial \mathbf{g}/\partial \mathbf{x}$ が正則でなければならない。スカラーの例 $x^2 = \theta$ では、$\theta = 0$ のとき $\partial g/\partial x = 2x^\star = 0$ となるため公式はゼロ除算を起こすが、これは原点において $\sqrt{\theta}$ の接線が垂直になることを反映している。第 2 に、この公式はソルバが実際に収束してい

ることを前提としている。収束していないときに何が起きるかを見るため、$x^2 = 4$ に対して初期値 $x = 1$ からニュートン法を実行し、$k = 1, 2, 3$ ステップで早期終了させて、陰関数の公式 $1/(2\tilde{x}_k)$（未収束の反復点 $\tilde{x}_k$ で評価したもの）と、k ステップ展開したソルバ自体の導関数（θ に関する中心差分で推定）とを比較してみよう。

```
def newton(theta, k, x=1.0):                        # k Newton steps on x^2 = theta
    for _ in range(k):
        x = x - (x * x - theta) / (2 * x)
    return x

theta, h = 4.0, 1e-6
for k in (1, 2, 3):
    x_k = newton(theta, k)
    unrolled = (newton(theta + h, k) - newton(theta - h, k)) / (2 * h)
    print(f'k={k}: x = {x_k:.5f}, formula {1 / (2 * x_k):.4f}, '
          f'unrolled {unrolled:.4f}, exact 0.25')
```

```
k=1: x = 2.50000, formula 0.2000, unrolled 0.5000, exact 0.25
k=2: x = 2.05000, formula 0.2439, unrolled 0.2900, exact 0.25
k=3: x = 2.00061, formula 0.2499, unrolled 0.2509, exact 0.25
```

1、2、3 ステップ後、ニュートン法の反復点はそれぞれ 2.5、2.05、2.00061 に達する。真の導関数の値 0.25 に対し、陰関数の公式は 0.2000、0.2439、0.2499 を返し、展開したソルバの微分値は 0.5000、0.2900、0.2509 となる。収束前はどちらも不正確である。陰関数の公式はソルバの残差誤差をそのまま引き継ぎ、打ち切られた反復経路を微分する展開微分はこの例ではさらに大きく外れている。対処法は明快で、まずソルバを十分に収束させればよい。

1 つだけ根本的な問題が未解決のまま残っている。自動微分は積 $\mathbf{J}_K \cdots \mathbf{J}_1$ を浮動小数点の丸めの範囲で正確に計算するが、その積の値自体を変えることはできない。多数の層を経るうちに、ヤコビ行列の長い積は依然として幾何級数的に縮小したり爆発したりし得る（34.4 節）。どれほど厳密に求めたところで、10^{-20} という勾配が訓練において 0 と同然であることに変わりはない。勾配消失や勾配爆発を抑えるには、注意深い初期化とアーキテクチャの設計が必要であり、これについては 4.4 節以降で取り上げる。

34.5.4 まとめ

プログラムの導関数はその基本ステップのヤコビ行列の積であり、自動微分はその積をシードベクトルに対して評価することで、すべてのステップを行列・行列積ではなく行列・ベクトル積として計算する。フォワードモードは二重数上でプログラムを実行して 1 パスあたり 1 つのヤコビ行列・ベクトル積（1 つの入力方向）を計算し、リバースモードは順伝播の間にテープを記録してそれを逆向きに再生することで 1 パスあたり 1 つのベクトル・ヤコビ行列積（1 つの出力方向）を計算する。どちらのモードが低コストになるかはヤコビ行列の形状によって決まる。スカラー損失の出力はただ 1 つであるため、1 回の逆方向スイープで全パラメータの勾配が、損失評価のおよそ 3 倍のコストで得られる。関数 $(\sum_j x_j^2)^2$（入力数 $n = 200$）では、リバースモードの演算回数が 802 回であったのに対し、フォワードモードは $80{,}200$ 回を要した。テープのメモリ使用量は計算長 T に比例して増大するが、勾配チェックポイントを用いれば順伝播 1 回分の追加計算と引き換えにピークメモリを $2\sqrt{T}$ に抑えられる。最後に、リバースモードの上にフォワードモードを合成すれば勾配評価 1 回分の小さな定数倍のコストでヘッセ行列・ベクトル積を計算でき、陰関数定理を用いれば収束した解において連立一次方程式を 1 回解くだけでソルバの出力を微分できる。

34.5.5 演習問題

1. [code] **二重数ステップと複素ステップ。** $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ の $\mathbf{x}_0 = (0.5, 0.5)$ におけるヤコビ行列の第 2 列 $\mathbf{J}\mathbf{e}_2 = (\cos 0.5,\ 1)$ を考える。中心差分 $(\mathbf{f}(\mathbf{x}_0 + t\mathbf{e}_2) - \mathbf{f}(\mathbf{x}_0 - t\mathbf{e}_2))/(2t)$ は、打ち切りによる t^2 オーダーの誤差と丸めによる u/t オーダーの誤差を併せ持つ（ここで $u \approx 10^{-16}$ は float64 の丸めの単位である）。

   1. 最適なステップサイズと最小誤差を予測し、その予測を $t = 10^{-1}, 10^{-2}, \dots, 10^{-12}$ に対して確かめよ（およそ 2×10^{-11} の最小誤差が $t = 10^{-5}$ 付近で得られるはずである）。さらに、`Dual` に $\mathbf{e}_2$ をシードとして与えた 1 回のパスと比較せよ。なぜ二重数の計算にはステップサイズが現れないのか。
   2. (a) と同様の実験を第 1 列 $\mathbf{J}\mathbf{e}_1$ について行い、そこでは任意の t に対して中心差分が丸め誤差を除いて厳密になる理由を説明せよ（ヒント：$\mathbf{f}$ は x に関する二次多項式である）。
   3. ε を虚数単位 i に置き換えると、複素ステップ推定値 $\operatorname{Im} \mathbf{f}(\mathbf{x}_0 + it\mathbf{e}_2)/t$ が得られる (Squire and Trapp 1998)。これは本問の関数のように複素引数へ解析接続できる $\mathbf{f}$ に対して有効である。$\mathbf{f}(\mathbf{x}_0 + it\mathbf{e}_2)$ を t の三次まで展開し、この推定値の誤差が $O(t^2)$ であることを示せ。なぜこの推定値ではほぼ等しい数値どうしの減算が生じないのか。
   4. NumPy の複素数演算を用いて $t = 10^{-1}, 10^{-2}, \dots, 10^{-20}$ に対する複素ステップ推定値を計算し、その誤差を (a) の結果と比較せよ。$\varepsilon^2 = 0$ と $i^2 = -1$ のどのような違いが、複素ステップにおける $O(t^2)$ の誤差を生むのか。

2. [code] **ハイパーパラメータ勾配。** 特徴量ごとに 1 つのペナルティ $\lambda_j > 0$ を持つリッジ回帰は、次の最適化問題を解く。

$$\mathbf{w}^\star(\lambda) = \operatorname*{argmin}_{\mathbf{w}} \ \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \sum_{j=1}^{d} \lambda_j w_j^2$$

   ここで $(\mathbf{X}, \mathbf{y})$ は訓練データであり、データ $(\mathbf{X}_{\mathrm{v}}, \mathbf{y}_{\mathrm{v}})$ 上での検証損失は $V(\lambda) = \|\mathbf{X}_{\mathrm{v}}\mathbf{w}^\star(\lambda) - \mathbf{y}_{\mathrm{v}}\|^2$ である。

   1. $\mathbf{w}^\star$ が $\mathbf{g}(\mathbf{w}, \lambda) = (\mathbf{X}^\top\mathbf{X} + \operatorname{diag}(\lambda))\,\mathbf{w} - \mathbf{X}^\top\mathbf{y} = \mathbf{0}$ の解であることを示し、Equation 34.51 を用いて $\partial \mathbf{w}^\star / \partial \lambda$ を導出せよ。
   2. $\nabla_\lambda V = -\mathbf{a} \odot \mathbf{w}^\star$ となることを示せ。ただし $\mathbf{a}$ は行列 $\mathbf{X}^\top\mathbf{X} + \operatorname{diag}(\lambda)$ を係数とする 1 つの連立一次方程式の解である。その右辺を書き下せ。また、$\partial \mathbf{w}^\star / \partial \lambda$ を経由するフォワード方式では d 個のペナルティに対して連立一次方程式を何回解く必要があり、この随伴方式では何回で済むか。
   3. V と随伴勾配を NumPy で実装し、訓練例 50 個、検証例 50 個、特徴量 $d = 10$ 個のランダムなデータで評価せよ。求めた勾配を V の中心差分と比較して検証せよ。

   Alan Edelman および Steven G. Johnson、Matrix Calculus for Machine Learning and Beyond, problem set 2（MIT 18.S096、IAP 2023、問題 1）より改変。

3. [code] **ヘッセ行列・ベクトル積。●** 関数 $f(\mathbf{x}) = \left(\sum_j x_j^2\right)^2$（34.5.2.4 節で入力 `xs0`、$n = 200$ に対して評価したもの）は加算と乗算のみから構成されているため、`Dual` と `Var` の両方で評価できる。

   1. ∇f とヘッセ行列 $\mathbf{H}$ を f に対して導出し、$\mathbf{H}\mathbf{v}$ を $\mathbf{H}$ を一度も構築しない形で表せ。
   2. ランダムな方向 $\mathbf{v}$ を生成し、各入力 x_j を `Var(Dual(x_j, v_j))` として包んで `backprop` を実行せよ。得られた随伴変数の値パートが ∇f をなし、ε 成分が $\mathbf{H}\mathbf{v}$ をなすことを示し、その両方を (a) の結果および 34.5.3.2 節のヘッセ行列・ベクトル積を同じ f に

適用した結果と比較せよ。逆伝播クロージャのどの演算が二重数の算術で実行され、なぜそれによって Equation 34.49 の意味でのフォワード・オーバー・リバース計算になるのか。

3. 今度はテープ自体を入れ子にせよ。各入力を `Var(Var(x_j))` として包んで backprop を実行すると、随伴変数は `Var` オブジェクトになる。それらから $\sum_j (\nabla f)_j v_j$ を作り、それに対して backprop を実行することで、リバース・オーバー・リバースによって $\mathbf{H}\mathbf{v}$ を求めよ。たとえば `Var` オブジェクトの生成数を `Var.__init__` 内のカウンタで数え、両手法の生成数を通常の勾配計算 1 回分と比較せよ。逆伝播を記録しているのはどちらの手法であり、そのメモリコストはどうなるか。

4. [code] **テープ上のチェックポイント保存。**●$x_k = x_{k-1} - 0.01\, x_{k-1}^2$（$k = 1, \dots, T$、ただし $T = 400$、$x_0 = 1$）とおき、x_T を x_0 に関して微分せよ。`Var` には減算がないため、各ステップを `x + (-0.01) * x * x` として記録すること。

   1. 通常の backprop で連鎖全体をテープに記録し、逆方向スイープが始まる時点でテープが保持しているノード数を数えよ。
   2. backprop は出力の随伴変数にシードとして 1 を与えている。これを任意のシードを受け取れるように一般化せよ。区間 $x_i \mapsto x_j$ を新しい葉ノード x_i から記録し、一般化した backprop を x_j からシード $\partial x_T / \partial x_j$ で実行すると、葉ノードの随伴変数に $\partial x_T / \partial x_i$ が返されることを示せ。
   3. Figure 34.26 のスケジュールを s ステップの区間で実装せよ。すなわち、通常の浮動小数点数による順伝播で $x_0, x_s, x_{2s}, \dots$ を保存し、続く逆伝播では最後の区間から最初の区間へ 1 区間ずつ再記録して (b) を適用する。得られた結果が (a) の勾配と一致することを確かめよ。
   4. $s \in \{1, 5, 10, 20, 100, 400\}$ について、保存されるチェックポイント数、ステップ数で測った最大のテープ長、および最初の順伝播を含めたステップ $x \mapsto x - 0.01x^2$ の評価回数を報告せよ。メモリ使用量 $T/s + s$（Equation 34.48）、すなわち $401, 85, 50, 40, 104, 401$ と、その最小点 $s = \sqrt{T} = 20$ を確認せよ。さらに $T = 100$ および $T = 1600$ についても、条件 $s \le 800$ を満たす T の約数にわたって同様の実験を繰り返せ（1600 ステップを単一区間とすると backprop で Python のデフォルトの再帰上限を超える）。メモリを最小化する s は T に対してどのようにスケールし、最適なスケジュールは何回のステップ評価を必要とするか。

5. [extended] **平衡層。**●●深層平衡層 (Bai et al. 2019) は次の不動点を出力する。

$$\mathbf{z}^\star = \tanh(\mathbf{W}\mathbf{z}^\star + \mathbf{c}), \tag{34.52}$$

ここで $\mathbf{c}$ は層への入力に依存するベクトルである。損失を $\ell = \tfrac{1}{2}\|\mathbf{z}^\star - \mathbf{y}\|^2$ とし、$\mathbf{D} = \operatorname{diag}\big(1 - (z_1^\star)^2, \dots, 1 - (z_d^\star)^2\big)$ を不動点における tanh の導関数とする。

   1. 写像 $\mathbf{z} \mapsto \tanh(\mathbf{W}\mathbf{z} + \mathbf{c})$ が $\|\mathbf{W}\|_2 < 1$ のとき縮小写像となり、したがって不動点が存在して一意であること、さらにそのとき $\mathbf{I} - \mathbf{D}\mathbf{W}$ が正則になることを示せ。また、Equation 34.51 を用いて $\nabla_{\mathbf{W}} \ell$ を導出し、それが $(\mathbf{D}\mathbf{a})(\mathbf{z}^\star)^\top$（ただし随伴変数 $\mathbf{a}$ は $(\mathbf{I} - \mathbf{D}\mathbf{W})^\top \mathbf{a} = \mathbf{z}^\star - \mathbf{y}$ の解）に等しいことを示せ。
   2. $d = 20$ とし、標準正規分布に従う要素を持つ $\mathbf{G} \in \mathbb{R}^{d \times d}$ から $\mathbf{S} = \mathbf{G} + \mathbf{G}^\top$、$\mathbf{W} = \rho\, \mathbf{S} / \|\mathbf{S}\|_2$（$\rho \in \{0.5, 0.9, 0.99\}$）を作り、$\mathbf{c}$ と $\mathbf{y}$ の各要素を独立に $\mathcal{N}(0, 0.1^2)$ から生成せよ。NumPy を用い、不動点反復によって $\mathbf{z}^\star$ を求め、連立一次方程式を 1 回解くことで (a) の勾配を計算せよ。そのいくつかの要素を ℓ の中心差分と比較して検証せよ。
   3. 展開による微分は、K 回の反復を $\mathbf{z}_0 = \mathbf{0}$ から実行し、そのすべてを通して逆伝播する。1 反復あたり 1 回の VJP として、この逆方向スイープを手作業で実装せよ。K に

対して、(b) の勾配を基準とした展開勾配の誤差を、順伝播の誤差 $\|\mathbf{z}_K - \mathbf{z}^\star\|$ とともにプロットせよ。両誤差の減衰率を **DW** のスペクトル半径と比較せよ。K が大きくなるにつれて 2 つの誤差の比はどのように振る舞うか、またそれはなぜか。各勾配計算のメモリ使用量は K に対してどのようにスケールするか。

4. 連立一次方程式の求解を、反復 m ステップによる更新 $\mathbf{a} \leftarrow (\mathbf{z}^\star - \mathbf{y}) + (\mathbf{DW})^\top \mathbf{a}$（初期値 $\mathbf{a} = \mathbf{0}$）に置き換えよ。これは打ち切りノイマン級数であり、Geng et al. (2021) が**ファントム勾配**として用いているものである。この打ち切り随伴変数が、反復 m 回（初期値 $\mathbf{z}^\star$）を通した誤差逆伝播法と同じ勾配を与えることを示せ。打ち切り勾配と厳密な勾配のなす角の余弦（コサイン）を、m に対して各 ρ ごとにプロットせよ。$m = 1$ のときの勾配は連立一次方程式を解く必要がなく、Wu Fung et al. (2022) のヤコビ行列フリー誤差逆伝播法に一致する。これは依然として ℓ の降下方向になっているか。

6. [code] **1 回の順伝播による二階導関数**。ε を $\varepsilon^3 = 0$（ただし $\varepsilon^2 \neq 0$）を満たす元として添加し、$a + b\varepsilon + c\varepsilon^2$ を 3 つ組 (a, b, c) と書く。

   1. 積 $(a,b,c)(a',b',c') = (aa',\ ab' + ba',\ ac' + bb' + ca')$ を導出し、解析関数 φ に対して $\varphi(a + b\varepsilon + c\varepsilon^2) = \varphi(a) + \varphi'(a)\,b\,\varepsilon + (\varphi'(a)\,c + \tfrac{1}{2}\varphi''(a)\,b^2)\varepsilon^2$ が成り立つことを示せ。
   2. `Dual2` に `__add__`、`__mul__`、`sin`、`exp` を実装し、$f(x) = \sin(x^2) + e^x$ を $(1.3, 1, 0)$ 上で評価せよ。結果が $(f(1.3),\ f'(1.3),\ \tfrac{1}{2}f''(1.3))$ になることを示し、$f''(1.3) \approx -3.2806$ を手計算の公式 $f''(x) = 2\cos x^2 - 4x^2 \sin x^2 + e^x$ と比較して確かめよ。
   3. $\ell : \mathbb{R}^n \to \mathbb{R}$ の入力に方向 $\mathbf{v}$ をシードとして与えたとき、ε^2 成分は何を返すか（答えはスカラー値 $\tfrac{1}{2}\mathbf{v}^\top \mathbf{H}\mathbf{v}$ である）。偏極恒等式によってベクトル $\mathbf{Hv}$ を得るにはこのようなパスが何回必要か、1 回のフォワード・オーバー・リバースのパスと対比して答えよ。
   4. ε を 3×3 行列（ただし $\varepsilon^3 = \mathbf{0}$）として表し、(a) の積を行列積によって検証せよ。

34.6 積分法

これまでの章では、連続確率変数の確率がどのように計算されるかを問うことなく、それを用いてきた。正規分布をその確率密度によって記述し（2.1.3 節）、期待損失をその密度に関する平均として表し、どちらも所与のものとして扱ってきたのである。本節では、ある区間にわたって連続的に広がる量をどのように足し合わせるかを問う。

おなじみの釣鐘曲線である標準正規分布は、連続的な和になぜ注意が必要なのかをよく示している。標準正規確率変数 Z は密度 $p(z) = e^{-z^2/2}/\sqrt{2\pi}$ をもつ。$z = 0$ において曲線の高さは $p(0) \approx 0.40$ に達するが、ちょうどゼロをとる確率は $P(Z = 0) = 0$ であり、他のどの 1 点についても同様である。その一方で、区間は確かに確率をもつ。たとえば $P(-1 \le Z \le 1) \approx 0.68$ である。幾何学的には、この数値は p の下の -1 から 1 までの面積であり、$1/\sqrt{2\pi}$ は曲線の下の全体の面積を 1 にする定数である。では、$\sqrt{2\pi}$ はどこから現れ、0.68 はどのように計算するのだろうか。

こうした問いは釣鐘曲線にとどまらない。連続変数に関するあらゆる確率や期待損失は、この種の面積である。変分オートエンコーダ（variational autoencoder、VAE）や方策勾配法のようにランダムサンプリングを通じてモデルを訓練する場合には、このような面積をモデルパラメータに関して微分することも必要になる。

方針は明快である。領域を細いスライスに切り分け、それぞれの面積を足し合わせ、スライスの幅をゼロへと縮める。その極限が積分である。続いて微分積分学の基本定理（fundamental theorem of calculus）が、この無限の足し算を微分の逆演算へと置き換え、大半の積分を 1 行

で解決してくれる。ただし釣鐘曲線はこれだけでは解決せず、その正規化定数（normalizing constant）である $\sqrt{2\pi}$ を求めるには 2 次元を経由する回り道が必要になる。本節の軸となるのは、基本定理 Equation 34.55、多変数の変数変換（change of variables）の公式 Equation 34.63、そしてガウス積分（Gaussian integral）Equation 34.65 の 3 つの結果である。大学初年次の微分積分をすでに身につけている読者は、34.6.2.2 節まで流し読みしてよい。パラメータに関する積分の微分はこれらすべての道具の上に成り立っており、34.7 節で独立した節を設けて扱う。測度論とルベーグ積分（Lebesgue integral）は割愛し、関心のある読者には 34.8 節を案内する。

34.6.1 定積分

34.6.1.1 リーマン和

確率密度は単位長さあたりの確率を与えるため、幅 ϵ の短い区間における確率は、おおよそ密度と幅の積になる。より長い区間では、それを細いスライスに切り分け、各スライスで高さに幅を掛けて結果を足し合わせる。この考え方に密度特有の部分は何もなく、同じ手順が任意の関数に対して通用する。

まずは手計算で確かめられる例から始めよう。$f(x) = x^2$ を区間 $[0, 1]$ 上で考え、この区間を N 個の等しい幅 $\epsilon = 1/N$ のスライスに分割する。i 番目のスライス $[i/N, (i+1)/N]$ では、左端点における f の値 $(i/N)^2$ を高さとする長方形で面積を近似する。N 個の長方形をすべて足し合わせると、次式が得られる。

$$\sum_{i=0}^{N-1} \frac{1}{N}\left(\frac{i}{N}\right)^2 = \frac{1}{N^3}\sum_{i=0}^{N-1} i^2 = \frac{(N-1)N(2N-1)}{6N^3} = \frac{1}{3} - \frac{1}{2N} + \frac{1}{6N^2},$$

ここで途中の変形には古典的な恒等式 $\sum_{i=0}^{N-1} i^2 = (N-1)N(2N-1)/6$ を用いた。N が大きくなるにつれて長方形の和は $\frac{1}{3}$ に収束し、極限とのずれはおよそ $\frac{1}{2N}$、すなわちスライス幅 ϵ に関して 1 次の誤差となる。

補助関数 `left_sum` は、任意の関数 f に対してこの**左端点則**（left-endpoint rule）を実装したものである。すなわち、$[a, b]$ を幅 ϵ のスライスに分割し、各スライスの左端で f を評価する（Figure 34.27）。本節のすべてのグリッド和には `left_sum` を用いることとし、まずは x^2 で手早く確認してみよう。

```
def left_sum(f, a, b, eps):
    """Left-endpoint Riemann sum of f on [a, b] with slices of width eps."""
    x = a + eps * np.arange(round((b - a) / eps))   # left ends of the slices
    return float(np.sum(eps * f(x)))

for N in [10, 100, 1000]:
    print(f'N={N:<5} left sum={left_sum(lambda x: x**2, 0, 1, 1 / N):.6f}  '
          f'formula={1/3 - 1/(2*N) + 1/(6*N**2):.6f}')
```

```
N=10    left sum=0.285000  formula=0.285000
N=100   left sum=0.328350  formula=0.328350
N=1000  left sum=0.332834  formula=0.332834
```

数値計算による和は、表示されたすべての桁で閉形式の公式と一致している。

定義：この構成において、スライスの幅を等しくする必要も左端点を使う必要もない。より一般に、**分割**（partition）$P : a = x_0 < x_1 < \cdots < x_N = b$ は $[a, b]$ を部分区間へと切り分け、**代**

表点（tag）$\xi_i \in [x_{i-1}, x_i]$ は各スライスの内部から 1 つのサンプル点を選び、**分割の幅**（mesh）$\|P\| = \max_i (x_i - x_{i-1})$ は最も広いスライスの幅を測る。有界関数 f が $[a,b]$ 上で**リーマン積分可能**（Riemann integrable）であるとは、代表点の選び方によらず、分割の幅をゼロへと縮めたときにすべての代表点付きリーマン和が近づくただ 1 つの数値 I が存在することをいう。

$$\int_a^b f(x)\,dx = I = \lim_{\|P\| \to 0} \sum_{i=1}^{N} f(\xi_i)(x_i - x_{i-1}). \tag{34.53}$$

要するに、定積分とはスライスを限りなく細くしたときのスライス和の共通の極限である。先ほどの手計算により、$\int_0^1 x^2\,dx = \tfrac{1}{3}$ が確立されたことになる。$[a,b]$ 上のすべての連続関数はリーマン積分可能であり、有限個の跳躍不連続点をもつすべての有界関数も同様である (Rudin 1976)。有理数全体の指示関数のような一部の病的な有界関数はリーマン積分をもたないが、ルベーグ積分（34.8 節）はそのような関数へと積分を拡張しつつ、リーマン積分が存在するときは常にそれと一致する。

定義から直ちに 2 つの初等的な性質が導かれ、以下の証明の原動力となる。第 1 に、積分は**単調**（monotone）である。$f \le g$ が $[a,b]$ 上のいたるところで成り立つならば、f のどの代表点付きリーマン和も対応する g の代表点付きリーマン和以下となり、極限をとっても不等号が保たれるため $\int_a^b f\,dx \le \int_a^b g\,dx$ が得られる。第 2 に、積分は**部分区間に関して加法的**（additive over subintervals）である。任意の $a \le b \le c$ に対して $\int_a^c f\,dx = \int_a^b f\,dx + \int_b^c f\,dx$ が成り立つ。なぜなら、点 b を含む任意の分割は $[a,b]$ の分割と $[b,c]$ の分割へと綺麗に二分されるからである。

通し例：ガウス積分に到達するまで、本節を通して 1 つの具体例を扱う。

$$\int_0^2 \frac{x}{1+x^2}\,dx = \tfrac{1}{2}\log 5 \approx 0.80472.$$

この値がなぜ $\tfrac{1}{2}\log 5$ になるかは、評価公式（evaluation rule、Equation 34.56）を確立した時点で明らかになる。Figure 34.27 に、3 通りのスライス幅におけるこの曲線の下の左端点長方形を示す。

左端点和はどれほどの速さで $\tfrac{1}{2}\log 5$ に収束するだろうか。x^2 の $[0,1]$ 上での誤差は $\frac{1}{2N} = \frac{\epsilon}{2}$ であった。1 次収束が一般的であるならば、スライスを 10 倍細くすると誤差も 10 分の 1 に縮まるはずである。誤差を $0.2\,\epsilon$ と並べて出力し、確かめてみよう。

```
f = lambda x: x / (1 + x**2)
truth = math.log(5) / 2
for eps in [1e-1, 1e-2, 1e-3, 1e-4]:
    s = left_sum(f, 0, 2, eps)
    print(f'eps={eps:.0e}  sum={s:.6f}  truth-sum={truth - s:.2e}  '
          f'0.2*eps={0.2 * eps:.0e}')
print(f'truth = (1/2) log 5 = {truth:.6f}')
```

```
eps=1e-01  sum=0.783785  truth-sum=2.09e-02  0.2*eps=2e-02
eps=1e-02  sum=0.802710  truth-sum=2.01e-03  0.2*eps=2e-03
eps=1e-03  sum=0.804519  truth-sum=2.00e-04  0.2*eps=2e-04
eps=1e-04  sum=0.804699  truth-sum=2.00e-05  0.2*eps=2e-05
truth = (1/2) log 5 = 0.804719
```

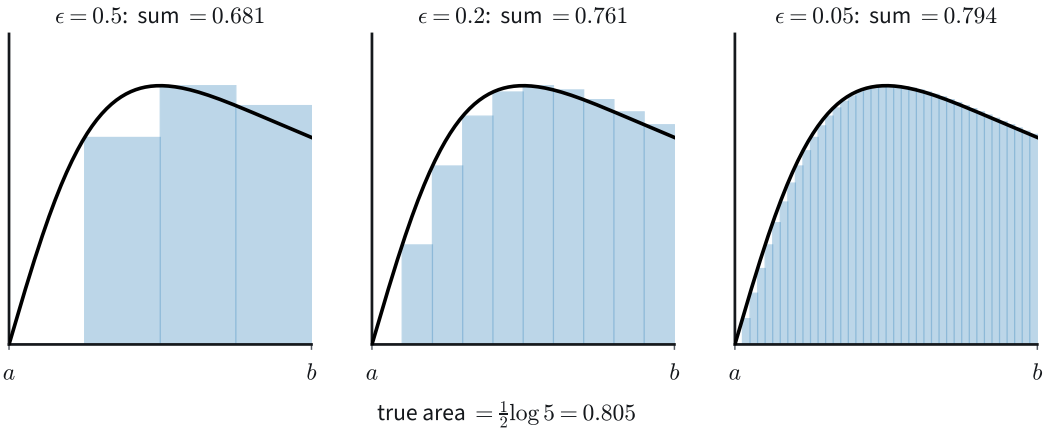


Figure 34.27: $f(x) = x/(1+x^2)$ の区間 $[0, 2]$ 上におけるスライス幅 $\epsilon = 0.5$、0.2、0.05 の左端点長方形。各スライスは左端点における f の高さをとり、長方形の和は面積 $\frac{1}{2}\log 5$ に近づく（$\epsilon \to 0$ のとき）。

ϵ が 10 分の 1 に縮まるたびに誤差も 10 分の 1 に縮まり、どの和も真の値を下回っている。定数 0.2 はちょうど $f(2) - f(0) = 0.4$ の半分になっている。これは偶然ではなく、基本定理の準備が整い次第、その背後にある公式を導出する。

34.6.1.2 符号付き面積

横軸より下にある長方形は負の高さ $f(\xi_i) < 0$ をもち、負の符号を伴ってリーマン和に加わる。したがって積分は**符号付き**（signed）の面積を測ることになり、$f < 0$ となる領域は $\int_0^1 (-1)\,dx = -1$ のように負の寄与をする。また規約により、右から左へ積分すると結果の符号も反転し、$\int_b^a f\,dx = -\int_a^b f\,dx$ となる。この規約によって、a、b、c の順序によらず加法性が成り立つことが保証される。Figure 34.28 に、1 本の曲線上でこれら両方の規約を示す。

34.6.2 微分積分学の基本定理

34.6.2.1 定理

スライスを足し合わせることで $\frac{1}{3}$ という x^2 の積分値が得られたのは、たまたま $\sum i^2$ が閉形式をもっていたからにすぎない。通し例では左端点和が $\sum_i \epsilon \cdot i\epsilon/(1 + i^2\epsilon^2)$ となり、これを簡約できる代数的な恒等式は存在しない。スライスを一切足し合わせずに積分を計算する方法が必要である。

突破口は、積分の上端を変数として扱うことから開ける。基点 a を固定し、**累積面積関数**（area-so-far function）を次式で定義しよう。

$$F(x) = \int_a^x f(y)\,dy, \tag{34.54}$$

この関数は、a から x まで累積された符号付き面積を記録する。このたった 1 つの関数に f のあらゆる定積分が符号化されている。加法性により、任意の区間 $[x_1, x_2]$ 上の面積は、単に 2 つの累積面積の差として表されるからである。

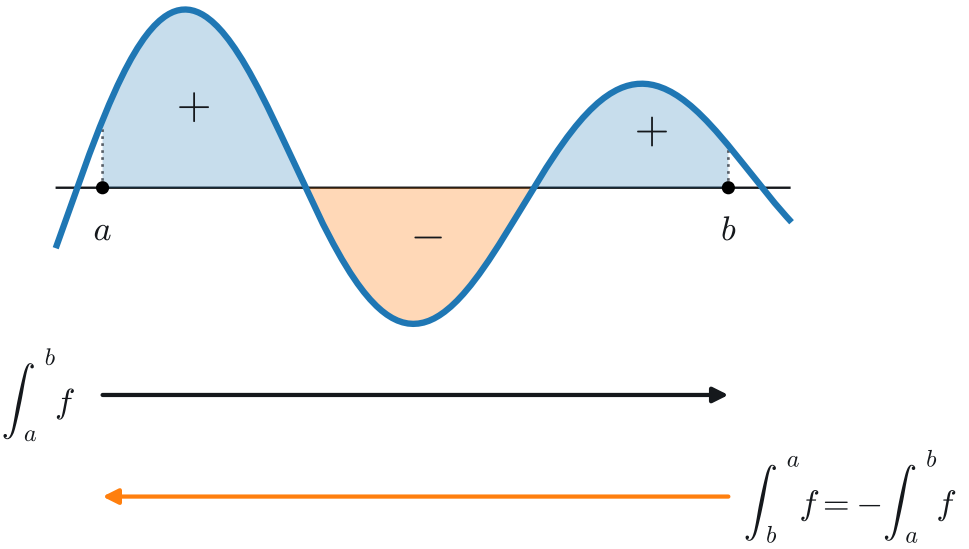


Figure 34.28: 符号付き面積。軸より上の膨らみは正、軸より下の膨らみは負として数えられるため、$\int_a^b f$ は網掛け領域を符号付きで合計する。積分区間の上下端を入れ替えると、同じ領域を右から左へたどることになり合計の符号が反転する。

$$\int_{x_1}^{x_2} f(x)\,dx = F(x_2) - F(x_1),$$

この関係は Figure 34.29 に示すとおりである。したがって F さえ見つかれば、問題全体が解決する。

F は、x をほんの少し右へ動かしたときにどう変化するだろうか。Figure 34.30 には、x までの累積面積と並んで、幅 ϵ の細い帯（$x+\epsilon$ まで動かすことで加わる部分）が示されている。この帯は高さ $f(x)$ の長方形にきわめて近いため、$F(x+\epsilon) - F(x) \approx \epsilon f(x)$ となる。両辺を ϵ で割ると、$F'(x) = f(x)$ であることが示唆される。

定理（微分積分学の基本定理）： **f を $[a,b]$ 上でリーマン積分可能な関数とし、$F(x) = \int_a^x f(y)\,dy$ とおき、さらに f が点 $x \in (a,b)$ で連続であるとする。このとき、F は x で微分可能であり、次式が成り立つ。**

$$\frac{dF}{dx}(x) = f(x). \tag{34.55}$$

言葉で言えば、面積が累積していく変化率は、その時点における曲線の高さに等しい。

証明： x を $\epsilon > 0$ だけわずかに動かし、新たに加わる面積に着目する。Equation 34.54 と加法性により、その変化量は 1 本の細い短冊となる。

$$F(x+\epsilon) - F(x) = \int_x^{x+\epsilon} f(y)\,dy.$$

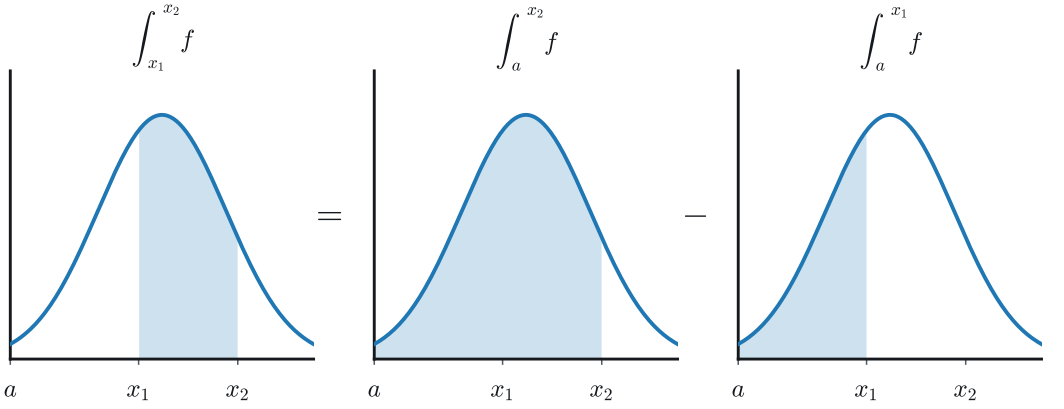


Figure 34.29: f の下の x_1 から x_2 までの面積は、図の左端にある基点 a から累積した 2 つの面積の差として表される（$\int_{x_1}^{x_2} f\,dx = \int_{a}^{x_2} f\,dx - \int_{a}^{x_1} f\,dx$）。

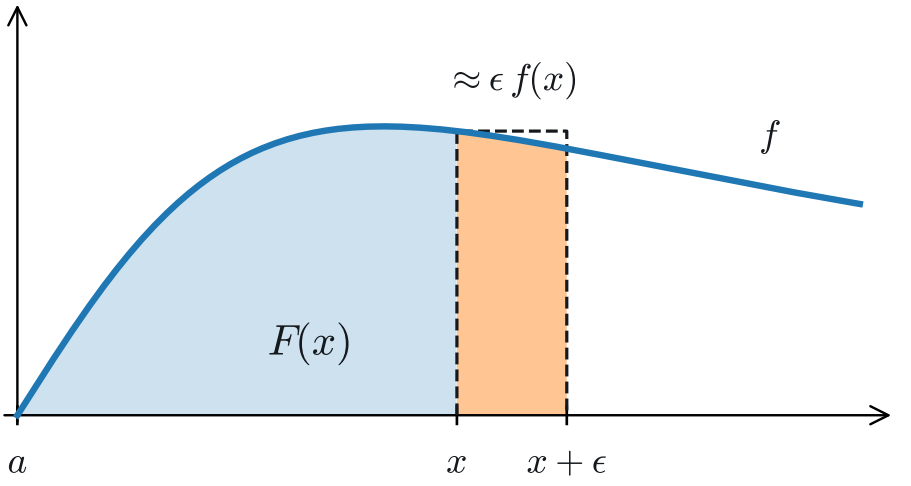


Figure 34.30: 累積面積 $F(x)$（$f(y) = y/(1+y^2)$ の下の $a = 0$ から $x = 1.2$ までの淡色部分）と、$x + \epsilon$ まで動かすことで加わる帯（濃色部分）。この帯は高さ $f(x)$ の破線の長方形に近いため、$F(x+\epsilon) - F(x) \approx \epsilon f(x)$ となる。

$m_\epsilon = \inf_{[x,x+\epsilon]} f$ と $M_\epsilon = \sup_{[x,x+\epsilon]} f$ を、この短冊上における f の最も厳しい定数の下界および上界とする。可積分関数は有界であるため、いずれも有限である。$m_\epsilon \le f \le M_\epsilon$ が $[x, x+\epsilon]$ 全体で成り立つため、積分の単調性により、短冊の面積は幅 ϵ の 2 つの長方形の間にはさまれる。

$$\epsilon\, m_\epsilon \;\le\; F(x+\epsilon) - F(x) \;\le\; \epsilon\, M_\epsilon.$$

ここで全体を ϵ で割る。f は x で連続であるため、任意の $\delta > 0$ に対し、区間 $[x, x+\epsilon]$ 全体はやがて $|f(y) - f(x)| < \delta$ となる領域の内部に収まる。したがって m_ϵ と M_ϵ はともに $f(x)$ へと収束し（$\epsilon \to 0$）、はさみうちの原理により $\lim_{\epsilon\to 0} \frac{F(x+\epsilon)-F(x)}{\epsilon} = f(x)$ が導かれる。$\epsilon < 0$ のときは短冊が x の左側に位置するが、差分商における 2 箇所の符号反転が打ち消し合うため同じ評価が成り立つ。この極限が $F'(x)$ である。■

この証明が、定義から読み取った単調性と加法性の 2 つの性質しか使っていないことに注目しよう。この定理により、面積を求める問題は**原始関数**（antiderivative）、すなわち関数 G であって $G' = f$ を満たすものを見つける問題へと帰着する。そのような関数は、34.1.2 節の導関数の表からそのまま読み取れることが多い。どの原始関数を用いても、次の**評価公式**によって定積分が得られる。**f が $[a,b]$ 上で連続であり、$G' = f$ が $[a,b]$ 全体で成り立つならば、次式が成り立つ。**

$$\int_a^b f(x)\,dx = G(b) - G(a). \tag{34.56}$$

なぜ F だけでなく、どの原始関数でもよいのだろうか。それは、F と G が同じ導関数 f を (a,b) 上で共有するため、その差 $G - F$ の導関数はゼロとなり、平均値の定理（34.1.3 節）によって定数でなければならないからである。その加法定数は $G(b) - G(a) = F(b) - F(a) = \int_a^b f\,dx$ において打ち消し合う。

「$[a,b]$ 全体で」という要件は不可欠である。$(-1/x)' = x^{-2}$ であるからといって盲目的に公式を適用すると、$\int_{-1}^{1} x^{-2}\,dx = [-1/x]_{-1}^{1} = -2$ となり、真に正の被積分関数に対して負の答えが出てしまう。ここで公式が適用できないのは、$G' = f$ が $x = 0$ で破綻しており、そこではどちらの関数も定義されていないからである。実際、この積分は発散する（34.6.2.2 節）。

逆向きに読めば、あらゆる微分公式が積分公式になる。$(y^3)' = 3y^2$ からはスライスを 1 つも切ることなく $\int_0^1 y^2\,dy = \frac{1}{3}$ が得られ、$(e^t)' = e^t$ からは $\int_0^x e^t\,dt = e^x - 1$ が得られる。通し例については、$G(x) = \frac{1}{2}\log(1+x^2)$ を試してみよう。連鎖律により $G'(x) = \frac{1}{2}\cdot\frac{2x}{1+x^2} = \frac{x}{1+x^2}$ となるため、次式が得られる。

$$\int_0^2 \frac{x}{1+x^2}\,dx = G(2) - G(0) = \tfrac{1}{2}\log 5 - \tfrac{1}{2}\log 1 = \tfrac{1}{2}\log 5,$$

これにより、先ほどのリーマン和が近づいていた極限が確かめられた。

左端点則の誤差：評価公式からは、左端点和の誤差に現れた定数 0.2 がどこから来たのかも説明できる。x_i から始まるスライス上で 1 次展開を行うと、$f(x) \approx f(x_i) + f'(x_i)(x - x_i)$ となる。$[x_i, x_i+\epsilon]$ 上で積分すると、このスライスの真の面積はその長方形 $\epsilon f(x_i)$ をおよそ $\frac{\epsilon^2}{2} f'(x_i)$ だけ上回ることがわかる。これら三角形の過剰分をすべてのスライスにわたって足し合わせると、$\frac{\epsilon}{2}$ と f' の左端点和の積になり、これは $\frac{\epsilon}{2}\int_a^b f'\,dx = \frac{\epsilon}{2}(f(b) - f(a))$ を近似する。したがって ϵ の 1 次の範囲で次式が成り立つ。

$$\text{left sum} - \int_a^b f(x)\,dx \approx \frac{\epsilon}{2}\big(f(a) - f(b)\big).$$

x^2 の $[0,1]$ 上の積分では、この式から $-\frac{\epsilon}{2} = -\frac{1}{2N}$ が得られ、手計算で求めた主要誤差項と一致する。通し例では $f(0) = 0$ と $f(2) = 0.4$ より $-0.2\,\epsilon$ となる。この評価は純粋な微分積分というより数値解析に属するが、本節の残り 3 つの実験に現れる残差を説明してくれるほか、36.5.1.7 節でランダムサンプリングを評価する際の基準線にもなる。

実は基本定理は $\int_0^2 f = G(2) - G(0)$ 以上のことを教えてくれる。累積面積関数 $F(x)$ は $G(x) - G(0)$ とすべての x において等しくなければならないのである。F を $[0,2]$ にわたる累積左端点和として計算し、すべてのグリッド点で $G(x) - G(0)$ と比較してこれを確かめてみよう。$[0,x]$ 上の左端点和はバイアス $\frac{\epsilon}{2}\big(f(0) - f(x)\big)$ を伴うため、両者は機械精度までは一致しない。このバイアスが最大になるのは f がピークをとる場所、すなわち $x = 1$（このとき $f(1) = \frac{1}{2}$）であるため、区間全体での最大誤差は $\frac{\epsilon}{4}$ になるはずである。

```
G = lambda x: np.log(1 + x**2) / 2            # antiderivative of f
for eps in [1e-2, 1e-3, 1e-4]:
    x = eps * np.arange(round(2 / eps))       # left ends of the slices of [0, 2]
    F = np.cumsum(eps * f(x))                 # F[k]: left sum on [0, x_k + eps]
    gap = np.abs(F - (G(x + eps) - G(0))).max()
    print(f'eps={eps:.0e}  max over x of |F(x) - (G(x) - G(0))| = {gap:.2e}')
```

```
eps=1e-02  max over x of |F(x) - (G(x) - G(0))| = 2.51e-03
eps=1e-03  max over x of |F(x) - (G(x) - G(0))| = 2.50e-04
eps=1e-04  max over x of |F(x) - (G(x) - G(0))| = 2.50e-05
```

どのスライス幅においても、最大誤差は予測どおり $\frac{\epsilon}{4}$ となっている。累積和は右端点 $x = 2$ だけでなく、区間全体にわたって原始関数を追跡している。

34.6.2.2 広義積分

正規分布や指数分布をはじめ多くの確率密度は非有界な定義域上にある一方、Equation 34.53 は有限区間 $[a,b]$ 上でのみ積分を定義している。そこで、積分を構成したときと同じく極限をとることで定義を拡張する。$[a,\infty)$ 上の**広義積分**（improper integral）は次式で定められる。

$$\int_a^{\infty} f(x)\,dx = \lim_{b\to\infty} \int_a^b f(x)\,dx, \tag{34.57}$$

これは b まで累積された面積の、大きな b における極限である。この極限が存在して有限であるとき、積分は**収束する**といい、そうでないときは**発散する**という。実数直線全体にわたって積分するには、$\int_{-\infty}^{\infty} f\,dx$ を任意の都合のよい点で 2 つの片側広義積分に分割し、その**両方**が独立に収束しなければならない。対称な極限だけでは不十分である。$\int_{-b}^{b} x\,dx = 0$ はすべての b に対して成り立つが、$\int_0^{\infty} x\,dx = \infty$ であるため、$\int_{-\infty}^{\infty} x\,dx$ は存在しない。

べき乗則 x^{-s}（指数 $s > 0$）は、裾が有限の面積を囲めるほど十分に速く減衰するかどうかを判定する標準的な基準である。$[1,b]$ 上で評価公式を適用し、$b \to \infty$ の極限をとると次式が得られる。

$$\int_1^b x^{-s}\,dx = \frac{b^{1-s}-1}{1-s} \xrightarrow{b\to\infty} \begin{cases} \dfrac{1}{s-1}, & s>1 \text{ (converges)},\\ \infty, & s\le 1 \text{ (diverges)}. \end{cases}$$

（$s=1$ では上の式は $0/0$ の形になるが、定積分は $\log b$ であり、これは $s\to 1$ としたときの公式の極限に一致し、$b\to\infty$ で発散する。）したがって $\int_1^\infty x^{-2}\,dx = 1$ である一方、$\int_1^\infty x^{-1}\,dx = \infty$ となり、面積が有限になるか無限になるかの境界線はちょうど $s=1$ にある。

べき乗則は典型的な**裾の重い**（heavy-tailed）分布である（36.1 節では、重い裾が平均にどう影響するかも調べる）。裾が $p(x)\propto x^{-s}$ に従う密度では、正規化定数が有限になるのは $s>1$ のときに限られ、また同じ判定基準を $x\cdot x^{-s}=x^{1-s}$ に適用して調べられる平均が有限になるのは $s>2$ のときに限られる。

端点で発散する被積分関数も、同じ極限の手法で扱える。すなわち $\int_0^1 f\,dx = \lim_{a\to 0^+}\int_a^1 f\,dx$ とする。べき乗則 x^{-s} に対しては、条件が逆転する。

$$\int_0^1 x^{-s}\,dx = \lim_{a\to 0^+}\frac{1-a^{1-s}}{1-s} = \frac{1}{1-s} \quad \text{for } s<1,$$

一方、$s\ge 1$ では積分は発散する。たとえば $s=\tfrac12$ では、$x^{-1/2}$ は 0 で無限大へと跳ね上がるが、その下の面積はわずか 2 である。無限遠の近くでは、べき乗則が有限の面積をもつには $1/x$ より速く減衰しなければならず、ゼロの近くでは $1/x$ より緩やかに増大しなければならない。

広義積分を数値的に近似するときには、2 種類の異なる誤差が入り込む。定義域を有限の b で打ち切ることによる誤差と、有限の幅 ϵ のスライスを足し合わせることによる誤差である。$\int_0^\infty e^{-x}\,dx = 1$ については、この 2 つを切り分けられる。以下では、厳密な部分積分値 $\int_0^b e^{-x}\,dx = 1-e^{-b}$（原始関数 $-e^{-x}$ から得られる）を、$\epsilon = 10^{-3}$ とした左端点和と並べて出力する。

```
for b in [1, 2, 5, 10, 20]:
    left = left_sum(lambda x: np.exp(-x), 0, b, 1e-3)
    print(f'b={b:<3}  left sum={left:.6f}  exact 1-e^-b={1 - math.exp(-b):.6f}')
```

```
b=1    left sum=0.632437  exact 1-e^-b=0.632121
b=2    left sum=0.865097  exact 1-e^-b=0.864665
b=5    left sum=0.993759  exact 1-e^-b=0.993262
b=10   left sum=1.000455  exact 1-e^-b=0.999955
b=20   left sum=1.000500  exact 1-e^-b=1.000000
```

厳密な部分積分値は急速に 1 へと収束する。b で打ち切ることによる**打ち切り誤差**（truncation error）は e^{-b} であり、$b=20$ の時点で小数第 6 位より小さくなっている。しかし左端点和は 1.0005 付近に落ち着き、1 にはならない。この残差が**離散化誤差**（discretization error）、すなわち先ほどの左端点則のバイアス $\frac{\epsilon}{2}\big(f(0)-f(b)\big)\to\frac{\epsilon}{2}=0.0005$ である。主要項のオーダーでは、2 つの誤差は単純に足し合わされる。b を大きくすれば打ち切り誤差が消え、ϵ を小さくすれば離散化誤差が消える。

34.6.2.3 部分積分

あらゆる微分法則は、評価公式を通して読むことで対応する積分法則を生み出す。積の微分法則は、部分積分（integration by parts）になる。

連続微分可能な関数 u と v に対し、$[a,b]$ 上で次式が成り立つ。

$$\int_a^b u(x)\,v'(x)\,dx = \Big[\,u(x)\,v(x)\,\Big]_a^b - \int_a^b u'(x)\,v(x)\,dx, \tag{34.58}$$

ここで $[\,h(x)\,]_a^b$ は $h(b)-h(a)$ の略記である。証明は 1 行で済む。積の微分法則により uv は $u'v+uv'$ の原始関数であるから、Equation 34.56 より $\int_a^b (u'v+uv')\,dx = [\,u(x)v(x)\,]_a^b$ となり、両辺から $\int_a^b u'v\,dx$ を引けば Equation 34.58 が得られる。

この公式がしているのは、境界項が生じる代わりに、一方の因子にかかっていた微分をもう一方の因子へと移すことである。新しい被積分関数 $u'v$ が元の uv' より単純になるときはいつでも、この変形が威力を発揮する。$u(x)=x$、$v'(x)=e^{-x}$（したがって $u'=1$、$v=-e^{-x}$）と選び、Equation 34.58 を $[0,b]$ 上で適用して $b\to\infty$ とすると、次式が得られる。

$$\int_0^\infty x\,e^{-x}\,dx = \Big[-x\,e^{-x}\Big]_0^\infty + \int_0^\infty e^{-x}\,dx = 0+1 = 1.$$

両端点において $xe^{-x}\to 0$ となるため境界項は消え、先ほど計算した広義積分だけが残る。この積分は指数分布の平均である（36.3 節）。

期待値は積の積分であるため、扱いが困難な因子から扱いやすい因子へと微分を移す手法は、確率論や機械学習で繰り返し登場する定石である。釣鐘曲線は、この技法を最も鮮やかに示してくれる。正規化されていない釣鐘型の重み $w(z)=e^{-z^2/2}$ を考えると、これは $w'(z)=-z\,w(z)$ を満たす。g を連続微分可能な関数で $g(z)w(z)\to 0$（$z\to\pm\infty$）を満たすものとし、以下の 2 つの積分が収束すると仮定する。$[-a,b]$ 上で部分積分を行い、続いて $a\to\infty$ と $b\to\infty$ の極限を別々にとると、次式が得られる。

$$\int_{-\infty}^{\infty} g'(z)\,w(z)\,dz = \Big[\,g(z)\,w(z)\,\Big]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} g(z)\,w'(z)\,dz = \int_{-\infty}^{\infty} z\,g(z)\,w(z)\,dz. \tag{34.59}$$

釣鐘型の重みのもとでは、z を掛けることは g を微分することと同じ効果をもつ。任意の多項式 g に対して上記の両方の仮定が成り立つ。$g(z)=z$ とおくと $\int z^2 w\,dz = \int w\,dz$ となり、釣鐘型の重みのもとでの z^2 の平均は 1 となる。これは正規化定数 $\int w\,dz$ がいくらになるかによらない（それが $\sqrt{2\pi}$ になることは Equation 34.66 で示す）。$g(z)=z^3$ とおくと $\int z^4 w\,dz = 3\int z^2 w\,dz = 3\int w\,dz$ となるため、z^4 の平均は 3 である。w を標準正規密度 $p=w/\sqrt{2\pi}$ へと正規化すれば、この式は $\mathbb{E}[g'(Z)]=\mathbb{E}[Z\,g(Z)]$ となり、標準正規分布に対するスタインの恒等式（Stein's identity）として知られる。未知の因子から既知の因子へと導関数を移す操作は、スコアマッチングの背後にある重要なステップでもある（38.6 節、(Hyvärinen 2005)）。$\mathbb{R}^d$ において対応する規則は 1 次元の導関数の代わりに発散を移すものであり、発散、ラプラシアン、およびその多変数の恒等式は、それらが最初に必要となる 38.5.2.1 節で導入する。

34.6.2.4 置換積分

連鎖律を評価公式を通して逆向きに読むと、置換積分（substitution）の公式が得られる。u が $[a,b]$ 上で連続微分可能であり、f が $u([a,b])$ を含む区間上で連続で原始関数 G をもつとする。連鎖律により $(G\circ u)'(x) = f(u(x))\,u'(x)$ であるから、次式が成り立つ。

$$\int_a^b f(u(x))\,u'(x)\,dx = \int_a^b (G\circ u)'(x)\,dx = G(u(b)) - G(u(a)) = \int_{u(a)}^{u(b)} f(y)\,dy.$$

これを右から左へ読んだものが、**変数変換の公式**（change-of-variables formula）である。

$$\int_{u(a)}^{u(b)} f(y)\,dy = \int_a^b f(u(x))\,\frac{du}{dx}\,dx. \tag{34.60}$$

言葉で言えば、y に関する積分は x に関する積分へと書き換えることができ、その際には被積分関数に局所伸縮率 du/dx を掛ければよい。なお、u は単調である必要さえない。

1 本の細いスライスを考えれば、この伸縮率の意味は一目瞭然である。$[a,b]$ を幅 ϵ のスライスに分割する。写像 u は、x にあるスライスを幅がおよそ $\epsilon\,u'(x)$ のスライス（位置 $y = u(x)$）へと移し、そこでの曲線の高さは $f(y) = f(u(x))$ である。したがってその面積は $f(u(x))\,u'(x)\,\epsilon$ となり、これこそが x の積分で足し合わせる量にほかならない（Figure 34.31）。

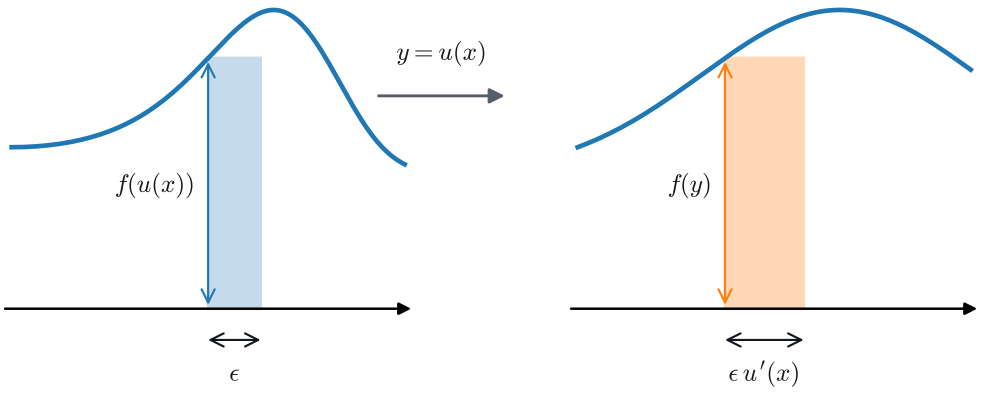


Figure 34.31: 置換 $y = u(x)$ のもとでの細い短冊。左は x 側（公式の右辺）で、幅 ϵ で位置 x にあり高さ $f(u(x))$ をもつ短冊である。右は y 側で、その像は幅 $\epsilon\,u'(x)$ で同じ高さをもつ。両者の面積を一致させるには因子 du/dx が必要になる。

実際の計算では、被積分関数の中に $f(u(x))\,u'(x)$ の形を見抜くことで置換積分を行う。$\int_0^1 x\,e^{-x^2}\,dx$ において $u(x) = x^2$ とおけば $u'(x) = 2x$ となり、被積分関数は $\frac{1}{2}e^{-u(x)}u'(x)$ となって次式が得られる。

$$\int_0^1 x\,e^{-x^2}\,dx = \frac{1}{2}\int_0^1 e^{-y}\,dy = \frac{1-e^{-1}}{2} \approx 0.316.$$

ここでは因子 x が、置換に必要な $u'(x)$ を補ってくれた。その x を取り除くと $\int e^{-x^2}\,dx$ が残るが、これは初等的な原始関数をもたず、釣鐘曲線の正規化定数 $1/\sqrt{2\pi}$ の背後にあるまさにその積分である。これを Equation 34.65 で計算するために、以下では 2 次元へ移って極座標に変換することで不足している因子を手に入れる。

34.6.3 重積分

ここからは、(Folland 1999) から標準的な 2 つの定理、すなわちフビニの定理と多変数の変数変換の公式を証明なしで引用する。

34.6.3.1 二重積分とフビニの定理

$\int e^{-x^2}\,dx$ への道筋は 2 次元を経由するため、まずは積分を区間から平面上の領域へと拡張する。2 次元の釣鐘曲面 $f(x,y) = e^{-x^2-y^2}$ を正方形 $[-2,2]^2$ 上で考えると、これは底面の上に体積を囲んでいる。面積を近似したときとまったく同じように、この体積を近似しよう。すなわち、底面を $\epsilon \times \epsilon$ の正方形で敷き詰め、各正方形の上に高さ f の角柱を立て、角柱の体積を足し合わせる（Figure 34.32）。関数 f が長方形 $R = [a,b] \times [c,d]$ 上で与えられたとき、$\epsilon \to 0$ としたときのこれらの直方体和の極限が**二重積分**（double integral）$\iint_R f(x,y)\,dx\,dy$ であり、曲面の下の符号付き体積を表す。

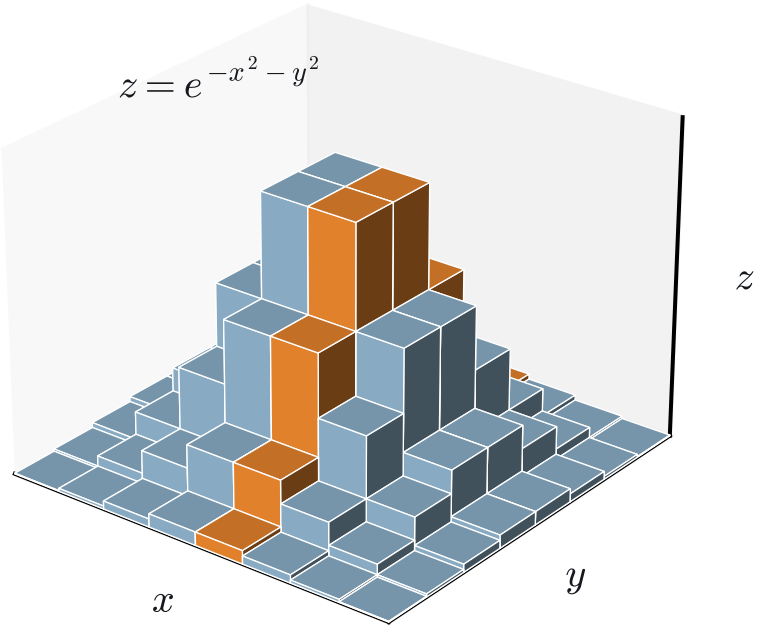


Figure 34.32: 釣鐘曲面 $z = e^{-x^2-y^2}$（正方形 $[-2,2]^2$ 上）。底面を $\epsilon \times \epsilon$ の正方形（$\epsilon = 0.5$）で敷き詰め、それぞれに高さ f の直方体を置くと曲面の下の体積が近似される。固定した x における強調表示された直方体の列は、フビニの定理における 1 つの内側の和に対応する。

1 次元の道具を使って、このような体積をどう計算できるだろうか。有限和であれば、どのような順序で合計しても結果は変わらない。グリッドの正方形を i（x 方向）と j（y 方向）で添字付けすれば、直方体を列ごとに足し合わせることができる（Figure 34.33）。

$$\sum_{i,j} \epsilon^2 f(a+\epsilon i, c+\epsilon j) = \sum_{i} \epsilon \Big(\sum_{j} \epsilon\, f(a+\epsilon i, c+\epsilon j) \Big).$$

内側の和は $\int_c^d f(a+\epsilon i, y)\,dy$、すなわち $x = a+\epsilon i$ における曲面の断面積（Figure 34.32 の強調された列）の左端点近似であり、外側の和はそのスライス断面積を x にわたって積分する。

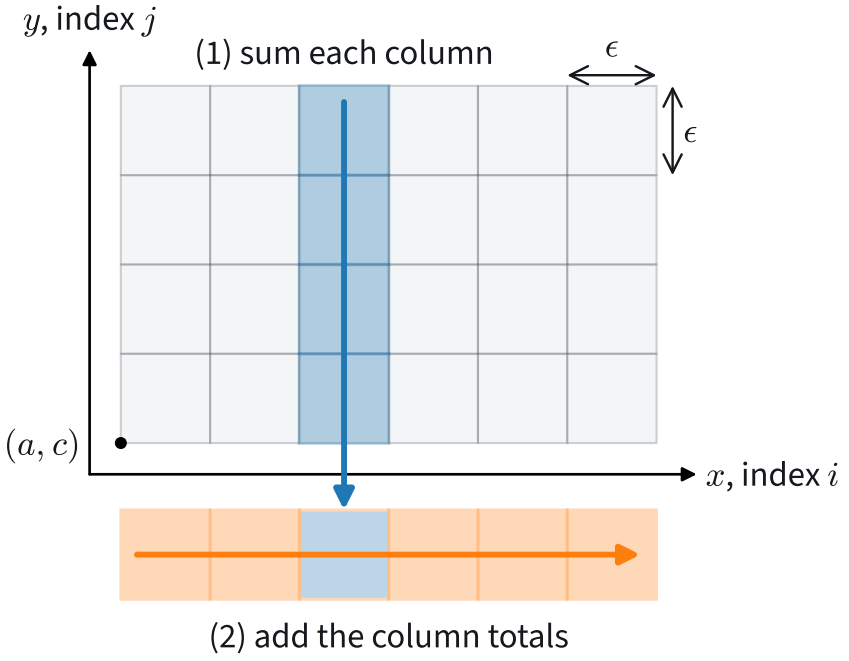


Figure 34.33: $\epsilon \times \epsilon$ のセルからなる角 (a, c) のグリッド（i が x 方向、j が y 方向の添字）の合計。(1) 固定した i において各列を j にわたって足し合わせる。(2) 列ごとの合計を左から右へ足し合わせる。有限和の順序を入れ替えても何も変わらないというこの離散的な事実が、累次積分の根底にある。

有限のグリッド和の順序を入れ替えるのは初等的であるが、$\epsilon \to 0$ の極限をとっても等号が保たれることを保証するには、1 つの定理と条件が必要になる。**フビニの定理**によれば、f が R 上で**絶対可積分**（absolutely integrable）、すなわち $\iint_R |f(x,y)|\,dx\,dy < \infty$ であるならば、二重積分はいずれの順序の累次積分（iterated integral）にも等しい。

$$\iint_R f(x,y)\,dx\,dy = \int_c^d \left(\int_a^b f(x,y)\,dx \right) dy = \int_a^b \left(\int_c^d f(x,y)\,dy \right) dx. \tag{34.61}$$

幾何学的に言えば、体積はどちらの方向にスライスしても計算できる。$f \geq 0$ のときは可積分性

の確認なしに同じ等式が成り立ち、3 辺すべてが $+\infty$ となる場合も含まれる（**トネリの定理**）。したがって確率密度や二乗損失などの非負の被積分関数では確認が不要であるが、符号が変化する被積分関数に対しては、まず $\iint |f| < \infty$ を確かめなければならない。同時密度から変数を周辺化して消去するときや、期待値と別の積分の順序を交換するときには、常にフビニの定理が働いている。

最もよく現れる特別な場合は、積の形をした被積分関数 $f(x,y) = g(x)h(y)$ である。$g(x)$ は y に関して定数であるため、内側の積分の外へそのままくくり出せて $\int_c^d g(x)h(y)\,dy = g(x)\int_c^d h(y)\,dy$ となり、次式が得られる。

$$\iint_R g(x)\,h(y)\,dx\,dy = \int_a^b g(x)\,dx \int_c^d h(y)\,dy. \tag{34.62}$$

積の被積分関数は因子ごとに積分できる。決定的に重要なのは、2 次元の釣鐘曲面が $e^{-x^2-y^2} = e^{-x^2}e^{-y^2}$ とこの形に因数分解されることであり、以下で行うガウス積分の計算はこの事実に基づいている。

絶対可積分性の仮定は単なる形式的な条件ではない。$f(x,y) = (x^2-y^2)/(x^2+y^2)^2$ を $[0,1]^2$ 上で考えよう。簡単に微分して確かめられるように $\partial_y\big(y/(x^2+y^2)\big) = f(x,y)$ であるから、評価公式より $\int_0^1 f(x,y)\,dy = \frac{1}{1+x^2}$（$x>0$）となり、さらに x で積分すると $\int_0^1 \frac{dx}{1+x^2} = \arctan 1 = \frac{\pi}{4}$ が得られる。ところが $f(x,y) = -f(y,x)$ であるため、逆の順序で積分すると $-\frac{\pi}{4}$ になってしまう。2 つの順序が食い違うのは $\iint |f| = \infty$ だからである。原点の近くで $|f|$ は距離の逆二乗のオーダーで増大し、その全体の体積は発散する。

領域 D が長方形でない場合、$\int_D f(\mathbf{x})\,d\mathbf{x}$ は**ゼロ拡張**（zero-extension）によって定義する。すなわち、f を 0 と D の外側で定めておき、D を含む任意の長方形上で積分する。

34.6.3.2 多変数の変数変換

1 次元の伸縮率 du/dx は高次元へ自然に一般化され、それがどのような形をとるべきかは線形変換を見るだけですでにわかる。線形変換はあらゆる面積や体積をその行列式の絶対値倍に拡大・縮小し（33.2 節）、行列式の符号は単に向きを追跡するにすぎない。たとえば写像 $\phi(u,v) = (au, bv)$（$a,b>0$）は行列式 ab をもち、単位円板（面積 π）を楕円 $x^2/a^2 + y^2/b^2 \le 1$ へと引き伸ばすため、その面積は πab となる。

滑らかな非線形写像は局所的には線形に見え、そのヤコビ行列が線形近似の役割を果たす。体積の拡大率が点ごとに変化するだけである。$\Omega \subseteq \mathbb{R}^n$ を開集合とし、$\phi : \Omega \to \mathbb{R}^n$ をその像の上への C^1 **級微分同相写像**（diffeomorphism）とする。すなわち、ϕ は連続微分可能かつ単射であり、そのヤコビ行列 $\mathbf{J}_\phi$（34.4 節で扱った偏微分の行列）は $\det \mathbf{J}_\phi(\mathbf{x}) \neq 0$ を Ω 上のいたるところで満たす。単射性は写像が空間を折り返して重ね合わせるのを防ぎ、非ゼロの行列式は領域を低次元へ押し潰すのを防ぐ。これらの条件のもとで、任意の絶対可積分関数 f に対して次式が成り立つ。

$$\int_{\phi(\Omega)} f(\mathbf{y})\,d\mathbf{y} = \int_{\Omega} f(\phi(\mathbf{x}))\,\big|\det \mathbf{J}_\phi(\mathbf{x})\big|\,d\mathbf{x}. \tag{34.63}$$

体積 $d\mathbf{x}$ をもつ点 $\mathbf{x}$ における微小立方体は、体積 $\big|\det \mathbf{J}_\phi(\mathbf{x})\big|\,d\mathbf{x}$ をもつ点 $\mathbf{y} = \phi(\mathbf{x})$ における微小平行多面体へと写され、ヤコビ行列式がその局所的な歪みに合わせて被積分関数を補正する。Figure 34.34 に 1 次元と 2 次元の図を並べて示す。

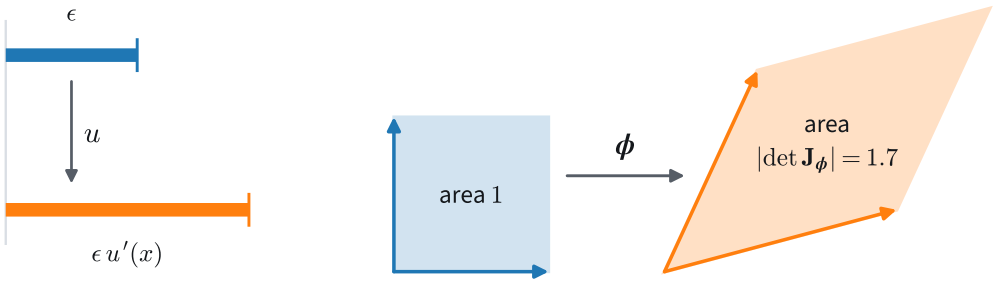


Figure 34.34: 左：1 次元の置換では、長さ ϵ の線分が $u'(x)$ 倍に伸縮される。右：2 次元では、線形変換 ϕ が単位正方形を平行四辺形へと写し、その面積が体積拡大率 $\left|\det \mathbf{J}_\phi\right|$ となる。

なぜ Equation 34.63 には $\left|\det \mathbf{J}_\phi\right|$ のように絶対値がつき、単なる $\det \mathbf{J}_\phi$ ではないのだろうか。左辺が順序付けられた端点の間ではなく、向きをもたない集合上で積分しているからである。1 次元では積分区間の上下端の順序が符号を担っている。もし u が減少関数であれば、Equation 34.60 の積分区間は逆向きになる一方で $u' < 0$ となるため、2 つの負号が打ち消し合い、像の区間上で $|u'|$ を掛けて積分するのと同値になる。$\mathbb{R}^n$ 内の領域には自然な順序が存在しないため、この符号なしの形式だけが一般化されるのである。

f が確率密度であるとき、Equation 34.63 は滑らかな可逆写像のもとで密度がどう変換されるかを表しており、正規化フロー（normalizing flow）が出力の密度を計算する仕組みそのものである（36.2 節）。

34.6.3.3 ガウス積分

釣鐘曲線の正規化定数 $1/\sqrt{2\pi}$ は、1 次元積分 $\int_{-\infty}^{\infty} e^{-x^2}\,dx$ に由来する。ここでは基本定理を使っても閉形式は得られない。e^{-x^2} は連続であるため原始関数は確かに存在するが、それは初等関数では表せない本質的に新しい関数だからである。驚くべきことに、それでも実数直線全体にわたる積分は、2 次元を経由する回り道をとることで閉形式で求められる。その議論は 3 つのステップからなり、本節で最も美しい計算である。（正規化した原始関数は**誤差関数**（error function）$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt$ であり、これから証明する結果により 1 へと収束する（$x \to \infty$）。標準ライブラリでは `math.erf` として提供されている。）

I. 二乗と因数分解： 正方形 $[-b,b]^2$ 上で積の公式 Equation 34.62 を適用すると、$\iint_{[-b,b]^2} e^{-x^2-y^2}\,dx\,dy = \left(\int_{-b}^{b} e^{-x^2}\,dx\right)^2$ となる。両辺で $b \to \infty$ とすると次式が得られる。

$$\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2 = \lim_{b\to\infty} \iint_{[-b,b]^2} e^{-x^2-y^2}\,dx\,dy = \iint_{\mathbb{R}^2} e^{-x^2-y^2}\,dx\,dy. \tag{34.64}$$

求めたい 1 次元積分の二乗は、単に 2 次元の釣鐘曲面の下の体積にほかならず、この曲面は原点の周りで回転対称である。

II. 極座標： 回転対称性に導かれて、極座標 $\phi(r,\theta) = (r\cos\theta, r\sin\theta)$ を開集合 $\Omega = (0,\infty) \times (0,2\pi)$ 上で導入する。この集合上で写像は単射であり、そのヤコビ行列式は次のようになる。

$$\left|\det \mathbf{J}_\phi\right| = \left|\det \begin{bmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{bmatrix}\right| = r(\cos^2\theta + \sin^2\theta) = r.$$

幾何学的には、動径方向の辺 dr と弧長 $r\,d\theta$ をもつ極座標セルの面積はおよそ $r\,dr\,d\theta$ であり（Figure 34.35）、原点から遠いセルほど幅が広がることを反映している。決定的に重要なのは、この因子 r が、先ほどの置換積分の例で現れた因子 x（$\int_0^1 x\,e^{-x^2}\,dx$）と同じ役割を果たし、不足していた因子を補ってくれることである。

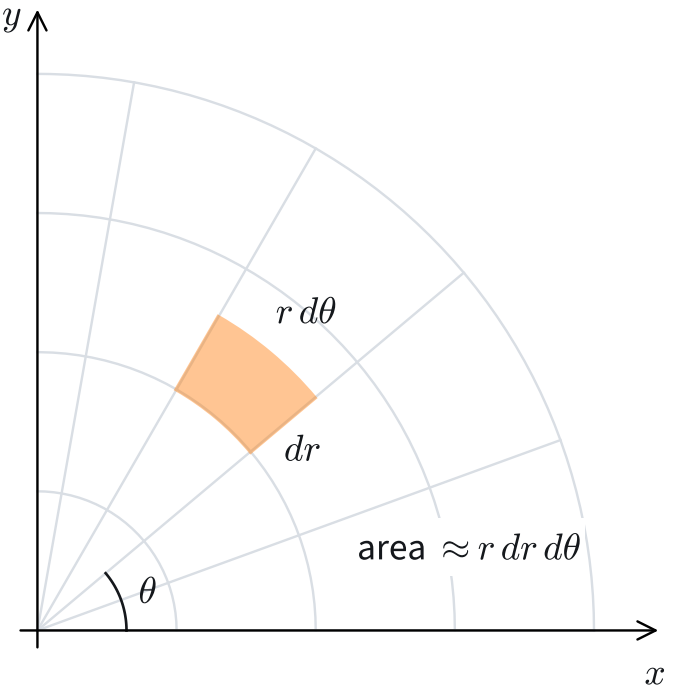


Figure 34.35: 1 つのセルを網掛けした極座標グリッド。動径方向の辺 dr と弧の辺 $r\,d\theta$ により、面積はおよそ $r\,dr\,d\theta$ となる。極座標に変換すると、被積分関数に $\left|\det \mathbf{J}_\phi\right| = r$ が掛かる。

III. 積分：@eq-mdl-change-var-nd を適用し、原始関数 $-\frac{1}{2}e^{-r^2}$（re^{-r^2} の原始関数）を用いると、次式が得られる。

$$\iint_{\mathbb{R}^2} e^{-x^2-y^2}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} r\,e^{-r^2}\,dr\,d\theta = 2\pi\cdot\frac{1}{2} = \pi.$$

Equation 34.64 の正の平方根をとれば、次式が導かれる。

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}. \tag{34.65}$$

こうして、e^{-x^2} の下の全体の面積は $\sqrt{\pi} \approx 1.7725$ と求まる。

3 つの技術的な要点を確認しておけば、この議論は完全に厳密になる。第 1 に、被積分関数が非負であるため、拡大する正方形上の積分と拡大する円板上の積分は同じ極限に収束する。どの円板もある正方形の内部に収まり、どの正方形もある円板の内部に収まるため、広がっていく領域の形状は結果に影響しない。第 2 に、この極座標写像は非負の半直線 $\theta = 0$ を除いているが、その面積はゼロであり積分に一切寄与しない。第 3 に、変数変換は被積分関数が有界かつ連続である有界な長方形 $(0, b) \times (0, 2\pi)$ 上で適用し、そのあとで $b \to \infty$ の極限をとる。

これで標準正規分布の正規化定数が、たった 1 回の置換から導かれる。$z = \sqrt{2}\,x$ と Equation 34.60 でおけば、次式が得られる。

$$\int_{-\infty}^{\infty} e^{-z^2/2}\,dz = \sqrt{2}\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{2\pi}. \tag{34.66}$$

釣鐘関数 $e^{-z^2/2}$ を $\sqrt{2\pi} \approx 2.5066$ で割ると全体の面積が 1 に規格化される。これが標準正規密度に現れる定数の由来である。

ステップ I を数値的に確かめてみよう。拡大する正方形 $[-b, b]^2$ は本当に平面を覆い尽くして π に収束するのだろうか。以下のコードでは、$[-b, b]^2$ 上で $\epsilon = 0.01$ とした 2 次元グリッド和を、正方形上の厳密な体積 $\left(\int_{-b}^{b} e^{-x^2}\,dx\right)^2 = \pi\,\mathrm{erf}(b)^2$ および極限値 π と並べて計算する。

```
eps = 0.01
for b in [1, 2, 3, 6]:
    g = -b + eps * np.arange(round(2 * b / eps))    # left ends of the slices
    x, y = np.meshgrid(g, g, indexing='ij')
    grid = np.sum(eps**2 * np.exp(-x**2 - y**2))
    print(f'b={b}  grid sum={grid:.6f}  pi*erf(b)^2={math.pi * math.erf(b)**2:.6f}'
          f'  pi={math.pi:.6f}')
```

```
b=1  grid sum=2.230949  pi*erf(b)^2=2.230985  pi=3.141593
b=2  grid sum=3.112266  pi*erf(b)^2=3.112270  pi=3.141593
b=3  grid sum=3.141454  pi*erf(b)^2=3.141454  pi=3.141593
b=6  grid sum=3.141593  pi*erf(b)^2=3.141593  pi=3.141593
```

グリッド和は $\pi\,\mathrm{erf}(b)^2$ を精度よく追跡しながら π へと上昇していき、$b = 3$ の時点で正方形の外に残された体積はわずか 10^{-4} 程度にすぎない。$\epsilon = 0.01$ で小数第 4 位以上まで一致するのには明快な理由がある。対称な区間上の偶関数に対しては、左端点則の 1 次バイアス $\frac{\epsilon}{2}\big(f(-b) - f(b)\big)$ が恒等的にゼロになるからである。

いまや標準正規確率 $P(-1 \le Z \le 1) \approx 0.68$ も計算できる。$z = \sqrt{2}\,t$ と置換すると、$p(z) = e^{-z^2/2}/\sqrt{2\pi}$ の下の面積は誤差関数へと変換される。

$$P(-1 \le Z \le 1) = \frac{1}{\sqrt{2\pi}}\int_{-1}^{1} e^{-z^2/2}\,dz = \frac{1}{\sqrt{\pi}}\int_{-1/\sqrt{2}}^{1/\sqrt{2}} e^{-t^2}\,dt = \mathrm{erf}\!\left(1/\sqrt{2}\right) \approx 0.683.$$

これで釣鐘曲線に関して冒頭で立てた 2 つの問いはともに解決し、$p(0) \approx 0.40$ であるのに $P(Z = 0) = 0$ となる見かけ上のパラドックスも解消する。密度が測っているのは単位長さあたりの確率であり、1 点は長さゼロの区間だからである。

冒頭で挙げた問いのうち 1 つがまだ残っている。ランダムサンプリングを通じてモデルを訓練するには、密度のパラメータが動いたときにこの種の面積がどう変化するかを知る必要があり、そのためには積分を微分しなければならない。

34.6.4 まとめ

定積分は細いスライスの和の極限であり、曲線の下の符号付き面積を測る。微分積分学の基本定理はこの極限を原始関数の計算へと置き換える。累積面積関数が $F' = f$ を満たすため、任意の原始関数 G によって $\int_a^b f\,dx = G(b) - G(a)$ が得られるからである。部分積分と置換積分は、それぞれ積の微分法則と連鎖律を逆向きに読んだものである。多次元では、フビニの定理が体積積分を 1 次元の累次積分へと帰着させ、変数変換の伸縮率は局所的な体積の歪みを表す $|\det \mathbf{J}_\phi|$ となる。これを極座標と組み合わせることで、$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$ と標準正規分布の正規化定数 $\sqrt{2\pi}$ が導かれる。

34.6.5 演習問題

1. **累積面積としての活性化関数：** H を単位ステップ関数、すなわち $H(t) = 1$（$t > 0$）、$H(t) = 0$（$t \le 0$）とする。また $\phi(t) = e^{-t^2/2}/\sqrt{2\pi}$ を標準ガウス密度、$\Phi(x) = \int_{-\infty}^{x} \phi(t)\,dt$ をその分布関数とする。
   1. $\int_{-\infty}^{x} H(t)\,dt = \mathrm{ReLU}(x)$ であることを示せ。基本定理 Equation 34.55 のどの仮定が $x = 0$ において破れているか。また、その点における積分の片側導関数はそれぞれどうなるか。
   2. $\Phi(x) = \Phi(0) + \int_0^x \phi(t)\,dt$ と表すことで $\Phi' = \phi$ を導け。続いて $t = \sqrt{2}\,s$ と置換し、$\Phi(x) = \tfrac{1}{2}\bigl(1 + \mathrm{erf}(x/\sqrt{2})\bigr)$ であることを示せ。
   3. 4.1 節の GELU 活性化関数は $x\Phi(x)$ である。その導関数を導出し、導関数が最大および最小となる点を求め、`math.erf` を用いてその点での値を計算せよ。GELU の傾きはどの区間に収まるか。また、その区間は ReLU の傾きである 0 および 1 と比べてどのようになっているか。
2. **重い裾：** 裾指数 $\alpha > 0$ をもつ**パレート密度**は、$p(x) = \alpha x^{-\alpha-1}$（$x \ge 1$ のとき）、それ以外で $p(x) = 0$ と定義される。また**コーシー密度**は、$p(x) = \frac{1}{\pi(1+x^2)}$（$\mathbb{R}$ 上）である。
   1. すべての $\alpha > 0$ に対してパレート密度が正規化されていることを示せ。α がどの範囲にあるとき $\mathbb{E}[X]$ および $\mathbb{E}[X^2]$ は有限になるか。有限となる範囲で $\mathbb{E}[X]$ を計算せよ。
   2. 原始関数 $\arctan x$（$\frac{1}{1+x^2}$ の原始関数）を用いて、コーシー密度が正規化されていることを示せ。
   3. コーシー密度において $\int_{-b}^{b} x\,p(x)\,dx = 0$ がすべての $b > 0$ に対して成り立つことを示せ。それにもかかわらず、$\int_{-\infty}^{\infty}$ の定義（Equation 34.57 に続く定義）のもとでコーシー密度が平均をもたないのはなぜか。また、$\int_0^b x\,p(x)\,dx$ は b とともにどれほどの速さで増大するか。
3. **部分積分によるガンマ関数： ガンマ関数**は $\Gamma(t) = \int_0^\infty u^{t-1}e^{-u}\,du$ で定義される。$t < 1$ のとき被積分関数は $u = 0$ で非有界となり、0 の近くでの積分は極限 $\int_0^1 = \lim_{a\to 0^+}\int_a^1$ によって定義される（34.6.2.2 節と同様）。
   1. どの t に対して積分は $u = 0$ で収束するか。その条件を $\int_0^1 x^{-s}\,dx$ の判定基準（34.6.2.2 節）と比較せよ。
   2. 部分積分 Equation 34.58 を一般の $t > 0$ において適用することで $\Gamma(t+1) = t\,\Gamma(t)$ を示し、$\Gamma(n+1) = n!$ がすべての整数 $n \ge 0$ に対して成り立つことを導け。この議

論のどこで $t > 0$ を用いているか。さらにこの結果を用いて、指数密度 e^{-u}（$u \geq 0$ 上、その平均は 34.6.2.3 節で求めた）の分散を計算せよ。

3. $u = x^2$ と置換し、ガウス積分 Equation 34.65 を用いて $\Gamma(\tfrac{1}{2})$ の値を求めよ。
4. 積の公式 Equation 34.62 を d 回適用すると $\int_{\mathbb{R}^d} e^{-\|\mathbf{x}\|^2}\, d\mathbf{x} = \pi^{d/2}$ が得られる。極座標形式では同じ積分が $S_{d-1}\int_0^\infty r^{d-1}e^{-r^2}\, dr$ となる（ここで S_{d-1} は $\mathbb{R}^d$ における単位球面の表面積である）。動径方向の積分が $\tfrac{1}{2}\Gamma(d/2)$ になることを示し、$S_{d-1} = 2\pi^{d/2}/\Gamma(d/2)$ を導け。続いて $S_{d-1}r^{d-1}$ を $r \in [0,1]$ 上で積分し、単位球体の体積が $V_d = \pi^{d/2}/\Gamma(d/2+1)$ となることを示せ。V_d を $d \in \{1,2,3,5,10,20\}$ に対して計算し、なぜ $V_d \to 0$ となるのかを説明せよ。超立方体 $[-1,1]^{10}$ 上で各軸あたり 3 点とる中点グリッドには 3^{10} 個の点があるが、そのうち単位球体の内部に入る点の割合はどれだけか。また、そのことから高次元におけるグリッドについて何が言えるか。

4. **積分順序の交換：**フビニの定理 Equation 34.61 と、長方形でない領域に対するゼロ拡張を組み合わせれば、積分の順序を自由に選ぶことができる。
   1. $\int_0^1 \big(\int_y^1 e^{-x^2}\, dx\big)\, dy$ の内側の積分は初等的な原始関数をもたない。三角形の積分領域を図示し、積分の順序を入れ替えて閉形式で積分値を求めよ。
   2. $X \geq 0$ が密度 p をもつとする。$x = \int_0^x dt$ を $\mathbb{E}[X] = \int_0^\infty x\, p(x)\, dx$ の内部に代入し、2 つの積分を交換することで $\mathbb{E}[X] = \int_0^\infty P(X > t)\, dt$ を示せ。$\mathbb{E}[X] = \infty$ のときでさえこの交換が正当化されるのはなぜか。密度 $\lambda e^{-\lambda x}$（$x \geq 0$）に対してこの公式を確かめよ。
5. **ガウス分布の正規化定数とモーメント：**本文では標準正規分布を正規化し、釣鐘型の重みに対するスタインの恒等式 Equation 34.59 を用いてそのモーメントを計算した。$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\!\big(-\frac{(x-\mu)^2}{2\sigma^2}\big)$（$\sigma > 0$）とする。
   1. $x = \mu + \sigma z$ と置換し、Equation 34.66 を用いて p が正規化されていることを示せ。どのような μ と σ^2 が密度 $\frac{1}{\sqrt{\pi}}e^{-x^2}$ を与えるか。
   2. X が密度 p をもち、g が連続微分可能で $g(x)\,p(x) \to 0$（$x \to \pm\infty$）を満たすとする。$p'(x) = -\frac{x-\mu}{\sigma^2}\,p(x)$ と部分積分を用いて次式を示せ。

$$\mathbb{E}\big[(X-\mu)\,g(X)\big] = \sigma^2\,\mathbb{E}\big[g'(X)\big]. \tag{34.67}$$

   これから $\mathbb{E}[(X-\mu)^2] = \sigma^2$ および $\mathbb{E}[(X-\mu)^4] = 3\sigma^4$ を導け。
   3. $\Sigma = \mathbf{L}\mathbf{L}^\top$ を $d \times d$ の共分散行列（ただし $\mathbf{L}$ は可逆）とする。変数変換 $\mathbf{x} = \mathbf{L}\mathbf{z}$ を Equation 34.63 に適用し、積の公式 Equation 34.62 と組み合わせて次式を示せ。

$$\int_{\mathbb{R}^d} \exp\!\big(-\tfrac{1}{2}\mathbf{x}^\top\Sigma^{-1}\mathbf{x}\big)\, d\mathbf{x} = (2\pi)^{d/2}\,(\det\Sigma)^{1/2}. \tag{34.68}$$

34.7 期待値とその勾配

これまでに最小化してきた損失のほとんどすべては、ミニバッチ内のデータ例、ランダムなドロップアウトマスク、あるいは入力に注入されたノイズに関する平均である。ランダム性の源がモデルパラメータに依存しない限り、そのような平均の微分は単純明快である。平均の導関数は導関数の平均に等しいため、ミニバッチ勾配降下法（minibatch gradient descent）は何ら特別な仕組みを必要とせず直接これを計算できる。難しさが生じるのは、パラメータが確率分布そのものを制御する場合である。変分オートエンコーダ（variational autoencoder、VAE）は、エンコーダが出力する平均と分散をもつ分布から潜在コード（latent code）を引き、方策勾配（policy gradient）エージェント（14.7 節）は、改善しようとしているまさにその方策（policy）から行

動をサンプリングする。いずれの場合も、目的関数はパラメータとともに分布が変化するサンプルに関する期待値となるが、いったん引かれたサンプルは単なる数値にすぎない。そこには、自身を生成したパラメータの記憶は一切残っていないのである。

この種の最も単純な問題を考えてみよう。$X \sim \mathcal{N}(\theta, 1)$ を平均 θ、単位分散をもつ正規確率変数とし、次の期待二乗損失を考える。

$$\ell(\theta) = \mathbb{E}\big[X^2\big] = \theta^2 + 1.$$

この閉形式は、$X = \theta + Z$（ただし Z は標準正規確率変数）とおけば直ちに導かれる。すなわち $\mathbb{E}[(\theta + Z)^2] = \theta^2 + 2\theta\,\mathbb{E}[Z] + \mathbb{E}[Z^2]$ であり、ここで対称性から $\mathbb{E}[Z] = 0$、また $\mathbb{E}[Z^2] = 1$ である（34.6.2.3 節）。$\theta = 1.5$ における厳密な導関数の値は $\ell'(1.5) = 3$ となる。しかし実際には、モデルは X のサンプルを引くことはできても、$\ell(\theta)$ を閉形式で評価することはできない。では、サンプルのみからどのようにして $\ell'(\theta)$ を推定できるだろうか。

微分積分は、根本的に異なる 2 つの答えを与えてくれる。どちらも期待値を積分として表し（34.6 節）、微分をその積分の内側へと通す。そこから先は道が分かれる。**スコア関数**（score-function）推定量はサンプルには手をつけず確率密度を微分するのに対し、**パスワイズ**（pathwise）推定量はノイズの密度には手をつけずサンプルを微分する。どちらの手法があらゆる設定で優位に立つわけでもなく、両者を比較すること（34.7.3 節）が本節の中心的な目標である。それに先立ち、まずは期待値がどのように積分として書かれるかを簡単に振り返り、微分を積分の内側へ通す操作がいつ正当化されるのかを確かめておこう。このステップこそ、巧妙な誤りが紛れ込みやすい箇所だからである。

34.7.1 積分としての期待値

何かを微分する前に、密度と期待値の記法を定めておこう。確率の章ではこれら両方の概念を詳しく展開するが（36.1 節）、ここでは前節の積分の言葉を用いて改めて述べる。領域 $\mathcal{X}$ 上の確率**密度**（probability density）とは、全体の積分が 1 に等しい非負関数のことである。

$$p(x) \geq 0, \qquad \int_{\mathcal{X}} p(x)\,dx = 1. \tag{34.69}$$

確率変数 X（密度 p をもつ）が区間 $[a, b]$ に入る確率は、その区間上における p の下の面積 $P(a \leq X \leq b) = \int_a^b p(x)\,dx$ で与えられる（36.1 節）。密度は確率そのものではなく単位長さあたりの確率を測るため、その値は 1 を超えうる。たとえば $[0, \tfrac{1}{2}]$ 上の一様密度は、その台（support）上のいたるところで 2 に等しい。本節の主な通し例は**標準正規**分布 $\mathcal{N}(0, 1)$ であり、その密度 $p(z) = e^{-z^2/2}/\sqrt{2\pi}$ は 34.6 節のガウス積分によって正規化されている。

期待値（expectation）は、各結果にその確率密度で重み付けしたうえで、X あるいは関数 $g(X)$ の重み付き平均を計算するものである。

$$\mathbb{E}[X] = \int_{\mathcal{X}} x\,p(x)\,dx, \qquad \mathbb{E}[g(X)] = \int_{\mathcal{X}} g(x)\,p(x)\,dx. \tag{34.70}$$

期待値 $\mathbb{E}[g(X)]$ は、積分が絶対収束するとき、すなわち $\int_{\mathcal{X}} |g(x)|\,p(x)\,dx < \infty$ が成り立つときに確定した値をもつ。裾の重い分布はこの条件に反することがあり、たとえば演習問題 2 のコーシー密度は有限の平均をもたない。

深層学習では、期待値が閉形式の式をもつことはめったになく、しかも数千次元もの空間で定義されることが多いため、グリッド上の数値積分（numerical quadrature）は到底現実的ではない。そこで代わりに、独立なサンプル $x_1, \dots, x_n$ を p から引き、標本平均 $\frac{1}{n}\sum_{i=1}^{n} g(x_i)$ を作る。これは**モンテカルロ**推定値として知られる。1 サンプル $g(X)$ の標準偏差が σ であれば、モンテカルロ平均の二乗平均平方根誤差（root-mean-square error、RMSE）は次元にかかわらず $\sigma/\sqrt{n}$ となる（36.5.1.7 節）。本節で扱う 2 つの勾配推定量はいずれもこの形のモンテカルロ平均である。どちらも同じ真の勾配に対する不偏推定量であるため、違いは 1 サンプルあたりのばらつきにしか現れない。したがって、1 サンプルの標準偏差 σ が両者を比較する自然な尺度となる。

34.7.2 積分記号下の微分

34.7.2.1 順序交換

例題の損失 $\ell(\theta) = \mathbb{E}[X^2]$ は、パラメータ θ に応じて変化する積分である。一般に、パラメータが積分に作用する箇所は 2 つある。微分積分学の基本定理（34.6 節）が扱う積分区間の端点を通じて作用する場合と、被積分関数そのものを通じて作用する場合である。

積分区間の端点が固定されており、被積分関数のみが θ に依存するときは、微分を積分の内側へ通すことで微分できる。

$$\frac{d}{d\theta}\int f(x,\theta)\,dx = \int \frac{\partial f}{\partial \theta}(x,\theta)\,dx. \tag{34.71}$$

幾何学的には、全体の面積の変化率は、各点における高さの変化率を積分したものに等しい。以下では、この微分と積分の入れ替えを**順序交換**（swap）と呼ぶ。2 つの勾配推定量はいずれもこれに依拠している。導関数は差分商の極限であり、積分は和の極限であり、有限個の和と差分は常に可換であるから、一見するとこの恒等式には何の問題もないように思える。しかし、2 つの極限の交換は決して自動的に成り立つものではなく、正則性条件（regularity condition）を必要とする。x の有界区間上では、$\partial_\theta f$ が両変数に関して連続であれば十分である。非有界な領域上では、それに加えて x の可積分関数であって、$|\partial_\theta f(x,\theta')|$ をすべての θ'（θ の近傍）に対して上から抑えるものが存在することも必要となる。これは優収束定理（dominated convergence theorem）からの直接の帰結である（証明は省き (Folland 1999) による）。正規密度を用いた通し例 $f(x,\theta) = x^2 p_\theta(x)$ では、ガウス分布の裾がいかなる多項式の増大よりも速く減衰するため、このような優関数による上界が常に存在する。

34.7.2.2 ライプニッツ則

積分区間の端点も θ とともに動く場合、単純な順序交換だけでは不完全であり、誤った答えを与えてしまう。ライプニッツ則（Leibniz rule）は不足している境界項を補うものであり、内部の変化と移動する境界からの寄与を単純に足し合わせればよい（Figure 34.36）。

命題（ライプニッツ則）： **f および $\partial_\theta f$ が、$\{(x,\theta) : x \in [a(\theta), b(\theta)]\}$（ただし θ はある開区間を動く）を含む領域上で連続であり、かつ a と b が微分可能であるとする。このとき、次式が成り立つ。**

$$\frac{d}{d\theta}\int_{a(\theta)}^{b(\theta)} f(x,\theta)\,dx = \int_{a(\theta)}^{b(\theta)} \frac{\partial f}{\partial \theta}(x,\theta)\,dx + f\big(b(\theta),\theta\big)\,b'(\theta) - f\big(a(\theta),\theta\big)\,a'(\theta). \tag{34.72}$$

証明： $H(\alpha,\beta,\theta)=\int_\alpha^\beta f(x,\theta)\,dx$ と定義する。微分積分学の基本定理により $\partial_\beta H = f(\beta,\theta)$ および $\partial_\alpha H = -f(\alpha,\theta)$ となり、一方で有界区間 $[\alpha,\beta]$ 上での順序交換により $\partial_\theta H = \int_\alpha^\beta \partial_\theta f\,dx$ が得られる。これら 3 つの偏微分はすべて連続であるため、多変数の連鎖律（34.3 節）が合成写像 $\theta\mapsto H(a(\theta),b(\theta),\theta)$ に適用でき、3 つの項がそれぞれ $a'(\theta)$、$b'(\theta)$、1 で重み付けされて結合される。■

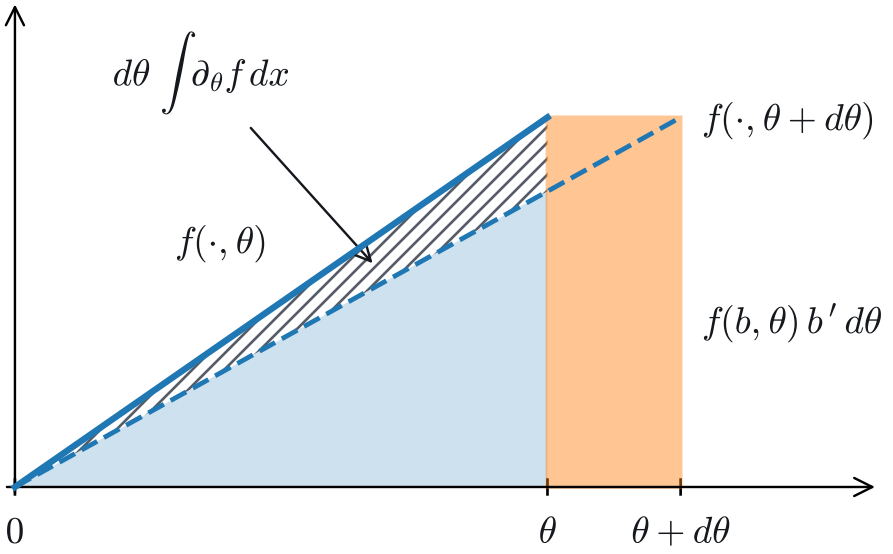


Figure 34.36: 一様分布の例である $f(x,\theta)=x/\theta$（区間 $[0,\theta]$ 上）に対するライプニッツ則。θ が $d\theta$ だけ増加すると、被積分関数の曲線は $f(\cdot,\theta+d\theta)$（破線。ここでは下がるため、斜線領域は失われる面積を表す）へと移動する一方、右側の境界は外側へ動いて高さ $f(b,\theta)$、幅 $b'\,d\theta$ の細い帯を掃き込む。

境界項を無視することは些細な誤差にとどまらず、導関数の符号を反転させてしまうことさえある。X が $[0,\theta]$ 上の一様分布に従い、その区間上で密度 $1/\theta$ をもつとしよう。その平均は $\mathbb{E}[X]=\int_0^\theta \frac{x}{\theta}\,dx=\frac{\theta}{2}$ であり、導関数は $+\frac12$ である。被積分関数だけを微分すると $\int_0^\theta -\frac{x}{\theta^2}\,dx=-\frac12$ となり、符号が逆になってしまう。ここに上端の境界からの寄与 $f(\theta,\theta)\,b'(\theta)=\frac{\theta}{\theta}\cdot 1=1$ を加えることで、$-\frac12+1=+\frac12$ と真の導関数が復元される。分布の台がパラメータに依存する場合には、単純な順序交換だけでは正当化されない（演習問題 1 では $\mathbb{E}[X^2]$ を計算する）。

34.7.2.3 Log-Integral-Exp

順序交換からは、log-sum-exp 関数の連続版も導かれる。連鎖律と Equation 34.71 を $\log\int e^{\theta x}p(x)\,dx$ に適用すると、次式が得られる。

$$\frac{d}{d\theta}\log\int e^{\theta x}p(x)\,dx=\frac{\int x\,e^{\theta x}p(x)\,dx}{\int e^{\theta x}p(x)\,dx}.$$

右辺は、x の平均を $e^{\theta x}p(x)$ に比例する指数的に**傾斜させた**（tilted）密度のもとでとったものである。標準正規密度の場合、$\theta z-\frac12 z^2=\frac12\theta^2-\frac12(z-\theta)^2$ と平方完成することで、傾斜密度

が $\mathcal{N}(\theta, 1)$ であり、かつ $\log \mathbb{E}[e^{\theta Z}] = \theta^2/2$ となることがわかる。実際、その導関数 θ は傾斜平均に一致している。log-sum-exp の勾配がソフトマックスベクトルであるのとちょうど同じように、log-integral-exp の勾配は期待値となる。この恒等式には、対数分配関数（log-partition function）の導関数 $A'(\eta)$ として 36.4 節で再び出会うことになる。

34.7.3 スコア関数勾配とパスワイズ勾配

期待損失に順序交換を適用すると、積分 $\int g(x)\, \partial_\theta p_\theta(x)\, dx$ が得られる。残念ながら、この積分をモンテカルロサンプリングで直接推定することはできない。$\partial_\theta p_\theta$ は確率密度ではないため（正と負の両方の値をとり、積分するとゼロになる）、そこからサンプルを引くことができないからである。しかし 2 つの単純な恒等式を使えば、この積分をサンプルから推定可能な期待値の形へと書き戻せる。

34.7.3.1 スコア関数推定量

p_θ を θ によってパラメータ化された密度とする。連鎖律を $\log p_\theta(x)$ に適用すると、次の**対数微分の恒等式**（log-derivative identity）が得られる。

$$\nabla_\theta\, p_\theta(x) = p_\theta(x)\, \nabla_\theta \log p_\theta(x). \tag{34.73}$$

言葉で言えば、密度の勾配は、密度そのものにその対数の勾配（**スコア**と呼ばれるベクトル $\nabla_\theta \log p_\theta$）を掛けたものに等しい。順序交換が正当化され、かつ p_θ の台が θ とともに動かないときには、スコアの期待値はゼロになる。すなわち $\mathbb{E}[\nabla_\theta \log p_\theta(X)] = \int \nabla_\theta p_\theta\, dx = \nabla_\theta \int p_\theta\, dx = \nabla_\theta 1 = \mathbf{0}$ である。$[0, \theta]$ 上の一様密度を見れば、なぜ台が固定されていることが不可欠なのかがわかる。そのスコアは $\partial_\theta \log(1/\theta) = -1/\theta$ であり、期待値は $-1/\theta \neq 0$ となる。というのも、$\int_0^\theta p_\theta(x)\, dx = 1$ を微分すると境界項 $p_\theta(\theta) = 1/\theta$ も現れるからである。なお、スコアの平均がゼロになる性質は、フィッシャー情報量（Fisher information、36.6 節）の出発点でもある。

Equation 34.73 に関数 $g(x)$（θ に依存しない）を掛けて積分すると、次の**スコア関数恒等式**（score-function identity）が得られる。

$$\nabla_\theta\, \mathbb{E}_{X \sim p_\theta}[g(X)] = \int g(x)\, \nabla_\theta p_\theta(x)\, dx = \mathbb{E}\big[g(X)\, \nabla_\theta \log p_\theta(X)\big]. \tag{34.74}$$

この恒等式により、期待損失の勾配は再び p_θ のもとでの期待値へと戻る。各サンプルの損失 $g(x)$ が、θ がそのサンプルの対数確率（log-probability）をどれだけ強く押し上げるかによって重み付けされるのである。これをモンテカルロ法で推定するには、サンプル $x_i \sim p_\theta$ を引き、$g(x_i) \nabla_\theta \log p_\theta(x_i)$ の平均をとればよい。強化学習（reinforcement learning、RL）において、これは方策勾配法（14.7 節）の REINFORCE 推定量である (Williams 1992)。（もし g も θ に直接依存する場合には、積の微分法則により単に $\mathbb{E}[\nabla_\theta g(X, \theta)]$ の項が加わる。）

スコアの平均がゼロであるため、定数の**ベースライン** b（x に依存しない）を任意に選んで $g(X)$ から引いても、推定量に統計的バイアスを生じさせることはない（14.8 節）。すなわち $\mathbb{E}[(g(X) - b) \nabla_\theta \log p_\theta(X)] = \mathbb{E}[g(X) \nabla_\theta \log p_\theta(X)]$ が成り立つ。b を適切に選べば、推定値の分散を大幅に減らすことができる。

34.7.3.2 パスワイズ推定量

2 つ目の恒等式は、サンプルに θ の記憶を取り戻させるものである。多くの場合、p_θ からのサンプルは、決定論的な変換 $x = T_\theta(\mathbf{z})$ を補助的なノイズ変数 $\mathbf{z} \sim q$（その分布は θ に依存しない）へ施すことで生成できる。たとえば正規確率変数 $\mathcal{N}(\mu, \sigma^2)$ は、単に $\mu + \sigma z$（ただし $z \sim \mathcal{N}(0,1)$）と表せる。期待値を $\mathbf{z}$ に関する積分として書き直せば、密度から θ が完全に取り除かれるため、順序交換によって次の**パスワイズ恒等式**（pathwise identity）が得られる。

$$\nabla_\theta \, \mathbb{E}_{X \sim p_\theta}[g(X)] = \nabla_\theta \int g(T_\theta(\mathbf{z}))\, q(\mathbf{z})\, d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim q}\Big[\nabla_\theta \, g(T_\theta(\mathbf{z}))\Big]. \tag{34.75}$$

この推定量は密度を微分する代わりに、サンプルそのものを通して微分する。自動微分が θ から T_θ を経て g に至る経路をたどるのである。これは**再パラメータ化トリック**（reparameterization trick）として知られ、変分オートエンコーダがガウス潜在変数（latent variable）を通して逆伝播する仕組みである (Kingma and Welling 2014; Rezende et al. 2014)。推定量自体の歴史はさらに古く、シミュレーションの分野ではパスワイズ微分や微小摂動微分（infinitesimal-perturbation derivative）の名で長く使われてきた (Mohamed et al. 2020)。正当に適用するには、T_θ が θ に関して微分可能であること、g が経路に沿って微分可能であること（Lipschitz 連続な g であれば、X が連続密度をもつ限り十分である）、そして順序交換のための可積分な優関数による上界が存在することが必要となる。なお、たとえ X の台が θ とともに動いたとしても、積分は $\mathbf{z}$（その密度 q は固定されたままである）に関してとられるため、境界項は一切現れないことに注目しよう。

g に対する微分可能性の要件は、無視してよい単なる技術的細目ではない。ステップ関数 $g(x) = \mathbf{1}(x > 0)$（すなわち 1 を正の x に対してとり、それ以外では 0 となる関数）を $X = \theta + z$ のもとで考えてみよう。真の期待値 $\mathbb{E}[g(X)] = P(z > -\theta)$ の導関数は $\varphi(\theta) = e^{-\theta^2/2}/\sqrt{2\pi}$ であり、その値は約 0.13（$\theta = 1.5$ のとき）である。ところがパスワイズ推定量のサンプル $\partial_\theta \mathbf{1}(\theta + z > 0)$ は 0 を値としてとり、それがすべての z（確率ゼロの事象である唯一の点 $z = -\theta$ を除く）で成り立つため、パスワイズ推定量は 0 を確率 1 で返してしまう。対照的に、スコア関数推定量のサンプル $\mathbf{1}(x > 0)\, z$ は期待値 $\int_{-\theta}^{\infty} z\, \varphi(z)\, dz = \big[-\varphi(z)\big]_{-\theta}^{\infty} = \varphi(\theta)$ をもち、厳密な導関数を復元する。スコア関数推定量は g やサンプリングのステップを通して微分することが一切ないため、不連続な損失にも、ベルヌーイ分布やカテゴリ選択のような離散分布（そこでは θ からサンプルへ至る微分可能な経路が存在しない）にもまったく同様に適用できる。

34.7.3.3 2 つの推定量の比較

これで同じ導関数に対する 2 つの不偏推定量（unbiased estimator）が出そろった。通し例である $\ell(\theta) = \mathbb{E}[X^2]$（ただし $X = \theta + z$）は十分に単純なため、両者の分散を手計算で求めることができる。パスワイズ推定量では、1 サンプルあたりの寄与は $\partial_\theta (\theta + z)^2 = 2X$ であり、その分散は $\mathrm{Var}(2X) = 4$ となる。スコア関数推定量では、正規位置族（normal location family）のスコアが $\partial_\theta \log p_\theta(X) = X - \theta = z$ であるから、1 サンプルあたりの寄与は $X^2 z$、あるいは $(X^2 - b)z$（ベースライン b を用いる場合）となる。$X^2 z$ の分散を求めるには、標準正規分布のモーメント $\mathbb{E}[z^2] = 1$、$\mathbb{E}[z^4] = 3$、$\mathbb{E}[z^6] = 15$ を用いる。これらはいずれもスタインの恒等式 $\mathbb{E}[g'(Z)] = \mathbb{E}[Z\, g(Z)]$ から導かれる（34.6.2.3 節）。対称性により、z の奇数乗の項は二次モーメント（second moment）においてすべて消える。

$$\mathbb{E}\big[(\theta + z)^4 z^2\big] = \theta^4\, \mathbb{E}[z^2] + 6\theta^2\, \mathbb{E}[z^4] + \mathbb{E}[z^6] = \theta^4 + 18\theta^2 + 15.$$

ここから平均の二乗 $(2\theta)^2$ を引くと、分散 $\theta^4 + 14\theta^2 + 15$ が残る。分散を最小化する定数ベースラインは $b^\star = \theta^2 + 3$ となることがわかり、これにより分散は $8\theta^2 + 6$ まで下がる（演習問題 2）。$\theta = 1.5$ における 3 つの推定量の 1 サンプル標準偏差は、パスワイズ推定量が 2、素のスコア

関数推定量が 7.18、最適ベースライン付きスコア関数推定量が 4.90 となる。ベースラインは目に見えて有効ではあるものの、この損失に関しては差を埋めきることはできない。$8\theta^2 + 6 > 4$ がすべての θ に対して成り立つからである。

これら 3 つの閉形式による予測を、1 回のモンテカルロシミュレーションで確かめてみよう。パスワイズ微分は単に $2x_i$ であるため、ここでは自動微分すら必要ない。ここまでの式変形が正しければ、3 つの標本標準偏差はそれぞれ 2、7.18、4.90 の近くに収まり、各標本平均は標準誤差（standard error）$\mathrm{sd}/\sqrt{n}$ の数倍以内で真の導関数の値 3 に一致するはずである。

```
theta, n = 1.5, 200_000
z = np.random.default_rng().standard_normal(n)     # noise, independent of theta
x = theta + z                                      # samples of N(theta, 1)
pathwise = 2 * x                                   # d/dtheta (theta + z_i)^2
score = x**2 * z                                   # g(x) times the score x - theta
baseline = (x**2 - (theta**2 + 3)) * z             # with the optimal baseline
exact_sd = {'pathwise': 2, 'score': math.sqrt(theta**4 + 14 * theta**2 + 15),
            'score+baseline': math.sqrt(8 * theta**2 + 6)}
print(f'true derivative 2*theta = {2 * theta}')
for name, s in zip(exact_sd, [pathwise, score, baseline]):
    print(f'{name:15s} mean={s.mean():.4f}  sd={s.std():.3f} '
          f'(exact {exact_sd[name]:.3f})  se={s.std() / math.sqrt(n):.4f}')
```

```
true derivative 2*theta = 3.0
pathwise        mean=3.0029  sd=2.001 (exact 2.000)  se=0.0045
score           mean=3.0204  sd=7.268 (exact 7.181)  se=0.0163
score+baseline  mean=3.0128  sd=4.960 (exact 4.899)  se=0.0111
```

典型的な実行結果では、3 つの平均はいずれも標準誤差の 2〜3 倍以内で 3 の近くに収まり、経験標準偏差も閉形式の予測と数パーセント以内で一致する。一定のサンプル数の予算のもとでは、パスワイズ推定値は素のスコア関数推定値より約 3.6 倍精度が高い。言い換えれば、素のスコア関数推定量が同じ精度に達するには約 13 倍のサンプル数を要する。

しかし、パスワイズ推定量のほうが常に分散が小さいと結論づけるのは誤りである。振動する損失 $g(x) = \sin(\omega x)$ を考えてみよう。このとき期待値の真の導関数は $\omega \cos(\omega\theta) e^{-\omega^2/2}$ となる。パスワイズ推定量のサンプル $\omega \cos(\omega X)$ は周波数 ω に比例して大きくなり、$\omega/\sqrt{2}$ に近い標準偏差をもつのに対し、スコア関数推定量のサンプル $\sin(\omega X)\, z$ の標準偏差は高々 $\sqrt{\mathbb{E}[z^2]} = 1$ である（$|\sin| \le 1$ だからである）。被積分関数が激しく振動する場合、関数値そのものは有界にとどまっていても、その導関数は巨大になりうる。すでに引いたサンプルを再利用し、$\omega = 1$ と $\omega = 3$ において 2 つの推定量を比較してみよう。

```
for omega in [1, 3]:
    true = omega * math.cos(omega * theta) * math.exp(-omega**2 / 2)
    path, score = omega * np.cos(omega * x), np.sin(omega * x) * z
    print(f'omega={omega}  true={true:+.4f}  pathwise {path.mean():+.4f} '
          f'(sd {path.std():.2f})  score {score.mean():+.4f} (sd {score.std():.2f})')
```

```
omega=1  true=+0.0429  pathwise +0.0431 (sd 0.66)  score +0.0405 (sd 0.55)
omega=3  true=-0.0070  pathwise -0.0025 (sd 2.12)  score -0.0106 (sd 0.71)
```

$\omega = 1$ の時点ですでにスコア関数推定量のほうがばらつきが小さく（約 0.5 対 0.7）、$\omega = 3$ では 3 倍の差をつけて優位に立つ（約 0.7 対 2.1）。どちらの標本平均も、標準誤差の数倍以内で厳密な導関数の値と一致している。

どちらの推定量も、あらゆる問題で優位に立つわけではない。パスワイズ推定量は g を微分するため、g' が大きいときに分散が増大する。一方、スコア関数推定量は g にスコアを掛けるため、g そのものが大きいときに分散が増大する。いずれの場合も、所定の精度を得るためにどれだけのサンプル数が必要かは、1 サンプルの標準偏差 σ によって決まる。この σ はサンプルを引く元の分布にも依存するため、σ を小さくするような提案分布（proposal distribution）を選ぶという発想から、直ちに重要度サンプリング（importance sampling）が導かれる。これについては 36.8 節で取り上げる。

34.7.4 まとめ

モデルのパラメータがサンプルの分布を支配するとき、期待損失の勾配をモンテカルロサンプリングで推定するには、まずその勾配を期待値の形に書き直さなければならない。そのための 2 つの方法はいずれも微分を積分の内側へと通すが、このステップには可積分な優関数による上界が必要であり、さらに積分区間の端点がパラメータとともに動く場合にはライプニッツ則の境界項も必要となる。スコア関数恒等式は対数密度を微分するものであり、任意の損失 g や離散確率変数に対して機能するが、サンプルの値が g の大きさに比例するため、適切なベースラインがなければノイズが大きくなりやすい。一方、パスワイズ恒等式は再パラメータ化されたサンプルを通して微分するものであり、微分可能な g を必要とするが、常にではないものの多くの場合ずっと低い分散を達成する。どちらの推定量も二乗平均平方根誤差 $\sigma/\sqrt{n}$ をもつため（36.5.1.7 節）、与えられた問題に対してどちらが望ましいかは、1 サンプルの標準偏差 σ によって決まる。

34.7.5 演習問題

1. **動く台：** X が区間 $[0,\theta]$（ただし $\theta > 0$）上の一様分布に従うとし、$\ell(\theta) = \mathbb{E}[X^2] = \int_0^\theta \frac{x^2}{\theta}\,dx = \frac{\theta^2}{3}$ かつ $\ell'(\theta) = \frac{2\theta}{3}$ とする。
   1. ℓ をライプニッツ則 Equation 34.72 を用いて微分せよ。その結果のどの部分が単純な順序交換 Equation 34.71 から得られ、どの部分が欠落するか。
   2. 台の上では $\log p_\theta(x) = -\log\theta$ である。スコア関数推定量 $X^2\,\partial_\theta \log p_\theta(X)$（Equation 34.74）の期待値を計算せよ。続いて g を $[0,\infty)$ 上の連続関数とする。$g(X)\,\partial_\theta \log p_\theta(X)$ の平均が、単純な順序交換 Equation 34.71 を $\mathbb{E}[g(X)] = \int_0^\theta g(x)\,p_\theta(x)\,dx$ に適用したものと等しいことを示せ。これが $\frac{d}{d\theta}\mathbb{E}[g(X)]$ の推定値としてもつ統計的バイアスは、$g(\theta)$ と θ を用いてどのように表されるか。
   3. $X = \theta U$（ただし U は $[0,1]$ 上の一様分布に従う）と表し、Equation 34.75 のパスワイズ推定量の期待値を計算せよ。X の台が θ とともに動くにもかかわらず、境界項が不要であるのはなぜか。
2. [code] **四次関数に対する推定量の分散：**●$X \sim \mathcal{N}(\theta, 1)$ とし、$X = \theta + z$ と表し、$s(X) = \partial_\theta \log p_\theta(X) = z$ をスコアとする。
   1. $g(X) = X^4$ とおくと、$\frac{d}{d\theta}\mathbb{E}[X^4] = 4\theta^3 + 12\theta$ となる。1 サンプルのパスワイズ推定量を Equation 34.75 から、1 サンプルのスコア関数推定量を Equation 34.74 からそれぞれ導出せよ。演習問題 5 (b) の恒等式を用いて、両者の平均がいずれも $4\theta^3 + 12\theta$ となることを示せ。
   2. 定数ベースライン b に対し、$(g(X) - b)\,s(X)$ の分散が $b^\star = \mathbb{E}[g(X)s(X)^2]/\mathbb{E}[s(X)^2]$ によって最小化されることを示せ。$g(X) = X^2$ の場合に $b^\star = \theta^2 + 3$ となることを示し、さらに $\mathbb{E}[z^2] = 1$、$\mathbb{E}[z^4] = 3$、$\mathbb{E}[z^6] = 15$ を用いて、本文で挙げた 3 つの分散 4、$\theta^4 + 14\theta^2 + 15$、$8\theta^2 + 6$ を導出せよ。
   3. $b^\star$ を $g(X) = X^4$ に対して求めよ。10^6 個のサンプルを用い、$\theta \in \{0, 1, 2\}$ においてパスワイズ推定量、スコア関数推定量、ベースライン付きスコア関数推定量の標準偏差を推定せよ。どの θ においてベースラインによる改善効果が最も大きいか。また、パスワイズ推定量との差を埋めきることはあるか。g' と g の大きさから、これらの順

序関係を説明せよ。

4. $g(x) = \mathbf{1}(x > 0)$ を $\theta = 1.5$ において考える。本文では、パスワイズ推定量がほとんどすべてのサンプルで 0 となる一方、スコア関数推定量 $\mathbf{1}(x > 0)\, z$ は正しい平均 $\varphi(\theta) = 0.1295$ をもつことを示した。シミュレーションによって両者を確かめよ。

3. [extended] **コイン投げを通じた勾配：●●**x が離散的な選択であるとき、θ からサンプルへ至る微分可能な経路は存在しないため、パスワイズ推定量は存在しない。Gumbel-softmax (Jang et al. 2017; Maddison et al. 2017) のような連続緩和は微分可能な代理変数を構成し、ストレートスルー推定量 (Bengio, Léonard, et al. 2013) は順伝播でハードサンプルを用い、逆伝播でその代理変数を用いるが、いずれも元の目的関数に対して統計的バイアスをもつ。本問ではそのバイアスを測定する。Tucker et al. (2017) はこれを簡単な例題で示している。1 つのベルヌーイ変数 $b \in \{0, 1\}$ が $P(b = 1) = \sigma(\theta)$（ただし θ はロジット）を満たし、目的関数が $\ell(\theta) = \mathbb{E}[(b - t)^2]$（ターゲット $t = 0.45$）であるとする。U を $(0, 1)$ 上の一様確率変数とし、$\zeta = \log U - \log(1 - U)$ とおき、温度 $\tau > 0$ に対して $z = \sigma((\theta + \zeta)/\tau)$（Gumbel-softmax 緩和の 2 クラスの場合）とおく。

   1. $\ell(\theta)$ と $\ell'(\theta)$ を閉形式で求めよ。どの確率 $\sigma(\theta)$ が ℓ を最小化するか。また、有限の θ でそれは達成されるか。
   2. $b = \mathbf{1}(z > \tfrac{1}{2})$ が $P(b = 1) = \sigma(\theta)$ をすべての τ に対して満たすことを示せ。
   3. $\ell'(\theta)$ に対する 1 サンプル推定量を次の 3 通り実装せよ。すなわち、REINFORCE Equation 34.74（ベースラインは $\ell(\theta)$）、緩和推定量 $\frac{d}{d\theta}(z - t)^2$、そして損失にはハードサンプルを用い導関数には緩和サンプルを用いるストレートスルー推定量 $2(b - t)\frac{dz}{d\theta}$ である。$\ell'(\theta)$ と比較して各推定量の平均と標準偏差を、$\theta \in \{-2, 0, 2\}$ および $\tau \in \{1, 0.5, 0.1\}$ に対して表にまとめよ。各推定量のバイアスは θ と τ にどのように依存するか。
   4. θ に対する確率的勾配降下法を、初期値 $\theta = 0$、ステップサイズ 1 で 5000 ステップにわたり、REINFORCE、緩和推定量、ストレートスルー推定量のそれぞれ（各 τ について）で実行せよ。ただし REINFORCE のベースラインは過去の損失の指数移動平均に置き換えること。各実行に沿って $\ell(\theta)$ をプロットせよ。さらに緩和目的関数 $R_\tau(\theta) = \mathbb{E}[(z - t)^2]$ を U 上の数値積分によって計算し、その最小点を緩和推定量の実行が落ち着く θ の値と比較せよ。τ を小さくすると、緩和推定量のバイアスと分散はどのようなトレードオフを示すか。

34.8 関連文献

本章全体を通して、深層学習の実践では日常的に踏み越えられる見通しのよい仮定のもとで主要な結果を証明してきた。実際には、ReLU ネットワークには鋭い折れ点（kink）が至るところにあり、リバースモードのテープは容易に GPU メモリを超過し、1,000 次元の積分はグリッド上に配置できない。以下の短い解説では、こうした仮定が崩れたときに何が起こるかをまとめ、続いて本章の主要内容を扱う教科書、講義、チュートリアルを厳選して紹介する。なかでも日常的な訓練に直結するため最初に読んでおきたいトピックが 2 つある。1 つは、折れ点を経由する誤差逆伝播法（backpropagation）がどのような意味で依然として妥当であるかを説明する保存場（conservative field）であり、もう 1 つは、長い計算においてメモリを扱いやすい範囲に抑える再帰的チェックポイント保存（recursive checkpointing）である。本書の他の箇所で本節の内容を前提とするものはない。

34.8.1 折れ点における微分

34.8.1.1 保存場

34.1 節で例示したように、標準的な規約 $\mathrm{ReLU}'(0) = 0$ を採用すると奇妙な現象が生じる。恒等関数を $\mathrm{ReLU}(x) - \mathrm{ReLU}(-x) = x$ と表すと、誤差逆伝播法は傾き 0 を $x = 0$ において返すが、この値は x の真の導関数でもなければ有効な劣勾配（subgradient）ですらない。古典的な劣微分（subdifferential）ではこの挙動を説明できない。構成要素の関数が凸でなくなった途端、和の法則や連鎖律（chain rule）が集合の包含関係としてしか成り立たなくなるからである。この隔たりを埋めるため、Bolte and Pauwels (2021) は**保存場**を導入した。これは構成上そのまま連鎖律を満たし、測度ゼロの集合を除くいたるところで通常の勾配と一致する、導関数の集合値への一般化である。各折れ点における傾きの選択をどのように固定しても、誤差逆伝播法は常にそのような場の要素を返す。では、なぜ折れ点からなる例外集合は問題にならないほど小さいのだろうか。ラデマッハの定理（Rademacher's theorem）(Rademacher 1919) により、$\mathbb{R}^n$ 上のすべてのリプシッツ関数はほとんどいたるところで微分可能であるため、ReLU ネットワークの損失が持つ折れ点は測度ゼロの集合にとどまる。

同論文は、誤差逆伝播法によるこれらの値で駆動される確率的勾配降下法（stochastic gradient descent）が依然として収束するかという実務上の問いにも決着をつけている。標準的なニューラルネットワークの層から構成された損失に対し、反復点の有界性と、ステップサイズおよび勾配ノイズ（gradient noise）に関する通常の条件を仮定すれば、反復点は保存場がゼロを含む臨界点（critical point）の集合へと収束する。この保証は厳密演算のもとで成り立つものであるが、有限精度の浮動小数点演算では、反復点が実際に折れ点の上に着地することがある。34.1 節では、Bertoin et al. (2021) が計測し 2023 年の正誤表で修正した 3 つの数値精度にわたる発生頻度を紹介するとともに、$\mathrm{ReLU}'(0)$ の選び方が結果的に訓練へほとんど影響しないことを示している。

34.8.1.2 劣勾配法

滑らかな最適化は、勾配と逆向きの小さなステップが目的関数を下げることを保証する降下補題（descent lemma）に依拠している。滑らかでない関数ではこの保証が崩れる。$|x|$ の最小点において、有効な劣勾配 $\frac{1}{2}$ は実際には上り方向を指すことが 34.1 節に示されている。それでもなお驚くべきことに、劣勾配法（subgradient method）は収束する。その解析が目的関数値（objective value）そのものではなく、最小点（minimizer）までのユークリッド距離を追跡するからである。$\sum_{t} \eta_t = \infty$ かつ $\sum_{t} \eta_t^2 < \infty$ を満たす減衰ステップサイズを用いれば、それまでに得られた最良の目的関数値は最小値へと収束する。最適な収束率（convergence rate）を得るには別のスケジュールが必要となる。劣勾配のノルムが高々 G である凸関数に対し、最小点から距離 R の点を出発点として定数ステップサイズ $R/(G\sqrt{T})$ を用いると、T 個の反復点のうち最良のものは最小値から $RG/\sqrt{T}$ 以内に到達する。次元が T を超える場合、劣勾配のみに依拠するいかなる手法も、この収束率を定数倍を超えて改善することはできない (Nesterov 2018)。和の法則が等号で成り立つための厳密な条件を含め、劣微分法の体系的な展開については Rockafellar (1970) を参照されたい。

34.8.2 ベクトル写像と行列写像の微分

34.8.2.1 ベクトル値写像に対する平均値の定理

1 次元では、34.1 節の平均値の定理（mean value theorem）により、ある中間点 ξ が存在して $f(b) - f(a) = f'(\xi)(b - a)$ が成り立つことが保証される。ベクトル値曲線 $\gamma : \mathbb{R} \to \mathbb{R}^n$ に対しても同じ等式が成り立つと考えたくなるが、実際にはまったく成り立たない。単位円 $\gamma(t) = (\cos t, \sin t)$ を考えてみよう。$t = 2\pi$ で 1 周すると曲線は出発点に戻るため $\gamma(2\pi) - \gamma(0) = \mathbf{0}$ となるが、速度 $\gamma'(\xi)$ は単位長さを持ち、$\mathbf{0}$ となる ξ は一切存在しない。高次

元で生き残るのは、**平均値不等式**（mean value inequality）$\|\gamma(b)-\gamma(a)\| \le \sup_{\xi} \|\gamma'(\xi)\|\,|b-a|$ のみである（定理 5.19、Rudin (1976)）。幸い、34.2 節におけるテイラー展開の上界では、この定理のベクトル版は一切必要ない。スカラー値の損失を線分上に制限することで、通常の 1 変数の定理がそのまま適用できるからである。

34.8.2.2 行列値写像のヤコビ行列

34.4 節ではスカラー損失を微分するため、すべての勾配は都合よく対応する行列変数と同じ形状を持つ。しかし、行列値関数を行列入力に関して微分すると、完全な導関数は 4 つの添字を持つ配列になる。代数計算を線形で馴染みのある形に保つ標準的な方法は、列を縦に並べる作用素 vec で両方の行列を平坦化し、4 つの添字を持つテンソルを通常のヤコビ行列（Jacobian matrix）へと変換することである。この平坦化のもとでは、行列の全微分 $d\mathbf{F} = \mathbf{A}\,d\mathbf{X}\,\mathbf{B}$ は $d\,\mathrm{vec}\,\mathbf{F} = (\mathbf{B}^{\top} \otimes \mathbf{A})\,d\,\mathrm{vec}\,\mathbf{X}$ となり、連鎖律は単にクロネッカー積（Kronecker product）どうしの乗算になる。Mike B. Giles (2008) は、行列積、逆行列、行列式、および標準的な行列分解に対するフォワード・リバース両方向の微分規則を使いやすい一覧にまとめている。

34.8.3 自動微分

34.8.3.1 最適なヤコビ行列の累積

34.4 節で示したように、連鎖するヤコビ行列を掛ける順序を変えるだけで、計算コストはおよそ層の幅に相当する倍率だけ変化し得る。一直線に並んだ K 個の因子の積であれば、最も低コストな括弧の付け方を求める問題は古典的な連鎖行列積問題であり、動的計画法（dynamic programming）によって $O(K^3)$ 時間で解ける。しかし、分岐やスキップ接続を持つ一般の計算グラフ（computation graph）では、消去順序の探索空間がはるかに広がる。各中間ノードは、そこへ入る辺とそこから出る辺の上の偏微分を任意の順序で掛け合わせることで消去できるからである。乗算回数が最小となる消去順序を求める問題は、実は NP 完全であることが知られている (Naumann 2008)。そのため、深層学習フレームワークは消去順序の大域的な最適化を試みず、フォワードモード、リバースモード、およびそれらの合成を提供して、モードの選択をユーザーに委ねている。Griewank and Walther (2008) は、頂点・辺・面の消去を完全な一般性のもとで展開し、微分コンパイラで用いられるヒューリスティクスを概説している。

34.8.3.2 再帰的チェックポイント保存

34.5 節で扱った 1 段階のチェックポイント保存方式は、ピーク時の活性化メモリ（activation memory）を保存状態数 $T/s + s$ へと削減し、順伝播（forward pass）を 1 回追加するコストと引き換えに $2\sqrt{T}$（最適なブロックサイズ s のとき）まで抑える (Chen et al. 2016)。しかし、非常に長い系列や展開された反復ソルバでは、$O(\sqrt{T})$ のメモリでさえ大きすぎることがある。同じ考え方を再計算される各区間の内部に再帰的に適用すれば、メモリは一気に $O(\log T)$ まで下がり、必要な再計算は $O(T \log T)$ となる (Griewank 1992)。各ステップのコストが等しい場合、任意の固定されたメモリバジェットに対して Griewank の二項チェックポイント保存スケジュールが最適であることが証明されている (Griewank and Walther 2008)。実際には、チェックポイント保存のユーティリティは、指定されたブロックごとに 1 段階のチェックポイント保存を実装している。

JAX は `jax.checkpoint` を提供しており、これらのブロックを入れ子にすることで再帰的なスケジュールを手動で構築できる。

34.8.3.3 テイラーモード

k 階の方向微分(directional derivative)を求めるためにフォワードモードの自動微分(automatic differentiation)を k 回入れ子にするのは無駄が多い。入れ子を 1 段増やすごとに状態が倍増し、1 つの値につき 2^k 個の成分を保持することになるからである。**テイラーモード**は、二重数(dual number)を打ち切りべき級数 $a_0 + a_1\varepsilon + \cdots + a_k\varepsilon^k$(ただし $\varepsilon^{k+1} = 0$)に置き換えることで、この指数的な肥大化を回避する。各基本演算は 1 回のパスで $k+1$ 個すべての係数を伝播する。具体的には、乗算は 2 つの係数系列を $O(k^2)$ 回の演算で畳み込み、exp、log、sin などの超越関数も同じ $O(k^2)$ のコストの漸化式から計算できる (Griewank and Walther 2008)。34.5 節にある、1 回の順伝播で二階導関数を計算する演習問題は $k = 2$ の場合にあたる。

`jax.experimental.jet` は任意の次数 k に対するテイラーモードを実装している。

34.8.3.4 フォワード勾配

リバースモードの微分では中間値のテープを保存する必要がある一方、フォワードモードではテープが不要になる代わりに、入力パラメータごとに 1 回のパスが必要となる。Baydin et al. (2022) は、ガウス分布に従うランダムな接ベクトル $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ に沿って単一の順伝播を実行し、$(\nabla \ell^\top \mathbf{v})\,\mathbf{v}$ を確率的勾配の推定値とすることで、この両方のコストを回避した。$\mathbb{E}[\mathbf{v}\mathbf{v}^\top] = \mathbf{I}$ であるためこの推定量は不偏であり、パラメータ数が n のとき平均二乗誤差(mean squared error)は $(n+1)\|\nabla \ell\|^2$ となる。メモリ上のテープは完全に不要になるが、そのトレードオフは厳しい。典型的な推定誤差は真の勾配のノルムのおよそ $\sqrt{n}$ 倍に達し、小規模なモデルでは許容できても、現代のニューラルネットワークのスケールでは現実的でない大きさになる。

34.8.4 積分

34.8.4.1 ルベーグ積分

リーマン積分は定義域を区間に分割するが、この方針は激しく不連続な関数に対して破綻する。34.6 節で示したように、有理数の指示関数では代表点付きの和が決して一定の値に落ち着かず、さらにリーマン可積分な関数列の各点極限でさえリーマン可積分にとどまるとは限らない。Lebesgue の着眼点は、定義域ではなく値域をスライスすることにある。すなわち、各高さの水準を f が上回る集合の大きさを測り、それらの測度を足し合わせるのである。有理数全体や極座標における半直線 $\theta = 0$ のような測度ゼロの集合は積分値に一切寄与しないため、有理数の指示関数はすっきりと 0 に積分される。34.6 節と 34.7 節のいずれにおいても、この理論の主力となる 3 つの定理、すなわちフビニ・トネリの定理、n 次元の変数変換(change of variables)、および無限領域上の積分記号下の微分を証明なしで用いた。これら 3 つの定理はすべて、極限と積分をいつ交換できるかを定める単調収束定理(monotone convergence theorem)と優収束定理(dominated convergence theorem)に基づいている。$f_k \to f$ が各点収束であり、かつ $|f_k| \le g$ が単一の可積分関数 g に対して成り立つならば、$\int f_k \to \int f$ となる (Folland 1999)。

34.8.4.2 準モンテカルロ法

36.5.1.7 節では、誤差が $N^{-2/d}$ で減衰する二次のグリッドと、次元によらず誤差が $N^{-1/2}$ で減衰するモンテカルロ積分を比較した。しかし、純粋な無作為標本では偶発的な密集や空白の偏りが避けられない。**低食い違い量**(low-discrepancy)点集合(ソボル列、ハルトン列、格子則など)は、単位超立方体をはるかに均一に埋め尽くすように構成されており、有界変動(Hardy と Krause の意味で)の被積分関数に対して $O(N^{-1}(\log N)^d)$ の誤差を達成する。ただし、純粋に最悪ケースの保証として見ると、この上界は厳しい現実を突きつける。定数因子を無視しても、この上界が $N^{-1/2}$ を下回るのは、数千点を超えてから($d = 2$ のとき)、さらには 10^{15} 点を超えてから($d = 5$ のとき)に限られる。それでも実際には、準モンテカルロ法(quasi-Monte Carlo)は高次元においてさえ通常のモンテカルロ法をしばしば上回る。現実の被積分関数は、多くの場

合、一度に少数の有効な座標にしか強く依存しないからである。決定論的な点集合からはそのままではエラーバーが得られないため、**ランダム化準モンテカルロ法**（randomized quasi-Monte Carlo）では点集合にランダムなシフトやスクランブルを施し、独立な反復試行から不偏な分散推定値を得られるようにする。Owen (2013) の第 15〜17 章では、決定論的手法とランダム化手法の両方が解説されている。

34.8.4.3 スコア関数とパスワイズを超える勾配推定量

34.7 節では、スコア関数推定量とパスワイズ推定量を導出し、定数ベースラインを用いてスコア関数推定量の分散を低減した。より洗練された制御変量（control variate）は、モデル自体の構造を活用する。REBAR (Tucker et al. 2017) は、連続緩和（continuous relaxation）のパスワイズ勾配をスコア関数推定量から引き、その解析的な期待値を足し戻すことで、離散確率変数に対して推定量を不偏に保ったまま分散を下げる。Gumbel-softmax (Jang et al. 2017; Maddison et al. 2017) やストレートスルー推定器（straight-through estimator）(Bengio, Léonard, et al. 2013) のような連続緩和そのものは、元の離散的な目的関数に対する統計的バイアスが生じる代わりに低い分散を実現する。第 3 のアプローチである**測度値微分**（measure-valued derivative）は、確率密度の導関数を 2 つの密度のスケール付きの差として表し、両方からサンプリングすることで勾配を推定する。Mohamed et al. (2020) はこれら 3 つの系統を単一の枠組みに統合し、それぞれが適用できる条件を明示している。

34.8.5 書籍、講義、チュートリアル

書籍

- Deisenroth、Faisal、Ong による『Mathematics for Machine Learning』は、第 5 章（「Vector Calculus」）で機械学習に向けた勾配、ヤコビ行列、および連鎖律を扱っている（34.1 節、34.2 節）。
- Gilbert Strang による『Calculus』（MIT OpenCourseWare）は無料で公開されている教科書であり、積分、微分積分学の基本定理、および重積分に関する各章が 34.6 節の副読本となる。
- Petersen と Pedersen による『The Matrix Cookbook』は行列微分の恒等式をまとめた公式集であり、全微分法（34.4 節）で導出した結果を検算するのに役立つ。
- Griewank と Walther による『Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation』および Naumann による『The Art of Differentiating Computer Programs』は、フォワードモードとリバースモード、VJP と JVP、チェックポイント保存、および微分コンパイラに関する標準的な専門書である（34.5 節）。

講義とチュートリアル

- 3Blue1Brown による「Essence of Calculus」、および MIT OpenCourseWare の「Single Variable Calculus (18.01SC)」と「Multivariable Calculus (18.02SC)」。
- Edelman と Johnson による「Matrix Calculus for Machine Learning and Beyond (18.S096)」（MIT OCW）、Parr と Howard による「The Matrix Calculus You Need For Deep Learning」、および Laue、Mitterreiter、Giesen による matrixcalculus.org。
- Christopher Olah による「Calculus on Computational Graphs: Backpropagation」、Baydin らによる「Automatic Differentiation in Machine Learning: a Survey」、JAX ドキュメント内の「The Autodiff Cookbook」と「Autodidax」（34.5 節）、ならびにスコア関数（REINFORCE）推定量、パスワイズ再パラメータ化、制御変量を用いた分散低減（variance reduction）を扱う Mohamed らの「Monte Carlo Gradient Estimation in Machine Learning」（34.7 節）。

Chapter 35

最適化

本書に登場するすべてのニューラルネットワークは、同じ基本的なループで訓練される。すなわち、損失の勾配を計算し、その逆方向にステップを進め、これを繰り返す。第 9 章では、実際のネットワーク上でモメンタム、適応的ステップサイズ、学習率スケジュールがこのループにどう作用するかを実験的に確認したが、そこでは 3 つの疑問が未解決のまま残された。なぜ学習率には越えられない上限があるのか。その上限を十分に下回る安全なステップであっても、進行が極端に遅くなりうるのはなぜか。そして、なぜ勾配ノイズ（gradient noise）によって訓練が落ち着かなくなるのか。

最適化理論は、これらそれぞれの問いを目的関数の幾何構造と結びつける。最も急な方向の曲率が安全な最大ステップサイズを定め、最も急な曲率と最も緩やかな曲率の比が反復回数を定め、凸性（convexity）が停留点（stationary point）が大域的に最適かどうかを決定する。本章全体を通して、細長い鉢状の関数 $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ に繰り返し立ち返る。この関数はある方向がもう一方の方向より 1,000 倍急になっており、急な壁に沿って安全を保てるステップサイズでは、平坦な谷に沿った方向には反復点をわずかにしか動かせない。第 9 章が実践と計測に焦点を当てたのに対し、本章では自己完結した証明と背後にある幾何構造を与える。

本章は 3 部構成で展開する。まず厳密な勾配が得られる設定のもとで、35.1 節ではステップサイズの上限と反復回数を証明し、モメンタムによって曲率比がその平方根に置き換わる仕組みを示す。一方、35.3 節では停留点が大域的最小点となる条件を確立する。両者の間に位置する 35.2 節では、曲率を積極的にモデル化するニュートン法、準ニュートン法（quasi-Newton method）、信頼領域法（trust-region method）を扱う。次にノイズを含む勾配に目を転じ、35.4 節では固定ステップの SGD がステップサイズに比例するノイズフロアで停滞する理由を示し、35.5 節では座標ごとのスケーリングと AdamW をヘビーボールモメンタムと比較検証する。最後に、35.6 節と 35.7 節ではノルムバジェット（2.7 節）の背後にある KKT 条件、射影、近接ステップ、および双対性（duality）を展開し、35.8 節では浮動小数点精度、桁落ち、および条件性（conditioning）を解析する（5.5 節）。

本章では、勾配とヘッセ行列（34.2 節）、および固有値分解（33.3 節）に関する知識を前提とする。深層ネットワークの損失は大域的に滑らかでも凸でもないため、本章の定理は局所的なモデルとして機能するものであり、その適用限界は各節で明示する。なお、内点法（interior-point method）と作用素分割法（operator-splitting method）については 35.9 節で概観する。

記法

本章全体を通して、Table 35.1 にまとめた記法の慣例に従う。各節ではその節で頻繁に用いる記号を改めて示し、単一の節に固有の記法は初出箇所で導入する。

Table 35.1: 最適化の章で用いる記号。星印 $\star$ は最適点・最適値を、アスタリスク $*$ は凸共役を表す。行列は太字とする。

記号	意味
f、$\ell(\mathbf{w})$	目的関数、重みの関数としての訓練損失
$\mathbf{x}^\star$、$f^\star$	最小点と最小値
∇f、$\nabla^2 f$、$\mathbf{g}_k$	勾配、ヘッセ行列、ステップ k で用いる勾配（または確率的勾配）
L、μ	滑らかさの定数と強凸性の定数（$\mu\mathbf{I} \preceq \nabla^2 f \preceq L\mathbf{I}$）
$\kappa = L/\mu$、$\kappa(\mathbf{A})$	問題の条件数、行列の条件数 $\sigma_{\max}(\mathbf{A})/\sigma_{\min}(\mathbf{A})$
λ_i、$\lambda_{\max}(\mathbf{A})$	ヘッセ行列または二次関数の行列 $\mathbf{A}$ の固有値
η、β	ステップサイズ（学習率）、モメンタム係数
k、t	反復の添字、適応的手法におけるオプティマイザのステップ
ϵ	目標精度、Adam において除算を有限に保つ定数
n、d、$\mathcal{B}$、b	データ例の数、変数の数、ミニバッチとそのサイズ
Σ	各データ例の勾配の共分散
$\mathbf{m}_t$、$\mathbf{v}_t$、β_1、β_2	Adam の一次・二次モーメント推定値、およびそれらの減衰率
$\mathbf{P}$	前処理行列（更新ステップは $-\eta\,\mathbf{P}^{-1}\mathbf{g}$）
g_i、h_j	不等式制約 $g_i(\mathbf{x}) \le 0$、等式制約 $h_j(\mathbf{x}) = 0$
λ、ν	不等式制約と等式制約の乗数（λ は減衰項 $\frac{\lambda}{2}\lVert\mathbf{w}\rVert^2$ の重みも表す）
$\mathcal{L}$、q	ラグランジアン、双対関数
$p^\star$、$d^\star$	主問題と双対問題の最適値
f^*	f の凸共役
Π_C、$\operatorname{prox}_{\psi}$	集合 C へのユークリッド射影、ψ の近接作用素
u、$\varepsilon_{\text{mach}}$	浮動小数点形式の丸めの単位と計算機イプシロン（$u = \varepsilon_{\text{mach}}/2$）

35.1 勾配降下法と曲率

実践において、学習率（learning rate）は試行錯誤で決められるのが普通である。適当な値を選び、損失の推移を観察し、訓練が発散すれば値を下げる。各ステップは損失の勾配と逆向きに進むため、勾配が方向を定め、学習率は進む距離だけを決める。しかし、この距離の選び方を誤ると、訓練は極端に遅くなるか、あるいは不安定になる。勾配に沿ったステップはどこまで信頼できるのか。そしてその限界によって、どれだけの反復回数が強いられるのだろうか。

わずか 2 変数の関数でも、この難しさはすでに露わになる。楕円状のボウル $f(x, y) = \frac{1}{2}(x^2 + 10\,y^2)$ を考えよう。この関数は y 方向に沿って x 方向より 10 倍急峻に曲がっている。初期点を $(2, 1)$ とすると、そこでは $f = 7$、$\nabla f = (2, 10)$ である。勾配と逆向きにサイズ η のステップを進むと、到達点は $(2 - 2\eta,\, 1 - 10\eta)$ となる。$\eta = 0.01$ では損失はほとんど動かず、6.01 に下がるにとどまる。$\eta = 0.1$ では 1.62 まで一気に減少する。ところが $\eta = 0.25$ では、逆に 12.4 へと**上昇する**。これらのステップはすべて下り坂を向いていたにもかかわらず、ステップ長を変えただけでこの違いが生じる。

原因は曲率（curvature）にある。急峻な y 軸に沿って安全を保てるほど短いステップは、平坦な x 軸に沿っては這うようにしか進まない。最良の固定学習率を選んでも、69 回の反復を費やしてようやく誤差が 10^6 分の 1 に縮まる（6 桁の精度に相当する）。曲率の比を 1000 に広げると（本章の通し例である $\tfrac{1}{2}(x_1^2 + 1000\,x_2^2)$）、勾配降下法（gradient descent）は $6{,}908$ ステップを要する。ここにモメンタムを加えれば、その回数は 315 まで減る。以下では、ヘッセ行列が $\mathrm{diag}(1, 10)$ でどの数値も検算しやすい緩やかなボウルを手計算で扱い、より硬い通し例については対応するステップ数を併記する。

解析は 3 段階で進める。第 1 に、滑らかさ（smoothness）を定式化して 34.2 節の局所線形近似を有限長のステップに対する保証へと引き上げ、降下補題（descent lemma）Equation 35.5 を導く。第 2 に、二次関数上で勾配降下法を厳密に解き、曲率の比が条件数（condition number）として現れて最適ステップ Equation 35.10 を定めることを見る。最後にモメンタムを導入し、その $\sqrt{\kappa}$ 収束則 Equation 35.15 を導出する。本節を通して扱う手法はいずれも曲率を尊重するにとどまり、曲率そのものをモデル化することはしない。ニュートン法のように曲率を直接モデル化する手法は 35.2 節で扱う。また、本節では厳密な勾配が得られるものと仮定し、ミニバッチ訓練におけるノイズを含んだ勾配は 35.4 節に譲る。以下のコードはすべて素の NumPy で書かれているため、各更新式を数式と直接照らし合わせて読むことができる。

35.1.1 降下方向と滑らかさ

すべての勾配ステップは、どの方向へ進むか、そしてどこまで進むかという 2 つの問いに答える必要がある。第 1 の問いには、一次のテイラー展開から直ちに答えが出る。第 2 の問いに答えるには、勾配そのものがどれほど速く変化しうるかを知る必要がある。この仮定こそが、局所的な線形近似を有限長のステップに対する保証へと変える。

35.1.1.1 最急降下方向

冒頭の例では、他に優れた方向がないか問うことなく $-\nabla f$ に沿って進んだ。なぜ負の勾配が自然な選択なのかを見るため、$f:\mathbb{R}^d\to\mathbb{R}$ を一次まで展開しよう。小さなステップサイズ $\eta>0$ で方向 $\mathbf{d}$ に進むと、関数値は次のように変化する。

$$f(\mathbf{x}+\eta\,\mathbf{d}) = f(\mathbf{x}) + \eta\,\nabla f(\mathbf{x})^{\top}\mathbf{d} + o(\eta).$$

一次近似の範囲では、変化量は単に η と方向微分（directional derivative）$\nabla f(\mathbf{x})^{\top}\mathbf{d}$ の積である。ベクトル $\mathbf{d}$ は、点 $\mathbf{x}$ において $\nabla f(\mathbf{x})^{\top}\mathbf{d}<0$ を満たすとき、その点における**降下方向**（descent direction）と呼ばれる。

命題（降下方向と最急降下方向）。 *f は点 $\mathbf{x}$ で微分可能であり、$\nabla f(\mathbf{x})\neq\mathbf{0}$ を満たすとする。*

1. *$\nabla f(\mathbf{x})^{\top}\mathbf{d}<0$ ならば、$f(\mathbf{x}+\eta\,\mathbf{d})<f(\mathbf{x})$ が十分小さなすべての $\eta>0$ に対して成り立つ。*
2. *すべての単位ベクトルの中で、方向微分を最小化する方向は正規化された負の勾配に一意に定まる。*

$$\min_{\|\mathbf{d}\|=1} \nabla f(\mathbf{x})^{\top}\mathbf{d} = -\|\nabla f(\mathbf{x})\|, \qquad \text{attained only at } \mathbf{d} = -\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}. \tag{35.1}$$

証明。 第 1 の主張を示すため、$\varphi(\eta)=f(\mathbf{x}+\eta\,\mathbf{d})$ とおく。連鎖律により $\varphi'(0)=\nabla f(\mathbf{x})^{\top}\mathbf{d}<0$ であるから、差分商 $(\varphi(\eta)-\varphi(0))/\eta$ は十分小さなすべての $\eta>0$ に対して真に負となり、し

たがって $\varphi(\eta) < \varphi(0)$ が従う。第 2 の主張については、コーシー・シュワルツの不等式より $\nabla f(\mathbf{x})^{\top}\mathbf{d} \geq -\|\nabla f(\mathbf{x})\|\,\|\mathbf{d}\| = -\|\nabla f(\mathbf{x})\|$ がすべての単位ベクトル $\mathbf{d}$ に対して成り立ち、等号が成立するのは $\mathbf{d}$ が $\nabla f(\mathbf{x})$ の負のスカラー倍であるとき、かつそのときに限る。そのような単位ベクトルは $-\nabla f(\mathbf{x})/\|\nabla f(\mathbf{x})\|$ に一意に定まり、これが下界を達成する。■

冒頭のボウルでは、$(2,1)$ における最急降下の単位方向ベクトルは $-(2,10)/\sqrt{104} = (-0.196,-0.981)$ であり、その傾きは $-\|\nabla f\| = -10.198$ となる。座標軸方向 $(-1,0)$ も下り坂を向いているが、傾きは -2 とずっと緩やかである。実際、降下方向は $-\nabla f$ と鋭角をなすベクトル全体の開半空間を占めており、負の勾配はその中で最も急な方向を選び出しているにすぎない。この結果はあくまで傾き（無限小ステップの極限）を記述するものであるため、次に 4 つの単位方向について、小さな有限ステップ $\eta = 0.01$ における実際の減少量をこれらの傾きがどれほど忠実に予測するか確かめよう。

```
def f(v):
    return 0.5 * (v[0]**2 + 10 * v[1]**2)

def grad(v):
    return np.array([v[0], 10 * v[1]])

x = np.array([2.0, 1.0])
g = grad(x)
steepest = -g / np.linalg.norm(g)
candidates = [('steepest -grad/|grad|', steepest),
              ('diagonal -(1,1)/sqrt(2)', -np.ones(2) / np.sqrt(2)),
              ('axis     (-1,0)', np.array([-1.0, 0.0])),
              ('uphill   +(1,1)/sqrt(2)', np.ones(2) / np.sqrt(2))]
eta = 0.01
for name, d in candidates:
    print(f'{name}:  slope = {g @ d:+8.4f},  '
          f'f(x + eta d) - f(x) = {f(x + eta * d) - f(x):+.6f}')
```

```
steepest -grad/|grad|:  slope = -10.1980,  f(x + eta d) - f(x) = -0.101498
diagonal -(1,1)/sqrt(2):  slope =  -8.4853,  f(x + eta d) - f(x) = -0.084578
axis     (-1,0):  slope =  -2.0000,  f(x + eta d) - f(x) = -0.019950
uphill   +(1,1)/sqrt(2):  slope =  +8.4853,  f(x + eta d) - f(x) = +0.085128
```

$\eta = 0.01$ では、どの方向でも実際の変化量が η と方向傾きの積に 1% 以内で一致しており、短いステップに対しては一次モデルが信頼できる指針となる。微分可能性が保証するのはそこまで、すなわち十分小さなステップで関数値が減少することだけである。では「十分小さい」とは具体的にどれほど小さいのか。次にこの問いへ進もう。

35.1.1.2 滑らかさと降下補題

勾配降下法は、固定された**ステップサイズ**（または学習率）$\eta > 0$ を用いて最急降下ステップを反復する。

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\nabla f(\mathbf{x}_k). \tag{35.2}$$

本節全体を通して、k は反復の添字を表し、$\mathbf{g}_k = \nabla f(\mathbf{x}_k)$ と略記する。一次のテイラー展開は方向 $-\mathbf{g}_k$ の妥当性を示したが、それをどこまで遠くまで信頼してよいかについては何も語らない。反復点が移動して勾配が大きく変化してしまえば、もはや $\mathbf{g}_k$ は f を適切に近似しない。有限のステップサイズに保証を与えるには、勾配が変化しうる速さを上から抑える必要がある。

定義（L 滑らかさ）。微分可能な関数 f が **L 滑らか**であるとは、単位距離あたりの勾配の変化量が高々 L に抑えられることをいう。

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \;\le\; L\,\|\mathbf{x}-\mathbf{y}\| \qquad \text{for all } \mathbf{x},\mathbf{y}. \tag{35.3}$$

f が 2 回連続微分可能であるとき、L 滑らかであることは、ヘッセ行列のすべての固有値が $[-L, L]$ に収まることと同値である。固有値が有界であれば Equation 35.3 が成り立つことは、ヘッセ行列を $\mathbf{y}$ から $\mathbf{x}$ への線分に沿って積分すればわかり、逆は差分商の極限をとることで従う（演習問題 2a）。要するに、L はあらゆる方向の曲率を上から抑える定数である。冒頭のボウルは定数のヘッセ行列 $\operatorname{diag}(1, 10)$ を持つため、$L = 10$ である。対照的に、曲率が際限なく増大する関数は大域的な滑らかさの定数を持たない。たとえば $\tfrac{1}{4}x^4$ は $f''(x) = 3x^2$ となるため、有限な大域的定数 L は存在しない。そのような場合は、後ほどバックトラッキング直線探索（backtracking line search）を用いて対処する。

滑らかさは、局所的な接平面を大域的な上界、すなわち二次上界（quadratic ceiling）へと引き上げる。

$$f(\mathbf{y}) \;\le\; f(\mathbf{x}) + \nabla f(\mathbf{x})^{\top}(\mathbf{y}-\mathbf{x}) + \tfrac{L}{2}\,\|\mathbf{y}-\mathbf{x}\|^2. \tag{35.4}$$

幾何学的には、f のグラフは点 $\mathbf{x}$ における接平面から、距離の二乗の $\tfrac{L}{2}$ 倍を超えて上方に外れることは決してない。この不等式を証明する前に、勾配ステップ $\mathbf{y} = \mathbf{x} - \eta\,\mathbf{g}$ を代入して 2 つの項の競合を見てみよう。一次の線形減少量 $\eta\,\|\mathbf{g}\|^2$ は η に比例して大きくなるのに対し、二次の曲率ペナルティ $\tfrac{L}{2}\eta^2\|\mathbf{g}\|^2$ は η^2 に比例して増大する。小さな η では線形減少が勝るが、$\eta = 2/L$ で両項はちょうど打ち消し合う。

命題（降下補題）。 *f が L 滑らかであるならば、勾配ステップ $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\nabla f(\mathbf{x}_k)$ Equation 35.2 は次式を満たす。*

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \eta\left(1 - \tfrac{L\eta}{2}\right)\|\mathbf{g}_k\|^2. \tag{35.5}$$

特に、f は $0 < \eta < 2/L$ かつ $\mathbf{g}_k \neq \mathbf{0}$ である限り真に減少する。保証される減少量は $\eta = 1/L$ で最大となり、そのとき $f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) - \tfrac{1}{2L}\|\mathbf{g}_k\|^2$ が成り立つ。

証明。まず二次上界 Equation 35.4 を示す。f を $\mathbf{x}$ と $\mathbf{y}$ を結ぶ線分上に制限して $\psi(t) = f(\mathbf{x} + t(\mathbf{y}-\mathbf{x}))$（$t \in [0,1]$）とおくと、$\psi'(t) = \nabla f(\mathbf{x}+t(\mathbf{y}-\mathbf{x}))^{\top}(\mathbf{y}-\mathbf{x})$ となる。微分積分学の基本定理、コーシー・シュワルツの不等式、およびリプシッツ条件 Equation 35.3 を適用すると、次式が得られる。

$$\begin{aligned} f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^{\top}(\mathbf{y}-\mathbf{x}) &= \int_0^1 \big(\nabla f(\mathbf{x}+t(\mathbf{y}-\mathbf{x})) - \nabla f(\mathbf{x})\big)^{\top}(\mathbf{y}-\mathbf{x})\,dt \\ &\le \int_0^1 L\,t\,\|\mathbf{y}-\mathbf{x}\|^2\,dt = \tfrac{L}{2}\|\mathbf{y}-\mathbf{x}\|^2. \end{aligned}$$

ここに勾配ステップ $\mathbf{y} = \mathbf{x}_k - \eta\,\mathbf{g}_k$ を代入すると、次式を得る。

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \eta\,\|\mathbf{g}_k\|^2 + \tfrac{L}{2}\,\eta^2\,\|\mathbf{g}_k\|^2.$$

係数 $\eta(1 - L\eta/2)$ が真に正となるのは $0 < \eta < 2/L$ のとき、かつそのときに限り、$\eta = 1/L$ で最大値をとる。■

なお、この証明は $\mathbf{x}_k$ と $\mathbf{x}_{k+1}$ への線分上の各点との間で勾配を比較しているだけなので、∇f がその線分上で L リプシッツ連続であれば十分である。冒頭のボウル（$\|\mathbf{g}\|^2 = 104$、$L = 10$）でこの補題を試してみよう。$\eta = 1/L = 0.1$ では、上界により $f \le 7 - 104/20 = 1.8$ が保証される。$\eta = 0.01$ では $f \le 7 - 104 \cdot 0.01 \cdot 0.95 = 6.012$ が保証される。$\eta = 0.25 > 2/L$ では因子 $1 - L\eta/2$ が負に転じ、上界が初期値を上回ってしまうため、もはや何の保証も与えない。これら 3 つのステップが実際に到達する関数値と比べると、これらの上界はどれほどタイトだろうか。

```
x, L = np.array([2.0, 1.0]), 10.0
g = grad(x)
for eta in (0.01, 0.1, 0.25):
    bound = f(x) - eta * (1 - L * eta / 2) * (g @ g)
    print(f'eta = {eta:4.2f}:  bound {bound:7.4f},  actual {f(x - eta * g):7.4f}')
```

```
eta = 0.01:  bound  6.0120,  actual  6.0102
eta = 0.10:  bound  1.8000,  actual  1.6200
eta = 0.25:  bound 13.5000,  actual 12.3750
```

安定な 2 つのステップでは、上界は驚くほどタイトである。6.0102 は 6.012 のすぐ下にあり、1.62 も 1.8 をきれいに下回っている。$\eta = 0.25$ では上界 13.5 は初期値 $f = 7$ よりはるかに大きく、実際このステップはオーバーシュートして 12.375 に達する。Figure 35.1 に両曲線を η の関数としてプロットした。探索方向に沿っては $\mathbf{g}^\top \operatorname{diag}(1, 10)\,\mathbf{g} = 1004$ であるから、真の目的関数は放物線 $7 - 104\,\eta + 502\,\eta^2$ となる一方、二次上界は $7 - 104\,\eta + 520\,\eta^2$ となる。2 つの放物線がこれほど密着しているのは、この勾配がほぼ硬い軸に沿っており、その方向の曲率が L に極めて近いためである。

要するに、ステップ $\eta = 1/L$ は二次上界を最小化し、f はその上界の下に収まることが保証される。34.1 節では、これと同じ幾何学的議論を 1 変数で展開している。

35.1.1.3 凸性を仮定しない停留点

驚くべきことに、凸性（convexity）を一切仮定しなくても、降下補題だけで長期的な反復に定量的な制約が課される。すなわち、勾配がいつまでも大きいままでいることはできない。

命題（勾配降下法は近似停留点を見つける）。 *f は L 滑らかであり、下から f_{low} で抑えられるとし、Equation 35.2 を $\eta = 1/L$ で実行する。このとき、すべての $K \ge 1$ に対して次式が成り立つ。*

$$\min_{0 \le k < K} \|\nabla f(\mathbf{x}_k)\|^2 \;\le\; \frac{2L\,(f(\mathbf{x}_0) - f_{\text{low}})}{K}. \tag{35.6}$$

証明。 $\eta = 1/L$ のとき、降下補題を変形すると $\|\mathbf{g}_k\|^2 \le 2L\,(f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}))$ となる。$k = 0, \dots, K-1$ にわたって和をとると、右辺は望遠鏡和となって相殺し、$2L\,(f(\mathbf{x}_0) - f(\mathbf{x}_K)) \le 2L\,(f(\mathbf{x}_0) - f_{\text{low}})$ となる。K 個の数の最小値はその平均以下であるから、両辺を K で割れば求める上界が得られる。■

したがって、最初の K 個の反復点のうち少なくとも 1 つは勾配ノルム（gradient norm）の二乗が $O(1/K)$ となり、これには f の下界と滑らかさの定数しか必要としない。ただし、この結果

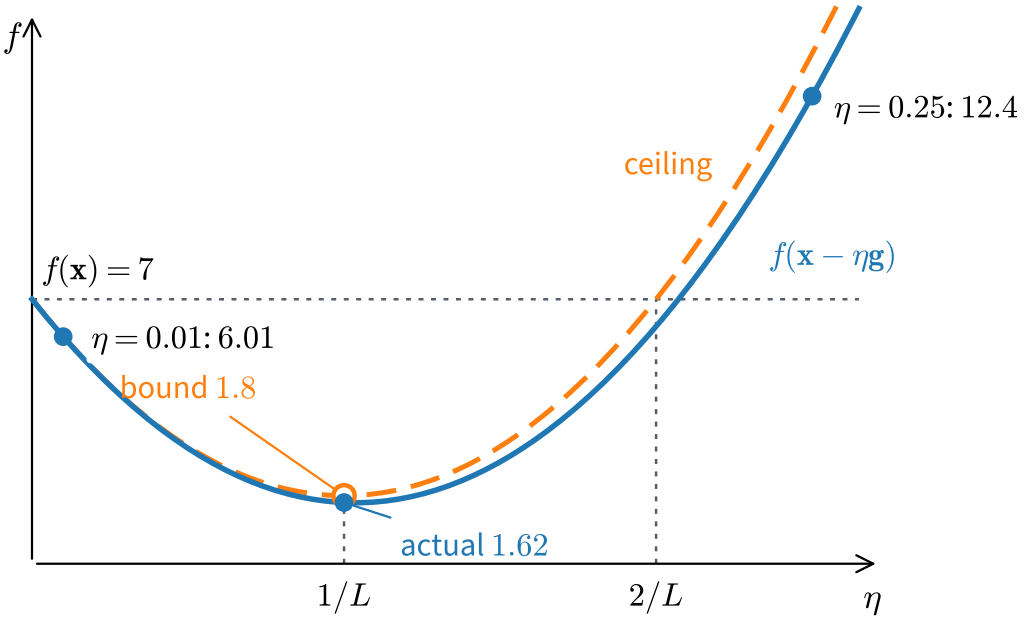


Figure 35.1: 冒頭の例における降下補題。$-\mathbf{g}$ に沿って $(2, 1)$ から進むとき、真の目的関数は下の放物線、二次上界 $f - \eta\|\mathbf{g}\|^2 + \frac{L}{2}\eta^2\|\mathbf{g}\|^2$ は上の放物線であり、$\eta = 0$ で互いに接する。上界を $\eta = 1/L = 0.1$ で最小化すると $f \le 1.8$ が保証され、実際のステップは 1.62 に達する。$2/L = 0.2$ を超えると上界は初期値を上回り何も保証しなくなる。$\eta = 0.25$ では目的関数は 12.4 まで上昇する。

は大域的な L の存在を仮定している点に留意しておこう。ReLU ネットワークは滑らかではないため、そこではこの上界は形式的な保証ではなく理想化されたモデルとして機能する。

しかし、勾配が小さいからといって最小点であるとは限らない。関数 $f(x, y) = \frac{1}{2}x^2 + \cos y$ を考えよう。この関数は下から -1 で抑えられ、かつ 1 滑らかである（ヘッセ行列 $\mathrm{diag}(1, -\cos y)$ の固有値が $[-1, 1]$ に収まる）。勾配降下法を $\eta = 1$ として $(1, 0)$ から開始すると、1 ステップで x は 0 に到達し、y は一切変化しない。というのも、$\partial f/\partial y = -\sin y$ が $y = 0$ でゼロになるからである。アルゴリズムは $(0, 0)$ で $f = 1$ のまま停止してしまう。ここは f が x 方向に上向き、y 方向に下向きに曲がる鞍点（saddle point）であり、真の最小値 -1（点 $(0, \pi)$ で達成）からは程遠い。鞍点でも勾配はゼロになるため定理自体は成り立っているが、到達したのがどの種類の停留点（stationary point）なのかまでは判別できない。この停滞自体は初期点の偶然によるものである。代わりに $(1, y_0)$（任意の小さな $y_0 \neq 0$）から開始すれば、各ステップ $y \leftarrow y + \eta \sin y$ によって y は 0 から離れていき、やがて最小点に落ち着く。一般に、負の曲率を持つ方向を備えた鞍点へ勾配降下法が収束するような初期点の集合は測度 0 であるから（9.1.2 節）、ランダム初期化（random initialization）を用いれば鞍点への収束は避けられる（ただし反復点がその近傍に長時間滞留することはある）。凸目的関数（convex objective）であれば、この問題はそもそも生じない。すべての停留点が自動的に大域的最小点（global minimum）となるからである（35.3 節）。

35.1.1.4 バックトラッキング直線探索

$\eta = 1/L$ を使うには、L の値を知っている必要がある。しかし実際にはそれが既知であることはまれであり、仮に大域的な L が存在したとしても、局所的な曲率は損失ランドスケープ（loss landscape）の場所によって変化するため、単一の大域定数では保守的になりすぎる。**直線探索**（line search）は、実行時にステップサイズを適応的に調整する手法である。候補ステップ η が与えられたとき、実際の減少量が一次モデルの予測する減少量の少なくとも一定の割合 $c \in (0, 1)$ に達した場合に限りそのステップを受理する。これが **Armijo 条件**（十分減少条件、sufficient decrease condition）である (Armijo 1966)。

$$f\big(\mathbf{x} - \eta \nabla f(\mathbf{x})\big) \;\le\; f(\mathbf{x}) - c\,\eta \,\|\nabla f(\mathbf{x})\|^2. \tag{35.7}$$

右辺は、テイラー線形予測の傾きを c 倍に緩めたものにすぎない。**バックトラッキング**は楽観的な試行ステップ η_0 から出発し、η を繰り返し半分にしながら Equation 35.7 が満たされるまで試す。降下補題により、$1 - L\eta/2 \ge c$、すなわち $\eta \le 2(1-c)/L$ となった時点でこの判定は必ず通過する。したがってループは常に有限回の半減で終了し、理論上のしきい値の 2 分の 1 以上にあたるステップ $\eta \ge \min\big(\eta_0, (1-c)/L\big)$ を、事前に L を一切知ることなく返す。

さらに都合がよいことに、この保証には反復点が実際に訪れる領域での L さえあれば足りる。四次関数 $f(x) = \frac{1}{4}x^4$（大域的な L を持たない）を考え、$x_0 = 3$、$c = \frac{1}{2}$ から開始しよう。バックトラッキングは f を決して増加させないため、すべての反復点は $[-3, 3]$ にとどまり、そこでは $f''(x) = 3x^2 \le 27$ が成り立つ。$\eta \le 1/27$ の試行ステップは x を $x(1 - \eta x^2)$ へ移すが、この点は $\frac{2}{3}x$ と x の間に位置するため線分全体が $[-3, 3]$ の内部に収まり、局所的には $L = 27$ で十分となる。この四次関数の曲率は原点から離れると急峻になり、ゼロ付近では消えるため、軌跡全体に適した単一の固定ステップサイズは存在しない。$x_0 = 3$ から両戦略を比較してみよう。一方は固定ステップ $\eta = 0.3$（原点付近であれば完全に安全な値）、もう一方は $\eta_0 = 1$ から始めるバックトラッキングである。

```
quartic = lambda v: 0.25 * (v ** 4).sum()
quartic_grad = lambda v: v ** 3
```

```
x = np.array([3.0])
for _ in range(6):                              # fixed step, too large at x0
    x = x - 0.3 * quartic_grad(x)
print(f'fixed eta = 0.3 from x0 = 3:  |x_6| = {abs(x[0]):.2e}  (diverged)')

x, etas = np.array([3.0]), []
for _ in range(25):                             # backtracking line search
    g = quartic_grad(x)
    eta = 1.0
    while quartic(x - eta * g) > quartic(x) - 0.5 * eta * (g * g).sum():
        eta *= 0.5                              # Armijo fails: halve the step
    x = x - eta * g
    etas.append(eta)
print('accepted steps:', np.array(etas[:8]), '...')
print(f'backtracking from x0 = 3:  f(x_25) = {quartic(x):.2e}  (monotone descent)')
```

```
fixed eta = 0.3 from x0 = 3:  |x_6| = 6.52e+103  (diverged)
accepted steps: [0.03125 0.0625  0.125   0.25    0.5     1.      1.      1.     ] ...
backtracking from x0 = 3:  f(x_25) = 1.09e-04  (monotone descent)
```

固定ステップは直ちに爆発し、わずか 6 回の反復で $|x_6| \approx 6.5 \times 10^{103}$ に達する。更新 $x \mapsto x(1-\eta x^2)$ が $|x|$ を縮小させるのは $\eta x^2 < 2$ のとき、かつそのときに限るため、$x_0 = 3$ からの固定ステップは $\eta < 2/9 \approx 0.22$ で収束し、それを超えると発散する。この安定性のしきい値が $2/9$ であり、$2/f''(x_0) = 2/27$ ではないことに注目しよう。$2/f''(x_0)$ という規則は二次関数のものである。原点へ向かうステップが受けるのは、f'' の 0 から x までの平均値（すなわち x^2）であり、最大値 $3x^2$ ではない。降下補題は線分上の最悪ケースの曲率に備えなければならないため、真の安全範囲の 3 分の 1 しか保証できないのである。

対照的に、バックトラッキングは最初のステップで 0.031 を受理する。その直前の試行値 0.0625 は Armijo 条件を満たさない。この四次関数において Armijo 条件が成り立つのは $\eta x^2 \le 0.456$（おおよそ $\eta \le 0.05$、$x_0 = 3$ のとき）のとき、かつそのときに限るからである。反復点が原点に近づいて曲率が平坦になるにつれ、直線探索がその場その場の局所曲率を探り当てるため、受理されるステップは自動的に大きくなっていく（$0.0625, 0.125, 0.25, 0.5, 1.0$ と増え、いずれも $0.3/x^2$ に近い）。もっとも、深層学習の訓練でバックトラッキングが使われることはほとんどない。ミニバッチ損失はノイズが大きすぎて 2 つの候補ステップを信頼性よく比較できないうえ、試行のたびに余分な順伝播のコストがかかるからである。代わりに深層学習の訓練では、通常スケジュールによってステップサイズを事前に指定する（9.7 節）。

35.1.1.5 緩和された滑らかさと勾配クリッピング

スケジュールでは、反復点が高曲率の領域に入ったときに反応できない。そのために訓練で用いられる安全策が勾配クリッピング（gradient clipping）であり、先ほどの四次関数はその仕組みをよく示している。曲率 $f''(x) = 3x^2$ は非有界だが、それが大きくなるのは勾配 $f'(x) = x^3$ も大きい場所に限られる。実際、$3x^2 \le 4 + |x|^3$ がすべての x に対して成り立つ。2 回微分可能な関数は、その曲率がこのように勾配によって抑えられるとき、(L_0, L_1) **滑らか**であるという (Zhang, He, et al. 2020)。

$$\|\nabla^2 f(\mathbf{x})\| \;\le\; L_0 + L_1 \|\nabla f(\mathbf{x})\| \qquad \text{for all } \mathbf{x}, \tag{35.8}$$

ここで左辺のノルムは、ヘッセ行列の固有値の絶対値の最大値を表す。$L_1 = 0$ とおけば L 滑らかさに戻り、四次関数は $L_0 = 4$ および $L_1 = 1$ でこの条件を満たす。この条件のもとで降下補

題が保証できるステップは、もはや定数 $1/L$ ではない。オーダー $1/(L_0 + L_1\|\mathbf{g}\|)$、すなわち勾配が小さいところでは一定のステップサイズ、勾配が大きいところでは一定の**距離** $\eta\,\|\mathbf{g}\|$（オーダー $1/L_1$）となる。$\mathbf{g}$ を $\min(1, c/\|\mathbf{g}\|)\,\mathbf{g}$（しきい値 c）に置き換える勾配クリッピングは、追加コストなしでこの形のステップを実現する。四次関数で $c = 1$ とすると、先ほど発散したステップ $\eta = 0.3$ でも、反復ごとに x を 0.3 ずつ動かして $|x| \le 1$ に到達させ、そこから収束する（演習問題 3）。

35.1.2 二次モデルと条件数

冒頭では、一方のボウルでは 69 ステップで済むのに、もう一方ではなぜ 6,908 ステップもかかるのかを説明すると約束した。どちらの目的関数も二次関数であり、二次関数上では勾配降下法を閉じた形で解くことができる。驚くべきことに、反復回数はたった 1 つの量、すなわち最大曲率と最小曲率の比に帰着する。

35.1.2.1 悪条件な谷

代数計算に入る前に、2 つの二次関数における勾配降下法を対比した Figure 35.2 を見てみよう。丸いボウル上では、反復点は最小点へ向かってほぼ一直線に進む。一方、細長い谷では壁が急峻で谷底が平坦になっている。壁が課す安定性の上限に近いステップをとると、谷を飛び越えて壁から壁へと跳ね返るばかりで、谷底に沿ってはほんの少しずつしか進まない。その結果、典型的なジグザグの軌跡が生じる。（図の谷は $\mathrm{diag}(1, 20)$ に対して $\eta = 0.097$ と、安定性の上限 0.1 のすぐ下に設定したものであり、キャプションに記した縮小率が生じる。）

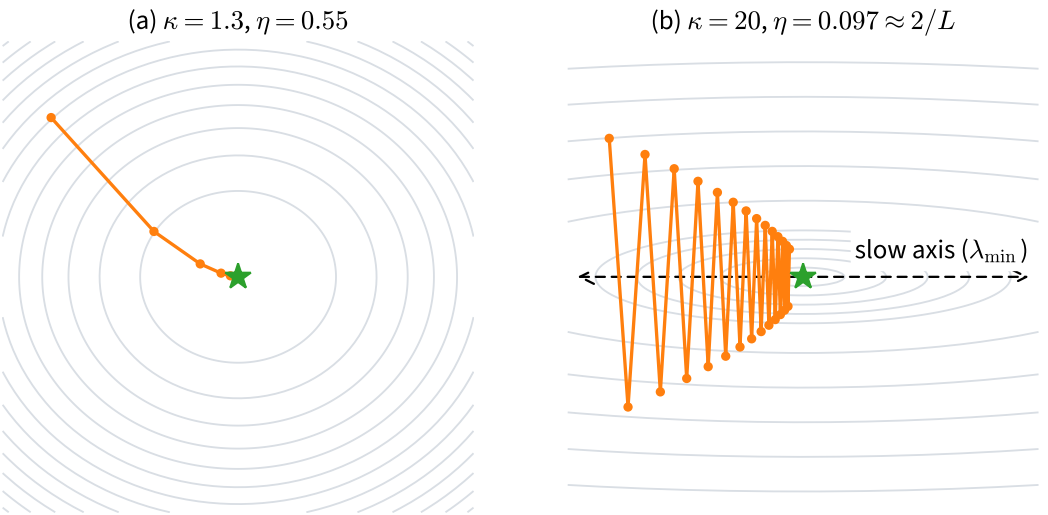


Figure 35.2: 2 つの二次関数 $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ における勾配降下法。左：良条件なボウルでは、反復点は最小点へほぼ一直線に向かう。右：悪条件な谷では、安定性の上限に近いステップをとると急峻なモードがオーバーシュートして符号を反転させ（因子 -0.94）、遅い軸はステップごとに 0.90 しか縮小しない。

二次関数を徹底的に解析する価値があるのは、あらゆる滑らかな関数が狭義極小点の近傍では二次関数のように振る舞うからである。最小点（minimizer）$\mathbf{x}^\star$ において正定値（positive definite）のヘッセ行列 $\mathbf{A}$ を持つとき、二次のテイラー展開は f を $f(\mathbf{x}^\star) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^\star)^\top \mathbf{A}\,(\mathbf{x} - \mathbf{x}^\star)$ で近似する。最小点を原点へ平行移動し、定数項を除けば、次のモデル問題が得られる。

$f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^{\top}\mathbf{A}\,\mathbf{x}$,

$\mathbf{A}$ symmetric positive definite, with eigenvalues $0 < \lambda_{\min} = \lambda_1 \le \cdots \le \lambda_d = \lambda_{\max}$.

その勾配は $\mathbf{A}\mathbf{x}$、ヘッセ行列は $\mathbf{A}$ であるから、滑らかさの定数は単に $L = \lambda_{\max}$ となる。最大曲率と最小曲率の比を次式で定義する。

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}},$$

これがこの問題の**条件数**である。冒頭のボウルでは 10（$\mathrm{diag}(1, 10)$）、本章の通し例では 10^3 となる。$\mathbf{A}$ は対称正定値行列であるため、κ は特異値の比 $\kappa(\mathbf{A})$（33.5.3.2 節で導入）と一致する。

35.1.2.2 モードごとの縮小と安定性の上限

Figure 35.2 のジグザグ挙動は、最急降下方向が課すステップサイズの上限から直接生じている。その上限を正確に特定し、それが平坦な方向の進行をどれほど遅らせるかを見るため、二次関数上の勾配降下法を厳密に解こう。まずはいつもどおり 1 変数の場合から始める。

$f(u) = \tfrac{\lambda}{2}u^2$（$\lambda > 0$）とする。1 回の勾配ステップは u_k を $u_{k+1} = u_k - \eta\lambda u_k$ へ移すので、次式が成り立つ。

$$u_{k+1} = (1 - \eta\lambda)\,u_k, \qquad u_k = (1 - \eta\lambda)^k\,u_0.$$

反復のたびに、誤差には同じスカラー因子 $1 - \eta\lambda$ が掛けられるだけである。収束するのは $|1 - \eta\lambda| < 1$、すなわち $0 < \eta < 2/\lambda$ のとき、かつそのときに限る。最適点にあたる $\eta = 1/\lambda$ では、反復点は 1 ステップで 0 に着地する。$1/\lambda < \eta < 2/\lambda$ では因子が負になるため、u_k はステップごとに符号を反転させながらゼロへ向かって縮小する。たとえば $\lambda = 10$ の場合、$\eta = 0.1$ では因子が 0、$\eta = 0.19$ では -0.9 となり、$\eta = 0.21$ では -1.1 となって発散する。

d 次元の二次関数もまったく同じ構造を保つ。$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x}$ であるから、Equation 35.2 の更新は次のようになる。

$$\mathbf{x}_{k+1} = (\mathbf{I} - \eta\mathbf{A})\,\mathbf{x}_k.$$

各ステップは同じ線形作用素を適用するため、k ステップ後には $\mathbf{x}_k = (\mathbf{I} - \eta\mathbf{A})^k\mathbf{x}_0$ となる。$\mathbf{A}$ は対称行列であるから、固有ベクトルからなる正規直交基底を持つ（33.3 節）。$\mathbf{x}_k$ をこの固有基底で展開した座標を $u_i^{(k)}$ としよう。行列 $\mathbf{I} - \eta\mathbf{A}$ は同じ固有ベクトルを共有し、対応する固有値は $1 - \eta\lambda_i$ となるため、各座標は完全に分離して独立に時間発展する。

$$u_i^{(k+1)} = (1 - \eta\lambda_i)\,u_i^{(k)}, \qquad \text{hence} \qquad u_i^{(k)} = (1 - \eta\lambda_i)^k\,u_i^{(0)}. \tag{35.9}$$

したがって、d 次元の二次関数上の勾配降下法とは、固有値すなわち**モード**ごとに 1 つずつ、計 d 個の独立な 1 変数漸化式を並べたものにほかならない。因子 $1 - \eta\lambda_i$ から、直ちに 2 つの帰結が導かれる。

- **収束**には、すべてのモードで $|1-\eta\lambda_i| < 1$ が必要であり、これは $0 < \eta < 2/\lambda_{\max} = 2/L$ に帰着する。このしきい値が**安定性の上限**（stability ceiling）である。二次関数上では、降下補題が十分条件として保証した境界 $2/L$ は、厳密な必要条件でもある。η をこれより大きくすると、最も硬いモードの因子が -1 を下回り、他のすべてのモードがきれいに減衰していても、その座標だけは振幅を増しながら振動する。
- **収束速度**は最も遅いモードによって決まる。誤差ノルムは $\rho(\eta)^k$ に従って減衰する。ここで $\rho(\eta) = \max_i |1-\eta\lambda_i|$ は $\mathbf{I} - \eta\mathbf{A}$ の**スペクトル半径**（spectral radius）、すなわち固有値の絶対値の最大値である。

$\mathbf{A} = \mathrm{diag}(1, 10)$ では、安定性の上限は $\eta = 0.2$ である。以下のスイープを実行する前に、60 ステップを $(1, 1)$ から進めた後に誤差が最小になるステップサイズを予想してみよう。慎重な 0.02 か、降下補題が推す $0.1 = 1/L$ か、それとも上限にもっと近い値だろうか。

```
A = np.diag([1.0, 10.0])                    # eigenvalues 1 and 10, so L = 10
x0 = np.array([1.0, 1.0])

def run_gd(A, x0, eta, steps):
    xs = [x0]
    for _ in range(steps):
        xs.append(xs[-1] - eta * (A @ xs[-1]))
    return np.array(xs)

print(' eta    |1-eta*1|  |1-eta*10|  spectral radius   |x_60|')
for eta in [0.02, 0.10, 2 / 11, 0.19, 0.20, 0.21]:
    f1, f10 = abs(1 - eta * 1.0), abs(1 - eta * 10.0)
    print(f'{eta:5.3f}    {f1:6.3f}     {f10:6.3f}        {max(f1, f10):6.3f}     '
          f'{np.linalg.norm(run_gd(A, x0, eta, 60)[-1]):9.1e}')
```

```
 eta    |1-eta*1|  |1-eta*10|  spectral radius   |x_60|
0.020     0.980      0.800         0.980       3.0e-01
0.100     0.900      0.000         0.900       1.8e-03
0.182     0.818      0.818         0.818       8.3e-06
0.190     0.810      0.900         0.900       1.8e-03
0.200     0.800      1.000         1.000       1.0e+00
0.210     0.790      1.100         1.100       3.0e+02
```

慎重な $\eta = 0.02$ は安全だが動きが鈍い（$\rho = 0.98$ で、誤差は 0.3 が 60 ステップ後にも残る）。$\eta = 0.1$ では硬いモードは 1 ステップで消えるものの、遅いモードが全体の収束率を $\rho = 0.9$ に律速する。興味深いことに、$\eta = 0.19$ も正反対の理由からまったく同じ収束率 $\rho = 0.9$ を示す。今度は硬いモードが因子 -0.9 で振動するためである。両者の間にある $\eta = 2/11 \approx 0.182$ は両端のモードをつり合わせ、他のどの選択よりも誤差を 2 桁小さく抑える（$\rho = 0.818$、誤差 8.3×10^{-6}）。上限ちょうどの $\eta = 0.2$ では、硬いモードの因子が -1 となって永久に ± 1 の間で符号を反転し続けるため、$\|\mathbf{x}_{60}\| = 1.0$ となる。それをわずかに超える $\eta = 0.21$ に踏み出すと、60 回の反復で誤差は 3×10^2 まで膨れ上がる。

35.1.2.3 最適なステップサイズ

安定範囲の中で、最も速く収束する固定ステップサイズ η はどれだろうか。先ほどのスイープがそのまま答えを示している。両端のモードをつり合わせればよいのである。任意の固定された η に対し、写像 $\lambda \mapsto |1-\eta\lambda|$ は頂点を $\lambda = 1/\eta$ に持つ V 字型を描く。区間 $[\lambda_{\min}, \lambda_{\max}]$ 上で V 字型関数が最大値をとるのはいずれかの端点であるから、$\rho(\eta) = \max\left(|1-\eta\lambda_{\min}|,\ |1-\eta\lambda_{\max}|\right)$ となる。η が $1/\lambda_{\max}$ を超えると、さらに大きくするにつれて $\lambda_{\min}$ にある遅いモードは速まる一

方、$\lambda_{\max}$ にある硬いモードは悪化する。

命題（二次関数上の最適固定ステップ）。$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^{\top}\mathbf{A}\mathbf{x}$ に対し、収束因子 $\rho(\eta) = \max_i |1-\eta\lambda_i|$ を最小化するステップサイズと最小値は次式で与えられる。

$$\eta^{\star} = \frac{2}{\lambda_{\min} + \lambda_{\max}}, \qquad \rho(\eta^{\star}) = \frac{\kappa - 1}{\kappa + 1}, \qquad \kappa = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{35.10}$$

証明。 任意の η に対して $\rho(\eta) \geq \max(1-\eta\lambda_{\min},\, \eta\lambda_{\max}-1)$ が成り立ち、2 つの数の最大値はそれらのいかなる凸結合（convex combination）以上でもある。重みとして $\lambda_{\max}/(\lambda_{\min}+\lambda_{\max})$ と $\lambda_{\min}/(\lambda_{\min}+\lambda_{\max})$ を選ぶと、η を含む項がきれいに消去される。

$$\rho(\eta) \;\geq\; \frac{\lambda_{\max}(1-\eta\lambda_{\min}) + \lambda_{\min}(\eta\lambda_{\max}-1)}{\lambda_{\min}+\lambda_{\max}} = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}} = \frac{\kappa-1}{\kappa+1}.$$

等号が成立するのは 2 つの項が一致する $1-\eta\lambda_{\min} = \eta\lambda_{\max}-1$ の場合に限り、これを解くと $\eta = \eta^{\star}$ となる。$\eta^{\star}$ において両端点の値はともに $(\kappa-1)/(\kappa+1)$ に等しくなり、V 字型の形状により内部のすべての固有値はその高さ以下に保たれるため、この下界が達成される。$\blacksquare$

Figure 35.3 は、この証明を $\mathrm{diag}(1, 10)$ 上で図示したものである。最適な V 字は、両端点が同じ高さに揃うものである。この場合 $\eta^{\star} = 2/11$、$\rho = 9/11 = 0.818$ となるため、6 桁の精度を得るには $\log 10^{6}/\log(11/9) \approx 69$ ステップを要する。本章の通し例（$\kappa = 1000$）では $\rho = 999/1001$ となり、同じ 6 桁の精度を得るのに 6,908 ステップが必要になる。

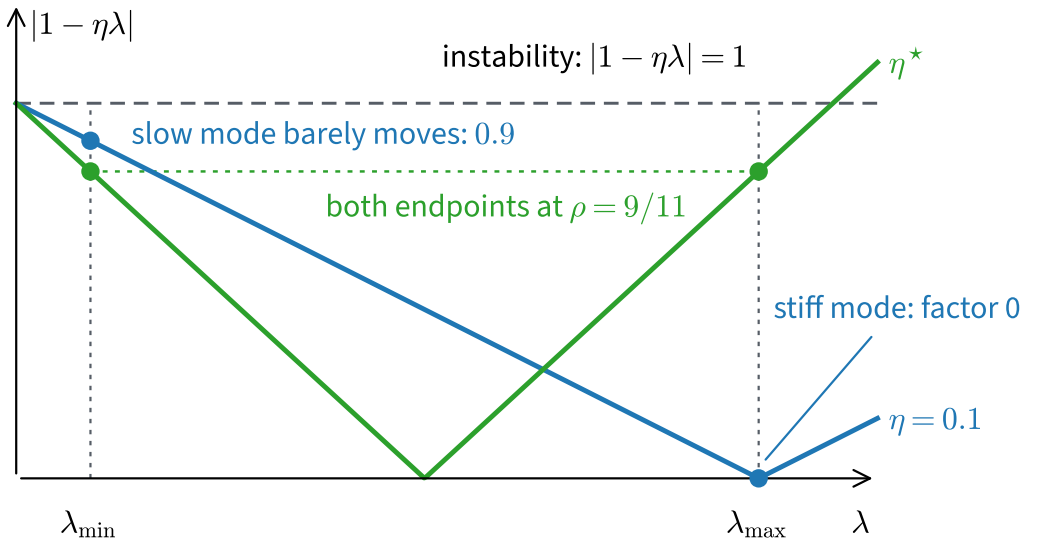


Figure 35.3: モードごとの縮小率 $|1-\eta\lambda|$ は $\lambda = 1/\eta$ を頂点とする V 字を描き、収束因子 $\rho(\eta)$ は両端の固有値 $\lambda_{\min} = 1$ と $\lambda_{\max} = 10$ における値のうち大きい方に等しい。$\eta = 0.1$ を選ぶと硬いモードは即座に消えるが遅いモードの縮小率は 0.9 にとどまる。最適ステップ $\eta^{\star} = 2/(\lambda_{\min}+\lambda_{\max})$ は両端点を $(\kappa-1)/(\kappa+1) = 9/11$ で等しくする。一方の端点を下げようとすれば、もう一方が上がってしまう。

$\kappa = 1$（完全に丸いボウル）のときは $\rho = 0$ となり、1 ステップで最小点に到達する。大きな κ に対しては $(\kappa-1)/(\kappa+1) \approx 1 - 2/\kappa$ であるから $\log(1/\rho) \approx 2/\kappa$ となり、誤差を ϵ 倍に縮めるには約 $\frac{\kappa}{2}\log\frac{1}{\epsilon}$ 回の反復が必要となる。すなわち、**勾配降下法の反復回数は条件数に対して線形に増大する**。ボトルネックは一文で言い表せる。安定性の要請からステップサイズが最も硬いモードに従う $\eta \lesssim 2/\lambda_{\max}$ に縛られるため、最も平坦なモードは反復ごとに $\eta\lambda_{\min} \approx 2/\kappa$ しか進めないからである。

$\eta^\star$ において、因子 ρ は漸近的な収束率よりもさらに強い意味を持つ。$\mathrm{diag}(1, 10)$ では、モードごとの因子は $+9/11$ と $-9/11$（先ほどのスイープの $\eta = 2/11$ の行）であり、遅いモードが単調に縮小する一方、硬いモードは符号を交互に反転させる。両座標とも同じ倍率で縮むため、$\|\mathbf{x}_k\|$ は平均的にではなく、毎ステップ厳密に $(\kappa-1)/(\kappa+1)$ 倍ずつ縮小する。

35.1.2.4 安定性の縁

安定性の上限は、ニューラルネットワークに対して単純な処方箋を示唆する。ヘッセ行列の最大固有値 $\lambda_{\max}(\nabla^2 f)$ である**鋭さ**（sharpness）を計算し、$\eta < 2/\lambda_{\max}$ と設定すればよいという考えである。しかし実際のニューラルネットワークの訓練はその通りには進まない。軌跡に沿ってヘッセ行列自体が変化していくからである。フルバッチ勾配降下法のもとでは、鋭さは通常おおよそ $2/\eta$ に達するまで上昇し続け、その後はそのしきい値のすぐ周りで推移する。その間、個々のステップでは損失が頻繁に跳ね上がるにもかかわらず、長期的には減少していく (Cohen et al. 2021)。9.1.4 節では小さな tanh ネットワーク上でこの実験を行い、詳しく調べている。ここでは問いを絞り、この振る舞いのどこまでを二次モデルで説明できるか考えよう。

答えは、おおよそ半分である。ヘッセ行列の固有基底において最大固有値モードの乗数は $1 - \eta\lambda_{\max}$ であり、-1 を下回るのはまさに $\lambda_{\max} > 2/\eta$ のときである。するとその座標は振幅を増しながら符号を反転させるため、全体として減少傾向にある損失が個々のステップでスパイクを起こす理由が説明できる。しかし固定された二次関数であれば、その増大は無限大へ発散してしまうはずである。鋭さを再び引き戻す仕組みは二次近似の外側にある。二次モデルが無視している三次のテイラー項が、その振動をより鋭さの低い平坦な領域への緩やかなドリフトへと変えるのである (Damian et al. 2023)。演習問題 8 では、モメンタムを加えるとこの安定性のしきい値が $(2 + 2\beta)/\eta$ に移動することを示す。

35.1.2.5 二次関数から凸関数へ

二次関数を超えて引き継がれるのは、κ に対する線形依存性である。微分可能な関数は、どの接平面に対してもその上または上方に位置するとき凸関数であり、さらに二次のマージンをもって接平面を上回るとき μ 強凸（strong convexity）であるという。すなわち $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{\mu}{2}\|\mathbf{y}-\mathbf{x}\|^2$ がすべての $\mathbf{x}, \mathbf{y}$ に対して成り立つことであり、35.3 節で両概念を詳しく展開する。f が L 滑らかかつ μ 強凸であれば、どの点においても曲率 μ と L の 2 つの二次関数の間に挟まれる。条件数は $\kappa = L/\mu$ が $\lambda_{\max}/\lambda_{\min}$ に取って代わり、$\eta = 1/L$ の勾配降下法は次式を満たす。

$$f(\mathbf{x}_k) - f^\star \leq \left(1 - \frac{1}{\kappa}\right)^k \left(f(\mathbf{x}_0) - f^\star\right), \tag{35.11}$$

これもやはり $O(\kappa \log\frac{1}{\epsilon})$ 回の反復を要する。f が強凸ではなく単に滑らかな凸関数であっても、勾配降下法は大域的最適解へ収束するが、その収束率はより遅い劣線形レート $O(1/k)$ になる。凸性も外してしまえば、残るのは停留性の上界 Equation 35.6 のみである。これらの証明は凸性に基づくものであり、35.3.3.3 節で展開し、二次関数上で達成されるよりシャープな収束率とも比較する。

35.1.3 モメンタムと加速

勾配降下法が $O(\kappa)$ 回の反復を要するのは、単一のステップサイズで最も硬いモードと最も平坦なモードの両方に同時に対応しなければならないからである。モメンタムは 1 ステップあたり 1 回の勾配評価というコストを保ったまま更新則を変更し、両方のモードが同じ速さで縮小できるようにする。

35.1.3.1 ヘビーボールモメンタム

谷の幾何構造を見れば、何を改めるべきかは直ちにわかる。急峻な壁に沿っては連続する勾配が逆向きを指すため（Figure 35.2 のジグザグ）、直近の勾配を平均すれば振動が打ち消し合う。平坦な谷底に沿っては連続する勾配は小さいものの一貫して同じ向きに揃っているため、それらを蓄積すれば速度が増す。1 つの単純な仕組みでこの両方が実現できる。反復点に記憶を伴う速度を持たせればよいのである。**ヘビーボールモメンタム** (Polyak 1964) は、通常の勾配ステップを次式に置き換える。

$$\mathbf{v}_{k+1} = \beta\,\mathbf{v}_k - \eta\,\nabla f(\mathbf{x}_k), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1}, \tag{35.12}$$

ここで、$\beta \in [0,1)$ はモメンタム係数である。速度 $\mathbf{v}_k$ は過去の勾配の指数減衰和となっている。$\mathbf{v}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$ であるから、$\mathbf{v}$ を消去すると等価な 2 ステップ形式が得られる。

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\nabla f(\mathbf{x}_k) + \beta\,(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

各更新は勾配ステップを進めるとともに、直前の変位の β 倍を加える。$\beta = 0$ とおけば通常の勾配降下法に戻る。

二次関数上では、各モードは再び分離する。勾配 λu_k を持つ単一のモードでは、2 ステップ更新式は $u_{k+1} = (1 + \beta - \eta\lambda)\,u_k - \beta\,u_{k-1}$ となり、各反復点は直前の 2 点に依存する。この形の 2 階漸化式は、減衰調和振動子（摩擦 $1-\beta$ のあるボウルの中を転がる重い球）の離散版にあたり、β が減衰の強さを制御する。

Figure 35.4 は、$\mathrm{diag}(1, 10)$ 上で勾配降下法の最適ステップ $\eta = 2/11$ に統一したときに生じる 3 つのレジームを示している。モメンタムが小さすぎると、経路は通常の勾配降下法と同様にジグザグになる。ちょうど適切な量であれば、1 回のオーバーシュートの後に滑らかに落ち着く。モメンタムが大きすぎると、球は過剰な慣性を持って最小点の周りを渦巻き状に回る。次にこれら 3 つのレジームすべてに厳密な収束率（convergence rate）を与え、可能な限り最速の収束率を達成するにはより大きなステップサイズも必要になることを見いだす。

35.1.3.2 $\sqrt{\kappa}$ 則

ここで、適切な組 (η, β) を選べば、すべてのモードが同一の縮小率 $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ で縮小することを示そう。$\kappa = 10$ の場合、この最適な組は $\beta^\star \approx 0.27$、$\eta_{\mathrm{HB}} \approx 0.23$ であり、縮小率は勾配降下法の 0.82 から 0.52 へと下がり、6 桁の精度に 27 ステップで到達する（勾配降下法では 69 ステップ）。驚くべきことに、この結果全体は 1 つのモードに関するただ 1 つの代数的事実から導かれる。

命題（ヘビーボール法はすべてのモードを $\sqrt{\beta}$ で縮小させる）。 *$0 < \beta < 1$ とする。$f(u) = \tfrac{\lambda}{2}u^2$ 上で、ヘビーボール法の反復点は $u_{k+1} = (1+\beta-\eta\lambda)u_k - \beta u_{k-1}$ に従う。$(1-\sqrt{\beta})^2 \le \eta\lambda \le (1+\sqrt{\beta})^2$ である限り、特性多項式の 2 つの根はともに絶対値 $\sqrt{\beta}$ を持ち、$|u_k| \le C\,k\,\beta^{k/2}$ がすべての $k \ge 1$ とある定数 C（u_0、u_1、および $\eta\lambda$ に依存）に対して成り立つ。線形因子 k が必要になるのは、根が重根となる両端点のみである。*

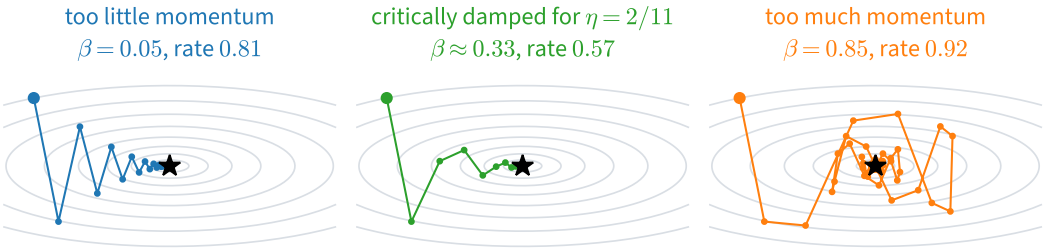


Figure 35.4: $\kappa = 10$ の谷 $\frac{1}{2}(x^2 + 10y^2)$ におけるヘビーボール法（3 つの実行すべてで勾配降下法のステップ $\eta = 2/11$ を使用）。モメンタムが小さすぎると勾配降下法のようにジグザグになり（硬いモードの係数 $1 + \beta - \eta\lambda_{\max}$ が負となり、負の実根を持つ）、このステップサイズに対する臨界減衰の β が最速となり、モメンタムが大きすぎると最小点の周りを渦巻き状に回る。最適な $\sqrt{\kappa}$ の縮小率である 0.52 に達するには、より大きなステップ η_{HB}（Equation 35.14）が必要であり、そのステップサイズでは $\beta = 0.05$ は発散する。

証明。試行解 $u_k = z^k$ を漸化式に代入し、z^{k-1} で割ると、次の特性方程式が得られる。

$$z^2 - (1 + \beta - \eta\lambda)\, z + \beta = 0. \tag{35.13}$$

2 つの根の積は β である。判別式 $(1+\beta-\eta\lambda)^2 - 4\beta$ が負のとき、すなわち $|1+\beta-\eta\lambda| < 2\sqrt{\beta}$（整理すると $(1-\sqrt{\beta})^2 < \eta\lambda < (1+\sqrt{\beta})^2$）のとき、2 つの根は複素共役対をなす。複素共役 $z, \bar{z}$ は $|z|^2 = z\bar{z} = \beta$ を満たすため、どちらの根も絶対値 $\sqrt{\beta}$ を λ の値にかかわらず持ち、一般解 $u_k = az^k + b\bar{z}^k$ は $|u_k| \le (|a| + |b|)\beta^{k/2}$ を満たす。区間の両端点では判別式がゼロとなり、実重根 $(1 + \beta - \eta\lambda)/2 = \pm\sqrt{\beta}$ が得られる。重根 z に対する一般解は $u_k = (a + bk)z^k$ となり、ここから余分な因子 k が生じる。■

系（最適調整）。両端のモードを複素根区間のちょうど両端に配置する。すなわち $\eta\lambda_{\min} = (1 - \sqrt{\beta})^2$ および $\eta\lambda_{\max} = (1+\sqrt{\beta})^2$ とおく。するとすべての中間固有値はこの区間の内部に入るため、あらゆるモードの根が絶対値 $\sqrt{\beta}$ を持つ。平方根をとって加減すると、$\sqrt{\eta\lambda_{\max}} - \sqrt{\eta\lambda_{\min}} = 2\sqrt{\beta}$ および $\sqrt{\eta\lambda_{\max}} + \sqrt{\eta\lambda_{\min}} = 2$ が得られる。第 2 式を η について解き、第 1 式を第 2 式で割れば $\sqrt{\beta} = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$ となる。したがって、次のパラメータ設定をとる。

$$\eta_{\text{HB}} = \left(\frac{2}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}} \right)^2, \qquad \beta^\star = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^2, \tag{35.14}$$

このとき、ヘビーボール法はすべてのモードを次の縮小率で縮小させる。

$$\rho_{\text{HB}} = \sqrt{\beta^\star} = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \;\approx\; 1 - \frac{2}{\sqrt{\kappa}}. \tag{35.15}$$

Equation 35.10 と比較してほしい。モメンタムは κ を $\sqrt{\kappa}$ に置き換え、反復計算量を $O(\kappa \log \frac{1}{\epsilon})$ から $O(\sqrt{\kappa} \log \frac{1}{\epsilon})$ へと劇的に削減する (Polyak 1964)。有限の k においては、両端のモードにおける重根が控えめな対数オーバーヘッドを加える。誤差は $k\rho^k$ のようにスケールするため、

$\log(1/\rho) \approx 2/\sqrt{\kappa}$ を用いると、精度 ϵ に達するステップ数は $k \approx \frac{\sqrt{\kappa}}{2}\big(\log\frac{1}{\epsilon} + \log k\big)$ の解となる。これは漸近則にオーダー $\sqrt{\kappa}\log\kappa$ の補正項を加えたものである。

Figure 35.5 は、この証明を $\kappa = 10$ について視覚化したものである。左図では、絶対値最大の根の絶対値が複素根区間全体にわたって $\sqrt{\beta}$ で水平に並んでいる。右図の複素平面では、すべてのモードの根の対が半径 $\sqrt{\beta^\star} = 0.52$ の円周上にきっちり乗っているのに対し、勾配降下法の実乗数は ± 0.82 まで広がっている。根が実軸から離れることを許すことで、モメンタムは平坦なモードに $1 - 2/\kappa$ の鈍足な進行を強いる代わりに、硬いモードと平坦なモードに単一の小さな絶対値を共有させるのである。

この命題は、Figure 35.4 の 3 つの軌跡（$\eta = 2/11$ により 2 つのモードが $\eta\lambda = 0.18$ と 1.82 に置かれている）の厳密な収束率も説明する。$\beta = 0.85$ では両モードとも区間の真に内部に入り、縮小率は $\sqrt{0.85} = 0.92$ となって複素根が渦巻き状のオーバーシュートを生む。区間が遅いモードを依然として捉える最小の β である $\beta = (1 - \sqrt{2/11})^2 \approx 0.33$ は、このステップサイズにおける最速の選択であり、縮小率 0.57 を達成する。$\beta = 0.05$ では区間 $[0.60, 1.50]$ が両モードとも外してしまうため、遅いモードは実根 0.81 を持ち、硬いモードはステップごとに符号を反転させる負の実根を持つ。このような符号交代は、連続時間（continuous-time）の物理的な振動子では決して起こりえない現象である。

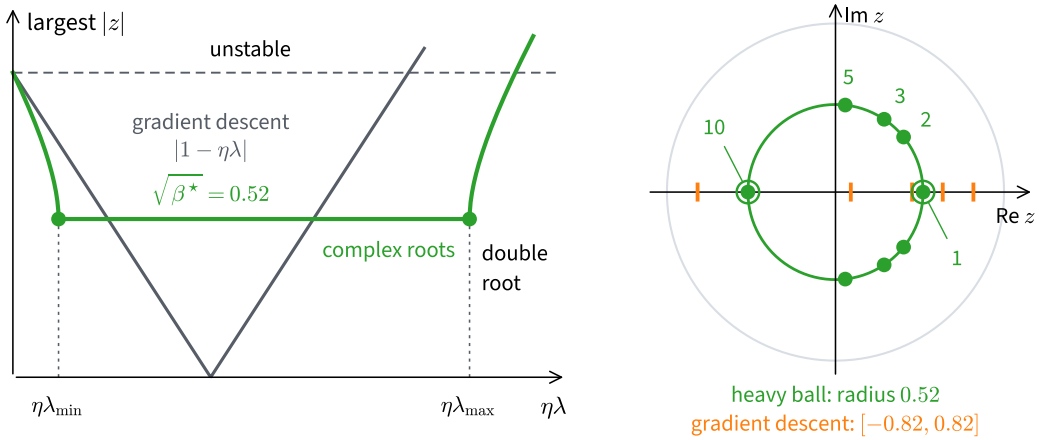


Figure 35.5: Equation 35.14 の調整を用いた $\kappa = 10$ の例におけるヘビーボール法。(a) 固定された β に対し、大きい方の根の絶対値は根が複素数となる領域全体で $\sqrt{\beta}$ の一定値をとり、最適調整は両端のモードをその区間の両端に配置する。(b) 複素平面において、すべてのモードの根の対（固有値 λ でラベル付け）は半径 $\sqrt{\beta^\star} = 0.52$ の円周上に乗る一方、勾配降下法の因子は実軸上の $[-0.82, 0.82]$ に広がる。両端のモードは重根となり、誤差に因子 k を生じさせる。

ただし、この $\sqrt{\kappa}$ 定理が厳密に成り立つのは二次関数に限られる点に注意が必要である。Lessard et al. (2016) は、1 次元の滑らかな強凸関数であって、Equation 35.14 の調整を用いたヘビーボール法が特定の初期値（初期化）から収束せず、反復点が安定なリミットサイクルに閉じ込められる例を構成した（演習問題 6）。したがって、深層学習でヘビーボールモメンタムが広く使われている背景 (Sutskever et al. 2013) は、大域的な収束定理ではなく、局所的な二次近似の描像と経験的な成功にある。

35.1.3.3 ネステロフの先読み

1 つの小さな修正によって大域的な保証が復活する。勾配を現在位置 $\mathbf{x}_k$ ではなく、モメンタムがまさに反復点を運ぼうとしている先読み点 $\mathbf{x}_k + \beta\,\mathbf{v}_k$ で評価するのである。**ネステロフの加速勾**

配法（Nesterov's accelerated gradient）(Nesterov 1983) は次式に従って更新する。

$$\mathbf{v}_{k+1} = \beta \mathbf{v}_k - \eta \nabla f(\mathbf{x}_k + \beta \mathbf{v}_k), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1}. \tag{35.16}$$

ヘビーボール法との違いは、この評価点のずれだけである。直観的には、モメンタムが反復点を最小点の向こう側へ行き過ぎさせようとしているとき、先読み点における勾配はすでに上り坂の引き戻す向きを指しており、オーバーシュートが起きる前にブレーキをかける。

命題（二次関数上のネステロフ法）。 $\kappa = L/\mu$ **とする。** $\eta = 1/L$ **および** $\beta = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ **とおくとき、固有値が** $[\mu, L]$ **に含まれる二次関数のすべてのモードについて、特性根の絶対値は高々** $1 - 1/\sqrt{\kappa}$ **となり、遅いモード** $\lambda = \mu$ **で重根を持つ。したがって、誤差は** $C\,k\,(1-1/\sqrt{\kappa})^{k}$ **で上から抑えられる。**

証明。 固有値 λ の単一モードにおいて、$a = 1 - \eta\lambda \in [0, 1 - 1/\kappa]$ とおく。速度の更新式は $v_{k+1} = a\beta v_k - \eta\lambda u_k$ となり、$u_{k+1} = a u_k + a\beta v_k$ を得る。ここに $v_k = u_k - u_{k-1}$ を代入すると漸化式 $u_{k+1} = (1+\beta)a\,u_k - \beta a\,u_{k-1}$ が得られ、その特性多項式 $z^2 - (1+\beta)a\,z + \beta a$ の判別式は $a\,[(1+\beta)^2 a - 4\beta]$ となる。今回の β の選び方では $(1+\beta)^2 = 4\kappa/(\sqrt{\kappa}+1)^2$ が成り立つ。遅いモード $a = (\kappa-1)/\kappa$ においては角括弧の中身が $4(\kappa-1)/(\sqrt{\kappa}+1)^2 - 4\beta = 4\beta - 4\beta = 0$ となり、$(1+\beta)a/2 = (\sqrt{\kappa}-1)/\sqrt{\kappa} = 1 - 1/\sqrt{\kappa}$ に重根が生じる。それより硬いすべてのモードはより小さな a を持つため、角括弧の中身は真に負となり、複素根（または 0 の根、ただし $a = 0$ の場合）を生じさせ、その絶対値は $\sqrt{\beta a} < \sqrt{\beta(1-1/\kappa)} = 1 - 1/\sqrt{\kappa}$ となる。$\blacksquare$

二次関数上でのネステロフ法の縮小率は、0.684 ($\kappa = 10$)、0.900 ($\kappa = 100$)、0.968 ($\kappa = 1000$) となり、ヘビーボール法の 0.519、0.818、0.939 よりわずかに遅いものの、同じ $\sqrt{\kappa}$ のスケーリングを保つ。決定的に重要なのは、ヘビーボール法とは異なり、ネステロフ法が二次関数を超えてもその収束保証を維持する点である（ここでは証明なしに結果のみ述べる (Nesterov 2018)）。任意の μ 強凸かつ L 滑らかな関数上で、同じ定数 β を用いたネステロフ法の更新は $f(\mathbf{x}_k) - f^\star \le C\,(1 - 1/\sqrt{\kappa})^{k}$ を満たす（単位の違いに注意しよう。これは距離の二乗としてスケールする関数値ギャップ $f - f^\star$ を抑える式であるため、二次関数の命題における距離の上界と比べて指数が半分に対応している）。強凸性を持たない滑らかな凸関数上でも、スケジュール $\beta_k \to 1$ を用いれば $f(\mathbf{x}_k) - f^\star = O(1/k^2)$ を達成し、勾配降下法の $O(1/k)$ を改善する。さらに、一般にはこの速さを上回ることはできない。次元が反復回数に比べて大きい限り、過去の勾配のみから反復点を構成するいかなる手法も、最悪ケースにおいて $O(1/k^2)$ や $\sqrt{\kappa}$ 依存性を凌駕できないからである (Nemirovski and Yudin 1983)。演習問題 7 では、この限界を証明する最悪ケースの関数を実際に構成する。

ここで、本節の冒頭で挙げた反復回数を実際に検証してみよう。$\mathrm{diag}(1, \kappa)$ 上で $(1, 1)$ から出発するとき、6 桁の精度を得るのに要するステップ数は、勾配降下法ではおおよそ $\frac{\kappa}{2}\log 10^{6} \approx 6.9\,\kappa$、ヘビーボール法では $6.9\sqrt{\kappa}$ を少し上回る程度になるはずである。

```
def iterations(lam, eta, beta=0.0, lookahead=False, tol=1e-6):
    """Steps until |x_k| <= tol |x_0| on f(x) = sum_i lam_i x_i^2 / 2."""
    x, v = np.array([1.0, 1.0]), np.zeros(2)
    n0 = np.linalg.norm(x)
    for k in range(1, 10**6):
        g = lam * (x + beta * v) if lookahead else lam * x
        v = beta * v - eta * g
        x = x + v
        if np.linalg.norm(x) <= tol * n0:
            return k
```

```
    print('kappa      GD   heavy ball   Nesterov     GD/kappa   HB/sqrt(kappa)')
    for kappa in [10.0, 100.0, 1000.0]:
        lam, sk = np.array([1.0, kappa]), np.sqrt(kappa)
        gd = iterations(lam, 2 / (1 + kappa))
        hb = iterations(lam, (2 / (1 + sk))**2, ((sk - 1) / (sk + 1))**2)
        na = iterations(lam, 1 / kappa, (sk - 1) / (sk + 1), lookahead=True)
        print(f'{kappa:5.0f} {gd:7d} {hb:9d} {na:10d}      '
              f'{gd / kappa:7.1f}     {hb / sk:7.1f}')
```

```
kappa      GD   heavy ball   Nesterov     GD/kappa   HB/sqrt(kappa)
   10      69         27         43          6.9          8.5
  100     691         93        155          6.9          9.3
 1000    6908        315        508          6.9         10.0
```

表はすべての予測を裏付けている。勾配降下法のステップ数を κ で割った比は 6.9 に張り付いたままであり、$6{,}908$ 回の反復（$\kappa = 1000$ のとき）に達するのに対し、ヘビーボール法はわずか 315 回で完了する。なお、ヘビーボール法の比 $k/\sqrt{\kappa}$ が緩やかに上昇している（8.5、9.3、10.0）ことに注目しよう。これは両端のモードがそれぞれ重根を持つためであり、その 2×2 反復行列はただ 1 本の固有ベクトルしか持たない重複固有値のジョルダンブロックをなし（33.3.1.5 節）、その k 乗が因子 k をもたらすからである。精密化した予測式 $k \approx \frac{\sqrt{\kappa}}{2}(\log 10^6 + \log k)$ は実測とよく一致し（8.6、9.2、9.8）、$\kappa = 10^4$ ではヘビーボール法は $1{,}056$ ステップ、比にして 10.6 を要する（演習問題 5）。ネステロフ法の 508 回（$\kappa = 1000$ のとき）は両者のちょうど中間に収まっており、縮小率 0.968 対ヘビーボール法の 0.939 から予測されるとおりである。3 手法はいずれもステップごとに勾配を 1 回しか評価しないため、モメンタムによる高速化には追加の勾配計算コストが一切かからない。しかし同時に、$\sqrt{\kappa}$ は勾配だけで達成しうる限界でもある。一次手法の下界により、高次元において過去の勾配のみから構成される手法はこれを超えられない。κ への依存性を完全に取り除くには、曲率を実際にモデル化するステップが必要となる（35.2 節）。

35.1.4 まとめ

滑らかさは、勾配ステップをどこまで信頼できるかの上限を定める。L 滑らかな目的関数上では、$\eta = 2/L$ まで降下が保証される。二次関数上では、最良の固定ステップは反復ごとに誤差を $(\kappa - 1)/(\kappa + 1)$ 倍に縮小させるが、安定性が最も硬いモードに支配される一方で進行速度が最も平坦なモードに支配されるため、ステップ数は条件数 κ に対して線形に増大せざるをえない。モメンタムはすべてのモードを共通の縮小率 $\sqrt{\beta}$ に揃え、最適調整された因子 $(\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$ によって、追加の勾配コストなしに反復回数を $O(\kappa)$ から $O(\sqrt{\kappa})$ へと削減する。ニューラルネットワーク上では、鋭さが選択したステップサイズの安定性の上限 $2/\eta$ まで上昇するが、この現象は二次モデルだけでは一部しか説明できない。残る問いは 3 つある。曲率を直接モデル化するとオプティマイザはどこまで達成できるのか（35.2 節）、停留点が大域的最小点であることがいつ保証されるのか（35.3 節）、そして勾配にノイズが加わると何が変わるのか（35.4 節）である。

35.1.5 演習問題

1. **前処理付き降下方向。** 35.1.1.1 節の命題では、方向の長さをユークリッドノルムで測っている。異なるノルムを選ぶと、異なる最急降下方向が得られる。
    1. $\mathbf{d} = -\mathbf{P}^{-1}\nabla f(\mathbf{x})$ が任意の正定値行列 $\mathbf{P}$ に対して降下方向となることを示せ。
    2. 長さを $\|\mathbf{d}\|_{\mathbf{P}} = \sqrt{\mathbf{d}^{\top}\mathbf{P}\,\mathbf{d}}$ で測るものとする。$\nabla f(\mathbf{x})^{\top}\mathbf{d}$ の $\|\mathbf{d}\|_{\mathbf{P}} = 1$ 上における最小点が $-\mathbf{P}^{-1}\nabla f(\mathbf{x})$ に比例することを示し、さらにどのような $\mathbf{P}$ を選べばニュートン方向 $-\big(\nabla^2 f(\mathbf{x})\big)^{-1}\nabla f(\mathbf{x})$（35.2 節の主題）が復元されるか特定せよ。変数変換

$\mathbf{u} = \mathbf{P}^{1/2}\mathbf{d}$ を用いればユークリッドノルムの場合に帰着する。

2. [code] **滑らかさの定数**。滑らかさの定数 L（Equation 35.3）は f の曲率を上から抑え、ロジスティック損失に対してはデータから直接計算できる。

   1. f を 2 回連続微分可能かつ L 滑らかとする。$\|\nabla^2 f(\mathbf{x})\,\mathbf{v}\| \le L\,\|\mathbf{v}\|$ がすべての $\mathbf{x}$ と $\mathbf{v}$ に対して成り立つことを示し、ヘッセ行列の固有値がすべて $[-L, L]$ に収まることを結論せよ。これにより、Equation 35.3 の直後で述べた同値性の逆方向が示される。

   2. 次の関数を考える。

$$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i\,\mathbf{x}_i^{\top}\mathbf{w})\right) + \frac{\nu}{2}\|\mathbf{w}\|^2$$

   ここでラベルは $y_i = \pm 1$、$\nu = 0.1$ である。これは `loss` 関数として 35.2.1 節のデータセルで用いるものであり、$\mathbf{x}_i^{\top}$ は第 i 行として行列 $\mathbf{X} \in \mathbb{R}^{n \times d}$（$n = 200$、$d = 5$ で、各成分は独立な標準ガウス分布に従う）から取り出したベクトルである。$L = \lambda_{\max}(\mathbf{X}^{\top}\mathbf{X})/(4n) + \nu$ が正当な滑らかさの定数であり、ヘッセ行列が $\mathbf{w} = \mathbf{0}$ においてこの上界を達成することを示せ。

   3. この L を、標準ガウス分布に従う成分を持つ 200×5 行列 $\mathbf{X}$ と、ラベル $y_i \in \{\pm 1\}$（確率 $P(y_i = 1) = 1/(1 + \exp(-\mathbf{x}_i^{\top}\mathbf{w}_{\text{true}}))$、$\mathbf{w}_{\text{true}} = (1, -2, 1.5, 0, 2)$ に従って生成）に対して計算せよ。どの乱数サンプルでも値は 0.4 付近になる。標準ガウス成分に対して $\lambda_{\max}(\mathbf{X}^{\top}\mathbf{X})/n \approx (1 + \sqrt{d/n})^2$ が成り立つことを用いて、その理由を説明せよ。さらに、$\eta = 0.5$（基準ステップ）と $\eta = 1/L$ のそれぞれについて、$\mathbf{w} = \mathbf{0}$ から出発して $\|\nabla f(\mathbf{w})\| \le 10^{-10}$ に達するまでに要するステップ数を数えよ。どちらが速いか。また、$\eta = 0.5$ は $2/L$ 未満に収まっているか。

3. [code] **四次関数におけるクリッピング**。$f(x) = \tfrac{1}{4}x^4$ とし、クリップ付き更新 $x \leftarrow x - \eta\,\min(1, c/|g|)\,g$（ただし $g = x^3$）を考える。

   1. $3x^2 \le 4 + |x|^3$ がすべての x に対して成り立ち、$|x| = 2$ で等号が成立することを示し、したがって f が Equation 35.8 を $L_0 = 4$ および $L_1 = 1$ で満たすことを示せ。また、いかなる L_0 を選んでも $L_1 = 0$ では成り立たないことも示せ。
   2. クリップ付き更新を $x_0 = 3$ から $\eta = 0.3$、$c = 1$ で実行せよ（クリップなしの反復が 6 ステップで 6.5×10^{103} に達した設定である）。何ステップがクリップされ、その各ステップは x をどれだけ動かすか。
   3. クリップ付き更新は、$|x|$ を任意の初期点 $x_0 \neq 0$ から毎ステップ減少させることを、$\eta\,c^{2/3} < 2$ の仮定のもとで示せ。クリップされる場合 $|x| > c^{1/3}$ とクリップされない場合を分けて扱え。クリッピングなしの固定ステップでは、なぜこの性質を決して持ちえないのか。

4. **他のノルムにおける最急降下**。●問題 1b では、内積から誘導されるノルム $\|\cdot\|_{\mathbf{P}}$ で長さを測った。内積に由来しないノルムを用いると、まったく異なる形の最急降下方向が生じる。

   1. $\nabla f(\mathbf{x})^{\top}\mathbf{d}$ の $\|\mathbf{d}\|_{\infty} = 1$ 上および $\|\mathbf{d}\|_{1} = 1$ 上における最小点と最小傾きをそれぞれ求め、最小点が一意となる条件を述べよ。
   2. パラメータを行列とし、その勾配 $\mathbf{G}$ の簡約特異値分解を $\mathbf{G} = \mathbf{U}\,\mathrm{diag}(\sigma_i)\,\mathbf{V}^{\top}$ とする。方向 $\mathbf{D}$ の方向傾きを $\mathrm{tr}(\mathbf{G}^{\top}\mathbf{D})$ とし、長さをスペクトルノルム $\|\mathbf{D}\|_2$ で測る。$\|\mathbf{D}\|_2 \le 1$ 上での最小傾きが $-\sum_i \sigma_i$ であり、それが $\mathbf{D} = -\mathbf{U}\mathbf{V}^{\top}$（Muon がモメンタム行列に適用する極因子、9.6 節）で達成されることを示せ。最初のステップと

して $\mathrm{tr}(\mathbf{G}^\top \mathbf{D}) = \sum_i \sigma_i \, \mathbf{u}_i^\top \mathbf{D} \, \mathbf{v}_i$ と書き直すと見通しがよい。これは Bernstein and Newhouse (2024) による最急降下の視点である。

5. [code] **ヘビーボール法：重根と因子 k。**●ヘビーボール法 Equation 35.12 を $f(u) = \tfrac{\lambda}{2} u^2$ に適用せよ。

   1. 1 回の反復を状態 (u_k, u_{k-1}) 上の線形変換として表し、その 2×2 行列の特性多項式が Equation 35.13 となることを示せ。
   2. Equation 35.14 の調整において、両端のモードは重複固有値 $\sqrt{\beta^\star}$ および $-\sqrt{\beta^\star}$ を持つ。このとき対応する行列が固有ベクトルを 1 本しか持たないこと、および k 乗したジョルダンブロック $\left(\begin{smallmatrix} \rho & 1 \\ 0 & \rho \end{smallmatrix}\right)$ の対角線の上の成分が $k\rho^{k-1}$ となることを示せ。
   3. $\|\mathbf{e}_k\| \approx k\rho^k$ および $\log(1/\rho) \approx 2/\sqrt{\kappa}$ のもとで、$k = \frac{\sqrt{\kappa}}{2}\big(\log \frac{1}{\epsilon} + \log k\big)$ を $\epsilon = 10^{-6}$ および $\kappa \in \{10, 100, 1000\}$ に対して数値的に解き、$k/\sqrt{\kappa}$ をモメンタムのコードセルの列 8.5、9.3、10.0 と比較せよ。
   4. モメンタムのコードセルのヘビーボール法を $\kappa = 10^4$ で実行し、そのステップ数を予測値と比較せよ。比 $k/\sqrt{\kappa}$ が、κ の増大に伴って決して定数に収束しない理由を説明せよ。

6. [code] **二次関数を超えたヘビーボール法。**●Lessard et al. (2016) は、勾配によって次の 1 次元関数を定義している。

$$\nabla f(x) = \begin{cases} 25x, & x < 1, \\ x + 24, & 1 \le x < 2, \\ 25x - 24, & x \ge 2. \end{cases}$$

   1. f が μ 強凸（$\mu = 1$）かつ L 滑らか（$L = 25$）であることを示せ。
   2. Equation 35.14 の調整を用いたヘビーボール法を 300 ステップ、$x_0 = 3.3$ かつ初速度ゼロから実行し、さらにネステロフ法 Equation 35.16 を $\eta = 1/L$ および $\beta = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ として同じ初期条件から実行せよ（ただし $\kappa = L/\mu$）。両手法について x_k を k に対してプロットせよ。
   3. ヘビーボール法の反復点は、区間 $x < 1$ を 2 回、区間 $x \ge 2$ を 1 回訪れる周期 3 のサイクルに漸近する。これらの区間にわたるヘビーボール法の更新を 3 つのサイクル点に関する連立一次方程式として表して解き、その解をシミュレーションと比較せよ。
   4. $[-6, 6]$ 上の初期点グリッドからヘビーボール法の実行を繰り返し、このサイクルに行き着く初期点の割合を報告せよ。f の各区間は曲率が $[\mu, L]$ に収まる二次関数であるにもかかわらず、Equation 35.15 の背後にあるモードごとの解析がなぜ f には適用できないのか。

7. [code] **一次手法に対する下界。**●この問題では、$O(1/k^2)$ の下界（Nemirovski and Yudin (1983) による結果のうち、35.1.3 節で滑らかな凸関数の場合として引用した部分）を、Nesterov (2018) の構成に従って導出する。ここで一次手法とは、反復点が $\mathbf{x}_k \in \mathbf{x}_0 + \mathrm{span}\{\nabla f(\mathbf{x}_0), \dots, \nabla f(\mathbf{x}_{k-1})\}$ を満たす任意のアルゴリズムを指すものとし、$\mathbf{x}_0 = \mathbf{0}$ ととる。$p \le d$ に対し、$\mathbf{A}_p \in \mathbb{R}^{d \times d}$ を先頭の $p \times p$ ブロックを除いてゼロとし、そのブロックは対角成分が 2、上下の副対角成分がともに -1 の三重対角行列であるとして、次式を定義する。

$$f_p(\mathbf{x}) = \frac{L}{4}\left(\tfrac{1}{2}\,\mathbf{x}^\top \mathbf{A}_p\, \mathbf{x} - x_1\right).$$

   1. f_p が凸関数かつ L 滑らかであることを示せ。

2. $\mathbf{x}$ の台（非ゼロ成分）が最初の j 個の座標に含まれるならば、$\nabla f_p(\mathbf{x})$ の台は最初の $j+1$ 個の座標に含まれることを示せ。ここから、任意の一次手法の第 k 反復点の台は最初の k 個の座標に含まれることを結論せよ。

3. f_p の最小点の各成分が $x_i^\star = 1 - i/(p+1)$（$i \le p$ のとき）およびそれ以降でゼロとなること、最小値が $-\frac{L}{8}\bigl(1 - \frac{1}{p+1}\bigr)$ であること、そして $\|\mathbf{x}^\star\|^2 \le (p+1)/3$ が成り立つことを示せ。

4. $2k+1 \le d$ とする。関数 f_{2k+1} と f_k は、台が最初の k 個の座標に含まれるすべてのベクトル上で一致する。この事実を用いて、f_{2k+1} に適用された任意の一次手法が、$\mathbf{x}^\star$ を f_{2k+1} の最小点とするとき、次式を満たすことを示せ。

$$f_{2k+1}(\mathbf{x}_k) - \min f_{2k+1} \;\ge\; \frac{3L\,\|\mathbf{x}_0 - \mathbf{x}^\star\|^2}{32\,(k+1)^2}.$$

この構成は $d \ge 2k+1$ を必要とするため、この下界が支配するのはおおよそ最初の $d/2$ 回の反復である。固定された次元では長い反復の末に漸近的により速く収束しうるが、d が数百万にのぼるニューラルネットワークでは、この制約が実質的な足かせになることはない。

5. $k \in \{5, 10, 25, 50, 100, 200, 400\}$ のそれぞれについて、k ステップの勾配降下法（$\eta = 1/L$）およびネステロフ法の更新 Equation 35.16（$\eta = 1/L$、$\beta_j = (j-1)/(j+2)$、ステップ $j = 1, 2, \ldots$）を f_{2k+1}（$d = 2k+1$）上で実行し、それぞれの最適性ギャップを上記の下界で割れ。k が大きくなるにつれて有界にとどまるのはどちらの比か。

8. [code] **モメンタムを伴う安定性の縁。●●** Cohen et al. (2021) は、モメンタムを加えると鋭さが $2/\eta$ とは異なるしきい値に落ち着くことを報告している。

   1. 問題 5a の反復行列を用いて、ヘビーボール法 Equation 35.12 が $f(x) = \frac{\lambda}{2}x^2$（$0 \le \beta < 1$）上で収束する必要十分条件が $0 < \eta\lambda < 2 + 2\beta$ であることを示せ。ネステロフ法の更新 Equation 35.16 についても、対応する安定区間を導出せよ。
   2. 9.1.4 節の安定性の縁の実験を変更し、$\beta \in \{0.25, 0.5\}$ に対してヘビーボール法およびネステロフ法の更新で訓練せよ。ただし各実行では、予測されるしきい値が 5 から 12 の間に入るようなステップサイズを選ぶこと。鋭さの推移を $2/\eta$ と (a) で導出したしきい値の両方と比較して追跡せよ。
   3. 鋭さはどちらのしきい値に落ち着くか。またこの結果は、モメンタムを用いて訓練する際に、初期化時点で測定した鋭さから η を選ぶことに対してどのような示唆を与えるか。

35.2 曲率法：ニュートン法、準ニュートン法、信頼領域法

勾配降下法は、最小化する関数の曲率を決して測らず、ただそれに耐えるだけである。最も急な方向がステップサイズの上限を抑え、その結果として最も平坦な方向は這うようにしか進まず、反復回数は条件数 κ に比例して（モメンタムを加えても $\sqrt{\kappa}$ に比例して）増大する（35.1.3 節）。共通の例題である $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ では、勾配降下法で 6,908 ステップ、モメンタムを用いても 315 ステップを要した。しかし、この問題は一見して直ちに解ける。勾配の各座標をそれぞれの曲率（1 または 1000）で割れば、わずか 1 ステップで最小点に直接到達するのである。曲率を単にやり過ごすのではなく考慮に入れたとき、オプティマイザはどこまで到達できるだろうか。

古典的な答えは、勾配降下法より 1 世紀半も前に誕生したニュートン法（Newton's method）である。各反復点において目的関数に二次近似を当てはめ、その二次関数の最小点へと一気に跳ぶ。本節では、このステップがどれほど速く収束するか、どれだけのコストがかかるか、どこで破綻するか、そして二階導関数が得られないときに勾配だけから曲率情報をどこまで復元できるかを調べる。最小点の近傍ではニュートン法は驚異的に速いが、ニューラルネットワークに対してはヘッセ行列の構成が高コストすぎ、さらに非凸関数上では極小点と同じくらい容易に鞍点へと引き寄せられてしまう。そこで続いて 2 つの打開策に目を向ける。連続する勾配の差分から曲率を再構築する準ニュートン法（quasi-Newton method）と、二次モデルが実際の関数とよく一致する近傍に各ステップを制限する信頼領域法（trust-region method）である。

深層ネットワークの訓練では、こうした古典的手法は一次のオプティマイザに比べてはるかに使われる頻度が低いが、その理由は率直に述べておく価値がある。完全なヘッセ行列は大きすぎて構成できず、またミニバッチノイズが勾配の差分から作る曲率推定を損なってしまうからである。それでもなお、根底にある考え方は深層学習の随所に再び現れる。Adam の座標ごとのスケーリングはニュートン法の逆ヘッセ行列の対角近似として働き（35.5 節）、曲率行列に加えるダンピングは信頼領域と同じ安定化の役割を果たす。心に留めておくべき中心的な結果は、ニュートン法の二次収束（quadratic convergence）を表す Equation 35.19 であり、準ニュートン法の項は初読時には飛ばしてもよい。なお 35.1 節からは、バックトラッキング直線探索（backtracking line search）の Armijo 条件と、鞍点で勾配降下法を停滞させたテスト関数 $\frac{1}{2}x^2 + \cos y$ という 2 つの要素を引き継ぐ。

35.2.1 ニュートン法

勾配降下法は、現在の点において f の線形近似を構築する。線形関数には最小値が存在しないため、アルゴリズム自身でステップ長を決めることができず、外部から与える学習率に頼らざるを得ない。これに対し二次近似は、上向きに曲がっている限り唯一の最小点をもつため、局所的な幾何構造によって長さが自動的に決まるステップを与える。

35.2.1.1 ニュートンステップ

ニュートン法では、局所的な曲率によってステップの方向と長さの両方を決定する（9.1.5.1 節）。$\mathbf{g}_k = \nabla f(\mathbf{x}_k)$ とおくと、関数 f は $\mathbf{x}_k$ の近傍において変位 $\mathbf{p}$ に関する 2 次のテイラー展開で近似できる。

$$m_k(\mathbf{p}) = f(\mathbf{x}_k) + \mathbf{g}_k^{\top}\mathbf{p} + \tfrac{1}{2}\,\mathbf{p}^{\top}\nabla^2 f(\mathbf{x}_k)\,\mathbf{p}. \tag{35.17}$$

ヘッセ行列 $\nabla^2 f(\mathbf{x}_k)$ が正定値であるとき、モデル m_k は唯一の最小点をもつ狭義凸の鉢状関数となり、その最小点は $\nabla m_k(\mathbf{p}) = \mathbf{g}_k + \nabla^2 f(\mathbf{x}_k)\,\mathbf{p} = \mathbf{0}$ とおくことで求まる。これを $\mathbf{p}$ について解いてその点へ移動すると、**ニュートン法**が得られる。

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \big(\nabla^2 f(\mathbf{x}_k)\big)^{-1}\nabla f(\mathbf{x}_k). \tag{35.18}$$

Figure 35.6 に、1 次元におけるこのようなステップを 1 回分示す。Equation 35.18 にはステップサイズが現れないことに注目してほしい。逆ヘッセ行列が各方向をそれぞれの曲率に応じて再スケーリングしているのである。より広く見れば、ニュートン方向は前処理付きステップ $\mathbf{d} = -\mathbf{P}^{-1}\nabla f$（ただし $\mathbf{P}$ は正定値行列であり、ここでは $\mathbf{P} = \nabla^2 f$）の族に属する。$\nabla f^{\top}\mathbf{d} = -\nabla f^{\top}\mathbf{P}^{-1}\nabla f < 0$ が成り立つため、このような方向はすべて降下方向である。実際、ステップ長をユークリッドノルムではなくヘッセ行列が誘導するノルムで測るとき、ニュートン方向はまさに最急降下方向にほかならない（演習問題 1）。

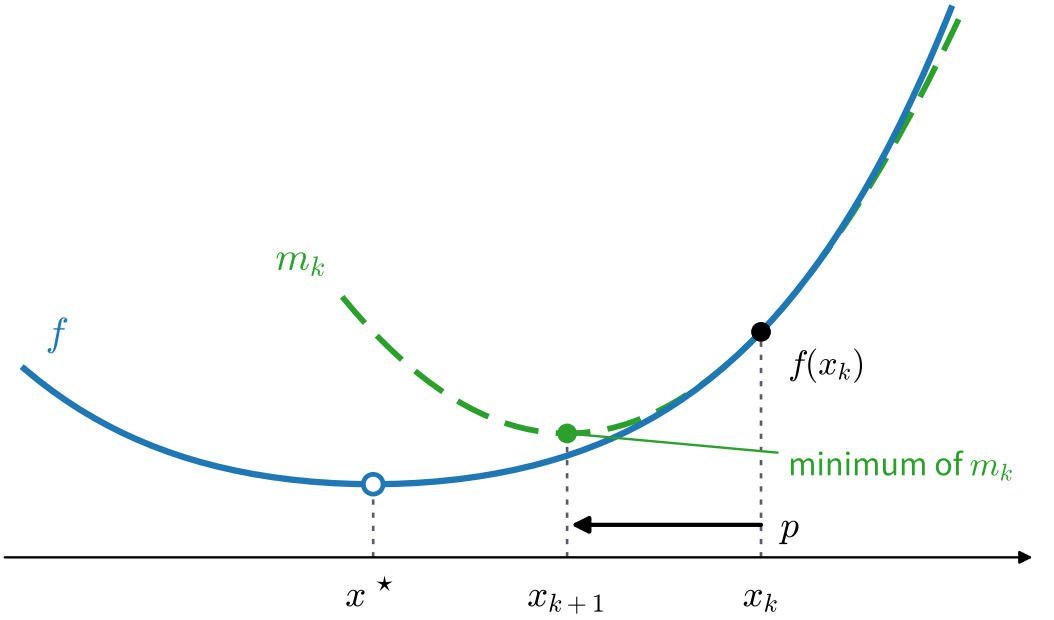


Figure 35.6: $f(x) = \frac{1}{4}x^4 + \frac{1}{2}x^2$ に対する $x_k = 1$ からの 1 回のニュートンステップ。モデル m_k（破線）は f の x_k における値・傾き・曲率を共有し、ステップ p はモデルの最小点 $x_{k+1} = \frac{1}{2}$ へと移動する。f の最小点は $x^\star = 0$ である。

正定値二次関数 $\tfrac{1}{2}\mathbf{x}^{\top}\mathbf{A}\mathbf{x}$ ではテイラーモデルが厳密に一致するため、1 回のニュートンステップで直ちに最小点 $\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{A}^{-1}\mathbf{A}\mathbf{x}_0 = \mathbf{0}$ に到達し、条件数 κ の影響を受けない。その理由は単なる行列の相殺よりも深いところにある。ニュートン法は**アフィン不変**（affine-invariant）であり、その軌跡は座標系の選び方に依存しない。任意の可逆な線形変数変換 $\mathbf{x} = \mathbf{T}\mathbf{y}$ のもとで、$f(\mathbf{T}\mathbf{y})$ に対してニュートン法を実行すると、f に対するニュートン法の反復点をそのまま写した点列が厳密に得られる。対照的に、勾配降下法は非直交なあらゆる $\mathbf{T}$ のもとで経路が変化し、直交行列のスカラー倍に対してのみ実効ステップサイズが再スケーリングされるにとどまる（演習問題 1）。任意の正定値二次関数は何らかの線形変換によって完全な球面へと変換できるため、条件数がニュートン法に影響を与えることはあり得ない。ニュートン法にとって細長い谷とは、引き伸ばされた紙の上に描かれた丸い鉢にすぎないのである。

35.2.1.2 二次収束

一般の非二次な目的関数では、テイラーモデルは局所近似にすぎないため、もはや 1 ステップでは済まない。それでも失われないのが驚異的な収束速度である。反復点が最小点の近傍に入ると、ニュートン法は幾何収束よりもはるかに速く収束し、その証明には微分積分学の基本定理以上のものはほとんど必要ない。$\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^{\star}$ を反復 k における誤差とする。

命題（二次収束）。 **$\nabla f(\mathbf{x}^{\star}) = \mathbf{0}$ とし、$\mathbf{x}^{\star}$ の周りのある球上でヘッセ行列が $\nabla^2 f(\mathbf{x}) \succeq \mu \mathbf{I}$ および $\|\nabla^2 f(\mathbf{x}) - \nabla^2 f(\mathbf{y})\| \le L_2 \|\mathbf{x} - \mathbf{y}\|$ を満たすと仮定する。$\mathbf{x}_k$ がこの球内にあるならば、次が成り立つ。**

$$\|\mathbf{e}_{k+1}\| \;\le\; \frac{L_2}{2\mu}\,\|\mathbf{e}_k\|^2 . \tag{35.19}$$

証明。 $\nabla^2 f_k = \nabla^2 f(\mathbf{x}_k)$ と略記する。$\nabla f(\mathbf{x}^{\star}) = \mathbf{0}$ であるため、ニュートン更新から $\mathbf{x}^{\star}$ を引くと $\mathbf{e}_{k+1} = \mathbf{e}_k - (\nabla^2 f_k)^{-1}\mathbf{g}_k = (\nabla^2 f_k)^{-1}\bigl[\nabla^2 f_k\,\mathbf{e}_k - (\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^{\star}))\bigr]$ が得られる。$\mathbf{x}^{\star}$ から $\mathbf{x}_k$ への線分に微分積分学の基本定理を適用すると、勾配の差は $\int_0^1 \nabla^2 f(\mathbf{x}^{\star} + t\mathbf{e}_k)\,\mathbf{e}_k\,dt$ と表されるため、角括弧内の項は $\int_0^1 \bigl[\nabla^2 f_k - \nabla^2 f(\mathbf{x}^{\star} + t\mathbf{e}_k)\bigr]\mathbf{e}_k\,dt$ となる。$\mathbf{x}_k$ と $\mathbf{x}^{\star} + t\mathbf{e}_k$ の差は $(1-t)\mathbf{e}_k$ であるから、ヘッセ行列のリプシッツ評価より、この積分のノルムは高々 $\int_0^1 L_2 (1-t)\|\mathbf{e}_k\|^2\,dt = \tfrac{L_2}{2}\|\mathbf{e}_k\|^2$ で抑えられる。これと $\|(\nabla^2 f_k)^{-1}\| \le 1/\mu$ を合わせれば証明が完了する。$\blacksquare$

Equation 35.19 が意味するところを読み取るため、両辺に $L_2/(2\mu)$ を掛けると $\tfrac{L_2}{2\mu}\|\mathbf{e}_{k+1}\| \le \bigl(\tfrac{L_2}{2\mu}\|\mathbf{e}_k\|\bigr)^2$ となる。スケールされた誤差は毎ステップ二乗される。初期値が 1 未満であれば誤差は単調に縮小し、それ以降のすべての反復点は球内にとどまる。いったん 0.1 を下回れば、10^{-2}、10^{-4}、10^{-8} へと急落し、各反復で正確な桁数がおよそ倍増していく。これを共通の例題における勾配降下法（精度を 1 桁上げるのに約 1,150 ステップを要する）と比較してほしい。ただし注意点も仮定の中にそのまま書き込まれている。この二次収束率はあくまで局所的なものであり、反復点がすでに $\mathbf{x}^{\star}$ の十分近くに入るまでは何も保証しない。

$\mathbf{x}^{\star}$ から遠い領域では、たとえヘッセ行列が正定値であっても、純粋なニュートンステップはオーバーシュートしうる。これに対する標準的な安全策が**ダンピング**である。更新を $\mathbf{x}_k + \alpha_k \mathbf{p}_k$（ただしニュートン方向は $\mathbf{p}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1}\mathbf{g}_k$）とし、$\alpha_k \le 1$ を Armijo 条件 $f(\mathbf{x}_k + \alpha \mathbf{p}_k) \le f(\mathbf{x}_k) + c\,\alpha\,\mathbf{g}_k^{\top}\mathbf{p}_k$ が成り立つまでバックトラッキングで選ぶ（35.1 節の Armijo 則を一般の方向へ拡張した形）。正定値なヘッセ行列をもつ最小点の近傍では、完全なステップ $\alpha = 1$ が $c < \tfrac{1}{2}$ である限り自動的にこの判定を通過するため、ひとたび二次収束が始まればダンピングによる余分なオーバーヘッドは生じない。

これらの予測をコードで検証するため、ℓ_2 正則化ロジスティック回帰を $n = 200$ 個の人工データ例（$d = 5$ 次元）に対して用いる。

$$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i\, \mathbf{x}_i^\top \mathbf{w})\right) + \frac{\nu}{2}\|\mathbf{w}\|^2, \qquad \nu = 0.1,$$

ここで二値ラベル $y_i = \pm 1$ はロジスティックモデルからサンプリングする（重み減衰の係数には ν を用い、λ は固有値のために取っておく）。以下のセットアップセルでは、参照用の最適解 $\mathbf{w}^\star$ も 4,000 回の勾配ステップ（ステップサイズ $\eta = 0.5$）によって計算しておく。このステップサイズは $2/L$ を十分に下回っている。ここでは $L = \lambda_{\max}(\mathbf{X}^\top\mathbf{X})/(4n) + \nu$ が約 0.4 だからである（演習問題 2）。

```
rng = np.random.default_rng()
n, d, nu = 200, 5, 0.1
w_true = np.array([1.0, -2.0, 1.5, 0.0, 2.0])
X = rng.normal(size=(n, d))
y = np.where(rng.random(n) < 1 / (1 + np.exp(-X @ w_true)), 1.0, -1.0)

def loss(w, X, y, nu):
    return np.logaddexp(0, -y * (X @ w)).mean() + nu / 2 * (w @ w)

def grad_rows(w, X, y, nu):                  # row i: gradient of example i
    s = 1 / (1 + np.exp(y * (X @ w)))        # sigmoid(-y_i x_i^T w)
    return -(y * s)[:, None] * X + nu * w

w_star = np.zeros(d)
for _ in range(4000):                        # full-batch reference optimum
    w_star = w_star - 0.5 * grad_rows(w_star, X, y, nu).mean(axis=0)
```

これで両方の予測を直接検証できる。共通の二次関数 $\mathrm{diag}(1, 1000)$ では、1 回のニュートンステップが、最適な固定ステップサイズの勾配降下法に必要だった $\log 10^6 / \log(1001/999) \approx 6{,}908$ ステップ（35.1.3 節）に取って代わるはずである。また、非二次なロジスティック損失では、局所収束の段階に入れば勾配ノルムの指数が反復ごとにおよそ倍増するはずである。

```
A2 = np.diag([1.0, 1000.0])                  # the running example, kappa = 1000
x = np.array([1.0, 1.0])
x_new = x - np.linalg.solve(A2, A2 @ x)      # one Newton step
rate = (1000 - 1) / (1000 + 1)
print(f'Newton on the quadratic: |x_1| = {np.linalg.norm(x_new):.1e} after one step')
print(f'GD at the optimal step contracts by {rate:.4f}; it needs '
      f'{int(np.ceil(np.log(1e-6) / np.log(rate)))} steps for |x_k| <= 1e-6 |x_0|')
w = np.zeros(d)                              # Newton on the logistic loss
for it in range(6):
    s = 1 / (1 + np.exp(y * (X @ w)))
    g = grad_rows(w, X, y, nu).mean(axis=0)
    hess = (X * (s * (1 - s))[:, None]).T @ X / n + nu * np.eye(d)
    w = w - np.linalg.solve(hess, g)
    print(f'Newton iteration {it + 1}:  |grad| = {np.linalg.norm(g):.1e}')
print(f'max |w - w_star| = {np.abs(w - w_star).max():.1e}')
```

```
Newton on the quadratic: |x_1| = 0.0e+00 after one step
GD at the optimal step contracts by 0.9980; it needs 6908 steps for |x_k| <= 1e-6 |x_0|
Newton iteration 1:  |grad| = 3.6e-01
Newton iteration 2:  |grad| = 4.7e-02
Newton iteration 3:  |grad| = 2.2e-03
Newton iteration 4:  |grad| = 5.4e-06
Newton iteration 5:  |grad| = 3.2e-11
Newton iteration 6:  |grad| = 4.4e-17
max |w - w_star| = 4.4e-16
```

予測はどちらも的中する。二次関数では、1 回のニュートンステップで誤差がゼロになるのに対し、最適な固定ステップの勾配降下法は 1 ステップあたり 0.998 倍ずつしか縮小せず、$6{,}908$ 回の反復を要する。ロジスティック損失では、勾配ノルムは十分の一の数倍程度から始まり、小さくなると指数がおよそ倍増していき、5 回目の反復までに 10^{-8} を下回る。わずか 6 回の反復の後、ニュートン法の反復点は参照用の最適解と約 10^{-15} の誤差で一致し、参照解そのものの精度に到達する。勾配ステップで $4{,}000$ 回かかった計算を、連立一次方程式を 6 回解くだけで成し遂げたのである。

35.2.1.3 1 ステップあたりのコスト

残念ながら、反復回数だけを比較するのはニュートン法を過大評価することになる。ヘッセ行列は d^2 個の成分を含み、Equation 35.18 の連立一次方程式を解くには一般に 1 ステップあたり $O(d^3)$ 回の演算を要する。$d = 5$ であればこのコストは無視できるが、控えめに見積もって 10^8 個のパラメータをもつニューラルネットワークでさえヘッセ行列の成分数は 10^{16} 個に達し、最初の 1 ステップを踏み出す前に厳密なニュートン法は選択肢から外れる（9.1.5.1 節）。実用的な代替手法としては、以下で述べる準ニュートン法のように勾配からヘッセ行列を近似するか、安価な対角前処理行列や構造化前処理行列を用いる（35.5 節）。

35.2.1.4 鞍点

計算コストはニュートン法の唯一の欠点ではなく、最も示唆に富む欠点というわけでもない。ニュートンステップは $\nabla m_k = \mathbf{0}$ を解くため、二次モデルの停留点が極小点であれ、極大点であれ、鞍点であれ、そこへ一気に跳んでしまう。関数 $f(x, y) = \tfrac{1}{2}x^2 + \cos y$（35.1 節）に立ち返ろう。この関数は原点に鞍点をもち、最小値 -1 を $(0, \pi)$ でとる。初期点を $(0.5, 0.3)$ とすると、ここでの勾配は $(0.5, -\sin 0.3) = (0.5, -0.296)$ であり、ヘッセ行列 $\operatorname{diag}(1, -\cos 0.3) = \operatorname{diag}(1, -0.955)$ は不定値となる。すなわち、曲面は x 方向に上向き、y 方向に下向きに曲がっている。この点からニュートンステップが y を動かす向きは、サイズ $\eta = 1$ の通常の勾配ステップと比べてどちらになるだろうか。

```python
def f_s(v):
    return 0.5 * v[0]**2 + np.cos(v[1])

def grad_s(v):
    return np.array([v[0], -np.sin(v[1])])

def hess_s(v):
    return np.diag([1.0, -np.cos(v[1])])

x_n = x_g = np.array([0.5, 0.3])
ys_n, ys_g = [], []
for _ in range(6):
```

```
    x_n = x_n - np.linalg.solve(hess_s(x_n), grad_s(x_n))    # Newton
    x_g = x_g - 1.0 * grad_s(x_g)                            # GD, eta = 1
    ys_n.append(x_n[1])
    ys_g.append(x_g[1])
print('Newton y:', np.array(ys_n[:3]), f'  f = {f_s(x_n):.4f}')
print('GD     y:', np.array(ys_g).round(4), f'  f = {f_s(x_g):.4f}')
```

```
Newton y: [-9.33624961e-03  2.71275924e-07 -6.61744490e-21]   f = 1.0000
GD     y: [0.5955 1.1565 2.0718 2.9489 3.1404 3.1416]   f = -1.0000
```

ニュートン法の最初のステップは x を 0 に、y を $0.3 - \tan 0.3 = -0.0093$ に写し、2 回目のステップは y を 2.7×10^{-7} に写す。ここで起きているのはニュートン法の急速な局所収束（$y - \tan y \approx -y^3/3$ であるため、ここでは三次収束でさえある）であり、反復点を鞍点 $(0, 0)$（$f = 1$）へと一直線に追い込んでいる。これに対し勾配降下法は y を逆方向へ動かし、0.60、1.16、2.07、2.95、3.140、3.1416 を経て最小点 $(0, \pi)$（$f = -1$）に到達する。より精緻なモデルが、かえってより悪い点へと導いたのである。y 方向の曲率が負であるため二次モデルの停留点は鞍点となっているのだが、ニュートン法は鞍点と極小点の違いを区別できない。信頼領域（35.2.3 節）がこの欠陥を解消する。

35.2.2 準ニュートン法

ニュートン法は明示的な二階導関数を必要とする。代わりに、すでに計算している勾配から曲率を推定できないだろうか。あるステップに沿って勾配が急激に変化するとき、目的関数はその方向に大きな曲率をもっているはずである。こうした変化を記録するオプティマイザは、二階導関数を一度も評価することなく、ヘッセ行列のモデルを徐々に組み立てることができる。

35.2.2.1 割線法

1 次元では、この発想は直ちに形になる。直近 2 つの勾配を結ぶ弦の傾きによって f'' を推定すればよい。

$$h_k = \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}, \qquad x_{k+1} = x_k - \frac{f'(x_k)}{h_k}.$$

これが**割線法**（secant method）であり、ニュートンステップの $f''(x_k)$ を勾配の有限差分に置き換えたものにほかならない。関数 $f(x) = e^x - 2x$（最小点は $\log 2 \approx 0.6931$）を考え、$x_0 = 0$ および $x_1 = 1$ から開始しよう。最初の割線の傾きは $h_1 = (e-2) - (1-2) = e - 1 \approx 1.718$ であり、これから $x_2 = 1 - (e-2)/1.718 \approx 0.582$ が得られる。そこから誤差は 1.1×10^{-1}、1.6×10^{-2}、9.3×10^{-4}、7.7×10^{-6}、3.6×10^{-9} と縮小していく。各誤差は直前 2 つの誤差の積のおよそ半分になっており（この半分の係数は最小点における $f'''/(2f'')$ に由来する）、正確な桁数は 1 ステップあたり 1.6 に漸近する倍率で増加する。これはニュートン法の倍率 2 よりわずかに遅いものの、各反復で必要なのは 1 回の勾配評価のみであり、二階導関数は不要である。1 次元の二次関数 $\frac{\lambda}{2}x^2$ では、割線の傾きは厳密に λ と等しくなり、x_1 からのステップは最小点へ直接到達する。

d 次元では、対応する割線条件は 1 本のベクトル方程式にすぎず、$d \times d$ 行列を特定するには到底足りない。1 回のステップで明らかになるのは 1 つの方向に沿った曲率だけだからである。残りの方向を埋めるには、原理に基づいた規則が必要になる。

35.2.2.2 BFGS

準ニュートン法は、逆ヘッセ行列の近似 $\mathbf{M}_k$ を保持し、観測された勾配の変化を用いて各ステップの後にこれを更新する。$\mathbf{x}_k$ から $\mathbf{x}_{k+1}$ へ移動した後、次の割線ペアを記録する。

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \qquad \mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

二次関数 $\tfrac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ では勾配の差分が $\mathbf{y}_k = \mathbf{A}\mathbf{s}_k$ を満たすため、真の逆ヘッセ行列は $\mathbf{y}_k$ を $\mathbf{s}_k$ に写し戻す。そこで、更新後の近似行列 $\mathbf{M}_{k+1}$ にも同じ**割線方程式**（secant equation）を満たすことを課す。

$$\mathbf{M}_{k+1}\,\mathbf{y}_k = \mathbf{s}_k. \tag{35.20}$$

Equation 35.20 を満たすすべての対称行列のうち、**BFGS**（Broyden、Fletcher、Goldfarb、Shanno にちなむ）は、適切に重み付けされたフロベニウスノルムにおいて直前の推定値 $\mathbf{M}_k$ に最も近い行列を選ぶ。

$$\mathbf{M}_{k+1} = (\mathbf{I} - \gamma_k \mathbf{s}_k \mathbf{y}_k^\top)\,\mathbf{M}_k\,(\mathbf{I} - \gamma_k \mathbf{y}_k \mathbf{s}_k^\top) + \gamma_k \mathbf{s}_k \mathbf{s}_k^\top, \qquad \gamma_k = \frac{1}{\mathbf{y}_k^\top \mathbf{s}_k}. \tag{35.21}$$

Equation 35.21 を導出しなくとも、その仕組みはそのまま読み取れる（見やすさのためペアの添字 k は省略する）。右側の因子 $\mathbf{I} - \gamma \mathbf{y}\mathbf{s}^\top$ は、$\mathbf{y}$ をゼロに写す（$\gamma\,\mathbf{s}^\top\mathbf{y} = 1$ であるため）一方、$\mathbf{s}$ に直交するすべてのベクトルをそのまま保つ。したがって第 1 項は $\mathbf{M}_k$ が以前 $\mathbf{y}$ 方向について予測していた内容を消去し、第 2 項が新しい観測を書き込むことで $\mathbf{M}_{k+1}\mathbf{y} = \gamma\,\mathbf{s}\,(\mathbf{s}^\top\mathbf{y}) = \mathbf{s}$ となり、割線方程式が満たされる。

それに劣らず重要な点として、この更新は正定値性を保つため、次の探索方向 $-\mathbf{M}_{k+1}\nabla f(\mathbf{x}_{k+1})$ が引き続き降下方向であることが保証される。任意の非ゼロベクトル $\mathbf{z}$ に対して $\mathbf{w} = (\mathbf{I} - \gamma \mathbf{y}\mathbf{s}^\top)\mathbf{z}$ とおくと、$\mathbf{z}^\top \mathbf{M}_{k+1}\mathbf{z} = \mathbf{w}^\top \mathbf{M}_k \mathbf{w} + \gamma(\mathbf{s}^\top\mathbf{z})^2$ となる。$\mathbf{M}_k \succ 0$ かつ $\gamma > 0$ のとき、両項はともに非負であり、同時に 0 になることはない。なぜなら、$\mathbf{w} = \mathbf{0}$ ならば $\mathbf{z} = \gamma(\mathbf{s}^\top\mathbf{z})\,\mathbf{y}$ となり、これには $\mathbf{s}^\top\mathbf{z} \neq 0$ が必要だからである。この条件 $\gamma > 0$ は**曲率条件**（curvature condition）$\mathbf{y}_k^\top \mathbf{s}_k > 0$ であり、ステップに沿って方向微分が増加したことを意味する。これは任意の狭義凸目的関数では自動的に成り立ち、一般の目的関数では直線探索において Wolfe 条件として知られる第 2 の判定条件によって課される (Nocedal and Wright 2006)。

割線ペアはどれほど速く真の曲率を再構築するだろうか。初期点 $(1, 1)$ から 2 次元の二次関数 $\mathrm{diag}(1, 10)$ に対して、$\mathbf{M}_0 = \mathbf{I}$ および厳密な直線探索 $\alpha = -\mathbf{g}^\top\mathbf{p} / (\mathbf{p}^\top \mathbf{A}\mathbf{p})$ を用いて BFGS を実行し、各ステップ後の $\mathbf{x}$ と $\mathbf{M}$ を確認してみよう。

```
def bfgs_update(M, s, y):
    gamma = 1.0 / (y @ s)
    V = np.eye(len(s)) - gamma * np.outer(s, y)
    return V @ M @ V.T + gamma * np.outer(s, s)

A = np.diag([1.0, 10.0])
x, M = np.array([1.0, 1.0]), np.eye(2)
for k in range(2):
    g = A @ x
    p = -M @ g
    alpha = -(g @ p) / (p @ A @ p)           # exact line search
```

```
    s = alpha * p
    M = bfgs_update(M, s, A @ s)              # y = A s on a quadratic
    x = x + s
    print(f'step {k + 1}: x = {x.round(4)},  M = {M.round(4).tolist()}')
print(f'A^-1 = {np.linalg.inv(A).tolist()},  '
      f'max |M_2 - A^-1| = {np.abs(M - np.linalg.inv(A)).max():.0e}')
```

```
step 1: x = [ 0.8991 -0.009 ],  M = [[1.009, -0.0001], [-0.0001, 0.1]]
step 2: x = [0. 0.],  M = [[1.0, -0.0], [-0.0, 0.1]]
A^-1 = [[1.0, 0.0], [0.0, 0.1]],  max |M_2 - A^-1| = 1e-16
```

わずか 1 ステップの後、$\mathbf{M}_1 \approx \mathrm{diag}(1.009, 0.100)$ は、直前に探索した急な方向に沿ってすでに $\mathbf{A}^{-1}$ に近づいている。2 ステップ目の後には、浮動小数点の丸め誤差の範囲内で $\mathbf{M}_2 = \mathbf{A}^{-1}$ となり、$\mathbf{x}_2 = \mathbf{0}$ に達する。この 2 次元での挙動はそのまま一般化できる。任意の d 次元の正定値二次関数において厳密な直線探索を用いると、各割線方程式はその後の更新を経ても満たされたまま保たれ、各ステップは互いに $\mathbf{A}$ 共役（$\mathbf{s}_i^{\top}\mathbf{A}\mathbf{s}_j = 0$、ただし $i \neq j$）となる。その結果、d ステップ後には $\mathbf{M}_d\mathbf{A} = \mathbf{I}$ となり、BFGS は厳密な最小点に到達する（演習問題 2）。

ただし、有限回での終了は厳密な直線探索を伴う二次関数に特有の性質であり、一般的な保証と受け取るべきではない。非二次な目的関数では直線探索は非厳密になり、真のヘッセ行列もステップごとに変化するうえ、確率的勾配が $\mathbf{y}_k$ を汚染する。2 つのミニバッチ勾配の差は、曲率ではなくサンプリングノイズに支配されることが多いからである。さらに、完全な BFGS は密な $d \times d$ 行列を保持する。**L-BFGS** は直近 m 個の割線ペアのみを保持し、得られる逆ヘッセ行列近似（定数倍した単位行列にランクが高々 $2m$ の補正を加えたもの）を $O(md)$ の計算時間とメモリで適用することで、このメモリコストを回避する (Liu and Nocedal 1989)。決定論的な中規模問題では非常に有効に機能するものの、新たなミニバッチノイズによって勾配の差分が信頼できなくなる状況では依然として扱いにくい。

35.2.3 信頼領域

ニュートン法が $(0.5, 0.3)$ で道を誤ったのは、局所的な二次モデルを空間全体にわたって信頼したからである。そのモデルは単一の点における導関数から構築されたものであり、近傍でしか f を正確に反映しない。信頼領域法はこの限界を文字通りに受け止め、負の曲率の方向を含む完全な曲率情報を保持したまま、モデルを信頼できる領域内にステップを制限する。

35.2.3.1 部分問題

点 $(0.5, 0.3)$ において、$\frac{1}{2}x^2 + \cos y$ の二次モデルは曲率 -0.955 を y 方向にもつ。下向きに際限なく曲がっているため、制約なしの最小点を求めることは意味をなさない。そこで**信頼領域法**は、近似が成り立つと考える範囲を表す半径 Δ_k の球の内部で二次モデルを最小化する。

$$\min_{\|\mathbf{p}\| \le \Delta_k} m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^{\top}\mathbf{p} + \tfrac{1}{2}\mathbf{p}^{\top}\mathbf{B}_k\mathbf{p}. \tag{35.22}$$

ここで $\mathbf{B}_k$ は厳密なヘッセ行列、またはその対称近似である。任意の連続関数はコンパクト集合上で最小値を達成するため、$\mathbf{p}$ を閉球に制限することで、$\mathbf{B}_k$ が負の固有値をもつ場合でも最小点の存在が保証される。さらに都合のよいことに、この制約付き問題の厳密解は明快に特徴付けられる。

命題（部分問題の解）。 **$\mathbf{p}$ を $\|\mathbf{p}\| \le \Delta$ を満たすステップとする。これが Equation 35.22 の解となるのは、次を満たす $\tau \ge 0$ が存在するとき、かつそのときに限られる。**

$$(\mathbf{B}+\tau\mathbf{I})\,\mathbf{p}=-\mathbf{g},\qquad \mathbf{B}+\tau\mathbf{I}\succeq\mathbf{0},\qquad \tau\,(\Delta-\|\mathbf{p}\|)=0.$$

この結果は Nocedal and Wright (2006) の定理 4.1 であり、演習問題 3 でこれらの条件の十分性を証明する。幾何学的には、信頼領域は一様なシフト τ を $\mathbf{B}$ のすべての固有値に加え、シフト後の二次関数をちょうど凸にする。ここで 2 つの場合が生じる。$\mathbf{B}$ が正定値であり、かつ制約なしのニュートンステップがすでに球の内部に収まっているときは、$\tau=0$ となってそのままニュートンステップを採用する。そうでない場合、解は境界 $\|\mathbf{p}\|=\Delta$ 上にあり、τ はスカラー方程式 $\|(\mathbf{B}+\tau\mathbf{I})^{-1}\mathbf{g}\|=\Delta$ の区間 $\tau>-\lambda_{\min}(\mathbf{B})$ における根として定まる（この区間では τ が大きくなるにつれて左辺は狭義単調減少する）。

このスカラー方程式が根をもたないのは、退化した**ハードケース**、すなわち $\mathbf{g}$ が $\mathbf{B}$ の最小固有値に属する固有ベクトルと直交する場合に限られる（演習問題 3 の小問 d で扱う）。35.6 節で見るように、シフト τ は制約 $\tfrac12\|\mathbf{p}\|^2\le\tfrac12\Delta^2$ を課すラグランジュ乗数にほかならない。

反復の間で、半径 Δ_k はモデルが f の実際の減少量をどれほど正確に予測したかに応じて適応的に変化する。Equation 35.22 を解いた後、予測された減少量に対する実際の減少量の比を作る。

$$r_k=\frac{f(\mathbf{x}_k)-f(\mathbf{x}_k+\mathbf{p}_k)}{m_k(\mathbf{0})-m_k(\mathbf{p}_k)}. \tag{35.23}$$

この比が 1 に近いことは、二次モデルが目的関数とよく一致したことを示す。以下の実装では、$r_k>0.1$ である限りステップを採択する。それとは独立に、半径を $\Delta\leftarrow\Delta/4$ へと縮小する操作を $r_k<\tfrac14$ のときに行い、$r_k>\tfrac34$ かつステップが境界 $\|\mathbf{p}_k\|=\Delta_k$ に達したときには半径を 2 倍にする。

35.2.3.2 鞍点における信頼領域ステップ

点 $(0.5,0.3)$ において $\Delta=0.5$ とし、1 回の信頼領域ステップを手計算で求めてみよう。この点では $\mathbf{g}=(0.5,-0.296)$、$\mathbf{B}=\operatorname{diag}(1,-0.955)$ である。$\mathbf{B}+\tau\mathbf{I}$ を半正定値にするには $\tau\ge0.955$ が必要であるため、$\tau=0$ は不可能であり、ステップは球の境界上になければならない。最適性条件の連立方程式は $(1+\tau)p_1=-0.5$ および $(\tau-0.955)p_2=0.296$（制約は $p_1^2+p_2^2=0.25$）となり、その解は $\tau\approx1.60$、$\mathbf{p}\approx(-0.19,0.46)$ である。$(0.31,0.76)$ へ移動すると、f は 1.080 から 0.771 へと下がる。モデルが予測した減少量 0.316 に対して実際の減少量は 0.309 であるため、比は $r=0.98$ となり、ステップは採択されて半径は 2 倍になる。決定的に重要なのは、モデルが減少する負の曲率方向に沿って信頼領域ステップが鞍点から**遠ざかる**向きに進むのに対し、制約なしのニュートンステップ $(-0.5,-0.31)$ は鞍点へ向かって一直線に進んだという点である。

この部分問題の解が数回の反復でどこへ向かうかを見るため、まず `tr_step(g, B, Delta)` を実装する。この関数は、$\mathbf{B}\succ\mathbf{0}$ かつ $\|\mathbf{B}^{-1}\mathbf{g}\|\le\Delta$ のときには制約なしのニュートンステップを返し、そうでないときには τ に関して二分法を行い、$\|(\mathbf{B}+\tau\mathbf{I})^{-1}\mathbf{g}\|=\Delta$ となる点を求める。

```
def tr_step(g, B, Delta):
    """Minimize g p + p B p / 2 over |p| <= Delta; return (p, tau)."""
    eig_min = np.linalg.eigvalsh(B).min()
    if eig_min > 0 and np.linalg.norm(np.linalg.solve(B, -g)) <= Delta:
        return np.linalg.solve(B, -g), 0.0     # Newton step fits: tau = 0
    lo = hi = max(0.0, -eig_min)
    hi += 1e3
    for _ in range(100):                       # bisection for |p(tau)| = Delta
```

```
            tau = (lo + hi) / 2
            p = np.linalg.solve(B + tau * np.eye(len(g)), -g)
            lo, hi = (tau, hi) if np.linalg.norm(p) > Delta else (lo, tau)
        return p, tau
```

初期点 $\mathbf{x} = (0.5, 0.3)$、半径 $\Delta = 0.5$ から開始して f_s に対し信頼領域法を 6 反復実行し、各ステップでの反復点、目的関数値、ステップノルム $\|\mathbf{p}\|$、シフト τ、一致比 r、半径 Δ を出力する。

```
x, Delta = np.array([0.5, 0.3]), 0.5
print(' k        x            f      |p|    tau      r  Delta')
for k in range(6):
    g, B = grad_s(x), hess_s(x)
    p, tau = tr_step(g, B, Delta)
    r = (f_s(x) - f_s(x + p)) / -(g @ p + 0.5 * p @ B @ p)
    print(f'{k:2d} ({x[0]:6.3f},{x[1]:6.3f}) {f_s(x):7.4f} '
          f'{np.linalg.norm(p):6.3f} {tau:6.3f} {r:6.2f} {Delta:5.2f}')
    if r > 0.1:                                 # accept the step
        x = x + p
    if r < 0.25:
        Delta /= 4                              # poor prediction: shrink
    elif r > 0.75 and np.linalg.norm(p) > 0.99 * Delta:
        Delta *= 2                              # good, and on the boundary
print(f'final x = ({x[0]:.4f}, {x[1]:.4f}),  f = {f_s(x):.4f}')
```

```
 k        x            f      |p|    tau      r  Delta
 0 ( 0.500, 0.300)  1.0803  0.500  1.596   0.98  0.50
 1 ( 0.307, 0.761)  0.7711  1.000  1.419   0.87  1.00
 2 ( 0.180, 1.753) -0.1652  2.000  0.311   0.41  2.00
 3 ( 0.043, 3.749) -0.8205  0.696  0.000   0.88  2.00
 4 ( 0.000, 3.054) -0.9962  0.088  0.000   1.00  2.00
 5 ( 0.000, 3.142) -1.0000  0.000  0.000   1.00  2.00
final x = (0.0000, 3.1416),  f = -1.0000
```

1 行目は先ほどの手計算と一致している。半径は 0.5 から 1、さらに 2 へと倍増し、3 回目のステップ（比 0.41）は半径を変えずに採択される。3 番目の反復点 $(0.18, 1.75)$ に達するまでに、アルゴリズムはヘッセ行列が正定値となる領域に入っている。次の反復点以降は制約なしのニュートンステップが $\tau = 0$ のまま信頼領域の球内に収まるため、手法は継ぎ目なくニュートン法へと切り替わり、最小点 $(0, \pi)$（$f = -1$）へ二次収束する。6 回の反復で十分であり、Figure 35.7 に両者の軌跡の対比を示す。

信頼領域の部分問題を厳密に解くには、$\mathbf{B}_k + \tau \mathbf{I}$ を τ の各試行値に対して分解する必要がある。そのため大規模な実装では、1 回あたり勾配評価 1 回分とほぼ同じコストで計算できるヘッセ行列・ベクトル積を用いて近似的に解く（34.5 節）。こうした高速化を用いてもなお、大規模でノイズの多い深層学習の目的関数において、BFGS や信頼領域法が確率的勾配降下法に取って代わることはない。

関数 $\frac{1}{2}x^2 + \cos y$ の鞍点は、ここまでで 2 つのオプティマイザの足をすくった。勾配降下法は直線 $y = 0$ 上の初期点から開始したときにそこで停滞し、ニュートン法は一般的な初期点からそこへ一直線に飛び込んだ。信頼領域はそこから脱出できるものの、各局所モデルを真の目的関数と突き合わせて検証することが不可欠であった。しかし、こうした予防策が一切不要となる大きな関数クラスが存在する。鞍点も偽の盆地ももたず、すべての停留点が大域的最小点となる関数群である。そのような関数を見分けることが 35.3 節の主題である。

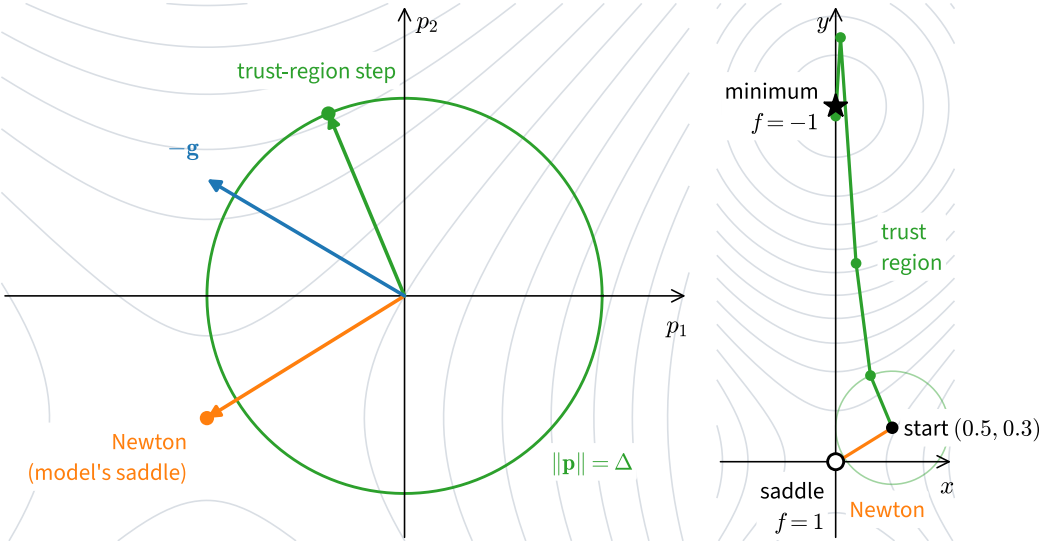


Figure 35.7: 点 $(0.5, 0.3)$ から開始した $\frac{1}{2}x^2 + \cos y$（ヘッセ行列は $\mathrm{diag}(1, -0.96)$）に対するニュートン法と信頼領域法。(a) 局所的な二次モデルは鞍面をなす。ニュートンステップはその停留点へと移動するのに対し、半径 $\Delta = 0.5$ の球上でモデルを最小化すると、負の曲率方向に沿って鞍点から遠ざかる。(b) ニュートン法は鞍点 $(0, 0)$（$f = 1$）へ二次収束する一方、信頼領域法（初期半径を表示）は最小点 $(0, \pi)$（$f = -1$）に 6 ステップで到達し、その間に半径を $0.5, 1, 2$ と拡大していく。

35.2.4 まとめ

ニュートン法は各反復点において f の局所的な二次モデルを最小化する。条件数にかかわらず任意の正定値二次関数を 1 ステップで解き、正定値かつリプシッツ連続なヘッセ行列をもつ最小点の近傍では反復ごとに誤差を二乗する。ただし、その速さには 2 つの障壁が伴う。各ステップに $O(d^2)$ のメモリと $O(d^3)$ の演算量を要するためニューラルネットワークでは到底扱えないこと、そして制約なしのステップが極小点と同じくらい強く鞍点にも引き寄せられることである。BFGS は過去のステップ差分と勾配差分から逆ヘッセ行列を再構築することで明示的な二階導関数を不要にし、d ステップで d 次元の二次関数上の逆ヘッセ行列を完全に復元する。信頼領域法は、各ステップを球内に閉じ込め、予測された減少量と f の実際の減少量を比較して球を拡大・縮小することで、任意の二次モデルを安全に利用できるようにする。

35.2.5 演習問題

1. **再パラメータ化と前処理。** $\mathbf{T}$ を可逆行列、$h(\mathbf{y}) = f(\mathbf{T}\mathbf{y})$ とし、$\mathbf{x} = \mathbf{T}\mathbf{y}$ を座標変換とする。
   1. ∇h と $\nabla^2 h$ を f の導関数を用いて表せ。さらに、ニュートン法の反復 Equation 35.18 を h に対して $\mathbf{y}_0 = \mathbf{T}^{-1}\mathbf{x}_0$ から開始すると、$\mathbf{y}_k = \mathbf{T}^{-1}\mathbf{x}_k$（ここで $\mathbf{x}_k$ は f に対するニュートン法の反復点である）を満たすことを示せ。
   2. ステップサイズ η の勾配降下法を h に対して実行し、すべての反復点を $\mathbf{x}_k = \mathbf{T}\mathbf{y}_k$ によって元の座標へ写す。写された反復点が $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\mathbf{P}^{-1}\nabla f(\mathbf{x}_k)$（ただし $\mathbf{P}$ はある正定値行列）に従うことを示し、$\mathbf{P}$ を求めよ。問題 1b により、この方向はノルム $\|\cdot\|_{\mathbf{P}}$ における最急降下方向である。$\|\mathbf{x}\|_{\mathbf{P}}$ は座標 $\mathbf{y}$ ではどのように表されるか。
   3. $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\top}\mathbf{A}\,\mathbf{x}$（$\mathbf{A} = \mathrm{diag}(1, 1000)$、ニュートン法の項で扱った $\kappa = 1000$ の二次関数）に対し、対角行列 $\mathbf{T}$ として、h 上で $\eta = 1$ の勾配降下法が 1 ステップで最小点

に到達するようなものを求めよ。そのような $\mathbf{T}$ を一般の正定値行列 $\mathbf{A}$ に対して構成するには何が必要か。

2. [code] **二次関数上の BFGS。** ●$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^{\top}\mathbf{A}\mathbf{x} - \mathbf{b}^{\top}\mathbf{x}$（$\mathbf{A}$ は正定値）とし、BFGS Equation 35.21 を厳密な直線探索のもとで $\mathbf{M}_0 = \mathbf{I}$ から実行する。
   1. $\mathbf{p}_k$ に沿った厳密な直線探索によって $\nabla f(\mathbf{x}_{k+1})^{\top}\mathbf{p}_k = 0$ となり、さらに $\mathbf{y}_k = \mathbf{A}\mathbf{s}_k$ が成り立つことを示せ。
   2. 遺伝性を示せ。すなわち、$\mathbf{M}_k\mathbf{y}_j = \mathbf{s}_j$ がすべての $j < k$ について成り立ち、かつステップ $\mathbf{s}_0, \dots, \mathbf{s}_{k-1}$、$\mathbf{s}_k$ が互いに $\mathbf{A}$ 共役であるならば、$\mathbf{M}_{k+1}\mathbf{y}_j = \mathbf{s}_j$ がすべての $j \le k$ について成り立つことを示せ。（ヒント：$\mathbf{s}_j^{\top}\mathbf{y}_k = \mathbf{s}_j^{\top}\mathbf{A}\mathbf{s}_k = 0$ が $j < k$ に対して成り立つ。）以上から、線形独立な d 回のステップの後に $\mathbf{M}_d\mathbf{A} = \mathbf{I}$ となることを結論せよ。
   3. 本文の BFGS ループを $d = 6$ 次元で、固有値が 1 から 1000 にわたるランダムに回転させた二次関数に対して実行せよ。各ステップの後に $\max_{j \le k} \|\mathbf{M}_{k+1}\mathbf{y}_j - \mathbf{s}_j\|$ を出力し、これが丸め誤差の水準にとどまり、かつ $\mathbf{M}_6 = \mathbf{A}^{-1}$ となることを確認せよ。同じ 6 回の勾配評価の後、最適な固定ステップサイズを用いた勾配降下法は最小点からどれだけ離れているか。
   4. 対称ランク 1 更新 $\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{(\mathbf{s}-\mathbf{M}_k\mathbf{y})(\mathbf{s}-\mathbf{M}_k\mathbf{y})^{\top}}{(\mathbf{s}-\mathbf{M}_k\mathbf{y})^{\top}\mathbf{y}}$ もまた割線方程式を満たす。2×2 の具体例によって、$\mathbf{y}^{\top}\mathbf{s} > 0$ であっても正定値性が保たれるとは限らないことを示し、本文で述べた 2 つの確認事項のうちどちらが成り立たなくなるかを述べよ。
3. [code] **鞍点における信頼領域。** ●モデル $m(\mathbf{p}) = \mathbf{g}^{\top}\mathbf{p} + \tfrac{1}{2}\mathbf{p}^{\top}\mathbf{B}\mathbf{p}$ において、$\mathbf{B} = \mathrm{diag}(1, -1)$、$\mathbf{g} = (\tfrac{1}{2}, -\tfrac{3}{10})$ とする。
   1. m が $\mathbb{R}^2$ 上で下に非有界であること、およびニュートン点 $-\mathbf{B}^{-1}\mathbf{g} = (-\tfrac{1}{2}, -\tfrac{3}{10})$ が最小点ではなく m の鞍点であることを示せ。
   2. 部分問題の命題の条件が十分条件であることを証明せよ。すなわち、$(\mathbf{B} + \tau\mathbf{I})\mathbf{p} = -\mathbf{g}$ が $\tau \ge 0$、$\mathbf{B} + \tau\mathbf{I} \succeq \mathbf{0}$、$\tau(\Delta - \|\mathbf{p}\|) = 0$ とともに成り立つならば、任意の $\|\mathbf{q}\| \le \Delta$ が $m(\mathbf{q}) \ge m(\mathbf{p}) + \tfrac{\tau}{2}(\|\mathbf{p}\|^2 - \|\mathbf{q}\|^2) \ge m(\mathbf{p})$ を満たすことを証明せよ。（ヒント：$m(\mathbf{q}) - m(\mathbf{p})$ を $\mathbf{p}$ の周りで展開し、$\mathbf{g} = -(\mathbf{B} + \tau\mathbf{I})\mathbf{p}$ を用いる。）
   3. $\Delta = \tfrac{1}{2}$ に対し、$\tau \ge 1$ を $\|(\mathbf{B} + \tau\mathbf{I})^{-1}\mathbf{g}\| = \Delta$ に関する二分法によって求め（答えは $\tau \approx 1.65$、$\mathbf{p} \approx (-0.19, 0.46)$ となる）、このステップが負の曲率をもつ座標を増加させるのに対し、ニュートンステップはそれを減少させることを確認せよ。
   4. $\mathbf{g}$ を $(\tfrac{1}{2}, 0)$ に置き換えよ。$\|(\mathbf{B} + \tau\mathbf{I})^{-1}\mathbf{g}\| = \tfrac{1}{2}/(1 + \tau) \le \tfrac{1}{4}$ がすべての $\tau > 1$ について成り立ち、したがって境界の方程式は $\tau > 1$ の範囲に解をもたないことを示せ。球上の最小点を直接求め（それらは $\tau = 1$ および $p_2 = \pm\sqrt{\Delta^2 - p_1^2}$ を満たす）、勾配からは見えない負の曲率方向に対して球が何をするかを 1 文で説明せよ。

35.3 凸性

訓練が終了したとき、停止した点について通常わかっていることは 1 つしかない。そこではもはや損失が減少しないということである。安全なステップサイズを用いた勾配降下法は、勾配がほぼゼロになるそのような点に確実に到達し（35.1 節）、ニュートン法はさらに速くそこへ到達する。たとえその停留点が結果的に鞍点であったとしてもである（35.2 節）。しかし、どちらの保証からも、はるかに良い点が別の場所に隠れているかどうかはわからない。本節では、停留点が実際に利用可能な最良の点となるのはいつか、そして勾配降下法がそこへどれほど速く収束するかを問う。

単純な 1 次元の実験から、これが決して空論ではないことがわかる。500 個のランダムな初期点からまったく同じ勾配降下法のループを、まずボウル $f(x) = (x - \tfrac{1}{2})^2$ に対して、次に傾いた二重井戸関数 $\phi(x) = (x^2 - 1)^2 + x/2$ に対して実行するとしよう。ボウルでは、すべての試行が同じ最小点に到達する。一方、二重井戸関数では、ほぼ半数の試行が浅い方の谷に捕らわれ、そ

の損失は大域的最小値よりほぼ 1 だけ高い値にとどまる。立ち往生したこれらの試行でも勾配はゼロになっており（停留性の保証が約束するのはそれだけである）、勾配だけを見ても近くにより深い谷が存在する手がかりはまったく得られない。

これら 2 つの関数の違いは、ただ 1 つの幾何学的性質に帰着する。ボウル上の 2 点間に引いた弦はすべてグラフの上側を通るが、二重井戸関数には曲線を下側へ横切る弦が存在する。弦が常にグラフ上またはその上側にある関数は**凸**（convex）と呼ばれ、そのような関数に対しては冒頭の問いにこれ以上ないほど明快な答えが与えられる。すなわち、すべての極小点は大域的最小点である。その証明はわずか 3 行で済む（35.3.2.2 節）。さらにその背後には、いっそう有用な事実がある。凸関数の任意の接線は定義域全体にわたってグラフの下側にとどまるため、ただ 1 点における局所的な情報がいたるところで関数を制約するのである。この接線による下界を勾配降下法の反復点に沿って足し合わせれば、任意の初期点から成り立つ収束率が得られる。また、確率変数について平均をとれば、確率論および情報理論の章で中心的な役割を果たすイェンセンの不等式（Jensen's inequality）が得られる。

本節ではかなり広い範囲を扱うが、すべての結果が同じ重みをもつわけではない。特に常に念頭に置いておきたい考え方は 3 つある。接線による下界、極小点が大域的最小点に一致するという事実、そして勾配降下法の線形収束率（その証明は、凸性が成り立たなくても生き残るただ 1 つの不等式に依拠している）である。まず凸集合と凸関数に対する 3 つの同値な判定条件から始め、それらの主要な帰結を導き、凸損失を見分けるための簡潔な演算規則をまとめる。深層ネットワークの損失は凸ではないため、最後にこの理論のどの部分が深層ネットワークにも引き継がれるのかを考察して締めくくる。急いでいる読者は、35.3.2 節の弦条件と接線条件だけを読み、そのまま 35.3.3 節へ進んでよい。本節を通して 2 つの例を繰り返し用いる。収束率を測るための本章の通し例である二次関数 $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ と、あらゆる凸性の判定条件をことごとく破る二重井戸関数 ϕ である。

35.3.1 凸集合

凸関数は定義域の点を結ぶ線分に沿って定義されるため、まずはそれらの点が属する集合から始めよう。損失の定義域、制約付き問題の実行可能領域（feasible region）、そしてすべての最小点からなる集合は、いずれも集合である。幸い、必要な道具立てはごく少ない。1 つの定義、3 つの代表的な例の族、そして既存の凸集合から新たな凸集合を構成する 1 つの規則だけである。

35.3.1.1 定義と例

幾何学的には、任意の 2 点を結ぶまっすぐな線分が完全にその集合の内部にとどまるとき、その集合は凸である。形式的には、集合 $C \subseteq \mathbb{R}^d$（d 次元の点からなる集合）が**凸**であるとは、次の条件を満たすことをいう。

$$\theta \mathbf{x} + (1-\theta)\,\mathbf{y} \in C \qquad \text{for all } \mathbf{x}, \mathbf{y} \in C \text{ and all } \theta \in [0,1]. \tag{35.24}$$

θ が 1 から 0 へ動くにつれて、結合 $\theta\mathbf{x} + (1-\theta)\mathbf{y}$ は $\mathbf{x}$ から $\mathbf{y}$ までの線分を描く。Figure 35.8 では、凸集合と非凸な三日月形領域を対比し、さらに深層学習で最も頻繁に現れる 2 つの凸集合である確率単体（probability simplex）と半空間（half-space）を示している。

この定義によって何が除外されるかを見るために、円環領域 $\{\mathbf{x} : 1 \le \|\mathbf{x}\| \le 2\}$ を考えよう。この集合は $(\frac{3}{2}, 0)$ と $(-\frac{3}{2}, 0)$ の両方を含むが、その中点である原点は中央の穴の中にあり集合に含まれない。中点がたった 1 つ欠けているだけで、Equation 35.24 は破られる。

本章で出会う凸集合のほとんどは、3 つの族で網羅される。第 1 に、任意の超平面（hyperplane）$\{\mathbf{x} : \mathbf{a}^\top \mathbf{x} = b\}$ と任意の半空間 $\{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \le b\}$ は、$\mathbf{a}^\top \mathbf{x}$ が線形であるため凸である。実際、b に等しい（あるいは高々 b である）2 つの数の重み付き平均は、再び b に等しい（あ

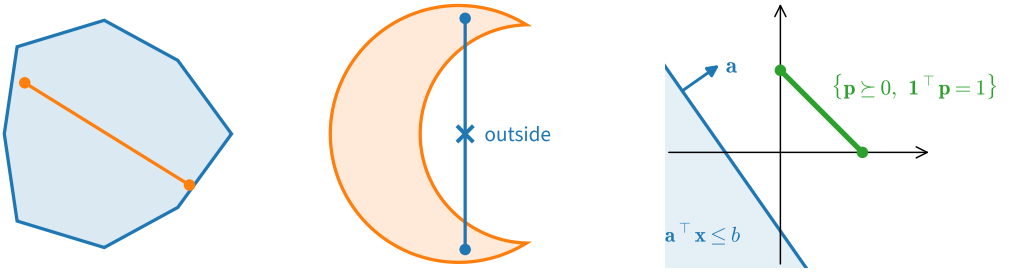


Figure 35.8: 凸集合と非凸集合の比較。左：任意の 2 点間の線分が内部にとどまる凸集合。中央：2 点間の弦が集合の外部を通る非凸な三日月形領域。右：確率単体 $\{\mathbf{p} \succeq 0,\ \mathbf{1}^\top \mathbf{p} = 1\}$ と半空間 $\{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \le b\}$（いずれも凸集合）。

るいは高々 b である）。第 2 に、任意のノルム球 $\{\mathbf{x} : \|\mathbf{x}\| \le r\}$ は、三角不等式と斉次性 $\|\theta \mathbf{x} + (1-\theta)\mathbf{y}\| \le \theta\|\mathbf{x}\| + (1-\theta)\|\mathbf{y}\| \le r$ により凸である。このただ 1 つの確認だけで、重み減衰や勾配クリッピングに現れる ℓ_2 球も、スパース性を誘導する正則化に現れる ℓ_1 球もカバーされる。第 3 は確率単体 $\Delta = \{\mathbf{p} \in \mathbb{R}^d : \mathbf{p} \succeq 0,\ \mathbf{1}^\top \mathbf{p} = 1\}$ であり、ここで $\mathbf{p} \succeq 0$ はすべての成分が非負であることを意味する。アテンション重み、クラス確率、混合重みはいずれも Δ 上に存在し、その凸性は次に述べる規則からただちに従う。

35.3.1.2 既存の凸集合からの構成

有限個か無限個かを問わず、任意の凸集合の族の共通部分は常に凸集合である。2 点が共通部分に属するならば、その族のすべての集合に属するため、それらを結ぶ線分もすべての集合の内部にとどまり、したがって共通部分の内部にとどまる。直感的には、凸集合にはくぼみがなく、くぼみのない集合を別のくぼみのない集合で切り取ってもくぼみが生じることはない。対照的に、合併集合にはそのような保証はない。$[0,1] \cup [2,3]$ は 2 つの凸区間の合併であるが、1 から 2 までの線分は集合の外へ出てしまう。

この共通部分の規則 1 つで、中心的な 2 つの集合の凸性が一度に示される。確率単体は 1 つの超平面（$\mathbf{1}^\top \mathbf{p} = 1$）と d 個の半空間（$-p_i \le 0$）の共通部分であるため凸である。同様に、対称半正定値（positive semidefinite、PSD）行列の集合も、無限個の半空間の共通部分であるため凸である。実際、固定した各ベクトル $\mathbf{z}$ に対して、制約 $\mathbf{z}^\top \mathbf{A} \mathbf{z} \ge 0$ は $\mathbf{A}$ の成分に関する 1 つの線形不等式にほかならない。

内側から外側へ向かって凸集合を構成することもできる。集合 S の**凸包**（convex hull）とは、重み付き平均 $\sum_i \theta_i \mathbf{x}_i$（ただし $\mathbf{x}_i \in S$、$\theta_i \ge 0$、$\sum_i \theta_i = 1$）全体の集合である。凸包は凸集合であり、実際、S を含む最小の凸集合となる（演習問題 2）。この見方に立てば、確率単体とは標準基底ベクトル $\mathbf{e}_1, \ldots, \mathbf{e}_d$ の凸包にほかならない。すなわち、あらゆる確率分布は確定的な状態の重み付き平均である。

35.3.2 凸関数

凸集合にくぼみがないのと同様に、凸関数とは、グラフが上向きに湾曲して第 2 の谷へ落ち込むことのない関数である。以下では、それぞれ 0 階・1 階・2 階の微分を用いる 3 つの同値な凸性の判定条件を導き、続いて 2 つの発展的な概念を扱う。勾配降下法の収束の速さを支配する強凸性（strong convexity）と、角をもつ損失を扱うための劣勾配（subgradient）である。3 つの判定条

件のうち 1 つだけを覚えるなら、2 番目の条件にしてほしい。接線による下界こそ、本節のほぼすべての証明を動かす原動力だからである。

35.3.2.1 弦条件

第 1 の判定条件は微分を一切必要としないため、滑らかな目的関数だけでなく絶対値関数やヒンジ損失にも同じように容易に適用できる。関数 $f : C \to \mathbb{R}$（C は凸な定義域）が**凸**であるとは、グラフが任意の 2 点を結ぶ弦の上または下側にあることをいう。

$$f\left(\theta \mathbf{x} + (1-\theta)\,\mathbf{y}\right) \;\le\; \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y}) \qquad \text{for all } \mathbf{x}, \mathbf{y} \in C,\ \theta \in [0,1]. \tag{35.25}$$

ここで左辺は、f を $\mathbf{x}$ と $\mathbf{y}$ を結ぶ線分上の点で評価した値であり、右辺はその真上にある、$(\mathbf{x}, f(\mathbf{x}))$ と $(\mathbf{y}, f(\mathbf{y}))$ を結ぶまっすぐな弦の高さを与える。具体例で確認してみよう。$f(x) = x^2$ において $x = 0$、$y = 2$、$\theta = \tfrac{1}{2}$ とすると、グラフの高さは $f(1) = 1$ であるのに対し、弦の高さは $\tfrac{1}{2}\cdot 0 + \tfrac{1}{2}\cdot 4 = 2$ であり、確かに $1 \le 2$ が成り立つ。任意の x、y、θ に対して二乗を展開すると、弦とグラフの高さの差は次のようになる。

$$\theta x^2 + (1-\theta) y^2 - \left(\theta x + (1-\theta) y\right)^2 = \theta(1-\theta)(x-y)^2 \;\ge\; 0.$$

したがって x^2 は凸であり、その差は 2 点間の距離の二乗に比例して大きくなる。このまったく同じ式に、まもなくイェンセンの不等式における分散として再び出会うことになる。

関連する 2 つの概念にもここで名前を付けておこう。不等式 Equation 35.25 が $\mathbf{x} \neq \mathbf{y}$ かつ $\theta \in (0,1)$ のとき常に狭義の不等号で成り立つならば、f を**狭義凸**（strictly convex）と呼ぶ。また、$-f$ が凸であるならば、f を**凹**（concave）と呼ぶ。線形関数 $f(x) = x$ は凸（かつ凹）であるが、すべての弦がグラフそのものに重なるため狭義凸ではないことに注意しよう。また、なぜ定義域 C が凸でなければならないかも確認しておきたい。その要請がなければ、中間の点 $\theta\mathbf{x} + (1-\theta)\mathbf{y}$ がそもそも f の定義される領域に属さない可能性があるからである。

凸関数と凸集合は、同じ幾何学的構造の表裏をなしている。関数 f の**エピグラフ**とは、そのグラフ上および上側の領域 $\{(\mathbf{x}, t) : t \ge f(\mathbf{x})\}$ のことである。エピグラフ内の 2 点を結ぶ弦がグラフの上側にとどまるのはまさに Equation 35.25 が成り立つときであるため、関数 f が凸であることと、そのエピグラフが凸集合であることは同値である。また凸性は、**劣レベル集合**（sublevel set）$\{\mathbf{x} \in C : f(\mathbf{x}) \le c\}$ を通じても関数から集合へと受け継がれる。実際、$f(\mathbf{x}) \le c$ かつ $f(\mathbf{y}) \le c$ であれば、弦不等式からただちに $f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \le \theta c + (1-\theta)c = c$ が得られる。任意の不等式制約 $g(\mathbf{x}) \le 0$（ただし g は凸関数）が自動的に凸な実行可能領域を切り出すのはこのためである。Figure 35.9(a) に放物線に対するこれら両方の集合を示す。

ただし注意してほしいのは、この逆は成り立たないという点である。$f(x) = \sqrt{|x|}$ の劣レベル集合はすべて凸区間 $\{|x| \le c^2\}$ となるが、f 自体は凸ではない。$x = 0$、$y = 1$、$\theta = \tfrac{1}{2}$ において、グラフは $\sqrt{1/2} \approx 0.71$ にあるのに対し、弦はその下側の 0.5 を通る（Figure 35.9(b)）。

35.3.2.2 極小点は大域的最小点

驚くべきことに、弦条件だけでも冒頭で立てた問いに答えるには十分である。Figure 35.10 は 2 つのランドスケープを対比している。凸目的関数における単一の吸引領域と、二重井戸関数における競合する 2 つの吸引領域である。

命題（極小点は大域的最小点）。 **f が凸集合 C 上で凸であるならば、f の C 上における極小点はすべて大域的最小点である。**

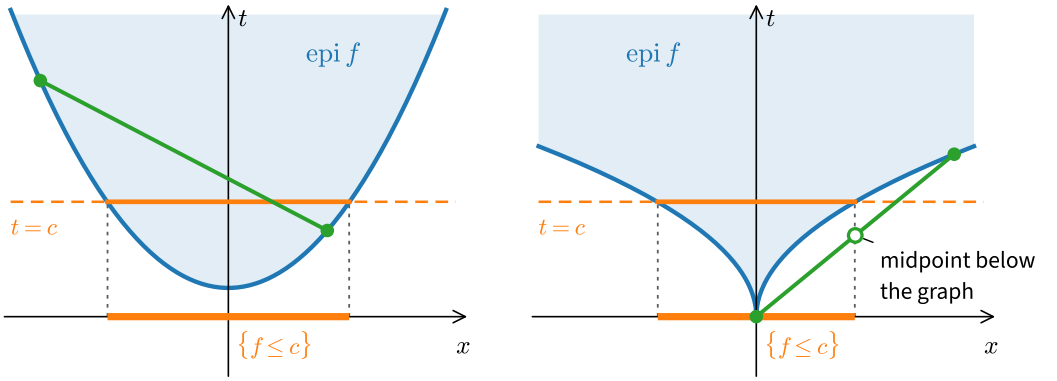


Figure 35.9: エピグラフと劣レベル集合。(a) 凸関数 $f(x) = \frac{1}{2}x^2 + \frac{1}{4}$ のエピグラフ（網掛け部分、上方へ限りなく続く）は凸集合であり、その任意の 2 点間の弦は内部にとどまる。エピグラフを高さ c で切り、x 軸へ射影すると劣レベル集合 $\{f \leq c\}$ が得られる。(b) $f(x) = \sqrt{|x|}$ の劣レベル集合はすべて区間となるが、エピグラフは凸ではない。弦の中点がグラフの下側に出てしまう。

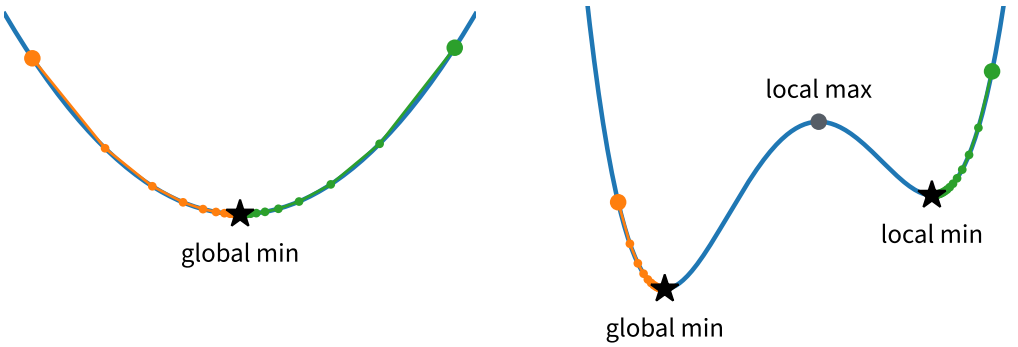


Figure 35.10: (a) 凸目的関数では、小さな固定ステップサイズの勾配降下法がどちらの初期点からも大域的最小点に到達する。(b) 二重井戸関数では、異なる初期点から同じ更新を行うと異なる極小点に到達する。2 つの吸引領域は極大点で接している。

証明。 $\mathbf{x}^\star$ を極小点とし、$\mathbf{y} \in C$ を任意の別の点とする。十分小さい $\theta > 0$ に対して、点 $\mathbf{z}_\theta = (1-\theta)\mathbf{x}^\star + \theta \mathbf{y}$（$\mathbf{y}$ へ向かう線分上の点）は $\mathbf{x}^\star$ が最小となる近傍内に収まる。局所最小性と弦不等式を組み合わせると次を得る。

$$f(\mathbf{x}^\star) \;\le\; f(\mathbf{z}_\theta) \;\le\; (1-\theta) f(\mathbf{x}^\star) + \theta f(\mathbf{y}).$$

両辺から $(1-\theta) f(\mathbf{x}^\star)$ を引き、$\theta > 0$ で割れば、$f(\mathbf{x}^\star) \le f(\mathbf{y})$ が得られる。■

幾何学的な描像も同様に明快である。もし真に小さな値を与える点 $\mathbf{y}$ が C 内のどこかに存在したならば、$\mathbf{x}^\star$ から $\mathbf{y}$ へ向かう弦はただちに下向きに傾き、グラフを $f(\mathbf{x}^\star)$ より下へ、しかも $\mathbf{x}^\star$ のいくらでも近くで押し下げることになり、局所最小性に矛盾する。

ここで冒頭の実験を実際に動かし、この命題が数値的にどう現れるかを見てみよう。同じ勾配降下法のループ（ステップサイズ 0.05、600 ステップ）をボウル $(x - \tfrac{1}{2})^2$ と二重井戸関数 ϕ の両方に適用し、同じ 500 個の点（$[-1.5, 1.5]$ 上の一様分布から抽出）から開始する。ステップサイズが小さいため、各反復点は中央の山を飛び越えることなく坂を転がり落ちる。したがって ϕ 上の試行は、極大点を境にして初期点が属する側の井戸に落ち着くはずである。その極大点を $\phi'(x) = 4x^3 - 4x + \tfrac{1}{2}$ の根として計算し、それより右側から始まる初期点の割合が、浅い右側の井戸に収束する試行の割合と一致するかを確かめる。

```
def gd(grad_fn, x, eta=0.05, steps=600):
    for _ in range(steps):
        x = x - eta * grad_fn(x)
    return x

phi = lambda x: (x**2 - 1)**2 + 0.5 * x        # tilted double well
dphi = lambda x: 4 * x * (x**2 - 1) + 0.5      # its derivative
rng = np.random.default_rng()
inits = rng.uniform(-1.5, 1.5, size=500)       # 500 random starting points
bowl = gd(lambda x: 2 * (x - 0.5), inits)      # convex: (x - 1/2)^2
well = gd(dphi, inits)
left, right = well[well < 0], well[well > 0]
print(f'bowl: all {bowl.size} runs end at x = {bowl.mean():.6f}'
      f' (spread {np.ptp(bowl):.0e})')
for name, w in (('left ', left), ('right', right)):
    print(f'well, {name}: {w.size} runs, x = {w.mean():.4f}, phi = {phi(w.mean()):.4f}')
x_max = np.sort(np.roots([4, 0, -4, 0.5]).real)[1]   # local max of phi
print(f'local max at {x_max:.4f}: predicted share right {(1.5 - x_max) / 3:.3f},'
      f' observed {right.size / inits.size:.3f}')
```

```
bowl: all 500 runs end at x = 0.500000 (spread 7e-16)
well, left : 254 runs, x = -1.0575, phi = -0.5148
well, right: 246 runs, x = 0.9304, phi = 0.4833
local max at 0.1271: predicted share right 0.458, observed 0.492
```

ボウルでは、500 回の試行すべてが数値丸めの範囲内で $x = 0.5$ に収束する。一方、二重井戸関数では試行が 2 つの陣営にくっきりと分かれる。一方は $x \approx -1.0575$（$\phi \approx -0.515$）、もう一方は $x \approx 0.9304$（$\phi \approx 0.483$）であり、後者はほぼ 1 だけ悪い。2 つの吸引領域を隔てる極大点は $x \approx 0.127$ にあるため、一様な初期点のうち $(1.5 - 0.127)/3 \approx 0.458$ の割合がその右側に落ちる。ランダムな抽出を繰り返しても、観測される割合は 46% の前後数パーセントポイント以内に収まり、500 個の初期点によるサンプリング誤差の範囲で予測と一致する。

両方の実験で動かしたのはまったく同じオプティマイザであり、変えたのは目的関数だけである。凸性は第 2 の吸引領域を消し去るが、それ自体が勾配降下法による最小点への到達を保証するわけではない。到達のためには、35.1.1 節で見たように、滑らかさと安全なステップサイズが依然として必要である。

35.3.2.3 一階条件

弦条件には微分を必要としないという利点があるが、最適化の証明は勾配を材料として組み立てられる。f が微分可能であるとき、凸性は同値な一階の形式をとる。すなわち、任意の点における接超平面が関数の**大域的な下からの評価式**（global under-estimator）となる。

$$f(\mathbf{y}) \;\geq\; f(\mathbf{x}) + \nabla f(\mathbf{x})^{\top}(\mathbf{y}-\mathbf{x}) \qquad \text{for all } \mathbf{x},\mathbf{y} \in C. \tag{35.26}$$

これを $f(x) = x^2$ の $x = 1$ において確かめてみよう。接線は $1 + 2(y-1) = 2y - 1$ であり、グラフと接線の縦方向の差は $y^2 - (2y-1) = (y-1)^2 \geq 0$ となってすべての y に対して成り立つ。Equation 35.26 がどれほど強い主張であるか、立ち止まって味わってほしい。ただ 1 つの点 $\mathbf{x}$ で計算したただ 1 つの勾配が、定義域内の他のすべての点において f を下から抑えるのである。本節の残りのほぼすべての証明が、この 1 本の不等式の上に築かれる。Figure 35.11 は最初の 2 つの判定条件を並べて示している。弦はグラフの上側にあり、接線はグラフの下側にある。

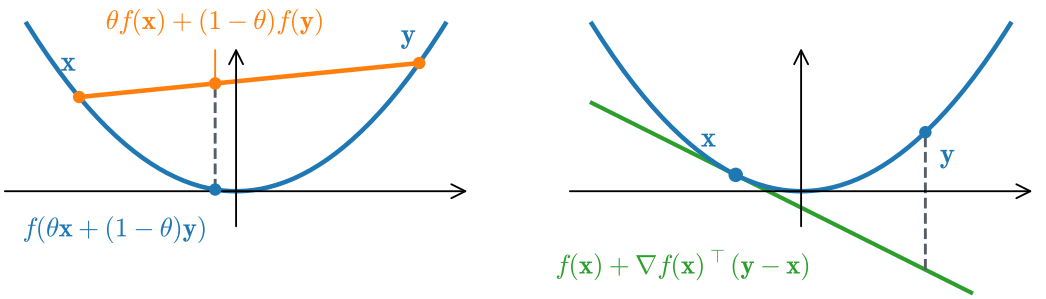


Figure 35.11: 2 つの同値な条件。(a) グラフの 2 点を結ぶ弦はグラフの上側にある：$f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y})$。(b) 任意の点における接線はグラフの下側にある：$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^{\top}(\mathbf{y}-\mathbf{x})$。

35.3.2.4 二階条件

滑らかな目的関数に対して凸性を確かめる最も手早い方法は、多くの場合、行列による判定である。f が**開**凸定義域 C 上で 2 回微分可能であるとき、f が凸であることと、そのヘッセ行列がいたるところで半正定値であることは同値である。

$$\nabla^2 f(\mathbf{x}) \succeq 0 \qquad \text{for all } \mathbf{x} \in C.$$

1 次元では、これはおなじみの微分積分の規則 $f''(x) \geq 0$、すなわち曲線が上向きに曲がるという条件に帰着する。d 次元では、$\mathbf{x}$ を通るすべての 1 次元スライスが上向きに曲がることを意味する。実際、$\mathbf{v}^{\top}\nabla^2 f(\mathbf{x})\,\mathbf{v}$ は f の方向 $\mathbf{v}$ の直線に沿った二階導関数だからである（34.2 節）。

この行列による判定から、定番の 3 つの例がただちに片づく。第 1 に、二次関数 $\frac{1}{2}\mathbf{x}^{\top}\mathbf{A}\mathbf{x} + \mathbf{b}^{\top}\mathbf{x}$（$\mathbf{A}$ は対称行列）は定数のヘッセ行列 $\mathbf{A}$ をもつため、凸であることと $\mathbf{A} \succeq 0$ であることは同値で

ある。第 2 に、最小二乗損失 $\frac{1}{2}\|\mathbf{X}\mathbf{w}-\mathbf{y}\|^2$（計画行列 $\mathbf{X}\in\mathbb{R}^{n\times d}$、$n$ 個の例、d 個の重み）のヘッセ行列は $\mathbf{X}^\top\mathbf{X}$ であり、$\mathbf{v}^\top\mathbf{X}^\top\mathbf{X}\mathbf{v}=\|\mathbf{X}\mathbf{v}\|^2\geq 0$ がすべての $\mathbf{v}$ について成り立つため、最小二乗法は常に凸である（33.3.2.2 節）。第 3 に、スカラー関数 e^x、$-\log x$、$x\log x$ は、それぞれの二階導関数 e^x、$1/x^2$、$1/x$ がいたるところで正であるため、いずれもその定義域上で凸である。

なぜ定義域 C が開集合であることにこだわったのだろうか。低次元の定義域上では、C の内部にとどまる方向だけが問題になるからである。たとえば x^2-y^2 は、2 次元全体のヘッセ行列が不定値であるにもかかわらず、直線 $y=0$ に制限すれば凸関数になる。

ここで、3 つの判定条件がまったく同じ関数クラスを特徴づけていることを確かめよう。証明は 4 つの短いパートからなるが、記憶に留める価値があるのは 2 番目のパートである。弦の**中間**点において接線による下界を適用すると勾配の項が相殺されるという手法であり、これとまったく同じ技巧をイェンセンの不等式の証明でも用いる。

命題（3 つの特徴づけの一致）。 **f を開凸定義域 C 上で微分可能な関数とする。このとき、弦条件 Equation 35.25 と一階条件 Equation 35.26 は同値である。さらに f が 2 回微分可能であれば、両者は $\nabla^2 f(\mathbf{x})\succeq 0$ が C 上で成り立つこととも同値である。**

証明。（弦条件 $\Rightarrow$ 一階条件。） $\mathbf{x},\mathbf{y}\in C$ と $\theta\in(0,1]$ を固定する。中間点 $\mathbf{x}+\theta(\mathbf{y}-\mathbf{x})$ において弦不等式を適用し、両辺から $f(\mathbf{x})$ を引いて θ で割ると次を得る。

$$\frac{f(\mathbf{x}+\theta(\mathbf{y}-\mathbf{x}))-f(\mathbf{x})}{\theta}\;\leq\; f(\mathbf{y})-f(\mathbf{x}).$$

$\theta\downarrow 0$ とすると、左辺は方向微分 $\nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x})$ に収束し、Equation 35.26 が得られる。

（一階条件 $\Rightarrow$ 弦条件。） $\mathbf{z}=\theta\mathbf{x}+(1-\theta)\mathbf{y}$ とおき、点 $\mathbf{z}$ における接線の下界を各端点に向かって 1 回ずつ、計 2 回適用する。

$$f(\mathbf{x})\geq f(\mathbf{z})+\nabla f(\mathbf{z})^\top(\mathbf{x}-\mathbf{z}),\qquad f(\mathbf{y})\geq f(\mathbf{z})+\nabla f(\mathbf{z})^\top(\mathbf{y}-\mathbf{z}).$$

第 1 の不等式に θ を、第 2 の不等式に $1-\theta$ を掛けて足し合わせる。1 次の項はまとまって $\nabla f(\mathbf{z})^\top(\theta\mathbf{x}+(1-\theta)\mathbf{y}-\mathbf{z})=0$ となり、Equation 35.25 が残る。

（二階条件 $\Rightarrow$ 一階条件。） f を $\mathbf{x}$ と $\mathbf{y}$ を結ぶ線分に制限し、$g(t)=f(\mathbf{x}+t(\mathbf{y}-\mathbf{x}))$ と定義する。その二階導関数は $g''(t)=(\mathbf{y}-\mathbf{x})^\top\nabla^2 f(\mathbf{x}+t(\mathbf{y}-\mathbf{x}))\,(\mathbf{y}-\mathbf{x})\geq 0$ である。ラグランジュの剰余項をもつテイラーの定理（34.1 節）により、$g(1)=g(0)+g'(0)+\frac{1}{2}g''(\tau)$ が、ある $\tau\in(0,1)$ に対して成り立つ。非負の剰余項を落とせば Equation 35.26 が得られる。

（一階条件 $\Rightarrow$ 二階条件。） 背理法のため、$\mathbf{v}^\top\nabla^2 f(\mathbf{x})\,\mathbf{v}<0$ がある点 $\mathbf{x}\in C$ と方向 $\mathbf{v}$ に対して成り立つと仮定する。f を $\mathbf{v}$ に沿って展開すると次を得る。

$$f(\mathbf{x}+t\mathbf{v})=f(\mathbf{x})+t\,\nabla f(\mathbf{x})^\top\mathbf{v}+\frac{t^2}{2}\,\mathbf{v}^\top\nabla^2 f(\mathbf{x})\,\mathbf{v}+o(t^2).$$

十分小さい $t>0$ に対しては、負の 2 次の項が剰余項 $o(t^2)$ を支配するため、$f(\mathbf{x}+t\mathbf{v})$ は接線の値 $f(\mathbf{x})+t\nabla f(\mathbf{x})^\top\mathbf{v}$ を真に下回り、Equation 35.26 に矛盾する。■

35.3.2.5 3 つの判定条件の数値的検証

2 つの損失に対して、3 つの判定条件をすべて数値的に試してみよう。用いるのは、最小二乗目的関数 $f(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$（ランダムな例 $n = 20$ 個、重み $d = 2$ 個）と、冒頭の実験で用いた二重井戸関数 ϕ である。最小二乗法については、条件からの逸脱量の厳密な式がすでにわかっている。x^2 のときと同様に二乗を展開すれば、弦ギャップ（グラフ引く弦）は $-\frac{1}{2}\theta(1-\theta)\|\mathbf{X}(\mathbf{u}-\mathbf{v})\|^2$、接線ギャップ（接線引くグラフ）は $-\frac{1}{2}\|\mathbf{X}(\mathbf{v}-\mathbf{u})\|^2$ となり、いずれも非正である。以下のコードは、各損失について 1,000 組のランダムな弦と接線を $[-1.5, 1.5]^d$ 内でサンプリングし、それぞれの最大の違反量を報告するとともに、サンプル点全体にわたるヘッセ行列の最小固有値を調べる。

```
X, y = rng.normal(size=(20, 2)), rng.normal(size=20)  # least squares, d = 2
losses = {'least squares': (2, lambda w: 0.5 * ((X @ w - y) ** 2).sum(),
                            lambda w: X.T @ (X @ w - y), lambda w: X.T @ X),
          'double well': (1, lambda w: phi(w[0]), dphi,  # phi'' = 12 x^2 - 4
                          lambda w: np.diag(12 * w**2 - 4))}
print('               worst chord gap  worst tangent gap  smallest Hessian eig')
for name, (d, f, grad, hess) in losses.items():
    U, V = rng.uniform(-1.5, 1.5, size=(2, 1000, d))  # 1000 random pairs
    T = rng.uniform(size=1000)
    chord = [f(t * u + (1 - t) * v) - (t * f(u) + (1 - t) * f(v))
             for u, v, t in zip(U, V, T)]             # graph minus chord
    tangent = [f(u) + grad(u) @ (v - u) - f(v)        # tangent minus graph
               for u, v in zip(U, V)]
    eig = min(np.linalg.eigvalsh(hess(u))[0] for u in U)
    print(f'{name:15s}{max(chord):15.3f}{max(tangent):19.3f}{eig:22.3f}')
i = int(np.argmax(chord))                             # the double well's worst
print(f'worst chord of the double well: from {U[i, 0]:.2f} to {V[i, 0]:.2f}')
```

```
               worst chord gap  worst tangent gap  smallest Hessian eig
least squares           -0.003             -0.020                11.949
double well              0.997              2.992                -4.000
worst chord of the double well: from -1.03 to 0.99
```

最小二乗法では、式の上での予測どおり、サンプリングされた弦がグラフの下に沈むことは 1 度もなく、接線がグラフの上に出ることもなく、ヘッセ行列の固有値は正にとどまる。一方、二重井戸関数は 3 つの検査すべてに不合格となる。最悪の弦は 2 つの谷の点を結んでグラフよりほぼ 1 だけ下を通り、接線は曲線を貫き、二階導関数は中央の山の近くで -4 付近まで落ち込む（そこでは $\phi''(0) = -4$ である）。ただし、これらの検査の間にある非対称性に注意しよう。ランダムサンプリングは凸性を**反証する**ことはできるが（ϕ にとっては不適切な弦が 1 本あれば完全な反例となる）、凸性を**証明する**ことは決してできない。最小二乗法に対しては、違反を見つけられなかったというだけである。最小二乗法の凸性を証明しているのはヘッセ行列による判定であり、それは $\mathbf{X}^\top\mathbf{X}$ がすべての $\mathbf{w}$ において同一の PSD 行列だからである。

35.3.2.6 狭義凸性と強凸性

単なる凸性はいくらでも平坦な区間を許容するため、そのような平坦な場所では勾配降下法がいくらでも遅く這うように進むことがあり得る。速い収束を保証するには、いたるところでゼロから離れて下から抑えられた曲率が必要になる。Figure 35.12(a) に示す 3 つの単純な関数がこの階層構造をよく表している。直線 $f(x) = x$ は凸であるが、弦がグラフ上にぴったり重なるため狭義凸ではない。4 次関数 x^4 は狭義凸である（二階導関数 $12x^2$ がゼロになるのは 0 のみであり、傾きはいたるところで狭義単調に増加する）が、$f''(0) = 0$ のため最小点において完全に平坦に

なっている。これに対し、二次関数 x^2 はすべての点で一定の曲率 $f''(x) = 2$ をもつ。

強凸性は、この観察を定量的な下界へと定式化する。$\mu > 0$ に対し、関数 f が μ **強凸**であるとは、曲率 μ の二次関数を引いてもなお凸関数として残ること、すなわち $h(\mathbf{x}) = f(\mathbf{x}) - \frac{\mu}{2}\|\mathbf{x}\|^2$ が凸であることをいう。f が微分可能であるとき、h（その勾配は $\nabla f(\mathbf{x}) - \mu\mathbf{x}$）に一階条件を適用し、恒等式 $\|\mathbf{y}\|^2 - \|\mathbf{x}\|^2 - 2\mathbf{x}^\top(\mathbf{y} - \mathbf{x}) = \|\mathbf{y} - \mathbf{x}\|^2$ を用いると、同値な不等式が得られる。

$$f(\mathbf{y}) \;\geq\; f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\,\|\mathbf{y} - \mathbf{x}\|^2 \qquad \text{for all } \mathbf{x}, \mathbf{y}. \tag{35.27}$$

幾何学的には、f のグラフは点 $\mathbf{x}$ における接線の上側にあるだけでなく、そこで接する曲率 μ の放物線よりも上側になければならない。h に二階条件を適用すれば行列による判定条件が得られる。すなわち、2 回微分可能な f が μ 強凸であることと、いたるところで $\nabla^2 f(\mathbf{x}) \succeq \mu\mathbf{I}$（ただし $\mathbf{I}$ は単位行列）が成り立つことは同値である。したがって x^2 は 2 強凸である一方、x^4 は 0 のいかなる近傍においても強凸にならない。$f''(0) = 0 < \mu$ がすべての $\mu > 0$ に対して成り立ってしまうからである。実用上のうれしい副産物にも注目しよう。任意の凸損失にリッジペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$ を加えると、その和はただちに λ 強凸になる。$\frac{\lambda}{2}\|\mathbf{w}\|^2$ を引けば元の凸損失に戻るからである。重み減衰は単にパラメータを縮小するだけでなく、強凸性を保証するのである。

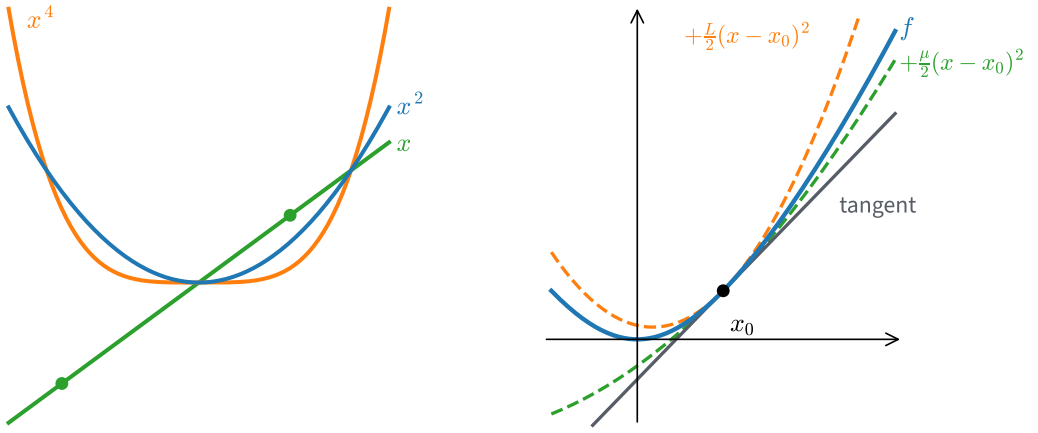


Figure 35.12: (a) 凸・狭義凸・強凸：x は弦がグラフ上に乗り、x^4 は原点で平坦になり、x^2 はいたるところで湾曲する。(b) μ 強凸かつ L 滑らかな関数（ここでは $f(x) = \frac{1}{2}x^2 + 3\log\cosh x$）は、$x_0 = 1$ で接する曲率 $\mu = 1$ と $L = 4$ の 2 つの放物線の間に挟まれる。両者の比が κ である。

強凸性は、滑らかさと自然な対をなしている。35.1.1 節で見たように、f が L **滑らか**であるとは、その勾配が L リプシッツ連続であること、すなわち $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$ を満たすことであった。2 回微分可能な凸関数においては、これは $\nabla^2 f \preceq L\mathbf{I}$ を意味する。滑らかさはグラフの上側に二次上界 $f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$ を置き、強凸性はグラフの下側に二次下界 Equation 35.27 を置く。両者を合わせると、f は曲率 μ と L をもつ 2 つの接放物線の間に挟まれる。Figure 35.12(b) はこれを $f(x) = \frac{1}{2}x^2 + 3\log\cosh x$ について描いたものであり、その二階導関数 $f''(x) = 1 + 3/\cosh^2 x$ は $\mu = 1$ から $L = 4$ の範囲を動く。

比 $\kappa = L/\mu$ は**条件数**（condition number）であり、35.1.2 節で二次関数に対して導入した固有値比を一般化したものである。通し例の二次関数 $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ では、極端な曲率は $\mu = 1$ と $L = 1000$ であり、$\kappa = 10^3$ となる。

35.3.2.7 劣勾配

ℓ_1 ノルム $\|\mathbf{x}\|_1$ やヒンジ損失 $\max(0, 1 - z)$ といった重要な損失には鋭い角があり、そこでは勾配が存在しないため Equation 35.26 をそのまま書くことはできない。しかし接線による下界をあきらめる代わりに、勾配が果たしていた**役割**（大域的な線形の下からの評価式を与えること）を残し、その役割自体を定義へと転じる。ベクトル $\mathbf{g}$ が f の点 $\mathbf{x}$ における**劣勾配**であるとは、次の条件を満たすことをいう。

$$f(\mathbf{y}) \;\geq\; f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{y} - \mathbf{x}) \qquad \text{for all } \mathbf{y}. \tag{35.28}$$

点 $\mathbf{x}$ におけるすべての劣勾配からなる集合を**劣微分**（subdifferential）と呼び、$\partial f(\mathbf{x})$ と書く。凸関数 f が微分可能な点では、劣微分はただ 1 つのベクトルからなる集合 $\{\nabla f(\mathbf{x})\}$ に潰れる。しかし角においては、グラフを下から支える傾き全体の集合へと扇状に広がる（Figure 35.13）。$f(x) = |x|$ の $x = 0$ においては、すべての傾き $g \in [-1, 1]$ が $|y| \geq gy$ を任意の y に対して満たすため $\partial f(0) = [-1, 1]$ となる。同様に、ヒンジ損失 $\max(0, 1 - z)$ の折れ点 $z = 1$ では $\partial f(1) = [-1, 0]$ が得られる。

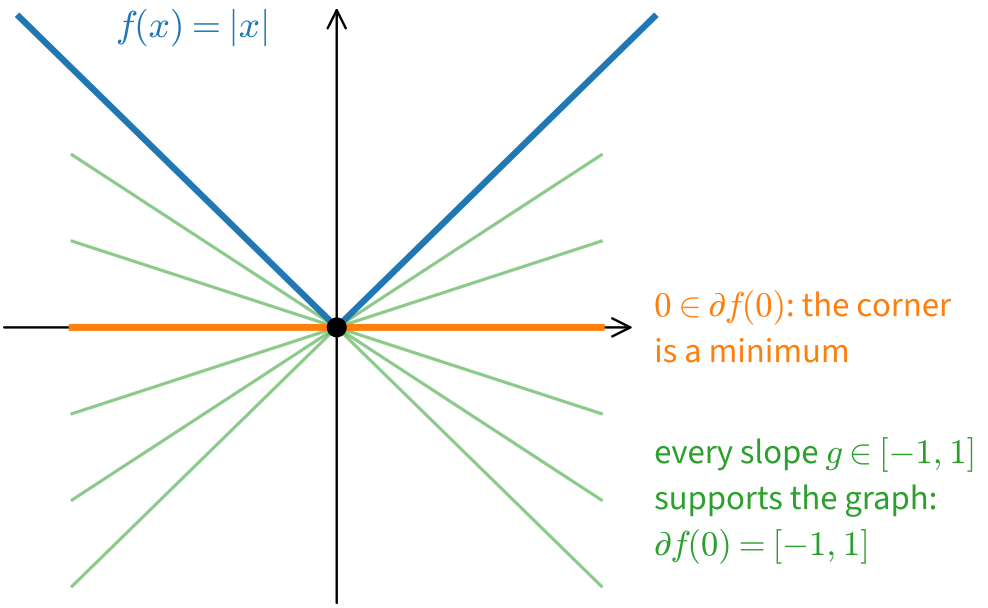


Figure 35.13: $f(x) = |x|$ の折れ点では勾配は存在しないが、劣微分は集合として定まる。すべての傾き $g \in (-1, 1)$ が V 字の下側で支持直線を定め、両端の傾き ± 1 は枝そのものに重なるため、$\partial f(0)$ は -1 から 1 までの閉区間全体となる。傾きゼロの要素（オレンジ）は最適性の証明 $0 \in \partial f(0)$ を与え、微分可能性なしにこの角が最小点であることが保証される。

ここからただちに、微分を用いない最適性の判定条件が得られる。すなわち、$\mathbf{x}^\star$ が f を最小化することと $\mathbf{0} \in \partial f(\mathbf{x}^\star)$ は同値である。証明は 1 行の同語反復にすぎない。$\mathbf{g} = \mathbf{0}$ を Equation 35.28 に代入すれば、$f(\mathbf{y}) \geq f(\mathbf{x}^\star)$ がすべての $\mathbf{y}$ について成り立つという式になり、これは文字どおり大域的最小点の定義そのものである。

凸関数は常に劣勾配をもつのだろうか。定義域のすべての内点において、答えはイエスである。この事実はイェンセンの不等式の証明で用いることになる。1 次元ではその理由は初等的である。

任意の 3 点 $a < b < c$（f が凸である開区間内の点）に対し、点 b における弦不等式を整理すると次を得る。

$$\frac{f(b)-f(a)}{b-a} \;\le\; \frac{f(c)-f(a)}{c-a} \;\le\; \frac{f(c)-f(b)}{c-b},$$

つまり、弦の傾きは左から右へ向かって非減少である。その結果、差分商 $q(x) = (f(x) - f(x_0))/(x - x_0)$ は任意の内点 x_0 において x に関して非減少となるため、その片側極限である左微分と右微分 $f'_-(x_0) \le f'_+(x_0)$ がともに有限な値として存在する。任意の傾き $g \in [f'_-(x_0), f'_+(x_0)]$ は劣勾配となる。実際、$y > x_0$ に対しては $q(y) \ge f'_+(x_0) \ge g$ であり、これを変形すれば $f(y) \ge f(x_0) + g(y - x_0)$ となる。$y < x_0$ の場合も対称である。

同じ保証は d 次元にも拡張される。凸関数はその定義域のすべての内点において少なくとも 1 つの劣勾配をもつため、常に支持超平面がグラフの下側に収まる（Rockafellar (1970)、Boyd and Vandenberghe (2004) 2.5 節）。本書ではこの結果を証明なしに用いる。ただし、探索方向としての劣勾配は真の勾配よりはるかに弱いことに注意しておこう。$-\mathbf{g}$ の方向にステップを進めても、f が減少するとは限らないのである（最小点 $x = 0$ において $|x|$ を考えると、正当な劣勾配 $g = \frac{1}{2}$ によるステップは $-\eta/2$ へ移動し、損失を増大させる）。そのため、劣勾配法には減衰するステップサイズが必要となり、収束も遅く、誤差は $1/\sqrt{k}$ のオーダーでしか減少しない。

35.3.3 凸性の帰結

本書の以降の章では凸性の 3 つの主要な帰結を活用するが、それら 3 つはすべて同じ幾何学的源泉から生まれる。すなわち、いたるところで関数を下から抑える支持直線または支持超平面である。この下界を確率変数について平均すれば、イェンセンの不等式が得られる。停留点において評価すれば、大域的最適性が証明され、35.3.2.2 節で弦条件のみから導いた結果がさらに研ぎ澄まされる。勾配降下法のステップに沿って足し合わせれば、局所的な前進が大域的な収束率へと変わる。イェンセンの不等式はそれ単独で完結しているため、最適化のみを目的として読んでいる場合はそのまま次の項へ進んでよい。

35.3.3.1 イェンセンの不等式

期待損失、平均対数尤度、KL ダイバージェンスはいずれも同じ基本的な問いに直面する。平均の関数値と、関数値の平均とはどう比較されるかという問いである。最も単純な 2 点の例を見れば、不等号がどちらを向くかがわかる。確率変数 Z が値 0 と 2 をそれぞれ確率 $\frac{1}{2}$ でとるとし、$f(x) = x^2$ とする。f を平均において評価すると $f(\mathbb{E}[Z]) = f(1) = 1$ となるのに対し、実現値にわたって f の平均をとると $\mathbb{E}[f(Z)] = \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 4 = 2$ となる。この差 1 はまさに Z の分散にほかならない。実際、$f(x) = x^2$ に対しては差 $\mathbb{E}[Z^2] - (\mathbb{E}[Z])^2$ は常に分散となる。Figure 35.11(a) を振り返ってほしい。弦不等式は重み θ と $1-\theta$ をもつ 2 点分布に対してまさにこの比較をしており、先ほど計算した差 $\theta(1-\theta)(x-y)^2$（x^2 に対する弦とグラフの差）はその分布の分散そのものである。

この議論には 2 点に限られる理由はまったくない。Equation 35.25 を帰納的に適用すれば（演習問題 2）、次の有限形式が得られる。

$$f\left(\sum_i \theta_i \mathbf{x}_i\right) \;\le\; \sum_i \theta_i\, f(\mathbf{x}_i), \qquad \theta_i \ge 0,\; \sum_i \theta_i = 1. \tag{35.29}$$

和が 1 になる非負の重みは点 $\mathbf{x}_i$ 上の離散確率分布を定めるため、有限個の平均を任意の期待値に置き換えるのが自然な一般化となる。

命題（イェンセンの不等式）。 **f を凸集合 $C \subseteq \mathbb{R}^d$ 上の凸関数とし、$\mathbf{Z}$ を C に値をとる確率ベクトルで、$\mathbb{E}\|\mathbf{Z}\| < \infty$ かつ平均 $\mathbf{m} = \mathbb{E}[\mathbf{Z}]$ が C の内部に属するものとする。このとき、次が成り立つ。**

$$f(\mathbf{m}) \;\le\; \mathbb{E}\left[f(\mathbf{Z})\right]. \tag{35.30}$$

さらに f が狭義凸であるならば、等号が成り立つのは $\mathbf{Z} = \mathbf{m}$ が確率 1 で成り立つとき、かつそのときに限る。

証明。 $\mathbf{m}$ は C の内部にあるため、f は少なくとも 1 つの劣勾配 $\mathbf{g} \in \partial f(\mathbf{m})$ をもつ。点 $\mathbf{m}$ における支持超平面は、あらゆる実現値に対して一斉に f を下から抑える。すなわち Equation 35.28 により、次が成り立つ。

$$f(\mathbf{Z}) \;\ge\; f(\mathbf{m}) + \mathbf{g}^{\top}(\mathbf{Z} - \mathbf{m}).$$

ここで両辺の期待値をとる。$\mathbb{E}[\mathbf{Z} - \mathbf{m}] = \mathbf{0}$ であるため 1 次の項は消え、$\mathbb{E}[f(\mathbf{Z})] \ge f(\mathbf{m})$ が残る。f が狭義凸であるとき、この支持超平面はグラフと点 $\mathbf{m}$ **のみ**で接する。実際、もしある $\mathbf{x} \neq \mathbf{m}$ でも接したとすれば、狭義凸性により $f\big(\frac{\mathbf{x}+\mathbf{m}}{2}\big) < \frac{1}{2}f(\mathbf{x}) + \frac{1}{2}f(\mathbf{m})$ が強いられ、中点においてグラフが支持超平面より真に下側に来てしまい矛盾する。したがって、各点ごとの不等式は $\mathbf{Z} \neq \mathbf{m}$ である限り狭義で成り立つため、期待値で等号が成り立つには確率 1 で $\mathbf{Z} = \mathbf{m}$ でなければならない。$\blacksquare$

副産物として、同じ各点ごとの下界から $f(\mathbf{Z})$ が可積分な確率変数によって下から抑えられることがわかるため、$\mathbb{E}[f(\mathbf{Z})]$ は常に確定した値をもつ（$+\infty$ になり得るが、その場合も不等式は自明に成り立つ）。平たく言えば、平均を固定したまま確率変数のばらつきを広げると、凸関数の期待値は増加しこそすれ減少することはない。凹関数 f に対しては不等号の向きが逆転し、$\mathbb{E}[f(\mathbf{Z})] \le f(\mathbb{E}[\mathbf{Z}])$ となる。

機械学習においてイェンセンの不等式が最も頻繁に使われる場面は、KL ダイバージェンスが決して負にならないことの証明である。37.1 節では、この同じ結果にギブスの不等式（Gibbs' inequality）として出会う。

系（KL ダイバージェンスの非負性）。 **共通の有限アルファベット上の確率分布 p と q について、$q(x) > 0$ が $p(x) > 0$ となるすべての点で成り立つとする。このとき、**

$$D_{\mathrm{KL}}(p \,\|\, q) \;=\; \sum_{x} p(x) \log \frac{p(x)}{q(x)} \;\ge\; 0,$$

が成り立ち、等号成立は $p = q$ のとき、かつそのときに限る。

証明。 $Z \sim p$ を抽出し、狭義凸関数 $-\log$ と正の確率変数 $R = q(Z)/p(Z)$ に対してイェンセンの不等式を適用する。

$$D_{\mathrm{KL}}(p \,\|\, q) = \mathbb{E}_p\left[-\log R\right] \;\ge\; -\log \mathbb{E}_p[R] = -\log \sum_{x : p(x) > 0} p(x)\, \frac{q(x)}{p(x)} \;\ge\; -\log 1 = 0,$$

ここで最後の不等号には $\sum_{x:p(x)>0} q(x) \le 1$ を用いた。イェンセンの不等式の等号成立条件により、等号が成り立つのは、R が確率 1 で定数であり、かつ q が p の台の外に確率質量をもたないとき、すなわち $q = c\,p$（ただし $c = 1$）のとき、かつそのときに限る。$\blacksquare$

同じ発想を凹関数 log に適用すれば、古典的な**相加・相乗平均の不等式**（arithmetic–geometric mean inequality）が得られる。任意の正の数 $x_1, \dots, x_N$ に対し、凹性から $\log\big(\frac{1}{N}\sum_i x_i\big) \geq \frac{1}{N}\sum_i \log x_i$ が成り立ち、両辺の指数をとれば次式が得られる。

$$\frac{1}{N}\sum_{i=1}^{N} x_i \;\geq\; \left(\prod_{i=1}^{N} x_i\right)^{1/N},$$

ここで等号が成り立つのは、すべての x_i が等しいとき、かつそのときに限る（log が狭義凹関数だからである）。同様に、期待値の対数を下から評価するエビデンス下界（evidence lower bound、ELBO）（36.7 節）も、log に対するイェンセンの不等式の直接的な応用である。

イェンセンギャップと KL ダイバージェンスの等号成立付近の挙動を、ともに数値的に検証してみよう。第 1 に、$Z \sim \mathcal{N}(0,1)$ と $f(x) = e^x$ に対しては両辺を閉じた式で計算でき（$\mathbb{E}[e^Z] = \sqrt{e} \approx 1.649$ に対して $e^{\mathbb{E}[Z]} = 1$）、モンテカルロ推定がどれほど正確にこれに一致するかを確認できる。第 2 に、q を p へ $(1-\varepsilon)p + \varepsilon q$ によって混ぜ合わせていくとき、KL ダイバージェンスはどれほど速くゼロへ近づくだろうか。ε に関して線形だろうか、それともさらに速いだろうか。

```
rng = np.random.default_rng()
z = rng.normal(size=1_000_000)                # Z ~ N(0, 1)
print(f'E[exp(Z)] = {np.exp(z).mean():.2f} vs exp(E[Z]) = {np.exp(z.mean()):.2f}'
      f'  (population: {np.exp(0.5):.4f} vs 1)')
kl = lambda p, q: (p * np.log(p / q)).sum(axis=1)
p = rng.dirichlet(np.ones(10), size=1000)     # 1000 random pairs, 10 symbols
q = rng.dirichlet(np.ones(10), size=1000)
print('median KL(p || (1-eps) p + eps q), eps = 1e-2, 1e-3, 1e-4:',
      *[f'{np.median(kl(p, (1 - e) * p + e * q)):.1e}' for e in (1e-2, 1e-3, 1e-4)])
```

```
E[exp(Z)] = 1.65 vs exp(E[Z]) = 1.00  (population: 1.6487 vs 1)
median KL(p || (1-eps) p + eps q), eps = 1e-2, 1e-3, 1e-4: 1.6e-04 1.6e-06 1.6e-08
```

モンテカルロ平均はちょうど 1.65 対 1.00 の付近に落ち着く。サンプル数を増やせば推定ノイズは縮まるが、構造的なイェンセンギャップは一定のままである。KL ダイバージェンスの実験では、ダイバージェンスは ε よりはるかに速く消失し、ε を 10 分の 1 にするごとに中央値はおよそ 100 分の 1 に減少する。対数を展開すればその理由がわかる。すなわち、$D_{\mathrm{KL}}(p \,\|\, (1-\varepsilon)p + \varepsilon q) \approx \frac{\varepsilon^2}{2}\sum_x (q(x)-p(x))^2/p(x)$ が、$\varepsilon\, q(x)/p(x)$ がすべての x について十分小さくなった段階で成り立つからである。（p がいずれかの記号に極小の確率を割り当てるペアはこの 2 次の領域に入るのが最も遅れるため、ここでは最大値ではなく中央値を報告している。）等号には 2 次のオーダーで近づき、極限 $q \to p$ においてのみ達成される。

35.3.3.2 極小点と大域的最小点

凸関数のすべての極小点が大域的最小点であることはすでに証明した（35.3.2.2 節）。これと対になる 3 つの事実も、同じくらい手軽にそれぞれ 1 行で導かれる。第 1 に、すべての大域的最小点からなる集合は凸である。$\mathbf{x}$ と $\mathbf{y}$ がともに最小値 $f^\star$ を達成するならば、弦不等式によって両者の間のすべての点が $f^\star$ 以下に抑えられるからである。第 2 に、開集合上の微分可能な凸関数 f に対しては、$\nabla f(\mathbf{x}) = \mathbf{0}$ からただちに $f(\mathbf{x}) = f^\star$ が従う。勾配がゼロのとき、一階条件は $f(\mathbf{y}) \geq f(\mathbf{x})$ がすべての $\mathbf{y}$ に対して成り立つと読めるからである。第 3 に、勾配が小さければ準最適性ギャップ（suboptimality gap）も小さいことが保証される。一階条件を最小点 $\mathbf{x}^\star$ に向けて適用すれば、$f(\mathbf{x}) - f^\star \leq \nabla f(\mathbf{x})^\top(\mathbf{x} - \mathbf{x}^\star) \leq \|\nabla f(\mathbf{x})\|\,\|\mathbf{x} - \mathbf{x}^\star\|$ が得られる。したがって、

35.1.1 節で示した停留性の結果（$\eta = 1/L$ の勾配降下法が最小の勾配ノルムをゼロへ収束させること）は、凸目的関数に対しては、反復点が $\mathbf{x}^\star$ から有界な距離内にとどまる限り、**損失値**そのものに対する保証へと格上げされる。35.3.3.3 節では、反復点が実際に有界な距離にとどまることを証明し、正確な収束率を突き止める。

単なる凸性が保証**しない** 2 つの点には注意しておこう。最小点は一意であるとは限らず（凸関数は線分全体に沿って平坦になり得る）、そもそも存在するとも限らない（e^x は 0 に $x \to -\infty$ で限りなく近づくが、決してその値をとらない）。狭義凸性は複数の最小点を排除し、強凸性は存在と一意性の両方を保証する。

命題（一意性）。 狭義凸な f の最小点は高々 1 つである。連続な μ 強凸関数 f は、空でない内部をもつ空でない閉凸集合 C 上でただ 1 つの最小点をもつ。

証明。 $\mathbf{x} \neq \mathbf{y}$ がともに値 $f^\star$ を達成する大域的最小点であったと仮定する。$\theta = \frac{1}{2}$ における狭義の弦不等式により、両者の中点は $f(\frac{\mathbf{x}+\mathbf{y}}{2}) < f^\star$ を満たすことになり矛盾する。強凸性は狭義凸性を導くため、残るは存在の証明のみである。ここでは本書の制約なし損失をカバーする、微分可能な f と $C = \mathbb{R}^d$ の場合について証明する（一般の場合も、$f - \frac{\mu}{2}\|\cdot\|^2$ の C の内点における劣勾配を用いれば同じ手順で従う）。任意の初期点 $\mathbf{x}_0$ を固定する。Equation 35.27 において 1 次の項は少なくとも $-\|\nabla f(\mathbf{x}_0)\|\,\|\mathbf{y} - \mathbf{x}_0\|$ であるため、$f(\mathbf{y}) > f(\mathbf{x}_0)$ が $\|\mathbf{y} - \mathbf{x}_0\| > 2\|\nabla f(\mathbf{x}_0)\|/\mu$ である限り成り立つ。よって劣レベル集合 $\{\mathbf{y} : f(\mathbf{y}) \le f(\mathbf{x}_0)\}$ は有界であり、f の連続性から閉集合でもある。空でない有界閉集合上のすべての連続関数は最小値を達成し、この劣レベル集合上の最小値は自動的に $\mathbb{R}^d$ 全体での大域的最小値となる。■

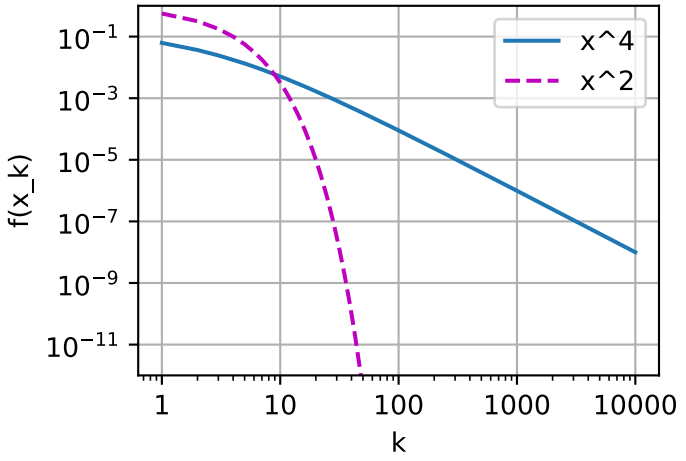
C と連続性に関する技術的な仮定は、定義域の境界で上向きに不連続に跳び上がる病的な関数を除外しているにすぎず、本書で扱う損失ではそのような事態は決して起こらない。対照的に、e^x に最小点が存在しない現象は実際の機械学習にも現れる。演習問題 3 では、これが線形分離可能なデータに対するロジスティック回帰へ直接つながることを確かめる。

35.3.3.3 収束率

凸性から勾配降下法が**どこで**止まるかはわかるが、そこへ**どれほど速く**到達するかはわからない。しかも速度に関しては、単なる凸性が約束してくれることは驚くほど少ない。$f(x) = x^2$ と $f(x) = x^4$ を比べてみよう。どちらも一意な最小点 0 をもつ狭義凸関数であるが、強凸なのは x^2 だけである。まったく同じ勾配降下法のループ（$\eta = \frac{1}{8}$、初期点 $x_0 = 1$）を両方の関数に対して 10^4 ステップ実行し、どれほど異なる振る舞いを見せるかを観察しよう。

```
eta, steps = 1 / 8, 10_000
x2, x4 = np.ones(steps + 1), np.ones(steps + 1)  # iterates from x_0 = 1
for j in range(steps):                           # same start, same step
    x2[j + 1] = x2[j] - eta * 2 * x2[j]          # f(x) = x^2
    x4[j + 1] = x4[j] - eta * 4 * x4[j]**3       # f(x) = x^4
k = np.arange(1, steps + 1)
print('x^2: first k with x_k < 1e-6:', k[x2[1:] < 1e-6][0])
print('x^4: x_k * sqrt(k) at k = 1e2, 1e3, 1e4:',
      (x4[1:] * np.sqrt(k))[[99, 999, 9999]].round(3))
d2l.plot(k, [x4[1:]**4, x2[1:]**2], 'k', 'f(x_k)', xscale='log',
         yscale='log', ylim=(1e-12, 1), legend=['x^4', 'x^2'])
```

```
x^2: first k with x_k < 1e-6: 49
x^4: x_k * sqrt(k) at k = 1e2, 1e3, 1e4: [0.973 0.996 1.   ]
```



対比は劇的である。x^2 では、各ステップが反復点に定数因子 $1-2\eta = 0.75$ を掛け、10^{-6} 未満へわずか 49 ステップで押し下げるため、損失はほぼ一瞬でプロットの下端を突き抜ける。対照的に x^4 では、各ステップが x_k に掛ける縮小率 $1-4\eta x_k^2$ は 1 へとじわじわ近づいていく（反復点が 0 の平坦な谷へ近づくにつれてそうなる）。出力された積 $x_k\sqrt{k}$ が 1 付近に落ち着くことから、x_k は $1/\sqrt{k}$ の速さでしか減衰せず、なお 0.01 の値に 10,000 ステップ後でもとどまっていることがわかる。そのため両対数プロット上では、損失ギャップ x_k^4 は傾き -2 の直線を描く。狭義凸性は複数の最小点を排除するが、速い収束は保証できない。幾何収束率を得るには $\mu > 0$ が必要である。

次の 2 つの命題は、この対比を完全に一般的な形で定式化する。すなわち、滑らかな強凸関数はすべて x^2 のように幾何収束するのに対し、$\mu > 0$ がなければ保証できるのは $O(1/k)$ の減衰にとどまる。どちらの証明も、35.1.1 節の**降下補題**（descent lemma）から出発する。L 滑らかな関数 f と勾配ステップ $\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{L}\nabla f(\mathbf{x}_k)$ に対し、次が成り立つ。

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \frac{1}{2L}\,\|\nabla f(\mathbf{x}_k)\|^2.$$

各ステップは目的関数を少なくとも $\|\nabla f(\mathbf{x}_k)\|^2/(2L)$ だけ減少させる。一般の非凸関数では、ここから言えるのは勾配がゼロへ縮むことだけである。凸性は、勾配と準最適性ギャップを結ぶ欠けた環を補う。$f(\mathbf{x}_k) - f^\star$ がまだ大きい限り、前進を続けるのに十分な大きさの勾配が必ず存在するため、反復点が最適点から遠く離れた場所で停滞することはあり得ない。

命題（滑らかな凸関数に対する勾配降下法）。 *f を凸かつ L 滑らかな関数とし、$\mathbf{x}^\star$ をその最小点とする。$\eta = 1/L$ を用いた勾配降下法は、すべての $k \ge 1$ に対して次を満たす。*

$$f(\mathbf{x}_k) - f(\mathbf{x}^\star) \;\le\; \frac{L\,\|\mathbf{x}_0 - \mathbf{x}^\star\|^2}{2k}. \tag{35.31}$$

したがって損失ギャップは、滑らかさの定数と最小点までの初期距離の二乗を係数として、少なくとも $1/k$ の速さで減衰する。具体的な数値を入れると、$L = 1$ かつ $\|\mathbf{x}_0 - \mathbf{x}^\star\| = 10$ であれば上界は $50/k$ となり、ギャップが 10^{-2} 未満になることが 5,000 ステップ以内に保証される。

証明。 降下補題と、一階条件 Equation 35.26 を点 $\mathbf{x}_k$ において最適点へ向けて評価した式 $f(\mathbf{x}_k) - f(\mathbf{x}^\star) \le \nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{x}^\star)$ を組み合わせる。

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^\star) \;\le\; \nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{x}^\star) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|^2.$$

ここで巧妙な変形を行う。平方完成によって、右辺を $\mathbf{x}^\star$ までの 2 つの二乗距離の差に書き換えるのである。

$$\begin{aligned}\nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{x}^\star) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|^2 \\ = \frac{L}{2}\left(\|\mathbf{x}_k - \mathbf{x}^\star\|^2 - \left\|\mathbf{x}_k - \mathbf{x}^\star - \tfrac{1}{L}\nabla f(\mathbf{x}_k)\right\|^2\right).\end{aligned}$$

第 2 のノルムはまさに $\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|^2$ であることに注目しよう。$j = 0, \dots, k-1$ にわたって和をとると、右辺は望遠鏡和としてきれいに相殺される。

$$\begin{aligned}\sum_{j=0}^{k-1}\left(f(\mathbf{x}_{j+1}) - f(\mathbf{x}^\star)\right) &\le \frac{L}{2}\left(\|\mathbf{x}_0 - \mathbf{x}^\star\|^2 - \|\mathbf{x}_k - \mathbf{x}^\star\|^2\right) \\ &\le \frac{L}{2}\|\mathbf{x}_0 - \mathbf{x}^\star\|^2.\end{aligned}$$

降下補題により $f(\mathbf{x}_j)$ は決して増加しないため、最後の項 $f(\mathbf{x}_k) - f(\mathbf{x}^\star)$ がこの和の最小の項となり、したがって $k\,(f(\mathbf{x}_k) - f(\mathbf{x}^\star))$ は和全体で上から抑えられる。k で割れば Equation 35.31 が得られる。■

この証明には注目すべき特徴が 2 つある。第 1 に、次元 d が一度も現れない。10 億個の重みをもつモデルであっても、L と初期距離が同じであればまったく同じ最悪ケースの保証を享受する。第 2 に、和をとる前の 1 ステップごとの不等式は非負の左辺 $f(\mathbf{x}_{k+1}) - f^\star \ge 0$ をもつため、$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \le \|\mathbf{x}_k - \mathbf{x}^\star\|$ が強制される。すなわち、1 ステップ進むごとにあらゆる最小点へ単調に近づいていくのである。

強凸性を加えると、この $1/k$ の減衰は幾何収束率へと一変する。

命題（強凸性のもとでの線形収束率）。 **f を μ 強凸かつ L 滑らかな関数とし、$\mathbf{x}^\star$ をその最小点とする。$\eta = 1/L$ を用いた勾配降下法は次を満たす。**

$$f(\mathbf{x}_k) - f(\mathbf{x}^\star) \;\le\; \left(1 - \frac{\mu}{L}\right)^k \left(f(\mathbf{x}_0) - f(\mathbf{x}^\star)\right). \tag{35.32}$$

言葉で言えば、反復を 1 回行うごとに、残っている損失ギャップの少なくとも割合 $\mu/L = 1/\kappa$ が削り取られる。

証明。 証明は 2 つの短いステップに分かれる。まず、強凸性により勾配ノルムの二乗が準最適性ギャップによって下から抑えられる。Equation 35.27 の両辺を $\mathbf{y}$ について最小化する。左辺はその最小値 $f(\mathbf{x}^\star)$ を達成し、一方右辺は $\mathbf{y}$ に関する単純な二次関数であり、$\mathbf{y} = \mathbf{x} - \frac{1}{\mu}\nabla f(\mathbf{x})$ において最小化され、そのときの値は $f(\mathbf{x}) - \frac{1}{2\mu}\|\nabla f(\mathbf{x})\|^2$ となる。これを整理すると次式を得る。

$$\frac{1}{2}\|\nabla f(\mathbf{x})\|^2 \;\ge\; \mu\,(f(\mathbf{x}) - f(\mathbf{x}^\star)) \qquad \text{for all } \mathbf{x}. \tag{35.33}$$

次に、Equation 35.33 を降下補題へ直接代入する。

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^\star) \le f(\mathbf{x}_k) - f(\mathbf{x}^\star) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|^2 \le \left(1 - \frac{\mu}{L}\right)\left(f(\mathbf{x}_k) - f(\mathbf{x}^\star)\right).$$

この縮小を k 回適用すれば証明が完了する。■

この証明の後半をよく見てほしい。そこでは凸性を一度も使っておらず、用いたのは不等式 Equation 35.33 だけである。この驚くべき事実には 35.3.5 節で立ち返る。また、強凸でない場合と比べたときの劇的な高速化にも注目しよう。相対精度 ϵ に到達するのに必要なステップ数は、もはや $O(1/\epsilon)$ ではなくおよそ $\kappa \log(1/\epsilon)$ で済む。$\kappa = 10^3$ の通し例の二次関数では、6 桁の精度改善 $(1 - 10^{-3})^k \le 10^{-6}$ に要するステップ数は、この上界によれば約 13,800 ステップである。

Equation 35.32 は実際にはどれほど精密な評価なのだろうか。ヘッセ行列 $\mathbf{X}^\top\mathbf{X}$ が構成上、固有値 1 と 10 をもつ最小二乗問題（$\mu = 1$、$L = 10$、$\kappa = 10$）で測ってみよう。$\eta = 1/L$ で勾配降下法を実行し、連続する損失ギャップの実際の 1 ステップあたりの比 $(f(\mathbf{w}_{k+1}) - f^\star)/(f(\mathbf{w}_k) - f^\star)$ を追跡する。

```
rng = np.random.default_rng()
Q, _ = np.linalg.qr(rng.normal(size=(20, 2)))  # orthonormal columns
R, _ = np.linalg.qr(rng.normal(size=(2, 2)))   # random rotation
X = Q @ np.diag([1.0, np.sqrt(10.0)]) @ R      # X^T X has eigenvalues 1, 10
y = rng.normal(size=20)
f = lambda w: 0.5 * ((X @ w - y) ** 2).sum()
mu, L = np.linalg.eigvalsh(X.T @ X)[[0, -1]]
w_star = np.linalg.lstsq(X, y, rcond=None)[0]
w, gaps = np.zeros(2), [f(np.zeros(2)) - f(w_star)]
for _ in range(60):                            # gradient descent, eta = 1/L
    w = w - X.T @ (X @ w - y) / L
    gaps.append(f(w) - f(w_star))
r = np.array(gaps[1:]) / np.array(gaps[:-1])   # per-step contraction of f - f*
print(f'mu = {mu:.4f}, L = {L:.4f}: bound 1 - mu/L = {1 - mu / L:.4f},'
      f' its square {(1 - mu / L)**2:.4f}')
print('first three ratios:', r[:3].round(4), ' largest ratio:', r.max().round(4))
```

```
mu = 1.0000, L = 10.0000: bound 1 - mu/L = 0.9000, its square 0.8100
first three ratios: [0.4788 0.81   0.81  ]  largest ratio: 0.81
```

先ほどの上界が約束する 1 ステップあたりの縮小率は 0.9 であるが、2 ステップ目以降に実測された比は 0.81、すなわち $(1-\mu/L)^2$ となっている。（最初の 1 回目の比だけはランダムな初期方向に依存し、実行ごとに変動する。）更新を $\mathbf{X}^\top\mathbf{X}$ の固有ベクトル座標で見れば、何が起きたのかがわかる。$\eta = 1/L$ とすると、最初の 1 ステップが $\mathbf{w} - \mathbf{w}^\star$ の硬い固有成分に $1 - L/L = 0$ を掛けて完全に消し去るのである。2 次元では遅い成分だけが残る。その $\mathbf{w}^\star$ までの距離は 1 ステップごとに $1 - \mu/L$ 倍に縮むため、距離の二乗に比例する損失ギャップは $(1-\mu/L)^2$ 倍に縮む。より高い次元では、速い固有成分が減衰するにつれて、この比は $(1-\mu/L)^2$ へと近づいていく。

先ほどの短い証明では、縮小率の二乗分を大まかに見積もって手放していたのである。より精密な解析によれば、$\eta = 1/L$ の勾配降下法は実際には、$f - f^\star$ を 1 ステップあたり $(1-\mu/L)^2$ の割合で、**すべての** L 滑らかかつ μ 強凸関数に対して縮小させ (Taylor et al. 2018)、二次関数において等号が達成される。演習問題 5 では、これと密接に関連する上界 $\kappa(1-\mu/L)^{2k}$ を証明する。$\kappa = 10^3$ の通し例の二次関数に戻ると、損失ギャップで 6 桁の精度（最小点までの距離で 3 桁の

精度）を得るのに要するステップ数は、単純な上界では約 $13{,}800$ ステップ、真の $(1-\mu/L)^2$ のレートでは約 $6{,}900$ ステップ、さらに最適な固定ステップサイズ $\eta = 2/(\mu+L)$ ではその半分の $3{,}454$ ステップとなる。これは、同じ最適ステップサイズで距離の精度 6 桁を得るために要した $6{,}908$ ステップ（35.1 節）のちょうど半分である。最初の 2 倍の差は我々の証明の緩さから生じ、もう 1 つの 2 倍の差はステップサイズを 2 倍にしたことから生じている。

35.3.4 凸性の判定

大域的最適性や明快な収束率は、与えられた損失関数がそもそも凸であるかどうかを見分けられて初めて役に立つ。ヘッセ行列を手計算するのはすぐに煩雑になるうえ、折れ点ではそもそも使えないため、実務においてゼロから凸性を確認することはほとんどない。その代わりに、単純な凸の構成要素を組み合わせることで凸目的関数を見分ける。

35.3.4.1 凸関数の演算規則

4 つの基本的な演算が凸性を保存し、機械学習に現れるほぼすべての凸損失は、基本的な凸関数からこれら 4 つの規則だけを使って構成されている。

1. **非負重み付き和**。$f_1, \dots, f_m$ が凸であり $w_i \ge 0$ ならば、$\sum_i w_i f_i$ も凸である。各 f_i の弦不等式に w_i を掛けて足し合わせるだけでよい。特に、訓練例ごとの凸な損失を訓練セット全体で平均すれば、常に凸目的関数が得られる。

2. **アフィン写像との前合成**。f が凸ならば、$\mathbf{x} \mapsto f(\mathbf{A}\mathbf{x}+\mathbf{b})$ も凸である。アフィン写像は $\mathbf{x}$ と $\mathbf{y}$ を結ぶ線分を、まったく同じ重み θ を保ったままそれらの像を結ぶ線分へと写すため、弦不等式がそのまま引き継がれる。その結果、モデルの**予測**に関して凸である任意の損失は、その予測を生成する**線形モデルの重み**に関しても自動的に凸となる。

3. **各点最大値と上限**。すべての f_i が凸ならば、$\max_i f_i$ も凸である。最も手早い証明はエピグラフを用いるものである。各点最大値のエピグラフは個々のエピグラフの共通部分に一致し、凸集合の共通部分は凸である。したがって、アフィン関数のいかなる族の上側包絡線（upper envelope）も、その族がどれほど大きくても凸となる。

4. **単調合成**。g が凸であり、h が凸**かつ非減少**であるならば、合成関数 $h \circ g$ は凸である。2 行で証明できる。

$$h\big(g(\theta\mathbf{x}+(1-\theta)\mathbf{y})\big) \;\le\; h\big(\theta g(\mathbf{x}) + (1-\theta) g(\mathbf{y})\big) \;\le\; \theta\, h(g(\mathbf{x})) + (1-\theta)\, h(g(\mathbf{y})),$$

   ここで第 1 の不等号は g の凸性と h の単調性を合わせて用い、第 2 の不等号は h の凸性を用いている。h に対する単調性の条件を忘れないようにしよう。$h(t) = t^2$ は凸であるが非減少ではなく、これを凸関数 $g(x) = x^2 - 1$ と合成すると非凸な二重井戸関数 $(x^2-1)^2$ が生じる（ϕ はこれに $x/2$ を加えたものである）。

これら 4 つの規則により、標準的な凸損失のほぼすべてが一挙に保証される。ヒンジ損失 $\max(0, 1 - y\,\mathbf{w}^\top\mathbf{x})$ は $\mathbf{w}$ の 2 つのアフィン関数の最大値である（規則 3 と 2）。ℓ_1 ノルム $\|\mathbf{w}\|_1 = \sum_i \max(w_i, -w_i)$ はアフィン関数の最大値の和である（規則 1、3、2）。ロジスティック損失 $\log(1 + e^{-y\,\mathbf{w}^\top\mathbf{x}})$ は、スカラー関数 $t \mapsto \log(1+e^{t})$（二階導関数 $e^t/(1+e^t)^2$ が真に正であるため凸）にアフィン写像を前合成し（規則 2）、訓練例にわたって和をとったものである（規則 1）。これらのいずれにリッジペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$ を加えても、強凸な目的関数が得られる。ただし、規則 3 のもとでは凸性が自動的に強凸性へ格上げされるわけではないことに注意しよう。$\max(x^2, 1-x)$ は凸であるが、平坦なアフィン直線 $1-x$ と区間 $(-1.618, 0.618)$（$x^2 = 1-x$ の 2 つの根の間）全体にわたって一致するため、強凸ではない。

この一覧から目立って欠けているものに注目してほしい。アフィンでも凸でもない内側の写像との合成である。2 層ニューラルネットワーク $\mathbf{W}_2\,\mathrm{relu}(\mathbf{W}_1\mathbf{x})$ は、その重みに関してまさにそのような合成になっており、ここが凸性の崩れる境界線である（35.3.5 節）。

35.3.4.2 log-sum-exp とソフトマックス共分散

1 つの基本的な損失が、上の 4 つの規則のすぐ外側に位置している。あらゆるソフトマックス交差エントロピー損失の心臓部にある log-sum-exp 関数 $\mathrm{lse}(\mathbf{x}) = \log \sum_{i=1}^{d} e^{x_i}$ である。幾何学的には、$\mathrm{lse}(\mathbf{x})$ は $\max_i x_i$ の滑らかな上からの近似であり、その超過量は決して $\log d$ を超えない。これを無理に合成規則へ当てはめるよりも、ヘッセ行列を通じて直接その凸性を示す方がよい。実はそのヘッセ行列は、驚くほど明快な確率論的解釈をもっている。

命題（log-sum-exp の凸性）。 lse **の点 x におけるヘッセ行列は**

$$\nabla^2 \mathrm{lse}(\mathbf{x}) = \mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top, \qquad \mathbf{s} = \mathrm{softmax}(\mathbf{x}), \tag{35.34}$$

で与えられ、これは s から抽出されたクラスのワンホットエンコーディングの共分散行列に一致する。すべての共分散行列は PSD であるため、lse は凸である。

証明。 1 回微分すると $\partial\,\mathrm{lse}/\partial x_i = e^{x_i}/\sum_k e^{x_k} = s_i$ となるため、lse の勾配はまさにソフトマックスベクトル $\mathbf{s}$ である。$s_i = e^{x_i}/\sum_k e^{x_k}$ を商の微分公式でもう 1 回微分すると次式を得る。

$$\frac{\partial s_i}{\partial x_j} = \frac{\delta_{ij}\, e^{x_i}}{\sum_k e^{x_k}} - \frac{e^{x_i}\, e^{x_j}}{\left(\sum_k e^{x_k}\right)^2} = s_i\,(\delta_{ij} - s_j),$$

これは (i,j) 成分において $\mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top$ と一致する（34.4 節で導出したソフトマックスのヤコビ行列と同じものである）。ここで Y を確率 $P(Y=i) = s_i$ で抽出されるランダムなクラス添字とし、$\mathbf{e}_Y$ をそのワンホットベクトルとする。このとき $\mathbb{E}[\mathbf{e}_Y] = \mathbf{s}$ かつ $\mathbb{E}[\mathbf{e}_Y\mathbf{e}_Y^\top] = \mathrm{diag}(\mathbf{s})$ となるため、ヘッセ行列は文字どおり $\mathrm{Cov}(\mathbf{e}_Y) = \mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top$ に一致する。すべての共分散行列は PSD である。実際、任意の方向 $\mathbf{v}$ に対して明示的に書けば次のようになる。

$$\mathbf{v}^\top \nabla^2 \mathrm{lse}(\mathbf{x})\,\mathbf{v} = \sum_i s_i v_i^2 - \left(\sum_i s_i v_i\right)^2 = \mathbb{E}\left[v_Y^2\right] - \left(\mathbb{E}[v_Y]\right)^2 = \mathrm{Var}(v_Y) \;\geq\; 0. \;\blacksquare$$

言い換えれば、方向 $\mathbf{v}$ に沿った log-sum-exp の曲率は、ソフトマックス分布のもとでの座標 v_Y の分散にほかならない。

この公式から、実用上の 3 つの帰結がただちに従う。第 1 に、ロジットの関数としての交差エントロピー損失 $\ell(\mathbf{z}) = \mathrm{lse}(\mathbf{z}) - z_y$（正解クラス y）は log-sum-exp から 1 次項を引いたものであるため凸である。第 2 に、ソフトマックス回帰（3.1 節）はアフィンなロジット $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ を $\ell(\mathbf{z})$ に入力するため、規則 2 により損失は重みに関して凸となる。すなわち、固定された特徴量の上で線形分類器を訓練する問題は常に凸最適化問題である。第 3 に、分散の公式から、全成分が 1 のベクトル $\mathbf{1}$ の方向に**ゼロ**固有値が存在することがただちに予測される。$\mathrm{Var}(v_Y) = 0$ が、すべての成分 v_i が等しいときに成り立つからである。この平坦な方向は、数値的に安定なソフトマックスの計算を可能にするシフト不変性 $\mathrm{lse}(\mathbf{x} + c\mathbf{1}) = \mathrm{lse}(\mathbf{x}) + c$ を反映している。また、まったく同じヤコビ行列 $\mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top$ が、あらゆるアテンションヘッドの逆伝播も支配している。アテンション重みはクエリ・キースコアのソフトマックスだからである（10.2 節）。

log-sum-exp の曲率は最大でどれほど大きくなり得るだろうか。演習問題 6 で証明するように、ロジットがどのような値であってもヘッセ行列の固有値が $\frac{1}{2}$ を超えることは決してないため、$\frac{1}{2}$ が大域的な滑らかさの定数となる。その上限にはどのような場所で近づき、また典型的なロジットにおける損失の曲率はどの程度なのだろうか。以下では、ランダムな 6 クラスのロジットベクトルを 10,000 個抽出し、温度の乗数 c でスケール変換して、全サンプルにわたるヘッセ行列の最大固有値と中央値の両方を報告する。

```
z = rng.normal(size=(10_000, 6))              # 10,000 draws of six logits
print('  c   largest eigenvalue   median')
for c in (0.1, 1.0, 3.0, 10.0, 30.0):
    s = np.exp(c * z - (c * z).max(axis=1, keepdims=True))
    s /= s.sum(axis=1, keepdims=True)         # softmax of each row
    hess = s[:, :, None] * np.eye(6) - s[:, :, None] * s[:, None, :]
    top = np.linalg.eigvalsh(hess)[:, -1]     # largest eigenvalue per draw
    print(f'{c:5.1f}  {top.max():14.4f}  {np.median(top):12.4f}')
```

```
  c   largest eigenvalue   median
 0.1          0.2213        0.1844
 1.0          0.4557        0.3071
 3.0          0.4991        0.2998
10.0          0.5000        0.0162
30.0          0.5000        0.0000
```

予測どおり、最大固有値は決して $\frac{1}{2}$ を超えず、ソフトマックス交差エントロピーがあらゆる温度においてロジットに関し $\frac{1}{2}$ 滑らかであることが確認できる。ロジットが小さいとき（$c = 0.1$）、ソフトマックスは 6 クラスにわたってほぼ一様となり、曲率は $\frac{1}{6}$ をわずかに上回るあたりに集まる。中程度のスケールでは典型的な曲率がピークを迎え、中央値は 0.3 付近となる。しかし c をさらに大きくすると二極化が生じる。典型的なサンプルは強い確信をもつ（ほぼワンホットになる）ため、曲率の中央値はゼロに向かって潰れていく。すなわち、確信度の高い予測の周りでは損失はほぼ完全に平坦になる。その一方で、10,000 個のサンプルにわたる**最大値**は $\frac{1}{2}$ へと迫っていく。これは上位 2 つのロジットがほぼ同値になり、ソフトマックスが確率を 2 クラスに均等に分け合う稀なサンプルから生じている。

35.3.5 凸性を超えて

深層ニューラルネットワークの損失ランドスケープは凸ではない。この最後の項では、わずか 2 つのスカラー重みを用いてその理由を示し、凸性なしに勾配降下法の線形収束率を保つより弱い条件を特定し、さらに無限に多くの大域的最小点が存在するときにオプティマイザがどのように 1 つの解を選び出すのかを考察する。

35.3.5.1 非凸なネットワーク

$f(a, b) = (ab - 1)^2$ を考えよう。これは 2 つのスカラー重みを直列に重ねた線形モデルに二乗損失を課したものであり、考え得る最も単純な「深層」ネットワークである。Figure 35.14 に示すように、その大域的最小点は双曲線 $\{ab = 1\}$ をなす。35.3.3 節では任意の凸関数の最小点集合が凸でなければならないことを証明したが、双曲線はどう見ても凸ではない。2 つの最小点 $(1, 1)$ と $(-1, -1)$ の中点は原点 $(0, 0)$ であり、そこでは $f(0, 0) = 1$ となって最小値 0 よりはるかに高い値をとる。原点は最小点ではなく停留点であり、そのヘッセ行列 $\nabla^2 f(0, 0)$ は固有値 ± 2 をもつため鞍点となっている。

実際のニューラルネットワークも、これと密接に関連する構造的な理由から凸性を失う。任意の

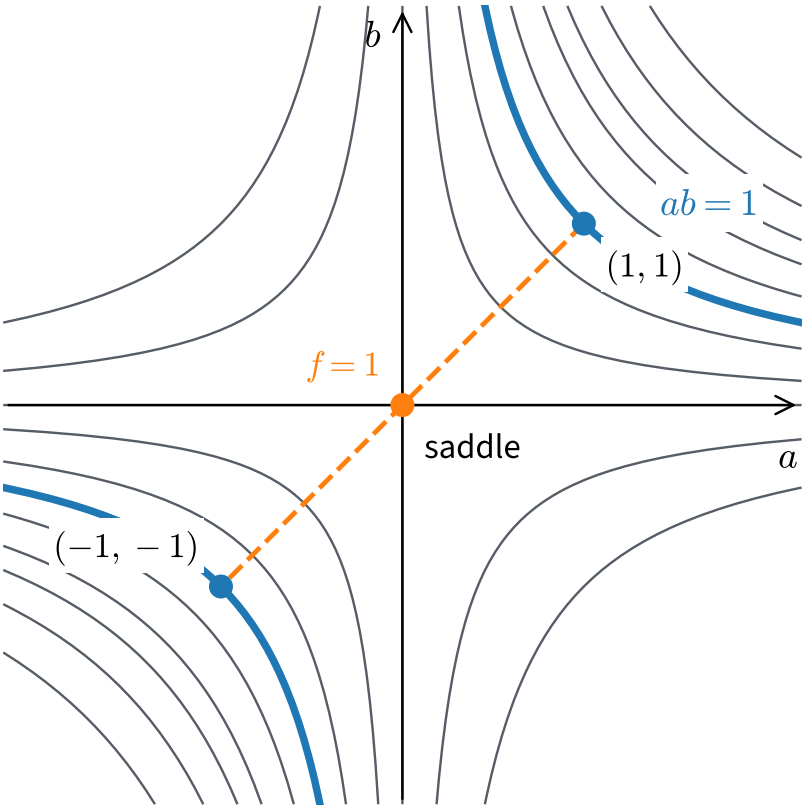


Figure 35.14: 2 重みモデル $(ab-1)^2$。その最小点は双曲線 $ab=1$ をなし、これは凸ではない。$(1,1)$ と $(-1,-1)$ の中点は原点であり、そこでは $f=1$ かつヘッセ行列の固有値は ± 2 である。

層の隠れユニットをその入力重みおよび出力重みとともに並べ替えても、ネットワークの入出力関数は変化しないため、あらゆる最小点にはパラメータ空間全体に散らばる膨大な数の対称なコピーが自動的に伴う。一般にこれらのコピーは互いに異なる点であり、そのうち 2 つを平均すると関数が変化して損失が増大する。ちょうど $(1,1)$ と $(-1,-1)$ の平均が原点の鞍点に落ちたのと同じである。異なる点を含みながらその中点を含まない最小点集合は、凸目的関数のものにはなり得ない。

2 個の ReLU ユニットしかもたない極小のネットワークでさえ、すでにこの破綻を示す。パラメータベクトル $(w_1, v_1, w_2, v_2) = (1, 1, -1, 1)$ は $v_1\,\mathrm{relu}(w_1 x) + v_2\,\mathrm{relu}(w_2 x)$ において $|x|$ を計算し、2 つのユニットを入れ替えてもまったく同じく $|x|$ を計算する。ところが、これら 2 つのパラメータベクトルを平均すると入力重みが打ち消し合ってゼロになり、ネットワークは恒等的にゼロを返す関数へと潰れてしまう。

35.3.5.2 Polyak–Łojasiewicz 条件

強凸性のもとでの線形収束率の証明がどのように進んだかを思い出してみよう。凸性を使ったのは最初のステップで Equation 35.33 を導くときだけであり、第 2 のステップでは凸性に一切触れていなかった。この中間の不等式には固有の名前が付いている。関数 f（下限 $f^\star$）が定数 $\mu > 0$ の **Polyak–Łojasiewicz（PL）不等式**を満たすとは、$\frac{1}{2}\|\nabla f(\mathbf{x})\|^2 \geq \mu\,(f(\mathbf{x}) - f^\star)$ がすべての $\mathbf{x}$ について成り立つことをいう。これは 1963 年に Polyak と Łojasiewicz が独立に導入した条件である (Polyak 1963; Łojasiewicz 1963)。直感的には、勾配が小さくなることが許されるのは損失がすでに最適値に近い場所に限られる、ということを述べている。

命題（PL 条件のもとでの線形収束率）。 *f を L 滑らかで下に有界（$f^\star = \inf f$）な関数とし、定数 μ の PL 不等式を満たすとする。このとき、$\eta = 1/L$ を用いた勾配降下法は $f(\mathbf{x}_k) - f^\star \leq (1 - \mu/L)^k\,(f(\mathbf{x}_0) - f^\star)$ を満たす。*

証明。 強凸性の証明の後半をそのまま繰り返せばよい。降下補題と PL 不等式を組み合わせると $f(\mathbf{x}_{k+1}) - f^\star \leq (1 - \mu/L)(f(\mathbf{x}_k) - f^\star)$ が得られ (Karimi et al. 2016)、これを k 回反復すれば証明が完了する。■

PL 条件がこれほど魅力的なのは、かなりの非凸性を許容する点にある。PL 関数は複数の最小点や、最小点がなす平坦な多様体、さらには負の曲率をもつ領域さえもち得る。しかし、譲れない要件が 2 つある。すべての停留点が大域的最小点でなければならないこと（$\nabla f(\mathbf{x}) = \mathbf{0}$ から $f(\mathbf{x}) = f^\star$ が強いられるため）、**かつ**最小点集合から離れるにつれて $f - f^\star$ が少なくとも 2 次のオーダーで増大しなければならないことである。4 次関数 x^4 は、なぜ第 2 の要件が重要かを示している。停留点は大域的最小点の 1 つしかもたないが、$\frac{1}{2}f'(x)^2 = 8x^6$ は μx^4 を $x \to 0$ においていかなる $\mu > 0$ に対しても上回り続けられないため、PL 条件を満たさない。まさにこの破綻こそ、35.3.3.3 節で勾配降下法が x^4 上を這うようにしか進めなかった理由である。要するに、強凸性から PL 条件は従う（先ほど証明したとおりである）が、その逆は成り立たない。

非凸な PL 関数の古典的な例は $f(x) = x^2 + 3\sin^2 x$ であり、その二階導関数 $2 + 6\cos 2x$ は -4 まで落ち込む一方で最大値 $L = 8$ に達する。区間 $[-5, 5]$ 上でその PL 定数を数値的に推定し、$x_0 = 4.5$ からステップサイズ $\eta = \frac{1}{16} = 1/(2L)$ を用いて勾配降下法を実行してみよう。（$1/(2L)$ を $1/L$ の代わりに選ぶのは、$\eta = 1/L$ では原点における局所的な縮小率 $1 - f''(0)\,\eta$ がゼロになり、漸近的な収束率が隠れてしまうためである。）$\eta = 1/(2L)$ においては、二次上界により 1 ステップあたり $\frac{3}{8L}f'(x_k)^2$ の減少が保証され、PL 不等式がこれを縮小率 $1 - \frac{3\mu}{4L}$ へと変換する。

```
f_pl = lambda x: x**2 + 3 * np.sin(x)**2       # nonconvex; f* = 0 at x = 0
df_pl = lambda x: 2 * x + 3 * np.sin(2 * x)
```

```
xs = np.linspace(-5, 5, 100001)
xs = xs[np.abs(xs) > 1e-3]                    # avoid 0/0 at the minimizer
mu = (0.5 * df_pl(xs)**2 / f_pl(xs)).min()
print(f'PL constant on [-5, 5]: mu = {mu:.4f}')
print(f"min f'' = {(2 + 6 * np.cos(2 * xs)).min():.2f}  (f is not convex)")
path = [4.5]
for _ in range(80):                           # gradient descent, eta = 1/16
    path.append(path[-1] - df_pl(path[-1]) / 16.0)
path = np.array(path)
ratios = f_pl(path[1:]) / f_pl(path[:-1])     # per-step contraction of f - f*
j = ratios.argmax()
print(f'guaranteed {1 - 3 * mu / 32:.4f}; largest ratio {ratios[j]:.4f}'
      f' from x = {path[j]:.2f}; last ratio {ratios[-1]:.4f}')
```

```
PL constant on [-5, 5]: mu = 0.1755
min f'' = -4.00  (f is not convex)
guaranteed 0.9835; largest ratio 0.9778 from x = 2.19; last ratio 0.2500
```

区間 $[-5, 5]$ 上での PL 定数は $\mu \approx 0.1755$ であり、f'' の最小値は -4 である。すなわち、この関数は非凸でありながら PL 条件を満たしている。この区間を調べるだけで十分である。というのも、その外側では比 $\frac{1}{2}f'(x)^2/f(x)$ はこの値を優に上回り続け、2 へと $|x| \to \infty$ で収束するからである。演習問題 4 でこの下界を証明する。

この μ を用いると、保証される縮小率は 1 ステップあたり約 0.98 となる。実測された最大の比は約 0.978 で、$x \approx 2.2$ の付近（$\frac{1}{2}f'(x)^2/f(x)$ が最小値 μ をとる場所）から生じており、そこでは理論保証はほぼタイトである。最小点の近傍では、この比は 0.25 に落ち着く。この極限値は局所的な二次近似から説明できる。0 の近くでは $f \approx \frac{1}{2}f''(0)\,x^2$（ただし $f''(0) = 8$）となるため、各ステップが x に $1 - 8/16 = \frac{1}{2}$ を掛け、x の二乗に比例するギャップには $\frac{1}{4}$ を掛けるからである。

PL 条件は空間の一部でのみ成り立つこともある。$(ab-1)^2$ では勾配が $\frac{1}{2}\|\nabla f\|^2 = 2(a^2+b^2)f$ を満たすため、PL 条件は定数 $2r^2$ で領域 $\{a^2+b^2 \geq r^2\}$ 上において成り立つ一方、原点の鞍点の近くではいかなる正の定数でも成り立たない。幅の広いネットワークの理論解析では、補間解の近傍においてこのような局所的な PL 不等式が示されている (C. Liu et al. 2022)。ただし、これが与えるのは訓練損失の収束率であり、汎化については何も語らない。

35.3.5.3 暗黙的バイアス

損失が凸であるか否かにかかわらず多数の大域的最小点をもつとき、そのどれに到達するかはオプティマイザが決める。すなわち、オプティマイザは**暗黙的バイアス**（implicit bias）をもつ。深層学習において、これは単なる珍しい現象ではない。訓練例よりも多くの重みをもつネットワークは、何通りもの方法でデータに適合でき、そのすべてが同じ訓練損失を達成しながら未見の点では異なる予測を出すため、どの解を選ぶかがモデルの汎化性能を左右するのである。最も単純な例は、訓練例が重みより少なく $\mathbf{X}$ が行フルランクである劣決定最小二乗問題である。すべての勾配 $\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$ は $\mathbf{X}$ の行空間に属するため、$\mathbf{w}_0 = \mathbf{0}$ から出発した反復点は決して行空間の外へ出ない。したがって極限は行空間内の唯一の補間解、すなわち擬似逆行列が計算する**最小ノルム**補間解となる（演習問題 4、33.5 節）。分離可能なデータに対するロジスティック回帰では損失はそもそも最小点をもたないが、反復点の方向は最大マージン分離超平面、すなわちハードマージンサポートベクターマシンの解へと収束する (Soudry et al. 2018)（演習問題 8）。どの最小点が選ばれるかは、モデル・損失・初期点・更新則に依存する。たとえば固定された対角前処理行列 $\mathbf{D}$ を用いると、$\mathbf{w}^\top\mathbf{D}^{-1}\mathbf{w}$ を $\|\mathbf{w}\|^2$ の代わりに最小化する補間解へと到達する（演習問題 7）。その

ため、座標ごとにステップサイズを変える Adam（9.5 節）は、勾配降下法が選ぶ解と同じ解を選ぶとは限らない。

本節のすべての結果は、厳密な勾配が得られ、かつ重みが $\mathbb{R}^d$ 全体を自由に動けることを仮定している。実際の訓練では平均の意味でしか正しくないミニバッチ勾配を用い（35.4 節）、また球や単体や半空間に閉じ込められた重みベクトルを扱うには 35.6 節の射影や乗数が必要となる。

35.3.6 まとめ

凸関数はその弦の下側にある。微分可能な関数ではこれはすべての接線の上側にあることと同値であり、2 回微分可能な関数ではヘッセ行列が PSD であることと同値となる。また、角においては劣勾配が勾配の役割を担う。ただ 1 つの勾配がいたるところで関数を下から抑えるため、停留点は大域的最小点となり、降下補題は勾配降下法の大域的な収束率へと転じる。すなわち、$L\|\mathbf{x}_0 - \mathbf{x}^\star\|^2/(2k)$（滑らかな凸関数 f の場合）となり、強凸性のもとでは 1 ステップあたり $1-\mu/L$ の縮小率、したがっておよそ $\kappa \log(1/\epsilon)$ ステップとなる。イェンセンの不等式は期待値に対する弦不等式であり、$D_{\mathrm{KL}} \geq 0$ を導く。4 つの閉包規則によってヒンジ損失、ℓ_1 ノルム、ロジスティック損失の凸性が保証され、log-sum-exp（およびそれに伴うソフトマックス交差エントロピー）はそのヘッセ行列が共分散となるため凸である。ニューラルネットワークの損失は一般に非凸である。しかし PL 不等式が成り立つ領域では凸性なしでも線形収束率が維持され、多数の最小点が存在する場合にはオプティマイザのダイナミクスがそのうちの 1 つを選び出す。

35.3.7 演習問題

1. **ノルムと最大値の劣微分**。定義 Equation 35.28 に基づき、以下の各劣微分を求めよ。
   1. $\partial \|\mathbf{x}\|_2$ をすべての $\mathbf{x} \in \mathbb{R}^d$（$\mathbf{x} = \mathbf{0}$ を含む）において求めよ。
   2. $f(\mathbf{x}) = \max_i x_i$ とし、$\mathcal{A}(\mathbf{x}) = \{i : x_i = f(\mathbf{x})\}$ をその有効座標の集合とする。$\partial f(\mathbf{x})$ が $\{\mathbf{e}_i : i \in \mathcal{A}(\mathbf{x})\}$ の凸包であることを示せ。最大プーリングを通じた誤差逆伝播法は、受け取った勾配を最大値をとる単一の入力へと渡すため、この凸包の 1 つの頂点 $\mathbf{e}_i$ を用いている。2 つ以上の入力が最大値で並んだとしよう。$-\mathbf{e}_i$ に沿った微小なステップでは f が減少しないことを示せ。どのような $\mathbf{g} \in \partial f(\mathbf{x})$ に対して $-\mathbf{g}$ は降下方向となるか。また、そのうち f の $-\mathbf{g}/\|\mathbf{g}\|$ に沿った方向微分を最も負にするものはどれか。
   3. (a) と最適性条件 $\mathbf{0} \in \partial(\cdot)$ を用いて、$\lambda\|\mathbf{w}\|_2 + \frac{1}{2}\|\mathbf{w}-\mathbf{z}\|^2$ を $\mathbf{w} \in \mathbb{R}^d$ について最小化せよ（これにより $\lambda\|\cdot\|_2$ の近接作用素が得られる）。解のゼロパターンを、座標ごとのソフト閾値処理 $w_i = \mathrm{sign}(z_i)\max(|z_i| - \lambda, 0)$ のゼロパターンと比較せよ。$\mathbf{w}$ が 1 つの隠れユニットの出力重みをまとめたベクトルであるとき、このペナルティはどのような働きをするか。

2. **凸結合とイェンセンの不等式**。$\sum_{i=1}^N \theta_i \mathbf{x}_i$（ただし $\theta_i \geq 0$、$\sum_i \theta_i = 1$）により N 個の点の凸結合を表す。
   1. N に関する帰納法により、凸集合 C が C の任意の点の凸結合を含むことを示せ。これは 35.3.1 節における凸包の特徴づけを完結させる包含関係である。
   2. (a) を凸関数 f のエピグラフに適用し、イェンセンの不等式の有限形式 Equation 35.29 を導け。
   3. f を 2 回微分可能で、ある区間上で $\alpha \leq f'' \leq \beta$ を満たす関数とし、その区間は確率変数 Z（平均 m、分散 s^2）のとる値をすべて含むとする。イェンセンギャップ $\mathbb{E}[f(Z)] - f(m)$ が $\alpha s^2/2$ と $\beta s^2/2$ の間に収まることを示せ。本文の 2 点の例、すなわち $Z \in \{0, 2\}$、$f(x) = x^2$（このとき $\alpha = \beta = 2$）においてこの上下界を確認せよ。さらにこの結果を $f = -\log$ に適用し、$\log \mathbb{E}[Z] - \mathbb{E}[\log Z]$ を（Z が $[a, b] \subset (0, \infty)$ に値をとるとき）上下から評価せよ。

3. **最小点の存在。** $\ell(\mathbf{w}) = \sum_i \log(1 + e^{-y_i \mathbf{w}^\top \mathbf{x}_i})$ をラベル $y_i \in \{\pm 1\}$ に対するロジスティック損失とし、$\mathbf{X}$ を $\mathbf{x}_i^\top$ を行にもつ行列とする。35.3.3 節の一意性の命題によれば、狭義凸関数の最小点は高々 1 つであり、空でない内部をもつ空でない閉凸集合上の連続な強凸関数はただ 1 つの最小点をもつ。

   1. ある $\mathbf{w}$ が $y_i \mathbf{w}^\top \mathbf{x}_i > 0$ をすべての i について満たす、すなわちデータが原点を通る超平面によって線形分離可能であるとする。このとき $\inf_{\mathbf{w}} \ell(\mathbf{w}) = 0$ であり、かつ ℓ が最小点をもたないことを示せ。
   2. ℓ は、$\mathbf{X}$ が列フルランクであるとき常に狭義凸となることを示せ。これが (a) と矛盾しないのはなぜか。
   3. $\ell(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$ が、任意の $\lambda > 0$ と任意のデータセットに対してただ 1 つの最小点をもつことを示せ。このペナルティは一意性の命題のどの仮定を補っているか。

4. **凸性を超えた PL 条件。** $f(x) = x^2 + 3\sin^2 x$ を調べるコードセルでは、その PL 定数を $\mu \approx 0.1755$（区間 $[-5, 5]$ 上）と測定した。

   1. $|f'(x)|$ を下から、$f(x)$ を上から、$|x| \geq 5$ の範囲で評価せよ。以上から、PL 不等式 Equation 35.33 が $\mathbb{R}$ 全体にわたってその区間上で測定した定数のまま成り立つことを結論せよ。
   2. $\mathbf{X} \in \mathbb{R}^{n \times d}$（$n < d$）を行フルランクとし、$f(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ とする。f は強凸ではなく無限に多くの最小点をもつが、$\mu = \lambda_{\min}(\mathbf{X}\mathbf{X}^\top)$（$\mathbf{X}\mathbf{X}^\top$ の最小固有値）として PL 不等式を満たすことを示せ。このとき $\eta = 1/L$ の勾配降下法はどのような線形収束率を達成し、L は $\mathbf{X}\mathbf{X}^\top$ とどう関係するか。

5. [code] **距離の縮小。●** 収束率 Equation 35.32 は 1 ステップあたりの因子 $1 - \mu/L$ を $f - f^\star$ に対して保証するが、収束率検証セルでは $(1-\mu/L)^2$ が測定された。f を $\mu\mathbf{I} \preceq \nabla^2 f \preceq L\mathbf{I}$ を満たす 2 回連続微分可能な関数とし、$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$ とする。ここでは 35.1 節の 2 つの事実、すなわち二次上界 $f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$ と、二次関数における最良の固定ステップサイズ $\eta^\star = 2/(\mu + L)$ を用いる。

   1. $\nabla f(\mathbf{x}_k) = \bar{\mathbf{H}}_k(\mathbf{x}_k - \mathbf{x}^\star)$（ただし $\bar{\mathbf{H}}_k = \int_0^1 \nabla^2 f(\mathbf{x}^\star + s(\mathbf{x}_k - \mathbf{x}^\star))\, ds$）と表すことで、次を示せ。

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \leq \max\bigl(|1 - \eta\mu|,\, |1 - \eta L|\bigr)\, \|\mathbf{x}_k - \mathbf{x}^\star\|. \tag{35.35}$$

   2. $\eta = 1/L$、$\kappa = L/\mu$ とおく。(a) と、ともに点 $\mathbf{x}^\star$ においてとった下界 Equation 35.27 および二次上界を組み合わせることで、$f(\mathbf{x}_k) - f^\star \leq \kappa\,(1 - \mu/L)^{2k}\,(f(\mathbf{x}_0) - f^\star)$ を示せ。どの k 以降でこの上界は Equation 35.32 より強くなるか。

   3. $f - f^\star$ の 1 ステップあたりの縮小率を、収束率検証セルの $\kappa = 10$ の計画行列に対して $\eta^\star = 2/11$ のもとで予測せよ。（ヒント：両方の固有成分に絶対値 $9/11$ の数が掛けられる。）そのセルのステップサイズを変更して予測を検証し、$\eta = 1/L$ のときとは異なり最初の 1 回目の比がもはや特別ではなくなる理由を説明せよ。

6. **最大値の平滑化。●** ヒンジ型損失や最悪ケース目的関数の形であるアフィン関数の最大値は、2 つの要素が同値になる場所ごとに折れ点をもつ。温度 $\tau > 0$ に対し、$\mathrm{lse}_\tau(\mathbf{z}) = \tau \,\mathrm{lse}(\mathbf{z}/\tau)$（$\mathbf{z} \in \mathbb{R}^m$）とおく。この平滑化手法は Nesterov (2005) によるものである。

   1. $\max_j z_j \leq \mathrm{lse}(\mathbf{z}) \leq \max_j z_j + \log m$ を示し、そこから $\max_j z_j \leq \mathrm{lse}_\tau(\mathbf{z}) \leq \max_j z_j + \tau \log m$ を結論せよ。

2. Equation 35.34 を用い、$\nabla^2 \mathrm{lse}_\tau(\mathbf{z})$ の最大固有値が高々 $1/(2\tau)$ であることを、$\mathrm{Var}(v_Y)$ を $\mathbb{E}[(v_Y - c)^2]$（c は適切に選んだ定数）で上から抑えることによって示せ。この上界は達成されるか。

3. $F(\mathbf{w}) = \max_j (\mathbf{A}\mathbf{w} + \mathbf{r})_j$ および $F_\tau(\mathbf{w}) = \mathrm{lse}_\tau(\mathbf{A}\mathbf{w} + \mathbf{r})$ とし、F_τ は距離 R 以内（$\mathbf{w}_0$ から測って）に最小点をもつと仮定する。(b) を用いて、F_τ が L 滑らか（ただし $L = \|\mathbf{A}\|_2^2/(2\tau)$）であり、かつ k ステップの勾配降下法を F_τ に対して $\eta = 1/L$ で実行すると次が成り立つことを示せ。

$$F(\mathbf{w}_k) - \min F \le \tau \log m + \frac{\|\mathbf{A}\|_2^2 R^2}{4\tau k}. \tag{35.36}$$

この上界を最小化するように τ を選び、精度 ϵ を保証するステップ数を求めよ。勾配降下法を誤差 $O(LR^2/k^2)$ の加速法に置き換えると、必要なステップ数はどう変わるか。

7. [code] **暗黙的バイアスとオプティマイザ。**●演習問題 4（33.5 節）では、$\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$（ただし $\mathbf{X} \in \mathbb{R}^{n \times d}$ は $n < d$ かつ行フルランク）に対する $\mathbf{w}_0 = \mathbf{0}$ からの勾配降下法が、$\mathbf{X}$ の行空間内にとどまり最小ノルム補間解 $\mathbf{X}^+\mathbf{y}$ へ収束することを机上で示した。ここでは初期点を $\mathbf{w}_0 = \mathbf{0}$ に固定したままアルゴリズムを変える。ランダムな 4×10 の系をとり、各手法の極限とそのノルムを `np.linalg.pinv(X) @ y` と比較せよ。

   1. $\eta = 1/\lambda_{\max}(\mathbf{X}^\top\mathbf{X})$ で勾配降下法を実行し、$\mathbf{w}_k$ の行空間に直交する成分のノルムを射影行列 $\mathbf{I} - \mathbf{X}^+\mathbf{X}$ を用いて計算し追跡せよ。
   2. ヘビーボールモメンタムは $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\nabla f(\mathbf{w}_k) + \beta(\mathbf{w}_k - \mathbf{w}_{k-1})$ と更新する。その反復点が行空間にとどまることを示し、極限を予測せよ。(a) のステップサイズと $\beta = 0.9$ を用いて予測を検証せよ。
   3. 更新を対角前処理付きステップ $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\,\mathbf{D}\mathbf{X}^\top(\mathbf{X}\mathbf{w}_k - \mathbf{y})$ に置き換えよ。ここで $\mathbf{D}$ は成分を $[0.1, 10]$ から一様に抽出した固定対角行列であり、$\eta = 1/\lambda_{\max}(\mathbf{D}^{1/2}\mathbf{X}^\top\mathbf{X}\mathbf{D}^{1/2})$ とする。変数変換 $\mathbf{u} = \mathbf{D}^{-1/2}\mathbf{w}$ を用いて極限を閉じた式で求め、数値的に検証せよ。この極限は依然として最小ノルム補間解か。この結果は、座標ごとのステップサイズが Adam（9.5 節）のように $\mathbf{D}$ として働く適応的手法の暗黙的バイアスについて何を示唆するか。
   4. 厳密な巡回座標降下法を 1000 スイープ実行せよ。ただし $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$、$\mathbf{b} = \mathbf{X}^\top\mathbf{y}$ とし、更新 $w_j \leftarrow (b_j - \sum_{i \ne j} A_{ji} w_i)/A_{jj}$ を $j = 1, \dots, d$ の順に適用する。極限がデータを補間することを確認せよ。それは最小ノルム補間解か。また、行空間の議論のどの段階が成り立たなくなるか。座標を一様ランダムに選んで同じ実行を 3 回繰り返せ。極限は乱数の抽出結果に依存するか。

8. [extended] **勾配降下法の最大マージンバイアス。**●●原点を通る超平面によって分離可能なデータ上では、ロジスティック損失は最小点をもたない（演習問題 3 の (a)）。それでも Soudry et al. (2018) は、この損失に対する勾配降下法が一定の方向へ発散することを証明している。すなわち、反復点は $\mathbf{w}_k = \hat{\mathbf{w}}\log k + \rho_k$（ただしほとんどすべてのデータセットに対して ρ_k は有界）を満たす。ここで

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \|\mathbf{w}\| \quad \text{subject to} \quad y_i \mathbf{w}^\top \mathbf{x}_i \ge 1 \text{ for all } i \tag{35.37}$$

はハードマージン SVM の解であり、そのマージンは $\gamma = 1/\|\hat{\mathbf{w}}\|$ である。ベクトル $\hat{\mathbf{w}}$ は $\hat{\mathbf{w}} = \sum_i \alpha_i y_i \mathbf{x}_i$（$\alpha_i \ge 0$、かつ $\alpha_i = 0$ が $y_i \hat{\mathbf{w}}^\top \mathbf{x}_i = 1$ 以外のすべての点で成り立つ）によって特徴づけられ、等号を満たす点がサポートベクターである。

1. 発散の方向を発見的に導け。ロジスティック損失をその裾 e^{-q}（ただしマージン $q = y_i \mathbf{w}^\top \mathbf{x}_i$）で置き換え、$\mathbf{w}_k = g(k)\,\mathbf{u} + O(1)$（$\|\mathbf{u}\| = 1$、$g(k) \to \infty$）と仮定する。負の勾配が漸近的に、点 $y_i \mathbf{x}_i$（最小のマージン $y_i \mathbf{u}^\top \mathbf{x}_i$ をもつもの）の非負結合になることを示せ。以上から、$\mathbf{u}$ が $\hat{\mathbf{w}}$ に比例することを結論せよ。
2. 増大の速さを導け。勾配フロー $\dot{\mathbf{w}} = -\nabla \ell(\mathbf{w})$ においてサポートベクターの項のみを残し、$\dot{g} \approx c\, e^{-\gamma g}$ がある定数 $c > 0$ に対して成り立つことを示して、$\|\mathbf{w}(t)\| \approx \|\hat{\mathbf{w}}\| \log t$ を結論せよ。
3. サポートベクターとして $\mathbf{x}_1 = (0.5, 1.5)$ と $\mathbf{x}_2 = (1.5, 0.5)$（ラベル $+1$）、それらの符号反転（ラベル -1）、さらに各クラス 6 個ずつ計 12 個のランダムな点（$y_i \hat{\mathbf{w}}^\top \mathbf{x}_i > 1$ を $\hat{\mathbf{w}} = (\tfrac{1}{2}, \tfrac{1}{2})$ に対して満たすもの）をとる。$\mathbf{X}$ をサイズ 16×2 で各行が $\mathbf{x}_i^\top$ である行列とする。ロジスティック損失の和に対する勾配降下法を $\mathbf{w}_0 = \mathbf{0}$ から $\eta = 1/\|\mathbf{X}\|_2^2$ で 10^6 ステップ実行せよ（ベクトル化した NumPy なら数秒で終わる）。対数軸にとった k に対して、ノルム $\|\mathbf{w}_k\|$、訓練損失、$\mathbf{w}_k$ と $\hat{\mathbf{w}}$ のなす角、およびマージンギャップ $\gamma - \min_i y_i \mathbf{w}_k^\top \mathbf{x}_i / \|\mathbf{w}_k\|$ をプロットせよ。さらに、$\|\mathbf{w}_k\|$ の $\log k$ に対する傾きを (b) と比較せよ。
4. 訓練誤差は最初の数ステップ以内にゼロへ到達する。$k = 10^3$ と $k = 10^6$ の間で、マージンギャップと訓練損失はそれぞれ何倍変化するか。この比較は、訓練誤差がゼロになった瞬間に訓練を打ち切ることについて何を示唆するか。

9. **対数分配関数の凸性。** この演習問題では、log-sum-exp の命題を 36.4 節の土台となる対数分配関数へ一般化する。有限集合 $\mathcal{X}$、重み $h(x) > 0$、統計量 $\mathbf{T}(x) \in \mathbb{R}^p$、および自然パラメータ $\eta \in \mathbb{R}^p$（ここではステップサイズではない）に対し、$A(\eta) = \log \sum_{x \in \mathcal{X}} h(x) \exp(\eta^\top \mathbf{T}(x))$ および $p_\eta(x) = h(x) \exp(\eta^\top \mathbf{T}(x) - A(\eta))$ とおく。

   1. $\nabla A(\eta) = \mathbb{E}_\eta[\mathbf{T}]$ および $\nabla^2 A(\eta) = \mathrm{Cov}_\eta[\mathbf{T}]$ を示せ。ただし期待値と共分散は p_η のもとでとる。
   2. A が凸であること、および狭義凸となるための必要十分条件が、いかなる非ゼロの $\mathbf{v}$ に対しても $\mathbf{v}^\top \mathbf{T}(x)$ が $\mathcal{X}$ 上で定数にならないことであると結論せよ。
   3. Equation 35.34 を $\mathcal{X} = \{1, \dots, d\}$、$\mathbf{T}(x) = \mathbf{e}_x$、$h \equiv 1$ とおくことで復元し、A が平坦になる方向を答えよ。
   4. 観測値 $x_1, \dots, x_n$ と平均統計量 $\bar{\mathbf{T}} = \frac{1}{n}\sum_i \mathbf{T}(x_i)$ に対し、平均負の対数尤度 $A(\eta) - \eta^\top \bar{\mathbf{T}} - \frac{1}{n}\sum_i \log h(x_i)$ が η に関して凸であること、および最小点が $\mathbb{E}_\eta[\mathbf{T}] = \bar{\mathbf{T}}$（モーメントマッチング）を満たすことを示せ。(c) のソフトマックスの場合において、1 度も観測されないクラスがあると最小点が存在しないのはなぜか。

35.4 確率的勾配法

ニューラルネットワークの訓練では、損失の厳密な勾配を使うことはほとんどない。更新のたびにデータセット全体を走査する代わりに、各ステップでは数百個の例からなるランダムなミニバッチ上で勾配を計算する。ここまでのすべての節では厳密な勾配を仮定しており、そこでは曲率がステップ数を決定し、凸性によってあらゆる停留点が大域的最小点となった。ミニバッチ勾配は平均的には正しい方向を指すもののノイズを含んでおり、そのノイズによって最適化の中心的な問いは「実行に何ステップ必要か」から「最小点にどこまで近づけるか」へと変化する。本節では、勾配ノイズがもたらす代償と、それを克服する方法を考察する。

考え得る最も単純な損失でさえ、その代償は明らかになる。$f(x) = \frac{1}{2}x^2$（最小点は 0）を考え、すべての勾配に単位分散の独立なノイズを加えながら、大きさ $\eta = 0.1$ の定数ステップをとるとしよう。厳密な勾配であれば、各ステップで x は 10 分の 1 だけ縮小し、反復点を望むだけ 0 に近づけることができる。しかしノイズを含む勾配では、反復点は決して落ち着かない。その二乗平均は $0.1/1.9 \approx 0.053$ に収束するため、何ステップ進めようとも最小点から約 0.23 離れたあ

たりをさまよい続ける。損失をどこまで下げられるかは、ノイズの大きさ、ステップサイズ、そして許容できるステップ数という 3 つの量で決まる。

まず、ミニバッチ勾配のノイズを定量化する。次に二次目的関数上で、固定ステップサイズがどこで頭打ちになるか、またそこに到達するまでに何ステップかかるかを計算する。続いて、減衰するステップサイズがその頭打ちをどう解消するか、そして $1/k$ スケジュールがどのような定数を必要とするかを示したのち、固定された予算に合わせて調整した定数ステップが凸損失と非凸損失の両方で達成する誤差の上界を導く。最後に、有限個の訓練セット上で保存しておいたスナップショット勾配を用いてノイズを打ち消す方法を示す。本節全体を通じて念頭に置くべき要点は 3 つある。ノイズフロアはステップサイズに比例すること、$1/k$ スケジュールがその収束率を達成するには十分大きな定数が必要であること、そして K ステップの固定予算では $1/\sqrt{K}$ に比例するステップサイズが求められることである。分散低減（variance reduction）を扱う最後の小節は、初読時には飛ばしても構わない。

35.4.1 ミニバッチ勾配

35.4.1.1 全勾配のコスト

典型的な訓練目的関数は、n 個の例の損失を平均したものである。

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}),$$

ここで、f_i は例 i における損失であり、$\mathbf{x} \in \mathbb{R}^d$ はモデルパラメータを保持する。その勾配 $\nabla f = \frac{1}{n}\sum_i \nabla f_i$ も平均であるため、評価には n 個すべての例を参照する必要がある。$n = 10^9$ のとき、勾配降下法の 1 ステップにはデータセット全体の 1 パスが必要となり、しかも勾配降下法は通常数千ステップを要する。本章の通し例である $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$（曲率が 1000 倍異なる）でさえ、最適な固定ステップで誤差を 100 万分の 1 に縮小するには約 7,000 回の反復が必要となる（35.1 節）。**確率的勾配降下法**（stochastic gradient descent、SGD）は、全体の平均をランダムな**ミニバッチ** $\mathcal{B}$（b 個のインデックスからなる）上の平均に置き換えることで、このコストを回避する。

$$\widehat{\mathbf{g}}_b(\mathbf{x}) = \frac{1}{b}\sum_{i \in \mathcal{B}} \nabla f_i(\mathbf{x}), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k\, \widehat{\mathbf{g}}_b(\mathbf{x}_k). \tag{35.38}$$

これにより、各ステップでは b 個の例だけを調べればよく、n 個すべてを調べる必要はなくなる。サイズ $b = 256$ のミニバッチは、10^9 個の例全体を 1 パス走査する場合と比べて約 400 万倍安価である。本節を通して k はオプティマイザのステップを表し、ステップサイズ η_k は k に応じて変化してよいものとする。解析にあたっては、毎ステップ新たなミニバッチを抽出し、その b 個のインデックスは $\{1, \dots, n\}$ 上で独立かつ一様であるとする（復元サンプリング）。実際の訓練ループでは通常、データセットをシャッフルして非復元サンプリングを行う。この場合、以下で導出する勾配の分散に $(n-b)/(n-1)$ が掛かるが、この係数は $b \ll n$ である限り 1 に近い（9.2 節）。

35.4.1.2 不偏性と $1/b$ 則

ミニバッチ勾配はどの程度正確だろうか。その質は 2 つの量で捉えられる。1 つは平均であり、平均的に正しい方向を指しているかを表す。もう 1 つは真の勾配からの平均二乗距離であり、これ

を**ノイズパワー**と呼ぶ。このノイズパワーは、個々の例の勾配がその平均の周りにどれだけ広く散らばっているかに依存し、共分散行列によって測られる。

命題（ミニバッチ勾配は不偏であり、ノイズパワーは $\operatorname{tr}\Sigma/b$ である）。$\mathbf{x}$ を固定し、次式とおく。

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \big(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\big)\big(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\big)^{\top}$$

これは一様に抽出された 1 つの例の勾配の共分散である。b 個からなる $\mathcal{B}$ のインデックスが $\{1,\dots,n\}$ 上で独立かつ一様であるならば、次が成り立つ。

$$\mathbb{E}\big[\hat{\mathbf{g}}_b(\mathbf{x})\big] = \nabla f(\mathbf{x}), \qquad \mathbb{E}\big[\|\hat{\mathbf{g}}_b(\mathbf{x}) - \nabla f(\mathbf{x})\|^2\big] = \frac{\operatorname{tr}\Sigma}{b}. \tag{35.39}$$

証明。一様な単一のインデックス i に対して、$\mathbb{E}[\nabla f_i(\mathbf{x})] = \frac{1}{n}\sum_i \nabla f_i(\mathbf{x}) = \nabla f(\mathbf{x})$ である。したがって、ミニバッチ内の各項は平均 $\nabla f(\mathbf{x})$ を持ち、その平均も同様である。b 個のインデックスは独立であり、各項は共分散 Σ を持つため、その和の共分散は $b\Sigma$ となり、b で割ると共分散が $1/b^2$ 倍されて Σ/b となる。最後に、平均ゼロの任意の確率ベクトルの期待二乗ノルムは、その共分散行列のトレースに等しい。$\blacksquare$

最適化の過程では、$\mathbf{x}$ は現在の反復点 $\mathbf{x}_k$ であり、ミニバッチは過去のステップとは独立に抽出されるため、両方の等式は $\mathbf{x}_k$ を条件として成り立つ。なお、不偏性は一様サンプリングに依存する。サンプラが例 i を確率 p_i で抽出する場合、推定量を不偏に保つには各勾配に $1/(np_i)$ の重みを掛け直す必要がある。

ノイズパワーは $1/b$ のオーダーで減衰するため、その二乗平均平方根振幅は $1/\sqrt{b}$ の速さでしか減衰しない。つまり、勾配ノイズを 10 分の 1 に減らすには、1 ステップあたり 100 倍の例が必要になる。一方、真の勾配信号の二乗ノルムはバッチサイズによらず $\|\nabla f\|^2$ である。信号とノイズのパワーが等しくなるのはバッチサイズが $b_{\text{noise}} = \operatorname{tr}\Sigma/\|\nabla f\|^2$ のときであり、本編ではこれを**勾配ノイズスケール**（gradient-noise scale）と呼んでいる（9.8 節）。演習問題 1 では、曲率で重み付けした対応量を導出する。

具体的なデータ上でこのノイズがどれほど大きいかを確認するため、以下のステップサイズ実験でも再利用する小規模なロジスティック回帰問題を設定しよう。この問題は、標準正規分布に従う特徴量を持つ $n = 200$ 個の例 $\mathbf{a}_i \in \mathbb{R}^5$ と、ラベル $y_i \in \{-1, +1\}$（真の重み $(1, -2, 1.5, 0, 2)$ を持つロジスティックモデルから生成）からなり、正則化付き損失は次式で与えられる。

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \log\big(1 + e^{-y_i \mathbf{a}_i^{\top}\mathbf{x}}\big) + \frac{0.1}{2}\|\mathbf{x}\|^2.$$

二次のペナルティ項により、f は $\mu \geq 0.1$ の強凸関数となる。例 i の勾配は $\nabla f_i(\mathbf{x}) = -y_i s_i \mathbf{a}_i + 0.1\,\mathbf{x}$ であり、ここで $s_i = 1/(1 + e^{y_i \mathbf{a}_i^{\top}\mathbf{x}})$ はモデルが誤ったラベルに割り当てる確率である。以下のセルでは、データセットを生成して損失を定義するとともに、全 n 個の例ごとの勾配を $n \times d$ 行列の各行として返すベクトル化された補助関数を定義する。

```
n, d, reg = 200, 5, 0.1                          # examples, parameters, l2 weight
x_true = np.array([1.0, -2.0, 1.5, 0.0, 2.0])
rng = np.random.default_rng()
A = rng.normal(size=(n, d))                       # row i: the features a_i
```

```
    y = np.where(rng.random(n) < 1 / (1 + np.exp(-A @ x_true)), 1.0, -1.0)

    def loss(x, A, y, reg):
        """The regularized logistic loss f(x)."""
        return np.mean(np.log1p(np.exp(-y * (A @ x)))) + 0.5 * reg * x @ x

    def example_grads(x, A, y, reg):
        """The n x d matrix whose row i is the gradient of f_i at x."""
        s = 1 / (1 + np.exp(y * (A @ x)))            # probability of the wrong label
        return -(y * s)[:, None] * A + reg * x
```

上の命題は厳密に成り立つため、シミュレーションは主に検算と、この問題における $\operatorname{tr}\Sigma$ の実際のスケールを示す役割を果たす。以下では、各サイズにつき $10{,}000$ 個のランダムなミニバッチを $\mathbf{x} = \mathbf{0}$ で評価し、経験誤差を $\operatorname{tr}\Sigma/b$ と比較する。

```
    G = example_grads(np.zeros(d), A, y, reg)        # per-example gradients at x = 0
    g = G.mean(axis=0)                                # the full gradient
    tr_sigma = ((G - g)**2).sum(axis=1).mean()
    print(f'tr Sigma = {tr_sigma:.3f}')
    for b in [1, 4, 16, 64]:
        idx = rng.integers(0, n, size=(10000, b))    # minibatches, with replacement
        err = ((G[idx].mean(axis=1) - g)**2).sum(axis=1).mean()
        print(f'b = {b:2d}: measured {err:.4f}, tr Sigma / b = {tr_sigma / b:.4f}, '
              f'ratio {err * b / tr_sigma:.3f}')
```

```
tr Sigma = 1.124
b =  1: measured 1.1077, tr Sigma / b = 1.1245, ratio 0.985
b =  4: measured 0.2787, tr Sigma / b = 0.2811, ratio 0.991
b = 16: measured 0.0703, tr Sigma / b = 0.0703, ratio 1.001
b = 64: measured 0.0176, tr Sigma / b = 0.0176, ratio 1.002
```

1 例あたりのノイズパワー $\operatorname{tr}\Sigma$ は約 1.1 であり、4 つのバッチサイズすべてにわたって、実測誤差はサンプリング変動による数パーセント以内の精度で $\operatorname{tr}\Sigma/b$ と一致している。1 ステップのノイズがわかったところで、そのノイズが最適化の実行全体にわたってどのような影響を及ぼすかを問うことができる。

35.4.1.3 ノイズフロア

勾配ノイズは多数のステップにわたって平均化されて消えるのか、それとも蓄積するのか。9.2 節では、SGD が谷の内部で落ち着く点群を測定し、その半径を導出した。ここでは軌跡全体を閉じた形で解くことで、その点群に到達するまでに何ステップかかるかも明らかにする。最も単純な場合である二次関数の 1 座標 $f(x) = \tfrac{1}{2}\lambda x^2$（曲率 $\lambda > 0$、最小点 0）に、すでにその仕組みのすべてが含まれている。確率的勾配を、真の勾配 λx_k にランダムなノイズ ξ_k（x_k を与えたときの条件付き平均がゼロで分散が σ^2）を加えたものとしてモデル化しよう。ここで σ^2 は Σ/b の対角成分の役割を果たすため、σ^2 に比例するすべての量は $1/b$ としてスケールする。SGD の 1 ステップは次のようになる。

$$x_{k+1} = x_k - \eta\,(\lambda x_k + \xi_k) = \rho\, x_k - \eta\,\xi_k, \qquad \rho = 1 - \eta\lambda.$$

各ステップにおいて、決定論的な部分は厳密な勾配降下法と同様に反復点を係数 ρ で最小点へと引き寄せ、確率的な部分はランダムな揺さぶり $-\eta\xi_k$ を与える。反復点の大きさを追跡する

ため、その二次モーメント $e_k = \mathbb{E}[x_k^2]$ に着目する。更新式を二乗して期待値をとると、交差項 $-2\rho\eta\,\mathbb{E}[x_k \xi_k]$ は ξ_k が x_k を条件として平均ゼロを持つため消え、次式が残る。

$$e_{k+1} = \rho^2 e_k + \eta^2 \sigma^2 .$$

各ステップで二次モーメントは ρ^2 倍に縮小し、新たなノイズパワー $\eta^2\sigma^2$ が注入される。本節のほぼすべての結果は、このたった 1 つの漸化式から読み取ることができる。定数の駆動項を持つ線形漸化式は、不動点 V（すなわち $V = \rho^2 V + \eta^2\sigma^2$）を求めることで解ける。この不動点方程式を元の漸化式から引くと、剰余 $e_k - V$ が毎ステップ ρ^2 倍に縮小することがわかり、次式が得られる。

$$e_k = V + \rho^{2k}\,(e_0 - V) \tag{35.40}$$

ここで、**ノイズフロア**は次式で与えられる。

$$V = \frac{\eta^2\sigma^2}{1-\rho^2} = \frac{\eta\,\sigma^2}{\lambda\,(2-\eta\lambda)} \;\approx\; \frac{\eta\,\sigma^2}{2\lambda} \qquad (\eta\lambda \ll 1). \tag{35.41}$$

ノイズフロアはステップサイズおよびノイズ分散に対して線形に増大し、曲率に反比例して縮小する。いずれの式も、二次関数上の厳密な勾配降下法と同じ安定性の上限 $0 < \eta < 2/\lambda$ を必要とする。これを超えると、ノイズがなくても二次モーメントは発散する。

軌跡は 2 つの段階にきれいに分かれる。過渡項 $\rho^{2k}(e_0 - V)$ は決定論的な勾配降下法と同様に幾何級数的に減衰するのに対し、フロア V は純粋にノイズの影響によるものである。最小点から遠い領域では過渡項が支配的であり、SGD は勾配降下法とほぼ同様に振る舞う。最小点に近づくと、内側へ引き戻す力 $\eta\lambda x$ は弱まる一方でランダムな揺さぶりは同じ分散 $\eta^2\sigma^2$ を保ち続け、やがて両者は $e = V$ で釣り合う。それ以降、反復点は決して収束せず、その分布は最小点の周りの定常的な**ノイズ球**（noise ball）へと落ち着く。このノイズ球の二乗平均平方根半径はおよそ $\sqrt{\eta\sigma^2/(2\lambda)}$ であり、$\sqrt{\eta}$ に比例してスケールする。Figure 35.15 に、2 次元の二次関数ボウルにおける両段階の様子を示す。

実用上関心があるのは、二乗距離よりも期待損失である。$\mathbb{E}f(x_k) = \frac{1}{2}\lambda e_k$ であるから、損失の下限は次のようになる。

$$\tfrac{1}{2}\lambda V = \frac{\eta\,\sigma^2}{2\,(2-\eta\lambda)} \;\approx\; \frac{\eta\,\sigma^2}{4}.$$

驚くべきことに、ステップサイズが小さいとき曲率 λ は完全に相殺される。平坦な方向ほど反復点は遠くまでさまようが、その倍率と平坦さによって損失が割り引かれる倍率がちょうど一致するのである。

d 次元の二次関数上では、ヘッセ行列の固有ベクトルに沿った各座標がこのスカラー漸化式の d 個の独立なコピーへと分離し、それぞれが固有値 λ_i とノイズ分散 σ_i^2 を持つ。したがって、全体の期待損失の下限はその和 $\sum_i \eta\sigma_i^2/(2(2-\eta\lambda_i))$ となる。σ_i^2 は固有基底における Σ/b の対角成分であるため、その総和は $\operatorname{tr}\Sigma/b$ に等しい。すべての $\eta\lambda_i \ll 1$ が成り立つとき、全体の損失の下限はおよそ $\eta\operatorname{tr}\Sigma/(4b)$ へと簡約され、ステップサイズとバッチサイズにはその比 η/b を通じてのみ依存する。ゆえに、ステップサイズを半分にすることとバッチサイズを 2 倍にすることは同じ倍率だけ下限を引き下げる。これが 9.8 節の大バッチ学習率則の背後にある結びつきである。

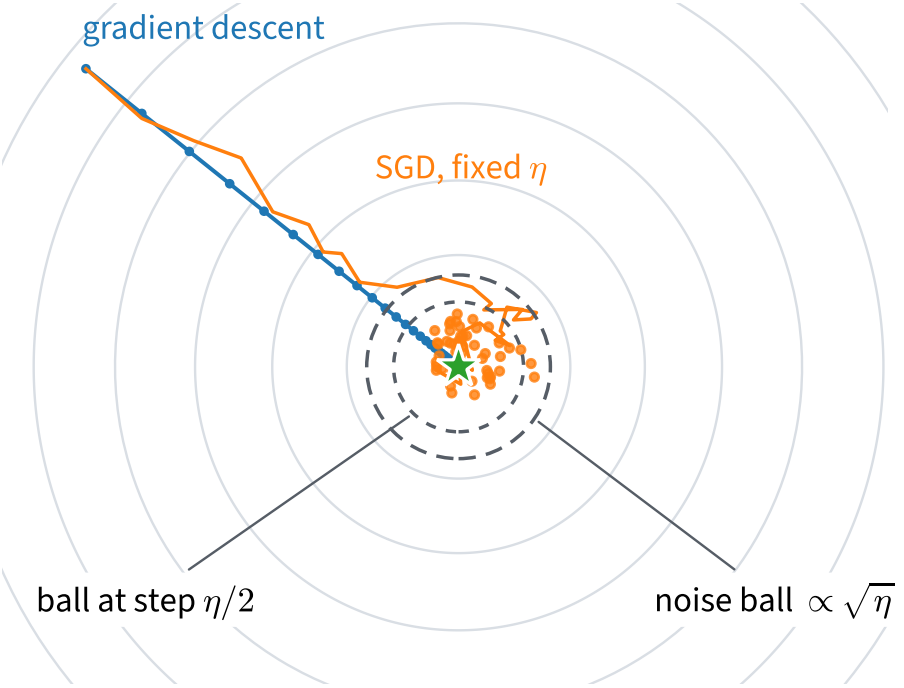


Figure 35.15: $\frac{1}{2}\|\mathbf{x}\|^2$ 上での $\eta = 0.15$、標準偏差 0.6 の勾配ノイズにおける勾配降下法と固定ステップ SGD。フルバッチの経路は最小点に収束するのに対し、SGD は勾配降下法のように降下したのちノイズ球の内部で変動し続ける。破線の円は、このステップサイズと半分のステップサイズにおけるノイズ球を表す。散らばりを囲むよう座標ごとの二乗平均平方根距離 $\sqrt{\eta\sigma^2/(2\lambda)}$ の 3 倍で描かれた両円の半径は、$\sqrt{2}$ 倍異なる。

実行はどれほど速くフロアに到達するだろうか。冒頭の例と同じく $\lambda = \sigma = 1$ としよう。そこでは $\eta = 0.1$ のとき $V = 0.053$ であった。ステップを $\eta = 0.3$ に引き上げると、フロアは $V = 0.176$（3 倍強の高さで、損失の下限は 0.088）に上昇する。$x_0 = 3$（$e_0 = 9$）から出発すると、過渡項 $e_k - V$ が V の 10 分の 1 未満に落ちるまでに 36 ステップを要する（$\eta = 0.1$、$\rho^2 = 0.81$ の場合）のに対し、わずか 9 ステップで済むのが $\eta = 0.3$（$\rho^2 = 0.49$）の場合である。ステップサイズを 3 分の 1 にするとフロアもおよそ 3 分の 1 に下がるが、そこに到達するには 4 倍のステップ数が必要になる。本節の残りで扱う手法はすべて、この速度と精度のトレードオフを乗り切るための異なるアプローチである。

ここで通し例の $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ に戻り、毎ステップ、分散 $\sigma^2 = 1$ で平均ゼロの独立なノイズを勾配 $(x_1, 1000\,x_2)$ の各座標に加えることでノイズを直接モデル化しよう。2 つの座標は $\lambda_1 = 1$ および $\lambda_2 = 1000$ のスカラー漸化式に従う。$\eta = 1/L = 10^{-3}$、すなわち安定性の上限 $2/1000$ の半分に設定する。硬い座標では $\rho_2 = 0$ となり、わずか 1 ステップで過渡項が完全に消え、フロア $V_2 = 10^{-3}/1000 = 10^{-6}$ 上に残る。遅い座標は $\rho_1 = 0.999$ とフロア $V_1 = 10^{-3}/1.999 = 5.0 \times 10^{-4}$ を持ち、その過渡項が 1,000 分の 1 に縮小するまでには $\log 1000/(-2\log 0.999) \approx 3{,}450$ ステップを要する。全体の損失の下限は 7.5×10^{-4} であり、その内訳は遅い座標からの 2.5×10^{-4} と硬い座標からの 5×10^{-4} である。皮肉な結果に注目してほしい。勾配降下法が 1 ステップで除去してしまう硬い座標が、損失の下限の 3 分の 2 を占めているのである。というのも、$\eta\lambda_2 = 1$ におけるその損失の下限は $\eta\sigma^2/2$ となり、小ステップ時の値 $\eta\sigma^2/4$ の 2 倍に達するからである。悪条件性は平坦な方向を減速させ、ノイズは両方向に沿って損失を押し上げる。

閉じた形で求めたこのフロアが厳密に成り立つのは、加法ノイズを伴う二次関数の場合のみである。次節では、「フロアが η に比例する」というスケーリングが非二次関数であるロジスティック損失でも成り立つかを検証し、ステップサイズの減衰がどのようにそのフロアを突き破るかを見ていく。

35.4.2 ステップサイズの減衰

35.4.2.1 Robbins–Monro 条件

ノイズフロアは η に対して線形にスケールするため、最小点に実際に到達するにはステップサイズをゼロに向けて縮小する必要がある。しかし、縮小が速すぎると反復点は目標の手前で失速してしまう。単位サイズの勾配のもとでは、移動できる総距離は $\sum_k \eta_k$ を超えられないからである。Robbins and Monro (1951) の古典的な条件は、これら両方の要件を同時に捉えている。

$$\sum_{k} \eta_k = \infty, \qquad \sum_{k} \eta_k^2 < \infty. \tag{35.42}$$

第 1 の条件は反復点が出発点からいくらでも遠くまで移動できることを保証し、第 2 の条件はノイズによって注入される累積分散 $\sigma^2 \sum_k \eta_k^2$ を有限に抑える。スケジュール $\eta_k = c/k$ は両方の条件を満たすが、$\eta_k = c/\sqrt{k}$ と定数ステップは第 1 の条件しか満たさない。最小点までの距離に対して分散が高々二次でしか増大しない不偏勾配と組み合わせることで、これら 2 つの条件は任意の強凸損失において SGD が最小点へ収束することを保証する (Robbins and Monro 1951; Bottou et al. 2018)。

スカラー二次関数上では、平均二乗収束に必要な条件は実際にはこれより弱い。漸化式を展開すると、$e_k \to 0$ となるには $\eta_k \to 0$ かつ $\sum_k \eta_k = \infty$（ただし $\eta_k < 2/\lambda$）であれば十分だとわかる。特に $\eta_k = c/\sqrt{k}$ とすると $e_k \approx c\sigma^2/(2\lambda\sqrt{k})$ となり、これは単に各時点のステップサイズに対応する瞬時的なフロアを追跡しているにすぎない。より強い二乗総和可能条件は、一般の

定理においてほとんどすべての個別の標本経路に沿った収束を保証するために必要となるものである。

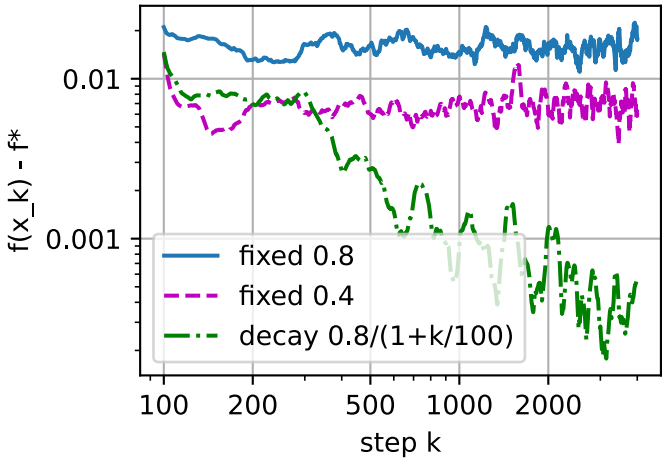
非二次目的関数については 2 つの疑問が残る。ノイズフロアは依然として η に対して線形にスケールするのか、そして減衰スケジュールはそのフロアを下回ることができるのかである。先のロジスティック回帰問題で確かめるため、まず基準となる最適値 $f^\star$ を $4{,}000$ 回の全勾配ステップで計算し、続いて毎ステップの準最適性ギャップ（suboptimality gap）$f(\mathbf{x}_k) - f^\star$ を記録するミニバッチ SGD ループを定義する。

```
x_star = np.zeros(d)
for _ in range(4000):                              # full-gradient reference optimum
    x_star -= 0.5 * example_grads(x_star, A, y, reg).mean(axis=0)
f_star = loss(x_star, A, y, reg)

def sgd(schedule, A, y, reg, f_star, rng, steps=4000, b=8):
    """Minibatch SGD from x = 0; returns the gap f(x_k) - f* after every step."""
    x, gaps = np.zeros(A.shape[1]), np.empty(steps)
    for k in range(steps):
        idx = rng.integers(0, len(y), size=b)
        x -= schedule(k) * example_grads(x, A[idx], y[idx], reg).mean(axis=0)
        gaps[k] = loss(x, A, y, reg) - f_star
    return gaps
```

二次モデルが信頼できる指針であるならば、固定ステップサイズを半分にすれば SGD が頭打ちになるギャップも半分になるはずである。ここでは $4{,}000$ ステップの訓練を $b = 8$ で実行し、定数ステップ $\eta = 0.8$ および $\eta = 0.4$ と、減衰スケジュール $\eta_k = 80/(k + 100) = 0.8/(1 + k/100)$（初期値は 0.8）を比較する。以下のセルでは、各実行の最後の $1{,}000$ ステップにわたる平均ギャップを表示してそれらの比を報告し、100 ステップ移動平均を両対数軸上にプロットする。

```
rules = {'fixed 0.8': lambda k: 0.8, 'fixed 0.4': lambda k: 0.4,
         'decay 0.8/(1+k/100)': lambda k: 0.8 / (1 + k / 100)}
gaps = {name: sgd(s, A, y, reg, f_star, rng) for name, s in rules.items()}
tail = [gap[-1000:].mean() for gap in gaps.values()]
print('mean gap, last 1000 steps: ' + ', '.join(f'{t:.1e}' for t in tail))
print(f'floor(0.8) / floor(0.4) = {tail[0] / tail[1]:.2f}, '
      f'floor(0.8) / decay = {tail[0] / tail[2]:.0f}')
window = np.ones(100) / 100                        # plot 100-step moving averages
d2l.plot(np.arange(100, 4001), [np.convolve(gap, window, 'valid') for gap in gaps.values()],
         'step k', 'f(x_k) - f*', xscale='log', yscale='log', legend=list(gaps))
```

```
mean gap, last 1000 steps: 1.6e-02, 6.9e-03, 3.8e-04
floor(0.8) / floor(0.4) = 2.37, floor(0.8) / decay = 43
```



固定ステップの両実行はいずれも頭打ちになる。$\eta = 0.8$ では、最後の 1,000 ステップの平均ギャップは $1\text{〜}2 \times 10^{-2}$ 付近に落ち着き、ステップサイズを半分にするとギャップもおよそ半分になる。繰り返し実行すると、その比は約 $1.7\text{〜}2.8$ の間で変動し、ロジスティック損失が二次関数でないにもかかわらず二次モデルが予測する 2 倍を中心として分布する。一方、減衰スケジュールはおよそ 20〜90 倍も $\eta = 0.8$ のフロアより低い値に到達し、両対数プロット上の曲線はミニバッチによる揺らぎを伴いながらもなお下降し続けている。序盤の大きなステップが最小点までの距離を素早く縮め、終盤の小さなステップがノイズ球を縮小させるのである。

35.4.2.2 $1/k$ スケジュールにおける定数

スケジュール $\eta_k = c/(k + k_0)$ は任意の $c > 0$ に対して Robbins–Monro 条件の両方を満たし、シフト k_0 によって初期ステップは安定性の上限より十分低く保たれる。では、どのような c を選んでも $O(1/k)$ の収束率が達成されるのだろうか。曲率 $\lambda = \mu$ のスカラー二次関数がその厳密な答えを与える。$t = k + k_0$ と書くと、二次モーメントの漸化式は次のようになる。

$$e_{k+1} = \left(1 - \frac{c\mu}{t}\right)^2 e_k + \frac{c^2 \sigma^2}{t^2}.$$

ここでは縮小が $1/t$ の速さで弱まる一方、注入されるノイズパワーは $1/t^2$ の速さで減少する。どちらの効果が支配的になるかは c に決定的に依存する。c が小さすぎると、反復点が最小点へ向かって縮小するよりも速くステップサイズが縮んでしまい、ノイズではなくスケジュール自体がボトルネックとなる。

命題（$1/k$ スケジュールの定数）。 **曲率 μ のスカラーモデルにおいてステップサイズを $\eta_k = c/(k + k_0)$ とするとき、$2c\mu > 1$ ならば $k\, e_k \to A$ となる。ただし、定数は次式で与えられる。**

$$A = \frac{c^2 \sigma^2}{2c\mu - 1}. \tag{35.43}$$

$2c\mu < 1$ のとき、e_k は $k^{-2c\mu}$ のオーダーで減衰し、これは $1/k$ より真に遅い。また境界 $2c\mu = 1$ では $\log k / k$ のオーダーで減衰する。任意の $c > 0$ に対して $e_k \to 0$ が成り立つ。

ここでは完全な証明は省略し、主要項の係数比較によって定数 A を導出するとともに、残る 2 つの領域の概略を直感的に説明するにとどめる。演習問題 2 では、しきい値より上では帰納法により、下では漸化式の展開によって完全な証明を与える。

定数の導出。漸近形 $e_k \approx A/t$ を仮定してみよう。漸化式の左辺は $A/(t+1) \approx A/t - A/t^2$ となる。右辺では $(1-c\mu/t)^2 \approx 1 - 2c\mu/t$ と展開することで、$A/t - 2c\mu A/t^2 + c^2\sigma^2/t^2$（$t^{-3}$ 以上の高次項を除く）が得られる。$1/t$ の項は自動的に一致し、$1/t^2$ の係数を等しいとおくと $-A = -2c\mu A + c^2\sigma^2$、すなわち $A(2c\mu - 1) = c^2\sigma^2$ が得られる。左辺の微妙な項に注目してほしい。t から $t+1$ へ進むことで生じる $-A/t^2$ は縮小項と同じオーダーであり、これがしきい値 $2c\mu > 1$ を生み出している。このシフトを無視すると、$A = c\sigma^2/(2\mu)$ がすべての $c > 0$ で成り立つという誤った予測になってしまう。

しきい値未満（直感的な説明）。$2c\mu < 1$ のとき、初期誤差と序盤のノイズの双方の減衰率は累積縮小によって決まる。$\log(1-u) \approx -u$ および $\sum_{j \le t} 1/j \approx \log t$ より、縮小率の積は $\prod_j (1 - c\mu/(j+k_0))^2 \approx (k_0/t)^{2c\mu}$ のように振る舞い、$1/t$ よりも遅く減衰する。境界 $2c\mu = 1$ では、ステップ j で注入されたノイズがステップ t においておよそ $c^2\sigma^2/(jt)$ の重みで生き残り、j について和をとることで余分な $\log k$ の因子が生じる。

この感度を具体的な数値で確認するため、$\mu = \sigma = 1$ としよう。$c = 1$ とすると $A = 1$ となり、$e_0 = 1$、$k_0 = 10$ から漸化式を反復すると $k\,e_k = 1.0001$（$k = 10^6$ 時点）が得られる。$c = 2$ に倍増させても定数の増加は緩やかで、$A = 4/3$ にとどまる。一方、しきい値を下回ると被害ははるかに大きい。$c = 0.25$ では両対数傾きが -0.50（$k = 10^5 \sim 10^6$ の区間）となり、$c = 0.1$ では傾きがわずか -0.20 になる。A を c について最小化すると最適定数 $c = 1/\mu$ が得られ、このとき $A = \sigma^2/\mu^2$、$e_k \approx \sigma^2/(\mu^2 k)$ となる。注目すべきことに、定数ステップ実行の反復点を平均化すれば、ステップサイズを一切減衰させずにこれとまったく同じ最適定数を達成できる（演習問題 5）。$c = 1/(2\mu)$ における鋭いしきい値ゆえに、$1/k$ スケジュールは実用上脆いものとなる。最小曲率 μ が既知であることはまれであり、c を過小に見積もると収束率そのものが犠牲になるからである。この脆さこそ、実用的な学習率スケジュールが $1/k$ よりも緩やかに減衰する主要な理由の 1 つである。

通し例の $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ では、シフト k_0 が第 2 の困難を生む。最初のステップにおける安定性から $c/k_0 < 2/L$、すなわち $k_0 > cL/2$ が要求される。したがって遅い座標に対する最適定数 $c = 1/\mu = 1$ を選ぶと、$k_0 > 500$ が強いられる。$k_0 = 1000$、$\mathbf{x}_0 = (1, 1)$ とすると、遅い座標の過渡項は初期誤差のうち $(k_0/(k+k_0))^2 \approx 0.008$ の割合を $k = 10^4$ ステップ後にも残しており、期待損失は 4.2×10^{-3} にとどまる。これは定数ステップの下限 7.5×10^{-4} よりも 5 倍以上悪い。硬い座標が要求する小さな初期ステップが遅い座標の前進を阻み、条件数が k_0 の内部に再び現れるのである。

35.4.2.3 厳密な漸化式のもとでのスケジュールの形状

現代の訓練で $1/k$ 減衰が使われることはまれであり、実務ではコサイン減衰やウォームアップ・安定・減衰が好まれる（ニューラルネットワーク上での比較は 9.7 節を参照）。深層ネットワーク上でこれらのスケジュールに順位をつける一般定理は存在しないものの、二次モデルを用いれば、サンプリングもランダムノイズも一切なしに**任意の**スケジュールの期待損失を厳密に計算できる。先に導出した二次モーメントの漸化式は η_k が変化してもステップごとに成り立つため、単に次式が得られる。

$$e_{k+1} = (1 - \eta_k \lambda)^2 e_k + \eta_k^2 \sigma^2, \qquad \mathbb{E} f(x_k) = \tfrac{1}{2}\lambda e_k.$$

この更新を反復すれば、K ステップの予算にわたる厳密な期待損失曲線を K 回の乗算で計算できる。ここでは同じ予算 K にわたって 4 つの形状を比較する（Figure 35.16）。すなわち、定数ステップ η_0、$1/k$ スケジュール $c/(k+k_0)$（$c = \eta_0 k_0$ として初期値を η_0 に揃える）、**コサイン減衰**（cosine decay）(Loshchilov and Hutter 2016)、そして**ウォームアップ・安定・減衰**

(warmup–stable–decay、WSD) (Hu et al. 2024) であり、WSD は η_0 を予算の最初の 80% にわたって一定に保ち、その後ゼロまで線形に減衰させる。

$$\eta_k^{\text{cos}} = \tfrac{1}{2}\eta_0\Big(1+\cos\frac{\pi k}{K}\Big),$$
$$\eta_k^{\text{WSD}} = \begin{cases} \eta_0, & k < 0.8K, \\ \eta_0\,\dfrac{K-k}{0.2K}, & k \geq 0.8K. \end{cases}$$

コサイン減衰と WSD はどちらも η_0 から始まりゼロで終わるが、コサイン減衰が実行全体を通じて連続的に減衰するのに対し、WSD は最後の 5 分の 1 まで減衰を一切遅らせる。冒頭の短いウォームアップ区間（35.5.4 節で議論する）は、二次モデルがステップ 0 からすでに η_0 で安定しているため省略する。WSD が大規模な事前学習で広く使われるのには運用上の理由がある。定数プラトー上のどのチェックポイントからでも任意の終端ステップに向けて減衰させられるため、総予算をあらかじめ確定しておく必要がないのである。この二次モデルが評価するのは最終的な期待損失であり、そうした柔軟性ではない。

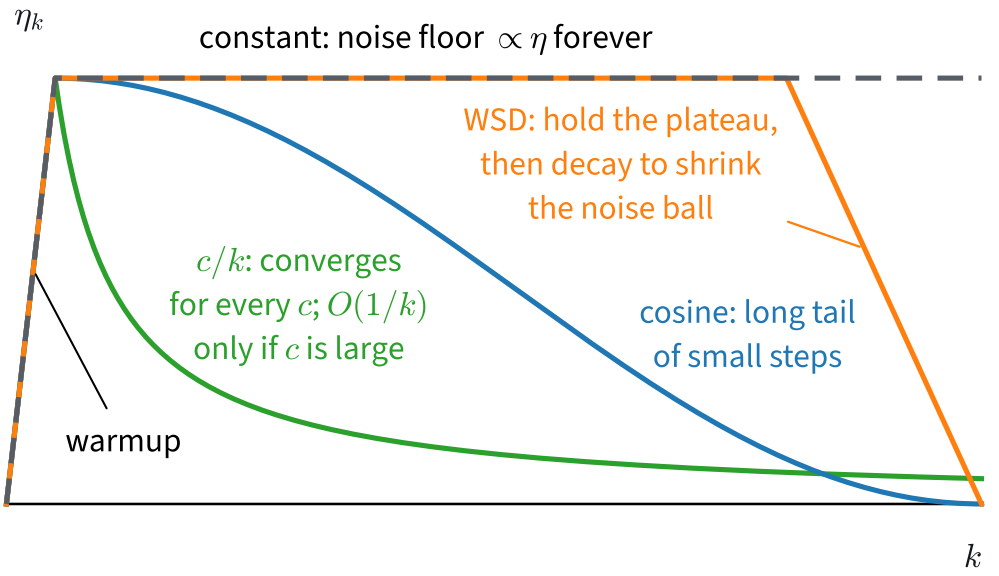
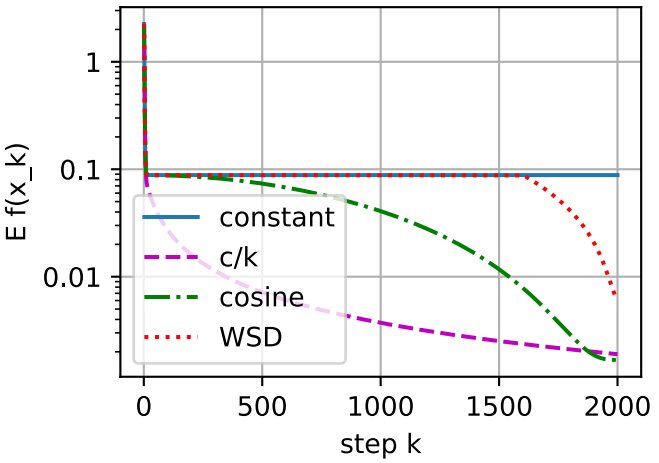


Figure 35.16: 1 つの予算 K にわたるスケジュールの形状。定数、$1/k$ 減衰（図中ラベルは c/k）、コサイン減衰、およびウォームアップ・安定・減衰（WSD）であり、いずれも以下の計算では省略する短い線形ウォームアップを冒頭に伴う。図はステップサイズの系列をプロットしたものであり、それぞれが二次モデル上で生み出す期待損失は以下で計算する。

これら 4 つのスケジュールのうち、最も低い最終損失に到達するのはどれだろうか。数値を計算する前に予想してみてほしい。以下のセルでは、$\lambda = \sigma = 1$、$K = 2000$、$\eta_0 = 0.3$、$x_0 = 3$、そして $k_0 = 50$（$1/k$ スケジュール用）として厳密な漸化式を反復し、予算の 80% 時点および終了時点での期待損失を表示したうえで $\mathbb{E}f(x_k)$ をプロットする。

```
lam, sigma, K, eta0, x0 = 1.0, 1.0, 2000, 0.3, 3.0

def expected_loss(schedule, lam, sigma, K, x0):
    """E f(x_k) = lam e_k / 2 for k = 1..K from the exact second-moment recursion."""
    e, Ef = x0**2, np.empty(K)
    for k in range(K):
        eta = schedule(k)
        e = (1 - eta * lam)**2 * e + eta**2 * sigma**2
        Ef[k] = 0.5 * lam * e
    return Ef

schedules = {
    'constant': lambda k: eta0,
    'c/k': lambda k: eta0 / (1 + k / 50),
    'cosine': lambda k: 0.5 * eta0 * (1 + np.cos(np.pi * k / K)),
    'WSD': lambda k: eta0 if k < 0.8 * K else eta0 * (K - k) / (0.2 * K)}
traces = {name: expected_loss(s, lam, sigma, K, x0) for name, s in schedules.items()}
print(f'floor {eta0 * sigma**2 / (2 * (2 - eta0 * lam)):.4f}   '
      + ''.join(f'{name:>10s}' for name in traces))
for label, k in [('E f at 80%', int(0.8 * K) - 1), ('E f at end', K - 1)]:
    print(f'{label:15s}' + ''.join(f'{Ef[k]:10.2e}' for Ef in traces.values()))
d2l.plot(np.arange(1, K + 1), list(traces.values()), 'step k', 'E f(x_k)',
         yscale='log', legend=list(traces))
```

```
floor 0.0882     constant       c/k    cosine       WSD
E f at 80%       8.82e-02  2.36e-03  7.87e-03  8.82e-02
E f at end       8.82e-02  1.90e-03  1.68e-03  6.13e-03
```



漸化式は決定論的であるため、数値は毎回完全に同一となる。定数ステップは損失の下限 0.0882 の 10 分の 1 以内にステップ 9 までに落ち着き、そこにとどまる。WSD は減衰段階が始まるまで定数ステップの実行**そのもの**であるため、予算の 80% 時点ではまったく同じフロア上に位置する。その後、最後の 400 ステップで損失を 14.4 分の 1 の 6.1×10^{-3} まで引き下げており、改善のすべてがこの区間で生じている。コサイン減衰と $1/k$ 減衰はそれぞれ 1.7×10^{-3} と 1.9×10^{-3} という近い値で終了し、WSD よりも約 3〜4 倍低い損失に達する。

$1/k$ スケジュールの終端値は先の命題とよく一致する。ここでは $c = 0.3 \cdot 50 = 15$ であるため

$A = 225/29 \approx 7.8$ となり、$\mathbb{E} f(x_K) \approx \frac{1}{2} A/(K + k_0) = 1.9 \times 10^{-3}$ が得られる。ただし、この順位はこれらの二次関数パラメータに対する厳密な結果であり、ニューラルネットワークにそのまま当てはまるとは限らないことは念頭に置いておきたい。二次関数そのものにおいては、定数プラトーを維持しても最終損失は改善しない。WSD の減衰段階を長くするほど最終損失は単調に減少し、実行全体にわたる線形減衰では 2.7×10^{-3} に達する。

35.4.3 固定予算における収束率

減衰スケジュールは、終わりを定めない実行のために設計されたものである。しかし実際には、計算予算 K があらかじめわかっていることが多く、問いは次のように書き換えられる。K ステップに対してどの定数ステップサイズを選ぶべきか、またそれによってどの程度の精度が保証されるのか。強凸性がなければ幾何級数的な縮小率も厳密なノイズフロアも存在しないため、厳密な軌跡ではなく上界を証明することになる。以下に示す 2 つの上界はいずれも同じ 2 項構造を共有している。すなわち、$1/(\eta K)$ として減少する過渡項と、η として増大するノイズ項の和である。両者を釣り合わせると $\eta \propto 1/\sqrt{K}$ が選ばれる（Figure 35.17）。

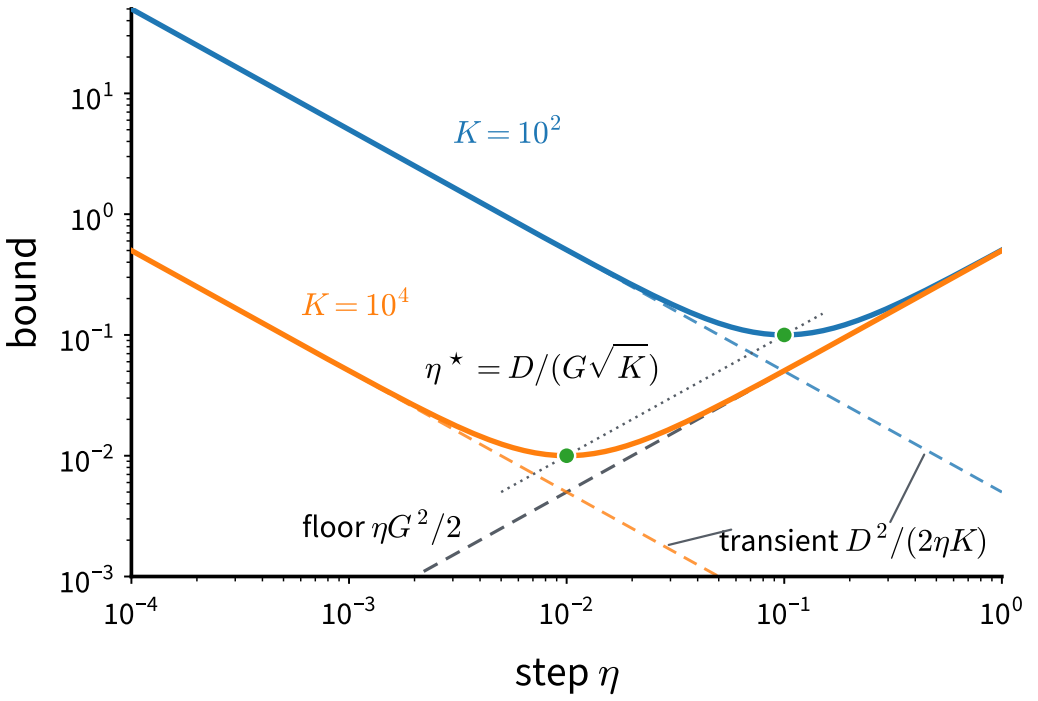


Figure 35.17: 2 項上界 $D^2/(2\eta K) + \eta G^2/2$（Equation 35.44）をステップサイズ η に対してプロットした図（$D = G = 1$、2 通りの予算）。過渡項は η とともに減少し、フロアはそれとともに上昇する。両者の和は最小値 $DG/\sqrt{K}$ に $\eta^\star = D/(G\sqrt{K})$ で達するため、予算を $100\times$ 大きくすると最適ステップと上界の双方が $10\times$ 低くなる。

35.4.3.1 凸損失における平均反復点

最初の上界に必要なのは凸性だけであり、35.3 節の劣勾配不等式を通じて適用する。すなわち、$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{s}^\top(\mathbf{y} - \mathbf{x})$ がすべての $\mathbf{y}$ およびすべての劣勾配 $\mathbf{s} \in \partial f(\mathbf{x})$ に対して成り立つ（$\nabla f(\mathbf{x})$ に帰着するのは f が微分可能な点である）。

命題（凸損失上の SGD、平均反復点）。 f を最小点 $\mathbf{x}^\star$ および最適値 $f^\star = f(\mathbf{x}^\star)$ を持つ凸関数とする。更新 $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\, \mathbf{g}_k$ を $\|\mathbf{x}_0 - \mathbf{x}^\star\| \le D$ を満たす初期点から実行し、各確率的勾配が $\mathbb{E}[\mathbf{g}_k \mid \mathbf{x}_k] \in \partial f(\mathbf{x}_k)$ および $\mathbb{E}[\|\mathbf{g}_k\|^2 \mid \mathbf{x}_k] \le G^2$ を満たすとする。このとき、平均反復点 $\bar{\mathbf{x}}_K = \frac{1}{K}\sum_{k=0}^{K-1} \mathbf{x}_k$ は次を満たす。

$$\mathbb{E}\big[f(\bar{\mathbf{x}}_K)\big] - f^\star \;\le\; \frac{D^2}{2\eta K} + \frac{\eta\, G^2}{2}, \tag{35.44}$$

さらに、$\eta = D/(G\sqrt{K})$ と選ぶと $\mathbb{E}[f(\bar{\mathbf{x}}_K)] - f^\star \le DG/\sqrt{K}$ が得られる。

証明。 最小点までの二乗距離を追跡する。1 回の更新を展開すると次式が得られる。

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|^2 = \|\mathbf{x}_k - \mathbf{x}^\star\|^2 - 2\eta\, \mathbf{g}_k^\top(\mathbf{x}_k - \mathbf{x}^\star) + \eta^2 \|\mathbf{g}_k\|^2.$$

ここで $\mathbf{x}_k$ が与えられたときの条件付き期待値をとる。$\mathbf{s}_k = \mathbb{E}[\mathbf{g}_k \mid \mathbf{x}_k]$ と書くと、交差項は $-2\eta\, \mathbf{s}_k^\top(\mathbf{x}_k - \mathbf{x}^\star)$ となる。$\mathbf{y} = \mathbf{x}^\star$ における劣勾配不等式より、$\mathbf{s}_k^\top(\mathbf{x}_k - \mathbf{x}^\star) \ge f(\mathbf{x}_k) - f^\star$ が成り立つ。議論全体はこのステップにかかっている。損失ギャップが大きいときは常に、勾配が $\mathbf{x}^\star$ からの変位と同じ向きに揃い、反復点をより近くへと引き寄せるのである。最後の項を $\eta^2 G^2$ で上から抑え、全期待値をとって $r_k = \mathbb{E}\|\mathbf{x}_k - \mathbf{x}^\star\|^2$ とおくと、次式を得る。

$$2\eta\, \mathbb{E}\big[f(\mathbf{x}_k) - f^\star\big] \;\le\; r_k - r_{k+1} + \eta^2 G^2.$$

$k = 0, \dots, K-1$ にわたって和をとると距離が望遠鏡和となって打ち消し合い、$r_0 - r_K \le r_0 \le D^2$ であるから $2\eta \sum_{k<K} \mathbb{E}[f(\mathbf{x}_k) - f^\star] \le D^2 + K\eta^2 G^2$ となる。両辺を $2\eta K$ で割り、凸関数 f にイェンセンの不等式 $f(\bar{\mathbf{x}}_K) \le \frac{1}{K}\sum_{k<K} f(\mathbf{x}_k)$ を適用すると Equation 35.44 が得られる。2 つの項は $\eta = D/(G\sqrt{K})$ で一致し、そのとき各項は $DG/(2\sqrt{K})$ に等しい。$\blacksquare$

2 つの項の意味は明快である。$D^2/(2\eta K)$ は距離 D を K ステップ（ステップサイズ η）で移動するための過渡コストであり、$\eta G^2/2$ は 1 ステップあたりに注入されるノイズフロアである。特筆すべきことに、この証明は滑らかさを一切必要としないため、ヒンジ損失のように角を持つ非滑らかな損失に対しても上界が成り立つ。予算を 100 倍にすると、最適ステップサイズと保証される誤差の双方が 10 分の 1 に減少する。この $O(1/\sqrt{K})$ という収束率は最悪ケースにおいて改良不可能である。本命題が扱う問題クラス全体では、$\mathbb{E}\|\mathbf{g}\|^2 \le G^2$ を満たす不偏な確率的勾配のみを観測するいかなるアルゴリズムも、$DG/\sqrt{K}$ の定数倍より良い保証を与えることはできない (Nemirovski and Yudin 1983)。

強凸性を加えるとスケーリングは劇的に変化する。スカラー二次関数上では、予算 K に対する最適定数ステップは $e_K = \Theta(\log K / K)$ を達成する。強凸性によって、過渡項が ρ^{2K} のように幾何級数的に縮小し、$1/(\eta K)$ のような代数的減衰ではなくなるからである。

2 つの注意点がある。第 1 に、この命題が上界を与えるのは**平均**反復点 $\bar{\mathbf{x}}_K$ に対してである。最終反復点が従う上界はこれより弱く、既知の予算に対する最終反復点の最悪ケース最適スケジュールはゼロまで線形に減衰するものとなる (Defazio et al. 2023)。第 2 に、仮定 $\mathbb{E}\|\mathbf{g}_k\|^2 \le G^2$ は $\mathbb{R}^d$ 全体上の二次関数のように勾配が非有界な損失を除外する。ただし、有界な特徴量を持つ Lipschitz 損失（正則化なしロジスティック回帰など）や、反復点が有界集合内にとどまる場合には成り立つ。また最悪ケースの上界であるため、素性の良い問題では保守的になり得る。実際、球内に制限した先のロジスティック回帰課題では、真のギャップを約 1 桁過大評価する。

35.4.3.2 非凸損失におけるランダム反復点

ニューラルネットワークでは非凸な損失とノイズを含む勾配に同時に直面するが、ここまでの結果は一度に一方の困難だけを扱ってきた。関数 f の勾配が L-Lipschitz であるとき（L 滑らかな関数）、損失曲面はいたるところで曲率 L の接放物線の下側に位置する。

$$f(\mathbf{y}) \;\le\; f(\mathbf{x}) + \nabla f(\mathbf{x})^{\top}(\mathbf{y}-\mathbf{x}) + \frac{L}{2}\|\mathbf{y}-\mathbf{x}\|^{2} \quad \text{for all } \mathbf{x},\mathbf{y}.$$

厳密な勾配と $\eta = 1/L$ のもとでは、この二次上界から凸性の仮定なしに $\min_{k<K}\|\nabla f(\mathbf{x}_k)\|^2 \le 2L\Delta/K$ が導かれる。ここで $f^\star$ は f の下界であり、$\Delta = f(\mathbf{x}_0) - f^\star$ である（35.1 節）。一方、二次関数上でノイズを含む勾配を用いると、定数ステップはノイズフロアで頭打ちになる。Ghadimi and Lan (2013) の定理はこの両方の要素を組み合わせたものである。SGD の各ステップを $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\mathbf{g}_k$ とモデル化し、確率的勾配が次を満たすとする。

$$\mathbb{E}\left[\mathbf{g}_k \mid \mathbf{x}_k\right] = \nabla f(\mathbf{x}_k), \qquad \mathbb{E}\left[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|^2 \mid \mathbf{x}_k\right] \;\le\; \sigma^2. \tag{35.45}$$

言葉で言えば、確率的勾配は不偏であり、そのノイズパワーは一様に σ^2 以下に抑えられる。Equation 35.39 より、上限が有限でありさえすれば、ミニバッチ勾配は $\sigma^2 = \sup_{\mathbf{x}} \operatorname{tr}\Sigma(\mathbf{x})/b$ として両方の仮定を満たす。以下で扱う有界な tanh モデルではこの上限は有限となるが、$\mathbb{R}^d$ 全体上の最小二乗法では最小点からの距離とともに勾配ノイズが増大するため無限大となる。

命題（SGD は近似停留点を見つける）。 **f を L 滑らかで下に $f^\star$ で有界な関数とし、$\Delta = f(\mathbf{x}_0) - f^\star$ とおく。Equation 35.45 を満たす確率的勾配と定数ステップサイズ $\eta \le 1/L$ を用いて SGD を実行するとき、次が成り立つ。**

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla f(\mathbf{x}_k)\|^2\right] \;\le\; \frac{2\Delta}{\eta K} + L\eta\sigma^2, \tag{35.46}$$

さらに、$\eta = \min\!\left(1/L,\ \sqrt{2\Delta/(L\sigma^2 K)}\right)$ と選ぶと、実行とは独立にインデックス R を $\{0,\dots,K-1\}$ から一様に抽出したとき、次が保証される。

$$\mathbb{E}\left[\|\nabla f(\mathbf{x}_R)\|^2\right] \;\le\; \frac{2L\Delta}{K} + 2\sigma\sqrt{\frac{2L\Delta}{K}}. \tag{35.47}$$

証明。 L 滑らかさによる上界を実際のステップ $\mathbf{y}-\mathbf{x} = -\eta\,\mathbf{g}_k$ に適用する。

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \eta\,\nabla f(\mathbf{x}_k)^{\top}\mathbf{g}_k + \tfrac{L\eta^2}{2}\,\|\mathbf{g}_k\|^2.$$

$\mathbf{x}_k$ を与えた条件付き期待値をとると、2 つのことが同時に達成される。不偏性により中央の項は $-\eta\|\nabla f(\mathbf{x}_k)\|^2$ となり、分散分解によって二次の項は信号とノイズの和 $\mathbb{E}[\|\mathbf{g}_k\|^2 \mid \mathbf{x}_k] = \|\nabla f(\mathbf{x}_k)\|^2 + \mathbb{E}[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|^2 \mid \mathbf{x}_k] \le \|\nabla f(\mathbf{x}_k)\|^2 + \sigma^2$ に分解される。項をまとめると次式が得られる。

$$\mathbb{E}\left[f(\mathbf{x}_{k+1}) \mid \mathbf{x}_k\right] \;\le\; f(\mathbf{x}_k) - \eta\left(1 - \tfrac{L\eta}{2}\right)\|\nabla f(\mathbf{x}_k)\|^2 + \tfrac{L\eta^2\sigma^2}{2}.$$

これは標準的な降下補題（descent lemma）に、1 ステップあたり $\frac{1}{2}L\eta^2\sigma^2$ の分散ペナルティが加わったものである。$\eta \le 1/L$ のときは常に $1 - \frac{L\eta}{2} \ge \frac{1}{2}$ が成り立つ。全期待値をとって $k = 0, \dots, K-1$ にわたる和をとると、損失は望遠鏡和により $\mathbb{E}[f(\mathbf{x}_K)] - f(\mathbf{x}_0) \ge -\Delta$ となり、両辺を $\eta K/2$ で割れば Equation 35.46 が得られる。その左辺は、$\mathbb{E}[\|\nabla f(\mathbf{x}_R)\|^2]$（一様ランダムなインデックス R に対する期待値）にほかならない。最後に、釣り合いをとったステップサイズには 2 つの場合がある。$\sqrt{2\Delta/(L\sigma^2 K)} \le 1/L$ ならば Equation 35.46 の 2 項は等しくなり、それぞれ $\sigma\sqrt{2L\Delta/K}$ となる。そうでない場合は $\eta = 1/L$ かつ $\sigma^2 < 2L\Delta/K$ であるため、上界は $2L\Delta/K + \sigma^2 \le 2L\Delta/K + \sigma\sqrt{2L\Delta/K}$ となる。いずれの場合も Equation 35.47 で上から抑えられる。■

$\sigma = 0$ のときは第 2 項が消え、決定論的な収束率 $2L\Delta/K$ が復元される。ノイズが存在し、かつ予算が十分大きくて $K > 2L\Delta/\sigma^2$（したがって $\eta < 1/L$）となる領域では、ノイズ項が支配的となり $K^{-1/2}$ の速さでしか減衰しない。たとえば $L = \Delta = \sigma = 1$、$K = 10^4$ の場合、釣り合いのステップは $\eta \approx 0.014$ となり、上界は $2 \times 10^{-4} + 0.028$ となる。つまりノイズ項は決定論的な項の 140 倍も大きい。また、この上界が抑えているのは勾配ノルムの**二乗**である点にも注意が必要である。予算を 100 倍にすると $\mathbb{E}\|\nabla f\|^2$ は 10 分の 1 に減少するが、これは勾配ノルムそのものを 10 分の 1 に縮小するには 10^4 倍ものステップが必要になることを意味する。勾配推定量に統計的バイアスがある場合、証明の中で余分な内積項が生き残り、K を増やしても決して消えない定数のフロアが加わる（演習問題 4）。

なぜこの定理は最良の反復点ではなく**ランダム**な反復点 $\mathbf{x}_R$ を返すのだろうか。厳密な勾配があれば、毎ステップ $\|\nabla f(\mathbf{x}_k)\|$ を観測できるため、勾配が最小の反復点を単に選べばよい。しかし確率的勾配では、真の勾配ノルムはアルゴリズムからは見えない。そこでインデックス R を一様ランダムに選ぶという実装可能な規則を採用すると、Equation 35.46 の時間平均がまさにその期待二乗勾配ノルムに一致するのである。

凸の場合と比べると、2 つの点が変わっている。損失ギャップではなく二乗勾配ノルムを抑えていること、そして平均反復点ではなくランダム反復点で評価していることである。凸性がなければ停留点が大域的最小点であるとは限らないため、1 次法が保証できるのは勾配が小さいことだけである。凸の場合の上界と同様、この $K^{-1/2}$ という収束率は、Equation 35.45 を満たす確率的勾配のみにアクセスできるあらゆる手法にとって最悪ケースで最適である (Arjevani et al. 2023)。

35.4.3.3 単一の tanh ニューロンにおける収束率

この命題は上界であるため、実験によって収束率がタイトであることを証明することはできず、実際の実行が $K^{-1/2}$ という上限に対してどう振る舞うかを確かめられるのみである。以下では、ランダム反復点の指標と最良の反復点の双方を小規模な非凸問題上で測定する。具体的には、単一の tanh ニューロンを最小二乗法によって 5 次元のノイズを含む $n = 64$ 個の例に当てはめる。

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\big(\tanh(\mathbf{a}_i^\top \mathbf{x}) - y_i\big)^2.$$

各成分は滑らかで有界であるが、引数が大きい領域で tanh が平坦なプラトーへと飽和するため、いずれも凸ではない。以下のセルではデータを生成し、パラメータベクトルのバッチ（X の各行に 1 つ）に対して厳密な全勾配を計算するベクトル化された補助関数を定義する。

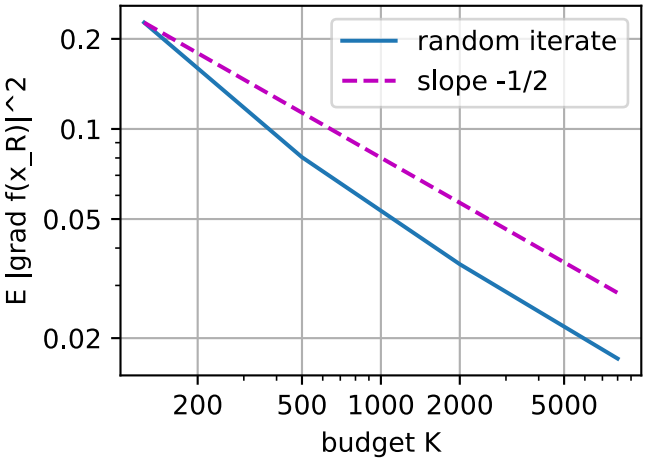
```
n_nc, d_nc = 64, 5                              # a small nonconvex problem
A_nc = rng.normal(size=(n_nc, d_nc))
y_nc = np.tanh(A_nc @ rng.normal(size=d_nc)) + 0.5 * rng.normal(size=n_nc)
```

```
def full_grad(X, A, y):
    """Exact gradient of mean_i (tanh(a_i^T x) - y_i)^2 at each row x of X."""
    t = np.tanh(X @ A.T)                          # (runs, n)
    return 2 * ((t - y) * (1 - t**2)) @ A / len(y)
```

予算 $K \in \{125, 500, 2000, 8000\}$ に対し、定理のスケーリング $\eta = 0.5/\sqrt{K}$ を用いて単一例の SGD を実行する。このとき 20 回の独立な実行を `X` の各行として並列に進める。各予算について、以下のセルは実行全体にわたる $\|\nabla f(\mathbf{x}_k)\|^2$ の平均（ランダム反復点の指標）とその最小値（最良の反復点）の双方を記録し、K に対する両対数傾きを表示したうえで、ランダム反復点の曲線を基準傾き $-1/2$ と並べてプロットする。

```
runs, Ks, avg, best = 20, np.array([125, 500, 2000, 8000]), [], []
for K in Ks:
    eta = 0.5 / np.sqrt(K)                        # the theorem's eta ~ 1/sqrt(K)
    X = np.zeros((runs, d_nc))                    # one row per independent run
    total, low = np.zeros(runs), np.full(runs, np.inf)
    for k in range(K):
        g2 = (full_grad(X, A_nc, y_nc)**2).sum(axis=1)   # |grad f(x_k)|^2
        total, low = total + g2, np.minimum(low, g2)
        i = rng.integers(0, n_nc, size=runs)             # one example per run
        t = np.tanh((X * A_nc[i]).sum(axis=1))
        X -= eta * (2 * (t - y_nc[i]) * (1 - t**2))[:, None] * A_nc[i]
    avg.append(total.mean() / K), best.append(low.mean())

def slope(v):
    return np.polyfit(np.log(Ks), np.log(v), 1)[0]

for name, v in [('random iterate', avg), ('best iterate', best)]:
    print(f'{name:15s}' + ' '.join(f'{u:.1e}' for u in v) + f'   slope {slope(v):.2f}')
d2l.plot(Ks, [avg, avg[0] * (Ks / Ks[0])**-0.5], 'budget K', 'E |grad f(x_R)|^2',
         xscale='log', yscale='log', legend=['random iterate', 'slope -1/2'])
```

```
random iterate 2.3e-01 8.0e-02 3.5e-02 1.7e-02   slope -0.62
best iterate   1.3e-02 2.9e-03 7.0e-04 1.6e-04   slope -1.06
```



予算を 64 倍に増やすと、ランダム反復点の指標は約 12 分の 1（実行ごとに概ね 9〜16 分の 1）に低下し、両対数傾きは -0.6 付近となる。予想どおり、定理の上界である $-1/2$ よりわずかに急である。最良の反復点はそれよりはるかに速く低下し、傾きは -1 前後かそれ以上に急になるが、前述のとおりノイズを含む勾配しか観測できないアルゴリズムにはそれを特定する術がない。

35.4.4 分散低減

35.4.4.1 制御変量

ステップサイズを減衰させれば η を縮めることでノイズフロアを解消できるが、前進が代数的な収束率に鈍化する代償を伴う。目的関数が n 個の例にわたる**有限**和である場合には、第 2 の選択肢がある。勾配ノイズそのものを除去するのである。最小点 $\mathbf{x}^\star$ では全勾配はゼロになるが、個々の例の勾配 $\nabla f_i(\mathbf{x}^\star)$ は一般にゼロにはならない。そして、これらの非ゼロの成分勾配こそが、定数ステップサイズが注入し続けるノイズの正体である。

対策が一切不要となる重要な領域も 1 つある。モデルが訓練セットを補間する（すべての例に損失ゼロで適合する）場合、最小点においてすべての $\nabla f_i(\mathbf{x}^\star)$ がゼロになり、強凸目的関数上の定数ステップ SGD はノイズフロアなしで線形収束する（演習問題 6）。大規模なニューラルネットワークは訓練セットを補間することが多く、このことが加法ノイズのフロアから予測されるよりも定数ステップ SGD がはるかに良好に機能する一因を説明している。

成分勾配が $\mathbf{x}^\star$ でゼロにならない場合には、その分散を直接打ち消さなければならない。これを行うのがモンテカルロ法の標準的な技法である。ノイズを含む推定量 $\nabla f_i(\mathbf{x})$ から、厳密な平均が既知である相関した確率変数を引き、その平均を足し戻すのである。このような補正項は**制御変量**（control variate）と呼ばれ、方策勾配のベースラインでも再び登場する（14.8 節）。有限和において自然な制御変量は、同じ例の勾配 $\nabla f_i(\tilde{\mathbf{x}})$（保存しておいたスナップショット $\tilde{\mathbf{x}}$ で評価したもの）であり、すべての i にわたるその平均はデータセット全体を 1 パス走査するごとに 1 回計算される全勾配 $\nabla f(\tilde{\mathbf{x}})$ である。これにより、**SVRG**（stochastic variance-reduced gradient、確率的分散低減勾配）推定量が得られる (Johnson and Zhang 2013)。

$$\tilde{\mathbf{g}}_i(\mathbf{x}) = \nabla f_i(\mathbf{x}) - \nabla f_i(\tilde{\mathbf{x}}) + \nabla f(\tilde{\mathbf{x}}). \tag{35.48}$$

言葉で言えば、SVRG はスナップショットにおける厳密な全勾配から出発し、$\tilde{\mathbf{x}}$ と $\mathbf{x}$ の間で単一の例の勾配がどれだけ変化したかによってそれを補正する。一様に抽出されたインデックス i に対して、この推定量は次のように不偏性を保つ。

$$\mathbb{E}_i[\tilde{\mathbf{g}}_i(\mathbf{x})] = \nabla f(\mathbf{x}) - \nabla f(\tilde{\mathbf{x}}) + \nabla f(\tilde{\mathbf{x}}) = \nabla f(\mathbf{x}),$$

一方で、反復点とスナップショットが収束するにつれて、その分散はゼロへと縮小する。

命題（SVRG の分散はスナップショットまでの距離で抑えられる）。各成分 f_i が L_i-Lipschitz 勾配を持ち、$L_{\max} = \max_i L_i$ とするとき、一様に抽出されたインデックス i に対して次が成り立つ。

$$\mathbb{E}_i\big\|\tilde{\mathbf{g}}_i(\mathbf{x}) - \nabla f(\mathbf{x})\big\|^2 \;\le\; \mathbb{E}_i\big\|\nabla f_i(\mathbf{x}) - \nabla f_i(\tilde{\mathbf{x}})\big\|^2 \;\le\; L_{\max}^2\,\|\mathbf{x} - \tilde{\mathbf{x}}\|^2. \tag{35.49}$$

証明。左辺は確率ベクトル $\nabla f_i(\mathbf{x}) - \nabla f_i(\tilde{\mathbf{x}})$ の分散であり、その平均は $\nabla f(\mathbf{x}) - \nabla f(\tilde{\mathbf{x}})$ である。任意の確率ベクトルの分散は高々その二次モーメント以下であるため、第 1 の不等式は直ちに従う。二次モーメントの各項に L_i-Lipschitz 上界を適用すれば、第 2 の不等式が得られる。$\blacksquare$

ノイズパワーは $\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$ の定数倍で上から抑えられるため、$\mathbf{x}$ と $\tilde{\mathbf{x}}$ の双方が $\mathbf{x}^\star$ に近づくにつれて自動的にゼロとなり、定数ステップサイズであってもノイズフロアが取り除かれる。直感的には、近接する 2 点で同じ例を評価するとほぼ同一のノイズを共有する勾配が生じるため、差をとることで共有ノイズが相殺され、勾配の小さな真の変化だけが残るのである。

35.4.4.2 SVRG と SAGA

上界 Equation 35.49 を縮小させ続けるには、スナップショットが定期的に反復点に追いつかなければならないが、毎ステップ全勾配を再計算しては確率的最適化の意味が失われてしまう。SVRG はエポック単位で動作することでこれを解決する。各エポックの冒頭で現在の反復点をスナップショット $\tilde{\mathbf{x}}$ として保存し、その全勾配 $\nabla f(\tilde{\mathbf{x}})$ を計算する。続いて、新たなランダムインデックスを用いて m 回の内側更新 $\mathbf{x} \leftarrow \mathbf{x} - \eta\,\tilde{\mathbf{g}}_i(\mathbf{x})$ を実行し、最後の内側反復点を次のスナップショットに設定する。したがって 1 エポックのコストは、例の勾配評価 $n + 2m$ 回分となる。Johnson and Zhang (2013)（定理 1）は、すべての f_i が $L_{\max}$-Lipschitz 勾配を持つ凸関数であり、f が μ 強凸で、$\eta < 1/(4L_{\max})$ かつ 1 エポックあたりの縮小率が 1 未満になるほど m が十分大きければ、期待準最適性 $\mathbb{E}[f(\tilde{\mathbf{x}})] - f^\star$ がエポック数に対して幾何級数的に減衰することを証明している（定理ではランダムな内側反復点を次のスナップショットに選ぶが、以下の実装では実用上同様に振る舞う最終反復点を用いる）。

リッジ回帰上で SVRG を検証しよう。これは $f_i(\mathbf{x}) = \tfrac{1}{2}(\mathbf{a}_i^\top \mathbf{x} - y_i)^2 + \tfrac{0.1}{2}\|\mathbf{x}\|^2$、例数 $n = 300$、パラメータ数 $d = 12$ からなる強凸な有限和である。厳密な最小点は正規方程式を解くことで求まり、各成分の勾配は L_i-Lipschitz（$L_i = \|\mathbf{a}_i\|^2 + 0.1$）である。以下のセルではデータを生成して成分勾配と全勾配を定義し、$L_{\max}$ と理論上のステップサイズ上限 $1/(4L_{\max})$ を表示する。

```
n_r, d_r, reg_r = 300, 12, 0.1                  # ridge regression, a finite sum
A_r = rng.standard_normal((n_r, d_r))
y_r = A_r @ rng.standard_normal(d_r) + 0.5 * rng.standard_normal(n_r)
H_r = A_r.T @ A_r / n_r + reg_r * np.eye(d_r)     # Hessian of f
x_star_r = np.linalg.solve(H_r, A_r.T @ y_r / n_r)

def component_grad(x, i, A, y, reg):
    """Gradient of f_i(x) = (a_i^T x - y_i)^2 / 2 + reg |x|^2 / 2 (i: index or array)."""
    return A[i] * (A[i] @ x - y[i])[..., None] + reg * x

def full_grad_sum(x, A, y, reg):
    """Gradient of the finite sum f = mean_i f_i."""
```

```
    return A.T @ (A @ x - y) / len(y) + reg * x

def objective(x, A, y, reg):
    """The ridge objective f(x)."""
    return 0.5 * np.mean((A @ x - y)**2) + 0.5 * reg * x @ x

L_max = (A_r**2).sum(axis=1).max() + reg_r       # largest per-example smoothness
print(f'L_max = {L_max:.0f}, 1 / (4 L_max) = {1 / (4 * L_max):.4f}')
```

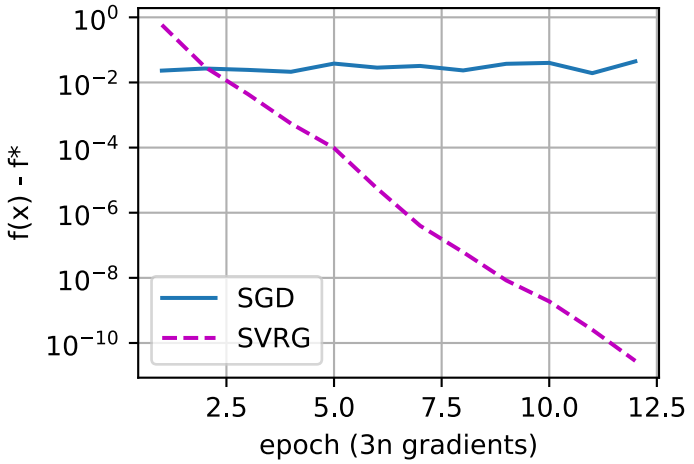
```
L_max = 34, 1 / (4 L_max) = 0.0074
```

データのランダムな生成に対して $L_{\max}$ は概ね 25〜40 の範囲に入り、定理のステップサイズ上限 $1/(4L_{\max})$ はおよそ 0.006〜0.01 の間となる。

次に、SVRG（$m = n$）と単一例の SGD を、同じ定数ステップサイズ $\eta = 0.02$（定理の上限の 2 倍超）のもとで 12 エポック（各エポックあたり例の勾配評価 $3n$ 回のコスト）にわたり比較する。あわせて、先に導出した二次関数のフロア公式も SGD 上で検証する。定常状態における損失ギャップは、ヘッセ行列の固有ベクトルにわたる $\eta\sigma_i^2/(2(2-\eta\lambda_i))$ の和に等しくなるはずである。ここで σ_i^2 は、$\mathbf{x}^\star$ における例の勾配の固有ベクトル i に沿った分散である。

```
eta, epochs = 0.02, 12
f_star_r = objective(x_star_r, A_r, y_r, reg_r)
snap, gaps_svrg = np.zeros(d_r), []
for _ in range(epochs):                          # n + 2n example gradients per epoch
    g_snap = full_grad_sum(snap, A_r, y_r, reg_r)
    x = snap.copy()
    for i in rng.integers(0, n_r, size=n_r):
        x -= eta * (component_grad(x, i, A_r, y_r, reg_r)
                    - component_grad(snap, i, A_r, y_r, reg_r) + g_snap)
    snap = x
    gaps_svrg.append(objective(snap, A_r, y_r, reg_r) - f_star_r)
x, gaps_sgd = np.zeros(d_r), []
for _ in range(epochs):
    for i in rng.integers(0, n_r, size=3 * n_r):  # the same 3n budget per epoch
        x -= eta * component_grad(x, i, A_r, y_r, reg_r)
    gaps_sgd.append(objective(x, A_r, y_r, reg_r) - f_star_r)
G_r = component_grad(x_star_r, np.arange(n_r), A_r, y_r, reg_r)  # noise at x*
ev, Q = np.linalg.eigh(H_r)                      # Hessian eigenvalues, eigenvectors
s2 = ((G_r @ Q)**2).mean(axis=0)                 # noise variance per eigenvector
floor = (eta * s2 / (2 * (2 - eta * ev))).sum()  # the loss floors add
print(f'SGD: predicted floor {floor:.1e}, mean gap in epochs 4-12 '
      f'{np.mean(gaps_sgd[3:]):.1e}')
print('SVRG: gap after epochs 1, 4, 12: '
      + ', '.join(f'{gaps_svrg[e - 1]:.0e}' for e in (1, 4, 12)))
d2l.plot(np.arange(1, epochs + 1), [gaps_sgd, gaps_svrg], 'epoch (3n gradients)',
         'f(x) - f*', yscale='log', legend=['SGD', 'SVRG'])
```

```
SGD: predicted floor 2.2e-02, mean gap in epochs 4-12 3.2e-02
SVRG: gap after epochs 1, 4, 12: 6e-01, 5e-04, 3e-11
```



SGD は序盤で急速に低下したのち、10^{-2} の数倍程度で頭打ちとなり、ちょうどフロア公式が予測する位置に落ち着く。繰り返し実行すると、実測されるプラトーは予測値の約 10% 下から 40% 上の間に収まる。下回るよりわずかに上回ることが多いのは、公式が $\mathbf{x}^\star$ でのみ勾配の分散を評価しているのに対し、最小点から離れると例の勾配がわずかに広く散らばるためである。SVRG は出だしこそ遅れるものの、最終的には大きく引き離す。最初の 1 エポック後（出発点に固定されたスナップショットのもとで n 回の内側ステップしか進まない）では、ギャップはまだ SGD を上回っている。しかし第 2 エポック以降は 1 エポックあたり約 8 分の 1（実行により 5〜13 分の 1）のペースで縮小し、10^{-13}〜10^{-8} に第 12 エポックまでに到達する。特筆すべきことに、SVRG は定理の上限の 2 倍を超えるステップサイズでこの線形収束を達成しており、理論上のステップ条件が十分条件であって必要条件ではないことを示している。この優位性を支える 3 つの仮定は念頭に置いておきたい。すなわち、有限和であること、データセットを複数パス走査できること、そして線形収束率のための強凸性である。各例を 1 度しか観測しないストリーミングデータでは、スナップショットをとるべき有限和が存在しない。

SAGA (Defazio et al. 2014) は、スナップショットを各例について直近に評価された勾配を保持するテーブル $\mathbf{t}_1, \ldots, \mathbf{t}_n$ に置き換えることで、定期的な全データ走査を完全に不要にする。各ステップでは推定量 $\nabla f_i(\mathbf{x}) - \mathbf{t}_i + \frac{1}{n}\sum_j \mathbf{t}_j$ を構成したのち、$\mathbf{t}_i$ を $\nabla f_i(\mathbf{x})$ で上書きする。SAGA は 1 ステップあたり新たな例の勾配を 1 つ計算するだけで済むが、代償として $n \times d$ 個の数値を保存する必要がある（ただし一般化線形モデルでは 1 例あたり 1 個のスカラーで足りる。演習問題 3 を参照）。

残念ながら深層ニューラルネットワークにおいては、SVRG も SAGA も一貫した改善をもたらしていない。損失ランドスケープが急速に変化したり、データ拡張（data augmentation）によってパスごとに各例が摂動を受けたりすると、制御変量がすぐに陳腐化してしまうからである。他方で、悪条件な谷の例以来つきまとってきた第 2 の制約も残されている。本節のすべての手法は全座標に単一の大域的なステップサイズを適用するため、通し例における最も硬い方向によって η が $2/1000$ 以下に制限されてしまうのである。次節では、これまでに観測した勾配から各座標が独自のステップサイズを適応的に調整できるようにすることで、このボトルネックを取り除く。

35.4.5 まとめ

サイズ b のミニバッチ勾配は ∇f の不偏推定量であり、そのノイズパワーは $\operatorname{tr}\Sigma/b$ である。二次関数上では、固定ステップサイズ η によって各座標の二次モーメントが縮小率 $(1-\eta\lambda)^2$ で幾何級数的に減衰してノイズフロア $\eta\sigma^2/(\lambda(2-\eta\lambda))$ へと収束し、座標ごとの損失の下限

$\eta\sigma^2/(2(2-\eta\lambda))$ が次元全体にわたって足し合わされる。フロアは η に対して線形に増大する一方、過渡項は η が大きいほど速く縮小するため、いかなる定数ステップも最小点には到達できず、本節の各手法はいずれもこのトレードオフに対処している。Robbins–Monro スケジュールはフロアを解消するが、$\eta_k = c/k$ が $O(1/k)$ を達成するのは $c > 1/(2\mu)$ の場合に限られる。K ステップの固定予算においては、定数ステップ $\eta \propto 1/\sqrt{K}$ を選ぶことで、凸損失上の平均反復点に対して $DG/\sqrt{K}$ が、滑らかな非凸損失上のランダム反復点における期待二乗勾配に対して $O(\sigma\sqrt{L\Delta/K})$ が保証される。最後に有限和においては、SVRG の制御変量が不偏性を保ちつつ分散を高々 $L_{\max}^2\|\mathbf{x}-\tilde{\mathbf{x}}\|^2$ に抑えるため、強凸問題上で固定ステップサイズのまま線形収束することが可能となる。

35.4.6 演習問題

1. [code] **ノイズスケールと臨界バッチサイズ。**●1 回のミニバッチステップは損失をどれだけ引き下げ、その改善は b にどう依存するだろうか。$\mathbf{g} = \nabla f(\mathbf{x})$、$\mathbf{H} = \nabla^2 f(\mathbf{x})$ と書き、ステップ $\mathbf{x} - \eta\,\hat{\mathbf{g}}_b$（Equation 35.38）の後の損失を $\mathbf{x}$ の周りでの 2 次テイラー展開によって近似せよ。以下の解析は McCandlish et al. (2018) に従う。
   1. 共分散 Σ/b（$\hat{\mathbf{g}}_b$ の共分散、Equation 35.39 の証明で導出）を用いて、ステップ後の期待損失が $\eta_{\text{opt}}(b) = \eta_{\max}/(1 + B_{\text{noise}}/b)$ で最小化され、そのときの期待改善量が $\Delta_{\max}/(1 + B_{\text{noise}}/b)$ となることを示せ。ただし、各量は次式で定義される。
   $$\eta_{\max} = \frac{\|\mathbf{g}\|^2}{\mathbf{g}^\top \mathbf{H}\mathbf{g}}, \qquad \Delta_{\max} = \frac{\|\mathbf{g}\|^4}{2\,\mathbf{g}^\top \mathbf{H}\mathbf{g}}, \qquad B_{\text{noise}} = \frac{\operatorname{tr}(\mathbf{H}\Sigma)}{\mathbf{g}^\top \mathbf{H}\mathbf{g}}.$$
   2. Goyal et al. (2017) の線形スケーリング則は、η に b と同じ倍率を掛ける。どのようなバッチサイズの領域でこの規則が $\eta_{\text{opt}}(b)$ から導かれるか。また、$b \gg B_{\text{noise}}$ となった領域では、バッチを大きくすることで 1 ステップあたり何が得られるか。
   3. $\mathbf{H}$ が単位行列の定数倍であるとき、B_{noise} は $B_{\text{simple}} = \operatorname{tr}\Sigma/\|\mathbf{g}\|^2$、すなわち本文の勾配ノイズスケール b_{noise} に帰着する。本節のロジスティック回帰問題について、B_{simple} と B_{noise} を $\mathbf{x} = \mathbf{0}$ ならびに 5、10、20 回の全勾配ステップ（$\eta = 0.5$）の後に計算せよ。計算には `example_grads` とヘッセ行列 $\mathbf{H}(\mathbf{x}) = \frac{1}{n}\sum_i s_i(1-s_i)\,\mathbf{a}_i\mathbf{a}_i^\top + 0.1\,\mathbf{I}$（ただし $s_i = 1/(1+e^{y_i \mathbf{a}_i^\top \mathbf{x}})$）を用いよ。反復点が最適点に近づくにつれてノイズスケールはどう変化するか。B_{noise} を (b) の 2 つの領域を分ける臨界バッチサイズとみなしたとき、訓練の実行を通じてバッチサイズをどのように変化させるべきか。
2. [code] $1/k$ **スケジュールにおける定数。**●●本文では定数 A（Equation 35.43）を主要項の比較によって求めた。本演習ではこれを厳密に証明し、通し例に適用する。
   1. $t = k + k_0$ とし、$u_{k+1} \le (1 - a/t)\,u_k + q/t^2$ がすべての $k \ge 0$ で成り立つとする（ただし $a > 1$、$q > 0$、$u_k \ge 0$、$k_0 \ge a$）。帰納法により、$u_k \le \nu/(k + k_0)$（ただし $\nu = \max\{q/(a-1),\ k_0 u_0\}$）を証明せよ。（ヒント：$1 - a/t \ge 0$ かつ $1/t - 1/t^2 \le 1/(t+1)$ である。）
   2. この補題を漸化式 $e_{k+1} = (1 - c\mu/t)^2 e_k + c^2\sigma^2/t^2$ に適用せよ。$(1 - c\mu/t)^2 \le 1 - (2c\mu - (c\mu)^2/k_0)/t$ が $t \ge k_0$ で成り立ち、したがって任意の $2c\mu > 1$ に対して $e_k = O(1/k)$ となる（k_0 が十分大きい場合）ことを示せ。また a と q を特定し、$q/(a-1) \to A$（$k_0 \to \infty$ のとき）を示せ。
   3. $2c\mu < 1$ のとき、$e_k = \Theta(k^{-2c\mu})$ となることを、積 $\prod_{j<k}(1 - c\mu/(j+k_0))^2$ および展開したノイズ和から示せ。
   4. $\mu = \sigma = 1$、$k_0 = 10$、$e_0 = 1$ として漸化式を $k = 10^6$ まで $c \in \{0.1, 0.25, 0.5, 1, 2\}$ の各値について反復せよ。$k = 10^5 \sim 10^6$ の区間における両対数傾きと、$k\,e_k$ の 10^6 時点での値を報告せよ。A の最小点の 2 倍の定数と半分の定数では、どちらの代償が大きいか。

5. 座標ごとに単位ノイズを伴う通し例 $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ では、最初のステップにおける安定性から $c/k_0 < 2/1000$ が強いられる。$c = 1/\mu = 1$、$k_0 = 1000$、$\mathbf{x}_0 = (1, 1)$、$K = 10^4$ とせよ。2 座標の漸化式によって期待損失を計算し（約 4.2×10^{-3} となる）、定数ステップ $\eta = 10^{-3}$（7.5×10^{-4}）および同じ η_0 から出発するコサイン減衰（1.0×10^{-4}）と比較せよ。$1/k$ 減衰はどちらの座標の前進を阻むか。また、なぜ条件数が k_0 に再び現れるのか。

3. [code] **SVRG の内側ループと SAGA**。本節のリッジ回帰データと関数 `component_grad`、`full_grad_sum`、`objective` を用い、例の勾配評価 $36n = 10{,}800$ 回（SVRG のセルで報告した 12 エポック分の予算）のもとで実験せよ。各実行は、1 エポックごとに確認してギャップを 10^{-10} 未満に下げるまでに要した評価回数で測ること。
   1. 内側ループを m ステップ実行し（1 エポックのコストは $n + 2m$ 回の評価となる）、$m \in \{10, 30, 75, 300, 1200\}$ を $\eta \in \{0.01, 0.02\}$ のそれぞれで変化させよ。各ステップサイズにおいて最良の m はどれか。スナップショット勾配のコストを償却することと制御変量が陳腐化することのトレードオフ、および最良の m が η に応じて変化する理由を説明せよ。
   2. SAGA を実装せよ。すなわち、すべての例の直近の勾配を保持するテーブルを用意し（$\mathbf{x}_0$ において n 回の評価で初期化する）、推定量として $\nabla f_i(\mathbf{x}) - \mathbf{t}_i + \frac{1}{n}\sum_j \mathbf{t}_j$ を用いよ（$\mathbf{t}_i$ は例 i に対するテーブルの項目）。両方のステップサイズで SAGA を最良の SVRG 実行と比較せよ。この問題でテーブルは何個の数値を保持するか。また、各例につきスカラー $\mathbf{a}_i^\top \mathbf{x} - y_i$ のみを保存する場合には何個で足りるか。

4. **バイアスのある勾配推定量**。●Equation 35.45 の分散上界を（ここでは条件付き平均の周りで測るものとして）保ちつつ、不偏性を $\mathbb{E}[\mathbf{g}_k \mid \mathbf{x}_k] = \nabla f(\mathbf{x}_k) + \delta_k$（ただし $\|\delta_k\| \le \delta$）に置き換える。
   1. Equation 35.46 の証明を $\eta \le 1/L$ のもとでやり直し、次式を示せ。

$$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}\left[\|\nabla f(\mathbf{x}_k)\|^2\right] \;\le\; \frac{2\Delta}{\eta K} + L\eta\sigma^2 + \delta^2. \tag{35.50}$$

      展開する前に、$L\eta \le 1$ を項 $\frac{L\eta^2}{2}\|\nabla f(\mathbf{x}_k) + \delta_k\|^2$ に適用せよ。
   2. 項 δ^2 が除去できないことを示せ。すなわち、1 次元の f と定数の統計的バイアスを与え、$\sigma = 0$ の SGD が $|f'|^2 = \delta^2$ となる点に収束する例を構成せよ。
   3. 非復元で一様に抽出された単一のミニバッチは、依然として全勾配の不偏推定量である。では、以下の推定量のうちこの意味で統計的バイアスを持つものはどれか。エポックごとに毎回新たなランダム順ですべての例を 1 度ずつ訪問するランダムな再シャッフル（9.2 節）の各ステップにおける例の勾配、固定された最大ノルムにクリッピングされた確率的勾配、非同期訓練のように τ ステップ前の反復点で評価された勾配。

5. [code] **反復点平均化**。●Polyak and Juditsky (1992) は、緩やかに減衰するステップまたは定数ステップの実行における反復点を平均化すると、最良の定数が達成されることを示した。定数ステップ $\eta \in (0, 2/\lambda)$ を持ち、定常分布から出発するスカラーモデルを考える。
   1. $\mathrm{Cov}(x_j, x_k) = V\rho^{|j-k|}$ を示せ。
   2. 平均 $\bar{x}_K = \frac{1}{K}\sum_{k<K} x_k$ が $\mathbb{E}[\bar{x}_K^2] = \frac{V}{K}\cdot\frac{1+\rho}{1-\rho} - O(K^{-2})$ を満たし、かつ $V(1+\rho)/(1-\rho) = \sigma^2/\lambda^2$ がそのようなすべての η に対して成り立つこと、すなわち主要項において平均反復点はステップサイズに依存しないことを示せ。
   3. $K = 2000$、実行回数 4,000 回として、$\eta = 0.3$ および $\eta = 0.1$（$\lambda = \sigma = 1$）でシミュレーションにより確認せよ。いずれの場合も $\mathbb{E}[\bar{x}_K^2] \approx 5 \times 10^{-4}$ となるはずである（最終反復点のフロアはそれぞれ 0.18 と 0.053 である）。
   4. 代わりに $x_0 = 3$ から出発すると、平均は過渡項の影響を記憶する。その $\mathbb{E}[\bar{x}_K^2]$ への寄与がおよそ $(x_0/(\eta\lambda K))^2$ であることを示し、どのようなときに実行の後半のみを

平均化するほうが良いか判定せよ。

5. 最良の $1/k$ スケジュール（定数は $A = \sigma^2/\mu^2$）と比較せよ。収束率 $\sigma^2/(\mu^2 K)$ に至るこれら 2 つの経路は、それぞれ何を必要とするか。

6. [code] **補間と線形収束。●●** 各 f_i が L_i-Lipschitz 勾配を持つ凸関数であり、f が μ 強凸で、モデルがすべての訓練例に厳密に適合する場合のようにすべての f_i が同一の点 $\mathbf{x}^\star$ で最小化されると仮定する。以下の解析は Vaswani et al. (2019) に従う。
   1. $\mathbb{E}_i \|\nabla f_i(\mathbf{x})\|^2 \le 2L_{\max}(f(\mathbf{x}) - f^\star)$ を示せ。（ヒント：凸かつ L 滑らかな関数 h が $\mathbf{x}^\star$ で最小化されるとき、$\|\nabla h(\mathbf{x})\|^2 \le 2L(h(\mathbf{x}) - h(\mathbf{x}^\star))$ が成り立つ。）
   2. 定数 $\eta \le 1/L_{\max}$ を用いる単一例の SGD が $\mathbb{E}\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|^2 \le (1 - \eta\mu)\,\mathbb{E}\|\mathbf{x}_k - \mathbf{x}^\star\|^2$ を満たすこと、すなわちフロアなしで線形収束することを導け。このとき $\operatorname{tr}\Sigma(\mathbf{x}^\star)$ はいくらになるか。
   3. 例数 $n = 50$、パラメータ数 $d = 100$、$\eta = 1/L_{\max}$、$\mathbf{x}_0 = \mathbf{0}$ の最小二乗法でこれを確認せよ。反復点は $\mathbf{A}$ の行空間内にとどまり、そこでは f が強凸であるため (b) が適用できる。整合するラベルの場合にも、標準偏差 0.5 のノイズで汚染されたラベルの場合にもギャップが線形に減少することを示し、なぜここではノイズが影響しないのかを説明せよ。
   4. $n = 200$、$d = 100$ として同様に実験せよ。今度はノイズ付きラベルによってフロアが復活する。それはなぜか。

35.5 適応的手法

本書で扱う Transformer は、大半の大規模ニューラルネットワークと同様に、通常の確率的勾配降下法（stochastic gradient descent、SGD）ではなく Adam またはその変種である AdamW（9.5.3 節）で訓練される。その主な理由はステップサイズ（step size）にある。35.4 節の各手法は、すべての座標に単一のステップサイズを適用し、それを時間方向にのみ調整する。しかし現代のネットワークには 10^7〜10^{11} 個ものパラメータがあり、その曲率（curvature）は何桁も異なるうえ、事前にはわからない。すべての座標が 1 つのステップサイズを共有すると、最も急な方向がステップの上限を抑えてしまい、平坦な方向はほとんど動かなくなる。本節では、オプティマイザが勾配のみから座標ごとに個別のステップサイズをどのように選べるのか、そしてその戦略がどこで破綻するのかを問う。

本章を通じた例題を見れば、この違いがいかに大きいかがわかる。細長い谷 $f(\mathbf{x}) = \tfrac{1}{2}(x_1^2 + 1000\,x_2^2)$ において $(1, 1)$ から出発する場合を考えよう。最適な単一のステップサイズ $\eta = 2/1001$ を用いる勾配降下法（gradient descent）では、f を 10^{-8} 未満にするのに 6,160 ステップを要する。x_2 を安定に保てるほど小さなステップは、x_1 にとっては 1,000 倍も小さすぎるからである。新しいステップに直前のステップの定数倍を加える 35.1 節のヘビーボール法（heavy-ball method）では、これが 296 ステップまで短縮される（これらの回数が同節で挙げた $6{,}908$ ステップおよび 315 ステップより少し少ないのは、同節が最小点までの距離で停止したのに対し、本節では f の値で停止するためである）。これに対し、観測した勾配のみから推定した個別のステップサイズを各座標に与えるオプティマイザは、わずか数十ステップしか必要としない。さらに驚くべきは、その回数が何に依存しないかという点である。谷をさらに 100 倍細くしても、反復回数はたった 6 ステップしか増えない。

このような目を引く数値は作り出すのも容易なら読み誤るのも容易であるため、本節の多くの部分は比較を公平にすることに充てられる。ヘビーボール法は理論上の最適設定で実行し、Adam には単純なステップサイズのスイープを与え、さらに谷を 45° 回転させた場合も調べる。このより公平な検証のもとでは、Adam の優位性は最初の数値が示唆するものとはかなり異なって見えてくる。

根底にある仕組みは単純であり、AdaGrad、RMSProp、Adam がするように、各座標のステップ

を過去の勾配の二乗平均平方根（root mean square、RMS）で割ることである。本節ではまず、この正規化によって Adam が各座標のスケールに対して不感になることを証明し、続いてその対称性が破れる境界、すなわち谷の回転、凸な損失の周期的な系列、重み減衰（weight decay）、訓練序盤のステップ、および層の行列構造へと議論を進める。特に重要な結果は、スケール不変性（scale invariance）の命題と AdamW の不動点（fixed point）解析の 2 つである。AdaGrad のリグレット上界と末尾の構造化前処理行列（structured preconditioner）については、初読時には飛ばしてもよい。本節を通して、t はオプティマイザのステップ番号、h_i は座標 i の曲率、λ は重み減衰係数を表す。

35.5.1 座標ごとのステップサイズ

35.5.1.1 対角二次関数

二次関数に対する勾配降下法は、条件数（condition number）$\kappa = h_{\max}/h_{\min}$ に比例するステップ数を要する。単一のステップサイズを $2/h_{\max}$ 未満に保たなければならないからである（35.1 節）。その解決策が見えてくる最も単純な設定は、次の 2 変数の対角二次関数である。

$$f(\mathbf{x}) = \tfrac{1}{2}(h_1 x_1^2 + h_2 x_2^2), \qquad \frac{\partial f}{\partial x_i} = h_i x_i.$$

各偏微分がそれぞれの座標のみに依存するため、2 つの座標でステップサイズを共有すべき必然性はない。各座標に個別のステップサイズ η_i を与えれば、x_i は $x_i - \eta_i h_i x_i = (1 - \eta_i h_i)\, x_i$ へと更新され、$\eta_i = 1/h_i$ と選べばすべての座標が 1 ステップでゼロに到達する。$h = (1, 1000)$ の場合、理想的なステップサイズは 1 と 10^{-3} である。これは対角なヘッセ行列に対するニュートン法（Newton's method）にほかならず、反復回数から条件数が消え去る。

この理想と実用的なオプティマイザの間には 2 つの障害が立ちはだかる。曲率 h_i が未知であり訓練の過程で変化すること、そして観測される勾配がノイズを含むミニバッチ推定値であることである。したがって、勾配の大きさを代理指標として用い、勾配のみから各座標のステップサイズを推定しなければならない。対角二次関数では、勾配の成分 $g_i = h_i x_i$ の大きさは $h_i |x_i|$ となる。各座標の大きさが同程度である限り、勾配が大きい座標は曲率も高いため、勾配の大きさで割ることは実質的に曲率で割ることに相当する。ただし座標間でスケールが大きく異なる場合でも、$g_i / |g_i|$ の大きさは依然として 1 であるため、正規化は 1 ステップあたりの移動距離を η に固定する働きをし、真のニュートンステップ（Newton step）である x_i をとるわけではない。

35.5.1.2 AdaGrad

曲率の違いは、各座標に固有のステップサイズを与える理由の 1 つである。これとは独立した 2 つ目の理由として、座標ごとに更新頻度が大きく異なることが挙げられる。オンライン最適化や非滑らかな凸最適化（convex optimization）における古典的な選択である $\eta/\sqrt{t}$ のような大域的な減衰ステップサイズは、すべての座標を同一に扱う。Bag-of-Words モデルにおける希少語の重みのように、座標 1 は毎ステップ ± 1 の勾配を受け取る一方、座標 2 は 10^4 ステップに 1 回しか勾配を受け取らないとしよう。$t = 10^4$ の時点では、大域スケジュールは両方のステップサイズを $\eta/100$ まで減衰させてしまうため、希少な座標に関して得られた唯一の証拠によっても、その座標は η の 100 分の 1 しか動かない。これに対し、各座標が自身の二乗勾配を個別に累積すれば、累積和はそれぞれ 10^4 と 1 になり、η をそれぞれの平方根で割ることで、頻繁に更新される座標には $\eta/100$ を与えつつ、希少な座標のステップサイズを η のまま保てる。

これが **AdaGrad** である (Duchi et al. 2011)。勾配の各成分を二乗したベクトルを $\mathbf{g}_t \odot \mathbf{g}_t$ と書き（以降は $\mathbf{g}_t^2$ と略記する）、ベクトルに対する平方根や除算も同様に要素ごとに適用するものとする。AdaGrad は二乗勾配の累積和 $\mathbf{s}_t$ を保持し、ステップをその平方根で割る。

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathbf{g}_t \odot \mathbf{g}_t, \qquad \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{s}_t} + \epsilon} \mathbf{g}_t, \tag{35.51}$$

初期値は $\mathbf{s}_0 = \mathbf{0}$ とし、小さな定数 ϵ でゼロ除算を防ぐ（本編の 9.5 節では ϵ を平方根の内側に置いているが、ここでの定性的な議論に違いはない）。これにより、各座標のステップサイズは全体のステップ数ではなく、その座標自身の勾配履歴に応じて減衰するようになる。

計量の観点。最急降下はステップ長をどう測るかに依存する。重み付きノルム $\|\mathbf{d}\|_{\mathbf{P}} = \sqrt{\mathbf{d}^\top \mathbf{P}\, \mathbf{d}}$（$\mathbf{P}$ は正定値行列）のもとでの最急降下方向は $-\mathbf{P}^{-1}\nabla f$ である（演習問題 1、35.1 節）。AdaGrad は対角計量 $\mathbf{P}_t = \operatorname{diag}(\sqrt{\mathbf{s}_t})$ における最急降下であり、座標 i に沿う移動には、その勾配が大きかったことの累積的な証拠 $\sqrt{\sum_{\tau \le t} g_{\tau,i}^2}$ に比例するペナルティが課される。ステップを進める前に勾配を再スケーリングする $\mathbf{P}_t$ のような行列を**前処理行列**（preconditioner）と呼ぶ。ニュートン法が真の曲率 $\operatorname{diag}(h_i)$ を用いるのに対し、AdaGrad は勾配の履歴でこれを代用する（演習問題 1）。

リグレットの観点。AdaGrad はもともと**オンライン凸最適化**（online convex optimization）のために設計された手法である。この枠組みでは、凸な損失 $f_1, f_2, \dots$ が 1 つずつ順番に到着し、学習者は $\mathbf{x}_t$ を f_t を観測する前に選ばなければならない。T ラウンドにわたる性能は**リグレット** $\sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}} \sum_{t=1}^{T} f_t(\mathbf{x})$、すなわち事後的に選んだ最良の固定点に対する累積過剰損失によって測られる。この枠組みにおいて、アルゴリズムの平均リグレット（リグレットを T で割った値）がゼロに収束するとき、そのアルゴリズムは学習できているといえる。

ユークリッド直径 D_2 の凸集合上で勾配ノルムが $\|\mathbf{g}_t\| \le G$ と有界であるとき、適切に調整した大域ステップサイズをもつオンライン勾配降下法はオーダー $D_2 G \sqrt{T}$ のリグレットを達成する（演習問題 4 で証明する）。これに対し、AdaGrad は次のオーダーのリグレットを達成する (Duchi et al. 2011)。

$$D_\infty \sum_{i=1}^{d} \sqrt{\sum_{t=1}^{T} g_{t,i}^2},$$

ここで D_∞ は ℓ_∞ ノルムにおける集合の直径である。d 個の各座標は、全座標にわたる最悪ケースの勾配ノルムではなく、自身の勾配履歴に応じた分だけを負担すればよい。

ボックス $[-1, 1]^d$ 上では $D_2 = \sqrt{d}\, D_\infty$ が成り立つ。勾配の大きさが少数の座標群に集中するとき、AdaGrad の上界は最大で $\sqrt{d}$ 分の 1 まで小さくなり、すべての座標に均等に分散するときは 2 つの上界は一致する。座標 i に確率 p_i で非ゼロ成分が生じるワンホット勾配では、2 つの上界の期待値の比は高々 $\sum_i \sqrt{p_i}/\sqrt{d}$ である（演習問題 2）。p_i が一様な場合、個々の勾配がいかに疎であってもこの比は 1.00 となるのに対し、指数 1、$d = 10^4$ の Zipf 則に従う単語頻度では 0.63 まで下がる。したがって AdaGrad の優位性は、単なるスパース性（sparsity）そのものではなく、特定の座標への勾配の**集中**から生じる。

Equation 35.51 に現れる平方根は恣意的なものではなく、リグレットの証明において競合する 2 つの項をちょうど釣り合わせる厳密な指数である（演習問題 4c）。

しかしオンライン凸最適化を離れると、上限のない累積が AdaGrad の最大の弱点となる。分散 σ^2 の持続的な勾配ノイズ（gradient noise）のもとでは、累積和が $s_{t,i} \approx \sigma^2 t$ と増大するため、実効ステップサイズは $\eta/(\sigma\sqrt{t})$ のように縮小していく。つまり AdaGrad は、最適化の進捗ではなくステップ数に縛られた減衰スケジュールを内部に抱えている。この $1/\sqrt{t}$ という減衰率は凸リグレットに対しては最適だが、長時間の非凸な訓練では損失がまだ改善中であるかどうかにか

かわらずステップサイズを容赦なく減衰させ、オプティマイザが終盤のプラトーに達する頃にはステップサイズがほとんど残らなくなってしまう。

35.5.1.3 RMSProp

RMSProp (Tieleman and Hinton 2012) は、全期間の累積和を指数移動平均（exponential moving average）に置き換えることでこの減衰を防ぎ、スケール推定が直近の勾配に追随し続けるようにする。

$$\mathbf{v}_t = \beta_2\,\mathbf{v}_{t-1} + (1-\beta_2)\,\mathbf{g}_t^2, \qquad \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{v}_t} + \epsilon}\,\mathbf{g}_t. \tag{35.52}$$

この漸化式を展開すると、$\mathbf{v}_t$ は k ステップ前の二乗勾配に重み $(1-\beta_2)\beta_2^k$ を割り当てることがわかる。したがって古い勾配の重みは約 $\log 2/(1-\beta_2)$ ステップごとに半減し、RMSProp で慣例的に使われる $\beta_2 = 0.9$ では 7 ステップごと、$\beta_2 = 0.999$ では 693 ステップごとに半分になる。こうして構造的にステップサイズが消失することはなくなる。その代償として、単発の大きな勾配も同じ時間スケールで忘れ去られてしまい、この脆弱性は後述する反例で突くことになる。

35.5.1.4 Adam とバイアス補正

Adam (Kingma and Ba 2015) は、RMSProp の二次モーメント推定（second-moment estimate）に勾配そのものの指数移動平均を組み合わせ、さらに両方の移動平均に対してゼロ初期化に伴う偏りを補正する。9.5 節の記法に従えば、更新式は次のようになる。

$$\begin{aligned} \mathbf{m}_t &= \beta_1\,\mathbf{m}_{t-1} + (1-\beta_1)\,\mathbf{g}_t, & \hat{\mathbf{m}}_t &= \mathbf{m}_t/(1-\beta_1^t), \\ \mathbf{v}_t &= \beta_2\,\mathbf{v}_{t-1} + (1-\beta_2)\,\mathbf{g}_t^2, & \hat{\mathbf{v}}_t &= \mathbf{v}_t/(1-\beta_2^t), \\ \mathbf{x}_{t+1} &= \mathbf{x}_t - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}, \end{aligned} \tag{35.53}$$

ここで初期値は $\mathbf{m}_0 = \mathbf{v}_0 = \mathbf{0}$ であり、標準のデフォルト値は $\beta_1 = 0.9$、$\beta_2 = 0.999$、$\epsilon = 10^{-8}$ である。一次モーメント（first moment）$\mathbf{m}_t$ は 35.1.3 節のヘビーボールの速度を $1-\beta_1$ 倍したものであり、これによって累積和が重み付き平均へと変わり、通常の RMSProp がそのままステップに反映してしまうミニバッチノイズ（minibatch noise）を平滑化する。

$1-\beta_2^t$ で割る操作は、$\mathbf{v}_0 = \mathbf{0}$ から出発することを補償する。漸化式を展開すると $\mathbf{v}_t = (1-\beta_2)\sum_{\tau \le t} \beta_2^{t-\tau}\,\mathbf{g}_\tau^2$ となる。二乗勾配のスケールが定常であり、$\mathbb{E}[\mathbf{g}_\tau^2] = \overline{\mathbf{g}^2}$ がすべての τ について成り立つならば、等比級数の和をとることで次式が得られる。

$$\mathbb{E}[\mathbf{v}_t] = (1-\beta_2)\,\overline{\mathbf{g}^2}\sum_{\tau=1}^{t}\beta_2^{t-\tau} = (1-\beta_2^t)\,\overline{\mathbf{g}^2}. \tag{35.54}$$

したがって定常性のもとでは、$\hat{\mathbf{v}}_t$ が $\overline{\mathbf{g}^2}$ の不偏推定量であることがすべてのステップ t で成り立ち、β_1 についても同様の計算から $\hat{\mathbf{m}}_t$ が平均勾配の不偏推定量であることがわかる。

この補正がない場合、ゼロ初期化による不足分は大きく、しかも長く残る。デフォルトの $\beta_2 = 0.999$ では、$\mathbf{v}_{10}$ は定常時の目標値の約 1% しか捉えられず、$t = 1000$ に達してもまだ 63% にすぎない（9.5 節）。さらに注目すべきは、その誤差が時間とともにどう推移するかである。両方のバイアス補正（bias correction）を省くと、更新量 $\mathbf{m}_t/\sqrt{\mathbf{v}_t}$ のスケールは係数

$(1-\beta_1^t)/\sqrt{1-\beta_2^t}$ だけずれ、最初は 3.16（$t=1$）から始まり、一見意外なことにピークの 6.6（$t=12$）まで上昇する。一次モーメントの移動平均は約 10 ステップで立ち上がるのに対し、二次モーメント（second moment）の移動平均は約 1,000 ステップを要するためである。Figure 35.18 に、この不足分とそれによって生じるステップの膨張の両方を示す。

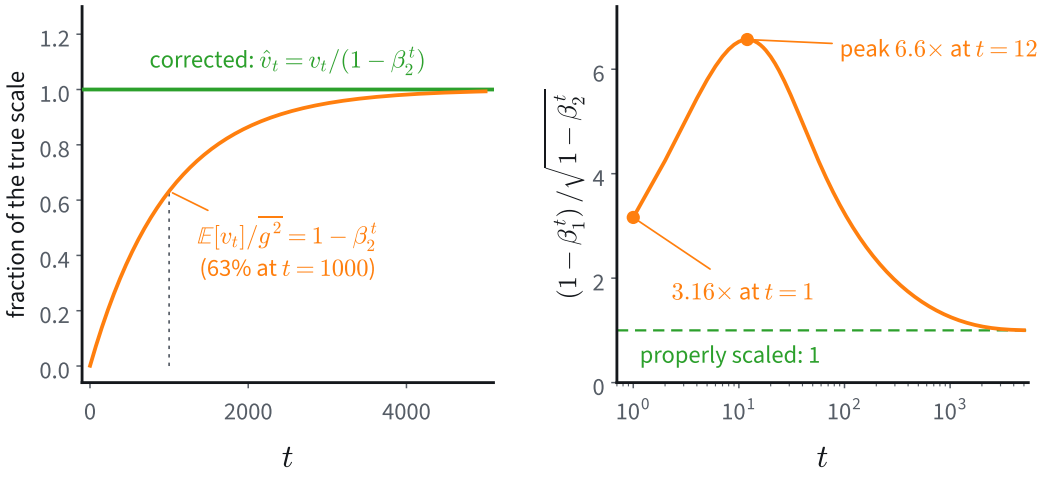


Figure 35.18: Adam におけるバイアス補正。左：ゼロから出発すると、移動平均 $\mathbf{v}_t$ は二乗勾配スケールのうち割合 $1-\beta_2^t$ しか捉えられない（63% に達するのに $t=1000$ を要する、$\beta_2=0.999$ の場合）が、$1-\beta_2^t$ で割ればすべてのステップで本来のスケールが復元される。右：バイアス補正を行わないと、更新量 $\mathbf{m}_t/\sqrt{\mathbf{v}_t}$ は $(1-\beta_1^t)/\sqrt{1-\beta_2^t}$ 倍に膨張し、3.16（$t=1$）から始まり、分子が 10 ステップで立ち上がる一方、分母は 1,000 ステップを要するため 6.6（$t=12$）でピークに達し、1 に戻るのは $1/(1-\beta_2)$ の時間スケールになってからである。したがって補正なしの Adam は、前処理行列が最も少ない勾配しか反映していないときに最大のステップを踏んでしまう。

$\hat{\mathbf{v}}_t$ が不偏であっても、ステップサイズ自体が不偏になるわけではない。Adam は $\sqrt{\hat{\mathbf{v}}_t}$ で割り、$\hat{\mathbf{v}}_t$ では割らないため、平方根の凹性により分母に下向きの統計的バイアスが生じる。すなわちイェンセンの不等式（Jensen's inequality）より、$\mathbb{E}\sqrt{\hat{v}_t} < \sqrt{\mathbb{E}[\hat{v}_t]}$ が（$\hat{v}_t$ が定数でない限り）成り立つ。最初のステップはその極端な例である。$t=1$ では $\hat{m}_1 = g_1$ かつ $\sqrt{\hat{v}_1} = |g_1|$ となるため、比は $\operatorname{sign}(g_1)$ となり、ステップの大きさは勾配がいかなる値であってもすべての座標で η となる（$\eta\,|g_1|/\sigma$ のように勾配のスケール σ で正規化された値にはならない）。これがどの程度影響するかは 35.5.4 節で検証する。

35.5.1.5 スケール不変性

冒頭の例題では $h_2 = 1000\,h_1$ であったが、Adam はまるでその 1,000 倍の比が存在しないかのように谷を横断する。その理由は更新式がもつ厳密な対称性にあり、これこそが本節の中心的な結果である。

命題（Adam は各座標のスケールを無視する）。 $\epsilon = 0$ **とし、** $\mathbf{C} = \operatorname{diag}(c_1, \dots, c_d)$ **（すべての** $c_i > 0$**）とする。2 つの問題の勾配がすべての点において** $\nabla \tilde{f}(\mathbf{x}) = \mathbf{C}\,\nabla f(\mathbf{x})$ **（任意の** $\mathbf{x}$**）という関係にあるとする。このとき、同一の** β_1**、**β_2**、ステップスケジュール、および初期点をもつ Adam は、更新が定義され続ける限り、両方の問題上で同一の反復点列を生成する。**

言い換えれば、2 つ目の問題は 1 つ目の問題において座標 i の勾配を正の定数 c_i 倍に再スケーリ

ングしたものである。その問題に関する量にはチルダを付けて表す。

証明。 t に関する帰納法で示す。反復点が $\mathbf{x}_t$ まで一致していると仮定する。すると $\tilde{\mathbf{g}}_\tau = \mathbf{C}\mathbf{g}_\tau$ がすべての $\tau \le t$ に対して成り立つ。$\mathbf{m}$ と $\mathbf{v}$ の漸化式はそれぞれ $\mathbf{g}_\tau$ と $\mathbf{g}_\tau^2$ に関して線形であるため、$\tilde{m}_{t,i} = c_i m_{t,i}$ および $\tilde{v}_{t,i} = c_i^2 v_{t,i}$ が得られる。バイアス補正の分母は両方の問題で同じスカラーであるため、2 つ目の問題における補正後の比 $\hat{m}_{t,i}/\sqrt{\hat{v}_{t,i}}$ は $c_i \hat{m}_{t,i}/(c_i\sqrt{\hat{v}_{t,i}}) = \hat{m}_{t,i}/\sqrt{\hat{v}_{t,i}}$ となる。したがって更新量は一致し、$\tilde{\mathbf{x}}_{t+1} = \mathbf{x}_{t+1}$ となる。$\blacksquare$

対角二次関数では、$\nabla f(\mathbf{x}) = \operatorname{diag}(h)\,\mathbf{x}$ は丸いボウル $\tfrac{1}{2}\|\mathbf{x}\|^2$ の勾配に $\mathbf{C} = \operatorname{diag}(h)$ を掛けたものにほかならない。その結果、1000:1 の谷における Adam は、1 ステップごとに丸いボウル上の Adam とまったく同じ軌跡（trajectory）をたどり、しかも丸いボウル上では等しい値から出発した 2 つの座標は永遠に等しいままである。各座標は $\tfrac{1}{2}x^2$ 上の同じ 1 次元の反復に従う。

$$x_{t+1} = x_t\Bigl(1 - \frac{\eta}{\sqrt{\hat{v}_t}}\Bigr) \qquad (\beta_1 = 0),$$

ここで $\hat{v}_t$ は過去の二乗 x_τ^2（$\tau \le t$）の移動平均である。初期の反復点のほうが大きいため、2 ステップ目以降は $\sqrt{\hat{v}_t}$ が $|x_t|$ を上回り、各更新の移動量は η 未満となって縮小率 $1 - \eta/\sqrt{\hat{v}_t}$ で幾何級数的に縮小する。$\eta = 0.1$ のとき、この縮小率は 0.9（$t = 1$）から 0.67（$t = 48$）へと、$\hat{v}_t$ が縮小していく反復点に追いつくにつれて強まっていく。反復点そのものは h_2/h_1 にまったく依存しない。曲率比に依存するのは停止判定だけであり、それは $f(\mathbf{x}_t) = \tfrac{1}{2}(h_1 + h_2)\,x_t^2$ が共通の反復点を各曲率で重み付けするためである。したがって、谷をはるかに細くしても幾何級数的な縮小ステップが数回増えるだけで済み、κ 倍に膨らむことは決してない。

一方でステップ数が**実際に**依存するのは、η を単位として測った初期距離である。曲率にかかわらず、序盤のステップは座標あたりおよそ η の長さをもつからである。次項の実験では、これら両方の予測を直接検証する。

要するに、Adam はすべての座標をその座標自身の勾配の大きさを単位として測るため、座標ごとのスケーリングは Adam からは見えなくなる。ただし、この不変性には 3 つの留保条件がある。第 1 に、$\epsilon > 0$ の場合、ある座標の勾配が ϵ のオーダーまで縮小すると対称性が破れる。第 2 に、この命題は座標を混ぜ合わせる基底変換については何も述べておらず、これについては次項で回転した谷を用いて検証する。第 3 に、Adam は一般には符号降下法（sign descent）ではない。更新が**符号ステップ**（sign step）$\eta\,\operatorname{sign}(\mathbf{g}_t)$ に帰着するのは、$t = 1$ のとき、$\beta_1 = \beta_2 = 0$ のとき（演習問題 5d）、および $\beta_1 = 0$ で各座標の勾配の大きさが $\hat{\mathbf{v}}$ の記憶期間にわたって一定に保たれるときの 3 つの特別な場合に限られる。

35.5.2 座標スケーリングとその限界

35.5.2.1 軸に沿った谷と回転した谷

勾配降下法が要した 6,160 ステップのうち、座標スケーリング単独でどれだけ削減できるのか、また谷を回転させたときにもその利得は保たれるのだろうか。座標ごとのスケーリングとモメンタムを切り分けるため、4 つの手法を比較する。最適な単一ステップサイズを用いた勾配降下法、最適なステップサイズとモメンタムを用いたヘビーボール法、再スケーリングのみを行う $\beta_1 = 0$ の Adam、そしてデフォルトのモメンタム $\beta_1 = 0.9$ をもつ Adam である。問題を角度 θ だけ回転させるには、$\operatorname{diag}(h)$ を $\mathbf{A}_\theta = \mathbf{R}_\theta \operatorname{diag}(h)\,\mathbf{R}_\theta^\top$（$\mathbf{R}_\theta$ は 2×2 回転行列）に置き換え、初期点もそれに合わせて $\mathbf{x}_0 = \mathbf{R}_\theta(1, 1)^\top$ と回転させることで、初期点が両方の曲率軸に沿って同じ成分を保つようにする。$\theta = \pi/4$ では $\mathbf{A} = \left(\begin{smallmatrix} 500.5 & -499.5 \\ -499.5 & 500.5 \end{smallmatrix}\right)$ となる。以下のコードでは、谷の目的関数、目標許容誤差に達するまでのステップ数を数えるループ、および 3 種類のオプティマイザの更新式を定義する。

```
h = np.array([1.0, 1000.0])                     # curvatures h_i, kappa = 1000

def rotated(theta):
    """A = R diag(h) R^T and the start R (1, 1): the valley turned by theta."""
    c, s = np.cos(theta), np.sin(theta)
    R = np.array([[c, -s], [s, c]])
    return R @ np.diag(h) @ R.T, R @ np.ones(2)

def run(step, A, x0, K=40000, tol=1e-8):
    """First t with f(x_t) = x_t^T A x_t / 2 < tol, or None within K steps."""
    x, state = x0.copy(), {}
    for t in range(1, K + 1):
        x = step(x, A @ x, t, state)
        if 0.5 * x @ A @ x < tol:
            return t
    return None

def gd(eta):
    return lambda x, g, t, state: x - eta * g

def heavy_ball(eta, beta):
    def step(x, g, t, state):
        state['d'] = beta * state.get('d', 0.0) - eta * g    # velocity
        return x + state['d']
    return step

def adam(eta, beta1=0.9, beta2=0.999, eps=1e-8):
    def step(x, g, t, state):
        state['m'] = beta1 * state.get('m', 0.0) + (1 - beta1) * g
        state['v'] = beta2 * state.get('v', 0.0) + (1 - beta2) * g * g
        m_hat, v_hat = state['m'] / (1 - beta1**t), state['v'] / (1 - beta2**t)
        return x - eta * m_hat / (np.sqrt(v_hat) + eps)
    return step
```

ステップ数を見る前に、各手法にどのような情報が与えられているかに注目しよう。勾配降下法とヘビーボール法は、真の曲率 1 と 1000 から閉じた式で計算した理論上の最適設定で実行する。実際のニューラルネットワークで両端の極端な曲率を知る術はないため、これはかなり有利なハンディキャップである。一方、Adam には閉じた公式がないため、実務者が行うのと同じく 0.002 から 2 まで対数等間隔に並べた 7 個のステップサイズ（約 3 倍刻み）でスイープし、各行の最良値を読み取る。なお、このグリッドは意図的に $\eta = 1$ を避けている。Adam の最初のステップはすべての座標で長さ η をもつため（35.5.4 節）、$(1, 1)$ から $\eta = 1$ で出発すると 1 ステップで直接原点に着地してしまうからである。それは初期値の偶然による一致であり、オプティマイザの速度の公平な尺度にはならない。

```
mu, L = h.min(), h.max()
rho = (np.sqrt(L) - np.sqrt(mu)) / (np.sqrt(L) + np.sqrt(mu))
tuned = {'GD, eta = 2/(mu + L)': gd(2 / (mu + L)),
         'heavy ball, tuned': heavy_ball(4 / (np.sqrt(L) + np.sqrt(mu))**2,
                                         rho**2)}
valleys = {'aligned': rotated(0.0), 'rotated 45': rotated(np.pi / 4)}
print(f'{"steps to f < 1e-8":28s}{"aligned":>9s}{"rotated 45":>12s}')
```

```
    for name, step in tuned.items():
        hits = [run(step, *problem) for problem in valleys.values()]
        print(f'{name:28s}{hits[0] or "never":>9}{hits[1] or "never":>12}')
    etas = np.geomspace(0.002, 2.0, 7)              # step sizes for Adam
    print(f'\n{"Adam, eta =":28s}' + ''.join(f'{eta:>7.2g}' for eta in etas))
    for beta1 in (0.0, 0.9):
        for name, problem in valleys.items():
            hits = [run(adam(eta, beta1=beta1), *problem) for eta in etas]
            print(f'{f"beta1 = {beta1}, {name}":28s}'
                  + ''.join(f'{k or "never":>7}' for k in hits))
```

```
steps to f < 1e-8             aligned  rotated 45
GD, eta = 2/(mu + L)             6160        6160
heavy ball, tuned                 296         296

Adam, eta =                  0.002 0.0063   0.02  0.063    0.2   0.63      2
beta1 = 0.0, aligned          2122    760    249     78     22  never  never
beta1 = 0.0, rotated 45      never  never  never  never  never  never  never
beta1 = 0.9, aligned          2033    648    195    165    205    170    196
beta1 = 0.9, rotated 45      12862   9292   5898   3171   1581    799    559
```

まず軸に沿った谷を見てみよう。座標ごとのスケーリング単独（$\beta_1 = 0$）では、最良のステップサイズにおいてわずか 22 ステップしか要しない。勾配降下法の 6,160 ステップやヘビーボール法の 296 ステップと比べると、真の曲率を 2 つとも与えられたヘビーボール法より 13 倍少ないステップ数を、どちらの曲率も知らされていないオプティマイザが達成している。この行からは、Adam の調整が実際には何を合わせているのかも明らかになる。反復回数はほぼ $1/\eta$ に比例して変化する。序盤のステップ長が約 η であり、総ステップ数は主にステップ長を単位として測った最小点までの距離を反映するからである。ただし最大の 2 つのステップサイズでは、スケーリング単独では決して収束せず、反復点は最小点を飛び越えて各座標が $+\eta/2$ と $-\eta/2$ の間で跳ね返る周期軌道に陥る。つまり Adam は、曲率という未知量を、η を単位とする最小点までの距離という別の未知量と引き換えにしているのである。

Adam のデフォルトのモメンタム（$\beta_1 = 0.9$）を加えると、ステップサイズが 0.02 から 2 までの全範囲にわたり、反復回数は 165〜205 ステップに収まる。ここで際立つ特徴が 2 つある。1 つ目は頑健性（robustness）である。η を 100 倍変えても反復回数は 4 分の 1 程度しか動かず、モメンタムなしでは永久に振動していた最大の 2 つのステップサイズでも、モメンタムの移動平均が交互に反転する符号を減衰させるためきれいに収束する。ヘビーボール法にはそのような余裕（マージン）はない。その最適ステップサイズは安定限界 $(2 + 2\beta)/\lambda_{\max}$ の 0.1% 以内に位置しており（演習問題 8a、35.1 節）、296 ステップという結果は最大曲率をほぼ厳密に知っていることを前提としている。2 つ目の特徴は、その頑健性の代償である。モメンタムによって、Adam はスケーリング単独の場合より約 7 倍遅くなっている。その原因は $\sqrt{\beta}$ の縮小限界（35.1.3 節）にある。モメンタム β をもつ反復は、複素根をもつ任意のモードを 1 ステップあたり $\sqrt{\beta}$ の速さで縮小させ、それ以上速くはならない。座標ごとのスケーリングを施した後は、両座標とも実質的に条件数 1 の丸いボウル上を走るため、そこでの最適なモメンタムはゼロである。$\beta_1 = 0.9$ では 1 ステップあたりの縮小率は $\sqrt{0.9} \approx 0.95$ となり、許容誤差に達するまで約 230 ステップと予測され、実測値とよく一致する。悪条件（ill-conditioned）な谷に沿う進行を加速するモメンタムも、丸いボウルの上ではブレーキとして働くにもかかわらず、Adam のデフォルト設定はこれを無条件に適用する。

次に回転した谷に目を向けよう。勾配降下法とヘビーボール法は、更新式が回転不変であり問題の回転に伴ってそのまま回転するため、以前とまったく同じステップ数で収束する。しかし Adam の更新式はそうではない。スケーリング単独では、グリッド上のどのステップサイズでも

許容誤差に到達できない。モメンタム付きの Adam は収束するものの、グリッドの上端付近に限られる。最良のステップ数は $\eta = 2$ における 559 ステップでヘビーボール法の 296 ステップのほぼ 2 倍を要し、最小のステップサイズでは 12,862 ステップと通常の勾配降下法の 2 倍ものステップ数を要する。ステップサイズに対する頑健性も失われ、グリッド全体でのステップ数は 20 倍以上の開きを見せる。

したがって公平な結論は表裏二面をもつ。座標軸が主曲率方向に一致しているとき、座標ごとのスケーリングははるかに少ない事前知識でありながら、完璧に調整されたヘビーボール法に 1 桁の差をつけて勝つ。一方、谷が回転すると対角スケーリングの恩恵は完全に消え失せ、残された収束性はすべてモメンタムに由来するものとなり、その役割であれば適切に調整されたヘビーボール法のほうが 2 倍速くこなす。この両面をさらに詳しく見ていこう。

軸に沿った谷では、スケール不変性の命題により、硬い（曲率の大きい）座標が 1,000 分の 1 の実効ステップを受け取る一方で、両座標は同一の反復点をたどると予測される。以下では、反復点と実効ステップサイズ $\eta/\sqrt{\hat{\mathbf{v}}_t}$ をスケーリングのみの Adam（$\epsilon = 0$、$\eta = 0.1$）について出力する。さらに最後の 2 行では、曲率比を 5 桁にわたって変化させた場合と、初期点を原点から 100 倍遠くへ離した場合を比較し、反復回数を実際に支配している要因を検証する。

```
A, x = rotated(0.0)
step, state, eta, gap = adam(0.1, beta1=0.0, eps=0.0), {}, 0.1, 0.0
for t in range(1, 49):
    x = step(x, A @ x, t, state)
    gap = max(gap, abs(x[0] - x[1]))
    if t in (1, 10, 48):
        eff = eta / np.sqrt(state['v'] / (1 - 0.999**t))
        print(f't = {t:2d}: x = {x[0]:.3g}, {x[1]:.3g};  '
              f'eta/sqrt(v_hat) = {eff[0]:.3g}, {eff[1]:.3g}')
print(f'largest |x_1 - x_2| over the run: {gap:.1e}')
counts = [run(adam(0.1, beta1=0.0), np.diag([1.0, r]), np.ones(2))
          for r in (1.0, 1e3, 1e5)]
print('steps to f < 1e-8 at curvature ratios 1, 1e3, 1e5:', *counts)
print('steps from (100, 100) at ratio 1e3:',
      run(adam(0.1, beta1=0.0), A, 100 * np.ones(2)))
```

```
t =  1: x = 0.9, 0.9;  eta/sqrt(v_hat) = 0.1, 0.0001
t = 10: x = 0.262, 0.262;  eta/sqrt(v_hat) = 0.152, 0.000152
t = 48: x = 3.91e-06, 3.91e-06;  eta/sqrt(v_hat) = 0.329, 0.000329
largest |x_1 - x_2| over the run: 5.6e-17
steps to f < 1e-8 at curvature ratios 1, 1e3, 1e5: 40 48 54
steps from (100, 100) at ratio 1e3: 4145
```

最初の 1 ステップ目から、実効ステップサイズは正確に $1000{:}1$ の比をなしており、ヘッセ行列を一度も作ることなく勾配の大きさだけから真の曲率比が復元されている。2 つの座標は終始、浮動小数点精度（floating-point precision）の範囲で一致し、平坦な座標の実効ステップは 0.1 から 0.33 へと、縮小する勾配に $\hat{v}_t$ が適応するにつれて大きくなる。曲率比を 1 から 10^3、さらに 10^5 へと引き上げても、ステップ数は 40 から 48、54 へと増えるにすぎない。条件性が 10 万倍悪化しても、追加コストはわずか 14 ステップである。ところが初期点を 100 倍遠くに移すと、ステップ数は 4,145 まで跳ね上がる。反復回数は初期距離を η 単位で測っており、曲率の影響はほとんど受けないのである。

なぜ 45° の回転がこのすべてにとって致命的なのだろうか。回転後の行列は対角成分が 500 付近となり、急な方向 $(-1, 1)/\sqrt{2}$ が両方の座標軸に等しく射影される。そのため各座標の勾配は急なモードに支配され、両座標ともほぼ同一のスケール推定値を蓄積することになる（実行全体を

通じて両者の $\hat{v}$ は 1% 以内で一致する）。座標ごとの前処理行列は各座標に 1 つのスカラーしか割り当てられないため、ある座標の勾配に含まれる急な成分を縮小しようとすれば、平坦な成分も同じだけ縮小せざるを得ない。両座標のスケールが等しくなると、前処理行列は単位行列のスカラー倍に退化する。こうしてスケーリングのみの Adam（$\beta_1 = 0$）は、ステップサイズが急なモードに縛られた勾配降下法に先祖返りし、平坦なモードは這うようにしか進まなくなる。また $\beta_1 = 0.9$ の Adam はヘビーボール法に先祖返りする。モメンタムは最適値の 0.88 に近いものの、実効ステップ $\eta/\sqrt{\hat{v}_t}$ が曲率ではなく勾配履歴に支配されるため、グリッド上の最良点であっても調整済みヘビーボール法の 2 倍のステップ数を要する。

モメンタムがない場合、回転した谷での実行は最終的に振動に陥る。初めのうち急な成分が縮小するにつれて $\hat{\mathbf{v}}$ が減少し、実効ステップサイズが増大してやがて急なモードの安定限界 $2/h_{\max}$ を超える。軸に沿った谷での最適ステップサイズである $\eta = 0.2$ では、損失は一旦 10^{-3} 付近まで下がる（$t = 1{,}700$ 前後）ものの、その時点で急なモードが振動を始める。この定常的な振動によって勾配の大きさが一定に固定されると $\hat{\mathbf{v}}_t \to \mathbf{g}_t^2$ となり、更新は符号ステップ $\eta\,\mathrm{sign}(\mathbf{g}_t)$（谷を横切る長さ $\eta\sqrt{2}$）へと退化する。こうして反復点は振幅 $\eta/\sqrt{2}$、損失 $f = 250\,\eta^2 = 10$ の 2 周期軌道に永久に閉じ込められる（演習問題 5 ではステップサイズのグリッド全体でこの式を検証する）。モメンタムはこの交互に反転するステップを平均化して打ち消すため、$\beta_1 = 0.9$ の実行はやがて目標許容誤差に到達する。Figure 35.19 は、見やすく描画できる程度に緩やかな谷の上で、これら 3 つの挙動を示したものである。

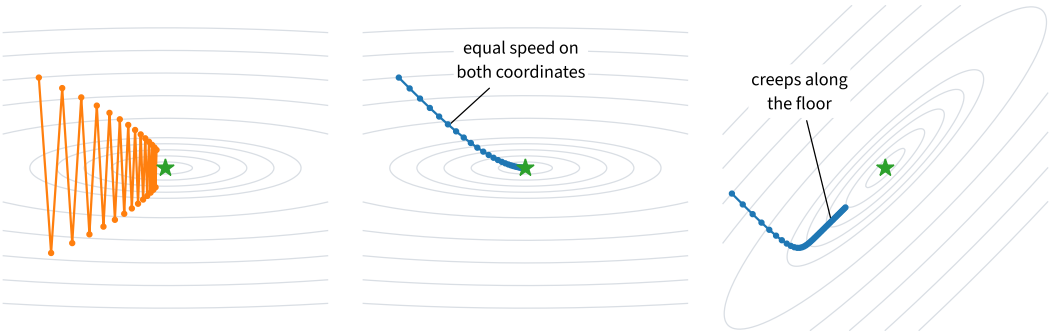


Figure 35.19: 同じ初期点から出発した 2 次元の谷（$\kappa = 20$。本文の実験では等高線が細すぎて描画できない $\kappa = 10^3$ を使用）における比較。(a) 安定限界付近の単一ステップサイズを用いた勾配降下法は、急な方向に振動しながら平坦な方向には這うように進む。(b) 座標ごとの正規化（$\beta_1 = 0$ の Adam）は両座標を同じ速さで動かし、滑らかな曲線を描いて最小点へ収束する。(c) 谷と初期点を 45° 回転させると座標軸と曲率方向がずれ、反復点は谷底に沿ってわずかにしか進めなくなる（40 ステップ時点で $f = 0.19$ であるのに対し、(b) では 4×10^{-7}）。

座標ごとのスケーリングが悪条件性を解消するのは、座標軸が曲率の固有ベクトルと一致するとき、かつそのときに限る。 スケーリングは対角行列（diagonal matrix）によって計量を変える操作であり、いかなる対角行列も座標が混ざり合う方向の曲率を捉えることはできない。外積（outer product）$\mathbf{g}_\tau \mathbf{g}_\tau^\top$ から構成したフル行列であれば問題の回転に伴って回転するためこの欠陥を免れるが、大規模なニューラルネットワークではその保持と逆行列計算のコストが高すぎて実用的ではない。本節末尾で扱う構造化前処理行列は、座標を混合するコンパクトな因子を用いることで、重み行列におけるこの限界に対処する。

35.5.2.2 Adam に対する凸問題の反例

AdaGrad が過去の勾配を無期限に記憶するのに対し、RMSProp と Adam は過去の勾配を忘れていく。この忘却によって、単純な凸問題でさえ解けなくなることがあるのだろうか。Reddi et al.

(2018) は、凸な損失の 1 次元周期系列を用いて、実際に起こり得ることを示した。

命題（Adam は収束するとは限らない）。 $C > 2$ **とし、** $x \in [-1, 1]$ **上での次の周期的な損失によるオンライン最適化を考える。**

$$f_t(x) = \begin{cases} C\,x, & t \bmod 3 = 1, \\ -x, & \text{otherwise}. \end{cases} \tag{35.55}$$

バイアス補正なしの Adam を $\beta_1 = 0$、$\beta_2 = 1/(1 + C^2)$、**ステップサイズ** $\eta_t = \eta/\sqrt{t}$（**ただし** $\eta < \sqrt{1 - \beta_2}$）**とし、各反復点を** $[-1, 1]$ **上へ射影（projection）しながら** $x_1 = 1$ **から実行する。このとき、すべてのステップで** $x_t > 0$（**任意の** t）**かつ** $x_{3k+1} = 1$（**すべての** $k \geq 0$）**となる。3 ステップの各周期がリグレットを少なくとも** $2C - 4 > 0$ **増加させるため、平均リグレットがゼロに収束することはない。** (Reddi et al. 2018)

ここで $[-1, 1]$ への射影とは単なるクリッピングである（一般的な射影は 35.6 節で扱う）。3 ステップの各周期にわたる勾配の和は $C - 2 > 0$ であるため、事後的に最良の固定点は $x^\star = -1$ である。それにもかかわらず、Adam は毎周期の開始時点で区間内の最悪点である $x = +1$ に戻ってしまう。各周期で Adam は少なくとも $C - 2$ の損失を被る（最初のステップでは $x = 1$ にいるため C を被り、残る 2 ステップでもそれぞれ少なくとも -1 を被る）一方、$x^\star = -1$ での損失は $-(C - 2)$ であるため、1 周期あたり少なくとも $2C - 4$ のリグレット増分が生じる。証明は周期に関する帰納法で進み、演習問題 3 ではその要となるステップを確認する。そこでは $\eta < \sqrt{1 - \beta_2}$ という条件により、下向きのステップが $(0, 1]$ の内部にとどまることが保証される。また、この破綻は $\beta_1 = 0$ と置いたことによる特殊な現象ではない。同論文は $\beta_1 < \sqrt{\beta_2}$ を満たす任意の固定ペアに対し、Adam が収束に失敗するオンライン凸系列と確率的凸問題の両方を構成している。

$C = 4$、$\beta_2 = 1/17$ とし、v が定常的な周期サイクルに落ち着いた後の 1 周期を見れば、その仕組みを直接確認できる。勾配 $+4, -1, -1$ に伴い、二次モーメント推定値 v は 15.12、1.83、1.05 という値をとり、$\sqrt{v}$ は 3.89、1.35、1.02 となる。その結果生じる 3 回の変位 $-\eta g_t/\sqrt{v_t}$ は次のようになる。

$$-1.03\,\eta, \qquad +0.74\,\eta, \qquad +0.98\,\eta,$$

これにより、1 周期あたり $+0.69\,\eta$ の正味のドリフト（drift）が $+1$ に向かって生じる。正規化によってまれな大勾配 $+C$ による移動がおよそ η（大きさ 1 の勾配による移動とほとんど変わらない大きさ）に頭打ちにされるうえ、v は 2 ステップのうちに C^2 のスパイクを忘れてしまう。そのため、2 回の -1 の勾配が x を押し上げる量の合計が、1 回の $+C$ の勾配が引き下げる量を上回るのである。これに対し、ステップ $\eta g_t/C$ をもつ通常の SGD は $-\eta$、$+0.25\,\eta$、$+0.25\,\eta$ だけ動き、最適点に向かって正味 $-0.5\,\eta$ 進む。定数ステップ SGD のノイズフロア（35.4 節）とは異なり、Adam の逆方向のドリフトはステップサイズを縮小しても消えず、あらゆるステップサイズで $+1$ の方向を向く。

この欠陥を修正するため、Reddi et al. (2018) は **AMSGrad** を提案した。AMSGrad は分母の $\mathbf{v}_t$ を、補正なし移動平均の累積最大値 $\tilde{\mathbf{v}}_t = \max(\tilde{\mathbf{v}}_{t-1}, \mathbf{v}_t)$ に置き換える。これにより実効ステップ $\eta_t/\sqrt{\tilde{v}_t}$ が非増加となるため、大きな勾配が決して忘れられなくなる。先ほどの周期的な例では、$\tilde{v}$ が終始 15.12 に固定されるため、3 回の変位は $-1.03\,\eta$、$+0.26\,\eta$、$+0.26\,\eta$ へと変わり、最適点に向かう正味 $-0.51\,\eta$ の移動となる。したがって Adam は $+1$ へ収束する一方、AMSGrad と SGD は -1 へ収束するはずである。以下のセルでは、同じ $1/\sqrt{t}$ 減衰のもとで、Adam が最初から悪い端点に置かれることのないよう区間の中点 $x = 0$ から出発させて 3 手法すべてを検証する。

```
def online(update, C=4.0, T=15000, alpha=0.5):
    """Projected iterates on the periodic losses; beta_1 = 0, no bias correction."""
    beta2 = 1 / (1 + C * C)
    x, v, vmax, xs = 0.0, 0.0, 0.0, []
    for t in range(1, T + 1):
        g = C if t % 3 == 1 else -1.0            # rare +C, frequent -1
        v = beta2 * v + (1 - beta2) * g * g
        vmax = max(vmax, v)                      # running maximum (AMSGrad)
        x = min(1.0, max(-1.0, x - alpha / np.sqrt(t) * update(g, v, vmax, C)))
        xs.append(x)
    return xs

updates = {'adam': lambda g, v, vmax, C: g / (np.sqrt(v) + 1e-8),
           'amsgrad': lambda g, v, vmax, C: g / (np.sqrt(vmax) + 1e-8),
           'sgd': lambda g, v, vmax, C: g / C}
print('x_t at t =        30      300     3000    15000   (x* = -1)')
for name, update in updates.items():
    xs = online(update)
    print(f'{name:9s}' + ''.join(f'{xs[t - 1]:9.3f}' for t in (30, 300, 3000, 15000)))
```

```
x_t at t =        30      300     3000    15000   (x* = -1)
adam           0.745    1.000    1.000    1.000
amsgrad       -0.953   -0.985   -0.995   -0.998
sgd           -0.954   -0.986   -0.995   -0.998
```

ここで alpha は命題における η の役割を果たしており、$\eta = 0.5 < \sqrt{16/17}$ であるためステップサイズに関する仮定を満たしている。区間の中点から出発してもなお、Adam は $+1$ へ一直線に向かう。$t \approx 300$ 以降は毎周期 $+1$ に戻り、0.97 を下回ることすらなくなる。その一方で、AMSGrad と SGD は最適点 -1 へと着実に近づき、小数第 2 位まで一致する。この反例は Adam に一般的な凸問題での収束保証（convergence guarantee）がないことを証明するものであるが、典型的な深層学習問題での失敗を予測するものとしてではなく、あくまで理論的な存在結果として受け止めるべきである。

35.5.3 分離型重み減衰

35.5.3.1 結合型減衰と分離型減衰

現代のほぼすべての訓練レシピ（training recipe）には重み減衰が含まれており、そのため適応的オプティマイザには「減衰項を前処理行列に通すべきか」という設計上の選択が迫られる。SGD のもとでは、この区別は存在しない。損失に ℓ_2 ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$ を加えると勾配に $\lambda\mathbf{w}$ が加わり、得られる更新 $\mathbf{w}_t - \eta\,(\mathbf{g}_t + \lambda\mathbf{w}_t) = (1 - \eta\lambda)\,\mathbf{w}_t - \eta\,\mathbf{g}_t$ は、損失に対する勾配ステップであると同時に重みの一様な縮小（shrinkage）でもある（9.5.3 節、ただし同節では重みを $\mathbf{x}$ と表記している）。しかし Adam の内部では、これら 2 つの仕組みは乖離する。ペナルティ付きの勾配を Equation 35.53 に入力して減衰項を取り出すと（見通しをよくするため $\beta_1 = 0$ と置けば更新量は $-\eta\,(\mathbf{g}_t + \lambda\mathbf{w}_t)/(\sqrt{\widehat{\mathbf{v}}_t} + \epsilon)$ となる）、座標 i の 1 ステップあたりの縮小量は次のようになる。

$$\text{coupled } (\ell_2 \text{ through Adam}): \quad \frac{\eta\,\lambda}{\sqrt{\widehat{v}_{t,i}} + \epsilon}\, w_{t,i} \qquad \text{versus} \qquad \text{decoupled:} \quad \eta\,\lambda\, w_{t,i}. \tag{35.56}$$

結合型（coupled）の Adam では、前処理行列が損失の勾配とペナルティの勾配の両方を割るため、各座標の縮小率はその勾配スケールに反比例する。すなわち、大きな勾配をもつ重みはほとんど減衰しない一方、勾配の小さい静かな重みは強く減衰させられる。**AdamW** (Loshchilov and Hutter 2019) はこの 2 つの操作を分離（decoupled）し、損失の勾配のみを前処理行列に通して、重み減衰はパラメータに直接作用させる。

$$\mathbf{w}_{t+1} = (1 - \eta\lambda)\,\mathbf{w}_t - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}. \tag{35.57}$$

これにより、SGD とまったく同様にすべての座標が毎ステップ同じ係数 $1 - \eta\lambda$ で縮小し、モーメントの移動平均は損失の勾配のみを追跡するようになる。

`optax.adam` には重み減衰の引数がなく、`optax.adamw` が分離型の更新則を実装している。結合型の更新則を用いるには、損失に明示的なペナルティを加えるか、`optax.chain(optax.add_decayed_weights(lam), optax.adam(eta))` を使って Adam に渡す前の勾配に $\lambda\mathbf{w}$ を加える。

Equation 35.56 の違いが最も鮮明に現れる検証は、損失の勾配が平均ゼロの純粋なノイズであり、重み減衰だけが系統的なドリフトとなる重みを調べることである。ノイズの標準偏差が σ であれば $\sqrt{\hat{v}} \approx \sigma$ となるため、結合型 Adam では重みが約 $e^{-\eta\lambda T/\sigma}$（$T$ ステップ後）まで縮小すると予測されるのに対し、AdamW ではすべての重みが $(1-\eta\lambda)^T$ へと σ によらず縮小するはずである。以下では、10^4 個の重み（ノイズスケール $\sigma = 10$）と別の 10^4 個の重み（$\sigma = 0.1$）をすべて $w = 1$ に初期化し、各手法を 4,000 ステップ（$\eta = 10^{-3}$、$\lambda = 0.1$）実行して、グループごとの平均値をこれらの予測値と比較する。

```
sigma, eta, lam, T, n = np.array([[10.0], [0.1]]), 1e-3, 0.1, 4000, 10_000
rng = np.random.default_rng()

def decay(decoupled, beta1=0.9, beta2=0.999, eps=1e-8):
    """Mean of w over n weights per noise scale after T Adam steps from w = 1."""
    w, m, v = np.ones((2, n)), np.zeros((2, n)), np.zeros((2, n))
    for t in range(1, T + 1):
        g = sigma * rng.standard_normal((2, n))     # loss gradient: pure noise
        if decoupled:
            w = (1 - eta * lam) * w                 # decay outside the preconditioner
        else:
            g = g + lam * w                         # l2 penalty inside the gradient
        m = beta1 * m + (1 - beta1) * g
        v = beta2 * v + (1 - beta2) * g * g
        w = w - eta * (m / (1 - beta1**t)) / (np.sqrt(v / (1 - beta2**t)) + eps)
    return w.mean(axis=1)

print('per-step decay rates: coupled', eta * lam / sigma.ravel(), ' AdamW', eta * lam)
print('mean w, coupled:', decay(False).round(3),
      '  predictions exp(-eta lam T / sigma):', np.exp(-eta * lam * T /
      ↪  sigma.ravel()).round(3))
print('mean w, AdamW:  ', decay(True).round(3),
      '  prediction (1 - eta lam)^T:', round((1 - eta * lam)**T, 3))
```

```
per-step decay rates: coupled [1.e-05 1.e-03]  AdamW 0.0001
mean w, coupled: [0.961 0.028]   predictions exp(-eta lam T / sigma): [0.961 0.018]
mean w, AdamW:   [0.671 0.671]   prediction (1 - eta lam)^T: 0.67
```

単一の λ の値から、2 つのグループには 1 ステップあたり 10^{-5} と 10^{-3} という 100 倍も異なる結合型減衰率が与えられる。ノイズの大きい重みは予測どおり初期の大きさの 0.96 を保っている一方、静かな重みは約 0.03 まで低下しており、予測値の 0.018 をわずかに上回る。このわずかな差の原因は明快である。$w = 1$ 付近では、静かな重みに対するペナルティ勾配 $\lambda w = 0.1$ がノイズと同程度の大きさをもち、その重み自身の $\hat{v}$ を押し上げて初期の減衰を遅らせるためである。これとは対照的に、AdamW のもとでは両グループともぴったり $0.67 = (1 - \eta\lambda)^{4000}$ に到達する。正規化されたノイズステップの平均はゼロであり、乗法的な減衰が毎ステップ同一であるため、AdamW ではこの予測が期待値において厳密に成り立つ。

35.5.3.2 不動点

純粋なノイズの実験では勾配が決して落ち着かないため、停留点（stationary point）は存在しない。しかし厳密な勾配のもとではオプティマイザは静止でき、その不動点において λ は驚くほど異なる意味を帯びる。

命題（結合型減衰と分離型減衰の不動点）。 勾配 $\mathbf{g}(\mathbf{w}) = \nabla \ell(\mathbf{w})$（$\ell$ は損失）が厳密（ノイズなし）であり、$\epsilon > 0$ とする。モーメント推定値が定常状態に落ち着いた後に更新量がゼロになる点 $\mathbf{w}$ を、その更新の不動点と呼ぶ。(a) $\ell + \frac{\lambda}{2}\|\mathbf{w}\|^2$ 上の Adam が $\mathbf{w}$ に不動点をもつのは $\nabla \ell(\mathbf{w}) + \lambda \mathbf{w} = \mathbf{0}$ のとき、かつそのときに限り、これはペナルティ付き損失に対する SGD と同じ停留点である。(b) AdamW が $\mathbf{w}$ に不動点をもつのは、すべての座標 i について次式が成り立つとき、かつそのときに限る。

$$\lambda\, w_i (|g_i| + \epsilon) = -g_i, \qquad g_i = \partial_i \ell(\mathbf{w}). \tag{35.58}$$

したがって AdamW のすべての不動点はボックス $|w_i| < 1/\lambda$ の厳密な内部に存在し、$\epsilon \to 0$ の極限ではこの条件は二分法に帰着する。すなわち、$g_i = 0$ であるか、あるいは $|w_i| = 1/\lambda$ かつ w_i が g_i と逆符号になるかのいずれかである。

証明。 不動点ではモーメントの移動平均に入力されるベクトル $\mathbf{u}$ が定数となるため、$\hat{\mathbf{m}}_t \to \mathbf{u}$ かつ $\hat{\mathbf{v}}_t \to \mathbf{u}^2$ となる。(a) 結合型 Adam では $\mathbf{u} = \mathbf{g} + \lambda \mathbf{w}$ であり、座標 i の更新量は $-\eta\, u_i / (|u_i| + \epsilon)$ に収束するため、これがゼロになるのは $u_i = 0$ のとき、かつそのときに限る。(b) AdamW では $\mathbf{u} = \mathbf{g}$ であるため、座標 i の更新量は $-\eta\lambda w_i - \eta\, g_i/(|g_i| + \epsilon)$ に収束する。これをゼロと置いて両辺に $(|g_i| + \epsilon)/\eta$ を掛けると Equation 35.58 が得られる。絶対値をとれば $|w_i| = |g_i|/(\lambda(|g_i| + \epsilon)) < 1/\lambda$ となる。さらに $\epsilon \to 0$ とすると条件は $\lambda w_i |g_i| = -g_i$ となり、これは $g_i = 0$ または $\lambda w_i = -\mathrm{sign}(g_i)$ のときにちょうど成り立つ。■

この命題は不動点の性質を特徴づけるものであり、不動点の存在やそこへの反復点の収束を主張するものではない。また、この二分法は単なる極限における珍現象でもない。$1 - \lambda|w_i| = \epsilon/(|g_i| + \epsilon)$ が成り立つため、$\epsilon = 10^{-8}$ においては、すべての座標がボックス境界から割合 $\epsilon/|g_i|$ 以内に張り付いているか、さもなくば ϵ のオーダーの勾配をもつかの一方になる。

1 次元の例を見ると、この対比が具体的になる。制約なしの最小点が $\ell(w) = \frac{1}{2}(w - 3)^2$ に対して $w = 3$ である関数を考えよう。$\lambda = \frac{1}{4}$ のとき、結合型 Adam はペナルティ付き最小点 $3/(1 + \lambda) = 2.4$ で静止するのに対し、AdamW は 3.0 で静止する。ボックス $|w| \le 4$ の制約が有効にならないため、分離型重み減衰は最小点をそのまま残すのである。一方 $\lambda = 1$ のときは、結合型 Adam が 1.5 で静止するのに対し、AdamW はちょうどボックスの境界上にあたる 1.0 に張り付く。

このように、λ の 2 つの解釈はまったく異なる。ノイズが支配的な訓練領域では、AdamW が全座標に毎ステップ一様な縮小を適用する一方、結合型 Adam は勾配スケールによってそれを歪めてしまう。ところが厳密な勾配での不動点においては役割が逆転する。結合型 Adam は重みにガウス事前分布（Gaussian prior）を置いた最大事後確率推定（maximum a posteriori、MAP）の

解を保つのに対し（36.6 節）、AdamW は λ を半径 $1/\lambda$ の ℓ_∞ 制約ボックスへと変えるのである (Xie and Li 2024)。演習問題 6 では、このボックスを制約付き最適化（constrained optimization）と直接結びつける。

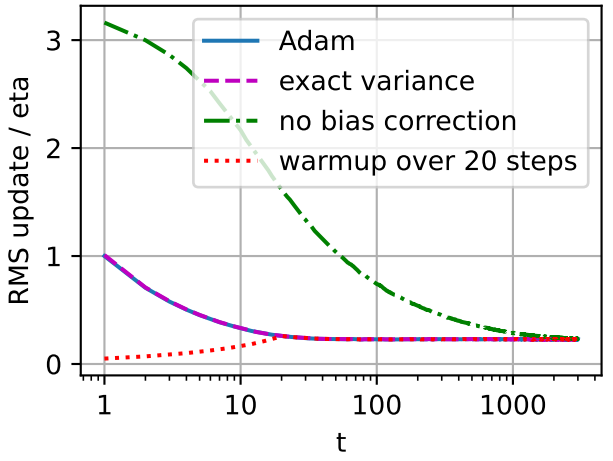
35.5.4 ウォームアップ

ステップ $t = 1$ では、バイアス補正後の Adam のモーメントは $\hat{\mathbf{m}}_1 = \mathbf{g}_1$ および $\hat{\mathbf{v}}_1 = \mathbf{g}_1^2$ となるため、更新量は単に $\eta\,\mathrm{sign}(\mathbf{g}_1)$、すなわちすべての座標でフルサイズ η のステップとなる。たった 1 つのミニバッチから得られた証拠だけで、勾配が 10^{-6} の座標も勾配が 10^{3} の座標とまったく同じ距離だけ動くのである。**ウォームアップ**は、訓練の序盤ステップにわたってステップサイズをゼロ付近から徐々に引き上げることで (Goyal et al. 2017)、こうした初期の更新を小さく抑える。この初期の不安定性は前処理行列 $\hat{\mathbf{v}}_t$ が $t = 1$ において 1 つのサンプルにしか基づいていないせいだと考えたくなるかもしれない。しかし単純な実験を行ってみると、別の要因が働いていることがわかる。

大きさ η のステップは本当に大きすぎるのだろうか。向きが一貫した勾配をもつ座標にとっては定常状態のステップも η であるため、大きすぎるわけではない。しかし勾配が純粋なノイズである座標にとっては、それは無駄な動きとなる。この無駄な動きがどれほど大きいかを測るため、各ステップで 2×10^4 個の独立な勾配座標を $\mathcal{N}(0, 1)$ からサンプリングし、デフォルトの Adam を実行して、二乗平均平方根の更新量を η 単位で追跡する。ここでは 4 本の曲線を比較する。標準の Adam、$\hat{\mathbf{v}}_t$ を厳密な分散 $\sigma^2 = 1$ に置き換えた Adam（前処理行列の推定誤差を完全に除いたもの）、両方のバイアス補正を取り除いた Adam、そして最初の 20 ステップにわたる線形ウォームアップを掛けた Adam である。

```
n, T, beta1, beta2, eps, T_w = 20_000, 3000, 0.9, 0.999, 1e-8, 20
m, v = np.zeros(n), np.zeros(n)
rms = {'Adam': [], 'exact variance': [], 'no bias correction': [],
       f'warmup over {T_w} steps': []}
for t in range(1, T + 1):
    g = rng.standard_normal(n)                      # pure noise, sigma = 1
    m = beta1 * m + (1 - beta1) * g
    v = beta2 * v + (1 - beta2) * g * g
    m_hat, v_hat = m / (1 - beta1**t), v / (1 - beta2**t)
    step = m_hat / (np.sqrt(v_hat) + eps)          # Adam's update / eta
    for r, u in zip(rms.values(), (step, m_hat, m / (np.sqrt(v) + eps),
                                   min(1.0, t / T_w) * step)):
        r.append(np.sqrt(np.mean(u**2)))
d2l.plot(np.arange(1, T + 1), list(rms.values()), 't', 'RMS update / eta',
         legend=list(rms), xscale='log')
ts = (1, 2, 3, 10, 20, 30, 100, 1000, 3000)
print(f'{"t:":24s}' + ''.join(f'{t:>6d}' for t in ts))
for name, r in rms.items():
    print(f'{name + ":":24s}' + ''.join(f'{r[t - 1]:6.2f}' for t in ts))
```

```
t:                          1     2     3    10    20    30   100  1000  3000
Adam:                    1.00  0.71  0.58  0.33  0.26  0.24  0.23  0.23  0.23
exact variance:          1.01  0.71  0.58  0.33  0.26  0.24  0.23  0.23  0.23
no bias correction:      3.16  3.00  2.86  2.17  1.62  1.34  0.74  0.29  0.24
warmup over 20 steps:    0.05  0.07  0.09  0.17  0.26  0.24  0.23  0.23  0.23
```



標準の Adam は $1.00\,\eta$ の RMS 更新量から始まり、$0.33\,\eta$（$t = 10$）まで下がり、約 50 ステップ以内に $\sqrt{(1-\beta_1)/(1+\beta_1)}\,\eta = 0.23\,\eta$ へと落ち着く。これは独立な単位ノイズの指数移動平均がもつ定常状態の大きさである。したがって純粋なノイズ座標においては、Adam の序盤ステップは定常時の約 4 倍の大きさになり、この過渡状態は約 10 ステップ続く。

原因はノイズを含んだ前処理行列にあるのだろうか。2 本目の曲線はそれを否定する。$\hat{\mathbf{v}}_t$ を厳密な分散に置き換えても、RMS 更新量はすべてのステップで 0.01 以内（2×10^4 個の座標によるサンプリング誤差の範囲）で変わらない。4 倍の膨張は**モメンタム**の移動平均に由来するものであり、独立なノイズを平均化して打ち消すまでに約 $1/(1-\beta_1) = 10$ ステップを要するのである。$t = 1$ において前処理行列が実際に行っているのは異なるスケールの座標を均一化することであり、すべての座標が $\sigma = 1$ を共有している設定ではその作用が隠れているにすぎない。

はるかに影響が大きいのはバイアス補正である。これがないと、最初の更新量は $3.16\,\eta$ まで跳ね上がり（$(1-\beta_1)/\sqrt{1-\beta_2}$ という Figure 35.18 の倍率と一致する）、過渡状態も 10 ステップから数百ステップへと引き伸ばされる。補正なしの更新量は $0.75\,\eta$ 前後（$t = 100$）、さらに $0.29\,\eta$（$t = 1000$）も残るが、これは補正なしの $\mathbf{v}_t$ が $1/(1-\beta_2) = 1000$ という遅い時間スケールでしか立ち上がらないためである。バイアス補正のないオプティマイザでは、自身のゼロ初期化の影響を打ち消すためだけに 1,000 ステップものウォームアップが必要になってしまう。バイアス補正を備えていれば、わずか 20 ステップの線形な立ち上げで残りの膨張を抑え込め、ウォームアップ曲線は終始 $0.26\,\eta$ 未満に保たれる。

これで数十ステップのウォームアップは説明できるが、大規模訓練でよく使われる数千ステップものウォームアップは説明できない。より長いウォームアップが対処しているのは、このノイズモデルには含まれない第 2 の仕組み、すなわち曲率である。ステップサイズが安定となるのは、最も急な曲率方向が課す上限 $2/\lambda_{\max}$ 未満の領域に限られる（35.1.2 節）。ランダム初期化（random initialization）の直後には、この曲率が異常に大きく、かつ急激に変化し得るため、訓練の後半では安全なステップサイズであっても初期には不安定になることがある。9.7.3 節では実際のニューラルネットワーク上でこれら両方の仕組みを再訪し、序盤のステップサイズを直接計測する。

35.5.5 構造化前処理行列

35.5.5.1 クロネッカー因子

回転した谷は、あらゆる対角前処理行列を打ち負かした。本節で扱ってきた手法はすべて、正定値行列（positive definite matrix）を用いた前処理付きの形式 $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\,\mathbf{P}_t^{-1}\mathbf{g}_t$（$\mathbf{P}_t$ が前処

理行列）を共有している。勾配降下法は $\mathbf{P} = \mathbf{I}$ を使い、ニュートン法はヘッセ行列 $\nabla^2 f$ を使い、Adam は過去の勾配から推定した対角行列を使う。フルサイズの曲率行列であれば、座標軸に沿っているか否かを問わずあらゆる方向の急峻さを捉えられるが、巨大すぎて保持できない。たった 1 枚の 4096×4096 重み行列でさえ 1.7×10^7 個のパラメータをもち、そのフル曲率行列は 2.8×10^{14} 個もの要素を含んで、倍精度（double precision）では約 2 ペタバイトものメモリを必要とする。

この両極端の中間を可能にしているのが、ニューラルネットワークのパラメータが行列の形に編成されているという事実である。線形層 $\mathbf{y} = \mathbf{W}\mathbf{a}$ において、重み行列を $\mathbf{W} \in \mathbb{R}^{m \times n}$、入力を $\mathbf{a} \in \mathbb{R}^n$、出力に関する勾配を $\delta = \partial \ell / \partial \mathbf{y} \in \mathbb{R}^m$ としよう。$\mathbf{W}$ に関する勾配はランク 1 の外積 $\delta \mathbf{a}^\top$ となる。行列の列を縦に並べて得られるベクトルを $\mathrm{vec}(\mathbf{V})$（$\mathbf{V}$ は任意の行列）と書くと、$\mathrm{vec}(\delta \mathbf{a}^\top) = \mathbf{a} \otimes \delta$ が成り立つ。ここで**クロネッカー積**（Kronecker product）$\mathbf{B} \otimes \mathbf{D}$ とは、第 (i, j) ブロックが $b_{ij}\mathbf{D}$ であるブロック行列を表す。

したがって、この層の外積による曲率、すなわちフィッシャー情報行列（Fisher information matrix）$\mathbb{E}[\nabla \log p \, \nabla \log p^\top]$（36.6 節で学ぶ）のうち $\mathbf{W}$ に対応するブロックは $\mathbb{E}\big[(\mathbf{a}\mathbf{a}^\top) \otimes (\delta\delta^\top)\big]$ となる。**K-FAC** (Martens and Grosse 2015) は、この期待値をクロネッカー積 $\mathbf{S}_{\text{in}} \otimes \mathbf{S}_{\text{out}}$ で近似する。ここで $\mathbf{S}_{\text{in}} = \mathbb{E}[\mathbf{a}\mathbf{a}^\top]$ は $n \times n$ 行列であり、$\mathbf{S}_{\text{out}} = \mathbb{E}[\delta\delta^\top]$ は $m \times m$ 行列である。この分解は入力と出力勾配が統計的に独立であれば厳密に成り立つ。訓練中のネットワークでそれが成り立つとは誰も考えていないが、安価に逆行列を計算できるためにこの近似が採用されている。

具体的には、クロネッカー積の逆行列はベクトル化する前の勾配行列に直接作用する。対称かつ正則な因子に対しては、次式が成り立つ。

$$\big(\mathbf{S}_{\text{in}} \otimes \mathbf{S}_{\text{out}}\big)^{-1} \mathrm{vec}(\mathbf{V}) = \mathrm{vec}\big(\mathbf{S}_{\text{out}}^{-1} \mathbf{V} \mathbf{S}_{\text{in}}^{-1}\big). \tag{35.59}$$

この恒等式を確かめるには、積の第 j 列である $\mathbf{DVB}$ の列を $\sum_k b_{kj} \mathbf{D}\mathbf{v}_k$（$\mathbf{v}_k$ は第 k 列、$\mathbf{V}$ の列ベクトル）と書けばよい。これらの列を縦に並べるとブロック積 $(\mathbf{B}^\top \otimes \mathbf{D})\,\mathrm{vec}(\mathbf{V})$ が得られるため、$\mathrm{vec}(\mathbf{DVB}) = (\mathbf{B}^\top \otimes \mathbf{D})\,\mathrm{vec}(\mathbf{V})$ となる。$(\mathbf{B} \otimes \mathbf{D})^{-1} = \mathbf{B}^{-1} \otimes \mathbf{D}^{-1}$ と各因子の対称性を用い、$\mathbf{B} = \mathbf{S}_{\text{in}}^{-1}$ および $\mathbf{D} = \mathbf{S}_{\text{out}}^{-1}$ と置けば Equation 35.59 が導かれる。このようにして、$(mn)^2$ 個の要素をもつ曲率行列の逆行列計算と適用が、サイズ n と m の手頃な 2 つの逆行列を通じて実行できる。以下のセルでは、先ほどの 4096×4096 の層についてメモリ量と演算回数を比較する。

```
m = n = 4096
print(f'stored entries: full {(m * n)**2:.1e}, two factors {m**2 + n**2:.1e}')
print(f'flops to invert: full {(m * n)**3:.1e}, two factors {m**3 + n**3:.1e}')
```

```
stored entries: full 2.8e+14, two factors 3.4e+07
flops to invert: full 4.7e+21, two factors 1.4e+11
```

因子分解により、保持する要素数は 2.8×10^{14} 個から 3.4×10^7 個へ、逆行列計算のコストは 4.7×10^{21} 回の演算から約 1.4×10^{11} 回へと削減される。

35.5.5.2 K-FAC、Shampoo、Muon

クロネッカー積の恒等式には、どの行列を因子分解するか、そしてそれにどの指数を適用するかという 2 つの選択肢が残されている。現代の構造化オプティマイザは、層の勾配行列に対する両側からの前処理（preconditioning）形式 $\mathbf{G} \mapsto \mathbf{P}_{\text{l}}^{-a} \mathbf{G} \mathbf{P}_{\text{r}}^{-a}$ を共有している（ここでの勾配行列 $\mathbf{G}$ は太字であり、リグレット解析におけるスカラーの勾配上界 G とは区別する）。これらの手法を並べると、Adam の対角スケーリングから行列レベルの正規化へと至る自然な発展が見えてくる。

K-FAC は層の入力と出力勾配の二次モーメントから因子を構成し、指数を $a = 1$ とする。これはフィッシャー情報行列による前処理、すなわち**自然勾配**（natural gradient）(Amari 1998) を近似しており、フィッシャー行列を計量として用いた AdaGrad の計量的解釈にあたる。ソフトマックス交差エントロピー（cross-entropy）および二乗損失（squared loss）に対しては、（ラベルをモデルからサンプリングした）フィッシャー行列はガウス・ニュートン行列と一致する。ガウス・ニュートン行列とは、ネットワークの二階導関数を含む項を落としたヘッセ行列の半正定値（positive semidefinite）近似である。したがってこれらの損失に対して、K-FAC は真の曲率を近似している。

これに対し **Shampoo** (Gupta et al. 2018) は、AdaGrad の発想に沿って累積した勾配行列から左右の因子を直接構成し、$\mathbf{P}_{\mathrm{l}} = \sum_t \mathbf{G}_t \mathbf{G}_t^{\top}$ および $\mathbf{P}_{\mathrm{r}} = \sum_t \mathbf{G}_t^{\top} \mathbf{G}_t$ と置いて指数を $a = \frac{1}{4}$ とする。**Muon** (Jordan et al. 2024) は記憶をもたない Shampoo とみなすことができ、現在の行列 $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^{\top}$（正方かつ正則と仮定する）のみを用いて次のように計算する。

$$\left(\mathbf{G}\mathbf{G}^{\top}\right)^{-1/4} \mathbf{G} \left(\mathbf{G}^{\top}\mathbf{G}\right)^{-1/4} = \mathbf{U}\Sigma^{-1/2}\,\Sigma\,\Sigma^{-1/2}\mathbf{V}^{\top} = \mathbf{U}\mathbf{V}^{\top},$$

これは $\mathbf{G}$ の極因子（polar factor）である。この変換は $\mathbf{G}$ の特異ベクトルを保ったまま、すべての特異値（singular value）を 1 に揃える。Adam が更新の各座標を均一化するのに対し、Muon は特異方向を均一化する。しかも座標ごとの正規化とは異なり、この写像は行および列の直交回転と可換である（演習問題 5d）。この直交化をモメンタム行列に適用する実用オプティマイザとしての Muon は本編（9.6 節）で扱い、行列積（matrix multiplication）のみを用いて極因子を計算する Newton–Schulz 反復については 33.5.4.2 節で解析する。

これら 3 つの手法はいずれも、フルサイズのニュートン法を再現するものではない。Muon は二次モーメント行列を一切保持せず 1 ステップあたり数回の行列積しか必要としないのに対し、K-FAC と Shampoo は 2 つの因子の逆行列や行列べき根を計算する必要があり、通常はそのコストを数ステップにわたって償却する。さらに、本節で示した肯定的な結果はすべて、曲率が座標軸に沿っているか、個々の層のクロネッカー構造に沿っていることを仮定している。深層ニューラルネットワークの真の曲率は対角でも固定でもないため、これらの適応的手法が実際の訓練でこれほど高い性能を発揮する理由を説明することは、今なお活発に研究されている未解決問題である。

35.5.6 まとめ

AdaGrad、RMSProp、Adam に共通する基本原理である「各座標の更新量を過去の勾配の二乗平均平方根で割る」という操作は、曲率が座標軸に沿っている限り条件数を消去する。$\epsilon = 0$ のとき、Adam の反復点は座標ごとの再スケーリングに対して厳密に不変であり、軸に沿った谷の上ではスケーリング単独で最適調整されたヘビーボール法に 1 桁の差をつけて勝つことができる。しかし谷を 45° 回転させるとその優位性は消え失せ、スケーリング単独では決して収束せず、モメンタム付きの Adam でもヘビーボール法の約 2 倍のステップ数を要する。

過去の勾配を忘却することは、さらに微妙な問題をもたらす。凸な損失の周期的な系列上では、Adam の短い二次モーメント記憶によって定義域の最悪点へと駆動されることがあり、この破綻は AMSGrad の累積最大値によって修正される。重み減衰を伴う場合、ℓ_2 ペナルティを Adam の内部に通すと MAP の不動点は保たれるものの、各座標の縮小量がその勾配の大きさに反比例してスケーリングされてしまう。これに対し AdamW は重み減衰を分離してすべての座標が毎ステップ一様に縮小するようにし、厳密な勾配での不動点においては λ を半径 $1/\lambda$ の ℓ_∞ ボックスへと変える。

最後に、純粋なノイズ座標における Adam の序盤の更新量は、前処理行列ではなくモメンタムの移動平均が原因となって定常時の約 4 倍の大きさになり、バイアス補正を省くと数百ステップに

わたってさらに大きく膨張する。一方、実務で使われるより長いウォームアップは初期の曲率に対処している。対角スケーリングの先には、K-FAC、Shampoo、Muon のようなクロネッカー分解型の前処理行列があり、2 つのコンパクトな因子を用いて重み行列内の座標間の構造を捉え、なかでも Muon は層の更新における特異方向を均一化する。

35.5.7 演習問題

1. **計量の観点からの AdaGrad**。演習問題 1b（35.1 節）では、ノルム $\|\mathbf{d}\|_{\mathbf{P}} = \sqrt{\mathbf{d}^\top \mathbf{P}\mathbf{d}}$ のもとでの最急降下方向が $-\mathbf{P}^{-1}\nabla f$ に比例することを示した。
   1. $\mathbf{P}_t = \operatorname{diag}\big(\sqrt{\mathbf{s}_t}\big) + \epsilon \mathbf{I}$（ただし $\mathbf{s}_t = \sum_{\tau \le t} \mathbf{g}_\tau^2$）と置いて Equation 35.51 を導出せよ。
   2. 目的関数 f を cf（定数 $c > 0$）に置き換え、$\epsilon = 0$ とする。AdaGrad の更新はどう変化するか。もし $\mathbf{P}_t$ を $\mathbf{s}_t$ から構成し、$\sqrt{\mathbf{s}_t}$ を使わなかった場合はどう変化するか。両者を勾配降下法およびニュートン法と比較せよ。
   3. ある 1 つの座標の勾配が独立同分布に従い $\mathbb{E}[g_{t,i}^2] = \sigma^2$ を満たすとする。上記 2 つの選択のそれぞれのもとで、その座標の実効ステップは t に関してどの速さで減衰し、σ にどう依存するか。
2. **集中した勾配と AdaGrad の上界**。定数倍を除けば、調整された大域ステップをもつ射影付きオンライン勾配降下法は、リグレット $D_2 G\sqrt{T}$ をユークリッド直径 D_2 の凸集合上で達成する（$\|\mathbf{g}_t\| \le G$ のとき。演習問題 4）。一方、対角 AdaGrad のリグレットは $D_\infty \sum_{i=1}^{d} \sqrt{\sum_{t=1}^{T} g_{t,i}^2}$ であり、ここで D_∞ は ℓ_∞ ノルムにおける集合の直径である (Duchi et al. 2011)。ボックス $[-1, 1]^d$ とワンホット勾配を考え、各ステップで独立に選ばれた 1 つの座標 i（選ばれる確率を p_i とする）が $+1$ または -1 の勾配を受け取り、他のすべての座標は 0 を受け取るものとする。
   1. 両方の上界を計算せよ。AdaGrad については、各平方根の期待値をイェンセンの不等式で上から評価し、勾配降下法の上界に対する AdaGrad の上界の比が高々 $\sum_i \sqrt{p_i}/\sqrt{d}$ であることを示せ。
   2. 一様な p_i の場合、および Zipf 頻度 $p_i \propto i^{-\alpha}$（$d = 10^4$、$\alpha \in \{1, 2, 3\}$）の場合についてこの比を求めよ。
   3. 自然言語テキストにおける単語頻度は $\alpha \approx 1$ の Zipf 則に従い、そのときの比は約 0.63 となる。Bag-of-Words モデルは、係数 $\sqrt{d}$（35.5.1 節）の利得のうちどれだけを実際に得られるか。また、その利得をすべて得るのはどのような勾配分布のときか。
3. **手計算による Reddi の反例**。$C = 4$、$\beta_1 = 0$、命題の設定である $\beta_2 = 1/(1+C^2) = 1/17$ とし、`online` と同様にバイアス補正なしとする。
   1. 固定ステップ η、$[-1, 1]$ の内部にある反復点、および周期パターンに落ち着いた v_t を仮定する。v_a を v が $+C$ のステップ直後にとる値とし、3 ステップの漸化式を解いて v_a を閉じた形で求め、本文中の周期的な値 15.12、1.83、1.05 と 3 回の変位を導出せよ。また SGD の 3 回のステップ $\eta g_t/C$ を計算せよ。Adam は SGD と比べてどのステップを拡大しているか、またそれはなぜか。
   2. AMSGrad は v_t を $\tilde{v}_t = \max(\tilde{v}_{t-1}, v_t)$ に置き換える。実効ステップ $\eta_t/\sqrt{\tilde{v}_t}$ が、t に関して η_t が非増加である限り非増加となることを示せ。周期状態における 3 回の変位を計算し、正味の変位が $-0.51\,\eta$ となることを示せ。
   3. 命題では $\eta < \sqrt{1-\beta_2}$ という条件がステップ $\eta_t = \eta/\sqrt{t}$ に対して必要であった。これがどこで使われるかを示せ。すなわち、$x = 1$ から周期を開始し、$v \ge (1-\beta_2)C^2$ が $+C$ の勾配の直後に成り立つことを用いて、周期の最初のステップが $(0, 1)$ 内に着地し、射影が作用しないことを示せ。
4. **オンライン勾配降下法のリグレット**。●射影付きオンライン勾配降下法は $\mathbf{x}_{t+1} = \Pi_{\mathcal{C}}(\mathbf{x}_t - \eta_t \mathbf{g}_t)$（ただし $\mathbf{g}_t = \nabla f_t(\mathbf{x}_t)$）を実行する。ここで $\Pi_{\mathcal{C}}$ は閉じた凸集合 $\mathcal{C}$（直径 D）へのユークリッド射影であり、$\|\mathbf{g}_t\| \le G$、かつステップサイズ η_t は非増加であるとする

(Zinkevich 2003)。

1. リグレットが次式を満たすことを証明せよ。

$$\sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{C}} \sum_{t=1}^{T} f_t(\mathbf{x}) \;\le\; \frac{D^2}{2\eta_T} + \frac{G^2}{2}\sum_{t=1}^{T}\eta_t. \tag{35.60}$$

   $f_t(\mathbf{x}_t) - f_t(\mathbf{x}) \le \mathbf{g}_t^\top(\mathbf{x}_t - \mathbf{x})$ から出発し、射影が $\mathcal{C}$ の各点への距離を増加させないことを用いよ。
2. $\eta_t = D/(G\sqrt{t})$ とするとリグレットが高々 $\frac{3}{2}DG\sqrt{T}$ となり、一方で $\eta_t = c/t$ とすると上界が T に関して（任意の $c > 0$ に対して）線形に増大することを示せ。
3. これら 2 つの上界を演習問題 1c の減衰率と組み合わせ、AdaGrad が $\sqrt{\mathbf{s}_t}$ で割り、$\mathbf{s}_t$ では割らない理由を説明せよ。

5. [code] **回転感度**。●`rotated`、`run`、および `#adaptive-stochastic-methods-adam-vs-gd` セルの更新関数ファクトリを用いよ。
   1. $\mathbf{A}_\theta$ 上で $\mathbf{R}_\theta(1,1)^\top$ から出発した勾配降下法およびヘビーボール法の反復点が、軸に沿った谷におけるそれぞれの反復点に $\mathbf{R}_\theta$ を掛けたものになり、したがってステップ数が θ に依存しないことを示せ。
   2. $\beta_1 = 0$、$\theta = \pi/4$ において、f の 40,000 ステップ後の値を表のグリッドの各 η について出力し、本文の値 $250\,\eta^2$ と比較せよ。勾配降下法に対する Adam の優位性のうち、回転後も残る部分をもたらしているのは Adam のどの要素か。
   3. $\theta = \pi/16$ における Adam の座標ごとの実効ステップを出力せよ。これほど小さな回転であっても、軸に沿った谷で Adam が築いたステップサイズ比の大半が失われるのはなぜか。
   4. 演習問題 4（35.1 節）では、$-\operatorname{sign}(\mathbf{g})$ が ℓ_∞ ノルムのもとでの最急降下方向であり、$-\mathbf{U}\mathbf{V}^\top$ がスペクトルノルムのもとでの最急降下方向（ただし $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top$）であることを確認した。$\beta_1 = \beta_2 = 0$ かつ $\epsilon = 0$ の Adam が $-\operatorname{sign}(\mathbf{g}_t)$ の方向にステップを進めることを示せ。続いて $\mathbf{Q}_1$ と $\mathbf{Q}_2$ を直交行列とし、写像 $\mathbf{G} \mapsto \mathbf{U}\mathbf{V}^\top$ が $\mathbf{G} \mapsto \mathbf{Q}_1\mathbf{G}\mathbf{Q}_2$ と可換であることを示すとともに、符号写像が可換にならない 2×2 の例を 1 つ挙げよ。この結果は、Adam と比較した Muon の回転感度について何を示唆しているか。

6. [extended] **AdamW の暗黙の ℓ_∞ 制約**。●●35.5.3 節の命題から、AdamW の不動点はボックス $|w_i| < 1/\lambda$ 内にあり、$\epsilon \to 0$ では勾配がゼロでないすべての座標でその境界上に乗ることがわかった。Xie and Li (2024) は、$\beta_1 \le \beta_2$ かつ和が発散する非増加ステップスケジュールをもつフルバッチ AdamW が収束するならば、その極限は $\min_{\mathbf{w}} \ell(\mathbf{w})$（制約 $\|\mathbf{w}\|_\infty \le 1/\lambda$）の KKT 点であることを証明している。
   1. 実行可能な $\mathbf{w}$ がこの問題の KKT 点となる（一般的な条件の導出は 35.6 節を参照）のは、すべての座標が $\partial_i \ell(\mathbf{w}) = 0$（$|w_i| < 1/\lambda$ のとき）および $w_i\,\partial_i\ell(\mathbf{w}) \le 0$（$|w_i| = 1/\lambda$ のとき）を満たす場合である。このような点では、いかなる実行可能方向も一次近似において ℓ を減少させないことを示せ。
   2. 定理の設定である $\epsilon = 0$ を AdamW の条件 Equation 35.58 に代入せよ。$\partial_i\ell \neq 0$ である座標が KKT 条件を満たすことを示せ。勾配がゼロになる座標に対して、この条件だけでは $|w_i| \le 1/\lambda$ を示せないのはなぜか。
   3. $n = 100$、$d = 5$、標準正規分布に従う特徴量、およびラベル $y_i = \operatorname{sign}(\mathbf{x}_i^\top \mathbf{w}_{\text{true}})$ をもつ線形分離可能なロジスティック回帰のデータを生成せよ（正則化なしの損失は有限の最小点をもたない）。フルバッチ AdamW を $\epsilon = 10^{-16}$ で 5×10^4 ステップにわたり、η をコサインスケジュールで 10^{-2} から 0 まで減衰させて $\lambda \in \{0.05, 0.1, 0.2\}$ のそれぞれで実行せよ。得られた結果を、射影勾配降下法（ここでは各勾配ステップの後にすべての座標を $[-1/\lambda, 1/\lambda]$ にクリップする）で求めたボックス制約付き問題の解、および $\ell + \frac{\lambda}{2}\|\mathbf{w}\|^2$ 上での Adam の解と比較せよ。
   4. サイズ 10 および 50 のミニバッチ勾配を用いて AdamW の実行を繰り返せ。反復点はボックスに対してどこに落ち着くか、またそれはなぜか。

5. これらの結果から、AdamW において λ は何を制御しているといえるか。またそれは、ガウス事前分布に基づく λ の解釈（36.6 節）とどう異なるか。

7. **クロネッカー恒等式とその計算コスト**。列方向の積み重ね（列スタック）として vec を用いよ。
   1. $\operatorname{vec}(\mathbf{DVB}) = (\mathbf{B}^{\top} \otimes \mathbf{D})\operatorname{vec}(\mathbf{V})$ を積の第 j 列を書き下すことで証明し、さらに $(\mathbf{B} \otimes \mathbf{D})(\mathbf{B}' \otimes \mathbf{D}') = (\mathbf{BB}') \otimes (\mathbf{DD}')$ を証明せよ。
   2. 対称かつ正則な因子に対して Equation 35.59 を導け。因子が対称でない場合にはどの式に置き換わるか、また行方向に並べる（行スタックの）場合にはどの式になるか。
   3. $m \times n$ の層について、フルサイズの $mn \times mn$ 行列の逆行列を求めて適用するのに必要な演算回数と、2 つの因子の逆行列を求めて Equation 35.59 により適用する演算回数（$O(m^3 + n^3 + mn(m+n))$）を比較せよ。両方の回数を $m = n = 4096$ の場合、および $m = 4096$、$n = 16384$ の場合について評価せよ。
   4. 単一の勾配 $\mathbf{G} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\top}$ に対して $\mathbf{P}_{\mathrm{l}} = \mathbf{G}\mathbf{G}^{\top}$、$\mathbf{P}_{\mathrm{r}} = \mathbf{G}^{\top}\mathbf{G}$ と置く。Shampoo のステップ $\mathbf{P}_{\mathrm{l}}^{-1/4}\mathbf{G}\,\mathbf{P}_{\mathrm{r}}^{-1/4}$ が $\mathbf{UV}^{\top}$ に等しいことを示せ。$\mathbf{G}$ が正方行列でないとき、逆 4 乗根は何を意味する必要があるか、またその結果はどの特異ベクトルを残すか。

35.6 制約付き最適化

ここまでに最適化したパラメータはすべて任意の実数値をとることができたため、最小点を見つけるとは勾配がゼロになる点を探すことであった。しかし、深層学習に現れる多くの量にはそのような自由度がない。アテンション重みは非負で和が 1 にならなければならず、勾配クリッピングは更新のノルムを閾値以下に抑え、ノルムバジェットは重みベクトルの大きさを制限し、サポートベクターマシンはすべての訓練マージンが少なくとも 1 であることを要求する。変数が実行可能領域（feasible set）の内部に留まらなければならないとき、最適点において勾配がゼロになることはめったになく、前節までのゼロ勾配による判定は破綻する。制約のもとにある最適点をどのように見分け、どのように計算すればよいのだろうか。

勾配クリッピングは最も単純な具体例を与える。ベクトル $\mathbf{y} = (3, 4)$ を、そのノルムが高々 1 になるようにクリップしたいとしよう。単位長に再スケーリングすると $(0.6, 0.8)$ が返り、これは単位円板上で $\mathbf{y}$ に最も近い点である。言い換えれば、$(0.6, 0.8)$ は二乗距離 $f(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ を円板 $\|\mathbf{x}\| \le 1$ 上で最小化し、最適値 $f = 8$ を達成する。ところが、その点での勾配は $(-2.4, -3.2)$ であり、ゼロにはほど遠い。$(0.6, 0.8)$ を最適にしているのは、**許容される**いかなる方向も f を減少させないという事実である。すなわち、$\|\mathbf{x}\| \le 1$ を保つあらゆる移動は、1 次の範囲で f を増加させるか変化させないかのいずれかである。このような問題を扱うには、ゼロ勾配条件に代わる基準と、そのままでは実行可能領域の外へ飛び出してしまう素の勾配降下法に代わる手法の両方が必要になる。

「勾配がゼロである」に代わる条件では、制約ごとに**乗数**（multiplier）と呼ばれる 1 つのスカラーを導入し、最適点がその制約をどれほど強く押しているかを測る。点 $(0.6, 0.8)$ では、境界が外向き法線に沿って強さ 4 で押し返すため、$(-2.4, -3.2) + 4\,(0.6, 0.8) = \mathbf{0}$ が成り立つ。乗数を用いると一階最適性条件（first-order optimality conditions）が得られる。すなわち、等式制約に対するラグランジュ条件と、不等式制約に対するカルーシュ・キューン・タッカー（KKT）条件である。一方、$(3, 4)$ を $(0.6, 0.8)$ へと写した射影（projection）からは、直接的なアルゴリズムが示唆される。通常の勾配ステップを進めたあと、最も近い実行可能点へ射影して戻せばよい。さらに、硬い境界が折れ目を持つペナルティ（たとえば Lasso が各重みに課す絶対値ペナルティ）に置き換わるとき、射影は**近接ステップ**（proximal step）へと一般化され、小さな重みを厳密なゼロへと送り込む。本節全体を通して、凸性（35.3 節）こそがこれらの条件を大域的最適性の保証へと変え、アルゴリズムの収束を保証する鍵となる。

本節を支える柱は 3 つある。KKT 条件とその凸問題における十分性、射影を特徴づける鈍角不等

式（obtuse-angle inequality）、そしてソフト閾値処理（soft-thresholding）である。

35.6.1 等式制約とラグランジュ乗数

35.6.1.1 接条件とラグランジュ条件

等式制約は不等式制約よりも単純であるため、まずはクリッピングの例の中身の詰まった円板をその境界に置き換えるところから始めよう。すなわち、$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$（ただし $\mathbf{y} = (3, 4)$）を、$h(\mathbf{x}) = \frac{1}{2}(\|\mathbf{x}\|^2 - 1) = 0$ と表される単位円上で最小化する。より一般に、微分可能な目的関数 $f : \mathbb{R}^n \to \mathbb{R}$ を 1 つの等式制約のもとで最小化する問題を考える。

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad h(\mathbf{x}) = 0.$$

実行可能領域は全次元の領域ではなく曲面（2 次元では曲線）であるため、実行可能な移動はすべてその曲面上に留まらなければならない。

Figure 35.20 に最適点の幾何学的な特徴を示す。いま、f の等位集合（level set）が円と**交差する**実行可能点に立っているとしよう。このとき ∇f は円に沿った非ゼロの成分を持つため、その成分と逆向きに円に沿って滑るように動けば、実行可能領域から出ることなく f を減らせる。したがって、そのような点は最適点ではあり得ない。最適点が現れうるのは等位集合が円に**接する**場所だけであり、そこでは円に沿ったどの移動も 1 次の範囲で f を変化させない。互いに接する曲線は平行な法線ベクトルを共有するため、最適点において ∇f は ∇h のスカラー倍でなければならない。

命題（ラグランジュ条件）$\mathbf{x}^\star$ を f の $\{\mathbf{x} : h(\mathbf{x}) = 0\}$ 上における極小点とする。f は $\mathbf{x}^\star$ で微分可能であり、h は $\mathbf{x}^\star$ の近傍で連続微分可能であり、かつ $\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$ であるとする。このとき、一意なスカラー $\nu^\star$（ラグランジュ乗数）が存在して次式を満たす。

$$\nabla f(\mathbf{x}^\star) + \nu^\star \, \nabla h(\mathbf{x}^\star) = \mathbf{0}. \tag{35.61}$$

言葉で言えば、目的関数の勾配は制約の法線ベクトルの定数倍と釣り合っている。これは 34.2.4 節で証明したラグランジュ条件と同じものであり、そこでは制約を $g(\mathbf{x}) = c$、条件を $\nabla f = \mu \nabla g$ と書いていた。$h = g - c$ および $\nu = -\mu$ とおけば 2 つの形式は一致する。ここでプラスの符号を用いるのは、次に導入するラグランジアンの符号と揃えるためである。

接点 $(0.6, 0.8)$ を Figure 35.20 で確認すると、$\nabla f = \mathbf{x} - \mathbf{y} = (-2.4, -3.2)$ かつ $\nabla h = \mathbf{x} = (0.6, 0.8)$ である。この 2 つのベクトルは実際に平行で $\nabla f = -4 \, \nabla h$ を満たしており、これから $\nu^\star = 4$ が得られる。

仮定 $\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$ は**制約想定**（constraint qualification）の最初の例であり、単なる技術的な細目ではなく本質的な条件である。$f(\mathbf{x}) = x_1$ を制約 $h(\mathbf{x}) = x_1^2 + x_2^2 = 0$ のもとで最小化することを考えよう。実行可能領域は $\mathbf{0}$ のただ 1 点のみからなり、これが自動的に最小点となる。ところが $\nabla f(\mathbf{0}) = (1, 0)$ である一方 $\nabla h(\mathbf{0}) = \mathbf{0}$ であるため、いかなる乗数 $\nu^\star$ も Equation 35.61 を満たすことができない。実行可能領域が接線方向を持たない 1 点へと潰れてしまっているため、接条件の議論が足掛かりを失うのである。

35.6.1.2 ラグランジアン

ここで、$\mathbf{x}$ と ν という 2 つの未知数と、勾配が平行であることおよび実行可能性という 2 つの要請があることに注目しよう。これら両方の要請は、$(\mathbf{x}, \nu)$ の 1 つの関数の停留性へとまとめるこ

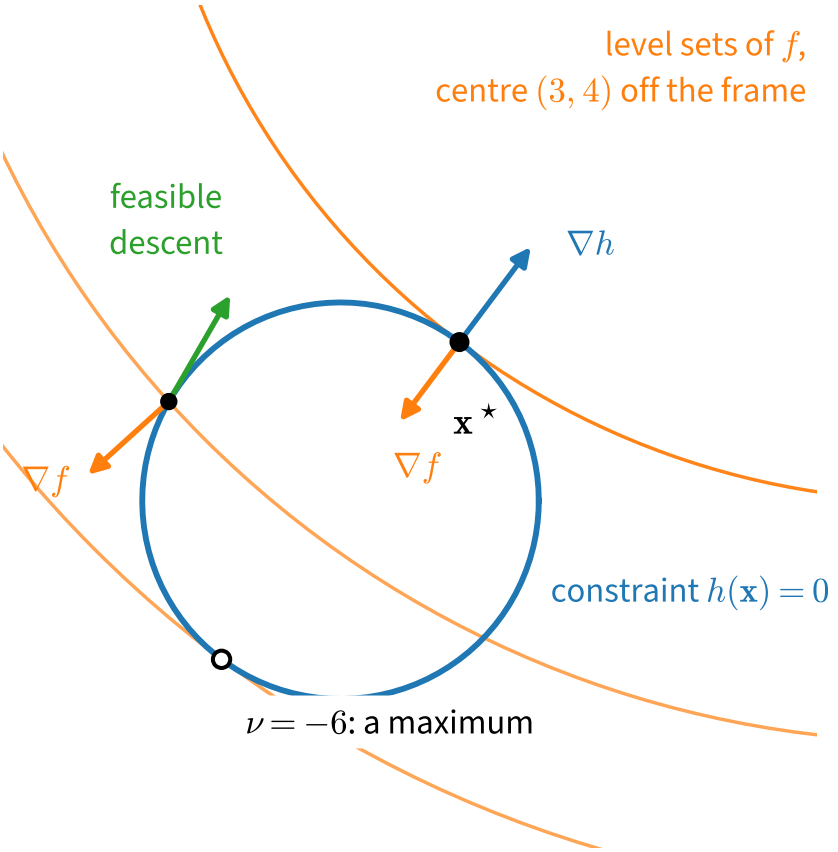


Figure 35.20: $f = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$ ($\mathbf{y} = (3, 4)$) の単位円 $h(\mathbf{x}) = 0$ 上での最小化。等位集合が円と交差する場所では、∇f が制約に沿った成分を持ち、実行可能な降下方向を与える。$\mathbf{x}^\star = (0.6, 0.8)$ では等位集合が円に接し、$\nabla f = -4\,\nabla h$ となる。対蹠点 $(-0.6, -0.8)$ も $\nu = -6$ で接条件を満たし、最大点となる。

とができる。この関数は**ラグランジアン**と呼ばれ、乗数で重み付けした制約を目的関数に加えたものである。

$$\mathcal{L}(\mathbf{x}, \nu) = f(\mathbf{x}) + \nu\, h(\mathbf{x}). \tag{35.62}$$

$\mathcal{L}$ のすべての偏微分をゼロとおくと次式が得られる。

$$\nabla_{\mathbf{x}} \mathcal{L} = \nabla f + \nu \nabla h = \mathbf{0} \qquad \text{and} \qquad \frac{\partial \mathcal{L}}{\partial \nu} = h(\mathbf{x}) = 0,$$

これにより、Equation 35.61 と等式制約が同時に復元される。このようにして、$\mathbf{x}$ に関する制約付き最適化問題は、$(\mathbf{x}, \nu)$ に関する制約なしの停留問題へと帰着する。

ラグランジアンを使って単位円上の問題を解いてみよう。$\mathbf{x}$ に関する停留性から $\nabla_{\mathbf{x}}\mathcal{L} = (\mathbf{x} - \mathbf{y}) + \nu\mathbf{x} = \mathbf{0}$ となるため、$\mathbf{x} = \mathbf{y}/(1+\nu)$ は原点と $\mathbf{y}$ を通る直線上になければならない。さらに実行可能性 $\|\mathbf{x}\| = 1$ を課すと $|1+\nu| = \|\mathbf{y}\| = 5$ が必要となり、次の 2 つの解が得られる。

- $\nu = 4 : \mathbf{x} = (0.6, 0.8)$ であり、距離 4 だけ $\mathbf{y}$ から離れ、$f = 8$ となる。
- $\nu = -6 : \mathbf{x} = (-0.6, -0.8)$ であり、距離 6 だけ $\mathbf{y}$ から離れ、$f = 18$ となる。

2 番目の解は、円上で**最も遠い**点（$\mathbf{y}$ からの距離が最大となる点）である。ここから重要な注意点が浮かび上がる。ラグランジュ条件は必要条件であって十分条件ではない。というのも、曲面上の f の極大点や鞍点を含むすべての停留点で成り立つからである。この例では、乗数の符号が最小点と最大点を区別している。次節で不等式制約へ進むと、KKT 条件がこの符号を一般的な規則へと昇華させる。

複数の等式制約 $h_j(\mathbf{x}) = 0$（$j = 1, \dots, p$）がある場合は、各制約にそれぞれ乗数を割り当て、ラグランジアンは $\mathcal{L} = f + \sum_j \nu_j h_j$ となる。制約の勾配 $\nabla h_j(\mathbf{x}^\star)$ が線形独立である限り、同じ議論により $\nabla f(\mathbf{x}^\star)$ は制約の法線が**張る空間**（span）に属する。すなわち、目的関数の勾配は制約の勾配の線形結合と釣り合う。

35.6.1.3 超平面上の最近接点

線形制約を持つ最も単純な制約付き問題は、超平面上で原点に最も近い点を求める問題である。すなわち、$f(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{x}\|^2$ を制約 $h(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} - b = 0$ のもとで最小化する。$\mathcal{L} = \tfrac{1}{2}\|\mathbf{x}\|^2 + \nu(\mathbf{a}^\top\mathbf{x} - b)$ の勾配をゼロとおくと $\mathbf{x} + \nu\mathbf{a} = \mathbf{0}$ となるため、最近接点は法線ベクトル $\mathbf{a}$ の方向になければならない。$\mathbf{x} = -\nu\mathbf{a}$ を制約に代入すると $-\nu\|\mathbf{a}\|^2 = b$ となり、$\nu^\star = -b/\|\mathbf{a}\|^2$ と定まる。したがって、解と最適値は次のようになる。

$$\mathbf{x}^\star = \frac{b}{\|\mathbf{a}\|^2}\,\mathbf{a}, \qquad p^\star = \frac{b^2}{2\,\|\mathbf{a}\|^2}.$$

幾何学的には、最近接点は距離 $b/\|\mathbf{a}\|$ だけ単位法線 $\mathbf{a}/\|\mathbf{a}\|$ に沿って進んだ位置にあり、33.1 節の射影の公式と一致する。

$\mathbf{a} = (3, 4)$、$b = 5$ とすると、再び $(0.6, 0.8)$ に到達し、今度は $\nu^\star = -0.2$、$p^\star = \tfrac{1}{2}$ となる。直線 $3x_1 + 4x_2 = 5$ は点 $(0.6, 0.8)$ で単位円に接している。ここで、乗数が単なる代数計算の補助にとどまらないことに注目しよう。b を 5 から 5.1 へとわずかに動かして直線を原点から少し遠ざけると、最適値は約 $0.02 = -\nu^\star \cdot 0.1$ だけ増加する。35.7.3 節で一般の場合について示すように、乗数が常に測っているのはまさにこの量、すなわち制約をずらしたときの最適値の感度である。

35.6.2 不等式制約と KKT 条件

35.6.2.1 有効制約と非有効制約

機械学習に現れる制約の大半は片側のみを制限するものである。たとえば、ノルムが高々 r であること、確率が少なくとも 0 であること、あるいはリソースバジェットを超えないことなどである。不等式制約は、次の標準形で書く。

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \le 0, \; i = 1, \dots, m,$$

このように書くと、各 g_i は領域の真の内部で負、境界上でゼロとなる。解 $\mathbf{x}^\star$ において、すべての不等式は 2 つの状態のいずれかに分類される。$g_i(\mathbf{x}^\star) < 0$ ならば、その制約は**非有効**（inactive）である。解は領域の真の内部にあるため、局所的にはその制約は何の影響も及ぼさない。一方、$g_i(\mathbf{x}^\star) = 0$ ならば、その制約は**有効**（active）である。解はまさに境界上に位置し、そこで不等式は等式のように働く。

勾配クリッピングは、この両方の状態を明快に示してくれる。ベクトル $\mathbf{y}$ を半径 r のユークリッド球へ射影するため、$f(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ を制約 $g(\mathbf{x}) = \tfrac{1}{2}(\|\mathbf{x}\|^2 - r^2) \le 0$ のもとで最小化するとしよう。乗数 λ を導入すると、$f + \lambda g$ の停留性から $(\mathbf{x} - \mathbf{y}) + \lambda \mathbf{x} = \mathbf{0}$ が得られるため、$\mathbf{x} = \mathbf{y}/(1 + \lambda)$ は常に $\mathbf{y}$ を動径方向に再スケーリングした点となる。どちらの状態になるかは $\|\mathbf{y}\|$ によって決まる。

- $\|\mathbf{y}\| \le r$ のとき、$\mathbf{y}$ はすでに実行可能であり、そのまま f を最小化する。制約は非有効で、$\lambda^\star = 0$ かつ $\mathbf{x}^\star = \mathbf{y}$ となる。たとえば $\mathbf{y} = (0.3, 0.4)$ は $r = 1$ のときそのまま変化しない。
- $\|\mathbf{y}\| > r$ のとき、内点が最適になることはあり得ない。内点が最小点であるためには f の制約なし最小点でなければならないが、唯一の制約なし最小点は実行不可能な点 $\mathbf{y}$ だからである。したがって制約は必ず有効となり、$\|\mathbf{x}^\star\| = r$ から $\lambda^\star = \|\mathbf{y}\|/r - 1 > 0$ が定まる。$\mathbf{y} = (3, 4)$、$r = 1$ のときは、$\lambda^\star = 4$、$\mathbf{x}^\star = (0.6, 0.8)$、および $p^\star = \tfrac{1}{2} \cdot 4^2 = 8$ が得られる。

両方のケースは 1 つの式にまとまる。

$$\mathbf{x}^\star = \frac{\mathbf{y}}{\max\left(1, \; \|\mathbf{y}\|/r\right)},$$

この式は、短いベクトルはそのまま残し、長いベクトルはノルム r に再スケーリングする。これを勾配ベクトルに適用したものが**勾配クリッピング**である (Pascanu et al. 2013)。コード上では、この場合分けは 1 回の `max` に集約でき、クリップされた点において f の勾配が $\lambda^\star$ 倍の制約勾配 $\nabla g(\mathbf{x}) = \mathbf{x}$ によって打ち消されることを直接確認できる。

```
y, r = np.array([3.0, 4.0]), 1.0
lam = max(np.linalg.norm(y) / r - 1, 0.0)    # multiplier: 0 if y is inside the ball
x = y / (1 + lam)                            # x* = y / max(1, |y|/r)
print('x* =', x, ' lambda* =', lam, ' f(x*) =', 0.5 * ((x - y)**2).sum())
print('grad f(x*) =', x - y, ' lambda* grad g(x*) =', lam * x,
      ' sum =', (x - y) + lam * x)
```

```
x* = [0.6 0.8]  lambda* = 4.0  f(x*) = 8.0
grad f(x*) = [-2.4 -3.2]  lambda* grad g(x*) = [2.4 3.2]  sum = [0. 0.]
```

∇f がゼロからほど遠くても停留性は成り立っている。目的関数の勾配 $(-2.4, -3.2)$ と境界からの押し返し $(2.4, 3.2)$ の和がゼロになり、乗数 $\lambda^\star = 4$ が両者の長さの比を与えている。

乗数の符号が持つ幾何学的な意味に注目しよう。球の外へ出ると $g(\mathbf{x})$ は増加するため、その勾配 $\nabla g = \mathbf{x}$ は外向きを指す。停留性を $-\nabla f = \lambda \nabla g$（$\lambda > 0$）と書くと、下り方向 $-\nabla f$ が外向きに引っ張る一方で、境界の壁がそれを押し返していることがわかる。仮に $-\nabla f$ が内側を向いていたとすれば、球の内部へ踏み出すことで f を下げられるため、その点は最適ではあり得ない。したがって負の乗数は排除される。実際、$\lambda = -6$ とすると円上の遠い側の点 $(-0.6, -0.8)$ が選ばれ、これは円板上での最小点ではなく最大点になる。等式制約とは異なり、不等式の壁は 1 つの方向にしか押し返さないのである。

35.6.2.2 カルーシュ・キューン・タッカー条件

不等式制約と等式制約を組み合わせると、次の一般的な最適化問題が得られる。

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \le 0,\; i = 1, \dots, m, \qquad h_j(\mathbf{x}) = 0,\; j = 1, \dots, p. \tag{35.63}$$

この問題の**ラグランジアン**は各制約に 1 つずつ乗数を割り当て、不等式制約の乗数には非負であることを課す。

$$\mathcal{L}(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \, g_i(\mathbf{x}) + \sum_{j=1}^{p} \nu_j \, h_j(\mathbf{x}), \qquad \lambda_i \ge 0. \tag{35.64}$$

制約想定（具体的にどの条件を用いるかは次項で述べる）が成り立つ任意の極小点 $\mathbf{x}^\star$ において、次の 4 つの**カルーシュ・キューン・タッカー（KKT）条件**を満たす乗数が存在する (Karush 1939; Kuhn and Tucker 1951)。

$$\begin{aligned}
&\text{stationarity:} && \nabla f(\mathbf{x}^\star) + \textstyle\sum_i \lambda_i^\star \nabla g_i(\mathbf{x}^\star) + \sum_j \nu_j^\star \nabla h_j(\mathbf{x}^\star) = \mathbf{0}, \\
&\text{primal feasibility:} && g_i(\mathbf{x}^\star) \le 0, \quad h_j(\mathbf{x}^\star) = 0, \\
&\text{dual feasibility:} && \lambda_i^\star \ge 0, \\
&\text{complementary slackness:} && \lambda_i^\star \, g_i(\mathbf{x}^\star) = 0 \quad \text{for every } i.
\end{aligned} \tag{35.65}$$

最初の 3 つの条件、すなわち勾配の釣り合い（停留性）、制約そのものの成立（主実行可能性）、不等式乗数に対する片方向の符号制限（双対実行可能性）は、これまでの例ですでに見てきたとおりである。4 つ目の**相補性条件**（complementary slackness）は、各 i について $\lambda_i^\star$ と $g_i(\mathbf{x}^\star)$ の少なくとも一方がゼロになることを述べている。その結果、すべての非有効制約（$g_i(\mathbf{x}^\star) < 0$）は自動的に $\lambda_i^\star = 0$ となり、停留性の式から脱落する。ただし、微妙な非対称性に留意しておこう。乗数が正であればその制約は確実に有効であるが、逆に有効制約であっても、たまたま力を及ぼさずに $\mathbf{x}^\star$ に接しているだけならば $\lambda_i^\star = 0$ となりうる。球への射影の例では、相補性条件は手計算で行った場合分けをそのまま符号化しており、点 $(0.6, 0.8)$ では $\lambda^\star g(\mathbf{x}^\star) = 4 \cdot 0 = 0$ が成り立っている。

Figure 35.21 は、停留性と双対実行可能性を合わせた幾何学的構造を示している。最適点において、降下方向 $-\nabla f$ は有効制約の外向き法線の非負線形結合でなければならない。言い換えれば、$-\nabla f$ はそれらの外向き法線が張る錐（**法線錐**（normal cone））の内部に含まれ、非有効制約は一切寄与しない。

ラグランジュ条件と同様、KKT 条件は一階条件であり、それ単独では十分条件にならない。$f(x) = -x^2$ を $[-1, 1]$ 上（すなわち $x - 1 \le 0$ かつ $-1 - x \le 0$）で最小化することを考え

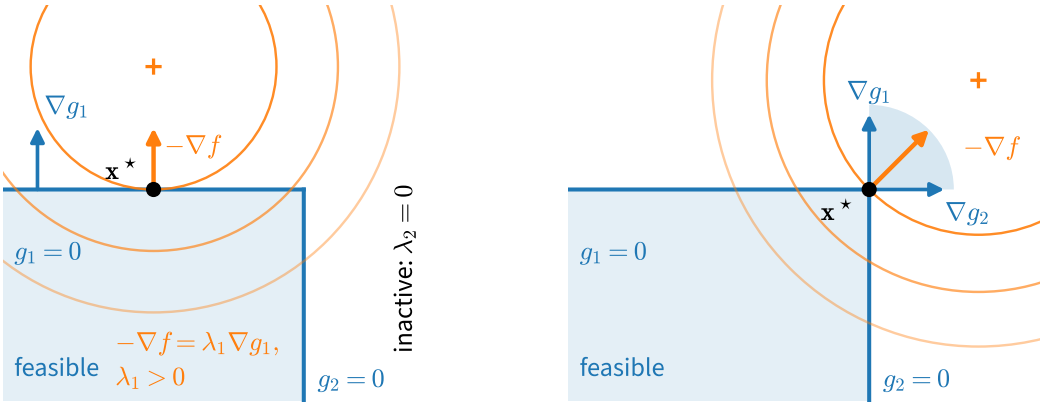


Figure 35.21: 2 つの不等式制約で定義された実行可能領域における KKT 条件の幾何構造。(a) では、一方の制約が $\mathbf{x}^\star$ で有効、もう一方が非有効であり、$-\nabla f$ は $\lambda_1 > 0$ で有効な外向き法線と同じ向きに揃い、$\lambda_2 = 0$ となる。(b) では、角の点で両方の制約が有効となるため、$-\nabla f$ は両方の外向き法線が張る網掛けの法線錐の内部になければならない。オレンジ色の弧は、f の制約なし最小点 (+) のまわりの等位集合を表す。

よう。$x = 0$ では勾配がゼロになり、両方の制約が非有効で、両方の乗数をゼロとおけば 4 つの KKT 条件すべてを満たすが、$x = 0$ は**最大点**である。真の最小点である $x = \pm 1$ も KKT 点であり、それぞれ有効な境界制約に対して乗数 2 を持つ。

35.6.2.3 制約想定と十分性

こうして、互いに補い合う 2 つの問いが残る。第 1 に、極小点が KKT 条件を満たすことが**保証される**のはいつか。第 2 に、KKT 点が最小点であることが**保証される**のはいつか。

1 つ目の問いに対する答えは、等式制約の場合と並行している。すなわち、制約想定が成り立つ限り、極小点は必ず KKT 条件を満たす。最もよく使われる一般的な条件は**線形独立制約想定**（linear independence constraint qualification、LICQ）である。$\mathbf{x}^\star$ を Equation 35.63（ただし f、g_i、h_j は連続微分可能）の極小点としよう。すべての有効な不等式制約の勾配とすべての等式制約の勾配を合わせた集合 $\{\nabla g_i(\mathbf{x}^\star) : g_i(\mathbf{x}^\star) = 0\} \cup \{\nabla h_j(\mathbf{x}^\star) : j = 1, \dots, p\}$ が線形独立であれば、Equation 35.65 を満たす乗数の存在が保証される（Nocedal and Wright (2006) 第 12 章）。LICQ は $\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$ の直接的な一般化であり、これが破れたときに何が起きうるかは演習問題 1 で確認する。

凸問題に対しては、未知の解における勾配を調べる代わりに、1 つの実行可能点を見つけるだけで済むはるかに単純な判定条件がある。問題 Equation 35.63 は、f とすべての g_i が凸関数であり、すべての h_j がアフィン関数であるとき**凸**であるという。凸問題において**スレーター条件**（Slater's condition）(Slater 1950) が成り立つとは、次を満たす点 $\bar{\mathbf{x}}$ が存在することをいう。

$$\begin{aligned} g_i(\bar{\mathbf{x}}) &< 0 \quad \text{for every } g_i \text{ that is not affine}, \\ g_i(\bar{\mathbf{x}}) &\leq 0 \quad \text{for every affine } g_i, \\ h_j(\bar{\mathbf{x}}) &= 0 \quad \text{for every } j. \end{aligned} \tag{35.66}$$

平たく言えば、$\bar{\mathbf{x}}$ は実行可能であり、かつ曲がったすべての不等式制約の真の内部になければならない。すべての制約がアフィンであれば、単に実行可能であるだけでスレーター条件は満た

される。（ここでは扱わないが、f またはいずれかの g_i が $\mathbb{R}^n$ の部分集合上でのみ定義されている場合は、$\bar{\mathbf{x}}$ はその定義域の相対的内部にも属していなければならない。）ユークリッド球の場合は、$\bar{\mathbf{x}} = \mathbf{0}$ とすれば $g(\mathbf{0}) = -r^2/2 < 0$ となるため直ちに条件を満たす。微分可能な f と g_i を持つ凸問題でスレーター条件が成り立つとき、すべての最小点は KKT 点となる (Boyd and Vandenberghe 2004)。その背後にあるより深い理由は強双対性（strong duality）であり、35.7.2 節ではまさに同じ条件からこれを導く。

凸性は 2 つ目の問いにも決着をつけ、本節で最も有用な結果をもたらす。

命題（凸性のもとでの KKT 条件の十分性） 問題 Equation 35.63 において、f とすべての g_i が凸かつ微分可能であり、すべての h_j がアフィンであるとする。$(\mathbf{x}^\star, \lambda^\star, \nu^\star)$ が KKT 条件 Equation 35.65 を満たすならば、$\mathbf{x}^\star$ は大域的最小点である。

証明： $\varphi(\mathbf{x}) = \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star)$ とおく。$\lambda_i^\star \geq 0$ であるから、φ は凸関数の非負結合にアフィン項を加えたものであり、したがって φ は凸関数である。停留性より $\nabla \varphi(\mathbf{x}^\star) = \mathbf{0}$ であるから、$\mathbf{x}^\star$ は**大域的な**最小点（φ の $\mathbb{R}^n$ 全体における最小点）となる。ここで任意の実行可能点 $\mathbf{x}$ をとる。$\lambda_i^\star \geq 0$、$g_i(\mathbf{x}) \leq 0$、および $h_j(\mathbf{x}) = 0$ より、次式が成り立つ。

$$f(\mathbf{x}) \;\geq\; f(\mathbf{x}) + \sum_i \lambda_i^\star g_i(\mathbf{x}) + \sum_j \nu_j^\star h_j(\mathbf{x}) = \varphi(\mathbf{x}) \;\geq\; \varphi(\mathbf{x}^\star) = f(\mathbf{x}^\star),$$

ここで最後の等号は、相補性条件（$\sum_i \lambda_i^\star g_i(\mathbf{x}^\star) = 0$）と主実行可能性（$h_j(\mathbf{x}^\star) = 0$）から従う。 ■

この短い証明の中に織り込まれている、より広い原理に注目しよう。**任意の**固定された乗数（$\lambda \geq 0$）に対して、すべての実行可能点 $\mathbf{x}$ で最初の不等式が成り立つため、ラグランジアンを $\mathbf{x}$ に関して最小化すれば、常に制約付き最適値の下界が得られる。そして KKT 乗数においては、その下界は厳密に一致する。35.7 節では、この考察を出発点として双対性（duality）の理論全体を構築する。したがって凸問題においては、KKT 条件は常に十分条件であり、さらにスレーター条件のもとでは必要条件にもなって大域的最小点を完全に特徴づける。

35.6.3 射影と射影勾配降下法

KKT 条件は最適点を**見分ける**方法を教えてくれ、単純な制約であればそれを使って手計算で解くこともできる。より一般的に解を**計算する**には、勾配降下法を最近接の実行可能点を求める操作と組み合わせる。すなわち、通常の勾配ステップを進め、実行可能領域の外へ出るたびにそのまま内側へ射影して戻すのである。

35.6.3.1 凸集合への射影

空でない閉凸集合 $C \subseteq \mathbb{R}^n$ に対し、**射影**とは点 $\mathbf{y}$ を C へ写す操作であり、C 内で $\mathbf{y}$ に最も近い唯一の点として次式で定義される。

$$\Pi_C(\mathbf{y}) = \operatorname*{argmin}_{\mathbf{x} \in C} \; \tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2.$$

なぜ唯一の最小点が常に存在するのだろうか。最適点を除外しないまま、C と $\mathbf{y}$ を中心とする十分大きな閉球との共通部分をとることができるからである。この共通部分は有界閉集合（したがってコンパクト）となるため、連続な目的関数はそこで最小値を達成する。さらに二乗距離の狭義凸性（strict convexity）によって複数の点が同値で並ぶ可能性が排除され、Π_C は一意に定まる写像となる。

すでに超平面への射影（最近接点問題）と球への射影（クリッピングの公式）という 2 つの射影を閉形式で求めた。半空間（half-space）$\{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \geq b\}$ への射影は、超平面の公式から直ちに従う。$\mathbf{y}$ がすでに不等式を満たしていればそのままにし、満たしていなければ $\mathbf{a}$ に沿って境界超平面上までまっすぐ移動する。

$$\Pi(\mathbf{y}) = \mathbf{y} + \frac{\max(0,\ b - \mathbf{a}^\top \mathbf{y})}{\|\mathbf{a}\|^2}\,\mathbf{a}.$$

閉凸集合への射影は、きわめて明快な幾何学的特徴づけを満たす。$\hat{\mathbf{x}} = \Pi_C(\mathbf{y})$ を $\tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ の C 上での最小点としよう。他の任意の点 $\mathbf{x} \in C$ に対し、凸性により $\hat{\mathbf{x}}$ から $\mathbf{x}$ までの線分全体が C の内部に収まる。したがって $\hat{\mathbf{x}}$ から $\mathbf{x}$ へ向かって動いても 1 次の範囲で目的関数は減少せず、方向微分は非負、すなわち $(\hat{\mathbf{x}} - \mathbf{y})^\top (\mathbf{x} - \hat{\mathbf{x}}) \geq 0$ となる。符号を反転させると、次の**鈍角不等式**（または変分不等式（variational inequality））が得られる。

$$(\mathbf{y} - \Pi_C(\mathbf{y}))^\top (\mathbf{x} - \Pi_C(\mathbf{y})) \ \leq\ 0 \qquad \text{for all } \mathbf{x} \in C. \tag{35.67}$$

幾何学的には、**残差**ベクトル $\mathbf{y} - \Pi_C(\mathbf{y})$ は少なくとも 90° の角を、$\Pi_C(\mathbf{y})$ から任意の点 $\mathbf{x} \in C$ へ向かうベクトルとの間になす（Figure 35.22 の (a)）。言い換えれば、$\Pi_C(\mathbf{y})$ を通り法線 $\mathbf{y} - \Pi_C(\mathbf{y})$ を持つ超平面は C の**支持超平面**（supporting hyperplane）であり、集合 C 全体をその反対側に保つ。

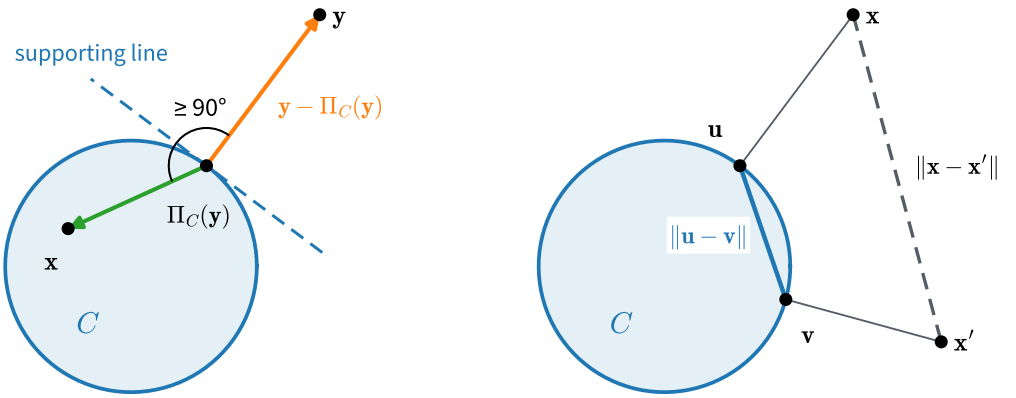


Figure 35.22: (a) 残差 $\mathbf{y} - \Pi_C(\mathbf{y})$ は少なくとも 90° の角を C の内部へ向かうあらゆる方向との間になすため、C は $\Pi_C(\mathbf{y})$ を通る支持直線の反対側に完全に収まる。(b) 閉凸集合への射影は距離を決して増加させない（$\|\mathbf{u} - \mathbf{v}\| \leq \|\mathbf{x} - \mathbf{x}'\|$）。

本節の共通の例でこの不等式を確かめてみよう。単位円板へ射影される $\mathbf{y} = (3, 4)$ に対して、残差は $(2.4, 3.2) = 4\,(0.6, 0.8)$ となり、またしても乗数と外向き法線の積が現れる。任意の $\mathbf{x}$（$\|\mathbf{x}\| \leq 1$）に対して、次式が成り立つ。

$$(2.4, 3.2)^\top (\mathbf{x} - (0.6, 0.8)) = 4\,(0.6 x_1 + 0.8 x_2) - 4 \ \leq\ 4\,\|\mathbf{x}\| - 4 \ \leq\ 0,$$

ここで最初の不等号にはコーシー・シュワルツの不等式 $\mathbf{u}^\top \mathbf{v} \leq \|\mathbf{u}\|\,\|\mathbf{v}\|$ を用いた。ここでの支持直線は $3x_1 + 4x_2 = 5$ であり、まさに先ほどの最近接点の例に現れた超平面そのものである。

同様に重要なこととして、鈍角不等式は**十分条件**でもある。ある点 $\hat{\mathbf{x}} \in C$ が Equation 35.67 を満たすとし、$\varphi(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ とおこう。その $\hat{\mathbf{x}}$ における勾配は $\hat{\mathbf{x}} - \mathbf{y}$ である。微分可能な凸関数は常にその接平面の上側にあるため（35.3 節）、$\varphi(\mathbf{x}) \geq \varphi(\hat{\mathbf{x}}) + (\hat{\mathbf{x}} - \mathbf{y})^{\top}(\mathbf{x} - \hat{\mathbf{x}})$ が成り立つ。仮定により最後の項はすべての $\mathbf{x} \in C$ に対して非負であるから、$\varphi(\mathbf{x}) \geq \varphi(\hat{\mathbf{x}})$ が C 全体で成り立ち、$\hat{\mathbf{x}} = \Pi_C(\mathbf{y})$ が証明される。したがって、Equation 35.67 は射影を**完全に特徴づける**。射影を用いたアルゴリズムに必要な性質はすべて、射影が決して距離を引き伸ばさないという事実をはじめとして、この不等式から直接導かれる。

命題（射影の非拡大性） 空でない閉凸集合 C とすべての $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ に対して、次式が成り立つ。

$$\|\Pi_C(\mathbf{x}) - \Pi_C(\mathbf{y})\| \;\leq\; \|\mathbf{x} - \mathbf{y}\|.$$

証明： $\mathbf{u} = \Pi_C(\mathbf{x})$ および $\mathbf{v} = \Pi_C(\mathbf{y})$ とおく。Equation 35.67 を 2 回、まず $\mathbf{x}$ に対して $\mathbf{v} \in C$ を用いて、次に $\mathbf{y}$ に対して $\mathbf{u} \in C$ を用いて適用すると、次式が得られる。

$$(\mathbf{x} - \mathbf{u})^{\top}(\mathbf{v} - \mathbf{u}) \leq 0, \qquad (\mathbf{y} - \mathbf{v})^{\top}(\mathbf{u} - \mathbf{v}) \leq 0.$$

この 2 つの不等式を足し合わせると $(\mathbf{x} - \mathbf{y} - (\mathbf{u} - \mathbf{v}))^{\top}(\mathbf{v} - \mathbf{u}) \leq 0$ となり、整理すると次式になる。

$$\|\mathbf{u} - \mathbf{v}\|^2 \;\leq\; (\mathbf{x} - \mathbf{y})^{\top}(\mathbf{u} - \mathbf{v}) \;\leq\; \|\mathbf{x} - \mathbf{y}\|\,\|\mathbf{u} - \mathbf{v}\|$$

ここで後半の不等号はコーシー・シュワルツの不等式による。両辺を $\|\mathbf{u} - \mathbf{v}\|$ で割れば証明が完了する（なお $\|\mathbf{u} - \mathbf{v}\| = 0$ のときは主張は自明である）。■

パネル (b)（Figure 35.22）はこの性質を視覚的に表している。円板の外にある 2 点間の距離は、常にそれらの射影間の距離以上である。ここでは C の凸性が本質的である。非凸な 2 点集合 $C = \{-1, 1\}$ をとると、$y = 0.01$ は 1 へ射影される一方、$y = -0.01$ は -1 へ射影され、0.02 の入力の差が 2 の出力の差へと拡大してしまう。さらに $y = 0$ では射影が一意ですらなくなる。非凸集合への射影は不連続にも多価にもなりうるのである。

35.6.3.2 射影勾配降下法

射影勾配降下法（projected gradient descent、PGD）は、勾配ステップのたびに射影を施すことで、射影を反復ソルバへと仕立て上げる。

$$\mathbf{x}_{k+1} = \Pi_C\left(\mathbf{x}_k - \eta\,\nabla f(\mathbf{x}_k)\right). \tag{35.68}$$

構成から明らかなように、すべての反復点は実行可能に保たれる。さらに都合のよいことに、PGD の不動点（fixed point）はまさに制約付き停留点と一致する。任意の微分可能な f、閉凸集合 C、およびステップサイズ $\eta > 0$ に対し、不動点方程式 $\mathbf{x}^\star = \Pi_C(\mathbf{x}^\star - \eta\nabla f(\mathbf{x}^\star))$ が成り立つのは、$\mathbf{x}^\star \in C$ が鈍角不等式 Equation 35.67 を $\mathbf{y} = \mathbf{x}^\star - \eta\nabla f(\mathbf{x}^\star)$ に対して満たすとき、かつそのときに限る。$\mathbf{x}^\star$ を相殺して $\eta > 0$ で割ると、この不等式は次の形になる。

$$\nabla f(\mathbf{x}^\star)^{\top}(\mathbf{x} - \mathbf{x}^\star) \;\geq\; 0 \qquad \text{for all } \mathbf{x} \in C.$$

平たく言えば、実行可能な方向の中に降下方向は 1 つも存在しない。これは一般的な制約付き一階条件であり、$\nabla f(\mathbf{x}^\star) = \mathbf{0}$ へと $C = \mathbb{R}^n$ のときに帰着する。f が凸であれば、接平面の不等式からこの条件は直ちに大域的最小点の十分条件となり、f が非凸であれば停留性を保証する。さ

らに $C = \{\mathbf{x} : g_i(\mathbf{x}) \le 0\}$ がスレーター条件を満たす微分可能な凸関数 g_i で定まるとき、この条件は KKT 条件と同値になる。

余分な射影ステップによって勾配降下法は遅くなるのだろうか。驚くべきことに遅くはならず、非拡大性（nonexpansiveness）から 3 行の証明が得られる。

命題（PGD は少なくとも GD と同じ速さで縮小する） C **を空でない閉凸集合とし、** f **は 2 回連続微分可能で** $\mu \mathbf{I} \preceq \nabla^2 f \preceq L\mathbf{I}$ **があるような** $0 < \mu \le L$ **に対して成り立つとし、** $\mathbf{x}^\star$ **を** f **の** C **上での最小点とし、** $\eta > 0$ **とする。このとき、Equation 35.68 の反復列は次式を満たす。**

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \;\le\; \rho \,\|\mathbf{x}_k - \mathbf{x}^\star\|, \qquad \rho = \max\left(|1-\eta\mu|,\; |1-\eta L|\right). \tag{35.69}$$

証明： (i) $\mathbf{x}^\star$ は PGD の不動点であるため、$\mathbf{x}^\star = \Pi_C(\mathbf{b})$（ただし $\mathbf{b} = \mathbf{x}^\star - \eta\nabla f(\mathbf{x}^\star)$）と書ける。(ii) この $\mathbf{b}$ の選び方が急所となるステップであり、非拡大性を $\mathbf{x}_k - \eta\nabla f(\mathbf{x}_k)$ と $\mathbf{b}$ に適用すると、射影がそっくり取り除かれる。

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \;\le\; \big\|(\mathbf{x}_k - \mathbf{x}^\star) - \eta\left(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star)\right)\big\|.$$

(iii) $\mathbf{x}^\star$ と $\mathbf{x}_k$ を結ぶ線分に微分積分学の基本定理を適用すると、$\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star) = \bar{\mathbf{H}}(\mathbf{x}_k - \mathbf{x}^\star)$（ただし $\bar{\mathbf{H}} = \int_0^1 \nabla^2 f(\mathbf{x}^\star + t(\mathbf{x}_k - \mathbf{x}^\star))\, dt$）が得られる。この平均化されたヘッセ行列は対称ですべての固有値が $[\mu, L]$ に属するため、$\mathbf{I} - \eta\bar{\mathbf{H}}$ の各固有値は $1 - \eta h$（$h \in [\mu, L]$）の形をしており、その絶対値は高々 ρ である。したがって右辺は高々 $\rho\,\|\mathbf{x}_k - \mathbf{x}^\star\|$ で抑えられる。$\blacksquare$

ステップ (ii) で射影を取り除いてしまえば、あとに残るのは制約なし勾配降下法の縮小だけである。二次関数 $f = \tfrac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x}$ に対しては $\bar{\mathbf{H}} = \mathbf{A}$ となり、ステップ (iii) はモードごとの厳密な縮小率 $|1 - \eta\lambda_i(\mathbf{A})|$（35.1 節）を復元し、その値は 1 より真に小さく保たれる（$0 < \eta < 2/L$ のとき）。強凸ではない凸かつ L 滑らかな f に対しても同様に、$\eta = 1/L$ とした PGD は勾配降下法の $O(1/k)$ の目的関数値の収束率をそのまま引き継ぐ (Nesterov 2018)。

縮小率 ρ が実際にどれほどタイトかを試してみよう。悪条件な二次関数 $f(\mathbf{x}) = \tfrac{1}{2}(x_1^2 + 1000\,x_2^2)$（条件数 $\kappa = 10^3$）を、半空間 $x_1 + x_2 \ge 1$（すなわち $g(\mathbf{x}) = 1 - x_1 - x_2 \le 0$）上で最小化する。KKT 条件は手計算で解ける。制約なし最小点 $\mathbf{0}$ は実行不可能であるため、制約は有効でなければならない。停留性 $(x_1, 1000\,x_2) - \lambda(1,1) = \mathbf{0}$ から $x_1 = \lambda$ および $x_2 = \lambda/1000$ が得られ、これらを $x_1 + x_2 = 1$ に代入すると、$\lambda^\star = 1000/1001 \approx 0.999$、$\mathbf{x}^\star = (1000/1001,\; 1/1001) \approx (0.999, 0.001)$、および $p^\star = 500/1001 \approx 0.4995$ が求まる。

ここで、ステップサイズ $\eta = 0.0015$（安定限界 $2/L = 0.002$ を十分に下回る値）で PGD を実行する。先ほどの命題から、1 ステップあたりの縮小率 $\rho = \max(1-\eta,\; |1 - 1000\,\eta|) = 0.9985$ が保証される。$(3, 4)$（距離 4.47 だけ $\mathbf{x}^\star$ から離れた点）から出発すると、この上界からは最大 $\log(4.47 \times 10^6)/\log(1/\rho) \approx 10{,}200$ ステップを距離 10^{-6} に達するまでに要すると予測される。以下のコードでは PGD を実行し、1 ステップでの最悪の縮小比、実際のステップ数、毎ステップ境界に当たり始める時点、境界に沿った縮小比、および最終的な勾配を調べる。

```
A = np.diag([1.0, 1000.0])                      # f(x) = 1/2 x^T A x, kappa = 1000
x_star = np.array([1000 / 1001, 1 / 1001])      # KKT point of x1 + x2 >= 1
project = lambda y: y + max(0.0, 1 - y.sum()) / 2  # onto the half-space x1 + x2 >= 1
eta = 1.5e-3
rho = max(abs(1 - eta), abs(1 - 1000 * eta))    # the GD contraction factor
x, dist, projected = np.array([3.0, 4.0]), [], []
for k in range(1000):
```

```
        y = x - eta * A @ x
        projected.append(y.sum() < 1)                  # did the step leave C?
        x = project(y)
        dist.append(np.linalg.norm(x - x_star))
    dist, projected = np.array(dist), np.array(projected)
    ratios = dist[1:] / dist[:-1]
    steps = np.argmax(dist < 1e-6) + 1
    on_line = np.flatnonzero(~projected).max() + 2     # every step from here on is projected
    print(f'bound rho = {rho:.4f}   largest ratio in a step = '
          f'{ratios[dist[:-1] > 1e-12].max():.4f}')
    print(f'steps to |x - x*| < 1e-6: {steps}   every step projected from step {on_line} on')
    print(f'ratio per step on the line: {np.median(ratios[on_line:steps - 1]):.4f}'
          f'   1 - 500.5 eta = {1 - 500.5 * eta:.4f}')
    print(f'grad f at the end = {A @ x}')
```

```
bound rho = 0.9985   largest ratio in a step = 0.9977
steps to |x - x*| < 1e-6: 738   every step projected from step 733 on
ratio per step on the line: 0.2492   1 - 500.5 eta = 0.2492
grad f at the end = [0.999001 0.999001]
```

1 ステップでの最大の比は 0.9977 と $\rho = 0.9985$ に迫っており、1 ステップあたりの最悪ケース評価はほぼタイトである。ところが総ステップ数は嬉しい驚きをもたらす。738 ステップで済み、10,200 ステップには遠く及ばないのである。出力された診断情報からその理由がわかる。軌跡が 2 つの明確な段階に分かれて進むためである。第 1 段階では、最初のステップのあと制約は非有効のままであり、急峻な座標 x_2 は数十回の反復で潰れる一方、平坦な座標 x_1 は制約なし勾配降下法とまったく同様に毎ステップ $1-\eta$ 倍ずつじわじわと内側へ這い進む。$x_1 = 3$ から境界線に到達するまでにおよそ $\log 3/\eta \approx 730$ ステップかかる。ステップ 733 以降は、すべての更新が境界に当たって直線 $x_1 + x_2 = 1$ 上へ射影される。この直線に沿うと二次関数の曲率は $(1+1000)/2 = 500.5$ となるため、$\mathbf{x}^\star$ までの距離は毎ステップ $1 - 500.5\,\eta = 0.249$ 倍に縮まり（直線に沿った 1 次元勾配降下法の厳密な縮小率）、あとほんの数ステップで実行が完了する。ρ は平面上のあらゆる場所における最悪の曲率に備えなければならないのに対し、いったん制約が有効になれば**境界に沿った**曲率だけが効いてくるのである。最後に、終了時点での勾配 $(0.999, 0.999) = \lambda^\star(1,1) = -\lambda^\star \nabla g$ にも注目しよう。ゼロになるどころか、手計算で導いた乗数 $\lambda^\star = 1000/1001$ をそのまま返してくれる。

Π_C が安価に計算できる場面では、PGD は主力となる手法である。非負制約では単に負の成分をゼロにするだけで非負象限（nonnegative orthant）へ射影でき、重み行列の各行に対する最大ノルム制約では更新のたびに各行を球面上へ再スケーリングすればよい。深層学習におけるおそらく最も有名な応用は、入力摂動 δ（$\|\delta\|_\infty \le \epsilon$）に対する射影勾配上昇法によって敵対的サンプルを生成する手法であり、そこでの射影は単に各成分を $[-\epsilon, \epsilon]$ にクリップするだけである (Madry et al. 2018)。なお、勾配クリッピングも球への射影を用いるが、射影するのはパラメータではなく**勾配ベクトル**であるため、重みに対するいかなる制約の PGD でもないことに注意しよう（演習問題 2 でその違いを浮き彫りにする）。クリッピングが達成しているのは、勾配とともに曲率が増大する領域でもステップを安全に保つことである（35.1.1.5 節）。

35.6.3.3 単体への射影

アテンション重みやクラス確率は**確率単体**(probability simplex)$\Delta = \{\mathbf{x} : \mathbf{x} \ge 0,\ \sum_i x_i = 1\}$ の上に乗っている。実数スコアのベクトル $\mathbf{y}$ が与えられたとき、Δ 上で最も近い点をどのように見つければよいだろうか。それには次の問題を考える。

$$\min_{\mathbf{x}} \ \tfrac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2 \quad \text{subject to} \quad \textstyle\sum_i x_i = 1, \ \ -x_i \le 0,$$

これは小規模な**二次計画問題**（quadratic program、QP）、すなわちアフィン制約のもとで凸二次目的関数を最小化する問題であり、KKT 条件を使うと 1 つのスカラーを求める問題に帰着する。和の制約に対する乗数 τ と、乗数 $\lambda_i \ge 0$（非負制約 $-x_i \le 0$ に対するもの）を導入しよう。停留性から $x_i - y_i + \tau - \lambda_i = 0$ が得られ、相補性条件によって各座標は 2 つのグループに分かれる。$x_i > 0$ となる座標では $\lambda_i = 0$ でなければならず、したがって $x_i = y_i - \tau$ となる。一方 $x_i = 0$ となる座標では、停留性から $\lambda_i = \tau - y_i$ となり、双対実行可能性 $\lambda_i \ge 0$ から $y_i \le \tau$ が課される。両方のケースをまとめると次式が得られる。

$$x_i^\star = \max(y_i - \tau,\, 0), \qquad \text{with } \tau \text{ set by } \sum_i \max(y_i - \tau,\, 0) = 1. \tag{35.70}$$

言い換えれば、単体への射影とは、すべてのスコアから同じ閾値 τ を引き、負になった値をゼロにクリップするだけの操作である。注目すべきことに、この閾値の式は**任意の** τ に対して停留性、相補性条件、双対実行可能性、および非負性を自動的に満たしており、和が 1 であるという制約 $\sum_i x_i = 1$ だけが τ を一意に定める。左辺の $\varphi(\tau) = \sum_i \max(y_i - \tau, 0)$ は連続な区分線形関数であり、正の値をとる領域では狭義単調減少であるため、一意な τ（$\varphi(\tau) = 1$ の解）が存在する。

簡単な手計算で τ の見つけ方を確認しよう。すでに降順に並んだベクトル $\mathbf{y} = (0.9, 0.5, -0.2)$ をとり、上位 k 個のスコアがゼロより大きく生き残ると推測してみる。すると和の制約 $\sum_{j\le k}(y_j - \tau) = 1$ から閾値の候補 $\tau_k = (\sum_{j\le k} y_j - 1)/k$ が予測され、これが自己無撞着になるのは k 番目のスコアが τ_k より真に大きく保たれる場合に限られる。

- $k = 1$: $\tau_1 = -0.1$ であり、$y_1 = 0.9 > \tau_1$ を満たす。
- $k = 2$: $\tau_2 = 0.2$ であり、$y_2 = 0.5 > \tau_2$ を満たす。
- $k = 3$: $\tau_3 \approx 0.067$ であるが、$y_3 = -0.2 < \tau_3$ となる。

妥当な最大の選択は $k = 2$ であり、$\tau = 0.2$ および $\mathbf{x}^\star = (0.7, 0.3, 0)$ が得られる（Figure 35.23）。比較として、同じスコアにソフトマックスを適用すると $(0.50, 0.33, 0.17)$ となり、すべての成分が真に正に保たれる。

2 つの単純な考察から、この手計算は一般的なアルゴリズムへと変わる。第 1 に、**サポート** $\{i : x_i^\star > 0\}$ は常に上位のスコアからなる。実際、$y_i > y_j$ かつ $x_j^\star > 0$ ならば $x_i^\star \ge y_i - \tau > y_j - \tau = x_j^\star > 0$ が成り立つ。（**有効な**符号制約は残りのゼロ座標である。）第 2 に、スコアを $u_1 \ge \cdots \ge u_n$ と降順にソートすると、判定条件 $u_k > \tau_k$ は真のサポートサイズ以下のすべての k で成り立ち、それを超えると破綻する。実際、$u_k > \tau_k$ を変形すると $\sum_{j<k}(u_j - u_k) < 1$、すなわち $\varphi(u_k) < 1 = \varphi(\tau)$ となり、φ が狭義単調減少であることから、これは $u_k > \tau$ と同値である。したがって、$\mathbf{y}$ をソートし、最大の k（$u_k > \tau_k$ を満たすもの）を選んで $\tau = \tau_k$ とおけばよく、計算時間は $O(n \log n)$ となる (Held et al. 1974; Duchi et al. 2008)。

アテンションスコアに適用するとき、この単体射影はスコアの低い位置に厳密なゼロの重みを割り当てることから **sparsemax** と呼ばれる (Martins and Astudillo 2016)。ソフトマックスと sparsemax の違いは正則化項 1 つだけである。二乗距離を展開すると、sparsemax は $\mathbf{y}^\top\mathbf{x} - \tfrac{1}{2}\|\mathbf{x}\|^2$ を Δ 上で最大化するのに対し、ソフトマックスは二次のペナルティをシャノンエントロピーに置き換えた $\mathbf{y}^\top\mathbf{x} + H(\mathbf{x})$ を最大化する（35.7 節）。

射影を実装したら、その出力を検証するのがよい習慣である。最初に行う自然なテストは KKT 残差の確認であるが、どの条件が実際に誤りを識別できるかには注意が必要である。以下のセルで

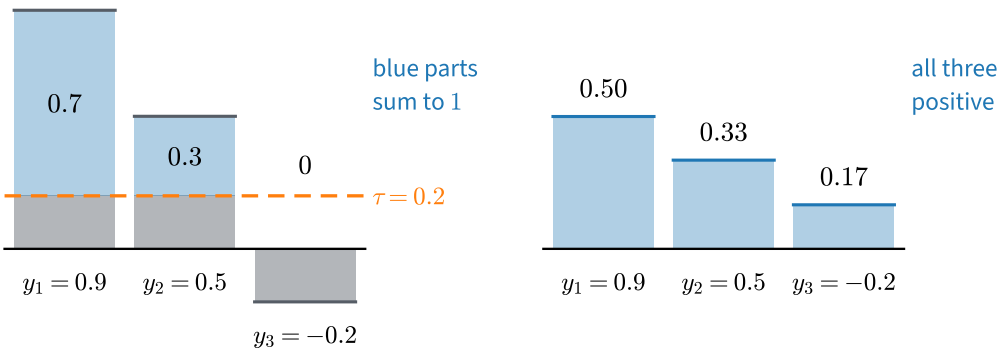


Figure 35.23: $\mathbf{y} = (0.9, 0.5, -0.2)$ の確率単体への射影。(a) ではすべてのスコアが閾値 $\tau = 0.2$ だけ引き下げられてゼロでクリップされ、τ は $\sum_i x_i = 1$ によって一意に定まる。(b) の同じスコアに対するソフトマックスでは、すべての座標が真に正に保たれる。

は、6 個のランダムなスコアに対して、計算された τ と摂動を加えた値 $\tau + 0.3$ の両方でテストし、さらに鈍角不等式 Equation 35.67 による最適性の証明書を確認する。Equation 35.67 の左辺は $\mathbf{x}$ に関してアフィンであるため、Δ 上での最大値は頂点で達成される。したがって、n 個の標準基底ベクトル $\mathbf{e}_i$ を調べるだけで単体全体を検証するのに十分である。

```
def project_simplex(y):
    """Sort-and-threshold projection onto {x >= 0, sum x = 1}; returns x* and tau."""
    u = np.sort(y)[::-1]
    tau_k = (np.cumsum(u) - 1) / np.arange(1, len(y) + 1)   # tau_k for k = 1..n
    k = np.nonzero(u > tau_k)[0][-1]                          # largest k with u_k > tau_k
    return np.maximum(y - tau_k[k], 0.0), tau_k[k]

y = np.random.default_rng().normal(size=6)
x, tau = project_simplex(y)
print('x* =', x.round(3), ' zeros:', int((x == 0).sum()), ' tau =', round(tau, 3))
for t in (tau, tau + 0.3):                       # three conditions hold for every tau ...
    xt, lam = np.maximum(y - t, 0), np.maximum(0, t - y)
    print(f'tau = {t:6.3f}: stationarity {abs(xt - y + t - lam).max():.0e}, '
          f'comp. slack {abs(lam * xt).max():.0e}, dual feas. {lam.min() >= 0}, '
          f'sum x = {xt.sum():.3f}')             # ... only the budget differs
cert = ((y - x) @ (np.eye(6) - x).T).max()      # max_i (y - x*)^T (e_i - x*)
print(f'certificate max_i (y - x*)^T(e_i - x*) = {cert:.0e}  '
      f'(<= 0 up to rounding: {cert <= 1e-12})')
```

```
x* = [0.    0.    0.    0.    0.988 0.012]  zeros: 4  tau = 0.946
tau =  0.946: stationarity 0e+00, comp. slack 0e+00, dual feas. True, sum x = 1.000
tau =  1.246: stationarity 0e+00, comp. slack 0e+00, dual feas. True, sum x = 0.688
certificate max_i (y - x*)^T(e_i - x*) = 3e-17  (<= 0 up to rounding: True)
```

通常、6 個の座標のうち 3〜5 個が厳密なゼロに設定される。これらは有効な非負制約であり、それぞれ正の乗数 $\tau - y_i$ を伴う。ここで診断上の落とし穴に注目しよう。停留性、相補性条件、および双対実行可能性は、**両方**の値（正しい τ と誤った値）に対して丸め誤差の範囲内で通って

しまう。誤りを捉えるのは和の制約だけであり、したがってこれら 3 つの残差を調べるだけでは解の正しさを確認できない。一方、頂点による証明書は確実に機能する。丸め誤差の範囲内で $\max_i (\mathbf{y}-\mathbf{x}^\star)^\top(\mathbf{e}_i-\mathbf{x}^\star) \le 0$ であることを確かめれば、単体上のどの点もこれ以上 $\mathbf{y}$ に近づけないことが証明される。

35.6.4 近接作用素と座標降下法

35.6.4.1 ソフト閾値処理と近接作用素

重みを抑制する方法は、硬い境界を設けることだけではない。**Lasso** (Tibshirani 1996) は、硬い壁の代わりに折れ目を持つ ℓ_1 ペナルティを最小二乗損失に加える。

$$F(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{w}-\mathbf{b}\|^2 + \lambda\|\mathbf{w}\|_1, \qquad \mathbf{A}\in\mathbb{R}^{n\times d},$$

その最小点は、多くの座標を厳密なゼロに設定するのが特徴である。ここでペナルティの重みをあえて λ と書いたのは、それがノルム制約 $\|\mathbf{w}\|_1 \le c$ に対するラグランジュ乗数のように働くからである。すなわち、F のすべての最小点は、それが消費する ℓ_1 バジェット c のもとで最小二乗損失を最小化しており、逆に任意のバジェット $c>0$ はあるペナルティ λ に対応する。35.7.3.2 節では、Lasso と ℓ_2 重み減衰（weight decay）の両方について、この双方向の対応を証明する。

数値的に何が起きるかを見るため、5 個のゼロ座標を持つスパースなベクトル $\mathbf{w}$ を $\mathbb{R}^8$ 内に用意し、$n=100$ 個のノイズを含む線形観測値を生成して $\lambda=3$ と設定しよう。ステップサイズは $\eta = 1/L$ と選ぶ。ここで $L = \lambda_{\max}(\mathbf{A}^\top\mathbf{A})$ は $\mathbf{A}^\top\mathbf{A}$ の最大固有値であり、最小二乗項の滑らかさの定数である。

```
rng = np.random.default_rng()
w_planted = np.array([2.0, 0, 0, -1.5, 0, 0, 0, 1.0])
A = rng.normal(size=(100, 8))
b = A @ w_planted + 0.05 * rng.normal(size=100)
lam = 3.0
L = np.linalg.eigvalsh(A.T @ A).max()
eta = 1 / L
F = lambda w: 0.5 * ((A @ w - b)**2).sum() + lam * np.abs(w).sum()
soft = lambda z, t: np.sign(z) * np.maximum(np.abs(z) - t, 0.0)  # derived below
print(f'L = lambda_max(A^T A) = {L:.1f};  |a_j|^2 range from '
      f'{(A**2).sum(0).min():.0f} to {(A**2).sum(0).max():.0f}')
```

```
L = lambda_max(A^T A) = 144.6;  |a_j|^2 range from 83 to 127
```

ここで L は通常 130〜190 の範囲に収まるのに対し、列の二乗ノルム $\|\mathbf{a}_j\|^2$（$\mathbf{A}$ の各列）は 100 前後になる。

試すべき最も素朴なアルゴリズムは劣勾配降下法（subgradient descent）である。35.3 節で学んだとおり、$|w|$ の劣勾配（subgradient）は $\operatorname{sign}(w)$（$w\neq 0$ のとき）であり、折れ目では区間 $[-1,1]$ 全体（$w=0$ のとき）となる。各ステップで $\operatorname{sign}(\mathbf{w})$ を選ぶと、更新式 $\mathbf{w} \leftarrow \mathbf{w} - \eta\left(\mathbf{A}^\top(\mathbf{A}\mathbf{w}-\mathbf{b}) + \lambda \operatorname{sign}(\mathbf{w})\right)$ が得られる。これで 5 個のゼロを復元できるだろうか。

```
w_sub, signs = np.zeros(8), []
for k in range(50):                                  # subgradient descent, step 1/L
    w_sub = w_sub - eta * (A.T @ (A @ w_sub - b) + lam * np.sign(w_sub))
    signs.append(np.sign(w_sub))
```

```
null = w_planted == 0
S = np.array(signs)[-20:, null]
print(f'exact zeros: {int((w_sub == 0).sum())} of 8;  sign flips of the null '
      f'coordinates, last 20 steps: {(S[1:] != S[:-1]).sum()} of 95')
print(f'largest null coordinate: {np.abs(w_sub[null]).max():.3f};  '
      f'eta * lam = {eta * lam:.3f}')
```

```
exact zeros: 0 of 8;  sign flips of the null coordinates, last 20 steps: 89 of 95
largest null coordinate: 0.032;  eta * lam = 0.021
```

厳密なゼロを生成することには完全に失敗する。どの座標もゼロに着地せず、5 個の**ヌル**座標（あらかじめゼロに設定した座標）はほぼ毎回の反復で符号を反転させ、約 $\eta\lambda$ の振幅で行ったり来たり振動する。原因はゼロにある折れ目である。真の Lasso 解において、ヌル座標における滑らかな項の勾配は絶対値が λ より真に小さい。そのためゼロの近くでは、ペナルティ項 $-\eta\lambda\,\mathrm{sign}(w_j)$ がステップを支配し、毎回およそ $\eta\lambda$ だけ座標をゼロの反対側へ蹴り出してしまう。劣勾配降下法は各ステップで区間 $[-\lambda,\lambda]$ からただ 1 つの傾きを選んで決め打ちするため、固定ステップサイズでは滑らかな項の勾配をちょうど打ち消す傾きに落ち着くことが決してできないのである。

解決策は単純であり、折れ目のあるペナルティを線形化する代わりに**厳密に**扱えばよい。まずは 1 次元から始め、$\lambda|x| + \tfrac{1}{2}(x-z)^2$ を $x \in \mathbb{R}$ 上で最小化しよう。目的関数は凸であるため、x が最適であるための必要十分条件は、0 がその劣勾配の集合に含まれることである。

- $x \neq 0$ のとき、目的関数は x で微分可能であるため $\lambda\,\mathrm{sign}(x) + x - z = 0$ となる。$x > 0$ のときはこれから $x = z - \lambda$ が得られ、これには $z > \lambda$ が必要である。一方 $x < 0$ のときは $x = z + \lambda$ が得られ、これには $z < -\lambda$ が必要である。
- $x = 0$ のとき、劣勾配は区間 $[-\lambda,\lambda] - z$ をなし、これが 0 を含むための必要十分条件は $|z| \le \lambda$ である。

これら 2 つのケースは実数直線を分割し、1 つの式にまとめると**ソフト閾値処理**になる。

$$\mathrm{prox}_{\lambda|\cdot|}(z) \;=\; \mathrm{sign}(z)\,\max\left(|z| - \lambda,\, 0\right). \tag{35.71}$$

絶対値が高々 λ の入力はすべて厳密なゼロへ吸い寄せられ、それより大きい入力はゼロに向かって λ だけ縮小される。たとえば $\lambda = 0.7$ のとき、$z = 0.5$ は 0 へ、$z = 2$ は 1.3 へ、$z = -2$ は -1.3 へと写される（Figure 35.24）。この不感帯（dead zone）$[-\lambda,\lambda]$ こそが、ℓ_1 正則化（regularization）が厳密なスパース性（sparsity）を生み出す仕組みであり、対照的に ℓ_2 正則化は重みをゼロに向かって縮小するだけで決してゼロにはしない。

ソフト閾値処理は、はるかに一般的な操作の 1 つの例にすぎない。**近接作用素**（proximal operator）は、関数 ψ に対して $\psi(\mathbf{x})$ を小さくすることと参照点 $\mathbf{z}$ の近くに留まることのトレードオフをとる。

$$\mathrm{prox}_{\psi}(\mathbf{z}) \;=\; \operatorname*{argmin}_{\mathbf{x}} \; \left(\psi(\mathbf{x}) + \tfrac{1}{2}\,\|\mathbf{x} - \mathbf{z}\|^2\right). \tag{35.72}$$

この写像はすべての**真閉凸**（proper closed convex）関数 ψ、すなわち凸であり、$\mathbb{R} \cup \{+\infty\}$ に値をとり、少なくとも 1 点で有限値をとり、かつ閉じたエピグラフを持つ関数に対して一意に定まる。これらの条件のもとで、Equation 35.72 の目的関数は 1 強凸となるため、一意な最小点を持つ (Parikh and Boyd 2014)。

3 つの特別な場合を見ることで、近接作用素が既知の概念をいかに統一しているかがわかる。

- $\psi = 0$ のとき、近接作用素は単なる恒等写像である。

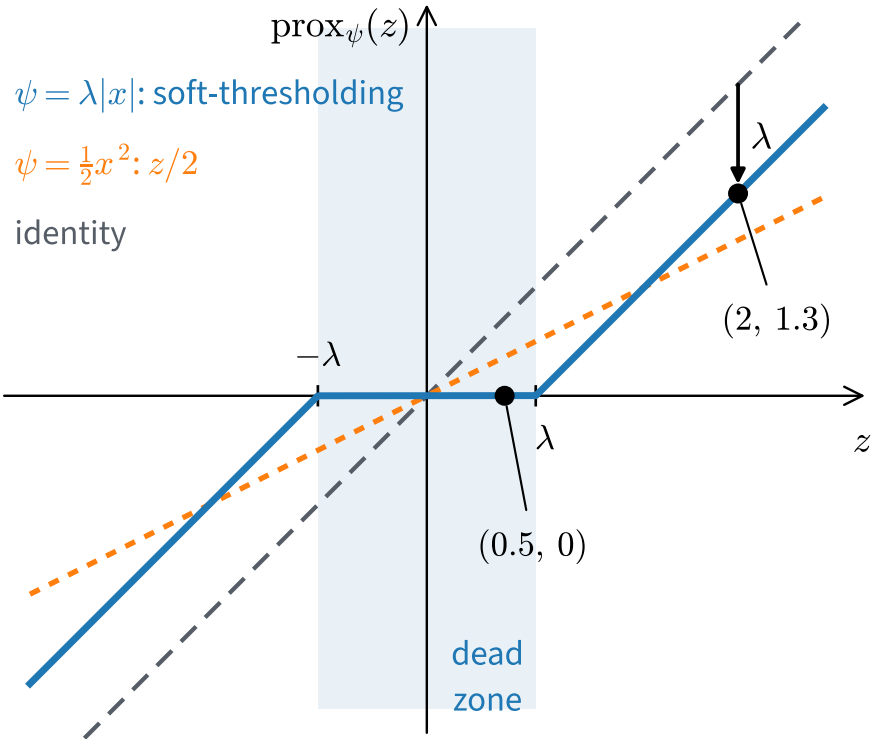


Figure 35.24: ソフト閾値処理 $\text{prox}_{\lambda|\cdot|}(z)$（$\lambda = 0.7$、実線）。不感帯 $[-\lambda, \lambda]$ 内の入力は 0 へ写され（$z = 0.5$ など）、それ以外の入力はすべて 0 に向かって λ だけ移動する（$z = 2$ など）。破線は恒等写像を表す。点線は二乗ペナルティ $\frac{1}{2}x^2$ に対応する写像であり、すべての入力を半分にするが $z = 0$ 以外をゼロにすることはない。

- ψ が閉凸集合 C の**指示関数**（indicator）（値は 0（C 上）および $+\infty$（外部））であるとき、最小点は必ず C に属さなければならないため、近接作用素はユークリッド射影 $\operatorname{prox}_{\psi} = \Pi_C$ に帰着する。近接作用素とは、集合から関数へと一般化された射影なのである。
- $\psi = \lambda\|\cdot\|_1$ のとき、目的関数は座標ごとに完全に分離するため、近接作用素はソフト閾値処理 Equation 35.71 を各座標に適用する操作となる。

演習問題 5 では重みのブロック全体を一括でゼロにするユークリッドノルムの近接作用素を導出し、演習問題 6 ではノルムの近接作用素とその双対球への射影の和が恒等写像になることを示す。

35.6.4.2 近接勾配降下法

Lasso の目的関数は合成形式 $F = s + \psi$ をしており、滑らかな損失 s と、近接作用素を安価に計算できる非平滑なペナルティ ψ を組み合わせたものである。いま、s が L リプシッツ連続な勾配を持つ凸関数であり、ψ が真閉凸であり、$0 < \eta \le 1/L$ であるとする。降下補題（descent lemma）（35.1 節、$1/\eta \ge L$ を使用）より、s は二次モデル $s(\mathbf{x}_k) + \nabla s(\mathbf{x}_k)^{\top}(\mathbf{x}-\mathbf{x}_k) + \frac{1}{2\eta}\|\mathbf{x}-\mathbf{x}_k\|^2$ によって上から抑えられる。ここで ψ まで線形化するのではなく、s のこの二次上界に**厳密な** ψ を加えたものを最小化する。平方完成すると、その最小点は次式で与えられる。

$$\operatorname*{argmin}_{\mathbf{x}} \; \psi(\mathbf{x}) + \frac{1}{2\eta}\left\|\mathbf{x} - (\mathbf{x}_k - \eta\nabla s(\mathbf{x}_k))\right\|^2 \;=\; \operatorname{prox}_{\eta\psi}\left(\mathbf{x}_k - \eta\,\nabla s(\mathbf{x}_k)\right),$$

これは滑らかな部分に対する通常の勾配ステップに続けて、折れ目のある部分に対する近接ステップを施す操作である。この更新を次のように反復する。

$$\mathbf{x}_{k+1} = \operatorname{prox}_{\eta\psi}\left(\mathbf{x}_k - \eta\,\nabla s(\mathbf{x}_k)\right), \tag{35.73}$$

こうして得られる手法が**近接勾配降下法**（proximal gradient descent）であり、特に **ISTA**（iterative shrinkage-thresholding algorithm、反復縮小閾値アルゴリズム）の名で $\psi = \lambda\|\cdot\|_1$ のときに知られている。この手法は射影勾配降下法（ψ として C の指示関数をとる）と通常の勾配降下法（$\psi = 0$ ととる）の両方を包含する。

F が微分可能でなくても、$\eta = 1/L$ とした近接勾配降下法は滑らかな勾配降下法と同じ $O(1/k)$ の収束率 $F(\mathbf{x}_k) - F^\star \le L\|\mathbf{x}_0 - \mathbf{x}^\star\|^2/(2k)$ を達成する（Beck and Teboulle (2009) 定理 3.1）。さらに、s が 2 回連続微分可能かつ μ 強凸（$\mu\mathbf{I} \preceq \nabla^2 s$）であれば、先ほどの PGD の縮小の証明が一字一句そのまま通用する。証明に必要なのは $\mathbf{x}^\star$ が不動点であることと、$\operatorname{prox}_{\eta\psi}$ が Π_C と同様に非拡大であることだけだからである（演習問題 7 でこれを証明する）。

Lasso の場合、近接ステップは閾値 $\eta\lambda$ のソフト閾値処理であり、ゼロから $\eta\lambda$ 以内にあるすべての座標を直接ゼロへと吸い寄せる。この閾値は劣勾配降下法が振動していた振幅とぴったり一致するため、ISTA はヌル座標をきれいにゼロに保てるはずである。実際に確かめてみよう。

```
w = np.zeros(8)
for k in range(50):                                  # ISTA: gradient step, then prox
    w = soft(w - eta * A.T @ (A @ w - b), eta * lam)
print(f'exact zeros: {int((w == 0).sum())} of 8;  support recovered: '
      f'{np.array_equal(w != 0, w_planted != 0)};  '
      f'max deviation on the support: {np.abs(w - w_planted)[~null].max():.3f}')
```

```
exact zeros: 5 of 8;  support recovered: True;  max deviation on the support: 0.040
```

劣勾配降下法と 1 ステップあたり同じ計算コストで同じ 50 回の反復を行うと、ISTA は真のサポートを正確に復元し、5 個のヌル座標すべてをゼロに設定して 3 個の有効な座標を非ゼロのまま残す。これら 3 個の成分は真の値から約 0.05 以内に収まっている。これ以上真の値に大きく近づくことはないが、それは ℓ_1 ペナルティによって真の Lasso 解自体がゼロに向かっておよそ $\lambda/\|\mathbf{a}_j\|^2 \approx 0.03$ だけ縮小されているためである。

35.6.4.3 座標降下法

初読時には本項を読み飛ばしてもよい。座標降下法（coordinate descent）は Lasso に対する標準的なソルバであるが、ニューラルネットワークの訓練に用いられる手法ではないからである。

Lasso において、他のすべての座標を固定したまま 1 つの座標 w_j に沿って最小化する問題は、先ほど解いた 1 次元の問題そのものに帰着する。$\mathbf{r}_{-j} = \mathbf{b} - \sum_{i \neq j} \mathbf{a}_i w_i$ を座標 j を除いて計算した残差としよう。j 番目の部分問題 $\frac{1}{2}\|\mathbf{a}_j w_j - \mathbf{r}_{-j}\|^2 + \lambda |w_j|$ は、定数項を除いて $\|\mathbf{a}_j\|^2 \left(\frac{1}{2}(w_j - c_j/\|\mathbf{a}_j\|^2)^2 + (\lambda/\|\mathbf{a}_j\|^2)|w_j| \right)$ に等しい（ただし $c_j = \mathbf{a}_j^\top \mathbf{r}_{-j}$）。したがって、閾値 $\lambda/\|\mathbf{a}_j\|^2$ のソフト閾値処理によって閉形式で解ける。

$$w_j \leftarrow \frac{\operatorname{sign}(c_j)\,\max(|c_j| - \lambda,\, 0)}{\|\mathbf{a}_j\|^2}, \qquad c_j = \mathbf{a}_j^\top \mathbf{r}_{-j}. \tag{35.74}$$

ある特徴量と現在の残差との相関 c_j の絶対値が高々 λ であれば、その重みは直ちにゼロに設定される。この更新式で座標を順番に巡回する手法こそ、広く使われている `glmnet` パッケージの背後にあるアルゴリズムである (Friedman et al. 2010)。座標を更新するたびに残差を差分更新すれば、全 d 座標を 1 周するスイープのコストは 1 回の勾配計算と同じになる。

同じ原理は、任意の微分可能な目的関数 f が座標ごとの滑らかさの定数 L_j を持ち、次式を満たす場合にも適用できる。

$$f(\mathbf{x} + t\mathbf{e}_j) \le f(\mathbf{x}) + t\,\partial_j f(\mathbf{x}) + \tfrac{1}{2} L_j t^2, \tag{35.75}$$

この二次上界を t に関して最小化すると、次の**座標降下法**の更新式が得られる。

$$x_j \leftarrow x_j - \frac{1}{L_j} \partial_j f(\mathbf{x}). \tag{35.76}$$

L_j は 1 つの座標軸に沿った曲率を測るため、大域的な滑らかさの定数 L より大幅に小さくなり得て、より大きなステップをとることができる。実際、先ほどの Lasso 問題では $L_j = \|\mathbf{a}_j\|^2 \approx 100$ であるのに対し、$L \in [130, 190]$ であった。正定値二次関数 $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}$ に対しては、$L_j = A_{jj}$ で上界が厳密に一致し、次の更新式が得られる。

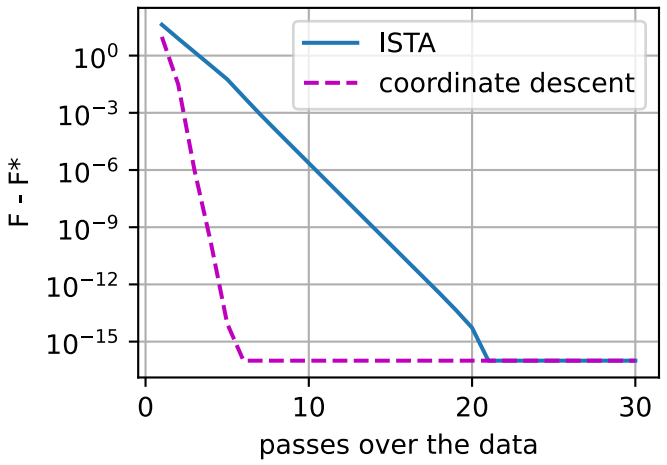
$$x_j \leftarrow x_j - \frac{(\mathbf{A}\mathbf{x} - \mathbf{b})_j}{A_{jj}} \tag{35.77}$$

この更新は f を座標 j に沿って厳密に最小化する。すべての座標を巡回的にスイープすると、$\mathbf{A}\mathbf{x} = \mathbf{b}$ に対する古典的なガウス・ザイデル反復が復元される。

1 回の座標スイープは、同じ計算コストの ISTA の 1 ステップよりどれほど速いだろうか。先ほどの Lasso 問題において、目的関数のギャップ $F(\mathbf{w}) - F^\star$ を 10^{-8} 未満に押し下げるまでに各手法がデータを何回走査するかを数えてみよう。

```
w_ref = np.zeros(8)
for k in range(5000):                               # reference optimum by ISTA
    w_ref = soft(w_ref - eta * A.T @ (A @ w_ref - b), eta * lam)
F_star, col2 = F(w_ref), (A**2).sum(0)              # col2[j] = |a_j|^2
w_i, w_c, r = np.zeros(8), np.zeros(8), b.copy()    # r = b - A w_c, kept up to date
gap_i, gap_c = [], []
for sweep in range(30):
    w_i = soft(w_i - eta * A.T @ (A @ w_i - b), eta * lam)   # one ISTA step
    for j in range(8):                                       # one coordinate sweep
        c_j = A[:, j] @ r + col2[j] * w_c[j]                 # a_j^T r_{-j}
        w_new = soft(c_j, lam) / col2[j]
        r -= A[:, j] * (w_new - w_c[j])
        w_c[j] = w_new
    gap_i.append(F(w_i) - F_star); gap_c.append(F(w_c) - F_star)
passes = lambda gap: next((k + 1 for k, g in enumerate(gap) if g < 1e-8),
                          '> 30')                    # not reached in 30 passes
print(f'passes to gap < 1e-8: ISTA {passes(gap_i)}, coordinate descent {passes(gap_c)}')
d2l.plot(np.arange(1, 31), [np.maximum(gap_i, 1e-16), np.maximum(gap_c, 1e-16)],
         'passes over the data', 'F - F*', legend=['ISTA', 'coordinate descent'],
         yscale='log')                               # gaps floored at rounding level
```

```
passes to gap < 1e-8: ISTA 13, coordinate descent 4
```



対数プロット上では、どちらの手法も直線を描いて減少する（線形収束）が、座標降下法のほうがはるかに急激に減少し、通常はギャップ 10^{-8} に 4〜6 回のスイープで到達するのに対し、ISTA では 12〜25 ステップを要する。本節で証明したどの結果からも、この正確な高速化倍率は予測できない。座標降下法に対する最もシャープな収束率はランダムに座標を選ぶ設定で証明されており (Nesterov 2012)、ここで用いたような巡回的スイープも保証された収束率で収束するものの、その定数はより弱いものになる (Beck and Tetruashvili 2013)。

座標降下法には重要な制約もある。非平滑な折れ目が座標ごとに分離しない場合、座標降下法は最適点から遠く離れた場所で立ち往生しうる。たとえば $f(x, y) = \max(|x|, |y|)$ において点

$(1,1)$ にいるとき、x 単独または y 単独に沿って動いても f を減少させることはできないが、大域的最小値 0 は原点にある。座標降下法が Lasso でうまく機能するのは、その非平滑ペナルティ $\lambda \sum_j |w_j|$ が座標ごとに分離可能だからにほかならない (Tseng 2001)。スパース線形モデルやブロック交互最小化においては、座標降下法は今なお第一選択の手法である。しかし密なニューラルネットワークの訓練では出番がない。現代のアクセラレータは、逐次的なスカラー更新ではなく並列な行列積のために設計されているからである。

本節で行ってきたことすべてを貫く 1 つのパターンに注目しよう。乗数を求めたいときは毎回、まず $\mathbf{x}$ について解いていた。35.7 節ではこの問いを裏返し、乗数を**直接**求める方法、そして任意の乗数の選択が「いかなる実行可能点も所定の下界を下回れない」ことを示す証明書をいかに与えるかを学ぶ。$\lambda^\star = 4$、$p^\star = 8$ であった先ほどのクリッピングの例が、その冒頭を飾ることになる。

35.6.5 まとめ

制約付き最適点では、勾配がゼロになる必要はなく、代わりにいかなる実行可能な方向も目的関数を減少させられないという条件が成り立つ。$\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$ を満たす 1 つの等式制約に対しては、これは $\nabla f + \nu \nabla h = \mathbf{0}$ が、解が制約をどれほど強く押しているかを測るラグランジュ乗数 ν に対して成り立つことを意味する。不等式制約を加えると 4 つの KKT 条件が導かれ、そこでは双対実行可能性（$\lambda_i \geq 0$）が片方向の壁を表し、相補性条件（$\lambda_i g_i = 0$）がすべての非有効制約の乗数をゼロにする。凸問題においてはすべての KKT 点が大域的最小点であり、制約想定（LICQ やスレーター条件など）のもとではすべての最小点が KKT 点となる。

制約付きの解を反復的に計算するため、射影勾配降下法は各勾配ステップに続けてユークリッド射影 Π_C を適用する。閉凸集合への射影は一意であり、鈍角不等式によって特徴づけられ、かつ非拡大であるため、射影勾配降下法は制約なし勾配降下法の収束率をそのまま保つ。球、半空間、および確率単体への射影は、いずれも 1 つのスカラー乗数について解く問題に帰着する。最後に、近接作用素は射影を集合から非平滑な関数へと一般化する。ソフト閾値処理は $\lambda|\cdot|$ の近接作用素であり、ISTA と座標降下法はいずれもこれを用いることで、劣勾配降下法では到達できない厳密なゼロを伴う Lasso の解を求める。

35.6.6 演習問題

1. **退化した制約**。ラグランジュ条件 Equation 35.61 と KKT 条件 Equation 35.65 は、制約想定のもとでのみ最小点において成り立つ。以下の各問題は、いずれかの制約想定に違反している。
    1. $f(\mathbf{x}) = x_1 + x_2^2$ を $\mathbb{R}^3$ 上で制約 $h_1(\mathbf{x}) = x_3 = 0$ および $h_2(\mathbf{x}) = x_3 - x_1^2 = 0$ のもとで最小化せよ。実行可能領域が直線であることを示し、最小点を求め、両方の制約勾配がいたるところで非ゼロであることを示せ。また、最小点においていかなる乗数 ν_1, ν_2 も $\nabla f + \nu_1 \nabla h_1 + \nu_2 \nabla h_2 = \mathbf{0}$ を満たさないことを示せ。さらに、乗数が存在するような別の 2 つの等式制約によって同じ直線を表せ。制約想定とは実行可能領域そのものの性質か、それともそれを記述する関数の性質か。
    2. $f(x) = x$ を $\mathbb{R}$ 上で制約 $x^2 \leq 0$ のもとで最小化せよ。$p^\star$ を求め、最小点においていかなる $\lambda \geq 0$ も KKT 停留条件を満たさないことを示せ。LICQ とスレーター条件のうちどちらが破れているか。それぞれについて理由を説明せよ。
    3. $f(x) = x$ を $\mathbb{R}$ 上で、制約 $-x \leq 0$ を $g_1(x) = g_2(x) = -x$ と 2 回課したもとで最小化せよ。LICQ が破れ、スレーター条件が成り立つことを示せ。最小点において KKT 条件を満たすすべての組 $(\lambda_1^\star, \lambda_2^\star)$ を求めよ。緩和した制約 $-x \leq u_1$ および $-x \leq u_2$ のもとで最適値 $p^\star(u_1, u_2)$ を計算し、すべての KKT 乗数の組が

$p^\star(\mathbf{u}) \geq p^\star(\mathbf{0}) - {\lambda^\star}^\top \mathbf{u}$ をすべての $\mathbf{u}$ に対して満たすことを確かめよ。$p^\star$ は $\mathbf{0}$ で微分可能か。また、関係式 $\lambda^\star = -\nabla p^\star(\mathbf{0})$ に取って代わるものは何か。

2. **射影の不動点。** 閉凸集合 C に対し、射影勾配降下法 Equation 35.68 の不動点は、ステップサイズ $\eta > 0$ の選び方にかかわらず制約付き停留点と一致する。

   1. $C = \{\mathbf{x} : \mathbf{x} \geq 0\}$ とする。$\mathbf{x}^\star$ が不動点であるための必要十分条件は、すべての座標 i について、$\partial f/\partial x_i(\mathbf{x}^\star) = 0$ かつ $x_i^\star \geq 0$ であるか、または $\partial f/\partial x_i(\mathbf{x}^\star) > 0$ かつ $x_i^\star = 0$ であることだと示せ。さらに、これらの条件が制約 $-x_i \leq 0$ に対する KKT 条件に一致することを示し、乗数を $\nabla f(\mathbf{x}^\star)$ を用いて表せ。
   2. 勾配クリッピングは $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\, \Pi_{B_c}(\nabla f(\mathbf{x}_k))$（ただし $B_c = \{\mathbf{z} : \|\mathbf{z}\| \leq c\}$）と更新する。この更新式の不動点を求め、パラメータ球 $\{\mathbf{x} : \|\mathbf{x}\| \leq r\}$ 上への射影勾配降下法の不動点と比較せよ。2 つの更新式のうち、制約付き問題を解いているのはどちらか。
   3. 二値重みはパラメータを非凸集合 $C = \{-1, 1\}$ に制限し、C への射影は近いほうの要素へ丸める（同距離のときは 1 とする）。$f(x) = \tfrac{1}{2}(x - a)^2$（$0 < a < 1$）とせよ。どのようなステップサイズのとき $x = 1$ が Equation 35.68 の不動点になり、どのようなステップサイズのとき $x = -1$ が不動点になるか。η が両方の範囲を超えると反復列はどう振る舞うか。f を C 上で最小化するのはどちらの点か。また、どちらかの点は $f'(x^\star)(x - x^\star) \geq 0$ をすべての $x \in C$ に対して満たすか。この集合に対しては、不動点の議論のどのステップが破綻するか。
   4. 半空間上の PGD のセルを、$\eta = 2.2/1000$（0.0015 の代わり）で実行せよ。射影は非拡大であるにもかかわらず、実行が発散するのはなぜか。縮小に関する命題のどの因子が 1 を超えているか。

3. [code] **Sparsemax と ℓ_1 球。** 関数 `project_simplex` は単体射影 Equation 35.70、すなわち sparsemax を計算し、その第 1 出力が射影後のベクトルである。

   1. すべてのスコアに同じ定数を加えても射影が変わらないことを示せ。スコアを温度 $T > 0$ で割り、$u_1 > u_2$ を $\mathbf{y}$ の上位 2 つの成分とする。$\mathbf{y}/T$ の射影がワンホットになる必要十分条件が $T \leq u_1 - u_2$ であることを示し、$T \to \infty$ としたときの極限を求めよ。10 個の標準正規分布に従うスコアについて、$\mathbf{y}/T$ の射影の非ゼロ成分の個数を T に対してプロットせよ。
   2. S を $\mathbf{x}^\star$ のサポートとし、どの y_i も閾値 τ に等しくないと仮定する。ヤコビ行列 $\partial \mathbf{x}^\star / \partial \mathbf{y}$ を Equation 35.70 から導出し (Martins and Astudillo 2016)、中心有限差分と比較して確かめよ。sparsemax を通る逆伝播は、長さ n の入力に対して何回の算術演算を必要とするか。
   3. ℓ_1 球 $\{\mathbf{x} : \|\mathbf{x}\|_1 \leq z\}$ への射影が $\mathbf{y}$ の各成分の符号を保ち、$\|\mathbf{y}\|_1 > z$ のときには $\operatorname{sign}(\mathbf{y}) \odot z\, \Pi_\Delta(|\mathbf{y}|/z)$ に等しくなることを示せ (Duchi et al. 2008)。これを `project_simplex` を用いて実装せよ。得られた結果を変分不等式 $(\mathbf{y} - \hat{\mathbf{x}})^\top (\mathbf{x} - \hat{\mathbf{x}}) \leq 0$ を $2n$ 個の点 $\pm z\, \mathbf{e}_i$ で調べることで検証せよ。なぜこれらの点だけで十分なのか。

4. **球面上の二階条件。●** ラグランジュ条件は一階条件であり、極小点と鞍点を区別できない。二階の判定法では制約の曲率を考慮に入れる必要がある。

   1. f と h を 2 回連続微分可能とし、$\mathbf{x}^\star$ が Equation 35.61 を乗数 $\nu^\star$ で満たすとする。任意の曲線 $\mathbf{x}(t)$ が $\{h = 0\}$ 上にあり、$\mathbf{x}(0) = \mathbf{x}^\star$ かつ $\dot{\mathbf{x}}(0) = \mathbf{v}$ を満たすとき、次式を示せ。

$$\frac{d^2}{dt^2} f(\mathbf{x}(t))\bigg|_{t=0} = \mathbf{v}^\top \nabla_{\mathbf{x}}^2 \mathcal{L}(\mathbf{x}^\star, \nu^\star)\, \mathbf{v},$$

ここでラグランジアンには Equation 35.62 を用いる。これから、極小点では $\mathbf{v}^\top \nabla^2_{\mathbf{x}} \mathcal{L}\, \mathbf{v} \geq 0$ がすべての接線方向 $\mathbf{v}$ に対して成り立つと結論せよ。この判定に f のヘッセ行列単独を用いるのが誤りであるのはなぜか。なお逆に、接空間上で正定値であれば狭義の極小点となる（Nocedal and Wright (2006) 第 12 章）。

2. $\mathbf{A}$ を相異なる固有値 $a_1 < a_2 < \cdots < a_n$ と単位固有ベクトル $\mathbf{q}_1, \dots, \mathbf{q}_n$ を持つ対称行列とし、$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$ を制約 $\mathbf{x}^\top \mathbf{x} = 1$ のもとで最小化せよ。ラグランジュ条件を満たす点が $\pm \mathbf{q}_i$（乗数 $\nu^\star = -a_i$）であることを示し、$\nabla^2_{\mathbf{x}} \mathcal{L}$ の $\mathbf{q}_i$ における接空間上での固有値を求めよ。これらの点のうち、極小点、極大点、鞍点はそれぞれどれか。また、$\mathbf{q}_i$ は互いに独立な降下方向を何本持つか。

3. 球面上への射影勾配降下法 $\mathbf{x} \leftarrow (\mathbf{x} - \eta \nabla f(\mathbf{x}))/\|\mathbf{x} - \eta \nabla f(\mathbf{x})\|$ は非凸集合への射影を行う。この f に対して、この更新が $\mathbf{I} - 2\eta \mathbf{A}$ に対するべき乗法になっていることを示せ。$a_1 + a_n > 0$ と仮定する。ランダムな初期点から出発したとき、反復列 $\mathbf{x}_k$ と関数値 $f(\mathbf{x}_k)$ はどう振る舞うか。その答えは $\eta > 0$ にどう依存するか。

5. **ブロックソフト閾値処理（グループ Lasso）**。ベクトル全体に課すペナルティ $\lambda \|\mathbf{w}\|_2$ は $\mathbf{w} = \mathbf{0}$ でのみ折れ目を持つ。その劣勾配は、$\mathbf{w}/\|\mathbf{w}\|$（$\mathbf{w} \neq \mathbf{0}$ のとき）および単位球内のすべてのベクトル（$\mathbf{0}$ のとき）に λ を掛けたものである。

   1. $\operatorname{prox}_{\lambda \|\cdot\|_2}(\mathbf{z}) = \max\left(0,\ 1 - \lambda/\|\mathbf{z}\|\right) \mathbf{z}$ を示せ。
   2. ゼロが生じるパターンを座標ごとのソフト閾値処理 Equation 35.71 と比較せよ。ブロック近接作用素が厳密に $\mathbf{0}$ を返すのはいつか。また、ブロック内の 1 つの座標だけをゼロにすることはあり得るか。
   3. ペナルティ $\lambda \sum_{u} \|\mathbf{w}_u\|_2$（出力側の重みベクトル $\mathbf{w}_u$ をネットワークの隠れユニット u にわたって足し合わせたもの）は、ネットワークにどのような作用をもたらすか。
   4. [code] (a) の結果を 2 次元グリッド上の総当たり最小化と比較して、$\mathbf{z} = (0.3, 0.4)$ および $\mathbf{z} = (3, 4)$（$\lambda = 1$）について検証せよ。答えはそれぞれ $\mathbf{0}$ と $(2.4, 3.2)$ になるが、$(2.4, 3.2)$ は本節のどこに現れたか。

6. **モロー分解**。●**凸共役**とは、ψ に対して $\psi^*(\mathbf{y}) = \sup_{\mathbf{x}} \left(\mathbf{y}^\top \mathbf{x} - \psi(\mathbf{x})\right)$ で定義される関数である。真閉凸な ψ に対しては、2 つの事実がこれを近接作用素 Equation 35.72 と結びつける。すなわち、$\mathbf{x} = \operatorname{prox}_{\psi}(\mathbf{z})$ が成り立つ必要十分条件は $\mathbf{z} - \mathbf{x} \in \partial \psi(\mathbf{x})$（劣勾配の集合）であり、かつ $\mathbf{y} \in \partial \psi(\mathbf{x})$ が成り立つ必要十分条件は $\mathbf{x} \in \partial \psi^*(\mathbf{y})$ である (Parikh and Boyd 2014)。

   1. モロー分解 $\mathbf{z} = \operatorname{prox}_{\psi}(\mathbf{z}) + \operatorname{prox}_{\psi^*}(\mathbf{z})$ を導出せよ。
   2. $\psi = \lambda \|\cdot\|_1$ の共役は直方体 $\{\mathbf{y} : \|\mathbf{y}\|_\infty \leq \lambda\}$ の指示関数である。この分解により、ソフト閾値処理 Equation 35.71 が $\mathbf{z}$ からクリップされた $\mathbf{z}$ のコピーを引いた形に書けることを示せ。
   3. $\psi = \lambda \|\cdot\|_2$ の共役は ℓ_2 球（半径 λ）の指示関数であり、その近接作用素は球への射影、すなわち勾配クリッピングにおけるノルムによるクリップ写像である。この分解を用いて、クリッピングの公式からグループ Lasso の演習問題で導いたブロックソフト閾値処理の公式を復元せよ。$\mathbf{z} = (3, 4)$、$\lambda = 1$ のとき、本節に登場したどの 2 つのベクトルの和が $\mathbf{z}$ になるか。

7. [code] **加速近接勾配法**。●FISTA (Beck and Teboulle 2009) は、ISTA の更新式 Equation 35.73（$F = s + \psi$ に対するもの）にモメンタムを加えた手法である。$\mathbf{v}_1 = \mathbf{x}_0$、$t_1 = 1$ とおき、$k \geq 1$ に対して次式を反復する。

$$\begin{aligned} \mathbf{x}_{k} &= \operatorname{prox}_{\eta\psi}\!\left(\mathbf{v}_{k} - \eta \nabla s(\mathbf{v}_{k})\right), \\ t_{k+1} &= \tfrac{1}{2}\Big(1 + \sqrt{1 + 4t_{k}^{2}}\Big), \\ \mathbf{v}_{k+1} &= \mathbf{x}_{k} + \frac{t_{k} - 1}{t_{k+1}}\,(\mathbf{x}_{k} - \mathbf{x}_{k-1}). \end{aligned} \tag{35.78}$$

$\eta = 1/L$ とすると、その目的関数のギャップは $F(\mathbf{x}_{k}) - F^{\star} \le 2L\|\mathbf{x}_{0} - \mathbf{x}^{\star}\|^{2}/(k+1)^{2}$ を満たし、ISTA の $O(1/k)$ を上回る。

1. 射影の証明と同様に、近接作用素が非拡大であること、すなわち $\|\operatorname{prox}_{\eta\psi}(\mathbf{u}) - \operatorname{prox}_{\eta\psi}(\mathbf{v})\| \le \|\mathbf{u} - \mathbf{v}\|$ を、$\operatorname{prox}_{\eta\psi}(\mathbf{u})$ と $\operatorname{prox}_{\eta\psi}(\mathbf{v})$ を特徴づける 2 つの劣勾配不等式を足し合わせることで示せ。
2. 同じ Lasso 問題について ISTA のセルに FISTA を追加し、両手法の $F(\mathbf{x}_{k}) - F^{\star}$ を 100 ステップにわたって対数軸上にプロットせよ。ただし $F^{\star}$ には 10^{4} ステップの ISTA で得られた値を用いよ。10 ステップ後ではどちらの手法が先行しており、30 ステップ後ではどちらか。PGD の縮小に関する命題と (a) を用いて、ISTA が直線状に減少する理由を説明せよ。
3. FISTA の反復列に沿って $F(\mathbf{x}_{k})$ は単調減少するか。
4. $\mathbf{A}$ を 30×100 のガウスランダム行列に置き換え、同じ 3 個の非ゼロ係数を埋め込んで両手法を 1000 ステップ実行せよ。今度は k のどの範囲で FISTA が先行するか。また、$\mathbf{A}^{\top}\mathbf{A}$ が特異であるにもかかわらず、ISTA が依然として終盤に高速な線形収束段階に入るのはなぜか。

35.7 双対性

反復ソルバの進行が止まり収束が報告されたとき、返された点は本当に最適なのだろうかという自然な疑問が残る。制約のない凸関数であれば、勾配がゼロであることによって直ちに決着がつく（35.3 節）。しかし制約がある場合、最適解は通常、勾配がゼロでない境界上に位置するため（35.6 節）、この単純な判定法は使えなくなる。したがって、数値ソルバにも数学的な証明にも、最適値に対する信頼できる下界という別の道具が必要になる。本節では、そのような下界を体系的に構成する方法を展開し、KKT 条件におけるラグランジュ乗数が実際に何を測っているのかを明らかにする。

この難しさを具体的に見るため、35.6.2 節のクリッピング問題に立ち返ろう。そこでは、二乗距離 $\tfrac{1}{2}\|\mathbf{x} - \mathbf{x}_{0}\|^{2}$（目標点 $\mathbf{x}_{0} = (3, 4)$ までの距離）を単位円板 $\|\mathbf{x}\| \le 1$ 上で最小化し、最適解 $\mathbf{x}^{\star} = (0.6, 0.8)$ と最適値 8 を求めた（ここでは目標点を $\mathbf{x}_{0}$ と書き、$\mathbf{y}$ は使わない。$\mathbf{y}$ は後でラベルおよび凸共役（convex conjugate）の引数として用いるためである）。いま、反復ソルバが $\mathbf{x} = (0.55, 0.835)$ で停止したとしよう。この点は $\|\mathbf{x}\|^{2} = 0.9997 \le 1$ を満たすため実行可能点（feasible point）であり、目的関数値 8.01 を達成する。この 8.01 がほぼ最適なのか、それとも円板内の別の点で 7 に到達するのかを、どうすれば判定できるだろうか。真の最適解における勾配は $(-2.4, -3.2)$ とゼロにはほど遠く、円板内には無限に多くの点が含まれるため、1 つずつ調べることは問題外である。

中心となる発想は、制約を目的関数に折り込むことである。非負の係数 $\lambda \ge 0$ を制約関数 $\tfrac{1}{2}(\|\mathbf{x}\|^{2} - 1)$ に掛けて目的関数に加える。すべての実行可能点においてこの追加項は非正となるため、修正した目的関数を $\mathbb{R}^{2}$ **全体**にわたって最小化すれば、その値は必ず制約付き最適値以下になる。このように $\lambda \ge 0$ を選ぶごとに制約なし最小化を通じて下界が得られ、とくに $\lambda = 4$ とすると下界 8 が得られるため、ソルバが返した 8.01 は最適値から 0.01 以内にあることが直ちに

証明される。このような下界のうち最も厳しいものを探すことで、対になる最適化問題である**双対問題**（dual problem）が定義される。注目すべきことに、元の問題が非凸であっても、双対問題は常に凸問題となる。狭義実行可能点（strictly feasible point）を持つ凸問題では、双対問題の最適値は制約付き最適値と一致し、最適乗数は各制約を緩和したときに最適値がどれだけ速く改善するかを表す。

以下ではまず、この下界の考え方を用いてクリッピング問題の最適性を証明し、あらゆる最適化問題に対して成り立つ 2 行の議論である**弱双対性**（weak duality）を述べる。次に、本節で唯一繊細な幾何学的ステップを含む証明によって凸問題に対する**強双対性**（strong duality）を確立し、最適乗数を**シャドープライス**として解釈することで、重み減衰（weight decay）をノルム制約に直接結びつける。続いて、線形制約付き問題の双対を計算する便利な道具として**凸共役**を導入した後（初読時には流し読みしてもよい）、サポートベクターマシン（support vector machine、SVM）、注水法（water-filling）による電力配分、および最良の双対下界が最適値を真に下回る非凸問題という 3 つの示唆に富む例を順に見ていく。計算はすべて NumPy で扱えるほど小規模であり、プロットには `d21` を用いる。

35.7.1 双対問題

最初の目標は、クリッピング問題のいかなる実行可能点も 8 未満の値をとらないことを、実行可能点を 1 つずつ調べることなく証明することである。本節のすべての下界は、任意の実行可能点においてすべての不等式制約関数は非正である、というただ 1 つの基本的な事実に基づいている。

35.7.1.1 1 つの乗数から得られる下界

クリッピング問題を振り返ろう。目的関数 $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|^2$（ただし $\mathbf{x}_0 = (3, 4)$）を、制約 $g(\mathbf{x}) = \frac{1}{2}(\|\mathbf{x}\|^2 - 1) \le 0$ のもとで最小化する問題である。係数 $\frac{1}{2}$ を g に含めているのは、単にその勾配を $\mathbf{x}$ に一致させるためである。KKT 条件から、解はすでにわかっている。最適解は $\mathbf{x}^\star = \mathbf{x}_0/\|\mathbf{x}_0\| = (0.6, 0.8)$、最適値は $p^\star = \frac{1}{2}(5-1)^2 = 8$ であり、最適乗数は $\lambda^\star = 4$ となる。この乗数は停留条件 $\nabla f(\mathbf{x}^\star) + \lambda^\star \mathbf{x}^\star = (-2.4, -3.2) + 4\,(0.6, 0.8) = \mathbf{0}$ から定まる。

ここで、任意の乗数 $\lambda \ge 0$ と任意の実行可能点 $\mathbf{x}$ をとる。$g(\mathbf{x}) \le 0$ かつ $\lambda \ge 0$ であるから、その積 $\lambda g(\mathbf{x})$ は非正であり、したがって $f(\mathbf{x}) + \lambda g(\mathbf{x}) \le f(\mathbf{x})$ が成り立つ。左辺を $\mathbb{R}^2$ 全体にわたって最小化すれば値はさらに小さくなるか変わらないため、その制約なし最小値は高々 $f(\mathbf{x})$ である。これはすべての実行可能な $\mathbf{x}$ に対して成り立つため、最良の実行可能な $\mathbf{x}$ に対しても成り立つ。

$$\min_{\mathbf{x} \in \mathbb{R}^2} \big[f(\mathbf{x}) + \lambda g(\mathbf{x}) \big] \;\le\; \min_{\|\mathbf{x}\| \le 1} f(\mathbf{x}) \;=\; p^\star \qquad \text{for every } \lambda \ge 0.$$

左辺を具体的に計算してみよう。最小化する関数はラグランジアン $\mathcal{L}(\mathbf{x}, \lambda) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|^2 + \frac{\lambda}{2}(\|\mathbf{x}\|^2 - 1)$ である。任意の $\lambda \ge 0$ に対し、これは $\mathbf{x}$ に関する凸二次関数でヘッセ行列は $(1+\lambda)\mathbf{I}$ となるため、一意な最小点（minimizer）は勾配 $(\mathbf{x} - \mathbf{x}_0) + \lambda \mathbf{x}$ がゼロになる点、すなわち $\mathbf{x}(\lambda) = \mathbf{x}_0/(1+\lambda)$ で達成される。λ を大きくするほど、$\mathbf{x}_0$ は原点に向かってより強く縮小される。この $\mathbf{x}(\lambda)$ をラグランジアンに代入すると（$\mathbf{x}(\lambda) - \mathbf{x}_0 = -\lambda \mathbf{x}_0/(1+\lambda)$ および $\|\mathbf{x}_0\|^2 = 25$ を用いる）、$\frac{1}{2}\|\mathbf{x}(\lambda) - \mathbf{x}_0\|^2 = \frac{25}{2}\lambda^2/(1+\lambda)^2$ および $\frac{\lambda}{2}\|\mathbf{x}(\lambda)\|^2 = \frac{25}{2}\lambda/(1+\lambda)^2$ となる。これら 2 つの分数を合わせると因子 $(1+\lambda)$ が約分され、最小値は λ の単純な関数として残る。

$$q(\lambda) \;=\; \frac{25}{2}\,\frac{\lambda}{1+\lambda} \;-\; \frac{\lambda}{2}. \tag{35.79}$$

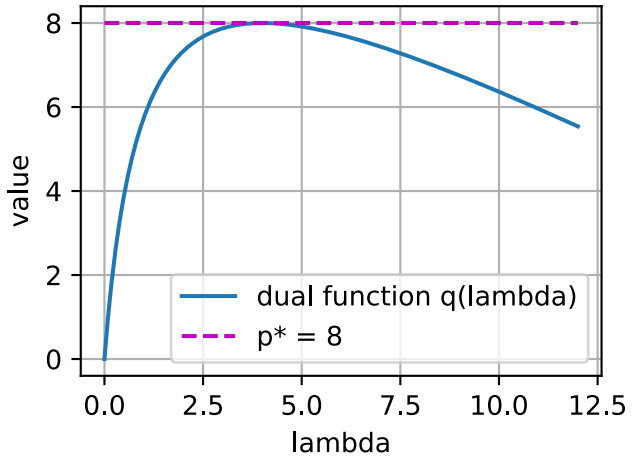
これら 2 つの項は互いに逆方向へ引っ張り合う。λ が大きくなるにつれて、第 1 項は 12.5 に向かって増加する一方、第 2 項（定数 $-\frac{1}{2}$ が g の中に含まれることに由来する項）は際限なく線形に減少する。最も厳しい下界は、これら 2 つの効果が釣り合う点にある。

いくつか数値を代入してみると、この釣り合いが具体的によくわかる。$q(0) = 0$、$q(1) = 5.75$、$q(2) = 7.33$、$q(3) = 7.875$、$q(4) = 8$、$q(5) = 7.92$、$q(8) = 7.11$ である。これらの値はどれ 1 つをとっても $p^\star$ に対する正当な下界となっている。微分すると $q'(\lambda) = \frac{25}{2(1+\lambda)^2} - \frac{1}{2}$ となり、これは $(1+\lambda)^2 = 25$、すなわち $\lambda = 4$ のときにゼロになる。さらに $q''(\lambda) = -25/(1+\lambda)^3 < 0$ であるから、$\lambda = 4$ が一意な最大点であり、$q(4) = 10 - 2 = 8 = p^\star$ を達成する。最良の下界は真の最適値に一致し、それを達成する乗数は KKT 乗数にほかならない。これにより、ソルバが得た値 8.01 は最適値から 0.01 以内にあることが保証される。最適化していない $\lambda = 3$ のような適当な値でさえ、すでに最適値から 0.135 以内にあることを保証してくれる。

なぜ $\lambda = 4$ で最適値に到達するのだろうか。ラグランジアンの最小点における制約値 $g(\mathbf{x}(\lambda)) = \frac{1}{2}(25/(1+\lambda)^2 - 1)$ に注目しよう。$\lambda < 4$ のときは乗数が弱すぎて最小点を単位円板の内側へ引き込めず（たとえば $\mathbf{x}(1) = (1.5, 2)$ のノルムは 2.5 である）、$g(\mathbf{x}(\lambda)) > 0$ となって下界は実行不可能な点を反映したものになる。逆に $\lambda > 4$ のときは乗数が最小点を内側へ引き込みすぎてしまい（たとえば $\mathbf{x}(8)$ のノルムは $\frac{5}{9}$ である）、$\lambda g(\mathbf{x}(\lambda)) < 0$ が下界を引き下げてしまう。ちょうど $\lambda = 4$ のときに最小点 $\mathbf{x}(4) = (0.6, 0.8)$ が境界円上に乗り、ペナルティ項 $\lambda g(\mathbf{x}(\lambda))$ がゼロとなって、下界が実行可能点の目的関数値と一致するのである。

$q(\lambda)$ と $p^\star$ を λ の値のグリッド上で並べてプロットし、これらの下界がどのように振る舞うかを確認しよう。

```
x0 = np.array([3.0, 4.0])                          # the point to clip
f = lambda x: 0.5 * np.sum((x - x0) ** 2)          # objective
g = lambda x: 0.5 * (np.sum(x ** 2) - 1.0)         # constraint g(x) <= 0
x_of = lambda lam: x0 / (1 + lam)                  # minimizer of L(., lam)
q = lambda lam: f(x_of(lam)) + lam * g(x_of(lam))  # dual function
p_star = f(x0 / np.linalg.norm(x0))                # value at the clipped point
lams = np.linspace(0.0, 12.0, 1201)
qs = np.array([q(l) for l in lams])
d2l.plot(lams, [qs, np.full_like(lams, p_star)], 'lambda', 'value',
         legend=['dual function q(lambda)', 'p* = 8'])
```



得られた曲線は凹関数であり、至るところで水平線 $p^\star = 8$ の下側に留まりつつ、$\lambda = 4$ で 1 度だけ接する。驚くべきことに、ここでの凹性は円板の幾何形状とは何の関係もない。以下で示すように、双対関数（dual function）は**あらゆる**最適化問題に対して凹関数となる。

35.7.1.2 ラグランジュ双対関数

上の下界の議論には、クリッピング問題に固有の性質は何ひとつ使われていなかった。同じ構成法を 35.6.2 節の一般の制約付き問題に適用しよう。

$$\min_{\mathbf{x}\in\mathcal{D}} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \le 0,\; i = 1,\dots,m, \qquad h_j(\mathbf{x}) = 0,\; j = 1,\dots,p,$$

この問題の最適値を $p^\star$ とし、すべての実行可能点にわたる f の下限として定義する（実行可能点が存在しない場合は $p^\star = +\infty$ とする）。ここで $\mathcal{D}$ は f およびすべての制約関数が定義される定義域である。これは通常単なる $\mathbb{R}^n$ であるが、本節末尾の双対ギャップ（duality gap）の例で $[0,1]$ に対して行うように、ある集合をハード制約として残す場合はその集合となる。ラグランジアンは各制約に対して 1 つずつ乗数を導入し、次式で定義される。

$$\mathcal{L}(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \, g_i(\mathbf{x}) + \sum_{j=1}^{p} \nu_j \, h_j(\mathbf{x}),$$

ここで、不等式制約には $\lambda_i \ge 0$ を対応させ、等式制約には符号を制限しない $\nu_j \in \mathbb{R}$ を対応させる。また、$\lambda \succeq 0$ と書いて、$\lambda_i \ge 0$ がすべての座標 i について成り立つことを表す。

乗数を固定してラグランジアンを $\mathbf{x}$ について最小化すると、下界の構成は (λ, ν) の関数になる。**ラグランジュ双対関数**（Lagrange dual function）は次式で定義される。

$$q(\lambda, \nu) = \inf_{\mathbf{x}\in\mathcal{D}} \; \mathcal{L}(\mathbf{x}, \lambda, \nu). \tag{35.80}$$

言葉で言えば、$q(\lambda,\nu)$ とは、f がどこまで小さくなりうるかを、ハード制約を取り除いて λ と ν で重み付けした線形ペナルティに置き換えたもとで問うものである。クリッピング問題では、これにより Equation 35.79 が得られた。一般に、この下限は達成されるとは限らず、$-\infty$ になることさえある。たとえば、x を制約 $-x \le 0$ のもとで最小化する場合を考えよう。ラグランジアン $(1-\lambda)x$ は $\lambda = 1$ でない限り下に有界でないため、$q(1) = 0$ であり、$q(\lambda) = -\infty$ がそれ以外のすべての λ に対して成り立つ。$-\infty$ という下界は自明に正しいが何の役にも立たない。$q(\lambda,\nu) > -\infty$ となる乗数の集合を q の**実効定義域**（effective domain）と呼ぶ。後ほどサポートベクターマシンの双対問題を導出する際、この暗黙の定義域制限は明示的な等式制約として現れる。

q の 2 つの基本的な性質は、f、g_i、h_j に凸性や微分可能性すら含むいかなる仮定も置かずに成り立つ。

命題（双対関数は凹関数である）。 **任意の主問題（primal problem）に対し、$q(\lambda,\nu)$（Equation 35.80）は凹関数である。**

証明。 任意の $\mathbf{x} \in \mathcal{D}$ を固定する。乗数は $\mathcal{L}$ に線形に入るため、写像 $(\lambda,\nu) \mapsto \mathcal{L}(\mathbf{x},\lambda,\nu)$ はアフィンである。双対関数 q は、このアフィン関数族の各点ごとの下限である。したがって、その**ハイポグラフ**（グラフ上またはその下側にある点の集合）は、これらのアフィン関数の下側にある半空間の共通部分となる。凸集合の任意の共通部分は凸集合であるため、q は凹関数である。■

Figure 35.25 に、クリッピング問題に対するこの構成を示す。各破線は固定された 1 つの $\mathbf{x}$ に対するラグランジアンを λ の関数として描いたものであり、q はそれらすべての直線の下側に留まる曲線である。クリッピング問題の目的関数はたまたま凸関数であるが、上の議論ではその性質を一切使っていない。直線群の下側包絡線（lower envelope）は、f と g がどのような形状であっても凹関数になる。

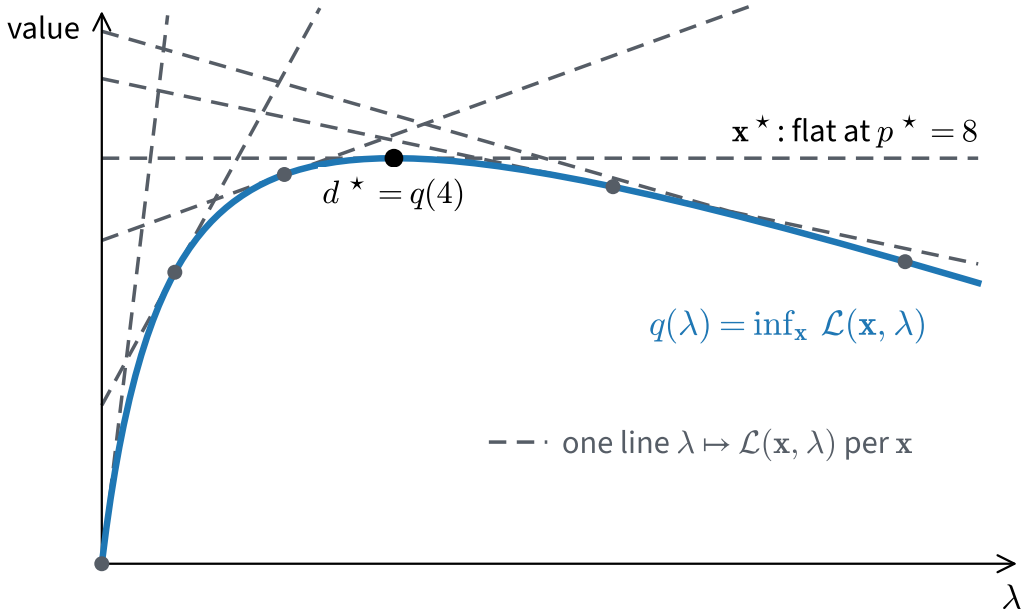


Figure 35.25: 下側包絡線としてのクリッピング問題の双対関数。固定された各 $\mathbf{x}$ は 1 本の直線 $\lambda \mapsto \mathcal{L}(\mathbf{x}, \lambda)$（破線）を与え、$q(\lambda)$ はそれらの直線が λ でとる最小値となる。描かれている直線は最小点 $\mathbf{x}(\lambda) = \mathbf{x}_0/(1+\lambda)$（いくつかの λ の値に対するもの）に対応し、それぞれが q に自身の λ で接している（黒丸）。$\mathbf{x}^\star = \mathbf{x}(4)$ の直線は $g(\mathbf{x}^\star) = 0$ のため水平であり、q に最大値 $d^\star = q(4) = p^\star = 8$ で接する。

命題（弱双対性）。$\mathbf{x}$ が実行可能であり $\lambda \succeq 0$ ならば、$q(\lambda, \nu) \le f(\mathbf{x})$ が成り立つ。したがって、次式を得る。

$$d^\star \;=\; \sup_{\lambda \succeq 0,\, \nu} q(\lambda, \nu) \;\le\; \inf_{\mathbf{x}\ \text{feasible}} f(\mathbf{x}) \;=\; p^\star. \tag{35.81}$$

証明。任意の実行可能な $\mathbf{x}$ と任意の $\lambda \succeq 0$ に対し、$\lambda_i g_i(\mathbf{x}) \le 0$（$\lambda_i \ge 0$ かつ $g_i(\mathbf{x}) \le 0$ であるため）および $\nu_j h_j(\mathbf{x}) = 0$ が成り立つ。したがって、次式が成り立つ。

$$q(\lambda, \nu) \;\le\; \mathcal{L}(\mathbf{x}, \lambda, \nu) \;=\; f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) + \sum_j \nu_j h_j(\mathbf{x}) \;\le\; f(\mathbf{x}).$$

左辺で (λ, ν)（ただし $\lambda \succeq 0$）にわたる上限をとり、右辺で実行可能な $\mathbf{x}$ にわたる下限をとれば、Equation 35.81 が得られる。■

弱双対性は、クリッピング問題におけるすべての $q(\lambda)$ を保証付きの下界に変えた 2 行の不等式にほかならない。またこれは、35.6.2 節における KKT 条件の十分性を支える仕組みその

ものでもある。そこでは同じ不等式の連鎖を KKT 乗数において評価することで、相補性条件（complementary slackness）により両方の不等号が等号で成立する。

$q(\lambda, \nu)$ を $\lambda \succeq 0$ および ν について最大化することを**双対問題**と呼ぶ。これは凸集合上で凹関数を最大化する問題であるため、たとえ主問題がどれほど強い非凸性を持っていても、双対問題は**常に**凸最適化問題となる。その最適値 $d^\star$ はこの構成法が与えうる最も厳しい下界であり、双対実行可能なすべての (λ, ν) が正当な下界を与える。非負の差 $p^\star - d^\star \geq 0$ を**双対ギャップ**と呼ぶ。クリッピング問題ではこのギャップはゼロであったが、本節最後の具体例では真に正のギャップが生じる。となると次に生じる自然な問いは、ギャップがゼロになることを事前に保証できるのはいつか、ということである。

35.7.2 強双対性

クリッピング問題では、最良の双対下界が主最適値と厳密に一致した。しかし非凸問題では、双対下界が主最適値に真に届かず、解の一部しか保証できないことがある。幸い、凸性とただ 1 つの狭義実行可能点という 2 つの穏やかな条件さえあれば、このギャップを埋めるには十分である。達成可能な制約値と目的関数値を単純な幾何学的視点から眺めると、なぜこの両方が必要なのかがよくわかる。

35.7.2.1 スレーター条件

命題（スレーター条件は強双対性を導く）。 問題が凸であり（f とすべての g_i が凸、すべての h_j がアフィン）、スレーター条件（35.6.2.3 節）を満たすとする。すなわち、実行可能点 $\bar{\mathbf{x}}$ が $\mathcal{D}$ の相対的内部（relative interior）に存在し、$g_i(\bar{\mathbf{x}}) < 0$ がアフィンでないすべての不等式制約 g_i に対して成り立つとする。このとき $d^\star = p^\star$ が成り立ち、さらに $p^\star$ が有限であれば双対最適解が達成される。

これは 35.6.2 節で凸問題に対して導入した制約想定（constraint qualification）と同じものである。すなわち、湾曲したすべての不等式制約の真に内側に少なくとも 1 つの点が存在しなければならない（多次元における完全な証明については Boyd and Vandenberghe (2004) の 5.3.2 節を参照）。クリッピング問題はスレーター条件を容易に満たす。f と g はともに凸であり、原点 $\mathbf{0}$ は $g(\mathbf{0}) = -\tfrac{1}{2} < 0$ を満たすからである。したがって、導関数を 1 度も計算する前から $q(4) = p^\star$ は保証されていたのである。

凸性と狭義実行可能性が**なぜ**ギャップを埋めるのかを理解するため、単一の不等式制約 $g(\mathbf{x}) \leq 0$ を持つ問題を考えよう。すべての点 $\mathbf{x} \in \mathcal{D}$ をその制約値と目的関数値の組 $(g(\mathbf{x}), f(\mathbf{x}))$ に写し、これらの組の上側および右側にあるすべての点を集めた集合を作る。

$$\mathcal{A} = \big\{(u, t) : u \geq g(\mathbf{x}),\ t \geq f(\mathbf{x}) \text{ for some } \mathbf{x} \in \mathcal{D}\big\}.$$

この (u, t) 平面において実行可能点は $u \leq 0$ に対応するため、主最適値 $p^\star$ とは単に高さ t のうち $\mathcal{A}$ が縦軸 $u = 0$ 上で到達する最小値にほかならない。いま $\lambda \geq 0$ を固定する。すべての (u, t) のうち与えられた $\mathbf{x}$ から生じるものの上では、線形関数 $t + \lambda u$ は $(g(\mathbf{x}), f(\mathbf{x}))$ で最小となるため、$q(\lambda) = \inf_{(u,t) \in \mathcal{A}} (t + \lambda u)$ となる。幾何学的には、直線 $t + \lambda u = c$（傾き $-\lambda$）を下から押し上げ、ちょうど $\mathcal{A}$ を下から支えるところで止めたときの縦軸 $u = 0$ 上の切片が $q(\lambda)$ である。集合 $\mathcal{A}$ 全体がこの支持直線（supporting line）の上または上側に位置するため、切片 $q(\lambda)$ が $p^\star$ を超えることは決してない。これが弱双対性を視覚化した姿である（Figure 35.26）。

この幾何学的描像があれば、単一の不等式制約に対する強双対性の証明は見通しのよいものになる。

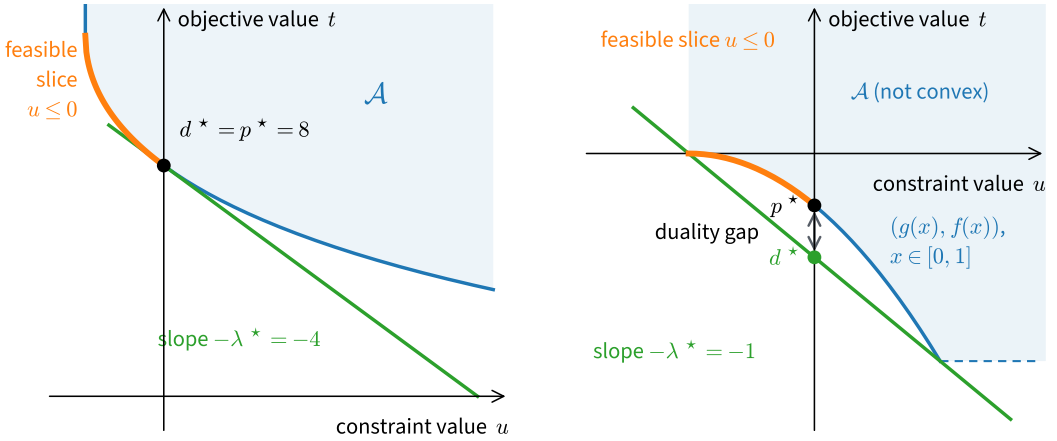


Figure 35.26: 双対性の支持直線による幾何構造。各 $\mathbf{x}$ は $(g(\mathbf{x}), f(\mathbf{x}))$ に写され、網掛け領域 $\mathcal{A}$ はこれらの組と、その上側および右側にあるすべての点を含む。双対関数を λ で評価することは、傾き $-\lambda$ の直線を $\mathcal{A}$ を下から支えるまで押し上げることに相当し、$u = 0$ におけるその切片が $q(\lambda)$ となる。(a) クリッピング問題：$\mathcal{A}$ は凸集合で $\mathbf{0}$ は狭義実行可能点であるため、最適な支持直線は $(0, p^\star)$ に傾き -4 で接し、$d^\star = p^\star = 8$ を与える。(b) $-x^2$ を $[0,1]$ 上で制約 $x \le \frac{1}{2}$ のもとで最小化する問題：$\mathcal{A}$ が非凸であるため、最も高い支持直線でも $(0, p^\star)$ の真に下側を通り、$d^\star = -\frac{1}{2} < -\frac{1}{4} = p^\star$ となる。

証明 ($g(\bar{\mathbf{x}}) < 0$ かつ $p^\star$ が有限である単一不等式制約の場合)。第 1 に、$\mathcal{A}$ は凸集合である。任意の 2 点 (u_1, t_1) と (u_2, t_2) を $\mathcal{A}$ からとり（それぞれ $\mathbf{x}_1$ と $\mathbf{x}_2$ に対応するとする）、任意の $\theta \in [0,1]$ をとる。g の凸性より、内分点 $\mathbf{x}_\theta = \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2$ は $g(\mathbf{x}_\theta) \le \theta g(\mathbf{x}_1) + (1-\theta) g(\mathbf{x}_2) \le \theta u_1 + (1-\theta) u_2$ を満たし、同じ不等式が f に対しても成り立つ。ゆえに (u_1, t_1) と (u_2, t_2) の凸結合（convex combination）は $\mathcal{A}$ に属する。

第 2 に、$(0, p^\star)$ は $\mathcal{A}$ の境界上にある。すなわち、この点は $\mathcal{A}$ 内の点列の極限である一方、$(0, p^\star - \varepsilon) \notin \mathcal{A}$ がすべての $\varepsilon > 0$ に対して成り立つ。凸集合の任意の境界点には支持直線が存在する。すなわち、ゼロでないベクトル $(\lambda, \gamma) \neq (0,0)$ が存在して、$\gamma t + \lambda u \ge \gamma p^\star$ がすべての $(u,t) \in \mathcal{A}$ に対して成り立つ（Boyd and Vandenberghe (2004) の 2.5.2 節を参照。これは 35.3 節で劣勾配（subgradient）を導いた幾何学的事実と同じものである）。$\mathcal{A}$ は上方向および右方向に限りなく広がっているため、$\gamma \ge 0$ かつ $\lambda \ge 0$ でなければならない。

では、$\gamma = 0$ となることはありうるだろうか。もしそうだとすると、支持直線は垂直で $\lambda > 0$ となり、$\lambda u \ge 0$（したがって $u \ge 0$）が $\mathcal{A}$ 全体で成り立つことになる。しかしこれは、狭義実行可能点 $\bar{\mathbf{x}}$ が $u = g(\bar{\mathbf{x}}) < 0$ を与えることと矛盾する。したがって $\gamma > 0$ である。両辺を γ で割ると、$t + (\lambda/\gamma)\, u \ge p^\star$ がすべての $(u,t) \in \mathcal{A}$ に対して得られ、これは $q(\lambda/\gamma) \ge p^\star$ を意味する。弱双対性と合わせれば $q(\lambda/\gamma) = p^\star$ となり、双対最適解が達成されてその値は $p^\star$ に一致する。■

この証明において、2 つの仮定がいかにきれいに役割分担しているかに注目しよう。凸性は $\mathcal{A}$ を凸集合にし、$(0, p^\star)$ における支持直線の存在を保証するために 1 度だけ使われる。スレーター条件は、その支持直線が（無限大の乗数に対応する）垂直線になるのを防ぐために 1 度だけ使われる。Figure 35.26 のパネル (a) のクリッピング問題では、ノルム $\|\mathbf{x}\| = r$ の点は最小目的関数値 $\frac{1}{2}(5-r)^2$ と制約値 $u = \frac{1}{2}(r^2 - 1)$ を持つため、$\mathcal{A}$ の下側境界は曲線 $t = \frac{1}{2}\big(5 - \sqrt{1+2u}\big)^2$ となる。その $u = 0$ における傾きは -4 であり、傾き $-\lambda^\star = -4$ の最適な支持直線と一致する。パネル (b) は本節末尾で扱う非凸の例を先取りしたものである。そこでは $\mathcal{A}$ が逆向きに湾曲して

いるため、$\mathcal{A}$ の下側にあるどの直線も $(0, p^\star)$ の真に下を通過してしまう。

35.7.2.2 鞍点

強双対性は、双対関数に言及することすらなく、ラグランジアンのみを用いて言い換えることもできる。実際、主問題は min–max 問題として書き直せる。任意の $\mathbf{x} \in \mathcal{D}$ に対して、次式が成り立つ。

$$\sup_{\lambda \succeq 0,\, \nu} \mathcal{L}(\mathbf{x}, \lambda, \nu) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \text{ is feasible,} \\ +\infty & \text{otherwise.} \end{cases}$$

もし $\mathbf{x}$ がいずれかの制約に違反していれば、対応する乗数を大きくすることで $\mathcal{L}$ を $+\infty$ へ発散させられる。一方、$\mathbf{x}$ が実行可能であれば、乗数側にできる最善の選択は余裕のあるすべての制約に対して $\lambda_i = 0$ と置くことであり、結果として $f(\mathbf{x})$ が残る。したがって $p^\star = \inf_{\mathbf{x}} \sup_{\lambda \succeq 0, \nu} \mathcal{L}$ となる一方、定義より $d^\star = \sup_{\lambda \succeq 0, \nu} \inf_{\mathbf{x}} \mathcal{L}$ である。弱双対性とは、任意の 2 変数関数に対して成り立つ普遍的な不等式 $\sup \inf \le \inf \sup$ にほかならない。強双対性は、最小化と最大化の順序を入れ替えても値が変わらないことを主張する。両方の最適解が達成されるとき、この順序交換可能性はラグランジアンに鞍点が存在することと同値である。

命題（強双対性と鞍点）。 強双対性が成り立ち、主最適解 $\mathbf{x}^\star$ と双対最適解 $(\lambda^\star, \nu^\star)$ の両方が達成されるとする。このとき $(\mathbf{x}^\star, \lambda^\star, \nu^\star)$ は、ラグランジアンの 鞍点（saddle point）である。すなわち、すべての $\mathbf{x} \in \mathcal{D}$、すべての $\lambda \succeq 0$、およびすべての ν に対して、次式が成り立つ。

$$\mathcal{L}(\mathbf{x}^\star, \lambda, \nu) \;\le\; \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star) \;\le\; \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star). \tag{35.82}$$

逆に、$\mathcal{L}$ の $\mathbf{x} \in \mathcal{D}$ および $(\lambda \succeq 0, \nu)$ に関する任意の鞍点は双対ギャップをゼロにし、その鞍点値は $p^\star = d^\star$ となる。

証明。 q の定義と弱双対性より、次式が成り立つ。

$$d^\star = q(\lambda^\star, \nu^\star) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star) \;\le\; \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star) \;\le\; f(\mathbf{x}^\star) = p^\star,$$

ここで最後の不等号は、$\lambda_i^\star \ge 0$ かつ $g_i(\mathbf{x}^\star) \le 0$ によりすべてのペナルティ項が非正となることから従う。強双対性 $(d^\star = p^\star)$ により、この不等式の連鎖はすべて等号で成り立たなければならない。左側の等号は、$\mathbf{x}^\star$ が $\mathcal{L}(\cdot, \lambda^\star, \nu^\star)$ を最小化することを意味し、Equation 35.82 の右側の不等式を与える。右側の等号からは $\sum_i \lambda_i^\star g_i(\mathbf{x}^\star) = 0$ が従い、これは相補性条件にほかならない。ゆえに、任意の $\lambda \succeq 0$ と ν に対して $\mathcal{L}(\mathbf{x}^\star, \lambda, \nu) = f(\mathbf{x}^\star) + \sum_i \lambda_i g_i(\mathbf{x}^\star) \le f(\mathbf{x}^\star) = \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star)$ が成り立ち、Equation 35.82 の左側の不等式が得られる。

逆に、$(\mathbf{x}^\star, \lambda^\star, \nu^\star)$ が鞍点であれば、次式が成り立つ。

$$\begin{aligned} \inf_{\mathbf{x}} \sup_{\lambda \succeq 0, \nu} \mathcal{L} &\le \sup_{\lambda \succeq 0, \nu} \mathcal{L}(\mathbf{x}^\star, \lambda, \nu) = \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star) \\ &= \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star) \le \sup_{\lambda \succeq 0, \nu} \inf_{\mathbf{x}} \mathcal{L}, \end{aligned}$$

これより $p^\star \le d^\star$ がわかる。逆向きの不等式は弱双対性から得られるため、$p^\star = d^\star$ が成り立つ。 ■

クリッピングの例では、鞍点は $(\mathbf{x}^\star, \lambda^\star) = ((0.6, 0.8), 4)$ である。$\mathbf{x}$ を $\mathbf{x}^\star$ からずらすと $\mathcal{L}(\cdot, 4)$ は増加し、一方で λ を動かしても $\mathcal{L}(\mathbf{x}^\star, \lambda) = 8$ は $g(\mathbf{x}^\star) = 0$ のため一定のまま保たれる。

強双対性は実用上 3 つの利点をもたらす。

- **双対問題は主問題より単純または小規模になりうる。**サポートベクターマシンの双対問題はデータ点ごとに 1 つの変数を持ち、単純な符号制約と高々 1 つの等式制約しか持たない。また、演習問題 8 のリッジ回帰の双対問題は、特徴量の次元ではなくデータ例の数に等しいサイズの連立一次方程式（linear system）に帰着する。
- **任意の双対実行可能点が停止基準を与える。**任意の主実行可能な $\mathbf{x}$ と双対実行可能な (λ, ν) に対し、差 $f(\mathbf{x}) - q(\lambda, \nu)$ は $f(\mathbf{x})$ が $p^\star$ からどれだけ離れうるかの上界を与える。
- **双対解から主問題の解を復元できる。**@eq-mdl-opt-saddle-point より、$\mathbf{x}^\star$ は $\mathcal{L}(\cdot, \lambda^\star, \nu^\star)$ の最小点集合の**中に**含まれる。その最小点が一意であるとき（たとえば f が狭義凸のとき）、直ちに $\mathbf{x}^\star$ が得られる（クリッピング問題で $\mathbf{x}(4) = \mathbf{x}_0/5 = \mathbf{x}^\star$ となったのと同様である）。最小点が一意でない場合は、実行可能性と相補性条件によって最小点集合の中から $\mathbf{x}^\star$ が特定される。線形計画問題では $\mathcal{L}(\cdot, \lambda^\star)$ は $\mathbf{x}$ に関して平坦になり、後述するオフセットにペナルティを課さない SVM では、ラグランジアンの最小化だけではオフセットが定まらず、相補性条件がそれを決定する。

鞍点の命題はまた、35.6.2 節で凸問題に対する十分条件として示した KKT 条件の必要性にも決着をつける。微分可能な f、g_i、h_j を $\mathcal{D} = \mathbb{R}^n$ 上に持つ凸問題がスレーター条件を満たし、最適解を $\mathbf{x}^\star$ で達成するとしよう。このとき強双対性が成り立ち、達成された双対最適解 $(\lambda^\star, \nu^\star)$ が存在する。$\mathbf{x}^\star$ は $\mathcal{L}(\cdot, \lambda^\star, \nu^\star)$ を $\mathbb{R}^n$ 全体にわたって最小化するため、その点において勾配 $\nabla_{\mathbf{x}} \mathcal{L}$ はゼロになり、さらに今見たとおり相補性条件も成り立つ。主実行可能性および双対実行可能性と合わせれば、これらはまさに 4 つの KKT 条件そのものである。すなわちスレーター条件のもとで、KKT 条件は必要かつ十分である。

最後に、この min–max 構造は明示的な制約を持たない深層学習の問題にも現れる。敵対的生成ネットワーク（16.1 節）や敵対的訓練（adversarial training）は、直接 $\min_\theta \max_\phi$ ゲームとして定式化される (Goodfellow et al. 2014; Madry et al. 2018)。鞍点が存在しなければ、min max 値と max min 値の間にギャップが生じうる。さらに鞍点が存在する場合であっても、最小化変数に対する勾配降下ステップと最大化変数に対する勾配上昇ステップを同時に行うと収束しないことがある。実際、原点を鞍点とする単純な双線形目的関数 xy に対しては、同時降下上昇法は任意の正のステップサイズで外側へ渦巻状に発散する（演習問題 5）。

35.7.3 価格としての乗数

制約にはどれほどの価値があるだろうか。クリッピング問題において、単位円板をわずかに広げたとき、$\mathbf{x}_0$ にどれだけ近づけるだろうか。最適なラグランジュ乗数は、この問いに直接答えてくれる。さらに都合のよいことに、同じ感度解析の議論から、損失が凸である場合には重み減衰と重みに対するノルム制約がまったく同じ解の族を描くことも明らかになる。

35.7.3.1 シャドープライス

各制約の価値を測るため、右辺に摂動を加えて最適目的関数値がどのように変化するかを追跡しよう。摂動ベクトル $\mathbf{u} \in \mathbb{R}^m$ と $\mathbf{v} \in \mathbb{R}^p$ に対し、摂動後の最適値を次式で定義する。

$$p^\star(\mathbf{u}, \mathbf{v}) = \inf \{ f(\mathbf{x}) : g_i(\mathbf{x}) \le u_i,\ h_j(\mathbf{x}) = v_j \}.$$

$(\mathbf{0}, \mathbf{0})$ では元の最適値 $p^\star(\mathbf{0}, \mathbf{0})$ が復元され、一方で $u_i > 0$ とすることは第 i 不等式制約を緩和することに相当する。実はこの関数はすでに幾何学的に目にしている。単一の不等式制約の場合、

$p^\star(u)$ は高さ t のうち Figure 35.26 の集合 $\mathcal{A}$ が横座標 u で到達する最小値にほかならない。$\mathcal{A}$ の下側境界が $p^\star(u)$ のグラフであり、傾き $-\lambda^\star$ の支持直線は至るところでその下側にある。次の命題は、この幾何学的不等式を任意の個数の制約へと拡張するものである。

命題（乗数はシャドープライスである）。 強双対性が成り立ち、双対最適解 $(\lambda^\star, \nu^\star)$ が存在するとする。このとき、すべての $\mathbf{u}$ と $\mathbf{v}$ に対して次式が成り立つ。

$$p^\star(\mathbf{u}, \mathbf{v}) \;\geq\; p^\star(\mathbf{0}, \mathbf{0}) - {\lambda^\star}^{\top}\mathbf{u} - {\nu^\star}^{\top}\mathbf{v}, \tag{35.83}$$

さらに $p^\star$ が $(\mathbf{0}, \mathbf{0})$ で微分可能であれば、その点において $\nabla_{\mathbf{u}} p^\star = -\lambda^\star$ および $\nabla_{\mathbf{v}} p^\star = -\nu^\star$ が成り立つ。

証明。 $\mathbf{x}$ を摂動後の問題に対する任意の実行可能点とすると、$g_i(\mathbf{x}) \leq u_i$ および $h_j(\mathbf{x}) = v_j$ が成り立つ。強双対性、下限としての q の定義、$\lambda_i^\star \geq 0$、および摂動後の制約を用いると、次式が得られる。

$$\begin{aligned} p^\star(\mathbf{0}, \mathbf{0}) = q(\lambda^\star, \nu^\star) &\leq f(\mathbf{x}) + \sum_i \lambda_i^\star g_i(\mathbf{x}) + \sum_j \nu_j^\star h_j(\mathbf{x}) \\ &\leq f(\mathbf{x}) + {\lambda^\star}^{\top}\mathbf{u} + {\nu^\star}^{\top}\mathbf{v}. \end{aligned}$$

摂動後の問題で実行可能なすべての $\mathbf{x}$ にわたる下限をとれば、Equation 35.83 が得られる。さらに $p^\star$ が原点で微分可能であれば、非負関数 $p^\star(\mathbf{u}, \mathbf{v}) - p^\star(\mathbf{0}, \mathbf{0}) + {\lambda^\star}^{\top}\mathbf{u} + {\nu^\star}^{\top}\mathbf{v}$ は大域的最小値 0 を $(\mathbf{0}, \mathbf{0})$ で達成するため、その点において勾配はゼロでなければならない。$\blacksquare$

最適乗数 $\lambda_i^\star$ は制約 i の**シャドープライス**と呼ばれ、その制約を緩和したときに最適値が改善する変化率を測る。なお、Equation 35.83 は大域的な不等式であり、**いかなる**大きさの摂動に対しても成り立つ。制約 i を $u_i > 0$ だけ緩和しても最適値は高々 $\lambda_i^\star u_i$ しか下がらない一方、同じ量だけ厳しくすると最適値は少なくとも $\lambda_i^\star u_i$ だけ上がる。一次近似では、$p^\star$ が微分可能なとき変化量は $-\lambda_i^\star u_i$ となる。余裕のある制約では相補性条件により $\lambda_i^\star = 0$ となるが、これも直感と一致する。有効になっていない制約を緩めても最適値は改善しないからである。

クリッピング問題でこれを確認してみよう。$g(\mathbf{x}) \leq 0$ を $g(\mathbf{x}) \leq u$ に緩和すると円板は半径 $r = \sqrt{1 + 2u}$ へ拡大し、$p^\star(u) = \tfrac{1}{2}\bigl(5 - \sqrt{1+2u}\bigr)^2$（先ほど求めた $\mathcal{A}$ の下側境界）が得られる。その $u = 0$ における導関数は確かに $-4 = -\lambda^\star$ となっている。制約を $u = \tfrac{1}{2}$ だけ緩和すると（半径 $\sqrt{2}$ の円板）、新しい最適値は $p^\star(\tfrac{1}{2}) = \tfrac{1}{2}(5 - \sqrt{2})^2 \approx 6.43$ となり、Equation 35.83 が保証する大域的下界 $8 - 4 \cdot \tfrac{1}{2} = 6$ をしっかり上回っている。

1 つ留意しておきたい繊細な点がある。乗数は制約のスケーリングの仕方に依存するということである。ここでの $\lambda^\star = 4$ が価格付けしているのは u、すなわち二乗半径の**半分**であり、半径 r そのものではない。半径 1 単位あたりの変化率を連鎖律で求めると $\mathrm{d}p^\star/\mathrm{d}r = -(5 - r)$ となり、これは r に u に関する導関数を掛けたものに等しく、$-\lambda^\star$ と一致するのはたまたま $r = 1$ だからにすぎない。制約を定数倍すれば乗数もそれに応じてスケーリングされる。たとえば $\|\mathbf{x}\|^2 - 1 \leq 0$ のように係数 $\tfrac{1}{2}$ を付けずに書けば、乗数は半分の 2 になる。

35.6.1.3 節の最近接点問題は等式制約の場合の例となっている。そこでは、$\tfrac{1}{2}\|\mathbf{x}\|^2$ の超平面 $\mathbf{a}^{\top}\mathbf{x} = b$ 上での最小値は $p^\star(b) = b^2/(2\|\mathbf{a}\|^2)$ であり、$\partial p^\star/\partial b = -\nu^\star$ は Equation 35.83 を摂動 $h(\mathbf{x}) = \mathbf{a}^{\top}\mathbf{x} - b = v$（超平面を $b + v$ へ平行移動させるもの）に適用した結果そのものである。これにより、同節で数値的に確認した $0.02 = 0.2 \cdot 0.1$ という増加量が説明される。

35.7.3.2 重み減衰とノルム制約

深層学習では、モデルの複雑さを制御するために、重み減衰（2.7 節）を訓練損失 $\ell(\mathbf{w})$ に加えるか、あるいは重みにハードなノルム制約を課すことがよく行われる。本書の記法に従えば、これら 2 つの問題は次のように書ける。

$$\min_{\mathbf{w}} \ \ell(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \qquad \text{and} \qquad \min_{\mathbf{w}} \ \ell(\mathbf{w}) \quad \text{subject to} \quad \tfrac{1}{2}\|\mathbf{w}\|^2 \le \tfrac{1}{2}r^2.$$

ペナルティ付き問題の停留点（stationary point）では $\nabla \ell(\mathbf{w}) + \lambda \mathbf{w} = \mathbf{0}$ が成り立つ。一方、制約付き問題では、乗数 $\lambda \ge 0$ を $g(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 - \frac{1}{2}r^2$ に対して導入すると、KKT 停留条件 $\nabla \ell(\mathbf{w}) + \lambda \mathbf{w} = \mathbf{0}$ が得られる。重み減衰係数がラグランジュ乗数の役割を果たす形で、まったく同じ方程式が現れるのである。

損失 ℓ が**凸**であるとき、この同値性は双方向に成り立つ。まずペナルティから制約へ向かう方向として、$\mathbf{w}_\lambda$ をペナルティ付き目的関数の最小点とし、ノルムバジェットを $r = \|\mathbf{w}_\lambda\|$ と選ぶ。すると $\mathbf{w}_\lambda$ は実行可能であり、制約は有効になっており（相補性条件を満たす）、KKT 停留条件も自動的に成り立つ。凸問題では KKT 条件が十分条件であるため（35.6.2 節）、$\mathbf{w}_\lambda$ は制約付き問題の解となる。

次に制約からペナルティへ向かう方向として、任意のバジェット $r > 0$ をとる。原点 $\mathbf{w} = \mathbf{0}$ が狭義実行可能点であるため、スレーター条件が成り立つ。制約付き最小値が $\mathbf{w}^\star$ で達成されるなら、KKT 必要条件から乗数 $\lambda^\star(r) \ge 0$ が存在し、$\mathbf{w}^\star$ は凸のペナルティ付き目的関数 $\ell(\mathbf{w}) + \frac{\lambda^\star}{2}\|\mathbf{w}\|^2$ を最小化する。$\lambda^\star > 0$ である限り、二次項によってこの目的関数は狭義凸となるため、$\mathbf{w}^\star$ がその一意な最小点となる。一方、バジェットに余裕があるときは $\lambda^\star = 0$ となり、ペナルティ付き問題は正則化なしの最小化に帰着する。シャドープライスの命題より $\lambda^\star(r) = -\partial p^\star / \partial(\frac{1}{2}r^2)$ が成り立つ。すなわち重み減衰係数とは、二乗ノルムの半分 1 単位あたりの訓練損失の限界削減量にほかならない。

実際、どちらの方向の議論も ℓ_2 ノルムに固有のものではない。$\frac{1}{2}\|\mathbf{w}\|^2$ を任意の凸ペナルティ $\Omega(\mathbf{w})$、たとえば Lasso（35.6.4.1 節）で用いられる ℓ_1 ノルム $\|\mathbf{w}\|_1$ に置き換えてみよう。任意の最小点 $\mathbf{w}_\lambda$（$\ell + \lambda\Omega$ の最小点）は、バジェット $\Omega(\mathbf{w}) \le \Omega(\mathbf{w}_\lambda)$ の制約付き問題を解く。実際、そのバジェット内のすべての $\mathbf{w}$ に対して $\ell(\mathbf{w}) \ge \ell(\mathbf{w}_\lambda) + \lambda\,(\Omega(\mathbf{w}_\lambda) - \Omega(\mathbf{w})) \ge \ell(\mathbf{w}_\lambda)$ が成り立つからである。逆に、凸な ℓ と $\inf \Omega$ を真に上回る任意のバジェットに対してはスレーター条件が成り立ち、鞍点の命題によって制約付き問題の任意の最小点が $\ell + \lambda^\star\Omega$ も最小化することが保証される。

ただし、完全な同値性が成り立つためには ℓ の凸性が不可欠である。非凸損失の場合、ペナルティ付き問題の停留点は依然として対応する制約付き問題の KKT 点になるものの、大域的最適解の集合は食い違うことがあり、**いかなる**ペナルティ重み λ を用いても到達できないノルムバジェットが生じうる（演習問題 6）。

同じ二次項に対する第 3 の解釈にも触れておこう。ノイズ分散（noise variance）σ^2 を持つ二乗損失（squared loss）と、重みに対する等方的なガウス事前分布（Gaussian prior）$\mathcal{N}(\mathbf{0}, \tau^2\mathbf{I})$ を仮定すると、ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$（ただし $\lambda = \sigma^2/\tau^2$）は定数項と全体のスケールを除いて負の対数事前分布と一致する（36.6 節）。このように、単一の数値 λ は、重み減衰係数、ノルムバジェットに対するラグランジュ乗数、およびノイズ分散と事前分布の分散の比という 3 通りの読み方ができる。

この同値性を数値的に検証してみよう。ランダムなデータ上の凸最小二乗損失 $\ell(\mathbf{w}) = \frac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2$ に対し、まず $(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})\mathbf{w} = \mathbf{A}^\top\mathbf{b}$ によってペナルティ付き問題を閉形式で解き、$r = \|\mathbf{w}_\lambda\|$ とおく。次に、ステップサイズ $1/\lambda_{\max}(\mathbf{A}^\top\mathbf{A})$ の射影勾配降下法（projected gradient descent）を用い、クリッピングによって半径 r の球へ射影しながら制約付き問題を解く。制約付きソルバ

は同じ重みを復元できるだろうか。また、停留条件から計算した乗数（$\nabla \ell(\mathbf{w}) + \lambda \mathbf{w} = \mathbf{0}$ と $\mathbf{w}$ の内積をとって得られる $\lambda = -\mathbf{w}^\top \nabla \ell(\mathbf{w}) / r^2$）は λ と一致するだろうか。

```
rng = np.random.default_rng()
A, b = rng.normal(size=(30, 5)), rng.normal(size=30)
grad = lambda w: A.T @ (A @ w - b)                    # gradient of 1/2 |Aw - b|^2
step = 1.0 / np.linalg.eigvalsh(A.T @ A).max()        # 1 / lambda_max(A^T A)
print(' lambda   r = |w_pen|   |w_con - w_pen|   recovered multiplier')
for lam in [0.1, 1.0, 10.0]:
    w_pen = np.linalg.solve(A.T @ A + lam * np.eye(5), A.T @ b)  # penalty form
    r = np.linalg.norm(w_pen)                         # its norm becomes the budget
    w = np.zeros(5)                                   # constraint form: PGD
    for _ in range(5000):
        w -= step * grad(w)
        w *= min(1.0, r / np.linalg.norm(w))          # project onto the r-ball
    lam_rec = -(w @ grad(w)) / r ** 2                 # KKT: grad l + lam w = 0
    print(f'{lam:7.2f}  {r:10.4f}  {np.linalg.norm(w - w_pen):15.2e}  {lam_rec:20.6f}')
```

```
 lambda   r = |w_pen|   |w_con - w_pen|   recovered multiplier
   0.10      0.3606         5.99e-17              0.100000
   1.00      0.3471         3.94e-17              1.000000
  10.00      0.2550         5.72e-17             10.000000
```

どの行においても、制約付き問題の解はペナルティ付き問題の解と機械精度で一致し、停留条件から復元した乗数も表示された小数点以下 6 桁すべてで λ と一致している（具体的なバジェット r の値は乱数の抽出ごとに変わる）。ペナルティと制約は同じ解の経路に対する 2 通りのパラメータ化にすぎず、ラグランジュ乗数が両者の間を変換しているのである。

35.7.4 凸共役

双対関数 q を評価するには、通常、乗数の組を選ぶたびに新たな制約なし最小化問題を解き直す必要がある。しかし制約が線形である場合には、共通する計算をくくり出して、その最小化を最初に 1 度だけ済ませておくことができる。制約 $\mathbf{A}\mathbf{x} \preceq \mathbf{b}$ に対してラグランジアンは $f(\mathbf{x}) + \lambda^\top (\mathbf{A}\mathbf{x} - \mathbf{b})$ であるから、次式が成り立つ。

$$q(\lambda) = \inf_{\mathbf{x}} \left[f(\mathbf{x}) + (\mathbf{A}^\top \lambda)^\top \mathbf{x} \right] - \mathbf{b}^\top \lambda .$$

この下限は f と単一のベクトル $\mathbf{A}^\top \lambda$ のみに依存し、$\mathbf{A}$ や $\mathbf{b}$ に個別には依存しないことに注目しよう。よく使われる各目的関数 f に対して $\inf_{\mathbf{x}} [f(\mathbf{x}) - \mathbf{y}^\top \mathbf{x}]$ の値を $\mathbf{y}$ の関数として 1 度だけ計算しておけば、目的関数 f を持つ**いかなる**線形制約付き問題の双対問題も、対応表からそのまま読み取れるようになる。その対応表こそが凸共役である。

35.7.4.1 支持直線

まず 1 次元で $f(x) = \frac{1}{2} x^2$ を考え、傾き $y = 2$ をとろう。直線 $t = 2x + c$（傾き 2）のうち、至るところで放物線以下にあるものはどれだろうか。$2x + c \le \frac{1}{2} x^2$ がすべての x で成り立つ必要があり、これは $c \le \min_x (\frac{1}{2} x^2 - 2x) = -2$（$x = 2$ で達成）を意味する。そのような直線のうち最も高い $t = 2x - 2$ は点 $(2, 2)$ で放物線に接し、そこでは $f'(2) = 2$ である。傾き 2 の直線をグラフに接するまで下げていくと縦軸切片は -2 に落ち着き、数値 $\max_x (2x - \frac{1}{2} x^2) = 2$ はその切片の符号を反転させたものになる。この様子を Figure 35.27(a) に示す。

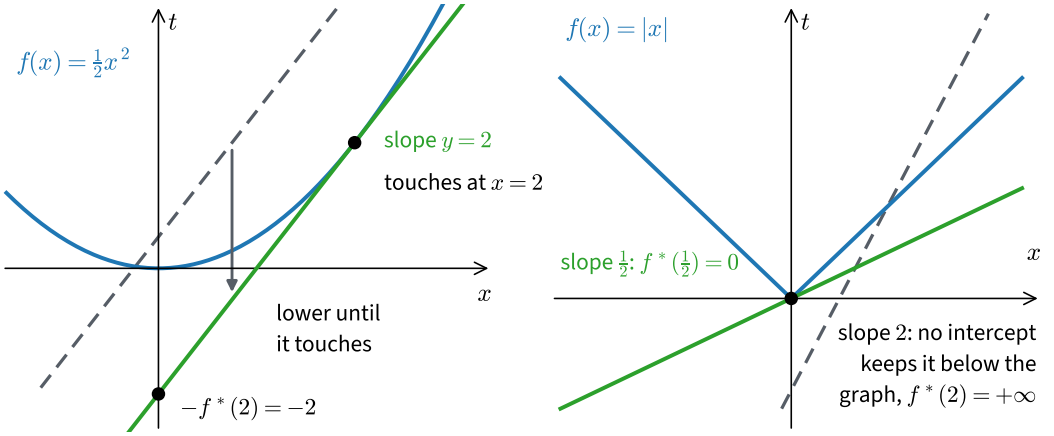


Figure 35.27: 支持直線としての凸共役。(a) $f(x) = \frac{1}{2}x^2$ と傾き 2 の場合、傾き 2 の破線をグラフに接するまで下げると $x = 2$ で接して切片は -2 となり、$f^*(2) = 2$ を与える。(b) $f(x) = |x|$ の場合、傾き $\frac{1}{2}$ の直線は原点でグラフを下から支えて切片 0 となる一方、傾き 2 の直線はどのような切片をとっても $|x|$ より速く上昇するため、$f^*(\frac{1}{2}) = 0$ および $f^*(2) = +\infty$ となる。このように、凸共役が有限になるのは、グラフの下側に支持直線が存在する傾きに限られる。

関数 $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ の**凸共役**（フェンシェル・ルジャンドル変換とも呼ばれる）は、すべての傾きベクトル $\mathbf{y}$ に対してこの符号反転した切片を記録したものである（本書では共役にアスタリスク $*$ を、最適解・最適値に星印 $\star$ を用いる）。

$$f^*(\mathbf{y}) \;=\; \sup_{\mathbf{x}} \; \left(\mathbf{y}^\top \mathbf{x} - f(\mathbf{x})\right). \tag{35.84}$$

幾何学的には、$f^*(\mathbf{y})$ は傾き $\mathbf{y}$ を持ち至るところで f のグラフ以下にあるアフィン関数のうち最も高いものの縦軸切片にマイナスを付けた値である。凸共役は、f を各点での値によってではなく、f の支持直線の族によって記述する。$f(x) = \frac{1}{2}x^2$ と任意の傾き y に対して上限は $x = y$ で達成され、$f^*(y) = \frac{1}{2}y^2$ となる。すなわち、この二次関数は自分自身の共役（自己共役）である。

f が値 $+\infty$ をとることを許すのは、制限された定義域を表現する便利な方法である。$f(\mathbf{x}) = +\infty$ となる点は上限に一切寄与せず、このような拡大実数値関数の凸性は単にそのエピグラフの凸性を意味する。

上限そのものも $+\infty$ になりうる。$f(x) = |x|$ と傾き $y = 2$ を考えよう。$x > 0$ では $2x - |x| = x$ が際限なく大きくなる。つまり、切片をどれほど低くしても、傾き 2 の直線は V 字型のグラフの下側に留まることができない。一方、$|y| \le 1$ のときは常に $yx - |x| \le 0$ が成り立ち、$x = 0$ で等号となるため、原点を通る傾き y の直線がグラフを下から支える。したがって、$f^*(y) = 0$（$|y| \le 1$ のとき）であり、それ以外では $f^*(y) = +\infty$ となる（Figure 35.27(b)）。言い換えれば、絶対値関数の共役は区間 $[-1, 1]$ の**指示関数**（indicator function）、すなわち集合の内部で 0、外部で $+\infty$ をとる関数である。

Equation 35.84 から直ちに 2 つの性質が従う。第 1 に、f^* は f が凸であるかどうかにかかわらず**常に**凸関数である。固定された各 $\mathbf{x}$ に対して式 $\mathbf{y}^\top \mathbf{x} - f(\mathbf{x})$ は $\mathbf{y}$ に関してアフィンであり、アフィン関数族の各点ごとの上限はそのエピグラフがそれらの上側半空間の共通部分となるため凸関数になる（これは q が凹関数になることを示した議論のちょうど鏡像である）。

第 2 に、$f^*(\mathbf{y})$ はすべての $\mathbf{x}$ にわたる上限であるから、$f^*(\mathbf{y}) \geq \mathbf{y}^\top \mathbf{x} - f(\mathbf{x})$ がすべての $\mathbf{x}$ と $\mathbf{y}$ に対して成り立つ。項を整理すると**フェンシェル・ヤングの不等式**が得られる。

$$\mathbf{y}^\top \mathbf{x} \;\leq\; f(\mathbf{x}) + f^*(\mathbf{y}) \qquad \text{for all } \mathbf{x}, \mathbf{y}. \tag{35.85}$$

等号が成り立つのは、$\mathbf{x}$ が Equation 35.84 の上限を達成するとき、かつそのときに限る。f が凸かつ微分可能であれば、これは $\mathbf{y} = \nabla f(\mathbf{x})$ のとき、かつそのときに限って生じる。というのも、このとき $\mathbf{x} \mapsto \mathbf{y}^\top \mathbf{x} - f(\mathbf{x})$ は凹関数となり、勾配がゼロになる点で最大化されるからである。ここでは凸性が重要である。非凸の二重井戸関数（double well）$\phi_0(x) = (x^2 - 1)^2$ では $\phi_0'(0) = 0$ であるにもかかわらず、$\phi_0^*(0) = -\min \phi_0 = 0$ かつ $\phi_0(0) + \phi_0^*(0) = 1 > 0$ となる。これは $x = 0$ がグラフを下から水平な直線で支える点ではなく極大点だからである。自己共役対 $\frac{1}{2}\|\mathbf{x}\|^2$ に対しては、フェンシェル・ヤングの不等式はおなじみの不等式 $\mathbf{y}^\top \mathbf{x} \leq \frac{1}{2}\|\mathbf{x}\|^2 + \frac{1}{2}\|\mathbf{y}\|^2$ に帰着する。

共役へ移ることで何か情報が失われるだろうか。任意の凸関数 f が**固有**（proper）（決して $-\infty$ をとらず、恒等的に $+\infty$ でもない）かつ**閉**（closed）（エピグラフが閉集合である）ならば、支持直線族はグラフを完全に決定し、共役をもう 1 度とると元の f が復元される。すなわち $f^{**} = f$ が成り立つ (Rockafellar 1970)。一方、f が非凸である場合（かつ少なくとも 1 つのアフィン関数で下から抑えられる場合）、f^{**} は下側から抑える最も大きい閉凸関数、すなわち f の凸包絡関数（convex envelope）となる（演習問題 1d）。

本節および演習問題で用いる 4 つの共役対を以下にまとめる。

- $\frac{1}{2}\|\mathbf{x}\|^2$ と $\frac{1}{2}\|\mathbf{y}\|^2$（自己共役対）。
- $\frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ と $\frac{1}{2}\mathbf{y}^\top \mathbf{A}^{-1}\mathbf{y}$（ただし $\mathbf{A} \succ 0$ であり、上限は $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$ で達成される）。
- e^x と、$y \log y - y$（$y > 0$ のとき。さらに値 0 を $y = 0$ で、$+\infty$ を $y < 0$ でとる。上限は $x = \log y$ で達成され、たとえば $f^*(1) = -1$、$f^*(e) = 0$ となる）。
- $\lambda\|\mathbf{x}\|_1$ とボックス $\{\mathbf{y} : \|\mathbf{y}\|_\infty \leq \lambda\}$ の指示関数（上の絶対値関数のベクトル版）。

35.7.4.2 線形制約付き問題の双対

ここで、凸共役を導入する動機となった線形制約付き問題、すなわち $f(\mathbf{x})$ を制約 $\mathbf{A}\mathbf{x} \preceq \mathbf{b}$ のもとで最小化する問題に戻ろう。下限からマイナス符号をくくり出すと、そのまま f^* を定義する上限へと変形できる。

$$\begin{aligned} q(\lambda) &= \inf_{\mathbf{x}} \left[f(\mathbf{x}) + (\mathbf{A}^\top \lambda)^\top \mathbf{x} \right] - \mathbf{b}^\top \lambda \\ &= -\sup_{\mathbf{x}} \left[(-\mathbf{A}^\top \lambda)^\top \mathbf{x} - f(\mathbf{x}) \right] - \mathbf{b}^\top \lambda \\ &= -f^*(-\mathbf{A}^\top \lambda) - \mathbf{b}^\top \lambda. \end{aligned} \tag{35.86}$$

等式制約 $\mathbf{A}\mathbf{x} = \mathbf{b}$ の場合も、まったく同じ 3 行の変形で $q(\nu) = -f^*(-\mathbf{A}^\top \nu) - \mathbf{b}^\top \nu$（ただし ν の符号は制約なし）が得られる。なお、これは q が凹関数である理由に対する第 2 の説明にもなっている。f^* は凸関数でその引数は λ に関して線形であり、マイナス符号が凸性を凹性へと反転させるからである。

Equation 35.86 を考えられる最小の例で試してみよう。半平面内で原点に最も近い点を求める問題、すなわち $\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{x}\|^2$（制約は $\mathbf{a}^\top \mathbf{x} \leq b$、ただし $\mathbf{a} = (1, 2)$、$b = -1$）である。原点は制約に違反するため（$\mathbf{a}^\top \mathbf{0} = 0 > -1$）、最適解において制約は有効になっていなければならない。$f^*(\mathbf{y}) = \frac{1}{2}\|\mathbf{y}\|^2$ と $\|\mathbf{a}\|^2 = 5$ を Equation 35.86 に代入すると、1 ステップで双対関数が得られる。

$$q(\lambda) = -\tfrac{1}{2}\lambda^2\|\mathbf{a}\|^2 - b\lambda = \lambda - \tfrac{5}{2}\lambda^2.$$

$q'(\lambda) = 1 - 5\lambda = 0$ とおくと $\lambda^\star = \tfrac{1}{5}$ および $d^\star = \tfrac{1}{5} - \tfrac{1}{10} = \tfrac{1}{10}$ が得られる。$\lambda^\star$ におけるラグランジアンの最小点は共役の上限を達成する点であり、$\mathbf{x} = -\lambda^\star \mathbf{a} = (-0.2, -0.4)$ となる。直接確かめると $\mathbf{a}^\top \mathbf{x} = -1 = b$ かつ $f(\mathbf{x}) = \tfrac{1}{2}(0.04 + 0.16) = \tfrac{1}{10} = d^\star$ であり、弱双対性によって $\mathbf{x}$ が最適解であることが保証される。これは 35.6.1.3 節の射影公式 $(b/\|\mathbf{a}\|^2)\,\mathbf{a}$ を別の超平面に対して再現したものであり、主問題が 2 変数であったのに対し、双対問題は 1 変数となってわずか 1 行で解けたことになる。

35.7.4.3 log-sum-exp とエントロピー

深層学習のあらゆる場面で、ソフトマックスは生のスコアを指数変換して正規化することで確率へと変換する。なぜ他の正値関数ではなく指数関数なのだろうか。log-sum-exp 関数 $\mathrm{lse}(\mathbf{x}) = \log \sum_i e^{x_i}$ の凸共役がその明快な答えを与えてくれる。確率単体（probability simplex）を $\Delta = \{\mathbf{p} : \mathbf{p} \succeq 0,\ \mathbf{1}^\top \mathbf{p} = 1\}$、シャノンエントロピーを $H(\mathbf{p}) = -\sum_i p_i \log p_i$ と表そう。

$\mathrm{lse}^*(\mathbf{p}) = \sup_{\mathbf{x}} \big(\mathbf{p}^\top \mathbf{x} - \mathrm{lse}(\mathbf{x})\big)$ を計算するため、まず上限がいつ有限になりうるかを調べる。もし負の座標 p_i が 1 つでもあれば、$x_i \to -\infty$ とすることで $\mathbf{p}^\top \mathbf{x}$ は $+\infty$ へ発散する一方、$\mathrm{lse}(\mathbf{x})$ は有界に留まるため、上限は $+\infty$ となる。また $\mathbf{1}^\top \mathbf{p} \neq 1$ であれば、すべての座標を定数だけシフトさせて $\mathbf{x} = c\mathbf{1}$ とおくと $c(\mathbf{1}^\top \mathbf{p} - 1) - \log n$（$\mathrm{lse}(c\mathbf{1}) = c + \log n$ であるため）となり、やはり $c \to \pm\infty$ としたときに発散する。ゆえに $\mathrm{lse}^*(\mathbf{p})$ が有限となるのは確率単体 Δ 上に限られる。

$\mathbf{p}$ が Δ の内部にあるとき（すべての $p_i > 0$）、$\mathbf{x}$ に関する勾配をゼロとおくと $\mathrm{softmax}(\mathbf{x}) = \mathbf{p}$ が得られ、その解は $\mathbf{x} = \log \mathbf{p}$（加法定数の自由度を除いて一意）となって値 $\sum_i p_i \log p_i$ を与える。一方、$\mathbf{p}$ が Δ の境界上にあり一部の $p_i = 0$ となっている場合、ソフトマックスの出力は常に真に正であるため有限な最大点は存在しない。しかし、該当する座標で $x_i \to -\infty$ とすれば同じ値に収束するため、この式は Δ 全体へ連続的に拡張される（ただし標準的な規約 $0 \log 0 = 0$ を用いる）。lse は凸かつ至るところ有限であるため $\mathrm{lse}^{**} = \mathrm{lse}$ が成り立ち、次の共役対が得られる。

$$\begin{aligned} \mathrm{lse}^*(\mathbf{p}) &= \begin{cases} \sum_i p_i \log p_i = -H(\mathbf{p}), & \mathbf{p} \in \Delta, \\ +\infty, & \text{otherwise}, \end{cases} \\ \mathrm{lse}(\mathbf{x}) &= \max_{\mathbf{p} \in \Delta} \big(\mathbf{p}^\top \mathbf{x} + H(\mathbf{p})\big) \quad \text{(dually)}. \end{aligned} \tag{35.87}$$

Equation 35.87 の 2 行目を言葉で読んでみよう。log-sum-exp とは、期待スコア $\mathbf{p}^\top \mathbf{x}$ にエントロピーボーナス $H(\mathbf{p})$ を加えた量を、n 個の結果上のすべての確率分布にわたって最大化したものである。どの分布がこの最大値を達成するだろうか。フェンシェル・ヤングの不等式の等号成立条件より、それは $\mathbf{p} = \nabla \mathrm{lse}(\mathbf{x})$ であり、その第 i 成分は $\partial\,\mathrm{lse}/\partial x_i = e^{x_i} / \sum_j e^{x_j}$、すなわちソフトマックスそのものである（そのヘッセ行列は 35.3 節で計算した）。言い換えれば、同じ期待スコア $\mathbf{p}^\top \mathbf{x}$ を達成するすべての確率分布の中で、ソフトマックスは最大エントロピーを持つ分布であり、36.4.2 節における最大エントロピー法からの導出と一致する（演習問題 2 ではその制約側からソフトマックスを導出する）。エントロピー項がなければ、$\mathbf{p}^\top \mathbf{x}$ を Δ 上で最大化するとすべての確率質量が $\arg\max_i x_i$ に集中するが、エントロピーを加えることでそのハードな選択が滑らかな確率分布へと軟化される。これとまったく同じ共役性の構成を f ダイバージェンスの凸生成関数に適用すると、f-GAN の訓練に用いられる変分下界が得られる（37.3.4 節）。

35.7.5 具体例

ここからは、双対性が実際の問題で何を明らかにするかを浮き彫りにする 3 つの具体例を順に見ていこう。サポートベクターマシンでは、相補性条件によって解がごく少数のデータ点に集中し、さらに双対問題ではデータが内積のみを通じて現れる。注水法では、ラグランジュ乗数が電力 1 単位あたりの通信レートという具体的な工学的単位を持つ。最後に、1 次元の非凸問題を通じて、凸性が防いでいた真に正の双対ギャップを具体的に確認する。

35.7.5.1 サポートベクターマシンの双対問題

線形分離可能な訓練データ $(\mathbf{x}_i, y_i)$（ラベルは $y_i \in \{\pm 1\}$）が与えられたとき、サポートベクターマシン (Boser et al. 1992) は可能な限り最大のマージン（margin）を持つ分離超平面 $\mathbf{w}^\top \mathbf{x} + b = 0$ を求める。その主問題は次式で表される。

$$\min_{\mathbf{w}, b} \ \tfrac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i\,(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \ \ i = 1, \dots, n. \tag{35.88}$$

点 $\mathbf{x}$ から超平面までの距離は $|\mathbf{w}^\top \mathbf{x} + b| / \|\mathbf{w}\|$ であるから（33.1 節）、2 つのマージン超平面 $\mathbf{w}^\top \mathbf{x} + b = \pm 1$ は決定境界（decision boundary）の両側にそれぞれ距離 $1/\|\mathbf{w}\|$ の位置にあり、互いの間隔は $2/\|\mathbf{w}\|$ となる。したがって $\tfrac{1}{2}\|\mathbf{w}\|^2$ を最小化することはマージンを最大化することに相当する。これは訓練点ごとに 1 つのアフィン不等式制約を持つ凸二次計画問題である。制約がアフィンであるため、スレーター条件には実行可能点が 1 つ存在すれば十分であり、データを分離する任意の $(\mathbf{w}, b)$ を定数倍してすべての点が少なくとも 1 のマージンを持つようにスケーリングすればよい。ゆえに強双対性が保証される。

各制約に非負の乗数 $\alpha_i \geq 0$ を 1 つずつ導入し、ラグランジアンを次式で作る。

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \tfrac{1}{2}\|\mathbf{w}\|^2 + \sum_{i} \alpha_i \left(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\right).$$

双対関数 $q(\alpha) = \inf_{\mathbf{w}, b} \mathcal{L}$ を計算するため、まず $\mathcal{L}$ がペナルティの課されていないオフセット b に関してアフィンであり、その係数が $-\sum_i \alpha_i y_i$ であることに注目しよう。この係数がゼロでない限り、$b \to \pm\infty$ とすることで $\mathcal{L}$ は $-\infty$ へ発散するため、$q(\alpha) = -\infty$ が $\sum_i \alpha_i y_i \neq 0$ のときは常に成り立つ。ここに Equation 35.80 の実効定義域が実際に機能している姿が見て取れる。有限性の条件 $\sum_i \alpha_i y_i = 0$ が、双対問題における明示的な等式制約になるのである。この定義域上では b の項が消え、$\mathbf{w}$ に関する狭義凸な二次関数が残るため、その一意な最小値は $\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{0}$ となる点で達成される。

$$\mathbf{w} = \sum_i \alpha_i y_i \, \mathbf{x}_i, \qquad \text{on the domain} \qquad \sum_i \alpha_i y_i = 0.$$

したがって、最適な重みベクトルは訓練入力をそれぞれの乗数で重み付けした線形結合となる。この式を $\mathcal{L}$ に代入すると、**SVM の双対問題**が得られる。

$$\max_{\alpha \geq 0, \ \sum_i \alpha_i y_i = 0} \ \sum_i \alpha_i \ - \ \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j \, y_i y_j \, \mathbf{x}_i^\top \mathbf{x}_j. \tag{35.89}$$

ソルバを選ぶ前から、2 つの際立った帰結が導かれる。第 1 に、相補性条件 $\alpha_i(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)) = 0$ より、$\alpha_i > 0$ となる任意の点は $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$ を満たし、いずれかのマージン超平面上に

ぴったり乗っていなければならない。これらの点を**サポートベクター**と呼ぶ。Figure 35.28 では、丸で囲まれた 4 点の中にしかサポートベクターは存在しえない。それらの点では制約が有効になっており、Figure 35.21 と同様にその法線ベクトルが ∇f と釣り合う一方、$\alpha_i = 0$ である点はすべて $\mathbf{w}$ から完全に脱落する。第 2 に、訓練入力は Equation 35.89 の中に内積 $\mathbf{x}_i^\top \mathbf{x}_j$ のみを通じて現れる。これらの内積をカーネル関数（kernel function）$k(\mathbf{x}_i, \mathbf{x}_j)$ に置き換えれば直ちに非線形分類器が得られるが、これは主問題の定式化では隠れていた構造である。

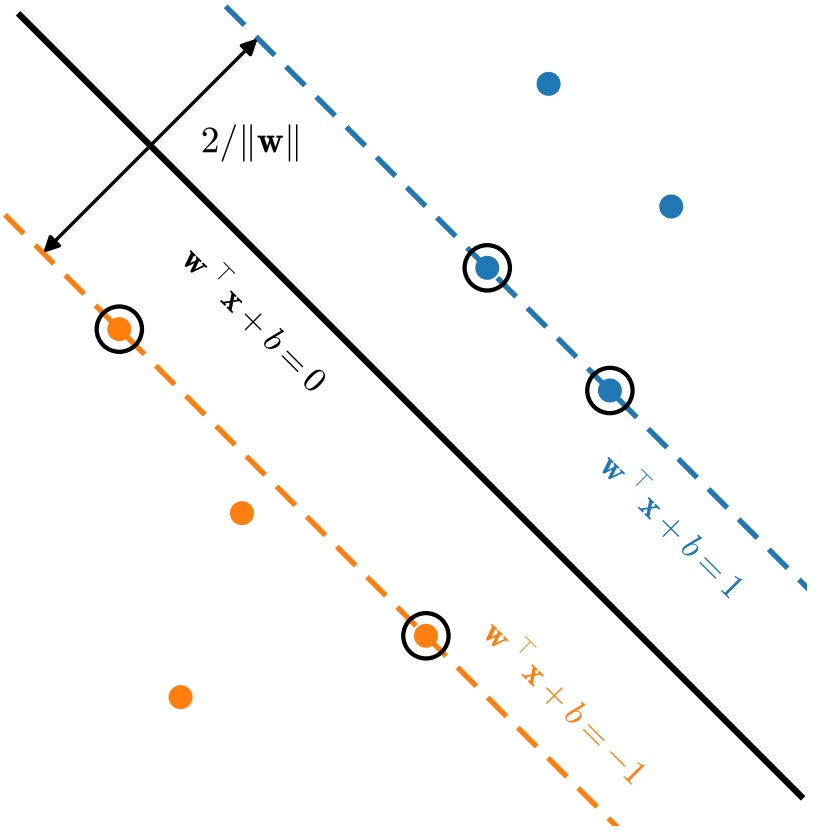


Figure 35.28: 下のコードセルで用いる 8 点の最大マージン分離。実線は決定境界 $\mathbf{w}^{\star\top}\mathbf{x} + b^\star = 0$（$\mathbf{w}^\star = (\frac{4}{7}, \frac{4}{7})$、$b^\star = -\frac{3}{7}$）であり、破線はマージン超平面 $\mathbf{w}^{\star\top}\mathbf{x} + b^\star = \pm 1$（互いの間隔は $2/\|\mathbf{w}^\star\|$）である。マージン超平面上にある丸で囲まれた 4 点のみが $\alpha_i > 0$ を持ちうる。

等式制約 $\sum_i \alpha_i y_i = 0$ が生じたのは、ひとえにペナルティなしのオフセット b によってラグランジアンがある方向にアフィンとなったためである。双対問題の実行可能領域（feasible region）を非負象限（負の成分をゼロにクリップするだけで射影できる単純な集合）に保つため、コードではオフセットを重みに折り込んだ密接に関連する変種を解くことにしよう。すなわち、各入力に定数特徴量を加えて $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$ とし、$\tilde{\mathbf{w}} = (\mathbf{w}, b)$ とおいて、$\frac{1}{2}\|\tilde{\mathbf{w}}\|^2$ を制約 $y_i \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}_i \geq 1$ のもとで最小化する。この変種は b にもペナルティを課すため決定境界は Equation 35.88 とわずかに異なりうるが、引き換えにラグランジアンがすべての主変数に関して狭義凸となり、等式制約が不要になる。$\mathbf{Q}_{ij} = y_i y_j \tilde{\mathbf{x}}_i^\top \tilde{\mathbf{x}}_j$ と定義すると、双対問題は次式となる。

$$\max_{\alpha \succeq 0} \quad \mathbf{1}^\top \alpha - \tfrac{1}{2} \alpha^\top \mathbf{Q}\, \alpha, \tag{35.90}$$

これは非負象限上の凹二次関数の最大化問題である。射影勾配上昇法（35.6.3 節の更新則のステップ符号を反転させたもの）を適用すると、$\alpha \leftarrow \max(\mathbf{0},\, \alpha + \eta\,(\mathbf{1} - \mathbf{Q}\alpha))$ を安全なステップサイズ $\eta = 1/\lambda_{\max}(\mathbf{Q})$ で反復することになる。平面上の 8 点に対してこの上昇法を実行し、復元された超平面がデータを分離するか、正の乗数がすべてマージン 1 の点に乗っているか、そして双対値が主問題の値と一致するかという 3 つの予測を検証しよう。

```
X = np.array([[1.0, 1.5], [2.0, 0.5], [2.5, 2.0], [1.5, 3.0],
              [-1.0, -0.5], [-2.0, 1.0], [-1.5, -2.0], [0.5, -1.5]])
y = np.array([1.0, 1.0, 1.0, 1.0, -1.0, -1.0, -1.0, -1.0])
Xa = np.hstack([X, np.ones((len(X), 1))])     # fold the offset b into w
G = y[:, None] * Xa                           # rows y_i x~_i
Q = G @ G.T                                   # Q_ij = y_i y_j x~_i^T x~_j
alpha = np.zeros(len(X))
eta = 1.0 / np.linalg.eigvalsh(Q).max()       # 1 / lambda_max(Q)
for _ in range(5000):                         # projected gradient ascent
    alpha = np.maximum(0.0, alpha + eta * (1.0 - Q @ alpha))
w = G.T @ alpha                               # w~* = sum_i alpha_i y_i x~_i
margins = y * (Xa @ w)
dual, primal = alpha.sum() - 0.5 * alpha @ Q @ alpha, 0.5 * w @ w
print('alpha* =', alpha.round(4))
print('rank of Q =', np.linalg.matrix_rank(Q), '  multipliers > 0:', int((alpha >
 ↪  1e-8).sum()))
print('w* =', w[:2].round(4), ' b* =', w[2].round(4),
      ' separates data:', bool((np.sign(Xa @ w) == y).all()))
print('margins y_i (w^T x_i + b) =', margins.round(4))
print(f'primal = {primal:.6f}, dual = {dual:.6f}, gap = {abs(primal - dual):.1e}, '
      f'comp. slack = {np.abs(alpha * (margins - 1.0)).max():.1e}')
```

```
alpha* = [0.1309 0.0732 0.     0.     0.     0.2442 0.     0.3885]
rank of Q = 3   multipliers > 0: 4
w* = [0.5714 0.5714]  b* = -0.4286  separates data: True
margins y_i (w^T x_i + b) = [1.     1.     2.1429 2.1429 1.2857 1.     2.4286 1.    ]
primal = 0.418367, dual = 0.418367, gap = 4.4e-16, comp. slack = 3.0e-16
```

3 つの予測はすべて成り立っている。4 つの乗数が正であり、まさにその 4 点がマージン 1 に位置している一方、残りの 4 点はマージンが 1.29 から 2.43 の間にあり $\alpha_i = 0$ となっている。出力された数値は厳密解 $\tilde{\mathbf{w}}^\star = (\tfrac{4}{7}, \tfrac{4}{7}, -\tfrac{3}{7})$（すなわち $\mathbf{w}^\star = (\tfrac{4}{7}, \tfrac{4}{7})$、$b^\star = -\tfrac{3}{7}$）と一致し、$p^\star = d^\star = \tfrac{41}{98} \approx 0.418367$、双対ギャップは 10^{-16} 付近となっている（主問題の値と双対値が等しいことは、最適解における恒等式 $\sum_i \alpha_i^\star = \|\tilde{\mathbf{w}}^\star\|^2$ からも直接従う。演習問題 4a を参照）。

この出力結果は、双対解の解釈に関する繊細な点も浮き彫りにしている。主問題の目的関数 $\tfrac{1}{2}\|\tilde{\mathbf{w}}\|^2$ は狭義凸であるため、主最適解 $\tilde{\mathbf{w}}^\star$ は一意である。しかし双対最適解 $\alpha^\star$ は一意では**ない**。ここでは $\mathbf{Q} = \mathbf{G}\mathbf{G}^\top$ であり、$\mathbf{G}$ は 8×3 行列（各行が $y_i \tilde{\mathbf{x}}_i^\top$）であるため、$\mathbf{Q}$ のランクは高々 3 である。4 つのサポートベクターは $\mathbb{R}^3$ 内の 4 本のベクトルに対応するため線形従属であり、それら 4 つの添字上にのみ非ゼロ成分を持つ非ゼロベクトル $\mathbf{d}$ で $\mathbf{G}^\top \mathbf{d} = \mathbf{0}$ を満たすものが存在する。$\alpha^\star$ を $\mathbf{d}$ の方向に動かしても二次項は変化せず（$\mathbf{Q}\mathbf{d} = \mathbf{0}$ のため）、またすべてのサポートベクターがマージン 1 を持つことから $\mathbf{1}^\top \mathbf{d} = \sum_i d_i\, y_i \tilde{\mathbf{w}}^{\star\top} \tilde{\mathbf{x}}_i = \tilde{\mathbf{w}}^{\star\top} \mathbf{G}^\top \mathbf{d} = 0$ となるため一次項も変化しない。したがって乗数ベクトルからなる線分全体が最適解となり、いずれも同じ $\tilde{\mathbf{w}}^\star$ と同じ双対値を与える。$\mathbf{0}$ から出発する射影勾配上昇法は、単にその線分上の 1 点を返したにすぎない。この線分の各端点では、マージン 1 を持つ 4 点のうち 1 点で実際に $\alpha_i = 0$ となる（演習問題 4c）。相補性条件は一方向の含意であることに注意しよう。すなわち、$\alpha_i > 0$ ならばマージンが 1 であることは保証されるが、マージンが 1 だからといって $\alpha_i > 0$ が強制されるわけではない。

35.7.5.2 注水法

2 つ目の具体例は、閉形式解（closed-form solution）と忘れがたい物理的描像の両方を備えている。送信機が総電力バジェット P を n 個の独立な通信チャネルに配分しなければならないとしよう。チャネル i は背景雑音電力 $N_i > 0$ を持ち、そこに電力 $P_i \geq 0$ を割り当てると 1 回の使用あたり $\log(1 + P_i/N_i)$ nat の伝送レートが得られる (Cover and Thomas 1999)（Cover と Thomas は実ガウスチャネルに対して係数 $\tfrac{1}{2}$ を含めているが、これは以下のラグランジュ乗数を定数倍するだけで最適配分は変えない）。総伝送レートを最大化するため、次の問題を解く。

$$\max_{\mathbf{P}} \; U(\mathbf{P}) = \sum_{i=1}^{n} \log\left(1 + \frac{P_i}{N_i}\right) \quad \text{subject to} \quad P_i \geq 0, \;\; \sum_i P_i = P. \tag{35.91}$$

凹関数 U の最大化は、アフィン制約のもとでの凸関数 $-U$ の最小化と同値である。すべての成分が真に正である実行可能な配分が存在するためスレーター条件が成り立ち、KKT 条件は必要十分条件となる。バジェットの等式制約に用いる乗数 ν（$\sum_i P_i - P = 0$ に対応）と、非負制約に用いる乗数 $\lambda_i \geq 0$（$-P_i \leq 0$ に対応）を導入すると、$-U$ のラグランジアンの停留条件は次式となる。

$$\frac{1}{N_i + P_i} = \nu - \lambda_i .$$

ここから 2 つの場合を読み取ろう。チャネル i に正の電力が割り当てられるとき（$P_i > 0$）、相補性条件により $\lambda_i = 0$ が強制されるため $N_i + P_i = 1/\nu$ となり、電力を受けるすべての稼働チャネルはまったく同じ合計レベル $1/\nu$ に揃う。一方、チャネル i に電力が割り当てられないとき（$P_i = 0$）、双対実行可能性 $\lambda_i = \nu - 1/N_i \geq 0$ から $N_i \geq 1/\nu$ が要求される。この共通のしきい値を $\theta = 1/\nu$ と書けば、最適配分は次の簡潔な形で表される。

$$P_i^{\star} = \max(\theta - N_i,\, 0), \qquad \text{with } \theta \text{ set by } \sum_i \max(\theta - N_i,\, 0) = P.$$

これが古典的な**注水法**の規則である。各チャネルを底の高さが N_i にある器と見立て、器の並び全体へ体積 P の水を注ぎ込む場面を想像してほしい。Figure 35.29 に示すように、水面は一様な水位 θ に落ち着き、最も雑音の小さい静かなチャネルに最も多くの電力が注がれる一方、底の高さが θ を超える器は完全に乾いたまま残る。$\sum_i \max(\theta - N_i, 0)$ は連続な区分線形関数であり、$\theta \leq \min_i N_i$ でゼロ、$\theta > \min_i N_i$ で狭義単調増加であるため、任意のバジェット $P > 0$ に対して一意な水位 θ が定まり、二分法によって容易に求められる（35.6.3 節の単体への射影は、各座標から共通のレベルを引いてゼロでクリップするという、このしきい値構造をちょうど反転させた形になっている）。

シャドープライスの命題から、$\nu^{\star}$ には直ちに工学的な意味が与えられる。U の最大化は $-U$ の最小化であるから、バジェットを P から $P + \upsilon$ へ緩和すると $-U^{\star}$ は変化率 $-\nu^{\star}$ で変化し、したがって $\mathrm{d}U^{\star}/\mathrm{d}P = \nu^{\star} = 1/\theta$ となる。すなわち最適乗数は、追加電力 1 単位あたりに得られる追加の通信レートを表す。5 つのチャネルに対する注水法の配分を計算し、このシャドープライスによる予測を $P \pm 10^{-4}$ における有限差分（finite difference）と比較してみよう。

```
N = np.array([0.1, 0.4, 0.8, 1.6, 2.5])       # channel noise levels
P = 3.0                                       # total power budget

def water_fill(P, N, steps=60):
```

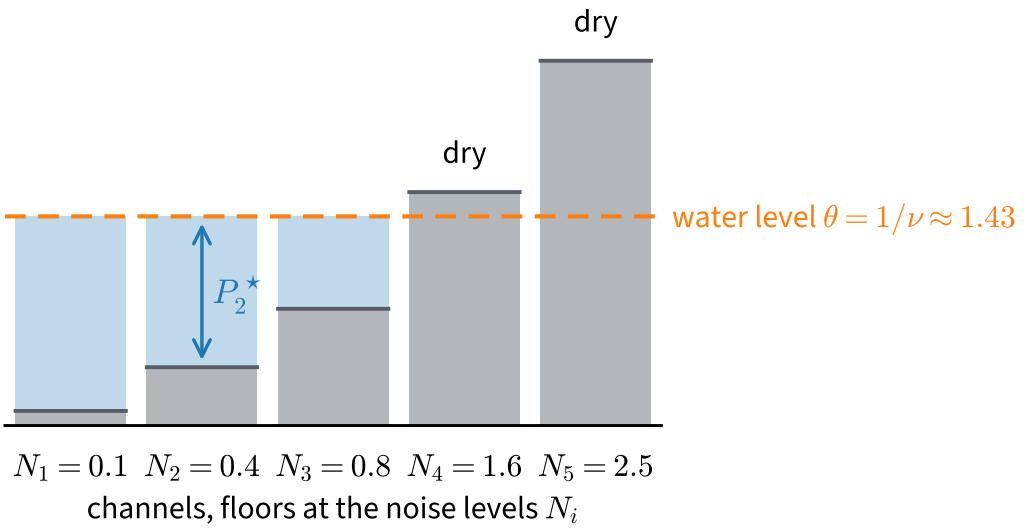


Figure 35.29: 下のコードセルの雑音フロアとバジェットに対する注水法。電力 $P = 3$ を底の高さ N_i の器に注ぐと、雑音が小さい上位 3 つのチャネルが共通の水位 $\theta = 1/\nu^\star \approx 1.43$ まで満たされてそれぞれ $P_i^\star = \theta - N_i$ が割り当てられる一方、底が θ より高い 2 つの器には相補性条件の要請どおり何も割り当てられない。

```
    lo, hi = N.min(), N.min() + P            # bracket for the water level
    for _ in range(steps):                   # bisection on the level theta
        mid = 0.5 * (lo + hi)
        lo, hi = (mid, hi) if np.maximum(0, mid - N).sum() < P else (lo, mid)
    theta = 0.5 * (lo + hi)
    return theta, np.maximum(0.0, theta - N)

theta, power = water_fill(P, N)
print('water level theta =', f'{theta:.6f}', ' P* =', power.round(4),
      ' total =', f'{power.sum():.6f}')
print('wet channels at the common level:', (N + power)[power > 0].round(6))
print('dry channels above the level:    ', bool((N[power == 0] >= theta).all()))
rate = lambda power: np.log(1.0 + power / N).sum()
dP = 1e-4                                    # shadow price, two ways
nu = 1.0 / theta
sens = (rate(water_fill(P + dP, N)[1]) - rate(water_fill(P - dP, N)[1])) / (2 * dP)
print(f'nu = 1/theta = {nu:.6f} vs finite-difference dU*/dP = {sens:.6f}')
```

```
water level theta = 1.433333  P* = [1.3333 1.0333 0.6333 0.     0.     ]  total = 3.000000
wet channels at the common level: [1.433333 1.433333 1.433333]
dry channels above the level:     True
nu = 1/theta = 0.697674 vs finite-difference dU*/dP = 0.697674
```

電力バジェットは機械精度で満たされており、稼働している 3 つのチャネルはすべて共通の水位 $\theta = (3 + 0.1 + 0.4 + 0.8)/3 \approx 1.4333$ に揃い、雑音の大きい 1.6 と 2.5 のチャネルは双対実行可能性の要請どおり乾いたまま留まっている。さらに、乗数 $\nu^\star = 1/\theta = 0.697674$ は有限差分による感度 $\mathrm{d}U^\star/\mathrm{d}P$ と表示された 6 桁すべてで一致している。

35.7.5.3 明示的な双対ギャップ

弱双対性はあらゆる問題に対して成り立つが、強双対性はそうではない。1 次元の非凸の例を見ると、そのギャップが具体的によくわかる。定義域 $\mathcal{D} = [0, 1]$ 上で次の問題を考えよう。

$$\min_{x \in [0,1]} \; f(x) = -x^2 \quad \text{subject to} \quad g(x) = x - \tfrac{1}{2} \leq 0.$$

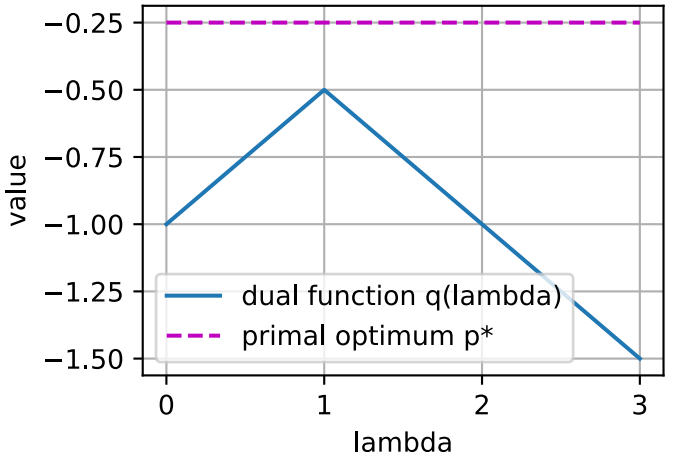
$f(x) = -x^2$ は凸ではなく凹であるため、この問題は非凸問題である。実行可能領域は $[0, \frac{1}{2}]$ であり、最小値は右端点で達成されて $p^\star = f(\frac{1}{2}) = -\frac{1}{4}$ となる。ここでは制約 $g(x) \leq 0$ のみを双対化し、$\mathcal{D} = [0, 1]$ はハードな定義域制約として残すことで、Equation 35.80 の下限を $[0, 1]$ 上でとることにする（弱双対性はそのまま変わらず成り立つ）。任意の $\lambda \geq 0$ に対し、ラグランジアン $\mathcal{L}(x, \lambda) = -x^2 + \lambda(x - \frac{1}{2})$ は x に関して凹関数であるため、$[0, 1]$ 上での最小値は必ず両端点のいずれか、すなわち $\mathcal{L}(0, \lambda) = -\lambda/2$ または $\mathcal{L}(1, \lambda) = \lambda/2 - 1$ で達成される。したがって、次式が得られる。

$$q(\lambda) = \min\left(-\frac{\lambda}{2}, \; \frac{\lambda}{2} - 1\right),$$

保証されているとおり、これは区分線形な凹関数である。2 本の直線は $\lambda^\star = 1$ で交わり、そこでは $d^\star = -\frac{1}{2} < -\frac{1}{4} = p^\star$ となって、大きさ $\frac{1}{4}$ の真に正の双対ギャップが生じる。$q(\lambda)$ を $p^\star$ と並べてプロットし、この非凸問題の双対曲線を本節冒頭のクリッピング問題のプロットと比較してみよう。

```
xs = np.linspace(0.0, 1.0, 4001)              # the domain D = [0, 1]
fx, gx = -xs**2, xs - 0.5                     # objective and constraint on D
p_star = fx[gx <= 0.0].min()                  # primal optimum: -1/4
lams = np.linspace(0.0, 3.0, 601)
q = np.array([(fx + lam * gx).min() for lam in lams])  # dual function on D
d_star, lam_star = q.max(), lams[q.argmax()]
print(f'p* = {p_star:.4f},  d* = {d_star:.4f} at lambda = {lam_star:.2f},'
      f'  gap = {p_star - d_star:.4f}')
d2l.plot(lams, [q, np.full_like(lams, p_star)], 'lambda', 'value',
         legend=['dual function q(lambda)', 'primal optimum p*'])
```

```
p* = -0.2500,  d* = -0.5000 at lambda = 1.00,  gap = 0.2500
```



滑らかに上昇して $p^\star$ に接したクリッピング問題の曲線とは異なり、このテント型の曲線は主最適値より丸々 4 分の 1 だけ低い $-\frac{1}{2}$ でピークを迎える。狭義実行可能点が確実に存在することにも注目しよう。実際、$x = \frac{1}{4}$ は $\mathcal{D}$ の内部にあり $g(\frac{1}{4}) = -\frac{1}{4} < 0$ を満たすため、このギャップはスレーター条件の不成立ではなく、もっぱら凸性の欠如によって引き起こされている。Figure 35.26 のパネル (b) に立ち返ると、像の曲線 $\{(u, -(u + \frac{1}{2})^2) : u \in [-\frac{1}{2}, \frac{1}{2}]\}$ が下向きに湾曲しているため、$\mathcal{A}$ の下側にあるいかなる支持直線も $(0, p^\star)$ に届くことができず、最も高い支持直線（傾き -1）でも $-\frac{1}{2}$ までしか保証できないのである。

未解決の課題が 2 つ残っている。第 1 に、非凸問題では双対問題は最適値の下界を与えるものの、最適値に到達するとは限らない。第 2 に、たとえ鞍点が存在する場合でも、$\mathbf{x}$ と λ を同時に更新すると収束しないことがあり、これがミニマックス訓練における中心的な難しさとなっている。深層ネットワーク全体の訓練は非凸であるが、その構成要素の多く（射影、クリップ付き更新、最終層の当てはめ、凸信頼領域部分問題など）は凸であり、本節の内容はそれらにそのまま適用できる。もっとも、ここまでの本章の導出はすべて厳密な実数演算を仮定してきた。次節では、最適化アルゴリズムそのものではなく有限の浮動小数点精度（floating-point precision）が結果を左右するときに何が起きるかを見ていく。

35.7.6 まとめ

任意の制約付き最適化問題に対し、ラグランジュ双対関数 $q(\lambda, \nu) = \inf_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}, \lambda, \nu)$ は凹関数であり、弱双対性 $q(\lambda, \nu) \le p^\star$ を満たすため、$\lambda \succeq 0$ を選ぶたびに主最適値の下界が保証される。主問題が凸でありスレーター条件（アフィンでないすべての不等式制約に対する狭義実行可能点の存在）を満たすとき、強双対性によってギャップはゼロになり（$d^\star = p^\star$）、達成された主・双対最適解の組はラグランジアンの鞍点をなす。線形制約の場合、双対関数は凸共役を用いて $q(\lambda) = -f^*(-\mathbf{A}^\top \lambda) - \mathbf{b}^\top \lambda$ に帰着し、log-sum-exp と負のエントロピーのような共役対から直ちに双対定式化が得られる。強双対性のもとでは、最適乗数はシャドープライスとして働き（$\lambda_i^\star = -\partial p^\star / \partial u_i$ および $\nu_j^\star = -\partial p^\star / \partial v_j$ が $p^\star$ の微分可能な点で成り立つ）、凸損失において重み減衰がノルム制約と一致する理由、SVM の乗数がサポートベクター以外でゼロになる理由、そして注水法の乗数 $1/\theta$ が電力 1 単位あたりの通信レートを測る理由を説明する。凸性がなければ、$-x^2$ を $[0, 1]$ 上で制約 $x \le \frac{1}{2}$ のもとで最小化する例が示したように、真に正の双対ギャップが残りうる。

35.7.7 演習問題

1. **共役対**。@eq-mdl-opt-conjugate から以下の各凸共役を計算せよ。
   1. $f(\mathbf{x}) = \lambda\|\mathbf{x}\|_1$（ただし $\lambda > 0$）。
   2. $f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$（ただし $\mathbf{A} \succ 0$）。フェンシェル・ヤングの不等式 Equation 35.85 はこの対に対して何を主張するか。
   3. ソフトプラス $f(t) = \log(1+e^t)$（定義域 $\mathbb{R}$）。ロジット t のソフトラベル $p \in [0,1]$ に対する二値交差エントロピーは $\log(1+e^t) - pt$ である。フェンシェル・ヤングの不等式を用いて、これが少なくとも二値エントロピー $H(p) = -p\log p - (1-p)\log(1-p)$ 以上であり、等号が成り立つのは $p = 1/(1+e^{-t})$（t のシグモイド）のとき、かつそのときに限ることを示せ。さらに、両者の差を KL ダイバージェンスとして表せ。
   4. 二重井戸関数 $\phi_0(x) = (x^2-1)^2$（35.3.4 節の関数 ϕ から傾き項 $x/2$ を除いたもの）は凸ではない。その凸包絡関数 ϕ_0^{**} と、$\phi_0^{**} < \phi_0$ となる集合を求めよ。

   Boyd および Vandenberghe の『Convex Optimization』例 3.22、3.26 を改変。

2. **最大エントロピーと温度**。確率ベクトル $\mathbf{p} \in \mathbb{R}^n$ のエントロピーは $H(\mathbf{p}) = -\sum_i p_i \log p_i$ である。本問では制約側からソフトマックスを導出する。
   1. H を制約 $\sum_i p_i = 1$ のもとで 1 つのラグランジュ乗数を用いて最大化せよ（符号制約 $p_i \geq 0$ は無視する）。唯一の停留点が一様分布であることを示せ。符号制約を無視しても答えが変わらないのはなぜか。
   2. すべてが等しいわけではないスコア $s_1, \dots, s_n$ と、目標値 $\bar{s}$（$\min_i s_i$ と $\max_i s_i$ の真に間にある値）を固定し、制約 $\sum_i p_i s_i = \bar{s}$ を追加せよ。最大点が $\mathbf{p} = \mathrm{softmax}(\beta \mathbf{s})$（スカラー β はスコア制約の乗数から定まる）となることを示せ。また、$\mathrm{softmax}(\beta\mathbf{s})$ のもとでの平均スコアが β に関して狭義単調増加であり、目標値を満たす β がただ 1 つ存在することを示せ。
   3. $H^\star(\bar{s})$ を最大エントロピーとする。$\mathrm{d}H^\star/\mathrm{d}\bar{s} = -\beta$ であり、35.6.1.3 節の乗数がオフセット b を価格付けするのと同様に、この乗数がスコア制約を価格付けすることを示せ。どのような目標値 $\bar{s}$ に対して β は負になるか。また、ソフトマックス温度 $1/\beta$ は $\bar{s}$ が $\max_i s_i$ に近づくときどうなるか。

3. **ℓ_1 最小化に対する双対保証**。基底追跡法

$$\min_{\mathbf{x}} \ \|\mathbf{x}\|_1 \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{35.92}$$

   （ただし $\mathbf{A} \in \mathbb{R}^{m\times n}$、$m < n$）は、劣決定な連立一次方程式からスパースな解を選び出す。制約はアフィンであるため、スレーター条件には実行可能点が 1 つあれば十分であり、方程式が解を持つ限り $d^\star = p^\star$ が成り立つ。また $\|\cdot\|_1$ の凸共役はボックス $\{\mathbf{y} : \|\mathbf{y}\|_\infty \leq 1\}$ の指示関数である。
   1. Equation 35.86 の等式制約版を用い、ν を $-\nu$ に置き換えると、双対問題が $\max_\nu \mathbf{b}^\top \nu$（制約 $\|\mathbf{A}^\top \nu\|_\infty \leq 1$）になることを示せ。
   2. $\mathbf{A} = \begin{pmatrix} 1 & 2 \end{pmatrix}$ および $b = 2$ に対して両方の問題を手計算で解き、$d^\star = p^\star$ を確認せよ。
   3. $\mathbf{x}^\star$ と $\nu^\star$ を最適解とする。$x_i^\star = 0$ が $|(\mathbf{A}^\top \nu^\star)_i| < 1$ となるすべての成分で成り立ち、$(\mathbf{A}^\top \nu^\star)_i = \mathrm{sign}(x_i^\star)$ が $x_i^\star \neq 0$ となるすべての成分で成り立つことを示せ。候補解 $\mathbf{x}$（$\mathbf{A}\mathbf{x} = \mathbf{b}$ を満たす）が与えられたとする。問題を解くことなくベクトル ν によって $\mathbf{x}$ が最適解であることを保証するには、どのような条件を満たせばよいか。

4. **サポートベクターと双対値**。オフセットを折り込んだ SVM 双対問題 Equation 35.90 において、$\alpha^\star$ を最適解とし、$\tilde{\mathbf{w}}^\star = \sum_i \alpha_i^\star y_i \tilde{\mathbf{x}}_i$ とおく。

1. 非負象限上の双対問題の KKT 条件を書き下すことで、$\sum_i \alpha_i^\star = \|\tilde{\mathbf{w}}^\star\|^2$ を示せ。さらに、双対最適値が主問題の値 $\frac{1}{2}\|\tilde{\mathbf{w}}^\star\|^2$ に等しいことを導け。
2. $\alpha_j^\star = 0$ である訓練点を削除しても、残りの乗数は縮小された問題に対して最適なままであり、$\tilde{\mathbf{w}}^\star$ も変化しないことを示せ。1 つ抜き検証では、n 個の各点を残りの $n-1$ 点で訓練した SVM によって分類する。1 つ抜き誤り率は高々 $\alpha_i^\star > 0$ となる点の割合以下であることが、すべての最適解 $\alpha^\star$ に対して成り立つことを導け。どの最適解 $\alpha^\star$ が最良の上界を与えるか。
3. [code] SVM のコードセルの行列 $\mathbf{Q}$ のランクは 3 である。S を $\alpha_i^\star > 0$ となる 4 つの添字の集合とし、$\mathbf{G}_S$ を `G` の対応する行からなる部分行列とする。非ゼロの $\mathbf{d} \in \mathbb{R}^4$ で $\mathbf{G}_S^\top \mathbf{d} = \mathbf{0}$ を満たすものを求め（`np.linalg.svd` を $\mathbf{G}_S^\top$ に適用せよ）、$\alpha_S^\star + t\mathbf{d}$ をいずれかの成分がゼロに達するまで両方向へ動かせ。両方の端点が双対実行可能であり、双対値 $\frac{41}{98}$ を持ち、同じ $\tilde{\mathbf{w}}^\star$ を与えることを検証せよ。各端点において、マージン 1 のどの点が $\alpha_i = 0$ になるか。このことは、乗数から「サポートベクター」を読み取ることについて何を意味するか（端点は $(0, 0.204, 0, 0, 0, 0.192, 0, 0.441)$ および $(0.204, 0, 0, 0, 0, 0.273, 0, 0.359)$ となるはずである）。

5. [code] **勾配降下上昇法。**●主双対法や GAN の訓練では、最小化変数を勾配降下法で、最大化変数を勾配上昇法で同時に更新する。

   1. 原点を鞍点とする双線形ゲーム $\min_x \max_y \, xy$ に対し、同時降下上昇法 $x \leftarrow x - \eta y$ および $y \leftarrow y + \eta x$ を線形変換として表せ。任意の $\eta > 0$ に対し、ゼロでないあらゆる初期点から鞍点を離れて渦巻状に発散することを示せ。また、y の更新に新しい x を用いる交互更新版が、行列式 1 の線形変換であることを示せ。$\eta < 2$ のとき、その反復点はどのように振る舞うか。
   2. 35.6.1.3 節の最近接点問題のラグランジアン $\mathcal{L}(\mathbf{x}, \nu) = \frac{1}{2}\mathbf{x}^\top\mathbf{x} + \nu(\mathbf{a}^\top\mathbf{x} - b)$ に対し、ここでは $\mathbf{a} = (1, 2)$、$b = 1$、初期値 $\mathbf{x} = \mathbf{0}$、$\nu = 0$ として同時降下上昇法を実行せよ。反復点が $(\mathbf{x}^\star, \nu^\star)$ に収束するステップサイズの範囲を求めよ。さらにその範囲を、組 $(\mathbf{a}^\top\mathbf{x}/\|\mathbf{a}\|, \nu)$ と、$\mathbf{x}$ のうち $\mathbf{a}$ に直交する成分のそれぞれに作用する更新の固有値から導出せよ。
   3. 双線形ゲームの目的関数に $\frac{\epsilon}{2}x^2$（ただし $0 < \epsilon \leq 2$）を加えよ。このとき同時降下上昇法はどのステップサイズに対して収束するか。また、前問のラグランジアンのどの項が ϵ の役割を果たしているか。

6. [code] **凸性がない場合のペナルティと制約の比較。**●凸損失に対しては、35.7.3.2 節で重み減衰とノルム制約が同じ解を描くことを示した。代わりに非凸の 1 次元損失 $\ell(w) = (w-1)^2(w-3)^2 - w$（極小点は $w = 1.16$ と $w = 3.11$ の付近にある）を考えよ。

   1. w の細かいグリッド上で、対数グリッド上の各ペナルティ付き最小点 $w_\lambda = \operatorname{argmin}_w \ell(w) + \frac{\lambda}{2}w^2$（各 λ に対するもの）と、制約付き最適値 $p^\star(r) = \min\{\ell(w) : w^2 \leq r^2\}$（$r \in [0, 3.2]$）を計算せよ。点 $(w_\lambda^2, \ell(w_\lambda))$ を曲線 $(r^2, p^\star(r))$ の上に重ねてプロットせよ。バジェット r のうち、いかなるペナルティ付き最小点のノルム $|w_\lambda|$ にも決して一致しないものはどの範囲か。
   2. w_λ が $\ell(w) + \frac{\lambda}{2}w^2$ を最小化することは、傾き $-\frac{\lambda}{2}$ で $(w_\lambda^2, \ell(w_\lambda))$ を通る直線が集合 $\{(w^2, \ell(w)) : w \in \mathbb{R}\}$ のすべての点の下側にあることと同値であることを示せ。この事実と Figure 35.26 の支持直線の描像を用いて、欠落するバジェットが生じる理由を説明せよ。
   3. 欠落する範囲から制約が有効となるバジェット r を 1 つ選べ。本文と同様に制約を $\frac{1}{2}w^2 \leq \frac{1}{2}r^2$ と書いたときの制約付き解の KKT 乗数 λ を計算し、この λ を用いたペナルティ付き目的関数がより小さい値をとる点を求めよ。さらに、この制約付き問題の双対ギャップ $p^\star(r) - d^\star(r)$ を計算せよ。

7. [extended] **エントロピー正則化付き最適輸送。●●**確率ベクトル $\mathbf{p} \in \mathbb{R}^n$ と $\mathbf{r} \in \mathbb{R}^m$、およびコスト行列 $\mathbf{C} \in \mathbb{R}^{n \times m}$ が与えられたとき、最適輸送は最も低コストな輸送計画 Γ で質量 $\mathbf{p}$ を質量 $\mathbf{r}$ へ移すものを求める。

$$\mathrm{OT} = \min_{\Gamma \geq 0} \sum_{i,j} C_{ij}\Gamma_{ij} \quad \text{subject to} \quad \Gamma\mathbf{1} = \mathbf{p}, \;\; \Gamma^\top \mathbf{1} = \mathbf{r}, \tag{35.93}$$

これは nm 変数の線形計画問題である。エントロピー正則化 (Cuturi 2013) では、目的関数に $\varepsilon \sum_{i,j} \Gamma_{ij}(\log \Gamma_{ij} - 1)$ (ただし $\varepsilon > 0$) を加える。正則化後の最適解を Γ_ε とする。シンクホーン反復によるその解法は、分布間の比較に広く用いられる (Peyré and Cuturi 2019)。

   1. 周辺分布の制約に対する項 $\phi^\top(\mathbf{p} - \Gamma\mathbf{1}) + \psi^\top(\mathbf{r} - \Gamma^\top\mathbf{1})$ を用いてラグランジアンを作れ。$\Gamma_\varepsilon = \mathrm{diag}(\mathbf{u})\,\mathbf{K}\,\mathrm{diag}(\mathbf{v})$ (ただし $K_{ij} = e^{-C_{ij}/\varepsilon}$、$u_i = e^{\phi_i/\varepsilon}$、$v_j = e^{\psi_j/\varepsilon}$) となることを示せ。符号制約 $\Gamma_{ij} \geq 0$ をラグランジアンから省いてよいのはなぜか。
   2. 双対関数が

$$q(\phi, \psi) = \mathbf{p}^\top\phi + \mathbf{r}^\top\psi - \varepsilon \sum_{i,j} e^{(\phi_i + \psi_j - C_{ij})/\varepsilon}, \tag{35.94}$$

      となること、およびこれを ϕ について（ψ を固定して）厳密に最大化すると要素ごとのシンクホーン更新 $\mathbf{u} = \mathbf{p}/(\mathbf{K}\mathbf{v})$ が得られることを示せ。これより、シンクホーンのアルゴリズムは凹関数上のブロック座標上昇法であり、双対値が決して減少しないことを導け。すべての双対値が OT の下界にもなるのはなぜか。
   3. $H(\Gamma) = -\sum_{i,j}\Gamma_{ij}\log\Gamma_{ij}$ とおき、すべての実行可能な輸送計画が $\max(H(\mathbf{p}), H(\mathbf{r})) \leq H(\Gamma) \leq H(\mathbf{p}) + H(\mathbf{r})$ を満たすことを認めるものとする。このとき $0 \leq \sum_{i,j} C_{ij}(\Gamma_\varepsilon)_{ij} - \mathrm{OT} \leq \varepsilon \min(H(\mathbf{p}), H(\mathbf{r}))$ を示せ。
   4. コストが $[0, 1]$ の範囲にあるとする。float64 では、e^{-x} は x が約 745 を超えるとゼロに丸められる。どのような ε に対して $\mathbf{K}$ の成分がゼロになりうるか、またそのとき乗法的な更新 $\mathbf{u} = \mathbf{p}/(\mathbf{K}\mathbf{v})$ に何が起きるか。ϕ と ψ を直接更新し、$\mathbf{K}$ との積の代わりに log-sum-exp を用いればこの問題を回避できる理由を説明せよ。

8. [code] **リッジ回帰の双対問題。** $\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$（ただし $\mathbf{A} \in \mathbb{R}^{n \times d}$）を、$\min_{\mathbf{w},\mathbf{z}} \frac{1}{2}\|\mathbf{z}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$（制約 $\mathbf{A}\mathbf{w} - \mathbf{b} - \mathbf{z} = \mathbf{0}$、乗数 $\alpha \in \mathbb{R}^n$）と書き換えよ。

   1. ラグランジアンを $\mathbf{z}$ と $\mathbf{w}$ について最小化し、$\mathbf{z} = \alpha$、$\mathbf{w} = -\mathbf{A}^\top\alpha/\lambda$、および双対関数 $q(\alpha) = -\frac{1}{2}\|\alpha\|^2 - \frac{1}{2\lambda}\alpha^\top\mathbf{A}\mathbf{A}^\top\alpha - \mathbf{b}^\top\alpha$ を導け。なぜ強双対性が成り立つのか。
   2. q を最大化して $\alpha^\star = -\lambda(\mathbf{A}\mathbf{A}^\top + \lambda\mathbf{I})^{-1}\mathbf{b}$ および $\mathbf{w}^\star = \mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top + \lambda\mathbf{I})^{-1}\mathbf{b}$ を求めよ。SVM の重みベクトルがデータ点の線形結合になるのと同様に、なぜ解が $\mathbf{A}$ の行ベクトルの線形結合になるのか、またなぜデータが $\mathbf{A}\mathbf{A}^\top$ のみを通じて現れるのかを説明せよ。
   3. データ例数 $n = 20$、特徴量数 $d = 50$ に対し、主問題の解法 $(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})\mathbf{w} = \mathbf{A}^\top\mathbf{b}$ と双対公式が約 10^{-14} の精度で一致すること、および主問題の値と双対値が一致することを数値的に検証せよ。$n \ll d$ のとき計算コストが低いのはどちらの解法か。また、$\mathbf{A}\mathbf{A}^\top$ をカーネル行列に置き換えると何が得られるか。

35.8 数値安定性と条件性

本章のこれまでの計算はすべて浮動小数点（floating point）で実行し、その結果をあたかも厳密な実数であるかのように読み取ってきた。大抵の場合、この便利な仮定はそのまま通用する。しかし時折、訓練損失が NaN になったり、線形ソルバが一見まったく妥当に見えながら大きく誤った解を返したりすることがある。本節では、最適化アルゴリズムではなく有限精度演算が計算結果を左右するのはどのような場合か、そしてそれにどう対処できるかを考える。

2 つの小さな例から、数値的な問題が生じる 2 通りの経路がわかる。第 1 に、3 つのロジット $(1000, 1001, 1002)$ は確率 $(0.09, 0.24, 0.67)$ を持つごく普通のソフトマックスを定める。ところが教科書通りの式をそのまま評価すると、倍精度（fp64）でさえ 3 つの NaN が返る。第 2 に、`np.linalg.solve` をある 12×12 の連立一次方程式に適用すると、与えられた方程式と 10 進法で 17 桁目しか違わない方程式の厳密解であるにもかかわらず、真の解から 30% もずれた答えが返る。前者の失敗は NaN の損失を出して訓練を停止させるが、後者は気づかれないまま通り過ぎてしまう。

これら 2 つの失敗の原因は根本的に異なる。素朴なソフトマックスの式は、いかなる標準的な浮動小数点形式（floating-point format）でも保持できないほど巨大な数 e^{1000} を経由するが、e^{1000} を作らない代数的に同値な書き換えを行えば、最終桁の数単位まで正確な値が得られる。対照的に、この 12×12 の連立一次方程式はきわめて敏感であり、データの 16 桁目に生じた丸め誤差（rounding error）が解の最初の桁を動かしてしまうため、ソルバをどのように書き換えてもその感度を取り除くことはできない。

これら 2 つの状況を見分けるため、まずコンピュータが実数をどのように格納し、1 回の丸めがどれだけの誤差を生むかを調べる。次に、ソフトマックスと交差エントロピー（cross-entropy）を形式が保持できない数を決して経由しないように書き換え、さらに分散についてもほぼ等しい大きな数同士を決して減算しないように書き換える。続いて、感度の尺度である関数の条件数（condition number）を導入し、不適切なアルゴリズムと本質的に難しい問題を切り分ける。最後に、連立一次方程式の条件数が、最小二乗法による求解で保持できる桁数と、35.1 節で見た勾配降下法に必要なステップ数の双方をいかに支配するかを確認する。5.5 節では現代の訓練が実際に低精度形式をどう活用するかを扱ったが、本節ではそれらの形式での計算がなぜ成功し、なぜ失敗するのかを導出する。

個々の公式以上に重要な考え方が 3 つある。浮動小数点演算の相対丸めモデル、不安定なアルゴリズムとは良条件（well-conditioned）な問題を悪条件（ill-conditioned）な中間ステップへ迂回させるものだという原理、そしてソルバの解の誤差を問題の感度に結びつける上界である。総和の順序とリッジ正則化に関する項は、初読時には飛ばしてもよい。

```
import ml_dtypes
from fractions import Fraction
```

35.8.1 浮動小数点演算

35.8.1.1 表現と間隔

コンピュータに 1.1 を格納させると、実際には何が格納されるのだろうか。この仕組みは、2 進小数点以下を 2 ビットだけ保持するトイ 2 進形式で考えると最もわかりやすい。1 と 2 の間では、この形式は $(1.00)_2 = 1$、$(1.01)_2 = 1.25$、$(1.10)_2 = 1.5$、$(1.11)_2 = 1.75$ の 4 つの値しか保持できない。これら 4 つの値に 2 のべき乗を掛けると他の区間も埋まり、$2, 2.5, 3, 3.5$ が 2 と 4 の間に、$0.5, 0.625, 0.75, 0.875$ が 0.5 と 1 の間に並ぶ。隣り合う 2 のべき乗に挟まれた区間 $[2^e, 2^{e+1})$ である各**オクターブ**にはちょうど 4 つの値が入るため、大きさが 2 倍になるたびに隣り合う数の絶対間隔も 2 倍になる。

グリッド点の間に落ちる実数は、最も近いグリッド点へ丸められる。たとえば $1.1 \mapsto 1$（相対誤差 -0.091）、$1.3 \mapsto 1.25$（誤差 -0.038）、$3.3 \mapsto 3.5$（誤差 $+0.061$）となる。ちょうど中間に位置する点は同距離（タイ）となり、標準的な演算では最下位ビットが 0 である側の隣接値へ丸める。したがって中点 1.125 は 1 へ丸められ、誤差は -0.111 となる。どの相対誤差も $1/8$ を超えないことに注目しよう。丸めによって数が動く量は高々間隔の半分であり、どのオクターブでも間隔の半分はそのオクターブの最小値の $1/8$ だからである。Figure 35.30 に実数直線上のこのグリッドを示す。

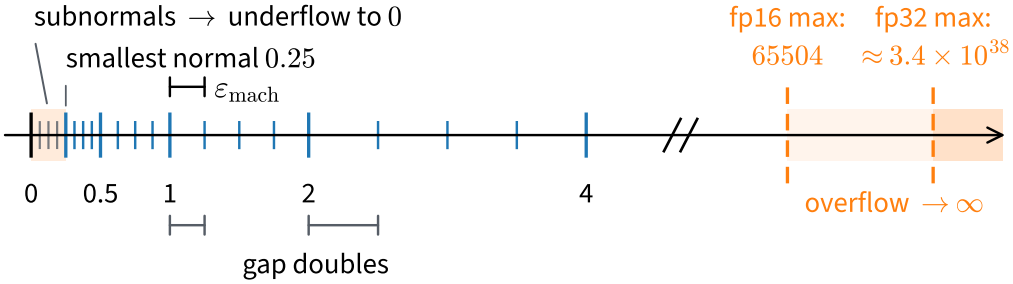


Figure 35.30: 仮数部 2 ビットのトイ形式（1 オクターブあたり 4 値、$\varepsilon_{\text{mach}} = 1/4$、最小の正規化数 0.25）における実数直線上の浮動小数点数。隣り合う値との絶対間隔は 2 のべき乗ごとに 2 倍になる一方、相対間隔は $\varepsilon_{\text{mach}}$ 付近に保たれる。最小の正規化数より下では非正規化数が同じ間隔のままゼロまで続き、軸の省略記号の先では実際の形式がそれぞれのオーバーフロー閾値（fp16 では 65504、fp32 では約 3.4×10^{38}）で終わる。

実際の浮動小数点形式もこれとまったく同じ仕組みであり、単にビット数が多いだけである。浮動小数点数とは、2 進小数点以下に固定長 p 桁の 2 進数字を持つ基数 2 の科学記数法であり、次の式で表される。

$$x = (-1)^{s} \cdot (1.b_1 b_2 \ldots b_p)_2 \cdot 2^{e}, \tag{35.95}$$

ここで、s は符号ビット、$b_1, \ldots, b_p$ は**仮数部**（mantissa または significand）のビット、e は固定された範囲から選ばれる整数の指数部である。指数部はその形式が表現できる大きさの範囲を定め、仮数部は相対精度を定める。オクターブ $[2^{e}, 2^{e+1})$ の内部では、隣り合う値は 2^{e-p} ずつ離れている。1 からその次に表現可能な数までの間隔 $\varepsilon_{\text{mach}} = 2^{-p}$ は**計算機イプシロン**（machine epsilon）と呼ばれ、先ほどのトイ形式では $p = 2$、$\varepsilon_{\text{mach}} = 1/4$ である。

実数 x を最も近い表現可能な値へ丸めると、その移動量は高々間隔の半分である 2^{e-p-1} に収まる。オクターブ全体で $|x| \geq 2^{e}$ であるから、相対誤差は高々 2^{-p-1} である。許容範囲内の指数部を持ち Equation 35.95 の形で表される値を**正規化数**（normal number）と呼ぶことにすると、正規化数へ丸められるすべての実数 x は次を満たす。

$$\mathrm{fl}(x) = x\,(1+\delta), \qquad |\delta| \leq u = \tfrac{1}{2}\,\varepsilon_{\text{mach}} = 2^{-p-1}. \tag{35.96}$$

言い換えれば、1 回の丸めが数に与える摂動は高々 2^{p+1} 分の 1 であり、トイ形式では 8 分の 1、fp32（$p = 23$）では約 1700 万分の 1、bf16（$p = 7$）では 256 分の 1 である。この上界 u を**丸めの単位**（unit roundoff）と呼ぶ。標準の IEEE 演算はすべての基本演算の結果を最も近い表現可能な値へ丸めるため、計算された和 $x \oplus y$ は $(x+y)(1+\delta)$（ただし $|\delta| \leq u$）に等しくなり、同じ保証が $-$、$\times$、$/$ に対しても成り立つ (IEEE 2019; Goldberg 1991)。

この明快な相対モデルには 3 つの例外がある。形式が保持できる上限を超える結果は ∞ へと**オーバーフロー**する。最小の正規化数より小さい結果は**アンダーフロー**し、ゼロまでの残りの隙間を埋める**非正規化数**（subnormal number）を段階的に辿って減少する。$0/0$ や ∞/∞ のような未定義の演算は NaN を生じる。これら 3 つはいずれも相対的ではなく絶対的なスケールの現象であり、標準的な形式を一覧した直後に改めて取り上げる。

35.8.1.2 形式

深層学習では、倍精度（fp64）より幅の狭い 5 つの形式が用いられる。各形式の計算機イプシロン $\varepsilon_{\text{mach}}$、最小の正規化数、最大の有限値をライブラリから直接確認してみよう。

```
print(f'{"dtype":>14} {"eps":>10} {"smallest normal":>16} {"max":>10}')
for dt in [np.float16, np.float32, ml_dtypes.bfloat16,
           ml_dtypes.float8_e4m3fn, ml_dtypes.float8_e5m2]:
    fi = ml_dtypes.finfo(dt)                  # values have dtype dt
    print(f'{np.dtype(dt).name:>14} {float(fi.eps):10.3e} '
          f'{float(fi.smallest_normal):16.3e} {float(fi.max):10.3e}')
```

```
         dtype        eps  smallest normal        max
       float16  9.766e-04        6.104e-05  6.550e+04
       float32  1.192e-07        1.175e-38  3.403e+38
      bfloat16  7.812e-03        1.175e-38  3.390e+38
  float8_e4m3fn  1.250e-01        1.562e-02  4.480e+02
    float8_e5m2  2.500e-01        6.104e-05  5.734e+04
```

fp32 は仮数部に $p = 23$ ビット（10 進で約 7 桁）を保持し、最大 3.4×10^{38} まで達する。**fp16** は $p = 10$ ビット（10 進で約 3 桁）を保持するが、指数部の範囲が狭く、65504 を超えるとオーバーフローし、$2^{-14} \approx 6.1 \times 10^{-5}$ 未満では非正規化数になる。**bf16**（ライブラリでは `bfloat16`）はこれとは逆のトレードオフをとる。fp32 の 8 個の指数部ビットをすべて維持してその広大なダイナミックレンジを保つ代わりに、仮数部をわずか $p = 7$ ビット（$\varepsilon_{\text{mach}} = 2^{-7} \approx 0.0078$、10 進で約 2〜3 桁）に抑えている。なお、bf16 についてよく引用される 2^{-8} という値は、丸めの単位 $u = \varepsilon_{\text{mach}}/2$ である。2 つの fp8 形式はこのトレードオフをさらに推し進め、E4M3 は仮数部 3 ビット（10 進で約 1 桁）を保持して上限が 448 となり、E5M2 は fp16 と同じ指数部範囲で仮数部 2 ビットを保持する。これほどダイナミックレンジが狭いため、fp8 が実用になるのはテンソルごと、あるいは値のブロックごとに独立したスケール係数を組み合わせた場合に限られる。

bf16 は単に fp32 の上位 16 ビット（同じ符号ビット、同じ 8 個の指数部ビット、および先頭 7 個の仮数部ビット）であるため、fp32 から bf16 への変換は下位 16 ビットを丸め落とすことに帰着する。この丸めは、生のビットパターンに対する整数演算で直接実装できる。切り捨てられる 16 ビットの中点は `0x8000` であるから、`0x7FFF`（中点より 1 小さい値）を加え、さらに残す最下位ビットが 1 であればもう 1 を加えてから、下位 16 ビットをゼロにクリアする。この加算で残すビット側へ桁上がりが生じるのは、切り捨て部分が中点より真に大きいか、あるいはちょうど中点にあって残す最下位ビットが奇数であるとき、かつそのときに限る。これがわずか 1 行のビット操作による最近傍偶数丸め（round-to-nearest, ties-to-even）であり、演習問題 5 ではこの上に確率的丸めを構築する。中点より上の入力 1 つと、厳密な中点（タイ）2 つで動作を確認しよう。

```
def to_bf16(x):
    """Round float32 to the nearest bfloat16, ties to even."""
    bits = np.atleast_1d(np.asarray(x, np.float32)).view(np.uint32)
    bits = (bits + 0x7FFF + ((bits >> 16) & 1)) & 0xFFFF0000
    return bits.astype(np.uint32).view(np.float32)
```

```
    for x, expected in [(1 + 2**-8 + 2**-10, 1 + 2**-7),   # above the midpoint
                        (1 + 2**-8, 1.0),                  # tie, 1 is even
                        (1 + 3 * 2**-8, 1 + 2**-6)]:       # tie, 1 + 2^-6 is even
        print(f'{x:.8f} -> {to_bf16(x).item():.8f}   (expected {expected:.8f})')
```

```
1.00488281 -> 1.00781250   (expected 1.00781250)
1.00390625 -> 1.00000000   (expected 1.00000000)
1.01171875 -> 1.01562500   (expected 1.01562500)
```

予想通り、最初の入力は 1 と $1 + 2^{-7}$ の中点より上にあるため切り上げられる。2 番目の入力はちょうどその中点上にあり、偶数の隣接値である 1 へ切り捨てられるのに対し、3 番目の入力は $1 + 2^{-7}$ と $1 + 2^{-6}$ の中点上にあり、偶数の隣接値である $1 + 2^{-6}$ へ切り上げられる。次に、fp32 でも同じタイ規則を確認し、大きさに応じて間隔がどう広がるかを観察するとともに、各形式でオーバーフローせずに指数関数を計算できる最大のロジットを求めてみよう。

```
    eps = np.finfo(np.float32).eps
    one = np.float32(1.0)
    print('1 + eps   != 1 :', one + eps != one)
    print('1 + eps/2 == 1 :', one + np.float32(eps / 2) == one)
    print('gap between float32 neighbors at 1    :', np.spacing(one))
    print('gap between float32 neighbors at 2^20 :', np.spacing(np.float32(2.0**20)))
    for dt in [np.float16, np.float32, np.float64]:
        print(f'{np.dtype(dt).name}: exp(x) overflows for x >',
              f'{np.log(np.finfo(dt).max):.2f}')
```

```
1 + eps   != 1 : True
1 + eps/2 == 1 : True
gap between float32 neighbors at 1    : 1.1920929e-07
gap between float32 neighbors at 2^20 : 0.125
float16: exp(x) overflows for x > 11.09
float32: exp(x) overflows for x > 88.72
float64: exp(x) overflows for x > 709.78
```

ここで $1 + \varepsilon_{\text{mach}}/2$ は 1 とその上側の隣接値との中点であるため、偶数丸めによって 1 に戻される。間隔の半分という最悪ケースの丸め誤差は実際に発生し、相対誤差は $u/(1+u)$ となって上界 $|\delta| \le u$ のわずかに内側に収まる。一方、2^{20} における間隔は 1 における間隔の 2^{20} 倍（約 100 万倍）である。そして最後の 3 行は e^x がオーバーフローする境界を特定しており、fp16 では 11.09、fp32 では 88.72、fp64 では 709.78 を超えるとオーバーフローする。

35.8.1.3 オーバーフロー、アンダーフロー、微小な更新

指数関数は加法的なスケールを乗法的なスケールに変えるため、さほど大きくないロジットでもオーバーフローしうる。fp16 では、上限 $\log 65504 \approx 11.09$ は通常の未正規化な分類器スコアがとりうる範囲の十分に内側にある。

その反対側では、e^{-x} がアンダーフローする。最小の正規化数を下回ると、形式は同じ絶対間隔を保ちながら有効ビットを 1 つずつ失う非正規化数を辿り、やがて完全に 0 へ丸められる。後から $\log 0$ をとると $-\infty$ が生じ、逆伝播は `inf` や `NaN` で埋め尽くされる。fp16 では $2^{-14} \approx 6.1 \times 10^{-5}$ 未満で非正規化数が始まり、最小の非正規化数は $2^{-24} \approx 6 \times 10^{-8}$ であり、$2^{-25} \approx 3 \times 10^{-8}$ 未満の値はすべてそのままゼロへ丸められる。

加算からは、訓練に直接関わる第 2 の絶対閾値が生じる。勾配ステップは重み w を $w - \eta g$ に置き換えるが、更新量 ηg が w 周辺の間隔に比べて小さいと、加算結果は単に元の w に丸め戻され、ステップは跡形もなく消えてしまう。

命題（微小な更新の消失）。$w > 0$ を、$2^e \le w < 2^{e+1}$ を満たす仮数部 p ビット形式の浮動小数点数とし、Δ を実数とする。$|\Delta| < 2^{e-p-1}$ ならば $\mathrm{fl}(w + \Delta) = w$ が成り立つ。ただし $w = 2^e$ かつ $\Delta < 0$ の場合は例外であり、このときは $|\Delta| < 2^{e-p-2}$ に対して同じ結論が成り立つ。

証明。 $[2^e, 2^{e+1}]$ では表現可能な数は 2^{e-p} 間隔で並ぶため、w の隣接値は $w \pm 2^{e-p}$ である。ただし $w = 2^e$ より下では 1 つ下のオクターブ $[2^{e-1}, 2^e)$ が始まり、間隔は半分の 2^{e-p-1} になる。最近傍丸めが w を返すのは、$w + \Delta$ がその側にある隣接グリッド点よりも w に近いとき、すなわち $|\Delta|$ がその間隔の半分より真に小さいときである。■

重みの更新 $\Delta = -\eta g$ において $2^e < w < 2^{e+1}$ のとき、この命題は次のようになる。

$$\mathrm{fl}(w - \eta g) = w \qquad \text{whenever } |\eta g| < 2^{e-p-1} = u\,2^e. \tag{35.97}$$

要するに、およそ $u|w|$ より小さい更新はすべて失われる。相対モデル Equation 35.96 は積や商をよく記述するが、大きさの大きく異なる 2 つの数を足し合わせる場合には、この鋭い絶対的なカットオフが見えなくなる。

$w = 1$ のようにちょうど 2 のべき乗の上にある重みは、命題の例外ケースに該当する。1 より下では間隔が半分になるため、増加方向では u 未満で消える（5.5 節で実験した通りである）のに対し、減少方向ではすでに $u/2$ 未満で消えてしまう。したがって bf16 では、1 に等しい重みは $2^{-9} \approx 0.002$ より小さいあらゆる減少を無視するため、学習率が 10^{-3} でオーダー 1 の勾配が与えられても重みは完全に凍結されたままになる。対応する閾値は fp16 では $2^{-12} \approx 2.4 \times 10^{-4}$、fp32 では $2^{-25} \approx 3 \times 10^{-8}$ であり、混合精度（mixed precision）訓練で重みのマスターコピーを fp32 で保持するのはこのためである。これらの閾値を跨ぐステップを試してみよう（bf16 の更新は fp32 で計算し、`to_bf16` で丸める）。

```
one32, one16 = np.float32(1.0), np.float16(1.0)
print('bfloat16: 1 - 1e-3 ->', to_bf16(one32 - np.float32(1e-3)).item())
print('bfloat16: 1 - 2e-3 ->', to_bf16(one32 - np.float32(2e-3)).item())
print('fp16    : 1 - 1e-4 ->', one16 - np.float16(1e-4))
print('fp16    : 1 - 3e-4 ->', one16 - np.float16(3e-4))
print('fp32    : 1 - 1e-3 ->', one32 - np.float32(1e-3))
```

```
bfloat16: 1 - 1e-3 -> 1.0
bfloat16: 1 - 2e-3 -> 0.99609375
fp16    : 1 - 1e-4 -> 1.0
fp16    : 1 - 3e-4 -> 0.9995
fp32    : 1 - 1e-3 -> 0.999
```

確かに、10^{-3} のステップでは bf16 の重みは 1 に張り付いたままであるのに対し、2×10^{-3} では 0.00195 の閾値を超えて隣接値 $1 - 2^{-8} = 0.99609375$ へ跳ぶ。同様に、fp16 は 10^{-4} を飲み込んでしまうが、3×10^{-4} に対しては $1 - 2^{-11} \approx 0.9995$ へ進み、fp32 は更新をそのまま保持する。

一方、勾配が直面するのはアンダーフローの閾値である。fp16 では、6.1×10^{-5} 未満の勾配は精度を失い、3×10^{-8} 未満の勾配はゼロに消える。逆伝播の前に損失へ定数 S を掛けると、すべての勾配が S 倍に拡大されて正規化数の範囲へ引き上げられ、その後 fp32 で S で割れば本来のスケールに戻る（実践的な手順は 5.5.3.3 節で扱い、演習問題 1 では S の許容範囲を求める）。

確率的丸め（stochastic rounding）(Gupta et al. 2015) は、これとは異なる方法で更新消失のカットオフを取り除く。常に最も近いグリッド点へ丸めるのではなく、格納される値の期待値が厳密な結果に一致するよう選ばれた確率でランダムに切り上げまたは切り捨てを行う。すると、最近傍丸めでは小さすぎて消えてしまう更新であっても、多数のステップにわたって期待値の意味で蓄積されるようになる（演習問題 5）。

35.8.2 安定なソフトマックスと交差エントロピー

ソフトマックス、log-sum-exp、交差エントロピーはいずれもロジットの指数をとるため、先ほど計算したオーバーフローとアンダーフローの閾値に正面から突き当たる。議論を具体的に保つため、本項を通して 1 つの通し例を用いる。ロジット $\mathbf{z} = (0, t)$ を持ち、正解ラベルが起こりにくい方のクラス 0 である 2 クラス分類器である。その確率は $\mathrm{softmax}(\mathbf{z}) = \big(1/(1+e^t),\, e^t/(1+e^t)\big)$、log-sum-exp は $\log(1+e^t)$ であり、クラス 0 に対する交差エントロピー損失（cross-entropy loss）は $\log(1+e^t) \approx t$（大きな t のとき）となる。これらの出力自体はいずれも極端な値ではなく、$t = 100$ であっても確率は 3.7×10^{-44} と 1、損失は単に 100 である。

35.8.2.1 最大値シフト

ロジット $\mathbf{z} \in \mathbb{R}^n$ が与えられたとき、ソフトマックスベクトルは次の式で定義される。

$$\mathrm{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}.$$

$t = 100$ でこの式をそのまま評価すると e^{100} が作られ、fp32 のオーバーフロー上限である 88.72 を超えてしまう。第 2 成分は `inf/inf`（`NaN`）になり、第 1 成分は $1/\infty = 0$ へ潰れる。真の確率自体は fp32 に余裕をもって収まるのであり、収まらないのは中間の指数関数だけであることに注目しよう。

幸い、分子と分母に同じ正の定数 e^{-c} を掛けても比は変わらないため、ソフトマックスは**シフト不変**（shift-invariant）である。

$$\mathrm{softmax}(\mathbf{z} - c\mathbf{1}) = \mathrm{softmax}(\mathbf{z}) \qquad \text{for every } c \in \mathbb{R}. \tag{35.98}$$

対数の中の和から e^c を括り出すと、ソフトマックスの分母の対数である **log-sum-exp** $\mathrm{lse}(\mathbf{z}) = \log \sum_j e^{z_j}$ についても対になる恒等式が得られる。

$$\mathrm{lse}(\mathbf{z}) = c + \log \sum_{j=1}^{n} e^{z_j - c}. \tag{35.99}$$

どちらの恒等式も任意の $c \in \mathbb{R}$ に対して成り立ち、$c = \max_j z_j$ と選べば両方の計算が安全になる。シフト後の指数 $z_j - c$ はすべて高々 0 となるため、どの指数関数もオーバーフローしない。同時に最大の項が $e^0 = 1$ に等しくなるため、和は必ず $[1, n]$ に収まり、その対数が $-\infty$ へアンダーフローすることもない。これを fp32 の $(0, 100)$、冒頭の fp64 の $(1000, 1001, 1002)$、そして両方式とも有限に留まる $(0, 10)$ で試してみよう。最後のケースではビット単位まで完全に同じ値が返るだろうか。

```python
def softmax_naive(z):
    e = np.exp(z)
    return e / e.sum()

def softmax_stable(z):
    e = np.exp(z - z.max())        # shift by the max: largest exponent is 0
    return e / e.sum()

z = np.array([0.0, 100.0], dtype=np.float32)
with np.errstate(over='ignore', invalid='ignore'):
    print('naive  softmax(0, 100):', softmax_naive(z))
print('stable softmax(0, 100):', softmax_stable(z))
z = np.array([1000.0, 1001.0, 1002.0])       # the opening's logits, fp64
with np.errstate(over='ignore', invalid='ignore'):
    print('naive  softmax(1000, 1001, 1002):', softmax_naive(z))
print('stable softmax(1000, 1001, 1002):', softmax_stable(z).round(4))
z = np.array([0.0, 10.0], dtype=np.float32)
a, b = softmax_naive(z), softmax_stable(z)
print('softmax(0, 10), naive minus stable, in ulps:',
      (a - b) / np.spacing(b))
```

```
naive  softmax(0, 100): [ 0. nan]
stable softmax(0, 100): [3.8e-44 1.0e+00]
naive  softmax(1000, 1001, 1002): [nan nan nan]
stable softmax(1000, 1001, 1002): [0.09   0.2447 0.6652]
softmax(0, 10), naive minus stable, in ulps: [2. 0.]
```

$t = 100$ では、シフト版ソフトマックスは $(3.8 \times 10^{-44}, 1)$（真の 3.7×10^{-44} が有効ビット 5 ビットを残して非正規化数のグリッドに丸められた値）を返すのに対し、直接評価する式は 0 と NaN を返す。精度を高くしても問題は解決しない。冒頭のロジット $(1000, 1001, 1002)$ は fp64 でさえ e^{1000} がオーバーフローし、直接評価する式は 3 つの NaN になるが、シフト版は問題なく $(0.09, 0.24, 0.67)$ を返す。$t = 10$ では、実数上の恒等式は丸めの順序まで一致することを保証しないため、2 つの式は最下位ビットまで一致するか、あるいは最終桁単位（unit in the last place、**ulp**）で 1〜2 単位だけ異なる。なお、境界ケースが 1 つ残る。最大値シフトは非有限のロジットを救うことはできず、行全体が $-\infty$ であると $c = -\infty$ となって $z_j - c$ が未定義になる。

35.8.2.2 log-sum-exp の上下界

log-sum-exp はあらゆる指数分布族（exponential family）モデルの対数正規化項であり（36.4 節）、1 回の減算を除けば交差エントロピー損失そのものである。シフト恒等式 Equation 35.99 から、この値が最大のロジットにどれほど近いかが直ちにわかる。

命題（log-sum-exp のサンドイッチ不等式）。 任意の $\mathbf{z} \in \mathbb{R}^n$ に対して次が成り立つ。

$$\max_j z_j \;\le\; \operatorname{lse}(\mathbf{z}) \;\le\; \max_j z_j + \log n.$$

証明。 $c = \max_j z_j$ を Equation 35.99 に代入する。各指数関数 $e^{z_j - c}$ は $(0, 1]$ に属し、最大値を達成する項は 1 に等しいため、和は $[1, n]$ に、その対数は $[0, \log n]$ に属する。これに c を加えれば上下両側の評価が得られる。■

したがって log-sum-exp は**ソフト最大値**（soft maximum）である。2 クラスの通し例では、

$\mathrm{lse}(0,t) = \log(1+e^t)$ は $\log 2 \approx 0.69$ 以内の差で $\max(0,t)$ に追随し、2 つのロジットが並ぶ $t=0$ で最大の差に達し、$|t|$ が大きくなるにつれて最大値にぴたりと漸近する（Figure 35.31）。

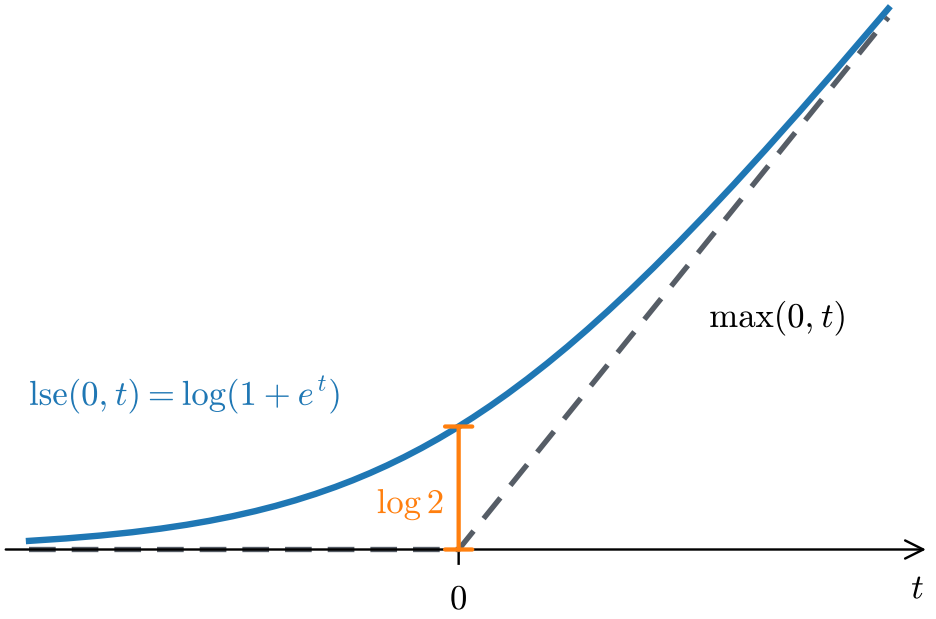


Figure 35.31: ソフト最大値としての log-sum-exp。$\mathrm{lse}(0,t) = \log(1+e^t)$ は $\log 2$ 以内の差で $\max(0,t)$ に収まり、$n=2$ におけるサンドイッチ不等式と一致する。差は 2 つのロジットが並ぶ $t=0$ で最大となり、$|t|$ が大きくなるにつれて消える。

さらに、その勾配はソフトマックスそのもの、すなわち $\partial\,\mathrm{lse}(\mathbf{z})/\partial z_i = e^{z_i}/\sum_j e^{z_j} = \mathrm{softmax}(\mathbf{z})_i$ である。ラベル y に対するロジットに関する交差エントロピーの勾配が $\mathrm{softmax}(\mathbf{z}) - \mathbf{e}_y$ へと簡約されるのはこのためであり、また 35.3 節では log-sum-exp が凸関数であることも示される。

シフトしたロジットからシフト後の log-sum-exp を引くと**対数ソフトマックス**（log-softmax）が得られ、確率そのものを一度も作らずに対数確率（log probability）を直接計算できる。

$$\log \mathrm{softmax}(\mathbf{z})_i = (z_i - c) - \log \sum_{j=1}^{n} e^{z_j - c}, \qquad c = \max_j z_j. \tag{35.100}$$

最終結果の指数をとらないため、微小な確率が対数をとる前にゼロへアンダーフローすることはない。ここで Equation 35.100 の微妙な点に注目しよう。代数的に同値な式 $z_i - \mathrm{lse}(\mathbf{z})$ は、まず大きな数 $\mathrm{lse} = c + \log\sum_j e^{z_j - c}$ を作ってからそれを大きな数 z_i から引くため、浮動小数点では目立って精度が落ちる。この括弧の付け方によってどれほどの差が生じるだろうか。ロジット $(1000, 1001, 1002)$ に対して fp32 で両方の形を比較してみよう。

```
z = np.array([1000.0, 1001.0, 1002.0], dtype=np.float32)
c = z.max()
s = np.log(np.exp(z - c).sum())          # log sum exp(z - c), in [0, log 3]
```

```
    with np.errstate(over='ignore'):
        print('naive log(sum(exp(z))):', np.log(np.exp(z).sum()))
    lse = c + s
    print('lse = c + s           :', lse, '  float32 gap at 1002:', np.spacing(c))
    log_p_sub = z - lse                       # forms lse, then subtracts it
    log_p = (z - c) - s                       # never forms c + s
    print(f'sum of exp(z - lse)    : {np.exp(log_p_sub).sum():.8f}')
    print(f'sum of exp((z - c) - s): {np.exp(log_p).sum():.8f}')
```

```
naive log(sum(exp(z))): inf
lse = c + s           : 1002.4076   float32 gap at 1002: 6.1035156e-05
sum of exp(z - lse)    : 1.00001323
sum of exp((z - c) - s): 1.00000000
```

直接計算は `inf` へオーバーフローする。$1000 > 709.78$ であるため fp64 でも同様であり、冒頭で `inf/inf` が 3 つの `NaN` になった原因もまさにこれである。シフトを用いた値 $\mathrm{lse} = 1002.4076$ は、サンドイッチ区間 $[1002, 1002 + \log 3]$ の内側に問題なく収まっている。さらに興味深いことに、対数ソフトマックスの 2 つの式は食い違う。$z_i - \mathrm{lse}$ の指数をとって得た確率の和は 1.0000132 になるのに対し、Equation 35.100 では和が 1.0000000 になる。

なぜ $z_i - \mathrm{lse}$ は精度を失ったのだろうか。意外なことに、減算自体は一切の誤差を生んでいない。2 つの浮動小数点数が互いに 2 倍以内の範囲にあるとき、両者はともに小さい方の数のグリッド間隔の整数倍であり、その差を表すのに余分なビットを必要としないため、差は厳密に表現可能だからである（Sterbenz の補題、演習問題 6）。

誤差が混入したのはそれより前、lse を格納した時点である。1002 付近では fp32 の数は $2^{-14} \approx 6.1 \times 10^{-5}$ 間隔で並んでいるため、真の値 1002.4076060 は 1.32×10^{-5} だけ切り捨てられる。その結果、$z_i - \mathrm{lse}$ として計算されたすべての対数確率は 1.32×10^{-5} だけ大きくなりすぎ、得られる各確率は $e^{1.32 \times 10^{-5}}$ 倍に膨らみ、それらの和は 1 を 1.32×10^{-5} だけ上回る。指数の絶対誤差は指数関数を通すと同程度の相対誤差に変わり、減算は入力にすでに潜んでいた誤差を露呈させたにすぎない。これが 35.8.3 節の中心テーマである。

35.8.2.3 ロジットからの交差エントロピー

ラベル y に対する交差エントロピー損失は負の対数ソフトマックスにほかならないため、Equation 35.100 からロジットを用いて直接表せる。

$$-\log \mathrm{softmax}(\mathbf{z})_y = \mathrm{lse}(\mathbf{z}) - z_y. \tag{35.101}$$

この方法で損失を計算すれば、必要なのは安定な log-sum-exp が 1 回と減算が 1 回だけであり、確率を一度も作る必要がない。通し例では、クラス 0 に対する損失は $\mathrm{lse}(0, t) - 0 = \log(1 + e^t)$ となる。これに対し、先に確率を計算すると、正解クラスの確率 $e^{-t}/(1 + e^{-t}) \approx e^{-t}$ は $t > 87.3$ で fp32 の正規化数の範囲を下回り、$t = 104$ 付近で完全にゼロへアンダーフローして、その負の対数を ∞ へ飛ばしてしまう。この崖を跨ぐ $t = 103$ と 104 を含めて、ライブラリの組み込み関数で 2 つの経路を比較してみよう。

```
    print('    t  CE from logits    CE via probabilities')
    for t in [20.0, 60.0, 103.0, 104.0]:
        logits = jnp.array([[0.0, t]])
        y = jnp.array([0])                          # label = the unlikely class
```

```
    from_logits = optax.softmax_cross_entropy_with_integer_labels(logits, y)
    from_probs = -jnp.log(jax.nn.softmax(logits)[0, 0])
    print(f't{t:5.0f}  {float(from_logits[0]):15.4f}  {float(from_probs):15.4f}')
```

```
   t  CE from logits    CE via probabilities
  20          20.0000          20.0000
  60          60.0000          60.0000
 103         103.0000              inf
 104         104.0000              inf
```

ロジットから直接計算すると、どの行でも損失は t に等しくなり、確率を経由する計算も $t = 20$ と 60 ではこれと一致する。しかし $t = 103$ では確率 e^{-103} が非正規化数の範囲に落ち込み、その先の振る舞いはライブラリが非正規化数をどう扱うかによって分かれる。

XLA はコンパイルされた演算を非正規化数のゼロフラッシュ付きで実行するため、確率は 0 になり、損失は `inf` へと $t = 103$ と 104 の両方で跳ね上がる。

実践上の教訓は明快である。実際に確率が必要になるまでは、損失、尤度（likelihood）、事後分布（posterior）をすべて対数空間に保っておくことである。3.3.5.2 節の融合損失が Equation 35.101 を実装しているのもこのためであり、37.1 節では交差エントロピーが何を測っているかを解釈し、36.11 節では同じアンダーフローを避けるために画素ごとの対数確率を数百個足し合わせている。

35.8.3 桁落ち

35.8.3.1 関数の条件数

fp32 では、$\log(1 + x)$ を $x = 10^{-8}$ で評価すると 0（相対誤差 100%）が返るのに対し、ライブラリ関数 `log1p` は 10^{-8} を返す。$\log(1 + x)$ は $x = 0$ 付近で本質的に計算が難しいのだろうか、それとも式の形に非があるのだろうか。こうした問いに体系的に答えるには、特定のアルゴリズムに依らず、関数が入力の誤差をどれほど強く増幅するかを測る量が必要である。

微分可能な関数 f の、点 x（ただし $f(x) \neq 0$）における**相対条件数**（relative condition number）とは、x に対する微小な相対摂動が $f(x)$ の相対摂動へと何倍になって現れるかを表す係数である。x を $x(1 + \delta)$ へ摂動させると、出力は一次近似で $f(x(1 + \delta)) \approx f(x) + xf'(x)\,\delta$ へ変化する。この変化量を $f(x)$ で割ると、次の増幅率が得られる。

$$\kappa_f(x) = \left| \frac{x\, f'(x)}{f(x)} \right|. \tag{35.102}$$

たとえば指数関数では $\kappa_f(x) = |x|$ であり、$x = 88$ において丸めの単位 1 つ分の入力誤差は出力で 88 単位分の誤差になる。これは先ほど対数ソフトマックスのコードセルで見た仕組みそのものであり、指数の絶対誤差が確率における同程度の相対誤差に変わった理由である。なお、κ_f は厳密な数学関数の一次の性質である。f をどのように評価するかについては何も語らず、また $f(x) = 0$ となる点では相対誤差が定義されないため破綻する。

減算は 2 つの入力をとり、その両方に同じ一次の評価が成り立つ。$\hat{a} = a(1+\delta_a)$ と $\hat{b} = b(1+\delta_b)$ が相対誤差 $|\delta_a|, |\delta_b| \leq u$ を持ち、減算自体は厳密であるとする。このとき $\hat{a} - \hat{b} = (a - b) + (a\delta_a - b\delta_b)$ であるから、三角不等式（triangle inequality）より差の相対誤差は次のように抑えられる。

$$\frac{|(\hat{a}-\hat{b})-(a-b)|}{|a-b|} = \frac{|a\delta_a - b\delta_b|}{|a-b|} \le \frac{|a|+|b|}{|a-b|}\,u. \tag{35.103}$$

係数 $(|a|+|b|)/|a-b|$ は減算の条件数であり、$a \approx b$ となると急激に増大する。すでに丸め誤差を含んでいるほぼ等しい 2 つの数を引く現象は**桁落ち**（catastrophic cancellation）と呼ばれる。正確な上位桁同士が打ち消し合い、下位桁にあったノイズが答えの上位桁へと繰り上がるのである。参考として、代表的な問題の条件数を以下の表にまとめる。

問題	κ_f	性質・領域
$a-b$	$(\lvert a\rvert+\lvert b\rvert)/\lvert a-b\rvert$	$a \approx b$ のとき大
$\log(1+x)$	$x/((1+x)\log(1+x))$	1.000（$x=10^{-8}$）、0.82（$x=0.5$）
e^x	$\lvert x\rvert$	大きな $\lvert x\rvert$ で大
$1-\cos x$	$x\sin x/(1-\cos x)$	2 に収束（$x\to 0$）
$\sqrt{x}$	$1/2$	常に一定
データの分散（平均 m、ばらつき s）	高々 $2(1+\lvert m\rvert/s)$	後述

2 行目から冒頭の問いに決着がつく。$\log(1+x)$ は $x=0$ 付近で完全に良条件である（$\kappa_f \approx 1$）。素朴なアルゴリズム `log(1 + x)` が破綻するのは、悪条件な中間ステップへ迂回するためである。この式はまず $y = 1+x$ を作り、続いて $\log y$ を $y \approx 1$ で評価するが、そこでは対数関数が $\kappa_{\log}(y) = 1/|\log y| \approx 1/x = 10^8$ を持つ。$1+x$ を丸めることでオーダー u の相対誤差が y に生じ、それを $\log$ が 10^8 倍に拡大するため、fp32 では 100% を超える誤差になる。対照的に、`log1p` は $1+x$ の丸め誤差が結果に入り込まないよう書かれており、ごく小さい x に対しては単に x を返す。

ここから本節全体を貫く基本原理が得られる。すなわち、**不安定なアルゴリズムとは、良条件な問題を悪条件な中間ステップへ迂回させるものである。** κ_f が小さいにもかかわらず計算結果が悪いなら、原因はアルゴリズムにあるので式を書き換えればよい。逆に κ_f 自体が大きいなら、いかなるアルゴリズムでも答えを救うことはできず、代わりに問題そのものを変えなければならない。これが 35.8.4 節の主題である。

35.8.3.2 ほぼ等しい数の減算

これをコードで確かめよう。以下のセルでは $\log(1+x)$ の実験を再現したあと、10 進法の先頭 7 桁が一致する 2 つの fp32 の数 1.0002344 と 1.0002341（真の差は 3.0×10^{-7}）を減算する。

```
x = np.float32(1e-8)
print('float32 rounds 1 + x to:', np.float32(1.0) + x)
print('log(1 + x) =', np.log(np.float32(1.0) + x), '   log1p(x) =', np.log1p(x))
a, b = np.float32(1.0002344), np.float32(1.0002341)
print(f'stored a = {float(a):.10f}, b = {float(b):.10f}')
print('a - b =', a - b, '=', (a - b) / np.spacing(np.float32(1.0)), 'ulps of 1')
print('float64: log(1 + 1e-20) =', np.log(1 + 1e-20),
      '  log1p(1e-20) =', np.log1p(1e-20))
```

```
float32 rounds 1 + x to: 1.0
log(1 + x) = 0.0    log1p(x) = 1e-08
stored a = 1.0002343655, b = 1.0002341270
```

```
a - b = 2.3841858e-07 = 2.0 ulps of 1
float64: log(1 + 1e-20) = 0.0   log1p(1e-20) = 1e-20
```

1 と 2 の間では、fp32 におけるあらゆる差は $\varepsilon_{\text{mach}} = 1.19 \times 10^{-7}$ の整数倍であり、真の差 3.0×10^{-7} はそのグリッド上に乗らない。減算の結果は 2 ulp にあたる 2.38×10^{-7}（20% 小さい）となったが、今回も減算自体は誤差を一切生んでいない。2 つの入力は 1.0002343655 と 1.0002341270 として格納されており、セルが出力したのはそれらの厳密な差であって、誤差は格納された入力に最初から焼き込まれていたのである。ここでの桁落ち係数 Equation 35.103 は $2.0005/(3 \times 10^{-7}) \approx 7 \times 10^{6}$ である。これに $u \approx 6 \times 10^{-8}$ を掛けると最大 0.4 の相対誤差が許容され、観測された 0.2 はまさにその上界の内側に収まっている。

同じ桁落ちは、$\log(1+x)$ や $e^{x}-1$ の 0 付近（`log1p` と `expm1` がこれを回避する）、$1-\cos x$ の 0 付近（$2\sin^2(x/2)$ と書き換えられる）、および二次方程式の解の公式で $b^2 \gg 4|ac|$ のとき（演習問題 2）にも現れる。これらのどのケースでも、代数的な書き換えによって問題となる減算を取り除くことができる。

減算を消去できず妥協を受け入れるしかない場面が有限差分（finite difference）である。数値勾配チェックでは、自動微分（automatic differentiation）を前進差分 $(f(x+h)-f(x))/h$ と比較する。ステップ幅 h はどれほど小さくすべきだろうか。テイラーの定理より、数学的な打ち切り誤差（truncation error）は約 $\frac{h}{2}|f''(x)|$ であり、$h \to 0$ とともに縮小する。一方、各関数評価にはおよそ $u|f(x)|$ の丸め誤差が伴い、その差を h で割ると誤差は約 $2u|f(x)|/h$ に拡大され、$h \to 0$ とともに発散する。総誤差が最小になるのはこれら 2 項が釣り合う次の点である。

$$h^{\star} = 2\sqrt{\frac{u\,|f(x)|}{|f''(x)|}}, \qquad \text{with error about } 2\sqrt{u\,|f(x)f''(x)|}.$$

最適な誤差は $\sqrt{u}$ に比例するため、最良の前進差分であっても形式が持つ桁数の約半分しか維持できない。$f = \sin$、$x = 1$ の場合、ここから fp64 では $h^{\star} \approx 2 \times 10^{-8}$（誤差 1.8×10^{-8}）、fp32 では $h^{\star} \approx 5 \times 10^{-4}$ と予測される。両方の形式で h を 10^{-2} から 10^{-12} までスイープし、このトレードオフを直接確認しよう。

```
print('      h   error in fp64   error in fp32')
for h in 10.0 ** -np.arange(2, 13):
    err = [abs(float((np.sin(t(1) + t(h)) - np.sin(t(1))) / t(h)) - np.cos(1.0))
           for t in (np.float64, np.float32)]  # forward difference of sin at 1
    print(f'{h:7.0e}  {err[0]:14.1e}  {err[1]:14.1e}')
```

```
      h   error in fp64   error in fp32
  1e-02         4.2e-03         4.2e-03
  1e-03         4.2e-04         4.0e-04
  1e-04         4.2e-05         2.8e-04
  1e-05         4.2e-06         3.9e-03
  1e-06         4.2e-07         6.3e-02
  1e-07         4.2e-08         5.6e-02
  1e-08         3.0e-09         5.4e-01
  1e-09         5.3e-08         5.4e-01
  1e-10         5.8e-08         5.4e-01
  1e-11         1.2e-06         5.4e-01
  1e-12         4.3e-05         5.4e-01
```

fp64 では、誤差は h に比例して 4×10^{-3}（打ち切り誤差が支配的な $h = 10^{-2}$）から約 10^{-8}

($h = 10^{-8}$ 付近；個々の h における正確な値は丸めノイズで変動する）まで減少し、その後は桁落ちが支配的になって 4×10^{-5}（$h = 10^{-12}$）へと再び上昇する。fp32 では、$h = 10^{-3}$〜10^{-4} にある最適領域でも正しい桁は約 3 桁しか残らず、$h = 10^{-8}$ 以下では $1 + h$ がそのまま 1 に丸められて差分商が 0 となり、誤差は導関数（derivative）全体の値 $\cos 1 = 0.54$ に一致してしまう。数値勾配チェックを常に fp64 で実行すべき理由はここにある。

また、精度を高くしても桁落ちは決して治らず、破綻が起きる入力がより小さい領域へ押しやられるだけであることにも注意しよう。先ほどの `log1p` のセルの最終行では、fp64 でも `log(1 + 1e-20)` が 0 を返し（fp32 で 10^{-8} のときに起きたのと同様）、一方で `log1p` は両方の形式で正確さを保っていた。式を定式化し直すことで、悪条件なステップそのものが取り除かれるのである (Higham 2002)。

35.8.3.3 1 パス分散

桁落ちは、一見まったく無害に見える公式の中にも潜んでいる。標本分散（sample variance）の計算はその古典的な例であり、正規化層やデータ前処理パイプラインが絶えず分散を計算する深層学習にも直結する問題である。

データ $x_1, \dots, x_N$ が平均 m と標準偏差 s を持つとき、教科書でおなじみの次の恒等式がある。

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - m)^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - m^2,$$

この式を見ると、x_i と x_i^2 の和だけを累積して 1 パスで分散を計算したくなる。厳密な算術ではこの恒等式に何の欠陥もないが、浮動小数点では罠になる。$s \ll |m|$ のとき、右辺の 2 項はいずれも m^2 付近にある一方でその差は s^2 であるから、桁落ち係数 Equation 35.103 はおよそ $2m^2/s^2$ になる。$m = 10^9$、$s = 1$ の場合、この係数は 2×10^{18} に達し、fp64 が吸収できる 10^{16} を 2 桁も上回る。

この精度喪失のうち、問題自体によって避けられない部分はどれだけだろうか。分散の条件数を調べてみよう。各入力 x_i を相対誤差 δ_i（$|\delta_i| \le u$）で摂動させ、x_i を $x_i + x_i\delta_i$ に置き換える。x_i に関する分散の微分は $\frac{2}{N}(x_i - m)$ となる。誘導される m の変化には $\sum_i (x_i - m) = 0$ が掛かるからである。$x_i = (x_i - m) + m$ と書けば、分散の一次の変化量は次のようになる。

$$\frac{2}{N}\sum_{i=1}^{N}(x_i - m)\, x_i\delta_i = \frac{2}{N}\sum_{i=1}^{N}(x_i - m)^2\delta_i + \frac{2m}{N}\sum_{i=1}^{N}(x_i - m)\,\delta_i.$$

第 1 の和の大きさは高々 $2s^2 u$ である。コーシー・シュワルツの不等式より、第 2 の和の大きさは高々 $2|m|u \cdot \frac{1}{N}\sum_i |x_i - m| \le 2|m| s\, u$ である。s^2 で割ると、分散の相対条件数は高々 $2(1 + |m|/s) \approx 2|m|/s$、今の例では約 2×10^9 であることがわかる（$\delta_i = u\,\mathrm{sign}(x_i - m)$ と選べば、ガウス分布に従うデータでは $1.6\,|m|/s$ となる）。したがって問題自体が許容する誤差は $2 \times 10^9 u \approx 2 \times 10^{-7}$ にすぎず、約 7 桁の精度が残るはずである。これに対し、素朴な 1 パス公式はオーダー $(m/s)^2 u$ の誤差を被る。35.8.4 節で正規方程式が行列の条件数を 2 乗するのと同様に、この公式は条件数を **2 乗**してしまうのである。

安定な 1 パスのアルゴリズムは、すべての偏差をスケール s に保ち、m^2 のスケールには持ち込まない。Welford (1962) は逐次平均 m_k と**中心化**（centered）二乗の逐次和 $M_k = \sum_{i \le k}(x_i - m_k)^2$ を保持し、新しい各サンプルと直前の平均との偏差 $d_k = x_k - m_{k-1}$ から両者を次のように更新する。

$$m_k = m_{k-1} + \frac{d_k}{k}, \qquad M_k = M_{k-1} + d_k\,(x_k - m_k). \tag{35.104}$$

M_k の更新における非対称性に注目しよう。**古い**平均からの偏差と、**新しい**平均からの偏差を掛け合わせている。まさにこの非対称性によって、漸化式が厳密に成り立つのである。

命題（Welford の漸化式）。 $m_0 = M_0 = 0$ **とするとき、漸化式 Equation 35.104 はすべての** $k \ge 1$ **に対して厳密な算術のもとで次を満たす。**

$$m_k = \frac{1}{k}\sum_{i=1}^{k} x_i, \qquad M_k = \sum_{i=1}^{k} (x_i - m_k)^2.$$

証明。 平均の恒等式 $k\,m_k = (k-1)\,m_{k-1} + x_k$ は第 1 の漸化式から直ちに導かれる。M_k については k に関する帰納法で示す。$m_k - m_{k-1} = d_k/k$ および $x_k - m_k = d_k\,(k-1)/k$ を用い、第 k 項を切り離して最初の $k-1$ 項を m_{k-1} の周りで中心化し直す。

$$\begin{aligned}\sum_{i=1}^{k}(x_i - m_k)^2 &= \sum_{i=1}^{k-1}\Big((x_i - m_{k-1}) - \tfrac{d_k}{k}\Big)^2 + (x_k - m_k)^2 \\ &= M_{k-1} + (k-1)\tfrac{d_k^2}{k^2} + \tfrac{(k-1)^2}{k^2} d_k^2 .\end{aligned}$$

交差項は $\sum_{i\le k-1}(x_i - m_{k-1}) = 0$ により消え、帰納法の仮定より最初の和は M_{k-1} に一致する。残る 2 項をまとめると $\tfrac{k-1}{k}\,d_k^2 = d_k \cdot d_k\tfrac{k-1}{k} = d_k(x_k - m_k)$ となり、漸化式が加える項とちょうど一致する。$\blacksquare$

Welford の漸化式が浮動小数点でも精度を保つことの証明はより難しく、ここでは直感的なあらすじのみを述べる（完全な解析は Higham (2002) を参照）。一見すると、x_k と m_{k-1} はともに 10^9 付近にあるため、Welford の更新でも大きな数同士を減算しているように見える。しかし Sterbenz の補題によりこの減算は厳密であり、x_k も厳密な入力データであるため、d_k に含まれる誤差は逐次平均に由来するものだけ、すなわち $|m|$ のスケールで 10^5 ステップにわたり高々数十 ulp にとどまる。したがって M_k に加えられる量はすべてスケール s の偏差となり、更新時の丸め誤差は s^2 に対する相対誤差として入り、m^2 に対するものにはならない。平均に一定のオフセット ε が加わっても、中心化二乗和のずれは $N\varepsilon^2$ にとどまる。一次の項には $\sum_i (x_i - m) = 0$ が掛かるからである。実際、Welford 法の誤差は問題自体の感度 $u|m|/s \approx 1.1 \times 10^{-7}$ に達するのに対し、素朴な 1 パス公式は $(m/s)^2 u \approx 100$ もの誤差を被る。

素朴な 1 パス法、Welford 法、2 パスの基準値という 3 つの手法すべてを、10^5 個の fp64 サンプル（平均 10^9、分散 1）で比較してみよう。

```
rng = np.random.default_rng()
x = 1e9 + rng.normal(0.0, 1.0, size=100_000)    # mean 1e9, variance 1

naive = (x**2).mean() - x.mean()**2             # one pass, cancels
two_pass = ((x - x.mean())**2).mean()           # subtract the mean first

m, M = 0.0, 0.0                                 # Welford: one pass
for k, xk in enumerate(x, start=1):
    d = xk - m
    m += d / k                                  # m_k
```

```
    M += d * (xk - m)                          # M_k
welford = M / len(x)

print(f'naive E[x^2] - E[x]^2 : {naive:12.6f}   (= {naive / 128:.0f} x 128)')
print(f'Welford, one pass     : {welford:12.6f}')
print(f'two-pass reference    : {two_pass:12.6f}')
print(f'Welford vs two-pass   : {abs(welford - two_pass) / two_pass:.1e} relative')
print('float64 gap near 1e18 :', np.spacing(1e18))
```

```
naive E[x^2] - E[x]^2 :  -256.000000   (= -2 x 128)
Welford, one pass     :     0.992369
two-pass reference    :     0.992369
Welford vs two-pass   : 8.8e-09 relative
float64 gap near 1e18 : 128.0
```

素朴な 1 パス推定値は厳密に 128 の整数倍（典型的には 0、± 128、± 256）として出力され、真の分散 1 には遠く及ばず、頻繁に負の値にさえなる。なぜ 128 なのだろうか。$\frac{1}{N}\sum x_i^2$ と m^2 はともに $[2^{59}, 2^{60})$ に属し、そこでは隣り合う fp64 の数が $2^{59-52} = 128$ 離れているため、両者の差は 128 の倍数にしかなりえない。予測された約 100 の誤差がグリッドに量子化された姿である。一方、Welford の漸化式は 2 パスの基準値と少なくとも 7 桁一致し、逆符号の丸め誤差が和全体で部分的に打ち消し合うため 8〜9 桁一致することも多い。

データセット全体のストリーミング統計量を計算する場合など、1 パスで分散を累積する場面では常に同じ注意が必要になる。（なお、バッチ正規化が推論用に保持する移動モーメントは指数移動平均（exponential moving average、7.3 節）であり、大きな逐次和同士を減算することはない。）

35.8.3.4 総和の順序

素朴な分散も 2 パスの基準値も 10^5 個の数を足し合わせるが、浮動小数点では加算の順序が結果に影響する。各加算は $(1+\delta_k)$ の因子を導入するため、単純な左から右への総和では最初の項が $n-1$ 回すべての丸めを通過する。u の一次までで、左から右への総和は次を満たす。

$$\left|\mathrm{fl}\Big(\sum_{i=1}^{n} x_i\Big) - \sum_{i=1}^{n} x_i\right| \le (n-1)\,u \sum_{i=1}^{n} |x_i|.$$

初期の項がその後のすべての加算の丸め誤差を蓄積するため、最悪ケースの誤差は n に対して線形に増大する。**ペアワイズ**総和法は配列の前半と後半を再帰的に足し合わせることでこれを回避し、どの項も $\log_2 n$ 回の加算しか通過しないため、上界は $u \log_2 n \sum_i |x_i|$ へと縮小する。NumPy の `sum` はこの考え方のブロック版を用いている。Kahan の補償付き総和法はさらに一歩進め、各加算の丸め誤差を逐次補正変数に保持することで n に依存しない誤差上界を達成する（演習問題 8）。10^7 個の fp32 の 0.1（格納されている値は 0.100000001490 ...）を左から右への順序とペアワイズの両方で足し合わせてみよう。ペアワイズ用の配列はゼロでパディングして 2^{24} 要素にし、隣り合うペアの加算を繰り返す。

```
x = np.full(10_000_000, 0.1, dtype=np.float32)
exact = len(x) * float(x[0])                # x[0] = 0.100000001490...
s = np.concatenate([x, np.zeros(2**24 - len(x), np.float32)])  # pad to 2^24
while len(s) > 1:                           # pairwise: add neighbors, halve
    s = s[0::2] + s[1::2]
```

```
    print(f'in order (cumsum): {np.cumsum(x)[-1]:16.4f}')
    print(f'pairwise         : {s[0]:16.4f}')
    print(f'exact            : {exact:16.4f}')
```

```
in order (cumsum):     1087937.0000
pairwise         :     1000000.0000
exact            :     1000000.0149
```

左から右への総和は $1{,}087{,}937$ で終わり、上界 $(n-1)u\sum_i |x_i| \approx 6\times 10^5$ の内側にはあるものの、ほぼ 9% も大きすぎる。ペアワイズ総和法は $1{,}000{,}000$ に着地し、厳密な和との差は 0.015 以内であり、その上界 $u\log_2 n\sum_i |x_i| \approx 1.4$ の十分内側に収まっている。

左から右への総和におけるドリフトの犯人は、微小な重み更新を消し去ったのと同じグリッド効果である。各被加数は、その時点の逐次和のグリッド間隔へ丸められるのである。和が $2^{19}=524{,}288$ を超えると fp32 の数は $1/16$ 間隔になるため、0.1 は 1.6 目盛り分に相当し、毎回の加算が 2 目盛り分の 0.125（25% 過大）へと切り上げられる。2^{20} を超えると間隔は倍の $1/8$ になり、各被加数は再び 0.125 へ丸められる。それより前の 2^{17} から 2^{19} の間では、逆に 0.09375 へ切り捨てられていた。浮動小数点の加算は結合法則を満たさないため、GPU スレッド間の並列リダクションはデバイスごと、あるいは実行ごとにビット単位まで同一の結果を生むとは限らない。幸い、Welford の漸化式もペアワイズリダクションに対応している。演習問題 7 の統合則は 2 つの部分集合の統計量を結合するものであり、分散訓練においてデバイス間で逐次モーメントを統合する際にもこの方法が用いられている。

35.8.4 条件性

行列の条件数とリッジ回帰（ridge regression）のフィルタ係数は 33.5 節で展開した。ここで新たに扱うのは、前進誤差と後退誤差、後退誤差を測る公式、そして正規方程式（normal equations）が桁数の面で支払う代償である。

35.8.4.1 ほぼ平行な 2 直線

ここまでは計算の答えが 1 つの数であった。連立一次方程式では答えはベクトルになり、33.5.3.2 節では次の連立一次方程式を取り上げた。

$$x_1 + x_2 = 2, \qquad x_1 + 1.0001\,x_2 = 2.0001,$$

この例は、そのような答えがいかに脆くなりうるかを示している。解は $(1,1)$ であるが、第 2 式の右辺を 10^{-4}（データの相対変化にして 3.5×10^{-5}）だけ増やすと、解は $(0,2)$ へ移動する。各方程式は平面上の直線であり、解は 2 直線の交点である。これら 2 直線は傾き -1 と -0.9999 を持ち、0.003° の角度で交わるため、一方をわずかに平行移動させるだけで交点が他方の直線に沿って大きく滑ってしまう（Figure 35.32）。

同じ数値を逆向きに読んでみよう。あるソルバが元の方程式に対して $\hat{\mathbf{x}} = (0,2)$ を返したとする。真の解までの距離である**前進誤差**（forward error）は 100% である。しかし $(0,2)$ は、b_2 を 10^{-4} だけ変えた近傍の問題の厳密解でもある。このようなデータの変更量のうち最小のものが**後退誤差**（backward error）である。ごく小さな後退誤差を持つソルバは近傍の問題を厳密に解いているが、それでも真の答えからは大きく離れうる。

どれほど離れうるかはソルバではなく問題の性質である。ここで再び 35.8.3.1 節の条件数が、今度は関数 $\mathbf{b}\mapsto \mathbf{A}^{-1}\mathbf{b}$ に対して現れる。すなわち、あらゆる右辺とあらゆる変化の方向にわたってとった、入力の相対変化が出力の相対変化へと拡大されうる最大の倍率である。特異値

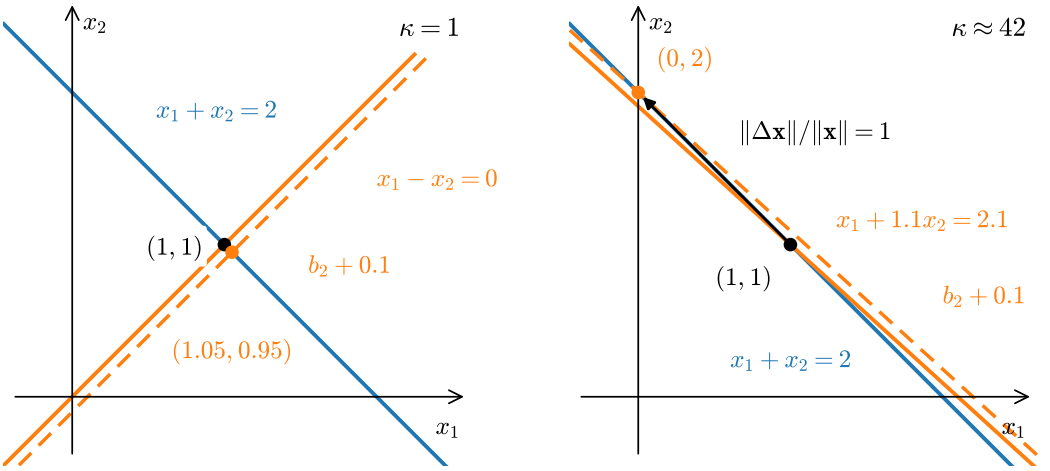


Figure 35.32: 連立一次方程式は、表す直線がほぼ平行なときに敏感になる。(a) 直交する 2 直線が $(1, 1)$ で交わる場合。第 2 の直線を 0.1 ずらすと交点は 5% 移動する（増幅率 1、$\kappa = 1$）。(b) 2 直線 $x_1 + x_2 = 2$ と $x_1 + 1.1x_2 = 2.1$ も $(1, 1)$ で交わるが、同じだけずらすと交点は $(0, 2)$ へ移動し、100% の変化が 3.4% の $\mathbf{b}$ の変化から生じる（増幅率 29、上界は $\kappa \approx 42$）。本文の方程式は 1.0001 を 1.1 の代わりに置いたものであり、なす角 $0.003°$、$\kappa \approx 4 \times 10^4$ で同じ幾何構造を持つ。

$\sigma_1 \geq \cdots \geq \sigma_d > 0$ を持つ行列では、その倍率は $\kappa(\mathbf{A}) = \sigma_1/\sigma_d$ であり（33.5.3.2 節）、先ほどの 2 直線では約 4×10^4 になる。シフト $(0, 10^{-4})$ は最も増幅される方向と一致していないため、そのうちの 2.8×10^4 倍を実現している。

35.8.4.2 後退誤差と前進誤差

連立一次方程式 $\mathbf{Ax} = \mathbf{b}$ と計算解 $\hat{\mathbf{x}}$ に対し、前進誤差は $\|\hat{\mathbf{x}} - \mathbf{x}\|/\|\mathbf{x}\|$ であり、後退誤差は $\hat{\mathbf{x}}$ を厳密解にする最小のデータ変化の大きさである。アルゴリズムの後退誤差が u のオーダーにあるとき、すなわち与えられた問題からデータの丸め約 1 回分だけ異なる問題の厳密解を返すとき、そのアルゴリズムは**後退安定**（backward stable）であるという。特異値分解（SVD）や QR 分解は後退安定であり、`np.linalg.solve` が用いるピボット選択付き LU 分解も実用上そうである。一方、1 パス分散は後退安定ではない。いかなるデータセットの分散も非負であるため、負の出力はいかなるデータの厳密な分散にもなりえないからである。

この定義はデータのあらゆる変化にわたる最小化を要求する。幸い、行列のみの変化に対しては残差（residual）から計算できる閉じた形の式が存在する。

命題（残差からの後退誤差）。 $\hat{\mathbf{x}} \neq \mathbf{0}$ とし、$\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ とおく。**最小の $\|\Delta\mathbf{A}\|_2$（ただし $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$ を満たすもの）は $\|\mathbf{r}\|/\|\hat{\mathbf{x}}\|$ であり、$\Delta\mathbf{A} = \mathbf{r}\hat{\mathbf{x}}^{\top}/\|\hat{\mathbf{x}}\|^2$ によって達成される。**

証明。 そのような任意の $\Delta\mathbf{A}$ は $\Delta\mathbf{A}\hat{\mathbf{x}} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}} = \mathbf{r}$ を満たすため、$\|\mathbf{r}\| \leq \|\Delta\mathbf{A}\|\,\|\hat{\mathbf{x}}\|$ が成り立つ。ランク 1 行列 $\mathbf{r}\hat{\mathbf{x}}^{\top}/\|\hat{\mathbf{x}}\|^2$ は $\hat{\mathbf{x}}$ を $\mathbf{r}$ に写し、そのノルムは $\|\mathbf{r}\|\,\|\hat{\mathbf{x}}\|/\|\hat{\mathbf{x}}\|^2 = \|\mathbf{r}\|/\|\hat{\mathbf{x}}\|$ である。■

$\mathbf{A}$ の大きさに対する相対的な後退誤差は次のようになる。

$$\omega = \frac{\|\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\|}{\|\mathbf{A}\|\,\|\hat{\mathbf{x}}\|}. \tag{35.105}$$

残差が小さければ、問題の条件がどうであれ後退誤差が小さいことが保証される。2 直線の方程式において $\hat{\mathbf{x}} = (0, 2)$ とすると、$\mathbf{r} = (0, -10^{-4})$ であり、$\omega = 10^{-4}/(2.00005 \cdot 2) = 2.5 \times 10^{-5}$ となる。$\mathbf{b}$ にも摂動を許す定式化は Rigal と Gaches によるものである（Higham (2002) を参照）。浮動小数点では残差自体も丸め誤差を伴って計算されるため、出力された ω の値のうち u 程度より小さいものはノイズである。

条件数は後退誤差を前進誤差へと変換する。これはぜひ覚えておくべき結果であり、その証明に必要なのは 2 つのノルム $\|\mathbf{A}\| = \sigma_1$ と $\|\mathbf{A}^{-1}\| = 1/\sigma_d$ のみである（33.5 節）。

命題（前進誤差 $\le$ 条件数 $\times$ 後退誤差）。 *$\mathbf{A}$ を正則行列とし、$\mathbf{A}\mathbf{x} = \mathbf{b}$ とする。また、計算解 $\hat{\mathbf{x}}$ が近傍の方程式 $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$（ただし $\|\Delta\mathbf{A}\| \le \omega \|\mathbf{A}\|$）を厳密に解くとする。このとき次が成り立つ。*

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \;\le\; \kappa(\mathbf{A})\,\omega. \tag{35.106}$$

証明。 $\mathbf{A}\mathbf{x} = \mathbf{b}$ を $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$ から引くと $\mathbf{A}(\hat{\mathbf{x}} - \mathbf{x}) = -\Delta\mathbf{A}\,\hat{\mathbf{x}}$ となり、したがって $\hat{\mathbf{x}} - \mathbf{x} = -\mathbf{A}^{-1}\Delta\mathbf{A}\,\hat{\mathbf{x}}$ から次が得られる。

$$\|\hat{\mathbf{x}} - \mathbf{x}\| \le \|\mathbf{A}^{-1}\|\,\|\Delta\mathbf{A}\|\,\|\hat{\mathbf{x}}\| \le \frac{1}{\sigma_d}\,\sigma_1\,\omega\,\|\hat{\mathbf{x}}\| = \kappa(\mathbf{A})\,\omega\,\|\hat{\mathbf{x}}\|. \qquad \blacksquare$$

ここでの誤差は、手元にあるベクトル $\hat{\mathbf{x}}$ に対する相対値として測られている。真の解 $\mathbf{x}$ に対する相対誤差については、$\|\hat{\mathbf{x}}\| \le \|\mathbf{x}\| + \|\hat{\mathbf{x}} - \mathbf{x}\|$ を代入して $\|\hat{\mathbf{x}} - \mathbf{x}\|$ について解くと、上界 $\kappa\omega/(1 - \kappa\omega)$（$\kappa\omega < 1$ のとき）が得られ、一次近似では同じである。2 直線の方程式では $\kappa\omega = 4 \times 10^{4} \cdot 2.5 \times 10^{-5} = 1.0$ であり、$\hat{\mathbf{x}}$ に対する相対誤差は $\|(-1, 1)\|/\|(0, 2)\| = 0.71$ となって上界の内側に収まる。行列ではなく右辺の誤差に対する対応する上界は 33.5.3.2 節に示されている。

後退安定なソルバは $\omega \approx u$ を満たし、この上界の $\log_{10}$ をとると次の経験則が得られる。

$$\text{correct digits in } \hat{\mathbf{x}} \;\approx\; \text{digits carried by the format} \;-\; \log_{10}\kappa(\mathbf{A}).$$

この経験則は最悪ケースの評価である。$\kappa = 10^{3}$ の行列では、fp32 が持つ 7 桁のうち 3 桁が失われうる。一方、本章の通し例である $\tfrac{1}{2}(x_1^2 + 1000x_2^2)$ は同じ κ を持つにもかかわらず 1 桁も失わない。行列が対角行列であり、各未知数が正しく丸められる 1 回の除算だけで求まるからである。

35.8.4.3 例：ヒルベルト行列

ヒルベルト行列（Hilbert matrix）$\mathbf{H}_{ij} = 1/(i + j - 1)$（$i, j = 1, \dots, n$）は、$n$ に対して指数関数的に増大する条件数を持ち、その各行は先ほどの 2 直線と同様にしてほぼ平行になっていく。ここでは $\mathbf{H}\mathbf{x} = \mathbf{b}$ を $\mathbf{b} = \mathbf{H}\mathbf{1}$（厳密解は $\mathbf{1}$）として解き、前進誤差を先ほどの経験則および後退誤差 Equation 35.105 と比較する。ソルバを評価する前に 1 つ注意点がある。$\mathbf{b} = \mathbf{H}\mathbf{1}$ を fp64 で計算した時点で $\mathbf{b}$ の各成分が丸められ、ソルバが動き出す前に相対サイズ u のデータ変化が生じているのである。（$1/3$ や $1/7$ などの行列成分の丸め自体は、格納された行列から $\mathbf{b}$ を作っているためここでは悪影響を与えない。）この丸めだけでどれほどの誤差が生じるだろうか。すべての fp64 の数は有理数であるから、格納された連立一次方程式は `Fraction` 型の成分に対するガウス・ジョルダン消去法によって丸め誤差を一切伴わずに解くこともできる。以下の表では、その厳密求解の誤差を `np.linalg.solve` の誤差と並べて示す。

```
    def solve_exact(A, b):
        """Gauss-Jordan elimination in exact rational arithmetic."""
        M = [[Fraction(a) for a in row] + [Fraction(c)] for row, c in zip(A, b)]
        n = len(M)
        for j in range(n):              # no pivoting: A is positive definite here
            for i in range(n):
                if i != j:
                    r = M[i][j] / M[j][j]
                    M[i] = [a - r * c for a, c in zip(M[i], M[j])]
        return np.array([float(M[i][n] / M[i][i]) for i in range(n)])
```

次元 $n \in \{4, 6, 8, 10, 12\}$ に対して $n \times n$ のヒルベルト行列 `H` と右辺 `b = H @ x_true` を構成し、浮動小数点ソルバ `np.linalg.solve(H, b)` と `solve_exact(H, b)` を条件数、前進誤差、保持桁数、後退誤差にわたって比較しよう。

```
    print(' n     kappa  forward error  digits kept  16 - log10 kappa  backward error'
          '  exact solve error')
    for n in [4, 6, 8, 10, 12]:
        i = np.arange(n)
        H = 1.0 / (1.0 + i[:, None] + i[None, :])   # Hilbert matrix
        x_true = np.ones(n)
        b = H @ x_true
        x_hat = np.linalg.solve(H, b)
        kappa = np.linalg.cond(H)
        fwd = np.linalg.norm(x_hat - x_true) / np.linalg.norm(x_true)
        omega = (np.linalg.norm(b - H @ x_hat)
                 / (np.linalg.norm(H, 2) * np.linalg.norm(x_hat)))
        exact = np.linalg.norm(solve_exact(H, b) - x_true) / np.linalg.norm(x_true)
        print(f'{n:2d}  {kappa:8.1e}  {fwd:13.1e}  {-np.log10(fwd):11.1f}'
              f'  {16 - np.log10(kappa):16.1f}  {omega:14.1e}  {exact:17.1e}')
```

n	kappa	forward error	digits kept	16 - log10 kappa	backward error	exact solve error
4	1.6e+04	4.1e-14	13.4	11.8	0.0e+00	1.1e-13
6	1.5e+07	1.4e-10	9.8	8.8	1.3e-16	1.0e-10
8	1.5e+10	6.1e-08	7.2	5.8	5.7e-17	7.0e-08
10	1.6e+13	8.7e-05	4.1	2.8	1.0e-16	4.5e-05
12	1.8e+16	3.2e-01	0.5	-0.2	9.8e-17	1.4e-01

保持された桁数は、約 13 桁（$n = 4$）から 1 桁を大きく下回る水準（$n = 12$）まで落ち込む一方、後退誤差は約 10^{-16} 以下にとどまり続ける。すなわち、計算された各ベクトルは、格納された方程式と 16 桁目または 17 桁目しか違わない方程式を解いている。では、桁数が失われた責任はソルバにあるのだろうか。右端の列がその潔白を証明している。丸め誤差を一切犯さない格納済み方程式の厳密求解でも、**1** に対してすべての行で同じオーダーの誤差（約 10^{-7}、10^{-4}、10^{-1} がそれぞれ $n = 8$、10、12 で発生）が生じている。桁数はソルバが走る前、**b** を格納した時点ですでに失われていたのである。誤差が最も増幅される方向と完全には揃っていないため、経験則は約 1 桁（行によって約 0.5 桁から約 1.5 桁）だけ悲観的になっている。$n = 12$ では $\kappa u \approx 2$ となり、上界はもはや何も縛らない。末尾の数値は LAPACK のビルドによって多少変わるが、全体の傾向は変わらない。

35.8.4.4 正規方程式の条件性

最小二乗問題 $\min_{\mathbf{w}} \|\mathbf{A}\mathbf{w}-\mathbf{b}\|^2$（ただし $\mathbf{A}\in\mathbb{R}^{n\times d}$ は列フルランク）は、正規方程式 $\mathbf{A}^\top\mathbf{A}\mathbf{w}=\mathbf{A}^\top\mathbf{b}$ によって解かれる。これは代数的に同値でありながら、より悪条件な行列を経由する経路である（33.5.3.2 節）。

$$\kappa(\mathbf{A}^\top\mathbf{A}) = \kappa(\mathbf{A})^2. \tag{35.107}$$

これと同じ現象を 1 パス分散でもすでに見た。手軽な公式が問題の条件数を 2 乗してしまうのである。経験則によれば、正規方程式は約 $2\log_{10}\kappa$ 桁を失いうる。一方、QR 分解（$\mathbf{A}=\mathbf{Q}\mathbf{R}$、ただし $\mathbf{Q}$ は正規直交列を持ち $\mathbf{R}$ は三角行列）や特異値分解（SVD）による解法は $\mathbf{A}$ に直接作用するため、**残差 $\mathbf{b}-\mathbf{A}\mathbf{w}$ が小さいときには**失う桁数が約 $\log_{10}\kappa$ で済む。残差が大きい場合には、最小二乗問題自体がより敏感になる。その条件数はおよそ $\kappa+\kappa^2\tan\theta$ であり、ここで θ は $\mathbf{b}$ と $\mathbf{A}$ の列空間（column space）がなす角である。このとき QR 分解や SVD も $2\log_{10}\kappa$ 桁近くを失うが、問題自体が強制する以上に失うことは決してない。これに対し正規方程式は、残差の大小にかかわらず常にそれだけの桁数を失う。

本章の通し例は良条件すぎてこの差が見えない。その計画行列は $\mathbf{A}^\top\mathbf{A}=\mathrm{diag}(1,1000)$ を満たすため、$\kappa(\mathbf{A})=\sqrt{1000}\approx 32$ にすぎないからである。そこで以下のセルでは、ランダムな正規直交因子と列空間内の右辺（残差ゼロ）から、$\kappa(\mathbf{A})=10^3$ および 10^6 を持つより難しい計画行列を構成する。3 桁および 6 桁の差が現れるはずである。

```
rng = np.random.default_rng()
n, d = 100, 10
for kappa in [1e3, 1e6]:
    U, _ = np.linalg.qr(rng.normal(size=(n, d)))    # orthonormal columns
    V, _ = np.linalg.qr(rng.normal(size=(d, d)))
    sigma = np.logspace(0, -np.log10(kappa), d)     # kappa(A) as chosen
    A = U * sigma @ V.T                             # A = U diag(sigma) V^T
    w_true = rng.normal(size=d)
    b = A @ w_true                                  # zero residual
    w_ne = np.linalg.solve(A.T @ A, A.T @ b)        # normal equations
    w_ls = np.linalg.lstsq(A, b, rcond=None)[0]     # SVD-based solve
    digits = [-np.log10(np.linalg.norm(w - w_true) / np.linalg.norm(w_true))
              for w in (w_ne, w_ls)]
    print(f'kappa(A) = {kappa:.0e}: normal equations keep {digits[0]:4.1f} '
          f'digits, lstsq keeps {digits[1]:4.1f}')
```

```
kappa(A) = 1e+03: normal equations keep 10.3 digits, lstsq keeps 14.2
kappa(A) = 1e+06: normal equations keep  5.2 digits, lstsq keeps 12.0
```

$\kappa(\mathbf{A})=10^3$ では正規方程式が保持する桁数は約 $10\sim12$ 桁であるのに対し `lstsq` は約 14 桁であり、10^6 では約 $4\sim6$ 桁に対して約 $11\sim13$ 桁となる。正確な桁数は実行ごとに 1〜2 桁変動するが、両者の差は $\log_{10}\kappa$ の周りに分布し、大半がそこから 1 桁以内に収まる。`lstsq` や 33.5.3.1 節の擬似逆行列（pseudoinverse）が特異値分解（SVD）に基づいて計算され、$(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$ を用いない理由はここにある。

35.8.4.5 リッジ正則化と条件性

特徴量が強い多重共線性を持つか計画行列がランク落ちしているために $\kappa(\mathbf{A})$ 自体が大きい場合、いかなるアルゴリズムを選んでも助けにはならず、問題そのものを変えなければならない。答え

を変えずに問題を変える方法が 1 つある。**前処理行列**（preconditioner）は変数変換によって新しい問題をより良条件にし、その解を元の問題の解へと写し戻す。$\mathbf{A}$ の各列をそのノルムで割る操作はその一例であり（演習問題 4）、35.5 節の座標ごとのステップサイズもオプティマイザの内部で同じ働きをしている。

リッジ正則化は答えそのものも変化させる。これは $\frac{1}{2}\|\mathbf{A}\mathbf{w}-\mathbf{b}\|^2+\frac{\lambda}{2}\|\mathbf{w}\|^2$ を最小化する手法であり、その最小点（minimizer）は次の式で与えられる。

$$\mathbf{w}_\lambda=(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})^{-1}\mathbf{A}^\top\mathbf{b}, \tag{35.108}$$

これは 33.5.3.1 節で導いたリッジ解である。$\sigma_1\geq\cdots\geq\sigma_d\geq 0$ を行列 $\mathbf{A}\in\mathbb{R}^{n\times d}$ の特異値とし、ランクを超える添字では $\sigma_i=0$ とする。行列 $\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I}$ は $\mathbf{A}^\top\mathbf{A}$ と同じ固有ベクトルを持ち、固有値は $\sigma_i^2+\lambda\geq\lambda>0$ となる。したがって正定値であり、$\mathbf{A}$ がランク落ちしていても正則となって、条件数は次を満たす。

$$\kappa(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})=\frac{\sigma_1^2+\lambda}{\sigma_d^2+\lambda}. \tag{35.109}$$

$\sigma_1>\sigma_d$ である限り、この比は λ に関して狭義単調減少し（導関数は $(\sigma_d^2-\sigma_1^2)/(\sigma_d^2+\lambda)^2<0$）、$1$ へと $\lambda\to\infty$ で収束する。λ を加えることはスペクトルの床を押し上げる働きをする。$\lambda\ll\sigma_1^2$ の間は天井がほとんど動かないため、κ はほぼ $1/\lambda$ に比例して（$\lambda\gg\sigma_d^2$ となった領域で）減少する。二次オプティマイザのダンピング項や、ガウス過程回帰（第 40 章（英語版））でカーネル行列に加えるジッタは、別の名前で呼ばれているこの $\lambda\mathbf{I}$ にほかならない。

その代償は最小点が変化することである。現在の記法で、$\mathbf{v}_i$ を行列 $\mathbf{A}$ の右特異ベクトル、$\mathbf{w}_0=\mathbf{A}^+\mathbf{b}$ を最小ノルム最小二乗解とすると、33.5.3.1 節のフィルタ係数は次の形をとる。

$$\mathbf{w}_\lambda=\sum_{i=1}^{d}\frac{\sigma_i^2}{\sigma_i^2+\lambda}\,(\mathbf{v}_i^\top\mathbf{w}_0)\,\mathbf{v}_i. \tag{35.110}$$

リッジ正則化が最も強く減衰させるのは小さな σ_i を持つ方向であり、それはまさに正則化なしの求解がデータ誤差を $1/\sigma_i$ 倍に増幅してしまう方向である。Figure 35.33 に等位集合への効果を示す。各パネルはそれぞれの最小点を中心に描かれているため、最小点が原点へ向かって移動する様子自体は描画されていない。

この同じ κ が、リッジ目的関数に対する勾配降下法のステップ数をも支配する。最良の固定ステップサイズのもとで、条件数 $\kappa>1$ の二次関数に対する勾配降下法は 1 ステップごとに誤差を $\rho=(\kappa-1)/(\kappa+1)$ 倍に縮小させる（35.1 節）。誤差が $-\mathbf{w}^\star$ である原点から出発すると、相対誤差は $\|\mathbf{w}_k-\mathbf{w}^\star\|/\|\mathbf{w}^\star\|\leq\rho^k$（$k$ ステップ後）となるため、許容誤差 $0<\mathrm{tol}<1$ に達するまでのステップ数は次で抑えられる。

$$k\leq\left\lceil\frac{\log(1/\mathrm{tol})}{\log\frac{\kappa+1}{\kappa-1}}\right\rceil\leq\left\lceil\frac{\kappa}{2}\,\log\frac{1}{\mathrm{tol}}\right\rceil\quad\text{steps.} \tag{35.111}$$

第 2 の不等式は $\log\frac{\kappa+1}{\kappa-1}=2\,\mathrm{artanh}(1/\kappa)\geq 2/\kappa$ から従う。6 桁の精度を得るには $\frac{1}{2}\log 10^6=6.9$ ステップ（κ の 1 単位あたり）が上界となるが、3 次元以上での実行ではそれより早く収束しうる（演習問題 4）。

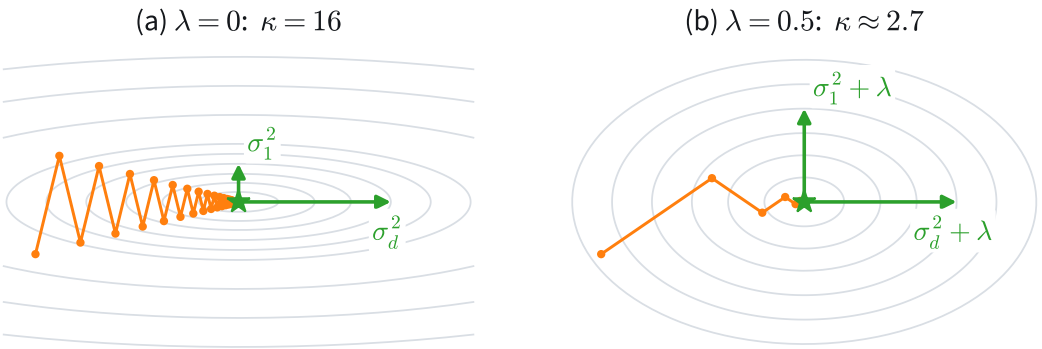


Figure 35.33: リッジ項 $\frac{\lambda}{2}\|\mathbf{w}\|^2$ を加える前後の最小二乗目的関数の等位集合。両パネルが見やすいよう $\kappa = 16$ で描いている（通し例の $\kappa = 1000$ では線のように潰れてしまう）。ペナルティは $\mathbf{A}^\top\mathbf{A}$ のすべての固有値を λ だけ押し上げる。$1/\sqrt{\sigma_i^2 + \lambda}$ に比例する半軸は曲率の小さい方向で長く、細長い谷が丸みを帯びたボウルへと変わる。オレンジ色：各パネルの最良の固定ステップサイズを用いた同じ初期点からの勾配降下法の反復点、緑色の矢印：曲率を添えた主軸。

与えられた λ によって何ステップ節約でき、そのコストはどれだけになるだろうか。通し例で両方を測定してみよう。目的関数 $\frac{1}{2}(w_1^2 + 1000w_2^2) - w_1 - 1000w_2$ は、$\mathbf{A}^\top\mathbf{A} = \mathrm{diag}(1, 1000)$ および $\mathbf{A}^\top\mathbf{b} = (1, 1000)$ を持つ最小二乗問題である。その最小点は $(1, 1)$ であるため、リッジ解の最小点には 2 つのフィルタ係数がそのまま現れる。関数 `gd_iterations` は最良の固定ステップサイズで勾配降下法を実行し、ステップ数とリッジ解の最小点を返す。

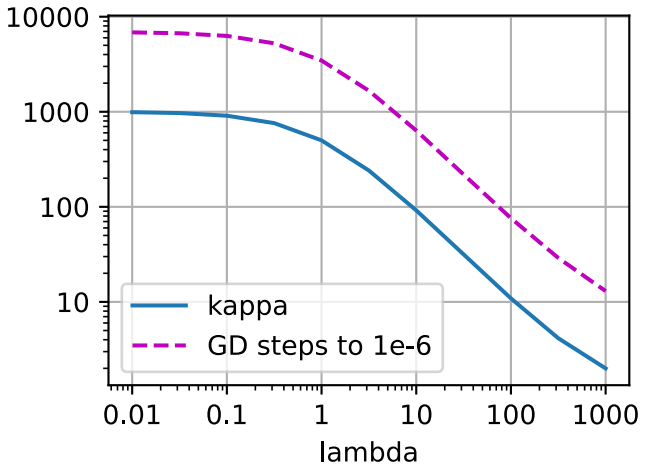
```
def gd_iterations(AtA, Atb, lam, tol=1e-6):
    """GD steps at step 2/(L+mu) to reach relative error tol on ridge."""
    M = AtA + lam * np.eye(len(Atb))
    w_star = np.linalg.solve(M, Atb)
    mu, L = np.linalg.eigvalsh(M)[[0, -1]]
    eta, w, k = 2 / (L + mu), np.zeros(len(Atb)), 0
    while np.linalg.norm(w - w_star) > tol * np.linalg.norm(w_star):
        w, k = w - eta * (M @ w - Atb), k + 1
    return k, w_star

AtA, Atb = np.diag([1.0, 1000.0]), np.array([1.0, 1000.0])
print('lambda   kappa  steps  steps/kappa  ridge minimizer')
for lam in [0, 1, 10, 100, 1000]:
    kappa = (1000 + lam) / (1 + lam)
    k, w_star = gd_iterations(AtA, Atb, lam)
    print(f'{lam:6d} {kappa:7.1f} {k:6d} {k / kappa:12.2f}  {w_star.round(4)}')
```

```
lambda   kappa  steps  steps/kappa  ridge minimizer
     0  1000.0   6908         6.91  [1. 1.]
     1   500.5   3458         6.91  [0.5   0.999]
    10    91.8    635         6.92  [0.0909 0.9901]
   100    10.9     76         6.98  [0.0099 0.9091]
  1000     2.0     13         6.51  [0.001 0.5  ]
```

条件数は 1000 から 2 へと（λ が 0 から 1000 へ動くにつれて）低下し、それに伴ってステップ数も 6,908 から 13 へと減少する。ステップ数と κ の比は 6.9 付近に保たれる（小さな κ では近似

$2/\kappa$ が緩くなり、整数ステップへの切り上げも効いてくる）。右端の列は節約されたステップ数の代償を示している。$\lambda = 1$ では、$\sigma^2 = 1$ の遅い方向が半分の 0.5 に縮小する一方、$\sigma^2 = 1000$ の硬い方向は 0.1% しか動かない。$\lambda = 1000$ では硬い方向も半分に縮小する。λ をスイープすると、κ とステップ数の関係を同時に確認できる。

```
lams = np.logspace(-2, 3, 11)
kappas = (1000 + lams) / (1 + lams)
steps = [gd_iterations(AtA, Atb, lam)[0] for lam in lams]
d2l.plot(lams, [kappas, steps], 'lambda', None,
         legend=['kappa', 'GD steps to 1e-6'], xscale='log', yscale='log')
```



両対数軸上では 2 本の曲線は 6.9 倍の間隔を保って平行に並び、傾きは -1 に近い（λ が固有値 1 と 1000 の十分内側にある間）。このようにリッジ正則化は連立一次方程式の求解と勾配降下法の双方の条件性を改善し、いずれに対しても解を縮小させることでその代償を支払う。35.7.3.2 節で示した通り、ペナルティ $\frac{\lambda}{2}\|\mathbf{w}\|^2$ は $\|\mathbf{w}\|$ に対する制約に対応しているため、統計的な理由から選ばれた λ が計算の条件性をも同時に定めることになる。

ここまでの道具立てはいずれも、100 層を通過する勾配のように反復の多数ステップにわたって増大する誤差は扱っていない。それについては 4.4 節で取り上げる。

35.8.5 まとめ

浮動小数点形式は、約 p ビットの相対精度と有限の指数部範囲で数を格納する。各丸めは高々 $u = 2^{-p-1}$ の相対誤差を生じる一方、オーバーフロー、アンダーフロー、微小な更新の消失は相対モデルには現れない絶対スケールの現象である。ソフトマックスと交差エントロピーは形式が保持できない値を経由するために破綻し、1 パス分散はほぼ等しい大きな数同士を減算するために破綻するが、いずれの真の答えも極端な値ではない。最大値シフト、ロジットから計算する交差エントロピー、Welford の漸化式はこうした迂回を避ける。相対条件数 $\kappa_f = |xf'/f|$ は不適切なアルゴリズムを難しい問題から切り分け、この値が小さいのに計算結果が悪ければアルゴリズムに原因がある。連立一次方程式では、$\kappa(\mathbf{A})$ が前進誤差を後退誤差（残差によって保証される）の κ 倍で抑えるため、約 $\log_{10}\kappa$ 桁が失われ、正規方程式を経由するとその 2 倍の桁数が失われる。リッジ正則化はすべての特異値の 2 乗に λ を加えることで求解と勾配降下法の双方に対して κ を引き下げるが、その代償として解はフィルタ係数 $\sigma_i^2/(\sigma_i^2 + \lambda)$ 倍に縮小する。

35.8.6 演習問題

1. **損失スケールの許容範囲**。損失スケーリングは損失に定数 S を掛け（これによりすべての勾配が S 倍になる）、勾配を fp32 に変換したあとで S で割る。fp16 では、最小の正規化数は $2^{-14} \approx 6.1 \times 10^{-5}$、最小の非正規化数は $2^{-24} \approx 6.0 \times 10^{-8}$（$2^{-25}$ 未満の値はゼロに丸められる）、最大の有限数は 65504 である。あるネットワークの厳密な非ゼロ勾配成分の大きさが 10^{-9} から 10^{-1} の間にあり、逆伝播ではスケールされた勾配を fp16 で計算するとする。
   1. スケールされたすべての勾配が fp16 の正規化数となるような 2 のべき乗 S をすべて求めよ。
   2. より深いネットワークの勾配の大きさが 10^{-12} から 10^{-1} の間にあるとする。いかなるスケールを用いてもそのすべてを正規化数に保つことはできないことを示せ。オーバーフローを避ける最大の 2 のべき乗 S をとるとき、大きさ 10^{-12} の勾配はスケール後に有効ビットを何ビット保持し、その相対丸め誤差はいくらになるか。
   3. 動的損失スケーリング（5.5.3.3 節）はオーバーフローを避ける最大のスケールを選び、それによって最小側の勾配を犠牲にする。そちら側を犠牲にする方が望ましいのはなぜか。
   4. 勾配を bf16 で計算する場合、最初の 2 つの小問の答えはどう変わるか。

2. [code] **問題とアルゴリズムの切り分け**。以下の各式を fp32 で評価する。それぞれについて、指定された領域における**関数**の相対条件数 Equation 35.102 と、**書かれた通りの式**の桁落ち係数 Equation 35.103 を計算せよ。問題とアルゴリズムのどちらに原因があるかを判定し、誤差が問題自体の感度に一致するよう式を書き換えよ。さらに両方の形を fp64 の結果と照合せよ。
   1. $ax^2 + bx + c = 0$（ただし $b > 0$ かつ $b^2 \gg 4|ac|$）の絶対値が小さい方の根を c の関数とみなしたもの（たとえば $a = c = 1$、$b = 10^4$）。（ヒント：共役な式を掛けよ。）
   2. 関数 $\log(1 - e^{-x})$（ただし $x > 0$；$\log(1 - q)$ を $\log q = -x$ から計算する式）を $x \in \{10^{-6}, 10^{-3}, 1, 20\}$ で評価したもの。小さい x で正確な定式化と大きい x で正確な定式化をそれぞれ 1 つずつ見つけ、切り替え点を選べ。
   3. ロジット t（ラベル 1）の二値交差エントロピー $\log(1 + e^{-t})$ を $t \in \{-800, -50, 0, 50\}$ で評価したもの。これらのうち、書かれた通りの式がオーバーフローするのはどの値か、また 0 を返すのはどの値か。$\max(-t, 0) + \operatorname{log1p}(e^{-|t|})$ がこの関数に等しく、決してオーバーフローせず、桁も失わないことを示せ。これは log-sum-exp のサンドイッチ不等式のどの部分にあたるか。
   4. $\sqrt{x+1} - \sqrt{x}$（$x = 10^8$ において）。

   小問 2 は Martin Mächler「Accurately Computing log(1 −exp(−|a|))」第 2 節を改変。

3. [code] **浮動小数点における正規方程式**。正規方程式は条件数を 2 乗する（Equation 35.107）。
   1. $\tau > 0$ に対し、$(m+1) \times m$ 行列 $\mathbf{A}$（Läuchli (1961)）は第 1 行がすべて 1 であり、その下に $\tau\mathbf{I}$ を並べた行列である。$\mathbf{A}^\top\mathbf{A}$、$\mathbf{A}$ の特異値、および $\kappa(\mathbf{A})$ を求めよ。$\tau = 10^{-8}$ のとき、行列 $\mathbf{A}^\top\mathbf{A}$ が fp64 で特異行列へ丸められることを示せ。35.8.4 節の経験則は、この $\mathbf{A}$ を用いた QR 分解ベースの最小二乗求解に対し、$\mathbf{b}$ が $\mathbf{A}$ の値域に属するとき正しい桁が何桁残ると予測するか。残差が大きいとき、その予測はどう変わるか。
   2. `#numerical-stability-conditioning-normal-equations` セルを変更し、$\kappa(\mathbf{A}) = 10^k$ を $k = 2, \dots, 10$ にわたってスイープせよ。正規方程式と `lstsq` の正しい桁数を k に対して、経験則の 2 直線 $16 - 2k$ および $16 - k$ と重ねてプロットせよ。経験則が正規方程式に正しい桁が 1 桁も残らないと予測するのはどの k か、またそこでは実際に何が観測されるか。

4. [code] **前処理としての特徴量スケーリング**。異なる単位で測られた特徴量は、計画行列の

各列に大きく異なるノルムを与える。$\mathbf{X}$ を標準正規分布に従う成分を持つ 200×5 行列とし、$\mathbf{A} = \mathbf{X}\,\mathrm{diag}(1, 3, 10, 30, 100)$ とおき、$\mathbf{b} = \mathbf{A}\mathbf{w}^\star$（$\mathbf{w}^\star$ はランダム）とおく。

1. `#numerical-stability-conditioning-ridge` セルの `gd_iterations`（$\mathbf{A}^\top\mathbf{A}$ と $\mathbf{A}^\top\mathbf{b}$ を引数にとる）をこの計画行列に $\lambda = 0$ で適用せよ。$\kappa(\mathbf{A}^\top\mathbf{A})$ を計算し、ステップ数を上界 Equation 35.111 と比較せよ。ステップ数が上界を下回る理由を、初期点の誤差が $\mathbf{A}^\top\mathbf{A}$ の両端の固有ベクトル方向に持つ成分から説明せよ。
2. $\mathbf{A}$ のすべての列をそのノルムで割り、同じ測定を繰り返せ。再スケーリング後の問題の最小点は元の $\mathbf{w}^\star$ へどのように写し戻されるか。
3. 演習問題 1（35.2 節）の小問 2 では、新しい座標 $\mathbf{u}$（$\mathbf{w} = \mathbf{T}\mathbf{u}$）における勾配降下法を、正定値行列 $\mathbf{P}$ で前処理された元の問題に対する勾配降下法（更新式 $\mathbf{w} \leftarrow \mathbf{w} - \eta\,\mathbf{P}^{-1}\nabla f(\mathbf{w})$）として表した。列スケーリングに対応するのはどの $\mathbf{T}$ とどの $\mathbf{P}$ か。また、$\mathbf{P}$ は $\mathbf{A}^\top\mathbf{A}$ の対角成分とどのように関係するか。

5. [code] **確率的丸め**。●確率的丸め (Gupta et al. 2015) は、実数 x（隣り合う表現可能な値 $x_- < x_+$ の間にある）を、x_+ へ確率 $q = (x - x_-)/(x_+ - x_-)$ で丸め、それ以外の場合は x_- へ丸める。

   1. 丸め後の値が期待値 x と分散 $q(1-q)\Delta^2 \le \Delta^2/4$（ただし $\Delta = x_+ - x_-$）を持つことを示せ。
   2. bf16 の重みが $w = 1$ から出発し、$K = 1000$ 回の -10^{-4} の更新を受け、毎回その直後に確率的丸めを行うとする。重みは $[0.5, 1]$ の範囲にとどまり、そこでの bf16 の間隔は $\Delta = 2^{-8}$ である。異なるステップの丸め誤差が互いに無相関であることを示し、最終的な重みの平均と標準偏差を予測せよ。最近傍丸めを用いた場合、最終的な重みはいくらになるか。
   3. 正の fp32 の数に対し、そのビットパターンに $\{0, \dots, 2^{16} - 1\}$ から一様ランダムに選んだ整数を加えてから下位 16 ビットをクリアすると、bf16 への確率的丸めになる。その理由を説明せよ。35.8.1 節の `to_bf16` 関数の横にこの丸めを実装し、1000 回の独立な実行によって前小問の予測を検証せよ。さらに、同じ更新を受ける fp32 のマスターコピーと比較せよ。

6. **厳密な減算**。●減算そのものが厳密であっても、不正確な差が返ることがある。

   1. 上界 Equation 35.103 が、適切な δ_a と δ_b（$|\delta_a|, |\delta_b| \le u$）に対して達成されることを示せ。
   2. Sterbenz の補題 (Sterbenz 1974) を証明せよ。すなわち、a と b が形式 Equation 35.95 の浮動小数点数であり $b/2 \le a \le 2b$ を満たすならば、$a - b$ も浮動小数点数となり、したがって計算された差は厳密である。ただし、その形式は非正規化数を含むと仮定せよ。
   3. `#numerical-stability-conditioning-log1p` セルでは、fp32 の差 $a - b$ が真の値 3.0×10^{-7} から約 20% ずれている。Sterbenz の補題と出力された ulp 数を用いてこの減算が厳密であったことを確認し、格納された a と b の相対誤差を計算したうえで、最初の小問の上界がその誤差を覆っていることを確かめよ。

7. [code] **逐次モーメントの統合**。●データ並列訓練では各シャード上で統計量を計算し、それらを結合する。シャード A が k_A 個の値を持ち、その平均を m_A、中心化二乗和を M_A とする。同様にシャード B が k_B 個の値を持ち、m_B と M_B を持つとする。

   1. 2 つのシャードの和集合における平均 m と中心化二乗和 M の厳密な公式を、k_A、m_A、M_A、k_B、m_B、M_B を用いて導出せよ。
   2. $k_B = 1$ の場合が Welford の漸化式 Equation 35.104 に一致することを示せ。
   3. 10^5 個のサンプル（`#numerical-stability-conditioning-welford` セルのもの）を、ほぼ等サイズの 2^j 個のシャードへ分割せよ（`np.array_split` を用い、$j \in \{1, 4, 8\}$ と

する）。各シャードの統計量を Welford の漸化式で計算し、二分木状にペアワイズ統合して、得られた分散を 2 パスの基準値と比較せよ。続いて、各シャードの和 $\sum_i x_i$ と $\sum_i x_i^2$ を足し合わせて 1 パス公式を適用する方法でもう一度統合し、両者の違いを説明せよ。

8. [code] **総和の誤差増大。**●$S_n = \sum_{i=1}^{n} x_i$ とし、$\hat{S}_n$ をその計算値とする。順次総和法は最悪ケースの上界 $|\hat{S}_n - S_n| \le (n-1)\,u \sum_i |x_i| + O(u^2)$ を持つ。

   1. $n = 2^k$ に対し、再帰的なペアワイズ総和法は項を半分ずつに分け、それぞれの半分を再帰的に足し合わせてから 2 つの結果を加算する。これが $|\hat{S}_n - S_n| \le u \log_2(n) \sum_i |x_i| + O(u^2)$ を満たすことを証明せよ。
   2. $n \in \{2^4, 2^8, 2^{12}, 2^{16}, 2^{20}\}$ 個の $[0, 1)$ 上の一様乱数を fp32 で格納し、順次総和法（`np.cumsum(x)[-1]`）、`np.sum`（ブロック化されたペアワイズ総和法）、および Kahan の補償付き総和法 (Kahan 1965) の相対誤差を float64 の基準値に対して測定せよ。Kahan の手法は補正値 c を保持し、各加算 $t = s + y$ のあとに $c = (t - s) - y$ とおいて次の項から c を引く。3 つの誤差を 2 つの上界とともに n に対してプロットせよ。
   3. 順次総和法の実測誤差は、最悪ケースの上界と比べて n に対しどのように増大するか、またそれはなぜか。本文における 10^7 個の 0.1 の総和が最悪ケースにはるかに近いのはなぜか。
   4. 大きな n において、高精度な 2 つの手法の誤差を律速している要因は何か。

   Steven G. Johnson「MIT 18.335 Introduction to Numerical Methods, Problem Set 1」問題 4 を改変。

9. [code] **混合精度反復改良法。**●●混合精度訓練は演算の大半を低精度で実行し、微小な補正を蓄積する量を高精度に保つ。反復改良法 (Moler 1967; Carson and Higham 2018) は同じ役割分担を $\mathbf{A}\mathbf{x} = \mathbf{b}$ に適用する。すなわち $O(n^3)$ の行列分解を低精度で実行し、$O(n^2)$ の残差計算だけを高精度で実行する。fp32 の解 $\mathbf{x}_0$ から出発し、次を繰り返す。

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k, \qquad (\mathbf{A} + \Delta\mathbf{A}_k)\,\mathbf{d}_k = \mathbf{r}_k, \qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k,$$

   ここで残差と更新は fp64 で計算し、補正ベクトル $\mathbf{d}_k$ は fp32 の求解から得る。

   1. fp32 の求解を後退安定としてモデル化し、$\|\Delta\mathbf{A}_k\| \le c\,u_{32}\,\|\mathbf{A}\|$（ただし $u_{32} = 2^{-24}$、c は適度な定数）とおき、fp64 の丸め誤差は無視する。誤差 $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}$ が $\mathbf{e}_{k+1} = (\mathbf{A} + \Delta\mathbf{A}_k)^{-1}\Delta\mathbf{A}_k\,\mathbf{e}_k$ を満たすことを示せ。さらに縮小率を $c\,\kappa(\mathbf{A})\,u_{32}$ を用いて導出し、それが 1 未満となる条件を述べよ。
   2. `#numerical-stability-conditioning-hilbert` セルのヒルベルト行列に対し、$n = 3, \dots, 10$ でこの反復を実装せよ。補正の計算には、`np.linalg.solve` を $\mathbf{H}$ と $\mathbf{r}_k$ の fp32 コピーに適用すること。10 反復にわたって前進誤差を記録し、最終値を fp64 による求解の誤差と比較せよ。反復改良法が fp64 の求解精度に到達するのはどの n か。収束はするものの 10 反復以内ではその精度に届かないほど遅いのはどの n か。誤差が増大するのはどの n か。
   3. 各 n について 1 反復あたりの実測縮小率を推定し、それを $\kappa(\mathbf{H})\,u_{32}$ および $\mathbf{I} - \hat{\mathbf{H}}^{-1}\mathbf{H}$ のスペクトル半径（ただし $\hat{\mathbf{H}}$ は $\mathbf{H}$ の fp32 コピーであり、逆行列の適用は fp64 で行う）と比較せよ。この実験において $\Delta\mathbf{A}_k$ を支配している摂動は何か、また最初の小問の上界はどれほど悲観的か。

35.9 関連文献

本節では本章の範囲を超えた話題を紹介するが、そのすべてが同じ緊急度を持つわけではない。そのうち 5 つはニューラルネットワークの訓練に直接関わるため、それぞれの見出しの先頭に配置している。すなわち、安定性の縁（edge of stability）、反復点の平均化（iterate averaging）、補間（interpolation）、フィッシャー行列、および鞍点法（saddle-point method）である。残りの話題は古典的な最適化理論および数値解析からのものであり、最後に一般的な教科書、講義、サーベイをまとめる。なお、訓練の実践的な側面については 9.11 節で概観している。本書の他の箇所は本節の内容に一切依存しないため、関心のある項目を選んで自由に拾い読みするとよい。

35.9.1 勾配法

35.9.1.1 安定性の縁における自己安定化

ニューラルネットワークをフルバッチ勾配降下法で訓練すると、驚くべき現象が起こる。損失の鋭さ（sharpness、ヘッセ行列の固有値のうち最大のもの）が安定限界 $2/\eta$ まで上昇したのち、発散するのではなくちょうどその付近にとどまり続けるのである (Cohen et al. 2021)。35.1.2.4 節の二次モデルは、鋭さが $2/\eta$ を超えた瞬間に最大固有モードが増大する理由を示すことでこの振る舞いの前半を説明するが、純粋な二次モデルでは際限のない発散が予測されるため、何が鋭さを再び引き下げるのかを説明できない。Damian et al. (2023) は、テイラー展開にもう 1 つだけ項、すなわち最大固有ベクトル方向の 3 次の項を残すことでこの謎を解明した。その方向に沿って振動が大きくなると、この 3 次の結合が反復点を鋭さそのものの勾配と逆向きに押し出す。それによって鋭さが再び $2/\eta$ 未満へと下がり、次のサイクルが始まるまで振動が減衰する。この 3 次の項がなければ、振動は歯止めなく増大し続けるだろう。

35.9.1.2 最適多項式と共役勾配法

二次関数 $f = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$（固有値が $[\mu, L]$ に含まれ、最小点が $\mathbf{0}$ であるもの）の最小化を考えよう。k ステップ目の更新がそれまでに得られた勾配の張る空間（span）にとどまる任意の一次手法は、$\mathbf{x}_k = p_k(\mathbf{A})\mathbf{x}_0$ という形の反復点を生成する。ここで、p_k は次数が高々 k で $p_k(0) = 1$ を満たす多項式である。このように見れば、二次関数に対する高速な勾配法を設計することは、区間 $[\mu, L]$ 全体にわたって可能な限り小さくなる多項式を選ぶことと等価である。区間全体での最大値を最小にする多項式はスケーリングされたチェビシェフ多項式であり、その最大値は $2\rho^k/(1+\rho^{2k}) \approx 2\rho^k$ となる。ここで縮小率 $\rho = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ は、ヘビーボール法で現れたものと完全に一致する（35.1.3 節）。次元 d が k に比べて大きいとき、固有値は区間全体に広がり得るため、この族に属するいかなる手法もその最悪ケースの収束率（convergence rate）を上回ることはできない。共役勾配法（conjugate gradient method）は実用上さらに優れた振る舞いを示す。各ステップにおいて、f を**実際の**行列 $\mathbf{A}$ と初期点 $\mathbf{x}_0$ に対して最小化する多項式を、μ や L を一切知ることなく自動的に見つけ出し、厳密演算のもとでは高々 d ステップで終了するのである。Nocedal and Wright (2006)（第 5 章）は、標準的な共役勾配法とその前処理付き版の両方を導出している。

35.9.1.3 半正定値計画による収束率の保証

タイトな収束率を手計算で証明するのは、非常に繊細な作業として知られている。35.3.3.3 節における手計算の証明ではステップごとの縮小率として $1-\mu/L$ が得られたが、同項で引用した厳密な縮小率は $(1-\mu/L)^2$ であり、必要なステップ数は半分になる。このよりタイトな評価は、**性能推定**（performance estimation）と呼ばれる技法から得られる。Drori and Teboulle (2014) は、巧みな代数的変形を手作業で探す代わりに、すべての L 滑らかな凸関数にわたるアルゴリズムの最悪ケース性能を、反復点と勾配がなすグラム行列（すべてのペアの内積を並べた表）上の最適化問題として定式化した。この問題を上から評価すると小さな半正定値計画問題（semidefinite

program、SDP）に帰着し、その双対解から人間が検証可能な証明が文字通り得られる。Taylor et al. (2018) はまさにこの方法で二乗の縮小率を導出した。また制御理論の視点に立つ Lessard et al. (2016) は、勾配を μ と L のみで制約された線形力学系内部の未知の非線形性とみなし、ステップ数によらずサイズが一定の半正定値計画問題によって収束率を保証している。注目すべきことに、同じ枠組みは反例の発見にもつながり、ヘビーボール法が収束しない関数（35.1.3 節）もこうして発見された。

35.9.2 確率的手法と適応的手法

35.9.2.1 定数ステップによる平均化

35.4.1 節で見たように、定数ステップの SGD はステップサイズ η に比例するノイズフロアに落ち着く。時間とともに η を縮小すればこのノイズフロアは解消されるが、終盤の進行が遅くなるうえ、スケジュールが調整に対して敏感になってしまう。反復点を平均化すれば、より明快な解決策が得られる。個々の反復点は最小点の周囲で跳ね回り続けるものの、それらの移動平均をとることで平均ゼロのノイズが相殺されるからである。Polyak and Juditsky (1992) は、$1/k$ より緩やかに減衰するステップサイズで得られた反復点を平均化すると、漸近的に最適な共分散が達成されることを示した。最小二乗法に対しては、Bach and Moulines (2013) がさらに踏み込み、**定数**のステップサイズですでに十分であることを示している。すなわち、平均化された反復点は $1/n$ のオーダーの誤差に n 個のデータ例で到達し、これには強凸性すら必要ない。

35.9.2.2 補間

古典的な確率的最適化では、最適点においても個々の訓練例がパラメータを互いに相反する方向へ引っ張り合うため、避けられないノイズフロアが生じると仮定する。これに対し、現代の過剰パラメータ化されたニューラルネットワークは、しばしば大きく異なるレジームで動作する。モデルがすべての訓練例に厳密に適合するとき、最小点では各訓練例の勾配がすべてゼロになる。するとノイズフロアは消失し、強凸な損失のもとでは、定数ステップの SGD がいかなる分散低減（variance reduction）も用いずに線形収束する（35.4.4 節）。Ma et al. (2018) は最小二乗法についてこの補間レジームを解析し、それ以上ミニバッチを大きくしても訓練が高速化しなくなる臨界バッチサイズ（critical batch size）を特定した。興味深いことに、このしきい値を支配しているのは勾配ノイズではなく、訓練例ごとの曲率とフルバッチの曲率の比である。また、勾配の分散が全勾配（full gradient）のノルムの二乗に比例して縮小するという関連条件である**強増大条件**（strong growth condition）のもとでは、SGD を加速してモメンタムと同等の収束率を達成することさえ可能になる (Vaswani et al. 2019)。

35.9.2.3 フィッシャー行列とガウス・ニュートン行列

K-FAC（35.5.5 節）などの二次近似は、フィッシャー情報行列 $\mathbf{F} = \mathbb{E}[\nabla \log p \nabla \log p^\top]$ の近似を目的とする。ここで期待値は、モデル自身の予測分布（predictive distribution）からラベルを抽出してとる (Amari 1998; Martens and Grosse 2015)。ソフトマックス交差エントロピーや二乗誤差のような標準的な損失に対しては、$\mathbf{F}$ はガウス・ニュートン行列 $\mathbf{J}^\top \mathbf{H}_{\text{out}} \mathbf{J}$ と一致する（$\mathbf{H}_{\text{out}}$ はネットワークの出力に関する損失のヘッセ行列である）。より一般に、この恒等式はネットワークの出力が指数分布族（exponential family）の尤度の自然パラメータ（natural parameter）を表すときには常に成り立つ（36.4 節）。実務上よくある落とし穴は、モデルからのサンプリングの代わりにデータセットの正解ラベルにわたって同じ外積を平均した**経験**フィッシャー行列（empirical Fisher）で代用してしまうことである。これは計算が安価であり、コードベース上でも単に「Fisher」と表記されることが多いが、一般には真のフィッシャー情報行列にもガウス・ニュートン行列にも一致しない (Martens 2020)。

35.9.2.4 ミラー降下法

AdaGrad を計量の観点から見ると（35.5.1 節）、ステップ長は二次のユークリッドノルムで測られる。この幾何構造は、あらゆる方向が同等に見える $\mathbb{R}^d$ のような制約なし空間には適しているが、各座標が非負かつ総和が 1 でなければならず乗法的な更新の方がはるかに自然である確率単体（probability simplex）の上では扱いにくい。ミラー降下法（mirror descent）は、勾配ステップの近接形式における二乗距離 $\frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$ をブレグマンダイバージェンス（Bregman divergence）$D_\phi(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x}) - \phi(\mathbf{y}) - \nabla\phi(\mathbf{y})^\top(\mathbf{x}-\mathbf{y})$ に置き換えることで、勾配降下法の幾何構造を適応させる (Nemirovski and Yudin 1983; Beck and Teboulle 2003)。ϕ として負のエントロピーを選ぶと、更新は指数勾配則（exponentiated gradient rule）$x_i \leftarrow x_i e^{-\eta g_i}/\sum_j x_j e^{-\eta g_j}$ となる。勾配が座標ごとに有界であるとき、この選択によって 35.5.1 節で定義したリグレットは、d 次元単体上において射影勾配降下法（projected gradient descent）の $\sqrt{d}$ スケーリングからはるかに緩やかな $\sqrt{\log d}$ へと改善される。

35.9.3 制約と双対性

35.9.3.1 鞍点法

ラグランジュ双対性（35.7.2 節）で登場した鞍点は、GAN の訓練（16.1 節）をはじめとする 2 人ゲームの均衡としても直接現れる。しかし残念ながら、一方のプレイヤーには勾配降下法を、もう一方のプレイヤーには勾配上昇法を同時に適用する素朴な戦略はうまく機能しない。単純な双線形ゲーム xy においてさえ、任意の正のステップサイズに対して鞍点から外側へと渦を巻いて発散してしまうのである（演習問題 5）。外挿勾配法（extragradient method）は、一歩先を覗き見ることでこの渦を断ち切る。すなわち、試行ステップを進め、その先読み点で勾配を評価し、得られた勾配を元の反復点から適用する。リプシッツ勾配を持つ凸凹（convex–concave）問題では、その平均化された反復点は $O(1/k)$ の速さで収束する (Nemirovski 2004)。また楽観的手法（optimistic method）は、直前のステップの勾配を試行勾配の安価な代用として再利用することで、半分の勾配コストで同様の安定化効果を達成しており、Daskalakis et al. (2018) はこの発想を GAN の訓練の安定化に応用して成果を上げている。

35.9.3.2 問題クラスと内点法

35.7.5 節で手計算により解いた双対問題は二次計画問題（quadratic program）であり、これは線形計画問題、二次計画問題、二次錐計画問題（second-order cone program）、半正定値計画問題という凸最適化クラスの標準的な入れ子階層の中に位置づけられる。現代の内点法（interior-point method）は、これらすべてのクラスを多項式時間で高い精度まで解くことができる。内点法は不等式 $g_i(\mathbf{x}) \le 0$ を直接扱う代わりに、滑らかな対数バリア（logarithmic barrier）$-\frac{1}{s}\sum_i \log(-g_i(\mathbf{x}))$ を目的関数に加え、s を大きくしながら制約なし最小点を追跡することで、境界へと向かう**中心パス**（central path）を描き出す。このパス上の各点において、量 $-1/(s\,g_i(\mathbf{x}))$ は自動的に双対実行可能な乗数をなし、双対ギャップ（duality gap）m/s（m は制約の個数）を保証する。これにより、35.7 節で手作業により構成した最適性の証明書が自動的に得られる (Boyd and Vandenberghe 2004; Nesterov 2018)。

35.9.3.3 作用素分割と ADMM

35.6.4.1 節の近接勾配ステップは、滑らかな損失と単一の単純な非滑らかペナルティの和を扱う。これに対し、構造を持つ多くの問題は合成形式 $\min_{\mathbf{x}} f(\mathbf{x}) + h(\mathbf{A}\mathbf{x})$ をとる。ここで両方の項は凸であり、ともに非滑らかであってもよく、そのフェンシェル双対（Fenchel dual）は $\max_{\mathbf{y}} -f^*(-\mathbf{A}^\top\mathbf{y}) - h^*(\mathbf{y})$ となる (Rockafellar 1970)。2 つの項を切り離して個別に扱えるようにするため、補助変数 $\mathbf{z} = \mathbf{A}\mathbf{x}$ を導入し、二次ペナルティ $\frac{\tau}{2}\|\mathbf{A}\mathbf{x}-\mathbf{z}\|^2$ をラグランジアンに加

える。まず $\mathbf{x}$ について最小化し、次に $\mathbf{z}$ について最小化し（これは h に対する近接ステップに帰着する）、最後に $\mathbf{y}$ について双対上昇ステップをとる操作を交互に繰り返すと、交互方向乗数法（alternating direction method of multipliers、ADMM）が得られ、これは任意の $\tau > 0$ に対して収束する。Boyd et al. (2011) はこの分割によって大規模な最適化問題を複数の計算機へ自然に分散できることを示しており、Parikh and Boyd (2014) は代表的な正則化項や制約に対する閉形式の近接作用素をまとめた便利なカタログを提供している。

35.9.3.4 フランク・ウルフ法

射影勾配降下法では、反復ごとに制約集合への射影が必要になる。これはボックス制約や ℓ_2 球に対しては容易であるが、核ノルム球（特異値の総和が与えられた上界以下である行列の集合）の上では、1 回の射影に完全な SVD のコストがかかる。フランク・ウルフ法（Frank–Wolfe method）(Frank and Wolfe 1956) は、制約集合上で目的関数の線形近似を代わりに最小化することで射影を完全に回避する。具体的には、$\mathbf{s}_k = \operatorname{argmin}_{\mathbf{s} \in C} \nabla f(\mathbf{x}_k)^{\top} \mathbf{s}$ を求め、凸結合（convex combination）$\mathbf{x}_{k+1} = (1 - \gamma_k)\mathbf{x}_k + \gamma_k \mathbf{s}_k$ をステップサイズ $\gamma_k = 2/(k+2)$ でとる。核ノルム球上ではこの線形部分問題に必要なのは勾配の最大特異ベクトル対だけであり、確率単体上では最小の座標を 1 つ見つける操作に帰着する。この手法は $O(LD^2/k)$ の速さで収束し（D は制約集合の直径）、さらに利点として、量 $\nabla f(\mathbf{x}_k)^{\top}(\mathbf{x}_k - \mathbf{s}_k)$ が追加の計算なしに毎ステップ $f(\mathbf{x}_k) - f^{\star}$ の上界を与える (Jaggi 2013)。

35.9.4 浮動小数点

35.9.4.1 確率的丸め誤差解析

35.8.3 節における総和の最悪ケースの上界は nu として線形に増大する（n 個の項を順次加算する場合）が、実際に計測される丸め誤差は通常それよりはるかに小さい。なぜなら最悪ケース解析では、個々の丸め誤差がすべて同じ最悪の方向へ積み重なると仮定しているからである。これに対し Higham and Mary (2019) は、各ステップの丸め誤差を独立で平均ゼロの確率変数としてモデル化し、高い確率で成り立つオーダー $\sqrt{n}\,u$ の上界を証明した。これは実験で典型的に見られるランダムウォーク的な相殺と一致する。同様に重要な点として、この解析は楽観的な $\sqrt{n}\,u$ の振る舞いがいつ破綻するかも明らかにしている。すなわち、10^7 個の 0.1 を足し合わせる同項の例のように、長い区間にわたって加算がすべて同じ方向へ丸められ、丸め誤差どうしが強く相関する場合である。

35.9.4.2 増大係数と反復改良法

35.8.4 節では、部分ピボット選択付き LU 分解が実用上は後退安定であることを述べた。厳密に言えば、その後退誤差は n の低次多項式と $\rho_n u$ の積で上から抑えられる。ここで**増大係数**（growth factor）ρ_n は、消去の過程で生じる最大の中間成分と $\mathbf{A}$ の最大成分との比を測る量である。最悪ケースでは、Wilkinson の有名な病的行列によって ρ_n は 2^{n-1} にまで達する。しかし実際の計算で現れるほぼすべての行列では ρ_n は穏やかな値にとどまり、万全の保証が必要な場合でも、QR 分解を用いれば約 2 倍の演算コストで指数関数的な増大を完全に排除できる (Higham 2002; Trefethen and Bau 1997)。すべての計算に倍精度のコストを払うことなく高い精度を回復するため、混合精度の反復改良法（iterative refinement）では、$\mathbf{A}$ を fp32 で安価に分解し、残差を fp64 で評価して、その低精度の因子を用いて補正量を求める。また 35.8 節の最後の演習問題では、この基本的なループが停滞するまでに $\mathbf{A}$ がどこまで悪条件になり得るかを調べる。この低精度の因子を、一般の連立一次方程式に対する反復ソルバである GMRES の内部で前処理行列として代わりに用いれば、混合精度の反復改良法をはるかに悪条件な行列へも適用できるようになる (Carson and Higham 2018)。

35.9.5 最適化全般の参考資料

書籍

- 『Convex Optimization』（Boyd、Vandenberghe 著）：凸集合と凸関数、双対性、および内点法を扱っており、無料の PDF が公開されている。35.3 節、35.6 節、35.7 節の基本文献である。
- 『Numerical Optimization』（Nocedal、Wright 著、第 2 版、Springer）：直線探索、信頼領域法、準ニュートン法、共役勾配法、および制約付き最適化を、35.1 節、35.2 節、35.6 節よりも踏み込んで解説している。
- 『Lectures on Convex Optimization』（Nesterov 著、第 2 版、Springer）：計算量限界と加速一次手法を扱う凸最適化の講義録であり、35.1 節と 35.3 節が引用する一般的な証明を収録している。
- 『Optimization for Data Analysis』（Wright、Recht 著、Cambridge、2022 年）：データ解析向けの勾配法、加速法、確率的手法、および座標降下法による最適化を扱っており、35.1 節から 35.5 節までの副読本となる。
- 『Accuracy and Stability of Numerical Algorithms』（Higham 著、第 2 版、SIAM）：丸め誤差解析、総和計算、ならびに連立一次方程式と最小二乗法の条件性を扱っている（35.8 節）。
- 『Mathematics for Machine Learning』（Deisenroth、Faisal、Ong 著）：勾配降下法、制約付き最適化、および凸性に関する自己完結した無料の章を収録している。

講義と講義動画

- 「Stanford EE364a: Convex Optimization I」（Boyd 担当）：Boyd、Vandenberghe の凸最適化の教科書に対応するスライド、課題、ソフトウェアであり、2023 年の講義動画も公開されている（35.3 節、35.7 節）。
- 「CMU 10-725: Convex Optimization」（Tibshirani 担当）：劣勾配、近接手法、確率的手法、双対性、および ADMM に関する機械学習向けの凸最適化スライドと講義録であり、2016 年秋の講義動画も利用できる（35.4 節、35.6 節、35.7 節）。

チュートリアル、講義ノート、サーベイ

- 「Why Momentum Really Works」（Goh 著、Distill）：凸二次関数におけるモメンタムを対話的な図で解説している（35.1.3 節）。
- 「Optimization Methods for Large-Scale Machine Learning」（Bottou、Curtis、Nocedal 著）：大規模学習のための確率的勾配法による最適化理論をまとめている（35.4 節）。
- 「An overview of gradient descent optimization algorithms」（Ruder 著）：モメンタム、ネステロフの加速法、AdaGrad、RMSProp、Adam といった勾配降下法の最適化アルゴリズムを並べて比較している（35.5 節）。
- 「What Every Computer Scientist Should Know About Floating-Point Arithmetic」（Goldberg 著）：浮動小数点演算における IEEE 形式、丸め、および桁落ちを解説している（35.8 節）。
- 「Optimization (Chapter 5)」（Hardt、Recht 著『Patterns, Predictions, and Actions』）：教師あり学習における最適化を簡潔に解説している（35.1 節、35.4 節）。

Chapter 36

確率と統計学習

本編全体を通して、回帰モデルは二乗誤差（squared error）で、分類器は交差エントロピー（cross-entropy）で訓練し（2.1 節、3.1 節）、有限のテストセット上で正解率を報告し、検証スコアを比較してモデルを選択してきた（2.6 節）。これらのステップはいずれも、有限の標本（sample）から推論（inference）を引き出している。コイン投げで表が 9 回（13 回中）出た場合を考えてみよう。経験的な表の割合は $9/13 \approx 0.69$ であるが、公正なコインであっても 8 回に 1 回を超える頻度で少なくとも 9 回は表が出る。テスト正解率とは、まさにその同じコインをテスト例ごとに 1 回投げたものにほかならない。本章では、有限の標本がいつ信頼できる結論を支え、いつ判断を誤らせるのかを説明する数学を展開する。

標本に伴う不確実性を整理するには、4 つの異なる概念が土台となる。結果がどのようにばらつくかを記述する**分布**、標本を数値へ写す**推定量**（estimator）、仮想的な標本間にわたる推定量の広がりとは異なる、手元の単一の標本で条件付けた**事後分布**（posterior）、そして有限の標本における小さな誤差が新たなデータにおける小さな誤差をいつ保証するかを裏づける**汎化限界**（generalization bound）である。

本章では、これら 4 つの概念を順に取り上げる。まず、連続密度（36.1 節）、同時分布とヤコビ行列（Jacobian）による変換（36.2 節）、代表的な名前付き分布（36.3 節）、そしてそれらを統一する指数分布族（exponential family、36.4 節）を確立する。これにより、線形回帰、ロジスティック回帰、ソフトマックス回帰がいずれも「予測マイナス観測」という同じ形の勾配を共有する理由が明らかになる。次に、36.5 節ではエラーバー、仮説検定、信頼区間、ブートストラップ法を展開し、36.6 節では最尤法（maximum likelihood）から訓練損失を導出してパラメータの分散をフィッシャー曲率へと結びつける。続いて、隠れ変数を動機としてエビデンス下界（evidence lower bound）と期待値最大化法（expectation–maximization）を導入し（36.7 節）、事後分布が閉じた形を持たない場合に備えてラプラス近似、モンテカルロ法、マルコフ連鎖モンテカルロ法、変分近似を扱う（36.8 節）。最後に、36.9 節で有限標本における裾の上界と高次元における球殻への集中を証明し、36.10 節でそれらを選択後の一様収束（uniform convergence）へと拡張し（第 3.5 章）、36.11 節で分布、平滑化、較正（calibration）を画像分類へと応用する。

1.6 節の内容に馴染みのある読者は冒頭のモーメントの代数計算を流し読みしてよく、また 36.8 節と 36.11 節は最初の通読時には読み飛ばしても後続の節の理解には支障がない。エントロピーとダイバージェンスの本格的な理論は第 37 章に引き継がれ、測度論（measure theory）、時系列（time series）、およびノンパラメトリックな拡張については 36.12 節で概観する。

36.1 確率変数

1.6 節では、数え上げによって確率を構築した。サイコロには 6 つの面があり、ラベルは 10 個のクラスのいずれかに属し、事象の確率はそこに含まれる結果の確率を足し合わせたものにすぎない。しかし、ニューラルネットワークが扱う数値の大半はそのようなものではない。ピクセルの強度、初期化時に抽出される重み、あるいは拡散モデル（diffusion model）が画像に加えるガウスノイズは、いずれも区間内の任意の値を取り得るため、値を 1 つずつ列挙するには多すぎる。では、そのうちのたった 1 つの値が起こる確率はいくらだろうか。

丸いボードに向かって無造作に投げられ、ボード上のどこに刺さる確率も等しいダーツを考えよう。そして、そのダーツが中心からちょうど半径の半分の位置に刺さったとする。その起こりやすさを尋ねられたら、距離を丸めて（たとえば半径の 0.45 倍から 0.55 倍の間として）答えるだろう。これはおおよそ 10 回に 1 回の割合で起こる。精度をもう 1 桁求めれば確率は 10 分の 1 になり、さらにもう 1 桁求めればさらに 10 分の 1 になる。桁数を無限に増やすと確率は 0 になるが、それでもダーツがどこかに刺さったことは確かである。

この難点を抜け出すには、1 点の確率を問うのをやめ、その近くで確率がどれだけの速さで蓄積するかを問えばよい。区間を 10 分の 1 に狭めるたびに確率も 10 分の 1 に減るため、確率と幅の比は一定に保たれる。この単位長さあたりの確率が**密度**（density）である。密度を用いれば、和を積分に置き換えることで、離散確率のあらゆる道具を連続的に変化する量へとそのまま引き継ぐことができる。

以下の内容の一部は 1.6 節の再訪であり、連続変数へと拡張して証明を与える。平均と分散の意味づけはすでにそこで述べたため、ここではただちに連続版の形と代数的な性質へ進む。それらと並行して、密度そのもの、累積分布関数からサンプルを抽出する汎用的な手法、Markov と Chebyshev による裾の上界、そして裾があまりに重いために両方の上界が破綻する釣鐘型の密度を導入する。本節から結果を 2 つだけ持ち帰るなら、密度と累積分布関数の関係 Equation 36.6 とマルコフの不等式 Equation 36.14 を押さえておきたい。本章における残りの裾の上界はすべてここから導かれる。本節全体を通してダーツの例を追い、中心からの距離という 1 つの数値に注目する。2 つの座標や密度の変換については 36.2 節に譲る。

36.1.1 密度

密度の考え方を 3 つの段階で展開しよう。まず 1 点において、限りなく狭くなるビンの極限として捉える。次に区間上において、曲線の下の面積として捉える。最後に、その面積を蓄積する関数を通じて捉える。この関数からは、乱数サンプルを抽出する実用的な方法も得られる。

36.1.1.1 ビンから密度へ

「単位長さあたりの確率」を厳密にするため、まずダーツが指定された距離の範囲内に刺さる確率を求めよう。R を、半径 1 のボードの中心からダーツまでの距離とする。ダーツが中心から距離 r 以内に刺さるのは、半径 r の円盤に当たるときに限られる。ボードのどの部分も等しく起こりやすいため、この確率は単にボードの面積に占めるその円盤の割合となる。

$$P(R \le r) = \frac{\pi r^2}{\pi} = r^2, \qquad 0 \le r \le 1.$$

半径の半分の内側には面積の 4 分の 1 しかないため、確率も 4 分の 1 しか含まれない。したがって、幅 ϵ で中心が 0.5 のビンに入る確率は $(0.5 + \epsilon/2)^2 - (0.5 - \epsilon/2)^2 = \epsilon$ となり、冒頭の例の数値が得られる。

ビン	幅 ϵ	確率	確率 / ϵ
$[0.45, 0.55]$	0.1	0.1	1
$[0.495, 0.505]$	0.01	0.01	1
$[0.4995, 0.5005]$	0.001	0.001	1

ビンを 10 分の 1 に狭めるたびに確率も 10 分の 1 になり、比は変わらない。この極限の比こそ名前をつけたい量である。確率変数（random variable）X の点 x における**確率密度**（probability density）とは、x の近傍における単位長さあたりの確率であり、次式で表される。

$$p(x) = \lim_{\epsilon \downarrow 0} \frac{P(x \le X < x + \epsilon)}{\epsilon},$$

ただし、この極限が存在するものとする。言い換えれば、短い区間に対する確率は、おおよそ区間の長さと密度の積になる。

$$P(x \le X < x + \epsilon) \approx \epsilon\, p(x), \tag{36.1}$$

ここで誤差は ϵ よりも速く 0 に近づく。これは密度を 1 点ずつ記述したものであり、当面はこれで十分である。区間を用いた厳密な定義は次の項で与える。ダーツの例では、任意の距離 r（区間 $(0, 1)$ 内）に対して、区間 $[r, r + \epsilon)$ の確率は $(r + \epsilon)^2 - r^2 = 2r\epsilon + \epsilon^2$ である。これを幅 ϵ で割ると $2r + \epsilon$ となり、$\epsilon \to 0$ とすると次式が得られる。

$$p(r) = 2r, \qquad 0 \le r \le 1. \tag{36.2}$$

密度は中心から外縁に向かって線形に増大する。これは半径 r の円の円周の長さをそのまま反映しており、外側に行くほどボードの面積が広いという単純な理由による。$r = 0.5$ における密度は 1 であり、上の表の比と一致する。

なお、密度は確率ではない。$r = 0.9$ ではダーツの密度は 1.8 であり、$[0, 0.1]$ のどこにも等しい起こりやすさで落ちる変数はその区間上で密度 10 を持つ。合計 1 の確率を長さ 0.1 に詰め込むからである。密度は任意の非負の値をとり得るが、10 を超えられるのは合計の長さが $1/10$ 未満の集合上に限られる。さもなければ、その集合の確率が 1 を超えてしまうからである。密度は速さと同じく「変化率」だと考えるとよい。自動車の速さは距離がどれだけの速さで積み上がるかを表すものであり、すでに走った距離を表すのではない。同様に、密度が 1.8（$r = 0.9$ において）であるとは、r が 0.9 を超えて増えるときに確率がどれだけの速さで蓄積するかを表しており、点 0.9 そのものにどれだけの確率が乗っているかを表すものではない。

1 つの点は長さが 0 であるため、確率も 0 になる。任意の $\epsilon > 0$ に対して点 x は $[x, x + \epsilon)$ に含まれるため、$P(X = x)$ は高々おおよそ $\epsilon\, p(x)$ であり、ϵ が 0 に近づくにつれて消える。したがって、密度を持つ任意の確率変数に対して次が成り立つ。

$$P(X = x) = 0 \qquad \text{for every } x. \tag{36.3}$$

これにより冒頭のパズルが解ける。ダーツが距離 0.5 に刺さる確率は、他のあらゆる厳密な距離と同じく 0 であるが、それでもダーツはどこかには必ず刺さる。連続確率においては、確率 0 の事象は不可能を意味しない。

正の確率を持つ 1 つの点は**アトム**と呼ばれる。密度を持つ変数はアトムを持たない一方、サイコロの出目のような離散変数はすべてアトムから成り立つ。両方を持つ変数もある。たとえば、標

準正規分布に従う入力を受け取る ReLU ユニットの出力は、0 を確率 $\frac{1}{2}$ でとり（アトム）、残りの半分の確率を密度によって $(0, \infty)$ 上に広げる（詳細は演習問題 2 で扱う）。

上の表は厳密な面積の公式から求めたが、シミュレーションには公式は一切不要である。円形のボードにダーツを一様に投げるには、ボードを囲む正方形 $[-1, 1]^2$ に一様に投げ、円の内側に刺さったものだけを残せばよい。これでボードのどの部分も等しい確率になる。以下では、3 つのビンのそれぞれに入ったダーツの割合を数え、ビンの幅で割る。

```
rng = np.random.default_rng()
xy = rng.uniform(-1, 1, size=(4 * 10**6, 2))          # darts on the square
on_board = (xy**2).sum(axis=1) <= 1                   # keep those on the board
r_dart = np.linalg.norm(xy[on_board], axis=1)         # distances R
for width in [0.1, 0.01, 0.001]:
    prob = np.mean(np.abs(r_dart - 0.5) <= width / 2)  # fraction in the bin
    print(f'width {width:<5}  probability {prob:.5f}  ratio {prob / width:.3f}')
```

```
width 0.1    probability 0.10008  ratio 1.001
width 0.01   probability 0.01009  ratio 1.009
width 0.001  probability 0.00104  ratio 1.041
```

予想どおり、ビンを 10 分の 1 に狭めるたびに割合も 10 分の 1 に減り、比は $1 = p(0.5)$ の近くに保たれる。しかし、厳密な表からは見えなかったのは、極端に狭いビンが実際にはどれほどノイズの影響を受けるかという点である。400 万本のダーツのうち約 3.1 百万本がボードに刺さり、そのうち約 3,000 本しか幅 0.001 のビンに入らないため、最後の比は実行ごとに数パーセント変動する。

36.1.1.2 面積としての確率

ここまでは、微小な区間の確率から密度を読み取ってきた。今度は逆向きに考えてみよう。密度が与えられたとき、ある区間全体の確率はどう求めればよいだろうか。実数直線をビン $[\epsilon i, \epsilon(i+1))$（幅 ϵ）に分割しよう。Equation 36.1 より、各ビンの確率はおおよそ $\epsilon\, p(\epsilon i)$ であり、ビンは互いに素であるため確率は足し合わせられる。幅と高さの積を足し合わせるとリーマン和（Riemann sum）になり、これは曲線の下の面積、すなわち定積分（34.6 節）へと ϵ が 0 に近づくにつれて収束する。X は必ず何らかの実数値をとるため、すべてのビンを合わせた確率は 1 になる。

$$1 = P(X \in \mathbb{R}) \approx \sum_i \epsilon\, p(\epsilon i) \;\xrightarrow[\epsilon \to 0]{}\; \int_{-\infty}^{\infty} p(x)\, dx.$$

a と b の間にあるビンだけを足し合わせれば、区間の確率はその区間上での密度の下の面積にほかならないことが示唆される。厳密に言えば、この議論は証明ではなく動機づけである。ビンはおおよそ $1/\epsilon$ 個あり、個々の誤差が ϵ に比べて小さくても、その合計が無視できるほど小さくなるのは誤差が x にわたって一様に縮む場合に限られるからである。議論を確かな土台に載せるため、この面積の公式を正式な定義としよう。

定義　確率変数 X が p を**密度に持つ**とは、$p \geq 0$ であり、かつすべての $a < b$ に対して次式が成り立つことをいう。

$$P(X \in (a, b]) = \int_a^b p(x)\, dx. \tag{36.4}$$

$a \to -\infty$ および $b \to \infty$ とすれば、いかなる密度もその下の全面積が 1 であることが確かめられる。実際、非負性（$p \geq 0$）と全面積が 1 であるという 2 つの条件さえ満たせば、任意の関数が有効な密度となる。Figure 36.1 に、2 つの峰を持つ密度に対するこの定義を示す。ダーツの例もこの条件を満たしている。$0 \leq a < b \leq 1$ に対して、積分 $\int_a^b 2r\,dr = b^2 - a^2$ は面積から求めた確率 $P(a < R \leq b)$ を再現し、$2r$ が確かにその密度であることが確認できる。とくに、半径の外側半分に刺さる確率は $\int_{0.5}^1 2r\,dr = 1 - 0.25 = 0.75$ であり、実にボードの 4 分の 3 が中心から半径の半分より外側にある。

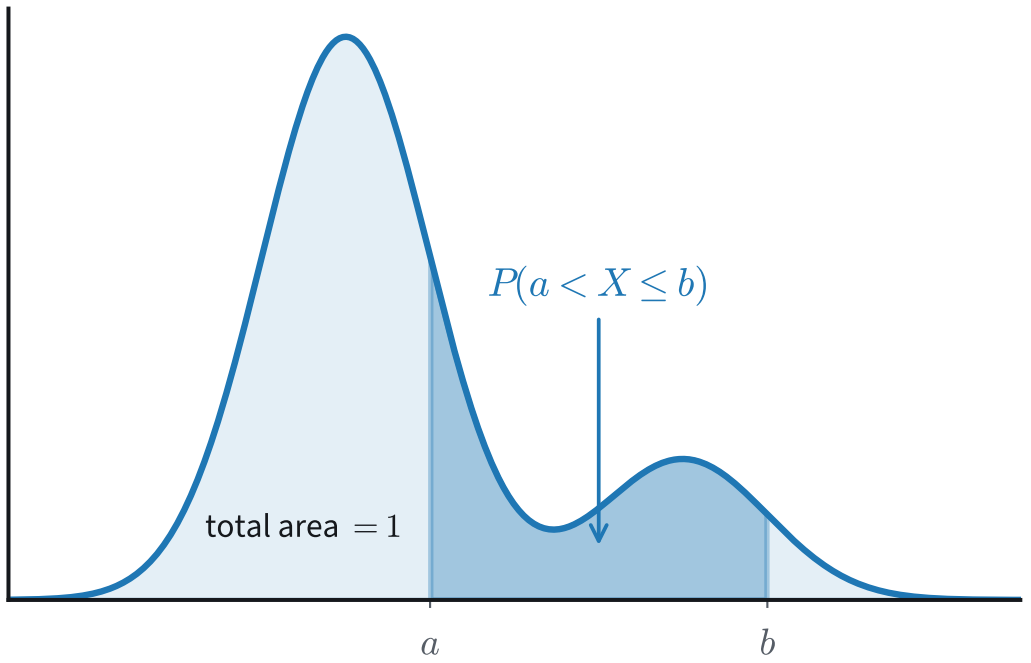


Figure 36.1: 確率密度 $p(x)$。曲線の下の全面積は 1 であり、X が a から b までの区間に入る確率は、その上の網掛け部分の面積 $\int_a^b p(x)\,dx$ である。密度そのものは確率ではなく（1 を超えることもある）、区間上の積分だけが確率となる。

36.1.1.3 累積分布関数

値そのものが真の確率であり、離散変数と連続変数を同じように扱える関数を用いると便利なことが多い。X の**累積分布関数**（cumulative distribution function、CDF）は、X が x 以下の値をとる確率を与える。X が密度を持つとき、これは単に x より左側にある密度の下の面積である。

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} p(t)\,dt. \tag{36.5}$$

言い換えれば、F は実数直線を左から右へ走査しながら確率を蓄積していく。その値は 0（$x \to -\infty$ の極限）から始まり、1（$x \to \infty$ の極限）へと上昇する。非負の密度 p は面積を増やすことしかしないため減少することは決してなく、X が密度を持つときは常に F は連続である。ダーツの例については冒頭ですでに計算したとおり、$F(r) = 0$（$r < 0$）、$F(r) = r^2$

$(0 \le r \le 1)$、$F(r) = 1$（$r > 1$）である。この関数こそが出発点であり、密度 $2r$ はそこから直接導かれた。

密度とは異なり、累積分布関数はあらゆる確率変数に対して定義される。0 または 1 を返す公平なコインの累積分布関数は次のようになる。

$$F(x) = \begin{cases} 0 & x < 0, \\ \frac{1}{2} & 0 \le x < 1, \\ 1 & x \ge 1, \end{cases}$$

これは各アトムの確率の分だけ上方にジャンプする**階段関数**である。$F(x) = P(X \le x)$ はちょうど x の上にあるアトムも含むため、この関数は各ジャンプにおいて上側の値をとる。より一般に、すべての累積分布関数は**右連続**（right-continuous）であり、すなわち $\lim_{y \downarrow x} F(y) = F(x)$ を満たす。混合型の変数は両方の振る舞いを併せ持つ。先ほど挙げた ReLU の出力の累積分布関数は、値が 0（$x < 0$）から始まり、$\frac{1}{2}$ へと $x = 0$ でジャンプし、そこから 1 に向かって滑らかに上昇する。したがって、単一の関数によって離散・連続・混合の確率変数が統一される。Figure 36.2 にそれぞれの例を示す。

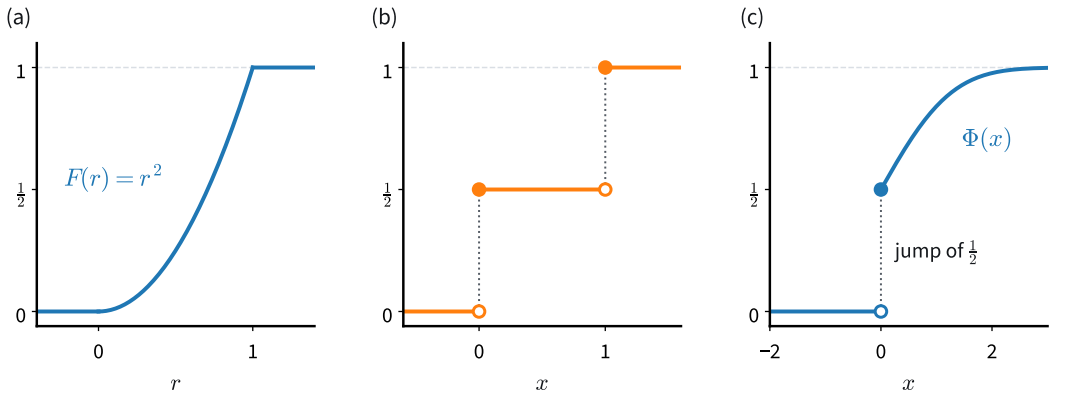


Figure 36.2: 3 つの累積分布関数。(a) 中心からダーツまでの距離：$F(r) = r^2$ は 0 から 1 まで連続的に上昇する。(b) 0 または 1 を返す公平なコイン：各アトムで $\frac{1}{2}$ ずつジャンプし、そこで上側の値をとる階段関数（黒丸）。(c) 標準ガウス分布の ReLU：F は 0（$x < 0$）から始まり、$\frac{1}{2}$ へと $x = 0$ でジャンプし、そこからはガウス分布の累積分布関数 $\Phi(x)$ に従う。

密度が存在するとき、累積分布関数と密度は同じ情報を持つ。p を積分すれば F が得られ、F を微分すれば p が復元される。これにより、冒頭で見た縮小するビンの極限が定理として確立される。

命題（密度は累積分布関数の導関数）　**X が密度 p（点 x で連続）を持つならば、F は x で微分可能であり、次式が成り立つ。**

$$F'(x) = p(x). \tag{36.6}$$

とくに、単位長さあたりの確率 $P(x \le X < x + \epsilon)/\epsilon$ は、$p(x)$ へと $\epsilon \downarrow 0$ の極限で収束する。

証明　Equation 36.5 より、F は p の累積面積を表す関数である。微分積分学の基本定理によれば、累積面積を表す関数の導関数（derivative）は、被積分関数が連続なすべての点で被積分関数

そのものに一致する（34.6 節）。後半の主張については、1 点 x の確率は高々 $\int_{x-\delta}^{x} p(t)\,dt$（任意の $\delta > 0$ に対して）であり、これらの積分は δ とともに 0 に収束する。したがって端点を区間に入れるか除くかで確率は変わらず、$P(x \le X < x+\epsilon) = F(x+\epsilon) - F(x)$ となるため、この比は単に F の差分商である。■

ダーツの例では、$F'(r) = 2r$ が $(0,1)$ 上で成り立ち、ビンから求めた密度が復元される。Figure 36.3 は 2 つの関数を並べて示したものであり、左図の網掛け面積 $\int_a^b p$ は右図の縦方向の上昇量 $F(b) - F(a)$ に等しい。

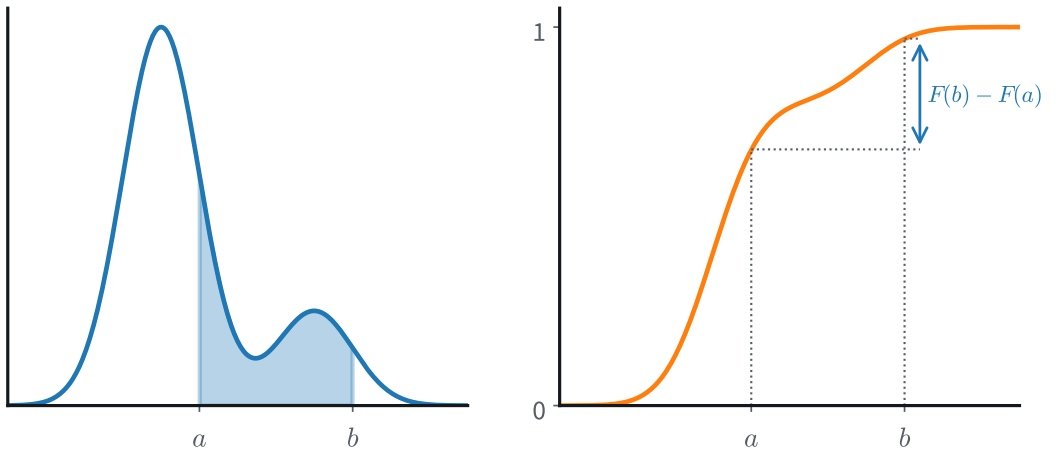


Figure 36.3: 左：密度 p と、a から b までの区間上で網掛けされた面積。右：対応する累積分布関数 $F(x) = \int_{-\infty}^{x} p$。0 から 1 へ増加する曲線であり、上昇量 $F(b) - F(a)$ は左図の網掛け面積に等しい。密度は累積分布関数の傾き $(F' = p)$ であり、累積分布関数は密度の下の累積面積である。

36.1.1.4 分位点と逆変換サンプリング

コンピュータを使えば、0 と 1 の間のどこにも等しい確率で落ちる乱数を手軽に利用できる。ダーツをはじめ任意の連続分布をシミュレーションするには、そうした一様乱数を目的の分布からのサンプルへと変換する方法が必要になる。

構成要素となるのは、あらゆる密度のなかで最も単純なものである。確率変数 U が $[0,1]$ **上で一様**であるとは、その密度が値 1 を $[0,1]$ 上でとり、それ以外では 0 であることをいう。このとき $[0,1]$ の任意の部分区間の確率はその長さに等しく、累積分布関数は単に $F_U(u) = u$ $(0 \le u \le 1)$ となる。

累積分布関数は値 x を確率水準 u へと写す。この写像を逆転させ、指定した割合 u の確率がそれ以下に収まるような値を問うことで、**分位点関数**（quantile function）が得られる。F が連続であり、区間 I（$0 < F < 1$ を満たす範囲）上で狭義単調増加であるとき、この関数は I を $(0,1)$ の上へ 1 対 1 に写し、その逆関数 $F^{-1} : (0,1) \to I$ が分位点関数となる。$u = 0.25, 0.5, 0.75$ における値は**四分位点**（quartile）と呼ばれ、真ん中の四分位点 $F^{-1}(0.5)$ は**中央値**（median）と呼ばれる。ダーツの例では $F(r) = r^2$ の逆関数は $F^{-1}(u) = \sqrt{u}$ となり、四分位点は 0.5、0.707、0.866 となる。ダーツの中央値は 0.707 であり、半径の半分よりかなり外側にある。ボードの 4 分の 3 が 0.5 より外側にあるためである。

注目すべきことに、分位点関数は汎用的なサンプラでもある。

命題（逆変換サンプリング） F を連続な累積分布関数とし、区間 I（$0 < F < 1$ を満たす範囲）上で狭義単調増加であるとする。また、U を $[0,1]$ 上で一様な確率変数とする。このとき、$X = F^{-1}(U)$ は累積分布関数 F を持つ。

証明 確率 1 で U は $(0,1)$ に属するため、X は定義され I に属する。点 x が I に属するとき、事象 $F^{-1}(U) \le x$ と $U \le F(x)$ は一致する。なぜなら F は I 上で増加関数だからである。また、U は $[0,1]$ 上で一様であるため、2 つ目の事象の確率は区間 $[0, F(x)]$ の長さに等しい。

$$P(F^{-1}(U) \le x) = P(U \le F(x)) = F(x).$$

点 x が I より左側にあるときは $F(x) = 0$ であり常に $X > x$ が成り立つ。一方、点 x が I より右側にあるときは $F(x) = 1$ であり常に $X \le x$ が成り立つ。■

なお、この仮定のもとでは、区間の外側で F が平坦であってもよい。実際、ダーツの累積分布関数もそうなっている（$F = 0$（$r < 0$ のとき）かつ $F = 1$（$r > 1$ のとき））。F^{-1} を**一般化逆関数**（generalized inverse）$F^{-}(u) = \min\{x : F(x) \ge u\}$ に置き換えれば、この仮定自体が不要となり、階段状の累積分布関数からもサンプリングできるようになる。

ダーツに対して逆変換サンプリング（inverse-transform sampling）を適用すると $R = \sqrt{U}$ となる。近道をして $R = U$ と置きたくなるかもしれないが、それがなぜ誤りかはコードを 1 行も実行せずにすぐわかる。$R = U$ とするとダーツの半分が $r < 0.5$ の内側に入ってしまうが、本来そこに入るべき割合は 4 分の 1 にすぎない。Figure 36.4 に、2 つの峰を持つ密度におけるこの構成の幾何学的な意味を示す。密度が高い場所では累積分布関数が急勾配で立ち上がるため、縦軸上で等間隔に並んだ水準は、横軸上では密集したサンプルの塊へと圧縮される。

これで、ダーツの距離を生成するまったく異なる 2 つの方法が得られた。正方形にダーツを投げてボード上のものだけを残す方法と、単に $\sqrt{U}$ を計算する方法である。以下のセルでは、素朴な近道である $R = U$ も並べて両者を比較する。

```
u = rng.uniform(size=10**6)
r = np.sqrt(u)                                          # R = F^{-1}(U)
print(f'P(R < 0.5): darts {np.mean(r_dart < 0.5):.3f}, sqrt(U) '
      f'{np.mean(r < 0.5):.3f}, naive U {np.mean(u < 0.5):.3f}')
print(f'mean distance: darts {r_dart.mean():.3f}, sqrt(U) {r.mean():.3f}')
```

```
P(R < 0.5): darts 0.250, sqrt(U) 0.250, naive U 0.500
mean distance: darts 0.667, sqrt(U) 0.667
```

2 つの正しい手法は一致する。どちらも距離の 4 分の 1 が 0.5 未満に入り、平均距離は次節で導出する 0.667 の近くになる。素朴な近道である U はサンプルの半分を 0.5 未満に入れてしまう。中心付近より外縁付近のほうがボードの面積がはるかに広いという事実を無視して、距離を $[0,1]$ 全体に均等に広げてしまうからである。

36.1.2 平均と分散

密度は分布全体を完全に規定するが、分布の中心がどこにあり、どれほど広く散らばっているかという 2 つの要約値だけで、知りたいことの大半がわかる場合が多い。1.6 節では離散変数に対してこの両者を導入し、その動機づけを与えた。ここではそれらを密度へと拡張し、本節以降で依拠する代数的な規則を証明する。

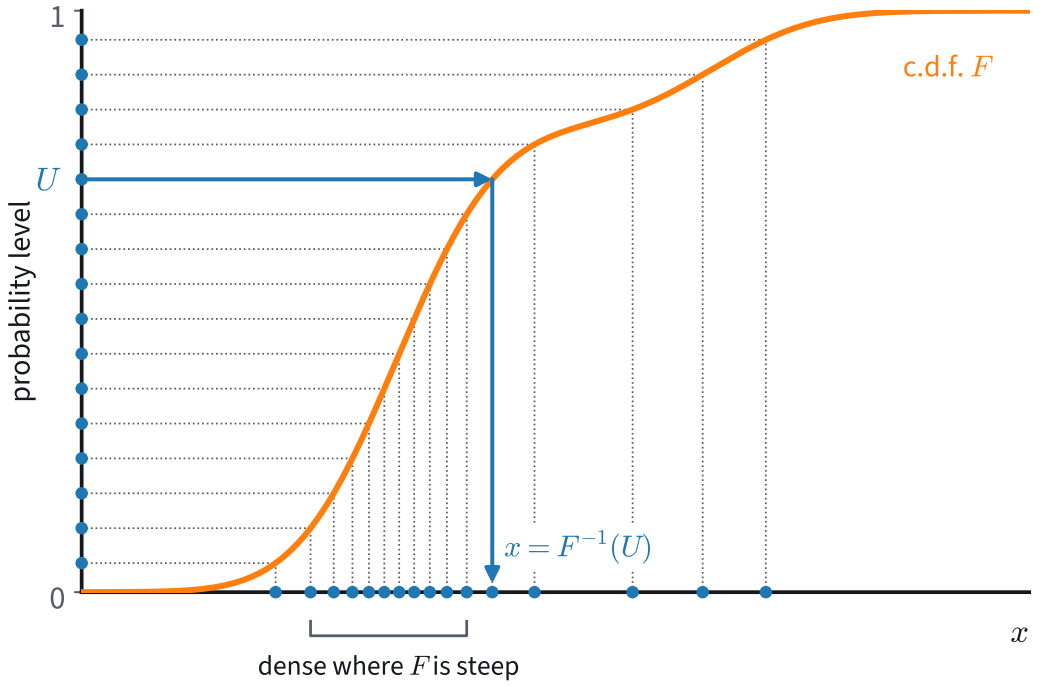


Figure 36.4: 逆変換サンプリング。縦軸（左）上の一様な水準 U が、累積分布関数 F で折り返されて横軸上の値 $x = F^{-1}(U)$ へと写される。F の傾きが急な場所、すなわち密度 $p = F'$ が大きい場所では、一様に広がった水準が密なサンプルの集まりへと圧縮されるため、折り返された点のヒストグラムは p を再現する。

36.1.2.1 平均

平均は X の各値をその確率で重み付けして平均したものである。密度を持つ変数では、先ほどと同様に実数直線を幅 ϵ のビンに切り分けると、位置 ϵi のビンはその値 ϵi と確率（おおよそ $\epsilon\, p(\epsilon i)$）の積だけ寄与し、その和は積分へと収束する。したがって、X の**平均**、すなわち**期待値**（expectation）は次式で与えられる。

$$\mu_X = \mathbb{E}[X] = \sum_i x_i\, p_i \quad \text{(discrete)}, \qquad \mu_X = \mathbb{E}[X] = \int_{-\infty}^{\infty} x\, p(x)\, dx \quad \text{(density)}. \tag{36.7}$$

離散の場合と同様、平均は釣り合いの点として働く。長さに沿って線密度 p を持つ細い棒は μ_X で釣り合う。この定義が意味を持つためには、離散の場合には $\sum_i |x_i|\, p_i < \infty$、密度の場合には $\int |x|\, p(x)\, dx < \infty$ が成り立つ必要がある。この条件が破れると平均は存在しない。実際、そのような具体的な密度にまもなく出会うことになる。同じ確率の重みを用いれば、任意の関数 $g(X)$ の平均も直接求めることができる。

$$\mathbb{E}[g(X)] = \sum_i g(x_i)\, p_i, \qquad \mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)\, p(x)\, dx,$$

ここでも $|g|$ の和または積分が有限であることを条件とする。これがいかに便利であるかに注目してほしい。g の値を X の確率で重み付けするだけでよく、変換後の変数 $g(X)$ の分布をあらかじめ求める必要がないのである。この規則はあまりにも自然に使えるため、$\mathbb{E}[g(X)]$ の定義ではなく定理であることをつい忘れてしまうことから、**無意識な統計学者の法則**（law of the unconscious statistician）という有名な呼び名がついている。なお、$\mathbb{E}[g(X)]$ は一般に $g(\mathbb{E}[X])$ とは異なる。g が凸関数であるとき、両者のどちらが大きいかを教えてくれるのがイェンセンの不等式（35.3.3.1 節）である。

ダーツの例では次のようになる。

$$\mathbb{E}[R] = \int_0^1 r \cdot 2r\, dr = \tfrac{2}{3}.$$

平均距離は 0.667 であり、0.5 ではない。これもやはり中心付近より外縁付近のほうがボードの面積が広いためである。また、平均が中央値 0.707 よりわずかに小さいことにも注目しよう。中心付近に刺さる比較的少数のダーツは、真ん中の順位の値をずらす以上に平均値を下方向へ引き下げるのである。

期待値の性質のうち最も頻繁に用いるのが線形性である。1.6 節でもすでに用いたが、ここで証明を与えよう。

命題（期待値の線形性）　平均が存在する確率変数 X, Y と定数 c, d に対して、次式が成り立つ。

$$\mathbb{E}[cX + d] = c\,\mathbb{E}[X] + d, \qquad \mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]. \tag{36.8}$$

証明　どちらの恒等式も、平均を定義する和（または積分）の線形性から従う。1 つ目の式については、$\mathbb{E}[cX+d] = \sum_i (cx_i + d)p_i = c\sum_i x_i p_i + d\sum_i p_i = c\,\mathbb{E}[X] + d$ が $\sum_i p_i = 1$ を用いて得られる。2 つ目の式については、p_{ij} を $(X, Y) = (x_i, y_j)$ となる確率、すなわち 1.6 節と

同様の組の同時分布（joint distribution）とする。和を分割し、周辺確率 $\sum_j p_{ij} = P(X = x_i)$ と $\sum_i p_{ij} = P(Y = y_j)$ に着目すれば、次式が得られる。

$$\mathbb{E}[X + Y] = \sum_{i,j} (x_i + y_j)\, p_{ij} = \sum_{i,j} x_i\, p_{ij} + \sum_{i,j} y_j\, p_{ij} = \mathbb{E}[X] + \mathbb{E}[Y].$$

密度の場合も、同時密度（p_{ij} の連続版であり 36.2 節で定義する）の上での積分によってまったく同じ計算が成り立つ。■

決定的に重要なのは、2 つ目の恒等式には独立性が一切不要だという点である。X と Y がどれほど強く互いに従属していても、和の平均は平均の和に等しい。データセット上の平均として期待損失をとるたびに、この事実に依拠している。

36.1.2.2 分散と標準偏差

平均は分布がどこに位置するかを示すが、その周りにどれほど密に集まっているかについては何も語らない。1.6 節と同様、広がりの程度は平均からの二乗距離の平均で測り、これを**分散**（variance）と呼ぶ。

$$\sigma_X^2 = \operatorname{Var}(X) = \mathbb{E}\big[(X - \mu_X)^2\big], \tag{36.9}$$

分散は $\mathbb{E}[X^2] < \infty$ である限り定義される。二乗を展開し、線形性 Equation 36.8 を μ_X を定数とみなして適用すると、通常はるかに計算しやすい形、すなわち「二乗の平均から平均の二乗を引いた式」が得られる。

$$\operatorname{Var}(X) = \mathbb{E}\big[X^2 - 2\mu_X X + \mu_X^2\big] = \mathbb{E}[X^2] - 2\mu_X\, \mathbb{E}[X] + \mu_X^2 = \mathbb{E}[X^2] - \mu_X^2. \tag{36.10}$$

したがって、密度を持つ変数の平均と分散は次の 2 つの積分で与えられる。

$$\mu_X = \int_{-\infty}^{\infty} x\, p(x)\, dx, \qquad \sigma_X^2 = \int_{-\infty}^{\infty} x^2 p(x)\, dx - \mu_X^2. \tag{36.11}$$

ダーツの例では $\mathbb{E}[R^2] = \int_0^1 r^2 \cdot 2r\, dr = \tfrac{1}{2}$ であるため、$\operatorname{Var}(R) = \tfrac{1}{2} - \tfrac{4}{9} = \tfrac{1}{18} \approx 0.056$ となる。また、一様変数 U（$[0,1]$ 上）では $\mu_U = \int_0^1 u\, du = \tfrac{1}{2}$ であり、$\operatorname{Var}(U) = \tfrac{1}{3} - \tfrac{1}{4} = \tfrac{1}{12}$ となる。

単位のリスケーリングやデータの中心化はアフィン変換（affine transformation）である。アフィン写像 $cX + d$ のもとでは、平行移動 d は分散から消え、スケール因子 c は二乗されて外に出る。実際、線形性より $\mu_{cX+d} = c\mu_X + d$ となるため、平均からの偏差は $(cX + d) - \mu_{cX+d} = c(X - \mu_X)$ となり、これを二乗して期待値をとれば次式が得られる。

$$\operatorname{Var}(cX + d) = c^2 \operatorname{Var}(X). \tag{36.12}$$

さらに 2 つの性質を心に留めておくとよい。第 1 に、分散は決して負にならず、分散が 0 になるのは X が確率 1 で定数であるとき、かつそのときに限る。非負の量 $(X - \mu_X)^2$ の期待値が 0 になり得るのは、その量が確率 1 で 0 である場合だけだからである。第 2 に、互いに独立な確率変

数の分散は足し合わせられる（1.6 節）。この事実は、変数同士が相互作用するときに独立性に代わって現れる共分散（covariance）の項とともに、36.2.2.1 節で証明する。

分散は偏差を二乗するため、ドルで測った利益に対して「ドルの二乗」になるように、単位も二乗になる。その平方根をとれば元の単位に戻り、**標準偏差**（standard deviation）が得られる。

$$\sigma_X = \sqrt{\operatorname{Var}(X)}, \tag{36.13}$$

標準偏差はアフィン写像のもとで $\sigma_{cX+d} = |c|\,\sigma_X$ と線形にスケールする。ダーツの例では、ボードの半径を単位として $\sigma_R = \sqrt{1/18} \approx 0.236$ である。

36.1.3 分布の裾

標準偏差は、平均の周りでの典型的な変動に対する自然なスケールを定める。次に生じる自然な問いは、平均だけ、あるいは平均と分散だけがわかっているとき、遠く離れた裾にどれほどの確率が乗り得るかということである。2 つの古典的な不等式が、あらゆる分布に対して一度にこの問いに答える。どちらの上界もかなり大まかであるが、驚くべきことに追加の仮定なしにはこれ以上鋭くできない最良の評価であることが、単純な 3 点分布によって示される。続いて、平均も分散も持たない釣鐘型の密度を取り上げ、これら 2 つの仮定が何を食い止めていたのかを明らかにする。

36.1.3.1 マルコフの不等式とチェビシェフの不等式

最初の上界は平均のみを用い、任意の非負の確率変数に適用できる。証明はわずか 1 行で済むにもかかわらず、本章におけるほぼすべての裾の上界は、うまく選んだ変数の関数にこの不等式を適用することで導かれる。

命題（マルコフの不等式） $X \geq 0$ **が平均を持ち、**$t > 0$ **であるとき、次式が成り立つ。**

$$P(X \geq t) \leq \frac{\mathbb{E}[X]}{t}. \tag{36.14}$$

証明 $\mathbf{1}_{X \geq t}$ を、値が 1（$X \geq t$ のとき）または 0（それ以外のとき）をとる指示変数とする。このとき、すべての結果に対して $X \geq t\,\mathbf{1}_{X \geq t}$ が成り立つ。実際、$X \geq t$ のときは右辺が $t \leq X$ となり、それ以外のときは $0 \leq X$ となる。両辺の期待値をとり、$\mathbb{E}[\mathbf{1}_{X \geq t}] = P(X \geq t)$ を用いれば $\mathbb{E}[X] \geq t\,P(X \geq t)$ が得られる。■

幾何学的には、$[0, \infty)$ 上において 0（t 未満）および t（t 以上）をとる階段関数は常に直線 $y = x$ の下側にあるため（Figure 36.5）、その平均値 $t\,P(X \geq t)$ も X の平均値の下側になければならない。t を平均の 10 倍に設定すれば、ただちに便利な経験則が得られる。所得、ファイルサイズ、勾配ノルムのいずれを扱う場合でも、非負の母集団において平均の 10 倍を超えられるのは高々 10 分の 1 である。

マルコフの不等式は平均しか使わないため、必然的に大まかな評価となる。これを二乗偏差 $(X - \mu_X)^2$（その平均は分散である）に適用すれば、両側の裾を同時に抑えられる。この結果は Bienaymé と Chebyshev によるものである (Chebyshev 1867)。

命題（チェビシェフの不等式） **有限かつ非ゼロの分散を持つ任意の** X **と任意の** $\alpha > 0$ **に対して、次式が成り立つ。**

$$P(|X - \mu_X| \geq \alpha \sigma_X) \leq \frac{1}{\alpha^2}. \tag{36.15}$$

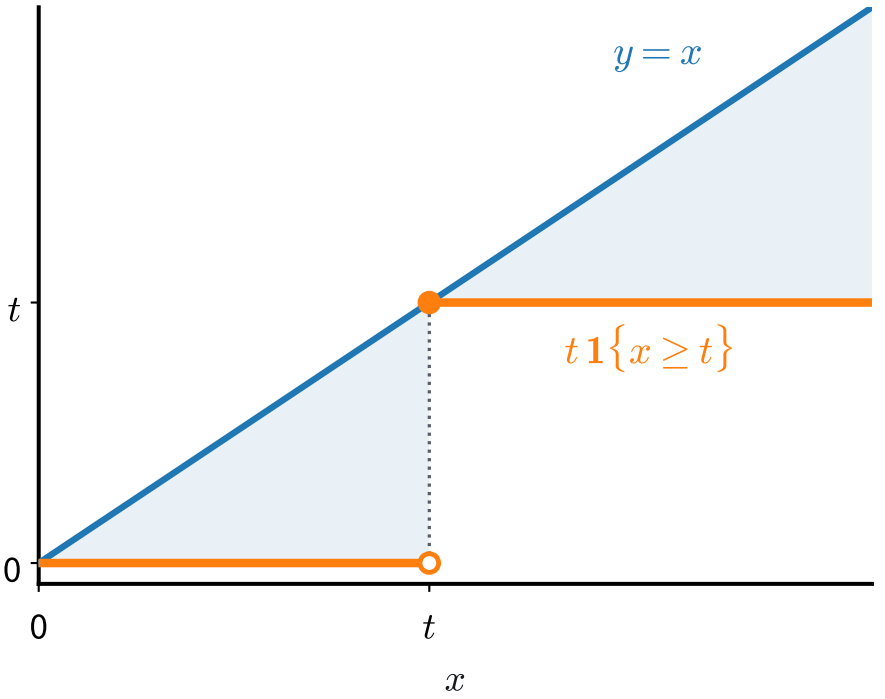


Figure 36.5: マルコフの不等式の幾何学的解釈。$x \geq 0$ において階段関数 $t\,\mathbf{1}\{x \geq t\}$（オレンジ）は直線 $y = x$（青）の下側にあるため、その平均 $t\,P(X \geq t)$ は $\mathbb{E}[X]$ を超えられない。網掛けされた隙間が、この上界で捨てられている余裕分に相当する。

証明 変数 $Z = (X - \mu_X)^2$ は非負で平均 $\mathrm{Var}(X) = \sigma_X^2$ を持ち、事象 $|X - \mu_X| \geq \alpha\sigma_X$ は事象 $Z \geq \alpha^2\sigma_X^2$ と同一である。$\sigma_X > 0$ より、しきい値 $t = \alpha^2\sigma_X^2$ は正であるため、マルコフの不等式 Equation 36.14 から $P(|X - \mu_X| \geq \alpha\sigma_X) \leq \sigma_X^2/(\alpha^2\sigma_X^2) = 1/\alpha^2$ が得られる。■

とくに、いかなる分布であっても、平均から標準偏差の 10 倍以上離れられるのは高々 1% である。同じ証明を絶対的なしきい値 $\epsilon > 0$（$\alpha\sigma_X$ の代わり）で実行すれば、有限の分散のみを仮定し、退化したケース $\sigma_X = 0$ も含む同値な形が得られる。

$$P(|X - \mu_X| \geq \epsilon) \leq \frac{\mathrm{Var}(X)}{\epsilon^2}.$$

ダーツの例では $\mu_R = 0.667$、$\sigma_R = 0.236$ である。平均から標準偏差の 2 倍下側は 0.195 であり、2 倍上側はボードの外に出てしまうため、$P(|R - \mu_R| \geq 2\sigma_R) = P(R \leq 0.195) = 0.195^2 \approx 0.038$ となり、チェビシェフの上界である $\frac{1}{4}$ よりはるかに小さい。

ダーツに対して $\frac{1}{4}$ がこれほど緩いことを踏まえると、あらゆる分布に成り立つ上界をもっと鋭くできるだろうか。さらなる仮定なしには不可能であり、わずか 3 点のみに台を持つ分布がそれを証明する。X が値 $a - 2$ と $a + 2$ をそれぞれ確率 q でとり、値 a を残りの確率 $1 - 2q$（ただし $0 < q \leq \frac{1}{2}$）でとるとしよう。これを **3 アトムの例**と呼ぶ。対称性より平均は a であり、分散は $\mathrm{Var}(X) = \mathbb{E}[(X - a)^2] = 4q + 4q = 8q$ となるため、$\sigma_X = 2\sqrt{2q}$ である。ここで $\alpha = 2$ とする。q が小さくなると、しきい値 $2\sigma_X = 4\sqrt{2q}$ もそれに伴って縮むが、外側の 2 つのアトムは距離 2（中心 a から）の位置に固定されたままである。Figure 36.6 に、その結果生じる 3 つの領域を示す。$q > \frac{1}{8}$ のとき、しきい値は 2 を超えるため、外側のアトムはどちらも裾に入らず確率は 0 となる。$q = \frac{1}{8}$ では、しきい値がちょうど 2 に一致して $\sigma_X = 1$ となり、$a \pm 2$ が平均から標準偏差の 2 倍の位置に並ぶ。このとき外側の両アトムが対象となり、合計確率は $2q = \frac{1}{4}$ となってチェビシェフの上界で等号が成立する。$q < \frac{1}{8}$ のときも外側の両アトムはしきい値の外側にあるが、合わせた確率質量 $2q$ は $\frac{1}{4}$ を下回る。

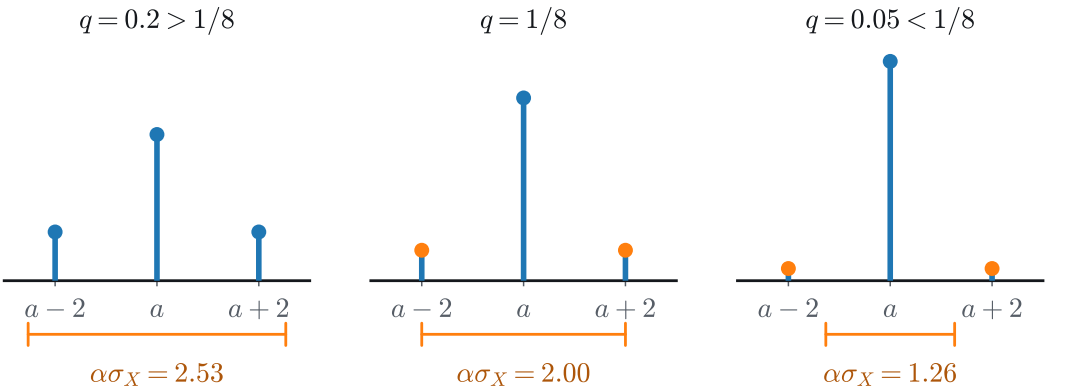


Figure 36.6: 3 アトムの例におけるチェビシェフの上界：確率質量 q、$1 - 2q$、q が点 $a - 2$、a、$a + 2$ の縦棒として描かれており、しきい値距離 $\alpha\sigma_X = 4\sqrt{2q}$（$\alpha = 2$ の場合）が $a - \alpha\sigma_X$ から $a + \alpha\sigma_X$ までの水平ブラケットで示されている。3 つのパネルはそれぞれ $q > 1/8$、$q = 1/8$（上界で等号が達成される場合）、$q < 1/8$ の領域を表す。オレンジ色のアトムはしきい値以上にあるため、事象 $|X - a| \geq \alpha\sigma_X$ に数えられる。

同じ構成ですべての $\alpha \geq 1$ に対しても決着がつく。$q = 1/(2\alpha^2)$（高々 $\frac{1}{2}$ である）を選ぶと $\sigma_X = 2\sqrt{2q} = 2/\alpha$ となるため、しきい値 $\alpha\sigma_X = 2$ が外側のアトムの真上に来て

$P(|X-a| \geq \alpha\sigma_X) = 2q = 1/\alpha^2$ となる。ここで等号が成立するため、$1/\alpha^2$ より小さい定数ではすべての分布に対して成り立たないことが証明される（$\alpha < 1$ のときは上界が 1 を超えるため何も主張しない）。

したがって、チェビシェフの不等式は与えられた分散を持つあらゆる分布のなかでの最悪ケースを捉えている。普段目にする分布に対してはかなり緩い評価になる。平均 μ、標準偏差 σ のガウス分布、すなわち密度 $e^{-(x-\mu)^2/(2\sigma^2)}/(\sigma\sqrt{2\pi})$ を持つ釣鐘曲線（詳細は 36.3.2.3 節で学ぶ）では、標準偏差の 2 倍や 3 倍の外側にある確率はチェビシェフの上界が許す値よりはるかに小さい。

平均からの距離	チェビシェフの上界	ガウス分布
2σ 以上	25%	4.6%
3σ 以上	11%	0.27%

最悪ケースの分布は、3 アトムの例がそうであるように、裾の確率質量をすべて単一の距離に集中させる。典型的な分布では裾が広がりながら急速に薄くなっていく。

36.1.3.2 重い裾

チェビシェフの不等式には有限の分散が必要であり、平均そのものにも $\int |x|\,p(x)\,dx < \infty$ が必要である。驚くべきことに、一見すると滑らかで無害に見える釣鐘曲線でも、この両方の条件が破れることがある。**コーシー分布**は次の密度を持つ。

$$p(x) = \frac{1}{\pi(1+x^2)}.$$

$\int_{-\infty}^{\infty} \frac{dx}{1+x^2} = [\arctan x]_{-\infty}^{\infty} = \pi$ であるため、これは有効な密度である。裾が $e^{-x^2/2}$ のように急落するガウス分布とは異なり、コーシー分布の裾は $1/(\pi x^2)$ のように代数的にしか減衰しない。まず二次モーメント（second moment）が破綻する。$[-L, L]$ 上で積分すると次式が得られる。

$$\int_{-L}^{L} x^2 p(x)\,dx = \frac{2}{\pi}(L - \arctan L),$$

この値は 5.43、62.7、636（それぞれ $L = 10$、100、1000 のとき）となり、被積分関数 $x^2/(\pi(1+x^2))$ が非ゼロの定数 $1/\pi$ に近づくため限りなく増大する。したがって $\mathbb{E}[X^2] = \infty$ となり、コーシー分布は有限の分散を持たない。

平均はさらに微妙な形で破綻する。被積分関数 $x\,p(x)$ は奇関数であるため、対称性から平均は 0 だと結論づけたくなるし、実際 $\int_{-L}^{L} x\,p(x)\,dx = 0$ がすべての L に対して成り立つ。しかし、期待値の定義には絶対収束が必要であり、

$$\int_{-L}^{L} |x|\,p(x)\,dx = \frac{1}{\pi}\log(1+L^2)$$

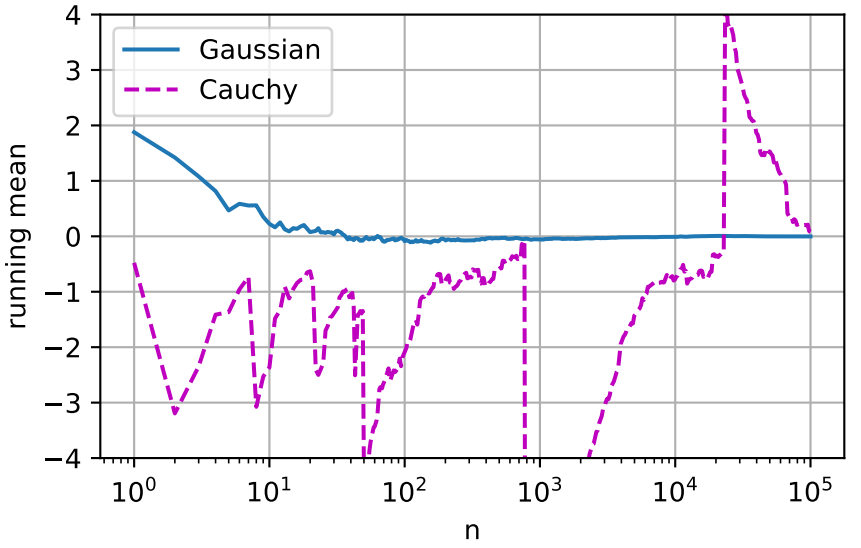
は $L \to \infty$ とするにつれて限りなく増大する。その結果、コーシー分布の平均は存在しない。絶対収束が成り立たない場合、積分の値は極限のとり方に依存してしまう。たとえば非対称な区間 $[-L, 2L]$ 上で積分すると、次のようになる。

$$\int_{-L}^{2L} x\,p(x)\,dx = \frac{1}{2\pi}\log\frac{1+4L^2}{1+L^2} \xrightarrow[L\to\infty]{} \frac{\log 2}{\pi} \approx 0.221.$$

落ち着くべき平均が存在しないとき、独立なコーシー分布からのサンプルを平均するとどうなるだろうか。それでも 1000 個のサンプルを平均すれば、1 回の抽出よりも 0 の近くに落ち着くと期待したくなるかもしれない（1 回の抽出が $[-1,1]$ の外に出る確率は $P(|X|>1) = 1 - \frac{2}{\pi}\arctan 1 = \frac{1}{2}$ である）。以下のセルでは、10^5 個のコーシー乱数の逐次平均を 10^5 個の標準ガウス乱数の逐次平均と並べてプロットし、さらに 1000 組の独立な平均（それぞれ 1000 個のコーシー乱数の平均）のうち 1 より大きく 0 から離れたままのものがいくつあるかを数える。

```
n = 10**5
idx = np.unique(np.logspace(0, 5, 500).astype(int)) - 1  # log-spaced n
running_mean = lambda x: (np.cumsum(x) / np.arange(1, n + 1))[idx]
d2l.plot(idx + 1, [running_mean(rng.standard_normal(n)),
                   running_mean(rng.standard_cauchy(n))],
         'n', 'running mean', legend=['Gaussian', 'Cauchy'],
         xscale='log', ylim=[-4, 4], figsize=(5, 3))
averages = rng.standard_cauchy((1000, 1000)).mean(axis=1)
print(f'averages of 1000 Cauchy draws with |average| > 1: '
      f'{np.mean(np.abs(averages) > 1):.2f}')
```

```
averages of 1000 Cauchy draws with |average| > 1: 0.49
```



ガウス分布の逐次平均は 0 に向かって滑らかに収束する。一方、コーシー分布の逐次平均は決して落ち着かない。たった 1 つの極端なサンプルが引き金となって突然大きく跳ね上がり、オーダー 1 のジャンプがどの n においても起こり得る。平均をとることによる改善はまったくない。1000 個のサンプルの平均のうち約半分が依然として 1 より大きく 0 から離れており、1 回の抽出における割合と変わらない。実際、証明抜きで述べると、独立な n 個の標準コーシー変数の平均は、単一の標準コーシー変数とまったく同じ分布に従う。

任意の指数関数 $e^{-\lambda t}$（$\lambda > 0$）よりも裾の減衰が遅いとき、その分布は**裾が重い**（heavy-tailed）

という。$P(|X| > t)$ が $t^{-\alpha}$ のように減衰するべき乗則の裾（power-law tail）はその代表例であり、すべてのモーメント $\mathbb{E}[|X|^k]$（$k \geq \alpha$）を無限大にする。コーシー分布は $\alpha = 1$ に対応する。ただし、裾が重いからといって必ずしもモーメントが無限大になるとは限らない。36.2.3.1 節で導出する対数正規分布（log-normal distribution）は、裾が重いにもかかわらずすべてのモーメントが有限である。深層学習において重い裾が問題になるのは、その上での標本平均の収束が遅くなるか、あるいはまったく収束しなくなるからである。たとえばミニバッチの勾配ノイズは裾が重いと論じられており (Simsekli et al. 2019)、この仮説は演習問題 5 で検証できる。

本節で扱ったのは一度に 1 つの数値だけであったが、実際のダーツは平面上に刺さり 2 つの座標を持つ。この 2 つの座標は無相関でありながら独立とはほど遠いことがわかる。それを理解するには平面上の密度が必要となるため、36.2 節で取り上げる。

36.1.4 まとめ

連続確率変数は個々の値すべてに確率 0 を割り当てるため、その分布は単位長さあたりの確率である密度を通じて記述する。密度を区間上で積分すればその区間の確率が得られ、累積分布関数はこの面積を左から右へと蓄積して $F' = p$ を満たし、累積分布関数の逆関数をとれば一様ノイズがその分布からのサンプルへと変換される。マルコフの不等式とチェビシェフの不等式は、平均または分散のみを用いて任意の分布の裾を上から抑える。チェビシェフの上界は最悪ケースでは厳密である一方、典型的な分布に対しては緩い。コーシー分布のような裾の重い分布は平均も分散も持たず、そのサンプルを平均しても広がりはまったく縮まらない。何よりもまず、マルコフの不等式を次節以降へ持ち帰ってほしい。36.9 節におけるすべての裾の上界は、この不等式から導かれる。

36.1.5 演習問題

1. **べき乗則の裾**。X が密度 $p(x) = 1/x^2$（$x \geq 1$ のとき）および $p(x) = 0$（それ以外のとき）を持つとする。
    1. p が有効な密度であることを確かめ、$P(X > 2)$ と累積分布関数 $F(x)$ をすべての実数 x に対して計算せよ。
    2. 分位点関数 $F^{-1}(u)$（$u \in (0,1)$）、中央値、および他の 2 つの四分位点を求めよ。さらに、この分布に対する逆変換サンプラ $X = F^{-1}(U)$（Figure 36.4）を書き下せ。
    3. $\mathbb{E}[X] = \infty$ を示せ。勾配クリッピングが勾配ノルムに対して行うように、X を水準 $c > 1$ でクリッピングすると $\min(X, c)$ が得られる。$\mathbb{E}[\min(X, c)]$ を計算し、それが c とともにどのように増大するか述べよ。
    4. コーシー密度 $p_C(x) = 1/(\pi(1 + x^2))$ に対して、本文で挙げた $\int_{-L}^{L} x\, p_C(x)\, dx$ と $\int_{-L}^{2L} x\, p_C(x)\, dx$ の値、および $L \to \infty$ としたときのそれぞれの極限を導出せよ。この 2 つの極限はコーシー分布の平均について何を示しているか。また、(a) の非負の変数 X の平均ではなぜそのような曖昧さが生じ得ないのか。
2. **整流ガウス分布**。$X \sim \mathcal{N}(0,1)$ が標準ガウス密度 $p_X(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ と累積分布関数 $\Phi(x) = \int_{-\infty}^{x} p_X(t)\, dt$ を持つとし、$Y = \max(0, X)$ とする。これは活性化前の値が標準正規分布に従う ReLU ユニットの出力であり、初期化時のユニットの一般的なモデルである。
    1. 累積分布関数 F_Y を Φ を用いて表し、その概形を描け。Y はどこにアトムを持ち、その確率質量はいくらか。Y は離散型か、密度を持つか、それとも混合型か。
    2. Y が密度を持たない理由を説明せよ。狭義単調かつ連続微分可能な変換 $Y = g(X)$ に対しては、36.2.3.1 節の変数変換の公式により密度 $p_Y(y) = p_X(g^{-1}(y))\, |(g^{-1})'(y)|$ が与えられる。ReLU ではこの公式のどの仮定が破れているか。

3. $[-c, c]$（しきい値 $c > 0$）へクリッピングすると（hard-tanh 活性化関数が $c = 1$ で行う操作に相当する）、$C = \min(\max(X, -c), c)$ が得られる。C の累積分布関数を書き下し、しきい値 c として、C のアトムが合わせて 10% の確率を持つような値を求めよ。この方程式には閉形式解がないため、Φ の数表を用いるか、`math.erf` と $\Phi(z) = \frac{1}{2}(1 + \mathrm{erf}(z/\sqrt{2}))$ を用いよ。
4. $\mathbb{E}[Y^2] = \frac{1}{2}$ および $\mathbb{E}[Y] = 1/\sqrt{2\pi}$ を示せ。ヒント：$X^2 = Y^2 + \min(0, X)^2$ と表し、X の対称性を用いよ。4.4.2.2 節の He 初期化は重みの分散を 2 倍にするが、この係数はこれら 2 つのモーメントのどちらを補償しているか。

3. [code] **ロジスティック分布と較正**。ロジスティック分布は累積分布関数 $F(x) = \mathrm{sigmoid}(x) = 1/(1 + e^{-x})$ を持つ。
   1. Equation 36.6 を用いてその密度を求め、さらに分位点関数を求めよ。四分位点はいくらか。
   2. X が連続かつ狭義単調増加な累積分布関数 F を持つとする。$F(X)$ が $[0, 1]$ 上で一様であることを示せ。これは逆変換サンプリングの逆命題である。
   3. ある確率的回帰モデルが、各入力 i に対して目的変数 y_i の予測分布として、正しい位置 μ_i を中心としスケール s を持つロジスティック分布、すなわち累積分布関数 $\mathrm{sigmoid}\big((y - \mu_i)/s\big)$ を出力するとする。実際の誤差 $y_i - \mu_i$ は標準ロジスティック分布に従うものとする。(a) の分位点関数を用いて 10^4 個の誤差を抽出し、$u_i = \mathrm{sigmoid}\big((y_i - \mu_i)/s\big)$ を $s \in \{0.5, 1, 2\}$ に対して計算して、u_i の 3 つのヒストグラムをプロットせよ。予測分布が狭すぎることを示すのはどの形状であり、広すぎることを示すのはどの形状か。このようなヒストグラムは、確率的予測の較正を確かめる標準的な手法である (Gneiting et al. 2007)。

4. **片側チェビシェフ不等式**。●チェビシェフの不等式 Equation 36.15 は両側の裾を一度に抑える。一方、カンテリの不等式は片側の裾を抑えるものであり、X（平均 μ、分散 σ^2）および任意の $t > 0$ に対して次式が成り立つ。

$$P(X - \mu \geq t) \leq \frac{\sigma^2}{\sigma^2 + t^2}. \tag{36.16}$$

   1. 任意の $u \geq 0$ に対して、マルコフの不等式 Equation 36.14 を $(X - \mu + u)^2$ に適用することで $P(X - \mu \geq t) \leq (\sigma^2 + u^2)/(t + u)^2$ を示せ。さらに u を適切に選んでカンテリの不等式を証明せよ。
   2. $t = \alpha\sigma$ としたときの上界を書き下し、Equation 36.15 と $\alpha = 1$ および $\alpha = 2$ において比較せよ。
   3. 与えられた μ、σ、t に対して、平均 μ と分散 σ^2 を持ち、この上界で等号を達成する 2 点上の分布を求めよ。

5. [extended] **重い裾の平均**。●●Simsekli et al. (2019) は、深層ネットワークにおける確率的勾配降下法（SGD）の勾配ノイズは裾が重く、ガウス分布よりも α 安定分布（$\alpha < 2$ で分散が無限大になる）でよく記述されると論じた。この課題では、その領域でモンテカルロ平均がどのように振る舞うかを測定し、続いて小規模なモデルでその前提を検証する。パレート密度 $p(x) = \alpha x^{-\alpha-1}$（$x \geq 1$、$\alpha > 0$）は、問題 1 の密度を $\alpha = 1$ の場合として含んでいる。
   1. 問題 1 の (b) と (c) をこの分布族へ一般化せよ。すなわち、$\mathbb{E}[X^k] < \infty$ となるのがちょうど $k < \alpha$ のときに限ることを示し、平均 μ（$\alpha > 1$ のとき）を計算し、逆変換サンプラを書き下せ。
   2. $1 < \alpha < 2$ とし、平均は有限だが分散は無限大であるとする。独立な n 個のサンプルの最大値が t を超える確率は、$nt^{-\alpha}$ が小さいとき $nt^{-\alpha}$ に近いことを示し、最大

値が典型的に到達するスケール t_n を求めよ。さらに、標本平均の誤差 $|\bar{X}_n - \mu|$ が $n^{1/\alpha - 1}$ のオーダーになること（$n^{-1/2}$ ではなく）を発見的に論じよ。Simsekli らが援用する一般化中心極限定理は、この議論を厳密にするものである。

3. $\alpha \in \{1.5, 2.5, 3\}$ および $n \in \{10, 10^2, \dots, 10^5\}$ に対して、$|\bar{X}_n - \mu|$ の中央値を 400 回の反復実験から推定し、中央値誤差の対数を $\log n$ に対して直線回帰して傾きを求めよ。$\alpha = 1.5$ における傾きを (b) のレート $n^{1/\alpha - 1}$ と比較せよ。また、$\alpha = 2.5$ と $\alpha = 3$ における傾きを、分散が有限のときに成り立つレート $n^{-1/2}$ と比較せよ。

4. 各反復において区間 $\bar{X}_n \pm 1.96\, \hat{\sigma}_n / \sqrt{n}$ を構成せよ。ここで $\hat{\sigma}_n$ は標本標準偏差であり、ガウス分布の平均に対しては μ を約 95% の確率で被覆する。この区間が実際にどれだけの頻度で μ を被覆するか記録せよ。$\alpha = 1.5$ かつ $n = 10^5$ における被覆率がなぜそのようになるのか、どの実行でも $\hat{\sigma}_n$ が有限であることを踏まえて説明せよ。この結果は、ミニバッチ勾配のような裾の重いサンプルから計算されたエラーバーに対して何を示唆するか。

5. Hill 推定量 (Hill 1975) は、上位 k 個のサンプル（m 個の正のサンプル $X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(m)}$ のうち）から裾指数を次式で推定する。

$$\hat{\alpha}_k = \Big(\frac{1}{k} \sum_{i=1}^{k} \log \frac{X_{(i)}}{X_{(k+1)}} \Big)^{-1}. \tag{36.17}$$

10^5 個のパレート乱数に対して $\alpha \in \{1.5, 3\}$ および $k \in \{10^2, 10^3, 10^4\}$ でこの推定量を確かめよ。次に、3.3.5 節のソフトマックス回帰を訓練せよ。訓練後のパラメータにおいて、ミニバッチ勾配とフルバッチ勾配の差の各座標を 1000 個のミニバッチにわたって収集せよ（バッチサイズは 32 と 256 の両方で行う）。それらの絶対値に対する $\hat{\alpha}_k$ を両方のバッチサイズについて k に対してプロットせよ。推定値は 2 を下回るか。また、結論は k の選び方にどれほど依存するか。

36.2 同時分布と変換

36.1 節では、半径 1 のボードに刺さったダーツを、中心からの距離 R（密度は $2r$）という 1 つの数値に還元した。しかし実際には、ダーツは水平座標 X と垂直座標 Y の両方を持つ点に着地する。機械学習でモデル化する量のほとんどは、このように複数の変数の組として現れる。画像は互いに値が依存し合う何千ものピクセルからなり、教師あり学習の問題はどれも入力とターゲットの対を扱う。個々の変数の密度を別々に知るだけでは不十分である。なぜなら、別々の密度からは変数同士がどのように連動するかが何もわからないからである。

本節では 2 つの問いを軸に議論を進める。1 つ目は、ある変数から別の変数について何がわかるかという問いである。ダーツがボードの上端に近い $Y = 0.9$ に着地したとすると、X は中心線から 0.44 以内になければならないため、Y を観測すれば X について確かに何らかの情報が得られる。ところが驚くべきことに、X に対する最良の単一の予測値は依然として 0 であり（Y が何であっても変わらない）、2 つの座標の共分散も 0 であることがわかる。従属性、予測可能性、相関は 3 つの異なる概念であり、これらを区別するには 2 変数の同時密度が必要になる。2 つ目は、ニューラルネットワークの層が入力に作用するときや、サンプラが一様ノイズを変形するときのように、確率変数を変換すると密度がどう変化するかという問いである。ダーツの距離が $R = \sqrt{U}$ として一様変数 U から生成できることはすでに見たとおりであり、平方根をとることで平坦な密度は線形な密度 $2r$ へと変わる。明らかに、変換が密度に及ぼす作用は、単に軸のラベルを付け替えるだけにとどまらない。

本節の 3 つの結果は、本書の後半でとりわけ重要な役割を果たす。全分散の法則 Equation 36.22、最良の二乗誤差予測器としての条件付き平均 Equation 36.23、そして変数変換の公式 Equation 36.31 である。共分散行列と、相関がコサインであるという幾何学的な補足については、初読時には読み飛ばしても差し支えない。

```
rng = np.random.default_rng()
```

36.2.1 同時密度と条件付き密度

連続確率変数の対について知り得るすべての情報は、2 変数の単一の関数に集約される。まずはその関数を定義し、そこから 3 つの本質的な量を取り出す方法を見ていこう。すなわち、一方の座標単独の分布、他方が観測されたもとでの一方の座標の分布、そして他方が与えられたもとでの一方の平均（回帰モデルが推定する対象）である。

36.2.1.1 同時密度と周辺密度

1 次元の密度が短い区間における単位長さあたりの確率を測るのと同様に、その 2 次元版は小さな正方形における単位面積あたりの確率を測る。**同時密度**（joint density）$p_{X,Y}$（確率変数の対 (X, Y) に対する密度）を用いると、一辺 ϵ の (x, y) にある小さな正方形に着地する確率は、おおよそその面積と密度の積として次のように表される。

$$P\big(X \in [x, x+\epsilon],\, Y \in [y, y+\epsilon]\big) \approx \epsilon^2\, p_{X,Y}(x, y).$$

1 次元の場合と同様に、平面の領域 $\mathcal{D}$ をこのような正方形で敷き詰めることで、正式な定義として採用する性質が導かれる。すなわち $p_{X,Y} \geq 0$ かつ $P((X,Y) \in \mathcal{D}) = \iint_{\mathcal{D}} p_{X,Y}(x,y)\,dx\,dy$ であり、したがって平面全体にわたる同時密度の下の全体積は 1 に等しい。n 変数の場合も、n 次元の直方体と n 重積分を用いれば同じ定義が成り立つ。

ダーツの例では、単位円板のどの部分も等しく確からしく、ボードの面積は π であるため、同時密度は円板の内部で $p_{X,Y}(x,y) = 1/\pi$、外部で 0 となる。この密度は円板上では平坦であるにもかかわらず、x の関数と y の関数の積としては書けないことに注意しよう。ボードの円形の縁が 2 つの座標を結びつけているのである。

同時密度がわかっていても、一方の座標単独の分布にしか関心がないことは多い。これを**周辺分布**（marginal distribution）と呼ぶ。細い縦の帯 $X \in [x, x+\epsilon]$ の確率はおおよそ $\epsilon\, p_X(x)$ である。この帯を y に沿って小さな正方形に切り分け、それらの確率を足し合わせると次式を得る。

$$\epsilon\, p_X(x) \approx \sum_i P\big(X \in [x, x+\epsilon],\, Y \in [\epsilon i, \epsilon(i+1)]\big) \approx \sum_i \epsilon^2\, p_{X,Y}(x, \epsilon i).$$

因子 ϵ を 1 つ約分し、残った和をリーマン和（Riemann sum）として y について極限をとれば、次の式が得られる。

$$p_X(x) = \int_{-\infty}^{\infty} p_{X,Y}(x, y)\, dy. \tag{36.18}$$

言葉で言えば、x における周辺密度とは、x における縦の帯に沿って同時密度を積分したものである（Figure 36.7）。すなわち、不要な変数を単に**積分消去**（integrate out）すればよい。ダーツ

の場合、x における縦の帯はボードを $-\sqrt{1-x^2}$ から $\sqrt{1-x^2}$ まで横切るため、周辺密度は次のようになる。

$$p_X(x) = \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \frac{1}{\pi}\, dy = \frac{2}{\pi}\sqrt{1-x^2}, \qquad -1 \le x \le 1,$$

これは半円を描く。簡単に確かめておくと、$\int_{-1}^{1} \sqrt{1-x^2}\, dx$ は単位円板の半分の面積 $\pi/2$ であるから、p_X を積分すると確かに 1 になる。同時密度が平坦であっても、周辺密度は曲線になる。ボードの中央を通る縦の帯のほうが、縁の近くを通る帯よりも長いからである。

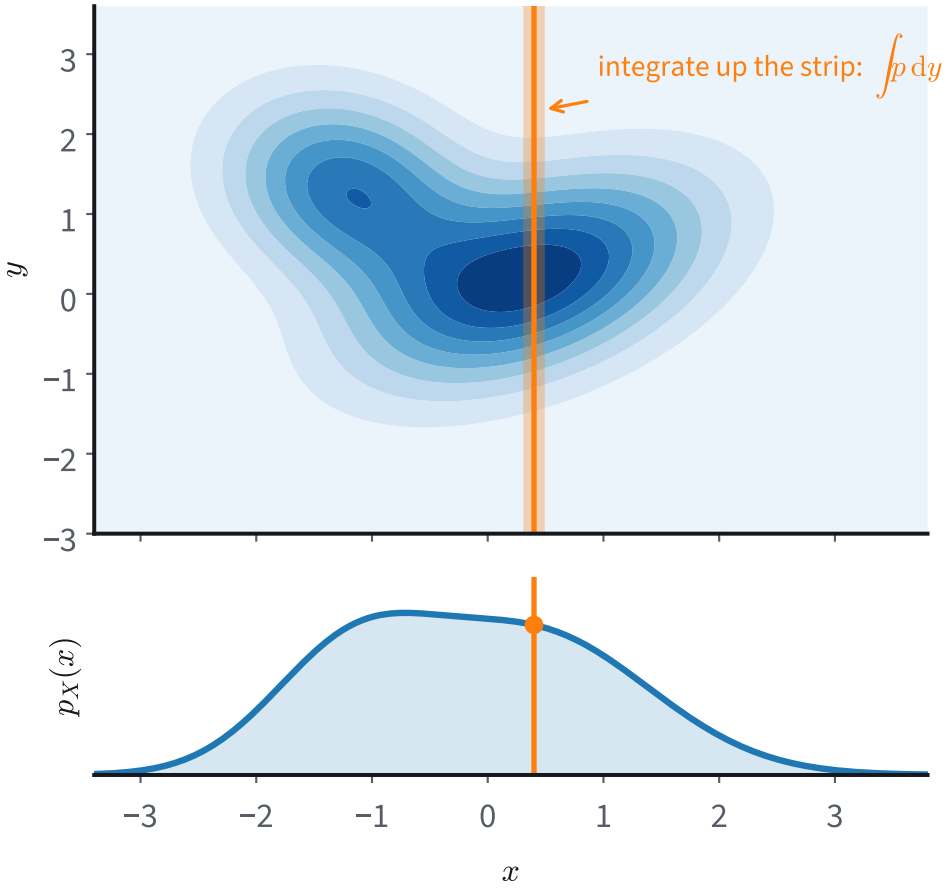


Figure 36.7: 等高線で示した連続同時密度 $p_{X,Y}(x,y)$ と、位置 x で強調表示した 1 本の縦の帯。この帯に沿ってすべての y にわたり同時密度を積分すると、下の周辺密度 $p_X(x)$（x 軸に沿ってプロット）の高さが得られる。この帯を x 全体に走査すると周辺密度全体が描かれる。

36.2.1.2 条件付き密度と独立性

$Y = y$ であることが観測されたとき、X の密度はどう変わるべきだろうか。離散変数であれば、答えは単に $P(X = x \mid Y = y) = P(X = x, Y = y)/P(Y = y)$ である。しかし連続密度の

場合、この式は $P(Y = y) = 0$ のため 0 を 0 で割ることになってしまう。解決策は、そもそも密度を定義したときと同じ極限の議論である。正の確率を持つ細い水平な帯 $Y \in [y, y + \epsilon]$ で条件付け、その帯の幅を縮めていくのである。分子に正方形近似を、分母に 1 変数の規則を適用すると、次のようになる。

$$P(X \in [x, x + \epsilon] \mid Y \in [y, y + \epsilon]) \approx \frac{\epsilon^2 \, p_{X,Y}(x, y)}{\epsilon \, p_Y(y)} = \epsilon \, \frac{p_{X,Y}(x, y)}{p_Y(y)}.$$

この帯の内部では、短い x 区間の確率はその長さ ϵ と $p_{X,Y}(x, y)/p_Y(y)$ の積になる。この比は、X について $Y = y$ が与えられたときの単位長さあたりの確率にあたるため、**条件付き密度**（conditional density）を次のように定義する。

$$p_{X|Y}(x \mid y) = \frac{p_{X,Y}(x, y)}{p_Y(y)}, \qquad p_Y(y) > 0. \tag{36.19}$$

固定した任意の y（ただし $p_Y(y) > 0$）に対して、これは x に関する正当な密度である。非負であり、周辺化の規則 Equation 36.18 によって積分値が $p_Y(y)/p_Y(y) = 1$ となるからである。幾何学的には、同時密度を水平線 $Y = y$ に沿ってスライスし、その断面の面積が 1 になるようスケールし直している。ダーツの場合、水平線 $Y = y$ はボードを弦 $[-\sqrt{1 - y^2}, \sqrt{1 - y^2}]$ 上で横切るため、条件付き密度は次のようになる。

$$p_{X|Y}(x \mid y) = \frac{1/\pi}{(2/\pi)\sqrt{1 - y^2}} = \frac{1}{2\sqrt{1 - y^2}} \qquad \text{for } |x| \le \sqrt{1 - y^2}.$$

$Y = y$ が与えられたもとで、水平位置 X は高さ y の弦上で一様に分布する。この弦は $|y|$ が大きくなるにつれて短くなるため、Y を観測すると X の分布は変化する。

Equation 36.19 を変形すると、**連鎖律**（chain rule）$p_{X,Y}(x, y) = p_{X|Y}(x \mid y) \, p_Y(y)$ が得られる。同時密度を $p_{X|Y} \, p_Y = p_{Y|X} \, p_X$ と 2 通りに書き、$p_{X|Y}$ について解くと、次の**密度に対するベイズの定理**（Bayes' rule）が得られる。

$$p_{X|Y}(x \mid y) = \frac{p_{Y|X}(y \mid x) \, p_X(x)}{p_Y(y)}. \tag{36.20}$$

1.6 節で離散変数に対してベイズの定理が果たしたのと同様に、この恒等式は、Y が X からどのように生成されるかを表すモデルを、X の（Y を観測した後の）分布へと変換する。

Y を観測しても X について新しい情報が何も得られず、$p_{X|Y}(x \mid y) = p_X(x)$ が $p_Y(y) > 0$ である限り常に成り立つ特別な場合、2 つの変数は**独立**（independent）であるといい、$X \perp Y$ と書く。連鎖律により、独立性は同時密度が $p_{X,Y}(x, y) = p_X(x) \, p_Y(y)$ のように因数分解できることと同値である（確率 0 の集合を除く）。一方から他方を導くには、$p_Y(y)$ を掛けるか、それで割ればよい。

直感的には、独立性とは、X の条件付き密度が位置も形状も同じ x の関数となり、どの高さ y でも変わらないことを意味する。単位正方形上、すなわち $p_{X,Y}(x, y) = 4xy$（$[0, 1]^2$ 上）という同時密度を考えてみよう。その周辺密度は $p_Y(y) = \int_0^1 4xy \, dx = 2y$ であり、これより $p_{X|Y}(x \mid y) = 4xy/(2y) = 2x$ となって y によらないため、$X \perp Y$ である。ダーツはその逆の例であり、条件付き密度はどの y でも一様であるが、その台となる弦の長さは y に依存する。

Figure 36.8 が示すように、形状が同じだけでは独立性には十分ではない。相関のある対では、3 つの高さにおける条件付き密度は同じ形状を共有するものの中心がずれているため従属であり、同時密度が因数分解できるときに限って 3 つのスライスがすべて一致する。

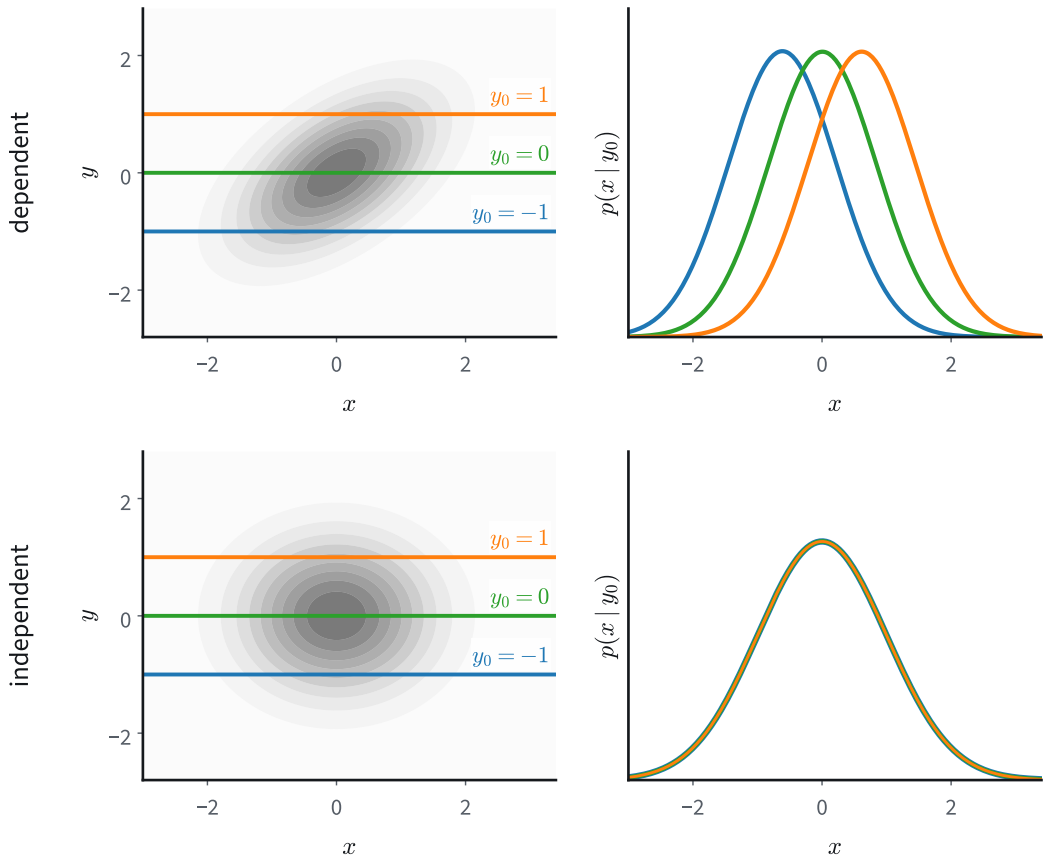


Figure 36.8: スライスとしての条件付け。左：同時密度と $y_0 = -1, 0, 1$ における 3 本の水平スライス。右：各スライスを面積 1 に再正規化したものが条件付き密度 $p(x \mid y_0)$ である。上段：相関のある対。3 つの条件付き密度は同じ形状だが中心がずれているため、条件付き密度は y_0 に依存し、対は従属である。下段：同時密度が因数分解できる対。3 つの条件付き密度が一致し、これが独立性である。

独立性からは、和の密度に対する簡潔な公式も得られる。$X \perp Y$ のとき、対 $(X, X+Y)$ は同時密度 $p_X(x)\,p_Y(s-x)$ を点 (x, s) で持ち、x を Equation 36.18 によって積分消去すると**畳み込み**（convolution）$p_{X+Y}(s) = \int p_X(x)\,p_Y(s-x)\,dx$ が得られる。

36.2.1.3 条件付き期待値

回帰モデルは入力からターゲットを予測する。前項との整合性を保つため、ここでは X をターゲット、Y を観測される入力とする（本編の他章では逆の記号割り当てを用いる）。二乗損失による訓練では、関数 h のうち $\mathbb{E}[(X - h(Y))^2]$ をできるだけ小さくするものを探索する。どの関数が最良だろうか。任意の細い帯 $Y \approx y$ の中で、モデルは広がりを持つターゲットの候補全体に対して 1 つの数値を予測しなければならない。任意の確率変数 X に対し、平均二乗距離の意味で最も近い定数 c はその平均 μ_X である。実際、μ_X の周りで二乗を展開すると

$\mathbb{E}[(X-c)^2] = \mathrm{Var}(X) + (\mu_X - c)^2$ となる。このことは、各帯の内部における X の平均を予測すべきであることを示唆している。そこで、X について $Y=y$ が与えられたときの**条件付き期待値**（conditional expectation）を、条件付き密度の平均として次のように定義する。

$$\mathbb{E}[X \mid Y{=}y] = \int x\, p_{X|Y}(x \mid y)\, dx. \tag{36.21}$$

これは単に、X の値を $Y=y$ となるすべての結果にわたって平均したものである。y を動かすと $\mathbb{E}[X \mid Y{=}y]$ は y の関数を描き、そこに確率変数 Y を代入すると、$\mathbb{E}[X \mid Y]$ という Y のみに依存する確率変数になる。同じ帯内での平均により、$\mathbb{E}[g(X,Y) \mid Y{=}y] = \int g(x,y)\, p_{X|Y}(x \mid y)\, dx$ が任意の関数 g に対して定義される。因子 $h(Y)$ は各帯の内部では定数であるため、条件付き期待値の外へそのままくくり出せる。

$$\mathbb{E}[h(Y)\, g(X,Y) \mid Y] = h(Y)\, \mathbb{E}[g(X,Y) \mid Y].$$

同様に、1 つの帯の内部における X の広がりは**条件付き分散**（conditional variance）$\mathrm{Var}(X \mid Y) = \mathbb{E}[X^2 \mid Y] - (\mathbb{E}[X \mid Y])^2$ であり、これは 36.1 節の分散の計算公式を各帯の内部に適用したものにほかならない。

直感的には、帯ごとの平均をさらに平均すれば全体の平均に戻り、全体の分散は帯内部のばらつきと帯間のばらつきの和に分解できるはずである。実際そのどちらも成り立ち、とりわけ 2 つ目の公式は非常に有用である。

命題（全期待値と全分散の法則） *X, Y が同時密度を持ち、X が有限の分散を持つとする（離散変数の組についても、積分を和に置き換えれば同じ恒等式が成り立つ）。このとき、次式が成り立つ。*

$$\begin{aligned} \mathbb{E}[X] &= \mathbb{E}\big[\mathbb{E}[X \mid Y]\big], \\ \mathrm{Var}(X) &= \mathbb{E}\big[\mathrm{Var}(X \mid Y)\big] + \mathrm{Var}\big(\mathbb{E}[X \mid Y]\big). \end{aligned} \tag{36.22}$$

証明 **塔性質**（tower property）として知られる 1 つ目の恒等式については、内側の期待値を書き下し、連鎖律 $p_{X|Y}(x \mid y)\, p_Y(y) = p_{X,Y}(x,y)$ を適用して帯を再び同時密度へとまとめればよい。

$$\mathbb{E}\big[\mathbb{E}[X \mid Y]\big] = \int \Big(\int x\, p_{X|Y}(x \mid y)\, dx \Big) p_Y(y)\, dy = \iint x\, p_{X,Y}(x,y)\, dx\, dy = \mathbb{E}[X].$$

条件付き密度が定義されない y（すなわち $p_Y(y) = 0$ となる値）には Y が確率 0 でしか着地しないため、両辺のどちらにも寄与しない。$g(x,y)$ を x の代わりに用いて同じ計算を行えば、$\mathbb{E}\big[\mathbb{E}[g(X,Y) \mid Y]\big] = \mathbb{E}[g(X,Y)]$ が得られる。2 つ目の恒等式については、右辺の両項を展開する。塔性質を X^2 に適用すると次式を得る。

$$\begin{aligned} \mathbb{E}\big[\mathrm{Var}(X \mid Y)\big] &= \mathbb{E}\big[\mathbb{E}[X^2 \mid Y]\big] - \mathbb{E}\big[(\mathbb{E}[X \mid Y])^2\big] \\ &= \mathbb{E}[X^2] - \mathbb{E}\big[(\mathbb{E}[X \mid Y])^2\big]. \end{aligned}$$

さらに $\mathrm{Var}(Z) = \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2$（36.1 節）を確率変数 $\mathbb{E}[X \mid Y]$（塔性質によりその平均は $\mathbb{E}[X]$ である）に適用すると、次式を得る。

$$\mathrm{Var}\big(\mathbb{E}[X \mid Y]\big) = \mathbb{E}\big[(\mathbb{E}[X \mid Y])^2\big] - \mathbb{E}[X]^2.$$

これら 2 式を足し合わせると、中央の項 $\mathbb{E}[(\mathbb{E}[X \mid Y])^2]$ が打ち消し合い、$\mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathrm{Var}(X)$ が残る。■

2 つ目の恒等式は、右辺に現れる E、V、V、E の並びにちなんで、しばしば**イブの法則**（Eve's law）と呼ばれる。この式は X の全分散を 2 つの部分に分解する。1 つは各帯の内部における分散の平均であり、これは Y を知っても決して取り除けない。もう 1 つは帯ごとの平均それ自体の分散であり、これが Y によって説明される部分である。統計学では、データの変動をグループに帰属させるために、この分解を分散分析（analysis of variance）という名で用いる。次に示すとおり、同じ分解によって先ほどの回帰の問いにも決着がつく。

命題（条件付き平均は最良の二乗誤差予測器である） **X が有限の分散を持ち、h を $\mathbb{E}[h(Y)^2] < \infty$ を満たす任意の関数とする。このとき、次式が成り立つ。**

$$\mathbb{E}\big[(X - h(Y))^2\big] = \mathbb{E}\big[\mathrm{Var}(X \mid Y)\big] + \mathbb{E}\big[(\mathbb{E}[X \mid Y] - h(Y))^2\big]. \tag{36.23}$$

証明 $m(Y) = \mathbb{E}[X \mid Y]$ とおき、予測誤差を $X - h(Y) = (X - m(Y)) + (m(Y) - h(Y))$ の 2 つの部分に分解する。二乗すると 2 つの二乗項と 1 つの交差項が現れるが、注意を要するのは交差項だけである。1 つ目の二乗項の期待値は、塔性質により $\mathbb{E}[\mathrm{Var}(X \mid Y)]$ となる。交差項 $2(X - m(Y))(m(Y) - h(Y))$ の条件付き平均は 0 である。なぜなら、因子 $m(Y) - h(Y)$ は Y のみに依存するため帯内の平均からくくり出せ、残りが $\mathbb{E}[X - m(Y) \mid Y] = m(Y) - m(Y) = 0$ となるからである。したがって塔性質により、その全体の平均も 0 になる。■

右辺の 2 つの項に注目しよう。第 1 項は h にまったく依存せず、一方で第 2 項は非負であり $h(Y) = \mathbb{E}[X \mid Y]$ のときに 0 になる。したがって、条件付き平均が二乗誤差のもとでの最適な予測器であり、いかなる予測器も帯内分散の平均より小さい期待二乗誤差を達成することはできない。ゆえに、二乗損失で訓練されたネットワークは $\mathbb{E}[\text{target} \mid \text{input}]$ を推定しているのであり、これは 2.1 節におけるモデルの解釈（そこでは $\mathbb{E}[y \mid \mathbf{x}]$ と表記した）そのものである。

2 つの例でこの分解を具体的に見てみよう。まずダーツの例に戻り、X を Y から予測してみる。$Y = y$ が与えられたとき、水平座標 X は 0 に関して対称な弦上で一様に分布するため、$\mathbb{E}[X \mid Y = y] = 0$ となる。半幅 w の区間上の一様変数は分散 $w^2/3$ を持つため、$\mathrm{Var}(X \mid Y = y) = (1 - y^2)/3$ となる。対称性により $\mathbb{E}[X^2] = \mathbb{E}[Y^2] = \tfrac{1}{2}\mathbb{E}[R^2] = \tfrac{1}{4}$ であるから、イブの法則より次式が得られる。

$$\mathrm{Var}(X) = \mathbb{E}\Big[\frac{1 - Y^2}{3}\Big] + \mathrm{Var}(0) = \frac{1 - 1/4}{3} + 0 = \frac{1}{4}.$$

分散はすべて説明不能な項に残る。Y を知ることで X の範囲は狭まるものの中心はまったく動かないため、X を Y から予測する最良の二乗誤差予測器は定数 0 であり、これは X と Y が従属であっても変わらない。

2 つ目の例として、2 値ラベル Z を、値 1（確率 0.2）または値 0（確率 0.8）をとるように抽出する。$Z = 1$ のときは X を正規分布 $\mathcal{N}(3, 1)$（平均 3、分散 1）から抽出し、$Z = 0$ のときは X を $\mathcal{N}(-1, 1)$ から抽出する。この混合分布は、Figure 36.1 の 2 つの峰を持つ密度を生み出す。全体の平均は $0.2 \cdot 3 + 0.8 \cdot (-1) = -0.2$ である。各成分の内部での分散は 1 であるため、$\mathbb{E}[\mathrm{Var}(X \mid Z)] = 1$ となる。成分間では、$\mathbb{E}[X \mid Z]$ は値 3 と -1 をとり、その分散は $0.2 \cdot 0.8 \cdot (3 - (-1))^2 = 2.56$ である。したがって全分散は 3.56 であり、ラベル Z はそのうちの $2.56/3.56 = 72\%$ を説明する。100 万個のサンプルを抽出すれば、経験的な成分内分散と成

分間分散の項はそれぞれ 1 と 2.56 の近くに落ち着くだろう。コードを実行する前に考えてみてほしい。両者の和は標本分散とおおよそ一致するだけだろうか、それとも厳密に一致するだろうか。

```
n = 10**6
z = rng.uniform(size=n) < 0.2                          # component label Z
x = np.where(z, rng.normal(3, 1, n), rng.normal(-1, 1, n))
w = np.array([np.mean(~z), np.mean(z)])                # P(Z=0), P(Z=1)
cond_mean = np.array([x[~z].mean(), x[z].mean()])      # E[X | Z]
within = w @ np.array([x[~z].var(), x[z].var()])       # E[Var(X | Z)]
between = w @ (cond_mean - x.mean())**2                # Var(E[X | Z])
print(f'Var(X) {x.var():.3f} = within {within:.3f} + between {between:.3f}')
```

```
Var(X) 3.561 = within 1.000 + between 2.561
```

厳密に一致する。それぞれの項は理論値（1 と 2.56）の近くに落ち着き、小数第 3 位にサンプリングノイズが現れるが、両者の和はどの実行でも出力の最終桁の丸め誤差を除いて標本分散に一致する。イブの法則は、経験的なグループの割合とグループ内分散に確率と条件付き分散の役割を担わせることで、標本そのものに対しても厳密な恒等式として成り立つのである。

36.2.2 共分散と相関

条件付き平均は、X の平均が Y に伴ってどう動くかを記述するため、それ自体が 1 つの関数になる。これに対し、単一の要約値で済ませたいことも多い。すなわち、X と Y はともに大きくなる傾向があるのか、それとも一方が大きいときにもう一方は小さくなる傾向があるのか、という指標である。その数値こそが共分散である。ただし共分散には、本節の残りで対処または明らかにする 2 つの限界がある。両変数の単位を引き継いでしまうこと（これは相関によって解消される）と、直線的な傾向しか捉えられないこと（これはどのようなスケール変換でも解消できない）である。

36.2.2.1 共分散

1.6 節では、**共分散**（covariance）を、それぞれの平均からの偏差の積の期待値として定義した。離散変数 X, Y が値 (x_i, y_j) を確率 p_{ij} でとるとき、共分散は次の式で与えられる。

$$\sigma_{XY} = \operatorname{Cov}(X,Y) = \sum_{i,j} (x_i - \mu_X)(y_j - \mu_Y)\, p_{ij} = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y], \tag{36.24}$$

ここで 2 番目の形は、分散の計算公式と同様に積を展開することで得られる。特に、$\operatorname{Cov}(X,X) = \operatorname{Var}(X)$ である。本節を通じて両変数の分散は有限であると仮定する。このとき $|xy| \le (x^2 + y^2)/2$ であるから、XY の期待値の存在が保証される。

具体例として、$X \in \{1, 3\}$、$Y \in \{-1, 3\}$ とし、同時確率が次で与えられるとする。

$$\begin{aligned} P(X{=}1, Y{=}{-1}) &= \tfrac{\theta}{2}, & P(X{=}1, Y{=}3) &= \tfrac{1-\theta}{2}, \\ P(X{=}3, Y{=}{-1}) &= \tfrac{1-\theta}{2}, & P(X{=}3, Y{=}3) &= \tfrac{\theta}{2}, \end{aligned}$$

ここで $\theta \in [0,1]$ はパラメータである。これを **4 セルの例**と呼ぶことにする。各変数はそれぞれの 2 つの値を確率 $\tfrac{1}{2}$ でとるため、$\mu_X = 2$、$\mu_Y = 1$ である。積の期待値は次のように計算できる。

$$\mathbb{E}[XY] = \tfrac{1}{2}(-\theta + 3(1-\theta) - 3(1-\theta) + 9\theta) = 4\theta,$$

これより $\mathrm{Cov}(X,Y) = 4\theta - 2$ が得られる。すなわち、共分散は $+2$（$\theta = 1$ のとき）、-2（$\theta = 0$ のとき）、そして 0（$\theta = \tfrac{1}{2}$ のとき）となる。

同時密度を持つ連続変数の場合、和は二重積分に置き換わる。

$$\sigma_{XY} = \iint (x - \mu_X)(y - \mu_Y)\, p_{X,Y}(x,y)\, dx\, dy, \tag{36.25}$$

すなわち、同時密度にわたって偏差の積を平均する。ダーツの例では、両座標の平均は 0 であり、$\mathrm{Cov}(X,Y) = \mathbb{E}[XY] = \frac{1}{\pi}\iint_{\text{disk}} xy\, dx\, dy = 0$ となる。被積分関数が x に関して奇関数である一方、円板が対称だからである。それにもかかわらず、弦が縮むことから見たように X と Y は従属である。共分散は線形な傾向しか見ないが、ダーツの円形の従属関係には線形な傾向がないため、共分散が 0 であっても独立性は導かれない（1.6 節では離散の例として $Y = X^2$ を挙げた）。Figure 36.9 は円板上でこれら両方の事実を示している。

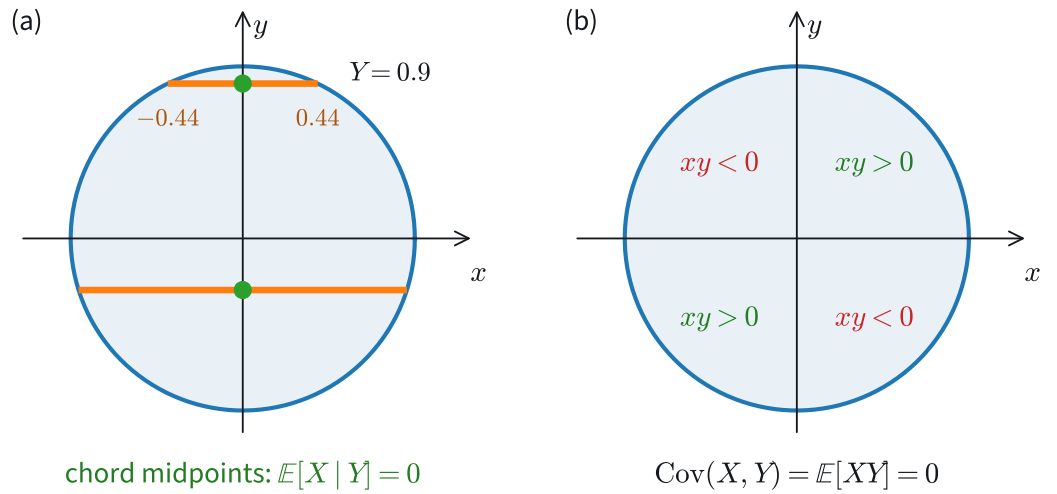


Figure 36.9: ダーツの座標は従属であるが、平均においては予測不能であり無相関である。(a) 高さ $Y = 0.9$ の弦は X を $|X| \le 0.44$ に閉じ込めるのに対し、中央付近の弦ははるかに長いため、X は Y に従属している。どの弦も中心は $x = 0$（緑の点）にあるため、$\mathbb{E}[X \mid Y] = 0$ となる。(b) 積 xy は 2 つの象限で正、残り 2 つの象限で負となり、いずれも面積が等しいため $\mathrm{Cov}(X,Y) = 0$ となる。

共分散は両変数の単位も引き継ぐ。Figure 36.10 の下段は上段とまったく同じサンプルをプロットしたものだが、Y の単位をドルではなくセントで表している。どの点群の形状もまったく変わらないにもかかわらず、標本共分散はすべて 100 倍に跳ね上がる。単位の情報がなければ共分散で意味を持つのは符号だけであり、以下で導入する相関係数は単位を完全に取り除く。

共分散は各引数に関して線形である。定数を足しても変数とその平均が同じだけ平行移動するため偏差からは消え、定数倍の因子はそのまま外に出るため $\mathrm{Cov}(cX + d, Y) = c\,\mathrm{Cov}(X,Y)$ となる。同様に、和の偏差は偏差の和であるから、期待値の線形性により $\mathrm{Cov}(X + Z, Y) = \mathrm{Cov}(X,Y) + \mathrm{Cov}(Z,Y)$ が成り立ち、第 2 引数についても同じ規則が成り立つ。さらに、独

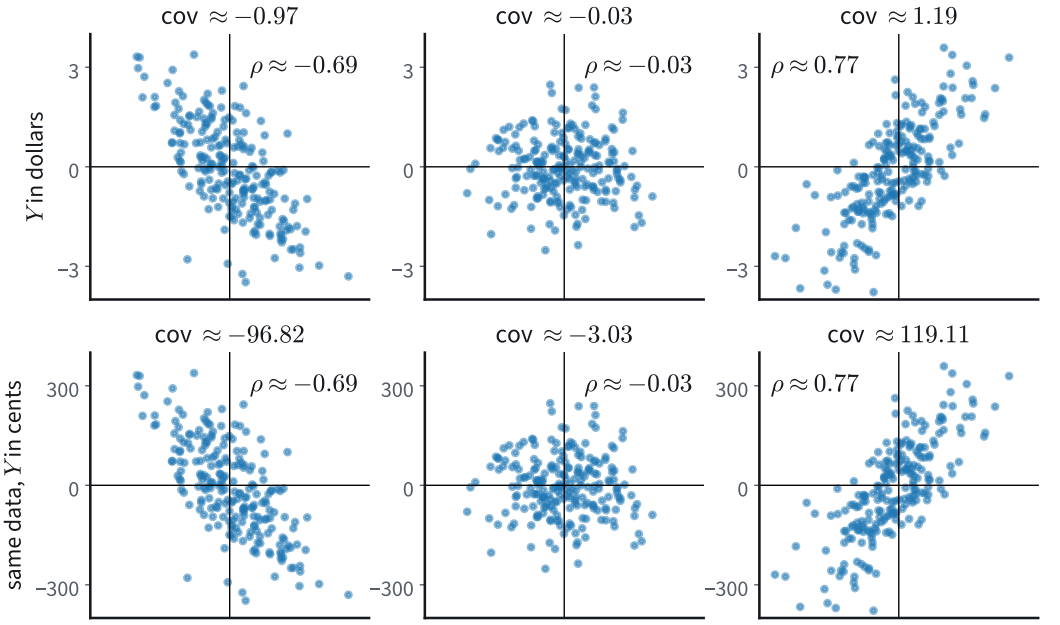


Figure 36.10: 上段：共分散を負、0、正（左から右）に調整した (X, Y) の散布図（Y の単位はドル）。負の共分散では点群が右下がりに傾き、0 では丸くなり、正では右上がりに傾く。下段：まったく同じサンプルの Y をセント単位で表したもの。標本共分散がすべて 100 倍になる一方、各パネルの相関 ρ（上部隅）は変わらない。共分散の大きさは単位を持ち、スケールに依存しないのはその符号だけである。

立変数の共分散は常に 0 である。同時密度が因数分解できるときは常に、$\mathbb{E}[XY]$ の二重積分が $\mathbb{E}[X]\,\mathbb{E}[Y]$ へと分解するからである。

共分散を用いると、和の分散の公式も完成する。$\tilde{X} = X - \mu_X$、$\tilde{Y} = Y - \mu_Y$ とおく。線形性により $X + Y$ の平均は $\mu_X + \mu_Y$ となり、したがって偏差は $\tilde{X} + \tilde{Y}$ となる。$\mathbb{E}[(\tilde{X} + \tilde{Y})^2] = \mathbb{E}[\tilde{X}^2] + 2\,\mathbb{E}[\tilde{X}\tilde{Y}] + \mathbb{E}[\tilde{Y}^2]$ を展開すると、次の式が得られる。

$$\operatorname{Var}(X + Y) = \operatorname{Var}(X) + \operatorname{Var}(Y) + 2\operatorname{Cov}(X, Y). \tag{36.26}$$

X と Y が独立であるとき、共分散は消え、分散は単純に足し合わされる。

この公式から導かれる最も重要な帰結は、多数の抽出値を平均したときに何が起きるかである。n 個の偏差の和の二乗を同様に展開すると、$\operatorname{Var}(\sum_i X_i) = \sum_i \sum_j \operatorname{Cov}(X_i, X_j)$ が得られる。いま $X_1, \dots, X_n$ が独立で、共通の分散 σ^2 を持つとする。すべての交差項が消えるため和の分散は $n\sigma^2$ となり、さらに $c = 1/n$ 倍することで分散は n^2 で割られる。したがって、平均 $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ の分散は次のようになる。

$$\operatorname{Var}(\bar{X}_n) = \frac{1}{n^2} \cdot n\sigma^2 = \frac{\sigma^2}{n}. \tag{36.27}$$

独立な n 個の抽出値を平均すると、分散は n 分の 1 に、標準偏差は $\sqrt{n}$ 分の 1 になる。つまり、典型的な誤差を半分にするには 4 倍の抽出数が必要になる。この $\sigma/\sqrt{n}$ のスケーリングは、シミュレーションで生成したサンプルの平均で期待値を近似するあらゆる**モンテカルロ**推定の基礎となっている（36.5.1.7 節）。ダーツの例では、独立に n 回投げたときの平均距離の分散は $1/(18n)$ であるため、556 本投げれば標準偏差を 0.01 まで下げられる。なお、$\sigma/\sqrt{n}$ の収束率には有限の分散が必要である。一方、$\bar{X}_n$ が平均に収束することを主張する**大数の法則**（law of large numbers）そのものに必要なのは、有限の平均だけである（36.5 節）。コーシー分布はそのどちらも持たず、n をどれほど大きくしても抽出値の平均が落ち着くことはない。

36.2.2.2 相関

4 セルの例における共分散 $4\theta - 2$ が大きいかどうかは、X と Y の単位に依存する。単位を取り除くには、まったく同じ単位を持つ 2 つの標準偏差で割ればよい。**相関係数**（correlation coefficient）は次式で定義される。

$$\rho(X, Y) = \frac{\operatorname{Cov}(X, Y)}{\sigma_X \sigma_Y}. \tag{36.28}$$

1.6 節では、ρ が常に -1 と 1 の間に収まることを述べた。この評価を証明し、等号がいつ成り立つかを正確に確かめよう。

命題（相関係数の範囲）有限かつ非ゼロの分散を持つ任意の X, Y に対して、$-1 \le \rho(X, Y) \le 1$ が成り立つ。等号は、$Y = cX + d$ が定数 $c \neq 0$ と d に対して確率 1 で成り立つとき、かつそのときに限り成立し、$\rho = 1$（$c > 0$ のとき）、$\rho = -1$（$c < 0$ のとき）となる。

証明分散は決して負にならないため、$\operatorname{Var}(tX + Y) \ge 0$ がすべての実数 t に対して成り立つ。和の分散の公式とアフィンスケーリングの公式を適用すると、これは t に関する二次関数になる。

$$\operatorname{Var}(tX + Y) = \operatorname{Var}(X)\,t^2 + 2\operatorname{Cov}(X, Y)\,t + \operatorname{Var}(Y) \;\ge\; 0 \quad \text{for every } t.$$

最高次の係数が正で決して 0 を下回らない二次関数は実数解を高々 1 つしか持てないため、その判別式は非正でなければならない。

$$\big(2\,\mathrm{Cov}(X,Y)\big)^2 - 4\,\mathrm{Var}(X)\,\mathrm{Var}(Y) \le 0, \qquad \text{that is} \qquad \mathrm{Cov}(X,Y)^2 \le \mathrm{Var}(X)\,\mathrm{Var}(Y).$$

両辺を $\sigma_X^2\sigma_Y^2$ で割ると $\rho^2 \le 1$ が得られる。$\rho^2 = 1$ のときは判別式が 0 となり、この二次関数は重解 $t^\star$ を持って $\mathrm{Var}(t^\star X + Y) = 0$ を満たす。分散が 0 の確率変数は確率 1 で定数であるため、確率 1 で $Y = -t^\star X + d$ となり、また $t^\star \neq 0$ である（$\mathrm{Var}(Y) > 0$ のため）。逆に、$Y = cX + d$（$c \neq 0$）であれば $\mathrm{Cov}(X,Y) = c\,\mathrm{Var}(X)$ かつ $\sigma_Y = |c|\sigma_X$ となるため、$\rho = c/|c| = \pm 1$ が得られる。■

相関は X と Y がどれだけ密接に直線に沿っているかをその直線の傾きの符号とともに測るが、傾きの大きさは無視する。実際、$Y = 0.5X$ と $Y = 2X$ はどちらも $\rho = 1$ である。4 セルの例では $\sigma_X = 1$、$\sigma_Y = 2$ であるため、$\rho = (4\theta - 2)/2 = 2\theta - 1$ となり、-1 から 1 まで（θ が 0 から 1 まで動くにつれて）変化する。Figure 36.11 は ρ の 2 つの死角を示している。同じ相関を持ちながら傾きが異なる 2 つの点群と、放物線上に並びながら相関が 0 になる点群である。

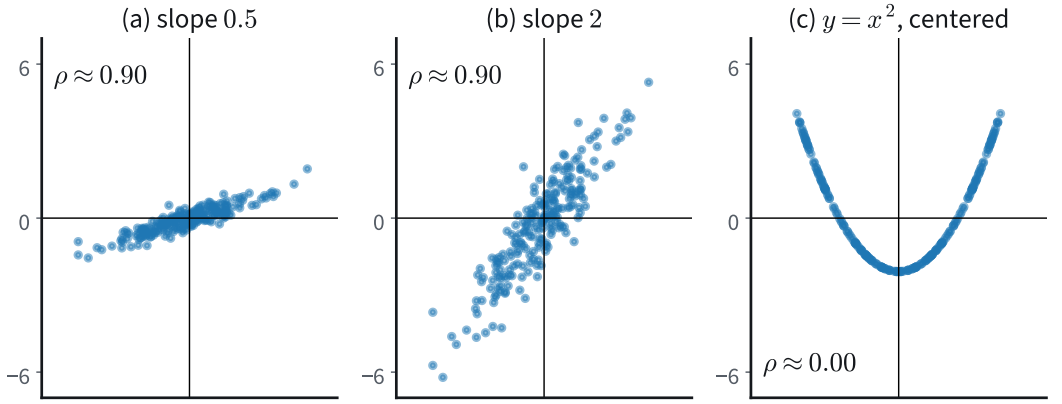


Figure 36.11: 相関は線形関係の密着度と符号を測り、傾きは測らない。(a) と (b) はともに $\rho \approx 0.9$ だが、傾きはそれぞれ 0.5 と 2 である。また非線形な関係も見逃す。(c) では Y は X の関数であるが、$\rho \approx 0$ である。

上の証明がコーシー・シュワルツの不等式 $|\mathbf{u}^\top \mathbf{v}| \le \|\mathbf{u}\|\,\|\mathbf{v}\|$ の古典的な証明そのものであることに気づくだろう。これは偶然ではなく、相関は文字通りコサインなのである。平均 0 で有限の分散を持つ確率変数の空間では、積の期待値が内積として働く（確率 1 で一致する 2 つの変数は同一視する）。中心化された変数 $\tilde{X}$ と $\tilde{Y}$ に対して、この内積は $\langle \tilde{X}, \tilde{Y} \rangle = \mathbb{E}[\tilde{X}\tilde{Y}] = \mathrm{Cov}(X,Y)$ であり、誘導される長さは標準偏差 $\|\tilde{X}\| = \sqrt{\mathbb{E}[\tilde{X}^2]} = \sigma_X$、そして相関はコサイン $\langle \tilde{X}, \tilde{Y} \rangle / (\|\tilde{X}\|\,\|\tilde{Y}\|)$ となる。幾何学的には、無相関な変数は直交しており、$\rho = \pm 1$ はそれらが平行であることを意味する。

36.2.2.3 共分散行列

n 個の変数を同時に扱う場合、すべての変数の対に共分散が存在するため、それらを行列にまとめるのが自然である。確率ベクトル $\mathbf{X} = (X_1, \dots, X_n)^\top$（平均は μ）に対し、**共分散行列**（covariance matrix）は偏差ベクトル自身の外積の期待値として次のように定義される。

$$\Sigma = \mathbb{E}\big[(\mathbf{X}-\mu)(\mathbf{X}-\mu)^{\top}\big], \qquad \Sigma_{ij} = \mathrm{Cov}(X_i, X_j). \tag{36.29}$$

対角成分には個々の分散が並び、非対角成分には対ごとの共分散が並ぶ。$\mathrm{Cov}(X_i,X_j) = \mathrm{Cov}(X_j,X_i)$ であるため、この行列は対称である。また半正定値でもある。任意の固定ベクトル $\mathbf{a}$ に対し、スカラー射影 $\mathbf{a}^{\top}\mathbf{X} = \sum_i a_i X_i$ の偏差は $\mathbf{a}^{\top}(\mathbf{X}-\mu)$ となるため、次式が成り立つ。

$$\mathrm{Var}(\mathbf{a}^{\top}\mathbf{X}) = \mathbb{E}\big[\mathbf{a}^{\top}(\mathbf{X}-\mu)(\mathbf{X}-\mu)^{\top}\mathbf{a}\big] = \mathbf{a}^{\top}\Sigma\,\mathbf{a} \;\geq\; 0.$$

言葉で言えば、二次形式 $\mathbf{a}^{\top}\Sigma\,\mathbf{a}$ はデータを方向 $\mathbf{a}$ に射影したときの分散にほかならず、分散が負になることは決してない。ダーツの例では $\Sigma = \mathrm{diag}(\tfrac{1}{4},\tfrac{1}{4})$ であり、各座標は無相関で同じ広がりを持つ。本書の後半では、Σ の固有ベクトルが主成分分析（principal component analysis）の方向（33.5 節）として、また多変量ガウス分布の楕円状の等高線の主軸（36.3 節）として登場する。

36.2.3 確率変数の変換

機械学習では確率変数の変換が絶えず行われる。活性化関数は活性化前の値に作用し、サンプラは一様ノイズを変形し、正規化フロー（normalizing flow）は単純なガウス分布を複雑なデータ分布へと歪める。X が既知の密度を持ち、$Y = g(X)$ とおくとき、Y の密度はどうなるだろうか。

36.2.3.1 1 次元の規則

最初に思いつきそうな $p_Y(y) = p_X(g^{-1}(y))$ という推測は誤りであり、その理由はダーツの例からすでにわかる。ダーツの距離は $R = \sqrt{U}$（ただし U は $[0,1]$ 上の一様変数）として生成できるが、この写像は U の平坦な密度を傾いた密度 $2r$ へと変える。変換のもとで保存されるのが密度の高さではなく確率質量だからである。幅 Δx を持つ x の周りの狭い区間はおおよそ $p_X(x)\,\Delta x$ の確率を持ち、g によって幅おおよそ $\Delta y = |g'(x)|\,\Delta x$ の $y = g(x)$ の周りの区間へと写されるが、この区間も同じ確率を持たなければならない。

$$p_Y(y)\,|g'(x)|\,\Delta x \approx p_X(x)\,\Delta x.$$

g が数直線を引き伸ばす箇所では、同じ確率がより広い区間に広がるため、密度は拡大率 $|g'(x)|$ の分だけ薄くならなければならない。逆に g が数直線を圧縮する箇所では、密度は高くならなければならない。Figure 36.12 は $Y = e^X$ に対して面積の等しいこれら 2 つの帯を示している。

命題（1 次元の変数変換） **X が密度 p_X を持ち、この密度が開区間 I（ただし $P(X \in I) = 1$）上で連続であるとする。また $g : I \to \mathbb{R}$ は連続微分可能で $g' \neq 0$ が I 上で成り立ち、したがって g は狭義単調かつ単射であるとする。このとき $Y = g(X)$ は次の密度を持つ。**

$$p_Y(y) = p_X\!\big(g^{-1}(y)\big)\,\left|\frac{dg^{-1}}{dy}(y)\right| \tag{36.30}$$

これは区間 $g(I)$ 上での式であり、その外側では 0 である。

証明 まず g が増加関数であるとする。このとき $g(X) \leq y$ が成り立つのは $X \leq g^{-1}(y)$ のとき、かつそのときに限るため、累積分布関数は $F_Y(y) = F_X(g^{-1}(y))$ を $y \in g(I)$ において満たす。連鎖律および $F_X' = p_X$（36.1 節）を用いて微分すると、$p_Y(y) = p_X(g^{-1}(y))\,\frac{dg^{-1}}{dy}(y)$

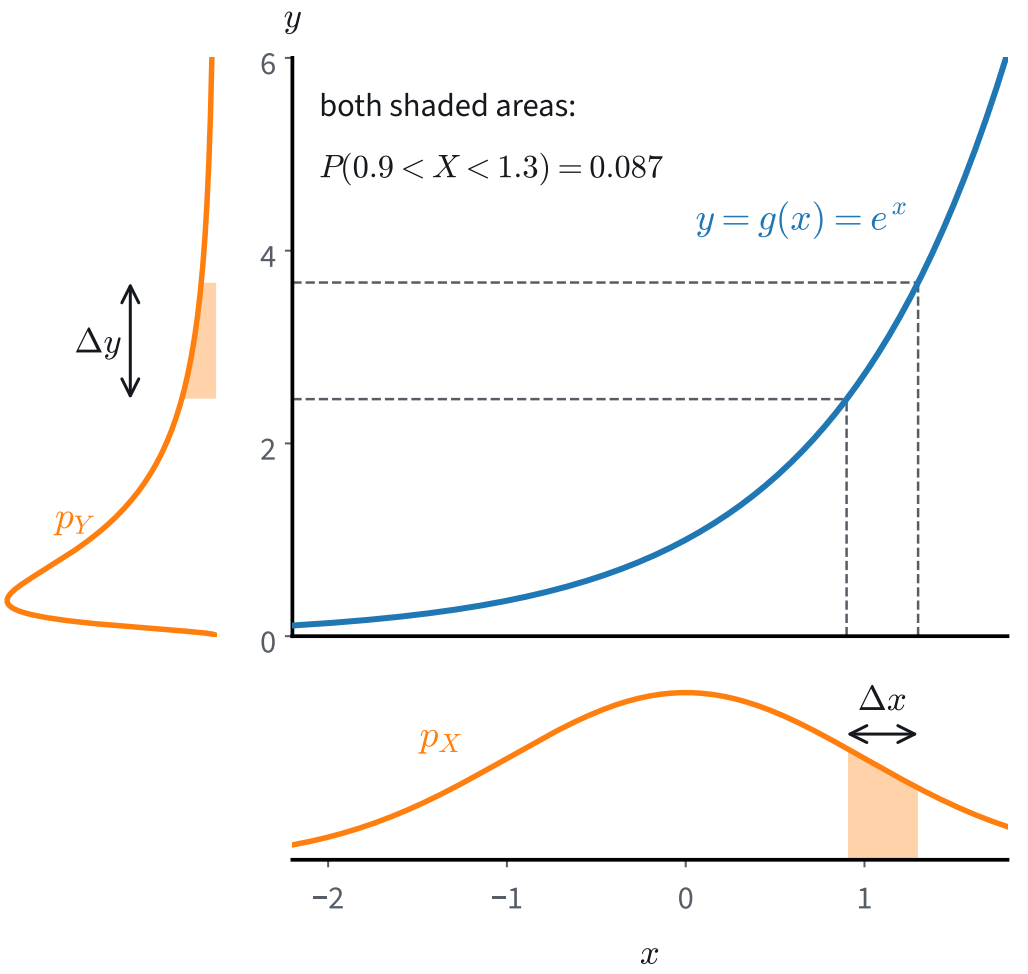


Figure 36.12: 変数変換。幅 Δx の p_X の下の区間（下）は、g によって幅 $\Delta y = g'(x)\,\Delta x$ の p_Y の下の区間（左）に写される。網掛けされた 2 つの面積は同じ確率を表すため、g が引き伸ばす箇所（$\Delta y > \Delta x$）では、密度 p_Y は因子 $|g'(x)|$ の分だけ低くならなければならない。ここでは $X \sim \mathcal{N}(0, 1)$、$Y = e^X$ としている。

が得られる。一方、g が減少関数であれば、$g(X) \le y$ は $X \ge g^{-1}(y)$ のときに成り立つため、$F_Y(y) = 1 - F_X(g^{-1}(y))$ となる。この導関数には負号が現れるが、それが dg^{-1}/dy の負号と打ち消し合うため、絶対値をとれば両方の場合を一度に扱える。いずれの場合も、F_Y は全確率を担う $g(I)$ 上で連続微分可能であるから、微分積分学の基本定理によりその導関数は Y の密度となる。$\blacksquare$

言葉で読み解くと、y における新しい密度は、y に写される点での元の密度に、逆写像の局所的な圧縮率を掛けたものである。$dg^{-1}/dy = 1/g'(x)$（ただし $x = g^{-1}(y)$）であるから、これは引き伸ばしの議論から導いた $1/|g'(x)|$ の因子とまったく同じである。

逆変換サンプリングは変数変換の一種であり、この公式を使えばダーツの距離のサンプラからその密度がただちに復元できる。$X = U$ を $I = (0,1)$ 上の一様変数とし、$g(u) = \sqrt{u}$ とおくと、$g^{-1}(r) = r^2$ かつ $|dg^{-1}/dr| = 2r$ となる。Equation 36.30 より、ダーツの距離の密度 $p_R(r) = 1 \cdot 2r$ が得られる。

g が単射でない場合、複数の入力点が同じ出力 y に写り得るため、それぞれの確率の寄与を単純に足し合わせればよい。

$$p_Y(y) = \sum_{x:\, g(x)=y} \frac{p_X(x)}{|g'(x)|},$$

これは、値 y の逆像が有限個しかなく、その各点で $g'(x) \neq 0$ を満たす場合に成り立つ。$Y = X^2$ かつ $y > 0$ のとき、2 つの逆像 $\pm\sqrt{y}$ はどちらも $|g'| = 2\sqrt{y}$ を満たすため、$p_Y(y) = [p_X(\sqrt{y}) + p_X(-\sqrt{y})]/(2\sqrt{y})$ となる。ここで $g' \neq 0$ という仮定は本質的である。ReLU 活性化関数は $g' = 0$ となる負の半直線全体 $(-\infty, 0)$ の確率質量をすべて 1 点 0 に潰してしまい、純粋な密度ではなく点質量（point atom）を生じさせる（演習問題 2）。

具体例として、$X \sim \mathcal{N}(0,1)$ が標準正規密度 $p_X(x) = e^{-x^2/2}/\sqrt{2\pi}$ を持つとし、$Y = e^X$ とおく。このとき $g^{-1}(y) = \log y$ および $|dg^{-1}/dy| = 1/y$（$y > 0$）であるから、変数変換の公式により次の**対数正規**（log-normal）密度が得られる。

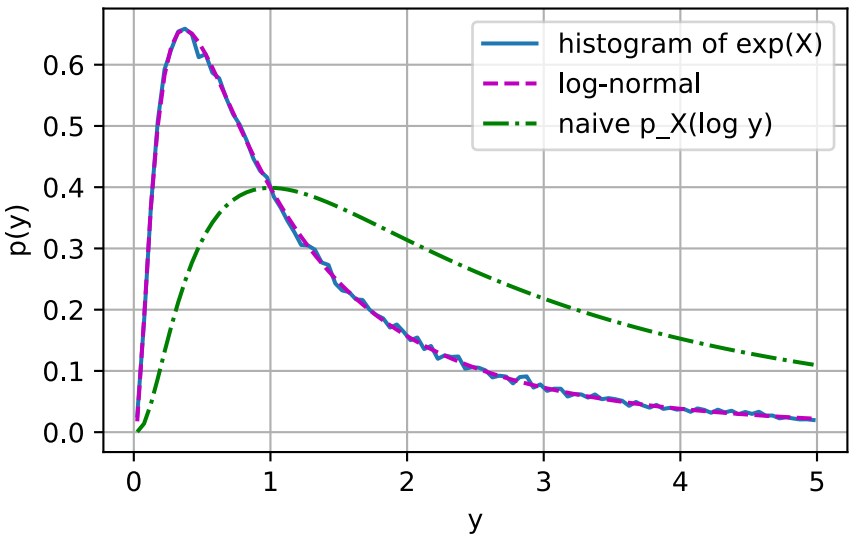
$$p_Y(y) = \frac{1}{y\sqrt{2\pi}} \exp\!\Big(-\tfrac{1}{2}(\log y)^2\Big), \qquad y > 0.$$

先頭の因子 $1/y$ が変数変換による補正項である。$\log p_Y(y) = -\log y - \tfrac{1}{2}(\log y)^2 + \text{const}$ の導関数を 0 とおくと $\log y = -1$ となるため、真の密度は $y = e^{-1} \approx 0.368$ でピークをとる。対照的に、素朴な推測 $p_X(\log y)$ は $y = 1$ でピークをとり、そもそも正当な密度ですらない。$y = e^x$ を代入すると全体の面積は $\int p_X(x)\, e^x\, dx = \mathbb{E}[e^X] = e^{1/2} \approx 1.65$ となってしまうからである（平方完成により $x - x^2/2 = \tfrac{1}{2} - \tfrac{1}{2}(x-1)^2$ となる）。

同じ計算からすべてのモーメント $\mathbb{E}[Y^k] = \mathbb{E}[e^{kX}] = e^{k^2/2}$ が求まり、そのどれもが有限である。それにもかかわらず、上側の裾 $P(Y > t) = P(X > \log t)$ は t のいかなる指数関数よりも遅く減衰する。したがって対数正規分布は裾が重い（heavy-tailed）が、コーシー分布とは異なりあらゆる次数の有限なモーメントを持つ。

10^5 個の e^X のサンプルのヒストグラムを、2 つの候補曲線と並べてプロットしてみよう。素朴な曲線が面積を 0.65 だけ余分に囲んでしまうことはすでにわかっている。図を見る前に、その余分な面積がどこに現れるか予想してみてほしい。

```
y = np.exp(rng.standard_normal(10**5))                # Y = e^X
edges = np.linspace(0, 5, 101)
mids = (edges[:-1] + edges[1:]) / 2
hist = np.histogram(y, edges)[0] / (len(y) * (edges[1] - edges[0]))
p_X = lambda x: np.exp(-x**2 / 2) / np.sqrt(2 * np.pi)
d2l.plot(mids, [hist, p_X(np.log(mids)) / mids, p_X(np.log(mids))],
         'y', 'p(y)', figsize=(5, 3),
         legend=['histogram of exp(X)', 'log-normal', 'naive p_X(log y)'])
```



ヒストグラムはサンプリングノイズの範囲内で対数正規密度によく一致しており、ピークは $y = 1$ よりずっと左側の e^{-1} 付近にある。素朴な曲線はピークの位置を誤っている。真の密度に y を掛けたものに等しいため、$y < 1$ ではデータを下回り、$y > 1$ では裾に向かうにつれて増大し続ける倍率でデータを上回るのである。

36.2.3.2 多変量の規則

多次元では、変換は長さではなく面積や体積を引き伸ばすため、拡大率は行列式になる。最も単純な場合として、線形変換 $\mathbf{Y} = \mathbf{A}\mathbf{X}$（ただし $\mathbf{A} = \mathrm{diag}(2, 3)$）から始めよう。この写像は単位正方形を、大きさ 2×3 で面積 $6 = \det \mathbf{A}$ の長方形へと写す。この長方形は元の正方形と同じ確率を持たなければならないため、$\mathbf{Y}$ の密度は対応する点における $\mathbf{X}$ の密度の 6 分の 1、すなわち $p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{X}}(\mathbf{A}^{-1}\mathbf{y})/6$ にならなければならない。

非線形写像では局所的な拡大率が点ごとに変化するが、これは重積分の変数変換の定理によって扱える（34.6 節）。g を $\mathbb{R}^n$ の 2 つの領域間の連続微分可能な全単射とし、その逆写像も連続微分可能であるとする。また、逆写像の偏微分を並べたヤコビ行列を $J_{g^{-1}}$ と書く。局所的に g^{-1} は体積を $\left|\det J_{g^{-1}}\right|$ 倍にスケールするため、$\mathbf{x} = g^{-1}(\mathbf{y})$ を領域 $\mathcal{D}$ の逆像上の積分に代入すると次式が得られる。

$$\int_{g^{-1}(\mathcal{D})} f(\mathbf{x})\, d\mathbf{x} = \int_{\mathcal{D}} f(g^{-1}(\mathbf{y}))\, \left|\det J_{g^{-1}}(\mathbf{y})\right|\, d\mathbf{y}.$$

ここで $\mathbf{Y} = g(\mathbf{X})$ とおく。事象 $\mathbf{Y} \in \mathcal{D}$ は事象 $\mathbf{X} \in g^{-1}(\mathcal{D})$ と同一であるため、$f = p_{\mathbf{X}}$ としてこの定理を適用すると次のようになる。

$$P(\mathbf{Y} \in \mathcal{D}) = \int_{g^{-1}(\mathcal{D})} p_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} = \int_{\mathcal{D}} p_{\mathbf{X}}(g^{-1}(\mathbf{y}))\, \left|\det J_{g^{-1}}(\mathbf{y})\right| d\mathbf{y}.$$

したがって、右辺の被積分関数が $\mathbf{Y}$ の密度となる。

$$\begin{aligned} p_{\mathbf{Y}}(\mathbf{y}) &= p_{\mathbf{X}}(g^{-1}(\mathbf{y}))\, \left|\det J_{g^{-1}}(\mathbf{y})\right|, \\ \log p_{\mathbf{Y}}(\mathbf{y}) &= \log p_{\mathbf{X}}(\mathbf{x}) - \log \left|\det J_{g}(\mathbf{x})\right| \\ &\qquad \text{(equivalently, at } \mathbf{x} = g^{-1}(\mathbf{y})). \end{aligned} \tag{36.31}$$

2 行目では $J_{g^{-1}}(\mathbf{y}) = J_g(\mathbf{x})^{-1}$ を用いた。サンプルを g に通すと、その対数密度から局所的な体積拡大率の対数が引かれるだけである。$\log|\det(J_2 J_1)| = \log|\det J_1| + \log|\det J_2|$ が成り立つため、写像を合成するとそれぞれの対数行列式の和が差し引かれる。これこそが、**正規化フロー**が厳密な尤度を計算する仕組みそのものである。すなわち、対数行列式を安価に評価できる可逆な層を積み重ねるのであり (Rezende and Mohamed 2015)、38.3.3 節では微分方程式のステップを層とするフローを構築する。また、演習問題 5 ではカップリング層をゼロから実装する。

最後に、ダーツの例でもう一度これを確認しよう。ダーツの位置を極座標である距離 R と角度 Θ により $(x, y) = (r\cos\theta, r\sin\theta)$ と表す。(r, θ) から (x, y) へのこの写像のヤコビ行列式は $\cos\theta \cdot r\cos\theta - (-r\sin\theta)\sin\theta = r$ であるため、多変量の規則から次式が得られる。

$$p_{R,\Theta}(r, \theta) = \frac{1}{\pi} \cdot r = 2r \cdot \frac{1}{2\pi}, \qquad 0 \le r \le 1,\ 0 \le \theta < 2\pi.$$

同時密度はきれいに因数分解される。角度 Θ は $[0, 2\pi)$ 上の一様分布に従い、距離 R は再び密度 $2r$ を（今度はヤコビ行列から直接）復元し、2 つの変数は独立である。デカルト座標ではダーツの X と Y は従属であったが、適切な座標系をとれば、まったく同じダーツが 2 つの独立な変数によって記述される。

本節では、平均に関する中心的な問いが 1 つ未解決のまま残されている。独立な n 個の抽出値の平均が分散 σ^2/n を持つこと（Equation 36.27）はわかったが、その分布はまだわかっていない。チェビシェフの不等式（36.1 節）を使えば、この分散から $\sigma^2/(n\epsilon^2)$ という上界（平均が ϵ 以上の誤差を生じる確率に対する上界）が得られるが、有界な抽出値やガウス分布からの抽出値に対してはこの上界はあまりに悲観的であり、真の誤差確率は n に関して指数関数的に減少する。36.3.2.3 節では平均がどの分布に近づくかを特定し、36.9 節ではその指数減衰率を確立する。ただし、そのどちらに取り組むにも、まずはダーツだけにとどまらない豊かな分布のカタログが必要である。

36.2.4 まとめ

周辺密度は不要な変数を積分消去することで得られ、条件付き密度は同時密度を細い帯に沿ってスライスして再正規化することで得られる。条件付き平均は最適な二乗誤差予測器であり、全分散の法則はある変数の分散を、条件付ける変数が説明する部分と説明できない部分とに分解する。共分散と相関は一般の従属性ではなく線形な関連を測るものであり、有限の分散を持つ独立な n 個の抽出値を平均すると分散は σ^2/n に減少する。滑らかな可逆写像のもとでは、密度はヤコビ行列式の絶対値で与えられる局所的な体積拡大率で割られる。ダーツの例はこれらすべての概念

を 1 つに結びつける。すなわち、デカルト座標では従属でありながら無相関であるが、極座標に移ると 2 つの独立な変数になる。

36.2.5 演習問題

1. **無相関だが従属。**白色化（たとえば 33.5 節の主成分を用いたもの）は、各座標を無相関にする。次の変数の対は、各座標がガウス分布に従い共分散が 0 になる例である。$X \sim \mathcal{N}(0,1)$ とし、S を X と独立で $P(S=1)=P(S=-1)=\tfrac{1}{2}$ を満たすランダムな符号とし、$Y = SX$ とおく。

   1. 累積分布関数を計算することで $Y \sim \mathcal{N}(0,1)$ であることを示し、さらに $\mathrm{Cov}(X,Y)=0$ であることを示せ。
   2. $P(|X|\le 1, |Y|\le 1)$ と $P(|X|\le 1)\,P(|Y|\le 1)$ を比較せよ。X と Y は独立か。
   3. $X+Y$ が 0 に点質量を持つため、$X+Y$ はガウス分布に従わないことを示せ。確率 1 で点 (X,Y) は 2 本の直線 $y=\pm x$ のいずれかの上に乗るが、直線は面積を持たないため直線上の二重積分は 0 である。各座標がそれぞれ密度を持つにもかかわらず、組 (X,Y) は同時密度を持たないことを結論づけよ。

2. **相関のある推定量の平均化。**本文では独立な抽出に対して $\mathrm{Var}(\bar{X}_n)=\sigma^2/n$ を導出した。従属な推定量の平均はこれとは異なる振る舞いをする。

   1. $Y_1,\dots,Y_n$ がそれぞれ分散 σ^2 を持ち、任意の対の間の相関が ρ であるとする（同じデータで訓練された n 個のアンサンブルメンバーの予測などがこれにあたる）。次式を示せ。

$$\mathrm{Var}\Big(\frac{1}{n}\sum_{i=1}^{n} Y_i\Big) = \sigma^2\Big(\rho + \frac{1-\rho}{n}\Big). \tag{36.32}$$

      $n\to\infty$ のとき、平均化によって何が達成されるか。

   2. $\mathbf{a}^\top \Sigma\, \mathbf{a} \ge 0$ を共分散行列 Σ（$Y_1,\dots,Y_n$ の共分散行列）に適用し、$\rho \ge -1/(n-1)$ であることを示せ。また、この下界を達成する変数の例を挙げよ。

3. [code] **ガウス変数のシグモイド変換。**$X\sim\mathcal{N}(0,s^2)$ をシグモイドユニットの活性化前の値とし、$Y=\mathrm{sigmoid}(X)$（その逆関数は $\mathrm{logit}(y)=\log(y/(1-y))$）とおく。

   1. Equation 36.30 を用いて、密度 p_Y を区間 $(0,1)$ 上で求めよ。
   2. どのような s に対して、$y=\tfrac{1}{2}$ で p_Y が極小値をとるか。$\log p_Y$ を $z=\mathrm{logit}(y)$ の関数として書くと計算が簡単になる。
   3. `#random-variables-change-of-variables` セルを修正し、Y のヒストグラムと p_Y を $s\in\{1,\sqrt{2},3\}$ のそれぞれについて比較せよ。$s=3$ におけるヒストグラムは、誤差逆伝播法がこれらのユニットの勾配を伝播させる際に通る導関数 $\mathrm{sigmoid}'(X)=Y(1-Y)$ について何を示唆するか。

4. **説明分散。**●回帰モデル h がターゲット X を入力 Y から予測するとする。このモデルが説明する分散の割合は $R^2 = 1 - \mathbb{E}\big[(X-h(Y))^2\big]/\mathrm{Var}(X)$ である。

   1. Equation 36.23 と全分散の法則 Equation 36.22 を用い、任意の予測器について $R^2 \le \mathrm{Var}(\mathbb{E}[X\mid Y])/\mathrm{Var}(X)$ が成り立つことを示せ。本文の 2 成分混合分布において X をラベル Z から予測する場合と、ダーツにおいて X を Y から予測する場合について、この上限の値を求めよ。

2. 入力が与えられたもとで、ターゲットは -1 または $+1$ をそれぞれ確率 $\frac{1}{2}$ でとるとする。期待二乗誤差を最小化する予測値は何か。また、ターゲットがその値をとる頻度はどれだけか。期待絶対誤差 $\mathbb{E}\big[|X - h(Y)|\big]$ を最小化する予測値は何か。

5. [code] **アフィンカップリング層とボックス・ミュラー法。**●正規化フローは、Equation 36.31 における対数行列式を安価に計算できる可逆写像を積み重ねる。RealNVP (Dinh, Sohl-Dickstein, et al. 2017) のアフィンカップリング層は、$\mathbf{x} \in \mathbb{R}^2$ を次式へと写す。

$$y_1 = x_1, \qquad y_2 = x_2 \, e^{s(x_1)} + t(x_1), \tag{36.33}$$

ここで s と t は任意の関数であり、たとえば小さなニューラルネットワークである。

1. 逆写像とヤコビ行列 $J_g(\mathbf{x})$ を書き下せ。また、$\log \left|\det J_g(\mathbf{x})\right| = s(x_1)$ が、s と t がどれほど複雑であっても成り立つことを示せ。
2. X_1 と X_2 を独立な標準正規変数とし、$s(x_1) = \tanh(x_1)$、$t(x_1) = x_1^2/2$ とおく。Equation 36.31 を用いて $\log p_{\mathbf{Y}}(\mathbf{y})$ を閉じた形で表せ。NumPy を用い、$p_{\mathbf{Y}}$ を領域 $[-6, 6] \times [-12, 24]$ 上の間隔 0.02 のグリッドで評価し、そのリーマン和が 1 に近いことを確かめよ。また、いくつかのランダムな点で、対数行列式を有限差分によるヤコビ行列の行列式と照合して確認せよ。
3. 第 1 座標はこの層をそのまま通過する。Y_1 の分布は、s と t が何であってもどうなるか。合成写像が両方の周辺分布を変化させられるようにするには、2 つ目のカップリング層をどのように配置すべきか。また、その合成写像の対数行列式はどうなるか。
4. フローは一様ノイズからガウス分布へ向かう向きにも構成できる。U_1, U_2 を独立な $(0, 1)$ 上の一様変数とし、$R = \sqrt{-2 \log U_1}$、$\Theta = 2\pi U_2$、$(Z_1, Z_2) = (R\cos\Theta, R\sin\Theta)$ とおく。1 次元の規則を用いて R の密度を求め、次に (R, Θ) の同時密度を求め、本文の極座標のヤコビ行列を使ってデカルト座標へ変換することで、Z_1 と Z_2 が独立な標準正規変数であることを示せ。これがボックス・ミュラー法である。10^5 個のサンプルを用いて、(Z_1, Z_2) の標本共分散行列が単位行列に近いことを確かめよ。

36.3 分布

36.1 節と 36.2 節では、確率質量関数、確率密度、期待値、分散といったランダムな量を扱うための一般的な言語を整備した。深層学習におけるあらゆる予測モデルは、明示しているかどうかにかかわらず、この言語を話している。二値分類器は 1 つの数値、すなわちラベルが 1 である確率を出力する。言語モデルは語彙に含まれるすべてのトークンに対して、総和が 1 になるよう制約された確率を出力する。回帰ネットワークは実数値を予測し、二乗誤差を最小化するように訓練される。いずれの場合も、モデルはラベルに対する確率分布を暗黙のうちに選択しており、その選択が最小化すべき損失関数を決定している。では、それらはどのような分布であり、どこから現れ、その平均と分散はどのように計算できるのだろうか。

驚くべきことに、たった 1 回のコイン投げから出発するだけで、この問いのほぼすべてに答えることができる。コインを 13 回投げて 9 回表が出たとする。このコインに偏りがない（公正である）としたら、この結果はどれほど珍しいのだろうか。この問いに答えるには、繰り返しコインを投げたときに表が出る回数の分布が必要になるが、実はこれは分類器がテスト例を何個正しく分類できるかを支配する分布とまったく同じである。すぐに確かめるように、9 回表が出る確率は約 9% である。

本節の中心的な考え方は、少数の基本的な分布がすべてその 1 回のコイン投げから生まれるという点にある。決まった回数のコイン投げで表の数を数えれば二項分布（binomial distribution）が得られる。めったに表が出ないコイン投げを多数回行う極限をとればポアソン分布（Poisson

distribution）が導かれ、次の希少な事象が起こるまでの待ち時間は指数分布（exponential distribution）を与える。通常のコイン投げを多数回足し合わせてスケールを調整すれば、ガウス分布（Gaussian distribution）が現れる。どの分布もより単純な要素から組み立てられているため、その平均と分散は構成法から直接、毎回短く本質を突いた議論によって導かれる。

適切な分布を選ぶことは、単なる記法の問題ではない。たとえば分散が小さすぎるカウントモデルを使うと、そこから計算されるエラーバーはどれも実際より見かけ上狭くなってしまう。本節の 3 つの結果、すなわちポアソン極限とその待ち時間、中心極限定理（central limit theorem）、そしてガウスベクトルに対する条件付けの公式は、本節の範囲を超えて広く重要となる。対照的に、離散一様分布（discrete uniform distribution）、ガンマ分布（Gamma distribution）、ラプラス分布（Laplace distribution）は初読時には流し読みしても構わない。

Figure 36.13 に、これから学ぶ各分布が互いにどのように構成されるかをまとめる。

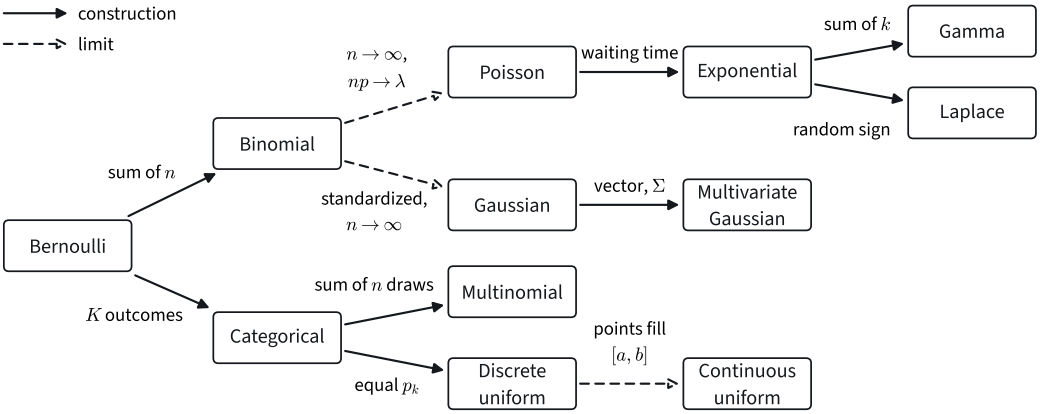


Figure 36.13: 本節で扱うすべての分布は 1 回のコイン投げから派生する。実線の矢印は（和、待ち時間、ランダムな符号、座標のベクトル化、または等確率の割り当てによって）新たな分布を構成する操作を表し、破線の矢印は試行回数や点の数が増大する極限を表す。ポアソン分布と指数分布は同一の事象のストリームを、それぞれ時間枠内の個数および到着間隔の時間として記述する。

36.3.1 離散分布

離散確率変数は、$\{0, 1\}$ や $\{0, 1, 2, \dots\}$ のような可算集合に値をとる。その分布は確率質量関数（probability mass function、pmf）$P(X = k) \geq 0$（ただし $\sum_k P(X = k) = 1$）によって定まる。平均は $\mathbb{E}[X] = \sum_k k\, P(X = k)$ であり、分散は $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ である。Figure 36.14 に、ここで登場する離散分布を通し例で評価した概観を示す。

36.3.1.1 ベルヌーイ分布

最も単純なランダムな量は、結果が 2 通りしかないものである。コインが表になるか裏になるか、ユーザーが広告をクリックするか無視するか、分類器がある例に正解するか不正解になるかといった具合である。この試行は、1 とラベル付けした結果が生じる確率 p というたった 1 つの数値で完全に記述される。

ベルヌーイ確率変数 $X \sim \mathrm{Bernoulli}(p)$ は、値 1 を確率 p でとり、値 0 を確率 $1 - p$ でとる（ただし $0 \leq p \leq 1$）。これら 2 つの場合は 1 つの式にすっきりとまとめられる。

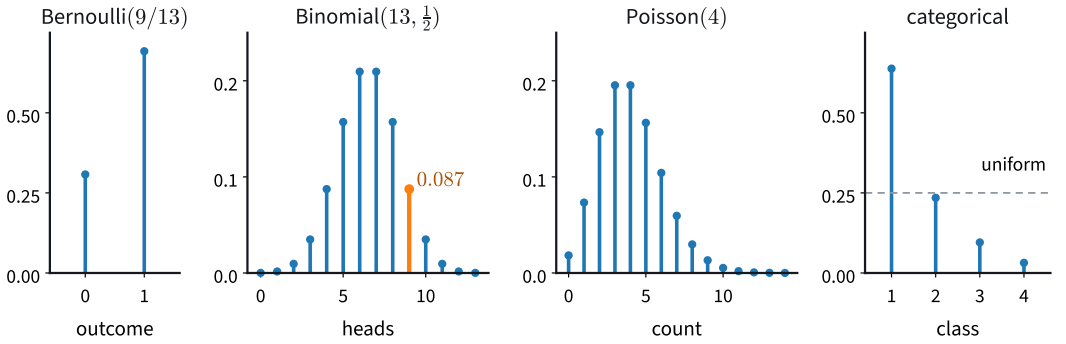


Figure 36.14: 通し例における 4 つの確率質量関数。Bernoulli$(9/13)$：観測された表の頻度に一致するコインの 1 回投げ。Binomial$(13, \frac{1}{2})$：公正なコインを 13 回投げたときの表の回数であり、オレンジ色の棒は $P(X = 9) = 715/8192 \approx 0.087$ を強調している。Poisson(4)：平均レートが毎分 4 件であるサーバーへのリクエスト数。ロジット $(2, 1, 0.1, -1)$ から生成された 4 クラスのカテゴリカル分布（一様確率 $1/4$ と対比して表示）。二項分布とポアソン分布はそれぞれの平均（6.5 と 4）付近でピークをとり共通の縦軸スケールを用い、結果の数が少ない残り 2 つのパネルは別の縦軸スケールを共有している。

$$P(X = x) = p^x (1 - p)^{1-x}, \qquad x \in \{0, 1\}. \tag{36.34}$$

ここで指数はスイッチの役割を果たす。$x = 1$ のときこの式は p になり、$x = 0$ のときは $1 - p$ になる。

モーメントの計算は容易である（1.6 節と同様）。平均は単に $\mathbb{E}[X] = 1 \cdot p + 0 \cdot (1 - p) = p$ となる。X は 0 と 1 の値しかとらないため $X^2 = X$ が成り立ち、$\mathbb{E}[X^2] = p$ でもある。したがって分散は $\mathrm{Var}(X) = p - p^2 = p(1 - p)$、すなわち 2 つの結果の確率の積となる。分散は結果が確定的であるとき（$p = 0$ または $p = 1$）に 0 となり、最大値 $\frac{1}{4}$ に達するのは公正なコイン（$p = \frac{1}{2}$）のときである。

深層学習では、二値分類器はラベル y が 1 に等しい確率の推定値 $\hat{p}$ を出力する。Equation 36.34 より、観測されたラベルに割り当てられる負の対数確率は $-\log P(y) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$ であり、これが**二値交差エントロピー**（binary cross-entropy）損失である。したがって、二値交差エントロピーで分類器を訓練することは、各ラベルにベルヌーイ分布を当てはめることと等価である。

以下では 2 種類の異なるコインが登場するため、両者を明確に区別しておくとよい。冒頭の問いで扱ったのは $p = \frac{1}{2}$ の公正なコインである。これに対し、観測データと確率が一致するコインは $p = 9/13 \approx 0.69$ をもち、その 1 回投げの分散は $\frac{9}{13} \cdot \frac{4}{13} \approx 0.21$ である。

36.3.1.2 二項分布

冒頭のパズルに戻ろう。知りたいのは、公正なコインを 13 回投げたときに表が出る回数の分布、とりわけ 9 回表が出る確率がどれほどかということである。

まずは自明でない最小のケースである 2 回のコイン投げから始めよう。起こり得る 4 通りの結果は TT、HT、TH、HH であり、次を与える。

$$P(X=0)=(1-p)^2, \qquad P(X=1)=2p(1-p), \qquad P(X=2)=p^2.$$

$P(X=1)$ に現れる因子 2 は表 1 回と裏 1 回の並べ方が 2 通りあることを表しており、二項展開により 3 つの確率の和は $(p+1-p)^2=1$ となる。

一般に、n 回のコイン投げで表が出る回数は、各試行に対応する n 個の指示変数の和で表される。各コイン投げ $X_1,\dots,X_n$ が独立であり、それぞれが Bernoulli(p)（共通の p）に従うとき、それらの和は**二項**分布に従う。

$$X=\sum_{i=1}^{n} X_i \sim \text{Binomial}(n,p). \tag{36.35}$$

組み合わせは $\binom{n}{k}$ 通り（どの k 回が n 回のコイン投げのうち表になるかの選び方）あり、独立性により特定のどの並びも確率 $p^k(1-p)^{n-k}$ をもつ。並びの総数に 1 つの並びの確率を掛ければ、二項分布の確率質量関数が得られる。

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}, \qquad k=0,1,\dots,n. \tag{36.36}$$

$n=2$ のとき、これは先ほどの 3 つの確率を再現する。公正なコインの場合、13 回投げて 9 回表が出る並びは $\binom{13}{9}=715$ 通りあり、それぞれの確率は $2^{-13}=1/8192$ なので、次のようになる。

$$P(X=9)=\frac{715}{8192}\approx 0.087.$$

表が 9 回**以上**出る確率は $(715+286+78+13+1)/8192$ すなわち $1093/8192\approx 0.133$ である。裾確率（tail probability）が 13.3% であることからコインの公平性を疑うのに十分かどうかは統計学の問題であり、ここではひとまず脇に置いておく。

平均と分散。確率質量関数から直接 $\sum_k k\,P(X=k)$ を計算するのは手間がかかる。代わりに、和による表現 Equation 36.35 を使えば両方のモーメントが 1 行で求まる。期待値の線形性は確率変数が独立か従属かを問わず成り立つため、次を得る。

$$\mathbb{E}[X]=\sum_{i=1}^{n}\mathbb{E}[X_i]=np.$$

分散については、36.2 節で見たように独立な確率変数の共分散は 0 であるため、**独立な**変数の和の分散はそれぞれの分散の和になる。

$$\text{Var}(X)=\sum_{i=1}^{n}\text{Var}(X_i)=np(1-p).$$

このように、n 回の独立なコイン投げは、1 回投げの n 倍の平均と n 倍の分散をもつ。公正なコインを 13 回投げる場合、平均は 6.5、分散は 3.25、標準偏差は $\sqrt{3.25}\approx 1.80$ である。9 回表が出るという結果は、平均より標準偏差の $(9-6.5)/1.80\approx 1.4$ 倍だけ上に位置している。

2 つのモーメントの公式が異なる仮定に依拠している点は見落としやすいため注意しておきたい。平均 np はコイン投げが互いに従属していても成り立ち、i 回目のコイン投げが固有の確率 p_i をもつ場合でも平均は単に $\sum_i p_i$ となる。しかし分散の公式には、独立性が決定的に必要である。もし 13 回の「コイン投げ」が実際には公正なコインを 1 回投げた結果を 13 回コピーしたものだったとすれば、X は 0 または 13 の値しかとらず、その分散は $13^2 \cdot \frac{1}{4} = 42.25$ と、$np(1-p)$ の 13 倍にもなってしまう。

隣接項の比。@eq-mdl-binomial-pmf を直接計算しようとすると階乗が現れ、n が大きいときにはすぐにオーバーフローしてしまう。幸いなことに、隣り合う確率の比をとれば階乗がすべて約分されるため、階乗を 1 つも使わずに確率質量関数全体を計算できる。

$$\frac{\binom{n}{k+1}}{\binom{n}{k}} = \frac{k!\,(n-k)!}{(k+1)!\,(n-k-1)!} = \frac{n-k}{k+1}, \qquad \text{so} \qquad \frac{P(X=k+1)}{P(X=k)} = \frac{n-k}{k+1}\cdot\frac{p}{1-p}.$$

この比には直接的な組み合わせ論的意味がある。k 個の表からなる集合から出発して、残りの $n-k$ 個の裏のどれか 1 つを選んで表に裏返すことを考えると、得られる $k+1$ 個の表の各集合には $k+1$ 通りの経路で到達でき、因子 $p/(1-p)$ が裏の確率を表の確率へと置き換える。$P(X=0) = (1-p)^n$ から出発してこの比を繰り返し掛けていけば、確率質量関数全体が生成される。さらに、この比は 1 を超えるのが $k+1 < (n+1)p$ の間であり、その後は 1 を下回るため、二項分布の確率質量関数は平均付近の単一のピークまで滑らかに上昇してから減少することがわかる。

この漸化式は Python でわずか 3 行で書ける。以下でポアソン極限とガウス分布の釣鐘曲線の両方をこれと比較するため、再利用可能な関数にまとめておこう。

```
def binomial_pmf(n, p, kmax):
    """P(X=k) for k = 0..kmax, built from P(X=0) by the successive ratio."""
    pmf = [(1 - p) ** n]
    for k in range(kmax):
        pmf.append(pmf[-1] * max(n - k, 0) / (k + 1) * p / (1 - p))
    return np.array(pmf)

n, p = 13, 0.5
pmf = binomial_pmf(n, p, n)
print(f'P(X=9): by the ratio {pmf[9]:.4f}, by counting {math.comb(13, 9) / 2**13:.4f}')
print(f'P(X>=9) = {pmf[9:].sum():.4f}')
print(f'mean np = {n * p}, variance np(1-p) = {n * p * (1 - p)}')
```

```
P(X=9): by the ratio 0.0873, by counting 0.0873
P(X>=9) = 0.1334
mean np = 6.5, variance np(1-p) = 3.25
```

期待どおり、比の漸化式から $715/8192 \approx 0.0873$ と裾確率 0.1334 が再現される。

36.3.1.3 ポアソン分布

現実世界のカウントデータの多くには、自然な上限 n が存在しない。ある Web サーバーが毎分平均 $\lambda = 4$ 件のレートでリクエストを受け取っており、到着数に決まった上限はないとしよう。リクエストが 1 件も来ない静かな 1 分間や、10 件以上のリクエストが突発的に押し寄せるバーストは、どれほどの確率で起こるだろうか。

これをコイン投げの枠組みに結びつけるため、1 分間を n 個の微小な部分区間に分割してみよう。ここでは 2 つの単純化の仮定をおく。各部分区間は十分に短く同じスライスに 2 件以上のリクエストが到着することはないとし、各スライスは到着確率が等しく $4/n$ である独立な試行として振る舞うものとする。これらの仮定のもとで、1 分間の合計件数は $\mathrm{Binomial}(n, 4/n)$ に従う。その平均は $n \cdot \frac{4}{n} = 4$ となってどの n でも一定であり、分散 $4(1-\frac{4}{n})$ は着実に 4 へと近づいていく。リクエストが 0 件である静かな 1 分間の確率は $(1-\frac{4}{n})^n$ であり、これは 0.0115（$n=20$）、0.0169（$n=100$）、0.0182（$n=1000$）となって $e^{-4} \approx 0.0183$ に収束する。スライスを細かくするにつれて確率は安定し、きれいな極限分布の存在を示唆している。

極限の導出。 $\mathrm{Binomial}(n, \lambda/n)$ においてカウント k を固定し、$n \to \infty$ としたときの極限を考えよう。二項分布の確率質量関数を書き直すと 4 つの因子が現れ、そのうち 3 つは初等的な極限をもつ。

$$\binom{n}{k}\left(\frac{\lambda}{n}\right)^k\left(1-\frac{\lambda}{n}\right)^{n-k} = \underbrace{\frac{n(n-1)\cdots(n-k+1)}{n^k}}_{\to 1}\cdot\frac{\lambda^k}{k!} \cdot \underbrace{\left(1-\frac{\lambda}{n}\right)^{n}}_{\to e^{-\lambda}} \cdot \underbrace{\left(1-\frac{\lambda}{n}\right)^{-k}}_{\to 1}.$$

第 1 の因子はそれぞれ 1 に近づく k 個の分数の積であり、第 3 の因子は指数関数を定義する古典的な極限 $(1-\lambda/n)^n \to e^{-\lambda}$ であり、最後の因子は k が固定されているため 1 に近づく。成功確率の任意の列 p_n（ただし $np_n \to \lambda$）に対しても $(1-p_n)^n \to e^{-\lambda}$ が成り立つため、まったく同じ議論が通用する。この結果は**希少事象の法則**（law of rare events）として知られ、残った因子からレート $\lambda > 0$ の**ポアソン**確率質量関数が得られる。

$$P(X=k) = \frac{\lambda^k e^{-\lambda}}{k!}, \qquad k = 0, 1, 2, \ldots \tag{36.37}$$

これを $X \sim \mathrm{Poisson}(\lambda)$ と書く。個々には起こりにくい多数の独立な機会からカウントが生じ、全体の期待値が一定の λ であるときにはいつでもポアソン分布が現れる。確率の総和が 1 になることは、テイラー級数 $\sum_{k\ge 0} \lambda^k/k! = e^{\lambda}$ から直ちにわかる。

平均と分散。 両方のモーメントは確率質量関数から直接読み取れる。$\mathbb{E}[X]$ の和では $k=0$ の項が消え、因子 k が $k!$ と約分され、添字を $j = k-1$ にずらせばポアソン確率変数の全確率が残る。

$$\mathbb{E}[X] = \sum_{k\ge 1} k\,\frac{e^{-\lambda}\lambda^k}{k!} = \lambda \sum_{k\ge 1} \frac{e^{-\lambda}\lambda^{k-1}}{(k-1)!} = \lambda \sum_{j\ge 0} \frac{e^{-\lambda}\lambda^{j}}{j!} = \lambda.$$

まったく同じ手法を $k(k-1)$ に適用すると $\mathbb{E}[X(X-1)] = \lambda^2$ が得られ、そこから分散が導かれる。

$$\mathrm{Var}(X) = \mathbb{E}[X(X-1)] + \mathbb{E}[X] - \mathbb{E}[X]^2 = \lambda^2 + \lambda - \lambda^2 = \lambda.$$

ポアソン分布を特徴づける性質は、平均と分散が等しいことである。先ほどの Web サーバーでは $P(X=0) = e^{-4} \approx 0.018$、$P(X=4) = \frac{4^4}{4!}e^{-4} \approx 0.195$、$P(X \ge 10) \approx 0.008$ となり、リクエストが来ない静かな 1 分間は 1 時間に約 1 回、10 件以上のバーストは約 2 時間に 1 回起こる。平均と分散が等しいというこの性質は、鋭い診断基準にもなる。多数の分にわたって観測したリ

クエスト数の分散が平均の数倍に達しているなら、モデル化の仮定の少なくとも 1 つが破綻しており（時間帯によってトラフィックが変動しているか、リクエストが連鎖的な後続リクエストを引き起こしているのかもしれない）、ポアソンモデルから求めたエラーバーは危険なほど狭くなってしまう。ポアソン分布のレート自体を確率変数とする標準的な対処法は、演習問題 1 で扱う。

ポアソン過程。単位時間あたりレート λ で区間を細分化する描像を時間軸全体に広げると、連続時間で事象が到着するモデルが得られる。長さ t の任意の時間枠には nt 個のスライスが含まれるため、その事象数は $\mathrm{Binomial}(nt, \lambda/n)$ となり、これは $\mathrm{Poisson}(\lambda t)$ に収束する。重ならない時間枠は互いに素なスライスから構成されるため、それぞれのカウントは独立になる。この連続時間の事象ストリームが、レート λ の**ポアソン過程**（Poisson process）である。次節では、事象間の待ち時間が指数分布に従うことを見る。

二項分布と同様に、ポアソン分布の確率質量関数も $P(X = k+1)/P(X = k) = \lambda/(k+1)$ という単純な隣接項の比を満たす。これを使ってサーバーの確率を計算し、シミュレーションによる $100{,}000$ 分間のデータとモーメントを照合してみよう。

```
rng = np.random.default_rng()

def poisson_pmf(lam, kmax):
    """P(X=k) for k = 0..kmax, built from P(X=0) = e^-lam by the ratio lam/(k+1)."""
    pmf = [math.exp(-lam)]
    for k in range(kmax):
        pmf.append(pmf[-1] * lam / (k + 1))
    return np.array(pmf)

pmf = poisson_pmf(4.0, 60)
print(f'P(0) = {pmf[0]:.3f}, P(4) = {pmf[4]:.3f}, P(X>=10) = {pmf[10:].sum():.3f}')
counts = rng.poisson(4.0, 100000)
print(f'100,000 simulated minutes: mean {counts.mean():.2f}, '
      f'variance {counts.var(ddof=1):.2f}')
```

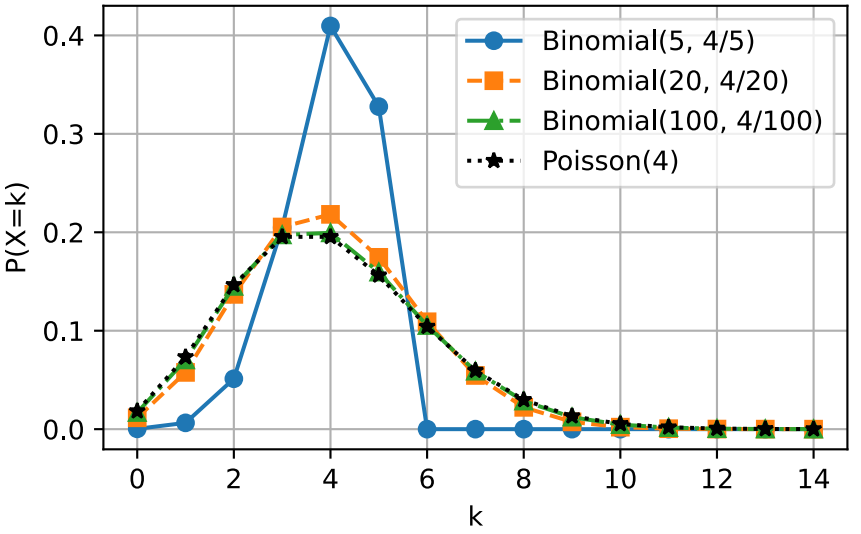
```
P(0) = 0.018, P(4) = 0.195, P(X>=10) = 0.008
100,000 simulated minutes: mean 4.01, variance 3.98
```

シミュレーションで得られたカウントは、平均も分散も 4 のすぐ近くに収まっている。

二項分布からポアソン分布への極限。$\mathrm{Binomial}(n, 4/n)$ はどれほど速く $\mathrm{Poisson}(4)$ に収束するのだろうか。両者の確率質量関数を重ねて描き、**全変動距離**（total variation distance）$\frac{1}{2}\sum_k |P_{\mathrm{Bin}}(k) - P_{\mathrm{Poi}}(k)|$ を用いてずれを測ってみよう。この距離は、任意の事象に対して 2 つの分布が割り当てる確率の差の最大値に等しい。

```
ns, ks = (5, 20, 100), np.arange(15)
d2l.plot(ks, [binomial_pmf(n, 4 / n, 14) for n in ns] + [poisson_pmf(4.0, 14)],
         xlabel='k', ylabel='P(X=k)', fmts=('o-', 's--', '^-.', 'k*:'),
         legend=[f'Binomial({n}, 4/{n})' for n in ns] + ['Poisson(4)'],
         figsize=(5, 3))
for n in (20, 100, 1000):
    tv = 0.5 * np.abs(binomial_pmf(n, 4 / n, 60) - poisson_pmf(4.0, 60)).sum()
    print(f'n = {n:4d}: total-variation distance {tv:.3f}')
```

```
n =   20: total-variation distance 0.056
n =  100: total-variation distance 0.010
n = 1000: total-variation distance 0.001
```



$n = 5$ では、二項分布は 5 件を超えることができず、ピークも目立って高すぎる。$n = 100$ になると、2 つの確率質量関数は見た目では区別がつかなくなる。全変動距離は 0.056（$n = 20$）から 0.010（$n = 100$）、さらに 0.001（$n = 1000$）へと低下し、ほぼ $1/n$ に比例して縮小する。

36.3.1.4 カテゴリカル分布と多項分布

K クラスの分類器や、語彙から次のトークンを予測する言語モデルは、3 つ以上の結果の中から 1 つを選ぶ。カテゴリカル分布とは、要するに K 面のコインである。

カテゴリカル確率変数は $\{1, \dots, K\}$ に値をとり、その確率はベクトル $\mathbf{p} = (p_1, \dots, p_K)$（ただし $p_k \geq 0$ かつ $\sum_k p_k = 1$）で指定される。

$$P(X = k) = p_k. \tag{36.38}$$

$K = 2$ のとき、これはベルヌーイ分布に帰着する。ニューラルネットワークでは確率を直接予測することはまれであり、代わりにネットワークは**ロジット**と呼ばれる制約のない実数スコア $z_1, \dots, z_K$ を出力し、**ソフトマックス**関数によってカテゴリカル分布へと変換する。

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}. \tag{36.39}$$

指数関数をとることで全成分が正になることが保証され、総和で割ることで和が 1 になる。たとえばロジット $\mathbf{z} = (2, 1, 0.1, -1)$ からは $\mathbf{p} \approx (0.638, 0.235, 0.095, 0.032)$ が得られる。真のクラスをワンホットベクトル $\mathbf{y}$ と書けば、観測されたラベルに割り当てられる負の対数確率は $-\log p_y = -\sum_k y_k \log p_k$ となり、これは 3.1.2 節の**ソフトマックス交差エントロピー**（softmax cross-entropy）損失にほかならない。二値交差エントロピーがベルヌーイ分布を当てはめるのとまったく同様に、ソフトマックス交差エントロピーは各ラベルにカテゴリカル分布を当てはめる。

離散一様分布。最も単純なカテゴリカル分布は、すべての結果に同じ確率を与えるものである。$\{1, \dots, n\}$ 上で $p_k = 1/n$ としたものが**離散一様**分布 Unif$\{1, \dots, n\}$ であり、公正なサイコロ

や一様ランダムに抽出されたインデックスを表す。その平均と分散は、よく知られた和の公式 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ と $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ から導かれる。

$$\begin{aligned}\mathbb{E}[X] &= \frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2},\\ \operatorname{Var}(X) &= \frac{(n+1)(2n+1)}{6} - \frac{(n+1)^2}{4}\\ &= \frac{(n+1)\bigl(2(2n+1) - 3(n+1)\bigr)}{12} = \frac{n^2-1}{12}.\end{aligned}$$

平均はちょうど中点に位置し、分散は結果が長さ $n-1$ の範囲に広がるため n^2 のオーダーで増大する。公正な 6 面サイコロの場合、平均は 3.5、分散は $35/12 \approx 2.92$ である。

多項分布。ベルヌーイ試行を n 回繰り返すと二項分布が得られるのと同じように、カテゴリカル分布から n 個の独立なサンプルを引き、各クラスの出現回数を集計すると**多項**分布（multinomial distribution）が得られる。カウントベクトル $\mathbf{x} = (x_1, \dots, x_K)$（制約 $\sum_k x_k = n$ を満たす）の確率質量関数は次で与えられる。

$$P(\mathbf{x}) = \frac{n!}{x_1! \cdots x_K!} \prod_{k=1}^{K} p_k^{x_k}. \tag{36.40}$$

二項分布のときと同様に、先頭の多項係数は n 回の抽出でカウント $\mathbf{x}$ を生じる並びの総数を数え上げており、後ろの積はいずれか 1 つの並びが生じる確率を与える。$K = 2$ のとき係数は $\binom{n}{x_1}$ に帰着し、二項分布が復元される（$13!/(9!\,4!) = 715$）。任意の 1 つのクラス k だけに注目すれば、各抽出はクラス k に入るか入らないかのいずれかであるため、$x_k \sim \text{Binomial}(n, p_k)$ となり、$\mathbb{E}[x_k] = np_k$、$\operatorname{Var}(x_k) = np_k(1-p_k)$ が成り立つ。全クラスを同時に考えると、カウントベクトルの平均は $n\mathbf{p}$、共分散行列は $n\bigl(\operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^{\top}\bigr)$ となる（演習問題 7 で導出する）。非対角成分がすべて負であることに注目しよう。抽出の総数 n が固定されているため、あるクラスの回数が増えれば必ず他のクラスの回数が減るからである。$n = 1$ のとき、カウントベクトルは単にカテゴリカル分布からの 1 回の抽出をワンホットエンコーディングで表したものになる。

例のロジットに対するソフトマックス確率を計算し、得られたカテゴリカル分布から 10,000 個のサンプルを抽出して経験頻度を確認してみよう。

```
z = np.array([2.0, 1.0, 0.1, -1.0])          # logits for K = 4 classes
p_cat = np.exp(z) / np.exp(z).sum()           # softmax
draws = rng.choice(4, size=10000, p=p_cat)
print('softmax p   =', p_cat.round(3))
print('frequencies =', (np.bincount(draws, minlength=4) / 10000).round(3))
```

```
softmax p   = [0.638 0.235 0.095 0.032]
frequencies = [0.638 0.24  0.091 0.031]
```

経験頻度は約 1 パーセントポイント以内の誤差で $\mathbf{p}$ と一致している。

36.3.2 連続分布

連続確率変数は、確率密度関数 $p(x) \geq 0$（ただし $\int p(x)\,dx = 1$）によって記述される。36.1 節で見たとおり、確率は密度曲線の下の面積であり、期待値と分散は積分で与えられる。まずは指

数分布から始めよう。ポアソン過程によってすでにその準備が整っているからである。

36.3.2.1 指数分布

先ほどの Web サーバーに戻ろう。ポアソン分布からは、1 分間に何件のリクエストが到着するかがわかる。一方、タイムアウトを設定するエンジニアは、同じトラフィックのストリームに対して密接に関連する別の問いを立てる。次のリクエストが到着するまでどれだけの時間待つことになるだろうか、という問いである。

興味深いことに、これら 2 つの問いは同じ答えを共有している。次のリクエストまで 1 分より長く待つことは、その 1 分間に到着するリクエストが 0 件であることとまったく同じ事象であるため、$P(\text{wait} > 1) = P(N = 0) = e^{-4} \approx 0.018$ となり、すでに計算したポアソン確率と一致する。Figure 36.15 に、両方の分布が単一の事象ストリームをどのように記述するかを示す。

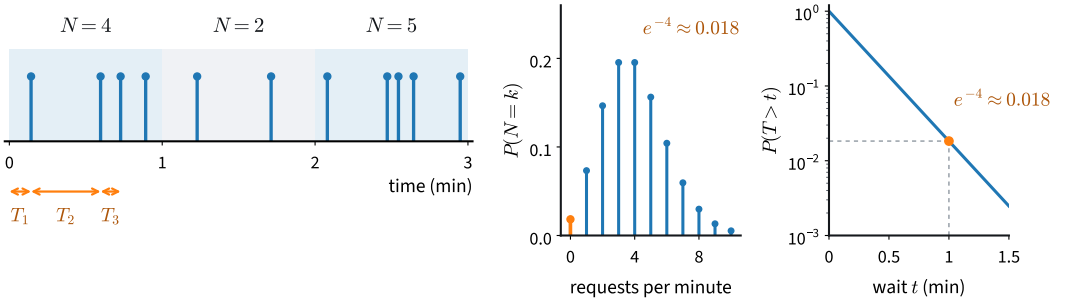


Figure 36.15: 毎分平均 4 件のレートで到着する 1 本のリクエストストリームを 2 通りの見方で捉えた図。(a) 3 分間にわたる事象発生時刻。長さ 1 の各時間枠には $N = 4$、2、5 件のリクエストが含まれ、T_1、T_2、T_3 は最初の 3 つの待ち時間を表す。(b) 1 分間のリクエスト数は Poisson(4) に従うため、静かな 1 分間の確率は $e^{-4} \approx 0.018$ である。(c) 待ち時間が t 分を超える確率は e^{-4t} となり、対数軸上で直線を描く。1 分より長く待つことと静かな 1 分間を観測することは同じ事象である。

まったく同じ推論が任意の時間長 $t \geq 0$ に当てはまる。レート λ のポアソン過程において、待ち時間 T（時刻 0 から最初の事象が起こるまでの時間）が t を超えるのは、区間 $[0, t]$ に事象が 1 つも含まれないとき、かつそのときに限る。$[0, t]$ 内のカウントは Poisson(λt) に従うため、次が得られる。

$$P(T > t) = P(N(t) = 0) = e^{-\lambda t}, \qquad t \geq 0.$$

したがって累積分布関数（cumulative distribution function）は $F(t) = 1 - e^{-\lambda t}$ であり、これを微分するとレート $\lambda > 0$ の**指数**分布の確率密度が得られる。

$$p(t) = \lambda e^{-\lambda t} \quad (t \geq 0), \qquad p(t) = 0 \quad (t < 0), \qquad F(t) = 1 - e^{-\lambda t}, \qquad T \sim \text{Exp}(\lambda). \tag{36.41}$$

言い換えれば、この密度は空の時間枠が生じる確率が減衰する速さを測っている。ポアソン過程では、任意の到着時刻より未来のスライスは新たな独立試行であるため、各事象の後の待ち時間も同じ分布に従う。すなわち、連続する到着の間隔は互いに独立な Exp(λ) 確率変数となる。

平均と分散。部分積分（integration by parts）を 2 回行えばモーメントはすぐに求まる。$\int_0^\infty u\,e^{-u}\,du = [-ue^{-u}]_0^\infty + \int_0^\infty e^{-u}\,du = 1$ および $\int_0^\infty u^2 e^{-u}\,du = [-u^2e^{-u}]_0^\infty + 2\int_0^\infty u\,e^{-u}\,du = 2$ を用い、$u = \lambda t$ と変数変換すると次を得る。

$$\mathbb{E}[T] = \int_0^\infty t\,\lambda e^{-\lambda t}\,dt = \frac{1}{\lambda}, \qquad \mathbb{E}[T^2] = \frac{2}{\lambda^2}, \qquad \mathrm{Var}(T) = \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}.$$

直感のとおり、到着レートが高いほど待ち時間は短くなる。毎分 4 件のリクエストがある場合、平均待ち時間は $\frac{1}{4}$ 分すなわち 15 秒である。中央値（median）m は $F(m) = \frac{1}{2}$ の解として $m = \log 2/\lambda \approx 10.4$ 秒と求まる。つまり平均は 15 秒であるにもかかわらず、全待ち時間の半分は約 10 秒未満で終わる。密度が $t = 0$ でピークをとり右側に長い裾を引くため、平均が中央値より大きく引き上げられるのである。

無記憶性。リクエストが来ないまま、すでに s 分間待ったとしよう。そこからさらに少なくとも t 分間待たなければならない確率はいくらだろうか。

$$P(T > s + t \mid T > s) = \frac{P(T > s + t)}{P(T > s)} = \frac{e^{-\lambda(s+t)}}{e^{-\lambda s}} = e^{-\lambda t} = P(T > t).$$

すでに待った時間は、残りの待ち時間について何の情報ももたらさない。$(0, \infty)$ 上のすべての連続分布のうち、この**無記憶**性（memoryless property）をもつのは指数分布だけである（演習問題 2 で証明する）。無記憶性はまた、このモデルが適用できる限界をも示している。使用に伴ってフィラメントが摩耗する電球は、古くなるほど次の 1 時間に切れる確率が高まるため、その寿命は指数分布には従わない。

ガンマ分布。では、時刻 T_k すなわち k 番目のリクエストが到着するまでの時間を問うたらどうなるだろうか。T_k は k 個の独立な指数待ち時間の和であるため、期待値の線形性と独立性から直ちに平均 k/λ と分散 k/λ^2 がわかる。その密度もポアソン過程から直接導かれる。k 番目の到着が時刻 t より後に起こるのは、高々 $k-1$ 個の事象しか $[0, t]$ 内に起こらないとき、かつそのときに限るので、次が成り立つ。

$$P(T_k > t) = P(N(t) \le k - 1) = \sum_{j=0}^{k-1} e^{-\lambda t}\frac{(\lambda t)^j}{j!}.$$

第 j 項を t に関して微分すると、積の微分法則により $-\lambda e^{-\lambda t}\frac{(\lambda t)^j}{j!} + \lambda e^{-\lambda t}\frac{(\lambda t)^{j-1}}{(j-1)!}$ が現れる。この第 2 項が 1 つ前の項の第 1 項と打ち消し合うことに注目しよう。和全体が望遠鏡和として相殺され、最後（$j = k-1$）の項の第 1 項だけが残る。

$$p(t) = -\frac{d}{dt}P(T_k > t) = \frac{\lambda^k\, t^{k-1} e^{-\lambda t}}{(k-1)!}, \qquad t > 0.$$

$k = 1$ のとき、指数分布の密度が復元される。この分布族を整数 k 以外にも拡張するため、$(k-1)!$ を**ガンマ関数**（gamma function）$\Gamma(z) = \int_0^\infty u^{z-1}e^{-u}\,du$（$z > 0$）に置き換える。部分積分により $\Gamma(z+1) = z\Gamma(z)$ が示され、$\Gamma(1) = 1$ であるから、すべての非負整数 n に対して $\Gamma(n+1) = n!$ が成り立つ。これにより、形状 $\alpha > 0$、レート $\beta > 0$ の**ガンマ分布**が次のように定義される。

$$p(t) = \frac{\beta^{\alpha}\, t^{\alpha-1} e^{-\beta t}}{\Gamma(\alpha)}, \qquad t > 0, \qquad T \sim \text{Gamma}(\alpha, \beta). \tag{36.42}$$

$u = \beta t$ と置換すると $p(t)$ の積分は $\Gamma(\alpha)/\Gamma(\alpha) = 1$ に帰着し、同じ置換から $\mathbb{E}[T] = \Gamma(\alpha+1)/(\beta\,\Gamma(\alpha)) = \alpha/\beta$ および $\text{Var}(T) = \alpha/\beta^2$ が得られる。$\alpha = k$、$\beta = \lambda$ のとき、これらは k 番目の待ち時間のモーメントと一致する。Web サーバーの例では、4 番目のリクエストは平均 1 分、標準偏差 0.5 分で到着し、$P(T_4 > 1) = P(N(1) \le 3) \approx 0.43$ となる。つまり平均は 4 件であっても、全体の 43% の分では到着するリクエストが 4 件未満になる。

サンプリング。 36.1 節で見たように、逆変換法（inverse-transform method）では一様乱数 U に対して可逆な累積分布関数 F の逆関数を計算することで、任意の連続分布からのサンプル $X = F^{-1}(U)$ に変換できる。指数分布では $F^{-1}(q) = -\log(1-q)/\lambda$ であるため、$T = -\log(1-U)/\lambda$ となる。$1-U$ と U は同じ分布に従うが、実装上は $1-U$ と書くことが重要である。乱数生成器（random number generator）は $[0,1)$ の値を返すため、$U = 0$ が起こり得て $-\log U$ が無限大へオーバーフローしてしまうからである。

ポアソン過程と指数分布の結びつきが寸分の隙もないものであれば、この構成を逆向きに動かせるはずである。すなわち、独立な指数分布の間隔を生成し、それを累積して到着時刻を作り、1 分あたりのポアソンカウントを復元できるはずだ。60,000 個の間隔をレート 4 で生成し、最初の 10,000 分間を調べて検証してみよう。

```
lam = 4.0                                       # requests per minute
gaps = -np.log(1 - rng.random(60000)) / lam     # inverse transform: Exp(lam)
times = np.cumsum(gaps)                         # event times, about 15,000 min
counts = np.bincount(times[times < 10000].astype(int), minlength=10000)
print(f'requests per minute: mean {counts.mean():.2f}, variance {counts.var(ddof=1):.2f}')
print(f'silent minutes {np.mean(counts == 0):.4f},   waits over 1 min '
      f'{np.mean(gaps > 1):.4f},   e^-4 = {math.exp(-4):.4f}')
print(f'mean wait {gaps.mean():.3f} min,   minutes with at most 3 requests '
      f'{np.mean(counts <= 3):.3f}')
```

```
requests per minute: mean 4.00, variance 4.05
silent minutes 0.0196,   waits over 1 min 0.0187,   e^-4 = 0.0183
mean wait 0.250 min,   minutes with at most 3 requests 0.435
```

シミュレーションによるカウントは確かに 4 に近い平均と分散をもっている。静かな 1 分間の割合と 1 分を超える待ち時間の割合はどちらも e^{-4} のすぐ近くにあり、リクエストが高々 3 件である分の割合もガンマ分布による予測 $P(T_4 > 1) \approx 0.43$ と一致している。指数分布に従う間隔を生成することで、ポアソンカウントが再現されるのである。

36.3.2.2 一様分布

本節のあらゆるサンプラは、$[0,1)$ から一様ランダムに抽出された 1 つの数値という共通の素材から出発する。この分布を任意の有限区間へと一般化したものが、連続一様分布（continuous uniform distribution）である。

$a < b$ を満たす区間 $[a,b]$ 上の**一様**分布 $U(a,b)$ は一定の確率密度をもち、その値は積分すると 1 になるという条件から定まる。

$$p(x) = \begin{cases} \dfrac{1}{b-a} & x \in [a,b], \\ 0 & \text{otherwise.} \end{cases} \tag{36.43}$$

同じ長さの部分区間にはすべて同じ確率、すなわちその長さを $b-a$ で割った値が割り当てられる。両方のモーメントは直接積分すれば求まる。分散については、区間の中点が原点になるよう中心化し、$-c$ から c まで（ただし $c=\frac{b-a}{2}$）の区間とすると計算がすっきりする。

$$\mathbb{E}[X]=\int_a^b \frac{x}{b-a}\,dx=\frac{b^2-a^2}{2(b-a)}=\frac{a+b}{2},\qquad \mathrm{Var}(X)=\int_{-c}^{c}\frac{y^2}{2c}\,dy=\frac{c^2}{3}=\frac{(b-a)^2}{12}.$$

特に、$U(0,1)$ は平均 $\frac{1}{2}$、分散 $\frac{1}{12}$ をもち、$U(-\sqrt{3},\sqrt{3})$ は分散 $(2\sqrt{3})^2/12=1$ をもつ。この公式が離散一様分布の連続極限になっていることに注目しよう。間隔 h で等間隔に並ぶ等確率の n 個の点の分散は $h^2(n^2-1)/12$ であり、$nh=b-a$ を保ちながら点が $[a,b]$ を埋め尽くす極限で $(b-a)^2/12$ に収束する。逆変換を通じれば、先ほど指数分布で行ったように $U(0,1)$ を任意の連続分布へと変形できる。見た目には、一様分布は平坦な密度と不連続な端をもち、裾をまったくもたないため、ガウス分布とは似ても似つかない。だからこそ、以下で中心極限定理が一様分布の和を釣鐘曲線へと変えていく様子を観察するうえで理想的なテストケースとなる。

36.3.2.3 ガウス分布

隣接項の比を使えば任意の二項確率を高精度に計算できるが、それだけでは分布全体がどのような形をしているかは見えてこない。ピーク付近の形状を表す単純な閉じた式は存在するのだろうか。驚くべきことにそれは存在し、しかもそこにはコインの平均と分散しか現れない。ここから、確率論と機械学習において最も重要な分布へと導かれる。

平均付近での対数比。$q=1-p$ とおき、カウントを平均からのずれ x によって測ることにすると $k=np+x$ となる。これを二項分布の隣接項の比に代入すると次を得る。

$$\frac{P(X=k+1)}{P(X=k)}=\frac{(n-k)\,p}{(k+1)\,q}=\frac{(nq-x)\,p}{(np+x+1)\,q}=\frac{1-x/(nq)}{1+(x+1)/(np)}.$$

x が標準偏差 $\sqrt{npq}$ 程度の大きさであり n が大きいとき、$x/(nq)$ と $(x+1)/(np)$ はどちらも微小である。対数をとり、線形近似 $\log(1+\epsilon)\approx\epsilon$ を適用すると次のようになる。

$$\log\frac{P(X=k+1)}{P(X=k)}\approx-\frac{x}{nq}-\frac{x+1}{np}=-\frac{x}{npq}-\frac{1}{np},$$

ここで $\frac{1}{nq}+\frac{1}{np}=\frac{p+q}{npq}=\frac{1}{npq}$ を使って分数をまとめた。主要項を見ると、$\log P(X=k)$ から $\log P(X=k+1)$ への 1 ステップの変化量は、平均からの距離 x に対して傾き $-1/(npq)$ で線形に変化する。1 ステップの差分が線形に変化するのはどのような関数だろうか。放物線である。実際、放物線 $-x^2/(2npq)$ の変化量は

$$-\frac{(x+1)^2-x^2}{2npq}=-\frac{x}{npq}-\frac{1}{2npq}$$

と表される（x が 1 増えるとき）。これは微小な定数 $(p-q)/(2npq)$ の違いを除けば $\log P$ のステップ変化と一致する。この定数項は公正なコインでは完全に消え、一般の場合でもピーク周辺の $\sqrt{npq}$ ステップにわたって累積してなお $O(1/\sqrt{n})$ にしかならない。以上から、次の結論が得られる。

$$\log P(X = np + x) \approx C - \frac{x^2}{2npq}$$

これはある定数 C に対して、平均から数標準偏差以内にあるすべてのカウント $k = np + x$ で成り立ち、np が整数であるかどうかによらない（13 回のコイン投げでは $np = 6.5$ である）。両辺の指数をとれば、二項分布はピーク付近で釣鐘曲線 $e^{C} e^{-x^2/(2npq)}$ の形をとることがわかる。その中心は平均にあり、幅は標準偏差 $\sqrt{npq}$ で決まる。高次の剰余項を厳密に評価せず落としたため、これは厳密な証明ではなく導出のスケッチであるが、これを厳密化したものが中心極限定理の最古の形態である**ド・モアブル・ラプラスの定理**（de Moivre–Laplace theorem）である。

正規化定数。定数 C を決めるには、確率の総和が 1 にならなければならないという事実を使う。整数のカウントは間隔 1 で並んでいるため、すべての k にわたる釣鐘曲線の和は積分 $e^{C}\int_{-\infty}^{\infty} e^{-x^2/(2npq)}\,dx$ でよく近似される。したがって問題はすべて $I = \int_{-\infty}^{\infty} e^{-x^2/2}\,dx$ の評価に帰着する。一見すると、$e^{-x^2/2}$ は初等的な原始関数をもたないため、この積分は絶望的に思える。ここで使われる古典的な技巧は、代わりに I^2 を計算することである。平面上の二重積分とみなして極座標 $x = r\cos\theta$、$y = r\sin\theta$（面積要素 $dx\,dy = r\,dr\,d\theta$）に移れば、被積分関数は初等的に積分できるようになる。

$$\begin{aligned} I^2 &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)/2}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} e^{-r^2/2}\,r\,dr\,d\theta \\ &= 2\pi\left[-e^{-r^2/2}\right]_0^{\infty} = 2\pi. \end{aligned}$$

ゆえに $I = \sqrt{2\pi}$ である。極座標変換のヤコビ行列から生じる余分な因子 r のおかげで、$r\,e^{-r^2/2}$ は初等的な原始関数 $-e^{-r^2/2}$ をもつようになる（$e^{-x^2/2}$ 単体にはこれがない）。続いて $x = \sqrt{npq}\,u$ と変数変換（change of variables）すれば $\int_{-\infty}^{\infty} e^{-x^2/(2npq)}\,dx = \sqrt{2\pi npq}$ となるため、$e^{C} = 1/\sqrt{2\pi npq}$ と定まり、次式が得られる。

$$P(X = np + x) \approx \frac{1}{\sqrt{2\pi npq}}\, e^{-x^2/(2npq)}.$$

公正なコインを 13 回投げる例でこれを試してみよう。$npq = 3.25$ であり、$k = 9$ は $x = 2.5$ に対応するため、釣鐘曲線による近似値は $e^{-2.5^2/6.5}/\sqrt{2\pi\cdot 3.25} \approx 0.0846$ となり、これほど小さな n にもかかわらず厳密値 0.0873 に驚くほど近い。以下では、n が大きくなるにつれて釣鐘曲線による近似がどれほど速く精密になるかを測定する。

```
    for n in (13, 100, 1000):
        k = np.arange(n + 1)
        bell = np.exp(-(k - n / 2)**2 / (n / 2)) / np.sqrt(np.pi * n / 2)
        exact = binomial_pmf(n, 0.5, n)
        peak = n // 2
        print(f'n = {n:4d}: relative error at the peak {abs(bell[peak] / exact[peak] - 1):.2%}')
    print(f'n = 13, k = 9: bell {np.exp(-2.5**2 / 6.5) / np.sqrt(np.pi * 6.5):.4f},'
          f' exact {binomial_pmf(13, 0.5, 13)[9]:.4f}')
```

```
n =   13: relative error at the peak 1.66%
n =  100: relative error at the peak 0.25%
```

```
n = 1000: relative error at the peak 0.03%
n = 13, k = 9: bell 0.0846, exact 0.0873
```

$n = 13$、$k = 9$ での釣鐘曲線の近似値は約 3% 低いだけであり、ピークにおける相対誤差は 1.7%（$n = 13$）から 0.25%（$n = 100$）、0.03%（$n = 1000$）へと低下し、ほぼ $1/n$ のオーダーで縮小する。

標準ガウス分布。指数の放物線の形から、自然なスケール変換が示唆される。標準化されたカウント $Y^{(n)} = (X - np)/\sqrt{npq}$ は、平均 0、分散 1 をどの n に対してももち、そのとり得る値は間隔 $1/\sqrt{npq}$ で並んでいる。この単位で書き直すと、釣鐘近似は $P(X = k) \approx \phi(y)/\sqrt{npq}$（ただし $y = (k - np)/\sqrt{npq}$）となる。これは滑らかな密度 $\phi(y)$ にグリッド間隔を掛けたものであり、この関数は次で定義される。

$$\phi(y) = \frac{1}{\sqrt{2\pi}}\, e^{-y^2/2}.$$

これが標準化された二項分布の極限分布である**標準ガウス分布**（standard Gaussian）$\mathcal{N}(0,1)$ の密度である。$I = \sqrt{2\pi}$ より全積分は 1 となり、原点周りの対称性から平均は 0 である。分散については、$u = x$、$dv = x\,e^{-x^2/2}\,dx$ とおいて部分積分すると次が得られる。

$$\int_{-\infty}^{\infty} x^2\,\frac{e^{-x^2/2}}{\sqrt{2\pi}}\,dx = \frac{1}{\sqrt{2\pi}}\Big(\big[-x\,e^{-x^2/2}\big]_{-\infty}^{\infty} + \int_{-\infty}^{\infty} e^{-x^2/2}\,dx\Big) = 0 + 1 = 1.$$

したがって標準ガウス分布の分散は 1 である。Y を μ だけ平行移動し、$\sigma > 0$ 倍にスケール変換すると $X = \mu + \sigma Y$ が作られる。36.2 節の変数変換の公式により、その密度は $\phi((x-\mu)/\sigma)/\sigma$ となり、これが一般の**ガウス**（または正規）分布を定義する。

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\,\exp\!\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big), \qquad X \sim \mathcal{N}(\mu, \sigma^2). \tag{36.44}$$

$X = \mu + \sigma Y$ であるから、平均は μ、分散は σ^2 となる。つまり 2 つのパラメータがそのまま 1 次と 2 次のモーメントになっており、σ が釣鐘の幅を制御する。

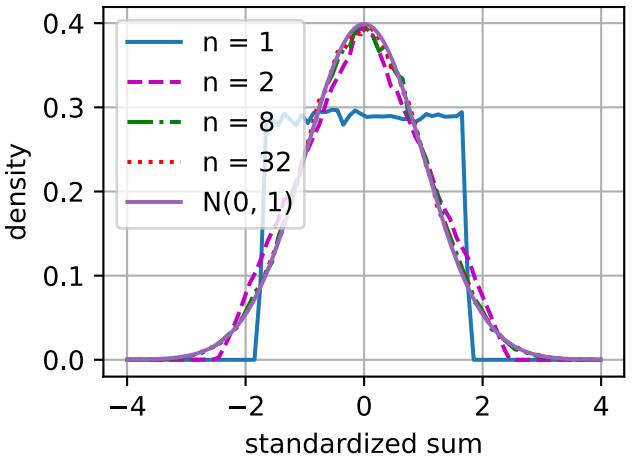
中心極限定理。ここまでの導出を振り返ると、最終的な分布の形にはコイン投げ固有の性質は平均と分散以外ほとんど入り込んでいなかった。実際、一般の定理に必要なのはそれだけである。$X_1, X_2, \ldots$ を平均 μ と有限の分散 $0 < \sigma^2 < \infty$ をもつ独立同分布（independent and identically distributed）の確率変数とし、$S_n = \sum_{i=1}^{n} X_i$ とおく。このとき、すべての実数 y に対して次が成り立つ。

$$\lim_{n\to\infty} P\Big(\frac{S_n - n\mu}{\sigma\sqrt{n}} \le y\Big) = \Phi(y) = \int_{-\infty}^{y} \phi(u)\,du. \tag{36.45}$$

言葉で述べれば、有限の分散をもつ**いかなる**分布であっても、その独立なコピーを多数足し合わせて標準化した和は標準ガウス分布に収束する。各項がベルヌーイ変数の場合がド・モアブル・ラプラスの定理であり、一般の場合の証明については (Wasserman 2013) を参照されたい。極限において生き残るのは個々の項の平均と分散だけであり、センサーノイズや測定ノイズのように多数の小さな独立した影響の積み重ねから生じる量がこれほど確実にガウス分布に従う理由はここにある。

一様分布の和。中心極限定理は、実際に動く様子を観察するとさらに印象的である。完全に平坦で鋭い端をもち裾がまったくない $U(0,1)$ から出発し、n 個の独立なサンプルの和をとり、平均 $n/2$ と標準偏差 $\sqrt{n/12}$ で標準化して、得られるヒストグラムを ϕ と比較してみよう。

```
ns, x = (1, 2, 8, 32), np.arange(-4, 4, 0.01)
curves = []
for n in ns:                                    # standardized sums of n uniforms
    s = (rng.random((100000, n)).sum(1) - n / 2) / np.sqrt(n / 12)
    hist, edges = np.histogram(s, bins=80, range=(-4, 4), density=True)
    curves.append(np.interp(x, (edges[:-1] + edges[1:]) / 2, hist))
curves.append(np.exp(-x**2 / 2) / np.sqrt(2 * np.pi))     # N(0, 1) density
d2l.plot(x, curves, xlabel='standardized sum', ylabel='density',
         legend=[f'n = {n}' for n in ns] + ['N(0, 1)'])
```



$n = 1$ では一様分布の平坦な台地が見え、$n = 2$ では鋭い三角形になる。$n = 8$ の時点ですでにヒストグラムはガウス分布の釣鐘曲線をよく捉えており、$n = 32$ になるとプロットの解像度では 2 つの曲線の見分けがつかなくなる。

ただし、分散が有限であるという仮定を外すことはできない。36.1 節で見たように、コーシー分布からのサンプルの平均はどれほど多くのサンプルを平均しても決して落ち着かないのに対し、n 個の標準ガウス変数の平均は $\mathcal{N}(0, 1/n)$ に従い 0 の周りに鋭く集中する。

ガウスノイズと二乗誤差。回帰ネットワークが予測値 $\hat{y}$ を出力し、観測ラベルをガウスノイズを伴う $y \sim \mathcal{N}(\hat{y}, \sigma^2)$ としてモデル化するとき、y に割り当てられる負の対数確率は $-\log p(y) = (y-\hat{y})^2/(2\sigma^2) + \frac{1}{2}\log(2\pi\sigma^2)$ となる。正のスケール因子と定数項の違いを除けば、これは二乗誤差損失である（2.1.3 節）。

36.3.2.4 ラプラス分布

二乗誤差は二次関数的に増大するため、残差（residual）10 には残差 1 の 100 倍もの重いペナルティが課され、少数の外れ値（outlier）によって当てはめが大きく引きずられてしまう。線形にしか増大しない損失である絶対誤差は、より裾の重いノイズ分布に対応する（2.1.3.3 節）。

これを構成するには、$P(S = 1) = P(S = -1) = \frac{1}{2}$ を満たす偏りのないランダムな符号 S と、それと独立な平均 $b > 0$（レート $1/b$）の指数確率変数 E をとり、$X = \mu + SE$ とおけばよ

い。$S=1$ のときは常に X は μ より大きく $X-\mu=E$ となるため、$x>\mu$ における密度は $\frac{1}{2}\cdot\frac{1}{b}e^{-(x-\mu)/b}$ となる。対称性から、μ より小さい側でも $x-\mu$ を $|x-\mu|$ に置き換えた同じ式が成り立つ。こうして**ラプラス**分布が得られる。

$$p(x)=\frac{1}{2b}\,e^{-|x-\mu|/b},\qquad X\sim\mathrm{Laplace}(\mu,b). \tag{36.46}$$

分布の左右それぞれの半分が、指数分布の半分の確率質量を担っている。密度は μ の周りで対称であり、その点で鋭く尖ったピークをとり、両側に向かって指数関数的に減衰する。

平均と分散。両方のモーメントは $X=\mu+SE$ から直ちに導かれる。独立性により $\mathbb{E}[SE]=\mathbb{E}[S]\,\mathbb{E}[E]=0$ となるため、$\mathbb{E}[X]=\mu$ である。また常に $S^2=1$ であるから、分散は単に指数分布の二次モーメント（second moment）$2/\lambda^2$（ただし $\lambda=1/b$）に一致する。

$$\mathrm{Var}(X)=\mathbb{E}\big[S^2E^2\big]=\mathbb{E}[E^2]=2b^2.$$

ラプラスノイズと絶対誤差。回帰ネットワークがラベルを $y\sim\mathrm{Laplace}(\hat{y},b)$ とモデル化するとき、負の対数確率は $-\log p(y)=|y-\hat{y}|/b+\log 2b$ となり、スケールと定数の違いを除いて絶対誤差に一致する。モデルの重みに対する事前分布（prior distribution）としてラプラス密度を用いれば、LASSO の ℓ_1 ペナルティ $\|\mathbf{w}\|_1$ が導かれる (Tibshirani 1996)。さらに、位置パラメータ μ をデータ $x_1,\dots,x_m$ に当てはめる際、ラプラス確率の最大化は $\sum_i|x_i-\mu|$ の最小化となり、これは標本中央値で達成される。たった 1 つの異常点が標本平均（sample mean）をいくらでも遠くへ引きずり得るのに対し、中央値は高々 1 順位しかずれない。

裾の比較。@fig-mdl-prob-continuous-pdfs に、同じ分散をもつようスケールを揃えた一様分布、ガウス分布、ラプラス分布の密度を比較する。分散を $\sigma^2=2b^2$ で揃えたとき、ラプラス確率変数が中心から標準偏差の 3 倍を超えて離れる確率は $P(|X-\mu|>3\sigma)=e^{-3\sigma/b}=e^{-3\sqrt{2}}\approx 0.014$ であり、ガウス分布の 0.0027 と比べて約 5 倍もの確率質量が裾に残る。

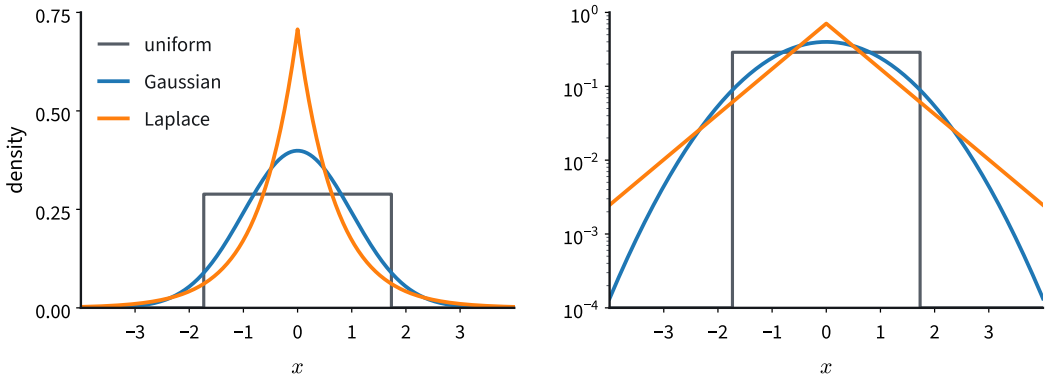


Figure 36.16: 分散 1 をもつ 3 つの密度（$[-\sqrt{3},\sqrt{3}]$ 上の一様分布、標準ガウス分布、$b=1/\sqrt{2}$ のラプラス分布）。(a) 線形スケールでの密度。(b) 対数スケールでの同じ密度。ガウス分布は放物線、ラプラス分布は 2 本の直線、一様分布は $\pm\sqrt{3}$ で途切れる平坦な台地となる。標準偏差の 3 倍より外側では、ラプラス分布はガウス分布の約 5 倍の裾確率をもつ。

ラプラス分布をその構成法（ランダムな符号と指数乱数の積）から直接サンプリングし、標準偏差の 3 倍を超えるサンプルがどれほどの頻度で現れるかを数えて、この裾の比較を確かめてみよう。

```
b, m = 1.0, 200000
lap = rng.choice([-1.0, 1.0], m) * rng.exponential(b, m)   # S * E
sigma = math.sqrt(2) * b                                    # matched sd
gau = rng.normal(0.0, sigma, m)
print(f'sample variances: Laplace {lap.var(ddof=1):.2f}, '
      f'Gaussian {gau.var(ddof=1):.2f}   (2b^2 = 2)')
print(f'P(|X| > 3 sd): Laplace {np.mean(np.abs(lap) > 3 * sigma):.4f}, '
      f'Gaussian {np.mean(np.abs(gau) > 3 * sigma):.4f}')
```

```
sample variances: Laplace 2.01, Gaussian 2.00   (2b^2 = 2)
P(|X| > 3 sd): Laplace 0.0144, Gaussian 0.0026
```

どちらの標本分散（sample variance）もほぼ 2 になり、標準偏差の 3 倍を超える裾の割合はそれぞれ 0.014 と 0.003 の近くに収まっている。

36.3.3 多変量ガウス分布

深層学習において、ランダムな量がスカラーであることはまれである。画像のノイズ、生成モデルにおける潜在表現、入力のバッチ全体にわたる予測は、いずれも各座標が相互作用するベクトルである。多変量ガウス分布（multivariate Gaussian）は、相関をもつベクトル上の分布でありながら、必要となるほぼすべての操作のもとで解析的に扱いやすい唯一の分布である。通し例として、それぞれの分散が 2、共分散が 1 である 2 座標の組を考えよう。

$$\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

36.3.3.1 等高線と固有ベクトル

$\mathbb{R}^d$ 上の**多変量ガウス分布** $\mathcal{N}(\mu, \Sigma)$ は、平均ベクトル μ と対称な正定値（positive definite）共分散行列 Σ によってパラメータ化される。その密度は次の式で与えられる。

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\!\left(-\tfrac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1} (\mathbf{x}-\mu)\right), \tag{36.47}$$

ここで $|\Sigma|$ は Σ の行列式（determinant）を表す。$d = 1$ のとき、これは Equation 36.44 に帰着する。$\mathbf{x}$ は指数部の二次形式（quadratic form）のみを通じて密度に現れるため、その二次形式が一定となる等位集合上で密度は一定になる。これらの等高線はどのような幾何学的形状をしているのだろうか。

通し例の手計算。 $\mu = \mathbf{0}$ とし、座標軸を 45° 回転して和と差の座標 $u = (x_1 + x_2)/\sqrt{2}$ および $w = (x_1 - x_2)/\sqrt{2}$ に移ろう。これらの分散と共分散は Σ の成分から直接求まる。

$$\operatorname{Var}(u) = \frac{2 + 2 + 2 \cdot 1}{2} = 3, \qquad \operatorname{Var}(w) = \frac{2 + 2 - 2 \cdot 1}{2} = 1, \qquad \operatorname{Cov}(u, w) = \frac{2 - 2}{2} = 0.$$

この回転後の座標系では、共分散行列は対角行列になる。新しい軸は単に Σ の固有ベクトル（eigenvector）$\mathbf{v}_1 = (1, 1)/\sqrt{2}$ と $\mathbf{v}_2 = (1, -1)/\sqrt{2}$ であり、対応する固有値（eigenvalue）は $\gamma_1 = 3$ と $\gamma_2 = 1$ である（実際、$\Sigma \mathbf{v}_1 = (3, 3)/\sqrt{2} = 3\mathbf{v}_1$ が成り立つ）。

一般の場合。スペクトル定理により、任意の対称行列（symmetric matrix）は固有ベクトルからなる正規直交基底（orthonormal basis）をもち、$\Sigma = \mathbf{V}\,\mathrm{diag}(\gamma_1, \dots, \gamma_d)\mathbf{V}^\top$（ただし $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$）と表せる。また正定値性から、すべての固有値は $\gamma_i > 0$ を満たす。固有座標 $\mathbf{u} = \mathbf{V}^\top(\mathbf{x} - \mu)$ に移ると、二次形式は単純な重み付き二乗和に変わる。

$$(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu) = \mathbf{u}^\top\,\mathrm{diag}(\gamma_1^{-1}, \dots, \gamma_d^{-1})\,\mathbf{u} = \sum_{i=1}^{d} \frac{u_i^2}{\gamma_i}.$$

この和が r^2 に等しい等位集合は、固有ベクトル $\mathbf{v}_i$ に沿った楕円体であり、その半軸長は $r\sqrt{\gamma_i}$ となる。したがって、任意の多変量ガウス分布の等高線は、Σ の固有ベクトルを主軸とし、標準偏差 $\sqrt{\gamma_i}$ に比例して引き伸ばされた楕円体になる。通し例では、45° 傾いた軸比 $\sqrt{3} : 1$ の楕円になる（Figure 36.17 左）。また $|\Sigma| = \prod_i \gamma_i$ であるから、密度は $\mathbf{u}$ において d 個の 1 次元ガウス分布の積へと分解する。すなわち固有ベクトルの方向に沿って、各座標は互いに独立で分散 γ_i をもつ。特別な場合として $\Sigma = \sigma^2\mathbf{I}$ のときは、等高線は球面となり、元の座標が最初から互いに独立である。

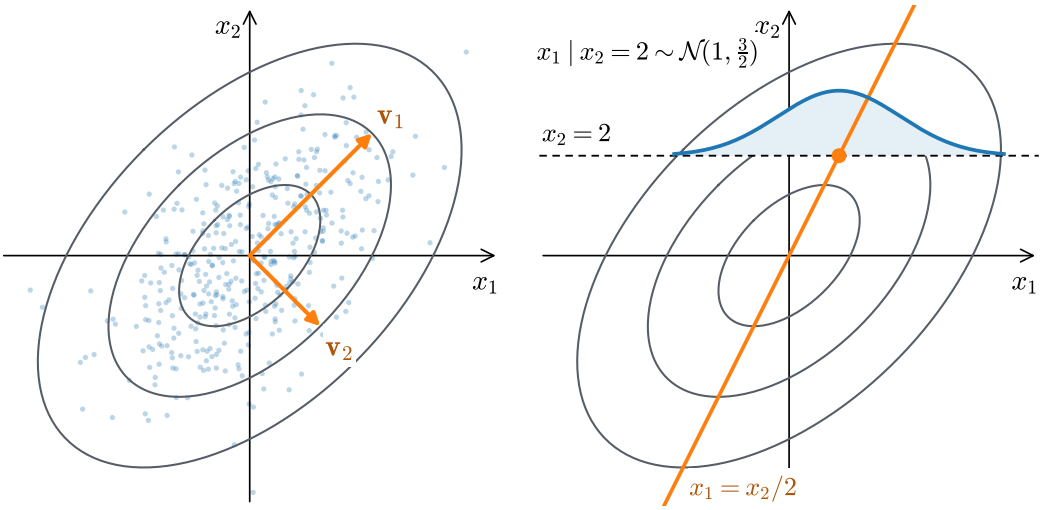


Figure 36.17: 行が $(2, 1)$ と $(1, 2)$ である共分散行列 Σ をもつ通し例の 2 変量ガウス分布。(a) 400 個のサンプルと、指数部の二次形式が 1、4、9 に等しい 3 本の密度等高線。楕円の軸は Σ の固有ベクトルに沿い、半軸長は $\sqrt{\gamma_1} = \sqrt{3}$ と $\sqrt{\gamma_2} = 1$ に比例する。矢印 $\mathbf{v}_1$ と $\mathbf{v}_2$ は中央の等高線まで達している。(b) $x_2 = 2$ を観測すると破線に沿って密度が切り出され、x_1 の条件付き密度 $\mathcal{N}(1, \frac{3}{2})$ が得られる。すべてのスライスにわたり、条件付き平均は直線 $x_1 = x_2/2$ 上に乗り、条件付き分散は $3/2$ で一定に保たれる。

1 つ注意点を早めに挙げておこう。高次元では、平均付近に質量が集中した釣鐘という低次元の直感は誤解を招きやすい。$\mathbb{R}^d$ における標準ガウスベクトルのノルムはほぼ常に $\sqrt{d}$ の近くにあり、独立に引いた 2 つのサンプルはほぼ直交（orthogonal）する。これら 2 つの事実は 36.9 節で証明する。

36.3.3.2 線形変換とサンプリング

実際には、既存のガウスベクトルから座標を混ぜ合わせたり部分空間（subspace）へ射影（projection）したりして、新たなガウスベクトルを日常的に構成する。これらはいずれも線形変

換（linear map）であり、ガウス分布が非常に扱いやすい大きな理由の 1 つは、線形変換がガウス性を保つことにある。まずはサンプリングに必要な可逆正方行列の場合についてこれを証明しよう。長方形行列による射影については、ガウスベクトルの任意の部分ベクトルがそれ自体ガウス分布に従うことを次項で示してから導く。

命題（可逆な線形変換）。 $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ を $\mathbb{R}^d$ 上の確率ベクトルとし、$\mathbf{A}$ を可逆な $d \times d$ 行列、$\mathbf{b} \in \mathbb{R}^d$ とする。このとき次が成り立つ。

$$\mathbf{A}\mathbf{x} + \mathbf{b} \sim \mathcal{N}\big(\mathbf{A}\mu + \mathbf{b},\ \mathbf{A}\Sigma\mathbf{A}^{\top}\big).$$

証明。 期待値の線形性により、上記の平均と共分散は有限の二次モーメントをもつ任意の分布で成り立つ。実際、$\mathbb{E}\big[(\mathbf{A}\mathbf{x} - \mathbf{A}\mu)(\mathbf{A}\mathbf{x} - \mathbf{A}\mu)^{\top}\big]$ は $\mathbf{A}\,\mathbb{E}\big[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^{\top}\big]\,\mathbf{A}^{\top} = \mathbf{A}\Sigma\mathbf{A}^{\top}$ に等しい。あとは変換後の密度がガウス密度であることだけを確かめればよい。36.2 節の変数変換の公式により、$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ の密度は $p_{\mathbf{y}}(\mathbf{y}) = p_{\mathbf{x}}(\mathbf{A}^{-1}(\mathbf{y}-\mathbf{b}))/|\det \mathbf{A}|$ となる。$\mathbf{c} = \mathbf{y} - \mathbf{A}\mu - \mathbf{b}$ とおけば、指数部は $-\tfrac{1}{2}\mathbf{c}^{\top}\mathbf{A}^{-\top}\Sigma^{-1}\mathbf{A}^{-1}\mathbf{c} = -\tfrac{1}{2}\mathbf{c}^{\top}(\mathbf{A}\Sigma\mathbf{A}^{\top})^{-1}\mathbf{c}$ と書き直され、正規化因子も $|\det \mathbf{A}|\,|\Sigma|^{1/2} = |\mathbf{A}\Sigma\mathbf{A}^{\top}|^{1/2}$ を満たす。したがって $p_{\mathbf{y}}$ は主張どおりの平均と共分散をもつガウス密度である。$\blacksquare$

ここでは可逆性が不可欠である。もし $\mathbf{A} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ であれば、出力の 2 つの座標は常に同じ値になり、$\mathbf{A}\Sigma\mathbf{A}^{\top} = \left(\begin{smallmatrix} 6 & 6 \\ 6 & 6 \end{smallmatrix}\right)$ は特異行列になる。結果として得られる確率ベクトル（random vector）は完全に直線 $y_1 = y_2$ 上に集中し、平面上の 2 次元密度をもたない。

行列因子によるサンプリング。 手元の乱数生成器が独立な標準正規乱数しか生成できないとき、どのようにして $\mathcal{N}(\mu, \Sigma)$ からサンプリングすればよいだろうか。独立に引いた d 個の乱数を並べて $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ を作れば、その同時密度は ϕ の d 個の積になる。$\mathbf{L}\mathbf{L}^{\top} = \Sigma$ を満たす任意の正方行列 $\mathbf{L}$ に対し、$\mathbf{L}$ は必ず可逆行列（invertible matrix）になり（Σ が正定値行列であるため）、先ほどの命題から $\mathbf{x} = \mu + \mathbf{L}\mathbf{z}$ は平均 μ、共分散 $\mathbf{L}\,\mathbf{I}\,\mathbf{L}^{\top} = \Sigma$ のガウス分布に従うことが保証される。

要するに、行列因子を掛けることで独立なノイズが相関をもつノイズへと変換される。乱数 $\mathbf{z}$ を固定したとき $\mathbf{x}$ は μ と $\mathbf{L}$ の滑らかで微分可能な関数になるため、この同じ構成法は変分オートエンコーダ（variational autoencoder、VAE）が確率的な層を通して逆伝播を行うための**再パラメータ化トリック**（reparameterization trick）そのものでもある（34.7.3 節）。最も高速に計算できる因子は**コレスキー因子**（Cholesky factor）である。任意の対称正定値行列は、対角成分が正の下三角行列 $\mathbf{L}$ を用いて $\Sigma = \mathbf{L}\mathbf{L}^{\top}$ と一意に分解できる。通し例では、$\mathbf{L}\mathbf{L}^{\top}$ の各成分を Σ に順番に一致させることで、まず $L_{11} = \sqrt{2}$、次に $L_{21} = \Sigma_{21}/L_{11} = 1/\sqrt{2}$、最後に $L_{22} = \sqrt{\Sigma_{22} - L_{21}^2} = \sqrt{3/2}$ が求まる（実際、$L_{21}^2 + L_{22}^2 = \tfrac{1}{2} + \tfrac{3}{2} = 2$ である）。

固有因子 $\mathbf{V}\,\mathrm{diag}(\sqrt{\gamma_1}, \dots, \sqrt{\gamma_d})$ など、他のどの平方根因子を使っても同様にうまくいく。演習問題 5 で示すように、同じ Σ に対する任意の 2 つの因子は直交行列（orthogonal matrix）を掛ける違いしかもたない。実際のライブラリのサンプラは実装ごとに異なり、多くはコレスキー因子を用いる一方、NumPy の `multivariate_normal` はデフォルトで特異値分解（singular value decomposition）から得られる因子を用いている。

ライブラリのサンプラを呼び出す代わりに、仕組みが動く様子を確認できるようコレスキー分解の手順を直接実装し、標本共分散とその固有値の両方を検証してみよう。

```
Sigma = np.array([[2.0, 1.0], [1.0, 2.0]])
L = np.linalg.cholesky(Sigma)                  # Sigma = L L^T
x = rng.standard_normal((400000, 2)) @ L.T     # each row is x = L z
C = np.cov(x.T)
```

```
    print('L =', L.round(3).tolist())
    print('sample covariance =', C.round(2).tolist())
    print('its eigenvalues   =', np.linalg.eigvalsh(C).round(2))
```

```
L = [[1.414, 0.0], [0.707, 1.225]]
sample covariance = [[2.0, 1.0], [1.0, 2.0]]
its eigenvalues   = [1. 3.]
```

コレスキー因子は手計算の結果と一致し、経験共分散は Σ と約 0.01 以内で一致し、その固有値も 1 と 3 の近くに収まっている。

36.3.3.3 条件付け

ガウスベクトルの一部の座標を観測し、残りの座標を推論したい場面は頻繁にある。通し例において $x_2 = 2$ を観測したとき、x_1 はどのあたりにあると期待すべきだろうか。ベクトルを 2 つのブロック $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ に分割し、それに合わせて平均と共分散もブロック分割しよう。

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}.$$

ガウス分布の条件付けの背後にある仕掛けは幾何学的で明快である。$\mathbf{x}_1$ から、$\mathbf{x}_2$ に基づく $\mathbf{x}_1$ の線形予測を引けばよい。すると残るのは、$\mathbf{x}_2$ についての情報を一切もたない残差である。

命題（残差と周辺分布）。 $\mathbf{B} = \Sigma_{12}\Sigma_{22}^{-1}$ とおく。このとき残差 $\mathbf{r} = \mathbf{x}_1 - \mathbf{B}\mathbf{x}_2$ は $\mathbf{x}_2$ と独立であり、第 2 ブロックの周辺分布（marginal distribution）は $\mathbf{x}_2 \sim \mathcal{N}(\mu_2, \Sigma_{22})$ となる。

証明。 Σ_{22} は正定値行列の主対角ブロックであるため正定値であり、したがって可逆である。構成法より、残差は $\mathbf{x}_2$ と無相関になる。実際、$\operatorname{Cov}(\mathbf{r}, \mathbf{x}_2) = \Sigma_{12} - \mathbf{B}\Sigma_{22} = \mathbf{0}$ である。線形変換 $\mathbf{x} \mapsto (\mathbf{r}, \mathbf{x}_2)$ は正方ブロック行列 $\left(\begin{smallmatrix} \mathbf{I} & -\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{smallmatrix}\right)$ に対応し、その行列式は 1 であるため、可逆な線形変換に関する命題より $(\mathbf{r}, \mathbf{x}_2)$ は同時ガウス分布に従う。共分散の非対角ブロックが消えているため（$\operatorname{Cov}(\mathbf{r}, \mathbf{x}_2) = \mathbf{0}$）、その共分散行列はブロック対角である。結果として共分散の逆行列もブロック対角となり、指数部の二次形式は $\mathbf{r}$ の関数と $\mathbf{x}_2$ の関数の和へと綺麗に分離し、行列式も 2 つのブロックの行列式の積に分解する。したがって同時密度は $\mathbf{r}$ のガウス密度と $\mathbf{x}_2$ のガウス密度の積に因数分解される。ゆえに 2 つのブロックは独立であり、$\mathbf{x}_2$ は平均 μ_2、共分散 Σ_{22} をもつ。$\blacksquare$

証明全体の鍵は、残差が $\mathbf{x}_2$ と無相関になるように $\mathbf{B}$ を選んだ点にある。副産物として、後半の主張からガウスベクトルの任意の周辺部分ベクトルがそれ自体ガウス分布に従うことも証明された（第 1 ブロックについては、可逆な線形変換であるブロックの順序交換を行えばよい）。これにより、先ほど後回しにした長方形行列による線形変換も直ちに解決する。

系（横長行列）。 $\mathbf{A}$ を行フルランク $m \le d$ の $m \times d$ 行列とし、$\mathbf{b} \in \mathbb{R}^m$ とすると、$\mathbf{A}\mathbf{x} + \mathbf{b} \sim \mathcal{N}(\mathbf{A}\mu + \mathbf{b}, \mathbf{A}\Sigma\mathbf{A}^{\top})$ が成り立つ。実際、$\mathbf{A}$ に $d - m$ 個の行を追加して可逆な $d \times d$ 行列を作れば、それによる $\mathbf{x}$ の像はガウス分布に従い、$\mathbf{A}\mathbf{x} + \mathbf{b}$ は単にその像の第 1 ブロックとして得られる。

ここまで来れば、$\mathbf{x}_2$ による条件付けは造作もない。$\mathbf{x}_1 = \mathbf{r} + \mathbf{B}\mathbf{x}_2$ であり $\mathbf{r}$ は $\mathbf{x}_2$ と独立であるため、$\mathbf{x}_2$ を固定することはガウス変数 $\mathbf{r}$ を既知のベクトル $\mathbf{B}\mathbf{x}_2$ だけ平行移動するにすぎない。残差の平均は $\mu_1 - \mathbf{B}\mu_2$、共分散は $\Sigma_{11} - \mathbf{B}\Sigma_{21} - \Sigma_{12}\mathbf{B}^{\top} + \mathbf{B}\Sigma_{22}\mathbf{B}^{\top}$ すなわち $\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$ であるから、条件付き分布 $\mathbf{x}_1 \mid \mathbf{x}_2$ は次のパラメータをもつガウス分布となる。

$$\begin{aligned} \mu_{1|2} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x}_2 - \mu_2), \\ \Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}. \end{aligned} \tag{36.48}$$

Equation 36.48 には際立った特徴が 2 つある。条件付き平均は観測ブロック $\mathbf{x}_2$ のアフィン関数であり、条件付き共分散は観測値に一切依存しない。通し例では $\mathbf{B} = \frac{1}{2}$ なので、$x_2 = 2$ を観測すると条件付き平均は $\frac{1}{2} \cdot 2 = 1$、条件付き分散（conditional variance）は $2 - 1 \cdot \frac{1}{2} \cdot 1 = \frac{3}{2}$ となる。したがって $x_1 \mid x_2 = 2 \sim \mathcal{N}(1, \frac{3}{2})$ となり、$x_2 = 2$ における水平断面（Figure 36.17 右）と一致する。

$\mathbf{B}$ が $\mathbf{x}_2$ から $\mathbf{x}_1$ を予測する線形回帰（linear regression）の行列、すなわちそれを差し引くと相関（correlation）が 0 になるような $\mathbf{x}_2$ の係数行列になっていることに注目しよう。多変量ガウス分布においては、条件付き期待値（conditional expectation）は共分散行列から直接読み取れる線形回帰**そのもの**である。一方、線形代数（linear algebra）で Σ_{22} の**シューア補行列**（Schur complement）と呼ばれる条件付き共分散（33.2.3.5 節）は、差し引かれる項 $\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$ が半正定値（positive semidefinite）であるため、決して Σ_{11} を超えない。x_2 を観測することで、x_1 に関する不確実性は分散 2 から $\frac{3}{2}$ へと縮小する。$\mathbf{x}_2$ が未知関数の観測値を表し、$\mathbf{x}_1$ が新しいテスト入力における関数値を表すとき、この公式はガウス過程回帰（Gaussian process regression）となる（第 40 章（英語版））。

注意深い読者は、共分散が 0 であっても一般には独立性（independence）を意味しないのではないかと疑問に思うかもしれない。そのとおりであり、上の証明はより強い性質、すなわち $(\mathbf{r}, \mathbf{x}_2)$ が**同時**ガウス分布に従うことに依拠していた。同時ガウス性（joint Gaussianity）がなければ、この結論は成り立たない。たとえば $Z \sim \mathcal{N}(0,1)$ とし、S をそれと独立で偏りのないランダムな符号としよう。すると SZ も周辺分布としては $\mathcal{N}(0,1)$ に従い、$\mathrm{Cov}(SZ, Z) = \mathbb{E}[S]\,\mathbb{E}[Z^2] = 0$ を満たすが、$|SZ| = |Z|$ であるため 2 つの変数は強く従属している。

行列の公式を一切使わずに、コレスキー分解で作ったサンプル上でガウス分布の条件付けを直接確かめることもできる。x_2 が幅 0.05 以内で 2 の近くに入るサンプルだけを残し、その x_1 の値の分布を調べるとともに、残差 $r = x_1 - \frac{1}{2}x_2$ を確認してみよう。

```
r = x[:, 0] - 0.5 * x[:, 1]                    # residual x1 - B x2, B = 1/2
print(f'Cov(r, x2) = {np.cov(r, x[:, 1])[0, 1]:.3f},  Var(r) = {r.var(ddof=1):.3f}')
near = x[np.abs(x[:, 1] - 2) < 0.05, 0]        # x1 of samples with x2 near 2
print(f'{near.size} samples with x2 near 2: mean of x1 {near.mean():.2f}, '
      f'variance {near.var(ddof=1):.2f}')
```

```
Cov(r, x2) = -0.002,  Var(r) = 1.502
4135 samples with x2 near 2: mean of x1 1.02, variance 1.53
```

予測どおり、残差の共分散はほぼ 0（x_2 との間）であり、分散はほぼ $\frac{3}{2}$ となる。約 4,000 個のサンプルがこのスライスに入り、それらの x_1 の値は平均が 1 付近、分散が $\frac{3}{2}$ 付近となって $\mathcal{N}(1, \frac{3}{2})$ と一致している。

自然な問いが 2 つ残されている。第 1 に、ここまでに出会ったほぼすべての分布は単一の代数的なひな型を共有しており、36.4 節ではこれを利用して損失、勾配、共役更新を統一的に扱う。第 2 に、冒頭の $715/8192$ という計算はコインが公正であることを仮定していたが、13 回のコイン投げから未知の表の確率（偏り）を推論し、その不確実性を定量化することは 36.5 節の主題となる。

36.3.4 まとめ

以下の表に、本節で導入した分布を登場順にまとめる。カテゴリカル分布と多項分布の行では、交差エントロピー損失（cross-entropy loss）がクラスラベル（class label）を符号化する方式に合わせて、各抽出をワンホットベクトルとして表している。

分布	確率質量関数または確率密度関数	平均	分散	モデルの仮定
Bernoulli(p)	$p^x(1-p)^{1-x}$	p	$p(1-p)$	1 回の yes/no の結果
Binomial(n,p)	$\binom{n}{k}p^k(1-p)^{n-k}$	np	$np(1-p)$	固定回数の n 回の独立試行、共通の p
Poisson(λ)	$\lambda^k e^{-\lambda}/k!$	λ	λ	多数の希少な独立機会、平均と分散が一致
Categorical$(\mathbf{p})$	p_k	$\mathbf{p}$	$\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$	K 通りの結果のうちの 1 つ
Multinomial$(n,\mathbf{p})$	Equation 36.40	$n\mathbf{p}$	$n(\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top)$	固定回数の n 回の独立抽出
Unif$\{1,\dots,n\}$	$1/n$	$(n+1)/2$	$(n^2-1)/12$	等確率の値
Exp(λ)	$\lambda e^{-\lambda t}$、$t \ge 0$	$1/\lambda$	$1/\lambda^2$	一定の事象発生レート（経年劣化なし）
Gamma(α,β)	$\beta^\alpha t^{\alpha-1}e^{-\beta t}/\Gamma(\alpha)$	α/β	α/β^2	α 番目の事象の発生時刻（α が整数の場合）
$U(a,b)$	$1/(b-a)$（$[a,b]$ 上）	$(a+b)/2$	$(b-a)^2/12$	同じ長さの区間がすべて等確率
$\mathcal{N}(\mu,\sigma^2)$	$\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/(2\sigma^2)}$	μ	σ^2	多数の小さな独立した影響の和、有限の分散
Laplace(μ,b)	$\frac{1}{2b}e^{-\lvert x-\mu\rvert/b}$	μ	$2b^2$	対称で指数関数的に減衰する裾
$\mathcal{N}(\mu,\Sigma)$	Equation 36.47	μ	Σ	同時ガウス分布に従う座標群

本章の他の節でも重要な分布がいくつか登場する。コーシー分布は 36.1 節、対数正規分布（log-normal distribution）は 36.2 節、ベータ分布とディリクレ分布は 36.4 節で扱い、さらに負の二項分布（negative binomial distribution）、幾何分布（geometric distribution）、スチューデントの t 分布が演習問題 1、2、4 に登場する。

少数のまとまった分布族によって大半の機械学習システムのラベルとノイズがモデル化され、各分布の負の対数確率が標準的な訓練損失（training loss）を与える。これらはいずれも、和、極限、待ち時間、ランダムな符号、あるいは相関をもつベクトルを通じて、たった 1 回のコイン投げから派生する。そうした構成法のおかげで、各分布の平均、分散、サンプリングアルゴリズムがわずか数行で導かれる。中心極限定理は、有限の分散をもつ多数の小さな独立した影響が足し合わされるときに必ずガウス分布が現れる理由を説明してくれる。最後に、多変量ガウス分布は線形変換と条件付けのもとで閉じており、あるブロックを別のブロックで条件付ける操作は直接線形回帰へと帰着する。

36.3.5 演習問題

1. **ガンマ・ポアソン混合分布**。ポアソン分布のレート自体を確率変数とし、密度 Equation 36.42 をもつ $\lambda \sim \mathrm{Gamma}(\alpha,\beta)$ に従うものとし、$X \mid \lambda \sim \mathrm{Poisson}(\lambda)$ とする。
   1. 全期待値の法則 $\mathbb{E}[X] = \mathbb{E}\big[\mathbb{E}[X \mid \lambda]\big]$ と全分散の法則
   $$\mathrm{Var}(X) = \mathbb{E}\big[\mathrm{Var}(X \mid \lambda)\big] + \mathrm{Var}\big(\mathbb{E}[X \mid \lambda]\big)$$
   を用いて、$\mathbb{E}[X]$ と $\mathrm{Var}(X)$ を求めよ。分散を平均 $m = \alpha/\beta$ と形状 α を用いて表せ。α と β のどのような極限でポアソン分布が再び現れるか。
   2. X が**負の二項分布**の確率質量関数

$$P(X=k)=\frac{\Gamma(k+\alpha)}{k!\,\Gamma(\alpha)}\Big(\frac{\beta}{1+\beta}\Big)^{\alpha}\Big(\frac{1}{1+\beta}\Big)^{k},\qquad k=0,1,2,\dots, \tag{36.49}$$

をもつことを、ガンマ分布の正規化定数を用いて λ を積分消去することで示せ。

3. あるカウントデータのデータセットが標本平均 2、標本分散 6 をもつとする。モデルが両方に一致するように α と β を選べ。このとき、どのような名前付きの分布が得られるか。

2. **無記憶性の逆**。●指数分布は無記憶である。この問題ではその逆を証明する。
   1. $X>0$ を連続確率変数とし、その生存関数 $G(t)=P(X>t)$ がすべての $s,t\ge 0$ に対して $G(s+t)=G(s)\,G(t)$ を満たすとする。ある $\lambda>0$ に対して $G(t)=e^{-\lambda t}$ となることを示せ。まず $G(1)$ から出発し、有理数 t へ式を拡張したうえで、G が非増加であることを用いよ。なぜ $G(1)$ は 0 にも 1 にもなり得ないのか。
   2. X が $\{1,2,\dots\}$ に値をとり、すべての整数 $s,t\ge 0$ に対して $P(X>s+t\mid X>s)=P(X>t)$ を満たすとする。ある $p\in(0,1]$ に対して $P(X>t)=(1-p)^t$ となり、X が幾何分布に従うことを示せ。

3. [extended] **単語出現数のバースト性**。●●Church and Gale (1995) は、テキスト中の単語出現数が文書間でポアソンモデルの許容範囲を超えて変動すること、そして負の二項分布のようなポアソン混合分布のほうがよく当てはまることを発見した。この知見を『The Time Machine』（8.2 節）上で小規模に再現せよ。`d2l.download(d2l.DATA_URL + 'timemachine.txt')` でテキストをダウンロードし、`d2l.TimeMachine._preprocess` と同様に英字のみを残して単語に分割し、その単語系列を文書の役割を果たす長さ $L=500$ 語の連続したチャンクに区切れ。
   1.「the」「and」「of」「time」「traveller」「weena」「morlocks」の各単語について、すべてのチャンクでの出現回数を数え、カウントの標本平均に対する標本分散の比を計算せよ。ポアソンモデルはどのような比を予測するか。そこから最も大きく外れるのはどの単語か。
   2. 各単語のカウントに対してポアソンモデルと負の二項分布モデル（問題 1b）を最尤法によって当てはめ、最大化された対数尤度を比較せよ。負の二項分布は平均 m と形状 α でパラメータ化せよ。最尤推定量 m は標本平均に一致するため、たとえば対数グリッド上の 1 次元探索が必要なのは α のみである。
   3. 各単語について、その単語を含まないチャンクの経験比率および経験的な**適応性** $P(X\ge 2\mid X\ge 1)$ を、当てはめた両モデルの予測値と比較せよ。適応性は何を測っており、なぜポアソンモデルは「traveller」のような名前に対してこれを過小評価するのか。
   4. チャンク長 $L\in\{100,250,500,2000\}$ に対して (a) を繰り返せ。チャンク内の単語のレートが、固定された形状をもつガンマ分布から抽出された 1 語あたりレートの L 倍であると仮定する。問題 1a を用いて分散平均比が L に伴いどのように増大するかを予測し、実測値と比較せよ。

4. **ガンマ尺度混合分布としてのスチューデントの t 分布**。●精度（分散の逆数）自体がランダムであるガウス分布は、いかなるガウス分布よりも重い裾をもつ。その結果として得られるのがスチューデントの t 分布であり、36.5 節では小標本の平均に対してこれを用いる。密度 Equation 36.42 をもつ $\tau\sim\text{Gamma}(\nu/2,\nu/2)$（ある $\nu>0$ に対し形状とレートがともに $\nu/2$）とし、$X\mid\tau\sim\mathcal{N}(\mu,1/\tau)$ とする。
   1. τ を積分消去することで、X が次の密度

$$p(x)=\frac{\Gamma\big(\frac{\nu+1}{2}\big)}{\sqrt{\pi\nu}\,\Gamma\big(\frac{\nu}{2}\big)}\Big(1+\frac{(x-\mu)^2}{\nu}\Big)^{-(\nu+1)/2}, \tag{36.50}$$

   すなわち自由度 ν の**スチューデントの t 分布**をもつことを示せ。**ヒント：** $a,c>0$ に

対して $\int_0^\infty \tau^{a-1} e^{-c\tau}\,d\tau = \Gamma(a)/c^a$ が成り立つ。

2. $p(x)$ が $|x|^{-(\nu+1)}$ のオーダーで減衰し、したがって $\mathbb{E}|X-\mu|^k$ が有限になるのは $k < \nu$ のときに限ることを示せ。36.1 節に登場した分布のうち、$\nu = 1$ の場合に対応するのはどれか。
3. $\mathrm{Var}(X) = \nu/(\nu-2)$（$\nu > 2$）となることを、全分散の法則

$$\mathrm{Var}(X) = \mathbb{E}\big[\mathrm{Var}(X \mid \tau)\big] + \mathrm{Var}\big(\mathbb{E}[X \mid \tau]\big).$$

を用いて示せ。
4. [code] 混合分布の手順（まず τ を引き、次に X を引く）によって $\nu = 3$、$\mu = 0$ のサンプルを 10^5 個生成し、$|X| > 3$ となる割合をガウス分布の 0.0027 と比較せよ。続いて、20 個ずつのサンプルの標本平均を 10^4 個生成し、その標準偏差を $\sqrt{3/20}$ と比較せよ。

5. **1 つの共分散行列に対する 2 つの因子。**●$\Sigma = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ とする。
   1. コレスキー因子 $\mathbf{L}$ を手計算で求め、さらに本文で求めた固有ベクトルと固有値から固有因子 $\mathbf{M} = \mathbf{V}\,\mathrm{diag}(\sqrt{\gamma_1}, \sqrt{\gamma_2})$ を計算せよ。$\mathbf{L}\mathbf{L}^\top = \mathbf{M}\mathbf{M}^\top = \Sigma$ を確かめよ。
   2. $\mathbf{M}_1\mathbf{M}_1^\top = \mathbf{M}_2\mathbf{M}_2^\top$ を満たす任意の 2 つの可逆な因子は、直交行列 $\mathbf{Q}$ を用いて $\mathbf{M}_2 = \mathbf{M}_1\mathbf{Q}$ を満たすことを示せ（**ヒント：**$\mathbf{Q} = \mathbf{M}_1^{-1}\mathbf{M}_2$ とおく）。$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ のとき、$\mathbf{Q}\mathbf{z}$ の分布はどうなるか。ここから $\mu + \mathbf{M}_1\mathbf{z}$ と $\mu + \mathbf{M}_2\mathbf{z}$ が同じ分布をもつことを結論せよ。
   3. **白色化。**$\mathbf{z} = \mathbf{L}^{-1}(\mathbf{x} - \mu)$ が共分散 $\mathbf{I}$ をもつこと（$\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ のとき）を示せ。また、$\mathbf{z}$ を 1 つの $\mathbf{x}$ に対して求める計算が約 d^2 回の演算で済む（$\mathbf{L}$ が既知の場合）のに対し、約 $d^3/3$ 回の演算が $\mathbf{L}$ の計算には必要となる理由を説明せよ。
   4. [code] 各因子を用いて 10,000 点をサンプリングして両方の経験共分散を表示し、三角行列ソルバを用いてコレスキーサンプルを白色化したうえで、結果の共分散を表示せよ。

6. **重ね合わせ、間引き、到着順序。**●レート λ_1 と λ_2 のポアソン過程である 2 つの独立な発生源からサーバーにリクエストが届くとする。
   1. 合流したストリームがレート $\lambda_1 + \lambda_2$ のポアソン過程になることを示せ。区間細分化の描像を用いるか、二項定理によって独立な $\mathrm{Poisson}(\lambda_1)$ と $\mathrm{Poisson}(\lambda_2)$ のカウントの和が $\mathrm{Poisson}(\lambda_1 + \lambda_2)$ に従うことを示せ。
   2. レート λ の過程の各リクエストが確率 q で独立にログに記録されるとする。記録されたリクエストがレート $q\lambda$ のポアソン過程をなすことを、リクエスト数で条件付けて二項分布をポアソン分布にわたって足し合わせることで示せ。毎分 $\lambda = 4$、$q = \frac{1}{4}$ のとき、ある 1 分間に何もログに記録されない確率はいくらか。
   3. ちょうど n 個の事象が $[0, t]$ 内に生じたという条件のもとで、それらの発生時刻は n 個の独立な $U(0, t)$ からのサンプルを昇順に並べたものと同じ分布に従うことを示せ。どのスライスも等確率で事象を含むという細分化の描像を用いよ。
   4. [code] 指数分布の間隔によってレート 4 の過程を 1,000 分間シミュレーションし、$q = \frac{1}{4}$ で間引きを行って、間引き後の毎分カウントの平均と分散が 1 に近いことを確認せよ。ちょうど 3 個の事象が生じた分における各事象の分内の位置を集め、そのヒストグラムが平坦であることを確認せよ。

7. **多項分布の共分散。**●多項分布のカウントベクトルを $\mathbf{x} = \sum_{i=1}^{n} \mathbf{e}_{Y_i}$、すなわち n 個の独立なワンホットベクトルの和（$P(Y_i = k) = p_k$）として表す。
   1. $\mathbb{E}[\mathbf{e}_Y] = \mathbf{p}$ および $\mathrm{Cov}(\mathbf{e}_Y) = \mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$ を示し、そこから $\mathbb{E}[\mathbf{x}] = n\mathbf{p}$ および $\mathrm{Cov}(\mathbf{x}) = n\big(\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top\big)$ を導け。さらに $\mathrm{Var}(x_k) = np_k(1-p_k)$ と $\mathrm{Cov}(x_j, x_k) = -np_jp_k$（$j \neq k$）を読み取れ。すべての共分散が負になるのはなぜか。
   2. $\mathrm{Cov}(\mathbf{x})\mathbf{1} = \mathbf{0}$ を示し、これを制約 $\sum_k x_k = n$ によって説明せよ。

3. 公正なサイコロを 12 回振るとき、Var(x_k)、Cov(x_j, x_k)、および 2 つのカウント間の相関を求めよ。
4. 残りの $K-1$ 個のクラスを 1 つにまとめることで、x_k 単体が Binomial(n, p_k) に従うことを示せ。

36.4 指数分布族

線形回帰、ロジスティック回帰、ソフトマックス回帰は、それぞれ異なる損失関数を最小化する。しかし、それらの損失をモデルの重みに関して微分すると、3 つともまったく同じ訓練規則に帰着する。すなわち、予測と観測の差に入力を掛けたものになる。一見別々にみえる 3 つの導出が 1 つの式に収束するとき、その背後には共通の構造が潜んでいるのが常である。36.3 節の分布カタログでも、各分布の平均や分散を求めるたびに独自の巧妙な代数的工夫が必要にみえたが、やはり同じ疑問に行き着く。

本章を通して登場する、13 回のコイントスで 9 回表が出たコインを考えよう。このコインのバイアスに対する自然な推定値は $9/13 \approx 0.69$ である。同じ 13 回のコイントスにバイアス項のみを持つロジスティック回帰モデルを当てはめると、モデルは間接的な経路をたどって $9/13$ に到達する。最終層が対数オッズ（log-odds）$\log(9/4) \approx 0.81$ を計算し、シグモイドがその数値を $9/13$ へと写し戻すのである。確率を直接扱うのではなく、なぜわざわざ対数オッズを経由する回り道をとるのだろうか。

1 つの考え方が、この両方の謎を解き明かす。カタログに載っているほぼすべての分布は、少数のデータ統計量の線形結合の指数関数を通してのみパラメータが現れる、単一の正準形（canonical form）で書き表せる。この構造を持つ分布は**指数分布族**（exponential family）をなし、コインの場合、指数部に現れるパラメータこそが対数オッズにほかならない。驚くべきことに、この形式だけから 4 つの中心的な性質が導かれる。第 1 に、平均と分散は単一の関数の 1 階および 2 階の導関数にすぎない。第 2 に、最尤法は勾配が常に「予測マイナス観測」となる凸最適化問題になる。第 3 に、クラスラベルをモデル化するとソフトマックスが自動的に現れる。第 4 に、同じ形式の事前分布はカウントを足し合わせるだけで更新される。

深層学習においては 2 番目の性質が中心的な結果であり、1 番目の性質がその証明を与える（36.4.3.2 節および 36.4.3.3 節）。この 2 つを合わせれば、カウントのような新しい種類のラベルにどの損失関数を組み合わせるべきか、またコインがすべての試行で表になったときのように最尤法による当てはめが有限の解を持たないのはいつか、といったことがわかる。なお、本節のうち 2 つの箇所は初読時には読み飛ばしてもよい。指数形式がどこから生じるかを説明する最大エントロピーによる導出（36.4.2 節）と、ノイズを伴う選択および統計物理学からのソフトマックスの別導出である。

36.4.1 多様な分布を統一する単一の形式

まずはカタログの中で最も単純な分布である 1 回のコイントスから始め、対数オッズが自然に姿を現すまで式を書き換えてみよう。このわずかな式変形から、一般形がどのような姿になるべきかが見えてくる。

36.4.1.1 指数形式によるベルヌーイ分布

バイアス p のコインでは、表（$x=1$）が確率 p で、裏（$x=0$）が確率 $1-p$ で出る。前節ではこの両方のケースを 1 つの式 $p^x(1-p)^{1-x}$ にまとめた。結果 x とパラメータ p がどのように関わるかを見るため、対数をとって x を含む項をくくり出そう。

$$\log P(X=x) = x\log p + (1-x)\log(1-p) = x\log\frac{p}{1-p} + \log(1-p).$$

2 つの役割がいかにきれいに分離しているかに注目してほしい。結果は x と対数オッズ $\log\frac{p}{1-p}$ の積を通してのみ現れ、第 2 項は p のみに依存する。ここで対数オッズに $\eta = \log\frac{p}{1-p}$ という名前を付け、第 2 項を η で書き直そう。p について解くと $p = e^\eta/(1+e^\eta)$ となるため、$1-p = 1/(1+e^\eta)$ であり、$\log(1-p) = -\log(1+e^\eta)$ となる。両辺の指数をとると次式が得られる。

$$P(X=x) = \exp\big(x\eta - \log(1+e^\eta)\big), \qquad x \in \{0,1\}.$$

これでコインの確率は、x の線形関数から x にまったく依存しない項を引いたものの指数関数となった。この第 2 項は単なる正規化定数に対応する。実際、正規化前の 2 つの重みは $e^0 = 1$ と e^η であり、その和は $1+e^\eta$ である。

この式に現れた 3 つの構成要素は、本節を通して繰り返し登場する。**統計量**（statistic）$T(x) = x$ は、結果のうち本質的に効いてくる部分である。そこに掛かるのが**自然パラメータ**（natural parameter）η であり、ベルヌーイ分布の場合は対数オッズ、すなわち**ロジット**とも呼ばれる量にあたる。最後に、**対数分配関数**（log-partition function）$A(\eta) = \log(1+e^\eta)$ は正規化定数の対数であり、**ソフトプラス**として見覚えのある関数だろう。先ほどの変数変換を逆にたどると、次のようになる。

$$p = \frac{e^\eta}{1+e^\eta} = \operatorname{sigmoid}(\eta), \qquad \operatorname{sigmoid}(\eta) = \frac{1}{1+e^{-\eta}}.$$

したがって、ロジスティック回帰のシグモイドとは、コインの自然パラメータをその平均へと写し戻す写像にほかならない。誰かが恣意的に発明する必要はなかったのである。

本章のコインの例では、経験的なバイアス $9/13$ に対応する自然パラメータは次のようになる。

$$\hat{\eta} = \log\frac{9/13}{4/13} = \log\frac{9}{4} \approx 0.811,$$

そしてシグモイドはこれをそのまま元の値へと戻す。すなわち $\operatorname{sigmoid}(\log\frac{9}{4}) = \frac{9/4}{1+9/4} = \frac{9}{13}$ である。なぜ $\hat{\eta}$ が最尤法による対数オッズの値になるのか、さらに $\hat{\eta}$ の周りでの損失の曲率がその推定値の不確実性について何を教えてくれるのかは、まもなく確認する。

36.4.1.2 定義と属する分布

ベルヌーイ分布の書き換えには、データの統計量、それに掛かるパラメータ、そして正規化定数という 3 つの要素があった。これを複数の統計量へと同時に一般化し、さらに各結果に対してパラメータに依存しない固定の重みを許容すると、カタログの大半を収める単一のテンプレートが得られる。

ある分布の族は、その確率質量関数または確率密度関数が次の形で書けるとき、**指数分布族**であるという。

$$p(\mathbf{x} \mid \eta) = h(\mathbf{x})\,\exp\big(\eta^\top T(\mathbf{x}) - A(\eta)\big). \tag{36.51}$$

この形式では、データとパラメータは内積 $\eta^\top T(\mathbf{x})$ を通してのみ相互作用する。**基底測度**（base measure）$h(\mathbf{x}) \geq 0$ はパラメータとは独立に各結果へ固定の重みを割り当て、$h > 0$ となる領域が分布の台（support）を定める。**自然パラメータ** $\eta \in \mathbb{R}^d$ は調整可能なパラメータであり、**十分統計量**（sufficient statistic）$T(\mathbf{x}) \in \mathbb{R}^d$ は自然パラメータが掛かるデータの特徴量である。**対数分配関数**は正規化定数（normalizing constant）の対数であり、次式で定義される。

$$A(\eta) = \log \int h(\mathbf{x})\, \exp(\eta^\top T(\mathbf{x}))\, d\mathbf{x}, \tag{36.52}$$

連続密度の場合は積分をとり、離散的な $\mathbf{x}$ の場合は和をとる。平たく言えば、A とは Equation 36.51 の積分または和を 1 にするための定数であり、有効な自然パラメータとは $A(\eta)$ が有限になるようなパラメータのことである。ベルヌーイ分布は $h(x) = 1$、$T(x) = x$、$A(\eta) = \log(1 + e^\eta)$ とした特別な場合にあたる。

なぜ特徴量 $T(\mathbf{x})$ は**十分**と呼ばれるのだろうか。それは、η についてデータから知り得る情報のすべてを保持しているからである。独立な n 個の観測に対する同時尤度は、次のように因数分解される。

$$\prod_{i=1}^{n} p(\mathbf{x}_i \mid \eta) = \Big(\prod_{i=1}^{n} h(\mathbf{x}_i)\Big) \exp\Big(\eta^\top \sum_{i=1}^{n} T(\mathbf{x}_i) - n A(\eta)\Big).$$

パラメータは和 $\sum_i T(\mathbf{x}_i)$ を通してのみデータと関わる。この和が等しい 2 つのデータセットは、η に依存しない定数倍だけが異なる尤度を生じるため、まったく同じパラメータ推定値を与える。コインの場合、この和は単なる表の合計回数である。13 回のコイントスのどの並べ替えもあらゆるバイアスのもとで同じ確率を持つため、9 回表が出たことさえわかれば、その出現順序はコインについてそれ以上の情報を何も付け加えない。

各項を Equation 36.51 と照らし合わせれば、この族に属する他の分布もすぐに特定できる。レート λ のポアソン分布では、事象が k 回起きる確率は $\frac{\lambda^k e^{-\lambda}}{k!} = \frac{1}{k!}\exp(k\log\lambda - \lambda)$ であるから、$h(k) = 1/k!$、$T(k) = k$、$\eta = \log\lambda$、$A(\eta) = e^\eta = \lambda$ と読み取れる。平均と分散がともに未知のガウス分布には 2 つの統計量が必要になる。指数部の二乗を展開して x と x^2 の項に整理すると、次のようになる。

$$\frac{1}{\sqrt{2\pi}} \exp\Big(\underbrace{\tfrac{\mu}{\sigma^2}}_{\eta_1}\, x + \underbrace{\big(-\tfrac{1}{2\sigma^2}\big)}_{\eta_2}\, x^2 - \underbrace{\big(\tfrac{\mu^2}{2\sigma^2} + \log\sigma\big)}_{A(\eta)}\Big),$$

したがって、$T(x) = (x, x^2)^\top$ および $h(x) = 1/\sqrt{2\pi}$ となる。第 2 の自然パラメータが厳密に負でなければならない点に注目しよう。$\eta_2 < 0$ のとき、因子 $e^{\eta_2 x^2}$ がガウス分布の急速な裾の減衰をもたらすのに対し、$\eta_2 \geq 0$ では A の正規化積分が発散する。Table 36.1 に、これらを含む代表的な指数分布族の分布をまとめる。

Table 36.1: 指数分布族に属する分布。各行の確率または密度は $h(\mathbf{x})\exp(\eta^\top T(\mathbf{x}) - A(\eta))$ であり、二項分布では試行回数 n、ラプラス分布では位置 μ を固定する必要がある。

分布	$h(\mathbf{x})$	$T(\mathbf{x})$	η	$A(\eta)$
ベルヌーイ分布（p）	1	x	$\log\frac{p}{1-p}$	$\log(1+e^\eta)$
二項分布（n, p）	$\binom{n}{x}$	x	$\log\frac{p}{1-p}$	$n\log(1+e^\eta)$

分布	$h(\mathbf{x})$	$T(\mathbf{x})$	η	$A(\eta)$
ポアソン分布（λ）	$1/k!$	k	$\log\lambda$	e^{η}
カテゴリカル分布（one-hot $\mathbf{y}$）	1	$\mathbf{y}$	ロジット $\mathbf{z}$	$\log\sum_k e^{z_k}$
指数分布（λ）	1（$x\geq 0$ のとき）	x	$-\lambda$	$-\log(-\eta)$
$\mathcal{N}(\mu,\sigma^2)$	$1/\sqrt{2\pi}$	(x,x^2)	$\left(\frac{\mu}{\sigma^2},-\frac{1}{2\sigma^2}\right)$	$-\frac{\eta_1^2}{4\eta_2}-\frac{1}{2}\log(-2\eta_2)$
ラプラス分布 (μ,b)	1	$\lvert x-\mu\rvert$	$-1/b$	$\log(-2/\eta)$

カテゴリカル分布の行はロジットを用いてクラス確率を表しており、ここからソフトマックスが生じる。その起源については 36.4.4 節で詳しく探る。さらに高次元にはもう 2 つの重要な分布がある。十分統計量 $\mathbf{x}$ と $\mathbf{x}\mathbf{x}^{\top}$、自然パラメータ $\Sigma^{-1}\mu$ と $-\frac{1}{2}\Sigma^{-1}$ を持つ多変量ガウス分布、そして試行回数を固定した多項分布である。なお、h、η、T への分解は一意ではないことに留意しておこう。のちに重要となる唯一の冗長性はカテゴリカル分布の行にある。one-hot ベクトルの K 個の成分の和は常に 1 になるため、K 個のロジットのうち 1 つは冗長なのである（36.4.3.2 節）。

36.4.1.3 指数分布族に属さない分布

すべての分布が Equation 36.51 に収まるわけではなく、2 つの古典的な例外はそれぞれ示唆に富む異なる理由から指数分布族からはみ出す。

まず、一様分布 $U(0,\theta)$ を考えよう。その密度は $1/\theta$（区間 $[0,\theta]$ 上）であり、その外側では 0 である。ここでは、台そのものがパラメータ θ とともに移動する。対照的に、指数分布族では指数因子が決して 0 にならないため、台は $h>0$ となる集合に固定されている。台が移動すると、推定の性質は根本から変わる。たとえば 3 個のサンプル 0.1、0.2、0.9 を観測したとする。尤度は θ^{-3}（$\theta\geq 0.9$ のとき）および 0（$\theta<0.9$ のとき）となる（それより小さな θ からは 0.9 が決して生成されないからである）。したがって、最尤推定量は標本最大値 $\hat{\theta}=\max_i x_i=0.9$ になる。

まもなく証明するように、指数分布族の内部では、最尤法は常に十分統計量のモデル平均をその標本平均と等しくおく。その処方箋をここにそのまま当てはめると、平均 $\theta/2$ を $\bar{x}=0.4$ と等しくおくことになり、$\theta=0.8$ が得られるが、この値のもとでは実際に観測されたサンプル 0.9 が起こり得ないことになってしまう。一様分布の尤度はデータの和ではなく最大値を通してデータに依存するため、本節の仕組みは適用できないのである。演習問題 7（36.6 節）では、標本サイズが増えるにつれてこれら 2 つの推定量がどう振る舞うかを比較する。

裾の重い分布は、第 2 の理由から指数分布族の外に出る。**スチューデントの** t 分布は、精度 $1/\sigma^2$ 自体がランダムなガウス分布をとることで生じる。その精度について平均をとると、急速に減衰する e^{-x^2} の裾が多項式減衰の裾に置き換わる（密度の導出は演習問題 4 を参照）。スケール 1、自由度 ν の場合、n 個の観測の尤度は因子 $\left(1+(x_i-\mu)^2/\nu\right)^{-(\nu+1)/2}$ の積となり、データの有限個の和へと圧縮することはできない。実際、Pitman、Koopman、Darmois による古典的な定理によれば、台が固定された正則な分布族のうち、固定次元の十分統計量を持つのは指数分布族に限られる (Darmois 1935; Koopman 1936; Pitman 1936)。

実用上の帰結として、すべての指数分布族で成り立つことをのちに証明する負の対数尤度の凸性は、その外側では崩れ得る。スチューデントの t 分布では、データがよく離れた 2 つのクラスタに分かれているとき、位置パラメータ μ に関する負の対数尤度は各クラスタの近くに別々の極小値を持つようになる。

36.4.2 最大エントロピー

ここまでは、なじみのある分布を書き換えるだけで指数形式が現れた。しかしさらに踏み込んで、そもそもこの形式はどこから生じるのかと問うてみよう。ある確率的な量についてわかっているのが、少数の統計量の平均値だけだとする。そのとき、どの分布を選ぶべきだろうか。驚くべきことに、それらの平均値以外の仮定を一切置かない分布は、常に指数分布族に属する。

36.4.2.1 エントロピーと偏りのあるサイコロ

「平均値以外の仮定を一切置かない」ということを厳密にするには、分布がどれほど広がっているか、言い換えれば特定の値に偏らず余計な決めつけをしていないかを測る尺度が必要である。有限集合上の分布 $p = (p_1, \dots, p_K)$ に対して、**エントロピー**は次式で定義される。

$$H[p] = -\sum_{k=1}^{K} p_k \log p_k,$$

単位は nat とし、慣例として $0 \log 0 = 0$ と定める。各項は驚き $-\log p_k$（結果 k に対する値）をその確率で重み付けしているため、エントロピーとは平均的な驚きにほかならない。公正な 6 面サイコロではどの面にも $\log 6$ の驚きが割り当てられ、エントロピーは $\log 6 \approx 1.79$ nat となる。常に 6 の目しか出ないサイコロには驚きがまったくなく、エントロピーは 0 である。K 個の結果上のあらゆる分布の中で、一様分布が最大可能なエントロピー $\log K$ を達成する。この事実はすぐあとで証明しよう。

E. T. Jaynes はこの問題を具体的なパズルとして定式化した (Jaynes 1957)。ある 6 面サイコロの出目の平均が 4.5（公正な場合の 3.5 ではなく）であるとだけ知らされ、それ以外は何もわからないとする。$\{1, \dots, 6\}$ 上で平均 4.5 を持つ分布は数多く存在する。確率 $1/2$ を 3 に、確率 $1/2$ を 6 に割り当てる分布もその 1 つだが、この分布は 1、2、4、5 が決して出ないと大胆に断定しており、知らされた情報にはそのような根拠はどこにもない。この分布のエントロピーはわずか $\log 2 \approx 0.69$ である。

Jaynes の**最大エントロピー原理**（principle of maximum entropy）は、既知の平均と一致するすべての分布の中からエントロピーが最大の分布、すなわち手持ちの知識と整合する範囲で最も余計な仮定を置かない選択を採用する。証明に先立って答えを述べよう。一意な最大点は、一様分布の**指数傾斜**（exponential tilt）で与えられる。

$$p_k = \frac{e^{\eta k}}{\sum_{j=1}^{6} e^{\eta j}}, \qquad k = 1, \dots, 6,$$

ここで η は平均が 4.5 になるよう調整する。3 と 6 の目に集中した分布は、この平均を達成するために公正なサイコロの $\log 6 - \log 2 \approx 1.10$ nat（1.79 nat のうち）ものエントロピーを手放していた。下のコードを実行する前に、指数傾斜がどれだけのエントロピーを手放すことになるか少し予想してみてほしい。指数傾斜の平均は η に関して単調に増加するため、単純な二分法で正確な η の値が求まる。

```
k = np.arange(1, 7)
def tilt(eta):                                  # p_k proportional to exp(eta k)
    w = np.exp(eta * k)
    return w / w.sum()
```

```
lo, hi = -5.0, 5.0
for _ in range(50):                                  # bisection: the mean increases with eta
    eta = (lo + hi) / 2
    lo, hi = (eta, hi) if (tilt(eta) * k).sum() < 4.5 else (lo, eta)
p = tilt(eta)
print('eta =', round(eta, 3), ' p =', p.round(3))
print('H =', round(-(p * np.log(p)).sum(), 3), ' log 6 =', round(np.log(6), 3))
```

```
eta = 0.371  p = [0.054 0.079 0.114 0.165 0.24  0.347]
H = 1.614  log 6 = 1.792
```

得られる分布は各面にわたって等比的に変化し、どの面も 1 つ下の面より $e^{0.371} \approx 1.45$ 倍出やすくなる。そのエントロピーは 1.61 nat であり、公正なサイコロと比べてわずか 0.18 nat しか失われていない。また $\eta = 0$ とおけば公正なサイコロに戻る。Figure 36.18 では、3 通りの目標平均に対する指数傾斜を比較している。

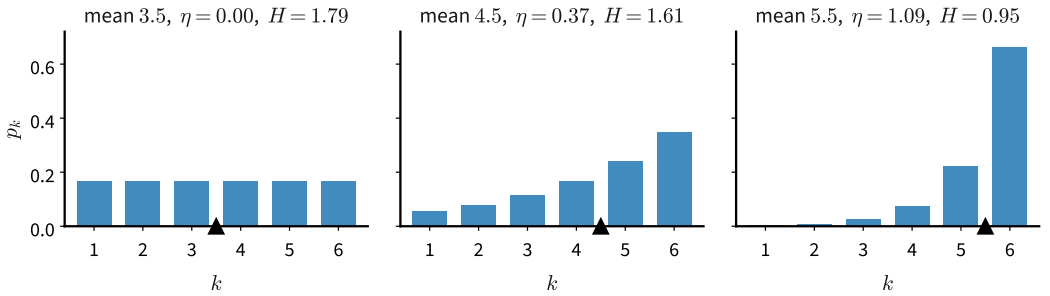


Figure 36.18: 出目の平均を 3.5、4.5、5.5（黒三角）に固定したサイコロの最大エントロピー分布。いずれも一様分布の指数傾斜 $p_k \propto e^{\eta k}$ であり、要求される平均が高くなるにつれて傾斜パラメータ η は大きくなり、エントロピー H は低下する。

この指数傾斜がすでに指数分布族の形式 Equation 36.51（6 つの面上で $h \equiv 1$、統計量 $T(k) = k$、自然パラメータ η）になっていることに注目しよう。統計量の平均を固定してエントロピーを最大化すると、その統計量の定数倍の指数関数へとまっすぐ導かれた。これはサイコロに限った偶然ではない。

36.4.2.2 最大エントロピーの命題

一般の定理を述べるには、結果の空間が自然な基準重みを備えている場合を扱えるよう、エントロピーを少しだけ拡張しておく必要がある。分布 q と基底測度 h に対し、**基準測度に対する相対エントロピー**（以下、h に対する相対エントロピー）を次式で定義する。

$$H_h[q] = -\sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{h(\mathbf{x})}$$

これは離散的な結果に対する式であり、連続密度の場合は和を積分に置き換える。$h \equiv 1$ とおけば、上で定義した通常のエントロピーに戻る。

証明では、任意の候補分布 q を指数分布族の分布 p と直接比較する。同じ空間上の 2 つの分布 p と q について、$p(\mathbf{x}) > 0$ が $q(\mathbf{x}) > 0$ となるすべての点で成り立つとする。このとき、q の p に対する**カルバック・ライブラー・ダイバージェンス**は、$D_{\mathrm{KL}}(q\|p) = \sum q \log(q/p)$（$q > 0$ とな

る結果にわたる和、密度の場合は対応する積分）で定義される (Cover and Thomas 1999)。エントロピーとこのダイバージェンスについては 37.1 節で詳しく学ぶが、ここで必要なのは 1 本の接線だけで証明できる初等的な事実 1 つだけである。対数関数は凹関数であるため、$\log u \le u - 1$ がすべての $u > 0$ に対して成り立ち、等号は $u = 1$ のときに限られる（Figure 36.19）。

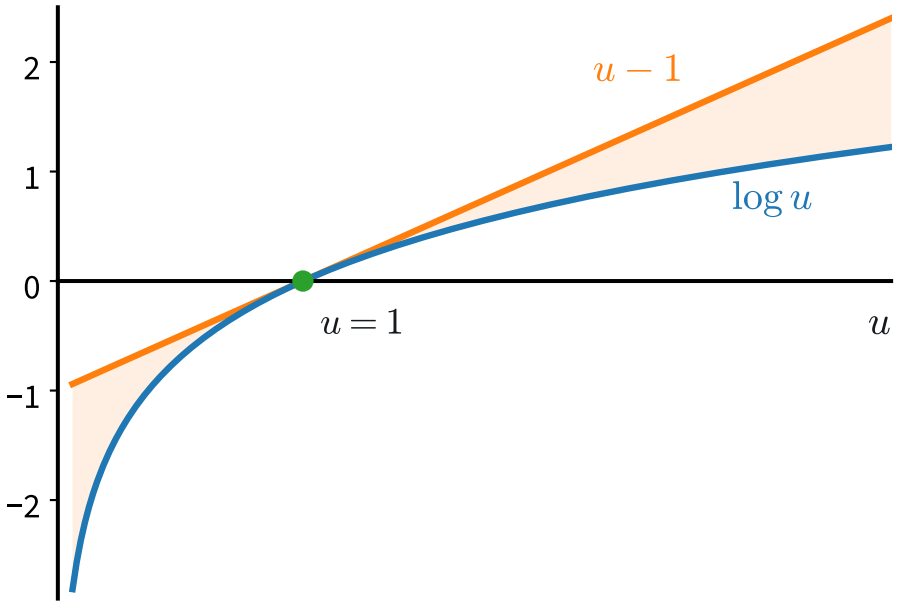


Figure 36.19: 接線による上界 $\log u \le u - 1$。凹関数である対数（青）は、$u = 1$ での接線（オレンジ）よりもすべての $u > 0$ において下側に位置し、その 1 点でのみ接する。

補題（ギブスの不等式）：カルバック・ライブラー・ダイバージェンスは決して負にならない。すなわち、

$$D_{\mathrm{KL}}(q \| p) = \sum_{\mathbf{x}:\, q(\mathbf{x})>0} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \ge 0, \tag{36.53}$$

であり、等号が成り立つのは $q = p$ のとき、かつそのときに限る。

証明：接線による上界 $\log u \le u - 1$ を $u = p(\mathbf{x})/q(\mathbf{x})$ に適用し、$q(\mathbf{x}) > 0$ となるすべての結果について和をとる。

$$-D_{\mathrm{KL}}(q \| p) = \sum_{q>0} q \log \frac{p}{q} \le \sum_{q>0} q\Big(\frac{p}{q} - 1\Big) = \sum_{q>0} p - 1 \le 0.$$

最初の不等式で等号が成り立つには、$p = q$ が $q > 0$ となるすべての点で成り立つ必要がある。すると $\sum_{q>0} p = 1$ が強制され、p には q の台の外側に残る確率質量がなくなるため、いたるところで $p = q$ となる。■

命題（最大エントロピー）：$p(\mathbf{x}) = h(\mathbf{x}) \exp(\eta^\top T(\mathbf{x}) - A(\eta))$ を指数分布族に属する分布とし、q を（連続の場合は密度を持ち）h が 0 になるすべての点で 0 となり、有限の平均統計量 $\mathbb{E}_q[T(\mathbf{x})]$ を持ち、かつその平均において p と一致する（すなわち $\mathbb{E}_q[T(\mathbf{x})] = \mathbb{E}_p[T(\mathbf{x})]$ を満たす）任意の

分布とする。このとき $H_h[q] \le H_h[p]$ が成り立ち、等号は $q = p$ のとき、かつそのときに限り成り立つ。

証明： $H_h[q]$ の中の対数に p を挟んで $\log \frac{q}{h} = \log \frac{q}{p} + \log \frac{p}{h}$ と分解する。第 1 項からは $-D_{\mathrm{KL}}(q\|p) \le 0$ がギブスの不等式より得られる（$p > 0$ が $h > 0$ のすべての点で成り立つため適用できる）。第 2 項は十分統計量に関して線形であり、$\log \frac{p}{h} = \eta^{\top} T - A(\eta)$ となるため、q は T の平均を通してのみ現れる。

$$-\sum_{\mathbf{x}} q \log \frac{p}{h} = -\eta^{\top} \mathbb{E}_q[T] + A(\eta) = -\eta^{\top} \mathbb{E}_p[T] + A(\eta) = H_h[p].$$

中央の等号では平均の制約を用い、最後の等号では p を q の代わりに置いて同じ計算を繰り返した。2 つの項を合わせると $H_h[q] = H_h[p] - D_{\mathrm{KL}}(q\|p) \le H_h[p]$ が得られ、等号は $q = p$ の場合に限り成り立つ。■

鍵となるステップは 2 番目にある。$\log(p/h)$ が T に関して線形であるため、交差項は q に $\mathbb{E}_q[T]$ のみを通して依存し、その値は制約によって固定されている。平たく言えば、**指数分布族のある分布が平均の制約を満たすなら、その分布は同じ制約を満たすすべての分布の中でエントロピーを最大にする一意な分布である。**注目すべきことに、ラグランジュ乗数は一切不要であり、この証明は単なる停留点ではなく大域的最大値を直接保証している。

偏りのあるサイコロでは、$\eta \approx 0.371$ とした指数傾斜が $h \equiv 1$、$T(k) = k$ の指数分布族に属し、かつ平均 4.5 を持つため、これが一意な最大点となる。さらに、なじみ深い 3 つの分布も系として直ちに従う。

- **制約なし**（$T \equiv 0$）：すべての q が許容され、最大点は $p \propto h$ となる（ただし h の和が有限であるとする）。$h \equiv 1$ とした有限集合上では、これはエントロピー $\log K$ の一様分布であり、公正なサイコロは $K = 6$ の場合にあたる。
- $\mathbb{R}$ **上での平均と分散**：$T(x) = (x, x^2)$ かつ定数の h をとると、最大点はガウス分布になる。定数の h は H_h を一定値だけ平行移動させるにすぎないため、所定の平均と分散を持つすべての密度関数の中でガウス分布が最大のエントロピーを持つ。
- **カウント上の平均**（$\{0, 1, 2, \dots\}$ 上で $T(k) = k$）：$h(k) = 1/k!$ とれば、H_h を平均 m のもとで最大化する最大点はレート $\lambda = m$ のポアソン分布になる一方、$h \equiv 1$ とれば幾何分布 $p_k = (1 - r) r^k$（ただし $r = m/(1 + m)$）が得られる。演習問題 2 で詳しく調べるように、どの分布が現れるかは基底測度によって決まる。

指数分布族のいずれかの分布が制約を満たすという仮説を落とさないよう注意してほしい。たとえば平均 m のみを $\mathbb{R}$ 全体の上で固定し、$h \equiv 1$ とした場合を考えよう。候補となる密度 $e^{\eta x}$ は $\mathbb{R}$ 上でいかなる η に対しても可積分ではないため、該当する分布は存在しない。それどころか最大点自体が存在しない。実際、ガウス分布 $\mathcal{N}(m, s^2)$ は平均 m とエントロピー $\frac{1}{2} \log(2\pi e s^2)$ を持ち、これは $s \to \infty$ とすると際限なく大きくなる。

36.4.3 対数分配関数

36.3 節の分布カタログでは、平均と分散を求めるたびに別々の工夫が必要にみえた。ベルヌーイ分布では $X^2 = X$、二項分布では独立なコイントスの和、ポアソン分布では級数の添字ずらし、ガウス分布では部分積分といった具合である。しかし指数分布族の内部では、それらの計算はすべて対数分配関数 A を微分するという単一の操作に帰着する。1 階導関数が平均を、二階導関数が分散を与え、しかも同じ関数がそのまま最尤法を支配する。

36.4.3.1 モーメントの性質

$A(\eta)$ は指数関数 $\exp(\eta^\top T(\mathbf{x}))$ の積分または和の対数であった。このような対数を微分すると、因子 $T(\mathbf{x})$ が $\mathbf{x}$ の確率で重み付けされた形で降りてくる。それはまさに T の期待値にほかならない。

命題（十分統計量の平均）：指数分布族の分布 Equation 36.51 に対して、次式が成り立つ。

$$\nabla_\eta A(\eta) = \mathbb{E}[T(\mathbf{x})]. \tag{36.54}$$

証明：@eq-mdl-log-partition を積分記号の下で微分する。この微分と積分の交換は、$A(\eta) < \infty$ となる自然パラメータの集合の内部で正当化される (Folland 1999)。この集合はヘルダーの不等式により凸であり、以降は η をその内部にとる。$Z(\eta) = e^{A(\eta)} = \int h(\mathbf{x}) e^{\eta^\top T(\mathbf{x})} d\mathbf{x}$ と書けば、連鎖律より $\nabla A = (\nabla Z)/Z$ となり、ここで分子は次のようになる。

$$\nabla_\eta Z = \int h(\mathbf{x})\, T(\mathbf{x})\, e^{\eta^\top T(\mathbf{x})}\, d\mathbf{x}.$$

これを $Z = e^A$ で割ると因子 e^{-A} が積分の中に入り、密度 Equation 36.51 が完成する。

$$\begin{aligned} \nabla_\eta A &= \frac{1}{Z} \int h(\mathbf{x})\, T(\mathbf{x})\, e^{\eta^\top T(\mathbf{x})}\, d\mathbf{x} \\ &= \int T(\mathbf{x}) \underbrace{h(\mathbf{x})\, e^{\eta^\top T(\mathbf{x}) - A(\eta)}}_{p(\mathbf{x}\mid\eta)}\, d\mathbf{x} = \mathbb{E}[T(\mathbf{x})]. \quad \blacksquare \end{aligned}$$

同様の積分と微分の交換のもとで $\nabla A = (\nabla Z)/Z$ をもう 1 回微分すると、十分統計量の共分散行列が得られる。

$$\nabla^2 A = \frac{\nabla^2 Z}{Z} - \frac{(\nabla Z)(\nabla Z)^\top}{Z^2} = \mathbb{E}[T\,T^\top] - \mathbb{E}[T]\,\mathbb{E}[T]^\top = \mathrm{Cov}(T).$$

要するに、A の傾きは T の平均であり、その曲率は T の共分散である。あらゆる共分散行列は半正定値であるため（$\mathbf{v}^\top \mathrm{Cov}(T)\mathbf{v} = \mathrm{Var}(\mathbf{v}^\top T) \geq 0$ がすべての $\mathbf{v}$ に対して成り立つ、33.3.2.2 節）、対数分配関数 A は常に凸関数となる。

なじみのある 2 つの例で確認してみよう。コインの場合、$A(\eta) = \log(1 + e^\eta)$ からベルヌーイ分布の平均 $A'(\eta) = \mathrm{sigmoid}(\eta) = p$ と、その分散 $A''(\eta) = \mathrm{sigmoid}(\eta)(1 - \mathrm{sigmoid}(\eta)) = p(1-p)$ が得られる。$\hat{\eta} = \log \frac{9}{4}$ において、これらはそれぞれ $9/13 \approx 0.692$ および $\frac{9}{13} \cdot \frac{4}{13} = \frac{36}{169} \approx 0.213$ となる。ポアソン分布では $A(\eta) = e^\eta$ は微分しても自分自身のままであるから、$\lambda = 4$（$\eta = \log 4$）において A' と A'' はともに 4 になる。モーメントは単なる導関数であるため、コードで書ける任意の A に対して自動微分で直接計算できる。両方のケースを確かめてみよう。

```
def derivatives(A, eta):
    """First and second derivative of the scalar function A at eta."""
    return float(jax.grad(A)(eta)), float(jax.grad(jax.grad(A))(eta))

cases = [('Bernoulli', jax.nn.softplus, math.log(9 / 4), 9 / 13, 36 / 169),
```

```
          ('Poisson', jnp.exp, math.log(4), 4, 4)]
for name, A, eta, mean, var in cases:
    dA, d2A = derivatives(A, eta)
    print(f"{name:9s} A' = {dA:.4f} (mean {mean:.4f})  A'' = {d2A:.4f} (variance {var:.4f})")
```

```
Bernoulli A' = 0.6923 (mean 0.6923)  A'' = 0.2130 (variance 0.2130)
Poisson   A' = 4.0000 (mean 4.0000)  A'' = 4.0000 (variance 4.0000)
```

どちらの導関数も、表示された 4 桁すべてにおいて解析的な平均および分散と一致している。分布カタログではべき級数の計算を要した「ポアソン分布の平均と分散が等しい」という有名な性質が、ここでは $A = A' = A''$（$A(\eta) = e^{\eta}$ に対して）という 1 行に凝縮されていることに注目してほしい。

36.4.3.2 モーメントマッチングとしての最尤法

モーメントの性質により、最尤法は期待値に関する驚くほど明快な条件へと姿を変える。すなわち、当てはめたモデルは十分統計量の標本平均を再現しなければならない。独立な観測 $\mathbf{x}_1, \dots, \mathbf{x}_n$ が与えられたとき、Equation 36.51 の積からなる尤度の負の対数をとると次式が得られる。

$$-\sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \eta) = n\,A(\eta) - \eta^{\top} \sum_{i=1}^{n} T(\mathbf{x}_i) - \sum_{i=1}^{n} \log h(\mathbf{x}_i). \tag{36.55}$$

ここでデータは和 $\sum_{i=1}^{n} T(\mathbf{x}_i)$ を通してのみ現れる。A は凸関数であり第 2 項は η に関して線形であるため、負の対数尤度は η に関して凸関数となる。その勾配を 0 とおき、モーメントの性質 Equation 36.54 を適用すると次式が得られる。

$$\nabla A(\hat{\eta}) = \bar{T}, \qquad \bar{T} = \frac{1}{n}\sum_{i=1}^{n} T(\mathbf{x}_i).$$

したがって最尤法は、モデルのもとでの十分統計量の期待値を観測された標本平均に一致させる。この原理を**モーメントマッチング**と呼ぶ。しかも目的関数が凸であるため、この方程式の解は自動的に大域的な最小点となる。

13 回のコイントスで 9 回表が出たコインで、これが何を与えるか見てみよう。負の対数尤度は $13A(\eta) - 9\eta = 13\log(1 + e^{\eta}) - 9\eta$ である。導関数 $13\,\mathrm{sigmoid}(\eta) - 9$ が 0 になるのはまさに $\mathrm{sigmoid}(\hat{\eta}) = 9/13$ のときであり、ここから $\hat{\eta} = \log\frac{9}{4} \approx 0.811$ が得られる。この最小点において、損失の値は次のようになる。

$$13\log\frac{13}{4} - 9\log\frac{9}{4} \approx 8.02 \text{ nats},$$

これはバイアス $9/13$ のコインのエントロピー $-\frac{9}{13}\log\frac{9}{13} - \frac{4}{13}\log\frac{4}{13} \approx 0.617$ の 13 倍である。最適点において、コイントス 1 回あたりの平均損失は当てはめられたコインのエントロピーに一致する。

この恒等式は $h \equiv 1$ であるすべての指数分布族で成り立つ。$\hat{\eta}$ における平均の負の対数尤度は $A(\hat{\eta}) - \hat{\eta}^{\top}\bar{T}$ である一方、当てはめられた分布のエントロピーは $A(\hat{\eta}) - \hat{\eta}^{\top}\nabla A(\hat{\eta})$ である。$\nabla A(\hat{\eta}) = \bar{T}$ であるから、この 2 つの数値は一致する。

最小点における損失の二階導関数は $13A''(\hat{\eta}) = 13 \cdot \frac{36}{169} = \frac{36}{13} \approx 2.77$ である。曲率が大きいということは、$\hat{\eta}$ から離れると損失が急激に増加し、対数オッズが狭い範囲に絞り込まれることを意味する。13 回のコイントスではまだかなりの不確定性が残っており、$1/\sqrt{2.77} \approx 0.60$ だけ $\hat{\eta} \approx 0.81$ から動かしても、損失は約 0.5 nat しか増えない。36.6 節ではこの曲率をフィッシャー情報量として位置づけ、エラーバーへと変換する。Figure 36.20 に、A の $\hat{\eta}$ における接線と負の対数尤度の形状を示す。

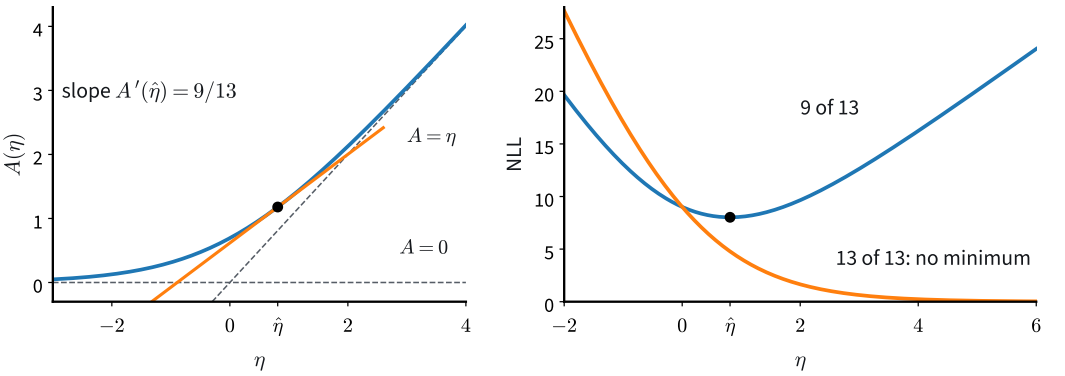


Figure 36.20: 左：ベルヌーイ分布の対数分配関数 $A(\eta) = \log(1 + e^{\eta})$ とその漸近線、および $\hat{\eta} = \log(9/4)$ での接線（傾きは平均 $9/13$）。右：負の対数尤度 $13A(\eta) - x\eta$ のグラフ。13 回中 $x = 9$ 回表が出た場合は最小点 $\hat{\eta}$ を持ち、$x = 13$ 回表が出た場合はすべての η に対して減少し続け、有限の最小点を持たない。

ここで重要な注意点がある。モーメントマッチング方程式 $\nabla A(\hat{\eta}) = \bar{T}$ が解を持つのは、$\bar{T}$ がその分布族のいずれかの分布によって実際に達成可能な平均である場合に限られる。ベルヌーイ分布のコインでは、達成可能な平均は $\text{sigmoid}(\eta)$（η は実数）、すなわち開区間 $(0, 1)$ である。もし 13 回のコイントスがすべて表になったなら、$\bar{T} = 1$ は境界上にあり達成できない。このとき負の対数尤度は $13A(\eta) - 13\eta = 13\log(1 + e^{-\eta})$ となり、あらゆる η に対して単調に減少する。すなわち、値は 9.01（$\eta = 0$ のとき）、0.087（$\eta = 5$ のとき）、0.0006（$\eta = 10$ のとき）と下がっていく。0 に漸近するが決して到達しないため、有限の最小点は存在せず、当てはめられた確率は 1 へと近づいていく（$\hat{\eta} \to \infty$ のとき）。

ロジスティック回帰も、訓練セット内の 2 つのクラスを超平面で完全に分離できるときにはまったく同じ壁に突き当たる。重みを大きくするほど損失が減り続け、重みが無限大へと発散してしまうのである。パラメータに事前分布を入れる（本節の最後で導入する）か、重み減衰を用いれば有限の最適解が回復する。詳細は演習問題 4（36.6 節）で確認する。

最小点が存在する場合、負の対数尤度が狭義凸であればその最小点は一意である。これは $\text{Cov}(T)$ が正定値であるとき、すなわち十分統計量の自明でないいかなる線形結合も台全体で定数にならないときに成り立つ。このようなパラメータ化は**最小**（minimal）と呼ばれる。K 個すべてのロジットでパラメータ化したカテゴリカル分布は最小ではない。K 個の one-hot 座標の和は常に 1 になるため、すべてのロジットに同じ定数を加えても確率は一切変化しないからである。その結果、負の対数尤度は $\mathbf{1}$ の方向に平坦になり、最小点は 1 点ではなく直線をなす。それでも深層学習ライブラリはあえて K 個すべてのロジットを保持し、このシフト不変性を利用して指数をとる前に最大のロジットを引くことで数値安定性を確保している（3.3.5 節）。

36.4.3.3 一般化線形モデル

教師あり学習のモデルは、ラベル y の入力 $\mathbf{x}$ が与えられたもとでの条件付き分布を予測する。**一般化線形モデル**（generalized linear model）は、その分布を指数分布族 Equation 36.51 から選び、入力を線形変換 $\mathbf{W}^{\top}\mathbf{x}$ を通してモデルに組み込む（バイアス項は定数の特徴量に吸収させる）。最も明快な選択は、この線形変換をそのまま自然パラメータに等しくおく $\eta = \mathbf{W}^{\top}\mathbf{x}$ であり、この組み合わせは**正準リンク**（canonical link）と呼ばれる。モデルが予測する十分統計量の平均を $\mu(\mathbf{x}) = \nabla A(\eta) = \mathbb{E}[T(y) \mid \mathbf{x}]$ と書くと、1 つの例に対する負の対数尤度は $A(\eta) - \eta^{\top} T(y) - \log h(y)$ となる。モーメントの性質 Equation 36.54 より、η に関する勾配は単に $\mu(\mathbf{x}) - T(y)$ である。$\eta = \mathbf{W}^{\top}\mathbf{x}$ に連鎖律を適用すると、直ちに次式が得られる。

$$\nabla_{\mathbf{W}} \big(-\log p(y \mid \mathbf{x}) \big) = \mathbf{x} \, \big(\mu(\mathbf{x}) - T(y) \big)^{\top}. \tag{36.56}$$

どの指数分布族を選んでも、勾配は常に入力と予測誤差（予測平均マイナス観測統計量）の積になる。モーメントの性質を手にした今、本節の冒頭で提示した謎はわずか 2 行で解決する。さらに、損失は $\mathbf{W}$ に関して凸である。凸関数 A と $\mathbf{W}$ の線形関数を合成し、そこに線形項を加えたものだからである。ただし、このすっきりした勾配は正準リンクに固有の性質であることに留意しよう。36.4.4 節で扱うプロビットモデルは同じ線形スコアをガウス分布の累積分布関数に入力するため、その勾配には予測値に応じて変化する余分な因子が掛かる。指数分布族を入れ替えるだけで、4 つのなじみ深いモデルが現れる。

ラベル y	分布族	$T(y)$	平均 $\mu(\mathbf{x})$	モデル
実数	ガウス分布（分散 1）	y	η	線形回帰
はい・いいえ	ベルヌーイ分布	$y \in \{0, 1\}$	$\mathrm{sigmoid}(\eta)$	ロジスティック回帰
K クラスのうち 1 つ	カテゴリカル分布	one-hot $\mathbf{y}$	$\mathrm{softmax}(\eta)$	ソフトマックス回帰
カウント	ポアソン分布	y	e^{η}	ポアソン回帰

あまり見慣れないラベルでこの規則を確かめるため、カウントデータに対するポアソンモデルを考えよう。ある入力に対してモデルが平均 $e^{\eta} = 3$ を予測し、観測されたカウントが $y = 5$ だったとすると、η に関する損失の勾配は $3 - 5 = -2$ となる。したがって勾配降下法のステップは η を増加させ、予測平均を 5 に向けて押し上げる。

表の 1 行目では、ガウス分布の分散を 1 に固定することで二次項 $e^{-y^2/2}$ が基底測度 $h(y)$ に組み込まれるため、負の対数尤度は定数項を除いて 2.1 節の二乗損失に一致する。深層ニューラルネットワークは、この最終層（自然パラメータへの線形変換とそれに続く指数分布族の負の対数尤度）をそのまま保ちつつ、生の入力 $\mathbf{x}$ を学習された表現に置き換える。ネットワークの重み全体に関して損失はもはや凸ではないが、出力層の自然パラメータに関する勾配は依然として「予測マイナス観測」のままである (Bishop 2006)。

36.4.4 ソフトマックスの起源

本編を通して、分類器は K 個のロジットからなるベクトル $\mathbf{z}$ を**ソフトマックス**によってクラス確率へと変換してきた。

$$\mathrm{softmax}(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad k = 1, \dots, K. \tag{36.57}$$

この写像が指数分布族の表のカテゴリカル分布の行から現れることは、すでに見たとおりである。ソフトマックスには明らかに望ましい性質がある。各確率は対応するロジットとともに増加し、すべての確率は正で、その和は 1 になる。しかし、そうした性質を持つ写像は他にも数多くある。たとえば $\log(1+e^{z_k})$ を e^{z_k} の代わりに正規化しても、ロジットのランキングを保つ正の確率が得られる。ソフトマックスを際立たせているのは、ロジットの差が対数オッズ $z_i - z_j = \log(p_i/p_j)$ になるという点である。驚くべきことに、異なる 3 つの分野に由来する 3 つの独立した議論がすべてこの同じ式へと収束する。すなわち、人間のノイズを伴う選択のモデル、統計物理学のボルツマン分布、そして指数分布族である。3 番目の議論が最も短く、本節の残りの部分が依拠するのもこれだけである。

36.4.4.1 ノイズを伴う選択とガンベルノイズ

第 1 の議論は、そもそも離散的なラベルがどのように生成されるかをモデル化する。商品を選ぶ買い物客や画像にラベルを付けるアノテータのような意思決定者を想像してみよう。各選択肢 k に潜在的なスコア z_k を割り当てているものの、ときおり誤りを犯すとする。ノイズを伴う選択をモデル化する自然な方法は、各スコアに独立なランダムノイズ ϵ_k を加え、摂動後のスコアが最も高い選択肢を選ぶことである。

$$y = \operatorname*{argmax}_k \, (z_k + \epsilon_k).$$

このモデルのもとでは、クラス k の確率は選択肢 k がノイズ込みの競争に勝つ確率となるため、その具体的な式はノイズの分布によって決まる。たいていの人が最初に試すのはガウスノイズだろう。選択肢が 2 つで $\epsilon_k \sim \mathcal{N}(0, s^2)$ のとき、選択肢 1 が勝つのは $\epsilon_2 - \epsilon_1 < z_1 - z_2$ の場合である。差 $\epsilon_2 - \epsilon_1$ 自体が分散 $2s^2$ のガウス分布に従うため、勝つ確率は次のようになる。

$$P(y=1) = \Phi\Big(\frac{z_1 - z_2}{\sqrt{2}\, s}\Big),$$

ここで Φ は標準ガウス分布の累積分布関数（cumulative distribution function、CDF）を表す（密度は ϕ と書く）。これが**プロビット**モデルであり、選択確率はノイズのスケールを単位として測ったスコア差のみに依存する。

残念ながら、プロビットモデルは選択肢が 3 つ以上になった途端に閉じた形を失う。選択肢 k が残りの $K-1$ 個の競合すべてに勝つ確率は $K-1$ 個のガウス分布の CDF の積の積分となり、初等的な公式が存在しないため、訓練の毎ステップですべての訓練例に対して勾配ともども数値積分しなければならなくなる。

ガウスノイズからガンベルノイズへ切り替えると、その積分は完全に消え去る。標準**ガンベル**分布の CDF は $G(x) = \exp(-e^{-x})$ である。この分布は極値理論において、ガウス分布や指数分布のように指数関数的に減衰する裾を持つ分布から独立に引いた多数のサンプルの最大値を適切にシフト・スケーリングしたときの極限分布として自然に現れる (Gumbel 1954)。さらに McFadden (1974) は、経済的選択をモデル化するためにこの分布を導入した。

命題（ガンベルノイズはソフトマックスを与える）： $\epsilon_1, \dots, \epsilon_K$ **が独立な標準ガンベル確率変数であるならば、ノイズ付き argmax はソフトマックス分布に従う。**

$$P\Big(\operatorname*{argmax}_j \, (z_j + \epsilon_j) = k\Big) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

証明：$S = \sum_j e^{z_j}$ とおく。摂動後の各スコア $z_j + \epsilon_j$ の CDF は $G(t - z_j) = \exp(-e^{z_j} e^{-t})$ である。選択肢 k がスコア $t = z_k + \epsilon_k$ をとったという条件のもとで、独立性より他の $K - 1$ 個の選択肢すべてが t を下回る確率は $\prod_{j \neq k} \exp(-e^{z_j} e^{-t}) = \exp\bigl(-(S - e^{z_k}) e^{-t}\bigr)$ となる。これに密度 $e^{z_k} e^{-t} \exp(-e^{z_k} e^{-t})$（$t = z_k + \epsilon_k$ の密度）を掛けて t について積分し、$v = e^{-t}$ と置換すると次式が得られる。

$$P(y = k) = \int_{-\infty}^{\infty} e^{z_k} e^{-t} \exp\bigl(-S e^{-t}\bigr)\, dt = e^{z_k} \int_0^{\infty} e^{-Sv}\, dv = \frac{e^{z_k}}{S}. \quad \blacksquare$$

この計算がうまくいくのは、ガンベル分布の CDF の積がやはりガンベル分布の CDF になるからである。全 K 個の選択肢にわたる最大スコアは $P\bigl(\max_j (z_j + \epsilon_j) \le t\bigr) = \exp(-S e^{-t}) = G(t - \log S)$ を満たす。つまり勝者のスコアは、標準ガンベル確率変数を $\log\text{-sum-exp} \log \sum_j e^{z_j}$ だけシフトしたものにすぎない。この見方に立てば、log-sum-exp はロジットの滑らかな最大値として働き、その勾配であるソフトマックスは各選択肢がその最大値を達成する確率を与える。演習問題 1 では、競合する指数分布の待ち時間を用いた同じ恒等式の別証明を扱う。

この命題を逆向きに読めば、そのままサンプリングアルゴリズムにもなる。すなわち、$\mathrm{softmax}(\mathbf{z})$ からカテゴリを 1 つサンプリングするには、各ロジットに独立な標準ガンベルノイズを加えて argmax をとればよい。これが広く使われている **Gumbel-max トリック**である。

36.4.4 節の残りでは本編にならい、T を十分統計量ではなく温度（ソフトマックス温度）を表す記号として用いる。$\mathrm{argmax}_k (z_k + T\epsilon_k) = \mathrm{argmax}_k (z_k / T + \epsilon_k)$ が任意の $T > 0$ に対して成り立つため、ガンベルノイズを T 倍にスケーリングすると $\mathrm{softmax}(\mathbf{z}/T)$ からのサンプリングになる。ノイズの大きい意思決定は高い温度に対応するわけである。4 つのロジットに対して 10^5 回の選択を $T = 1$ と $T = 2$ で実際にサンプリングし、経験的な勝率を厳密なソフトマックス確率と比較して Gumbel-max トリックを確かめてみよう。

```
z = np.array([2.0, 1.0, 0.1, -1.0])                    # four logits
softmax = lambda v: np.exp(v) / np.exp(v).sum()
rng = np.random.default_rng()
n = 100_000
for T in (1.0, 2.0):                                    # noise scale = temperature
    winners = np.argmax(z + T * rng.gumbel(size=(n, 4)), axis=1)
    freq = np.bincount(winners, minlength=4) / n
    print(f'T = {T:.0f}: argmax frequencies {freq.round(3)}')
    print(f'      softmax(z / T)       {softmax(z / T).round(3)}'
          f'  max diff {np.abs(freq - softmax(z / T)).max():.3f}')
```

```
T = 1: argmax frequencies [0.639 0.236 0.094 0.031]
      softmax(z / T)       [0.638 0.235 0.095 0.032]  max diff 0.001
T = 2: argmax frequencies [0.451 0.275 0.174 0.099]
      softmax(z / T)       [0.451 0.274 0.174 0.101]  max diff 0.002
```

経験頻度は $\mathrm{softmax}(\mathbf{z}/T)$ と数千分の 1 以内で一致しており、これは 10^5 回のサンプルにおけるモンテカルロ法のノイズ下限ちょうどである。ノイズのスケールを 2 倍にすると、ロジットを半分にしたときとまったく同様に勝率が平坦化される。

選択肢が 2 つの場合、独立な 2 つの標準ガンベル確率変数の差はロジスティック分布に従う。その CDF は $\mathrm{sigmoid}(x) = 1/(1 + e^{-x})$ であるから、この**ロジット**モデルは $P(y = 1) = \mathrm{sigmoid}(z_1 - z_2)$ を与える。シグモイドはスケール調整したガウス分布の CDF に極めてよく追随するため（$|\mathrm{sigmoid}(1.7x) - \Phi(x)| < 0.01$ がすべての x で成り立つ）、当てはめたプロビッ

トモデルとロジットモデルは通常の訓練例に対してほぼ同一の確率を出力する。両者が大きく異なるのは裾の振る舞いである。確信度高く誤ラベルが付いた外れ値に対し、ロジットモデルの損失はスコア差に関して線形にしか増えないのに対し、プロビットモデルの損失は二次関数的に増大するため、外れ値がプロビットモデルの当てはめをはるかに強く引きずってしまう（演習問題4）。

36.4.4.2 温度とボルツマン分布

第 2 の議論は統計物理学に由来する。統計物理学でも、まったく同じ指数法則がエネルギー状態を支配している。気体中の分子のように温度 T で熱平衡にある物理系は、状態 k（エネルギー E_k）を次の確率でとる。

$$p_k = \frac{\exp(-E_k/(k_{\mathrm{B}}T))}{\sum_j \exp(-E_j/(k_{\mathrm{B}}T))},$$

ここで k_{B} はボルツマン定数である。エネルギーの低い状態ほど出現しやすく、温度 T は確率が低エネルギー状態へどれほど鋭く集中するかを制御する。Boltzmann は気体分子に対してこの分布を発見し、Gibbs (1902) がその上に統計力学を築き上げた。温度をエネルギーの単位で測り（$k_{\mathrm{B}} = 1$ とおき）、エネルギーを負のロジットと同一視すれば（$E_k = -z_k$）、ボルツマン分布は $\mathrm{softmax}(\mathbf{z}/T)$ になる。この言葉で言えば、ロジットとは負のエネルギーであり、最も起こりやすいクラスとは最低エネルギー状態にほかならない。

前節の最大エントロピーの命題を使えば、物理学を一切持ち出さずに同じ式を導ける。物理学者は平均エネルギーを固定してそれ以外には何も仮定しないが、分類における対応する制約は平均ロジット $\sum_k p_k z_k$ を固定することである。K 個のクラスを $h \equiv 1$ の結果とし、ロジット z_k を統計量とみなそう。K 個のクラス上のすべての分布のうち、所定の平均ロジット（$\mathbf{z}$ の最小成分と最大成分の間に厳密に入る値）を達成する最大エントロピー分布は $p_k \propto e^{\eta z_k}$ である。目標平均がロジットの単純平均を上回るときは常に $\eta > 0$ となり、$T = 1/\eta$ とおけば $\mathrm{softmax}(\mathbf{z}/T)$ が復元される。T が ∞ から 0 へと冷えるにつれて、平均ロジットはロジットの単純平均から最大値へと滑らかに上昇する。シミュレーションで用いたロジット $\mathbf{z} = (2, 1, 0.1, -1)$ では、標準のソフトマックス（$T = 1$）は平均ロジット 1.49 を達成し、単純平均 0.525 と最大値 2 の間に位置している。

36.4.4.3 指数分布族としてのカテゴリカル分布

第 3 の議論は、本節の前半ですでに明らかにしたものである。クラスラベルを one-hot ベクトル $\mathbf{y}$ として符号化すると、カテゴリカル分布は $P(\mathbf{y}) = \exp(\mathbf{z}^{\top}\mathbf{y} - A(\mathbf{z}))$、対数分配関数は $A(\mathbf{z}) = \log \sum_k e^{z_k}$ となり、ロジットを自然パラメータとする指数分布族になる。モーメントの性質より、その平均は $\nabla A(\mathbf{z}) = \mathrm{softmax}(\mathbf{z}) = \mathbb{E}[\mathbf{y}]$ である。負の対数尤度は $\mathbf{z}$ に関して自動的に凸となり、Equation 36.56 よりその勾配は単に $\mathrm{softmax}(\mathbf{z}) - \mathbf{y}$ となる。

3 つの議論はすべて同じ関数に行き着くが、それぞれが異なる側面に光を当てている。ノイズを伴う選択は厳密なサンプラを与え、ボルツマン分布は温度スケーリングの意味を説明し、指数分布族は凸な損失と簡潔な勾配をもたらす。

36.4.5 共役事前分布

最尤法は、13 回のコイントスで 9 回表が出たという結果を $9/13$ という 1 つの数値に圧縮する。しかし点推定だけでは、その値にどれほど確信を持てるかはわからない。13 回中 9 回の表でも、1300 回中 900 回の表でも、比率はまったく同じになるからである。知識をコインのバイアスに関

する確率分布として表せばその不確実性を直接捉えられるし、理想的には新しいコイントスが得られるたびにゼロからやり直すのではなく単純な更新で取り込めることが望ましい。

ここからは次節以降の記法に合わせ、バイアス自体を未知の確率変数として扱い θ と書く。データを見る前の θ に関する分布を**事前分布**（prior）と呼ぶ。ベイズ則は事前分布に尤度を掛けて正規化し、$p(\theta \mid x) \propto p(\theta)\,P(x \mid \theta)$ によって**事後分布**（posterior）を作り出す。事後分布が事前分布とまったく同じ分布族に属し、データを観測してもパラメータを更新するだけで済むとき、その事前分布は尤度に対して**共役**（conjugate）であるという。

36.4.5.1 コインに対するベータ事前分布

コインのバイアスに対する事前分布は、$[0,1]$ 上に台を持つ密度でなければならない。θ の関数として見ると、表が x 回（n 回のコイントス中）出る尤度は $\theta^{x}(1-\theta)^{n-x}$ であるから、同じ代数形式を持つ事前分布を選べば指数を足し合わせるだけで尤度と結合できる。その正規化定数には 36.3 節で導入したガンマ関数 $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,dt$ を用いる。これは $\Gamma(z+1) = z\,\Gamma(z)$ および $\Gamma(n+1) = n!$ によって階乗を一般化する関数である。パラメータ $a, b > 0$ を持つ**ベータ**分布は、次の密度を持つ。

$$\operatorname{Beta}(\theta \mid a, b) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{B(a,b)}, \qquad 0 \le \theta \le 1, \tag{36.58}$$

ここで正規化定数 $B(a,b) = \int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta$ は $\Gamma(a)\Gamma(b)/\Gamma(a+b)$ に等しい。密度に θ を掛けると a が 1 増えるだけなので、この正規化定数から平均がたちどころに求まる。

$$\mathbb{E}[\theta] = \frac{1}{B(a,b)}\int_0^1 \theta^{a}(1-\theta)^{b-1}\,d\theta = \frac{B(a+1,b)}{B(a,b)} = \frac{a}{a+b},$$

ここでは $\Gamma(a+1) = a\,\Gamma(a)$ と $\Gamma(a+b+1) = (a+b)\Gamma(a+b)$ を使ってガンマ関数の因子を約分した。同じ工夫を 2 回適用すれば分散も求まり、平均を $m = a/(a+b)$ と書けば次のようになる。

$$\operatorname{Var}(\theta) = \frac{ab}{(a+b)^2(a+b+1)} = \frac{m(1-m)}{a+b+1}.$$

$a+b$ の値が大きいほど密度は平均の周りにより鋭く集中し、$a = b = 1$ のときは $[0,1]$ 上の平坦な一様密度になる。

ここで、表が x 回（n 回のコイントス中）観測されたとしよう。事前分布に尤度を掛けると、指数がそのまま足し合わされる。

$$\underbrace{\theta^{a-1}(1-\theta)^{b-1}}_{\text{prior}} \cdot \underbrace{\theta^{x}(1-\theta)^{n-x}}_{\text{likelihood}} = \theta^{(a+x)-1}(1-\theta)^{(b+n-x)-1},$$

これを正規化すれば、観測されたカウントの分だけパラメータがシフトしたベータ密度に再びなる。

$$\theta \mid x \;\sim\; \operatorname{Beta}\bigl(a+x,\; b+n-x\bigr). \tag{36.59}$$

この更新には明快な**擬似カウント**（pseudo-count）としての解釈がある。すなわち、a と b は表と裏の事前のカウントとして働き、表が x 回、裏が $n-x$ 回観測されると、それらが集計値にそのまま加算される。事後平均は次式で与えられる。

$$\mathbb{E}[\theta \mid x] = \frac{a + x}{a + b + n},$$

これは擬似カウントを織り込んだうえでの表の割合にほかならない。この値は事前平均 $a/(a+b)$ と最尤推定量 x/n の間に位置し、x/n への収束は n が大きくなるにつれて進む。

ただし、1 つ微妙な違いに注意しておこう。事前分布が何回のコイントスに相当するかは、どの要約量を計算するかによって変わる。事後分布の両パラメータが 1 を超えるとき、事後最頻値は $(a+x-1)/(a+b+n-2)$ となるため、最頻値に対して事前分布が寄与するのは表 $a-1$ 回と裏 $b-1$ 回である。混乱しないようここで明確に整理しておくと、**a と b は事後平均における擬似カウントであるのに対し、$a-1$ と $b-1$ は事後最頻値における擬似カウントである**。一様事前分布 $\mathrm{Beta}(1,1)$ のもとでは、平均には仮想的な表 1 回と裏 1 回が含まれるが、最頻値には一切含まれない。

一様事前分布のもとで本章のコインに戻ると、13 回のコイントスで 9 回表が出た結果により、$\mathrm{Beta}(1,1)$ は $\mathrm{Beta}(10,5)$ へと更新される。その事後平均は $10/15 = 2/3$ であり、これは**ラプラスの継起則**（Laplace's rule of succession）として知られる古典的な公式 $(x+1)/(n+2)$ である (Laplace 1814)。最頻値 $9/13$ は最尤推定量と一致し、標準偏差は $\sqrt{(2/3)(1/3)/16} \approx 0.118$ となる。

もし代わりに 130 回のコイントスで 90 回表が出たなら、事後分布はより鋭い $\mathrm{Beta}(91,41)$ となり、平均は 0.689、標準偏差は 0.040 になる。10 倍の回数のコイントスを集めると、標準偏差は $2.9 \approx \sqrt{133/16}$ 分の 1 に縮小する。すなわち事後分布の不確実性は $1/\sqrt{n}$ のレートで狭まっていき、これは 36.5 節で見る標準誤差と同じスケーリングである。

次のコイントスで表が出る確率には、どの値を割り当てるべきだろうか。θ が与えられたもとではその確率は θ であるから、事後分布の信念にわたって平均をとると $P(\text{heads} \mid x) = \int_0^1 \theta\, p(\theta \mid x)\, d\theta = \mathbb{E}[\theta \mid x]$ となる。先ほどのコインでは、この予測確率は $2/3$ である。ラプラスの継起則は、次に表が出る確率をそのまま与えているのである。

共役性のおかげで、ベイズ則の正規化積分を計算せずに済んだ。では、事前分布が共役でない場合はどうすればよいだろうか。1 次元であれば、事前分布と尤度の積を θ の細かいグリッド上で評価し、その和で正規化するだけでよい。たとえば自然パラメータ上のガウス事前分布 $\eta \sim \mathcal{N}(0,1)$ を考えよう。これは 36.8 節でロジスティック回帰の重みに置くのと同じ種類の事前分布である。この事前分布はコインがおそらく公正に近いという信念を表しており、ベルヌーイ尤度に対して共役ではない。実際、変数変換によって θ に関する密度を求めると $\phi(\eta)/(\theta(1-\theta))$（ただし ϕ は標準ガウス密度）となり、ベータ密度にはならない。同じグリッド上で両方の事前分布のもとでの事後分布を計算し、要約量を比較してみよう。

```
x, n = 9, 13                                  # 9 heads in 13 flips
theta = np.linspace(0, 1, 100_001)[1:-1]      # interior grid points
eta = np.log(theta) - np.log1p(-theta)        # natural parameter
priors = {'Beta(1, 1)': np.ones_like(theta),  # densities in theta
          'eta ~ N(0, 1)': np.exp(-eta**2 / 2) / (theta * (1 - theta))}
for name, prior in priors.items():
    post = prior * theta**x * (1 - theta)**(n - x)
    post /= post.sum()                        # grid weights sum to one
```

```
    mean = post @ theta
    sd = math.sqrt(post @ (theta - mean)**2)
    print(f'{name:13s}: mean {mean:.3f}, sd {sd:.3f}, '
          f'mode {theta[post.argmax()]:.3f}')
```

```
Beta(1, 1)   : mean 0.667, sd 0.118, mode 0.692
eta ~ N(0, 1): mean 0.644, sd 0.111, mode 0.665
```

一様事前分布のもとでは、グリッド計算によって Beta$(10, 5)$ の厳密なモーメント、すなわち平均 0.667、標準偏差 0.118、最頻値 $9/13 \approx 0.692$ が再現される。対数オッズにガウス事前分布を置くと事後分布は公正なコインへと引き寄せられ、平均は 0.644、最頻値は 0.665 に下がり、標準偏差もわずかに縮んで 0.111 となる。グリッド計算なら非共役な事前分布も共役な場合と同じくらい容易に扱えるが、それは θ がスカラーだからにすぎない。高次元のパラメータベクトルにスケールする手法は 36.8 節で展開する。Figure 36.21 は、コイントスが積み重なるにつれてベータ事後分布がどのように鋭くなるかを可視化したものである。

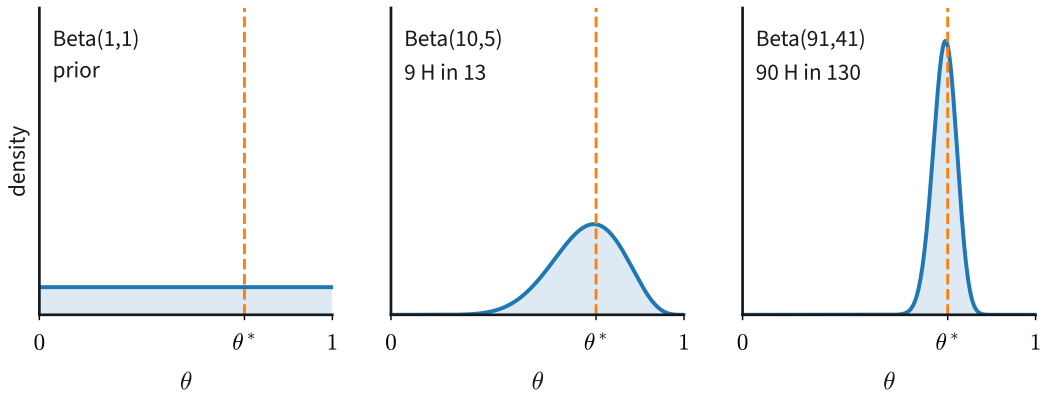


Figure 36.21: コインのバイアス θ のベイズ更新。左：平坦な事前分布 Beta$(1, 1)$。中央：事後分布 Beta$(10, 5)$（真のバイアス $\theta^* = 0.7$（破線）を持つコインを 13 回投げて 9 回表が出た後、平均 $2/3$、標準偏差 0.12）。右：事後分布 Beta$(91, 41)$（130 回投げた後、標準偏差 0.04）。各更新では観測された表と裏の回数が擬似カウントに加算され、事後分布の幅は $1/\sqrt{n}$ のオーダーで縮小する。

36.4.5.2 すべての指数分布族における共役性

一見すると、ベータ分布の更新は都合のよい代数的偶然のようにみえる。しかし実は、これは指数分布族の普遍的な性質である。すべての指数分布族は共役事前分布を持ち、その更新は常に十分統計量の加算に帰着する。

命題（共役事前分布）：尤度 Equation 36.51 と、その自然パラメータ上の次の事前分布を考える。

$$p(\eta) \propto \exp\!\big(\nu^\top \eta - \kappa\, A(\eta)\big), \tag{36.60}$$

ここで $\nu \in \mathbb{R}^d$ と $\kappa \in \mathbb{R}$ は、密度が正規化可能となるように選ぶ。n 個の独立な観測 $\mathbf{x}_1, \dots, \mathbf{x}_n$ を得た後、事後分布はまったく同じ形式を保ち、ν は $\nu + \sum_i T(\mathbf{x}_i)$ に、κ は $\kappa + n$ に更新される。

証明：η に依存しない因子を除くと、尤度からの寄与は $\exp\!\big(\eta^{\top}\sum_i T(\mathbf{x}_i) - nA(\eta)\big)$ である。これに事前分布を掛けると、指数部が 1 行でまとまる。

$$e^{\nu^{\top}\eta - \kappa A(\eta)} \cdot e^{\eta^{\top}\sum_i T(\mathbf{x}_i) - nA(\eta)} = e^{(\nu + \sum_i T(\mathbf{x}_i))^{\top}\eta - (\kappa + n)A(\eta)}.$$

この積を正規化すれば事後分布が得られるため、事後分布は更新後のパラメータ $\nu + \sum_i T(\mathbf{x}_i)$ と $\kappa + n$ を持つ同じ分布族に属する。■

ハイパーパラメータと呼ばれる事前分布のパラメータ ν と κ は、**擬似データ**（pseudo-data）のように振る舞う。すなわち ν は十分統計量の事前の和として、κ は事前の標本サイズとして働く (Bishop 2006)。ベルヌーイ分布のコインでは、この事前分布はロジット $\eta = \log\frac{\theta}{1-\theta}$ で書かれており、η から θ へ変数変換すると $\mathrm{Beta}(\nu, \kappa - \nu)$ に変換される。そして一般の更新則 $\nu \to \nu + x$、$\kappa \to \kappa + n$ は、ベータ分布の更新 $a \to a + x$、$b \to b + n - x$ をそのまま再現する。演習問題 7 では、ヤコビ行列を含めてこの変数変換を具体的に計算する。

36.4.5.3 ガンマ・ポアソン共役とディリクレ・多項共役

広く使われるさらに 2 つの共役ペアも同じパターンに従う。1 つは事象のレートに対するもの、もう 1 つは単語やクラス上のカテゴリカル確率に対するものである。レート λ の関数として見ると、n 個のポアソンカウント $x_1, \dots, x_n$ の尤度は $\lambda^{\sum_i x_i} e^{-n\lambda}$ に比例する。これに対応する共役事前分布は、形状パラメータ $a > 0$ とレートパラメータ $b > 0$ を持つ**ガンマ**分布であり、その密度は $\mathrm{Gamma}(\lambda \mid a, b) = \frac{b^a}{\Gamma(a)}\lambda^{a-1}e^{-b\lambda}$（$\lambda > 0$、平均 a/b）で与えられる。事前分布と尤度を掛け合わせると事後分布 $\mathrm{Gamma}\big(a + \sum_i x_i,\ b + n\big)$ が得られるため、事前分布は仮想的な a 回の事象が仮想的な b 単位期間にわたって生じたことを表している。演習問題 7 では、一般の命題からこのペアを直接導出する。

ベータ事前分布を 2 つの結果から K 個のカテゴリへと一般化するには、確率ベクトル $\mathbf{p}$（ただし $p_k \geq 0$ かつ $\sum_k p_k = 1$）上の密度である**ディリクレ**分布を用いる。集中度パラメータ $a_k > 0$ を持つディリクレ分布の密度は次式で与えられる。

$$\mathrm{Dir}(\mathbf{p} \mid \mathbf{a}) = \frac{\Gamma(\sum_k a_k)}{\prod_k \Gamma(a_k)} \prod_{k=1}^{K} p_k^{a_k - 1},$$

この分布の平均は $\mathbb{E}[p_k] = a_k / \sum_j a_j$ であり、$K = 2$ のときにはベータ分布に帰着する。各カテゴリのカウント $x_1, \dots, x_K$ を観測すると尤度は $\prod_k p_k^{x_k}$ に比例するため、事後分布は $\mathrm{Dir}(\mathbf{a} + \mathbf{x})$ となる。すなわち、各カテゴリの観測カウントがそれぞれの擬似カウントに加算される。

これが実用上なぜ重要かを見るため、3 語の語彙でカウント $(3, 0, 1)$ が得られたとしよう。最尤推定量 $(0.75, 0, 0.25)$ は未出現の 2 番目の単語に確率 0 を割り当ててしまうため、将来その単語を含む文書が現れるとあり得ない文書だと判定してしまう。そこで一様事前分布 $\mathrm{Dir}(1, 1, 1)$ を $\mathbf{p}$ に置くと、事後分布は $\mathrm{Dir}(4, 1, 2)$ となり、その平均は $(4/7, 1/7, 2/7) \approx (0.571, 0.143, 0.286)$ となる。正規化の前にすべてのカウントに 1 を加える**プラス 1 平滑化**（add-one smoothing）は、このディリクレ事前分布のもとでの事後平均にほかならない。36.11 節のナイーブベイズ分類器では、その 2 クラス版を用いる。

閉じた形で事後分布が求まるのは原則というより例外であり、共役ペアの外側ではベイズ推論に数値近似が必要になる（36.8 節）。しかしそれに取り組む前に、点推定に関するより基本的な問いに答えておく必要がある。13 回のコイントスからエラーバーなしの単一の数値 $9/13$ を得たが、この $9/13$ は真のバイアスからどれほど離れ得るのだろうか。また、13 回中 11 回表が出たコインは 9 回表が出たコインと本当に異なるのだろうか。36.5 節ではこれらの問いを取り上げる。

36.4.6 まとめ

分布カタログのほぼすべての分布は指数分布族の形式 $h(\mathbf{x})\exp(\eta^{\top}T(\mathbf{x})-A(\eta))$ を共有しており、そこではデータとパラメータが内積 $\eta^{\top}T(\mathbf{x})$ のみを通して関わり合い、統計量の平均を固定してエントロピーを最大化するとまさにこの形式が現れる。対数分配関数 A を微分するとモーメントが生成される。すなわち ∇A は T の平均であり、$\nabla^2 A$ はその共分散であることから、A が凸であることが直ちに証明される。その結果、最尤法はモデルの十分統計量の期待値を標本平均に一致させる凸最適化問題となり、その標本平均が達成可能である限り有限の解を持つ。指数分布族に正準リンクを組み合わせれば、勾配が常に「予測マイナス観測に入力を掛けたもの」になる一般化線形モデルが得られ、線形回帰、ロジスティック回帰、ソフトマックス回帰、ポアソン回帰が統一されると同時に、プロビットのような非正準リンクが異なる振る舞いを示す理由も説明される。ソフトマックスはカテゴリカル分布族の平均写像として現れる（ガンベルノイズやボルツマン分布からも同様に導かれる）。最後に、すべての指数分布族は十分統計量を加算するだけで更新される共役事前分布を持ち、コインの場合、$\mathrm{Beta}(a,b)$ は事後平均に対して a と b の擬似カウントを、最頻値に対しては $a-1$ と $b-1$ の擬似カウントを寄与する。

36.4.7 演習問題

1. **指数分布の競合：** $X_1,\dots,X_m$ を $X_i\sim\mathrm{Exp}(\lambda_i)$ に従う互いに独立な確率変数とする（たとえば m 本の独立な事象ストリームにおける次の事象までの待ち時間）。
   1. $\min_i X_i\sim\mathrm{Exp}\big(\sum_i\lambda_i\big)$ となることを示せ。
   2. j 番目のストリームが最初に発火する確率が $P(X_j=\min_i X_i)=\lambda_j/\sum_i\lambda_i$ となることを示せ。
   3. $V_1,\dots,V_K$ を独立な $\mathrm{Exp}(1)$ の確率変数とし、$\mathbf{z}\in\mathbb{R}^K$ をロジットのベクトルとする。(b) を用いて、$\operatorname{argmax}_k\,(z_k-\log V_k)$ がソフトマックス確率 Equation 36.57 に従うことを示せ。$-\log V_k$ は標準ガンベル分布に従うため、これはガンベルノイズがソフトマックスを与えることの 2 つ目の証明になっている。
2. **非負整数上の最大エントロピー：** 分布 $(p_0,p_1,p_2,\dots)$（$\{0,1,2,\dots\}$ 上）であって平均 $m>0$ が固定されたものを考える。
   1. シャノンエントロピー $H[p]=-\sum_k p_k\log p_k$ を、制約 $\sum_k p_k=1$ および $\sum_k k\,p_k=m$ のもとでラグランジュ乗数（35.6.1 節）を用いて最大化せよ。停留点が幾何分布 $p_k=(1-r)\,r^k$ となることを示し、r を m で表せ。
   2. $h(k)=1/k!$ に対する相対エントロピー $H_h[p]=-\sum_k p_k\log\big(p_k/h(k)\big)$ について同じ導出を繰り返し、停留点が $\mathrm{Poisson}(m)$ となることを示せ。
   3. $m=4$ において、両方の分布のシャノンエントロピーを数値的に計算せよ。どちらが大きいか、またなぜそうでなければならないか。2 つの分布のもとで $P(X\ge 15)$ を比較せよ。基底測度の選択は、大きなカウントに関してどのような仮定を織り込んでいるか。
   4. ラグランジアンの停留点は最大点であるとは限らない。(a) と (b) の 2 つの分布がそれぞれ平均 m を持つ指数分布族に属することを具体的に示すことで、最大エントロピーの命題から両者が大域的な最大点であることを証明せよ。
3. **対数分配関数からのモーメント計算：** モーメントの性質 Equation 36.54 とその 2 次の形式 $\nabla^2 A=\mathrm{Cov}(T)$ は、モーメントの計算を微分へと帰着させる。
   1. ポアソン分布では $A(\eta)=e^{\eta}$ である。A から平均と分散を導出せよ。関数 A のどの性質によって両者が等しくなるか。
   2. ガウス分布では、自然パラメータは $\eta_1=\mu/\sigma^2$ および $\eta_2=-1/(2\sigma^2)$ である。$A=-\eta_1^2/(4\eta_2)-\tfrac{1}{2}\log(-2\eta_2)$ となることを示し、これを微分して $\mathbb{E}[x]$、$\mathbb{E}[x^2]$、$\mathrm{Var}(x)$、$\mathrm{Cov}(x,x^2)$ を復元せよ。
   3. 位置 μ を固定したスケール b のラプラス分布は、十分統計量 $T(x)=|x-\mu|$ を持つ。

これを Equation 36.51 の形で書き、A を用いて $\mathbb{E}\,|x-\mu|$ と $\mathrm{Var}|x-\mu|$ を計算せよ。得られた結果をラプラス分布の分散 $2b^2$ と照合して確かめよ。

4. ガウス分布からの n 個の観測に対し、2 つのモーメントマッチング方程式 $\nabla A(\hat{\eta}) = \bar{T}$ を書き下し、$\hat{\mu}$ と $\hat{\sigma}^2$ について解け。$n=1$ のときはどうなるか。その答えを、$\bar{T}$ が達成可能な平均であるという条件と関連づけて説明せよ。

4. [code] **プロビットとロジット：** 2 つのクラスに対するスコア差を $m = z_1 - z_2$ とするとき、ロジットモデルは $P(y=1) = \mathrm{sigmoid}(m)$ を与え、ノイズスケール s のプロビットモデルは $P(y=1) = \Phi\big(m/(\sqrt{2}\,s)\big)$ を与える。
   1. 独立な 2 つの標準ガンベル確率変数の差が CDF $\mathrm{sigmoid}(x)$ を持つことを示せ。標準ガンベル分布の分散は $\pi^2/6$ である。標準ガウス分布と同じく分散が 1 になるようなスケール c（CDF が $\mathrm{sigmoid}(cx)$ となる分布に対する値）を求めよ。
   2. スケール c として $\max_x \big|\,\mathrm{sigmoid}(cx) - \Phi(x)\big|$ を最小化する値、およびそのときの最大誤差の値を数値的に求めよ。なぜこのスケールは (a) の値より小さくなるのか。
   3. クラス 2 の例が決定境界の誤った側（$m>0$）にあるとする。ロジットモデルのもとでの負の対数尤度 $-\log \mathrm{sigmoid}(-m)$ が m と $m + e^{-m}$ の間に収まり、かつ m に関する導関数が 1 未満であることを示せ。さらに不等式 $\frac{x}{1+x^2}\,\phi(x) \le 1-\Phi(x) \le \frac{\phi(x)}{x}$ $(x>0)$ を用いて、プロビットモデルのもとでの負の対数尤度 $-\log\Phi\big(-m/(\sqrt{2}\,s)\big)$ が $m^2/(4s^2)$ のオーダーで増大し、その導関数が m に関して線形に増大することを示せ。
   4. 100 個の入力 $x \sim \mathcal{N}(1,1)$（クラス 1）と 100 個の入力 $x \sim \mathcal{N}(-1,1)$（クラス 2）をサンプリングし、$m = wx + b$ とおく。両方のモデルのもとで最尤法により (w,b) を当てはめよ。次に $x=6$ にクラス 2 とラベル付けされた例を 3 個追加し、再び当てはめよ。各モデルについて、これらの誤ラベル例がある場合とない場合の傾き w の比を計算し、モデル間の違いを (c) を用いて説明せよ。なお $\log\Phi$ の計算には `scipy.special.log_ndtr` のような数値的に安定なルーチンを用いよ。

5. **最小カテゴリカル分布族 :●** カテゴリカルな結果を one-hot ベクトル $\mathbf{x} \in \{0,1\}^K$ として表し、クラス確率 $\mathbf{p} = \mathrm{softmax}(\mathbf{z})$ を $T(\mathbf{x}) = \mathbf{x}$、自然パラメータ $\mathbf{z}$、$A(\mathbf{z}) = \log\sum_k e^{z_k}$ の指数分布族として書く。
   1. $\nabla^2 A(\mathbf{z})$ を計算し、これが共分散 $\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$（one-hot ベクトル $\mathbf{x}$ の共分散）に等しいことを示せ。なお、任意の one-hot ベクトルは $\mathbf{x}\mathbf{x}^\top = \mathrm{diag}(\mathbf{x})$ を満たす。$\nabla^2 A(\mathbf{z})\,\mathbf{1} = \mathbf{0}$ となること、およびデータセットに対する負の対数尤度の勾配が $\mathbf{1}$ と任意の $\mathbf{z}$ において直交することを示せ。小さなステップサイズの勾配降下法を初期点 $\mathbf{z}_0$ からロジットに対して実行し、データにはすべてのクラスが含まれているとする。勾配降下法が決して変化させない $\mathbf{z}$ の関数は何か。したがって、無数にある最小点のうちどれに近づくか。
   2. $z_K = 0$ に固定する。すると $\eta = (z_1, \dots, z_{K-1})$ および $T(\mathbf{x}) = (x_1, \dots, x_{K-1})$ を持つ最小表現が得られる。$\nabla^2 A(\eta)$ が、すべての $p_k > 0$ である限り正定値となることを示せ。各クラスのカウント $n_1, \dots, n_K$ が与えられたとき、モーメントマッチング条件 $\nabla A(\hat{\eta}) = \bar{T}$ を最尤法による推定値 $\hat{\eta}$ について解け。あるクラスで $n_k = 0$ のときはどうなるか。

6. [code] **ポアソン回帰 :●** カウントに対する一般化線形モデルは、ポアソン分布の自然パラメータを特徴量のアフィン関数とする。すなわち $y_i \mid \mathbf{x}_i \sim \mathrm{Poisson}(e^{\eta_i})$、$\eta_i = \mathbf{w}^\top \mathbf{x}_i + b$ とおく。
   1. 負の対数尤度 Equation 36.55 を用いて損失を $(\mathbf{w}, b)$ の関数として書き、その勾配とヘッセ行列を計算せよ。損失が凸であることを示せ。勾配が 0 であるという条件は、当てはめられた平均 $\hat{m}_i = e^{\hat{\eta}_i}$ にどのような方程式を課すか。
   2. $n = 1000$ 個の例を $\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$、$\mathbf{w} = (-0.3, 1.0)$、$b = 0.5$ により生成せよ。0 を初期値として $\mathbf{u} = (\mathbf{w}, b)$ をニュートン法（更新ステップ $\mathbf{u} \leftarrow \mathbf{u} - \big(\nabla^2 L(\mathbf{u})\big)^{-1} \nabla L(\mathbf{u})$

を損失 L に対して繰り返す手法）で当てはめ、得られた解において (a) の方程式が成り立つことを確認せよ。

3. 各レートに独立な $g_i \sim \text{Gamma}(1,1)$（平均 1）を掛け、条件付き平均を e^{η_i} のまま保ちつつカウントが過分散になるようにせよ。そのようなデータセット 200 個と、乗数を掛けないデータセット 200 個のそれぞれにポアソンモデルを当てはめよ。データセット間にわたる推定値の平均と標準偏差を比較し、平均の振る舞いを (a) の方程式を用いて説明せよ。

7. **自然パラメータからの共役事前分布：●**スカラーの自然パラメータ η に対し、共役事前分布 Equation 36.60 は $p(\eta) \propto \exp(\nu\,\eta - \kappa\,A(\eta))$ となり、n 個の観測によって $\nu \to \nu + \sum_i T(x_i)$ および $\kappa \to \kappa + n$ と更新される。
   1. ポアソン分布では $\eta = \log\lambda$、$A(\eta) = e^{\eta}$ である。ヤコビ行列（ヤコビアン）を含めて η から λ へ変数変換し、λ 上の事前分布がガンマ分布になることを示せ。その形状パラメータとレートパラメータを特定し、この更新が 36.4.5.3 節のガンマ事後分布を再現することを確かめよ。
   2. ベルヌーイ分布では $\eta = \log\frac{\theta}{1-\theta}$、$A(\eta) = \log(1+e^{\eta})$ である。θ 上の事前分布が $\text{Beta}(\nu, \kappa-\nu)$ になることを示し、これが正規化可能となる (ν,κ) の値の範囲を求めよ。
   3. ロジット上の平坦な事前分布 $\nu = \kappa = 0$ をとる。これは θ 上にどのような密度を誘導するか、またその密度は正規化可能か。表が x 回（n 回のコイントス中）出た後、事後分布が正規化可能となるのはいつか、またその平均はいくらか。これを、一様事前分布のもとでの事後分布 Equation 36.59（$\text{Beta}(1,1)$ を θ 上においた場合）の平均と比較せよ。この比較は「平坦な事前分布」という表現について何を示唆しているか。

36.5 統計学

本書のすべてのモデルは、最終的にホールドアウトデータ上で計算される数値（テスト正解率、検証損失、あるいはベンチマークスコア）によって評価される。この数値はモデル固有の性質として扱われがちだが、実際にはたまたまテストセットに入った特定のデータ例も反映している。ある分類器がテストの 1,000 問中 912 問に正解し、91.2% の正解率を出したとする。同じ情報源から得られた別の 1,000 問に対しては、スコアはわずかに高くも低くもなるだろう。つまり、91.2% という値は実のところ確率変数の 1 つの実現値にすぎない。では、2 つ目のモデルが同じ問題群のうち 921 問に正解したとしたらどうだろうか。このモデルは本当に優れているのか、それともたまたま今回の問題のバッチがこのモデルに有利だっただけなのだろうか。

統計学は、実際には抽出されなかった他のあらゆる標本にわたって推定値がどのように変動するかを問うことで、このような疑問に答える。この変動はある分布、すなわち推定値の**標本分布**（sampling distribution）に従い、本節で扱うすべての道具はこの分布を記述し、それに基づいて推論するために作られている。その中心と広がりから、推定値が系統的に的を外しているか、そしてどれほど広くばらつくかがわかる。仮説検定（hypothesis testing）は、観測された差が偶然だけで生じるばらつきの範囲内に無理なく収まるかどうかを調べる。信頼区間（confidence interval）は、データから棄却できないパラメータ値をすべて集めたものである。そして、標本分布を閉じた式で表せないときには、ブートストラップ法によって手元のデータから直接シミュレーションする。

深層学習においてこれらの道具が重要なのは、どのモデルを運用するか、どのハイパーパラメータが優れているか、新しいアイデアが有効だったかといったエンジニアリング上の意思決定のほとんどが、ノイズを含んだ数値の比較に基づいているからである。それ自体のノイズより小さな差に基づいて行動するのは、コイン投げで決めるのと変わらない。本章のこれまでの節では背後の分布が既知であるときにデータがどう振る舞うかを記述してきたが、本節では逆向きに進み、観測データから誠実なエラーバー付きの推定値を導く。データは外界から得られたものである必

要さえない。コンピュータ上で生成したサンプルを平均して期待値を近似するとき（モンテカルロ推定）、その結果もまた 1 つの標本平均（sample mean）にほかならないため、同じ理論からエラーバーが得られる（36.5.1.7 節）。同様に、テスト正解率は 1 問ごとに 1 回投げられるコインのように振る舞うため、前の節で登場したコイン（9 回の表が 13 回の試行で出た例）も妥当性確認の例として随所に再登場する。

本節から 3 つの考え方だけを覚えるとしたら、次の点に注目してほしい。すなわち、平均二乗誤差（mean squared error）は統計的バイアス（bias）の二乗と分散にきれいに分解されること、p 値は帰無仮説（null hypothesis）が真であるときに一様分布に従うこと、そして信頼区間とは仮説検定が棄却しないパラメータ値の集合にほかならないことである。最初に読む際は、一致性（consistency）とデルタ法（delta method）に関する議論は軽く読み流してよい。ただし、本節全体を通じて 1 つだけ注意点を心に留めておく必要がある。以下に示すあらゆる保証は、推定量（estimator）や検定をデータを**見る前に**選んだことを前提としている。そうでない場合にこれらの保証がいかに容易に崩れるかは、本節の最後で確認する。

```
from scipy.stats import norm
```

```
rng = np.random.default_rng()
```

36.5.1 推定量とその標本分布

先ほどの 91.2% という正解率をどこまで信頼できるかを判断する前に、その数値を生み出した規則に正確な名前を与え、テスト問題がどのように得られたかを表す単純な確率モデルを用意する必要がある。

36.5.1.1 推定量と標本比率

推定量とは、標本 $x_1, \dots, x_n$ を受け取り、数値 $\hat{\theta}_n$ によって未知のパラメータ θ を近似するあらゆる規則のことである。標本平均、標本中央値（sample median）、テストセット上の正答割合はいずれも推定量である。データはランダムであるため $\hat{\theta}_n$ もまた確率変数であり、サイズ n の標本抽出を繰り返したときの分布がその**標本分布**である。

この分類器のテスト正解率をモデル化するため、$x_i = 1$ をモデルが問題 i に正解した場合とし、それ以外の場合は $x_i = 0$ とする。分類器の真の正解率が θ である大きな問題プールからテスト問題が独立に抽出されると仮定すると、指示変数 x_i は平均 θ の独立同分布（i.i.d.）なベルヌーイ確率変数となる。推定値は標本比率（sample proportion）$\hat{\theta}$、すなわち n 問中の正答割合である。期待値の線形性より $\mathbb{E}[\hat{\theta}] = \theta$ であり、1.6 節と同様にその分散は $\mathrm{Var}(\hat{\theta}) = \theta(1-\theta)/n$ となる。言い換えれば、標本比率はまさに真の正解率を中心としており、典型的な誤差は $1/\sqrt{n}$ の速さで縮小する。$\theta = 0.912$ と $n = 1{,}000$ を代入すると、標準偏差は $\sqrt{0.912 \cdot 0.088 / 1000} \approx 0.0090$、すなわち約 0.9 パーセントポイントとなる。中心極限定理（central limit theorem）により、この標本分布はガウス分布でよく近似されるため、約 95% のテストセットにおいて、正解率は幅 ± 1.8 パーセントポイントで θ の周囲に収まる。

数式だけに頼るのではなく、この標本分布を直接観察してみよう。10,000 個の独立なテストセット（各 1,000 問）を、真の正解率が 0.912 である分類器についてシミュレーションし、得られる正解率の平均と広がりを確認する。

```
theta, n = 0.912, 1000
acc = rng.binomial(n, theta, 10000) / n      # accuracy on 10,000 test sets
print(f'mean {acc.mean():.4f}, std {acc.std():.4f}, '
      f'formula {np.sqrt(theta * (1 - theta) / n):.4f}')
```

```
mean 0.9120, std 0.0090, formula 0.0090
```

数式の予測どおり、シミュレーションされた正解率は 0.912 の周りに集まり、標準偏差は 0.0090 となる。実際には抽出しなかった仮想的なテストセット上の正解率を表すこの 10,000 個の数値の集まりこそが、本節のすべての道具が推論の対象とするものである。

ここで、921 問に正解した 2 つ目のモデルに戻ろう。もしそのスコアが別の独立な 1,000 問のテストセットで測定されていたならば、2 つの正解率の間に観測された 0.009 という差の標準偏差は $\sqrt{0.0090^2 + 0.0085^2} \approx 0.012$ となる（ここで 0.0085 は同じ公式を 0.921 で評価した値である）。0.009 の差はそれ自体の標準誤差（standard error）の約 0.7 倍にすぎず、2 つのモデルを区別するには到底足りない。しかし今回の例では、両方のモデルが**同じ** 1,000 問で評価されている。36.5.2.6 節では、この共有テストセットを活用することで比較がどれほど鋭敏になるかを確認する。

36.5.1.2 統計的バイアスと分散

推定量が真の値から外れる仕方は、大きく 2 通りある。平均的に中心からずれた場所を狙ってしまうか、自身の平均の周りに大きくばらつくかである。$\hat{\theta}_n$ の**統計的バイアス**（以下では単にバイアスとも呼ぶ）は 1 つ目の誤差、すなわち推定量の平均値と真のパラメータとの系統的なずれを測るものであり、次式で定義される。

$$\mathrm{Bias}(\hat{\theta}_n) = \mathbb{E}[\hat{\theta}_n] - \theta, \tag{36.61}$$

ここで期待値は標本の無作為抽出に関してとる。起こり得るすべての θ の値に対してバイアスが 0 であるとき、その推定量を**不偏**（unbiased）と呼ぶ。**分散**は 2 つ目の誤差、すなわち $\hat{\theta}_n$ が自身の中心の周りでどれほど広く変動するかを測るものであり、その平方根が**標準誤差**である。

$$\mathrm{Var}(\hat{\theta}_n) = \mathbb{E}\big[(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^2\big], \qquad \mathrm{se}(\hat{\theta}_n) = \sqrt{\mathrm{Var}(\hat{\theta}_n)}. \tag{36.62}$$

端的に言えば、バイアスは推定値が平均してどこに着地するかを表し、標準誤差は 1 回の抽出で得られる典型的な値がその平均からどれほど離れるかを表す。先ほどの正解率の例では、標本比率のバイアスは 0、標準誤差は 0.009 である。分散は θ の周りではなく推定量自身の平均の周りで測られる点に注意しよう。データを完全に無視して常に 0.5 を出力する頑固な規則は分散が 0 であり、その誤差はすべてバイアスから生じる。

36.5.1.3 標本分散

標本比率が不偏推定量（unbiased estimator）であることから、不偏性は自然に成り立つもののように思えるかもしれない。しかし、データから標準誤差を計算しようとした途端に、バイアスを持つ推定量が現れる。標本平均の標準誤差は母分散（population variance）σ^2 に依存するが、これはほとんどの場合未知であり、同じ標本から推定しなければならない。最も自然な選択は**プラグイン分散**（plug-in variance）、すなわち標本平均 $\bar{x}$ からの二乗偏差の平均であり、次式で与えられる。

$$s_0^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

驚くべきことに、s_0^2 は系統的に小さくなりすぎる。それはなぜだろうか。$\bar{x}$ は、**この**特定の標本に対して二乗偏差和を最小化する唯一の点だからである。それとは異なる点である真の平均 μ からの偏差を測れば和は常に大きくなるため、μ を $\bar{x}$ に置き換えると分子が縮小するのである。

自明でない最小の標本サイズである $n = 2$ を考えると、この縮小の大きさは一目瞭然となる。このとき標本平均は単に x_1 と x_2 の中点なので、両方の偏差は $\pm(x_1 - x_2)/2$ に等しくなり、次式が得られる。

$$\sum_{i=1}^{2}(x_i - \bar{x})^2 = \frac{(x_1 - x_2)^2}{2}.$$

x_1 と x_2 は互いに独立で平均 μ、分散 σ^2 を持つため、その差 $x_1 - x_2$ は平均 0、分散 $2\sigma^2$ を持つ。したがって右辺の期待値は σ^2 となる。これを $n = 2$ で割ると平均して $\sigma^2/2$ となり、真の分散の半分にしかならない。代わりに $n - 1 = 1$ で割れば、平均して σ^2 が復元される。驚くべきことに、これとまったく同じ $n - 1$ の補正によって、あらゆる標本サイズ n でバイアスが取り除かれる。

命題（不偏標本分散）。 *$x_1, \dots, x_n$ を平均 μ、有限の分散 σ^2 を持つ独立同分布の確率変数とし、$n \geq 2$ とする。このとき、次が成り立つ。*

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \qquad \text{satisfies} \qquad \mathbb{E}[s^2] = \sigma^2. \tag{36.63}$$

証明。 $\bar{x}$ の周りの偏差と μ の周りの偏差を比較するため、$x_i - \bar{x} = (x_i - \mu) - (\bar{x} - \mu)$ と書いて二乗を展開する。$\sum_i (x_i - \mu) = n(\bar{x} - \mu)$ を用いて交差項を整理すると、次式が得られる。

$$\begin{aligned}\sum_{i=1}^{n}(x_i - \bar{x})^2 &= \sum_{i=1}^{n}(x_i - \mu)^2 - 2(\bar{x} - \mu)\cdot n(\bar{x} - \mu) + n(\bar{x} - \mu)^2 \\ &= \sum_{i=1}^{n}(x_i - \mu)^2 - n(\bar{x} - \mu)^2.\end{aligned}$$

交差項は最後の項の 2 倍に逆符号をつけたものになるため、中心を $\bar{x}$ にとると、μ にとった場合に比べて二乗和から $n(\bar{x} - \mu)^2$ が引かれる。ここで両辺の期待値をとる。各項 $\mathbb{E}[(x_i - \mu)^2]$ は σ^2 に等しいため、第 1 項の和の期待値は $n\sigma^2$ である。一方、標本平均の分散は σ^2/n であるから、$\mathbb{E}[n(\bar{x} - \mu)^2] = \sigma^2$ となる。したがって二乗偏差和の期待値は $(n-1)\sigma^2$ であり、$n-1$ で割ることで $\mathbb{E}[s^2] = \sigma^2$ が得られる。$\blacksquare$

この係数を覚えるには、**自由度**（degrees of freedom）という考え方が役に立つ。n 個の偏差 $x_i - \bar{x}$ には和が 0 になるという制約があるため、自由に動けるのはそのうち $n - 1$ 個だけである。すなわち、データから $\bar{x}$ を推定することで自由度を 1 つ消費している。プラグイン分散 s_0^2 のバイアスは $\frac{n-1}{n}\sigma^2 - \sigma^2 = -\sigma^2/n$ であり、n が小さいときには無視できないが、標本が大きくなるにつれて消えていく。

ここで 1 つ注意点がある。不偏性は非線形変換を施すと保たれない。凹関数である平方根関数にイェンセンの不等式（Jensen's inequality）を適用すると $\mathbb{E}[s] \le \sqrt{\mathbb{E}[s^2]} = \sigma$ となり、s が定数でない限り狭義の不等号が成り立つ。$n = 2$ のガウスデータでは標本標準偏差は $s = |x_1 - x_2|/\sqrt{2}$ であり、その期待値は $\mathbb{E}[s] = \sigma\sqrt{2/\pi} \approx 0.80\,\sigma$ となる。つまり、不偏分散推定量の平方根をとっても、標準偏差は平均して 20% 過小評価される。

数値計算ライブラリでは、これら 2 つの分母を「自由度の補正値（delta degrees of freedom）」フラグで切り替える。`np.var(x, ddof=0)` は n で割り、`np.var(x, ddof=1)` は $n-1$ で割る。

36.5.1.4　バイアス・分散分解

バイアスと分散がどのように相互作用するかを見るため、一方を犠牲にしてもう一方を抑える推定量を意図的に作ってみよう。$n = 30$ 個のノイズを含む測定値 $x_i \sim \mathcal{N}(\theta, \sigma^2)$（ただし $\theta = 1$、$\sigma = 4$）を観測したとする。通常の標本平均 $\bar{x}$ は不偏であり、分散は $\sigma^2/n = 16/30 \approx 0.533$（標準誤差 0.73）となる。ここで**縮小平均**（shrunk mean）$c\,\bar{x}$（ただし係数は $0 < c < 1$）を考える。これは標本平均を 0 の方向へ引き寄せる推定量である。0 へ引き寄せることで $c\theta - \theta = (c-1)\theta$ のバイアスが生じるが、同時に分散は $c^2\sigma^2/n$ へと圧縮される。$c = 0.65$ のときバイアスは -0.35 になる一方、標準誤差は 0.73 から 0.47 へと大きく低下する。Figure 36.22 に両者の標本分布を並べて示す。

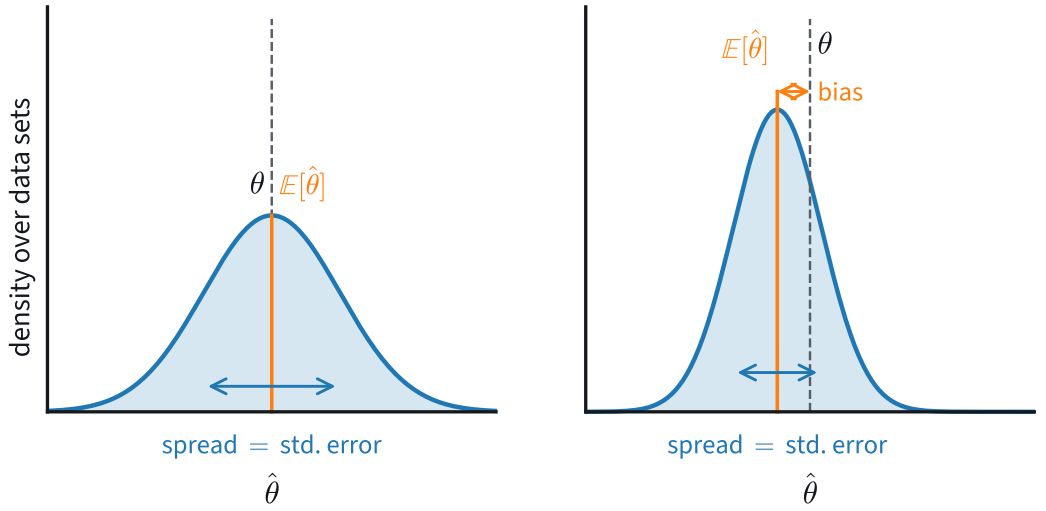


Figure 36.22: $\theta = 1$ に対する 2 つの推定量の標本分布（30 個の観測値を $\mathcal{N}(1, 16)$ から抽出した場合）。左：標本平均 $\bar{x}$（バイアス 0、標準誤差 0.73）。右：縮小平均 $0.65\,\bar{x}$（バイアス -0.35、標準誤差 0.47）。バイアスは中心の θ からのずれであり、標準誤差は分布の幅である。

どちらの推定量が実際には優れているのだろうか。縮小平均は系統的に的を外しているが、ばらつきがずっと小さいため、1 回の抽出で得られる値は多くの場合 θ に対して $\bar{x}$ よりも近くに着地する。この比較に決着をつけるには、両方の誤差を同時に考慮する単一のスコアが必要である。**平均二乗誤差**（MSE）とは、推定量から真のパラメータまでの二乗距離の平均であり、次式で定義される。

$$\mathrm{MSE}(\hat{\theta}_n) = \mathbb{E}\big[(\hat{\theta}_n - \theta)^2\big]. \tag{36.64}$$

都合のよいことに、MSE は先ほど定義した 2 つの量へと綺麗に分解される。

命題（バイアス・分散分解）。 $\hat{\theta}_n$ **が有限の分散を持つならば、次が成り立つ。**

$$\mathrm{MSE}(\hat{\theta}_n) = \mathrm{Bias}(\hat{\theta}_n)^2 + \mathrm{Var}(\hat{\theta}_n). \tag{36.65}$$

証明。 $m = \mathbb{E}[\hat{\theta}_n]$ を推定量の中心とし、二乗の内部で m を足して引く。

$$(\hat{\theta}_n - \theta)^2 = (\hat{\theta}_n - m)^2 + 2(\hat{\theta}_n - m)(m - \theta) + (m - \theta)^2.$$

ここで項ごとに期待値をとる。第 1 項は $\mathrm{Var}(\hat{\theta}_n)$ となり、第 3 項は定数 $\mathrm{Bias}(\hat{\theta}_n)^2$ である。交差項は、$m - \theta$ が定数であり、かつ $\mathbb{E}[\hat{\theta}_n - m] = 0$ が m の定義から成り立つため 0 になる。$\blacksquare$

この恒等式には見事な幾何学的解釈がある。実はこれ自体がピタゴラスの定理にほかならない。確率変数を内積（inner product）$\langle U, V \rangle = \mathbb{E}[UV]$ を備えたベクトルとして扱うと、平均 0 の変動 $\hat{\theta}_n - m$ は任意の定数 a と直交する（$\langle \hat{\theta}_n - m,\, a \rangle = a\,\mathbb{E}[\hat{\theta}_n - m] = 0$ となるためである）。したがって、全誤差 $\hat{\theta}_n - \theta$ をこの変動と定数バイアス $m - \theta$ の和として書くことは、長さが se と $|\mathrm{Bias}|$ である互いに垂直な 2 辺へと分解することを意味し、それらの二乗和が斜辺の二乗 MSE に等しくなる。この視点に立てば、$\mathbb{E}[X]$ は X の定数からなる部分空間（subspace）への直交射影（orthogonal projection）であり、33.1 節で登場した射影において期待値がドット積の役割を果たしたものになっている。Figure 36.23 にこの直角三角形を示す。

この分解を使えば、推定量を定量的に順位付けできる。30 個の観測値からなる先ほどの問題では、公式から予測される MSE は標本平均で 0.533（すべて分散）、縮小平均では $0.123 + 0.225 = 0.348$（$0.65\,\bar{x}$ の場合）となる。標本中央値はどうだろうか。ガウス分布の対称性から不偏ではあるものの、その分散の公式はまだ導いていない。そこで、10,000 個のサイズ 30 のデータセットをシミュレーションし、3 つの推定量をすべて経験的に比較してみよう。

```
theta, sigma, n = 1.0, 4.0, 30
data = rng.normal(theta, sigma, (10000, n))   # 10,000 data sets of size 30
estimates = {'mean': data.mean(axis=1),
             '0.65 * mean': 0.65 * data.mean(axis=1),
             'median': np.median(data, axis=1)}
```

次に、`estimates` に格納した各推定量について、10,000 個のシミュレーションデータセットにわたる経験的な二乗バイアス、分散、および全 MSE を計算し、和 $\mathrm{Bias}^2 + \mathrm{Var}$ を解析的な予測値と比較する。

```
predicted = {'mean': sigma**2 / n,            # bias^2 + variance, by formula
             '0.65 * mean': (0.35 * theta)**2 + 0.65**2 * sigma**2 / n,
             'median': None}                  # no formula yet
print(f'{"estimator":>12} {"bias^2":>7} {"var":>6} {"MSE":>6} {"predicted":>9}')
for name, est in estimates.items():
    bias2 = (est.mean() - theta) ** 2
    var = est.var()                           # a plain average: divides by the count
    mse = ((est - theta) ** 2).mean()
    pred = '-' if predicted[name] is None else f'{predicted[name]:.3f}'
    print(f'{name:>12} {bias2:7.3f} {var:6.3f} {mse:6.3f} {pred:>9}')
```

```
   estimator  bias^2    var    MSE predicted
        mean   0.000  0.521  0.521     0.533
```

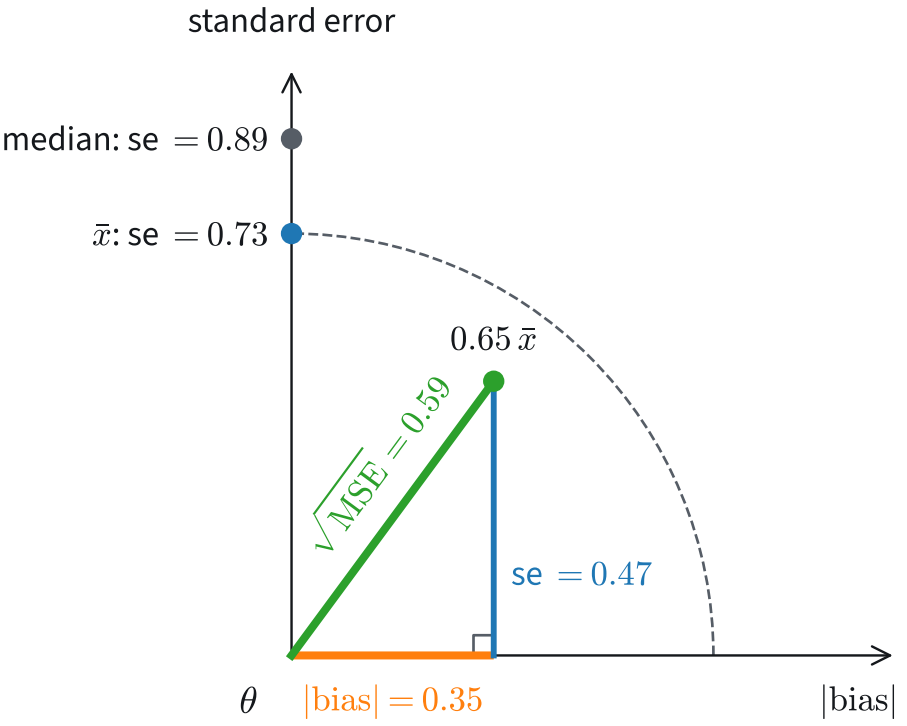


Figure 36.23: $\theta = 1$ の推定量（30 個の観測値を $\mathcal{N}(1, 16)$ から抽出）におけるピタゴラスの定理としてのバイアス・分散分解。各推定量は横座標 $|\text{bias}|$、縦座標 se に位置し、原点からの距離が $\sqrt{\text{MSE}}$ となる。$\bar{x}$ を $0.65\,\bar{x}$ に縮小すると、長さ 0.35 の水平辺が加わる代わりに垂直辺が 0.73 から 0.47 へと短縮され、斜辺は 0.59 に減少し、$\bar{x}$ と同じ精度を表す破線の円の内側に入る。標本中央値は不偏だが、円の外側に位置する。

```
0.65 * mean   0.124  0.220  0.345     0.348
     median   0.000  0.781  0.781         -
```

シミュレーション結果は公式を裏付けている。標本平均の MSE は 0.53 付近であるのに対し、縮小平均は $0.12 + 0.23 = 0.35$ に収まる。標本中央値は約 0.80 と後れをとっており、不偏であるにもかかわらず、その分散は $\bar{x}$ のおよそ 1.5 倍に達している。ここから得られる教訓は明確である。バイアスを持つ推定量が不偏推定量をはっきり上回ることがあり、不偏であることは精度の保証にはならない。

36.5.1.5 縮小平均とトレードオフ

この比較から、ただちに実用上の疑問が生じる。分散を下げる引き換えに、どれだけのバイアスを受け入れるべきなのだろうか。縮小平均 $c\,\bar{x}$ については、このトレードオフを閉じた形で解くことができる。そのバイアスと分散を Equation 36.65 に代入すると、次式が得られる。

$$\mathrm{MSE}(c) = (1-c)^2\theta^2 + c^2\frac{\sigma^2}{n}.$$

c を 0 に向けて縮小していくと、二乗バイアスが増大する一方で分散は減少する。c に関して微分し、$-2(1-c)\theta^2 + 2c\,\sigma^2/n = 0$ とおくことで、最適な縮小係数とその最小誤差が次のように求まる。

$$c^\star = \frac{\theta^2}{\theta^2 + \sigma^2/n}, \qquad \mathrm{MSE}(c^\star) = \frac{\theta^2\,\sigma^2/n}{\theta^2 + \sigma^2/n} < \frac{\sigma^2}{n}.$$

最適な縮小（shrinkage）は縮小しない標本平均を常に上回り、とりわけノイズ σ^2/n が信号 θ^2 に比べて大きいときに効果が最も大きくなることに注目しよう。これまでの数値例（$\theta = 1$、$\sigma = 4$、$n = 30$）では、この公式から先ほど選んだ係数である $c^\star = 1/1.533 \approx 0.65$ が得られ、MSE は 0.533 から 0.348 へと削減される。Figure 36.24 は、両方の誤差成分とその和を c に対してプロットしたものである。ここでは減少する放物線と増加する放物線の単純な和になっているため、典型的な U 字型曲線が厳密に現れる。

もちろん、ここには落とし穴がある。$c^\star$ は未知のパラメータ θ に依存するため、データから直接計算することはできない。さらに悪いことに、θ を知らずに固定した縮小係数を選ぶと大きな裏目に出ることがある。もし真の平均が $\theta = 10$ であったなら、同じ $c = 0.65$ を使うと -3.5 のバイアスが生じ、MSE は $12.25 + 0.23 \approx 12.5$ となって、0.533 である $\bar{x}$ より 20 倍以上悪化してしまう。0 への縮小が有効なのは、真の値が実際に 0 の近くにあるときだけである。演習問題 1 では θ の推定値を $c^\star$ に代入するとどうなるかを調べ、演習問題 9 では有名な驚くべき事実を明らかにする。3 次元以上では、同様のデータ駆動型の縮小が**あらゆる** θ の値に対して標本平均を上回るのである。

これと同じバイアス・分散のトレードオフ（bias–variance trade-off）が、教師あり予測をも支配している。任意の固定された入力 x において、無作為なデータセットで訓練されたモデルの予測 $\hat{f}(x)$ は、真の回帰関数 $f^*(x)$ の推定量である。新しいラベルに対する期待二乗誤差（squared error）は、その推定量の MSE にラベルノイズの削減不能な分散を加えたものに等しい（2.6 節）。多くのモデル族において、柔軟性を高めるとバイアスが減る一方で分散が増大し、正則化（regularization）はちょうど縮小係数 c のように働く (Geman et al. 1992)。ただし、現代の過剰パラメータ化されたモデルにおけるテスト誤差（test error）は、必ずしも単純な U 字型を描くとは限らない点に留意しよう。モデルが訓練データに完全に適合する補間閾値（interpolation threshold）を超えると、テスト誤差が再び下降することがある（36.10.3.2 節）。

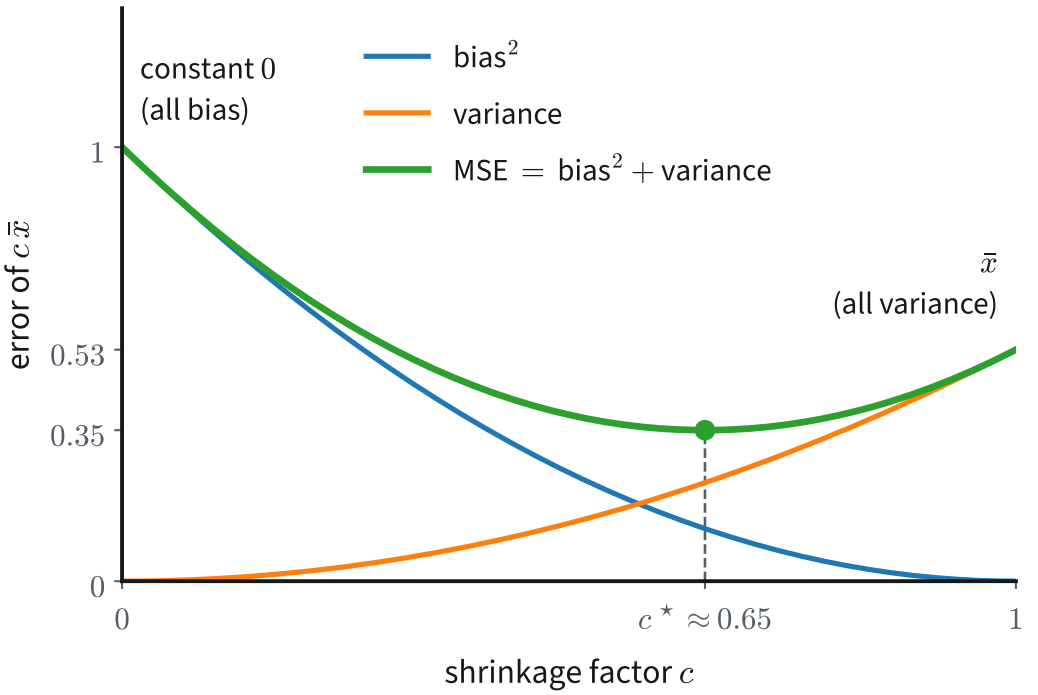


Figure 36.24: 縮小平均 $c\,\bar{x}$ の二乗バイアスと分散を c の関数として描いた図（$\theta = 1$、$\sigma = 4$、$n = 30$）。両者の和である MSE は $c^\star \approx 0.65$ で最小値に達し、0.35 まで低下する（縮小しない標本平均の 0.53、すなわち $c = 1$ との比較）。

36.5.1.6 一致性と有効性

バイアスと分散は、固定された標本サイズ n において推定量を評価するものである。それとは別の問いとして、データを際限なく集めていったときに何が起きるか、すなわち推定値が最終的に真の値へと収束するかどうかが問題になる。推定量 $\hat{\theta}_n$ が**一致推定量**（consistent estimator）であるとは、それが θ に**確率収束**（convergence in probability）することをいい、次式を意味する。

$$P(|\hat{\theta}_n - \theta| > \varepsilon) \to 0 \quad \text{as } n \to \infty, \text{ for every } \varepsilon > 0,$$

これを $\hat{\theta}_n \xrightarrow{P} \theta$ と書く。直感的には、任意の許容誤差 $\varepsilon > 0$ に対して、θ から ε を超えて外れる確率が標本の増大とともに 0 へと縮んでいくということである。

バイアス・分散分解を使えば、一致性を手早く確認できる。マルコフの不等式（36.1 節）を非負の確率変数 $(\hat{\theta}_n - \theta)^2$ と閾値 ε^2 に適用すると、大きな誤差が生じる確率を MSE によって直接上から抑えられる。

$$P(|\hat{\theta}_n - \theta| > \varepsilon) \le \frac{\operatorname{MSE}(\hat{\theta}_n)}{\varepsilon^2} = \frac{\operatorname{Bias}(\hat{\theta}_n)^2 + \operatorname{Var}(\hat{\theta}_n)}{\varepsilon^2}.$$

したがって、$n \to \infty$ のときバイアスと分散の両方が 0 に向かうならば、その推定量は一致推定量であることが保証される。どちらの条件も単独では十分でない。最初の観測値だけを使う $\hat{\theta}_n = x_1$ はすべての n で不偏だが、分散が σ^2 のまま動かないため決して収束しない。逆に $\bar{x} + 1$ は分散が $\sigma^2/n \to 0$ となるものの、誤った値へと収束してしまう。プラグイン分散 s_0^2 は好ましい折衷例を示している。あらゆる有限の n ではバイアスを持つものの、バイアス $-\sigma^2/n$ と分散（4 次モーメントが有限と仮定した場合）はいずれも n が大きくなるにつれて消えるため、一致推定量となる。

この上界を標本平均に適用すると、平均をとる操作が機能する根拠となる基本定理が得られる。

命題（大数の弱法則）。 *$x_1, \dots, x_n$ が平均 μ、有限の分散 σ^2 を持つ独立同分布の確率変数であるとき、標本平均 $\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ は次を満たす。*

$$P(|\bar{x}_n - \mu| > \varepsilon) \le \frac{\sigma^2}{n\,\varepsilon^2} \quad \text{for every } \varepsilon > 0, \qquad \text{so} \qquad \bar{x}_n \xrightarrow{P} \mu. \tag{36.66}$$

証明。 期待値の線形性より $\mathbb{E}[\bar{x}_n] = \mu$ であるから、バイアスは 0 である。各変数は独立であるため和の分散は分散の和に等しく、$\operatorname{Var}(\bar{x}_n) = n\sigma^2/n^2 = \sigma^2/n$ となる。$\operatorname{MSE} = \sigma^2/n$ を上のマルコフの上界に代入すれば目的の不等式が得られ、その右辺は $n \to \infty$ で 0 に収束する。$\blacksquare$

実際には、有限の平均 $\mathbb{E}|x_1| < \infty$ というより弱い仮定のもとでも大数の法則（law of large numbers）は成り立つが、その証明はより長くなる (Wasserman 2013)。この単純な定理は機械学習の多くの基盤を支えている。積分のモンテカルロ推定がなぜ収束するのか、大規模なテストセット上の正解率がなぜ真の正解率を反映するのか、そして平均訓練損失がなぜ期待損失を追跡するのかは、すべてこの定理から説明される。もっとも、この上界自体はかなり緩い。91.2% の正解率を持つ分類器を 1,000 問で評価する場合、テスト正解率が θ から 3 パーセントポイントを超えて外れる確率が高々 $0.0803/(1000 \cdot 0.03^2) \approx 0.09$ であることしか保証されないのに対し、ガウス近似ではその確率は 0.001 付近になる。この不等式の真の強みは一般性にあり、分散が有限であること以外には何も仮定しない。36.9 節では、裾に関してより強い仮定が成り立つときにはるかにタイトな上界を導出する。

一致性は最低限の基準にすぎず、2 つの一致推定量の優劣を区別するものではない。不偏推定量のうち、より分散の小さいものはより**有効**（efficient）であるといい、より少ない標本で同じ精度を達成できることを意味する。ガウスデータでは標本平均も標本中央値も μ に対して不偏だが、n が大きいとき中央値の分散はおよそ $\frac{\pi}{2}\,\sigma^2/n$ となる（演習問題 6）。したがって中央値が平均と同じ標準誤差を達成するには約 $\pi/2 \approx 1.57$ 倍のデータが必要となり、その効率は $2/\pi \approx 64\%$ となる。先ほどの $n = 30$ のシミュレーションでも、すでに 0.80 対 0.53 としてこの比率が示唆されていた。対照的に、標本比率はベルヌーイ分布のパラメータに対するあらゆる不偏推定量の中で最小の分散を達成しており、これはクラメール・ラオの下界（Cramér–Rao bound）として知られる基本的な下限に一致する（36.6.3.4 節）。

36.5.1.7 モンテカルロ推定

ここまでの標本は、テスト問題やノイズを含む測定値、訓練の実行結果といった外部データから得られていた。しかし大数の法則は標本がどこから来たかを問わないため、標本平均は強力な計算手法へと姿を変える。深層学習で登場する期待値のほとんど（期待損失、正規化定数（normalizing constant）、あるいはそのいずれかの勾配）は閉じた形を持たない積分であり、数値積分のグリッドが絶望的な計算量になる数千次元に及ぶことも珍しくない。グリッドを構築できない場合でも、たいていサンプリングは可能である。この発想は最初期の電子計算機にまで遡る (Metropolis and Ulam 1949)。すなわち、$\mathbb{E}[g(X)]$（ただし確率変数 X は密度 p を持つ）を近似するには、独立なサンプル $x_1, \dots, x_n$ を p から抽出し、関数値の平均をとればよい。

$$\mathbb{E}[g(X)] \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i). \tag{36.67}$$

モンテカルロ推定は、評価点を等しい重み $1/n$ でランダムに選んだリーマン和（Riemann sum）とみなすことができ、密度 p が高い領域に自然と多くの点が配置される。大数の法則 Equation 36.66 により、この平均は不偏であり、$\mathbb{E}[g(X)]$ へと $n \to \infty$ で収束する。その収束率（convergence rate）は分散によって決まる。すなわち、$X_1, \dots, X_n$ が密度 p に独立同分布に従い、$\sigma^2 = \mathrm{Var}(g(X))$ が有限であれば、次式が成り立つ。

$$\mathrm{Var}\Big(\frac{1}{n}\sum_{i=1}^{n} g(X_i)\Big) = \frac{\sigma^2}{n}. \tag{36.68}$$

数学的にはここに新しいことは何もない。大数の法則で見た標本平均の分散 σ^2/n において、$g(X_i)$ が x_i の役割を果たしているだけである（36.2 節）。注目すべきはその帰結である。二乗平均平方根（root mean square）誤差は $\sigma/\sqrt{n}$ に従って縮小する。一見するとこれは遅く、精度を 10 進法で 1 桁上げるには 100 倍のサンプルが必要になる。しかし、x の次元が指数に一切現れないこと、すなわちこの次元への非依存性こそが、高次元でサンプリングが機能する理由のすべてである。不可欠な要件は分散が有限であることただ 1 つであり、36.1 節で見たように裾が重い分布では平均が破綻し得る（演習問題 5 を参照）。

これを決定論的なグリッドと比較してみよう。グリッド間隔 ϵ を d 次元で達成するには、$N = \epsilon^{-d}$ 個の評価点が必要になる。34.6 節の左端点則の誤差は $\epsilon = N^{-1/d}$ に比例し、各セルの中心で評価する**中点則**（midpoint rule）のような 2 次精度の公式を使えば $\epsilon^2 = N^{-2/d}$ まで改善される。一方、モンテカルロ法はどの次元でも $N^{-1/2}$ のレートを保つ。1 次元では中点グリッドの圧勝であり、$d = 4$ で指数が並び、$d = 10$ になるとグリッドが $N^{-1/5}$ まで鈍化するのに対し、モンテカルロ法は依然として $N^{-1/2}$ で収束する。分子の σ は d とともに増大し得るものの、指数が悪化することは決してない（演習問題 7）。

この $1/\sqrt{n}$ のレートを具体的な積分で確かめてみよう。$\int_0^1 e^{-x^2}dx = \mathbb{E}[e^{-U^2}]$（ただし U は $[0,1]$ 上の一様分布）を考える。その厳密値は $\frac{\sqrt{\pi}}{2}\,\mathrm{erf}(1) = 0.746824$ である。以下では 10 のべき乗ごとに累積平均を追跡し、実際の誤差を予測スケール $\sigma/\sqrt{n}$ と比較する（ここで $\sigma = \mathrm{sd}(e^{-U^2}) \approx 0.2$ はサンプルから推定する）。

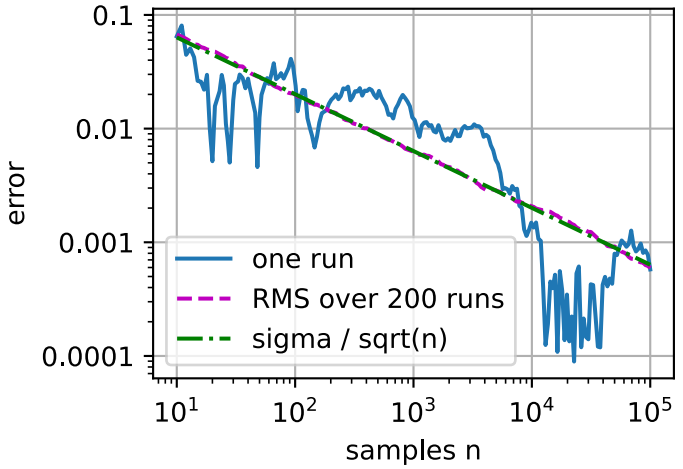
```
exact = math.sqrt(math.pi) / 2 * math.erf(1)
g = np.exp(-rng.uniform(size=10**5)**2)             # g(U), U uniform on [0, 1]
running = np.cumsum(g) / np.arange(1, 10**5 + 1)    # average of the first n
sigma = g.std()
for n in [10, 100, 1000, 10000, 100000]:
    print(f'n={n:<6}  estimate={running[n - 1]:.5f}  '
          f'error={abs(running[n - 1] - exact):.1e}  sigma/sqrt(n)={sigma / n**0.5:.1e}')
print(f'exact (sqrt(pi)/2) erf(1) = {exact:.6f}')
```

```
n=10      estimate=0.81159  error=6.5e-02  sigma/sqrt(n)=6.3e-02
n=100     estimate=0.76842  error=2.2e-02  sigma/sqrt(n)=2.0e-02
n=1000    estimate=0.75833  error=1.2e-02  sigma/sqrt(n)=6.3e-03
n=10000   estimate=0.74830  error=1.5e-03  sigma/sqrt(n)=2.0e-03
n=100000  estimate=0.74625  error=5.8e-04  sigma/sqrt(n)=6.3e-04
exact (sqrt(pi)/2) erf(1) = 0.746824
```

観測された誤差は予測スケール $\sigma/\sqrt{n}$ をよく追跡しているが、各行はあくまで 1 回のランダムな実現値にすぎない。1 回の誤差が $\sigma/\sqrt{n}$ の 2〜3 倍を超えることはまれであり、ときにはそれを大きく下回ることもある。標本サイズにわたって誤差が連続的にどう振る舞うかを見るため、対数等間隔にとった 200 個の n の値における 1 回の累積平均を、200 回の独立な実行にわたる二乗平均平方根誤差（Equation 36.68 が厳密に予測する量）と並べてプロットしてみよう。

```
n = np.unique(np.logspace(1, 5, 200).astype(int))
sq = np.zeros(len(n))
for _ in range(200):                                # 200 independent runs
    g_run = np.exp(-rng.uniform(size=10**5)**2)
    sq += (np.cumsum(g_run)[n - 1] / n - exact)**2
rms = np.sqrt(sq / 200)                             # root-mean-square error
ratio = rms * np.sqrt(n) / sigma
print(f'RMS error / (sigma/sqrt(n)) over all n: {ratio.min():.2f} to {ratio.max():.2f}')
d2l.plot(n, [np.abs(running[n - 1] - exact), rms, sigma / np.sqrt(n)],
         'samples n', 'error', xscale='log', yscale='log',
         legend=['one run', 'RMS over 200 runs', 'sigma / sqrt(n)'])
```

```
RMS error / (sigma/sqrt(n)) over all n: 0.92 to 1.11
```



2 本の曲線の違いに注目しよう。単一の累積平均は、近い n における値がほぼ同じサンプルを共有しているため理論線の上下をさまよい、誤差が 0 を横切るたびに鋭く落ち込む。対照的に、200 回の独立な実行にわたって平均した二乗平均平方根誤差は、$\sigma/\sqrt{n}$ から約 20% 以内の範囲にどの n でも収まり、4 桁の範囲すべてにわたって傾き $-\frac{1}{2}$ で着実に減少していく。

36.5.2 仮説検定

エラーバーは単一の推定値がどれほど不確かか教えてくれるが、機械学習における意思決定の大半は 2 つの推定値の比較である。新しいアーキテクチャやオプティマイザ、正則化の工夫は、本当にベースラインを上回ったのだろうか。本項では、まずシミュレーションによって比較検定をゼロから構築し、続いてその閉じた形の対応物を導出し、最後に真の改善を確実に検出するために実験がどれだけのデータを必要とするかを問う。

36.5.2.1 2 つのモデルの比較

ニューラルネットワークの訓練は本質的にランダムである。重みの初期化（weight initialization）、データのシャッフル、ドロップアウトのマスクはすべて乱数シード（random seed）によって変化するため、まったく同じ訓練スクリプトを 2 回実行してもテスト正解率はわずかに異なる。モデル A とモデル B をそれぞれ 20 個のシードで訓練し、テスト正解率を記録したとしよう。シード間のノイズが 1 回の実行結果を約 1 パーセントポイント変動させる一方で、モデル B は真に 0.8 パーセントポイント優れていると想像してみてほしい。この 40 個の数値だけから、B の方が優れていると判定できるだろうか。自然な要約量は 2 つの平均正解率の差であり、これが**検定統計量**（test statistic）の役割を果たす。

すべての仮説検定は、覆そうとする懐疑的な主張を定式化することから始まる。ここでの**帰無仮説** H_0 は、両方の正解率リストが同じ分布から生じており、2 つのモデルの平均的な性能は等しいと主張する。**対立仮説**（alternative hypothesis）H_A は、両者の分布が異なると主張する。そのうえで次のように問う。もし H_0 が真であったなら、今回観測された大きさ以上の差が生じることはどれほど驚くべきことだろうか。偶然だけではその差がめったに起こらないのであれば、H_0 を**棄却する**（reject）。そうでなければ、それを**棄却しない**（fail to reject）にとどめる。この非対称な言い回しは意図的なものである。法廷での「無罪（not guilty）」判決が潔白を証明するわけではないのと同様に、H_0 を棄却できなかったとしても 2 つのモデルが同等であることが証明されるわけではない。

36.5.2.2 並べ替え検定

観測された差が H_0 のもとで驚くべきものかどうかを判断するには、H_0 が成り立つときのその分布、すなわち**帰無分布**（null distribution）を知る必要がある。驚くべきことに、**並べ替え検定**（permutation test）を使えば、ガウス性やその他のパラメトリックな関数形を一切仮定せずに、データから直接この分布を構築できる (Fisher 1935)。理屈は単純である。もし両方のリストが本当に同じ分布から得られたのであれば、「A」と「B」というラベルには何の情報も含まれない。40 個の数値を 20 個ずつの 2 グループにどのように割り当て直しても、実際に観測された分割とまったく同じ確率で生じたはずである。ラベルをシャッフルして平均の差を再計算する操作を R 回繰り返せば、純粋な偶然が生み出す差の範囲を描き出せる。以下の補助関数は R 回のシャッフルすべてを一度にベクトル化して計算し、シャッフル後の差の絶対値が観測された差の絶対値以上になる回数を数える。

```
def perm_p(acc_a, acc_b, rng, R=2000):
    """Two-sided permutation p-value and the R shuffled gaps."""
    k = len(acc_a)
    observed = acc_b.mean() - acc_a.mean()
    pooled = np.tile(np.concatenate([acc_a, acc_b]), (R, 1))
    shuffled = rng.permuted(pooled, axis=1)    # R relabelings
    gaps = shuffled[:, k:].mean(1) - shuffled[:, :k].mean(1)
    p = (1 + (np.abs(gaps) >= abs(observed)).sum()) / (R + 1)
    return p, gaps
```

観測されたラベル割り当て自体も $R+1$ 個のシャッフルの 1 つに含めると、p **値**が次式で得られる。

$$p = \frac{1 + \#\{r : |\text{gap}_r| \geq |\text{gap}_{\text{obs}}|\}}{R+1},$$

これは H_0 のもとで、実際に観測されたものと同等以上に極端な差が観測される確率を推定している。p 値がごく小さいということは、もしラベルに意味がなければこれほど大きな差はめったに生じないことを意味する。この p 値が**両側**（two-sided）である点に注目しよう。どちらの向きであれ大きな差は H_0 に反する証拠として数えられる（もし B が A を上回るかどうかだけを問うのであれば、片側検定として観測された正の差以上のシャッフル差のみを数えることになる）。

データを見る前に**有意水準**（significance level）α（慣例的には $\alpha = 0.05$）を定め、H_0 を $p \leq \alpha$ のときに棄却すると決めておく。上の設定値を使って 1 回のシミュレーション実験を行い、この検定の動きを見てみよう。

```
k = 20                                  # training seeds per model
acc_a = rng.normal(0.850, 0.010, k)     # per-seed accuracy, model A
acc_b = rng.normal(0.858, 0.010, k)     # model B: a true gap of 0.008
p, gaps = perm_p(acc_a, acc_b, rng, R=10000)
observed = acc_b.mean() - acc_a.mean()
print(f'observed gap {observed:.4f}, permutation p-value {p:.3f}')
```

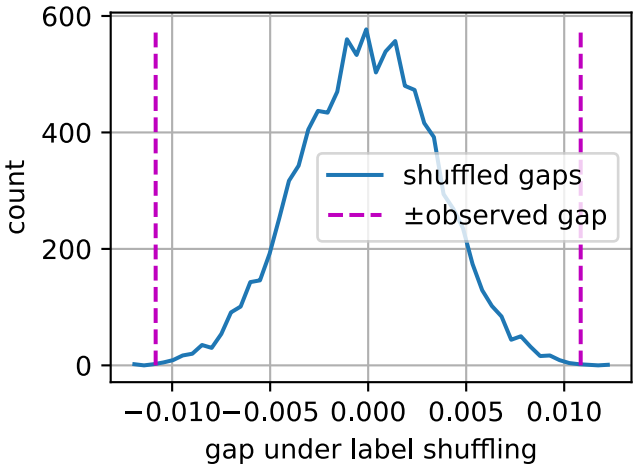
```
observed gap 0.0108, permutation p-value 0.001
```

実験のたびに新しいシードが抽出されるため、観測される差は実行ごとに真の値 0.008 の周りで変動する。大半の実行では p が 0.05 を下回って検定は H_0 を棄却するが、ここでも起こり得るように、無視できない割合の実行では棄却に至らない。1 パーセントポイント丸ごとのシードノイ

ズがある状況では、モデルあたり 20 シードでは 0.8 ポイントの改善を捉えきれないことがある。これがどれくらいの頻度で起きるかは後ほど正確に測定し、それが検定の欠陥ではなく実験の標本サイズを反映したものであることを確認する。

シャッフルされた差のヒストグラムを、観測された差（および符号を反転させた鏡像）と並べてプロットすると、帰無分布を視覚的に確認できる。

```
counts, edges = np.histogram(gaps, bins=50)
centers = (edges[:-1] + edges[1:]) / 2
top = counts.max()
d2l.plot([centers, [observed] * 2, [-observed] * 2],
         [counts, [0, top], [0, top]], 'gap under label shuffling', 'count',
         legend=['shuffled gaps', '±observed gap'],
         fmts=('-', 'm--', 'm--'))
```



並べ替え検定はデータの形状について何も仮定していないにもかかわらず、シャッフルされた差は 0 を中心とする整った釣鐘型の曲線に落ち着き、観測された差は通常その片方の裾野に位置する。何より優れているのは、これとまったく同じ手順が中央値の差、誤り率の差、検証損失の差など、**あらゆる** 2 群間の要約統計量に対してそのまま機能することである（平均を閉じた形で比較する場合、古典的な対応物はウェルチの t 検定である）。あらゆる仮説検定は同じ規律に従う。まず H_0 と H_A を明示し、α を固定して意味のある効果を検出するのに十分な標本サイズを選び、そのうえで初めてデータを収集して p 値を計算し、H_0 を $p \le \alpha$ のときに棄却するのである。

36.5.2.3 z 検定

シャッフルは帰無分布を経験的に構築するが、分散が既知のガウスデータを仮定すれば、p 値と検出率の両方について閉じた形の公式が得られる。$x_1, \ldots, x_n$ が $\mathcal{N}(\mu, \sigma^2)$（$\sigma$ は既知）に独立同分布に従うとし、$H_0 \colon \mu = 0$ を検定したいとする。標本平均をその標準誤差で標準化し、次の検定統計量を作る。

$$T = \frac{\sqrt{n}\,\bar{x}}{\sigma}.$$

H_0 のもとでは $\bar{x} \sim \mathcal{N}(0, \sigma^2/n)$ であるため、T は標準正規分布に従う。標準正規分布の累積分布

関数を Φ と書き、その分位点 z_q（q 分位点）を $\Phi(z_q) = q$ で定義する（たとえば $z_{0.975} \approx 1.96$）。両側 p 値は $2(1 - \Phi(|T|))$ であるから、条件 $p \le \alpha$ は $|T| \ge z_{1-\alpha/2}$ と同値である。$\alpha = 0.05$ では、H_0 が棄却されるのは標本平均が標準誤差の 1.96 倍を超えて 0 から離れたときである。Figure 36.25 はこれら両側の棄却域を示している。

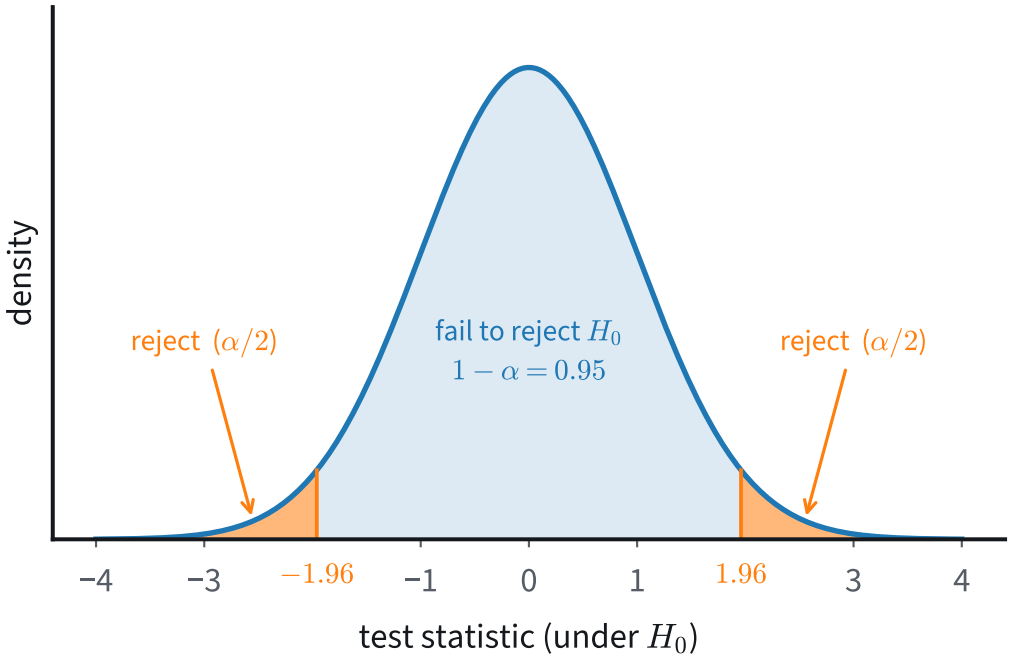


Figure 36.25: 両側 z 検定（$\alpha = 0.05$）の棄却域。H_0 のもとで標準化された統計量は標準正規分布に従い、中央の領域が確率 $1 - \alpha = 0.95$ を占め、± 1.96 を超える両裾が合わせて確率 α を占める。どちらかの裾に入る統計量が観測されることは H_0 のもとではまれであるため、H_0 を棄却する。

先ほどの 2 つのモデルの比較には、この検定の直接的な 2 標本版が対応する。モデルあたり k 個の独立なシードがあり、シードごとの標準偏差が σ であるとき、平均の差 $\bar{x}_B - \bar{x}_A$ の分散は $2\sigma^2/k$ となるため、$T = (\bar{x}_B - \bar{x}_A)/(\sigma\sqrt{2/k})$ は H_0 のもとで標準正規分布に従う。$\sigma = 0.01$ と $k = 20$ を代入すると、差の標準誤差は 0.0032 となる。したがって 0.008 の真の改善は 0 より標準誤差の 2.5 倍だけ上に位置し、1.96 の閾値は超えているものの、余裕のあるマージンとはいえない。

36.5.2.4 2 種類の過誤と検出力

あらゆる仮説検定には 2 通りの誤り方があり、優れた実験計画はその両方を制御下に置く。**第一種の過誤**（type I error、偽陽性）は、真である H_0 を棄却し、存在しない改善を主張してしまうときに生じる。**第二種の過誤**（type II error、偽陰性）は、偽である H_0 を棄却できず、本物の改善を見逃してしまうときに生じる。それぞれの確率は次式で表される。

$$\alpha = P(\text{reject } H_0 \mid H_0 \text{ true}), \qquad \beta = P(\text{fail to reject } H_0 \mid H_0 \text{ false}),$$

ここで β は、0.8 パーセントポイントの真の差といった特定の対立仮説のもとで評価される。余

事象の確率 $1-\beta$ は検定の**検出力**（power、統計的検出力）と呼ばれ、実験がその真の効果を実際に検出できる確率を表す。Figure 36.26 に起こり得る 4 つの結果をまとめる。

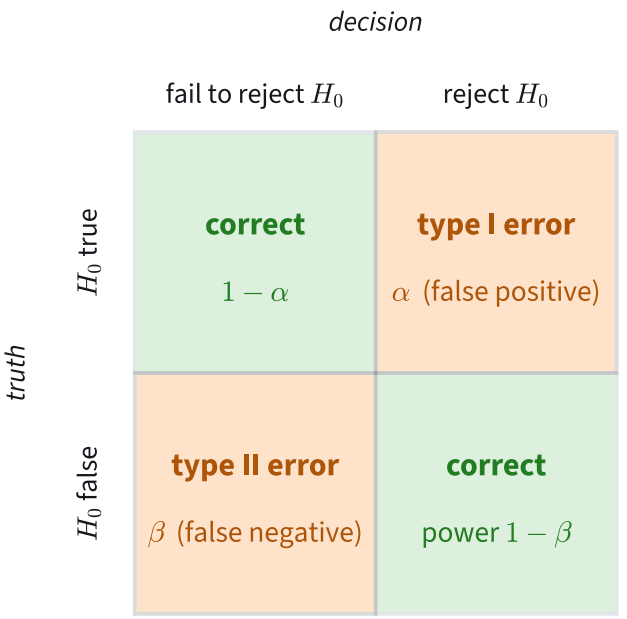


Figure 36.26: 仮説検定の 2×2 決定行列。対角成分が正しい判定、非対角成分が第一種の過誤（確率 α）と第二種の過誤（確率 β）を表し、正しく棄却するセルが統計的検出力（$1-\beta$）を表す。

「$p \le \alpha$ のときに棄却する」という単純な規則によって、なぜ第一種の過誤の確率が厳密に α に保証されるのだろうか。それは H_0 のもとで、p 値が $[0,1]$ 上の一様分布に従うからである。この事実は証明しておく価値がある。有意水準の意味を説明するだけでなく、後で扱う多重検定を理解する鍵にもなるからである。

命題（帰無仮説のもとでの p 値の一様性）。検定統計量 T が連続な累積分布関数 F を H_0 のもとで持つとし、$p = 1 - F(T)$（片側）または $p = 2\min\{F(T), 1-F(T)\}$（両側）とおく。このとき、$P_{H_0}(p \le \alpha) = \alpha$ がすべての $\alpha \in [0,1]$ に対して成り立つ。

証明。 まず $U = F(T)$ が $[0,1]$ 上の一様分布に従うことを示す。任意の $u \in (0,1)$ に対し、t_u を t のうち $F(t) = u$ を満たす最大の値とする。このような値が存在するのは、F が連続であり 0 から 1 まで動くからである。F は非減少であるため、$F(T) \le u$ が成り立つのは $T \le t_u$ のとき、かつそのときに限り、したがって $P(U \le u) = F(t_u) = u$ が得られる。片側 p 値については、ただちに $P(1-U \le \alpha) = P(U \ge 1-\alpha) = \alpha$ が従う。両側 p 値については、事象 $2\min\{U, 1-U\} \le \alpha$ は $U \le \alpha/2$ または $U \ge 1-\alpha/2$ を意味する。互いに排反なこれら 2 つの事象はそれぞれ確率 $\alpha/2$ を持つため、その和事象の確率は α となる。$\blacksquare$

並べ替え検定の p 値については、さらに単純な数え上げの議論から同じ保証が得られる。H_0 のもとでは、観測されたラベル割り当てと R 回のランダムなシャッフルは完全に交換可能（exchangeable）であるため、観測された差は $R+1$ 通りの順位のどれにも等確率で入る。したがって $p \le \alpha$ が起きるのは、観測された差の絶対値が上位 $\lfloor \alpha(R+1) \rfloor$ 個以内に入るときに限られ、$P_{H_0}(p \le \alpha) \le \alpha$ が成り立つ（同順位がある場合は p が大きくなる方向にしか働かない）。分子と分母の $+1$ が重要なのはこのためであり、有意水準 α の検定が真の帰無仮説を棄却する割

合が高々 α に抑えられることを保証している。

では、p 値の分布は H_0 が偽であるときどうなるだろうか。対比を見るため、2,000 回の 2 標本 z 検定（モデルあたり 20 シード）を、まず真の差がない場合、次に真の差が 0.008 である場合についてシミュレーションし、得られる p 値を 10 個の等幅ビンでヒストグラムにしてみよう。

```
se = 0.010 * np.sqrt(2 / k)                 # se of the gap, k seeds per model
for gap in (0.0, 0.008):
    acc_a = rng.normal(0.850, 0.010, (2000, k))
    acc_b = rng.normal(0.850 + gap, 0.010, (2000, k))
    T = (acc_b.mean(axis=1) - acc_a.mean(axis=1)) / se
    p = 2 * norm.sf(np.abs(T))               # two-sided z-test p-values
    hist = np.histogram(p, bins=10, range=(0, 1))[0] / 2000
    print(f'gap {gap:.3f}:', ' '.join(f'{h:.2f}' for h in hist),
          f'  p <= 0.05: {np.mean(p <= 0.05):.2f}')
```

```
gap 0.000: 0.10 0.09 0.10 0.10 0.10 0.09 0.10 0.10 0.11 0.10   p <= 0.05: 0.05
gap 0.008: 0.82 0.07 0.04 0.02 0.02 0.01 0.01 0.00 0.00 0.01   p <= 0.05: 0.72
```

H_0 が成り立つとき（差 0.000）、ヒストグラムは平坦になる。p 値の約 10 分の 1 ずつが各ビンに入り、約 5% が 0.05 以下に収まる。一方、0.008 の真の差があるときには、p 値は 0 の近くに鋭く集中し、約 80% が 0.1 未満に入り、70%〜74% が 0.05 以下に入る。この棄却割合が検定の検出力であり、以下で導く閉じた形の公式からは 0.72 と予測される。

その公式を導出する前に、p 値に関する 2 つの典型的な誤解に注意しておこう (Wasserstein and Lazar 2016)。第 1 に、p 値は**仮に H_0 が真であるとしたときに**これほど極端なデータが観測される確率であり、H_0 が真である確率では**ない**（後者を求めるには事前分布（prior）とベイズの定理が必要になる）。第 2 に、大きな p 値は H_0 が成り立つことを証明するものではなく、単に今回の実験では効果を検出できなかった（おそらく検出力が不足していた）ことを示すにすぎない。

1 標本 z 検定について、この検出力を陽に計算してみよう。真の平均が $\mu \neq 0$ のとき、統計量 T は平均 $\sqrt{n}\,\mu/\sigma$、分散 1 の正規分布に従う。**標準化効果量**（standardized effect）$\delta = |\mu|/\sigma$（ノイズを単位として測った効果量）を定義すると、検定は $|T| \geq z_{1-\alpha/2}$ のときに棄却し、その確率は次式で近似される。

$$1 - \beta \approx \Phi\big(\delta\sqrt{n} - z_{1-\alpha/2}\big),$$

ここでは無視できるほど小さい反対側の裾 $\Phi(-\delta\sqrt{n} - z_{1-\alpha/2})$ を落としている。このように、検出力は効果量 δ および $\sqrt{n}$ とともに上昇する。大きな効果 $\delta = 1$ に対しては、わずか $n = 8$ 個の観測値でも検出力は $\Phi(2.83 - 1.96) \approx 0.81$ に達する。

2 標本のモデル比較では、統計量の平均は $\delta\sqrt{k/2}$（ただし $\delta = 0.008/0.010 = 0.8$）となるため、$z$ 検定の検出力はモデルあたり k シードのとき $\Phi(0.8\sqrt{k/2} - 1.96)$ となり、約 0.72（20 シードの場合）、あるいはわずか 0.24（5 シードの場合）となる。この公式は σ が既知であることを前提としているため、事前に何も推定しない並べ替え検定が各シード数において 400 回の新たなシミュレーション実験でどの程度の検出力を示すか確かめてみよう。

```
def power(k, gap, rng, num_experiments=400):
    """Fraction of fresh experiments in which the permutation test rejects."""
    rejections = 0
    for _ in range(num_experiments):
```

```
            acc_a = rng.normal(0.850, 0.010, k)
            acc_b = rng.normal(0.850 + gap, 0.010, k)
            rejections += perm_p(acc_a, acc_b, rng)[0] <= 0.05
        return rejections / num_experiments

    for k in (20, 5):
        print(f'{k:2d} seeds: rejects in {power(k, 0.008, rng):.2f} of runs, '
              f'formula {norm.cdf(0.8 * np.sqrt(k / 2) - 1.96):.2f}')
```

```
20 seeds: rejects in 0.67 of runs, formula 0.72
 5 seeds: rejects in 0.19 of runs, formula 0.24
```

400 回の実行にわたって、並べ替え検定が棄却するのは約 70%（20 シードの場合）であり、わずか約 20%（5 シードの場合）にとどまる（それぞれの割合は実行ごとに約 2 パーセントポイント変動する）。どちらの数値も z 検定の公式をわずかに下回っているが、これは並べ替え検定が実質的にノイズをデータから推定しており、その分だけ検出力がわずかに下がるためである。つまり 20 シードを使ったとしても、本物の 0.8 ポイントの改善を 10 回中約 3 回は見逃してしまい、わずか 5 シードでは 5 回中 4 回も見逃すことになる。検出力が 0.2 しかない実験で棄却できなかったとしても、そこからはほとんど何もわからない。実務では通常、少なくとも 0.8 の検出力を目指す。

36.5.2.5 標本サイズ

事後になって実験の検出力不足に気づくのではなく、あらかじめ標本サイズ（sample size）を設計しておくことができる。検出力の公式を $1-\beta$ に等しいとおくと $\delta\sqrt{n} - z_{1-\alpha/2} = z_{1-\beta}$ となり、これを n について解けば、標準化効果量 δ を有意水準 α、検出力 $1-\beta$ で検出するために必要な標本サイズが求まる。

$$n = \frac{\left(z_{1-\alpha/2} + z_{1-\beta}\right)^2}{\delta^2}. \tag{36.69}$$

$1/\delta^2$ という急激なスケーリングに注目しよう。検出したい効果量を半分にすると、4 倍のデータが必要になる。標準的な基準である $\alpha = 0.05$、検出力 0.8 では $z_{0.975} \approx 1.960$ および $z_{0.8} \approx 0.842$ であるため、使いやすい経験則として $n \approx 7.85/\delta^2$ に簡略化される。Figure 36.27 に効果量ごとの検出力曲線を示す。

各群 k 個の観測値からなり共通の標準偏差 σ を持つ独立な 2 群を比較する場合、差の分散は 2 倍の $\mathrm{Var}(\bar{x}_B - \bar{x}_A) = 2\sigma^2/k$ になる。それに伴い、1 群あたりに必要な標本サイズも 2 倍になる。

$$k = \frac{2\left(z_{1-\alpha/2} + z_{1-\beta}\right)^2}{\delta^2}, \qquad \delta = \frac{|\mu_B - \mu_A|}{\sigma}. \tag{36.70}$$

$\delta = 0.8$ であった先ほどのモデル比較では、80% の検出力に到達するにはモデルあたり $k = 2 \cdot 7.85/0.64 \approx 25$ シードが必要であり、20 シードではわずかに届かなかった理由がここから説明される。

36.5.2.6 対応のある比較

ここで冒頭の問いに戻ろう。912 問と 921 問（1,000 問中）に正解した 2 つの分類器の比較である。先ほど見たように、両者のスコアを独立なものとして扱うと差はノイズに埋もれてしまった。しかし決定的に重要なのは、両方のモデルが**同じ** 1,000 問に解答していることである。問題の中

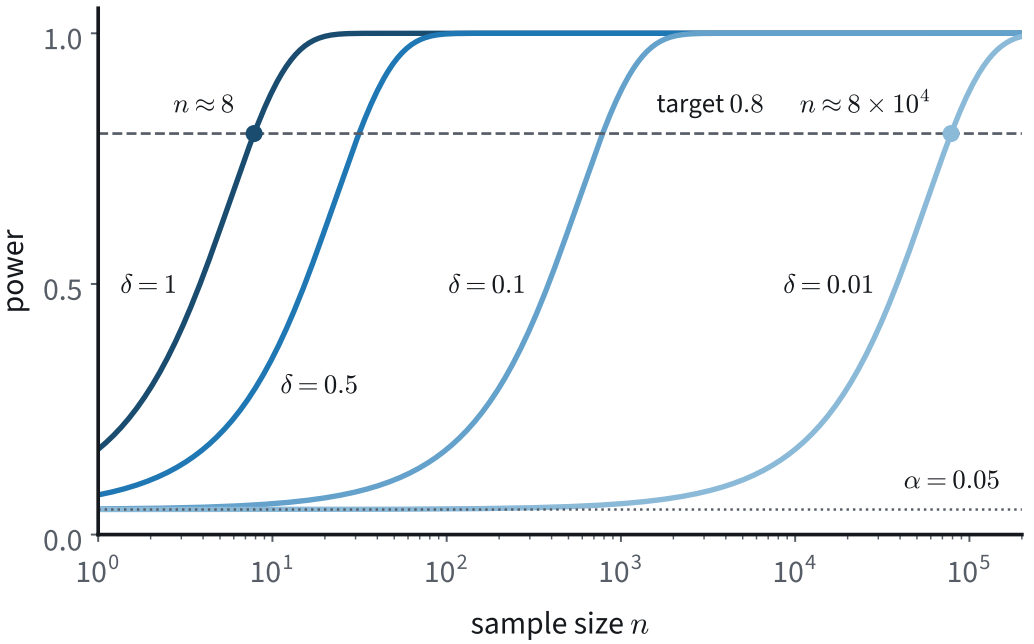


Figure 36.27: 両側 z 検定（$\alpha = 0.05$）の検出力と標本サイズ n の関係（曲線は標準化効果量 δ ごとに描かれている）。各曲線は偶然だけで棄却が起きる α（点線）付近から始まり、1 に向かって上昇する。破線は目標検出力 0.8 を示しており、到達点は $n \propto 1/\delta^2$ に従う。約 8 個の観測値で到達する $\delta = 1$ から、約 78,500 個を要する $\delta = 0.01$ まで大きく変化する。

には両モデルにとって簡単なものもあれば難しいものもあり、対応のない（unpaired）分析ではそうした共有される問題難易度のばらつきまですべてノイズに混ぜ込んでしまう。

対応のある（paired）比較は、問題ごとの差 $d_i \in \{-1, 0, 1\}$（問題 i におけるモデル B の正誤からモデル A の正誤を引いた値）を直接扱うことで、その共有変動を取り除く。両モデルの判定が一致する問題では常に $d_i = 0$ となる。いま 41 問で判定が食い違い、モデル B のみが正解した問題が 25 問、モデル A のみが正解した問題が 16 問で、差し引き $+9$ 問（0.9 パーセントポイント）のリードだったとしよう。2 つのモデルが交換可能であるという帰無仮説のもとでは、各不一致は公平なコイン投げのようなものであり、A と B のどちらに有利になる確率も等しい。したがって適切な帰無シミュレーションは、非ゼロの各 d_i の符号をランダムに反転させる手続きとなり、これを**符号反転並べ替え検定**（sign-flip permutation test）と呼ぶ。この 41 個の不一致に対して実行してみよう。

```
d = np.r_[np.ones(25), -np.ones(16), np.zeros(959)]   # B only, A only, agree
nonzero = d[d != 0]                                    # agreements never change
signs = rng.choice([-1, 1], size=(100000, len(nonzero)))
flipped = (signs * nonzero).sum(axis=1) / len(d)       # gaps under H_0
p = (1 + (np.abs(flipped) >= abs(d.mean())).sum()) / (len(flipped) + 1)
print(f'gap {d.mean():.3f}, sign-flip p-value {p:.2f}')
```

```
gap 0.009, sign-flip p-value 0.21
```

符号反転検定は $p \approx 0.21$ を返す（ランダムな符号反転を厳密な二項確率に置き換えればマクネマー検定になる）。まったく同じ集計値に対する対応のない分析では、差が標準誤差のわずか 0.7 倍にとどまり $p \approx 0.47$ となったのと比べてほしい。対応づけによって差の標準誤差は 0.012 から $\sqrt{41}/1000 \approx 0.006$ へと半減する。両モデルが一致した 959 問がもはやノイズを持ち込まないからである。それでもなお、25 対 16 という 41 問の不一致の内訳は確証を得るには小さすぎる。1,000 問のテストにおける 9 問差のマージンだけでは、モデル B の方が優れているとは結論づけられない。

対応づけが最も威力を発揮するのは、評価をゼロから設計するときである。2 つのモデルの真の正解率が 90% と 91% であり、問題の 4% で判定が食い違うとしよう（B のみ正解が 2.5%、A のみ正解が 1.5%）。このとき d_i は平均 0.01、分散 $\mathbb{E}[d_i^2] - 0.01^2 = 0.04 - 0.0001 = 0.0399$ を持ち、標準化効果量は $\delta = 0.01/\sqrt{0.0399} \approx 0.050$ となる。Equation 36.69 より、対応のある検定は 80% の検出力におよそ $7.85 \cdot 0.0399/0.01^2 \approx 3{,}100$ 問で到達する。対照的に、2 つのモデルを別々の独立なテストセットで評価すると、問題ペアあたりの対応のない分散は $0.9 \cdot 0.1 + 0.91 \cdot 0.09 = 0.1719$ のまま残り、およそ $7.85 \cdot 0.1719/0.01^2 \approx 13{,}500$ 問が**モデルごとに**必要になる。

この例では、対応づけによって各テストセットは 4 分の 1 以下に縮小し、ラベル付けの総労力は 9 分の 1 近くにまで削減される。しかも 2 つのモデルの一致率が高くなるほど、この節約効果はさらに大きくなる。実践上の原則は明快である。競合するモデルは常に同じテストセット上で評価し、問題ごとの差を分析すべきである。同じ原理は乱数シードにも当てはまる。2 つのモデルが同じ訓練シードを共有している場合は、シードごとの差の符号を反転させて検定すればよい（演習問題 5）。

36.5.2.7 多重検定

ここまでの有意性の保証 $P(\text{reject} \mid H_0) = \alpha$ はすべて、事前に選んだ**単一の**検定を実行することを前提としている。しかし機械学習では、ハイパーパラメータのスイープによって一度に何十もの比較を行うのが日常茶飯事である。実際にはベースラインより優れたものが 1 つもない状況で、20 個の新しい設定をベースラインと比較し、それぞれを $\alpha = 0.05$ で検定したとしよう。各検定

が独立であれば、少なくとも 1 つの設定が誤って有意に見えてしまう確率は $1 - 0.95^{20} \approx 0.64$ に達する。勝った設定だけをつまみ食いしたり、$p \le \alpha$ になるまで分析に繰り返し手を加えたりする行為（p **ハッキング**）は、純粋なノイズを論文に載せられそうな見かけ上の改善へと確実に変えてしまう。

最も単純な防御策は**ボンフェローニ補正**（Bonferroni correction）である (Bonferroni 1936)。m 回の検定を行うとき、個々の閾値を α/m に引き下げる。A_j を検定 j が真の帰無仮説を誤って棄却する事象とすると、ユニオンバウンド（union bound）から次式が得られる。

$$P\Big(\bigcup_{j=1}^{m} A_j\Big) \le \sum_{j=1}^{m} P(A_j) = m \cdot \frac{\alpha}{m} = \alpha.$$

この上界がいかに一般的であるかに注目しよう。**1 つでも**偽発見が生じる確率を α 以下に抑えるにあたり、m 個の検定がどのように相関しているかについて一切の仮定を置いていない。20 個の独立な検定を $0.05/20$ で実行すれば、少なくとも 1 つの偽陽性が生じる確率は 0.64 から $1 - (1 - 0.0025)^{20} \approx 0.049$ へと低下する。

残念ながら、m が数百や数千に達すると、m で割ったボンフェローニの閾値は厳しくなりすぎて、本物の発見まで埋もれてしまう。大規模なスクリーニングでは、統計学者は代わりに**偽発見率**（false discovery rate）、すなわち棄却した仮説のうち偽陽性が占める割合の期待値を制御する。ベンジャミニ・ホッホバーグ法 (Benjamini and Hochberg 1995) は、この率を α 以下に保つ（p 値が互いに独立であるか正の従属性を持つ場合）。演習問題 2 でこの規則を導入し、シミュレーションで検証する。

36.5.3 信頼区間

p 値は、単一の候補パラメータ値に関する二択の問いに答えるものである。しかし実務では通常、それ以上の情報、すなわちデータと整合するパラメータ値の全範囲を、推定値の周りのエラーバーとして直接示したい。

36.5.3.1 定義と解釈

信頼区間は、反復実験のうち指定された割合で、固定された未知のパラメータを確実に捕捉する規則によって生成される (Neyman 1937)。形式的には、$1 - \alpha$ 信頼区間とはデータから計算されるランダムな区間 C_n であって、次式を満たすものである。

$$P_\theta(C_n \ni \theta) \ge 1 - \alpha \quad \text{for every } \theta. \tag{36.71}$$

ランダム性がどこにあるかを強調するため、ここではあえて $C_n \ni \theta$ と書き、$\theta \in C_n$ とは書かない。θ は固定された数値であるのに対し、区間 C_n はデータセットごとに移動する。確率 $1 - \alpha$ は**手続き**の性質である。多数の独立な実験にわたって、得られる 95% 信頼区間のうち少なくとも 95% が θ を被覆する。1 回の計算で得られた個々の区間は θ を含んでいるか外しているかのどちらかであり、その区間自体にはどちらが起きたかを示す手がかりは一切ない。Figure 36.28 は、100 個の独立標本から構成した 100 個の区間によってこの様子を示している。実務で使われる多くの区間は、大標本極限においてのみこの保証を満たす。すなわち、**漸近**（asymptotic）$1 - \alpha$ 区間とは、$\liminf_{n\to\infty} P_\theta(C_n \ni \theta) \ge 1 - \alpha$ をすべての θ に対して満たすものである。

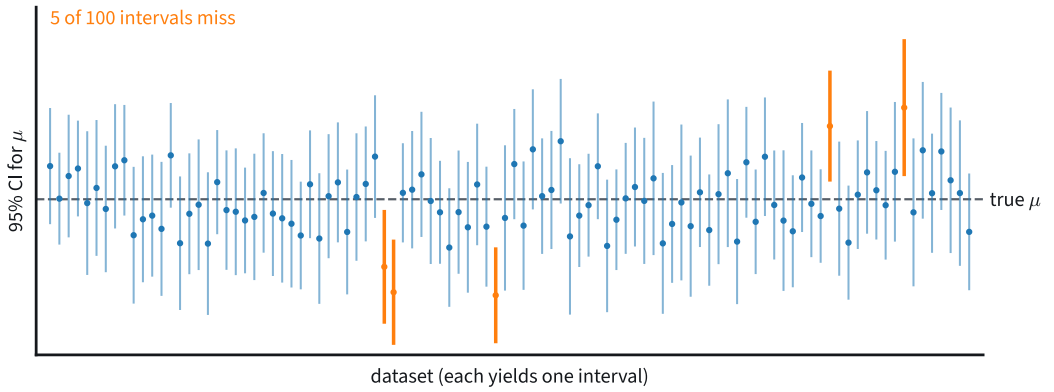


Figure 36.28: それぞれ独立標本から構成された 100 個の 95% 信頼区間を、固定された真の平均（破線）に対してプロットした図。大半の区間は真の値を被覆しているが、オレンジ色の区間は外れている。95% という保証はこの手続きの長期的なヒット率を表すものであり、個々の区間からは成功したかどうかを知ることはできない。

36.5.3.2 ガウス信頼区間

すべての段階が厳密に成り立つ最も明快な場合から始めよう。$x_1, \dots, x_n$ が $\mathcal{N}(\mu, \sigma^2)$（$\sigma$ は既知）に独立同分布に従うとする。$\bar{x} \sim \mathcal{N}(\mu, \sigma^2/n)$ であるから、半幅を $h = z_{1-\alpha/2}\,\sigma/\sqrt{n}$ とおけば、$\bar{x}$ が幅 h 以内で μ の周りに収まる確率は $1-\alpha$ となる。ここでこの不等式を裏返し、μ に関する命題として読み替えてみよう。

$$|\bar{x} - \mu| \le h \quad \Longleftrightarrow \quad \mu \in [\bar{x} - h,\, \bar{x} + h].$$

ゆえに、$P_\mu(\mu \in [\bar{x}-h, \bar{x}+h]) = 1-\alpha$ がすべての μ に対して成り立つ。この単純な代数的反転こそが、信頼区間の背後にある考え方のすべてである。$\alpha = 0.05$ では、この区間は $\bar{x} \pm 1.96\,\sigma/\sqrt{n}$ となる。

現実の問題では σ が既知であることはまれなため、標本標準偏差 s を Equation 36.63 から代入する。s をデータから推定すると、スチューデント化された統計量（studentized statistic）$T = (\bar{x}-\mu)/(s/\sqrt{n})$ はもはや正規分布には従わない。ガウスデータの場合、これはスチューデントの t 分布（自由度 $n-1$）に従う (Student (Gosset 1908))。この分布は、正規分布よりも重い裾を持つことで s に含まれる追加のランダム性を織り込んでいる（ここではこの分布を事実として認めるが、演習問題 4 ではランダムな精度を持つガウス分布として導出する）。$z_{1-\alpha/2}$ を t 分位点 $t_{n-1,1-\alpha/2}$ に置き換えると、次の区間が得られる。

$$\left[\bar{x} - t_{n-1,1-\alpha/2}\,\frac{s}{\sqrt{n}},\; \bar{x} + t_{n-1,1-\alpha/2}\,\frac{s}{\sqrt{n}}\right], \tag{36.72}$$

これはガウスデータに対して、被覆率（coverage）$1-\alpha$ をすべての $n \ge 2$ で達成する。標本サイズが小さいとき、この違いは極めて大きい。$t_{n-1,0.975}$ は 12.71（$n=2$）、2.776（$n=5$）、1.984（$n=100$）となり、1.96 に向かって n の増大とともに収束していく。応用分野全体で定番として使われている区間は、単純に大標本極限を代入した次式である。

$$\left[\bar{x} - 1.96\,\frac{s}{\sqrt{n}},\ \bar{x} + 1.96\,\frac{s}{\sqrt{n}}\right], \tag{36.73}$$

これは漸近 95% 区間であり、あらゆる有限の n においてわずかに狭すぎる。ガウスデータに対する実際の被覆率は $P(|t_{n-1}| \le 1.96)$ であり、$n = 100$ では公称値に近い（0.947）ものの、0.878 まで低下する（$n = 5$ の場合）。

データがガウス分布に従わない場合はどうなるだろうか。中心極限定理（36.3.2.3 節）により、分散が有限である限り $\sqrt{n}(\bar{x} - \mu)/\sigma$ は標準正規分布へ収束し、s も大数の法則により σ へ収束する。したがって、Equation 36.73 は有限の分散を持つ**あらゆる**分布の平均に対する漸近 95% 区間であり続ける。n がどれほど大きければ十分かはデータの形状に依存する。歪んだ分布ではより多くのサンプルが必要になり（演習問題 3）、100% に近い正解率では多くのテストセットで満点が出て標本分散が 0 になるため、はるかに多くのサンプルを要する（演習問題 4）。いずれの場合も半幅は $1/\sqrt{n}$ に比例するため、エラーバーを半分に縮めるには 4 倍のデータが必要になる。

テスト正解率は 0/1 の指示変数の平均であり $s^2 \approx \hat{\theta}(1 - \hat{\theta})$ が成り立つため、同じ公式から $\hat{\theta} \pm 1.96\sqrt{\hat{\theta}(1 - \hat{\theta})/n}$ が得られる。これは**ワルド区間**（Wald interval）として知られている。これを 1,000 問の分類器と、繰り返し登場しているコインの例（9 回の表が 13 回のコイン投げで出た例）について計算してみよう。

```
for n_correct, n in ((912, 1000), (9, 13)):
    theta_hat = n_correct / n
    half = 1.96 * np.sqrt(theta_hat * (1 - theta_hat) / n)
    print(f'{n_correct:3d} of {n:4d}: '
          f'({theta_hat - half:.3f}, {theta_hat + half:.3f})')
```

```
912 of 1000: (0.894, 0.930)
  9 of   13: (0.441, 0.943)
```

分類器については $91.2\% \pm 1.8$ パーセントポイントが得られる。コインについては区間 $(0.44, 0.94)$ が余裕をもって $\frac{1}{2}$ を含んでおり、13 回のコイン投げではコインに偏り（バイアス）があると結論づけるにはまったく足りないことが確認できる。その偏りを ± 0.05 の幅まで絞り込むには、およそ $1.96^2 \cdot 0.69 \cdot 0.31 / 0.05^2 \approx 330$ 回のコイン投げが必要になる。

これを同じコインに対するベイズ的な**信用区間**（credible interval）と対比すると示唆に富む。ベイズ分析では θ 自体を確率変数として扱い、事前分布を置いたうえで、観測されたコイン投げの結果が与えられたときの事後確率（posterior probability）$1 - \alpha$ を含む区間を報告する。一様事前分布のもとでは事後分布（posterior）は $\mathrm{Beta}(10, 5)$ となり（36.4 節）、中央の 95% 信用区間（事後分布の 2.5% 分位点と 97.5% 分位点の間）は $(0.42, 0.87)$ となる。ここでは数値がよく似ているものの、2 つの区間が主張している内容は根本的に異なる。信用区間は、選んだ事前分布のもとで θ が $(0.42, 0.87)$ に入る確率が 0.95 であると述べる。一方、信頼区間は、その構成手続きが固定された真の θ を反復実験の少なくとも 95% で捕捉すると述べており、これは θ が何であるかによらない。また、ワルド区間がその保証を満たすのは漸近的にのみである点にも留意しよう。13 回のコイン投げでは、実際の被覆率は約 91%（$\theta = \frac{1}{2}$ のとき）であり、50% 未満にまで落ち込むこともある（$\theta = 0.05$ のとき）。

被覆率は反復実験にわたる頻度であるため、シミュレーションによって直接検証できる。1,000 個の独立なデータセット（それぞれ $n = 100$ 個の標準正規分布からの観測値）を抽出し、各データセット上で漸近区間 Equation 36.73 を計算して、真の平均 0 を被覆する個数を数えてみよう。

```
mu, n, trials = 0.0, 100, 1000
data = rng.normal(mu, 1.0, (trials, n))          # one row = one data set
mu_hat = data.mean(axis=1)
se = data.std(axis=1, ddof=1) / np.sqrt(n)
hits = np.abs(mu_hat - mu) <= 1.96 * se          # does the interval contain mu?
print(f'first interval ({mu_hat[0] - 1.96 * se[0]:.3f}, '
      f'{mu_hat[0] + 1.96 * se[0]:.3f}) contains 0: {hits[0]}')
print(f'{hits.sum()} of {trials} intervals contain the true mean '
      f'(nominal 950, exact 947)')
```

```
first interval (-0.191, 0.215) contains 0: True
937 of 1000 intervals contain the true mean (nominal 950, exact 947)
```

最初の区間は実行の約 95% で 0 を被覆し、被覆した合計回数は 947（通常 ± 14 以内）を中心として分布し、950 にはならない。この 3 回の不足はランダムなノイズではない。$n = 100$ における厳密な t 分位点は $t_{99,0.975} = 1.984$ であり 1.96 ではないため、1.96 を使った区間の真の被覆率は $P(|t_{99}| \le 1.96) = 0.947$ となるのである。± 14 という 947 の周りのばらつきは、単に二項分布の標準偏差の 2 倍、すなわち $2\sqrt{1000 \cdot 0.947 \cdot 0.053} \approx 14$ にすぎない。これと同じ手順によって、k 個の訓練シードにわたる平均正解率のエラーバーも求められる。シード間の標準偏差を $\sqrt{k}$ で割り、適切な分位点を掛ければよい。

36.5.3.3 検定と信頼区間

1.96 という数値がちょうど 2 回登場したことに気づいただろうか。1 回目は z 検定の棄却閾値として、2 回目はガウス信頼区間の乗数としてである。これは偶然ではなく、検定と信頼区間はまさに同じコインの裏表なのである。

σ が既知のガウスデータを考えよう。両側 z 検定（帰無仮説 $H_0 : \mu = \mu_0$）が H_0 を棄却しないのは、まさに $|T| \le z_{1-\alpha/2}$ のときである（ただし $T = \sqrt{n}(\bar{x} - \mu_0)/\sigma$）。この不等式を変形すると $|\bar{x} - \mu_0| \le h$（ただし $h = z_{1-\alpha/2}\,\sigma/\sqrt{n}$）となり、これは単に $\mu_0 \in [\bar{x} - h,\, \bar{x} + h]$ を意味する。要するに、**95% 信頼区間とは、5% 水準の仮説検定が棄却せずに残す帰無仮説値の集合である**。その被覆保証も、検定の第一種の過誤の確率からただちに従う。すなわち、すべての μ に対して $P_\mu(\mu \in [\bar{x} - h, \bar{x} + h]) = 1 - P_\mu(\text{the test of } H_0 : \mu \text{ rejects}) = 1 - \alpha$ が成り立つ。

この双対性（duality）がこれほど有用なのは、それが**あらゆる**仮説検定の族に対して成り立つからである。すべての候補値 θ_0 に対して有意水準 α の検定（帰無仮説 $H_0 : \theta = \theta_0$）が与えられていれば、検定が棄却しないすべての値 θ_0 を集めることで、同じ 1 行の議論により自動的に $1 - \alpha$ 信頼集合（confidence set）が得られる。陽な公式が使えなかったり不自然な振る舞いをしたりするときには、代わりに検定を反転させればよい。たとえば、あるモデルが 29 問（30 問中）に正解した場合、ワルド区間は不合理にも正解率 1 を超えてはみ出してしまうが、厳密な二項検定を反転させれば $[0, 1]$ の内部に収まる妥当な区間が得られる（演習問題 4）。

またこの双対性は、2 つのモデルを比較する際のより良い習慣も示唆している。$\bar{x}_B - \bar{x}_A \pm 1.96\,\sigma\sqrt{2/k}$（$k$ シードにわたる場合）のように、差に対する信頼区間を報告するのである。この区間が 0 を含まないことは仮説検定が棄却することと同値であると同時に、実際の改善幅がどの程度であり得るかも一目でわかる。

36.5.3.4 デルタ法

実務では $\hat{\theta}$ そのものではなく、その滑らかな関数 $g(\hat{\theta})$（正解率の対数オッズ（log odds）、2 つの損失の比、あるいは相対誤差削減率など）に関心があることも多い。エラーバーを g を通してど

のように伝播させればよいだろうか。g が θ で微分可能であるとき、1 次のテイラー展開（Taylor expansion）から次式が得られる。

$$g(\hat{\theta}) \approx g(\theta) + g'(\theta)\,(\hat{\theta} - \theta).$$

θ の近傍では、関数 g は小さなずれを単にその傾き $|g'(\theta)|$ 倍に拡大・縮小する。標準偏差をとり、$\hat{\theta}$ を未知の θ の代わりに g' へ代入すると、**デルタ法**が得られる (Wasserman 2013)。

$$\mathrm{se}(g(\hat{\theta})) \approx \left|g'(\hat{\theta})\right| \mathrm{se}(\hat{\theta}). \tag{36.74}$$

具体的な数値で試してみよう。正解率が $\hat{\theta} = 0.90$（$n = 1{,}000$ 問）であるとき、標準誤差は $\mathrm{se}(\hat{\theta}) = \sqrt{0.9 \cdot 0.1 / 1000} \approx 0.0095$ である。その対数オッズ $g(\hat{\theta}) = \log \frac{\hat{\theta}}{1-\hat{\theta}} \approx 2.20$ の導関数（derivative）は $g'(\hat{\theta}) = 1/(\hat{\theta}(1-\hat{\theta})) \approx 11.1$ となるため、対数オッズの標準誤差はおよそ $11.1 \cdot 0.0095 \approx 0.105$ となる。

同じ線形化は、推定値のベクトル $\hat{\theta}$（共分散行列（covariance matrix）Σ を持つ）へもそのまま拡張でき、$\mathrm{Var}(g(\hat{\theta})) \approx \nabla g^{\top} \Sigma \, \nabla g$ となる。Σ の非対角成分である共分散（covariance）が重要である点に注意しよう。同じテストセット上で測定した 2 つの損失の比をとる場合、分子と分母は連動して変動するからである。

他のあらゆる線形近似と同様に、デルタ法にも明確な限界がある。第 1 に $g'(\theta) \neq 0$ が必要である。$g(\theta) = (\theta - \tfrac{1}{2})^2$ を $\theta = \tfrac{1}{2}$ で評価したような停留点（stationary point）では 1 次の項が消え、$g(\hat{\theta})$ の変動レートは $1/n$ と速くなって通常の $1/\sqrt{n}$ ではなくなる。第 2 に、$\hat{\theta}$ は g がほぼ線形とみなせる領域内に留まっている必要があり、これは 100% に極めて近い正解率では破綻する（演習問題 4）。そして統計量に扱いやすい導関数がそもそも存在しないときには、まったく別のアプローチ、すなわちリサンプリングが必要になる。

36.5.4 ブートストラップ法

ここまで計算してきた標準誤差はすべて解析的な公式から得られていた。しかし実務で関心を持つ量の多く（標本中央値、AUC、BLEU スコアなど）は、正解率と同様にテストセットごとに変動するにもかかわらず、その分散を表す簡潔な公式を持たない。ブートストラップ法は、標本分布をデータから直接シミュレーションすることでこの問題を解決する。以下では、まず答えがわかっている例でこの手法を検証し、次に単純な公式が存在しない場合へ適用し、最後に示唆に富む 2 つの失敗例を調べる。

36.5.4.1 プラグイン原理

もしデータを生成している真の分布 F がわかっていれば、**いかなる**統計量の標本分布を求めることも造作ない。新しいデータセットを F から何千個も抽出し、それぞれについて統計量を再計算すればよいからである。もちろん、実際には F は未知である。そこで**ブートストラップ法** (Efron 1979) は大胆な置き換えを 1 つ行う。未知の F を**経験分布**（empirical distribution）$\hat{F}_n$、すなわち確率 $1/n$ を n 個の各観測データ点に割り当てる分布で置き換えるのである。サイズ n の新しい標本を $\hat{F}_n$ から抽出することは、手元のデータから n 個の点を**復元抽出**でサンプリングすることにほかならない。アルゴリズム全体は 3 つのステップで完結する。

1. 観測標本から n 個の点を復元抽出して**ブートストラップリサンプル**を作る（元の点のうち複数回現れるものもあれば、一度も現れないものもある）。
2. そのリサンプル上で統計量 $\hat{\theta}^{*}$ を再計算する。

3. これを R 回繰り返し、反復推定値（レプリケート）$\hat{\theta}^*_1, \ldots, \hat{\theta}^*_R$ を集める。

これら R 個のレプリケートの標準偏差が $\mathrm{se}(\hat{\theta})$ の推定値となり、その $\alpha/2$ 分位点と $1-\alpha/2$ 分位点が**パーセンタイル信頼区間**（percentile confidence interval）をなす。Figure 36.29 にこの一連の流れを示す。その単純さの背後には 2 つの重要な仮定がある。観測値が独立同分布であること、そしてデータの小さな摂動に対して統計量が滑らかに変化することである。

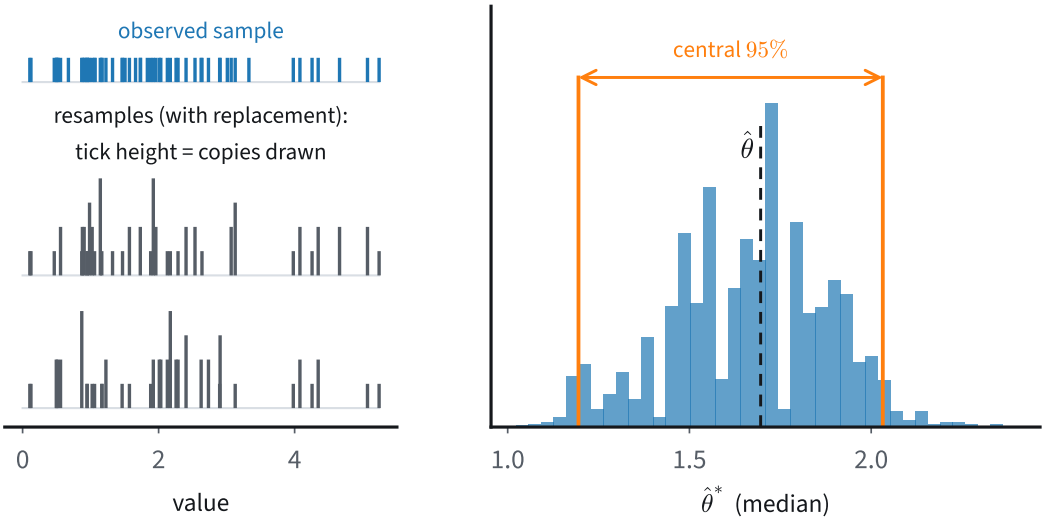


Figure 36.29: 通常のブートストラップ法。1 つの観測標本（左上）から出発し、同じサイズのリサンプルを復元抽出によって多数生成して（左下、目盛りの高さは各観測値が選ばれた回数を表す）、統計量（ここでは中央値）を再計算する。レプリケートのヒストグラム（右）は $\hat{\theta}$（破線）の周りの標本分布を近似しており、その標準偏差が標準誤差の推定値となり、中央の分位点がパーセンタイル区間をなす。ヒストグラムがぎざぎざに見えるのは、どのリサンプルの中央値も少数の観測値のいずれかに一致せざるを得ないためである。この軽微な偏在は、最大値においては完全な破綻へと変わる。

36.5.4.2 ブートストラップ標準誤差

難しい統計量でブートストラップ法を信頼する前に、既知の公式と照らし合わせて確認しておこう。冒頭のテストセットは正解 912 問、不正解 88 問で、理論的な標準誤差は 0.0090 であった。この 1,000 個の二値指示変数を 5,000 回リサンプリングし、ブートストラップ法が同じ数値を再現するか確かめてみよう。

```
correct = np.r_[np.ones(912), np.zeros(88)]          # the opening's test set
idx = rng.integers(0, len(correct), (5000, len(correct)))  # 5,000 resamples
boot = correct[idx].mean(axis=1)                     # accuracy of each resample
print(f'bootstrap se {boot.std(ddof=1):.4f}, '
      f'formula {np.sqrt(0.912 * 0.088 / 1000):.4f}')
```

```
bootstrap se 0.0089, formula 0.0090
```

2 つの推定値は数パーセント以内で一致する。では次に、実用的な閉じた形の標準誤差を持たない統計量に目を向けよう。平均 1 の指数分布から抽出した 200 個の観測値の中央値である。この分布は右に歪んでおり、真の中央値 $\log 2 \approx 0.69$ は平均よりも小さい位置にある。中央値の標準

誤差にも漸近公式 $1/(2f(m)\sqrt{n})$（ただし $f(m)$ は中央値 m における密度）は存在するものの（演習問題 6）、現実の問題では $f(m)$ がわからない。今回の合成実験ではこの密度が**既知**であり、$f(m) = e^{-\log 2} = \tfrac{1}{2}$ から理論的な基準値 $1/\sqrt{200} \approx 0.071$ が得られるため、これを使ってブートストラップ法の精度を評価できる。

```
data = rng.exponential(1.0, 200)    # its median's se formula needs the unknown density
theta_hat = np.median(data)
idx = rng.integers(0, len(data), (10000, len(data)))  # resample with replacement
boot = np.median(data[idx], axis=1)                   # one median per resample
lo, hi = np.percentile(boot, [2.5, 97.5])
print(f'median {theta_hat:.3f}, bootstrap se {boot.std(ddof=1):.3f}, '
      f'95% percentile interval ({lo:.3f}, {hi:.3f})')
```

```
median 0.641, bootstrap se 0.072, 95% percentile interval (0.513, 0.808)
```

密度を一切知らされていないにもかかわらず、ブートストラップ法は通常 0.07 付近の値を返す。異なるデータセットにわたって標本中央値は $\log 2$ の周りで変動し（通常 0.15 以内）、ブートストラップ標準誤差自体もおよそ 0.05 から 0.11 の間でばらつく。なぜこれほど大きく変動するのだろうか。それは、どのリサンプルの中央値も標本の中央付近にあるごく少数の元データ点のいずれかに一致せざるを得ず、ブートストラップ分布が離散的に偏り、それらの点同士のランダムな間隔に敏感になるためである。その結果、パーセンタイル区間はやや粗くなり、ときには $\log 2$ を辛うじて含む程度になることもある。それでもなお、中央値における背後の密度が真に正であることから、中央値はブートストラップ法が妥当であり続けるだけの滑らかさ（smoothness）を備えている。

36.5.4.3 ブートストラップ法の失敗例

ブートストラップ法がいつ破綻するかを知っておくことも同じくらい重要である。その中核にある 2 つの仮定、すなわち滑らかさと独立性（independence）は、いずれも実務において崩れ得る。

第 1 に、統計量が極端なデータ点に鋭敏に依存しすぎるとき、ブートストラップ法は破綻する。古典的な反例は標本最大値である。$x_1, \dots, x_n$ が $[0, \theta]$ 上の一様分布に独立同分布に従うとし、推定量を $\hat{\theta} = \max_i x_i$ とする。どのブートストラップリサンプルも、最大値がまったく同じ $\hat{\theta}$ になる——ただし n 回の抽出すべてがたまたま最大の観測値 1 点を取り逃がした場合を除いてであり、したがって次式が成り立つ。

$$P(\hat{\theta}^* = \hat{\theta}) = 1 - \left(1 - \frac{1}{n}\right)^n \;\to\; 1 - \frac{1}{e} \approx 0.63.$$

つまり、ブートストラップ分布は確率質量の 63% をたった 1 点に積み上げてしまうが、実際の $\hat{\theta}$ の真の標本分布は連続である。さらに悪いことに、いかなるリサンプルも $\hat{\theta}$ を超えることはできず、しかもその値は真に θ 未満であるため、パーセンタイル区間が θ を含む確率は 0 になってしまう。

第 2 に、互いに独立でない単位をリサンプリングするとブートストラップ法は破綻する。これは現代の自然言語処理やコンピュータビジョンのベンチマークで頻繁に陥る罠である。G 個の文章からなり、1 つの文章あたり m 問の設問がある読解ベンチマークを考えよう。モデルがある文章を誤解すると、その文章に属する複数の設問をまとめて間違えやすくなる。同じ文章の 2 問に対する正誤指示変数が相関（correlation）ρ を持ち、異なる文章間は独立であると仮定すると、全体の正解率の分散は次式となる。

$$\mathrm{Var}(\hat{\theta}) = \frac{\theta(1-\theta)}{Gm}\bigl(1 + (m-1)\rho\bigr),$$

この分散は、$1 + (m-1)\rho$ 倍に膨らんでいる（独立な Gm 問の場合と比較して）。個々の設問をリサンプリングする素朴なブートストラップ法はこの文章内相関を無視してしまうため、標準誤差を $\sqrt{1 + (m-1)\rho}$ 分の 1 に過小評価する。$m = 20$、控えめな相関 $\rho = 0.1$ でさえ、エラーバーを $\sqrt{2.9} \approx 1.7$ 分の 1 に過小評価してしまう (Miller 2024)。対処法は明快であり、個々の設問ではなく文章単位で丸ごとリサンプリングすればよい（演習問題 8）。

最後に、重要な未解決の問いを 2 つ残して本節を締めくくろう。第 1 に、本節では与えられた推定量をどのように評価しエラーバーを付けるかを学んだが、そもそも優れた推定量をどのように**構成**するか、あるいはいかなる推定量でもパラメータをどこまで鋭く特定できるかという限界についてはまだ問うていない。36.6 節では、尤度（likelihood）関数とその曲率（curvature）を通じて両方の問いに答える。第 2 に、本節のあらゆる保証はデータを**見る前に**固定された単一の推定量または仮説検定を前提としていた。検証セット（validation set）で最高スコアを出したという理由で選ばれたモデルはこの前提を満たさない。多数の候補から勝者を選ぶことは、姿を変えた多重検定にほかならないからである。36.10 節では、モデル族全体にわたって保証が同時に成り立つためにどれほど大きな標本が必要かを問う。

36.5.5 まとめ

ランダムなデータから計算されるあらゆる数値はそれ自体が確率変数であり、仮想的な別のデータセット間にわたる標本分布によって、その数値をどこまで信頼できるかが決まる。平均二乗誤差は統計的バイアスの二乗と分散に分解されるため、縮小平均のようにバイアスを持つ推定量が不偏な標本平均を上回ることがある。積分を期待値とみなせば標本平均はモンテカルロ推定となり、その誤差レート $\sigma/\sqrt{n}$ は決定論的なグリッドとは異なり次元が上がっても悪化しない。仮説検定は、観測された差が偶然ではめったに起こらないときに帰無仮説を棄却する。帰無仮説のもとで p 値は一様分布に従うため、$p \le \alpha$ のときに棄却すれば偽陽性率は α に制御される一方、統計的検出力は標準化効果量と $\sqrt{n}$ の積とともに増大し、競合モデルを同じ問題上で対応づけることで劇的に向上する。信頼区間とは、仮説検定が棄却しないパラメータ値の集合にほかならない。最後に、解析的な標準誤差が存在しない場合には、ブートストラップ法がデータのリサンプリングによって標本分布をシミュレーションする。これは独立な観測値の滑らかな統計量に対してはよく機能する一方、最大値のような極値やクラスタ構造を持つテストセットに対しては破綻する。

36.5.6 演習問題

1. [code] **標本平均の縮小**。本文では、縮小平均 $c\,\bar{x}$（ただし $x_1, \dots, x_n$ は $\mathcal{N}(\theta, \sigma^2)$ に独立同分布に従う）が最小の MSE を達成するのは $c^\star = \theta^2/(\theta^2 + \sigma^2/n)$ のときであることを示した。

   1. 本文における標本平均・縮小平均・中央値のシミュレーションを、係数 $c \in [0, 1]$ のグリッドへと拡張せよ（$\theta = 1$、$\sigma = 4$、$n = 30$ とする）。統計的バイアスの二乗、分散、MSE を c に対して `d2l.plot` でプロットし、経験的な最小点を $c^\star$ と比較せよ。
   2. 点 a に向かって（0 ではなく）縮小する推定量 $a + c(\bar{x} - a)$ を考える。その MSE と最適な係数を計算せよ。どのような a を選ぶべきか。また、その答えは本文における 0 という選択について何を意味するか。
   3. $c^\star$ は θ を含んでいるため推定量ではない。$\bar{x}$ を θ に、s^2 を σ^2 に代入して $\hat{c} = \bar{x}^2/(\bar{x}^2 + s^2/n)$ とし、$\hat{c}\,\bar{x}$ の MSE を $\theta \in \{0, 1, 3, 10\}$ においてシミュレーションで測定せよ。どのような場合に $\bar{x}$ を上回り、どのような場合に下回るか。演習問題 9 では、3 次元以上において同様の代入があらゆる θ で標本平均を上回ることを示す。

(c) のヒント：$\theta = 0$ ではプラグイン係数自体にノイズが含まれる。その分布を調べてみるとよい。

2. [code] **多重比較。**ハイパーパラメータのスイープでは多数の設定を 1 つのベースラインと比較するため、一度に多くの検定を実行することになる。

    1. いずれもベースラインより優れていない 20 個のモデルを、それぞれ独立な検定（$\alpha = 0.05$）によってベースラインと比較する。少なくとも 1 つの比較が有意になる確率を計算し、ボンフェローニ閾値 $\alpha/20$ における同じ確率も求めよ。また、検定間に従属性がある場合でも、ボンフェローニ補正によって少なくとも 1 つの偽陽性が生じる確率が α 以下に抑えられることを示せ。
    2. ベンジャミニ・ホッホバーグ法 (Benjamini and Hochberg 1995) は、m 個の p 値を $p_{(1)} \le \cdots \le p_{(m)}$ と並べ替え、最大の k（$p_{(k)} \le k\alpha/m$ を満たすもの）を見つけて、小さい方から k 個の p 値に対応する仮説を棄却する。$m = 200$ 個の設定からなるスイープ（そのうち $m_0 = 180$ 個は効果なし）をシミュレーションせよ。z 統計量を、効果のない設定については $\mathcal{N}(0,1)$ から、残りの $\mathcal{N}(3,1)$ から他の 20 個について抽出し、両側 p 値 `2 * norm.sf(np.abs(z))` を計算せよ。1,000 回の反復にわたり、補正なしの閾値（棄却が 1 つもない場合は偽発見割合を 0 とし、閾値は 0.05 とする）、ボンフェローニ補正、およびベンジャミニ・ホッホバーグ法のそれぞれについて、真の発見数と偽発見数の平均および偽発見割合（棄却された仮説のうち偽である割合）の平均を報告せよ。独立な検定に対して、ベンジャミニ・ホッホバーグ法は偽発見割合の期待値が高々 $\alpha m_0/m$ であることを保証する。シミュレーション結果はこの理論と一致するか。

3. [code] **小さな n における被覆率。**信頼区間 Equation 36.73 はガウス分位点 1.96 を t 分位点の代わりに用いており、標本平均が近似的にガウス分布に従うことを仮定している。

    1. 本文の被覆率シミュレーション（標準正規データセットから 1,000 個の区間を構成する実験）を、$n \in \{2, 5, 10, 30, 100\}$ および試行回数 100,000 回で実行し、区間の被覆率を記録せよ。次に、1.96 を 97.5% 分位点（t 分布、自由度 $n-1$）である 12.71、2.776、2.262、2.045、1.984（各 n に対応）に置き換え、再び被覆率を記録せよ。
    2. $n = 2$ とし、1.96 を 1.0（75% 分位点、自由度 1 の t 分布）に置き換えて、区間の被覆率が 50% になるようにせよ。幅の中央値より短い区間と長い区間に分けて、それぞれの被覆率を計算せよ。この差は Equation 36.71 と矛盾するか。また、短い区間が得られたとき、それを推定精度が高い徴候と解釈すべきか。
    3. データを平均 1 の指数分布から $n = 10$ で抽出し、t 分位点 2.262 はそのまま維持せよ。被覆率を測定し、真の平均より完全に下側に外れた区間と完全に上側に外れた区間を別々に数えよ。この非対称性を分布の形状から説明せよ。

4. [code] **境界付近におけるデルタ法。**@eq-mdl-delta-method より、正解率 $\hat{\theta} = 0.90$（$n = 1{,}000$ 個のテスト例で測定）の対数オッズ $\log \frac{\hat{\theta}}{1-\hat{\theta}}$ の標準誤差は約 0.105 である。

    1. 正しい予測の個数を二項分布（パラメータは 100,000 個のテストセットに対して $n = 1{,}000$、$\theta = 0.9$）から抽出してテストセットをシミュレーションせよ。対数オッズの標準偏差をデルタ法による値と比較せよ。
    2. $\theta = 0.999$ で同様の実験を行え。$\hat{\theta} = 1$ となるテストセットの割合を推定して厳密値と比較し、そのようなテストセットにおける対数オッズがどうなるか述べよ。
    3. 1 次展開の背後にあるどの仮定が $\theta = 0.999$ では破綻するか。誤答数の期待値 $n(1-\theta)$ を両方の θ の値について計算し、n 単独ではなくこの個数がデルタ法やワルド区間を信頼できるかどうかを決める理由を説明せよ。
    4. あるベンチマークに 30 問あり、モデルがそのうち 29 問に正解したとする。ワルド区間 $\hat{\theta} \pm 1.96\sqrt{\hat{\theta}(1-\hat{\theta})/30}$ を計算し、その端点がどこに来るか確認せよ。次に検定を

反転させよ。すなわち、各 θ_0（$[0.5, 1]$ 上の細かいグリッド）について、厳密な両側 p 値（帰無仮説 $H_0 : \theta = \theta_0$）である $\min\{1,\, 2\min\{P_{\theta_0}(K \le 29),\, P_{\theta_0}(K \ge 29)\}\}$（ただし $K \sim \text{Binomial}(30, \theta_0)$）を計算し、棄却されずに残る θ_0（$\alpha = 0.05$）を集めよ。2 つの区間を比較し、さらに $\theta = 0.95$ における 100,000 回のベンチマーク・シミュレーションでの被覆率を比較せよ。

5. [code] **対応のある比較。**●同じテスト例または同じシード上で評価された 2 つのモデルはノイズの一部を共有しており、対応のある検定はその共有ノイズを取り除く。

   1. 本文では 2 つのモデルが 41 問で食い違い、内訳は 25 対 16 であった。H_0 のもとで B のみが正解する問題数は $\text{Binomial}(41, \tfrac{1}{2})$ に従う。厳密な両側 p 値を計算し、36.5.2.6 節の符号反転コードセルによるモンテカルロ値と比較せよ。次に正味の差を 9 に固定したまま不一致の総数を変え、p を 9 対 0、17 対 8、25 対 16、55 対 46 のそれぞれについて計算せよ。判定を左右するのは正味の差と不一致数のどちらか。
   2. 両モデルが共有するシード効果を含めてシードごとの正解率をモデル化する。シード s における正解率を $0.850 + u_s + e_{A,s}$ および $0.858 + u_s + e_{B,s}$ とする（ただし $u_s \sim \mathcal{N}(0, 0.01^2)$ と $e_{A,s}, e_{B,s} \sim \mathcal{N}(0, 0.003^2)$ は互いに独立）。シードごとの差に対する符号反転検定を厳密検定として実装せよ。すなわち、k 個のシードに対してランダム抽出の代わりに全 2^k 通りの符号パターンを列挙せよ。1,000 回の実験（10 シード）にわたり、その統計的検出力を対応のない並べ替え検定 `perm_p` と比較せよ。
   3. 5 シードで同様の実験を行え。厳密な符号反転検定が返し得る最小の両側 p 値はいくらであり、それは $\alpha = 0.05$ において何を意味するか。ランダムな符号パターンを 999 個抽出するモンテカルロ版（`perm_p` がランダムなシャッフルを抽出するのと同様の方法）は、これらの実験の数パーセントで棄却してしまう。それはなぜか。

6. **中央値の標準誤差。**●$x_1, \dots, x_n$ を累積分布関数 F、中央値 m、中央値における密度 $f(m) > 0$ を持つ独立同分布の確率変数とする。標本中央値 $\hat{m}$ が高々 $m + t$ になるのは、観測値の少なくとも半分が高々 $m + t$ であるとき、かつそのときに限る。

   1. 経験累積分布関数 $\hat{F}_n(m+t)$ は平均 $F(m+t)$、分散 $F(m+t)(1-F(m+t))/n$ を持つ。中心極限定理を用いて、$P(\hat{m} \le m + t) \approx \Phi\big(2\sqrt{n}\, f(m)\, t\big)$ が小さな t に対して成り立つことを示し、$\text{se}(\hat{m}) \approx 1/\big(2 f(m) \sqrt{n}\big)$ を結論せよ。また、因子 $1/f(m)$ を傾き g'（Equation 36.74）と関連づけよ。
   2. 平均 1 の指数分布において $n = 200$ としてこの近似値を評価し、本文における中央値のブートストラップを実行した際に出力されるブートストラップ標準誤差と比較せよ。コードセルを何度か再実行すると、出力値はおよそ 0.05 から 0.11 の間で変動する。平均の標準誤差に対するブートストラップ推定値はそれほど変動しないにもかかわらず、中央値の標準誤差に対するブートストラップ推定値がデータセットごとにこれほど大きく変動するのはなぜか。
   3. 比 $\text{Var}(\hat{m})/\text{Var}(\bar{x})$ を、大きな n のもとでガウスデータとラプラスデータのそれぞれについて計算せよ。裾の重いデータに対しては、位置パラメータの推定量としてどちらを選ぶべきか。

   (b) のヒント：リサンプルの中央値は $\hat{m}$ 付近にある少数の観測値のいずれかになり、それらの値の間隔はランダムである。

7. [code] **グリッドとモンテカルロ法の比較。**●36.5.1.7 節では指数を比較した。すなわち、N 回の評価を d 次元で行うとき、2 次精度グリッドの誤差は $N^{-2/d}$ のオーダー、モンテカルロ法の誤差は $N^{-1/2}$ のオーダーとなる。この比較を $f(\mathbf{x}) = \exp(-\|\mathbf{x}\|^2)$（領域は $[0,1]^d$）上で検証せよ。その積分値はフビニの積の法則から $\big(\tfrac{\sqrt{\pi}}{2}\,\text{erf}(1)\big)^d$ である（34.6 節）。

   1. m^d 個の超立方体からなる全グリッド上の中点則と、モンテカルロ平均 Equation 36.67

（$N = m^d$ 個の一様サンプル上）を実装せよ。$d \in \{1, 2, 4, 5, 6\}$ および N を最大約 10^6 まで変化させ、相対誤差を N に対して両対数軸でプロットせよ（モンテカルロ法には 20 回の反復にわたる二乗平均平方根誤差を用いること）。傾きは予測された指数と一致するか。

2. $d = 4$ では両者の指数が並ぶ。そこではどちらの手法がより高精度か。各レートの前に付く定数は何によって決まり、それぞれの定数は d とともにどのように変化するか。
3. 両方の誤差は理論的に予測できる。f の中点和は 1 次元の中点和の積になる。また、モンテカルロ法の二乗平均平方根誤差は $\mathrm{sd}\big(f(\mathbf{U})\big)/\sqrt{N}$（Equation 36.68 より、ただし $\mathbf{U}$ は $[0,1]^d$ 上の一様分布）であり、積の法則によって $\mathbb{E}\big[f(\mathbf{U})^2\big]$ は 1 次元積分へと因数分解される。予測される両方の相対誤差を $d \in \{4, 5, 6\}$、$N \approx 10^5$ において計算し、実測値と比較せよ。どの d で 2 つの手法の精度が逆転するか。また、20 回の反復によってその d における優劣をどれほど確実に判定できるか。

8. [code] **クラスタ構造を持つテストセット。**●評価セットは 1 つの文章に関する複数の設問のように互いに関連する項目のクラスタから構成されることが多く、その場合、独立同分布を仮定したブートストラップ法は標準誤差を過小評価する (Miller 2024)。あるテストセットが G 個のクラスタ（各クラスタに m 項目）からなり、期待正解率は θ、同一クラスタ内の 2 項目の正誤指示変数は相関 ρ を持ち、異なるクラスタに属する項目は互いに独立であるとする。

   1. 共通の分散と対ごとの相関を持つ m 個の変数（共通分散 v、対ごとの相関 ρ）の平均が分散 $v\big(1 + (m-1)\rho\big)/m$ を持つことを示し、テストセット上の正解率 $\hat{\theta}$ の分散が次式で与えられることを結論せよ。

$$\mathrm{Var}(\hat{\theta}) = \frac{\theta(1-\theta)}{Gm}\big(1 + (m-1)\rho\big). \tag{36.75}$$

   2. $G = 100$、$m = 20$ としてシミュレーションを行え。各クラスタの正解率を $\theta_g \sim \mathrm{Beta}(6.3, 2.7)$ から抽出し、θ_g が与えられたもとで各項目を条件付き独立に抽出せよ。この設定では $\theta = 0.7$ および $\rho = 1/(6.3 + 2.7 + 1) = 0.1$ となる。300 個のテストセットにわたり、$\hat{\theta}$ の標準偏差を、項目単位でリサンプリングする（本文の正解率ブートストラップと同様の）独立同分布ブートストラップ標準誤差の平均、およびクラスタ全体をリサンプリングするクラスタ・ブートストラップ標準誤差の平均と比較せよ（それぞれ $R = 1{,}000$ 回のリサンプルを用いること）。独立同分布ブートストラップ法は標準誤差を何分の 1 に過小評価するか。その倍率を (a) の理論予測と比較せよ。

9. [code] **スタインのパラドックス。**●●$\mathbf{x} \sim \mathcal{N}(\theta, \mathbf{I}_d)$ を観測し、推定量をリスク $\mathbb{E}\|\hat{\theta} - \theta\|^2$（二乗を二乗ノルムに置き換えた平均二乗誤差）で評価する。推定量 $\mathbf{x}$ のリスクは d であり、これはすべての θ について一定である。James and Stein (1961) は、$d \geq 3$ のとき推定量

$$\hat{\theta}_{\mathrm{JS}} = \left(1 - \frac{d-2}{\|\mathbf{x}\|^2}\right)\mathbf{x} \tag{36.76}$$

がすべての θ に対してより小さいリスクを持つことを示した。

   1. スタインの補題 (Stein 1981) を証明せよ。すなわち、$z \sim \mathcal{N}(\theta, 1)$ であり、h が微分可能で $\mathbb{E}|h'(z)| < \infty$ を満たすならば、$\mathbb{E}[(z-\theta)h(z)] = \mathbb{E}[h'(z)]$ が成り立つことを部分積分により示せ。
   2. 推定量 $\mathbf{x} + \mathbf{g}(\mathbf{x})$ において $\mathbf{g}$ が弱微分可能（ここでは原点以外で微分可能かつ $\mathbb{E}\|\mathbf{g}(\mathbf{x})\|^2 < \infty$）であるとき、そのリスクが

$$d + \mathbb{E}\|\mathbf{g}(\mathbf{x})\|^2 + 2\sum_{i=1}^{d} \mathbb{E}\left[\frac{\partial g_i}{\partial x_i}(\mathbf{x})\right]. \tag{36.77}$$

となることを示せ。この恒等式を用いて、$\widehat{\theta}_{\mathrm{JS}}$ のリスクが $d - (d-2)^2\,\mathbb{E}\big[1/\|\mathbf{x}\|^2\big]$ であることを示せ。この議論のどこで $d \geq 3$ が必要になるか。

3. $\mathbf{x}$、$\widehat{\theta}_{\mathrm{JS}}$、および正値部分推定量 $\max(0, 1-(d-2)/\|\mathbf{x}\|^2)\mathbf{x}$ のリスクを、$d = 10$ および $\|\theta\| \in \{0, 1, 2, 5, 10, 20\}$ においてシミュレーションし、$\|\theta\|$ に対してプロットせよ。
4. 最適な係数 $c^\star$ を $c\,\mathbf{x}$ について d 次元で書き下せ（本文の $c^\star$ の多次元版）。$\widehat{\theta}_{\mathrm{JS}}$ では未知の量がデータのどの関数で置き換えられており、なぜこの代用量は d が大きくなるにつれて正確になるのか。演習問題 1 (c) と比較せよ。

10. [extended] **実行中の検定の覗き見**。●●A/B テストや訓練中の評価では、データが到着する最中に結果を監視し、有意に見えた時点で打ち切ることがよく行われる (Johari et al. 2017)。$x_1, x_2, \ldots$ を $\mathcal{N}(\mu, 1)$ に独立同分布に従う確率変数、$H_0 : \mu = 0$、$S_n = \sum_{i \leq n} x_i$ とする。z 検定（時点 n）は $|S_n|/\sqrt{n} \geq 1.96$ のときに棄却する。

    1. 4,000 回の実行（長さ $N = 10{,}000$）を H_0 のもとでシミュレーションせよ。等間隔にとった K 回の確認時点のうち少なくとも 1 回で棄却が起きる確率を、$K \in \{1, 2, 5, 10, 100, 1000, 10000\}$ について推定せよ。重複対数の法則によれば、監視を無制限に続けるとこの確率は 1 に収束する。作成した表ではどれほどの速さで増加するか。
    2. 事前分布 $\mathcal{N}(0, \tau^2)$ を μ に置き、混合尤度比を

$$\Lambda_n = \int \prod_{i=1}^{n} \frac{p(x_i \mid \mu)}{p(x_i \mid 0)}\, \mathcal{N}(\mu; 0, \tau^2)\, d\mu. \tag{36.78}$$

    と定義する。$\Lambda_n = (1 + n\tau^2)^{-1/2} \exp\!\big(\tau^2 S_n^2 / (2(1+n\tau^2))\big)$ が成り立ち、かつ H_0 のもとでこれがマルチンゲールであること、すなわち $\mathbb{E}[\Lambda_{n+1} \mid x_1, \ldots, x_n] = \Lambda_n$ を満たすことを示せ。
    3. ヴィルの不等式（ここでは事実として認める）によれば、$\Lambda_0 = 1$ を満たす非負マルチンゲールに対して $P(\sup_n \Lambda_n \geq 1/\alpha) \leq \alpha$ が成り立つ。これを用いて、最初の n（$\Lambda_n \geq 1/\alpha$ を満たす時点）で棄却する検定は、データをどれほど頻繁に覗き見ても第一種の過誤が高々 α に抑えられることを結論せよ。また、$|S_n|$ に対する棄却境界を導出し、n に対するその増大の仕方を固定境界 $1.96\sqrt{n}$ と比較せよ。
    4. この逐次検定の第一種の過誤を $\tau^2 \in \{0.01, 0.1, 1\}$ についてシミュレーションで確認せよ。次に $\mu = 0.2$ とし、固定標本の z 検定（標本サイズ n は Equation 36.69 から統計的検出力 0.8 に対して定める）を基準とする。3 つの τ^2 の値のそれぞれについて、時点 n までの逐次検定の検出力、停止時刻の中央値、および時点 N までの検出力を報告せよ。τ^2 の選択はこれらの数値にどう影響するか。また、どの確認時点でも妥当な検定は何を手放し、代わりに何を得ているか。

36.6 最尤法

コインを 13 回投げて、表が 9 回、裏が 4 回出たとしよう。誰であっても、表が出る確率を $9/13 \approx 0.69$ と見積もるだろう。しかし、直ちに 2 つの自然な疑問が生じる。なぜ他ならぬその値なのか、そしてその値をどれほど信用してよいのか、という疑問である。このコインが実は公平なコインである可能性は残っていないだろうか。

1 つの原理がこの両方の疑問に答えてくれる。観測されたデータが最も起こりやすくなるようなパラメータを選ぶ、という原理である。これが**最尤法**（maximum likelihood）の原理であり、本書におけるほぼすべての訓練損失の根底にある。コインの場合、この原理は $9/13$ を返す。ニューラルネットワークの場合、ターゲットに含まれるノイズについての仮定を、最小化すべき損失へ

と変換する。これまでの章で経験的にうまく機能するという理由から採用してきた二乗誤差や交差エントロピー（cross-entropy）は、いずれもこのようにして導かれる。さらに注目すべきことに、同じ損失から、データがパラメータをどれほど強く絞り込んでいるかも読み取れる。最小点の周りで損失が鋭く湾曲しているほどエラーバーは小さくなり、先ほどのコインについても、13 回の試行で公平なコインの可能性を棄却できるかどうかがその曲率からわかる。

先の 36.5 節では、標本平均や標本比率のように閉形式で与えられる推定量の誤差を評価した。本節では、推定量が最適化問題によって定義され、そのエラーバーは損失そのものから直接得られる。まず、これまでの章で用いてきた標準的な損失が最尤法からどのように導かれるかを示す。続いて本節の核心である、最尤推定量はどれほど正確かという問いに移る。その主結果、すなわち標本が大きくなるにつれて推定誤差はガウス分布に近づき、その分散は損失の曲率によって決まるという事実は、ぜひ押さえておきたい。最後に、3 回連続で表が出た場合を例に、データが少ないときに何が破綻するか、そしてデータを見る前に抱いていた事前の信念がいかにして正則化項へと変わるかを見る。各観測の一部が隠れているモデルについては、続く 36.7 節で扱う。

36.6.1 最尤法の原理

確率モデルは、それが生成しうるあらゆるデータセットに対して確率を割り当てる。ひとたびデータが手に入ったら、この視点を逆転させる。すなわち、観測されたデータを固定し、パラメータの各候補のもとでそのデータがどれほど起こりやすいかを問うのである。

モデルのパラメータを θ、観測されたデータセットを X とする。**尤度**（likelihood）とは、観測データの確率（連続データの場合はその密度）をパラメータの関数とみなした $P(X \mid \theta)$ のことである。**最尤推定値**（maximum likelihood estimate、MLE；推定規則としては**最尤推定量**）とは、この関数を最大化するパラメータの値であり、次式で定義される。

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(X \mid \theta). \tag{36.79}$$

あらゆる候補パラメータの中で、MLE は実際に観測された事象が最も起こりやすかったようなパラメータである。なお、この規則は θ を確率変数ではなく未知の定数として扱う。つまり確率はあくまでデータセットに割り当てられ、パラメータの値に割り当てられることは決してない。この区別は、36.6.4 節で事前分布（prior）を導入する際に重要となる。

36.6.1.1 コインの例

偏りのあるコインは、この原理を最後まで手計算で追うことができる最も単純なモデルである。唯一のパラメータは、表が出る確率 θ である。36.3 節ではこの数値を p と呼んだが、ここでは推定すべき未知数であるため θ と書く。各試行は独立なので確率は積の形になり、表が n_H 回、裏が n_T 回出る特定の系列の確率は次式となる。

$$P(X \mid \theta) = \theta^{n_H}(1-\theta)^{n_T}. \tag{36.80}$$

θ の関数とみなしたとき、この式が尤度である。13 回の試行結果の系列 HHHTHTTHHHHHT（表が $n_H = 9$ 回、裏が $n_T = 4$ 回）に対しては、尤度は $\theta^9(1-\theta)^4$ となる。公平なコインがこの系列に割り当てる確率は $2^{-13} \approx 1.2 \times 10^{-4}$ であるのに対し、$\theta = 9/13$ のコインは 3.3×10^{-4} と、約 2.7 倍の確率を割り当てる。

最大点を見つけるには、まず対数をとる。対数関数は狭義単調増加であるため、最大点の位置を変えずに積を微分しやすい和へと変換してくれる。導関数を 0 とおくと、次が得られる。

$$\frac{d}{d\theta}\Big[9\log\theta + 4\log(1-\theta)\Big] = \frac{9}{\theta} - \frac{4}{1-\theta} = 0 \quad\Longrightarrow\quad 9(1-\theta) = 4\theta \quad\Longrightarrow\quad \hat{\theta} = \frac{9}{13}.$$

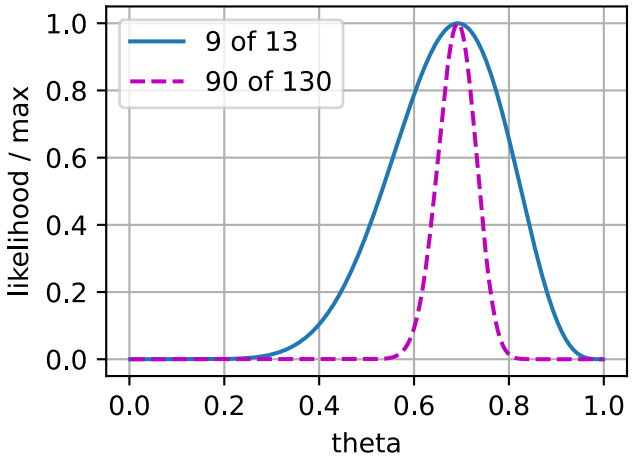
端点 $\theta = 0$ と $\theta = 1$ は、表と裏の両方を含むあらゆる系列に対して確率 0 を割り当てるため、最大点は開区間 $(0, 1)$ の内部になければならず、この停留点が実際に大域的な最大点となる。一般の尤度 Equation 36.80 に対して同じ計算を繰り返すと、次式が得られる。

$$\frac{n_H}{\theta} - \frac{n_T}{1-\theta} = 0 \quad\Longrightarrow\quad \hat{\theta} = \frac{n_H}{n_H + n_T}.$$

コインの偏りの MLE は、直感どおり単に観測された表の相対頻度にほかならない。ただし、ここで導関数を詳しく見ておこう。各試行を $x \in \{0, 1\}$（表を $x = 1$、裏を $x = 0$）と符号化すると、導関数は試行ごとの項 $x/\theta - (1-x)/(1-\theta)$ の和になっている。各項は 1 つの観測の対数確率の勾配であり、後に $\hat{\theta}$ の精度を測る際に主役となる。

ここで、同じ相対頻度が 10 倍の試行回数、すなわち 130 回中 90 回の表から得られたとしよう。点推定値は比率のみに依存するため変わらない。変わるのは、データがその値をどれほど鋭く選び出すかである。以下では、2 つの曲線が同じ縦軸を共有できるよう、それぞれの尤度を最大値で割ってプロットする。

```
theta = np.linspace(0.001, 0.999, 999)

def likelihood(theta, n_H, n_T):  # divided by its maximum
    lik = theta**n_H * (1 - theta)**n_T
    return lik / lik.max()

d2l.plot(theta, [likelihood(theta, 9, 4), likelihood(theta, 90, 40)], 'theta',
         'likelihood / max', legend=['9 of 13', '90 of 130'])
```



どちらの曲線も $9/13 \approx 0.69$ でピークに達するが、2 つ目の曲線は幅が約 3 分の 1 に狭まっている。本節の後半では、このピークの幅を正確なエラーバーへと変換する。

36.6.1.2 負の対数尤度

コインでは閉形式解が得られた。しかし数百万個のパラメータをもつモデルには閉形式解がないため、数値的に最適化する必要があり、浮動小数点演算では生の尤度は使いものにならない。n 個の独立なデータ点があるとき、尤度は n 個の確率の積であり、次式で表される。

$$P(X \mid \theta) = \prod_{i=1}^{n} p(x_i \mid \theta),$$

各因子はいずれも $[0, 1]$ の範囲にある。$1/2$ 付近の因子を 10 億個掛け合わせるとおよそ $(1/2)^{10^9}$ となり、いかなる浮動小数点形式でも勾配を計算するはるか手前でゼロへとアンダーフローしてしまう。

対数をとれば、その積は表現可能な範囲に余裕をもって収まる和へと変わる。

$$\log\big((1/2)^{10^9}\big) = 10^9 \cdot \log(1/2) \approx -6.93 \times 10^8.$$

したがって、尤度の最大化は**対数尤度**（log-likelihood）$\log P(X \mid \theta)$ の最大化と同値である。最適化ルーチンは目的関数の最大化ではなく損失の最小化として実装されるため、符号を反転させて**負の対数尤度**（negative log-likelihood、NLL）を次のように定義する。

$$\ell(\theta) = -\log P(X \mid \theta) = -\sum_{i=1}^{n} \log p(x_i \mid \theta). \tag{36.81}$$

NLL は観測ごとの項の和であり、各データ点に対するモデルの「驚き（サプライズ）」を測る量となっている。コインの場合は $\ell(\theta) = -\big(n_H \log\theta + n_T \log(1-\theta)\big)$ となり、閉形式解が存在しないかのように勾配降下法で最小化することもまったく問題なくできる。以下では、約 900 万回のコイン投げに対して勾配降下法を実行し、結果を $n_H/(n_H+n_T)$ と比較する。

```
n_H = 8675309
n_T = 256245
theta = jnp.float32(0.5)

def nll(theta):
    return -(n_H * jnp.log(theta) + n_T * jnp.log(1 - theta))

grad_nll = jax.grad(nll)
lr = 1e-9
for _ in range(100):
    theta = theta - lr * grad_nll(theta)
theta, n_H / (n_H + n_T)
```

```
(Array(0.9713101, dtype=float32), 0.9713101437890875)
```

勾配降下法は 0.9713 に収束し、表示されたすべての桁で観測頻度と一致している。学習率 10^{-9} は極端に小さく見えるが、損失の和をとっているためにこれが必要になる。約 900 万回の試行にわたって和をとった損失の勾配には n の因子が掛かるため、安定な最大ステップサイズもそれに反比例して小さくなるからである（35.1.2 節）。本書全体を通してミニバッチ損失を例の和ではなく平均として定義しているのは、このためである。

また、対数をとることで勾配もすっきりした加法的な形になる。

$$\nabla_\theta \, \ell(\theta) = -\sum_{i=1}^{n} \frac{\nabla_\theta \, p(x_i \mid \theta)}{p(x_i \mid \theta)},$$

このように例ごとに独立な 1 つの項が得られるのに対し、尤度に直接積の微分法則を適用すれば n 個の積（各々が $n-1$ 個の因子からなる）が生じることになる。その結果、ランダムに抽出したミニバッチから $\nabla \ell$ の不偏推定値が得られ、これによって確率的勾配降下法が可能になる。

36.6.2 負の対数尤度としての損失

コインの例では、確率モデルから直接 NLL を導出した。これに対しこれまでの章では、二乗誤差や交差エントロピーといった訓練損失を直接書き下し、うまく機能するからという理由で使ってきた。実はそうした標準的な損失の大半は姿を変えた負の対数尤度であり、したがって損失関数を選ぶことはデータの確率モデルを選ぶことにほかならない。

36.6.2.1 交差エントロピーとエントロピーの下限

分類器の訓練損失は平均 NLL であり、その訓練曲線はゼロに達するのではなく正の値で頭打ちになることが多い。その値が何を表し、どこまで下がりうるのかを理解するために、平均 NLL を個々の例にわたる和ではなく、異なる結果にわたる和として書き直してみよう。標本の**経験分布**（empirical distribution）を $\hat{p}_{\text{data}}$ とする。すなわち、結果 x が n_x 回（全 n 回の観測中）現れるとき、その観測頻度である確率 $\hat{p}_{\text{data}}(x) = n_x / n$ を割り当てる。先ほどのコインの系列では、$\hat{p}_{\text{data}}(\text{H}) = 9/13$ および $\hat{p}_{\text{data}}(\text{T}) = 4/13$ となる。

ここで、データ点にわたる和を、$x_i = x$ であるすべての観測をまとめ上げることで**異なる結果**にわたる和へと組み替える。コインの例では、9 回の表が 9 個の $\log p_\theta(\text{H})$ にまとまるため、平均には表が重み $9/13$ で、裏が重み $4/13$ で入る。一般に、次式が成り立つ。

$$\begin{aligned} \frac{1}{n}\,\ell(\theta) &= -\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(x_i) = -\sum_{x} \frac{n_x}{n}\, \log p_\theta(x) \\ &= -\sum_{x} \hat{p}_{\text{data}}(x)\, \log p_\theta(x) = \text{CE}(\hat{p}_{\text{data}},\, p_\theta)\,. \end{aligned}$$

右辺は経験分布からモデルへの**交差エントロピー** $\text{CE}(p, q) = -\sum_x p(x) \log q(x)$ である。正の定数 $1/n$ を掛けても最小点は動かないため、離散的な結果に対する最尤法は交差エントロピーの最小化と同値になる。わずか 1 行の項のまとめ直しによって、分類損失がこの名前で呼ばれる理由が説明できる。

この交差エントロピーはどこまで小さくなりうるだろうか。実は、データによって固定される部分とモデルが制御できる部分とにきれいに分解できる。36.4.2 節で見たように、有限集合上の分布 p のエントロピーは $H(p) = -\sum_x p(x)\log p(x)$ すなわち平均の驚きであった。カルバック・ライブラー（KL）ダイバージェンス $D_{\text{KL}}(p \,\|\, q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$ は、真の分布が p であるときに q を用いることで生じる余分な驚きの平均を測る（両者の詳細は 37.1 節で展開する）。本節を通して自然対数を用いるため、これら 3 つの量の単位はいずれも **nat** である。$\sum_x p(x)\log p(x)$ を足して引くと、次式が得られる。

$$\mathrm{CE}(p,q) = -\sum_x p(x)\log q(x) = -\sum_x p(x)\log p(x) + \sum_x p(x)\log\frac{p(x)}{q(x)} = H(p) + D_{\mathrm{KL}}(p\,\|\,q).$$

エントロピーは p のみに依存し、モデル q に依存するのは KL ダイバージェンスだけである点に注目しよう。ギブスの不等式（36.4.2 節）により、KL ダイバージェンスは決して負にならず、ゼロになるのは $q = p$ のときに限られる。

$p = \hat{p}_{\text{data}}$、$q = p_\theta$ とおけば、訓練損失は観測頻度のエントロピーを決して下回ることができず、その下限に達するのはモデル族がその観測頻度を厳密に再現できる場合に限られることがわかる。訓練によって減らせるのは、経験頻度からモデルへの KL ダイバージェンスだけである。先ほどのコインでは、下限は $H(9/13) = -\frac{9}{13}\log\frac{9}{13} - \frac{4}{13}\log\frac{4}{13} = 0.617$ nat となる。公平なコインのモデルは交差エントロピーが $\log 2 = 0.693$ nat であり、0.076 nat の KL ギャップを残す。また $\theta = 0.9$ のモデルは 0.164 nat のギャップを残す。Figure 36.30 に曲線全体を示す。

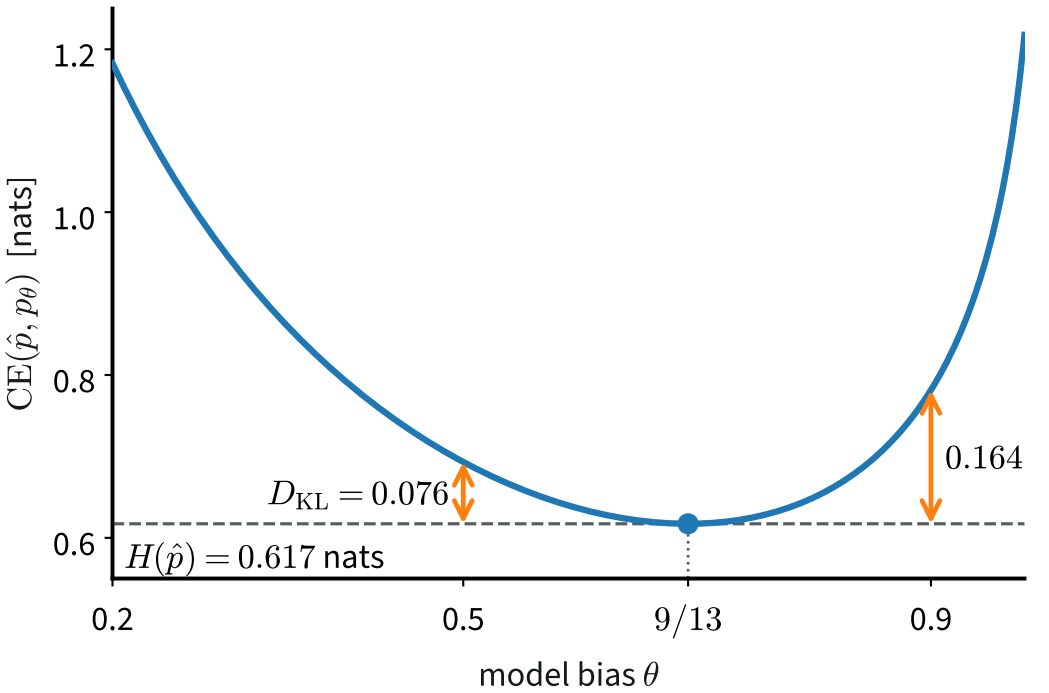


Figure 36.30: コインデータ（表 9 回、裏 4 回）に対する試行あたりの交差エントロピーをモデルの偏り θ の関数として描いた図。損失は観測頻度のエントロピーである 0.617 nat（破線）を決して下回らず、MLE である $9/13$ でその下限に接し、それ以外の任意の θ ではデータの頻度からモデルへの KL ダイバージェンス（オレンジ色）の分だけ下限を上回る。具体的には $\theta = 1/2$ で 0.076 nat、$\theta = 0.9$ で 0.164 nat となる。

同じ恒等式は分類器にも当てはまる。分類器は $p_\theta(y \mid \mathbf{x})$ をモデル化し、その平均 NLL は訓練セット上で $-\log p_\theta(y_i \mid \mathbf{x}_i)$ の平均をとる。入力ごとに例をグループ化すれば、異なる各入力において交差エントロピーの分解が復元され、その入力における経験ラベル頻度が $\hat{p}_{\text{data}}$ の役割を果たす（演習問題 1）。ワンホットラベルとソフトマックス出力の場合、この平均 NLL はまさしく 3.1 節のソフトマックス交差エントロピー損失に一致する。典型的なデータセットでは各訓練入

力は 1 度しか現れないため、各入力における経験ラベル分布はワンホットとなり、ラベルにどれほどノイズが含まれていても訓練損失の下限はゼロになる。実務家が直面する削減不能な下限はホールドアウトデータ上で現れ、そこでは期待損失が入力を与えたもとでのラベルの条件付きエントロピーを下回ることはできない（37.1.3.1 節）。

言語モデルは、ラベル集合が非常に大きい同じ分類器にほかならない。入力はそこまでのテキスト、ラベルは次のトークンであり、訓練ではトークンあたりの平均 NLL を最小化する。この値は通常、その指数をとった**パープレキシティ**（8.3.2 節）として報告される。これは同じ平均の驚きをもたらす等確率なトークンの個数に相当する。言語モデルでは、ホールドアウトでの下限はゼロからほど遠い。たいていの文には自然な続き方が何通りもあるため、次のトークンの条件付きエントロピーは大きく、いかに巨大なモデルであってもホールドアウト損失をその下限より小さくすることはできない。

36.6.2.2 確率密度

連続データの場合、尤度の定義にはもう 1 段階の手続きが必要になる。特定の厳密な実数値が 1 つ観測される確率はゼロであるため、文字どおりにとればどの θ に対しても尤度は 0 となり、パラメータの候補どうしを区別できなくなってしまう。そこで代わりに、各観測が小さな許容幅 ϵ の範囲内で一致することだけを求め、そのうえで ϵ が最適化から消去されることを確かめる。

密度 $p(x \mid \theta)$ に従う独立同分布（i.i.d.）な観測 $x_1, \dots, x_n$ に対し、すべての X_i が幅 ϵ の区間（x_i の周囲）に入る確率は、ϵ の一次までで次のように近似できる。

$$P\big(X_i \in [x_i, x_i + \epsilon]\ \forall i \mid \theta\big) \approx \epsilon^n \prod_{i=1}^{n} p(x_i \mid \theta).$$

負の対数をとると、次式が得られる。

$$-\log P\big(X_i \in [x_i, x_i + \epsilon]\ \forall i \mid \theta\big) \approx -n \log \epsilon - \sum_{i} \log p(x_i \mid \theta).$$

許容幅 ϵ は加法定数 $-n \log \epsilon$ を通じてのみ現れ、この項は θ に依存しない。求める精度が 4 桁であっても 400 桁であってもその定数がずれるだけで最小点は動かないため、これを落として離散の場合とまったく同様に $-\sum_i \log p(x_i \mid \theta)$ を最小化する。ただし、密度の値そのものは確率ではない点に留意したい。確率を近似するのは積 $\epsilon\, p(x_i \mid \theta)$ だけであるため、p は x の逆数の単位をもち、1 を超えることもありうる。

母集団における交差エントロピー恒等式は、連続データにもそのまま引き継がれる。標本上の負の対数密度の平均が引き続き目的関数であり、真のデータ生成密度 $p_\star$ のもとでのその期待値は次式となる。

$$\mathbb{E}_{x \sim p_\star}\big[-\log p(x \mid \theta)\big] = h(p_\star) + D_{\mathrm{KL}}(p_\star \,\|\, p_\theta),$$
$$h(p_\star) = -\int p_\star(x) \log p_\star(x)\, dx,$$

これは和を積分に置き換え、同じ項を足して引く議論から導かれる。**微分エントロピー**（differential entropy）$h(p_\star)$ は θ に依存しないため、母集団レベルでの最尤法は連続の場合もやはり KL ダイバージェンスを最小化する。なお、n 個の異なる実数値の経験分布は密度をもたないため、離散の場合の有限標本における恒等式に対応するものは連続データには存在しない。

36.6.2.3 ガウスノイズと二乗誤差

ターゲットに含まれるノイズのモデルを指定すれば、標準的な回帰損失も同じ原理から直ちに現れる。2.1.3 節では線形回帰についてこの関係を導いたが、その導出はモデルが線形であることに一切依存していない。各ターゲットがモデルの予測に**固定された**分散をもつガウスノイズを加えたものに等しい、すなわち $y_i \sim \mathcal{N}(\hat{y}_i, \sigma^2)$（ただし $\hat{y}_i = f_\theta(\mathbf{x}_i)$）であるとしよう。ガウス密度 $(2\pi\sigma^2)^{-1/2}\exp\!\big(-(y-\hat{y})^2/(2\sigma^2)\big)$ の負の対数を n 個の独立な例にわたって足し合わせると、次式が得られる。

$$-\log \prod_{i=1}^{n} \mathcal{N}(y_i;\, \hat{y}_i,\, \sigma^2) = \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \frac{n}{2}\log(2\pi\sigma^2). \tag{36.82}$$

分散 σ^2 は固定されているため、第 2 項は θ に関して定数であり、先頭の係数 $1/(2\sigma^2)$ も正の定数である。したがって、この NLL の最小化は $\sum_i (y_i - \hat{y}_i)^2$ の最小化と同値である。すなわち二乗誤差による回帰は、残差がガウス分布に従うという仮定のもとでの最尤法にほかならない。同じ手順により、以下に示す 4 つをはじめとする多くのなじみ深い損失が、条件付き分布 $p(y \mid \mathbf{x})$ の NLL と結びつく。各行は分布の負の対数をとり、θ に依存しない定数を落とすことで導かれる。

ノイズモデル $p(y \mid \mathbf{x})$	負の対数尤度 $-\log p(y \mid \mathbf{x})$	深層学習の損失
ガウス分布、固定分散 σ^2	$\frac{1}{2\sigma^2}(y-\hat{y})^2 + \text{const}$	平均二乗誤差（MSE）
ベルヌーイ分布	$-y\log\hat{p} - (1-y)\log(1-\hat{p})$	二値交差エントロピー（BCE）
カテゴリカル分布（ワンホット y）	$-\sum_k y_k \log \hat{p}_k$	ソフトマックス交差エントロピー
ラプラス分布、固定スケール s	$\frac{1}{s}\lvert y-\hat{y}\rvert + \text{const}$	平均絶対誤差（MAE）

これらの損失のいずれにおいても、損失関数を選ぶことはノイズモデルを選ぶことである。ただし、有用な損失のすべてがこの形で導かれるわけではない。サポートベクターマシンのヒンジ損失にはモデルの出力に依存しない正規化定数が存在しないため、加法定数の違いを除いてもいかなる条件付き分布の NLL にもならない（演習問題 3）。

ノイズモデルの選択は、データに外れ値が 1 つ混入した瞬間に決定的な違いを生む。5 つのセンサ測定値 $\{1, 2, 3, 4, 100\}$ に単一の定数 c を当てはめる場面を考えよう。ここで最後の測定値は故障したセンサから得られたものである。ガウスノイズのもとでは $\frac{1}{2}\sum_i (y_i - c)^2$ を最小化し、その導関数 $-\sum_i (y_i - c)$ は標本平均 $c = 22$ でゼロになる。一方、ラプラスノイズのもとでは $\sum_i \lvert y_i - c\rvert$ を最小化する。これは V 字型の曲線の和であり、c における傾きは c より小さい測定値の個数から大きい測定値の個数を引いた値に等しい。この傾きは中央値の左側では負、右側では正となるため、最小点は中央値 $c = 3$ に位置する。もし 5 番目の測定値が 5 であったなら、どちらのノイズモデルも 3 を返していたはずである。Figure 36.31 が示すとおり、異常な測定値はガウスモデルによる当てはめを 22 まで大きく引きずる一方、ラプラスモデルによる当てはめにはまったく影響を与えない。

36.6.3 最尤推定量の精度

ここで冒頭の問いに戻ろう。$9/13$ という値にどれほど確信をもってよいのだろうか。標本比率に対しては、36.5 節で標準誤差 $\sqrt{\hat{\theta}(1-\hat{\theta})/n}$ を与えた。これは標本抽出を繰り返したときの $\hat{\theta}$ の標準偏差を推定する量である。しかし一般のモデルでは MLE は単純な標本平均ではないため、損失そのものからエラーバーを直接読み取る一般的な方法が必要になる。その役割を担うのが、コインの例で登場した例ごとの勾配と、NLL の曲率という 2 つの要素である。

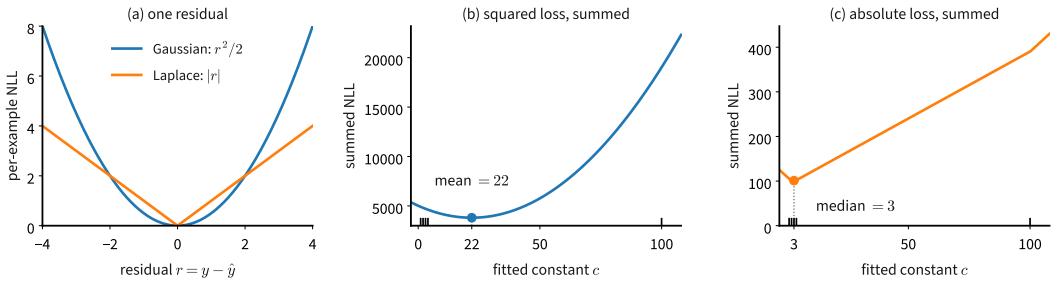


Figure 36.31: 測定値 $1, 2, 3, 4, 100$ に対する単一の定数 c の当てはめ。(a) 残差の関数としての例ごとの負の対数尤度。ガウスノイズでは二次関数、ラプラスノイズでは V 字型となる。(b) 二乗損失の和は放物線をなし、その最小点である平均 22 は外れ値の方向へ大きく引き寄せられる。(c) 絶対誤差損失の和は各測定値（黒の目盛り）で折れ曲がりをもち、中央値 3 で最小値に達する。各点はどれだけ遠く離れていても、傾き 1 単位分の力でしか c を引っ張らない。

36.6.3.1 スコア関数とフィッシャー情報量

1 つの観測に対する**スコア関数**（score、値としては単に**スコア**とも呼ぶ）とは、対数確率のパラメータに関する勾配である。

$$s(x;\theta) = \nabla_\theta \log p(x \mid \theta).$$

これはコインにおける試行ごとの勾配を任意の確率モデルへと一般化したものであり、方程式 $\sum_i s(x_i;\hat{\theta}) = \mathbf{0}$ が MLE を定める一階最適性条件となる。

スコアの際立った性質として、真のパラメータにおいてその期待値がゼロになることが挙げられる。理由を確かめるため、正規化条件 $\int p(x \mid \theta)\,dx = 1$ を積分記号の下で微分しよう。

$$\begin{aligned}\mathbf{0} = \nabla_\theta \int p(x \mid \theta)\,dx &= \int \nabla_\theta\, p(x \mid \theta)\,dx \\ &= \int s(x;\theta)\, p(x \mid \theta)\,dx = \mathbb{E}_\theta\big[\, s(x;\theta)\,\big],\end{aligned}$$

ここで 3 つ目の等号には対数微分の恒等式 $\nabla_\theta p = p\,\nabla_\theta \log p = p\,s$ を用いた。言葉で言えば、モデルから抽出されたデータにわたって平均をとると、対数尤度の勾配はそのデータを生成したパラメータにおいてゼロになる。スコアの平均がゼロであるため、その共分散行列は二次モーメントに等しい。この共分散こそが**フィッシャー情報量**（Fisher information）である (Fisher 1925)。

$$\begin{aligned}I(\theta) &= \mathrm{Var}_\theta\big[\, s(x;\theta)\,\big] = \mathbb{E}_\theta\big[\, s(x;\theta)\, s(x;\theta)^\top\,\big] \\ &= -\,\mathbb{E}_\theta\big[\nabla_\theta^2 \log p(x \mid \theta)\big].\end{aligned} \tag{36.83}$$

中央の等号は平均がゼロである性質を反映している。最後の等号は**情報量恒等式**（information identity）であり、スコアの分散が NLL の期待曲率に等しいことを表す。これは $\mathbb{E}_\theta[s] = \mathbf{0}$ を再び積分記号の内部で微分することで導かれる。積の微分法則により、微分は 2 つの項に分かれる。

$$\begin{aligned}\mathbf{0} &= \nabla_\theta \int s(x;\theta)\, p(x \mid \theta)\, dx \\ &= \int \nabla_\theta^2 \log p(x \mid \theta)\; p(x \mid \theta)\, dx + \int s(x;\theta)\, \big(\nabla_\theta\, p(x \mid \theta)\big)^{\top} dx.\end{aligned}$$

第 2 の積分に $\nabla_\theta\, p = p\, s$ を代入すると $\mathbb{E}_\theta[\, s\, s^{\top}]$ となり、次式が得られる。

$$\mathbb{E}_\theta\big[\nabla_\theta^2 \log p(x \mid \theta)\big] + \mathbb{E}_\theta[\, s\, s^{\top}\,] = \mathbf{0},$$

これを移項すれば、直ちに Equation 36.83 の最後の等号が得られる。直感的には、鋭く湾曲した NLL はパラメータを強く絞り込むのに対し、平坦な NLL はパラメータを曖昧なままにする。フィッシャー情報量は、モデルから抽出されたデータにわたって平均した、観測 1 個あたりの NLL の曲率を測っている。

上の 2 つの導出では、断りなく微分を積分の内側へ入れた。以降の結果はいくつかの**正則条件**（regularity conditions）に依拠しており、ここで一度にまとめておく。

- 密度が θ に関して 2 回連続微分可能である。
- 密度の台（support）、すなわち $p(x \mid \theta) > 0$ となる x の集合が θ に依存しない。
- 微分と積分の順序交換が可能である（34.7 節の手法）。これが上の各導出における 2 つ目の等号を正当化しており、台が動く場合にはこの段階が破綻する。
- 真のパラメータがパラメータ空間の内部に存在する。
- フィッシャー情報量が有限かつ非特異（正則）である。

フィッシャー情報量が特異になる場合や、台がパラメータとともに動く場合に具体的に何が破綻するかは、演習問題 6 および 7 で確認する。

複雑さの異なる 3 つのモデルについて順に $I(\theta)$ を計算してみよう。第 1 にコインでは、1 回の試行 $x \in \{0, 1\}$ の対数尤度は $\log p(x \mid \theta) = x \log\theta + (1 - x)\log(1 - \theta)$ であるため、そのスコアと二階導関数は次式となる。

$$s(x;\theta) = \frac{x}{\theta} - \frac{1 - x}{1 - \theta}, \qquad \frac{d}{d\theta}\, s(x;\theta) = -\frac{x}{\theta^2} - \frac{1 - x}{(1 - \theta)^2}.$$

$-\mathbb{E}[\,\cdot\,]$ をとり $\mathbb{E}[x] = \theta$ を用いれば、1 試行あたりのフィッシャー情報量と、その逆数を n で割った値が得られる。

$$I(\theta) = \frac{1}{\theta} + \frac{1}{1 - \theta} = \frac{1}{\theta(1 - \theta)}, \qquad \frac{I(\theta)^{-1}}{n} = \frac{\theta(1 - \theta)}{n}. \tag{36.84}$$

1 試行あたりの情報量が $\theta = 1/2$ で最小になることに注目しよう。各試行がコインの偏りについて運ぶ情報量が最も少ないため、公平なコインが最も特定しにくいのである。

次に、2 パラメータのモデルで行列形式を見てみよう。平均 μ、標準偏差 σ のガウス分布では、1 つの観測の NLL は定数項を除いて $\log\sigma + (x - \mu)^2/(2\sigma^2)$ である。μ に関する二階導関数は $1/\sigma^2$ である一方、σ に関する二階導関数は $-1/\sigma^2 + 3(x - \mu)^2/\sigma^4$ であり、その期待値は $2/\sigma^2$ となる。交差偏導関数 $2(x - \mu)/\sigma^3$ の期待値はゼロであるため非対角成分は消え、$I(\mu, \sigma) = \mathrm{diag}(1/\sigma^2,\, 2/\sigma^2)$ となる。NLL は σ 方向には μ 方向の 2 倍鋭く湾曲しており、これはデータが平均よりもスケールの方を強く絞り込むことを意味する。Figure 36.32 に 30 個のサンプルに対するこのお椀型の曲面を示す。

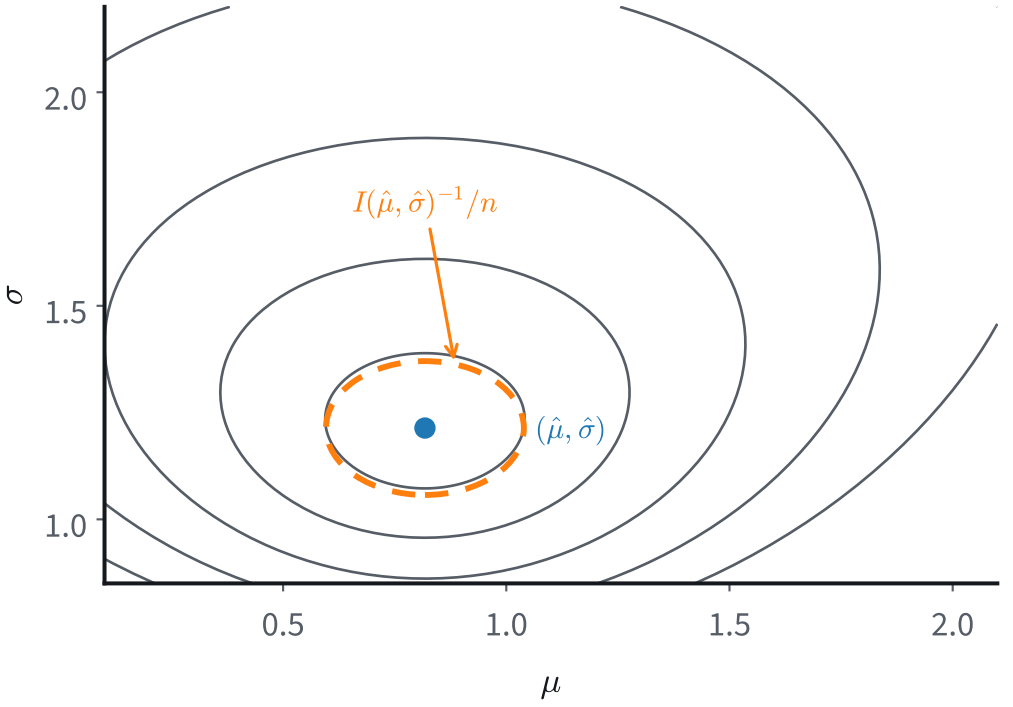


Figure 36.32: ガウス分布のパラメータ (μ, σ) に対する負の対数尤度（$n = 30$ 個のサンプルを $\mathcal{N}(1, 1.5^2)$ から抽出、点は MLE を示す）。破線の楕円は $I(\hat{\mu}, \hat{\sigma})^{-1}/n$ を表しており、NLL が最小値より半分だけ高い最も内側の等高線をよく捉え、μ 方向には σ 方向よりも広くなっている。

コインでもガウス分布でも MLE は標本平均の形になるため、フィッシャー情報量を使わなくてもそのばらつきを計算できた。理論の真価が表れるのは、MLE が平均にならないモデルである。**コーシー位置モデル**（Cauchy location model）は次の密度をもつ。

$$p(x \mid \theta) = \frac{1}{\pi\bigl(1 + (x-\theta)^2\bigr)},$$

これは θ を中心とする対称な釣鐘型の分布であるが、36.1 節で示したとおり裾が極めて重いため平均が存在しない。$u = x - \theta$ とおくと、スコアは $s = 2u/(1+u^2)$、その導関数は $\partial_\theta s = -2(1-u^2)/(1+u^2)^2$ となる。三角関数による置換 $u = \tan\varphi$ を使えば、フィッシャー情報量は一目で求まる。$d\varphi = du/(1+u^2)$ より、標準コーシー変数の角度 φ は $(-\pi/2, \pi/2)$ 上の一様分布に従い、次式が成り立つ。

$$s = \frac{2\tan\varphi}{1+\tan^2\varphi} = \sin 2\varphi, \qquad \text{so} \qquad I = \mathbb{E}\bigl[\sin^2 2\varphi\bigr] = \frac{1}{2}.$$

したがって、コーシー分布の各観測はその中心について半単位分の情報量をもっており、これは $\sigma^2 = 2$ のガウス分布の観測と同じである。また、スコアの絶対値は決して 1 を超えず、遠く離れた点に対してはゼロへと減衰することにも注目しよう。極端な外れ値が MLE をほとんど動かさないのはこのためである。

閉形式で解く工夫が使えない場合でも、自動微分によって I の両方の表現を直接計算できる。下のセルでは、$\theta^\star = 0$ でコーシー分布から 10,000 個の観測を抽出し、スコアの平均、スコアの分散、および負の平均曲率を表示する。一様な角度表現を用いて計算すると、これら 3 つの推定値のモンテカルロ標準誤差はそれぞれ約 0.007、0.004、0.008 となる。

```
def log_p(theta, x):  # log density of the Cauchy location model
    return -jnp.log1p((x - theta)**2) - jnp.log(jnp.pi)

x = np.random.default_rng().standard_cauchy(10_000)  # theta* = 0
score = jax.vmap(jax.grad(log_p), in_axes=(None, 0))
curv = jax.vmap(jax.grad(jax.grad(log_p)), in_axes=(None, 0))
s, h = score(0.0, x), curv(0.0, x)
print(f'mean score {s.mean():.3f}, variance of score {s.var():.3f}, '
      f'-mean curvature {-h.mean():.3f}')
```

```
mean score 0.003, variance of score 0.495, -mean curvature 0.499
```

`jax.grad` が 1 つの観測の対数密度を微分し、`vmap` がそれを全 10,000 個の観測にわたって適用する。`grad` を 2 回適用すれば曲率が得られる。

3 つの標本推定値はいずれも、理論値である 0、0.5、0.5 から標準誤差の 2〜3 倍以内に収まっている。

36.6.3.2 コインのエラーバー

フィッシャー情報量を使えば、本節の冒頭で求めたかった $9/13$ のエラーバーが手に入る。次項では、大きな n に対して MLE の分散が近似的に $1/\bigl(nI(\theta)\bigr)$ となることを示す。ひとまずその結果を認めることにすれば、直ちに適用できる。コインでは Equation 36.84 より $1/(nI) = \theta(1-\theta)/n$ であり、$\widehat{\theta}$ を未知の θ に代入すると推定標準誤差は次式となる。

$$\widehat{\mathrm{se}} = \sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}} = \sqrt{\frac{(9/13)(4/13)}{13}} = 0.128.$$

したがって $\hat{\theta} = 0.69 \pm 0.13$ となる。近似的な 95% 信頼区間 $\hat{\theta} \pm 1.96\,\widehat{\mathrm{se}} = [0.44, 0.94]$ は余裕をもって $1/2$ を含んでおり、13 回の試行ではこのコインを公平なコインと区別することはできない。95% の信頼度で θ を ± 0.05 の幅まで絞り込むには、およそ $1.96^2\,\hat{\theta}(1-\hat{\theta})/0.05^2 = 1.96^2 \cdot 0.213/0.05^2 \approx 330$ 回の試行が必要になる。

重要なのは、損失そのものからまったく同じ数値が直接出てくることである。コインの NLL の二階導関数を $\hat{\theta}$ で評価すると、次式となる。

$$\ell''(\hat{\theta}) = \frac{n_H}{\hat{\theta}^2} + \frac{n_T}{(1-\hat{\theta})^2} = \frac{9}{(9/13)^2} + \frac{4}{(4/13)^2} = 61.0,$$

そして $1/\sqrt{61.0} = 0.128$ である。コインの場合、この一致は厳密である。なぜなら $n_H = n\hat{\theta}$ かつ $n_T = n(1-\hat{\theta})$ より、観測された曲率は $n/\big(\hat{\theta}(1-\hat{\theta})\big) = nI(\hat{\theta})$ にちょうど等しいからである。正則条件を満たす任意のモデルにおいて、大きな n では同じ関係が近似的に成り立つ。モデルを当てはめた後、最小点における NLL の曲率（ヘッセ行列）を計算し、その平方根の逆数をとればエラーバーになる。パラメータがベクトルの場合、個々のパラメータのエラーバーは逆ヘッセ行列の対角成分の平方根である。深層ニューラルネットワークはこれらの正則条件に違反しており、この手順のうち何が有効なまま残るかについては 36.6.3.5 節で説明する。また、本節の冒頭で見た尤度のプロットも曲率から説明できる。データ量が 10 倍になると曲率も 10 倍になり、ピークの幅は $\sqrt{10} \approx 3.2$ 倍細く絞られるのである。

36.6.3.3 一致性と漸近正規性

なぜ MLE の分散は近似的に $1/(nI)$ になるのだろうか。これに答えるには、まずそもそも MLE が正しい値へ向かうことの保証が必要であり、次にその目標値の周りでどのように揺らぐかを記述する必要がある。これまでの節で学んだ 3 つの道具がその役割を果たす。36.5 節からは一致性（consistency、$\hat{\theta} \xrightarrow{P} \theta^\star$）と弱大数の法則を用いる。また 36.3 節からは中心極限定理を用いる。これは、平均ゼロ・分散 v の独立同分布な n 個の項の和を $\sqrt{n}$ で割ったものが、$\mathcal{N}(0, v)$ に分布収束（$\xrightarrow{d}$）するという定理である。

MLE が真のパラメータに近づく理由は、36.6.2.1 節の交差エントロピー恒等式から説明できる。データが真のパラメータ $\theta^\star$ をもつモデルから抽出されているとしよう。平均 NLL は独立同分布な項の平均であるため、大数の法則により母集団 NLL へと収束する。

$$\mathbb{E}_{\theta^\star}\big[-\log p(x \mid \theta)\big] = H(p_{\theta^\star}) + D_{\mathrm{KL}}(p_{\theta^\star} \,\|\, p_\theta),$$

ここで連続密度の場合は、微分エントロピー h を H の代わりに用いる（36.6.2.2 節）。ギブスの不等式（36.4.2 節）により、この目的関数は KL ダイバージェンスがゼロになるところ、すなわち $\theta = \theta^\star$ で最小値を達成する。モデルが**識別可能**（identifiable）であるとき、つまり異なるパラメータベクトルが常に異なる分布を定めるとき、この最小点は一意である。ニューラルネットワークは識別可能ではない。ある層の隠れユニットを、そこに入出力される重みごと並べ替えれば、計算される関数を一切変えずにパラメータベクトルだけが変わるからである。そのようなモデルにおいて一致性とは、単一のパラメータベクトルについての主張ではなく、当てはめられた分布、あるいはその分布を生み出すパラメータの同値類についての主張となる。

一致性の保証を証明なしで述べておこう (Wasserman 2013)。次の 4 つの条件が成り立つとき、MLE は一致推定量となる。すなわち、データが $p_{\theta^\star}$ からの独立同分布であること、モデルが識別可能であること、母集団の最小点が十分に分離されており θ が $\theta^\star$ から離れている限り最適値にいくらでも近づくことはないこと（コンパクトなパラメータ集合上で目的関数が連続であれば一意性で十分である）、そして平均 NLL が θ に関して母集団 NLL に一様収束することである。一様性は見落とされがちな微妙な要件である。大数の法則が保証するのは固定された各 θ での収束であり、各点収束だけでは最小点の収束までは導けない。もちろんコインについては定理に頼るまでもなく、大数の法則から直接 $\widehat{\theta} = n_H/n \to \theta^\star$ が従う。

一致性から $\widehat{\theta}$ の行き先はわかる。エラーバーを得るには、$\theta^\star$ の周りでの揺らぎの大きさと形状が必要である。スカラーパラメータ θ の場合、導出は 3 つの短いステップで済み、唯一の巧妙な着想は NLL そのものではなく一階最適性条件をテイラー展開することにある。合計スコアを $S_n(\theta) = \sum_i s(x_i;\theta)$ と書こう。

I. 定義より、MLE は $S_n(\widehat{\theta}) = 0$ を満たす。$S_n(\widehat{\theta})$ を $\theta^\star$ の周りで一次まで展開すると、次式が得られる。

$$0 = S_n(\widehat{\theta}) \approx S_n(\theta^\star) + S_n'(\theta^\star)\,(\widehat{\theta} - \theta^\star).$$

II. 推定誤差について解き $\sqrt{n}$ を掛けると、2 つの平均の比として表せる。

$$\sqrt{n}\,(\widehat{\theta} - \theta^\star) \approx \frac{S_n(\theta^\star)/\sqrt{n}}{-S_n'(\theta^\star)/n}.$$

III. 分子は、平均ゼロ・分散 $I(\theta^\star)$ をもつ独立同分布な n 個のスコアの和を $\sqrt{n}$ で割ったものであり、中心極限定理により $\mathcal{N}\big(0, I(\theta^\star)\big)$ に分布収束する。分母は独立同分布な n 個の項 $-\partial_\theta s(x_i;\theta^\star)$ の平均であり、情報量恒等式からその期待値は $I(\theta^\star)$ に等しいため、大数の法則により $I(\theta^\star)$ に収束する。分散 I のガウス変数を定数 I で割れば、分散 $I/I^2 = 1/I$ のガウス分布が得られる。

フィッシャー情報量の 2 つの表現がここで見事に出会うことに注目しよう。分子ではスコアの分散が使われ、分母では期待曲率が使われている。情報量恒等式によって両者が同じ値 I に一致するからこそ、最終的な公式はこれほど簡潔になるのである。ベクトルの場合も行列を用いてまったく同じ代数計算をたどることで、共分散 $I^{-1} I I^{-1} = I^{-1}$ が得られる。厳密な証明ではテイラー展開の剰余項も上から評価する必要があり、下の命題における可積分な曲率の上界がその役割を果たす。

命題（漸近正規性）正則条件、上記の一致性の条件、および上界 $\|\nabla^2_\theta \log p(x \mid \theta)\| \le M(x)$（ただし $\mathbb{E}_{\theta^\star}[M(x)] < \infty$ がすべての θ（$\theta^\star$ の近傍）で成り立つ）のもとで、次が成立する。

$$\sqrt{n}\,(\widehat{\theta} - \theta^\star) \;\xrightarrow{\;d\;}\; \mathcal{N}\!\big(\mathbf{0},\; I(\theta^\star)^{-1}\big) \tag{36.85}$$

出典は (Wasserman 2013) である。上の 3 ステップ以上の証明には立ち入らないが、これが本節の中心結果である。エラーバー、MLE の有効性、そして理論の限界はすべてこの結果の上に成り立っている。平たく言えば、MLE の推定誤差は近似的にガウス分布に従い、$1/\sqrt{n}$ の速さで縮小し、その共分散は逆フィッシャー情報量で与えられる。有限の n では近似 $\mathrm{Cov}(\widehat{\theta}) \approx I(\theta^\star)^{-1}/n$ を用い、これがコインに適用したエラーバーであった。とりわけ、エラーバーを半分にするには 4 倍のデータが必要になる。モデルが誤特定されている場合には、I の分散表現と曲率表現がもはや一致せず、共分散は 36.6.3.5 節の「サンドイッチ」公式へと置き換わる。

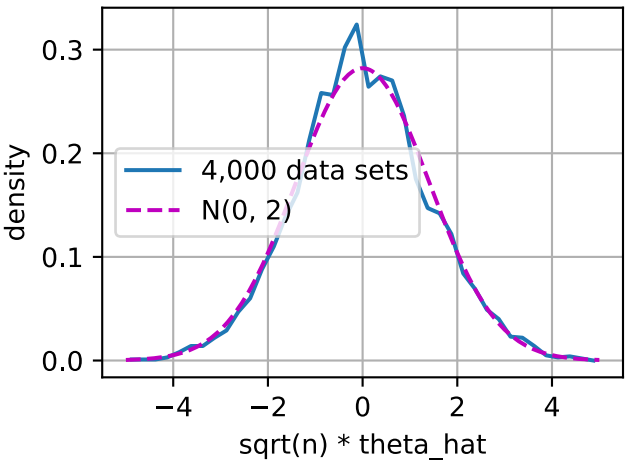
非自明な設定で漸近正規性を検証するには、MLE が標本平均にならないモデルが必要であり、コーシー位置モデルがまさにうってつけである。その尤度方程式 $\sum_i 2u_i/(1+u_i^2)=0$ には閉形式解がないため、観測された曲率を期待曲率 $nI = n/2$ に置き換えたニュートン法である**フィッシャースコアリング**（Fisher scoring）を用いて数値的に解く。

$$\theta \leftarrow \theta + \frac{S_n(\theta)}{nI} = \theta + \frac{2}{n} S_n(\theta).$$

観測曲率を nI に置き換えるのは単なる計算上の便宜にとどまらない。期待曲率は常に正であるのに対し、コーシー分布の NLL の観測曲率は最小点から離れると負になりうるため、素のニュートンステップでは山を登る方向へ進んでしまうからである。ここではすでに解に近い標本中央値を各実行の初期値とし、$\theta^\star = 0$ で生成した $n = 100$ 個の観測からなる 4,000 個のデータセットに対し、それぞれ 10 回のスコアリングステップで当てはめを行う。$n = 100$ の時点ですでに漸近近似が正確であれば、$\sqrt{n}\,\hat\theta$ のヒストグラムは $\mathcal{N}(0, 1/I) = \mathcal{N}(0, 2)$ に一致するはずである。

```
reps, n = 4000, 100
x = np.random.default_rng().standard_cauchy((reps, n))  # theta* = 0
theta = np.median(x, axis=1)
for _ in range(10):  # Fisher scoring: theta += S_n / (n I) with I = 1/2
    u = x - theta[:, None]
    theta += (2 / n) * (2 * u / (1 + u**2)).sum(axis=1)
z = np.sqrt(n) * theta  # sqrt(n) (theta_hat - theta*)
hist, edges = np.histogram(z, bins=40, range=(-5, 5), density=True)
grid = np.linspace(-5, 5, 201)
d2l.plot([(edges[:-1] + edges[1:]) / 2, grid],
         [hist, np.exp(-grid**2 / 4) / np.sqrt(4 * np.pi)],
         'sqrt(n) * theta_hat', 'density', legend=['4,000 data sets', 'N(0, 2)'])
print(f'standard deviation of sqrt(n) * theta_hat: {z.std():.2f}')
```

```
standard deviation of sqrt(n) * theta_hat: 1.40
```



ヒストグラムは中心、幅、高さのいずれにおいても理論曲線によく一致している。実行ごとの $\sqrt{n}\,\hat\theta$ の標準偏差はおよそ 1.40〜1.47 の間に入り、漸近極限 $\sqrt{2} \approx 1.41$ に近い。なお $n = 100$ では、分散はまだ極限値より数パーセント高いところにある。

36.6.3.4 クラメール・ラオの下界

$1/(nI)$ という分散は優れた値なのだろうか、それともももっと巧妙な推定量ならこれを上回れるのだろうか。実は、フィッシャー情報量は「平均的に正しい」いかなる推定量も突破できない厳格な下界をも定めている。36.5 節で見たとおり、すべての θ に対して $\mathbb{E}_\theta[\tilde{\theta}] = \theta$ が成り立つとき、推定量 $\tilde{\theta}$ は**不偏**（unbiased）であった。

命題（クラメール・ラオの下界） 正則条件のもとで、独立同分布な n 個の観測からスカラーパラメータを推定する任意の不偏推定量 $\tilde{\theta}$ は次を満たす。

$$\operatorname{Var}(\tilde{\theta}) \;\geq\; \frac{1}{n\,I(\theta)} \tag{36.86}$$

出典は (Cramér 1946; Radhakrishna Rao 1945) である。言い換えれば、いかなる不偏推定量も、尤度の曲率が許す以上に高い精度を達成することはできない。証明はわずか 3 行で済み、その唯一の着想は推定量とスコアの相関をとることである。

証明 データセット全体のスコアを $S_n = \partial_\theta \log P(X \mid \theta)$ とおく。これは独立な例ごとのスコア n 個の和であり、平均ゼロ、分散 $nI(\theta)$ をもつ。不偏性より $\int \tilde{\theta}(X)\, P(X \mid \theta)\, dX = \theta$ が成り立つ。両辺を積分記号の下で微分し、$\partial_\theta P = P\, S_n$ を代入すると次式を得る。

$$1 = \int \tilde{\theta}(X)\, S_n\, P(X \mid \theta)\, dX = \mathbb{E}[\tilde{\theta}\, S_n] = \operatorname{Cov}(\tilde{\theta},\, S_n),$$

ここで最後の等号には $\mathbb{E}[S_n] = 0$ を用いた。共分散に対するコーシー・シュワルツの不等式を適用すれば、$1 \leq \operatorname{Var}(\tilde{\theta})\operatorname{Var}(S_n) = \operatorname{Var}(\tilde{\theta})\, nI(\theta)$ が得られる。$\blacksquare$

不偏性は左辺の「1」を通じて効いており、固定された台は積分記号の下で微分する段階で使われている。演習問題 7 では、台が θ とともに動くときにこの下界を上回る不偏推定量の具体例を示す。ベクトルパラメータに対しては、半正定値順序のもとで下界は $\operatorname{Cov}(\tilde{\theta}) \succeq I(\theta)^{-1}/n$ へと一般化される。

MLE の漸近分散 Equation 36.85 はこの下界に厳密に一致しており、この意味で MLE は**漸近有効**（asymptotically efficient）である。すなわち大標本極限において、これより小さい分散を達成する不偏推定量は存在しない。コインの場合、n_H/n は分散 $\theta(1-\theta)/n = 1/(nI)$ をもつ不偏な標本平均であるため、極限だけでなくあらゆる有限の n でこの下界が達成される。しかし一般には、MLE が下界に到達するのは $n \to \infty$ においてのみである。有限の n では統計的バイアスをもちうるため、その場合は下界の対象外となる。次にガウス分布の分散で見るとおりである。

漸近有効性は実務においてどれほど重要だろうか。コーシー位置モデルでは、標本中央値が自然な対抗馬であり、標本平均は素朴な推定量である。対称性により、MLE と中央値はいずれも不偏である。分散の n 倍に対するクラメール・ラオの下界は $1/I = 2$ であるのに対し、中央値の漸近理論は $\pi^2/4 \approx 2.47$ を予測する。下の出力を見る前に、標本平均がどうなるか少し予想してみてほしい。このセルでは先ほどの 4,000 個のデータセットを再利用する。

```
print(f'n Var(MLE) = {n * theta.var():.2f}, '
      f'n Var(median) = {n * np.median(x, axis=1).var():.2f}, '
      f'n Var(mean) = {n * x.mean(axis=1).var():.3g}')
```

```
n Var(MLE) = 1.96, n Var(median) = 2.46, n Var(mean) = 5.32e+05
```

MLE はおよそ 1.9〜2.2 の間に入り、下界である 2 のすぐ近くに収まる一方、中央値は 2.5 付近にある。つまり MLE は約 20% 少ない観測数で同じエラーバーを達成する。対照的に、標本平均は実行ごとに激しく暴れる途方もない分散を示す。独立な n 個のコーシー変数の平均は依然として同じコーシー分布に従うため、平均をとっても一切改善しないのである。

36.6.3.5 理論の限界

ここまでの保証は、前述の正則条件のもと、かつ $n \to \infty$ の極限において成り立つものである。以下の最初の 4 つのケースはいずれもそれらの条件の 1 つを破っており、演習問題で詳しく扱う。5 つ目のケースである誤特定されたモデルは、深層学習のすべての実務家が身を置く状況であるため、ここで詳しく検討する。

- **有限標本での統計的バイアス**。ガウス分布の分散の MLE は残差二乗和を n で割るため、平均的に σ^2 を $(n-1)/n$ 倍だけ過小評価する（36.5 節）。MLE は極限において最適なのであり、あらゆる n で不偏なわけではない。
- **境界**。偏り 0.95 のコインのように真のパラメータが定義域の端近くにある場合、ガウス近似が正確になるまでにはるかに多くのデータを要する。精度を支配するのは n 単体ではなく、まれな結果の期待出現回数だからである。境界上そのものでは、MLE が存在しないことすらある（演習問題 4）。
- **パラメータとともに動く台**。$[0, \theta]$ 上の一様分布では、MLE は $\max_i x_i$ である。その誤差は $1/\sqrt{n}$ ではなく $1/n$ の速さで縮小し、不偏になるよう定数倍した推定量はクラメール・ラオの下界を上回る（演習問題 7）。
- **特異な情報行列**。ReLU ネットワークのリスケーリング対称性によりフィッシャー情報行列 I は零ベクトルをもち、したがって I^{-1} は存在しない（演習問題 6）。

モデルの誤特定（misspecification）。モデル族のいかなる要素 p_θ も真のデータ生成分布と一致しないとき、MLE は真の分布からの KL ダイバージェンスを最小化するパラメータへと収束する (White 1982)。その極限でもスコアの期待値は依然としてゼロになるが、スコアの分散 B と NLL の期待曲率 A はもはや一致しない。その結果、先ほどの導出のステップ III から得られる分散は B/A^2（$1/I$ ではなく）となり、行列形式では**サンドイッチ**共分散（sandwich covariance）$\mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}$ となる（演習問題 8）。たとえば、平均 m、分散 $m + m^2/\alpha$ をもつ過分散なカウントデータに、レート λ のポアソンモデルを当てはめる場合を考えよう。ポアソン分布の MLE は標本平均であるため、依然として m を復元し、その真の分散は $(m + m^2/\alpha)/n$ である。ところがポアソンモデルのフィッシャー情報量 $I = 1/\lambda$ が予測する分散は m/n であり、これは $1 + m/\alpha$ 分の 1、すなわち 3 分の 1（$m = 4$、$\alpha = 2$ のとき）だけ過小になっている。

深層ニューラルネットワークは誤特定されており、かつ識別不能でもあるため、個々の重みに古典的なエラーバーを付けることはできない。それでも理論が裏づけるのは、当てはめられた予測分布に関する主張である。パラメータの対称性は予測分布を変えないからである。なお、フィッシャー情報行列そのものは本書の後半で 2 度再登場する。1 度目はパラメトリックな分布族における近傍の分布間の KL ダイバージェンスの局所曲率（ヘッセ行列）として（37.3 節）、2 度目は自然勾配最適化で用いられる前処理行列としてである（35.5 節）。

36.6.4 MAP 推定と正則化

13 回のコイン投げからは妥当な推定値が得られたが、3 回のコイン投げでは最尤法が完全に破綻しうる。3 回投げて HHH が出たとしよう。MLE は $\hat{\theta} = 3/3 = 1$ となり、裏に確率ゼロを割り当てるため、その後に裏が 1 度でも観測された瞬間に無限大の NLL を被ってしまう。訓練中に一度も現れなかった特徴量とクラスの組み合わせに遭遇したカウントベースの分類器も、これとまったく同じ病的な現象に直面する。コインを手にしたことのある人なら誰一人として、3 回表が出たからといって裏が不可能だとは考えない。データを見る前の段階で「コインが完全に片面しか出ないことはめったにない」という知識を原理に基づいて表現する方法が必要である。

パラメータ上の確率分布を導入すれば、そのような事前知識を捉えられる。最尤法はデータセットにしか確率を割り当てなかったため、事前知識を取り入れる余地がなかった。**事前分布** $p(\theta)$ を導入すると、ベイズの定理は事前分布と尤度を組み合わせ、観測データが与えられたもとでのパラメータ上の分布を与える。

$$p(\theta \mid X) = \frac{P(X \mid \theta)\, p(\theta)}{P(X)}, \qquad P(X) = \int P(X \mid \theta)\, p(\theta)\, d\theta. \tag{36.87}$$

左辺は**事後分布**（posterior）であり、データを見た後に更新された信念を表す。分母は**エビデンス**、すなわちすべてのパラメータ値にわたって平均をとったデータの周辺確率であり、決定的に重要な点として θ に依存しない。**最大事後確率推定**（maximum a posteriori、MAP）による推定値（MAP 推定値）とは、事後分布の最頻値（ピーク）である。エビデンスは θ に関して定数であるため、事後分布を最大化して負の対数をとると次式が得られる。

$$\begin{aligned} \hat{\theta}_{\text{MAP}} &= \operatorname*{argmax}_{\theta}\ P(X \mid \theta)\, p(\theta) \\ &= \operatorname*{argmin}_{\theta}\ \Big[\underbrace{-\textstyle\sum_i \log p(x_i \mid \theta)}_{\text{data fit (NLL)}}\ \underbrace{-\log p(\theta)}_{\text{penalty}} \Big]. \end{aligned} \tag{36.88}$$

したがって MAP 推定は、おなじみの NLL に加法的ペナルティである負の対数事前分布を加えたものを最小化する。極端なパラメータ値に低い確率を割り当てる事前分布はそうした値にペナルティを科し、まさしく正則化項の役割を果たす。HHH に対して、仮想的な表 1 回と裏 1 回を加えることに相当する事前分布（後述する Beta$(2,2)$ 事前分布）を用いれば、$\hat{\theta}_{\text{MAP}} = (3+1)/(3+2) = 4/5$ が得られる。このとき、続いて裏が出たときのコストは $-\log(1/5) = 1.6$ nat となり、無限大ではなく有限の驚きで済む。

36.6.4.1 ガウス事前分布と重み減衰

深層学習における標準的な正則化手法である重み減衰（weight decay）は、ガウス事前分布がもたらす MAP のペナルティ項である。これが成り立つのは L_2 ペナルティ形式の重み減衰に対してであり、Adam とともに用いられる分離型の変種（9.5.3 節）はもはや厳密な MAP ペナルティではない。2.7 節では線形回帰についてこの対応関係を導いたが、ここではターゲットにガウスノイズを含む任意のモデルに対して本節の記法で記述しよう。各パラメータが事前には小さく互いに独立に $\theta_j \sim \mathcal{N}(0, \tau^2)$ に従うと仮定する。ここで τ は想定される典型的な大きさを表す。事前密度は $\exp\!\big(-\|\theta\|_2^2/(2\tau^2)\big)$ に比例するため、その負の対数は定数項を除いて $\|\theta\|_2^2/(2\tau^2)$ に等しい。これをガウス分布の NLL Equation 36.82 と組み合わせると、MAP の目的関数 Equation 36.88 は次式となる。

$$\hat{\theta}_{\text{MAP}} = \operatorname*{argmin}_{\theta}\ \frac{1}{2\sigma^2} \sum_i (y_i - \hat{y}_i)^2 + \frac{1}{2\tau^2}\|\theta\|_2^2.$$

全体に正の定数 σ^2/n を掛けても最小点は変わらず、2.7 節の重み減衰の目的関数が復元される。

$$\frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\big(y_i - \hat{y}_i\big)^2 + \frac{\lambda}{2}\|\theta\|_2^2, \qquad \lambda = \frac{\sigma^2}{n\tau^2}.$$

本書で通常そうしているようにバイアス b にペナルティを科さないことは、b 上に平坦な事前分布を置くことに対応する。平均損失におけるペナルティ重み λ が $1/n$ に比例して縮小することにも注目しよう。事前分布はデータが少ないときに最も強く効き、標本が大きくなるにつれて消えていく。なお 35.7.3.2 節では、同じペナルティを重みのノルムに対する制約として同値に解釈する。

ガウス事前分布をスケール s の**ラプラス**事前分布 $p(\theta) \propto e^{-\|\theta\|_1/s}$ に置き換えると、L_1 ペナルティ $\|\theta\|_1/s$ が得られる。このペナルティはゼロに鋭い折れ曲がりをもつため、厳密なゼロを取りやすくし、スパースなパラメータベクトルを促す。Figure 36.33 はこの綱引きを 1 次元で描いたものである。NLL は MLE へ向かって引っ張り、負の対数事前分布は事前平均へ向かって引っ張り、MAP 推定値はその中間に落ち着く。

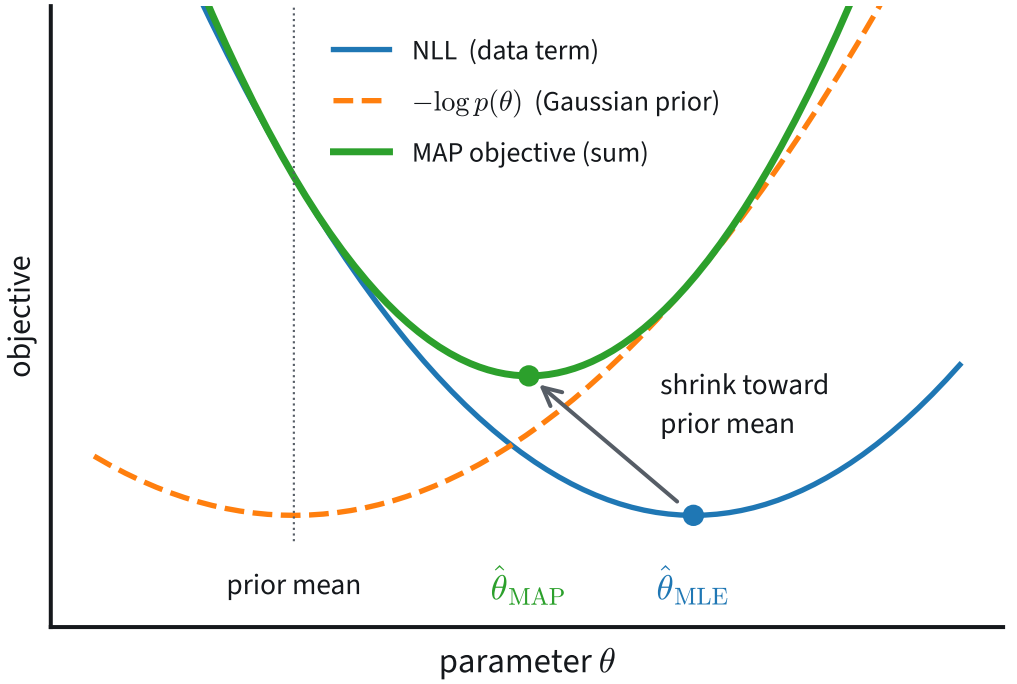


Figure 36.33: 最大事後確率推定はデータと事前分布のバランスをとる。負の対数尤度（青）は MLE で最小化されるが、事前平均を中心とするガウス事前分布の負の対数（オレンジ、破線）を加えると MAP の目的関数（緑）が得られ、最小点が MLE から事前平均の方向へ引き寄せられる。事前分布が狭い（τ が小さい）ほど強く引き寄せられ、逆に $\tau \to \infty$ では事前分布が平坦になって MAP は MLE に帰着する。

同じ図から、2 つの極限的な場合も直ちに理解できる。事前分布が平坦になる（$\tau \to \infty$ または $s \to \infty$）とペナルティは消滅する。また標本サイズ n が大きくなると、事前分布が固定されたまま NLL が n に比例して深くなる。いずれの極限においても、MAP は再び MLE へと収束する。

36.6.4.2 ベータ事前分布と擬似カウント

コインではパラメータ θ が $[0,1]$ 上にあるため、ベータ分布族が自然な事前分布となる。$a, b > 0$ をもつ Beta(a,b) 事前分布の密度は $\theta^{a-1}(1-\theta)^{b-1}$ に比例する。これにコインの尤度 Equation 36.80 を掛けると指数に観測回数がそのまま加算されるため、事後分布もベータ分布族にとどまる。これは 36.4 節で見た共役更新にほかならない。

$$p(\theta \mid X) \;\propto\; \theta^{\,n_H + a - 1}\,(1-\theta)^{\,n_T + b - 1}.$$

この事後分布は、表 $n_H + a - 1$ 回・裏 $n_T + b - 1$ 回のコインの尤度とまったく同じ関数形をしている。したがって両方の指数が正である限り、その最大化はコインの MLE を求めることと同一であり、「表の指数を合計の指数で割った値」として次式で与えられる。

$$\hat{\theta}_{\text{MAP}} \;=\; \frac{n_H + a - 1}{n + a + b - 2}, \qquad n = n_H + n_T. \tag{36.89}$$

つまり事後最頻値（posterior mode）にとっては、経験頻度をとる前に仮想的な表 $a-1$ 回と仮想的な裏 $b-1$ 回を標本に付け加えたかのように事前分布が働く。これは 36.4 節の慣例とも整合している。すなわち a と b は事後平均（posterior mean）の**擬似カウント**（pseudo-counts）であるのに対し、$a-1$ と $b-1$ は事後最頻値の擬似カウントである。$\text{Beta}(2,2)$ 事前分布は最頻値に対して表裏それぞれ 1 回ずつの仮想試行を加え、先ほどの $9/13 \approx 0.692$ を $(9+1)/(13+2) = 2/3$ に、HHH を $4/5$ にシフトさせる。強い事前分布 $\text{Beta}(30,30)$ は、観測された試行回数がその 58 回分の仮想試行を上回るまで、MAP 推定値を $1/2$ 付近に繋ぎ止める。

これら 2 つの要約量は、ある古典的な公式において一致する。平坦な事前分布 $\text{Beta}(1,1)$ のもとでの事後平均はラプラスの継起則 $(n_H + 1)/(n + 2)$ であり (Laplace 1814)、カウントベースの分類器では 1 加算平滑化（add-one smoothing）として広く知られている。これとまったく同じ比が $\text{Beta}(2,2)$ のもとでの MAP 推定値となる一方、平坦な $\text{Beta}(1,1)$ 事前分布のもとでの MAP は単なる生の MLE を返すにすぎない。したがって、1 加算平滑化は平坦な事前分布のもとでの事後平均であって、平坦な事前分布による MAP 推定値ではない。

36.6.4.3 事後最頻値と事後分布

事後最頻値には微妙な弱点がある。モデルをどのようにパラメータ化するかに依存してしまうのである。非線形な変数変換のもとでは、あらゆる領域の確率が同一に保たれるにもかかわらず、確率密度にはヤコビ行列式（ヤコビアン）の因子が掛かるためピークの位置がずれる。$\text{Beta}(1,1)$ 事前分布は θ に関しては平坦だが、まったく同じ信念を対数オッズ（log-odds）$\eta = \log \frac{\theta}{1-\theta}$ で表すと平坦ではなくなり、η に関して MAP 推定値をとると $10/15$（$9/13$ ではなく）が得られる（演習問題 5）。一方、尤度の値はパラメータ空間の各点に直接結びつけられたデータの確率であるため、ヤコビ行列式の因子は生じない。その結果、MLE は**同変**（equivariant）である。すなわち、任意の 1 対 1 変換 g に対して、$\hat{\theta}$ が θ の MLE であれば、$g(\hat{\theta})$ は $g(\theta)$ の MLE となる (Wasserman 2013)。たとえばコインの対数オッズの MLE は、単に $\log \frac{9/13}{4/13} = \log \frac{9}{4}$ である。最尤法が「平坦な事前分布をもつ MAP」としてではなく事前分布なしで定義されるのはこのためである。もし前者として定義すれば、パラメータを書き換えるたびに答えが動いてしまう。なお、同変性にも固有のトレードオフがある。同じ演習問題で示すとおり、あらゆる再パラメータ化のもとで同変かつ不偏であり続ける推定量は存在しない。

さらに根深い限界は、MAP も MLE と同様に、得られた知識を単一のパラメータベクトルへと潰してしまう点にある。完全な事後分布 $p(\theta \mid X)$ は、データを見た後にパラメータがどれほど不確実なまま残っているかを捉えている。その不確実性にわたって平均をとる予測 $\int p(x \mid \theta)\, p(\theta \mid X)\, d\theta$ はパラメータの不確実性を考慮に入れるのに対し、最頻値を代入するだけの予測はそれを捨て去ってしまう。このような積分が閉形式をもつことはまれであるため、それらをいかに近似するかが 36.8 節の主題となる。

しかしその前に、より直接的な障害が立ちはだかる。本節の議論はすべて、各観測の確率をパラメータの関数として直接書き下せることを前提としていた。各データ点が複数の隠れた生成源のいずれかから生じるものの、どれから生じたかは観測できない場合、その確率は隠れた可能性に

わたる和となり、和の対数はもはや単純な項へと綺麗に分解できなくなる。36.7 節では、その代わりに何を最適化すべきかを示す。

36.6.5 まとめ

最尤法は観測されたデータが最も起こりやすくなるパラメータを選ぶ原理であり、実際には例ごとの驚きの和である負の対数尤度を最小化することを意味する。標準的な訓練損失の大半はこのようにノイズモデルの選択から導かれる。すなわちガウスノイズからは二乗誤差、ラプラスノイズからは絶対誤差、離散的な結果からは交差エントロピーが生じ、離散の場合の平均損失は経験頻度のエントロピーを決して下回ることができない。同じ損失の曲率は推定の精度を定量化する。正則条件のもとで MLE の誤差は近似的に分散 $1/(nI)$ のガウス分布に従い、ここでフィッシャー情報量 I はスコアの分散と NLL の期待曲率の両方に等しく、不偏推定量に対するクラメール・ラオの下界に一致する。これらの古典的な保証はモデルが誤特定されている場合や識別不能である場合には破綻し、深層ニューラルネットワークはその両方に該当する。最後に、事前分布を導入すると MAP 推定は NLL と正則化項の和へと変わり、ガウス事前分布からは重み減衰が、ベータ事前分布からは擬似カウントが導かれるが、それでも MAP 推定値は事後分布を単一の点へと縮約してしまうことに変わりはない。

36.6.6 演習問題

1. **交差エントロピーの下限。**離散的な結果に対しては、36.6.2.1 節より平均 NLL は $H(\hat{p}_{\text{data}}) + D_{\text{KL}}(\hat{p}_{\text{data}} \,\|\, p_\theta)$ に等しいため、経験エントロピーを下回ることはできない。
   1. 表 9 回・裏 4 回のコインの系列に対してこの下限を nat 単位で求め、さらに公平なコインのモデル $\theta = 1/2$ における KL ギャップを評価せよ。
   2. 分類器が $p(y \mid \mathbf{x})$ をモデル化し、異なる各入力に対して個別にラベル上の任意の分布を表現できるとする。訓練セット上で達成可能な最小の平均 NLL が経験条件付きエントロピー $\sum_{\mathbf{x}} \hat{p}_{\text{data}}(\mathbf{x})\, H\big(\hat{p}_{\text{data}}(\cdot \mid \mathbf{x})\big)$ であることを示せ。また、100 個の異なる入力がそれぞれ 10 回ずつ現れ、そのうちちょうど 1 回だけ誤ったラベルが付いている二値分類タスクについて、この値を評価せよ。
   3. 次に、典型的な画像データセットのように各訓練入力が 1 度しか現れないとする。(b) の下限はいくらになるか。このとき訓練損失がゼロに近いことからラベルのノイズの多寡について何も言えないのはなぜか。また、実際に測定可能な量のうち、母集団の条件付きエントロピー $H(Y \mid X)$ を下限としてもつものはどれか。
2. **学習されるノイズ分散。**ガウス分布の NLL Equation 36.82 では分散 σ^2 を固定していたが、ここでは分散も併せて学習する。
   1. 1 つの共通な σ^2 を θ と並ぶパラメータとする。NLL を σ^2 について（θ を固定して）最小化し、残る目的関数が $\frac{n}{2}\log\big(\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2\big)$ に定数を加えたものになることを示せ。θ に関する最小点は依然として最小二乗法による当てはめと一致するか。
   2. ネットワークが各入力に対して平均 $\hat{y}(\mathbf{x})$ と分散 $\hat{\sigma}^2(\mathbf{x}) > 0$ を出力するとする。加法定数を除くと、訓練セット上での NLL は次式となる。

$$\sum_{i=1}^{n} \left[\frac{(y_i - \hat{y}(\mathbf{x}_i))^2}{2\hat{\sigma}^2(\mathbf{x}_i)} + \frac{1}{2}\log \hat{\sigma}^2(\mathbf{x}_i) \right].$$

   ネットワークが 1 つの訓練点に厳密に一致しつつその分散を限りなく小さくできるとき、この目的関数が下に有界でなくなることを示せ。36.7 節における混合ガウスモデルの尤度のどの性質が同じ原因から生じるか。
   3. 例 i の項の $\hat{y}(\mathbf{x}_i)$ に関する勾配を計算せよ。予測分散が残差の二乗に追従し、$\hat{\sigma}^2(\mathbf{x}_i) \approx r_i^2$（ただし $r_i = y_i - \hat{y}(\mathbf{x}_i)$）となるとする。このとき勾配の大きさは $|r_i|$ にどのよ

うに依存し、二乗誤差の場合とどう異なるか。また、この重み付けは現在当てはめが悪い領域における平均の学習にどのような影響を与えるか (Seitzer et al. 2022)。

3. **尤度ではない損失**。$\phi(y,o)$ をラベル y とモデル出力 o の損失とし、$Z(o)=\sum_y e^{-\phi(y,o)}$ とおく。もし $Z(o)$ が o に依存しなければ、$e^{-\phi(y,o)}/Z$ は y の（o を与えたときの）条件付き分布となり、その NLL は ϕ に定数を加えたものになるため、ϕ の最小化は最尤法となる。
   1. ラベル $y\in\{-1,+1\}$ に対し、ロジスティック損失 $\phi(y,o)=\log(1+e^{-yo})$ が $Z(o)=1$ を満たすことを示し、このモデルが $y=+1$ に割り当てる確率を特定せよ。
   2. サポートベクターマシンのヒンジ損失 $\phi(y,o)=\max(0,1-yo)$ について、$|o|\le 1$ の場合と $|o|>1$ の場合に分けて $Z(o)$ を計算せよ。そこから、ヒンジ損失が定数差を除いても y の（o を与えたときの）いかなる条件付き分布の NLL にもならないことを結論づけよ。ヒンジ損失にどの項を加えれば $e^{-\phi(y,o)}/Z(o)$ の NLL が得られるか。また、その項は $|o|$ が大きくなるにつれてどのように振る舞うか。
   3. 実数ターゲット y と予測 $\hat{y}$ に対し、フーバー損失のように残差のみに依存する任意の損失 $\phi(y,\hat{y})=\rho(y-\hat{y})$（ただし $\int e^{-\rho(r)}\,dr<\infty$）をとる。$Z(\hat{y})=\int e^{-\rho(y-\hat{y})}\,dy$ が $\hat{y}$ に依存しないことを示せ。このような回帰損失では本問の条件が自動的に成り立つのに対し、分類損失では強い制約となるのはなぜか。
4. [code] **境界：分離可能なデータとまれな結果**。最適なパラメータが定義域の端にあるとき、最尤法は異常な振る舞いを示す。
   1. ラベル付きの 2 点（$x=-1$ で $y=0$、および $x=+1$ で $y=1$）にバイアスなしのロジスティック回帰 $P(y=1\mid x)=\mathrm{sigmoid}(wx)$ を当てはめる。NLL が $2\log(1+e^{-w})$ であり、すべての w に対して単調に減少して 0 に近づくものの、いかなる有限の w もその下限を達成しないことを示せ。本文中のどのコイン系列がこれと同じ問題を抱えているか。また $w\to\infty$ は $x=0.1$ にある新しい点に対して何を予測するか。
   2. ガウス事前分布 $w\sim\mathcal{N}(0,\tau^2)$ を加える。MAP 推定値が $2\,\mathrm{sigmoid}(-w)=w/\tau^2$ の解となることを示し、$\tau=1$ の場合に数値的に解け。さらに一般化し、任意の線形分離可能なデータセットに対してロジスティック回帰の NLL は下限 0 をもち最小点をもたないが、ガウス事前分布を導入すれば有限の MAP 推定値が回復されることを示せ。
   3. コインにおいて、$\hat{\theta}=n_H/n$ の歪度は $(1-2\theta)/\sqrt{n\theta(1-\theta)}$ である。これを $\theta=0.7$ と $\theta=0.95$ について $n=400$ で評価し、どの n で $\theta=0.95$ の歪度が $\theta=0.7$（$n=400$ のとき）と同じ水準に達するかを求めよ。また θ が 1 に近いとき、裏の期待出現回数 $n(1-\theta)$ を用いて歪度を表せ。ガウス極限が正確かどうかを決めるのが n 単体ではなくその期待回数であるのはなぜか。
   4. $n=400$ 回のコイン投げからなる 20,000 個のデータセットをシミュレーションし（`rng.binomial(n, theta, 20000) / n`）、$\sqrt{n}(\hat{\theta}-\theta)$ のヒストグラムを $\theta=0.7$ と $\theta=0.95$ について $\mathcal{N}(0,\theta(1-\theta))$ と重ねて描け。$\theta=0.95$ のヒストグラムが理論曲線からどのように乖離するかを記述せよ。
5. **対数オッズにおける MAP**。コインをその対数オッズ $\eta=\log\frac{\theta}{1-\theta}$ によってパラメータ化する。このとき $\theta=\mathrm{sigmoid}(\eta)$ かつ $\frac{d\theta}{d\eta}=\theta(1-\theta)$ である。一様事前分布 $\mathrm{Beta}(1,1)$（θ 上）と、表 9 回・裏 4 回の系列はそのまま用いる。
   1. 密度の変数変換の公式 $p_\eta(\eta)=p_\theta(\theta)\,|d\theta/d\eta|$（36.2 節）を用いて、この信念が導く η の事前密度を求めよ。それがロジスティック密度 $e^{-\eta}/(1+e^{-\eta})^2$、すなわち $\log\frac{U}{1-U}$（U は一様確率変数）の分布であることを示し、計算を行わずに逆変換サンプリングを用いてその結果を説明せよ。
   2. η の MAP 推定値を求め、それを θ に写し戻せ。得られた値を、θ パラメータ化での MAP 推定値、36.6.4.3 節の MLE $\log\frac{9}{4}$、およびラプラスの継起則と比較せよ。
   3. 一般の事前分布 $\mathrm{Beta}(a,b)$ と出現回数 n_H,n_T に対して (b) を繰り返せ。対数オッズ

での MAP 推定値は、θ 上の事後分布のどの要約量を再現するか。

4. MLE は同変であるため、θ^2 の MLE は $\widehat{\theta}^2$ である。その統計的バイアスが $\mathbb{E}[\widehat{\theta}^2] - \theta^2 = \mathrm{Var}(\widehat{\theta}) = \theta(1-\theta)/n$ であることを示し、$\theta = 9/13$、$n = 13$ で評価せよ。そこから、すべての滑らかな 1 対 1 の再パラメータ化のもとで同変であり、かつそのすべてに対して不偏であるような推定量は存在しないことを結論づけよ。

6. **ReLU ネットワークの対称性。**●スカラー入力 x とパラメータ $\theta = (a_1, w_1, \dots, a_h, w_h)$ をもつネットワーク $f_\theta(x) = \sum_{j=1}^{h} a_j \,\mathrm{ReLU}(w_j x)$、および固定された σ^2 をもつ回帰モデル $y \sim \mathcal{N}(f_\theta(x), \sigma^2)$ を考える。識別可能性は 36.6.3 節で定義されている。
   1. (a_j, w_j) を $(c\,a_j,\ w_j/c)$（任意の $c > 0$）に置き換えても f_θ が変化しないことを示せ。どの 2 つのユニットも一致しないとき、隠れユニットの置換によって同じ関数を与える異なるパラメータベクトルは何通り生じるか。
   2. Equation 36.83 から、このモデルのフィッシャー情報量が $I(\theta) = \mathbb{E}_x\big[\nabla_\theta f_\theta(x)\, \nabla_\theta f_\theta(x)^\top\big]/\sigma^2$ となることを示せ。(a) のリスケーリングを c に関して $c = 1$ で微分し、非ゼロベクトル $\mathbf{v}$ で $I(\theta)\mathbf{v} = \mathbf{0}$ を満たすものを求めよ。Equation 36.85 の背後にあるどの仮説（条件）が破綻するか。
   3. 平均二乗誤差 $\frac{1}{n}\sum_i (y_i - f_\theta(x_i))^2$ の a_j に関するヘッセ行列の対角成分は $\frac{2}{n}\sum_i \mathrm{ReLU}(w_j x_i)^2$ である。この成分は (a) のリスケーリングのもとでどのように変化するか。w_j に対する対角成分を計算し、ReLU ユニットでは（折れ曲がり点 $w_j x_i = 0$ を除いて）残差項が含まれない理由を説明せよ。また、その成分はリスケーリングのもとでどう変化するか。このことは、ヘッセ行列の固有値に基づく最小点の平坦性の指標に対して何を意味するか (Dinh, Pascanu, et al. 2017)。

7. **動く台。**●$x_1, \dots, x_n$ が未知のパラメータ $\theta > 0$ をもつ $[0, \theta]$ 上の一様分布に独立に従うとする。このモデルは 36.6.3 節の正則条件の 1 つに違反している。
   1. 尤度を θ の関数として書き下し、MLE が $\widehat{\theta} = \max_i x_i$ であることを示せ。導関数をゼロとおく方法ではこの解が見つからないのはなぜか。
   2. $P(n(\theta - \widehat{\theta}) > t) = \big(1 - \frac{t}{n\theta}\big)^n$（$0 \le t \le n\theta$）を示し、$n \to \infty$ としたときの極限を求めよ。誤差は n に関してどの速さで縮小するか。また極限分布はガウス分布になるか。
   3. 台の上でスコアは $s(x;\theta) = -1/\theta$ である。$\mathbb{E}_\theta[s]$ を計算せよ。スコアの期待値がゼロになる導出のどの段階が破綻しているか（演習問題 1（34.7 節）と比較せよ）。
   4. 不偏推定量 $\frac{n+1}{n}\widehat{\theta}$ の分散は $\theta^2/(n(n+2))$ である。この値を、Equation 36.86 が下界として与えるはずの値 $1/(n\,\mathbb{E}_\theta[s^2])$ と比較せよ。ここでは下界の証明のどの段階が破綻しているか。

8. [extended] **モデルの誤特定とサンドイッチ共分散。**●●本問では、フィッシャー情報量の 2 つの表現が一致しない場合について漸近正規性の発見的な導出を行う。モデル族のどの分布もデータ生成法則と一致しないとき、MLE は母集団の KL ダイバージェンスを最小化する擬真のパラメータ $\theta^\dagger$ へ収束し（36.6.3.5 節）、そのばらつきはもはや I^{-1}/n ではなくなる (White 1982)。データ生成法則のもとでの期待値を用いて、$\mathbf{A} = -\mathbb{E}\big[\nabla_\theta^2 \log p(x \mid \theta^\dagger)\big]$ および $\mathbf{B} = \mathbb{E}\big[s(x;\theta^\dagger)\, s(x;\theta^\dagger)^\top\big]$ とおく。
   1. 一階条件 $\sum_i s(x_i;\widehat{\theta}) = \mathbf{0}$ を $\theta^\dagger$ の周りで一次まで展開し、$\frac{1}{\sqrt{n}}\sum_i s(x_i;\theta^\dagger)$ に中心極限定理を適用することで、次式が成り立つことを論じよ。

$$\sqrt{n}\,(\widehat{\theta} - \theta^\dagger) \xrightarrow{d} \mathcal{N}\big(\mathbf{0},\ \mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}\big). \tag{36.90}$$

   モデルが誤っているにもかかわらず $\mathbb{E}\big[s(x;\theta^\dagger)\big] = \mathbf{0}$ が成り立つのはなぜか。また、モデルが正しいときにはこの共分散が Equation 36.85 の共分散に帰着することを示せ。

2. 平均 m、形状パラメータ α（分散は $m + m^2/\alpha$）の負の二項分布からのカウントデータに、レート λ のポアソンモデルを当てはめる。$\hat{\lambda}$、$\lambda^{\dagger}$、$\mathbf{A}$、$\mathbf{B}$ を求めよ。逆フィッシャー情報量は $\hat{\lambda}$ の分散を何分の 1 に過小評価するか。
3. 1 つの特徴量をもつポアソン回帰 $\log \lambda(x) = w_0 + w_1 x$ をニュートン法によって当てはめよ。このモデルに対する更新式は $\mathbf{w} \leftarrow \mathbf{w} + (\mathbf{Z}^{\top} \operatorname{diag}(\lambda_i)\mathbf{Z})^{-1}\mathbf{Z}^{\top}(\mathbf{y} - \lambda)$ であり、ここで各行 $(1, x_i)$ をもつ計画行列を $\mathbf{Z}$ とし、$\lambda_i = \lambda(x_i)$ とする。ここでは 1 つの観測がペア (x_i, y_i) であり、$\log p(y_i \mid x_i, \mathbf{w})$ が $\log p(x \mid \theta)$ の役割を $\mathbf{A}$ と $\mathbf{B}$ において担う。当てはめた $\hat{\mathbf{w}}$ における標本平均によって両者を推定せよ。次に、2,000 個のデータセット（各 $n = 500$ 組のペア）を、$x \sim \mathcal{N}(0, 1)$、$\mathbf{w} = (1, 0.5)$、および $\alpha = 2$ の負の二項分布に従うカウントから生成せよ。すなわち、形状 α、平均 $\lambda(x)$ のガンマ分布からレートを引き、そのレートでポアソン分布からカウントを引く。そして $\hat{w}_1$ の 3 種類の標準誤差を比較せよ。逆フィッシャー情報量から求めたもの、サンドイッチ公式から求めたもの、およびデータセット間の $\hat{w}_1$ のばらつきから求めたものの 3 つである。最初の 2 つから構成した公称 95% 信頼区間のうち、それぞれどの割合が $w_1 = 0.5$ を被覆するか。さらに $\alpha = 10^6$ として同じ実験を繰り返せ。
4. 演習問題 6c（36.4 節）は、カウントが過分散であってもポアソン回帰の当てはめが依然として $\mathbf{w}$ を復元することを示す。モデルのどの部分が誤っていれば $\mathbf{w}^{\dagger}$ は $\mathbf{w}$ と異なることになるか。また、その場合 (c) のサンドイッチ信頼区間はどのパラメータを被覆するか。

36.7 潜在変数、EM、ELBO

36.6 節で扱った尤度はどれも、観測ごとに 1 つの因子を持つきれいな積の形をしていたため、対数をとれば項ごとに微分できる和へと変換できた。この扱いやすさは、モデルが生成するすべての量を観測できるという暗黙の仮定の上に成り立っていたのである。

実際には、データの重要な一部が欠けていることが少なくない。2 つの異なる母集団から得られた 500 個の測定値（一方から 300 個、もう一方から 200 個）を集めたものの、どの測定値がどちらの母集団から生じたのか誰も記録していなかったとしよう。もしそのラベルが残っていれば、モデルの当てはめは母集団ごとに混合比・平均・分散を計算するだけの 1 行で済むはずである。しかしラベルがなければ、各測定値の確率は、それが生じ得た 2 つの母集団にわたる和となる。厄介なことに、和の対数は分解できないため、すべてのパラメータがあらゆる項の中で絡み合い、勾配を 0 と置いても閉形式解（closed-form solution）を持たない方程式しか残らない。顧客のクラスタ、文書のトピック、あるいは生成器が画像へと変換するランダムなコードなど、隠れた構造を通じて観測を説明するモデルでは常に同じ障壁が現れる。このような観測されない量は**潜在変数**（latent variable）と呼ばれる。

本節の中心的なアイデアは単純でありながら驚くほど有効である。すなわち、見えないものを確定的なハードラベルではなく確率を使って推測し、そのソフトな推測が実際のデータであるかのようにモデルを当てはめるのである。このアイデアがうまく機能する理由は、推測が正しい必要すらない点にある。**いかなる**確率的な推測に対しても、そこから得られる扱いやすい目的関数は真の対数尤度（log-likelihood）の下側に位置し、その推測が隠れたラベルに関する事後分布（posterior distribution）に一致するときに限り、対数尤度にぴったり接する。この下界こそ、覚えておくべき重要な結果である。期待値最大化法（expectation–maximization）はこの下界を直接登り、厳密な事後分布が計算できない場合には、変分推論（variational inference）や変分オートエンコーダ（variational autoencoder）がこれを近似的に登る。

以下では、この 1 つのアイデアを最初から最後まで展開していく。まず混合モデル（mixture model）を定式化し、次に下界を導出してそのギャップを測り、最後に得られたアルゴリズムを先ほどの 500 点に対して実行する。その際、両方の母集団を復元できる初期値から始める場合と、

対称な初期値から始めて途中で立ち往生する場合の両方を確かめる。

36.7.1 混合モデル

このラベル欠損のパズルを数式で表してみよう。潜在変数 z を持つ確率モデル（probabilistic model）は、隠れた量と観測される量の同時分布（joint distribution）を次のように定める。

$$p(x, z \mid \theta) = p(z \mid \theta)\, p(x \mid z, \theta),$$

このうち観測されるのは x のみである。実際に観測されたデータの尤度を計算するには、隠れ変数がとり得るすべての値について和をとる（z が連続ならば積分して消去する）必要があり、これにより**周辺**尤度（marginal likelihood）が得られる。

$$p(x \mid \theta) = \sum_{z} p(z \mid \theta)\, p(x \mid z, \theta). \tag{36.91}$$

言葉で言えば、モデルが x に割り当てる確率は、モデルが x を生成し得たあらゆる経路を、隠れ変数の値ごとに 1 項ずつ足し合わせたものになっている。

代表的な例が**混合ガウスモデル**（Gaussian mixture model、GMM）である。このモデルでは、潜在変数 $z \in \{1, \dots, K\}$ が成分 k を混合確率 π_k で選び、続いて観測がその成分のガウス分布 $x \mid z{=}k \sim \mathcal{N}(\mu_k, \sigma_k^2)$ から抽出される。n 個の観測をまとめると、負の対数尤度（negative log-likelihood、NLL）は次のようになる。

$$\ell(\theta) = -\sum_{i=1}^{n} \log \Big(\sum_{k=1}^{K} \pi_k \, \mathcal{N}(x_i;\, \mu_k, \sigma_k^2) \Big), \qquad \theta = (\pi_k, \mu_k, \sigma_k)_{k=1}^{K}.$$

厄介さのすべては、和が対数の**内側**に入り込んでいることに起因する。対数は混合分布のところで止まってしまい、個々のガウス密度まで届かない。もし届けば、指数関数と打ち消し合って単純な二次式が残るはずである。これを**完全データ**（complete-data）の問題と比べてみよう。仮にオラクルがすべてのラベル z_i を教えてくれたとすれば、尤度は成分ごとにきれいに分離し、最尤法（maximum likelihood）は既知の推定量へと帰着する。各混合重み π_k は単に成分 k に属するデータ点の割合となり、コインの偏りの推定量 $\hat{\theta} = n_H/n$（36.6 節）とまったく同じ形になる。一方、μ_k と σ_k^2 はそれらのデータ点の通常の標本平均と分散になる。ラベルが欠けているときに限って問題が難しくなるのだから、自然な戦略が浮かび上がる。欠けているラベルを確率として推定し、そのうえで簡単な完全データの問題を解けばよいのである。

1 つのデータ点に対するそのような確率的推測は、どのような形になるだろうか。$x = 0.5$ における 1 つの観測を、等しい重みを持つ 2 つの成分 $\mathcal{N}(-1, 1)$ と $\mathcal{N}(1, 1)$ のもとで考えよう。$x = 0.5$ における両成分の密度はそれぞれ $e^{-(1.5)^2/2} = e^{-1.125}$ と $e^{-(0.5)^2/2} = e^{-0.125}$ に比例するため、ベイズの定理により左側の成分には次の事後確率（posterior probability）が割り当てられる。

$$\frac{e^{-1.125}}{e^{-1.125} + e^{-0.125}} = \frac{1}{1 + e} = 0.27.$$

ベイズの定理はハードな選択を強いるのではなく、このデータ点を 2 つの成分の間で 27% 対 73% に分割する。観測をずっと右側の、たとえば $x = 3$ に動かせば、事後分布は 99.8% の確信

度で右側の成分に属すると判定し、中点の $x = 0$ に置けば、分割はちょうど半々の 50% 対 50% になる。ソフトな推測は、データが明確な領域では確信を持ち、成分が重なり合う領域では適切に判断を留保する（Figure 36.34）。

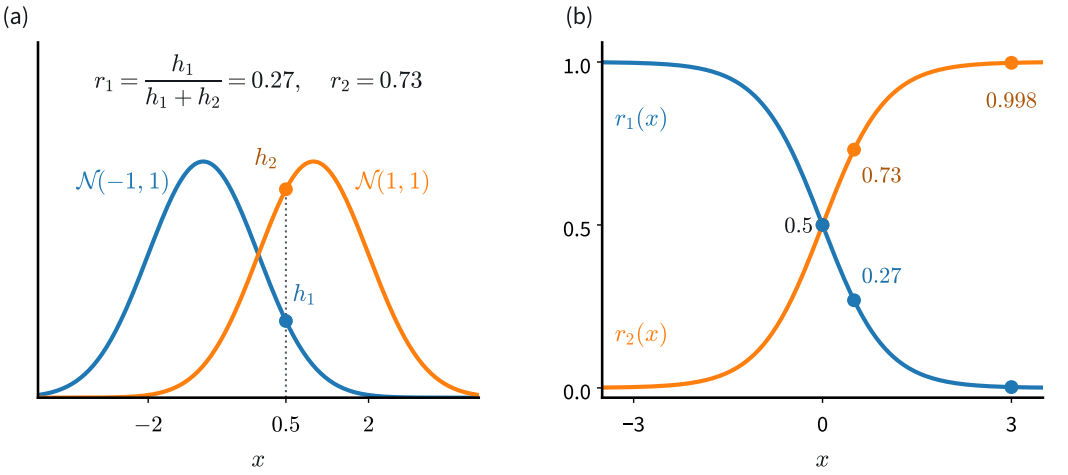


Figure 36.34: 等しい重みを持つ 2 つの成分 $\mathcal{N}(-1, 1)$ と $\mathcal{N}(1, 1)$ における、密度の高さの比としてのソフト割り当て。(a) $x = 0.5$ において各密度は高さ h_1 と h_2 を持ち、左側の成分は割合 $r_1 = h_1/(h_1 + h_2) = 0.27$ を受け取る。(b) 2 つの事後確率 $r_1(x)$ と $r_2(x)$。$x = 0$ で均等分割、0.27 対 0.73（$x = 0.5$）、そして右側の成分に対する 0.998（$x = 3$）となる。

36.7.2 エビデンス下界

この直感的な戦略を数学的保証のあるアルゴリズムへと昇華させるため、対数が和の内側へと戻るような対数尤度の下界を構成しよう。$q(z)$ を潜在変数上の**任意の**確率分布とし、ソフトな推測を表すものとする。周辺尤度の内側で $q(z)$ を掛けて割り、イェンセンの不等式（Jensen's inequality）を適用する。対数関数は凹関数であるため、$\log \mathbb{E}[\cdot] \geq \mathbb{E}[\log(\cdot)]$（35.3.3.1 節）が成り立つ。これにより次式を得る。

$$\log p(x \mid \theta) = \log \sum_z q(z) \, \frac{p(x, z \mid \theta)}{q(z)} \;\geq\; \sum_z q(z) \log \frac{p(x, z \mid \theta)}{q(z)} \;=:\; \mathcal{L}(q, \theta). \tag{36.92}$$

ここで和は、すべての z（ただし $q(z) > 0$）にわたってとる。仮に q が、同時分布が正となるある z に確率 0 を割り当てていたとしても、最初の等号が単に $\geq$ に変わるだけなので、この下界は依然として成り立つ。なお、周辺尤度 $p(x \mid \theta)$ は潜在変数 z に対して、エビデンス $P(X)$（36.6.4 節におけるベイズの定理の分母）がモデルパラメータに対して果たすのとまったく同じ役割を果たしている。すなわち、観測されない量をすべて和で消去した後の、観測データの確率である。そのため、この量も**エビデンス**と呼ばれ、右辺の $\mathcal{L}(q, \theta)$ は**エビデンス下界**（evidence lower bound、**ELBO**）と呼ばれる。比の対数を分解すると、ELBO は見通しのよい 2 つの項に分かれる。

$$\mathcal{L}(q, \theta) = \mathbb{E}_q\big[\log p(x, z \mid \theta)\big] + H(q).$$

第 1 項は期待**完全データ**対数尤度であり、扱いやすい目的関数をソフトな推測 q で平均したものである。第 2 項はその推測のエントロピーであり、θ には一切依存しない。さらに都合のよいことに、ELBO と真の対数尤度の間の厳密なギャップを測ることもできる。その正体は KL ダイバージェンスであり、EM と変分推論はどちらもこのただ 1 つの恒等式を土台としている。

命題（ELBO のギャップは KL ダイバージェンスである）。 任意の θ と、いかなる分布 q（ただし事後分布 $p(z \mid x, \theta)$ の台の外側に質量を持たないもの）に対しても、次式が成り立つ。

$$\log p(x \mid \theta) = \mathcal{L}(q, \theta) + D_{\mathrm{KL}}\big(q(z) \,\|\, p(z \mid x, \theta)\big). \tag{36.93}$$

したがって $\mathcal{L}(q,\theta) \le \log p(x \mid \theta)$ であり、等号は q が事後分布 $p(z \mid x, \theta)$ に一致するとき、かつそのときに限り成り立つ。

証明。 ベイズの定理 $p(z \mid x,\theta) = p(x, z \mid \theta)/p(x \mid \theta)$ を KL ダイバージェンスの定義に直接代入すると、次のようになる。

$$D_{\mathrm{KL}}\big(q \,\|\, p(z \mid x,\theta)\big) = \mathbb{E}_{q}\left[\log \frac{q(z)\, p(x \mid \theta)}{p(x, z \mid \theta)}\right] = \log p(x \mid \theta) - \mathcal{L}(q,\theta),$$

ここで $\log p(x \mid \theta)$ は z に依存しないため、そのまま期待値の外に出せる。式を整理すれば Equation 36.93 が得られる。ギブスの不等式（36.4.2 節）より、KL ダイバージェンスは常に非負であり、2 つの引数が一致するとき、かつそのときに限り 0 になる。$\blacksquare$

前項の具体例 $x = 0.5$ でこの恒等式を確かめてみよう。この例では、真の対数エビデンスは $\log p(x) = -1.42$ である。判断を留保した推測 $q = (\tfrac{1}{2}, \tfrac{1}{2})$ を用いると、ELBO の値は -1.54 となり、不足分の 0.12 nat は $(\tfrac{1}{2}, \tfrac{1}{2})$ から事後分布 $(0.27, 0.73)$ への厳密な KL ダイバージェンスに一致する。言い換えれば、下界が対数エビデンスに届かない分は、推測と真の事後分布とのずれそのものであり、それ以外の何ものでもない。Figure 36.35 に、θ を変化させたときのこの下界の振る舞いを示す。

36.7.3 期待値最大化法

ELBO の恒等式により、難しかった最適化問題は $\mathcal{L}(q,\theta)$ 上の**座標上昇法**（coordinate ascent）へと変換される。すなわち、一方の引数を固定したままもう一方に関して最大化し、役割を入れ替えて繰り返せばよい。独立な観測からなるデータセット $X = \{x_1, \dots, x_n\}$ が与えられたとき、この下界は各観測に個別に適用でき、データ点ごとに 1 つの推測 q_i を置いてデータセット全体にわたる和をとる形になる。**期待値最大化**（EM）アルゴリズムは、これら 2 つの座標を交互に更新する (Dempster et al. 1977)。

- **E ステップ：** $\theta^{(t)}$ を固定し、推測 q_i に関して最大化する。Equation 36.93 の等号成立条件より、各データ点に対する最適な選択はその厳密な事後分布 $q_i^{(t)}(z) = p(z \mid x_i, \theta^{(t)})$ である。これによりすべての KL ギャップが 0 になり、現在のパラメータにおいて下界は**タイト**になる。すなわち $\sum_i \mathcal{L}(q_i^{(t)}, \theta^{(t)}) = \log P(X \mid \theta^{(t)})$ が成り立つ。

- **M ステップ：** 推測 $q_i^{(t)}$ を固定し、パラメータ θ に関して最大化する。

$$\theta^{(t+1)} = \operatorname*{argmax}_{\theta} \sum_{i} \mathbb{E}_{q_i^{(t)}}\big[\log p(x_i, z \mid \theta)\big],$$

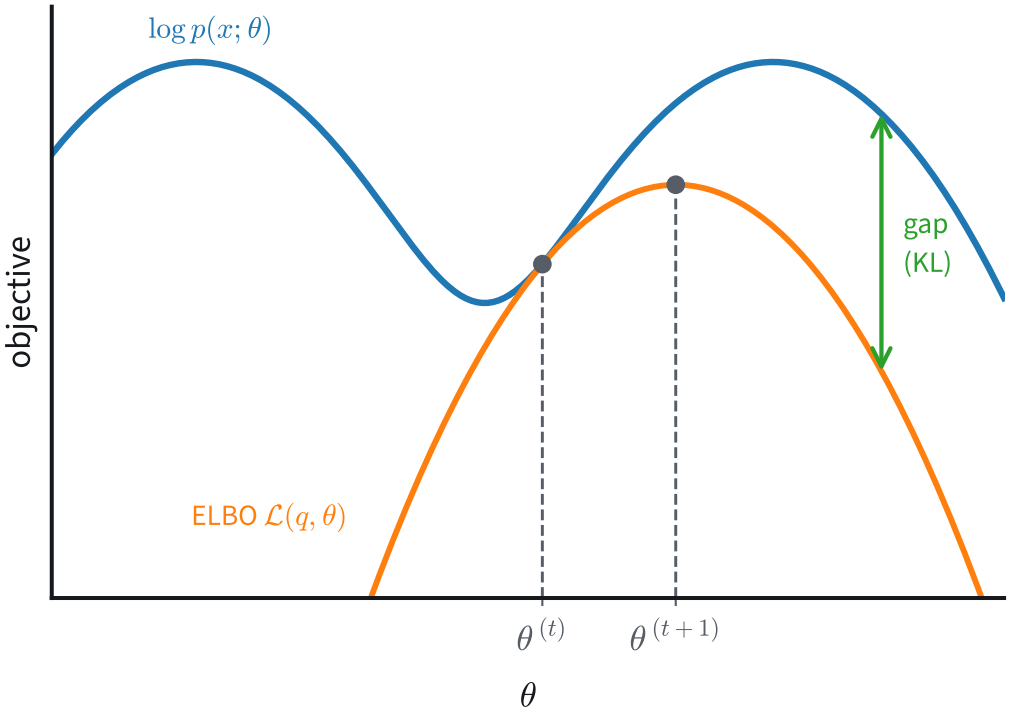


Figure 36.35: エビデンスとその下界。固定された推測 q に対し、ELBO $\mathcal{L}(q,\theta)$ はエビデンス $\log p(x \mid \theta)$ の下側にすべての θ で位置し、両曲線の縦方向のギャップは q から事後分布 $p(z \mid x,\theta)$ への KL ダイバージェンスとなる。EM では、E ステップで q を現在の反復点 $\theta^{(t)}$ における事後分布に設定し、ギャップを閉じてその点で下界をエビデンスに接しさせる。続く M ステップでは、下界をその頂点 $\theta^{(t+1)}$ まで登る。下界は $\theta^{(t)}$ でエビデンスに接し、かつ決してその上に出ないため、$\theta^{(t+1)}$ におけるエビデンスは少なくとも $\theta^{(t)}$ と同じ高さ以上になることが保証される。

ここでエントロピー項 $H(q_i^{(t)})$ は θ に依存しないため消える。こうして、欠けているオラクルのラベルをソフトな確率で代用した、扱いやすい完全データの目的関数へと戻ってくる。

なぜこの交互更新の手順を信頼できるのだろうか。それは Figure 36.35 の幾何学的構造により、1 ステップごとに真の対数尤度が必ず改善する（少なくとも悪化しない）ことが保証されているからである。

命題（EM は尤度を決して減少させない）。EM のすべての反復は $\log P(X \mid \theta^{(t+1)}) \geq \log P(X \mid \theta^{(t)})$ を満たす。

証明。 1 行で表せる 3 つの事実をつなぎ合わせる。

$$\log P(X \mid \theta^{(t+1)}) \;\geq\; \sum_i \mathcal{L}(q_i^{(t)}, \theta^{(t+1)}) \;\geq\; \sum_i \mathcal{L}(q_i^{(t)}, \theta^{(t)}) \;=\; \log P(X \mid \theta^{(t)}).$$

最初の不等号は新しいパラメータで評価した ELBO の下界 Equation 36.92 であり、2 番目の不等号は M ステップが θ に関して下界を最大化することから成り立ち、最後の等号は E ステップによって確保されたタイトさそのものである。$\blacksquare$

混合ガウスモデルでは、両方のステップが閉形式で求まる。E ステップでは**負担率**（responsibility）、すなわちデータ点 i が成分 k から生成された事後確率を、先ほど $x = 0.5$ に対して手計算したのと同様に計算する。

$$r_{ik} = \frac{\pi_k \, \mathcal{N}(x_i;\, \mu_k, \sigma_k^2)}{\sum_{j} \pi_j \, \mathcal{N}(x_i;\, \mu_j, \sigma_j^2)}.$$

続く M ステップでは、負担率で重み付けしたカウントと平均化によって各成分を当てはめ直す。$n_k = \sum_i r_{ik}$ を成分 k に割り当てられた実効的なデータ点数とおけば、次式が得られる。

$$\pi_k = \frac{n_k}{n}, \qquad \mu_k = \frac{1}{n_k} \sum_i r_{ik} \, x_i, \qquad \sigma_k^2 = \frac{1}{n_k} \sum_i r_{ik} \, (x_i - \mu_k)^2. \tag{36.94}$$

これらの式がいかに見慣れた形をしているかに注目してほしい。通常の完全データに対する推定量（混合比、標本平均、標本分散）を、単にソフト化しただけなのである。各データ点は 1 つの成分にハードな 1 票を投じる代わりに、負担率に応じて複数の成分へと票を分割する。実際、すべての成分が単一の固定された分散 σ^2 を共有するよう制約して $\sigma^2 \to 0$ の極限をとれば、負担率は最も近い平均への $0/1$ 割り当てへと硬化し、EM はそのまま K-means アルゴリズムに帰着する。

アルゴリズム全体は十数行のコードで書ける。300 点を $\mathcal{N}(-2, 0.8^2)$ から、200 点を $\mathcal{N}(1.5, 0.6^2)$ から生成し、1 次元の混合ガウスモデルに対する短い関数 `em`（K 個のガウス成分）を書いて、最終的に当てはめられたパラメータとともに各ステップの対数尤度を記録してみよう。

```
rng = np.random.default_rng()
x = np.concatenate([rng.normal(-2.0, 0.8, 300), rng.normal(1.5, 0.6, 200)])

def gauss(x, mu, sigma):
    return np.exp(-0.5 * ((x - mu) / sigma)**2) / (sigma * np.sqrt(2 * np.pi))
```

```
def em(x, pi, mu, sigma, steps):
    """EM for a 1-D Gaussian mixture: log-likelihood trace and final fit."""
    pi, mu, sigma = (np.array(v, dtype=float) for v in (pi, mu, sigma))
    lls = []
    for _ in range(steps):
        joint = pi * gauss(x[:, None], mu, sigma)  # (n, K)
        lls.append(np.log(joint.sum(axis=1)).sum())
        r = joint / joint.sum(axis=1, keepdims=True)  # E-step
        n_k = r.sum(axis=0)  # M-step
        pi, mu = n_k / len(x), (r * x[:, None]).sum(axis=0) / n_k
        sigma = np.sqrt((r * (x[:, None] - mu)**2).sum(axis=0) / n_k)
    return np.array(lls), pi, mu, sigma
```

先の命題は対数尤度が決して減少しないことを保証するが、EM がどれほど速く登るかについては何も語らない。そこで、かなり悪い初期推測（中心が ± 0.5 にあり大きく重なり合う単位分散の 2 成分）から出発して 30 回の反復を実行し、最初の 6 ステップにおける対数尤度の値と最終ステップの値を比べてみよう。

```
lls, pi, mu, sigma = em(x, [0.5, 0.5], [-0.5, 0.5], [1, 1], 30)
print('steps 0-5:', lls[:6].round(1), ' step 29:', lls[-1].round(1))
print(f'pi = {pi.round(2)}, mu = {mu.round(2)}, sigma = {sigma.round(2)}')
```

```
steps 0-5: [-1252.9  -984.7  -953.8  -901.   -868.5  -862.7]  step 29: -861.4
pi = [0.6 0.4], mu = [-1.94  1.5 ], sigma = [0.82 0.55]
```

対数尤度は約 400 nat 上昇し、およそ $-1{,}280$ からおよそ -860 へと達する（どちらの数値も無作為標本によって数十 nat ほど変動する）。驚くべきことに、改善のほぼすべてが直後に起きている。わずか 6 ステップ後には対数尤度は最終値から約 1 nat 以内に入っており、復元された重み・平均・スケールもすべて誤差約 0.2 以内で真の生成パラメータ $(0.6, 0.4)$、$(-2, 1.5)$、$(0.8, 0.6)$ の近くに落ち着く。

このようにきれいに収束する一方で、EM には知っておくべき 4 つの限界がある。1 つ目は無害なものである。K 個の成分のラベルを入れ替えると、あらゆる最大値に対して $K!$ 個の同一なコピーが生じる。この**ラベルスイッチング**は、ニューラルネットワークの隠れユニットを入れ替えたときに生じる識別不能性（non-identifiability）と同じものであり、これによってある解が別の解より劣るということはない。

2 つ目の限界はより重大である。EM は最初に到達した停留点（stationary point）で停止してしまい、そこが大域的最大値であるかどうかは問わない。対称な初期化はこの現象を鮮明に映し出す。次のセルを実行する前に少し手を止め、両方の成分がまったく同じ平均とスケールから出発したとき EM がどう振る舞うか予想してみてほしい。

```
lls2, pi2, mu2, sigma2 = em(x, [0.5, 0.5], [0, 0], [1, 1], 30)
print(f'log-likelihood {lls2[0]:.1f} -> {lls2[-1]:.1f}, '
      f'{lls[-1] - lls2[-1]:.1f} below the previous fit')
print(f'pi = {pi2.round(2)}, mu = {mu2.round(2)}, sigma = {sigma2.round(2)}')
print(f'mean of x = {x.mean():.2f}, standard deviation of x = {x.std():.2f}')
```

```
log-likelihood -1381.5 -> -1012.9, 151.5 below the previous fit
pi = [0.5 0.5], mu = [-0.57 -0.57], sigma = [1.83 1.83]
mean of x = -0.57, standard deviation of x = 1.83
```

2 つの成分は決して分離しない。まったく同じ状態で始まるため、あらゆるデータ点に対して同一の負担率を受け取り、M ステップでも互いに貼り付いたまま更新されるからである。その結果、EM はデータセット全体の平均と標準偏差を当てはめただけの単一ガウス分布の解に落ち着き、2 クラスタの解より 100 nat 以上も低い値にとどまる。EM のすべての不動点（fixed point）は尤度の停留点ではあるが、必ずしも最大点とは限らない。実務において複数のランダム初期化から EM を再実行するのはこのためである。ニューラルネットワークの層における隠れユニット間のまったく同じ対称性を破るのもランダム初期化である（4.4 節）。等しい初期値から出発したユニットは等しい勾配を受け取り、永久に等しいままにとどまる。

3 つ目の限界は連続な混合モデル特有の性質であり、尤度そのものが上に有界でないことに起因する。第 2 成分を最初のデータ点の真上（$\mu_2 = x_1$）に置き、重みを $\pi_2 = 1/n$、標準偏差を ϵ としよう。すると x_1 におけるその密度は $\frac{1}{n}(2\pi)^{-1/2}\epsilon^{-1}$ となる一方、第 1 成分は残りの $n-1$ 個の点をそのまま覆い続けるため、次式が得られる。

$$\ell(\theta) \le \log\left(n\sqrt{2\pi}\right) + \log\epsilon + \text{const},$$

ここで定数項は他の $n-1$ 個の点からの寄与をまとめたものである。$\epsilon \to 0$ とすると負の対数尤度は $-\infty$ へと発散するため、最尤法による大域的な「解」は、単一のデータ点の上に立つ無限に鋭いスパイクにすぎないことになる（Figure 36.36）。分散に下限を設けるか、分散に事前分布（prior）を置けば、このような退化したスパイクは取り除ける。初期点と同等以上に良いパラメータの集合がコンパクトになりさえすれば、EM のすべての集積点は正則化された問題の停留点になることが保証される (Wu 1983)。

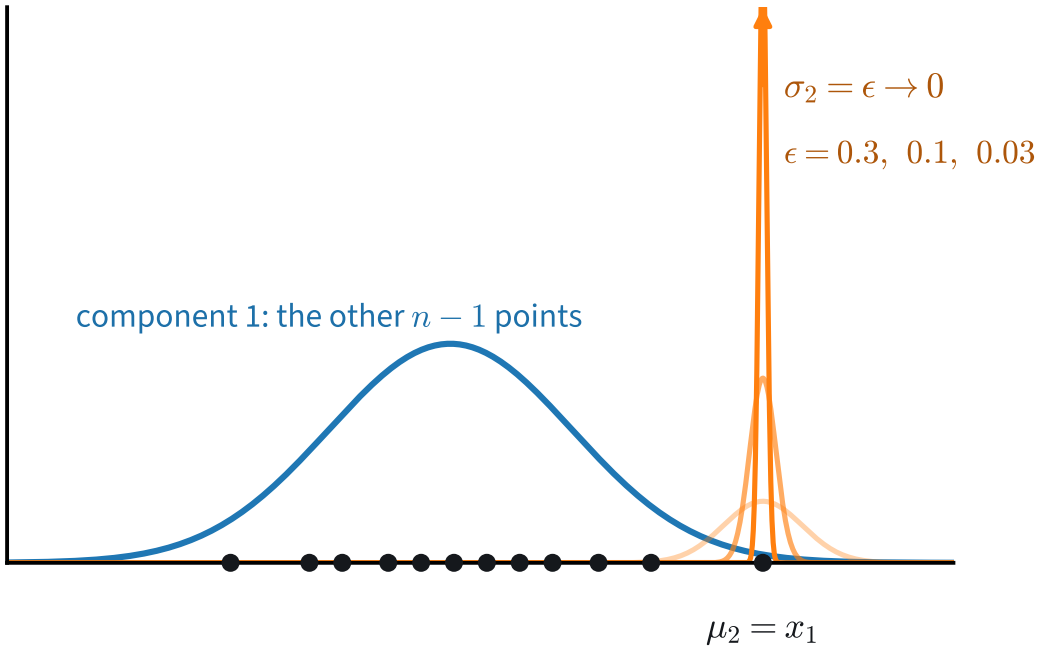


Figure 36.36: 混合ガウスモデルの尤度における退化した最大値。第 1 成分（青）がデータ点（黒点）のうち $n-1$ 個を覆い、第 2 成分（オレンジ）が残りの 1 点 $\mu_2 = x_1$ の上に標準偏差 $\sigma_2 = \epsilon$ で乗っている。ϵ が 0.3、0.1、0.03 へと縮むにつれ、x_1 における密度は $1/\epsilon$ に比例して増大し、それに伴って尤度も際限なく大きくなる。

4 つ目の限界は、EM の枠組みそのものを超えた先へとつながるものである。単調性の証明は、E ステップに**厳密な**事後分布を代入することに依存していた。そのため、近似的な E ステップで妥協せざるを得ない場合にはその保証が失われる。たとえば**変分オートエンコーダ** (Kingma and Welling 2014) は、深層非線形生成器が閉形式の事後分布を持たないため、ELBO という同じ目的関数を保ちつつも、データ点ごとの厳密な事後分布を、デコーダと同時に訓練されるエンコーダのニューラルネットワーク $q_\phi(z \mid x)$ に置き換える。そして厳密な EM ステップの代わりに、確率的勾配を用いて ELBO を登るのである。

ここに至って、下界はそれを導入したアルゴリズム自身の枠を超えて広がりを見せる。ELBO において隠れ変数 z がクラスタのラベルである必要はどこにもない。z の位置にモデルパラメータそのものを置けば、E ステップが求める事後分布はパラメータに関する事後分布となり、それこそ 36.6 節の末尾で求めたかった対象にほかならない。この事後分布も閉形式で得られることは稀であるため、続く 36.8 節では、それを近似しサンプリングするための実践的な手法を扱う。

36.7.4 まとめ

モデルが隠れ変数を含むとき、その周辺尤度は対数の内側に和を閉じ込めてしまい、本来なら単純なカウントと平均化で済むはずの問題を閉形式解のない最適化問題へと変えてしまう。隠れ変数に対する確率的推測 q を導入すると、扱いやすい完全データの目的関数が下界として復活する。すなわち、ELBO は q から真の潜在事後分布への KL ダイバージェンスの分だけ、対数エビデンスの下側に位置する。EM はそのギャップを閉じる操作（E ステップ）と、得られた下界をパラメータに関して最大化する操作（M ステップ）を交互に繰り返す。各反復で尤度が決して減少しないことは保証されるものの、EM は最適でない停留点で停止することがあり、連続な混合モデルでは退化したスパイクを防ぐために分散の下限や事前分布を必要とし、さらに単調性の保証を保つには厳密な事後分布を必要とする。

36.7.5 演習問題

1. [code] **EM の速度と欠損情報量。**●混合分布 $p(x \mid \pi) = \pi f_0(x) + (1-\pi) f_1(x)$ を考える。ただし成分 $f_0 = \mathcal{N}(\cdot\,; 0, 1)$ と $f_1 = \mathcal{N}(\cdot\,; \Delta, 1)$ は既知であり、混合重み π のみが未知であるとする。
   1. EM が重みを $\pi^{(t+1)} = M(\pi^{(t)})$ によって更新することを示せ。ただし $M(\pi) = \frac{1}{n}\sum_i r_i(\pi)$ であり、負担率は $r_i(\pi) = \pi f_0(x_i) / p(x_i \mid \pi)$ である。
   2. 不動点 $\hat\pi = M(\hat\pi)$ の近傍では、各反復で誤差が近似的に $M'(\hat\pi)$ 倍される。観測情報量を $I_{\text{obs}} = -\frac{d^2}{d\pi^2}\sum_i \log p(x_i \mid \pi)$（$\hat\pi$ で評価したもの）、ラベルが観測されたとした場合にラベルが持つ完全データ情報量を $I_{\text{com}} = n / (\hat\pi(1-\hat\pi))$ と定義する（36.6 節と比較せよ）。このとき次式を示せ。

$$M'(\hat\pi) = 1 - \frac{I_{\text{obs}}}{I_{\text{com}}}. \tag{36.95}$$

   両辺を負担率で表し、$\sum_i r_i(\hat\pi) = n\hat\pi$ を用いよ。Dempster et al. (1977) はこの結果の一般形を証明している。すなわち、EM の収束率は潜在変数によって隠された情報量の割合に支配される。
   3. $n = 10{,}000$ 個の点を $\pi = 0.3$ として各 $\Delta \in \{0.5, 1, 2, 4\}$ について抽出し、$\pi^{(0)} = 0.5$ から EM を実行して、連続する誤差の比の実測値を (b) からの予測と比較せよ。十分に長い実行から得た $\hat\pi$ を基準としたとき、各 Δ で誤差を 10^{-6} 未満にするには何回の反復が必要か。
   4. ラベルが観測された同じサイズのデータセットと比較したとき、比 $I_{\text{obs}}/I_{\text{com}}$ は $\hat\pi$ の標準誤差について何を意味するか。

2. **手計算による下界の確認**。本文で扱った単一の観測 $x = 0.5$ と、等しい重みを持つ 2 つの成分 $\mathcal{N}(-1, 1)$ および $\mathcal{N}(1, 1)$ を考える。
   1. $\log p(x)$ を計算し、あわせて ELBO Equation 36.92 を一様な推測 $q = (\frac{1}{2}, \frac{1}{2})$ と事後分布 $q = (0.27, 0.73)$ のそれぞれについて計算して、本文に挙げた値と一致することを確かめよ。
   2. KL ダイバージェンスを直接計算することで、一様な推測に対して Equation 36.93 が成り立つことを確かめよ。
   3. ハードな推測では q のすべてを右側の成分に置く。このときの ELBO とそのギャップを計算せよ。ハードな推測と M ステップを交互に繰り返すアルゴリズムでも ELBO は決して減少しないが、対数尤度が増加する保証がなくなるのはなぜか。
3. **極限としての K-means**。混合ガウスモデルの K 個すべての成分が等しい重みと単一の固定分散 σ^2 を共有し、平均のみを当てはめるものとする。
   1. 負担率 r_{ik} を、$-(x_i - \mu_k)^2/(2\sigma^2)$ の k にわたるソフトマックスとして表せ。温度の役割を果たすのはどの量か。また、最近傍の平均が一意である限り、$\sigma^2 \to 0$ のときその最近傍の平均に対する負担率が 1 に収束することを示せ。
   2. このとき M ステップ Equation 36.94 が各平均をそこに割り当てられたデータ点の平均値に設定すること、および両ステップを合わせると $\sum_i (x_i - \mu_{z_i})^2$（ただし z_i はデータ点 i に割り当てられた成分）を決して増加させないことを示せ。
   3. 本文の対称な初期状態では、両方の平均が同じ点から始まる。そのような初期状態から K-means はどのように動作するか。また、このことは実装において同値を解消する方法について何を示唆するか。

36.8 ベイズ計算

本書の「基礎」パート全体を通じて、最尤法（maximum likelihood）、あるいは正則化項を伴う事後分布（posterior distribution）のピークにおける最大事後確率推定（maximum a posteriori、MAP）によって、各モデルを単一のパラメータベクトルへと訓練してきた。36.6 節ではさらに一歩進め、その推定値にエラーバーを付した。それでもなお、訓練データをほぼ同様にうまく説明できるパラメータベクトルが他に数多く存在するにもかかわらず、予測は依然としてその単一のベクトルから得ていた。こうしたもっともらしいモデル同士はどの程度食い違うのか、そして食い違うときには何を予測すべきだろうか。

具体例として、入力が 0 から 4 の間にある 20 個のラベル付きデータ点に、切片と傾きを持つロジスティック回帰を当てはめる場合を考えよう。訓練データよりはるかに小さいテスト入力 -3 において、当てはめた MAP の直線はラベル 1 に約 1,000 分の 4 の確率を割り当てる。ところが、その 20 個の点にほぼ同じくらいよく当てはまる他のパラメータベクトルは、その入力において何桁も異なる予測を出す。データが与えられた下での各ベクトルのもっともらしさで重み付けして予測を平均すると、得られる確率はほぼ 5 倍大きくなる。一方、訓練データのちょうど内側にある入力 2 では、単一ベクトルによる予測と平均された予測は 0.01 以内で一致する。不正検知のアラートや医療診断のような稀な事象では、予測確率における 5 倍の差が意思決定を左右しうる。

この重み付き平均こそがベイズ予測であり、その計算はベイズの定理そのものと同じくらい古い問題である。事後分布はエビデンス、すなわちすべてのパラメータベクトルにわたる積分で割ることで得られるが、この積分が閉じた形で求まることは滅多にない。1774 年、Laplace は比率の事後分布をそのピークに一致するガウス分布で置き換えることで積分を回避した。1953 年には、Metropolis らが正規化定数（normalizing constant）の未知な分布に比例してランダムウォークに各状態を訪問させる方法を示した (Metropolis et al. 1953)。1990 年代までには、研究者たちはこの積分問題を最適化問題へと変換する手法も確立した。これらすべての手法を貫く統一的な仕掛けは 1 つである。いずれの手法も密度の比のみに依拠しており、そこでは未知の定数が約分されて消えるため、正規化定数を除いてしかわからない分布の上で平均を計算できるのである。

本節では、4 つの問いを順に立てることで 4 つのアプローチを展開する。第 1 に、事後分布をそのピークに一致するガウス分布で近似できるか。第 2 に、より単純な分布からサンプルを引き、各サンプルを補正できるか。第 3 に、事後確率質量に比例した時間を各領域で過ごすランダムウォークを作れるか。第 4 に、単純な分布を事後分布にできるだけ近づけるよう最適化できるか。最初の 3 つの手法はそれぞれ直前の手法の限界を克服するものであり、4 つ目の手法はサンプリングを最適化に置き換える。定数倍を除いてのみ既知の分布を扱ううえで、低次元の事後分布は最も見通しのよい舞台であるが、同じ発想は本書の随所で再び現れる。それぞれがどこで再登場するかは 36.8.5 節で整理する。これら 4 つの手法はいずれも、新しい入力 $\mathbf{x}_\star$ におけるラベル $y_\star$ の**事後予測**（posterior predictive）分布を次式により計算することを目的とする。

$$p(y_\star \mid \mathbf{x}_\star, \mathcal{D}) = \int p(y_\star \mid \mathbf{x}_\star, \theta)\, p(\theta \mid \mathcal{D})\, d\theta, \tag{36.96}$$

この式は、データ $\mathcal{D}$ が与えられた下での事後密度を重みとして、あらゆるパラメータベクトルの予測を平均したものである。

36.8.1 ベイズロジスティック回帰の事後分布

近似手法を公平に比較するには、閉じた形を持たないにもかかわらず厳密解と突き合わせて検証できる事後分布が必要である。コインに対するベータ事前分布が何回コイントスを繰り返してもベータ分布のままであるように、事後分布が事前分布と同じ分布族にとどまるとき、事前分布は尤度に対して**共役**（conjugate）であった（36.4 節）。ガウス事前分布はロジスティック尤度に対して共役ではないため、任意のパラメータベクトルにおいて未正規化事後分布の値は評価できても、それを解析的に積分することはできない。しかし、ここで通して用いる例にはパラメータが 2 つしかないため、細かい 2 次元グリッドを使えば、あらゆる近似の良し悪しを判定するための厳密な基準値が得られる。

36.8.1.1 モデルと未正規化事後分布

1 次元のロジスティック回帰では、各二値ラベル $y_i \in \{0, 1\}$ の確率は入力 x_i の線形関数によって次のように定まる。

$$P(y_i = 1 \mid x_i, \theta) = \mathrm{sigmoid}(\theta_0 + \theta_1 x_i) = \mathrm{sigmoid}(\mathbf{z}_i^\top \theta), \qquad \mathrm{sigmoid}(z) = \frac{1}{1 + e^{-z}},$$

ここで、$\mathbf{z}_i = (1, x_i)^\top$ である。予測確率が $\tfrac{1}{2}$ を横切るのは線形関数がゼロを横切る $x = -\theta_0/\theta_1$ においてであり、傾き θ_1 はその点での曲線の立ち上がりの急峻さを制御する。切片と傾きには独立なガウス事前分布（Gaussian prior）$\theta_j \sim \mathcal{N}(0, \tau^2)$（事前標準偏差は τ）を置く。ベイズの定理により、事後分布は n 個のラベルの尤度と事前分布の積に比例する。

$$p(\theta \mid \mathcal{D}) \propto \left[\prod_{i=1}^{n} \mathrm{sigmoid}(\mathbf{z}_i^\top \theta)^{y_i} \big(1 - \mathrm{sigmoid}(\mathbf{z}_i^\top \theta)\big)^{1 - y_i}\right] \exp\!\left(-\frac{\|\theta\|^2}{2\tau^2}\right). \tag{36.97}$$

角括弧内の積が尤度であり、指数関数の因子が事前分布である。ここに欠けている正規化定数は**エビデンス** $p(\mathcal{D}) = \int p(\mathcal{D}, \theta)\, d\theta$ であり、すべてのパラメータベクトルにわたる閉じた形を持たない積分である。右辺を**未正規化事後分布**（unnormalized posterior）と呼んで $\tilde{p}(\theta) = p(\mathcal{D}, \theta)$ と表せば、正規化された事後分布は単に $\tilde{p}/p(\mathcal{D})$ となる。

本節を通して用いるデータセットとして、$n = 20$ 個の入力を $[0, 4]$ から一様に抽出し、真のパラメータベクトル $\theta^\star = (-3, 1.5)$ からラベルを生成する。この確率曲線は $\frac{1}{2}$ を $x = 2$ で横切り、$\mathrm{sigmoid}(-3) \approx 0.05$（$x = 0$）から $\mathrm{sigmoid}(3) \approx 0.95$（$x = 4$）まで上昇する。事前分散は $\tau^2 = 4$ に設定する。データ点が 20 個であれば、本節で扱うあらゆる近似の効果がはっきりと浮かび上がるだけの広がりが事後分布に残る。実行のたびに新しいデータセットを生成するのではなく、具体的な基準値を引用できるよう、代表的な 20 個の点のサンプルを 1 つ固定する（以下で測定する事後分布の要約統計量に関して、1,500 個のシミュレーションデータセットの中から最も典型的なものを選んだ）。実行する各サンプラは毎回新しい乱数を使うため、サンプラの結果は典型的な範囲として報告する。コード中では、`log_joint` が $\log \tilde{p}$ を評価する際、大きなロジットでのオーバーフローを防ぐために `logaddexp(0, z)` を用いて $\log(1 + e^z)$ を計算している。

```
rng = np.random.default_rng()
n, prior_var = 20, 4.0                       # prior_var is tau^2
theta_star = np.array([-3.0, 1.5])

def sigmoid(z):
    return np.exp(-np.logaddexp(0.0, -z))

x = np.array([0.1, 0.3, 0.3, 0.4, 0.8, 1.4, 1.5, 1.6, 1.8, 2.1,
              2.2, 2.3, 2.6, 2.7, 3.0, 3.1, 3.4, 3.5, 3.6, 3.7])
y = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1])
Z = np.column_stack([np.ones(n), x])         # rows z_i = (1, x_i)

def log_joint(theta, Z=Z, y=y, prior_var=prior_var):
    """log p(D, theta) up to a constant; theta has final dimension 2."""
    logits = np.einsum('...d,nd->...n', theta, Z)
    log_lik = (y * logits - np.logaddexp(0.0, logits)).sum(axis=-1)
    return log_lik - 0.5 * (theta**2).sum(axis=-1) / prior_var

print(f'{n} points, {y.sum()} of them labeled 1')
```

```
20 points, 10 of them labeled 1
```

36.8.1.2 勾配とヘッセ行列

以下のいくつかの手法は $\log \tilde{p}$ の傾きと曲率に依拠するため、最初に一度だけ導出しておくと見通しがよい。各対数尤度項 $y_i \, \mathbf{z}_i^\top \theta - \log(1 + e^{\mathbf{z}_i^\top \theta})$ を微分すると $(y_i - p_i)\, \mathbf{z}_i$（ただし $p_i = \mathrm{sigmoid}(\mathbf{z}_i^\top \theta)$）が得られ、ガウス事前分布からは $-\theta/\tau^2$ が加わる。入力行 $\mathbf{z}_i^\top$ を縦に並べて $n \times 2$ 行列 $\mathbf{Z}$ とすれば、勾配 $\nabla \log \tilde{p}(\theta) = \mathbf{Z}^\top(\mathbf{y} - \mathbf{p}) - \theta/\tau^2$ が得られる。その幾何学的な意味は明快である。各訓練点は θ を $\mathbf{z}_i$ の方向に予測残差 $y_i - p_i$ に比例して押し動かし、事前分布はパラメータを原点へと穏やかに引き戻す。もう一度微分すると、次の負のヘッセ行列が得られる。

$$\mathbf{H}(\theta) = -\nabla^2 \log \tilde{p}(\theta) = \mathbf{Z}^\top \, \mathrm{diag}\big(p_i(1 - p_i)\big)\mathbf{Z} + \tau^{-2}\mathbf{I}. \tag{36.98}$$

ここでデータの項は、入力の重み付きグラム行列（Gram matrix）である。各重み $p_i(1-p_i) \le \frac{1}{4}$ は $p_i = \frac{1}{2}$ で最大となるため、モデルの不確実性が最も高い点ほど曲率に大きく寄与する。事前項 $\tau^{-2}\mathbf{I}$ を加えることで、すべての固有値が少なくとも τ^{-2} だけ押し上げられるため、$\mathbf{H}(\theta)$ は至るところで狭義正定値となる。したがって対数事後密度は狭義凹関数であり、唯一の最大値を持

つ。たとえ訓練データが線形分離可能であっても、対数尤度が正になることはない一方で二次形式の事前ペナルティはあらゆる方向へ際限なく減少するため、その最大値は有限にとどまる。

```
def grad_log_joint(theta, Z=Z, y=y, prior_var=prior_var):
    """Gradient of log p(D, theta); one row per row of theta."""
    theta = np.atleast_2d(theta)
    return (y - sigmoid(theta @ Z.T)) @ Z - theta / prior_var

def neg_hessian(theta, Z=Z, y=y, prior_var=prior_var):
    """H(theta) = Z^T diag(p (1 - p)) Z + I / tau^2."""
    p = sigmoid(Z @ theta)
    return Z.T @ (Z * (p * (1 - p))[:, None]) + np.eye(2) / prior_var
```

パラメータが 2 つしかなければ、これらの導関数を手で書き下すのは造作もない。より大規模なモデルでは、$\log \tilde{p}$ のみを実装し、勾配とヘッセ行列は自動微分に任せることになる。

なお、本節のすべての手法は、値・勾配・ヘッセ行列のいずれを用いるにせよ、もっぱら $\tilde{p}$ のみを通じて事後分布にアクセスする。

36.8.1.3 事後最頻値とラプラス近似

最初の最も計算コストの低い近似は、すでに実行方法を知っている最適化の上に直接構築される。事後分布のピークは MAP 推定値、すなわち 36.6 節で扱ったペナルティ付きの当てはめであり、ニュートン法 $\theta \leftarrow \theta + \mathbf{H}(\theta)^{-1}\nabla \log \tilde{p}(\theta)$ によってわずか数回の反復でそこに到達できる。あとはヘッセ行列をもう 1 回評価するだけで、その点推定（point estimate）はガウス分布へと変わる。

具体的には、事後最頻値（posterior mode）θ_{MAP} を中心とするガウス分布を当てはめる。$\log \tilde{p}$ を θ_{MAP} の周りで 2 次までテイラー展開（Taylor expansion）すると、$\mathbf{H} = \mathbf{H}(\theta_{\mathrm{MAP}})$ として次式が得られる。

$$\log \tilde{p}(\theta) \approx \log \tilde{p}(\theta_{\mathrm{MAP}}) - \tfrac{1}{2}(\theta - \theta_{\mathrm{MAP}})^{\top}\mathbf{H}(\theta - \theta_{\mathrm{MAP}}). \tag{36.99}$$

最頻値では勾配が消えるため、1 次の項は現れない。加法的な定数を除けば、右辺は平均 θ_{MAP}、共分散 $\mathbf{H}^{-1}$ を持つガウス分布の対数密度にほかならない。したがって、指数をとることで次の**ラプラス近似**（Laplace approximation）が得られる。

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta_{\mathrm{MAP}}, \mathbf{H}^{-1}). \tag{36.100}$$

この近似が適切に定義されるためには、$\log \tilde{p}$ が内点にある最頻値で 2 回微分可能であり、負のヘッセ行列 $\mathbf{H}$ が正定値でなければならないが、ここではその両方が成り立っている。直観的には、鋭いピークは曲率が大きいため、分散の小さい幅の狭いガウス分布を生み出す。このガウス分布が一致させるのは事後分布のピークの位置と局所的な曲率だけであり、それ以外の性質は一切反映しない。

1 次元の例を見れば、この振る舞いは一目瞭然である。歪んだ事後分布 $\mathrm{Gamma}(4, 1)$ を考えよう。その未正規化密度は $\theta^3 e^{-\theta}$（$\theta > 0$）である。対数密度 $3\log\theta - \theta$ の導関数は $3/\theta - 1$ であり、最頻値 $\theta = 3$ でゼロとなり、その点での負の二階導関数は $3/\theta^2 = \tfrac{1}{3}$ となる。したがってラプラス近似は $\mathcal{N}(3, 3)$ となり、標準偏差は $\sqrt{3} \approx 1.73$ である。厳密な事後分布は平均 4、分散 4 を持つため、このガウス分布は左に寄りすぎているうえに幅も狭すぎ、さらには質量の $\Phi(-\sqrt{3}) \approx 4.2\%$ を、真の密度がゼロである $\theta = 0$ 未満に割り当ててしまう（Figure 36.37）。

一様事前分布の下で表が 9 回、裏が 4 回出たコインに同じ計算を適用すると、36.6.3.2 節で求めた最尤推定量（maximum likelihood estimate、MLE）$9/13$ とその標準誤差 0.128 が復元される。実際、平坦な事前分布の下では、ラプラス近似は単に MLE とその漸近的なエラーバーを組み合わせたものにほかならない（演習問題 3）。

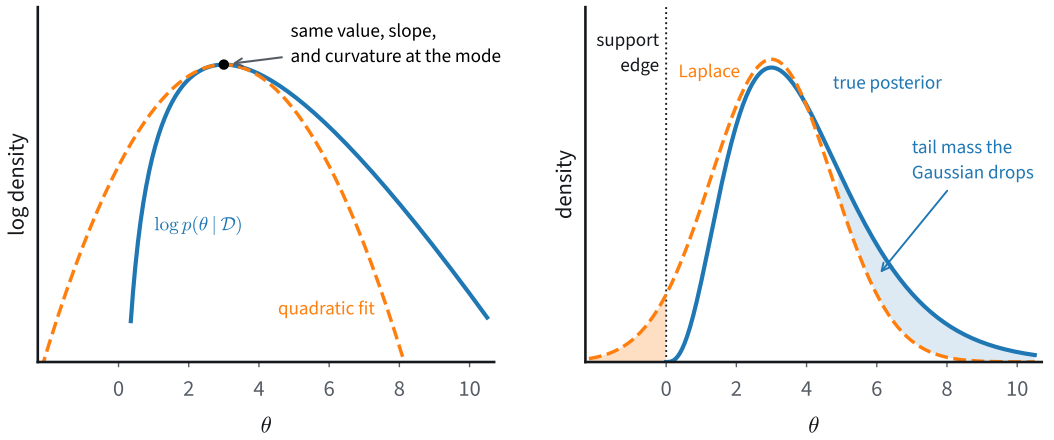


Figure 36.37: 歪んだ 1 次元事後分布である最頻値 3 の Gamma(4,1) 密度に対するラプラス近似。左：対数密度への二次近似は、最頻値における値・傾き・曲率のみに一致し、それ以外には一致しない。右：指数をとると、ラプラスガウス分布 $\mathcal{N}(3, 3)$ はピークには一致するものの歪みを捉え損ね、質量の 4.2%（オレンジ色の網掛け）を事後分布がゼロであるサポート境界 $\theta = 0$ 未満に割り当てる一方で右裾を過小評価している。真の平均は 4 である。

未知のエビデンスがこの計算に一切現れないことに注目しよう。$\log p(\mathcal{D})$ は $\log p(\theta \mid \mathcal{D})$ における加法的な定数であるため、微分すると消えるからである。さらに都合のよいことに、Equation 36.99 の右辺の指数を積分すれば、エビデンス自体の推定値 $p(\mathcal{D}) \approx \tilde{p}(\theta_{\text{MAP}})(2\pi)^{d/2} \det(\mathbf{H})^{-1/2}$（$d$ 次元の場合）まで得られる（演習問題 6）。

ラプラス近似は単一のピークにおける局所的な導関数しか用いないため、歪度（skewness）、重い裾、パラメータの境界、他の最頻値（モード）を捉えることができず、分布の中心も平均ではなく最頻値に置かれる。スケーリングも課題となる。d 個のパラメータを持つニューラルネットワークでは $\mathbf{H}$ の要素数が d^2 個に上るため、実応用では対角成分のみを残すか、近似を最終層のみに制限するのが一般的である。

コード上では、原点から出発する 10 回のニュートンステップで最頻値が求まり、1 回の逆行列計算でラプラス共分散が得られる。

```
theta_map = np.zeros(2)
for _ in range(10):                                  # Newton's method
    theta_map += np.linalg.solve(neg_hessian(theta_map),
                                 grad_log_joint(theta_map)[0])
laplace_cov = np.linalg.inv(neg_hessian(theta_map))
laplace_sd = np.sqrt(np.diag(laplace_cov))
laplace_rho = laplace_cov[0, 1] / (laplace_sd[0] * laplace_sd[1])
print('MAP                :', theta_map.round(3))
print('Laplace sd         :', laplace_sd.round(3))
print('Laplace correlation:', round(laplace_rho, 3))
```

```
MAP                : [-2.138  1.117]
Laplace sd         : [1.004 0.457]
Laplace correlation: -0.861
```

事後最頻値は $(-2.14, 1.12)$ にあり、データ生成パラメータ $(-3, 1.5)$ よりも緩やかな傾きの直線となっている。ラプラス標準偏差は切片で 1.00、傾きで 0.46 であり、真の値は各座標において標準偏差の 1 倍弱だけ離れた位置にある。また、-0.86 という強い負の相関にも注目しよう。傾きが急な直線ほど、同じ点群を通過するためにはより低い位置から出発しなければならないからである。

36.8.1.4 グリッドによる基準値

ラプラス近似が実際どれほど正確かを確かめるには、ベンチマークとして厳密な事後分布が必要である。パラメータが 2 つしかなければ、細かい $m \times m$ のグリッド上で $\log \tilde{p}$ を評価し、数値安定性のために最大値を引いてから指数をとり、重みの和が 1 になるよう正規化できる。d 次元ではこのような格子（lattice）に m^d 回の評価が必要になるため、グリッド積分はあくまで低次元専用の基準手法である。2 次元であっても領域をどこに置くかは決める必要がある。ここでは θ_{MAP} を中心として各方向にラプラス標準偏差の 7 倍の範囲をとることで、領域外に無視できるほどの裾しか残らないようにする。

```
k, m = 7, 241                   # half-width in Laplace sds, points per axis
grid_axes = [np.linspace(c - k * s, c + k * s, m)
             for c, s in zip(theta_map, laplace_sd)]
B0, B1 = np.meshgrid(*grid_axes, indexing='ij')
grid_theta = np.stack([B0.ravel(), B1.ravel()], axis=1)
grid_logp = log_joint(grid_theta)
grid_weight = np.exp(grid_logp - grid_logp.max())
grid_weight /= grid_weight.sum()
post_mean = grid_weight @ grid_theta
centered = grid_theta - post_mean
post_cov = (centered * grid_weight[:, None]).T @ centered
post_sd = np.sqrt(np.diag(post_cov))
post_rho = post_cov[0, 1] / (post_sd[0] * post_sd[1])
print('grid mean       :', post_mean.round(3), '  MAP:', theta_map.round(3))
print('grid sd         :', post_sd.round(3), '  Laplace:', laplace_sd.round(3))
print('grid correlation:', round(post_rho, 3))
print('(mean - MAP)/sd :', ((post_mean - theta_map) / post_sd).round(2))
```

```
grid mean       : [-2.347  1.237]   MAP: [-2.138  1.117]
grid sd         : [1.042 0.48 ]   Laplace: [1.004 0.457]
grid correlation: -0.858
(mean - MAP)/sd : [-0.2   0.25]
```

グリッド計算によって、ラプラス近似が見落としていた歪度が明らかになる。傾きの事後平均（posterior mean）1.24 は MAP 推定値 1.12 より事後標準偏差の 4 分の 1 だけ上にあり、切片の平均 -2.35 は MAP 推定値より標準偏差の 5 分の 1 だけ下にある。ラプラス標準偏差は 4〜5% 狭すぎ、相関は 2 桁まで一致している。こうした幅の誤差は控えめなものであるが、より重大な食い違いは、事後分布が急な傾きの直線側へと歪んでいるために平均と最頻値が一致しないことである。

36.8.1.5 プラグイン予測と事後予測

これで冒頭の問いに直接答える準備が整った。**プラグイン**予測 $\mathrm{sigmoid}(\mathbf{z}_\star^\top \theta_{\mathrm{MAP}})$ は最頻値のみに依拠するのに対し、事後予測 Equation 36.96 は事後分布の重みを使ってグリッド全体で $\mathrm{sigmoid}(\mathbf{z}_\star^\top \theta)$ を平均する。訓練データの下側・内側・上側に位置する 3 つのテスト入力 $x_\star \in \{-3, 2, 7\}$ において両者の予測を、もっともらしいパラメータベクトルにわたる予測値の中央 90% を含む区間とともに比較してみよう。

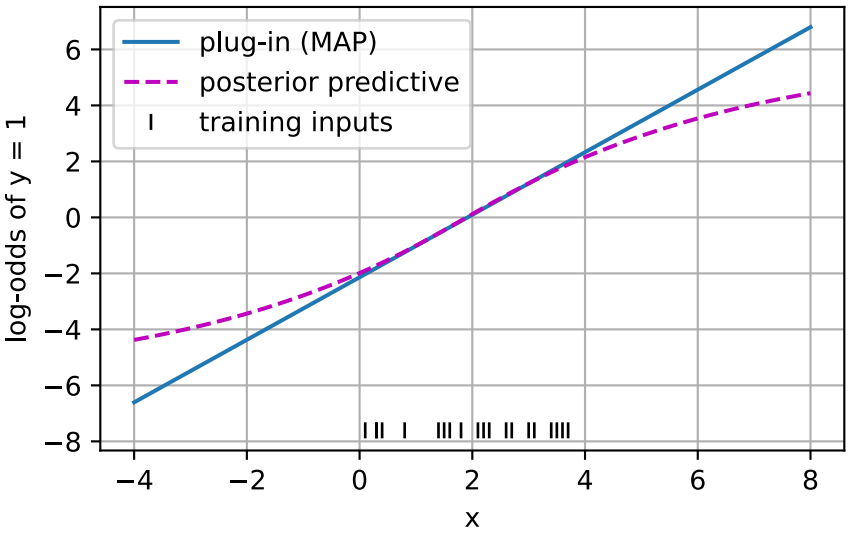
```
x_new = np.array([-3.0, 2.0, 7.0])
Z_new = np.column_stack([np.ones(len(x_new)), x_new])
grid_probs = sigmoid(grid_theta @ Z_new.T)     # each grid vector's answer
plug_in = sigmoid(Z_new @ theta_map)
predictive = grid_weight @ grid_probs
for j, x0 in enumerate(x_new):
    order = np.argsort(grid_probs[:, j])
    cdf = np.cumsum(grid_weight[order])
    lo, hi = grid_probs[order[np.searchsorted(cdf, [0.05, 0.95])], j]
    print(f'x={x0:+.0f}: plug-in {plug_in[j]:.4f}, predictive '
          f'{predictive[j]:.4f}, middle 90%: [{lo:.2g}, {hi:.4f}]')
a = grid_theta @ Z_new[0]                      # logits at x = -3
mu = grid_weight @ a
s2 = grid_weight @ (a - mu)**2
print(f'x=-3, log scale: plug-in {np.log(plug_in[0]):.2f}, '
      f'mean logit {mu:.2f}, plus s^2/2 {mu + s2 / 2:.2f},')
print(f'                 log E[e^a] {np.log(grid_weight @ np.exp(a)):.2f}, '
      f'predictive {np.log(predictive[0]):.2f}')
```

```
x=-3: plug-in 0.0041, predictive 0.0187, middle 90%: [3.8e-05, 0.0887]
x=+2: plug-in 0.5238, predictive 0.5292, middle 90%: [0.32, 0.7364]
x=+7: plug-in 0.9966, predictive 0.9826, middle 90%: [0.92, 1.0000]
x=-3, log scale: plug-in -5.49, mean logit -6.06, plus s^2/2 -3.19,
                 log E[e^a] -3.66, predictive -3.98
```

訓練データの範囲内にある $x_\star = 2$ では、2 つの予測（0.524 と 0.529）は 0.01 以内で一致し、もっともらしいパラメータベクトルによる予測も 0.32 から 0.74 という穏やかな範囲に収まる。データから大きく外れた $x_\star = -3$ では、もっともらしい予測の中央 90% は 4×10^{-5} から 0.09 まで 3 桁以上にわたって広がり、事後予測（0.019）はプラグイン値（0.0041）の 4.6 倍になる。$x_\star = 7$ でも、ラベル 0 の確率に対して同じ増幅が生じる（0.017 対 0.003）。これほど 0 や 1 に近い確率は対数オッズ（log odds）スケール $\log \frac{P}{1-P}$ で比較するのが最も見やすいため、下端に訓練入力の位置を記したうえで、より広い範囲にわたって両方の曲線をプロットしよう。

```
def log_odds(p):
    return np.log(p) - np.log1p(-p)

xs = np.linspace(-4, 8, 121)
Z_xs = np.column_stack([np.ones_like(xs), xs])
plug_xs = Z_xs @ theta_map                     # log-odds of the plug-in
pred_xs = log_odds(grid_weight @ sigmoid(grid_theta @ Z_xs.T))
rug = np.full(n, min(plug_xs.min(), pred_xs.min()) - 1)
d2l.plot([xs, xs, x], [plug_xs, pred_xs, rug],
         xlabel='x', ylabel='log-odds of y = 1', fmts=('-', 'm--', 'k|'),
         legend=['plug-in (MAP)', 'posterior predictive', 'training inputs'],
```

```
        figsize=(5, 3))
```



対数オッズスケールでは、プラグイン予測は単なる直線 $\theta_0 + \theta_1 x$ になる。$[0, 4]$ の内側では事後予測はこの直線にぴたりと沿うが、訓練データの外側ではゼロに向かって折れ曲がる。この乖離は 2 つの競合する効果によって生じる。第 1 に、事後平均は MAP 推定値よりも急な傾きを持つため、極端な予測を $\frac{1}{2}$ から**遠ざける**方向に押しやる。第 2 に、広がりを持つロジットの上でシグモイドを平均することは、予測を $\frac{1}{2}$ に**近づける**方向に引き寄せる。データから離れた領域では、傾きの緩やかなもっともらしいパラメータベクトルが典型的なベクトルよりもはるかに大きな確率を生み出し、その大きな確率が平均を支配するからである。

どちらの効果がどれだけ勝るのだろうか。簡単な概算によって、平均化の効果の大きさを見積もることができる。確率が小さい領域では $\text{sigmoid}(a) \approx e^a$ であるから、予測確率はおおよそ e^a の事後平均（ただしロジットは $a = \mathbf{z}_\star^\top \theta$）となる。もし a が平均 μ、分散 s^2 のガウス分布に従うなら、積率母関数（モーメント母関数）より $\mathbb{E}[e^a] = e^{\mu + s^2/2}$ となり、対数確率（log probability）は平均ロジットにおける値より $s^2/2$ だけ押し上げられる。$x_\star = -3$ ではグリッド計算より $s^2/2 = 2.87$ となるため、$e^{2.87} \approx 18$ 倍の増加が予測されるはずである。ところが実際の倍率は 4.6 倍、対数スケールではわずか 1.51 の上昇にとどまる。消えた 1.36 nat はどこへ行ったのだろうか。上のコードの最後の 2 行は、プラグイン予測から事後予測までを 1 段階ずつたどっている。

$x_\star = -3$ における段階	対数確率	変化量
プラグイン：最頻値におけるロジット	-5.49	
ロジットの事後平均	-6.06	-0.57
ガウスロジットと同様に $s^2/2$ を加算	-3.19	$+2.87$
ロジットの実際の分布上で e^a を平均	-3.66	-0.47
$\text{sigmoid}(a)$ を平均（e^a の代わり）：事後予測	-3.98	-0.32

3 つの下向き補正によって、2.87 と 1.51 の差はすべて説明がつく。第 1 に、$s^2/2$ の押し上げはプラグインのロジットではなく平均ロジットを起点としており、傾きがより急な事後平均は 0.57 だけ低い位置にある。第 2 に、ロジットはガウス分布ではなく歪んでいる。負の大きな値の方向へ長い裾を持ち、e^a の平均が質量のほぼすべてを得るゼロ方向への裾は短いため、さらに 0.47 削られる。第 3 に、平均を支配するパラメータベクトルは $x_\star = -3$ においてすでに微小とはいえない確率を持つため、シグモイドが e^a より下に曲がり、最後の 0.32 を差し引く。以上を合わせると、$2.87 - 0.57 - 0.47 - 0.32 = 1.51$ となる。

したがってガウス分布による計算は、厳密な法則というよりは有用な指針であり、変化の方向、大まかなスケール、および $x_\star$ に対する漸近的な依存性を正しく捉えている。具体的には、ロジットの分散 $\mathrm{Var}(\theta_0) + 2x_\star \mathrm{Cov}(\theta_0, \theta_1) + x_\star^2 \mathrm{Var}(\theta_1)$ はデータからの距離に対して二次関数的に増大するのに対し、平均のシフトは線形にしか増大しない。訓練入力から遠く離れると必然的に分散の項が勝り、データの内側では両方の効果とも無視できるほど小さいままにとどまる（演習問題 4）。

なお、このより大きな予測確率が、データを生成した真の値に必ずしも近いわけではないことに注意しよう。真の直線は $\mathrm{sigmoid}(-3 - 1.5 \cdot 3) = \mathrm{sigmoid}(-7.5) \approx 0.0006$ を $x_\star = -3$ において与えるが、20 個の点では傾きを確定できず事前分布がそれをゼロに向かって縮小（shrinkage）させるため、この値はプラグイン予測と事後予測のどちらよりも小さい。それでもこの真の確率は、典型的なデータセットで期待されるとおり、90% 事後区間の内側に余裕をもって収まっている。事後予測は真のパラメータを知っていると主張するのではなく、事前分布と 20 個の観測に整合するすべてのパラメータベクトルにわたって平均した $y_\star = 1$ の厳密な確率を与えるのである。本節の残りの部分では、パラメータ空間が大きすぎてグリッドが使えないときに、このような平均をどう計算するかを問う。

36.8.2 棄却サンプリングと重要度サンプリング

ラプラス近似は高速である反面、最頻値を中心としており歪度を無視してしまう。もし厳密な事後分布から直接サンプルを抽出できれば、その経験平均によって任意の事後期待値をすっきり推定でき、歪度も裾も自動的に捉えられるはずである。本節で扱う 2 つの手法は、扱いやすい**提案**分布（proposal distribution）q からのサンプルを、比 $\tilde{p}/q$ のみに依拠して事後分布からの厳密なサンプルまたは重み付きサンプルへと変換する。

36.8.2.1 棄却サンプリング

最も単純な厳密サンプラである棄却サンプリング（rejection sampling）は、q から候補を引き、一部を残して残りを捨てる。サンプリングと評価が容易な提案分布 q に加え、定数 M を $M q(\theta) \geq \tilde{p}(\theta)$ がすべての θ で成り立つように選び、未正規化事後分布の上を覆う包絡線（エンベロープ）を作る。各サンプル $\theta \sim q$ に対し、次の確率でこれを受理して残す。

$$a(\theta) = \frac{\tilde{p}(\theta)}{M\, q(\theta)}. \tag{36.101}$$

幾何学的には、この受理確率は提案された点における包絡線の高さに対する目標分布の高さの比である。受理されたサンプルはすべて $p(\theta \mid \mathcal{D})$ からの厳密なサンプルとなり、全体の受理率は $p(\mathcal{D})/M$ となる。この確認は 2 行で済む。提案された点が微小領域 $d\theta$（θ の周囲）に入る確率は $q(\theta)\, d\theta$ であり、それが受理される確率は $a(\theta)$ であるから、その領域に入って残される同時確率（joint probability）は $\tilde{p}(\theta)\, d\theta / M$ となる。すべての θ にわたって積分すると周辺受理確率 $\int \tilde{p}/M = p(\mathcal{D})/M$ が得られ、同時密度をこの周辺確率で割れば、残されたサンプルの条件付き密度 $\tilde{p}(\theta)/p(\mathcal{D}) = p(\theta \mid \mathcal{D})$ が得られる。

重要なことに、未知のエビデンスは受理率にのみ現れ、受理・棄却の判定規則そのものには一切現れない。提案分布 q として事前分布を選べば、$\tilde{p}/q$ は単なる尤度となり、M はその最大値となる。すなわち、事前分布からパラメータベクトルを引き、最尤法における最大尤度でその尤度を割った確率で各サンプルを残せばよい。

データの蓄積に伴って棄却サンプリングがどれほど急速に劣化するかを見るため、一様事前分布の下で表が 9 回、裏が 4 回出たコインを考えよう。尤度 $\theta^9(1-\theta)^4$ は $\theta = 9/13$ でピークに達して高さ $M = (9/13)^9(4/13)^4 \approx 3.3 \times 10^{-4}$ をとり、一方エビデンスは $p(\mathcal{D}) = \int_0^1 \theta^9(1-\theta)^4\,d\theta = 9!\,4!/14! \approx 1.0 \times 10^{-4}$ であるから、受理率は 0.305 となる。データを 10 倍（表 90 回、裏 40 回）にすると受理率は 0.10 に落ち、100 倍にすると 0.03 まで下がる。事前分布が固定されたまま事後分布の幅が $1/\sqrt{n}$ のオーダーで縮むため、受理率は 1 次元では $1/\sqrt{n}$、また $n^{-d/2}$（d 次元の場合）のオーダーで減衰するのである。一様事前分布からの 100,000 個のサンプルによる簡単なシミュレーションでこの計算を確認できる。

```
N = 100_000
theta_coin = rng.random(N)                    # draws from the uniform prior
lik = theta_coin**9 * (1 - theta_coin)**4     # likelihood of 9 heads, 4 tails
M = (9 / 13)**9 * (4 / 13)**4                 # its maximum, at theta = 9/13
kept = theta_coin[rng.random(N) < lik / M]
print(f'acceptance {len(kept) / N:.3f}, mean {kept.mean():.3f}, '
      f'sd {kept.std():.3f}')
```

```
acceptance 0.306, mean 0.666, sd 0.117
```

実用的には 2 つの深刻な障壁が棄却サンプリングを制約する。第 1 に、妥当な上界 M を知っている必要があるが、単純なモデル以外でそれが得られることは稀である。第 2 に、パラメータあたりの観測数を一定に保ったとしても、受理率は次元 d に対して少なくとも指数関数的に減衰する。

36.8.2.2 重要度重み

棄却サンプリングは、せっかく生成したサンプルの大半を捨ててしまう。比 $w = \tilde{p}/q$ を使ってコイントスをして候補を捨てる代わりに、すべてのサンプルを残して w で重み付けすることができる。なぜこれがうまくいくのかを見るため、任意の事後期待値を 2 つの積分の比として表し、両方の積分の中で q を掛けて割ってみよう。

$$\begin{aligned}\mathbb{E}_{p(\cdot\mid\mathcal{D})}[h(\theta)] &= \frac{\int h\,\tilde{p}\,d\theta}{\int \tilde{p}\,d\theta} = \frac{\int h\,\frac{\tilde{p}}{q}\,q\,d\theta}{\int \frac{\tilde{p}}{q}\,q\,d\theta} \\ &= \frac{\mathbb{E}_q[w(\theta)h(\theta)]}{\mathbb{E}_q[w(\theta)]}, \qquad w(\theta) = \frac{\tilde{p}(\theta)}{q(\theta)}.\end{aligned} \tag{36.102}$$

分母 $\mathbb{E}_q[w]$ は未知のエビデンス $p(\mathcal{D}) = \int \tilde{p}$ に等しいが、同じ正規化定数が分子と分母の両方を定数倍するため、これを明示的に計算する必要はまったくない。2 つの期待値をサンプル $\theta_1, \ldots, \theta_N \sim q$ にわたる標本平均で置き換えると、**自己正規化重要度サンプリング**（self-normalized importance sampling）$\sum_s w_s h(\theta_s) / \sum_s w_s$ が得られる。大数の法則（law of large numbers）により、この比は $N \to \infty$ で厳密な事後期待値に収束するが、有限の N では 2 つの確率的な平均の比であるためにわずかな統計的バイアスを伴う。多数の尤度項の積は浮動小数点演算ですぐにアンダーフローを起こすため、log-sum-exp トリックと同様、常に対数重みを計算し、最大の対数重みを引いてから指数をとる。q が事前分布のとき、重要度重み（importance

weight）は尤度そのものに帰着し、事前分布からの各サンプルは観測データをどれだけうまく説明できるかによって重み付けされる。

36.8.2.3 有効標本サイズ

すべてのサンプルを残すからといって、すべてのサンプルが等しく役に立つわけではない。たった 1 つのサンプルが重みのほぼすべてを占めるような重み付き標本は、N がどれほど大きくても実質的に 1 サンプル分の価値しかない。これを定量化するため、正規化された重み $\bar{w}_s = w_s / \sum_r w_r$ から**有効標本サイズ**（effective sample size、ESS）を次のように定義する。

$$N_{\mathrm{eff}} = \frac{1}{\sum_{s=1}^{N} \bar{w}_s^2}. \tag{36.103}$$

すべての重みが等しいとき（$\bar{w}_s = 1/N$）は $N_{\mathrm{eff}} = N$ となり、1 つのサンプルがすべての重みを独占するときは $N_{\mathrm{eff}} = 1$ となる。大まかに言えば、ESS はその重み付き標本が厳密な事後分布からの独立サンプル何個分に相当するかを推定するものである。しかし、まもなく見るとおり、ESS は 2 つの重要な点で誤解を招くことがある。

命題。$N \to \infty$ のとき、次が成り立つ。

$$\frac{N_{\mathrm{eff}}}{N} \to \frac{(\mathbb{E}_q[w])^2}{\mathbb{E}_q[w^2]} = \frac{1}{1 + \chi^2(p \,\|\, q)}, \qquad \chi^2(p \,\|\, q) = \int \frac{p^2}{q}\, d\theta - 1, \tag{36.104}$$

ここで、p は正規化された事後分布である。同じ提案分布 q に対して、この極限効率は常に棄却サンプリングの受理率 $p(\mathcal{D})/M$ 以上である。

証明。@eq-mdl-bayes-is-ess の分子と分母を N^2 で割ると $N_{\mathrm{eff}}/N = (\frac{1}{N}\sum_s w_s)^2 / (\frac{1}{N}\sum_s w_s^2)$ となり、両方の平均に大数の法則が適用できる。$w = p(\mathcal{D})\, p/q$ を代入すると $\mathbb{E}_q[w^2] = p(\mathcal{D})^2 \int p^2/q$ および $\mathbb{E}_q[w] = p(\mathcal{D})$ が得られ、χ^2 ダイバージェンスによる表現が導かれる。最後に、$w \le M$ より $\mathbb{E}_q[w^2] \le M\, \mathbb{E}_q[w]$ であるから、この極限は少なくとも $\mathbb{E}_q[w]/M = p(\mathcal{D})/M$ 以上である。■

したがって ESS は、q が p にどれだけ近いかを表す経験的な指標である。大きな N の極限では、重要度サンプリングは包絡線の上界 M を一切必要とすることなく、棄却サンプリングが受理する数以上の有効サンプルを確保する。一様事前分布を提案分布とする先ほどのコインの例に戻ると、$\mathbb{E}_q[w^2] = \int_0^1 \theta^{18}(1-\theta)^8\, d\theta$ であり、極限の ESS 比率は 0.42 となって棄却サンプリングの受理率 0.305 を上回る。棄却サンプリングのコードセルで生成したまったく同じサンプルを再利用し、3 分の 2 を捨てる代わりに重み付けし直すだけでよいのである。

```
w = lik / lik.sum()                          # normalized weights
post_coin = w @ theta_coin
print(f'ESS fraction {1 / (w @ w) / N:.3f}, mean {post_coin:.3f}, '
      f'sd {np.sqrt(w @ (theta_coin - post_coin)**2):.3f}')
```

```
ESS fraction 0.424, mean 0.666, sd 0.118
```

この極限公式はまた、スケールの不適切な提案分布がいかに急激に破綻するかも暴き出す。標準偏差 s_p の 1 次元ガウス目標分布と、同じ中心を持ち標準偏差 $s_q = r s_p$ のガウス提案分布を考えると、極限の ESS 比率は $\sqrt{2r^2 - 1}/r^2$ となる（演習問題 5）。この曲線は 0.83（$r = 0.8$）をと

り、最大値 1（$r = 1$）に達したのち、0.66（$r = 2$）、0.46（$r = 3$）へと緩やかに減少する。ところが $r \leq 1/\sqrt{2}$ では、$\mathbb{E}_q[w^2] = \infty$ となるために突如としてゼロへと崩壊する。狭すぎる提案分布は、広すぎる提案分布よりもはるかに壊滅的な失敗を招くのである。

重要度サンプリングの恒等式 Equation 36.102 が成り立つことは、2 つの条件によって保証される。すなわち、$q(\theta) > 0$ が $\tilde{p}(\theta) > 0$ なるすべての点で成り立つこと、および $\mathbb{E}_q|wh| < \infty$ であることである。中心極限定理、ひいては $O(1/\sqrt{N})$ のエラーバーを得るには、さらに $\mathbb{E}_q[w^2]$ と $\mathbb{E}_q[w^2 h^2]$ が有限である必要がある。しかもこれらの条件が満たされていても、Figure 36.38 が示すように、有限標本における ESS は危険なほど楽観的な値を出すことがある。

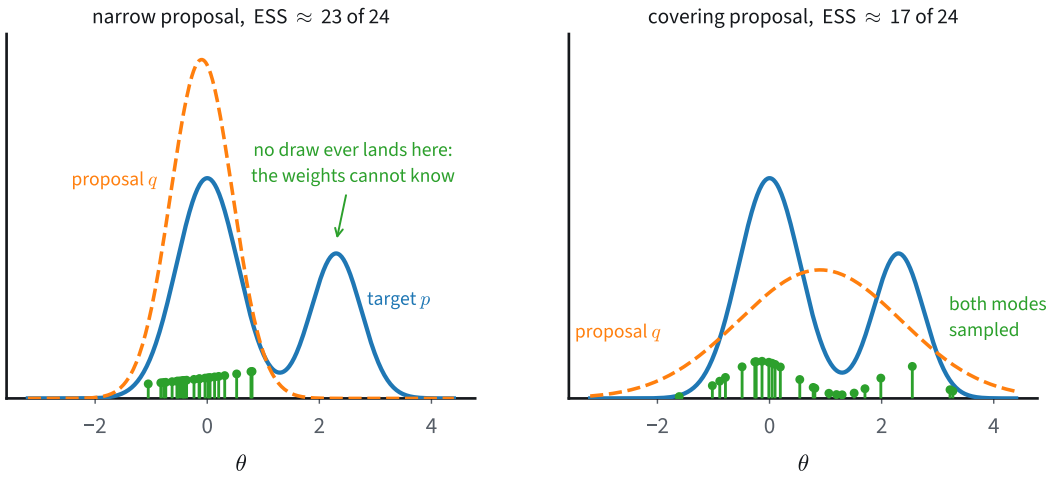


Figure 36.38: 2 つの異なる提案分布を用いた二峰性目標分布の重要度サンプリング。縦棒は 24 個のサンプルの正規化された重みを表す。左：幅の狭い提案分布は右側のモードを一度も訪れないため、推定値が大きく誤っているにもかかわらず重みはほぼ一様に見える（ESS は 23 / 24）。右：より幅の広い提案分布は両方のモードを覆っており、重みの均一性は下がる（ESS は 17 / 24）ものの、推定値ははるかに正確である。ESS が測るのは重みの不均等さであり、提案分布が事後確率質量のすべてを見つけられたかどうかではない。

36.8.2.4 ロジスティック事後分布に対する提案分布

ロジスティック回帰の事後分布に対しては、提案分布をどう選べばよいだろうか。20,000 サンプルのガウス事前分布と 5,000 サンプルのラプラス近似という 2 つのガウス分布を、サンプル数に対する ESS の割合、最大の正規化重み、およびグリッド基準値に対する推定事後平均の誤差（事後標準偏差を単位として測定）を記録することで比較してみよう。

```
def gaussian_logpdf(samples, mean, cov):
    diff = samples - mean
    _, logdet = np.linalg.slogdet(cov)
    maha = np.einsum('ni,ij,nj->n', diff, np.linalg.inv(cov), diff)
    return -0.5 * (len(mean) * np.log(2 * np.pi) + logdet + maha)

def importance(log_target, mean, cov, N):
    """Self-normalized importance sampling from a Gaussian proposal."""
    samples = rng.multivariate_normal(mean, cov, N)
```

```
        log_w = log_target(samples) - gaussian_logpdf(samples, mean, cov)
        w = np.exp(log_w - log_w.max())
        w /= w.sum()
        return w @ samples, 1 / (w @ w), w.max()

    for name, mean, cov, N in [('prior', np.zeros(2), prior_var * np.eye(2), 20000),
                               ('Laplace', theta_map, laplace_cov, 5000)]:
        estimate, ess, w_max = importance(log_joint, mean, cov, N)
        error = np.abs(estimate - post_mean) / post_sd
        print(f'{name:8s}: ESS {ess / N:.1%} of {N}, largest weight '
              f'{w_max:.4f}, error in sds {error.round(3)}')
```

```
prior   : ESS 4.2% of 20000, largest weight 0.0025, error in sds [0.01  0.007]
Laplace : ESS 84.8% of 5000, largest weight 0.0037, error in sds [0.011 0.007]
```

事前分布からのサンプルの大半は尤度がほぼゼロの領域に落ちるため、事前分布を提案分布に使うと ESS は全サンプルの約 4% にしかならない。ラプラスガウス分布によって提案分布の照準を事後分布に合わせると、ESS は劇的に向上し、まれにずっと低い値に落ちる実行があるものの、通常はサンプルの 80% から 90% が有効となる。どちらの提案分布でも、事後平均の推定誤差は通常グリッド基準値から事後標準偏差の数パーセント以内に収まり、0.15 を超える誤差が出ることは滅多にない。

ところがラプラス提案分布には、通常の ESS の出力結果からは決して見えてこない巧妙な欠陥が潜んでいる。原点から遠く離れた領域では、事後分布の裾は事前分布に支配される。任意の半直線 $\theta = t\mathbf{v}$ に沿って、対数尤度は t に関して高々線形にしか減衰しないのに対し（演習問題 8）、対数事前密度は $-t^2/(2\tau^2)$ として二次関数的に減衰するからである。その結果、ガウス提案分布 $\mathcal{N}(\mathbf{m}, \mathbf{S})$ に対して、積分 $\mathbb{E}_q[w^2] = \int \tilde{p}^2/q$ は $\mathbf{S}$ の固有値が 1 つでも $\tau^2/2 = 2$ を下回ると発散する。今の例ではラプラス共分散の 2 つの固有値（およそ 1.17 と 0.05）がいずれも 2 を大きく下回っているため、重要度重みは 1 本の細い軸方向だけでなく**あらゆる**方向で無限大の分散を持つ。

では、なぜ 5,000 サンプルの実行結果はあれほど健全に見えたのだろうか。重みが爆発する領域はラプラス標準偏差の何倍も離れた場所にあるため、5,000 サンプルの実行の大半はたまたまそこに足を踏み入れないからである。しかし 100 回中数回の実行では、遠くに落ちた 1 つのサンプルが総重みの巨大な割合を独占し、ESS が急落する。1 回実行しただけではこの不安定性は見抜けず、繰り返し実行するか裾の解析を行って初めて明らかになる。重みの分散を有限に戻す簡単な対策は 2 つある。ガウス分布をより裾の重いスチューデント t 提案分布に置き換えること（演習問題 8）、あるいは $q = 0.9\,\mathcal{N}(\theta_{\mathrm{MAP}}, \mathbf{H}^{-1}) + 0.1\,\text{prior}$ のような**防御的混合分布**（defensive mixture）を使うことである。この混合密度は常に事前密度の少なくとも 10 分の 1 以上あるため、すべての重みは尤度の 10 倍で上から抑えられ、しかも尤度は決して 1 を超えない。

重要度サンプリングは未知のエビデンスを自動的に相殺してくれるが、それでも事後分布全体を覆う単一の大域的な提案分布 q を必要とする。高次元空間では、そのような役割を果たせる単純な大域分布は存在しない。

36.8.3 マルコフ連鎖モンテカルロ法

事後分布全体を覆う単一の大域的な提案分布を探し求める代わりに、局所的な移動を通じてパラメータ空間を探索し、各領域の事後確率質量に比例した時間をそこで過ごすランダムウォークを設計することができる。このようなランダムウォークは**マルコフ連鎖**（Markov chain）であり、次の状態は**遷移規則**（transition rule）$T(\theta' \mid \theta)$（θ から θ' へ移動する確率）を通じて現在の状態のみに依存する。現在の状態を分布 π から引いたときに次の状態も同じく π に従うなら、π はその連鎖に関して**定常**（stationary）であるといい、次式で表される。

$$\sum_{\theta} \pi(\theta)\, T(\theta' \mid \theta) = \pi(\theta') \quad \text{for every } \theta', \tag{36.105}$$

連続パラメータの場合、この和は積分に置き換わる。ここでの目標は、事後分布を定常分布（stationary distribution）として持つ単純な遷移規則を構成することである。

36.8.3.1 メトロポリス規則

メトロポリス法（Metropolis algorithm）は、任意の対称な提案分布 $g(\theta' \mid \theta) = g(\theta \mid \theta')$（たとえば局所的なガウスステップ $\theta' = \theta + \mathbf{L}\epsilon$）からそのような連鎖を構成する。現在の状態 θ から候補 $\theta' \sim g(\cdot \mid \theta)$ を提案する。候補の事後密度が現在以上、すなわち $\tilde{p}(\theta') \geq \tilde{p}(\theta)$ であれば、必ず θ' へ移動する。下り方向への移動となる場合は、確率 $\tilde{p}(\theta')/\tilde{p}(\theta)$ でこれを受理し、そうでなければ θ にとどまる。決定的に重要なのは、提案が棄却されたときには現在の状態を繰り返し、その繰り返しも連鎖における次の状態として数えることである（Figure 36.39）。

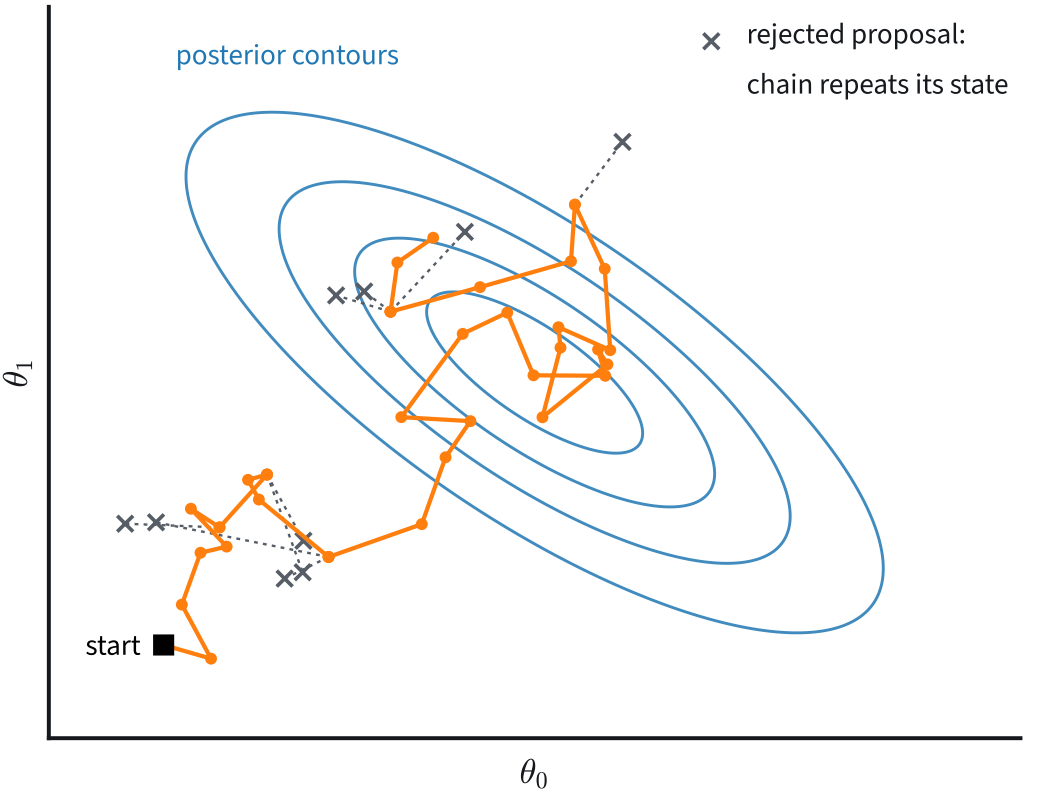


Figure 36.39: 事後分布の等高線上におけるメトロポリスランダムウォーク。開始点の四角形から、各ステップで局所的な移動を提案する。上り方向の移動は常に受理され（折れ線）、下り方向の移動は確率的に受理され、棄却された提案（罰印）では連鎖がもう 1 ステップその場にとどまる。長期的には、任意の領域で過ごす時間の割合がその事後確率質量に一致する。

Hastings は、提案分布の方向的な偏り（バイアス）を補正する因子 $g(\theta \mid \theta')/g(\theta' \mid \theta)$ を密度比（density ratio）に掛けることで、この規則を非対称な提案分布へと一般化した (Hastings 1970)。これにより、候補は次の確率で受理される。

$$\alpha(\theta, \theta') = \min\left(1, \frac{\tilde{p}(\theta')\, g(\theta \mid \theta')}{\tilde{p}(\theta)\, g(\theta' \mid \theta)}\right). \tag{36.106}$$

提案分布が滅多に訪れない状態への移動ほど受理されやすくなり、g が対称なときには補正因子は 1 に等しくなる。ここでもやはり、未知のエビデンスは $\tilde{p}$ の値の比の中で相殺される。

なぜ棄却を数に入れることが不可欠なのかは、最小のマルコフ連鎖を見ればよくわかる。目標分布 $\pi = (\tfrac{1}{4}, \tfrac{3}{4})$ を持つ 2 つの状態と、常に**もう一方**の状態を提案する対称な提案分布を考えよう。状態 1 からは目標密度の比が 3 なので、連鎖は必ず状態 2 へジャンプする。状態 2 からは比が $\tfrac{1}{3}$ なので、状態 1 へ確率 $\tfrac{1}{3}$ でジャンプし、状態 2 に確率 $\tfrac{2}{3}$ でとどまる。行を現在の状態として遷移行列を書き下せば、定常性を直接確かめられる。

$$\mathbf{T} = \begin{pmatrix} 0 & 1 \\ \tfrac{1}{3} & \tfrac{2}{3} \end{pmatrix}, \qquad \left(\tfrac{1}{4},\, \tfrac{3}{4}\right)\mathbf{T} = \left(\tfrac{3}{4}\cdot\tfrac{1}{3},\; \tfrac{1}{4} + \tfrac{3}{4}\cdot\tfrac{2}{3}\right) = \left(\tfrac{1}{4},\, \tfrac{3}{4}\right).$$

もし棄却を捨てて受理された移動だけを記録したなら、系列は単に $2, 1, 2, 1$ を交互に繰り返すだけとなり、$(\tfrac{1}{2}, \tfrac{1}{2})$ となって本来の $(\tfrac{1}{4}, \tfrac{3}{4})$ は得られない。棄却後に状態 2 を繰り返し訪問することこそが、状態 2 に本来の重みを与えているのである（Figure 36.40）。

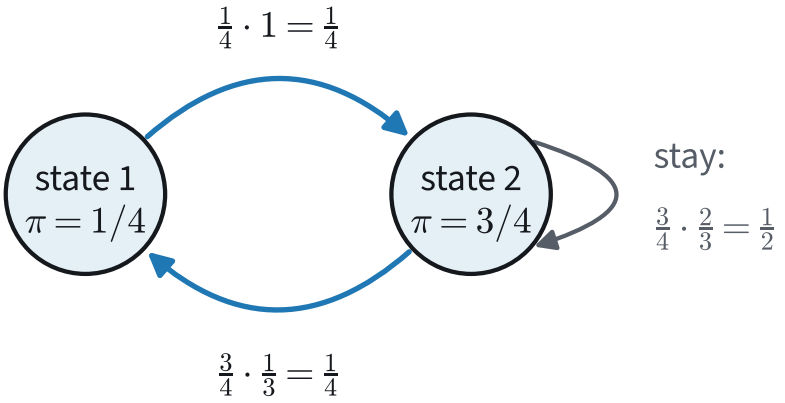


Figure 36.40: 目標分布 $\pi = (1/4, 3/4)$ を持つ 2 状態メトロポリス連鎖。状態 1 から状態 2 への確率フローは $\tfrac{1}{4}\cdot 1$ であり、逆方向のフローは $\tfrac{3}{4}\cdot\tfrac{1}{3}$ である。両者はともに $1/4$ に等しいため、滞在比率 $(1/4, 3/4)$ が毎ステップ保たれる。自己ループは状態 2 からの棄却された移動を表しており、受理された移動のみを記録すると $(1/2, 1/2)$ になってしまう。

36.8.3.2 詳細釣り合いと定常性

Figure 36.40 は、メトロポリス・ヘイスティングス法の背後にある中核的な仕組みを捉えている。1 ステップのうちに状態 1 から状態 2 へ流れる確率質量と逆向きに流れる確率質量が等しいため、どちらの状態も確率を得たり失ったりしない。このペアごとの釣り合いがあらゆる状態の組に対して成り立つとき、目標分布が定常分布であることが保証される。本節から理論的な結果を 1 つだけ覚えて帰るとすれば、この結果にせよ。

命題。遷移規則が次の**詳細釣り合い**（detailed balance）を満たすとする。

$$\pi(\theta)\,T(\theta' \mid \theta) = \pi(\theta')\,T(\theta \mid \theta') \quad \text{for all } \theta, \theta', \tag{36.107}$$

このとき、π はその連鎖の定常分布である。さらに、メトロポリス・ヘイスティングス連鎖は $\pi \propto \tilde{p}$ に対して詳細釣り合いを満たす。

証明。@eq-mdl-bayes-detailed-balance の両辺を θ にわたって足し合わせると Equation 36.105 の左辺が得られ、一方右辺は $\pi(\theta')\sum_{\theta} T(\theta \mid \theta') = \pi(\theta')$ へと簡約される。なぜなら、θ' からの遷移確率の和は 1 だからである。後半の主張について、$\theta' = \theta$ のときは詳細釣り合いが自動的に成り立つ。$\theta' \neq \theta$ のときは、移動が起きるのは θ' が提案されかつ受理されるとき、かつそのときに限り、次式が成り立つ。

$$\begin{aligned} \pi(\theta)\,T(\theta' \mid \theta) &= \pi(\theta)\,g(\theta' \mid \theta)\,\alpha(\theta, \theta') \\ &= \min\bigl\{\pi(\theta)\,g(\theta' \mid \theta),\ \pi(\theta')\,g(\theta \mid \theta')\bigr\}, \end{aligned}$$

この式は明らかに θ と θ' に関して対称である。π の正規化定数は α の中で約分されて消えるため、$\tilde{p}$ を π の代わりに代入してよい。$\blacksquare$

鍵となる代数的なステップは πg を最小値関数の内側へ引き入れることであり、これによって対称性が一目でわかるようになる。連続な状態空間では、棄却された移動が作る点質量に対してもう少し丁寧な扱いが必要になる（演習問題 1）。

定常性だけでは、任意の初期点から出発した連鎖が π に収束することまでは保証されない。連鎖はさらに、正の事後確率質量を持つあらゆる領域に到達可能でなければならず、領域間を決定論的に巡回してもならない。これら 2 つの緩やかな条件の下で、連鎖が任意の領域で過ごす時間の長期的な割合はその領域の事後確率（posterior probability）に収束する。

メトロポリス法の実装はわずか数行で済む。以下では、ラプラス共分散のコレスキー因子（Cholesky factor）$\mathbf{L}\mathbf{L}^{\top} = \mathbf{H}^{-1}$ を用いてガウス提案分布の形状を整える。

```
def metropolis(log_target, start, proposal_chol, n_steps):
    current = np.array(start, dtype=float)
    current_logp = log_target(current)
    states, accepted = np.empty((n_steps, len(current))), 0
    for t in range(n_steps):
        proposal = current + proposal_chol @ rng.standard_normal(len(current))
        proposal_logp = log_target(proposal)
        if np.log(rng.random()) < proposal_logp - current_logp:   # accept
            current, current_logp = proposal, proposal_logp
            accepted += 1
        states[t] = current            # a rejection repeats the state
    return states, accepted / n_steps
```

ラプラスガウス分布の 2 倍の広がりで散らした過分散（over-dispersed）な初期点 $\theta_{\mathrm{MAP}} + 2\mathbf{L}\epsilon$ から、それぞれ 12,000 ステップからなる 4 本の独立な連鎖を走らせ、大きさ $1.1\,\mathbf{L}\epsilon$ のステップを提案する。初期の状態にはまだ各連鎖がどこから出発したかの影響が残っているため、平均を計算する前に各連鎖の最初の 2,000 回の反復、すなわち**ウォームアップ**期間を捨てる。

```
def run_chains(scale, n_chains=4, n_steps=12000, warmup=2000):
    L = np.linalg.cholesky(laplace_cov)
    chains, rates = [], []
    for _ in range(n_chains):
        start = theta_map + 2 * L @ rng.standard_normal(2)  # over-dispersed
        states, rate = metropolis(log_joint, start, scale * L, n_steps)
        chains.append(states[warmup:])
        rates.append(rate)
    return np.array(chains), np.array(rates)

chains, rates = run_chains(1.1)
print('acceptance rates:', rates.round(3))
```

```
acceptance rates: [0.533 0.527 0.52  0.537]
```

どの連鎖でも全提案のおよそ半分が受理される。定常分布を決めるのはガウス提案分布ではなくメトロポリス受理規則であることを思い出そう。したがって、提案分布の形状が悪くても連鎖の進行が遅くなるだけで、統計的バイアスが持ち込まれることはない。

36.8.3.3 マルコフ連鎖の混合に関する診断

独立標本（independent samples）とは異なり、マルコフ連鎖における連続する状態には相関があり、どの連鎖も初期化した場所から出発する。連鎖から計算した平均を信頼する前に、3 つの診断上の問いを立て、それぞれに具体的な指標で答えることにしよう。

各連鎖は初期点を忘れ、同じ分布に収束したか。視覚的には各連鎖のトレースプロットを描けば停滞やドリフトが露わになり、$\widehat{R}$ は連鎖間の不一致を定量化する。それぞれ N 個の状態からなる複数の連鎖が与えられたとき、W を連鎖内分散の平均とし、B を連鎖平均の分散の N 倍とする。すると次式が成り立つ。

$$\widehat{R}^2 = \frac{\frac{N-1}{N} W + \frac{1}{N} B}{W}. \tag{36.108}$$

分子は全体の事後分散（posterior variance）を推定するものであり、連鎖同士が離れたままでいる間は過大評価する傾向がある。一方、各連鎖が事後分布の一部しか探索していない間は、W は事後分散を過小評価する。したがって $\widehat{R}$ は 2 つの分散推定量を比較しており、連鎖同士が混ざり合うにつれて 1 に近づく。実務では、まず各連鎖を前半と後半の 2 つに分割してから**分割** $\widehat{R}$ を計算する。こうすれば、単一の連鎖がドリフトしている場合でも、互いに食い違う 2 本の連鎖として検出される。出力を信頼するには 0.01 以内の差で 1 の近くに収まっていることが必要条件であるが、それだけで十分条件になることは決してない。すべての連鎖が同じ遠くのモードを見逃していれば、連鎖同士は何事もないかのように一致してしまうからである。

相関を持つこれらの状態は、実際には独立サンプル何個分に相当するのか。ρ_k を k ステップ離れた状態間の自己相関（autocorrelation）とする。積分自己相関時間（integrated autocorrelation time）$\kappa = 1 + 2\sum_{k \geq 1} \rho_k$ は、連鎖の何ステップ分が 1 個の独立サンプルと同じ情報量を持つかを数える量である。したがって、c 本の連鎖（各 N 状態）からは有効標本サイズ $N_{\text{eff}} = cN/\kappa$ が得られ、これは重要度サンプリングにおけるものと同じ意味と記号を持つ。

系列相関の最も単純なモデルは 1 次自己回帰過程（first-order autoregressive process）AR(1) であり、$x_t = \phi x_{t-1} + \varepsilon_t$（$\varepsilon_t$ は独立なノイズで $|\phi| < 1$）で与えられる。各ステップで連鎖は直前の値の割合 ϕ を保持する。ラグ k の自己相関は $\rho_k = \phi^k$ であり、これから次式が得られる。

$$\kappa = 1 + 2\sum_{k \geq 1} \phi^k = \frac{1+\phi}{1-\phi}.$$

$\phi = 0.9$ のとき $\kappa = 19$ となり、10,000 個の状態は約 530 個の独立サンプルにしかならない。$\phi = 0.99$ では $\kappa = 199$ となり、わずか 50 個程度の独立サンプルしか残らない。

非常に大きなラグにおける推定自己相関はノイズに支配されるため、ここでの実装では連続するペア $\rho_{2k} + \rho_{2k+1}$ ごとにまとめ、最初に現れる負のペアで和を打ち切る。実用レベルの推論パッケージでは両方の診断指標がさらに改良されており、ランク正規化された $\widehat{R}$ や、分布の中心部（バルク）と裾で別々に計算した有効標本サイズが用いられている。

推定された事後平均はどれほど正確か。マルコフ連鎖の中心極限定理により、推定された事後平均の**モンテカルロ標準誤差**（Monte Carlo standard error、MCSE）は、事後標準偏差を $\sqrt{N_{\text{eff}}}$ で割ったものになる。

```
def split_rhat(chains):
    """Split R-hat per coordinate; chains has shape (c, N, d)."""
    half = chains.shape[1] // 2
    halves = np.concatenate([chains[:, :half], chains[:, half:2 * half]])
    W = halves.var(axis=1, ddof=1).mean(axis=0)
    B = half * halves.mean(axis=1).var(axis=0, ddof=1)
    return np.sqrt(((half - 1) / half * W + B / half) / W)

def autocorrelation(x):
    """rho_k of one sequence for k = 0, ..., N - 1, by FFT."""
    x = x - x.mean()
    f = np.fft.rfft(x, n=2 * len(x))
    acov = np.fft.irfft(f * np.conj(f))[:len(x)]
    return acov / acov[0]

def chain_ess(chains):
    """c N / kappa per coordinate, summing rho in pairs (Geyer)."""
    c, N, d = chains.shape
    ess = np.empty(d)
    for j in range(d):
        rho = np.mean([autocorrelation(chain[:, j]) for chain in chains],
                      axis=0)
        pairs = rho[:N - 1:2] + rho[1:N:2]        # rho_2k + rho_2k+1
        negative = np.flatnonzero(pairs < 0)
        n_pairs = negative[0] if len(negative) else len(pairs)
        ess[j] = c * N / (-1 + 2 * pairs[:n_pairs].sum())
    return ess
```

これらの診断指標を実際に試してみよう。もしエラーバーが適切に較正されていれば、4 本の連鎖から得られた MCMC 平均は、厳密なグリッド平均からおよそ 1 MCSE 以内に収まるはずである。あわせて、測定された傾きの自己相関時間 κ が、ラグ 1 の自己相関 ρ_1 で評価した AR(1) の公式とどれほどよく一致するかも確認する。

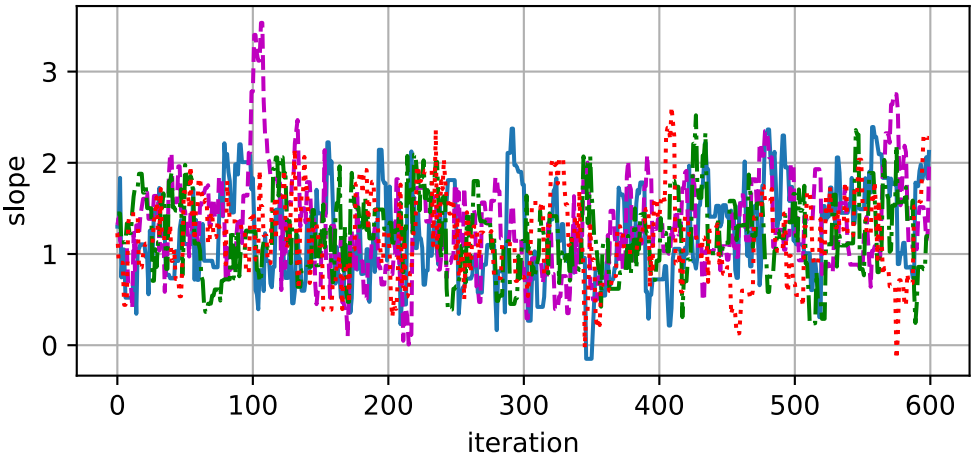
```
flat = chains.reshape(-1, 2)
ess = chain_ess(chains)
mcmc_mean, mcmc_sd = flat.mean(axis=0), flat.std(axis=0, ddof=1)
```

```
mcse = mcmc_sd / np.sqrt(ess)
rho1 = np.mean([autocorrelation(chain[:, 1])[1] for chain in chains])
print('split R-hat       :', split_rhat(chains).round(3))
print('ESS               :', ess.round().astype(int), 'of', len(flat))
print('(MCMC - grid)/MCSE:', ((mcmc_mean - post_mean) / mcse).round(1))
print(f'slope: rho_1 {rho1:.2f}, AR(1) kappa {(1 + rho1) / (1 - rho1):.1f},'
      f' measured kappa {len(flat) / ess[1]:.1f}')
```

```
split R-hat       : [1.001 1.001]
ESS               : [4351 4248] of 40000
(MCMC - grid)/MCSE: [0.2 0.2]
slope: rho_1 0.81, AR(1) kappa 9.4, measured kappa 9.4
```

最後に、ウォームアップ後の最初の 600 反復における傾きのトレースプロットを描いて診断チェックを締めくくる。

```
d2l.plot(np.arange(600), [chain[:600, 1] for chain in chains],
         xlabel='iteration', ylabel='slope', figsize=(6, 2.5))
```



診断結果は良好である。分割 $\widehat{R}$ は 0.01 以内の差で 1 の近くに収まり、4 本のトレースプロットは密に絡み合っている。ESS は保持された 40,000 個の状態のおよそ 10 分の 1 であり、$\kappa \approx 10$ となってラグ 1 の自己相関 $\rho_1 \approx 0.8$ から得られる AR(1) の予測とぴったり整合する。最も心強いことに、MCMC 平均と厳密なグリッド平均の差はおよそ 1 MCSE であり、不確実性の見積もりが実際の推定誤差と確かに一致していることが確認できる。

36.8.3.4 提案分布のスケールと混合の失敗

メトロポリス法の実行から出力するのが最も簡単な数値は受理率であるが、それ単体からわかることはほとんどない。提案スケールを 0.05 および 10 に設定して（1.1 の代わりに）4 本の連鎖を再実行してみよう。数値を見る前に少し立ち止まり、この両極端のうちどちらの方が有効サンプル数が少なくなるか予想してみてほしい。

```
for scale in [0.05, 10]:
    bad_chains, bad_rates = run_chains(scale)
    print(f'scale {scale:5}: acceptance {bad_rates.mean():.3f}, max R-hat '
          f'{split_rhat(bad_chains).max():.2f}, '
          f'min ESS {chain_ess(bad_chains).min():.0f}')
```

```
scale  0.05: acceptance 0.977, max R-hat 1.22, min ESS 33
scale    10: acceptance 0.021, max R-hat 1.01, min ESS 541
```

微小なステップ（スケール 0.05）では提案の約 97% が受理されるにもかかわらず、1 ステップあたりほとんど動かないため、$\widehat{R}$ は 1.01 を大きく超え、40,000 回の反復からわずか数十個の有効サンプルしか得られない。巨大なステップ（スケール 10）では提案の約 2% しか受理されず、連鎖は長い区間にわたって同じ状態を繰り返して停滞するものの、それでも数百個の有効サンプルは確保できている。最適なステップサイズはこれら両極端の間の程よい位置にある（演習問題 2）。より本質的には、闇雲なランダムウォークが上り方向へ進むのは偶然頼みであるのに対し、$\log \tilde{p}$ の勾配に導かれる提案分布は設計上はじめから上り方向へドリフトする。このような勾配情報を活用した移動はランジュバン動力学（Langevin dynamics）（38.6.2 節）から生じ、そのメトロポリス調整版は演習問題 9 で展開する。

36.8.4 変分推論

サンプリング手法は事後期待値をその都度新たに推定し、常にモンテカルロノイズを伴う。**変分推論**（variational inference）は根本的に異なる道筋をとり、積分を最適化へと変換する。扱いやすい分布族 q_ψ（変分パラメータ ψ で添字付けられる）を選び、ψ を最適化して q_ψ が事後分布をできるだけ忠実に近似するようにしたうえで、q_ψ の下での期待値を閉じた形または高速な独立サンプリングによって計算するのである。

36.8.4.1 エビデンス下界

q_ψ を最適化するには、未知の正規化定数 $p(\mathcal{D})$ を必要とせずに事後分布への近さを測る目的関数が要る。幸い、そのような目的関数はすでに 36.7 節で導出している。そこでは潜在変数上の近似分布 q が対数エビデンスの下界を与え、その差（ギャップ）が KL ダイバージェンスとなっていた。パラメータベクトル θ を潜在変数、データセット $\mathcal{D}$ を観測とみなせば、まったく同じ恒等式が得られる。

$$\log p(\mathcal{D}) = \underbrace{\mathbb{E}_{q_\psi}\big[\log p(\mathcal{D},\theta) - \log q_\psi(\theta)\big]}_{\mathcal{L}(\psi)} + D_{\mathrm{KL}}\big(q_\psi \,\|\, p(\cdot \mid \mathcal{D})\big). \tag{36.109}$$

KL ダイバージェンスは常に非負であるため、$\mathcal{L}(\psi)$ は q_ψ の**エビデンス下界**（evidence lower bound、ELBO）となる。

$$\mathcal{L}(\psi) = \mathbb{E}_{q_\psi}[\log p(\mathcal{D},\theta) - \log q_\psi(\theta)] \le \log p(\mathcal{D}). \tag{36.110}$$

Equation 36.109 の右辺にある 2 つの項の和は定数 $\log p(\mathcal{D})$ に固定されているため、$\mathcal{L}(\psi)$ が 1 単位増えるごとに $D_{\mathrm{KL}}(q_\psi \,\|\, p(\cdot \mid \mathcal{D}))$ はちょうど 1 単位減少する。したがって、ψ に関して ELBO を最大化することは KL ダイバージェンスを最小化することと同値であり、未知のエビデンスは加法的な定数として消える。期待値の中のすべての項は点ごとに評価でき、期待値自体も q_ψ からのサンプルを使って容易に推定できる。

36.8.4.2 KL ダイバージェンスの向き

Equation 36.109 における引数の順序に注目しよう。変分推論が最小化するのは $D_{\mathrm{KL}}(q\|p) = \int q \log(q/p)$、すなわち第 1 引数に q を置いた**逆方向** KL ダイバージェンス（reverse KL divergence）である。その被積分関数は $q > 0$ かつ $p \approx 0$ となる場所ではどこでも発散するため、逆方向 KL は事後分布が質量を持たない領域に q が質量を置くことに対して重いペナルティを課す。その結果、q は低密度領域から縮小して単一の最頻値（モード）に張り付く傾向がある。つまり、逆方向 KL は**モード探索型**（mode-seeking）である。対照的に、**順方向**ダイバージェンス（forward divergence）$D_{\mathrm{KL}}(p\|q) = \int p \log(p/q)$ は $p > 0$ かつ $q \approx 0$ となる領域にペナルティを課すため、q は事後分布の質量全体を覆うように引き伸ばされる。つまり、順方向 KL は**質量被覆型**（mass-covering）である。ガウス分布族の場合、順方向 KL の最小化は事後分布の厳密な平均と共分散に一致させることに帰着するが、その計算には扱いにくい事後分布 p の下での期待値が必要になるのに対し、逆方向 KL は q の下での期待値しか必要としない。Figure 36.41 は、二峰性の目標分布におけるこれら 2 つの振る舞いを対比している。

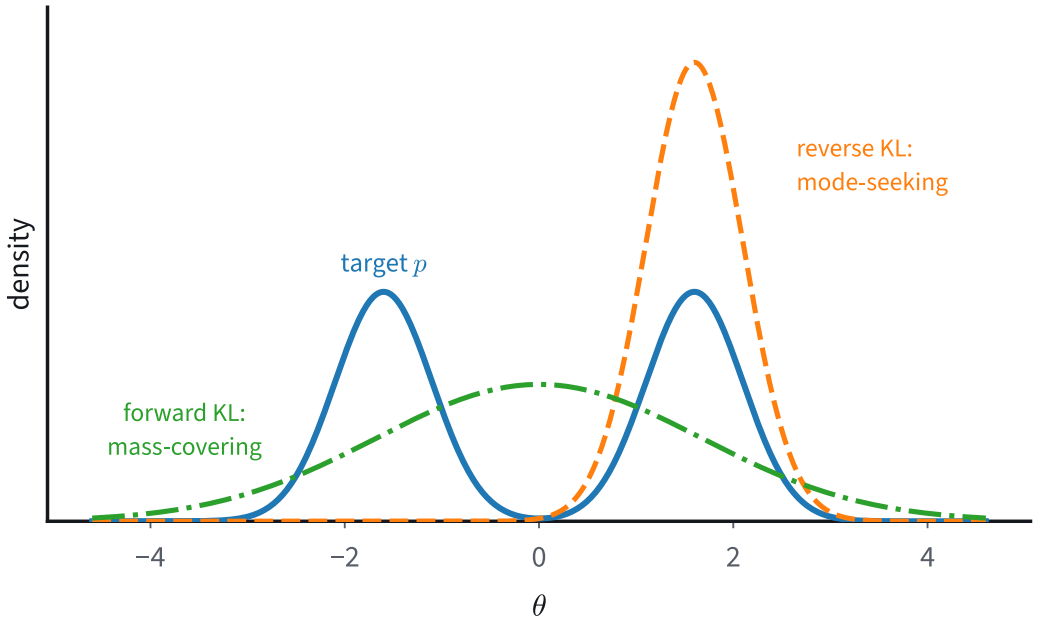


Figure 36.41: KL ダイバージェンスの向きによって、単純すぎる近似がどのように破綻するかが決まる。二峰性の目標分布に対し、変分推論が求める逆方向 KL のガウス分布は単一のモードに張り付いて不確実性を過小評価するのに対し、順方向 KL のガウス分布は両方のモードを覆い、その間にある低密度の谷間にピークを置く。

36.8.4.3 平均場ガウス分布の当てはめ

最も単純な変分分布族は、パラメータを互いに独立としてモデル化する**平均場**（mean-field）ガウス分布 $q = \mathcal{N}(\mathbf{m}, \mathrm{diag}(\mathbf{s}^2))$ である。$\mathbf{m}$ と $\log \mathbf{s}$ に関する ELBO の勾配を計算するため、サンプルを $\theta = \mathbf{m} + \mathbf{s} \odot \epsilon$（ただし $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$）と再パラメータ化する。ノイズの分布は変分パラメータに依存しないため、勾配を期待値の内側へ移動できる（34.7.3 節）。エントロピー $-\mathbb{E}_q[\log q]$ は加法的な定数を除いて $\sum_j \log s_j$ に等しいため、次の不偏な勾配公式が得られる。

$$\nabla_{\mathbf{m}}\mathcal{L} = \mathbb{E}[\nabla_\theta \log p(\mathcal{D},\theta)], \qquad \nabla_{\log \mathbf{s}}\mathcal{L} = \mathbb{E}[\nabla_\theta \log p(\mathcal{D},\theta)\odot \mathbf{s}\odot\epsilon] + \mathbf{1}, \tag{36.111}$$

ここで、$\mathbf{1}$ はエントロピー $\sum_j \log s_j$ を $\log \mathbf{s}$ で微分した項に由来する。第 1 の勾配は平均的に上り方向へ平均を動かし、第 2 の勾配はエントロピーの外向きに広げる力と事後分布の下向きの曲率が釣り合うまで q を膨張させる。ロジスティック事後分布に対して最適化を実行する前に、目標分布がガウス分布である場合を厳密に解くことで、何が起きるかを予測しておこう。

命題。 精度行列 (precision matrix) Λ を持つガウス目標分布 $p = \mathcal{N}(\mu, \Lambda^{-1})$ に対し、$D_{\mathrm{KL}}(q\|p)$ を最小化する平均場ガウス分布は、平均 μ と、目標分布の**条件付き**分散である分散 $s_j^2 = 1/\Lambda_{jj}$ を持つ。相関 ρ を持つ 2 次元の場合、各 s_j は真の周辺標準偏差の $\sqrt{1-\rho^2}$ 倍となる。

証明。 q に依存しない定数を除くと、$D_{\mathrm{KL}}(q\|p) = -\sum_j \log s_j + \frac{1}{2}\mathbb{E}_q[(\theta-\mu)^\top \Lambda(\theta-\mu)]$ となる。q の下では各座標が平均 m_j、分散 s_j^2 で独立であるため、ゆらぎの交差項は期待値をとるとすべて消え、$\sum_j \Lambda_{jj} s_j^2 + (\mathbf{m}-\mu)^\top\Lambda(\mathbf{m}-\mu)$ が残る。Λ は正定値であるから、$\mathbf{m}$ に関する二次形式は $\mathbf{m} = \mu$ で最小となる。s_j に関する導関数をゼロとおく式 $-1/s_j + \Lambda_{jj} s_j = 0$ より、$s_j^2 = 1/\Lambda_{jj}$ が得られる。$d = 2$ の場合、2×2 共分散行列の逆行列を求めると $1/\Lambda_{jj} = \sigma_j^2(1-\rho^2)$ となる (ここで σ_j は周辺標準偏差である)。■

対角ガウス分布は相関のある事後分布に合わせて主軸を回転させることができず、逆方向 KL の下では低密度の隅にはみ出すことに対して厳しいペナルティを支払う。そのため、事後分布の周辺全体の幅ではなく、事後分布を切断した条件付きスライスの幅まで収縮する (Figure 36.42)。この収縮は、分解された分布族と逆方向 KL を組み合わせたことによる固有の帰結である点に注意しよう。完全共分散ガウス分布をまったく同じ ELBO 目的関数でガウス目標分布に当てはめれば、誤差なく元の分布が復元される。ここでのロジスティック回帰の例では $\rho = -0.86$ であるから、上の命題より平均場の標準偏差は真の周辺標準偏差のおよそ $\sqrt{1-0.86^2} = 0.51$ 倍、すなわち約半分の幅になると予測される。

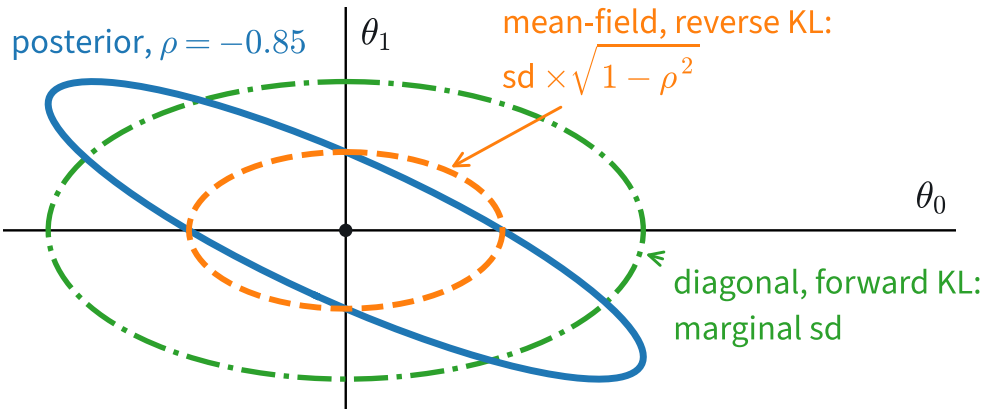


Figure 36.42: 標準偏差 $(1, 0.5)$、相関 $\rho = -0.85$ を持つ相関付きガウス事後分布に当てはめた対角ガウス分布 (2 標準偏差の楕円として表示)。分解された分布族は傾くことができないため、逆方向 KL (オレンジ) は周辺標準偏差の $\sqrt{1-\rho^2} \approx 0.53$ 倍である条件付きの幅まで収縮するのに対し、順方向 KL (緑) は低密度領域へはみ出しながらも周辺の幅を保つ。

この予測を数値的に確かめてみよう。ラプラス近似を初期値とし、1 ステップあたり 256 個の再パラメータ化されたサンプルと $1/\sqrt{t}$ で減衰する学習率を用いて、Equation 36.111 による確率

的勾配上昇法を 3,000 反復実行する。最後の 1,000 反復を平均して最適化ノイズを平滑化することで、当てはめられた標準偏差を、グリッドから求めた相関 ρ を使った命題の予測値と直接比較できる。

```
    m_vi, log_s_vi = theta_map.copy(), np.log(laplace_sd)
    m_avg, log_s_avg = np.zeros(2), np.zeros(2)
    n_steps, n_avg = 3000, 1000
    for t in range(1, n_steps + 1):
        eps = rng.standard_normal((256, 2))
        s_vi = np.exp(log_s_vi)
        g = grad_log_joint(m_vi + s_vi * eps)        # at theta = m + s * eps
        lr = 0.05 / np.sqrt(t)
        m_vi += lr * g.mean(axis=0)
        log_s_vi += lr * ((g * eps * s_vi).mean(axis=0) + 1)
        if t > n_steps - n_avg:                      # average the last iterates
            m_avg += m_vi / n_avg
            log_s_avg += log_s_vi / n_avg
    s_avg = np.exp(log_s_avg)
    print('             mean             sd')
    print('grid      ', post_mean.round(3), post_sd.round(3))
    print('Laplace   ', theta_map.round(3), laplace_sd.round(3))
    print('mean-field', m_avg.round(3), s_avg.round(3))
    print('predicted sd, grid sd * sqrt(1 - rho^2):',
          (post_sd * np.sqrt(1 - post_rho**2)).round(3))
```

```
             mean             sd
grid       [-2.347  1.237] [1.042 0.48 ]
Laplace    [-2.138  1.117] [1.004 0.457]
mean-field [-2.275  1.219] [0.531 0.244]
predicted sd, grid sd * sqrt(1 - rho^2): [0.536 0.247]
```

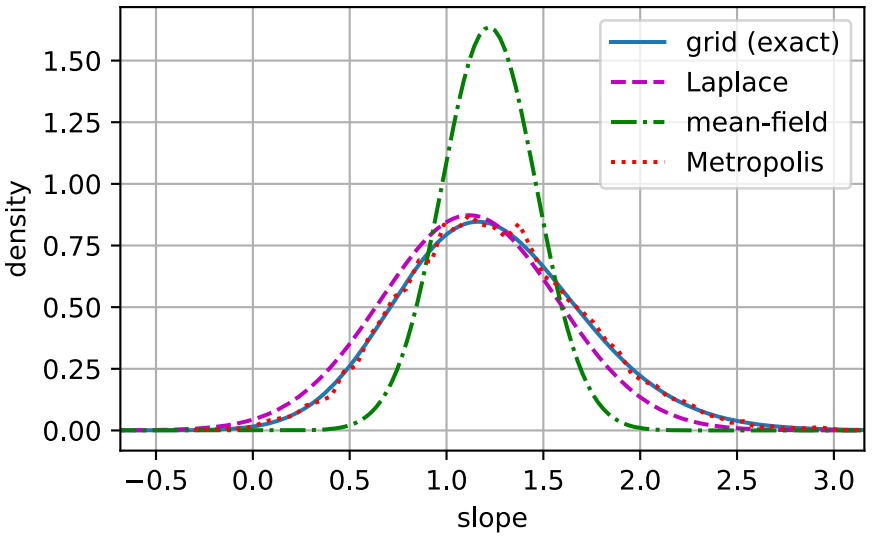
平均場近似の平均はグリッド平均から事後標準偏差のおよそ 10 分の 1 以内に収まる一方、標準偏差は真の幅の約半分まで縮み、解析的な予測値（0.54 と 0.25）と 2% 以内で一致する。最適化はきれいに収束しているため、残る食い違いはすべて制約の強い変分分布族に起因するものである。完全共分散ガウス分布、混合分布、正規化フロー（normalizing flow）といったより表現力の高い分布族を用いれば、より高い計算コストと引き換えにこのギャップを埋められる。ELBO の収束が確認してくれるのはあくまで**選んだ分布族の内部で**最適化が完了したことだけであり、それ自体はその分布族が実際に事後分布に適合していることを決して保証しない点を心に留めておこう。

36.8.5 各手法の比較

これで、まったく同じ事後分布に対する 4 つの異なる近似が出揃った。厳密なグリッド曲線、ラプラス近似および平均場近似のガウス分布、そしてメトロポリス法によるサンプルのヒストグラムを、傾き θ_1 の周辺密度の上で 1 つの図に並べて比較しよう。

```
    def gauss_pdf(t, mean, sd):
        return np.exp(-0.5 * ((t - mean) / sd)**2) / (sd * np.sqrt(2 * np.pi))

    b1 = grid_axes[1]
    exact = grid_weight.reshape(m, m).sum(axis=0) / (b1[1] - b1[0])
    hist, edges = np.histogram(flat[:, 1], bins=60, density=True)
```

```
d2l.plot([b1, b1, b1, (edges[:-1] + edges[1:]) / 2],
         [exact, gauss_pdf(b1, theta_map[1], laplace_sd[1]),
          gauss_pdf(b1, m_avg[1], s_avg[1]), hist],
         xlabel='slope', ylabel='density', figsize=(5, 3),
         xlim=(post_mean[1] - 4 * post_sd[1], post_mean[1] + 4 * post_sd[1]),
         legend=['grid (exact)', 'Laplace', 'mean-field', 'Metropolis'])
```



メトロポリス法のヒストグラムは、歪度も含めて厳密な事後分布を忠実に捉えている。ラプラス近似の曲線は真の平均より左にある最頻値を中心としており、わずかに幅が狭すぎる。平均場ガウス分布は厳密な周辺分布のおよそ半分の幅しかなく、その構成上、真の相関 $\rho = -0.86$ に対してゼロの相関を報告する。

手法	表現	主な診断指標	典型的な破綻様式
グリッド	格子上の重み付き点	グリッド端における質量	計算コスト m^d（次元 d）
ラプラス近似	最頻値における単一のガウス分布	サンプルとの比較	歪み、裾、境界、他のモード
棄却サンプリング	独立な厳密サンプル	受理率	受理率が $n^{-d/2}$ で低下、上界 M が必要
重要度サンプリング	独立な重み付きサンプル	重みの ESS、最大重み、再実行	裾の軽い提案分布や位置のずれた提案分布
MCMC	相関のある事後サンプル	トレース、分割 $\widehat{R}$、ESS、MCSE	混合の遅さ、未訪問のモード
変分推論	分布族内の最適化された分布	サンプルとの比較	狭すぎる分布族、分解された q の過信

実務ではどの手法を手に取るべきだろうか。パラメータがほんの数個であれば、グリッドを使えばよい。滑らかで単峰な事後分布に対しては、ラプラス近似が高速な第 1 の答えと強力な初期提案分布を与えてくれる。期待値に信頼できるエラーバーが必要なときは、3 つの診断指標と組み合わせた MCMC に頼るのがよい。事後分布の近似をより大きな最適化ループの内部に組み込む必要がある場面では変分推論が威力を発揮するが、分解された分布族は不確実性を過小評価する

ことを念頭に置いておく必要がある。2 つのガウス近似はいずれもそれ単体で完結した精度検証手段を持たないため、可能な限りサンプラと突き合わせて検証すべきである。

スケールについては率直に述べておく価値がある。これら 4 つの手法はいずれも、大規模なニューラルネットワークの全パラメータの事後分布へはそのまま きれいにスケールしない。次元が増えるにつれて重要度重みは崩壊し、ランダムウォーク MCMC は独立サンプル 1 個あたりにますます多くのステップを要するようになり、完全なヘッセ行列は d^2 個の要素を持ち、平均場分布 q は先ほど導出した相関因子の分だけ不確実性を過小評価する。さらに悪いことに、ニューラルネットワークのアーキテクチャ上の対称性によって、隠れユニットのあらゆる置換と符号反転ごとに事後分布のすべてのモードが複製される（演習問題 7）。

大規模なネットワークにおいて、本節の冒頭で立てた問いに対して最も広く使われている答えは、4 つの手法のどれよりも単純である。異なるランダム初期化から同じネットワークを複数回訓練し、予測確率を平均するのである。このような**深層アンサンブル**（deep ensemble）(Lakshminarayanan et al. 2017) は、独立に訓練された少数のネットワークを事後分布からのサンプルの代用とする、事後予測 Equation 36.96 の大雑把なモンテカルロ推定（Monte Carlo estimate）である。各実行は異なるモードに落ち着くため、その平均は複数のモードにまたがることになり、これは単一のラプラス近似や平均場近似にはできない芸当である。典型的に訓練データから遠く離れた領域で各メンバーの予測が食い違うと、平均された予測の確信度は低くなる。ただし、各メンバーは事後分布からのサンプルではなく、等しい重みを付けるべきだという保証もどこにもない。また各メンバーの誤差には相関があるため、メンバーを増やす効果にも限界がある。ペアごとの相関が ρ のとき、平均の分散は単一メンバーの分散の ρ 倍より小さくなることは決してない（36.2 節の演習問題 2）。

それでもなお、本節の中核的なアイデアは、正規化定数を除いてしか分布がわからないあらゆる場面で深層学習の随所に再び現れる。エビデンス下界は、エンコーダネットワークが q を出力する変分オートエンコーダ（variational autoencoder、VAE）(Kingma and Welling 2014)（17.7 節）や離散拡散モデル（17.9 節）を訓練するのに使われる。重要度重みと有効標本サイズによって、方策勾配法は過去の軌跡を再利用できる（15.2 節）。そして勾配誘導型の提案から受理・棄却ステップを取り除いたものは、スコアベース生成モデルを支えるランジュバンサンプラとなる（17.1 節）。最後に、あらゆるサンプリング手法は有限個のサンプルにわたる経験平均を返す。次節では、サンプルが独立であるときに、そのような有限標本平均がその期待値の周りにどれほど密に集中するかを問う。

36.8.6 まとめ

モデルを単一のパラメータベクトルへと訓練することは、その点推定があたかも確実に既知であるかのように扱うことに等しい。訓練データから遠く離れた領域では、もっともらしいパラメータベクトル同士が何桁も食い違うことがあり、事後予測においてそれらを平均すると、稀な事象の確率が数倍変化しうる。その平均を計算するには、正規化定数を除いてしかわからない事後分布に関する積分が必要になる。本節で扱った 4 つの手法はすべて、未知の定数が相殺される比や差のみを扱うことでこの定数を回避している。すなわち、最頻値における局所的な曲率（ラプラス近似）、密度比 $\tilde{p}/q$（棄却サンプリングと重要度サンプリング）、受理比 $\tilde{p}(\theta')/\tilde{p}(\theta)$（メトロポリス・ヘイスティングス法）、そして逆方向 KL ダイバージェンスと定数しか違わない最適化目的関数（変分推論）である。サンプリング手法には組み込みの診断指標（重要度サンプリングにおける重みの ESS、MCMC における分割 $\widehat{R}$、自己相関時間 κ、および MCSE）が備わっている一方、ガウス近似はサンプルと突き合わせて検証しなければならない。大規模なニューラルネットワークに対する完全なベイズ推論（Bayesian inference）は依然として計算困難であるが、エビデンス下界、重要度重み、ランジュバン動力学は、未正規化分布を扱うたびに本書の各所で再び登場する。

36.8.7 演習問題

1. **メトロポリス・ヘイスティングス法の受理確率。** 受理確率 Equation 36.106 は、任意の提案分布を事後分布を目標とするマルコフ連鎖へと変える。
   1. 連続な状態空間では、遷移カーネルは密度部分 $g(\theta' \mid \theta)\alpha(\theta, \theta')$ と、現在の状態 θ における質量 $r(\theta) = 1 - \int g(\theta' \mid \theta)\alpha(\theta, \theta')\, d\theta'$（棄却確率）の点質量（アトム）からなる。このアトムを考慮に入れたうえで、事後分布が定常分布であることを示せ。**ヒント：** 1 ステップ後、アトムは $\pi(\theta')r(\theta')$ だけ点 θ' における密度に寄与する。
   2. 独立サンプラは、現在の状態によらず $\theta' \sim q$ を提案する。このサンプラの受理確率が、Equation 36.102 の重要度重み w を用いて $\min(1, w(\theta')/w(\theta))$ となることを示せ。典型的な提案の重みよりもはるかに大きな重みを持つ状態に到達したあと、この連鎖はどのように振る舞うか。
   3. Barker の規則は、対称な提案を確率 $\tilde{p}(\theta')/(\tilde{p}(\theta) + \tilde{p}(\theta'))$ で受理する。これが詳細釣り合いを満たすことを示せ。本文の 2 状態連鎖に対する遷移行列を書き下し、メトロポリス連鎖の方が両方の状態からより頻繁に移動することを示せ。2 状態連鎖では、状態の任意の関数の自己相関は $\rho_k = \lambda^k$（λ は遷移行列の第 2 固有値）となる。両方の規則について λ と $\kappa = (1 + \lambda)/(1 - \lambda)$ を計算せよ。同じステップ数に対してより小さいモンテカルロ誤差を与えるのはどちらか。**ヒント：** 非対角成分は、Barker の規則では $3/4$ と $1/4$ となるのに対し、メトロポリス法では 1 と $1/3$ となる。
2. [code] **提案分布のスケール。** 関数 `run_chains` は `laplace_cov` のコレスキー因子にスケール（本文中では 1.1）を掛けている。
   1. 各連鎖 12,000 ステップとウォームアップを保ったまま、スケールを $\{0.05, 0.2, 0.5, 1.1, 1.7, 2.4, 3, 10\}$ にわたって変化させよ。各値について、受理率、分割 $\widehat{R}$、および `chain_ess` の最小 ESS を 4 本の連鎖の対数同時密度評価回数 $48{,}000$ で割った値を記録せよ。評価 1 回あたりの ESS を受理率に対してプロットせよ。評価 1 回あたりの ESS がその最大値の 2 分の 1 以上にとどまるのは、どの受理率の範囲か。
   2. d 次元の標準ガウス目標分布と提案分布 $\mathcal{N}(\theta, (\ell^2/d)\mathbf{I})$ に対し、Roberts et al. (1997) は $d \to \infty$ において $\ell \approx 2.38$（このとき受理率は約 0.234）で効率が最大になることを示している。ロジスティック事後分布はラプラス近似に近いため、コレスキー因子によっておおよそ標準ガウス分布に変換される。この結果は $d = 2$ に対してどのスケールを示唆するか。スイープで得られた最良のスケールおよびその受理率とどのように比較されるか。
   3. 本文中の診断指標のうち、スケール 0.05 の実行に異常のフラグを立てるのはどれか。受理率はその中に含まれるか。その実行における事後平均は、それ自身の MCSE を単位としてグリッド平均からどれだけ離れているか。
3. **ラプラス近似とパラメータ化。** コインを投げて表が 9 回、裏が 4 回出たとする。一様事前分布の下で、表の確率 θ の事後分布は $\text{Beta}(10, 5)$ となり、平均は $2/3$、標準偏差は 0.118、中央 95% 区間は $[0.419, 0.872]$ である。
   1. パラメータ θ におけるラプラス近似 Equation 36.100 は、本文で計算したとおり平均 $9/13$、標準偏差 0.128 を持つ。その中央 95% 区間と、$[0, 1]$ の外側に割り当てられる確率を求めよ。
   2. 対数オッズ $\eta = \log(\theta/(1-\theta))$ では、事後密度は $\text{sigmoid}(\eta)^{10}(1 - \text{sigmoid}(\eta))^{5}$ に比例する（ヤコビ行列、1 次元ではヤコビアン $d\theta/d\eta = \theta(1-\theta)$ が表裏それぞれに 1 回分のカウントを加えるためである）。その最頻値が $\eta = \log 2$ であることを示し、そこでの対数密度の曲率を計算し、ラプラス近似の中央 95% 区間を θ に写し戻せ。
   3. 2 つの近似のうち、厳密な事後分布により近いのはどちらか。$\theta > 1/2$ のような事象の厳密な事後確率がパラメータ化に依存しないのに対し、ラプラス近似がパラメータ化に依存するのはなぜか。

4. [code] **穏健化された予測。**●ラプラス近似 Equation 36.100 の下では、テスト入力におけるロジット $a = \mathbf{z}_\star^\top \theta$（ただし $\mathbf{z}_\star = (1, x_\star)^\top$）は、平均 $\mu_a = \mathbf{z}_\star^\top \theta_{\text{MAP}}$、分散 $s_a^2 = \mathbf{z}_\star^\top \mathbf{H}^{-1} \mathbf{z}_\star$ のガウス分布に従う。プロビット近似 $\text{sigmoid}(a) \approx \Phi(\lambda a)$（ただし $\lambda^2 = \pi/8$ であり、Φ は標準正規分布の累積分布関数）を用いると、予測積分 Equation 36.96 は閉じた形に変換される (MacKay 1992)。
   1. $\mathbb{E}[\Phi(\lambda a)] = \Phi(\lambda \mu_a / \sqrt{1 + \lambda^2 s_a^2})$ が $a \sim \mathcal{N}(\mu_a, s_a^2)$ に対して成り立つことを示せ。**ヒント：**$\Phi(\lambda a)$ を $P(U \le \lambda a)$（U は独立な標準正規確率変数）と書け。これから、穏健化された予測 $\text{sigmoid}(\mu_a / \sqrt{1 + \pi s_a^2/8})$ が $\text{sigmoid}(\mu_a)$ と $1/2$ の間にあることを結論づけよ。
   2. `theta_map` と `laplace_cov` から $x_\star \in \{-3, 2, 7\}$ における穏健化された予測を計算し、実行結果として出力されるプラグイン予測および事後予測の値と比較せよ。
   3. 同じ入力において、プラグイン予測 $\text{sigmoid}(\mathbf{z}_\star^\top \theta_{\text{MAP}})$、事後平均における予測 $\text{sigmoid}(\mathbf{z}_\star^\top \mathbb{E}[\theta \mid \mathcal{D}])$、および事後予測 $\mathbb{E}[\text{sigmoid}(\mathbf{z}_\star^\top \theta) \mid \mathcal{D}]$ の 3 つの量を計算せよ。最頻値から平均への変化と、平均における予測から予測の平均への変化という 2 つのステップのうち、予測を $1/2$ から遠ざけるのはどちらで、逆にそこへ近づけるのはどちらか。$x_\star = 2$ では両ステップとも小さいのに、$x_\star = -3$ や $x_\star = 7$ ではそうならないのはなぜか。このことは、過信した予測を穏健化するためにラプラス近似を用いることに対してどのような示唆を与えるか。
5. [code] **重みの ESS の予測。**●本文の命題により、N_{eff}/N は $(\mathbb{E}_q[w])^2 / \mathbb{E}_q[w^2]$ へと、サンプル数 N が増えるにつれて近づく。
   1. 1 次元において正規化された目標分布を $\mathcal{N}(\mu, s_p^2)$、提案分布を $\mathcal{N}(\nu, s_q^2)$ とし、$r = s_q / s_p$ とおく。$s_q^2 > s_p^2/2$ のとき
   $$\frac{\mathbb{E}_q[w^2]}{(\mathbb{E}_q[w])^2} = \frac{r^2}{\sqrt{2r^2 - 1}} \exp\left(\frac{(\mu - \nu)^2}{2 s_q^2 - s_p^2} \right),$$
   が成り立ち、$s_q^2 \le s_p^2/2$ のときはこの比が無限大になることを示せ。どの r が $\mu = \nu$ のときにこの比を最小にするか。
   2. 相関を無視し、グリッドから求めた平均と標準偏差を持つ独立なガウス分布で事後分布を近似して、上の比が各座標の積に分解されるようにせよ。事前分布を提案分布とする 20,000 個のサンプルの ESS を予測し、出力された値と比較せよ。$\tau = 0.5$ と $\tau = 10$ の事前提案分布についても同様に行え。すなわち、モデルのコードセルで `prior_var` $= \tau^2$ と設定し、重要度サンプリングのセルまで再実行して予測値と実測値を比較せよ。公式の中のどの因子が狭い事前分布を不適切な提案分布にし、どの因子が広い事前分布を不適切な提案分布にするか。
   3. 同じ近似の下で、提案分布 $\mathcal{N}(\theta_{\text{MAP}}, 0.3\,\mathbf{I})$ の重みの分散が無限大になるのは実行結果のどの座標か。5,000 個のサンプルでこれを 10 回実行し、ESS、最大の正規化重み、および推定事後平均を記録せよ。推定値がもっともらしい値に見える 1 回の実行結果だけでは、提案分布が適切であることの証拠にならないのはなぜか。
   4. 事前分布からの棄却サンプリングでは、確率 $L(\theta) / \max L$ でサンプルを残す（ここで L は尤度、$\max L$ は最尤推定量における尤度の値であり、線形分離可能なデータでは上限である 1 とする）。元のデータと 10 倍の点数を持つデータで受理率を測定せよ。事後分布の体積から n に伴う減衰を説明し、同じ事前分布からの重要度サンプリングの ESS 比率とその受理率を比較せよ。
6. [code] **エビデンスの推定。**●正規化された提案分布を q とし、$w = p(\mathcal{D}, \theta)/q(\theta)$ とおき、独立なサンプル $\theta_s \sim q$ に対して $\hat{Z}_N = \frac{1}{N} \sum_{s=1}^{N} w(\theta_s)$ とする。
   1. $\hat{Z}_N$ が $p(\mathcal{D})$ の不偏推定量であること、および $\mathbb{E}[\log \hat{Z}_N] \le \log p(\mathcal{D})$ が成り立つことを示せ。また、$\mathbb{E}[\log \hat{Z}_1]$ が q の ELBO Equation 36.110 に等しく、$\mathbb{E}[\log \hat{Z}_N]$ が N に関して減少しないことを示せ (Burda et al. 2016)。

2. 関数 `log_joint` は 2 次元事前分布の正規化定数 $-\log(2\pi\tau^2)$ を省略している。これを加えたうえで、グリッド上の密度を足し合わせてグリッドの 1 マスの面積を掛ける方法、ラプラス提案分布からの 5,000 サンプルを用いた $\widehat{Z}_N$ による方法、およびラプラスエビデンス $\log \tilde{p}(\theta_{\mathrm{MAP}}) + \log 2\pi - \frac{1}{2}\log\det \mathbf{H}$ による方法の 3 通りで $\log p(\mathcal{D})$ を計算せよ。$p(\mathcal{D})$ のラプラス推定値は数パーセント低く、$\log p(\mathcal{D})$ で約 0.04 小さくなる。グリッド計算のセルで見られた事後分布のどの性質がこの符号を説明するか。
3. ラプラス近似と平均場近似のそれぞれから 10^5 個のサンプルを用いて ELBO を推定し、各推定値の MCSE を報告せよ。Equation 36.109 より、$\log p(\mathcal{D})$ と q の ELBO の差は $D_{\mathrm{KL}}(q \| p(\cdot \mid \mathcal{D}))$ である。このダイバージェンスによって 2 つのガウス分布を比較せよ。

7. [code] **2 つのモードを持つ事後分布。**●$y_i \sim \mathcal{N}(\theta^2, 1)$（真の値は $\theta = 1$）に従う 50 個の観測を生成し、θ には事前分布 $\mathcal{N}(0, 4)$ を置く。この尤度は θ に θ^2 を通じてのみ依存する（ちょうど tanh 隠れユニットの入力重み・バイアス・出力重みの符号を同時に反転させてもニューラルネットワークの損失が変わらないのと同様である）。したがって、事後分布は $\theta \mapsto -\theta$ の下で対称となる。
   1. 1 次元グリッド上で事後分布を計算せよ。$\mathbb{E}[\theta \mid \mathcal{D}]$、$\mathbb{E}[\theta^2 \mid \mathcal{D}]$、およびモードにおける事後密度に対する $\theta = 0$ における事後密度の比を報告せよ。
   2. 順方向ダイバージェンス $D_{\mathrm{KL}}(p\|q)$ を最小化するガウス分布が p の平均と分散に一致することを示せ。今回の事後分布に対してこのガウス分布はどこに最頻値を置き、その点は p の下でどれほどもっともらしいか。
   3. 正のモードにおけるラプラス近似、$m = 0.5$ から出発して変分推論のセルの ELBO 上昇法で当てはめた 1 次元ガウス分布、および提案標準偏差 0.2 で $\theta \in \{0.5, 1, 1.5, 2\}$ から出発する 4 本のメトロポリス連鎖という 3 つの局所近似を求めよ。続いて、初期値を $\theta \in \{-2, -0.5, 0.5, 2\}$ に変えて連鎖を再実行せよ。単一のモードしか報告しないのはどの手法か。各実行において、診断指標（連鎖については分割 $\widehat{R}$ と ESS、変分近似については ELBO）のうち問題を明らかにするのはどれか。
   4. 単一モードの近似でも依然として正しく推定できるのはどのような事後期待値か。h 個の tanh 隠れユニットを持つネットワークには、各ユニットに対するこの問題の符号反転に加え、隠れユニットの $h!$ 通りの置換が存在する（演習問題 6）。等方的なガウス事前分布の下で、その事後分布には 1 つのモードの等価なコピーが何個存在するか。このことは、ニューラルネットワークの事後分布を本節のいずれかの手法で近似することについて何を示唆するか。
8. [code] **ロジスティック事後分布の裾。**●●重要度サンプリングにおいて中心極限定理が成り立つには、重みの二次モーメントが有限である必要がある。重要度サンプリングのセルにおけるラプラス提案分布は、通常は高い ESS に達する。以下を通じて、$\tau^2 = $ `prior_var` $= 4$ は事後分布 Equation 36.97 における事前分散である。
   1. 単位ベクトル $\mathbf{v}$ を固定し、$\theta = t\mathbf{v}$ とおく。Equation 36.97 の対数尤度が $-c(\mathbf{v})t + O(1)$（$t \to \infty$）に等しくなることを示せ。ここで $c(\mathbf{v})$ は $|\mathbf{z}_i^\top \mathbf{v}|$ を、$y_i = 1$ かつ $\mathbf{z}_i^\top \mathbf{v} < 0$、または $y_i = 0$ かつ $\mathbf{z}_i^\top \mathbf{v} > 0$ を満たす観測にわたって足し合わせた和である。どのようなデータセットに対して、$c(\mathbf{v}) > 0$ がすべての $\mathbf{v}$ で成り立つか。
   2. これから、ガウス提案分布 $\mathcal{N}(\mathbf{m}, \mathbf{S})$ は $\mathbb{E}_q[w^2] = \infty$（$\mathbf{S}$ のある固有値が $\tau^2/2$ より小さいとき）および $\mathbb{E}_q[w^2] < \infty$（$\mathbf{S}$ のすべての固有値が $\tau^2/2$ を超えるとき）となることを結論づけよ。重要度サンプリングのセルにおける 2 つの提案分布のうち、有限分散の重みを持つのはどちらか。この答えを、事後分布をガウス分布で近似したときに演習問題 5a の公式がラプラス提案分布に対して与える答えと比較せよ。
   3. `laplace_cov` の各固有ベクトルに沿って両方向に調べ、θ_{MAP} からのマハラノビス距離がいくつを超えると $\tilde{p}^2/q$ が θ_{MAP} における値を超えるかを求めよ。ラプラス提案分布からの 1 個のサンプルがそれほど遠くに落ちる確率はどの程度か。その答えと実行時に出力される ESS との整合性を説明せよ。

4. Geweke (1989) は事後最頻値における多変量スチューデント t 近似から重要度密度を構成している。自由度 3、位置 θ_{MAP}、スケール行列 `laplace_cov` を持つスチューデント t 提案分布を用い、その重みが有限の分散を持つことを示せ。20 回の実行にわたり、ESS と裾確率 $P(\theta_1 > \bar{\theta}_1 + 3 s_1 \mid \mathcal{D})$（$\bar{\theta}_1$ と s_1 はグリッドから求めた傾きの平均と標準偏差）の推定値についてラプラス提案分布と比較せよ。基準値には 2 倍の幅を持つグリッド上で事後分布を評価した値を用いよ。また本文の防御的混合分布も試し、なぜその重みが有界になるのかを説明せよ。

9. [code] **ランジュバン提案。●●** メトロポリス調整付きランジュバンアルゴリズムは $\theta' = \theta + \frac{\epsilon^2}{2} \mathbf{C} \nabla \log \tilde{p}(\theta) + \epsilon \, \mathbf{C}^{1/2} \xi$（ただし $\xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$、$\mathbf{C}$ は前処理行列）を提案する。ここでは $\mathbf{C}$ をラプラス共分散 $\mathbf{H}^{-1}$、$\mathbf{C}^{1/2}$ をそのコレスキー因子とせよ。
   1. 提案密度 $g(\theta' \mid \theta)$ と受理規則 Equation 36.106 におけるヘイスティングス因子を書き下せ。この提案分布が対称でないのはなぜか。
   2. `grad_log_joint` を用いてこのサンプラを実装し、$\epsilon \in \{0.5, 1, 1.5\}$ に対して 4 本の連鎖を実行せよ。受理率、分割 $\widehat{R}$、および勾配評価 1 回あたりの ESS を報告し、ランダムウォークにおける対数同時密度評価 1 回あたりの ESS と比較せよ。
   3. 受理ステップを除いた連鎖は未調整ランジュバンアルゴリズムとなる。$\epsilon = 1$ でこれを実行し、平均と標準偏差をグリッドの値と比較せよ。離散化による統計的バイアスとモンテカルロ誤差の 2 つの誤差のうち、メトロポリスステップが取り除くのはどちらか。

36.9 集中不等式

モデルのテスト正解率を報告するとき、それは有限標本に基づく測定値を報告しているにほかならない。36.5 節では、中心極限定理（central limit theorem）を用いてその測定値にエラーバー $\pm 1.96 \, \hat{\sigma} / \sqrt{n}$ を付与した。このガウス近似は手軽で精度も高いことが多いが、厳密に成り立つのは $n \to \infty$ の極限においてのみであり、推定された標準偏差 $\hat{\sigma}$ のプラグインに依存している。それより前の 36.1 節では、任意の有限な n で近似なしに成り立つ上界としてチェビシェフの不等式を見たが、あまりにも緩すぎることがわかった。ここから自然な問いが生じる。手元にある実際の標本サイズで成り立ち、しかもめったに失敗しないことを要求してもなお有用であり続ける保証が欲しいとき、テストセット誤差は真の値からどれだけ離れうるのだろうか。

この隔たりを具体的に見るため、830 個の誤りを 10,000 枚のテスト画像に対して出す分類器、すなわち観測された誤り率が 8.3% の場合を考えよう。チェビシェフの不等式は、n 個の独立な抽出の平均がその期待値（真の平均）から t 以上外れる確率を $\sigma^2 / (n t^2)$ で上から抑える。0/1 の結果の分散は高々 $\sigma^2 \le \frac{1}{4}$ であるため、チェビシェフの不等式は真の誤り率が $\pm 2.2\%$ 以内で 8.3% の周りに収まることを 95% の信頼度で保証する。これはすでにガウス区間より広く、さらに悪いことに高い信頼性を求めると破綻してしまう。100 万回に 1 回しか失敗しない保証を求めると、チェビシェフの不等式が返す答えは $\pm 500\%$ となり、何の情報も与えない。しかし、0/1 の結果の平均が真の平均から大きく外れる真の確率は、n に関して**指数関数的に**減衰し、$1/n$ のような遅い減衰にはならない。この指数減衰を捉えた上界は、$\pm 1.4\%$（信頼度 95%）および $\pm 2.7\%$（100 万回に 1 回の失敗確率）という答えを与える。

こうした鋭い上界の背後にある核心的な工夫は驚くほど単純である。マルコフの不等式を平均そのものではなくその指数関数に適用することで、独立な項の和を積へと変換するのである。Sergei Bernstein は 1920 年代にこの種の上界を発展させ、Herman Chernoff は 1952 年に統計的検定がどれほど速く信頼できるようになるかを測るためにこの手法を用い、Wassily Hoeffding は 1963 年に機械学習の随所で用いられる簡潔な定式化を与えた (Hoeffding 1963)。

実務家はなぜ、100 万回に 1 回しか失敗しないような保証を求めることがあるのだろうか。実際には、ただ 1 つのモデルだけを評価することはまれであり、学習アルゴリズムは多数の候補を比

較して最良のものを残す。そのうちの**どれ 1 つとして**偶然よく見えているのではないと保証するには、個々の候補に対する失敗確率を極めて小さく抑える必要があり、現実的な標本サイズでそれを可能にするのは指数関数的に縮小する裾だけである。このテーマは 36.10 節で本格的に展開する。本節ではまず、指数型の裾上界を用いて目標の精度を達成するためのテストセットのサイズを決定する。続いて、まったく同じ道具立てを $\mathbb{R}^d$ 内の確率ベクトルの各座標に適用し、2 つの際立った幾何学的事実を明らかにする。すなわち、$\mathbb{R}^{784}$ の標準ガウスベクトルはほぼ確実に 28 の数パーセント以内の長さを持ち、独立な 2 つのランダムな方向はほぼ直交するのである。これら 2 つの事実は、ニューラルネットワークの初期化と高次元の埋め込みの双方を支えている。

本節の結果のうち、次の 3 つは記憶にとどめておく価値がある。チェルノフ法 Equation 36.112、ヘフディングの信頼区間 Equation 36.116、そしてユークリッドノルムの集中 Equation 36.121 である。

```
rng = np.random.default_rng()
```

36.9.1 指数型の裾上界

ガウス信頼区間が狭いのは、ガウス分布の裾が $e^{-z^2/2}$ のように減衰するからであるが、この減衰レートは中心極限定理に由来するため、大きな n の極限でしか保証されない。ここで欲しいのは、それと同じ速い減衰と、任意の有限標本サイズで成り立つ保証との両立である。そのような上界を構築する前に、チェビシェフの不等式が真の値からどれほどかけ離れているかを確認しておこう。

36.9.1.1 チェビシェフの減衰レート

チェビシェフの上界とは、単にマルコフの不等式（36.1 節）を二乗偏差に適用したものにすぎない。標本平均 $\bar{X}$ を考えよう。これは n 個の独立な抽出の平均であり、各抽出は平均 μ と有限の分散 σ^2 を持つとする。二乗偏差の期待値は $\mathbb{E}[(\bar{X}-\mu)^2] = \mathrm{Var}(\bar{X}) = \sigma^2/n$ であるから、マルコフの不等式より次式が得られる。

$$P(|\bar{X}-\mu| \ge t) \le \frac{\sigma^2}{n t^2}.$$

ここでは、外れる確率は $1/n$ の速さでしか減少しない。失敗確率を 1 から 10^{-6} まで（偏差 t を固定したまま）引き下げるには、標本サイズを 100 万倍に増やさなければならない。

実際に外れる確率はどれほど小さいのだろうか。厳密に計算できる最も単純な設定は公平なコインである。$\hat{p}$ を n 回の独立なコイン投げで表が出た割合とし、$\hat{p}$ が $\frac{1}{2}$ から $t = 0.1$ 以上外れる確率を求めよう。1 回の試行の分散は $\frac{1}{4}$ であるから、チェビシェフの上界は $0.25/(0.01\,n) = 25/n$ となる。厳密な確率は二項確率の和であり、$n = 5{,}000$ でも 0 にアンダーフローしないよう対数空間で評価する。

```
def exact_tail(n, t):
    """P(|p_hat - 1/2| >= t) for n fair flips, a binomial sum in log space."""
    k = np.arange(n + 1)
    log_pmf = n * np.log(0.5) + np.concatenate(
        ([0.0], np.cumsum(np.log((n - k[1:] + 1) / k[1:]))))
    in_tail = log_pmf[np.abs(k / n - 0.5) >= t - 1e-12]  # fp-safe boundary
    m = in_tail.max()
```

```
    return np.exp(m) * np.exp(in_tail - m).sum()  # stable log-sum-exp
```

$n = 1{,}000$ のとき、チェビシェフの上界は 40 回の実験のうち 1 回の割合で外れる可能性を許容する。出力の表を読む前に、そのようなずれが実際にはどのくらいの頻度で起きるか予想してみてほしい。

```
t = 0.1
print(f'{"n":>6} {"exact":>10} {"Chebyshev":>10}')
for n in (10, 100, 1000, 5000):
    print(f'{n:6d} {exact_tail(n, t):10.2e} {0.25 / (n * t**2):10.2e}')
```

```
     n      exact  Chebyshev
    10   7.54e-01   2.50e+00
   100   5.69e-02   2.50e-01
  1000   2.73e-10   2.50e-02
  5000   1.30e-45   5.00e-03
```

$n = 10$ では厳密な裾確率は 0.75 である一方、上界は 2.5 と自明な値にとどまる。$n = 100$ では上界は 0.25 まで下がり、厳密値 0.057 の約 4 倍となる。しかしその先では、2 つの曲線は劇的に乖離していく。$n = 1{,}000$ では厳密な裾確率が 2.7×10^{-10} であるのに対し、チェビシェフの上界は依然として 0.025 を許容しており、8 桁もの開きがある。$n = 5{,}000$ になると、その差は 42 桁にまで広がる。n を 10 倍にするごとにチェビシェフの上界は単に 10 分の 1 になるだけだが、厳密な裾確率はコインを 100 回投げるごとに一定の桁数ずつ小さくなり、対数スケールでは直線に沿って下降していく。

残念ながら、分布についてさらなる仮定を置かない限り、チェビシェフの上界をこれ以上引き締めることはできない。36.1 節で見たとおり、3 点上に台を持つある分布では等号が成立するため、平均と分散のみに依拠するいかなる上界も一般にはこれを改善できないからである。しかし、コイン投げには 2 つの有限モーメントよりはるかに豊かな構造がある。とる値が有界であるため、高次のモーメントもすべて小さく抑えられるのである。すべてのモーメントを一度に活用するために、モーメント母関数に目を向けよう。

36.9.1.2 チェルノフ法

高次のモーメントをどう活用するかを見るため、マルコフの不等式とチェビシェフの不等式に共通する仕組みに注目しよう。関数 g がいたるところで指示関数 $\mathbf{1}\{x \geq t\}$ の上側にある、すなわち指示関数の**優関数**（majorant）であるとする。期待値をとれば直ちに $P(X \geq t) = \mathbb{E}[\mathbf{1}\{X \geq t\}] \leq \mathbb{E}[g(X)]$ が得られる。マルコフの不等式は線形な優関数 $g(x) = x/t$ を非負の X に対して選び、チェビシェフの不等式は二次の優関数 $g(x) = (x/t)^2$ を（中心化された偏差 $X - \mu$ に適用して）選ぶ。ここで任意の $\lambda > 0$ に対し、指数関数 $e^{\lambda(x-t)}$ も実数直線全体で指示関数の上側にあることに注目しよう。Figure 36.43 の (a) は、これら 3 つの優関数を非負の X について比較したものである。上界は単に $g(X)$ の平均（期待値）であるから、その鋭さを決めるのは、関数 g の値が確率変数 X の確率質量の集中している領域でどれだけ小さく保たれるかである。

この指数関数の期待値には標準的な名前がついている。確率変数 X の**モーメント母関数**（moment generating function、MGF）とは、期待値が有限となる範囲で $\mathbb{E}[e^{\lambda X}]$ を λ の関数とみなしたものである。0 を中心とする公平なコイン $Z \in \{-\frac{1}{2}, \frac{1}{2}\}$ の場合、それは単に $\frac{1}{2}(e^{\lambda/2} + e^{-\lambda/2}) = \cosh(\lambda/2)$ となる。この MGF は任意の λ に対して有限になる（X が有界である限り成り立つ）が、任意の $\lambda > 0$ に対して無限大に発散してしまうこともある（スチューデントの t 変数のように右裾が多項式的にしか減衰しない場合）。そのような裾の重い確率変数には、以下の指数関数による手法は適用できない。

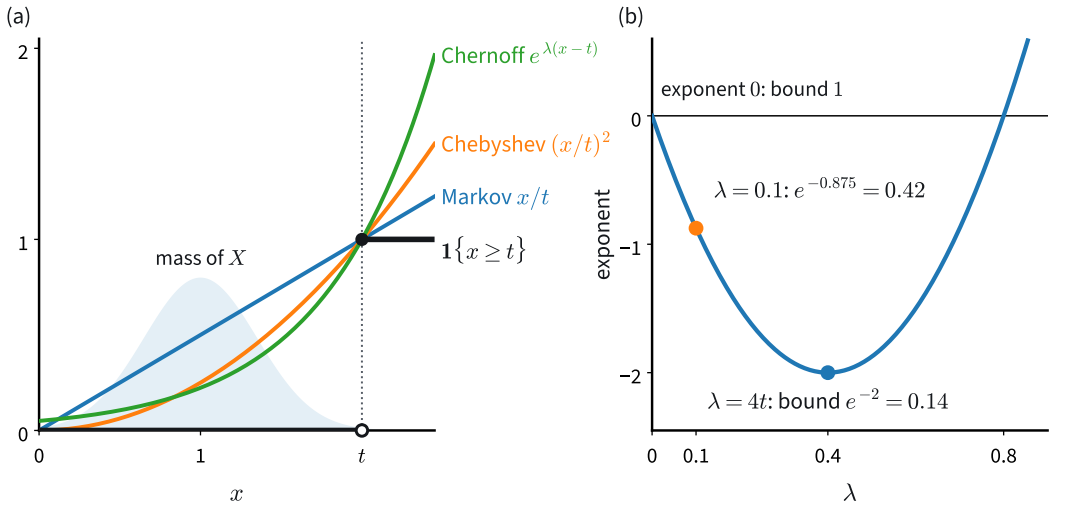


Figure 36.43: 裾上界は指示関数の上側にある関数の期待値として与えられる。(a) 閾値 $t = 2$ に対する指示関数 $\mathbf{1}\{x \geq t\}$（黒）と、その上側にある 3 つの関数：マルコフの x/t、チェビシェフの $(x/t)^2$、およびチェルノフの $e^{\lambda(x-t)}$（$\lambda = 1.5$）。網掛けの山は X の確率質量が存在する領域を表す。各上界はその質量のもとでの関数の平均であり、λ を大きくするとその領域で指数関数は小さくなる。(b) チェルノフ指数 $-\lambda nt + n\lambda^2/8$（公平なコイン $n = 100$ 枚の平均、$t = 0.1$）。任意の $\lambda > 0$ が有効な上界を与え、放物線の最小点 $\lambda = 4t$ において最良の上界 e^{-2} が得られる。

非負の確率変数 $e^{\lambda X}$ にマルコフの不等式を適用すれば、1 行で上界が得られる。$x \mapsto e^{\lambda x}$ は $\lambda > 0$ のとき狭義単調増加であるため、事象 $\{X \geq t\}$ は $\{e^{\lambda X} \geq e^{\lambda t}\}$ と一致する。したがって、次式が成り立つ。

$$P(X \geq t) = P\big(e^{\lambda X} \geq e^{\lambda t}\big) \leq e^{-\lambda t}\,\mathbb{E}\big[e^{\lambda X}\big], \qquad \text{hence} \qquad P(X \geq t) \leq \inf_{\lambda > 0} e^{-\lambda t}\,\mathbb{E}\big[e^{\lambda X}\big]. \tag{36.112}$$

$\lambda > 0$ をどう選んでも有効な上界が得られ、下限（infimum）をとる操作はその中で最も鋭いものを選び出しているにすぎない。この 2 段階の処方箋が**チェルノフ法**（Chernoff method）である。

指数関数が適切な優関数であるのには、同じくらい決定的な第 2 の理由がある。$S = X_1 + \cdots + X_n$ が独立な確率変数の和であるとき、$e^{\lambda S} = \prod_{i=1}^{n} e^{\lambda X_i}$ は独立な因子の積になり、独立な確率変数の積の期待値はそれぞれの期待値の積になる。

$$\mathbb{E}\big[e^{\lambda S}\big] = \prod_{i=1}^{n} \mathbb{E}\big[e^{\lambda X_i}\big].$$

このように、和のモーメント母関数は各項の MGF の積へと因数分解されるため、チェルノフ上界の対数は n 個の項の和に分解され、n に対して線形にスケールする。多項式の優関数はこの性質を持たない。実際、$\mathbb{E}[S^4]$ は 4 次モーメントの積には分解されない。残る課題は、和を構成する 1 つの項の MGF を上から抑えることだけである。

36.9.1.3 公平なコイン

公平なコインの場合を最後まで具体的に計算してみよう。この最も単純なケースに、一般の証明のすべてのステップがすでに含まれているからである。中心化されたコイン投げ $Z = X - \tfrac{1}{2}$ は値 $\pm\tfrac{1}{2}$ を等確率でとるため、その MGF は $\mathbb{E}[e^{\lambda Z}] = \cosh(\lambda/2)$ である。$\cosh u$ と $e^{u^2/2}$ のテイラー級数を項別に比較すると、次式が得られる。

$$\cosh u = \sum_{k\geq 0} \frac{u^{2k}}{(2k)!} \leq \sum_{k\geq 0} \frac{(u^2/2)^k}{k!} = e^{u^2/2},$$

これは $(2k)! \geq 2 \cdot 4 \cdots (2k) = 2^k k!$ が成り立つからである。実際、$(2k)!$ の偶数因子だけでもすでに $2^k k!$ になり、残る奇数因子はすべて少なくとも 1 以上である。$u = \lambda/2$ とおけば、1 回の中心化されたコイン投げは $\mathbb{E}[e^{\lambda Z}] \leq e^{\lambda^2/8}$ をすべての λ に対して満たすことがわかる。比較すると、ガウス分布 $\mathcal{N}(0, \sigma^2)$ は（積分内で平方完成することにより）$\mathbb{E}[e^{\lambda X}] = e^{\lambda^2\sigma^2/2}$ を満たすため、中心化されたコインの MGF は標準偏差 $\tfrac{1}{2}$ のガウス分布の MGF を決して超えない。

ここで n 回の独立なコイン投げを考える。事象 $\hat{p} - \tfrac{1}{2} \geq t$ は $\sum_{i=1}^{n} Z_i \geq nt$ と同値であるから、チェルノフ法と積の法則より次式が得られる。

$$P(\hat{p} - \tfrac{1}{2} \geq t) \leq e^{-\lambda n t} \prod_{i=1}^{n} \mathbb{E}[e^{\lambda Z_i}] \leq \exp\!\Big(-\lambda n t + \frac{n\lambda^2}{8}\Big).$$

指数部は λ に関する単純な放物線である。Figure 36.43 の (b) は $n = 100$、$t = 0.1$ の場合をプロットしたもので、このとき指数部は $-10\lambda + 12.5\lambda^2$ に等しい。任意の $\lambda > 0$ が有効な上界を与え、たとえば $e^{-0.875} \approx 0.42$ が $\lambda = 0.1$ で得られる。最も鋭い上界は放物線の頂点 $\lambda = 4t$ で達成され、そこで指数部は $-4nt^2 + 2nt^2 = -2nt^2$ へと簡約される。対称性により下側の裾も同じ上界に従うため、両側の裾を足し合わせると次式が得られる。

$$P(|\hat{p} - \tfrac{1}{2}| \geq t) \leq 2e^{-2nt^2}.$$

これにより、t 以上外れる確率は nt^2 に関して指数関数的に減衰する。$n = 100$、$t = 0.1$ では $2e^{-2} \approx 0.27$ となり、$n = 1{,}000$ では $2e^{-20} \approx 4.1 \times 10^{-9}$ まで低下する。チェビシェフの上界 0.025 と比べれば劇的な改善である。

なぜ指数型の優関数はこれほど効果的に働くのだろうか。閾値 $S = nt$ において関数 $e^{\lambda(S-nt)}$ は 1 に等しい一方、確率質量の大部分が集まっている平均 $S = 0$ においては $e^{-\lambda nt}$ まで縮小し、n に関して指数関数的に小さくなる。λ を無限大へと飛ばせない理由は、この関数が $S > 0$ で急激に増大するため、上方向の揺らぎが過大な重みを持ってしまうことにある。$e^{n\lambda^2/8}$ で上から抑えられる MGF の積がそのペナルティを測っており、最適な $\lambda = 4t$ が両者の厳密なバランスをとっている。

以上の導出全体が、コインのただ 1 つの性質だけに依拠していた点に注目しよう。すなわち、中心化された 1 回の試行の MGF が $e^{\lambda^2/8}$、つまりコインの値域の半分である標準偏差 $\tfrac{1}{2}$ のガウス分布の MGF で抑えられるという性質である。ヘフディングの補題はこの 1 ステップを一般化し、区間 $[a, b]$ に値をとる**すべての**確率変数が、$(b-a)/2$ で $\tfrac{1}{2}$ を置き換えた同じ上界に従うことを示す。

36.9.1.4 ヘフディングの補題とヘフディングの不等式

テストセットの損失が公平なコインと同じ振る舞いをすることはめったにない。8.3% という誤り率は大きく偏ったコインのように振る舞い、クリップされた損失は 0 と 1 だけでなく連続的な値をとる。そこで、公平なコインで得た指数型の上界が**任意の**有界な確率変数に対して成り立つようにしたい。これを実現する補題が本節における主要な技術的ステップであり、その証明は分散に関する 1 つの初等的な事実に基づいている。

命題（有界な確率変数の分散） Y **が** $[a,b]$ **に値をとるとき、** $\mathbf{Var}(Y) \le (b-a)^2/4$ **が成り立つ。等号は、** Y **が値** a **と** b **をそれぞれ確率** $\tfrac{1}{2}$ **でとるときに成立する。**

証明： 平均は二乗偏差の期待値を最小化する。実際、$\mathbb{E}[(Y-c)^2] = \mathrm{Var}(Y) + (\mathbb{E}[Y]-c)^2$ が中心 c の選び方によらず成り立つからである。ここで $c = (a+b)/2$ ととれば、$[a,b]$ のすべての点は中点から $(b-a)/2$ 以内の距離にあるため、次式が得られる。

$$\mathrm{Var}(Y) \le \mathbb{E}\Big[\Big(Y - \frac{a+b}{2}\Big)^2\Big] \le \Big(\frac{b-a}{2}\Big)^2.$$

Y が両端点 a と b に等しい確率質量を置くとき、2 つの不等号はいずれも等号になる。■

$[0,1]$ 上の公平なコインはまさにこの極端なケースにあたり、分散 $\tfrac{1}{4}$ を持つ。この分散の上界を手にすれば、ヘフディングの補題を証明できる。

命題（ヘフディングの補題） X **を** $\mathbb{E}[X]=0$ **を満たし** $[a,b]$ **に値をとる確率変数とする。このとき、すべての** $\lambda \in \mathbb{R}$ **に対して次式が成り立つ。**

$$\mathbb{E}\big[e^{\lambda X}\big] \le \exp\!\left(\frac{\lambda^2 (b-a)^2}{8}\right). \tag{36.113}$$

証明： 対数 MGF である $\psi(\lambda) = \log \mathbb{E}[e^{\lambda X}]$ の増大が、放物線 $\lambda^2(b-a)^2/8$ を超えないことを示したい。X は有界であるため、ψ はいたるところ有限かつ滑らかであり、期待値記号の下で微分できる。$\lambda = 0$ において $\psi(0) = \log 1 = 0$ であり、さらに次が成り立つ。

$$\psi'(\lambda) = \frac{\mathbb{E}[X e^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]}, \qquad \text{so} \qquad \psi'(0) = \mathbb{E}[X] = 0.$$

指数分布族の対数分配関数（36.4 節）とまったく同様に、ψ の二階導関数は分散になる。**傾斜**（tilted）分布 q_λ を、X の分布に $e^{\lambda x}$ の重みを掛けて正規化したものとして定義しよう。q_λ のもとでは $\psi'(\lambda)$ は X の平均であり、もう一度微分すると次式が得られる。

$$\psi''(\lambda) = \frac{\mathbb{E}[X^2 e^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]} - \left(\frac{\mathbb{E}[X e^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]}\right)^2 = \mathrm{Var}_{q_\lambda}(X).$$

重要なのは、指数傾斜は各点の重みを付け替えるだけであり、確率質量を $[a,b]$ の外へ移すことは決してないという点である。したがって、先ほどの分散の上界はすべての q_λ（すなわちすべての λ）に対して成り立ち、$\psi''(\lambda) \le (b-a)^2/4$ を与える。あとはラグランジュの剰余項を用いたテイラーの定理によって証明が完結する。ある ξ が 0 と λ の間に存在して、次式が成り立つ。

$$\psi(\lambda) = \psi(0) + \lambda\,\psi'(0) + \frac{\lambda^2}{2}\,\psi''(\xi) \le \frac{\lambda^2 (b-a)^2}{8},$$

両辺の指数をとれば Equation 36.113 が得られる。$\blacksquare$

言葉で言えば、有界で中心化された任意の確率変数のモーメント母関数は、標準偏差 $(b-a)/2$ のガウス分布のモーメント母関数以下となる。中心化されたコインでは $b-a=1$ であるから、この補題はテイラー級数によって証明した上界 $\cosh(\lambda/2) \le e^{\lambda^2/8}$ を直ちに再現する。このたった 1 つの部品が揃えば、コインに対する証明は独立な有界確率変数の任意の平均へとそのまま引き継がれる。

命題（ヘフディングの不等式） **$X_1, \dots, X_n$ を $X_i \in [a,b]$ を満たす独立な確率変数とし、$\bar{X} = \frac{1}{n}\sum_i X_i$ とする。このとき、すべての $t > 0$ に対して次式が成り立つ。**

$$P\big(|\bar{X} - \mathbb{E}[\bar{X}]| \ge t\big) \le 2\exp\!\left(-\frac{2nt^2}{(b-a)^2}\right). \tag{36.114}$$

証明： $Z_i = X_i - \mathbb{E}[X_i]$ とおく。各 Z_i は平均 0 で長さ $b-a$ の区間に値をとるため、すべての項にヘフディングの補題が適用できる。任意の $\lambda > 0$ に対し、チェルノフ法 Equation 36.112 と積の法則より次式が得られる。

$$\begin{aligned} P(\bar{X} - \mathbb{E}[\bar{X}] \ge t) &= P\Big(\sum_{i} Z_i \ge nt\Big) \le e^{-\lambda n t} \prod_{i=1}^{n} \mathbb{E}\big[e^{\lambda Z_i}\big] \\ &\le \exp\!\left(-\lambda n t + \frac{n\lambda^2 (b-a)^2}{8}\right). \end{aligned}$$

これは $(b-a)^2$ で 1 を置き換えただけで、コインの場合に最小化したものと同じ放物線である。最小値は $\lambda = 4t/(b-a)^2$ で達成され、そのとき指数部は $-2nt^2/(b-a)^2$ に等しい。$-Z_i$ に同じ議論を適用すれば下側の裾も同じ量で抑えられ、両側の裾を足し合わせることで係数 2 が生じる。$\blacksquare$

このように、n 個の独立な有界変数の標本平均が期待値から t 以上外れる確率は、nt^2 に関して指数関数的に減衰する（ここで t は値域の幅 $b-a$ を単位として測られる）。この上界が何を必要と**しない**かに注目しよう。ガウス性も、分散の推定値も、漸近極限も不要である。求められるのは有界性と独立性だけであり、任意の有限な n に対して成り立つ。ただし独立性は本質的な要件であり、マルコフ連鎖の相関を持つサンプルにはこの上界は適用できない。

証明を振り返ると、実はすべての区間の長さが等しい必要はどこにもなかったことに気づくだろう。独立な中心化確率変数が $Y_i \in [a_i, b_i]$ を満たすなら、ヘフディングの補題による上界の積をとることで $n(b-a)^2$ が $\sum_i (b_i - a_i)^2$ に置き換わり、同じ二次関数の最小化から次式が得られる。

$$P\Big(\Big|\sum_{i} Y_i\Big| \ge s\Big) \le 2\exp\!\Big(-\frac{2s^2}{\sum_i (b_i - a_i)^2}\Big). \tag{36.115}$$

この異なる値域に対する不等式は、のちほど高次元におけるランダムな方向を解析する際に活用する。

ヘフディングの不等式を実用的なテストセットのサイズへと換算してみよう。$[0,1]$ に値をとる損失では $b-a=1$ であるから、失敗確率が δ 以下となる条件は $n \ge \log(2/\delta)/(2t^2)$ である。誤り率を誤差 $t = 0.01$ 以内、失敗確率 $\delta = 0.05$ で推定するには、$n \ge \log 40 / (2 \cdot 10^{-4}) = 18{,}444.4$、すなわち 18,445 個のテスト例が必要になる。これは第 3.5 章で挙げた数値であり、漸近的なガウス近似が示唆する 9,604 個のおよそ 2 倍にあたる。

同じことだが、Equation 36.114 の右辺を δ とおいて t について解けば、実務で使われる信頼区間の形式が得られる。すなわち、少なくとも確率 $1-\delta$ で次式が成り立つ。

$$\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \le (b-a)\sqrt{\frac{\log(2/\delta)}{2n}}. \tag{36.116}$$

ガウス区間 $\hat{\mu} \pm 1.96\,\hat{\sigma}/\sqrt{n}$ と同様、区間の半幅は $1/\sqrt{n}$ のレートで縮小するが、こちらには厳密な有限標本保証が伴っている。

830 個の誤りを 10,000 枚の画像に対して出した分類器に $\delta = 0.05$ で立ち返ると、ヘフディングの不等式は $\pm 1.36\%$ を与えるのに対し、ガウス区間は $\pm 1.96\sqrt{0.083 \cdot 0.917/10^4} = \pm 0.54\%$ となる。両者の違いは定数のみにある。ヘフディングの不等式は経験標準偏差 $\hat{\sigma} = 0.28$ を最悪ケースの値 $(b-a)/2 = 0.5$ に置き換え、ガウス分布の分位点 1.96 を $\sqrt{2\log(2/\delta)} = 2.72$ に置き換えるため、区間が 2.5 倍広くなるのである。データの実際のばらつきがその値域よりはるかに小さいときには、ヘフディングの不等式はどうしても保守的になるが、この隔たりはすぐ後でベルンシュタインの不等式が埋めることになる。

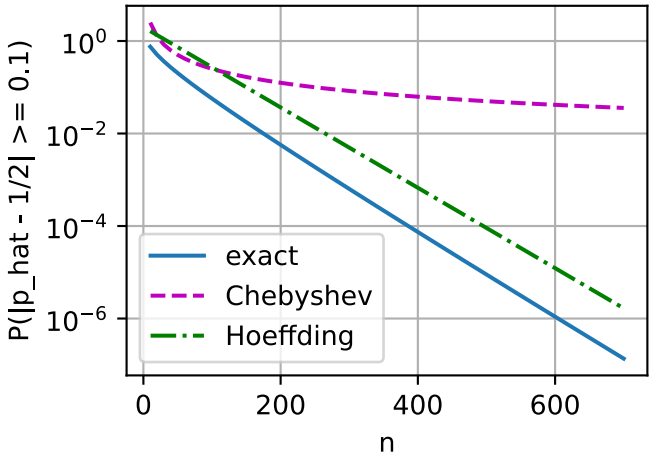
ヘフディングの補題における定数 $\frac{1}{8}$ をさらに小さく改善できるのではないかと疑問に思うかもしれない。しかしそれは不可能である。確率 $\frac{1}{2}$ ずつを $[a,b]$ の両端点に置く確率変数が、実際にこの定数 $\frac{1}{8}$ を λ の 2 次の項において達成するからである（演習問題 1）。

36.9.1.5 数値で見る公平なコイン

ヘフディングの上界 $2e^{-2nt^2}$ が、公平なコインの $t = 0.1$ における厳密な裾確率にどれほど迫るかを確かめてみよう。両者の値とその比を表示し、さらにチェビシェフの上界と合わせて n に対する推移を対数スケールでプロットする。

```
print(f'{"n":>6} {"exact":>10} {"Hoeffding":>10} {"ratio":>8}')
for n in (10, 100, 1000, 5000):
    hoeffding = 2 * np.exp(-2 * n * t**2)
    print(f'{n:6d} {exact_tail(n, t):10.2e} {hoeffding:10.2e} '
          f'{hoeffding / exact_tail(n, t):8.1f}')

ns = np.arange(10, 701, 10)
d2l.plot(ns, [np.array([exact_tail(n, t) for n in ns]),
              0.25 / (ns * t**2), 2 * np.exp(-2 * ns * t**2)],
         'n', 'P(|p_hat - 1/2| >= 0.1)',
         legend=['exact', 'Chebyshev', 'Hoeffding'], yscale='log')
```

```
     n      exact  Hoeffding    ratio
    10   7.54e-01   1.64e+00      2.2
   100   5.69e-02   2.71e-01      4.8
  1000   2.73e-10   4.12e-09     15.1
  5000   1.30e-45   7.44e-44     57.2
```



$n = 10$ ではどちらの上界も 1 を超え、情報を持たない。$n = 100$ では、実はチェビシェフの 0.25 の方がヘフディングの 0.27 よりわずかに鋭い。標本サイズが小さい領域では先頭の定数が効いてくるため、厳密な分散 $\frac{1}{4}$ を知っているチェビシェフの不等式が有利になるからである。しかし n が大きくなると、指数減衰レートが完全に支配的になる。ヘフディングの上界は厳密な裾確率に対し、15 倍以内に $n = 1{,}000$ で収まり、57 倍以内に $n = 5{,}000$ でもとどまっている。

対数プロット上では、厳密な裾確率とヘフディングの上界はほぼ平行な直線を描く。コインを 1 回投げるごとに、ヘフディングの上界の対数は $2t^2 = 0.02$ ずつ減少するのに対し、厳密な裾確率は約 0.0201 ずつ減少する。このわずかに急な傾きは、カルバック・ライブラー・ダイバージェンス $D_{\mathrm{KL}}(0.6 \| 0.5) = 0.6 \log 1.2 + 0.4 \log 0.8$、すなわち表の確率が 0.6 の偏ったコインと公平なコインとの間のダイバージェンスにほかならない（37.3 節）。ベルヌーイ変数の厳密な MGF を用いてチェルノフ法を適用すれば、この厳密な減衰レートが導かれる（演習問題 3）。

もちろん、n がごく小さいときには、いかなる不等式も存在しない証拠を作り出すことはできない。本章の導入部で登場した、表が 9 回（13 回中）出たコインを思い出そう。これは $t = 2.5/13$ だけ $\frac{1}{2}$ からずれた事象である。このようなずれに対するヘフディングの上界は $2e^{-2 \cdot 13 \cdot (2.5/13)^2} = 2e^{-0.96} \approx 0.77$ であり、何も棄却できない。厳密な両側二項確率でさえ 0.27 である。13 回のコイン投げでは、公平なコインである可能性を到底排除できないのである。

36.9.1.6 準ガウス確率変数と準指数確率変数

誤り率 8.3% の分類器に対するヘフディングの区間がガウス区間より 2.5 倍広くなったのは、0/1 損失の値域だけを使い、その小さな分散を無視しているからであった。**ベルンシュタインの不等式**は、その両方を取り入れる。

命題（ベルンシュタインの不等式） $X_1, \dots, X_n$ **を独立な確率変数とし、**$\mathbf{Var}(X_i) \le \sigma^2$ **および** $|X_i - \mathbb{E}[X_i]| \le M$ **を満たすとする。このとき、すべての** $t > 0$ **に対して次式が成り立つ。**

$$P\big(|\bar{X} - \mathbb{E}[\bar{X}]| \ge t\big) \le 2 \exp\left(- \frac{n t^2}{2(\sigma^2 + M t/3)} \right). \tag{36.117}$$

ここでも証明はチェルノフ法 (Boucheron et al. 2013) であり、分散と値域の双方を反映した MGF の上界と組み合わせることで得られる（詳細は演習問題 7 で導出する）。分母に現れる 2 つの領域に注目しよう。t が小さいときは σ^2 が $Mt/3$ を支配し、上界は**真の**分散 σ^2 を持つガウス

分布の裾のように振る舞う。一方、t が大きいときは 1 次の項 $Mt/3$ が支配的になり、裾は t に関して指数関数的に減衰するにとどまり、ガウス分布のように t^2 では減衰しない。

分散を考慮することは、実務上どれほど役に立つのだろうか。誤り率 2% の分類器を $n = 10{,}000$ 個のテスト例で評価する場合を考えよう。各 0/1 損失は分散 $\sigma^2 = 0.02 \cdot 0.98$ を持ち、中心化された偏差は $M = 1$ を満たす。Equation 36.117 の右辺を δ とおくと、t に関する二次方程式 $nt^2/(2(\sigma^2 + t/3)) = \log(2/\delta)$ が得られる。以下では $\delta = 0.05$ において、誤り率 8.3% と 2% の両方についてこの二次方程式を解き、ヘフディングの区間およびガウス区間と比較する。ベルンシュタインの区間については、（観測された誤り率を分散に代入した）プラグイン推定値と、出力の直後で説明する完全保証版の 2 通りを計算する。

```
def half_widths(p, n, delta=0.05, z=1.96):
    """Hoeffding, Bernstein (plug-in, guaranteed), Gaussian half-widths."""
    L = np.log(2 / delta)
    hoeffding = np.sqrt(L / (2 * n))
    c = 2 * L / 3  # Bernstein with M = 1: n t^2 - c t - 2 L var = 0
    bernstein = lambda var: (c + np.sqrt(c**2 + 8 * n * L * var)) / (2 * n)
    plug_in = t = bernstein(p * (1 - p))
    for _ in range(20):  # true rate at most p + t: a fixed point in t
        t = bernstein((p + t) * (1 - p - t))
    return hoeffding, plug_in, t, z * np.sqrt(p * (1 - p) / n)
```

`half_widths(p, 10000)` を 8.3% と 2.0% の両方の誤り率で評価すると、4 つの手法すべてにおける 95% 信頼区間の半幅を次のように比較できる。

```
for p in (0.083, 0.02):
    hoeffding, plug_in, bernstein, gauss = half_widths(p, 10000)
    print(f'error {p:.1%}: Hoeffding ±{hoeffding:.2%}, Bernstein '
          f'±{plug_in:.2%} (plug-in) ±{bernstein:.2%} (guaranteed), '
          f'Gaussian ±{gauss:.2%}')
```

```
error 8.3%: Hoeffding ±1.36%, Bernstein ±0.76% (plug-in) ±0.79% (guaranteed), Gaussian ±0.54%
error 2.0%: Hoeffding ±1.36%, Bernstein ±0.39% (plug-in) ±0.43% (guaranteed), Gaussian ±0.27%
```

誤り率 2% では、ベルンシュタインのプラグイン半幅は $\pm 0.39\%$ まで縮まり、ヘフディングの $\pm 1.36\%$ と比べて約 3.5 倍鋭く、さらに 1.4 倍以内に漸近ガウス区間の $\pm 0.27\%$ へと迫っている。誤り率 8.3% では改善幅はやや小さくなるものの依然として大きく、$\pm 1.36\%$ が $\pm 0.76\%$ まで狭まる。

厳密に言えば、これら 2 つのプラグイン値（$\pm 0.39\%$ と $\pm 0.76\%$）はまだ真の有限標本保証にはなっていない。上のコードでは観測された誤り率 $\hat{p}$ を分散 $p(1-p)$ に代入したが、ベルンシュタインの不等式が必要とするのは**真の**分散の上界であり、これはガウス区間に対して向けたプラグインの問題点とまったく同じである。幸い、この修正にコストはほとんどかからない。上界が成り立っている限り真の誤り率は高々 $\hat{p} + t$ であるから、最悪ケースの分散 $(\hat{p} + t)(1 - \hat{p} - t)$ を用いて同じ方程式を解けばよい。（コードセルで行っているように）この不動点方程式を反復すると、半幅はわずかに広がるだけで、完全に保証された $\pm 0.43\%$ と $\pm 0.79\%$ が得られる。これにより冒頭の問いに対する鋭い答えが定まる。830 個の誤りを 10,000 枚の画像に対して出した分類器では、真の誤り率は約 0.8% 以内で 8.3% の周りに 95% の信頼度で収まるのである。実用上のモデルは通常数パーセントの誤り率で動作するため、分散を考慮した上界は大きな違いをもたらす。

一歩引いて見直すと、ヘフディングの証明が依拠していたのは和の各項のただ 1 つの性質、すなわちモーメント母関数がガウス分布のモーメント母関数で上から抑えられることだけであった。

この性質に名前をつけておけば、同じ証明を非有界な確率変数へと拡張できるとともに、どこから先でその議論が成り立たなくなるかも明確になる。確率変数 X が**準ガウス**（sub-Gaussian）で**分散プロキシ**（variance proxy）σ^2 を持つとは、その中心化された MGF が $\mathcal{N}(0,\sigma^2)$ の MGF を決して超えないことをいう。

$$\mathbb{E}\big[e^{\lambda(X-\mathbb{E}[X])}\big] \le \exp\left(\frac{\lambda^2\sigma^2}{2}\right) \qquad \text{for all } \lambda \in \mathbb{R}. \tag{36.118}$$

任意の準ガウス確率変数は、少なくともガウス分布と同程度に軽い裾を持つ。実際、$\lambda = t/\sigma^2$ としてチェルノフ法を適用すれば $P(|X-\mathbb{E}[X]| \ge t) \le 2e^{-t^2/(2\sigma^2)}$ が得られる。ただし、ここでの σ^2 はあくまで**プロキシ**であり、必ずしも真の分散ではないことに留意しよう。Equation 36.118 の両辺を λ の 2 次まで比較すると、このプロキシは決して $\mathrm{Var}(X)$ より小さくならないことがわかり、むしろはるかに大きくなる場合がある。たとえば 0/1 の結果で誤り率が 2% のとき真の分散は 0.0196 であるが、ヘフディングの補題が与えるプロキシは $\frac{1}{4}$ であり、まさにそのためにヘフディングの信頼区間はあれほど広くなったのである。

代表的な 3 つの分布族が準ガウスである。ガウス分布 $\mathcal{N}(\mu,\sigma^2)$ は等号成立のもとでプロキシ σ^2 を持ち、$[a,b]$ 上の任意の確率変数はヘフディングの補題によりプロキシ $(b-a)^2/4$ を持ち、一様なランダム符号 $\xi \in \{-1,+1\}$ はプロキシ 1 を持つ（先ほどの級数比較より $\mathbb{E}[e^{\lambda\xi}] = \cosh\lambda \le e^{\lambda^2/2}$ が成り立つためである）。何より好都合なことに、準ガウス性は平均操作のもとで保たれる。$X_1,\dots,X_n$ が独立でプロキシ σ^2 を持つなら、積の法則により $\bar{X}$ はプロキシ σ^2/n を持ち、したがって $P(|\bar{X}-\mathbb{E}\bar{X}| \ge t) \le 2e^{-nt^2/(2\sigma^2)}$ が成り立つ。

しかし、性質のよい確率変数がすべて準ガウスであるわけではなく、すぐ次に必要となる量がまさにその反例となっている。ガウスベクトルの長さの二乗は標準ガウス変数の二乗和であるが、1 つの標準ガウス変数 x について平方完成を行うと次式が得られる。

$$\mathbb{E}\big[e^{\lambda(x^2-1)}\big] = \frac{e^{-\lambda}}{\sqrt{1-2\lambda}} \qquad \text{for } \lambda < \tfrac{1}{2},$$

一方、すべての $\lambda \ge \frac{1}{2}$ に対してこの期待値は無限大に発散する。MGF が $\lambda = \frac{1}{2}$ で発散してしまうため、どのような分散プロキシをとっても Equation 36.118 が x^2 に対して成り立つことはない。その右裾は $e^{-t/2}$ のように減衰し、ガウス型の上界で抑えるには遅すぎるからである。

それでもなお、制限された $\lambda = 0$ の近傍の区間上ではガウス型の MGF 上界が**実際に**成り立つ。確率変数 X が**準指数**（sub-exponential）でパラメータ (ν,α) を持つとは、次式を満たすことをいう。

$$\mathbb{E}\big[e^{\lambda(X-\mathbb{E}[X])}\big] \le \exp\left(\frac{\lambda^2\nu^2}{2}\right) \qquad \text{for } |\lambda| \le \frac{1}{\alpha}. \tag{36.119}$$

ここで ν^2 は $\lambda = 0$ の近傍における局所的な分散プロキシとして働き、$1/\alpha$ は許容区間の半幅を定め、チェルノフ法で選べる λ の大きさに上限を課す。36.9.2.1 節では、x^2 が $\nu = 2$、$\alpha = 4$ の準指数となることを見る。準指数確率変数の平均にチェルノフ法を適用すると、最適化はこの許容区間の境界を境に自然と 2 つに分かれる。初読時には以下の証明を読み飛ばしてもよい。次項で用いるのは命題の主張のみである。

命題（準指数分布の裾上界） $X_1,\dots,X_n$ **を独立な** (ν,α)**-準指数確率変数とする。このとき、すべての** $t > 0$ **に対して次式が成り立つ。**

$$P\big(|\bar{X} - \mathbb{E}[\bar{X}]| \ge t\big) \le 2\exp\!\left(-\frac{n}{2}\min\!\Big(\frac{t^2}{\nu^2},\, \frac{t}{\alpha}\Big)\right). \tag{36.120}$$

証明： 任意の $0 < \lambda \le 1/\alpha$ に対し、チェルノフ法と積の法則により $P(\bar{X} - \mathbb{E}[\bar{X}] \ge t)$ は $\exp(-\lambda n t + n\lambda^2\nu^2/2)$ で上から抑えられる。$t \le \nu^2/\alpha$ のときは、制約なし最小点 $\lambda = t/\nu^2$ が許容区間の内側に収まり、$e^{-nt^2/(2\nu^2)}$ を与える。$t > \nu^2/\alpha$ のときは、その最小点が許容区間の外に出るため、λ を許容される最大値 $\lambda = 1/\alpha$ に固定する。すると指数部は $-nt/\alpha + n\nu^2/(2\alpha^2)$ となり、これは高々 $-nt/\alpha + nt/(2\alpha) = -nt/(2\alpha)$ である（$\nu^2/\alpha < t$ による）。両方の領域を合わせると、指数部は高々 $-\frac{n}{2}\min(t^2/\nu^2, t/\alpha)$ となる。下側の裾も $\lambda \in [-1/\alpha, 0)$ を用いてまったく同様に従う。$\blacksquare$

これで準ガウス変数の平均との対比は極めて明快になった。有界または準ガウスな確率変数の平均は、あらゆるスケールでガウス型の裾 e^{-ct^2} を持つ。これに対し、準指数確率変数の平均は ν^2/α 以下の適度な偏差に対しては同じガウス型の裾を持つが、さらに外側ではより遅い指数型の裾 $e^{-nt/(2\alpha)}$ へと遷移する。

36.9.2 高次元における確率

ここまでの上界は、単一の標本平均を保証するものであった。今度は $\mathbb{R}^d$ 内の d 個の独立な座標を持つ確率ベクトルを考えよう。その長さの二乗も、他のベクトルとの内積も、それら d 個の座標にわたる和である。したがって、高次元空間の幾何学は集中不等式によって支配される。ここから読み始める場合でも、本節の前半から必要な結果は 2 つだけである。長さの二乗に対する準指数分布の裾上界 Equation 36.120 と、内積に対する異なる値域のヘフディング上界 Equation 36.115 である。

36.9.2.1 薄い球殻

標準ガウスベクトル $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{784})$（$28 \times 28$ の画像と同じ数の座標を持つベクトル）をサンプリングしてみよう。その確率密度は原点で最大になる。しかし驚くべきことに、$\mathbf{x}$ が原点から距離 14（典型的な長さの半分）以内に落ちる確率は約 10^{-110}、すなわち 784 個の標準ガウス変数の二乗和が高々 196 以下になる確率にすぎない。コンピュータ上でどれほど多くのサンプルを抽出しても、最頻値の近くに落ちる点は 1 つとして現れないだろう。その代わり、典型的なサンプルはノルム 28.0 ± 0.7 を持つ。

その理由は、$\|\mathbf{x}\|^2 = \sum_{i=1}^{d} x_i^2$ が d 個の独立な項（各平均は 1）の和であり、したがって $\|\mathbf{x}\|^2/d$ が単なる標本平均になっていることにある。x_i^2 は非有界であるためヘフディングの不等式は適用できないが、先ほど導いた準指数上界を用いれば見事に扱える。

命題（ノルムの集中） $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ **とする。すべての** $0 < \varepsilon \le 1$ **に対して次式が成り立つ。**

$$P\!\left(\Big|\frac{\|\mathbf{x}\|}{\sqrt{d}} - 1\Big| \ge \varepsilon\right) \le 2\exp\!\left(-\frac{d\,\varepsilon^2}{8}\right). \tag{36.121}$$

証明： 議論は 3 つの短いステップで進む。第 1 に、べき級数の比較（演習問題 4）から、中心化された二乗の MGF である $e^{-\lambda}/\sqrt{1-2\lambda}$ は次式を満たす。

$$\mathbb{E}\big[e^{\lambda(x_i^2 - 1)}\big] \le e^{2\lambda^2} \qquad \text{for } |\lambda| \le \tfrac{1}{4}. \tag{36.122}$$

Equation 36.119 の用語で言えば、x_i^2 は $\nu = 2$、$\alpha = 4$ の準指数である。第 2 に、準指数分布の裾上界 Equation 36.120 を $n = d$、$t = \varepsilon$ として適用すると次式が得られる。

$$P\Big(\Big|\frac{\|\mathbf{x}\|^2}{d} - 1\Big| \ge \varepsilon\Big) \le 2\exp\Big(-\frac{d}{2}\min\Big(\frac{\varepsilon^2}{4}, \frac{\varepsilon}{4}\Big)\Big) = 2e^{-d\varepsilon^2/8} \qquad \text{for } \varepsilon \le 1.$$

第 3 に、二乗ノルムからノルムそのものへと移る。$z = \|\mathbf{x}\|/\sqrt{d} \ge 0$ とおく。$|z - 1| \ge \varepsilon$ が成り立つときは常に、$|z^2 - 1| = |z - 1|\,(z + 1) \ge \varepsilon$ も成り立つ（$z + 1 \ge 1$ による）。したがって、ノルムに関する事象は二乗ノルムに関する事象に含まれ、その上界をそのまま引き継ぐ。■

要するに、標準ガウスベクトルの長さが $\sqrt{d}$ から一定の割合以上外れる確率は、次元 d に関して指数関数的に減衰する。なお、$\varepsilon \le 1$ という制限は二**乗**ノルムを経由して評価したことに伴う見かけ上の制約にすぎない。二乗ノルムの上側の裾は $\varepsilon = 1$ を超えると $e^{-d\varepsilon/8}$ の指数減衰にとどまるが（演習問題 4）、ノルム自体はあらゆるスケールでガウス型の裾を保つ。実際、ノルムが $(1+\varepsilon)\sqrt{d}$ を超えると二乗ノルムは $(1 + 2\varepsilon + \varepsilon^2)d$ を超え、その確率は高々 $e^{-d\min(s, s^2)/8}$（ただし $s = 2\varepsilon + \varepsilon^2$）であり、$s \ge \varepsilon^2$ かつ $s^2 \ge 4\varepsilon^2$ であるから、これは高々 $e^{-d\varepsilon^2/8}$ で抑えられる。

この典型的なノルムが形作る球殻の厚さはどの程度だろうか。$\mathbb{E}[x^4] = 3$ であるから、各座標の二乗は $\mathrm{Var}(x^2) = 3 - 1 = 2$ を満たし、$\mathrm{Var}(\|\mathbf{x}\|^2) = 2d$ となる。平方根を d の周りで線形化すると $\|\mathbf{x}\| \approx \sqrt{d} + (\|\mathbf{x}\|^2 - d)/(2\sqrt{d})$ となるため、$\|\mathbf{x}\|$ の標準偏差は近似的に $\sqrt{2d}/(2\sqrt{d}) = 1/\sqrt{2} \approx 0.71$ となり、次元によらない。驚くべきことに、$\mathbb{R}^d$ における標準ガウス分布は半径 $\sqrt{d}$ の薄い球殻（thin shell）上に集中し、その厚さは d がどれほど大きくなっても定数オーダーのままである。

なぜ確率質量のすべてが最頻値からこれほど離れた場所に集まるのだろうか。同時確率密度 $(2\pi)^{-d/2}e^{-\|\mathbf{x}\|^2/2}$ は $\mathbf{x}$ にその半径 $r = \|\mathbf{x}\|$ を通じてのみ依存する一方、半径 r の球の表面積は r^{d-1} に比例して増大する。密度に表面積を掛けると、半径 $\|\mathbf{x}\|$ の分布は $r^{d-1}e^{-r^2/2}$ に比例することがわかる。r が大きくなるにつれて、体積因子 r^{d-1} は爆発的に増大する一方で密度 $e^{-r^2/2}$ は急落し、両者の積は $(d-1)\log r - r^2/2$ の導関数が 0 になる点、すなわち $r = \sqrt{d-1}$ でピークを迎える。Figure 36.44 の (a) は、この体積と密度のせめぎ合いを $d = 20$ において示している。

ノルム $\sqrt{d}$ の典型的な点における点ごとの密度は、原点での値の $e^{-d/2}$ 倍（極小の 10^{-170}、$d = 784$ のとき）にすぎないが、半径 $\sqrt{d}$ にはそれを補って余りある表面積があるため、ほぼすべての確率質量がそこに存在する。(b) は $\|\mathbf{x}\|$ の厳密な密度を $d = 2$ から $d = 512$ まで示したものであり、ピークは $\sqrt{d}$ に従って外側へ移動していく一方、その幅は $1/\sqrt{2}$ の近くに留まり続けている。

なお、ここで導いた上界は厳密な数値までは再現しないものの、d に関する指数スケーリングを正しく捉えている。ノルムが 14 未満となる事象は $d = 784$ において $\varepsilon = \frac{1}{2}$ に対応し、このとき Equation 36.121 が与える値は $2e^{-784/32} = 2e^{-24.5} \approx 5 \times 10^{-11}$ であり、厳密値 10^{-110} とは差がある。分母の定数 8 は緩いが、d に関する指数減衰という定性的な結論は的を射ている。

36.9.2.2 ランダムな方向の近似直交性

33.1 節では、固定された単位ベクトルと $\mathbb{R}^d$ 内のランダムな方向のなす角のコサインが分散 $1/d$ を持つことを見た。これをチェビシェフの不等式と組み合わせても、$1/(d\varepsilon^2)$ のオーダーの弱い裾評価しか得られなかった。集中不等式はこれを指数型の裾へと鋭くし、それによって同時にほぼ直交しうるランダムな方向が何本あるかを数えられるようになる。

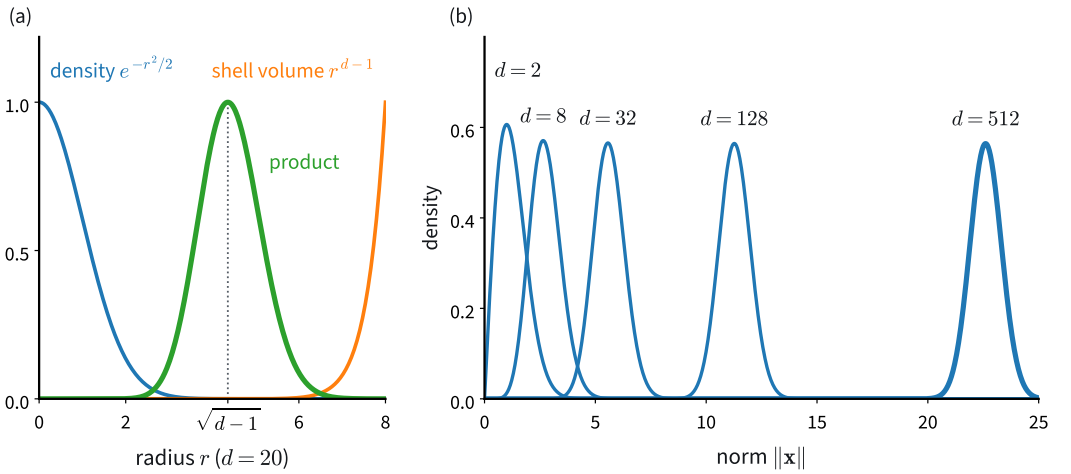


Figure 36.44: 薄い球殻。(a) $d = 20$ において、ガウス密度 $e^{-r^2/2}$ は半径 r とともに減少し、球殻の体積因子 r^{d-1} は半径 r とともに増大する。両者の積である $\|\mathbf{x}\|$ の密度は、密度そのものが最大となる原点から遠く離れた $\sqrt{d-1}$ でピークに達する（各曲線は最大値が 1 になるようスケールしてある）。(b) $\|\mathbf{x}\|$ の厳密な密度（$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$、$d = 2, 8, 32, 128, 512$）。中心は $\sqrt{d}$ に従って外側へ移動し、幅は $1/\sqrt{2}$ の近くに保たれる。

ランダム符号ベクトルを用いると最も見通しよく計算できる。任意の単位ベクトル $\mathbf{u}$ を固定し、独立な一様ランダム符号 $x_i \in \{-1, +1\}$ を抽出する。各積 $u_i x_i$（内積 $\langle \mathbf{x}, \mathbf{u} \rangle$ の各項）は平均 0 であり、区間 $[-|u_i|, |u_i|]$（長さ $2|u_i|$）に収まる。これらの値域を異なる値域のヘフディング上界 Equation 36.115 に代入すると、分母は $4 \sum_i u_i^2 = 4$ となり、次式が得られる。

$$P\big(|\langle \mathbf{x}, \mathbf{u} \rangle| \geq s\big) \leq 2e^{-s^2/2}. \tag{36.123}$$

ランダム符号ベクトルと任意の固定された単位ベクトルとの内積は、$\pm|u_i|$ の寄与の大部分が打ち消し合うため、ガウス型の裾を伴って定数オーダーにとどまる。一方、$\mathbf{x}$ のノルムは $\sqrt{d}$ である。したがって $s = \varepsilon\sqrt{d}$ とおけば、直ちに $P(|\cos(\mathbf{x}, \mathbf{u})| \geq \varepsilon) \leq 2e^{-d\varepsilon^2/2}$ が得られる。標準ガウスベクトル $\mathbf{x}$ に対しても同じ振る舞いが成り立つ。内積 $\langle \mathbf{x}, \mathbf{u} \rangle$ は任意の単位ベクトル $\mathbf{u}$ に対してそれ自体が標準ガウス変数となるため、まったく同じ裾上界 $2e^{-s^2/2}$ に従い、さらに薄い球殻の性質により $\|\mathbf{x}\|$ は $\sqrt{d}$ の数パーセント以内に保たれるからである。

指数型の裾が手に入れば、もう 1 つの道具である**ユニオンバウンド**を使うことで**多数の**ペアを同時に制御できる。ユニオンバウンドとは、複数の事象が独立であるかどうかにかかわらず、そのうち少なくとも 1 つが起こる確率は個々の事象の確率の和以下になるという性質である。m 本の独立なランダム符号ベクトルをとろう。これらは $m(m-1)/2 < m^2/2$ 個の異なるペアを作り、任意の 1 ペアが $|\cos| \geq \varepsilon$ となる確率は高々 $2e^{-d\varepsilon^2/2}$ である。ユニオンバウンドにより、**いずれかの**ペアが ε を超える確率は高々 $m^2 e^{-d\varepsilon^2/2}$ となる。したがって、確率 $1 - \delta$ 以上ですべてのペアのコサインが ε 未満に収まるための条件は $m \leq \sqrt{\delta}\, e^{d\varepsilon^2/4}$ となる。高次元では、**指数関数的に**多くのほぼ直交するベクトルを $\mathbb{R}^d$ 内に詰め込むことができるのである。

具体的に $d = 784$、$\delta = 0.05$ を代入してみよう。この緩い上界でさえ、およそ 570 本の方向について全ペアで $|\cos| < 0.2$ となることを保証し、およそ 10^7 本の方向について全ペアで $|\cos| < 0.3$ となることを保証する。しかもこれらは全ペアにわたる最悪ケースの保証であり、

典型的な 1 ペアの $|\cos|$ はそれよりはるかに小さく、$\sqrt{2/(\pi d)} \approx 0.028$ 程度、すなわち標準偏差 $1/\sqrt{d}$ のガウス変数の絶対値の平均となる。

もちろん、ニューラルネットワークが学習した埋め込みは独立な確率ベクトルではないが、まさにそのためにこの計算が役に立つ。$d = 784$ において、0.3 というコサイン類似度は、独立なランダム方向が生み出す値より標準偏差（$1/\sqrt{784} = 0.036$）の 8 倍以上も上にある。したがって、ランダムな方向は、訓練済みの埋め込みに現れる真の幾何学的構造を測るための帰無モデル（null model）として機能する。

36.9.2.3 数値で見る薄い球殻

理論的な上界に含まれる定数は緩いため、実際の数値を直接測定してみよう。先ほどの接線近似の議論は $\|\mathbf{x}\|$ の標準偏差が $1/\sqrt{2} \approx 0.71$ 付近にあらゆる次元 d でとどまると予測し、ガウス近似は $|\cos|$ の平均を $\sqrt{2/(\pi d)}$ と予測する。以下では 10,000 組の標準ガウスベクトルのペアを $d = 2, 20, 200, 2000$ でサンプリングし、両方の予測を検証する。なお、1 つの行だけは予測と一致しないことに注目してほしい。

```
for d in (2, 20, 200, 2000):
    x, y = rng.standard_normal((2, 10000, d))
    nx = np.linalg.norm(x, axis=1)
    cos = (x * y).sum(1) / (nx * np.linalg.norm(y, axis=1))
    print(f'd = {d:4d}: mean ||x||/sqrt(d) = {(nx / np.sqrt(d)).mean():.3f}, '
          f'sd ||x|| = {nx.std():.2f}, mean |cos| = {np.abs(cos).mean():.4f}'
          f' vs {np.sqrt(2 / (np.pi * d)):.4f}')
```

```
d =    2: mean ||x||/sqrt(d) = 0.890, sd ||x|| = 0.66, mean |cos| = 0.6367 vs 0.5642
d =   20: mean ||x||/sqrt(d) = 0.987, sd ||x|| = 0.71, mean |cos| = 0.1808 vs 0.1784
d =  200: mean ||x||/sqrt(d) = 0.999, sd ||x|| = 0.72, mean |cos| = 0.0566 vs 0.0564
d = 2000: mean ||x||/sqrt(d) = 1.000, sd ||x|| = 0.71, mean |cos| = 0.0180 vs 0.0178
```

予測どおり、ノルムの標準偏差は 0.65 付近（$d = 2$）から始まり、それ以降はノルム自体が約 1.3 から 45 へと増大しても 0.7 のすぐ近くに落ち着く。漸近公式と一致しない唯一の行は $d = 2$ であり、そこでの $|\cos|$ の平均は約 0.64 である。平面内では 2 つの等方的なベクトルのなす角は一様分布に従うため $\mathbb{E}|\cos\theta| = 2/\pi$ となるのに対し、$\sqrt{2/(\pi d)}$ は高次元近似だからである。$d = 20$ 以降では、実測値（約 0.18、0.057、0.018）は公式と数パーセント以内で一致する。

さらに 2 つの問いを立てると、この幾何学がいっそう鮮明になる。第 1 に、典型的なノルムの 10% 以内、すなわち球殻 $(1 \pm 0.1)\sqrt{d}$ の内側に確率質量のどれだけの割合が収まるだろうか。第 2 に、ユークリッド距離は依然として近い点と遠い点を区別できるだろうか。この分離度を測るため、クエリ点の**コントラスト** $(d_{\max} - d_{\min})/d_{\min}$ を 200 個のランダムな点への距離にわたって計算し、100 個のクエリにわたるコントラストの中央値を報告する。

```
eps = 0.1
for d in (2, 20, 200, 2000):
    x = rng.standard_normal((10000, d))
    ratio = np.linalg.norm(x, axis=1) / np.sqrt(d)
    shell = np.mean(np.abs(ratio - 1) <= eps)          # mass in the 10% shell
    query, points = x[:100], x[100:300]                # 100 queries, 200 points
    dist = np.linalg.norm(query[:, None] - points[None], axis=2)  # (100, 200)
    contrast = (dist.max(1) - dist.min(1)) / dist.min(1)
    print(f'd = {d:4d}: mass in (1 ± 0.1) sqrt(d) = {shell:6.1%}, '
```

```
            f'median contrast = {np.median(contrast):.3f}')
```

```
d =    2: mass in (1 ± 0.1) sqrt(d) =  14.6%, median contrast = 35.638
d =   20: mass in (1 ± 0.1) sqrt(d) =  46.8%, median contrast = 1.114
d =  200: mass in (1 ± 0.1) sqrt(d) =  95.8%, median contrast = 0.275
d = 2000: mass in (1 ± 0.1) sqrt(d) = 100.0%, median contrast = 0.077
```

$d = 2$ では、10% の球殻は確率質量の約 15% しか捉えないが、$d = 20$ では約 47%、$d = 200$ では約 95% を捉える。この 95% という値は先ほどの接線近似による見積もりと一致する。すなわち、$d = 200$ では比 $\|\mathbf{x}\|/\sqrt{d}$ の標準偏差が $1/\sqrt{2d} = 0.05$ となるため、± 0.1 は平均の両側に標準偏差 2 つ分の幅を張るからである。$d = 2000$ になると、10,000 個のサンプルすべてがこの球殻の内側に落ちる。その一方で、距離コントラストの中央値は約 30（$d = 2$、実行ごとに約 25 から 40 の間で変動）から、約 1.2、0.27、そして最終的には 0.08 へと急激に縮小する。$d = 2000$ では、典型的なクエリに対し、200 個のランダムな点のうち最も遠い点であっても、最近傍の点より 10% 未満しか離れていない。

36.9.2.4 深層学習への帰結

深層学習における 3 つの標準的な実践は、この高次元幾何学に直接依拠している。

初期化：LeCun 初期化 (LeCun, Leon Bottou, et al. 1998) は $d \times d$ 重み行列の各成分を分散 $1/d$ で独立にサンプリングし、He 初期化 (He et al. 2015) は分散 $2/d$ を ReLU ネットワークに対して用いる（4.4 節）。これらのスケールの古典的な導出は、重みのランダムな抽出全体にわたる期待値のもとで行われる。たとえば LeCun のスケールでは、各活性化前の値 $\langle \mathbf{w}, \mathbf{x} \rangle$ は分散 $\|\mathbf{x}\|^2/d$ を持ち、各座標が単位分散を持つ入力に対してはノルムの集中によってこれが 1 の近くに保たれる。しかし実際には、無限個のネットワークの平均ではなく**ただ 1 つの**ネットワークを初期化して訓練する。その期待値が手元の 1 回の抽出を実際にいつよく記述するのかを教えてくれるのが、集中不等式である。

He 初期化された幅 d の単一の ReLU 層では、活性化値の二乗ノルムの比 $\rho = \|\mathbf{h}_k\|^2/\|\mathbf{h}_{k-1}\|^2$ は平均 1、分散 $5/d$ を持つ（演習問題 5）。したがって、1 つの層は二乗ノルムをおよそ $\sqrt{5/d}$、すなわち約 10%（$d = 512$ のとき）摂動させる。K 層にわたると、これらの乗法的な揺らぎは対数スケールで加算され、$\log(\|\mathbf{h}_K\|^2/\|\mathbf{h}_0\|^2)$ の分散はおよそ $5K/d$ になる。ゆえに、期待分散に基づくヒューリスティックが典型的なネットワークをよく記述するのは $K \ll d$ の範囲に限られる。$K = 50$、$d = 512$ では対数の分散は約 0.5 に達し、初期化された典型的なネットワークの二乗ノルムはその時点で上下いずれかの方向に 2 倍ほどずれてしまう。

距離と最近傍：独立な 2 つの標準ガウスサンプルに対し、その差は $\mathbf{x} - \mathbf{y} \sim \mathcal{N}(\mathbf{0}, 2\mathbf{I})$ を満たすため、ノルムの集中が $(\mathbf{x}-\mathbf{y})/\sqrt{2}$ にそのまま適用できる。すなわち、各ペア間の距離は $(1 \pm \varepsilon)\sqrt{2d}$ の範囲に、高々 $2e^{-d\varepsilon^2/8}$ の例外確率を除いて収まる。方向に対して用いたのと同じユニオンバウンドを適用すれば、n 個の点の間のペアワイズ距離は**すべて**同時にその狭い帯域内に収まることになり、そのための条件 $d \geq (8/\varepsilon^2)\log(n^2/\delta)$ の閾値は n に対して対数的にしか増大しない。あらゆる距離がほぼ等しくなると、先ほどのコントラスト実験が示したとおり、最近傍と最遠傍の差は相対的にごくわずかなマージンしか持たなくなる。実際に学習された埋め込みが意味のある距離コントラストを保っているのは、実データがはるかに低次元の構造の近くに分布しているからにほかならない。裏を返せば、この同じ計算から強力な道具が得られる。k 次元へのランダム射影は、元の空間の次元によらず、n 個の点すべてのペアワイズ距離を k が $\log n$ のオーダーに達した時点で保存するのである（演習問題 6）。

潜在コード：潜在コード $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ 上で訓練された生成デコーダは、ノルムが $\sqrt{d}$ に近い入力ばかりを目にする。事前分布の平均（ノルム 0）も、独立な 2 つの潜在コードを結ぶ直線の弦の中点（ノルムが $\sqrt{d/2}$ 付近）も、デコーダが一度も訓練されていない空っぽの内部領域の奥深

くに位置してしまう。このため実務では、Figure 36.45 に示すように、潜在コード間の補間を直線ではなく球面上の弧に沿って行うのが一般的である（演習問題 2）。

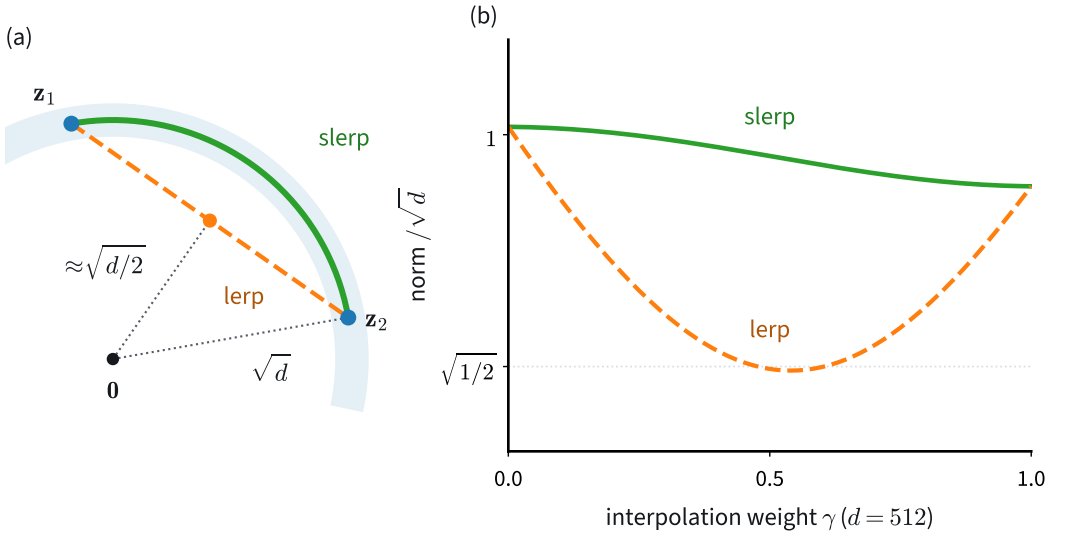


Figure 36.45: 潜在コード $\mathbf{z}_1, \mathbf{z}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ 間を補間する 2 つの方法。(a) 両コードとも半径 $\sqrt{d}$ の薄い球殻（網掛け）内にあり、ほぼ直交している。直線の弦（lerp）は内部を横切り、その中点のノルムは約 $\sqrt{d/2}$ まで落ち込むが、球面上の弧（slerp）は球殻内にとどまる。(b) 各経路に沿ったノルムを $\sqrt{d}$ で割った値（$d = 512$ におけるランダムな 1 ペア）。

最後に、1 つの重要な制約が残っている。本節で導いたすべての上界が保証するのは、データが抽出される**前**に固定された単一の量である。これに対し、学習アルゴリズムはデータを観察した**後**で多数の候補の中から最良のモデルを選び出すが、多数の公平なコインの中から最も成績のよかったものを選べば、それは必ず偏っているように見える。36.10 節では、このようなモデル選択を織り込むために集中不等式をどのように拡張するかを示す。

36.9.3 まとめ

n 個の独立な項の平均は、有限標本サイズ n においてその期待値からどれだけ離れうるのだろうか。チェビシェフの不等式が与える裾確率は $1/n$ の速さでしか減衰せず、めったに失敗しない保証が必要なときには遅すぎる。チェルノフ法はマルコフの不等式を $e^{\lambda X}$ に適用することで、独立和のモーメント母関数を積へと変換したうえで λ について最適化する。これをヘフディングの補題と組み合わせると、指数型の裾 $2e^{-2nt^2}$ が $[0,1]$ 内の損失に対して得られ、半幅 $\sqrt{\log(2/\delta)/(2n)}$ の有限標本信頼区間が導かれる。さらに、値域に比べて分散が小さいときには、ベルンシュタインの不等式がこの区間を大幅に引き締める。同じ考え方を確率ベクトルの d 個の座標に適用すると、標準ガウス分布が半径 $\sqrt{d}$、厚さ約 $1/\sqrt{2}$ の薄い球殻上に集中すること、そして指数関数的に多くのランダムな方向が互いにほぼ直交することが示される。以上の結果から、テストセットのサイズをどう設計すべきか、初期化のヒューリスティックが訓練される単一のネットワークに対していつ成り立つか、そして高次元の埋め込みにおけるコサイン類似度や距離をどう解釈すべきかが明らかになる。

36.9.4 演習問題

1. **準ガウスなランダム符号：**ランダム符号 ξ は値 -1 と $+1$ をそれぞれ確率 $\tfrac{1}{2}$ でとるため、$\mathbb{E}[e^{\lambda \xi}] = \cosh \lambda \le e^{\lambda^2/2}$ を満たす。
   1. チェルノフ法 Equation 36.112 を用い、平均 $\bar{\xi}$（n 個の独立なランダム符号の標本平均）が $P(|\bar{\xi}| \ge t) \le 2e^{-nt^2/2}$ を満たすことを導け。
   2. ヘフディングの補題 Equation 36.113 を $[a,b] = [-1,1]$ で適用しても ξ の分散プロキシが得られる。これを上界 $\cosh\lambda \le e^{\lambda^2/2}$ と比較せよ。さらに $\log\cosh\lambda = \lambda^2/2 - \lambda^4/12 + \cdots$ を $\lambda = 0$ の周りで展開し、すべての有界な確率変数に対して成り立つ限り、$\tfrac{1}{8}$ より小さいいかなる定数でも Equation 36.113 の $\tfrac{1}{8}$ を置き換えられない理由を説明せよ。
2. **潜在空間の補間：**生成モデルが潜在コード $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$（$d = 512$）をデコードするとする。独立な 2 つの潜在コードを $\mathbf{z}_1, \mathbf{z}_2$ とし、それらの線形補間を $\mathbf{z}(\gamma) = (1-\gamma)\mathbf{z}_1 + \gamma \mathbf{z}_2$（$\gamma \in [0,1]$）とする。
   1. $\mathbf{z}(\gamma)$ の分布と $\|\mathbf{z}(\gamma)\|$ の近似値を求めよ。ノルムが最小になるのはどの γ においてか。
   2. Equation 36.121 を用いて、中点 $\mathbf{z}(\tfrac{1}{2})$ が少なくとも $0.9\sqrt{d}$ 以上のノルムを持ち、典型的なコードが存在する球殻 $(1 \pm 0.1)\sqrt{d}$ に到達する確率の上界を求めよ。次に、厳密な確率 $P(\chi^2_{512} \ge 2 \cdot 0.81 \cdot 512)$（自由度 512 のカイ二乗変数に対する上側確率、`scipy.stats.chi2.sf`）を計算せよ。上界は何桁ほど緩いか。
   3. 球面補間 (White 2016) は、弦を次の弧に置き換える。
   $$\tilde{\mathbf{z}}(\gamma) = \frac{\sin((1-\gamma)\theta)}{\sin\theta}\,\mathbf{z}_1 + \frac{\sin(\gamma\theta)}{\sin\theta}\,\mathbf{z}_2,$$
   ここで、θ は $\mathbf{z}_1$ と $\mathbf{z}_2$ のなす角である。$\|\tilde{\mathbf{z}}(\gamma)\| = r$ がすべての γ に対して成り立つことを、$\|\mathbf{z}_1\| = \|\mathbf{z}_2\| = r$ のもとで示せ。また、θ は $d = 512$ のときおよそどのような値をとるか。それはなぜか。
3. [code] **コインに対するチェルノフ上界：**●$\hat{p}$ を、n 回の独立なコイン投げ（表が出る確率がバイアス p のコイン）における表の割合とし、$a > p$ とする。
   1. チェルノフ法 Equation 36.112 を $n\hat{p}$（そのモーメント母関数は $(1 - p + pe^{\lambda})^n$ である）に適用せよ。λ について最適化し、$P(\hat{p} \ge a) \le e^{-n\,D_{\mathrm{KL}}(a\|p)}$ を示せ。ただし $D_{\mathrm{KL}}(a\|p) = a\log\frac{a}{p} + (1-a)\log\frac{1-a}{1-p}$ とする。
   2. $D_{\mathrm{KL}}(a\|p) \ge 2(a-p)^2$ を、a に関する二階導関数を調べることで示し、(a) の上界がヘフディングの上界の片側より決して弱くならないことを確かめよ。さらに $p = \tfrac{1}{2}$、$a = 0.6$ に対して両方の指数を評価せよ。
   3. 両側上界 $2e^{-n\,D_{\mathrm{KL}}(0.6\|0.5)}$ を 36.9.1.5 節の表に新たな列として $n \in \{10, 100, 1000, 5000\}$ について追加せよ。厳密な裾確率のこの上界に対する比は、依然として n とともに減少する。この比は n に対してどのようにスケールするか。また、そのスケーリングはチェルノフ法が失う因子について何を物語っているか。
4. **カイ二乗分布の裾：**●Equation 36.121 の証明は、Equation 36.122 による $x^2 - 1$（$x \sim \mathcal{N}(0,1)$）の上界に基づいているが、本文の証明では導出なしにこれを用いた。
   1. $\mathbb{E}[e^{\lambda(x^2-1)}] = e^{-\lambda}/\sqrt{1-2\lambda}$ が $\lambda < \tfrac{1}{2}$ で成り立つことを、ガウス積分において平方完成することで示せ。
   2. $\log \mathbb{E}[e^{\lambda(x^2-1)}]$ を 2λ のべき級数に展開し、Equation 36.122 を証明せよ。この級数は 2 次の項から始まる。
   3. Equation 36.121 の証明では $\lambda = \varepsilon/4$ とおいたが、これが許容されるのは $\varepsilon \le 1$ のときに限られる。$\varepsilon > 1$ に対しては $\lambda = \tfrac{1}{4}$ とおき、$P(\|\mathbf{x}\|^2 \ge (1+\varepsilon)d)$ を上から評価せよ。得られる結果は $e^{-c\varepsilon^2}$ と $e^{-c\varepsilon}$ のどちらの形で ε に関して減衰するか。また、それは Equation 36.120 のどちらの領域に属するか。

5. [code] **深さにわたる活性化値のノルム：●**バイアスを持たない ReLU ネットワーク（幅 d、深さ K）$\mathbf{h}_k = \mathrm{ReLU}(\mathbf{W}_k \mathbf{h}_{k-1})$ を考え、その重みは $\mathcal{N}(0, 2/d)$ からの独立なサンプルとする（He 初期化、4.4 節）。$\rho_k = \|\mathbf{h}_k\|^2 / \|\mathbf{h}_{k-1}\|^2$ とおく。このようなノルムが深さと幅によってどう変動するかは、Hanin and Nica (2020) によって解析された。
   1. $\mathbf{h}_{k-1} \neq \mathbf{0}$ が与えられたとき、比 ρ_k は $\frac{2}{d}\sum_{j=1}^{d} \mathrm{ReLU}(g_j)^2$（ただし $g_j \sim \mathcal{N}(0,1)$ は独立）と同じ分布に従うことを、$\mathbf{h}_{k-1}$ の方向によらず示せ。さらに $\mathbb{E}[\rho_k] = 1$ および $\mathrm{Var}(\rho_k) = 5/d$ を、$\mathbb{E}[g^2] = 1$ と $\mathbb{E}[g^4] = 3$ を用いて導け。
   2. $\log(\|\mathbf{h}_K\|^2 / \|\mathbf{h}_0\|^2)$ を K 個の独立な項の和として表せ（各層で $\rho_k = 0$ となる確率は 2^{-d} であるが、これは無視せよ）。$\log \rho$ を 1 の周りで 2 次まで展開し、この和の平均と分散を K/d を用いて予測せよ。
   3. 1,000 個のネットワークを各幅 $d \in \{32, 128\}$ と各深さ $K \in \{1, 10, 50\}$ について、それぞれ新しい標準ガウス入力を与えてシミュレーションせよ。$\log(\|\mathbf{h}_K\|^2 / \|\mathbf{h}_0\|^2)$ の平均と分散を (b) と比較し、さらに比 $\|\mathbf{h}_K\|^2 / \|\mathbf{h}_0\|^2$ 自体の平均と中央値を報告せよ。4.4 節の分散計算によれば、あらゆる深さで $\mathbb{E}[\|\mathbf{h}_K\|^2 / \|\mathbf{h}_0\|^2] = 1$ となる。$K = 50$、$d = 32$ での実行を数回繰り返せ。1,000 個のネットワークにわたる標本平均が 1 の周りでこれほど大きくばらつくのはなぜか。また、そこではなぜ期待値が典型的なネットワークを記述しないのか。
6. [code] **距離を保存するランダム射影：●**$\mathbf{A} \in \mathbb{R}^{k \times d}$ を、独立な $\mathcal{N}(0, 1/k)$ の成分を持つ行列とする。
   1. 固定された $\mathbf{v} \neq \mathbf{0}$ に対し、比 $\|\mathbf{A}\mathbf{v}\|^2 / \|\mathbf{v}\|^2$ が $\|\mathbf{g}\|^2/k$（$\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_k)$）と同じ分布に従い、したがって Equation 36.121 より $P(|\|\mathbf{A}\mathbf{v}\|/\|\mathbf{v}\| - 1| \geq \varepsilon) \leq 2e^{-k\varepsilon^2/8}$ が成り立つことを示せ。ヒント：$\mathbf{v}$ が第 1 座標軸に沿うように回転させるとよい。
   2. n 個の点に対し、(a) を $n(n-1)/2$ 個の差ベクトルに適用してユニオンバウンドをとれ。すべてのペアワイズ距離が係数 $1 \pm \varepsilon$ の範囲内に少なくとも確率 $1 - \delta$ で保存されるための条件が $k \geq (8/\varepsilon^2)\log(n^2/\delta)$（$d$ に依存しない上界）となることを示せ。これがジョンソン・リンデンシュトラウスの補題である。$n = 1{,}000$、$\varepsilon = 0.3$、$\delta = 0.05$ に対してこの値を評価せよ。
   3. 1,000 個の標準ガウス点（$d = 2{,}000$）上でこれを検証せよ。$k \in \{50, 200, 800\}$ 次元へ射影し、全ペアにわたる最大の歪み $|\|\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\| / \|\mathbf{x}_i - \mathbf{x}_j\| - 1|$ を、典型的な揺らぎ $1/\sqrt{2k}$ と併せて報告せよ。ペアワイズ距離はグラム行列から $\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\,\mathbf{x}_i^\top \mathbf{x}_j$ により計算し、$1{,}000 \times 1{,}000 \times 2{,}000$ 配列のブロードキャストは使わないこと。得られた 2 つの数値の比をペアの個数から説明せよ。すでに $k = 200$ の時点で最大の歪みは $\varepsilon = 0.3$ を下回る一方、(b) ではるかに大きな k が要求された。上界のどの部分が緩いのか、定数 8 か、それとも $\log n$ か。
7. **ベルンシュタインの不等式：●**X を中心化された確率変数とし、$|X| \leq M$ および $\mathrm{Var}(X) = \sigma^2$ を満たすとする。
   1. $\mathbb{E}[X^k] \leq \sigma^2 M^{k-2}$ が $k \geq 2$ で成り立つことを示し、そこから $\mathbb{E}[e^{\lambda X}] \leq 1 + \sigma^2(e^{\lambda M} - 1 - \lambda M)/M^2$ が $\lambda > 0$ で成り立つことを導け。
   2. $e^u - 1 - u \leq u^2/(2(1 - u/3))$ が $0 \leq u < 3$ で成り立つことを、$u^k/k!$ と $(u^2/2)(u/3)^{k-2}$ の比較、すなわち $k! \geq 2 \cdot 3^{k-2}$ を用いることで示せ。さらに $1 + y \leq e^y$ と合わせて、$\mathbb{E}[e^{\lambda X}] \leq \exp(\lambda^2\sigma^2/(2(1 - \lambda M/3)))$ が $0 < \lambda < 3/M$ で成り立つことを導け。
   3. このような確率変数 n 個の平均に、$\lambda = t/c$（ただし $c = \sigma^2 + Mt/3$）としてチェルノフ法を適用せよ。$\lambda < 3/M$ および $1 - \lambda M/3 = \sigma^2/c$ を確認し、指数部がちょうど $-nt^2/(2c)$ になることを導いて Equation 36.117 を証明せよ。
   4. (b) により X が Equation 36.119 の意味で $\nu^2 = 2\sigma^2$、$\alpha = 2M/3$ の準指数になることを示せ。
   5. $n = 10{,}000$、$\delta = 0.05$ のとき、最悪ケースの分散 $\sigma^2 = \frac{1}{4}$（0/1 の結果における最

大分散）ではベルンシュタインの上界がヘフディングの上界より弱くなることを示せ。分散がどの値を下回るとベルンシュタインの上界が勝るか。また、それはどの範囲の誤り率に対応するか。

36.10 一様収束と汎化

前節では、データから計算した平均にエラーバーを付ける方法を示した。ホールドアウトされた10,000 枚の画像で評価した分類器は、95% の信頼度で、テスト誤差から約 1.4 パーセントポイント以内に真の誤差を持つ。ただし、この保証は見落としやすい条件に依拠している。分類器はテスト画像を抽出する**前**に固定されていなければならないのである。実際には、そのような手順をとることはめったにない。多数の設定を訓練し、検証スコアを調べ、最良のものを残す。報告されるスコアは競争の勝者に属するものであり、その勝者は部分的には標本上で最も運が良かった設定にすぎない。

第 3.5 章では、この問題とともに Vapnik と Chervonenkis による解決策を導入した (Vapnik and Chervonenkis 1971)。すなわち、すべての候補関数に対して同時に成り立つ保証を要求することで、学習者がどの関数を選んでも自動的にカバーされるようにするのである。候補のリストが有限であれば、上界の幅は候補数の対数に応じて広がるだけで済む。ここではその有限クラスの結果を簡単に振り返り、選び取れる運がどれほど大きいかを示す具体的な実験と組み合わせよう。1,000 枚の公平なコインをそれぞれ 100 回投げ、表が最も多く出たコインを残すのである。そのコインは何回の表を示すだろうか。

しかし本節の核心は、実際に訓練する無限の関数クラスにある。こうしたクラスに対しては、純粋なコイン投げで決めたラベルにどれほどよく適合できるかによってクラスの豊かさを測る、後の時代のアイデアである**ラデマッハ複雑度**（Rademacher complexity）を用いる。この指標は、線形モデルの容量がパラメータ数ではなく重みノルムに依存する理由を説明するとともに、一様収束（uniform convergence）が限界に達する場所も明らかにする。それは本編で扱った深層ネットワークのように、ランダムなラベルに適合できるほど柔軟なモデルにおいてである。最後のパートでは、メカニズムのあらゆる要素を閉じた形で計算できるほど単純なモデルを用いて、4.5.2 節の二重降下（double descent）の実験を再訪する。

```
rng = np.random.default_rng()
```

36.10.1 一様収束

固定されたモデルをテストセット上で評価することと、学習者を訓練することの違いは、関数の選択と標本の抽出という 2 つの事象の順序だけにある。ここでは、その順序を逆にすることでスコアがどれほど歪むか、そして選択が標本に依存するようになった途端にどれほど多くのデータが必要になるかを定量化する。

36.10.1.1 多数のコインから選ぶ最良のコイン

まず、本節を通して用いる記法で単一関数に対する保証を述べよう。標本 $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ は未知の分布からの n 個の独立な抽出からなり、損失 ℓ は $[0,1]$ に値をとって各予測を評価する。たとえば 0-1 損失は、誤りに対して 1、それ以外に対して 0 をとる。

関数 f の**経験リスク**（empirical risk）とは標本上の平均損失 $\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i)$ であり、**リスク**（risk）とは新たな抽出に対する期待損失 $R(f) = \mathbb{E}[\ell(f(\mathbf{x}), y)]$ である（0-1 損失の場合、これらは本編における経験誤差 $\epsilon_S(f)$ と母集団誤差 $\epsilon(f)$ にあたる）。f が S の抽出前に固定されていれば、$\hat{R}(f)$ は n 個の $[0,1]$ 内の独立な項（平均 $R(f)$）の平均となる。したがっ

て、ヘフディングの不等式（Hoeffding's inequality、36.9 節）により $P(|\hat{R}(f) - R(f)| \geq t)$ は $2e^{-2nt^2}$ で上から抑えられる。この失敗確率を $\delta = 0.05$ とおいて t について解くと、本編でヘフディング半径（Hoeffding radius）と呼んだ半幅 $\sqrt{\log(2/\delta)/(2n)}$ が得られる。$n = 10{,}000$ のとき、この値は 0.0136、すなわち冒頭で述べた 1.4 パーセントポイントとなる。

学習者は事象の順序を逆にする。関数のクラス $\mathcal{F}$ を探索し、予測器 $\hat{f}$ として観測標本上で経験リスク $\hat{R}$ が最小になるものを返す。各 $\hat{R}(f)$ は真のリスク $R(f)$ の周りで変動するため、学習者はたまたま都合のよいランダムな変動を示した関数を自然に選び取ってしまう。$\hat{R}(\hat{f})$ は多数の変動する量の最小値であるため、固定された各 f に対して個別に成り立つ不等式ではもはやカバーできない。

コイン投げの実験を行えば、この選択効果がどれほど大きくなり得るかがよくわかる。$k = 1{,}000$ 枚の公平なコインをそれぞれ $m = 100$ 回投げるとしよう。事前に指定した 1 枚のコインについては、$\delta = 0.05$ でのヘフディングの不等式により表の割合のずれとして $\sqrt{\log 40/200} = 0.136$ が許容されるため、そのコインは少なくとも 95% の確率で表が 37〜63 回の範囲に収まる（厳密な確率は 99.3% である）。しかし、1,000 枚のコインのうち最良のものにはそのような保証はない。この実験を 2,000 回繰り返し、勝者となったコインがどれほど大きく外れ、1 枚のコインに対する帯域をどれほどの頻度で逸脱するかを測定してみよう。

```
k, m, repeats = 1000, 100, 2000          # coins, flips per coin, experiments
best = rng.binomial(m, 0.5, size=(repeats, k)).max(axis=1) / m
one = np.sqrt(np.log(2 / 0.05) / (2 * m))       # Hoeffding, one fixed coin
many = np.sqrt(np.log(2 * k / 0.05) / (2 * m))  # union bound over k coins
print(f'best of {k} coins: median {np.median(best):.2f}, 90% of experiments '
      f'in [{np.percentile(best, 5):.2f}, {np.percentile(best, 95):.2f}]')
print(f'95% half-width: one coin {one:.3f}, all {k} coins {many:.3f}')
print(f'experiments in which the best coin leaves the one-coin band: '
      f'{np.mean(best > 0.5 + one):.0%}')
```

```
best of 1000 coins: median 0.66, 90% of experiments in [0.64, 0.69]
95% half-width: one coin 0.136, all 1000 coins 0.230
experiments in which the best coin leaves the one-coin band: 96%
```

典型的な実験では、最良のコインは 66 回の表を示し、10 回中 9 回の実験で 64〜69 回の範囲に入る。実行の 96%〜97% で単一コインの帯域を逸脱するため、事前に指定したどのコインに対しても成り立つ保証が、事後に選んだコインに対してはほぼ毎回破綻することになる。どのコインにも偏りはない。勝者が 16 パーセントポイントも外れるのは、ひとえに 1,000 個のランダムな変動を調べて最大のものを選んだからである。

Figure 36.46 を見れば、その理由は一目瞭然である。個々のコインが帯域の上端を超える確率は 0.0033 であり、単独ではまれな事象にすぎない。しかし 1,000 枚のコインの中では、約 3.3 枚が上端を超えると期待される。1 枚のコインではほとんど到達しない裾の領域も、1,000 枚のコインを投げれば十分に埋まるのである。

検証セット上でのモデル選択は、ラベルを貼り替えただけのまったく同じ実験である。各設定が 1 枚のコインであり、各検証例が 1 回のコイン投げにあたる。独立な公平コインとは異なり、検証される各設定は同じデータ例を共有し、真の誤差もそれぞれ異なるが、次に導出する上界は候補間の独立性も平均が等しいことも必要としない。

では、選択された関数をもカバーする保証はどうすれば得られるだろうか。いま、クラス全体にわたる最大のずれを考える。

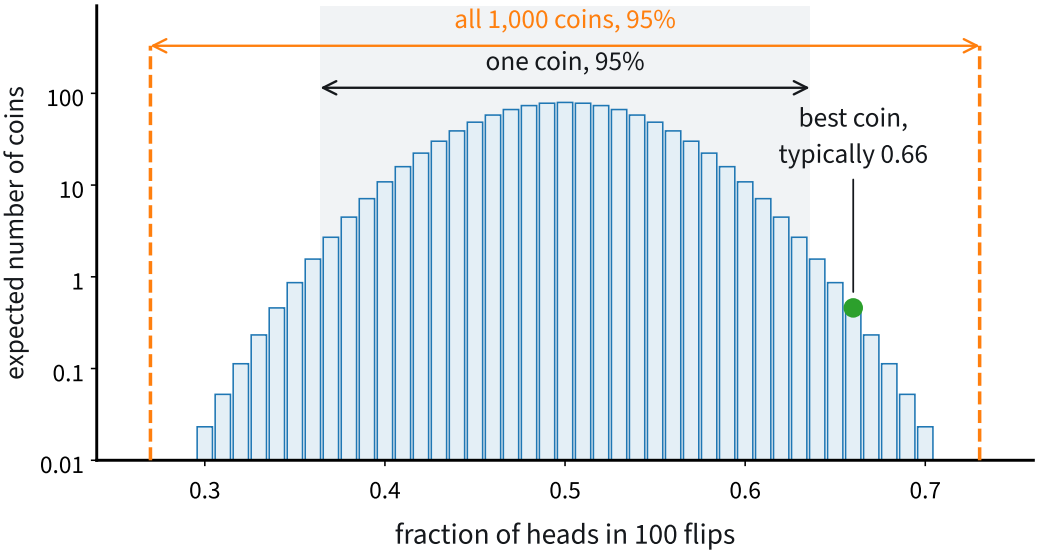


Figure 36.46: それぞれ 100 回投げた 1,000 枚の公平なコインにおける表の割合の期待ヒストグラム（対数スケール表示）。単一コインに対する 95% ヘフディング帯域はコインの 99% をカバーするが、その上端より上に約 3.3 枚のコインが入ると期待され、最良のコインは典型的に 0.66 付近に着地する。1,000 枚すべてのコインに対するより広いユニオンバウンド帯域は、最良のコインもカバーする。

$$\sup_{f \in \mathcal{F}} \left| \hat{R}(f) - R(f) \right|,$$

この上限が高々 t であると仮定する。すると、すべての関数の経験リスクはその真のリスクから t 以内に収まり、とりわけ学習者が選んだ関数についても同じことが成り立つ。この上限に対する上界は**一様収束**上界と呼ばれる (Vapnik and Chervonenkis 1971)。データが到着する前にすべての候補に対して保証が成り立っていれば、どの候補を選んでもすでにカバーされているため、データを見た後の選択が安全になるのである（Figure 36.47）。

36.10.1.2 有限クラスとユニオンバウンド

最も単純な設定は、事前に指定されたハイパーパラメータ設定のグリッドのように、あらかじめ固定された有限個の候補リストである。この場合の結果は本編の第 3.5 章ですでに述べており、その証明はわずか 2 行で終わる。ヘフディングの不等式により、固定された単一の f が t 以上ずれる確率は $2e^{-2nt^2}$ で抑えられる。すべての $|\mathcal{F}|$ 個の候補にわたってユニオンバウンド（36.9 節）を適用すると、それらの**いずれか**が t 以上ずれる確率は $2|\mathcal{F}|e^{-2nt^2}$ で抑えられる。この全体の失敗確率を δ とおいて t について解けば、最初の一様収束定理が得られる。すなわち、$[0,1]$ に値をとる損失とサイズ n の独立同分布（i.i.d.）標本に対し、少なくとも $1-\delta$ の確率で次が成り立つ。

$$\left| \hat{R}(f) - R(f) \right| \le \sqrt{\frac{\log(2|\mathcal{F}|/\delta)}{2n}} \qquad \text{simultaneously for every } f \in \mathcal{F}. \tag{36.124}$$

固定された単一の関数を評価する場合と比べると、$|\mathcal{F}|$ 個の候補から選択することは平方根の中

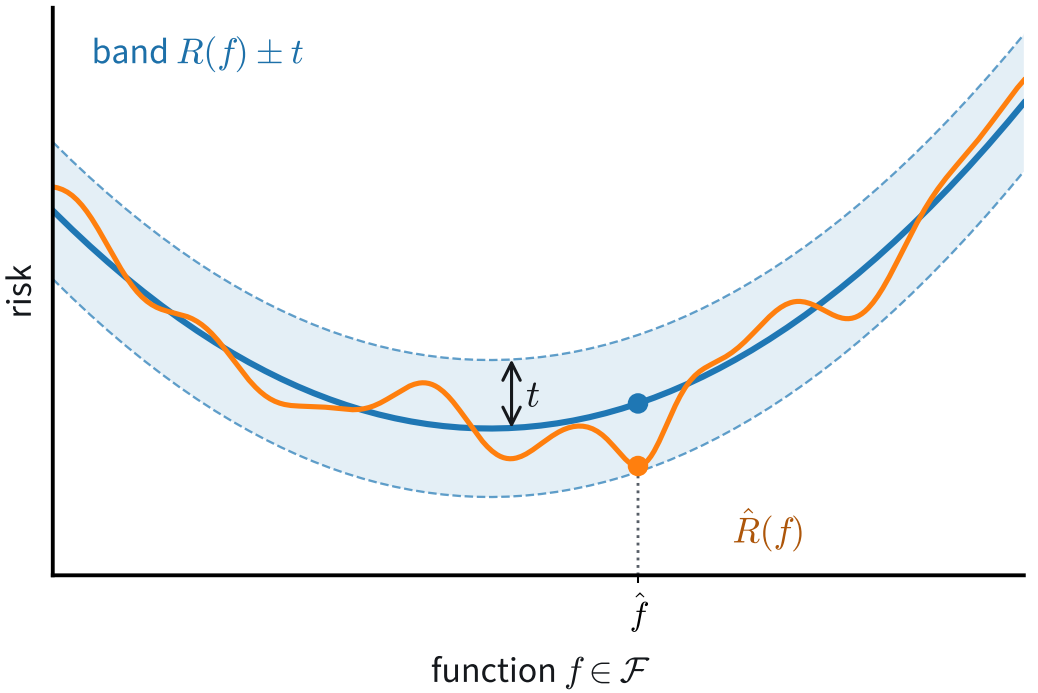


Figure 36.47: 一様収束。経験リスク $\hat{R}(f)$（オレンジ）は、クラス内のすべての関数に対して同時に、ずれ t 以内の範囲で真のリスク $R(f)$（青）の周囲に収まる。学習者は経験リスクの最小点 $\hat{f}$ を選ぶが、これは真のリスクより下側に落ち込んだ谷に位置している。帯域はその関数もカバーするため、$R(\hat{f}) \leq \hat{R}(\hat{f}) + t$ が成り立つ。

に $\log|\mathcal{F}|$ を加えるだけであり、それ以外のペナルティはない。

先ほどのコインの実験で $|\mathcal{F}| = 1{,}000$、$n = 100$ とおくと半幅は $\sqrt{\log(2{,}000/0.05)/200} = 0.230$ となり、少なくとも 95% の確率で、勝者のコインを含むすべてのコインが表 27〜73 回の範囲に収まる。確かにこの上界は保守的である。いずれかのコインが表 27〜73 回の外に出る真の確率は約 0.17% であり、許容予算の 5% を大きく下回る。この 30 倍近い緩さのほぼすべては単一コインに対するヘフディングの不等式に由来している。1 枚のコインがこの範囲を外れる真の確率が 1.7×10^{-6} であるのに対し、ヘフディングの不等式は 5×10^{-5} まで許容するためであり、まれな独立事象に対するユニオンバウンド自体はほぼタイトである。保守的であれ何であれ、単一コインの帯域が妥当でなかったのに対し、この帯域は妥当な保証を与えている。

10,000 例からなる検証セットについては、本編で半幅が単一関数で 0.014、100 万個の関数で 0.030 となることを記録した。Equation 36.124 を標本サイズ n について解くと、探索のコストがいっそう明確になる。

$$n = \frac{\log(2/\delta) + \log|\mathcal{F}|}{2t^2}.$$

$\log|\mathcal{F}|$ が 1 増えるごとに、さらに $1/(2t^2)$ 個のデータ例が必要になる。許容誤差 $t = 0.01$ のもとでは、候補モデルの数を 2 倍にするコストは追加の検証例 $\log 2/(2 \cdot 10^{-4}) \approx 3{,}500$ 個に相当する。

この対数依存性は、ヘフディングの指数的な裾がもたらす直接の恩恵である。仮にチェビシェフの不等式（Chebyshev's inequality）を代わりに用いると（$[0,1]$ に値をとる損失の分散は高々 $\tfrac{1}{4}$ であることに注意する）、同じユニオンバウンドから得られるのは $t = \sqrt{|\mathcal{F}|/(4n\delta)}$ となる。2,000 個の設定を 10,000 例上で $\delta = 0.05$ として評価する場合、チェビシェフの不等式は $t = 1$ となって損失の値域全体に広がり何も教えてくれないのに対し、ヘフディングの不等式は $t = 0.024$ を与える。

証明は最初のステップで 1 つの決定的な仮定に依拠している。候補クラス $\mathcal{F}$ が標本の抽出前に固定されていなければならないという点である。公開ベンチマークのリーダーボードでは、過去の提出のテストスコアを調べた後に新しい提出が選ばれるたび、この仮定が破られる (Dwork et al. 2015)。ひとたび候補がホールドアウトセットに適応するようになると、誤差保証はオーダー $\sqrt{\log k/n}$（固定された k 個の候補の場合）から、オーダー $\sqrt{k/n}$（ランダムな推測器 k 個から構成した適応的多数決の場合）へと劣化する（演習問題 5）。

36.10.2 ラデマッハ複雑度

実際に訓練する関数クラスは無限である。すべての重みベクトル $\mathbf{w} \in \mathbb{R}^d$ がそれぞれ異なる線形分類器を定めるため、無限個の事象にわたるユニオンバウンドは空虚なものになる。古典的な学習理論では、クラスを有限個の代表元で近似するか（被覆数、covering number）、あるいは n 個の点上で実現できる相異なるラベル付けの個数を数えるか（VC 次元、第 3.5 章で概説）によってこの問題を回避する。ここではより直接的なアプローチをとり (Bartlett and Mendelson 2002)、実務家なら誰でも検証できる操作的な問いを立てる。すなわち、そのクラスは純粋なノイズにどれほどよく適合できるかという問いである。

本節の中心となる 2 つの結果はこのパートに収められている。一様収束定理 Equation 36.126 と、ノルム有界な線形モデルに対する複雑度上界 $Br/\sqrt{n}$（Equation 36.128）である。唯一技巧的なステップは、前者の証明における対称化（symmetrization）の議論である。

36.10.2.1 ランダム符号との相関

ある関数クラスが信号をまったく含まないラベルに適合できるのであれば、実際のラベルに含まれるノイズにも同じくらい容易に適合できてしまい、その経験リスクから真のリスクについてわかることはほとんどなくなる。クラスがノイズを暗記する容量を調べるため、架空のランダムなラベルを与えてみよう。$\varepsilon_1, \dots, \varepsilon_n$ を独立な公平ランダム符号とし、それぞれが値 $+1$ または -1 を確率 $\tfrac{1}{2}$ でとるものとする。これらは**ラデマッハ変数**（Rademacher variables）と呼ばれる。与えられた符号ベクトル ε に対し、平均 $\frac{1}{n}\sum_i \varepsilon_i f(\mathbf{x}_i)$ は f が標本上でそれらの符号とどれほど相関するかを測り、$f \in \mathcal{F}$ にわたる上限はそのクラスが達成できる最良の相関を測る。

定義。実数値関数のクラス $\mathcal{F}$ の標本点 $S = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ 上における**経験ラデマッハ複雑度**（empirical Rademacher complexity）とは、そのクラスがランダム符号との間で達成する最良の相関の期待値であり、次式で与えられる。

$$\widehat{\mathfrak{R}}_S(\mathcal{F}) = \mathbb{E}_{\varepsilon}\left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i \, f(\mathbf{x}_i) \right], \tag{36.125}$$

また、その**ラデマッハ複雑度** $\mathfrak{R}_n(\mathcal{F}) = \mathbb{E}_S[\widehat{\mathfrak{R}}_S(\mathcal{F})]$ とは、サイズ n の無作為標本にわたるこの量の平均である。

任意の符号パターンに一致できるクラスは高いスコアを示し、自由度のほとんどないクラスはゼロに近いスコアとなる。厳密な上限は関数の値域に依存する。すべての $f \in \mathcal{F}$ が $[-1,1]$ に値をとるなら、どの相関も高々 1 であり、$[-1,1]$ への**すべて**の関数からなるクラスは、相異なる点上で 1 を達成する（任意の符号パターンに対して $f(\mathbf{x}_i) = \varepsilon_i$ とおけばよい）。$[0,1]$ に値をとるクラスの場合、とり得る最大の複雑度は $\tfrac{1}{2}$ である。最良の選択は $f(\mathbf{x}_i) = 1$ を $\varepsilon_i = +1$ の箇所でとり、それ以外では 0 とおくことであり、プラス符号の割合がスコアとなってその平均は $\tfrac{1}{2}$ になる。後ほど調べる線形関数のような非有界なクラスにはこのような天井はない。標本上のあらゆる符号パターンを補間できるなら、その複雑度は少なくとも 1 になり、さらに大きくなることもある。

数値の感覚をつかむため、最も単純な非自明クラスとして 2 つの定数関数 $\{+1, -1\}$ を考えよう。符号ベクトルとの最良の相関は $\max(\bar{\varepsilon}, -\bar{\varepsilon}) = |\bar{\varepsilon}|$ となる。ただし $\bar{\varepsilon}$ は符号の標本平均である。$n = 2$ のとき、等確率で生じる 4 つのパターン $(+,+), (+,-), (-,+), (-,-)$ に対して $|\bar{\varepsilon}| = 1, 0, 0, 1$ となるため、$\widehat{\mathfrak{R}}_S = \tfrac{1}{2}$ である。$n = 4$ では、符号の和は ± 4 が確率 $\tfrac{2}{16}$、± 2 が確率 $\tfrac{8}{16}$、それ以外では 0 となるので、$\widehat{\mathfrak{R}}_S = \tfrac{3}{8}$ が得られる。大きな n に対しては、$\bar{\varepsilon}$ は平均 0、分散 $1/n$ の正規分布に近似的に従うため、$\widehat{\mathfrak{R}}_S \approx \sqrt{2/(\pi n)}$ となる。$n = 100$ では厳密値が 0.0796 であるのに対し、正規近似からは 0.0798 が得られる。わずか 2 つの関数しかもたないクラスであっても、ノイズとレート $1/\sqrt{n}$ で相関する。n 個のランダム符号の平均がそのスケールで変動するからにほかならない。対照的に、単一要素クラス（singleton class）$\{f\}$ の複雑度は 0 である。選ぶ余地がないため、$\mathbb{E}[\frac{1}{n}\sum_i \varepsilon_i f(\mathbf{x}_i)] = 0$ となるからである。

線形クラスの場合、最良の相関には明快な幾何学的解釈がある。Figure 36.48 に先取りして示すように、それは入力ベクトルの符号付き和の長さを n で割ったものにほかならない。このことは後ほど 36.10.2.3 節で証明する。

$[-1,1]$ に値をとる有限クラスは**マサールの補題**（Massart's lemma）$\widehat{\mathfrak{R}}_S(\mathcal{F}) \le \sqrt{2 \log |\mathcal{F}| / n}$ を満たすため（演習問題 2）、その複雑度は関数の個数に対して対数的にしか増大しない。より一般に、ラデマッハ複雑度はパラメータ数や要素数にかかわらず、実際の標本点上でクラスがどれほど自由に変化できるかを捉える。

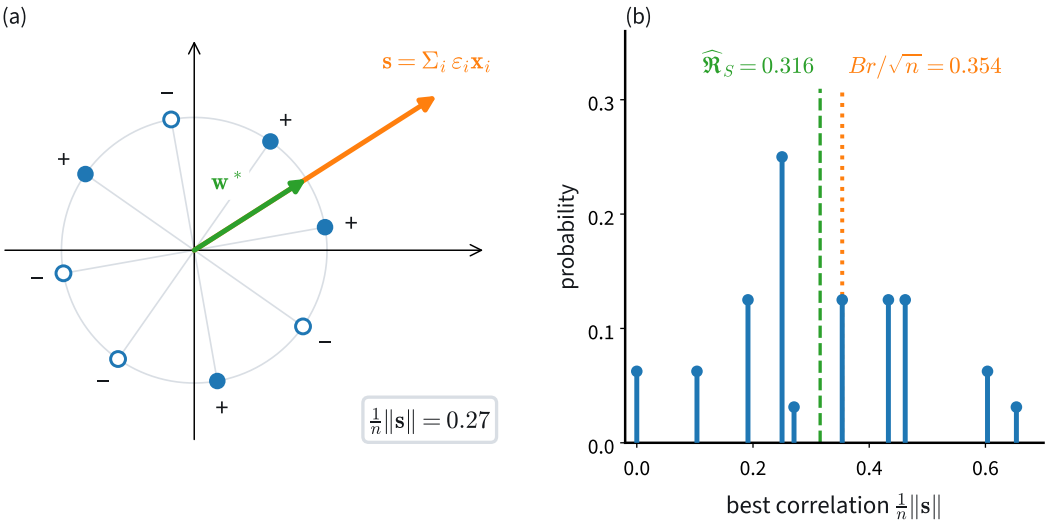


Figure 36.48: 平面上の 8 個の単位ベクトルと単位ノルム線形関数のクラスに対する線形クラスのラデマッハ複雑度。(a) 単一の符号パターンにおいて、塗りつぶした点は $+1$、白抜きの点は -1 を表す。最良の相関は、単位重みベクトル $\mathbf{w}^*$ を符号付き和 $\mathbf{s} = \sum_i \varepsilon_i \mathbf{x}_i$ と同じ向きに揃えたときに達成され、その値は $\|\mathbf{s}\|/n = 0.27$ に等しい。(b) 全 256 通りの符号パターンにわたる最良の相関の厳密な分布。その平均である経験ラデマッハ複雑度は 0.316 であり、本文で証明する上界 $Br/\sqrt{n} = 0.354$ と比較できる。

36.10.2.2 一様収束定理

これで、ラデマッハ複雑度がクラス全体にわたる経験リスクと真のリスクの最大ギャップを制御することを示す準備が整った。予測と損失は異なる関数クラスに属するため、まず記法上の小さな区別をしておく。当てはめるモデルを含むのは**予測器クラス**（predictor class）$\mathcal{F}$ である一方、平均を比較する対象となる量は損失である。そこで、次のように**損失クラス**（loss class）を定義する。

$$\mathcal{G} = \{(\mathbf{x}, y) \mapsto \ell(f(\mathbf{x}), y) : f \in \mathcal{F}\},$$

このクラスに属する関数は $[0,1]$ に値をとる。ラベル付きデータ例を $z = (\mathbf{x}, y)$ と書き、$R(g) = \mathbb{E}[g(z)]$ および $\hat{R}_S(g) = \frac{1}{n}\sum_i g(z_i)$ を $g \in \mathcal{G}$ に対して定める。予測器のリスクとは、対応する損失関数のリスクにほかならない。

命題（ラデマッハ複雑度による一様収束）（Bartlett and Mendelson 2002）。$S = (z_1, \dots, z_n)$ **を独立同分布標本とし、** $\mathcal{G}$ **を** $[0,1]$ **に値をとる関数のクラスとする。このとき、少なくとも** $1 - \delta$ **の確率で次が成り立つ。**

$$R(g) \le \hat{R}_S(g) + 2\,\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{\log(1/\delta)}{2n}} \qquad \text{simultaneously for every } g \in \mathcal{G}. \tag{36.126}$$

この上界が有限クラスの上界とまったく同じ 3 項構造をもっていることに注目しよう。すなわち、訓練標本上の経験リスク、クラスの豊かさを測る複雑度項、そして固定された単一の関数に対し

ても現れる信頼項である。ここでは、豊かさの項がランダム符号に適合するクラスの能力の 2 倍になっている。

証明。まず最悪ケースのギャップの期待値の上界 $\mathbb{E}_S[\sup_g (R(g) - \hat{R}_S(g))] \le 2\mathfrak{R}_n(\mathcal{G})$ を 3 つのステップで示し、続いて集中不等式から高確率での主張を導く。

I. ゴースト標本（ghost sample）。証明内部の純粋な解析上の道具として、独立なコピー $S' = (z'_1, \dots, z'_n)$ を標本 S に対して導入する。$R(g) = \mathbb{E}_{S'}[\hat{R}_{S'}(g)]$ であり、また期待値の上限はつねに上限の期待値以下であるため、次を得る。

$$\mathbb{E}_S\Big[\sup_{g\in\mathcal{G}} (R(g) - \hat{R}_S(g))\Big] \le \mathbb{E}_{S,S'}\Big[\sup_{g\in\mathcal{G}} \frac{1}{n}\sum_{i=1}^{n} \big(g(z'_i) - g(z_i)\big)\Big]. \tag{36.127}$$

これにより、未知の母集団リスクが新たな標本上の経験リスクに置き換えられた。

II. 符号の入れ替え。両方の点は同じ分布からの独立な抽出であるため、z_i と z'_i を入れ替えても (S, S') の同時分布はまったく変わらない。しかしその入れ替えは、i 番目の差 $g(z'_i) - g(z_i)$ の符号をすべての $g \in \mathcal{G}$ に対して同時に反転させる。したがって、符号 $\varepsilon_1, \dots, \varepsilon_n \in \{-1, +1\}$ を任意に固定したとき、i 番目の差に ε_i を掛けても Equation 36.127 の右辺は変化しない。ランダム符号にわたって平均をとっても同様に不変である。

$$\mathbb{E}_{S,S'}\Big[\sup_g \frac{1}{n}\sum_i \big(g(z'_i) - g(z_i)\big)\Big] = \mathbb{E}_{S,S',\varepsilon}\Big[\sup_g \frac{1}{n}\sum_i \varepsilon_i \big(g(z'_i) - g(z_i)\big)\Big].$$

III. 分割。和の上限は上限の和以下であるため、右辺は次式で上から抑えられる。

$$\mathbb{E}_{S',\varepsilon}\Big[\sup_g \frac{1}{n}\sum_i \varepsilon_i\, g(z'_i)\Big] + \mathbb{E}_{S,\varepsilon}\Big[\sup_g \frac{1}{n}\sum_i (-\varepsilon_i)\, g(z_i)\Big] = 2\,\mathfrak{R}_n(\mathcal{G}),$$

ここで最後の等号は、$-\varepsilon$ が ε と同じ分布に従い、両方の期待値がともに $\mathfrak{R}_n(\mathcal{G})$ に等しくなることによる。係数 2 はこの分割から生じる。

期待値から高確率の上界へ移るため、$\Phi(S) = \sup_g (R(g) - \hat{R}_S(g))$ と定義する。単一の標本点 z_i を置き換えても、各経験リスク $\hat{R}_S(g)$ の変化は高々 $1/n$ であり（g は $[0,1]$ に値をとるため）、したがって上限 $\Phi(S)$ の変化も高々 $1/n$ である。ここで、ヘフディングの不等式を和から「単一の入力では大きく変化しない任意の関数」へと一般化する**マクダイアミッドの不等式**（McDiarmid's inequality）(McDiarmid 1989) を適用すると、$P(\Phi - \mathbb{E}\Phi \ge t) \le e^{-2nt^2}$ が得られる（演習問題 1）。右辺を δ とおき、$\mathbb{E}\Phi \le 2\mathfrak{R}_n(\mathcal{G})$ と組み合わせれば Equation 36.126 が得られる。$\blacksquare$

この**対称化**の議論の背後にある直観はきわめて明快である。ゴースト標本は母集団の身代わりを務める。実際の標本点とゴースト標本点は交換可能であるため、どのペアを入れ替えても分布は一切変わらず、その入れ替えをランダム化してしまえばコイン投げだけが残る。つまり、訓練リスクと母集団リスクの最悪ケースのギャップは、クラスがランダム符号をどれほどよく追いかけられるかによって決まるのである。

実務上の細部が 1 つ残っている。定理は期待複雑度 $\mathfrak{R}_n(\mathcal{G})$ を用いて上界を述べているが、実際のデータセット上では手元の標本に対する経験複雑度 $\widehat{\mathfrak{R}}_S(\mathcal{G})$ しか評価できない。標本点を 1 つ変更したときの $\widehat{\mathfrak{R}}_S(\mathcal{G})$ の変化も高々 $1/n$ であるため、マクダイアミッドの不等式をもう一度適

用すれば、信頼項を広げた代償のもとで $\widehat{\mathfrak{R}}_S(\mathcal{G})$ を $\mathfrak{R}_n(\mathcal{G})$ の代わりに Equation 36.126 へ代入でき、信頼項は $3\sqrt{\log(2/\delta)/(2n)}$ となる。以下の数値検証ではこの経験形式を用いる。

極端な場合で定理を確かめておくことはつねに有益である。単一要素クラスでは $\mathfrak{R}_n = 0$ となり、Equation 36.126 はそのまま片側のヘフディングの不等式に帰着する。有限クラスでは、マサールの補題により定数は緩くなるものの Equation 36.124 と同じ形が復元される。たとえば 2,000 個の設定と 10,000 例の場合、複雑度項だけで $2\sqrt{2\log 2000/n} = 0.078$ となるのに対し、ユニオンバウンドでは半幅全体でも 0.024 である。ラデマッハ複雑度は無限クラスを苦もなく扱える引き換えとして、有限クラスでは定数倍のペナルティを払うことになる。二値分類器の場合、n 個の点上で実現可能なラベル付けの総数を数えることで、$\mathfrak{R}_n$ は $\sqrt{d_{\mathrm{VC}}\log n / n}$ の定数倍で上から抑えられる (Boucheron et al. 2005)。ここで d_{VC} は本編で導入した VC 次元、すなわちそのクラスがあらゆる可能なラベル付けを実現できる最大の点数である。

反対の極端なケースとして、$[0,1]$ に値をとる任意の損失クラスは $\mathfrak{R}_n(\mathcal{G}) \le \tfrac{1}{2}$ を満たす。等号が成り立つとき、複雑度項 $2\mathfrak{R}_n(\mathcal{G})$ は 1 に等しくなり、上界は最大 100% までのいかなるリスクも許容する空虚なものになる。$\{-1,+1\}$ に値をとる二値分類器と 0-1 損失 $g(\mathbf{x},y) = (1 - y f(\mathbf{x}))/2$ のもとでは、損失の複雑度と予測器の複雑度は厳密に 2 倍の比になる。実際、相関を展開すると $\sum_i \varepsilon_i g(z_i) = \tfrac{1}{2}\sum_i \varepsilon_i + \tfrac{1}{2}\sum_i (-\varepsilon_i y_i) f(\mathbf{x}_i)$ となる。第 1 の和は平均ゼロで f に依存せず、一方で固定されたラベル $y_i \in \{-1,+1\}$ に対して積 $-\varepsilon_i y_i$ は再び独立な公平ランダム符号となる。上限と期待値をとれば次式が得られる。

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) = \tfrac{1}{2}\,\widehat{\mathfrak{R}}_S(\mathcal{F}).$$

したがって、二値予測器のクラスが標本上のあらゆる符号パターンに適合できるなら、$\widehat{\mathfrak{R}}_S(\mathcal{F}) = 1$ となり、損失の複雑度の 2 倍が 1 に達するため、一様収束定理は誤り率が 100% 以下であること以上の保証を何も与えない。

36.10.2.3 線形クラス

汎化定理がどれほど役に立つかは、実際に関心のあるクラスに対してその複雑度項を評価できるかどうかにかかっている。重みノルムが有界な線形予測器の場合、その計算はわずか 4 行で済み、しかもその答えは驚くべきことに入力次元に依存しない。

命題（ノルム有界な線形クラスのラデマッハ複雑度）。 **$\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x}\rangle : \|\mathbf{w}\|_2 \le B\}$ とし、すべての標本入力が $\|\mathbf{x}_i\|_2 \le r$ を満たすとする。このとき、次が成り立つ。**

$$\widehat{\mathfrak{R}}_S(\mathcal{F}) = \frac{B}{n}\,\mathbb{E}_{\varepsilon}\Big\|\sum_{i=1}^{n}\varepsilon_i \mathbf{x}_i\Big\| \le \frac{B\,r}{\sqrt{n}}. \tag{36.128}$$

証明。 内側の上限は閉じた形で書ける。内積の線形性とコーシー・シュワルツの不等式により、最大値は重みベクトル $\mathbf{w}$（長さ B）が符号付き和の方向を向くときに達成される。

$$\sup_{\|\mathbf{w}\|\le B} \frac{1}{n}\sum_{i=1}^{n}\varepsilon_i \langle \mathbf{w}, \mathbf{x}_i\rangle = \sup_{\|\mathbf{w}\|\le B}\frac{1}{n}\Big\langle \mathbf{w}, \sum_i \varepsilon_i \mathbf{x}_i\Big\rangle = \frac{B}{n}\Big\|\sum_{i=1}^{n}\varepsilon_i \mathbf{x}_i\Big\|.$$

ε に関して期待値をとれば命題の等号が得られる。上界を得るには、平方根の凹性（$\mathbb{E}\|\cdot\| \le \sqrt{\mathbb{E}\|\cdot\|^2}$）を用いて二乗ノルムを展開する。

$$\mathbb{E}_{\varepsilon}\Big\|\sum_{i}\varepsilon_i \mathbf{x}_i\Big\| \le \sqrt{\sum_{i,j}\mathbb{E}[\varepsilon_i\varepsilon_j]\,\langle \mathbf{x}_i,\mathbf{x}_j\rangle} = \sqrt{\sum_{i}\|\mathbf{x}_i\|^2} \le r\sqrt{n}.$$

独立な公平符号は $\mathbb{E}[\varepsilon_i\varepsilon_j]=0$（$i\neq j$）を満たすため、交差項はすべて消える。これに B/n を掛ければ $Br/\sqrt{n}$ が得られる。■

Figure 36.48 のパネル (a) はこの証明の最初のステップを図示している。任意の符号パターンに対し、最適な重みベクトルは単に入力の符号付き和と同じ向きに揃う。

注目すべきことに、この証明で緩みが生じるのは 1 箇所だけである。恒等式 $\mathbb{E}\|\sum_i \varepsilon_i\mathbf{x}_i\|^2 = \sum_i\|\mathbf{x}_i\|^2$ は厳密に成り立つため、ギャップはすべて符号付き和の平均の長さをその二乗平均平方根の長さで置き換えることから生じる。この不等式は、符号パターン間で長さがあまり変動しないときにはつねにタイトになる。図の 8 点では、緩みは 11% である（0.316 対 0.354）。2 次元では、よく広がった多数の単位ベクトルの符号付き和は 2 次元ガウスベクトルのように振る舞い、その平均の長さは二乗平均平方根の長さの $\sqrt{\pi}/2\approx 0.89$ 倍になるため、緩みは n にかかわらず 11% 付近にとどまる。しかし 20 次元では、36.9.2.1 節の薄殻現象（thin-shell phenomenon）により符号付き和の長さは二乗平均平方根の周りに鋭く集中し、緩みは約 1% まで縮小する。

Equation 36.128 に何が含まれていないかに注目しよう。入力次元 d が一切現れないのである。次元は B と r を通じて間接的に関与するにすぎない。この上界において、線形モデルの容量は扱う特徴量が 10 個であろうと 100 万個であろうと、重みノルムと入力半径の積によって支配される。これは重み減衰（2.7 節）に対する明快な理論的動機を与える。$\|\mathbf{w}\|$ を小さく保てば、訓練されたモデルを含む関数クラスが縮小するのである。また、この次元に依存しない性質は線形モデルに限られない。出力重みの ℓ_1 ノルムが高々 B_1 であり、隠れ層の重みベクトルの ℓ_2 ノルムが高々 B_2 である 2 層 ReLU ネットワークは、幅にかかわらずラデマッハ複雑度が高々 $2B_1B_2r/\sqrt{n}$ となる（演習問題 8）。

注意深い読者は自然な疑問を抱くかもしれない。命題では半径 B をデータを見る**前**に固定する必要があるのに対し、訓練された重みベクトルのノルムは訓練の**後**になって初めてわかるという点である。この指摘はもっともだが、半径の列にわたるユニオンバウンドによって解決できる。各半径 $B_j=2^j$ に対して失敗確率 $\delta_j=\delta/(j(j+1))$（総和は δ になる）で定理を適用し、当てはめた重みベクトルを含む最小の B_j で上界を評価すればよい。$\|\mathbf{w}\|>1$ のとき、これによるコストは複雑度項で高々 2 倍にとどまり、穏やかな $\sqrt{\log\log\|\mathbf{w}\|/n}$ のペナルティが加わるだけで済む（演習問題 3）。

Equation 36.128 を線形分類器の汎化上界へと変換するには、あと 1 つ橋渡しが残っている。一様収束定理が必要とするのは**損失**クラス $\mathcal{G}$ の複雑度であるのに対し、先ほどの命題が上から抑えたのは**予測器**クラス $\mathcal{F}$ であった。Lipschitz 損失関数がこの 2 つを結び付ける。

補題（縮小補題）（Boucheron et al. 2013）。**$\phi_1,\dots,\phi_n:\mathbb{R}\to\mathbb{R}$ がそれぞれ L-Lipschitz であるとする。このとき、任意の関数クラス $\mathcal{F}$ と点 $\mathbf{x}_1,\dots,\mathbf{x}_n$ に対して次が成り立つ。**

$$\mathbb{E}_{\varepsilon}\Big[\sup_{f\in\mathcal{F}}\sum_{i=1}^{n}\varepsilon_i\,\phi_i(f(\mathbf{x}_i))\Big] \le L\,\mathbb{E}_{\varepsilon}\Big[\sup_{f\in\mathcal{F}}\sum_{i=1}^{n}\varepsilon_i\,f(\mathbf{x}_i)\Big]. \tag{36.129}$$

平たく言えば、距離を高々 L 倍にしか引き伸ばさない変換を関数クラスに合成しても、そのラデマッハ複雑度は高々 L 倍にしかならない。各 ϕ_i が異なってよいという点は、まさに教師あり学習の損失が求める性質である。データ例 i において、ϕ_i は写像 $u\mapsto \ell(u,y_i)$（ラベル y_i を固定したもの）にあたる。定数 c を ϕ_i に加えても和は期待値ゼロの $c\,\varepsilon_i$ だけずれるにすぎないため、中心化の条件も不要である。ここでは縮小補題（contraction lemma）を証明なしで認める。

また一様収束定理では、損失が $[0,1]$ に有界であることも必要になる。ヒンジ損失やロジスティック損失といった一般的な訓練損失は非有界であり、ノルム有界な線形クラス上ではおよそ $1+Br$ に達し得るため、定理を直接適用するにはその最大値で割って再スケーリングした損失を抑える必要がある。解析上より見通しがよいのは、最初から有界な損失を用いることである。すなわち、分類マージン $u = yf(\mathbf{x})$ に適用する**ランプ損失**（ramp loss）であり、次式で定義される。

$$\phi_\gamma(u) = \min\{1, \max\{0,\, 1 - u/\gamma\}\}.$$

ランプ損失は誤分類で 1 をとり（$u \le 0$）、0 に下がるのは予測が少なくともマージン γ 以上の余裕をもって正解したときであり（$u \ge \gamma$）、その間を線形に補間する。Figure 36.49 に示すとおり、この損失はいたるところで 0-1 損失を上から抑え、$(1/\gamma)$-Lipschitz である。

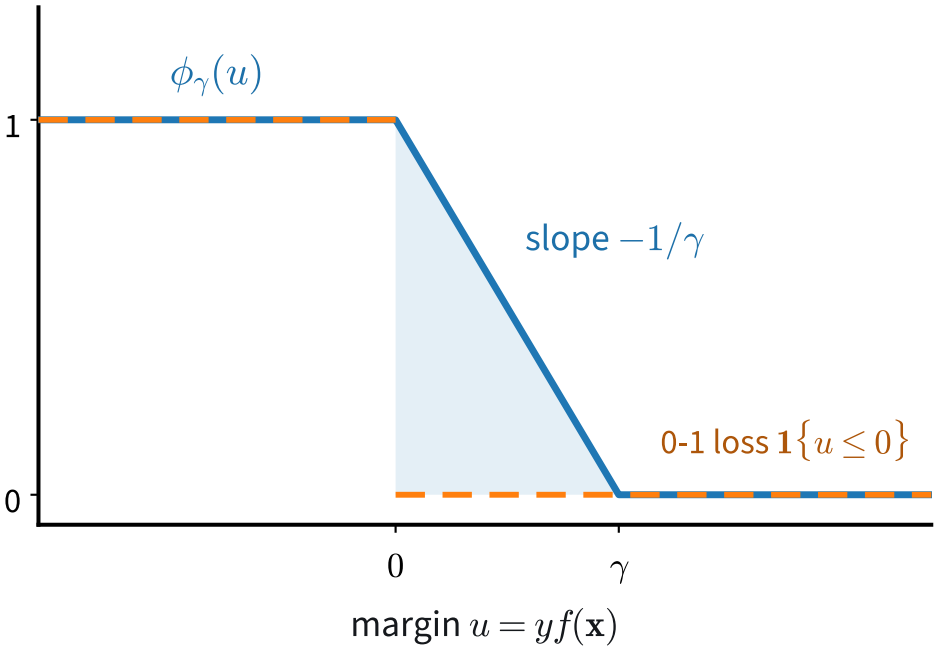


Figure 36.49: ランプ損失 ϕ_γ（青）をマージン $u = yf(\mathbf{x})$ の関数としてプロットしたもの、および 0-1 損失 $\mathbf{1}\{u \le 0\}$（破線）。両者は $0 < u < \gamma$（網掛け部分）を除いて一致し、この領域では予測は正しいもののマージンに届かず、ランプ損失が傾き $-1/\gamma$ で減少する。

標本上の経験ランプ損失を $\hat{R}_\gamma(\mathbf{w}) = \frac{1}{n}\sum_i \phi_\gamma(y_i \langle \mathbf{w}, \mathbf{x}_i \rangle)$ と書く。すべての入力が $\|\mathbf{x}\| \le r$ を満たすとき、縮小補題と線形クラスの命題を組み合わせることで、損失クラスの複雑度は $Br/(\gamma\sqrt{n})$ で上から抑えられる。これを Equation 36.126 に代入すると**マージン上界**（margin bound）が得られる。すなわち、少なくとも $1-\delta$ の確率で、すべての $\mathbf{w}$（$\|\mathbf{w}\| \le B$）に対して同時に次が成り立つ。

$$P(y\langle \mathbf{w}, \mathbf{x} \rangle \le 0) \le \hat{R}_\gamma(\mathbf{w}) + \frac{2Br}{\gamma\sqrt{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}. \tag{36.130}$$

左辺は真の分類誤り率であり、右辺は訓練ランプ損失に計算可能な 2 つの項を加えたものである。これらの数値のスケールを見るため、特定のデータセット上で測定するのではなく比を選ん

だ例として、$Br/\gamma = 10$、$n = 50{,}000$、$\delta = 0.05$ という設定を考えよう。このとき複雑度項は 0.089、信頼項は 0.005 で、合計は 0.095 となる。したがって $B = 10\gamma/r$ を事前に固定しておけば、訓練セットをマージン γ で分離するクラス内の任意の線形分類器は、95% の信頼度で真の誤り率が 9.5% 未満であると保証される。対照的に、線形モデルをランダムラベルにマージン γ で無理やり適合させると Br/γ が押し上げられる。$Br/\gamma \approx 10.6$ を超えると 2 つのペナルティ項の和が 0.1 を上回り、$Br/\gamma \approx 110$ 付近では複雑度項だけで 1 に達して上界は空虚になる。

36.10.2.4　数値で見るラデマッハ複雑度

Equation 36.128 は線形クラスの経験ラデマッハ複雑度をユークリッドノルムの期待値として表しているため、ランダムな符号ベクトルにわたって平均をとることで直接推定できる。以下では、50 個の単位ノルム点に対して $d = 2$、20、および $1{,}000$ 次元でこれを計算する。先ほどの緩みの解析によれば、2 次元では推定値が上界 $Br/\sqrt{n} = 1/\sqrt{50}$ を約 11% 下回り、20 次元に達すると上界から約 1% 以内に収まると予測される。

```
n, B, num_signs = 50, 1.0, 2000
signs = rng.choice([-1.0, 1.0], size=(num_signs, n))  # rows: sign vectors
for d in (2, 20, 1000):
    X = rng.standard_normal((n, d))
    X /= np.linalg.norm(X, axis=1, keepdims=True)       # ||x_i|| = r = 1
    sup_corr = B * np.linalg.norm(signs @ X, axis=1) / n  # closed-form sup
    print(f'd = {d:4d}: Monte-Carlo R_S = {sup_corr.mean():.3f}, '
          f'bound B r / sqrt(n) = {B / np.sqrt(n):.3f}')
```

```
d =    2: Monte-Carlo R_S = 0.123, bound B r / sqrt(n) = 0.141
d =   20: Monte-Carlo R_S = 0.141, bound B r / sqrt(n) = 0.141
d = 1000: Monte-Carlo R_S = 0.141, bound B r / sqrt(n) = 0.141
```

モンテカルロ推定値は平面上で約 0.125 となり、20 次元および 1,000 次元では 0.139〜0.141 の範囲に入って上界の 0.141 にぴたりと張り付き、予測どおりの結果となる。次元を 20 から 1,000 へと 50 倍に増やしても、複雑度は変化しない。

スペクトルの反対側の端には、訓練標本上のいかなるラベル割り当ても厳密に再現できるほど豊かな、**補間**（interpolate）可能なクラスがある。$p = 2n$ 個の標準ガウス特徴量上の最小二乗法を考えよう。ここで特徴量行列 $\Phi \in \mathbb{R}^{n \times p}$ は n 個の点の特徴ベクトルを行として並べたものである。33.5.3.1 節の擬似逆行列を用いると、解 $\mathbf{w} = \Phi^{+}\varepsilon$ は任意の符号ベクトル ε に対して最小ノルム補間解を返し、構成上すべてのパターンで相関 1 を達成する。これらの当てはめすべてを収めるのにどれほど大きな重み球が必要かは、先ほどの命題からわかる。あらゆる符号パターンを補間するクラスは $\widehat{\mathfrak{R}}_S \geq 1$ を満たさねばならず、一方で Equation 36.128 はそれを $Br/\sqrt{n}$ で上から抑えるため、すべての補間解を含む球は $Br/\sqrt{n} \geq 1$ を満たさなければならない。実際の当てはめはこの下限にどれほど近づくだろうか。次に、2,000 個のランダム符号ベクトルに当てはめを行い、その重みノルムを理論予測値 $\sqrt{n/(p-n-1)}$（演習問題 4 で導出）と比較し、それらすべてを包含する最小の球について $Br/\sqrt{n}$ を評価する。

```
p = 2 * n                                   # more features than points
Phi = rng.standard_normal((n, p))
W = np.linalg.pinv(Phi) @ signs.T           # min-norm fit of each sign vector
w_norm = np.linalg.norm(W, axis=0)
r = np.linalg.norm(Phi, axis=1).max()       # input radius: largest row norm
row_norm = np.linalg.norm(Phi, axis=1).mean()
print(f'mean ||w|| = {w_norm.mean():.2f} (predicted '
```

```
        f'{np.sqrt(n / (p - n - 1)):.2f}), largest {w_norm.max():.2f}')
    print(f'typical fit: mean ||w|| * mean row norm / sqrt(n) = '
        f'{w_norm.mean() * row_norm / np.sqrt(n):.2f}')
    print(f'r = {r:.1f}, B r / sqrt(n) with B = largest ||w||: '
        f'{w_norm.max() * r / np.sqrt(n):.1f}')
```

```
mean ||w|| = 1.00 (predicted 1.01), largest 1.41
typical fit: mean ||w|| * mean row norm / sqrt(n) = 1.44
r = 11.8, B r / sqrt(n) with B = largest ||w||: 2.4
```

各当てはめの重みノルムは 1 付近であり、Φ の各行の平均の長さは $\sqrt{p} = 10$ 付近であるため、典型的な当てはめでは $\|\mathbf{w}\| r/\sqrt{n} \approx 1.4$ となる。2,000 通りの符号パターンにわたると、最大の重みノルムは 1.3〜1.8、最長の行ノルムは 11〜12.5 の範囲に入り、2,000 個すべての当てはめを包む球では $Br/\sqrt{n}$ が 2〜3 となる。したがって、純粋なノイズを補間するのに必要なノルムは理論上の下限のわずか 1.5〜3 倍にすぎない。線形クラスの上界 $Br/\sqrt{n}$ は、情報量をもたなくなる領域においてさえほぼタイトなままである。

同じ算術はニューラルネットワークにもそのまま当てはまる。与えられたアーキテクチャが表現できるすべての分類器のクラスを考えよう。ランダム符号 ε 上で訓練したときにほぼすべての点で一致する関数が見つかれば、Equation 36.125 の上限は 1 に近い値をその ε に対してとる。これが典型的な符号ベクトルで起きるなら、ラデマッハ複雑度自体が 1 に近くなる。4.5.1.1 節では、MLP が訓練画像上のランダムラベルに誤差ゼロで適合し、Zhang et al. (2021) の画像分類実験を再現した（これらの実験は 10 クラスで少数のラベル付けを試したものであるため、形式的な証明ではなく経験的証拠である）。このとき損失クラスの複雑度の 2 倍は 1 に近くなり、Equation 36.126 はいかなるテスト誤差も許容してしまう。それにもかかわらず、まったく同じネットワークが実際のラベルで訓練すると良好に汎化する。したがって、テスト誤差を支配している要因は関数クラス全体にわたるいかなる最悪ケースの上界からも見えず、データ、オプティマイザ、そして訓練が到達する特定の解に依存している。その仕組みを見るため、学習された解を閉じた形で解析できるモデルに目を向けよう。

36.10.3 補間と二重降下

4.5.2 節では、隠れ層をランダム初期化のまま凍結したネットワークの幅 h をスイープした。テスト誤差は、補間が初めて可能になり隠れ表現の最小特異値が潰れるちょうどその閾値 $h = n$ で急騰し、さらに h を増やすと再び減少に転じた。ここでは、メカニズムのあらゆる要素を陽に計算できる合成データ上でその構成を再現し、本編の説明に 2 つの要素をつけ加える。特異値による当てはめ重みの閉じた公式と、テスト誤差を真の目標関数に起因する部分とラベルノイズに起因する部分へ分解する式である。特徴量は $\phi(\mathbf{x}) = \mathrm{ReLU}(\mathbf{V}\mathbf{x})$（$\mathbf{V}$ は凍結されたランダム行列）であり、これは**ランダム特徴量**（random-features）モデルとして知られる (Rahimi and Recht 2007)。当てはめるのは線形の出力ヘッド $\mathbf{w}$ のみである。初読の読者は、本章の流れを損なうことなくこのパートを読み飛ばしてよい。

36.10.3.1 最小ノルム補間解

補間閾値 $p = n$ がなぜこれほど劇的な振る舞いを示すのかを理解するため、解を閉じた形で書き下そう。p 個の特徴量と n 個の訓練点があるとき、訓練特徴ベクトルを $\Phi \in \mathbb{R}^{n \times p}$ の各行として積み重ね、$\mathbf{w} = \Phi^{+}\mathbf{y}$ を当てはめる。$p = n$ 未満ではこれは通常の最小二乗法であり、$p = n$ より上ではユークリッドノルム最小の補間解となる。

最小ノルム補間解は、数ある補間解の中から恣意的に選んだものではなく、訓練が見つけ出す解そのものである。二乗損失の勾配 $\Phi^{\top}(\Phi\mathbf{w} - \mathbf{y})$ は訓練特徴ベクトルの線形結合であるため、

$\mathbf{w} = \mathbf{0}$ から出発した勾配降下法は Φ の行空間から決して外に出ない。その行空間内に存在する唯一の補間解が $\Phi^{+}\mathbf{y}$ であり、勾配降下法はノルムに対するペナルティを一切課さなくともそこへ収束する。データに同等に適合する多数の解の中から特定の 1 つを選び出すこのようなオプティマイザの選好は、**暗黙的バイアス**（implicit bias）と呼ばれる。

特異値分解を用いると、閾値未満・閾値上・閾値超の 3 つの領域すべてが明快に理解できる。Φ がフルランク $q = \min(n, p)$ をもち、特異値分解 $\Phi = \sum_{i=1}^{q} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\top}$（$\sigma_1 \geq \cdots \geq \sigma_q > 0$）をもつとしよう。このとき擬似逆行列による解は次式となる。

$$\mathbf{w} = \Phi^{+}\mathbf{y} = \sum_{i=1}^{q} \frac{\langle \mathbf{u}_i, \mathbf{y} \rangle}{\sigma_i} \mathbf{v}_i. \tag{36.131}$$

$\mathbf{y}$ の各左特異ベクトル $\mathbf{u}_i$ 方向の成分はその特異値 σ_i で割られるため、最小特異値 $\sigma_{\min} = \sigma_q$ が最も強力な増幅器として働く。$\mathbf{y}$ のうち $\mathbf{u}_q$ 方向に落ちる成分は何であれ $\mathbf{w}$ に $1/\sigma_{\min}$ 倍に拡大されて入り込み、この増幅はラベルノイズと、特徴量ではうまく近似できない真の目標関数の成分の両方を直撃する。

$p < n$ のとき、当てはめ予測 $\Phi\mathbf{w}$ は単に $\mathbf{y}$ を p 次元の $\mathbf{u}_i$ が張る空間へ射影し、残りを訓練残差として残す。しかし $p = n$ では、それら n 個の方向が $\mathbb{R}^n$ 全体を張るため、当てはめは厳密になり、表現しにくい成分を吸収する残差が一切残らない。正方のランダム ReLU 特徴量行列は概収束の意味でほぼ確実に正則であるが、条件が**良い**ことはめったにない。その最小特異値は典型的に極小であり、33.5.3.2 節の条件数解析により、解は $\mathbf{v}_q$ 方向に大きく飛び出す。$p = n$ を超えて特徴量を追加すると、行列が $\mathbb{R}^n$ を張るための余裕が増え、$\sigma_{\min}$ は再び上昇する。

閾値を超えると、特徴量集合が入れ子になっている限り（$(p+1)$ 特徴量モデルがそれまでの p 個の特徴量をすべて保持して新たに 1 つ追加する構成）、特徴量を追加するにつれて解のノルムは減少するほかない。これは 4.5.2 節のパディングの議論から従う。新しい特徴量に重みゼロを割り当てれば、p 特徴量の最小ノルム補間解はそのまま同じノルムをもつ妥当な $(p+1)$ 特徴量の補間解になるため、新たな最小ノルムはそれより大きくなり得ないからである。なお、この単調性は厳密な補間に依拠している。$p = n$ 未満では最小二乗解は補間を行わず、そのノルムは実際には p とともに**増大**する。以下のスイープでは、約 0.5（$p = 2$）から 2 近く（$p = 15$）まで上昇する。

線形クラスの上界は、スイープに沿って監視すべき自然な量を示唆している。すなわち**容量積**（capacity product）$\|\mathbf{w}\| \cdot \max_i \|\phi(\mathbf{x}_i)\| / \sqrt{n}$ であり、これは当てはめた重みを含む最小の球に対する $Br/\sqrt{n}$ に対応する。このスイープでは有界な分類損失ではなく非有界な回帰目標に対する二乗損失を用いるため、この量は形式的な上界としてではなく純粋な診断指標として借用する。閾値を超えると、特徴量半径が $\sqrt{p}$ のように増大する一方で $\|\mathbf{w}\|$ は縮小するため、積全体の正味の挙動は事前には自明でない。

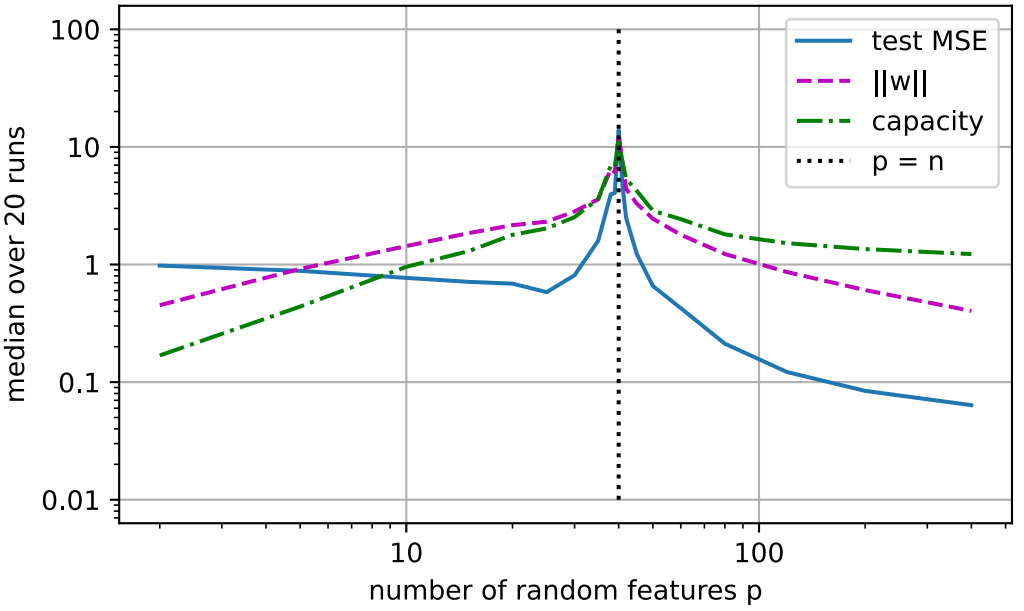
36.10.3.2 二重降下のスイープ

具体的な問題上で、これらの量が連動して変化する様子を観察しよう。$n = 40$ 個の訓練点を線形教師 $y = \langle \beta, \mathbf{x} \rangle$（$d = 15$ 次元、$\|\beta\| = 1$）から抽出し、標準偏差 0.1 のガウスラベルノイズを加える。入れ子になった ReLU ランダム特徴量を構成し、`pinv` によって線形ヘッドを当てはめることで、$p = 40$ 未満では最小二乗解、それより上では最小ノルム補間解を得る。閾値のピークが鋭く解像されるよう、p を 2 から 400 まで（$p = 39$、40、41 を含めて）スイープする。各 p において、テスト MSE、訓練 MSE、$\|\mathbf{w}\|$、$\sigma_{\min}$、および容量積に加え、36.10.3.3 節で解析するテスト誤差の 2 つの成分、すなわち**クリーン成分**（clean part、ノイズのないラベルに当てはめたときのテスト誤差）と**ノイズ成分**（noise part、ラベルノイズによって注入される追加のテスト誤差）を記録する。20 回の独立な試行にわたって中央値をとることで、曲線を読みやすく保つ。

```
n, n_test, d_in, p_max, trials, noise = 40, 500, 15, 400, 20, 0.1
p_grid = np.array([2, 5, 10, 15, 20, 25, 30, 35, 38, 39, 40, 41, 42, 45, 50,
                   60, 80, 120, 200, 400])
keys = ('test', 'train', 'norm', 's_min', 'capacity', 'clean', 'noise',
        'ratio_q', 'ratio')
stats = np.zeros((len(keys), trials, len(p_grid)))
ols = np.zeros(trials)                              # least squares on x itself
for tr in range(trials):
    beta = rng.standard_normal(d_in)
    beta /= np.linalg.norm(beta)                    # teacher: y = <beta, x>
    X, X_te = (rng.standard_normal((size, d_in)) for size in (n, n_test))
    y_clean, y_te = X @ beta, X_te @ beta
    y = y_clean + noise * rng.standard_normal(n)    # noisy training labels
    ols[tr] = np.mean((X_te @ np.linalg.lstsq(X, y, rcond=None)[0] - y_te) ** 2)
    V = rng.standard_normal((p_max, d_in)) / np.sqrt(d_in)
    Phi, Phi_te = np.maximum(X @ V.T, 0), np.maximum(X_te @ V.T, 0)
    for j, p in enumerate(p_grid):                  # nested feature prefixes
        F, F_te = Phi[:, :p], Phi_te[:, :p]
        P = np.linalg.pinv(F)                       # least squares / min norm
        w = P @ y
        r = np.linalg.norm(F, axis=1).max()         # feature radius
        _, S, Vt = np.linalg.svd(F, full_matrices=False)
        ratio = ((np.linalg.norm(F_te @ Vt.T, axis=0) ** 2 / n_test)
                 / (S ** 2 / n))                    # test/train feature power
        stats[:, tr, j] = (
            np.mean((F_te @ w - y_te) ** 2),                # test MSE
            np.mean((F @ w - y) ** 2),                      # training MSE
            np.linalg.norm(w),                              # ||w||
            S.min(),                                        # sigma_min
            np.linalg.norm(w) * r / np.sqrt(n),             # capacity
            np.mean((F_te @ (P @ y_clean) - y_te) ** 2),    # clean part
            noise ** 2 * np.linalg.norm(F_te @ P) ** 2 / n_test,  # noise part
            ratio[-1], ratio.mean())                # along v_q, average
med = dict(zip(keys, np.median(stats, axis=1)))
```

下の図は、テスト MSE、解のノルム、および容量積の中央値を両対数軸上にプロットしたものである。

```
d2l.plot([p_grid] * 3 + [[n, n]],
         [med['test'], med['norm'], med['capacity'], [1e-2, 1e2]],
         'number of random features p', 'median over 20 runs',
         legend=['test MSE', '||w||', 'capacity', 'p = n'], xscale='log',
         yscale='log', fmts=('-', 'm--', 'g-.', 'k:'), figsize=(6, 3.5))
```



下の表は、代表的な 7 つの特徴量数における詳細な内訳を出力したものである。最後の 2 列は 36.10.3.3 節に対応する量であり、最終行は生の入力に直接最小二乗法を当てはめた古典的ベースラインを報告している。

```
print('   p ' + ' '.join(f'{key:>8}' for key in keys))
for p in (15, 38, 40, 42, 50, 80, 400):
    row = [med[key][p_grid == p][0] for key in keys]
    print(f'{p:4d} ' + ' '.join(f'{v:8.2g}' for v in row))
print(f'least squares on the {d_in} raw inputs: test MSE {np.median(ols):.4f}')
```

```
   p     test    train     norm    s_min capacity    clean    noise  ratio_q    ratio
  15     0.71     0.31      1.8     0.94      1.3     0.72   0.0069        4      1.8
  38        4   0.0049      6.5    0.087      6.9      4.5     0.25  5.5e+02       27
  40       14  4.6e-29       13    0.019       12      9.9      3.3  1.2e+04  3.3e+02
  42      2.5  1.4e-29      4.4    0.069      5.4      2.7      0.3  7.1e+02       30
  50     0.66  5.5e-30      2.5     0.24      2.9     0.63    0.061       71      6.1
  80     0.21  7.2e-30      1.2     0.74      1.8     0.19    0.021        9      2.1
 400    0.064  1.5e-29      0.4      2.8      1.2    0.056   0.0093      1.9     0.93
least squares on the 15 raw inputs: test MSE 0.0055
```

この表は、閾値未満・閾値上・閾値超の 3 つの領域に分けて読むと最もわかりやすい。

閾値未満では、モデルは訓練点数より少ないパラメータをもち、古典的に振る舞う。テスト誤差はまず約 1（$p=2$）から約 0.65〜0.85（$p=15$ 付近）へと下がり、その後 p が 40 に近づくにつれて再び上昇して、おなじみの古典的な U 字曲線を描く。この領域と何かを比較する前に、2 つの注意点が欠かせない。第 1 に、最良の地点でさえこの古典的な当てはめは貧弱である。目標値の分散は 1（つねにゼロと予測したときの誤差）であるため、U 字曲線の最小点でも分散の約 3 分の 1 しか取り除けていない。第 2 に、はるかに優れた古典的モデルが存在する。真の教師は 15 個の生の入力に関して線形であるため、生の入力に対する通常の最小二乗法は期待テスト誤差 $0.01 \cdot 15/(40-15-1) \approx 0.006$（ノイズ分散に $d/(n-d-1)$ を $d=15$ として掛けた値）を達成し、これは出力の最終行からも確認できる。ランダム特徴量モデルは設計上ハンディキャ

ップを負っているのである。ReLU の尾根から線形関数を再構成しなければならず、40 点では到底足りない。

閾値上では、3 つの列が連動して動く。訓練誤差は 10^{-3} の数倍程度（$p = 38$）から約 10^{-28} へと急落し、浮動小数点精度の範囲で厳密な補間となる。同時に、最小特異値は 0.01〜0.04 付近まで落ち込み、$p = 38$ や $p = 42$ のときより数分の 1 に小さくなる。Equation 36.131 は $\sigma_{\min}$ で割るため、解は爆発する。$\|\mathbf{w}\|$ は 10〜30 に跳ね上がり、テスト MSE は 10〜70 へと急騰して U 字曲線の最小値より 1〜2 桁高くなる。これらの範囲が広いのは、正方ガウス行列で $\mathbb{E}[1/\sigma_{\min}^2] = \infty$ となるのとちょうど同じくピークが裾の重い分布をもつためであり、20 試行の中央値でさえ実行ごとに目立って変動する。

閾値超では、急騰したすべての量が再び下がる。最小特異値は約 2.8（$p = 400$）まで回復する。パディングの議論が保証するとおり、$\|\mathbf{w}\|$ はすべての試行で（中央値だけでなく）単調に減少し、約 0.4 に達する。テスト MSE もそれに追随して低下し、約 0.2（$p = 80$）、さらに 0.04〜0.07（$p = 400$）まで下がる。その一方で、1 つの列は完全に沈黙する。訓練誤差は浮動小数点の範囲でゼロのまま $p = 40$ 以降の全域に張り付くため、訓練損失に基づくいかなる基準も $p = 42$ と $p = 400$ を見分けられない（両者のテスト誤差が 1 桁以上異なるにもかかわらずである）。解のノルム、$\sigma_{\min}$、および容量積はいずれも両者を明確に識別する。

では、このスイープは実際のところ何を証明しているのだろうか。劇的な曲線が一見して示唆するほど多くはない。入れ子になったランダム特徴量の族の内部では、最も幅の広いモデルがすべてのラベルノイズを暗記する地点をはるかに超えてなお、最良の過少パラメータモデルを含むあらゆる狭いモデルを約 10 倍の差で打ち負かす。これは紛れもなく驚くべき現象である。しかしそれでも、生の入力に対する通常の最小二乗法にはさらに 10 倍の差で敗れる。補間モデルが正しく特定された小さなモデルを上回れるかどうかは別の問いであり、この設定での答えは否である。

容量積は 10〜35 付近でピークに達し（$p = 40$）、その後は約 1.2〜1.3（$p = 400$）まで低下する。テスト誤差曲線の形状を映し出してはいるが、これを汎化保証書と取り違えてはならない。本節のどの定理も非有界な二乗損失をカバーしておらず、この積は目標値の再スケーリングに対して不変でもなく、さらに B はデータを見る前に固定されたのではなく当てはめ後の解から読み取ったものだからである。

最後に、ピークを引き起こしている原因はラベルノイズでは**ない**ことに注意しよう。`noise = 0` としてスイープを再実行してもほぼ同じ高さのピークが現れ、その理由は表からわかる。$p = 40$ の時点で、テスト誤差のクリーン成分はすでにノイズ成分より大きく、典型的には数倍に達しているからである。線形教師は有限個の ReLU 特徴量の線形結合としては書けないため、$\mathbf{u}_q$ 方向の近似誤差こそが $\sigma_{\min}$ によって割られる正体なのである。教師を最初の 10 個の特徴量の厳密な線形結合に置き換えてノイズを取り除けば、ピークは完全に消滅する（演習問題 6）。

36.10.3.3 良性過学習

謎が 1 つ残っている。$p = 400$ において、モデルはノイズを含むすべての訓練ラベルを残差ゼロで補間しているにもかかわらず、スイープ中で最小のテスト誤差を達成している。なぜその領域では、ノイズへの適合がこれほどわずかな悪影響しかもたらさないのだろうか。

テスト誤差を計算可能な 2 つの部分に分解することで、この問いに答えることができる。ノイズを含む訓練ラベルを $\mathbf{y} = \mathbf{y}_0 + \xi$ と書く。ここで $\mathbf{y}_0$ はノイズのないラベル、ξ は各例あたり分散 $\sigma_\xi^2 = 0.01$ をもつ独立なノイズである。擬似逆行列の線形性により $\mathbf{w} = \Phi^{+}\mathbf{y}_0 + \Phi^{+}\xi$ となるため、テスト予測は $\Phi_{\text{te}}\Phi^{+}\mathbf{y}_0 + \Phi_{\text{te}}\Phi^{+}\xi$ の 2 項に分かれる。ξ は平均ゼロであるため交差項は期待値において消え、n_{te} 個のテスト点にわたる期待テスト MSE の厳密な分解が得られる。

$$\mathbb{E}_{\xi}\big[\text{test error}\big] = \underbrace{\frac{1}{n_{\text{te}}}\big\|\Phi_{\text{te}}\Phi^{+}\mathbf{y}_0 - \mathbf{y}_{\text{te}}\big\|^2}_{\text{clean part}} + \underbrace{\frac{\sigma_\xi^2}{n_{\text{te}}}\big\|\Phi_{\text{te}}\Phi^{+}\big\|_F^2}_{\text{noise part}}.$$

表の `clean` 列と `noise` 列はこれら 2 つの項を出力している。$p = 40$ ではノイズ成分は約 1〜4（ラベルノイズ分散 0.01 の 100 倍以上）となり、クリーン成分はさらに大きい。$p = 42$ になるとノイズ成分は 0.1 の数倍程度まで下がり、$p = 80$ で約 0.02、$p = 400$ では 0.01 をわずかに下回る。同じ区間において、クリーン成分はピーク時の約 8〜60 から約 0.05（$p = 400$）まで低下するため、閾値以降の改善の大部分は目標関数の近似精度の向上に由来している。

$p = 40$ で急騰を引き起こしたまさにその仕組みである「小さな特異値による除算」が、なぜ $p = 400$ ではほぼ無害になるのだろうか。Equation 36.131 より、$\mathbf{u}_i$ 方向のノイズ成分は $1/\sigma_i$ 倍にスケーリングされて $\mathbf{v}_i$ 方向の重みに入り込み、テスト特徴量の $\mathbf{v}_i$ 方向成分に比例してテスト予測を動かす。$\|\Phi_{\text{te}}\Phi^{+}\|_F^2 = \sum_i \|\Phi_{\text{te}}\mathbf{v}_i\|^2/\sigma_i^2$ と $\sigma_i^2 = \|\Phi\mathbf{v}_i\|^2$ を併せて用いると、$p \geq n$ におけるノイズ成分は次の示唆に富む形をとる。

$$\sigma_\xi^2 \cdot \frac{1}{n}\sum_{i=1}^{n} \frac{\|\Phi_{\text{te}}\mathbf{v}_i\|^2/n_{\text{te}}}{\|\Phi\mathbf{v}_i\|^2/n},$$

これはノイズ分散に、各右特異方向 $\mathbf{v}_i$ に沿ったテスト特徴量の二乗平均エネルギーと訓練特徴量の二乗平均エネルギーの比の平均を掛けたものである。極小の特異値 σ_i が問題を引き起こすのは、テスト入力が同じ方向に大きなエネルギーをもつ場合に限られる。$p = n$ における極小の $\sigma_{\min}$ は 40 個の訓練点にたまたま生じた偶然の産物であり、$\mathbf{v}_q$ 方向ではテスト特徴量の二乗平均が訓練特徴量より典型的に数千倍も大きくなる。対照的に $p = 400$ では、小さな σ_i は訓練点でもテスト点でも同様に特徴量が小さい方向を表している。表の最後の 2 列がこのことを裏付けており、`ratio_q`（$\mathbf{v}_q$ 方向）は約 2 にすぎず、`ratio`（全方向の平均）は 1 をわずかに下回るため、小さな分母が小さな分子によって打ち消される。過学習が**良性**（benign）と呼ばれるのは、この意味においてである。なお、良性とはゼロではなく**有界**という意味であることに注意しよう。$p = 400$ においても、予測は依然としてノイズを含む 1 個の訓練ラベルとほぼ同程度の分散を抱えている。

Bartlett et al. (2020) は線形回帰に対して、入力の共分散行列の 2 つの有効ランクを通じてこの現象を厳密に定式化しており、演習問題 7 でこれを直接計算する。深層ニューラルネットワークでは特徴量自体が学習され、そのスペクトルも訓練を通じて変化するため、訓練済みネットワークのテスト誤差を第一原理から予測する理論はいまだ手の届かないところにある。本節で扱ったすべての形式的な上界はデータを見る前に選んだクラス内の最悪の関数をカバーするものであるのに対し、実際に運用する予測器はオプティマイザが見つけ出す特定の解である。関数クラスだけでなくアルゴリズムとデータに依存する保証によってその隔たりを埋めることこそ、本節が残す中心的な未解決問題である。

36.10.4 まとめ

データを見る前に固定した単一の関数に対する集中上界は、標本上で最良のスコアを出したという理由で選ばれたモデルをカバーしない。一様収束は、クラス内のすべての関数の推定誤差を同時に上から抑えることでこの隔たりを埋める。有限クラス $\mathcal{F}$ に対しては、ヘフディングの不等式

とユニオンバウンドにより、追加データ例 $\log|\mathcal{F}|/(2t^2)$ 個（許容誤差 t のもと）のコストでこれが達成される。無限クラスに対しては、ゴースト標本を用いた対称化によって最悪ケースのギャップがランダム符号との相関に帰着し、損失クラスのラデマッハ複雑度の 2 倍が経験リスクに加わる。ノルム有界な線形モデル（$\|\mathbf{w}\|_2 \le B$、$\|\mathbf{x}_i\|_2 \le r$）では、この複雑度は次元に依存せず高々 $Br/\sqrt{n}$ となる。一様収束が限界に達するのは、ランダムラベルに適合する深層ネットワークのように任意の符号パターンを補間できるほど柔軟なクラスにおいてであり、そこでは上界が空虚になる。ランダム特徴量のスイープは、そのような領域でもなお汎化が成功し得る理由を鮮やかに示している。テスト誤差は $\sigma_{\min}$ が潰れる場所で急騰し、最小ノルム補間解が縮小するにつれて低下するが、これらは関数クラス全体ではなく学習された解の性質なのである。

36.10.5 演習問題

1. **マクダイアミッドの不等式。**●McDiarmid (1989) の不等式は、ヘフディングの不等式を和から**有界差分**をもつ関数へと拡張する。いま、i 番目の引数のみを変化させたときの $g(x_1,\dots,x_n)$ の値の変化が高々 c_i であり、かつ $X_1,\dots,X_n$ が独立であるとする。このとき、

$$P\big(g(X_1,\dots,X_n) - \mathbb{E}[g(X_1,\dots,X_n)] \ge t\big) \le \exp\left(-\frac{2t^2}{\sum_{i=1}^{n} c_i^2}\right), \tag{36.132}$$

が成り立ち、下側の裾に対しても同じ上界が成り立つ。この主張の証明にはマルチンゲールを用いるため、ここでは証明なしで認めることとする。

   1. この不等式から、独立な項 $X_i \in [a_i, b_i]$ の平均に対するヘフディングの不等式を導出せよ。
   2. $\Phi(S) = \sup_{g\in\mathcal{G}} \big(R(g) - \hat{R}_S(g)\big)$ を、損失クラス $\mathcal{G}$（値域は $[0,1]$）に対して定める。Φ が有界差分 $c_i = 1/n$ をもつことを示せ。さらにマクダイアミッドの不等式を本文の対称化上界 $\mathbb{E}_S[\Phi(S)] \le 2\mathfrak{R}_n(\mathcal{G})$ と組み合わせ、Equation 36.126 を導出せよ。
   3. $[0,1]$ 内のデータに対し、標本平均は有界差分 $1/n$ をもつ。奇数サイズ n の $[0,1]$ 内の標本であって、1 点を変更すると標本中央値が 1 だけ動くものを構成せよ。このことは、36.5 節における中央値のブートストラップ推定値にマクダイアミッドの不等式でエラーバーを付けることに対して何を意味するか。

2. **マサールの補題と ℓ_1 球。**●有限クラスに対するマサールの補題と、ℓ_1 有界な線形クラスのラデマッハ複雑度は、いずれもサブガウス変数の最大値の期待値に対する 1 つの上界に基づいている。

   1. $Z_1,\dots,Z_N$ を中心化された分散プロキシ σ^2 のサブガウス変数、すなわち $\mathbb{E}[e^{\lambda Z_j}] \le e^{\lambda^2\sigma^2/2}$ がすべての λ で成り立つ変数とする（互いに独立である必要はない）。$\mathbb{E}[\max_j Z_j] \le \sigma\sqrt{2\log N}$ が成り立つことを、$e^{\lambda \mathbb{E}[\max_j Z_j]}$ を $\sum_j \mathbb{E}[e^{\lambda Z_j}]$ で上から抑えて $\lambda > 0$ について最適化することにより示せ。さらにこれから、マサールの補題 $\widehat{\mathfrak{R}}_S(\mathcal{F}) \le \sqrt{2\log|\mathcal{F}|/n}$ を $[-1,1]$ に値をとる有限クラスに対して導け。
   2. $\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x}\rangle : \|\mathbf{w}\|_1 \le B\}$ とし、$\|\mathbf{x}_i\|_\infty \le r$ がすべての i で成り立つとする。Equation 36.128 の証明にならい、コーシー・シュワルツの不等式の代わりに双対性 $\sup_{\|\mathbf{w}\|_1\le B} \langle \mathbf{w}, \mathbf{v}\rangle = B\|\mathbf{v}\|_\infty$ を用いて、$\widehat{\mathfrak{R}}_S(\mathcal{F}) \le Br\sqrt{2\log(2d)/n}$ を示せ。なぜここでは次元 d が現れ、しかも対数的にしか効かないのか。
   3. 入力が $\{-1,+1\}^d$ に属し、目標の重みベクトルが k 個の非ゼロ成分（それぞれ ± 1 に等しい）をもつとする。目標ベクトルを含む最小の半径 B をそれぞれ用いたとき、(b) の上界を Equation 36.128 と比較せよ。どの範囲の k に対して ℓ_1 上界の方が小さくなるか（$d = 10^4$ の場合）。

3. **マージン上界。**●$\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \|\mathbf{w}\|_2 \le B\}$、$\|\mathbf{x}\|_2 \le r$ とし、ラベルを $y \in \{-1, +1\}$ とする。またランプ損失 $\phi_\gamma(u) = \min\{1, \max\{0, 1 - u/\gamma\}\}$（固定した $\gamma > 0$）をマージン $u = yf(\mathbf{x})$ に作用させる。

    1. $\phi_\gamma(yf(\mathbf{x}))$ が少なくとも 0-1 損失 $\mathbf{1}\{yf(\mathbf{x}) \le 0\}$ 以上であり、ϕ_γ が $(1/\gamma)$-Lipschitz であることを示せ。さらに、各予測 $f(\mathbf{x}_i)$ に固定されたラベル y_i を掛けても $\widehat{\mathfrak{R}}_S(\mathcal{F})$ が変化しないことを示せ。
    2. (a)、縮小補題 Equation 36.129、Equation 36.126、および Equation 36.128 を組み合わせて Equation 36.130 を導け。$n = 50{,}000$、$\delta = 0.05$ のとき、複雑度項と信頼項の和を小さく保てる最大の比 Br/γ（和が 0.1 未満となる上限）を求めよ。
    3. この上界はデータの抽出前に固定した半径 B に対して成り立つが、訓練された重みベクトルのノルムは事後にしかわからない。半径 $B_j = 2^j$、失敗確率 $\delta_j = \delta/(j(j+1))$ としてすべての $j \ge 1$ に上界を適用し、j にわたってユニオンバウンドをとれ。データを見た後に選ばれた $\|\mathbf{w}\| > 1$ を満たす重みベクトルに対して成り立つ上界を書き下せ。さらに、半径を選べる自由度の代償が複雑度項において高々 2 倍と、オーダー $\sqrt{\log\log \|\mathbf{w}\| / n}$ の追加項にとどまることを示せ。

4. [code] **ランダム符号への当てはめ。**●36.10.2.4 節の 2 番目のコードセルでは、$2{,}000$ 個の符号ベクトル $\varepsilon \in \{-1, +1\}^n$ に対し $\mathbf{w} = \Phi^{+}\varepsilon$ を $p = 2n$ 個の標準ガウス特徴量上で当てはめ、得られた重みのノルムを測定している。

    1. $p < n$ とし、Φ が列フルランクをもつとする。相関 $\frac{1}{n}\sum_i \varepsilon_i (\Phi \mathbf{w})_i$ の符号に関する期待値が p/n になることを示せ。行列 $\Phi\Phi^{+}$ は射影である。
    2. $p > n$ とし、r を Φ の最大の行ノルムとし、B をクラス $\{\|\mathbf{w}\| \le B\}$ がすべての符号ベクトルの補間解を含むほど十分大きくとる。このとき $Br/\sqrt{n}$ が 1 より小さくなり得ない理由を説明せよ。標準ガウス行列 Φ で $p > n + 1$ のとき、$\mathbb{E}[(\Phi\Phi^{\top})^{-1}] = \mathbf{I}_n/(p - n - 1)$ が成り立つ。この恒等式を用いて、典型的な $\|\mathbf{w}\|$、および $\|\mathbf{w}\|$ に Φ の典型的な行ノルムを掛けて $\sqrt{n}$ で割った典型値を、p と n の関数として予測せよ。
    3. $n = 50$ とし、$p \in \{10, 25, 45, 55, 100, 400, 1000\}$ に対してセルを再実行して（$p < n$ ではノルムの代わりに相関を出力すること）、(a) および (b) と比較せよ。(b) の容量値は $p \to \infty$ のときどうなるか。また、最大の行ノルムを用いた版が 1 より上にとどまるのはなぜか。

5. [code] **リーダーボード攻撃。**●●あるリーダーボードが、提出された各ラベルベクトルの 0-1 損失を $n = 4{,}000$ 例（ラベル $\mathbf{y} \in \{0, 1\}^n$）のホールドアウトセット上で報告するものとする。Blum and Hardt (2015) の**ブースティング攻撃**はタスクに関する知識を一切使わない。k 個の一様ランダムなラベルベクトルを提出し、報告された損失が高々 $\frac{1}{2}$ であったものを残して、それらの座標ごとの多数決を提出するのである。新たなデータ上では、この多数決の損失は $\frac{1}{2}$ である。

    1. 一様ランダムな $\mathbf{y}$ と $k \in \{10, 30, 100, 300, 1000, 3000\}$ に対してこの攻撃を実装し、多数決に対して報告される損失を記録せよ。その値と $\frac{1}{2}$ との距離を、事前登録された Equation 36.124 による $k+1$ モデルのプール（$\delta = 0.05$）に許容されるずれと比較せよ。どの k から攻撃がこの許容ずれを超え、また命題のどの仮定に違反しているか。
    2. Blum and Hardt (2015) は報告される損失が $\frac{1}{2} - \Omega(\sqrt{k/n})$ になることを証明しているが、定数は与えていない。ガウス近似を用いて発見的にこの定数を導出せよ。まず残されたベクトルが $\mathbf{y}$ に対して示す過剰一致率の期待値を見積もり、次にそのような約 $k/2$ 個のベクトルの多数決が 1 つの座標で正解する確率を見積もること。得られた定数を (a) の測定値と比較せよ。

3. Blum and Hardt (2015) の **Ladder** メカニズムは、提出がそれまでに公開された最良スコアをステップ幅 η より大きく上回ったときにのみ新しいスコアを η の倍数に丸めて公開し、それ以外では前回のスコアを繰り返す。Ladder に対しては、攻撃者は公開スコアを更新させたベクトルしか残せない。両者を $\eta \in \{0.005, 0.01\}$ および $k \in \{100, 1000, 3000\}$ で実装せよ。このとき多数決は $\frac{1}{2}$ をどれだけ下回り、(a) の許容ずれと比べてどうなるか。

6. [code] **正方行列のケース**。●36.10.3.2 節の二重降下実験のスイープを用いよ。

   1. 保存されたノルムから、閾値未満では入れ子の単調性が成り立たないことを示し、その領域でパディングの議論が適用できない理由を説明せよ。
   2. $p = n$ において、$\mathbf{w}$ を Equation 36.131 に沿って分解せよ。最小特異方向の係数 $|\langle \mathbf{u}_n, \mathbf{y} \rangle| / \sigma_{\min}$ と $\|\mathbf{w}\|$ を試行にわたる中央値として並べて報告せよ。この 1 つの方向がノルムのどれだけの割合を占めているか。
   3. `noise = 0` としてスイープを再実行せよ。次に教師を、最初の 10 個の特徴量の厳密な線形結合である目標関数 $y = \langle \mathbf{c}, \phi_{1:10}(\mathbf{x}) \rangle$（$\|\mathbf{c}\| = 1$、ノイズなし）に置き換えよ。2 つの曲線の様子を述べよ。それぞれの場合において、$\sigma_{\min}$ は何を割っているか。また、目標関数が 10 個の特徴量しか使っていないとき、特徴量を追加するにつれて最小ノルム補間解がかえって**悪化**するのはなぜか。
   4. 標準偏差 0.1 のラベルノイズを (c) の目標関数に加えよ。このときテスト誤差が最も高くなるのはどこか、そのピークはノイズ分散と比べてどうなるか、また誤差が最も低くなるのはどこか。

7. [extended] **良性スペクトル**。●●Bartlett et al. (2020) は、線形回帰における最小ノルム補間解がいつ小さな過剰リスクをもつかを特徴づけており、これは 36.10.3.3 節で定性的に説明したメカニズムに対応する。$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$、$\Sigma = \mathrm{diag}(\lambda_1, \dots, \lambda_p)$（$\lambda_1 \geq \lambda_2 \geq \cdots$）とし、$y = \langle \theta^*, \mathbf{x} \rangle + \xi$（$\xi \sim \mathcal{N}(0, \sigma^2)$）とし、$n < p$ 個の訓練入力を $\mathbf{X} \in \mathbb{R}^{n \times p}$ に積み重ねて $\hat{\theta} = \mathbf{X}^{+}\mathbf{y}$ とおく。過剰リスクは $(\hat{\theta} - \theta^*)^{\top} \Sigma (\hat{\theta} - \theta^*)$ である。

   1. ノイズに関する過剰リスクの期待値が、統計的バイアス項 $\theta^{*\top}(\mathbf{I} - \mathbf{P})\Sigma(\mathbf{I} - \mathbf{P})\theta^*$（ただし $\mathbf{P} = \mathbf{X}^{+}\mathbf{X}$）と分散項 $\sigma^2 \,\mathrm{tr}\big((\mathbf{X}^{+})^{\top}\Sigma\mathbf{X}^{+}\big)$ の和になることを示せ。両方の項を実装せよ（テストセットは不要である）。
   2. 同論文の有効ランクは $\rho_k = \sum_{i>k} \lambda_i / \lambda_{k+1}$ および $\varrho_k = \big(\sum_{i>k} \lambda_i\big)^2 / \sum_{i>k} \lambda_i^2$ で定義される。k^* を、条件を満たす最小の k（$\rho_k \geq b n$ となるもの）とおくと、分散項は定数因子を除いてオーダー $\sigma^2 (k^*/n + n/\varrho_{k^*})$ となる。ここでは $b = 1$ とせよ。$n = 100$、$\sigma = 1$ とし、$\lambda_i = 1$ をもち単位ノルムの θ^* を含む 5 つの強い方向と、q 個の弱い方向（それぞれ $\lambda_i = 1/q$）を $q \in \{100, 300, 10^3, 3 \cdot 10^3, 10^4\}$ に対して用いよ。測定された分散を $\sigma^2 (k^*/n + n/\varrho_{k^*})$ と比較せよ。$q = 100$ では何が起き、それはなぜか。
   3. べき乗則スペクトル $\lambda_i = i^{-\alpha}$（$p = 40n$ で打ち切り）に対して、$\alpha \in \{0.5, 1, 2\}$、$\theta^* = \mathbf{e}_1$、$n \in \{50, 100, 200, 400\}$ として同様の実験を行え。どのスペクトルで過剰リスクが n とともに減少するか。それ以外のスペクトルではバイアスと分散のどちらが破綻し、ρ_0/n、k^*/n、および n/ϱ_{k^*} はそれをどのように表しているか。

8. **幅ではなくノルム：2 層ネットワーク**。●●$\mathcal{F}_1 = \{\mathbf{x} \mapsto \mathrm{ReLU}(\langle \mathbf{v}, \mathbf{x} \rangle) : \|\mathbf{v}\|_2 \leq B_2\}$ および $\mathcal{F}_2 = \{\mathbf{x} \mapsto \sum_{j=1}^{h} w_j \,\mathrm{ReLU}(\langle \mathbf{v}_j, \mathbf{x} \rangle) : \|\mathbf{w}\|_1 \leq B_1,\ \|\mathbf{v}_j\|_2 \leq B_2\}$ を、$\|\mathbf{x}_i\| \leq r$ を満たす点上で考える。

   1. $\widehat{\mathfrak{R}}_S(\mathcal{F}_1) \leq B_2 r / \sqrt{n}$ を示せ。
   2. $\mathcal{F}_2$ が、凸包を B_1 倍した集合（$\mathcal{F}_1 \cup (-\mathcal{F}_1)$ の凸包の定数倍）に含まれること、およびクラスの凸包をとっても経験ラデマッハ複雑度は変わらないことを示せ。ヒント：

$\sum_j w_j f_j = \|\mathbf{w}\|_1 \sum_j \frac{|w_j|}{\|\mathbf{w}\|_1} (\pm f_j)$。

3. $0 \in \mathcal{F}_1$ を用いて $\mathbb{E}_\varepsilon \sup_{f \in \mathcal{F}_1} |\sum_i \varepsilon_i f(\mathbf{x}_i)| \leq 2n \widehat{\mathfrak{R}}_S(\mathcal{F}_1)$ を示し、$\widehat{\mathfrak{R}}_S(\mathcal{F}_2) \leq 2B_1 B_2 r/\sqrt{n}$ を導け。
4. 幅 h はこの上界に現れない。この上界は両方の層に対する重み減衰について何を示唆するか。また、なぜこれでは 4.5.1.1 節のランダムラベル実験を説明できないのか。

36.11 ナイーブベイズ

3.3.5 節では、画像が与えられたときのラベルの確率に対して直接勾配降下法を実行し、Fashion-MNIST 上でソフトマックス回帰を訓練した。この当てはめでは 7,850 個のパラメータ（各ピクセルと各クラスの組に対する重み 1 つずつと、10 個のオフセット）が得られ、約 83% の正解率を達成し、3.4 節の較正（calibration）チェックも通過した。しかし、分類器を構築する方法は勾配降下法だけではない。代わりに、典型的なコートやサンダルがどのような見た目をしているかを捉えることで各クラスがどのように画像を生成するかを記述し、ベイズの定理によってその記述を逆転させて予測を得ることもできる。この別のアプローチはどの程度うまく機能し、それを選ぶことで何を失うのだろうか。

直ちに立ちはだかる障害はスケールの問題である。コートの見た目を記述するとは、あり得るすべての画像に確率を割り当てることを意味する。たとえ各ピクセルを 1 ビット（塗られているか空白か）に単純化したとしても、あり得る 28×28 画像の総数は観測可能な宇宙に存在する原子の数をはるかに上回るのに対し、手元にある訓練画像は 60,000 枚にすぎない。**ナイーブベイズ**分類器は、たった 1 つの大胆な仮定によってこの障害を回避する。すなわち、クラスがひとたび判明すれば、各特徴量は互いについて何の情報ももたらさないと仮定するのである。

この着想は、含まれる単語によって文書を分類した Maron の初期の研究にさかのぼり (Maron 1961)、1990 年代のスパムフィルタを支えた (Sahami et al. 1998)。独立性の仮定のもとでは、スパムフィルタは各単語がスパムメールと通常メールでそれぞれどの程度の頻度で現れるかさえ知っていればよく、衣類分類器は各クラスで各ピクセルがどの程度の頻度で塗られているかさえ知っていればよい。訓練は数え上げに帰着し、予測は乗算に帰着する。

ナイーブベイズは、各クラスがどのように入力を生成するかというモデルから構築される最も単純な**生成**（generative）分類器である。オプティマイザを使わずにデータを 1 回走査するだけで当てはめが完了し、欠損した特徴量も対応する因子を取り除くだけで簡潔に扱える。その独立性の仮定は、現実世界のほとんどのデータと同様に画像に対しても明らかに成り立たないが、誤った独立性の仮定が分類器の判定や確率にどのような影響を与えるかをこれほど明快に示してくれるモデルはほかにない。

本節は本章のケーススタディという位置づけにある。36.10 節の汎化限界（generalization bound）の後ではここでの数学は軽めであるが、本章で学んだ道具立てを 1 つの具体的なモデルの上に結集させることに本節の意義がある。最尤法（maximum likelihood）でパラメータを当てはめ、ベータ事後分布でそれらを平滑化し、ブートストラップ法で正解率にエラーバーを付け、信頼性チェックでその確率を検証する。

以下の実装では通常の NumPy のみを用い、深層学習フレームワークからはデータローダだけを利用する。

```
d2l.use_svg_display()
```

36.11.1 分類のためのベイズの定理

入力が与えられたときのラベルの確率に至るには 2 つのルートがある。本書のこれまでの章では、その確率を直接学習した。本節では、入力のモデルを経由する間接的なルートをたどり、一般のままではなぜ計算困難になるのかを確認したうえで、この分類器の名前の由来である仮定によってそれを実用可能にする。

36.11.1.1 生成ルート

分類器はラベル $y \in \{1, \dots, K\}$ を入力 $\mathbf{x}$ に割り当てる。もし真の事後分布 $p(y \mid \mathbf{x})$（入力が与えられたときの各ラベルの確率）が既知であれば、最も高い頻度で正解する決定規則は、単に最も確率の高いラベルを選ぶものとなる。

$$\hat{y} = \operatorname*{argmax}_{y} \, p(y \mid \mathbf{x}). \tag{36.133}$$

Fashion-MNIST では $K = 10$ である。784 個の各ピクセルを塗られているか空白かで記録すると、$\mathbf{x}$ は 2^{784} 通りの二値パターンのいずれかになる。$784 \log_{10} 2 = 236.0$ であるから、これは約 10^{236} 通りの異なる画像に相当する。$p(y \mid \mathbf{x})$ のルックアップテーブルを作ろうとすればすべてのパターンに対して 1 行が必要になるが、60,000 枚の訓練画像ではそのほとんどすべてが空欄のまま残ってしまう。

ベイズの定理（36.2 節）により、事後分布は $p(y \mid \mathbf{x}) = p(\mathbf{x} \mid y)\, p(y) / p(\mathbf{x})$ と書き換えられる。分母の $p(\mathbf{x})$ は y に依存しないため、どのクラスが最大値を達成するかは変わらない。

$$\hat{y} = \operatorname*{argmax}_{y} \, p(\mathbf{x} \mid y)\, p(y). \tag{36.134}$$

各クラスについて、分類器はそのクラスのもとで画像がどれほど生じやすいかに、そのクラス自体が全体でどれほど一般的かを掛け合わせる。この積 $p(\mathbf{x} \mid y)\, p(y)$ を**ベイズ分子**（Bayes numerator）と呼ぶ。K 個の分子をそれらの総和で割れば $p(\mathbf{x})$ が復元され、完全な事後分布が得られる。

これにより、分類は大きく 2 つの系統に分かれる。**生成分類器**は、クラス条件付き分布（class-conditional distribution）$p(\mathbf{x} \mid y)$ と事前分布（prior）$p(y)$ を組み合わせて各クラスの入力がどのように生じるかをモデル化し、ベイズの定理を適用する。一方、**識別分類器**（discriminative classifier）は $p(y \mid \mathbf{x})$ を直接モデル化し、入力そのものがどのように生成されるかは一切記述しない。ソフトマックス回帰（3.1 節）はその典型例である。

ベイズの定理を使ってもスケールの問題は何も解決していないのではないか、すなわち $p(\mathbf{x} \mid y)$ は依然として 2^{784} 通りのパターン上の分布ではないか、と疑問に思うのも無理はない。その指摘は正しく、確率の連鎖律を使えば組合せ爆発がどこに潜んでいるかを正確に特定できる。特徴量を 1 つずつ展開すると、次のようになる。

$$p(\mathbf{x} \mid y) = p(x_1 \mid y)\, p(x_2 \mid x_1, y) \cdots p(x_d \mid x_1, \dots, x_{d-1}, y). \tag{36.135}$$

各因子はそれより前の特徴量の全履歴に条件付けられているため、最後の因子だけでも 2^{d-1} 通りの履歴にわたるテーブルが必要になる。次項では、これらの履歴をすべて切り捨てる。

36.11.1.2 ナイーブな仮定

現実的な規模のデータセットから $p(\mathbf{x} \mid y)$ を推定するには、パラメータ数を大幅に減らす必要がある。ナイーブな仮定を平易な言葉で言えば、衣類のクラスがひとたびわかれば、ある 1 つのピクセルが塗られていると知っても他のどのピクセルについてもそれ以上の情報は得られない、ということである。記号で表せば、各特徴量は**ラベルが与えられたもとで条件付き独立**（conditionally independent given the label）であり、連鎖律の各因子は自身の履歴を忘れる。

$$p(\mathbf{x} \mid y) = \prod_{i=1}^{d} p(x_i \mid y). \tag{36.136}$$

二値ピクセルの場合、各因子はたった 1 つの数値、すなわちピクセル i がクラス y において塗られている確率によって決まる。制約のない $p(\mathbf{x} \mid y)$ はクラスごとに $2^{784} - 1$ 個の自由な確率を必要とするのに対し、因数分解されたモデルが必要とするのは 784 個だけである。これら 784 個の数値はそれぞれ通常の頻度であり、クラスあたり 6,000 枚の訓練画像があれば高々 $0.5/\sqrt{6000} \approx 0.0065$ の標準誤差で推定できる（36.5 節）。また、どちらのモデルもクラス事前分布のために $K - 1 = 9$ 個の数値を必要とする。

なお、この仮定は d 個すべての特徴量に対する同時条件であり、特徴量のペアだけを調べても検証するには不十分である。ある 1 つのクラス内で、x_1 と x_2 が独立な公平ビットであり、$x_3 = x_1 \oplus x_2$ はそのうちちょうど一方が 1 のときに 1 をとるとしよう。特徴量のどのペアも独立性を満たすが、3 番目の特徴量は他の 2 つによって完全に決定される。その結果、パターン $(1, 1, 1)$ の真の確率は 0 であるにもかかわらず、3 つの周辺分布の積はこれに $\frac{1}{8}$ を割り当ててしまう（演習問題 1）。

各変数からそれが直接影響を与える変数へと矢印を引いた有向グラフとして描くと、この因数分解は星型の構造をなす。すなわち、ラベル y がすべての特徴量の親となり、特徴量同士を結ぶ辺は存在しない（Figure 36.50 左）。特徴量間の条件付き依存関係を保持するより表現力の高いモデルでは、特徴量間に辺が追加される（右）。

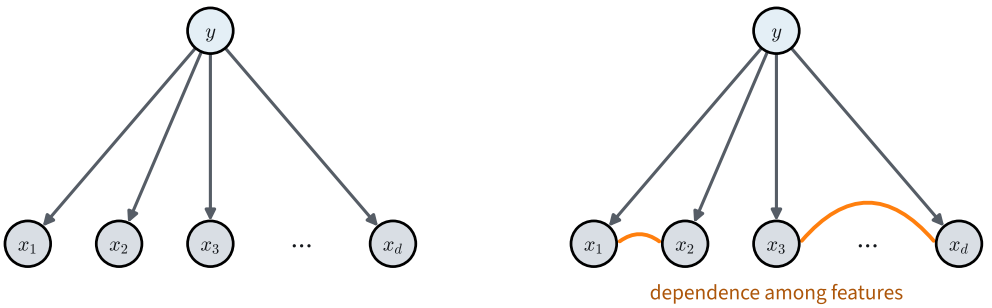


Figure 36.50: 条件付き独立の構造。左：ナイーブベイズでは、ラベル y がすべての特徴量の親であり特徴量間に辺はなく、$p(\mathbf{x} \mid y) = \prod_i p(x_i \mid y)$ を表す。右：特徴量間の条件付き依存性を保持するモデルでは、特徴量間に辺が追加される。

因数分解 Equation 36.136 をベイズの定理に代入すると、**ナイーブベイズ則**（naive Bayes rule）が得られる (Maron 1961)。

$$\widehat{y} = \operatorname*{argmax}_{y} \; p(y) \prod_{i=1}^{d} p(x_i \mid y). \tag{36.137}$$

画像に対して、この仮定が誤っていることは明らかである。衣類の隣接するピクセルは同じ袖や同じ裾に属するため、ともに塗られているか、ともに空白であるかのどちらかになりやすい。以下では、Fashion-MNIST 上でこの仮定がどれほど大きく破綻しているか、そしてその破綻が分類器に何をもたらすかを定量的に確かめる。

36.11.1.3 2 単語のスパムフィルタ

784 ピクセルへスケールアップする前に、最小の例を手計算で追ってみよう。メッセージに単語「free」が含まれるかどうかと、「meeting」が含まれるかどうかという 2 つの二値特徴量だけを調べるメールフィルタを考える。訓練セットに 10 通のメッセージ（スパム 4 通、通常メール 6 通）があり、出現数が次のとおりであったとする。

	スパム（4 通）	通常メール（6 通）
「free」を含む	3	1
「meeting」を含む	0	4

単純な数え上げだけで、モデルが必要とするすべての数値が揃う。事前分布は $\widehat{p}(\text{spam}) = 4/10 = 0.4$ である。単語「free」はスパムの $3/4$ と通常メールの $1/6$ に現れ、「meeting」はスパムの $0/4$ と通常メールの $4/6$ に現れる。ここで、新しいメッセージが「free」を含み、「meeting」を含まないとする。その 2 つのベイズ分子は次のようになる。

$$\text{spam: } 0.4 \cdot \tfrac{3}{4} \cdot (1 - \tfrac{0}{4}) = 0.30, \qquad \text{ordinary: } 0.6 \cdot \tfrac{1}{6} \cdot (1 - \tfrac{4}{6}) = \tfrac{1}{30} \approx 0.033,$$

これより、スパムの事後確率（事後分布）は $0.30/(0.30 + 0.033) = 0.90$ となる。同じ計算はオッズで表すとさらに見通しがよい。観測された各特徴量は、事前オッズにその尤度比（likelihood ratio）、すなわち観測された事象がスパムのもとで生じる確率を通常メールのもとで生じる確率で割った値を掛ける。

$$\begin{aligned} \frac{p(\text{spam} \mid \mathbf{x})}{p(\text{ordinary} \mid \mathbf{x})} &= \underbrace{\frac{0.4}{0.6}}_{\text{prior}} \cdot \underbrace{\frac{3/4}{1/6}}_{\text{"free"}} \cdot \underbrace{\frac{1}{2/6}}_{\text{no "meeting"}} \\ &= \frac{2}{3} \cdot 4.5 \cdot 3 = 9. \end{aligned}$$

9 対 1 のオッズは確率 $9/10$ に対応する。以下で構築する 784 ピクセルの分類器も、尤度比が 2 個ではなく 784 個になるだけで、これとまったく同じ計算を実行する。この小さなフィルタには、カウントが 0 になるとき、対数へ移行するとき、そして特徴量が二重に数えられるときの計 3 回、改めて立ち返る。

36.11.2 数え上げによる学習

ナイーブベイズ則には、クラス事前分布と、クラスおよび特徴量の組ごとに 1 つの確率が必要である。これらの推定値はどこから得られるのか、またコンピュータは数値的なアンダーフローで値を失うことなく 785 個の確率の積をどのように計算できるのだろうか。

36.11.2.1 最尤法と平滑化

先ほどのスパムフィルタでは、観測された頻度をそのまま代入した。最尤法はこの選択を正当化すると同時に、それがどこで破綻するかも明らかにする。$\theta_{iy} = p(x_i = 1 \mid y)$ を、特徴量 i がクラス y において活性化する確率とする。クラス y が n_y 回、全 n 個の訓練例の中に現れ、特徴量 i がそのうち n_{iy} 個の例で活性化しているとする。ナイーブベイズの因数分解のもとでは、訓練セット全体の対数尤度はクラス事前分布に関する 1 つの項と、クラスおよび特徴量の組ごとに独立な項へと分解される。

$$\sum_y n_y \log p(y) + \sum_y \sum_{i=1}^{d} \Big[n_{iy} \log \theta_{iy} + (n_y - n_{iy}) \log(1 - \theta_{iy}) \Big]. \tag{36.138}$$

角括弧内の各項は、n_{iy} 回の表を、n_y 回投げたバイアス θ_{iy} のコインから得る対数尤度にほかならないため、すべてのパラメータを個別に当てはめることができる。コインのバイアスの最尤推定量は観測頻度であるから（36.6 節）、$\hat{\theta}_{iy} = n_{iy}/n_y$ および $\hat{p}(y) = n_y/n$ が得られる。

しかし、生の頻度には深刻な落とし穴がある。特徴量 i がクラス y の訓練例に一度も現れなければ $\hat{\theta}_{iy} = 0$ となり、将来特徴量 i が活性化した例が与えられると、他のすべての特徴量が何を示していようとベイズ分子として 0 がクラス y に割り当てられる。たった 1 つの特徴量がそのクラスを棄却してしまうのである。2 単語のスパムフィルタにもすでにそのようなゼロが含まれている。「meeting」は訓練用のスパムメール 4 通に一度も現れなかったため、「free」と「meeting」の両方を含むメッセージのスパム分子は $0.4 \cdot \frac{3}{4} \cdot 0 = 0$ となり、100% の確信をもって通常メールと判定されてしまう。Fashion-MNIST のピクセルがあるクラスで一度も塗られないことがどの程度の頻度で起きるかは、後ほど確認する。

標準的な対処法は**ラプラス平滑化**（Laplace smoothing）である (Laplace 1814)。その特徴量が活性化した仮想的な例を 1 つ、非活性な仮想的な例を 1 つ加えることで、次の推定値を得る。

$$\hat{\theta}_{iy} = \frac{n_{iy} + 1}{n_y + 2}. \tag{36.139}$$

これにより、すべての推定値は $\frac{1}{2}$ の方向へわずかに引き寄せられ、推定確率が 0 や 1 になることは決してなくなる。

ラプラス平滑化には明快なベイズ的解釈もある。θ に一様な事前分布をおいたもとで、n_{iy} 回の成功が n_y 回の試行において観測されると事後分布 $\mathrm{Beta}(n_{iy} + 1,\, n_y - n_{iy} + 1)$ が得られ、その平均はちょうど $(n_{iy} + 1)/(n_y + 2)$ となる（36.4 節）。Laplace はこれを継起の法則（rule of succession）と呼んだ。

2 単語のスパムフィルタに戻ると、平滑化によって「free」と「meeting」の推定確率はスパムでは $4/6$ と $1/6$ に、通常メールでは $2/8$ と $5/8$ に変わる一方、事前分布は 0.4 のままである。両方の単語を含むメッセージのベイズ分子は $0.4 \cdot \frac{4}{6} \cdot \frac{1}{6} = 0.044$ と $0.6 \cdot \frac{2}{8} \cdot \frac{5}{8} = 0.094$ になり、スパム確率は 0.32 となって平滑化なしの 0 にはならない。その一方で、最初のメッセージ（「free」を含み「meeting」を含まない）の確率は 0.90 から 0.80 へと下がる。これは、スパムが会議（meeting）に決して言及しないという証拠としてはスパム 4 通では控えめにすぎないという事実を反映している。

36.11.2.2 対数空間におけるスコア

平滑化を施しても、ナイーブベイズ則をそのまま評価すると $d+1$ 個の数値（いずれも 0 から 1 の間）を掛け合わせることになり、画像ではその積は極端に小さくなる。仮にすべての因子が $\frac{1}{2}$ であったとすると、784 個の積は $2^{-784} \approx 10^{-236}$ となる。正の最小の単精度浮動小数点数は約 $1.4 \cdot 10^{-45}$、正の最小の倍精度浮動小数点数は約 $4.9 \cdot 10^{-324}$ であるため、単精度では積が直ちに 0 へとアンダーフローする。倍精度であれば 2^{-784} には耐えられるが、以下で計算する実際の画像のスコアには 10^{-1000} 付近まで落ち込むものもあり、倍精度でも持ちこたえられない。

対数をとれば、積は和に変わる。log は狭義単調増加であるため、最大となるクラスは変化しない（この手法一般については 35.8 節で論じている）。ベイズ分子の対数として**クラススコア**（class score）$s_y(\mathbf{x})$ を定義すると、決定規則は次のようになる。

$$\hat{y} = \operatorname*{argmax}_y \; s_y(\mathbf{x}), \qquad s_y(\mathbf{x}) = \log p(y) + \sum_{i=1}^{d} \log p(x_i \mid y). \tag{36.140}$$

785 個の確率の積が 785 個の対数の和に置き換わり、どの確率も 0 でない限り（これは平滑化によって保証される）すべての項は有限の値に留まる。平滑化なしのスパムフィルタでは、最初のメッセージの対数スコアは $\log 0.30 = -1.20$ と $\log \frac{1}{30} = -3.40$ であり、その差 $2.20 = \log 9$ は先ほど手計算で求めた事後オッズの対数にほかならない。

36.11.2.3 線形決定境界

二値特徴量の場合、クラススコアはきわめて見慣れた形に簡約され、その形からナイーブベイズがどのような分類器を表現できるかが明らかになる。二値特徴量は活性化しているときに $\log \theta_{iy}$、非活性のときに $\log(1-\theta_{iy})$ だけ寄与するため、$\log p(x_i \mid y) = x_i \log \theta_{iy} + (1-x_i)\log(1-\theta_{iy})$ と書ける。x_i が掛かる項と掛からない項をまとめると、次の式が得られる。

$$s_y(\mathbf{x}) = \mathbf{w}_y^{\top}\mathbf{x} + b_y, \qquad w_{iy} = \log \frac{\theta_{iy}}{1-\theta_{iy}}, \qquad b_y = \log p(y) + \sum_{i=1}^{d} \log(1-\theta_{iy}). \tag{36.141}$$

塗られた各ピクセルは w_{iy}、すなわちピクセル i がクラス y で塗られる対数オッズ（log-odds）を加算し、オフセット b_y は完全に空白の画像のスコアを表す。どのクラススコアも $\mathbf{x}$ のアフィン関数であるため、任意の 2 クラス間でスコアが等しくなる決定境界（decision boundary）は超平面となる (Bishop 2006)。これが本節で覚えておくべき 2 つの結果のうちの 1 つ目である。すなわち、ベルヌーイナイーブベイズはソフトマックス回帰とまったく同じモデル族に属する線形分類器であり、パラメータ配列の形状も完全に一致する（784×10 個の重みと 10 個のオフセット）。両手法の違いは、頻度を数え上げるか $p(y \mid \mathbf{x})$ に対して勾配降下法を実行するかという、それらの数値の決め方だけにある。2 クラスの場合、事後分布はスコア差のシグモイド $p(y=1 \mid \mathbf{x}) = \operatorname{sigmoid}\big((\mathbf{w}_1 - \mathbf{w}_0)^{\top}\mathbf{x} + b_1 - b_0\big)$ に帰着し、これはロジスティック回帰と同一の形式である（演習問題 2）。

境界が線形のまま保たれるかどうかは特徴量の分布に依存する。ガウスナイーブベイズは、各連続特徴量 x_i をクラス y において平均 μ_{iy}、分散 σ_{iy}^2 のガウス分布としてモデル化し、クラススコアに $-(x_i-\mu_{iy})^2/(2\sigma_{iy}^2)$ だけ寄与させる。すべての特徴量が全クラスで同じ分散を共有していれば、x_i^2 の項はどのスコアでも同一となってあらゆる 2 クラス比較で相殺され、線形な境界が残る。クラス固有の分散を持つ場合には x_i^2 の項が生き残り、境界は各 x_i について二次になるが、交差項 $x_i x_j$ は依然として現れない（演習問題 3）。

ガウス分布の場合を見ると、当てはめ手法の違いがどれほど重要かを視覚的に理解できる。Figure 36.51 では、各クラス内で 2 つの特徴量が強く相関しており、両方の点群が斜めに傾いている。ナイーブベイズは各クラスに座標軸に沿ったガウス分布を当てはめる（両クラスで分散が等しいため、境界は直線になる）。座標軸に沿った楕円はこの傾きを無視するため、得られる境界は両方の点群を横切ってしまい、19% の点を誤分類する。右のパネルは、十分なデータを用いて $p(y \mid \mathbf{x})$ を直接当てはめることで得られた線形境界を示している。この境界は傾きに沿って配置され、7% の誤り率を達成する。これはこの問題における最適値であり、いかなる種類の分類器もこれを上回ることはできない。

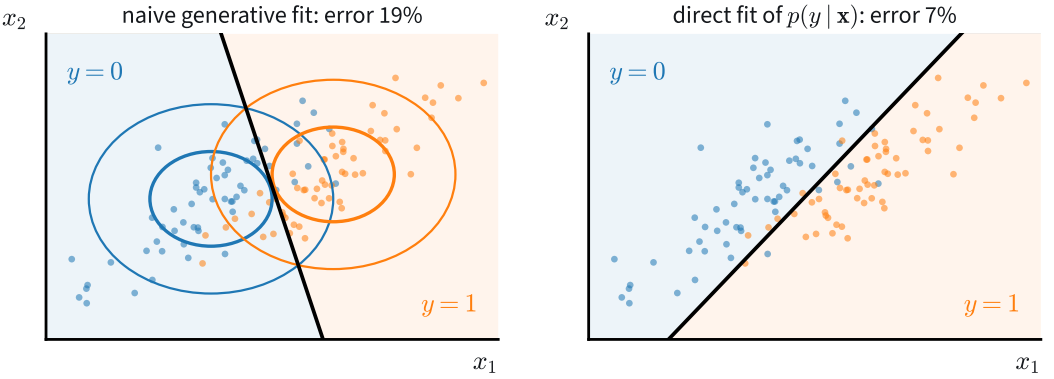


Figure 36.51: 各クラス内で 2 つの特徴量が相関している同一データ上で線形分類器に至る 2 つのルート。左：ナイーブな生成モデルの当てはめは、各クラスを独立な特徴量を持つガウス分布（標準偏差の 1 倍および 2 倍の楕円）としてモデル化し、ベイズの定理によって境界を導出する。右：$p(y \mid \mathbf{x})$ に直接当てはめた線形境界。誤り率は等しい事前分布に対する母集団での値である。

歴史的には、ナイーブベイズの代表的な応用先はピクセルではなくテキストであった。実際のスパムフィルタは、先ほどの 2 単語の例を数万語へと拡大したものにすぎない (Manning et al. 2008)。文書はどの単語が出現するかによっても、各単語が何回出現するかによっても表現でき、クラススコアはいずれの場合も線形になる。本節では引き続きピクセルを扱うが、演習問題 4 ではこれら 2 つのテキスト事象モデルを導出し、同一の文書に対して両者の判定が食い違い得ることを示す。

36.11.3 数え上げによる Fashion-MNIST 分類

2 単語のスパムフィルタであれば手計算で容易に確かめられる。しかし、目に見えて相関している 784 個のピクセルに対しても同じ単純な数え上げが通用するかどうかは実際に試してみるほかなく、Fashion-MNIST を使えばまったく同じ画像上でソフトマックス回帰と結果を直接比較できる。

36.11.3.1 二値ピクセル

Fashion-MNIST (Xiao et al. 2017) は 10 種類の衣類クラスにわたる 60,000 枚の訓練画像と 10,000 枚のテスト画像からなり、各画像は 28×28 のグレースケールグリッドである（3.2 節）。背景ピクセルは厳密に 0 であるため、$x_i = 1$ をピクセル i が正（衣類の一部）である場所に設定し、背景上では $x_i = 0$ とおくことで、各画像を二値ベクトルに変換できる。ローダから両分割の生画像を取得し、784 個の二値特徴量からなるベクトルへと平坦化する。

```
data = d2l.FashionMNIST()
def binarize(images):  # 1 where the pixel belongs to the garment
    return (images.reshape(-1, 784) > 0).astype(np.float32)

(X, Y), (X_test, Y_test) = data.train, data.val
X, X_test = binarize(X), binarize(X_test)
print(X.shape, X_test.shape, f'ink fraction {X.mean():.2f}')
```

```
(60000, 784) (10000, 784) ink fraction 0.50
```

全ピクセルのおよそ半分が塗られている。どのクラスも 6,000 枚の訓練画像と 1,000 枚のテスト画像を持つため、推定される事前分布は一様な 0.1 となり、すべてのクラススコアに同じ定数を加えるだけであるが、実装の一般性を保つためコードには事前項を残しておく。

36.11.3.2 テンプレートとサンプル

訓練に必要なのは、各クラスの画像枚数 n_y と、そのうち各ピクセルが塗られている画像の枚数 n_{iy} という 2 種類のカウントだけである。以下のセルでは、ラプラス平滑化を施した推定値 $\hat{\theta}_{iy}$ をクラスごとに 1 行として `P_xy` に格納する。同時に、平滑化に関して先ほど提起した実務上の疑問、すなわちあるクラスの訓練画像で一度も塗られないピクセルがいくつあり、そのうちのいずれかが塗られているテスト画像が何枚あるかにも答える。

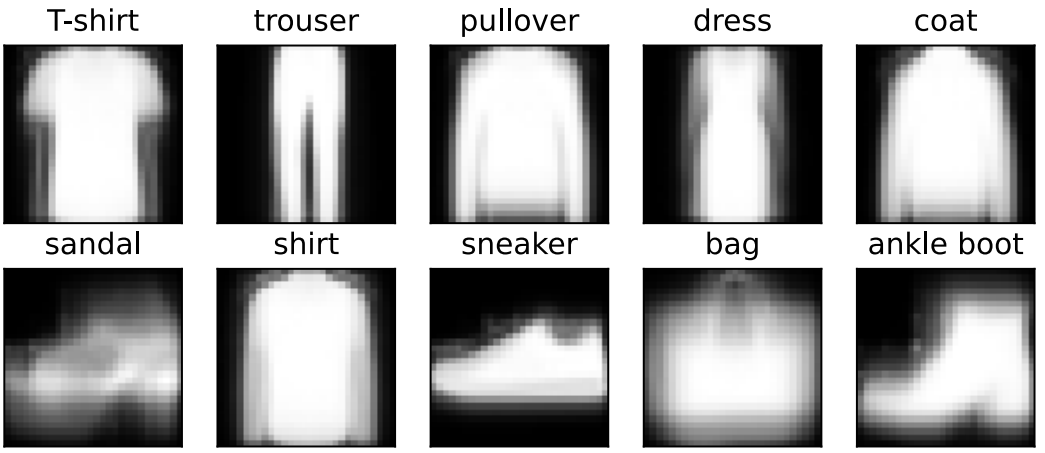
```
n_y = np.bincount(Y, minlength=10)              # images per class
n_x = np.stack([X[Y == y].sum(axis=0) for y in range(10)])
P_y = n_y / n_y.sum()                           # prior p(y)
P_xy = (n_x + 1) / (n_y + 2)[:, None]           # smoothed theta_iy
print('prior:', P_y)
print('never-inked pixels per class:', (n_x == 0).sum(axis=1))
unseen = (X_test * (n_x[Y_test] == 0)).sum(axis=1) > 0
print('test images inking one of their class:', unseen.sum())
```

```
prior: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
never-inked pixels per class: [ 0 13  1  0  3  3  0 57  0  1]
test images inking one of their class: 5
```

ここではゼロカウントはかなりまれである。10 クラス中 4 クラスでは 1 つもなく、ズボンでは 13 個、スニーカーでは 57 個である（スニーカーは平たい形状をしており、画像フレームの上部が空白のままになるためである）。テストセット全体でも、正解クラスが訓練中に一度も塗らなかったピクセルが塗られている画像は 5 枚しかない。しかし平滑化がなければ、これら 5 枚の画像はいずれも正解クラスに対するスコアが $-\infty$ となり、残りの 783 ピクセルがどれほど強くそのクラスを支持していても必ず誤分類されてしまう。

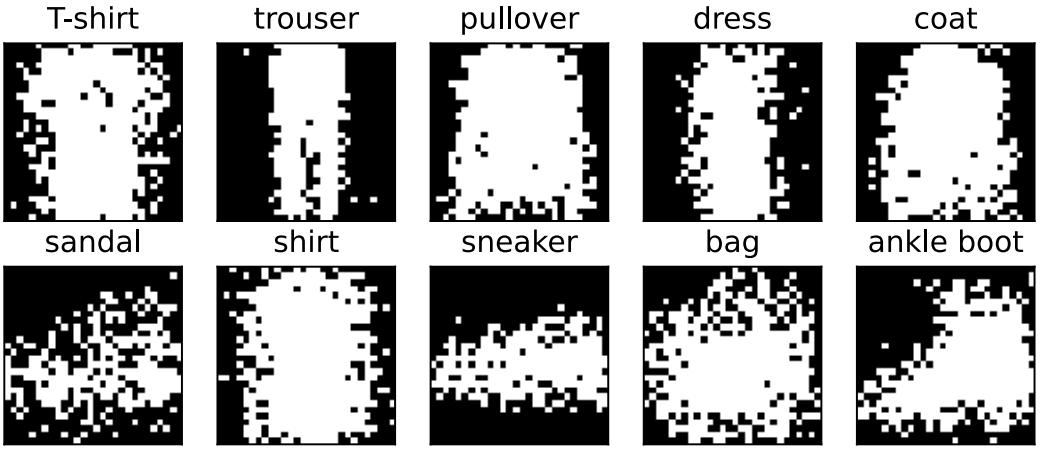
`P_xy` の各行は訓練画像のうち各ピクセルが塗られている割合を記録しているため、10 個の行を 28×28 の画像に形状変換して戻せば、モデルが各衣類クラスについて何を学習したかを視覚化できる。

```
d2l.show_images(P_xy.reshape(10, 28, 28), 2, 5,
                titles=data.text_labels(range(10)), cmap='gray');
```



得られるテンプレートは、10 種類の衣類の柔らかくぼやけた輪郭のように見える。各テンプレートは単にそのクラスのピクセルごとの平均であるため、このぼやけは衣類の形状や位置の実際のばらつきを反映している。独立性を仮定するかどうかによらず、データのピクセルごとの頻度に一致するモデルであればまったく同じテンプレートを生成するため、このぼやけは仮定ではなくデータ自体に由来するものである。ナイーブな仮定が付け加えるのは、これらの平均と事前分布さえあれば知るべきことはすべて尽きているという主張である。その主張が何を意味するかを確かめるため、すべてのピクセルについて確率 $\hat{\theta}_{iy}$ の独立なコインを投げることで、生成モデルから合成画像をサンプリングしてみよう。

```
rng = np.random.default_rng()
samples = (rng.random((1000, 10, 784)) < P_xy).astype(np.float32)
d2l.show_images(samples[0].reshape(10, 28, 28), 2, 5,
                titles=data.text_labels(range(10)), cmap='gray');
```



各サンプルは全体の大まかなシルエットこそ正しく捉えているものの、境界上のすべてのピクセルがそれぞれ独立なコイン投げで決まるため、縁がギザギザで斑点状に散らばっている。この視覚的な違いは直接定量化できる。実際の画像の 1 行を横に走査すると、塗られた部分（インク）と背景は数回しか切り替わらないが、モデルのサンプルでははるかに頻繁に切り替わるはずである。以下では、訓練セット全体とクラスあたり 1,000 枚の生成サンプルとで、全体のインク率と 1 行

あたりのインク・背景間の平均切り替わり回数の両方を比較する。

```
def switches(images):  # ink/background changes along a row, mean per row
    return np.abs(np.diff(images.reshape(-1, 28, 28), axis=2)).sum(axis=2).mean()

print(f'ink fraction: data {X.mean():.3f}, samples {samples.mean():.3f}')
print('switches per row, data:   ',
      np.round([switches(X[Y == y]) for y in range(10)], 1))
print('switches per row, samples:',
      np.round([switches(samples[:, y]) for y in range(10)], 1))
```

```
ink fraction: data 0.498, samples 0.498
switches per row, data:    [3.8 3.2 3.1 3.1 3.1 3.  3.3 1.5 3.1 2.6]
switches per row, samples: [5.2 4.1 4.8 4.7 4.8 8.2 5.4 4.  8.2 5.9]
```

すべてのピクセルが正しい周辺確率を持つ以上当然ではあるが、サンプルが持つ全体のインク量は実データと一致している。それにもかかわらず、インクと背景の間の切り替わりははるかに頻繁に起きており、プルオーバーでは 1 行あたり約 3 回ではなく約 5 回、サンダルやバッグでは約 3 回ではなく約 8 回に達する。実際の衣類が滑らかな輪郭を持つのは隣接するピクセル同士が同じ値をとりやすいためであり、この局所的な一致こそまさに独立性の仮定が切り捨ててしまうものにほかならない。

36.11.3.3 仮定の検証

生成されたサンプルは画像全体で独立性の仮定が破綻していることを示しているが、たった 1 組のピクセルペアを見るだけでも具体的な数値でそれを裏付けられる。プルオーバーの下裾に沿った 26 行目の 14 列目と 15 列目にある隣接する 2 つのピクセルを考えよう。もしこれら 2 つのピクセルがクラスが与えられたもとで条件付き独立であれば、両方が塗られているプルオーバーの割合はそれぞれのインク率の積に等しくなるはずである。訓練用のプルオーバーとサンプリングされた 1,000 枚のプルオーバーの両方について、その積と実際の同時頻度を比較してみよう。

```
i, j = 26 * 28 + 14, 26 * 28 + 15     # two neighbouring hem pixels
for name, Z in (('data', X[Y == 2]), ('model', samples[:, 2])):
    p_i, p_j, both = Z[:, i].mean(), Z[:, j].mean(), (Z[:, i] * Z[:, j]).mean()
    print(f'{name:5s}: P(i)={p_i:.2f}  P(j)={p_j:.2f}  '
          f'P(i)P(j)={p_i * p_j:.2f}  P(i,j)={both:.2f}')
```

```
data : P(i)=0.55  P(j)=0.55  P(i)P(j)=0.30  P(i,j)=0.54
model: P(i)=0.55  P(j)=0.52  P(i)P(j)=0.29  P(i,j)=0.29
```

どちらのピクセルも約 55% のプルオーバーで塗られている。独立性のもとでは、両方が同時に塗られる割合は $0.55 \times 0.55 \approx 30\%$ になるはずである。ところが実データでは、54% のプルオーバーで両方が塗られており、片方のピクセル単独とほぼ変わらない頻度になっている。プルオーバーの裾は 26 行目に届くかその手前で終わるかのどちらかであるため、これら 2 つの隣接ピクセルはほぼ 1 つの特徴量が 2 回記録されたかのように振る舞う。一方、モデルのサンプルは構成上その積に一致する。

36.11.3.4 テスト正解率

テストセットを分類するため、アフィン形式 Equation 36.141 を用いる。`P_xy` と `P_y` から重み w_{iy} とオフセット b_y を計算し、1 回の行列積ですべてのテスト画像に対して全 10 クラスのスコアを求め、続く分析のために予測結果を保存する。ソフトマックス回帰はこれとまったく同じ形

式のスコアで約 83% の正解率に達したが、代わりに数え上げを使うことでどれほど正解率を譲ることになるのだろうか。

```
W = np.log(P_xy) - np.log1p(-P_xy)              # w_iy, log-odds per pixel
b = np.log1p(-P_xy).sum(axis=1) + np.log(P_y)   # b_y, score of a blank image
s = X_test @ W.T + b                            # s_y(x), shape (10000, 10)
pred = s.argmax(axis=1)
print(f'test accuracy {(pred == Y_test).mean():.4f}, training accuracy '
      f'{((X @ W.T + b).argmax(axis=1) == Y).mean():.4f}')
```

```
test accuracy 0.7059, training accuracy 0.7171
```

ナイーブベイズは約 71% のテスト画像を正しく分類しており、ランダムな当て推量のベースラインである 10% を大きく上回る。数え上げには乱数性が一切入らないため、この数値は実行ごとに完全に決定論的である。サンプリングに伴う不確実性のエラーバーについてはすぐ後で付与する。

訓練正解率も同様に示唆に富んでいる。当てはめに用いた 60,000 枚の画像に対する正解率は 72% であり、未知のテスト画像に対する値よりわずか 1 パーセントポイント高いにすぎない。訓練セットに過学習したモデルであれば両者の間に大きな乖離が生じるが、ここではそれがほとんど見られない。したがって、誤りの原因は訓練データの不足ではない。モデルの仮定が制約的すぎるために、新しい画像に対するのと同じ誤りを自身の訓練セット上でも犯しているのである。36.5 節の言葉で言えば、誤差は分散ではなく統計的バイアス（bias）に支配されている。画像をさらに集めても状況はほとんど改善しない。必要なのはより良いパラメータの組であり、以下で行うソフトマックス回帰との比較によって、まったく同じモデル族の中にはるかに優れた線形分類器が存在することが確認できる。

また、これらの対数スコアを使えば、仮に生の積 Equation 36.137 をそのまま評価していたら何が起きていたかも確認できる。各スコアはベイズ分子の自然対数であるため、$\log 10$ で割れば底 10 の指数が得られる。各分子を、正の最小の単精度数、および倍精度が精度を失い始める境界である正の最小の倍精度**正規化数**と比較してみよう。

```
log10 = s / np.log(10)                    # decimal exponent of each numerator
print(f'log10 of the numerators: largest {log10.max():.0f}, smallest '
      f'{log10.min():.0f}, median best {np.median(log10.max(axis=1)):.0f}')
f32 = -149 * np.log(2)     # log of 2**-149, the smallest positive float32
f64 = -1022 * np.log(2)    # log of 2**-1022, the smallest normal float64
print(f'images whose best numerator underflows float32: '
      f'{(s.max(axis=1) < f32).mean():.2f}')
print(f'numerators below the smallest normal float64: {(s < f64).mean():.2f}, '
      f'images whose best one is: {(s.max(axis=1) < f64).mean():.4f}')
print(f'accuracy of the raw float64 product: '
      f'{(np.exp(s).argmax(axis=1) == Y_test).mean():.4f}')
```

```
log10 of the numerators: largest -32, smallest -1002, median best -81
images whose best numerator underflows float32: 0.96
numerators below the smallest normal float64: 0.44, images whose best one is: 0.0003
accuracy of the raw float64 product: 0.7058
```

ベイズ分子はおよそ 10^{-32} から 10^{-1000} までの範囲に広がっており、典型的な画像における最大クラスの分子は 10^{-81} 付近にある。テスト画像の 96% では 10 個の分子のうち最大のものでさえ正の最小の単精度浮動小数点数を下回るため、単精度のまま積を計算すると 10 個すべてがゼロになってまったく分類できなくなる。倍精度では全分子の 44% が最小の正規化数 float64 を下回るが、そのほとんどは敗北クラスに属するものである。最大の分子は 10,000 枚中 3 枚を除くすべ

ての画像で表現可能な範囲に留まるため、生の float64 の積による正解率は 10,000 枚中 1 枚以内の差で対数空間での正解率と一致する。とはいえ、倍精度が 784 ピクセルに耐えられる余裕はわずかにすぎない。典型的な最大スコアが 10^{-81} であることを踏まえると、入力長が 4 倍になれば float64 でもアンダーフローしてしまう。対数空間で計算すれば、あらゆるスコアが通常の浮動小数点の範囲内に無理なく収まる。

36.11.4 分類器の評価

71% という見出しの正解率を前にすると、直ちに 3 つの疑問が生じる。この推定値はどれほど正確なのか。残りの 29% はどのような取り違えによるものなのか。そして、分類器が報告する確率は信頼できるのだろうか。

36.11.4.1 エラーバー

ここでのテスト正解率は 10,000 枚のランダムなテスト画像にわたる平均であるため、別のテスト標本をとればわずかに異なる数値が得られるはずである。各テスト画像は正解か不正解かという 1 回のベルヌーイ試行の結果を与える。これは本章の冒頭で登場したコイン投げをテスト画像ごとに 1 回ずつ行うのと同じであるから、観測される正解率 $\hat{p}$ は n 回の試行の標本平均であり、標準誤差 $\sqrt{\hat{p}(1-\hat{p})/n}$ を持つ（36.5 節）。$\hat{p} = 0.706$、$n = 10{,}000$ のとき、次のようになる。

$$\sqrt{\frac{0.706 \cdot 0.294}{10^4}} \approx 0.0046, \qquad 1.96 \times 0.0046 \approx 0.009,$$

これより、上下いずれの方向にも約 0.9 パーセントポイント広がる 95% 信頼区間（confidence interval）が得られる。ブートストラップ法（36.5 節）を使えば、公式に頼らない第 2 の確認ができる。テスト画像を復元サンプリングし、各再標本で正解率を再計算して、得られる分布の中央 95% をとればよい。

```
rng = np.random.default_rng()
correct = pred == Y_test                          # per-image 0/1 outcomes
n, p = len(correct), correct.mean()
boot = correct[rng.integers(0, n, (1000, n))].mean(axis=1)  # 1,000 resamples
print(f'accuracy {p:.4f}, bootstrap 95% interval '
      f'({np.percentile(boot, 2.5):.3f}, {np.percentile(boot, 97.5):.3f})')
print(f'binomial half-width 1.96 se = {1.96 * np.sqrt(p * (1 - p) / n):.4f}')
```

```
accuracy 0.7059, bootstrap 95% interval (0.697, 0.714)
binomial half-width 1.96 se = 0.0089
```

ブートストラップ法による区間は約 0.70 から 0.715 までとなり、二項分布に基づく半幅 0.009 と一致する。したがって正解率は $71\% \pm 0.9$ ポイントと報告すべきであり、「70.59%」の小数点以下の桁はサンプリングノイズにすぎない。なお、まったく同じテストセット上で 2 つの分類器を比較する場合には両者の誤りに相関が生じるため、2 つの別々の信頼区間を比較するよりも、画像ごとの差に対する対応のある分析を行う方がはるかに検出力が高い（演習問題 5）。

36.11.4.2 混同と重み

全体の正解率は分類器がどの程度の頻度で誤るかを示すのに対し、3.4 節の混同行列（confusion matrix）は**どのように**誤るかを示す。3.3 節の慣例に従い、行を予測クラス、列を正解クラスに対応させると、各列はある衣類のテスト画像 1,000 枚がどのクラスに振り分けられたかを表す。

```
    conf = np.zeros((10, 10), dtype=int)
    np.add.at(conf, (pred, Y_test), 1)       # N[i, j]: class j predicted as i
    print(conf)
```

```
[[748   9   8  50   2   0 218   0   0   0]
 [  0 907   0  53   3   0   4   0   3   0]
 [ 14  18 665   5 372   0 232   0  14   0]
 [122  48  12 808 104   2  93   0  68   7]
 [  8  16 242  45 483   0 222   0   6   0]
 [  0   0   0   0   0 680   0   5   1   0]
 [ 83   0  50  35  25   2 167   0  20   0]
 [  1   0   0   0   0 232   1 893  55 113]
 [ 24   2  23   4  11   9  63   0 832   4]
 [  0   0   0   0   0  75   0 102   1 876]]
```

10×10 のカウント表は、最も多い誤りを抜き出すと格段に読み取りやすくなる。以下では、非対角成分のうち上位 6 個を「正解クラス→予測クラス」の形式で出力するとともに、各クラス個別の正解率（対角成分を列和で割った値）を表示する。

```
    names = data.text_labels(range(10))
    off = conf - np.diag(np.diag(conf))      # errors only
    order = np.argsort(off, axis=None, kind='stable')[::-1][:6]  # largest first
    pairs = [f'{names[t]}->{names[q]} {off[q, t]}'  # true -> predicted
             for q, t in zip(*np.unravel_index(order, off.shape))]
    print(', '.join(pairs[:3]))
    print(', '.join(pairs[3:]))
    print('per-class accuracy:', np.round(np.diag(conf) / conf.sum(axis=0), 2))
```

```
coat->pullover 372, pullover->coat 242, sandal->sneaker 232
shirt->pullover 232, shirt->coat 222, shirt->T-shirt 218
per-class accuracy: [0.75 0.91 0.66 0.81 0.48 0.68 0.17 0.89 0.83 0.88]
```

誤りは特定の組に強く集中している。コートがプルオーバーと誤分類されるのが 372 回、プルオーバーがコートとされるのが 242 回あり、シャツはプルオーバー、コート、T シャツのそれぞれと 200 回以上取り違えられ、サンダルはスニーカーと 232 回取り違えられている。ズボン（91%）とスニーカー（89%）は容易である一方、コート（48%）とシャツ（17%）ははるかに難しい。これらは 3.3 節でソフトマックス回帰が苦戦した 2 つの混同グループ、すなわち上半身の衣類 4 種と履物 3 種とまったく同じであるが、数え上げによるモデルは上半身グループで大幅に成績を落としている。ソフトマックス回帰がシャツを 40〜60% の割合で認識したのに対し、ナイーブベイズは約 6 分の 1 しか正しく認識できていない。

71% というのは、二値ピクセル上の線形分類器にとって単なる上限なのだろうか。ソフトマックス回帰はこれとまったく同じ形式の超平面を定めるが、$p(y \mid \mathbf{x})$ の交差エントロピーを最小化することで重みを決定する。当てはめ手順だけが異なる比較を行うため、同じ二値ピクセル上でミニバッチ確率的勾配降下法によりソフトマックス回帰を訓練してみよう。パラメータ更新には、ロジットに関する交差エントロピーの勾配が予測確率ベクトルからワンホットラベルを引いたものに等しいという 3.1 節の結果を用いる。

```
    rng = np.random.default_rng()
    W_sr, b_sr = np.zeros((10, 784)), np.zeros(10)
    for epoch in range(20):                  # minibatches of 256 images
```

```
    for idx in np.array_split(rng.permutation(len(X)), len(X) // 256):
        O = X[idx] @ W_sr.T + b_sr                    # logits
        E = np.exp(O - O.max(axis=1, keepdims=True))
        G = E / E.sum(axis=1, keepdims=True) - np.eye(10)[Y[idx]]
        W_sr -= 0.05 * G.T @ X[idx] / len(idx)        # mean-loss gradient
        b_sr -= 0.05 * G.mean(axis=0)
pred_sr = (X_test @ W_sr.T + b_sr).argmax(axis=1)
conf_sr = np.zeros((10, 10), dtype=int)
np.add.at(conf_sr, (pred_sr, Y_test), 1)
print(f'accuracy: softmax regression {(pred_sr == Y_test).mean():.3f}, '
      f'naive Bayes {(pred == Y_test).mean():.3f}')
print('pullover<->coat errors: softmax regression',
      conf_sr[2, 4] + conf_sr[4, 2], 'naive Bayes', conf[2, 4] + conf[4, 2])
```

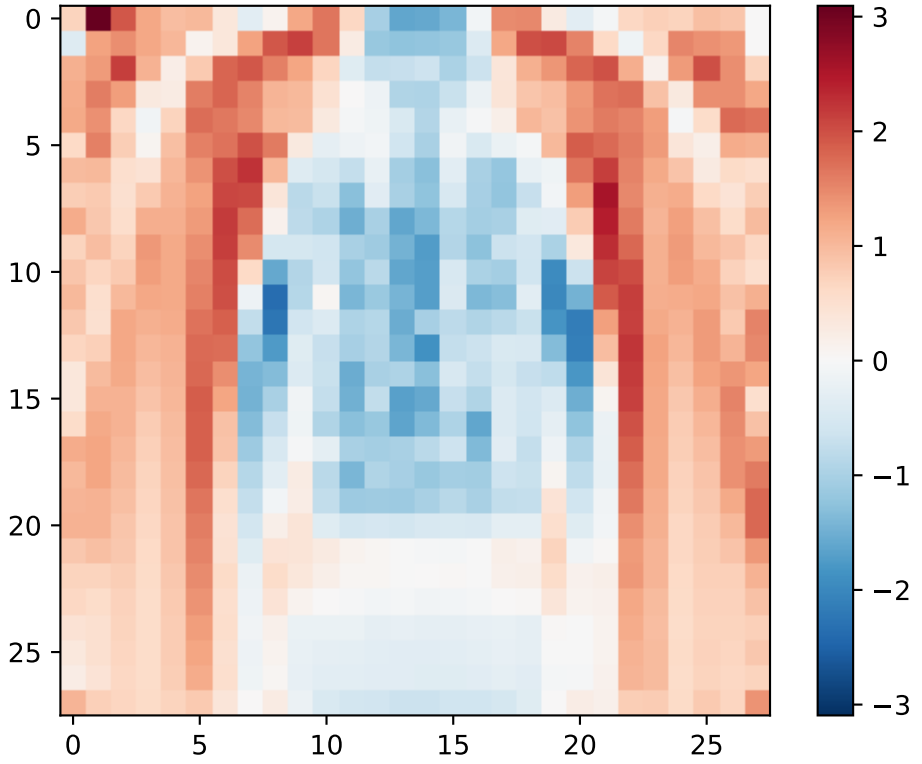
```
accuracy: softmax regression 0.828, naive Bayes 0.706
pullover<->coat errors: softmax regression 289 naive Bayes 614
```

ソフトマックス回帰はナイーブベイズを 12 パーセントポイント上回る約 83% の正解率に達し、プルオーバーとコートの間の誤りもおよそ半減させる。したがって、線形モデル族そのものがボトルネックなのではなく、この差はすべて数え上げによる重みの決め方に起因している。なぜ数え上げが重みを歪めるのかについては、次項で較正を調べるときに突き止める。

その前に、ナイーブベイズの重みを直接観察して、分類器が 2 つのクラスを分離するためにどの特徴量を頼りにしているかを見てみよう。プルオーバーとコートのスコア差は重みベクトル $\mathbf{w}_{\text{pullover}} - \mathbf{w}_{\text{coat}}$ を持つアフィン関数であり、ピクセル i が空白ではなく塗られているときには、このベクトルの第 i 成分だけスコアがプルオーバー側へシフトする。プルオーバー寄りを赤、コート寄りを青としてこの差ベクトルを 28×28 画像としてプロットし、絶対値が最大の成分とその背後にあるカウント数を出力しよう。

```
w = (W[2] - W[4]).reshape(28, 28)        # pullover minus coat, in nats
m = np.abs(w).max()
d2l.plt.imshow(w, cmap='RdBu_r', vmin=-m, vmax=m)
d2l.plt.colorbar()
k = np.abs(w).argmax()
print(f'largest |weight| {m:.1f} nats at row {k // 28}, column {k % 28}: '
      f'inked in {n_x[2, k]:.0f} pullovers and {n_x[4, k]:.0f} coats')
```

```
largest |weight| 3.1 nats at row 0, column 1: inked in 21 pullovers and 0 coats
```



プルオーバーの方がコートよりも頻繁に塗られる画像の両脇や上端の行に沿って、赤いピクセルが並んでいる。胴体部分は -1 付近の重みを持つ青いピクセルで埋め尽くされているが、これらの青いピクセルが塗られているときにコートの強い証拠になると解釈するのは誤りである。各重みは、空白のピクセルに**対する**塗られたピクセルの相対的なスコア差を測っていることを思い出そう。胴体のピクセルはプルオーバーの約 99%、コートの 99.5% 以上で塗られているため、塗られているのを観測してもスコアは約 -0.01 nat しか動かない。-1 という重みは実際には、胴体のピクセルがまれに空白のままだったときにモデルがプルオーバー側へ加算する寄与を表している。実質的な判定は両脇の部分で行われているのである。一方、単一で最大の重み（約 3 nat）はプルオーバー約 20 枚とコート 0 枚で塗られている隅のピクセルに位置しており、その大きさは豊富なデータではなく主にラプラスの擬似カウントによって決まっている。

生成モデルの当てはめをさらに批判する前に、識別モデルに対して持つ 1 つの本質的な利点に注目しておこう。遮蔽物の陰に隠れるなどしてテスト時に一部のピクセルが欠損している場合、ナイーブベイズは単に積からそれらの因子を取り除くだけでよい。$p(x_i \mid y)$ を未観測の x_i がとり得る両方の値にわたって周辺化（marginalization）すると和が 1 になるため、生成モデルのもとではこれは数学的に厳密である。ソフトマックス回帰にはこれに対応する周辺化の規則がなく、欠損ピクセルに何らかの値を補完（imputation）しなければならない（演習問題 9）。

36.11.4.3 較正

クラススコアの役割は 10 個のクラスを順位付けするだけではない。指数関数をとればベイズ分子が復元され、正規化すれば先ほど省いた分母 $p(\mathbf{x})$ が復元されるため、モデルの事後確率（事後分布）はスコアのソフトマックスそのものになる。

$$p(y \mid \mathbf{x}) = \frac{\exp s_y(\mathbf{x})}{\sum_{y'} \exp s_{y'}(\mathbf{x})}. \tag{36.142}$$

したがって、すべての予測には主張される確信度（confidence）、すなわちこれら事後確率の最大値が伴う。3.4 節と同様、分類器が主張する確信度が経験的な頻度と一致するとき、すなわち確信度 c でなされたすべての予測のうち実際に割合 c が正解であるとき、その分類器は**較正されている**という。較正を検証するため、テスト画像を確信度のビンにグループ分けし、各ビンにおける平均確信度と実際の正解率を比較する。

期待較正誤差（expected calibration error、ECE）は、この表を 1 つの数値に集約したものであり、主張される確信度と実際の正解率との差を、各ビンに入るテスト画像の割合で重み付けして全ビンにわたり平均した値である。以下ではまず最大スコアを引く数値的に安定な形式（35.8.2 節）でソフトマックスを計算し、あわせて第 1 位と第 2 位のクラススコアの差の中央値も出力する。

```
    post = np.exp(s - s.max(axis=1, keepdims=True))      # softmax, stably
    post /= post.sum(axis=1, keepdims=True)
    c = post.max(axis=1)                                 # claimed confidence
    print(f'mean confidence {c.mean():.3f}, accuracy {correct.mean():.3f}')
    edges, ece = [0, 0.9, 0.99, 0.999, 1], 0
    for lo, hi in zip(edges[:-1], edges[1:]):
        m = (c > lo) & (c <= hi)
        ece += m.mean() * abs(c[m].mean() - correct[m].mean())
        print(f'({lo}, {hi}]: {m.sum():5d} images, claimed {c[m].mean():.3f}, '
              f'achieved {correct[m].mean():.3f}')
    top2 = np.sort(s, axis=1)[:, -2:]
    print(f'ECE {ece:.2f}, median top-two score gap '
          f'{np.median(top2[:, 1] - top2[:, 0]):.0f} nats')
```

```
mean confidence 0.992, accuracy 0.706
(0, 0.9]:   257 images, claimed 0.726, achieved 0.350
(0.9, 0.99]:   297 images, claimed 0.961, achieved 0.384
(0.99, 0.999]:   265 images, claimed 0.996, achieved 0.419
(0.999, 1]:  9181 images, claimed 1.000, achieved 0.735
ECE 0.29, median top-two score gap 61 nats
```

結果は鮮烈である。モデルは平均して 99% の確信度を主張しながら、実際には予測の 71% しか当てられていない。全テスト画像の約 92% が最上位のビンに入り、そこではモデルが小数第 3 位まで確実だと主張しているにもかかわらず、実際の正解率は約 73% にとどまる。下位 3 つのビンでも、73% から 99.6% の確信度を主張しながら実際の正解率は 35〜42% にすぎない。すべてのビンが過信状態にあるため、期待較正誤差は平均確信度と正解率の差そのものに等しく、約 0.29 となる（3.4 節のソフトマックス回帰では 0.03 未満であった）。ビンが上がるにつれて正解率も上昇しているため確信度のランキング自体は依然として有用な情報を含んでいるが、報告される確率を額面どおりに受け取ることはできない。

原因はプルオーバーの裾で見た現象と同じであり、相関するピクセルが独立な証拠として数えられていることにある。この破綻の最も単純な形は単一の特徴量が複数回複製された場合であり、その計算は 3 行で済む。事前分布が π_0 と π_1 の 2 つのクラスと、特徴量 x（尤度比は $r(x) = p(x \mid y=1)/p(x \mid y=0)$）を考えよう。データセットに k 個の同一コピーが x について含まれているとする。k 個のコピーは常に一致するため 1 個のコピー以上の情報は持たず、真の対数事後オッズは $\log(\pi_1/\pi_0) + \log r(x)$ である。ところがナイーブベイズは、これらのコピーを条件付き独立として扱い、k 個の同一因子 $p(x \mid y)$ を掛け合わせて次の値を報告する。

$$\log \frac{\pi_1}{\pi_0} + k \log r(x). \tag{36.143}$$

重複した証拠は対数オッズを k 倍にする。事前分布が等しいとき、対数オッズに正の定数を掛けても符号は決して反転しないため、ナイーブベイズはどの k に対してもまったく同じクラスを予測するが、$\bar{r}^k$ という事後オッズを主張する。実際には $\bar{r}$（ただし $\bar{r} = \max(r, 1/r)$）こそが正しい値である。これが本節で覚えておくべき 2 つ目の結果である。すなわち、ランキングは生き残るが、確信度は崩壊する。

具体的な数値で見ると、この水増しがいかに激しいかがはっきりわかる。$p(x = 1 \mid y = 1) = 0.8$、$p(x = 1 \mid y = 0) = 0.2$ であり、活性化しているときに $r = 4$ となる、80% の割合で当たる手がかりを考えよう。コピーが 1 個なら正しい確信度 0.80 が得られる。しかしコピーを 2 個、3 個、4 個、5 個と増やすと、予測が実際に当たる割合は依然として 80% のままであるにもかかわらず、ナイーブベイズが主張する確信度は 0.94、0.985、0.996、0.999 へと跳ね上がる。2 単語のスパムフィルタで単語「free」を複製しても同じことが起き、最初のメッセージのオッズは 9 から $\frac{2}{3} \cdot 4.5^2 \cdot 3 = 40.5$ へ跳ね上がって、報告されるスパム確率は 0.976 まで上昇する。

ただし、ランキングの安全性を過大評価してはならない。上の議論は事前分布が等しいことに依存しており、この但し書きは重要である。事前オッズが $\pi_1/\pi_0 = 1/5$ で $r = 4$ のとき、コピーが 1 個なら事後オッズは $4/5$ となって正しくクラス 0 を予測するが、コピーが 2 個になると $16/5$ となって予測がクラス 1 へ反転してしまう。繰り返された証拠は、1 回の観測だけでは覆すべきでない事前分布を圧倒し得るのである。ランキングが保たれるのは、証拠がもともと事前分布と同じ向きにある場合か、1 個のコピーだけでも事前分布を上回る場合に限られる。

Fashion-MNIST の 784 個のピクセルは文字どおりのコピーではないが、広い領域にわたって強く相関している。先ほどの 2 つの裾ピクセルはほぼ 1 つの特徴量が 2 回数えられたかのように振る舞っており、そのような相関領域が何十個も積み重なることで、第 1 位と第 2 位のクラススコアの間に中央値で約 60 nat もの差が生じる。$e^{-60} \approx 10^{-26}$ であるから、ソフトマックスは 60 nat の差を報告確信度 1.000 へと変換する。極端な過信にもかかわらずナイーブベイズがクラスを有用に順位付けできることは、このモデルの古典的な性質として知られている (Domingos and Pazzani 1997)。それでもランキングが完全に無傷というわけではない。相関したパッチの大きさは一様ではないため、冗長なピクセルからなる広い領域が、独立な情報を持つ小さな領域を容易に多数決で押し切ってしまう。隣接ピクセルと同じ情報を繰り返すだけのピクセルの重みを下げられるソフトマックス回帰に対して 12 パーセントポイント遅れをとった理由は、この不均衡で無理なく説明できる（ただし、ここでの実験だけではそのメカニズムを切り分けられない）。

過信（overconfidence）には驚くほど単純な事後的修正法があり、ここではその名前だけ紹介しておく。温度スケーリング（temperature scaling、3.4 節）は、ホールドアウト検証データ上で当てはめた単一のスカラー（ソフトマックス温度）T ですべてのクラススコアを割る手法であり、クラスのランキングを一切変えない。同一コピーが k 個あり事前分布が等しい場合には、$T = k$ とおけば厳密な事後分布が復元される。演習問題 8 では、数え上げモデルに対して T を当てはめる。また、過信はナイーブベイズ特有の問題ではない。深層ニューラルネットワークも、通常はこれより軽度でありアーキテクチャや訓練レシピに依存するものの、同様の統計的バイアスに陥り得る (Guo et al. 2017)。そのため、出力確率に基づいて意思決定を行うあらゆる分類器において、信頼性チェックを行うことは良い実践である。

自然な疑問が 1 つ残っている。クラスあたり 6,000 枚の画像がある設定では、識別モデルの当てはめが圧勝した。しかし訓練例がきわめて少ない場合には、優劣のランキングがしばしば逆転する。Ng and Jordan (2002) は、ナイーブベイズのパラメータが次元 d に対して対数オーダーの訓練例数で収束するのに対し、ロジスティック回帰は d に対して線形オーダーの訓練例数を要し得ることを証明した。演習問題 10 では、プルオーバーとコートを用いてこの逆転現象を探る。

36.11.5 まとめ

ベイズの定理は各クラスがどのように入力を生成するかを表す生成モデルを分類器へと変換するが、制約のない $p(\mathbf{x} \mid y)$ のテーブルを 10^{236} 通りの二値画像にわたって推定することは不可能である。クラスが与えられたもとで特徴量間の条件付き独立を仮定すると、モデルはラプラスの擬似カウント 1 個で平滑化された、クラスおよび特徴量の組ごとの単一の頻度へと縮約されるため、訓練は数え上げに帰着する。ナイーブベイズ分子の対数は二値特徴量に関してアフィンであるため、ベルヌーイナイーブベイズはソフトマックス回帰とまったく同じモデル族に属する線形分類器となる。

Fashion-MNIST 上では、この数え上げによる分類器の正解率は約 $71\% \pm 0.9$ パーセントポイントに達するが、同じ二値ピクセルに直接当てはめたソフトマックス回帰を 12 ポイント下回る。また、相関するピクセルが独立な証拠として扱われ、重複した証拠が対数オッズを何倍にも増幅するため、平均確信度として 99% を報告する。この極端な過信は、実務で分類器の確率を利用する際には必ず較正を検証すべきであることを示している。

36.11.6 演習問題

1. **XOR とナイーブな仮定。** 4 つの点 $(0,0)$、$(0,1)$、$(1,0)$、$(1,1)$ にラベル $y = x_1 \oplus x_2$、すなわちそれぞれ 0、1、1、0 が付いているとする。訓練セットには各点が 1 回ずつ現れる。
   1. $p(y)$ と $\theta_{iy} = p(x_i = 1 \mid y)$ の最尤推定量（最尤推定値）、および 4 点すべてのクラススコアを計算せよ。ナイーブベイズ則 Equation 36.137 はどのような予測を下すか。
   2. スコアのアフィン形式 Equation 36.141 を用いて、ベルヌーイナイーブベイズのパラメータをどのように選んでも 4 点すべてを正しく分類できないことを示せ。退化したパラメータ $\theta_{iy} \in \{0, 1\}$ を許しても解決しないのはなぜか。（各クラスは自身に属する 2 点に正の確率を与えなければならない。）
   3. 2 つの因子 $p(x_1 \mid y)\, p(x_2 \mid y)$ を同時確率表 $p(x_1, x_2 \mid y)$ に置き換えよ。ベイジアンネットワーク (Koller and Friedman 2009) の言葉では、これは Figure 36.50 の右パネルに示されている x_1 と x_2 の間の辺を追加することにあたる。4 つの点からこのモデルを推定し、4 点を正しく分類できることを示せ。そのクラススコアがもはや $\mathbf{x}$ に関してアフィンでないのはなぜか。
   4. ある 1 つのクラス内で、x_1 と x_2 を独立な公平ビットとし、$x_3 = x_1 \oplus x_2$ とおく。3 つの特徴量のどのペアも独立であることを示せ。8 通りの各パターンの確率を 3 つの周辺分布の積と比較せよ。クラス内のすべてのペアごとの相関を調べても仮定 Equation 36.136（条件付き独立）を確認できないのはなぜか。
2. **ロジスティック回帰としてのナイーブベイズ。** 2 クラス $y \in \{0, 1\}$ を考え、事前分布を $\pi_1 = p(y = 1)$、ベルヌーイパラメータを $\theta_{iy} = p(x_i = 1 \mid y)$ とおく。
   1. ナイーブベイズの事後分布が $p(y = 1 \mid \mathbf{x}) = \mathrm{sigmoid}(\mathbf{w}^\top \mathbf{x} + b)$（ただし $\mathrm{sigmoid}(z) = 1/(1 + e^{-z})$、$w_i = \log \frac{\theta_{i1}(1-\theta_{i0})}{\theta_{i0}(1-\theta_{i1})}$）となることを示せ。また、$b$ を θ_{iy} と π_1 を用いて表せ。w_i は、クラス 1 でクラス 0 よりも頻繁に塗られるピクセルに対してどちらの符号を持つか。
   2. 逆に、任意の組 $(\mathbf{w}, b)$ がある $\theta_{iy} \in (0, 1)$ と $\pi_1 \in (0, 1)$ の選択から生じることを示せ。したがって、ナイーブベイズとロジスティック回帰は同じ事後分布の集合を定めており、その当てはめ方だけが異なる。
   3. 訓練セットの 2 クラスが線形分離可能であるとする。2 クラスを分離する $(\mathbf{w}, b)$ に係数 $t \to \infty$ を掛けたとき、ロジスティック回帰の条件付き尤度はどのように変化するか。このことは $(\mathbf{w}, b)$ の最尤推定量に対して何を意味するか。同じデータに対して、ラプラス平滑化を伴うナイーブベイズはどのような $(\mathbf{w}, b)$ を返すか。
3. **ガウスのクラス条件付き分布。** 単一の特徴量が $x \mid y = 0 \sim \mathcal{N}(0, 1)$ および $x \mid y = 1 \sim \mathcal{N}(0, 4)$ に従い、両クラスの事前分布はともに $1/2$ であるとする。

1. x のうち、ベイズ分類器がクラス 1 に割り当てるものの集合を求めよ。x に対する単一のしきい値ではこの集合を作れないのはなぜか。また、どのような変換特徴量を用いれば境界がしきい値になるか。
2. d 次元において、クラス固有の分散を持つガウスナイーブベイズのクラススコアは次の式で与えられる。

$$\log p(y) - \sum_{i=1}^{d} \left[\frac{(x_i - \mu_{iy})^2}{2\sigma_{iy}^2} + \frac{1}{2} \log\left(2\pi\sigma_{iy}^2\right) \right]. \tag{36.144}$$

   2 クラス間の境界が交差項 $x_i x_j$ を持たない二次曲面になることを示せ。
3. 2 次元上の 2 クラスが等しい事前分布を持ち、クラス条件付き分布 $\mathcal{N}(\mathbf{0}, \Sigma_+)$ および $\mathcal{N}(\mathbf{0}, \Sigma_-)$（ただし $\Sigma_\pm = \left(\begin{smallmatrix} 1 & \pm\rho \\ \pm\rho & 1 \end{smallmatrix}\right)$、$0 < \rho < 1$）に従うとする。2 つのクラススコアの差が $2\rho x_1 x_2/(1-\rho^2)$ となり、したがってベイズ分類器がクラス $+$ を割り当てるのが $x_1 x_2 > 0$ のとき、かつそのときに限られること、さらに各クラスの単独のスコアは $x_1^2 + x_2^2$ にも依存することを示せ。両クラスは同じ周辺分布を持つため、ガウスナイーブベイズは無制限のデータがあってもランダム予測の水準にとどまる。$\mathcal{N}(\mathbf{0}, \Sigma_+)$ のもとでは $P(x_1 x_2 > 0) = \frac{1}{2} + \frac{1}{\pi}\arcsin\rho$（Sheppard の公式）が成り立つ。ベイズ誤差を ρ の関数として求め、$\rho = 0.1$ と $\rho = 0.9$ に対してその値を計算せよ。

4. **テキストに対する 2 つの事象モデル**。語彙 V が **goal**、**match**、**vote**、**poll** の 4 語からなるとする。訓練セットには、スポーツ文書が「goal goal match」「goal match match」「match vote」の 3 件、政治文書が「vote poll poll」「vote vote match」「poll vote」の 3 件含まれている。多項モデルは、文書をクラス固有の V 上の分布からの独立な単語抽出の系列として扱う。ベルヌーイモデルは、V の各単語についてその文書内に出現するかどうかだけを記録する。
   1. ラプラス平滑化を用いて両方のモデルを推定せよ。多項モデルでは、クラスの単語ごとの出現回数に 1 を加え、クラスの総単語数に $|V|$ を加える。ベルヌーイモデルでは、単語が出現する確率を $(n_{iy} + 1)/(n_y + 2)$ により推定する（ここで n_{iy} はクラス y の文書のうち単語 i を含む文書の件数である）。
   2. 文書「goal match poll poll」を両方のモデルで分類せよ。それぞれの判定を決定づけているのはどの単語か。
   3. ある文書をそれ自身と k 回連結せよ。2 つのモデルの対数事後オッズはどのように変化するか。実際のテキストでは、文書内に一度出現した単語は再び出現しやすい傾向があり (Church and Gale 1995)、この知見は演習問題 3（36.3 節）でも再現されている。これは多項モデルのどの仮定に違反するか。
5. [code] **対応のある比較**。本文では、`pred` と `pred_sr` の正解率によってナイーブベイズとソフトマックス回帰を比較した。
   1. 両モデルに対して同じ再標本インデックスを用いて画像ごとの正誤の差にブートストラップ法を適用し、正解率の差に対する 95% 区間を求めよ。
   2. この区間の幅を、2 つの周辺区間の幅および対応のない標準誤差 $\sqrt{p_1(1-p_1)/n + p_2(1-p_2)/n}$ と比較せよ。2 つの正誤ベクトルの相関を計算し、それを用いて対応付けが有効である理由を説明せよ。
   3. 両モデルとも誤るテスト画像、ナイーブベイズだけが正解するテスト画像、ソフトマックス回帰だけが正解するテスト画像の枚数をそれぞれ数えよ。正解率の差を決定するのはこれらのうちどの枚数か。また、ナイーブベイズの誤りの一部だけを修正したモデルであれば、この表はどのようになるか。
6. **重複および相関した証拠**。2 つのクラスが等しい事前分布を持ち、ある二値特徴量が $p(x = 1 \mid y = 1) = 0.8$ および $p(x = 1 \mid y = 0) = 0.2$ を満たすとする。
   1. パイプラインがこの特徴量を k 回保存するとする。実際には 80% の割合でしか当たらない予測に対してナイーブベイズが少なくとも 99.9% の確信度を主張する最小の k

を求めよ。
   2. 次に、部分的にのみ一致する 2 つのコピーを保存する。y が与えられたもとで、2 番目のコピーは確率 q で 1 番目と一致し、それ以外の場合は同じクラス条件付き分布から新たに独立にサンプルされるとする。両コピーとも x と同じ周辺分布を持ち、したがってナイーブベイズが依然として $16/17 \approx 0.94$ という確信度を両コピーが 1 のときに主張することを示せ。また、正しい事後分布が $r/(1+r)$（ただし $r = 4(0.8+0.2q)/(0.2+0.8q)$）で与えられることを示し、$q \in \{1, 0.75, 0.5, 0.25, 0\}$ に対してその値を計算せよ。過剰計上の度合いはコピー間の相関にどのように依存するか。
   3. ソフトマックスの前にすべてのクラススコアを $T = k$ で割ると、(a) において正しい事後分布が復元されることを示せ。これに等しい事前分布が必要となるのはなぜか。（事前項も k で割られるためである。）
7. [code] **ラベルシフト**。ラベルシフト（3.6 節）のもとでは、ターゲットの事前分布 $p_T(y)$ がソースの事前分布 $p_S(y)$ と異なる一方、$p(\mathbf{x} \mid y)$ は同じままである。
   1. ターゲットの事前分布のもとでの事後分布が $p_T(y \mid \mathbf{x}) \propto p_S(y \mid \mathbf{x})\, p_T(y)/p_S(y)$ となることを示せ。ナイーブベイズがこれを利用するのに再訓練を必要としないのはなぜか。
   2. プルオーバーが画像の 90% を占めるテストセット（テスト用プルオーバー全 1,000 枚と、他のクラスからランダムに抽出した 111 枚）を構築せよ。その上で、`pred` の正解率を、$p_T(\text{pullover}) = 0.9$ および他のすべてのクラスで $p_T(y) = 1/90$ とした (a) の補正規則の正解率と比較せよ。
   3. この補正はプルオーバーのスコアに $\log 9 \approx 2.2$ nat を加え、他のクラスから 2.2 を引く。これを `#mdl-naive-bayes-calibration-3` セルのスコア差と比較し、(b) で測定した変化の大きさを説明せよ。
8. [code] **温度スケーリング**。●演習問題 6 では、クラススコアを（ソフトマックス温度）$T = k$ で割ることで特徴量の k 重の複製が打ち消されることを見た。Fashion-MNIST では実効的な重複度は未知であるため、温度スケーリング (Guo et al. 2017) は代わりにホールドアウトデータ上で T を当てはめ、事後分布を $s_y(\mathbf{x})/T$ のソフトマックスとする。
   1. 最初の 50,000 枚の訓練画像でモデルを再当てはめし、T を残りの 10,000 枚における事後分布の平均負の対数尤度を最小化するように選べ。T を変えてもテスト正解率が変化しないのはなぜか。
   2. 当てはめた T を、同じ 50,000 枚モデルのテストセット上のスコアに適用し、ビンの境界を 0、0.5、0.7、0.9、0.99、1 として `#mdl-naive-bayes-calibration-3` セルの信頼性表を再び作成せよ。主張される平均確信度を正解率と比較せよ。
   3. 演習問題 6 の設定において、実データ上のスコア差の標準偏差がモデルが予測する標準偏差の $\sqrt{k}$ 倍になることを示せ。プルオーバーとコート、T シャツとシャツ、プルオーバーとシャツの組についてこの比の二乗を計算し、当てはめた T と比較せよ。
   4. 演習問題 7 で構築したテストセット上で、温度スケーリング後の事後分布に同演習の事前分布補正を適用せよ。温度スケーリングの後では、補正された事前分布によってより多くの予測が変化するのはなぜか。
9. [code] **欠損ピクセル**。すべてのテスト画像のランダムな半分のピクセルを、ブールマスク `M`（`X_test` と同じ形状で、観測可能なピクセルに対して `True` をとる）を用いて隠せ。
   1. この生成モデルのもとで、隠されたピクセルを $p(\mathbf{x} \mid y)$ から周辺化して消去すると対応する因子が取り除かれ、観測可能なピクセルのスコアが $\log p(y) + \sum_{i \text{ visible}} \log p(x_i \mid y)$ になることを示せ。これを 2 つの行列積 `(X_test * M) @ np.log(P_xy).T + ((1 - X_test) * M) @ np.log(1 - P_xy).T` に対数事前分布を加える形で実装し、全ピクセルを用いた場合の正解率 71% と比較せよ。
   2. 代わりに、隠されたピクセルを 0、1、または訓練画像の平均で埋めて、アフィン形式でスコアを計算せよ。それぞれの正解率を Equation 36.141 から説明せよ。また、0 で埋めることはすべてのクラスのスコアに何を加算するか。

3. 隠されたピクセルを 0 に設定した画像に対して、本文で訓練したソフトマックス回帰のスコアを計算せよ。(a) で生成モデルが用いた操作のうち、識別モデルに欠けている操作は何か。

10. [extended] **生成学習と識別学習の比較。●●**Ng and Jordan (2002) は、ナイーブベイズと、演習問題 2 により同じ事後分布族を条件付き尤度の最大化で当てはめるロジスティック回帰とを比較している。同論文では、ナイーブベイズのパラメータは次元 d に対して対数オーダーの訓練例数で母集団の値に近づくことが証明されている。一方、ロジスティック回帰が漸近誤差に近づくために必要な訓練例数は高々 d の線形オーダーであり、最悪ケースでは少なくとも d の線形オーダーになる。同論文の 15 個のデータセットの多くで、訓練セットが小さいときにはナイーブベイズの方がテスト誤差が低く、大きいときにはロジスティック回帰の方が低くなる。
    1. 各クラスが少なくとも m 個の訓練例を持つとする。Hoeffding の不等式（36.9 節）とユニオンバウンドを用いて、2 クラスモデルの全 $2d$ 個の平滑化なしの推定値 $\hat{\theta}_{iy}$ がすべて母集団の値の ε 以内に入る確率が、次式のもとで少なくとも $1-\delta$ となることを示せ。
    $$m \geq \frac{\log(4d/\delta)}{2\varepsilon^2}. \tag{36.145}$$
    また、$d = 784$、$\varepsilon = 0.05$、$\delta = 0.05$ に対してこの下界を計算せよ。
    2. $|\hat{\theta}_{iy} - \theta_{iy}|$ に対する一様上界だけでは、クラススコアの変化を上界で抑えることはできない。本文でプルオーバー対コートの最大の重みを持っていた隅のピクセルのように、θ_{iy} が 0 または 1 に近いピクセルではなぜ抑えられないのか。スコアがパラメータと同じレートで収束するためには、θ_{iy} について何を仮定しなければならないか。
    3. データをプルオーバーとコートに限定せよ。訓練セットの総サイズ $n \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$ および全 12,000 枚の画像について、両クラスを含むランダムな訓練部分集合を、20 個ずつ（$n = 200$ 以下の各サイズ）および 5 個ずつ（それより大きい各サイズ）抽出せよ。ラプラス平滑化を伴うナイーブベイズと、重み減衰なしのロジスティック回帰を当てはめよ。小さい n では訓練データが線形分離可能となり、演習問題 2c によりロジスティック損失は最小点を持たないため、ゼロ初期化から固定回数のフルバッチ勾配ステップを実行せよ。`d2l.plot` を用いて、両分類器の平均テスト誤差を対数軸上の n に対してプロットせよ。ナイーブベイズの方が低い誤差を示す n の範囲は存在するか、存在するならどの範囲か。
    4. ナイーブな仮定（条件付き独立）が成り立つ合成データを用いて (c) を繰り返せ。すなわち、均衡したラベルを抽出し、各ピクセルを Bernoulli(θ_{iy}) から独立にサンプルせよ（ただし θ_{iy} は 2 クラスの全訓練画像から推定した値とする）。2 本の学習曲線とそれぞれの漸近誤差はどのように変化するか。
    5. Ng and Jordan (2002) の定理 4 は、ナイーブベイズの過剰誤差を、母集団モデルのスコア差がゼロ付近に落ちる確率によって上から抑えている。実際のピクセルと (d) の合成データについてこの確率を比較し、その比較結果と (c) および (d) の漸近誤差とをあわせることで、2 組の学習曲線の振る舞いがどのように説明できるかを述べよ。

36.12 関連文献

本節の末尾にあるリストには、本章で学んだ内容に対応する書籍や講義をまとめている。それに先立ち、まずは本章で割愛した話題に目を向けよう。本章の構成順に沿って、各小項目ではこれまで扱った手法の限界から出発し、研究文献がそれをどのように乗り越えているかを解説する。なかでもコンフォーマル予測（conformal prediction）と信頼列（confidence sequence）の 2 つは実務でただちに役立つ手法であり、実際の実験における雑多な条件のもとで 36.5 節の区間推定や検定を補強する。また、VC 次元のように主として理論的な関心事であるものもあれば、勾配

ノイズが本当に重い裾（heavy tail）を持つのかどうかのように、現在も活発に議論されている話題もある。本書の以降の箇所は本節の内容に依存しないため、さらなる探究のための地図として活用されたい。

36.12.1 確率モデル

まずは分布の扱いに関する 3 つの拡張から始めよう。すなわち、離散と連続の狭間に位置する微妙な分布のクラス、分散が無限大となる平均に対する極限法則、そして選択肢どうしが似通っているときにソフトマックスが見せる古典的な癖である。

36.12.1.1 アトムも密度も持たない分布

36.1 節では、密度を持つ確率変数は点質量（アトム）を持たないことを述べ、あらゆる分布を離散分布、密度を持つ連続分布、またはその 2 つの混合として扱った。驚くべきことに、その逆は成り立たない。すなわち、アトムを持たないだけでは密度の存在は保証されないのである。理由を見るために、区間 $[0, 1]$ 内の乱数を 3 進展開で構成することを考えよう。各桁を公正なコインで独立に選び、数字は 0 と 2 のみを用いるものとする。この累積分布関数はいたるところで連続であるため、正の確率を持つ単独の点は存在しない。それにもかかわらず、確率質量のすべては全長が 0 であるカントール集合（Cantor set）の上に集中している。長さ 0 の集合上で積分して 1 になる密度関数は存在しない。

この分類を完結させるのが**ルベーグ分解**（Lebesgue decomposition）である。実直線上のあらゆる確率分布は、離散部分、密度を持つ部分、そしてカントール分布のような**特異連続**（singular continuous）部分の混合として表される。密度を持つことは、長さに関して**絶対連続**（absolutely continuous）であることと厳密に対応しており、その場合にはラドン・ニコディムの定理（Radon–Nikodym theorem）が密度を与える。Billingsley (1995) はこのような測度論的基礎の上に確率論を展開しており、そこでは期待値が分布に対する単一の積分として定式化され、本章で扱った和と積分は単なる 2 つの特別な場合として現れる。

36.12.1.2 重い裾と安定法則

36.1 節で述べたように、独立なコーシー分布からのサンプルの平均はまったく同じコーシー分布に従い、また演習問題 5 で見たように、べき乗則の裾を持つ確率変数の平均は、おなじみの $1/\sqrt{n}$ のレートよりはるかに遅くしか収束しない。ガウス分布に近づかないとすれば、いったい何に収束するのだろうか。裾 $P(|X| > x)$ が $x^{-\alpha}$（ただし $0 < \alpha < 2$）のように減衰するとき、分散は無限大となり、適切に中心化した和を $n^{1/\alpha}$ で（$\sqrt{n}$ ではなく）スケーリングすると、α **安定**法則（stable law）に収束する。コーシー分布は $\alpha = 1$ の対称安定法則であり、ガウス分布はその境界である $\alpha = 2$ に位置する。Gnedenko and Kolmogorov (1954) は、これらの極限法則の古典理論を展開している。

深層学習では、Simsekli et al. (2019) が α 安定分布としてモデル化したミニバッチ勾配ノイズと、訓練された重み行列の固有値スペクトル (Martin and Mahoney 2021) の双方において、重い裾が注目を集めている。もっとも、こうした主張は割り引いて受け止める必要がある。この種の経験的な主張はいずれも裾指数（tail index）α の推定に依拠しているが、その推定値は裾のどこまで外側を見るかによって大きく変動し得るからである。

36.12.1.3 選択モデルと無関係な選択肢

36.4.4 節では、各選択肢のスコアに独立なガンベルノイズを加えることでソフトマックスを導出した。この独立性の帰結として、任意の 2 つの選択肢間のオッズ比（odds ratio）$p_i/p_j = e^{z_i - z_j}$ は、選択肢リストにある他のすべての選択肢に依存しない。この性質は、2 つの選択肢がほぼ同

一である途端に破綻する。自家用車の運転と赤いバスの利用を半々で使い分けている通勤者を想像してみよう。市がまったく同じ青いバスを新たに導入したとしても、この通勤者は引き続き半分の割合で車を運転し、残りの半分を 2 種類のバスで分け合うはずである。ところがソフトマックスのもとでは、3 つの選択肢すべてに確率 $\frac{1}{3}$ が割り当てられてしまう。

入れ子ロジットモデル（nested logit model）や相関のあるガウスノイズを用いたプロビットモデルのように、類似した選択肢のノイズ間に相関を許せば、閉じた形式の選択確率は得られなくなるものの、車の選択確率を $\frac{1}{2}$ 付近に保つことができる。McFadden (1974) はガンベルノイズからロジットを導出するとともに、Luce の**無関係な選択肢からの独立性**（independence of irrelevant alternatives）として知られるこの制約を、モデル化における中心的な限界として指摘した。

36.12.2 推定と検定

36.5 節の信頼区間と仮説検定は、ガウス近似、誤差に対する既知のモデル、またはデータを見る前に固定された標本サイズという 3 つの好都合な前提に依拠していた。以下の 3 つの話題では、こうした前提を 1 つずつ手放したときに何が起こるかを見ていく。

36.12.2.1 改良されたブートストラップ区間

36.5 節で扱った単純なパーセンタイルブートストラップ法は、再サンプリングされた推定値の統計的バイアスや歪度（skewness）をそのまま引き継ぐため、小標本や歪んだ統計量に対しては実際の被覆率（coverage）が公称水準を大きく下回ることがある。2 つの改良手法がこのずれの大部分を解消する。**スチューデント化**（studentized）ブートストラップ法は、自己正規化されたピボット量 $(\hat{\theta}^{*} - \hat{\theta})/\widehat{\mathrm{se}}^{*}$ を（$\hat{\theta}^{*}$ そのものに代えて）再サンプリングする。一方、**BCa**（バイアス補正・加速、bias-corrected and accelerated）区間は、バイアスと θ に応じた標準誤差の変化の双方を考慮してパーセンタイルのカットオフを移動させる。片側区間では、いずれの改良も被覆誤差をオーダー $1/\sqrt{n}$ からオーダー $1/n$ へと縮小させる (Hall 1992)。数千件のテスト例に対する正解率を報告する際にはほとんど必要ないが、小規模な評価セットや誤り率の比を扱う場面ではただちに威力を発揮する。これらの手法については DiCiccio and Efron (1996) が明快な解説を与えている。

36.12.2.2 コンフォーマル予測

訓練済みのニューラルネットワークは点予測を出力するが、パラメトリックな誤差モデルを仮定しない限り、被覆率が保証された予測区間（prediction interval）は与えてくれない。**分割コンフォーマル予測**（split conformal prediction）は、モデル化の仮定を一切置かずにこれを実現する。ホールドアウトした n 個の較正点を用い、たとえば残差 $|y - f(\mathbf{x})|$ によってモデルが各点をどれだけ外したかをスコア化する。そして新しい入力に対しては、スコアが小さい方から数えて $\lceil (n+1)(1-\alpha) \rceil$ 番目の較正スコア以下となるすべてのラベルを返す。

較正点とテスト点が交換可能（exchangeable）である限り、得られる予測集合は**任意の**モデルおよび**任意の**データ分布に対して、少なくとも $1-\alpha$ の確率で真のラベルを含む。証明には並べ替え検定（permutation test）と同じ対称性を用いる。すなわち、交換可能性のもとでは、全 $n+1$ 個のスコアの中でのテストスコアの順位は一様に分布する。実務上は 1 つ注意点がある。この保証は入力全体にわたる平均として成り立つものであり、個々の入力で条件付けたときに成り立つわけではない。理論の全体像は Vovk et al. (2022) が展開しており、Angelopoulos and Bates (2023) は分類と回帰の両方についてコード付きのわかりやすいチュートリアルを提供している。

36.12.2.3 信頼列

36.5 節の z 検定は、結果を 1 つでも見る前に標本サイズ n を決めていることを前提としている。それとは逆に、進行中の実験をバッチごとに覗き見して $p < 0.05$ となった瞬間に打ち切るなら、十分に長く観察を続ける限り誤警報はほぼ確実に発生する。演習問題 10 では、どれほど頻繁に確認しても妥当性を保つ混合ガウスマルチンゲール検定を構成した。**信頼列**はこの考え方を区間推定全体へと推し進めたものであり、単一のパラメトリックモデルに限らず任意の有界データや準ガウス（sub-Gaussian）データに対して、すべての標本サイズ n で同時に成り立つ区間を与える。この自由度と引き換えに区間幅は広がるものの、最も鋭い境界を用いれば、余分な幅の増大は $\sqrt{\log \log n}$ 程度と驚くほど緩やかである。Howard et al. (2021) は非負優マルチンゲール（supermartingale）からこれらの列を構成しており、Johari et al. (2017) は常時妥当（always-valid）な p 値が商用の A/B テストプラットフォームにどのように導入されたかを解説している。

36.12.3 事後分布を用いた計算

36.8 節では、2 パラメータの事後分布に対して、単一のマルコフ連鎖であるランダムウォーク・メトロポリス法を適用した。しかし実際のベイズモデルは数千から数百万個のパラメータを持つため、より効率的なサンプラとスケーラブルな近似手法が求められる。

36.12.3.1 ギブスサンプリングとハミルトニアンモンテカルロ

ランダムウォーク・メトロポリス法（36.8.3 節）では、あらゆる方向について提案分布のスケールを調整する必要があり、小さな拡散的ステップで少しずつしか進めない。事後分布が高次元であったり強い相関を持っていたりすると、連鎖は大半の時間を行き来に費やしてしまう。**ギブスサンプリング**は、厳密な条件付き分布から 1 座標ずつ順番に更新することで提案分布の調整を不要にし、すべての移動が採択されるようにする。ただし難点として、それらの条件付き分布から容易にサンプリングできなければならず、また座標間の相関が強いと依然として収束は遅くなる (Geman and Geman 1984)。

ハミルトニアンモンテカルロはより物理学的なアプローチをとる。すなわち、$-\log \tilde{p}$ をポテンシャルエネルギーとみなし、ランダムな運動量（モメンタム）を注入したうえで、勾配を用いて生じる軌跡を多数ステップにわたってシミュレートする。このダイナミクスはエネルギーをほぼ保存するため、1 回の提案で事後分布上を遠くまで滑らかに移動しつつ、高い確率で採択される。手法とその調整法については Neal (2011) が明快な入門を与えており、また Hoffman and Gelman (2014) は No-U-Turn Sampler によって軌跡長を自動化し、これが Stan の標準エンジンとなった。

36.12.3.2 ニューラルネットワークの事後分布

36.8 節のすべてのアルゴリズムは、各ステップでデータセット全体にわたる完全な対数同時確率を評価し、またラプラス近似では $d \times d$ のヘッセ行列を保持して逆行列を求める必要がある。数百万個のパラメータと数百万件の訓練例を持つニューラルネットワークでは、そのどちらも実行不可能である。**確率的勾配ランジュバン動力学**（stochastic-gradient Langevin dynamics）は、演習問題 9 の未調整ランジュバン更新にミニバッチ勾配と減衰するステップサイズを組み込み、メトロポリス採択判定を手放す代わりに 1 ステップあたりのコストを通常の SGD と同一にする (Welling and Teh 2011)。一方、ラプラス近似は別の方法でスケールアップを図る。層ごとのクロネッカー分解ブロックや最終層のみのヘッセ行列など、曲率のうち最も情報量の多い部分だけを保持するのである。Daxberger et al. (2021) は、こうした軽量な近似がごくわずかな計算量でありながら、はるかに高コストなベイズ手法に匹敵する性能をしばしば達成することを示している。

36.12.4 集中と学習理論

以下の最初の 2 つの話題では、36.9 節と 36.10 節の道具立てを、単純な平均やラデマッハ複雑度（Rademacher complexity）の先へと一般化する。後半の 2 つの話題では、本章の理論パートの締めくくりで残された謎に取り組む。すなわち、仮説クラスに潜む最悪の関数に対してではなく、アルゴリズムが実際に見つけ出す特定の解に対してどのように保証を証明するかという問題である。

36.12.4.1 有界差分とリプシッツ関数

36.9 節のチェルノフ法は、独立な項の和であることを前提としている。しかし、モデルクラス全体にわたる真のリスクと経験リスクの最大乖離など、関心のある多くの量は独立な入力に依存するものの単純な和の形をしていない。**マクディアミッドの不等式**（McDiarmid's inequality）は、ヘフディングの上界をそのような関数へと一般化する。「各項が長さ c_i の区間に収まる」という条件を「入力 i のみを変えたとき関数の変化量は高々 c_i である」に置き換えても、積率母関数（moment generating function）の積をマルチンゲールに置き換える証明によって、まったく同じ裾の上界 $e^{-2t^2/\sum_i c_i^2}$ が成り立つ。

ガウスベクトルはさらに明快な性質を持つ。任意の次元 d の標準ガウスベクトルに対して、すべての L リプシッツ関数 f は $P(|f - \mathbb{E}f| \geq t) \leq 2e^{-t^2/(2L^2)}$ を満たす。ユークリッドノルムは 1 リプシッツであるため、薄い球殻（thin shell）の周りでの揺らぎはオーダー 1 にとどまり、それは d がどれほど大きくなっても変わらない。Boucheron et al. (2013) は、これら両方の結果とその背後にある理論的枠組みを包括的に扱っている。

36.12.4.2 VC 次元と被覆数

36.10 節では、ラデマッハ複雑度を通じて無限の仮説クラスを制御した。これに代わる古典的なアプローチでは、有限個の点の集合に対してクラスが何通りの異なるラベル付けを生成できるかを数える。二値分類器の族について、全 2^{d+1} 通りのラベル付けをいかなる $d+1$ 個の点に対しても実現できないとする（たとえば $\mathbb{R}^m$ 内の線形半空間では $d = m + 1$ である）。このときザウアーの補題（Sauer's lemma）により、このクラスが生成するラベル付けの総数は高々 $(en/d)^d$ 通りであることが任意の $n \geq d$ 個の点に対して保証され、それらのラベル付けにわたるユニオンバウンドからオーダー $\sqrt{d \log n / n}$ の汎化ギャップが得られる。これが Vapnik and Chervonenkis (1971) による著名な議論である。実数値関数クラスに対しては、代わりに各解像度 ϵ でクラスを有限個の代表元によって被覆する。スケールにわたって和をとる**チェイニング**と呼ばれる技法を用いると、余分な $\log n$ 因子を取り除くことができる (Vershynin 2018)。

36.12.4.3 PAC-Bayes 限界

36.10 節のすべての限界は、データを見る前に固定された仮説クラス内の最悪の関数に対して備えるものであった。**PAC-Bayes** 限界は視点を転換し、データを観測した**後**に選ばれる関数上の事後分布 Q を、あらかじめ固定した事前分布 P を基準として評価する。このとき、クラスの複雑度項は平方根の中の $D_{\mathrm{KL}}(Q\|P)/n$ に置き換わる (McAllester 1999)。仮説クラス全体がどれほど巨大であっても、事後分布が事前分布の近くにとどまる限り、支払う複雑度のペナルティは小さくて済む。この考え方を土台として、Dziugaite and Roy (2017) は SGD が見つけた重みの周囲でガウス事後分布を最適化し、訓練例よりはるかに多くのパラメータを持つ確率的ネットワークに対して、二値 MNIST タスクにおける非自明な（nonvacuous）汎化限界を導出した。

36.12.4.4 アルゴリズム的安定性

ランダムラベル実験（4.5.1.1 節）から明らかなように、ニューラルネットワークが表現**し得る**関数のクラスは、実データ上で SGD が実際に訪れる部分集合よりはるかに大きい。**アルゴリズム的安定性**（algorithmic stability）は、仮説クラスではなく学習アルゴリズムそのものを抑える。任意の 1 個の訓練例を入れ替えたとき、学習されたモデルの任意のテスト点における損失の変化が高々 β であれば、汎化ギャップの期待値もオーダー β となり、マクディアミッドの不等式によってその期待値を高確率の限界へと引き上げることができる (Bousquet and Elisseeff 2002)。滑らかでリプシッツ連続な凸損失に対して、SGD は β がステップサイズの総和を n で割った値に比例する形で安定となるため、訓練のステップ数が少ないかステップサイズが小さいほど、よりタイトな保証が得られる (Hardt et al. 2016)。

36.12.5 カウントに基づく分類器

最後に、36.11 節から続く 2 つの発展的話題で締めくくろう。1 つは単一の温度だけでは不十分なときに較正のずれた確率を補正する手法であり、もう 1 つは純粋に識別的な当てはめでは直接扱えないラベルなしデータからの学習を生成分類器によって実現する手法である。

36.12.5.1 単一温度を超える較正

温度スケーリング（演習問題 8）が機能するのは、分類器のスコアが単一の乗法的スケールを除いて正確である場合である。歪みがより複雑なときには、さらに表現力の高い写像が役立つ。**Platt スケーリング**は二値スコアを確率へ写すロジスティック曲線の内部で傾きとオフセットの両方を当てはめ (Platt 1999)、一方アイソトニック回帰（isotonic regression）は任意の非減少関数を当てはめる。Zadrozny and Elkan (2002) はアイソトニック回帰を適用して、ナイーブベイズをはじめとする分類器を較正した。ここでのトレードオフはおなじみのものである。柔軟な較正器ほど過学習を避けるためにより多くのホールドアウトデータを必要とし、現代の深層ネットワークに対しては単一の温度だけでも同等にうまく機能することが多い (Guo et al. 2017)。

36.12.5.2 半教師ありナイーブベイズ

多くの実問題では、ラベル付き例の取得にはコストがかかる一方で、ラベルなしの入力は豊富に手に入る。ナイーブベイズは各クラスがどのように特徴量を生成するかをモデル化するため、欠測しているクラスを潜在変数として扱えば、ラベルなし例も尤度の中に自然に組み込める。すると 36.7 節の EM アルゴリズムによって、ラベルなし例に対するクラスの事後分布の計算と、その事後分布を重み付きの小数ラベルとして用いた単語カウントやピクセルカウントの更新を交互に繰り返すことができる。Nigam et al. (2000) は、ごく少数の文書しかラベル付けされていない状況でテキスト分類の正解率が大幅に向上することを実証する一方、生成モデルのデータへの当てはまりが悪い場合にはラベルなしデータがかえって性能を損ない得ることも警告している。

36.12.6 確率・統計の一般的な参考資料

書籍

- 『Introduction to Probability』（Blitzstein・Hwang）は、確率変数、期待値、条件付け、および 36.1 節と 36.3 節で扱った代表的な分布への親しみやすい入門書である。Harvard 大学の講義 Stat 110 に合わせた無料 PDF が公開されている。
- 『Mathematics for Machine Learning』（Deisenroth・Faisal・Ong）では、第 6 章において 36.4 節の指数分布族を含む確率と分布の自己完結した解説が与えられている。

- 『All of Statistics』（Wasserman）は、推定、検定、信頼区間、ブートストラップ法をコンピュータ科学者にも読みやすく明快にまとめた書籍であり、36.5 節と 36.6 節の範囲を網羅している。サポートページには正誤表、コード、データセットが用意されている。
- 『Computer Age Statistical Inference』（Efron・Hastie）は、ブートストラップ法、大規模検定、および頻度主義推論とベイズ推論の関係を掘り下げており、36.5 節を補完する一冊である。
- 『Pattern Recognition and Machine Learning』（Bishop）は、指数分布族、混合モデルと EM（36.7 節）、および 36.8 節のラプラス近似と変分近似を扱っている。Microsoft Research から無料 PDF が提供されている。
- 『Probabilistic Machine Learning: An Introduction』（Murphy）は、MAP 推定（36.6 節）や生成分類器（36.11 節）を含め、確率と決定理論の視点から機械学習を展開しており、無料の草稿と実行可能なノートブックが公開されている。
- 『Information Theory, Inference, and Learning Algorithms』（MacKay）は、推論と情報理論を有機的に結びつけた書籍であり、ラプラスの方法、モンテカルロ法、変分推論に関する各章は 36.8 節の副読本となる。
- 『Bayesian Data Analysis』（Gelman ほか）は、36.8 節に関連する事前分布、事後分布、収束診断を含むマルコフ連鎖モンテカルロ法、および実践的なベイズワークフローを扱っており、第 3 版が無料 PDF として公開されている。
- 『High-Dimensional Probability』（Vershynin）は、集中不等式、準ガウス変数、高次元幾何学を展開し、36.9 節の内容をさらに発展させている。
- 『Understanding Machine Learning: From Theory to Algorithms』（Shalev-Shwartz・Ben-David）は、36.10 節と同様に一様収束、ラデマッハ複雑度、マージン限界を導出しており、著者らによる無料 PDF が提供されている。

講義とビデオ講義

- Statistics 110: Probability（Harvard 大学、Blitzstein）では、36.1 節と 36.3 節に対応する講義動画、演習課題、および解答付きの練習問題 250 問以上が公開されている。
- Stat 110 講義動画（YouTube）には、確率、分布、条件付けに関する講義シリーズ全編が収録されている。

チュートリアルと講義ノート

- Probability Theory Review（Stanford 大学 CS229、Maleki・Do）は、機械学習向けに確率変数、期待値、分散、ガウス分布、多変量確率を復習する資料であり、36.1 節の簡潔な補助資料として役立つ。

Chapter 37

情報理論とダイバージェンス

本書のほとんどのモデルは、交差エントロピー損失（cross-entropy loss）を最小化することで訓練される。これは、観測された結果に高い確率を割り当てるという直観的な目標を、勾配降下法のための滑らかな目的関数へと変換するものである。しかし、その損失の数値は実際には何を測っているのだろうか。言語モデルが文字あたり 1.2 nat を示したとき、その値は原理的にどこまで下がり得るのか。また、交差エントロピーを変分オートエンコーダ（variational autoencoder）の KL ペナルティや敵対的生成ネットワーク（generative adversarial network）のクリティックに置き換えるとき、何を比較しているのだろうか。

Claude Shannon は 1948 年、ノイズのある回線を通してどれだけ効率よくメッセージを送れるかという問いに取り組むなかで、これらの疑問に答えた。予測不可能性の尺度である**エントロピー**は、いかなるモデルの交差エントロピーも下回ることのできない厳密な下限を定める（3.1.2.3 節）。ここから導かれる 3 つの帰結が、深層学習全体を貫いている。第 1 に、交差エントロピーはデータのエントロピーを、データからモデルへの **KL ダイバージェンス**の分だけ上回り、この差は両者が一致するときに限り 0 になる。第 2 に、ビット単位で表したモデルの総損失は、高々 2 ビットのオーバーヘッドを除けば文字どおり圧縮ファイルサイズそのものであり、これによって言語モデルのパープレキシティ（8.3.2 節）に直接的な操作的意味が与えられる。第 3 に、学習されたクリティックはダイバージェンスや共有情報量の下界しか与えない。対照学習（contrastive learning）において N 個の候補から対応する相手 1 つを識別できても保証されるのは高々 $\log N$ nat であり、弱い敵対的クリティック（16.2 節）のもとでは遠く離れた分布どうしが近く見えてしまうことがある。

本章は 3 部構成をとる。第 1 部では、エントロピーと交差エントロピーを展開し（37.1 節）、対数損失（log-loss）を可逆圧縮（lossless compression）およびパープレキシティへと直接結びつける（37.2 節）。第 2 部では、密度比ダイバージェンス（density-ratio divergence）の決定的な破綻を取り上げる（37.3 節）。モデルのサンプルがデータと重ならないときには必ず KL が発散し、有界な f ダイバージェンスは飽和して勾配が 0 になってしまう。滑らかなテスト関数の期待値や最適輸送（optimal transport）のコストを比較すれば（37.4 節）有用な勾配を回復でき、一方で対数密度（log density）の傾きを一致させれば（37.5 節）、計算困難な分配関数（partition function）を扱うことなく非正規化モデルを比較できる。第 3 部では、相互情報量（mutual information）、データ処理、ファノの不等式（Fano's inequality）、情報ボトルネック（information bottleneck）を通じて確率変数の対を調べ（37.6 節）、最終的にサンプルに基づく推定と InfoNCE へと至る（37.7 節）。なお、最後の 2 節は 37.1 節の後だけで自己完結しているため、輸送とスコアの話題を第 16 章および第 17 章まで取っておきたい場合は、そのまま先に読み進めてもよい。

本章では、イェンセンの不等式（第 35 章）、条件付き分布、および最尤法（第 36 章）を既知とす

る。誤り訂正符号（error-correcting code）、コルモゴロフ複雑性（Kolmogorov complexity）、情報幾何学（information geometry）については、37.8 節で概観する。

37.1 エントロピー、交差エントロピー、KL ダイバージェンス

分類器を訓練するとき最初に確認するのは損失曲線であり、その形状の読み方はすぐに身につく。右下がりの曲線は順調な兆候であり、横ばいの曲線は疑わしい。一方で、縦軸の数値そのものを解釈するのはずっと難しい。これまでのところ説明できる数値は 1 つだけである。初期化したばかりの 10 クラス分類器は約 2.30 の損失を示すが、これは 4.2 節で未訓練ネットワークの健全性チェックに使った値である。この数値は $\ln 10$ であり、モデルが 10 個のラベルをすべて等確率とみなしているときに、そのうち 1 つを指定するためのコストにあたる。曲線の反対側の端は、さらに深い問いを投げかける。最終的な損失 0.6 は良い値なのだろうか。曲線が横ばいになったとき、そのプラトーは訓練のバグなのか、それともデータの避けられない性質なのか。また、ラベルの一部に誤りがある場合、モデルはそもそもどこまで損失を下げられるのだろうか。

交差エントロピー損失（cross-entropy loss）は $-\log q(y)$ の平均であり、実際に生じたラベル y に対してモデルがどれだけ驚いたかを測る。この平均は自然に 2 つの部分に分解される。1 つ目は**エントロピー**であり、ラベルの真の分布をすでに知っている観測者が経験する平均的な驚きを表す。エントロピーはデータそのものによって決まるため、いかなるモデルもこれを減らすことはできない。2 つ目は**カルバック・ライブラー（KL）ダイバージェンス**であり、真の分布の代わりに不完全なモデルを使うことで生じる余分な驚きを表す。この量は決して負にならず、モデルが真の分布と一致するときに限り 0 になる。したがって、損失の値は、この削減不能な下限に照らして初めて評価できる。

前章では、これらの考え方のうち 2 つにすでに出会った。最尤法（maximum likelihood）が交差エントロピー（cross-entropy）を最小化すること（36.6.2.1 節）、そしてギブスの不等式（Gibbs' inequality）として知られる結果、すなわち KL ダイバージェンスが決して負にならないこと（36.4.2 節）である。前章ではこれらを道具として用いたが、本節ではこれら自体を主題とする。まずは 1 組のトランプと 4 記号の情報源から始め、そこではエントロピーが単に「はい・いいえ」の質問の回数を数えるものにすぎないことを見る。そこから 2 つの分布の比較へと進み、続いて分類器に戻ってその損失の下限を求め、最後にターゲットをソフト化した場合や結果が連続な実数値である場合に何が変わるかを調べる。

対数の底は単に単位の選択にすぎない。底 2 は情報量を**ビット**で測り、底 e は **nat** で測る。**本書では一貫して nat で計算を進める**。これは数値計算ライブラリや報告される訓練損失で使われる自然対数 `log` に合わせたものである。「はい・いいえ」の質問の回数を数えるときにはビットの方が便利であるため、ビット単位の量には下付き添字 2 を付ける（たとえばエントロピーなら H_2）。1 ビットは $\ln 2 \approx 0.693$ nat に等しい。Table 37.1 に、本章全体で用いる記号をまとめる。

Table 37.1: 情報理論の章で用いる記号。2 値エントロピー $H_b(p)$ は確率変数ではなく数値の関数である。相互情報量 $I(X;Y)$ および最後の 5 行は後の節で定義する。

記号	意味	単位
$I(x) = -\log p(x)$	結果 x の自己情報量（サプライザル）（慣例により相互情報量 $I(X;Y)$ と同じ文字を用いる）	nat
$H(X)$, $H(P)$	離散確率変数または離散分布のエントロピー（下付き添字 2 は H_2 や CE_2 のようにビットを表す）	nat
$H_b(p)$	表の確率が p であるコインの 2 値エントロピー	nat

記号	意味	単位
$h(X)$	連続確率変数の微分エントロピー	nat（X の単位に依存）
$D_{\mathrm{KL}}(P\|Q)$	P から Q への KL ダイバージェンス	nat
$\mathrm{CE}(P,Q)$	P から Q への交差エントロピー	nat
k	クラス数（結果の集合のサイズ）	
n	例の数、またはメッセージ中の記号数、$\mathbb{R}^n$ の次元	
$\mathbf{p}$, $\mathbf{q}$	k クラス上の確率ベクトル（真の分布、モデル）	
$\mathbf{z}$	ロジット（$\mathbf{q} = \mathrm{softmax}(\mathbf{z})$）	
ϵ	ラベル平滑化の質量（後の節ではノイズ変数）	
τ	温度	
ρ	ラベルノイズ率（記録されたラベルがクリーンなクラスでない確率）	
$H(X \mid Y)$, $I(X;Y)$	条件付きエントロピー、相互情報量	nat
$D_f(P\|Q)$	ダイバージェンス関数 f を持つ f ダイバージェンス	
T	クリティック（2 つの分布のもとでの平均がダイバージェンスの限界を与える関数）	
g	積分確率指標のテスト関数	
γ	2 つの分布のカップリング（輸送計画）	
N	比較する 2 つの標本の各点数、または対照バッチ内の候補数	

37.1.1 情報量とエントロピー

37.1.1.1 自己情報量

シャッフルしたばかりの 1 組のトランプに関する 4 つの報告を考えてみよう。「カードが見える」という報告は、事象が確実であるため何も伝えない。「ハートが見える」は結果を等確率な 4 つのスートのうち 1 つに絞り込み、「スペードの 3 である」は等確率な 52 枚のカードのうち 1 枚を特定し、デッキ全体の並び順の報告は $52!$ 通りの可能な並び順のうち 1 つを特定する。事象の確率が低いほど、それを観測したときに得られる情報は大きくなる。Shannon (Shannon 1948) はこの考え方を、結果 x（確率 $p(x)$ で生じる）の**自己情報量**（self-information）によって定量化した。

$$I(x) = -\log p(x).$$

この量は結果の**サプライザル**とも呼ばれ、3.1.2.3 節ではこちらの用語を用いた。結果を予期していなかったほど、驚きは大きくなる。重要なのは、この驚きは確率を割り当てた主体に属するという点であり、確信を持って誤っているモデルはありふれた事象に対しても強く驚くことになる。確実な事象が持つ情報量は 0 であり、結果の確率が半分になるとサプライザルは $\ln 2$ nat（1 ビット）増加する。最初の 3 つのカードの報告が持つ情報量は、それぞれ 0、$\ln 4 \approx 1.39$、$\ln 52 \approx 3.95$ nat である。公正なコインを 1 回投げて得られる情報量を 1 ビットとするビット単位では、それぞれ 0、2、5.70 ビットとなる。

なぜ対数を選ぶのだろうか。ここで欲しい性質は 3 つある。確実な事象の情報量は 0 であること、まれな事象ほど多くの情報量を持つこと、そして 2 つの独立な事象の情報量は足し合わされることである。独立な事象の確率は掛け合わされるため、3 つ目の要件は $I(p_1 p_2) = I(p_1) + I(p_2)$ と書ける。この恒等式を満たす減少関数は $-\log p$ の正の定数倍に限られる（Csiszár (2008)、証明は演習問題 1 で扱う）。定数倍の選択は、単に単位の選択に相当する。

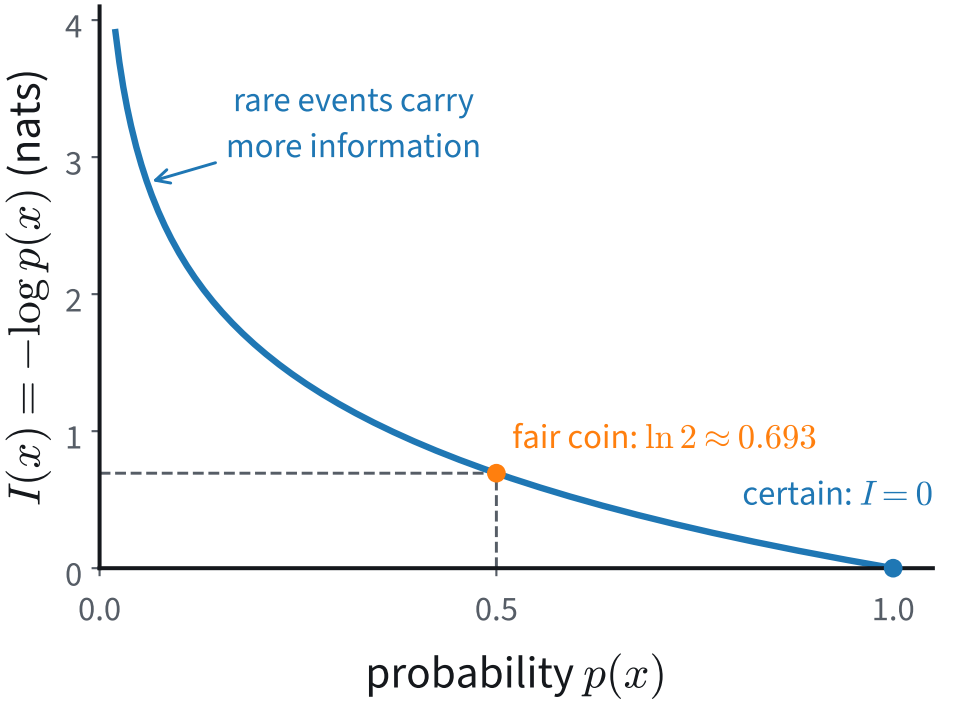


Figure 37.1: 確率 p の関数として表した nat 単位の自己情報量 $I(x) = -\log p(x)$。確実な事象（$p = 1$）の情報量は 0 であり、公正なコイン（$p = \frac{1}{2}$）は $\ln 2 \approx 0.693$ nat を持ち、$p \to 0$ で曲線は発散する。

Figure 37.1 にこの曲線を示す。単調減少かつ凸であり、$p = 1$ で 0 となり、$p \to 0$ で発散する。4 つ目のカードの報告であるデッキ全体の並び順は、この曲線のはるか左方の $\ln 52!$ に位置する。その値は対数ガンマ関数（log-gamma function）を用いて $\ln 52! = \ln \Gamma(53)$ として計算できるため、68 桁の整数 52! を直接作る必要はない。

```
deck_order = math.lgamma(53)          # ln 52! without forming 52!
print(f'the order of the deck: {deck_order:.2f} nats '
      f'= {deck_order / math.log(2):.1f} bits')
```

```
the order of the deck: 156.36 nats = 225.6 bits
```

シャッフルされたデッキの並び順を指定することは 156 nat の情報量を持ち、これは公正なコイン投げ約 226 回分に相当する。

37.1.1.2 エントロピー

サプライザルは結果が観測された後にしか計算できない。事前の段階では、平均してどれほど驚くと期待されるかを問うことができる。X を、値 x_i を確率 p_i でとる離散確率変数とする。その**エントロピー**は、結果の自己情報量の期待値として次の式で定義される。

$$H(X) = -\mathbb{E}_{x\sim P}[\log p(x)] = -\sum_i p_i \log p_i, \tag{37.1}$$

ここで規約として $0\log 0 = 0$ と定めるが、これは連続性（$t\log t \to 0$ が $t \to 0$ で成り立つこと）によって正当化される。すなわち、起こり得ない結果の寄与は 0 である。エントロピーは分布 P（X の分布）にのみ依存するため、$H(P)$ と $H(X)$ を区別せずに用いる。

有限集合上では、ビット単位のエントロピーは具体的な操作的意味を持つ。すなわち、結果を特定するために優れた質問者が行う「はい・いいえ」の質問の平均回数を表す。分布 $P = (\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$（4 つの記号 a, b, c, d 上の分布）を考えよう。本節を通してこの例を用いることとし、すべての確率が $\frac{1}{2}$ のべき乗であることから、これを **2 進情報源**（dyadic source）と呼ぶ。優れた質問者はまず「それは a か」と尋ね、半分の確率でその場で結果を確定させる。そうでなければ「それは b か」、さらに「それは c か」と尋ね、ここでも「いいえ」であれば d だけが残る。質問回数の期待値は次のとおりである。

$$\tfrac{1}{2}\cdot 1 + \tfrac{1}{4}\cdot 2 + \tfrac{1}{8}\cdot 3 + \tfrac{1}{8}\cdot 3 = 1.75,$$

これはビット単位のエントロピー $H_2(P) = \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + 2\cdot\frac{1}{8}\log_2 8 = 1.75$ ビット（すなわち 1.213 nat）と一致する。各結果は $\log_2(1/p_i)$ 回の質問で特定され、これはビット単位の自己情報量にほかならない。

対照的に、確率を無視する質問者は「a または b か」と尋ねてから「その 2 つのうちどちらか」と尋ねるため、常に 2 回の質問を必要とする。ここで、質問戦略とは実は姿を変えた 2 元符号（binary code）であることに注目しよう。「はい」に 1、「いいえ」に 0 を割り当てれば、結果を特定する回答の系列がそのまま符号語（codeword）となり、ここでは 1、01、001、000 がそれぞれ a、b、c、d に対応する。ここで平均質問回数がエントロピーとちょうど一致したのは、すべての確率が $\frac{1}{2}$ のべき乗だったからである。符号の言葉で言えば、この回数が大きく外れることは決してないことが 37.2 節で証明される。有限集合上のいかなる分布に対しても、最良の戦略が必要とする平均質問回数は H_2 から $H_2 + 1$ 回の範囲に収まる。

コードでは、関数 `entropy` が確率 0 の要素をマスクして除外することで $0\log 0 = 0$ を実装している。まずは 2 進情報源から始め、本節を通してこの関数を再利用する。

```python
def entropy(p):
    """Entropy of a probability vector, in nats (0 log 0 = 0 by masking)."""
    p = np.asarray(p, dtype=float)
    p = p[p > 0]
    return float(-np.dot(p, np.log(p)))

p_dyadic = np.array([1 / 2, 1 / 4, 1 / 8, 1 / 8])
h = entropy(p_dyadic)
print(f'H = {h:.3f} nats = {h / math.log(2):.2f} bits')
```

```
H = 1.213 nats = 1.75 bits
```

質問で数えたのと同じ 1.75 ビットが返される。

$0 \le p_i \le 1$ であるため、各項 $-p_i \log p_i$ は非負であり、したがって $H(X) \ge 0$ が成り立つ。等号成立は点質量（point mass、すなわち結果が確定している変数）の場合に限られる。また、エントロピーは確率の並びだけに依存し、結果の数値や名前には依存しないことにも注目したい。値 1、100、10^6、10^9 をこの 2 進情報源の確率でとる変数も、エントロピーは同じく 1.75 ビットである。

37.1.1.3 結合エントロピーと条件付きエントロピー

機械学習では、ラベルは常に入力と対になって現れ、文字はそれに先行するテキストの後に続く。したがって、2 つの変数を合わせたエントロピーと、一方の変数が既知となったときに他方に残るエントロピーの両方が必要になる。前者には新しい概念は不要である。対 (X, Y)（同時確率質量関数 $p(x, y)$ を持つ）は単に結果が対である 1 つの確率変数にすぎないため、その**結合エントロピー**（joint entropy）は $H(X, Y) = -\mathbb{E}[\log p(X, Y)]$ である。**条件付きエントロピー**（conditional entropy）は、Y に関して、X が観測された後に残る平均サプライザルであり、次の式で定義される。

$$H(Y \mid X) = -\mathbb{E}[\log p(Y \mid X)] = \sum_x p(x)\, H(Y \mid X = x),$$

ここで、$H(Y \mid X = x)$ は条件付き分布 $p(\cdot \mid x)$ のエントロピーである。$\log p(x, y) = \log p(x) + \log p(y \mid x)$ が成り立つため、両辺の期待値をとるとエントロピーの**連鎖律**（chain rule）が得られる。

$$H(X, Y) = H(X) + H(Y \mid X). \tag{37.2}$$

言葉で言えば、対のエントロピーは、第 1 の変数のエントロピーに、第 1 の変数がわかった後に第 2 の変数に残る不確実性を加えたものである。連鎖律を繰り返し適用すると、$H(X_1, \dots, X_n) = \sum_{i=1}^{n} H(X_i \mid X_1, \dots, X_{i-1})$ が得られる。すなわち、系列のエントロピーは、各記号をそれ以前のすべての記号で条件付けたエントロピーの総和である。言語モデルは、これと同じ左から右への順序で、1 記号ずつテキストを予測する。

質問ゲームはこの恒等式を直接的に例示している。X を最初の質問「それは a か」への回答とし、Y を 2 進情報源の結果とする。この回答は公正なコインと同じであるため、$H_2(X) = 1$ ビットである。「はい」の後はもう尋ねることは残らない（0 ビット）。「いいえ」の後は、結果は b、c、d のいずれかであり、その確率は $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$、エントロピーは 1.5 ビットである。2 つの回答にわたって平均をとると、$H_2(Y \mid X) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1.5 = 0.75$ ビットとなる。連鎖律により、$H_2(X, Y) = 1 + 0.75 = 1.75$ ビットであり、最初の回答が 1 ビットを与え、残りの質問が残りを与える。この和は当然ながら $H_2(Y)$ に等しい。結果 Y を知れば回答 X は完全に定まるため $H(X \mid Y) = 0$ であり、逆順の連鎖律は $H(X, Y) = H(Y) + 0$ となるからである。

37.1.1.4 有限集合上の最大エントロピー

k 個の結果を持つ変数のエントロピーは、最大でどれほど大きくなり得るだろうか。最小の場合である 1 回のコイン投げから始めよう。表の出る確率が p のベルヌーイ変数は、次の **2 値エントロピー**（binary entropy）を持つ。

$$H_b(p) = -p \log p - (1 - p) \log(1 - p),$$

これは確率変数ではなくスカラー p の関数である。Figure 37.2 にこの曲線を示す。結果が確定する $p \in \{0, 1\}$ でエントロピーは 0 になり、公正なコインで最大値 $H_b(\frac{1}{2}) = \ln 2 \approx 0.693$ nat（1 ビット）に達する。$p = 0.1$ の偏りのあるコインでは $H_b(0.1) \approx 0.325$ nat となるが、これは結果がはるかに当てやすいためである。

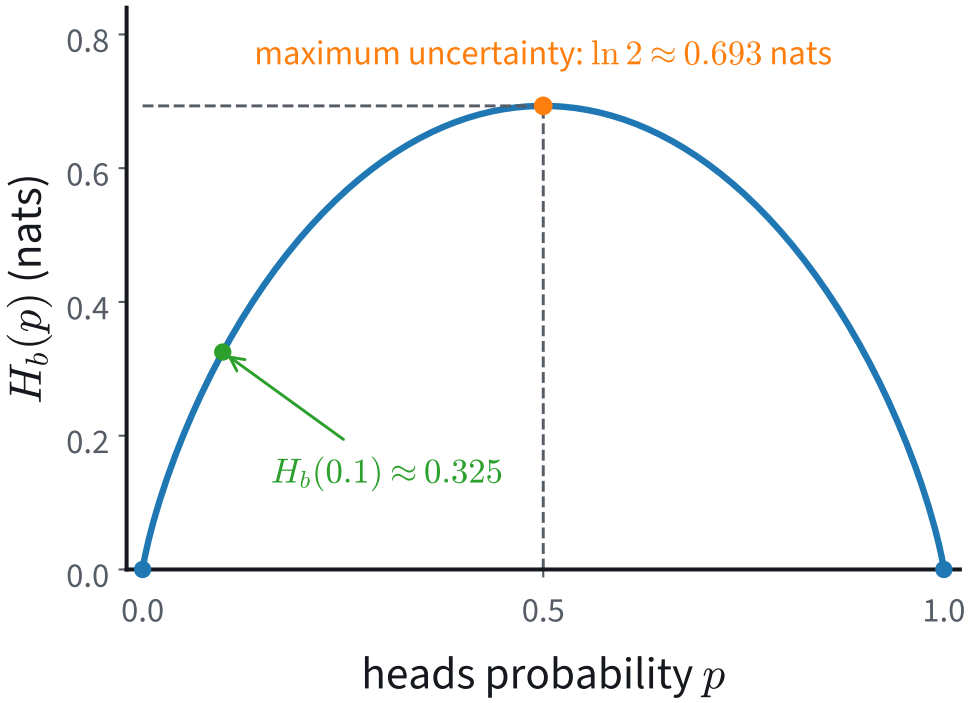


Figure 37.2: nat 単位の 2 値エントロピー $H_b(p)$。$p = \frac{1}{2}$ に関して対称であり、そこで最大値 $\ln 2 \approx 0.693$ nat をとる。決定論的な両端 $p = 0$ および $p = 1$ では結果に驚きがなく、エントロピーは 0 になる。曲線は凹であり、任意の弦はその下側にある。

この曲線が凹であることに注目しよう。曲線上の任意の 2 点を結ぶ弦は曲線の下側にある。曲線は一様なコインで最大となり、同じ原理が任意の有限個の結果に対しても成り立つ。X が高々 k 個の値をとるならば、次が成り立つ。

$$H(X) \le \log k,$$

等号が成り立つのは、X が k 個すべての値の上で一様分布に従うとき、かつそのときに限る。一様分布は確率質量を可能な限り均等に広げるため、結果の推測が最も難しくなる。4 つの結果に対する上界は $\log_2 4 = 2$ ビットであり、これは 0.25 ビットだけ 2 進情報源の 1.75 ビットを上回る。証明はギブスの不等式を導入するまで少しの間だけ先送りしよう。ギブスの不等式が揃えば、わずか 1 行で証明できる。その不等式は 2 つの分布を比較するものであるため、次は分布の比較へと進む。

37.1.2 KL ダイバージェンスと交差エントロピー

37.1.2.1 カルバック・ライブラー・ダイバージェンス

モデル Q が真の分布 P をどれほど悪く近似しているかを測りたいとしよう。情報理論はこれに**カルバック・ライブラー（KL）ダイバージェンス**（別名**相対エントロピー**（relative entropy））で答える (Kullback and Leibler 1951)。これは、Q を信じている人が、実際には P からサンプリングされる結果に対して被る余分なサプライザルの平均である。質問ゲームにおいて、ビット単位の KL ダイバージェンスは、Q 向けに設計された戦略が P からサンプリングされる結果に対して必要とする余分な質問の平均回数（整数の質問回数への丸めを除く）にあたる。P と Q が確率質量関数（または確率密度）p と q を持つとする。P から Q への KL ダイバージェンスは次の式で定義される。

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{x\sim P}\left[\log \frac{p(x)}{q(x)}\right]. \tag{37.3}$$

期待値の中の量は $\log p(x) - \log q(x) = (-\log q(x)) - (-\log p(x))$ であり、これは Q を信じる人のサプライザルから P を信じる人のサプライザルを引いた差である。この差は、Q が結果を過小評価する場所（$q(x) < p(x)$）で正となり、Q が過大評価する場所で負となる。KL ダイバージェンスは、この差を真の分布 P のもとで平均する。

P がサンプルと分子の両方を与えるため、この定義は非対称であり、一般に $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$ である。$p(x) = 0$ となる結果は、規約 $0\log(0/q) = 0$ に従い何も寄与しない。しかし、$p(x) > 0$ かつ $q(x) = 0$ となる結果が 1 つでもあれば、$D_{\mathrm{KL}}(P\|Q) = +\infty$ となる。

小さな数値例を見れば定義が具体的になるため、本節の残りではこの例を引き続き用いる。3 つの結果上の真の分布を $P = (0.6, 0.3, 0.1)$、モデルを $Q = (0.2, 0.5, 0.3)$ とする。このとき次が得られる。

$$D_{\mathrm{KL}}(P\|Q) = 0.6\ln\frac{0.6}{0.2} + 0.3\ln\frac{0.3}{0.5} + 0.1\ln\frac{0.1}{0.3} = 0.659 - 0.153 - 0.110 = 0.396 \text{ nats},$$

一方、逆方向は $D_{\mathrm{KL}}(Q\|P) = 0.365$ nat となる。ビット単位では、2 つの方向はそれぞれ 0.571 と 0.527 である。両者の値が異なるだけでなく、1 つ目の値からは、Q を信じる質問者が 1 回の結果あたり必要より 0.57 回多く質問してしまうことがわかる（P から得られる結果の長い列で平均した場合）。

以下は離散 KL ダイバージェンスのコードであり、`entropy` と同様に $0\log 0 = 0$ を P の確率 0 の要素をマスクして除外することで実装している。通し例である対に対して両方向で評価するとともに、3 番目の結果を起こり得ないとするモデル $Q' = (0.5, 0.5, 0)$ に対しても評価する。

```
def kl_divergence(p, q):
    """KL(P || Q) for two probability vectors, in nats; inf where q = 0 < p."""
    p, q = np.asarray(p, dtype=float), np.asarray(q, dtype=float)
    m = p > 0
    if np.any(q[m] == 0):
        return float('inf')
    return float(np.dot(p[m], np.log(p[m] / q[m])))

p = np.array([0.6, 0.3, 0.1])
q = np.array([0.2, 0.5, 0.3])
```

```
print(f'KL(P||Q) = {kl_divergence(p, q):.4f} nats, '
      f'KL(Q||P) = {kl_divergence(q, p):.4f} nats')
print(f"KL(P||Q') = {kl_divergence(p, [0.5, 0.5, 0.0])}")
```

```
KL(P||Q) = 0.3961 nats, KL(Q||P) = 0.3653 nats
KL(P||Q') = inf
```

特に覚えておくべきは 2 行目の結果である。モデル Q' は 10 回に 1 回起こる結果に確率 0 を割り当てており、その KL ダイバージェンスは無限大になる。ある事象を不可能だと断言するモデルは、その事象が実際に起きたとき無限の驚きを被るのである。

37.1.2.2 ギブスの不等式

平均的な余分なサプライザルが負になることはあり得るだろうか。もしあり得るなら、誤ったモデルの方が真の分布そのものよりも平均して驚かないことになる。ギブスの不等式はそれを退ける (Gibbs 1902)。平均において真の分布を上回るモデルは存在しない。以降の議論はすべてこの結果に依拠するため、凹関数の平均の帰結としてその完全な証明を示そう。任意の凹関数 f と有限の平均を持つ任意の確率変数 X に対して、**イェンセンの不等式**（Jensen's inequality）は次のことを主張する。

$$\mathbb{E}[f(X)] \le f(\mathbb{E}[X]).$$

言葉で言えば、先に平均をとってから f を適用した方が大きな値になる。X が 2 つの値をとるとき、これは単に Figure 37.2 において弦が曲線の下側にあるという事実にほかならない。f が狭義凹である場合、等号が成り立つのは X が確率 1 で定数となるときに限られる（一般の場合の証明は 35.3.3.1 節にある）。ここでイェンセンの不等式を必要とするのは、狭義凹関数である対数関数ただ 1 つに対してである。すなわち、平均の対数は対数の平均以上である。

命題（ギブスの不等式）：同じ空間上の任意の分布 P と Q に対して、次が成り立つ。

$$D_{\mathrm{KL}}(P\|Q) \ge 0,$$

等号が成り立つのは $P = Q$ のとき、かつそのときに限る。

証明：$q(x) = 0$ となる x が $p(x) > 0$ の範囲に存在すれば、ダイバージェンスは $+\infty$ となり証明すべきことはないため、そうでないと仮定する。ダイバージェンスを $\log(q(x)/p(x))$ の $x \sim P$ のもとでの平均にマイナスを付けた形に書き換え、対数関数にイェンセンの不等式を適用する。

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= -\mathbb{E}_{x\sim P}\left[\log \frac{q(x)}{p(x)}\right] \ge -\log \mathbb{E}_{x\sim P}\left[\frac{q(x)}{p(x)}\right] \\ &= -\log \sum_{x\,:\,p(x)>0} q(x) = -\log Q(\operatorname{supp} P) \ge 0. \end{aligned}$$

先頭のマイナス符号によって、イェンセンの不等式の「高々」が「少なくとも」に反転する。証明の残りは 2 行目で完結する。期待値は P の台（support）、すなわち $p(x) > 0$ を満たす結果の集合の上でとられる。因子 $p(x)$ が約分されて消え、その台の上での Q による確率の総和が残る。この値は高々 1 であるため、その負の対数は非負となる。連続密度に対しても、和を積分に置き換えれば同じ手順が成り立つ。等号が成り立つには、2 つの不等号がともに等号でなければなら

ない。対数関数は狭義凹であるため、1 つ目の不等号が等号になるのは、$q(x)/p(x)$ が定数 c に、P に関してほとんどすべての x で（すなわち P の確率が 0 の集合を除いて）等しい場合に限られ、2 つ目の等号には $Q(\operatorname{supp} P) = 1$ が必要となる。そこで $q = c\,p$ を P の台の上で足し合わせる（または積分する）と $c = 1$ が強制され、したがって（密度の場合はやはり確率 0 の集合を除いて）$q = p$ となる。$\blacksquare$

この証明はただ 1 つの着想に基づいている。真の分布のもとで平均すると、比 q/p は高々 1 であり、したがってその対数の平均は高々 0 となり、等号成立は q/p がまったく変動しない場合（すなわち $q = p$）に限られる。モデルが余分なサプライザルを一切生まないのは、モデルが正しい場合だけである。

ギブスの不等式により、証明を残していた一様分布の上界も直ちに示される。任意の分布 P（k 個の結果上）と同じ集合上の一様分布 U に対して $D_{\mathrm{KL}}(P\|U) = \sum_i p_i \log(k\,p_i) = \log k - H(P)$ が成り立ち、これは非負であって $P = U$ のときに限り 0 になる。

また、条件付けによって平均的なエントロピーが増えることはあり得ないことも証明できる。対 (X, Y) の同時分布と、その周辺分布（marginal distribution）の積を比較しよう。対数比は $\log \frac{p(x,y)}{p(x)\,p(y)} = \log p(y \mid x) - \log p(y)$ であり、同時分布のもとでのその期待値は $H(Y) - H(Y \mid X)$ となる。この期待値は KL ダイバージェンスであるため、次が成り立つ。

$$H(Y) - H(Y \mid X) = D_{\mathrm{KL}}(p_{X,Y} \,\|\, p_X\, p_Y) \geq 0,$$

等号が成り立つのは、X と Y が独立なとき、かつそのときに限る。平均的には、X を知ることで Y がより不確実になることは決してない。ただし、**個別の**観測では不確実性が増すこともあり得る。Y は、$X = 0$ のとき確実に裏となり、$X = 1$ のとき公正なコインになるとし、$X = 1$ は 5 回に 1 回の割合で起こるとしよう。全体として、Y が表になる確率は 0.1 であるため、$H(Y) = H_b(0.1) \approx 0.325$ nat である。$X = 1$ を観測すると Y に関する不確実性は $\ln 2 \approx 0.693$ nat へと上昇するが、条件付きエントロピーは X の両方の結果にわたって平均をとるため、$H(Y \mid X) = 0.2 \cdot 0.693 \approx 0.139$ nat と、$H(Y)$ の半分未満になる。

37.1.2.3 交差エントロピー

分類器の訓練には、KL ダイバージェンスと密接に関係する量が用いられる。**交差エントロピー**（P から Q への交差エントロピー）は、Q を信じる人が P からサンプリングされる結果に対して経験する平均サプライザルであり、次の式で定義される。

$$\mathrm{CE}(P, Q) = -\mathbb{E}_{x \sim P}[\log q(x)]. \tag{37.4}$$

期待値の中で $\log p(x)$ を足して引くと（これは $H(P)$ が有限である限り正当化される）、モデルのサプライザルは真の分布自身のエントロピーと過剰なダイバージェンスの和に分解される。

$$\mathrm{CE}(P, Q) = H(P) + D_{\mathrm{KL}}(P\|Q). \tag{37.5}$$

したがって交差エントロピーは、削減不能なエントロピーの下限と非負の過剰分の和である。ギブスの不等式により $\mathrm{CE}(P, Q) \geq H(P)$ が成り立ち、等号成立は $Q = P$ のとき、かつそのときに限る。通し例の対では、$\mathrm{CE}(P, Q) = -(0.6 \ln 0.2 + 0.3 \ln 0.5 + 0.1 \ln 0.3) = 1.294$ nat、$H(P) = 0.898$ nat であり、両者の差 0.396 nat は先ほど計算した KL ダイバージェンスに一致する。$H(P)$ は Q に依存しないため、交差エントロピーを Q に関して最小化することは KL ダイバージェンスを最小化することと同値である。離散データに対する最尤法はまさにこれを行っ

ており、その平均負の対数尤度（negative log-likelihood）は経験データ分布からモデルへの交差エントロピーに等しい（36.6.2.1 節）。

Figure 37.3 は、2 進情報源 P と一様モデル $U = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ に対する質問ゲームでこの分解を図示したものである。P に適合した質問を行えば、平均して $H_2(P) = 1.75$ 回の質問で済む。一様な戦略のもとで固定した 2 問を尋ねると $\mathrm{CE}_2(P, U) = 2$ 回かかるため、無駄は $D_{\mathrm{KL}}(P \| U) = 0.25$ ビットとなる。質問戦略は 2 元符号であるため、これらの数値はそのまま 1 記号あたりのビット単位の平均符号語長でもある（なお、この特定の対では逆方向のダイバージェンス $D_{\mathrm{KL}}(U \| P)$ もたまたま 0.25 ビットに等しくなるため、先ほど KL ダイバージェンスの非対称性を示す際には別の対を用いた）。

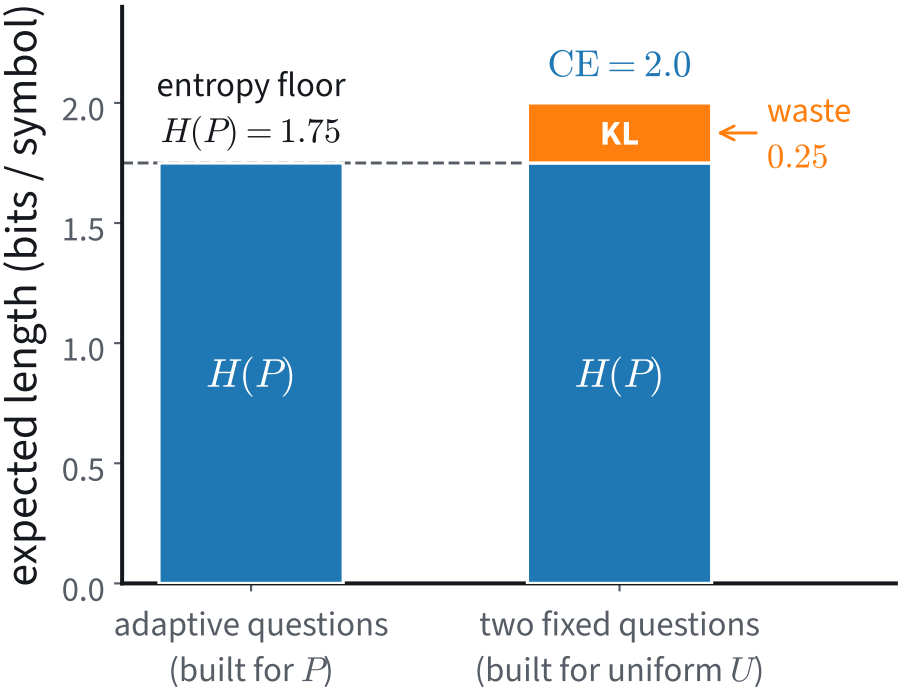


Figure 37.3: 分解 Equation 37.5 を 2 進情報源 $P = (\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ と一様モデル U に対してビット単位で表した図。P に適合した質問では平均 $H_2(P) = 1.75$ 問、固定した 2 問では $\mathrm{CE}_2(P, U) = 2.0$ 問を要し、無駄は $D_{\mathrm{KL}}(P \| U) = 0.25$ ビットである。

37.1.2.4 KL ダイバージェンスの連鎖律

エントロピーは対に沿って分解される。すなわち、(X, Y) の驚きは X の驚きと Y に残る驚きの和である（37.1.1.3 節）。KL ダイバージェンスもまったく同じ理由で同様に分解される。$P_{X,Y}$ と $Q_{X,Y}$ を 2 つの同時分布とし、それぞれ $p(x, y) = p(x)\, p(y \mid x)$ および $q(x, y) = q(x)\, q(y \mid x)$ と分解する。このとき比 $p(x, y)/q(x, y)$ の対数は 2 項の和となり、$P_{X,Y}$ のもとで期待値をとると KL ダイバージェンスの**連鎖律**が得られる。

$$D_{\mathrm{KL}}(P_{X,Y} \| Q_{X,Y}) = D_{\mathrm{KL}}(P_X \| Q_X) + \mathbb{E}_{x \sim P_X}\left[D_{\mathrm{KL}}(P_{Y|x} \| Q_{Y|x})\right]. \tag{37.6}$$

対に関する誤ったモデルは、まず X についての誤差の代償を払い、次に Y について（X が既知

となったもとでの）誤差の代償を平均的に払う。なお、この平均はモデルではなく真の分布 P_X にわたってとられる。この恒等式自体は 1 行の式変形で得られるが、本書の以降の議論はその 3 つの帰結に大きく依拠している。

第 1 に、KL ダイバージェンスは独立な座標にわたって加法的に足し合わされる。X と Y が P と Q の両方で独立であれば、条件付きの項はもはや x に依存せず、対のダイバージェンスは 2 つのダイバージェンスの和になる。したがって、通し例の対から独立に 10 回サンプルを引くと、P と Q は $10 \cdot 0.396 = 3.96$ nat だけ引き離される。長いサンプルの方が短いサンプルよりも 2 つの分布をはるかに明確に識別できるのはこのためである。第 2 に、条件付きの項はギブスの不等式により非負であるため、2 つの同時分布間のダイバージェンスは少なくともそれらの周辺分布間のダイバージェンス以上になる。変数を捨てることは、2 つの分布の識別をより難しくする方向にしか働かない。潜在変数モデル（latent variable model）はこの性質を利用している。データと潜在変数にわたる同時分布のダイバージェンスは定数差を除けば計算可能な目的関数であり、それはデータ単独のダイバージェンスを条件付き項の分だけ上回る。この差がエビデンス下界（evidence lower bound）のギャップである（36.7 節）。第 3 に、この規則を系列に対して繰り返し適用すると次が得られる。

$$D_{\mathrm{KL}}(P_{X_1,\dots,X_n} \| Q_{X_1,\dots,X_n}) = \sum_{i=1}^{n} \mathbb{E}_{x_{<i} \sim P}\big[D_{\mathrm{KL}}(P_{X_i \mid x_{<i}} \| Q_{X_i \mid x_{<i}})\big],$$

したがって、系列全体に対する自己回帰モデル（autoregressive model）の過剰損失は、個々のステップにおける過剰損失の総和となる。それゆえ、次の各トークンをうまく予測することは系列全体をうまくモデル化することと同じ課題であり、37.2.4.1 節ではこの分解を用いてあらゆる言語モデルの損失を下から抑える。

37.1.3 分類損失

k クラス分類問題において、1 つの例に対するターゲットは記録されたラベル y にすべての質量を置くワンホット分布であり、モデルは予測確率分布 $\mathbf{q} = (q_1, \dots, q_k)$ を出力する。両者の間の交差エントロピー Equation 37.4 は単一の項 $-\log q_y$ へと縮約され、これは観測されたラベルに対するモデルのサプライザルそのものである。n 個の例（ラベル y_i と予測 $\mathbf{q}_i$）にわたって平均をとると、次の**交差エントロピー損失**が得られる。

$$\mathrm{CE} = -\frac{1}{n} \sum_{i=1}^{n} \log q_{i,y_i},$$

単位は nat である。深層学習フレームワークは、この損失を**ロジット z**（ソフトマックスをとると **q** になる非正規化スコア）から直接計算する。先に確率を計算すると大きなロジットでオーバーフローし、非常に小さな負のロジットで $\log 0$ へとアンダーフローするため、内部では数値的に安定な対数ソフトマックス（log-softmax）を適用している（3.3.5.2 節、安定な形式の導出は 35.8.2 節を参照）。続く各項では、この損失からデータ中のノイズ、確率の較正、およびソフト化したターゲットについて何がわかるかを解き明かす。

37.1.3.1 削減不能な損失

本節の冒頭で、一部のラベルが誤っているときに損失はどこまで下がり得るかと問いかけた。固定された訓練セット上では、その答えから有益な知見は得られない。データセットを丸暗記できるほど大きなニューラルネットワークであれば、誤ったものも含めて記録されたすべてのラベルに適合し、訓練損失を 0 に近づけられるからである。この問いが実質的な意味を持つのはモデル

がまだ見ていない未知の例に対してのみであり、そこではワンホットターゲットは真実のすべてではない。実データでは、ラベルが入力の決定論的な関数であることはまれであり、本質的な曖昧さやアノテーション誤りのために同じ画像に異なるラベルが付くことがある。入力 $\mathbf{x}$ に対する真のターゲットは、そのラベルが抽出される条件付き分布であり、記録されたラベルはそこからの 1 つのサンプルにすぎない。

入力 $\mathbf{x}$ とラベル $y \in \{1, \dots, k\}$ が同時分布に従い、$p(\cdot \mid \mathbf{x})$ がラベルの真の条件付き分布、$q(\cdot \mid \mathbf{x})$ が任意のモデルの予測であるとする。未知の例の数が増えるにつれ、その平均損失は $(\mathbf{x}, y)$ にわたる期待値へと収束する。これを**母集団損失**（population loss）と呼ぶ。いかなるモデルも、この母集団損失を入力が与えられたもとでのラベルの平均エントロピーより下に押し下げることはできない。

命題（削減不能な分類損失）：任意のモデルの母集団交差エントロピー損失は次を満たす。

$$\begin{aligned} \mathbb{E}_{(\mathbf{x},y)}\big[-\log q(y \mid \mathbf{x})\big] &= \mathbb{E}_{\mathbf{x}}\big[H(p(\cdot \mid \mathbf{x}))\big] + \mathbb{E}_{\mathbf{x}}\big[D_{\mathrm{KL}}(p(\cdot \mid \mathbf{x}) \,\|\, q(\cdot \mid \mathbf{x}))\big] \\ &\geq \mathbb{E}_{\mathbf{x}}\big[H(p(\cdot \mid \mathbf{x}))\big], \end{aligned} \tag{37.7}$$

等号が成り立つのは、$q(\cdot \mid \mathbf{x}) = p(\cdot \mid \mathbf{x})$ がほとんどすべての $\mathbf{x}$ に対して（すなわち確率 0 の集合を除くすべての入力に対して）成り立つとき、かつそのときに限る。

証明： $\mathbf{x}$ で条件付ける。y にわたる内側の期待値は交差エントロピー $\mathrm{CE}(p(\cdot \mid \mathbf{x}), q(\cdot \mid \mathbf{x}))$ であるため、分解 Equation 37.5 が各 $\mathbf{x}$ において点ごとに適用できる（すべてのエントロピーは高々 $\log k$ で有限である）。外側の期待値を $\mathbf{x}$ にわたってとれば等式が得られる。ギブスの不等式によりすべての KL 項は非負であり、非負の項の平均が 0 になるのは、そのほとんどすべてが 0 であるとき、かつそのときに限る。■

この下限は、入力が与えられたもとでのラベルの条件付きエントロピーである（37.1.1.3 節）。これはデータ固有の性質であり、いかなるアーキテクチャ、追加の訓練データ、あるいはより長い訓練スケジュールによっても下げることはできない。本節のすべての結果のなかで、損失曲線を読むたびに念頭に置くべきなのはこの事実である。以下の 3 つの事例が、その意味するところを具体的に示している。

- 一様分布を予測するモデルは、ラベルにかかわらずすべての入力で $\ln k$ の交差エントロピーを被り、おなじみの 2.30 nat（$k = 10$ のとき）を与える。
- ラベルが入力の決定論的な関数であるとき、すべての $H(p(\cdot \mid \mathbf{x}))$ は 0 となり、完璧なモデルは損失 0 を達成する。
- 対称なラベルノイズのもとでは、クリーンなクラスは入力によって定まる一方、記録されるラベルは確率 $1-\rho$ でクリーンなクラスと一致し、残りの $k-1$ 個の各クラスにはそれぞれ確率 $\rho/(k-1)$ で割り振られる。どの入力においても、条件付きエントロピーは $H = -(1-\rho)\log(1-\rho) - \rho \log \frac{\rho}{k-1} = H_b(\rho) + \rho \log(k-1)$ となる。$\rho = 0.1$ かつ $k = 10$ のとき、この下限は $0.325 + 0.220 = 0.545$ nat である。$1-\rho > \rho/(k-1)$ である限りクリーンなクラスを予測することが最適であり続けるため、達成可能な最高の正解率は $1-\rho$（ここでは 90%）となる。

次のセルを実行する前に少し立ち止まり、このノイズのある設定でクリーンなクラスに確率 0.99 を割り当てる確信度の高い予測器の損失を予想してみてほしい。この予測器は 10 回中 9 回は正解し、しかも毎回ほぼ確信している。以下のコードでは、これを一様予測器、および真の条件付き分布 $q = p$ を出力する**ベイズ予測器**（Bayes predictor）と比較する。

```
k, rho = 10, 0.1
p_noisy = np.array([1 - rho] + [rho / (k - 1)] * (k - 1))  # clean class first
```

```
    predictors = {'uniform': np.full(k, 1 / k),
                  'confident 0.99': np.array([0.99] + [0.01 / (k - 1)] * (k - 1)),
                  'Bayes q = p': p_noisy}
    print(f'floor = H(p) = {entropy(p_noisy):.3f} nats')
    for name, q_hat in predictors.items():
        ce = float(-np.dot(p_noisy, np.log(q_hat)))
        print(f'{name:15s} CE = {ce:.3f} nats')
```

```
floor = H(p) = 0.545 nats
uniform         CE = 2.303 nats
confident 0.99  CE = 0.689 nats
Bayes q = p     CE = 0.545 nats
```

過信した予測器は 0.689 の損失を被り、下限を大きく上回る。反転したラベルのたびに不意を突かれ、その高い確信度が反転した 10 分の 1 のラベルで支払うコストは、残りのラベルで得る利益をはるかに上回るからである。下限に到達するのはベイズ予測器だけである。

これで冒頭の問いに答えられる。最終的な損失 0.6 は良い値なのだろうか。下限が 0 であるクリーンなラベルに対しては、0.6 にはまだモデルを改善する余地が十分に残っている。しかし、このようなノイズのあるラベルに対しては下限から 0.06 nat 以内に位置しており、改善の余地はほとんど残っていない。下限でのプラトーは単にデータの性質であり、下限より上でのプラトーだけがモデルまたはその訓練の限界を示す。なお、以上の議論はすべて未知のデータに対するものであることを忘れないでほしい。ホールドアウト分割において下限を**下回る**損失が出た場合、それは優れたモデルによるものではあり得ない。それが代わりに何を意味するかは演習問題 2 で問う。

ここまでは、単一のペナルティ（$-\log q_y$）、そのままのワンホットラベル、および可算個の結果に対して損失の下限を導出した。本節の残りでは、これら 3 つの仮定を順に緩め、他のペナルティ、ソフト化したターゲット、そして連続な実数値の結果を調べていく。

37.1.3.2 固有スコアリングルール

損失の下限が真の条件付き確率によってのみ達成されるという性質は、分類器の出力を確率として解釈したい場合にまさに望まれるものである。決定理論ではこの性質に名前が付けられている。**スコアリングルール**は、ペナルティ $S(\mathbf{q}, y)$ を、確率ベクトル $\mathbf{q}$ を報告して実際にクラス y が生じた場合に対して課す規則である。正直な報告が最適であるとき、すなわちラベルが $\mathbf{p}$ に従うもとで期待ペナルティ $\mathbb{E}_{y\sim\mathbf{p}}[S(\mathbf{q}, y)]$ が $\mathbf{q} = \mathbf{p}$ を報告することで最小化されるとき、そのルールは**固有**（proper）であるという。さらに、この正直な報告が唯一の最小点であるとき、**狭義固有**（strictly proper）であるという。交差エントロピー損失は**対数スコア**（log score）$S(\mathbf{q}, y) = -\log q_y$ であり、ギブスの不等式から狭義固有であることがわかる。その期待ペナルティは $\mathrm{CE}(\mathbf{p}, \mathbf{q})$ であり、$\mathbf{q} = \mathbf{p}$ で一意に最小化されるからである。

その結果、真の分布を含むほど十分に表現力の高いモデルクラス上で母集団最適解に達すれば、交差エントロピーで訓練された分類器は真の条件付き確率を出力する。とりわけ、その予測は**較正**（calibration）されている。すなわち、モデルがあるクラスに対して確率 0.8 と予測したすべての入力のうち、実際にそのクラスが生じる割合は 80% になる（ただし実際には、有限の訓練セットや制限されたモデルクラスのために、訓練済みニューラルネットワークの較正がずれることは依然として起こり得る）。

狭義固有なスコアリングルールは対数スコアだけではない。**ブライアスコア**（Brier score）$S(\mathbf{q}, y) = \sum_j (q_j - \mathbf{1}(j = y))^2$ も狭義固有であり、対数スコアとは異なって、$-\log q_y$ が発散するような確信度の高い誤りに対しても有界なペナルティしか課さない (Gneiting and Raftery

2007)。演習問題 3 ではその過剰ペナルティを計算し、一見もっともらしいが実は固有でないルールについても検討する。

37.1.3.3 ラベル平滑化

ここまで、訓練ターゲットはワンホットラベルであった。これをよりソフトなターゲットに置き換えると、損失がロジットに要求するものも損失の下限の位置も変化するが、そうすることには実用上の理由がある。ワンホットターゲットは、ソフトマックス層が実際には出力できないものを要求する。真のクラスに確率 1 を割り当てるには、そのロジットが他のすべてのロジットより無限に大きくなければならないからである。その結果、損失は有限の最小点を持たず、訓練が続く限り勾配降下法はロジット間の差を広げ続ける。**ラベル平滑化**（label smoothing）(Szegedy et al. 2016) は、ワンホットターゲット $\mathbf{e}_y$ に比率 ϵ で全 k クラス上の一様分布を混合することでこれを解消する。

$$\mathbf{p}^{\epsilon} = (1-\epsilon)\,\mathbf{e}_y + \frac{\epsilon}{k}\,\mathbf{1}. \tag{37.8}$$

このターゲットのもとでは、真のクラスには確率 $1-\epsilon+\epsilon/k$ が、他のすべてのクラスには ϵ/k が割り当てられる。訓練の目的関数は、この平滑化されたターゲットに対する通常の交差エントロピー $\mathrm{CE}(\mathbf{p}^{\epsilon}, \mathbf{q})$ である（訓練レシピを扱う 7.6 節で用いたものと同じ損失である）。これは $\mathbf{q} = \mathbf{p}^{\epsilon}$ で最小化され、ソフトマックスは真のクラスのロジットと他の各クラスのロジットとの間に有限のギャップを持たせることでこれを達成する。

$$z_y - z_j = \log \frac{1-\epsilon+\epsilon/k}{\epsilon/k} = \log\!\Big(1 + \frac{k(1-\epsilon)}{\epsilon}\Big).$$

$k = 10$ クラスかつ $\epsilon = 0.1$ のとき、この最適なロジットギャップは $\ln 91 \approx 4.51$ であり、最適点での損失は平滑化されたターゲットのエントロピー $H(\mathbf{p}^{\epsilon}) = 0.500$ nat に等しい。

平滑化を行わないときにロジットがどれほどの速さでドリフトするかを見ておくのは有益である。ロジットが 0 から始まり、ステップサイズ 1 の勾配降下法で更新される 1 つの例を考えよう。ロジットに関する交差エントロピーの勾配は $\mathbf{q} - \mathbf{p}^{\epsilon}$ である。ワンホットターゲットでは、誤ったクラスはすべて同値のまま保たれ、各ステップで真のクラスと残りのクラスとのギャップ g は $(1-q_y) + q_j = k/(e^{g} + k - 1)$ だけ広がる。この増分は約 ke^{-g} となる（g が大きくなったとき）。g を ke^{-g} だけ増やすと、e^{g} には $e^{ke^{-g}} \approx 1 + ke^{-g}$ が掛かり、その結果、各ステップで約 k が e^{g} に加算される。したがって t ステップ後には $e^{g} \approx kt$、すなわち $g \approx \ln(kt)$ になると予測される。ギャップの増大は決して止まらないが、その成長は対数的であり、ステップ数を 10 倍にしても $\ln 10 \approx 2.30$ しか増えない。10,000 ステップ後には $g \approx \ln 10^{5} \approx 11.51$ となるはずである。次のセルでは、平滑化なしと平滑化ありの両方でこの勾配降下法を実行する。

```
k, checkpoints = 10, (10, 100, 1000, 10000)
for eps in (0.0, 0.1):
    p_eps = np.full(k, eps / k)
    p_eps[0] += 1 - eps                  # the true class first
    z = np.zeros(k)                      # the logits of one example
    for step in range(1, checkpoints[-1] + 1):
        q = np.exp(z - z.max())
        z -= q / q.sum() - p_eps         # gradient step, step size 1
        if step in checkpoints:
            log_q = z - z.max() - np.log(np.exp(z - z.max()).sum())
```

```
            print(f'eps = {eps}  step {step:5d}  gap = {z[0] - z[1]:5.2f}  '
                  f'loss = {-p_eps @ log_q:.4f}')
    print(f'optimum for eps = 0.1: gap = ln 91 = {math.log(91):.2f}, '
          f'loss = H(p_eps) = {entropy(p_eps):.4f}')
```

```
eps = 0.0  step    10  gap =  4.29  loss = 0.1158
eps = 0.0  step   100  gap =  6.87  loss = 0.0093
eps = 0.0  step  1000  gap =  9.21  loss = 0.0009
eps = 0.0  step 10000  gap = 11.51  loss = 0.0001
eps = 0.1  step    10  gap =  3.81  loss = 0.5243
eps = 0.1  step   100  gap =  4.51  loss = 0.5003
eps = 0.1  step  1000  gap =  4.51  loss = 0.5003
eps = 0.1  step 10000  gap =  4.51  loss = 0.5003
optimum for eps = 0.1: gap = ln 91 = 4.51, loss = H(p_eps) = 0.5003
```

ワンホットターゲットでは、ロジットギャップは 4.29、6.87、9.21、11.51 に達する。近似式 $\ln(kt)$ の予測値は 4.61、6.91、9.21、11.51 であり、実際のギャップは（ke^{-g} の近似がまだ粗い）初期段階では予測よりわずかに小さいものの、$1{,}000$ ステップ以降は小数第 2 位まで一致する。その間、損失は決して 0 に到達することなく少しずつ下がり続ける。訓練セットに適合できるほど大きなニューラルネットワークでは、この際限のないドリフトは、成長し続ける重みと、誤ラベルの例も含めたすべての訓練例に対する 1 付近の予測確率として現れる。対照的に $\epsilon = 0.1$ では、ギャップは 4.51 に落ち着いて（ステップ 100 までに到達）そこに留まり、損失も 0.500 で静止する。

これで訓練は有限の最適解を持つようになるが、平滑化を用いたモデルの解釈には注意が必要である。平滑化されたターゲットを誤差なく予測できたとしても、訓練損失は 0 ではなく $H(\mathbf{p}^{\epsilon})$ でプラトーに達する。さらに、その出力はもはや真の条件付き確率ではなく、どのクラスも $1 - \epsilon + \epsilon/k$（ここでは 0.91）を超える確率を受け取ることはない。実務では $\epsilon = 0.1$ が標準的な選択肢であり、7.6 節の訓練レシピでもこれを採用している。演習問題 4 では、ラベル平滑化が誤ったクラスのロジットにどのような影響を与えるかを調べる。

37.1.3.4 知識蒸留

ソフトなターゲットは、別の訓練済みモデルから得ることもできる。**知識蒸留**（knowledge distillation）(Hinton et al. 2015) は、より大きな**教師**（teacher）ネットワークを模倣するようにコンパクトな**生徒**（student）ネットワークを訓練する手法であり、26.1 節（英語版） では言語モデルに適用した。生徒はワンホットラベルで訓練する代わりに、クラス全体にわたる教師の確率分布そのものをターゲットとして訓練する。手書きの「2」が「0」よりも「7」にはるかに似ているという事実は、ワンホットラベルが捨ててしまう情報であるが、教師の誤クラスに対するロジットの中には残っている（ラベル平滑化で訓練された教師がこの情報をあまり伝えられない理由は、演習問題 4 で問う）。

任意の固定されたターゲット分布の場合と同様、損失は生徒がターゲット自体を再現したときに最小化されるため、分解 Equation 37.5 からここで新たに付け加わることはない。説明を要するのは損失のスケールであり、そこには一見すると恣意的に見える係数が掛かっている。

よく訓練された教師のソフトマックス出力はほぼワンホットであるため、誤クラスに割り当てられる確率は通常きわめて小さく、生徒を十分に引き寄せる力を持たない。そこでまず、ソフトマックスを適用する前にロジットを**温度**（temperature）$\tau > 0$ で割ることで、両方の分布を**ソフト化**する。

$$p_j^{(\tau)} = \mathrm{softmax}(\mathbf{z}^{\mathrm{tea}}/\tau)_j, \qquad q_j^{(\tau)} = \mathrm{softmax}(\mathbf{z}^{\mathrm{stu}}/\tau)_j.$$

$\tau > 1$ に設定すると、ロジット間のギャップが圧縮され、確率質量が誤クラス側へと移る。蒸留損失は、こうしてソフト化された分布間の KL ダイバージェンスに τ^2 を掛けたものである。

$$\mathcal{L}_{\text{distill}} = \tau^2 \, D_{\mathrm{KL}}\big(\mathbf{p}^{(\tau)} \,\|\, \mathbf{q}^{(\tau)}\big). \tag{37.9}$$

この損失が 0 になるのは、生徒がソフト化された教師と一致するとき、かつそのときに限る。なぜ係数 τ^2 が必要なのかを見るために、スケール前の KL 項を生徒のロジットに関して微分してみよう。生徒に依存するのは交差エントロピーの部分 $-\sum_j p_j^{(\tau)} \log q_j^{(\tau)}$ だけであり、温度 τ における対数ソフトマックスの導関数は、おなじみの「ソフトマックス引くターゲット」の式を $1/\tau$ 倍したものになる。

$$\frac{\partial}{\partial z_j^{\text{stu}}} D_{\mathrm{KL}}\big(\mathbf{p}^{(\tau)} \| \mathbf{q}^{(\tau)}\big) = \frac{1}{\tau}\big(q_j^{(\tau)} - p_j^{(\tau)}\big).$$

したがって、生徒の各ロジットは、そのクラスに対する生徒と教師のソフト化確率の差を τ で割った分だけ動かされる。

高温では、その確率の差自体も縮小する。平均ロジット $\bar{z}$ を引いてもソフトマックスは変化せず、大きな τ に対しては一次のテイラー展開により $e^{(z_j - \bar{z})/\tau} \approx 1 + (z_j - \bar{z})/\tau$ と近似できる。中心化された偏差 $z_j - \bar{z}$ の和は 0 であるため、ソフトマックスの分母は約 k となり、$\mathrm{softmax}(\mathbf{z}/\tau)_j \approx \frac{1}{k}\big(1 + (z_j - \bar{z})/\tau\big)$ が得られる。その結果、$q_j^{(\tau)} - p_j^{(\tau)}$ は $1/\tau$ のオーダーで縮小し、スケール前の勾配は $1/\tau^2$ のオーダーで縮小する。

$$\frac{1}{\tau}\big(q_j^{(\tau)} - p_j^{(\tau)}\big) \;\approx\; \frac{1}{k\,\tau^2}\Big(\big(z_j^{\text{stu}} - \bar{z}^{\text{stu}}\big) - \big(z_j^{\text{tea}} - \bar{z}^{\text{tea}}\big)\Big).$$

損失に τ^2 を掛けることでこの減衰が打ち消され、τ を変化させても勾配ノルムはほぼ一定に保たれる。蒸留損失が単独で使われることはまれであり、通常はハードラベルに対する標準的な交差エントロピー損失に固定の重みで加算されるため、この安定性は実務上不可欠である。τ^2 の係数がなければ、τ を調整した際に意図せず 2 つの損失の相対的な重みまで変わってしまう。この係数があることで、温度は教師の小さな誤クラス確率を生徒にどれだけ見せるかだけを制御できる。

教師ロジット $(5, 2, -1)$ と生徒ロジット $(3, 3, 0)$ を用いて、このスケーリングを数値的に確認しよう。どちらのベクトルも平均が 2 であるため、勾配の τ^2 倍は $\frac{1}{3}\big((1,1,-2) - (3,0,-3)\big) = (-0.667, 0.333, 0.333)$（ノルム 0.816）に収束するはずである。下のセルでは、自動微分を用いて 5 つの温度にわたってスケール前の KL 項の勾配を計算し、τ^2 を掛ける前後のノルムを比較する。

```
z_tea = jnp.array([5.0, 2.0, -1.0])
z_stu = jnp.array([3.0, 3.0, 0.0])

def distill_kl(z_stu, z_tea, tau):
    p_tau = jax.nn.softmax(z_tea / tau)
    return (p_tau * (jnp.log(p_tau) - jax.nn.log_softmax(z_stu / tau))).sum()

print(' tau   |grad|  |tau^2 grad|  tau^2 grad')
for tau in (1.0, 2.0, 5.0, 10.0, 20.0):
    g = np.asarray(jax.grad(distill_kl)(z_stu, z_tea, tau))
    print(f'{tau:4.0f}  {np.linalg.norm(g):.4f}  '
```

```
            f'{np.linalg.norm(tau**2 * g):12.3f}  {np.round(tau**2 * g, 3)}')
    limit = (z_stu - z_stu.mean()) - (z_tea - z_tea.mean())
    limit = np.asarray(limit) / len(limit)
    print(f'limit {"":6s}  {np.linalg.norm(limit):12.3f}  {np.round(limit, 3)}')
```

```
 tau  |grad|  |tau^2 grad|  tau^2 grad
   1  0.6391         0.639  [-0.462  0.441  0.022]
   2  0.2190         0.876  [-0.672  0.549  0.123]
   5  0.0368         0.920  [-0.741  0.478  0.263]
  10  0.0088         0.884  [-0.719  0.413  0.306]
  20  0.0021         0.854  [-0.697  0.374  0.322]
limit                0.816  [-0.667  0.333  0.333]
```

τ^2 の係数がないと勾配は崩壊する。そのノルムは 0.64（$\tau = 1$）から 0.0021（$\tau = 20$）へと約 300 分の 1 に落ち込み、最後に τ を 2 倍にしたときにはちょうど $1/\tau^2$ が予測するとおり約 4 分の 1 になっている。一方、係数を含めればノルムは 0.64 から 0.92 の間に留まる。漸近極限の 0.816 をわずかにオーバーシュートして $\tau = 5$ 付近でピークに達した後、$1/\tau$ のオーダーの速さでゆっくりと極限へ戻っていく（0.854 が $\tau = 20$ での値である）。また、$\tau = 1$ では損失が通常の KL ダイバージェンスに帰着し、さらにワンホットな教師（エントロピーが 0）に対しては通常の交差エントロピー損失になるため、標準的な教師あり訓練はその特別な場合となっていることにも注目したい。

37.1.4 確率密度に対するエントロピーと KL ダイバージェンス

ここまではすべての結果が離散かつ可算であった。対照的に、回帰目標は確率密度によって記述される実数であるため、和は積分に置き換わる。KL ダイバージェンスはこの移行後もそのままの性質を保つが、エントロピーはそうではない。連続なエントロピーは負の値になり得るうえ、変数を測定する単位に依存する。その結果、連続モデルの負の対数尤度損失は 0 を下回ることがあり、データをリスケーリングするたびにその下限もシフトする。絶対的な意味を保ち続けるのは下限からの過剰分であり、それに伴って同じデータ上で評価した任意の 2 つのモデル間の損失の差である（初読時には、続く 3 つの項を飛ばして本節の最後の段落から再開してもよい）。

37.1.4.1 微分エントロピー

連続確率変数 X（密度 p）に対し、離散エントロピーの和を積分に置き換えると**微分エントロピー**（differential entropy）が得られる。

$$h(X) = -\int p(x) \log p(x)\, dx. \tag{37.10}$$

これは負の対数密度の期待値であるが、密度の値は確率ではない。密度は $1/x$ の単位を持ち、容易に 1 を超え得る。微分エントロピーには小文字の h を用いて離散の H と区別するが、それは両者の振る舞いが大きく異なるからである。

最も単純な例からすでにその理由がわかる。区間 $[0, a]$ 上の一様密度は区間全体で $1/a$ に等しいため、$h = -\int_0^a \frac{1}{a} \log \frac{1}{a}\, dx = \log a$ となる。$a = \frac{1}{2}$ のとき、これは $-\ln 2 \approx -0.693$ nat となり、いかなる離散分布もとり得ない負のエントロピーを与える。さらに厄介なことに、まったく同じ長さを m ではなく cm で測ると、a が 100 倍になり、背後にある不確実性は一切変わっていないにもかかわらず $\ln 100 \approx 4.61$ nat が h に加算される。

これと同種のシフトは、任意の滑らかな変数変換のもとで生じる。$y = g(x)$ が可逆で導関数 g' を持つならば、Y の点 $g(x)$ における密度は $p_X(x)/|g'(x)|$ であり、したがって次が成り立つ。

$$h(Y) = h(X) + \mathbb{E}\big[\log|g'(X)|\big].$$

確率ベクトル $\mathbf{x} \in \mathbb{R}^n$ に対しても、積分領域を $\mathbb{R}^n$ とし、ヤコビ行列式 $|\det \partial g/\partial \mathbf{x}|$ で $|g'|$ を置き換えれば、定義もこの変換則もまったく同じ形を保つ。変数を引き伸ばせば微分エントロピーは増加し、圧縮すればエントロピーは際限なく下がっていく。37.1.4.3 節で導出するように、標準偏差 σ のガウス分布の微分エントロピーは $h = \tfrac{1}{2}\log(2\pi e\sigma^2)$ であり、$\sigma < (2\pi e)^{-1/2} \approx 0.24$ になった途端に負の値となる。

幸いなことに、KL ダイバージェンスはこのような単位への依存性を持たない。データ密度 p とモデル密度 q の両方に同じ変数変換を適用すると、両方とも同じヤコビ行列式の因子で割られるため、比 p/q の中でそれが相殺されるからである。したがって、KL ダイバージェンスは可逆な再パラメータ化に対して不変である。たとえば、2 つの一様密度 p_1（区間 $[0, a_1]$ 上）と p_2（区間 $[0, a_2]$ 上、$a_1 \le a_2$）に対しては $\int p_1 \log(p_1/p_2)\,dx = \log(a_2/a_1)$ となり、$a_1 = \tfrac{1}{2}$、$a_2 = 1$ であれば、m 単位で測っても cm 単位で測っても $\ln 2 \approx 0.693$ nat である。連続モデルの訓練に用いる交差エントロピー $-\mathbb{E}_P[\log q]$ は h と同じ定数だけシフトするが、その定数はモデルに依存しないため、勾配も最小点も変化しない。$h(p)$ が有限である限り、分解 Equation 37.5 は h で H を置き換えた形でそのまま成り立つため、密度に対する最尤法は母集団レベルで $D_{\mathrm{KL}}(p\|q_\theta)$ を最小化する。

また、微分エントロピーは単に細かいビニングのもとでの離散エントロピーの極限ではないことにも注意したい。実数直線を幅 Δ のセルに分割し、X_Δ を X がどのセルに含まれるかを記録する変数とする。点 x_i（セル i の内部の点）に対して $P(X_\Delta = i) \approx p(x_i)\Delta$ となり、次の極限が成り立つ。

$$H(X_\Delta) + \log\Delta \to h(X) \quad \text{as } \Delta \to 0$$

これは、リーマン和が積分に収束するほど p が十分に正則であるとき（たとえば p が連続であり、p と $p\log p$ の両方がリーマン可積分であるとき）にいつでも成り立つ。離散エントロピー $H(X_\Delta)$ 自体は $\log(1/\Delta)$ のように発散する。グリッドを細かくするほど、常により多くの「はい・いいえ」の質問が必要になるからである。そして $h(X)$ は、その増大分を差し引いた後に残る有限のオフセットにあたる。区間 $[0,1]$ 上の一様密度に対して $\Delta = \tfrac{1}{8}$ とすると、8 つのセルはすべて等確率となるため $H_2(X_\Delta) = 3$ ビット、$\log_2\Delta = -3$ であり、その和は $0 = h$ となる。演習問題 6 ではこの極限を証明し、ガウス分布上で数値的に検証する。

37.1.4.2 ガウス分布間の KL ダイバージェンス

回帰モデルが実数値とその不確実性をあわせて予測するとき、最もよく出力されるのは平均と分散からなるガウス分布である。そのようなモデルをデータと比較したり、2 つのガウスモデルどうしを比較したりするには、2 つのガウス分布間の KL ダイバージェンスが必要になるが、これには簡潔な閉形式がある。$P = \mathcal{N}(\mu_1, \sigma_1^2)$ と $Q = \mathcal{N}(\mu_2, \sigma_2^2)$ に対して、次が成り立つ。

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu_1,\sigma_1^2)\,\big\|\,\mathcal{N}(\mu_2,\sigma_2^2)\big) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \tag{37.11}$$

この公式は、標準偏差の対数比と、P の分散および平均の二乗誤差（いずれもモデルの分散 σ_2^2 との相対値として測られる）を組み合わせたものである。ギブスの不等式が保証するとおり、この

値は非負であり、$\mu_1 = \mu_2$ かつ $\sigma_1 = \sigma_2$ のときに限り 0 になる。また、σ_1 と σ_2 が異なる形で現れていることから一目でわかるとおり非対称である。演習問題 7 では定義からこの式を直接導出する。

最も単純な場合として、分散が等しく平均が 1 だけシフトした $P = \mathcal{N}(0, 1)$ と $Q = \mathcal{N}(1, 1)$ から始めよう。このとき公式はどちらの方向でも $(\mu_1 - \mu_2)^2/(2\sigma^2) = \frac{1}{2}$ nat に帰着する。次に、代わりに分散が異なる場合として $P = \mathcal{N}(0, 1)$ と $Q = \mathcal{N}(0, 4)$（したがって $\sigma_2 = 2$）をとろう。式に代入すると $D_{\mathrm{KL}}(P\|Q) = \ln 2 + \frac{1}{8} - \frac{1}{2} \approx 0.318$ nat となるのに対し、逆方向は $D_{\mathrm{KL}}(Q\|P) = -\ln 2 + 2 - \frac{1}{2} \approx 0.807$ nat と、2.5 倍以上の大きさになる。

Figure 37.4 を見ると、それぞれのダイバージェンスがどこから生じているかがわかる。どちらの方向でも、被積分関数は 2 つの密度が交差する $|x| = (8 \ln 2/3)^{1/2} \approx 1.36$ で符号を変える。パネル (a) では、幅の広いモデルが幅の狭いデータを説明しており、中心部（$|x| < 1.36$、寄与は $+0.424$）で過剰に驚く一方、裾野では真の分布よりわずかに驚きが小さい（寄与は -0.106）。パネル (b) では、幅の狭いモデルが幅の広いデータを説明しなければならず、中心部の寄与は -0.240 である一方、実際に生じる結果に幅の狭いモデルがごくわずかな確率しか割り当てない裾野が $+1.046$ という大きな寄与をもたらす。

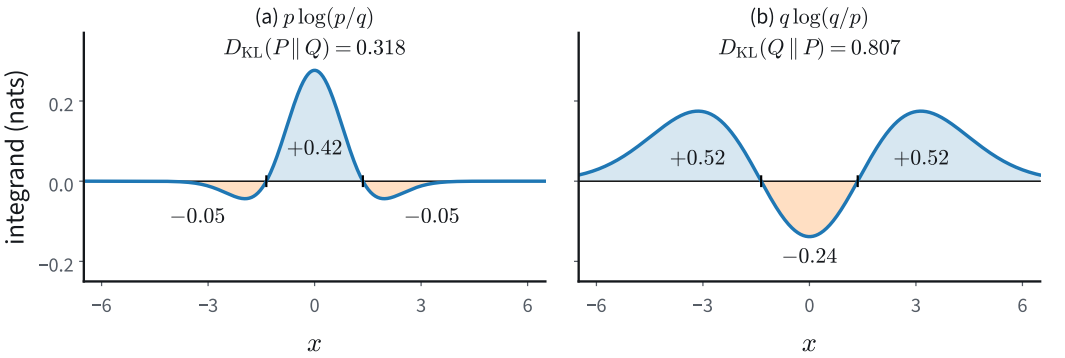


Figure 37.4: $\mathcal{N}(0, 1)$ と $\mathcal{N}(0, 4)$ の間の KL ダイバージェンスの内訳。(a) 幅の広いモデルで幅の狭い分布を説明する場合：正の中心部とわずかに負の裾野の和が 0.318 nat になる。(b) 幅の狭いモデルで幅の広い分布を説明する場合：幅の狭いモデルがほとんど確率を割り当てない裾野が 1.05 nat 寄与し、中心部の -0.24 と合わせて合計 0.807 になる。どちらの方向でも $|x| = 1.36$ で符号が変わる。

幅の狭いモデルが幅の広いデータから受ける驚きは、幅の広いモデルが幅の狭いデータから受ける驚きよりもはるかに大きい。一般に KL ダイバージェンスは、モデルがほぼ除外していた結果が実際に起こるたびに厳しいペナルティを課す一方、データが現れない領域にモデルが余分な確率を広げているだけの場合には穏やかな値にとどまる。この非対称性が、生成モデルを一方の方向で当てはめる場合と逆方向で当てはめる場合とで何を意味するかは、37.3.2 節の主題である。

また、この定義は閉形式が存在しない場合に KL ダイバージェンスを推定する一般的な方法も示唆している。すなわち、P からサンプルを引き、$\log p(x) - \log q(x)$ の平均をとればよい。後の節でサンプルに基づく推定値を用いるため、このガウス分布の対でどれほどの精度が出るか確認しておこう。次のセルでは、閉形式とモンテカルロ推定量およびその標準誤差を実装する。

```
def gaussian_kl(mu1, sigma1, mu2, sigma2):
    """Closed-form KL(N(mu1, sigma1^2) || N(mu2, sigma2^2)), in nats."""
```

```
        return (math.log(sigma2 / sigma1)
                + (sigma1 ** 2 + (mu1 - mu2) ** 2) / (2 * sigma2 ** 2)
                - 0.5)

    def gaussian_log_pdf(x, mu, sigma):
        """Log-density of N(mu, sigma^2) at x."""
        return (-0.5 * math.log(2 * math.pi * sigma ** 2)
                - (x - mu) ** 2 / (2 * sigma ** 2))

    def mc_kl(mu1, sigma1, mu2, sigma2, n=200000):
        """Monte Carlo estimate of the same KL from n samples x ~ P, with its
        standard error."""
        x = np.random.default_rng().normal(loc=mu1, scale=sigma1, size=n)
        r = gaussian_log_pdf(x, mu1, sigma1) - gaussian_log_pdf(x, mu2, sigma2)
        return float(r.mean()), float(r.std() / math.sqrt(n))
```

平均シフトの場合については、推定量の精度を事前に予測できる。対数比は $\log p(x) - \log q(x) = \frac{1}{2} - x$ であり、分散は 1（P のもとで）であるため、$n = 2 \cdot 10^5$ 個のサンプルの平均は標準誤差 $1/\sqrt{n} = 0.0022$ を持つ。

しかし分散が異なる対では、2 つの方向の推定のしやすさは同じではない。符号を除けばどちらも同じ関数 $\ln 2 - \frac{3}{8}x^2$ の平均をとるが、一方は幅の狭いガウス分布からサンプリングするのに対し、もう一方は x^2 が 4 倍大きく変動する幅の広いガウス分布からサンプリングする。3 つのダイバージェンスをすべて推定して確かめてみよう。

```
pairs = {'N(0,1) || N(1,1)': (0.0, 1.0, 1.0, 1.0),
         'N(0,1) || N(0,4)': (0.0, 1.0, 0.0, 2.0),
         'N(0,4) || N(0,1)': (0.0, 2.0, 0.0, 1.0)}
print(f'{"P || Q":18s} {"closed":>7s} {"sampled":>8s} {"std err":>8s}')
for name, args in pairs.items():
    estimate, std_err = mc_kl(*args)
    print(f'{name:18s} {gaussian_kl(*args):7.4f} {estimate:8.4f} {std_err:8.4f}')
```

```
P || Q             closed  sampled  std err
N(0,1) || N(1,1)   0.5000   0.5007   0.0022
N(0,1) || N(0,4)   0.3181   0.3183   0.0012
N(0,4) || N(0,1)   0.8069   0.8066   0.0047
```

標準誤差はそれぞれ 0.0022、0.0012、0.0047 となり、いずれの推定値も閉形式から標準誤差の数倍以内に収まっている。幅の広いガウス分布からサンプリングする方向は逆方向よりノイズが 4 倍大きく、同じ精度に達するには 16 倍のサンプル数を要することに注目しよう。Figure 37.4 のパネル (b) がその理由を説明している。このダイバージェンスの大半は、対数比が大きくサンプルがまばらな裾野から生じているためである。これと同じサンプリングの難しさには、37.7 節ではるかに大きなスケールで再び出会うことになる。

37.1.4.3 最大エントロピーとガウス分布

有限集合上では、一様分布が不確実性を最大化する。実数直線上では、どの密度が同じ役割を果たすだろうか。何の制約も置かなければ、そのような密度は存在しない。$\mathbb{R}$ 全体で一様な密度は存在せず、密度を引き伸ばせば微分エントロピーは際限なく大きくなるからである。そこで代わりに、ガウスモデルが出力する 2 つのモーメント、すなわち平均 μ と分散 σ^2 を固定してみよう。これら 2 つのモーメントを持つすべての密度のなかで、ガウス分布が最大の微分エントロピーを

持つ。この厳密な意味において、ガウスノイズのモデルは最も仮定の少ないモデルである。平均と分散だけを定め、それ以外には何ひとつ仮定を置かない。

証明は、一様分布の上界を示した 1 行の議論とほぼ同じであり、ひねりが 1 つだけ加わる。離散の場合には $-\log u(x) = \log k$ がすべての結果で一定であったため、どの分布 P も一様モデルに対して同じ交差エントロピー $\log k$ を持っていた。ここでは φ を平均 μ、分散 σ^2 のガウス密度とする。そのサプライザル $-\log \varphi(x) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{(x-\mu)^2}{2\sigma^2}$ はもはや定数ではないが、x の二次関数であり、二次関数の期待値は P に依存するものの、その依存は固定した 2 つのモーメントだけを通じたものになる。任意の密度 P（平均 μ、分散 σ^2）に対して、次が成り立つ。

$$\mathrm{CE}(P, \varphi) = \tfrac{1}{2}\log(2\pi\sigma^2) + \frac{\mathbb{E}_P\big[(x-\mu)^2\big]}{2\sigma^2} = \tfrac{1}{2}\log(2\pi\sigma^2) + \tfrac{1}{2} = \tfrac{1}{2}\log(2\pi e\sigma^2).$$

注目すべきことに、φ に対する交差エントロピーは、そのような P のどれがデータを生成したかに依存しない。$P = \varphi$ と置けば交差エントロピーはエントロピーに帰着するため、この定数はガウス分布自身の微分エントロピー、すなわち先ほど引用した閉形式にほかならない。同じモーメントを持つ他の任意の密度 P に対しては、分解 Equation 37.5（h で H を置き換えたもの）とギブスの不等式から次が得られる。

$$h(P) = \mathrm{CE}(P, \varphi) - D_{\mathrm{KL}}(P \| \varphi) \le \tfrac{1}{2}\log(2\pi e\sigma^2) = h(\mathcal{N}(\mu, \sigma^2)), \tag{37.12}$$

等号が成り立つのは $P = \mathcal{N}(\mu, \sigma^2)$ のとき、かつそのときに限る（$h(P) = -\infty$ の場合、不等式は自明に成り立つ）。この上界とのギャップ $D_{\mathrm{KL}}(P\|\varphi)$ にも直接的な解釈がある。すなわち、データがガウス分布でないときに最良のガウスモデルが被る過剰損失である（演習問題 8）。

一様分布とガウス分布は、単一の原理の 2 つの具体例である。候補となる密度のサプライザルが、制約を満たすすべての分布のもとで同じ定数に平均されるのであれば、その候補は制約を満たすすべての分布のなかでエントロピーを最大化する。36.4.2 節では任意の統計量の期待値に関する制約に対してこの結果を完全な一般性のもとで確立しており、そこでエントロピーを最大化する分布は指数分布族（exponential family）をなす。非負整数 $\{0, 1, 2, \ldots\}$ 上の幾何分布（geometric distribution）は 3 つ目の例である。そのサプライザルは結果に関してアフィンであるため、与えられた平均を持つ非負整数上のすべての分布のなかでエントロピーを最大化する。

一歩引いて、まだ未解決のまま残っている問いを確認しよう。先ほど、離散モデル Q' は実際に生じる結果に確率 0 を割り当てたために KL ダイバージェンスが無限大になった。Figure 37.4 の幅の狭いガウス分布もそれに近く、ほぼ除外していた裾野でダイバージェンスの大半を積み上げていた。少数の乱数を画像へと写す生成モデルは、ピクセル空間内の薄い曲面上にすべての確率質量を置く。もし実データがその曲面上にちょうど乗っていなければ、曲面がデータのすぐ隣を通っていようとはるか遠くに離れていようと、KL ダイバージェンスは無限大になる。37.3 節では密度比 p/q から構築されるあらゆるダイバージェンスがこの盲点を持つことを示し、37.4 節ではそのような状況でも遠近を識別できる比較尺度を展開する。2 つ目の問いはさらに身近なところにある。先ほど「はい・いいえ」の質問の回数を数えてそれを 2 元符号語として読み替えたが、それはサプライザルが整数のビット数になる 2 進情報源に対してだけであった。37.2 節では任意の情報源に対する符号を構成し、実際の圧縮器がどこまでエントロピーに迫れるかを問う。

37.1.5 まとめ

交差エントロピー損失は、データに対するモデルの平均サプライザルを測る。この損失は、いかなるモデルも下げられないデータのエントロピーと、データからモデルへの KL ダイバージェンスの和に分解される。ギブスの不等式により、後者は決して負にならず、真の分布に対してのみ

0 になる。したがって、いかなる損失の値も、その削減不能な下限である「入力が与えられたもとでのラベルの条件付きエントロピー」に照らして判断しなければならない。未知のデータ上では、真の条件付き確率だけがこの下限に到達できる。この下限は、入力がラベルを一意に定める場合には 0 となり、割合 ρ の対称なラベルノイズのもとでは $H_b(\rho) + \rho \log(k-1)$ となる。

ビット単位では、有限情報源のエントロピーは結果を特定するのに必要な「はい・いいえ」の質問の平均回数（1 問以内の誤差を除く）に等しく、高々 $\log_2 k$（k 個の結果に対して）である。ターゲットをソフト化すると最適解が移動する。ラベル平滑化は正の損失下限を伴う有限の最適ロジットギャップを生み出し、知識蒸留はソフト化された教師に一致させつつ、係数 τ^2 によって温度が変化しても勾配のスケールを一定に保つ。連続密度に対しては、微分エントロピーは負になり得るうえ測定単位によってシフトするが、KL ダイバージェンスと交差エントロピーの最小点はいずれも不変に保たれる。

37.1.6 演習問題

1. **自己情報量の一意性。**●自己情報量がスケールを除いて一意であることを示せ。すなわち、狭義減少関数 $I(p)$（区間 $(0,1]$ 上）が $I(1) = 0$、および $I(p_1 p_2) = I(p_1) + I(p_2)$ をすべての $p_1, p_2 \in (0,1]$ に対して満たすならば、$I(p) = -c \log p$ がある定数 $c > 0$ に対して成り立つことを示せ。関数 $g(t) = I(e^{-t})$（$t \geq 0$）を用いて議論せよ。

2. [code] **ラベルノイズの下限。** k クラス上のラベルが、入力とは独立に確率 $1-\rho$ で正しく、そうでなければ残りの $k-1$ クラス上で一様に振られるとする。

   1. クリーンなラベルが入力の関数であると仮定する。任意の分類器の母集団交差エントロピーが少なくとも $H_b(\rho) + \rho \log(k-1)$ であること、最適な予測はノイズを含む条件付き分布そのものであること、そして最適な正解率は $1-\rho$ になる（$1-\rho > \rho/(k-1)$ のとき）ことを示せ。
   2. $k = 10$ および $\rho = 0.05, 0.1, 0.2$ に対してこの下限を計算せよ。
   3. すべての分割にこのようなノイズが 10% 含まれることがわかっている 10 クラスのデータセットで、同僚が 0.40 nat の検証損失を報告した。考えられる説明を列挙し、そのうちどれがこの下限によって棄却され、どれが棄却されないかを述べよ。
   4. 3.3 節のソフトマックス回帰を、訓練ラベルとテストラベルの 10% をこの方法で反転させた Fashion-MNIST 上で訓練せよ。テスト損失を 0.545 と、テスト正解率を 90% と比較せよ。ギャップは大きいはずである。Fashion-MNIST は (a) のどの仮定に反しているか、線形モデル固有の誤差は何を上乗せしているか、そして実際に適用される下限はどのように推定できるか。

3. **その他のスコアリングルール。** スコアリングルール $S(\mathbf{q}, y)$ は、期待ペナルティ $\mathbb{E}_{y \sim \mathbf{p}}[S(\mathbf{q}, y)]$ が $\mathbf{q} = \mathbf{p}$ で一意に最小化されるとき狭義固有である。対数スコアでは、この性質はギブスの不等式そのものである。

   1. $S(\mathbf{q}, y) = \sum_j (q_j - \mathbf{1}(j = y))^2$ で定まるブライアスコアの期待値が、正直な報告に対するブライアスコアの期待値を $\|\mathbf{q} - \mathbf{p}\|^2$ だけ上回り、したがってブライアスコアが狭義固有であることを示せ (Gneiting and Raftery 2007)。
   2. 線形スコア $S(\mathbf{q}, y) = -q_y$ は、観測されたクラスに割り当てられた確率を高く評価する。どの報告がその期待ペナルティを最小化するか。線形スコアは固有か。ラベルがクラス 1 となる確率が 0.6、クラス 2 となる確率が 0.4 である入力に対して、線形スコアで訓練された分類器は何を報告するか。

4. **曖昧な入力に対するラベル平滑化。**●ラベル平滑化はターゲット Equation 37.8 を用いて訓練する。

   1. 訓練分布において、ある入力 $\mathbf{x}$ がラベル y を確率 $p(y \mid \mathbf{x})$ で持つとする。この入力に

対する平滑化損失の期待値が $\mathbf{q}^*(\cdot \mid \mathbf{x}) = (1-\epsilon)\, p(\cdot \mid \mathbf{x}) + \frac{\epsilon}{k}\mathbf{1}$ によって最小化されることを示せ。$p(\cdot \mid \mathbf{x})$ は $\mathbf{q}^*$ からどのように復元できるか。この最小点が 1 つのクラスに割り当て得る最大の確率はいくらか。

2. ロジット $\mathbf{z}$ と $\mathbf{q} = \mathrm{softmax}(\mathbf{z})$ を持つ 1 つの例に対し、$\mathrm{CE}(\mathbf{p}^\epsilon, \mathbf{q})$ の z_j に関する勾配は $q_j - p_j^\epsilon$ である。$\epsilon > 0$ のとき、すべての停留点において誤クラスのロジットがすべて等しくなることを示せ。
3. ワンホットターゲット（$\epsilon = 0$）では、誤クラスのロジットに関する勾配は q_j であり、損失は停留点を持たない。それでも 1 つの例のロジットに対する勾配降下法が 2 つの誤クラスロジット間のギャップを縮小させることを示し、この縮小がなぜ $q_y \to 1$ につれて停滞するのかを説明せよ。ニューラルネットワークでは、すべての例のロジットが共有された特徴から計算される。したがって $\epsilon = 0$ での訓練後にも生き残り得る誤クラスロジット間の差はどのようなものか。また、$\epsilon > 0$ で訓練するとなぜそれらが消えるのか。
4. Müller et al. (2019) は、ラベル平滑化で訓練した教師を用いると、蒸留損失 Equation 37.9 のもとでより性能の低い生徒が得られることを観察している。(b) と (c) の結果を用いてこの観察を説明せよ。

5. [code] **高温における蒸留。**●Hinton et al. (2015) はその 2.1 節において、高温では蒸留がゼロ平均ロジットの二乗誤差による一致に帰着すると述べている。

   1. $q_j = p_j(1+u_j)$（したがって $\sum_j p_j u_j = 0$）と置き、$D_{\mathrm{KL}}(\mathbf{p}\|\mathbf{q}) = -\sum_j p_j \log(1+u_j)$ を二次まで展開して次を導け。

$$D_{\mathrm{KL}}(\mathbf{p}\|\mathbf{q}) = \frac{1}{2}\sum_j \frac{(q_j - p_j)^2}{p_j} + O(\|\mathbf{q}-\mathbf{p}\|^3).$$

   この展開を、大きな τ に対する $\mathrm{softmax}(\mathbf{z}/\tau)$ の近似（37.1.3.4 節）と組み合わせ、$\tau \to \infty$ のとき次が成り立つことを示せ。

$$\tau^2 D_{\mathrm{KL}}(\mathbf{p}^{(\tau)} \,\|\, \mathbf{q}^{(\tau)}) \to \frac{1}{2k}\sum_{j=1}^{k}\Big((z_j^{\mathrm{stu}} - \bar{z}^{\mathrm{stu}}) - (z_j^{\mathrm{tea}} - \bar{z}^{\mathrm{tea}})\Big)^2.$$

   2. $\tau^2 D_{\mathrm{KL}}(\mathbf{p}^{(\tau)}\|\mathbf{q}^{(\tau)})$ を $\tau = 1, 2, 5, 10, 100$ に対して計算し（ロジットには $\mathtt{z_tea} = (5, 2, -1)$ と $\mathtt{z_stu} = (3, 3, 0)$ を 37.1.3.4 節と同様に用いる）、極限値と比較せよ。
   3. 低温では、教師のどのロジットが損失にほとんど影響しなくなるか。Hinton et al. (2015) は、これが利点にも欠点にもなり得ると論じている。教師自身の交差エントロピー訓練がこれらのロジットに何をもたらすかに基づき、双方の立場からそれぞれ 1 つずつ理由を挙げよ。

6. [code] **量子化エントロピーと微分エントロピーの比較。** X が密度 p を持ち、X_Δ がセル $[i\Delta, (i+1)\Delta)$ のどれに X が含まれるかを記録するとする。

   1. $H(X_\Delta) + \log\Delta \to h(X)$ が $\Delta \to 0$ で成り立つことを、p が連続であり p と $p\log p$ がリーマン可積分である場合について示せ。積分の平均値の定理により、$P(X_\Delta = i) = p(x_i)\Delta$（$x_i$ はセル内の点）と表して議論せよ。
   2. 区間 $[0, 1]$ 上の一様密度と $\Delta = \frac{1}{8}$ に対して手計算でこの極限を確認せよ。1 を割り切らないセル幅（たとえば $\Delta = 0.3$）ではどうなるか。

3. $H(X_\Delta) + \log \Delta$ を $X \sim \mathcal{N}(0,1)$ および $\Delta = 0.5, 0.1, 0.01$ に対して数値的に計算し（セルの確率にはガウス分布の分布関数を用いよ）、$h = \frac{1}{2}\log(2\pi e) \approx 1.4189$ nat と比較せよ（$\Delta = 0.5$ では 1.4292 が得られるはずである）。
4. なぜ $H(X_\Delta)$ 自体は $\Delta \to 0$ で発散するのか。回帰目標を幅 Δ のセルにビニングしてソフトマックス分類器で予測するとき、その損失は最良でもどの量に近づくか。また、連続モデルの負の対数密度と比較する前に何を足さなければならないか。

7. **ガウス分布の KL ダイバージェンス。** $P = \mathcal{N}(\mu_1, \sigma_1^2)$ とする。
   1. 閉形式 Equation 37.11 を、$\mathbb{E}_{x\sim P}[\log p(x) - \log q(x)]$ を書き下し、$\mathbb{E}_{x\sim P}[x] = \mu_1$ および $\mathbb{E}_{x\sim P}[(x-\mu_1)^2] = \sigma_1^2$ を用いることで導出せよ。
   2. Q が平均 $\mu_1 + \delta$、標準偏差 $\sigma_1 e^{\eta}$ を持つとする。$D_{\mathrm{KL}}(P\|Q)$ と $D_{\mathrm{KL}}(Q\|P)$ の両方を δ と η に関して二次まで展開せよ。2 つの方向が初めて異なるのは何次の項からか。
   3. $P = \mathcal{N}(0,1)$ と $Q = \mathcal{N}(0,4)$ に対してこの二次近似を計算し、2 つの方向の厳密値 0.318 nat および 0.807 nat と比較せよ。

8. **ガウスモデルの過剰損失。** 密度 P が平均 μ、分散 σ^2、および有限の微分エントロピーを持つターゲットに対して、回帰モデルがガウス分布 $\mathcal{N}(m, s^2)$ を出力するとする。
   1. $\mathrm{CE}(P, \mathcal{N}(m,s^2))$ が $m = \mu$ かつ $s^2 = \sigma^2$ で最小になること、そしてこの最良のガウスモデルが下限 $h(P)$ を $D_{\mathrm{KL}}(P\|\varphi) = h(\varphi) - h(P)$ だけ上回ることを示せ（ここで φ は $\mathcal{N}(\mu,\sigma^2)$ の密度である）。
   2. 区間 $[0, a]$ 上の一様密度（分散 $a^2/12$）、およびラプラス密度 $\frac{1}{2b}e^{-|x|/b}$（分散 $2b^2$、微分エントロピー $1 + \log(2b)$）に対してこの過剰損失を計算せよ（それぞれ約 0.176 nat と約 0.072 nat が得られるはずである）。
   3. いずれの答えもスケール a や b に依存しない。計算を行うことなく、37.1.4.1 節で述べた単位変換の性質からその理由を説明せよ。
   4. 与えられた分散に対してガウス分布は最大のエントロピーを持つため、それより高い下限を持つ密度は存在しない。では、過剰損失にも最大値は存在するか。2 点 $\mu \pm \sigma$ の近くに集中する密度を考えよ。この答えは、よく離れた 2 つのモードを持つターゲットに対するガウスモデルについて何を物語っているか。

37.2 符号、圧縮、パープレキシティ

37.1 節では、エントロピーと交差エントロピーを平均符号語長として解釈し、記号あたり x ビットの損失を達成するモデルであれば、テキストを記号あたり約 x ビットに圧縮できるはずだと示唆した。しかし、その主張を確かめたのはサプライザルがたまたま整数ビットになる都合のよい 1 つの情報源に対してだけであり、それ以外の分布に対する符号はまだ構成していなかった。そのような符号を実際に構築しない限り、対数損失（log-loss）とファイルサイズの結びつきは比喩の域を出ない。本節では、2 つの問いを立てることでこのギャップを埋める。モデルの対数損失に長さが一致する符号を構成できるのか、そして実用的な圧縮器はその限界にどこまで迫れるのか、という問いである。

日常的に使われる圧縮器にも、すでにその答えのヒントがある。8.2 節で扱った小説『The Time Machine』を、27 記号のアルファベット（英小文字 26 文字とスペース）からなる 173,428 文字に整形したものを考えよう。繰り返し現れる文字列を後方参照に置き換えてから各記号を個別に符号化する汎用圧縮器 `zlib` は、このテキストを文字あたり 2.84 ビットで保存する。これに対し、直前の 3 文字から各文字を予測し、読み進めながら出現回数を更新していく小さなカウントモデルは、同じテキストを文字あたり 2.55 ビットに圧縮する。予測が優れているほどファイルは短くなるのであり、本節ではその理由を解き明かす。符号によって対数損失がファイルサイズへと変換されれば、本章の冒頭で触れた文字あたり 1.2 nat といった損失は具体的な保存レートとなり、

2 つのモデルの比較はそれらが生成するファイルの長さを比べるのと同じくらい直接的なものになる。

この結果に向けて 4 つのステップで議論を進める。第 1 に、記号間に区切り文字のない 2 進ストリームを受信者が解釈しなければならないとき、符号語をどこまで短くできるかを問う。第 2 に、すべての符号語を整数ビットに制限することによるペナルティを測り、そのペナルティを記号ごとに支払わずに済むストリーム符号を構築する。第 3 に、各文字が過去に依存する実際のテキストへと目を向け、ファイルを理論上どこまで短くできるか、そしてモデルを学習するコストが全体の長さにどう組み込まれるかを見る。最後に、これらの符号長を、言語モデルがホールドアウトされたテキストに対して報告する評価指標であるパープレキシティと結びつける。本節を通じて、符号語が 2 進文字列であることから nat ではなくビットを単位として用いる。なお、x ビットは $x \ln 2$ nat に等しい。

本書の標準的なインポートに加え、テキストの前処理用に `re` を、ベースラインの圧縮器として `zlib` を用いる。

```
import re
import zlib
```

37.2.1 記号符号

最も単純な通信問題から始めよう。送信者は、分布 P（k 通りの結果上の分布）から独立に抽出された記号のストリームを観測し、記号間に区切り文字を入れない単一の 2 進文字列として送信しなければならない。一方、受信者は元の系列を一意に復元しなければならない。

37.2.1.1 接頭符号とクラフトの不等式

記号符号（symbol code）では、各結果 i に 2 進文字列である**符号語**（codeword）を長さ ℓ_i で割り当てる。区切り文字なしで、受信者はどうやって 1 つの符号語の終わりと次の符号語の始まりを見分ければよいのだろうか。最も単純な保証は、どの符号語も他のいかなる符号語の接頭辞にもならないようにすることである。このような符号は**プレフィックスフリー**、あるいは単に**接頭符号**（prefix code）と呼ばれる。電話番号も同じ規則に従っている。ダイヤルした数字の並びが有効な番号に一致した時点で、同じ並びから始まるより長い番号は存在しないため、交換機は番号の入力が完了したことを即座に判別できる。4 つの記号に対して符号 $0, 10, 110, 111$ はプレフィックスフリーであり、受信ストリーム `110010111` は `110|0|10|111` というただ 1 通りの方法でのみ区切ることができる。

接頭符号を二分木として可視化すると（Figure 37.5）、符号語長に対する制約が直ちに明らかになる。長さ ℓ の符号語は深さ ℓ の葉で終わり、その下にあるすべての子孫を除外するため、木全体の割合 $2^{-\ell}$ を占有する。異なる符号語どうしは重なり合えないため、それぞれが占有する割合は合計予算 1 の単一の木の中に収まらなければならず、それゆえ短い符号語は高くつく。図の長さ $(1, 2, 3, 3)$ は $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1$ を費やし、木全体を使い切っている。対照的に、長さ 1 の符号語を 4 つ用意するには予算が 2 必要になるが、長さ 1 の 2 進文字列は 2 つしか存在しないため不可能である。長さ $(1, 2, 4)$ は $\frac{1}{2} + \frac{1}{4} + \frac{1}{16} = 0.8125$ を費やすだけで、たとえば $0, 10, 1100$ のように余力を残して無理なく収まる。

同じ木は、37.1 節の「はい・いいえ」質問ゲームの戦略も表している。各内部ノードが二択の質問を投げかけ、各符号語はある記号に至るまでの回答の系列を記録する。「はい」を 0 とすると、符号 $0, 10, 110, 111$ は順に「1 番目の記号か」「2 番目の記号か」「3 番目の記号か」と尋ねることに対応する。前節では「はい」に 1 を用いたため、その鏡像である $1, 01, 001, 000$ が得られていた。

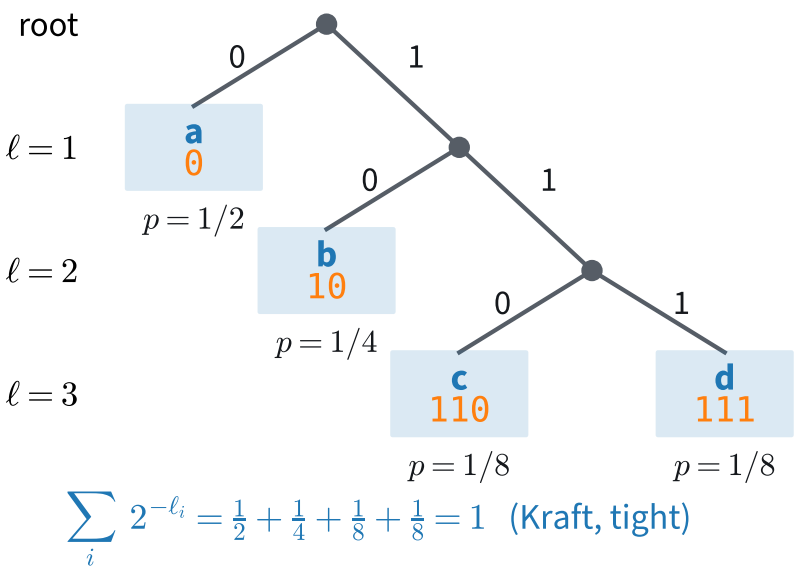


Figure 37.5: 二分木として表したプレフィックスフリー符号 $0, 10, 110, 111$。各符号語は根から葉への経路（左の辺が 0、右の辺が 1）であり、どの符号語も他の符号語を通過しない。深さ ℓ の葉は木全体の割合 $2^{-\ell}$ を占有し、ここでは 4 つの葉が $2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 1$ と予算全体を使い切っている。記号 a, b, c, d とその確率は、前節で扱った 2 進情報源のものである。

命題（クラフトの不等式） 符号語長 $\ell_1, \dots, \ell_k$ を持つプレフィックスフリーな 2 進符号は、次の不等式が成り立つとき、かつそのときに限り存在する。

$$\sum_{i=1}^{k} 2^{-\ell_i} \le 1. \tag{37.13}$$

証明 各 2 進符号語 c（長さ ℓ）に、半開区間 $[0.c,\ 0.c + 2^{-\ell})$、すなわち $[0, 1)$ 内の実数のうち 2 進展開が c から始まるもの全体からなる区間を対応させる。たとえば、符号語 10 は $[0.10_2, 0.11_2) = [\tfrac{1}{2}, \tfrac{3}{4})$ に対応し、その幅は 2^{-2} である。Figure 37.5 では、この区間は b の葉と、その下で取り除かれた部分木全体を表している。このような任意の 2 つの **2 進**（dyadic）区間は、一方が他方に含まれる（入れ子になる）か、互いに素であるかのいずれかである。もし c が c' の接頭辞であれば、c' から始まるすべての数は c からも始まるため区間は入れ子になり、そうでなければ c と c' は短い方の長さ以内のどこかの位置で異なるため、両方から始まる数は存在せず区間は互いに素になる。したがって、プレフィックスフリー符号は $[0, 1)$ 内の互いに素な区間に対応し、その合計幅 $\sum_i 2^{-\ell_i}$ は高々 1 である。

逆に、符号語長が Equation 37.13 を満たすとする。これらを $\ell_1 \le \cdots \le \ell_k$ と非減少順に並べ、幅 $2^{-\ell_i}$ の区間を 0 から順に隙間なく並べていく。i 番目の区間は $s_i = \sum_{m<i} 2^{-\ell_m}$ から始まり、$s_i + 2^{-\ell_i} \le 1$ で終わる。先行するすべての長さが $\ell_m \le \ell_i$ を満たすため、和の各項 $2^{-\ell_m}$（s_i を構成する項）は $2^{-\ell_i}$ の整数倍であり、したがって s_i 自身も $2^{-\ell_i}$ の整数倍となる。ゆえに i 番目の区間は、先頭 ℓ_i 桁の 2 進数字を s_i から取り出した符号語の 2 進区間である。これらの区間は互いに素であるため、証明の前半より、どの符号語も他の符号語の接頭辞にはならないことが保証される。■

注目すべきことに、クラフトの予算はプレフィックスフリー符号に固執したために生じる人工的な制約ではない。たとえ一部の符号語が他の符号語の接頭辞になっていても、符号語のあらゆる連結がただ 1 通りにしか区切れないとき、その符号は**一意復号可能**（uniquely decodable）であるという。McMillan は、すべての一意復号可能符号がやはり Equation 37.13 を満たさなければならないことを証明した (McMillan 1956)（演習問題 1）。したがって、一意復号可能符号で達成可能な符号語長の組はすべて接頭符号ですでに達成可能であり、考察の対象を接頭符号に限定しても何も失われない。

37.2.1.2 シャノン長とエントロピー限界

予算がわかったので、次はどの符号語長が期待符号長 $\mathbb{E}[\ell] = \sum_i p_i \ell_i$ を最小化するかを問うことができる。頻出する記号には明らかに短い符号語を割り当てるべきだが、長さ 1 の符号語を 1 つ割り当てるだけでクラフトの予算全体の半分を消費してしまう。

理想的なバランスを見つけるため、一時的に整数制約を緩和し、長さ ℓ_i を実数として扱ってみよう。どの符号語を短くしても $\mathbb{E}[\ell]$ は狭義に減少するため、予算を使い残すことが有利になることは決してなく、最適解は $\sum_i 2^{-\ell_i} = 1$ を満たさなければならない。等式制約付き最適化に対するラグランジュの未定乗数法（34.2.4 節）により、最小点では目的関数の勾配が制約関数の勾配に比例しなければならない。$\mathbb{E}[\ell]$ の ℓ_i に関する導関数は p_i であり、一方 $\sum_i 2^{-\ell_i}$ の導関数は $-2^{-\ell_i}\ln 2$ である。したがって、$2^{-\ell_i} = c\,p_i$ が単一の定数 c を用いてすべての i について成り立つ。両辺を i にわたって足し合わせると $c = 1$ が定まり、最適な実数符号語長とそれによる期待符号長が次のように得られる。

$$\ell_i = \log_2 \frac{1}{p_i}, \qquad \mathbb{E}[\ell] = \sum_i p_i \log_2 \frac{1}{p_i} = H_2(P).$$

各記号の理想的な長さは、単にその**サプライザル** $\log_2(1/p_i)$、すなわち 37.1 節の自己情報量をビット単位で測ったものにほかならず、その結果得られる期待符号長は 2 進エントロピー $H_2(P)$ である。もちろん、実際の符号語は整数の長さを持たなければならない。2 進情報源 $P = (\tfrac12, \tfrac14, \tfrac18, \tfrac18)$ の場合、サプライザル $1, 2, 3, 3$ はすでに整数であるため、Figure 37.5 の符号は無駄なく $\mathbb{E}[\ell] = 1.75$ ビット $= H_2(P)$ を達成する。本章で継続して用いる 2 つの分布の組のうち 1 つ目である $P = (0.6, 0.3, 0.1)$ の場合、サプライザルは 0.74、1.74、3.32 ビットとなる。それぞれを次の整数へ切り上げると**シャノン長**（Shannon lengths）$1, 2, 4$ が得られ、これは先ほど余力を残して二分木に収まった長さの組と同じである。そのクラフト和は 0.8125 であり、期待符号長は $\mathbb{E}[\ell] = 0.6 + 0.6 + 0.4 = 1.6$ ビットとなる（これに対し $H_2(P) = 1.295$ である）。

整数長をどれほど慎重に選んでも、エントロピーを下回る接頭符号は存在せず、一方でシャノン長がエントロピーを 1 ビット以上超えることも決してない。これら両方の上下界は、37.1 節で示した 1 つの恒等式、すなわち交差エントロピーの分解から導かれる。これをビット単位で改めて記すと次のようになる（添字の 2 と上付き添字の (2) は底 2 の対数を表す）。モデル Q が P から抽出されたデータに対して持つ期待対数損失は、P のエントロピーと、P から Q へのカルバック・ライブラー・ダイバージェンスとに分解される。

$$\mathrm{CE}_2(P, Q) = -\sum_i p_i \log_2 q_i = H_2(P) + D_{\mathrm{KL}}^{(2)}(P\|Q), \qquad D_{\mathrm{KL}}^{(2)}(P\|Q) \geq 0. \tag{37.14}$$

ここで KL ダイバージェンスの非負性はギブスの不等式（Gibbs' inequality）そのものである。いかなるモデルも情報源の真のエントロピーより低い期待対数損失を達成することはできず、

等号は $Q = P$ の場合に限り成り立つ。継続して用いている分布の組 $P = (0.6, 0.3, 0.1)$ と $Q = (0.2, 0.5, 0.3)$ では、これら 3 つの量は（小数最終桁の丸めの範囲で）$1.867 = 1.295 + 0.571$ ビットとなる。

命題（エントロピーによる符号長の上下界） P を、$p_i > 0$ がすべての i について成り立つ分布とする（確率 0 の記号には符号語は不要である）。すべてのプレフィックスフリーな 2 進符号は $\mathbb{E}[\ell] \geq H_2(P)$ を満たし、等号は $p_i = 2^{-\ell_i}$ がすべての i について成り立つとき、かつそのときに限り成立する。逆に、シャノン符号（Shannon code）（長さ $\ell_i = \lceil \log_2(1/p_i) \rceil$ を持つ符号）はプレフィックスフリーであり、$\mathbb{E}[\ell] < H_2(P) + 1$ を満たす。

証明 下界については、符号をそれが暗に定める確率分布と比較する。$Z = \sum_i 2^{-\ell_i}$ とおくと、クラフトの不等式より $Z \leq 1$ を満たすので、正規化された分布 $q_i = 2^{-\ell_i}/Z$ を定義する。ℓ_i について解くと $\ell_i = -\log_2 q_i - \log_2 Z$ となり、P のもとで期待値をとると次式が得られる。

$$\mathbb{E}[\ell] = \sum_i p_i \ell_i = -\sum_i p_i \log_2 q_i - \log_2 Z = H_2(P) + D_{\mathrm{KL}}^{(2)}(P \| Q) - \log_2 Z \geq H_2(P). \tag{37.15}$$

2 つの過剰項はいずれも非負である。すなわち、ギブスの不等式より $D_{\mathrm{KL}}^{(2)}(P\|Q) \geq 0$ であり、また $-\log_2 Z \geq 0$ も $Z \leq 1$ から成り立つ。等号成立には両方の項が 0 になる必要があり、それは $Q = P$ かつ $Z = 1$、すなわち $p_i = 2^{-\ell_i}$ がすべての i について成り立つことを意味する。上界については、シャノン長が $2^{-\ell_i} \leq p_i$ を満たすことから $\sum_i 2^{-\ell_i} \leq \sum_i p_i = 1$ となるため、クラフトの条件を満たし有効な接頭符号を定める。天井関数による切り上げが加える量は狭義に 1 ビット未満であるから $\ell_i < \log_2(1/p_i) + 1$ であり、P のもとで平均をとれば $\mathbb{E}[\ell] < H_2(P) + 1$ が得られる。$\blacksquare$

恒等式 Equation 37.15 は、それが確立する不等式以上に多くのことを教えてくれる。すなわち、接頭符号がビットを無駄にする原因は次の 2 通りしか存在しない。符号が暗に定める分布 Q が P と異なること、またはクラフトの予算の一部を未消費のまま残すこと（$Z < 1$）である。継続して用いている分布 P に対してシャノン符号は記号あたり $1.6 - 1.295 = 0.305$ ビットを無駄にするが、そのうち分布の不一致に由来するのはわずか 0.005 ビットであり、残りの 0.300 ビットは未消費の予算に由来する。

この命題の前半と後半を合わせると、いかなる記号符号の最適期待符号長も $H_2(P)$ ビットと $H_2(P) + 1$ ビットの間に挟まれる。これはまさに 37.1 節で予告した、「はい・いいえ」質問の期待回数に対する上下界である。エントロピーの下限が達成されるのは、すべての確率がちょうど $\frac{1}{2}$ のべき乗になっている場合に限られる。

より賢い記号符号を使えば、この 1 ビットのオーバーヘッドをどこまで取り戻せるだろうか。ハフマンのアルゴリズム（Huffman's algorithm）(Huffman 1952) は、最も確率の低い 2 つの記号を 1 つの合成記号へと繰り返し統合することで最適な接頭符号を構成する手法であり（演習問題 2）、`zlib` の内部で使われている記号符号器でもある。継続して用いている P に対して、ハフマンのアルゴリズムは長さ $1, 2, 2$ を選び、平均 1.4 ビット（エントロピーをわずか 0.1 ビット上回るだけ）を達成してシャノン符号の無駄の 3 分の 2 を解消する。しかし、最適な記号符号であっても避けられない壁に直面する。いかなる符号語も 1 ビットより短くはできないため、極端に偏った情報源ではこの 1 ビットの下限が節約分のほぼすべてを飲み込んでしまうのである。

`a` を確率 0.9 で、`b` を確率 0.1 で出力する情報源を考えよう。そのエントロピーは記号あたり 0.47 ビットであり、各 `a` が持つサプライザルはわずか $\log_2(1/0.9) = 0.15$ ビットにすぎない。それでも、いかなる記号符号もすべての `a` に少なくとも丸々 1 ビット、すなわちその情報量の 6 倍以上を費やさなければならない。もし `a` が確率 0.999 で生じるなら、エントロピーは記号あた

り 0.011 ビットまで下がるにもかかわらず、どの記号符号も依然として 1 ビットを費やす。この「$+1$」のペナルティは、記号ごとに 1 回丸めを行うコストである。一度に n 記号のブロックをまとめて符号化すれば、このコストを償却できる。n 記号のブロックをより大きなアルファベット上の単一の記号として扱えば、切り上げはブロックごとに 1 回で済み、記号あたりのオーバーヘッドは $1/n$ 未満に抑えられる。ハフマンのアルゴリズムで `a` と `b` の記号の**ペア**を符号化するだけでも、レートは元の記号あたり 1 ビットから直ちに 0.645 ビットへと下がる。

37.2.1.3 不一致符号と交差エントロピー

実際には、真のデータ分布 P が既知であることはまれであり、代わりにモデル Q から符号を構築する。Q に対するシャノン符号は長さ $\ell_i = \lceil \log_2(1/q_i) \rceil$ を割り当てるが、これには $q_i > 0$ が（P が出力し得るすべての記号について）成り立つ必要がある。この符号を P から実際に生成されたデータに用いるとき、その期待符号長は次式を満たす。

$$\mathrm{CE}_2(P, Q) \;\le\; \mathbb{E}_P[\ell] \;<\; \mathrm{CE}_2(P, Q) + 1, \tag{37.16}$$

これは $\log_2(1/q_i) \le \ell_i < \log_2(1/q_i) + 1$ がすべての記号 i について成り立ち、P のもとで平均をとっても両方の不等号が保たれるからである。したがって交差エントロピーとは、モデル Q のために設計された符号の期待符号長（ただし最大 1 ビットの丸め誤差を除く）にほかならず、Equation 37.14 により KL ダイバージェンスはモデル Q を真の分布 P の代わりに用いることで支払う正確なペナルティとなる。

これらの上下界を数値的に確かめてみよう。以下のセルでは、モデル `q` に対するシャノン符号を構成し、データ分布 `p` のもとでの期待符号長を計算して、クラフト和と交差エントロピーを並べて出力する。本節で扱った 2 つの情報源をそれぞれ一致符号と不一致符号のもとでテストする。すなわち、2 進情報源と一様モデルの組、および継続して用いている P と $Q = (0.2, 0.5, 0.3)$ の組である。

```
def shannon_lengths(q):
    return [math.ceil(math.log2(1 / q_i)) for q_i in q]

def code_report(name, p, q):
    """The Shannon code built from q, used on data from p (all in bits)."""
    ells = shannon_lengths(q)
    kraft = sum(2 ** -ell for ell in ells)
    avg = sum(p_i * ell for p_i, ell in zip(p, ells))
    ce2 = -sum(p_i * math.log2(q_i) for p_i, q_i in zip(p, q))
    print(f'{name:<16} lengths {ells}  Kraft {kraft:.4f}  '
          f'E[ell] {avg:.3f}  CE2 {ce2:.3f}')

dyadic, uniform = [1/2, 1/4, 1/8, 1/8], [1/4] * 4
p, q = [0.6, 0.3, 0.1], [0.2, 0.5, 0.3]
code_report('dyadic, own', dyadic, dyadic)
code_report('dyadic, uniform', dyadic, uniform)
code_report('P, own', p, p)
code_report('P, code from Q', p, q)
```

```
dyadic, own      lengths [1, 2, 3, 3]  Kraft 1.0000  E[ell] 1.750  CE2 1.750
dyadic, uniform  lengths [2, 2, 2, 2]  Kraft 1.0000  E[ell] 2.000  CE2 2.000
P, own           lengths [1, 2, 4]  Kraft 0.8125  E[ell] 1.600  CE2 1.295
P, code from Q   lengths [3, 1, 2]  Kraft 0.8750  E[ell] 2.300  CE2 1.867
```

分布が一致している 2 つの行は手計算の結果を再現している。ここでは CE2 がエントロピーに一致し、期待符号長はそれぞれ 1.75 ビットと 1.6 ビットである。不一致の行からは、誤ったモデルを使うコストが浮かび上がる。Q からシャノン符号を構成すると長さは $3, 1, 2$ となり、図らずも最長の符号語を P の最頻出記号に割り当ててしまう。P からのデータに対しては記号あたり平均 2.3 ビットとなり、区間 $1.867 \le 2.3 < 2.867$（Equation 37.16 による予測）のちょうど内側に収まる。一様符号は 2 進情報源のすべての記号に 2 ビットを割り当てる。丸めは一切生じないため下界が等号で成立し、$2.0 = 1.75 + 0.25$ ビットはエントロピーと 0.25 ビットの KL ダイバージェンスの和にきれいに分解される。最後に Kraft 列を見ると、各符号がどれだけの予算を未使用のまま残しているかがわかる。2 進情報源では未使用分は 0 であり、未使用分が 0.1875 となるのが P のシャノン符号、0.125 となるのが Q から構成した符号である。

1 つ微妙な点に注意しよう。不等式 Equation 37.16 は特に Q のシャノン符号に対して成り立つものであり、Q から構成し得るあらゆる符号に当てはまるわけではない。Q に対するハフマン符号は長さ $2, 1, 2$ を持ち、クラフトの予算をすべて使い切り、平均はわずか $1.2 + 0.3 + 0.2 = 1.7 < 1.867$ ビット（P からのデータに対して）にとどまる。ここに矛盾はない。この符号が暗に定める分布は $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ であって Q ではなく、その 2 進分布がたまたま P のより近くに位置しているのである。Equation 37.15 が示したとおり、符号の過剰な長さはそれが暗に定める分布と未消費の予算のみによって決まり、Q はそれが生み出す符号語長を通じてのみ影響する。

37.2.2 ストリーム符号と記述長

記号符号は、記号ごとに丸めのペナルティを支払う。次に、1 つのメッセージにつき 1 回だけ丸めを行う符号を構築して言語モデルに直接結びつけ、さらにモデル自体を記述するコストをどのように織り込むべきかを問う。

37.2.2.1 算術符号化

記号のブロックをまとめて符号化すれば記号ごとの丸め損失はなくなるが、すぐさま組み合わせ論の壁に突き当たる。100 個の 2 進記号からなるブロックの符号帳には 2^{100} 個もの項目が必要となり、到底保存しきれないからである。本当に欲しいのは、ルックアップテーブルを持たずにブロック符号の効率を実現する仕組み、すなわちメッセージ全体に対する単一の符号語をメッセージ自体から逐次的に計算する符号である。この種の符号は**ストリーム符号**（stream code）と呼ばれ、**算術符号化**（arithmetic coding）はその古典的な構成法である (Rissanen 1976; Witten et al. 1987)。符号語を表から引く代わりに、算術符号化はメッセージを $[0, 1)$ の部分区間として表現する。Figure 37.6 は、10 記号のメッセージ aaaaaaaaba（$q(\mathtt{a}) = 0.9$ の偏った情報源から生成されたもの）を真のモデルのもとで符号化する様子を示している。

エンコーダは現在の区間（初期値は $[0, 1)$）を保持する。次の記号 x_i を符号化する際、エンコーダは現在の区間を、モデルの条件付き確率 $q(\cdot \mid x_{<i})$ に比例する幅で取り得る記号ごとのスライスに分割し、実際に生じた記号に対応するスライスだけを残す。各ステップで区間幅に観測記号の確率が掛け合わされるため、$x_1, \dots, x_n$ を処理し終えた後の最終区間の幅は次式で与えられる。

$$w = \prod_{i=1}^{n} q(x_i \mid x_{<i}),$$

これはまさに、モデルがメッセージ全体に割り当てる確率に一致する。長さ n の相異なるメッセージは互いに素な区間に入るため、最終区間の内部にある任意の 1 点がメッセージを一意に特定する。同じモデルを再現するデコーダは、送信された数値がどのスライスに含まれるかを調べ、対応する記号を出力し、自身の区間をエンコーダと歩調を合わせて狭めていく。デコーダはまた、事前に n を知っておくか専用のメッセージ終端記号を含めることで停止時期を知る必要がある

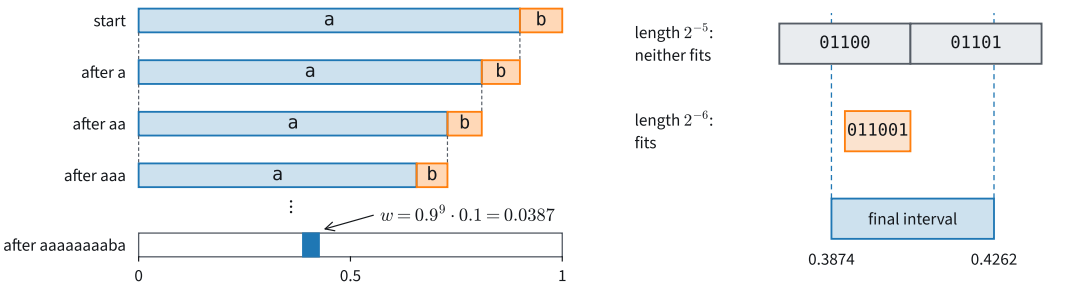


Figure 37.6: モデル $q(\mathrm{a}) = 0.9$ のもとでのメッセージ aaaaaaaaba の算術符号化。(a) 各記号は現在の区間をモデルがその記号に割り当てた部分区間へと狭めていき、10 記号後の残りの幅はメッセージの確率 $w = 0.9^9 \cdot 0.1 = 0.0387$ になる。(b) 最終区間は、その完全に内側に収まる 2 進区間によって符号化される。この区間と重なる長さ 2^{-5} の 2 進区間はいずれも完全には内側に収まらないのに対し、長さ 2^{-6} の 0.390625 から 0.40625 までの区間は完全に収まる。その 2 進アドレス 011001 が符号語であり、サプライザルが 4.69 ビットのメッセージを 6 ビットで表している。

が、ここではそのフレーミングコストは脇に置く。残る作業は最終区間内の 1 点を 2 進法で指定することだけであり、クラフトの不等式の証明で用いた 2 進区間がこれを明快に解決する。

幅 w の任意の区間は、長さ $2^{-\ell}$ の 2 進区間を必ず含むことが $\ell = \lceil \log_2(1/w) \rceil + 1$ と選ぶことで保証される。なぜだろうか。このように選べば $2^{-\ell} \le w/2$ が成り立つため、区間の左端以上である最初の $2^{-\ell}$ の倍数は必ず区間の左半分に入り、そこを始点とする長さ $2^{-\ell}$ の 2 進区間が右端をはみ出すことはあり得ないからである。Figure 37.6 の (b) は両方の状況を示している。長さ 2^{-5} の 2 進区間は最終区間の端からはみ出してしまうが、長さ 2^{-6} の区間は完全に内部に収まる。$\ell < \log_2(1/w) + 2$ であるから、得られる符号語長は次式を満たす。

$$\ell \;<\; \sum_{i=1}^{n} \log_2 \frac{1}{q(x_i \mid x_{<i})} + 2, \tag{37.17}$$

すなわち、(既知のメッセージ長 n に対して) メッセージはモデルのもとでの総サプライザルに高々 2 ビットを加えた長さで符号化される。いまや丸めは記号ごとに 1 回ではなく、メッセージ全体で 1 回だけ行われる。さらに、相異なる長さ n のメッセージに対する符号語は互いに素なメッセージ区間の内部にある 2 進区間を指定するため、クラフトの不等式の証明の前半により、メッセージ全体の上の接頭符号をなす。したがってエントロピー下界は、単一の記号に対してと同様にメッセージ全体に対しても成り立つ。どのようなモデルを選ぼうとも、期待符号長がメッセージのエントロピーを下回ることは決してない。

この区間縮小の手順を aaaaaaaaba に対して実行し、記号ごとのシャノン符号と長さを比較してみよう。以下のセルでは区間を追跡し、2 進符号語を取り出してその 2 進区間が最終区間の内部に収まることを検証し、両方のビット数を出力する。

```
msg = 'aaaaaaaaba'                  # nine a's, one b
q = {'a': 0.9, 'b': 0.1}            # an i.i.d. model of the source

lo, hi = 0.0, 1.0
for ch in msg:                      # keep the observed symbol's slice
```

```
        width = hi - lo
        if ch == 'a':                    # 'a' owns the left 90% of the interval
            hi = lo + width * q['a']
        else:                            # 'b' owns the right 10%
            lo = lo + width * q['a']

    ell = math.ceil(-math.log2(hi - lo)) + 1      # codeword length
    m = math.ceil(lo * 2 ** ell)                  # first multiple of 2^-ell >= lo
    assert lo <= m / 2 ** ell and (m + 1) / 2 ** ell <= hi
    surprisal = -math.log2(math.prod(q[ch] for ch in msg))
    shannon_bits = sum(math.ceil(math.log2(1 / q[ch])) for ch in msg)
    print(f'final interval [{lo:.4f}, {hi:.4f}), width {hi - lo:.4f}')
    print(f'surprisal {surprisal:.2f} bits, codeword {m:0{ell}b} ({ell} bits)')
    print(f'symbol-by-symbol Shannon code: {shannon_bits} bits')
```

```
final interval [0.3874, 0.4262), width 0.0387
surprisal 4.69 bits, codeword 011001 (6 bits)
symbol-by-symbol Shannon code: 13 bits
```

最終的な幅 0.0387 は予想どおり積 $0.9^9 \cdot 0.1$ に一致している。このメッセージのサプライザルは 4.69 ビットであり、算術符号化はこれを 6 ビット（`011001`）で保存するのに対し、記号ごとのシャノン符号は情報量の小さい 9 個の `a` のそれぞれに丸々 1 ビットを費やすため 13 ビットを要する。この文字列はたまたま情報源の平均的な構成（9 個の `a` と 1 個の `b`）を持っているため、そのサプライザルは $10\,H_2(0.9, 0.1) = 4.69$ ビットに等しい。`b` を 2 個含むメッセージであれば、より多くのビットが必要になる。

素朴な浮動小数点による区間縮小は、短いメッセージにしか機能しない。ステップごとに幅が倍率 q で縮むため、およそ 100 記号も進むと倍精度浮動小数点数の 53 ビットの仮数部では両端点が区別できなくなるからである（演習問題 3）。実用的な算術符号器は固定幅の整数演算を用いてこれを解決し、両端点の上位ビットが一致し次第そのビットを順次出力して、作業区間を常に最大の精度へと再スケーリングし続ける (Witten et al. 1987)。再スケーリングを行えばメッセージは任意の長さになり得て、2 ビットのオーバーヘッドも記号あたりでは消失する。すなわち、n 記号のテキストは、モデルのビット単位の平均対数損失の n 倍で保存される。

37.2.2.2 圧縮器としての言語モデル

算術符号化の議論では、記号が独立であることを一切仮定していなかった。エンコーダとデコーダが有限精度のもとで次の記号の確率を 1 ビットの違いもなく同一に計算できる限り、任意の条件付きモデル $q(\cdot \mid x_{<i})$ によって区間縮小を駆動できる。自己回帰言語モデルを算術符号器と組み合わせれば、それだけで可逆圧縮器になる。トークナイザ、モデル、符号器を共有していれば、いかなる文書もモデルのもとでの総サプライザルに高々 2 ビットを加えた長さへと圧縮される（実用上のファイル形式ではわずかなフレーミングのオーバーヘッドが加わり、またモデル自体を事前に固定する場合には送受信の双方があらかじめそのモデルを保持している必要がある）。

読み進めながらその場で学習する最も単純なモデルは、カウントモデルである。**適応的 m 次モデル**は、各文字をその直前の m 文字、すなわち**コンテキスト** s から次式を用いて予測する。

$$q(x \mid s) = \frac{c(s, x) + \tfrac{1}{2}}{c(s) + \tfrac{k}{2}},$$

ここで、$c(s, x)$ はこれまでに文字 x が s の直後に現れた回数、$c(s) = \sum_x c(s, x)$ はコンテキ

スト s の合計出現回数、k はアルファベットサイズである。すべてのカウントに $\frac{1}{2}$ を加えることで、どの文字も確率 0（無限ビットを要してしまう）にならないことが保証される。決定的に重要なのは、モデルが各文字を符号化した**後に**カウントを更新する点である。デコーダもすでに復元した文字に対して同じ更新を行うため、エンコーダとデコーダは完全に歩調を合わせて動作し、モデルパラメータを送信する必要が一切なくなる。

この圧縮器を『The Time Machine』で試してみよう。最初のセルではテキストをダウンロードして英小文字とスペースのみに整形し、適応的符号長を文字あたりビット数で計算する関数を定義する。

```
fname = d2l.download(d2l.DATA_URL + 'timemachine.txt',
                     sha1_hash='090b5e7e70c295757f55df93cb0a180b9691891a')
with open(fname) as f:
    text = re.sub('[^A-Za-z]+', ' ', f.read()).lower()   # 26 letters, space

def adaptive_bits(text, m, k):
    """Bits per character of the adaptive order-m model with half counts."""
    counts, totals, bits = {}, {}, 0.0
    for i, ch in enumerate(text):
        s = text[max(0, i - m):i]                # the context
        c = counts.setdefault(s, {})
        bits -= math.log2((c.get(ch, 0) + 0.5) / (totals.get(s, 0) + k / 2))
        c[ch] = c.get(ch, 0) + 1                 # learn after coding
        totals[s] = totals.get(s, 0) + 1
    return bits / len(text)
```

次に、長さ 0 から 4 までのコンテキストを評価し、最高圧縮レベルの `zlib` と比較する。長いコンテキストほどカウントが十分に集まれば優れた予測を行えるが、学習すべきカウントの数もはるかに多くなる。出力結果は、そのトレードオフがどこで逆転するかを示している。

```
k, n = len(set(text)), len(text)
freq = [text.count(ch) / n for ch in set(text)]
h_freq = -sum(f * math.log2(f) for f in freq)
print(f'{n} characters over {k} symbols, letter-frequency entropy '
      f'{h_freq:.3f} bits')
bits = [adaptive_bits(text, m, k) for m in range(5)]
for m, b in enumerate(bits):
    print(f'order {m}: {b:.2f} bits/char = {b * math.log(2):.2f} '
          f'nats/char, perplexity 2^bits = {2 ** b:.2f}')
print(f'zlib -9: {8 * len(zlib.compress(text.encode(), 9)) / n:.2f} bits/char')
print(f'order 0 minus the letter-frequency entropy: {bits[0] - h_freq:.4f} '
      f'bits/char, {(bits[0] - h_freq) * n:.0f} bits in all')
```

```
173428 characters over 27 symbols, letter-frequency entropy 4.078 bits
order 0: 4.08 bits/char = 2.83 nats/char, perplexity 2^bits = 16.91
order 1: 3.29 bits/char = 2.28 nats/char, perplexity 2^bits = 9.81
order 2: 2.75 bits/char = 1.90 nats/char, perplexity 2^bits = 6.70
order 3: 2.55 bits/char = 1.77 nats/char, perplexity 2^bits = 5.84
order 4: 2.78 bits/char = 1.92 nats/char, perplexity 2^bits = 6.85
zlib -9: 2.84 bits/char
order 0 minus the letter-frequency entropy: 0.0011 bits/char, 183 bits in all
```

コンテキストを追加すると、符号長はコンテキストなしの文字あたり 4.08 ビットから、3 文字の

コンテキストでの文字あたり 2.55 ビットまで下がり、`zlib` の 2.84 を上回る。しかし 4 文字目を追加すると改善は逆転し、4 次モデルは文字あたり 2.78 ビットへと再び悪化する。取り得る $27^4 \approx 530{,}000$ 通りのコンテキストの大半は小説中にごくまれにしか現れないため、モデルはそれらの次の文字の確率を十分に学習できず、カウントがウォームアップする間に費やされた余分なビットはすべてそのままファイルサイズに跳ね返るのである。

対照的に、コンテキストをまったく使わない場合、学習すべきことはごくわずかしかない。出力の 1 行目は小説の経験的な文字頻度のエントロピーである文字あたり 4.078 ビットを報告しており、これはコンテキストを使わない符号器がそれらの頻度を事前に知っていた場合に達成できるレートである。0 次モデルはゼロから出発するが、最後の行はそれらの頻度を学習するのにかかった正確なコストを測っており、文字あたりわずか 0.0011 ビット、小説全体でも 183 ビットにすぎない。26 個の文字頻度の学習は安上がりだが、50 万個のコンテキストの学習はそうはいかない（nat とパープレキシティの列については 37.2.4.1 節で改めて扱う）。

適応的モデルはパラメータの送信を不要にする代わりに、学習途上にあるファイルの序盤では予測精度が低くなるというコストを払う。事前学習済みモデルはこれとは逆のトレードオフをとる。重みさえ一度共有されていれば、どのファイルも最初の 1 バイト目から高い圧縮率で圧縮できる。Delétang et al. (2023) は 700 億パラメータの言語モデルを用いてこのトレードオフを実証している。その次トークン確率で算術符号器を駆動すると、ImageNet の画像パッチは元のサイズの 43.4%（PNG では 58.5%）、LibriSpeech の音声は 16.4%（FLAC では 30.3%）に圧縮される。ただし、これらの数値にはモデル自体のサイズは含まれていない。1 パラメータあたり 2 バイトとすると、重みは 140 GB を占める（各データセットは 1 GB にすぎない）ため、モデルサイズを含めた途端に標準コーデックには勝てなくなる。

37.2.2.3 最小記述長

『The Time Machine』の実験では、各適応的モデルは読み進めながら自身のカウントを学習するコストを支払っていた。その負担を取り除くと、ランキングは完全に逆転する。まず小説全体からカウントを集め、そのカウントを使って同じ小説をスコア付けすると、より大きなルックアップテーブルはテキストをそのまま暗記できるため、4 次は 3 次に勝ち、5 次は 4 次に勝つといった具合になる（演習問題 4）。事後的にデータへどれだけよく適合するかだけで判断すれば、表現力の高いモデルが負けることは決してない。しかし、モデルをまだ持っていない受信者には、まずそのモデルの記述を送らなければならない。**最小記述長**（minimum description length、MDL）の原理は、モデルとデータを合わせた総符号長を最小化するモデル $\mathcal{M}$ を選ぶことでこの問題を解決する。

$$\hat{\mathcal{M}} = \operatorname*{argmin}_{\mathcal{M}} \big[L(\mathcal{M}) + L(\mathcal{D} \mid \mathcal{M}) \big], \tag{37.18}$$

ここで、$L(\mathcal{M})$ はモデルを指定するのに必要なビット数であり、$L(\mathcal{D} \mid \mathcal{M})$ はそのモデルのもとでのデータ $\mathcal{D}$ の対数損失、すなわち前項までの議論によりビット単位の符号長である。MDL のもとでは、より精巧なモデルが選ばれるのは、モデルの記述に要するビット数よりもデータ上で節約できるビット数の方が大きい場合に限られる。この意味で、MDL はオッカムの剃刀（Occam's razor）をビット単位で定量化したものである。さらに、モデルの記述が事前分布から $L(\mathcal{M}) = -\log_2 P(\mathcal{M})$ によって与えられるとき、この和の最小化は負の対数事前確率と負の対数尤度の和の最小化にほかならないため、MDL は最大事後確率推定（maximum a posteriori estimation）と一致する（36.6.4 節）。

単純なコイン投げの例によって、両方の項が具体的になる。$n = 1{,}000$ 回のコイン投げで 600 回表が出たとする。公平なコインという仮説 $\mathcal{M}_0$ には送信すべきパラメータがなく、各試行を 1 ビットで符号化するため、$L(\mathcal{D} \mid \mathcal{M}_0) = 1{,}000$ ビットとなる。偏りのあるコインという仮説 $\mathcal{M}_1$ では、まずその偏り（表の確率）θ を送信しなければならず、そのためには有限の精度を選ぶ必

要がある。n 回の試行における表の標本比率は $1/\sqrt{n}$ のスケールで揺らぐため、θ の桁を $1/\sqrt{n}$ より細かく送ってもサンプリングノイズを記述するのにビットを浪費するだけである（演習問題 5）。間隔 $1/\sqrt{n}$ で区間 $(0,1)$ 上に設けたグリッドには $\sqrt{n}$ 個の点があるため、最も近いグリッド点を指定するコストは $L(\mathcal{M}_1) = \log_2 \sqrt{n} = \tfrac{1}{2}\log_2 1{,}000 \approx 5$ ビットとなる。θ が与えられれば、データの符号化には $L(\mathcal{D} \mid \mathcal{M}_1) \approx 1{,}000\, H_2(0.6, 0.4) = 971$ ビットを要し、合計は $976 < 1{,}000$ ビットになる。したがって MDL は偏りのあるコインを選択する。パラメータに 5 ビットを費やすことで、データ上で 29 ビットが節約されるからである。もし表が 520 回しか出なかったとすれば、データ上の節約はわずか 1.2 ビットにとどまり、パラメータのコストである 5 ビットを下回るため、MDL は公平なコインを維持する。このように実数パラメータ 1 個あたり $\tfrac{1}{2}\log_2 n$ ビットを課す規準は、**ベイズ情報量規準**（Bayesian information criterion）として知られている。

実は、『The Time Machine』の実験はすでに MDL を適用していたのである。適応的符号長は、モデルを学習するコストがビットストリームの中に直接折り込まれた、データの有効かつ完全な記述になっている。0 次モデルでは、その学習コストはごくわずかである。$\tfrac{1}{2}\log_2 n \approx 8.7$ ビットを 26 個の自由な文字頻度のそれぞれに課すと約 226 ビットと予測され、実際に観測された 183 ビットにかなり近い値となる（この規準は漸近展開の主項のみを残したものである）。4 次モデルでは、観測されたコンテキストごとに別々の分布を学習するコストが、コンテキストの 4 文字目によって節約されるビット数を上回ったため、適応的符号長は自動的に 3 次を選好したのである。

ここで、素朴なパラメータ数による評価は大きく的を外してしまうことに注意しよう。4 次モデルは形式上 $27^4 \cdot 26 \approx 14$ 百万個の自由な頻度パラメータを持つため、1 個あたり 8.7 ビットとすると、ベイズ情報量規準は 120 百万ビットものペナルティを課すことになる（1 文字 8 ビットの非圧縮でもわずか 1.4 百万ビットしか要しない小説に対してである）。パラメータ数を数える漸近論は、パラメータよりもはるかに多くのデータがあることを前提としている。現実には、そうした 4 文字のコンテキストの大半は一度も現れず、適応的符号はそれらに対して 1 ビットも課さない。現代のニューラルネットワークを含め、モデルがデータよりも多くのパラメータを持つ状況では常に、適応的符号長こそが意味を保ち続ける記述長となる。

37.2.3 エントロピー率と典型系列

ここまでは、モデルを所与のものとしてその符号長を測ってきた。ここで問いを裏返そう。記憶を持つ情報源に対して、**考え得る最良の**モデルの符号はどこまで短くできるのだろうか。

37.2.3.1 エントロピー率

独立な記号に対しては、圧縮の基本下限は 1 記号あたりのエントロピーであった。しかし実際のテキストは依存関係に満ちている。`q` の後には `u` が続き、`th` の後には `e` や `a` が `z` よりはるかに高い頻度で現れる。記憶を持つ情報源を解析するため、系列 $X_1, X_2, \ldots$ は**定常**（stationary）であるとする。すなわち、任意の記号ブロックの同時分布が時間をずらしても変化しないものとする。

37.1.1.3 節で導入した結合エントロピーと条件付きエントロピーを用いる。本節の残りの部分では、確率変数に対するビット単位のエントロピーを H で表し、明示的な確率ベクトルの 2 進エントロピーは引き続き $H_2(p_1, \ldots, p_k)$ で表す。結合エントロピー $H(X_1, \ldots, X_n)$ は n 記号のブロック全体を単一の確率変数として扱い、条件付きエントロピー $H(X_n \mid X_{<n}) = \mathbb{E}\big[-\log_2 p(X_n \mid X_{<n})\big]$ は考え得る最良の次記号予測器、すなわち真の情報源そのものの期待対数損失を測る。連鎖律 $H(X, Y) = H(X) + H(Y \mid X)$ を繰り返し適用すると、ブロックエントロピーは条件付きエントロピーの和 $H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i \mid X_{<i})$ へと分解される。情報源の**エントロピー率**（entropy rate）とは、条件付きエントロピーの極限、あるいは同値な表現として長いブロックにおける記号あたりエントロピーの極限である。

$$H_{\text{rate}} = \lim_{n\to\infty} \frac{1}{n} H(X_1, \dots, X_n) = \lim_{n\to\infty} H(X_n \mid X_{<n}). \tag{37.19}$$

ブロックエントロピーは n 個の条件付きエントロピーの和であるため、第 1 の式は、第 2 の式にその極限が現れている各項の逐次平均になっている。任意の定常情報源に対して両方の極限は存在して一致し、かつ $\frac{1}{n}H(X_1, \dots, X_n)$ は H_{rate} に向かって単調に減少するが (Cover and Thomas 1999)、これらの事実はここでは認めることとする。直感的な意味は明快である。条件付けによってエントロピーが増加することは決してないため（37.1 節）、コンテキストの記号が 1 つ増えるごとに次の記号の不確実性は減少するか変化しないかのいずれかであり、非増加系列の逐次平均もまた非増加となる。記号が独立同分布であるとき、両方の極限は直ちに $H(X_1)$ へと帰着する。

2 状態のマルコフ連鎖を考えると、記憶がいかに大きく下限を引き下げるかがわかる。連鎖が a または b を出力し、現在の記号に確率 0.9 で留まり、確率 0.1 で切り替わるとしよう。対称性により両方の記号は同じ頻度で現れるため、単一の記号を単独で見ればエントロピーは $H(X_1) = 1$ ビットである。しかし直前の記号が与えられると、次の記号は切り替わる確率が 0.1 の偏りのあるコインのように振る舞うため、$H(X_n \mid X_{n-1}) = H_2(0.9, 0.1) = 0.47$ ビットとなり、さらに過去を遡っても新たな情報は加わらない。記憶があるというだけで、基本下限は記号あたり 1 ビットから 0.47 ビットへと下がるのである（演習問題 6 では一般の非対称な 2 状態連鎖を解析する）。

任意の定常情報源に対し、いかなる一意復号可能符号も平均して記号あたり H_{rate} ビット未満にはできない。実際、n 記号のブロックに対する任意の符号は n 記号ブロックのアルファベット上の記号符号であるから、（マクミランの定理により一意復号可能符号へ拡張された）エントロピー限界によってブロックあたり少なくとも $H(X_1, \dots, X_n)$ ビットを費やさなければならず、単調性から $H(X_1, \dots, X_n) \geq n H_{\text{rate}}$ が保証される。

同様に重要なこととして、この下限は達成可能である。情報源の真の条件付き確率で駆動される算術符号器は、Equation 37.17 により、メッセージのサプライザルに 2 ビットを加えた値より狭義に少ないビット数しか費やさない。n 記号のブロックのそれ自身の分布のもとでの期待サプライザルは $H(X_1, \dots, X_n)$ であるから、この符号器の平均符号長は記号あたり $\frac{1}{n}H(X_1, \dots, X_n) + \frac{2}{n}$ ビット未満となり、これは H_{rate} へと（n が大きくなるにつれて）収束する。

古典的な実験において、Shannon は印刷された英語の次の文字を人間に推測させ、英語のエントロピー率が文字あたりおおよそ 0.6 から 1.3 ビットの間に位置すると結論づけた (Shannon 1951)。これは一様ランダムな文字とスペースに必要な $\log_2 27 = 4.75$ ビットをはるかに下回る。ここで、実際に動作するいかなる圧縮器もエントロピー率の**上界**を与えることに注目しよう。先ほどの 3 次カウントモデルによる文字あたり 2.55 ビットもそのような上界の 1 つである一方、いかなる圧縮器も下界を証明することは決してできない。本章の冒頭に戻ると、文字あたり 1.2 nat の損失を持つ言語モデルは文字あたり 1.73 ビットを達成しており、カウントモデルの 2.55 を大きく下回るものの、依然として Shannon の推定範囲の上端よりは上にある。

現在得られている最良の上界は、大規模言語モデルによるものである。算術符号器を駆動することで、Delétang et al. (2023) のモデルは Wikipedia のテキストをバイトあたり 1 ビット未満、すなわち Shannon の推定範囲の内側まで圧縮する。ただし、これを英語のエントロピー率として解釈する前に 3 つの注意点がある。モデルのサイズが含まれていないこと、テスト用テキストの一部が訓練中に見られていた可能性があること、そして Wikipedia のマークアップの 1 バイトは Shannon の 27 文字アルファベットの 1 記号とは異なることである。真のエントロピー率がこれらの上界より下のどこに位置するのかは、誰にもわかっていない。

37.2.3.2 典型系列

ここまでは**平均**符号長の上下界を求めてきた。では、1 本の長いメッセージはどのような姿をしており、起こりやすいメッセージは実際に何通り存在するのだろうか。この問いに答えることで、起こりやすいメッセージの集合の大きさを数える指数という第 2 の角度からエントロピーに光が当たる。本節の残りの部分はこの項に依存しないため、初読時には読み飛ばしても構わない。数値を具体的に保つため、値 1 を確率 0.3 でとる独立な 2 進記号からなる、長さ $n = 200$ のメッセージを考えよう。

単独で最も確率の高いメッセージはすべてが 0 の系列であり、その確率は $0.7^{200} = 2^{-103}$ である。期待個数である 60 個の 1 を含むメッセージの確率は $0.3^{60}\,0.7^{140} = 2^{-176.3}$ であり、これは 2^{-nH}（ただし $H = H_2(0.3, 0.7) = 0.881$ ビット）に等しく、そのようなメッセージは $\binom{200}{60} = 2^{172.2}$ 通り存在する。一方、文字列の総数は 1 が 100 個のときにピークを迎え $2^{195.9}$ 通りにもなるが、その 1 本ずつの確率は 2^{-225} にすぎないため、それらすべてを合わせても全確率質量のわずか 2^{-29} しか担っていない。ほとんどのメッセージは起こりにくく、しかも最も起こりやすいメッセージは集団を代表するものではないのである。

表記 $x_{1:n}$ で系列 $x_1, \dots, x_n$ を表すことにする。記号は互いに独立であるため、記号あたりのサプライザルはそれぞれ期待値 $H = H_2(P)$ を持つ独立同分布な確率変数の標本平均になる。大数の法則により、この標本平均はその期待値へと確率収束する。

$$-\frac{1}{n}\log_2 P(X_{1:n}) = \frac{1}{n}\sum_{i=1}^{n} -\log_2 p(X_i) \xrightarrow{P} H. \tag{37.20}$$

したがって、任意の固定された許容誤差 $\delta > 0$ に対し、長いメッセージのサプライザルは 1 に近づく確率で誤差 $n\delta$ 以内の範囲（nH ビットの周囲）に収まる。この現象を**漸近的等分配性**（asymptotic equipartition property、AEP）と呼び、それによって捉えられるメッセージ全体は許容誤差 δ における**典型集合**（typical set）をなす。

$$A_\delta = \left\{ x_{1:n} : \left| -\frac{1}{n}\log_2 P(x_{1:n}) - H \right| < \delta \right\}. \tag{37.21}$$

命題（AEP） $X_1, \dots, X_n$ **を有限アルファベット上の分布** P**（2 進エントロピー** H**）からの独立な抽出とし、** $\delta > 0$ **とする。このとき次が成り立つ。(i)** $P(A_\delta) \to 1$（$n \to \infty$ **のとき）。(ii) すべての** $x_{1:n} \in A_\delta$ **は** $2^{-n(H+\delta)} < P(x_{1:n}) < 2^{-n(H-\delta)}$ **を満たす。(iii)** $|A_\delta| \le 2^{n(H+\delta)}$ **であり、また** $|A_\delta| \ge (1-\eta)\,2^{n(H-\delta)}$ **が** $P(A_\delta) \ge 1-\eta$ **である限り成り立つ。**

証明 (i) は Equation 37.20 から直ちに従い、(ii) は A_δ を定義する不等式において 2 のべき乗をとったものにすぎない。(iii) については、(ii) の上下界を A_δ のすべての元にわたって足し合わせることで $1 \ge P(A_\delta) > |A_\delta|\,2^{-n(H+\delta)}$ および $1-\eta \le P(A_\delta) < |A_\delta|\,2^{-n(H-\delta)}$ が得られ、ここから要素数に関する両方の上下界が直ちに従う。■

したがって指数スケールで見れば、ほぼすべての確率質量は約 2^{nH} 本の典型系列に集中し、それぞれの系列は約 2^{-nH} の確率で生じる。ただし、「約」という言葉の裏には $2^{\pm n\delta}$ の因子が隠れており、有限の長さではこれが無視できない大きさになることに留意しよう。実際、$n = 200$、$\delta = 0.1$ のとき、この幅は 2^{20} にもなる。

AEP から、シャノンの情報源符号化定理（Shannon's source-coding theorem）が直ちに導かれる (Shannon 1948; Cover and Thomas 1999)。圧縮するには、A_δ 内の各系列に $\lceil n(H+\delta) \rceil$ ビットの固定長インデックスを割り当て、非典型的なメッセージに対しては失敗を宣言すればよい。情報源の 1 記号あたりに費やすビット数を符号の**レート**と呼ぶ。ここでのレートは（丸めを除け

ば）$H+\delta$ であり、失敗確率は $P(A_\delta^c)\to 0$ となる。$\delta>0$ はいくらでも小さくできるため、H より大きいすべてのレートが達成可能である（先頭に 1 ビットのフラグを付加し、非典型的なメッセージには非圧縮のコピーへフォールバックするようにすれば、同じ漸近レートで厳密に可逆な符号にできる）。逆に、レート $R<H$ の任意の固定長符号は 2^{nR} 個の符号語しか持たないため、表現できるメッセージの集合 B の要素数は高々 2^{nR} 個にとどまる。$\delta<H-R$ と選ぶと、次式が得られる。

$$P(B)\le P(B\cap A_\delta)+P(A_\delta^c)<2^{nR}\,2^{-n(H-\delta)}+P(A_\delta^c)\to 0.$$

したがって、H 未満のレートで動作するいかなる固定長符号も、1 に収束する確率で失敗する。同じ定理は定常エルゴード情報源に対しても H_{rate} で H を置き換えることで成り立つが、この一般化は証明なしに認めることとする (Cover and Thomas 1999)。

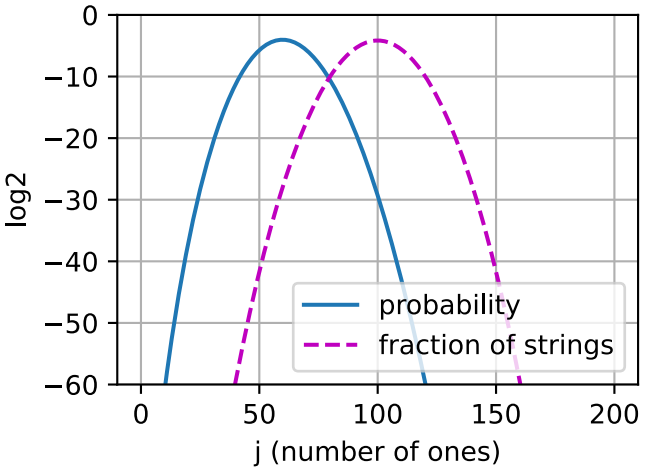
このベルヌーイ情報源に対しては、1 の個数 j が同じメッセージはすべて同じ確率を持つため、典型集合の要素数を**厳密に**数え上げることができる。以下のセルでは、全 2^{200} 通りの 2 進文字列を j ごとにグループ化し、$\delta=0.1$ における典型性の条件を満たすグループの個数と確率を合計して、得られた集合のサイズを基準値 $2^{nH}=2^{176.3}$ と比較する。

```
n, p, delta = 200, 0.3, 0.1
j = np.arange(n + 1)                                # number of ones
log_fact = np.concatenate([[0.0], np.cumsum(np.log(np.arange(1, n + 1)))])
log2_count = (log_fact[n] - log_fact[j] - log_fact[n - j]) / np.log(2)
bits_per_symbol = -(j * np.log2(p) + (n - j) * np.log2(1 - p)) / n
log2_prob_j = log2_count - n * bits_per_symbol       # log2 P(j ones)
h = -(p * np.log2(p) + (1 - p) * np.log2(1 - p))
typical = np.abs(bits_per_symbol - h) < delta
log2_size = np.logaddexp2.reduce(log2_count[typical])
prob = (2 ** log2_prob_j[typical]).sum()
print(f'H = {h:.3f} bits/symbol, nH = {n * h:.1f} bits')
print(f'typical: 2^{log2_size:.1f} of 2^{n} strings, '
      f'{j[typical].min()} <= j <= {j[typical].max()}')
print(f'probability they carry: {prob:.4f}')
print(f'bounds: {prob:.3f} * 2^{n * (h - delta):.1f} <= |A| '
      f'<= 2^{n * (h + delta):.1f}')
```

```
H = 0.881 bits/symbol, nH = 176.3 bits
typical: 2^188.8 of 2^200 strings, 44 <= j <= 76
probability they carry: 0.9893
bounds: 0.989 * 2^156.3 <= |A| <= 2^196.3
```

$2^{188.8}$ 通りの典型文字列（1 の個数が 44 個から 76 個までのもの）は、全 2 進文字列のうち 2^{11} 分の 1 を占めるにすぎないが、それら全体で確率 0.989 を担っている。その個数が 2^{nH} を数千倍上回っているのは、$\delta=0.1$ としたことで 1 を最大 76 個含む文字列（60 個付近の文字列よりもはるかに数が多い）まで許容されるためである。それでもなお、このサイズは命題が与える上下界である $0.989\cdot 2^{156.3}$ と $2^{196.3}$ の間に余裕をもって収まっている。確率と文字列の割合の両方を j の関数としてプロットし、両者がどれほど鮮やかに分離するか見てみよう。

```
d2l.plot(j, [log2_prob_j, log2_count - n], 'j (number of ones)', 'log2',
         legend=['probability', 'fraction of strings'],
         ylim=[-60, 0])
```



2 本の曲線のピークは大きく離れている。確率質量は $j = 60$ でピークに達するが、そこでの文字列は全可能性のうちわずか 2^{-28} の割合しか占めない。一方、文字列の個数は $j = 100$ でピークに達するが、それらの合計確率はわずか 2^{-29} にすぎない。単独で最も確率の高い文字列（$j = 0$、すべて 0）の確率は 2^{-103} であり、グラフの範囲から完全に外れてしまう。典型系列とは、最も確率の高い文字列でもなければ最も数の多い文字列でもなく、個数と個々の確率の積が最大になる狭い帯域に存在する系列なのである。

典型集合は証明のための強力な道具ではあるが、実用的な符号器としては不器用である。これに直接インデックスを振ると、すべての典型文字列に 189 ビット（13 ビットだけ $nH = 176.3$ より多い）を要するうえ、おおよそ 100 本に 1 本のメッセージで失敗する。これに対し、真のモデルを用いた算術符号化は厳密に可逆であり、Equation 37.17 により平均 $nH + 2 = 178.3$ ビット未満で済む。典型集合が与えてくれるのは幾何学的な直感である。すなわち、長いメッセージはほぼ確実に情報源の真の組成を反映し、約 2^{nH} 通りのメッセージがその組成を共有している。

37.2.4 パープレキシティ

実際には、言語モデルがその性能をファイルサイズとして報告することはほとんどない。代わりに、ホールドアウトされたテキストに対する単一の数値である**パープレキシティ**を報告するが、これは単に本節の符号長を別の単位で表したものにすぎない。

37.2.4.1 分岐数としてのパープレキシティ

n 個のトークンからなるテキストと、次のトークンの確率 $q(\cdot \mid x_{<i})$ を割り当てるモデルが与えられたとき、nat 単位でのトークンあたり平均対数損失とその指数である**パープレキシティ**(Jelinek et al. 1977) は、次式で定義される。

$$\mathrm{CE} = -\frac{1}{n}\sum_{i=1}^{n} \log q(x_i \mid x_{<i}), \qquad \mathrm{PPL} = \exp(\mathrm{CE}) = \Big(\prod_{i=1}^{n} q(x_i \mid x_{<i})\Big)^{-1/n}. \tag{37.22}$$

分布の引数を省略して書いた CE は、この特定のテキストに対するモデルの経験損失を表しており、情報源から抽出されたテキストにわたるその期待値が情報源とモデルの間の交差エントロピーである。第 2 の表式から、パープレキシティは観測されたトークンに割り当てられた確率の幾

何平均の逆数であることがわかる。8.3.2 節で導入したように、パープレキシティは**実効分岐数**（effective branching factor）として直ちに解釈できる。毎ステップで k 個のトークンに一様に確率を割り振るモデルのパープレキシティは k になり、決定論的な系列に対する完全な予測器のパープレキシティは 1 になる。

37.2.2.2 節における『The Time Machine』の結果は、文字レベルのテキストでこれを具体的に示している。何の知識も持たないモデルは全 27 記号から一様に推測するため、パープレキシティは 27 になる。文字頻度を学習するだけでパープレキシティは 16.91（1 文字あたりおおよそ 17 通りの等確率な選択肢）に下がり、3 文字のコンテキストを使えば 5.84 まで下がる。平均すると、このモデルが次の文字について抱える不確実性は、6 面サイコロの出目に対する不確実性とほぼ同程度なのである。なお、先ほどのコードでは 2 の「文字あたりビット数」乗としてパープレキシティを計算したのに対し、Equation 37.22 では e の「文字あたり nat 数」乗を用いているが、$\mathrm{PPL} = b^{\mathrm{CE}_b}$ が任意の対数の底 b に対して成り立つため、両者は同じ数値を与える（演習問題 8）。

37.2.4.2 パープレキシティの下限

符号長がエントロピー率を下回れないのと同様に、パープレキシティにも基本的な下限がある。定常情報源に対して、期待パープレキシティは少なくとも $2^{H_{\text{rate}}}$ である。CE は nat 単位で測られるのに対し、H_{rate}（Equation 37.19）はビット単位で測られるため、係数 $\ln 2$ によって両者の単位を変換する。

命題（パープレキシティの下限） $X_1, \dots, X_n$ **を定常情報源から抽出された系列とし、** q **を任意のモデル、CE と PPL をこのテキストに対する経験損失とパープレキシティ（Equation 37.22、モデル** q **のもとでの値）とする。このとき次式が成り立つ。**

$$\mathbb{E}[\mathrm{CE}] \geq H_{\text{rate}} \ln 2, \qquad \mathbb{E}[\mathrm{PPL}] = \mathbb{E}\big[e^{\mathrm{CE}}\big] \geq 2^{H_{\text{rate}}}. \tag{37.23}$$

証明 任意のプレフィックス $x_{<i}$ を固定する。$x_{<i}$ が与えられたときのモデルの次トークン期待損失は、真の条件付き分布 $p(\cdot \mid x_{<i})$ と $q(\cdot \mid x_{<i})$ の間の交差エントロピーであり、ギブスの不等式により情報源の条件付きエントロピー以上である。プレフィックスにわたって平均をとると $\mathbb{E}[-\log q(X_i \mid X_{<i})] \geq \mathbb{E}[-\log p(X_i \mid X_{<i})]$ が得られる。系列の対数確率は $\log p(x_{1:n}) = \sum_{i} \log p(x_i \mid x_{<i})$ と分解されるため、i について足し合わせて n で割ると $\mathbb{E}[\mathrm{CE}] \geq \frac{\ln 2}{n} H(X_1, \dots, X_n) \geq H_{\text{rate}} \ln 2$ となる。ここで第 2 の不等号は、記号あたりのブロックエントロピーの単調減少性から従う。最後に、e^{x} は凸関数であるから、イェンセンの不等式により $\mathbb{E}[e^{\mathrm{CE}}] \geq e^{\mathbb{E}[\mathrm{CE}]} \geq 2^{H_{\text{rate}}}$ が成り立つ。$\blacksquare$

要するに、いかなるモデルも平均的には、情報源が本来持つランダム性が許す以上にパープレキシティを低くすることはできない。規模感をつかむため、あるテキストのエントロピー率が文字あたり 1 ビット（英語に対する Shannon の推定範囲内の値）だったとしよう。すると、いかなる文字レベルモデルも期待パープレキシティ 2（1 文字につき公平なコイン投げ 1 回分）を下回ることはできず、先ほどの 3 次カウントモデルの 5.84 はその下限よりはるかに上に位置することになる。実際のテキストに対しては、この下限は上界を通してしか知ることのできないエントロピー率そのものと同じだけ不確かである。

この命題が下界を与えているのはあくまで**期待値**であり、1 本の短いテキストに対するパープレキシティはこの下限を容易に下回り得ることに十分注意しよう。再び、`a` を確率 0.9 で出力する情報源（下限は $2^{H} = 1.38$）を考える。真のモデルのもとでさえ、10 記号の文字列は 1.38 未満のスコアを、`b` を 1 つも含まないときはいつでも記録し、それは確率 $0.9^{10} = 35\%$ で起こる。この情報源からの長いテキストであっても、パープレキシティはほぼ半分の確率で依然として 2^{H} を下回るが、その差は長さが伸びるにつれて消失していく。より一般に、エルゴード情報源からの 1 本の長いテキストを真のモデルでスコア付けしたときのパープレキシティは $2^{H_{\text{rate}}}$ へと（$n \to \infty$ のとき）収束するが、この事実はここでは認めることとする。

37.2.4.3 トークナイザ間のパープレキシティ

パープレキシティはトークンあたりで平均されるため、その値はトークナイザに直接依存する。したがって、2 つのモデルを生のパープレキシティで比較できるのは、両者が同じトークナイザを共有している場合に限られる。8.3.2.4 節では 3 種類のトークナイザにわたってこの落とし穴を実演し、総サプライザルをテキストのバイト数で割ることでこれを解決した。本節の言葉で言えば、この対処法は単にパープレキシティを元のテキストの単位あたりの符号長へと戻しているにすぎない。n_{char} 文字のテキストが n_{tok} 個のトークンに分割されているとき、総符号長は $n_{\text{tok}} \log_2 \text{PPL}$ ビットとなり、次式が得られる。

$$\text{bpc} = \frac{n_{\text{tok}}}{n_{\text{char}}} \log_2 \text{PPL}, \tag{37.24}$$

ここで分母はテキスト自体のみに依存する。27 記号のテキストのようにすべての文字が 1 バイトである場合、文字あたりビット数は本編で用いたバイトあたりビット数と完全に一致する。

符号長からパープレキシティに至るまで、本節で扱ったすべての量は 1 つの決定的な要件に依拠している。それは、エンコーダとデコーダの双方が各記号に対するモデルの確率 $q(x_i \mid x_{<i})$ を計算できなければならないという点である。生成モデルの出力を実データと比較するときのように、2 つの分布にサンプルを通してしかアクセスできない場合には、そのような符号は存在しない。37.3 節と 37.4 節では、手元にサンプルしかない状況で分布を比較する手法を展開する。

37.2.5 まとめ

モデルの対数損失は、適切な符号と組み合わせることでファイルサイズそのものになる。記号ごとの接頭符号はいずれも各符号語を整数ビットに丸めなければならず、そのため最適な期待符号長は $H_2(P)$ ビットと $H_2(P)+1$ ビットの間に位置する。算術符号化はメッセージごとに 1 回だけ丸めを行うことでこの記号ごとのペナルティを取り除き、自己回帰言語モデルを含む任意の次記号モデルで駆動することで、系列をその総サプライザルに高々 2 ビットを加えた長さへと圧縮する。不一致モデル Q のもとでは期待符号長は交差エントロピー $\text{CE}_2(P, Q)$ となり、KL ダイバージェンスはモデル Q を P の代わりに用いるオーバーヘッドを測る。『The Time Machine』に対しては、適応的 3 次カウントモデルがテキストを文字あたり 2.55 ビットに圧縮して `zlib` を上回る一方、4 次では希少なコンテキストの学習に費やすビット数が 4 文字目による節約分を上回るため悪化し、最小記述長の原理を具体的に示している。記憶を持つ定常情報源に対しては、いかなる一意復号可能符号も平均して記号あたりエントロピー率 H_{rate} 未満のビット数にはできない。最後に、パープレキシティはこの符号長を単に指数化したものであり、実効分岐数として働き、期待値の下界 $2^{H_{\text{rate}}}$ に従い、異なるトークナイザ間では文字あたりビット数またはバイトあたりビット数へ変換し直した後にのみ比較できる。

37.2.6 演習問題

1. **マクミランの定理**。符号語のあらゆる連結がただ 1 通りにしか区切れないとき、その符号は一意復号可能である。
    1. 符号 $\{0, 01, 011\}$ が一意復号可能であるがプレフィックスフリーではないことを示し、クラフトの不等式を満たすことを確かめよ。
    2. すべての一意復号可能符号がクラフトの不等式 Equation 37.13 を満たすことを証明せよ。ヒント：$\big(\sum_i 2^{-\ell_i}\big)^m$ を、m 個の符号語からなるすべての列にわたる $2^{-(\text{total length})}$ の和として展開し、それらの列を合計長 L（ただし $m \le L \le m\ell_{\max}$）ごとにまとめる。一意復号可能性により高々 2^L 通りの合計長 L の列しか存在し得な

いため、和全体は高々 $m\ell_{\max}$ となる。あとは両辺の m 乗根をとって $m \to \infty$ とすればよい。

2. **ハフマン符号**。ハフマンのアルゴリズム (Huffman 1952) は、最も確率の低い 2 つの記号をその確率の和を持つ 1 つの記号へと繰り返し統合し、各統合の 2 つの要素に 0 と 1 を割り当てることで接頭符号を構成する。

    1. これを $P = (0.6, 0.3, 0.1)$ および $(0.4, 0.3, 0.2, 0.1)$ に対して実行せよ。それぞれの符号の符号語長と期待符号長を、シャノン符号および H_2 と並べて示せ。
    2. ハフマン符号がすべての接頭符号の中で最小の期待符号長を持つことを証明せよ。まず、最適符号においては $p_i > p_j$ ならば $\ell_i \le \ell_j$ が成り立つこと、また最長の 2 つの符号語は同じ長さを持ち木の中で兄弟ノードにとれることを示せ。次に、最も確率の低い 2 つの記号 i, j を統合すると問題が 1 記号少ない場合へと帰着し、$\mathbb{E}[\ell] = \mathbb{E}[\ell'] + p_i + p_j$（ただし ℓ' は統合後の符号の符号語長）が成り立つことを示せ。帰納法により結論を導け。ヒント：2 つの符号語を入れ替えて期待符号長を比較する。
    3. a を確率 0.9 で出力する情報源に対し、いかなる記号符号も記号あたり少なくとも 1 ビットを費やすことを示せ。次に、記号の**ペア**を 4 記号の情報源としてハフマンのアルゴリズムで符号化し、本文の「元の記号あたり 0.645 ビット」を確かめよ。3 つ組に対しても同様に行え。それぞれの場合で、エントロピーからの超過分は上界 $1/n$ と比べてどうなっているか。
    4. 長さ $1, 2, \dots, k-1, k-1$ を持つ線形推測符号を考えよ。確率を $p_1 \ge \cdots \ge p_k$ と並べたとき、$p_i \ge \sum_{j>i} p_j$ がすべての i について成り立つならばハフマンのアルゴリズムがこれらの長さを返し得ること、したがってそのとき線形推測符号が最適な接頭符号であることを示せ。情報源 $(0.4, 0.3, 0.2, 0.1)$ はこの条件に反している。この情報源に対して線形推測符号は最適か。

3. [code] **厳密な算術符号化**。●37.2.2.1 節の区間縮小では、`lo` と `hi` を浮動小数点数として保持している。$n = 1000$ 記号のメッセージ（各記号は a が確率 0.9 で生じ、それ以外は b となる）を 1 つ生成せよ。

    1. このメッセージに対してモデル $q(\mathtt{a}) = 0.9$ で浮動小数点による区間縮小を実行せよ。何記号目で `hi - lo` が確率の積から初めて 0.1% を超えてずれ、また何記号目で 0 になるか。両方の答えを倍精度浮動小数点数の 53 ビットの仮数部と関連づけよ。
    2. 厳密な有理数（`fractions.Fraction`）を用いて区間縮小を繰り返せ。本文のコードセルと同様に最終区間内の 1 点を指定し、モデルを再現してメッセージを復元するデコーダを実装せよ。さらに、その符号長をメッセージのサプライザルおよび $nH_2(0.9, 0.1)$ と比較せよ。
    3. 同じメッセージを不一致モデル $q(\mathtt{a}) = 0.7$ で符号化せよ。符号長の増加分を $nD^{(2)}_{\mathrm{KL}}(P\|Q)$ と比較せよ。1 本のメッセージに対してこれら 2 つの数値が異なるのはなぜか。

4. [extended] **適応的コンテキストモデルによる圧縮**。●●本文の `adaptive_bits` は、モデルの学習コストを内部に含んだ Equation 37.18 の意味での完全な記述長である。Equation 37.17 により、その予測で駆動される算術符号器が余分に費やすビット数は高々 2 ビットにとどまる。Blier and Ollivier (2018) はこのような逐次予測（prequential）符号を用いて深層ネットワークの記述長を測っている。

    1. 本文の実験を $m = 0, 1, \dots, 8$ に拡張せよ。各 m について、適応的符号長を、テキスト全体からカウントを集めた m 次モデルの文字あたり交差エントロピーおよび `zlib` と比較せよ。
    2. 適合済みモデルの交差エントロピーが単調に下がり続ける一方で、適応的符号長が m

に関して最小値を持つ理由を説明せよ。MDL はどの次数を選ぶか。適合済みモデルの交差エントロピーは Equation 37.18 のどの項を落としているか。

3. たとえば長いコンテキストのカウントが少ないときにより短いコンテキストへフォールバックするなどして、複数の次数のコンテキストを組み合わせる適応的モデルを設計せよ。試したすべてのモデルについて文字あたりビット数を `zlib` と併せて表にまとめ、最も重要だった設計上の選択について 1 段落で述べよ。

5. **コインに対する 2 部符号。**●系列 $\mathcal{D}$（n 回のコイン投げ）には表が j 回含まれ、$\hat{\theta} = j/n$ とする。仮説 $\mathcal{M}_0$ は公平なコインであり、記述すべきパラメータを持たず $L(\mathcal{D} \mid \mathcal{M}_0) = n$ ビットとなる。仮説 $\mathcal{M}_1$ は偏り θ を持つコインである。そのエンコーダは点 θ_q（$\hat{\theta}$ に最も近い、間隔 Δ で $(0,1)$ 上に設けたグリッド上の点）を送り、これに $L(\mathcal{M}_1) = \log_2(1/\Delta)$ ビットのコストがかかり、デコーダは n と Δ を知っているものとする。

   1. 分解 Equation 37.14 を用いて $L(\mathcal{D} \mid \mathcal{M}_1) = nH_2(\hat{\theta}, 1-\hat{\theta}) + nD^{(2)}_{\mathrm{KL}}(\hat{\theta} \,\|\, \theta_q)$ を示せ（ただしこの KL ダイバージェンスはこれらのパラメータを持つベルヌーイ分布どうしのものとする）。nat 単位では、この KL ダイバージェンスは 2 次の項まで展開すると $(\theta_q - \hat{\theta})^2 / (2\hat{\theta}(1-\hat{\theta}))$ となる。この展開を導出し、Equation 37.18 による $\mathcal{M}_1$ の総符号長が $nH_2(\hat{\theta}, 1-\hat{\theta})$ を超える量は高々およそ $\log_2(1/\Delta) + n\Delta^2/(8\hat{\theta}(1-\hat{\theta})\ln 2)$ ビットであることを示せ。
   2. どの間隔 Δ がこの上界を最小化するか。デコーダは Δ を知ったうえでなければ θ_q をデコードできない。それゆえ符号がこの最小点を使えないのはなぜか。また、間隔 $\Delta = 1/\sqrt{n}$（n のみに依存する）をとれば、過剰ビット数が $\tfrac{1}{2}\log_2 n + O(1)$ ビットに収まることを（$\hat{\theta}$ が 0 と 1 から有界に離れている限りにおいて）示せ。
   3. $n = 1000$ とし、近似 $L(\mathcal{M}_1) + L(\mathcal{D} \mid \mathcal{M}_1) \approx nH_2(\hat{\theta}, 1-\hat{\theta}) + \tfrac{1}{2}\log_2 n$ ビットを用いよ。本文では $\mathcal{M}_1$ が $j = 600$ において 24 ビットの差で勝つことを示した。$j = 520$ ではどちらの仮説が勝つか。$n/2$ から j がおよそどれだけ離れれば $\mathcal{M}_1$ が勝つか、またそれは公平なコインのもとでの標準偏差 $\sqrt{n}/2$（j の標準偏差）の何倍にあたるか。

6. **2 状態連鎖のエントロピー率。**$\{\mathtt{a}, \mathtt{b}\}$ 上のマルコフ連鎖が、a から b へ確率 0.1 で遷移し、b から a へ確率 0.3 で遷移し、その定常分布から出発するものとする。

   1. 定常分布とビット単位での $H(X_1)$ を求めよ。
   2. $H(X_n \mid X_{n-1}) = \sum_s \pi_s H_2(r_s, 1-r_s)$ ビットとなること（ただし π は定常分布、r_s は状態 s からの離脱確率）、およびエントロピー率がこの値に等しいことを示せ。さらにその値を計算せよ。
   3. $\frac{1}{n}H(X_1, \ldots, X_n) = \frac{1}{n}\big(H(X_1) + (n-1)H_{\text{rate}}\big)$ であり、これが H_{rate} へと減少することを示せ。
   4. 2 つの極限を考えよ。両方の離脱確率が $\tfrac{1}{2}$ に等しいとき、連鎖は独立同分布となりエントロピー率は 1 ビットである。両方の離脱確率が r に等しいとき、エントロピー率は 0 へと（$r \to 0$ のとき）収束する一方、$H(X_1)$ は 1 ビットのままである。$H(X_1)$ はこの連鎖がどれだけよく圧縮できるかについて何を語っているか。
   5. [code] 最初の連鎖を 10^5 ステップシミュレーションし、本文の `adaptive_bits` を用いて次数 0、1、2 で符号化せよ。どの次数がエントロピー率に到達し、また 0 次の符号長は何に収束するか。

7. [code] **典型集合と高確率集合。**●37.2.3.2 節のコードは、長さ $n = 200$ の 2 進文字列（$p = 0.3$ のベルヌーイ情報源からのもの）について、記号あたりのサプライザルが $\delta = 0.1$ ビット以内の誤差でビット単位のエントロピー H の周囲に収まるとき典型的であると判定する。その結果、$2^{188.8}$ 通りの典型文字列が見つかり、これは AEP の命題の上下界 $0.989 \cdot 2^{156.3} \le |A_\delta| \le 2^{196.3}$ の内側にあるものの、$2^{nH} = 2^{176.3}$ を大きく上回ってい

る。

1. 最も確率の高い文字列はどれか。その記号あたりサプライザルを計算し、典型的であるか判定せよ。この文字列が圧縮において問題にならないのはなぜか。
2. A を典型集合とし、B を $P(B) \geq 1-\eta$ を満たす任意の文字列集合とする。$P(A \cap B) \geq 1 - \eta - P(A^c)$ を示し、$|B| \geq \left(1 - \eta - P(A^c)\right) 2^{n(H-\delta)}$ を結論づけよ。
3. 確率が少なくとも 0.9 となる最小の集合は、確率の高い順に文字列を集めることで得られる。本文のコードと同様に 1 の個数 j ごとに文字列をグループ化し、この集合の要素数の $\frac{1}{n}\log_2$ を $n = 200$、1000、5000、20000 に対して計算せよ。これを H および $\frac{1}{n}\log_2 |A_\delta|$（同じ n における値）と比較せよ。2 つの集合のうち 2^{nH} のように増大するのはどちらか。また、δ を固定した典型集合がそうならないのはなぜか。

David Tse、EE376A/Stats376A: Information Theory, Homework 2 の問題 6 を改題。

8. **トークナイザ間のパープレキシティ。** パープレキシティはトークンあたりで計算されるため、テキストがどのようにトークンへ分割されるかに依存する。

1. b^{CE_b} が任意の対数の底 b に対して同じ値になることを示せ（ただし CE_b は底 b の単位でのトークンあたり交差エントロピーとする）。
2. 2 つの言語モデルを、10^6 文字からなる同一のテキスト上で評価する。モデル A のトークナイザはこのテキストを 2.5×10^5 トークンに分割し、A はトークンあたりパープレキシティ 20 を達成する。モデル B のトークナイザは 6.7×10^5 トークンに分割し、B はトークンあたりパープレキシティ 8 を達成する。Equation 37.24 を用いて両方の結果を文字あたりビット数に変換せよ。どちらのモデルがより精度よくテキストを予測しているか。
3. テキストのエントロピー率を文字あたり 1 ビット（英語に対して Shannon (1951) が推定した範囲内の値）とする。Equation 37.23 により、期待されるトークンあたりパープレキシティは少なくとも 2^H である（ただし H はビット単位での**トークン**あたりエントロピー率）。A のトークナイザを用いたとき、および B のトークナイザを用いたとき、任意のモデルが期待値として達成できる最小のパープレキシティはそれぞれいくらか。各モデルは自身の下限からどれだけ離れているか。
4. 2 つのパープレキシティは直接比較できない一方で、2 つの文字あたりビット数は直接比較できる理由を 2 文で説明せよ。

37.3 f ダイバージェンス

本書を通じて、最尤法（maximum likelihood）によってモデルを当てはめるときは常に、データとの一致度を KL ダイバージェンスというただ 1 つの数値で測ってきた。しかし 37.1 節の末尾で見たように、KL ダイバージェンスには決定的な盲点がある。実際に生じる事象にモデルが確率 0 を割り当てた瞬間、その値が無限大へと跳ね上がるのである。分類器がこの問題に直面することはめったにない。ソフトマックス層は厳密な 0 を決して出力しないからである。これに対し、低次元のノイズベクトルを画像空間へ写す生成ネットワークは、直ちにこの問題に突き当たる。生成されるサンプルは画像空間内の薄い曲面上に乗っており、実データを完全に外してしまうことがあるためである。では、2 つの分布を比較する方法には他にどのようなものがあり、ほぼ正しいモデルと大きく外れたモデルを見分けられるのはどの比較法だろうか。

最も単純な例を見るだけでも、何が問題になるかは明らかである。データが 1 点に集中して $P = \delta_0$ となっており、モデルはそこから距離 d だけ離れた 1 点にすべての質量を置いて $Q = \delta_d$ となっているとしよう。直感的には、$d = 0.01$ のモデルはほぼ正しく、$d = 100$ のモデルは大きく外れているのだから、有用な損失であればこの 2 つを区別できるはずである。5 つの標準的な比較法がどのような値を返すかを Table 37.2 に示す。

Table 37.2: 点質量 δ_0 と δ_d の 5 通りの比較。MMD の行では長さスケール 1 の RBF カーネルを用いている。

比較法	$d = 0.01$	$d = 1$	$d = 100$
KL ダイバージェンス $D_{\mathrm{KL}}(P\|Q)$	∞	∞	∞
イェンセン・シャノン・ダイバージェンス	$\log 2$	$\log 2$	$\log 2$
全変動距離	1	1	1
最大平均値差異 MMD^2	0.0001	0.79	2.00
ワッサースタイン距離 W_1	0.01	1	100

この表をよく見ると、5 行のうち 3 行は d が変化しても微動だにしない。これら 3 つのいずれかで訓練されるモデルは、$d = 0.01$ でも $d = 100$ でも同じ損失を受け取るため、d に関する勾配は 0 になる。第 16 章で扱った初期の敵対的生成ネットワーク（generative adversarial network、GAN）はまさにこの窮地に陥っていた。その目的関数は、定数の違いを除けばイェンセン・シャノン・ダイバージェンスの行に帰着するからである。

値が凍りついたこれら 3 つの行は、1 つの同じ族に属している。KL ダイバージェンスは、P と Q をその密度の比 p/q によって比較する。この密度比（density ratio）を保ったまま、そこに適用する凸関数を変えると、**f ダイバージェンス**（f-divergence）と呼ばれる広い族が得られる。この族に属するダイバージェンスはすべて、非負性の証明も、サンプルから推定する手法も、距離 d に対して不感になる構造的な理由も共通している。一方、d に応じて変化する残りの 2 行は密度比を一切使わない。これらについては 37.4 節で学ぶ。

この族を定式化したうえで、本節では深層学習の随所に現れる 3 つの問いに答えていく。第 1 に、KL ダイバージェンスにおいて最尤法のようにデータを第 1 引数に置くか、変分推論（variational inference）のようにモデルを第 1 引数に置くかで違いは生じるのか（37.3.2 節）。第 2 に、ダイバージェンスが小さいことは、P と Q を見分けようとする者にとって実際に何を保証するのか（37.3.3 節）。第 3 に、両方の分布がサンプルを通じてしかわからないとき、それでもなお計算できるものは何か（37.3.4 節）。37.2 節はこの最後の問いで締めくくられており、敵対的訓練（adversarial training）はその答えの上に成り立っている。

本節を通じて対数はすべて自然対数とするため、KL ダイバージェンスおよびイェンセン・シャノン・ダイバージェンスの単位は nat となる。データ分布または目標分布を P、モデルを Q と書き、それぞれの確率質量関数または密度を p と q、その**密度比**を $u = p/q$ と書く。

```
from scipy import integrate, optimize
```

Table 37.2 のすべての項目はコードで直接確かめられる。次のセルでは、δ_0 と δ_d を確率ベクトル $(1, 0)$ と $(0, 1)$ として 2 点からなる台（support）$\{0, d\}$ 上で表し、各比較法をその定義から計算する。これらの定義のうち 3 つは本節で、残りの 2 つは次節で導入する。

```
def kl_div(p, q):
    """D_KL(P||Q) on a finite support: 0 log 0 = 0, and inf if q = 0 < p."""
    s = p > 0
    with np.errstate(divide='ignore'):
        return np.sum(p[s] * np.log(p[s] / q[s]))
```

```
def five_comparisons(atoms, p, q, ell=1.0):
    """KL, JS, TV, RBF MMD^2 and W1 between pmfs p, q on sorted atoms."""
    m = (p + q) / 2                                  # the mixture for JS
    K = np.exp(-(atoms[:, None] - atoms[None, :]) ** 2 / (2 * ell ** 2))
    F_gap = np.abs(np.cumsum(p) - np.cumsum(q))[:-1]  # |F_P - F_Q| between atoms
    return {'KL': kl_div(p, q), 'JS': (kl_div(p, m) + kl_div(q, m)) / 2,
            'TV': np.abs(p - q).sum() / 2, 'MMD^2': (p - q) @ K @ (p - q),
            'W1': np.sum(F_gap * np.diff(atoms))}

delta_0, delta_d = np.array([1.0, 0.0]), np.array([0.0, 1.0])  # on atoms (0, d)
for d in (0.01, 1.0, 100.0):
    vals = five_comparisons(np.array([0.0, d]), delta_0, delta_d)
    print(f'd = {d:6}: ' + '  '.join(f'{k} {v:.4g}' for k, v in vals.items()))
```

```
d =   0.01: KL inf  JS 0.6931  TV 1  MMD^2 0.0001  W1 0.01
d =    1.0: KL inf  JS 0.6931  TV 1  MMD^2 0.7869  W1 1
d =  100.0: KL inf  JS 0.6931  TV 1  MMD^2 2  W1 100
```

37.3.1 f ダイバージェンス族

KL ダイバージェンスを一般化する前に、2 つの分布を比較する尺度が最低限満たすべき性質を明確にしておこう。そうしたうえで、KL ダイバージェンスの非負性の証明が本質的にどの要素を使っているかを問うと、f ダイバージェンス族が自然に浮かび上がる。

37.3.1.1 ダイバージェンスと距離関数

少なくとも「分布間の距離」と呼べるものは、2 つの分布が一致するときに 0 になり、異なるときには真に正の値をとるべきである。この性質を持つ 2 つの分布の関数を**ダイバージェンス**と呼ぶ。

$$D(P,Q) \geq 0, \qquad D(P,Q) = 0 \quad \text{if and only if} \quad P = Q.$$

それ以外の条件は一切課されない。一方、2 つのベクトル間のユークリッド距離 $\|\mathbf{x} - \mathbf{y}\|$（1.3 節）のような**距離関数**（metric）は、さらに $D(P,Q) = D(Q,P)$ という対称性を持ち、三角不等式（triangle inequality）$D(P,R) \leq D(P,Q) + D(Q,R)$ を満たさなければならない。KL ダイバージェンスはダイバージェンスである。実際、ギブスの不等式（37.1 節）により $D_{\mathrm{KL}}(P\|Q) \geq 0$ が保証され、等号が成り立つのは $P = Q$ の場合に限られる。しかし、距離の公理は 2 つとも満たしておらず、1 変量ガウス分布の組を考えればその両方が直ちにわかる。同節で計算したように、2 つのガウス分布間の KL ダイバージェンスは次式で与えられる。

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu_1,\sigma_1^2)\,\big\|\,\mathcal{N}(\mu_2,\sigma_2^2)\big) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \tag{37.25}$$

$P = \mathcal{N}(0,1)$ と $Q = \mathcal{N}(0,4)$ に対してこの式を適用すると、一方の向きでは $D_{\mathrm{KL}}(P\|Q) \approx 0.318$ nat、逆向きでは $D_{\mathrm{KL}}(Q\|P) \approx 0.807$ nat となるため、KL ダイバージェンスは対称ではない。三角不等式が破れる理由はまた別にある。平均が 0、1、2 で分散が 1 の 3 つのガウス分布をとると、隣り合う組に対してはそれぞれ $D_{\mathrm{KL}} = \frac{1}{2}$ となる一方、両端の組に対しては 2 となり、$2 > \frac{1}{2} + \frac{1}{2}$ となってしまう。KL ダイバージェンスは平均間の距離の二**乗**に比例して増大するため、真の長さというよりは二乗ユークリッド距離のように振る舞うのである。

どちらの性質の欠如も修復する必要はない。ダイバージェンスの最小化には対称性も三角不等式も使わないし、KL ダイバージェンスの非対称性はそれ自体が実質的なモデリング上の選択となる（37.3.2 節）。それでも、証明の中で P から Q、さらに Q から R へと比較をつなぐ必要があるときには、三角不等式が不可欠になる。幸い、これから構成する族には真の距離関数も含まれている。

37.3.1.2 f ダイバージェンスのテンプレート

KL を正当なダイバージェンスにしているのはギブスの不等式である。その証明を詳しく見てみよう。そこで使われる 2 つの性質を共有する関数であれば、他のどのような関数からも同様にダイバージェンスが得られるからである。

まず、KL ダイバージェンスを、Q のもとでの密度比 $u(x) = p(x)/q(x)$ の関数の期待値として書き直す。

$$D_{\mathrm{KL}}(P\|Q) = \sum_{x} p(x) \log \frac{p(x)}{q(x)} = \sum_{x} q(x)\, \frac{p(x)}{q(x)} \log \frac{p(x)}{q(x)} = \mathbb{E}_{x\sim Q}\big[u(x)\log u(x)\big].$$

このように見ると、KL ダイバージェンスは各結果を Q で重み付けし、その密度比に関数 $u\log u$ を通じてペナルティを課している。37.1 節から通しで用いている例である離散分布の組 $P = (0.6, 0.3, 0.1)$ と $Q = (0.2, 0.5, 0.3)$ でこの式を確かめてみよう。このとき密度比は $u = 3$、0.6、$\tfrac{1}{3}$ であり、それぞれに重み 0.2、0.5、0.3 が掛かる。

$$\begin{aligned} &0.2\,(3\log 3) + 0.5\,(0.6\log 0.6) + 0.3\,\big(\tfrac{1}{3}\log\tfrac{1}{3}\big) \\ &\qquad = 0.659 - 0.153 - 0.110 = 0.396 \text{ nats}, \end{aligned}$$

これは先ほど求めた KL ダイバージェンスと一致する。

次に、非負性の証明を見てみよう。37.1 節では、$-\log(q/p)$ に対して P のもとでイェンセンの不等式を 1 回適用した。代わりに Q のもとで関数 $u\log u$ に対して同じ 1 ステップの議論を行うと、そこで使われている事実は、関数が凸関数であることと $u = 1$ で 0 に等しいことの 2 つだけである。したがって、$u\log u$ をこの 2 つの性質を持つ任意の関数に置き換えても何ら支障はない。$f : (0,\infty) \to \mathbb{R}$ を $f(1) = 0$ を満たす凸関数とし、$f(0) = \lim_{u\downarrow 0} f(u)$ と定義する（これは $+\infty$ であってもよい）。当面は、$p(x) = 0$ が $q(x) = 0$ なるすべての点で成り立つと仮定し、これを $P \ll Q$ と書く。こうすれば、Q のもとでの期待値は P のすべてを捉えられる。このとき、P の Q に対する **f ダイバージェンス**は次式で定義される (Csiszár 1967)。

$$D_f(P\|Q) = \mathbb{E}_{x\sim Q}\left[f\!\left(\frac{p(x)}{q(x)}\right)\right]. \tag{37.26}$$

各点において、比 $u = p/q$ は P が x を Q に比べてどれだけ過剰または過少に表しているかを測り、f は $u = 1$ のときに 0 になるペナルティを割り当てる。この f を**ダイバージェンス関数**（divergence function）と呼ぶ。多くの文献ではこれをダイバージェンスの生成関数（generator）と呼び、以下のコードでもこれらの関数を `generators` という名前の辞書に格納している。しかし本文では、16.2 節と同様、**生成器**（generator）という語はサンプルを生成するネットワークのために取っておく。

ダイバージェンスがいつ 0 になるかを特定するには、凸性（convexity）を少しだけ強めた条件が必要になる。すなわち、関数 f が 1 **において狭義凸**（strictly convex）であるとは、$(1, 0)$ を通る

ある直線がそれ以外のすべての点で f より真に下側にあること、言い換えれば $f(u) > c\,(u-1)$ がすべての $u \neq 1$ とある傾き c に対して成り立つことをいう。

命題（非負性）。 任意の凸関数 f（ただし $f(1)=0$）および $P \ll Q$ に対し、

$$D_f(P\|Q) \geq 0,$$

が成り立つ。さらに f が 1 において狭義凸ならば、$D_f(P\|Q)=0$ となるのは $P=Q$ のとき、かつそのときに限る。

証明。 イェンセンの不等式（35.3.3.1 節）によれば、$\mathbb{E}[f(U)] \geq f(\mathbb{E}[U])$ が任意の凸関数 f と確率変数 U に対して成り立つ。これを $U = u(x)$（ただし $x \sim Q$）に適用する。証明の鍵はただ 1 つの恒等式、すなわち Q のもとで密度比の期待値が 1 になるという事実 $\mathbb{E}_Q[u] = \sum_x q(x)\,p(x)/q(x) = \sum_x p(x) = 1$ にある。したがって、次が成り立つ。

$$D_f(P\|Q) = \mathbb{E}_Q[f(u)] \geq f(\mathbb{E}_Q[u]) = f(1) = 0.$$

等号成立条件については、支持直線（supporting line）$c\,(u-1)$ として、f より 1 以外のすべての点で真に下側にあるものをとる。すると $D_f(P\|Q) \geq \mathbb{E}_Q[c\,(u-1)] = 0$ となり、その差は $f(u) - c\,(u-1)$ の期待値であって、これは $u \neq 1$ となるすべての点で真に正である。ゆえに $D_f(P\|Q)=0$ ならば、$u(x)=1$ がすべての x（ただし $q(x)>0$）で成り立ち、すなわち $P=Q$ となる。■

この証明の原動力となった恒等式 $\mathbb{E}_Q[u]=1$ からは、以降で繰り返し用いるもう 1 つの便利な帰結が得られる。すなわち、定数倍 $c\,(u-1)$ を f に加えても、$\mathbb{E}_Q[c\,(u-1)]=0$ であるため、いかなるダイバージェンスも変化しない。たとえば関数 $u\log u$ と $u\log u - u + 1$ は、同じ KL ダイバージェンスを定める。

ここまでは $P \ll Q$ を仮定してきたが、冒頭の点質量（point mass）の例はこの条件を満たしていない。生成モデルもこれと同じ状況、すなわち生成サンプルとデータが互いに素な領域を占めた状態から訓練を始めるため、D_f を拡張して q の 0 を扱えるようにしなければならない。$p(x)=q(x)=0$ となる結果からの寄与は 0 とする。$p(x)=0<q(x)$ となる点での寄与は、すでに定義にあるとおり単に $q(x)\,f(0)$ である。$q(x)=0<p(x)$ となる点では比が発散するため、その寄与を $q\,f(p/q)$ の $q \downarrow 0$ における極限として定義する。$t = p/q$ と置けばこれは $p\,f(t)/t$ と書き直され、その極限は $p(x)\,f'(\infty)$（ただし $f'(\infty) := \lim_{t\to\infty} f(t)/t$）となる。凸関数 f に対しては、割線の傾き $f(t)/t$ と導関数 $f'(t)$ が無限遠で同じ極限を持つため、この記法が正当化される。$f(0)$ と $f'(\infty)$ はいずれも $+\infty$ の値をとり得る。

ここで、この規則を点質量 δ_0 と δ_d（ただし $d \neq 0$ は任意）に適用しよう。$x=0$ では $q=0<p=1$ であり、$x=d$ では $p=0<q=1$ であるから、次を得る。

$$D_f(\delta_0\|\delta_d) = f(0) + f'(\infty) \qquad \text{for every } d \neq 0.$$

右辺から d が消えていることに注目してほしい。この式からは、Table 37.2 単独で見たものよりはるかに強い事実がわかる。すなわち、**あらゆる** f ダイバージェンスは互いに素な台の上で定数になる。ニアミスと大きな外れを区別できないため、それを損失として訓練されるモデルは、台どうしを隔てる距離から勾配を一切受け取れないのである。KL ダイバージェンスの場合、$f(u) = u\log u$ から $f(0)=0$ および $f'(\infty) = \lim_{t\to\infty} \log t = \infty$ が得られ、Table 37.2 の 1 行目が再現される。値が凍りついた残りの 2 行も、それぞれのダイバージェンス関数を書き下せば直ちに導かれる。

37.3.1.3 代表的なダイバージェンス関数

Table 37.3 の各行は f の 1 つの選択に対応しており、Figure 37.7 にはそのうち 6 つの曲線を並べて描いている。列 $f''(1)$ は $u=1$ における曲率（curvature）を表しており、次項で中心的な役割を果たす。

Table 37.3: 代表的なダイバージェンス関数。各行は 1 つのダイバージェンスの関数 f、$u=1$ における曲率、そのダイバージェンスが対称か、およびそれ自体（またはその平方根）が距離関数であるかを示す。

ダイバージェンス	$f(u)$	$f''(1)$	対称か	距離関数か
カルバック・ライブラー（順方向）	$u\log u$	1	いいえ	いいえ
逆方向 KL	$-\log u$	1	いいえ	いいえ
ピアソン χ^2	$(u-1)^2$	2	いいえ	いいえ
二乗ヘリンガー d_{H}^2	$(\sqrt{u}-1)^2$	$\frac{1}{2}$	はい	d_{H} が距離関数
全変動	$\frac{1}{2}\lvert u-1\rvert$	なし（角）	はい	はい
イェンセン・シャノン	$\frac{u}{2}\log u - \frac{u+1}{2}\log\frac{u+1}{2}$	$\frac{1}{4}$	はい	$\sqrt{\mathrm{JS}}$ が距離関数
α ダイバージェンス（$\alpha\neq 0,1$）	$\dfrac{u^{\alpha}-\alpha u+\alpha-1}{\alpha(\alpha-1)}$	1	$\alpha=\frac{1}{2}$ のみ	$\alpha=\frac{1}{2}$ で平方根が距離関数

逆方向 KL（reverse KL）は $f(u)=-\log u$ を用いる。というのも、$\mathbb{E}_Q[-\log(p/q)] = \mathbb{E}_Q[\log(q/p)] = D_{\mathrm{KL}}(Q\|P)$ が成り立つからである。したがって、KL ダイバージェンスの引数を入れ替えることは単に別の f を選ぶことにほかならず、両方の向きが同じ表の中に並んでいる。任意の f ダイバージェンスの引数を入れ替える一般的な手順は、演習問題 1a で扱う。

全変動（total variation）は $f(u)=\frac{1}{2}|u-1|$ を用い、これを整理すると $\sum_x q(x)\,\frac{1}{2}|p(x)/q(x)-1| = \frac{1}{2}\sum_x |p(x)-q(x)|$ となる。これを $\mathrm{TV}(P,Q)$ と書く。確率ベクトル間の ℓ_1 距離の半分にあたり、通しで用いている分布の組では $\mathrm{TV}=\frac{1}{2}(0.4+0.2+0.2)=0.4$ となる。この数値が P と Q の識別において何を意味するかは、37.3.3 節で詳しく見る。点質量に対しては、このダイバージェンス関数から $f(0)+f'(\infty)=\frac{1}{2}+\frac{1}{2}=1$ が得られ、Table 37.2 の 3 行目が説明される。

ピアソンの χ^2 ダイバージェンスは放物線 $f(u)=(u-1)^2$ を用い、$\chi^2(P\|Q) = \sum_x q(x)\,(p(x)/q(x)-1)^2 = \sum_x (p(x)-q(x))^2/q(x)$ を与える。通しで用いている分布の組では、その値は $0.16/0.2+0.04/0.5+0.04/0.3=1.0133$ となる。Q が小さい箇所で不一致がどれほど大きく重み付けされるかに注目しよう。$q=0.2$ である第 1 の結果だけで、全体の 0.8 を占めている。

イェンセン・シャノン・ダイバージェンス (Lin 1991) は、両方の分布をその均等な混合分布 $M=\frac{1}{2}(P+Q)$（確率質量関数を m とする）と比較することで KL ダイバージェンスを対称化する。

$$\mathrm{JS}(P,Q)=\frac{1}{2}D_{\mathrm{KL}}(P\,\|\,M)+\frac{1}{2}D_{\mathrm{KL}}(Q\,\|\,M). \tag{37.27}$$

M は P と Q の少なくとも一方が質量を置くすべての点に質量を置くため、比 p/m と q/m が発散することはない。実際、$p/m=2p/(p+q)\le 2$ であるから、各 KL 項は高々 $\log 2$ である。したがって JS は対称かつ有界で $0\le \mathrm{JS}(P,Q)\le \log 2$ を満たし、上限 $\log 2$ に達するのは

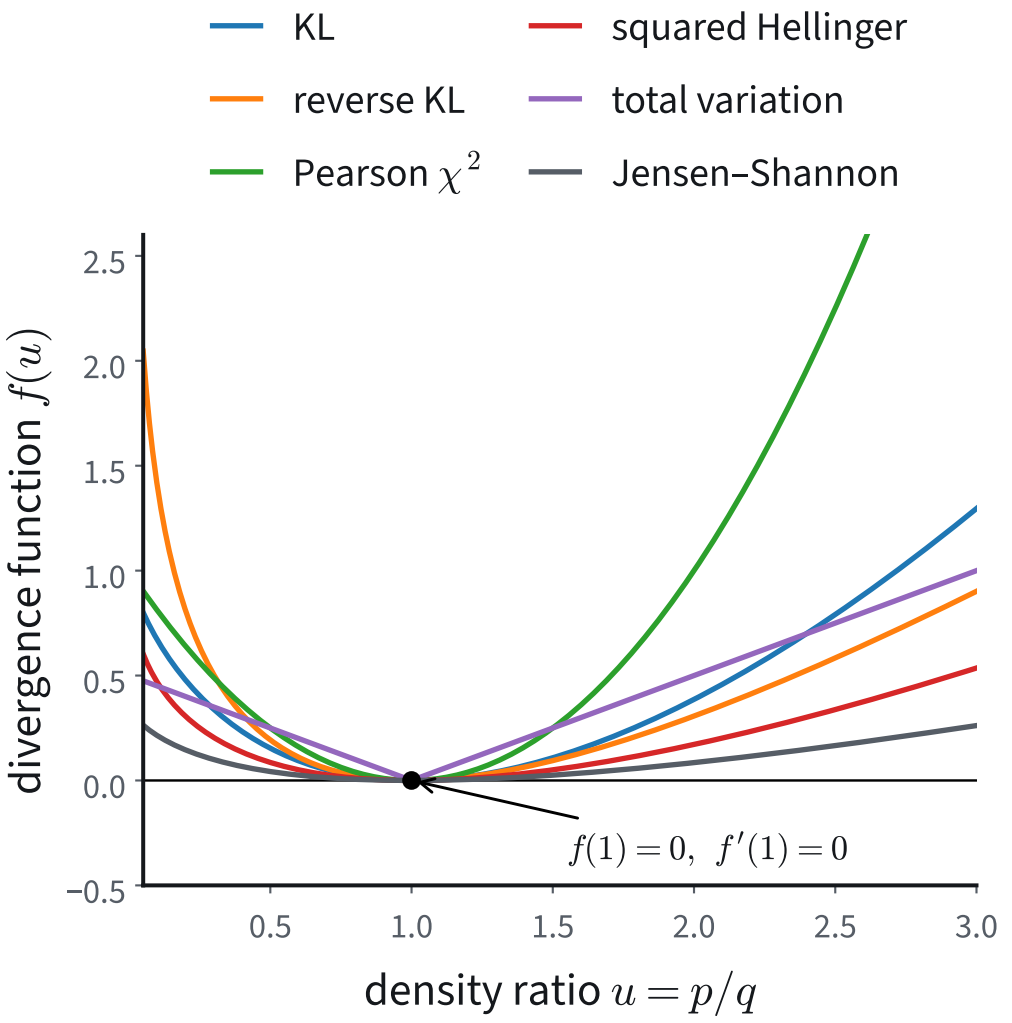


Figure 37.7: 6 つのダイバージェンス関数。滑らかな曲線はそれぞれダイバージェンスを変えない範囲で $f'(1)(u-1)$ だけずらしてあり、すべて $u=1$ で 0 に接している。$u=1$ の近傍では滑らかな曲線は表に示した曲率 $1, 1, 2, \frac{1}{2}, \frac{1}{4}$ を持つ放物線になる一方、全変動は鋭い角を持つ。$u=1$ から離れると不一致に対するペナルティの課し方は大きく異なり、逆方向 KL は $u \to 0$ で発散し、KL と χ^2 は超線形に増大し、全変動は線形に増大する。

P と Q が互いに素な台を持つ場合に限られる（このとき $p/m = 2$ が $p > 0$ なるすべての点で成り立つ）。驚くべきことに、その平方根は真の距離関数になる (Endres and Schindelin 2003)。Equation 37.27 を比 u を用いて書き直すと表のダイバージェンス関数が得られ（演習問題 1c）、これは $f(0) = f'(\infty) = \frac{1}{2}\log 2$ を満たす。その和は $\log 2$ であり、Table 37.2 の 2 行目に一致する。こうして、ただ 1 つの恒等式 $D_f(\delta_0 \| \delta_d) = f(0) + f'(\infty)$ によって、値が凍りついた 3 つの行すべてが説明された。

二乗ヘリンガーダイバージェンス（squared Hellinger divergence）は $d_{\mathrm{H}}^2(P,Q) = \sum_x \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2$ で定義される。その平方根 d_{H} はベクトル $\sqrt{p}$ と $\sqrt{q}$ の間のユークリッド距離にほかならないため、d_{H} は直ちに距離関数となる（演習問題 2b）。

表の最後の行は、この族の中に含まれる連続的な族である α **ダイバージェンス** (Amari 2016) であり、次の関数によって生成される。

$$f_\alpha(u) = \frac{u^\alpha - \alpha u + \alpha - 1}{\alpha(\alpha - 1)}. \tag{37.28}$$

すべての $\alpha \neq 0, 1$ に対して、この関数は $f_\alpha(1) = 0$ と曲率 $f_\alpha''(1) = 1$ を持つ凸関数である。$\alpha \to 1$ では $u\log u - u + 1$ に近づき、$\alpha \to 0$ では $-\log u + u - 1$ に近づく。$u - 1$ の定数倍を加えてもダイバージェンスは変わらないため、これら 2 つの極限から順方向 KL（forward KL）と逆方向 KL が復元される。両者の中間にあたる $\alpha = \frac{1}{2}$ では $2(\sqrt{u} - 1)^2$（二乗ヘリンガーの 2 倍）が得られ、$\alpha = 2$ では χ^2 の半分が得られる。したがって、ただ 1 つのパラメータ α によって、逆方向 KL からヘリンガー、順方向 KL、さらには χ^2 までが滑らかに補間される（演習問題 1d ではこの族をレーニイダイバージェンスと関連づける）。

表の最初の 6 行を、通しで用いている分布の組に対して両方の引数の順序で評価してみよう。KL ダイバージェンスは 0.396 nat と 0.365 nat となって 37.1 節の結果と一致し、TV と $\chi^2(P\|Q)$ も上の手計算と一致するはずである。

```
def f_divergence(f, p, q):
    """D_f(P||Q) = sum_x q(x) f(p(x)/q(x)); nats for KL/JS, else unitless."""
    return np.sum(q * f(p / q))

generators = {
    'KL':          lambda u: u * np.log(u),
    'reverse KL':  lambda u: -np.log(u),
    'chi^2':       lambda u: (u - 1) ** 2,
    'Hellinger^2': lambda u: (np.sqrt(u) - 1) ** 2,
    'TV':          lambda u: 0.5 * np.abs(u - 1),
    'JS':          lambda u: u / 2 * np.log(u) - (u + 1) / 2 * np.log((u + 1) / 2),
}

p = np.array([0.6, 0.3, 0.1])
q = np.array([0.2, 0.5, 0.3])
for name, f in generators.items():
    print(f'{name:12s} D_f(P||Q) = {f_divergence(f, p, q):.4f}   '
          f'D_f(Q||P) = {f_divergence(f, q, p):.4f}')
```

```
KL           D_f(P||Q) = 0.3961   D_f(Q||P) = 0.3653
reverse KL   D_f(P||Q) = 0.3653   D_f(Q||P) = 0.3961
chi^2        D_f(P||Q) = 1.0133   D_f(Q||P) = 0.8000
Hellinger^2  D_f(P||Q) = 0.1862   D_f(Q||P) = 0.1862
```

```
TV          D_f(P||Q) = 0.4000   D_f(Q||P) = 0.4000
JS          D_f(P||Q) = 0.0911   D_f(Q||P) = 0.0911
```

予想どおり KL の行は非対称であり、逆方向 KL の行は単に KL の行の 2 列を入れ替えたものになっている。χ^2 の行も非対称である（1.0133 対 0.8000）。これに対し、二乗ヘリンガー（0.1862）、TV（0.4）、JS はいずれも対称であり、JS は 0.0911 nat と上限 $\log 2 \approx 0.693$ を大きく下回っている。6 つのダイバージェンスが同じ分布の組を見て、0.09 から 1.01 まで 1 桁にわたる値を返したことになる。ところが、KL を 2 倍、二乗ヘリンガーを 4 倍、JS を 8 倍すると、スケールし直した 3 つの値は 0.79、0.74、0.73 となる。密度比のまったく異なる 3 つの関数がなぜほぼ一致するのか、そして係数 2、4、8 はどこから現れたのだろうか。

37.3.1.4 局所二次近似

Figure 37.7 の各曲線は、$u = 1$ から離れた領域ではまったく異なる振る舞いを見せる。しかし $u = 1$ の近傍では、定数倍の違いを除けばただ 1 つの関数へと重なり合う。これはおそらくこの族全体で最も驚くべき性質だろう。本節の以降の内容はこの項に依存しないため、初読時には読み飛ばしてもよい。

1 次項 $c\,(u-1)$ を加えてもいかなるダイバージェンスも変わらないため、値 $f(1) = 0$ も傾き $f'(1)$ も D_f には影響しない。滑らかなダイバージェンス関数どうしを区別する最初の性質は、曲率 $f''(1)$ である。$u = 1$ の近傍では、あらゆる滑らかな f は放物線 $\tfrac{1}{2}f''(1)(u-1)^2$ のように見え、$(u-1)^2$ の Q のもとでの期待値はまさに χ^2 ダイバージェンス $\mathbb{E}_Q[(u-1)^2] = \sum_x (p(x)-q(x))^2/q(x)$ にほかならない。そこで $r = \max_x |u(x)-1|$ により、密度比が 1 からどれだけ離れているかを測ることにする。

命題（局所二次近似）。 **$P \ll Q$、$0 < r < 1$ とし、f は $[1-r, 1+r]$ 上で 3 回微分可能でそこにおいて $|f'''| \le B$ を満たし、さらに $|u(x)-1| \le r$ がすべての x（ただし $q(x) > 0$）で成り立つとする。このとき、次が成り立つ。**

$$\left| D_f(P\|Q) - \frac{f''(1)}{2}\,\chi^2(P\|Q) \right| \le \frac{Br}{6}\,\chi^2(P\|Q). \tag{37.29}$$

証明。 $f(1) = 0$ より、剰余項付きのテイラーの定理から $f(u) = f'(1)(u-1) + \tfrac{1}{2}f''(1)(u-1)^2 + R(u)$（ただし $|R(u)| \le B|u-1|^3/6$）が得られる。項ごとに期待値 $\mathbb{E}_Q$ をとる。1 次項は $\mathbb{E}_Q[u-1] = 0$ により消えるため、異なる f の間の 1 次の違いはすべて打ち消される。2 次項は $\tfrac{1}{2}f''(1)\,\chi^2(P\|Q)$ となり、剰余項は $\mathbb{E}_Q|R| \le \tfrac{B}{6}\,\mathbb{E}_Q|u-1|^3 \le \tfrac{Br}{6}\,\mathbb{E}_Q(u-1)^2 = \tfrac{Br}{6}\,\chi^2$ を満たす。$\blacksquare$

要するに、$D_f \approx \tfrac{1}{2}f''(1)\,\chi^2$ が r に比例する相対誤差のもとで成り立つ。すなわち、P が Q に近いときは常に、滑らかな f ダイバージェンスはすべて互いにスカラー倍の関係にある。先ほど KL、二乗ヘリンガー、JS に掛けた係数 2、4、8 は単に $2/f''(1)$ であり、3 つの積がいずれも近似していた対象は $\chi^2(P\|Q) = 1.013$ である。通しで用いている分布の組ではこれらはまだ 4 分の 1 ほど小さいが、それも無理はない。最大の密度比が 3 であるため $r = 2$ となり、命題の仮定 $r < 1$ の外にあるからである。二次近似が精密になっていく様子を観察するため、P を Q に向かって線分 $P_t = Q + t\,(P - Q)$ に沿って動かしてみよう。するとすべての $u-1$ が t 倍され、$r = 2t$ となる。4 つの滑らかなダイバージェンスについて比 $D_f(P_t\|Q)/(\tfrac{1}{2}f''(1)\,\chi^2(P_t\|Q))$ を出力すると、この比は 1 に向かって t に比例する速さで収束するはずである。

```
    curvature = {'KL': 1.0, 'reverse KL': 1.0, 'Hellinger^2': 0.5, 'JS': 0.25}
    for t in (0.5, 0.1, 0.01):
        p_t = q + t * (p - q)                    # moves P_t toward Q as t -> 0
        chi2 = f_divergence(generators['chi^2'], p_t, q)
        ratios = [f_divergence(generators[name], p_t, q) / (c / 2 * chi2)
                  for name, c in curvature.items()]
        print(f't = {t:4}: ' + '  '.join(f'{name} {r:.3f}'
                                         for name, r in zip(curvature, ratios)))
```

```
t =  0.5: KL 0.844  reverse KL 0.747  Hellinger^2 0.789  JS 0.783
t =  0.1: KL 0.956  reverse KL 0.917  Hellinger^2 0.936  JS 0.935
t = 0.01: KL 0.995  reverse KL 0.990  Hellinger^2 0.993  JS 0.993
```

$t = 0.5$ では、4 つのダイバージェンスはそれぞれの二次近似のおよそ 75% から 85% に位置し、$t = 0.1$ では 91% を超え、$t = 0.01$ では 99% を超える。最後の 2 行の間では t が 10 分の 1 に減少し、1 からの不足分もそれぞれほぼ 10 分の 1 に減少しており、命題の 1 次因子 r と整合している。上界自体は控えめな評価になっている。たとえば $t = 0.1$ における KL では $r = 0.2$ かつ $|f'''(u)| = 1/u^2 \le 1/0.8^2$ であり、上界が許容する相対誤差は約 10% であるのに対し、実際の差はわずか 4% にとどまる。最適化における教訓は実践的である。滑らかな f ダイバージェンスどうしが実質的な違いを見せるのは遠く離れた分布に対してのみであり、それはモデルがまだデータから遠い訓練初期の領域なのである。

パラメトリックなモデル族の内部では、この共通の局所二次形式を支配するのは 36.6 節のフィッシャー情報行列（Fisher information matrix）$\mathbf{I}(\theta) = \mathbb{E}\big[\nabla_\theta \log q_\theta \, \nabla_\theta \log q_\theta^{\top}\big]$ である。微小なパラメータの変位 $\mathbf{h}$ に対し、$q_{\theta+\mathbf{h}}$ の q_θ に対する密度比は $u \approx 1 + \mathbf{h}^{\top}\nabla_\theta \log q_\theta$ と展開される。したがって $\chi^2(q_{\theta+\mathbf{h}} \| q_\theta) = \mathbb{E}[(u-1)^2] \approx \mathbf{h}^{\top}\mathbf{I}(\theta)\,\mathbf{h}$ となり、あらゆる滑らかな D_f はヘッセ行列 $f''(1)\,\mathbf{I}(\theta)$ を $\mathbf{h} = \mathbf{0}$ において持つ。35.5 節の自然勾配（natural gradient）は通常 KL ダイバージェンスから動機づけられるが、どの滑らかな f ダイバージェンスのもとでも同じ更新則を与えることになる。表の中で唯一の例外は全変動である。そのダイバージェンス関数は $u = 1$ に角を持っており二階導関数が存在しないため、それによって挙動がどう変わるかは演習問題 4b で P が Q に近づく極限を通じて調べる。

37.3.2 順方向 KL と逆方向 KL

本書の大半のモデルの背後にある訓練基準である最尤法は、データを第 1 引数に置いた KL ダイバージェンスを最小化する。これに対し、変分推論はモデルを第 1 引数に置いた KL ダイバージェンスを最小化する。これら両方の順序は Table 37.3 に別々の行として現れており、モデル Q_θ（密度を q_θ とする）をモデル族の外にある目標分布 P に当てはめるとき、モデルを正反対の方向へと押しやる。

- **順方向 KL**、すなわち $D_{\mathrm{KL}}(P \,\|\, Q_\theta) = \mathbb{E}_P[\log(p/q_\theta)]$ は、**真の分布**のもとで期待値をとる。$q_\theta(x) \to 0$ が $p(x) > 0$ なる点で生じると常に、ダイバージェンスは発散する。したがって順方向 KL は**ゼロ回避型**（zero-avoiding）、すなわち**モード被覆型**（mode-covering）である。モデルはデータのすべてのモード、つまり p が集中するあらゆる領域を覆うように広がらなければならず、たとえそのために P が決して訪れない空の領域に質量をこぼすことになっても避けられない。
- **逆方向 KL**、すなわち $D_{\mathrm{KL}}(Q_\theta \,\|\, P) = \mathbb{E}_{Q_\theta}[\log(q_\theta/p)]$ は、**モデル**のもとで期待値をとる。ここでは、真の分布が質量を持たない領域にモデルが質量を置くと重いペナルティが課される一方、モデルが無視したモードは目的関数に一切寄与しない。したがって逆方向 KL は**ゼロ強制型**（zero-forcing）、すなわち**モード探索型**（mode-seeking）であり、モデルは目標分布のうちうまく説明できる部分に集中する。

実際には、どちらの向きを選ぶかは、何が実際に計算できるかによって決まることが多い。順方向 KL には P からのサンプルとモデルの密度が必要であり、その最小化は最尤法にほかならない。36.6.2.1 節で示したように、負の対数尤度（negative log-likelihood）の平均は経験分布（empirical distribution）からモデルへの交差エントロピーに等しく、これは順方向 KL と θ に依らない定数だけしか違わない。一方、逆方向 KL にはモデルからのサンプルと目標分布の密度が必要であり、目標密度が正規化定数（normalizing constant）を除いてしかわからない場合でも機能する。変分推論（36.7 節）はまさにこの設定にあり、その目標分布 P は正規化定数の計算が困難な事後分布（posterior）であるため、$D_{\mathrm{KL}}(Q_\theta \| P)$ を計算可能な近似分布 Q_θ にわたって最小化する。

しかし、両方の分布でサンプルと密度がともに利用できる場合には、向きの選択は本質的な設計上の判断となる。教師モデルから蒸留される生徒ネットワーク（26.1 節（英語版））は、教師を第 1 引数に置くと教師が出力し得るあらゆる候補を覆うように広がり、生徒を第 1 引数に置くと教師の最も尤もらしい出力に集中する。同様に、15.3 節の KL ペナルティは学習される方策（policy）を第 1 引数に置くことで、参照方策が決してとらない行動から遠ざけている。

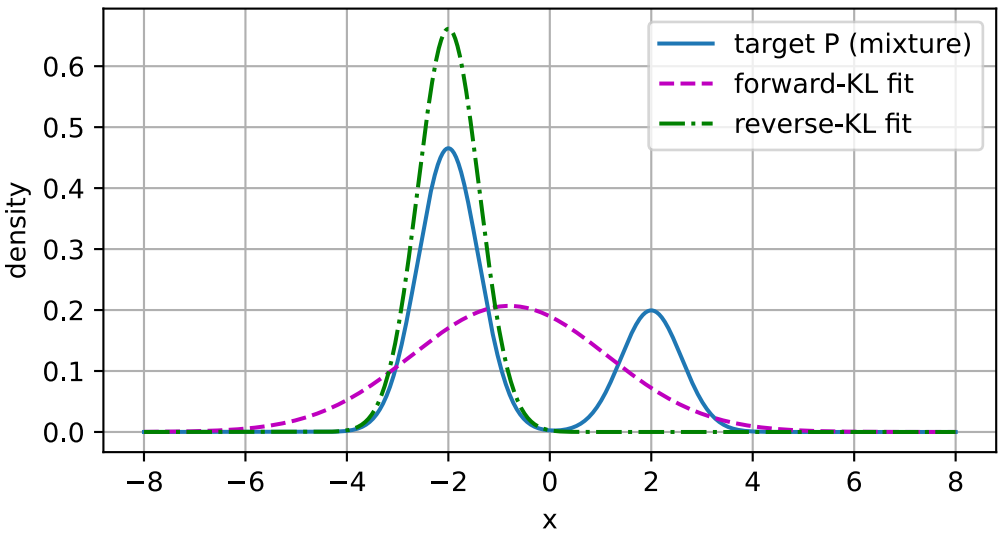
モデル族が P を含んでいれば、どちらの向きでも P が復元される。両者の道が分かれるのはモデルが**誤特定**（misspecified）されているときであり、それを最も明快に示す例が二峰性混合分布への単一ガウス分布の当てはめである。順方向 KL の場合、最適化すら必要ない。順方向 KL は定数から $\mathbb{E}_P[\log q(X)]$ を引いたものに等しいため、その最小点（minimizer）は経験分布を P に置き換えた最尤法によるガウス分布となり、指数分布族（exponential family）における最尤法は十分統計量（sufficient statistics）の期待値を一致させる（36.4.3.2 節）。すなわち、分散が有限である限り $\mu^* = \mathbb{E}_P[X]$ および $(\sigma^*)^2 = \mathrm{Var}_P(X)$ となる。逆方向 KL にはこのような閉形式解（closed-form solution）はなく、各モードの近くに別々の極小値を持つことがある。

二峰性混合分布 $P = 0.7\,\mathcal{N}(-2, 0.6^2) + 0.3\,\mathcal{N}(2, 0.6^2)$ に対しては、順方向 KL による当てはめ結果をあらかじめ計算できる。その平均は $0.7(-2) + 0.3(2) = -0.8$ である。分散は成分内分散と成分間分散の和であり、後者は重み w_1、w_2 と平均間の距離 Δ に対して $w_1 w_2 \Delta^2$ に等しいため、$0.36 + 0.7 \cdot 0.3 \cdot 4^2 = 3.72$ となる。逆方向 KL による当てはめについては、導関数を用いないオプティマイザを $\theta = (\mu, \log\sigma)$ 上で原点と小さい方のモードの近くという 2 つの初期点から走らせ、各 KL をグリッド上の数値積分によって評価する。次のセルでは、それぞれの当てはめについて平均、標準偏差、および両方向のダイバージェンスを出力する。

```
w, mus, sigmas = np.array([0.7, 0.3]), np.array([-2.0, 2.0]), np.array([0.6, 0.6])
x = np.linspace(-8.0, 8.0, 4001)

def log_components(x, w, mus, sigmas):
    """log(w_k N(x; mu_k, sigma_k^2)) for every component k; shape (k, len(x))."""
    return (np.log(w)[:, None] - 0.5 * np.log(2 * np.pi * sigmas[:, None] ** 2)
            - (x - mus[:, None]) ** 2 / (2 * sigmas[:, None] ** 2))

def log_mix(x, w, mus, sigmas):
    """Log-density of the mixture sum_k w_k N(mu_k, sigma_k^2), computed stably."""
    c = log_components(x, w, mus, sigmas)
    c_max = c.max(axis=0)
    return c_max + np.log(np.exp(c - c_max).sum(axis=0))

log_p = lambda x: log_mix(x, w, mus, sigmas)

def log_q(x, theta):
    mu, log_sigma = theta
    return (-0.5 * np.log(2 * np.pi) - log_sigma
```

```
            - (x - mu) ** 2 / (2 * np.exp(2 * log_sigma)))

def kl(log_a, log_b, x):
    """D_KL(A||B) by quadrature on the grid x; in nats."""
    return integrate.trapezoid(np.exp(log_a) * (log_a - log_b), x)

fwd = lambda th: kl(log_p(x), log_q(x, th), x)     # D_KL(P || Q_theta)
rev = lambda th: kl(log_q(x, th), log_p(x), x)     # D_KL(Q_theta || P)

th_fwd = optimize.minimize(fwd, x0=[0.0, 0.0], method='Nelder-Mead').x
th_rev = optimize.minimize(rev, x0=[0.0, 0.0], method='Nelder-Mead').x
th_rev2 = optimize.minimize(rev, x0=[2.5, -0.5], method='Nelder-Mead').x

print(f'{"fit":26s}{"mu":>8s}{"sigma":>7s}'
      f'{"forward KL":>12s}{"reverse KL":>12s}')
for name, th in [('forward-KL fit', th_fwd), ('reverse-KL fit', th_rev),
                 ('reverse-KL fit, 2nd start', th_rev2)]:
    print(f'{name:26s}{th[0]:+8.3f}{np.exp(th[1]):7.3f}'
          f'{fwd(th):12.3f}{rev(th):12.3f}')

d2l.plot(x, [np.exp(log_p(x)), np.exp(log_q(x, th_fwd)),
             np.exp(log_q(x, th_rev))], 'x', 'density',
         legend=['target P (mixture)', 'forward-KL fit', 'reverse-KL fit'],
         figsize=(6, 3))
```

```
fit                           mu  sigma  forward KL  reverse KL
forward-KL fit              -0.800  1.929       0.558       1.274
reverse-KL fit              -1.998  0.603       5.981       0.356
reverse-KL fit, 2nd start   +1.995  0.607      14.534       1.202
```



順方向 KL による当てはめは手計算と一致し、$\mu = -0.800$、$\sigma = 1.929 \approx \sqrt{3.72}$ となる。これは両方のモードにまたがる幅広い釣鐘曲線であり、P がほとんど質量を持たないモード間の谷にもかなりの質量を置いている。1 つ目の逆方向 KL による当てはめは $\mu = -1.998$、$\sigma = 0.603$ に

落ち着き、主要な成分にぴったりと寄り添う一方で小さい方の成分にはほとんど質量を残さない。その逆方向 KL である 0.356 nat は $\log(1/0.7) \approx 0.357$ に近く、これは 70% の成分をあたかも分布全体であるかのように扱うコストにあたる。小さい方のモードの近くから出発すると、オプティマイザは代わりに $\mu = +1.995$ に収束し、その逆方向 KL は 1.202 nat と $\log(1/0.3) \approx 1.204$ に近くなる。したがって、変分近似がどのモードを捉えるかは初期化に依存する。

逆方向 KL はモデルのもとで平均をとるため、モデルが訪れないモードはそもそも一切検査されない。目的関数が気づくのは、選ばれたモードの重みが 0.7（または 0.3）であって 1 ではないということだけである。変分事後分布が不確実性を過小評価しやすいことで知られる理由はまさにここにあり、この例でも逆方向 KL による当てはめの標準偏差は 0.60 であるのに対し、目標分布の標準偏差は 1.93 である。

最後の 2 列は、同一のモデルに対して 2 つの基準がどれほど鋭く対立するかを浮き彫りにしている。最尤法の基準である順方向 KL は、主要モードへの当てはめに 5.98 nat（幅広い当てはめの 0.558 nat の 10 倍以上）、小さい方のモードへの当てはめに 14.5 nat というペナルティを課す。いずれの場合も、取りこぼされたモードはモデルの平均から標準偏差の 6 倍以上離れた位置にあり、そこでは $\log q$ が大きな負の値をとるからである（演習問題 3c）。これに対し、逆方向 KL は最初の 2 つの当てはめに 0.356 nat 対 1.27 と逆の順序をつける。逆方向 KL が許容できないのは、幅広い当てはめがモード間の空の谷に置いた質量なのである。

密度が 0 になる場合の境界規則は、この対比を極限の形で見事に捉えている。順方向 KL では $f(u) = u\log u$ が $f'(\infty) = \infty$ を持つため、P が正の質量を持つ領域にモデルが質量を 0 しか置かないと無限大のコストがかかる一方、$f(0) = 0$ であることは、P が質量を持たない領域にモデルが質量を置いても境界項が一切加わらないことを意味する。逆方向 KL である $f(u) = -\log u$ では、これら 2 つの値がそのまま入れ替わる。

37.3.3 全変動、ピンスカーの不等式、データ処理

あるダイバージェンスが 0.4 に等しいとわかっても、その数値が実際に何を保証するのかを知らなければ、それ自体から読み取れることはほとんどない。全変動についてはその操作的な意味が完全に明快であり、ピンスカーの不等式はその意味の多くを KL ダイバージェンスへと引き継ぐ。

37.3.3.1 全変動

ダイバージェンス関数 $\frac{1}{2}|u-1|$ は Table 37.3 の中で最も単純な項目であり、比の 1 からの絶対距離だけにペナルティを課す。これに対応するダイバージェンスは ℓ_1 距離（p と q の間の距離）の半分であるが、その数値を直ちに解釈可能にする第 2 の同値な表現を持つ。すなわち全変動とは、P と Q が任意の事象 A に割り当てる確率の差の最大値である。

$$\mathrm{TV}(P,Q) = \frac{1}{2}\sum_{x}|p(x)-q(x)| = \sup_{A}|P(A)-Q(A)|. \tag{37.30}$$

もし $\mathrm{TV}(P,Q) = 0.03$ であれば、いかなる事象をもってしても 2 つの分布の確率に 3 パーセントポイントを超える差をつけることはできない。この上限を達成するのは事象 $A^\star = \{x : p(x) > q(x)\}$、すなわち P が Q よりも高い確率を与える結果の集合である。これを確かめるために任意の事象 A をとると、$P(A)-Q(A) = \sum_{x\in A}(p(x)-q(x)) \le \sum_{x\in A^\star}(p(x)-q(x))$ が成り立つ。A から $A^\star$ へ切り替えても、非負の項が加わり非正の項が除かれるだけだからである。さらに $\sum_x (p(x)-q(x)) = 0$ より、$p-q$ の正の部分と負の部分は全体の大きさが等しいため、右辺は $\frac{1}{2}\sum_x|p(x)-q(x)|$ に等しくなり、同じ上界が $Q(A)-P(A)$ にも当てはまる。

通しで用いている分布の組では、$A^\star$ は第 1 の結果のみからなり、先ほど求めたとおり TV $= 0.6 - 0.2 = 0.4$ となる。2 つの連続密度に対する同じ幾何学的構造を Figure 37.8 に示す。全変動は真の距離関数でもある。どちらの表現からも対称性は明らかであり、三角不等式は ℓ_1 の式から直ちに従う（演習問題 2a）。

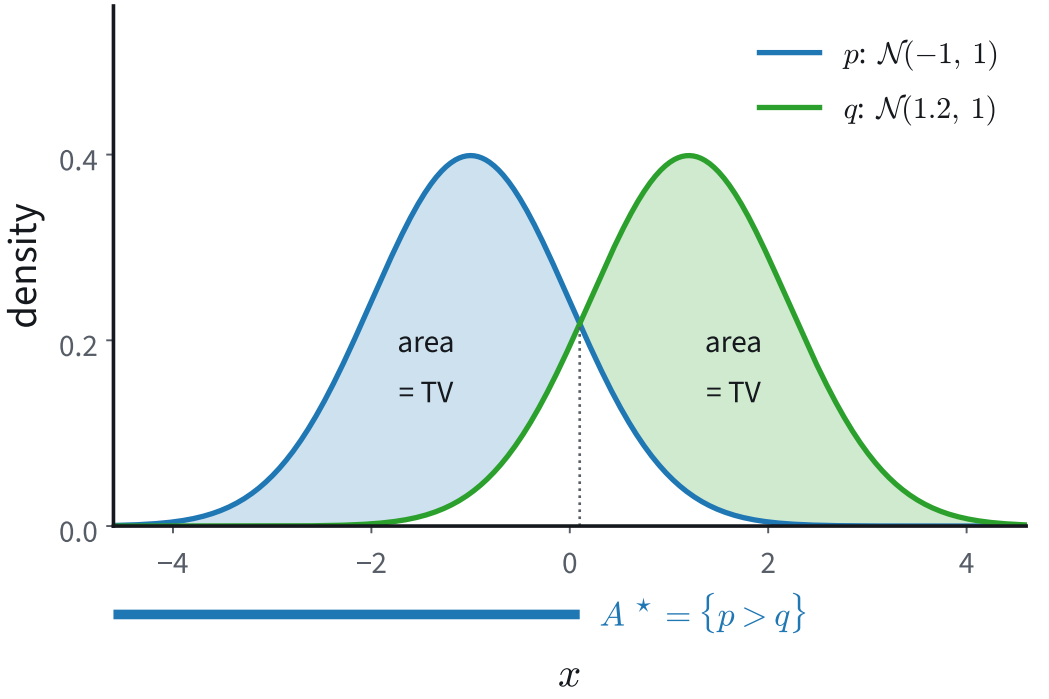


Figure 37.8: 等分散の 2 つの密度とそれらが食い違う領域。$p > q$ となる領域と $q > p$ となる領域は面積が等しく、その共通の面積が全変動距離である。上限を達成する事象は $A^\star = \{p > q\}$（ここでは交点より左の半直線）であり、そこでは P、それ以外では Q と判定するのが最適な 1 サンプル検定となる。

上限による表現は、TV に具体的な統計的意味を与える。等確率 $\frac{1}{2}$ で P または Q から抽出された 1 個のサンプルを検定者に渡し、どちらの分布から生成されたかを当てさせる場面を想像しよう。任意の決定論的な検定は、「P」と答える領域 A を定め、それ以外では「Q」と答えるものであり、次の確率で正解する。

$$\tfrac{1}{2}P(A) + \tfrac{1}{2}(1 - Q(A)) = \tfrac{1}{2}(1 + P(A) - Q(A)) \le \tfrac{1}{2}(1 + \mathrm{TV}(P, Q)),$$

ここで等号は $A = A^\star$ のときに成り立つ。あらゆる検定はなんらかの A に対するこの形をとるため（ランダム化検定は決定論的な検定の平均をとるにすぎず、それを上回ることはできない）、最適な検定者であっても公平なコイン投げより高々 $\mathrm{TV}/2$ だけ高い確率でしか正解できない。通しで用いている分布の組では、最適な検定は第 1 の結果に対して「P」と答え、70% の確率で正解する。GAN の識別器（discriminator、実データと生成サンプルを見分けようとするもの）も機械生成テキストの検出器もこの種の検定者であり、全変動は 1 サンプルから達成できる正解率に厳格な上限を課す。

37.3.3.2 ピンスカーの不等式

全変動はあらゆる統計的検定の限界を与える一方、ニューラルネットワークの損失は対数確率（log-probability）から構築されるため、実際に計算したり制御したりできるのは通常 KL ダイバージェンスである。ピンスカーの不等式はこの両者を橋渡しする。すなわち、KL ダイバージェンスが小さければ、いかなる検定も P と Q を確実に見分けることはできないと保証される。15.2 節における方策の信頼領域（trust region）はその典型例であり、単調改善の保証は全変動を用いて証明されるが、実装上は KL に対する制約を通じて強制される。

命題（ピンスカーの不等式）。 (Pinsker 1964; Csiszár 1967) **有限集合または可算集合上の分布 P と Q に対し、次が成り立つ。**

$$\mathrm{TV}(P,Q) \;\le\; \sqrt{\tfrac{1}{2}\, D_{\mathrm{KL}}(P\|Q)}. \tag{37.31}$$

証明。ステップ 1：2 値の場合。 $\mathrm{Ber}(a)$ で表の出る確率が a のコインを表す。$a \in [0,1]$ および $b \in (0,1)$ に対し、$D_{\mathrm{KL}}(\mathrm{Ber}(a)\|\mathrm{Ber}(b)) = a\log\frac{a}{b} + (1-a)\log\frac{1-a}{1-b}$ かつ $\mathrm{TV}(\mathrm{Ber}(a),\mathrm{Ber}(b)) = |a-b|$ である。したがって、$\rho(a) = D_{\mathrm{KL}}(\mathrm{Ber}(a)\|\mathrm{Ber}(b)) - 2(a-b)^2$ が非負であることを示せばよい。b を固定すると、明らかに $\rho(b) = 0$ であり、1 階導関数 $\rho'(a) = \log\frac{a}{b} - \log\frac{1-a}{1-b} - 4(a-b)$ も $a = b$ で 0 になり、$0 < a < 1$ における二階導関数は次式となる。

$$\rho''(a) = \frac{1}{a(1-a)} - 4 \ge 0,$$

これは $a(1-a) \le \frac{1}{4}$ による。$a = b$ において値と傾きの両方が 0 になる凸関数は $(0,1)$ 上のいたるところで非負であり、連続性（$0\log 0 = 0$ とする）により端点 $a = 0$ および $a = 1$ においても非負である。b が 0 または 1 のときは、$a = b$ でない限り KL ダイバージェンスが無限大になるため、不等式は自明に成り立つ。

ステップ 2：2 値への帰着。 $A^\star = \{p > q\}$ とし、$a = P(A^\star)$、$b = Q(A^\star)$ と置くと $\mathrm{TV}(P,Q) = a - b$ となる。$A^\star$ 内のすべての結果を 1 つのバケットにまとめ、$A^\star$ 外のすべての結果をもう 1 つのバケットにまとめることで標本空間を粗視化すると、KL ダイバージェンスは**減少**する（または変わらない）ことしか起こり得ない。これは対数和不等式（log-sum inequality）から従う。すなわち、非負の数 v_i と w_i に対し、

$$\sum_i v_i \log\frac{v_i}{w_i} \;\ge\; \Big(\sum_i v_i\Big)\log\frac{\sum_i v_i}{\sum_i w_i},$$

が成り立つ。これ自体、イェンセンの不等式を $t \mapsto t\log t$ に 1 回適用したものである。これを $A^\star$ とその補集合にそれぞれ適用すると、

$$D_{\mathrm{KL}}(P\|Q) \;\ge\; D_{\mathrm{KL}}(\mathrm{Ber}(a)\|\mathrm{Ber}(b)) \;\ge\; 2(a-b)^2 = 2\,\mathrm{TV}(P,Q)^2,$$

が得られ、これを整理すれば Equation 37.31 となる。■

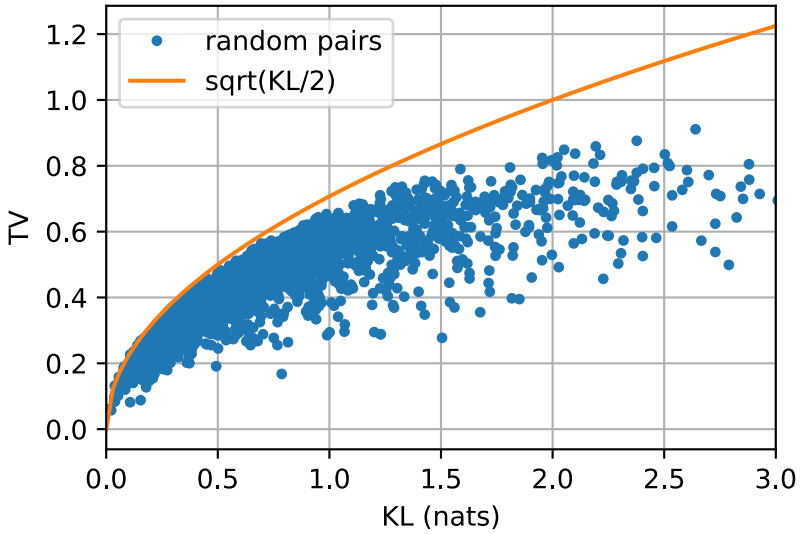
通しで用いている分布の組では、ピンスカーの不等式は $0.4 \le \sqrt{0.396/2} = 0.445$ となり、ステップ 2 を手計算で追うことができる。第 2 と第 3 の結果を統合すると、この組は $(0.6, 0.4)$ と $(0.2, 0.8)$ という 2 枚のコインに縮約される。両者の TV 距離は 0.4 のままであり、KL ダイバージェンスは 0.396 から $0.6\log 3 + 0.4\log\frac{1}{2} = 0.382$ nat へと減少し、ステップ 1 はそれを下から $2\,(0.4)^2 = 0.32$ で評価する。

偏り 0.6 のコインを公平なコインと比較すると、KL は $0.6\log 1.2 + 0.4\log 0.8 = 0.0201$ nat であるのに対し、$2\,(0.1)^2 = 0.02$ となる。つまり、ほぼ等しいコインどうしではピンスカーの不等式はほぼ等号に近い。しかし決定的に重要な点として、有用な逆向きの不等式は存在しない。まれにしか起きないたった 1 つの結果に $q = 0$ を置くモデルは、たとえ TV 距離がごくわずかであっても無限大の KL を持つ。また、TV は決して 1 を超えないため、大きく離れた分布に対してもこの上界は自明なものになる。たとえば平均が 50 だけ離れた分散 1 の 2 つのガウス分布は TV ≈ 1 であるのに対し、Equation 37.25 から KL $= 50^2/2 = 1250$ nat となる。

典型的な分布に対してピンスカーの上界はどれほど精密であり、また定数 $\frac{1}{2}$ を改良することはできるだろうか。第 1 の問いを確かめるため、単体上から一様に $10{,}000$ 組の 5 値分布のペアをサンプリングする。第 2 の問いを確かめるため、2 枚のコインを $a = b = \frac{1}{2}$ の周りで互いに近づける。ステップ 1 はそこでは 2 次の項まで一切の緩みを持たなかったため、比 $\mathrm{TV}/\sqrt{\mathrm{KL}/2}$ は 1 まで登りつめるはずである。

```
rng = np.random.default_rng()
pr, qr = rng.dirichlet(np.ones(5), 10000), rng.dirichlet(np.ones(5), 10000)
tv = 0.5 * np.abs(pr - qr).sum(axis=1)
kl_pq = np.sum(pr * np.log(pr / qr), axis=1)
ratio = tv / np.sqrt(kl_pq / 2)
print(f'TV / sqrt(KL/2) over 10,000 random pairs: median {np.median(ratio):.2f}, '
      f'max {ratio.max():.3f}')
for delta in (0.1, 0.01, 0.001):
    a, b = np.array([0.5, 0.5]), np.array([0.5 + delta, 0.5 - delta])
    print(f'coins 1/2 vs 1/2 + {delta:5}: TV / sqrt(KL/2) = '
          f'{delta / np.sqrt(kl_div(a, b) / 2):.6f}')
grid = np.linspace(0, 3, 100)
d2l.plot([kl_pq[:2000], grid], [tv[:2000], np.sqrt(grid / 2)], 'KL (nats)',
         'TV', legend=['random pairs', 'sqrt(KL/2)'], fmts=('.', '-'),
         xlim=[0, 3], figsize=(4.5, 3))
```

```
TV / sqrt(KL/2) over 10,000 random pairs: median 0.78, max 0.987
coins 1/2 vs 1/2 +   0.1: TV / sqrt(KL/2) = 0.989881
coins 1/2 vs 1/2 +  0.01: TV / sqrt(KL/2) = 0.999900
coins 1/2 vs 1/2 + 0.001: TV / sqrt(KL/2) = 0.999999
```



散布図のすべての点は曲線 $\sqrt{\mathrm{KL}/2}$ の下側に収まっている。中央値にあたるペアは上界の 0.8 付近に位置し、10,000 組の無作為なペアのうち最も精密なものは 1 から数パーセント以内に迫る。KL > 2 になると曲線は 1 を超え、上界は自明になる。ほぼ公平なコインに対しては、比は 0.9899、0.9999、0.999999 と上昇し、定数 $\frac{1}{2}$ をこれ以上厳しくできないことが確認できる（演習問題 4c）。Pinsker の元の証明では定数がこれより弱く、最良の係数 $\frac{1}{2}$ は Csiszár (1967)、およびそれとは独立に Kullback と Kemperman によって得られた。

37.3.3.3 データ処理

特徴抽出器や量子化器、あるいはその他の後処理ステップによって、2 つの分布をより見分けやすくすることはできるだろうか。ピンスカーの不等式の証明では結果を統合すると KL は減少するほかなく、37.1.2.4 節の連鎖律でもペアの一方の変数を捨てると同じことが起きた。同じ原理があらゆる f ダイバージェンスとあらゆる形態の処理に対して成り立つ。**通信路**（channel）$\kappa(y \mid x)$ とは、決定論的かランダム化されているかを問わず、入力 x を出力 y へ変換する任意の規則のことである。P をこの通信路に通すと出力分布 $(P\kappa)(y) = \sum_x \kappa(y \mid x)\, p(x)$ が生じ、Q を通すと同じように $Q\kappa$ が生じる。

命題（f ダイバージェンスに対するデータ処理不等式）。 任意の通信路 κ、凸関数 f（ただし $f(1) = 0$）、および有限集合または可算集合上の $P \ll Q$ に対し、次が成り立つ。

$$D_f(P\kappa \,\|\, Q\kappa) \;\le\; D_f(P\|Q). \tag{37.32}$$

証明。$\sum_y \kappa(y \mid x) = 1$ より、$D_f(P\|Q) = \sum_y \sum_x \kappa(y \mid x)\, q(x)\, f\!\left(p(x)/q(x)\right)$ と展開できる。出力 y として $(Q\kappa)(y) > 0$ を満たすものを固定する。正規化された重み $\kappa(y \mid x)\, q(x)/(Q\kappa)(y)$ は x に関して和をとると 1 になり、この重みのもとで密度比 $p(x)/q(x)$ の期待値をとると $(P\kappa)(y)/(Q\kappa)(y)$ が得られる。したがって内側の和にイェンセンの不等式を適用すると、下から $(Q\kappa)(y)\, f\!\left((P\kappa)(y)/(Q\kappa)(y)\right)$ で抑えられ、これは y 番目の項として $D_f(P\kappa\|Q\kappa)$ に現れるものにほかならない。出力 y が $(Q\kappa)(y) = 0$ となる場合には、$\kappa(y \mid x) = 0$ が $q(x) > 0$ なるすべての点で成り立つため $(P\kappa)(y) = 0$ でもあり、両辺のいずれにも寄与しない。最後に y にわたって和をとれば証明が完了する。■

どのダイバージェンス関数を選んでも、データを処理することによって 2 つの分布がより識別しやすくなることは決してない。深層学習では、この不等式を逆向きに読むことが多い。すなわち、学習された特徴量、量子化された出力、あるいはその他のデータ要約の上で測ったダイバージェンスは、常に元の入力間のダイバージェンスの下界になる。ピンスカーの不等式の証明で結果を統合した操作は、単に決定論的な通信路 $x \mapsto \mathbf{1}(x \in A^\star)$ にほかならない。通しで用いている分布の組では、この操作により KL は 0.396 から 0.382 へと下がり、χ^2 も 1.013 から $0.16/0.2 + 0.16/0.8 = 1$ へと下がる。また、KL の特別な場合は 37.6.2.2 節における相互情報量（mutual information）のデータ処理不等式（data-processing inequality）の原動力でもある。

可逆な変換は両方向の通信路として働くため、Equation 37.32 において等号を強制する。すなわち、あらゆる f ダイバージェンスは結果のラベルの付け替えや座標軸のスケール変換に対して不変である。対照的に、Table 37.2 の最後の 2 行は軸のスケールを変えると値が変わる。それらの距離は、d に対する幾何学的な感度を得る引き換えに、スケール不変性を手放しているのである。

37.3.4 フェンシェル共役とクリティック下界

分布 Q からサンプルを生成する生成ネットワークと、真の分布 P からのサンプルからなるデータセットが手元にあるとしよう。どちらのサンプル集合からも、Equation 37.26 が必要とする密度比 $p(x)/q(x)$ を直接知ることはできない。それでもサンプルだけから D_f を推定できるだろうか。実は、任意の f ダイバージェンスは P と Q のもとでの期待値だけを用いて下から評価でき、その下界によってあらゆるダイバージェンスが敵対的訓練の目的関数へと変わる。

37.3.4.1 下界の導出

中心となる道具は、35.7.4 節で導入した凸共役（convex conjugate）である。凸関数 f に対し、その共役は次式で定義される。

$$f^*(t) = \sup_{u} \big(ut - f(u) \big),$$

すなわち $-f^*(t)$ は、f の傾き t の接線の縦軸切片となる（Figure 37.9）。$f(u) = u \log u$ の場合、上限は $\log u + 1 = t$ の点で達成され、$f^*(t) = e^{t-1}$ となる。凸関数は常にそのすべての接線の上側にあるため、フェンシェル・ヤングの不等式が得られる。

$$f(u) \geq ut - f^*(t) \quad \text{for all } t, \tag{37.33}$$

ここで等号は、t が f の点 u における傾きと一致するとき、すなわち $t = f'(u)$ のときに常に成り立つ。

ここで Equation 37.33 を**ダイバージェンスの内部で各点ごとに**適用し、傾きが x に応じて変化できるようにしよう。具体的には、**クリティック**と呼ばれる任意の関数 $T(x)$ をとり、$u = p(x)/q(x)$ および $t = T(x)$ を代入する。

命題（クリティック下界）。 **f を $f(1) = 0$ を満たす微分可能な凸関数とし、$P \ll Q$ とし、T を $\mathbb{E}_P|T| < \infty$ かつ $\mathbb{E}_Q|f^*(T)| < \infty$ を満たす任意のクリティックとする。このとき、次が成り立つ。**

$$D_f(P\|Q) \;\geq\; \mathbb{E}_{x\sim P}[T(x)] - \mathbb{E}_{x\sim Q}[f^*(T(x))]. \tag{37.34}$$

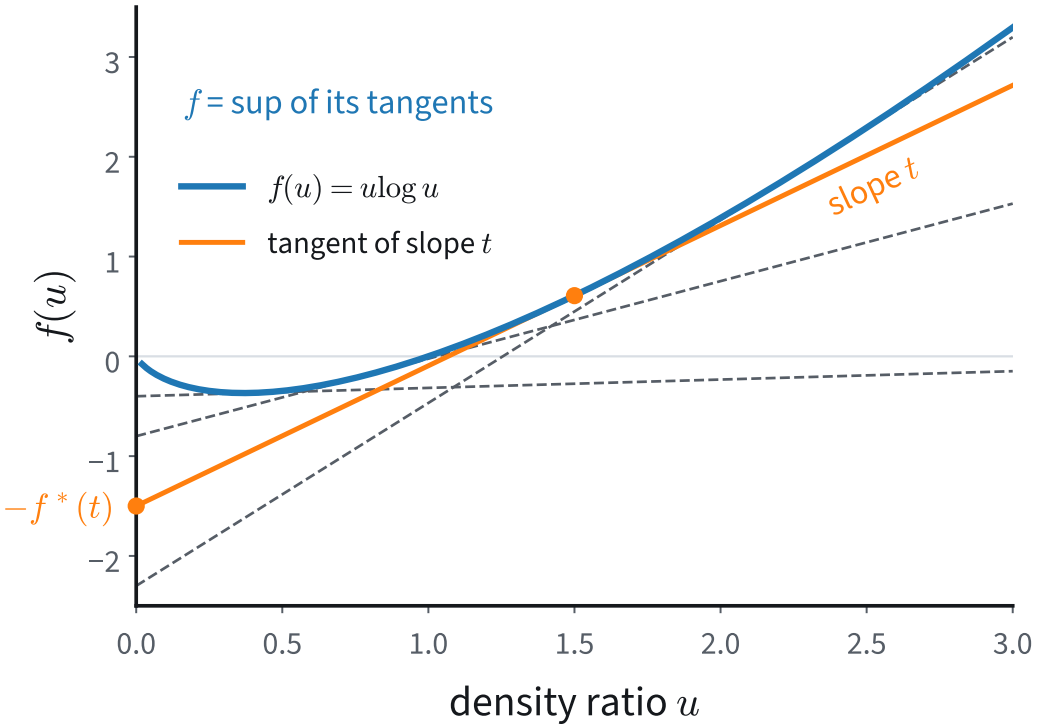


Figure 37.9: 凸なダイバージェンス関数（ここでは $f(u) = u \log u$）は、その接線群の上側包絡線である。傾き t の接線は縦軸切片 $-f^*(t)$（ここでは $-e^{t-1}$）を持つ。1 本の接線だけを残せばフェンシェル・ヤングの下界が得られ、傾きにわたって上限をとれば $f(u)$ が復元される。この不等式を密度比に各点ごとに適用するとクリティック下界が得られる。

さらに、すべてのクリティックにわたる右辺の上限は $D_f(P\|Q)$ に等しく、$T^\star(x) = f'(p(x)/q(x))$ によって $T^\star$ が有限となるすべての点で達成される (Nguyen et al. 2010; Nowozin et al. 2016)。

証明。 Equation 37.33 に $u = p(x)/q(x)$ と $t = T(x)$ を代入し、両辺に $q(x) \geq 0$ を掛ける。

$$q(x)\, f\!\left(\frac{p(x)}{q(x)}\right) \;\geq\; p(x)\, T(x) - q(x)\, f^*(T(x)).$$

x にわたって和（または積分）をとれば Equation 37.34 が得られる。フェンシェル・ヤングの不等式は $t = f'(u)$ で等号が成り立つため、$T^\star(x) = f'(p(x)/q(x))$ と選べばすべての x で不等式が等式になり、したがって期待値においても等号が成り立つ。■

Equation 37.34 の右辺にある 2 つの項は、実データにわたる T の期待値と生成サンプルにわたる $f^*(T)$ の期待値という単純な期待値であり、明示的な密度を一切必要としない。したがって T をニューラルネットワークとしてパラメータ化し、ミニバッチ上で訓練してこの下界を最大化できる。KL ダイバージェンスの場合、この下界はすべてのクリティックに対して $D_{\mathrm{KL}}(P\|Q) \geq \mathbb{E}_P[T] - \mathbb{E}_Q[e^{T-1}]$（任意の T）となる。Nowozin et al. (2016) による **f-GAN**（これにちなんで Equation 37.34 は f-GAN 下界とも呼ばれる）は、この変分表現を 2 人ゲームへと変換する。すなわち、クリティックは下界を最大化してダイバージェンスを追跡させ、Q を生成する生成器はそれを最小化する。このようなペアの実装と訓練は 16.2 節で扱う。

37.3.4.2 初期の GAN

Goodfellow et al. (2014) による初期の GAN は、この枠組みにおいて $f(u) = u\log u - (u+1)\log(u+1)$ と選ぶだけでそのまま導かれる。その凸共役は $f^*(t) = -\log(1 - e^t)$（ただし $t < 0$）である。クリティックを識別器 $D(x) \in (0,1)$ の対数として $T = \log D$ と書けば、クリティック下界はおなじみの二値交差エントロピー（binary cross-entropy）のゲームになる。

$$\mathbb{E}_P[T] - \mathbb{E}_Q[f^*(T)] = \mathbb{E}_{x\sim P}[\log D(x)] + \mathbb{E}_{x\sim Q}[\log(1 - D(x))].$$

最適クリティック $T^\star = f'(p/q) = \log\big(p/(p+q)\big)$ は、ベイズ最適な識別器 $D^\star = p/(p+q)$ に対応する。イェンセン・シャノンのダイバージェンス関数 f_{JS}（Table 37.3）と比較すると、$f(u) = 2f_{\mathrm{JS}}(u) - (u-1)\log 2 - \log 4$ であることがわかる。すなわち、イェンセン・シャノン関数の 2 倍に、$u-1$ の 1 次項（ダイバージェンスを変化させない）と定数 $f(1) = -\log 4$（値を平行移動させるだけである）を加えたものになっている。したがって $D^\star$ において、このミニマックスゲームは $2\,\mathrm{JS}(P,Q) - \log 4$ を達成し、通しで用いている分布の組ではその値は $2(0.0911) - 1.386 = -1.204$ に等しい。

37.3.4.3 下界の限界

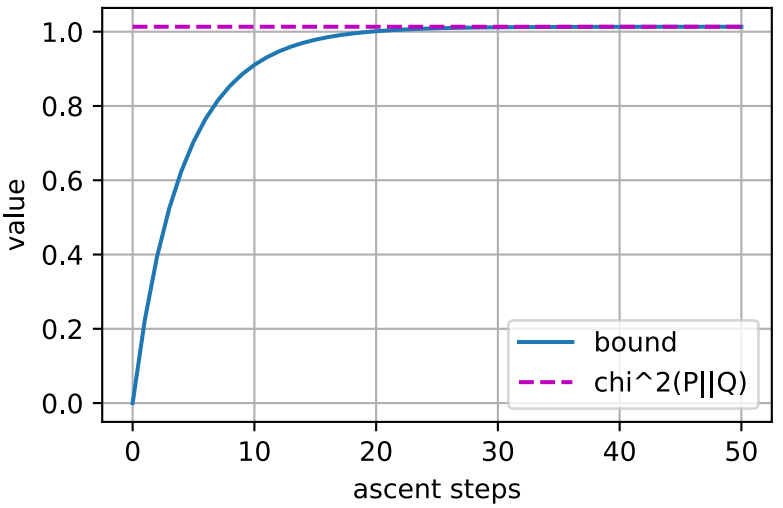
実際には、クリティック下界は 2 つの限界に直面し、そのうち第 1 の限界は直接測定できる。下界が等号を達成するのは $T^\star$ のみであるため、訓練が不十分なクリティックは真のダイバージェンスより小さい値を返すのである。クリティックが訓練を進めるにつれてその差をどう縮めていくかを観察してみよう。

通しで用いている離散分布の組では、期待値は単純な和であり、クリティックは各結果に 1 つずつ対応する 3 つの数値からなるベクトルにすぎない。χ^2 ダイバージェンスをとり、$f(u) = (u-1)^2$、共役 $f^*(t) = t + t^2/4$、最適クリティック $T^\star = 2(p/q - 1)$ としよう（ここでは共役の上限をすべての実数 u にわたってとっている。演習問題 6a でこれを導出し、$T^\star$ において何も損

なわれないことを確かめる）。$T = 0$（このとき下界は 0）から出発し、ステップサイズ 1 で下界に対して勾配上昇法（gradient ascent）を実行する。この下界の $T(x)$ に関する導関数は $p(x) - q(x)\,(1 + T(x)/2)$ である。命題により、この勾配上昇は下から $\chi^2(P\|Q) = 1.0133$ に向かって登っていかなければならない。妥当性確認として、最後の行では正しい向きを指しつつ 2 倍だけ行き過ぎた過信クリティック $2T^\star$ も評価する。

```
def fgan_bound(T, p, q, f_star):
    """E_P[T] - E_Q[f*(T)]: a lower bound on D_f(P||Q) for any critic T."""
    return np.sum(p * T) - np.sum(q * f_star(T))

chi2_star = lambda t: t + t ** 2 / 4    # conjugate of f(u) = (u - 1)^2
exact = f_divergence(generators['chi^2'], p, q)
T, bounds = np.zeros(3), []             # untrained critic: one number per outcome
for step in range(51):
    bounds.append(fgan_bound(T, p, q, chi2_star))
    T = T + (p - q * (1 + T / 2))       # ascent step 1 along p - q f*'(T)
shown = (0, 1, 2, 5, 10, 20, 50)
print(f'exact chi^2(P||Q) = {exact:.4f}')
print('steps ' + ''.join(f'{s:8d}' for s in shown))
print('bound ' + ''.join(f'{bounds[s]:8.4f}' for s in shown))
T_star = 2 * (p / q - 1)                # the optimal critic f'(p/q)
print(f'overconfident critic 2 T*: bound = '
      f'{fgan_bound(2 * T_star, p, q, chi2_star):.4f}')
d2l.plot(np.arange(51), [bounds, [exact] * 51], 'ascent steps', 'value',
         legend=['bound', 'chi^2(P||Q)'], figsize=(4.5, 2.8))
```

```
exact chi^2(P||Q) = 1.0133
steps        0       1       2       5      10      20      50
bound   0.0000  0.2240  0.3935  0.7036  0.9107  1.0013  1.0133
overconfident critic 2 T*: bound = 0.0000
```



5 ステップ後、クリティックは 0.70（真のダイバージェンス 1.01 の約 70%）を下界として保証し、10 ステップ後には 90%、20 ステップ後には 99% に達する。どの時点でも下界が真の値を上回ることはない。過信クリティック $2T^\star$ でさえ 0 を返し、未訓練のクリティックと変わらない。

期待値が厳密に計算できる限り、敵対的なダイバージェンス推定値は常に過小評価であり、その数値自体からはどれだけ取りこぼしているかは決してわからない。期待値を有限標本の平均に置き換えた場合、それが（期待値の意味で）正当な下界であり続けるのは、クリティックの訓練に使われていない新たなサンプル上で評価したときに限られる。

勾配更新の式からは、取りこぼされたダイバージェンスがどこに潜んでいるかもわかる。$T(x)$ は $q(x)$ に比例する速さで動くため、モデルのサンプルが最もまれな箇所でクリティックの学習は最も遅くなる。通しで用いている分布の組ではそれは第 1 の結果であり、まさに密度比が最大でダイバージェンスの大半が集中している場所である（演習問題 6b）。

第 2 の限界は、冒頭の互いに素な台の問題へと、今度はクリティックの視点から立ち返らせる。$T^\star = f'(p/q)$ を仮定 $P \ll Q$ の外へ形式的に拡張すると、密度比はデータの台の上では ∞、モデルの台の上では 0 に等しくなるため、$T^\star$ は前者で定数値 $f'(\infty)$、後者で定数値 $f'(0)$ をとる。生成器は自身のサンプル上で評価したクリティックの勾配を用いてパラメータを更新するため、そこで局所的に定数となっているクリティックからは勾配が 0 しか得られない。生成器には、どちらの方向へ動けばよいかを知らせる信号が一切届かないのである。

これら 2 つの限界はいずれも同じ根源、すなわち密度比 p/q に由来している。有限個のサンプルから p/q が直接手に入ることは決してなく、台が離れるやいなやこの比は 0 または ∞ へと潰れ、2 つの分布がどれだけ離れているかという手がかりをすべて消し去ってしまう。37.4 節では、密度比に一切頼らない比較法へと目を向ける。

37.3.5 まとめ

KL ダイバージェンスは広い族の 1 つの要素にすぎない。一般に f ダイバージェンスは、密度比 p/q の凸関数の Q のもとでの期待値をとるものである。イェンセンの不等式によりすべての要素の非負性が保証され、滑らかな要素はすべて $f''(1)$ という曲率の定数倍を除いて P と Q が近いときに一致し、いかなるデータの後処理も f ダイバージェンスを増加させることはない。KL の向きの選択はモデリング上の判断である。順方向 KL（最尤法）はすべてのモードを覆うように広がる一方、逆方向 KL（変分推論）は単一のモードに集中し得る。操作的には、P と Q を見分ける最良の 1 サンプル検定は確率 $\frac{1}{2}(1 + \mathrm{TV})$ で成功し、ピンスカーの不等式は TV を上から $\sqrt{D_{\mathrm{KL}}/2}$ で抑える。また凸共役を用いれば、任意のクリティックによって D_f を P と Q にわたる期待値のみから下界評価でき、敵対的訓練はこの仕組みによってサンプルからダイバージェンスを推定している。それでもこの族のすべての要素は、密度比が持つ同じ盲点を共有している。互いに素な台の上では、台どうしがどれほど近くても遠くても、D_f は定数 $f(0) + f'(\infty)$ に凍りついてしまうのである。

37.3.6 演習問題

1. **ダイバージェンス関数と引数の順序。** Table 37.3 の各行は、テンプレート中の f（Equation 37.26）の 1 つの選択に対応している。
   1. ダイバージェンス関数 f に対し、$f^\diamond(u) = u\, f(1/u)$ と定義する。$f^\diamond$ が $f^\diamond(1) = 0$ を満たす凸関数であること、および $D_f(Q\|P) = D_{f^\diamond}(P\|Q)$ が成り立つことを示せ。さらに $f^\diamond$ を KL 関数 $u \log u$ と χ^2 関数 $(u-1)^2$ に対して計算し、後者が定めるダイバージェンスを x に関する和として書き下せ。
   2. $c\,(u-1)$ を f に加えてもダイバージェンスは変化しない。D_f は、$f^\diamond(u) = f(u) + c\,(u-1)$ がなんらかの定数 c に対して成り立つとき常に対称であることを示せ。Table 37.3 で対称と記されている各行についてこの条件を確かめ、さらに $D_{f_\alpha}(Q\|P) = D_{f_{1-\alpha}}(P\|Q)$ が関数 Equation 37.28 に対して成り立つことを示せ。この恒等式を用いて、α ダイバージェンスの行の「対称か」列の記述を確かめよ。

3. Equation 37.27 を比 $u = p/q$ を用いて展開し、Table 37.3 のイェンセン・シャノン関数を導出せよ。
4. **レーニイダイバージェンス。** $\alpha > 0$、$\alpha \neq 1$ に対し、$D_\alpha(P\|Q) = \frac{1}{\alpha-1}\log\sum_x p(x)^\alpha\, q(x)^{1-\alpha}$ と定義する (Rényi 1961)。$\sum_x p^\alpha q^{1-\alpha} = 1 + \alpha(\alpha-1)\, D_{f_\alpha}(P\|Q)$ であり、D_α が D_{f_α} の単調増加関数であることを示せ。ここから $D_\alpha \geq 0$ であること、および D_α がデータ処理不等式 Equation 37.32 を満たすことを導け。さらに $\lim_{\alpha\to 0} D_\alpha(P\|Q)$ を求め、それが 0 になる（p と q が同じ結果の集合上で正であるとき）理由を説明せよ。したがって、レーニイ族が逆方向 KL に $\alpha \to 0$ で到達**しない**一方、D_{f_α} は到達するのはなぜか。

2. **ダイバージェンスの中の距離関数。** Table 37.3 の最後の列には、どのダイバージェンスが距離関数であるかが記されている。
1. TV に対する三角不等式を、ℓ_1 距離の半分という表現から証明せよ。
2. ヘリンガー距離 $d_{\mathrm{H}} = \sqrt{d_{\mathrm{H}}^2}$ がベクトル $\left(\sqrt{p(x)}\right)_x$ と $\left(\sqrt{q(x)}\right)_x$ の間のユークリッド距離であり、したがって d_{H} が距離関数であることを示せ。また、2 値上の 3 つの分布であって、d_{H}^2 と JS の両方が三角不等式に反するものを求めよ。

3. **モードを取りこぼすコスト。** モデル Q がデータ分布 P の一部を取りこぼしている状況を考える。
1. $P = (1-w)\,Q + wR$（ただし $0 < w < 1$）とし、R はある集合（それに対して Q が確率 0 を割り当てるもの）上の分布とする。$D_{\mathrm{KL}}(P\|Q)$、$D_{\mathrm{KL}}(Q\|P)$、$\mathrm{TV}(P,Q)$、$d_{\mathrm{H}}^2(P,Q)$ を w の関数として計算せよ。取りこぼされた質量にペナルティを課すのはどちらの向きの KL か。また、有限の値をとるダイバージェンスは $w \to 0$ でどのように振る舞うか。
2. 次に $P = w_1\mathcal{N}(m_1, s^2) + w_2\mathcal{N}(m_2, s^2)$（平均間の距離を $\Delta = |m_2 - m_1|$ とする）とし、モデル $Q = \mathcal{N}(m_1, s^2)$ は第 1 成分のみを保持するものとする。$D_{\mathrm{KL}}(Q\|P) \leq \log(1/w_1)$ がすべての Δ に対して成り立ち、極限 $\Delta \to \infty$ で等号となることを示し、この極限を (a) と関連づけよ。さらにこの上界を用いて、混合分布へのガウス分布の当てはめで出力された 0.356 nat と 1.202 nat という 2 つの逆方向 KL の値を説明せよ。
3. 2 つの成分が十分に離れており、ほとんどすべての x において無視できない密度を持つ成分が高々 1 つであると仮定する。このとき $D_{\mathrm{KL}}(P\|Q) \approx w_2\Delta^2/(2s^2) + w_1\log w_1 + w_2\log w_2$ となることを示せ。この近似式を先ほどの実験の混合分布に対して計算し、`kl(log_p(x), log_q(x, [-2.0, np.log(0.6)]), x)` で確かめよ。実験の順方向 KL の列には比較対象となる 2 つの値が出力されている。すなわち、5.981 nat（パラメータが $(-1.998, 0.603)$ であり厳密な $(-2, 0.6)$ とはわずかに異なる逆方向 KL の当てはめにおける値）と、0.558 nat（順方向 KL の当てはめにおける値）である。

4. **局所二次の領域。** 命題 Equation 37.29 は D_f と $\frac{1}{2}f''(1)\,\chi^2$ の差を上から評価している。
1. 演習問題 7b（37.1 節）では、ガウス分布間の KL ダイバージェンスの 2 つの向きが初めて食い違う次数を求めた。KL 関数と逆方向 KL 関数のどの性質がその次数を説明するか。証明のテイラー展開をもう 1 項先まで進め、局所二次近似の実験が出力する比が $1 + c\,t\,f'''(1)/f''(1) + O(t^2)$（ただし $c = \mathbb{E}_Q[(u-1)^3]/(3\chi^2(P\|Q))$）であり、通しで用いている分布の組では約 0.49）となることを示せ。4 つの関数に対して $f'''(1)/f''(1)$ を計算せよ。$t = 0.01$ において逆方向 KL の不足分が KL の 2 倍になり、また二乗ヘリンガーと JS が同じ値を出力するのはなぜか。
2. 全変動の関数 $\frac{1}{2}|u-1|$ は 1 において二階導関数を持たない。$P_t = Q + t\,(P-Q)$（ただし $t \in (0,1]$）と置くとき、$\mathrm{TV}(P_t, Q)$ と $\chi^2(P_t\|Q)$ は t に対してどのようにスケールするか。このことから、ピンスカーの不等式 Equation 37.31 が TV を KL の定数倍ではなく平方根で抑える理由を説明せよ。
3. P と Q を表の出る確率が b と $b+\delta$ のコインとする。命題を用いて $D_{\mathrm{KL}}(P\|Q) =$

$\delta^2/(2b(1-b)) + O(\delta^3)$ を示せ。演習問題 5a（37.2 節）では、ベルヌーイ分布の KL ダイバージェンスから直接同じ展開を得ている。因子 $1/(b(1-b))$ はこのコインのフィッシャー情報量である（36.6 節）。$\mathrm{TV}/\sqrt{D_{\mathrm{KL}}/2}$ の $\delta \to 0$ における極限を求め、この極限でピンスカーの不等式が等号を達成する b の値を求めよ。ピンスカーの不等式に続くコインの実験では $b = \frac{1}{2}$ をとり、比として 0.999999 が $\delta = 10^{-3}$ で出力されている。この比が 1 に近づく速さが δ^2 であって δ ではないのはなぜか。

5. **ヘリンガーによる評価と検定。**●ピンスカーの不等式は $D_{\mathrm{KL}} > 2$ になると自明になる。$\mathrm{BC}(P,Q) = \sum_x \sqrt{p(x)q(x)} = 1 - \frac{1}{2}d_{\mathrm{H}}^2(P,Q)$ をバタチャリア係数とする。
   1. $\frac{1}{2}d_{\mathrm{H}}^2(P,Q) \le \mathrm{TV}(P,Q)$ を、$|p-q| = |\sqrt{p}-\sqrt{q}|\,(\sqrt{p}+\sqrt{q})$ を用いて証明せよ。
   2. $\mathrm{TV}(P,Q) \le d_{\mathrm{H}}(P,Q)\sqrt{1 - \frac{1}{4}d_{\mathrm{H}}^2(P,Q)}$ を 33.1 節のコーシー・シュワルツの不等式を用いて証明し、この上界を $\mathrm{TV}^2 \le 1 - \mathrm{BC}^2$ と書き直せ。
   3. イェンセンの不等式を用いて $\mathrm{BC}(P,Q) \ge e^{-D_{\mathrm{KL}}(P\|Q)/2}$ を示し、ブレタニョール・フーバーの不等式 $\mathrm{TV}(P,Q) \le \sqrt{1 - e^{-D_{\mathrm{KL}}(P\|Q)}}$ を導け (Bretagnolle and Huber 1978; Canonne 2022)。D_{KL} がどの値をとるとき、これは Equation 37.31 より精密になるか。
   4. ある検定が n 個の独立標本を受け取り、それらはすべて P からのものであるか、すべて Q からのものであるかのいずれかが確率 $\frac{1}{2}$ ずつで生じるものとする。(c) と 37.1.2.4 節の独立標本にわたる KL の加法性を用いて、いかなる検定も少なくとも確率 $\frac{1}{4}e^{-nD_{\mathrm{KL}}(P\|Q)}$ で誤ることを示せ。誤り確率を高々 η に抑えるには何個のサンプルが必要か。ピンスカーの不等式からはこの種の評価が得られないのはなぜか。
6. **共役とクリティック。**クリティック下界 Equation 37.34 にはダイバージェンス関数の共役 f^* が必要である。
   1. 共役 $f^*(t) = t + t^2/4$（χ^2 関数 $f(u) = (u-1)^2$ に対して $u \in \mathbb{R}$ にわたる上限をとったもの）と最適クリティック $T^\star = 2(p/q - 1)$ を導出せよ。関数の本来の定義域 $u \in (0,\infty)$ 上で計算をやり直し、上限が $t + t^2/4$（$t \ge -2$ のとき）および -1（$t \le -2$ のとき）となることを示せ。
   2. $\mathbb{R}$ 上の共役を用いる。すべてのクリティック T に対し、$\chi^2(P\|Q)$ と下界の差が $\frac{1}{4}\mathbb{E}_Q\big[(T-T^\star)^2\big]$ に等しいことを示せ。ここから、定数倍したクリティック $T = cT^\star$ が保証する下界は $\chi^2(P\|Q)\,(2c - c^2)$ であることを導き、実験が $2T^\star$ に対して出力する値の理由を説明せよ。さらに、実験の勾配上昇 1 ステップによって $T(x) - T^\star(x)$ が $1 - q(x)/2$ 倍されることを示せ。10 ステップ後に差の大半を占めているのは通しで用いている分布の組のどの結果であり、それはなぜか。この勾配上昇はどのステップサイズに対して収束するか。
   3. 全変動の関数 $f(u) = \frac{1}{2}|u-1|$ の共役を $u \in (0,\infty)$ 上で計算せよ。値が $[-\frac{1}{2}, \frac{1}{2}]$ に収まるクリティックだけで下界を最大化するのに十分であることを示し、$\mathrm{TV}(P,Q) = \sup_g\big(\mathbb{E}_P[g] - \mathbb{E}_Q[g]\big)$（上限はすべての関数 g のうち $|g| \le \frac{1}{2}$ を満たすものにわたる）を結論づけよ。これは 37.4.1 節で学ぶ種類の上限表現である。この上限を達成するのはどの g か。

37.4 テスト関数と輸送

サンプルがデータを外しているモデルは、f ダイバージェンスからは何も学習できない。互いに素なサポート上では、この族に属するダイバージェンスはすべて定数に帰着し（37.3.1.2 節）、定数の勾配は 0 だからである。ただしこの診断は、2 つの点質量という極端な例から導かれたものであった。現実の確率分布は通常広がりを持っており、2 つのガウス分布は裾が決して 0 にならないため至るところで重なり合う。わずかな重なりがあれば、勾配は戻ってくるのだろうか。ま

た、f ダイバージェンスには実用上の負担も残されていた。生のサンプルからは得られない密度比 p/q に依存するため、ダイバージェンスを推定するだけでもクリティックネットワークを訓練しなければならなかったのである（37.3.4 節）。クリティックを一切用いずに、サンプルだけから 2 つの分布を比較することはできるだろうか。

最初の問いに数値で答えるため、データ分布を $P = \mathcal{N}(0, 1)$ とし、モデルを同じ釣鐘曲線を標準偏差の d 倍だけ平行移動させた $Q = \mathcal{N}(d, 1)$ とする（d はモデルの唯一のパラメータである）。以下では、必要なライブラリをインポートしたあと、このペアに対して Table 37.2 の 5 つの比較指標と、それぞれの d に関する傾き（訓練中にモデルが受け取る勾配）を評価する。コード中で用いる公式は前節および本節で導出するが、ひとまず出力される数値だけに注目してほしい。以下ではこれを冒頭の表と呼ぶ。

```
from scipy import optimize
from scipy.stats import norm
```

```
def js(d, x=np.linspace(-12, 20, 6401)):
    """JS divergence of N(0, 1) and N(d, 1) by quadrature on the grid x."""
    log_p, log_q = norm.logpdf(x), norm.logpdf(x, loc=d)
    log_m = np.logaddexp(log_p, log_q) - math.log(2)    # log of the mixture
    terms = np.exp(log_p) * (log_p - log_m) + np.exp(log_q) * (log_q - log_m)
    return terms.sum() * (x[1] - x[0]) / 2

comparisons = {                         # N(0, 1) against N(d, 1), RBF ell = 1
    'KL': lambda d: d ** 2 / 2,
    'JS': js,
    'TV': lambda d: 2 * norm.cdf(d / 2) - 1,
    'MMD^2': lambda d: 2 / math.sqrt(3) * (1 - math.exp(-d ** 2 / 6)),
    'W1': lambda d: d}
h = 1e-3                                # step of the central difference
print('   d' + ''.join(f'{name:>8s} {"slope":>7s}' for name in comparisons))
for d in (0.5, 1.0, 2.0, 4.0, 8.0):
    print(f'{d:4.1f}' + ''.join(
        f'{f(d):8.3f} {(f(d + h) - f(d - h)) / (2 * h):7.1e}'
        for f in comparisons.values()))
```

```
  d       KL   slope       JS   slope       TV   slope   MMD^2   slope       W1   slope
0.5    0.125 5.0e-01    0.030 1.2e-01    0.197 3.9e-01   0.047 1.8e-01    0.500 1.0e+00
1.0    0.500 1.0e+00    0.111 2.0e-01    0.383 3.5e-01   0.177 3.3e-01    1.000 1.0e+00
2.0    2.000 2.0e+00    0.337 2.2e-01    0.683 2.4e-01   0.562 4.0e-01    2.000 1.0e+00
4.0    8.000 4.0e+00    0.633 6.9e-02    0.954 5.4e-02   1.074 1.1e-01    4.000 1.0e+00
8.0   32.000 8.0e+00    0.693 2.0e-04    1.000 1.3e-04   1.155 7.2e-05    8.000 1.0e+00
```

2 つのガウス分布の間隔が標準偏差の半分であれば、5 つの傾きはすべて 0.1 から 1 の間に収まっており、どの目的関数を用いてもモデルを効果的に訓練できる。ところが $d = 8$ になると、イェンセン・シャノン・ダイバージェンスは 0.693 で、全変動（total variation）は 1.000 で飽和し（互いに素な点質量で見た定数 $\log 2$ および 1 と同じ値である）、どちらの傾きも約 10^{-4} まで落ち込む。裾が重なっていても救われないのである。KL ダイバージェンスは依然として d そのものに等しい健全な傾きを保つが、それはガウスモデルがあらゆるデータ点に正の密度を割り当てるからにすぎない。サンプルが低次元曲面上に乗る陰的生成器（implicit generator）には、そのような密度は存在しない。残るのは、点質量に対しても有益な情報を保っていた 2 つの距離である。MMD^2 は最初こそ増加するものの、$d = 8$ までには頭打ちとなり、値 1.155 に対して傾きは 10^{-4} 未満になる。あらゆる間隔において線形のまま d に一致し、傾き 1 を保ち続けるのは

W_1 だけである。

MMD も W_1 も密度比 p/q には依存しない。その代わりに MMD は、ある関数クラスを P と Q がどのように**平均**するかを比較する。平均はサンプルから直接推定でき、どの関数クラスを選ぶかによって、2 つの分布がどれだけ離れていても有用な信号を生み出せるかが決まる。一方、W_1 は幾何学的な問いから出発する。一方の分布をもう一方へと変形するには、確率質量をどれだけ遠くまで運ばなければならないかという問いである。驚くべきことに、これもまた平均の比較になることがわかる。本節の残りの議論は 2 つの結果を中心に構成される。固定された特徴の組の上では、平均の比較は補助ネットワークを訓練することなく閉形式で計算できる。他方、質量を動かす最小コストは、傾きが高々 1 の関数全体にわたる平均の差の最大値に等しく、これが Wasserstein GAN（16.2 節）の背後にある双対性（duality）である。本節を通して P はデータ、Q はモデルを表し、実直線上の具体例を用いて議論を進める。

37.4.1 積分確率指標と MMD

37.4.1.1 テスト関数のクラス

全変動はれっきとした距離（metric）であるが、それでも冒頭の表では平坦になってしまった。その変分形式を見れば、なぜそうなったのか、そしてどう修正すればよいかの両方がわかる。全変動とは、任意の事象に関する確率の P と Q の最大の不一致、すなわち任意の事象 A にわたる（37.3.3 節）$\mathrm{TV}(P,Q) = \sup_A |P(A) - Q(A)|$ である。事象の確率はその指示関数（indicator function）の期待値にほかならないため、全変動はある関数クラス上でとった 2 つの平均の差の上限（supremum）となっている。ここで関数を指示関数だけに限定すべき理由はない。任意のクラス $\mathcal{G}$ に属する**テスト関数**（test function）g が与えられたとき、**積分確率指標**（integral probability metric、IPM）(Müller 1997) は次のように定義される。

$$\mathrm{IPM}_{\mathcal{G}}(P,Q) = \sup_{g \in \mathcal{G}} \left(\mathbb{E}_{x\sim P}[g(x)] - \mathbb{E}_{x\sim Q}[g(x)] \right). \tag{37.35}$$

$\mathcal{G}$ を、空間内の各点にスコアをつける監査人のパネルだと考えてみよう。IPM は、パネル内の監査人が見つけ出せる平均スコアの最大の差を報告する。ここで考えるクラスはすべて対称である。すなわち、$-g$ が $\mathcal{G}$ に属するのが g が属するときであるため、この差は常に非負であり、P と Q に関して対称になる。「距離（metric）」という言葉が約束するように $P = Q$ のときに限って IPM が 0 になるかどうかは、異なる任意の 2 分布を識別できるほど $\mathcal{G}$ が豊かであるかどうかにかかっている。37.3.4 節のクリティックも同様の監査の役割を果たしていたが、決定的な違いがある。変分下界 $D_f(P\|Q) \geq \mathbb{E}_P[T] - \mathbb{E}_Q[f^*(T)]$ では、凸共役（convex conjugate）f^* が制約のないクリティック T を自動的に抑制していた。これに対し IPM では、g が P と Q の両方で直接平均されるため、g を制約するものは $\mathcal{G}$ に属すること以外には何もない。したがって、すべては適切なクラスを選ぶことにかかっている。

ここでは次の 3 つの関数クラスが主役となる。

- $|g| \leq \frac{1}{2}$ を満たす有界関数からは全変動が復元される。指示関数から $\frac{1}{2}$ を引いた関数はこの上界を満たし、定数は平均の差をとると相殺され、$g = \frac{1}{2}\,\mathrm{sign}(p-q)$ が上限を達成する。
- 固定された特徴の組に関して線形な関数からは、次に構築する**最大平均値差異**（maximum mean discrepancy、MMD）(Gretton et al. 2012) が得られる。
- 傾きが高々 1 の関数からはワッサースタイン距離（Wasserstein distance）が得られる。これについては 37.4.2 節で詳しく扱う。

なぜ毎回、可能な限り最大のクラス $\mathcal{G}$ を選ばないのだろうか。有用なクラスは互いに競合する 2 つの要件を満たす必要があり、$\mathcal{G}$ を大きくしすぎると第 1 の要件が破綻するからである。第 1 に、標本平均が真の期待値に追随しなければならない。サンプルを用いれば密度比なしに期待値を直

接推定できるとはいえ、監査人のパネルが豊かすぎると、有限個の標本間に常に見せかけの差を見つけ出してしまう。たとえば、**同じ**連続分布から抽出した 1,000 点ずつの 2 つの独立標本は、共通の点を 1 つも持たない。第 1 の標本の指示関数は、第 1 の標本上では平均 1、第 2 の標本上では平均 0 となるため、経験全変動は 1 となり、これは P と Q が同一であるかどうかに依存しない。これを避けるには、そのような鋭く尖った関数を排除できるほど $\mathcal{G}$ が十分に小さいか、滑らかでなければならない。

第 2 に、P と Q が離れるにつれて差も増大し続けなければならず、そのためには距離とともに増大する関数が $\mathcal{G}$ に含まれている必要がある。全変動にはそのような関数が存在しない。テスト関数が $\tfrac{1}{2}$ で上から抑えられているため、差が 1 を超えることは決してないからである。最大の不一致が生じるのは事象 $\{p > q\}$ 上であり、冒頭の表の 2 つのガウス分布ではこれは半直線 $x < d/2$ となる。両方のガウス分布のもとでこの半直線の確率を計算すると、$\mathrm{TV} = \Phi(d/2) - \Phi(-d/2) = 2\Phi(d/2) - 1$ が得られる（ここで Φ は標準正規分布の累積分布関数（cumulative distribution function、CDF）である）。この値は 0.954（$d = 4$）に達し、10^{-4} 以内まで 1 に迫る（$d = 8$）。階段関数が 2 つの釣鐘曲線をほぼ完全に分離してしまえば、それ以上引き離してもスコアはほとんど変化しない。全変動はこれら両方の要件を満たしておらず、続く 2 つのクラスもこの両要件に照らして評価していく。

37.4.1.2 平均特徴とカーネル

最小の監査人パネルから出発し、同じ平均を持ちながら分散が異なる 1 組のペア $P = \mathcal{N}(0, 1)$ と $Q = \mathcal{N}(0, 4)$ で各段階を検証しながら、より優れたクラスを一から構築していこう。

最も単純な選択として、線形テスト関数 $g(x) = w\,x$（$|w| \le 1$）から始める。このとき $\mathbb{E}_P g - \mathbb{E}_Q g = w\,(\mathbb{E}_P X - \mathbb{E}_Q X)$ であるから、$|w| \le 1$ にわたって上限をとると、2 つの平均の距離 $|\mathbb{E}_P X - \mathbb{E}_Q X|$ が得られる。このパネルは分散に対して盲目であり、検証用のペアに対しては 0 を報告する。

位置だけでなく広がりも捉えるため、2 つ目の特徴を追加しよう。特徴写像 $\varphi(x) = (x, x^2)$ を定義し、線形結合 $g(x) = \mathbf{w}^{\top}\varphi(x)$（$\|\mathbf{w}\| \le 1$）をとる。$\mathbf{m}_P = \mathbb{E}_P[\varphi(x)]$ で P の**平均特徴**（mean feature）を表せば（同様に $\mathbf{m}_Q$ を Q に対して定める）、平均の差は $\mathbb{E}_P g - \mathbb{E}_Q g = \mathbf{w}^{\top}(\mathbf{m}_P - \mathbf{m}_Q)$ となる。コーシー・シュワルツの不等式により、単位球上における線形関数の最大値は、その係数ベクトルのユークリッドノルムにほかならない。

$$\mathrm{MMD}(P, Q) = \sup_{\|\mathbf{w}\| \le 1} \mathbf{w}^{\top}(\mathbf{m}_P - \mathbf{m}_Q) = \|\mathbf{m}_P - \mathbf{m}_Q\|. \tag{37.36}$$

このように MMD は P と Q の平均特徴ベクトル間の距離となり、上限をとる操作は完全に消え去った。検証用のペアでは平均特徴は $\mathbf{m}_P = (0, 1)$ と $\mathbf{m}_Q = (0, 4)$ であり、$\mathrm{MMD}^2 = 0^2 + 3^2 = 9$ となる。こうして、平均と二次モーメントの両方の違いを検出できるようになった。

次に、二乗ノルムを展開しよう。2 つの期待値の積は独立なサンプルに対する積の期待値に等しいため、$\|\mathbf{m}_P\|^2 = \mathbb{E}\langle \varphi(x), \varphi(x') \rangle$ と書ける（ただし $x, x' \sim P$ は独立）。独立な $y, y' \sim Q$ および交差項についても同様に展開すると、$\|\mathbf{m}_P - \mathbf{m}_Q\|^2 = \mathbb{E}\langle \varphi(x), \varphi(x') \rangle + \mathbb{E}\langle \varphi(y), \varphi(y') \rangle - 2\,\mathbb{E}\langle \varphi(x), \varphi(y) \rangle$ が得られる。ここで特徴 $\varphi(x)$ は決して単独では現れず、その内積だけが現れている点に注目しよう。そこで**カーネル関数**（kernel）を $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ と定義すると、次式が得られる。

$$\mathrm{MMD}^2(P, Q) = \mathbb{E}_{x, x' \sim P}[k(x, x')] + \mathbb{E}_{y, y' \sim Q}[k(y, y')] - 2\,\mathbb{E}_{x \sim P, y \sim Q}[k(x, y)]. \tag{37.37}$$

この式には直観的な幾何学的意味がある。最初の 2 項はそれぞれ P の**内部**および Q の**内部**における平均類似度を測り、第 3 項は両者の**間**の類似度を測っている。P と Q からのサンプルが混ざり合っているときは、標本内類似度と標本間類似度が一致して 3 つの項が打ち消し合い、別々のクラスタに分離しているときは内部の項が支配的になる。最初の段階は**線形カーネル** $k(x,x')=xx'$ に、2 番目の段階は**二次カーネル** $k(x,x')=xx'+x^2x'^2$ に対応している。

二次カーネルでさえまだ死角が残っている。平均と分散が一致する任意の 2 分布に対して距離が 0 になってしまうのである。x^3、次いで x^4 と順に追加していってもきりがない。そこで決定的な一歩となるのが、有限個の特徴のリストを書き下すことなく、カーネルを類似度の尺度として直接選ぶことである。**RBF カーネル** $k(x,x')=e^{-(x-x')^2/(2\ell^2)}$ は、2 点がおよそ**長さスケール**（length scale）ℓ 以内にあるときに両者を類似しているとみなす。直接選んだとはいえ、このカーネルも依然として特徴の内積になっている。実際、$e^{-x^2/(2\ell^2)}e^{-x'^2/(2\ell^2)}e^{xx'/\ell^2}$ と因数分解して最後の因子をべき級数展開すると、無限に多くの特徴 $\varphi_j(x)=x^j e^{-x^2/(2\ell^2)}/(\ell^j\sqrt{j!})$ $(j=0,1,2,\dots)$ にわたる内積が現れる。これらは重みが急速に減衰するあらゆる次数の減衰単項式である。いまやパネルには無限に多くの監査人が含まれているが、その評決を下すのに必要なのは相変わらず Equation 37.37 の 3 つの平均だけである。$\mathbb{R}^n$ においては、単に $(x-x')^2$ を $\|x-x'\|^2$ に置き換えるだけで、他は何も変わらない。

このカーネルを用いれば死角は一切残らない。$\mathrm{MMD}(P,Q)=0$ となるのは $P=Q$ のとき、かつそのときに限られ (Gretton et al. 2012)、RBF-MMD は真の距離となる。この性質を持つカーネルは**特性的**（characteristic）であるといい、線形カーネルと二次カーネルはそうではない。

ガウス分布に対しては、RBF の期待値を閉形式で計算できる。$Z\sim\mathcal{N}(\mu,s^2)$ とすると、標準的なガウス積分により次式が得られる。

$$\mathbb{E}\big[e^{-Z^2/(2\ell^2)}\big]=\Big(1+\frac{s^2}{\ell^2}\Big)^{-1/2}\exp\!\Big(-\frac{\mu^2}{2(\ell^2+s^2)}\Big). \tag{37.38}$$

独立な 2 つのガウス分布の差は再び分散の和を持つガウス分布になるため、Equation 37.37 の 3 つの項をすべて直ちに読み取れる。検証用のペアである $\mathcal{N}(0,1)$ 対 $\mathcal{N}(0,4)$ において $\ell=1$ とすると、差 $x-x'$、$y-y'$、$x-y$ はいずれも平均 0 で、分散はそれぞれ 2、8、5 となり、$\mathrm{MMD}^2=3^{-1/2}+9^{-1/2}-2\cdot 6^{-1/2}\approx 0.094$ が得られる。冒頭の表の平行移動した 2 つのガウス分布に対しては、同じ積分から $\frac{2}{\sqrt{3}}\big(1-e^{-d^2/6}\big)$ が導かれる（演習問題 1a）。Table 37.2 の点質量に対しては平均をとる必要すらなく、Equation 37.37 から直ちに $1+1-2k(0,d)=2-2e^{-d^2/2}$ が得られ、同表の 4 行目と一致する。いずれの場合も d に依存するのは交差項だけであり、d が長さスケールの数倍を超えると急速に減衰する。したがって差は平坦になってしまい、第 2 の要件を満たさない。RBF の監査人は滑らかではあるが、近視眼的なのである。

37.4.1.3 サンプルからの推定

カーネルが真価を発揮するのは第 1 の要件、すなわちサンプルからの信頼性の高い推定である。P と Q からの有限標本が与えられると、Equation 37.37 の各期待値は点のペアにわたる平均になる。不偏推定量（unbiased estimator）を得るため (Gretton et al. 2012)、各点をそれ自身と組にする対角項は除外する（残すと標本内平均に上向きの統計的バイアスが生じるためである）。以下では、個数 $N=1{,}000$ で $\mathcal{N}(0,1)$ から抽出した標本と、1,000 個を $\mathcal{N}(0,4)$ から抽出した標本に対し、線形・二次・RBF の各カーネルのもとでこの推定量を評価する。それぞれの母集団での真値は、先ほど計算したとおり 0、9、0.094 である。RBF カーネルについてはさらに 2 組のペアも試す。$\mathcal{N}(0,1)$ からの 2 つ目の独立標本（母集団値は 0）と、平行移動させた $\mathcal{N}(0.5,1)$ からの標本（母集団値は冒頭の表の 0.047）である。目的は、標本推定値がこれらの母集団値にどれだけ密接に追随するかを確かめることにある。

```
def mmd2_unbiased(x, y, k):
    """Unbiased MMD^2 between samples x and y for the kernel k(a, b)."""
    kxx = k(x[:, None], x[None, :])
    kyy = k(y[:, None], y[None, :])
    n, m = len(x), len(y)
    return ((kxx.sum() - np.trace(kxx)) / (n * (n - 1))
            + (kyy.sum() - np.trace(kyy)) / (m * (m - 1))
            - 2 * k(x[:, None], y[None, :]).mean())

linear = lambda a, b: a * b                         # features phi(x) = x
quadratic = lambda a, b: a * b + a ** 2 * b ** 2    # features (x, x^2)
rbf = lambda a, b: np.exp(-(a - b) ** 2 / 2)        # RBF kernel, ell = 1

rng = np.random.default_rng()
n = 1000
x_p = rng.standard_normal(n)                # P = N(0, 1)
y_var = 2 * rng.standard_normal(n)          # N(0, 4)
y_same = rng.standard_normal(n)             # N(0, 1) again
y_shift = rng.standard_normal(n) + 0.5      # N(0.5, 1)
for name, y, k, pop in [
        ('linear,    N(0,1) vs N(0,4)', y_var, linear, 0.0),
        ('quadratic, N(0,1) vs N(0,4)', y_var, quadratic, 9.0),
        ('RBF,       N(0,1) vs N(0,4)', y_var, rbf,
         3 ** -0.5 + 9 ** -0.5 - 2 * 6 ** -0.5),
        ('RBF,       N(0,1) vs N(0,1)', y_same, rbf, 0.0),
        ('RBF,       N(0,1) vs N(0.5,1)', y_shift, rbf,
         2 / np.sqrt(3) * (1 - np.exp(-1 / 24)))]:
    print(f'{name:30s} MMD^2 = {mmd2_unbiased(x_p, y, k):+.4f}  '
          f'(population {pop:.4f})')
```

```
linear,    N(0,1) vs N(0,4)    MMD^2 = +0.0142  (population 0.0000)
quadratic, N(0,1) vs N(0,4)    MMD^2 = +8.3040  (population 9.0000)
RBF,       N(0,1) vs N(0,4)    MMD^2 = +0.0873  (population 0.0942)
RBF,       N(0,1) vs N(0,1)    MMD^2 = -0.0004  (population 0.0000)
RBF,       N(0,1) vs N(0.5,1)  MMD^2 = +0.0546  (population 0.0471)
```

線形カーネルは 2 つの平均が一致しているため 0 に近い数値を返す。具体的には、その不偏推定値は 2 つの標本平均の差の二乗からその差の分散の推定値（およそ $(1+4)/1{,}000 = 0.005$）を引いたものに等しく、その結果、-0.005 を大きく下回ることはまれである一方、約 20 回に 1 回の実行で $+0.015$ を超える。二次カーネルは 4 乗が標本誤差を増幅するため、9 の前後 3 程度の範囲に落ち着く。RBF カーネルのもとでは、分散が異なるペアの推定値は約 0.094 ± 0.03、平均が異なるペアでは約 0.047 ± 0.02 となり、標本誤差が信号の無視できない割合を占めうる。しかし、**同じ**分布 $\mathcal{N}(0,1)$ からの 2 つの独立標本の間では、3 つの平均に含まれる 1 次の標本誤差が打ち消し合うため、RBF 推定値は驚くほどばらつきが小さく、0 から数千分の 1 以内に収まる（わずかに負になることもある）。同じ分布からの 2 標本間に全変動が 1 という見せかけの差を報告した場面でも、RBF カーネルは何の差も見出さないのである。

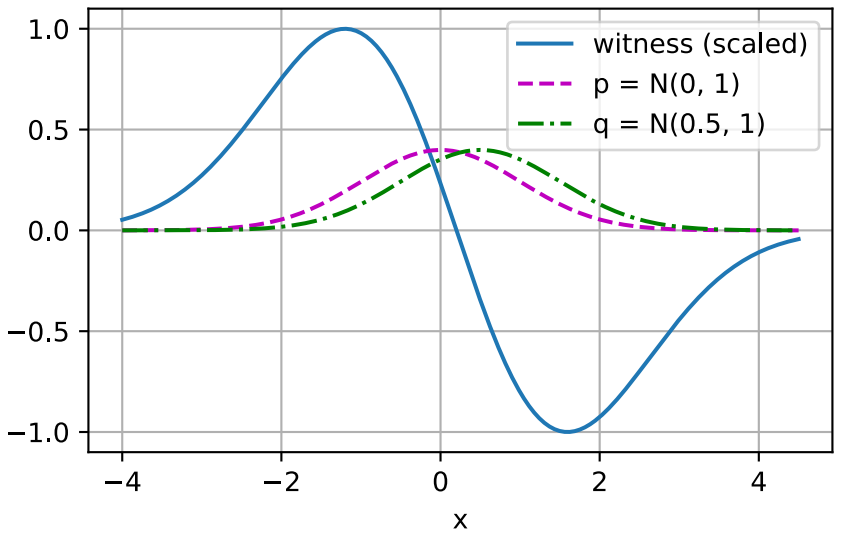
要するに、RBF 推定量は非ゼロの差を測るときには比較的大きな誤差を伴うものの、$P = Q$ のときには並外れた精度を発揮する。この性質により、RBF 推定量は強力な **2 標本検定**（two-sample test）となる。各標本あたり $1{,}000$ 点あれば、RBF の MMD^2 が 100 分の数程度であっても 2 つの分布が異なることの強い証拠となり、しかもクリティックの訓練も密度の評価も一切不要である。

37.4.1.4 証人関数

IPM は上限として定義されるため、実際にどのテスト関数が最大の差を達成したのかを問うのは自然である。MMD の場合、Equation 37.36 の上限は $\mathbf{w}^\star \propto \mathbf{m}_P - \mathbf{m}_Q$ によって達成される。得られるテスト関数は**証人関数**（witness）と呼ばれ、$g^\star(x) \propto \langle \mathbf{m}_P - \mathbf{m}_Q, \varphi(x) \rangle = \mathbb{E}_P[k(x, X)] - \mathbb{E}_Q[k(x, Y)]$ となる。これもまたカーネルの評価しか必要としない。平行移動したガウス分布 $\mathcal{N}(0, 1)$ と $\mathcal{N}(0.5, 1)$ では、2 つの密度は $x = 0.25$ を挟んで鏡像の関係にあるため、母集団における証人関数はちょうど 0.25 で符号を変える。以下では、グリッド上でサンプルから証人関数を推定し、2 つの密度と並べてプロットする。あわせて、スケール前の証人関数に対する $\mathbb{E}_P g - \mathbb{E}_Q g$、すなわち $g = \langle \mathbf{m}_P - \mathbf{m}_Q, \varphi \rangle$ に対する値も出力する。この値は $\|\mathbf{m}_P - \mathbf{m}_Q\|^2 = \mathrm{MMD}^2$ に等しく、ここでは自己ペアを含めて推定している。

```
g = lambda s: (rbf(s[:, None], x_p[None, :]).mean(axis=1)    # the unscaled
               - rbf(s[:, None], y_shift[None, :]).mean(axis=1))  # witness
grid = np.linspace(-4.0, 4.5, 400)
witness = g(grid)
print(f'witness changes sign at x = {grid[np.argmax(witness < 0)]:.2f}')
print(f'E_P g - E_Q g = {g(x_p).mean() - g(y_shift).mean():.4f}')
d2l.plot(grid, [witness / np.abs(witness).max(), norm.pdf(grid),
                norm.pdf(grid, loc=0.5)], 'x', '',
         legend=['witness (scaled)', 'p = N(0, 1)', 'q = N(0.5, 1)'],
         figsize=(5, 3))
```

```
witness changes sign at x = 0.22
E_P g - E_Q g = 0.0554
```



予想どおり、証人関数は P の密度が高い領域で正、Q の密度が高い領域で負となり、$x = 0.25$ 付近で 0 を横切る（各標本 1,000 点では、実行ごとにゼロ交差点の位置が 0.1〜0.2 ほど変動する）。出力された差は前のセルの不偏推定値と約 0.001 以内で一致しており、わずかなずれは自己ペアに由来する。

証人関数のプロットの裾は、ゼロ交差点以上に重要な事実を物語っている。差 $\mathbb{E}_P[g] - \mathbb{E}_Q[g]$ を

縮めるには、生成モデルは各サンプルを g の上り坂の方向へ押し上げなければならないため、モデルのサンプル位置における証人関数の傾きこそが、そのサンプルが受け取る勾配信号となる。2 つの釣鐘曲線の間では証人関数は急峻であり、データへ向かう強い引力を生み出す。ところが裾の方へ長さスケールの数倍も離れると、証人関数は平坦になって 0 に近づき、遠く離れたモデルのサンプルには勾配がほとんど届かなくなる。冒頭の表の MMD^2 の列を、1 つのサンプルの視点から目の当たりにしているわけである。

したがって実用上、固定カーネル MMD は 2 つの異なる役割を担う。ノンパラメトリックな 2 標本検定としての役割と、敵対的最適化を回避する生成モデルの訓練目的関数としての役割である (Li et al. 2015)。たとえば、画像生成器の評価に用いられる Kernel Inception Distance（16.5 節）は、画像の深層特徴量に対して三次カーネルで不偏 MMD 推定量を適用したものにほかならない。長さスケール ℓ を慎重に選ぶことは不可欠であり、$\ell = 1$ では $d = 8$ における傾きが 7×10^{-5} まで崩壊してしまった（演習問題 1b ではより大きなスケールを調べる）。実際には、ℓ をサンプル間のペアごとの距離の中央値に設定するか、複数の長さスケールにわたって RBF カーネルを足し合わせることが多く、16.2 節で GAN の**生成器**を MMD で訓練する際にもこの戦略が用いられている。

37.4.2 最適輸送とワッサースタイン距離

冒頭の表には、決して平坦にならない列が 1 つだけあった。W_1 が一定の傾き 1 を保ち続けたのは、f ダイバージェンスが無視し、固定カーネルが局所的にしか捉えられない基礎空間の幾何学を取り込んでいるからである。W_1 のもとで点質量 $\delta_{0.01}$ が δ_0 に近いのは、単にその質量が距離 0.01 だけ移動すれば済むためである。この着想は 1781 年にまで遡る。当時 Gaspard Monge は、盛り土を同じ体積の穴へと最小の総仕事量で運ぶにはどうすればよいかを問うた。その最小仕事量こそ、2 つの分布がどれだけ離れているかを測る自然な尺度となる。

37.4.2.1 輸送計画

実直線上の 6 点 $0, 1, \dots, 5$ をサポートとし、P が左側に、Q が右側に集中している 2 つの離散分布を考えよう（Figure 37.10 左上）。P を Q の形に組み替えるには、各出発点 x から各到着点 y へどれだけ質量を動かすかを指定しなければならない。このような**輸送計画**（transport plan）とは、同時分布 γ（ペア (x, y) 上で定義され、周辺分布が P と Q に一致するもの）のことである。すなわち、各 x から出ていく質量の合計は $P(x)$ に等しく、各 y に到着する質量の合計は $Q(y)$ に等しい。確率論では、輸送計画は P と Q の**カップリング**とも呼ばれる。**1 次ワッサースタイン距離**（Wasserstein-1 distance。土砂移動距離、earth mover's distance とも呼ばれる）は、有効なすべての計画にわたる移動距離の期待値の最小値である。$\Pi(P, Q)$ を、有限な一次モーメントを持つ P と Q（$\mathbb{R}^n$ 上）のすべてのカップリングの集合と書くことにすれば、次式で与えられる。

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}\big[\|x - y\|\big]. \tag{37.39}$$

添字の 1 は距離の 1 乗を表しており、距離の二乗を用いる版である W_2 もまもなく登場する。2 つの点質量 δ_0 と δ_d の間では、可能な計画は全質量を 0 から d へ移すただ 1 つしかないため、$W_1(\delta_0, \delta_d) = |d|$ となり、勾配は ± 1 である（$d \neq 0$ のとき）。冒頭の表の平行移動した 2 つのガウス分布では、すべての点を d だけ平行移動させるのがコスト d の有効な計画であり、$\mathbb{E}_\gamma|x - y| \geq |\mathbb{E}_\gamma[x - y]| = d$ が任意のカップリング γ に対して成り立つため、これより優れた計画は存在しない。いずれの場合も W_1 は間隔 d そのものを復元し、f ダイバージェンスに欠けていた明瞭な勾配信号を与える。

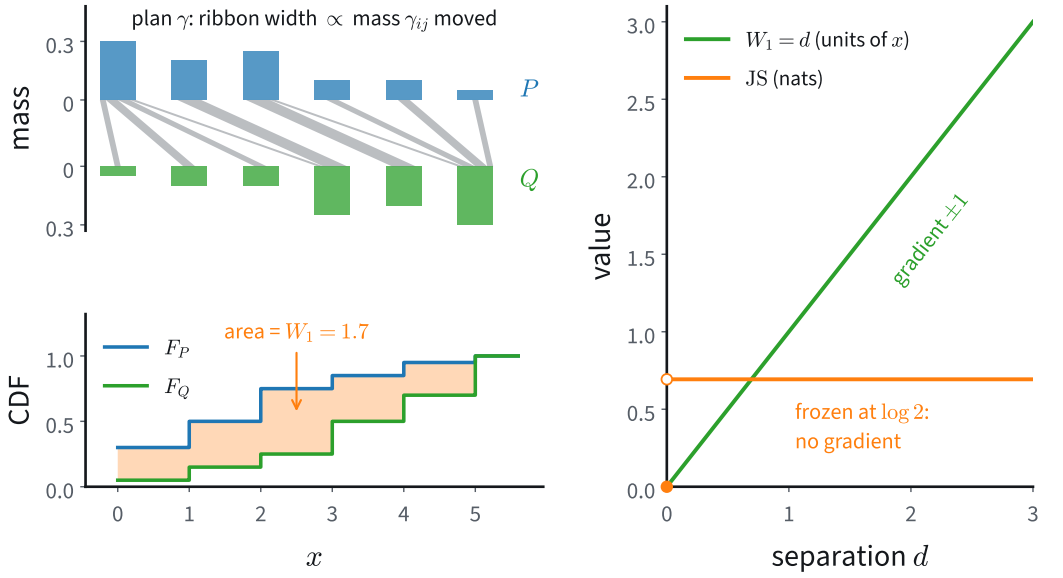


Figure 37.10: 直線上の輸送。左上：6 点上の 2 つの分布間の輸送計画。各リボンは P（上）の点から Q（下）の点へ質量を運び、その幅は移動する質量に比例する。左下：P と Q の CDF。両者の間の塗りつぶされた面積は 1.7 である。右：距離 d だけ離れた 2 つの点質量に対し、イェンセン・シャノン・ダイバージェンス（単位は nat）は $\log 2$ で平坦なままである一方、W_1（単位は x）は d に対して線形に増大する。

37.4.2.2 カントロヴィッチ・ルビンシュタイン双対性

計算手順として見ると、Equation 37.39 は手強そうに見える。同時分布 γ にわたって最小化しており、サンプルからパラメータ化することさえ厄介だからである。しかし有限点集合上では、これはおなじみの最適化問題、すなわち線形計画問題（linear program）にすぎない。未知変数は輸送計画の各成分であり、輸送コストも周辺分布の制約もそれらの成分に関して線形である。あらゆる線形計画問題には、制約に対する乗数に関して最大化する**双対**問題（dual problem）が存在する（35.7.1 節）。最適輸送（optimal transport）においては、この双対問題が同時分布にわたる探索を単一の関数に関する最大化へと置き換える。この再定式化こそが、深層学習において W_1 を実用的なものにしている。

命題（カントロヴィッチ・ルビンシュタイン双対性） 分布 P および Q（$\mathbb{R}^n$ 上で有限な一次モーメントを持つもの）に対し、

$$W_1(P,Q) = \sup_{\|g\|_{\mathrm{Lip}} \le 1} \left(\mathbb{E}_{x\sim P}[g(x)] - \mathbb{E}_{y\sim Q}[g(y)] \right), \tag{37.40}$$

が成り立つ。ここで上限はすべての 1 リプシッツ関数 g、すなわち $|g(x) - g(y)| \le \|x - y\|$ をすべての x, y について満たす関数にわたってとる。

カップリングにわたる下限（infimum）が、単一のテスト関数にわたる上限へと姿を変えた。すなわち W_1 は、先ほど予告した 3 つ目のクラスである傾き高々 1 の関数上の IPM（Equation 37.35）にほかならない。1 リプシッツな監査人は急激なジャンプができないため、指示関数のように個々のサンプル点を丸暗記することはできない。それでいて有界関数や RBF カーネルとは異なり際限なく増大できるため、第 2 の要件も満たしている。たとえば冒頭の表の平行移動した 2 つのガウ

ス分布では、$g(x) = -x$ は 1 リプシッツであり、期待値は 0（P のもと）と $-d$（Q のもと）になるため、差 $d = W_1(P, Q)$ を達成する。さらに都合のよいことに、モデルのサンプルがデータからどれだけ遠く離れていようとも、その勾配は至るところ大きさ 1 でデータの方向を指し示す。

ニューラルネットワークはそのようなテスト関数を近似できるため、**WGAN** (Arjovsky et al. 2017) は双対目的関数 Equation 37.40 を直接訓練するにあたり、近似的に 1 リプシッツとなるよう制約したニューラルクリティックを g の役割に据える。16.2 節ではこの目的関数を他の敵対的損失と比較し、勾配ペナルティ（gradient penalty）によって傾きを 1 に引き寄せるクリティックを訓練する。また 16.4 節では、後発の手法がペナルティの中心を 0 に置く理由を示す。傾き 1 を好むペナルティはすでにデータと一致している生成器までも押し動かし続けるのに対し、0 を中心とするペナルティはサンプル位置でのクリティックの勾配を小さく保つだけで、リプシッツ制約を課さないためである。なお、クリティックが近似的にしかリプシッツでなく、かつニューラルアーキテクチャに制限されているとき、その報告するスコアはもはや厳密な W_1 ではないことに留意しておこう。

37.4.2.3 直線上の輸送

1 次元の場合についてカントロヴィッチ・ルビンシュタイン双対性（Kantorovich–Rubinstein duality）を証明しよう。1 次元ではこの議論から、最適化を一切必要としない閉形式の公式も同時に得られる（高次元での証明は Peyré and Cuturi (2019) を参照）。直観的には、各点 $t \in \mathbb{R}$ をどれだけの質量が横切るかを数えるだけでよい。t の左側（t を含む）には、P が質量 $F_P(t)$ を持ち、Q が質量 $F_Q(t)$ を持つため、有効な輸送計画はいずれも少なくとも $|F_P(t) - F_Q(t)|$ の質量を t を越えて運ばなければならない。さらに、x から y へ運ばれる微小質量は、あらゆる点 t（x と y の間にあるもの）を横切るため、その移動距離 $|x - y|$ は横切る点からなる区間の長さに等しい。すべての微小質量について足し合わせれば、総輸送コストは t に関する積分（t を横切る総質量の積分）となる。

命題（直線上の W_1） 分布 P および Q（$\mathbb{R}$ 上で CDF F_P、F_Q と有限な一次モーメントを持つもの）に対し、

$$W_1(P, Q) = \int_{-\infty}^{\infty} |F_P(t) - F_Q(t)|\, dt = \sup_{\|g\|_{\mathrm{Lip}} \le 1} \big(\mathbb{E}_P[g] - \mathbb{E}_Q[g]\big), \tag{37.41}$$

が成り立つ。したがって、実直線上ではカントロヴィッチ・ルビンシュタイン双対性が成り立つ。

証明 ステップ 1：あらゆる計画のコストは少なくとも CDF 間の面積以上である。 $(X, Y) \sim \gamma$ に対し、$|X - Y| = \int \mathbf{1}(t \text{ lies between } X \text{ and } Y)\, dt$ と書く。被積分関数は非負であるため、期待値と積分の順序を交換できる。

$$\begin{aligned} \mathbb{E}_\gamma |X - Y| &= \int \Big(\gamma(X \le t < Y) + \gamma(Y \le t < X) \Big)\, dt \\ &\ge \int \big| \gamma(X \le t < Y) - \gamma(Y \le t < X) \big|\, dt. \end{aligned}$$

鍵となるのは、最後の積分の内部にある差を整理することである。$\gamma(X \le t, Y \le t)$ を足して引くと、$\gamma(X \le t < Y) - \gamma(Y \le t < X) = \gamma(X \le t) - \gamma(Y \le t) = F_P(t) - F_Q(t)$ となる（γ が周辺分布 P と Q を持つため）。したがって、あらゆる輸送計画のコストは少なくとも $\int |F_P - F_Q|$ 以上である。

ステップ 2：この下界を達成する計画が 1 つ存在する。 分位点を直接対応させよう。すなわち、P の下位 10% を Q の下位 10% へ送り、以下同様に直線上を上っていく。形式的に

は、U を $(0,1)$ 上の一様確率変数とし、$X = F_P^{-1}(U)$ および $Y = F_Q^{-1}(U)$ とおく。ここで分位点関数（quantile function）$F^{-1}(\lambda) = \inf\{x : F(x) \ge \lambda\}$（$\lambda \in (0,1)$）は、$F^{-1}(\lambda) \le t$ となるのが $\lambda \le F(t)$ のとき、かつそのときに限られるという性質を満たす。$\Pr(X \le t) = \Pr(U \le F_P(t)) = F_P(t)$ であり Y についても同様であるから、これは有効なカップリングを定める。X と Y がともに増加するため、これを**単調計画**（monotone plan）と呼ぶ。このカップリングのもとでは、$\gamma(X \le t < Y) = \Pr\big(F_Q(t) < U \le F_P(t)\big)$ および $\gamma(Y \le t < X) = \Pr\big(F_P(t) < U \le F_Q(t)\big)$ となる。すべての t において、これら 2 つの確率のうち少なくとも一方は 0 である。つまり単調計画は、同じ点を挟んで両方向に質量を送ることが決してない。したがってステップ 1 の不等式は等号で成立し、最小輸送コストは CDF 間の面積に等しい。

ステップ 3：いかなる 1 リプシッツなテスト関数も、どの計画のコストも上回らない。この**弱双対性**（weak duality）の議論は任意の次元で成り立つ。任意の 1 リプシッツ関数 g と任意のカップリング γ に対し、各点で $g(x) - g(y) \le \|x - y\|$ が成り立つため、γ のもとで期待値をとれば直ちに $\mathbb{E}_P[g] - \mathbb{E}_Q[g] \le \mathbb{E}_\gamma\|x - y\|$ が得られる（35.7.1 節の弱双対性と比較されたい）。

ステップ 4：この面積を達成する 1 リプシッツなテスト関数が 1 つ存在する。 $g(x) = g(0) + \int_0^x g'(t)\,dt$（$|g'| \le 1$）という形の関数だけを考えれば十分である。このような g はすべて 1 リプシッツであり、それ以外の関数もすでにステップ 3 で上から抑えられているからである。定数 $g(0)$ は $\mathbb{E}_P[g] - \mathbb{E}_Q[g]$ の中で相殺される。$\int_0^X g'(t)\,dt = \int_0^\infty g'(t)\,\mathbf{1}(X > t)\,dt - \int_{-\infty}^0 g'(t)\,\mathbf{1}(X \le t)\,dt$ と書き換え、期待値と積分の順序を交換すると（一次モーメントが有限であることから、34.6 節のフビニの定理により正当化される）、次式が得られる。

$$\mathbb{E}_P[g] - \mathbb{E}_Q[g] = \int_{-\infty}^{\infty} g'(t)\,\big(F_Q(t) - F_P(t)\big)\,dt.$$

$g'(t) = \operatorname{sign}\big(F_Q(t) - F_P(t)\big)$ と選べば、右辺は $\int |F_P(t) - F_Q(t)|\,dt$ となる。以上 4 つのステップを合わせると、1 リプシッツ関数全体にわたる上限は少なくとも面積以上であり（ステップ 4）、かつ最小計画コスト以下であるが（ステップ 3）、その最小計画コストは面積に等しい（ステップ 1 および 2）。■

結局のところ実直線上では、W_1 は 2 つの CDF に挟まれた面積にすぎず、同じサイズの 2 つの標本に対してはソートした値の平均絶対差に帰着する（演習問題 2a）。線形計画問題全体がこれほどきれいに簡約されるのは驚くべきことである。この簡約を Figure 37.10 の 6 点の例で確かめてみよう。隣接するアトム間の 5 つの区間において、CDF の差は 0.25、0.35、0.5、0.35、0.25 であり、その和は図の塗りつぶされた面積である 1.7 となる。以下のコードでは 3 通りの方法で W_1 を計算する。5 つの CDF の差を足し合わせる方法、変数 36 個の線形計画問題（6×6 輸送行列のうち行和が p、列和が q となるものの上での問題）を解く方法、そしてステップ 2 の単調計画（各経路が対応する分位点区間の重なりを運ぶ計画）を構成してそのコストを評価する方法である。最後に、線形計画法が単調計画を返したかどうかも確認する。

```
atoms = np.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0])
p_w = np.array([0.30, 0.20, 0.25, 0.10, 0.10, 0.05])
q_w = np.array([0.05, 0.10, 0.10, 0.25, 0.20, 0.30])

F_p, F_q = np.cumsum(p_w), np.cumsum(q_w)
w1_cdf = np.sum(np.abs(F_p - F_q)[:-1] * np.diff(atoms))

C = np.abs(atoms[:, None] - atoms[None, :])      # cost matrix |x_i - y_j|
```

```
    n_atoms = len(atoms)
    A_eq = np.zeros((2 * n_atoms, n_atoms * n_atoms))
    for i in range(n_atoms):
        A_eq[i, i * n_atoms:(i + 1) * n_atoms] = 1   # row sums    = p
        A_eq[n_atoms + i, i::n_atoms] = 1            # column sums = q
    res = optimize.linprog(C.ravel(), A_eq=A_eq,
                           b_eq=np.concatenate([p_w, q_w]), method='highs')
    plan_lp = res.x.reshape(n_atoms, n_atoms)
    # Monotone plan (Step 2): source i holds the quantiles (F_p[i-1], F_p[i]],
    # destination j holds (F_q[j-1], F_q[j]]; route i -> j carries their overlap
    lo_p, lo_q = F_p - p_w, F_q - q_w
    mono = np.clip(np.minimum(F_p[:, None], F_q[None, :])
                   - np.maximum(lo_p[:, None], lo_q[None, :]), 0, None)
    print(f'W1 from the CDF gaps     : {w1_cdf:.4f}')
    print(f'W1 from the LP           : {res.fun:.4f}')
    print(f'cost of the monotone plan: {np.sum(mono * C):.4f}')
    print(f'LP plan = monotone plan  : {np.allclose(plan_lp, mono)}, '
          f'mass the LP plan moves leftward: {np.tril(plan_lp, -1).sum():.2f}')
```

```
W1 from the CDF gaps     : 1.7000
W1 from the LP           : 1.7000
cost of the monotone plan: 1.7000
LP plan = monotone plan  : False, mass the LP plan moves leftward: 0.00
```

3 つの方法はすべて $W_1 = 1.7$ で一致する。5 つの CDF の差を足し合わせるだけで、36 変数の線形計画問題を解いた場合と寸分違わぬ答えが得られるのである。ところが最後の行には小さな驚きがある。線形計画法は単調計画を返さなかったのである。その代わりに、まったく同じコストを持ち、同じく左方向へは一切質量を動かさない別の計画を見つけた。

つまり、この例では最適な W_1 計画は一意ではなく、その理由も見抜くのは容易である。至るところで $F_P \geq F_Q$ であるため、質量を左方向へ動かす必要がそもそもない。質量を右方向（$y \geq x$）にのみ動かす**いかなる**計画に対しても $|x - y| = y - x$ が成り立つため、経路がどのように交差しようとも、総コストは 2 つの平均の差 $3.35 - 1.65 = 1.7$ に等しくなる。図に描かれた単調計画は、同等に優れた多数の計画の 1 つにすぎない。

本書の他の箇所に登場する 2 つの手法では、絶対距離をその二乗に置き換える。画像生成器の評価に用いる Fréchet Inception Distance（FID）（16.5 節）と、二乗距離を最小化してノイズとデータを組にするフローマッチングのミニバッチ最適輸送（38.7 節）である。**2 次ワッサースタイン距離**（Wasserstein-2 distance）は、二乗距離の平均をとってから平方根をとった量 $W_2(P, Q) = \left(\inf_{\gamma} \mathbb{E}_{\gamma}\|x - y\|^2\right)^{1/2}$ である。二乗コストのもとでは、交差する 2 本の輸送リボンは交差を解くよりも常に真に大きなコストがかかるため、実直線上では単調計画が**唯一の**最適解となる（演習問題 2d）。平均 μ_1, μ_2、標準偏差 σ_1, σ_2 を持つ 2 つの 1 次元ガウス分布の間では、この距離は $W_2^2 = (\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2$ ときれいに求まる（演習問題 4a）。Fréchet Inception Distance はこの公式の多変量版であり、画像の特徴量に当てはめた多変量ガウス分布に対して分散を共分散行列に置き換えて適用したものである。

37.4.2.4 エントロピー正則化とシンクホーン反復

本項は初読時には読み飛ばしてもよい。1 次元を超えると一般に閉形式の CDF 公式は存在せず、両側に N 点ずつある線形計画問題は N^2 個の変数を持つ。最適輸送を高速かつ微分可能にする標準的な手法が**エントロピー正則化**（entropic regularization）である (Cuturi 2013)。コスト行列 $\mathbf{C}$ と正則化強度 $\varepsilon > 0$ が与えられたとき、$\varepsilon \sum_{ij} \gamma_{ij}(\log \gamma_{ij} - 1)$ を輸送コスト $\sum_{ij} \gamma_{ij} C_{ij}$

に加える。有効な計画はすべて $\sum_{ij}\gamma_{ij}=1$ を満たすため、この追加項は $-\varepsilon$ と γ のエントロピーの積に定数を足したものに等しく、少数の鋭い経路に全質量を集中させる計画を抑制する。エントロピー正則化こそが最適輸送を深層学習の実用的な損失関数へと変えた立役者であり、正則化輸送コストを用いて生成モデルを直接訓練することを可能にした (Genevay et al. 2018)。

驚くべきことに、正則化された問題はほぼ閉形式で解ける。ラグランジアン（35.7.1 節）を、乗数 ξ_i（各行和制約 $\sum_j\gamma_{ij}=p_i$ に対応）および乗数 ζ_j（各列和制約 $\sum_i\gamma_{ij}=q_j$ に対応）を用いて作り、γ_{ij} に関する導関数を 0 とおくと、$C_{ij}+\varepsilon\log\gamma_{ij}-\xi_i-\zeta_j=0$、すなわち $\gamma_{ij}=e^{-(C_{ij}-\xi_i-\zeta_j)/\varepsilon}$ が得られる。各経路が運ぶ質量は、そのコスト C_{ij} が $\xi_i+\zeta_j$ をどれだけ上回るかに応じて（ε を単位として）指数関数的に減衰する。行列形式で書けば次のようになる。

$$\gamma_\varepsilon=\operatorname{diag}(\mathbf{a})\,\mathbf{K}\,\operatorname{diag}(\mathbf{b}),\qquad \mathbf{K}=e^{-\mathbf{C}/\varepsilon}\ \text{(entrywise)},\tag{37.42}$$

ここで $\mathbf{a}=e^{\xi/\varepsilon}$ と $\mathbf{b}=e^{\zeta/\varepsilon}$ は正のスケーリングベクトルである。正則化輸送計画は、単なる**ギブスカーネル** $\mathbf{K}$（正の重みからなる行列であり、MMD のカーネルとは無関係である）を行ごと・列ごとにリスケーリングしたものにすぎない。求めるべき未知数は $\mathbf{a}$ と $\mathbf{b}$ だけであり、それぞれの周辺制約が一方を決定する。すなわち、$\mathbf{b}$ を固定すれば成分ごとに $\mathbf{a}=\mathbf{p}/(\mathbf{K}\mathbf{b})$ とおくことで行和 $\mathbf{p}$ が満たされ、$\mathbf{a}$ を固定すれば $\mathbf{b}=\mathbf{q}/(\mathbf{K}^\top\mathbf{a})$ とおくことで列和 $\mathbf{q}$ が満たされる。

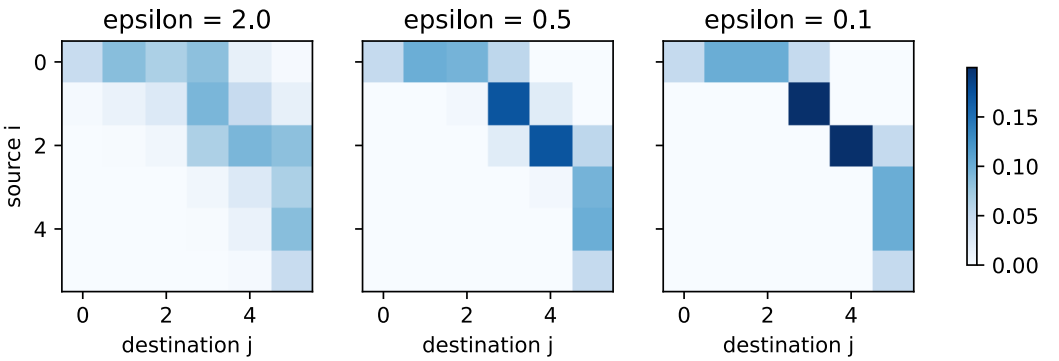
シンクホーン反復（Sinkhorn iterations）はこの 2 つの更新を交互に繰り返し、両方の周辺分布が一致するまで $\mathbf{b}$、次いで $\mathbf{a}$ を更新する。$\mathbf{a}$ の更新直後には行和は厳密に一致しているため、列和が $\mathbf{q}$ と誤差 10^{-9} 以内で一致した時点で停止する。各ステップで必要なのは行列・ベクトル積が 2 回だけであり、GPU 上で高速に動作するうえ、反復を通じた微分も容易である。以下では、正則化計画の（エントロピーペナルティを除いた）輸送コスト $\sum_{ij}(\gamma_\varepsilon)_{ij}C_{ij}$ を指して**エントロピーコスト**（entropic cost）として報告する。$\varepsilon\to0$ とすると、このコストは正則化なしの厳密な最適値に近づくが、収束が劇的に遅くなり (Peyré and Cuturi 2019)、行列 $\mathbf{K}$ の成分がアンダーフローを起こしうる。そのため実用上の実装では、対数空間で $\log\mathbf{a}$ と $\log\mathbf{b}$ を更新する（演習問題 5b）。

ε をどれほど小さくする必要があり、また小さな ε がどれほど高コストになるかを確かめるため、二乗コスト $C_{ij}=(x_i-x_j)^2$ を用いた 6 点の例にシンクホーン反復を適用しよう。この設定では Figure 37.10 の単調計画が唯一の最適解であり、10 本の有効な経路のコストは $W_2^2=3.5$ である。以下のセルでは線形計画問題でこの最適値を確認したあと、3 つの ε の値でシンクホーン反復を実行し、反復回数、エントロピーコスト、および 10^{-3} を超える質量を運ぶ経路の本数を報告する。

```
def sinkhorn(p, q, C, eps, tol=1e-9, max_iters=10 ** 6):
    """Entropic OT by Sinkhorn iterations, stopped once the column sums are
    within tol of q (L1); returns the plan, its cost and the iterations."""
    K = np.exp(-C / eps)                          # the Gibbs kernel
    a = np.ones_like(p)
    for it in range(1, max_iters + 1):
        b = q / (K.T @ a)                         # match the column sums
        a = p / (K @ b)                           # match the row sums ...
        if np.abs(b * (K.T @ a) - q).sum() < tol:  # ... columns still right?
            break
    plan = a[:, None] * K * b[None, :]
    return plan, np.sum(plan * C), it

C2 = (atoms[:, None] - atoms[None, :]) ** 2       # squared cost (x_i - y_j)^2
```

```
res2 = optimize.linprog(C2.ravel(), A_eq=A_eq,
                        b_eq=np.concatenate([p_w, q_w]), method='highs')
print(f'squared cost, LP optimum : {res2.fun:.5f}')
plans, eps_values = [], (2.0, 0.5, 0.1)
for eps in eps_values:
    plan, cost, iters = sinkhorn(p_w, q_w, C2, eps)
    plans.append(plan)
    print(f'epsilon = {eps}: {iters:6d} iterations, entropic cost = '
          f'{cost:.5f}, routes above 1e-3 = {(plan > 1e-3).sum()}')
d2l.show_heatmaps(d2l.tensor(np.stack(plans)[None]), 'destination j',
                  'source i', titles=[f'epsilon = {e}' for e in eps_values],
                  figsize=(9, 2.8), cmap='Blues')
```

```
squared cost, LP optimum : 3.50000
epsilon = 2.0:     47 iterations, entropic cost = 4.06489, routes above 1e-3 = 25
epsilon = 0.5:    186 iterations, entropic cost = 3.56963, routes above 1e-3 = 14
epsilon = 0.1: 154325 iterations, entropic cost = 3.50002, routes above 1e-3 = 10
```



線形計画法により $W_2^2 = 3.5$ が確認できる。$\varepsilon = 2$ では、正則化計画は各出発点の質量を多数の到着先ににじませ（25 本の経路が全 36 本中 10^{-3} 超の質量を運ぶ）、コストを 4.06 まで押し上げる。$\varepsilon = 0.5$ では有効な経路が 14 本に絞られ、コストは 3.57 となる。$\varepsilon = 0.1$ になると単調計画の 10 本の経路だけが生き残り、コストは 3.50002 に達して厳密な最適値との差が 2×10^{-5} 以内に収まる。ヒートマップでは、ぼやけた帯がくっきりとした階段状のパターンへと鋭さを増していく。ε の自然なスケールは競合する経路間のコスト差であり、このグリッド上ではオーダー 1 である。ε をそのスケールの 10 分の 1 に設定すれば、最適値を精度よく復元できる。

反復回数を見ると、その代償が明らかになる。シンクホーン反復が要した回数は約 50 回（$\varepsilon = 2$）および約 190 回（$\varepsilon = 0.5$）であったのに対し、約 150,000 回（$\varepsilon = 0.1$）へと跳ね上がった。ε を 5 分の 1 に縮めただけで、計算量は 800 倍以上に膨れ上がったのである。シンクホーン反復が高速なのは、ぼやけた輸送計画を許容できる場合に限られる（演習問題 6b でこの増大を定量的に追跡する）。

37.4.3 固定カーネルと訓練されたクリティック

MMD と W_1 は、2 つの要件をちょうど真っ二つに分け合っている。固定カーネルは小さく滑らかな関数クラスを定め、MMD は 3 つの標本平均に帰着し、N 個の点からの推定誤差は**いかなる**次元においても $O(N^{-1/2})$ で減衰する (Gretton et al. 2012)。残念ながら、だからといってカーネルが次元の呪い（curse of dimensionality）を免れるわけではない。誤差の収束レートは次元に依存しないものの、信号の大きさは次元に依存するからである。固定カーネルのもとでは、与え

られた 2 分布間の母集団 MMD は次元が高くなるにつれて縮小するのが一般的であり (Ramdas et al. 2015)、Kernel Inception Distance や MMD 損失が生のピクセル上ではなく学習された特徴量上で計算される理由はそこにある。また固定カーネルは、長さスケールの数倍を超えるとデータを見失ってしまう。これとは対照的に、1 リプシッツ関数はあらゆる間隔で安定した勾配を与え、まさにデータから遠く離れたモデルが必要とする性質を備えている。しかし、指示関数のように不連続なジャンプはできないとはいえ、すべての 1 リプシッツ関数からなるクラスは高次元では依然として膨大である。次元 $n > 2$ における 2 標本間では、経験 W_1 の誤差は $O(N^{-1/n})$ でしか減衰しない (Weed and Bach 2019)。100 次元空間では、その推定誤差を半分にするのに 2^{100} 倍ものデータが必要になる計算である。高次元のサンプルからそのように直接 W_1 を推定する者はいない。代わりに WGAN のクリティックははるかに小さなニューラルネットワークのクラス上を探索し、シンクホーン反復はより小規模な離散マッチング問題を扱うのである。

カーネルも敵対的クリティックも、まずはモデルからサンプルを抽出することから始まる。生成モデルによってはサンプリング自体がボトルネックとなる場合があり、続く 37.5 節では、モデルからサンプルを一切抽出せずに 2 つの分布を比較する方法を問う。

37.4.4 まとめ

裾の重なりは f ダイバージェンスを救わない。2 つの単位ガウス分布が標準偏差の 8 倍離れると、イェンセン・シャノン・ダイバージェンスと全変動の傾きはいずれも 10^{-4} 付近まで落ち込む。積分確率指標は、テスト関数のクラスを P と Q がどう平均するかを比較することで密度比を回避し、どのクラスを選ぶかによってサンプル効率と距離が見渡せる範囲の両方が決まる。固定された特徴に関して線形なテスト関数からは、平均特徴ベクトル間の距離である MMD が得られる。MMD はクリティックを訓練することなく 3 つのカーネル平均として閉形式で評価でき、次元によらず $O(N^{-1/2})$ のレートで推定できるが、次元が増すにつれて MMD 自体が縮小する傾向があり、また RBF カーネルは長さスケールの数倍を超えると盲目になる。傾きが高々 1 の関数からは W_1、すなわち P を Q へ輸送する「質量 × 距離」の最小コストが得られ、これはあらゆる間隔で傾き 1 を維持し、Wasserstein GAN のクリティックの基盤をなす。1 次元では W_1 は 2 つの CDF に挟まれた面積に帰着し、高次元では訓練されたクリティックを用いるか、小規模な離散問題に対してエントロピー正則化付きのシンクホーン反復を用いる必要がある。

37.4.5 演習問題

1. [code] **閉形式の MMD** ガウス積分 Equation 37.38 を長さスケール ℓ の RBF カーネルに適用し、$P = \mathcal{N}(0, 1)$ および $Q = \mathcal{N}(d, 1)$ とせよ。
    1. $\text{MMD}^2 = \frac{2\ell}{\sqrt{\ell^2+2}}\big(1 - e^{-d^2/(2(\ell^2+2))}\big)$ となることを示し、$\ell = 1$ のとき冒頭の表の MMD^2 の列と一致することを確かめよ。小さなシフト d に対して MMD^2 を最大化する長さスケールはいくらか。また $\ell \to 0$ および $\ell \to \infty$ のときどうなるか。
    2. d に関する傾きを $d = 8$ において $\ell = 1$、4、16 について求めよ。得られた 3 つの値は、データから遠く離れた位置から出発するモデルの長さスケールについて何を物語っているか。
2. [code] **1 次元の輸送●**実直線上では、W_1 は CDF 間の面積 Equation 37.41 である。
    1. 同じサイズの標本 $x_1, \dots, x_N$ と $y_1, \dots, y_N$ 上の 2 つの経験分布に対し、証明中の単調計画を用いて $W_1 = \frac{1}{N}\sum_i |x_{(i)} - y_{(i)}|$（ソートした値の平均絶対差）となることを示せ。サイズ 7 の 2 つの無作為標本において、6 点の例の線形計画問題と照合してこれを確認せよ。
    2. MMD の実験における `x_p` と `y_shift` の間、および `x_p` と `y_same` の間で、ソート標本による W_1 を計算せよ。次に、20 組の独立標本のペア（各サイズ N、$\mathcal{N}(0, 1)$ から

抽出）にわたって $N \in \{10^2, 10^3, 10^4\}$ のそれぞれで平均をとり、その減少レートを求めよ。この推定値は決して負にならない。$P = Q$ のとき、なぜこのことから不偏性が否定されるのか。

3. `q_w` を $(0.40, 0.05, 0.05, 0.10, 0.10, 0.30)$ に変更して 2 つの CDF が交差するようにし、W_1 を計算するセルを再実行せよ。証明のステップ 4 と同様に、テスト関数 g を $g(0) = 0$ かつ隣接アトム間の傾きが $\operatorname{sign}(F_Q(t) - F_P(t))$ となるように構成し、$\mathbb{E}_P[g] - \mathbb{E}_Q[g]$ が LP の値と一致することを確かめよ。LP の解 `plan_lp` において質量を運ぶすべての経路 (i, j) について、$g(x_i) - g(x_j)$ と $|x_i - x_j|$ を比較せよ。任意の最適計画と任意の最適テスト関数において、なぜ質量を運ぶすべての経路で両者が一致しなければならないのか。
4. **二乗コストのもとでの一意性**。ある計画が x から y' へ、かつ x' から y へ質量を動かしており、$x < x'$ かつ $y < y'$ であるため 2 本のリボンが交差しているとする。二乗コストのもとでは、同じ質量を x から y へ、x' から y' へ送ることで単位質量あたり $2(x' - x)(y' - y)$ だけコストが節約されることを示し、単調計画が唯一の最適計画であることを導け。4 つの移動がすべて右向きであるとき、コスト $|x - y|$ のもとで同じ入れ替えはどれだけコストを節約するか。6 点の例に対して線形計画法が返した計画 `plan_lp` の中から交差する 2 本のリボンを見つけ、自分の答えを確認せよ。

3. **直線上の最適クリティック** Equation 37.41 の証明のステップ 4 は、傾き $g'(t) = \operatorname{sign}(F_Q(t) - F_P(t))$ を持つテスト関数が W_1 を達成することを示している。

1. $P = \mathcal{N}(0, 1)$ および $Q = \mathcal{N}(d, 1)$（$d > 0$）に対し、至るところで $F_P > F_Q$ となり、ステップ 4 が本文で直観的に見出したクリティック $g(x) = -x$ を返すことを示せ。
2. $P = \mathcal{N}(0, 1)$ および $Q = \mathcal{N}(0, 4)$ に対し、2 つの CDF が 0 でのみ交差すること、$g(x) = -|x|$ が最適であること、および $W_1 = \sqrt{2/\pi} \approx 0.80$ となることを示せ（$X \sim \mathcal{N}(0, \sigma^2)$ に対して $\mathbb{E}|X| = \sigma\sqrt{2/\pi}$ である）。このペアに対して線形カーネルは 0 を報告していた。このクリティックは Q のサンプルをどちらの方向へ動かすか。
3. $P = \frac{1}{2}\delta_0 + \frac{1}{2}\delta_3$、$Q = \frac{1}{2}\delta_1 + \frac{1}{2}\delta_3$ とする。$W_1(P, Q)$ を求め、ステップ 4 で用いた形の最適クリティックをすべて記述せよ。さらに一般に、そのようなクリティックが最適となるのは、$g'(t) = \operatorname{sign}(F_Q(t) - F_P(t))$ がほとんどすべての t（質量が横切らなければならない $F_P(t) \neq F_Q(t)$ の点）で成り立つとき、かつそのときに限られ、それ以外の点での傾きは任意であることを示せ。
4. WGAN のクリティックは、データと生成サンプルの間の点における $(|g'(x)| - 1)^2$（点 x で評価）にペナルティを課して訓練されることが多い (Gulrajani et al. 2017)。この選択を (c) を用いて説明せよ。$P = Q$ のときどのクリティックが最適となり、このペナルティはそのうちのどれを選好するか。訓練においてその帰結がどうなるかは 16.4 節で示される。

4. [code] **ガウス分布間の W_2 とフレシェ距離**実直線上では、単調計画 $X = F_P^{-1}(U)$、$Y = F_Q^{-1}(U)$ が二乗コストに対して最適である。

1. $P = \mathcal{N}(\mu_1, \sigma_1^2)$ と $Q = \mathcal{N}(\mu_2, \sigma_2^2)$ に対し、単調計画が $Y = \mu_2 + (\sigma_2/\sigma_1)(X - \mu_1)$ で与えられることを示し、$W_2^2(P, Q) = (\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2$ を導け。同じ計画を用いて、等分散の場合に $W_1 = |\mu_1 - \mu_2|$（冒頭の表の W_1 の列）を復元せよ。
2. 両方の標準偏差を σ、平均の間隔を d とする。このとき $W_2^2 = d^2$ がすべての σ に対して成り立つ一方、2 つのガウス分布間の KL ダイバージェンスは $d^2/(2\sigma^2)$ である。$\sigma \to 0$ としてガウス分布が Table 37.2 の点質量へと近づくとき、両者はそれぞれ何を報告するか。
3. $\mathbb{R}^n$ において、W_2 の二乗（$\mathcal{N}(\mu_1, \Sigma_1)$ と $\mathcal{N}(\mu_2, \Sigma_2)$ の間）は、二乗**フレシェ距離**

$\|\mu_1-\mu_2\|^2+\mathrm{tr}\big(\Sigma_1+\Sigma_2-2\,(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}\big)$ に等しい (Dowson and Landau 1982)。$n=1$ のとき (a) に帰着すること、および共分散行列が対角行列のとき各座標にわたる (a) の和に帰着することを確かめよ。さらに、対角行列の場合を直接証明せよ（二乗コストは座標ごとの和であるため、いかなる計画のコストも座標ごとの最適値の和を下回らず、各座標の単調計画を組み合わせればその下界を達成する）。

4. Fréchet Inception Distance（FID）は 2 つの特徴量分布のそれぞれにガウス分布を当てはめ、当てはめた分布間の二乗フレシェ距離を報告する。P を混合分布 $0.7\,\mathcal{N}(-2,0.6^2)+0.3\,\mathcal{N}(2,0.6^2)$ とし、同じ平均と分散を持つ $Q=\mathcal{N}(-0.8,3.72)$ を考える。当てはめた 2 つのガウス分布間のフレシェ距離はいくらか。それぞれの分布から抽出した 10^5 個のソート済みサンプルから $W_2^2(P,Q)$ を推定せよ（約 0.63 になるはずである）。フレシェ距離が 0 であることは何を保証し、何を保証しないか。

5. [code] **エントロピー輸送とシンクホーンダイバージェンス●●**6 点の例の輸送 LP を、独立な計画からのダイバージェンスによって次のように正則化する。

$$\mathrm{OT}_\varepsilon(P,Q)=\min_{\gamma}\sum_{i,j}\gamma_{ij}C_{ij}+\varepsilon\,D_{\mathrm{KL}}(\gamma\,\|\,p\,q^\top),$$

ここで γ は行和 p、列和 q を持つ計画全体を動く。

1. このような任意の計画に対して $D_{\mathrm{KL}}(\gamma\,\|\,p\,q^\top)=-H(\gamma)+H(P)+H(Q)$（$H$ はエントロピー）が成り立ち、したがって `sinkhorn` が最小点を返すことを示せ。また $\mathrm{OT}_\varepsilon(P,Q)\to\sum_{i,j}p_iq_jC_{ij}$（$\varepsilon\to\infty$ のとき）となること、および $\mathrm{OT}_\varepsilon(P,P)>0$ がすべての $\varepsilon>0$ に対して（P が点質量でない限り）成り立つことを示せ。$C_{ij}=|x_i-x_j|$ のもとで、アトム上の分布 P のうち極限 $\sum_{i,j}p_iq_jC_{ij}$ を最小化するものは Q の中央値（`q_w` ではアトム 3 と 4）に全質量を集中させることを示せ。大きな ε を用いた OT_ε で訓練されたモデルは何を学習するか。
2. シンクホーンダイバージェンス $S_\varepsilon(P,Q)=\mathrm{OT}_\varepsilon(P,Q)-\tfrac12\mathrm{OT}_\varepsilon(P,P)-\tfrac12\mathrm{OT}_\varepsilon(Q,Q)$ はこの統計的バイアスを取り除く (Genevay et al. 2018; Feydy et al. 2019)。`p_w` と `q_w` に対し、絶対値コスト `C` のもとで $\varepsilon\in\{10^{-3},10^{-2},\dots,10^{3}\}$ における $S_\varepsilon(P,Q)$ を計算せよ。$\varepsilon=10^{-3}$ では $\mathbf{K}$ の成分がアンダーフローを起こすため、そこでは `scipy.special.logsumexp` を用いて $\log\mathbf{a}$ と $\log\mathbf{b}$ 上でシンクホーン更新を実行せよ。S_ε が W_1 に近づき（$\varepsilon\to0$ のとき）、また $\tfrac12\mathcal{E}(P,Q)$ に近づく（$\varepsilon\to\infty$ のとき）ことを確認せよ。ここで $\mathcal{E}(P,Q)=2\mathbb{E}|X-Y|-\mathbb{E}|X-X'|-\mathbb{E}|Y-Y'|$ はエネルギー距離 (Székely and Rizzo 2013) であり、Equation 37.37 のカーネルを $-|x-y|$ に置き換えた式である。2 つ目の極限を (a) から導出せよ。

6. [code] **小さな ε におけるシンクホーン反復●**6 点の例の `sinkhorn`、`p_w`、`q_w` を用いよ。

1. **早期停止**。二乗コスト `C2`、$\varepsilon=0.1$、`max_iters=5000` で `sinkhorn` を呼び出し、列和が収束する前にループが終了するようにせよ。返されるコストは最適値 3.5 を下回る。返された行列の列和から `q_w` を引いた値を出力せよ。どの到着先への質量が不足しており、なぜそれによってコストが下がるのか。行和 `p_w` と列和 `q_w` を持つ行列のコストが 3.5 未満になり得ないのはなぜか。
2. **反復回数**。`sinkhorn` が `C2` のもとで $\varepsilon\in\{2,1,0.5,0.3,0.2,0.15,0.1\}$ に対して要する反復回数を数え、回数の対数を $1/\varepsilon$ に対してプロットせよ。小さな ε における点列はどのような増大を示し、$\varepsilon=0.05$ に対していくつの反復回数を予測するか。
3. **同値な最適解**。絶対値コスト `C` のもとでは、質量を右方向にのみ動かすあらゆる計画が `p_w` と `q_w` に対して最適となる。`sinkhorn` を `C` のもとで $\varepsilon=0.1$ および 0.02 で実

行せよ。何本の経路が質量を運ぶか。極限の計画が $\gamma_{ij} = a_i b_j$（$j \geq i$ のとき）および 0（それ以外）という形を持つことを示せ。有効な計画上では、正則化項は $-\varepsilon$ とエントロピーの積に定数を足したものである。これを用いて、なぜ極限の計画が最適計画の中で最大エントロピーを持つ計画になるのかを説明せよ。

37.5 スコアに基づく比較

ソフトマックス層は 10 個のロジットの指数をとってその和で割ることで確率へと変換するが、その和の計算に頭を悩ませる者はいない。では、ニューラルネットワークに 10 個のクラスではなく、起こり得るすべての入力に対してスコアを割り当てさせる場合を想像してみよう。慣例により符号は反転させる。ネットワークはもっともらしい入力（ロジットであれば高くなる場所）には低く、もっともらしさの低い入力には高くなるエネルギー $E(\mathbf{x})$ を出力し、$\tilde{q}(\mathbf{x}) = e^{-E(\mathbf{x})}$ に比例する密度を定める。すると 10 個のクラスにわたる和は、起こり得るすべての入力にわたる積分 $Z = \int \tilde{q}(\mathbf{x})\, d\mathbf{x}$ へと置き換わるが、高次元においてこの積分はまったく計算困難である。このような**エネルギーベースモデル**は、ある入力が別の入力に比べてどれだけもっともらしいかは教えてくれるが、その実際の確率は教えてくれない。

これまでに構築してきた分布の比較手法は、いずれもこの状況に対処できない。最尤法には正規化された密度 $q = \tilde{q}/Z$ が必要である。クリティック（37.3.4 節）、カーネル、輸送計画（37.4 節）のいずれに基づくにせよ、サンプルに基づく手法は Z を回避できるが、モデルからサンプルを抽出する必要があり、エネルギー関数はそのままではサンプルを与えてくれない。各サンプルは、より低いエネルギーへ向けてノイズを伴う小さなステップを多数繰り返す長いランダムウォークによって生成しなければならず、その各ステップでネットワークを通じた勾配計算が必要になる（38.6.2 節）。

しかし、勾配を 1 回計算するだけなら安価である。必要なコストはわずか 1 回の逆伝播であり、実はそれだけで十分なのである。$\log \tilde{q}$ の入力 $\mathbf{x}$ に関する勾配は対数密度の傾きを測るものであり、定数の傾きは 0 であるため Z を完全に消去する。この勾配は**スコア**（またはスコア関数）と呼ばれ、本節ではスコアのみを用いて分布を比較する。2 つのスコア間の平均二乗差はスコアマッチングの目的関数を与え、これは第 17 章で扱う拡散モデル（diffusion model）の基盤をなしている。とはいえ、傾きの比較には、スコアに基づくモデルを訓練する者なら誰もが理解しておくべき重要な盲点が伴う。傾きは純粋に局所的な量であるため、遠く離れたモード間で確率質量がどう配分されているかを知ることができないのである。この盲点がどれほど深刻かを以下で定量化していく。

前節までと同様、サンプルを通じてのみ観測されるデータ分布を P と表し、モデルを Q と表す。ただし本節では、モデルについてはそのスコア s_Q のみが既知であるとする。ここで 2 つの中心的な問いを立てる。傾きの差は 2 つの分布について何を教え（何を見落とす）のか、そしてそもそもデータが有限個の点集合であって傾きを持たないとき、どうやって傾きどうしを比較できるのか、という問いである。38.6.1 節はこの 2 つの答えの上に直接築かれる。また 3 つ目の問いとして、サンプリングできないモデルから与えられたサンプルが生成されたかどうかを検定する方法も探るが、この部分は初読時には飛ばしても構わない。最後に、本章で扱ったすべての比較手法を訓練目的関数の表へとまとめ、それらをどう使い分けるかについての実践的な指針を示す。

```
from scipy import integrate
```

本節の通し例は、1 次元のガウス分布および 2 つのガウス分布の混合分布である。まずは 3 つの小さなヘルパ関数から始める。`log_components` と `log_mix` は最大の項をくくり出すことで裾がアンダーフローしないように混合ガウス分布（Gaussian mixture）の対数密度を数値的に安定して評価し（35.8.2 節）、`kl` はグリッド上の 2 つの密度間の KL ダイバージェンスを nat 単位で計算

する。

```
def log_components(x, w, mus, sigmas):
    """log(w_k N(x; mu_k, sigma_k^2)) for every component k; shape (k, len(x))."""
    return (np.log(w)[:, None] - 0.5 * np.log(2 * np.pi * sigmas[:, None] ** 2)
            - (x - mus[:, None]) ** 2 / (2 * sigmas[:, None] ** 2))

def log_mix(x, w, mus, sigmas):
    """Log-density of the mixture sum_k w_k N(mu_k, sigma_k^2), computed stably."""
    c = log_components(x, w, mus, sigmas)
    c_max = c.max(axis=0)
    return c_max + np.log(np.exp(c - c_max).sum(axis=0))

def kl(log_a, log_b, x):
    """D_KL(A||B) by quadrature on the grid x; in nats."""
    return integrate.trapezoid(np.exp(log_a) * (log_a - log_b), x)
```

37.5.1 スコアとフィッシャーダイバージェンス

37.5.1.1 スコア

空間内のあらゆる点において、スコアは「密度はどの方向に、どれだけ急峻に上昇するか」という単純な幾何学的疑問に答える。形式的には、分布 P が微分可能な密度 p を持つとき、**スコア**とは**データ点に関する**対数密度の勾配であり、次式で定義される。

$$s_P(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}). \tag{37.43}$$

古典統計学では「スコア」が対数尤度の**パラメータ**に関する勾配を指すため（36.6 節）、混同しないよう注意してほしい。ここでは $\mathbf{x}$ に関して微分する。1 次元ガウス分布 $\mathcal{N}(\mu, \sigma^2)$ の場合、スコアは単純に次の式となる。

$$s(x) = -\frac{x - \mu}{\sigma^2},$$

これは x を平均 μ へと引き戻す、剛性 $1/\sigma^2$ のばねのように働く。混合分布 $p = \sum_k w_k\, p_k$ に対して連鎖律を適用すると、$s(x) = \sum_k r_k(x)\, s_k(x)$ が得られる。ここで $r_k(x) = w_k p_k(x)/p(x)$ は、x が成分 k から生じた事後確率、すなわち**負担率**（responsibility）である。したがって各点は、各成分のばねに重みを付けて合成した力を受ける。どのモードの近傍でも、そのモードの負担率は 1 に近いため、点はそのモード単独のばねの力を受ける。一方、モード間の谷に沿っては、Figure 37.11 に示すように、負担率が一方のアトラクタから次のアトラクタへと滑らかに支配権を引き渡す。以下ではこの式をそのまま実装し、数値安定性のために負担率を対数空間で計算する。

```
def mix_score(x, w, mus, sigmas):
    """Score of a Gaussian mixture: responsibility-weighted component scores."""
    r = np.exp(log_components(x, w, mus, sigmas) - log_mix(x, w, mus, sigmas))
    return np.sum(r * -(x - mus[:, None]) / sigmas[:, None] ** 2, axis=0)
```

エネルギーベースモデルにおいてスコアがこれほど有用なのは、いかなる正規化定数も無視できるからである。$q(\mathbf{x}) = \tilde{q}(\mathbf{x})/Z$ であれば、次式が成り立つ。

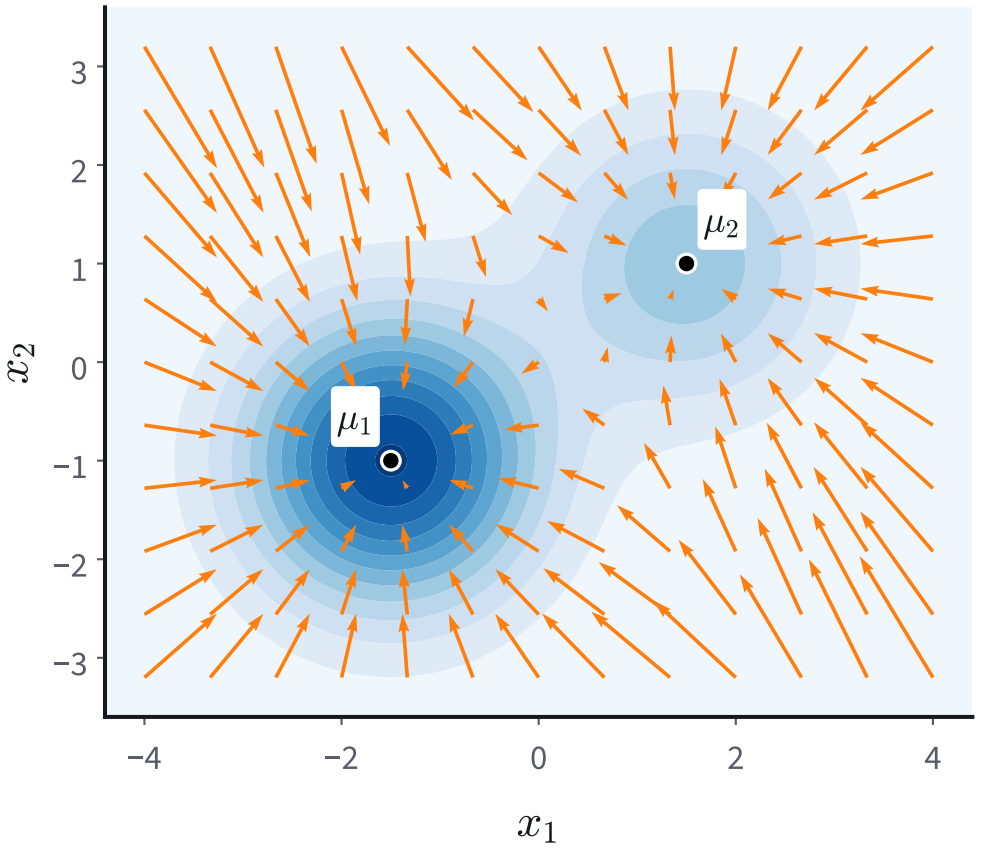


Figure 37.11: 2 成分混合ガウス分布のスコア場。矢印は対数密度の勾配を表す。各モードの近傍ではそのモードの平均へ引き寄せるばねとして働き、モード間の谷に沿っては負担率が一方のアトラクタからもう一方のアトラクタへと点を引き渡す。参考として背後に密度の等高線を描いている。

$$\nabla_{\mathbf{x}} \log q(\mathbf{x}) = \nabla_{\mathbf{x}} \log \tilde{q}(\mathbf{x}) - \nabla_{\mathbf{x}} \log Z = \nabla_{\mathbf{x}} \log \tilde{q}(\mathbf{x}),$$

これは Z が $\mathbf{x}$ に依存しないためである。たとえば、非正規化 4 次密度 $\tilde{q}(x) = e^{-x^4/4}$ のスコアは $-x^3$ であり、$Z \approx 2.56$ であるという事実を一切知ることなく、あらゆる点でこのスコアを評価できる。

37.5.1.2 フィッシャーダイバージェンス

スコアからは Z が消えるため、スコアから構成した比較尺度は非正規化モデルにそのまま適用できる。最も自然な選択は、データのもとで平均した 2 つのスコア場の平均二乗差であり、これは**フィッシャーダイバージェンス**として知られる。

$$D_{\mathrm{F}}(P\|Q) = \frac{1}{2}\, \mathbb{E}_{\mathbf{x}\sim P}\Big[\big\|s_P(\mathbf{x}) - s_Q(\mathbf{x})\big\|^2\Big]. \tag{37.44}$$

明らかに $D_{\mathrm{F}}(P\|Q) \geq 0$ であり、両方の密度が同一の連結領域上で狭義に正である限り、これは $P = Q$ のとき、かつそのときに限り 0 となる真のダイバージェンスである（37.3.1.1 節）。連結領域上では、スコアが一致すると $\log p - \log q$ が定数とならざるを得ず、両方の密度の積分は 1 でなければならないため、その定数は 0 に限られる。

2 つのガウス分布を比較すると、フィッシャーダイバージェンスと KL ダイバージェンスの重要な違いが浮かび上がる。両方の分布が分散 σ^2 を持ち、平均が距離 Δ だけ離れているとき、両者のスコアは定数 Δ/σ^2 だけすべての x において異なるため、次式が得られる。

$$D_{\mathrm{F}} = \frac{\Delta^2}{2\sigma^4} \qquad \text{whereas} \qquad D_{\mathrm{KL}} = \frac{\Delta^2}{2\sigma^2},$$

ここで 37.1 節のガウス分布に対する KL の公式を用いた。$\sigma = 1$ のとき、2 つの値は一致する。ここで座標 x を 2 倍にスケール変換し、Δ と σ の両方を 2 倍にしてみよう。KL ダイバージェンスは変化しないのに対し、フィッシャーダイバージェンスは元の値の 4 分の 1 に減少する。無次元量（nat 単位で測られる）である KL ダイバージェンスとは異なり、フィッシャーダイバージェンスは傾きの二乗であるがゆえに $1/x^2$ の単位を持つ。したがってその数値はデータのスケーリングに依存するため、あるデータセット上のスコアマッチング損失を別のデータセット上の損失と直接比較することはできず、nat 単位の符号化バジェットとして解釈することもできない。分散が異なる場合については、演習問題 1a で導出する。

37.5.1.3 遠く離れたモードの混合重み

Equation 37.44 の期待値は、データ分布 P のもとで傾きの差を平均する。データが十分に離れた複数のモードを持つとき、この 2 つの要素は相性が悪い。まったく同じガウス成分から構成され、混合重み（mixing weight）だけを入れ替えた 2 つの混合分布を考えてみよう。

$$P_w = w\,\mathcal{N}(-m, 1) + (1 - w)\,\mathcal{N}(m, 1), \qquad w = 0.7 \text{ against } w = 0.3.$$

分布としては、この 2 つの混合分布は大きく離れている。モードどうしが十分に離れると、両者の KL ダイバージェンスは離散重みベクトル $(0.7, 0.3)$ と $(0.3, 0.7)$ の間の KL ダイバージェンスに近づき、その値は $0.7 \log \frac{7}{3} + 0.3 \log \frac{3}{7} = 0.339$ nat となって m がどれほど大きくなっても変わらない。

しかし、両者のスコアはほとんど異ならない。どちらのモードの近傍でも、**両方**の混合分布においてそのモードの負担率は 1 に近いため、両方のスコア場はともにそのモード単独のばねへと帰着する。陽に書き下すと（演習問題 2b で導出するように）、P_w のスコアは次のようになる。

$$s_w(x) = -x + m\tanh(mx - c_w), \qquad c_w = \tfrac{1}{2}\log\frac{w}{1-w}.$$

この式は、十分左側ではばね $-(x+m)$ のように、十分右側ではばね $-(x-m)$ のように振る舞い、$x = c_w/m$ 付近で両者の間を遷移する。混合重み w が、この遷移の位置を通じて**のみ**現れることに注目してほしい。$m = 5$ のとき、遷移の位置は $x = 0.085$（$w = 0.7$ の場合）および $x = -0.085$（$w = 0.3$ の場合）となり、まさに 2 つのモードに挟まれた空っぽな谷の真ん中にある。そこでの密度は $e^{-m^2/2}/\sqrt{2\pi} \approx 1.5 \times 10^{-6}$ にすぎず、データは 2 つのスコアが食い違う唯一の場所をほとんど決して訪れない。m を大きくしながら両方のダイバージェンスを数値積分で計算してみよう。KL ダイバージェンスが 0.339 nat 付近にとどまる一方、フィッシャーダイバージェンスはおよそ $e^{-m^2/2}$ のように急減すると予想される。

```
xs = np.linspace(-14.0, 14.0, 14001)
w_a, w_b, unit = np.array([0.7, 0.3]), np.array([0.3, 0.7]), np.ones(2)
for m in (1.0, 3.0, 5.0):
    mus_m = np.array([-m, m])
    log_a, log_b = log_mix(xs, w_a, mus_m, unit), log_mix(xs, w_b, mus_m, unit)
    score_gap = mix_score(xs, w_a, mus_m, unit) - mix_score(xs, w_b, mus_m, unit)
    fisher = 0.5 * integrate.trapezoid(np.exp(log_a) * score_gap ** 2, xs)
    print(f'm = {m:.0f}: KL = {kl(log_a, log_b, xs):.3f} nats, '
          f'Fisher = {fisher:.1e}')
```

```
m = 1: KL = 0.184 nats, Fisher = 1.1e-01
m = 3: KL = 0.337 nats, Fisher = 7.1e-03
m = 5: KL = 0.339 nats, Fisher = 4.0e-06
```

予想どおり、KL ダイバージェンスが 0.339 nat に落ち着く一方、フィッシャーダイバージェンスは約 0.1 から約 4×10^{-6} へと 4 桁以上も急落する。$m = 5$ では $e^{-m^2/2} = e^{-12.5} \approx 4 \times 10^{-6}$ のオーダーとなり、ほぼ空の谷から予測されたとおりである。純粋にスコアのみから構築された損失は、遠く離れたモード間で確率質量がどう分けられているかを見ることができない。先ほど**連結**領域にこだわったのはこのためである。2 つの非連結な領域上では、スコアが同一であっても $\log p - \log q$ が各領域で異なる定数をとり得る。その結果、フィッシャーダイバージェンスの最小化によって訓練されたモデルは、すべてのモードの位置と形状を正しく捉えながら、自らの質量の 30% を、データが 70% を置いている場所に割り当ててしまうことがあり得る (Wenliang and Kanagawa 2020)。

幸い、ノイズを加えれば谷は埋まる。両方の混合分布に標準偏差 2 の独立なガウスノイズを加えると、各成分の分散は 1 から 5 へと広がる。標準偏差の 10 倍離れていたモードどうしは標準偏差の 4.5 倍の距離にまで縮まり、谷における密度は 1.5×10^{-6} から約 0.015 へと上昇する。こうなれば、2 つのスコアが食い違う領域をデータが実際に訪れるようになる（演習問題 2c でこの改善幅を定量化する）。拡散モデルがスコアマッチングを行う前に多数のスケールにわたるノイズでデータを摂動させる大きな理由はここにあり、17.1 節では訓練済みのニューラルネットワーク上でまさにこの盲点を計測する。

37.5.2 スタインの恒等式

フィッシャーダイバージェンスには、さらに直接的な 2 つ目の障壁がある。データ分布のスコア s_P に依存するにもかかわらず、データはサンプルとしてしか観測できないという点である。有限

個の点集合には傾きが存在しない。s_P がどこに現れるかを見るため、1 次元で二乗を展開してみよう。

$$D_{\mathrm{F}}(P\|Q) = \tfrac{1}{2}\,\mathbb{E}_P[s_P^2] - \mathbb{E}_P[s_P\, s_Q] + \tfrac{1}{2}\,\mathbb{E}_P[s_Q^2]. \tag{37.45}$$

第 3 項はモデルが提供する関数のデータに関する期待値であるため、サンプルから直接推定できる。第 1 項は Q に一切依存しないため、モデルの当てはめやランキングにおいては無関係な定数である。厄介なのはすべて中央の交差項にあり、そこでは未知のデータスコアとモデルスコアが掛け合わされている。驚くべきことに、あらゆる滑らかな密度が満たすたった 1 つの恒等式によって、この交差項は完全に消去される。以下では 1 次元の密度に対してこの恒等式を述べて証明し、それを 2 回活用する。1 回目はデータ分布に適用して交差項を取り除くためであり、2 回目はモデル分布に適用してサンプルが Q から得られたものかを検定するためである。

命題（スタインの恒等式）$p > 0$ を $\mathbb{R}$ 上の微分可能な密度とし、そのスコアを $s = (\log p)'$ とする。また g を微分可能な関数とし、$g(x)\,p(x) \to 0$（$x \to \pm\infty$ のとき）、$\mathbb{E}_p|g'| < \infty$、および $\mathbb{E}_p|g\,s| < \infty$ を満たすとする。このとき、次が成り立つ。

$$\mathbb{E}_{x\sim p}\big[g'(x) + g(x)\,s(x)\big] = 0. \tag{37.46}$$

証明：定義より $s = p'/p$ であるため、$p' = s\,p$ となる。部分積分（34.6.2.3 節）を行うと、次式が得られる。

$$\int g'\,p\,dx = \big[g\,p\big]_{-\infty}^{\infty} - \int g\,p'\,dx = -\int g\,s\,p\,dx,$$

ここで、g に関する仮定より境界項は消え、両方の積分は収束する。■

標準正規分布では $s(x) = -x$ であり、スタインの恒等式は $\mathbb{E}[g'(X)] = \mathbb{E}[X g(X)]$ となる。たとえば $g(x) = x$ と置けば、直ちに $\mathbb{E}[X^2] = 1$ が得られる。このガウス分布の場合はスタインの補題（Stein's lemma）(Stein 1981) であり、一般形は (Stein 1986) によるものである。この恒等式がこれほど強力なのは、傾きに関する知識を期待値に関する等式へと変換してくれるからであり、期待値こそサンプルから推定できる量だからである。

37.5.2.1 スコアマッチング

データ分布 P は、そこからのサンプルしか観測できないものの、滑らかな密度を持つと仮定しよう。スタインの恒等式を P に適用し（$s = s_P$）、モデルのスコアをテスト関数に選ぶと（$g = s_Q$）、$\mathbb{E}_P[s_Q' + s_Q\, s_P] = 0$ が得られる。したがって Equation 37.45 の交差項は $-\mathbb{E}_P[s_Q']$ となり、s_P が完全に消去される。

$$D_{\mathrm{F}}(P\|Q) = \mathbb{E}_{x\sim P}\Big[\tfrac{1}{2}\,s_Q(x)^2 + s_Q'(x)\Big] + \tfrac{1}{2}\,\mathbb{E}_P[s_P^2]. \tag{37.47}$$

部分積分によって、導関数を持たないデータ側からモデル側へと微分が移されたのである。末尾の項 $\frac{1}{2}\mathbb{E}_P[s_P^2]$ はどのモデルに対しても同じ値であるため、フィッシャーダイバージェンスの最小化は最初の期待値の最小化と等価になる。それはモデル自身が提供する量のデータ点にわたる平均であり、Z も s_P も必要としない。この手法は**スコアマッチング**と呼ばれ、Hyvärinen (2005) が非正規化モデルを当てはめるために導入し、自然画像の画像パッチ上で実証したものである。

37.5.1 節で導入した非正規化 4 次モデルを見ると、この手法が実際にどれほど鮮やかに機能するかがわかる。スケールパラメータ $\theta > 0$ を $\tilde{q}_\theta(x) = e^{-\theta x^4/4}$ によって導入すると、$s_Q(x) = -\theta x^3$ および $s'_Q(x) = -3\theta x^2$ となる。データにわたる標本平均を上線で表すと、Equation 37.47 のモデルに依存する部分は θ に関する単純な二次関数で推定される。

$$\overline{\tfrac{1}{2}\,s_Q^2 + s'_Q} = \tfrac{1}{2}\,\theta^2\,\overline{x^6} - 3\,\theta\,\overline{x^2}, \qquad \text{which is minimized at} \qquad \hat{\theta} = \frac{3\,\overline{x^2}}{\overline{x^6}}.$$

わずか 2 つの標本モーメントだけで、$e^{-\theta x^4/4}$ を一度も積分することなくパラメータを当てはめることができる。最尤法では $Z(\theta)$ が必要となり異なる推定量が導かれるが（演習問題 4）、ガウスモデルの場合は両者の推定量は一致する（演習問題 3）。

1 次元では、s'_Q は導関数をもう 1 回とるだけで済む。しかし $\mathbb{R}^n$ ではヤコビ行列（Jacobian）のトレース、すなわち n 個の二階導関数（座標ごとに 1 つ）の和となるため、スコアの計算では 1 回で済んだ逆伝播が n 回の逆伝播へと膨れ上がり、画像に対してはあまりにも高コストになる。38.6.1 節ではスコアマッチングを $\mathbb{R}^n$ へと拡張し、データにノイズを加えることでこれらの二階導関数の計算を回避できることを示したうえで、その結果から拡散モデルの訓練損失を導出する。

37.5.2.2 サンプルに対する検定

スタインの恒等式は逆向きにも使える。すなわち、与えられたサンプルがあるモデル Q から得られたものかどうかを検定する用途である。モデル自身にこの恒等式を適用すると、条件を満たすすべてのテスト関数 g に対して、量 $g' + g\,s_Q$ は Q のもとで平均 0 を持つ。g を $g' + g\,s_Q$ へ写す線形作用素は、Q の**スタイン作用素**（Stein operator）と呼ばれる。$g' + g\,s_Q$ の標本平均が、ある g に対して 0 から大きく離れていれば、そのサンプルが Q からサンプリングされたものではないという根拠が得られる。この検査に必要なものに注目してほしい。サンプル点において評価したモデルのスコアだけでよく、正規化定数 Z もモデルからのサンプルも一切不要である。

単一のテスト関数でどれだけの違いを捉えられるだろうか。モデルを $Q = \mathcal{N}(0,1)$ とすると、そのスコアは $-x$ であるため、スタイン作用素は g を $g'(x) - x\,g(x)$ へと写す。$g(x) = 1$ を選ぶと $-x$ となり標本平均を検査し、$g(x) = x$ を選ぶと $1 - x^2$ となり二次モーメントを 1 と照合する。単純なモーメントとしての解釈を持たない 3 つ目の選択肢として、$g(x) = \sin x$ を試してみよう。以下では 200,000 個の点を 3 つの分布、すなわち真のモデル $\mathcal{N}(0,1)$、平均をずらした分布 $\mathcal{N}(0.5,1)$、および裾を広げた分布 $\mathcal{N}(0,4)$ からサンプリングし、各テスト関数の標本平均をその標準誤差とともに出力する。最初の 2 つのモーメントだけから、平均をずらしたサンプルでは -0.5 と $1 - 1.25 = -0.25$、広げたサンプルでは 0 と $1 - 4 = -3$ になると予想される。

```
rng = np.random.default_rng()
n = 200_000
# Stein operator of the model Q = N(0, 1), whose score is -x:  g'(x) - x g(x)
tests = [('g = 1', np.ones_like, np.zeros_like),
         ('g = x', lambda t: t, np.ones_like),
         ('g = sin x', np.sin, np.cos)]
samples = [('N(0, 1)', rng.normal(0.0, 1.0, n)),
           ('N(0.5, 1)', rng.normal(0.5, 1.0, n)),
           ('N(0, 4)', rng.normal(0.0, 2.0, n))]
print(f'{"sample":10s}' + ''.join(f'{name:>18s}' for name, _, _ in tests))
for s_name, x in samples:
    vals = [g_prime(x) - x * g(x) for _, g, g_prime in tests]
    print(f'{s_name:10s}' + ''.join(f'{v.mean():+10.3f} ± {v.std() / np.sqrt(n):.3f}'
```

```
                                    for v in vals))
```

```
sample                 g = 1             g = x          g = sin x
N(0, 1)       -0.000 ± 0.002    -0.001 ± 0.003    +0.000 ± 0.002
N(0.5, 1)     -0.501 ± 0.002    -0.253 ± 0.004    -0.147 ± 0.002
N(0, 4)       -0.002 ± 0.004    -3.005 ± 0.013    -0.404 ± 0.003
```

モデル自身のサンプルに対しては、すべての標本平均が 0 から標準誤差の数倍以内に収まる。平均をずらしたサンプルに対しては 3 つのテスト関数すべてが容易にずれを捉え、それぞれ約 -0.50、-0.25、-0.15 という値を報告する。一方、広げたサンプルでは様相が異なる。平均が依然として 0 であるため、$g = 1$ は 0 に近い値（0.01 程度以内）を報告し、分散における 4 倍もの誤差を完全に見逃してしまう。不一致を検出できるのは $g(x) = x$（-3 付近）と $g(x) = \sin x$（-0.4 付近）だけである。個々のテスト関数はサンプルの 1 つの側面しか調べないため、誤った分布であってもその側面においてモデルと容易に一致し得るのである。

なぜ $g = 1$ が平均のシフトを捉えながら分散の違いを見逃すのかを理解するため、任意の滑らかなデータ分布 P がそれ自身のスタインの恒等式 $\mathbb{E}_P[g'] = -\mathbb{E}_P[g\, s_P]$ を満たすことに注目しよう。これをモデルのスタイン期待値に代入すると、次式が得られる。

$$\mathbb{E}_P[g' + g\, s_Q] = \mathbb{E}_P[(s_Q - s_P)\, g]. \tag{37.48}$$

つまりテスト関数の期待値は、2 つのスコアの差に g の重みを付けて平均したものにほかならない。平均をずらしたサンプルでは、そのスコア差は定数 $-x + (x - 0.5) = -0.5$ となり、$g = 1$ がこれを余すところなく捉える。一方、広げたサンプルではスコア差が $-x + x/4 = -\tfrac{3}{4}x$ という奇関数になるため、任意の対称な密度に対して $g = 1$ の重みで積分すると 0 になってしまう。いかなる単一のテスト関数も、起こり得るすべてのスコア差と相関を持つことはできないのである。

37.5.3 カーネルスタイン不一致度

1 つのテスト関数では 1 つの性質しか検査できないため、37.4.1 節で最大平均値差異（maximum mean discrepancy、MMD）に対して行ったのと同じ手法をとる。すなわち、再生核特徴空間（reproducing kernel feature space）の単位球に含まれるすべての関数という関数族全体にわたって一度に検定を行い、最大の違反量をとるのである。驚くべきことに、無限に多くのテスト関数にわたるこの上限もまた、データ点のペアに対するカーネルの評価値から構成される閉形式へと帰着する。

$\mathcal{A}_Q$ を Q のスタイン作用素とし、$(\mathcal{A}_Q g)(x) = g'(x) + s_Q(x)\, g(x)$ で定義する。スタインの恒等式によれば $\mathbb{E}_Q[\mathcal{A}_Q g] = 0$ が成り立つため、P がモデルの恒等式にどれだけ大きく違反しているかを $\sup_g \mathbb{E}_P[\mathcal{A}_Q g]$ によって測る。

テスト関数として $g(x) = \langle \mathbf{w}, \varphi(x) \rangle$ の形のものを考える。ここで φ は特徴写像であり、$\|\mathbf{w}\| \le 1$ とする。線形性により、スタイン作用素は特徴量に直接作用して $(\mathcal{A}_Q g)(x) = \langle \mathbf{w}, \psi(x) \rangle$ となる。ただし $\psi = \varphi' + s_Q\, \varphi$ である。コーシー・シュワルツの不等式により、単位球上における $\langle \mathbf{w}, \mathbb{E}_P[\psi] \rangle$ の上限は単純に $\big\| \mathbb{E}_P[\psi] \big\|$ となる。このノルムを二乗すると $\mathbb{E}\langle \psi(x), \psi(x') \rangle$（ただし $x, x' \sim P$ は 2 つの独立なサンプル）が得られる。内積を展開すると 4 つの項が現れ、それぞれがカーネル $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ とその偏微分によって次のように表される。

$$\begin{aligned} \langle \varphi'(x), \varphi(x') \rangle &= \partial_x k, \qquad \langle \varphi(x), \varphi'(x') \rangle = \partial_{x'} k, \\ \langle \varphi'(x), \varphi'(x') \rangle &= \partial_x \partial_{x'} k. \end{aligned}$$

これら 4 つの項にスコアの重みを付けてまとめることで、**スタインカーネル**（Q に対するカーネル）が次のように定義される。

$$\begin{aligned} k_Q(x,x') = s_Q(x)\,s_Q(x')\,k(x,x') + s_Q(x)\,\partial_{x'}k(x,x') \\ + \partial_x k(x,x')\,s_Q(x') + \partial_x\partial_{x'}k(x,x'). \end{aligned} \tag{37.49}$$

すると、二乗**カーネルスタイン不一致度**（kernel Stein discrepancy、KSD）は、データからの 2 つの独立なサンプルにわたるスタインカーネルの期待値となる。

$$\mathrm{KSD}^2(P,Q) = \mathbb{E}_{x,x'\sim P}\big[k_Q(x,x')\big]. \tag{37.50}$$

この期待値は P のみに依存するため、標本 $x_1,\dots,x_N$（P から抽出）から次の不偏推定量が得られる。

$$\widehat{\mathrm{KSD}}^2 = \frac{1}{N(N-1)}\sum_{i\neq j} k_Q(x_i,x_j),$$

これは MMD で用いたのと同じ非対角ペアの平均である。統計学では、異なる組にわたるこの形の平均は U 統計量（U-statistic）と呼ばれ、以下の関数名 `ksd2_ustat` もそこに由来する。

KSD が実際に何を測っているかを理解するため、Equation 37.48 を φ の各座標に適用しよう。すると $\mathbb{E}_P[\psi] = \mathbb{E}_P[(s_Q - s_P)\,\varphi]$ が得られ、ノルムを二乗すると次式になる。

$$\mathrm{KSD}^2(P,Q) = \mathbb{E}_{x,x'\sim P}\big[(s_Q - s_P)(x)\,k(x,x')\,(s_Q - s_P)(x')\big]. \tag{37.51}$$

すなわち KSD は、フィッシャーダイバージェンス $D_{\mathrm{F}}(P\|Q)$ をカーネルで平滑化した親戚にあたる。各点でのスコア差を個別に二乗する代わりに、データ点のペアにわたってスコア差を掛け合わせ、$k(x,x')$ で重みを付けるのである。残念ながら、これは KSD がモード重みの盲点を受け継ぐだけでなく、さらに悪化させることを意味する。37.5.1 節の離れた混合分布に対しては、ペアの**両方**の点がスコアの異なる空っぽの谷に落ちなければならなくなるため、KSD^2 はフィッシャーダイバージェンスの減衰レートの二乗にあたる e^{-m^2} 程度で減衰する（演習問題 2e）。

両方のスコアが既知であるとき、Equation 37.51 を使えば母集団での正確な値を予測できる。データを $P = \mathcal{N}(0,1)$ とし、3 つの候補モデルに対して検定してみよう。真のモデル $\mathcal{N}(0,1)$ に対してはスコア差が消えるため、$\mathrm{KSD}^2 = 0$ となる。平均をシフトしたモデル $\mathcal{N}(1,1)$ に対しては至るところでスコア差が $-(x-1) - (-x) = 1$ となるため、$\mathrm{KSD}^2 = \mathbb{E}[k(X,X')]$ となる。RBF カーネル $k(x,x') = e^{-(x-x')^2/(2\ell^2)}$（長さスケール ℓ）のもとでは、これは 37.4.1 節で出会ったガウス積分と同じである。$X - X' \sim \mathcal{N}(0,2)$ より期待値は $(1 + 2/\ell^2)^{-1/2}$ に等しく、$\ell = 1$ において $\mathrm{KSD}^2 = 1/\sqrt{3} \approx 0.577$ を与える。

3 つ目の候補は、非正規化 4 次密度 $\tilde{q}(x) = e^{-x^4/4}$（37.5.1 節）であり、その正規化定数は一切必要ない。スコアは $-x^3$ であるためスコア差は $x - x^3$ となり、結果として得られる Equation 37.51 の二重積分は初等的な閉形式を持たないため、以下の `ksd2_population` ではグリッド上の数値積分でこれを評価する（ここでそれができるのは、この合成ベンチマークでは真のデータ密度がわかっているからにほかならない）。

1 次元の RBF カーネルに対し、Equation 37.49 に現れる導関数は $\partial_{x'}k = \frac{x-x'}{\ell^2}\,k$、$\partial_x k = -\frac{x-x'}{\ell^2}\,k$、および $\partial_x\partial_{x'}k = \big(\frac{1}{\ell^2} - \frac{(x-x')^2}{\ell^4}\big)\,k$ である。以下のコードでは、`dk_dy` が第 2 引数に

関する導関数を表し、kappa が値 $k_Q(x_i, x_j)$ の行列を格納する。ここでは 2,000 個の点からなる $\mathcal{N}(0,1)$ の単一標本を 3 つのモデルすべてに対して検定し、ksd2_ustat には各モデルのスコア関数だけを渡す。

```
def ksd2_ustat(x, score, ell=1.0):
    """U-statistic KSD^2 with the RBF kernel exp(-(a-b)^2 / (2 ell^2))."""
    d = x[:, None] - x[None, :]
    k = np.exp(-d ** 2 / (2 * ell ** 2))
    dk_dx = -d / ell ** 2 * k
    dk_dy = d / ell ** 2 * k
    d2k = (1 / ell ** 2 - d ** 2 / ell ** 4) * k
    s = score(x)
    kappa = (s[:, None] * s[None, :] * k + s[:, None] * dk_dy
             + dk_dx * s[None, :] + d2k)
    n = len(x)
    return (kappa.sum() - np.trace(kappa)) / (n * (n - 1))

def ksd2_population(score, ell=1.0):
    """KSD^2 of the data P = N(0, 1) against a model with this score, from
    the score difference s_Q - s_P weighted by the kernel, by quadrature."""
    t = np.linspace(-10.0, 10.0, 2001)
    h = np.exp(-t ** 2 / 2) / np.sqrt(2 * np.pi) * (score(t) + t)
    k = np.exp(-(t[:, None] - t[None, :]) ** 2 / (2 * ell ** 2))
    return h @ k @ h * (t[1] - t[0]) ** 2

x_p = rng.standard_normal(2000)          # the data sample, P = N(0, 1)
for name, s in [('true model  N(0,1)', lambda t: -t),
                ('wrong model N(1,1)', lambda t: -(t - 1)),
                ('quartic exp(-x^4/4)', lambda t: -t ** 3)]:
    print(f'KSD^2 vs the {name:19s}: {ksd2_ustat(x_p, s):+.4f}  '
          f'(population {ksd2_population(s):.4f})')
```

```
KSD^2 vs the true model  N(0,1) : +0.0015  (population 0.0000)
KSD^2 vs the wrong model N(1,1) : +0.5681  (population 0.5774)
KSD^2 vs the quartic exp(-x^4/4): +0.4494  (population 0.3208)
```

真のモデルに対しては、推定値は 0 から数千分の 1 以内に収まる（0 の不偏推定量であれば起こり得るように、わずかに負になることさえある）。シフトしたモデルに対しては、予測した $1/\sqrt{3} \approx 0.58$ の近く（通常は約 0.1 以内）に着地する。4 次モデルに対しては 0.1 の数倍程度の値となり、明らかに 0 から大きく離れている。ここでの母集団の値は 0.32 であるが、単一の標本では 0.1 以上ずれることも珍しくない。3 次のスコア $-x^3$ が裾で急激に増大するため、ごく少数の極端な点が平均を支配してしまうからである。

3 つの検定すべてを通じて変化させたのは、スコア関数だけであることに注目してほしい。データのサンプルは固定したままであり、どのモデルからも一度もサンプリングしておらず、4 次モデルの正規化定数 $Z \approx 2.56$ がコードに現れることも一度もなかった。

RBF カーネルを用いる場合、母集団における値 0 は信頼できる。スコアに関する穏やかな正則性条件のもとで、滑らかなデータ密度に対して $\mathrm{KSD}^2(P, Q) = 0$ となるのは $P = Q$ のとき、かつそのときに限られるからである (Liu et al. 2016; Chwialkowski et al. 2016)。それでも、2 成分混合分布の例は有益な教訓を与えてくれる。「理論上は非ゼロである」からといって、遠く離れた分布に対してその値が天文学的に小さくなることまで防げるわけではないのである。

KSD を $\mathbb{R}^n$ へと拡張する際に変わるのは記法だけである。g はベクトル場となり、スタイン作用

素は $s_Q^\top g + \nabla \cdot g$ となり、Equation 37.49 のスカラーの積はベクトルの内積に置き換わり、最後の項は $\sum_i \partial_{x_i} \partial_{x_i'} k$ になる。実用的には、KSD は非正規化モデルの適合度検定（goodness-of-fit test）や MCMC サンプラの診断指標として使われる。また、スタイン作用素をカーネル自身に適用すると、粒子群を Q へと押し動かす決定論的な更新方向が得られ、これはスタイン変分勾配降下法（Stein variational gradient descent）として知られるアルゴリズムである (Liu and Wang 2016)。なお、本書で扱う訓練目的関数の中に KSD を直接用いるものはない。

37.5.4 訓練目的関数と対応するダイバージェンス

前節までの 2 節にわたって分布比較の 3 つの主要な族を構築し、本節ではスコアに基づく比較を 2 つ追加した。Figure 37.12 に、これら 3 つの族とそれらが共有するダイバージェンスを整理する。この図では、3 つの族のいずれにも属さないフィッシャーダイバージェンスと KSD は省いている。KSD は積分確率指標（integral probability metric、テスト関数にわたる上限）の形をとるものの、そのテスト関数のクラスがモデルのスコアに依存し、したがって Q とともに変化するためである。

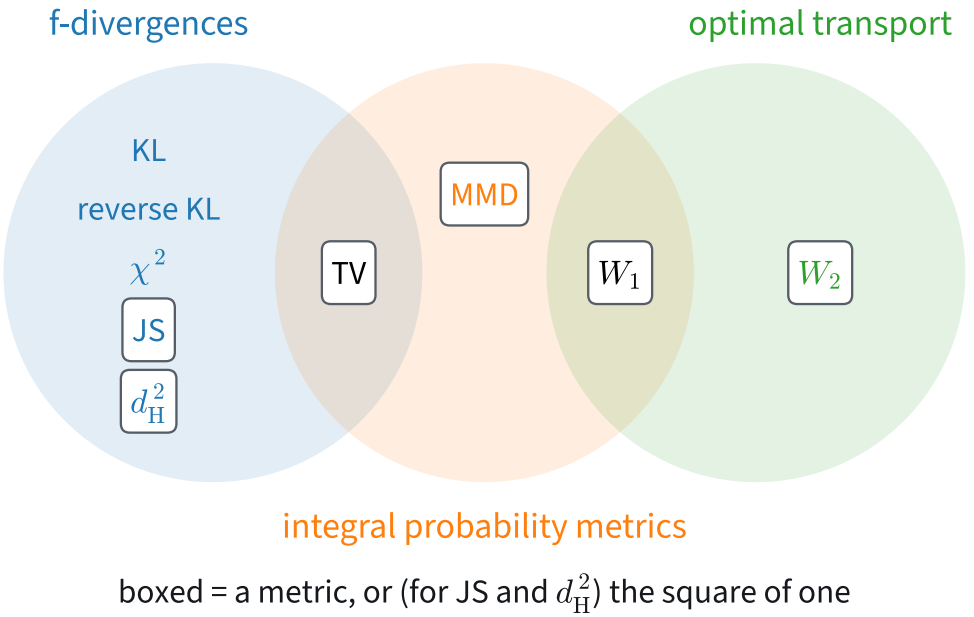


Figure 37.12: 比・テスト関数・輸送の 3 つの族と共通する要素。f ダイバージェンスは密度比の凸関数を平均し、積分確率指標はテスト関数のクラス上で平均の差を最大化し、最適輸送距離は質量を移動させる最小コストを測る。全変動は最初の 2 つの族に属し、1 次ワッサースタイン距離は後ろの 2 つの族に属する。（全変動は非ゼロの距離だけ質量を動かすたびにコスト 1 を課す輸送コストでもあるが、見やすさのため図では省いている。）枠で囲んだ名称は距離（計量）であり、JS と二乗ヘリンガーダイバージェンス d_{H}^2 は距離の二乗である。

Table 37.4 は、これらの比較手法の上に構築された訓練目的関数を整理したものである。各目的関数について、モデルとデータに求める要件と、期待値が厳密で（敵対的目的関数の場合は 37.3.4 節で実サンプルと生成サンプルを識別するよう訓練された補助ネットワークであるクリティックが最適である）理想的な極限において最小化されるダイバージェンスを示している。4 列目

では、37.3 節の冒頭で導入した診断テストケースを再訪する。すなわち、単一点に集中したデータ $P = \delta_0$ と、モデルのサンプル $Q = \delta_d$（距離 d だけ離れた点）の比較である。d を変化させても一定値にとどまる比較尺度は、そのようなモデルに対してデータへ向かう勾配を一切与えない。

Table 37.4: 必要とする情報順に並べた訓練目的関数と、平均が厳密かつクリティックが最適なときにそれらが最小化するダイバージェンス。点質量の列は δ_0 対 δ_d（$d \neq 0$）に対する値であり、$f(0)$ と $f'(\infty)$ はそれぞれ $f(u)$ の 0 における極限と $f(u)/u$ の無限大における極限を表す。また MMD の項目では長さスケール ℓ の RBF カーネルを用いている。

目的関数が必要とする情報	目的関数	ダイバージェンス	距離 d 離れた 2 つの点質量	解説箇所
データ点におけるモデル密度	最尤法（自己回帰モデル、フロー）	順方向 KL	∞	37.3.2 節
クリティックを通じた両者のサンプル	オリジナル GAN	イェンセン・シャノン	$\log 2$（定数）	37.3.4 節
クリティックを通じた両者のサンプル	f-GAN	選択した D_f	$f(0) + f'(\infty)$（定数）	37.3.4 節
両者のサンプル（固定カーネルならクリティック不要）	モーメントマッチング、2 標本検定	MMD	$\mathrm{MMD}^2 = 2 - 2e^{-d^2/(2\ell^2)}$（$\ell$ の数倍を超えると平坦）	37.4.1 節
両者のサンプル、1 リプシッツのクリティック	WGAN	W_1	d	37.4.2 節
モデルのサンプル、そこでのモデル密度、非正規化ターゲット	変分推論、変分オートエンコーダの事後分布	逆方向 KL	∞	37.3.2 節
モデルのスコア、データのサンプル	スコアマッチング、拡散モデル	フィッシャー	未定義（点質量はスコアを持たない）	37.5.1 節
モデルのスコア、データのサンプル	適合度検定、サンプラの品質評価	KSD	未定義（点質量はスコアを持たない）	37.5.3 節

この表は行がモデルの提供できる情報順に並んでいるため、上から下へ読むことで実践的な選択ガイドとして使える。モデルの正規化済み密度を評価でき、かつデータが現れ得るあらゆる場所で正であるならば、最尤法で訓練するのがよい。順方向 KL はデータしか必要とせず、不偏な標本推定値を与え、データのどのモードであっても取りこぼせばペナルティを課すからである。モデルがサンプルを生成することしかできない場合には、3 つの選択肢がある。GAN（16.1 節）のように学習されたクリティックを訓練する手法は、どの特徴量が実サンプルと生成サンプルを区別するかが事前にはわからない画像などの複雑なデータのサンプル限定モデルに対する標準的な選択肢である。今日では、その大半は拡散モデルから蒸留された 1 ステップ生成器や潜在オートエンコーダのデコーダである。学習されたクリティックの欠点は、推定されたダイバージェンスに下向きの統計的バイアスがかかり、モデルがデータから遠く離れると平坦になってしまうことである（37.3.4 節）。最適な 1 リプシッツのクリティックにはそのような平坦な領域はない。W_1 があらゆる距離で一定の傾きを保つからである。ただし、訓練されたクリティックが最適なクリティックと同じように振る舞うとは限らない。16.2.4.2 節の実験では、訓練されたロジスティッククリティックは遠く離れた生成サンプルに対してもなお傾きを持ち、通常の両側勾配ペナルティによって傾き 1 付近に保たれた W_1 クリティックは生成器をデータまで運んだあと、そのまま通り過ぎさせてしまう。固定カーネルを用いれば 2 つ目のネットワークの訓練を完全に回避でき、

任意の次元で $O(N^{-1/2})$ の推定誤差を達成できるため（37.4.1 節）、2 標本検定やサンプルの評価に理想的である。もっとも、固定カーネルの MMD は生データの次元が増すにつれて弱まり、長さスケールの数倍を超えると勾配が消失するため、通常は学習された特徴量に対して適用される。

最後の 3 行は、正規化定数が未知である設定を扱っている。変分推論（variational inference）のように、**ターゲット**密度が非正規化である一方、**モデル**がサンプルと計算可能な密度の両方を提供する場合には、単一のモードへ崩壊するリスクを伴いつつも逆方向 KL が適用できる。モデル自体がそのスコアを通じてのみ既知である場合には、拡散モデルが行うように（17.1 節）遠く離れたモードの重みを見失わないよう複数スケールにわたるノイズを加えることを条件として、（スコアマッチングを通じた）フィッシャーダイバージェンスが自然な選択肢となる。

この表は各目的関数が**どの**比較尺度を目標とするかを教えてくれるが、有限のデータからそれをどれだけ高い信頼性で推定できるかは教えてくれない。直近の 3 節で扱ったサンプルに基づく推定量（クリティックによる下界、MMD^2、および KSD^2）はいずれも推定誤差を伴い、おまけにクリティックによる下界には低く出るバイアスまである。続く 2 節では、2 つの分布の比較から 2 つの確率変数間の従属性の計測へと焦点を移す。37.6 節では KL ダイバージェンスを用いて一方の確率変数がもう一方についてどれだけの情報を持つかを定義し、37.7 節では n 組のペアからなる標本が実際にどれだけの従属性を保証できるかを問う。

37.5.5 まとめ

モデルの正規化定数が計算困難であっても、入力に関する対数密度の勾配であるスコアを通じればデータと比較できる。微分によって正規化定数が消えるからである。フィッシャーダイバージェンスはデータスコアとモデルスコアの二乗差を平均し、スタインの恒等式は 1 回の部分積分によって未知のデータスコアを消去してスコアマッチングを導く。ただし、スコアは純粋に局所的な量であるため、遠く離れたモード間で確率質量がどう分割されているかを見ることはできない。標準偏差の 10 倍離れた 2 つの混合分布に対しては、フィッシャーダイバージェンスが 4×10^{-6} まで落ち込む一方で、KL ダイバージェンスは 0.339 nat のまま保たれる。逆にスタインの恒等式をモデル側に適用すると、モデルのスコアのみを用いてサンプルを検定するカーネルスタイン不一致度が得られる。結局のところ、適切な訓練目的関数はモデルが何を提供できるか、すなわち正規化済み密度か、サンプルか、スコアかによって決まるのである。

37.5.6 演習問題

1. **フィッシャーダイバージェンスとスタインのモーメント。** $\mathcal{N}(\mu, \sigma^2)$ のスコアが $-(x-\mu)/\sigma^2$ であったことを思い出そう。
   1. フィッシャーダイバージェンス Equation 37.44 を $\mathcal{N}(\mu_1, \sigma_1^2)$ と $\mathcal{N}(\mu_2, \sigma_2^2)$ の間で閉形式として計算し、分散が等しいときに $(\mu_1 - \mu_2)^2/(2\sigma^4)$ へ帰着することを確かめよ。フィッシャーダイバージェンスは対称か。
   2. スタインの恒等式 Equation 37.46（$\mathcal{N}(0,1)$ の場合）を $g(x) = x^k$ に適用して漸化式 $\mathbb{E}[X^{k+1}] = k\,\mathbb{E}[X^{k-1}]$ を導出し、それを用いて $\mathbb{E}[X^4]$ と $\mathbb{E}[X^6]$ を計算せよ。恒等式の条件を満たす任意の密度に対して、$g(x) = 1$ は何を与えるか。
   3. 分布 P（$\mathbb{R}$ 上）がすべての次数の有限モーメントを持ち、$\mathbb{E}_P[g'(X) - Xg(X)] = 0$ をすべての多項式 g に対して満たすとする。正規分布がそのモーメントによって一意に定まるという事実を用いて、なぜ $P = \mathcal{N}(0,1)$ となるかを説明せよ。次に $P(X=1) = P(X=-1) = \tfrac{1}{2}$ とし、すべての多項式 g（次数が高々 2 のもの）に対してはこの恒等式を満たすが、$g(x) = x^3$ に対しては成り立たないことを示せ。このことは、固定された次数以下の多項式上でのみスタインの恒等式を検査する適合度検定に対して何を意味するか。
2. [code] **混合重みに対するスコアの不感性。**● $P_w = w\,\mathcal{N}(-m,1) + (1-w)\,\mathcal{N}(m,1)$ とせよ。

1. 本文では $D_{\mathrm{F}}(P_{0.7}\|P_{0.3})$ と $D_{\mathrm{KL}}(P_{0.7}\|P_{0.3})$ を $m \in \{1,3,5\}$ について計算した。この計算を $m \in \{1,\dots,6\}$ に拡張せよ。KL ダイバージェンスが $0.4\log\frac{7}{3} \approx 0.339$ に近づくことを示し、$\log D_{\mathrm{F}}$ を m^2 に対して直線当てはめすることでフィッシャーダイバージェンスの減衰率を求めよ。
2. 成分の負担率からスコアの公式 $s_w(x) = -x + m\tanh(mx - c_w)$ を導出せよ。$s_{0.7}$ と $s_{0.3}$ の差が $x=0$ において最大値 $0.8\,m$ をとることを示し、$u = mx$ をフィッシャーダイバージェンスの積分に代入することで、その積分が $C\,m\,e^{-m^2/2}$（大きな m における漸近形であり、C は重みのみに依存する）としてスケールすることを示せ。これを (a) で当てはめた減衰率と比較せよ。この結果は、スコアマッチングによって学習される混合重みについて何を意味するか (Wenliang and Kanagawa 2020)。
3. 独立な $\mathcal{N}(0,\sigma^2)$ のノイズを両方の混合分布のサンプルに加え、各成分を $\mathcal{N}(\pm m, 1+\sigma^2)$ に変える。(a) を $m=5$ において $\sigma \in \{0, 0.5, 1, 2\}$ について繰り返せ。ノイズを加えることは各ダイバージェンスにどのような影響を与えるか。強いノイズは混合重みを明らかにするが各モードの形状をぼかし、弱いノイズはその逆となる。複数のノイズレベルにわたってスコアマッチングを行うモデルが、なぜその両方を復元できるのか (Song and Ermon 2019)。
4. P_t と Q_t を、独立な $\mathcal{N}(0,t)$ のノイズを両方に加えた後の 2 つの分布とする。等しい分散 σ^2（ノイズによって σ^2+t に増加）を持つ 2 つのガウス分布に対し、本文の閉形式から $\frac{d}{dt}D_{\mathrm{KL}}(P_t\|Q_t) = -D_{\mathrm{F}}(P_t\|Q_t)$ が成り立つことを確かめよ。この恒等式は滑らかな密度のあらゆる対に対して成り立つ。2 つの混合分布に対し、有限差分を用いて $m=5$ および $t \in \{1,4,9\}$ でこれを確認し、$D_{\mathrm{KL}}(P\|Q) = \int_0^\infty D_{\mathrm{F}}(P_t\|Q_t)\,dt$ を導け。どのノイズレベルにおいてフィッシャーダイバージェンスが最大になるか。この積分は、単一のノイズレベルでのスコアマッチングとすべてのノイズレベルにわたるスコアマッチングの違いについて何を教えてくれるか (Y. Song, Durkan, et al. 2021)。
5. カーネルスタイン不一致度も同じ盲点を共有する。2 つの混合分布に対して Equation 37.51 をグリッド上の数値積分で評価し（$\ell = 1$ の RBF カーネルを用い、$m \in \{1,3,5\}$ とする）、本文のフィッシャーダイバージェンスと比較せよ（$m=5$ ではおよそ 10^{-11} になるはずである）。本文の `ksd2_population` を改変し、データ分布としての $\mathcal{N}(0,1)$ を $P_{0.7}$ に置き換えよ。積分値に寄与するためにはペアの両方の点がどこに落ちなければならず、その領域の幅は ℓ に比べてどれほどか。ここから KSD^2 が e^{-m^2} のように減衰することを導け。より小さな長さスケールを選べば改善するか。

3. **ガウスモデルに対するスコアマッチング**。データ分布 P が平均 a と分散 v を持つとし、モデルを $Q = \mathcal{N}(\mu,\sigma^2)$ ととって、$J = \mathbb{E}_P\big[\frac{1}{2}s_Q^2 + s_Q'\big]$ を Equation 37.47 のモデルに依存する部分とする。
   1. $J = \big(v + (a-\mu)^2\big)/(2\sigma^4) - 1/\sigma^2$ となることを示し、$\mu = a$ かつ $\sigma^2 = v$ において（P のそれ以外の性質によらず）最小化されることを確かめよ。これを順方向 KL ダイバージェンスを最小化するガウス分布と比較せよ（37.3.2 節）。
   2. 最小点では $J = -1/(2v)$ となるため、P からその最良のガウス近似へのフィッシャーダイバージェンスは $\frac{1}{2}\big(\mathbb{E}_P[s_P^2] - 1/v\big)$ である。P に対するスタインの恒等式を $g(x) = x - a$ に適用し、続けてコーシー・シュワルツの不等式を用いることで $\mathbb{E}_P[s_P^2] \geq 1/v$ を証明し、等号成立は P がガウス分布のとき、かつそのときに限られることを示せ。与えられた分散のもとで、平均二乗スコアが最小となる密度はどれか。
   3. モデルを $\tilde{q}_\theta(x) = \exp(\theta_1 x + \theta_2 x^2)$ と再パラメータ化せよ。J が θ の二次関数となり、したがってスコアマッチングがデータのモーメントに関する 2 本の線形方程式を解くことに帰着することを示せ。また、任意の指数分布族のモデル $\tilde{q}_\theta(x) = \exp\big(\theta^\top \phi(x)\big)$ に対しても同様の性質が成り立つことを示せ。これに対し、最尤法では代わりに何を計算する必要があるか。
4. [code] **非正規化 4 次モデル**。$\tilde{q}_\theta(x) = e^{-\theta x^4/4}$（$\theta > 0$）とせよ。

1. スコアを計算せよ。スタインの恒等式を $g(x) = x$ および $g(x) = x^3$ に適用し、正規化定数を計算することなく、このモデルのもとで $\mathbb{E}[X^4] = 1/\theta$ および $\mathbb{E}[X^6] = (3/\theta)\,\mathbb{E}[X^2]$ が成り立つことを示せ。多項式 g を用いたスタインの恒等式によって $\mathbb{E}[X^2]$ も決定できるか。
2. 本文ではスコアマッチング推定量 $\hat{\theta} = 3\,\overline{x^2}/\overline{x^6}$（上線は標本平均を表す）を導出した。正規化定数が $Z(\theta) = \theta^{-1/4} Z(1)$ のようにスケールすること、および最尤法が $\hat{\theta} = 1/\overline{x^4}$ を与えることを示せ。(a) を用いて、両方の推定量がデータの異なる関数であるにもかかわらず、モデルからの大標本上ではともに真の θ へ収束することを論じよ。
3. U が形状 $\frac{1}{4}$、尺度 1 のガンマ分布に従うとき、ランダムな符号を付けた $(4U/\theta)^{1/4}$ は密度 $\tilde{q}_\theta/Z(\theta)$ を持つ。$1{,}000$ 個の点を $\theta = 1$ でサンプリングして両方の推定値を計算し、これを $2{,}000$ 回繰り返せ。どちらの推定量の方が標準偏差が小さく、何倍異なるか。スコアマッチングは Z の計算を回避する引き換えに、何を犠牲にしているか。
4. 今度はデータが標準正規分布に従い（KSD の実験と同様）、どの 4 次モデルも厳密には一致しない場合を考える。大標本上では、(b) の 2 つの推定量はそれぞれ $3\,\mathbb{E}[X^2]/\mathbb{E}[X^6] = \frac{1}{5}$ と $1/\mathbb{E}[X^4] = \frac{1}{3}$ に収束する。両方の当てはめのスコアにおいて `ksd2_population` を評価し、さらにそれを最小化する θ を求めよ。3 つの規準が 3 つの異なる 4 次モデルを選んだことになる。なぜこれは矛盾ではなく、どのような条件のもとでなら 3 つすべてが一致するか。

5. [code] **スタインカーネルの長さスケール**。KSD の実験と同様にデータを $P = \mathcal{N}(0,1)$ とし、長さスケール ℓ の RBF カーネルを用いよ。
   1. モデル $\mathcal{N}(\mu, 1)$ に対し、Equation 37.51 を用いて $\mathrm{KSD}^2 = \mu^2\,(1 + 2/\ell^2)^{-1/2}$ となることを示せ。$\ell \to \infty$ のときの極限はいくらか。また、どの単一のテスト関数 g を選べば、その同じ極限が $\mathbb{E}_P[g' + g\,s_Q]$ の二乗として得られるか。
   2. モデル $\mathcal{N}(0, 4)$（スコアは $-x/4$）に対し、$\mathrm{KSD}^2 = \frac{9}{16}\,\ell\,(\ell^2 + 2)^{-3/2}$ となることを示せ。（ヒント：XX' を独立変数 $X + X'$ と $X - X'$ を用いて書き直せ。）この値がデータの標準偏差である $\ell = 1$ で最大となり、$\ell \to \infty$ のとき 0 に近づくことを示せ。(a) の答えを用いてこの極限を説明せよ。
   3. 両方の公式を `ksd2_population` によって $\ell \in \{0.25, 1, 4, 16\}$ で確認し、さらにスコア $-x^3$ を持つ 4 次モデルに対して対応する 4 つの値を計算せよ。いかなる単一の長さスケールも、あらゆる誤ったモデルに対して同時に最適とはなり得ないのはなぜか。

37.6 相互情報量

ここまでの節では、データ分布とモデルのように、同じ変数上の 2 つの分布を比較してきた。これに対し、深層学習における多くの問いは変数の対を扱う。画像とそのキャプション、同じ写真から切り出した 2 つの領域、あるいは文とその翻訳などである。一方の変数は、もう一方の変数についてどれだけの情報を与えてくれるのだろうか。

従属性の標準的な尺度である相関係数は、線形な関係しか検出できない。$X \sim \mathcal{N}(0,1)$ とし、$Y = X^2 + 0.3\,\epsilon$ とおく。ここで $\epsilon \sim \mathcal{N}(0,1)$ は独立なノイズである。X^3 は 0 の周りで対称であるため $\mathrm{Cov}(X, Y) = \mathbb{E}[X^3] = 0$ となり、X と Y の相関は 0 である。しかし実際には、X は Y をほぼ決定している。X がわかれば、Y は標準偏差 0.3 のノイズの分しか変動しないのに対し、単体での Y はおよそ 0 から 9 まで広がる。本節で定義する量は、この従属性を難なく捉える。実際、この対には相関 0.9 の 2 変量ガウス分布よりも多くの共有情報量が割り当てられる。

求めているのは、2 つの変数が独立であるとき、かつそのときに限り 0 となり、従属性が強まるにつれて大きくなり、さらにどちらの変数を別の座標で表し直しても変化しない尺度である。これを構築するための道具はすでに揃っている。KL ダイバージェンス $D_{\mathrm{KL}}(P\|Q)$（37.1 節）であ

る。ギブスの不等式により、この量は常に非負であり、$P = Q$ の場合にのみ 0 になる。P を対の真の同時分布とし、Q を各成分が独立であった場合に従う積分布とすれば、**相互情報量**（mutual information）$I(X;Y)$ が得られる。

相互情報量の定義自体は単純であり、その真価はそこから導かれる帰結にある。本節では 3 つの基本的な結果を確立する。第 1 に、変数を処理しても他の何らかの対象との相互情報量は増加しないため、ニューラルネットワークのどの層も、その直下の層より多くの入力情報をもつことはない。第 2 に、分類器がある誤り率を達成するには、入力がラベルについて最低限の情報量をもっていなければならない。第 3 に、任意の確率モデルのテスト損失は、その情報量に対する下界を直接保証する。続いて、Shannon の 2 つの根源的な問い（雑音通信路はどれだけの情報を運べるか、非可逆な圧縮器はどれだけ少ない情報の保持で済むか）と、圧縮から発展した第 3 の問い（学習された表現は何を保持すべきか）へと進む。本節を通じて背後の分布は既知であると仮定する。サンプルのみから相互情報量を推定するのはより難しい問題であり、37.7 節で扱う。

これまでと同様、単位には nat を用いる。ここで 1 ビット $= \ln 2 \approx 0.693$ nat である。対数はすべて自然対数とし、一般の式では log、数値を評価する際には ln と書く。また 2 値エントロピーを $H_b(p) = -p \log p - (1-p)\log(1-p)$ で表す。

37.6.1 従属性の尺度

すべての定義を具体的に理解するため、まず 1 つの 2×2 確率表の上でそれぞれの量を計算する。この確率表は、本節を通じて通し例として繰り返し登場する。

37.6.1.1 結合エントロピーと条件付きエントロピー

X を公平なビットとし、Y を X が確率 $\delta = 0.1$ で反転したコピーとする。X をノイズのある回線で送られたビット、Y を受信されたビットと読むこともできるし、Y を 2 値ラベル、X を 90% の確率でそれに一致するノイズ付き特徴量と読むこともできる。表は対称であるため、以下の議論はどちらの解釈を選んでも変わらない。行を $x \in \{0,1\}$、列を $y \in \{0,1\}$ に対応させると、同時分布は次のようになる。

$$p_{X,Y} = \begin{pmatrix} 0.45 & 0.05 \\ 0.05 & 0.45 \end{pmatrix},$$

行と列の和をとると、周辺分布 $p_X = p_Y = (\tfrac{1}{2}, \tfrac{1}{2})$ が得られる。この対を**反転ビット**（flipped bit）と呼ぶ。以下では、添字によって分布の変数名を表し、条件付き分布 $p(y \mid x)$ のように引数から分布が明らかな場合には添字を省略する。

個々のエントロピー $H(X)$ と $H(Y)$ に加え、確率変数の対には 37.1.1.3 節で導入した 2 つの自然なエントロピーがある。**結合エントロピー**（joint entropy）は、対を単一の確率変数（ここでは 4 つの結果をもつ変数）とみなしたときのエントロピーである。

$$H(X,Y) = -\mathbb{E}_{(x,y)\sim P_{X,Y}}[\log p_{X,Y}(x,y)]. \tag{37.52}$$

この表では、$H(X,Y) = -2(0.45 \ln 0.45) - 2(0.05 \ln 0.05) = 1.018$ nat となる。これは 1 つの公平なビットがもつ 0.693 nat よりは大きいが、独立な 2 つの公平なビットがもつ 1.386 nat よりは小さい。

条件付きエントロピー（conditional entropy）は、Y について X が既知となった後に残る驚きを測る量であり、条件付き分布の各行 $p(y \mid x) = p_{X,Y}(x,y)/p_X(x)$ のエントロピーの平均で与えられる。

$$H(Y \mid X) = -\mathbb{E}_{(x,y)\sim P_{X,Y}}[\log p(y \mid x)] = \sum_{x} p_X(x)\, H(Y \mid X = x). \tag{37.53}$$

反転ビットでは、2 つの行は $(0.9, 0.1)$ と $(0.1, 0.9)$ であり、それぞれのエントロピーは $H_b(0.1) = 0.325$ nat であるから、$H(Y \mid X) = 0.325$ nat となる。X を観測した後でも、Y は 0.325 nat の驚きを（元の 0.693 nat のうち）依然として保持している。

これら 3 つの数値が $1.018 = 0.693 + 0.325$ を満たしていることに注目しよう。これは連鎖律 $H(X, Y) = H(X) + H(Y \mid X)$（37.1.1.3 節）にほかならない。すなわち、対がもつ驚きは、X の驚きに、Y の追加的な驚き（X が既知となった後のもの）を加えたものに等しい。

37.6.1.2 相互情報量

条件付きエントロピーは、Y の驚きのうち X では説明できない部分を捉える。いま求めたいのは、2 つの変数が共有する部分である。もし X と Y が独立であれば、その同時分布は周辺分布の積 $P_X \otimes P_Y$ となり、すべてのセルに $p_X(x)\, p_Y(y)$ が入る。**相互情報量**とは、実際の同時分布からこの独立な積分布への KL ダイバージェンスである。

$$I(X; Y) = D_{\mathrm{KL}}(P_{X,Y} \,\|\, P_X \otimes P_Y) = \mathbb{E}_{(x,y)\sim P_{X,Y}}\left[\log \frac{p_{X,Y}(x, y)}{p_X(x)\, p_Y(y)}\right]. \tag{37.54}$$

反転ビットでは、周辺分布の積はすべてのセルに 0.25 を置く。したがって、独立な積に対する同時確率の比は、対角成分では $0.45/0.25 = 1.8$（X と Y が一致する場合）となり、非対角成分では $0.05/0.25 = 0.2$（不一致の場合）となる。この対数比を同時分布のもとで平均すると次を得る。

$$I(X; Y) = 0.9 \ln 1.8 + 0.1 \ln 0.2 = 0.529 - 0.161 = 0.368 \text{ nats}.$$

一致は独立性が予測する頻度の 1.8 倍起こり、不一致は 1/5 の頻度しか起こらない。相互情報量とは、単にこの倍率の対数の期待値である。KL ダイバージェンスは誤ったモデルを用いた際に生じる余分な符号長も測るため（37.2 節）、$I(X; Y)$ は X と Y を独立として扱う符号が 1 対あたりに浪費する nat 数でもある。17.9 節では、ネットワークが 2 つのマスクトークンを独立に予測する際に、まさにこのペナルティが現れる。

Equation 37.54 は表を 1 行ずつ読むこともできる。分子と分母を $p_X(x)$ で割ると比は $p(y \mid x)/p_Y(y)$ となり、対に関する期待値は、条件付き分布と周辺分布の間の KL ダイバージェンスを x に関して平均した形に書き直せる（対象は Y の分布である）。

$$I(X; Y) = \mathbb{E}_{x\sim P_X}\Big[D_{\mathrm{KL}}\big(p(y \mid x) \,\|\, p_Y(y)\big)\Big]. \tag{37.55}$$

この形式では、相互情報量は x を観測することで Y の分布がその周辺分布から平均してどれだけ変化するかを測る。反転ビットでは、各条件付き行 $(0.9, 0.1)$ および $(0.1, 0.9)$ は、それぞれ $0.9 \ln 1.8 + 0.1 \ln 0.2 = 0.368$ nat だけ周辺分布 $(0.5, 0.5)$ から離れている。この非対称な見方は、入力とノイズ付きエンコーダの出力のように 2 つの変数が異なる役割を担う場合に特に自然であり、37.6.4 節で再び立ち返る。

命題（相互情報量の性質）。 任意の対 (X, Y) に対して $I(X; Y) = I(Y; X) \geq 0$ が成り立ち、等号は X と Y が独立であるとき、かつそのときに限り成り立つ。関係する各エントロピーが有限であるとき、次が成り立つ。

$$I(X;Y) = H(X) + H(Y) - H(X,Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X).$$

証明。 X と Y を入れ替えても Equation 37.54 の比は変わらないため、対称性が従う。非負性と等号成立条件は、$P_{X,Y}$ と $P_X \otimes P_Y$ にギブスの不等式を適用すれば直ちに従う。すなわち KL ダイバージェンスは非負であり、すべての点で $p_{X,Y}(x,y) = p_X(x)\,p_Y(y)$ となるとき、かつそのときに限り 0 になるが、これは独立性の定義そのものである。エントロピーの恒等式を得るには、比の対数を 3 つの項に分解して期待値をとればよい。

$$I(X;Y) = \mathbb{E}\left[\log p_{X,Y}(x,y)\right] - \mathbb{E}\left[\log p_X(x)\right] - \mathbb{E}\left[\log p_Y(y)\right] = -H(X,Y) + H(X) + H(Y).$$

2 通りの順序による連鎖律 $H(X,Y) = H(Y) + H(X \mid Y) = H(X) + H(Y \mid X)$ を代入すれば、残りの 2 つの形式が得られる。$\blacksquare$

したがって、X が Y について語る情報量は、Y が X について語る情報量と正確に一致する。最後の恒等式は相互情報量を不確実性の減少として解釈するものであり、X を観測すると Y の驚きは $H(Y)$ から $H(Y \mid X)$ へと低下する（反転ビットでは 0.693 nat から 0.325 nat へ下がる）。また非負性から劣加法性の上界 $H(X,Y) \le H(X) + H(Y)$ が導かれ、等号は独立性が成り立つときにのみ成立する。すなわち、変数の対がもつ驚きは、各成分を別々に考えたときの驚きの和を決して超えない。Figure 37.13 はこれらの関係を視覚的に表したものであり、命題の各恒等式は重なり部分の面積の異なる読み取り方にすぎない。

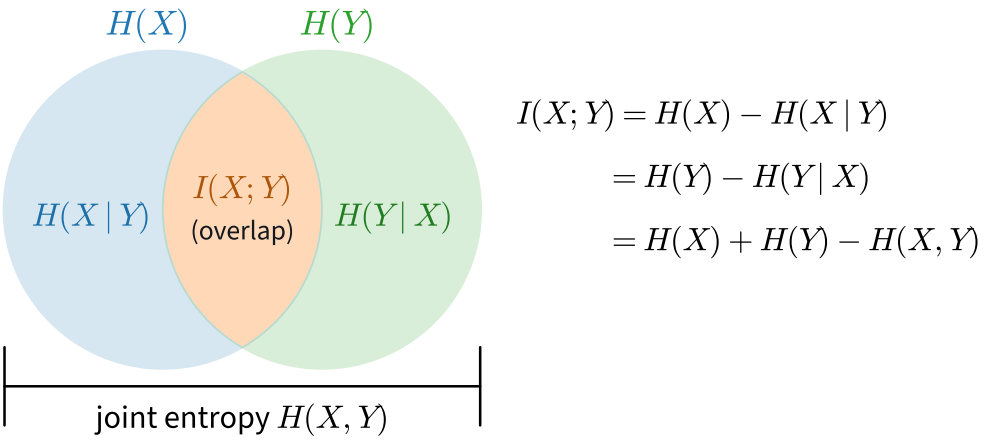


Figure 37.13: 相互情報量 $I(X;Y)$ をエントロピー $H(X)$ と $H(Y)$ の重なりとして表した図。三日月形の部分は条件付きエントロピー $H(X \mid Y)$ と $H(Y \mid X)$、和集合は結合エントロピー $H(X,Y)$ であり、$I(X;Y)$ の 3 つの等価な表現は面積を読み取ることで得られる。

以下の関数 `mutual_information` は任意の離散同時確率表に対してダイバージェンス形式を実装したものであり、本節全体で再利用する。またこのセルでは、各行のエントロピーを平均することで恒等式 $I(X;Y) = H(Y) - H(Y \mid X)$ も確認している。

```
p_xy = np.array([[0.45, 0.05],    # rows x, columns y: a fair bit X and
                 [0.05, 0.45]])   # its copy Y, flipped with probability 0.1

def entropy(p):
    """Entropy of a p.m.f. of any shape, in nats (0 log 0 = 0)."""
    p = p[p > 0]
    return float(-(p * np.log(p)).sum())

def mutual_information(p_xy):
    """I(X;Y) of a discrete joint p.m.f. (rows x, columns y), in nats."""
    p_x = p_xy.sum(axis=1, keepdims=True)
    p_y = p_xy.sum(axis=0, keepdims=True)
    m = p_xy > 0                                  # 0 log 0 = 0
    return float((p_xy[m] * np.log(p_xy[m] / (p_x * p_y)[m])).sum())

p_x, p_y = p_xy.sum(axis=1), p_xy.sum(axis=0)     # marginals from the joint
H_rows = sum(p_x[i] * entropy(p_xy[i] / p_x[i]) for i in range(2))   # H(Y|X)
I = mutual_information(p_xy)
print(f'I(X;Y) = {I:.4f} nats (KL form) = {entropy(p_y) - H_rows:.4f} '
      f'(H(Y) - H(Y|X)) = {I / np.log(2):.3f} bits')
```

```
I(X;Y) = 0.3681 nats (KL form) = 0.3681 (H(Y) - H(Y|X)) = 0.531 bits
```

どちらの式も 0.3681 nat、すなわち 0.531 ビットを返す。10 回に 1 回反転するコピーであっても、元のビットについて半ビット強の情報を伝えているのである。

37.6.1.3 ガウス対

ここまでは離散的な確率表を扱ってきたが、深層学習における大半の変数（活性化値、勾配、埋め込みなど）は連続である。連続な場合の中で、手計算で完全に求められるケースが 1 つある。2 変量ガウス分布である。これは 2 つ目の通し例として、本節後半のレート・歪み曲線および情報ボトルネック（information bottleneck）の双方の土台となる。連続密度に対しては、離散エントロピーは微分エントロピー h（37.1.4.1 節）に置き換わる。微分エントロピーは負になり得るうえ測定単位に依存するが、KL ダイバージェンスによる相互情報量の定義にはそのどちらの欠点もない。連鎖律の代数変形は離散性を用いていないため、各量が有限である限り $I(X;Y) = h(X) + h(Y) - h(X,Y) = h(Y) - h(Y \mid X)$ が成り立つ。ここで $h(Y \mid X)$ は条件付き密度 $p(y \mid x)$ の微分エントロピーを x に関して平均した量を表す。

命題（ガウス分布の相互情報量）。 **(X,Y) が分散 $\sigma_X^2, \sigma_Y^2 > 0$、相関係数 ρ（ただし $\rho^2 < 1$）の 2 変量ガウス分布に従うならば、次が成り立つ。**

$$I(X;Y) = -\tfrac{1}{2}\log\left(1-\rho^2\right). \tag{37.56}$$

証明。 $I(X;Y) = h(Y) - h(Y \mid X)$ を用いる。分散 σ^2 のガウス変数は微分エントロピー $\tfrac{1}{2}\log(2\pi e\sigma^2)$（37.1.4.3 節）をもつため、$h(Y) = \tfrac{1}{2}\log(2\pi e\sigma_Y^2)$ である。$X = x$ で条件付けると、Y は引き続きガウス分布に従い、その分散は $\sigma_Y^2(1-\rho^2)$ となって x に依存しない。これは Y の最良線形予測（X に基づく予測）を差し引いた残差の分散であり、2 変量ガウス分布ではこの残差は X と独立である（36.3.3.3 節）。したがって $h(Y \mid X) = \tfrac{1}{2}\log\left(2\pi e\sigma_Y^2(1-\rho^2)\right)$ となり、差をとれば $-\tfrac{1}{2}\log(1-\rho^2)$ が得られる。■

この結果は $1-\rho^2$、すなわち Y の分散のうち X によって説明されずに残る割合のみに依存する。

Y の単位が相殺されていることに注目されたい。これは 37.6.1.5 節で証明する座標不変性の最初の現れである。

$\rho = 0$ のとき変数は独立であり、$I(X;Y) = 0$ となる。$\rho \to \pm 1$ の極限では同時密度が直線上に退化し、$I(X;Y) \to \infty$ となる。これら両極端の間では、$\rho = 0.5$ で 0.144 nat、$\rho = 0.9$ で 0.830 nat、$\rho = 0.99$ で 1.959 nat となる。周辺分布が標準正規分布で $\rho = 0.9$ である 2 変量ガウス分布は本節を通じて繰り返し登場するため、これを $\rho = 0.9$ **のガウス対**と呼ぶ。

極限 $\rho \to \pm 1$ が示すように、相互情報量は実際に無限大になり得る。この現象は後ほど、ニューラルネットワークの決定論的な層と情報ボトルネックのエンコーダという 2 つの場面で再び登場する。連続確率変数 X に対して $Y = X$ であれば、X を知ることは Y を小数点以下無限桁まで特定することを意味する。Y が X の決定論的な関数であり、どの単一の値も正の確率をもたない場合にも同様のことが成り立つ。対照的に、Y が有限のエントロピーをもつ離散変数である場合には、相互情報量が無限大になることはあり得ない。このとき $I(X;Y) = H(Y) - H(Y \mid X) \le H(Y)$ が X によらず成り立つ（ここで $H(Y \mid X)$ は $p(\cdot \mid x)$ のエントロピーを x について平均した量である）。たとえば標準正規分布に従う X に対し、X の符号はちょうど $\ln 2$ nat の情報量をもつ。この同じ上限によって、16.1 節のイェンセン・シャノン・ダイバージェンスも $\ln 2$ 以下に抑えられる。そこで示したとおり、この量はサンプルとその生成元を識別する 2 値ラベルの間の相互情報量に等しいからである。

37.6.1.4 自己相互情報量

相互情報量はすべての結果の対にわたって平均をとるため、どの特定の対が従属性を生み出しているかは見えなくなる。しかし実際には、特定の 1 対に関心があることも多い。たとえば単語 **new** を見たとき、次の単語が **york** になる確率は偶然よりはるかに高くなるだろうか。Equation 37.54 の和の中にある被加数は、この問いに直接答えるものであり、**自己相互情報量**（pointwise mutual information）と呼ばれる。

$$\operatorname{pmi}(x, y) = \log \frac{p_{X,Y}(x, y)}{p_X(x)\, p_Y(y)}, \tag{37.57}$$

この量は、特定の対 (x, y) が実際に生じる頻度を、X と Y が独立であった場合に生じる頻度と比較する。正の pmi は 2 つの結果が引き合う（偶然の予測より高い頻度で共起する）ことを示し、負の pmi は反発し合うことを示す。そして $I(X;Y) = \mathbb{E}_{(x,y)\sim P_{X,Y}}[\operatorname{pmi}(x, y)]$ はその平均的な引力である。反転ビットでは、一致する 2 つのセルで pmi は $\ln 1.8 = +0.588$、不一致の 2 つのセルで $\ln 0.2 = -1.609$ となり、先ほど平均をとった対数と正確に一致する。

自己相互情報量は自然言語処理における古典的な道具であり (Church and Hanks 1990)、本来結びつきの強い単語対である**連語**（collocation）を、単に両方とも頻出語であるために共起する単語対から切り分ける。小さなコーパスで連続する単語対（バイグラム）を数え上げ、**new**、**machine**、**the** のそれぞれの直後に **york**、**learning**、**day** が何回現れるかを集計したとしよう。

出現回数	york	learning	day
new	38	3	9
machine	1	46	3
the	11	21	168

生の出現回数はここでは誤解を招く。「the day」が群を抜いて最多の対となっている（出現回数は 168 回であるのに対し、「new york」は 38 回）のは、単に **the** と **day** がともに高頻度語だから

である。以下のコードは、各同時頻度を 2 つの周辺単語頻度の積で割り、4 つの対に対する pmi を出力するとともに、表全体の相互情報量を求める。

```
counts = np.array([[38., 3., 9.],      # new york / new learning / new day
                   [1., 46., 3.],      # machine ...
                   [11., 21., 168.]])  # the ...
first, second = ['new', 'machine', 'the'], ['york', 'learning', 'day']

p = counts / counts.sum()
pmi = np.log(p / (p.sum(axis=1, keepdims=True) * p.sum(axis=0, keepdims=True)))
for (i, j) in [(0, 0), (1, 1), (2, 2), (2, 0)]:
    print(f'pmi({first[i]}, {second[j]}) = {pmi[i, j]:6.3f} nats '
          f'(count {counts[i, j]:.0f})')
print(f'I(first; second) = {float((p * pmi).sum()):.4f} nats')
```

```
pmi(new, york) =  1.517 nats (count 38)
pmi(machine, learning) =  1.372 nats (count 46)
pmi(the, day) =  0.336 nats (count 168)
pmi(the, york) = -1.109 nats (count 11)
I(first; second) = 0.4146 nats
```

周辺頻度で補正すると、真の連語が上位に浮上する。「new york」のスコアは 1.52 nat、「machine learning」は 1.37 であるのに対し、高頻度対「the day」は 0.34 にとどまる。一方、「the york」は -1.11 nat と負になり、独立性が予測する頻度の約 1/3 しか現れない。表全体の相互情報量である 0.41 nat は、1 語目が 2 語目を平均してどれだけ強く制約しているかを要約している。これと同じ周辺頻度の補正は、学習された単語埋め込み（19.1 節（英語版））にも再び現れる。隣接語の観測対を K 個のランダムな対から識別するよう skip-gram モデルを訓練することは、シフトされた自己相互情報量 $\mathrm{pmi}(x, y) - \log K$ を成分とする行列を暗黙のうちに行列分解することに相当する (Levy and Goldberg 2014)（演習問題 7（37.7 節））で導出する)。

37.6.1.5 相互情報量と相関

相関係数 $\rho(X, Y) = \mathrm{Cov}(X, Y)/(\sigma_X \sigma_Y)$(36.2 節)は線形な従属性しか検出できず、$\rho(X, Y) = 0$ は独立性を意味しない。対照的に、相互情報量は変数が独立であるとき、かつそのときに限り 0 になる。冒頭の例に戻ると、この違いが具体的になる。

$X \sim \mathcal{N}(0, 1)$ とし、独立なノイズを用いて $Y = X^2 + 0.3\,\epsilon$（ただし $\epsilon \sim \mathcal{N}(0, 1)$）とおく。この対の相関は $\rho = 0$ である。$I(X; Y) = h(Y) - h(Y \mid X)$ と書けば、1 次元積分を用いて相互情報量を評価できる。$X = x$ で条件付けると、Y に残るランダム性は加法的ノイズのみであり、分布を既知の定数 x^2 だけ平行移動しても微分エントロピーは変わらないため、$h(Y \mid X) = h(0.3\,\epsilon) = \tfrac{1}{2}\ln(2\pi e \cdot 0.09) = 0.215$ nat となる。$h(Y)$ を数値計算する前に、解析的な上界を求めておこう。Y の分散は $\mathrm{Var}(X^2) + 0.09 = 2.09$ であり、与えられた分散をもつすべての密度の中で微分エントロピーを最大化するのはガウス分布であるから（37.1.4.3 節）、$h(Y) \le \tfrac{1}{2}\ln(2\pi e \cdot 2.09) = 1.79$ nat、したがって $I(X; Y) \le 1.79 - 0.215 = 1.57$ nat が得られる。Y の周辺密度は x にわたる連続混合分布 $p_Y(y) = \int \mathcal{N}(x; 0, 1)\,\mathcal{N}(y; x^2, 0.09)\,dx$ であって閉形式をもたないため、数値積分によって $h(Y)$ を求める。

```
normal = lambda t, var: np.exp(-t**2 / (2 * var)) / np.sqrt(2 * np.pi * var)
xs = np.linspace(-6, 6, 6001)              # quadrature grid for X
ys = np.linspace(-2.5, 30, 6501)           # grid for Y = X^2 + 0.3 eps
dx, dy = xs[1] - xs[0], ys[1] - ys[0]
```

```
p_Y = np.concatenate([                    # p_Y(y) = int N(x;0,1) N(y;x^2,0.09) dx
    (normal(xs, 1) * normal(c[:, None] - xs**2, 0.09)).sum(axis=1) * dx
    for c in np.array_split(ys, 13)])     # 13 chunks keep memory small
h_Y = float(-(p_Y[p_Y > 0] * np.log(p_Y[p_Y > 0])).sum() * dy)
h_noise = 0.5 * np.log(2 * np.pi * np.e * 0.09)   # h(Y|X) = h(0.3 eps)
print(f'h(Y) = {h_Y:.3f}, h(Y|X) = {h_noise:.3f}, '
      f'I(X;Y) = h(Y) - h(Y|X) = {h_Y - h_noise:.3f} nats')
```

```
h(Y) = 1.356, h(Y|X) = 0.215, I(X;Y) = h(Y) - h(Y|X) = 1.141 nats
```

この対は 1.141 nat を共有している。これがガウス分布による上界 1.57 nat を大きく下回るのは、Y の密度（0 付近に集中し右に長い裾をもつ）がガウス分布から大きく外れているためである。それでいて、線形相関が 0 であるにもかかわらず、この値は 0.830 nat（$\rho = 0.9$ のガウス対）を上回っている。Figure 37.14 に 3 つのケースを並べて比較する。

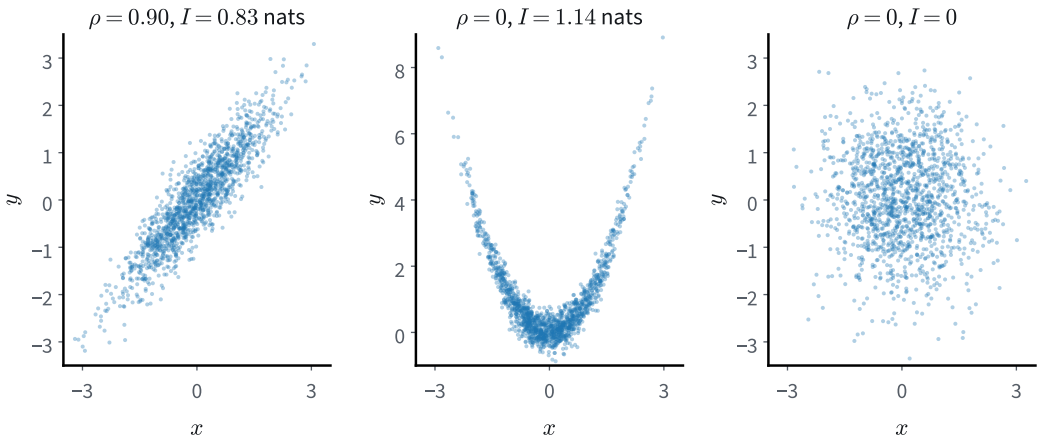


Figure 37.14: 相関と相互情報量による 3 つの対の比較。(a) 相関 0.9 のガウス対。(b) $Y = X^2 + 0.3\epsilon$。X が Y をほぼ決定しているにもかかわらず対称性により相関は 0 になる（(b) では縦軸のスケールが異なることに注意）。(c) 独立な対。相関は (b) と (c) を区別できないが、相互情報量は (b) を (a) より上位に置く。

相関と相互情報量には、もう 1 つ根本的な違いがある。座標変換に対する振る舞いである。気温を摂氏ではなく華氏で測ったり、価格を対数スケールで記録したりしても、その変数が他の対象についてどれだけの情報をもつかはまったく変わらない。したがって、原理に基づいた従属性の尺度はこうした変換のもとで不変であるべきである。

命題（再パラメータ化不変性）。ϕ と ψ をそれぞれ X の値と Y の値に対する可逆写像とする。このとき $I(\phi(X); \psi(Y)) = I(X; Y)$ が成り立つ。

証明。離散変数の場合、可逆写像は結果のラベルを付け替えるだけであり、$(\phi(X), \psi(Y))$ の同時分布は付け替え後の結果に対して (X, Y) の同時分布と同じ確率を割り当てるため、Equation 37.54 の和は変化しない。連続変数の場合、ここでは仕組みを示すために滑らかな場合を証明し、滑らかさの仮定は 37.6.2.2 節で取り除く。ϕ と ψ が連続微分可能かつ連続微分可能な逆写像をもつとし、$(x, y) \mapsto (\phi(x), \psi(y))$ によって変数変換する。変換後の対の $(\phi(x), \psi(y))$ における同時密度は $p_{X,Y}(x, y)\,|\det D\phi(x)|^{-1} |\det D\psi(y)|^{-1}$ となる一方、2 つの周辺密度にも因子 $|\det D\phi(x)|^{-1}$ と $|\det D\psi(y)|^{-1}$ が掛かる。これらのヤコビ行列の行列式因子は密度比の中で

相殺されるため、Equation 37.54 の被積分関数は対応する点で同じ値をとり、その期待値も変わらない。$x \mapsto x^3$ における原点のような確率 0 の例外点の集合は積分に寄与しない。■

微分エントロピーがこの不変性（再パラメータ化不変性（reparameterization invariance））をもたないのは、ヤコビ行列の行列式因子を打ち消す相手が存在しないためであり、それゆえ h は単位に依存する。また相関は（符号を除けば）アフィン変換のもとでしか保存されない。たとえば可逆写像 $x \mapsto x^3$ を $\rho = 0.9$ のガウス対の第 1 座標に適用しても、相互情報量は 0.830 nat のままであるが、相関は 0.9 から $3\rho/\sqrt{15} = 0.697$ へと低下する（$\mathbb{E}[X^3 Y] = 3\rho$ かつ $\mathrm{Var}(X^3) = 15$ が標準正規分布に従う X に対して成り立つため）。相互情報量が問うているのは、2 つの変数が**何らかの**座標系において互いについての情報をもつかであって、たまたま直線上に並んでいるかではない。

37.6.2 データ処理と予測

ここで、冒頭で予告した 3 つの結果を導出しよう。最初の 2 つは根本的な限界を表すものである。すなわち、データを処理しても情報は生み出せず、正確な予測には最低限の情報量が不可欠である。どちらも相互情報量の連鎖律という単一の恒等式から従う。3 つ目の結果は、任意の確率モデルの交差エントロピー損失を、入力が保持する情報量の下界へと直接変換するものである。

37.6.2.1 条件付き相互情報量と連鎖律

反転ビットの例をもう 1 段階拡張してみよう。Z が、Y をさらに独立に確率 0.2 で反転させることで生成されるとする。これは公平なビットを直列につながれた 2 本のノイズ回線で送る状況、あるいはラベルに続いてノイズ付き特徴量とそのノイズ付きコピーが得られる状況に相当する（Figure 37.15）。直列の 2 回反転は単一の反転として振る舞う。ビットが反転して届くのは 2 本の回線のうちちょうど一方だけが反転させるとき、かつそのときに限り、その確率は $0.1 \cdot 0.8 + 0.9 \cdot 0.2 = 0.26$ である。したがって対 (X, Z) は $\delta = 0.26$ の反転ビットであり、$I(X;Z) = \ln 2 - H_b(0.26) = 0.120$ nat となる。2 本目のノイズ回線によって、0.368 nat あった Y の情報量の 2/3 が失われたことになる。

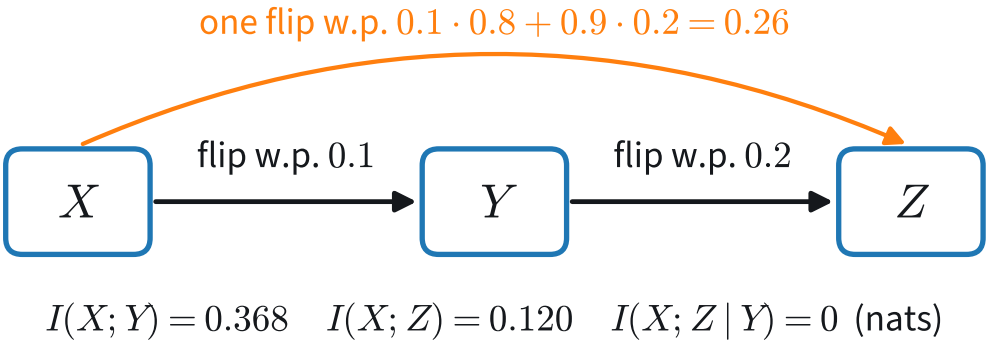


Figure 37.15: 直列の 2 回反転。公平なビット X が確率 0.1 で反転して Y となり、Y が確率 0.2 で反転して Z となる。2 回の反転は合成すると確率 0.26 の単一の反転になる。2 回目の反転で情報量の 2/3 が失われ、$I(X;Z \mid Y) = 0$ は、Z が X について、Y がすでに知っている以上の情報を何ももたないことを表す。

変数が 3 つになると、新たな問いが生じる。X と Y は、Z がすでに既知であるときにどれだけの情報を共有するだろうか。エントロピーに条件付き版があるのと同様に、相互情報量にも条件

付き相互情報量（conditional mutual information）がある。

$$I(X;Y \mid Z) = H(X \mid Z) - H(X \mid Y, Z), \tag{37.58}$$

ここで $H(X \mid Y, Z)$ は、X の対 (Y, Z) が与えられたもとでの条件付きエントロピーである。各条件付きエントロピーを期待値として書いて対数をまとめると、ダイバージェンス形式 $I(X;Y \mid Z) = \mathbb{E}_z\big[D_{\mathrm{KL}}(P_{X,Y|z} \,\|\, P_{X|z} \otimes P_{Y|z})\big]$ が得られる。これは各 z の値に対する条件付き分布の上で通常の相互情報量を計算し、平均をとったものにほかならない。KL ダイバージェンスの期待値であるため、$I(X;Y \mid Z)$ は常に非負であり、X と Y が Z を与えたもとで条件付き独立であるとき、かつそのときに限り 0 となる。

この定義により、対 (Y, Z) が X について保持する情報量を 2 つの部分に分解できる。$H(X \mid Z)$ を $I(X;Y,Z) = H(X) - H(X \mid Y, Z)$ の中で引いて足すと、次式を得る。

$$I(X;Y,Z) = \underbrace{H(X) - H(X \mid Z)}_{I(X;Z)} + \underbrace{H(X \mid Z) - H(X \mid Y, Z)}_{I(X;Y \mid Z)}.$$

この恒等式は**相互情報量の連鎖律**（chain rule for mutual information）であり、各項を KL 形式で書けば完全な一般性のもとで成り立つ。言葉で述べれば、(Y, Z) が X について教えてくれる情報量は、Z が教えてくれる情報量に、Y が Z の既知のもとでつけ加える情報量を足したものに等しい。まったく同様に、先に Y で条件付けてもよい。直列の 2 回反転では、2 通りの順序によって同じ合計が 2 通りに分解される。Z は Y と独立なノイズから生成されるため、Y が既知となれば、Z は X についてそれ以上の情報を何も加えない。すなわち $I(X;Z \mid Y) = 0$ であり、Y を先に置くと $I(X;Y,Z) = I(X;Y) + 0 = 0.368$ nat となる。一方、Z を先に置くと、同じ 0.368 nat が $I(X;Z) + I(X;Y \mid Z) = 0.120 + I(X;Y \mid Z)$ と分割され、$I(X;Y \mid Z) = 0.248$ nat が残る。ここでは、Z で条件付けることで、Y が X について明かす情報量が減少した。Z がすでにその情報の一部を保持しているからである。しかし次項で見るように、条件付けによって相互情報量が**増加**することもある。

37.6.2.2 データ処理不等式

ニューラルネットワークでは、各隠れ層はその直下の層から計算され、さらにその層も入力から計算される。このような処理によって、前の層に含まれていなかった入力についての情報を新たに生み出すことはできるだろうか。2 本のノイズ回線では答えは否であったが、その理由はビットという特殊な設定とは無関係である。

3 つの確率変数 X, Y, Z が**マルコフ連鎖**（$X \to Y \to Z$ と書く）をなすとは、Z が (X, Y) に Y を通じてのみ依存すること、すなわち $p(z \mid x, y) = p(z \mid y)$ が成り立つことである。これは逐次処理の構造をそのまま捉えている。まず Y が X から生成され（観測値、特徴マップ、あるいは隠れ層）、続いて Z が（場合によっては新たなランダム性を伴って）Y のみから計算される。同値な言い方をすれば、X と Z は Y が与えられたもとで条件付き独立であり、$I(X;Z \mid Y) = 0$ が成り立つ。

命題（データ処理不等式）。 *$X \to Y \to Z$ がマルコフ連鎖であり、関係する各エントロピーが有限であるならば、次が成り立つ。*

$$I(X;Z) \le I(X;Y),$$

等号が成り立つのは $I(X;Y \mid Z) = 0$ のとき、すなわち $X \to Z \to Y$ もマルコフ連鎖であるとき、かつそのときに限る (Cover and Thomas 1999)。

証明。 $I(X;Y,Z)$ を連鎖律により両方の順序で展開する。

$$I(X;Z) + I(X;Y \mid Z) = I(X;Y,Z) = I(X;Y) + I(X;Z \mid Y).$$

マルコフ性より最後の項は 0 となり、条件付き相互情報量の非負性から $I(X;Z) = I(X;Y) - I(X;Y \mid Z) \le I(X;Y)$ が得られる。等号は $I(X;Y \mid Z) = 0$ のとき、かつそのときに限り成り立つ。$\blacksquare$

決定論的であれランダム化されていようと、Y に対するいかなる処理も、X について保持する情報量を増やすことはできない。直列の 2 回反転では、この不等式は単に $0.120 \le 0.368$ と表される。

この不等式（データ処理不等式（data-processing inequality））はマルコフ連鎖の構造に厳密に依拠しており、条件付け一般の性質ではないことに注意されたい。マルコフ連鎖の外では、条件付けによって本来存在しなかった従属性が生じることがある。X と Y を独立な公平なビットとし、$Z = X \oplus Y$ をその排他的論理和とする。周辺分布では X と Y が共有する情報量は 0 であるが、いったん Z が明かされると、Y は X を完全に決定する。以下のコードは、直列の 2 回反転とこの XOR の 3 つ組の双方について相互情報量を計算する。

```
def cond_mi(p_abc):
    """I(A;B|C) of a joint p.m.f. indexed [a, b, c], in nats."""
    p_ac, p_bc, p_c = p_abc.sum(1), p_abc.sum(0), p_abc.sum((0, 1))
    m = p_abc > 0                                  # 0 log 0 = 0
    ratio = ((p_abc * p_c[None, None, :])[m]
             / (p_ac[:, None, :] * p_bc[None, :, :])[m])
    return float((p_abc[m] * np.log(ratio)).sum())

flip = lambda d: np.array([[1 - d, d], [d, 1 - d]])   # p(output | input)
p_xyz = 0.5 * flip(0.1)[:, :, None] * flip(0.2)[None, :, :]   # X -> Y -> Z
I_xy, I_xz = mutual_information(p_xyz.sum(2)), mutual_information(p_xyz.sum(1))
I_xz_y, I_xy_z = cond_mi(p_xyz.transpose(0, 2, 1)), cond_mi(p_xyz)
print(f'I(X;Y) = {I_xy:.4f}, I(X;Z) = {I_xz:.4f}, '
      f'I(X;Z|Y) = {I_xz_y:.4f}, I(X;Y|Z) = {I_xy_z:.4f}')
print(f'I(X;Y,Z) by the chain rule: {I_xz + I_xy_z:.4f} = {I_xy + I_xz_y:.4f}')
p_xor = np.zeros((2, 2, 2))
for a, b in np.ndindex(2, 2):
    p_xor[a, b, a ^ b] = 0.25          # independent fair bits, Z = X xor Y
print(f'XOR: I(X;Y) = {mutual_information(p_xor.sum(2)):.4f}, '
      f'I(X;Y|Z) = {cond_mi(p_xor):.4f} nats')
```

```
I(X;Y) = 0.3681, I(X;Z) = 0.1201, I(X;Z|Y) = 0.0000, I(X;Y|Z) = 0.2480
I(X;Y,Z) by the chain rule: 0.3681 = 0.3681
XOR: I(X;Y) = 0.0000, I(X;Y|Z) = 0.6931 nats
```

2 回反転では、$I(X;Z \mid Y) = 0$ によってマルコフ性が数値的に確認でき、$I(X;Y,Z)$ の 2 通りの連鎖律分解はいずれも 0.3681 nat を与える。排他的論理和では、Z で条件付けることにより、X と Y の相互情報量は 0 から 1 ビット分の全情報量である $\ln 2 = 0.693$ nat へと跳ね上がる。

処理によって情報がまったく失われないのはどのような場合だろうか。命題より、等号が成り立つのは $X \to Z \to Y$ もマルコフ連鎖であるとき、かつそのときに限る。すなわち Z が、Y から得られる X についての情報をすべて保持している場合である。可逆変換 $Z = g(Y)$ は明らかにこれを達成する。Y を Z から復元できるからである。しかし写像が Y の一部を捨てたとしても、X についての情報を失わないことがある。捨てる部分が X と無関係でありさえすればよい。た

とえばコイン投げの系列における表の合計回数は、その系列がコインの偏りについて語る情報をすべて保存している。出た順序はコインの偏りについて何の情報ももたないからである。この性質をもつ任意の要約量 $T(Y)$（すなわち $p(x \mid y)$ が y に $T(y)$ を通じてのみ依存し、したがって $I(X;T(Y)) = I(X;Y)$ となるもの）を**十分統計量**（sufficient statistic）と呼ぶ。指数分布族（36.4 節）の十分統計量はまさにこの種の要約量であり、そこでは Y がデータの役割を、X がパラメータ θ（$I(\theta;Y)$ が適切に定義されるよう事前分布を備えたもの）の役割を果たす。

データ処理不等式そのものには、エントロピーの有限性も確率密度の存在も必要ない。相互情報量は KL ダイバージェンスであり、ランダムであれ決定論的であれデータのいかなる写像も KL ダイバージェンスを増加させないため（可算集合については 37.3.3.3 節で証明しており、完全な一般性のもとで成り立つ）、演習問題 4 ではこの事実からデータ処理不等式の一般的な証明を導く。さらに、この不等式を可逆写像とその逆写像に 1 回ずつ適用すれば、滑らかさを仮定することなく 37.6.1.5 節の再パラメータ化不変性も得られる。

深層ネットワークにデータ処理不等式を適用する際には、誤解が生じやすい。ネットワークの各層はマルコフ連鎖 $X \to Z_1 \to Z_2 \to \cdots$ をなすため、$I(X;Z_\ell)$ は深さ ℓ とともに増加することはない。ここから本節の終わりまでは、Y を連鎖の中間変数ではなく予測対象のラベルを表すものとする。ネットワークがラベルについて知り得ることはすべて入力を経由しなければならないため、$Y \to X \to Z_1 \to Z_2 \to \cdots$ もマルコフ連鎖となり、$I(Y;Z_\ell)$ もまた ℓ とともに増加し得ない。いかなるアーキテクチャも入力に存在しなかったラベル情報を無から生み出すことはできず、前の層が捨てた情報を後の層が取り戻すこともできない。

これは、深さが役に立たないという意味では**ない**。隠れ層はラベルについての情報を増やすのではなく、単純な最終層で読み取れるように既存の情報を再編成するのである。ラベル Y を公平なビットとし、入力 $X \in \mathbb{R}^2$ は半径 1 の円周上の一様分布（$Y=0$ のとき）または半径 2 の円周上の一様分布（$Y=1$ のとき）に従うとする。入力はラベルを完全に決定するため $I(Y;X) = \ln 2$ であるが、2 つの同心円を分離できる直線は存在しない。最良の線形分類器は決定境界を内側の円に接するように置き、内側の円全体と、直線の外側に位置する外側の円の 1/3 を正しく分類して、正解率 $2/3$ を達成する。ここで単一の非線形特徴量 $Z = \|X\|^2$ を適用すると、その値は 1 と 4 の 2 通りしかとらない。両者の間に単純な閾値を設ければすべての点を完璧に分類できるが、$I(Y;Z) = \ln 2$ は $I(Y;X)$ より大きくなったわけではない。この変換は情報を一切増やさなかったが、すべての情報を線形に読み取れる形へと変えたのである。演習問題 2 では、制限された分類器の族が抽出できる分だけを数える利用可能情報量（usable information）の概念を定式化する。

入力側の連鎖 $X \to Z_1 \to Z_2 \to \cdots$ については、最後に 1 つ注意点がある。決定論的な層では $H(Z_\ell \mid X) = 0$ である。表現が離散化または量子化されていれば $I(X;Z_\ell)$ は単にエントロピー $H(Z_\ell)$ に帰着し、連続であれば $I(X;Z_\ell)$ は通常無限大になる（37.6.1.3 節）。したがって、層が入力についてどれだけ保持しているかの尺度として $I(X;Z_\ell)$ が意味をもつのは、層がノイズを注入するか離散出力を生成する場合に限られる。

37.6.2.3 ファノの不等式

正確な予測には相互情報量が不可欠である。誤り率 5% を 1,000 クラス分類で達成する分類器は、ラベルのエントロピーの大半を含む入力を受け取っていなければならず、ファノの不等式はその必要量を正確に教えてくれる。

命題（ファノの不等式）。Y が k 個の要素からなる集合に値をとるとし、$\hat{Y} = g(X)$ を Y に対する X から計算された同じ集合上の任意の推定値とし、$P_e = P(\hat{Y} \neq Y)$ をその誤り確率とする。このとき次が成り立つ。

$$H_b(P_e) + P_e \log(k-1) \geq H(Y \mid X) = H(Y) - I(X;Y). \tag{37.59}$$

左辺には、推定値 $\hat{Y}$ を修復するコストとしての明快な符号化解釈がある。まず $H_b(P_e)$ nat を使って $\hat{Y}$ が誤っているかどうかを指定し、誤っている場合（確率 P_e）には高々 $\log(k-1)$ nat を使って残りの $k-1$ 個のラベルのどれが正解かを指定する。推定値と修復情報の両方をもっていれば Y が正確にわかるため、修復のコストは Y について $\hat{Y}$ が残す不確実性を下回ることはない。そしてデータ処理不等式により、その不確実性は少なくとも X が残す不確実性以上である。

証明では、誤りが生じたかどうかを表す指示変数というただ 1 つの道具を使って、この直観を厳密な不等式へと仕立てる。残りは連鎖律だけである。

証明。 $W = \mathbf{1}\{\hat{Y} \neq Y\}$ を誤りの指示変数とする。推定値が与えられたもとでの誤りの有無と真のラベルの双方に関する残りの不確実性 $H(W, Y \mid \hat{Y})$ を 2 通りに展開する。第 1 に、W は $(Y, \hat{Y})$ によって完全に定まるためエントロピーを増やさず、$H(W, Y \mid \hat{Y}) = H(Y \mid \hat{Y})$ となる。第 2 に、連鎖律より $H(W, Y \mid \hat{Y}) = H(W \mid \hat{Y}) + H(Y \mid W, \hat{Y})$ が得られ、各項を個別に上から抑える。

- $H(W) - H(W \mid \hat{Y}) = I(W; \hat{Y}) \geq 0$ であるから、条件付けによってエントロピーは増加しない。したがって $H(W \mid \hat{Y}) \leq H(W) = H_b(P_e)$ である。
- $W = 0$ のときラベルは $\hat{Y}$ に等しく、不確実性は 0 である。$W = 1$ のとき（確率 P_e で生じる）、ラベルは $k-1$ 個ある $\hat{Y}$ 以外の値のいずれかであるため、そのエントロピーは高々 $\log(k-1)$ である。W について平均をとると $H(Y \mid W, \hat{Y}) \leq P_e \log(k-1)$ が得られる。

両方の上界を合わせると $H(Y \mid \hat{Y}) \leq H_b(P_e) + P_e \log(k-1)$ が得られる。最後に、$\hat{Y}$ は X のみから計算されるため 3 つ組 $Y \to X \to \hat{Y}$ はマルコフ連鎖をなし、データ処理不等式より $I(Y; \hat{Y}) \leq I(Y; X)$、すなわち同値な形として $H(Y \mid \hat{Y}) \geq H(Y \mid X)$ が従う。$\blacksquare$

第 2 段階における残りのラベル数 $k-1$ は、$\hat{Y}$ が同じラベル集合の中に値をとることに依拠している点に注意しよう。もし分類器に「棄権」という回答を許せば、$\log k$ で $\log(k-1)$ を置き換えなければならず、上の不等式は成り立たなくなる。実際、公平なビット Y が X と独立であり、分類器が常に棄権するとすれば、$P_e = 1$ となって左辺は $H_b(1) + \ln 1 = 0$ となり、$H(Y \mid X) = \ln 2$ を下回ってしまう。

最も単純な場合では、ファノの不等式は等号で成り立つ。反転ビットにおいて推定値を $\hat{Y} = X$ とすると、$P_e = 0.1$ かつ $k = 2$ であるから、左辺は $H_b(0.1) + 0.1 \ln 1 = 0.325$ となり、$H(Y \mid X)$ と厳密に一致する。

アーキテクチャや訓練手順がいかなるものであろうと、ファノの限界を超える分類器は存在しない。均衡した 1,000 クラスでは $H(Y) = \ln 1000 = 6.91$ nat であり、Equation 37.59 を $I(X; Y)$ について整理すると、各目標誤り率を達成するのに必要な最小の相互情報量がわかる。

```
H_b = lambda p: -p * np.log(p) - (1 - p) * np.log(1 - p)   # binary entropy
k = 1000
for p_e in (0.01, 0.05, 0.2, 0.5):
    need = np.log(k) - H_b(p_e) - p_e * np.log(k - 1)
    print(f'P_e = {p_e:4.2f}: I(X;Y) >= {need:.3f} of log k = {np.log(k):.3f} nats')
```

```
P_e = 0.01: I(X;Y) >= 6.783 of log k = 6.908 nats
P_e = 0.05: I(X;Y) >= 6.364 of log k = 6.908 nats
P_e = 0.20: I(X;Y) >= 5.026 of log k = 6.908 nats
P_e = 0.50: I(X;Y) >= 2.761 of log k = 6.908 nats
```

誤り率 5% に到達するには少なくとも 6.36 nat（ラベルエントロピー 6.91 nat の 92%）が必要であり、半分の確率で誤る分類器でさえ 2.76 nat、すなわち全体の 2/5 を必要とする。

37.6.2.4 損失による保証

ファノの不等式は分類誤り率を相互情報量の下界へと変換する。しかし言語モデルはハードな誤り率では評価されない。最尤法で訓練されるほぼすべてのモデルと同様、交差エントロピー損失を報告するのである。この損失からも相互情報量の下界が得られ、その証明はわずか 3 行で済む。

命題（損失は情報量を保証する）。 **$q(y \mid x)$ を入力に対するラベルの任意のモデルとし、交差エントロピー損失を $L = \mathbb{E}[-\log q(Y \mid X)]$（$P_{X,Y}$ のもとでの期待値）とする。このとき次が成り立つ。**

$$I(X;Y) \;\geq\; H(Y) - L, \tag{37.60}$$

さらに、均衡した k クラスに対しては $I(X;Y) \geq \log k - L$ が成り立つ。

証明。 定義より $I(X;Y) = H(Y) - H(Y \mid X)$ である。各 x に対し、$q(\cdot \mid x)$ の $p(\cdot \mid x)$ のもとでの交差エントロピーは、$p(\cdot \mid x)$ の真の条件付きエントロピーを $D_{\mathrm{KL}}(p(\cdot \mid x) \,\|\, q(\cdot \mid x)) \geq 0$ だけ上回る（37.1 節）。x について平均をとれば $H(Y \mid X) \leq L$ が得られ、これを $H(Y)$ から引けば証明が完了する。$\blacksquare$

ホールドアウトしたテスト損失は L を推定するため、任意の分類器や言語モデルのテスト損失は、入力がもつ情報量の下界を直接保証する。たとえば均衡したラベルに対する 10 クラス分類器のテスト損失が 0.3 nat であれば、入力は少なくとも $\ln 10 - 0.3 = 2.0$ nat（ラベルの 2.30 nat のうち）を保持していることが保証される。この下界はモデルの較正が不十分であったり訓練が不出来であったりしても成り立ち、弱いモデルは単に小さな下界を保証するにすぎない。これは分類器をデコーダとして用いた Barber and Agakov (2003) の変分下界であり、37.7.3 節ではこれをより広い相互情報量推定量の族へと発展させる。

この保証がタイトになるのは、$q(y \mid x)$ が真の条件付き分布 $p(y \mid x)$ に一致するとき、かつそのときに限る。反転ビットにおいて $\hat{Y} = X$ を確信度 c で予測するモデルの期待損失は $L = -0.9 \log c - 0.1 \log(1 - c)$ であり、以下ではこれを 4 つの確信度について評価する。

```
for c in (0.6, 0.8, 0.9, 0.99):
    L = -(0.9 * np.log(c) + 0.1 * np.log(1 - c))   # cross-entropy of q
    print(f'confidence {c:4.2f}: loss {L:.4f}, '
          f'certifies I(X;Y) >= {np.log(2) - L:.4f} nats')
```

```
confidence 0.60: loss 0.5514, certifies I(X;Y) >= 0.1418 nats
confidence 0.80: loss 0.3618, certifies I(X;Y) >= 0.3314 nats
confidence 0.90: loss 0.3251, certifies I(X;Y) >= 0.3681 nats
confidence 0.99: loss 0.4696, certifies I(X;Y) >= 0.2236 nats
```

較正されたモデル（$c = 0.9$）だけが全情報量である 0.368 nat を保証する。過信したモデル（$c = 0.99$）は誤るたびに大きなペナルティを支払い 0.22 nat しか保証できず、自信不足のモデル（$c = 0.6$）は 0.14 nat にとどまる。しかし 4 つのモデルはすべて同一のハード予測を出力し 10% の確率で誤るため、誤り率しか見ないファノの不等式はどのモデルに対しても 0.368 nat すべてを保証する。交差エントロピー損失は予測確率まで見ているため、較正のずれにペナルティを科すのである。どちらの結果も同じ条件付きエントロピー $H(Y \mid X)$ を上から抑え（ファノの不等式は誤り率により、損失による保証は損失により）、それぞれの上界を $I(X;Y)$ の下界へと変換している。実は両者の結びつきは見た目以上に深い。演習問題 6 で示すように、ファノの下

界自体がある特定のモデルに対する損失による保証にほかならず、自身がいつ不確かであるかを把握している適切に較正された分類器は、それより真に多くの情報量を保証できる。

37.6.3 通信と圧縮

Shannon は 2 つの根源的な工学的問いを立てたが、その答えは相互情報量の最大値と最小値に帰着する。雑音通信路はどれだけの情報を確実に運べるか、そして非可逆な圧縮器はどれだけ少ない情報の保持で済むか、という問いである。表現学習に関心がある読者は初読時には通信路容量（channel capacity）を流し読みしても構わないが、レート・歪み理論（rate–distortion theory）は本節の最後のパートが直接その上に構築されるため不可欠である。

37.6.3.1 通信路容量

各ビットを独立に確率 δ で反転させる通信回線を、どれだけの情報が生き残れるだろうか。無記憶**通信路**（channel）は、各入力記号に独立に適用される条件付き分布 $p(y \mid x)$ によって規定され、送信される記号 1 つを通信路の 1 回の使用と数える。通し例の反転ビットは、入力 X と出力 Y をもつまさにそのような通信路である。通信路のノイズは固定されているが、送信者は入力分布 $p(x)$ を自由に選ぶことができる。通信路の**容量**（capacity）とは、入力分布のあらゆる選び方にわたって達成可能な最大の相互情報量であり、1 回の使用あたりの nat 数で測られる。

$$C = \max_{p(x)} I(X;Y). \tag{37.61}$$

ほとんど 0 ばかりを送る送信者は、そもそも入力自体のエントロピーが小さいため、通信路を十分に活用できていない。シャノンの通信路符号化定理は、この量に操作的な意味を与える。符号の**レート**を通信路の 1 回の使用あたりに伝達するメッセージの nat 数と定義したとき、Shannon は、記号の長いブロックをまとめて符号化することで、C 未満の任意のレートにおいて誤り確率をいくらでも小さく抑えた高信頼な通信が可能である一方、C を超えるいかなるレートでも高信頼な通信は不可能であることを示した (Shannon 1948; Cover and Thomas 1999)。この定理は証明なしで認めることとする。

2 元対称通信路（binary symmetric channel）として知られるビット反転通信路では、容量は 1 行で計算できる。$I(X;Y) = H(Y) - H(Y \mid X)$ と書く。$p(y \mid x)$ のどの行も順序を除けば $(1-\delta, \delta)$ であるから、入力分布によらず $H(Y \mid X) = H_b(\delta)$ となる。一方、$H(Y) \le \ln 2$ であり、等号は Y が一様分布であるとき、かつそのときに限り成り立つ。$P(Y=1) = \delta + P(X=1)(1-2\delta)$ より、$\delta \neq \tfrac{1}{2}$ である限り、出力が一様になるのは入力が一様であるとき、かつそのときに限る。したがって次式を得る。

$$C = \ln 2 - H_b(\delta) \quad \text{nats per use}, \tag{37.62}$$

この最大値は一様な入力によって達成される。$\delta = 0.1$ では、（入力が最初から一様であった）反転ビットの 0.368 nat が復元され、直列の 2 本の回線（$\delta = 0.26$）では容量は 0.120 nat へと低下する。$\delta = \tfrac{1}{2}$ では出力は入力と独立になり、容量は 0 になる。$\delta = 1$ では容量は全情報量である $\ln 2$ nat に戻る。すべてのビットが反転していると知っている受信者は単にそれを反転し直せばよいからである。ノイズが情報を破壊するのは、受信者がそれを予測できない範囲に限られる。以下のコードでは、まず $\delta = 0.1$ の通信路に対して入力分布を動かし、続いて一様入力のもとで反転確率 δ を動かす。

```
inputs = (0.1, 0.3, 0.5, 0.7, 0.9)
I_in = [mutual_information(np.array([[1 - a], [a]]) * flip(0.1)) for a in inputs]
print('P(X=1):', '  '.join(f'{a:6.2f}' for a in inputs))
print('I(X;Y):', '  '.join(f'{v:6.4f}' for v in I_in))
deltas = (0.0, 0.1, 0.26, 0.5, 1.0)
C = [np.log(2) - entropy(np.array([d, 1 - d])) for d in deltas]
print('delta: ', '  '.join(f'{d:6.2f}' for d in deltas))
print('C:     ', '  '.join(f'{c:6.4f}' for c in C))
```

```
P(X=1):   0.10    0.30    0.50    0.70    0.90
I(X;Y): 0.1463  0.3160  0.3681  0.3160  0.1463
delta:    0.00    0.10    0.26    0.50    1.00
C:      0.6931  0.3681  0.1201  0.0000  0.6931
```

1 を 10% の割合でしか送らない入力は、そもそも $H_b(0.1) = 0.325$ nat しかもっておらず、そのうち通信路を通過できるのは 0.146 nat、すなわち一様入力が達成する量の 2/5 にすぎない。Figure 37.16 の左パネルに、容量を δ の関数として描く。ニューラルネットワークの内部にも独自の通信路が存在する。ドロップアウトは特徴量を確率 α で消去し、それ以外の場合には通過させるが、演習問題 7 では 2 値の特徴量がこの通信路を通して高々 $(1-\alpha)\ln 2$ nat しか送れないことを示す。

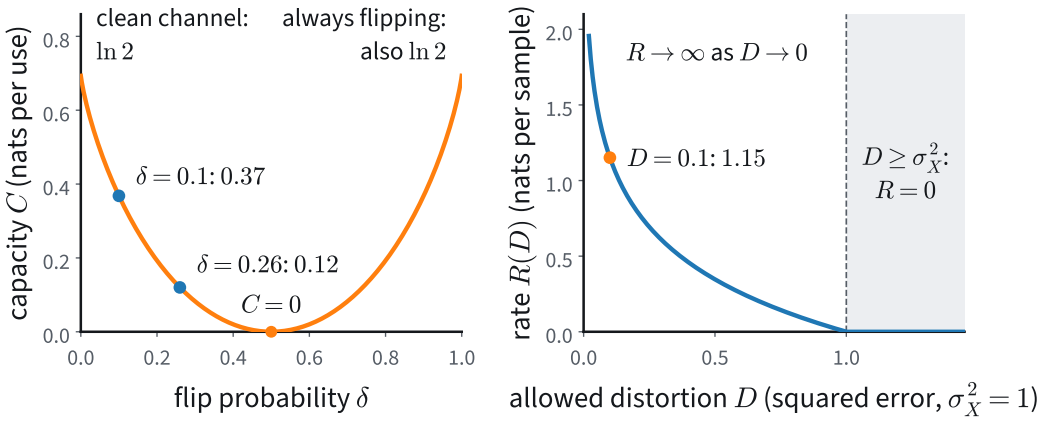


Figure 37.16: nat 単位で表した Shannon の 2 つの操作曲線。左：2 元対称通信路の容量 $C = \ln 2 - H_b(\delta)$。反転ビット（$\delta = 0.1$）と直列の 2 回反転（$\delta = 0.26$）に印をつけてある。確率 $1/2$ で反転する通信路は何も運べない一方、常に反転する通信路は $\ln 2$ すべてを運ぶ。右：分散 $\sigma_X^2 = 1$ の情報源に対するガウス分布のレート・歪み関数。レートは $D \to 0$ で発散し、D が情報源の分散に達すると 0 になる。

37.6.3.2 レート・歪み

可逆圧縮には 1 記号あたり $H(X)$ nat が必要であり（37.2 節）、連続情報源を厳密に可逆符号化しようとすれば無限に多くの nat が必要になる。しかし実用上は、近似的な再構成で十分なことがほとんどである。画像コーデックはピクセルをわずかに変化させてもよいし、学習された表現は無関係な細部を意図的に捨ててもよい。非負の**歪み**（distortion）尺度 $d(x, \hat{x}) \geq 0$ を選んで $\hat{x}$ が x をどれだけうまく再構成しているかを測り、平均歪みを高々 D まで許容するとしよう。1 サンプルあたりどれだけ少ない nat 数で済むだろうか。**レート・歪み関数**（rate–distortion

function）は、歪みの許容範囲内に収めるためにエンコーダ $p(\hat{x} \mid x)$ が保持しなければならない最小の相互情報量としてこの問いに答える。

$$R(D) = \inf_{p(\hat{x}|x):\ \mathbb{E}[d(X,\hat{X})] \le D} I(X;\hat{X}). \tag{37.63}$$

許容歪みを厳しくするほど、より多くの情報量が求められる。シャノンのレート・歪み定理は通信路符号化定理と対をなしており、ここでのレートは符号が情報源の 1 サンプルあたりに費やす nat 数を測る。すなわち、$R(D)$ を上回る任意のレートで動作するブロック符号は平均歪み D を達成できる一方、$R(D)$ 未満のいかなる符号もそれを達成できない (Shannon 1959)。

実際には、ハードな歪み制約をペナルティに置き換え、$I(X;\hat{X}) + \lambda\,\mathbb{E}[d(X,\hat{X})]$ を乗数 $\lambda > 0$ のもとですべてのエンコーダにわたって最小化するほうが扱いやすいことが多い。各エンコーダは歪み・レート平面において曲線 $R(D)$ 上またはその上方の 1 点に対応し、ペナルティ付き目的関数は傾き $-\lambda$ の直線上で一定となる。$R(D)$ は凸関数であるため、最小値は傾き $-\lambda$ の直線が曲線に下から接する点で達成される。λ を大きくするほど、より高いレートでより小さい歪みが選ばれる。37.6.4 節の情報ボトルネックは、まさにこのラグランジアン形式で定式化される。

命題（ガウス情報源のレート・歪み）。 **情報源 $X \sim \mathcal{N}(0,\sigma_X^2)$ と二乗誤差 $d(x,\hat{x}) = (x-\hat{x})^2$ のもとで、次が成り立つ。**

$$R(D) = \frac{1}{2}\log\frac{\sigma_X^2}{D} \quad (0 < D < \sigma_X^2), \qquad R(D) = 0 \quad (D \ge \sigma_X^2). \tag{37.64}$$

この式を歪みについて解くと $D = \sigma_X^2 e^{-2R}$ となる。レートを 1 ビット（$\ln 2$ nat）増やすごとに、二乗誤差は 1/4 に減少する。

証明。 主たるステップは下界の導出であり、37.1.4.3 節のただ 1 つの事実に依拠する。すなわち、分散 s^2 をもつすべての密度の中で微分エントロピーを最大化するのはガウス分布であり、その最大値 $\frac{1}{2}\log(2\pi e s^2)$ は s^2 とともに単調に増加する。$\mathbb{E}[(X-\hat{X})^2] \le D$ を満たす任意のエンコーダをとると、次が成り立つ。

$$\begin{aligned} I(X;\hat{X}) &= h(X) - h(X \mid \hat{X}) = h(X) - h(X-\hat{X} \mid \hat{X}) \\ &\ge h(X) - h(X-\hat{X}) \\ &\ge \tfrac{1}{2}\log(2\pi e\sigma_X^2) - \tfrac{1}{2}\log(2\pi e D) = \tfrac{1}{2}\log\frac{\sigma_X^2}{D}. \end{aligned}$$

2 つ目の等号は、既知の値 $\hat{X}$ を引いても条件付き密度が平行移動するだけでエントロピーは変わらないことから従う。1 つ目の不等号は条件付けがエントロピーを増加させないという事実を用い、2 つ目の不等号は再構成誤差 $X-\hat{X}$ の分散が高々 $\mathbb{E}[(X-\hat{X})^2] \le D$ であることからガウス分布の最大エントロピー上界を適用している。$D \ge \sigma_X^2$ のときは、何も送信せず常に平均 0 を出力する定数エンコーダがすでに許容歪みを満たすため、$R(D) = 0$ となる。

残るは $D < \sigma_X^2$ に対してこの下界を達成するエンコーダの構成である。$\sigma_X^2 = 1$ とおき、独立なガウスノイズを加えて $Z = X + \sigma\epsilon$（ただし $\epsilon \sim \mathcal{N}(0,1)$、$\sigma > 0$）とする。このとき $\mathrm{Cov}(X,Z) = 1$ かつ $\mathrm{Var}(Z) = 1 + \sigma^2$ であるから、二乗相関は $\rho_{XZ}^2 = 1/(1+\sigma^2)$ となる。Equation 37.56 を適用すると、このエンコーダが保持する相互情報量は次で与えられる。

$$I(X; X+\sigma\epsilon) = \tfrac{1}{2}\log\left(1+\sigma^{-2}\right). \tag{37.65}$$

ここで X を条件付き平均 $\hat{X} = \mathbb{E}[X \mid Z] = Z/(1+\sigma^2)$ によって再構成する。これは X の Z に対する傾き $\mathrm{Cov}(X,Z)/\mathrm{Var}(Z)$ の最適な線形回帰である。その平均二乗誤差は条件付き分散 $D = 1 - 1/(1+\sigma^2) = \sigma^2/(1+\sigma^2)$ であり、開区間 $(0,1)$ 全体を σ の変化に伴って動く。$\hat{X}$ は Z の可逆な定数倍であるため、$I(X;\hat{X}) = \frac{1}{2}\log(1+\sigma^{-2}) = \frac{1}{2}\log(1/D)$ が成り立つ。一般の分散 σ_X^2 に対しては、単に X と $\hat{X}$ を定数倍すればよい。■

この証明で用いたエンコーダ（独立なガウスノイズで汚された信号）は、信号対雑音比（signal-to-noise ratio）σ^{-2} の**ガウス通信路**（Gaussian channel）として知られ、Equation 37.65 はその容量である（演習問題 8）。拡散モデル（第 17 章）の順方向ノイズ付加過程はまさにこの通信路にほかならない。データ分布がいかなるものであっても、信号対雑音比 s までノイズを加えられた座標は、元のクリーンな値について高々 $\frac{1}{2}\log(1+s)$ nat しか保持しない。以下のコードは、100 万個のサンプル上でこのエンコーダを $\sigma^2 = 1/9$、1、3（それぞれ $D = 0.1$、0.5、0.75 に対応）においてシミュレーションし、二乗誤差を検証する。

```
rng = np.random.default_rng()
x = rng.standard_normal(1_000_000)
for s2 in (1 / 9, 1.0, 3.0):                                # sigma^2
    z = x + np.sqrt(s2) * rng.standard_normal(x.size)       # encoder
    x_hat = z / (1 + s2)                                    # E[X | Z]
    D = s2 / (1 + s2)
    mse = np.mean((x - x_hat)**2)
    print(f'sigma^2 = {s2:5.3f}: D = {D:.3f}, MSE {mse:.3f}, '
          f'I(X;Z) = R(D) = {0.5 * np.log(1 + 1 / s2):.3f} nats')
```

```
sigma^2 = 0.111: D = 0.100, MSE 0.100, I(X;Z) = R(D) = 1.151 nats
sigma^2 = 1.000: D = 0.500, MSE 0.498, I(X;Z) = R(D) = 0.347 nats
sigma^2 = 3.000: D = 0.750, MSE 0.749, I(X;Z) = R(D) = 0.144 nats
```

経験二乗誤差は小数第 3 位の数単位以内で D と一致する。$D = 0.1$ の歪みには 1 サンプルあたり 1.15 nat、$D = 0.5$ には 0.35 nat（半ビット）、$D = 0.75$ には 0.14 nat のコストがかかる。

このノイズ付きエンコーダが $R(D)$ を達成しているのは相互情報量としてであり、まだ実際のファイルサイズとしてではないことに留意されたい。実用的な符号はサンプルを離散的なビット列へと写像しなければならず、サンプルの長いブロックをまとめて符号化することで初めてレート・歪み曲線に漸近する。1 度に 1 サンプルずつ量子化するいかなるスカラー符号も、曲線より真に上にとどまる。最も単純な例は X の符号のみを保持するもので、$\ln 2 = 0.693$ nat を費やして $\pm\sqrt{2/\pi}$（$|X|$ の平均）を再構成し、二乗誤差 $1 - 2/\pi = 0.363$ を生じるが、レート・歪み曲線がその歪みに対して要求するのはわずか 0.506 nat である。

Figure 37.16 の右パネルに曲線全体を示す。$D \to 0$ につれて必要レートは限りなく上昇し（連続変数の厳密なコピーがもつ無限の情報量を反映している）、0 まで落ちるのは D が σ_X^2 に達したときである。$D = 0$ 付近でのこの急激な立ち上がりは、現代の画像生成器がモデル化の前に画像を圧縮する理由を説明している。知覚できないほど微細なピクセルの細部がビットの大半を消費してしまうため、17.7 節ではそうした細部をオートエンコーダに委ねるのである。

37.6.4 情報ボトルネック

優れた表現は、ラベルの予測に役立つものを残し、それ以外を捨てるべきである。一見すると、何かを捨てることは逆効果に思えるかもしれない。データ処理不等式によれば、情報を捨ててもベイズ最適な予測器が改善することは決してないからである。しかし実際には、無関係な細部には現実のコストが伴う。犬の写真には、照明や芝生、さらにはフレーム内のどこにたまたま犬が立っているかまでもが符号化されている。そうした細かな違いをすべて保存するには余分な nat が

かかり、有限枚の画像で訓練された分類器はそれらをラベルへの真の手がかりと容易に取り違えてしまう。情報ボトルネックは、このトレードオフを 2 つの相互情報量によって定式化する。それはラベル自体が歪みを定義するレート・歪み問題にほかならない。

37.6.4.1 ボトルネック目的関数

Z を入力 X から確率的エンコーダ $p(z \mid x)$ によって生成される表現とし、Y を目的ラベルとする。Z が連続であるとき、エンコーダは確率的でなければならない。決定論的な写像では通常 $I(X;Z) = \infty$ となってしまうからである（37.6.1.3 節）。**情報ボトルネック** (Tishby et al. 1999) は、表現 Z が X の情報をできる限り削ぎ落としつつ、Y についての情報をできる限り多く保持することを目指す。

$$\min_{p(z|x)} \; I(X;Z) - \beta\, I(Y;Z). \tag{37.66}$$

ここで $I(X;Z)$ はレート（表現が入力について保持する情報量）であり、$I(Y;Z)$（重み $\beta > 0$ を掛ける）は表現がラベルについて保持する情報量である。エンコーダは X しか参照しないため、変数はマルコフ連鎖 $Y \to X \to Z$ をなし、データ処理不等式により $I(Y;Z) \le \min\{I(X;Y),\, I(X;Z)\}$ という制約が課される。表現は入力以上にラベルを知ることは決してできず、また入力情報を保持せずにうまく予測することもできない。

乗数 β がこのトレードオフを制御しており、その両極端は容易に理解できる。$\beta \le 1$ である限り、何も保持しないことが最適である。実際、上の不等式から $I(X;Z) - \beta I(Y;Z) \ge (1-\beta)\, I(X;Z) \ge 0$ であり、X を完全に無視する定数エンコーダが $I(X;Z) = I(Y;Z) = 0$ を達成する。定数エンコーダが最適になるとき、ボトルネックは**崩壊**（collapse）するという。逆に β が大きくなるとラベルの項が支配的になり、エンコーダは Y の予測に役立つあらゆる情報を保持するようになる。その極限では $I(Y;Z)$ は上限 $I(X;Y)$ に近づき、表現は**最小十分統計量**（minimal sufficient statistic）、すなわち X の要約のうち Y を X 自身と同等の精度で予測できる最も粗いものへと近づく。もちろん、あらゆるニューラルネットワークがそうであるようにエンコーダがパラメトリックな族に制限されている場合には、その上限の手前で頭打ちになることがある。

命題（レート・歪み問題としてのボトルネック）。 *$Y \to X \to Z$ がマルコフ連鎖であるならば、次が成り立つ。*

$$I(Y;Z) = I(X;Y) - \mathbb{E}_{(x,z)\sim P_{X,Z}}\Big[D_{\mathrm{KL}}\big(p(y \mid x) \,\|\, p(y \mid z)\big) \Big]. \tag{37.67}$$

証明。 $I(Y;X,Z)$ を連鎖律により両方の順序で展開すると $I(Y;Z) + I(Y;X \mid Z) = I(Y;X) + I(Y;Z \mid X)$ となる。マルコフ性より $I(Y;Z \mid X) = 0$ である。残った条件付き相互情報量を KL 形式で書くと $I(Y;X \mid Z) = \mathbb{E}_{(x,z)}\big[D_{\mathrm{KL}}(p(y \mid x,z) \,\|\, p(y \mid z))\big]$ となり、マルコフ性により $p(y \mid x,z)$ は $p(y \mid x)$ に簡約される。$\blacksquare$

Equation 37.67 を目的関数に代入すると、Equation 37.66 は $I(X;Z) + \beta\, \mathbb{E}[d(X,Z)] - \beta I(X;Y)$ へと書き換わる（ただし歪みは $d(x,z) = D_{\mathrm{KL}}(p(y \mid x) \,\|\, p(y \mid z))$ である）。最後の項 $\beta I(X;Y)$ はエンコーダに依存しないため、情報ボトルネックはまさにレート・歪み問題のラグランジアン形式である。設計者が手作業で歪み尺度を作る代わりに、ラベルがそれを自動的に与えてくれる。符号 z は、x に対する忠実な表現であるのは、それが Y を x と同じ精度で予測するときなのである。ただし通常のレート・歪み問題とは異なり、この歪みは $p(y \mid z)$ を通じてエンコーダ自身に依存するため、最適化は格段に難しくなる。

37.6.4.2 ガウス情報ボトルネック

ある特別な設定では、情報ボトルネックのすべての量を閉形式で解くことができる (Chechik et al. 2005)。(X, Y) を $\rho = 0.9$ のガウス対（$I(X;Y) = 0.830$ nat）とし、37.6.3.2 節の加法的ノイズエンコーダを用いる。

$$Z = X + \sigma\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1),$$

このエンコーダはノイズのスケール σ のみでパラメータ化される。レート $I(X;Z)$ はすでに Equation 37.65 で与えられている。ラベルについては $\mathrm{Cov}(Y,Z) = \mathrm{Cov}(Y,X) = \rho$ かつ $\mathrm{Var}(Z) = 1+\sigma^2$ であるから、$\rho_{YZ}^2 = \rho^2/(1+\sigma^2)$ となる。Equation 37.56 を適用すると、両方の相互情報量が求まる。

$$I(X;Z) = \tfrac{1}{2}\log\left(1+\sigma^{-2}\right), \qquad I(Y;Z) = -\tfrac{1}{2}\log\left(1 - \frac{\rho^2}{1+\sigma^2}\right). \tag{37.68}$$

σ が ∞ から 0 へ減少するにつれて、表現は X について何も知らない状態から X を完全に知る状態へと滑らかに移行し、ラベル情報量は 0 からデータ処理不等式の上限である 0.830 nat まで上昇する。

このトレードオフを直接可視化するため、レートを $u = I(X;Z)$ と書く（R はレート・歪み関数 $R(D)$ のために取っておく）。Equation 37.68 を逆に解くと $\sigma^{-2} = e^{2u} - 1$、ゆえに $1/(1+\sigma^2) = 1 - e^{-2u}$ となり、ラベル情報量をレートの関数として直接表せる。

$$I(Y;Z) = -\tfrac{1}{2}\log\left(1 - \rho^2(1-e^{-2u})\right), \qquad \frac{d\,I(Y;Z)}{du} = \frac{\rho^2}{\rho^2 + (1-\rho^2)\,e^{2u}}.$$

導関数は ρ^2（$u=0$ における値）から 0 に向かって単調に減少するため、この曲線は狭義凹関数である。原点を傾き $\rho^2 = 0.81$ で出発するため、最初の 1 nat の入力情報から得られるラベル情報は高々 0.81 nat であり、その後に続く 1 nat ごとの増分は $I(Y;Z)$ が $I(X;Y)$ に近づくにつれて逓減していく。Chechik et al. (2005) は、ガウス対に対して線形・非線形を問わずいかなるエンコーダもこの曲線を上回れないことを証明しており、これが厳密な最適フロンティアとなる（この結果は証明なしで用いる）。任意の β を選ぶと、ラグランジアンの最小点はこのフロンティア上のただ 1 つの**動作点**（operating point）を選び出す。

命題（動作点と崩壊）。$\rho^2 < 1$ のガウス情報ボトルネックにおいて、ラグランジアン $u - \beta\,I(Y;Z)$ は u に関して凸であり、その導関数は $1-\beta\rho^2$（$u=0$ における値）となる。$\beta \le 1/\rho^2$ では最小点は $u = 0$（定数エンコーダ）である。$\beta > 1/\rho^2$ では、最小点とそのラベル情報量は次で与えられる。

$$u^{*} = \frac{1}{2}\log\frac{\rho^2(\beta-1)}{1-\rho^2}, \qquad I(Y;Z)^{*} = \frac{1}{2}\log\frac{\beta-1}{\beta(1-\rho^2)}. \tag{37.69}$$

証明。 $I(Y;Z)$ は u に関して凹であるため、ラグランジアン $u - \beta I(Y;Z)$ は凸であり、導関数は $1 - \beta\rho^2/(\rho^2 + (1-\rho^2)e^{2u})$ である。$u=0$ ではこの導関数は $1-\beta\rho^2$ に等しい。$\beta\rho^2 \le 1$ のときは常に、導関数は $u=0$ で非負かつ狭義単調増加であるため、$u \ge 0$ における最小値は $u=0$ で達成される。そうでない場合、導関数を 0 とおくと $\rho^2 + (1-\rho^2)e^{2u} = \beta\rho^2$、すなわち $e^{2u} = \rho^2(\beta-1)/(1-\rho^2)$ が得られる。$\rho^2 e^{-2u} = (1-\rho^2)/(\beta-1)$ を $I(Y;Z) = -\tfrac{1}{2}\log(1-\rho^2+\rho^2 e^{-2u})$ に代入すれば 2 つ目の式が得られる。■

先ほどは一般のデータ処理不等式の議論から $\beta \le 1$ での崩壊が保証された。しかし $\rho = 0.9$ のガウス対では、実際には崩壊が $\beta \le 1/\rho^2 = 1.235$ に至るまで続く。この閾値以下では、レートを 1 nat 増やすごとに目的関数に 1 が加わる一方で減少量は高々 $\beta\rho^2 \le 1$ にとどまる。フロンティアの傾きがその初期値 ρ^2 を決して超えないからである。閾値を超えると、β の増加に伴って動作点はフロンティアに沿って上限へと滑り上がっていく。$\beta = 2$ ではレート $u^* = \tfrac{1}{2}\ln(0.81/0.19) = 0.725$ nat とラベル情報量 0.484 nat が得られ、$\beta = 16$ ではそれぞれ 2.079 nat と 0.798 nat に達する。Figure 37.17 は、横軸にレート $I(X;Z)$、縦軸にラベル情報量 $I(Y;Z)$ をとった**情報平面**（information plane）上に、このフロンティアと各動作点を描いたものである。

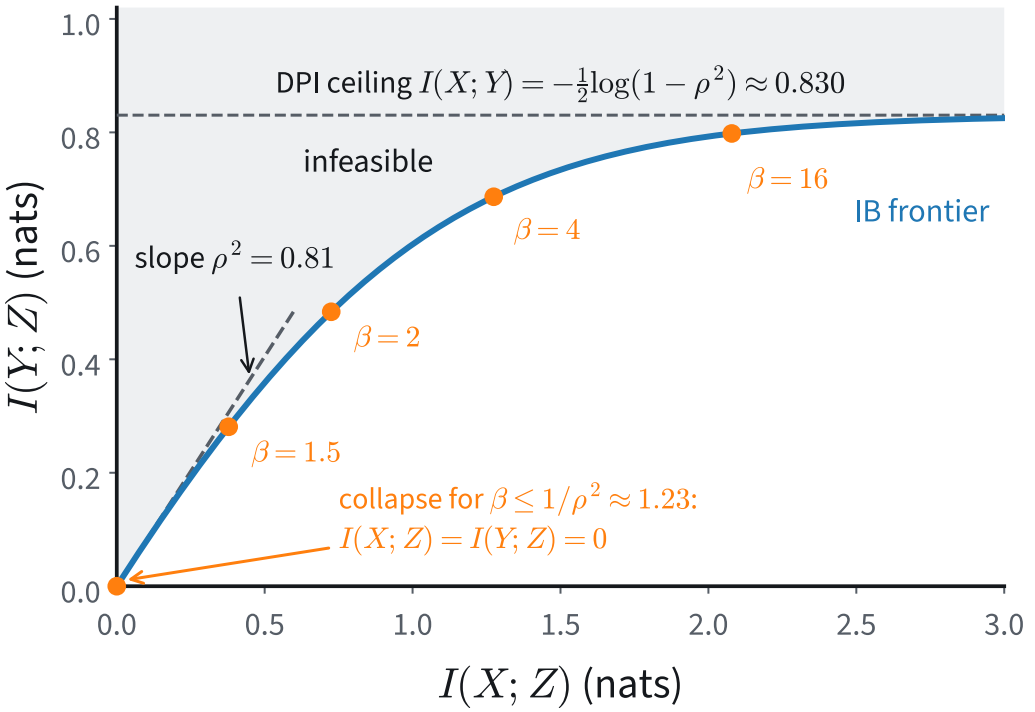


Figure 37.17: $\rho = 0.9$ のガウス情報ボトルネックの情報平面。凹フロンティアは達成可能な最大の $I(Y;Z)$ を各レート $I(X;Z)$ に対してたどり、その上方の網掛け領域は実行不可能である。破線の上限はデータ処理不等式の限界 $I(X;Y) = -\tfrac{1}{2}\log(1-\rho^2) \approx 0.830$ nat を表す。フロンティアは原点を傾き ρ^2 で出発し、乗数は傾きが $1/\beta$ に等しくなる点を選択する。$\beta \le 1/\rho^2 \approx 1.23$ ではその点は原点へと崩壊する。

これほど鋭い相転移は数値的にも確かめておく価値がある。命題によれば、最適エンコーダは $\beta = 1.2$ では何も保持しないが、$\beta = 1.25$ では情報を保持するはずである。以下では閉形式解を用いずに、4,001 個のノイズスケールからなるグリッド上でラグランジアンを直接最小化する。

```
rho = 0.9
sigma = np.logspace(-2, 2, 4001)                  # encoder Z = X + sigma eps
I_xz = 0.5 * np.log(1 + sigma**-2)
I_yz = -0.5 * np.log(1 - rho**2 / (1 + sigma**2))
for beta in (1.2, 1.25, 1.5, 2, 4, 16):
```

```
    i = np.argmin(I_xz - beta * I_yz)            # Lagrangian on the grid
    u = (0.5 * np.log(rho**2 * (beta - 1) / (1 - rho**2))
         if beta * rho**2 > 1 else 0.0)          # closed-form optimum
    print(f'beta = {beta:5.2f}: I(X;Z) grid {I_xz[i]:.3f}, closed form {u:.3f};'
          f' I(Y;Z) = {I_yz[i]:.3f}')
```

```
beta =  1.20: I(X;Z) grid 0.000, closed form 0.000; I(Y;Z) = 0.000
beta =  1.25: I(X;Z) grid 0.032, closed form 0.032; I(Y;Z) = 0.026
beta =  1.50: I(X;Z) grid 0.379, closed form 0.378; I(Y;Z) = 0.281
beta =  2.00: I(X;Z) grid 0.725, closed form 0.725; I(Y;Z) = 0.484
beta =  4.00: I(X;Z) grid 1.275, closed form 1.274; I(Y;Z) = 0.687
beta = 16.00: I(X;Z) grid 2.078, closed form 2.079; I(Y;Z) = 0.798
```

グリッド探索と閉形式解は 0.001 nat の精度で一致し、命題が予測したとおりの位置に崩壊の閾値があることが確認できる。$\beta = 1.2$ では定数エンコーダが最適であるのに対し、$\beta = 1.25$ ではエンコーダはすでに 0.032 nat を保持している。ノイズスケール σ をもつまったく同じこのエンコーダが、二乗誤差に対するレート・歪み曲線 $R(D)$ も $D = \sigma^2/(1+\sigma^2)$ において達成していることに注目しよう。レート・歪み理論はエンコーダが X をどれだけうまく再構成するかで評価するのに対し、情報ボトルネックは Y をどれだけうまく予測するかで評価するのである。

37.6.4.3 変分情報ボトルネック

ガウス対に対しては、ボトルネック目的関数に含まれる 2 つの相互情報量をどちらも解析的に計算できた。しかし実データでは真の同時分布が未知であるため、どちらも計算できない。観測できるのはサンプル (x, y) と、設計者が選んだエンコーダ $p(z \mid x)$ のみである。**変分情報ボトルネック**（variational information bottleneck）は、各相互情報量をサンプルから推定できる扱いやすい上下界に置き換えることで、この目的関数を訓練可能にする (Alemi et al. 2017)。

ラベルの項については、必要な下界がすでに手元にある。デコーダ $q(y \mid z)$（表現からラベルを予測する分類器）を導入し、その交差エントロピー損失を $L_Z = \mathbb{E}[-\log q(Y \mid Z)]$ とする。損失による保証 Equation 37.60 を対 (Z, Y) に適用すれば、直ちに $I(Y;Z) \geq H(Y) - L_Z$ が得られる。

一方、レート $I(X;Z)$ は**最小化**したい量であるため、今度は上界が必要になる。レートを条件付き形式 Equation 37.55、すなわち $I(X;Z) = \mathbb{E}_x\big[D_{\mathrm{KL}}(p(z \mid x) \,\|\, p(z))\big]$ で書こう。エンコーダ $p(z \mid x)$ は既知であるが、集約された周辺分布 $p(z)$ は $p(z \mid x)$ を未知のデータ分布で平均したものであるため未知である。そこで $p(z)$ を、標準正規分布などの任意に選んだ固定の事前密度 $r(z)$ に置き換える。恒等式 $\log \frac{p(z|x)}{r(z)} = \log \frac{p(z|x)}{p(z)} + \log \frac{p(z)}{r(z)}$ を用いて x と z について期待値をとると、次式を得る。

$$\mathbb{E}_x\Big[D_{\mathrm{KL}}(p(z \mid x) \,\|\, r(z))\Big] = I(X;Z) + D_{\mathrm{KL}}(p(z) \,\|\, r(z)) \;\geq\; I(X;Z). \tag{37.70}$$

選んだ事前分布 r と真の周辺分布 $p(z)$ の間にずれがあると、レートの上界は膨らむ方向にしか動かない。演習問題 9 では、ガウスエンコーダに対してこのギャップを明示的に計算する。両方の上下界を組み合わせると、Equation 37.70 の左辺にある KL 項に βL_Z を加えたものが、ボトルネック目的関数 Equation 37.66 の上界を（定数 $\beta H(Y)$ の差を除いて）与えることがわかる。どちらの項も訓練例に関する単純な期待値であるため、ニューラルネットワークは確率的勾配降下法（stochastic gradient descent）によってこの上界を直接最小化できる。

変分オートエンコーダ（variational autoencoder）を訓練したことがあれば、Equation 37.70 の左辺にはすぐ見覚えがあるだろう。変分オートエンコーダ（17.7 節）とは、入力自身をラベルと

したまったく同じ構成にほかならない。すなわち、エンコーダ $p(z \mid x)$、固定の事前分布 $r(z)$、そして別のクラスラベルを予測する代わりに x を z から再構成するデコーダからなる。その訓練損失は、再構成誤差に重み付きの KL ペナルティを加えたものである。本節の視点から見れば、その KL ペナルティはレートの上界であり、潜在コードが入力について蓄えられる nat 数を制限する一方、再構成誤差が歪みの役割を担っている。異なるのはトレードオフの重みを置く位置だけである。Equation 37.66 が保持される情報量に β を掛けるのに対し、変分オートエンコーダは KL レート項に重みを置くため、あちらでの大きな KL 重みはこちらでの小さな β に対応する。

デコーダの交差エントロピー損失と事前分布への KL ダイバージェンスは、本節の中でサンプルのみから評価できる唯一の量であり、一方の相互情報量には下界を、もう一方には上界を与えた。それ以外に計算してきた数値はすべて同時分布が既知であることに頼っていたが、実際の画像やテキストでは同時分布など決して手に入らない。37.7 節では、サンプルのみから相互情報量についてどこまで推定できるかを問う。

37.6.5 まとめ

相互情報量は、同時分布とその周辺分布の積との間の KL ダイバージェンス $I(X;Y) = D_{\mathrm{KL}}(P_{X,Y} \,\|\, P_X \otimes P_Y)$ であり、X と Y が独立であるとき、かつそのときに限り 0 となり、いずれの変数の可逆な再パラメータ化のもとでも不変である。線形相関とは異なり非線形な従属性を検出でき、相関が 0 であるにもかかわらず 1.14 nat を X と $Y = X^2 + 0.3\epsilon$ に割り当てる。データ処理不等式により、逐次処理は情報を生み出すことはできないが、ニューラルネットワークの層は分類器が読み取りやすいように既存の情報を並べ替えることができる。ファノの不等式は任意の目標誤り率を相互情報量の最小要件へと変換し、任意の確率モデルの交差エントロピー損失 L は $I(X;Y) \geq H(Y) - L$ を保証する。Shannon の通信路容量とレート・歪み関数は、それぞれ入力に関する相互情報量の最大値と、非可逆エンコーダに関する最小値として現れる。最後に、情報ボトルネック $\min I(X;Z) - \beta I(Y;Z)$ は表現学習をラベルによって歪みが与えられるレート・歪み問題として定式化する。ガウス対に対しては β が $1/\rho^2$ を超えるまで崩壊し、変分形式においてはそのレートペナルティが変分オートエンコーダでおなじみの KL 正則化項となる。

37.6.6 演習問題

1. [code] **低頻度対と PMI**。自己相互情報量 Equation 37.57 は、共起回数を 2 つの単語の頻度で補正する（37.6.1.4 節）。
    1. $\mathrm{pmi}(x,y) \leq \min\{-\log p_X(x), -\log p_Y(y)\}$ を示せ。2 つの上界のそれぞれが達成されるのはどのようなときか。
    2. `counts` を拡張し、バイグラム「sri lanka」の中でのみ出現回数 2 で現れる 4 番目の第 1 語 **sri** と 4 番目の第 2 語 **lanka** を追加せよ。「new york」、「machine learning」、「the day」、「sri lanka」の pmi を再計算し、さらに出現回数 1 でも同様に計算せよ。出現回数 0 のセルでは $\mathrm{pmi} = -\infty$ となって `(p * pmi).sum()` が `nan` を返すため、`mutual_information` と同様に非ゼロのセルに限定して和をとること。
    3. 「new york」と「sri lanka」がもつ寄与 $p_{X,Y}(x,y)\,\mathrm{pmi}(x,y)$（$I(X;Y)$ に対する寄与）を比較せよ。続いてすべての出現回数に 1 を加えて両方の pmi の値を再計算せよ。連語をランク付けするにはこれらのスコアのうちどれを用いるべきか。また、それはなぜか。
2. **情報量と利用可能情報量**。ラベル Y は公平なビットである。$Y = 0$ のとき入力 $X \in \mathbb{R}^2$ は半径 1 の円周上の一様分布に従い、$Y = 1$ のときは半径 2 の円周上の一様分布に従う。表現として $Z = \|X\|^2$ を計算する。

1. $I(Y;Z) = I(Y;X) = \ln 2$ が成り立つことを示せ（ただし $x \mapsto \|x\|^2$ は可逆ではない）。37.6.2.2 節のデータ処理不等式における等号成立条件のうち、どれが成り立っているか。
2. いかなる線形分類器 $\hat{y} = \mathbf{1}(\mathbf{w}^\top \mathbf{x} + b > 0)$ もすべての入力を正しく分類できない一方で、Z に対する閾値処理はすべて正しく分類できることを示せ。
3. ネットワークの各層はマルコフ連鎖 $X \to Z_1 \to Z_2 \to \cdots$ をなす。Z_ℓ を読み取るあらゆる種類の分類器の中で最良のものの誤り率は、ℓ が大きくなっても低下し得ないことを示せ。それにもかかわらず、Z_ℓ に当てはめた線形分類器の正解率は ℓ とともに上昇することが多い。なぜここに矛盾はないのか。
4. Xu et al. (2020) は、予測器族 $\mathcal{V}$（予測器 $q(y \mid x)$ の集合）が抽出できる情報量を測っている。$H_{\mathcal{V}}(Y \mid X) = \inf_{q \in \mathcal{V}} \mathbb{E}[-\log q(Y \mid X)]$ とおき、$H_{\mathcal{V}}(Y)$ を入力を無視する $\mathcal{V}$ 内の予測器にわたる同じ下限とする。**利用可能情報量**は次で定義される。

$$I_{\mathcal{V}}(X \to Y) = H_{\mathcal{V}}(Y) - H_{\mathcal{V}}(Y \mid X).$$

   $\mathcal{V}$ を与えられた入力上のロジスティック回帰とする。$I_{\mathcal{V}}(X \to Y) = 0$ かつ $I_{\mathcal{V}}(Z \to Y) = \ln 2$ を示せ。X については、期待ロジスティック損失が $(\mathbf{w}, b)$ に関して凸であり、$\mathbf{w}$ を回転させても不変であることを用いよ。37.6.2.2 節のどの不等式に $I_{\mathcal{V}}$ は反するか。

3. **シナジーとベン図**。X と Y を独立な公平なビットとし、$Z = X \oplus Y$ をその排他的論理和とする。本節ではこれについて $I(X;Y) = 0$ および $I(X;Y \mid Z) = \ln 2$ を計算した。
   1. マルコフ連鎖 $X \to Z \to Y$、$X \to Y \to Z$、$Y \to X \to Z$ のうち成り立つものはどれか。したがってデータ処理不等式はこの 3 つ組について何を主張するか。
   2. Figure 37.13 の 3 変数版では、3 つの円盤すべてに共通する領域が**交互作用情報量** $I(X;Y;Z) = I(X;Y) - I(X;Y \mid Z)$ に対応するはずである。$I(X;Y;Z) = H(X) + H(Y) + H(Z) - H(X,Y) - H(X,Z) - H(Y,Z) + H(X,Y,Z)$ を証明し、$I(X;Y;Z)$ が 3 つの引数に関して対称であると結論せよ。
   3. $I(X;Y;Z)$ を (a) の 3 つ組と、1 つの公平なビットの 3 つのコピー $X = Y = Z$ のそれぞれに対して求めよ。これら 2 つの値は、エントロピーの 3 円盤図に対して何を意味するか。

   Thomas M. Cover、Joy A. Thomas 著 Elements of Information Theory, 2nd ed. 演習問題 2.25、および David J. C. MacKay 著 Information Theory, Inference, and Learning Algorithms 演習問題 8.8 に基づく。

4. [code] **f ダイバージェンスからのデータ処理不等式**。入力をランダムな出力へ写す任意の通信路 κ と、入力の任意の 2 つの分布 P および Q に対し、出力分布は $D_f(P\kappa \,\|\, Q\kappa) \le D_f(P \,\|\, Q)$ をすべての f ダイバージェンスについて満たす（37.3.3.3 節）。KL ダイバージェンスと $\chi^2(P\|Q) = \sum_x p(x)^2/q(x) - 1$ はその 2 つの例である。
   1. $X \to Y \to Z$ をマルコフ連鎖とする。$P = P_{X,Y}$、$Q = P_X \otimes P_Y$ とし、通信路 κ を (x,y) から (x,z)（ただし $z \sim p(z \mid y)$）への写像とする。$P\kappa = P_{X,Z}$ かつ $Q\kappa = P_X \otimes P_Z$ を示し、$I(X;Z) \le I(X;Y)$ を結論せよ。
   2. 本節のデータ処理不等式の命題のうち、この証明経路からは得られない部分はどれか。
   3. 確率 0.1 と 0.2 の直列の 2 回反転について、2 回目の反転の前後それぞれで、同時確率表とその周辺分布の積との間の KL ダイバージェンスおよび χ^2 ダイバージェンスを計算せよ。

4. 公平な入力を確率 δ の反転通信路に通したとき $\chi^2(P_{X,Y} \,\|\, P_X \otimes P_Y) = (1-2\delta)^2$ となることを示し、(c) の数値を確認せよ。さらに確率 δ' で反転させると、このダイバージェンスに $(1-2\delta')^2$（ここでは 0.36）が掛かると結論せよ。(c) における比 $I(X;Z)/I(X;Y)$ も同じ倍率になるか。

5. **ファノの不等式とその逆。**●ファノの不等式 Equation 37.59 は、推定値 $\hat{Y} = g(X)$（ラベル Y が k 個の値をとるときの任意の推定値）の誤り率を上からではなく下から抑える。

   1. $\hat{Y}(x) = \arg\max_y p(y \mid x)$ を最尤ラベルとする。その誤り率が $P_e \le 1 - e^{-H(Y|X)}$ を満たすことを示せ。まず $\max_y p(y \mid x)$ を下から $\sum_y p(y \mid x)^2 = \mathbb{E}[p(Y \mid X) \mid X = x]$ で抑え、次にイェンセンの不等式を $\mathbb{E}[p(Y \mid X)] = \mathbb{E}\big[e^{\log p(Y|X)}\big]$ に 1 回適用せよ。
   2. 観測 X が $I(X;Y) = 9$ nat の情報を、均衡した 10,000 クラスのラベル Y について保持しているとする。Equation 37.59 と (a) を用いて、最良の分類器の誤り率を上下から挟み込め。不等式 Equation 37.59 は P_e について閉形式解をもたないため、数値的に解くこと。この区間の幅は、相互情報量から下流タスクの誤り率を予測することについて何を示唆しているか。

6. **損失による保証とファノの不等式。**均衡した $k = 10$ クラスの分類器がテスト誤差 $P_e = 0.08$ およびテスト損失 $L = 0.30$ nat をもつとする。

   1. $I(X;Y)$ について、ファノの不等式 Equation 37.59 は何 nat を保証し、損失による保証 Equation 37.60 は何 nat を保証するか。
   2. 任意の推定値 $\hat{Y} = g(X)$（誤り確率 P_e）に対し、モデル $q(y \mid x)$ が確率 $1 - P_e$ を $y = g(x)$ に置き、残りの P_e を他の $k-1$ 個のラベルに均等に配分するとする。その交差エントロピー損失が $H_b(P_e) + P_e \log(k-1)$ となることを示し、ファノの不等式はこのモデルの損失による保証であると結論せよ。これは 37.6.2.4 節で評価した 4 つのモデルのうちどれに当たるか。
   3. (a) の分類器は、その判定結果から (b) が構成するモデルよりも小さい損失を報告している。それが起こるためには、分類器の確信度はどのように振る舞わなければならないか。2 値ラベルをもつ対 (X, Y) と誤り率 P_e をもつモデルであって、ファノの不等式より真に多くの情報量を保証する具体例を挙げよ。

7. **2 元通信路。**●2 値エントロピー H_b を用い、nat 単位で計算せよ。

   1. **Z 通信路**は 0 をそのまま通過させ、1 を確率 $\frac{1}{2}$ で 0 に変える。$I(X;Y) = H(Y) - H(Y \mid X)$ を $a = P(X=1)$ の関数として計算し、$a = \frac{2}{5}$ で最大となってその最大値が $\ln \frac{5}{4}$ nat に等しいことを示せ。最良の入力が信頼できない記号を送る割合を半分未満にするのはなぜか。また、まったく送らないわけではないのはなぜか。
   2. 特徴量が値 ± 1 をとり、率 α のドロップアウト（確率 α で 0 に置き換え、それ以外は定数倍する処理）を通過するとする。この **2 元消失通信路**の容量が $(1-\alpha)\ln 2$ nat であることを示せ。
   3. 2 元対称通信路の容量が $\alpha = 0.5$ のドロップアウトと同じになる反転確率 δ を求めよ。ビットの反転が消去よりも大きく容量を損なうのはなぜか。

8. **ガウス通信路。**$Y = X + N$ とし、ノイズ $N \sim \mathcal{N}(0, \sigma_N^2)$ は X と独立であり、入力は電力制約 $\mathbb{E}[X^2] \le P$ に従うとする。

   1. 確率密度をもつ任意の入力に対して $I(X;Y) = h(Y) - h(N)$ が成り立つことを示せ。
   2. 分散 s^2 をもつ密度の中で微分エントロピーが最大になるのはガウス分布であり、その値は $\frac{1}{2}\log(2\pi e s^2)$ である。これを用いて $h(Y) \le \frac{1}{2}\log\big(2\pi e(P + \sigma_N^2)\big)$ を示し、

容量が $C = \frac{1}{2}\log(1 + P/\sigma_N^2)$ であってガウス入力により達成されると結論せよ。

3. 信号対雑音比 $P/\sigma_N^2 = 1, 10, 100$ における C を nat 単位およびビット単位で求めよ。また高い信号対雑音比においては、P を 2 倍にするごとに約 $\frac{1}{2}\ln 2$ nat（半ビット）が加わることを示せ。
4. 拡散モデルは、$\mathbb{E}[X^2] = 1$ となるようスケールされたデータの座標 X を $\sqrt{a}\,X + \sqrt{1-a}\,\epsilon$（ただし $0 < a < 1$、また $\epsilon \sim \mathcal{N}(0,1)$ は独立）に置き換える。ノイズを加えられた座標が保持する情報は高々 $\frac{1}{2}\log\frac{1}{1-a}$ nat（X についての情報量）であることを X の分布によらず示し、この上界を $a = 0.5$ と $a = 0.01$ で評価せよ。この上界が達成されるのはどのようなデータに対してか。

9. **変分オートエンコーダのレート上界。** $X \sim \mathcal{N}(0,1)$ とし、エンコーダを 37.6.4.2 節と同じく $Z = X + \sigma\epsilon$（したがって $p(z \mid x) = \mathcal{N}(z; x, \sigma^2)$）とし、事前分布を $r = \mathcal{N}(0,1)$ とする。$D_{\mathrm{KL}}(\mathcal{N}(\mu, s^2) \,\|\, \mathcal{N}(0,1)) = \frac{1}{2}(\mu^2 + s^2 - 1 - \log s^2)$ を用いてよい。

   1. Equation 37.70 の 3 つの項を計算し、恒等式を確かめよ。$\sigma = 1$ において、KL ペナルティはレートを何 nat 過大評価しているか。
   2. 符号を $Z' = Z/\sqrt{1+\sigma^2}$ へと定数倍し直せ。$I(X; Z') = I(X; Z)$ が成り立ち、かつ上界が等号で成り立つようになることを示せ。
   3. レートは符号の単位に依存しないが、この上界は単位に依存する。与えられたエンコーダに対して上界を等号にする事前分布 r はどれか。したがって KL ペナルティを最小化することは、レートを下げるほかに符号に対してどのような作用を及ぼすか。

37.7 相互情報量の推定

これまでに計算してきた相互情報量（mutual information）は、いずれも閉じた形の同時密度（joint density）から求められたものであった。しかし実際には、データに確率密度が添えられていることは決してない。その代わりに与えられるのは、発話音声とその書き起こし、あるいは検索クエリとそれに答えた文書といったペアの集まりであり、そうしたサンプルだけから、ペアの一方が他方についてどれだけの情報を明らかにするかを定量化したいのである。表現学習（representation learning）はこの考え方をさらに一歩推し進め、共有される情報量が大きくなるようにモデルを訓練することで、推定値そのものを訓練の目的関数へと変える。

よく知られた対照損失（contrastive loss）を例にとると、この課題が具体的になる。SimCLR や CLIP などのシステムは、1 枚の写真から切り出した 2 つのランダムクロップや、画像とそのキャプションの組のように、同一データ例のペアになった**ビュー**を用いて訓練する。一方のビューが与えられると、モデルは InfoNCE 損失として知られる交差エントロピー損失（cross-entropy loss）を最小化することで、バッチ内の候補（たとえば 128 個）の中から真の対応相手を識別するよう学習する。論文ではしばしば、この損失を $\ln 128 = 4.85$ から差し引き、その差を 2 つのビュー間の相互情報量として報告する。しかし、2 つのビューが実際にどれほど多くの情報を共有していようとも、その差が 4.85 nat を超えることは決してない。毎回正しい相手を選び出すモデルであっても、128 個の候補を識別できることを証明したにすぎないからである。このような量を用いてモデルを訓練すると同時にそれを測定値としても解釈する以上、真の測定値と下界（lower bound）とを区別し、さらに下界の緩み（slack）と学習されたモデルの誤差とを切り分ける必要がある。

サンプルから相互情報量を推定する方法は大きく 2 つある。直接的なアプローチは、ヒストグラムのビン内または最近傍（nearest neighbor）領域内に入るサンプル数を数える手法であり、低次元変数間の小さな情報量に対してはうまく機能するが、ベクトルに対しては破綻する。第 2 のアプローチは、未知の密度を学習されたモデルに置き換え、得られる期待値が $I(X; Y)$ の下界となるように構成したうえで、その下界を最大化する。本節の残りの議論を支える柱は 2 つの結果である。すなわち、分布に関する仮定なしに任意の推定量が保証できる限界という根本的な制約

（37.7.2.4 節）と、InfoNCE 損失の背後にある変分下界およびその対数的な上限（37.7.4 節）である。

```
from scipy.spatial.distance import cdist
from scipy.special import digamma, logsumexp
```

37.7.1 既知の情報量を持つペア

推定量を公平に評価するには、厳密な相互情報量があらかじめわかっているベンチマークが必要である。本節を通じて、37.6.1.3 節で導入した相関（correlation）ρ を持つ標準 2 変量ガウス分布のペアを用いる。このペアが共有する情報量は次の式で与えられる。

$$I(X;Y) = -\tfrac{1}{2}\log\left(1-\rho^2\right) \tag{37.71}$$

単位は nat であり、その値は 0.83 nat（$\rho = 0.9$ のとき）および 1.96 nat（$\rho = 0.99$ のとき）となる。なお、本節の対数はすべて自然対数である。

これから導出する変分下界は、37.6.1.4 節の**自己相互情報量**（pointwise mutual information）、すなわち $\operatorname{pmi}(x,y) = \log \frac{p(x,y)}{p(x)\,p(y)}$ に依拠している。これは、特定のペア (x,y) が独立性（independence）のもとと比べて同時分布（joint distribution）のもとでどれだけ起こりやすいかを測る量である。標準ガウス分布のペアに対して、pmi は次のような単純な二次関数（quadratic function）になる。

$$\operatorname{pmi}(x,y) = -\tfrac{1}{2}\log(1-\rho^2) - \frac{x^2 - 2\rho xy + y^2}{2(1-\rho^2)} + \frac{x^2+y^2}{2}. \tag{37.72}$$

この定義から直ちに 2 つの恒等式が導かれる。第 1 に、同時分布からサンプリングされたペアにわたって $\operatorname{pmi}(x,y)$ の平均をとると $I(X;Y)$ になる。第 2 に、その指数をとった $e^{\operatorname{pmi}(x,y)}$ を x と y の**独立な**サンプルにわたって平均すると 1 になる。これは $p(x)\,p(y)\,e^{\operatorname{pmi}(x,y)} = p(x,y)$ を積分すると 1 になるからである。両方の恒等式をサンプル上で確かめてみよう。第 2 の恒等式については、配列内で各 x を次のサンプルの y と組にすることで従属性を断ち切る。これらを**非対応ペア**（mismatched pairs）と呼ぶことにする。各非対応ペアは $P_X \otimes P_Y$ からの独立なサンプルであり、下界の計算に独立なペアが必要になるたびにこの単純な手法を再利用する。

```
def gauss_pair(rng, rho, n):
    """Samples of the standard Gaussian pair; n is a count or a shape."""
    x = rng.standard_normal(n)
    y = rho * x + np.sqrt(1 - rho**2) * rng.standard_normal(n)
    return x, y

def mi_gauss(rho):
    """I(X;Y) of the pair, in nats."""
    return -0.5 * np.log(1 - rho**2)

def pmi_gauss(x, y, rho):
    """Pointwise mutual information pmi(x, y) of the pair, in nats."""
    c = 1 - rho**2
    return (-0.5 * np.log(c) - (x**2 - 2 * rho * x * y + y**2) / (2 * c)
            + (x**2 + y**2) / 2)
```

```
rng = np.random.default_rng()
x, y = gauss_pair(rng, 0.9, 200_000)
print(f'I(X;Y) = {mi_gauss(0.9):.3f} nats, mean pmi over pairs = '
      f'{pmi_gauss(x, y, 0.9).mean():.3f}')
print(f'mean exp(pmi) over independent pairs = '
      f'{np.exp(pmi_gauss(x, np.roll(y, 1), 0.9)).mean():.3f}')
```

```
I(X;Y) = 0.830 nats, mean pmi over pairs = 0.831
mean exp(pmi) over independent pairs = 0.999
```

どちらの標本平均も、理論値と約 0.01 以内の精度で一致している。本節のすべての変分下界はこれら 2 つの恒等式の上に築かれており、統計的な困難のほぼすべてを引き起こすのは第 2 の恒等式の方である。$\rho = 0.9$ では e^{pmi} の振る舞いは穏やかであり、独立なペアにわたるその分散（variance）は $e^{2I} - 1$（演習問題 1）であって、ここではわずか 4.3 にすぎない。ところが $I = 6$ になると、その分散は $160{,}000$ へと爆発し、同じ $200{,}000$ サンプルの平均であっても約 0.9 もの不確実性が残る。

37.7.2 ノンパラメトリック推定

データから確率分布を推定する最も直接的な方法は、空間の各領域に何個のサンプルが入るかを数えることである。設計上の重要な選択は、それらの領域をどう定義するかにある。まずは最も単純な選択である固定グリッドから始めよう。

37.7.2.1 ヒストグラム

ヒストグラム推定量は、各変数の値域を b 個の等幅区間に分割し、$b \times b$ 個の各セルに入るサンプル数を数えて、得られた経験頻度を離散版の相互情報量の公式に代入する。この推定量は単純であるにもかかわらず、互いに逆方向に働く 2 つの競合する統計的バイアス（estimator bias）を抱えている。

第 1 のバイアスは、推定値を**下方向**に引き下げる。連続値をそのビンのインデックスに置き換えることは X に決定論的な関数を適用することにほかならず、データ処理不等式（data processing inequality、37.6.2.2 節）により、X の任意の関数が Y と共有する情報量は高々 X 自身が共有する情報量までにとどまる。したがってビン分割は情報を失わせることしかなく、従属性の幅が 1 つのビンより狭いときにはその損失が深刻になる。たとえば $\rho = 0.99$ では、y の x が与えられたもとでの条件付き標準偏差（standard deviation）は $\sqrt{1 - \rho^2} = 0.14$ である。おおよそ $[-4.5, 4.5]$ に広がるサンプルを 10 個のビンに分割するとセルの幅は 0.9 となり、条件付きの広がりの 6 倍以上になってしまう。

第 2 のバイアスは、推定値を**上方向**に押し上げる。プラグイン推定値は、経験同時度数表と、その行・列の周辺分布の積との間の KL ダイバージェンスである。そのため常に非負であり、表が厳密に積へ分解される場合に限り 0 になる。たとえ X と Y が完全に独立であっても、ランダムなサンプリングの揺らぎによって厳密な分解は妨げられるため、ノイズが従属性を装ってしまう。各セルに含まれるサンプル数が少ないほど、この上向きのバイアスは悪化する。

これら 2 つの誤差に加えて、ヒストグラムには厳密な上限がある。推定値はビン分割された X のエントロピーを超えることができず、b 個のビンが担える情報量は高々 $\log b$ nat である。ビン数が 10 個ならわずか 2.30 nat であり、X と Y がどれほど強く結びついていてもこれを超えることはない。本節で扱うすべての推定量において、この上限の変種に直面することになる。いかなる手法も、識別可能な状態（state）の数の対数を超える情報量を報告することはできないのである。

この推定量を `binned_mi` として実装し、4 つの分布から生成した 200,000 個のサンプルに対して $b = 10$、40、160 を用いてテストしてみよう。対象とする 4 つの分布は、独立なペア、$\rho = 0.9$ および $\rho = 0.99$ のガウス分布のペア、そして放物線ペア $Y = X^2 + 0.3\,\epsilon$（37.6.1.5 節）である。この放物線ペアは線形相関が 0 でありながら 1.141 nat を共有する。

```
def binned_mi(x, y, bins):
    """Plug-in MI of the bins x bins histogram of (x, y), in nats."""
    p = np.histogram2d(x, y, bins=bins)[0] / len(x)       # cell frequencies
    q = p.sum(axis=1, keepdims=True) * p.sum(axis=0, keepdims=True)
    nz = p > 0                                            # occupied cells
    return float((p[nz] * np.log(p[nz] / q[nz])).sum())

mi_parabola = 1.141  # by quadrature, sec_mdl-mi-nonlinear-correlation
n = 200_000
x = rng.standard_normal(n)
pairs = {'rho = 0': (gauss_pair(rng, 0.0, n), 0.0),
         'rho = 0.9': (gauss_pair(rng, 0.9, n), mi_gauss(0.9)),
         'rho = 0.99': (gauss_pair(rng, 0.99, n), mi_gauss(0.99)),
         'parabola': ((x, x**2 + 0.3 * rng.standard_normal(n)), mi_parabola)}
print('pair        truth  b = 10  b = 40  b = 160')
for name, ((u, v), truth) in pairs.items():
    print(f'{name:10}  {truth:5.3f}'
          + ''.join(f'  {binned_mi(u, v, b):6.3f}' for b in (10, 40, 160)))
```

```
pair        truth  b = 10  b = 40  b = 160
rho = 0     0.000   0.000   0.003   0.035
rho = 0.9   0.830   0.613   0.813   0.845
rho = 0.99  1.959   1.108   1.798   1.958
parabola    1.141   0.539   0.983   1.131
```

表には両方のバイアスがはっきりと現れている。$b = 10$ では、従属なすべてのペアで値が低く出ており、約 0.6 nat（真の値は 0.83 nat、$\rho = 0.9$ のとき）にとどまり、$\rho = 0.99$ と放物線ペアでは真の値の約半分にしかならない。$b = 160$ では、独立なペアに 100 分の数 nat ほどの値が計上され、$\rho = 0.9$ のペアでは実際に真の値を上回ってしまっている。これは粗いビン分割だけでは決して起こり得ない現象である。$\rho = 0.99$ かつ $b = 160$ では推定値が真の値の約 0.01 以内に収まっているが、これは逆向きの誤差がたまたま打ち消し合った結果にすぎず、そのすぐ上の行では同じグリッドが過大推定を起こしている。真の値が未知のとき、どの列を信頼すべきかをこの表から知る術はない。

3 つ目の弱点は座標への敏感さである。どちらかの変数に可逆変換を施しても相互情報量は変化しないが（37.6.1.5 節）、固定幅のグリッドは歪んでしまう。x を x^3 に置き換える操作を、$\rho = 0.9$ のガウス分布のペアに施したとしよう。標準正規分布からの 200,000 個のサンプルのうち最大値はおおよそ ± 4.5 であるため、x^3 の範囲はおおよそ ± 90 に広がる。すると 40 個の等幅ビンの幅はそれぞれ約 4.5 となり、サンプルの約 10 個中 9 個が 0 付近の 2〜3 個のビンに詰め込まれてしまう。明快な解決策は、ソートしたサンプル内での各値の位置を表す**順位**（rank）をビン分割することである。任意の狭義単調増加写像は順位を保存し、どの順位ビンにも自動的に同じ数のサンプルが入る。

```
x, y = gauss_pair(rng, 0.9, n)
rank = lambda a: np.argsort(np.argsort(a))       # position in sorted order
print(f'true I = {mi_gauss(0.9):.3f} nats; histograms with b = 40:')
```

```
    for name, u in (('x', x), ('x**3', x**3)):
        print(f'{name:4}: correlation {np.corrcoef(u, y)[0, 1]:.3f}, '
              f'raw {binned_mi(u, y, 40):.3f}, '
              f'ranks {binned_mi(rank(u), rank(y), 40):.3f} nats')
```

```
true I = 0.830 nats; histograms with b = 40:
x   : correlation 0.900, raw 0.817, ranks 0.816 nats
x**3: correlation 0.695, raw 0.480, ranks 0.816 nats
```

真の相互情報量は 0.830 nat のままであるにもかかわらず、x を 3 乗すると線形相関は 0.90 から約 0.7 へと低下し、生の値によるヒストグラム推定値は 0.3〜0.5 nat の間へと激減する（正確な値はビンの範囲を決める少数の極端なサンプルに依存する）。これに対し、順位をビン分割すれば、x と x^3 で桁の隅々まで完全に同一の推定値が得られ、真の値との差も 0.02 nat 以内に収まる。

順位を用いれば座標の問題は解決するが、高次元におけるヒストグラムの破綻はどうにも救いようがない。5 次元ベクトルのペアには 10 次元のグリッドが必要になり、各軸にわずか 10 個のビンを置くだけでも 10^{10} 個のセルが生じ、現実的なあらゆるサンプルサイズにおいてそのほとんどすべてが空になってしまう。ベクトルを扱うには、データ自身に解像度を決めさせ、サンプルが密な場所では小さな領域を、疎な場所では大きな領域を使うような推定量が必要である。

37.7.2.2 KSG 推定量

KSG として知られる Kraskov et al. (2004) の推定量は、固定されたヒストグラムのセルを、各データ点の周囲の適応的な最近傍領域に置き換える。これは低次元変数のペアに対する標準的なノンパラメトリック手法である（本項と次項は初読時には読み飛ばしてもよい）。

KSG では座標差の最大値（チェビシェフ距離、すなわち ℓ_∞ 距離）を用いて距離を測るため、半径 r の球は 2 次元では正方形、より高次元では超立方体になる。各サンプル点 i の周囲に、同時空間 (x, y) 内で m 番目の最近傍点に達する最小のボックスを描き、そのボックスの半幅を r_i とする。構成法から、このボックスには割合 m/n（全 n サンプル中）のサンプルが含まれ、(x_i, y_i) 近傍の同時確率（joint probability）質量の局所的な推定値を与える。

次に、まったく同じスケール r_i で 2 つの周辺確率を測る。$n_x^{(i)}$ を縦方向のストリップ（帯状領域）$|x - x_i| < r_i$ の厳密な内部に入る他のサンプルの数とし、$n_y^{(i)}$ を横方向のストリップ $|y - y_i| < r_i$ の厳密な内部に入る他のサンプルの数とする。半幅 r_i を共有しているからこそ、生のカウント数を確率密度の代わりに使えるのである。もしボックス全体で密度がほぼ一定であれば、ボックスが持つ確率質量は約 $p(x_i, y_i)\,(2r_i)^2$ となり、一方で 2 つのストリップが持つ確率質量はそれぞれ約 $p(x_i)\,2r_i$ と $p(y_i)\,2r_i$ となる。ボックス内の割合を 2 つのストリップ内の割合の積で割ると、幾何学的な因子 $2r_i$ が約分されて消える。その対数をとれば、純粋にカウントだけで計算された自己相互情報量の局所推定値が得られる。Figure 37.18 に、1 つの点におけるこれら 3 つのカウントを示す。

これら局所的な対数比を全 n 点にわたって平均すると、バイアス補正（bias correction）にあたる 1 つの有限標本補正を伴って $I(X;Y)$ の推定値が得られる。小さなカウントの対数をとると対数確率（log probability）の推定値にバイアスが生じる。すなわち、m 番目の近傍点（全 $n-1$ 個の近傍点のうち）に達するボックスについて、対数確率質量の期待値は $\mathbb{E}[\log p] = \psi(m) - \psi(n)$ となり、$\log(m/n)$ とは一致しない。ここで $\psi = (\log\Gamma)'$ はディガンマ関数である（導出は省略する）。対数を ψ に置き換えることで、KSG 推定量が次のように得られる。

$$\hat{I}_{\text{KSG}} = \psi(m) + \psi(n) - \frac{1}{n}\sum_{i=1}^{n}\left(\psi\left(n_x^{(i)} + 1\right) + \psi\left(n_y^{(i)} + 1\right)\right). \tag{37.73}$$

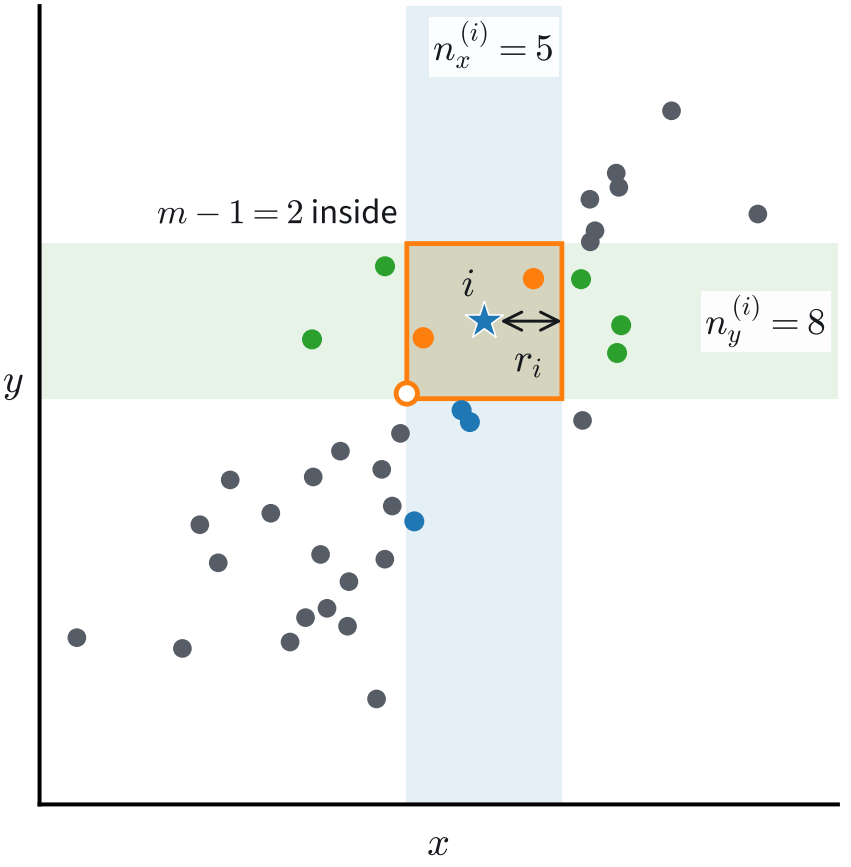


Figure 37.18: サンプル点 i（星印）における $m = 3$ の KSG カウント。半幅 r_i のボックスは同時平面内で 3 番目の最近傍点（左辺上の丸印。横ストリップ内にあるため $n_y^{(i)}$ にのみカウントされる）に達しており、2 つの近傍点が厳密に内部に入っている。同じ半幅の縦ストリップには $n_x^{(i)}$ 個、横ストリップには $n_y^{(i)}$ 個の他のサンプルが入る（青と緑の点は一方のストリップにのみ含まれる）。推定量の各被加数は近似的に $\log \frac{m/n}{(n_x/n)(n_y/n)}$ であり、カウントから計算された自己相互情報量となっている。

$\psi(j) \approx \log j$（j が大きいとき）が成り立つため、各被加数は近似的に $\log \frac{m\,n}{n_x^{(i)}\,n_y^{(i)}}$ となり、ボックス内の割合とストリップ内の割合の積との局所的な対数比に一致する。ストリップの項の中にある $+1$ はボックスと基準を揃えるためのものである。つまり、$\psi(m)$ は境界が m 番目の近傍点を通るボックスに対応するため、厳密な内部に $n_x^{(i)}$ 個の点を含むストリップは、あたかもその境界が $(n_x^{(i)}+1)$ 番目の点を通るかのように扱われる。両方のストリップのカウントには狭義の不等号を用い、点 i 自身を除外するため、ストリップの境界上にちょうど乗っている近傍点はカウントされない。すべての半径 r_i が局所的なサンプル密度に自動的に適応するため、調整すべきビン幅やカーネルのバンド幅は存在しない。

$n \times n$ のチェビシェフ距離行列を x と y のそれぞれについて `cdist` で求め、同時距離として要素ごとの最大値をとれば、すべての近傍カウントを一度に計算できる。

```
def ksg_counts(x, y, m=5):
    """Strip counts n_x, n_y of every sample point; x, y are (n, d)."""
    dx, dy = cdist(x, x, 'chebyshev'), cdist(y, y, 'chebyshev')
    d = np.maximum(dx, dy)                       # joint Chebyshev distance
    np.fill_diagonal(d, np.inf)
    r = np.partition(d, m - 1, axis=1)[:, m - 1]  # half-width: m-th neighbor
    n_x = (dx < r[:, None]).sum(axis=1) - 1      # strict, self excluded
    n_y = (dy < r[:, None]).sum(axis=1) - 1
    return n_x, n_y

def ksg_mi(x, y, m=5):
    """KSG nearest-neighbor estimate of I(X;Y) in nats; x, y are (n, d)."""
    n_x, n_y = ksg_counts(x, y, m)
    return float(digamma(m) + digamma(len(x))
                 - (digamma(n_x + 1) + digamma(n_y + 1)).mean())
```

Equation 37.73 の各被加数は近傍カウントのみに依存するため、カウントを用いること自体が KSG の報告し得る値に厳密な上限を課す。

命題（KSG のサンプルサイズによる上限）同一の点からの同時距離がどの 2 点についても等しくない任意の n 点のサンプルに対して、次が成り立つ。

$$\hat{I}_{\mathrm{KSG}} \le \psi(n) - \psi(m), \tag{37.74}$$

等号は $Y = X$ のときに成り立つ。

証明距離の同値（タイ）がないため、ちょうど $m-1$ 個のサンプルが点 i に対してその m 番目の近傍点より厳密に近い位置にある。同時チェビシェフ距離は 2 つの座標距離のうち大きい方であるため、これら $m-1$ 個の各点は $|x_j - x_i| < r_i$ と $|y_j - y_i| < r_i$ の両方を満たし、したがって両方のストリップに含まれる（すなわち $n_x^{(i)} \ge m-1$ かつ $n_y^{(i)} \ge m-1$）。ψ は狭義単調増加であるため、各被加数は高々 $\psi(m) + \psi(n) - 2\psi(m) = \psi(n) - \psi(m)$ である。$Y = X$ のときは同時距離と周辺距離が一致するため、両ストリップにはボックスの厳密な内部にある $m-1$ 個の点のみが含まれ、すべての被加数がこの上界を達成する。■

$n = 2{,}000$、$m = 5$ のとき、この上限は $\psi(2000) - \psi(5) = 6.09$ nat であり、$\log(n/m) = 5.99$ に近い。$Y = X$ かつ X が連続で真の相互情報量が無限大になる場合であっても、KSG が報告する値は 6.1 nat であり、サンプルサイズを 2 倍にしてもその上限は約 $\log 2$ しか上がらない。ここでも再び、ヒストグラムの $\log b$ という上限が、ビンを適応的な近傍に置き換えた形で現れて

いる。近傍をカウントする推定量が報告できるのは、サンプルから作れる互いに異なる近傍の数（おおよそ n/m）の対数が上限なのである。

37.7.2.3 高次元における KSG

残念ながら高次元では、幾何学的な理由から、KSG はサンプルサイズによる上限よりはるかに低いところで破綻する。ガウス分布のペアの d 個の独立なコピー（それぞれ $\rho = 0.9$）を考え、X と Y を d 次元ベクトルとする。対数密度（log density）比が d 個の独立な項の和に分解されるため、相互情報量は独立な座標にわたって加法的であり、真の値は $d \times 0.83$ nat となる。

KSG は同時密度と周辺密度が各近傍ボックス内でほぼ一定であることに依拠しているが、この仮定は次元が増えるにつれて崩壊する。$n = 2{,}000$、$m = 5$ のとき、各ボックスはサンプルの割合 $m/n = 0.0025$ を捕捉する。もし点が $2d$ 次元空間に一様に広がっていれば、そのようなボックスはどの軸に沿ってもデータ幅の割合 $(m/n)^{1/(2d)}$ にわたって広がることになり、0.05（$d = 1$ のとき）から、0.47（$d = 4$）、さらには 0.83（$d = 16$）へと跳ね上がる。ところが実際の従属性は $y = \rho x$ の周囲の細い帯の内部に集中している。x が与えられたとき、y の各座標の標準偏差は $\sqrt{1-\rho^2} = 0.44$ であり、約 0.07 倍（サンプル全体の広がりである約 -3.3〜3.3 に対する比率）にすぎない。その帯より何倍も広いボックスは、粗すぎるヒストグラムのビンとまったく同様に、従属性を塗りつぶしてしまう。2,000 個のサンプルに対して $d = 1, 2, 4, 8, 16$ で KSG を実行し、推定値とストリップ内の平均カウント数を、上限を達成する $Y = X$ の場合とあわせて記録してみよう。

```
n, rho = 2000, 0.9
print(' d  true I    KSG  mean strip count')
for dim in (1, 2, 4, 8, 16):
    x, y = gauss_pair(rng, rho, (n, dim))        # dim independent pairs
    n_x, n_y = ksg_counts(x, y)
    print(f'{dim:2d}  {dim * mi_gauss(rho):6.2f}  {ksg_mi(x, y):5.2f}  '
          f'{(n_x.mean() + n_y.mean()) / 2:16.1f}')
x = rng.standard_normal((n, 1))
print(f'Y = X:  KSG = {ksg_mi(x, x):.2f}, '
      f'ceiling psi(n) - psi(m) = {digamma(n) - digamma(5):.2f} nats')
```

```
 d  true I    KSG  mean strip count
 1    0.83   0.82              72.0
 2    1.66   1.65              48.9
 4    3.32   2.48              31.7
 8    6.64   2.81              26.3
16   13.29   2.79              26.2
Y = X:  KSG = 6.09, ceiling psi(n) - psi(m) = 6.09 nats
```

1 次元および 2 次元では、KSG は真の値から約 0.1 nat 以内に収まっている。しかし $d = 4$ になると、約 2.5 nat（真の値は 3.3 nat）へと落ち込み、1 nat 近く低くなる。その後は頭打ちになり、$d = 8$ でも $d = 16$ でも、KSG が報告する値は 3 nat 弱にとどまる（その間に真の相互情報量は 6.6 から 13.3 へと倍増している）。ここで推定を阻んでいるのはサンプルサイズによる上限ではないことに注目したい。3 nat という値は、出力された 6.1 nat という上限（$Y = X$ のとき）の半分未満だからである。

ストリップ内の平均カウント数を見れば、なぜ推定値が頭打ちになるかがわかる。各被加数は近似的に $\log \frac{m\,n}{n_x\,n_y}$ であるため、推定値が上昇できるのは周辺ストリップ内の点が減っていく場合に限られる。ところがそのカウント数は約 70 点（$d = 1$）から約 31 点（$d = 4$）へと減った後、26 点付近で横ばいになる。$d = 16$ において $d = 1$ のときと同じくらい局所的なボックスを保つに

は $m/n = 0.05^{32}$ が必要となり、およそ 10^{42} 点ものサンプルが求められる計算になる。局所的な近傍を維持するために必要なサンプルサイズは、次元に対して指数関数的に増大するのである。

37.7.2.4 分布によらない推定の限界

もっと巧妙な推定量を使えば、このボトルネックを回避できるかもしれない。その望みを検証するため、厳密な測定よりも弱い条件である**下界**を求めてみよう。具体的には、関数 B（n 個のサンプルの関数）であって、$B \le I(X;Y)$ が確率 0.95 以上でペア (X,Y) の**あらゆる分布に対して**成り立つものを求める（$1/2$ より大きい任意の信頼水準（confidence level）でもまったく同様に成り立つ）。2 つのビューが多量の情報を共有していることに確信を持ちたい表現学習にとっても、求められているのはまさに下界である。

考え得る最も単純なケースで、すでに根本的な障壁が明らかになる。$Y = X$ とおき、X を離散変数とすれば、$I(X;Y) = H(X)$ となる。推定量が離散記号を固有の距離や順序を持たない任意のラベルとして扱うなら、**どの**記号が現れたかを利用することはできない。調べられるのは重複のパターン、すなわち 1 回だけ現れた異なる値がいくつあり、2 回現れたものがいくつあるか、といった情報だけである。この要約統計量はサンプルの**タイプ**と呼ばれ、ここでは B がタイプのみに依存すると仮定する。以下の短い 4 つのステップによって限界が示される（実質的な働きをするのは 3 番目のステップである）。

1. X を M 個の記号上の一様分布（uniform distribution）とし、n 個のサンプルを抽出する。任意の 1 組のサンプルが衝突する確率は $1/M$ であるため、ユニオンバウンドにより $n(n-1)/2$ 組のうち**少なくとも 1 組**が一致する確率は高々 $n(n-1)/(2M)$ である。
2. 0.95 の保証を、$M = n^2$ 個の記号上の一様分布（エントロピーは $2\log n$）に適用する。**この**分布から抽出されたサンプル上では、$B \le 2\log n$ が少なくとも確率 0.95 で成り立つ。
3. 第 1 ステップより、サイズ n のサンプルを $M = n^2$ 個の記号から抽出するとき、それが n 個のすべて異なる値からなる確率は $\frac{1}{2}$ より大きい。確率の和が 1 を超える 2 つの事象は必ず重なりを持つため、$B \le 2\log n$ を満たすすべて異なる値からなるサンプルが少なくとも 1 つ存在する。
4. n 個のすべて異なる値からなるサンプルはいずれもまったく同じタイプを持つため、実際にどの分布から生成されたかにかかわらず、**あらゆる**重複なしサンプルに対して B は同じ値（高々 $2\log n$）を返さなければならない。

ここで、この限界を $M = 2^{256}$ 個の記号上の一様分布（エントロピーは $256\ln 2 = 177$ nat）で試してみよう。$n = 10^6$ 個のサンプルでは、すべてのサンプルが互いに異なる確率は少なくとも $1 - 4.3\times10^{-66}$ であり、そのようなサンプルに対して B が保証できるのは高々 $2\log 10^6 = 27.6$ nat にすぎない。100 万個のサンプルがあっても 177 nat を保証することはできず、それどころか 30 nat すら保証できないのである。演習問題 4 では定数をさらに精密化し、どれだけのサンプル数が必要になるかを問う。

McAllester and Stratos (2020) は、推定量がサンプルのタイプのみに依存するという制限を設けずに一般の定理を証明した。

> **定理（McAllester–Stratos、非形式的）** 分布によらない $I(X;Y)$ の任意の下界であって、一定の信頼水準のもとで n 個のサンプルから成り立つものは $O(\log n)$ である。

この障壁の背後には相互情報量の不変性がある。X または Y にどのような可逆なラベルの付け替えを施しても $I(X;Y)$ は変化しないため（37.6.1.5 節）、座標について一切の仮定を置かない推定量はあらゆるラベルの付け替えに対して整合的に振る舞わなければならず、手がかりとして残るのは重複のパターンだけになる。$O(\log n)$ の壁を超えるには、データについて何らかの仮定を置かなければならない。KSG は座標間の距離に意味があり密度が滑らかであると仮定するが、その代償として有限サンプルでの保証を持たない推定値となる。37.7.3 節の変分下界は $\log n$ よ

り大きな数値を報告し得るが、実際にそのような値が出たときにどこまで信頼できるかについては 37.7.4.4 節で確認する。

37.7.2.5 圧縮フェーズをめぐる論争

いま見てきた 2 つの限界、すなわち座標に依存するヒストグラムと対数的なサンプル上限は、深層学習における有名な論争に光を当てる。Shwartz-Ziv and Tishby (2017) は、$4{,}096$ 通りの 2 進パターン（12 ビット）上で小さな分類器を訓練し、訓練を通じて各隠れ層 Z の活性化値（activation）をビン分割して、$I(X;Z)$ と $I(Y;Z)$ の推定値をプロットした（X は入力、Y はラベル）。著者らは、両方の量が上昇する短い**適合**（fitting）フェーズに続いて、$I(X;Z)$ が減少する一方で $I(Y;Z)$ が高いまま保たれる長い**圧縮**（compression）フェーズが観測されたとし、この圧縮こそが深層ネットワークが汎化（generalization）する理由を説明するものだと提唱した。その後、Saxe et al. (2018) は、この圧縮フェーズは tanh のように飽和する活性化関数では現れるものの、ReLU 活性化関数では消滅し、それでも ReLU ネットワークはまったく遜色なく汎化することを示した。

ヒストグラムは実際には何を測っていたのだろうか。決定論的なネットワークでは、ビン分割された活性化値 Z_{bin} は入力 X の決定論的な関数である。したがって $H(Z_{\text{bin}} \mid X) = 0$ となり、ビン分割による推定値は $I(X;Z_{\text{bin}}) = H(Z_{\text{bin}})$、すなわち単にビン分割された活性化値のエントロピーへと帰着する。この量は入力数の対数である $\ln 4096 = 8.3$ nat を決して超えることができず、異なる入力が同じヒストグラムのセルに潰れるたびに減少する。これは、訓練によって飽和する tanh ユニットが両端の 2 つのビン（± 1 付近）へと追い込まれるときにまさに起きる現象である。真の相互情報量 $I(X;Z)$ 自体は微動だにしない。異なる入力が異なる活性化値を生み出す限り、有限の入力集合に対しては $I(X;Z) = H(X)$ であり、連続入力に対しては $I(X;Z)$ は無限大である（37.6.1.3 節）。プロットされた曲線が追跡していたのは、固定されたビン上で活性化値がどのようにクラスタ化するかという推定量側の性質であり、ネットワークの真の相互情報量ではない（演習問題 8）。

37.7.3 変分下界

カウントに基づく手法では高次元における大きな相互情報量を測定できないため、変分法では目標そのものを切り替える。$\log \frac{p(x,y)}{p(x)\,p(y)}$ に現れる未知の密度を直接推定する代わりに、補助モデルを導入し、そのモデルがどれほど不完全であっても期待スコアが必ず $I(X;Y)$ の**下界**となるように構成するのである。その期待値を最大化するようにモデルを訓練すれば、$I(X;Y)$ を決して超えることなく下界を引き締められる。こうして推定は最適化（optimization）へと変わる。その出力には性質の異なる 2 種類の誤差が含まれる。最適モデルに届かない可能性のある補助モデル由来の近似誤差と、母集団の期待値を有限バッチの平均で置き換えなければならないために生じる標本誤差（sampling error）である。

ここでは、Poole et al. (2019) によって単一の枠組みに統一された 4 つの変分下界を順に調べる。1 つ目は正規化されたデコーダを用いる下界（Barber–Agakov）である。続く 2 つは非正規化のスコア関数を用い、Donsker–Varadhan（DV）下界と、Nguyen・Wainwright・Jordan（NWJ）の下界を与える。4 つ目の InfoNCE は独立した節を設けるに値するほど重要である。いずれの証明も、ただ 1 つの道具、すなわち KL ダイバージェンスは常に非負であるというギブスの不等式（Gibbs' inequality、37.1 節）に依拠している。

37.7.3.1 Barber–Agakov 下界

最も単純な変分下界は、未知の事後分布（posterior distribution）$p(x \mid y)$ を計算可能なモデル $q(x \mid y)$ に置き換えるものである。

命題（Barber–Agakov 下界） 任意の条件付き密度 $q(x \mid y)$ に対して、次が成り立つ。

$$I(X;Y) \geq H(X) + \mathbb{E}_{(x,y)\sim P_{X,Y}}\left[\log q(x \mid y)\right], \tag{37.75}$$

等号は $q(x \mid y) = p(x \mid y)$ のとき、かつそのときに限り成り立つ。連続な X に対しては、微分エントロピー（differential entropy）$h(X)$ が $H(X)$ を置き換える。

証明 $I(X;Y) = H(X) - H(X \mid Y)$ から出発し、Equation 37.75 の右辺を差し引くと次のようになる。

$$\begin{aligned} I(X;Y) - H(X) - \mathbb{E}\left[\log q(x \mid y)\right] &= \mathbb{E}\left[\log \frac{p(x \mid y)}{q(x \mid y)}\right] \\ &= \mathbb{E}_{y}\left[D_{\mathrm{KL}}(p(\cdot \mid y) \,\|\, q(\cdot \mid y))\right] \geq 0. \end{aligned}$$

不等号は、各 y に対してギブスの不等式を適用し、P_Y にわたって平均をとることから従う。等号はすべての y に対して KL ダイバージェンスが 0 になるとき、すなわち $q(x \mid y) = p(x \mid y)$ のとき、かつそのときに限り成り立つ。■

Barber–Agakov 下界 (Barber and Agakov 2003) は、エンコーダ・デコーダの情報理論的な解釈を与える。もしデコーダ $q(x \mid y)$ が低い交差エントロピーで X を Y から再構成できるなら、X と Y は多くの情報量を共有しているに違いない。実用上の制約は $H(X)$ の項であり、これは通常未知である。幸いにも $H(X)$ は q に依存しないため、q に関して下界を**最大化**する際にはこの値を知る必要が一切ない。$H(X) = -\mathbb{E}[\log p(x)]$ と書き直せば等価な比の形式 $I(X;Y) \geq \mathbb{E}\left[\log\left(q(x \mid y)/p(x)\right)\right]$ が得られ、これが InfoNCE 下界を証明する際の鍵となるステップになる。

本節を通じて、モデルはその下界の値を**保証する**（certify）と表現する。この保証は母集団の期待値のレベルで成り立つものである点に留意されたい。有限バッチの平均はその保証値を推定しているにすぎず、その上下に変動し得る。

標準的な分類器は身近な特殊ケースである。相互情報量の対称性により、X と Y を入れ替えて分類器 $q(y \mid x)$ をラベルのデコーダとして扱うことができる。これにより 37.6.2.4 節の損失による保証（loss certificate）が復元される。すなわち、サンプル数の均衡した k 個のクラスとホールドアウト交差エントロピー L に対して、$I(X;Y) \geq \log k - L$ が成り立つ。$L \geq 0$ であるため、10 クラスの分類器は $\ln 10 = 2.30$ nat を超える情報量を決して保証できない。InfoNCE もこれとまったく同じ形をとることになる。

ガウス分布のペアを見ると、不一致なデコーダによってどれほどの情報が失われるかがよくわかる。二乗誤差（squared error）のみで予測器を訓練することは、単位分散のガウスデコーダ $q(x \mid y) = \mathcal{N}(x; ay, 1)$ を用いることに対応する。$h(X) = \frac{1}{2}\log(2\pi e)$ であるから、Barber–Agakov 下界を計算すると $\frac{1}{2} - \frac{1}{2}\mathbb{E}[(X - aY)^2] = \frac{1}{2} - \frac{1}{2}(1 - 2a\rho + a^2)$ となり、$a = \rho$ で最大値 $\rho^2/2$ をとる。$\rho = 0.9$ のとき、このデコーダが保証するのは 0.41 nat（真の値は 0.83 nat）にとどまり、ρ がどれほど 1 に近づいても $\frac{1}{2}$ nat を超えて保証することは決してない。失われた情報量は、デコーダの分散を 1 に固定し、真の条件付き分散（conditional variance）$1 - \rho^2$ に合わせなかったことに対するペナルティである（演習問題 2）。

37.7.3.2 Donsker–Varadhan 下界

2 つ目の下界は、Barber–Agakov の 2 つの要件をともに取り除くものであり、$H(X)$ も正規化された密度 q も必要としない。相互情報量は同時分布から周辺分布の積への KL ダイバージェンス

であるため（37.6 節）、KL ダイバージェンスに対する任意の下界は直ちに $I(X;Y)$ の下界を与える。Donsker–Varadhan 下界は、任意のスコア関数 T（37.3.4 節と同様、**クリティック**と呼ばれる）を用いることで、正規化の制約（constraint）を一切課さずにこれを実現する。

命題（Donsker–Varadhan 下界） 分布 P、Q および任意の関数 T（ただし $\mathbb{E}_P[T]$ が定義され $\mathbb{E}_Q[e^T] < \infty$ を満たすもの）に対して、次が成り立つ。

$$D_{\mathrm{KL}}(P\|Q) \geq \mathbb{E}_P[T] - \log \mathbb{E}_Q\left[e^T\right], \tag{37.76}$$

等号は $T = \log \frac{dP}{dQ} + c$（$c$ はある定数）のとき、かつそのときに限り成り立つ。$P \ll Q$ かつダイバージェンスが有限のとき、T にわたる上限は $D_{\mathrm{KL}}(P\|Q)$ を達成し、KL ダイバージェンスの Donsker–Varadhan 表現を与える (Donsker and Varadhan 1983)。

証明 P が Q に関して絶対連続でない場合、左辺は無限大となり不等式は自明に成り立つ。そうでない場合、クリティックを用いて Q に重み付けを施す。すなわち、**傾斜分布**（tilted distribution）$\tilde{Q}$ を密度 $\tilde{q}(x) = q(x)\, e^{T(x)}/Z$ によって定義する。ここで、$Z = \mathbb{E}_Q[e^T]$ は正規化定数（normalizing constant）である。このとき、次の式が成り立つ。

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) - (\mathbb{E}_P[T] - \log Z) &= \mathbb{E}_P\left[\log \frac{p}{q} - T + \log Z\right] \\ &= \mathbb{E}_P\left[\log \frac{p}{\tilde{q}}\right] = D_{\mathrm{KL}}(P\|\tilde{Q}) \geq 0 \end{aligned}$$

最後の不等号はギブスの不等式による。等号は $P = \tilde{Q}$、すなわち $p/q = e^T/Z$、言い換えれば $T = \log(p/q) + \log Z$ のとき、かつそのときに限り成り立つ。■

この下界の緩みは単に $D_{\mathrm{KL}}(P\|\tilde{Q})$ であるため、クリティックが正確であることは、Q を e^T で傾斜させたときに P が再現されることとちょうど同値である。また、T を任意の定数 c だけシフトしても 2 つの項の間で相殺されることにも注目したい。$P = P_{X,Y}$ および $Q = P_X \otimes P_Y$ とおけば、任意のクリティック $T(x,y)$ に対する相互情報量の下界が次のように得られる。

$$I(X;Y) \geq \mathbb{E}_{P_{X,Y}}[T] - \log \mathbb{E}_{P_X \otimes P_Y}\left[e^T\right],$$

その最適なクリティックは、任意の加法定数の違いを除いて $\mathrm{pmi}(x,y)$ である。実際には、第 1 の期待値は観測されたペアにわたって T を平均することで推定し、第 2 の期待値は同じミニバッチから作った非対応ペアにわたって e^T を平均することで推定する。T をニューラルネットワークでパラメータ化すると **MINE**（mutual information neural estimation）が得られる (Belghazi et al. 2018)。

37.7.3.3 NWJ 下界

Donsker–Varadhan 下界は期待値の意味ではタイトであるが、$\mathbb{E}_Q[e^T]$ の外側にある対数がミニバッチ上で厄介な統計的バイアスを生む。log は凹関数であるため、イェンセンの不等式より、バッチ平均の対数は平均的に真の期待値の対数より小さくなる。より小さな数を差し引くことになるため、バッチ推定値 $\overline{T} - \log \overline{e^T}$ は平均的に**上方向**へと押し上げられるが、これは下界の推定量としては望ましくない性質である。

初等的な接線不等式 $\log z \leq z/e$（すべての $z > 0$ で成り立ち、$z = e$ で等号成立）を使えば、この対数を取り除くことができる。これを $Z = \mathbb{E}_Q[e^T]$ に対して Equation 37.76 に適用すると、すべてのクリティック T について次の式が得られる。

$$I(X;Y) \geq \mathbb{E}_{P_{X,Y}}[T] - e^{-1}\,\mathbb{E}_{P_X \otimes P_Y}\left[e^{T}\right]. \tag{37.77}$$

これが **NWJ 下界**である (Nguyen et al. 2010)。固定された任意のクリティック T に対して、NWJ が DV を上回ることは決してなく、両者が一致するのは $Z = e$ のときに限られる。この下界は $T^* = 1 + \text{pmi}$ においてタイトになる。DV がクリティックへの定数の加算に対して不変であるのに対し、NWJ はその定数を 1 に固定する。$e^{-1}\,\mathbb{E}_Q[e^T] = \mathbb{E}_Q[e^{T-1}]$ と書き直せば、NWJ は単に 37.3.4 節の一般の f ダイバージェンスの双対下界を KL ダイバージェンスに特殊化した式 $D_{\mathrm{KL}}(P\|Q) \geq \mathbb{E}_P[T] - \mathbb{E}_Q[e^{T-1}]$（同節では凸共役（convex conjugate）を $u\log u$ に対してとることで導出した）にほかならないことがわかる。KL ダイバージェンスに関しては、Donsker–Varadhan がその一般の下界を狭義に精密化しているのである。

第 2 の期待値を対数が囲んでいないため、ミニバッチ上で Equation 37.77 を評価すると下界の**不偏**（unbiased）推定値が得られる。しかし、不偏であることは信頼できることを意味しない。最適なクリティックにおいて、$e^{T^*-1} = e^{\text{pmi}}$ は密度比（density ratio）$p(x,y)/(p(x)p(y))$ である。独立なペアにわたるその期待値は 1 だが、X と Y が強く従属しているとき、その期待値は 0 に近い多数の値の中に混じるごくわずかな巨大値によって支配される。そうしたまれなスパイクをたまたま取り逃したバッチは、第 2 項を著しく過小評価し、結果として下界を大きく超えてしまう。この振る舞いについては 37.7.4.4 節で定量化し、実際に測定する。

37.7.4 InfoNCE

37.7.4.1 対照損失

DV 下界や NWJ 下界の指数項 e^T は正規化されていないため、極端なペアが 1 つあるだけでバッチ全体が支配されてしまう。InfoNCE はバッチの**内部で**正規化して候補どうしを競わせることで、この不安定性を回避する。具体的には、1 つの**正**のペア $(x, y^+) \sim P_{X,Y}$ をサンプリングし（x を**アンカー**と呼ぶ）、あわせて $N-1$ 個の**負例** $y_2, \ldots, y_N$ を周辺分布 P_Y から独立にサンプリングする。全 N 個の候補をクリティック $T(x,y)$ でスコア付けし、そのスコアをソフトマックスに通すことで、どの候補が真の相手であるかを予測する。その結果得られる交差エントロピーが **InfoNCE 損失** (Oord et al. 2018) であり、次の式で定義される。

$$\mathcal{L}_{\text{NCE}} = -\mathbb{E}\left[\log \frac{e^{T(x,y^+)}}{e^{T(x,y^+)} + \sum_{j=2}^{N} e^{T(x,y_j)}}\right], \tag{37.78}$$

ここで期待値は、正のペアと $N-1$ 個の負例の両方にわたってとる。対応する相互情報量の推定値は、この損失を $\log N$ から差し引いたものである。

$$\hat{I}_{\text{NCE}} = \log N - \mathcal{L}_{\text{NCE}}. \tag{37.79}$$

論文によっては分母を平均 $\frac{1}{N}\sum_{j=1}^{N} e^{T(x,y_j)}$ として書くこともあり、その場合は対数比の期待値がそのまま $\hat{I}_{\text{NCE}}$ に等しくなる。概念的には、$\mathcal{L}_{\text{NCE}}$ は固定されたラベルではなくバッチ内の他のデータ例をクラスとする単なる N クラス交差エントロピーにほかならない。これは CPC、SimCLR、CLIP の背後にある訓練の目的関数であり (Oord et al. 2018; Chen et al. 2020; Radford et al. 2021)、そこでは x と y が同一データ例の 2 つのビューまたは 2 つのモダリティを表し、バッチ内の残りのデータ例が負例の役割を果たす。

Figure 37.19 にこの設定を示す。非自明な最小のケースである $N = 2$ では、モデルが直面する負例は 1 つだけであり、損失は真の相手と 1 つのディストラクタを見分ける二値交差エントロピ

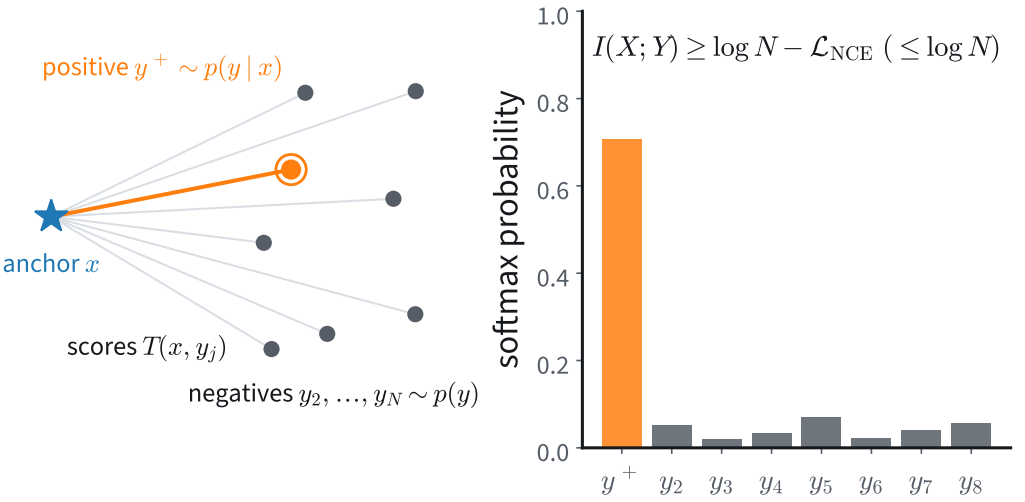


Figure 37.19: InfoNCE のタスク。左：アンカー x が、それと同時にサンプリングされた正例の相手 y^+、および周辺分布から独立にサンプリングされた $N-1$ 個の負例に対してクリティックによりスコア付けされる。各直線はスコア $T(x, y_j)$ を表す。右：N 個のスコアに対するソフトマックスが正例を識別しなければならない。得られる交差エントロピーは $I(X;Y) \geq \log N - \mathcal{L}_{\text{NCE}}$ を保証するが、この下界は決して $\log N$ を超えることができない。

ー（binary cross-entropy）に帰着する。このとき完璧なモデルであっても 2 つの選択肢から正しい方を選んだことしか示せないため、推定値は $\ln 2 = 0.69$ nat で頭打ちになる。

37.7.4.2 InfoNCE 下界

次の命題は、InfoNCE 損失の最小化が相互情報量の下界を保証することを確認するとともに、$\log N$ という上限がどこから生じるかを示すものである (Poole et al. 2019)。

命題（InfoNCE 下界）任意の可測なクリティック T および任意の $N \geq 1$ に対して、次が成り立つ。

$$I(X;Y) \geq \log N - \mathcal{L}_{\text{NCE}}. \tag{37.80}$$

さらに、右辺は高々 $\log N$ である。

証明候補に $y_1 = y^+, y_2, \ldots, y_N$ と番号を付け、単一の確率的なバッグ $V = (Y_1, \ldots, Y_N)$ にまとめる。ここで負例 $Y_{2:N}$ は (X, Y_1) と独立である。証明の方針は、Y をバッグ V に置き換え、条件付き分布（conditional distribution）として X が与えられたもとでのバッグ上の正規化された分布を構成し、Barber–Agakov 下界を適用することである。

ステップ 1：独立な負例は情報量を増やさない。相互情報量の連鎖律（37.6 節）より、$I(X; Y_1, Y_{2:N}) = I(X; Y_1) + I(X; Y_{2:N} \mid Y_1)$ が成り立つ。$Y_{2:N}$ は (X, Y_1) と独立であるため、X とも（Y_1 が与えられたもとで）独立のままである。したがって第 2 項は 0 となり、$I(X;V) = I(X;Y)$ が得られる。

ステップ 2：ソフトマックス比が正規化されたモデルを定義する。ここが鍵となる計算である。再スケーリングしたソフトマックススコアを次のように定義する。

$$\gamma(x, v) = \frac{e^{T(x, y_1)}}{\frac{1}{N}\sum_{j=1}^{N} e^{T(x, y_j)}},$$

これは $(0, N]$ に値をとる。$p(v)$ を、第 1 成分も含めて N 個すべてを P_Y から独立にサンプリングしたバッグの分布とする。全 N 個の成分は独立同分布であるため、それらを置換しても $p(v)$ は変化しない。したがって、全 N 個の比 $e^{T(x,Y_i)} / \left(\frac{1}{N}\sum_j e^{T(x,Y_j)}\right)$ は $p(v)$ のもとですべて同じ期待値を持ち、固定された任意の x に対して次の式が成り立つ。

$$\begin{aligned}\mathbb{E}_{p(v)}\left[\gamma(x, V)\right] &= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{p(v)}\left[\frac{e^{T(x,Y_i)}}{\frac{1}{N}\sum_j e^{T(x,Y_j)}}\right] \\ &= \mathbb{E}_{p(v)}\left[\frac{\frac{1}{N}\sum_i e^{T(x,Y_i)}}{\frac{1}{N}\sum_j e^{T(x,Y_j)}}\right] = 1.\end{aligned}$$

よって、$q(v \mid x) = p(v)\,\gamma(x, v)$ はバッグ上の有効な条件付き確率（conditional probability）分布となる。

ステップ 3：バッグに対して Barber–Agakov を適用する。 Y_1 は周辺分布 P_Y を持ち $Y_{2:N}$ と独立であるため、V の周辺分布も $p(v)$ である。Equation 37.75 の比の形式を、2 変数の役割を入れ替えて (X, V) に適用すると（I の対称性を用いる）、次の式が得られる。

$$I(X; V) \geq \mathbb{E}_{p(x,v)}\left[\log \frac{q(v \mid x)}{p(v)}\right] = \mathbb{E}\left[\log \gamma(X, V)\right] = \log N - \mathcal{L}_{\text{NCE}}.$$

最後の等号は、$\log \gamma$ が $\log N$ と正例に割り当てられた対数ソフトマックス確率との和に等しいことから成り立つ。これをステップ 1 と合わせれば下界が証明される。最後に、正例のソフトマックス確率は 1 を超えられないため $\mathcal{L}_{\text{NCE}} \geq 0$ であり、したがって右辺は高々 $\log N$ である。■

直観的には、N 個の候補から真の相手を選び出すことは、たとえ誤り率が 0 であっても高々 $\log N$ nat、すなわち N 個の選択肢からの 1 回の選択に含まれる情報量しか示さない。10 クラス分類器が $\ln 10$ nat を超える情報量を決して保証できなかったのと同様に、いかなるクリティックもそのタスクからそれ以上の情報量を絞り出すことはできない。N を増やせばこの上限が引き上げられるため、これが CPC、SimCLR、CLIP といった対照手法で大きなバッチを用いる動機の 1 つとして挙げられる。ただし、それが話のすべてではない。本節の最後で見るように、表現を優れたものにするのはよりタイトな下界ではないからである。なお、この損失は $T^*(x, y) = \text{pmi}(x, y) + c(x)$ という形の任意のクリティックによって最小化される。ここで $c(x)$ はアンカーのみに依存する任意の関数である（演習問題 5）。

37.7.4.3 厳密なクリティックのもとでの上限

厳密なクリティック $T = \text{pmi}$ を用いれば、$\hat{I}_{\text{NCE}}$ は真の $I(X; Y)$ に沿って $\log N$ までまっすぐ追随し、そこで鋭く折れ曲がって平坦になると考えるのが自然だろう。しかし $T = \text{pmi}$ を Equation 37.79 に代入すると、より滑らかで、かつ厳しい現実を突きつける振る舞いが明らかになる。

$$\hat{I}_{\text{NCE}} = I(X; Y) - \mathbb{E}\left[\log \frac{1}{N}\sum_{j=1}^{N} e^{\text{pmi}(x, y_j)}\right]. \tag{37.81}$$

推定値は、真の相互情報量から緩みの項（命題により非負）を引いたものに等しい。簡単な概算によって、この緩みの大きさを見積もることができる。対数の内側には N 個の項の平均がある。そのうち $N-1$ 個の負例は x と独立であるためそれぞれの平均は 1 であり（本節の冒頭で確かめた第 2 の恒等式）、平均全体に対しておおよそ 1 だけ寄与する。一方、正例は $e^{\mathrm{pmi}(x,y^+)}/N$ だけ寄与し、典型的なペアに対してこれはおおよそ e^I/N となる。確率的な平均をこの典型値で置き換えると、次の近似が得られる。

$$\hat{I}_{\mathrm{NCE}} \approx I - \log\left(1 + \frac{e^I}{N}\right) = -\log\left(e^{-I} + \frac{1}{N}\right). \tag{37.82}$$

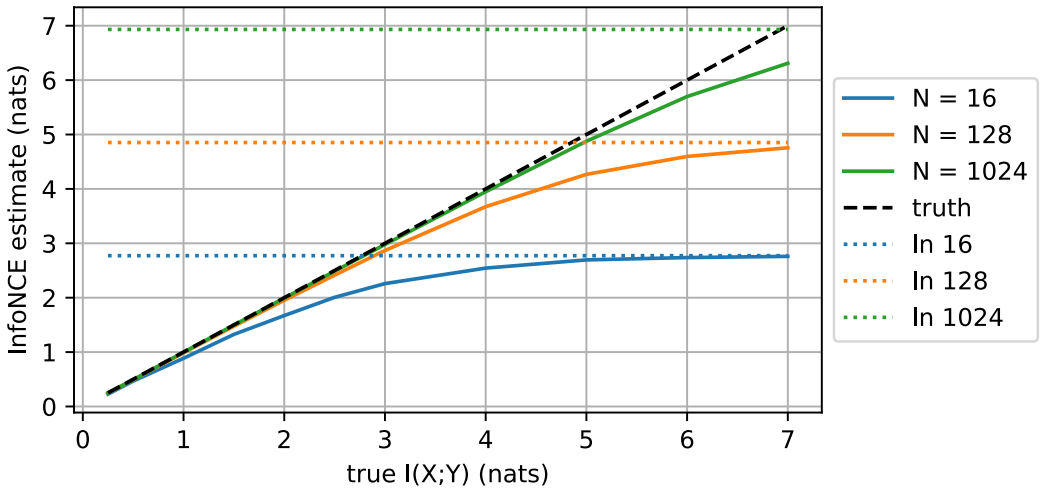
右辺は I と $\log N$ の滑らかな最小値である。これは I に密接に追随し（$e^I \ll N$ のとき）、$\log N$ に漸近し（$e^I \gg N$ のとき）、遷移領域ではその両方よりかなり下へと曲がり込む。ちょうど $I = \log N$ の地点では $\log N - \log 2$ となり、0.69 nat 低くなる。最初の予想が見落としていたのは、平均が 1 となる負例とともに正例自身が分母の中に入っているという事実である。正例が分母を支配するようになるのは、e^I が N を大きく上回ってからにすぎない。

裾の重い平均をその期待値で置き換えるのは大雑把な近似にすぎないため、Equation 37.82 を実験的に確かめておくべきだろう。$N = 128$ かつ $I = 4$ のとき、この式は 3.6 nat と予測する（上限 $\ln 128 = 4.85$ にはまだ十分な余裕があるように見えるにもかかわらずである）。以下では、バッチ内の全 $N \times N$ 個のペアを同時にスコア付けする。第 i 行はアンカー x_i をすべての y_j と組にするため、その対角成分が正例となり、$N-1$ 個の非対角成分が負例となる。各行は候補を共有しているものの、下界は依然として有効である（演習問題 6）。$I(X;Y)$ を変化させながら、$N = 16$、128、1024（上限はそれぞれ 2.77、4.85、6.93 nat）に対する経験的な推定値を真の値と比較してみよう。

```
def infonce_bound(f):
    """log N minus the InfoNCE loss of an N x N score matrix f[i, j]."""
    return np.log(len(f)) + np.mean(np.diag(f) - logsumexp(f, axis=1))

def exact_scores(x, y, rho):
    """The exact critic T = pmi on all N x N pairs (x_i, y_j) of a batch."""
    return pmi_gauss(x[:, None], y[None, :], rho)

true_mi = np.array([0.25, 0.5, 1, 1.5, 2, 2.5, 3, 4, 5, 6, 7])
rhos = np.sqrt(1 - np.exp(-2 * true_mi))                # invert mi_gauss
Ns = (16, 128, 1024)
ests = np.array([[np.mean([infonce_bound(exact_scores(*gauss_pair(rng, r, N), r))
                           for _ in range(200 if N <= 128 else 50)])
                  for r in rhos] for N in Ns])              # (N, level)
ceilings = [[np.log(N)] * len(true_mi) for N in Ns]
d2l.plot(true_mi, [*ests, true_mi, *ceilings], 'true I(X;Y) (nats)',
         'InfoNCE estimate (nats)',
         legend=[f'N = {N}' for N in Ns] + ['truth'] + [f'ln {N}' for N in Ns],
         fmts=('C0-', 'C1-', 'C2-', 'k--', 'C0:', 'C1:', 'C2:'),
         figsize=(5.5, 3), legend_loc='outside')
for I in (1, 4, 7):
    k = list(true_mi).index(I)
    print(f'I = {I}:  ' + '   '.join(f'N = {N}: {e:.2f}'
                                      for N, e in zip(Ns, ests[:, k])))
```

```
I = 1:  N = 16: 0.89   N = 128: 0.99   N = 1024: 1.00
```

```
I = 4:  N = 16: 2.54   N = 128: 3.67   N = 1024: 3.95
I = 7:  N = 16: 2.76   N = 128: 4.76   N = 1024: 6.31
```



どの曲線にも鋭い折れ曲がりはない。いずれも早い段階で対角線から離れ、それぞれの上限に向かって滑らかに曲がっており、出力された 9 個の値すべてが 0.1 nat 以内の誤差で Equation 37.82 に一致している。$I = 1$ では、どのバッチサイズでも真の値の約 0.1 nat 以内に収まる。$I = 4$ になる頃には、サイズ 128 のバッチが報告する値は 3.7 nat にとどまり、$N = 1024$ でさえわずかに遅れをとり始める。$I = 7$（上限 6.93 をわずかに超えた地点）では、$N = 1024$ の報告値は 6.3 にすぎない。

なぜこうなるかは、2 つの厳密な事実から説明できる。第 1 に、対数の内側の平均は少なくとも正例自身の項 $e^{\mathrm{pmi}(x,y^+)}/N$ 以上であるため、緩みは少なくとも $\mathbb{E}[\mathrm{pmi}(x,y^+)] - \log N = I - \log N$ となり、ここから $\log N$ の上限が復元される。第 2 に、この緩みが消えるのは極限 $N \to \infty$ においてのみである（演習問題 5）。厳密な上限 $\log N$ は、この有限バッチによる緩みの最も大雑把な兆候にすぎない。

この滑らかな圧縮により、推定値から真の値を逆算することも難しくなる。$N = 128$ のとき、測定値 3.7 は真の 4 nat に対応し、4.75 付近の測定値は 7 nat に対応するため、4 nat から無限大までのあらゆる真の値が、目盛りの上端わずか 1.2 nat の幅に押し込められてしまう。

画像分類（image classification）を例にとると、これらの数値の意味が実感できる。ある画像の 2 つのビューが、それぞれ均衡した 1,000 個のカテゴリの中からそのクラスを一意に決定するとしよう。ラベルはどちらのビューの関数でもあるため、データ処理不等式（37.6.2.2 節）より、2 つのビューは少なくともラベルのエントロピーである $\ln 1000 = 6.91$ nat を共有する。候補数 $N = 128$ の InfoNCE 損失ではそれに近い値すら決して保証できず、$N = 1024$ を用いても推定値は 6.3 にしかならない（真の共有情報量は 7 であるにもかかわらず）。2 つのビューが細粒度分類に十分な情報を保持しているかどうかは、このような数値からは到底読み取れないのである。

37.7.4.4 厳密なクリティックによる比較

これで DV、NWJ、InfoNCE を同じ土俵で比較できる。バッチサイズを $N = 128$ とし、いずれにも厳密な最適クリティック $T^* = 1 + \mathrm{pmi}$ を与える（定数を加えても InfoNCE は変化しない）。DV と NWJ は独立なペアをバッチ内の N 個の非対応ペアから取り出すが、先ほど指摘した裾の重い振る舞いを踏まえれば、コードを 1 行も実行する前に何が起きるかを予測できる。

典型的な同時ペアの pmi は I に近い。直積分布 $P_X \otimes P_Y$ のもとでは、そのようなペアは確率お

およそ e^{-I} で生じる。なぜなら $p(x)\,p(y) = p(x,y)\,e^{-\mathrm{pmi}}$ だからである。言い換えれば、非対応ペアのうちおおよそ e^{I} 個に 1 個が、本物のペアで通るほどもっともらしい見た目をしている。こうした**希少ペア**（rare pairs）はそれぞれ巨大な重み $e^{\mathrm{pmi}} \approx e^{I}$ を持ち、それら全体で期待値 $\mathbb{E}[e^{\mathrm{pmi}}] = 1$ のすべてを担っている。サイズ N のバッチには平均しておおよそ Ne^{-I} 個の希少ペアが含まれる。その数は約 6 個（$I = 3$）、2 個（$I = 4$）、そしてわずか 0.3 個（$I = 6$）となる。

バッチ内に希少ペアが複数含まれている限り、e^{pmi} の標本平均は 1 付近にとどまり、3 つの下界はすべて一致するはずである。しかし期待個数 Ne^{-I} が 1 を下回ると、大半のバッチには希少ペアが 1 つも含まれなくなる。このようなバッチを**枯渇**（starved）バッチと呼ぼう。枯渇バッチでは e^{pmi} の平均が 0 に近くなるため、NWJ は T^{*} のバッチ平均からほぼ何も引かない値（約 $I + 1$ であり、1 nat 高すぎる）を報告し、DV に至っては極小の数の対数を引くことで何 nat も過大に飛び出してしまう。

NWJ は不偏であるため、そうした枯渇バッチと釣り合いをとる何かが存在するはずである。その釣り合いをとるのが**ヒット**バッチ、すなわちたまたま $e^{\mathrm{pmi}} \geq N$ となる非対応ペアを含んだバッチである。そのようなペアが 1 つあるだけで、バッチ平均には少なくとも 1（e^{pmi} の平均への寄与）が加わり、NWJ を少なくとも 1 nat 引き下げて I 付近、ときにはそれよりはるかに低い値へと落とす。$I = 6$ では希少ペアは $e^{\mathrm{pmi}} \approx e^{6} = 403 > N$ を持つため、それを 1 つでも含むバッチはすべてヒットバッチになる。バッチあたりの希少ペアが平均 0.3 個であれば、およそ 4 バッチに 1 つがヒットバッチとなり、残りの大半は枯渇バッチになるはずである。

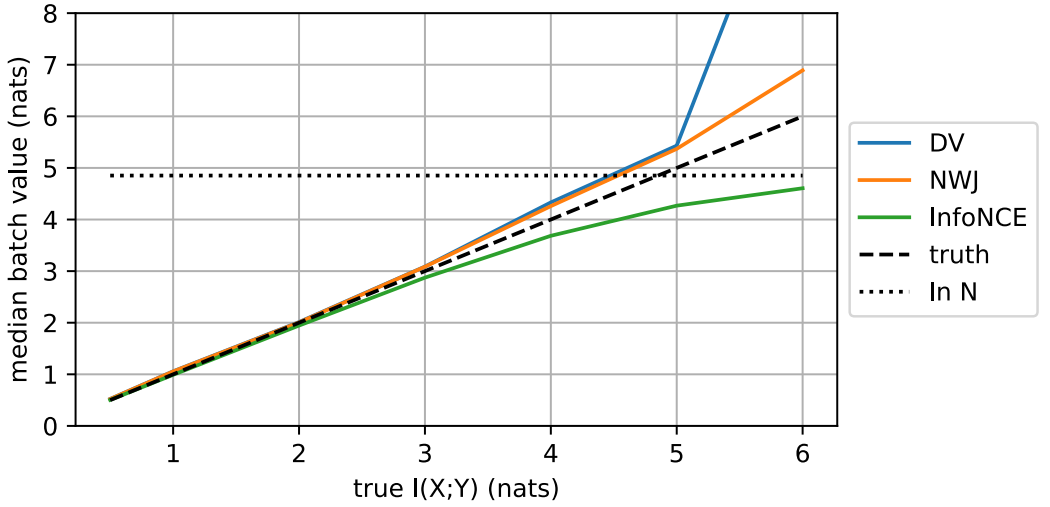
InfoNCE はこれとはまったく異なる振る舞いを見せる。バッチ内で正規化するため、もっともらしい負例が現れてもソフトマックスの高々 1 行にしか影響しない。そのため、Equation 37.82 が予測するとおりの地点、すなわち約 4.6 nat（$I = 6$ のとき）で安定するはずである。切り替わりの境界 $Ne^{-I} \approx 1$ が $I \approx \ln N$ で生じることに注目したい。バッチサイズの対数は、InfoNCE を制限するのとまったく同じだけ DV や NWJ も制限しているのであり、DV と NWJ は目に見える上限の代わりに爆発する分散の陰にその限界を隠しているにすぎない。

200 個のバッチを各 I の水準でサンプリングし、これらの予測を検証してみよう。裾の重い分布は標本平均（sample mean）や標準偏差を歪めるため、各下界について中央値（median）と四分位範囲を報告し、NWJ については標本平均も併記する。最後の 2 列は枯渇バッチ（平均が $e^{\mathrm{pmi}} < 0.1$）の割合とヒットバッチ（$\max e^{\mathrm{pmi}} \geq N$）の割合を記録し、プロットは 3 つの下界の中央値を比較している。

```
def quartiles(a):
    """Median and quartiles of a list of batch values, as text."""
    q1, med, q3 = np.percentile(a, [25, 50, 75])
    return f'{med:5.2f} [{q1:5.2f}, {q3:5.2f}]'

N, true_mi = 128, np.array([0.5, 1, 2, 3, 4, 5, 6])
medians = []
# Last two columns: share of batches whose mean e^pmi over mismatched pairs
# is below 0.1, and share holding a mismatched pair with e^pmi >= N.
print(f'{"I":>3}  {"DV median [IQR]":20}  {"NWJ median [IQR]":20}  '
      f'{"NWJ mean":>8}  {"InfoNCE median [IQR]":20}  {"<0.1":>4}  {">=N":>4}')
for I in true_mi:
    rho, batches = np.sqrt(1 - np.exp(-2 * I)), []
    for _ in range(200):
        x, y = gauss_pair(rng, rho, N)
        T_joint = 1 + pmi_gauss(x, y, rho)              # T* on true pairs
        T_prod = 1 + pmi_gauss(x, np.roll(y, 1), rho)   # T* on mismatched
        e_pmi = np.exp(T_prod - 1)                      # e^pmi, mismatched
```

```
            batches.append((T_joint.mean() - (logsumexp(T_prod) - np.log(N)),  # DV
                            T_joint.mean() - e_pmi.mean(),                     # NWJ
                            infonce_bound(exact_scores(x, y, rho)),
                            e_pmi.mean() < 0.1, e_pmi.max() >= N))
        dv, nwj, nce, low, rare = np.array(batches).T
        medians.append([np.median(a) for a in (dv, nwj, nce)])
        print(f'{I:3.1f}  {quartiles(dv)}  {quartiles(nwj)}  {nwj.mean():8.2f}'
              f'  {quartiles(nce)}  {low.mean():4.0%}  {rare.mean():4.0%}')
    d2l.plot(true_mi, [*np.array(medians).T, true_mi, [np.log(N)] * len(true_mi)],
             'true I(X;Y) (nats)', 'median batch value (nats)',
             legend=['DV', 'NWJ', 'InfoNCE', 'truth', 'ln N'], ylim=[0, 8],
             fmts=('C0-', 'C1-', 'C2-', 'k--', 'k:'), figsize=(5.5, 3),
             legend_loc='outside')
```

```
  I  DV median [IQR]       NWJ median [IQR]      NWJ mean  InfoNCE median [IQR]  <0.1   >=N
0.5   0.52 [ 0.43,  0.61]   0.52 [ 0.43,  0.60]      0.52   0.50 [ 0.45,  0.55]    0%    0%
1.0   1.06 [ 0.90,  1.16]   1.06 [ 0.89,  1.15]      1.01   0.99 [ 0.94,  1.04]    0%    0%
2.0   2.01 [ 1.83,  2.21]   2.01 [ 1.82,  2.19]      1.99   1.95 [ 1.89,  2.00]    0%    0%
3.0   3.09 [ 2.81,  3.43]   3.09 [ 2.79,  3.35]      2.96   2.87 [ 2.82,  2.92]    0%    4%
4.0   4.33 [ 3.69,  4.91]   4.26 [ 3.65,  4.64]      4.08   3.68 [ 3.63,  3.74]    9%    5%
5.0   5.43 [ 4.64,  9.74]   5.37 [ 4.57,  5.92]      4.97   4.27 [ 4.23,  4.32]   36%   32%
6.0  11.71 [ 5.53, 30.67]   6.89 [ 5.40,  7.01]      5.89   4.61 [ 4.57,  4.63]   57%   30%
```



$I = 3$ までは、3 つの中央値はいずれも真の値の約 0.2 nat 以内に収まり、枯渇バッチもヒットバッチもほとんど現れず、ばらつきが最も小さいのは InfoNCE である。そこを超えると、予測どおりに枯渇バッチが現れ始める。その割合は高々約 10 分の 1（$I = 4$）、約 3 分の 1（$I = 5$）、そして半数以上（$I = 6$）へと増えていく。

$I = 6$ の行では、NWJ の中央値は約 $7 = I + 1$ まで上昇し、上側四分位数もそのすぐ上に重なっている。つまり全バッチの半数が丸々 1 nat ほど過大推定しているのである。全体の 4 分の 1 前後を占めるヒットバッチが標本平均を引き戻し、6 から約 0.5 nat 以内に収めている。DV はさらに深刻であり、中央値は真の値より数 nat 高く、上側四分位数は数十 nat に達して曲線がプロットの上端から突き抜けてしまっている。これとは対照的に、InfoNCE は 4.6 nat で盤石に安定しており、四分位範囲も 0.1 未満である。

NWJ のばらつきは解析的にも計算できる。その非対応ペアの項は e^{pmi} のバッチ平均であり、分散は $(e^{2I}-1)/N$ となる。$I=6$ かつ $N=128$ では標準偏差が 36 nat にも達するが、これは 200 回の試行ではめったに含まれないほどまれな極端なヒットバッチによって引き起こされている（演習問題 1）。

以上のことは、母集団の期待値に対して成り立つ変分不等式と何ら矛盾するものではない。むしろ、変分下界が 37.7.2.4 節の分布によらない限界とどのように整合するかを示している。InfoNCE は $\log N$（ただし N は高々サンプルサイズ）を超えることができないため、構成上おのずと対数的な限界に従う。DV と NWJ はより大きな数値を自由に報告できるが、そうした値を報告するときには、そのバッチ推定値はもはや信頼できる下界ではなくなっているのである。

37.7.4.5 クリティックの訓練

実際の応用では真の pmi は未知であるため、クリティックをデータから学習しなければならない。対照損失が真の相互情報量より小さな値を報告したとき、その不足分が不完全なクリティックに由来するのか、それとも下界自体の緩みに由来するのかはどう見分ければよいだろうか。これに答えるため、CLIP や SimCLR で使われているものとまったく同じ形式のデュアルエンコーダ型クリティックを訓練してみよう。すなわち、2 つの小さなネットワーク $g, h : \mathbb{R} \to \mathbb{R}^{16}$ が x と y を埋め込み（embedding）へと写し、その内積（inner product）$T(x,y) = g(x)^{\top} h(y)$ をクリティックとする。するとバッチ内の全 $N \times N$ 個のスコアは 2 つの埋め込み行列の行列積（matrix multiplication）をたった 1 回計算するだけで求まり、ネットワークの評価回数を N^2 回から $2N$ 回へと削減できる。ここでは互いに素な 4,096 ペアのプールを $\rho = 0.99$（真の情報量は 1.96 nat）で 2 つ生成し、一方を訓練用、他方を評価用とする。

```
rho, n_pool = 0.99, 4096
x_tr, y_tr = (a.astype(np.float32) for a in gauss_pair(rng, rho, n_pool))
x_ev, y_ev = (a.astype(np.float32) for a in gauss_pair(rng, rho, n_pool))
```

各エンコーダは、64 個の ReLU ユニットからなる隠れ層を 2 つ持ち、Glorot 一様初期化（initialization）を用いた多層パーセプトロン（multilayer perceptron、MLP）である。得られる $N \times N$ のスコア行列上では、InfoNCE 損失は単なる標準的な多クラス交差エントロピーであり、第 i 行に対する正解クラスは対角成分の正のペアのインデックスである i となる。

```
def init_encoder(key, sizes=(1, 64, 64, 16)):
    """Glorot-uniform weights and zero biases of an MLP with these widths."""
    glorot = jax.nn.initializers.glorot_uniform()
    keys = jax.random.split(key, len(sizes) - 1)
    return [(glorot(k, (a, b)), jnp.zeros(b))
            for k, a, b in zip(keys, sizes[:-1], sizes[1:])]

def encode(params, x):
    """Embed each scalar of x with the MLP."""
    z = x[:, None]
    for W, b in params[:-1]:
        z = jax.nn.relu(z @ W + b)
    W, b = params[-1]
    return z @ W + b

def scores(params, x, y):
    """All N x N critic values T(x_i, y_j) = g(x_i) . h(y_j)."""
    params_g, params_h = params
```

```
    return encode(params_g, x) @ encode(params_h, y).T

def infonce_loss(params, x, y):
    """Cross-entropy of each row's softmax at its positive, the diagonal."""
    f = scores(params, x, y)
    return optax.softmax_cross_entropy_with_integer_labels(
        f, jnp.arange(len(f))).mean()

key_g, key_h = jax.random.split(jax.random.key(int(rng.integers(2**31))))
params = (init_encoder(key_g), init_encoder(key_h))
```

訓練プールからサンプリングしたペアのミニバッチを用いて、1,000 ステップの Adam（学習率 10^{-3}、バッチサイズ $N = 128$）で訓練し、訓練済みモデルを `critic_np` としてラップして `exact_scores` と同様に NumPy 配列を受け取れるようにする。

```
opt = optax.adam(1e-3)
state = opt.init(params)

@jax.jit
def train_step(params, state, x, y):
    grads = jax.grad(infonce_loss)(params, x, y)
    updates, state = opt.update(grads, state)
    return optax.apply_updates(params, updates), state

N = 128
for step in range(1000):
    idx = rng.choice(n_pool, N, replace=False)
    params, state = train_step(params, state, x_tr[idx], y_tr[idx])

def critic_np(a, b):
    """The trained critic on NumPy inputs, as an N x N NumPy array."""
    return np.asarray(scores(params, a, b))
```

クリティックの誤差と下界の緩みをきれいに切り分けるため、学習されたクリティックと厳密なクリティックの両方を**同じ**ホールドアウトバッチ上で評価する。以下では、評価プールをサイズ $N \in \{2, 8, 32, 128\}$ の互いに素なバッチに分割し、それらにわたって InfoNCE 下界の平均をとる。なお、$N = 128$ で訓練したクリティックを他の任意のバッチサイズで評価できるのは、最適なクリティック $\mathrm{pmi}(x, y) + c(x)$ が N に依存しないからである。

```
def infonce_eval(score, x, y, N):
    """Mean InfoNCE bound over the disjoint batches of N pairs in (x, y)."""
    return np.mean([infonce_bound(score(x[i:i + N], y[i:i + N]))
                    for i in range(0, len(x), N)])

exact_np = lambda a, b: exact_scores(a, b, rho)   # the exact critic T = pmi

print('   N   ln N   learned   exact')
for N_ev in (2, 8, 32, 128):
    print(f'{N_ev:4d}  {np.log(N_ev):5.2f}  '
          f'{infonce_eval(critic_np, x_ev, y_ev, N_ev):7.2f}  '
          f'{infonce_eval(exact_np, x_ev, y_ev, N_ev):6.2f}')
```

```
print(f'true I(X;Y) = {mi_gauss(rho):.2f} nats')
```

```
   N   ln N   learned   exact
   2   0.69     0.57    0.58
   8   2.08     1.42    1.42
  32   3.47     1.77    1.78
 128   4.85     1.89    1.90
true I(X;Y) = 1.96 nats
```

どのバッチサイズにおいても、学習されたクリティックは厳密なクリティックと 100 分の数 nat 以内で一致している。したがって、真の値 1.96 nat との残りのギャップはすべて下界そのものに属しており、N が増えるにつれて徐々に縮まり、両方のクリティックが約 1.9 nat（$N = 128$ のとき）に達する。$N = 8$ の行を詳しく見てみよう。上限 $\ln 8 = 2.08$ はすでに真の値 1.96 を上回っているため、約 0.55 nat の不足は上限に突き当たったせいではない。これは下界の滑らかな最小値による緩みであり、Equation 37.82 が約 0.6 nat と予測する量である。もちろん、スカラーのガウス分布ペアはクリティックにとって容易なタスクであり、高次元ではクリティックの近似誤差は無視できなくなるうえ、既知の正解（ground truth）がなければ 2 つの誤差要因を切り分けることはできない。

SimCLR などの対照システムでは、クリティックにわずかな変更を加える。埋め込み z_x と z_y を単位長に正規化し、そのコサイン類似度（cosine similarity）を**温度**（ソフトマックス温度、softmax temperature）τ で割って $T(x, y) = \mathrm{sim}(z_x, z_y)/\tau$ とおくのである (Chen et al. 2020)。温度はソフトマックスがどれだけ鋭く候補を分離するかを制御すると同時に、それ自体が第 2 の上限を導入する。コサイン類似度は $[-1, 1]$ に収まるため、どの 2 つのスコアの差も $2/\tau$ を超えることはなく、したがってコサイン型クリティックは $2/\tau$ nat 以上を決して保証できない（N がどれほど大きくても同様である。演習問題 3)。このため下界は 4 nat 未満（$\tau = 0.5$ のとき）や 20 nat 未満（$\tau = 0.1$ のとき）に制限される。ゆえに温度が大きいと、バッチサイズが制約になるよりずっと手前で下界が頭打ちになってしまう。

実用上はどの推定量を選ぶべきだろうか。高々数 nat を共有する 1 次元または 2 次元変数のペアには、最近傍カウントが適している。その領域では KSG が真の値の約 0.1 nat 以内に収まっていた。一方、高次元ベクトルに対しては、本節のどの推定量も大きな相互情報量を正確には測れないため、問いは 2 つのビューが**どれだけ**の情報を共有するかから、**少なくともどれだけ**共有するかへと移る。InfoNCE はその問いに高い信頼性で答えてくれる。メモリが許す限り多くの負例を用いて訓練し、$\log N - \mathcal{L}_{\mathrm{NCE}}$ を厳密に下界として解釈すればよい。$\log N$ から 1 nat 前後以内の測定値が得られた場合、そこからわかるのは両ビューが少なくともそれだけの情報を共有しているということだけであり、逆に低い測定値は単にクリティックの弱さを反映している可能性がある。DV と NWJ については、独自の得意領域が存在しない。バッチ推定値が信頼できる $\log N$ より数 nat 低い領域では、InfoNCE もほぼ同等の精度をはるかに高い安定性で達成し、それより高い領域では典型的なミニバッチが過大推定を起こすからである。

最後に、よりタイトな下界が自動的により優れた表現を生むわけではない。Tschannen et al. (2020) は、下流タスク（downstream task）の性能が、最大化する相互情報量の下界のタイトさとほとんど相関しないことを示した。むしろ緩い下界の方が強力な表現をもたらすことも多く、エンコーダやクリティックのアーキテクチャの選択は推定される情報量と同等以上に重要である。こうして 2 つの根本的な問いが未解決のまま残されている。高次元において大きな相互情報量をどうすれば信頼性高く推定できるのか、そして緩い変分下界を最大化することでなぜこれほど有効な表現が得られるのか、という問いである。

37.7.5 まとめ

有限のサンプルだけから相互情報量を直接測定できるのは、変数が低次元で共有情報量が控えめな場合に限られる。その領域を超えると、あらゆる推定量は識別可能な結果の数の対数によって制約される。すなわち、ヒストグラムでは $\log b$（ビン数が b 個のとき）、KSG の最近傍カウントではおおよそ $\log(n/m)$（しかも高次元ではその上限よりはるかに低いところで飽和する）、そして分布によらない任意の下界では $O(\log n)$ となる。変分法は、デコーダまたはクリティックでパラメータ化された下界を最大化することで密度推定を回避する。対照学習の交差エントロピー目的関数である InfoNCE は低分散で扱いやすく、厳密なクリティックのもとでおおよそ $-\log(e^{-I}+1/N)$ を与え、決して $\log N$（候補数 N）を超えることがない。Donsker–Varadhan と NWJ には厳密な上限はないものの、そのミニバッチ推定値が信頼できるのは $\log N$ より 2 nat ほど低い領域までに限られる。ガウス分布のベンチマークでは、学習されたデュアルエンコーダクリティックが厳密な pmi クリティックと 100 分の数 nat 以内で一致し、観測されたギャップが下界そのものに由来することが確認された。とはいえ、情報量の下界のタイトさそれ自体が、学習された表現の質を予測するわけではない。

37.7.6 演習問題

1. **指数をとった pmi の重い裾。●**(X,Y) を相関 ρ、$I=-\tfrac{1}{2}\log(1-\rho^2)$ を持つガウス分布のペアとし、$Q=P_X\otimes P_Y$ とする。Q のもとでは e^{pmi} の平均は 1 であるが（任意の ρ に対して成り立つ）、典型的な独立ペアの e^{pmi} は 0 の近くになる（ρ が 1 に近いとき）。
   1. $\mathbb{E}_Q[e^{2\,\mathrm{pmi}}]=\int p^2/q = 1/(1-\rho^2)=e^{2I}$ であり、したがって e^{pmi} が分散 $e^{2I}-1$ を Q のもとで持つことを示せ。**ヒント：**被積分関数は精度行列 $2\Sigma^{-1}-\mathbf{I}$ を持つガウス密度に比例する。ここで Σ はペアの共分散行列であり、$\det(2\Sigma^{-1}-\mathbf{I})=1$ が成り立つ。
   2. バッチにランダムな置換を施すと、平均して 1 つの y が元の対応する x と組のまま残る。このような不動点が置換ペアにわたるバッチ平均に e^{pmi}/N（すなわち e^{pmi} の平均への寄与）を加え、その期待値が e^{2I}/N（すなわち 0.43、23、1270。それぞれ $I=2,4,6$ かつ $N=128$ のとき）になることを示せ。なぜ `np.roll` はこの問題を回避できるのか。
   3. NWJ クリティック $T^*=1+\mathrm{pmi}$ に対し、非対応ペアが独立なサンプルであるときバッチ推定値の平均が I となり、その非対応ペアの項の分散が $(e^{2I}-1)/N$（標準偏差はそれぞれ 0.65、4.8、36 nat。ただし $I=2,4,6$ かつ $N=128$ のとき）になることを示せ。さらに、コードセルのようにずらして作ったペア (x_j,y_{j+1}) でも、隣り合う項どうしが変数を共有しているにもかかわらず、$N\ge 3$ に対して同じ分散になることを示せ。**ヒント：**共有されているペアで条件づけを行う。

2. **ガウスデコーダを用いた Barber–Agakov。**相関 ρ を持つ標準 2 変量ガウス分布のペア（相互情報量は $-\tfrac{1}{2}\log(1-\rho^2)$）と、デコーダ $q(x\mid y)=\mathcal{N}(x;ay,s^2)$（パラメータは $a\in\mathbb{R}$ および $s^2>0$）を考える。下界 Equation 37.75 において、微分エントロピー $h(X)=\tfrac{1}{2}\log(2\pi e)$ を用いよ。
   1. Equation 37.75 の右辺を a と s^2 の関数として計算せよ。両方のパラメータに関して最大化し、最大値が真の相互情報量に一致することを示せ。
   2. 本文では $s^2=1$ に固定し（これは二乗誤差のみで訓練することに相当する）、最良の下界 $\rho^2/2$ を求めた。真の相互情報量とのギャップが、命題の証明どおり $D_{\mathrm{KL}}(\mathcal{N}(0,1-\rho^2)\,\|\,\mathcal{N}(0,1))$ となることを示せ。$\rho\to 1$ のとき、このギャップはどのように振る舞うか。
   3. $\hat{x}(y)$ を X の任意の予測器とし、その平均二乗誤差を D とする。デコーダ $\mathcal{N}(x;\hat{x}(y),D)$ を用いて $I(X;Y)\ge -\tfrac{1}{2}\log D$ を示せ。$X\sim\mathcal{N}(0,v)$ のとき、対応

する下界はどうなるか。これをガウス分布のレート・歪み関数 $R(D) = \tfrac{1}{2}\log(v/D)$（$D < v$ のとき。37.6.3.2 節参照。すなわち X の任意の記述が二乗誤差 D を達成するのに必要な最小の情報量）と比較せよ。

3. **コサインクリティックの温度。** SimCLR 型のシステムは候補を $T(x,y) = s(x,y)/\tau$ によってスコア付けする。ここで $s(x,y) = \mathrm{sim}(z_x, z_y) \in [-1,1]$ はコサイン類似度であり、$\tau > 0$ は温度（ソフトマックス温度）である。1 つのアンカー x に対し、$s_j = s(x,y_j)$（ただし $y_1 = y^+$ が正例）と書き、ℓ を期待値 Equation 37.78 の内部の損失とする。

   1. $\mathcal{L}_{\mathrm{NCE}} \geq \log\big(1 + (N-1)e^{-2/\tau}\big)$ を示し、したがって $\hat{I}_{\mathrm{NCE}} < 2/\tau$ がすべての N に対して成り立つことを導け。上限 $\log N - \log\big(1+(N-1)e^{-2/\tau}\big)$ を $N = 4096$ かつ $\tau \in \{1, 0.5, 0.1\}$ に対して計算せよ（それぞれ約 2.0、4.0、8.3 nat となるはずである）。バッチサイズではなく温度が下界を制限するのはどのようなときか。

   2. $\partial \ell / \partial s_j = p_j/\tau$ がすべての負例 $j \geq 2$ に対して成り立つことを示せ（ただし p_j は候補 j のソフトマックス確率である）。また、$\partial \ell / \partial s_1$ を p_j を用いて表せ。

   3. $\tau \ell \to \max\{0, \max_{j\geq 2} s_j - s_1\}$（$\tau \to 0$ のとき）を示し、さらに次を示せ。

$$\ell = \log N - \frac{1}{\tau}\Big(s_1 - \frac{1}{N}\sum_{j=1}^{N} s_j\Big) + O(\tau^{-2}) \quad \text{as } \tau \to \infty. \tag{37.83}$$

   それぞれの極限において、どの負例が勾配を駆動するか (Wang and Liu 2021)。

4. **サンプルからのエントロピー下界。●**37.7.2.4 節では McAllester and Stratos (2020) の定理を非形式的に述べ、エントロピーに対する版を証明した。本問ではその証明を精密化して応用する。サンプル S として、n 個の独立同分布な値を離散分布 p から抽出する。S の**タイプ**は、各 $i \geq 1$ について、ちょうど i 回現れる異なる値が S の中にいくつあるかを記録したものであり、それらがどの値であるかは忘れる。B をタイプの関数であって、あらゆる離散分布に対して下側信頼限界となるもの、すなわち $P_{S\sim p^n}\big(B \leq H(p)\big) \geq 0.95$ がすべての p に対して成り立つものとする。また、$\mathcal{T}_0$ を n 個のすべて異なる値からなるサンプルのタイプとする。

   1. 本文では n^2 個の値上の一様分布にこの保証を適用した。任意の整数 $M \geq n^2/1.9$ でも同様に成り立ち、したがって $B(\mathcal{T}_0) \leq \log M$ となることを示せ。これにより $2\log n$ からどれだけ改善されるか。
   2. p を $M = 2^{256}$ 個の値上の一様分布とする。n がどれだけ大きくなれば $2\log n$ が $H(p)$ に達するか。この結果は、暗号鍵のエントロピーをサンプルから測定することについて何を物語っているか。

5. **最適な InfoNCE クリティック。●**1 つのアンカー x と 1 つのバッグ $v = (y_1, \dots, y_N)$（ただし $y_1 = y^+$）に対し、$\ell(x,v) = -\log\big(e^{T(x,y_1)} / \sum_{j=1}^{N} e^{T(x,y_j)}\big)$ とおくと、$\mathcal{L}_{\mathrm{NCE}}$（Equation 37.78）はその期待値となる。

   1. $T(x,y)$ を $T(x,y) + c(x)$ に置き換えても（c は任意の関数）、$\mathcal{L}_{\mathrm{NCE}}$ が変化しないことを示せ。

   2. バッグをシャッフルし、正例が一様ランダムな位置 J に置かれ、残りの $N-1$ 個の位置に負例が入るようにする。このとき次が成り立つことを示せ。

$$P(J = i \mid x, y_1, \dots, y_N) = \frac{e^{\mathrm{pmi}(x, y_i)}}{\sum_{j=1}^{N} e^{\mathrm{pmi}(x, y_j)}}. \tag{37.84}$$

ギブスの不等式を用いて、すべてのクリティック $T(x, y) = \mathrm{pmi}(x, y) + c(x)$ が $\mathcal{L}_{\mathrm{NCE}}$ を最小化することを導け。

3. このクリティックに対して、次が成り立つことを示せ。

$$\hat{I}_{\mathrm{NCE}} = I(X; Y) - \mathbb{E}\left[\log \frac{1}{N} \sum_{j=1}^{N} e^{\mathrm{pmi}(x, y_j)}\right]. \tag{37.85}$$

大数の法則を用いて、補正項が $N \to \infty$ で 0 に収束することを（$I(X; Y)$ が有限のとき）論じよ。$\rho = 0.99$ かつ $N = 8$ においてモンテカルロ法でこの補正項を評価し、厳密なクリティックの値（約 1.4）と真の値（1.96）とのギャップ（37.7.4.5 節）と比較せよ。

4. 近似 Equation 37.82 では、平均に対する負例の寄与を 1 に、$\mathrm{pmi}(x, y^+)$ を I に置き換えている。イェンセンの不等式を用いて、第 1 の置き換えは推定値を引き下げ、第 2 の置き換えは推定値を引き上げることを示せ。37.7.4.3 節の表では、どちらの効果が勝っているか。

6. **バッチ内負例と対称損失。** 37.7.4.5 節のコードセルでは、全 $N \times N$ 個のペア（N 個の独立なペア (x_i, y_i) からなるバッチから構成）をスコア付けし、第 i 行の非対角成分をアンカー x_i の負例として用いている。

   1. 各行が InfoNCE の命題の仮定を満たすこと、すなわちその候補 y_j（$j \neq i$）が P_Y からのサンプルであって (x_i, y_i) と独立であることを示せ。ここから、行損失の平均 $\mathcal{L}$ が期待値の意味で $I(X; Y) \geq \log N - \mathcal{L}$ を保証することを導け。
   2. 各行は候補を共有しているため、それぞれの損失は従属である。このことは下界の妥当性、行損失の平均の期待値、あるいはその分散に影響するか。
   3. CLIP は対称損失、すなわち行方向の交差エントロピーと列方向の交差エントロピーの平均を用いる。これも同様に $I(X; Y)$ の下界を与え、同じ上限 $\log N$ を持つことを示せ。
   4. 同時クリティック $T(x, y)$（連結したペアにネットワークを適用するもの）はバッチのスコア計算に N^2 回のネットワーク評価を要するのに対し、分離可能なクリティック $g(x)^\top h(y)$ は $2N$ 回で済む。CLIP のバッチサイズ $N = 32{,}768$ に対して両方の回数を計算せよ。分離可能なクリティックはペアのあらゆる関数を表現できるわけではない。ガウス分布のペアの pmi は 3 次元の埋め込みで分離可能に表せること、そして InfoNCE には 2 次元で十分であることを示せ。

7. **シフトされた PMI としてのネガティブサンプリング。●**word2vec の訓練手法であるネガティブサンプリング付き skip-gram は、中心語 w と文脈語 o をそれぞれのベクトルの内積 $s(w, o) = \mathbf{u}_o^\top \mathbf{v}_w$ によってスコア付けする。そして、コーパス内に現れるペアと、文脈語が K 個のノイズ語（分布 P からサンプリング）の 1 つであるペアとを識別するよう学習する。$p(w, o)$ をコーパス内の全（中心語、文脈語）ペアにおけるそのペアの相対頻度とし、周辺分布を $p(w)$ および $p(o)$ とする。コーパス全体で平均すると、損失は次のようになる。

$$\ell = -\sum_{w,o} p(w, o) \log \sigma(s(w, o)) - K \sum_{w} p(w) \sum_{o} P(o) \log \sigma(-s(w, o)), \tag{37.86}$$

ここで $\sigma(s) = 1/(1+e^{-s})$ はシグモイド関数である。埋め込みは十分な次元を持ち、すべてのスコア $s(w,o)$ を独立に選べると仮定せよ。

1. ℓ が $s^*(w,o) = \log \frac{p(w,o)}{p(w)\,P(o)} - \log K$ で最小化され、これが $\mathrm{pmi}(w,o) - \log K$ に等しくなること（P がユニグラム分布 $p(o)$ のとき）を示せ (Levy and Goldberg 2014)。
2. word2vec は出現頻度の 0.75 乗に比例する確率でノイズ語をサンプリングする。このとき s^* が $\mathrm{pmi}(w,o) - \log K$ から o のみに依存する項だけずれることを示せ。この項が負になるのはどのような文脈語に対してか。また、「sri lanka」（演習問題 1、37.6 節参照）のような低頻度語のペアに対してこの項はどう作用するか。
3. s^* を演習問題 5b の最適な InfoNCE クリティックと比較せよ。InfoNCE では自由なまま残される加法項を、ネガティブサンプリングの二値分類が固定するのはなぜか。

8. [extended] **小規模ネットワークにおける圧縮フェーズ。**●●Shwartz-Ziv and Tishby (2017) は、幅 12-10-7-5-4-3-2 の全結合 tanh ネットワークを、4096 通りの 2 進入力パターン（12 ビット）上で調べた。各隠れ層 Z について、すべてのユニットの活性化値を 30 個の等幅区間（$[-1,1]$ の分割）にビン分割し、ビン分割した値から $I(X;Z)$ と $I(Y;Z)$ を計算した。著者らが報告した圧縮フェーズをめぐる論争については 37.7.2.5 節で述べたとおりである。
   1. X を 4096 通りのパターン上の一様分布とし、Z_{bin} を決定論的ネットワークのビン分割された層とする。$I(X;Z)$ のビン分割推定値が $H(Z_{\text{bin}})$ に等しく、決して $\ln 4096 \approx 8.32$ nat を超えないことを示せ。訓練中にこの値が減少するのはどのようなときか。
   2. 確率ベクトル $\mathbf{w} \in \mathbb{R}^{12}$ を 1 つ固定し、$Y = 1$ となる条件を $\mathbf{w}^\top \mathbf{x}$ が全 4096 パターンにわたる中央値を超えるときと定める。2 つの出力ユニット上にソフトマックスを置いた tanh ネットワークを、ランダムに選んだパターンの 85% 上で交差エントロピー損失を用いた確率的勾配降下法（SGD）により訓練せよ。訓練を通じて、すべての隠れ層の情報平面 $\big(I(X;Z), I(Y;Z)\big)$ における軌跡をプロットせよ。
   3. 隠れユニットを ReLU に変えて (b) を繰り返せ。ただし各ユニットは 30 個の等幅区間（0 とその実行全体での最大の活性化値との間）にビン分割するものとする。
   4. 訓練中に $|z| > 0.99$ となる tanh ユニットの割合を記録せよ。(b) におけるビン分割した $I(X;Z)$ の減少は、この割合に連動しているか。(c) で活性化関数を変更してもなお維持される圧縮についての結論は何か (Saxe et al. 2018)。

37.8 関連文献

本章の主要な節では、明快に証明でき、かつ小さな例で確認できる結果に対象を絞ってきた。以下の各小項では、それぞれの節の議論を引き継ぎ、本筋には収まらなかった周辺の考え方を取り上げる。時間がない場合は、日々の実践で特に直接現れる 3 つのトピック、すなわちモデルの較正（calibration）、スケーリング則（scaling law）における削減不能損失の定数、および相互情報量（mutual information）推定量のバイアス・分散のトレードオフから読み始めるとよい。ビッツバック符号化（bits-back coding）も、その仕掛け自体がきわめて巧妙であるため一読の価値がある。通信路容量（channel capacity）と情報幾何学（information geometry）は後回しにしても差し支えなく、本書の以降の箇所がこのページの内容に依存することはない。

37.8.1 エントロピーと損失

37.8.1.1 訓練後の較正

原理的には、母集団の対数損失（log-loss）を最小化すれば真の条件付きクラス確率が復元され（37.1.3.1 節）、予測は自動的に較正される。しかし、訓練済みのニューラルネットワークがその最適解に落ち着くことはまれである。Guo et al. (2017) が指摘したことで知られるように、現代

のネットワークは特定の方向に偏って誤る、すなわち過信（overconfidence）に陥る傾向がある。その主な原因はワンホットの訓練ラベルにある。ワンホットラベルのもとでは、最大ロジットを他のロジットより無限に大きく引き離すほど損失が下がるため（37.1 節）、ネットワークが訓練セットを完全に分類できるようになった後は、ロジットどうしの差をさらに広げ続けることでしか損失を下げられない。訓練の終盤において同著者らは、ホールドアウトの分類誤差が横ばいのままであるにもかかわらず、ホールドアウトの対数損失が上昇し始めることを観察している。ネットワークは意思決定ではなく確率推定に対して過学習しているのである。温度スケーリング（temperature scaling、3.4 節）は、ホールドアウトの検証セット上で当てはめた単一のスカラーによって、この問題の大部分を解消する。これは蒸留（distillation）で用いられるものと同じ温度パラメータであり、ここでは過信した予測を軟化させる目的で転用されている。

このような事後的な再較正（post-hoc recalibration）はどこまで有効だろうか。$\bar{\mathbf{p}}(\mathbf{q})$ を、モデルが確率ベクトル $\mathbf{q}$ を出力するすべての入力における実際の経験クラス頻度とする。任意の交差エントロピーはエントロピーと KL ダイバージェンスの和に分解されるため、予測ベクトル $\mathbf{q}$ ごとに条件付きでこの恒等式を適用すると、ホールドアウトの対数損失は次の 2 つの項に分解される。

$$\mathbb{E}\big[-\log q_y\big] = \mathbb{E}_{\mathbf{q}}\big[D_{\mathrm{KL}}(\bar{\mathbf{p}}(\mathbf{q}) \,\|\, \mathbf{q})\big] + \mathbb{E}_{\mathbf{q}}\big[H(\bar{\mathbf{p}}(\mathbf{q}))\big].$$

第 1 項は較正のずれ（miscalibration）を nat 単位で測るものであり、モデルの予測確率が主張するとおりの頻度で実際に事象が起こるとき、かつそのときに限り 0 になる。第 2 項は、モデルの予測を知った後でもなお真のラベルについて残る不確実性である。予測確率を事後的にどのように再写像しても、第 1 項を 0 に近づけることはできるが、第 2 項を減らすことは決してできない。温度スケーリングはこのような再写像の 1 パラメータ族にほかならず、それゆえきわめて簡潔であるが、一部のクラスでは過信し別のクラスでは過小確信（underconfidence）に陥っているモデルを 1 つのスカラーで修正することはできない。より一般に、あらゆる固有スコアリングルール（proper scoring rule）は、ダイバージェンス項とエントロピー項への同様の分解をもつ (Gneiting and Raftery 2007)。

37.8.1.2 参照に対する相対的なエントロピー

37.1.4.1 節では、微分エントロピー（differential entropy）の厄介な性質を見た。単位をメートルからセンチメートルに変えるだけで、値が $\ln 100$ だけ増加してしまうのである。これに対し、KL ダイバージェンスはまったく変化しない。比の内部にある両方の密度が同じ係数でスケールされるからである。参照密度 m に対して測り、$-\int p \log(p/m)\, dx = -D_{\mathrm{KL}}(p \| m)$ と定義すれば、エントロピーにも同じ座標不変性を与えることができる。通常の微分エントロピー h は、たまたま選んだ単位系における $m = 1$ という特別な場合にすぎない。1 メートルあたり 1 の一様密度と 1 センチメートルあたり 1 の一様密度は同じ測度ではないため、先の $\ln 100$ のずれの正体は隠れた参照測度の変更だったのである。Jaynes (1968) は、次第に細かくなるグリッド上の離散エントロピーの極限としてこの相対形を導出しており、これを用いると最大エントロピー原理（maximum-entropy principle）は「制約が許す限り参照測度の近くにとどまれ」という自然な形をとる。符号の違いを除けば、変分オートエンコーダ（variational autoencoder、37.6.4.3 節）における KL ペナルティは事前分布（prior）が m の役割を果たすこの種のエントロピーであり、潜在空間（latent space）の滑らかな再パラメータ化のもとで不変に保たれる理由もそこにある。

37.8.2 符号と圧縮

37.8.2.1 モデルを用いない圧縮

37.2 節で扱ったすべての圧縮器は、次の記号を予測するために明示的な確率モデルに依拠している。レンペル・ジブ（Lempel–Ziv）族はその要件を完全に不要にする。LZ77 (Ziv and Lempel 1977) は繰り返し現れる任意の部分文字列をそれ以前の出現箇所を指す短いポインタに置き換え、

LZ78 (Ziv and Lempel 1978) はファイルを走査しながらフレーズの辞書をオンザフライで構築していく。注目すべきことに、両アルゴリズムは**普遍**（universal）である。すなわち、任意の定常エルゴード情報源（stationary ergodic source、大まかに言えば 1 つの長いサンプルから長期的な統計的性質が明らかになる情報源）に対し、情報源に関する事前知識を一切与えなくても、ファイルが大きくなるにつれて平均符号長が真のエントロピー率（entropy rate）へと近づく。もっとも実際には、その極限への収束は遅い。『The Time Machine』に対して、（LZ77 とハフマン符号化を組み合わせた）`zlib` は 1 文字あたり 2.84 ビットを要するのに対し、直前の 3 文字から各文字を予測する 37.2.2.2 節の単純なカウントモデルは、すでに 2.55 ビットまで到達している。

37.8.2.2 ビッツバック符号化

変分オートエンコーダのような潜在変数モデル（latent-variable model）では、37.2.2.1 節の算術符号器を直接駆動することはできない。符号器には厳密な周辺確率（marginal probability）$p(x)$ が必要であるのに対し、モデルが与えるのは ELBO（36.7 節）のみであり、これは $\log p(x)$ に対する、近似事後分布（approximate posterior）$q(z \mid x)$ から構成された下界だからである。驚くべきことに、圧縮ストリームのデコードは数学的にはサンプリングと同じであるという巧妙な双対性（duality）のおかげで、この下界もなお達成可能なファイル長に対応する。$q(z \mid x)$ 向けに設計されたデコーダにランダムなビット列を入力すると、デコーダは約 $\log_2 1/q(z \mid x)$ ビットを消費し、潜在サンプル z を q からの抽出結果として出力する。

ビッツバック符号化は、この知見を実用的な方式へと昇華させる。送信者がそれ以前のデータ点を収めた圧縮ファイルをすでに持っているとする。優れた圧縮は冗長性を取り除くため、そのファイル末尾のビット列はランダムなコイン投げと区別がつかないように見える。新しいデータ点 x を符号化するため、送信者はファイル末尾から $\log_2 1/q(z \mid x)$ ビットを借り受け、それを $q(z \mid x)$ で「デコード」して潜在変数 z を抽出する。次に送信者は、x を $p(x \mid z)$ で符号化し、z を事前分布 $p(z)$ で符号化して、その両方をファイルに追記する。x を展開するため、受信者はこの手順を逆にたどる。まず z を事前分布からデコードし、次に x を $p(x \mid z)$ からデコードしたうえで、z を $q(z \mid x)$ によって**再符号化**し、借りていたビット列を元の場所へとそっくり戻す。全体として、ファイルは $\log_2 1/p(x \mid z) + \log_2 1/p(z)$ ビットだけ増え、$\log_2 1/q(z \mid x)$ ビットだけ縮むため、q のもとでの正味の期待増加量は負の ELBO となる。受信者は送信者の操作を後入れ先出し（last-in, first-out）の順序で取り消さなければならないため、基盤となる符号器はスタックのように振る舞う必要がある。ビッツバックの議論自体は Wallace (1990) および Hinton and Camp (1993) に遡り、Frey and Hinton (1997) はキューとして振る舞う従来の算術符号化（arithmetic coding）の上に最初の動作する符号器を構築した。非対称記数系（asymmetric numeral systems）(Duda 2013) は本来的にスタックベースであり、これを用いて Townsend et al. (2019) は、小規模な変分オートエンコーダで MNIST をその負の ELBO のすぐ近くのレートまで圧縮できることを実証した。

37.8.2.3 2 部符号を超える記述長

37.2.2.3 節の 2 部符号（two-part code）は、選択した精度でモデルパラメータを明示的に符号化するため、総記述長（description length）がパラメータ空間をどれだけ細かく離散化するかに敏感になる。**逐次予測**（prequential）符号は、パラメータの送信を一切行わない。その代わり、37.2.2.2 節の適応カウントモデルとちょうど同じように、それまでに観測したデータ点に当てはめたモデルを用いて各データ点を符号化するため、モデルの学習コストはデータの符号長に直接吸収される。Blier and Ollivier (2018) は逐次予測符号化を用いて深層ニューラルネットワークの訓練ラベルを圧縮し、2 部符号や変分符号（ネットワークの重みにビッツバック符号化を適用する手法）よりもはるかに短い符号を得た。大幅に過剰パラメータ化されたネットワークであっても、この尺度で測ればラベルを確かに圧縮しているのである。Grünwald (2007) は、逐次予測符号と正規化最尤法（normalized maximum likelihood、事後的に選んだクラス最良のモデルに対する最悪ケースのリグレットを最小化する符号）の双方を詳しく展開している。この圧縮の視

点を理論的な極限まで推し進めると、データを出力する最短のコンピュータプログラムの長さであるコルモゴロフ複雑性（Kolmogorov complexity）へと至る。これは以上の考え方をすべて統一する概念であるが、計算不能であることでよく知られている (Li and Vitányi 2019)。

37.8.2.4 スケーリング則の定数

11.6 節では、多様なスケールにわたる言語モデルのホールドアウト損失を、定数 E と 2 つのべき乗則（power law）の項（一方はパラメータ数に応じて減衰し、もう一方は訓練トークン数に応じて減衰する項）の和としてモデル化している (Hoffmann et al. 2022)。E を自然テキストの真のエントロピーと読みたくなるかもしれないし、実際、最良の場合には E はトークンあたりのエントロピー率（37.2.3.1 節）、すなわちいかなる期待損失も下回ることのできない下限に対応する。しかし実際には、E はそれよりはるかに上にある損失から外挿して当てはめられた回帰定数にすぎず、エントロピー率の直接的な測定値ではない。先行研究である Kaplan et al. (2020) は加法定数を一切含まない同様のスケーリング曲線を当てはめていたが、損失はエントロピーの下限を下回れないため、この近似は十分に大きなスケールではいずれ破綻せざるを得ない。また、E がトークン単位で測られている点にも留意すべきである。異なるトークナイザのもとで当てはめられた定数は、双方をバイトあたりビット数（8.3.2.4 節）へ変換しない限り直接比較することはできない。

37.8.3 分布の比較

37.8.3.1 レーニイダイバージェンスとプライバシー

乱択アルゴリズムを繰り返し実行したとき、1 件の訓練レコードについてどれだけの情報が漏洩するだろうか。差分プライバシー（differential privacy）は、その 1 件のレコードだけが異なる 2 つのデータセット上でのアルゴリズムの出力分布間のダイバージェンスとして、この情報漏洩を定式化する。差分プライバシー付き SGD は各勾配ステップでノイズを注入し何千回もの反復を実行するため、独立なステップにわたってきれいに合成できるダイバージェンスが必要になる。次数 α のレーニイダイバージェンス（Rényi divergence）(Rényi 1961) は次の式で定義される。

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \log \sum_x p(x)^\alpha q(x)^{1-\alpha},$$

このダイバージェンスはその性質を備えている。f ダイバージェンスが和の内側で凸関数を適用するのに対し、ここでは対数が和の**外側**にあることがポイントである。独立な座標にわたって和が因数分解されるため、対数によってその積が和へと変換され、結果として D_α は KL ダイバージェンスと同様に、また χ^2 ダイバージェンスや全変動距離（total variation distance）とは異なり、独立なステップにわたってきれいに加法的に合成される。レーニイ差分プライバシー（Rényi differential privacy）(Mironov 2017) は、ノイズを加えた 1 ステップあたりの D_α を上から抑え、その上界を訓練実行全体にわたって足し合わせる。KL ダイバージェンスもステップ間で加算できるが、対数密度比 $\log(p(x)/q(x))$ を平均の意味でしか制御できない。一方、D_α は α に関して単調に増加し、$\alpha \to 1$ での KL ダイバージェンスから $\alpha \to \infty$ での最悪ケースの対数比までを補間するため、より大きな次数 α を選べば、まれに起こる情報漏洩の大きい出力に対しても保護を与えられる。これらの極限と不等式の網羅的な整理は Erven and Harremoës (2014) にまとめられており、演習問題 1d（37.3 節）では D_α と α ダイバージェンスの関係を扱う。

37.8.3.2 情報幾何学

37.3 節では簡単なテイラー展開により、$q_{\theta+\mathbf{h}}$ と q_θ の間の任意の滑らかな f ダイバージェンスが、2 次の精度でフィッシャー情報行列（Fisher information matrix）に関する二次形式

$\frac{1}{2} f''(1)\, \mathbf{h}^\top \mathbf{I}(\theta)\, \mathbf{h}$ に帰着することを見た。これは一見すると代数的な偶然のように思えるかもしれないが、チェンツォフの定理（Chentsov's theorem）は、それ以外の結果にはなり得なかったことを示している。正定値二次形式はあらゆる微小ステップ $\mathbf{h}$ に長さを割り当て、分布の空間をリーマン多様体（Riemannian manifold）へと変える（37.3.1.1 節の距離が 2 つの分布間の大域的な距離を測るのに対し、こちらは局所的なステップを測る）。有限集合上の分布について、Chentsov は、生のデータを十分統計量（sufficient statistic）に置き換えたときに不変に保たれるリーマン計量（Riemannian metric）は、正の定数倍を除いてフィッシャー情報量**のみ**であることを証明した (Čencov 1982)。あらゆる f ダイバージェンスはこの不変性を共有している。θ を知らなくても十分統計量からデータを再サンプリングできるため、37.3.3.3 節のデータ処理不等式（data-processing inequality）が両方向に適用され、等号が強制されるからである。したがって、任意の滑らかな f ダイバージェンスの局所二次近似は、フィッシャー情報量の定数倍でなければならない。この幾何構造のもとで長さを測った勾配ステップを踏むと自然勾配（natural gradient）が得られ、K-FAC はこれを層あたり 1 つのクロネッカー積（Kronecker product）で近似する（35.5.5 節）。また、15.2 節の KL 信頼領域（trust region）も、2 次の精度ではまさにこの計量における球である。Amari (2016) はこの幾何学的視点を全面的に展開し、最尤法と EM アルゴリズム（36.7 節）の両ステップがいかに直交射影（orthogonal projection）として理解できるかを示している。

37.8.3.3 シンクホーンダイバージェンスとスライスワッサースタイン距離

最適輸送距離を高次元ミニバッチ上の実用的な損失へと変える際には、計算コストと統計的バイアス（statistical bias）という 2 つの障壁に直面する。エントロピー正則化（entropic regularization、37.4.2.4 節）は計算上の問題を解決するものの、微妙なバイアスを持ち込む。ある分布からそれ自身への正則化輸送コストが厳密に非ゼロとなるため、これを最小化するよう訓練されたモデルはデータよりも狭い分布へと収縮しやすいのである。演習問題 5（37.4 節）のシンクホーンダイバージェンス（Sinkhorn divergence）は、2 つの自己輸送コストの平均を差し引くことでこの問題を解消する。その演習問題の先には、これを信頼できる訓練目的関数たらしめる定理がある。有界領域上でユークリッド距離またはその二乗を基礎コスト（ground cost）として用いると、シンクホーンダイバージェンスは非負であり、2 つの分布が一致するとき、かつそのときに限り 0 となり、さらにモデルがデータへと分布収束するとき、かつそのときに限り 0 に収束する (Feydy et al. 2019)。

スライスワッサースタイン距離（sliced Wasserstein distance）は、まったく異なる道筋をとる。2 つの高次元サンプルを 1 本の 1 次元直線上に射影すれば、最適輸送は射影された値を単にソートするだけの操作に帰着する（37.4.2 節）。この 1 次元のコストを多数のランダムな方向にわたって平均したものが、スライスワッサースタイン距離である (Bonneel et al. 2015)。ただし高次元における難点は、2 つの分布が実際に異なっている部分空間にランダムな方向が重なることがめったにない点にある。これを回避するため、Deshpande et al. (2018) は識別器（discriminator）を訓練して有用な特徴量を抽出し、生のピクセルではなくその学習された特徴空間上でスライスワッサースタイン距離を計算している。

37.8.3.4 Wasserstein GAN におけるリプシッツクリティック

37.4.2 節のカントロヴィチ・ルビンシュタイン双対（Kantorovich–Rubinstein dual）はリプシッツ定数 1 の関数（1-Lipschitz function）全体にわたって最適化を行うが、通常のニューラルネットワークはそのままではこの制約を満たさない。16.2 節ではこれを課す 3 つの主要な手法（重みクリッピング、スペクトル正規化、勾配ペナルティ）を比較し、16.4 節では勾配ペナルティを詳しく調べている。とりわけ Gulrajani et al. (2017) のペナルティは、実サンプルと生成サンプルの間を補間したランダムな点においてクリティックの勾配ノルムを 1 に近づけるものであり、これは対応づけられた点を結ぶ輸送半直線（transport ray）に沿って最適クリティックが傾き 1 をもつという幾何学的事実を反映している（演習問題 3d、37.4 節）。

37.8.3.5 スタイン変分勾配降下法

分布 Q がそのスコア s_Q を通じてのみ既知である場合、そこからサンプルを抽出するには通常、長い確率的ランダムウォークを実行する必要がある（37.5 節）。37.5.3 節では、カーネルスタイン不一致度（kernel Stein discrepancy）が、サンプル分布 P がそのようなモデル Q からどれだけ離れているかを測ることを見た。スタイン変分勾配降下法（Stein variational gradient descent）(Liu and Wang 2016) は、この不一致度を、粒子の集合を Q へ向かって能動的に輸送する決定論的アルゴリズムへと変換する。分布 P に従う任意の初期点群から出発し、各粒子 y は次の速度場（velocity field）に沿って微小ステップで移動する。

$$v(y) = \mathbb{E}_{x\sim P}\big[k(x,y)\, s_Q(x) + \nabla_x k(x,y)\big].$$

期待値の内側にある 2 つの項は相補的な役割を果たす。第 1 項は y の近傍におけるスコアのカーネル重み付き平均であり、確率密度のより高い領域へと粒子を上り方向に導く。第 2 項は y を近傍の粒子から押し離す斥力であり、ランジュバン動力学におけるランダムノイズの役割を引き継いでいる。これがなければ、すべての粒子が単一のモードへと潰れてしまうだろう。37.5.3 節の再生核ヒルベルト空間の単位球へと正規化すると、この速度場は $D_{\mathrm{KL}}(P\|Q)$ に対する最急降下方向となり、KL ダイバージェンスを $\|v\| = \mathrm{KSD}(P,Q)$ の速さで減少させる。驚くべきことに、同じスタイン作用素（Stein operator）が適合度検定（goodness-of-fit test）と粒子輸送則の両方の役割を果たしているのである。依然として多数の微小ステップは必要であるものの、更新は決定論的であり粒子どうしが協調的に広がるため、元論文における小規模ニューラルネットワークの重みのように、わずか数十個の粒子で事後分布を近似する用途にこの手法は適している。

37.8.4 相互情報量

37.8.4.1 容量に迫る符号

シャノンの通信路符号化定理（Shannon's channel coding theorem、37.6.3.1 節）は、通信路容量未満のあらゆるレートで高信頼な通信が可能であることを保証するが、Shannon の証明はデコードに指数時間を要するランダム符号帳（random codebook）に依拠している。その存在定理と実用的なハードウェアとの隔たりを埋めるには数十年を要した。疎な因子グラフ（factor graph）上でメッセージを伝播させることで効率的にデコードされる低密度パリティ検査（low-density parity-check、LDPC）符号は、実用上きわめて容量に近い性能を達成する (Richardson and Urbanke 2008)。ポーラ符号（polar code）(Arıkan 2009) はさらに一歩進み、対称 2 元入力通信路において容量を達成することが証明された初の明示的な符号族であり、符号化とデコードの双方が $O(n \log n)$ 回の演算で動作する（n はブロック長）。現代の 5G 無線規格では、これら両方の符号族が実用化されている。

37.8.4.2 学習型非可逆圧縮

37.6.3.2 節では、ガウス情報源に対する厳密なレート歪み（rate–distortion）曲線を導出した。自然写真に対しては閉じた形のレート歪み関数が知られていないため、学習型の画像コーデックはいずれも、その未知のフロンティアより上方に位置する実用的な動作点を表すことになる。Ballé et al. (2017) は、エンコーダ、デコーダ、および量子化された潜在表現上の学習された確率分布を定義する**エントロピーモデル**という 3 つの構成要素を同時に訓練する。丸め操作はほとんど至るところで勾配が 0 になるため、訓練中は量子化を加法的な一様ノイズに置き換える。その目的関数は $R + \lambda D$ を最小化する。ここで、レート R はエントロピーモデルのもとでの潜在変数の算術符号化長であり（37.2 節）、D はデコードされた画像の再構成歪みである。λ を変化させると傾き $-\lambda$ の点が選ばれ、コーデックのレート歪み曲線が描かれる。後続研究において Ballé et al. (2018) は、**超事前分布**（hyperprior）と呼ばれる第 2 のより小さな潜在変数でエントロピー

モデルを条件づけることにより、画像の複雑さの空間的な変動を捉え、同じ視覚品質のもとでビットレートを大幅に引き下げている。

37.8.5 相互情報量の推定

37.8.5.1 上界

37.7 節で導出した変分推定量は、いずれも相互情報量の**下界**である。これは共有情報量を最大化したいときには有用だが、本来破棄すべき機微属性や保護属性（protected attribute）に関する情報を学習表現が**ほとんど**保持していないことを保証するには使えない。真の上界を得るには、確率的エンコーダが提供する条件付き密度 $p(z \mid x)$ を利用できる必要がある。変分情報ボトルネック（variational information bottleneck、37.6.4.3 節）はそのような上界の 1 つであり、固定された周辺分布の近似 $r(z)$ を用いる。対照対数比上界（contrastive log-ratio upper bound、CLUB）(Cheng et al. 2020) は $r(z)$ の必要性を完全に取り除く手法であり、ミニバッチ内の対応ペアにわたる $\log p(z \mid x)$ の平均から、非対応ペアにわたる平均を差し引く。$p(z \mid x)$ が真のエンコーダ密度であるとき、CLUB は期待値の意味で上界となる。しかし、表現と保護属性の間には解析的な条件付き密度が存在しないため、代わりにニューラルネットワークを当てはめなければならない。その場合、当てはめた条件付き分布が周辺分布の積よりも真の同時分布（joint distribution）に対して KL ダイバージェンスの意味で近い場合に限り上界が成り立つが、この条件はそもそも未知であった当の密度なしには検証できない。

37.8.5.2 下界のバイアスと分散

厳密に最適なクリティックが与えられた場合でさえ、変分下界は鋭いトレードオフに直面する。NWJ は平均的には不偏であるが高い分散に悩まされるのに対し、InfoNCE は低分散である代わりに $\log N$ で頭打ちになる（37.7.4.4 節）。Poole et al. (2019) は、これら 2 つの下界の間を滑らかに補間する単一の混合パラメータを導入し、このトレードオフを連続的に調整できるようにしている。さらに深刻なことに、J. Song and Ermon (2020) は、分散の爆発が特定の目的関数に固有の副産物ではなく本質的な現象であることを証明した。すなわち、NWJ と Donsker–Varadhan の両下界において、たとえ厳密なクリティックを用いても、推定量の分散は真の相互情報量に対して指数関数的に増大する。同著者らが提案する対策はクリティックの密度比をクリップするものであり、統計的バイアスが生じる代償として分散を抑え込む。要するに、利用可能なあらゆる対策は、誤差を解消するのではなく分散とバイアスの間で移し替えているにすぎない。

実データでは正解の相互情報量がわかっていることはまれであるため、同著者らは既知の答えを必要としない自己無撞着性による妥当性確認（self-consistency sanity check）を提案している。信頼できる推定量であれば、一方の変数にその決定論的関数を付加しても推定値は変化せず、単一の対 (X, Y) をその独立な 2 つのコピーに置き換えれば推定値は 2 倍になるはずである。画像データセット上では、同著者らが評価した標準的な推定量はいずれもこれらの基本的なテストに不合格となっている。実践上の経験則として、数 nat 規模の相互情報量の推定値が報告されている場合には必ずバッチサイズの対数と比較し、手元のデータでこれらの整合性チェックを通過することを確認したうえで初めて推定量を信頼すべきである。

37.8.5.3 ノイズ付きネットワークとボトルネックの視点

37.7.2.5 節の圧縮論争は、Tishby and Zaslavsky (2015) がニューラルネットワークの各層を情報ボトルネックのフロンティア（37.6.4 節）上にプロットすることを提案したことに端を発する。このフロンティアは、達成可能な最大の $I(Y; Z)$ を与えられた $I(X; Z)$ ごとに追跡するものである。しかしその項で見たとおり、この着想は数学的な難点に突き当たる。連続入力を受け取る決定論的ネットワークでは $I(X; Z)$ が形式的に無限大となり、追跡すべき有限の圧縮曲線が存在しなくなるのである。Goldfeld et al. (2019) は、隠れ層の活性化値に小さなガウスノイズを注入

して $I(X;Z)$ を有限にすることで、この問いを良設定（well-posed）にした。このノイズ付きモデルのもとでは実際に訓練中に $I(X;Z)$ が減少し得るが、著者らはその減少の要因が、同一クラスの隠れ表現どうしが互いに引き寄せられる幾何学的なクラスタリングにあることを突き止めている。以前から決定論的ネットワークに対するビン分割ベースの相互情報量推定が捉えていた正体も、まさにこのクラスタリングにほかならないと同著者らは論じている。

37.8.6 情報理論全般の参考資料

書籍

- Elements of Information Theory — Cover & Thomas：エントロピー、KL ダイバージェンス（同書では相対エントロピーと呼ばれる）、相互情報量、およびそれらの連鎖律を第一原理から展開するとともに、情報源符号化（source coding）、通信路容量、レート歪み理論を扱う大学院レベルの古典的標準書である。37.1 節、37.2 節、37.6 節に対応する。
- Information Theory, Inference, and Learning Algorithms — David MacKay：オンラインで無料公開されている書籍である。全編を通じて情報理論と機械学習を織り交ぜて論じており、37.2 節に対応する記号符号（symbol code）とストリーム符号（算術符号化を含む）の章を備える。
- Information Theory: From Coding to Learning — Polyanskiy & Wu：統計学と機械学習に軸足を置いた現代的な大学院向けテキストであり、f ダイバージェンス、変分表現、データ処理の扱いは 37.3 節および 37.6 節に対応する。同書の母体となった MIT 6.441 の講義ノートも無料公開されている。
- Computational Optimal Transport — Peyré & Cuturi：データサイエンス向けにワッサースタイン距離、エントロピー正則化、シンクホーン反復を解説しており、37.4 節の最適輸送パートを補完する。全文は arXiv でも公開されている。
- The Minimum Description Length Principle — Peter Grünwald：2 部符号と普遍符号を通じて統計学習をデータ圧縮として展開し、37.2.2.3 節の内容を掘り下げている。中心的な考え方は無料公開されているチュートリアルでも解説されている。

講義・チュートリアル・ソフトウェア

- Stanford EE376A Information Theory — Tsachy Weissman：エントロピー、接頭符号（prefix code）、典型集合（typical set）を扱うわかりやすい講義であり、学生が筆記した公開講義ノートが提供されている。37.1 節および 37.2 節に対応する。
- Visual Information Theory — Christopher Olah：符号語長から出発してエントロピー、交差エントロピー、相互情報量へと積み上げていく直観的かつ視覚的な入門記事であり、37.1 節、37.2 節、37.6 節に対応する。
- From GAN to WGAN — Lilian Weng：生成モデルにおける KL、イェンセン・シャノン、ワッサースタインの各損失を比較する明快な解説であり、37.3 節および 37.4 節と並行した内容になっている。
- POT: Python Optimal Transport — Flamary et al.：厳密な最適輸送とエントロピー正則化付きシンクホーン反復の両方のソルバを提供するオープンソースの Python ライブラリであり、手元のデータセット上で 37.4.2 節の距離を計算するのに役立つ。
- Generative Modeling by Estimating Gradients of the Data Distribution — Yang Song：スコア関数がなぜ正規化定数（normalizing constant）を回避できるのか、またスコアマッチングがいかにしてスコア関数をデータに当てはめるのか（37.5 節の範囲）を平易に解説したうえで、サンプリングと拡散モデル（diffusion model）へと進むチュートリアルである。

基礎論文

- Language Modeling Is Compression — Delétang et al.：算術符号器と言語モデルの次ト

ークン確率を組み合わせる研究であり、まさに 37.2.2.2 節で実演した着想そのものを扱っている。

- f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization — Nowozin et al.：37.3.4 節で導出したとおり、任意の f ダイバージェンスがそのフェンシェル共役（Fenchel conjugate）による下界を通じて敵対的訓練（adversarial training）の目的関数を与えることを示している。
- A Kernel Two-Sample Test — Gretton et al.：37.4.1 節で扱った最大平均値差異（maximum mean discrepancy）、その不偏 2 サンプル推定量、およびそれに基づく仮説検定を導入した論文である。
- Wasserstein GAN — Arjovsky et al.：サポートが重ならない場合でも勾配が有用な情報を保つよう、イェンセン・シャノン・ダイバージェンスを 1 次ワッサースタイン距離に置き換えた研究であり、37.4.2 節に対応する。
- Estimation of Non-Normalized Statistical Models by Score Matching — Aapo Hyvärinen：対数密度の勾配をデータの勾配に一致させ、未知のデータスコアを部分積分（integration by parts）によって消去することで非正規化モデルを当てはめる手法を提案しており、これらが 37.5 節の 2 つの重要なステップとなっている。
- A Kernelized Stein Discrepancy for Goodness-of-fit Tests — Liu, Lee & Jordan：37.5.3 節で示したように、ペアごとのカーネル評価を用いて、スコアのみを通じて知られるモデルからサンプルが得られたかどうかを検定する。
- The Information Bottleneck Method — Tishby, Pereira & Bialek：X を圧縮しつつ Y に関する情報を保持するという問題を定式化しており、37.6.4 節で学んだ枠組みの原典である。
- Estimating Mutual Information — Kraskov, Stögbauer & Grassberger：最近傍距離から連続変数の相互情報量を推定する 37.7.2.2 節の推定量を提案した論文であり、scikit-learn の `mutual_info_regression` もこれに基づいている。
- On Variational Bounds of Mutual Information — Poole et al.：Barber–Agakov、Donsker–Varadhan、NWJ、InfoNCE の各下界を単一の枠組みで統一し、それらのバイアス・分散のトレードオフを解析しており、37.7.3 節に対応する。
- Representation Learning with Contrastive Predictive Coding — van den Oord et al.：37.7.4 節で展開したとおり、自己教師あり表現学習（self-supervised representation learning）に向けて InfoNCE 損失を相互情報量の下界として導入した論文である。

Chapter 38

ダイナミクス：微分方程式と生成フロー

残差ネットワーク（residual network）は層ごとに入力へ小さな更新を加え、勾配降下法は重みをステップごとに下り方向へ少しずつ動かし、拡散モデル（diffusion model）は学習されたベクトル場（vector field）に沿ってノイズを画像へと変換する。これら 3 つはいずれも、状態が変化する速さを定める規則である微分方程式（differential equation）を離散化したものである。ここからただちに 2 つの実践的な問いが生じる。サンプラはどれだけ少ないステップ数で実行できるのか。また、画像の例だけから速度やスコアの方向をどのように学習できるのだろうか。

訓練画像にノイズを徐々に加えていくことで、2 つ目の問いに逆方向から答えられる。ノイズを帯びた個々の粒子は予測不能にさまようが、粒子群全体の密度は決定論的に時間発展し、その時間発展を逆転させればガウスノイズをデータへと戻せる。これまでの章では、この仕組みまでは踏み込んでいない。誤差逆伝播法（backpropagation）は ODE の解ではなく固定された計算グラフを微分するものであり、静的な確率では移動する密度を追跡できず、$\sqrt{\Delta t}$ オーダーのブラウン増分は通常の連鎖律（chain rule）を破綻させるからである。

本章は 3 つの段階で展開する。第 1 に、決定論的な ODE によって単一の状態を厳密かつ数値的に動かす。ここでは、最も速く減衰するモードが勾配降下法およびすべての陽的ルンゲ・クッタ法のソルバのステップサイズに上限を課す一方（38.2 節）、より真っ直ぐな輸送経路によって少数ステップでの生成が可能になる（38.7.2 節）。その過程で、駆動される線形 ODE から状態空間モデル（state space model）の漸化式が導かれ（38.1.2.3 節、12.4 節）、深い残差ネットワーク（7.4 節）はニューラル ODE および連続正規化フロー（continuous normalizing flow）へとつながる（38.3 節）。第 2 に、確率解析（stochastic calculus）によって連鎖律を修復し（38.4 節）、フォッカー・プランク方程式を通じて粒子群の密度を追跡することで、時間反転に必要なのは**スコア** $\nabla \log p_t$ だけであることを示す（38.5 節）。第 3 に、ノイズを加えた対や補間された対に対する最小二乗回帰（least-squares regression）により、場の正解ラベルなしにスコアや周辺速度（marginal velocity）を直接学習する（38.6 節、38.7 節）。

第 17 章へ直行したい読者は、最初の一読では連続随伴法（continuous adjoint method、38.3.2 節）と最適輸送（optimal transport、38.7.3 節）を読み飛ばしてよい。本章のほとんどの実験では、離散化誤差を明確に切り分けられるよう、閉形式のベクトル場に対してサンプラを検証する。なお、固有値分解（33.3 節）、ヤコビ行列（34.4.1 節）、およびガウス分布の条件付けを前提とし、測度論（measure theory）は仮定しない。

記法

Table 38.1 に、複数の節にわたって繰り返し現れる記号をまとめる。各節ではその節で必要な記号を改めて示し、一度しか使わない文字はその場で定義する。表中の文字が 2 つの意味を持つ場合は、それぞれの意味がどの箇所で成り立つかを該当する行に記している。

Table 38.1: ダイナミクスの章で用いる記法。

記号	意味	初出箇所
$\mathbf{x}(t) \in \mathbb{R}^d$、$\dot{\mathbf{x}}$	状態、その時間微分	38.1 節
$\mathbf{x}_0$	$t = 0$ における状態（軌跡の始点、ノイズ付加過程ではデータ点）	38.1 節
$\mathbf{f}(\mathbf{x}, t)$	ODE のベクトル場、SDE のドリフト	38.1 節、38.4 節
Φ_t、$\Phi_{s\to t}$	フロー写像 $\Phi_t(\mathbf{x}_0) = \mathbf{x}(t)$、時刻 s の状態から時刻 t の状態への写像	38.1 節
L	$\mathbf{f}$ のリプシッツ定数	38.1 節
T	終端時刻（ホライズン）	38.1 節
$\mathbf{A}$、$\lambda_j = a_j + \mathrm{i}\, b_j$、$\mathbf{V}$、$\Lambda$	線形場 $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ の行列、その固有値、固有ベクトルを並べた行列、固有値の対角行列	38.1 節
λ	$\mathbf{A}$ またはヤコビ行列の固有値。逆時間サンプラでは注入するノイズ振幅の定数倍 λg	38.1 節、38.5 節
$\mathbf{K}$	回転を表す行列 $\left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$（$\dot{\mathbf{x}} = \mathbf{K}\mathbf{x}$）	38.1 節
$\mathbf{J}$	$\mathbf{f}$ のヤコビ行列	38.1 節
$\mathbf{x}^\star$	ベクトル場の不動点 $\mathbf{f}(\mathbf{x}^\star) = \mathbf{0}$（勾配フローでは損失の臨界点）	38.1 節
$\mathbf{u}(t)$、$\mathbf{B}$	入力とその行列（$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}(t)$）	38.1 節
h、$t_n = nh$、$\mathbf{x}_n$	ステップサイズ（38.4 節と 38.5 節では Δt と表記）、時間グリッド、t_n における状態（厳密解またはソルバの反復点）	38.1 節、38.2 節
$\bar{\mathbf{A}}$、$\bar{\mathbf{B}}$	入力を区間幅 h にわたり一定に保った厳密ステップ $\mathbf{x}_{n+1} = \bar{\mathbf{A}}\mathbf{x}_n + \bar{\mathbf{B}}\mathbf{u}_n$	38.1 節
$R(z)$、$z = h\lambda$	ソルバの安定性関数	38.2 節
ε_n	大域誤差 $\lVert \mathbf{x}_n - \mathbf{x}(t_n) \rVert$	38.2 節
$\mathcal{L}$、η、$\mathbf{H}$	損失、学習率、ヘッセ行列	38.1 節、38.2 節
N	終端時刻までのステップ数。ソルバのステップ $n = 1, \dots, N$（$Nh = T$、最初の 2 節ではこの総数にも n を用いる）。また 38.6 節では DDPM のノイズレベル $t = 1, \dots, N$	38.3 節
θ	ネットワークのパラメータ	38.3 節
$\mathbf{a}(t)$	随伴状態	38.3 節
p_t	時刻 t における状態の密度	38.3 節
ϵ	38.3 節ではトレース推定量のランダムプローブ。38.4 節以降では $\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon$ における標準ガウスノイズ（38.7 節ではノイズ側端点 $\mathbf{x}_1$ とも表記）	38.3 節、38.4 節
$\mathbf{X}_t$（1 次元では X_t）	時刻 t における確率過程の状態を表す確率変数（$\mathbf{x}$ はそのとる値）	38.4 節
$\mathbf{W}_t$	ブラウン運動	38.4 節
g、$g(t)$、$\mathbf{G}$	定数または時間変化する SDE のノイズ振幅（拡散係数）、ノイズ行列	38.4 節
θ	OU 過程 $dX = -\theta X\, dt + g\, dW$ のレート。38.3 節のスカラー例では $\dot{x} = \theta x$ のパラメータ	38.3 節、38.4 節
α_t、σ_t	$\mathbf{X}_t$ の、$\mathbf{X}_0$ を条件とする平均係数と標準偏差（σ_t はノイズレベル）	38.4 節

記号	意味	初出箇所
ϕ	状態の滑らかな関数。伊藤の補題における関数、およびフォッカー・プランク方程式の背後にあるテスト関数（他の箇所では角度）	38.4 節
ψ	ベクトル解析の準備における滑らかなスカラー場	38.5 節
$\mathbf{s}_t = \nabla \log p_t$、$\mathbf{s}_\theta$	スコア、学習されたスコア	38.5 節、38.6 節
$\mathbf{u}_t$、$\mathbf{u}_\theta$	p_t を輸送する速度（確率フロー速度、入力 $\mathbf{u}(t)$ とは無関係）、学習された速度	38.5 節、38.7 節
$\mathbf{j}$	確率流	38.5 節
$\bar{t} = T - t$	逆向きの時刻	38.5 節
w_k、$r_k(\mathbf{x})$、v_k	混合重み、負担率、各成分の分散	38.5 節
$\beta(t)$、β_t、$\bar{\alpha}_t$	分散保持 SDE のレート。DDPM のノイズスケジュール（およそ $\beta(t)$ とステップ幅の積であり、38.4 節では β_k と表記）とその累積積 $\bar{\alpha}_t = \prod_{s \le t}(1-\beta_s)$（本章の α_t^2 に相当）	38.4 節、38.6 節
$\bar{\mathbf{x}}_t$、$\bar{\sigma}_t$	DDIM 座標（リスケールされた点 $\mathbf{x}_t/\sqrt{\bar{\alpha}_t}$ とそのノイズ対信号比 $\sqrt{(1-\bar{\alpha}_t)/\bar{\alpha}_t}$）	38.6 節
$w(t)$、γ	損失におけるノイズレベル t の重み、ガイダンス強度	38.6 節
π	カップリング（データとノイズの組の同時分布、ペアリングとも呼ぶ）	38.7 節
i、$\mathbf{I}$	虚数単位（立体）、単位行列	章全体

38.1 常微分方程式

本章の導入部で述べたように、勾配ステップ、残差ブロック（residual block）、および拡散サンプラの 1 ステップには共通点がある。いずれも、状態がどれほどの速さで変化するかを表す矢印に沿った微小な移動である。次節ではそのようなステップがどれほど良い近似かを問うが、本節ではそもそもそれが**何の**ステップなのかを問う。天気図の上の風のように、すべての点で速度が指定されており、その中に気球を放つとしよう。気球の経路は一意に定まるだろうか。同じ場所から放たれた 2 つの気球が別々の道を進むことはあるだろうか。1 時間前に気球がどこにあったかを知ることができるだろうか。そして、気球は最終的にどこへ行き着くのだろうか。

1687 年の Newton の法則はまさにこの種の規則であり、物体の状態がどう変化するかを述べるものであって、将来どこにあるかを直接与えるものではない。このような規則がただ 1 つの解を持つことは自明ではない。それを保証する定理は 1890 年代に Picard と Lindelöf によって証明されたものであり、速度がある点から隣の点へと急激に変化しすぎないことを要請する。後ほど、一見無害でありながらこの条件を満たさない規則に出会うことになる。

モデルにはこの定理が不可欠である。ネットワークが速度を与え、運動によって入力が運ばれた先をモデルの出力とするとき、解の一意性によって出力は入力の関数となり、運動を逆向きにたどればその逆関数が得られる。これは生成フロー（generative flow）が必要とする性質である。線形の規則であればさらに踏み込んで、解を具体的に書き下すことができる。その行列の固有値は、運動が減衰するか、増大するか、あるいは回転するかを決定しており、状態空間モデル（state space model、12.4 節）の記憶や訓練の収束もそこから読み取れる。

まずは手計算で解ける 2 つの運動から始め、上記の定理を証明したうえで、線形方程式を完全に解く。以降の議論はすべて 2 つの結果に基づいている。すなわち、緩やかに変化する速度規則は各点を通る軌跡（trajectory）をただ 1 つ持ち、線形規則の運動が減衰するかどうかはその固有値の実部によって決まるということである。減衰する前にいったん増大する減衰系と振り子を扱う最後の 2 つの項は、初読時には飛ばしてもよい。

38.1.1 ベクトル場とフロー

38.1.1.1 減衰、回転、スパイラル

最も単純な運動は**減衰**（decay）である。「距離に比例する速さでゼロへ向かって動く」という規則は方程式 $\dot{x} = -x$ で表される。ここでドットは時間に関する導関数を表す。初期条件 $x(0) = x_0$ からの解は次のとおりである。

$$x(t) = e^{-t} x_0,$$

このことは微分してみれば確かめられ、$\dot{x} = -e^{-t}x_0 = -x$ となる。単位時間が経過するごとに、原点までの距離は $e^{-1} \approx 0.37$ 倍になる。

次は**回転**（rotation）である。平面上において、位置ベクトルを直角に回転させたものを速度としよう。

$$\dot{\mathbf{x}} = \mathbf{K}\mathbf{x}, \qquad \mathbf{K} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix},$$

すなわち $\dot{x}_1 = -x_2$ および $\dot{x}_2 = x_1$ である。$\mathbf{x}^{\top}\mathbf{K}\mathbf{x} = -x_1 x_2 + x_2 x_1 = 0$ であるから、速度は位置と直交しており、したがって $\mathbf{x}$ の長さは決して変化しない。

$$\frac{d}{dt}\|\mathbf{x}\|^2 = 2\,\mathbf{x}^{\top}\dot{\mathbf{x}} = 2\,\mathbf{x}^{\top}\mathbf{K}\mathbf{x} = 0.$$

点は円周上を動き、その速さ $\|\mathbf{K}\mathbf{x}\|$ は半径 $\|\mathbf{x}\|$ に等しいため、単位時間あたり 1 ラジアンの速さで回転する。したがって 1 周するのに 2π かかる。

回転に半分の速さの減衰を加えると、本節および続く 2 つの節を通して例として用いる方程式が得られる。

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \qquad \mathbf{A} = -\tfrac{1}{2}\mathbf{I} + \mathbf{K} = \begin{pmatrix} -0.5 & -1 \\ 1 & -0.5 \end{pmatrix}.$$

回転の項は長さを変えず、減衰の項が長さを縮めるため、$\frac{d}{dt}\|\mathbf{x}\|^2 = 2\mathbf{x}^{\top}\mathbf{A}\mathbf{x} = -\|\mathbf{x}\|^2$ となる。これは長さの二乗に対する先ほどの減衰方程式にほかならず、$\|\mathbf{x}(t)\|^2 = e^{-t}\|\mathbf{x}_0\|^2$、すなわち $\|\mathbf{x}(t)\| = e^{-t/2}\|\mathbf{x}_0\|$ が従う。点は単位角速度で回転しながら速さ $\frac{1}{2}$ の率で長さを失い、スパイラルを描いて原点へと吸い込まれていく（Figure 38.1）。$\mathbf{A}$ の固有値は $-0.5 \pm \mathrm{i}$ であり、その実部 $-\frac{1}{2}$ が減衰率を、虚部 ± 1 が角速度を与えている。これは偶然ではない。38.1.2 節で示すように、この種のあらゆる線形方程式は、各固有値に対応する減衰・増大・回転の和として表されるのである。

38.1.1.2 初期値問題

運動の規則の大半は線形ではないため、それらをすべて扱える一般的な形式が必要である。一般の問題は、速度の規則と出発点を与える次の形で書かれる。

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t), \qquad \mathbf{x}(0) = \mathbf{x}_0. \tag{38.1}$$

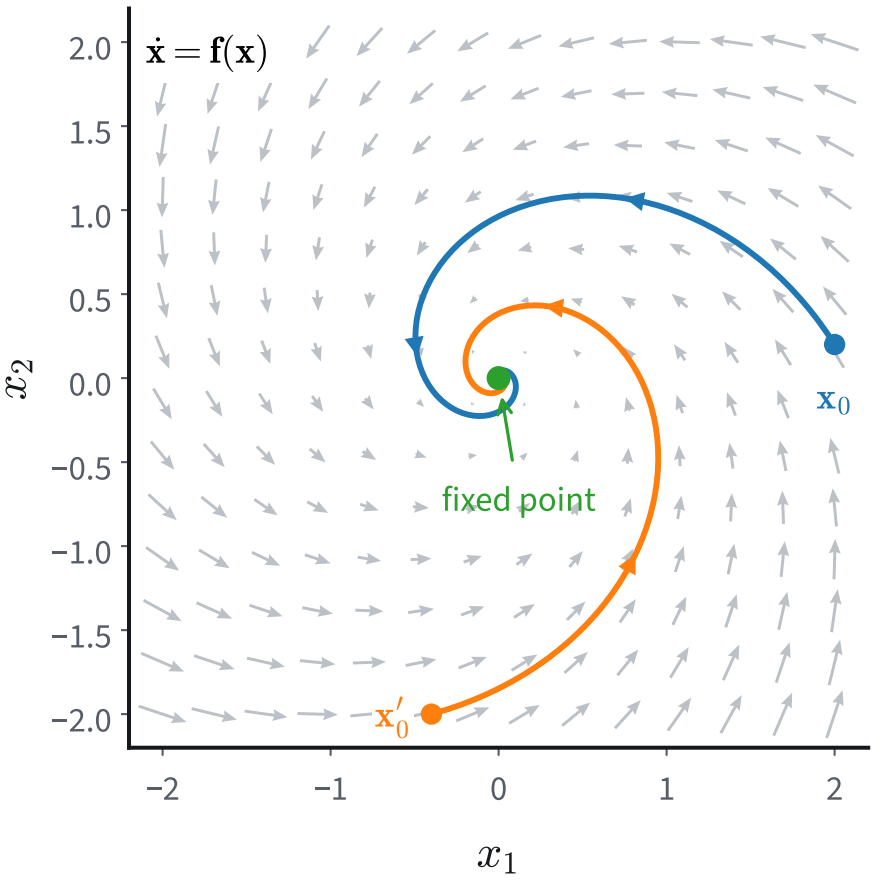


Figure 38.1: 速度場としての ODE。薄いグリッド状の矢印は場 $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$（ただし $\mathbf{A} = \left(\begin{smallmatrix} -0.5 & -1 \\ 1 & -0.5 \end{smallmatrix}\right)$、固有値は $-0.5 \pm \mathrm{i}$）を表す。色付きの 2 本の曲線は軌跡であり、それぞれが各瞬間で局所的な矢印に沿って進み、場がゼロになる唯一の点（不動点）である原点へとスパイラル状に収束する。

ここで、$\mathbf{x}(t) \in \mathbb{R}^d$ は状態である。関数 $\mathbf{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ は**ベクトル場**（vector field）であり、各時刻・各点に矢印を対応させる。**軌跡**とは微分可能な経路であって、各瞬間における接ベクトルがその時点の位置にある矢印と一致するものをいう。

ベクトル $\mathbf{x}_0$ は**初期値**（initial value）であり、Equation 38.1 は**初期値問題**（initial-value problem）と呼ばれる。$\mathbf{f}$ が t に依存しないとき、系は**自律的**（autonomous）であるといい、矢印は固定されたまま状態だけが動く。減衰、回転、スパイラルは、それぞれ $\mathbf{f}(\mathbf{x}) = -\mathbf{x}$、$\mathbf{Kx}$、$\mathbf{Ax}$ という自律的な場である。一方、気球が飛んでいるあいだに風向きが変わる天気図は自律的ではない。

両辺を 0 から t まで積分すると（34.6 節）、同値な**積分形**（integral form）が得られる。

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(s), s)\, ds. \tag{38.2}$$

解は、初期位置に累積された速度を足し合わせたものになる。未知の経路が両辺に現れているため、これは依然として方程式であって明示的な解の公式ではないが、以下の存在証明ではこの形を用いる。

38.1.1.3 存在と一意性

初期値問題はつねに解を持ち、しかもただ 1 つだけ持つのだろうか。必ずしもそうとは限らない。$\dot{x} = \sqrt{|x|}$ を初期値 $x_0 = 0$ で考えよう。定数経路 $x \equiv 0$ はこの方程式の解である。さらに、任意の待ち時間 $c \geq 0$ に対して、次の関数もまた解となる。

$$x_c(t) = \begin{cases} 0, & t \leq c, \\ (t-c)^2/4, & t > c, \end{cases}$$

実際、$\dot{x}_c = (t-c)/2 = \sqrt{x_c}$ が $t > c$ で成り立つからである。静止している粒子は、いつ出発してもよいことになる。解は出発時刻ごとに 1 つずつ軸から扇状に広がっており（Figure 38.2 の (a)）、以下ではこの図形を**扇**（fan）と呼ぶ。この場で定義されたモデルは、入力に対してただ 1 つの出力を対応させることができない。原因は $\sqrt{|x|}$ の傾きが 0 で非有界であることにある。わずかな変位 x が、それよりはるかに大きな速度 $\sqrt{|x|}$ を生み出してしまうのである。

性質のよい場に対しては、積分形 Equation 38.2 を反復することで解を構成できる。すなわち、経路を 1 つ推測し、それに沿って場が指定する速度を読み取り、初期位置からそれらを積分して新しい経路を得て、この操作を繰り返すのである。これを **Picard 反復**（Picard iteration）という。$\dot{x} = x$、$x_0 = 1$ の場合、定数経路 $\varphi_0 \equiv 1$ から出発して $\varphi_{k+1}(t) = 1 + \int_0^t \varphi_k(s)\, ds$ とおくと、次のようになる。

$$\varphi_1(t) = 1 + t, \qquad \varphi_2(t) = 1 + t + \tfrac{t^2}{2}, \qquad \varphi_3(t) = 1 + t + \tfrac{t^2}{2} + \tfrac{t^3}{6}.$$

この反復列は解 e^t に収束する。この例では反復を 1 回進めるごとに e^t のテイラー展開の項が 1 つずつ加わっていく。他の場では反復列の形は異なるものの、場がある点から別の点へと急激に変化しすぎない限り、やはり収束する。次の定理は、この「急激に変化しすぎない」という意味を厳密に定式化する。

定理（Picard–Lindelöf） **$\mathbf{f}(\mathbf{x}, t)$ が t について連続であり、$\mathbf{x}$ について大域的リプシッツ連続**（globally Lipschitz）**であるとする。すなわち、ある定数 L が存在して次を満たすとする。**

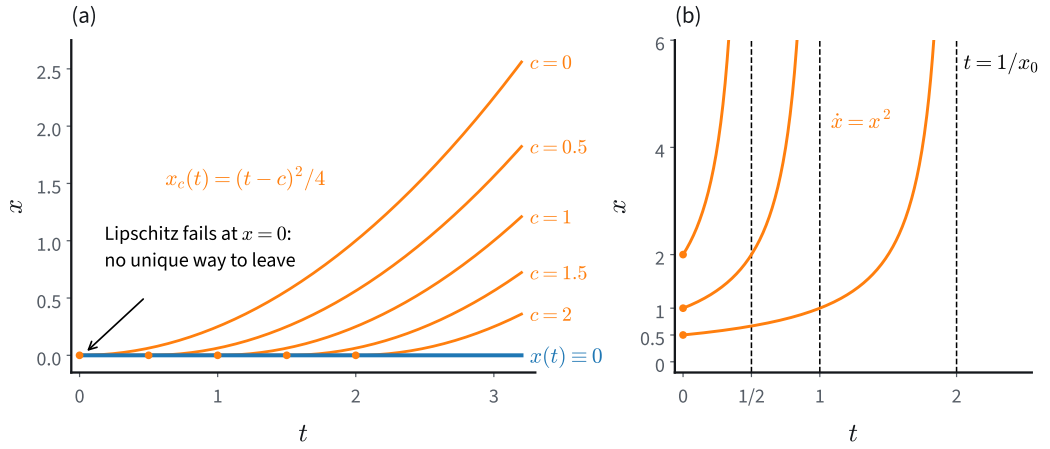


Figure 38.2: 以下の定理が適用できない 2 つの場。(a) $\dot{x} = \sqrt{|x|}$ の $x(0) = 0$ を通る解。定数解 $x \equiv 0$ に加え、放物線状の解 $x_c(t) = (t-c)^2/4$ が任意の出発時刻 c ごとに存在する。原点において $\sqrt{|x|}$ の傾きが非有界であるため、速度の変化の速さを抑える定数が存在せず（定理のリプシッツ条件が破れる）、解は一意ではない。(b) $\dot{x} = x^2$ の $x_0 = 2$、1、$1/2$（黒丸）から出発する解。ここでは場の傾きは各点の近傍で有界だが、すべての x に共通する 1 つの定数では抑えられず、各解 $x_0/(1 - x_0 t)$ は無限大に到達する時刻 $t = 1/x_0$（破線）までしか存在しない。

$$\|\mathbf{f}(\mathbf{x}, t) - \mathbf{f}(\mathbf{y}, t)\| \le L \, \|\mathbf{x} - \mathbf{y}\| \qquad \text{for all } \mathbf{x}, \mathbf{y}, t.$$

このとき、初期値問題 Equation 38.1 には区間 $[0, T]$ 上でただ 1 つの解が、任意の $\mathbf{x}_0$ および任意の $T > 0$ に対して存在する (Coddington and Levinson 1955)。

これこそ、場によってモデルを定義することを可能にする定理である。定数 L は、点から点へと移るときに矢印がどれほど速く変化しうるかを上から抑える。減衰では $L = 1$ であり、線形の場 $\mathbf{Ax}$ では $L = \|\mathbf{A}\|$ となる。ここで、ここでも本節の残りでも、行列のノルムとはスペクトルノルム（spectral norm）、すなわちその行列がベクトルを引き伸ばす最大の倍率を指す（33.4.2.2 節）。先ほどの扇は、そのような L が 0 の近傍に存在しないときに起こる現象である。以下の証明では、Picard 反復列が収束し、しかも唯一の解に収束することを示す。

証明連続な経路 φ（区間 $[0, \delta]$ 上で定義されるもの）がこの問題の解であるのは、それが次の **Picard 作用素**（Picard operator）の不動点（fixed point）であるとき、かつそのときに限る。

$$(P\varphi)(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\varphi(s), s) \, ds.$$

言葉で言えば、P は Picard 反復の 1 ステップであり、推測した経路に沿って速度を読み取り、初期位置からそれらを積分して足し合わせる操作である。（解は積分形を満たし、逆に連続な不動点は微分積分学の基本定理（fundamental theorem of calculus）によって微分可能であり、正しい導関数と正しい初期値を持つ。）

証明の核心となる非自明なステップは、リプシッツ条件による評価を積分の内側へ通すことである。$[0, \delta]$ 上の 2 つの経路に対して $d(\varphi, \psi) = \max_{t \le \delta} \|\varphi(t) - \psi(t)\|$ とおく。このとき $t \le \delta$ に対して次が成り立つ。

$$\|(P\varphi)(t)-(P\psi)(t)\| \le \int_0^t L\,\|\varphi(s)-\psi(s)\|\,ds \le L\,\delta\; d(\varphi,\psi).$$

$\delta = 1/(2L)$ ととれば、P を 1 回適用するごとに 2 つの候補経路のあいだの距離は半分になる。

定数経路 $\varphi_0 \equiv \mathbf{x}_0$ から出発し、$\varphi_{k+1} = P\varphi_k$ と定める。すると隣り合う反復点の差は高々 $2^{-k}d(\varphi_1,\varphi_0)$ となり、これらの差の和は幾何級数のように抑えられる。したがって反復列は距離 d に関して、すなわち $[0,\delta]$ 上で一様に、ある経路 φ へと収束する。ここでは解析学の事実を 1 つ証明なしで認めることにする。それは、連続な経路の一様極限は連続であるという事実である。$d(P\varphi_k, P\varphi) \le \tfrac{1}{2}d(\varphi_k,\varphi) \to 0$ であるから、経路の列 $P\varphi_k = \varphi_{k+1}$ は $P\varphi$ にも φ にも収束する。よって $\varphi = P\varphi$ が成り立ち、この極限は解である。また、2 つの解 φ,ψ があれば $d(\varphi,\psi) = d(P\varphi,P\psi) \le \tfrac{1}{2}d(\varphi,\psi)$ を満たすため、両者は一致する。

最後に、$t=\delta$ において $\varphi(\delta)$ から再び同じ議論を始める。ステップ幅 $1/(2L)$ は解がどこにあるかに依存しない。なぜなら L は至るところで同じだからであり、したがってこの長さの区間を $\lceil 2LT \rceil$ 個つなげば $[0,T]$ 全体が覆われ、各区間上にただ 1 つの解が定まる。$\blacksquare$

この定理は、リプシッツ評価が存在することと、その評価が大域的に成り立つことの両方を要請しており、それぞれの条件が異なる破綻を排除している。リプシッツ評価がない場合に何が起こるかは、扇の例が示している。一方、大域的な評価がなければ、解は爆発しうる。場 $\dot{x} = x^2$ は滑らかであり、手計算で解くことができる。$\frac{d}{dt}(-1/x) = \dot{x}/x^2 = 1$ より、量 $-1/x$ は単位速度で増加する。すなわち $-1/x(t) = t - 1/x_0$ であり、したがって次式が得られる。

$$x(t) = \frac{x_0}{1 - x_0 t},$$

この解は、初期値 $x_0 > 0$ から出発すると有限時刻 $t = 1/x_0$ で無限大に到達する（Figure 38.2 の (b)）。なぜこの例が定理の対象外なのかは、証明の最後のステップを見ればわかる。区間 $|x| \le \rho$ 上では、場は $L = 2\rho$ でリプシッツ連続である（$|x^2 - y^2| = |x+y|\,|x-y|$ による）。そのため、解が大きくなるにつれて再出発の区間幅 $1/(2L) = 1/(4\rho)$ が縮んでいき、それらを無限個足し合わせても有限時刻にしかならないことが起こりうる。有界集合上でのみリプシッツ連続な場は、しばらくのあいだしか解の存在を保証しない。

38.1.1.4　フロー写像

生成モデルは初期点の雲全体を一度に動かすため、ここからは場を固定して初期点を変化させることにしよう。まずは単純な場合として、時間変化しない場を取り上げる（時間変化する風の例は本項の最後で再び扱う）。自律的かつ大域的リプシッツ連続な場に対して、次を定義する。

$$\Phi_t(\mathbf{x}_0) = \mathbf{x}(t)$$

これは、時刻 t における、$\mathbf{x}_0$ から出発した唯一の軌跡の状態を表す。この**フロー写像**（flow map）$\Phi_t : \mathbb{R}^d \to \mathbb{R}^d$ は、空間全体を矢印に沿って時間 t だけ前へ進める。

減衰の場合、フロー写像は $\Phi_t(x) = e^{-t}x$ である。時間 s だけ流してからさらに時間 t だけ流すと $\Phi_t(\Phi_s(x)) = e^{-t}e^{-s}x = e^{-(t+s)}x = \Phi_{t+s}(x)$ となり、また $\Phi_{-t}(x) = e^{t}x$ は Φ_t を元に戻す。同じ法則があらゆる自律的な場に対して成り立つ。

$$\Phi_0 = \mathrm{id}, \qquad \Phi_{t+s} = \Phi_t \circ \Phi_s. \tag{38.3}$$

その理由は、矢印が時間とともに変化しないことにある。経路 $\tau \mapsto \mathbf{x}(s+\tau)$ は $\Phi_s(\mathbf{x}_0)$ から出発して同じ矢印に沿うため、一意性によりその点からの軌跡に一致する。減衰の例からは、この法則が負の時刻に対しても成り立つことが示唆される。実際そのとおりであり、場を逆向きにたどることは、別の場に同じ定理を適用することに帰着するからである。

系（フロー写像は全単射である） $\mathbf{f}(\mathbf{x})$ が自律的かつ大域的リプシッツ連続であるとする。このとき Φ_t はすべての $t \in \mathbb{R}$ に対して存在し、各 Φ_t は $\mathbb{R}^d$ 上の全単射（bijection）であって逆写像 Φ_{-t} を持つ。一方、$\mathbf{f}(\mathbf{x}, t)$ が時刻に依存し、かつ定理の仮定を満たすならば、任意の $T > 0$ に対して時刻 0 の状態を時刻 T の状態へと運ぶ写像は $\mathbb{R}^d$ 上の全単射である。

証明 場 $\mathbf{g}(\mathbf{y}) = -\mathbf{f}(\mathbf{y})$ は同じ定数 L でリプシッツ連続である。経路 $\mathbf{x}(t)$ が $\dot{\mathbf{x}} = \mathbf{f}$ の $[-T, 0]$ 上の解であるのは、$\mathbf{y}(s) = \mathbf{x}(-s)$ が $\dot{\mathbf{y}} = \mathbf{g}$ の $[0, T]$ 上の解であるとき、かつそのときに限る。なぜなら $\dot{\mathbf{y}}(s) = -\dot{\mathbf{x}}(-s) = -\mathbf{f}(\mathbf{y}(s))$ だからである。$\mathbf{g}$ に定理を適用すれば、任意の初期値に対して時間を逆向きにたどる一意な解が得られるため、Φ_t はすべての実数 t に対して定義される。さらに合成則 Equation 38.3 の証明では s や t の符号を使っていなかったから、$\Phi_{-t} \circ \Phi_t = \Phi_t \circ \Phi_{-t} = \Phi_0 = \mathrm{id}$ が従う。

この時間反転の議論には、矢印が静止している必要はなかった。時刻に依存する場に対しては、時刻 T からフィルムを逆再生すればよい。経路 $\mathbf{x}(t)$ が $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ の $[0, T]$ 上の解であるのは、$\mathbf{y}(s) = \mathbf{x}(T - s)$ が $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y}, s)$（ただし $\mathbf{g}(\mathbf{y}, s) = -\mathbf{f}(\mathbf{y}, T - s)$）の解であるとき、かつそのときに限る。この $\mathbf{g}$ もまた時刻について連続かつ同じ定数 L でリプシッツ連続であるため、定理により各点 $\mathbf{z}$ からは逆向きの経路がただ 1 つ出発する。これを順方向に読めば、各点 $\mathbf{z}$ に時刻 T で到達する唯一の軌跡にほかならない。すなわち、すべての終端状態はただ 1 つの初期状態から生じる。$\blacksquare$

時間とともに変化する場で失われるのは、Equation 38.3 の形そのものだけである。正午からの 1 時間の風と真夜中からの 1 時間の風は異なる写像を与えるため、写像は経過時間だけでなく開始時刻にも依存する。そこで $\Phi_{s \to t}(\mathbf{x})$ によって、時刻 t における状態を表そう（ただし軌跡は位置 $\mathbf{x}$ を時刻 s に通るものとする）。s から r へ進み、さらに r から t へ進むことは、先ほどと同じ一意性の議論により s から t へ進むことに等しい。

$$\Phi_{s \to s} = \mathrm{id}, \qquad \Phi_{r \to t} \circ \Phi_{s \to r} = \Phi_{s \to t}. \tag{38.4}$$

これら 3 つの時刻はどの順序で並んでいてもよいため、$\Phi_{t \to s}$ は $\Phi_{s \to t}$ の逆写像となる。自律的な場では $\Phi_{s \to t} = \Phi_{t - s}$ であり、Equation 38.4 は元の法則に帰着する。時刻に依存する場に対しても、Φ_t という記号を $\Phi_{0 \to t}$ の略記として引き続き用いる。この 2 時刻間の法則は単なる形式的な書き換えではない。17.8 節の少数ステップサンプラは、ネットワークが $\Phi_{s \to t}$ そのものを模倣するように訓練し、その一部はこの法則自体を訓練信号として用いる。すなわち、ネットワークによる 2 回の短いジャンプの合成が、1 回の長いジャンプと一致しなければならない。

ここから得られる帰結の 1 つとして、2 つの異なる軌跡が同じ時刻に同じ点に居合わせることは決してない。もしそのような合流点から時間を逆向きにたどれば、定理によってただ 1 つの経路しか許されないからである。リプシッツ連続な風に対しては、これで冒頭の最初の 3 つの問いに答えたことになる。気球の経路は一意に定まり、同時に同じ場所から放たれた 2 つの気球は離れずに進み、天気図を逆再生すれば 1 時間前にどこにあったかがわかる。

ノイズからデータへの可逆な写像であり、しかもその逆写像を計算できることこそ生成フローが必要とする性質であり、多くのニューラルネットワークが定める場はこの条件を満たす。たとえば $\mathbf{f}(\mathbf{x}) = \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$ のように、線形変換とリプシッツ連続な活性化関数から構成される場は大域的リプシッツ連続であり、その定数 L は高々、重みのノルムと活性化関数のリプシッツ定数の積（この例では $\|\mathbf{W}_2\| \, \|\mathbf{W}_1\|$）で抑えられる。したがって、場が時刻にも依存しうる 38.3 節のモデルは適定（well-posed）であり、先ほどの系によって入力から出力への写像は全単射となる。

38.1.2 線形 ODE と安定性

冒頭で挙げた最後の問い、すなわち運動が最終的にどこへ行き着くかという問いには、ある 1 つの族に対して完全な答えがある。線形の場は閉形式で解くことができ、さらに場がゼロになる点の近傍では、他の大半の場の振る舞いをも記述する。

38.1.2.1 行列指数関数

スカラー方程式 $\dot{x} = ax$ の解は $x(t) = e^{at}x_0$ である。次の線形系を考えよう。

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \qquad \mathbf{x}(0) = \mathbf{x}_0, \qquad \mathbf{A} \in \mathbb{R}^{d \times d},$$

これに対しても、指数部に行列を置いた同じ形の解を探す。任意の正方行列 $\mathbf{C}$ に対し、e^{z} を定めるべき級数によって**行列指数関数**（matrix exponential）を次のように定義する。

$$e^{\mathbf{C}} = \sum_{k=0}^{\infty} \frac{\mathbf{C}^k}{k!} = \mathbf{I} + \mathbf{C} + \frac{\mathbf{C}^2}{2} + \frac{\mathbf{C}^3}{6} + \cdots . \tag{38.5}$$

この級数はあらゆる行列に対して収束する。ある行列で引き伸ばしてから別の行列で引き伸ばしたときの倍率は高々それぞれの倍率の積、すなわち $\|\mathbf{C}\mathbf{D}\| \le \|\mathbf{C}\|\,\|\mathbf{D}\|$ で抑えられるため、$\|\mathbf{C}^k\| \le \|\mathbf{C}\|^k$ が成り立ち、級数の各項は収束するスカラー級数 $e^{\|\mathbf{C}\|}$ によって上から抑えられるからである。

命題（行列指数関数は線形 ODE を解く） $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$ **は** $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$、$\mathbf{x}(0) = \mathbf{x}_0$ **の一意な解である。**

証明級数 Equation 38.5 に $\mathbf{C} = \mathbf{A}t$ を代入したものを項別に微分する（任意の有界な t 区間上で、微分後の級数も先ほどと同じ優級数の評価により一様収束するため、項別微分が正当化される）。

$$\frac{d}{dt} e^{\mathbf{A}t} = \sum_{k=1}^{\infty} \frac{\mathbf{A}^k\, t^{k-1}}{(k-1)!} = \mathbf{A} \sum_{j=0}^{\infty} \frac{(\mathbf{A}t)^j}{j!} = \mathbf{A}\, e^{\mathbf{A}t}.$$

よって $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$ はこの ODE を満たし、$t = 0$ では級数が $\mathbf{I}$ になるため $\mathbf{x}(0) = \mathbf{x}_0$ も満たす。一意性は Picard–Lindelöf の定理から従う。実際、場 $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ は $L = \|\mathbf{A}\|$ でリプシッツ連続である。$\blacksquare$

行列のべき乗の無限級数は、このままでは振る舞いを読み取れる式ではない。固有値分解（eigendecomposition）を用いると、これは d 個のスカラー指数関数へと変わる。33.3 節で見たように、$\mathbf{A}$ が d 個の線形独立な固有ベクトル $\mathbf{v}_j$（$\mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j$）を持つならば、$\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_d)$ および $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$ とおくことで $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$ が成り立つのであった。

するとべき乗は $(\mathbf{A}t)^k = \mathbf{V}(\Lambda t)^k\mathbf{V}^{-1}$ のように簡約される。隣り合う $\mathbf{V}^{-1}\mathbf{V}$ がすべて打ち消し合うからであり、級数 Equation 38.5 の和をとると次式が得られる。

$$e^{\mathbf{A}t} = \mathbf{V} e^{\Lambda t} \mathbf{V}^{-1}, \qquad e^{\Lambda t} = \operatorname{diag}\!\left(e^{\lambda_1 t}, \dots, e^{\lambda_d t}\right). \tag{38.6}$$

固有値の節ではこの公式を引用するにとどめ、その導出は本節に委ねられていた。この式は、系が非結合化されることを意味している。初期状態を固有基底で $\mathbf{x}_0 = \sum_j c_j \mathbf{v}_j$ と展開すれば、次が得られる。

$$\mathbf{x}(t) = \sum_{j=1}^{d} c_j\, e^{\lambda_j t}\, \mathbf{v}_j .$$

各固有方向に沿って、系はスカラー方程式 $\dot{x} = \lambda_j x$ となり、他の方向とは独立に時間発展する。各項 $c_j e^{\lambda_j t}\mathbf{v}_j$ は系の**モード**と呼ばれる。

スパイラルの例では、一般公式よりもさらに見通しよく計算できる。その行列は $\mathbf{A} = -\tfrac{1}{2}\mathbf{I} + \mathbf{K}$ という和の形をしており、数に対して $e^{p+q} = e^{p}e^{q}$ と分解できるように指数関数を分解したい。行列において $e^{\mathbf{P}+\mathbf{Q}} = e^{\mathbf{P}}e^{\mathbf{Q}}$ という法則が成り立つのは $\mathbf{PQ} = \mathbf{QP}$ のときである。このときには $(\mathbf{P}+\mathbf{Q})^{k}$ が数の場合と同様に二項定理で展開できるからである。単位行列はあらゆる行列と可換であるため、$e^{\mathbf{A}t} = e^{-t/2}e^{\mathbf{K}t}$ となる。また $\mathbf{K}^2 = -\mathbf{I}$ より、$\mathbf{K}$ のべき乗は $\mathbf{I}, \mathbf{K}, -\mathbf{I}, -\mathbf{K}$ を巡回し、$e^{\mathbf{K}t}$ の級数は偶数次の項と奇数次の項、すなわち $\cos t$ と $\sin t$ の級数に分かれる。

$$e^{\mathbf{A}t} = e^{-t/2}\left(\cos t\,\mathbf{I} + \sin t\,\mathbf{K}\right) = e^{-t/2}\begin{pmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix}.$$

場そのものから導いたとおり、このフローは $e^{-t/2}$ 倍に縮小しながら t ラジアンだけ回転する。$\mathbf{x}_0 = (2, 0)$ から出発すると、$t = 2$ における状態は $e^{-1}(2\cos 2,\, 2\sin 2) = (-0.306,\, 0.669)$ となり、その長さは $2e^{-1} = 0.736$ である。この $\mathbf{A}$ に対して固有ベクトルと固有値 $-0.5 \pm \mathrm{i}$ は複素数になるが、積 $\mathbf{V}e^{\Lambda t}\mathbf{V}^{-1}$ は実数となる。

ここで一言注意しておくと、2 つの行列が可換でないときにはこの分解は成り立たない。ともに二乗するとゼロ行列になる $\mathbf{P} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ と $\mathbf{Q} = \mathbf{P}^{\top}$ に対して、$e^{\mathbf{P}}e^{\mathbf{Q}} = (\mathbf{I}+\mathbf{P})(\mathbf{I}+\mathbf{Q}) = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ であるのに対し、$e^{\mathbf{P}+\mathbf{Q}}$ の両対角成分は $\cosh 1 \approx 1.54$ となる。

$e^{\mathbf{A}t}$ に至る第 3 の道として、級数も固有ベクトルも必要としない複利の極限 $(\mathbf{I}+t\mathbf{A}/n)^{n}\,(n \to \infty)$ がある。各因子は、微小時間 t/n だけ場に沿って状態を進める操作である。この極限はどれほどの速さで収束するだろうか。固有基底ではこの積は $\mathbf{V}\,\mathrm{diag}\big((1+\lambda_j t/n)^{n}\big)\mathbf{V}^{-1}$ となるため、1 つの固有値について調べれば十分である。$\log(1+z) = z - z^2/2 + \cdots$ より、次が成り立つ。

$$\left(1 + \frac{\lambda t}{n}\right)^{n} = \exp\!\Big(n\log\big(1+\tfrac{\lambda t}{n}\big)\Big) \approx e^{\lambda t}\left(1 - \frac{(\lambda t)^2}{2n}\right),$$

したがって、各 $(1+\lambda_j t/n)^{n}$ は $e^{\lambda_j t}$ に収束し、その大きさに対する相対誤差は約 $|\lambda_j t|^2/(2n)$ となる。$t = 2$ におけるスパイラルでは、この比率は両固有値とも $2.5/n$ であり、$|e^{\lambda t}| = e^{-1}$ であるから、誤差は約 $0.9/n$ と予測される。次のコードセルでは、固有値分解から $e^{\mathbf{A}t}$ を計算し、3 つの n の値において複利計算による近似と比較する。

```
A = np.array([[-0.5, -1.0], [1.0, -0.5]])      # spiral: eigenvalues -0.5 +- i
t, x0 = 2.0, np.array([2.0, 0.0])
lam, V = np.linalg.eig(A)                      # A = V diag(lam) V^{-1}
expAt = (V @ np.diag(np.exp(lam * t)) @ np.linalg.inv(V)).real
x = expAt @ x0
print('eigenvalues of A:', lam.round(4))
print('x(2) =', x.round(4), f' length {np.linalg.norm(x):.6f}')
for n in [2 ** 10, 2 ** 16, 2 ** 24]:          # compounding (I + tA/n)^n
    err = np.abs(np.linalg.matrix_power(np.eye(2) + (t / n) * A, n) - expAt).max()
    print(f'n = 2^{int(np.log2(n))}: error {err:.1e}, n * error = {n * err:.2f}')
```

```
eigenvalues of A: [-0.5+1.j -0.5-1.j]
x(2) = [-0.3062  0.669 ]  length 0.735759
n = 2^10: error 8.8e-04, n * error = 0.90
n = 2^16: error 1.4e-05, n * error = 0.90
n = 2^24: error 5.4e-08, n * error = 0.90
```

状態 $(-0.3062,\, 0.6690)$ とその長さ 0.735759 は、手計算で求めた値と一致している。複利計算は予測された速度で収束し、それより速くはならない。誤差の最大成分に n を掛けた値は、$n = 2^{10}$ から $n = 2^{24}$ まで 0.90 のままである。これは金利を行列にした複利計算にほかならない。指数関数の定義としては優れているが、数値計算法としては不向きである。約 1700 万回の複利期間を経てもなお、誤差は 5×10^{-8} 残っている。

38.1.2.2 固有値による安定性

非結合化された形を見れば、線形系が落ち着くかどうかが一目でわかる。各固有値を $\lambda_j = a_j + \mathrm{i}\, b_j$ と書こう。そのモードの大きさは $|e^{\lambda_j t}| = e^{a_j t}$ である（$e^{\mathrm{i} b_j t} = \cos b_j t + \mathrm{i} \sin b_j t$ の絶対値が 1 であることによる）。実部が増大か減衰かを決め、虚部が回転を決める。対角化可能な $\mathbf{A}$ に対して、ここから次の分類が得られる。

- $\operatorname{Re} \lambda_j < 0$ がすべての j について成り立つならば、すべてのモードが減衰し、いかなる初期値からも $\mathbf{x}(t) \to \mathbf{0}$ となる。このとき原点は**漸近安定**（asymptotically stable）である。最も遅いモードが減衰率 $-\max_j \operatorname{Re} \lambda_j$ を決める。
- $\operatorname{Re} \lambda_j > 0$ がある j について成り立つならば、そのモードは指数関数的に増大し、$\mathbf{v}_j$ 方向の成分を持つあらゆる初期値は遠ざかっていく。このとき原点は**不安定**（unstable）である。
- その中間にあたる $\operatorname{Re} \lambda_j = 0$ のモードは、増大も減衰もしない。静止したままであるか、先ほどの回転の例のように永遠に周回し続ける。

複素共役ペア $a \pm \mathrm{i} b$ は一般の率を持つスパイラルであり、角速度 b で回転しつつ a の符号に応じて減衰または増大する。これを見るために、$\lambda = a + \mathrm{i} b$ の固有ベクトルを実ベクトルを用いて $\mathbf{v} = \mathbf{p} + \mathrm{i}\mathbf{q}$ と書く（$\mathbf{p}, \mathbf{q}$ は $b \neq 0$ より線形独立である）。$\mathbf{A}\mathbf{v} = (a + \mathrm{i} b)\mathbf{v}$ の実部と虚部をとると、次のようになる。

$$\mathbf{A}\mathbf{p} = a\mathbf{p} - b\mathbf{q}, \qquad \mathbf{A}\mathbf{q} = b\mathbf{p} + a\mathbf{q}.$$

したがって $\mathbf{A}$ は $\mathbf{p}$ と $\mathbf{q}$ が張る平面をそれ自身に写し、基底 $(\mathbf{q}, \mathbf{p})$ においては行列 $a\mathbf{I} + b\mathbf{K}$ で表される。するとスパイラルのときと同じ計算から次式が得られる。

$$e^{(a\mathbf{I} + b\mathbf{K})t} = e^{at}\left(\cos bt\, \mathbf{I} + \sin bt\, \mathbf{K}\right):$$

すなわち、e^{at} 倍のスケーリングと角度 bt の回転の合成である。この同じブロックを連続時間に流すのではなく離散時間で反復したものは、33.3.1.6 節に登場する。

2 次元では以上から 6 通りの相図が得られ、Figure 38.3 にそれぞれの代表的な場を 1 つずつ描いた。実固有値の場合、原点は**安定ノード**（stable node、両方とも負）、**不安定ノード**（unstable node、両方とも正）、または**鞍点**（saddle、正負が 1 つずつ）となる。複素共役ペアの場合は、**安定スパイラル**（stable spiral）、**不安定スパイラル**（unstable spiral）、あるいは実部がゼロのときの**渦心点**（center）となる。（固有値 0 は除外している。この場合は直線上のすべての点が静止点になる。）本節を通して用いている例は安定スパイラルである。

訓練において最もよく出会うのは上段のケースである。二次損失 $\tfrac{1}{2}\mathbf{x}^{\top}\mathbf{H}\mathbf{x}$ に対する連続時間の勾配降下法は線形の場 $\dot{\mathbf{x}} = -\mathbf{H}\mathbf{x}$ となり、対称行列 $\mathbf{H}$ の固有値はすべて実数であるため、このよう

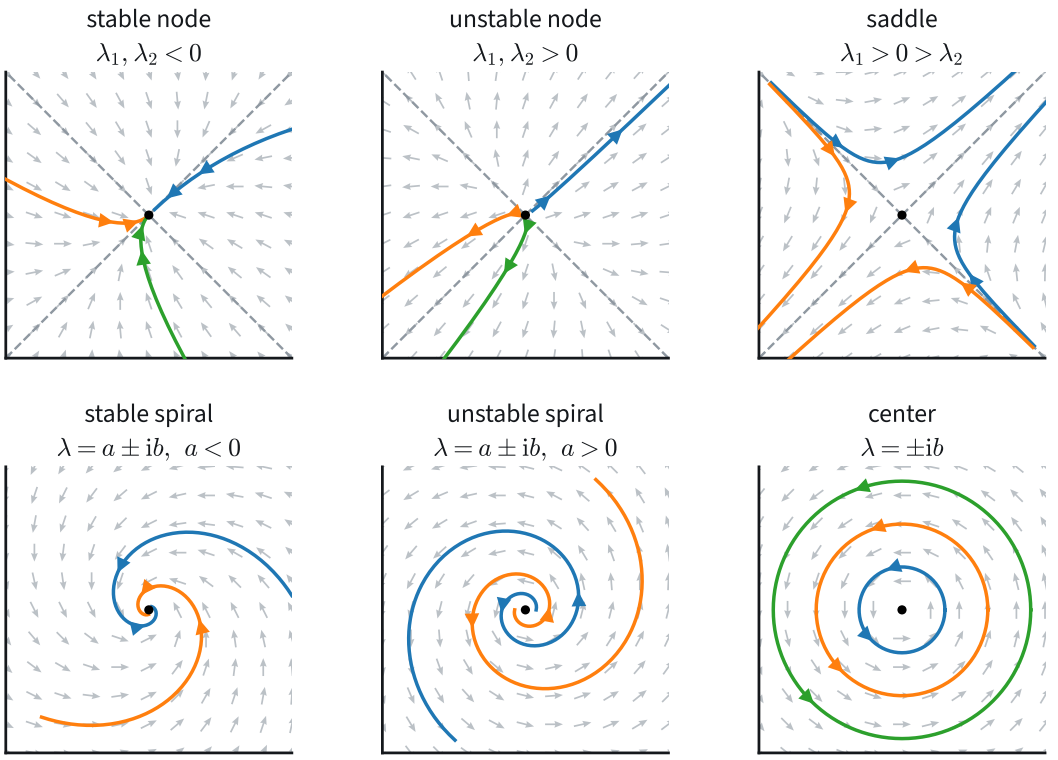


Figure 38.3: 6 種類の平面線形系。上段は実固有値の場合で、安定ノード（両方とも負）、不安定ノード（両方とも正）、鞍点（異符号）であり、破線は固有方向を表す。軌跡はこれらに沿って流入または流出する。下段は複素共役ペア $a \pm ib$ の場合で、安定スパイラル（$a < 0$）、不安定スパイラル（$a > 0$）、渦心点（$a = 0$、閉軌道）である。

な損失からはノードと鞍点しか生じない。回転が生じるには勾配ではない場が必要であり、たとえば最小構成の GAN における 2 人のプレイヤーは、渦心点となって平衡点の周りを周回し続ける（16.4 節）。

A の固有ベクトルが足りず対角化できない場合には、モードに多項式因子が加わって $t^k e^{\lambda t}$ の形になる。これが安定性の判定を左右するのは虚軸上の固有値に限られ、最小の具体例を演習問題 3 で扱う。

38.1.2.3 入力を伴う線形系

ここまでは、外部から系を押す力は存在しなかった。系列を読み込む状態空間モデル（12.4 節）の状態は、入力 $\mathbf{u}(t)$ を伴う線形方程式に従い、この入力は行列 $\mathbf{B}$ を介して状態に加わる。

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}(t), \qquad \mathbf{x}(0) = \mathbf{x}_0 .$$

行列指数関数はこの方程式も解くことができ、ある 1 つの着眼点から解が求まる。入力がなければ、量 $e^{-\mathbf{A}t}\mathbf{x}(t)$ は $\mathbf{x}_0$ に静止したままである。これは自由運動に沿って時刻 0 へと引き戻された状態にほかならない。入力があるとき、この量は入力が押す速さでしか動けない。実際、$\frac{d}{dt}e^{-\mathbf{A}t} = -e^{-\mathbf{A}t}\mathbf{A}$ であるから、積の微分法則より $\frac{d}{dt}\big(e^{-\mathbf{A}t}\mathbf{x}\big) = e^{-\mathbf{A}t}(\dot{\mathbf{x}} - \mathbf{A}\mathbf{x}) = e^{-\mathbf{A}t}\mathbf{B}\mathbf{u}$ となり、右辺にはもはや $\mathbf{x}$ が含まれない。0 から t まで積分して $e^{\mathbf{A}t}$ を掛けると、次式が得られる。

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 + \int_0^t e^{\mathbf{A}(t-s)}\,\mathbf{B}\mathbf{u}(s)\,ds. \tag{38.7}$$

これが**定数変化法**（variation of constants）の公式である。定数 $\mathbf{x}_0$（自由解 $e^{\mathbf{A}t}\mathbf{x}_0$ に現れるもの）が時間とともに変化することを許したためにこう呼ばれる。状態は、減衰した初期状態に過去のすべての入力（それぞれ到着してからの経過時間だけ減衰したもの）を足し合わせたものになる。

系列モデルは、あらゆる瞬間ではなくステップごとに 1 サンプルずつ入力を受け取る。そこで時間を長さ h のステップに区切り、グリッド時刻 $t_n = nh$ とグリッド状態 $\mathbf{x}_n = \mathbf{x}(t_n)$ を定め、各ステップのあいだ入力は一定、すなわち $\mathbf{u}(s) = \mathbf{u}_n$（$s \in [t_n, t_n + h)$）であると仮定しよう。これを**ゼロ次ホールド**（zero-order hold）という。$\mathbf{x}_n$ から時刻 t_n に出発する 1 ステップに対して Equation 38.7 を適用すると、定数 $\mathbf{u}_n$ が積分の外に出る。さらに $\tau = t_{n+1} - s$ と変数変換すれば残る積分は $\int_0^h e^{\mathbf{A}\tau}d\tau$ であり、これはすべてのステップで同じ行列になる。したがってグリッド上の状態は、いかなる近似も伴わずに次の線形漸化式（linear recurrence）に従う。

$$\mathbf{x}_{n+1} = \bar{\mathbf{A}}\,\mathbf{x}_n + \bar{\mathbf{B}}\,\mathbf{u}_n, \qquad \bar{\mathbf{A}} = e^{\mathbf{A}h}, \qquad \bar{\mathbf{B}} = \int_0^h e^{\mathbf{A}\tau}\,d\tau\;\mathbf{B} = \mathbf{A}^{-1}\big(e^{\mathbf{A}h} - \mathbf{I}\big)\mathbf{B}, \tag{38.8}$$

ただし最後の表式には $\mathbf{A}$ が正則（可逆）である必要がある。1 次元ではこれらは $\bar{a} = e^{h\lambda}$ および $\bar{b} = (e^{h\lambda} - 1)\,b/\lambda$ となり、12.4.1.2 節の状態空間モデルを構築する基礎となる係数である（そこではステップ幅を Δ、状態を h と書いている）。

ここで押さえておくべき要点は、$|e^{h\lambda}| = e^{h\,\mathrm{Re}\,\lambda} < 1$ が任意のステップ幅 $h > 0$ に対して、$\mathrm{Re}\,\lambda < 0$ である限りつねに成り立つことである。すなわち、元の方程式が安定であれば厳密な漸化式もつねに安定になる。これに対し、複利計算 1 期間分の因子 $1 + h\lambda$（途中で矢印を見直すことなく時間 h だけ場に沿って進むもの）と比較してみよう。$\dot{x} = -50x$ において $h = 0.05$ とすると、厳密な因子は $e^{-2.5} = 0.08$ であるのに対し、$1 + h\lambda = -1.5$ は状態の符号を反転させ

たうえに絶対値を拡大してしまう。このように場に沿って一歩ずつ進む手法がオイラー法であり、因子 -1.5 が訓練に何をもたらすかは 38.2 節で見る。

この漸化式が厳密であるのは各ステップのあいだ入力が一定のときに限られるが、実際の系列モデルはステップ間でも変動する信号のサンプルを読み込む。では、このホールドによる誤差はどれほど生じるのだろうか。次のコードセルでは、$\dot{x} = \lambda x + \sin t$ に対して $x(0) = 1$ から $T = 2$ まで、入力を各ステップ開始時の値に保持したスカラー漸化式を実行し、閉形式解と比較する。遅い減衰率 $\lambda = -1$ と速い減衰率 $\lambda = -50$ の両方について計算し、$1 + h\lambda$ を厳密な因子 $e^{h\lambda}$ と並べて表示する。

```
def zoh(lam, h, u, x0=1.0, T=2.0):
    """Exact steps of dx/dt = lam x + u(t), u held at u(t_n) over each step."""
    a, b = np.exp(h * lam), np.expm1(h * lam) / lam
    x = x0
    for n in range(round(T / h)):
        x = a * x + b * u(n * h)
    return x

def exact_sin(lam, T=2.0):
    """Solution of dx/dt = lam x + sin t with x(0) = 1, at time T."""
    c = 1 + lam ** 2
    return np.exp(lam * T) * (1 + 1 / c) - (lam * np.sin(T) + np.cos(T)) / c

print(f'{"lam":>5} {"h":>5} {"e^(h lam)":>10} {"1 + h lam":>10}'
      f' {"error at T":>11}')
for lam in [-1.0, -50.0]:
    for h in [0.5, 0.1, 0.02]:
        err = abs(zoh(lam, h, np.sin) - exact_sin(lam))
        print(f'{lam:5.0f} {h:5.2f} {np.exp(h * lam):10.3g}'
              f' {1 + h * lam:10.2f} {err:11.1e}')
```

```
  lam     h  e^(h lam)  1 + h lam  error at T
   -1  0.50      0.607       0.50     6.8e-02
   -1  0.10      0.905       0.90     9.7e-03
   -1  0.02       0.98       0.98     1.8e-03
  -50  0.50   1.39e-11     -24.00     1.6e-03
  -50  0.10    0.00674      -4.00     5.8e-04
  -50  0.02      0.368       0.00     9.2e-05
```

漸化式そのものは何ら近似をしていないため、生じる誤差は各ステップ中も正弦波が静止しているとみなすホールドの誤差だけである。この誤差はステップ幅にほぼ比例して縮まり、$\lambda = -1$ では 0.068、0.0097、0.0018（それぞれ $h = 0.5$、0.1、0.02 のとき）となる。注目すべき行は $\lambda = -50$、$h = 0.5$ のケースであり、ステップ幅は状態が過去を忘れる時間スケール $1/|\lambda|$ の 25 倍もの長さになっている。それでも厳密な因子は 1.4×10^{-11}、誤差は 0.0016 に収まっているのに対し、因子 $1 + h\lambda = -24$ を用いていれば状態は爆発していただろう。12.4.1.2 節のモデルはこの事実に支えられている。減衰率が負に保たれている限り、訓練がどのようなステップサイズを選ぼうとも漸化式が発散することはない。

38.1.2.4 過渡的増大

固有値による安定性判定には、まだ 1 つ補足しておくべき注意点がある。すべてのモードが減衰するとき状態はゼロへと向かうが、その途中で長さ $\|\mathbf{x}(t)\|$ が単調に減少するとは限らないのである。次の三角行列がその例を与える。

$$\dot{\mathbf{x}} = \begin{pmatrix} -1 & 10 \\ 0 & -2 \end{pmatrix} \mathbf{x}, \qquad \mathbf{x}_0 = (0, 1),$$

固有値は -1 と -2 で、どちらも負である。第 2 行には x_2 しか現れないため、$x_2(t) = e^{-2t}$ となる。すると第 1 行は $\dot{x}_1 = -x_1 + 10e^{-2t}$ という入力付きの減衰方程式になり、Equation 38.7 から次式が得られる。

$$x_1(t) = 10 \int_0^t e^{-(t-s)} e^{-2s}\, ds = 10\left(e^{-t} - e^{-2t}\right).$$

第 1 座標は 0 から出発し、その導関数 $10(2e^{-2t} - e^{-t})$ がゼロになる $e^{-t} = \frac{1}{2}$ まで上昇する。そこ（$t = \log 2 \approx 0.69$）では第 1 座標は $10(\frac{1}{2} - \frac{1}{4}) = 2.5$、また $x_2 = \frac{1}{4}$ となるため、長さ 1 で出発した状態の長さは $\sqrt{2.5^2 + 0.25^2} \approx 2.51$ に達する。それ以降 x_1 は減衰に転じ、長期的にはより遅いモードの減衰率に従う $10e^{-t}$ のように減衰する（Figure 38.4）。

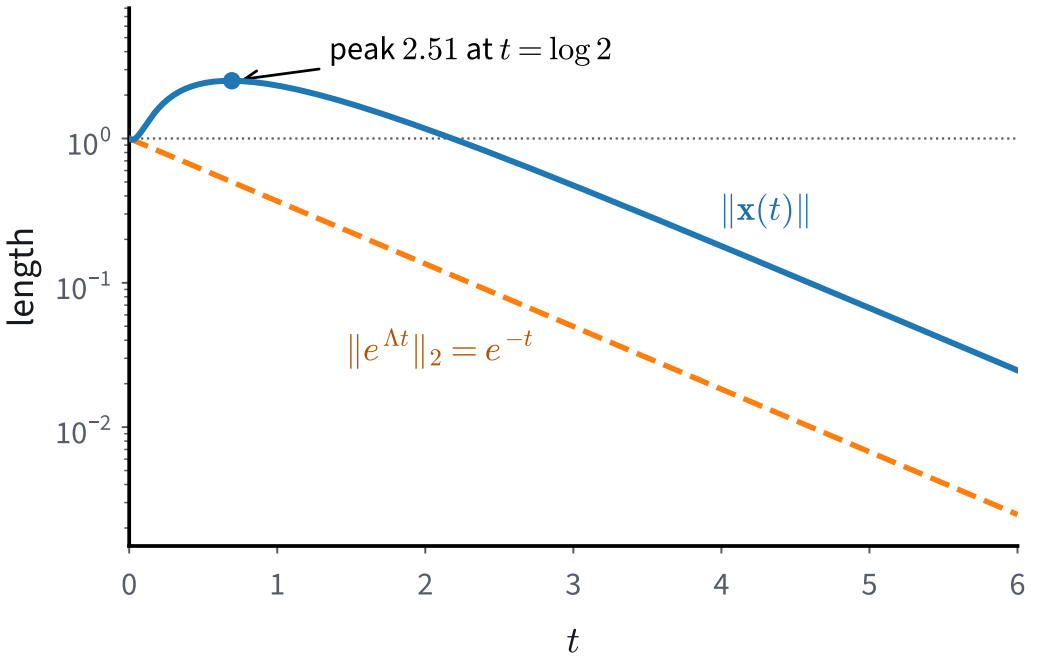


Figure 38.4: 三角行列からなる線形系の状態の長さ $\|\mathbf{x}(t)\|$（初期値 $\mathbf{x}_0 = (0, 1)$、対数軸）。2.51 まで上昇してから（$t = \log 2$）、$10e^{-t}$ のように減衰する。破線は $\|e^{\Lambda t}\|_2 = e^{-t}$（ただし $\Lambda = \mathrm{diag}(-1, -2)$）であり、同じ固有値を持つ対角行列のもとで単位長の初期状態が時刻 t にとりうる最大の長さを表す。2 本の曲線は最終的に同じ傾きに漸近するが、固有値が決めるのはこの終端の傾きだけである。

その理由は、離散時間の場合を詳しく扱った 33.4.2.3 節と同じである。固有ベクトル $\mathbf{v}_1 = (1, 0)$ と $\mathbf{v}_2 = (0.995, -0.0995)$ が張る直線のなす角はわずか 5.7° しかなく、短いベクトル $\mathbf{x}_0 = 10.0\,\mathbf{v}_1 - 10.05\,\mathbf{v}_2$ が 2 本の長いベクトルの差として表されているからである。これら 2 つの成分は異なる速さで減衰するため、両方が消え去る前に打ち消し合いが崩れる。固有値が与えるの

は漸近的な減衰率であって、過渡的増大（transient growth）の大きさではない。これは訓練曲線を眺めるときに心に留めておく価値がある。最終的に収束するモメンタムの実行であっても、最初の数ステップでは増大しうるのである（演習問題 5）。

38.1.2.5 不動点における線形化

閉形式解を持つ非線形の場はほとんどないが、場がゼロになる点の近傍では線形の場のように振る舞う。自律的な場の**不動点**とは、状態 $\mathbf{x}^\star$ であって $\mathbf{f}(\mathbf{x}^\star)=\mathbf{0}$ を満たすものであり、そこに置かれた粒子は決して動かない。連続微分可能な $\mathbf{f}$ と微小な変位 $\delta(t)=\mathbf{x}(t)-\mathbf{x}^\star$ に対して、テイラーの定理（34.2 節）より次式が得られる。

$$\dot{\delta}=\mathbf{f}(\mathbf{x}^\star+\delta)=\mathbf{J}\,\delta+o(\|\delta\|),\qquad \mathbf{J}=\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}^\star).$$

不動点の近傍では、変位よりも速く消える剰余項を除けば、場はそのヤコビ行列（Jacobian matrix）$\mathbf{J}$ が定める線形の場に一致する。この剰余項は無視してよいのだろうか。ほとんどの場合は無視できる。不動点は、$\mathbf{J}$ のどの固有値も実部がゼロでないとき**双曲型**（hyperbolic）であるという。双曲型不動点に対しては、ここでは証明なしに引用する Hartman–Grobman の定理 (Hartman 1960; Grobman 1959) により、$\mathbf{x}^\star$ の近傍における非線形場の軌跡は、連続な座標変換を除いて $\dot{\delta}=\mathbf{J}\delta$ の軌跡と一致する。したがって、ヤコビ行列の固有値から先ほどの分類によって局所的な安定性が決まる。

連続時間の勾配降下法 $\dot{\mathbf{x}}=-\nabla\mathcal{L}(\mathbf{x})$ に当てはめれば、これは見慣れた事実を別の言葉で言い換えたものにすぎない。不動点は損失の臨界点（critical point）であり、ヤコビ行列はヘッセ行列（Hessian matrix）にマイナスを付けた $-\mathbf{H}$ である。したがって、あらゆる方向に正の曲率を持つ極小点は安定ノードとなり、損失の鞍点はフローにおいても鞍点となる。

本節で具体的に計算する唯一の非線形の場として、減衰振り子を取り上げよう。重力加速度と長さの比 g/ℓ が 1 になるように時間の単位を選ぶと、最下点から測った角度 ϕ は $\ddot{\phi}=-\sin\phi-c\dot{\phi}$ に従う（$c\ge 0$ は減衰係数）。角速度 $\omega=\dot{\phi}$ を導入すれば 1 階の系になり、そのヤコビ行列と合わせて書くと次のようになる。

$$\dot{\phi}=\omega,\qquad \dot{\omega}=-\sin\phi-c\,\omega,\qquad \mathbf{J}(\phi,\omega)=\begin{pmatrix}0 & 1\\ -\cos\phi & -c\end{pmatrix}.$$

不動点は $\omega=0$ かつ $\sin\phi=0$ を満たす点であり、$\phi=0$ で真下に垂れ下がった状態と、$\phi=\pi$ で真上に倒立して釣り合った状態の 2 つである。

- 最下点では、$\mathbf{J}$ の特性多項式は $\lambda^2+c\lambda+1$ であり、その根は $\lambda=-c/2\pm \mathrm{i}\sqrt{1-c^2/4}$（$c<2$ のとき）である。$c=0.5$ では根は $-0.25\pm 0.97\,\mathrm{i}$ となり、安定スパイラルである。これは手を離した振り子の減衰振動にあたり、周期約 $2\pi/0.97$ で揺れながら振幅が $e^{-t/4}$ のように減衰していく。
- 最上点では、特性多項式は $\lambda^2+c\lambda-1$ であり、その根 $(-c\pm\sqrt{c^2+4})/2$ の積は -1 であるため、任意の c に対して 1 つは正、もう 1 つは負となる。$c=0.5$ では根は 0.78 と -1.28 であり、鞍点である。ある 1 本の曲線上の初期状態だけが最上点へと漸近する。これはちょうどよい初速度で振り上げられ、鉛筆を先端で立てたように釣り合って静止する振り子の運動に対応している。

ヤコビ行列とその固有値を求める同じ計算によって、ただ 1 つの例外を除き、あらゆる滑らかな場の不動点を分類できる。例外とは、実部が正の固有値は存在しないものの、虚軸上に固有値がある場合であり、このとき線形化だけでは安定性を判定できない。場 $\dot{x}=-x^3$ と $\dot{x}=x^3$ はど

ちらも原点でのヤコビ行列が 0 である。前者は安定で $x(t) = x_0/\sqrt{1 + 2x_0^2 t} \to 0$ となるが、後者は $t = 1/(2x_0^2)$ で無限大へと発散する。

この 2 つを見分けられたのは、たまたま方程式が解けたからにすぎない。1892 年の Lyapunov の着想は、解を求めずにこのような場合を判定する手段を与える。すなわち、運動によって減少することしかできない量を見つければよいのである。不動点 $\mathbf{x}^\star$ に対する **Lyapunov 関数**（Lyapunov function）とは、連続微分可能な状態の関数 V であって、$V(\mathbf{x}^\star) = 0$ かつ近傍の他のすべての点で $V(\mathbf{x}) > 0$ を満たし、いかなる軌跡に沿っても値が増加しないものをいう。

$$\frac{d}{dt} V(\mathbf{x}(t)) = \nabla V(\mathbf{x}(t))^{\top} \mathbf{f}(\mathbf{x}(t)) \le 0.$$

真ん中の式は連鎖律（chain rule）によるものであり、計算に必要なのは場だけであって軌跡そのものは不要である。V が小さい場所から出発した状態は、V がより大きい場所へは決して到達できないため、$\mathbf{x}^\star$ の近傍に留まり続け、不動点は安定となる。もし $\mathbf{x}^\star$ 以外の点で真の不等号が成り立つならば、近傍の状態は不動点へ収束する。先ほどの 2 つの 3 次の場に対して $V(x) = x^2$ ととると、その時間微分は $2x\dot{x}$ である。$\dot{x} = -x^3$ のもとではこれは $-2x^4$ となり、原点は安定である。一方、$\dot{x} = x^3$ のもとでは $+2x^4$ となり、今度はあらゆる軌跡に沿って x^2 が増大するため原点は不安定である。

実は、この考え方はすでに本節で使っていた。スパイラルでは長さの二乗 $V = \|\mathbf{x}\|^2$ が $\dot{V} = -V$ を満たしていた。連続時間の勾配降下法では、孤立した極小点での値を基準として測った損失が、その極小点の近傍における Lyapunov 関数になる。実際、$\frac{d}{dt}\mathcal{L} = \nabla \mathcal{L}^{\top} \dot{\mathbf{x}} = -\|\nabla \mathcal{L}\|^2$ が成り立つからである（演習問題 6）。そして振り子には運動エネルギーと位置エネルギーの和であるエネルギー $E = \tfrac{1}{2}\omega^2 + 1 - \cos\phi$ があり、最下点でゼロとなって次の速さで変化する。

$$\dot{E} = \omega\,\dot{\omega} + \sin\phi\,\dot{\phi} = \omega(-\sin\phi - c\,\omega) + \omega\sin\phi = -c\,\omega^2 \le 0.$$

減衰はエネルギーを奪うことしかできない。この事実は最下点から遠く離れた任意の大きさの揺れに対しても成り立ち、ヤコビ行列が何も語れない領域まで有効である。ただし折り返し点（$\omega = 0$）では一瞬だけエネルギーの変化が止まるため、不等号は狭義ではない。それにもかかわらず減衰振り子が最終的に静止することを示すにはもう一段の議論を要するが、ここでは割愛する。ヤコビ行列には機械的な計算手順があるのに対し、Lyapunov 関数にはそれがない。人間が見つけ出す必要があり、エネルギーや損失がその最初の候補となるのが常である。

ここまでは、すべて数式が仕事をしてくれた。しかしネットワークによって与えられる場には解の公式など存在せず、コンピュータにできるのは一歩ずつ進むこと、すなわち矢印を読み取り、少し進み、また矢印を読み取るという操作だけである。そのようなステップは模倣しようとしている本来の軌跡からどれほど逸れるのだろうか。また、元の方程式は破綻していないのにステップ計算だけが破綻することはあるのだろうか。

38.1.3 まとめ

常微分方程式（ordinary differential equation）は各点における速度を指定するものであり、最初の問いはそれによって運動が一意に定まるかどうかである。場が大域的リプシッツ連続であれば運動は定まる。すなわち、順方向にも逆方向にも各点を通る軌跡がただ 1 つ存在するため、フロー写像は全単射となり、場によってモデルを定義できる。線形系は $e^{\mathbf{A}t} = \mathbf{V} e^{\Lambda t} \mathbf{V}^{-1}$ によって解かれ、固有値の実部が減衰を、虚部が回転を決める。各ステップで一定に保たれた入力に対しては、方程式は厳密な漸化式へと変換され、元の方程式が安定であれば任意のステップサイズで安定となる。固有値が与えるのは長期的な減衰率であって過渡的増大の大きさではなく、双曲型不動点においてはヤコビ行列を通じて非線形の場を分類する。ヤコビ行列では判定できない場合

や変位が大きい場合には、エネルギーや損失のようにいかなる軌跡も増加させられない量である Lyapunov 関数が安定性を決定する。ただし、解の公式が存在しないときには、以上の議論だけでは軌跡を具体的に計算することはできない。

38.1.4 演習問題

1. **1 つの指数と 2 種類の破綻**。$\dot{x} = |x|^{\alpha}$（$\alpha > 0$）を考えよ。
   1. $0 < \alpha < 1$ のとき、$x(0) = 0$ を満たす解をすべて求め、$\alpha = \tfrac{1}{2}$ で Figure 38.2 の扇が得られることを確かめよ。Picard–Lindelöf の定理のどの仮定が、どの点で破れているか。
   2. $\alpha = 1$ のときは定理が適用できる。L はいくらか。また $x(0) = 0$ からの解は何か。
   3. $\alpha > 1$ かつ $x_0 > 0$ のとき、本文で $\dot{x} = x^2$ を解いたのと同様にして（$x^{1-\alpha}$ を微分せよ）方程式を解き、解が無限大に到達する時刻を求めよ。今度はどの仮定が破れているか。それにもかかわらず $x(0) = 0$ からの解は一意であることを示せ。（2 つの解はいずれも短い時間だけ $|x| \le 1$ 内に留まり、そこでは場は定数 α でリプシッツ連続である。短い区間上で証明の一意性のステップを適用し、再出発を繰り返す。）
   4. $\alpha = \tfrac{1}{2}$ の場合で時間を反転させよ。方程式 $\dot{x} = -\sqrt{x}$（$x \ge 0$）は、適切な単位のもとで底の穴から水が抜けるバケツの水位を表す（Torricelli の法則）。すべての解が有限時間で 0 に達し、そのままそこに留まることを示せ。空になったバケツを見てもいつ水が尽きたかはわからないが、フロー写像に関する系のどの主張が成り立たなくなっているか。
2. **軌跡の乖離**。ニューラル ODE が積分するような場（38.3 節）として、$\mathbf{f}(\mathbf{x}) = \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$ の形の場を考えよ。
   1. $\mathbf{f}$ がスペクトルノルムに関して定数 $L = \|\mathbf{W}_2\|\,\|\mathbf{W}_1\|$ でリプシッツ連続であることを示せ。証明では tanh のどの性質を使うか。tanh を ReLU に置き換えるとこの上界は変化するか。
   2. $\mathbf{x}(t)$ と $\mathbf{y}(t)$ を、リプシッツ定数 L を持つ自律的な場の 2 つの軌跡とする。$\|\mathbf{x}(t) - \mathbf{y}(t)\|^2$ を微分し、次を示せ。
      $$e^{-Lt}\|\mathbf{x}(0) - \mathbf{y}(0)\| \le \|\mathbf{x}(t) - \mathbf{y}(t)\| \le e^{Lt}\|\mathbf{x}(0) - \mathbf{y}(0)\|.$$
      下界は、異なる点から出発した 2 つの軌跡について何を意味するか。また上下の評価は、Φ_T とその逆写像のリプシッツ定数について何を意味するか。
3. **重複固有値**。行列 $\mathbf{A} = \left(\begin{smallmatrix} \lambda & 1 \\ 0 & \lambda \end{smallmatrix}\right)$（$\lambda$ は実数）は単一の固有値とただ 1 つの固有方向しか持たないため、Equation 38.6 は適用できない。
   1. $\mathbf{A} = \lambda \mathbf{I} + \mathbf{N}$ と書き、$\mathbf{N}^2 = \mathbf{0}$ を用いて $e^{\mathbf{A}t} = e^{\lambda t}\left(\begin{smallmatrix} 1 & t \\ 0 & 1 \end{smallmatrix}\right)$ を示せ。ここで指数関数を分解してよいのはなぜか。
   2. $\mathbf{x}_0 = (0, 1)$ から出発すると第 1 座標は $t e^{\lambda t}$ となる。$\lambda < 0$ のとき、その最大値とそれをとる時刻を求めよ。どの λ に対して、因子 t は λ の符号だけから下される判定を覆すか。
   3. 38.1.2.4 節の行列の 2 つの固有値を互いに近づけよ。$\left(\begin{smallmatrix} -1 & 1 \\ 0 & -1-\varepsilon \end{smallmatrix}\right)$ および $\mathbf{x}_0 = (0, 1)$ に対して $x_1(t)$ とその $\varepsilon \to 0$ における極限を計算せよ。この極限で 2 つの固有方向のなす角はどうなるか。またそのことは、対角化できない行列について何を示唆しているか。
4. [code] **ゼロ次ホールド**。スカラー方程式 $\dot{x} = \lambda x + b\,u(t)$（$\lambda < 0$）を考えよ。（12.4 節の離散化規則に関する演習問題では、この方程式を漸化式に変換する他の 3 つの手法とこのホールドを比較している。）
   1. Equation 38.8 を 1 次元に特殊化して $\bar{a} = e^{h\lambda}$ および $\bar{b} = (e^{h\lambda} - 1)\,b/\lambda$ を導き、$\bar{b}$ の $\lambda \to 0$ における極限を求めよ。定数入力に対して、漸化式と元の方程式が同じ値に落ち着くことを示せ。

2. 時間変化する入力のサンプル $u(t_n)$ に対して、漸化式は階段関数で駆動される方程式を厳密に解いている。この階段関数が平均的には $h/2$ だけ遅れた入力に相当することを説明し、h の 1 次の範囲で時刻 T における誤差の大きさが $\frac{h}{2}\left|\int_0^T e^{\lambda(T-s)}\, b\, u'(s)\, ds\right|$ になることを導け。$\lambda=-1$、$b=1$、$u=\sin$、$T=2$ に対してこの値を計算せよ。38.1.2.3 節のコードセルが $\lambda=-1$ に対して出力した 3 つの誤差とどれほど近いか。
3. 同項のコードセルを変更し、各ステップの中点で入力をサンプリングした $u(t_n+h/2)$ を用いるようにせよ。このとき $\lambda=-1$ に対する誤差は h に応じてどのように減少するか、またそれはなぜか。

5. [code] **過渡的増大と非正規性。**●本文では過渡的増大を起こす行列を 1 つ示した。本問ではその増大が固有ベクトル間の角度にどう依存するかを調べ、さらにモメンタムにおいて同じ現象に再び出会う。
1. $\mathbf{A}_\nu=\left(\begin{smallmatrix}-1 & \nu\\ 0 & -2\end{smallmatrix}\right)$ および $\mathbf{x}_0=(0,1)$ に対して $x_1(t)=\nu(e^{-t}-e^{-2t})$ を示し、x_1 の最大値とそれをとる時刻を求めよ。また、ν のうち $\|\mathbf{x}(t)\|$ が一度でも $\|\mathbf{x}_0\|$ を超える最小の値を数値的に求めよ。
2. $\mathbf{A}_\nu$ の 2 つの固有ベクトルのなす角を ν の関数として計算し、$\mathbf{x}_0$ を固有基底で表したうえで、2 つの大きな成分の打ち消し合いとしてこの増大を説明せよ。
3. $\max_t \|e^{\mathbf{A}_\nu t}\|_2$ を $\nu=1,5,10,20$ に対して数値的に計算し、$(0,1)$ からの軌跡のピーク値と比較せよ。前者が後者より小さくなりえないのはなぜか。
4. $\mathcal{L}(x)=\frac{\lambda}{2}x^2$ に対するヘビーボールモメンタムは漸化式 $x_{k+1}=(1+\beta-\eta\lambda)x_k-\beta x_{k-1}$ で表される。これは演習問題 6（38.2 節）において減衰振動子から導出されるものである。この漸化式は線形変換 $(x_{k+1},x_k)=\mathbf{M}(x_k,x_{k-1})$（ただし $\mathbf{M}=\left(\begin{smallmatrix}1+\beta-\eta\lambda & -\beta\\ 1 & 0\end{smallmatrix}\right)$）である。$\mathbf{M}$ が正規行列でないこと、すなわち $\mathbf{M}\mathbf{M}^\top\neq\mathbf{M}^\top\mathbf{M}$ を示せ（正規行列は直交する固有ベクトルを持ち、過渡的増大を起こさない。33.4.2.3 節参照）。$\beta=0.9$ とし、$\eta\lambda=0.1$ と $\eta\lambda=3$ における固有値を計算して、どちらの場合も絶対値が $\sqrt{\beta}\approx0.95$ であることを確かめよ。両方を静止状態 $x_0=x_{-1}=1$ から実行せよ。どちらの実行で $\|(x_{k+1},x_k)\|$ が減衰する前に増大するか、何ステップ増大してピークはいくつになるか、そして最初の 1 ステップで損失はどうなるか。この結果は、モメンタムの最初の数反復を見て発散と判断することについて何を示唆しているか。

6. **勾配フローは周回できない。** $\mathcal{L}$ を 2 回連続微分可能かつ下に有界な関数とし、勾配フロー $\dot{\mathbf{x}}=-\nabla\mathcal{L}(\mathbf{x})$ を考えよ。
1. $\frac{d}{dt}\mathcal{L}(\mathbf{x}(t))=-\|\nabla\mathcal{L}(\mathbf{x}(t))\|^2$ を示せ。ここから、いかなる軌跡も一度離れた点には再び戻らないこと、したがって勾配フローは閉軌道も渦心点も持たないことを導け。
2. 上の恒等式を 0 から T まで積分し、$\min_{t\le T}\|\nabla\mathcal{L}(\mathbf{x}(t))\|^2\le(\mathcal{L}(\mathbf{x}_0)-\inf\mathcal{L})/T$ を示せ。この上界の導出において不要だった損失への仮定はどれか。
3. ゲーム $\min_{x_1}\max_{x_2} x_1x_2$（$x_1$ は降下し、x_2 は値 x_1x_2 を上昇させる）の場は $\dot{x}_1=-x_2$、$\dot{x}_2=x_1$、すなわち回転 $\mathbf{K}$ である。この場がいかなる関数の負の勾配でもないことを示せ。（$\partial f_1/\partial x_2$ と $\partial f_2/\partial x_1$ を比較せよ。）このとき小問 (a) は、単一の損失を訓練することと 2 人のプレイヤーを互いに競わせて訓練することの違いについて何を語っているか。

7. **線形化の限界。**
1. 場 $\dot{x}=x^2$ は、本文の 2 つの 3 次の場と同様に原点でのヤコビ行列が 0 である。38.1.1.3 節で求めたその解は任意の初期値 x_0 に対して成り立つ。状態が原点へと近づくような初期値を求めよ。原点は安定か、不安定か、それとも片側からのみ安定か。Hartman–Grobman の定理はこれら 3 つの場について何を述べるか。
2. 減衰振り子において、最下点の静止点がスパイラルからノードへと変化する減衰係数 c を求めよ。$c=0.5$、$c=2$、$c=4$ に対して最下点近傍の相図の概形を描き、手を離した振り子がそれぞれの場合にどう動くかを述べよ。
3. 損失 $\mathcal{L}(w_1,w_2)=\frac{1}{2}(w_1w_2-1)^2$ は最小構成の深層線形ネットワークの損失であり、

2 つの重みの積を目標値 1 に当てはめるものである。その勾配フロー $\dot{\mathbf{w}} = -\nabla \mathcal{L}(\mathbf{w})$ を書き下し、すべての不動点を求め、原点を分類せよ。曲線 $w_1 w_2 = 1$ 上のすべての点において、ヤコビ行列の固有値が 0 と $-(w_1^2 + w_2^2)$ になることを示せ。不動点が曲線をなしているとき、その上のいかなる点も双曲型になりえないのはなぜか。また、それでも負の固有値から何がわかるか。

38.2 数値ソルバ

ネットワークを訓練したことがあれば誰でも、学習率（learning rate）が大きすぎたために計算が発散して爆発する様子を目にしたことがあるだろう。鋭い谷やプラトーを持つ損失曲面（loss surface）のせいにしたくなるかもしれない。しかし、考えられる限り最も単純な損失上でも同じことが起こる。ボウル型の損失 $\mathcal{L}(x) = 25x^2$ に対する勾配降下法（gradient descent）は更新ステップ $x \leftarrow x - \eta \cdot 50x$ をとるが、このボウルは唯一の最小点を持ち何ひとつ難しい点がないにもかかわらず、$\eta = 0.04$ を超えるあらゆる学習率で反復点は際限なく増大する。損失に問題がないとすれば、何が問題なのだろうか。

その答えは、より古い分野からもたらされる。勾配降下法とは、「傾きに比例する速さで下り方向へ進め」と述べる微分方程式（differential equation）$\dot{x} = -50x$ を 1 ステップずつ解く操作である。これは 38.1 節で解いた減衰方程式を 50 倍速くしたものであり、その厳密解 $x(t) = e^{-50t} x_0$ はどのような初期値から出発してもゼロへと減衰する。Euler は 1768 年に、矢印の向きに短い時間 h だけ進んでは矢印を読み直すことでこのような方程式を解く手法を提案したが、学習率はこの h の役割を果たしている。1 ステップ進めるごとに x は $1 - 50h$ 倍になる。$h = 0.01$ ではこの因子は 0.5 であり、計算される解は真の解と同様に減衰する。一方、$h = 0.05$ では -1.5 となり（厳密な因子は $e^{-2.5} = 0.08$ である）、計算される解は符号を反転させながら増大していく。方程式自体は安定であるが、ステップが安定ではないのである。前節では方程式の固有値（eigenvalue）がその厳密な運動を決めていたが、本節では同じ固有値が、その運動を追跡できるステップ幅の上限を決めることになる。

深層学習は、ステップを進めるあらゆる場面で、方程式とその離散ステップとの間にあるこの乖離に直面する。すなわち、勾配降下法（9.1 節）、残差ネットワーク（residual network）の各層（7.4 節）、そして拡散モデル（diffusion model）やフローマッチング（flow matching）モデルのサンプラ（17.5 節）である。サンプラは各ステップでネットワークを 1〜2 回評価するため、ステップ数が画像 1 枚を生成するコストを決める。

逐次的な数値解法の価値は、3 つの問いによって決まる。真の解からどれほど逸れるのか。同じ計算量でそのずれをどうすれば減らせるのか。そして、誤差が減衰せずに増大し始めるまでにステップ幅をどこまで大きくとれるのか。実は、これら 3 つの問いすべてにたった 1 つの対象が答えてくれる。1 ステップで解に掛けられる因子であり、先のボウルに対するオイラー法では $1 - 50h$ であった量である。本節の最後では勾配降下法に立ち返り、より優れたソルバを用いれば訓練はどうなるのかを問う。

38.2.1 前進オイラー法

軌跡（trajectory）とは、始点に道中で蓄積された速度を足し合わせたものである。短いステップの上で、Euler の提案は考えうる限り最も素朴な近似をとる。すなわち、速度がステップの始点での値をずっと保ち続けるとみなすのである。まず、答えのわかっている方程式でこの手法がどう振る舞うかを見てみよう。

$\dot{x} = x$ を $x_0 = 1$ から $T = 1$ まで解くことを考える。真の解は $e = 2.718$ である。区間を n 個のステップ（各ステップの長さ $h = 1/n$）に分割し、各ステップではその始点での速度で進むと

すると、$x \leftarrow x + hx = (1+1/n)x$ となる。n ステップ後には $x_n = (1+1/n)^n$ が得られる。これは 1 ドルを年利 100%、年 n 回複利で運用したときの元利合計にあたる。

n	1	2	4	8	16	32	365
$(1+1/n)^n$	2.000	2.250	2.441	2.566	2.638	2.677	2.7146
誤差	0.718	0.468	0.277	0.153	0.080	0.041	0.0037

隣り合う誤差の比は $1.53, 1.69, 1.82, 1.90, 1.95$ となっており、ステップ数を 2 倍にすると誤差は次第に半分へと近づいていく。それでも、毎日複利計算したところで e ドルには 3 分の 1 セント以上届かない。

この収束の遅い極限にはすでに前節で出会っている。複利型の式 $(\mathbf{I} + t\mathbf{A}/n)^n$（38.1 節）は、名前を付ける前のオイラー法そのものであり、線形な場に沿って n ステップ（ステップ幅 t/n）進めたものにほかならない。そこで求めた展開 $(1+\lambda t/n)^n \approx e^{\lambda t}(1 - \frac{(\lambda t)^2}{2n})$ において $\lambda t = 1$ とおけば、先の 1 ドルの誤差が得られる。その大きさは約 $e/(2n) = 1.36\,h$ である。このように h に比例する誤差を、以下では 1 次（次数 1）と呼ぶ。

一般のベクトル場 $\mathbf{f}(\mathbf{x}, t)$ に対し、ステップサイズ（step size）を h、時間グリッドを $t_n = nh$（これ以降、n はそれまでに進んだステップ数を表す）とおくと、**前進オイラー法**（forward Euler）は次式で定義される。

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n, t_n). \tag{38.9}$$

この手法は現在地での矢印に沿って時間 h だけ進み、そこで再び場を読み取る。言い換えれば、真の解のテイラー展開（Taylor expansion）を 1 次の項で打ち切ったものであり、1 ステップあたりの誤差は切り捨てた最初の項になる。解に沿って $\|\ddot{\mathbf{x}}(t)\| \le M$ が成り立つならば、次が成り立つ。

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + h\,\mathbf{f}(\mathbf{x}(t_n), t_n) + \mathbf{r}_n, \qquad \|\mathbf{r}_n\| \le \tfrac{Mh^2}{2}.$$

剰余項 $\mathbf{r}_n$ は**局所誤差**（local error）であり、真の解の上から出発したときに 1 ステップで生じるずれを表す。本当に知りたいのは**大域誤差**（global error）、すなわち固定された終端時刻 T まで全ステップを進めた後の真の解からの距離である。局所誤差が h^2 のオーダーであっても、大域誤差が h^2 のオーダーになるとは限らない。T に到達するまでには T/h ステップを要し、それぞれの局所誤差が累積するうえ、各ステップがすでに生じた誤差を増幅しうるからである。こうして h のべきが 1 つ失われる。その証明は短く、定数そのものよりも証明の構造が重要である。

命題（オイラー法は次数 1 で収束する）。 **$\mathbf{f}$ は定数 L で $\mathbf{x}$ に関してリプシッツ連続であるとする。すなわち、$\|\mathbf{f}(\mathbf{x},t) - \mathbf{f}(\mathbf{y},t)\| \le L\|\mathbf{x}-\mathbf{y}\|$ がすべての $\mathbf{x}$、$\mathbf{y}$、t に対して成り立つとする。また、$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},t)$ の初期値 $\mathbf{x}(0) = \mathbf{x}_0$ からの解が $\|\ddot{\mathbf{x}}(t)\| \le M$ を区間 $[0,T]$ 上で満たすとする。このとき、同じ初期値から出発するオイラー法の反復点 Equation 38.9 は次を満たす。**

$$\max_{n \le T/h} \|\mathbf{x}_n - \mathbf{x}(t_n)\| \;\le\; \frac{Mh}{2L}\left(e^{LT} - 1\right) \;=\; O(h).$$

証明。 $\varepsilon_n = \|\mathbf{x}_n - \mathbf{x}(t_n)\|$ とおき、更新 Equation 38.9 から上のテイラー展開の恒等式を引くと、次を得る。

$$\varepsilon_{n+1} \le \underbrace{\|\mathbf{x}_n - \mathbf{x}(t_n)\| + h\,\|\mathbf{f}(\mathbf{x}_n,t_n) - \mathbf{f}(\mathbf{x}(t_n),t_n)\|}_{\le (1+hL)\,\varepsilon_n \text{ by Lipschitz}} + \underbrace{\|\mathbf{r}_n\|}_{\le Mh^2/2}.$$

各ステップは既存の誤差を高々 $1+hL$ 倍し、そこへ高々 $Mh^2/2$ を加える。$\varepsilon_0 = 0$ から漸化式を展開すると等比級数の和になる。

$$\varepsilon_n \le \frac{Mh^2}{2}\sum_{j=0}^{n-1}(1+hL)^j = \frac{Mh^2}{2}\cdot\frac{(1+hL)^n - 1}{hL} \le \frac{Mh}{2L}\left(e^{LT}-1\right),$$

ここで $(1+hL)^n \le e^{nhL} \le e^{LT}$ を用いた。■

$\dot{x} = x$ を区間 $[0,1]$ 上で考えると、$M = \max|\ddot{x}| = e$ かつ $L = 1$ であるから、この上界は $\frac{e}{2}(e-1)h = 2.3\,h$ となり、真の誤差 $1.36\,h$ に対して次数は正しいものの定数は悲観的である。この上界の構造はあらゆるソルバで繰り返し現れる。1 ステップあたり h^{p+1} のオーダーの局所誤差は、大域誤差としては h^p のオーダーになる。これは T/h ステップにわたって累積し、過去の誤差をダイナミクスがどれほど強く増幅するかを測る因子 e^{LT} によって膨らむためである。大域誤差が h^p のオーダーとなる手法は**次数**（order）p を持つといい、オイラー法の次数は 1 である。

因子 e^{LT} が大雑把な部分である。定数 L は場の傾きの急さだけを記録し、どちら向きに傾いているかは区別しないため、この上界では軌跡を互いに引き離す場と引き寄せる場を見分けられない。ボウルの減衰方程式 $\dot{x} = -50x$ を $[0,1]$ 上で考えると、この因子は $e^{50} \approx 5\times 10^{21}$ となり、小さなステップ幅をとれば減衰をうまく追跡できるにもかかわらず、コンピュータが実行しうるいかなるステップ幅でも役に立たない上界になってしまう。収縮する場には専用の解析が必要であり、次数を高める手法を見た後でそこへ進むことにする。

38.2.2 ルンゲ・クッタ法

オイラー法は、軌跡が曲がって離れていくにもかかわらずステップ始点での傾きに固執する。これに対し、ルンゲ・クッタ法はステップ内部の複数の点で傾きを測り、それらの測定値を組み合わせる。Figure 38.5 に、ステップ幅 $h = 1$ で $\dot{x} = x$ を $x = 1$ から 1 ステップ進めた様子を示す。

38.2.2.1 中点法、ホイン法、RK4

最も単純な改良である**中点法**（midpoint method）は、中点での傾きを測るためだけにオイラー法で半ステップ進み、得られた傾きを使って本来のステップを進める。時間に依存しないベクトル場を $\mathbf{f}(\mathbf{x})$ と書けば、更新式は $\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}\big(\mathbf{x}_n + \tfrac{h}{2}\mathbf{f}(\mathbf{x}_n)\big)$ となる。1 次元でテイラー展開を 1 行計算すれば、なぜこれで精度が上がるのかがわかる。

$$x + hf\big(x + \tfrac{h}{2}f(x)\big) = x + hf + \tfrac{h^2}{2}f'f + O(h^3).$$

真の解の展開は $x(t+h) = x + h\dot{x} + \frac{h^2}{2}\ddot{x} + O(h^3)$ であり、ここで $\ddot{x} = \frac{d}{dt}f(x(t)) = f'f$ であるから、欠落していた h^2 の項が正しく一致している。したがって 1 ステップあたり 2 回の場評価で、局所誤差は $O(h^3)$、大域的な次数は 2 となる。

中点法は最初の評価を半ステップに使った。これに対し、**ホイン法**（Heun's method）は試行的にオイラー法で 1 ステップ分すべて進み、ステップの両端における傾きを平均する。

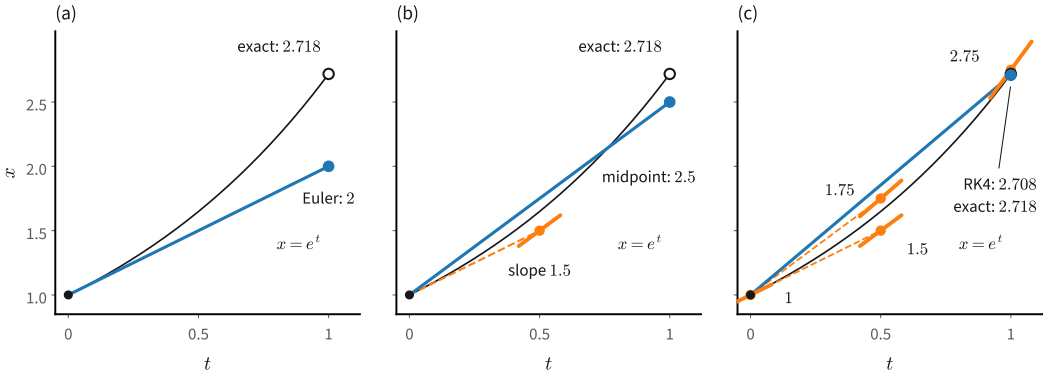


Figure 38.5: ステップ幅 $h = 1$ による $\dot{x} = x$（初期値 $x = 1$）の 1 ステップ。(a) オイラー法は始点の傾きに沿って進み、2 に到達する。(b) 中点法は中点での傾き 1.5 を測るためだけに半ステップ進み、その傾きを用いて 2.5 に到達する。(c) RK4 は 4 つの傾き 1、1.5、1.75、2.75 を測り、重み $\frac{1}{6}(1, 2, 2, 1)$ で平均して 2.708 に到達する。厳密値は $e = 2.718$ である。

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \tfrac{h}{2}\Big[\mathbf{f}(\mathbf{x}_n) + \mathbf{f}\big(\mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n)\big)\Big]. \tag{38.10}$$

中点法が積分における中点則（midpoint rule）に対応するなら、ホイン法は台形則（trapezoid rule）に対応する。同じ 1 行のテイラー展開から、同じ 2 回の評価で同じ次数 2 を持つことがわかる。この 2 つのうち覚えておくべきなのはホイン法である。多くの拡散モデルのサンプラがこの手法を採用しているからである（17.5 節）。

4 回の傾きの測定を用いると、1901 年の Kutta に遡る科学技術計算の標準的な汎用手法である**古典的ルンゲ・クッタ法（RK4）**（classical Runge–Kutta）が得られる。

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(\mathbf{x}_n,\, t_n),\\
\mathbf{k}_2 &= \mathbf{f}(\mathbf{x}_n + \tfrac{h}{2}\mathbf{k}_1,\, t_n + \tfrac{h}{2}),\\
\mathbf{k}_3 &= \mathbf{f}(\mathbf{x}_n + \tfrac{h}{2}\mathbf{k}_2,\, t_n + \tfrac{h}{2}),\\
\mathbf{k}_4 &= \mathbf{f}(\mathbf{x}_n + h\,\mathbf{k}_3,\, t_n + h),\\
\mathbf{x}_{n+1} &= \mathbf{x}_n + \tfrac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right).
\end{aligned} \tag{38.11}$$

この更新式は、中点での 2 つの測定値に 2 倍の重みを付けて 4 つの傾きを平均している。これらの重みは 3 次多項式を厳密に積分するシンプソン則 $\int_a^b g \approx \frac{b-a}{6}\big(g(a) + 4g(\frac{a+b}{2}) + g(b)\big)$ の重みそのものであり、中点に割り当てられた重みが 2 回の測定に等分されている。

38.2.2.2 安定性関数

なぜ次数が 4 になるのだろうか。線形な場の上では、1 回の計算で答えが出る。スカラー方程式 $\dot{y} = \lambda y$（λ は実数または複素数）を考えよう。これは**テスト方程式**（test equation）と呼ばれ、あらゆる線形場を一度に代表する方程式である。というのも、$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ の各モードは、λ を行列 $\mathbf{A}$ の固有値としてこの方程式に従うからである（38.1 節）。真の解には因子 $e^{h\lambda}$ が時間 h の間に掛かる。一方、各数値手法は 1 ステップごとに y に $z = h\lambda$ の定まった関数を掛けるため、その関数を e^z と比較できる。

命題（安定性関数）。テスト方程式において $z = h\lambda$ とおくと、各手法の 1 ステップは y にそれぞれの安定性関数（stability function）$R(z)$ を掛ける。

$$R_{\text{Euler}}(z) = 1 + z, \qquad R_{\text{mid}}(z) = R_{\text{Heun}}(z) = 1 + z + \frac{z^2}{2},$$
$$R_{\text{RK4}}(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}. \tag{38.12}$$

証明。 RK4 では 4 つの測定値はそれぞれ直前の測定値を代入することで得られ、$k_1 = \lambda y$、$k_2 = \lambda y(1 + \tfrac{z}{2})$、$k_3 = \lambda y(1 + \tfrac{z}{2} + \tfrac{z^2}{4})$、$k_4 = \lambda y(1 + z + \tfrac{z^2}{2} + \tfrac{z^3}{4})$ となる。したがって更新後の値は $y + \tfrac{z}{6}\left(6 + 3z + z^2 + \tfrac{z^3}{4}\right) y$ となり、上の多項式と y の積に一致する。オイラー法では $y + zy$ となる。中点法では $y + h\lambda(y + \tfrac{z}{2}y)$、ホイン法では $y + \tfrac{z}{2}(y + (1 + z)y)$ となり、いずれも $(1 + z + \tfrac{z^2}{2})y$ である。$\blacksquare$

各 R は厳密な因子 e^z のテイラー多項式をその手法の次数で打ち切ったものになっている。RK4 では $e^z - R_{\text{RK4}}(z) = z^5/120 + O(z^6)$ であるため、この方程式において 1 ステップの誤差は $O(h^5)$、T/h ステップ後の誤差は $O(h^4)$ となる。4 回連続微分可能かつ $\mathbf{x}$ に関してリプシッツ連続な非線形の場に対しても、より長いテイラー展開の照合によって同じ次数が得られ、中点法とホイン法は次数 2、RK4 は次数 4 となる (Hairer et al. 1993)。4 重に入れ子になった評価の展開は何ページにも及び、そこから得られる示唆も少ないため、ここでは証明しない。本項で押さえておくべき対象は安定性関数である。e^z に $z = 0$ の近傍でどれほど密接に追随するかがその手法の精度であり、ゼロから離れた場所でどれほど大きくなるかがどのステップ幅で安定になるかを決める。

38.2.2.3 同一コストでの次数比較

残る課題は、実際に次数を測定し、場の評価回数を揃えて各手法を比較することである。時間に依存しない場 `f` に対して、この手法族の両端にあたる 2 つのソルバはコードではそれぞれ数行で書ける。

```
def euler(f, x0, T, n):
    """Forward Euler: n steps of size h = T/n."""
    x, h = np.asarray(x0, dtype=float), T / n
    for _ in range(n):
        x = x + h * f(x)
    return x

def rk4(f, x0, T, n):
    """Classical Runge-Kutta: four slope measurements per step."""
    x, h = np.asarray(x0, dtype=float), T / n
    for _ in range(n):
        k1 = f(x)
        k2 = f(x + 0.5 * h * k1)
        k3 = f(x + 0.5 * h * k2)
        k4 = f(x + h * k3)
        x = x + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
    return x
```

両対数軸上では error $\approx Ch^p$ は傾き p の直線になるため、測定される傾きは 1 と 4 になるはずである。これを Figure 38.1 のスパイラル $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$（固有値は $-0.5 \pm \mathrm{i}$）で検証しよう。この軌

跡上の点は単位速度で回転しながら減衰率 $\frac{1}{2}$ で長さを失うため、$\mathbf{x}_0 = (2, 0)$ から出発したときの $T = 2$ における厳密な状態は $e^{-1}(2\cos 2,\, 2\sin 2) = (-0.306,\, 0.669)$ であり、その長さは 0.736 である（38.1 節）。

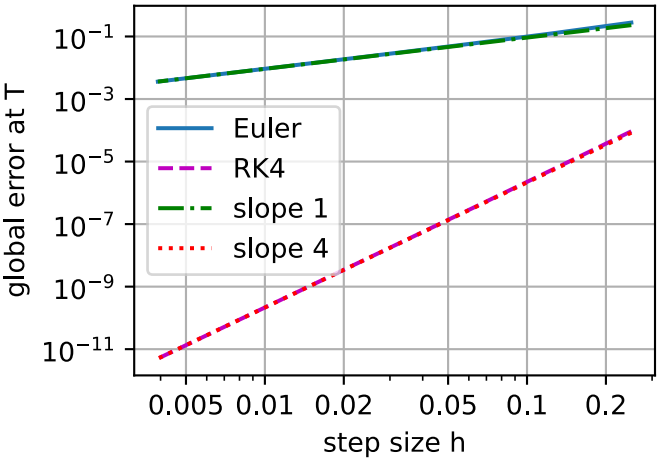
```
A = np.array([[-0.5, -1.0], [1.0, -0.5]])      # the spiral, eigenvalues -0.5 +- i
x0, T = np.array([2.0, 0.0]), 2.0
x_true = np.exp(-1) * np.array([2 * np.cos(2), 2 * np.sin(2)])  # e^{AT} x0
```

線形な場の上では、傾きだけでなく定数まで予測できる。このスパイラル上では、厳密な運動と同様にどちらのソルバの 1 ステップも回転とスケーリングの合成になる。したがって、各モードの因子がある割合だけずれるソルバは、真の終端点の長さ 0.736 に対しても同じ割合だけずれる。オイラー法におけるその割合は、複利計算で求めた $|\lambda T|^2/(2n)$ である。ここでは $|\lambda T|^2 = 5$ であるから、オイラー法の誤差は終端の長さの $2.5/n$ 倍、すなわち $1.84/n$ になるはずである。

RK4 における割合は安定性関数から導かれる。1 ステップあたりのずれの割合は $|z|^5/120$（ただし $|z| = |\lambda T|/n = \sqrt{5}/n$）であり、$n$ ステップ分の誤差が足し合わされる。その合計は終端の長さの $5^{5/2}/(120\,n^4)$ 倍、すなわち $0.343/n^4$ となる。次のコードセルでは、ステップ幅を 6 回半減させ、8 ステップから 512 ステップまで計算する。

```
f = lambda x: A @ x                            # the spiral field
ns = 2 ** np.arange(3, 10)                     # 8, 16, ..., 512 steps
hs = T / ns
err_eu = np.array([np.linalg.norm(euler(f, x0, T, n) - x_true) for n in ns])
err_rk = np.array([np.linalg.norm(rk4(f, x0, T, n) - x_true) for n in ns])
print('measured slope, Euler:', f'{np.polyfit(np.log(hs), np.log(err_eu), 1)[0]:.3f}')
print('measured slope, RK4  :', f'{np.polyfit(np.log(hs), np.log(err_rk), 1)[0]:.3f}')
print(f'error at 512 steps: Euler {err_eu[-1]:.1e}, RK4 {err_rk[-1]:.1e}')
print('RK4 with 128 steps (the same 512 evaluations):',
      f'{np.linalg.norm(rk4(f, x0, T, 128) - x_true):.1e}')
d2l.plot(hs, [err_eu, err_rk, err_eu[-1] * (hs / hs[-1]),
              err_rk[-1] * (hs / hs[-1]) ** 4],
         'step size h', 'global error at T', xscale='log', yscale='log',
         legend=['Euler', 'RK4', 'slope 1', 'slope 4'])
```

```
measured slope, Euler: 1.040
measured slope, RK4  : 4.022
error at 512 steps: Euler 3.6e-03, RK4 5.0e-12
RK4 with 128 steps (the same 512 evaluations): 1.3e-09
```



直線当てはめで得られた傾きは 1.04 と 4.02 であり、1 と 4 をわずかに上回っている。これは当てはめに $h = 0.25$ という粗いステップも含まれており、展開の次の項がまだ効いているためである。512 ステップにおける誤差は 3.6×10^{-3} と 5.0×10^{-12} であり、$1.84/n$ と $0.343/n^4$ の予測と一致する。もっとも、傾きだけを見るとコストが隠れてしまう。RK4 は 1 ステップあたり場を 4 回評価するため、公平に比較するにはステップ数を 4 分の 1 にしなければならない。同じ 512 回の場評価、すなわち 128 ステップで比較すると、RK4 の誤差は 1.3×10^{-9} に達し、オイラー法より 6 桁小さい。オイラー法の法則 $1.84/n$ でこれに並ぶには 10 億ステップ以上が必要になる。一般に、次数 p の手法では $\varepsilon^{-1/p}$ に比例するステップ数が誤差 ε の達成に必要となる。

実用的なソルバは、計算を進めながらステップサイズも適応的に選ぶ。Dormand–Prince 法のような**埋め込み型ペア**（embedded pair）は、同じ場評価から次数 4 と次数 5 の解を同時に計算し、両者の差を局所誤差の推定値として用いる (Dormand and Prince 1980)。そして推定誤差が大きい箇所ではステップを短くし、小さい箇所では長くする。演習問題 1 では、このような制御器のうち最も単純なものを構築する。

ここまでの手法はいずれも各ステップを白紙から始め、直前のステップで測定した傾きを捨ててしまう。これに対し、**多段階**（multistep）法は過去の測定値を保持する。最も単純な 2 段階の**アダムス・バッシュフォース法**（Adams–Bashforth method）は、各ステップで場を 1 回だけ評価し、直近 2 回の評価値から外挿する。

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\left[\tfrac{3}{2}\,\mathbf{f}(\mathbf{x}_n) - \tfrac{1}{2}\,\mathbf{f}(\mathbf{x}_{n-1})\right]. \tag{38.13}$$

角括弧の中身は $\mathbf{f}(\mathbf{x}_n) + \tfrac{1}{2}[\mathbf{f}(\mathbf{x}_n) - \mathbf{f}(\mathbf{x}_{n-1})]$ と書け、2 つの評価値を通る直線をさらに半ステップ先まで延長した値になっている。これはこれから進むステップの中点における傾きの推定値にあたるため、2 回目の場評価を行う代わりに記憶しておいた値から中点の傾きを得る中点法に相当し、オイラー法と同じコストで次数 2 を達成する。ただし失われる点も 2 つある。最初のステップには 1 つ前の値がないため 1 段階法で進める必要があり、また重み $\tfrac{3}{2}$ と $-\tfrac{1}{2}$ は等間隔ステップでしか成り立たないためステップサイズを変える際には再計算が必要になる。しかし 1 回の評価が大規模ネットワークの順伝播 1 回分にあたる場面では、計算量の節約がこの両方の欠点を上回るため、17.5 節の多段階サンプラはこの更新ステップを基盤として構築されている。

38.2.3 硬さと陰的手法

次数が測るのは、ステップ幅が小さいときに 1 ステップがどれほど精密かという性質である。これに対し、安定性は別の問いを立てる。すなわち、多数のステップにわたる誤差が減衰して消えるのか、それとも雪だるま式に増幅するのかという問いである。オイラー法や RK4 では、どれほど低い精度しか求めていなくても、この安定性によってステップサイズに上限が課される。

この問題を示す最小の系は、互いに分離した 2 つのモードを持つ $\dot{\mathbf{x}} = \operatorname{diag}(-50, -1)\,\mathbf{x}$ を初期値 $(1, 1)$ から $T = 1$ まで追跡するものである。その時刻までに速いモードは $e^{-50} \approx 10^{-22}$ まで減衰し、遅いモードは $e^{-1} = 0.368$ に留まっている。$t \approx 0.1$ 以降は遅いモードしか残っておらず、$h = 0.1$ のステップ幅でも十分に追跡できるはずである。ところが速い座標は冒頭の方程式 $\dot{x} = -50x$ そのものであり、積分を続ける限りオイラー法は毎ステップそこに $1 - 50h$ を掛け続ける。興味のある解が変化する速さに比べてヤコビ行列の固有値の一部に絶対値の大きな負の実部があるとき、その系は**硬い**（stiff）という。とうの昔に減衰し切ったモードが、依然としてステップ幅を決め続けてしまうのである。

どのようなステップ幅なら許されるのだろうか。それに答えるのが安定性関数である。テスト方程式 $\dot{y} = \lambda y$ の解が減衰するのは $\operatorname{Re}\lambda < 0$ のとき、かつそのときに限られるが、数値手法による解は初期値 y_0 から $R(z)^n y_0$（n ステップ後）へと変化する。したがって、計算されるモードが減衰する必要十分条件は $|R(z)| < 1$ である。前進オイラー法ではこれは $|1 + z| < 1$ であり、実数の $\lambda < 0$ に対しては $-2 < h\lambda < 0$、すなわち次式となる。

$$h < \frac{2}{|\lambda|}.$$

冒頭のボウルでは $2/|\lambda| = 2/50 = 0.04$ となり、これを超えると反復点が増大したあの学習率と一致する。

この上限より下では、オイラー法の誤差上界に欠けていた収縮する場の解析も同時に手に入る。1 ステップ進めると、すでに生じた誤差には $|1 + h\lambda| < 1$ が掛かるようになるが、一般の証明ではこの因子を $1 + hL$ でしか上から抑えられなかった。$h \le 1/|\lambda|$ のとき、あの証明の等比級数の和は $1/(h|\lambda|)$ 以下に収まり、誤差上界は指数因子が消えた $Mh/(2|\lambda|)$ となる。ボウルの減衰方程式を $x_0 = 1$ から解く場合、$M = 2500$ であるからこの上界は $25\,h$ となり、真の誤差の 3 倍以内に収まる。

38.2.3.1 後退オイラー法

ステップ幅を短くするのは 1 つの解決策である。もう 1 つの方法は一見奇妙に見えるが、ステップの**終点**における傾きを使ってオイラー法のステップを進めるというものである。

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_{n+1}, t_{n+1}).$$

これが**後退オイラー法**（backward Euler）である。未知数 $\mathbf{x}_{n+1}$ が両辺に現れるという厄介な点があるが、その対処はこの手法が何をもたらすかを確認するまで後回しにしよう。

命題（後退オイラー法は任意のステップ幅で安定である）。 *$\operatorname{Re}\lambda < 0$ であるテスト方程式において、後退オイラー法の安定性関数は $R(z) = 1/(1 - z)$ であり、$|R(z)| < 1$ がすべてのステップ幅 $h > 0$ に対して成り立つ。*

証明。 1 ステップの更新は $y_{n+1} = y_n + h\lambda y_{n+1}$ であるから $y_{n+1} = y_n/(1 - z)$ となり、$|1 - z| \ge \operatorname{Re}(1 - z) = 1 - \operatorname{Re} z > 1$ が $\operatorname{Re} z < 0$ のときに成り立つ。$\blacksquare$

その理由は代数計算なしでも理解できる。前進オイラー法は現在地の傾きで外挿するため、急峻なモードの上でその傾きを長いステップにわたって保つと、ゼロを大きく通り越してしまう。一方、後退オイラー法は、その点自身の傾きを逆向きにたどると現在地に戻るような点を探す。ほぼ減衰し切った点では傾きもほぼゼロであるため、どれほど長いステップをとっても行き過ぎることはない。

ここで、先ほど後回しにした厄介な点に向き合おう。未知の点で場を評価するため、各ステップで方程式を解く必要が生じる。場が線形であれば連立一次方程式（linear system）を 1 回解き、そうでなければニュートン法の反復を数回行うことになる（演習問題 3）。このような手法は**陰的**（implicit）と呼ばれ、公式によって新しい状態を直接計算する前進オイラー法や RK4 は**陽的**（explicit）と呼ばれる。陽的手法のステップ幅が精度ではなく安定性によって制限されてしまう場面では、陰的ステップは方程式を解く手間に見合う価値を持つ。

それでは 2 モード系に対し、5 通りのステップサイズで両手法を実行してみよう。後退オイラー法ではステップごとに連立一次方程式を 1 回解く。最初の行の結果は計算するまでもなく予測できるはずである。$h = 0.1$ では、前進オイラー法は速い座標に $1 - 5 = -4$ を 10 回掛けることになる。

```
def backward_euler_linear(A, x0, T, n):
    """Backward Euler on dx/dt = A x: solve (I - h A) x_new = x at each step."""
    x, h = np.asarray(x0, dtype=float), T / n
    for _ in range(n):
        x = np.linalg.solve(np.eye(len(x)) - h * A, x)
    return x

A_stiff = np.diag([-50.0, -1.0])              # one fast mode, one slow mode
x0_stiff = np.array([1.0, 1.0])
T = 1.0
print(f'exact at T = 1: fast {np.exp(-50):.1e}, slow {np.exp(-1):.4f}')
print(f'{"h":>5} {"fwd fast":>10} {"fwd slow":>9} {"bwd fast":>10} {"bwd slow":>9}')
for n in [10, 20, 25, 50, 100]:
    fwd = euler(lambda x: A_stiff @ x, x0_stiff, T, n)
    bwd = backward_euler_linear(A_stiff, x0_stiff, T, n)
    print(f'{T / n:5.2f} {fwd[0]:10.1e} {fwd[1]:9.4f} {bwd[0]:10.1e} {bwd[1]:9.4f}')
```

```
exact at T = 1: fast 1.9e-22, slow 0.3679
    h   fwd fast  fwd slow   bwd fast  bwd slow
 0.10    1.0e+06    0.3487    1.7e-08    0.3855
 0.05    3.3e+03    0.3585    1.3e-11    0.3769
 0.04   -1.0e+00    0.3604    1.2e-12    0.3751
 0.02    0.0e+00    0.3642    8.9e-16    0.3715
 0.01    7.9e-31    0.3660    2.5e-18    0.3697
```

$h = 0.1$ では、どちらの手法も遅いモードを真の値から約 5% 以内（0.349 と 0.386 であり、真値は 0.368）で捉えており、多くの用途には十分な精度である。それにもかかわらず、前進オイラー法は速い成分に対して $4^{10} \approx 10^6$ を返してしまう（真の値は 10^{-22} である）。$h = 0.05$ ではその因子は冒頭で見た -1.5 であり、これが 20 回掛けられる。$h = 0.04$ では因子は -1 となり、速い成分は毎ステップ符号を反転させるばかりで決して縮小しない（$h = 0.02$ の行にあるゼロは厳密値であり、このとき $1 - 50h = 0$ となって 1 ステップで速いモードが完全に消去される）。前進オイラー法には $h < 0.04$ が必要だが、そのステップ幅では遅いモードの誤差はすでに約 2% 以内に達している。5% の精度で十分だったのに、安定性のせいで 2% の精度まで計算させられたのである。

後退オイラー法はあらゆるステップサイズで安定である。ただし遅いモードの誤差は 1 次手法としての $O(h)$ の誤差のままであり、安定性が精度まで高めてくれるわけではない。

38.2.3.2 安定領域

前進オイラー法にはステップサイズの上限があり、後退オイラー法にはない。では他の手法はどこに位置するのだろうか。複素 z 平面をプロットすれば、それらを一目で比較できる。ある手法の**安定領域**（stability region）とは、更新が減衰する集合 $\{z : |R(z)| < 1\}$ のことである。安定領域が開左半平面 $\mathrm{Re}\, z < 0$ 全体を含むとき、その手法は **A 安定**（A-stable）であるという。すなわち、ステップ幅によらずすべての減衰モードを減衰させる手法である (Dahlquist 1963)。後退オイラー法の安定領域は $+1$ を中心とする閉単位円板の外側全体であるため、A 安定である。

Figure 38.6 にこれまでの例をプロットした。冒頭のステップ $h = 0.05$（$\dot{x} = -50x$）はオイラー法の円板の外にある点 $z = -2.5$ に対応し、$h = 0.01$ は円板内部の $z = -0.5$ に対応する。固有値 $\lambda = -0.5 \pm \mathrm{i}$ を持つスパイラルでは $|1 + z|^2 = (1 - h/2)^2 + h^2$ となり、これが 1 未満となるのは $h < 0.8$ のとき、かつそのときに限られる。$h = 0.8$ では 2 点 $-0.4 \pm 0.8\,\mathrm{i}$ がちょうど円周上にのる。純粋な回転は z を虚軸上に置くが、虚軸はオイラー法の円板と原点でしか接しないため、前進オイラー法は減衰のない振動をあらゆるステップサイズで増大させてしまう（演習問題 2 および 4）。一方、厳密なステップ e^{z}（38.1.2.3 節）は左半平面全体で絶対値が 1 未満となり、後退オイラー法はこの振る舞いを模倣しているのに対し、前進オイラー法はそうではない。

いかなる陽的ルンゲ・クッタ法も、段数をどれだけ増やそうと A 安定にはならない。その安定性関数 R は多項式であり（演習問題 2）、定数でない多項式は z が負の実軸に沿って遠ざかると際限なく増大するからである。したがって、次数を高くしても硬さは解消されない。RK4 の安定領域は実軸上では $z = -2.79$（$R_{\mathrm{RK4}}(z) = 1$ の負の根）で途切れる。ボウルの減衰方程式に対して RK4 が許容するステップ幅は $h < 2.79/50 = 0.056$ であり、オイラー法の 0.04 と大差ないうえ、1 ステップあたり場を 4 回も評価する。硬い場に対しては陰的手法が温存され、それ以外のあらゆる場面では陽的手法が標準として使われる理由はここにある。本書の読者が最も頻繁に出会う硬い場とは、ある方向では曲率（curvature）が大きく、別の方向では曲率が小さい損失の勾配である。

38.2.4 ソルバとしての勾配降下法

冒頭では 1 つの座標だけを扱ったが、座標が複数ある場合にはヘッセ行列（Hessian）の固有値ごとに同じ因子が現れる。損失 $\mathcal{L}$ 上をつねに負の勾配に沿って連続的に下っていく運動は、次の**勾配フロー**（gradient flow）である。

$$\dot{\mathbf{x}} = -\nabla \mathcal{L}(\mathbf{x}),$$

これは $\mathcal{L}$ の臨界点（critical point）でゼロになり、その軌跡に沿って損失が単調に減少する（$\frac{d}{dt}\mathcal{L}(\mathbf{x}(t)) = \nabla\mathcal{L}^{\top}\dot{\mathbf{x}} = -\|\nabla\mathcal{L}\|^2 \le 0$）ようなベクトル場である。学習率 η の勾配降下法 $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\nabla\mathcal{L}(\mathbf{x}_k)$ は、このフローに対する前進オイラー法 Equation 38.9 において $h = \eta$ とおいたものにほかならない。

35.1 節では、学習率の上限 $\eta < 2/\lambda_{\max}(\mathbf{H})$ をヘッセ行列 $\mathbf{H}$ を持つ二次関数上で導出し、その上限より下での収束速度を調べている。収束速度の議論は繰り返さないが、上限については 3 本の矢印で改めて導出しよう。そこに至る道筋こそがソルバとしての視点のもたらす新しい知見だからである。$\delta = \mathbf{x} - \mathbf{x}^{\star}$ を最小点 $\mathbf{x}^{\star}$（ヘッセ行列 $\mathbf{H}$）からの変位とする。最小点の近傍ではフローが線形化され、各固有値 λ_i（$\mathbf{H}$ の固有値）が減衰率 $-\lambda_i$ を持つ減衰モード、すなわちテスト方程式のコピーになる。

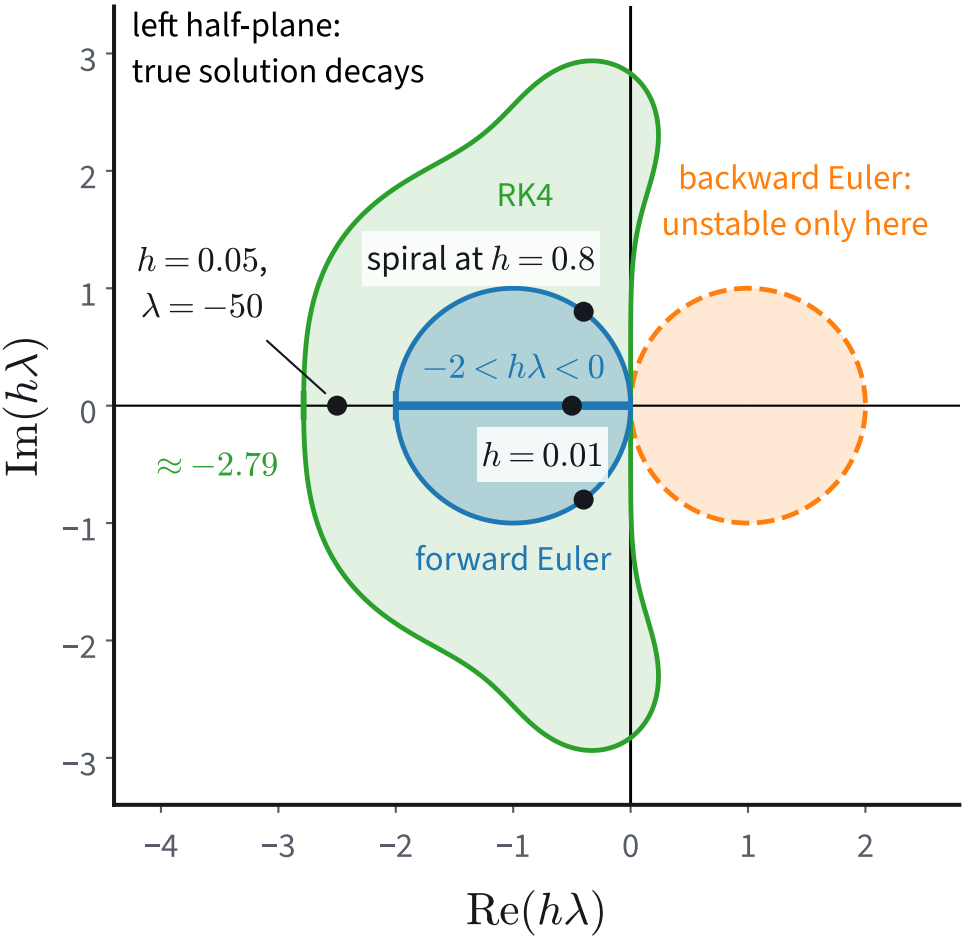


Figure 38.6: 複素平面 $z = h\lambda$ におけるテスト方程式 $\dot{y} = \lambda y$ の安定領域。減衰モードは左半平面に位置する。前進オイラー法が減衰するのは円板 $|1 + z| < 1$（実軸上のスライスは $-2 < h\lambda < 0$）の内部に限られる。RK4 はこれを実軸上で ≈ -2.79 に達する葉状の領域へと広げる。後退オイラー法は $+1$ の周りの円板を除く全域、特に左半平面全体で安定であり、これが A 安定性である。印を付けた点は、冒頭の例の $h = 0.05$ ステップと $h = 0.01$ ステップ、およびオイラー法の円板の境界上にある $h = 0.8$ のスパイラルである。

$$\dot{\mathbf{x}} = -\nabla \mathcal{L}(\mathbf{x}) \xrightarrow{\text{linearize}} \dot{\delta} = -\mathbf{H}\delta \xrightarrow{\text{Euler, } h=\eta} \delta_{k+1} = (\mathbf{I} - \eta \mathbf{H})\,\delta_k \xrightarrow{\text{stability}} \eta < \frac{2}{\lambda_{\max}(\mathbf{H})}.$$

このように、学習率の上限とはオイラー法の安定円板の境界であり、最も急峻なモードが $z = -\eta\lambda_{\max} = -2$ でそこに達する点にあたる。最初の矢印ではヘッセ行列を固定しているため、この上限は二次関数に関する主張である。実際のネットワーク上では訓練中にヘッセ行列が変化するため、この上限を超えて実行しても爆発しないことが少なくない。最大固有値が $2/\eta$ に向かって押し戻され、毎ステップではないものの長期的には損失が減少し続けるのである（35.1.2.4 節）。

この解釈が正しければ、上限はオイラー法に由来しており、別のソルバを使えば別の上限が現れるはずである。35.1 節で学習率をスイープさせた谷型の関数 $\mathcal{L}(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\top \mathbf{H}\mathbf{x}$（ただし $\mathbf{H} = \mathrm{diag}(10, 1)$ であり、同節とは平坦な座標と急峻な座標の順が逆である）でこの予測を検証してみよう。その勾配フロー $\dot{\mathbf{x}} = -\mathbf{H}\mathbf{x}$ は、減衰率 -10 の急峻なモードと減衰率 -1 の平坦なモードを持つ。初期点 $(1, 1)$ から 3 種類のソルバで 100 ステップ（ステップ幅 η）進める。前進オイラー法は勾配降下法そのものであり、上限は $2/10 = 0.2$ にある。RK4 は 1 ステップあたり勾配を 4 回評価し、その安定領域は実軸上で -2.79 まで届くため（Figure 38.6）、上限は 0.279 になるはずである。後退オイラー法 $\mathbf{x}_{k+1} = (\mathbf{I} + \eta\mathbf{H})^{-1}\mathbf{x}_k$ には上限がないはずである。次のコードセルでは、各実行について 1 ステップが 2 つのモードに掛ける因子 $R(-10\eta)$ と $R(-\eta)$、および 100 ステップ後の反復点の長さを出力する。これらの因子は単位行列（identity matrix）に 1 ステップを適用して実測している。

```
H = np.diag([10.0, 1.0])                        # Hessian of L(x) = x^T H x / 2
flow = lambda x: -H @ x                         # the gradient flow
solvers = {                                     # n steps of size eta on the flow
    'forward Euler': lambda x0, eta, n: euler(flow, x0, n * eta, n),
    'RK4': lambda x0, eta, n: rk4(flow, x0, n * eta, n),
    'backward Euler': lambda x0, eta, n: backward_euler_linear(-H, x0, n * eta, n)}
print(f'{"solver":<15}{"eta":>6}{"steep factor":>14}{"flat factor":>13}{"||x_100||":>12}')
for name, eta in [('forward Euler', 0.19), ('forward Euler', 0.21), ('RK4', 0.27),
                  ('RK4', 0.29), ('backward Euler', 0.25), ('backward Euler', 25)]:
    steep, flat = np.diag(solvers[name](np.eye(2), eta, 1))  # one step's matrix
    x_100 = solvers[name](np.array([1.0, 1.0]), eta, 100)
    print(f'{name:<15}{eta:6.2f}{steep:14.3f}{flat:13.3f}{np.linalg.norm(x_100):12.1e}')
```

```
solver            eta  steep factor  flat factor   ||x_100||
forward Euler    0.19        -0.900        0.810     2.7e-05
forward Euler    0.21        -1.100        0.790     1.4e+04
RK4              0.27         0.879        0.763     2.5e-06
RK4              0.29         1.187        0.748     2.8e+07
backward Euler   0.25         0.286        0.800     2.0e-10
backward Euler  25.00         0.004        0.038    3.2e-142
```

各ソルバは、それぞれの安定領域が途切れる場所に固有の上限を持っている。前進オイラー法は $\eta = 0.19$ では急峻なモードの因子が -0.9 となって収束し、$\eta = 0.21$ では -1.1 となって発散する。RK4 は $\eta = 0.27$ でもまだ収束し、0.29 で発散する。後退オイラー法は勾配降下法の上限を超えた $\eta = 0.25$ でも収束し、両者をはるかに上回る $\eta = 25$ でも 2 つの因子が 0.004 と 0.038 となって、100 ステップ後には反復点が測定できないほどゼロに近づく。冒頭で「損失に問題がないとすれば何が問題なのか」と問うたが、答えは「上限は商で決まる」ということである。

損失がその分母 $\lambda_{\max}$ を与え、ソルバがその分子を与える。分子は前進オイラー法では 2、RK4 では 2.79 であり、後退オイラー法にはそもそも上限が存在しない。

ではネットワークの訓練に RK4 を使うべきだろうか。答えは否である。上限は 1.4 倍高くなるものの 1 ステップに 4 回の勾配計算を要するため、勾配 1 回あたりに進められるフローの時間は高々 0.07 にとどまり、勾配降下法の 0.2 に及ばない。追加の勾配評価は勾配フローの経路をより忠実にたどるために費やされるが、オプティマイザにとって途中の経路自体には用がなく、欲しいのは経路が行き着く終着点だけである。しかも、経路から外れることは単なる誤差にとどまらない。η の 2 次の範囲で、勾配降下法の反復点は大きな勾配にペナルティを課す修正損失のフローをたどるため、オイラー法がソルバとして犯す誤差は正則化項として働くのである（演習問題 5）。

2 つの手法は破綻の仕方も異なる。オイラー法の急峻なモードの因子は -1 を通って安定範囲を抜け出すため、反復点は谷底を飛び越えながら毎回より高い位置へと着地する。一方、RK4 の因子は実軸全体で正であり、$+1$ を通って安定範囲を抜ける。$\eta = 0.29$ では因子が 1.187 となり、反復点は一度も符号を変えないまま最小点から遠ざかっていく。つまりジグザグの振動もまた、オイラー法に固有の現象なのである。その定番の処方箋であるモメンタム（momentum）は、そもそも別の方程式を解いている。Polyak のヘビーボール法は、質量と摩擦を持つ球の運動方程式 $m\ddot{\mathbf{x}} + c\dot{\mathbf{x}} = -\nabla \mathcal{L}(\mathbf{x})$ を離散化したものである (Polyak 1964)（演習問題 6）。

後退オイラー法はステップ幅を大きくするほど因子 $1/(1+\eta\lambda_i)$ が小さくなるため、性能は向上する一方である。難点は方程式の求解にある。その更新式 $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla \mathcal{L}(\mathbf{x}_{k+1})$ は、関数 $\mathcal{L}(\mathbf{x}) + \frac{1}{2\eta}\|\mathbf{x} - \mathbf{x}_k\|^2$ の勾配が $\mathbf{x}_{k+1}$ でゼロになることを意味しており、凸な損失に対しては損失そのものに適用した近接ステップ（proximal step）$\operatorname{prox}_{\eta\mathcal{L}}(\mathbf{x}_k)$（35.6.4.1 節）に一致する。二次関数上ではこれは 1 回の連立一次方程式の求解で済むが、ネットワーク上ではステップごとにそれ自体が 1 つの最適化問題となり、η が大きいときには元々解こうとしていた問題そのものになってしまう。

この谷型の関数は 38.2.3 節の硬い系を穏やかにしたものであり、減衰率として 10 と 1（50 と 1 の代わり）、ステップ幅として η（h の代わり）を用いている。任意の $\eta < 0.2$ に対して勾配降下法の平坦方向の因子 $1-\eta$ は 0.8 より大きいままであるため（出力表には 0.81（$\eta = 0.19$）と表示されている）、急峻な方向がどれほど速く収束しようと、平坦な方向は 1 ステップあたり長さの 5 分の 1 未満しか縮まない。一般には縮小する割合は $2\lambda_{\min}/\lambda_{\max}$ 未満であり、条件数（condition number）が 1000 であれば 500 分の 1 にすぎない。安定性が η の上限を $\lambda_{\max}$ によって決め、そのもとで $\lambda_{\min}$ が収束速度を決める。つまり、ヘッセ行列の条件数とは勾配フローの硬さにほかならず、二次関数における学習率の発散とはソルバの不安定性そのものなのである。

本節で扱ったすべての場では、自分たちで場を書き下していたため、最初の 1 ステップを踏み出す前から固有値がわかっていた。しかしネットワークで表される場には、そのような固有値のリストは付いてこない。38.3 節では、層をオイラー法のステップに見立てて学習された場そのものをモデルとし、そのようなモデルをどう訓練するか、そしてそれが確率密度（probability density）をどう変換するかを問う。

38.2.5 まとめ

数値ソルバは軌跡を離散的なステップで置き換えるものであり、その性能は 2 つの性質によって評価される。第 1 に、次数 p はステップ幅を小さくしたときに誤差がどれほど速く減少するかを表す。局所誤差 $O(h^{p+1})$ は累積して大域誤差 $O(h^p)$ となり、オイラー法では $p = 1$、RK4 では $p = 4$ である。第 2 に、安定領域はどのステップ幅なら爆発しないかを表す。前進オイラー法には $|1 + h\lambda| < 1$ が必要であるのに対し、後退オイラー法は毎ステップ方程式を解くことで左半平面全体で安定となり、とうに減衰し切った速いモードがなおステップ幅を制限するとき系は硬いという。線形な場の上では、これら両方の性質が、1 ステップでモードに掛けられる因子である安定性関数 $R(h\lambda)$ から読み取れる。勾配降下法は $\dot{\mathbf{x}} = -\nabla\mathcal{L}$ に対する前進オイラー法である

ため、その学習率の上限 $\eta < 2/\lambda_{\max}(\mathbf{H})$ はオイラー法の安定領域の境界に一致し、悪条件性は硬さにほかならない。

38.2.6 演習問題

1. [code] **適応的ステップサイズ。**●埋め込み型ペアは、次数の異なる 2 つの手法から局所誤差を推定する。最も単純なペアは、ホイン法の段 $\mathbf{k}_1 = \mathbf{f}(\mathbf{x})$ と $\mathbf{k}_2 = \mathbf{f}(\mathbf{x} + h\mathbf{k}_1)$ を用いるものである。オイラー法の値 $\mathbf{x} + h\mathbf{k}_1$ とホイン法の値 $\mathbf{x} + \tfrac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2)$ の差の大きさは $\Delta = \tfrac{h}{2}\|\mathbf{k}_2 - \mathbf{k}_1\|$ となる。制御器は $\Delta \le \texttt{tol}$ のときにホイン法の値を採用してそのステップを受理し、受理・棄却のいずれの場合も次のステップサイズで計算を続ける。

$$h \leftarrow h \min\!\Big(2,\ \max\!\big(0.2,\ 0.9\sqrt{\texttt{tol}/\Delta}\,\big)\Big).$$

   1. 指数 $1/2$ は、局所誤差が何次であることを仮定しているか。
   2. $h = 0.1$ から開始し、最後のステップがちょうど T で終わるように短縮する形でこの制御器を実装せよ。スパイラル場 `f` を `x0` から $T = 2$ まで、$\texttt{tol} = 10^{-2}, 10^{-3}, \dots, 10^{-6}$ に対して積分せよ。棄却されたステップの分も含めて場の評価回数を数え、厳密な終端点 `x_true` との誤差を比較せよ。評価回数は `tol` に対してどのようにスケーリングするか。
   3. `A_stiff` の場を $(1, 1)$ から $T = 10$ まで $\texttt{tol} = 10^{-3}$ で積分し、受理されたステップサイズを時間に対してプロットせよ。速いモードが減衰した後、ステップサイズはどの値に落ち着くか。この値をホイン法の安定性関数 $1 + z + z^2/2$（Equation 38.12）を用いて説明せよ。

2. **回転モード。**●本文の硬い系の例では固有値は実数であった。複素固有値 λ は $z = h\lambda$（Figure 38.6 の座標）を通じて現れ、その虚部がステップの安定性を左右しうる。

   1. Figure 38.1 のスパイラル場（固有値は $-0.5 \pm \mathrm{i}$）について、前進オイラー法は $h < 0.8$ で減衰することを本文で求めた。Equation 38.12 の安定性関数から、ホイン法が減衰する最大のステップ幅（条件は h の 3 次式になるので数値的に解け）および RK4 に対する最大のステップ幅を求めよ。ホイン法はステップあたり場を 2 回、RK4 は 4 回評価する。3 つの手法のうち、解を増大させずに場評価 1 回あたり最も長い時間を進められるのはどれか。
   2. 回転場 $\dot{\mathbf{x}} = \left(\begin{smallmatrix} 0 & -b \\ b & 0 \end{smallmatrix}\right)\mathbf{x}$ に対し、前進オイラー法は 1 ステップごとに $\|\mathbf{x}\|$ を $\sqrt{1 + h^2 b^2}$ 倍し、後退オイラー法は同じ因子で割ることを示せ。各手法において、$\|\mathbf{x}\|$ は 1 回転すなわち $2\pi/(hb)$ ステップ（$hb = 0.1$ とする）の間に何倍に変化するか。
   3. RK4 の安定性関数は Equation 38.12 の最後の式であり、虚軸上でのその絶対値の二乗が回転に対する振る舞いを決める。$|R(\mathrm{i}y)|^2 = 1 - y^6/72 + y^8/576$ が実数 y に対して成り立つことを示し、RK4 が回転を増幅させない hb の範囲を求めよ。
   4. s 段の陽的ルンゲ・クッタ法は、$\mathbf{k}_i = \mathbf{f}(\mathbf{x}_n + h\sum_{j<i} a_{ij}\mathbf{k}_j)$（$i = 1, \dots, s$）および $\mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_i b_i \mathbf{k}_i$ を計算する（Equation 38.11 の RK4 と同様である）。その安定性関数が z に関する高々 s 次の多項式であることを示せ。なぜこのことから、陽的手法では A 安定性が除外されるのか。

3. [code] **硬い非線形場における陰的ステップ。**●本文の硬い系は線形かつ対角であったため、後退オイラー法の各ステップは 1 回の連立一次方程式の求解で済んだ。代わりに、減衰振り子 $\dot{\phi} = \omega$、$\dot{\omega} = -\sin\phi - c\,\omega$（38.1.2.5 節）に強い減衰 $c = 50$ を入れ、$(\phi, \omega) = (1, 0)$ から $T = 2$ まで解くことを考えよ。

   1. ヤコビ行列 $\mathbf{J}(\phi, \omega) = \left(\begin{smallmatrix} 0 & 1 \\ -\cos\phi & -c \end{smallmatrix}\right)$ の最下点 $(0, 0)$ における固有値を求め、そこから導かれる前進オイラー法と RK4 のステップサイズの上限を計算せよ。

2. 速い過渡応答が収まった後の運動はほぼ $\dot{\phi} = -\sin\phi/c$ を満たすことを示し、この 1 次元方程式から $\phi(2)$ を求めよ。
3. 後退オイラー法の 1 ステップは、$\mathbf{g}(\mathbf{y}) = \mathbf{y} - \mathbf{x}_n - h\mathbf{f}(\mathbf{y}) = \mathbf{0}$ を $\mathbf{y} = \mathbf{x}_{n+1}$ について解く。これに対するニュートン反復 $\mathbf{y} \leftarrow \mathbf{y} - (\mathbf{I} - h\mathbf{J}(\mathbf{y}))^{-1}\mathbf{g}(\mathbf{y})$ を書き下し、ステップごとに 3 回のニュートン反復（初期値は $\mathbf{x}_n$）を行う後退オイラー法を実装せよ。
4. 前進オイラー法、RK4、後退オイラー法を $n = 20$、33、36、100 の等間隔ステップ（$h = 0.1$、0.061、0.056、0.02）で実行し、`rk4` を 2^{16} ステップ用いて計算した参照解と比較せよ。どの実行が爆発するか。また、小問 (a) の上限はその結果を予測しているか。続いて各手法について終端誤差が 10^{-4} 未満となる最小の n を求め、そのときに要する場とヤコビ行列の評価回数を数えよ。どの手法では精度が n を決め、どの手法では安定性が決めているか。

4. [code] **シンプレクティック・オイラー法と作用素分割**。●調和振動子 $\dot{q} = p$、$\dot{p} = -q$ は $q^2 + p^2$ を保存する。その円軌道に対して前進オイラー法は外側へ渦を巻いて離れ、後退オイラー法は内側へ渦を巻いて縮む。これに対し、**シンプレクティック・オイラー法**の更新は q を古い p で進めてから、p を新しい q で進める。すなわち、$q_{n+1} = q_n + hp_n$、$p_{n+1} = p_n - hq_{n+1}$ とする。

1. この更新を列ベクトルとして並べた状態に作用する行列 $\mathbf{S}_h$（(q,p) 上の変換）として書き下し、$\det \mathbf{S}_h = 1$ がすべての h に対して成り立つこと、すなわち厳密なフローと同様に面積を保存することを示せ。また、前進オイラー法の行列式が $1 + h^2$ となることも確認せよ。
2. $\mathbf{S}_h$ の固有値が互いに異なりかつ単位円周上にのる必要十分条件が $0 < h < 2$ であることを示せ。$h = 2$ のとき、および $h > 2$ のときには何が起こるか。
3. $q^2 + p^2 + hqp$ がこの更新によって厳密に保存されることを示せ。$(1,0)$ を通るその等位集合はどのような形をしており、計算された軌道は $h = 0.5$ と $h = 1$ において原点から最大どれだけ離れうるか。（ヒント：この二次形式の主軸の長さの比は $\sqrt{(2+h)/(2-h)}$ である。）
4. $(1,0)$ から $T = 200$ まで、前進オイラー法、後退オイラー法、シンプレクティック・オイラー法を用いて $h = 0.1$ と $h = 1$ で積分し、各実行に沿った $\|(q,p)\|$ の最大値と最小値を報告せよ。
5. この更新は、$\dot{q} = p$（p を固定）の時間 h にわたる厳密なフローに続けて、$\dot{p} = -q$（q を固定）の厳密なフローを適用したものである。和がこの振動子の行列となる 2 つの生成作用素 $\mathbf{P}$ と $\mathbf{Q}$ を書き下し、$e^{h\mathbf{P}}$、$e^{h\mathbf{Q}}$、$e^{h(\mathbf{P}+\mathbf{Q})}$ を計算して、$e^{h\mathbf{Q}}e^{h\mathbf{P}} = \mathbf{S}_h$ が $e^{h(\mathbf{P}+\mathbf{Q})}$ と h^2 のオーダーの項だけ異なることを示せ。38.1 節におけるスパイラルの閉形式解では、指数関数を誤差なしに分解できていた。そこで用いたどの性質がここでは成り立たないのか。対称分解 $e^{h\mathbf{Q}/2}e^{h\mathbf{P}}e^{h\mathbf{Q}/2}$ はハミルトニアンモンテカルロ (Neal 2011) の**リープフロッグ**積分器である。その次数はいくらか。

5. [code] **暗黙的勾配正則化**。●●勾配降下法は勾配フローに対する前進オイラー法であるため（38.2.4 節）、各ステップは $\mathcal{L}$ のフローから $O(\eta^2)$ だけ離れる。後退誤差解析では、修正ベクトル場 $\tilde{\mathbf{f}} = \mathbf{f} + \eta\mathbf{f}_1$ であって、その厳密な時間 η のフローがオイラー法の 1 ステップと $O(\eta^3)$ の範囲で一致するものを探す。Barrett and Dherin (2021) は、勾配降下法に対してこの修正場が再び勾配場となり、次の修正損失から導かれることを示した。

$$\tilde{\mathcal{L}}(\mathbf{x}) = \mathcal{L}(\mathbf{x}) + \frac{\eta}{4}\|\nabla\mathcal{L}(\mathbf{x})\|^2.$$

1. $\tilde{\mathbf{f}}$ の時間 η にわたる厳密なフローを 2 次まで展開し、$\mathbf{f}_1 = -\frac{1}{2}(\partial\mathbf{f}/\partial\mathbf{x})\,\mathbf{f}$ を示せ。さらに $\tilde{\mathcal{L}}$ を $\mathbf{f} = -\nabla\mathcal{L}$ の場合に導出せよ。追加された項はパラメータ空間のどのような領域にペナルティを課すか。

2. $\mathcal{L}(x) = \frac{\lambda}{2}x^2$（ただし $0 < \eta\lambda < 1$）に対し、勾配降下法の反復点が $\frac{\tilde{\lambda}}{2}x^2$（ただし $\tilde{\lambda} = -\log(1-\eta\lambda)/\eta$）の勾配フロー上に厳密にのることを示せ。$\tilde{\lambda}$ を η で展開し、$\tilde{\mathcal{L}}$ の曲率と比較せよ。
3. この二次関数に対する勾配降下法は $1 < \eta\lambda < 2$ でも依然として収束するが、反復点の符号は交互に反転する。リプシッツ定数を持つスカラーの自律的な場の軌跡は交差できないため、そのような場の時間 η 写像は大小順序を保存する。このことから、いかなるそのような場も時刻 $k\eta$ において反復点の値をとることはできないと結論せよ。さらに、ヘッセ行列 $\mathbf{H}$ を持つ d 次元の二次関数に対し、最大固有ベクトル方向の成分を持つ任意の初期点について、反復点は、いかなる勾配場 $-\nabla\big(\frac{1}{2}\mathbf{x}^\top\tilde{\mathbf{H}}\mathbf{x}\big)$ のフロー上にも $\eta \geq 1/\lambda_{\max}(\mathbf{H})$ となった時点でのらないことを示せ。
4. $\mathcal{L}(x) = x^4/4 + x^2/2$、$x_0 = 1.5$ に対し、勾配降下法を $T = 1$ まで $\eta = 0.1, 0.05, 0.025, 0.0125$ で実行せよ。$\mathcal{L}$ および $\tilde{\mathcal{L}}$ の勾配フローを $T = 1$ まで `rk4` を十分なステップ数で用いて計算せよ（後者は各 η ごとに個別に計算すること）。勾配降下法の 2 つの終端誤差の傾きを η に対して直線当てはめせよ。

6. **離散化された ODE としてのヘビーボール法。**●ヘビーボールモメンタム $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\nabla\mathcal{L}(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1})$ は、方程式 $m\ddot{\mathbf{x}} + c\dot{\mathbf{x}} = -\nabla\mathcal{L}(\mathbf{x})$（質量 m と摩擦係数 c を持つ球の運動）を離散化したものである。

   1. $\ddot{\mathbf{x}}(t_k)$ を中心差分 $(\mathbf{x}_{k+1} - 2\mathbf{x}_k + \mathbf{x}_{k-1})/h^2$ で、$\dot{\mathbf{x}}(t_k)$ を後退差分 $(\mathbf{x}_k - \mathbf{x}_{k-1})/h$ で置き換えよ。その結果がヘビーボール法の更新式になることを示し、η と β を m、c、h を用いて表せ。
   2. $\mathcal{L}(x) = \frac{\lambda}{2}x^2$ に対してこの ODE は減衰振動子となり、$c = 2\sqrt{m\lambda}$ で臨界減衰となる。この条件を η、β、λ を用いて表せ。ヘビーボール法の漸化式は、その特性多項式 $z^2 - (1 + \beta - \eta\lambda)z + \beta$ が重根を持つときに臨界減衰となる。対応する β を求め、小さい $\eta\lambda$ に対して 2 つの条件を比較せよ。
   3. この二次関数上のヘビーボール法の漸化式が収束する必要十分条件は、$0 < \eta\lambda < 2(1+\beta)$（ただし $0 \leq \beta < 1$）である（演習問題 8 の小問 (a)（35.1 節）で証明を求めている）。この条件を h、m、c、λ に関する条件に書き換えよ。元の ODE はすべての $\lambda > 0$ に対して減衰するが、その離散化はそうではない。安定となる最大のステップ幅 h は、$c \to 0$ の極限において振動子の角振動数 $\sqrt{\lambda/m}$ を用いてどう表されるか。悪条件な二次関数のヘッセ行列のどの固有値が、ステップ幅の上限を決めるか。

38.3 ニューラル ODE、随伴法、連続フロー

残差ブロック（residual block）は入力に補正項を加えるが、オイラー法（38.2 節）の 1 ステップも同じ形をしている。残差ネットワーク（residual network、7.4 節）が微分方程式の数値解だとすれば、方程式そのものをモデルとすることで何が得られ、どのようなコストがかかるのだろうか。

生成モデリングがその 1 つの答えを与える。最尤法（maximum likelihood）には、モデルが生成する出力の密度が必要である。正規化フロー（normalizing flow）は逆変換可能なネットワークであり、この密度は変数変換（change of variables）の公式から従う。しかし、その公式はサンプルごとにネットワークのヤコビ行列（Jacobian）の行列式（determinant）を必要とし、そのサイズは 3072 行にもなる（32×32 ピクセルのカラー画像の場合）。連続時間（continuous-time）に移ると、この行列式はトレース（trace）へと変わり、トレースは 1 回の逆伝播（backward pass）で推定できる。

ニューラル ODE（neural ODE）は方程式そのものをモデルとする (Chen et al. 2018)。すなわち、ベクトル場（vector field）をネットワークで表し、終端時刻における解を出力とする常微分

方程式（ordinary differential equation、ODE）$\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}, t)$ である。その訓練と密度の計算は、いずれも状態（state）と並んでもう 1 本の方程式を積分することに帰着する。訓練で用いるのは**随伴**方程式（adjoint equation）であり、これは軌跡（trajectory）のコストが制御入力に応じてどう変化するかを調べるため、1950 年代に最適制御理論で導入されたものである (Pontryagin et al. 1962)。密度の計算で用いるのは、ヤコビ行列のトレースの積分である。

残差ブロックの積み重ねと比べると、このモデルは可逆な写像、モデル自体から切り離されたステップ数、活性化値（activation）を保存せずに済む勾配（gradient）、そして計算可能な密度を手に入れる。ただし、それぞれの利点には落とし穴があり、本節ではその 4 つすべてを定量的に確かめる。すなわち、直線上のフローは 2 点を入れ替えられず、訓練されたベクトル場は使用したソルバに縛られ、状態を保存せずに計算した勾配にはドリフト（drift）が生じ、安価に求まるトレースにはノイズが乗る。

本節では、このようなモデルをソルバを通して訓練し、随伴法（adjoint method）を導出し、フローに沿った密度の変化を追う。持ち帰るべき結果は随伴方程式とトレース則 Equation 38.21 であり、拡散モデル（diffusion model）の尤度（likelihood）は後者に基づいている。トレース則だけが必要な読者は 38.3.3 節から読み始めてもよい。メモリとドリフトに関する項の内容は、これ以降の節では前提としない。

以下のセルでは、38.2 節のオイラーステップと RK4 ステップを通して訓練する。ループ `solve` はこれらのステップを N 回（ステップサイズ T/N）実行し、負の T を与えるとベクトル場を逆向きに走らせる。さらに後段で行う随伴法の検証では、スパイラルを表す行列 $\mathbf{A}$ とその初期値 $\mathbf{x}_0$、および行列指数関数（matrix exponential）$e^{\mathbf{C}s} = \mathbf{V} e^{\Lambda s} \mathbf{V}^{-1}$ を用いる（38.1 節）。

```
rng = np.random.default_rng()
A = np.array([[-0.5, -1.0], [1.0, -0.5]])      # spiral sink: eigenvalues -0.5 +- i
x0 = np.array([2.0, 0.0])

def expm_eig(C, s):
    """e^{Cs} via the eigendecomposition C = V diag(mu) V^{-1}."""
    mu, V = np.linalg.eig(C)
    return (V @ np.diag(np.exp(mu * s)) @ np.linalg.inv(V)).real

def euler_step(f, x, h):
    """One forward-Euler step of size h for dx/dt = f(x)."""
    return x + h * f(x)

def rk4_step(f, x, h):
    """One classical Runge-Kutta step of size h for dx/dt = f(x)."""
    k1 = f(x)
    k2 = f(x + h / 2 * k1)
    k3 = f(x + h / 2 * k2)
    k4 = f(x + h * k3)
    return x + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

def solve(step, f, x, T, N):
    """N steps of size T/N from time 0 to time T; T < 0 runs the field backward."""
    for _ in range(N):
        x = step(f, x, T / N)
    return x
```

38.3.1 ニューラル ODE

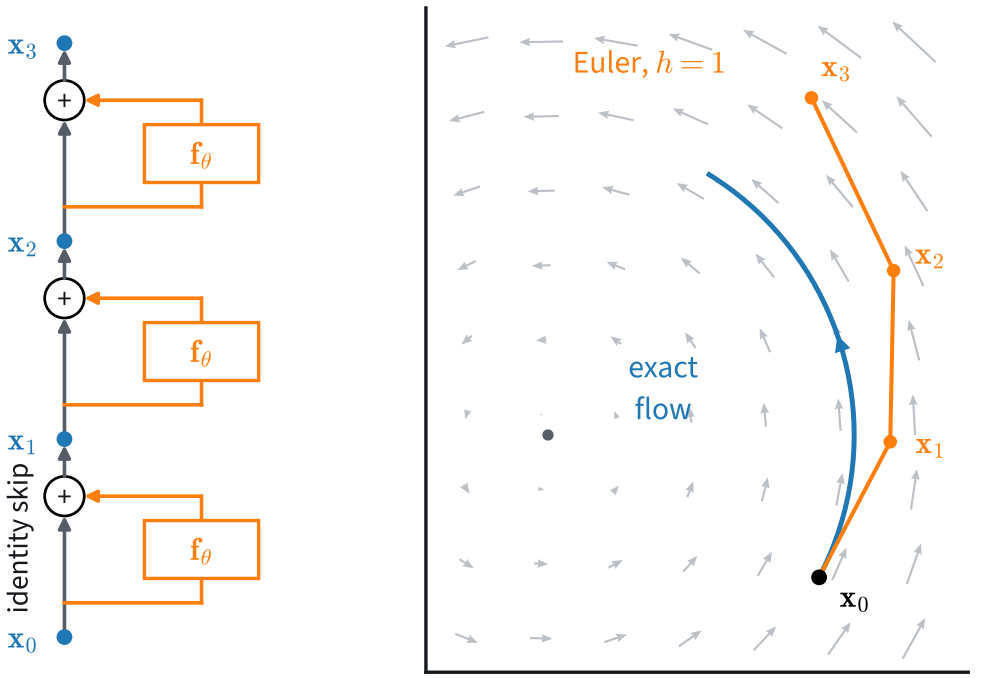
ニューラル ODE とは、残差ネットワークの深さを時間へと置き換えたものである。

38.3.1.1 残差ブロックから連続モデルへ

残差ブロック (He et al. 2016a) と、ODE $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ の前進オイラーステップを並べてみよう。

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathbf{f}_{\theta_l}(\mathbf{x}_l) \qquad \text{and} \qquad \mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n, t_n). \tag{38.14}$$

$h = 1$ かつ $\mathbf{f}(\cdot, t_l) = \mathbf{f}_{\theta_l}$ とおけば、2 つの更新は同じ式になる。したがって、ブロックごとに異なる重み（weight）を持つ ResNet は、各層に対応する時刻でサンプリングされた時間依存のベクトル場を、単位ステップ幅のオイラー法で積分したものにほかならない（Figure 38.7 はすべてのブロックが 1 つのベクトル場を共有する場合を描いている）。積分の終端時刻はその深さにあたり、10 個のブロックならベクトル場を時刻 10 まで積分することになる。



$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathbf{f}_\theta(\mathbf{x}_l) \quad \Leftrightarrow \quad \mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n) \;\; \text{with } h = 1$$

Figure 38.7: 左：1 つのベクトル場を共有する残差ブロックの積み重ね。各ブロックは恒等スキップ結合に学習された分岐を加える。右：同じ更新をサイズ $h = 1$ の前進オイラーステップとして見たもの。同じベクトル場の厳密なフローも併記しており、オイラーステップはそこからドリフトしていく。

極限をとるため、終端時刻 T を固定し、すべての時刻で 1 つのパラメータベクトル θ を共有させたまま、ステップを細かくしていく。層の言葉で言えば、各層の補正量を小さくしながらネットワークを深くしていくことになる（ベクトル場は依然として t に依存してよい。ネットワークが t

を入力として受け取ることで、1 つのパラメータベクトルでも異なるブロックの役割を果たせる）。N 個のステップ（ステップサイズ $h = T/N$、時刻 $t_n = nh$）をとると、次式が得られる。

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}_\theta(\mathbf{x}_n, t_n), \qquad \mathbf{x}_N \xrightarrow{N\to\infty} \mathbf{x}(T) = \mathbf{x}_0 + \int_0^T \mathbf{f}_\theta(\mathbf{x}(t), t)\,dt. \tag{38.15}$$

この収束（convergence）は 38.2 節で示したオイラー法の誤差評価から従う。すなわち、滑らかで $\mathbf{x}$ についてリプシッツ連続なベクトル場に対し、N ステップ後のオイラー反復は厳密解から $O(h)$ 以内に収まる。この極限として定まる対象が**ニューラル ODE** であり、出力を解 $\mathbf{x}(T)$（方程式 $\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}, t)$、初期条件 $\mathbf{x}(0) = \mathbf{x}_0$）とするモデルである。パラメータが定義するのはベクトル場であり、ODE ソルバ（オイラー法、RK4、適応的手法など）は後から選ぶものにすぎない。ソルバを変えれば計算される写像やそのコスト、誤差は変わるが、モデル自体は変わらない。

厳密な写像と数値計算された写像は、精度だけでなく定性的な性質まで異なり得る。ベクトル場 $\dot{x} = -x$ の厳密なフロー $\Phi_t(x) = e^{-t}x$ は、$\mathbb{R}$ 上の全単射をすべての t に対して定める。ところが、$h = 1$ のオイラーステップを 1 回適用すると $x + 1 \cdot (-x) = 0$ となり、あらゆる入力が 0 に写される。これに対し厳密な写像は、ベクトル場が t について連続かつ $\mathbf{x}$ について大域的リプシッツ連続である限り常に全単射である。なぜなら、$\mathbf{x}(T)$ からベクトル場を逆向きに走らせる問題もやはりただ 1 つの解を持ち、その解が出発点へと戻るからである（38.1.1.3 節）。

38.3.1.2 直線上での順序保存

可逆性は利点である一方、制約（constraint）でもある。ニューラル ODE は、その元となった残差ブロックの積み重ねと同等以上の表現力を持つのだろうか。実数直線上ではそうではなく、図を描けばその理由がわかる。(t, x) 平面において、リプシッツ連続なベクトル場の軌跡は決して交差しない。-1 と 1 を入れ替える 2 本の軌跡があるとすれば、途中で必ず出会わなければならない（Figure 38.8 の (a)）。

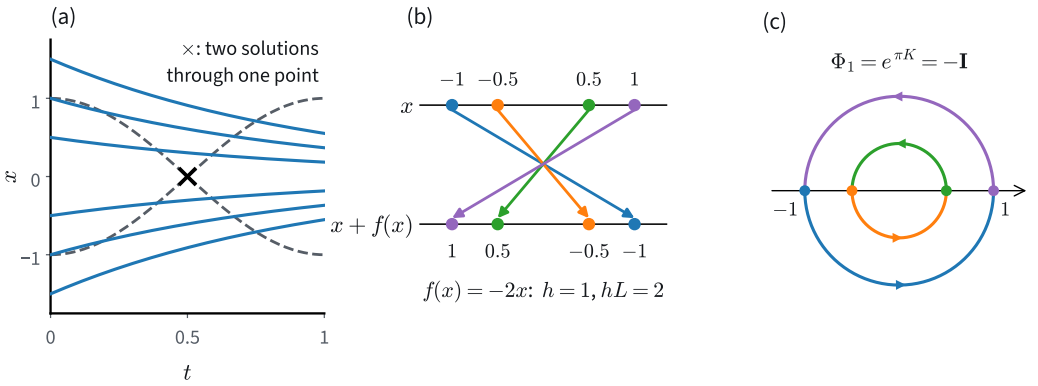


Figure 38.8: (a) 直線上ではリプシッツ連続なベクトル場（ここでは $\dot{x} = -x$）の軌跡は交差しないため、時刻 T の写像は順序を保つ。-1 と 1 を入れ替える破線の対を実現するには、1 点を通る 2 つの解が必要になる。(b) $x + f(x)$（ただし $f(x) = -2x$）という残差ステップは 1 ステップで 2 点を入れ替える。これは $hL = 2$ のオイラーステップであり、フローではない。(c) 次元を 1 つ増やすとこの障害は消える。角度 π の回転は、互いに交わらない半円に沿って $(x, 0)$ を $(-x, 0)$ へと運ぶ（色分けは (b) と同じ）。

命題（$\mathbb{R}$ 上のフローは順序を保つ） $f(x,t)$ は t について連続かつ x について大域的リプシッツ連続であるとし、$\Phi_t(x)$ を時刻 t における解（x を時刻 0 の初期値とするもの）と表す。このとき、任意の $T>0$ に対して写像 Φ_T は狭義単調増加である。

証明 $x<y$ とし、$\Phi_T(x)\geq\Phi_T(y)$ と仮定する。関数 $g(t)=\Phi_t(y)-\Phi_t(x)$ は連続で $g(0)>0$ かつ $g(T)\leq 0$ を満たすから、中間値の定理より $g(s)=0$ となる $s\in(0,T]$ が存在する。すなわち、2 本の軌跡は時刻 s にある点 z で出会う。ここで時刻 s から時間を逆向きにたどる。時間を反転させた 2 本の軌跡 $\tau\mapsto\Phi_{s-\tau}(x)$ と $\tau\mapsto\Phi_{s-\tau}(y)$ は、いずれも同じ初期値問題 $\dot{w}=-f(w,s-\tau)$、$w(0)=z$ の解であり、そのベクトル場もまたリプシッツ連続である。ピカール・リンデレーフの定理によりこの問題の解はただ 1 つであるため、逆向きの 2 本の軌跡は一致し、$\tau=s$ において $x=y$ となって矛盾（contradiction）が生じる。■

この証明の着想はただ 1 つ、軌跡が出会った時刻から逆向きに解の一意性を適用することにある。その帰結として、$\mathbb{R}$ 上のいかなるニューラル ODE も $x\mapsto -x$ を実現できない。この写像は -1 を 1 に、1 を -1 に移し、両者の順序を逆転させてしまうからである。

一方、残差ブロック $x+f(x)$（ただし $f(x)=-2x$）は、これを 1 ステップで実現する（Figure 38.8 の (b)）。このブロックは $h=1$ とした $\dot{x}=-2x$ のオイラーステップであるが、その厳密なフロー $e^{-2t}x$ が順序を逆転させることはない。両者が整合するのは、オイラーステップ $x+hf(x)$（ベクトル場のリプシッツ定数（Lipschitz constant）を L とする）が単調増加になる保証があるのは $hL<1$ のときに限られるからである。実際、$x>y$ に対して次式が成り立つ。

$$\big(x+hf(x)\big)-\big(y+hf(y)\big)\;\geq\;(x-y)-hL\,(x-y)\;=\;(1-hL)(x-y),$$

この右辺は $hL<1$ のとき正になる。いまの例では $hL=2$ であるため、ステップによって順序が逆転し得る。有限幅のステップはフローではなく、上の命題はそれを制約しない。

次元をもう 1 つ増やせば、この障害は取り除ける。回転のベクトル場 $\dot{\mathbf{x}}=\mathbf{K}\mathbf{x}$（38.1 節、ただし $\mathbf{K}=\left(\begin{smallmatrix}0&-1\\1&0\end{smallmatrix}\right)$）のフローは $e^{s\mathbf{K}}=\cos s\,\mathbf{I}+\sin s\,\mathbf{K}$ であり、平面を角度 s だけ回転させる。これを π 倍の速さで動かした $\dot{\mathbf{x}}=\pi\mathbf{K}\mathbf{x}$ の時刻 1 における写像は次のようになる。

$$\Phi_1=e^{\pi\mathbf{K}}=\cos\pi\,\mathbf{I}+\sin\pi\,\mathbf{K}=-\mathbf{I}.$$

このフローは半円に沿って $(x,0)$ を $(-x,0)$ へと運び、各軌跡は平面内で出会うことなくすれ違う（Figure 38.8 の (c)）。このように次元を追加する手法は**拡張**（augmentation）と呼ばれる (Dupont et al. 2019)。入力 x を $(x,0)$ として埋め込み（embedding）、平面上でフローを走らせたあと、第 1 座標を読み出すのである。フローとは空間の連続的な並べ替えであり、1 本の直線上には 2 点がすれ違う余地がない。

38.3.1.3 ソルバを通した訓練

ニューラル ODE の順伝播（forward pass）は N 回のソルバステップからなるループである。これを訓練する直接的な方法は、**離散化してから微分する**（discretize, then differentiate）ことである。すなわち、ソルバをそのまま計算グラフに書き下し、$\mathbf{x}_N$ に対する損失（loss）を計算して、展開されたステップ全体を逆伝播する。オイラー法をソルバに選べば、そのプログラムは 1 組の重みを共有する N ブロックの ResNet にほかならないため、通常の自動微分（automatic differentiation）で訓練（training）できる。

意図的に易しくした課題でこれを試してみよう。単位円上の 64 個の点を、中心 $(1.5,0)$、半軸 1.2 と 0.4 の楕円上の対応する点へと運ぶ課題であり、伸縮と平行移動の組み合わせにすぎない。アフィンなベクトル場 $\dot{\mathbf{x}}=\operatorname{diag}(\ln 1.2,\ln 0.4)\,\mathbf{x}+\mathbf{c}$ の厳密なフローは、適切な $\mathbf{c}$ を選べば誤差ゼロでこの変換を実現するため、もし誤差が見つかるとすれば解が存在しないせいではない。訓

練されたベクトル場は、高精度に積分したときにも目標点に到達するだろうか、それとも訓練時に使った 10 ステップでしか到達しないのだろうか。前者であれば訓練によってフローが見つかったことになり、後者であれば 10 ステップのプログラムが見つかったにすぎない。

ベクトル場は 32 個の tanh ユニットからなる隠れ層（hidden layer）を 1 つ持ち、損失は終端点での平均二乗誤差（mean squared error）とする。Adam（35.5 節）を用い、反復 0 から 400 までフルバッチで実行し、学習率（learning rate）は 0.05（最後の 100 反復では 0.005 に減衰）とする。同じ初期化（initialization）の重みから 2 回訓練する。1 回目は $N = 10$ 個のオイラーステップ（ステップサイズ $h = 0.1$）を通し、もう 1 回目は 1 ステップあたりベクトル場を 4 回評価する RK4 を 10 ステップ通して訓練する。

```
angle = np.linspace(0, 2 * np.pi, 64, endpoint=False)
X0_ring = np.stack([np.cos(angle), np.sin(angle)], axis=1)           # circle
X1_ring = np.stack([1.5 + 1.2 * np.cos(angle), 0.4 * np.sin(angle)], axis=1)
solvers = {'Euler': euler_step, 'RK4': rk4_step}
seed = int(rng.integers(2**31))               # one initialization for both solvers
```

出発点の円 `X0_ring`、目標の楕円 `X1_ring`、および共通の初期化シード `seed` を固定したうえで、`train` は 2 層の tanh ベクトル場 `net` を適合させる。具体的には、10 ステップの `solve(step, net, X, 1.0, 10)` を通して逆伝播することを、`solvers` 内の各ソルバについて実行する。

```
X, Y = jnp.array(X0_ring), jnp.array(X1_ring)
k1, k2 = jax.random.split(jax.random.key(seed))
glorot = jax.nn.initializers.glorot_uniform()  # Glorot-uniform weights, zero biases
params0 = {'W1': glorot(k1, (2, 32)), 'b1': jnp.zeros(32),
           'W2': glorot(k2, (32, 2)), 'b2': jnp.zeros(2)}
opt = optax.adam(optax.piecewise_constant_schedule(0.05, {300: 0.1}))

def net(params, X):
    return jnp.tanh(X @ params['W1'] + params['b1']) @ params['W2'] + params['b2']

def flow(params, step, X, T, N):
    """solve() as a fori_loop: N steps of size T/N along the field net(params)."""
    f = lambda x: net(params, x)
    return jax.lax.fori_loop(0, N, lambda n, X: step(f, X, T / N), X)

@jax.jit(static_argnames='step')
def train_step(params, opt_state, step):
    loss, grads = jax.value_and_grad(
        lambda p: ((flow(p, step, X, 1.0, 10) - Y) ** 2).mean())(params)
    updates, opt_state = opt.update(grads, opt_state)
    return optax.apply_updates(params, updates), opt_state, loss

def train(name, step):
    """Fit the field through 10 steps of one solver, from the shared params0."""
    params, opt_state, losses = params0, opt.init(params0), []
    for it in range(401):
        params, opt_state, loss = train_step(params, opt_state, step)
        if it % 200 == 0:
            losses.append(float(loss))
    print(f'{name}-trained: loss', ', '.join(f'{l:.1e}' for l in losses),
          'at iterations 0, 200, 400')
```

```
        return params

    trained = {name: train(name, step) for name, step in solvers.items()}
```

```
Euler-trained: loss 1.3e+00, 1.1e-05, 2.5e-06 at iterations 0, 200, 400
RK4-trained: loss 1.3e+00, 9.3e-06, 1.9e-06 at iterations 0, 200, 400
```

どちらの損失もオーダー 1 から 10^{-5} 未満へと減少する。訓練に用いた損失で判断する限り、2 つのベクトル場は同等に優秀である。

続いて 2 つのテストを行う。1 つ目は先ほどの問いを直接ぶつけるものである。訓練された各ベクトル場を、それぞれのソルバを用いて $N = 100$ および $N = 1000$ ステップ（いずれも $Nh = 1$）で積分する。N が大きいほど、計算される写像はそのベクトル場の時刻 1 における厳密なフローに近づく。セルを実行する前に、2 つの終端誤差がどうなるか予想してみてほしい。2 つ目のテストは、冒頭で挙げた最初の利点である可逆性に関するものである。オイラー法で訓練したベクトル場を順方向に N ステップ進めてから逆方向に N ステップ戻し、往復後の到達点が出発点からどれだけ離れているかを測る。厳密なフローであれば、すべての点が出発点に戻るはずである。

```
    dist = lambda P, Q: float(jnp.linalg.norm(P - Q, axis=1).mean())
    print('   N   Euler-trained   RK4-trained   Euler round trip')
    for N in (10, 100, 1000):
        err = [dist(flow(trained[k], solvers[k], X, 1.0, N), Y) for k in solvers]
        fwd = flow(trained['Euler'], euler_step, X, 1.0, N)
        back = flow(trained['Euler'], euler_step, fwd, -1.0, N)
        print(f'{N:4d}   {err[0]:13.1e}   {err[1]:11.1e}   {dist(back, X):16.1e}')
```

```
   N   Euler-trained   RK4-trained   Euler round trip
  10         2.1e-03       1.8e-03            5.5e-02
 100         1.7e-02       1.8e-03            5.6e-03
1000         1.9e-02       1.8e-03            5.6e-04
```

まず最後の列にある往復誤差から見よう。出発点からのずれは約 0.055（$N = 10$）、0.0055（$N = 100$）、0.00055（$N = 1000$）となり、ステップ数を 10 倍にするごとに 10 分の 1 に縮まる。前進オイラーステップと逆方向のオイラーステップの対はそれぞれ h^2 のオーダーの残差（residual）を残し、それが N 回積み重なってオイラー法の次数である $O(h)$ になる。ベクトル場を逆向きに走らせれば厳密なフローは元に戻るが、オイラー法のプログラムが元に戻るのは $h \to 0$ の極限においてのみである。なお、$hL < 1$ を満たすオイラー法のプログラムには確かに逆写像が存在する。各ステップが全単射になるからである（演習問題 4）。ただし、その逆写像を構成するのは陰的なステップであり、ステップ幅を $-h$ にしたオイラーステップではない。

次に終端誤差を見よう。訓練時と同じステップ数では、どちらのベクトル場も目標点から数千分の 1 以内に到達する。ところが驚くべきことに、ステップを細かくするとオイラー法で訓練したベクトル場はかえって悪化し、誤差が約 0.02 まで**増加**してから変化しなくなる。訓練が適合させたのは 10 ステップのプログラムだったのである。$N = 1000$ が近似しているそのベクトル場の時刻 1 における厳密なフローは、目標点から 0.02（楕円の長半径の 2% 未満）だけずれた位置に着地する。つまり、このベクトル場が定めるフローは 10 ステップのプログラムに近いものの、一致してはいない。

一方、RK4 で訓練したベクトル場は、10、100、1000 ステップのいずれでも同じ誤差を出力する。これほど滑らかなベクトル場であれば 10 ステップの RK4 で表示桁数まで正確にフローが計算されるため、プログラムへの適合がそのままフローへの適合となり、訓練後はステップ数を自由に選べる。粗いソルバを通して訓練したベクトル場はプログラムであり、高精度なソルバを通して

訓練したベクトル場はフローなのである。

ただし、その精度は訓練の反復ごとに支払わなければならない。RK4 を通した訓練では 1 反復あたりベクトル場を 40 回評価し、その 40 回の評価すべてを通して逆伝播するのに対し、オイラー法を通した訓練では 10 回で済む。オイラーステップを 100 回に増やしたり 2 次のソルバを使ったりしても、評価回数が変わるだけで本質は同じである。もう 1 つの選択肢は、訓練からソルバを完全に外すことである。閉形式で既知の目標値に対する回帰（regression）によって、1 データ例あたり 1 回のベクトル場評価で適合させれば、最初からフローそのものが学習される。38.6 節と 38.7 節では、このようにしてベクトル場を学習する。

38.3.2 随伴法

先ほどの訓練では 10 ステップのプログラムの勾配を用いたが、プログラムとフローは異なる写像であることがわかった。では、両者の勾配も異なるのだろうか。そもそも有限個のステップからなるプログラムではフローを厳密に計算できないのに、フローそのものの勾配を計算することなどできるのだろうか。

実は計算できる。その着想はリバースモード（reverse mode）の微分にすでに含まれている。リバースモードは、ある層の活性化値に対する損失の導関数（derivative）を層から層へと逆向きに伝えていくものであり、34.3.2 節ではこれをノードの随伴変数と呼んだ。随伴法は、あらゆる瞬間に 1 つずつノード $\mathbf{x}(t)$ があると考えて、同じ量を時間をさかのぼって運ぶ。中心となる対象は**随伴状態**（adjoint state）$\mathbf{a}(t) = \partial \mathcal{L} / \partial \mathbf{x}(t)$、すなわち時刻 t の状態に関する損失の勾配であり、これ自体がある線形常微分方程式に従うことがわかる。

38.3.2.1 スカラーの場合

一般論を述べる前に、すべての量が閉形式で書ける具体例を 1 つ最後まで計算してみよう。$\dot{x} = \theta x$、初期条件 $x(0) = x_0$、終端点での損失を $\mathcal{L} = \frac{1}{2} x(T)^2$ とする。直接計算すれば $x(T) = e^{\theta T} x_0$ であるから、$\partial x(T) / \partial \theta = T\, x(T)$ となり、次式が得られる。

$$\frac{\partial \mathcal{L}}{\partial \theta} = x(T)\, \frac{\partial x(T)}{\partial \theta} = T\, x(T)^2.$$

$\theta = T = x_0 = 1$ のとき、この値は $e^2 = 7.389$ である。

随伴法は別の道筋で同じ数値にたどり着く。時刻 t の状態に加えた微小変化 δ は、時刻 T までに $e^{\theta(T-t)}\delta$ へと拡大し、損失を $x(T)\, e^{\theta(T-t)}\delta$ だけ変化させるため、$a(t) = e^{\theta(T-t)} x(T)$ となる。この随伴状態は、ODE $\dot{a} = -\theta a$ を終端値 $a(T) = x(T)$（損失 $\ell(x) = \frac{1}{2}x^2$ の $x(T)$ における導関数の値）から逆向きに解いた解になっている。一方、パラメータの微小変化 $\delta\theta$ は各瞬間の速度を $x(t)\, \delta\theta$ だけ変え（$\partial f / \partial \theta = x$ であるため）、随伴状態がこれらの微小変化をそれぞれ損失の変化へと変換する。時間全体にわたって足し合わせると、次のようになる。

$$\int_0^T a(t)\, \frac{\partial f}{\partial \theta}\, dt = \int_0^T e^{\theta(T-t)} x(T)\, e^{\theta t} x_0\, dt = \int_0^T x(T)^2\, dt = T\, x(T)^2. \tag{38.16}$$

被積分関数は定数になっている。順方向に読むと x が増大する一方で a は同じ因子だけ縮小し、両者の積は常に $x(T)^2$ に保たれる（Figure 38.9）。求める勾配はその下の面積である。

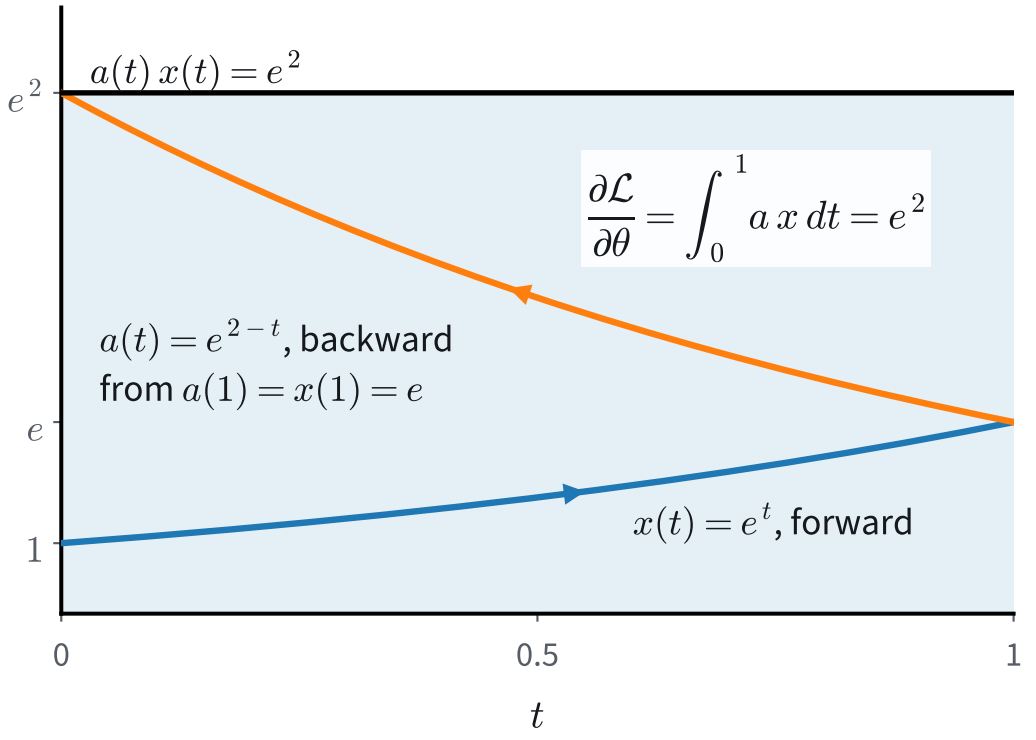


Figure 38.9: $\dot{x} = \theta x$（$\theta = 1$、$x_0 = 1$、$T = 1$）に対するスカラー随伴状態。状態は順方向に増大し、随伴状態は損失の勾配 $a(1) = x(1)$ から逆向きに積分され、両者の積は一定となる。パラメータ勾配は網掛け部分の面積 $Tx(T)^2 = e^2$ である。

38.3.2.2 随伴方程式

一般の場合もスカラーの場合と同じ構造を保つ。すなわち、損失の勾配から逆向きに走る随伴状態と、その随伴状態を $\partial \mathbf{f}/\partial \theta$ に対して積分するパラメータ勾配である。

命題（随伴方程式） **$\mathbf{f}$ は $\mathbf{x}$ と θ について連続微分可能とし、$\mathbf{x}(t)$ は $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \theta, t)$（パラメータを引数として明記したベクトル場 $\mathbf{f}_\theta$）の区間 $[0, T]$ における解（固定された初期値 $\mathbf{x}(0)$ から出発するもの）とする。また、$\mathcal{L} = \ell(\mathbf{x}(T))$（ただし ℓ は微分可能）とおく。このとき、随伴状態は次の線形 ODE を満たす。**

$$\dot{\mathbf{a}}(t) = -\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^{\top} \mathbf{a}(t), \qquad \mathbf{a}(T) = \nabla \ell(\mathbf{x}(T)), \tag{38.17}$$

ただしこれは時刻 T から逆向きに積分され、パラメータ勾配は次の積分で与えられる。

$$\frac{\partial \mathcal{L}}{\partial \theta} = \int_0^T \left(\frac{\partial \mathbf{f}}{\partial \theta}\right)^{\top} \mathbf{a}(t)\, dt, \tag{38.18}$$

ここで両方の偏微分（partial derivative）は軌跡 $(\mathbf{x}(t), \theta, t)$ に沿って評価される (Chen et al. 2018)。

証明（変分法による証明） パラメータに無限小の摂動 $\delta\theta$ を加える。$\mathbf{f}$ は C^1 級であるため解は θ について微分可能であり、軌跡の 1 次摂動 $\delta\mathbf{x}(t)$ はダイナミクスの**線形化**（linearization）に従う（ODE を θ で微分し、連鎖律（chain rule）を適用する）。

$$\delta\dot{\mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\,\delta\mathbf{x} + \frac{\partial \mathbf{f}}{\partial \theta}\,\delta\theta, \qquad \delta\mathbf{x}(0) = \mathbf{0}.$$

逆向き ODE Equation 38.17 によって $\mathbf{a}(t)$ を定義し、内積 $\mathbf{a}^{\top}\delta\mathbf{x}$ を時間で微分する。

$$\begin{aligned} \frac{d}{dt}\left(\mathbf{a}^{\top}\delta\mathbf{x}\right) &= \dot{\mathbf{a}}^{\top}\delta\mathbf{x} + \mathbf{a}^{\top}\delta\dot{\mathbf{x}} \\ &= -\mathbf{a}^{\top}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\,\delta\mathbf{x} + \mathbf{a}^{\top}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\,\delta\mathbf{x} + \mathbf{a}^{\top}\frac{\partial \mathbf{f}}{\partial \theta}\,\delta\theta = \mathbf{a}^{\top}\frac{\partial \mathbf{f}}{\partial \theta}\,\delta\theta. \end{aligned}$$

状態どうしの結合項は打ち消し合うが、随伴 ODE はまさにそうなるように定義したのである。0 から T まで積分し、左端で $\delta\mathbf{x}(0) = \mathbf{0}$ であることを用いると次式を得る。

$$\mathbf{a}(T)^{\top}\delta\mathbf{x}(T) = \int_0^T \mathbf{a}(t)^{\top}\frac{\partial \mathbf{f}}{\partial \theta}\,dt\;\delta\theta.$$

左辺は損失の 1 次変化 $\nabla\ell(\mathbf{x}(T))^{\top}\delta\mathbf{x}(T) = \delta\mathcal{L}$ であり、右辺は $\delta\theta$ に関して線形であって、その係数（coefficient）が Equation 38.18 である。残るは $\mathbf{a}$ が随伴状態に一致することの確認である。$\delta\theta = \mathbf{0}$ とし、微小変化 $\delta\mathbf{x}(s)$ を時刻 s で注入して同じ議論を行うと、内積 $\mathbf{a}^{\top}\delta\mathbf{x}$ は $[s, T]$ 上で一定となる。ゆえに $\delta\mathcal{L} = \mathbf{a}(T)^{\top}\delta\mathbf{x}(T) = \mathbf{a}(s)^{\top}\delta\mathbf{x}(s)$ が成り立ち、Equation 38.17 の解は $\partial\mathcal{L}/\partial\mathbf{x}(s)$ となる。$\blacksquare$

この逆向き ODE は $\mathbf{a}$ に関して線形であり、転置されたヤコビ行列をその係数行列に持つ。各ステップでは 1 回のベクトル・ヤコビ行列積（vector–Jacobian product）$\mathbf{a}^{\top}\partial\mathbf{f}/\partial\mathbf{x}$ を計算するが、

そのコストは $\mathbf{f}$ に対するリバースモードの計算 1 回分である（34.5 節）。同じパスで、パラメータが何個あろうともパラメータ勾配の被積分関数 $(\partial \mathbf{f}/\partial \theta)^{\top}\mathbf{a}$ が同時に求まる（なお、ReLU を用いたベクトル場は区分的 C^1 級にとどまるため、展開されたソルバに対して自動微分を区分ごとに適用することはできても、上の命題の適用対象には含まれない。tanh や GELU のような滑らかな活性化関数であれば命題の仮定を満たす）。

38.3.2.3 離散随伴としての誤差逆伝播法

これで、オイラー法のプログラムに対する誤差逆伝播法（backpropagation）を随伴 ODE と 1 行ずつ並べて比較できる。式を簡潔にするため、ベクトル場は自律的であるとし、そのヤコビ行列を $\mathbf{J} = \partial \mathbf{f}/\partial \mathbf{x}$ と書く。プログラム $\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n, \theta)$ を連鎖律で微分し、$\mathbf{a}_n = \partial \mathcal{L}/\partial \mathbf{x}_n$ とおく。1 ステップのヤコビ行列は $\mathbf{I} + h\mathbf{J}(\mathbf{x}_n)$ であるから、次式が成り立つ。

$$\begin{aligned} \mathbf{a}_n &= \big(\mathbf{I} + h\mathbf{J}(\mathbf{x}_n)\big)^{\top}\mathbf{a}_{n+1}, \qquad \mathbf{a}_N = \nabla \ell(\mathbf{x}_N), \\ \frac{\partial \mathcal{L}}{\partial \theta} &= \sum_{n=0}^{N-1} h \left(\frac{\partial \mathbf{f}}{\partial \theta}(\mathbf{x}_n)\right)^{\top} \mathbf{a}_{n+1}. \end{aligned} \tag{38.19}$$

これを、随伴 ODE Equation 38.17 を t_{n+1} から t_n へ逆向きに 1 ステップ進める陽的オイラーステップと比較してみよう。

$$\mathbf{a}_n = \mathbf{a}_{n+1} + h\,\mathbf{J}(\mathbf{x}_{n+1})^{\top}\mathbf{a}_{n+1}.$$

2 つの更新式の違いは、ヤコビ行列をどこで評価するか、すなわち順方向ステップの始点 $\mathbf{x}_n$ か終点 $\mathbf{x}_{n+1}$ かという点だけである。また、Equation 38.19 の和は積分 Equation 38.18 に対する長方形近似になっている。したがって、ソルバを通した誤差逆伝播法は**離散随伴**（discrete adjoint）であり、随伴 ODE にソルバを適用したものは**連続随伴**（continuous adjoint）を離散化したものである。文献ではこの 2 つの流儀を、それぞれ離散化してから最適化（discretize-then-optimize、先ほどの訓練における「離散化してから微分する」にあたる）および最適化してから離散化（optimize-then-discretize）と呼ぶ (Kidger 2022)。前者はプログラムの厳密な勾配であり、後者はフローの勾配を近似するものであって、オイラー法では両者の勾配は h の 1 次のオーダーで食い違う。本項の内容のなかで最も重要なのはこの事実である。

スカラーの例を見れば両方の勾配が一目でわかる。$\dot{x} = \theta x$ に対するオイラー法のプログラムは $x_N = (1 + h\theta)^N x_0$ であるから、$\partial x_N/\partial \theta = Nh(1+h\theta)^{N-1}x_0 = T\,x_N/(1+h\theta)$ となり、離散随伴からは $\partial \mathcal{L}/\partial \theta = T\,x_N^2/(1+h\theta)$ が得られる。$\theta = T = x_0 = 1$ とすると、この値は 6.116（$N = 10$）、7.244（$N = 100$）、7.374（$N = 1000$）となるのに対し、厳密値は $e^2 = 7.389$ である。その差 1.27、0.145、0.0148 は、N を 10 倍にするごとに 10 分の 1 に縮まる。

38.1 節のスパイラルのベクトル場 $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$（ただし $\mathbf{A} = \left(\begin{smallmatrix} -0.5 & -1 \\ 1 & -0.5 \end{smallmatrix}\right)$）を使えば、すべての要素が閉形式で書けたまま 2 次元で同じ検証ができる。線形なベクトル場ではヤコビ行列がどの点でも $\mathbf{A}$ で一定になるため、上に示した 2 つの漸化式は完全に一致する。したがって測るべき対象は、状態・随伴状態・和のすべてにオイラー法を用いたときに、両者がフローの厳密な勾配に対して持つ誤差である。

θ を行列 $\mathbf{A}$ の成分とし、$\mathcal{L} = \tfrac{1}{2}\|\mathbf{x}(T)\|^2$ とおく。軌跡は $e^{\mathbf{A}t}\mathbf{x}_0$、随伴状態は $\mathbf{a}(t) = e^{\mathbf{A}^{\top}(T-t)}\mathbf{x}(T)$（すなわち $\dot{\mathbf{a}} = -\mathbf{A}^{\top}\mathbf{a}$ の解）であり、$\partial \mathbf{f}/\partial A_{ij} = x_j \mathbf{e}_i$ であるから、勾配 Equation 38.18 は $\partial \mathcal{L}/\partial \mathbf{A} = \int_0^T \mathbf{a}(t)\,\mathbf{x}(t)^{\top} dt$ となる。

次のセルでは、逆方向スイープ Equation 38.19 を $T = 2$ について手で書き下す。出力の 1 行目は、同じプログラムに対する中心有限差分（finite difference）とこのスイープを $N = 1000$ ステップで比較するものであるが、これで確認できるのはスイープの実装に誤りがないことだけである。本当に重要なのは、厳密なフローの勾配との距離である。そのためにセルの後半では、連続随伴の積分を 2000 区間の台形公式で評価し（$\mathbf{a}$ と $\mathbf{x}$ は行列指数関数から計算）、3 通りのステップ数における離散随伴と比較する。

```
def loss_and_grad_unrolled(A, x0, T, N):
    """L = ||x_N||^2 / 2 through N unrolled Euler steps; dL/dA by hand-written
    reverse mode, the discrete adjoint."""
    h, xs = T / N, [x0]
    for _ in range(N):
        xs.append(xs[-1] + h * (A @ xs[-1]))   # forward sweep, tape the states
    a, gA = xs[-1].copy(), np.zeros_like(A)    # a_N = dL/dx_N = x_N
    for n in range(N - 1, -1, -1):             # backward sweep: one VJP per step
        gA += h * np.outer(a, xs[n])           # dL/dA += h a_{n+1} x_n^T
        a = a + h * (A.T @ a)                  # a_n = (I + hA)^T a_{n+1}
    return 0.5 * xs[-1] @ xs[-1], gA

T = 2.0
loss = lambda A: loss_and_grad_unrolled(A, x0, T, 1000)[0]
g_fd = np.array([(loss(A + 1e-5 * E) - loss(A - 1e-5 * E)) / 2e-5
                 for E in np.eye(4).reshape(4, 2, 2)]).reshape(2, 2)
gap_fd = np.abs(loss_and_grad_unrolled(A, x0, T, 1000)[1] - g_fd).max()
print(f'N =  1000: |backward sweep - central differences| = {gap_fd:.1e}')
n_quad = 2000                                  # continuous adjoint, trapezoid rule
ts = np.linspace(0.0, T, n_quad + 1)
xT = expm_eig(A, T) @ x0
AX = np.stack([np.outer(expm_eig(A.T, T - s) @ xT, expm_eig(A, s) @ x0)
               for s in ts])                   # a(s) x(s)^T
g_cont = (T / n_quad) * (AX[1:-1].sum(axis=0) + 0.5 * (AX[0] + AX[-1]))
for N in (100, 1000, 10000):
    gap = np.abs(loss_and_grad_unrolled(A, x0, T, N)[1] - g_cont).max()
    print(f'N = {N:5d}: |discrete - continuous adjoint|     = {gap:.1e}')
```

```
N =  1000: |backward sweep - central differences| = 1.2e-10
N =   100: |discrete - continuous adjoint|     = 2.9e-02
N =  1000: |discrete - continuous adjoint|     = 2.8e-03
N = 10000: |discrete - continuous adjoint|     = 2.8e-04
```

逆方向スイープは有限差分と約 10^{-10} の桁まで一致しており、これは有限差分自体の精度に等しい。一方、フローの勾配との差は 2.9×10^{-2} から 2.8×10^{-3}、さらに 2.8×10^{-4} へと減少し、スカラーの場合と同様に N を 10 倍にするごとに 10 分の 1 になる。

38.3.2.4 メモリとドリフト

随伴 ODE はヤコビ行列を評価するため、逆方向の各ステップで $\mathbf{x}(t)$ を必要とする。これを用意する方法は 2 つある。1 つは誤差逆伝播法と同様に順方向の状態をすべて保存しておく方法で、N に比例するメモリを消費する。もう 1 つは $\dot{\mathbf{x}} = \mathbf{f}$ を $\mathbf{x}(T)$ から $\mathbf{a}$ と一緒に逆向きに再積分する方法で、1 ステップあたりベクトル場の評価が 1 回増える代わりにメモリ使用量を N に対して定数に抑えられる。後者はうますぎる話に聞こえるが、最小の具体例を見ればその裏がわかる。誤差を増幅してしまうのである。

$\dot{x} = -\lambda x$（ただし $\lambda > 0$）に対し、逆方向の積分は $\dot{x} = +\lambda x$ を解くことになる。したがって、保存された $x(T)$ に含まれる絶対誤差 δ は、$t = 0$ に戻るころには $e^{\lambda T}\delta$ に膨れ上がる。

では、δ はどれほどの大きさだろうか。2 つのモードを持つ状態を想像してみよう。一方は $\lambda = 50$ という速い減衰率（38.2.3 節の硬い方程式の例における速いレート）で減衰し、もう一方はほとんど動かないとする。速いモードが対角方向を向いており、どの座標にもその成分が含まれているとしよう。時刻 T までに速いモードは減衰し切り、保存される各座標の大きさはオーダー 1 となって、倍精度（double precision）ではそれぞれ約 10^{-16} の桁で丸められる。丸め処理はモードの存在など知らないため、丸め誤差の一部は速いモードの方向に乗り、$\delta \approx 10^{-16}$ となる。その結果、再構成された初期値は 2.7×10^{-11}（$T = 0.25$）、7×10^{-6}（$T = 0.5$）、さらには 5×10^{5}（$T = 1$）もずれてしまい、状態そのものより大きくなってしまう。倍精度浮動小数点数が保持できるのは約 16 桁であるのに対し、増幅率 $e^{50} \approx 5 \times 10^{21}$ はそれを 5 桁も上回るからである。演習問題 3 ではこの誤差の増大を実際に測定する。

固有値（eigenvalue）の実部が -0.5 であるスパイラルの例では、$T = 2$ にわたる増幅率はわずか e にとどまる。順方向に強く収縮するフローは逆方向には強く膨張するため、再積分はメモリを節約する引き換えにドリフトを招く。硬いベクトル場に対しては、**チェックポイント保存**（checkpointing、34.5.2.5 節）の方が堅牢な選択である。少数のチェックポイントでのみ状態を保存し、その間の状態は直前のチェックポイントから順方向に再計算するため、フローに逆らって積分することが一切ない。$\sqrt{N}$ ステップごとにチェックポイントを置けば、逆方向スイープが保持する状態数は約 $2\sqrt{N}$ 個、すなわち 63 個（1000 個ではなく、$N = 1000$ の場合）で済み、追加コストは順伝播 1 回分のみである。

以上で勾配を求める方法が 3 通り出そろった。どれを使うべきだろうか。答えは、ソルバを通した誤差逆伝播法に、メモリが足りないときはチェックポイント保存を組み合わせる方法である。これなら実際に実行されるプログラムの厳密な勾配が返される。連続随伴はその勾配の収束先であり、理論的な考察を行うための式である。状態の逆方向再積分は、硬くないベクトル場に限って使える最後の手段にすぎない。

38.3.3 連続正規化フロー

ニューラル ODE は点を移動させる。最尤法で訓練される生成モデル（generative model）は、それぞれの出力がどれほど確からしいかも答えなければならないため、ここからはフローに沿った確率密度の変化を追っていく。

38.3.3.1 フローに沿った密度

サンプルをフローに通して押し出せば、フローは生成器（generator）になる。$\mathbf{x}(0)$ を単純な密度 p_0 からサンプリングして積分すればよい。このような生成器を最尤法で訓練するには、出力される点の密度が必要になる。**正規化フロー**とはこのようにして訓練される可逆なネットワークであり、その密度は 36.2 節の変数変換の公式から求まる。すなわち、可逆で微分可能な写像 $\mathbf{g}$（ヤコビ行列を $\mathbf{J}_{\mathbf{g}}$ とする）に対し、次式が成り立つ。

$$\log p_{\text{out}}(\mathbf{g}(\mathbf{x})) = \log p_{\text{in}}(\mathbf{x}) - \log \left|\det \mathbf{J}_{\mathbf{g}}(\mathbf{x})\right|. \tag{38.20}$$

対数密度（log density）は、$\mathbf{g}$ が $\mathbf{x}$ の近傍の体積を引き伸ばす倍率の対数だけ減少する。多数の層から構築された正規化フローでは、層ごと・サンプルごとに 1 回ずつ対数行列式を計算しなければならない。一般の行列式には $O(d^3)$ 回の演算がかかり、約 2×10^{10} 回にも達する（次元 $d = 3072$ の 32×32 カラー画像の場合）。そのため離散フローでは、行列式が対角成分の積になる三角ヤコビ行列や、$\det(\mathbf{I} + \mathbf{u}\mathbf{v}^{\top}) = 1 + \mathbf{v}^{\top}\mathbf{u}$ のような「単位行列＋低ランク行列」のヤコビ行列を持つ層へとアーキテクチャを制限する。これらはいずれも $O(d)$ で行列式を計算できる。

連続時間ではこの公式が単純になる。まず 1 次元から始めよう。ベクトル場 $\dot{x} = \theta x$（$\mathbb{R}$ 上）のフローは $\Phi_t(x) = e^{\theta t}x$ であり、あらゆる区間を $e^{\theta t}$ 倍に引き伸ばす。フローに運ばれる密度はその引き伸ばし率で割られるため、$\log p_t(x(t)) = \log p_0(x_0) - \theta t$ となり、したがって $\frac{d}{dt}\log p_t(x(t)) = -\theta$ である。この定数 θ はベクトル場の 1×1 ヤコビ行列であり、ゆえにそのトレースでもある。

2 次元では引き続きスパイラルを例にとろう。そのフロー $e^{\mathbf{A}t} = e^{-t/2}e^{t\mathbf{K}}$ は回転のあとに $e^{-t/2}$ 倍の縮小を施す変換である（38.1 節）。円板は半径が $e^{-t/2}$ 倍の円板へと移されるため、面積は e^{-t} 倍になり、各軌跡に沿った密度は e^{t} 倍に上昇する。ここで $\operatorname{tr}\mathbf{A} = -1$ であるから、対数密度の変化率はやはりトレースに負号をつけたものに一致する。Figure 38.10 は、この様子を面積保存のベクトル場および面積を膨張させるベクトル場と並べて描いたものである。スパイラルの最も濃い円板（$t = 1.2$）の面積は、最も薄い円板の $e^{-1.2} \approx 0.30$ 倍になっている。

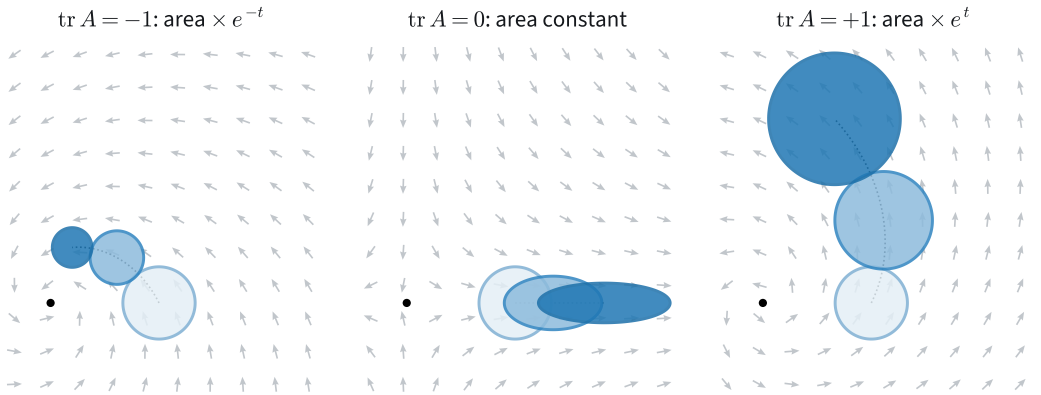


Figure 38.10: 3 つの線形ベクトル場の厳密なフローによって運ばれるサンプルの円板（$t = 0$、0.6、1.2 を表示し、色が濃いほど後の時刻を表す。点線は中心の軌跡）。軌跡に沿った密度は因子 $e^{-t\operatorname{tr}\mathbf{A}}$ 倍に変化する。フローが収縮するところでは密度が上昇し（左：スパイラル）、面積保存の鞍点では形状が変わっても密度は一定に保たれ（中央）、フローが膨張するところでは密度が低下する（右）。

38.3.3.2 瞬間的な変数変換

上の 2 つの例はいずれも線形であり、どちらでも対数密度の変化率はトレースの負値に等しかった。一般の場合を示すには、行列式に関する事実が 1 つあればよい。短いステップの間ではフロー写像（flow map）は恒等写像に近く、単位行列に近い行列の行列式はそのトレースによって決まるのである。

補題 任意の正方行列 $\mathbf{J}$ に対して、$\det(\mathbf{I} + h\mathbf{J}) = 1 + h\operatorname{tr}\mathbf{J} + O(h^2)$ が成り立つ。

証明 置換による行列式の定義式（33.2.3.2 節）は、各行・各列から 1 つずつ成分を選んだ積に符号をつけて、すべての置換にわたり足し合わせたものである。恒等置換以外の任意の置換は少なくとも 2 つの非対角成分を選ぶが、それらはそれぞれ $O(h)$ の大きさを持つため、寄与は $O(h^2)$ となる。恒等置換からの寄与は $\prod_i (1 + hJ_{ii}) = 1 + h\sum_i J_{ii} + O(h^2)$ である。$\blacksquare$

命題（瞬間的な変数変換） $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ において、$\mathbf{f}$ は $(\mathbf{x}, t)$ について連続微分可能かつ $\mathbf{x}$ について大域的リプシッツ連続であるとする。このとき各フロー写像 Φ_t は C^1 級の全単射であり、その逆写像も C^1 級となる（C^1 級のベクトル場の解は初期値に関して C^1 級であり (Hairer et al. 1993)、逆写像はベクトル場を逆向きに走らせるためである）。また、$\mathbf{x}(0) \sim p_0$ とする（ただし

p_0 は正の値をとる微分可能な密度）。このとき密度 p_t（$\mathbf{x}(t)$ の確率密度）が存在して正の値をとり、各軌跡に沿って次式が成り立つ (Chen et al. 2018)。

$$\frac{d}{dt}\log p_t(\mathbf{x}(t)) = -\operatorname{tr}\!\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)(\mathbf{x}(t), t). \tag{38.21}$$

証明密度の存在は Equation 38.20 を全単射 Φ_t に適用することで従い、それが正であることは、p_0 が正であり微分可能な逆写像を持つ写像のヤコビ行列が正則であることから従う。短い時間区間において、フローは写像 $\Phi_{t\to t+h}(\mathbf{x}) = \mathbf{x} + \int_t^{t+h} \mathbf{f}(\mathbf{x}(s), s)\, ds$ によって進む。$\mathbf{x}$ で微分し $\mathbf{J}$ の連続性を用いると、そのヤコビ行列は $\mathbf{I} + h\mathbf{J} + o(h)$ となる。ただし $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{x}$ は $(\mathbf{x}(t), t)$ で評価したものであり、$o(h)$ は h より速く 0 に収束する項を表す。この写像に Equation 38.20 を適用し、上の補題を用いて展開する（補題は高次の摂動を許容するため、$h\mathbf{J}$ を $h\mathbf{J} + o(h)$ に置き換えても行列式の変化は $o(h)$ にとどまる）。十分小さい h に対して行列式は 1 に近く正になるため、絶対値は外せて次式となる。

$$\begin{aligned}\log p_{t+h}(\mathbf{x}(t+h)) &= \log p_t(\mathbf{x}(t)) - \log\det(\mathbf{I} + h\mathbf{J} + o(h)) \\ &= \log p_t(\mathbf{x}(t)) - h\operatorname{tr}\mathbf{J} + o(h),\end{aligned}$$

ここでは $\log(1+u) = u + O(u^2)$ を用いた。両辺から $\log p_t(\mathbf{x}(t))$ を引き、h で割って $h \to 0$ とすればよい。$\blacksquare$

命題の仮定は見た目ほど厳しいものではない。求めているのはすべてのフロー写像が滑らかな全単射であることだけであり、tanh のような滑らかでリプシッツ連続な活性化関数を持つベクトル場であれば自然に満たされる。命題の実質的な内容は、トレースを含むあの 1 行の式にある。

ヤコビ行列のトレースはベクトル場の**発散**（divergence）$\nabla \cdot \mathbf{f}$ であり、フローが局所的に体積を膨張させる変化率を表す。ベクトル場がサンプルを押し広げるところ（$\operatorname{tr}\mathbf{J} > 0$）では各軌跡に沿った対数密度は減少し、サンプルを圧縮するところでは上昇する。Equation 38.21 を軌跡に沿って積分すると次式が得られる。

$$\log p_T(\mathbf{x}(T)) = \log p_0(\mathbf{x}(0)) - \int_0^T \operatorname{tr}\!\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right) dt. \tag{38.22}$$

離散フローにおける層ごとの対数行列式の和は、トレースの積分へと姿を変えた。任意の ODE ソルバで対 $(\mathbf{x}, \log p)$ を積分することによりこの量を計算するモデルが、**連続正規化フロー**（continuous normalizing flow、CNF）である (Chen et al. 2018)。このモデルでは $\mathbf{f}$ のアーキテクチャに一切の制約を課す必要がない。与えられた点 $\mathbf{x}(T)$ の密度を評価するには、その点から $\mathbf{x}(0)$ までベクトル場を逆向きに積分しながら途中でトレースを累積し、到達点での p_0 を読み取ればよい。拡散モデルの尤度もこの方法で計算される（17.6 節）。この公式には時間の向きに対する非対称性はなく、38.6 節や 38.7 節の生成器のようにノイズを終端時刻に置きデータを $t = 0$ に置く設定でも、まったく同じ公式がそのまま使える。

線形なベクトル場ではトレースの積分は $t\operatorname{tr}\mathbf{A}$ となり、Equation 38.22 を Equation 38.20（写像 $e^{\mathbf{A}t}$ に対するもの）と見比べれば $\log\det e^{\mathbf{A}t} = t\operatorname{tr}\mathbf{A}$ が得られる。対角化可能な $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$ に対しては、この恒等式は固有値から 1 行で導くこともできる。

$$\det e^{\mathbf{A}t} = \det\left(\mathbf{V} e^{\Lambda t}\mathbf{V}^{-1}\right) = \prod_j e^{\Lambda_{jj} t} = e^{t\operatorname{tr}\mathbf{A}}, \tag{38.23}$$

一方、先ほどの 2 つの公式の比較からは、対角化可能性を問わずあらゆる $\mathbf{A}$ に対してこの恒等式が従う。

上の命題は軌跡を 1 本ずつ追跡したものである。同じ法則を密度全体に対して一度に書き下せば流体力学における連続の式（continuity equation）となり、38.5 節ではノイズが加える項と合わせてこれを導出する。

38.3.3.3 トレース積分の検証

線形なベクトル場はトレースが定数で積分が単なる $t\,\mathrm{tr}\,\mathbf{A}$ になるため、命題の検証としては物足りない。経路に沿ってトレースが変化する非線形なベクトル場こそ本格的なテストになる。平面上のベクトル場 $\mathbf{f}(\mathbf{x}) = \mathbf{W}_2 \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$（隠れユニット（hidden unit）数 8、重みはランダム）を考えよう。ヤコビ行列とトレースは次のように陽に書ける。

$$\mathbf{J} = \mathbf{W}_2\,\mathrm{diag}(\mathbf{s})\,\mathbf{W}_1, \qquad \mathbf{s} = 1 - \tanh^2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \qquad \mathrm{tr}\,\mathbf{J} = \sum_i s_i\,(\mathbf{W}_1\mathbf{W}_2)_{ii},$$

ここではトレースの巡回性 $\mathrm{tr}\,(\mathbf{W}_2\,\mathrm{diag}(\mathbf{s})\,\mathbf{W}_1) = \mathrm{tr}\,(\mathrm{diag}(\mathbf{s})\,\mathbf{W}_1\mathbf{W}_2)$ を用いた。次のセルでは、対 $(\mathbf{x}, \int \mathrm{tr}\,\mathbf{J}\,dt)$ を RK4 で $T = 1$ まで積分し、その経路を記録する。途中で現れたトレースの最小値と最大値を出力したうえで、トレース積分を $\log|\det D\Phi_T|$ と比較する。ただし、フロー写像のヤコビ行列 $D\Phi_T$ は、フロー写像自体の中心有限差分（finite difference）から計算する。

```
n_hidden = 8
W1, b1 = rng.standard_normal((n_hidden, 2)), rng.standard_normal(n_hidden)
W2 = rng.standard_normal((2, n_hidden)) / np.sqrt(n_hidden)
field = lambda x: W2 @ np.tanh(W1 @ x + b1)
trJ = lambda x: np.sum((1 - np.tanh(W1 @ x + b1) ** 2)
                       * np.einsum('ij,ji->i', W1, W2))
aug = lambda z: np.append(field(z[:2]), trJ(z[:2]))  # d/dt (x, int tr J)
xa, N = rng.standard_normal(2), 200
zs = [np.append(xa, 0.0)]                       # RK4 path of (x, int tr J)
for _ in range(N):
    zs.append(rk4_step(aug, zs[-1], 1.0 / N))
zT, tr_path = zs[-1], [trJ(z[:2]) for z in zs]
D = np.zeros((2, 2))                            # D Phi_1 by central differences
for j in range(2):
    e = np.zeros(2); e[j] = 1e-5
    D[:, j] = (solve(rk4_step, field, xa + e, 1.0, N)
               - solve(rk4_step, field, xa - e, 1.0, N)) / 2e-5
logdet = np.log(abs(np.linalg.det(D)))
print(f'tr J along the path: from {min(tr_path):+.3f} to {max(tr_path):+.3f}')
print(f'trace integral {zT[2]:+.6f}  log|det D Phi| {logdet:+.6f}  '
      f'gap {abs(zT[2] - logdet):.1e}')
```

```
tr J along the path: from +0.064 to +0.222
trace integral +0.148995  log|det D Phi| +0.148995  gap 9.3e-11
```

トレースは確かに変動しており、1 行目の出力には経路上でトレースが動いた区間が示される。その幅は重みの乱数シードに依存するが、0.1 以上動くのが普通であり、乱数によっては区間が 0 をまたいで、経路のある部分では面積を膨張させ別の部分では圧縮することもある。それでもトレース積分と対数行列式は 8 桁以上の精度で一致しており、これはソルバと有限差分自体の精度に達している。命題の証明は線形性を一切使っていなかったのである。

38.3.3.4 Hutchinson トレース推定量

上の $\operatorname{tr}\mathbf{J}$ の式は、隠れ層 1 層のベクトル場という特別な構造を利用したものであった。一般のネットワークでトレースを求めるには、d 個ある $\mathbf{J}$ の対角成分が必要になり、座標ごとに 1 回ずつ、計 d 回のリバースモード計算がかかる。トレースへの書き換えによって $O(d^3)$ の行列式はすでに d 回のパスへと削減されたが、ランダム射影（projection）を使えばこの d 回のパスをさらに 1 回へと減らし、ノイズは乗るものの不偏な推定値を得ることができる (Hutchinson 1989)。

命題（Hutchinson 推定量） 任意の行列 $\mathbf{M} \in \mathbb{R}^{d\times d}$ と、任意の確率ベクトル（random vector）$\epsilon \in \mathbb{R}^d$（ただし $\mathbb{E}[\epsilon\epsilon^\top] = \mathbf{I}$ を満たすもの。たとえば標準ガウスベクトルや、独立に等確率で ± 1 をとるラデマッハベクトル）に対し、次式が成り立つ。

$$\operatorname{tr}(\mathbf{M}) = \mathbb{E}\big[\epsilon^\top \mathbf{M}\epsilon\big]\,. \tag{38.24}$$

証明 トレースの巡回性と、期待値（expectation）との交換を可能にするトレースの線形性により、$\mathbb{E}[\epsilon^\top\mathbf{M}\epsilon] = \mathbb{E}[\operatorname{tr}(\mathbf{M}\epsilon\epsilon^\top)] = \operatorname{tr}(\mathbf{M}\,\mathbb{E}[\epsilon\epsilon^\top]) = \operatorname{tr}(\mathbf{M})$ が成り立つ。$\blacksquare$

この推定量が $\mathbf{J}$ に触れるのは 1 回のベクトル・ヤコビ行列積 $\epsilon^\top\mathbf{J}$ を通じてのみである。これは $\mathbf{f}$ に対して ϵ をシードとするリバースモード計算を 1 回行うことに相当し、ヤコビ行列自体は一切構成しない。隠れ層が 1 つのベクトル場では、この積は次のように陽に書ける。

$$\epsilon^\top\mathbf{J} = \big((\epsilon^\top\mathbf{W}_2)\odot \mathbf{s}\big)\,\mathbf{W}_1,$$

その計算コストは $\mathbf{f}$ の評価とほぼ同じである。CNF はベクトル場を評価するたびにこの積を 1 回計算するため、尤度の計算に必要なのはベクトル場の評価 1 回あたり追加の逆伝播 1 回分だけである。これが Grathwohl et al. (2019) によって FFJORD と名付けられた手法である。

上の命題はプローブに関する平均がトレースに収束することを保証するが、その収束の速さまでは述べていない。プローブを 1 個しか使わない場合、まさにそのばらつきこそが問題のすべてとなる。$\mathbf{S} = (\mathbf{J}+\mathbf{J}^\top)/2$ とおくと、1 個のラデマッハプローブの分散（variance）は $2\sum_{i\neq j} S_{ij}^2$ であり、1 個のガウスプローブの分散は対角成分まで含めた同じ形の和 $2\sum_{i,j} S_{ij}^2$ となる（演習問題 5）。トレースが見ているのは $\mathbf{J}$ の対角成分だけであり、ラデマッハプローブのノイズが見ているのは非対角成分だけである。

トレースの値が予測できる $d = 50$ 次元のベクトル場でこの推定量を試してみよう。64 個の tanh ユニットを持つランダムな 1 隠れ層部分に原点への引き戻し項を加えた $\mathbf{f}(\mathbf{x}) = \mathbf{W}_2\tanh(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1) - \mathbf{x}$ を考える。ヤコビ行列は $\mathbf{W}_2\operatorname{diag}(\mathbf{s})\,\mathbf{W}_1 - \mathbf{I}$ となるため、引き戻し項はトレースに -50 寄与し、ランダムな部分はオーダー 1 の値だけ寄与する。また引き戻し項は各推定値に $-\epsilon^\top\epsilon$ を加える。ラデマッハプローブではこの値が厳密に -50 となり、ノイズはすべてランダムな部分から生じる。一方、ガウスプローブでは $\epsilon^\top\epsilon$ 自体が確率的に揺らぎ、その代償は先ほどの 2 つの分散公式に表れている。両者の差は対角項 $2\sum_i S_{ii}^2$ であり、いま各 $S_{ii} = J_{ii}$ は -1 に近いため、ガウスプローブの分散はラデマッハプローブの分散を約 $2\cdot 50 = 100$ 上回るはずである。次のセルではそれぞれの種類のプローブを $m = 1000$ 個ずつ引き、各 $\epsilon^\top\mathbf{J}$ をリバースモードで計算する。

```
d, n_hidden, m = 50, 64, 1000
W1 = rng.standard_normal((n_hidden, d)) / np.sqrt(d)
b1 = 0.1 * rng.standard_normal(n_hidden)
W2 = rng.standard_normal((d, n_hidden)) / np.sqrt(n_hidden)
x = rng.standard_normal(d)
s = 1 - np.tanh(W1 @ x + b1) ** 2
tr_tanh = np.sum(s * np.einsum('ij,ji->i', W1, W2))  # trace of W2 diag(s) W1
```

```
E = rng.choice([-1.0, 1.0], size=(m, d))        # Rademacher probes, one per row
E_gauss = rng.standard_normal((m, d))           # Gaussian probes, one per row
```

ここで、リバースモードのベクトル・ヤコビ行列積 $\epsilon_k^\top \mathbf{J}$ を点 x において全 $m = 1000$ 個のラデマッハプローブ（E）およびガウスプローブ（E_gauss）に対して評価し、得られたトレース推定値 $\epsilon_k^\top \mathbf{J} \epsilon_k$ とプローブ 1 個あたりの標準偏差（standard deviation）を厳密なトレース tr_tanh - d と比較する。

```
W1j, b1j, W2j = (jnp.array(w) for w in (W1, b1, W2))
field = lambda x: W2j @ jnp.tanh(W1j @ x + b1j) - x
_, vjp = jax.vjp(field, jnp.array(x))          # reverse mode at x
for name, P in (('Rademacher', E), ('Gaussian', E_gauss)):
    rows = jax.vmap(vjp)(jnp.array(P))[0]      # row k: eps_k^T J
    est = (rows * P).sum(axis=1)               # eps_k^T J eps_k
    mean, sd = float(est.mean()), float(est.std())
    print(f'{name:10s}: exact {tr_tanh - d:+6.2f}, mean of {m} probes '
          f'{mean:+6.2f} +- {sd / m ** 0.5:.2f}, one probe sd {sd:.2f}')
```

```
Rademacher: exact -50.60, mean of 1000 probes -50.48 +- 0.15, one probe sd 4.61
Gaussian  : exact -50.60, mean of 1000 probes -50.50 +- 0.36, one probe sd 11.51
```

どちらの平均（mean）も厳密なトレースから標準誤差（standard error）の数倍以内に収まっており、不偏推定量として期待されるとおりの振る舞いを示す。ラデマッハプローブ 1 個の標準偏差は約 5、すなわちトレースの 10 分の 1 である。厳密値の計算には 50 回のパスを要するところ、たった 1 回の逆伝播でトレースを 10% 程度の精度で特定できている。一方、ガウスプローブ 1 個の標準偏差は約 11 であり、$11^2 - 5^2 = 96$ となって予測値の 100 に近い。ラデマッハプローブの方がノイジーになることは決してなく、この例でははるかに静かである。

ただし、10% という相対精度はこのベクトル場の性質によるものであり、推定量自体の性質ではない。もし引き戻し項がなければ、すべてのラデマッハ推定値がちょうど $+50$ だけシフトし、標準偏差は 5 のまま変わらない一方、真のトレースはオーダー 1 になるため、単一のプローブからはほとんど何の情報も得られない。高次元のモデルがこれら 2 つの状況のどちらに近いかは、演習問題 5 で扱う。

尤度の計算に必要なのは 1 点でのトレースではなく、軌跡に沿った積分 $\int_0^T \operatorname{tr} \mathbf{J}\, dt$ であり、ソルバのステップごとに 1 回ずつ推定を行うことになる。このときプローブの与え方には、求解全体を通して 1 個のプローブを固定する方式と、ステップごとに新しいプローブを引き直す方式の 2 通りがある。どのような結果が予想されるだろうか。固定プローブは和 $\sum_n h\, \mathbf{J}(\mathbf{x}_n)$ を 1 つの行列とみなしてそのトレースを 1 回推定するのに等しいため、ノイズの大きさは N によらず単一推定のノイズのまま変わらない。これに対し新規プローブでは各ステップの誤差が独立になり、それぞれに $h = 1/N$ の重みがかかるため、和のノイズは $1/\sqrt{N}$ に比例して減少する。次のセルでは、先ほどと同じベクトル場のオイラー軌跡 1 本を $T = 1$ まで $N = 10$、100、1000 ステップでたどり、ラデマッハプローブを用いてそれぞれの方式を 200 回ずつ繰り返す（$\epsilon^\top \mathbf{J}$ には上で導いた陽な積の式を用いる）。

```
field = lambda x: W2 @ np.tanh(W1 @ x + b1) - x
diag_W1W2 = np.einsum('ij,ji->i', W1, W2)
for N in (10, 100, 1000):
    h, xs = 1 / N, [x]
    for _ in range(N - 1):                     # Euler trajectory x_0, ..., x_{N-1}
```

```
        xs.append(xs[-1] + h * field(xs[-1]))
    S = 1 - np.tanh(np.array(xs) @ W1.T + b1) ** 2  # s at every step, (N, n_hidden)
    exact = h * (S @ diag_W1W2).sum() - d      # h * sum_n tr J(x_n)
    held = np.broadcast_to(rng.choice([-1.0, 1.0], (200, 1, d)), (200, N, d))
    fresh = rng.choice([-1.0, 1.0], (200, N, d))
    row = f'N = {N:4d}: exact {exact:+.2f}'
    for name, P in (('held probe', held), ('fresh probes', fresh)):
        est = h * np.einsum('rnh,nh,rnh->r', P @ W2, S, P @ W1.T) - d  # eps^T eps = d
        row += f' | {name} {est.mean():+.2f}, sd {est.std():.2f}'
    print(row)
```

```
N =   10: exact -50.49 | held probe -50.69, sd 6.09 | fresh probes -50.39, sd 1.76
N =  100: exact -50.48 | held probe -51.03, sd 5.81 | fresh probes -50.47, sd 0.51
N = 1000: exact -50.48 | held probe -50.04, sd 5.52 | fresh probes -50.46, sd 0.19
```

厳密な積分値はやはり -50 に近く、どちらの方式の平均も標本誤差（sampling error）の範囲内でこれに一致する。つまり、いずれの方式でも推定は不偏である。ばらつきも予測どおりとなる。固定プローブの標準偏差はどの N でも約 5〜6 であるのに対し、新規プローブでは 1.5〜1.8（$N = 10$）、約 0.5〜0.6（$N = 100$）、そして 0.2 弱（$N = 1000$）となり、どの行でも固定プローブの値を $\sqrt{N}$ で割った大きさに一致している。

$N = 1000$ では新規プローブの方が約 30 倍もノイズが小さく、計算コストは同じである。それなのに、なぜわざわざプローブを固定するのだろうか。プローブを固定しておけば、ソルバが解くべき対象が常微分方程式のまま保たれるからである。ϵ を固定すれば推定値 $\epsilon^\top \mathbf{J}(\mathbf{x})\epsilon$ は状態の決定論的な関数になるため、どんなソルバでも適用でき、適応的ソルバで誤差を制御することもでき、随伴方程式から勾配も求まる。評価のたびに新しいプローブを引くと右辺にノイズが混入するため、2 回の評価を比較してステップ幅を判定するソルバは、2 回の乱数抽出を比較してしまうことになる。上のセルで新規プローブを使えたのは、オイラーステップを事前に固定していたからにすぎない。FFJORD はプローブを固定し、ノイズを受け入れる設計をとっている。

訓練においてはそのノイズがあっても問題にならない。訓練に必要なのは高精度な尤度ではなく、不偏な尤度だけだからである。トレース推定値は対数尤度（log-likelihood）に線形に入るため、推定された対数尤度とその勾配もやはり不偏となり、確率的勾配（stochastic gradient）にそれ以上の性質は必要ない（35.4 節）。プローブに由来するノイズは、ミニバッチ（minibatch）自体のノイズと同様に、ミニバッチ内および反復間にわたって平均化される。一方、1 個の固定プローブでは求められないのが、単一のデータ点に対する正確な尤度である。対数尤度における 5 の誤差は密度では $e^5 \approx 150$ 倍もの違いになるため、尤度を評価する際には多数のプローブで平均をとるか、固定したステップのグリッド上で新規プローブを引くか、あるいは厳密なトレースを計算する。

本節で扱ったノイズはプローブのなかにしか存在せず、軌跡そのものは決定論的であった。これに対し拡散モデルはステップごとに新しいノイズを状態そのものに加えるため、$\sqrt{h}$ のスケールで変化する増分によって本節が依拠してきたテイラー展開（Taylor expansion）が成り立たなくなる。そのような方程式を積分するには、独自の微分積分学が必要になる（38.4 節）。

38.3.4 まとめ

残差ブロックはオイラーステップであり、単一のベクトル場でステップ幅を限りなく小さくするとニューラル ODE が得られる。その厳密なフローは可逆であり、ステップ数はソルバに委ねられる。ただし、これら 2 つの利点には限界がある。$\mathbb{R}$ 上のフローは順序を保つため $x \mapsto -x$ には 2 つ目の次元が必要になり、10 ステップのオイラー法を通して訓練したベクトル場は 10 ステップのプログラムに適合するのに対し、RK4 を通して訓練したベクトル場は 4 倍のベクトル場評価と

引き換えにフローそのものに適合する。$\mathbf{x}(T)$ 上の損失の勾配は軌跡に沿った $(\partial \mathbf{f}/\partial \theta)^{\top}\mathbf{a}$ の積分で与えられ、随伴状態 $\mathbf{a}$ は $\nabla \ell(\mathbf{x}(T))$ から逆向きに線形 ODE を解くことで定まる。オイラー法を通した誤差逆伝播法はプログラムに対しては厳密であり、フローの勾配からは $O(h)$ 以内に収まる。状態を逆向きに再積分すればメモリを節約できるが、保存時の誤差が $e^{\lambda T}$ 倍に増幅される（減衰率 λ で減衰するモードの場合）。軌跡に沿って $\frac{d}{dt}\log p_t = -\operatorname{tr}\mathbf{J}$ が成り立ち、離散フローで行列式を要した箇所がトレースに置き換わる。Hutchinson 推定量はプローブ 1 個あたり 1 回のベクトル・ヤコビ行列積からトレースを統計的バイアスなしに求めるが、そこに乗るノイズは訓練では平均化されて消えるものの、単一の尤度評価では消えない。

38.3.5 演習問題

1. [code] **フローが表現できないもの。**●リプシッツ定数 L を持つベクトル場の 2 本の軌跡は $e^{-Lt}\|\mathbf{x}-\mathbf{y}\| \le \|\Phi_t(\mathbf{x})-\Phi_t(\mathbf{y})\| \le e^{Lt}\|\mathbf{x}-\mathbf{y}\|$ を満たし（演習問題 2、38.1 節）、38.3.1.2 節では $\mathbb{R}$ 上のいかなるフローも順序を逆転できないことを示した (Dupont et al. 2019)。
   1. $d=1$ とする。上の下界を用いて Φ_T が狭義単調増加であることの別証明を与え、$\Phi_T(1)-\Phi_T(-1) \ge 2e^{-LT}$ を示せ。順序と距離という 2 つの主張のうち、本文の命題から得られるのはどちらで、上の評価式を必要とするのはどちらか。
   2. 本文の訓練例は、円上の 2 点 $(0,\pm 1)$（距離 2）を楕円上の 2 点 $(1.5,\pm 0.4)$（距離 0.8）へと移す。時刻 T の厳密なフロー（ベクトル場のリプシッツ定数を L とする）がこれを実現できるのは $LT \ge \ln 2.5 \approx 0.92$ の場合に限られることを示せ。一方、長軸が 2 から 2.4 へ伸びることからは $LT \ge \ln 1.2$ しか得られないことも確認せよ。隠れ層 1 層の tanh ベクトル場は $L \le \|\mathbf{W}_2\|_2\|\mathbf{W}_1\|_2$ を満たす（tanh が 1-リプシッツ連続であるため）。RK4 で訓練したベクトル場についてこの上界を計算し、上の不等式を検証せよ。
   3. $\mathbb{R}$ を $\mathbb{R}^2$ に $x \mapsto (x,0)$ として埋め込む。任意の線形ベクトル場 $\dot{\mathbf{x}}=\mathbf{C}\mathbf{x}$ について、その時刻 1 の写像が $(x,0)$ を $(-x,0)$ へ（すべての x に対して）送るならば $e^{\mathbf{C}} = -\mathbf{I}$ が成り立つことを示せ。**ヒント：固有値 -1 が $e^{\mathbf{C}}$ に現れるには、ある固有値 μ が $\mathbf{C}$ に存在して $e^{\mu}=-1$ を満たす必要があるが、これは実数ではない。**さらに、軌跡が円ではなく楕円となるようなベクトル場を 1 つ求めよ。
   4. 訓練セルを変更し、写像 $x \mapsto -x$ を 64 個の等間隔な点（区間 $[-1,1]$）の上で、入出力次元 1 のベクトル場と $N=10$ 回のオイラーステップを用いて適合させよ。訓練後のモデルは何を出力するか。また、損失が訓練前に予測可能なある値に落ち着くのはなぜか。次元 2 のベクトル場を用い、入力を $(x,0)$ とし、第 1 座標に対して損失をとる設定でも同様の実験を行え。
   5. 1 次元モデルを $N=1$、$h=1$ で再度訓練し、それを $N=1$、および $N=100$ ステップ（サイズ $h=0.01$）で評価せよ。2 つの結果を本文の命題および条件 $hL<1$ と関連付けて説明せよ。

2. **累積損失を伴う随伴法。**●随伴方程式 Equation 38.17 と Equation 38.18 は終端点での損失を扱っている。これに対し、運動エネルギー $\int_0^T \|\mathbf{f}(\mathbf{x}(t),\theta,t)\|^2\,dt$（Finlay et al. (2020)）のような正則化項は、軌跡に沿ってコストを累積する。損失を次のように定める。

$$\mathcal{L} = \ell_T(\mathbf{x}(T)) + \int_0^T \ell(\mathbf{x}(t),\theta,t)\,dt.$$

   1. Equation 38.17 の変分的証明を拡張し、随伴方程式に湧き出し項が加わって $\dot{\mathbf{a}} = -(\partial \mathbf{f}/\partial \mathbf{x})^{\top}\mathbf{a} - \nabla_{\mathbf{x}}\ell$（終端条件 $\mathbf{a}(T)=\nabla \ell_T(\mathbf{x}(T))$）となること、および $\partial \mathcal{L}/\partial \theta = \int_0^T \left((\partial \mathbf{f}/\partial \theta)^{\top}\mathbf{a} + \nabla_{\theta}\ell\right)dt$ が成り立つことを示せ。

2. Finlay et al. (2020) はその代わりに、座標 E（$\dot{E} = \ell$、$E(0) = 0$）を状態に追加し、$E(T)$ を終端損失に加えている。Equation 38.17 を拡大状態 $(\mathbf{x}, E)$ に適用し、(a) の結果が再現されることを示せ。
3. 運動エネルギーでは $\ell = \|\mathbf{f}\|^2$ である。$\nabla_{\mathbf{x}} \ell$ をベクトル・ヤコビ行列積として表せ。このとき、随伴 ODE の 1 ステップには $\mathbf{f}$ に対するリバースモード計算が何回必要になるか。

3. [code] **軌跡の逆方向再構成。**●連続随伴では、$\mathbf{x}(t)$ を $\mathbf{x}(T)$ から $\mathbf{a}(t)$ とともに逆向きに積分することで、順方向の状態の保存を省くことができる。しかし 38.3.2 節で見たように、減衰率 λ で減衰するモードがあると、誤差 δ（保存された $\mathbf{x}(T)$ に含まれるもの）が $e^{\lambda T}\delta$ へと増幅される。

1. $\mathbf{R}$ を 0.6 ラジアンの回転行列とし、$\mathbf{C} = \mathbf{R}\,\mathrm{diag}(-50, -1)\,\mathbf{R}^\top$ とおく。$\dot{\mathbf{x}} = \mathbf{C}\mathbf{x}$ を初期値 $(1, 1)$ から時刻 T まで 2000 ステップの RK4（`solve` と `rk4_step`）で積分し、続いて得られた結果から負の終端時刻を与えて同じ時間だけ逆向きに積分せよ。$T = 0.25$、0.5、1、2 に対する再構成誤差（reconstruction error）を求め、その増大を $e^{50T}\delta$ と比較せよ。どの程度の δ が当てはまるか。また、その値が 10^{-17} に近く、本文で述べた 10^{-16} より小さいのはなぜか。**ヒント：$\mathbf{x}(T)$ の各座標の大きさはどれほどか。**
2. $\mathrm{diag}(-50, -1)$ そのものを用いて同じ実験を行え。$T = 0.5$ 以降、この行列では $\mathbf{C}$ の場合よりはるかに正確に初期値が再構成される。それはなぜか。
3. 活性化チェックポイント（5.7.3.3 節）は軌跡の各区間の始点で状態を保存する。一方、ハイブリッド方式では各区間の終点のチェックポイントから逆向きに積分して区間内の状態を再構成する。(a) の行列 $\mathbf{C}$ に対し、ドリフト $e^{50T}\delta$（$\delta = 10^{-16}$）が 10^{-8} 未満に収まる最大の区間長はいくらか。

4. **残差ステップの対数行列式。**●●瞬間的な変数変換 Equation 38.21 は無限小ステップの極限である。有限幅の残差ステップ $F(\mathbf{x}) = \mathbf{x} + \mathbf{g}(\mathbf{x})$ では、Behrmann et al. (2019) の可逆残差ネットワークのように厳密な対数行列式が必要になる。

1. $\mathbf{g}$ を定数 $c < 1$ のリプシッツ連続写像とする。$d = 1$ のときは 38.3.1.2 節の 1 行の議論（オイラーステップは $hL < 1$ のとき単調増加）から F が狭義単調増加、ゆえに単射であることがわかる。一般の次元 d に対し、$F(\mathbf{x}) = \mathbf{y}$ を $\mathbf{x}$ に関する不動点（fixed point）方程式として書き換え、ピカール・リンデレーフの定理の証明と同様にバナッハの不動点定理を用いて、F が $\mathbb{R}^d$ 上の全単射であることを示せ。この不動点反復はどのくらいの速さで収束するか。

2. $\|\mathbf{J}_{\mathbf{g}}\| \le c$（ただし $\mathbf{J}_{\mathbf{g}}$ は $\mathbf{g}$ のヤコビ行列）を示し、$\mathbf{J}_{\mathbf{g}}$ の固有値を用いて次式を示せ。

$$\log\det(\mathbf{I} + \mathbf{J}_{\mathbf{g}}) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}\,\mathrm{tr}\,(\mathbf{J}_{\mathbf{g}}^{k}).$$

3. オイラーステップでは $\mathbf{g} = h\mathbf{f}$ かつ $\mathbf{J}_{\mathbf{g}} = h\mathbf{J}$ である。$\log\det(\mathbf{I} + h\mathbf{J})$ を、1 ステップあたりに項 $h\,\mathrm{tr}\,\mathbf{J}$ を累積する Equation 38.22 と比較せよ。1 ステップあたりの差はどの程度で、T/h ステップ全体での合計の差はどの程度になるか。

4. Hutchinson 推定量 Equation 38.24 は $\mathrm{tr}(\mathbf{J}_{\mathbf{g}}^{k})$ を $\epsilon^\top \mathbf{J}_{\mathbf{g}}^{k} \epsilon$ によって推定する。級数を第 K 項で打ち切るとき、プローブ 1 個あたり何回のベクトル・ヤコビ行列積が必要になるか。

5. [code] **Hutchinson 推定量の分散。**●推定量 Equation 38.24 は $\mathrm{tr}\,\mathbf{M}$ を $\epsilon^\top \mathbf{M} \epsilon$（単一のプローブ ϵ）によって推定する。$\mathbf{S} = (\mathbf{M} + \mathbf{M}^\top)/2$ とおけば $\epsilon^\top \mathbf{M} \epsilon = \epsilon^\top \mathbf{S} \epsilon$ が成り立つ。

1. $\epsilon^\top \mathbf{M} \epsilon$ の分散が、ガウスプローブでは $2\|\mathbf{S}\|_F^2$、ラデマッハプローブでは $2\sum_{i \neq j} S_{ij}^2$ となることを示せ。単一のラデマッハプローブが厳密な値を与えるのはどのような行列か。
2. 引き戻し項を持つベクトル場のヤコビ行列 $\mathbf{J} = \mathbf{W}_2 \,\mathrm{diag}(\mathbf{s})\, \mathbf{W}_1 - \mathbf{I}$（サイズは 50×50）を、Hutchinson の実験の重みと評価点から陽に構成せよ。両方の分散を計算し、その平方根を実験で出力されたプローブ 1 個あたりの 2 つの標準偏差と比較せよ。標準誤差を 0.01 にするには、それぞれの分布で何個のプローブが必要か。
3. CNF において、$\mathbf{M}$ はヤコビ行列 $\partial \mathbf{f}/\partial \mathbf{x}$（次元 d）である。すべての成分が同程度の大きさを持つと仮定する。プローブ 1 個の相対標準誤差 $\sqrt{\mathrm{Var}}/|\,\mathrm{tr}\,\mathbf{M}|$ は、対角成分がすべて同符号のときと符号がランダムなときで、それぞれ d に対してどのようにスケールするか。
4. 訓練時、ミニバッチ内の B 個の各点はそれぞれ独自の軌跡と独自の固定プローブを持つ。平均対数尤度のノイズは B にどう依存するか。また、どの B のときに、1 つの点が新規プローブ（$N = 100$ ステップ）で持つノイズと一致するか。

6. [code] **離散随伴と連続随伴。●●**ソルバを通した誤差逆伝播法は実行されたプログラムを微分するのに対し、随伴 ODE Equation 38.17 にソルバを適用したものは厳密なフローの勾配を近似する (Kidger 2022)。ヤコビ行列 $\mathbf{J}(\mathbf{x}) = \partial \mathbf{f}/\partial \mathbf{x}$ と順方向状態 $\mathbf{x}_n$ を持つ自律的なベクトル場について、2 つの漸化式を 1 ステップずつ比較せよ。

   1. 前進オイラー法に対し、離散随伴 Equation 38.19 は $\mathbf{a}_n = (\mathbf{I} + h\mathbf{J}(\mathbf{x}_n))^\top \mathbf{a}_{n+1}$ を計算する一方、随伴 ODE を t_{n+1} から t_n へ戻す陽的オイラーステップは $\mathbf{a}_n = \mathbf{a}_{n+1} + h\mathbf{J}(\mathbf{x}_{n+1})^\top \mathbf{a}_{n+1}$ を計算する。滑らかな $\mathbf{f}$ に対してこの 2 つの更新の差を上から評価せよ。線形なベクトル場では両者は一致し、そのときフローの勾配との差はオイラー法自身の誤差になると本文で述べた。スカラーのプログラム $x_N = (1 + h\theta)^N x_0$ についてこれを厳密に確かめよ。すなわち、上の漸化式が本文で求めた勾配 $T\, x_N^2/(1 + h\theta)$ を計算すること、およびこれが $T\, x(T)^2(1 - h\theta(1 + \theta T)) + O(h^2)$ に等しいことを示せ。
   2. ホイン法の 1 ステップ $\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}\big[\mathbf{f}(\mathbf{x}_n) + \mathbf{f}(\mathbf{u}_n)\big]$（ただし $\mathbf{u}_n = \mathbf{x}_n + h\mathbf{f}(\mathbf{x}_n)$）を微分して、その離散随伴の漸化式を導け。この漸化式が、随伴 ODE を t_{n+1} から t_n へ進めるホイン法のステップにおいて $\mathbf{J}(\mathbf{x}_{n+1})$ を $\mathbf{J}(\mathbf{u}_n)$ に置き換えたものであることを示せ。2 つの更新は h の何次のオーダーで異なるか。
   3. 線形なベクトル場 $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ に対し、RK4 の 1 ステップは状態に $R(h\mathbf{A})$ を掛ける（ただし $R(z) = 1 + z + z^2/2 + z^3/6 + z^4/24$ は RK4 の安定性関数である。38.2 節、およびその虚軸上の振る舞いを扱う同節の演習問題 2 を参照）。RK4 の離散随伴が随伴 ODE の RK4 ステップに一致することを示せ。
   4. `loss_and_grad_unrolled` のオイラーステップを RK4 ステップに置き換え、手書きの逆方向スイープの代わりに中心有限差分で展開損失の勾配を計算せよ。連続随伴の勾配 `g_cont` との差を $N = 5$、10、20、40、80 ステップについて測定せよ。なお、`g_cont` を計算する台形公式は `n_quad = 2000` 区間を用いており約 10^{-7} の精度しかないため、先にこれを細かくしておくこと。どのような傾きが観測されるか。次数 p のソルバを通して計算された勾配は何次の精度を受け継ぐか。

7. [code] **リウヴィルの公式。●●**瞬間的な変数変換では軌跡に沿って密度を追跡した。本問では、フロー写像自身のヤコビ行列 $\mathbf{M}(t) = D\Phi_t(\mathbf{x}_0)$ を通じた別証明を与える。

   1. $\frac{d}{dt}\Phi_t(\mathbf{x}_0) = \mathbf{f}(\Phi_t(\mathbf{x}_0), t)$ を $\mathbf{x}_0$ で微分し、変分方程式 $\dot{\mathbf{M}} = \mathbf{J}(\mathbf{x}(t), t)\,\mathbf{M}$（初期条件 $\mathbf{M}(0) = \mathbf{I}$）を導け。
   2. ヤコビの公式 $\frac{d}{dt}\det \mathbf{M} = \mathrm{tr}(\mathbf{M}^{-1}\dot{\mathbf{M}})\det \mathbf{M}$ を可逆な $\mathbf{M}(t)$ に対して証明せよ。**ヒント：$\det \mathbf{M}(t + h) = \det \mathbf{M}(t)\,\det\big(\mathbf{I} + h\mathbf{M}^{-1}\dot{\mathbf{M}} + O(h^2)\big)$ と書き換え、本文の補題を用いる。**

3. $\log\left|\det D\Phi_T(\mathbf{x}_0)\right| = \int_0^T \operatorname{tr}\mathbf{J}(\mathbf{x}(t),t)\,dt$ を結論付け、Equation 38.22 を離散の変数変換 Equation 38.20（全単射 Φ_T に適用したもの）から導け。線形なベクトル場に対して、この結果は公式 Equation 38.23 に一致する。
4. $\dot{\mathbf{M}} = \mathbf{J}\mathbf{M}$ を 8 ユニットの tanh ベクトル場（38.3.3 節）と同時に RK4 で積分せよ（Hutchinson の実験で重みの変数名が再利用される前の、トレース積分を検証するセルで行うこと）。得られた $\log\left|\det\mathbf{M}(1)\right|$ を、そのセルが出力するトレース積分および対数行列式と比較せよ。

38.4 確率微分方程式

これまでの節で扱った軌跡はすべて始点によって定まり、フローはどこから出発したかを決して忘れない。すなわち、2 つの異なる始点は必ず 2 つの異なる終点に到達する。第 17 章の拡散モデル（diffusion model）は、始点を忘れる過程の上に構築されている。その**順過程**（forward process）は、始点の痕跡がまったく残らなくなるまで状態に新しいノイズを少しずつ混ぜていき、それを逆向きにたどる生成では既知の出発点から始められるようになる。このような過程も微分方程式を満たすが、通常の微分積分では扱えない方程式である。

まずは手計算で追えるほど小さな例から始めよう。1 つの数値 $x_0 = 2$ をとる。各ステップでその値の 99% を残し、標準偏差 $\sqrt{0.02} \approx 0.14$ の独立なガウス乱数を加える。

$$x_{k+1} = 0.99\,x_k + \sqrt{0.02}\,\xi_k, \qquad \xi_k \sim \mathcal{N}(0,1).$$

400 ステップ後には平均が $2 \cdot 0.99^{400} \approx 0.036$ となり、分散は 1 に近づく。ここでは 1 ステップあたりに削られる 2% と、新たに加わる 0.02 とが釣り合っている。他のどの始点から出発しても同じ場所に行き着き、ほぼ $\mathcal{N}(0,1)$ からのサンプルになる。この漸化式には一見奇妙な特徴がある。これがタイムステップ $h = 0.01$ で連続時間（continuous time）の過程を記述しているとするなら、減衰 $0.99 = 1 - h$ は x をオーダー h の量だけ変化させるのに対し、ノイズの大きさは $\sqrt{h}$ であって h ではない。この選択は必然である。大きさ h のノイズは $h \to 0$ とともに消えてしまうが、大きさ $\sqrt{h}$ のノイズは生き残り、その二乗はドリフトと同じオーダー h になる。通常の微分積分は微小変化の二乗を切り捨てるため、このたった 1 つの事実が新しい解析学を必要とするのである。

このようなノイズの極限がブラウン運動（Brownian motion）であり、1827 年に花粉から出た微粒子が水中で小刻みに揺れ動く様子を観察した植物学者 Robert Brown にちなんで名付けられた。Norbert Wiener は 1923 年にこれを数学的対象として定式化し、Kiyoshi Itô は 1940 年代に、たった 1 つの規則の上にその解析学を築き上げた (Itô 1944)。すなわち、短いブラウン増分の二乗を足し合わせると経過時間に一致するため、連鎖律（chain rule）に二階導関数（second derivative）の項が加わるという規則である。

本節では、コイントスからブラウン運動を構成し、上の規則からその解析学を構築する。続いて、冒頭の漸化式の背後にある過程、すなわち 17.2 節の順過程を解く。始点が与えられたとき、この過程の分布はあらゆる時刻でガウス分布になるため、拡散モデルは 1 回のサンプリングでデータを任意のノイズレベルまで汚すことができる。シミュレーションとその誤差の測定は、再読時に回してもよい。にわかには信じがたいのは二乗に関する規則だけであり、その証明は 6 行で済む。

38.4.1 ブラウン運動

38.4.1.1 順過程としてのノイズ

数学的な道具立てに入る前に、順過程が果たすべき役割を確認しておこう。$p_0 = p_{\text{data}}$ に従うデータから出発し、元のデータがどのような姿であっても、1 つの既知の分布に到達しなければならない。

$$p_0 = p_{\text{data}} \quad \longrightarrow \quad p_T \approx \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

そうすれば、生成は常にどこから出発すべきかがわかる。いかなる固定されたフローも、2 つのデータセットに対して同時にこれを達成することはできない。フローは可逆写像 Φ（38.1 節）であるため、もし 2 つの分布 p と q を同じ分布へ移すなら、Φ^{-1} を適用することで $p = q$ が導かれてしまうからである。

考えられる限り最も単純な 2 つの「データセット」、すなわち $x_0 = 2$ と $x_0 = -2$ における点質量に対して、3 つの過程を試してみよう。

- **ドリフトのみ**（$\dot{x} = -x$）。解 $x(t) = x_0 e^{-t}$ により、2 つの始点は $\pm 2e^{-3} = \pm 0.0996$（$t = 3$ のとき）に移る。両者はガウス分布ではなく点 0 に近づき、いかなる有限時刻においても依然として 2 つの異なる点のままである。
- **ノイズのみ**（$dX = \sqrt{2}\, dW$）。これは $X_t = x_0 + \sqrt{2}\, W_t$ を意味する。ここで W は時間とともに分散が線形に増大するノイズである（次項で構成する）。X_t の分布、すなわちその**法則**（law）は $\mathcal{N}(x_0, 2t)$ である。各点群の中心は ± 2 にとどまったまま、標準偏差 $\sqrt{2t}$ が際限なく増大する（値は 2.45、$t = 3$ のとき）。始点は広がり続ける幅との相対的な意味でしか忘れられず、極限の法則は存在しない。
- **両者の組み合わせ**（$dX = -X\, dt + \sqrt{2}\, dW$）。どちらの始点からでも $t = 3$ における法則は $\mathcal{N}(\pm 0.0996,\, 0.9975)$ になることが、38.4.3 節で証明される。2 つの法則は互いに、そして $\mathcal{N}(0, 1)$ ともほとんど区別がつかない。

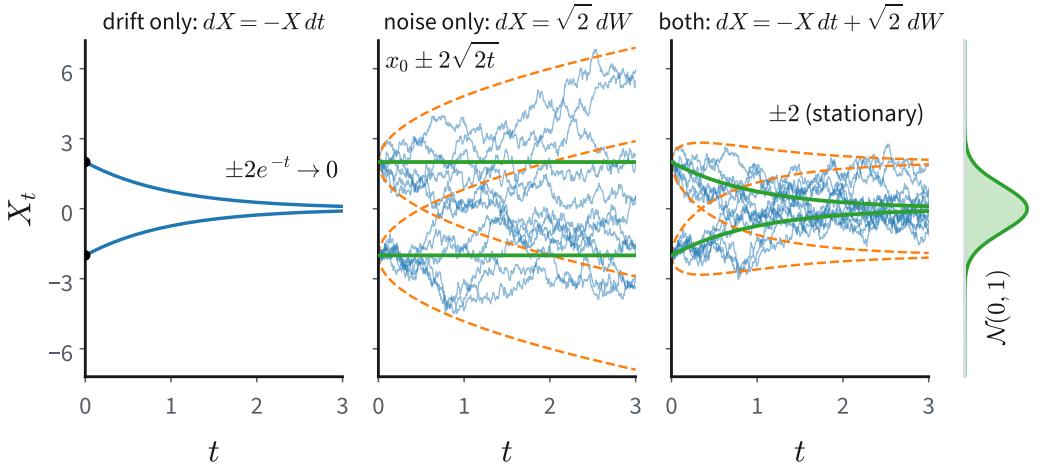


Figure 38.11: 2 つの始点 $x_0 = \pm 2$ から出発する、1 つの失敗例、1 つの利用可能な過程、そして本書で用いる過程。ドリフトのみではあらゆる始点が点 0 へ送られる。ノイズのみでは各始点が点群の中心に保たれたまま、標準偏差の 2 倍の包絡線 $x_0 \pm 2\sqrt{2t}$（破線）が際限なく広がる。両者を組み合わせると、あらゆる始点が同じ極限法則 $\mathcal{N}(0, 1)$（右端に横向きに描画）へと運ばれる。緑色の線は平均を表す。

Figure 38.11 にこれら 3 つを描く。完全に失敗するのは 1 つ目だけである。2 つ目は利用可能であり、実際に使われている。その分散 $2t$ がデータの分散よりはるかに大きくなれば、始点はほとんど影響しなくなる。この種の過程は極限法則を持たないため終端の分散を巨大にしなければならず、**分散爆発**（variance-exploding）と呼ばれる。3 つ目が本節を通じて用いる通し例であり、冒頭の漸化式はこれをシミュレーションしたものである。ドリフトが各始点をゼロへ引き寄せる一方、ドリフトが奪った分散をノイズが補填し、両者は分散 1 で釣り合う。

一般に、この種の順過程は決定論的なベクトル場にノイズ項を加えたものである。$\mathbb{R}^d$ 内のデータに対して、次のように書く。

$$d\mathbf{X} \;=\; \underbrace{\mathbf{f}(\mathbf{X}, t)\, dt}_{\text{drift}} \;+\; \underbrace{g(t)\, d\mathbf{W}}_{\text{diffusion}}, \tag{38.25}$$

この式は、微小時間 dt の間に、状態が速度 $\mathbf{f}$ と dt の積だけ移動し、ノイズ振幅 g で大きさが決まるランダムな揺らぎ（キック）を受けると読む。ドリフトが平均を操り、拡散（diffusion）が分散を注入する。これまでのところ、揺らぎ $d\mathbf{W}$ は単なる記号にすぎない。まずは 1 つの座標でこれに意味を与え、伊藤解析（Itô calculus）の最後に改めて $\mathbb{R}^d$ へ戻ることにする。

38.4.1.2 ランダムウォークからウィーナー過程へ

最も単純な確率過程（random process）である、コイントスで駆動されるランダムウォークの極限をとることで、この記号の背後に実体を与えよう。その極限から、冒頭の漸化式に平方根が現れた理由も明らかになる。

ステップ幅 Δt を固定し、公平なコインで符号を選びながら大きさ $\pm\sqrt{\Delta t}$ の独立なステップを踏むランダムウォークを考える。k ステップ後の時刻 $t = k\,\Delta t$ における位置は、k 個の独立な平均ゼロのステップ（それぞれの分散は Δt）の和である。独立な項の分散は加法性を持つため（36.2.2.1 節）、次が成り立つ。

$$\mathbb{E}\big[W_t^{(\Delta t)}\big] = 0, \qquad \mathrm{Var}\big(W_t^{(\Delta t)}\big) = k\,\Delta t = t,$$

これはステップサイズによらず成り立つ。たとえば $\Delta t = 0.01$ とすると、100 回の ± 0.1 のステップによって、分散は $100 \cdot 0.01 = 1$（時刻 1 時点）となる。同じ時間刻みで ± 0.01 のステップにすれば分散は 0.01 になり、± 1 のステップにすれば 100 になる。一般に、大きさ $c\,\Delta t^{a}$ のステップは分散 $c^2\, t\, \Delta t^{2a-1}$（時刻 t 時点）を与え、これは 0 へ収束するか（$a > \tfrac{1}{2}$ のとき）、あるいは ∞ へ発散する（$a < \tfrac{1}{2}$ のとき）。極限において有限かつ非ゼロの分散を残すのは平方根だけであり、後のすべての数値スキームのノイズ項に平方根が再び現れる理由はここにある。

$\Delta t \to 0$ とすると時刻 t までのステップ数は際限なく増え、中心極限定理（central limit theorem、36.3.2.3 節）によって位置はガウス分布に従うようになる。すなわち、全分散が t である多数の独立な平均ゼロの項の和は、近似的に $\mathcal{N}(0, t)$ となる。

この極限の過程は**ブラウン運動**または**ウィーナー過程**（Wiener process）と呼ばれる (Wiener 1923)。これは次の 4 つの性質によって定義される。

1. $W_0 = 0$;
2. **独立増分**（independent increments）：$s < t$ に対し、増分 $W_t - W_s$ は時刻 s までの経路全体と独立である。
3. **ガウス増分**（Gaussian increments）：$W_t - W_s \sim \mathcal{N}(0,\, t - s)$ が成り立つ。
4. 見本路（サンプルパス）が連続である。

これらの性質を持つ過程が存在することはウィーナーの定理（Wiener's theorem）であり、リスケールされたランダムウォークがそこへ収束することはドンスカーの定理（Donsker's theorem）である。本書では両者を既知のものとして認める。この過程は標準偏差 $\sqrt{t}$ でどこまでも広がり続け（Figure 38.12）、その増分は次の形をとる。

$$\Delta W \;=\; W_{t+\Delta t} - W_t \;=\; \sqrt{\Delta t}\,\xi, \qquad \xi \sim \mathcal{N}(0,1), \tag{38.26}$$

すなわち、標準正規乱数を $\sqrt{\Delta t}$ 倍したものである。

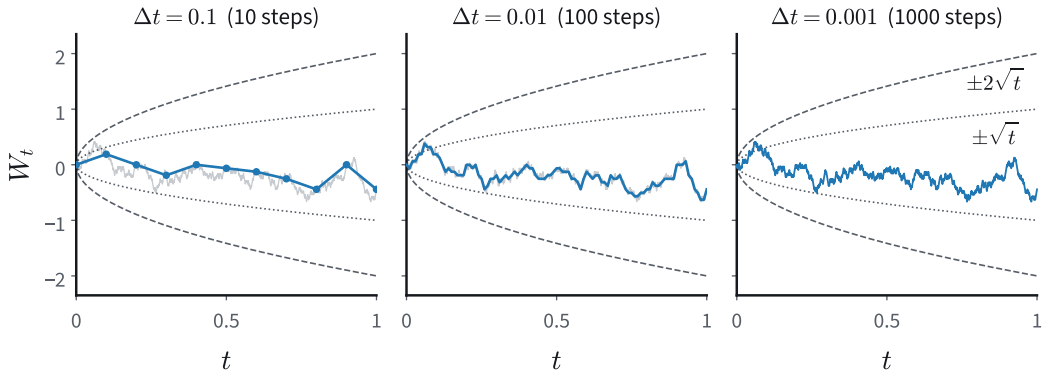


Figure 38.12: 1000 ステップ（ステップ幅 $\pm\sqrt{0.001}$）からなる 1 本のコイントス型ランダムウォークを、刻み幅 $\Delta t = 0.1, 0.01, 0.001$ のメッシュで観察した様子（薄い灰色は経路全体）。どの解像度で見てもステップ分散 Δt のランダムウォークになっており、メッシュを細かくすると大局的な経路を変えずに細部が埋められていく。各固定時刻における広がりは $\pm\sqrt{t}$ および $\pm 2\sqrt{t}$ の包絡線に従う。

独立増分の性質から、2 つの時刻における W の値同士の関係も決まる。$s \le t$ に対して $W_t = W_s + (W_t - W_s)$ と書く。増分は W_s と独立かつ平均ゼロであるため、$\mathrm{Cov}(W_s, W_t) = \mathbb{E}[W_s^2] = s = \min(s,t)$ となる。すなわち、このウォークは時刻 s における位置をそのまま先へ運び、それと無相関なノイズを加える。この公式には、1 行で終わる証明以上の価値がある。同時ガウス分布に従う値は平均と共分散によって定まるため、任意の有限個の時刻における値が同時ガウス分布に従い、平均ゼロ、共分散 $\min(s,t)$、かつ連続な経路を持つ過程はブラウン運動である。

同じ増分の性質が経路を粗く（ギザギザに）する。ブラウン経路は連続であるがいたるところ微分不可能であり、これは Paley、Wiener、Zygmund の定理である (Paley et al. 1933)。増分のスケーリング Equation 38.26 から、その直観的な理由がわかる。

$$\left|\frac{\Delta W}{\Delta t}\right| = \frac{|\xi|}{\sqrt{\Delta t}} \;\longrightarrow\; \infty \quad \text{in probability as } \Delta t \to 0,$$

したがって、ステップ幅が縮むにつれて差分商は際限なく大きくなる。これは厳密な証明ではないが（各固定時刻における導関数の存在を除外しているだけであり、「いたるところ」と言うにはすべての時刻を同時に扱う必要がある）、何が障壁になっているかは示している。

時間を c 倍、空間を $\sqrt{c}$ 倍にリスケールしても再びブラウン運動になるため（演習問題 2）、微分可能な曲線であれば拡大すると接線へと平坦化されるのに対し、ブラウン経路は拡大してもあら

ゆるスケールで同じ粗さを見せる。ゆえに dW/dt は存在せず、dW には積分の内部で意味を与えなければならない。その意味を与えるのが伊藤解析である。

38.4.1.3 ブラウン経路のシミュレーション

増分の表現 Equation 38.26 はそのままシミュレーションの手順にもなる。すなわち、$\sqrt{\Delta t}\,\xi$ のステップの累積和をとればよい。20,000 本の経路を区間 $[0,1]$ 上で 500 ステップにより生成し、1 つの問いを投げかけてみよう。任意の単一時刻では $W_t \sim \mathcal{N}(0,t)$ であるため、経路の 95.4% が $\pm 2\sqrt{t}$（Figure 38.12 の外側の包絡線）の内側に入る。では、すべての時刻にわたって包絡線の内側にとどまり続ける経路はどのくらいの割合だろうか。セルを実行する前に予想してみてほしい。

```
rng = np.random.default_rng()
T_bm, n_bm, n_paths = 1.0, 500, 20000
dt_bm = T_bm / n_bm
dW_bm = np.sqrt(dt_bm) * rng.standard_normal((n_paths, n_bm))
W_bm = np.cumsum(dW_bm, axis=1)              # W at t = dt, 2 dt, ..., 1
band = 2 * np.sqrt(dt_bm * np.arange(1, n_bm + 1))  # 2 sqrt(t) on the grid
inside = np.abs(W_bm) < band
print(f'inside +-2 sqrt(t) at t = 1:   {inside[:, -1].mean():.3f}'
      '   (Gaussian: 0.954)')
for stride in (10, 1):                       # same paths, coarse then fine grid
    print(f'inside at all {n_bm // stride} grid times: '
          f'{inside[:, stride - 1::stride].all(axis=1).mean():.3f}')
```

```
inside +-2 sqrt(t) at t = 1:   0.955   (Gaussian: 0.954)
inside at all 50 grid times: 0.701
inside at all 500 grid times: 0.545
```

単一の時刻 $t=1$ における割合は、当然ながら約 95% である。しかし、等間隔な 50 個の時刻すべてで内側にとどまる経路は 10 本中約 7 本しかなく、500 個すべての時刻となると半分強にすぎない。各時刻で成り立つ帯域が、経路全体に対して成り立つわけではない。観測する時刻を増やすたびに、経路が外へ出ているのを見つける機会が増えるからである。あらゆる瞬間で監視すれば、内側にとどまる経路は 1 本もなくなる。$t=0$ の近傍でブラウン経路は $\pm 2\sqrt{t}$ を何度も横切るからである（これは重複対数の法則（law of the iterated logarithm）と呼ばれるが、本書では証明しない）。ある 1 つの時刻における法則についての主張は、経路全体についての主張よりはるかに弱く、この 2 つは後にシミュレーションの弱誤差と強誤差として再び現れる（38.4.4.2 節）。

38.4.2 伊藤解析

38.4.2.1 二次変分

通常の微分積分でテイラー展開を 1 次で打ち切るのは、微小変化の二乗が無視できるほど小さいからである。ブラウン運動ではこれが成り立たず、どれほど成り立たないのかを示すのが本項の結果である。本節の残りの議論はすべてこの結果の上に成り立っている。

分割 $0 = t_0 < t_1 < \cdots < t_n = t$ をとり、区間幅を $\Delta t_i = t_{i+1} - t_i$、メッシュ幅を $h_{\max} = \max_i \Delta t_i$ として、この分割に沿った経路の増分の二乗を足し合わせる。微分可能な経路 x が $|x'| \le M$ を満たすとき、平均値の定理（mean value theorem）より $|\Delta x_i| \le M\,\Delta t_i$ となるため、次が成り立つ。

$$\sum_{i} (\Delta x_i)^2 \;\le\; M^2 \sum_{i} \Delta t_i^2 \;\le\; M^2\, h_{\max}\, t \;\longrightarrow\; 0$$

すなわち、メッシュ幅が縮むにつれてゼロへ収束する。滑らかな経路の増分の二乗はオーダー Δt^2 だからである。一方、ブラウン増分の二乗はオーダー Δt である。なぜなら $\mathbb{E}[(\Delta W_i)^2] = \Delta t_i$ であり、メッシュをどれほど細かくしてもこれらの和は t になるからである（Figure 38.13）。この和自体は確率変数だが、その揺らぎは極限において消失する。

命題（ブラウン運動の二次変分） メッシュ幅 $h_{\max} \to 0$ のとき、次が成り立つ。

$$Q_n \;=\; \sum_{i=0}^{n-1} \big(W_{t_{i+1}} - W_{t_i}\big)^2 \;\longrightarrow\; t \quad \text{in mean square:} \quad \mathbb{E}\big[(Q_n - t)^2\big] \to 0. \tag{38.27}$$

証明 各増分は $\mathcal{N}(0, \Delta t_i)$ に従うため、任意の分割に対して $\mathbb{E}[(\Delta W_i)^2] = \Delta t_i$ および $\mathbb{E}[Q_n] = \sum_i \Delta t_i = t$ が成り立つ。ばらつきについては、ガウス分布の 4 次モーメントが $3\,\Delta t_i^2$ であるため $\operatorname{Var}\big((\Delta W_i)^2\big) = 3\Delta t_i^2 - \Delta t_i^2 = 2\Delta t_i^2$ となり、各増分は独立であるため分散は足し合わされる。

$$\mathbb{E}\big[(Q_n - t)^2\big] = \operatorname{Var}(Q_n) = \sum_i 2\,\Delta t_i^2 \;\le\; 2\,h_{\max} \sum_i \Delta t_i = 2\,h_{\max}\,t \;\longrightarrow\; 0,$$

ゆえに平均二乗収束（mean-square convergence）が従う。$\blacksquare$

この主張は改めて注目に値する。個々の増分はすべてランダムであるにもかかわらず、極限ではそれらの二乗和からランダム性が消えるのである。n 個の区間に等分割した $[0,1]$ の一様グリッド上では、上の証明よりこの和の標準偏差は $\sqrt{2/n}$ となり、その値は 0.35（$n = 16$ のとき）および 0.003（$n = 2^{18}$ のとき）である。

この命題は通常、微分の積に関する実用規則である**伊藤の乗積表**（Itô multiplication table）として書かれる。

$$(dW)^2 = dt, \qquad dW\,dt = 0, \qquad (dt)^2 = 0. \tag{38.28}$$

1 つ目の規則は Equation 38.27 の略記であり、あくまで和に関する主張である。単一の増分の二乗は $\Delta t\,\xi^2$ であり、これが Δt に近い値をとることはめったにない。4 回に約 1 回は Δt の 10 分の 1 を下回り、6 回に約 1 回は Δt の 2 倍を上回る。証明が示したとおり、多数のステップにわたる和をとって初めて値が落ち着くのである。（コイントス型ウォークでは二乗したステップは毎回ちょうど Δt に等しいため、平均化を待つ必要がない。）残りの 2 つの規則は、$dW\,dt$（大きさ $\Delta t^{3/2}$）と $(dt)^2$ がいずれも Δt より速くゼロへ収束し、極限では何も寄与しないことを表している。

細かいメッシュでは揺らぎがきわめて小さくなるため、たった 1 本の経路から極限を確認できる。2^{18} 個の区間に分割した $[0,1]$ 上で 1 本の経路をサンプリングし、より粗い部分グリッド上で観察して、その二乗和を予測される標準偏差と並べて出力しよう。比較のため、滑らかな経路 $x(t) = \sin 2\pi t$ に対する同じ二乗和を $2\pi^2/n$（メッシュ幅 $1/n$ と $\int_0^1 x'(t)^2\,dt = 2\pi^2$ の積）と並べて出力する。

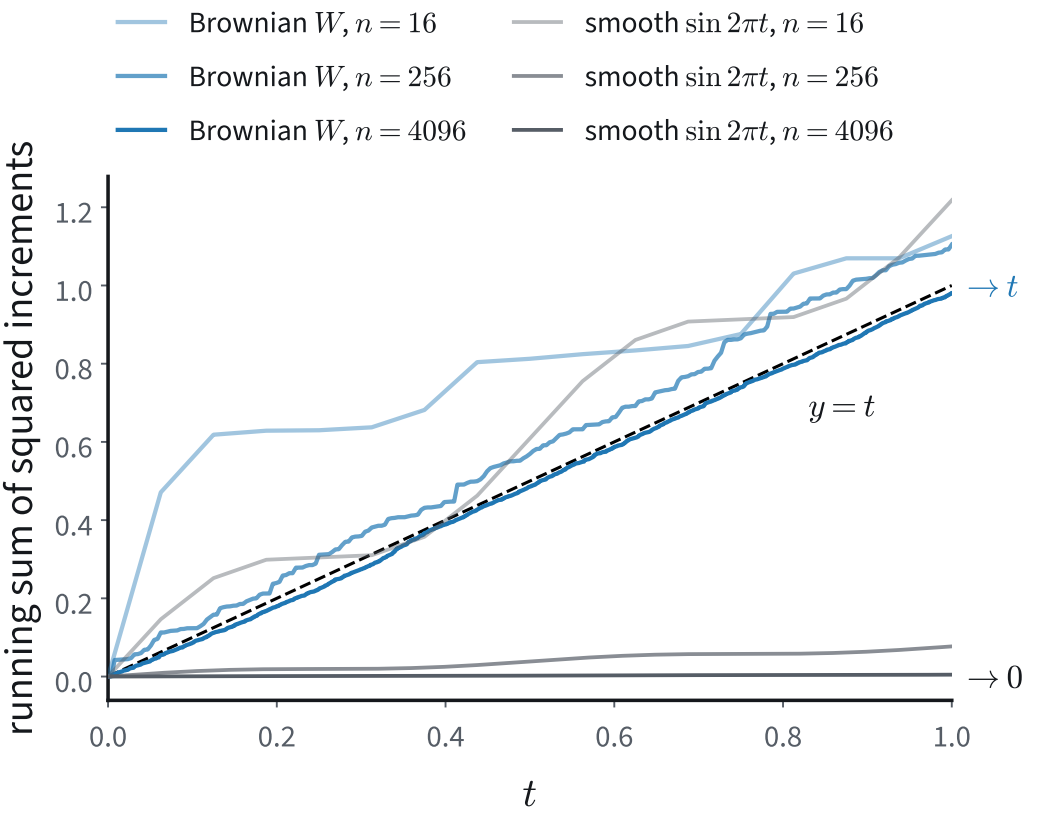


Figure 38.13: 1 本のブラウン経路に沿った増分の二乗の累積和（区間数 $n = 16$、256、4096 のメッシュ、色が濃いほど細かいメッシュを表す）。累積和は $y = t$ に近づく一方、滑らかな経路 $\sin 2\pi t$（灰色）に対する和は $2\pi^2/n$ のように縮小する。

```
n_fine = 2**18
dW_qv = np.sqrt(1.0 / n_fine) * rng.standard_normal(n_fine)
W_qv = np.concatenate([[0.0], np.cumsum(dW_qv)])
print(f'{"n":>7} {"QV of W":>8} {"sqrt(2/n)":>10} {"QV of sin":>10}'
      f' {"2 pi^2/n":>9}')
for k in range(4, 19, 2):
    n = 2**k
    W_n = W_qv[::n_fine // n]                # the same path on a coarser grid
    s_n = np.sin(2 * np.pi * np.linspace(0, 1, n + 1))
    print(f'{n:7d} {np.sum(np.diff(W_n)**2):8.4f} {np.sqrt(2 / n):10.4f} '
          f'{np.sum(np.diff(s_n)**2):10.6f} {2 * np.pi**2 / n:9.6f}')
```

```
      n  QV of W  sqrt(2/n)  QV of sin  2 pi^2/n
     16    1.5001     0.3536   1.217927  1.233701
     64    1.0363     0.1768   0.308177  0.308425
    256    0.9313     0.0884   0.077102  0.077106
   1024    0.9993     0.0442   0.019277  0.019277
   4096    1.0409     0.0221   0.004819  0.004819
  16384    1.0148     0.0110   0.001205  0.001205
  65536    1.0038     0.0055   0.000301  0.000301
 262144    1.0022     0.0028   0.000075  0.000075
```

ブラウン運動の列は 1 のまわりに、証明が予測する標準偏差 $\sqrt{2/n}$ でばらつく。16 個の区間ではおおむね約 0.5 から 1.6 の間に収まり、$n = 2^{18}$ では約 0.01 以内の差で 1 に近づく。滑らかな経路の列はまだ 1.22 であるが（$n = 16$ のとき）、$2\pi^2/n$ に従って縮小し、3 桁までその予測と一致するようになる（$n = 256$ 以降）。

38.4.2.2 伊藤積分

経路が粗いため、dW に関する積分を直接定義せざるを得ない。最も単純な $\int_0^t W\,dW$ から始めると、通常のリーマン和（Riemann sum）による定義では答えが 1 つに定まらないことがわかる。

積分の基準となる関数が滑らかであれば、被積分関数を各区間の左端点、右端点、中点のいずれで評価しても、リーマン和はすべて同じ極限に収束する。しかしここではそうならない。W 自身を被積分関数とすると、右端点和は左端点和よりも次の量だけ大きくなる。

$$\sum_i W_{t_{i+1}}\,\Delta W_i - \sum_i W_{t_i}\,\Delta W_i = \sum_i \left(W_{t_{i+1}} - W_{t_i}\right)\Delta W_i = \sum_i (\Delta W_i)^2 \;\longrightarrow\; t,$$

ここで極限には先ほど証明した命題を用いた。「同じ」積分に対する 2 つの離散化（discretization）が、有限かつ決定論的な量だけ食い違うのである。1000 ステップからなる 1 本の経路で確かめてみよう。

```
n_lr = 1000
dt_lr = 1.0 / n_lr
t_lr = np.linspace(0, 1, n_lr + 1)
dW_lr = np.sqrt(dt_lr) * rng.standard_normal(n_lr)
W_lr = np.concatenate([[0.0], np.cumsum(dW_lr)])
left = np.sum(W_lr[:-1] * dW_lr)             # integrand at the left endpoint
right = np.sum(W_lr[1:] * dW_lr)             # integrand at the right endpoint
```

```
print(f'left sum {left:.4f}   right sum {right:.4f}')
print(f'right - left {right - left:.4f}   sum of (dW)^2 {np.sum(dW_lr**2):.4f}')
```

```
left sum 0.1655   right sum 1.1462
right - left 0.9808   sum of (dW)^2 0.9808
```

左端点和は実行するたびに変化するランダムな数値である。右端点和は、上の恒等式が示すとおり、表示されたすべての桁において増分の二乗和の分だけ左端点和を上回る。実行結果から新たにわかるのはその差の大きさである。この差は通常、約 0.1 以内の幅で 1 の近くに収まり（標準偏差は $\sqrt{2/1000} \approx 0.045$）、メッシュをさらに細かくしても埋まることはない。

両方の和は極限を持ち、ここまでの道具だけでそれを求められる。両者の和は望遠鏡和（中間の項が打ち消し合う和）になる。なぜなら、任意の分割上で $\left(W_{t_{i+1}} + W_{t_i}\right)\Delta W_i = W_{t_{i+1}}^2 - W_{t_i}^2$ を足し合わせると W_t^2 になり、さらに両者の差は t に収束するからである。したがって平均二乗の意味で、左端点和は $\frac{1}{2}\left(W_t^2 - t\right)$ に、右端点和は $\frac{1}{2}\left(W_t^2 + t\right)$ に収束する。通常の微分積分であれば $\frac{1}{2}W_t^2$ と答えるところだが、これはどちらの極限でもなく両者の平均である。評価点を 1 つ選ばなければならず、その選び方によって答えが変わる。2 つの和のうち、どちらを積分と定めるべきだろうか。

Itô の選択は左端点であり、これにより被積分関数はこれから起こる増分を決して見ることができない。公平な賭けを考えればその理由がわかる。ΔW_i を i 回目の公平な賭けの結果、H_i をそこに賭ける賭け金と考えよう。賭け手は賭けが決着する前の時刻 t_i までにわかっている情報だけから賭け金を決めなければならず、獲得賞金の合計は $\sum_i H_i\,\Delta W_i$ となる。増分 ΔW_i は、時刻 t_i 時点で既知のあらゆる情報（H_i を含む）と独立であるため、各回の期待獲得額は $\mathbb{E}[H_i\,\Delta W_i] = \mathbb{E}[H_i]\,\mathbb{E}[\Delta W_i] = 0$ となり、期待獲得額を正にする賭け戦略は存在しない。もし結果を盗み見て賭け金を決められるなら、平均して利益を出せてしまうだろう。右端点和はそのような賭け金を置くことに相当し、実際にその平均はゼロではなく t になる。

定義はこの規則をそのまま書き下したものである。各区間における値がその区間の増分を引く前に決まっているとき、被積分関数は**予測可能**（predictable）であるという。**単純予測可能過程**（simple predictable process）H とは、各区間で定数 $H_s = H_i$（$s \in (t_i, t_{i+1}]$）をとる過程であり、ここで H_i は時刻 t_i までの経路によって定まり、$\mathbb{E}[H_i^2] < \infty$ を満たすものとする。その**伊藤積分**（Itô integral）は獲得賞金の和として次で定義される。

$$\int_0^t H_s\,dW_s \;=\; \sum_i H_i\left(W_{t_{i+1}} - W_{t_i}\right). \tag{38.29}$$

公平な賭けの議論から、この積分の平均はゼロ、すなわち $\mathbb{E}\left[\int_0^t H\,dW\right] = 0$ である。さらに分散も計算できる。

命題（伊藤の等長性） 単純予測可能な H に対して、次が成り立つ。

$$\mathbb{E}\left[\left(\int_0^t H_s\,dW_s\right)^2\right] \;=\; \int_0^t \mathbb{E}[H_s^2]\,ds. \tag{38.30}$$

証明 Equation 38.29 の和の二乗を展開する。注意を要するのは交差項である。$i < j$ の交差項は因子 ΔW_j を含み、これは時刻 t_j 時点で既知のあらゆる量（積 $H_i\,\Delta W_i\,H_j$ を含む）と独立である。したがって期待値は積に分解でき、$\mathbb{E}\left[H_i\,\Delta W_i\,H_j\,\Delta W_j\right] = \mathbb{E}\left[H_i\,\Delta W_i\,H_j\right]\mathbb{E}[\Delta W_j] = 0$

となる。対角項も同様に $\mathbb{E}\big[H_i^2\big]\,\mathbb{E}\big[(\Delta W_i)^2\big] = \mathbb{E}\big[H_i^2\big]\,\Delta t_i$ へと分解される。対角項を足し合わせれば $\sum_i \mathbb{E}[H_i^2]\,\Delta t_i = \int_0^t \mathbb{E}[H_s^2]\,ds$ が得られる。■

この等長性（isometry）によって、定義を階段状の過程から $H = W$ 自身のような被積分関数へも拡張できる。等長性は、写像 $H \mapsto \int H\,dW$ がノルム $\|H\|^2 = \int_0^t \mathbb{E}[H_s^2]\,ds$ を保つことを意味している。したがって、単純予測可能過程の列がこのノルムにおいて H に収束すれば、それらの積分も平均二乗収束し、その極限をもって $\int_0^t H\,dW$ の定義とする。平均ゼロの性質と等長性は極限でも保たれ、有限なノルムを持つすべての予測可能な H に対して成り立つ。また、この種の連続な被積分関数に対する左端点和はその積分に収束する。これらの主張の証明は省略する（Øksendal (2003) の第 3 章を参照）。

これにより、出発点とした積分の値が左端点和の極限として定まる。

$$\int_0^t W_s\,dW_s = \tfrac{1}{2}\left(W_t^2 - t\right). \tag{38.31}$$

これは通常の微分積分の答えから $t/2$ を引いたものである。この $-t/2$ があることで、公平な賭けが持つべき平均ゼロが実現される（$\mathbb{E}[W_t^2] = t$ であるため）。この積分に対して等長性を確かめることもできる。等長性はノイズ積分の分散を二次モーメント（second moment）の通常の積分へと変換する。ここでは $\int_0^1 \mathbb{E}[W_s^2]\,ds = \int_0^1 s\,ds = \tfrac{1}{2}$（$t = 1$ のとき）となり、閉形式の結果とも一致する。実際、W_1^2 は標準正規確率変数の二乗であって分散 2 を持つため、$\mathrm{Var}\big(\tfrac{1}{2}(W_1^2 - 1)\big) = \tfrac{1}{4}\cdot 2 = \tfrac{1}{2}$ となる。

他の評価点を選べば別の積分が定義される。中点の選択、あるいは左端点和と右端点和の平均は、**ストラトノビッチ積分**（Stratonovich integral）を定義する (Stratonovich 1966)。上の平均 $\tfrac{1}{2}W_t^2$ が示唆するとおり、この積分は通常の連鎖律に従うが、被積分関数がこれから起こる増分の半分を見てしまうため、平均ゼロの性質を失う。本書では 2 つの理由から伊藤の流儀を採用する。第 1 に平均ゼロであるため期待値の計算が容易になること、第 2 にステップの始点で係数を評価する最も単純なシミュレーションのステップが伊藤和になることである（演習問題 6）。

38.4.2.3 ドリフト、拡散、解

積分が手に入ったことで、順過程のテンプレート Equation 38.25 に数学的な意味が備わる。**確率微分方程式**（stochastic differential equation、SDE）は次のように書かれる。

$$dX = f(X,t)\,dt + g(X,t)\,dW, \tag{38.32}$$

この微分形は次の積分方程式の略記である。

$$X_t = X_0 + \int_0^t f(X_s,s)\,ds + \int_0^t g(X_s,s)\,dW_s,$$

すなわち、通常の積分と伊藤積分の和である。短いステップにおいて係数をその始点の値に固定すると、$\mathbb{E}[dX] = f\,dt$ および $\mathrm{Var}(dX) = g^2\,dt$ となる。したがって**ドリフト** f は条件付き平均速度であり、**拡散**係数（diffusion coefficient）g は分散が注入される速度 g^2 を定める。$g \equiv 0$ のときこの方程式は 38.1 節の ODE になり、$f \equiv 0$ かつ定数 g のときは $X_t = X_0 + g\,W_t$、すなわち Figure 38.11 のノイズのみの過程になる。

では、解とは何だろうか。ODE では 1 本の曲線であったが、ここでは各ブラウン経路に対して 1 本の曲線が対応する。f と g が x に関してリプシッツ連続で高々線形にしか増大せず（$|f(x,t)| + |g(x,t)| \le C(1+|x|)$）、かつ X_0 が W と独立で $\mathbb{E}[X_0^2] < \infty$ を満たすならば、始点とブラウン経路によって解の経路が一意に定まる（Øksendal (2003)、定理 5.2.1）。その証明は 38.1.1.3 節の Picard 反復を確率版に置き換えたものであり、積分の評価の代わりに等長性を用いる。したがって、ランダム性はすべて W の中にある。同じ始点から毎回新しいノイズを用いて方程式を何度も走らせれば、Figure 38.11 のような経路の群れが得られ、生成モデル（generative model）が関心を持つのは各時刻におけるその点群の法則である。

シミュレーションを行う際には 1 つの区別が重要になる。振幅が時間にのみ依存する場合（$g = g(t)$）、ノイズは**加法的**（additive）であるという。テンプレート Equation 38.25 や、通し例を含む本章のすべての拡散モデル SDE がこれに該当する。一方、水準に比例して変動する株価のように、振幅が状態に依存する場合（$g = g(X, t)$）、ノイズは**乗法的**（multiplicative）であるという。この違いは表面的に見えるかもしれないが、シミュレーションされた経路が真の経路をどれだけ正確に追跡できるかを決定づける（38.4.4.2 節）。

38.4.2.4 伊藤の補題

SDE を使って計算を進めるには連鎖律が必要になる。すなわち、X が SDE の解であるとき、関数 $\phi(X_t, t)$ はどの方程式を満たすのだろうか。通常の連鎖律を当てはめると $d(W^2) = 2W\,dW$ となるが、これが誤りであることはすでにわかっている。$\int_0^t 2W\,dW = W_t^2 - t$ が Equation 38.31 より成り立つからである。何かが欠けているのだ。

ランダムウォークの 1 ステップを見れば、欠けている項がどこから生じるのか、また他の関数に対してどのような形になるのかがわかる。$\phi(x) = x^2$ とし、大きさ $\pm\sqrt{\Delta t}$ の 1 ステップを点 x から各符号とも確率 $\tfrac{1}{2}$ で踏み出すとする。ステップ後の ϕ の平均は次のようになる。

$$\tfrac{1}{2}\left[(x+\sqrt{\Delta t})^2 + (x-\sqrt{\Delta t})^2\right] = x^2 + \Delta t = \phi(x) + \tfrac{1}{2}\phi''(x)\,\Delta t,$$

ここで $\phi'' = 2$ を用いた。この増加分はステップの二乗から生じており、コイントスではそれが毎回 Δt になるのに対し、通常の連鎖律はそれを捨ててしまうのである。

$x = 0.7$、$\Delta t = 0.25$ とすると、1 ステップ後の到達点は 0.2 または 1.2 となり、そこでの ϕ の値は 0.04 または 1.44 である。その平均 0.74 は $\phi(0.7) = 0.49$ よりも $0.25 = \Delta t$ だけ上にある（Figure 38.14）。凸関数（convex function）は対称なノイズから正の利得を受け取り、ブラウン増分の二乗はオーダー dt であるため、単位時間あたりの利得 $\tfrac{1}{2}\phi''$ は極限でも消えない。

一般の公式も同じ計算から導かれる。X を $dX = f\,dt + g\,dW$ の解とし、$\phi(x,t)$ を滑らかな関数とする。1 次までではもはや不十分であるため 2 次まで展開すると、ϕ の微小変化は次のようになる。

$$d\phi = \phi_t\,dt + \phi_x\,dX + \tfrac{1}{2}\phi_{xx}\,(dX)^2 + \cdots.$$

$dX = f\,dt + g\,dW$ を代入し、$(dX)^2$ を乗積表 Equation 38.28 に従って展開する。

$$(dX)^2 = f^2(dt)^2 + 2fg\,dt\,dW + g^2(dW)^2 = g^2\,dt.$$

ドリフト項と交差項は消え、ノイズ項が 1 次のオーダーで生き残る。項をまとめれば確率解析の連鎖律が得られる。ここまでの議論は発見的な導出であるため、結果は証明なしの定理として述べる。

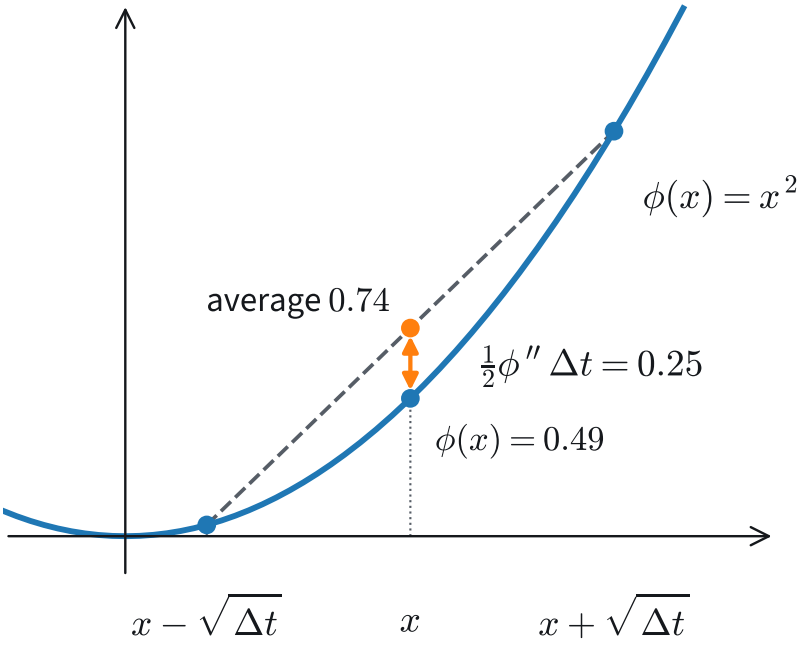


Figure 38.14: イェンセンのギャップとしての伊藤補正。$\pm\sqrt{\Delta t}$ の 1 ステップを x から踏み出すと、凸な ϕ の平均は $\phi(x)$ よりも $\frac{1}{2}\phi''(x)\,\Delta t$ だけ上に移る。ここでは $\phi = x^2$、$x = 0.7$、$\Delta t = 0.25$ であり、ギャップは 0.25 である。

定理（伊藤の補題） 前項の条件のもとで X を $dX = f\,dt + g\,dW$ の解とし、$\phi \in C^{1,2}$、すなわち時間に関して 1 回、状態に関して 2 回連続微分可能であるとする。このとき $Y_t = \phi(X_t, t)$ は次を満たす。

$$d\phi \;=\; \left(\phi_t + f\,\phi_x + \tfrac{1}{2} g^2\,\phi_{xx}\right) dt \;+\; g\,\phi_x\,dW. \tag{38.33}$$

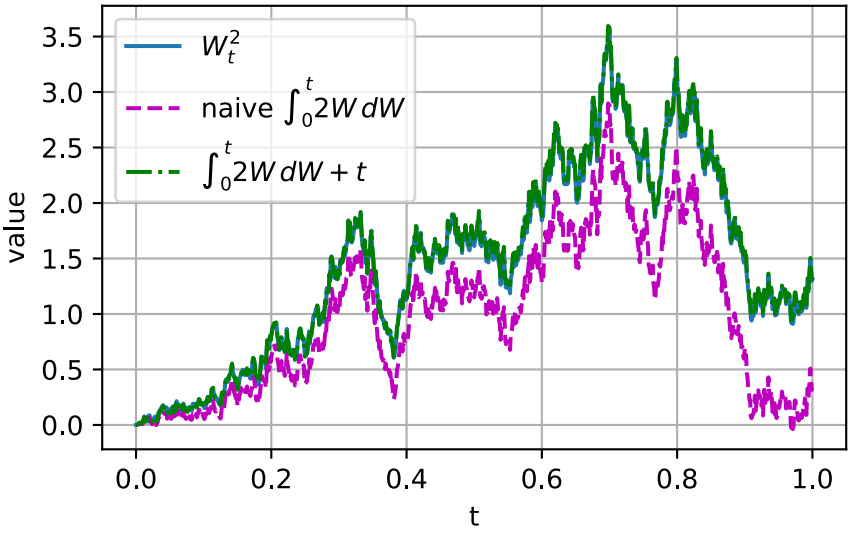
これがこの分野における実用上の基本公式であり、通常の連鎖律に、二階導関数の半分と分散が到着する速度との積を加えた形になっている。新しく加わった項 $\frac{1}{2}g^2\phi_{xx}\,dt$ は**伊藤補正**（Itô correction）と呼ばれる。証明ではテイラー展開の剰余項を平均二乗の意味で抑える必要があり（Øksendal (2003) の第 4 章）、状態に関する一階・二階の両方の導関数が不可欠である。したがって、この補題は $\phi(x) = |x|$ のような折れ目を持つ関数にはそのままでは適用できない。

例 1：W^2。 $X = W$（したがって $f = 0$、$g = 1$）および $\phi(x) = x^2$ とする。補題より $d(W^2) = 2W\,dW + dt$ となり、通常の連鎖律が落としていた dt が現れる。積分形で書けば $W_t^2 = \int_0^t 2W_s\,dW_s + t$ であり、これは再び Equation 38.31 に一致する。補題のたった 1 行が望遠鏡和の計算に取って代わったのである。

補正項が蓄積していく様子は実際に観察できる。任意のグリッド上で、W_t^2 と $\int_0^t 2W\,dW$ の左端点和との差は望遠鏡和 $W_t^2 - \sum_{t_i<t} 2W_{t_i}\Delta W_i = \sum_{t_i<t}(\Delta W_i)^2$ になるため、これは累積の二次変分（quadratic variation）に一致し、直線 t に沿って推移するはずである。先ほどの左端点和・右端点和で用いた経路を再利用しよう。

```
ito = np.concatenate([[0.0], np.cumsum(2 * W_lr[:-1] * dW_lr)])  # int 2W dW
gap = W_lr**2 - ito
print(f'final gap W_1^2 - int 2W dW = {gap[-1]:.4f}   (Ito correction: t = 1)')
print(f'max |gap_t - t| along the path = {np.abs(gap - t_lr).max():.4f}')
d2l.plot(t_lr, [W_lr**2, ito, ito + t_lr], 't', 'value',
         legend=['$W_t^2$', 'naive $\\int_0^t 2W\\,dW$',
                 '$\\int_0^t 2W\\,dW + t$'], figsize=(5, 3))
```

```
final gap W_1^2 - int 2W dW = 0.9808   (Ito correction: t = 1)
max |gap_t - t| along the path = 0.0505
```



naive とラベル付けされた曲線は伊藤積分 $\int_0^t 2W\,dW$ であり、通常の連鎖律が W_t^2 とみなしてしまう量である。この曲線は W_t^2 の下に垂れ下がるが、t を加えると元の曲線にぴったり重なる。経路全体にわたって差は通常、約 0.1 以内の幅で t の近くに収まる。Equation 38.27 の証明より、時刻 t におけるその標準偏差はこのグリッド上で $\sqrt{2t\,\Delta t} \le 0.045$ だからである。$t = 1$ における差は、前のセルで出力した増分の二乗和に一致する。

例 2：幾何ブラウン運動（geometric Brownian motion）。SDE $dX = \mu X\,dt + \nu X\,dW$ $(X_0 > 0)$ は、平均成長率 μ で増大し、自身の大きさに比例するノイズを伴う量をモデル化する。これは Samuelson による株価モデルであり (Samuelson 1965)、Black と Scholes のオプション価格公式の基礎となっている (Black and Scholes 1973)。そのノイズは乗法的である。$\phi(x) = \log x$ に補題を適用しよう。このとき $\phi' = 1/x$、$\phi'' = -1/x^2$、$g^2 = \nu^2 x^2$ である。対数関数は $x > 0$ でのみ滑らかであるため、ひとまず X が正にとどまると仮定する。

$$d\log X = \left(\mu - \tfrac{1}{2}\nu^2\right)dt + \nu\,dW, \qquad \text{so} \qquad X_t = X_0 \exp\!\left(\left(\mu - \tfrac{1}{2}\nu^2\right)t + \nu W_t\right). \tag{38.34}$$

得られた式は正の値をとり、これを W_t と t の関数とみなして補題を適用すれば元の方程式を満たすことが確かめられるため、先ほどの仮定は外せる。$-\frac{1}{2}\nu^2$ は $\phi'' < 0$ に対する補正であり、凹関数（concave function）は対称なノイズによって損をする。そのためノイズは典型的な成長率 $\mu - \frac{1}{2}\nu^2$ を平均成長率 μ よりも押し下げ、演習問題 3 では X_t の平均と中央値（median）がどれほど乖離するかを示す。後の項では、この厳密解を基準にしてシミュレーション誤差を測定する。

多次元でも構造は同じであり、結果は既知のものとして認める (Øksendal 2003)。$d\mathbf{X} = \mathbf{f}\,dt + \mathbf{G}\,d\mathbf{W}$ において、$\mathbf{X} \in \mathbb{R}^d$、$\mathbf{G}$ を行列、$\mathbf{W}$ を各座標が独立なスカラーのブラウン運動であるブラウン運動とすると、乗積表は $dW_i\,dW_j = \delta_{ij}\,dt$ となり、2 回連続微分可能な ϕ は次を満たす。

$$d\phi(\mathbf{X}) \;=\; \nabla\phi \cdot d\mathbf{X} \;+\; \tfrac{1}{2}\sum_{i,j} \left(\mathbf{G}\mathbf{G}^{\top}\right)_{ij} \frac{\partial^2 \phi}{\partial x_i\,\partial x_j}\,dt. \tag{38.35}$$

補正項は、各二階偏導関数をノイズの共分散速度 $\mathbf{G}\mathbf{G}^{\top}$ で重み付けする。本章の拡散モデル SDE は

すべての座標を対等に扱うため $\mathbf{G} = g(t)\,\mathbf{I}$ であり、このとき補正項は $\frac{1}{2}g(t)^2 \sum_i \partial^2\phi/\partial x_i^2 \; dt$、すなわち ϕ のラプラシアンの定数倍へと帰着する。次節ではこの形の補題が必要になる。

38.4.3 オルンシュタイン・ウーレンベック過程

ここで通し例に名前を与え、その解を求めよう。オルンシュタイン・ウーレンベック（OU）過程 (Uhlenbeck and Ornstein 1930) とは次の方程式で定まる過程である。

$$dX = -\theta X\,dt + g\,dW, \qquad \theta > 0, \tag{38.36}$$

すなわち、減衰 ODE $\dot{x} = -\theta x$（38.1.2 節）に定数振幅 g のノイズを加えたものである。考案者たちは、分子からの衝突で弾かれつつ摩擦によって減速する粒子の速度を表すためにこの式を書き下した。ドリフトはゼロから外れた変位をレート θ でゼロへと引き戻し、ノイズは絶えず新たな変位を生み出し続ける（Figure 38.15）。冒頭の漸化式は、$\theta = 1$、$g = \sqrt{2}$ としたこの過程を、各ステップでドリフトを固定しながらステップ幅 $h = 0.01$ のグリッド上で時間発展させたものである。1 ステップごとに $1 - \theta h = 0.99$ を掛け、標準偏差 $g\sqrt{h} = \sqrt{0.02}$ のノイズを加えている。

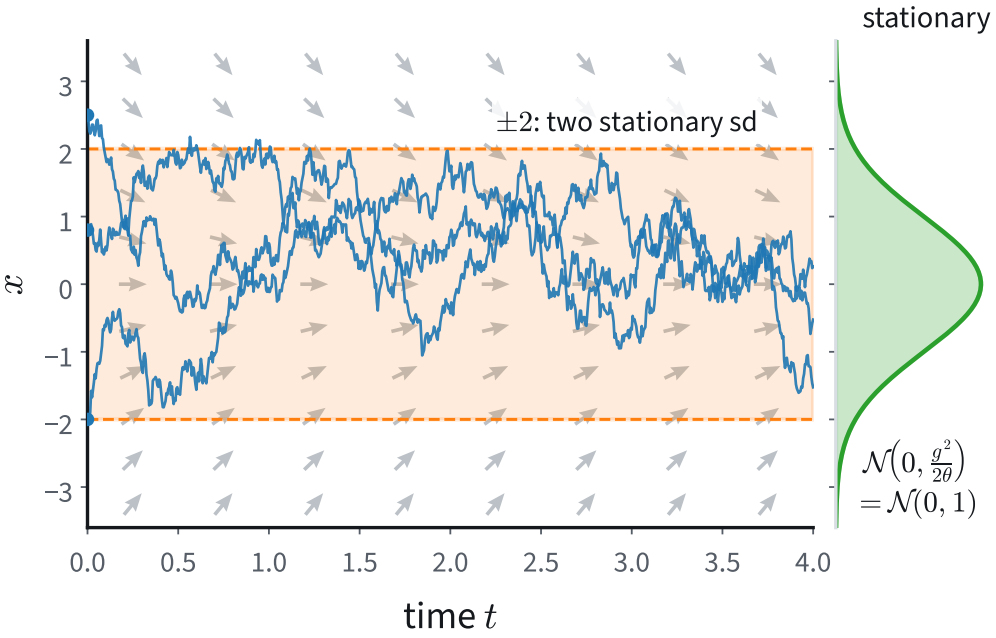


Figure 38.15: 平均回帰。ドリフト $-\theta x$（矢印）があらゆる経路を 0 へ向けてレート $\theta = 1$ で引き戻す一方、振幅 $g = \sqrt{2}$ のノイズが経路を動かし続ける。3 つの始点からの経路は $1/\theta$ の数倍の時間のうちに始点を忘れ、その後は極限法則の標準偏差の 2 倍の帯域 $\pm 2\sqrt{g^2/2\theta} = \pm 2$（stationary と表記）の内側で揺れ動く。右はその密度である。

38.4.3.1 遷移カーネル

OU 過程は、あらゆる時刻における法則が閉形式でわかる数少ない SDE の 1 つであり、拡散モデルがこの過程の上に構築される理由もそこにある。X_t の条件付き法則（始点 $X_0 = x_0$ が与えら

れたとき）を始点の関数として読んだものを**遷移カーネル**（transition kernel）と呼ぶ（畳み込みのカーネルとは無関係である）。OU 過程ではこれはガウス分布になる。本節のすべての結果の中で、本書の以降の章で最も頻繁に用いるのがこの結果である。

命題（OU 遷移カーネル） OU 過程 Equation 38.36 に対して、次が成り立つ。

$$X_t \mid X_0 = x_0 \;\sim\; \mathcal{N}\!\left(x_0\, e^{-\theta t},\; \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right) \right). \tag{38.37}$$

証明積分因子 $e^{\theta t}$ は 38.1.2.3 節において入力を伴う減衰方程式を（定数変化法の公式により）解いたが、入力がノイズであっても同様に機能する。伊藤の補題 Equation 38.33 を $\phi(x,t) = e^{\theta t}x$ に適用する。このとき $\phi_t = \theta e^{\theta t}x$、$\phi_x = e^{\theta t}$、$\phi_{xx} = 0$ であるため、補正項はゼロになる。

$$d(e^{\theta t}X_t) = \left(\theta e^{\theta t}X_t - \theta e^{\theta t}X_t\right)dt + e^{\theta t} g\, dW_t = g\, e^{\theta t}\, dW_t.$$

ドリフトが打ち消し合い、0 から t まで積分して $e^{-\theta t}$ を掛けると次式が得られる。

$$X_t = X_0\, e^{-\theta t} + g \int_0^t e^{-\theta(t-s)}\, dW_s. \tag{38.38}$$

残った積分の被積分関数は決定論的であるため、独立なガウス増分の重み付き和の極限となり、平均ゼロのガウス分布に従う（ガウス確率変数の平均二乗極限がガウス分布になることは認める）。その分散は伊藤の等長性 Equation 38.30 から求まる。

$$\operatorname{Var}(X_t \mid X_0) = g^2 \int_0^t e^{-2\theta(t-s)}\, ds = \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right). \qquad \blacksquare$$

解 Equation 38.38 は薄れゆく記憶として解釈できる。始点はレート θ で減衰し、時刻 s に加わったノイズの揺らぎは重み $e^{-\theta(t-s)}$ を伴って生き残る。つまり直近のノイズは効き、古いノイズは忘れ去られる。漸化式では、k ステップ前の揺らぎが重み $0.99^k \approx e^{-0.01k}$ で残る。これ以降の各節、および本編の 17.2 節では、平均係数と標準偏差という 2 つの係数を用いてこのカーネルを次のように書く。

$$\alpha_t = e^{-\theta t}, \qquad \sigma_t^2 = \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right), \qquad \text{so that} \qquad X_t \mid X_0 = x_0 \sim \mathcal{N}(\alpha_t x_0,\, \sigma_t^2). \tag{38.39}$$

データは平均 $\alpha_t x_0$ を通じてのみ入り、ノイズは分散 σ_t^2 を通じてのみ入る。通し例（$\theta = 1$、$g = \sqrt{2}$、$\sigma_t^2 = 1 - e^{-2t}$）を $x_0 = 2$ から出発させると、カーネルは $\mathcal{N}(1.213,\, 0.795^2)$ $(t = 0.5)$、$\mathcal{N}(0.736,\, 0.930^2)$ $(t = 1)$、そして $\mathcal{N}(0.037,\, 1.000^2)$ $(t = 4)$ となる。あらゆる時刻・あらゆる始点に対してカーネルがガウス分布であるため、始点を条件とするノイズ付き状態については、対数密度（log-density）の勾配 $-(x - \alpha_t x_0)/\sigma_t^2$ を含め、すべてが閉形式でわかる。生成はこの勾配に依存することが後に明らかになる。

漸化式はこの過程の離散化にすぎないため、その点群はカーネルに近く従うはずだが、厳密に一致するとは限らない。10,000 個の $x_0 = 2$ のコピーに対して単純なループで漸化式を実行し、5 つの時刻における点群を保存して、その平均と標準偏差を $2\alpha_t$ および σ_t と並べて出力し、あわせて $2\alpha_t \pm 2\sigma_t$ の内側に入る割合も出力しよう。

```
theta_ou, g_ou, h_ou = 1.0, np.sqrt(2.0), 0.01
X_ou = np.full(10000, 2.0)                    # 10,000 copies of x_0 = 2
snap_steps = {round(t / h_ou): t for t in (0.5, 1.0, 1.5, 2.0, 4.0)}
X_ou_snap = {}
for k in range(1, 401):
    xi = rng.standard_normal(X_ou.size)
    X_ou = (1 - theta_ou * h_ou) * X_ou + g_ou * np.sqrt(h_ou) * xi
    if k in snap_steps:
        X_ou_snap[snap_steps[k]] = X_ou.copy()
print(f'{"t":>4} {"mean":>6} {"2alpha_t":>8} {"std":>6} {"sigma_t":>7}'
      f' {"inside":>7}')
for t, X in X_ou_snap.items():
    alpha_t = np.exp(-theta_ou * t)
    sigma_t = np.sqrt(g_ou**2 / (2 * theta_ou) * (1 - alpha_t**2))
    inside = np.mean(np.abs(X - 2 * alpha_t) < 2 * sigma_t)
    print(f'{t:4.1f} {X.mean():6.3f} {2 * alpha_t:8.3f} {X.std():6.3f} '
          f'{sigma_t:7.3f} {inside:7.3f}')
```

```
   t   mean 2alpha_t    std sigma_t  inside
 0.5  1.212    1.213  0.801   0.795   0.955
 1.0  0.737    0.736  0.926   0.930   0.952
 1.5  0.440    0.446  0.970   0.975   0.957
 2.0  0.259    0.271  1.000   0.991   0.954
 4.0  0.043    0.037  1.004   1.000   0.954
```

平均と標準偏差は、標準誤差（standard error）の 2〜3 倍の範囲内でカーネルと一致する（平均では約 0.01、標準偏差では 0.007 が $10{,}000$ サンプルに対する標準誤差である）。ガウス分布と同様に、どの時刻でも点群の約 95% が帯域の内側に入っている。単一の点 2 として出発した点群は素早く広がった後、それ以上は広がらなくなる。このサンプル数では、離散化の影響はまだ見て取れない。

38.4.3.2 定常分布

順過程は始点を忘れなければならず、カーネル Equation 38.37 は OU 過程が実際にそうであることを示している。$t \to \infty$ のとき平均 $x_0 e^{-\theta t}$ はゼロへと減衰し、分散は $g^2/(2\theta)$ に近づくため、次が成り立つ。

$$X_t \mid X_0 = x_0 \;\longrightarrow\; \mathcal{N}\!\left(0,\, \frac{g^2}{2\theta}\right) \quad \text{in distribution, for every } x_0.$$

これこそが 38.4.1.1 節で求めた、あらゆるデータに共通の終着点である。この極限は単なる終着点にとどまらない。この分布から出発した過程はそのままそこにとどまり続け、この性質を持つ法則を**定常**（stationary）と呼ぶ。これを確かめるため、$X_0 \sim \mathcal{N}(0, g^2/2\theta)$ を W と独立にとろう。すると Equation 38.38 より、X_t は $X_0 e^{-\theta t}$ と分散 σ_t^2 の独立なガウス確率変数との和になるため、平均ゼロのガウス分布に従い、その分散は次のようになる。

$$e^{-2\theta t}\,\frac{g^2}{2\theta} + \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right) = \frac{g^2}{2\theta}.$$

始点の減衰によって失われる分散と、ノイズによって加わる分散とがちょうど釣り合っている。では、始点を忘れるにはどれほどの時間がかかるのか。その時間スケールは**緩和時間**（relaxation

time）と呼ばれる $1/\theta$ であり、その数倍の時間が経てば $e^{-\theta t}$ は無視できるほど小さくなる。通し例では $g^2/2\theta = 1$ かつ $1/\theta = 1$ であるため、定常分布（stationary distribution）は $\mathcal{N}(0,1)$ であり、$t = 4$ までに平均は 2 から 0.037 へと低下していた。

漸化式もそれ自身の定常分布を持つが、わずかに異なる値になる。1 ステップは分散 v を $(1-\theta h)^2 v + g^2 h$ へと写す。$v = 1$ から通し例の 1 ステップを進めると $0.9801 + 0.02 = 1.0001$ と 1 をわずかに上回り、この写像の不動点（fixed point）は次のようになる。

$$v_\infty = \frac{g^2 h}{1-(1-\theta h)^2} = \frac{0.02}{0.0199} = 1.005 \qquad (h = 0.01).$$

したがって漸化式は、厳密な分散 1 よりも 0.5% 高い値に落ち着く。10,000 個の点群では 0.5% のずれを見分けられないが、手計算なら見抜ける。これが離散化誤差の最初の兆候であり、38.4.4.2 節ではこの誤差が h に比例して縮小することを示す。

38.4.3.3 分散保持の正規化

定常分散 $g^2/2\theta$ は自由パラメータであり、拡散モデルでは次のように選んでこれを 1 に固定する。

$$g^2 = 2\theta \qquad \Longrightarrow \qquad X_t \mid X_0 = x_0 \;\longrightarrow\; \mathcal{N}(0,1).$$

通し例でもこの選択をとっている。このとき $\sigma_t^2 = 1 - \alpha_t^2$ となるため、ただ 1 つの係数でカーネルを記述できる。

これまでは始点を単一の数値 x_0 としていたが、今度はデータからのサンプル X_0 としよう。すると分布の意味で、解 Equation 38.38 は $X_t = \alpha_t X_0 + \sqrt{1-\alpha_t^2}\,\epsilon$（$\epsilon$ は X_0 と独立な標準ガウス確率変数）と表される。データ全体にわたってカーネルを平均したこの X_t の法則を、時刻 t における**周辺分布**（marginal）と呼ぶ。多くの論文ではこれを別の文字で表記する。すなわち、Ho et al. (2020) によるノイズ除去拡散確率モデル（denoising diffusion probabilistic model、DDPM）、つまり 17.2 節の離散時間拡散モデルに倣い、ここでの $\alpha_t^2 = e^{-2\theta t}$ を $\bar{\alpha}_t$ と書くのである。本編でも同じ記法を用いており、ここでもそれに従う。

$$X_t \;=\; \sqrt{\bar{\alpha}_t}\, X_0 \;+\; \sqrt{1-\bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0,1) \text{ independent of } X_0. \tag{38.40}$$

この周辺分布は、二乗和が 1 になる重みを用いたデータと新しいノイズの重み付き混合である。データの平均がゼロ、分散が 1 であるとしよう。独立な 2 項の分散をとると次式が得られる。

$$\mathrm{Var}(X_t) = \bar{\alpha}_t \mathrm{Var}(X_0) + (1-\bar{\alpha}_t) = \bar{\alpha}_t + (1-\bar{\alpha}_t) = 1 \qquad \text{for all } t.$$

これが**分散保持**（variance-preserving、VP）の意味である。単位分散のデータに対しては、極限だけでなくあらゆる時刻において周辺分散が 1 に等しくなる。この過程は、データ側が担う割合 $\bar{\alpha}_t$ とすでにノイズへ置き換わった割合 $1-\bar{\alpha}_t$ との間で合計を変えずにデータからノイズへと分散を移していく。もっとも、この名称が文字どおり当てはまるのは単位分散のデータに限られる。分散 s^2 のデータでは周辺分散が s^2 から 1 へと変化するため（演習問題 5）、データのスケールが重要になる。17.1 節のように $[-1,1]$ へスケールした画像の分散は 1 未満である一方、潜在拡散（latent diffusion）モデルではほぼ単位分散になるようリスケールする (Rombach et al. 2022)。

分散は保たれるが、分布の形状は保たれない。コイントスの結果のように、± 1 をそれぞれ確率 $\frac{1}{2}$ でとるデータを考えよう。その 4 次モーメントは 1 であるのに対し標準ガウス分布では 3 であり、過程が進むにつれて $\mathbb{E}[X_t^4] = 3 - 2\bar{\alpha}_t^2$ に従って上昇し、通し例では 2.264（$t = 0.25$）、さらに 2.963（$t = 1$）に達する（同じ演習問題を参照）。分散が微動だにしないまま、コインの分布は釣鐘型の曲線へと姿を変えるのである。

拡散モデルでは通常、レートを時間とともに変化させる。スコアベース拡散モデルの **VP-SDE** (Y. Song, Sohl-Dickstein, et al. 2021) は次式で与えられる。

$$dX = -\tfrac{1}{2}\beta(t)\, X\, dt + \sqrt{\beta(t)}\, dW, \tag{38.41}$$

これは時間依存のレート $\theta(t) = \beta(t)/2$ とノイズ振幅 $g(t)^2 = \beta(t)$ を持つ OU 過程であり、通し例は定数スケジュール $\beta = 2$ にあたる。あらゆる瞬間で $g^2 = 2\theta$ が保たれるため、スケジュール全体にわたって分散は 1 にとどまり、その周辺分布は Equation 38.40 において $\bar{\alpha}_t = e^{-\int_0^t \beta(s)\, ds}$ としたものになる（演習問題 7 でその遷移が厳密にガウス分布になることを証明する）。これが 17.2 節の分散保持経路、すなわち拡散モデルが逆転させることを学習する順過程である。

38.4.4 シミュレーション

38.4.4.1 オイラー・丸山法

ほとんどの SDE は閉形式解（closed-form solution）を持たないため、シミュレーションを行う必要がある。その方法は冒頭の漸化式がすでに示している。Equation 38.32 をグリッド $t_n = n\,\Delta t$ 上でシミュレーションするには、ODE に対する前進オイラー法（38.2 節）と同様に各ステップで係数を固定し、増分の表現 Equation 38.26 が定める大きさのガウスノイズを加える。**オイラー・丸山法**（Euler–Maruyama）の更新式 (Maruyama 1955) は次のとおりである。

$$\hat{X}_{n+1} = \hat{X}_n + f(\hat{X}_n, t_n)\,\Delta t + g(\hat{X}_n, t_n)\,\sqrt{\Delta t}\,\xi_n, \qquad \xi_n \sim \mathcal{N}(0,1)\ \text{i.i.d.}, \tag{38.42}$$

ここで $\hat{X}_n$ は X_{t_n} の近似値であり、$\hat{X}_0 = X_0$ とする。ノイズの揺らぎは $\sqrt{\Delta t}$ に比例してスケールする。他のいかなる指数でも極限でノイズが消失するか爆発してしまうからであり（38.4.1.2 節）、また $g = 0$ とすればこのスキームは前進オイラー法に一致する。

OU 過程に対して、1 ステップは次の形になる。

$$\hat{X}_{n+1} = (1 - \theta\,\Delta t)\,\hat{X}_n + g\sqrt{\Delta t}\,\xi_n = 0.99\,\hat{X}_n + 0.1414\,\xi_n, \tag{38.43}$$

2 番目の式は $\Delta t = h = 0.01$ における通し例であり、冒頭からずっと動かしてきた漸化式そのものである。

この過程に限っては、もっと良いやり方も可能であった。$g^2 = 2\theta$ のとき、1 区間にわたるカーネル Equation 38.37 は $\mathcal{N}(e^{-\theta\Delta t}x,\, 1 - e^{-2\theta\Delta t})$ であり、そこからのサンプリングは DDPM の順方向ステップ（17.2 節）、すなわち $x_k = \sqrt{1-\beta_k}\,x_{k-1} + \sqrt{\beta_k}\,\epsilon_k$（ただし $\beta_k = 1 - e^{-2\theta\Delta t}$）に一致する。したがって DDPM のステップは任意のステップサイズで厳密であり、オイラー・丸山法はその 1 次近似にあたる。$\theta = 1$、$\Delta t = 0.01$ のとき、厳密なステップは $e^{-0.01} = 0.99005$ を掛けて分散 $\beta_k = 1 - e^{-0.02} = 0.0198$ のノイズを加えるのに対し、近似では 0.99 と 0.02 であった。2 種類のベータを混同してはならない。ステップの β_k は 1 ステップでノイズに置き換わる分散の割合であり、$\beta(t)\,\Delta t$（Equation 38.41 のレート $\beta(t)$ を用いた量）におよそ等しく、こ

こでは 0.0198 対 $2 \cdot 0.01$ となっている。厳密なステップが使えるのは線形方程式に限られる（演習問題 7）。一般のドリフトには閉形式の遷移法則が存在しないため、近似スキームに頼るほかない。

以下の補助関数は、経路のバッチ全体に対して Equation 38.42 を実装したものである。増分 ΔW を引数として受け取るため、2 つのシミュレーションをまったく同じノイズで駆動できる。この機能はすぐに必要になる。ノイズが入った状態では「誤差」そのものを定義し直す必要があり、その定義には 2 種類あるからである。

```
def euler_maruyama(f, g, x0, t, dW):
    """Simulate dX = f dt + g dW on the grid t for a batch of paths."""
    X = np.empty((len(x0), len(t)))
    X[:, 0] = x0
    for k in range(len(t) - 1):
        h = t[k + 1] - t[k]
        X[:, k + 1] = (X[:, k] + f(X[:, k], t[k]) * h
                       + g(X[:, k], t[k]) * dW[:, k])
    return X
```

38.4.4.2 強収束と弱収束

シミュレーションされた経路はランダムであり、真の経路も同様である。2 つの誤差の定義は、何を比較するかという点で異なる。すなわち経路を比較するか法則を比較するかであり、これは包絡線の帯域実験で区別した 2 種類の主張に対応している。

ステップ数を $N = T/\Delta t$ とし、$\hat{X}_N$ が X_T を近似するものとする。数値スキームと厳密解を同じブラウン増分で駆動しよう。**強誤差**（strong error）$\mathbb{E}\,|\hat{X}_N - X_T|$ は経路ごとの追従性を測る。すなわち、シミュレーションされた軌跡が、同じノイズで駆動された真の軌跡をどれだけ忠実にたどるかを見る。一方、滑らかなテスト関数に対する**弱誤差**（weak error）$|\mathbb{E}\,\varphi(\hat{X}_N) - \mathbb{E}\,\varphi(X_T)|$（テスト関数は φ）は、時刻 T における法則が正しく再現されているかだけを測る。生成モデルが使うのは終端時刻のサンプルだけであるため、必要なのは弱い意味での精度である。強い意味での精度が重要になるのは、マルチレベルモンテカルロ法（multilevel Monte Carlo）のように異なる解像度の経路同士を結合する場合である。この手法は推定の大半を安価な粗い経路で行い、高価な細かい経路を少数だけ使ってそれを補正する（演習問題 8）。Figure 38.16 にこの 2 つの概念を並べて示す。

スキームが強収束次数（strong order）p を持つとは、その強誤差が高々 $C\,\Delta t^{p}$ であることをいい、弱収束次数（weak order）p も同様に定義される。オイラー・丸山法の次数は古典的な結果として知られている。

定理（オイラー・丸山法の収束次数、証明略） **f、g、φ を滑らかな関数とし、増大条件および X_0 のモーメント条件（厳密な定式化は Kloeden and Platen (1992) を参照）を満たすとする。**

1. **オイラー・丸山法は強収束次数 $\tfrac{1}{2}$ を持つ：$\mathbb{E}\,|\hat{X}_N - X_T| \le C\,\Delta t^{1/2}$。**
2. **ノイズが加法的（$g = g(t)$）であれば、強収束次数は 1 になる：$\mathbb{E}\,|\hat{X}_N - X_T| \le C\,\Delta t$。**
3. **加法的ノイズ・乗法的ノイズのいずれに対しても、弱収束次数は 1 である：$|\mathbb{E}\,\varphi(\hat{X}_N) - \mathbb{E}\,\varphi(X_T)| \le C\,\Delta t$。**

1 ステップあたりの誤差を見積もれば、これらの指数の理由がわかる。もちろんこれは厳密な証明ではない。各ステップで生じた誤差がその場にとどまるかのように足し合わせているだけであり、実際の証明では初期に生じた誤差がその後のダイナミクスによってどう伝播するかも制御しなければならない。

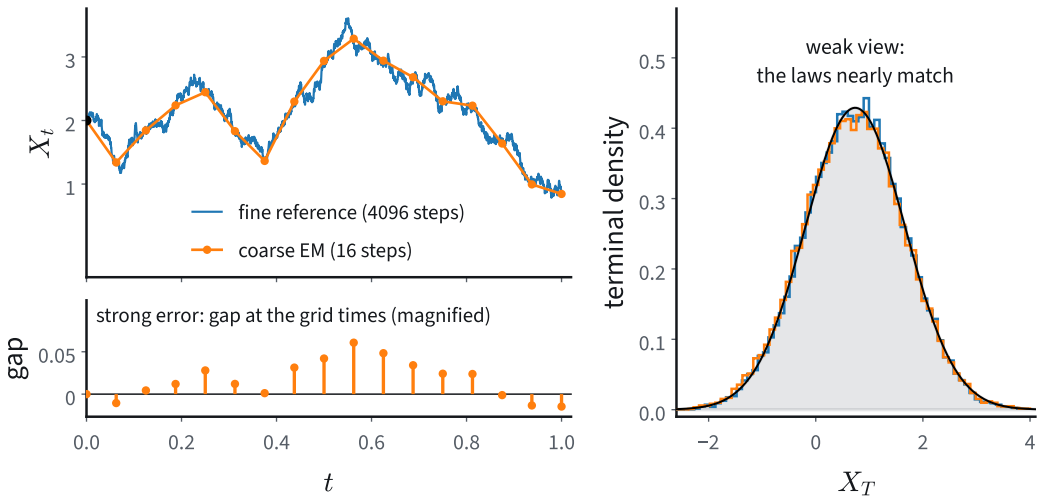


Figure 38.16: $X_0 = 2$ から出発する $[0, 1]$ 上の通し例に対する、オイラー・丸山法の強収束と弱収束。左：同じブラウン増分で駆動された 1 本の粗い経路（16 ステップ、オレンジ色）と細かい参照経路（4096 ステップ、青色）。下の帯は粗いグリッド時刻における両者の差を拡大したもので、強誤差が測るのはこの量である。右：20,000 本の経路に対する 2 つのステップサイズでの終端ヒストグラムは互いに、また厳密な法則（黒色）ともほぼ一致する（ただし 16 ステップでは分散が約 4% 大きすぎる）。

オイラー・丸山法はステップの間 g を固定するが、実際には g も経路とともに変化する。ステップ内部の時刻 s までに経路はおおよそ $g\,(W_s - W_{t_n})$ だけ動いているため、そこでのノイズ係数（noise coefficient）はおおよそ $g + g\,\partial_x g\,(W_s - W_{t_n})$ になる。このスキームが取りこぼす最大の項は第 2 項を dW で積分したものであり、これは Equation 38.31 をブラウン運動 $W_s - W_{t_n}$（時刻 t_n から新たに始まるもの）に適用した式にほかならない。

$$g\,\partial_x g \int_{t_n}^{t_{n+1}} \left(W_s - W_{t_n}\right) dW_s = \tfrac{1}{2}\, g\,\partial_x g\,\left((\Delta W)^2 - \Delta t\right).$$

ミルスタイン法（Milstein's scheme）はこの項を補い戻す (Milstein 1975)。そこに含まれるランダムな因子は、二乗に関する規則を 1 ステップずつ見たものに相当する。すなわち $(\Delta W)^2 = \Delta t$ は和をとれば成り立つが、個々のステップでは平均ゼロ・大きさ Δt のずれを伴う。$N = T/\Delta t$ 回のステップからこの種の無相関な項が積み重なり、ランダムウォークと同様に足し合わされて $\sqrt{N}\,\Delta t = \sqrt{T\,\Delta t}$ となる。これが次数 $\frac{1}{2}$ の正体である。

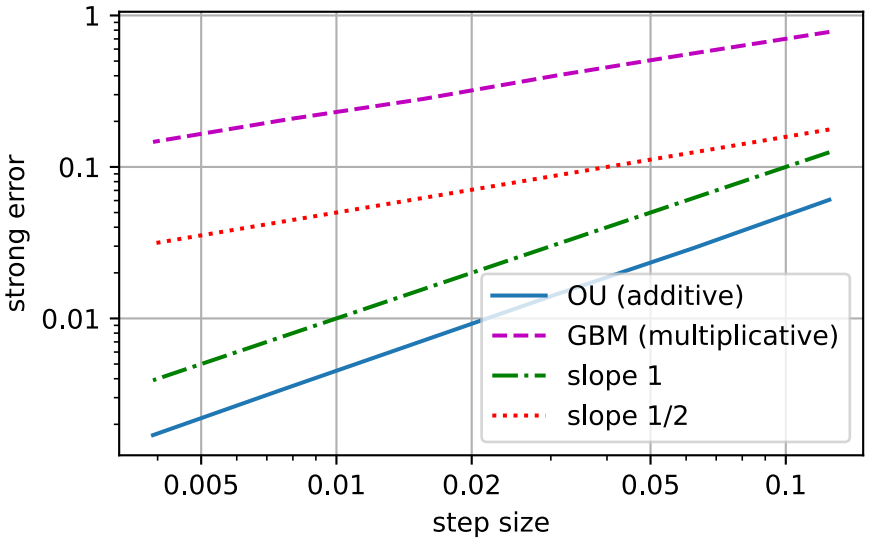
加法的ノイズでは $\partial_x g = 0$ となるため、この項自体が消える。その次に省略される確率的な項は 1 ステップあたり大きさ $\Delta t^{3/2}$ を持ち、足し合わせると $\sqrt{N}\,\Delta t^{3/2} = O(\Delta t)$ になり、決定論的な剰余項 $N\,\Delta t^2 = O(\Delta t)$ と同じオーダーとなって次数は 1 になる。一方、弱誤差で問題になるのは平均だけである。省略された確率的な項はすべて平均ゼロであり、1 ステップの統計的バイアスはオーダー Δt^2 であるため、ノイズの種類によらず N ステップ全体では $O(\Delta t)$ となる。

同じノイズのもとで両方の強収束次数を測定しよう。幾何ブラウン運動（$\mu = \nu = 1$、$X_0 = 1.5$）はその厳密解 Equation 38.34 と比較し、$X_0 = 2$ からの通し例は 4096 ステップのグリッド上の参照解と比較する。後者の厳密解 Equation 38.38 に含まれるノイズ積分自体が、グリッド上でし

か評価できないためである。粗い増分は連続する細かい増分の和として作るため、どの解像度も同じブラウン経路を見ることになる。

```
T, n_ref, n_em = 1.0, 2**12, 2000
mu_g, nu_g = 1.0, 1.0                       # GBM: multiplicative noise
ou_f, ou_g = lambda x, s: -theta_ou * x, lambda x, s: g_ou
gbm_f, gbm_g = lambda x, s: mu_g * x, lambda x, s: nu_g * x
x0_ou, x0_gbm = np.full(n_em, 2.0), np.full(n_em, 1.5)
dW_f = np.sqrt(T / n_ref) * rng.standard_normal((n_em, n_ref))
t_f = np.linspace(0, T, n_ref + 1)
ou_ref = euler_maruyama(ou_f, ou_g, x0_ou, t_f, dW_f)[:, -1]
gbm_exact = 1.5 * np.exp((mu_g - nu_g**2 / 2) * T + nu_g * dW_f.sum(axis=1))
dts, errs_ou, errs_gbm = [], [], []
for n in [2**k for k in range(3, 9)]:       # 8 to 256 steps
    dW = dW_f.reshape(n_em, n, n_ref // n).sum(axis=2)
    t = np.linspace(0, T, n + 1)
    ou = euler_maruyama(ou_f, ou_g, x0_ou, t, dW)[:, -1]
    gbm = euler_maruyama(gbm_f, gbm_g, x0_gbm, t, dW)[:, -1]
    dts.append(T / n)
    errs_ou.append(np.abs(ou - ou_ref).mean())
    errs_gbm.append(np.abs(gbm - gbm_exact).mean())
print('strong-order slopes:  OU (additive) '
      f'{np.polyfit(np.log(dts), np.log(errs_ou), 1)[0]:.2f},'
      '  GBM (multiplicative) '
      f'{np.polyfit(np.log(dts), np.log(errs_gbm), 1)[0]:.2f}')
d2l.plot(dts, [errs_ou, errs_gbm, np.array(dts), 0.5 * np.sqrt(dts)],
         'step size', 'strong error', xscale='log', yscale='log',
         legend=['OU (additive)', 'GBM (multiplicative)',
                 'slope 1', 'slope 1/2'], figsize=(5, 3))
```

```
strong-order slopes:  OU (additive) 1.03,  GBM (multiplicative) 0.48
```



測定された傾きは、加法的ノイズの OU 過程では 1 に近く、乗法的ノイズの幾何ブラウン運動で

は $\frac{1}{2}$ に近く、定理の 2 つの強収束次数と一致する。2 つの過程はスケールが異なるため曲線の高さ自体は異なるが、比較すべきはその下がり方である。ステップ幅を半分にするたびに、OU 過程の誤差は半分になり、幾何ブラウン運動の誤差は約 $\sqrt{2}$ 分の 1 になる。

弱収束次数の確認にはサンプリングすら不要である。OU 過程に適用したオイラー・丸山法は線形ガウス漸化式になるため、$\hat{X}_n$ がガウス分布に従えば $\hat{X}_{n+1}$ もガウス分布に従う。したがってシミュレーションの周辺分布はあらゆるステップでガウス分布となり、その平均と分散は次の漸化式に従う。

$$m_{n+1} = (1 - \theta\,\Delta t)\, m_n, \qquad v_{n+1} = (1 - \theta\,\Delta t)^2\, v_n + g^2\,\Delta t.$$

1 ステップごとに、平均には $1 - \theta\Delta t$ が掛かるのに対し、厳密解では $e^{-\theta \Delta t}$ が掛かる（0.900 対 0.905、$\Delta t = 0.1$ のとき）。分散の漸化式は等比級数の和 $v_n = g^2 \Delta t\,(1 - a^{2n})/(1 - a^2)$（ただし $a = 1 - \theta\Delta t$、$v_0 = 0$）となり、その極限は先ほど求めた不動点 v_∞ である。$T = 1$、初期値 $X_0 = 2$ として 6 通りのステップ数でこれらの閉形式を評価すれば、傾きを曖昧にする標本誤差を一切伴わずにシミュレーション法則の誤差が求まる。

```
mean_T, var_T = 2 * np.exp(-1), 1 - np.exp(-2)  # exact OU kernel at T = 1
print(f'{"n":>4} {"EM mean":>8} {"EM var":>7}'
      f'   exact {mean_T:.4f}, {var_T:.4f}')
dts, errs = [], []
for n in [10, 20, 40, 80, 160, 320]:
    h = 1 / n
    a_h = 1 - theta_ou * h                  # one-step mean multiplier
    m = 2 * a_h**n                          # EM mean after n steps
    v = g_ou**2 * h * (1 - a_h**(2 * n)) / (1 - a_h**2)   # geometric series
    print(f'{n:4d} {m:8.4f} {v:7.4f}')
    dts.append(h)
    errs.append([abs(m - mean_T), abs(v - var_T)])
s_m, s_v = np.polyfit(np.log(dts), np.log(np.array(errs)), 1)[0]
print(f'weak-order slopes:  mean {s_m:.2f},  variance {s_v:.2f}')
```

```
   n  EM mean  EM var   exact 0.7358, 0.8647
  10   0.6974  0.9247
  20   0.7170  0.8938
  40   0.7265  0.8791
  80   0.7311  0.8718
 160   0.7335  0.8682
 320   0.7346  0.8664
weak-order slopes:  mean 1.01,  variance 1.01
```

10 ステップではシミュレーション法則の平均が 0.697、分散が 0.925 となるのに対し、厳密値は 0.736 と 0.865（先ほどのカーネル $\mathcal{N}(0.736,\, 0.930^2)$）であり、320 ステップでは 0.735 と 0.866 になる。当てはめた傾きはどちらも 1.01 である。すなわち、ステップ幅を半分にするとシミュレーション法則の誤差も半分になり、弱収束次数 1 が確かめられる。

幾何ブラウン運動に対して同じ計算を行っても弱収束次数 1 が得られるため、乗法的ノイズは経路の追従では次数を半分失わせるが、法則の再現では何も失わせない。一方で、一見すると高精度に見えるスキームが誤った方程式へ収束してしまうこともある。ステップの両端でノイズ係数を平均するホイン法（Heun's method）は、伊藤解ではなくストラトノビッチ解へと収束してしまう（演習問題 6）。

シミュレーションは順過程のサンプルを生成し、カーネルは単一の始点からの法則を与える。しかし生成にはそれ以上のものが必要になる。データは単一の点ではない。ノイズ付きデータの周辺分布 p_t はすべての始点にわたってカーネルを平均したものであり、サンプルを通じてしか知り得ない実データに対しては閉形式を持たない。それにもかかわらず、過程を逆向きに走らせるには、単一の始点に対してなら書き下せた $\log p_t$ の勾配が必要になる。そして今のところ、私たちは p_t が従う方程式さえ手にしていないのである。

38.4.5 まとめ

順過程は始点を忘れなければならないが、いかなる決定論的フローにもそれはできないため、ノイズによって駆動される。微小ステップの極限で生き残るノイズの大きさは $\sqrt{\Delta t}$ であり、その増分の二乗を足し合わせると経過時間に一致する（$(dW)^2 = dt$）。そのため dW に関する積分は被積分関数をどこで評価するかに依存し、Itô が選んだ左端点評価によって平均ゼロと等長性 $\mathbb{E}[(\int H\,dW)^2] = \int \mathbb{E}[H^2]\,ds$ が成り立つ。同じ理由から、連鎖律には補正項 $\frac{1}{2} g^2 \phi_{xx}\,dt$ が加わる。これらの道具により、オルンシュタイン・ウーレンベック過程 $dX = -\theta X\,dt + g\,dW$ が解かれる。始点 x_0 が与えられたとき、その分布はあらゆる時刻で平均 $x_0 e^{-\theta t}$、分散 $\frac{g^2}{2\theta}(1 - e^{-2\theta t})$ のガウス分布となり、$g^2 = 2\theta$ と選べば拡散モデルの分散保持順過程になる。閉形式解を持たない方程式は、オイラー法のステップに大きさ $\sqrt{\Delta t}$ のノイズを加えたオイラー・丸山法によってシミュレーションされ、経路を加法的ノイズでは次数 1、乗法的ノイズでは次数 $\frac{1}{2}$ で追跡し、法則をその両方に対して次数 1 で再現する。

38.4.6 演習問題

1. **無限の全変動。** 経路 x の分割 $0 = t_0 < t_1 < \cdots < t_n = t$ に沿った全変動は $V_n = \sum_{i} |x(t_{i+1}) - x(t_i)|$ で定義される。経路ごとの積分 $\int H\,dx$ は、和 $\sum_i H(s_i)\,\Delta x_i$（評価点 $s_i \in [t_i, t_{i+1}]$ は任意）の極限として定義され、すべての連続な H に対して、メッシュ幅が縮むときに V_n が有界にとどまるならば存在する。

    1. $x = W$ とし、$\Delta t_i = t/n$ の一様分割を用いよ。$\mathbb{E}[V_n]$ と $\mathrm{Var}(V_n)$ を計算し（$\mathbb{E}|\xi| = \sqrt{2/\pi}$、ただし $\xi \sim \mathcal{N}(0,1)$ を用いよ）、$V_n \to \infty$ が $n \to \infty$ のとき確率収束の意味で成り立つことを導け。
    2. x を連続とし、任意の分割に対して $V_n \le V$ を満たすとする。メッシュ幅が縮むにつれて $\max_i |\Delta x_i| \to 0$ となることを示し、$\sum_i (\Delta x_i)^2 \le \max_i |\Delta x_i| \cdot V \to 0$、すなわちこのような経路の二次変分がゼロになることを導け。このとき、なぜ Equation 38.27 からブラウン経路の全変動が非有界であることの 2 つ目の証明が得られるのか。
    3. x を有界な全変動を持つ連続経路とする。左端点和 $\sum_i x(t_i)\,\Delta x_i$ と右端点和 $\sum_i x(t_{i+1})\,\Delta x_i$ がともに $\frac{1}{2}\left(x(t)^2 - x(0)^2\right)$ へ収束することを示せ。$x = W$ に対する 38.4.2.2 節の 2 つの極限と比較せよ。ブラウン経路では上の議論のどのステップが破綻するか。

2. [code] **ブラウンスケーリングと到達時刻。**

    1. $c > 0$ に対し $\tilde{W}_t = W_{ct}/\sqrt{c}$ とおく。38.4.1.2 節の性質 1〜4 を確かめ、$\tilde{W}$ がブラウン運動であることを導け。これは $\pm\sqrt{\Delta t}$ ウォークのどの性質を厳密にしたものか。
    2. τ_a を $|W_t|$ が初めて水準 $a > 0$ に達する時刻とする。(a) を用いて τ_a が $a^2 \tau_1$ と同じ分布を持つこと、したがって $\mathbb{E}[\tau_a] = a^2\,\mathbb{E}[\tau_1]$ が成り立つことを示せ。
    3. $\mathbb{E}[\tau_1] = 1$ を認めたうえで（これは $\mathbb{E}[W_\tau^2] = \mathbb{E}[\tau]$ から従い、さらにそれは Equation 38.31 と停止時刻までの伊藤積分の平均がゼロであることから導かれる）、$4{,}000$ 本のランダムウォークを分散 $\Delta t = 10^{-3}$ のガウスステップでシミュレーションし、$\mathbb{E}[\tau_a]$ を $a = 0.5, 1, 2$ に対してモンテカルロ標準誤差とともに推定せよ。推定値はわ

ずかに a^2 を上回る傾向があり、特に小さい a で顕著である。離散ウォークが水準を横切る仕組みからこの統計的バイアスの符号を説明し、それが a と Δt に対してどのようにスケールするかを述べよ。

4. 本文の実験では、半分強の経路が $\pm 2\sqrt{t}$ の内側に 500 個すべてのグリッド時刻でとどまった。(a) を用いて、この割合がグリッド時刻の個数 n のみに依存し、区間の長さにはよらないことを示せ。$n = 50$、500、5000 に対してこの割合を測定せよ。n が 10 倍になるごとに割合はどのくらいの倍率で減少するか。またそのことは極限 $n \to \infty$ について何を示唆しているか。

3. **幾何ブラウン運動と $-\nu^2/2$ 補正。** $dX = \mu X\,dt + \nu X\,dW$ ($X_0 > 0$) と、Equation 38.34 で与えられるその解を考えよ。

   1. $\mathbb{E}\big[e^{\lambda Z}\big] = e^{\lambda^2 v/2}$ ($Z \sim \mathcal{N}(0, v)$) を用いて、$\mathbb{E}[X_t] = X_0 e^{\mu t}$ および $\mathbb{E}[X_t^2] = X_0^2 e^{(2\mu+\nu^2)t}$ を示せ。また、X_t の中央値を求めよ。
   2. $\mu = 0.1$、$\nu = 0.6$ とする。$t \to \infty$ のとき、平均、中央値、および典型的な経路はそれぞれどうなるか。これらの主張が同時に成り立ちうるのはなぜか。
   3. どの ν に対して（固定された $\mu > 0$ のもとで）、中央値は成長しなくなるか。(b) のパラメータについて、`euler_maruyama` を用いて 100 本の経路を $t = 20$ まで $\Delta t = 10^{-2}$ でシミュレーションしてこの条件を確かめよ。終端で X_0 を下回る経路の割合はいくらか。また厳密解 Equation 38.34 はその割合をいくらと予測するか。

   Simo Särkkä と Arno Solin 著『Applied Stochastic Differential Equations』演習問題 4.3 より改編。

4. [code] **強収束次数とミルスタイン補正。** 38.4.4.2 節で説明したとおり、オイラー・丸山法はミルスタイン項 $\frac{1}{2} g\,\partial_x g\,\big((\Delta W)^2 - \Delta t\big)$ を省略しており、この項が消えない限り強収束次数は $\frac{1}{2}$ となる。

   1. Equation 38.34 より、$Y = \log X$ は加法的ノイズを持つ SDE を満たす。その SDE に `euler_maruyama` を適用し（強収束次数の実験の増分 `dW_f` とステップサイズを用いよ）、終端値の指数関数をとって `gbm_exact` に対する強誤差を測定せよ。また、その結果を説明せよ。
   2. 伊藤の補題を用いて、$dX = f(X)\,dt + g(X)\,dW$ ($g > 0$) に対し、変数 $Y = \psi(X)$（ただし $\psi' = 1/g$）が振幅 1 の加法的ノイズを持つことを示し、そのドリフトを求めよ。
   3. ミルスタイン法 (Milstein 1975) はオイラー・丸山法のステップにその補正項を加えた次式である。

$$\hat{X}_{n+1} = \hat{X}_n + f\,\Delta t + g\,\Delta W_n + \tfrac{1}{2}\,g\,\partial_x g\,\big((\Delta W_n)^2 - \Delta t\big), \tag{38.44}$$

   ここで f、g、$\partial_x g$ は $(\hat{X}_n, t_n)$ で評価する。強収束次数の実験における幾何ブラウン運動に対してこのスキームを実装し、同じ増分とステップサイズの上で強収束次数の傾きを測定せよ。$\Delta t = 2^{-8}$ において強誤差は何分の 1 に減少するか。

5. **厳密に見た分散保持。** X_t が、平均ゼロのデータ X_0 に対して Equation 38.40 に従うとする。

   1. データが単位分散を持つと仮定し、$\mathbb{E}[X_t^4] = 3 + \bar{\alpha}_t^2\,\big(\mathbb{E}[X_0^4] - 3\big)$ を示せ。さらにラデマッハデータ $X_0 = \pm 1$ に特殊化し、本文の予測式 $3 - 2\bar{\alpha}_t^2$ を導け。

2. 超過尖度 $\mathbb{E}[X^4]/\mathrm{Var}(X)^2 - 3$ はすべてのガウス分布に対してゼロになる。t に関して、X_t の超過尖度はどのレートで減衰するか。このレートを、条件付き平均 $\mathbb{E}[X_t \mid X_0]$ がデータを忘れるレートと比較せよ。
3. 代わりにデータが分散 s^2 を持つとする。$\mathrm{Var}(X_t)$ を求めよ。$s = 2$ かつ $\theta = 1$ のとき、$\mathrm{Var}(X_T)$ が 1% 以内の誤差で基準分布 $\mathcal{N}(0,1)$ の分散に近づくまでには、過程をどれだけ長く走らせる必要があるか。

6. [code] **伊藤とストラトノビッチ。**●$\int_0^t W\,dW$ に対する左端点和と右端点和を足し合わせると任意の分割上で W_t^2 になるため（38.4.2.2 節）、両者の平均は $\frac{1}{2}W_t^2$ に等しい。この種の和はストラトノビッチ積分 (Stratonovich 1966) を定義し、通常の連鎖律に従う。

   1. 38.2 節のホイン法は、ノイズ項を第 2 の傾きと同様に扱えば SDE へと拡張できる。予測子はオイラー・丸山法のステップ $\tilde{X} = \hat{X}_n + f(\hat{X}_n)\,\Delta t + g(\hat{X}_n)\,\Delta W_n$ であり、修正子は両方の係数を平均する。

$$\hat{X}_{n+1} = \hat{X}_n + \tfrac{1}{2}\left[f(\hat{X}_n) + f(\tilde{X})\right]\Delta t + \tfrac{1}{2}\left[g(\hat{X}_n) + g(\tilde{X})\right]\Delta W_n. \tag{38.45}$$

      強収束次数の実験における幾何ブラウン運動に対し、その増分 `dW_f` とステップサイズを用いてこれを実装せよ。終端値を厳密な伊藤解 Equation 38.34 および $X_0 \exp(\mu T + \nu W_T)$ と比較せよ。このスキームは 2 つのうちどちらに収束するか。

   2. 修正子のノイズ項を $\tilde{X} - \hat{X}_n$ の 1 次まで展開し、$(\Delta W_n)^2 \approx \Delta t$ を用いて、このスキームが修正されたドリフトを持つ伊藤 SDE に対するオイラー・丸山法のように振る舞うことを示せ。そのドリフトを求め、(a) の極限が説明できることを確かめよ。

   3. Equation 38.25 の形を持つすべての SDE に対して、2 つの流儀が同じ解を与えるのはなぜか。

7. **線形 SDE の厳密な遷移。**●連続な係数を持つ線形 SDE $dX = -\theta(t)X\,dt + g(t)\,dW$（$\theta$ と g は連続）を考え、$\Theta(t) = \int_0^t \theta(u)\,du$ とおく。

   1. $\phi(x,t) = e^{\Theta(t)}x$ に伊藤の補題を適用し、X_s が与えられたとき、後の時刻の状態 X_t（時刻 t）が平均 $e^{-(\Theta(t)-\Theta(s))}X_s$、分散 $\int_s^t e^{-2(\Theta(t)-\Theta(u))}\,g(u)^2\,du$ のガウス分布に従うことを示せ。
   2. (a) を VP-SDE Equation 38.41 に特殊化し、任意の長さの遷移が単位分散の状態を単位分散の状態へ写すことを示せ。逆に、(a) の線形 SDE において、$\mathrm{Var}(X_t) = 1$ が任意の $t \geq 0$ に対して（$\mathrm{Var}(X_0) = 1$ であるときは常に）成り立つと仮定し、$g(t)^2 = 2\,\theta(t)$ がすべての t で成り立つことを示せ。
   3. 半線形 SDE $dX = (-\theta X + q(X))\,dt + g\,dW$（定数 $\theta, g > 0$、滑らかな関数 q）を考えよ。指数オイラー・丸山ステップでは、q を値 $q(\hat{X}_n)$ に長さ Δt のステップの間固定し、残りの線形 SDE を (a) の方法で厳密に解く。このステップを書き下せ。それがオイラー・丸山ステップ Equation 38.42 と Δt の 1 次の範囲で一致すること、また q が定数のとき厳密になることを示せ。$q \equiv 0$ のとき、オイラー・丸山法が平均を減衰させるのは $\Delta t < 2/\theta$ の場合に限られる（38.2.3 節）。指数ステップのステップサイズを制限する要因は何か。
   4. 厳密なステップは任意のステップサイズで弱誤差がゼロになる。VP-SDE のドリフトと拡散のどの性質がそれを可能にしているか。拡散モデルの生成サンプラは、ドリフトが $-\frac{1}{2}\beta(t)\,x - \beta(t)\,\nabla \log p_t(x)$ である SDE を時間逆向きにシミュレーションする。これは VP ドリフトに、順方向周辺分布の**スコア** $\nabla \log p_t$（ガウス分布でないデー

タに対しては x の非線形関数）の定数倍を加えたものである。そのステップを厳密にできないのはなぜか。拡散サンプラに対する指数積分器 (Zhang and Chen 2023) は、このサンプラの決定論的な対応物、すなわちスコア項を半分にした同じ 2 つの部分からなるドリフトを持つ ODE（38.5 節）に (c) の考え方を適用する。このようなドリフトのどの部分をステップは厳密に扱い、どの部分を近似するか。

8. [extended] **マルチレベルモンテカルロ法。●●**オイラー・丸山法の経路にわたる平均によって $\mathbb{E}[\varphi(X_T)]$ を推定するとき、誤差の原因はスキームの弱い意味での統計的バイアスとサンプリングノイズの 2 つである。マルチレベルモンテカルロ法 (Michael B. Giles 2008) は異なるステップサイズの経路同士を結合することでコストを削減する手法であり、これは 38.4.4.2 節で挙げた強い意味での精度の用途の 1 つである。本問を通じて、平均二乗強誤差 $\mathbb{E}\big[(\hat{X}_N - X_T)^2\big]^{1/2}$ は同項の強誤差と同じ次数を持つものとする。

   1. 通常のモンテカルロ法は $\varphi(\hat{X}_N)$ を M 本の独立な経路にわたって平均し、そのコストは $MT/\Delta t$ ステップである。弱い意味でのバイアスを $c_1\,\Delta t$、分散を v（対象は $\varphi(\hat{X}_N)$）と仮定せよ。平均二乗誤差を ε^2 以下にするには、オーダー ε^{-3} ステップ（M と Δt を最良に選んだ場合）のコストがかかることを示せ。
   2. P_l をオイラー・丸山法の終端における φ の値（ステップ幅 $h_l = T\,2^{-l}$）とし、$\mathbb{E}[P_L] = \mathbb{E}[P_0] + \sum_{l=1}^{L} \mathbb{E}[P_l - P_{l-1}]$ と書く。各項は M_l 組の独立なペアから推定し、強収束次数の実験と同様に、粗い経路には連続する 2 個ずつの細かい増分の和を用いる。リプシッツ連続な φ に対し、$V_l = \mathrm{Var}(P_l - P_{l-1})$ を平均二乗強誤差によって上から抑えよ。$\varphi(x) = x$ とし、同実験のパラメータにおける OU 過程と幾何ブラウン運動に対して、V_l が h_l に応じてどのようにスケールするかを予測せよ。また、V_l を $l = 1, \dots, 8$ に対して `euler_maruyama` で測定し、細かいレベルにおける傾きを当てはめよ。
   3. ラグランジュ乗数を用いて、総コスト $\sum_l M_l/h_l$ を $\sum_l V_l/M_l \le \varepsilon^2/2$ の制約のもとで最小化せよ。最も細かいレベル L を、バイアスが $\varepsilon/\sqrt{2}$ 以下にとどまるように選べ。$V_l \propto h_l^{r}$ と仮定し、コストがオーダー ε^{-2}（$r = 2$ のとき）、およびオーダー $\varepsilon^{-2}(\log \varepsilon)^2$（$r = 1$ のとき）になることを示せ。
   4. 強収束次数の実験における幾何ブラウン運動の $\mathbb{E}[X_T] = X_0 e^{\mu T}$ に対し、(c) の M_l と測定した V_l を用いて両方の推定量を実装せよ。$\varepsilon \in \{0.2, 0.1, 0.05, 0.02\}$ に対して各推定量が使用する総ステップ数を記録し、数回の反復実行にわたってその誤差を確かめよ。ε^2 とコストの積を ε に対してプロットせよ。さらにオイラー・丸山法の代わりに演習問題 4c のミルスタイン法を用いて同じ実験を行え。`euler_maruyama` はすべての経路全体をメモリに保持するため、大きい M_l はバッチに分けてシミュレーションせよ。
   5. 不連続な $\varphi(x) = \mathbf{1}(x > X_0)$ に対して (b) の測定を繰り返せ。なぜ V_l の減衰は遅くなるのか。また、減衰が遅くなることでコストはどう変化するか。

38.5 フォッカー・プランク方程式と確率フロー

前節では、ノイズで揺さぶりながら粒子を原点へと引き戻す確率微分方程式（stochastic differential equation、SDE）$dX = -X\,dt + \sqrt{2}\,dW$ であるオルンシュタイン・ウーレンベック過程（Ornstein–Uhlenbeck process）をシミュレーションした。実行するたびに異なる経路が描かれ、個々の経路を予測することはできない。それでも、実行を重ねても毎回繰り返されるものがある。10 万個の粒子を $x = 2$ から出発させ、時刻 $t = 0.5$ におけるヒストグラムを見てみよう。どの実行でも、中心が 1.21 付近、標準偏差が 0.80 付近の同じ釣鐘曲線になる。経路のランダム性は平均化によって打ち消し合い、粒子群の密度は決定論的に動く。では、その密度を動かしているのはどの方程式だろうか。

Adriaan Fokker と Max Planck は、定常的な力とランダムな揺さぶりを受ける粒子に対してその方程式を書き下し (Fokker 1914; Planck 1917)、Andrey Kolmogorov はそれがあらゆる拡散過程の密度を支配することを示した (Kolmogorov 1931)。この方程式には 2 つの項がある。ドリフトが密度を運び、ノイズが密度を平滑化する。さらに、この平滑化は輸送、すなわち密度の高い場所から低い場所へと確率を動かすフローとして書き換えることもできる。密度のレベルで見れば、ノイズは決定論的な速度のように振る舞うのである。

この事実は生成モデル（generative model）にとって重要である。拡散モデル（diffusion model）は、ノイズ付加過程を逆向きに実行することでデータを生成する。多くの出発点が同じ終点へと至るため、ノイズを帯びた経路の終点を出発点へと戻す写像は存在しない。しかし、粒子群全体の密度がどのように変化したかはわかっており、密度が逆向きに変化するような過程を構成することはできる。その 1 つは、ノイズを外向きのフローに置き換えて得られる常微分方程式（ordinary differential equation、ODE）であり、その粒子はあらゆる時刻で元の確率過程（stochastic process）と同じヒストグラムを持つ。もう 1 つは再び確率過程であり、ノイズをデータへと変換する。いずれにおいても、ドリフトとノイズの大きさは自由に選べる。しかし 1 つだけ選べない量がある。あらゆる点と時刻において対数密度（log-density）が増加する方向である。

本節では、オルンシュタイン・ウーレンベック過程と、その方向が閉形式で書ける 2 モードの「データ」密度を用いて各ステップを確認する。実データに対してその方向は未知であり、これこそ本節が未解決のまま次節へ委ねる問いである。導出の細部を忘れたとしても、3 つの式だけは心に留めておいてほしい。フォッカー・プランク方程式（Fokker–Planck equation）Equation 38.52、その平滑化項を輸送へと書き換える恒等式 Equation 38.63、そして逆過程（reverse process）の族 Equation 38.71 である。発散定理（divergence theorem）に慣れ親しんでいる読者は 38.5.2.1 節を飛ばしてよく、末尾の尤度（likelihood）の公式とバイアスの計算は再読時に回してもよい。

38.5.1 経路から密度へ

38.5.1.1 経路と周辺分布

本節で扱うすべての SDE は次の形をとる。

$$d\mathbf{X} = \mathbf{f}(\mathbf{X}, t)\, dt + g(t)\, d\mathbf{W},$$

ここで、ドリフトは $\mathbf{f} : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$、スカラーのノイズ振幅は $g(t)$、d 次元のブラウン運動（Brownian motion）は $\mathbf{W}$、ランダムな初期値は $\mathbf{X}_0 \sim p_0$ である。短いステップのあいだに、粒子は $\mathbf{f}\,dt$ に加えて、各座標で分散 $g^2\,dt$ の独立なガウス分布に従う揺さぶりの分だけ移動する。

ブラウン運動の 1 つの実現値から 1 本の経路 $\mathbf{X}_t$ が生じ、それは実行するたびに異なる。本節の対象は経路ではなく、密度 $p_t(\mathbf{x})$（確率変数 $\mathbf{X}_t$ の密度）であり、これを**時間周辺分布**（time-marginal）と呼ぶ。これは $\mathbf{X}_0$ の抽出と途中のノイズという 2 つのランダム性の源について平均をとったものであり、この平均化によって決定論的な対象となる。

通し例として用いる過程は 38.4.3 節のオルンシュタイン・ウーレンベック（OU）過程 $dX = -\theta X\, dt + g\, dW$ である。1 点 $X_0 = x_0$ から出発したとき、その周辺分布はガウスカーネルとなり、次のように書ける。

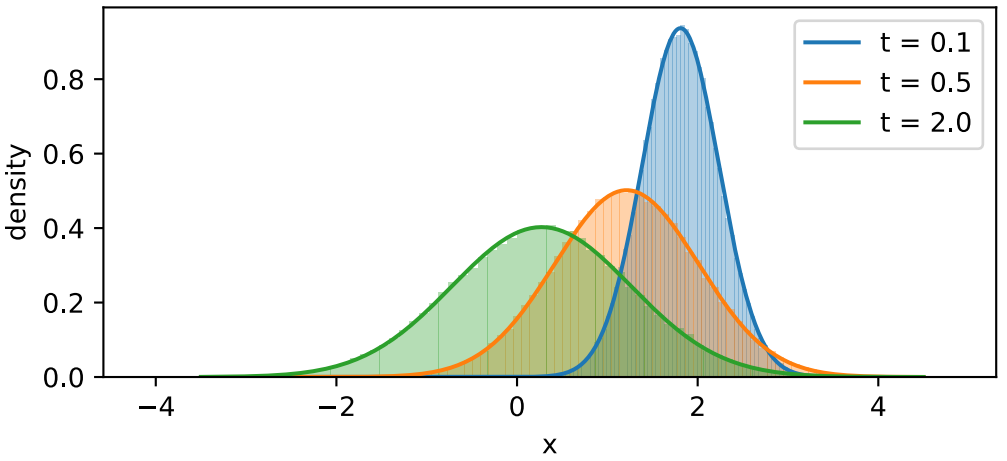
$$X_t \sim \mathcal{N}\!\left(\alpha_t x_0,\ \sigma_t^2\right), \qquad \alpha_t = e^{-\theta t}, \quad \sigma_t^2 = \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right). \tag{38.46}$$

本節全体を通して $\theta = 1$ および $g = \sqrt{2}$ とおく。このとき分散は $g^2/(2\theta) = 1$ で飽和し、定常分布は $\mathcal{N}(0, 1)$ となる。ドリフトは粒子群の中心を動かし、ノイズは粒子群を広げる。フォッカー・プランク方程式は、これら両方の作用を p_t 上の作用素へと変換する。

38.5.1.2 シミュレーションによる周辺分布

SDE の節では、このような粒子群のモーメントをカーネルと比較した。ここではヒストグラム全体を見る。100,000 個の粒子が $x_0 = 2$ から出発してステップ幅 $\Delta t = 0.002$ のオイラー・丸山法（Euler–Maruyama method、38.4.4.1 節）のステップで動き、時刻 $t = 0.1$、0.5、2 において粒子群を厳密なガウス密度と重ねて描く。

```
theta, g = 1.0, np.sqrt(2.0)            # OU: dX = -theta X dt + g dW

def ou_mean_var(x0, t, theta=theta, g=g):
    """Mean and variance of X_t started at x0: N(alpha_t x0, sigma_t^2)."""
    alpha = np.exp(-theta * t)
    return alpha * x0, g**2 / (2 * theta) * (1 - alpha**2)

rng = np.random.default_rng()
n, dt, snap_at = 100_000, 0.002, {50: 0.1, 250: 0.5, 1000: 2.0}  # step -> t
x, snaps = np.full(n, 2.0), {}          # every particle starts at x0 = 2
for k in range(1, max(snap_at) + 1):    # Euler--Maruyama
    x += -theta * x * dt + g * np.sqrt(dt) * rng.standard_normal(n)
    if k in snap_at:
        snaps[snap_at[k]] = x.copy()

grid = np.linspace(-3.5, 4.5, 400)
d2l.set_figsize((6, 2.5))
for (t, cloud), c in zip(snaps.items(), ('C0', 'C1', 'C2')):
    m, v = ou_mean_var(2.0, t)
    d2l.plt.hist(cloud, bins=80, density=True, alpha=0.35, color=c)
    d2l.plt.plot(grid, np.exp(-(grid - m)**2 / (2 * v)) / np.sqrt(2 * np.pi * v),
                 color=c, label=f't = {t}')
d2l.plt.xlabel('x'); d2l.plt.ylabel('density'); d2l.plt.legend();
```



3 つの時刻のいずれにおいても、ヒストグラムは曲線の上に重なっている。Equation 38.46 により、これらの曲線は平均 $2e^{-t}$、標準偏差 $\sqrt{1 - e^{-2t}}$ を持ち、粒子群に当てはめたものではなく公式から直接描いたものである。$t = 2$ においても平均はまだ $2e^{-2} \approx 0.27$ であり、標準偏差のおよそ 4 分の 1 だけ定常分布 $\mathcal{N}(0, 1)$ から離れている。初期位置を忘れるには時間がかかるので

ある。これは 1 回の実行結果にすぎないが、一致している 3 本の曲線にはランダム性が一切含まれていない。2 回目の実行ではすべての経路が異なるにもかかわらず、同じ曲線に一致するはずである。いよいよ、これらの曲線を動かす方程式を見つけよう。

38.5.2 フォッカー・プランク方程式

個々の経路は時間で微分できないが、$\mathbf{X}_t$ の滑らかな関数について全経路にわたる平均をとれば時間で微分できる。平均に関する記述を密度に関する記述へと書き換えるには、導関数を関数側から密度側へと移す必要があり、そのためには $\mathbb{R}^d$ 上の場に関する 3 つの事実が必要になる。

38.5.2.1 発散、ラプラシアン、部分積分

これら 3 つはいずれも 34.2 節の勾配から構成され、それぞれ後で再び必要となる物理的な意味を持つ。滑らかなベクトル場（vector field）$\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$ を流体の速度として思い描き、$\psi : \mathbb{R}^d \to \mathbb{R}$ を滑らかなスカラー場とする。

発散。単位時間あたりに小さな箱からどれだけの流体が流出するかを知りたい。$\mathbf{u}$ の**発散**（divergence）とは、次のスカラー場である。

$$\nabla \cdot \mathbf{u} = \sum_{i=1}^{d} \frac{\partial u_i}{\partial x_i}. \tag{38.47}$$

これは $\mathbf{x}$ のまわりの無限小の箱から流体が流出する正味の流量をその箱の体積で割ったものであり、湧き出し（ソース）では正、吸い込み（シンク）では負になる。平面上の 2 つの場がその典型例を示す。膨張する場 $\mathbf{u} = (x_1, x_2)$ では $\nabla \cdot \mathbf{u} = 1 + 1 = 2$ となるため、面積 A のどの箱も場所によらず $2A$ の割合で流体を失う。一方、回転 $\mathbf{u} = (-x_2, x_1)$ では $\nabla \cdot \mathbf{u} = 0 + 0 = 0$ であり、流体は循環するのみで、どの箱も流体を増減させない。したがって、発散がゼロであることは場そのものがゼロであることを意味しない。Figure 38.17 では 1 つの箱の上でこれら両方の性質を確かめている。

多数の小さな箱からの流出を足し合わせると、内部で隣接するすべての面での出入りが打ち消し合い、外側の境界を通過する流束だけが残る。これが**発散定理**である。有界領域 Ω とその外向き単位法線ベクトル $\mathbf{n}$ に対して、次が成り立つ。

$$\int_{\Omega} \nabla \cdot \mathbf{u} \, d\mathbf{x} = \oint_{\partial\Omega} \mathbf{u} \cdot \mathbf{n} \, dS. \tag{38.48}$$

領域内部の発散の総和は、その境界を通って外へ出る正味の流量に等しい。1 次元では、これは微分積分学の基本定理（fundamental theorem of calculus）$\int_a^b u'(x)\,dx = u(b) - u(a)$ にほかならない。本節ではこの定理を証明なしに用いるが、多変数微分積分のどの教科書にも証明が載っている。

ラプラシアン。関数がその周囲の値と比べてどうなっているかを測る量がほしい。**ラプラシアン**とは勾配の発散である（関数の前に置かれた Δ はこの作用素を表し、Δt のような増分ではない）。

$$\Delta \psi = \nabla \cdot (\nabla \psi) = \sum_{i=1}^{d} \frac{\partial^2 \psi}{\partial x_i^2}. \tag{38.49}$$

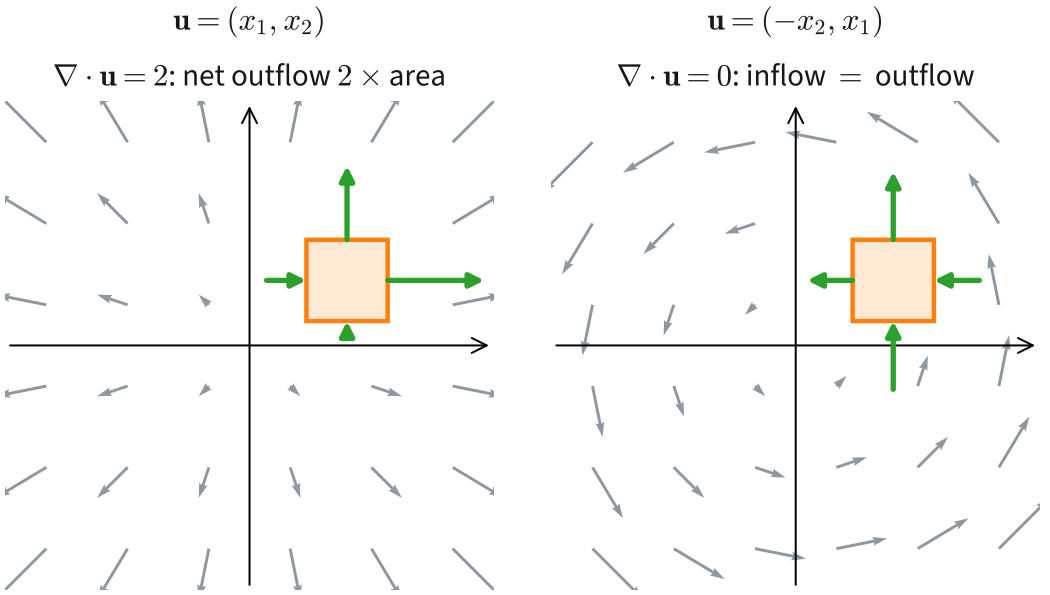


Figure 38.17: 一辺 0.5、中心 $(0.6, 0.4)$ の箱における正味の流出としての発散。緑の各矢印は辺の中点における場の法線成分を表し、流出は辺から離れる向きに、流入は辺に向かう向きに描かれている。(a) $\mathbf{u} = (x_1, x_2)$ では、流体は左辺と下辺から入り、右辺と上辺から出ていく。流出量は流入量を面積の $2\times$ だけ上回る。(b) 回転 $\mathbf{u} = (-x_2, x_1)$ では、右辺と下辺からの流入量が残り 2 辺からの流出量と等しい。

これはある点における ψ の値を、その点のまわりの半径 r の小さな球面上の平均と比較するものである。その平均は $\psi(\mathbf{x}) + \frac{r^2}{2d}\Delta\psi(\mathbf{x}) + O(r^4)$ となり、このことは演習問題 1 で証明する。1 次元で $\psi(x) = x^2$ とすると、「球面」は 2 点 $x \pm r$ の組であり、平均は $\frac{1}{2}\big((x-r)^2 + (x+r)^2\big) = x^2 + r^2$、また $\frac{r^2}{2}\psi'' = r^2$ となるため、この公式は剰余項なしで成り立つ（$x = 1$、$r = 0.3$ では平均は 1.09 である）。したがって $\Delta\psi > 0$ は、ψ が近傍の平均より下にあることを意味する。ラプラシアンがゼロであっても定数関数とは限らない。たとえば $\psi = x_1^2 - x_2^2$ は $\Delta\psi = 2 - 2 = 0$ を満たし、あらゆる円周上の平均と一致する。

部分積分。これは d 次元版の**部分積分**（integration by parts）であり、34.6.2.3 節の 1 次元の規則を拡張して、導関数を一方の因子から他方の因子へと移す道具である。ψ と $\mathbf{u}$ が滑らかであり、$\psi\,\mathbf{u}$ が無限遠で十分に速く減衰するならば（本節では、ψ が有界集合の外でゼロになるか、両因子がガウス分布の裾を持つために成り立つ）、次が成り立つ。

$$\int \psi\,(\nabla \cdot \mathbf{u})\,d\mathbf{x} = -\int \nabla\psi \cdot \mathbf{u}\,d\mathbf{x}. \tag{38.50}$$

導関数は $\mathbf{u}$ から ψ へと移り、符号が反転する。

証明積の微分法則より $\nabla\cdot(\psi\mathbf{u}) = \nabla\psi\cdot\mathbf{u} + \psi\nabla\cdot\mathbf{u}$ が得られる。両辺を半径 R の球体上で積分し、$R \to \infty$ とする。発散定理により、左辺は $\psi\mathbf{u}$ が半径 R の球面を通過する流束となり、減衰の仮定のもとでゼロに収束する。$\blacksquare$

まず $\mathbf{u} = \nabla p$ として適用し、次に役割を入れ替えてもう一度適用すれば、ラプラシアン全体が積分記号の向こう側へと移り、2 回の符号反転が打ち消し合う。すなわち $\int \psi\,\Delta p = -\int \nabla\psi\cdot\nabla p = \int p\,\Delta\psi$ となる。

38.5.2.2　伊藤の補題から PDE へ

ここで p_t に対する方程式を導出しよう。以下では一貫して、$\mathbf{f}$ が $\mathbf{x}$ について連続微分可能かつ t について連続であり、g が連続であり、p_t が滑らかかつ正であり、さらに t に関する導関数を積分記号の下でとれるものと仮定する。

基本的な着想は、平均を通して粒子群を観察することである。ϕ を、小さな箱の上で 1 に等しく、それより少し大きい箱の外で 0 になる滑らかなバンプ関数とすれば、$\mathbb{E}[\phi(\mathbf{X}_t)]$ はおおよそその箱の確率になる。このような平均のすべてが時間とともにどう変化するかがわかれば、密度がどう変化するかも定まる。そこで、有界集合の外でゼロになる任意の滑らかな**テスト関数**（test function）$\phi : \mathbb{R}^d \to \mathbb{R}$ をとる。

伊藤の補題（Itô's lemma、38.4.2.4 節）は SDE の経路に対する連鎖律（chain rule）である。$d\mathbf{X} = \mathbf{f}\,dt + \mathbf{G}\,d\mathbf{W}$ に対して、通常の連鎖律に 2 次の項 $\frac{1}{2}\sum_{ij}(\mathbf{G}\mathbf{G}^{\top})_{ij}\,\partial_i\partial_j\phi$ が加わる。ブラウン運動の増分の二乗 $(dW)^2$ が無視できる大きさではなく dt のオーダーを持つからである。$\mathbf{G} = g(t)\,\mathbf{I}$ のとき、この項は $\frac{1}{2}g^2$ と ϕ のラプラシアンの積になる。

$$d\phi(\mathbf{X}_t) = \big(\nabla\phi\cdot\mathbf{f} + \tfrac{1}{2}g^2\Delta\phi\big)\,dt + g\,\nabla\phi\cdot d\mathbf{W}.$$

関数は、通常の連鎖律にラプラシアン補正を加えたドリフト部分と、ノイズ部分によって変化する。ここで期待値をとろう。$g\nabla\phi$ の伊藤積分（Itô integral）は平均がゼロである。各ブラウン増分がその区間の始点における被積分関数と独立だからである（38.4.2.2 節）。したがってノイズ部分は消える。

$$\frac{d}{dt}\,\mathbb{E}[\phi(\mathbf{X}_t)] = \mathbb{E}\big[\nabla\phi\cdot\mathbf{f} + \tfrac{1}{2}g^2\Delta\phi\big]\,. \tag{38.51}$$

ϕ の平均は、伊藤の補題が与えた変化率の平均の速さで変化する。両辺は周辺密度に対する積分である。$\mathbb{E}[\,\cdot\,] = \int (\cdot)\, p_t(\mathbf{x})\, d\mathbf{x}$ と書き、d/dt を積分記号の下へ移すと次を得る。

$$\int \phi\, \partial_t p_t \, d\mathbf{x} = \int (\nabla\phi\cdot\mathbf{f})\, p_t \, d\mathbf{x} + \tfrac{1}{2}g^2 \int (\Delta\phi)\, p_t \, d\mathbf{x}.$$

右辺の導関数はまだ ϕ に作用しているので、これらを p_t の側へ部分積分 Equation 38.50 によって移す。ϕ は有界集合の外でゼロになるため、境界項は現れず、p_t の減衰条件も不要である。ドリフト項は $\psi = \phi$、$\mathbf{u} = \mathbf{f}p_t$ として 1 回の部分積分を要し、拡散項は 2 回を要して符号が打ち消し合う。

$$\int (\nabla\phi\cdot\mathbf{f})\, p_t \, d\mathbf{x} = -\int \phi\, \nabla\cdot(\mathbf{f}\, p_t)\, d\mathbf{x}, \qquad \int (\Delta\phi)\, p_t \, d\mathbf{x} = \int \phi\, \Delta p_t \, d\mathbf{x}.$$

両者を代入すると、ϕ に対する 1 つの積分にまとまる。

$$\int \phi \left(\partial_t p_t + \nabla\cdot(\mathbf{f}\, p_t) - \tfrac{1}{2}g^2 \Delta p_t \right) d\mathbf{x} = 0.$$

これは任意のテスト関数 ϕ に対して成り立つ。括弧内の関数は連続であり、あらゆるバンプ関数との積分がゼロになる連続関数は恒等的にゼロである（もし正になる点があれば、そこにバンプ関数を置くことで積分が正になってしまう）。したがって括弧内は各点でゼロとなり、p_t は**フォッカー・プランク方程式**（**コルモゴロフ前進方程式**（Kolmogorov forward equation）とも呼ばれる）を満たす。

$$\partial_t p_t(\mathbf{x}) = -\nabla\cdot\big(\mathbf{f}(\mathbf{x},t)\, p_t(\mathbf{x})\big) + \tfrac{1}{2}g(t)^2\, \Delta p_t(\mathbf{x}). \tag{38.52}$$

第 1 項はドリフトの寄与、第 2 項はノイズの寄与である。行列ノイズ $\mathbf{G}\,d\mathbf{W}$ の場合、第 2 項は $\frac{1}{2}\sum_{ij}\partial_i\partial_j(a_{ij}p_t)$（ただし $\mathbf{a} = \mathbf{G}\mathbf{G}^\top$）となり、何が変わるかは演習問題 5 で導く。物理学で用いられるこの方程式の多様な変種は Risken (1996) にまとめられている。

この導出が示しているのは片方向のみ、すなわち SDE の周辺分布がこの方程式を満たすということだけである。後ほどその逆、すなわち p_0 から出発する解が周辺分布**そのものである**という事実が必要になり、そのためには方程式の解がただ 1 つでなければならない。本節で用いるすべてのドリフトが満たす緩やかな条件のもとで、この一意性は実際に成り立つ。すなわち、$g > 0$ であり、ドリフトが $\mathbf{x}$ について t に関して一様にリプシッツ連続（$\mathbf{x}$ 方向の傾きがすべての t に共通する 1 つの定数で抑えられること）であればよい。このとき、与えられた初期分布を持つ確率密度（probability density）の中で解は一意に定まる（厳密な主張は Bogachev et al. (2015) を参照）。以下ではこの一意性を証明なしに用い、使う箇所ではその旨を明記する。

38.5.2.3　熱方程式と 2 つの項

ドリフトを切ってみよう。するとフォッカー・プランク方程式は**熱方程式**（heat equation）になる。

$$\mathbf{f} = \mathbf{0}: \qquad \partial_t p_t = \tfrac{1}{2} g^2 \, \Delta p_t. \tag{38.53}$$

密度は、近傍の平均より下にある場所ではどこでも上昇する。一般的な話に進む前に、1 点で確かめておこう。$g = 1$、$d = 1$ とし、$p_t = \mathcal{N}(0, t)$（0 から出発したブラウン運動の分布）をとる。$p = e^{-x^2/(2t)}/\sqrt{2\pi t}$ を直接微分すると、$\partial_t p = p\,(\tfrac{x^2}{2t^2} - \tfrac{1}{2t})$ および $\partial_{xx} p = p\,(\tfrac{x^2}{t^2} - \tfrac{1}{t})$ が得られる。$x = 0$、$t = 1$ では $p = 0.399$、$\partial_t p = -0.1995$、$\tfrac{1}{2}\partial_{xx} p = -0.1995$ となる。ピークはまさにラプラシアンが予測する速さで沈み込んでいく。

より一般に、ガウス分布 $\mathcal{N}(\mathbf{m}, v_t \mathbf{I})$ は、分散が $v_t = v_0 + g^2 t$（定数 g の場合）あるいは $v_t = v_0 + \int_0^t g(\tau)^2 \, d\tau$（$g$ が時間変化する場合）に従って増大するとき熱方程式の解となる。これはブラウン運動の $\mathrm{Var}(W_t) = t$ という法則にほかならない。ノイズのみの過程は 38.4.1.1 節の**分散爆発**（variance-exploding）族であり、拡散モデルでは増大する $g(t)$ とともに用いられる。

ここで Equation 38.52 の 2 つの項を読み解こう。第 1 項 $-\nabla \cdot (\mathbf{f} p)$ は**移流**（advection）であり、ドリフトが小さな箱へ運び込む質量より外へ運び出す質量の方が多い場所で密度を減少させる。第 2 項 $\tfrac{1}{2} g^2 \Delta p$ は**拡散**（diffusion）であり、密度が近傍の平均より低い場所では密度を引き上げ、高い場所では引き下げる。これは伊藤の $(dW)^2 = dt$ 補正を密度のレベルで見たものにほかならない。

38.5.2.4 OU 方程式のガウス解

OU 過程に対しては閉形式の答えがわかっているため、解の族全体にわたって方程式を検証できる。1 次元で $f(x) = -\theta x$ とすると、Equation 38.52 は次のようになる。

$$\partial_t p = \theta \, \partial_x (x \, p) + \tfrac{1}{2} g^2 \, \partial_{xx} p.$$

偏微分方程式（partial differential equation、PDE）の未知量は関数全体であり、手計算で求まることはめったにない。しかし、ここでは幸運に恵まれる。ガウス分布に対しては、この方程式が平均と分散それぞれに対する 2 本の常微分方程式へと帰着するのである。

命題（OU 方程式のガウス解） $p(x,t) = \mathcal{N}(x; m(t), v(t))$ **を、微分可能な平均** $m(t)$ **と分散** $v(t) > 0$ **を持つガウス密度とする。このとき、**p **が OU フォッカー・プランク方程式を満たすための必要十分条件は次が成り立つことである。**

$$\dot{m} = -\theta m, \qquad \dot{v} = g^2 - 2\theta v. \tag{38.54}$$

証明ガウス密度に対して $\partial_x p = -\tfrac{x-m}{v}\, p$ が成り立つ。直接微分すると、PDE の両辺はそれぞれ p と $(x-m)$ の二次多項式との積になり、まず左辺は次のようになる。

$$\partial_t p = p\left[\frac{\dot{m}}{v}(x-m) + \frac{\dot{v}}{2v^2}\big((x-m)^2 - v\big)\right],$$

一方、ドリフト項で $x = (x-m) + m$ と書けば、右辺は次のようになる。

$$\theta\,\partial_x(xp) + \tfrac{g^2}{2}\partial_{xx} p = p\left[-\frac{\theta m}{v}(x-m) + \Big(\frac{g^2}{2v^2} - \frac{\theta}{v}\Big)\big((x-m)^2 - v\big)\right].$$

関数 $1, (x-m), (x-m)^2$ は線形独立であるため、すべての x で両辺が一致するのは各係数が一致するときに限られる。1 次の項から $\dot{m} = -\theta m$ が、2 次の項から $\dot{v} = g^2 - 2\theta v$ が得られる（定数項からも後者と同じ式が出る）。■

平均は減衰し、分散は $g^2/(2\theta)$ へと緩和する。初期条件 $m(0) = m_0$、$v(0) = v_0$ からこの 2 本の ODE を解くと次を得る。

$$m(t) = m_0 e^{-\theta t}, \qquad v(t) = v_0 e^{-2\theta t} + \frac{g^2}{2\theta}(1 - e^{-2\theta t}).$$

1 点から出発する場合（$v_0 = 0$）、これはカーネル Equation 38.46 にすべての $t > 0$ で一致するため、38.4.3 節の遷移密度（transition density）はフォッカー・プランク方程式を満たす。

不動点 $\dot{m} = \dot{v} = 0$ は $m = 0$、$v = g^2/(2\theta)$ であり、定常ガウス分布が PDE の定常状態 $\partial_t p = 0$ に対応する（演習問題 3 ではガウス分布の形を仮定せずにこれを導出する）。また、$\theta = 0$ とおけば熱方程式の $\dot{v} = g^2$ が復元される。

この幸運には理由があり、限界もある。ドリフトが線形であるため、ガウス分布はガウス分布のまま保たれ、最初の 2 つのモーメントが閉じた系をなす。同じ演習問題で扱う二重井戸（double-well）ドリフト $f(x) = x - x^3$ では、$\mathbb{E}[X^2]$ の方程式に $\mathbb{E}[X^4]$ が入り、$\mathbb{E}[X^4]$ の方程式に $\mathbb{E}[X^6]$ が入って、この連鎖は決して閉じない。

38.5.2.5 線形性と混合分布

実データもまたガウス分布ではないため、より豊かな構造を持ちながらも閉形式で追跡できる密度が必要である。線形ドリフトのもとでは、混合ガウス分布（Gaussian mixture）がその条件を満たす。本節の通し例として用いる「データ」密度は次のとおりである。

$$p_0 = \tfrac{1}{2}\,\mathcal{N}(-2,\, 0.25) + \tfrac{1}{2}\,\mathcal{N}(2,\, 0.25),$$

これは標準偏差 0.5 のよく離れた 2 つのモードからなる分布である。p_0 からの抽出では、まず成分 $\mathcal{N}(\mu_k, v_k)$ を選び、次にそこから 1 点を抽出する。カーネル Equation 38.46 により、X_t は $\alpha_t X_0$ に独立な確率変数 $\mathcal{N}(0, \sigma_t^2)$ を足したものになる。したがって成分を固定すれば、それは 2 つの独立なガウス変数の和であり、平均 $\alpha_t \mu_k$、分散 $\alpha_t^2 v_k + \sigma_t^2$ を持つ。成分について平均をとれば周辺分布、すなわち時間発展した各成分の混合分布が得られる。$\theta = 1$、$g = \sqrt{2}$ のとき、これは $\mathcal{N}\big(\pm 2e^{-t},\, 1 - 0.75\, e^{-2t}\big)$ の混合分布となり、あらゆる時刻で閉形式で表される。

同じ事実は、p について線形であるフォッカー・プランク方程式からも直接読み取れる。解の混合 $w_1 p_t^{(1)} + w_2 p_t^{(2)}$ もまた解となり、Equation 38.52 の直後で述べた一意性により、それが混合初期分布に対する周辺分布となる。OU 過程のもとでは、各ガウス成分がそれぞれ単独でモーメント ODE Equation 38.54 に従う。

以降の実験はすべてこの周辺分布を用いるため、その累積分布関数（CDF）およびサンプラとあわせてここで一度だけ定義しておく。演習問題で変更できるよう、パラメータは明示的な引数としてある。

```
ws, mus, v0s = np.array([0.5, 0.5]), np.array([-2.0, 2.0]), np.array([0.25, 0.25])
erf = np.vectorize(math.erf)           # the error function, on arrays

def mixture_params(t, mus=mus, v0s=v0s, theta=theta, g=g):
    """Means and variances of the OU-evolved components (moment ODEs)."""
```

```
    alpha = np.exp(-theta * t)
    return alpha * mus, alpha**2 * v0s + g**2 / (2 * theta) * (1 - alpha**2)

def gauss(x, m, v):
    return np.exp(-(x - m)**2 / (2 * v)) / np.sqrt(2 * np.pi * v)

def p_t(x, t, ws=ws, mus=mus, v0s=v0s):
    """Exact marginal density of the mixture at time t."""
    m, v = mixture_params(t, mus, v0s)
    return sum(w * gauss(x, mk, vk) for w, mk, vk in zip(ws, m, v))

def mixture_cdf(x, t, ws=ws, mus=mus, v0s=v0s):
    m, v = mixture_params(t, mus, v0s)
    return sum(w * 0.5 * (1 + erf((x - mk) / np.sqrt(2 * vk)))
               for w, mk, vk in zip(ws, m, v))

def sample_p0(n, rng, ws=ws, mus=mus, v0s=v0s):
    k = rng.choice(len(ws), size=n, p=ws)   # component of each draw
    return mus[k] + np.sqrt(v0s[k]) * rng.standard_normal(n)
```

閉形式をコードに実装したので、PDE に直接代入して確かめられる。グリッド上で $t = 0.25$、1、3 において両辺を評価し、∂_t は時間に関する中心差分で、x の導関数は `np.gradient` で計算する。有限差分（finite difference）では厳密解に対しても残差（residual）が残るため、次のセルでは 2 つ目を 2 倍細かくした 2 種類のグリッドを用いる。これらの 2 次精度ステンシルの打ち切り誤差（truncation error）は 4 分の 1 に減少するはずである。

```
def fp_residual(t, points, eps=1e-5):
    """Largest Fokker--Planck residual on a grid of `points`, and max|dp/dt|."""
    xs = np.linspace(-5.0, 5.0, points)
    dx = xs[1] - xs[0]
    dpdt = (p_t(xs, t + eps) - p_t(xs, t - eps)) / (2 * eps)
    p = p_t(xs, t)
    rhs = (np.gradient(theta * xs * p, dx)                # -d/dx(f p)
           + 0.5 * g**2 * np.gradient(np.gradient(p, dx), dx))
    res = (dpdt - rhs)[5:-5]            # trim the one-sided edge stencils
    return np.abs(res).max(), np.abs(dpdt).max()

for t in (0.25, 1.0, 3.0):
    (r1, dpdt), (r2, _) = fp_residual(t, 4001), fp_residual(t, 8001)
    print(f't={t}: max|dp/dt| = {dpdt:.1e}; max|residual| at 4001, 8001 points: '
          f'{r1:.1e}, {r2:.1e} (ratio {r1 / r2:.1f})')
```

```
t=0.25: max|dp/dt| = 4.4e-01; max|residual| at 4001, 8001 points: 4.4e-06, 1.1e-06 (ratio 4.0)
t=1.0: max|dp/dt| = 1.7e-01; max|residual| at 4001, 8001 points: 6.3e-07, 1.6e-07 (ratio 4.0)
t=3.0: max|dp/dt| = 3.2e-03; max|residual| at 4001, 8001 points: 1.2e-06, 3.0e-07 (ratio 4.0)
```

残差は各時刻で $\partial_t p$ より 3〜5 桁小さく（4×10^{-6} 対 0.44 が $t = 0.25$ での値であり、$t = 3$ では密度がほとんど動かないものの 10^{-6} 対 3×10^{-3} となる）、3 つの時刻すべてにおいて細かいグリッドで収束次数どおり 4 分の 1 に減少している。残っているのは打ち切り誤差にほかならない。理論上の議論にはこの計算は必要なかったが、以降の全実験が依拠するコードに誤りがないことを保証する検算となっている。

38.5.3 スコア関数

フォッカー・プランク方程式の平滑化項は、実は姿を変えた輸送であることが後にわかる。その輸送の速度はただ 1 つの場、すなわち $\log p_t$ の勾配から作られる。恒等式 $\nabla p = p\,\nabla \log p$ によって、この勾配がラプラシアンの内部に現れる（38.5.4 節）。そこから先のすべての構成にこの場が必要となるため、混合分布を主な例としてまずこの場を調べよう。

38.5.3.1 定義と幾何

問いは、点 $\mathbf{x}$ からどの方向へ進めば、より確率の高い点へ最も速く向かえるかということである。密度 p_t の**スコア**（score）がその答えを与える。

$$\mathbf{s}_t(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}). \tag{38.55}$$

スコアはデータ空間上のベクトル場であり、対数密度の上り方向を指す。この勾配は入力 $\mathbf{x}$ に関するものであり、モデルのパラメータに関するものではない。37.5.1 節ではこのデータに関するスコアを導入し、最尤法（maximum likelihood）におけるパラメータに関するスコアと対比した。

$\mathbf{s}_t$ の長さは $\log p_t$ の増加率 $\|\nabla p_t\|/p_t$ である。したがって、スコアが大きくなるのは密度が大きい場所ではなく、裾やモード間の谷のように、密度が小さいにもかかわらず変化している場所である。スコアは p_t の臨界点（critical point）、すなわちモード、極小点、鞍点（saddle point）においてゼロになり、その導関数によってこれらを見分けられる。

まずガウス分布を考えよう。$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mu, v\mathbf{I})$ に対して対数密度は $-\|\mathbf{x}-\mu\|^2/(2v) + \text{const}$ であるから、次が成り立つ。

$$\nabla \log p(\mathbf{x}) = -\frac{\mathbf{x} - \mu}{v}. \tag{38.56}$$

これは 37.5.1 節で見た線形バネであり、平均へ向かって引き戻し、分散が小さいほど硬くなる。$\mu = 0$、$v = 0.25$、$x = 1$ のときその値は -4 となり、単位分散のときの 4 倍の引き戻し力になる。一般の共分散（covariance）に対しては、このバネは $-\Sigma^{-1}(\mathbf{x} - \mu)$ となる。Figure 38.18 に、2 次元における 2 モード密度のスコア場を示す。

38.5.3.2 混合ガウス分布のスコア

ここで扱うデータ密度は混合分布であるため、数値評価も解釈もできる形でそのスコアを求めておく必要がある。$p(x) = \sum_k w_k\, N_k(x)$（重み w_k、成分 $N_k(x) = \mathcal{N}(x; \mu_k, v_k)$）に対し、$\log p$ を直接微分すると次を得る。

$$\nabla \log p(x) = \frac{\sum_k w_k \nabla N_k(x)}{\sum_j w_j N_j(x)} = \sum_k r_k(x) \left(-\frac{x - \mu_k}{v_k} \right), \qquad r_k(x) = \frac{w_k N_k(x)}{\sum_j w_j N_j(x)}. \tag{38.57}$$

混合分布のスコアは、各成分のバネの重み付き平均である。重み $r_k(x)$ は成分 k の**負担率**（responsibility）、すなわち位置 x の点がその成分から生じた事後確率（posterior probability）であり、37.5.1 節と同様である。等しい重み、平均 $\pm\mu$、共通の分散 v_0 を持つ 2 成分の場合、負担率は x のロジスティック関数に帰着し、スコアは次の閉形式を持つ。

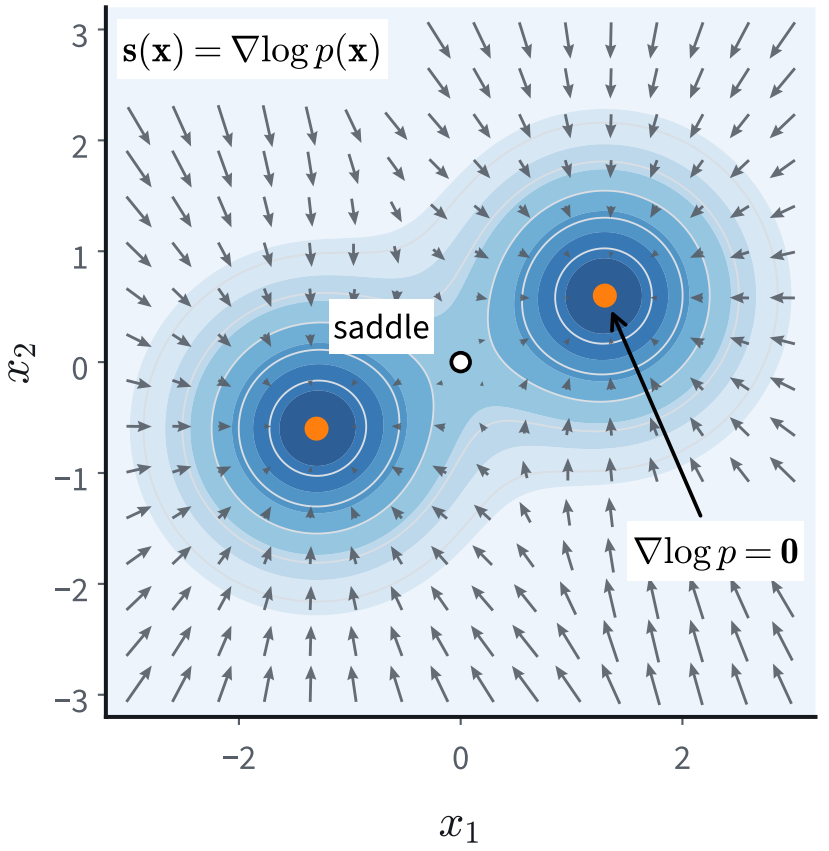


Figure 38.18: 2 成分混合ガウス分布のスコア $\mathbf{s}(\mathbf{x}) = \nabla \log p(\mathbf{x})$ を密度の等高線上に重ねて描いた図。矢印は上り方向を指し、臨界点でゼロになる。オレンジ色の点は 2 つのモードを表し、両者のあいだの対称点も曲率の異なるもう 1 つの零点である。矢印は真の長さ $\|\nabla p\|/p$ を持つが、限りなく増大する遠い裾では上限を 4 で打ち切っており、モードや鞍点の近くでは短くなる。

$$s_0(x) = \frac{\mu \tanh(\mu x / v_0) - x}{v_0}. \tag{38.58}$$

各モードの近くでは $\tanh \to \pm 1$ となり、スコアは局所的なバネ $-(x \mp \mu)/v_0$ に一致する。通し例の混合分布（$\mu = 2$、$v_0 = 0.25$）では $s_0(x) = 8\tanh(8x) - 4x$ となる。$x = 0$ ではスコアがゼロになり、これが 3 つ目の臨界点であって、ゼロから次の傾きで立ち上がる。

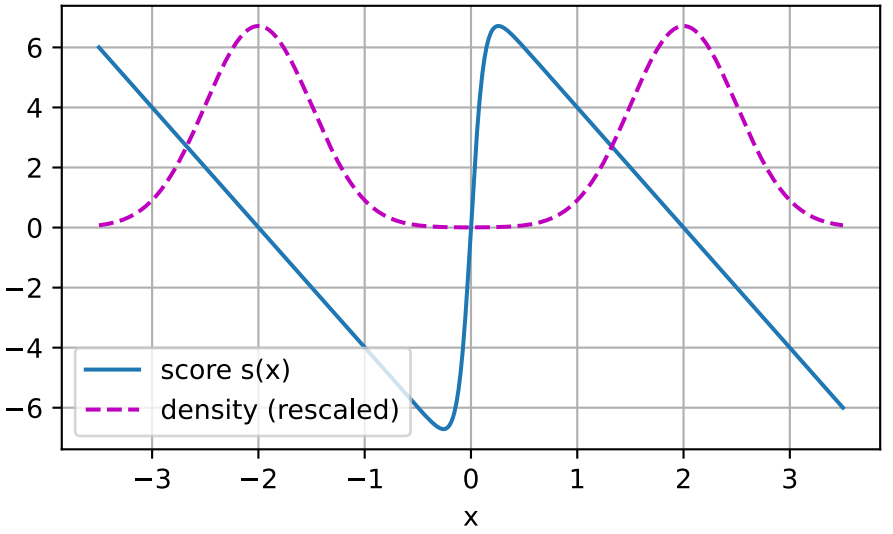
$$s_0'(0) = \frac{\mu^2 - v_0}{v_0^2} = 60.$$

傾きが正であることは、$x = 0$ が密度の極小点であり、上り方向のフロー $\dot{x} = s_0(x)$ の反発点（repeller）であることを意味する。混合分布が 2 つのモードを持つのは $\mu^2 > v_0$ のとき、かつそのときに限り、$\mu^2 \le v_0$ では 2 つのモードは 1 つに合流する（演習問題 4）。

コードでは一般式 Equation 38.57 を用い、負担率の計算には数値的に安定なソフトマックスを使う。後の実験であらゆる時刻における p_t のスコアが必要になるからである。tanh の式、零点、および傾き 60 は実装の検算として役立つ。

```
def score_t(x, t, ws=ws, mus=mus, v0s=v0s):
    """Exact score of p_t: responsibility-weighted component springs."""
    m, v = mixture_params(t, mus, v0s)
    logw = np.stack([np.log(w) - 0.5 * np.log(2 * np.pi * vk)
                     - (x - mk)**2 / (2 * vk) for w, mk, vk in zip(ws, m, v)])
    r = np.exp(logw - logw.max(axis=0))
    r /= r.sum(axis=0)                  # responsibilities, a stable softmax
    return sum(rk * -(x - mk) / vk for rk, mk, vk in zip(r, m, v))

gs = np.linspace(-3.5, 3.5, 1400)
mu, v0 = mus[1], v0s[1]
s = score_t(gs, 0.0)
s_tanh = (mu * np.tanh(mu * gs / v0) - gs) / v0
print(f'max|responsibility form - tanh form| = {np.abs(s - s_tanh).max():.1e}')
mid = (gs[:-1] + gs[1:]) / 2
print('score changes sign near', np.round(mid[s[:-1] * s[1:] < 0], 2) + 0.0)
delta = 1e-5                            # centered difference at x = 0
print(f'slope at 0: {(score_t(delta, 0.0) - score_t(-delta, 0.0)) / (2 * delta):.2f}, '
      f'formula (mu^2 - v0) / v0^2 = {(mu**2 - v0) / v0**2:.2f}')
dens = p_t(gs, 0.0)
d2l.plot(gs, [s, dens * np.abs(s).max() / dens.max()], 'x', '',
         legend=['score s(x)', 'density (rescaled)'], figsize=(5.5, 3))
```

```
max|responsibility form - tanh form| = 7.3e-15
score changes sign near [-2.  0.  2.]
slope at 0: 60.00, formula (mu^2 - v0) / v0^2 = 60.00
```



検算はすべて通る。2 つの式は機械精度で一致し、零点と傾きも計算どおりの位置にある。図から新たに読み取れるのは、スコアが大きくなる場所である。すなわち、密度がほぼゼロでありながらその対数がどちらのモードへ向かっても急勾配で登っていくモード間の谷と、分布の裾である。

38.5.4 連続の式と確率フロー ODE

フォッカー・プランク方程式はランダムに動く粒子を記述する。では、決定論的に動く粒子を支配するのはどの法則だろうか。ノイズを速度として書き直せば、これら 2 つの法則は一致することがわかる。

38.5.4.1 確率の保存

いったんノイズを忘れよう。すべての粒子が速度場 $\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x}, t)$ に従うとする。このとき粒子群の密度 q_t はどの PDE に従うだろうか。確率は生成も消滅もしないため、固定された領域 Ω の内部にある質量は、その境界を横切って出入りすることによってのみ変化しうる。粒子が境界の微小面を横切る速さは、密度と速度の積になる。この流束密度を**確率流**（probability current）と呼び、$\mathbf{j} = q\,\mathbf{u}$ と書く。これは各点に置いた面を単位面積あたりに横切る確率の流量を表す。領域における質量収支は次のようになる。

$$\frac{d}{dt}\int_{\Omega} q_t \, d\mathbf{x} = -\oint_{\partial\Omega} \mathbf{j}\cdot\mathbf{n}\, dS = -\int_{\Omega} \nabla\cdot(q_t\mathbf{u})\, d\mathbf{x}, \tag{38.59}$$

ここで、発散定理 Equation 38.48 によって境界を通る流束を体積積分へと書き換えた。領域内部の質量は、正味の流出量にマイナスをつけた速さで変化する。Ω は任意であるから被積分関数が各点で一致し、**連続の式**（continuity equation）が得られる。

$$\partial_t q_t + \nabla\cdot(q_t\,\mathbf{u}) = 0. \tag{38.60}$$

ある点における密度は、その点で確率流が発散する速さで減少する。1 次元では、この収支は $\partial_t \int_a^b q\,dx = j(a) - j(b)$、すなわち a から流入する量から b へ流出する量を引いたものになる。この様子を Figure 38.19 に描く。

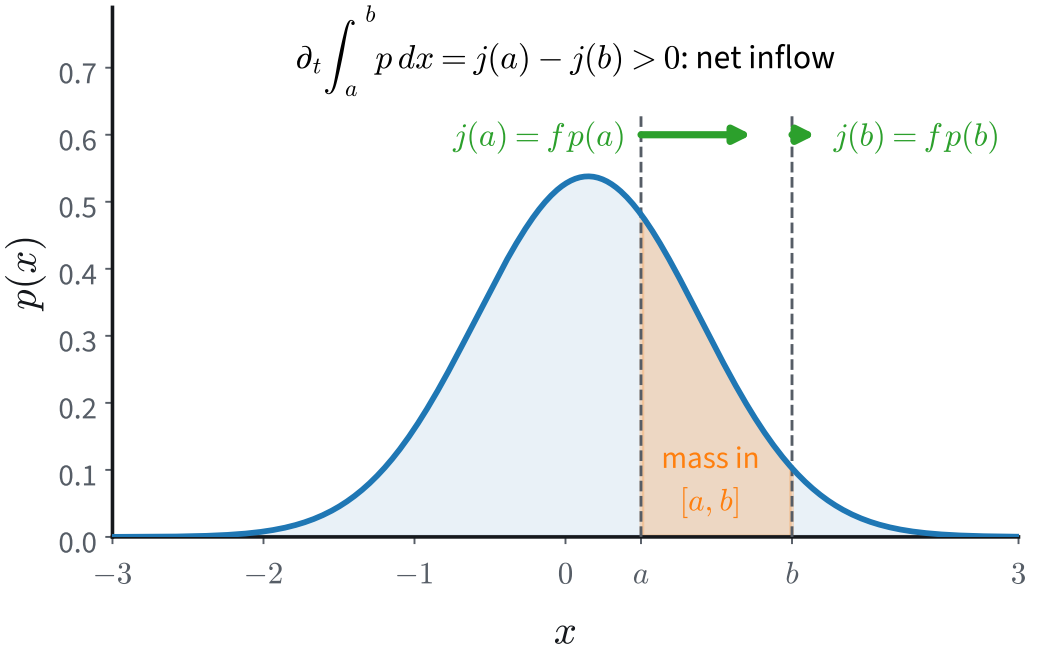


Figure 38.19: 流束の収支としての連続の式。ここでは速度を定数ドリフト f とし、密度を p と書いている。密度 $p(x)$（青）が f によって運ばれ、確率流 $j(x) = f\,p(x)$ を生じる。山の右側斜面にある a から b までの区間では、内部の質量（オレンジ）が正味の流入量 $j(a) - j(b)$ の速さで増加する。a からの流入が b からの流出を上回るのは、a における密度の方が高いからである。

この収支から、定義域の端で何が起こるかもわかる。$\mathbb{R}^d$ 全体で考える場合、確率流は無限遠で減衰すると仮定する。フォッカー・プランク方程式においては、これは $\mathbf{f}p_t$ と ∇p_t が減衰することを意味し、OU 過程のドリフトが線形に増大してもガウス分布の裾があればこの条件は保証される。このとき Equation 38.60 を $\mathbb{R}^d$ 全体で積分すると $\frac{d}{dt}\int q_t = 0$ となり、全確率が保存される。有界領域上では、**反射**（reflecting）境界が $\mathbf{j}\cdot\mathbf{n}=0$ を課すためやはり質量が保存される一方、**吸収**（absorbing）境界は $q=0$ を課すため質量が外へ流出しうる。

1 次元の 2 つの簡単な例で、この方程式の働きを見てみよう。平行移動 $u=c$ の場合、方程式は $\partial_t q + c\,\partial_x q = 0$ となり、解は $q_t(x) = q_0(x-ct)$ である。密度の形状は変わらずにそのまま横滑りする。縮小 $u=-x$ の場合、すべての粒子は $x(t)=x(0)e^{-t}$ に従って動き、粒子群は倍率 e^{-t} で縮み、各区間の質量が保たれるため各軌跡（trajectory）に沿って運ばれる密度は e^{t} 倍に上昇する。

この縮小の例は、運動する粒子から見た密度に対する一般的な規則を示唆している。積の微分法則を用いて発散を展開すると、次のようになる。

$$\nabla\cdot(q\,\mathbf{u}) = \mathbf{u}\cdot\nabla q + q\,\nabla\cdot\mathbf{u},$$

すると連続の式は $\partial_t q + \mathbf{u}\cdot\nabla q = -q\,\nabla\cdot\mathbf{u}$ となる。連鎖律により、左辺は q の、軌跡 $\mathbf{x}(t)$（$\dot{\mathbf{x}}=\mathbf{u}$ の解）に沿った時間変化率である。両辺を q で割れば次を得る。

$$\frac{d}{dt}\log q_t(\mathbf{x}(t)) = -\nabla\cdot\mathbf{u}(\mathbf{x}(t),t). \tag{38.61}$$

軌跡に沿って、対数密度は局所的な膨張率に等しい速さで減少する。先ほどの縮小の例では $\nabla\cdot u=-1$ であり、上で計算したとおり対数密度は速さ 1 で増加する。$\dot{\mathbf{x}}=\mathbf{u}$ の軌跡は、連続の式の**特性曲線**（characteristic）と呼ばれる。この規則は 38.3.3 節で軌跡に沿って証明した瞬間的な変数変換（change of variables）の公式にほかならない。

38.5.4.2　輸送としての拡散

フォッカー・プランク方程式 Equation 38.52 はこのままでは連続の式ではない。輸送項 $-\nabla\cdot(\mathbf{f}p)$ の隣にラプラシアンがあるからである。しかし連鎖律を使えば、このラプラシアンが輸送項へと変わる。わずか 1 行の計算だが、本節で最も重要な帰結をもたらす式変形である。$p>0$ であるすべての点で、次が成り立つ。

$$\nabla p = p\,\nabla\log p, \tag{38.62}$$

これは $\log p$ の導関数を逆向きに読んだ式にほかならない。したがって、次が得られる。

$$\tfrac{1}{2}g^2\,\Delta p = \tfrac{1}{2}g^2\,\nabla\cdot(\nabla p) = \nabla\cdot\big(p\cdot\tfrac{1}{2}g^2\nabla\log p\big). \tag{38.63}$$

拡散項は、密度と $+\frac{1}{2}g^2\nabla\log p$ の積である流束の発散となっている。これを連続の式の左辺へ移項すればマイナス符号がつき、速度 $-\frac{1}{2}g^2\nabla\log p$ として現れる。

命題（フォッカー・プランク方程式は連続の式である） $p_t>0$ **であるすべての点で、フォッカー・プランク方程式 Equation 38.52 は次と同値である。**

$$\partial_t p_t + \nabla\cdot(p_t\,\mathbf{u}_t) = 0, \qquad \mathbf{u}_t(\mathbf{x}) = \mathbf{f}(\mathbf{x},t) - \tfrac{1}{2}g(t)^2\,\nabla\log p_t(\mathbf{x}). \tag{38.64}$$

証明 Equation 38.63 を Equation 38.52 に代入し、2 つの発散を 1 つにまとめる。

$$\begin{aligned}\partial_t p &= -\nabla\cdot(\mathbf{f}p) + \nabla\cdot\big(p\cdot\tfrac{1}{2}g^2\nabla\log p\big)\\ &= -\nabla\cdot\big(p\,[\mathbf{f} - \tfrac{1}{2}g^2\nabla\log p]\big). \qquad \blacksquare\end{aligned}$$

密度のレベルで見ると、SDE はあたかも各粒子が速度 $\mathbf{u}_t$、すなわちドリフトに対数密度勾配を下る向きの押し出しを加えた速度に従っているかのように確率を動かす。対応する確率流は次のとおりである。

$$\mathbf{j} = \mathbf{f}\,p_t - \tfrac{1}{2}g^2\,\nabla p_t = p_t\,\mathbf{u}_t. \tag{38.65}$$

この速度がどのような姿をしているかは、熱方程式を見ればよくわかる。1 次元で $\mathbf{f}=0$、$p_t = \mathcal{N}(0, v_t)$（ただし $v_t = v_0 + g^2 t$）としよう。スコアは $-x/v_t$ であるから、速度は次のようになる。

$$u_t(x) = -\tfrac{1}{2}g^2\Big(-\frac{x}{v_t}\Big) = \frac{g^2}{2v_t}\,x.$$

速度は中心から外向きを指し、中心からの距離に比例し、密度が広がるにつれて弱まる。ODE $\dot{x} = g^2 x/(2v_t)$ は変数分離でき、$\frac{d}{dt}\log x = \frac{g^2}{2v_t} = \frac{1}{2}\frac{d}{dt}\log v_t$ より次を得る。

$$x(t) = x(0)\sqrt{v_t/v_0}. \tag{38.66}$$

すべての粒子が同じ倍率で引き伸ばされ、ガウス分布を $\sqrt{v_t/v_0}$ 倍に引き伸ばせば分散は v_t となって、ブラウン運動が生み出す分散と一致する。$v_0 = g = 1$、$t = 3$ のとき、すべての粒子は中心からの距離が 2 倍になる。一方、$x = 1$ から出発したブラウン粒子は、1 に標準偏差 $\sqrt{3} \approx 1.73$ の乱数を足した位置に着地する。2 つの過程はあらゆる時刻でヒストグラムが一致するが、個々の粒子の動かし方は異なる（Figure 38.20）。

この構成には限界もある。$v_0 \to 0$ とすると、小さい時刻における速度は $g^2 x/(2g^2 t) = x/(2t)$ となり、$t \to 0$ で非有界となる。リプシッツ連続なフローは 1 点を 1 点にしか写せないため、決定論的なフローが点質量をガウス分布へと広げられるのは、出発時点で爆発する場を介する場合に限られる。

次のセルでこの例を実行する。20,000 個の点を $\mathcal{N}(0,1)$ から 1 回抽出し、そこから $t=3$ におけるブラウン粒子群（独立な増分 $\mathcal{N}(0,3)$ を加算）と、$\dot{x} = x/(2(1+t))$ をホイン法（Heun's method、38.2 節）で積分したフロー粒子群を作る。そして両者の標準偏差と、フロー粒子群が初期サンプルを 2 倍したものからどれだけずれているかを表示する。

```
def heat_velocity(x, t, v0=1.0, g=1.0):  # u_t = -(1/2) g^2 score = g^2 x / (2 v_t)
    return g**2 * x / (2 * (v0 + g**2 * t))

n, T, steps = 20_000, 3.0, 600
dt = T / steps
z = rng.standard_normal(n)              # p_0 = N(0, 1)
x_bm = z + np.sqrt(T) * rng.standard_normal(n)   # Brownian motion to time T
x_ode = z.copy()
for k in range(steps):                  # Heun steps for the probability flow
    k1 = heat_velocity(x_ode, k * dt)
```

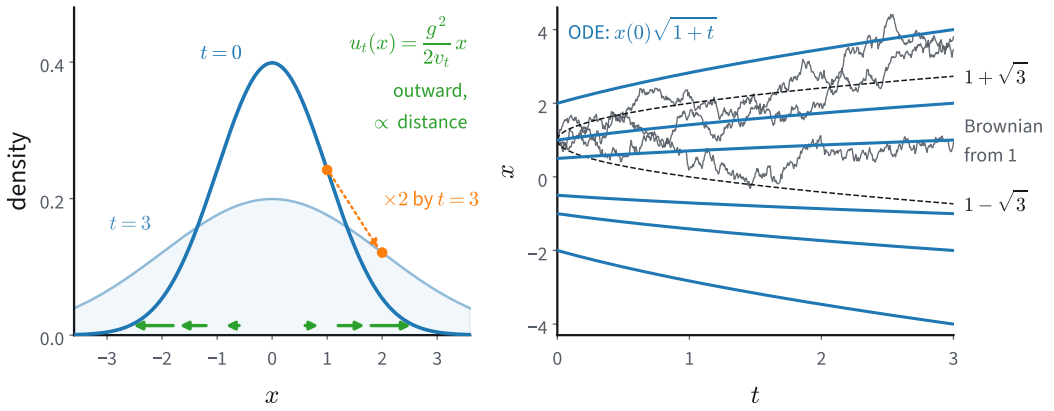


Figure 38.20: 熱方程式 $\partial_t p = \frac{1}{2}\partial_{xx} p$ を初期分布 $p_0 = \mathcal{N}(0,1)$ からフローとして解いた図。(a) $t=0$ と $t=3$ における密度。軸上の矢印は速度 $u_0(x) = x/2$ を表し、外向きで距離に比例する。$x=1$ にある粒子は $x=2$ へと $t=3$ までに移動し、そこでの密度は半分になる。(b) 滑らかで互いに交差しないフローの軌跡 $x(0)\sqrt{1+t}$ と、$x=1$ から出発する 3 本のブラウン経路およびその 1 標準偏差の包絡線 $1 \pm \sqrt{t}$ の比較。

```
    k2 = heat_velocity(x_ode + dt * k1, (k + 1) * dt)
    x_ode += 0.5 * dt * (k1 + k2)
print(f'std at t={T}: Brownian {x_bm.std():.3f}, flow {x_ode.std():.3f}, '
      f'exact {np.sqrt(1 + T):.3f}')
print(f'max|x_ode - 2 z| = {np.abs(x_ode - 2 * z).max():.1e}')
```

```
std at t=3.0: Brownian 2.002, flow 1.986, exact 2.000
max|x_ode - 2 z| = 6.1e-06
```

どちらの粒子群も標準偏差は 2.00（オーダー 0.01 の標本誤差の範囲内）となり、フロー写像（flow map）は $x \mapsto 2x$ に 10^{-5} 未満のホイン法の誤差の範囲で一致する。ランダムな揺さぶりと決定論的な引き伸ばしが、同じヒストグラムを生み出したのである。

38.5.4.3 確率フロー ODE

熱方程式は 1 つの例にすぎない。上の命題によれば、本節で扱う形のあらゆる SDE に対して同じ構成が成り立つ。すなわち、速度 $\mathbf{u}_t$ に従うことで次の ODE が得られる。

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}_t(\mathbf{x}) = \mathbf{f}(\mathbf{x}, t) - \tfrac{1}{2} g(t)^2 \, \nabla \log p_t(\mathbf{x}), \tag{38.67}$$

これが**確率フロー ODE**（probability-flow ODE）である (Y. Song, Sohl-Dickstein, et al. 2021)。次の命題は、生成モデルが本節から受け取る 2 つの主要結果のうちの 1 つ目である。

命題（決定論的な双子） p_t を $d\mathbf{X} = \mathbf{f}\,dt + g\,d\mathbf{W}$（$\mathbf{X}_0 \sim p_0$）の周辺密度とし、滑らかかつ正であってフォッカー・プランク方程式を満たすとする。また、$\mathbf{u}_t$ は t について連続、$\mathbf{x}$ について連続微分可能であり、かつ $\mathbf{x}$ について t に依存しない定数でリプシッツ連続であるとする。$\mathbf{x}(t)$ を Equation 38.67 の $\mathbf{x}(0) \sim p_0$ に対する解とする。このとき、$\mathbf{x}(t) \sim p_t$ がすべての t で成り立つ。すなわち、この ODE と SDE はすべての時間周辺分布を共有する。

証明 38.1.1.3 節の存在と一意性の定理により、t に依存しない定数でリプシッツ連続な場は、順方向にも逆方向にも各点を通る軌跡をただ 1 つ持つため、その時刻 t の写像は $\mathbb{R}^d$ の全単射となる。ODE の状態の密度を q_t とする。ここに循環論法はない。場 $\mathbf{u}_t$ は p から作られており q から作られたものではないため既知の速度場であり、q_t は連続の式 $\partial_t q + \nabla \cdot (q\mathbf{u}_t) = 0$ に従う。前の命題により p_t も同じ方程式を満たし、両者はともに初期分布 $q_0 = p_0$ から出発する。対数密度の規則 Equation 38.61 の導出はこの PDE しか使っていなかったため、各特性曲線に沿って $\log q_t$ と $\log p_t$ は同じ変化率 $-\nabla \cdot \mathbf{u}_t$ で変化し、出発時点では等しい。特性曲線はすべての点に到達するため、いたるところで $q_t = p_t$ が成り立つ。■

このリプシッツ連続の仮定は、OU 過程のドリフトや、正の成分分散を持つガウス分布および混合ガウス分布のスコア（その傾きは t が $[0, T]$ にある限り有界である）に対して成り立つ。

周辺分布は一致するが、経路は一致しない。SDE の経路はギザギザで他の経路と交差する。一方、ODE の軌跡は滑らかであり、解の一意性によって 2 本の軌跡が同じ時刻に同じ点で出会うことは決してない（Figure 38.20 の (b)）。

この ODE からは尤度も得られる。対数密度の規則 Equation 38.61 で $q_t = p_t$ とおき、軌跡に沿って 0 から T まで積分すると次を得る。

$$\log p_0(\mathbf{x}(0)) = \log p_T(\mathbf{x}(T)) + \int_0^T \nabla \cdot \mathbf{u}_t(\mathbf{x}(t))\, dt. \tag{38.68}$$

データ点の対数密度は、ほぼガウス分布に従い容易に評価できる時刻 T での像の対数密度に、途中で累積した膨張率を足したものになる。厳密なスコアの代わりに学習されたスコアを用いれば、これはソルバの誤差を除いて、その場が定義する**モデル**の対数密度となる (Y. Song, Sohl-Dickstein, et al. 2021)。拡散モデルの尤度を報告する際に用いられる 2 つの数値のうちの 1 つがこれであり、もう 1 つは変分境界（variational bound、17.6 節）である。

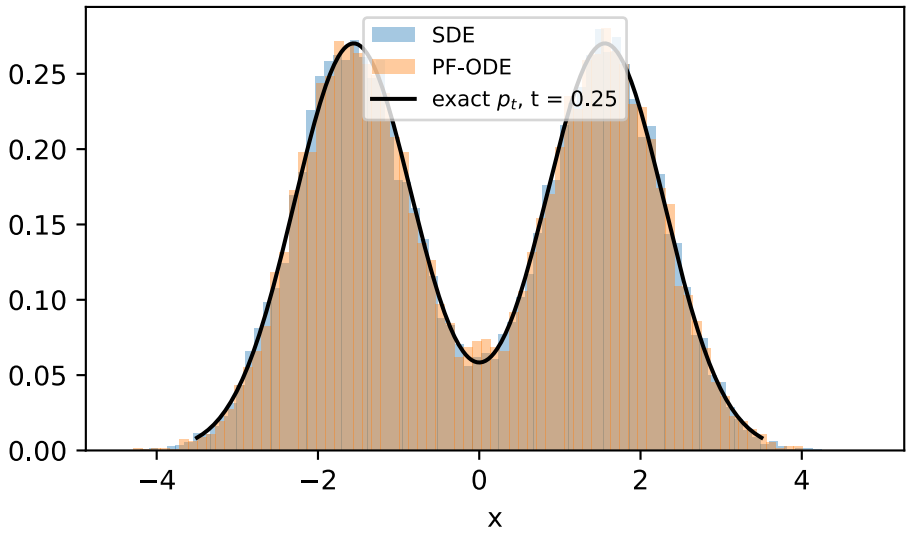
38.5.4.4 SDE と ODE の周辺分布の比較

熱フローは単一のガウス分布の引き伸ばしにすぎなかった。混合分布の上では、この命題にはさらに大きな仕事がある。密度が 2 モードから 1 モードへと形状を変えるなかでも、ギザギザな過程と滑らかな過程はあらゆる時刻で同じ密度を与えなければならない。p_0 からの 1 組のサンプルから、20,000 個の粒子からなる 2 つの粒子群を出発させる。一方はオイラー・丸山法のステップで SDE に従い、もう一方は厳密なスコアを用いたホイン法のステップで確率フロー ODE に従う。物差しとして、各粒子群と p_t の厳密な CDF との 1 標本コルモゴロフ・スミルノフ距離（Kolmogorov–Smirnov distance）、すなわち経験 CDF と厳密な CDF の最大乖離幅を計算する。サイズ n の厳密なサンプルでは、この距離が $1.36/\sqrt{n}$ を超える確率はわずか 5% である。出力を見る前に、スナップショット間でどちらの粒子群の距離がより大きく変動するか予想してみてほしい。

```
def pf_velocity(x, t, score=score_t, theta=theta, g=g):
    return -theta * x - 0.5 * g**2 * score(x, t)   # u_t = f - (1/2) g^2 s_t

def ks_exact(sample, t, **mixture):
    """One-sample Kolmogorov--Smirnov distance to the exact p_t."""
    xs_sorted, n = np.sort(sample), len(sample)
    F = mixture_cdf(xs_sorted, t, **mixture)
    return max((np.arange(1, n + 1) / n - F).max(), (F - np.arange(n) / n).max())

n, T, steps = 20_000, 3.0, 600
```

```
    dt = T / steps
    x_sde = sample_p0(n, rng)
    x_ode = x_sde.copy()                    # both clouds start from one sample
    snaps = {}
    for k in range(steps):
        t = k * dt
        x_sde += -theta * x_sde * dt + g * np.sqrt(dt) * rng.standard_normal(n)
        k1 = pf_velocity(x_ode, t)          # Heun steps for the ODE
        k2 = pf_velocity(x_ode + dt * k1, t + dt)
        x_ode += 0.5 * dt * (k1 + k2)
        if k + 1 in (50, 200, 600):         # t = 0.25, 1, 3
            snaps[(k + 1) * dt] = (x_sde.copy(), x_ode.copy())
    print(f'95% band for n = {n}: {1.36 / np.sqrt(n):.4f}')
    for t, (a, b) in snaps.items():
        print(f't={t:.2f}: KS(SDE) = {ks_exact(a, t):.4f}, KS(ODE) = {ks_exact(b, t):.4f}')
    gx = np.linspace(-3.5, 3.5, 400)
    d2l.set_figsize((5.5, 3))
    d2l.plt.hist(snaps[0.25][0], bins=80, density=True, alpha=0.4, label='SDE')
    d2l.plt.hist(snaps[0.25][1], bins=80, density=True, alpha=0.4, label='PF-ODE')
    d2l.plt.plot(gx, p_t(gx, 0.25), 'k', label='exact $p_t$, t = 0.25')
    d2l.plt.xlabel('x'); d2l.plt.legend(loc='upper center', fontsize=8);
```

```
95% band for n = 20000: 0.0096
t=0.25: KS(SDE) = 0.0061, KS(ODE) = 0.0030
t=1.00: KS(SDE) = 0.0034, KS(ODE) = 0.0030
t=3.00: KS(SDE) = 0.0100, KS(ODE) = 0.0030
```



ほとんどの実行で、両方の粒子群とも 3 つの時刻すべてにおいて 0.0096 の帯域内に収まり、距離は約 0.003 から 0.01 のあいだになる。帯域をわずかに超える値が出ても異常ではない。この帯域は 5% のしきい値であり、厳密なサンプルでも 20 回に 1 回はこれを超えるからである。$t = 0.25$ では周辺分布はまだ二峰性（bimodal）であり、両方のヒストグラムがその 2 つのモードを再現している。$t = 1$ までには単峰になり、モードが合流する時刻は 0.78 である（演習問題 4）。

さて、予想の答え合わせをしよう。ODE の距離は、表示されているすべての桁にわたって 3 つの

時刻すべてで同じ値になっている。p_0 を p_t へと運ぶ狭義単調増加写像はサンプルの経験 CDF もそのまま運ぶため、ODE の粒子群は初期サンプルが持っていた距離をそのまま保ち、そこに加わるのはソルバの誤差だけである。一方、SDE の粒子群はステップごとにノイズによって引き直されるため、距離が変動する。

38.5.5 時間反転

順方向の SDE はデータをノイズへと変える。生成に必要なのは、ノイズからデータへとこの過程を逆再生することである。これは不可能に聞こえるかもしれない。平滑化によって失われた細部はどんな操作でも復元できないため、任意の終端密度から熱方程式を逆向きに解くことはできないからである。しかし、ここで熱方程式を逆向きに解くわけではない。順方向の分布族 p_t はすでに与えられており、それを逆順にたどる過程を求めているだけである（Figure 38.21）。そのドリフトには、失われた細部を保持している $\nabla \log p_t$ が含まれることになる。確率フロー ODE を逆向きに積分したものは、まさにそのような過程の 1 つである。SDE の場合の答えはそれほど自明ではない。逆向きの過程も再び拡散過程となり、そのドリフトは ODE の 2 倍のスコア補正を持つ。

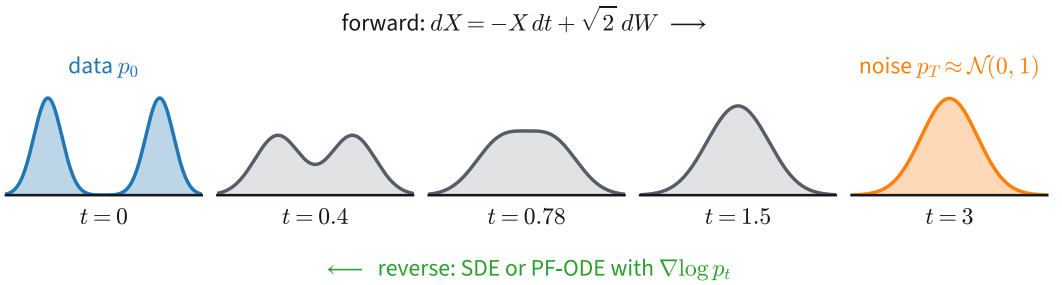


Figure 38.21: 通し例の混合分布 $p_0 = \frac{1}{2}\mathcal{N}(-2, 0.25) + \frac{1}{2}\mathcal{N}(2, 0.25)$ の、$\theta = 1$、$g = \sqrt{2}$ の OU 過程のもとでの順方向の周辺分布（各時刻で厳密）。本節の逆過程は、同じ周辺分布を右から左へとたどる。$t \approx 0.78$ における平坦な頂上は、2 つのモードが合流する地点である。

38.5.5.1 無限小ステップにおけるベイズの定理

微小な 1 ステップを逆転させるため、ベイズの定理を用いて粒子がどこから来たかを問う。短い時間幅 τ にわたる順方向のステップは、オイラー・丸山法のガウスカーネルである（38.4.4.1 節）。

$$p(\mathbf{x}_{t+\tau} \mid \mathbf{x}_t) \approx \mathcal{N}\big(\mathbf{x}_{t+\tau};\ \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\,\tau,\ g^2 \tau\, \mathbf{I}\big).$$

ここで終点 $\mathbf{x}_{t+\tau} = \mathbf{y}$ で条件付けよう。ベイズの定理により、確からしい出発点の分布は、順方向カーネルを時刻 t の周辺分布で重み付けしたものとして与えられる。

$$p(\mathbf{x}_t = \mathbf{x} \mid \mathbf{x}_{t+\tau} = \mathbf{y}) \propto p(\mathbf{y} \mid \mathbf{x})\ p_t(\mathbf{x}).$$

ある出発点が確からしいのは、そこから $\mathbf{y}$ に到達しえた場合、かつそもそもその出発点自体が確からしかった場合である。対数をとり、終点のまわりで $\log p_t(\mathbf{x})$ を展開すると $\log p_t(\mathbf{x}) \approx \log p_t(\mathbf{y}) + (\mathbf{x} - \mathbf{y}) \cdot \nabla \log p_t(\mathbf{y})$ となる。ステップ $\mathbf{x} - \mathbf{y}$ の大きさは $\sqrt{\tau}$ であるため、無視した 2 次の項は τ のオーダーであり、事後分布（posterior）にはより高次の項としてしか影響しない。同じ次数までで考えれば、$\mathbf{f}$ は $\mathbf{y}$ で評価してよい（$\mathbf{x}$ で評価する場合との差 $\mathbf{f}(\mathbf{x}, t) - \mathbf{f}(\mathbf{y}, t)$

は大きさが $\sqrt{\tau}$ であり、そこに τ が掛かるからである）。ここで、ガウス分布の指数部に 1 次の項を加えると、平方完成により分散は変わらないまま平均だけが「分散 × 傾き」の分だけシフトする。いま指数部は $-\|\mathbf{x}-(\mathbf{y}-\mathbf{f}(\mathbf{y},t)\,\tau)\|^2/(2g^2\tau)$、分散は $g^2\tau$、傾きは $\nabla \log p_t(\mathbf{y})$ であるから、次を得る。

$$p(\mathbf{x}\mid \mathbf{y}) \approx \mathcal{N}\big(\mathbf{x};\ \mathbf{y}-[\mathbf{f}(\mathbf{y},t)-g^2\nabla\log p_t(\mathbf{y})]\,\tau,\ g^2\tau\,\mathbf{I}\big). \tag{38.69}$$

逆向きのステップは同じ分散 $g^2\tau$ を持つガウス分布に従う揺さぶりであり、そのドリフトは順方向のドリフトに g^2 とスコアの積を丸ごと加えた補正を受けたものになる。周辺分布 p_t は事前分布（prior）として働き、逆向きのステップをデータ密度があった場所へと引き寄せる。

38.5.5.2 逆時間 SDE

この 1 ステップの計算は発見的な導出である。1 次までで展開し、近似カーネルから出発しているからである。しかし Anderson (1982) は、これが過程全体に対して正しい答えを与えることを証明した。この定理を証明なしに述べよう。拡散過程 $d\mathbf{X}=\mathbf{f}\,dt+g\,d\mathbf{W}$ の時間反転は、次の**逆時間 SDE**（reverse-time SDE）である。

$$d\mathbf{X} = \big[\mathbf{f}(\mathbf{X},t)-g(t)^2\,\nabla_{\mathbf{x}}\log p_t(\mathbf{X})\big]\,dt + g(t)\,d\bar{\mathbf{W}}, \tag{38.70}$$

これを $t=T$ から $t=0$ まで逆向きに積分する。ここで $\bar{\mathbf{W}}$ は時間を逆向きに進むブラウン運動であり、シミュレーションにおいては逆向きの各ステップで独立な新しいガウス増分を加えることを意味する。$\mathbf{X}_T\sim p_T$ から出発すると、この過程はすべての中間時刻で周辺分布 p_t を持ち、$t=0$ においてデータ密度を与える。この定理は逆転された過程全体を特定するものだが、以下ではサンプリングに必要な周辺分布に関する主張のみを証明する。

ステップとして読めば、この式は前項の計算そのものである。$t+\tau$ から t へ戻ることは $dt=-\tau$ のステップをとることにあたり、粒子は $-[\mathbf{f}-g^2\nabla\log p_t]\,\tau$ に分散 $g^2\tau$ の揺さぶりを加えた分だけ動く。これはガウス分布 Equation 38.69 にほかならない。実際の計算には、順方向に進む時計の方が扱いやすい。**逆向きの時計**（reversed clock）$\bar{t}=T-t$ を用いれば、同じ過程はドリフト $-\mathbf{f}+g^2\,\nabla\log p_t$ とノイズ g を持つ通常の SDE になる。時間の向きが変わるとドリフトは符号を変えるが、ノイズは向きを問わない。

最小の例は手計算で確かめられる。定常状態にある OU 過程、すなわち $p_t=\mathcal{N}(0,1)$ がすべての t で成り立つ場合（$\theta=1$、$g=\sqrt{2}$、スコア $-x$）を考えよう。確率フロー速度は $-x-\tfrac{1}{2}\cdot 2\cdot(-x)=0$ となり、ODE の粒子は動かず、粒子群は $\mathcal{N}(0,1)$ のまま保たれる。逆向きの時計では、逆 SDE はドリフト $-f+g^2 s_t = x-2x=-x$ とノイズ $\sqrt{2}$ を持ち、元の OU 過程と同じになる。定常な OU 過程はそれ自身の時間反転なのである。

もし係数を $\tfrac{1}{2}g^2$ にして g^2 の代わりに用いれば、逆ドリフトは $x-x=0$ となり、粒子群は定常分布にとどまるどころかブラウン運動のように広がってしまう。したがって逆 SDE には係数 g^2 丸ごと、すなわち ODE の 2 倍の補正が必要になる。この係数の理由は、サンプラの族を考えることで説明がつく。

命題（周辺分布を共有する 1 パラメータ族） **p_t を、決定論的な双子の命題の仮定および $\mathbf{f}$ が $\mathbf{x}$ についてリプシッツ連続であるという仮定のもとでの順方向 SDE の周辺密度とする。任意の $\lambda\ge 0$ に対し、ノイズ振幅 λg と次のドリフトを持つ逆時間 SDE は、**

$$\mathbf{b}_\lambda = \mathbf{f} - \tfrac{1+\lambda^2}{2}\,g^2\,\nabla\log p_t \tag{38.71}$$

周辺分布 p_t を、初期分布 p_T から出発したときに持つ。$\lambda = 1$ と選べば Anderson の逆 SDE Equation 38.70 になり、$\lambda = 0$ と選べば確率フロー ODE Equation 38.67 になる。

サンプラが注入するノイズが大きいほど、必要なスコア補正も大きくなる。

証明逆向きの時計 $\bar{t} = T - t$ で考え、密度の時間発展を逆再生したものを $q_{\bar{t}} = p_{T-\bar{t}}$ とおくと、$\partial_{\bar{t}} q = -\partial_t p = \nabla \cdot (\mathbf{f} q) - \frac{1}{2} g^2 \Delta q$ となる（ただし $\mathbf{f}$ と g は順方向の時刻 $T - \bar{t}$ で評価する）。一方、逆向きの時計における過程 $d\mathbf{Y} = \mathbf{b}'\, d\bar{t} + \lambda g\, d\mathbf{W}$ は、それ自身のフォッカー・プランク方程式 $\partial_{\bar{t}} q = -\nabla \cdot (q\, \mathbf{b}') + \frac{\lambda^2}{2} g^2 \Delta q$ に従う。両者が一致するためには次が必要である。

$$-\nabla \cdot (q\, \mathbf{b}') = \nabla \cdot (\mathbf{f} q) - \tfrac{1+\lambda^2}{2} g^2 \Delta q = \nabla \cdot \Big(q \Big[\mathbf{f} - \tfrac{1+\lambda^2}{2} g^2 \nabla \log q \Big] \Big),$$

ここで**両方**のラプラシアンを Equation 38.63 によって輸送形へと書き換えた。この式は $\mathbf{b}' = -\mathbf{f} + \frac{1+\lambda^2}{2} g^2 \nabla \log q$ に対して成り立つ。逆向きの時計におけるドリフト $\mathbf{b}'$ は順方向の時刻ではドリフト $-\mathbf{b}'$ となるため、Equation 38.71 が得られる。残るは、発展方程式の一致と初期分布の一致 $q_0 = p_T$ から周辺分布の一致を導く段階であり、ここで一意性が必要になる。$\lambda = 0$ の場合は、決定論的な双子の命題における特性曲線に沿った議論そのものである。$\lambda > 0$ の場合は、Equation 38.52 の直後で証明なしに述べた一意性による。実際、$\mathbf{b}'$ は $\mathbf{f}$ と $\mathbf{u}_t$ がリプシッツ連続であるときリプシッツ連続になるため、この一意性が適用できる。■

この証明から係数 2 の意味も明らかになる。ノイズ λg を持つ逆過程はそれ自身の密度を平滑化してしまうため、そのドリフトは 2 つ分の平滑化を打ち消さなければならない。まず 1 つ目の $\frac{1}{2} g^2 \nabla \log p_t$ が順方向の平滑化を打ち消す。ノイズを一切注入しない確率フロー ODE の補正はこれで全部である。さらに $\frac{\lambda^2}{2} g^2 \nabla \log p_t$ が、逆過程自身のノイズによる平滑化を補償する。Anderson の $\lambda = 1$ はこの半分ずつを両方とも含んでいる。もっとも、周辺分布だけではドリフトは一意に定まらない。同じ周辺分布を持つ他の任意のドリフトは $\mathbf{b}'$ と場 $\mathbf{w}$（ただし $\nabla \cdot (q\mathbf{w}) = 0$）の分だけ異なり、演習問題 7 ではそのような場を構成する。

この自由度は何の役に立つのだろうか。ノイズはサンプラに「忘却」をもたらす。初期分布が p_T からわずかにずれていたり、前のステップで誤差が生じたりして状態の分布に誤差が入ると、ODE はそれを $t = 0$ までそのまま運び切る。写像が全単射であり、何も失わないからである。これに対し SDE のノイズはその誤差の一部を洗い流してくれる（演習問題 10 でこれら両方の主張を厳密に示す）。その代償としてスコア項は大きくなり、ノイズを含む長いステップはそれ自身の誤差を新たに付け加える。拡散モデルのサンプラはこのトレードオフを踏まえて λ を選ぶ（17.5 節）。どの値を選んでもサンプラが必要とする場は同じ $\nabla \log p_t$ であるため、スコアの推定が 1 つあれば族全体を動かせる。

Figure 38.22 に、通し例の混合分布における 3 つの値の λ を示す。決定論的な経路は $\bar{t} \approx 2.2$ 付近、すなわち周辺分布が二峰性でなくなる順方向の時刻 0.78 付近で 2 つのモードへ向かって分岐する。

38.5.5.3 逆時間サンプリング

訓練されたネットワークの代わりに厳密なスコアを使って、実際にデータを生成してみよう。この実験に限って、データの重みを非対称にする。モードの位置と幅はそのままにし、質量の 5 分の 4 を左側へ置く。すなわち $p_0 = 0.8\, \mathcal{N}(-2, 0.25) + 0.2\, \mathcal{N}(2, 0.25)$ とする。対称なノイズから出発するサンプラは、モードの位置だけでなくその比率も正しく再現しなければならず、これは失敗しうる厳しいテストになる。

20,000 個の点を $\mathcal{N}(0, 1)$（p_T の $T = 3$ における代用）からサンプリングし、逆向きの時計でオイラー・丸山法のステップを用いて Anderson の逆 SDE を $t = 0$ まで積分する。次のセルで

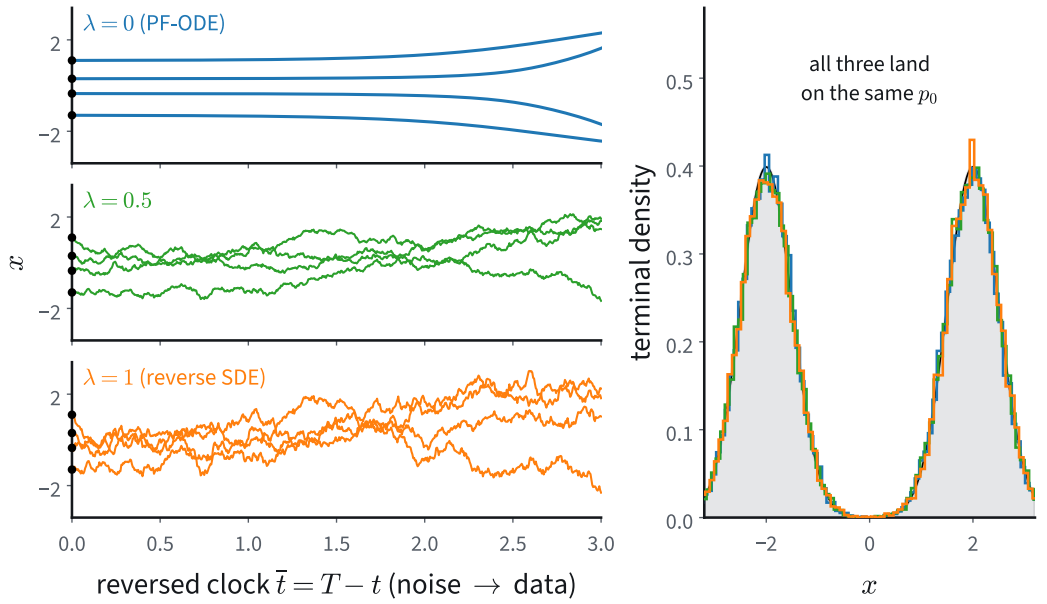
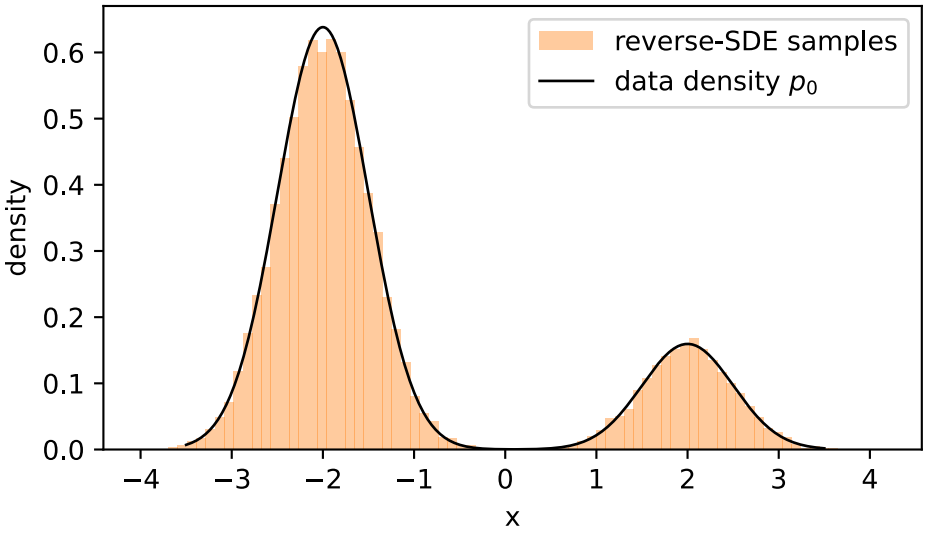


Figure 38.22: $t = T$ における同じ 4 つの出発点からの逆時間軌跡。通し例の混合分布の厳密なスコアを用い、ドリフトを $\mathbf{f} - \frac{1+\lambda^2}{2} g^2 \nabla \log p_t$、ノイズを λg としている。確率フロー ODE（$\lambda = 0$）は決定論的であり、$\lambda = 0.5$ と Anderson の逆 SDE（$\lambda = 1$）はノイズとより大きなスコア補正を加える。$t = 0$ における各 20,000 粒子の 3 つのヒストグラム（右）はいずれもデータ密度 p_0 と一致する。

は、まず p_T が $\mathcal{N}(0,1)$ からどれだけ離れているかを測り、続いて生成サンプルをデータと比較する。すなわち、原点より左側の質量、各モードの平均と幅、および p_0 に対する KS 距離（同じサイズの厳密なサンプルの KS 距離と並べて表示）を確認する。

```
uneven = dict(ws=np.array([0.8, 0.2]))  # four fifths of the data on the left
pT = p_t(gs, 3.0, **uneven)
print(f'max|p_T - N(0,1)| at T=3: {np.abs(pT - gauss(gs, 0.0, 1.0)).max():.4f} '
      f'(peak {pT.max():.2f})')
x = rng.standard_normal(n)              # N(0, 1) stands in for p_T
for k in range(steps):                  # reversed clock: t_bar = k dt
    t = T - k * dt                      # forward time
    x += (theta * x + g**2 * score_t(x, t, **uneven)) * dt \
         + g * np.sqrt(dt) * rng.standard_normal(n)
left, right = x[x < 0], x[x >= 0]
print(f'mass on the left: {len(left) / n:.3f} (data 0.800)')
print(f'means {left.mean():+.3f}, {right.mean():+.3f} (data -2, +2)')
print(f'widths {left.std():.3f}, {right.std():.3f} (data 0.5)')
x_ref = sample_p0(n, rng, **uneven)     # an exact sample of the same size
print(f'KS to the exact p_0: sampler {ks_exact(x, 0.0, **uneven):.4f}, '
      f'exact sample {ks_exact(x_ref, 0.0, **uneven):.4f}')
d2l.set_figsize((5.5, 3))
d2l.plt.hist(x, bins=80, density=True, alpha=0.4, color='C1',
             label='reverse-SDE samples')
d2l.plt.plot(gs, p_t(gs, 0.0, **uneven), 'k', lw=1, label='data density $p_0$')
d2l.plt.xlabel('x'); d2l.plt.ylabel('density'); d2l.plt.legend();
```

```
max|p_T - N(0,1)| at T=3: 0.0145 (peak 0.40)
mass on the left: 0.797 (data 0.800)
means -1.995, +2.000 (data -2, +2)
widths 0.509, 0.510 (data 0.5)
KS to the exact p_0: sampler 0.0083, exact sample 0.0072
```



サンプラはデータを復元する。モードは約 0.02 以内の誤差で ± 2 に位置し、その幅は 0.5 に近く、粒子の 5 分の 4 が 100 分の 1 以内の精度で左側に落ち着く。注目すべきはこの最後の数値で

ある。粒子の半分は原点より右側から出発しており、サンプラの中で重みに言及している箇所はスコア以外に存在しない。スコアはどこでその重みを保持しているのだろうか。$t = 0$ の近くではない。そこでは密度が無視できないあらゆる場所で Equation 38.57 の負担率が 0 か 1 になり、重みが何であれスコアは 2 つのバネ $-(x \pm 2)/0.25$ のいずれかに一致する。これは 37.5.1 節で見た、離れたモードの重みに対するスコアの不感性である。$t = 3$ でもない。そこでは 2 つの成分がほぼ同じガウス分布になっており、粒子がどこにあっても左側の成分の負担率は重み 0.8 に近い。しかし 2 つのバネもほぼ重なっている。両者の差は $2m_t/v_t$（各成分の平均を $\mp m_t$、分散を v_t としたとき）であり、その大きさは 0.2（$t = 3$）と、16（$t = 0$）に比べてごく小さい。ほぼ等しい 2 つのバネを重み 0.8 と 0.2 で平均しても、左向きのごく小さな一様の押し出しが生じるだけで、どの粒子がどちらのモードへ行くかは決まらない。

重みが働くのはその中間、すなわち 2 つのモードが分離していく最中である。2 つの成分の負担率が等しくなるのは点 $x_t^* = v_t \ln(0.8/0.2)/(2m_t)$ である。この点より左にある粒子は左の成分から来た確率の方が高く、逆過程のもとでは左の成分に行き着く確率の方が高くなる。$t = 3$ ではこの分水嶺（watershed）は $x^* \approx 7$ にあり、どの粒子よりもはるか右方に位置する。$t = 1$ になると 0.85 まで内側へ寄ってくるため、$x = +0.5$ にある粒子でさえ右より左に行き着く可能性の方が高い。$t = 0$ では谷の底の 0.09 に位置するが、その頃には谷に粒子は残っていない。高ノイズでは重みの情報はいたるところにあるが何も決めず、低ノイズでは決められるはずなのにそこに粒子がいないのである。

出発点の分布は厳密には正しくなかったが、ほとんど影響しなかった。重みが不均等な場合、p_T はデータを完全には忘れていない。その平均は $-1.2\,e^{-3} \approx -0.06$ であり、出力の 1 行目は最大 0.0145 の差が $\mathcal{N}(0,1)$ とのあいだにあることを示している（ピーク値は 0.40）。逆 SDE のノイズはこの不一致の大半を洗い流し、左モードの質量の減少は約 0.002 と、表示された比率の二項分布ノイズ 0.003 よりも小さい。何も忘れない確率フロー ODE では、その数倍の質量を取りこぼすことになる（演習問題 10）。

このサンプラは厳密ではなく、KS 距離の行はそのわずかな兆候を示しているにすぎない。どちらの距離も 0.0096 の帯域と同程度であるが、サンプラの距離の方がたいてい大きく、厳密なサンプルが帯域を超えるのが 20 回に 1 回であるのに対し、サンプラではおよそ 6 回に 1 回超える。このサンプルサイズの 1 回の実行だけでは、このようなサンプラを厳密なサンプラと区別することはできない。初期分布のずれに加えて、もう 1 つの誤差要因である離散化（discretization）がある。オイラー・丸山法の弱収束次数は 1 であるため（38.4.4.1 節）、その出力にはステップ幅に比例するバイアスが生じる。このバイアスはモードの幅に現れるが、1 回の実行ではそこでも測定できない。M 個のサンプルから推定した幅のモンテカルロ誤差は幅を $\sqrt{2M}$ で割った値であり、約 0.003（16,000 サンプルの左モード）および 0.006（4,000 サンプルの右モード）になるからである。

このバイアスはサンプリングを行わずに切り分けられる。右モード単独のガウスデータ $\mathcal{N}(2, 0.25)$ に対しては、周辺分布もガウス分布 $\mathcal{N}(m_t, v_t)$ となりスコアは $-(x - m_t)/v_t$ であるから、逆向きの各ステップは x のアフィン変換となり、サンプラの状態の平均 m と分散 V は厳密な漸化式に従う。逆ドリフトの傾きを $\kappa_t = \theta - g^2/v_t$ とおけば、漸化式は $m \leftarrow m + \Delta t\,(\kappa_t m + g^2 m_t/v_t)$ および $V \leftarrow (1 + \kappa_t \Delta t)^2 V + g^2 \Delta t$ となる。厳密な p_T から出発すれば、この漸化式には離散化以外の誤差が一切入らない。次のセルでは、300 から 2,400 ステップの一様グリッド上でこれを評価し、さらに 1,500 ステップを 2 通りの方法、すなわち一様配分と、そのうち 1,000 ステップを $t \le 0.5$ に集中させる配分で比較する。

```
def em_reverse_width(ts, mu=2.0, v0=0.25):
    """Exact output width of reverse Euler--Maruyama on data N(mu, v0),
    stepping down the time grid ts = (0, t_1, ..., T) from T to 0."""
    mu, v0 = np.array([mu]), np.array([v0])
```

```
        (m,), (V,) = mixture_params(ts[-1], mu, v0)     # start from the exact p_T
        for t, dt in zip(ts[:0:-1], np.diff(ts)[::-1]):
            (mt,), (vt,) = mixture_params(t, mu, v0)
            kappa = theta - g**2 / vt       # slope of the reverse drift in x
            m, V = (m + dt * (kappa * m + g**2 * mt / vt),
                    (1 + kappa * dt)**2 * V + g**2 * dt)
        return np.sqrt(V)

    grids = {f'{N} uniform steps': np.linspace(0, T, N + 1)
             for N in (300, 600, 1200, 2400, 1500)}
    grids['1500 steps, 1000 on t <= 0.5'] = np.concatenate(
        [np.linspace(0, 0.5, 1001), np.linspace(0.5, T, 501)[1:]])
    for name, ts in grids.items():
        print(f'{name:>28}: width excess {em_reverse_width(ts) - 0.5:.4f}')
```

```
          300 uniform steps: width excess 0.0096
          600 uniform steps: width excess 0.0047
         1200 uniform steps: width excess 0.0024
         2400 uniform steps: width excess 0.0012
         1500 uniform steps: width excess 0.0019
1500 steps, 1000 on t <= 0.5: width excess 0.0006
```

最初の 4 つのグリッドでは、真の幅 0.5 に対する幅の超過分は順に 0.0096、0.0047、0.0024、0.0012 となり、1 次のバイアスがそうあるべきとおり、ステップ幅を半分にすると半分になる。先ほどのサンプリング実行のグリッドである 600 ステップでは 0.005 である。表示された幅に見込まれる超過分はまさにこの程度であり、モンテカルロ誤差と同じ大きさであるため、1 回のサンプリング実行では分解できないが、漸化式なら明確に捉えられる。

最後の 2 行は、ステップをどこに配分すべきかを問うている。1,500 ステップを一様に使えば、600 ステップ時のバイアスの 10 分の 6 が取り除かれる。一方、$t \le 0.5$ の区間だけステップを 10 倍細かくすると（合計は同じく 1,500 ステップ）、10 分の 9 近くが取り除かれる。ステップを 10 倍細かくすると細分化した区間由来のバイアスは 10 分の 1 になるため、0.0047 から 0.0006 への減少は、出力のバイアスのほぼすべてが $t \le 0.5$ の区間、すなわち逆時間積分の最後の 6 分の 1 のステップから生じていることを物語っている。

これには 2 つの要因が重なっている。第 1 に、この区間では逆ドリフトが最も急になり、傾きは $\theta - g^2/v_0 = 1 - 8 = -7$（$t = 0$）に達する一方、$-1$ 程度にとどまるのが $t = T$ 付近であるため、1 ステップあたりの誤差が大きくなる。第 2 に、逆時間積分の序盤で生じた誤差は後続のすべてのステップで漸化式の係数 $(1 + \kappa_t \Delta t)^2 < 1$ によって縮小されるのに対し、終盤のステップで生じた誤差はそのまま出力に到達する。データの幅がさらに狭くなると傾き $\theta - g^2/v_0$ は限りなく大きくなり、ステップ幅は 38.2.3 節の安定性限界に突き当たるが、今の設定ではそこから十分に離れている。演習問題 9 ではこれら両方の効果を測定し、最良のステップ配置を探る。

ここまでの議論はすべて、明示的に書き下せるスコアに支えられていた。データは 2 成分混合分布であり、残る誤差はオイラー・丸山法のバイアスと、p_T と $\mathcal{N}(0, 1)$ のわずかな差だけであった。しかし実データにはそのような公式はなく、フォッカー・プランク方程式から p_t を数値計算することも論外である。各軸に 100 個のセルを切ったグリッドには 100^d 個の未知数が並び、しかも d はピクセルの数だからである。スコアは学習するほかない（38.6 節）。画像に対して手元にあるのは p_0 からのサンプルだけであり、それ以外には何もない。

38.5.6 まとめ

SDE $d\mathbf{X} = \mathbf{f}\,dt + g\,d\mathbf{W}$ の密度はフォッカー・プランク方程式 $\partial_t p = -\nabla\cdot(\mathbf{f}p) + \tfrac{1}{2}g^2\Delta p$ に従い、ドリフトが密度を輸送し、ノイズが密度を平滑化する。$\nabla p = p\nabla\log p$ であるから、この平滑化自体が速度 $-\tfrac{1}{2}g^2\nabla\log p$ による輸送となっており、したがって同じ密度が常微分方程式、すなわち確率フロー ODE $\dot{\mathbf{x}} = \mathbf{f} - \tfrac{1}{2}g^2\nabla\log p_t$ によって動かされる。この ODE はあらゆる時刻で SDE と同じ周辺分布を持ち、決して交差しない滑らかな軌跡と尤度を与える。同じ密度族は、ノイズ λg とドリフト $\mathbf{f} - \tfrac{1+\lambda^2}{2}g^2\nabla\log p_t$ を持つ過程の族によって逆順にたどることもできる。スコア補正のうち半分相当が順方向の平滑化を打ち消し、残りがサンプラ自身のノイズを補償する。Anderson の逆 SDE（$\lambda=1$）が ODE（$\lambda=0$）の 2 倍の補正を持つ理由はここにある。本節ではフォッカー・プランク方程式の解の一意性を認めたうえで周辺分布についてこれを証明し、過程全体に関する Anderson の定理は引用にとどめた。厳密なスコアで駆動した逆 SDE は、対称なノイズから重み 0.8 と 0.2 の混合分布をモードの位置から比率に至るまで復元した。その比率が決まるのは、モードが分離していく中間のノイズ領域である。これらすべてのサンプラにおいて、唯一の未知量はスコア $\nabla\log p_t$ である。

38.5.7 演習問題

1. **平均化としてのラプラシアン**。滑らかな関数 ψ を、半径 r、中心 $\mathbf{x}$ の球面上で平均すると $\psi(\mathbf{x}) + \tfrac{r^2}{2d}\Delta\psi(\mathbf{x}) + O(r^4)$ になることは 38.5.2.1 節で述べたとおりである。

   1. $\mathbf{e}$ が $\mathbb{R}^d$ の単位球面上で一様分布に従うとする。対称性により、期待値が $\mathbb{E}[e_i] = 0$、$\mathbb{E}[e_ie_j] = \delta_{ij}/d$、および $\mathbb{E}[e_ie_je_k] = 0$ となることを示せ。これらのモーメントを $\psi(\mathbf{x}+r\mathbf{e})$ のテイラー展開に用いて、上の平均化公式を証明せよ。r^3 の項が現れないのはなぜか。
   2. 1 次元では、球面は 2 点 $x\pm\delta$ の組である。熱方程式 $\partial_t p = D\,\partial_{xx}p$ において $\partial_{xx}p$ を中心差分 $(p_{i+1}-2p_i+p_{i-1})/\delta^2$ で置き換えた前進オイラー法の 1 ステップは、新しい各 p_i を p_{i-1}、p_i、p_{i+1} の重みの和が 1 となる重み付き結合に設定することを示せ。3 つの重みすべてが非負となる最大のステップ幅 Δt を求めよ。このとき、反復によって新たな最大値が決して生じないのはなぜか。

2. **弱形式からのモーメント方程式**。時間発展則 Equation 38.51 は、有界集合の外でゼロになるテスト関数に対して導出された。本節で扱う密度が持つガウス分布の裾によって正当化されるものとして、この法則が多項式の ϕ に対しても成り立つと仮定せよ。

   1. 1 次元 OU 過程に対して $\phi(x)=x$ と $\phi(x)=x^2$ を代入し、平均 $m(t)$ と分散 $v(t)$（いずれも X_t のモーメント）に対する ODE を導け。ガウス分布であるか否かにかかわらず、これらがモーメント ODE Equation 38.54 に任意の初期分布 p_0 で一致することを示せ。`mixture_params` と照らし合わせ、時刻 t における 2 モード混合分布の平均と分散を計算してこれを確かめよ。
   2. 線形 SDE $d\mathbf{X} = \mathbf{A}\mathbf{X}\,dt + g\,d\mathbf{W}$（$\mathbb{R}^d$ 上、$\mathbf{A}$ は定数行列）に対し、$\phi(\mathbf{x}) = x_i$ と $\phi(\mathbf{x}) = x_ix_j$ を用いて、平均と共分散が $\dot{\mathbf{m}} = \mathbf{A}\mathbf{m}$ および $\dot{\Sigma} = \mathbf{A}\Sigma + \Sigma\mathbf{A}^\top + g^2\mathbf{I}$ を満たすことを示せ。定常共分散はどの行列方程式を満たさなければならないか。

3. **確率流ゼロから導く定常密度**。

   1. OU フォッカー・プランク方程式で $\partial_t p = 0$ とおいた式 $0 = \theta\,\partial_x(xp) + \tfrac{1}{2}g^2\partial_{xx}p$ を考えよ。x で 1 回積分し、無限遠での減衰から積分定数（確率流 Equation 38.65 にマイナスをつけたもの）がゼロになることを論じよ。得られた 1 階 ODE を解き、ガウス分布の形を仮定することなく $p_\infty = \mathcal{N}(0, g^2/(2\theta))$ を示せ。
   2. 一般のドリフト $f(x)$ と定数 g に対して同様の議論を行い、p_∞ をポテンシャル U（ただし $f=-U'$）を用いて表せ。逆に、密度 $p\propto e^{-E(x)}$ が与えられたとき、p を定常

にするドリフトを求め、それを p のスコアを用いて書け。

3. 二重井戸ドリフト $f(x) = x - x^3$（$g = \sqrt{2}$）にこの結果を適用し、p_∞ のモードを求めよ。不安定なドリフト $f(x) = \theta x$（$\theta > 0$）では何が破綻するか。

4. **モードの合流。**スコア Equation 38.58 を持つ対称な 2 成分混合分布を考えよ。

   1. Equation 38.58 を負担率の公式 Equation 38.57 から導け。
   2. $\mu^2 > v_0$ のときスコアは 3 つの零点を持ち、$x = 0$ のみが $\mu^2 \le v_0$ のときの零点であることを示せ。各零点を密度の極大値（極大点）または極小値（極小点）に分類せよ。（ヒント：0 における傾きは $(\mu^2 - v_0)/v_0^2$ である。）
   3. OU 過程のもとでは混合分布は対称なまま保たれ、各成分の平均は $\pm \mu e^{-\theta t}$、各成分の分散は `mixture_params` が計算する値となる。本節のパラメータである時刻 t^* の条件、すなわち p_t が二峰性でなくなる時刻を、$\mu = 2$、$v_0 = 0.25$、$\theta = 1$、$g = \sqrt{2}$ に対して求めよ。
   4. t^* におけるこの変化は、ノイズから出発する逆時間サンプラにとって何を意味するか。（Biroli et al. (2024) はこの対称性の破れの高次元版を解析し、種分化と呼んでいる。）

5. [code] **状態依存ノイズ。**●行列ノイズを持つ SDE $d\mathbf{X} = \mathbf{f}\,dt + \mathbf{G}(\mathbf{X}, t)\,d\mathbf{W}$ に対し、$\mathbf{a} = \mathbf{G}\mathbf{G}^\top$ とおく。このとき伊藤の補正項は $\tfrac{1}{2}\sum_{ij} a_{ij}\,\partial_i \partial_j \phi$ となり、2 回の部分積分 $\int p\, a_{ij}\,\partial_i\partial_j\phi = \int \phi\,\partial_i\partial_j(a_{ij} p)$ によって $\partial_t p = -\nabla \cdot \mathbf{j}$ が得られる。ここで確率流は

$$j_i = f_i\, p - \tfrac{1}{2}\sum_k \partial_k (a_{ik}\, p)$$

であり、p で割ることで確率フロー速度は

$$\mathbf{u} = \mathbf{f} - \tfrac{1}{2}\,\mathbf{a}\,\nabla \log p - \tfrac{1}{2}\,\nabla \cdot \mathbf{a}, \qquad (\nabla \cdot \mathbf{a})_i = \sum_k \partial_k a_{ik}.$$

となる。定数行列の場合はスコア項の g^2 が $\mathbf{a}$ に置き換わるだけだが、状態に依存する $\mathbf{a}$ では最後の項が加わる。ここでドリフトがなく $dX = g(X)\,dW$（g は滑らかで正）で表される、周期境界条件付きの円周 $[0, 1)$ 上の過程を考えよ。

   1. 上の行列に対する確率流と一般の速度をこの SDE に特殊化せよ。定常密度では確率流が定数になり、周期性からその定数がゼロになることを示し、p_∞ を求めよ。
   2. 定数 g^2 を Equation 38.52 の中で単純に $g(x)^2$ に置き換えると、PDE $\partial_t p = \tfrac{1}{2} g(x)^2\,\partial_{xx} p$ が得られる。この PDE はどのような定常密度を予測するか。
   3. 50,000 個の粒子を一様分布から出発させ、オイラー・丸山法を用いて $g(x) = 1 + \tfrac{1}{2}\sin(2\pi x)$、$\Delta t = 10^{-3}$ で $t = 2$ までシミュレーションせよ（位置は 1 を法として折り返す）。ヒストグラムを両方の予測と比較せよ。粒子はどこに集積するか、またそれはなぜか。

6. [code] **グリッド上のフォッカー・プランク方程式。**●区間 $[-6, 6]$ を N 個のセル（幅 $\delta = 12/N$、中心 x_i）に分割し、$p_i(t)$ でセル i 内の密度を近似する。OU 方程式の保存型離散化では、各内部境界 $x_{i+1/2}$ に確率流

$$j_{i+1/2} = f(x_{i+1/2})\,\frac{p_i + p_{i+1}}{2} - \frac{g^2}{2}\,\frac{p_{i+1} - p_i}{\delta}, \tag{38.72}$$

を割り当て、両端の外側境界を通る確率流をゼロとおいて、$\dot{p}_i = -(j_{i+1/2} - j_{i-1/2})/\delta$ により時間発展させる。これは線形系 $\dot{\mathbf{p}} = \mathbf{F}\mathbf{p}$ である。

1. $\sum_i p_i\,\delta$ が任意の $\mathbf{p}$ に対して保存されること（発散定理 Equation 38.48 の離散版）を示し、外側境界の確率流をゼロとおくことが 38.5.4 節のどの境界条件に対応するか答えよ。
2. $\mathbf{F}$ を $\theta = 1$、$g = \sqrt{2}$、$\delta = 0.05$ に対して構築せよ。セル中心上の 2 モード混合分布 `p_t(x, 0.0)` から出発し、$t = 0.5$ までステップ幅 $\Delta t = 10^{-3}$ の前進オイラー法およびステップ幅 $\Delta t = 10^{-2}$ の後退オイラー法（各ステップで 1 回の `np.linalg.solve`）を用いて積分せよ。両方の結果を `p_t(x, 0.5)` と比較せよ。
3. 前進オイラー法のステップ幅を 10^{-3} から大きくしていき、丸め誤差を種とする不安定モードが成長するのに十分長い $t = 5$ までの実行で安定に保たれる最大のステップ幅を求めよ。このステップ幅を、最も負の固有値 $-\nu$（$\mathbf{F}$ の固有値）と上界 $\Delta t < 2/\nu$（38.2.3 節）を通じて関連付けよ。δ を半分にするとこの上限はどう変わるか。演習問題 1b の条件と比較せよ。
4. 各軸に 100 個のセルを持つ d 次元グリッドには 100^d 個の未知数がある。$d = 2$ および $d = 784$（28×28 画像のピクセル数）に対してこの個数を計算せよ。この結果は、拡散モデルにおけるフォッカー・プランク方程式の使われ方について何を意味するか。

Särkkä and Solin (2019) の演習問題 5.3 を改変。

7. **発散ゼロの速度場。●**$\mathbf{R}$ を定数の反対称 $d \times d$ 行列とする。

1. 任意の滑らかな正の密度に対して $\nabla \cdot (p\,\mathbf{R}\nabla \log p) = 0$（密度を p とする）が成り立つことを示せ。ここから、任意の $\beta \in \mathbb{R}$ に対して ODE $\dot{\mathbf{x}} = \mathbf{u}_t(\mathbf{x}) + \beta\mathbf{R}\nabla \log p_t(\mathbf{x})$（ただし $\mathbf{u}_t$ は Equation 38.64 の速度）が確率フロー ODE と同じ周辺分布を持つことを導け。
2. 定常状態にある 2 次元 OU 過程、すなわちすべての時刻で $p_t = \mathcal{N}(\mathbf{0}, \frac{g^2}{2\theta}\mathbf{I})$（すべての t）および $\mathbf{R} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ を考えよ。確率フロー速度がゼロになることを示し、修正した ODE の軌跡がどのようなものになるか述べよ。
3. 今度は回転を SDE 自体に組み込み、$d\mathbf{X} = (-\theta\mathbf{I} + \beta\mathbf{R})\mathbf{X}\,dt + g\,d\mathbf{W}$（$\mathbb{R}^2$ 上）とする。演習問題 2b の共分散方程式を用いて、$\frac{g^2}{2\theta}\mathbf{I}$ が任意の β に対して定常共分散となり、この SDE が小問 (b) と同じ定常密度を持つことを示せ。その定常確率流、すなわちこの SDE の確率流 Equation 38.65 を計算せよ。定常確率流をゼロに強制した演習問題 3a の議論が、2 次元では成り立たないのはなぜか。
4. $\mathbf{f} = -\nabla U$ と仮定し、$\mathbf{u}_t$ が勾配場であるとする。速度場 $\mathbf{w}$ のうち、その連続の式 Equation 38.60 の解が p_t となるすべての場の中で、$\mathbf{u}_t$ が運動エネルギー（期待値 $\mathbb{E}_{p_t}\|\mathbf{w}\|^2$）を最小にすることを示せ。なお、ベナムー・ブレニエの定式化 (Benamou and Brenier 2000) は経路 p_t 自体についても同じエネルギーを最小化する。

8. **厳密なガウス逆ステップ。●**$p_t = \mathcal{N}(m_t, v_t)$ をガウス分布である OU 過程の周辺分布とする。ステップ幅 τ にわたる厳密な順遷移は $X_{t+\tau} \mid X_t = x \sim \mathcal{N}(\alpha_\tau x, \sigma_\tau^2)$（ただし $\alpha_\tau = e^{-\theta\tau}$、$\sigma_\tau^2 = \frac{g^2}{2\theta}(1 - e^{-2\theta\tau})$）で与えられる（カーネル Equation 38.46 による）。

1. ガウス分布の条件付けを用いて、次を示せ。

$$X_t \mid X_{t+\tau} = y \;\sim\; \mathcal{N}\!\left(\frac{\sigma_\tau^2\, m_t + \alpha_\tau v_t\, y}{v_{t+\tau}},\; \frac{\sigma_\tau^2\, v_t}{v_{t+\tau}}\right). \tag{38.73}$$

ここで $v_{t+\tau} = \alpha_\tau^2 v_t + \sigma_\tau^2$ は時刻 $t + \tau$ における周辺分散である。

2. 平均と分散を τ の 1 次まで展開し、逆ステップ Equation 38.69 を、スコアの前の係数 g^2 まで含めて導出せよ。

9. [code] **ステップの配置。**●関数 `em_reverse_width` は、任意の時間グリッドに対してデータ $\mathcal{N}(2, 0.25)$ 上の逆時間オイラー・丸山法の厳密な出力幅を返す。本文では 1,500 ステップの 2 つのグリッドを比較した。

   1. 600 ステップの一様グリッドから始め、$t \le c$ の区間のみステップを 10 倍細かくせよ（ただし $c \in \{0.1, 0.25, 0.5, 1, 2\}$）。それぞれのステップ数と幅の超過分を記録せよ。ステップを 10 倍細かくすると細分化した区間由来のバイアスは 10 分の 1 になる。これを用いて、600 ステップのバイアスのうち $t \le c$ のステップから生じる割合を計算せよ。$t \le 0.1$ に何個の追加ステップを置けば、本文で一様グリッドに追加した 900 ステップ分の働きをするか。本文ではこれらのステップが支配的になる理由を 2 つ挙げた。一様グリッド上でそれぞれを測定せよ。すなわち、$t \in \{0.1, 0.5, 3\}$ から出発する単一の 1 ステップについて、厳密な分散から始めたときに生じる V の誤差（初期値を v_t としたとき）と、後続のステップが $t = 0$ に至るまでにその誤差に掛ける倍率を計算せよ。
   2. 総ステップ数を $N = 1{,}500$ に固定し、グリッド $t_i = T\,(i/N)^\rho$ を $\rho \in \{1, 1.5, 2, 3, 4, 6\}$ に対して試せ。どの ρ が最小の超過分を与えるか。ρ を大きくしすぎると再び悪化するのはなぜか。
   3. 二乗グリッド（$\rho = 2$）上のバイアスは、依然として $1/N$ に関して 1 次であるか。$N = 750$、1,500、3,000 で確かめよ。グリッドの変更が変えたのは次数か、それとも定数係数か。
   4. より幅の狭いデータ $v_0 = 0.01$（$t = 0$ における逆ドリフトの傾きは $1 - 2/0.01 = -199$ となる）に対して小問 (b) を繰り返し、各超過分を目標幅 0.1 と比較せよ。最良の ρ はどう移動するか。この結果は、一部の方向で分散が小さいデータに対して何を示唆するか。（17.5 節では、ノイズレベルに関するべき乗グリッドを用いて画像サンプラのステップを低ノイズ側へと密に集めている。）

10. [extended] **事前分布の不一致：ODE 対 SDE。**●●サンプラは厳密な終端分布ではなく基準の事前分布 $q_T = \mathcal{N}(0, 1)$（p_T の代用）から出発する。本課題では、逆 SDE と逆時間確率フロー ODE がその不一致をどのように $t = 0$ まで運ぶかを比較する。小さい終端時刻に対して不均等な重み $w = (0.8, 0.2)$（38.5.5 節の逆サンプラで用いた設定であり、p_T が q_T から目に見えてずれる（小さい T の場合））を用いよ。具体的には、`**uneven` を `p_t`、`score_t`、`mixture_cdf`、`sample_p0` に渡し、`score=lambda x, t: score_t(x, t, **uneven)` を `pf_velocity` に渡す。

   1. 本文と同様の Anderson の逆 SDE と、ホイン法のステップによる逆時間確率フロー ODE を、いずれも q_T から $T \in \{0.25, 0.5, 1, 2, 3\}$、ステップ幅 $\Delta t = 0.005$ で実行せよ。$x < 0$ となるサンプルの割合（目標値は 0.8）を記録せよ。なお、$T = 3$ における SDE は本文の実行と同じである。
   2. Anderson の定理は過程全体に対して成り立つため、$X_T = y$ から出発した逆 SDE は事後分布に従う $X_0 \sim p(x_0 \mid x_T = y)$ を生成する。これを用いて、SDE の左モード質量を、$y \sim q_T$ に関する p_T の負担率（Equation 38.57）の期待値として予測せよ。ODE については、x_T から x_0 への写像が単調増加であり p_T を p_0 へ写すことを用いて、標準正規分布の CDF および p_T と p_0 の CDF から左モード質量を予測せよ。両方の予測を数値積分で評価し、実測値と比較せよ。
   3. サンプラの出力の分布を q_0 とする。37.3.3 節のデータ処理不等式を用いて、ODE の出力が $D_{\mathrm{KL}}(q_0 \| p_0) = D_{\mathrm{KL}}(q_T \| p_T)$ を満たす一方、SDE の出力は $D_{\mathrm{KL}}(q_0 \| p_0) \le D_{\mathrm{KL}}(q_T \| p_T)$ を満たすことを証明せよ。
   4. $D_{\mathrm{KL}}(q_T \| p_T)$ と SDE の $D_{\mathrm{KL}}(q_0 \| p_0)$ を数値積分で各 T に対して計算せよ。事後分布 $p(x_0 \mid x_T = y)$ は 2 成分混合ガウス分布であり、その各成分は演習問題 8a から従う

ため、q_0 は y に関する 1 次元積分で表される。$T = 1$ において、SDE は不一致を何分の 1 に縮小するか。

5. $T = 1$ において、族 Equation 38.71 のサンプラ（$\lambda \in \{0, 0.5, 1, 2\}$）に対して小問 (a) の測定を繰り返せ。順方向の終端時刻 T を短くせざるをえない場合、どのサンプラを使うべきか。また、ステップサイズを固定したとき λ をどこまで大きくできるかを制限する要因は何か。

38.6 スコアマッチングと拡散モデル

ガウスノイズをデータに変換する方法はすでにわかっている。逆時間 SDE（reverse-time SDE）または確率フロー ODE（probability-flow ODE）を積分すればよい。その代わりに、どちらも訪れるすべての点におけるノイズ付加データのスコア関数 $\nabla \log p_t$ という 1 つの量を必要とする。前節の 2 モード混合分布では、数式がこれを与えてくれた。しかし画像では、手元にあるのはサンプルだけであり、数式はない。本節では画像の訓練セットに見立てて、その混合分布から抽出した 4,096 個のサンプルを用いる。たった 1 点においてすら値を直接評価できないベクトル場を、学習しなければならないのである。

最尤法（maximum likelihood）で密度を当てはめてから微分しても解決にはならない。ネットワーク出力の指数をとる柔軟な密度には、全空間にわたる積分である正規化定数（normalizing constant）が必要になるからである。Aapo Hyvärinen は 2005 年、スコアには正規化定数が不要であり、部分積分（integration by parts）によって未知のデータスコアを目的関数から消去できることを見いだした (Hyvärinen 2005)。Pascal Vincent はさらに単純な事実に気づいた (Vincent 2011)。ノイズ付加データのスコアは、あるノイズ付きの点を生成し得たクリーンな点すべてにわたって、「元のクリーンな点へ戻る方向」を平均したものに等しい。この方向はすべての訓練ペアに対して既知であるため、通常の最小二乗法（least squares）によってサンプルのみからスコアを学習できる。

拡散モデル（diffusion model）はこのようにして訓練される。「加えたノイズを予測せよ」という標準的な損失は、特定の重み付けを伴うこの回帰にほかならない。最もよく知られた 2 つのサンプラである DDPM と DDIM も新しい方程式を持ち込むわけではなく、それぞれ既存の方程式を離散化したものである。混合分布で検証する前に、1 次元のガウス分布を用いれば、各主張を手計算で確かめられる数値へと落とし込める。

本節ではニューラルネットワークの訓練はしない。まず 1 つのノイズレベルにおけるスコアを、厳密に解けるほど小さなモデルを用いて最小二乗法で学習する。続いて混合分布の厳密なスコアの上でサンプラを解析し、測定される各誤差がサンプラ自身に由来するようにする。最初は単一のノイズレベルで試すが、それだけでは不十分であることがわかり、次にすべてのレベルを同時に扱う。そして終盤で 1 度だけ、すべてのレベルで最小二乗法により当てはめたスコアの上でサンプラを動かす。ネットワークとその誤差は第 17 章の役割であり、そこでは本節の 3 つの結果、すなわち Vincent の定理、DDPM と 38.4 節の分散保持（variance-preserving）SDE との同一視、および DDIM とオイラー法との同一視を土台とする。Hyvärinen の恒等式の証明とガイダンスの項は、初読時には飛ばしてもよい。

38.6.1 スコアの学習

データのスコアを近似するベクトル場 $\mathbf{s}_\theta$（実際には d 入力 d 出力のネットワーク）を求めたいが、できるのはサンプルの抽出だけである。最初の課題は、モデルと真のベクトル場を比較し、かつサンプルから評価できる目的関数を構成することである。

38.6.1.1 正規化定数を含まない目的関数

ニューラルネットワークが最も柔軟に密度を定義するのはエネルギーを介する場合である。**エネルギーベースモデル**（energy-based model）とは $p_\theta(\mathbf{x}) = e^{-E_\theta(\mathbf{x})}/Z_\theta$ のことであり、ここで E_θ はネットワーク、$Z_\theta = \int e^{-E_\theta(\mathbf{x})}\, d\mathbf{x}$ は正規化定数である (LeCun et al. 2006)。最尤法には $\nabla_\theta \log Z_\theta$ が必要となるが、これは d 次元積分であり、θ を更新するたびに変化する。一方、$\mathbf{x}$ に関するスコアにはこれが現れない。Z_θ は $\mathbf{x}$ に関して定数であるため、$\nabla_{\mathbf{x}} \log p_\theta = -\nabla_{\mathbf{x}} E_\theta$ となるからである。

そこで、密度ではなくベクトル場どうしを比較する。データのスコアを $\nabla \log p$ とすると、**明示的スコアマッチング**（explicit score matching）の目的関数は、データが存在する場所で重み付けした 2 つのベクトル場間の平均二乗距離である。

$$J_{\text{ESM}}(\theta) = \tfrac{1}{2}\, \mathbb{E}_{\mathbf{x}\sim p}\left[\left\| \mathbf{s}_\theta(\mathbf{x}) - \nabla \log p(\mathbf{x}) \right\|^2 \right]. \tag{38.74}$$

この目的関数は、データが確率質量を持つすべての場所でモデルのベクトル場がデータのスコアに一致するとき、かつそのときに限り 0 となる。$\mathbf{s}_\theta$ が密度 q のスコアである場合、これは 37.5.1 節のフィッシャーダイバージェンス（Fisher divergence）に一致する（同節では、このようなダイバージェンスが何を識別でき、何を識別できないかも論じている）。ここでは $\mathbf{s}_\theta$ は任意のベクトル場でよい。d 個の出力を持つネットワークは、何らかの関数の勾配である必要がないからである。この目的関数から正規化定数は消えたが、唯一未知であるベクトル場 $\nabla \log p$ がまだ含まれている。Hyvärinen の部分積分がこれを消去する。

命題（Hyvärinen の恒等式） *p を滑らかな正の密度、$\mathbf{s}_\theta$ を滑らかなベクトル場とし、$p(\mathbf{x})\, \mathbf{s}_\theta(\mathbf{x}) \to \mathbf{0}$（$\|\mathbf{x}\| \to \infty$ のとき）が成り立ち、以下の期待値がすべて有限であるとする。このとき、次が成り立つ。*

$$J_{\text{ESM}}(\theta) = \mathbb{E}_{\mathbf{x}\sim p}\left[\tfrac{1}{2} \|\mathbf{s}_\theta(\mathbf{x})\|^2 + \nabla \cdot \mathbf{s}_\theta(\mathbf{x}) \right] + C, \tag{38.75}$$

ここで、$\nabla \cdot \mathbf{s}_\theta = \sum_i \partial s_i / \partial x_i$ は発散であり、$C = \tfrac{1}{2}\mathbb{E}_p \|\nabla \log p\|^2$ は θ に依存しない。

証明 Equation 38.74 の二乗を展開する。項 $\tfrac{1}{2}\mathbb{E}\|\nabla \log p\|^2$ は定数 C であり、項 $\tfrac{1}{2}\mathbb{E}\|\mathbf{s}_\theta\|^2$ はそのまま Equation 38.75 に現れるため、残るは交差項の変形である。1 次元では次のようになる。

$$-\mathbb{E}_{x\sim p}\left[s_\theta(x)\, (\log p)'(x) \right] = -\int s_\theta(x)\, \frac{p'(x)}{p(x)}\, p(x)\, dx = -\int s_\theta(x)\, p'(x)\, dx,$$

さらに境界項 $[s_\theta\, p]_{-\infty}^{\infty} = 0$ のもとで部分積分すると、$+\int s_\theta'(x)\, p(x)\, dx = \mathbb{E}_p[s_\theta']$ が残る。これはスタインの恒等式 $\mathbb{E}_p[s_\theta' + s_\theta\, (\log p)'] = 0$（37.5.2 節）にほかならず、導関数が未知の密度からモデル側へと移っている。d 次元では、各座標 i に対して s_i を s_θ の代わりに、$\partial_i p$ を p' の代わりに用いて同じ手順を適用し、まず x_i について積分する（仮定した可積分性のもとで、フビニの定理によりこれが正当化される）。これを i について足し合わせると、交差項は $\mathbb{E}_p[\sum_i \partial_i s_i] = \mathbb{E}_p[\nabla \cdot \mathbf{s}_\theta]$ となる。■

Equation 38.75 の右辺の各項はいずれも、与えられたモデルに対して評価できる量の p のもとでの期待値であるため、Z_θ も $\nabla \log p$ も使わずに標本平均で推定できる。この右辺を**陰的スコアマッチング**（implicit score matching）と呼ぶ。

ガウス分布を使えば、この恒等式が機能することが確かめられる。データを $x \sim \mathcal{N}(0, v)$（スコアは $-x/v$）とし、1 パラメータモデル $s_w(x) = -x/w$ をとる。このとき $\tfrac{1}{2}s_w^2 + s_w' = x^2/(2w^2) - 1/w$

となるため、陰的目的関数は $J(w) = v/(2w^2) - 1/w$ である。その導関数 $-v/w^3 + 1/w^2$ は $w = v$ で 0 になる。すなわち、目的関数はデータのスコアを一度も参照せずに分散を返す。定数項も一致する。最小点では $J(v) = -1/(2v)$ であり、また $C = \frac{1}{2}\mathbb{E}[x^2/v^2] = 1/(2v)$ であるため、そこでの明示的目的関数は $J(v) + C = 0$ となり、モデルがデータに一致するときに成り立つべき値と合致する。

実際には、期待値は標本平均になる。v（目的関数 J 中の分散）を、x^2 の 10,000 回の抽出（$\mathcal{N}(0, 2)$ からのサンプル）にわたる平均で置き換え、w のグリッド上で最小化してみよう。このとき最小点はその標本平均となり、その標準誤差は $v\sqrt{2/10^4} \approx 0.03$ であるため、2 の前後（小数第 2 位で数単位の誤差範囲内）の値が得られるはずである。

```
rng = np.random.default_rng()
v = 2.0
x = rng.normal(0.0, np.sqrt(v), 10000)        # data x ~ N(0, v)
w = np.linspace(0.5, 4.0, 701)                # model s_w(x) = -x / w
J_ism = (x**2).mean() / (2 * w**2) - 1 / w    # E[s_w^2 / 2 + s_w']
print(f'implicit score matching picks w = {w[J_ism.argmin()]:.2f}  (v = {v})')
```

```
implicit score matching picks w = 1.96  (v = 2.0)
```

実際にその通りになる。Equation 38.75 の 2 つの項は互いに逆向きに引き合っており、ガウス分布の例はその仕組みをよく示している。発散の項 $-1/w$ は、サンプルの位置で内側を向くベクトル場、すなわちデータがある場所の吸い込み口（シンク）を優遇し、その傾きが急であるほど目的関数を大きく引き下げる。一方、二乗ノルム $x^2/(2w^2)$ はベクトル場が際限なく大きくなるのを食い止める。両者が釣り合う点が真の分散に一致する。

陰的スコアマッチングには 2 つの限界がある。第 1 に、証明では境界項が消える必要がある。有界区間上のデータでは境界項が 0 にならず、陰的目的関数は誤ったベクトル場を選んでしまう。第 2 の限界はコストである。発散はヤコビ行列（Jacobian matrix）のトレース $\nabla \cdot \mathbf{s}_\theta = \operatorname{tr}\left(\partial \mathbf{s}_\theta / \partial \mathbf{x}\right)$ である。これを厳密に計算するにはヤコビ行列の 1 行ごとに 1 回のリバースモードの逆伝播が必要となり、合計で d 回のパスを要するが、画像では d が数百万にのぼる。Hutchinson の推定量（38.3.3 節）はランダムな方向ごとに 1 回のベクトル・ヤコビ行列積でトレースを置き換えるものの、分散が増える。ノイズ除去スコアマッチングは、このトレースを完全に回避する。

38.6.1.2 ノイズ除去スコアマッチング

トレースを含まない目的関数が必要であり、同時にサンプラが実際に必要とする場所（データ上だけでなくその近傍の点を含む領域）でのスコアも必要である。**ノイズを加えた**データのスコアを一致させることで、この両方の要件が満たされる。各サンプルにスケール σ のガウスノイズを加えて摂動させる。

$$\tilde{\mathbf{x}} = \mathbf{x} + \sigma\epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$
$$p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}};\, \mathbf{x},\, \sigma^2 \mathbf{I}).$$

ノイズ付加周辺密度 $p_\sigma(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}$ はデータの密度をガウス分布でぼかしたものであり、p へと σ の減少に伴って近づく。そのスコアはデータのスコアと同様に未知である。しかし、**条件付き分布**のスコアはガウス密度の対数から直ちに得られる。

$$\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2} = -\frac{\epsilon}{\sigma}. \tag{38.76}$$

このベクトルは、ノイズ付き観測からそれを生み出したクリーンサンプル（clean sample）へと戻る方向を指しており、生成したすべてのペアに対して既知である。**ノイズ除去スコアマッチング**（denoising score matching、DSM）は、これを回帰目標（regression target）として用いる。

$$J_{\mathrm{DSM}}(\theta) = \tfrac{1}{2}\,\mathbb{E}_{\mathbf{x}\sim p,\;\tilde{\mathbf{x}}\sim p_\sigma(\cdot\mid\mathbf{x})} \Big[\, \big\| \mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \big\|^2 \Big]. \tag{38.77}$$

この損失は、直接見ることはできないクリーンな点へ戻る方向を、ノイズ付きの点のみから予測するようネットワークに求める。目標はクリーンな点に依存する一方、求めたいスコアはノイズ付きの点のみに依存する。最小二乗法は目標の条件付き平均を当てはめるため、両者を結びつける。次の補題はその事実を述べたものである。初等的な内容であるが、本節と次節はいずれもこれに依拠しており、次節ではこれを速度に適用する。

補題（条件付き平均への回帰） (X, Y) **が同時分布に従い** $\mathbb{E}\|Y\|^2 < \infty$ **を満たすとし、** $\mathbf{m}(X) = \mathbb{E}[Y \mid X]$ **とおく。このとき、任意の可測関数 v（ただし** $\mathbb{E}\|\mathbf{v}(X)\|^2 < \infty$**）に対して次が成り立つ。**

$$\mathbb{E}\,\|\mathbf{v}(X) - Y\|^2 = \mathbb{E}\,\|\mathbf{v}(X) - \mathbf{m}(X)\|^2 + \mathbb{E}\,\|Y - \mathbf{m}(X)\|^2. \tag{38.78}$$

第 2 項は v を含まない。すなわち、ノイズを含む目標 Y **に対する最小二乗損失を最小化することは、定数項の差を除いて、条件付き平均** $\mathbf{m}(X)$ **に対する最小二乗損失を最小化することと同じ問題である。**

証明 $\pm\mathbf{m}(X)$ を挿入して展開する。交差項は $2\,\mathbb{E}\big[(\mathbf{v}(X) - \mathbf{m}(X))^{\top}(\mathbf{m}(X) - Y)\big]$ である。塔性質 $\mathbb{E}[Z] = \mathbb{E}[\mathbb{E}[Z \mid X]]$（36.2 節）により、まず Y について（X を固定したもとで）平均をとることができ、$\mathbb{E}[\mathbf{m}(X) - Y \mid X] = \mathbf{0}$ から交差項は 0 になる。■

命題（Vincent の定理） **ノイズ付加カーネル** $p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ **が** $\tilde{\mathbf{x}}$ **に関して微分可能であり、勾配と x に関する積分が交換可能（ガウスカーネルでは両方成り立つ）であるとし、さらに DSM の目標と** $\mathbf{s}_\theta(\tilde{\mathbf{x}})$ **が有限の二次モーメントを持つとする。このとき、次が成り立つ。**

$$J_{\mathrm{DSM}}(\theta) = \tfrac{1}{2}\,\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma}\Big[\, \big\| \mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla \log p_\sigma(\tilde{\mathbf{x}}) \big\|^2 \Big] + C,$$

ここで、 C **は** θ **に依存しない。右辺は** p_σ **に対する明示的目的関数 Equation 38.74 であるため、ノイズ付加データ上でのノイズ除去スコアマッチングと明示的スコアマッチングは、同じ最小点および同じ勾配を持つ。**

証明 $X = \tilde{\mathbf{x}}$、$Y = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ として補題を適用する。残る課題は Y の条件付き平均を求めることであり、その同定こそが本定理の実質的な内容である。すなわち、**周辺スコアは条件付きスコアの事後平均（posterior mean）に等しい**。事後分布 $p(\mathbf{x} \mid \tilde{\mathbf{x}})$ をベイズの定理で書き下し、続いて勾配と積分を交換すると、次のようになる。

$$\begin{aligned} \mathbb{E}\,[Y \mid \tilde{\mathbf{x}}] &= \int \frac{\nabla_{\tilde{\mathbf{x}}}\, p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})}{p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})}\, \frac{p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\, p(\mathbf{x})}{p_\sigma(\tilde{\mathbf{x}})}\, d\mathbf{x} \\ &= \frac{\nabla_{\tilde{\mathbf{x}}} \int p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}}{p_\sigma(\tilde{\mathbf{x}})} = \nabla \log p_\sigma(\tilde{\mathbf{x}}). \end{aligned}$$

■

この証明では、ガウス分布の性質としては微分可能性しか使っておらず、データ分布についてはいかなる性質も仮定していない。有限の訓練セットに対しては積分は和になり、定理はぼかされた訓練セットのスコアを返す。ここから示唆に富む重大な帰結が導かれる。あらゆるノイズレベルで最小値に到達したネットワークと、そのスコアを誤差なく追跡するサンプラがあれば、返されるのは訓練点だけであり、それ以外の点は一切生成されない（演習問題 3 で極限 $\sigma \to 0$ を調べる）。したがって、モデルが新しいサンプルを生み出せるのは、最小値の手前で止まる回帰のおかげであって、目的関数のおかげではない。いつそのような汎化が起きるかについて知られている知見は 17.10 節にまとめられている。ガウスノイズが選ばれる理由は、その条件付きスコア Equation 38.76 がノイズそのものを $-1/\sigma$ 倍したものになるからである。ゆえにスコアの予測とノイズ ϵ の予測は同じ課題であり、ノイズを予測するネットワーク ϵ_θ はスコアモデル $\mathbf{s}_\theta = -\epsilon_\theta/\sigma$ を定める。拡散モデルが用いるのは、この**ノイズ予測**（noise prediction）形式である。

例：ガウス分布再びデータを $x \sim \mathcal{N}(0, v)$ とし、レベル σ でノイズを加える。ノイズ付加周辺分布は $p_\sigma = \mathcal{N}(0, v+\sigma^2)$ であり、そのスコアは $-\tilde{x}/(v+\sigma^2)$ である。2 つのガウス分布に対するベイズの定理から、クリーンな点の事後分布は次のようになる。

$$x \mid \tilde{x} \;\sim\; \mathcal{N}\!\left(\frac{v}{v+\sigma^2}\,\tilde{x},\; \frac{v\sigma^2}{v+\sigma^2}\right),$$

これはノイズ付きの点をデータの平均へ向けて縮小（shrinkage）させた分布であり、その広がりは $\tilde{x}$ に依存しない。本項のすべての主張が具体的な数式になる。目標 $Y = (x-\tilde{x})/\sigma^2$ の条件付き平均は $\left(\frac{v}{v+\sigma^2} - 1\right)\tilde{x}/\sigma^2 = -\tilde{x}/(v+\sigma^2)$ となり、Vincent の定理の通り周辺スコアに一致する。達成可能な最小の平均二乗誤差、すなわち Equation 38.78 の第 2 項は、目標の事後分散（posterior variance）である $\mathrm{Var}(x \mid \tilde{x})/\sigma^4 = \frac{1}{\sigma^2} - \frac{1}{v+\sigma^2}$ となる。$v=1$、$\sigma=0.5$ のとき、この下限は $4 - 0.8 = 3.2$ であり、クリーンな点の事後平均は $0.8\,\tilde{x}$ である。100,000 組のペアを用いたモンテカルロ実行によって両方の数値を確かめよう。

```
x, eps = rng.standard_normal((2, 100000))            # v = 1
xt, y = x + 0.5 * eps, -eps / 0.5                     # sigma = 0.5; DSM target y
print(f'mean squared error of the conditional mean -xt/1.25: {((y + xt /
↪  1.25)**2).mean():.3f}'
      '  (floor 3.2)')
print(f'least-squares slope of x on xt: {(xt * x).mean() / (xt**2).mean():.3f}  (0.8)')
```

```
mean squared error of the conditional mean -xt/1.25: 3.186  (floor 3.2)
least-squares slope of x on xt: 0.802  (0.8)
```

最良の回帰関数であっても、目標に対する平均二乗誤差は 0 ではなく 3.2 になる。1 つのノイズ付きの点を説明し得るクリーンな点は多数存在し、回帰はその平均しか返せないからである。ここでは 3.2 の損失が完璧な当てはめを意味し、ゼロの損失は不可能である。傾き 0.80 は縮小係数 $v/(v+\sigma^2)$ である。この 2 つの観測事実は、いずれもガウス分布に限ったものではない。

この傾きは、Robbins が Tweedie に帰属させて発表した **Tweedie の公式**（Tweedie's formula）の一例である (Robbins 1956; Efron 2011)。Vincent の恒等式 $\mathbb{E}[(\mathbf{x}-\tilde{\mathbf{x}})/\sigma^2 \mid \tilde{\mathbf{x}}] = \nabla \log p_\sigma(\tilde{\mathbf{x}})$ を整理すると、次の式が得られる。

$$\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2\, \nabla \log p_\sigma(\tilde{\mathbf{x}}). \tag{38.79}$$

二乗誤差のもとで最良のノイズ除去器（denoiser）は、ノイズ付きの点をノイズ付加データのスコアの方向にステップサイズ σ^2 で 1 ステップ進める（Figure 38.23）。ガウス分布の場合は

$\tilde{x} - \sigma^2 \tilde{x}/(v+\sigma^2) = 0.8\,\tilde{x}$ となり、先ほど測定した傾きに一致する。

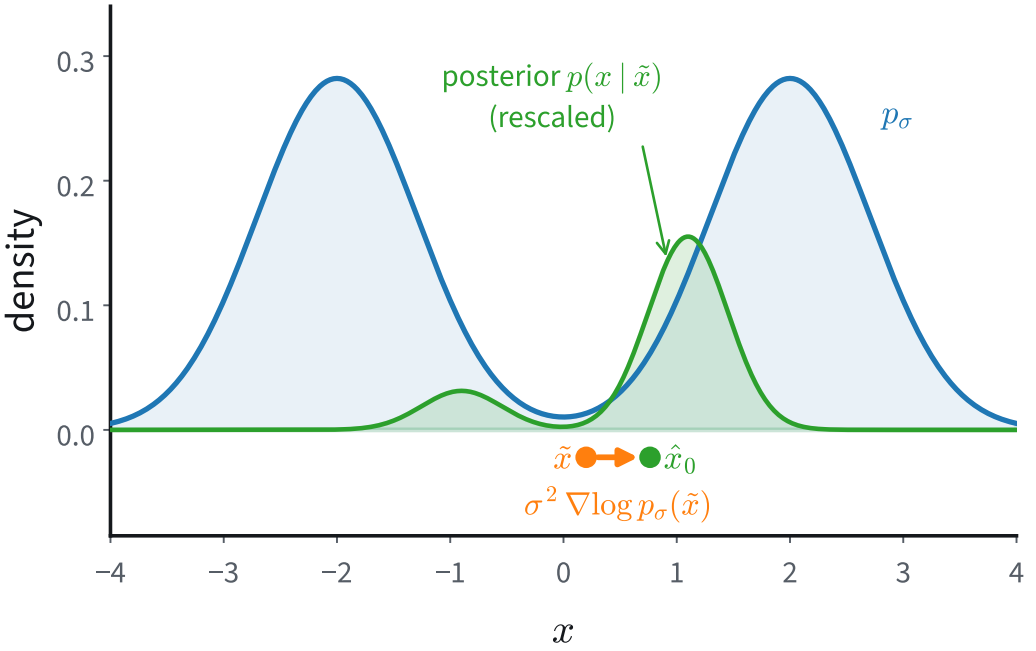


Figure 38.23: 混合分布 $\frac{1}{2}\mathcal{N}(-2, 0.5^2) + \frac{1}{2}\mathcal{N}(2, 0.5^2)$ に $\sigma = 0.5$ でノイズを加えた場合の Tweedie の公式。ノイズ付き観測 $\tilde{x}$ は、ノイズ付加密度 p_σ の低密度の谷に位置している。クリーンな点の事後分布は二峰性だが非対称であり、ステップ $\tilde{x} + \sigma^2 \nabla \log p_\sigma(\tilde{x})$ はその平均 $\hat{x}_0 = \mathbb{E}[x \mid \tilde{x}]$、すなわち最適なノイズ除去器に着地する。

損失の下限も同様に一般的である。最適点において、平均二乗誤差 $2J_{\text{DSM}}$ は $\mathbb{E}\|Y - \mathbf{m}(X)\|^2$、すなわち目標の平均事後分散であり回帰の**ベイズリスク**（Bayes risk）でもある量に等しい。これを計算するため、補題 Equation 38.78 をベクトル場 $\mathbf{v} = \mathbf{0}$ に適用すると、$\mathbb{E}\|Y\|^2 = \mathbb{E}\|\mathbf{m}(X)\|^2 + \mathbb{E}\|Y - \mathbf{m}(X)\|^2$ が得られる。目標 $Y = -\epsilon/\sigma$ は $\mathbb{E}\|Y\|^2 = d/\sigma^2$（$d$ 次元の場合）を満たし、その条件付き平均はスコアであるため、次が成り立つ。

$$\mathbb{E}\,\|Y - \nabla \log p_\sigma(\tilde{\mathbf{x}})\|^2 = \frac{d}{\sigma^2} - \mathbb{E}_{\tilde{\mathbf{x}} \sim p_\sigma}\,\|\nabla \log p_\sigma(\tilde{\mathbf{x}})\|^2 . \tag{38.80}$$

ガウス分布に対しては、これは再び $\frac{1}{\sigma^2} - \frac{1}{v+\sigma^2}$ となる。実際に観測される損失には当てはめ自体の誤差も含まれるため、ゼロではなくこの下限と比較すべきである。

またこの下限から、ノイズレベルが非常に小さいときに何が問題になるかもわかる。滑らかな密度を持つデータでは、下限は d/σ^2 のオーダーで $\sigma \to 0$ に伴って増大する。目標 $-\epsilon/\sigma$ が巨大になる一方で、その平均先であるスコアの平均二乗は有限にとどまる。したがって、単一の小さなノイズレベルでは目標の大部分がノイズで占められ、悪条件な回帰問題になってしまう。

Tweedie の公式は、最良のノイズ除去器がノイズ付きの点をどこへ送るかを表す。これをもう 1 回微分すると、ノイズ付きの点の変化に対してその出力がどれだけ速く変化するかがわかり、それがノイズ付加密度の曲がり具合（曲率）の鋭さを制限することが明らかになる。この上界が必要になるのは最後の項（サンプラを特定のクラスへ誘導する箇所）の 1 度だけであるが、事後分布が目の前にある今のうちに数行で証明しておこう。

命題（ノイズ付加密度の曲率） 任意のデータ分布と任意の $\sigma > 0$ に対して、次が成り立つ。

$$\nabla^2 \log p_\sigma(\tilde{\mathbf{x}}) = -\frac{\mathbf{I}}{\sigma^2} + \frac{1}{\sigma^4}\,\mathrm{Cov}\left[\mathbf{x} \mid \tilde{\mathbf{x}}\right], \tag{38.81}$$

したがって、$-\nabla^2 \log p_\sigma$ のいかなる固有値も $1/\sigma^2$ を超えない。

証明 Tweedie の公式（Equation 38.79）より $\nabla \log p_\sigma(\tilde{\mathbf{x}}) = (\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}] - \tilde{\mathbf{x}})/\sigma^2$ であるから、事後平均 $\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}] = \int \mathbf{x}\, p(\mathbf{x} \mid \tilde{\mathbf{x}})\, d\mathbf{x}$ を微分すればよい。事後分布はベイズの定理を通じて $\tilde{\mathbf{x}}$ に依存しており、$\nabla_{\tilde{\mathbf{x}}} \log p(\mathbf{x} \mid \tilde{\mathbf{x}}) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) - \nabla \log p_\sigma(\tilde{\mathbf{x}}) = (\mathbf{x} - \mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}])/\sigma^2$ が成り立つ。$\nabla p = p\,\nabla \log p$ を用いて積分記号下で微分すると、事後平均のヤコビ行列は次のようになる。

$$\int \mathbf{x}\, \frac{(\mathbf{x} - \mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}])^\top}{\sigma^2}\, p(\mathbf{x} \mid \tilde{\mathbf{x}})\, d\mathbf{x} = \frac{1}{\sigma^2}\,\mathrm{Cov}\left[\mathbf{x} \mid \tilde{\mathbf{x}}\right].$$

ここからヤコビ行列 $\mathbf{I}$（$\tilde{\mathbf{x}}$ のヤコビ行列）を引いて σ^2 で割れば、Equation 38.81 が得られる。共分散は半正定値であるため、求める上界が従う。$\blacksquare$

ノイズ付加密度は、加えたノイズ自身よりも鋭く曲がることはできない。すなわち、どのピークもそれをぼかしたガウス分布より細くはならない。ガウスデータの場合、Equation 38.81 の右辺は $-\frac{1}{\sigma^2} + \frac{v}{\sigma^2(v+\sigma^2)} = -\frac{1}{v+\sigma^2}$ となり、確かに $\log \mathcal{N}(0, v+\sigma^2)$ の曲率に一致する。

38.6.1.3 1 次元における最小二乗スコア

ガウス分布のスコアは線形であり、1 パラメータモデルで表現できた。では、ノイズ付きペアに対する最小二乗法は、曲がったスコアも復元できるだろうか。データとして前節の混合分布 $p = \frac{1}{2}\mathcal{N}(-2, 0.5^2) + \frac{1}{2}\mathcal{N}(2, 0.5^2)$ をとり、$\sigma = 0.5$ でノイズを加える。ノイズ付加により各成分の分散に σ^2 が加わるため、p_σ も再び混合分布となり、その成分分散は $0.25 + 0.25 = 0.5$ である。38.5.3 節で導出した通り、平均 $\pm\mu$ と共通の成分分散 v を持つ対称な混合分布のスコアは $(\mu \tanh(\mu x/v) - x)/v$ で与えられる。$\mu = 2$、$v = 0.5$ とすると、目標となる曲線は次の通りである。

$$\nabla \log p_\sigma(\tilde{x}) = 4 \tanh(4\tilde{x}) - 2\tilde{x}.$$

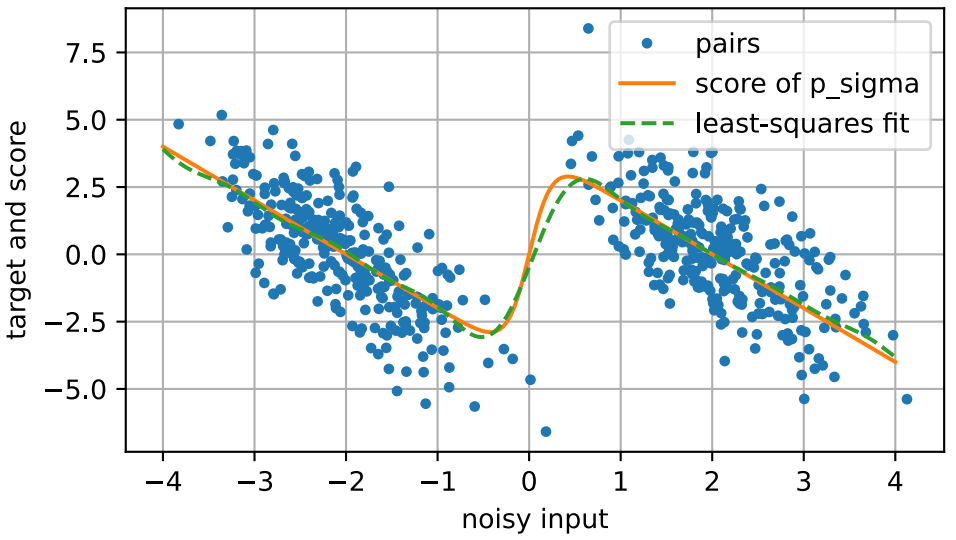
各モードの近傍では、このスコアは単一成分の線形スコア $-(\tilde{x} \mp 2)/0.5$ に近い。モード間では、-2.9 付近（$\tilde{x} = -0.45$）から $+2.9$（$\tilde{x} = 0.45$）へと立ち上がり、谷から外へと点を押し出す。

ここで用いるモデルはネットワークではない。17 個の固定された特徴量、すなわち幅 0.6 で $[-4.5, 4.5]$ 上に中心を並べた 16 個のガウスバンプと恒等関数 x の線形結合である。これを選ぶのは、DSM 損失 Equation 38.77 の当てはめがペア $(\tilde{x}, -\epsilon/\sigma)$ に対する通常の最小二乗法になり、`np.linalg.lstsq` で厳密に解けるからである。当てはめと真のスコアとのずれはすべて標本誤差（sampling error）であり、オプティマイザのせいにはできない。ノイズの抽出にはコストがかからないため、4,096 個の各クリーンな点に 4 回ずつノイズを加える。得られた 16,384 組のペアにモデルを当てはめて解析的なスコアと比較し、さらにデータから遠く離れた $x = 8$ でも両者を評価する。出力される平均二乗誤差は、回帰補題が分解する量 $2J_{\mathrm{DSM}}$ である。

```
n, sigma = 4096, 0.5
def mixture_score(q, var, means=(-2.0, 2.0), ws=(0.5, 0.5)):   # analytic score, any level
    logw = np.stack([np.log(w) - (q - mu)**2 / (2 * var) for mu, w in zip(means, ws)])
```

```
    r = np.exp(logw - logw.max(0))                                  # responsibilities, stable
↪   softmax
    return (r * np.stack([(mu - q) / var for mu in means])).sum(0) / r.sum(0)
def noisy_pairs(n, rng):                                            # n points of p, four noisings
↪   each
    x = np.repeat(rng.normal(4.0 * rng.integers(0, 2, n) - 2.0, 0.5), 4)   # x ~ p, the
↪   mixture
    eps = rng.standard_normal(x.size)
    return x + sigma * eps, -eps / sigma                            # the (input, target) pairs
centers = np.linspace(-4.5, 4.5, 16)
def features(q):                                                    # identity + 16 bumps, width 0.6
    return np.column_stack([q, np.exp(-(q[:, None] - centers)**2 / 0.72)])
def fit(xt, y):                                                     # least squares = the regression
    return np.linalg.lstsq(features(xt), y, rcond=None)[0]
xt, y = noisy_pairs(n, rng)
w_ls = fit(xt, y)
s_hat = lambda q: features(q) @ w_ls
g = np.linspace(-8, 8, 16001); dg = g[1] - g[0]
p_sigma = 0.5 * (np.exp(-(g + 2)**2) + np.exp(-(g - 2)**2)) / np.sqrt(np.pi)  # var 0.5 each
floor = 1 / sigma**2 - (p_sigma * mixture_score(g, 0.5)**2).sum() * dg
excess = (p_sigma * (s_hat(g) - mixture_score(g, 0.5))**2).sum() * dg
print(f'mean squared error: fit {((s_hat(xt) - y)**2).mean():.2f}, exact score '
      f'{((mixture_score(xt, 0.5) - y)**2).mean():.2f}, population floor {floor:.2f}')
print(f'excess risk {excess:.3f}; fit at x = 8: {s_hat(np.array([8.0]))[0]:+.1f}, '
      f'exact score {4 * np.tanh(32) - 16:.0f} (no pairs there)')
grid = np.linspace(-4, 4, 201)
d2l.plot([xt[:600], grid, grid], [y[:600], 4 * np.tanh(4 * grid) - 2 * grid, s_hat(grid)],
         'noisy input', 'target and score', fmts=('.', '-', '--'),
         legend=['pairs', 'score of p_sigma', 'least-squares fit'], figsize=(5.5, 3))
```

```
mean squared error: fit 2.12, exact score 2.12, population floor 2.11
excess risk 0.013; fit at x = 8: +2.3, exact score -12 (no pairs there)
```



プロットには回帰が目にする情報のすべてが現れている。すなわち、目標 $-\epsilon/\sigma$ の点群があらゆ

る $\tilde{x}$ において数単位にわたり広がっている。その点群の真ん中を条件付き平均である p_σ のスコアが貫いており、最小二乗による当てはめはその上に重なっている。出力される数値も回帰補題の通りである。母集団の下限 Equation 38.80、すなわち数値積分で計算した $\frac{1}{\sigma^2} - \mathbb{E}_{p_\sigma}(\nabla \log p_\sigma)^2$ は 2.11 である。これは曲線の周りにおける点群の縦方向の平均二乗の広がりを表す。当てはめと厳密なスコアはいずれもこの下限から 0.1 以内の平均二乗誤差を示し、両者どうしは 0.01 程度の精度で一致する。どちらも 2.11 ぴったりになる必要はない。各値は 1 組のペア標本にわたる平均であり、標本を取り直すごとに変動するからである。当てはめと真のスコアを隔てるのは補題の第 1 項である**過剰リスク**（excess risk）であり、約 0.01 と下限の 1% 未満に収まっている。

最後の数値はこの手法の限界を示している。$|\tilde{x}| \approx 5$ を超える領域にはノイズ付きペアが 1 つも落ちないため、$x = 8$ における当てはめは基底関数がたまたま外挿した値にすぎない。出力値が厳密値 -12 の近くに落ちることもあれば大きく外れることもあり、1 回の当てはめだけではどちらになるか判断できない。そこで、毎回新たなデータと新たなノイズを用いて実験を 20 回繰り返してみよう。1 回の標本抽出で値がどれだけ動くかを見るために厳密なスコアの平均二乗誤差を記録するとともに、2 つの点での当てはめ値を記録する。ペアが豊富に存在する右モードの裾 $x = 1$ と、ペアが皆無である $x = 8$ である。

```
runs = [noisy_pairs(n, rng) for _ in range(20)]                # 20 fresh draws of data and
↪  noise
errs = np.array([((mixture_score(xt_r, 0.5) - y_r)**2).mean() for xt_r, y_r in runs])
W = np.array([fit(xt_r, y_r) for xt_r, y_r in runs])           # (20, 17) fitted weights
print(f'exact score over 20 draws: mean squared error {errs.mean():.2f}, sd
↪  {errs.std():.3f}')
for q in (1.0, 8.0):
    s_q = W @ features(np.array([q]))[0]
    print(f'fit at x = {q:.0f} over 20 draws: from {s_q.min():+.1f} to {s_q.max():+.1f} '
          f'(exact score {4 * np.tanh(4 * q) - 2 * q:+.1f})')
```

```
exact score over 20 draws: mean squared error 2.11, sd 0.033
fit at x = 1 over 20 draws: from +1.9 to +2.1 (exact score +2.0)
fit at x = 8 over 20 draws: from -27.1 to +16.2 (exact score -12.0)
```

20 回の抽出にわたって、厳密なスコアの平均二乗誤差の標準偏差は 0.03〜0.05 となる。したがって、1 組のペア標本から下限を見積もれる精度は小数第 2 位で数単位までが限界である（演習問題 3 では単一の標本から同じ数値を導く）。$x = 1$ では、20 回の当てはめはすべて厳密なスコア $+2.0$ から約 0.2 以内に収まる。一方、$x = 8$ では数十単位にわたってばらつき、正負両方の符号をとる。抽出によっては当てはめたスコアが正となり、データから遠ざかる向きを指すことすらある。回帰はペアを 1 つも観測しなかった場所について何の根拠も持たないにもかかわらず、そこでも平然と数値を返す。最小二乗法はペアが落ちる場所でのみ当てはめを制約するため、データから遠く離れた位置から出発するサンプラは、そこで恣意的なベクトル場に出くわすことになる。

38.6.2　スコアを用いたサンプリング

前節のサンプラは、順過程（forward process）のあらゆるノイズレベルにおけるスコアを必要とした。しかし 1 つのレベルのスコアしかなくても、あるいは p 自身の厳密なスコアしかなくても、順過程を一切必要としない手法によってサンプルを抽出できる。この手法には、1908 年に流体中の粒子を力とランダムな衝撃の和として記述した Paul Langevin の名が冠されている。

38.6.2.1 ランジュバン動力学

対数密度（log-density）の坂を上向きにドリフトしながら、ノイズに揺さぶられる粒子を想像してほしい。ドリフトは粒子を高密度領域へと引き寄せ、ノイズは粒子が単一のモードへ潰れてしまうのを防ぐ。**ランジュバン動力学**（Langevin dynamics）とは、次の確率微分方程式（SDE）である。

$$d\mathbf{X} = \tfrac{1}{2}\nabla \log p(\mathbf{X})\, dt + d\mathbf{W}, \tag{38.82}$$

ここで、ドリフトはスコアの半分であり、ノイズは標準ブラウン運動 $\mathbf{W}$ である。主張したいのは、p がその定常法則（定常分布）になるということである。

ガウス分布 $p = \mathcal{N}(0, v)$（スコアは $-x/v$）については、この事実はすでに知っている。このとき Equation 38.82 はオルンシュタイン・ウーレンベック過程 $dX = -\frac{X}{2v}\,dt + dW$ となる。OU 過程 $dX = -\theta X\,dt + g\,dW$ の定常法則は $\mathcal{N}(0, g^2/(2\theta))$ であり（38.4.3 節）、ここでは $\theta = 1/(2v)$ と $g = 1$ から分散 v が得られる。次の命題は、スコアが高々線形にしか増大しない滑らかな目標分布へとこれを拡張する。

命題（定常性） *p を滑らかな正の密度とし、そのスコア $\nabla \log p$ が局所的リプシッツ連続かつ高々線形にしか増大しないことで、Equation 38.82 が爆発しない一意解を持ち、対応するフォッカー・プランク方程式も各初期密度に対して一意解を持つとする。このとき $\mathbf{X}_0 \sim p$ ならば、$\mathbf{X}_t \sim p$ がすべての $t \geq 0$ に対して成り立つ。*

証明単位拡散を持つ SDE $d\mathbf{X} = \mathbf{b}\,dt + d\mathbf{W}$ に対して、密度 q_t（$\mathbf{X}_t$ の密度）はフォッカー・プランク方程式 $\partial_t q = -\nabla \cdot (q\,\mathbf{b}) + \frac{1}{2}\Delta q$ を満たす（38.5.2 節）。ここに $\mathbf{b} = \frac{1}{2}\nabla \log p$ と $q = p$ を代入し、$p\,\nabla \log p = \nabla p$ を用いると、次のようになる。

$$-\nabla \cdot \big(\tfrac{1}{2}\, p\, \nabla \log p\big) + \tfrac{1}{2}\Delta p = -\tfrac{1}{2}\nabla \cdot (\nabla p) + \tfrac{1}{2}\Delta p = 0.$$

ゆえに $q_t \equiv p$ は初期密度 p に対する方程式の解である。解の一意性により、「p が方程式を満たす」ことから「$\mathbf{X}_t$ の法則は p である」ことが従う。■

ドリフトが対数密度の坂を上向きに確率質量を運ぶ速度は、ノイズがそれを広げる速度とちょうど釣り合っている。ただし定常性だけでは、別の分布から出発した連鎖についてはまだ何も言えず、実際のサンプラは常に別の場所から出発する。上の命題のような目標分布（滑らかで正であり、スコアが高々線形に増大する分布）に対しては、$\mathbf{X}_t$ の法則は任意の出発点から確かに p へと収束する (Roberts and Tweedie 1996)。ここではその証明は省略する。収束にどれだけの時間がかかるかはまた別の問題であり、以下の実験でそれを測定する。

サンプリングするには離散化する。Equation 38.82 に対するステップサイズ h のオイラー・丸山ステップが**ランジュバンサンプラ**（Langevin sampler）である。

$$\mathbf{x} \leftarrow \mathbf{x} + \tfrac{h}{2}\,\mathbf{s}(\mathbf{x}) + \sqrt{h}\,\xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

これに必要なのは、厳密または学習されたスコア $\mathbf{s}$ だけである。ただし離散化された連鎖は、元の SDE の定常性をそのまま受け継ぐわけではない。ガウス分布の場合、1 ステップは線形漸化式 $x \leftarrow (1 - \frac{h}{2v})\,x + \sqrt{h}\,\xi$ となり、その定常分散 V は $V = (1 - \frac{h}{2v})^2 V + h$ を満たす。したがって $V = v/(1 - h/(4v))$ となり、サンプラは分散を約 $h/4$ だけ過大評価する（演習問題 1 でこれを測定する）。

ガウス分布は単峰性であり、モードが 1 つしかないと本質的な困難が隠れてしまう。そこで、38.5.5 節の非対称な混合分布 $p = 0.8\,\mathcal{N}(-2, 0.5^2) + 0.2\,\mathcal{N}(2, 0.5^2)$ をサンプラに与

え、さらに当てはめではなく p の厳密なスコアを渡して、問題が起きればすべてサンプラ自身のせいであるようにしよう。p の確率質量の 5 分の 1 はゼロより右側にあり、重みによらず $\mathbb{E}X^2 = 4 + 0.25 = 4.25$ である。10,000 本の連鎖をステップサイズ $h = 0.01$ で 2,000 ステップ走らせる実験を、広い初期分布 $\mathcal{N}(0, 3^2)$ から、左の平均から、そして右の平均からの 3 通りで行う。出力を読む前に、広い初期分布から出発した連鎖のうちどれだけの割合がゼロより右側に落ち着くか、また一方のモードから出発した連鎖が何本もう一方のモードへ移るかを予想してみてほしい。

```
def langevin(q, h, steps, var=0.25, ws=(0.8, 0.2)):   # var: component variance of p
    for _ in range(steps):
        xi = rng.standard_normal(q.shape)
        q = q + 0.5 * h * mixture_score(q, var, ws=ws) + np.sqrt(h) * xi
    return q
warm = langevin(rng.normal(0.0, 3.0, 10000), 0.01, 2000)
print(f'broad start: P(X > 0) = {(warm > 0).mean():.3f}, '
      f'E[X^2] = {(warm**2).mean():.2f}  (truth 0.2, 4.25)')
for start, w in ((-2.0, 0.8), (2.0, 0.2)):              # every chain in one mode
    q = langevin(np.full(10000, start), 0.01, 2000)
    print(f'all chains at {start:+.0f} (weight {w}): '
          f'{(np.sign(q) != np.sign(start)).mean():.3f} end on the other side')
```

```
broad start: P(X > 0) = 0.477, E[X^2] = 4.25  (truth 0.2, 4.25)
all chains at -2 (weight 0.8): 0.006 end on the other side
all chains at +2 (weight 0.2): 0.025 end on the other side
```

10,000 本のうち重いモードから出発した連鎖で反対側に到達したのは、およそ 200 本に 1 本にすぎない。軽いモードから出発した連鎖でも、およそ 40〜50 本に 1 本である。谷では密度がごく小さく、ドリフトは連鎖がもともといたモードの側へと引き戻す向きを指す。谷を越えるには一方向にノイズが連続するというまれな事象が必要であり、その待ち時間は $-\log p$ の障壁の高さに対して指数関数的に増大する。

2 つの割合が異なるのは当然である。平衡状態では左から右へ流れる質量と右から左へ流れる質量が等しくなるため、重いモードの 4 分の 1 の質量しか持たない軽いモードは、4 倍の頻度で連鎖を手放さなければならない。サンプラの遷移率の比はおおむね正しい。問題は、その絶対値があまりにも小さいことである。2 つの状態を持つ系が初期比率を忘れる速さは 2 つの遷移率の和で決まるが、出力された 2 つの割合を足しても 1 回の実行あたり約 0.03 にしかならない。したがって、比率の誤差が e 分の 1 に縮むには $2{,}000/0.03 \approx 67{,}000$ ステップ、すなわち今回の実行長の 30 倍以上を要する。

これを踏まえると、1 行目の出力の見方が変わる。二次モーメントは 4.25 に小数第 2 位で数単位の精度で一致しており、これは 10,000 本の連鎖の標本誤差の範囲内である（ステップサイズによる統計的バイアス $h/4 \approx 0.0025$ はそれよりはるかに小さい）。各連鎖は、自分が滑り込んだモードの内部では平衡に達している。しかし、ほぼ半数の連鎖がゼロより右側に落ち着いており、そこには本来 p の質量の 5 分の 1 しかない。初期分布が対称であるため、各連鎖は谷に対して自分がいた側のモードへと滑り込み、そこからほとんど出ていかない。結果の比率は初期分布の比率であって、p の比率ではないのである（17.1 節では、3 つのモード上で学習されたスコアを用いて同じ不感性を測定する）。

連鎖がどこから出発したかを忘れるまでに要する時間を**混合時間**（mixing time）と呼ぶ。混合の遅さは SDE 自身の性質であるため、ステップサイズを小さくしても解決しない。ステップサイズを小さくして取り除けるのは、離散化による統計的バイアスだけである。

38.6.2.2 単一ノイズレベルの限界

単一ノイズレベルのスコアは、ここまでで 3 つの失敗を見せた。第 1 に、ノイズ付きペアが落ちなかった領域では、学習されたスコアは恣意的になる。第 2 に、正しいスコアを与えても、ランジュバン動力学はモード間の谷をほとんど越えられない。第 3 に、得られるサンプルは初期分布の比率を引き継ぎ、5 分の 1 しか質量がないはずのモードにほぼ半分の質量が集まってしまう。3 つ目の失敗はサンプラだけの責任ではない。よく離れたモードからなる混合分布が質量を持つ領域では、重みが何であれ、そのスコアは最も近いモード単独のスコアとほぼ変わらない（37.5.1 節）。すなわち低ノイズでは、スコア自体に比率の情報がほとんど刻まれていないのである。

38.5.5 節の逆時間サンプラは、同じ非対称な混合分布に対して同様に対称な初期分布から出発しながら、質量の 5 分の 4 を正しく左側へ戻した。このサンプラは、ランジュバン動力学に欠けているもの、すなわちあらゆるノイズレベルにおけるスコアを持っていたのである。比率が決着したのは中程度のノイズ領域であった。そこでは 2 つの成分がまだ重なり合いつつ引き寄せる力にはすでに差が生じているため、混合重みが粒子の行き先を決定する。

減少するノイズレベルの系列（sequence）を用いれば、3 つの限界すべてに対処できる。大きな σ ではノイズ付加密度に谷がなく、ノイズ付きペアはサンプラが訪れる領域全体を覆う。対称な混合分布の場合、p_σ の成分分散は $0.25 + \sigma^2$ であり、成分の標準偏差が中心から各平均までの距離 2 に達した時点（前節の演習問題 4）、すなわち $\sigma \geq 1.94$ で単峰性になる。重みが不均等であれば、谷はさらに早く消える。そこから σ が縮むにつれて細部の構造が戻り、すでに両モードへ質量を配分し終えたサンプラはその分布を精緻化するだけでよい。

アニーリング付きランジュバン動力学（annealed Langevin dynamics）は、各レベルに専用のスコアを用意してこのような系列に沿ってサンプラを走らせる手法である (Song and Ermon 2019)。演習問題 2 ではこの方法で単一モードに捕らわれた連鎖を解放し、17.2 節ではこのような系列が不均等な重みを復元することを示す。しかしノイズレベルを 1 つずつ選ぶやり方には、レベル間隔をどう決めるか、あるレベルから次のレベルへどう移るかという 2 つの問いが残る。順過程はこの両方を一挙に解決する。連続的なノイズレベルを供給し、その逆過程がそのままサンプラになるからである。

38.6.3 複数ノイズレベルにわたるスコア

ここから先、モデルはノイズレベルを第 2 引数として受け取る単一のベクトル場 $\mathbf{s}_\theta(\mathbf{x}, t)$ となり、次項で導入する損失によってすべてのレベルで同時に訓練される。サンプラは冒頭で挙げた 2 つ、すなわち 38.5.5 節の逆時間 SDE と 38.5.4.3 節の確率フロー ODE である。

38.6.3.1 順過程とノイズ条件付き損失

DSM にはクリーンな点が与えられたもとでのノイズ付きの点の条件付き法則が必要であり、目標が陽に書けるためにはその法則がガウス分布でなければならない。ドリフトが $\mathbf{x}$ に関して線形であり、ノイズの振幅が $\mathbf{x}$ に依存しないすべての SDE は、38.4.3 節で OU 過程について示した通りガウス遷移カーネルを持つ。38.4 節の 2 つの標準的な順過程はいずれもこの種のものであり、そのカーネルは両者共通の形で次のように書ける。

$$p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\alpha_t \mathbf{x}_0,\, \sigma_t^2 \mathbf{I}), \qquad \text{i.e.} \qquad \mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon.$$

係数 α_t は信号を縮小し、σ_t はノイズのスケールを定める。2 つの過程の違いは、この 2 つの数値だけである。

- **分散爆発**（variance-exploding、VE）SDE $d\mathbf{X} = \sqrt{\frac{d}{dt}\sigma^2(t)}\, d\mathbf{W}$（ただし $\sigma^2(t)$ は単調増加かつ $\sigma(0) = 0$）は、データを縮小せずにノイズを加える過程であり、$\alpha_t = 1$、$\sigma_t = \sigma(t)$

となる。これはアニーリング付きランジュバン動力学が降りていくノイズレベルの系列を連続化したものである。

- **分散保持**（VP）SDE は時間依存のレートを持つオルンシュタイン・ウーレンベック過程

$$d\mathbf{X} = -\tfrac{1}{2}\beta(t)\,\mathbf{X}\,dt + \sqrt{\beta(t)}\,d\mathbf{W},$$

であり、$\alpha_t = e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}$ かつ $\sigma_t^2 = 1 - \alpha_t^2$ となるため、単位分散のデータは任意の時刻 t で単位分散を保つ（38.4 節）。

本節の残りの部分では VP 過程を用いる。VE 過程は、DDIM が単純な形になる座標系としてあとで 1 度だけ再登場する。

任意の t でカーネルがガウス分布であるため、条件付きスコアは $-\epsilon/\sigma_t$ となり、各レベルで DSM がそのまま適用できる。重み $w(t) > 0$ をかけてレベル全体にわたり平均をとると、**ノイズ条件付き DSM 損失**（noise-conditional DSM loss）が得られる。

$$\mathcal{L}(\theta) = \tfrac{1}{2}\,\mathbb{E}_t\, w(t)\, \mathbb{E}_{\mathbf{x}_0,\,\epsilon}\left[\left\| \mathbf{s}_\theta(\alpha_t \mathbf{x}_0 + \sigma_t \epsilon, t) + \frac{\epsilon}{\sigma_t} \right\|^2\right], \tag{38.83}$$

ここで t は訓練範囲から一様に抽出する。$(t, \mathbf{x}_0, \epsilon)$ の 1 回の抽出が 1 組の回帰ペアになる。本節の以降ではこの損失を用いる。以下で述べる重みの選び方は、初読時には細部として読み飛ばして差し支えない。

重みが決めるのは、ネットワークが何を学習するかではなく、どのレベルに最も計算資源を割くかである。任意の正の重み w に対し、ネットワークに十分な表現力があれば、各レベルでの Vincent の定理により母集団における最小点は $\mathbf{s}(\cdot, t) = \nabla \log p_t$（すべての t に対して）となる。重みが変えるのは、各レベルの寄与の大きさである。Equation 38.80 より、レベル t における下限は $d/\sigma_t^2 - \mathbb{E}\|\nabla \log p_t\|^2$ であり、これは d/σ_t^2 のオーダーで小さな σ_t において増大する。したがって、重みなしの損失は最もノイズの小さいレベルに支配されてしまう。

標準的な重み付けは 2 つある。次項の離散時間モデルは $w(t) = \sigma_t^2$ を用い、この増大を打ち消すことで、すべてのレベルが同じオーダーの損失を寄与するようにする。

もう 1 つの**尤度重み付け**（likelihood weighting）$w = g^2$（順方向 SDE の拡散係数の二乗）は、スコアの誤差が尤度を制御するようになる重みである。この重みのもとでのスコアの二乗誤差を全レベルにわたって積分すると、逆時間 SDE が定めるモデルに対するデータからの KL ダイバージェンスを上から抑えられる。ただし、p_T とサンプラの出発点であるガウス分布との不一致を表す項が別途加わる（演習問題 7）。実際に計算できる損失にはこの誤差に加えて下限が含まれるが、尤度重み付けのもとではこの下限が厄介になる。VP 過程では $g^2 = \beta$ であり、重み付き下限は d/t のオーダーで $t \to 0$ に伴って増大するため、$t = 0$ までの積分が無限大に発散する。そのため、訓練は小さな時刻 $\varepsilon > 0$ から開始する（演習問題 4）。

生成には新たな仕組みは何もいらない。厳密なスコアの代わりに学習されたベクトル場を用いて、2 つのサンプラをそのまま動かせばよい。順方向 SDE $d\mathbf{X} = \mathbf{f}(\mathbf{X}, t)\,dt + g(t)\,d\mathbf{W}$（区間 $[0, T]$ 上）に対し、p_T を近似するガウス分布から出発して、逆時間 SDE（38.5.5 節）または確率フロー ODE（38.5.4.3 節）を $t = T$ から 0 まで逆向きに積分する。

$$d\mathbf{X} = \big[\mathbf{f} - g^2\,\mathbf{s}_\theta(\mathbf{X}, t)\big]\,dt + g\,d\bar{\mathbf{W}}, \qquad \frac{d\mathbf{x}}{dt} = \mathbf{f} - \tfrac{1}{2} g^2\,\mathbf{s}_\theta(\mathbf{x}, t), \tag{38.84}$$

ここで、$\bar{\mathbf{W}}$ は時間を逆向きに走るブラウン運動である。VP 過程では $\mathbf{f} = -\frac{1}{2}\beta\mathbf{x}$ および $g^2 = \beta$ となる。

こうして生成モデルは、順過程・学習されたスコア・サンプラの 3 つ組として定式化される。Figure 38.24 は 2 次元でこの 3 要素が働く様子を示したものであり、Figure 38.21 の順方向・逆方向密度と対をなす図である。

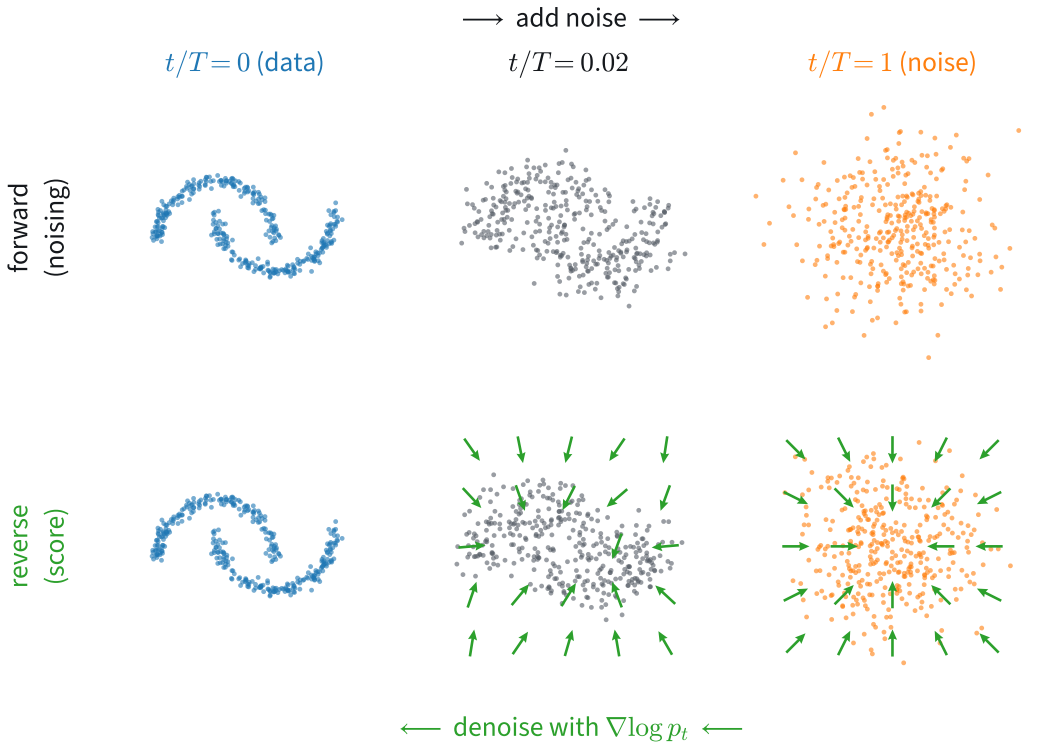


Figure 38.24: VP 順過程のもとでの 2 つの三日月（two-moons）分布（上段左から右へ $t/T = 0$、0.02、1）は、等方的なガウス分布へと近づいていく。下段は同じ周辺分布を右から左へ逆向きにたどり、短い矢印はノイズ付加データの厳密なスコア $\nabla \log p_t$ の向きを表す。高ノイズではスコアは原点への引力となり、低ノイズではデータへ向かう向きを指す。

38.6.3.2 DDPM と分散保持 SDE

ノイズ除去拡散確率モデル（denoising diffusion probabilistic model、DDPM）は離散時間で記述されており、一見すると別の構成法のように見える (Ho et al. 2020)。このモデルはノイズ付加連鎖・損失・サンプラの 3 要素からなるが、そのどれもがすでに手元にある概念に帰着する。ノイズ付加連鎖は $N = 1000$ ステップからなるマルコフ連鎖であり、スケジュール $\beta_1, \dots, \beta_N \in (0,1)$ を用いて次のように定義される。

$$\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ i.i.d.}, \tag{38.85}$$

各ステップで状態をわずかに縮小し、少量のノイズを加える。本項と次項では、t はレベル $t = 0, 1, \dots, N$ を数える添字とし、レベル t は SDE の時刻 $t\,\Delta$（ただし $\Delta = T/N$）を表すものとする。

この連鎖は VP-SDE とどう関係しているだろうか。$\beta_t = \beta((t-1)\Delta)\,\Delta$ とおくと、1 ステップは β_t の 1 次の範囲で VP-SDE のオイラー・丸山ステップ $(1 - \tfrac{1}{2}\beta_t)\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\epsilon_t$ に一致する。実はそれ以上のことが成り立ち、定数レートの場合については 38.4.4.1 節で示した通りである。すなわち、このステップはほぼ同じ長さの区間における VP 過程の**厳密な**遷移になっている。任意の 2 時刻間における VP カーネルは、時刻 0 からの式で β の積分範囲をその 2 時刻間に置き換えた形をとる。そこで $\int \beta(r)\,dr = -\log(1-\beta_t)$ となる区間をとろう。そのカーネルの平均係数は $e^{-\frac{1}{2}\int \beta} = \sqrt{1-\beta_t}$、分散は $1-(1-\beta_t) = \beta_t$ となり、Equation 38.85 の $\mathbf{x}_{t-1}$ が与えられたもとでの法則に一致する。標準スケジュールの最大ステップ $\beta_t = 0.02$ に対しても、この区間の積分値は $\int \beta = 0.0202$ であり、0.02 とほとんど変わらない。したがって DDPM の連鎖は離散時刻で観測された VP 過程にほかならず、$\mathbf{x}_t$ の（$\mathbf{x}_0$ が与えられたもとでの）法則は時刻 0 からの VP カーネルとなる。これこそが本項で記憶すべき結果であり、DDPM の訓練損失はここから 2 行で導かれる。なお、SDE を経由せずとも、DDPM 自身の記法のまま連鎖上で直接この周辺分布を確かめることもできる。

命題（$\bar\alpha$ 周辺分布） $\bar\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$ **とおく。このとき、$\mathbf{x}_0$ が与えられた条件のもとで次が成り立つ。**

$$\mathbf{x}_t = \sqrt{\bar\alpha_t}\,\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\,\epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{38.86}$$

証明 t に関する数学的帰納法による。$t = 0$ の場合は $\bar\alpha_0 = 1$ として成り立つ。$t-1$ で Equation 38.86 が成り立つと仮定し、Equation 38.85 に代入すると次のようになる。

$$\mathbf{x}_t = \sqrt{(1-\beta_t)\bar\alpha_{t-1}}\,\mathbf{x}_0 + \sqrt{(1-\beta_t)(1-\bar\alpha_{t-1})}\,\epsilon' + \sqrt{\beta_t}\,\epsilon_t,$$

ここで ϵ' と ϵ_t は独立な標準ガウス確率変数である。独立なガウス分布の和は分散の和を持つガウス分布になり、$(1-\beta_t)(1-\bar\alpha_{t-1}) + \beta_t = 1 - (1-\beta_t)\bar\alpha_{t-1} = 1 - \bar\alpha_t$ かつ $(1-\beta_t)\bar\alpha_{t-1} = \bar\alpha_t$ が成り立つ。■

前項のカーネルの記法で言えば、DDPM は $\alpha_t = \sqrt{\bar\alpha_t}$、$\sigma_t = \sqrt{1-\bar\alpha_t}$ に対応する。したがって訓練時に連鎖を逐次シミュレーションすることは一切ない。Equation 38.86 によって任意のレベルへ 1 ステップで飛び移ることができ、カーネルがガウス分布であるため DSM がそのまま適用でき、条件付きスコアは $-\epsilon/\sigma_t$ となる。ノイズ予測ネットワークを用いてスコアモデルを $\mathbf{s}_\theta(\mathbf{x}_t, t) = -\epsilon_\theta(\mathbf{x}_t, t)/\sigma_t$ とパラメータ化すると、次が成り立つ。

$$\left\| \mathbf{s}_\theta(\mathbf{x}_t, t) + \frac{\epsilon}{\sigma_t} \right\|^2 = \frac{1}{\sigma_t^2}\,\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2\,,$$

ゆえに DDPM の**単純損失**（simple loss）$\tfrac{1}{2}\mathbb{E}_{t,\mathbf{x}_0,\epsilon}\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2$ は、重みを $w(t) = \sigma_t^2 = 1-\bar\alpha_t$ としたノイズ条件付き DSM 損失 Equation 38.83 に一致する。「加えたノイズを予測せよ」とは、下限の $1/\sigma_t^2$ の増大を打ち消す重みを付けたノイズ除去スコアマッチングのことであり、どのレベルもオーダー d の損失を寄与する（(Ho et al. 2020) を読む際の注意として、原論文では α_t を $1-\beta_t$ の意味で書いており、本節の α_t が信号係数 $\sqrt{\bar\alpha_t}$ を表すのと異なるほか、原論文の単純損失には本節の係数 $\tfrac{1}{2}$ が付かない）。

第 3 の要素はサンプラである。**祖先サンプリング**（ancestral sampling）は、レベル N から 1 まで順にステップを進める。

$$\mathbf{x}_{t-1} = \frac{\mathbf{x}_t + \beta_t\,\mathbf{s}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\beta_t}} + \sqrt{\beta_t}\,\xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

ただし最後のステップではノイズを加えない。β_t の 1 次の範囲では、これは $\mathbf{x}_t + \frac{1}{2}\beta_t \mathbf{x}_t + \beta_t \mathbf{s}_\theta + \sqrt{\beta_t}\,\xi$ となり、VP 過程に対する Equation 38.84 の逆 SDE のオイラー・丸山ステップに一致する。なお、このモデルはもともと別の経路、すなわち潜在変数モデルの対数尤度に対する下界として導出された (Sohl-Dickstein et al. 2015; Ho et al. 2020)。17.6 節でたどるのがその経路であり、各レベルの重みが異なるだけで同じ回帰が得られる。

以下のサンプラでは、β_t を 10^{-4} から 0.02 まで $N = 1000$ レベルにわたり線形に変化させる標準スケジュールを用いる。VP 過程は単位分散のデータ向けに設計されているため、ここから先は混合分布を**標準化**する。すなわち $\sqrt{4.25}$ で割り、モードを $\pm 2/\sqrt{4.25} = \pm 0.970$、成分の標準偏差を 0.243 とする。サンプリングを行う前に、まずスケジュール自体を確認しておこう。3 つのレベルにおける $\bar{\alpha}_t$ と、最初および最後のレベルにおける信号に対するノイズの比 $\sqrt{(1-\bar{\alpha}_t)/\bar{\alpha}_t}$（次項で時間変数として用いる量）を出力する。

```
N = 1000
beta = np.linspace(1e-4, 0.02, N)                       # the DDPM schedule
alpha_bar = np.cumprod(1.0 - beta)
sigma_bar = np.sqrt((1 - alpha_bar) / alpha_bar)        # noise scale of x_t / sqrt(alpha_bar)
print(f'alpha_bar at t = 10, 100, 1000: {alpha_bar[9]:.3f}, {alpha_bar[99]:.3f}, '
      f'{alpha_bar[-1]:.0e}; sigma_bar from {sigma_bar[0]:.2f} to {sigma_bar[-1]:.0f}')
```

```
alpha_bar at t = 10, 100, 1000: 0.998, 0.897, 4e-05; sigma_bar from 0.01 to 157
```

信号は最初はゆっくりと薄れ、最後には完全に消え去る。$t = 100$ では $\bar{\alpha} = 0.897$ であり、ノイズ付加後の法則は依然として二峰性を保ち、モードは $\pm\sqrt{0.897}\cdot 0.970 = \pm 0.92$、成分の標準偏差は $\sqrt{0.897 \cdot 0.243^2 + 0.103} = 0.39$ である。$t = N$ では $\bar{\alpha}_N \approx 4 \times 10^{-5}$ となり、元のデータが何であったかにかかわらずサンプルは標準ガウス分布に従う。最初のレベルから最後のレベルまでの間に、信号に対するノイズの比は 0.01 から 157 へと 4 桁増大する。

38.6.4 決定論的サンプリングとガイダンス

祖先サンプリングは 1 レベルにつき 1 回、1 サンプルあたり $N = 1000$ 回ネットワークを呼び出す。同じネットワークを使って、はるかに少ない呼び出し回数でサンプルを生成できないだろうか。DDIM はそれを実現しており、その理由は ODE ソルバに関するある事実にある。まず厳密なスコアの上でその事実を証明し、次に当てはめたスコアの上で DDIM を 1 度走らせ、最後にサンプラを特定のクラスへ誘導したとき何を追跡しているのかを調べる。

38.6.4.1 ノイズ除去拡散陰的モデル

確率フロー ODE は決定論的かつ滑らかであるため、粗いレベルのグリッドでも十分に積分できるはずである。ノイズ除去拡散陰的モデル（denoising diffusion implicit model、DDIM）は、訓練済みのノイズ予測ネットワークをそのまま再利用しつつレベルを飛び越える更新則によってこれを実現する (J. Song et al. 2021)。レベル t において、ネットワークのノイズ推定値から周辺分布 Equation 38.86 を逆算することで、クリーンサンプルの推定値が次のように得られる。

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}}.$$

より前の任意のレベル $t' < t$ へ移るため、DDIM はこの推定値と**同じ**ノイズ推定値から、レベル t' の係数を用いてサンプルを再構成する。

$$\mathbf{x}_{t'} = \sqrt{\bar{\alpha}_{t'}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t'}}\,\epsilon_\theta(\mathbf{x}_t, t). \tag{38.87}$$

この更新は決定論的であり、新たなノイズは一切入らない。文字通りに読めば、この式は順方向の曲線 $\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon$（ペア $(\hat{\mathbf{x}}_0, \epsilon_\theta)$ が定める曲線）に沿ってサンプルを動かす。しかし最小二乗法で訓練されたネットワークが返すのは条件付き平均 $\mathbb{E}[\epsilon \mid \mathbf{x}_t]$ であって、そのサンプルを生成した実際のノイズではなく、しかもその推定値はレベルごとに変化する。では DDIM は一体何を積分しており、長いストライドをとるとどれほどの誤差が生じるのだろうか。

座標変換によって両方の問いに答えが出る。全体を $\sqrt{\bar{\alpha}_t}$ で割ってみよう。再スケーリングされた点 $\bar{\mathbf{x}}_t = \mathbf{x}_t/\sqrt{\bar{\alpha}_t}$ は $\mathbf{x}_0 + \bar{\sigma}_t\epsilon$ に等しく、ここで $\bar{\sigma}_t = \sqrt{(1-\bar{\alpha}_t)/\bar{\alpha}_t}$ は先ほどスケジュールとともに出力した信号に対するノイズの比である。この座標系では VP 過程は VE 過程、すなわちクリーンなデータにスケール $\bar{\sigma}$ のノイズを加えたものになり、そのスケールは最初のレベルの 0.01 から最後のレベルの 157 まで増大する。スケジュールが決めるのは、グリッドがどの $\bar{\sigma}$ の値を訪れるかだけである。なお $\bar{\mathbf{x}}_t$ と $\bar{\sigma}_t$ のバーは再スケーリングされた点とそのノイズレベルを表す記号であり、$\bar{\alpha}_t$ のバーは DDPM 由来の記法であって別の意味を持つ。

命題（DDIM はオイラー法である） 座標 $(\bar{\sigma}, \bar{\mathbf{x}})$ において、DDIM の更新（Equation 38.87）は次の形になる。

$$\bar{\mathbf{x}}_{t'} = \bar{\mathbf{x}}_t + (\bar{\sigma}_{t'} - \bar{\sigma}_t)\,\epsilon_\theta(\mathbf{x}_t, t), \tag{38.88}$$

これは常微分方程式（ODE）$d\bar{\mathbf{x}}/d\bar{\sigma} = \epsilon_\theta$ の 1 回のオイラーステップである。もし ϵ_θ が厳密なノイズ予測 $\hat{\epsilon}(\mathbf{x}_t, t) = \mathbb{E}[\epsilon \mid \mathbf{x}_t]$ であれば、この ODE は座標 $(\bar{\sigma}, \bar{\mathbf{x}})$ で書かれた VP 過程の確率フロー ODE に一致する。

証明ステップ (i) は単なる代数変形であり、実質的な内容は確率フロー ODE が新しい座標でこの形をとることを示す (ii) と (iii) にある。(ii) において、t は SDE の連続時間を表す。

(i) Equation 38.87 を $\sqrt{\bar{\alpha}_{t'}}$ で割り、$\hat{\mathbf{x}}_0 = \bar{\mathbf{x}}_t - \bar{\sigma}_t\epsilon_\theta$ を代入すると、$\bar{\mathbf{x}}_{t'} = \hat{\mathbf{x}}_0 + \bar{\sigma}_{t'}\epsilon_\theta = \bar{\mathbf{x}}_t + (\bar{\sigma}_{t'} - \bar{\sigma}_t)\epsilon_\theta$ となる。

(ii) Equation 38.84 の VP 確率フロー ODE $d\mathbf{x}/dt = -\frac{1}{2}\beta\mathbf{x} - \frac{1}{2}\beta\nabla_{\mathbf{x}}\log p_t$ において、$\alpha_t = \sqrt{\bar{\alpha}(t)}$、$\bar{\alpha}(t) = e^{-\int_0^t \beta}$ として変数変換する。$\bar{p}_t$ を $\bar{\mathbf{x}}_t$ の密度とすると、$\nabla_{\mathbf{x}}\log p_t = \nabla_{\bar{\mathbf{x}}}\log \bar{p}_t/\sqrt{\bar{\alpha}}$ である。$d\sqrt{\bar{\alpha}}/dt = -\frac{1}{2}\beta\sqrt{\bar{\alpha}}$ より縮小項どうしが打ち消し合い、$d\bar{\mathbf{x}}/dt = -\frac{\beta}{2\bar{\alpha}}\nabla\log\bar{p}_t$ となる。また $\bar{\sigma}^2 = 1/\bar{\alpha} - 1$ より $d\bar{\sigma}/dt = \beta/(2\bar{\alpha}\bar{\sigma})$ であり、両式の比をとれば $d\bar{\mathbf{x}}/d\bar{\sigma} = -\bar{\sigma}\,\nabla\log\bar{p}(\bar{\mathbf{x}})$ が得られる。

(iii) カーネル $\mathcal{N}(\mathbf{x}_0, \bar{\sigma}^2\mathbf{I})$ に対する Tweedie の公式は $\mathbb{E}[\mathbf{x}_0 \mid \bar{\mathbf{x}}] = \bar{\mathbf{x}} + \bar{\sigma}^2\nabla\log\bar{p}$ であるため、$\mathbb{E}[\epsilon \mid \bar{\mathbf{x}}] = (\bar{\mathbf{x}} - \mathbb{E}[\mathbf{x}_0 \mid \bar{\mathbf{x}}])/\bar{\sigma} = -\bar{\sigma}\nabla\log\bar{p}$ となり、(ii) の右辺に一致する。$\bar{\mathbf{x}}_t$ による条件付けは $\mathbf{x}_t$ による条件付けと同じである。■

したがって DDIM は、ノイズレベルを時間とする時計のもとでの、確率フロー ODE に対する 1 次ソルバである。その誤差はオイラー法の誤差にほかならない。すなわち、1 回のストライドは ϵ_θ を一定に固定する一方、真の $\hat{\epsilon}$ は軌跡に沿って変化する。次の 2 つの系は、この誤差の両極端を示す。

系（点質量は誤差なく積分される） データが単一の点 $\mathbf{x}_0$ であるならば、厳密な $\hat{\epsilon}$ を用いた DDIM のストライドは、その長さにかかわらず確率フローの軌跡上に厳密に着地する。この場合 $\hat{\epsilon} = (\bar{\mathbf{x}} - \mathbf{x}_0)/\bar{\sigma}$ であり、$d\bar{\mathbf{x}}/d\bar{\sigma} = (\bar{\mathbf{x}} - \mathbf{x}_0)/\bar{\sigma}$ の解に沿って比 $(\bar{\mathbf{x}} - \mathbf{x}_0)/\bar{\sigma}$ は一定に保たれる。右辺が定数である微分方程式に対するオイラーステップは厳密である。

系（ガウスデータは縮小する） データ $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ に対して厳密なフローは恒等写像であるが、厳密なノイズ予測を用いた 1 回の DDIM ストライドはすべてのサンプルに次の因子を掛ける。

$$\sqrt{\bar{\alpha}_t \bar{\alpha}_{t'}} + \sqrt{(1-\bar{\alpha}_t)(1-\bar{\alpha}_{t'})} = \cos(\phi_t - \phi_{t'}), \qquad \bar{\alpha}_t = \cos^2 \phi_t,$$

正の長さを持つあらゆるストライドに対して、この値は 1 より小さい。VP 過程はこのようなデータをそのまま保つ。どのレベルでも周辺分布は $\mathcal{N}(\mathbf{0}, \mathbf{I})$、スコアは $-\mathbf{x}$ であり、確率フローのドリフト $-\frac{1}{2}\beta\mathbf{x} - \frac{1}{2}\beta(-\mathbf{x})$ は 0 になるため、厳密なフローのサンプルはまったく動かない。Tweedie の公式より $\hat{\epsilon}(\mathbf{x}_t, t) = \sqrt{1-\bar{\alpha}_t}\,\mathbf{x}_t$ であり、これを Equation 38.87 に代入すると上の因子が得られる。$\bar{\alpha}_t = \cos^2 \phi_t$ とおけば、これは進んだ角度の余弦になる。バー付きの座標では厳密解が $\bar{\mathbf{x}} \propto \sqrt{1+\bar{\sigma}^2}$ となるため、解に沿って $\hat{\epsilon} = \bar{\sigma}\bar{\mathbf{x}}/(1+\bar{\sigma}^2)$ は変化する。ストライドの間これを固定してしまうことで、上の因子の分だけ信号を取り除きすぎてしまうのである。

上の系は 1 ストライドあたりの縮小率を与えている。サンプリング実行全体ではこの因子が掛け合わされるため、DDPM スケジュールを等間隔にとった 10、50、999 ストライドについてその積を計算してみよう（999 ストライドは 1,000 個のレベルを 1 つずつ進むことに対応する）。ガウス分布ではないデータでは、この誤差がどれほど残るだろうか。これを確かめるため、標準化した混合分布に対して厳密なノイズ予測 $\hat{\epsilon} = -\sqrt{1-\bar{\alpha}_t}\,\nabla \log p_t$ を用いて DDIM を走らせる。ここでレベル t におけるノイズ付加分布のモードは $\pm\sqrt{\bar{\alpha}_t}\,m$、成分分散は $\bar{\alpha}_t v_0 + 1 - \bar{\alpha}_t$ であり、クリーンな値は $m = 0.970$、$v_0 = 0.243^2$ である。同じ初期ノイズから出発した 10 ストライドおよび 50 ストライドの結果を 999 ストライドの結果と 2 通りの指標で比較する。対応する終点間の平均距離と、2 標本の経験分布関数（empirical distribution function）間の最大乖離であるコルモゴロフ・スミルノフ（KS）統計量である。

```
m, v0 = 2 / np.sqrt(4.25), 0.25 / 4.25                # standardized: modes +-m, variance v0
def ks(a, b):                                        # two-sample Kolmogorov-Smirnov statistic
    q = np.sort(np.concatenate([a, b]))
    return np.abs(np.searchsorted(np.sort(a), q, 'right') / len(a)
                  - np.searchsorted(np.sort(b), q, 'right') / len(b)).max()
def strides(K):                                       # K equal strides from level N to level 1
    return np.linspace(N - 1, 0, K + 1).round().astype(int)
for K in (10, 50, 999):                                 # the Gaussian corollary
    a = alpha_bar[strides(K)]
    print(f'N(0,1) data, {K:4d} strides: every sample times '
          f'{np.prod(np.sqrt(a[:-1] * a[1:]) + np.sqrt((1 - a[:-1]) * (1 - a[1:]))):.3f}')
def eps_hat(xt, a, ws=(0.5, 0.5)):                    # exact noise prediction of the mixture
    means = (-np.sqrt(a) * m, np.sqrt(a) * m)
    return -np.sqrt(1 - a) * mixture_score(xt, a * v0 + 1 - a, means, ws)
def ddim(z, K, eps_fn):                                 # K DDIM strides from noise z
    xt, idx = z.copy(), strides(K)
    for t1, t2 in zip(idx[:-1], idx[1:]):
        a1, a2 = alpha_bar[t1], alpha_bar[t2]
        e = eps_fn(xt, a1)
        xt = np.sqrt(a2) * (xt - np.sqrt(1 - a1) * e) / np.sqrt(a1) + np.sqrt(1 - a2) * e
    return xt
z = rng.standard_normal(8000)                         # one draw, shared by every stride count
x_ref = ddim(z, 999, eps_hat)
for K in (10, 50):
    x_K = ddim(z, K, eps_hat)
    print(f'mixture, {K:3d} strides vs 999: mean |gap| {np.abs(x_K - x_ref).mean():.3f}, '
```

```
            f'KS {ks(x_K, x_ref):.3f}')
```

```
N(0,1) data,   10 strides: every sample times 0.832
N(0,1) data,   50 strides: every sample times 0.964
N(0,1) data,  999 strides: every sample times 0.998
mixture,  10 strides vs 999: mean |gap| 0.081, KS 0.075
mixture,  50 strides vs 999: mean |gap| 0.015, KS 0.017
```

単位分散のガウスデータは、10 ストライドで 0.832 倍、50 ストライドで 0.964 倍、999 ストライドで 0.998 倍に縮小する。ノイズ予測は厳密であるため、これらの倍率は決定論的であり、純粋にオイラー法の誤差のみを測っている。ストライド数を 5 倍にすると不足分は 0.17 から 0.04 へと減り、1 次手法の理論通りの挙動を示す。混合分布でも同様である。10 ストライドの終点は 999 ストライドの結果から平均 0.08 離れており（モードが ± 0.97 にあるスケール上で）、KS 統計量も同じく 0.08 となる。50 ストライドに増やすと、両方の数値とも約 0.02 まで下がる。Figure 38.25 に両方の実験結果を示す。

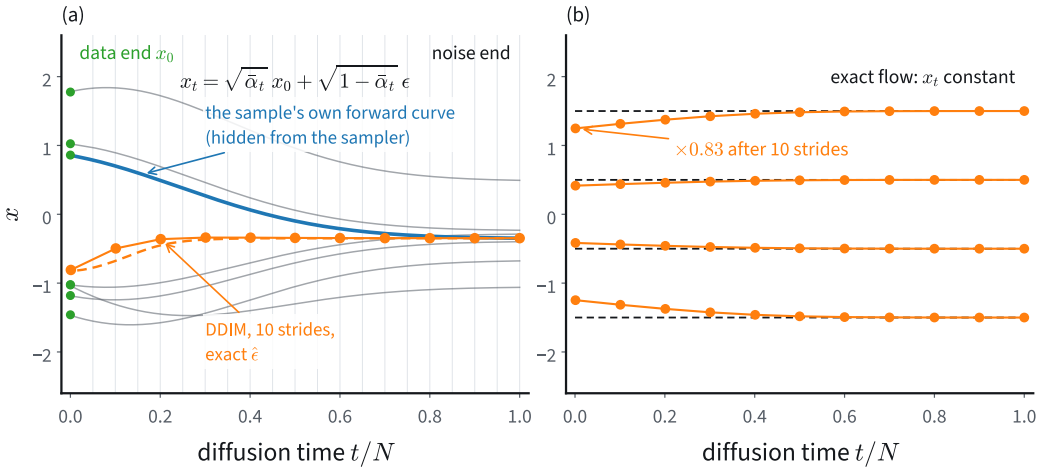


Figure 38.25: (a) 標準化した混合分布の各順方向ペア (x_0, ϵ) はそれぞれ 1 本の曲線を持ち、青いペアは谷を横切っている。同じノイズ側の終点から出発しても、厳密なノイズ予測を用いた DDIM は確率フロー ODE（10 ストライドが点、レベルごとに 1 歩ずつ進む 999 ストライドが破線）に沿って反対側のモードへ進み、そのサンプル固有の曲線を見ることは決してない。(b) 単位分散のガウスデータでは厳密なフローは恒等写像（破線）であるが、厳密なノイズ予測を用いた 10 回の DDIM ストライドはガウス分布の系が示すオイラー誤差により、すべてのサンプルを 0.83 倍に縮小する。

パネル (a) は、更新式を文字通りに読んだときの疑問に答えている。サンプルがたどるのは、そのノイズ終点を通る確率フローの軌跡であり、そのサンプルを生み出した順方向の曲線ではない。DDIM は逆過程（reverse process、38.5.5 節）の族における決定論的な極限にあたる。この族の各手法は選んだ量のノイズを再注入しつつ順過程と同じ単一時刻周辺分布を共有するため、1 つの訓練済みネットワークですべての手法に対応できる。それらの使い分けは 17.5 節で扱う。

38.6.4.2 当てはめたスコア上での DDIM

ここまでのサンプラにはすべて厳密なスコアを渡してきた。いよいよスコアを学習してサンプリングを行おう。対象はランジュバン動力学が敗れたデータ、すなわち $-m$ と $+m$ にある 2 つの

標準化モードに重みを付け直し、左が質量の 0.8、右が 0.2 を占めるようにした分布である。

モデルには単一レベルで機能した 17 個の固定特徴量をそのまま用い、今度は 50 ストライドの DDIM が訪れる 50 個の各レベルで個別に当てはめる。各レベルにおいて 4,096 個のクリーンな抽出に 4 回ずつノイズを加え、ノイズ ϵ をノイズ付きの点の特徴量に回帰する。その際、バンプが元のデータと同じように標準化データを覆うよう、特徴量は $\sqrt{4.25}\,x_t$ で評価する。あるレベルでペアが 1 つも届かないバンプには、小さな重み減衰（weight decay、2.7 節）で対処する。通常の最小二乗法では $x = 8$ での当てはめと同様にそうしたバンプの重みが不定になるが、ペナルティ項がそれをゼロに抑える。続いて前のセルの初期ノイズから DDIM を 2 回、すなわち非対称な混合分布の厳密なノイズ予測と当てはめたノイズ予測のそれぞれで走らせる。単一レベルでは学習できなかった情報を全レベルの回帰が捉えていれば、どちらの行も軽いモードの質量として 0.2 を示すはずである。

```
ws = (0.8, 0.2)                                          # the right mode is the light one
def draw(size):                                          # standardized modes +-m, weights ws
    return rng.normal(np.where(rng.random(size) < ws[0], -m, m), np.sqrt(v0))
x_pairs = np.repeat(draw(4096), 4)                       # four noisy copies per clean point
def fit_eps(a, x_pairs):                                 # ridge least squares for eps at one
 ↪ level
    e = rng.standard_normal(x_pairs.size)
    F = features(np.sqrt(4.25) * (np.sqrt(a) * x_pairs + np.sqrt(1 - a) * e))
    return np.linalg.solve(F.T @ F + 1e-6 * e.size * np.eye(17), F.T @ e)
coef = {a: fit_eps(a, x_pairs) for a in alpha_bar[strides(50)[:-1]]}   # the levels DDIM
 ↪ visits
x_exact = ddim(z, 50, lambda q, a: eps_hat(q, a, ws))
x_fit = ddim(z, 50, lambda q, a: features(np.sqrt(4.25) * q) @ coef[a])
x_true = draw(8000)
for name, xs in (('exact', x_exact), ('fitted', x_fit)):
    L, R = xs[xs < 0], xs[xs > 0]
    print(f'{name:6s} scores: light mode {(xs > 0).mean():.3f}, means {L.mean():+.2f} '
          f'{R.mean():+.2f}, widths {L.std():.3f} {R.std():.3f}, KS to p0 {ks(xs,
           ↪ x_true):.2f}')
print(f'truth: light mode 0.2, means {-m:+.2f} {m:+.2f}, widths {np.sqrt(v0):.3f}; '
      f'KS between the clouds {ks(x_exact, x_fit):.2f}')
```

```
exact  scores: light mode 0.192, means -0.96 +0.93, widths 0.227 0.248, KS to p0 0.03
fitted scores: light mode 0.180, means -0.95 +0.93, widths 0.231 0.270, KS to p0 0.04
truth: light mode 0.2, means -0.97 +0.97, widths 0.243; KS between the clouds 0.02
```

結果はまさにその通りになる。単一レベルの厳密なスコアを用いたランジュバン動力学がほぼ半分を残してしまった軽いモードに、どちらの行も 0.01〜0.02 程度の誤差でサンプルの 5 分の 1 を配分している。当てはめを用いたサンプラは、密度も、スコアも、0.2 という数値も一度も目にしていない。目にしたのは、ノイズ付きの点とそれを作ったノイズだけである。

2 つの行は同じ初期ノイズから出発しているため、共通するずれはサンプラの誤差であり、両者を隔てる差は当てはめの誤差である。両者に共通しているのは、軽いモードの平均が 0.97 より小数第 2 位で数単位小さく、重いモードの幅が約 0.23 と真の値 0.243 よりわずかに狭い点である。厳密なノイズ予測では誤差はオイラー法のものに限られるため、これが 50 ストライドの離散化誤差である。50 ストライドでガウスデータが 0.964 倍になったガウス分布の系から予想される通り、数パーセントの縮小が生じている。一方、当てはめは独自の目に見える誤差を 1 つ付け加えており、軽いモードの幅が 0.25〜0.28（厳密な行では約 0.24）に広がっている。2 つのモードの違いはデータに占める割合だけであるから、この余分な広がりはそこに起因する。最小二乗法は 17 個の重みをペアが集まる場所へ優先的に振り向けるが、ペアの 5 分の 1 しか軽いモードには落

ちない。KS 統計量もこの見方を裏づけており、どちらの行もデータの新規抽出から小数第 2 位で数単位の距離にあり、2 つの行どうしの距離もそれ以下である。

実際の拡散モデルは各レベルを別々に当てはめたりはしない。ネットワークがレベルを入力として受け取り、全レベルで重みを共有する（演習問題 8）。それでも回帰、サンプラ、そして 2 種類の誤差の構造は変わらない。

38.6.4.3 ベイズの定理によるガイダンス

p に対するサンプラは、p に含まれるあらゆる種類のサンプルを返す。しかし実際には、特定のクラスの画像やキャプションに合致する画像など、1 種類だけを求めたいことが多い。**ガイダンス**（guidance）は、サンプラが追跡するスコアを書き換えることで生成をクラス y へと誘導する。17.4 節では画像向けにガイダンス付きサンプラを構築して評価するが、ここでは混合分布で手計算により確かめられる 2 つの事実に絞る。すなわち、ガイダンスとはスコアに対するベイズの定理であり、強度を 1 より大きくすると、いかなるノイズ付加データも持ち得ない密度をサンプラが追跡するようになるという事実である。

等しい重みを持つ標準化混合分布において、y を「右のモード」とし、クラス $\mathcal{N}(m, v_0)$（ただし $m = 0.970$、$v_0 = 0.243^2$）のみからサンプルを得たいとしよう。ノイズレベル t におけるベイズの定理 $p_t(\mathbf{x} \mid y) \propto p_t(\mathbf{x})\, p_t(y \mid \mathbf{x})$ は、エビデンス $p(y)$ が $\mathbf{x}$ に依存しないため、スコアに対しては加法的な関係になる。

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x} \mid y) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \log p_t(y \mid \mathbf{x}). \tag{38.89}$$

条件付きスコアは、無条件スコアに分類器の対数確率（log-probability）の勾配を足したものであり、本節のどのサンプラもこれをそのまま入れて動かせる。**分類器ガイダンス**（classifier guidance、(Dhariwal and Nichol 2021)）は、ノイズ付き入力に対する分類器 $p_\phi(y \mid \mathbf{x}, t)$ から第 2 項をとり、それを**ガイダンス強度**（guidance scale）γ でスケール倍する。

$$\begin{aligned} \tilde{\mathbf{s}}(\mathbf{x}, t) &= \nabla \log p_t(\mathbf{x}) + \gamma\, \nabla \log p_\phi(y \mid \mathbf{x}, t), \\ \tilde{\mathbf{s}} &= \nabla \log \tilde{p}_t, \qquad \tilde{p}_t(\mathbf{x}) \propto p_t(\mathbf{x})\, p_\phi(y \mid \mathbf{x}, t)^{\gamma}. \end{aligned}$$

$\gamma = 1$ のとき、これはベイズの定理そのものである。$\gamma > 1$ のとき、誘導されたベクトル場は分類器の確率をべき乗した**傾斜**密度のスコアになる。**分類器なしガイダンス**（classifier-free guidance、CFG、(Ho and Salimans 2022)）は、Equation 38.89 を右から左へ $\nabla \log p_t(y \mid \mathbf{x}) = \nabla \log p_t(\mathbf{x} \mid y) - \nabla \log p_t(\mathbf{x})$ と読むことで分類器を不要にし、条件付きスコアモデル $\mathbf{s}_\theta(\mathbf{x}, t, y)$ と無条件スコアモデル $\mathbf{s}_\theta(\mathbf{x}, t, \emptyset)$ だけを用いる（これらは 1 つのネットワークでまかなえる。17.4.4.2 節）。

$$\tilde{\mathbf{s}}(\mathbf{x}, t) = \mathbf{s}_\theta(\mathbf{x}, t, \emptyset) + \gamma \left[\mathbf{s}_\theta(\mathbf{x}, t, y) - \mathbf{s}_\theta(\mathbf{x}, t, \emptyset) \right], \tag{38.90}$$

これは $(1 - \gamma)\, \mathbf{s}_\emptyset + \gamma\, \mathbf{s}_y$ と同値であり、$\gamma > 1$ では条件付きスコアのさらに先へと外挿する。

ある 1 つの事実が $\gamma > 1$ の解釈を制約する。傾斜密度 $\tilde{p}_t$ は一般に、**いかなる**クリーン分布のノイズ付加周辺分布にもなっていないのである。周辺分布 p_t は $\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0$ にスケール σ_t のノイズを加えた法則であるため、曲率上界 Equation 38.81 により、データが何であれ $-\nabla^2 \log p_t$ の固有値が $1/\sigma_t^2$ を超えることはない。ところが分類器を γ 乗すると、特にクラス間の中間付近で曲率が上乗せされる。この混合分布では、$\gamma > 1 + v_0 v_t / (\sigma_t^2 m^2)$ となった時点で中間付近において上界が破綻する。ここで $v_t = \bar{\alpha}_t v_0 + \sigma_t^2$ はノイズ付加されたクラスの分散である（演習問題 6）。この閾値は最もノイズの大きいレベルで 1.0625 であり、1.09（$t = 100$ のとき）にとど

まる。$\gamma = 3$ では 1,000 レベルのうち最もノイズの小さい約 10 レベルでしか上界が成り立たず、$\gamma = 10$ ではさらに少なくなる。

なぜこれが重要なのか。もし傾斜密度の列が単一のクリーン分布のノイズ付加周辺分布であったなら、そのスコア上で逆方向連鎖を走らせれば、p のときと同様にそのクリーン分布へと行き着くはずである。しかし実際にはそうではないため、ガイダンス付きの連鎖がどこに行き着くかは事前には保証されない。実際に確かめてみよう。38.6.3.2 節の祖先連鎖を厳密なスコア、すなわち混合分布の無条件スコアと単一ガウス分布 $\mathcal{N}(\sqrt{\bar{\alpha}_t}\, m,\, v_t)$ のクラス条件付きスコアを Equation 38.90 により $\gamma \in \{1, 3, 10\}$ で組み合わせたスコアの上で走らせる。

```
    def cfg_sample(gamma, n=8000):                      # ancestral chain with the guided score
        xt = rng.standard_normal(n)
        for t in range(N - 1, -1, -1):
            a, b = alpha_bar[t], beta[t]
            v = a * v0 + 1 - a                           # noised component variance
            s_uncond = mixture_score(xt, v, (-np.sqrt(a) * m, np.sqrt(a) * m))
            s_cond = (np.sqrt(a) * m - xt) / v            # class y = the right mode
            xt = (xt + b * ((1 - gamma) * s_uncond + gamma * s_cond)) / np.sqrt(1 - b)
            if t > 0:
                xt = xt + np.sqrt(b) * rng.standard_normal(n)
        return xt
    for gamma in (1.0, 3.0, 10.0):
        xs = cfg_sample(gamma)
        print(f'gamma = {gamma:4.1f}: mass right of 0 {(xs > 0).mean():.3f}, '
              f'mean {xs.mean():.3f}, std {xs.std():.3f}')
    print(f'exact conditional: mean {m:.3f}, std {np.sqrt(v0):.3f}')
```

```
gamma =  1.0: mass right of 0 1.000, mean 0.966, std 0.248
gamma =  3.0: mass right of 0 1.000, mean 1.034, std 0.231
gamma = 10.0: mass right of 0 1.000, mean 1.078, std 0.232
exact conditional: mean 0.970, std 0.243
```

$\gamma = 1$ では、連鎖は目的のクラスを忠実に再現する。無条件の混合分布では半分しか置かれないゼロより右側の領域にすべての質量が集まり、平均と標準偏差は約 0.01 以内の差で厳密値 0.970 および 0.243 に一致する。$\gamma = 3$ および 10 では、もはやモード間で移すべき質量が残っていないため、ガイダンスはモード自体をクラス境界から遠ざかる方向へ押し動かし、平均は約 1.03、さらには 1.07〜1.08 へと移動する。一方、幅はほとんど変化せず、約 5% の範囲で 0.24 付近にとどまる。

ここに曲率上界の破綻が現れている。クリーンなレベルでは 2 つのクラスは標準偏差の 8 倍離れていて分類器は確信を持っているため、そこでの傾斜密度は平均 0.970（任意の γ に対して）のクラスそのものである。単一の順過程の周辺分布をたどる連鎖であれば、そこへ着地するはずである。しかし $\gamma = 3$ と 10 で走らせた連鎖は、そこから約 0.06 および 0.1 だけ右にずれた場所へ着地する。連鎖が追跡するスコアの背後にある密度たちが、単一の過程に属していないからである。このずれは、ノイズ付加されたクラスどうしが重なり合う中間のレベルで生み出される（演習問題 5）。このように $\gamma > 1$ のガイダンスは、サンプルを他のクラスから遠ざけ、本来のクラスの位置を超えた先へと押しやる。単一のガウスモードしか持たないクラスには失うべき多様性がほとんどないが、画像ではその多様性の喪失がはっきりと測定される（17.4.4.2 節）。

ガイダンスの有無にかかわらず、本節のサンプラがいずれもスコアを何度も評価するのは、その経路が曲がっているからである。ガウス分布の系が示した通り、最も単純なデータでさえ DDIM の時計のもとでは経路が曲がってしまう。ここまで順過程をガウスカーネルのために選んできたが、その経路が追跡しやすいかどうかは一度も問わなかった。ノイズからデータへと向かう経路

のうち、まっすぐなものはどれだろうか。

38.6.5 まとめ

スコアには正規化定数が不要であり、ノイズ付加データに対してはすべての訓練ペアで既知の目標 $-\epsilon/\sigma$ の条件付き平均に等しい。したがってノイズ付きペアに対する最小二乗法によってサンプルのみからスコアを学習でき、その損失はゼロではなく事後分散の下限に落ち着く。ただし単一のノイズレベルでは不十分である。データから離れた場所では当てはめが恣意的になり、ランジュバン動力学は初期分布の比率を保ったまま、本来の質量が 5 分の 1 しかないモードにほぼ半分のサンプルを残してしまう。順過程のあらゆるレベルにわたるスコアがあれば、最もよく知られた 2 つのサンプラは既存の方程式の離散化として理解できる。DDPM は離散時刻で観測された分散保持 SDE であり、重み σ_t^2 を付けたノイズ除去スコアマッチングを損失とする。DDIM は座標 $(\bar{\sigma}, \bar{\mathbf{x}})$ における確率フロー ODE に対するオイラー法であり、点質量に対しては厳密である一方、ガウスデータに対しては縮小しすぎる。全レベルで最小二乗法により当てはめたスコアの上で DDIM を走らせると、単一レベルでは復元できなかった重み 0.8 と 0.2 が正しく得られた。ガイダンスはスコアに対するベイズの定理であり、$\gamma > 1$ ではいかなるノイズ付加密度よりも鋭く曲がる密度を追跡する。

38.6.6 演習問題

1. [code] **ランジュバン離散化による統計的バイアス。** ガウス目標分布 $p = \mathcal{N}(0, v)$ に対し、Equation 38.82 のステップサイズ h を用いたランジュバンサンプラは線形漸化式 $x \leftarrow (1 - \frac{h}{2v})\, x + \sqrt{h}\, \xi$ となる。
   1. この漸化式が定常分布を持つステップサイズの範囲を求め、その分散を計算し、統計的バイアスを h の 1 次まで展開せよ。
   2. `langevin` を定義しているセルの混合分布の各モードを $v = 0.25$ の独立したガウス分布とみなし、定常状態の $\mathbb{E}[X^2]$（$h \in \{0.01, 0.1, 0.25, 0.5\}$ における値）を予測せよ。予測した各バイアスを、10,000 本の連鎖からの推定量のモンテカルロ標準誤差と比較せよ。続いて `langevin` をこれら各ステップサイズで 2,000 ステップ、初期点 `rng.normal(0.0, 3.0, 10000)`（`warm` と同じ初期点）から実行せよ。どのステップサイズから実験でバイアスを識別できるか。
   3. メトロポリス調整付きランジュバンアルゴリズム (Roberts and Tweedie 1996) は、密度比 $p(\mathbf{x}')/p(\mathbf{x})$ を必要とする採択・棄却判定によってこのバイアスを取り除く。訓練済みのスコアネットワーク $\mathbf{s}_\theta$ しか持たないサンプラがこの判定を直接適用できないのはなぜか。

2. [code] **アニーリングによる単一モードからの脱出。** `langevin` を定義しているセルでは、$0.8\,\mathcal{N}(-2, 0.5^2) + 0.2\,\mathcal{N}(2, 0.5^2)$ の一方のモードから出発した連鎖はそこにとどまり、広い初期分布から出発するとほぼ半分の質量が右側に残った。アニーリング付きランジュバン動力学は、まず平滑化された密度 p_σ（ここでは成分分散 $0.25 + \sigma^2$ の混合分布）の上で、減少するノイズレベルの系列 $\sigma = 2, 1, 0.5, 0.25, 0.1, 0$ に沿ってサンプラを走らせる。
   1. 本文で求めた通り、等しい重みのもとでは p_σ は $\sigma \geq 1.94$ で谷を持たない。重みが 0.8 と 0.2 の場合、谷はさらに早く閉じる。細かいグリッド上で `mixture_score`（`ws=(0.8, 0.2)`）の符号変化の回数を数え、谷が消える最小の σ を求めよ。上の系列のうち、谷を持たないレベルはどれか。
   2. 系列の各レベルにおいて 300 ステップずつ `langevin` を、成分分散 $0.25 + \sigma^2$ を `var` に渡して実行せよ。これを 10,000 本の連鎖を -2 から出発させた場合と、同じく 10,000 本の連鎖を $+2$ から出発させた場合のそれぞれについて行え。またステップサイズについては、固定ステップ $h = 0.01$ の場合と、$h = 0.04\,(0.25 + \sigma^2)$（値は 0.01 に

$\sigma = 0$ で一致する）の場合の 2 通りで試せ。4 回の実行それぞれの終了時点における $X > 0$ の連鎖の割合を報告せよ。出発点を忘れるのはどちらのステップ規則か。その割合は目標値 0.2 に近づくが完全には一致しない。各レベルにおけるゼロより右側の p_σ の質量を計算し、どのレベルで比率が固定されたかを述べよ。

3. Song and Ermon (2019) は各レベルのステップサイズを σ^2 に比例してスケーリングしている。演習問題 1 の線形漸化式の縮小率を用いて、大きなノイズレベルで固定ステップが失敗する理由を説明せよ。

3. [code] **ベイズリスクの下限。**●`w_ls` を当てはめるセルでは、同一の 16,384 組のノイズ付きペアにわたる平均として当てはめと厳密なスコアそれぞれの平均二乗誤差を出力し、さらに数値積分による母集団の下限を出力している。

   1. 本文では回帰のベイズリスクである下限 Equation 38.80 を、補題で $\mathbf{v} = \mathbf{0}$ とおくことで導いた。Tweedie の公式（Equation 38.79）を用いてこの下限が $\mathbb{E}\ \mathrm{tr}\,\mathrm{Cov}[\mathbf{x} \mid \tilde{\mathbf{x}}]/\sigma^4$ にも等しいことを示し、$\mathbf{x} \sim \mathcal{N}(\mathbf{0}, v\mathbf{I})$ に対して 2 つの表式が一致することを確かめよ。$\sigma \to 0$ のとき、滑らかな密度を持つデータに対して下限はどう振る舞い、互いに異なる n 個の点からなる訓練セットに対してはどう振る舞うか。その極限において、訓練セットに対する最適なノイズ除去器は何を返すか。
   2. 出力された厳密なスコアの平均二乗誤差は、母集団の下限そのものではない。その標準誤差を 2 通りの方法で計算せよ。すなわち、16,384 個のペアごとの二乗誤差を互いに独立とみなした場合と、各クリーンな点に対する 4 個のノイズ付きコピーにわたる平均 4,096 個から計算した場合である。2 つが異なるのはなぜか。また、本文で出力した 20 回の新規抽出にわたる標準偏差 `errs.std()` と一致するのはどちらか。出力された値は母集団の下限と整合しているか。さらに、同じペア上で評価した当てはめと厳密なスコアとの差が、この標準誤差よりはるかに小さくなる理由を説明せよ。

4. **ノイズレベルにわたる重み付け。**●線形スケジュール $\beta(t) = \beta_{\min} + t\,(\beta_{\max} - \beta_{\min})$（区間 $[0,1]$ 上、$\beta_{\min} = 0.1$、$\beta_{\max} = 20$）を持つ VP-SDE を考える。このとき $\bar{\alpha}(t) = \exp\big(-\beta_{\min} t - \tfrac{1}{2}(\beta_{\max} - \beta_{\min})t^2\big)$ および $\sigma_t^2 = 1 - \bar{\alpha}(t)$ となる。$\mathbb{R}^d$ 上のデータが滑らかな密度を持ち、$\mathbb{E}\|\nabla \log p_t\|^2$ は $t \to 0$ で有界にとどまると仮定する。$R(t)$ をノイズ条件付き DSM 損失 Equation 38.83 のレベル t におけるベイズリスク、すなわち下限 Equation 38.80 において σ_t を σ の代わりに置いたものとする。

   1. $R(t) \approx d/(\beta_{\min} t)$（$t \to 0$ のとき）を示せ。
   2. $w(t)\,R(t)$ の $t \to 0$ における漸近挙動を、$w = 1$、DDPM の重み付け $w = \sigma_t^2$、および尤度重み付け $w = g^2 = \beta(t)$ のそれぞれについて求めよ。$\int_0^1 w(t)\,R(t)\,dt$ が有限になるのはどの重み付けか。また、各重み付けはどのノイズレベルを重視するか。
   3. 訓練では t を $[\varepsilon, 1]$ から一様に抽出する。尤度重み付けにおいて、この抽出のもとでの $w(t)\,R(t)$ の二次モーメントが $1/\varepsilon$ のオーダーで増大することを示せ。Y. Song, Durkan, et al. (2021)（5.1 節）は代わりに、t を $\beta(t)/\sigma_t^2$ に比例する $[\varepsilon, 1]$ 上の密度から抽出し、抽出されたレベルの DSM 項に $c\,\sigma_t^2$（ただし定数 $c = \frac{1}{1-\varepsilon}\int_\varepsilon^1 \beta(r)/\sigma_r^2\,dr$）を掛けている。この推定量が尤度重み付け損失に対して不偏（統計的バイアスが 0）であること、および最適点におけるその値 $c\,\sigma_t^2\,R(t)$ が $t \to 0$ で有界にとどまることを示せ。

5. [code] **ノイズレベル別のガイダンス。**●`cfg_sample` を定義しているセルでは、γ を 1 から 10 へ引き上げると、誘導されたサンプルの平均が約 0.97 から 1.07〜1.08 へと移動した。

   1. 本文では、クリーンなレベルにおける傾斜密度はクラスそのものであると述べた。これを確かめよ。連鎖の最後のレベル、すなわち `cfg_sample` における $t = 0$ では、右モードのクラス事後確率は x のロジスティック関数になる。その式を書き下し、そのレベルにおいて $p_t(x)\,p_t(y \mid x)^\gamma$ に比例する傾斜密度の平均を $\gamma \in \{1, 3, 10\}$ に対し

て数値積分で計算せよ。各平均は m からどれだけ離れており、出力されたサンプル平均はどれだけ離れているか。

2. `cfg_sample` を変更し、$\gamma = 10$ をステップ $t \geq 500$ のみ、$200 \leq t < 500$ のみ、または $t < 200$ のみで用い、それ以外のステップでは $\gamma = 1$ を用いるようにせよ。シフトを生み出しているのはどのノイズレベル帯か。その答えを、ノイズ付加された 2 つのクラスの分離度 $\sqrt{\bar{\alpha}_t}\, m / \sqrt{v_t}$ と関連づけよ。ここで m と v_0 は変数 `m` と `v0` であり、$v_t = \bar{\alpha}_t\, v_0 + 1 - \bar{\alpha}_t$ はノイズ付加された成分分散（変数 `v`、`cfg_sample` 内の定義）である。
3. Kynkäänniemi et al. (2024) は画像モデルにおいて、ガイダンスは高ノイズレベルでは有害であり、低ノイズレベルではほぼ不要であり、その中間で有益であることを見いだしている。この 1 次元の例は、このパターンのどの部分を再現でき、どの部分を示すことができないか。

6. **誘導された密度とノイズ付加周辺分布。**●Equation 38.90 の誘導されたベクトル場 ($\gamma > 1$) は、傾斜密度 $\tilde{p}_t(\mathbf{x}) \propto p_t(\mathbf{x})\, p_t(y \mid \mathbf{x})^{\gamma}$ のスコアである。ある密度がレベル t におけるノイズ付加周辺分布であるとは、$\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sigma_t \epsilon$（ただし $\sigma_t^2 = 1 - \bar{\alpha}_t$）の法則として、何らかの $\mathbf{x}_0$ のクリーン分布から得られることをいう。

   1. Equation 38.81 をクリーンな変数 $\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0$ に適用し、ノイズ付加周辺分布 p_t が次を満たすことを示せ。

$$\nabla^2 \log p_t(\mathbf{x}) = -\frac{\mathbf{I}}{\sigma_t^2} + \frac{\bar{\alpha}_t}{\sigma_t^4} \mathrm{Cov}\left[\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}\right].$$

      クリーン分布が何であれ、$-\nabla^2 \log p_t$ のいかなる固有値も $1/\sigma_t^2$ を超えないと結論せよ。

   2. `cfg_sample` において、ノイズ付加された 2 つのクラスは $\mathcal{N}(\pm\sqrt{\bar{\alpha}_t}\, m, v_t)$ である。ここで m と v_0 は変数 `m` と `v0` であり、$v_t = \bar{\alpha}_t\, v_0 + \sigma_t^2$ はノイズ付加された成分分散である。ラベル y は右のクラスを指す。$-(\log \tilde{p}_t)''(x) = 1/v_t + (\gamma - 1)\, a_t^2\, \varsigma(a_t x)\,(1 - \varsigma(a_t x))$ が成り立つことを示せ。ただし $a_t = 2\sqrt{\bar{\alpha}_t}\, m / v_t$ であり、$\varsigma(u) = 1/(1+e^{-u})$ はロジスティック関数である。

   3. 最小の γ（$\tilde{p}_t$ が (a) の上界にある x で違反するような値）を求めよ。それをレベル $t = 0$、100、500（`alpha_bar` の添字）で評価し、さらに $t \to N$ としたときの極限を求めよ。これらの値から、`cfg_sample` の連鎖が $\gamma = 3$ において追跡している密度について何が言えるか。

7. **スコア誤差と尤度。**●●尤度重み付け $w = g^2$ のもとでは、母集団スコア損失がデータからモデルへのカルバック・ライブラー（KL）ダイバージェンスを上から抑える（Y. Song, Durkan, et al. (2021) の定理 1）。同論文では経路測度を用いてこの上界を証明しているが、本問では 38.5.2 節のフォッカー・プランク方程式から導出する。これは SDE（命題 C.1）および確率フロー ODE（定理 3.1）に対してこの経路をとった Lu, Zheng, et al. (2022) に倣ったものである。順方向 SDE $d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\, dt + g(t)\, d\mathbf{W}$ が周辺分布 p_t（区間 $[0, T]$ 上）を持つとする。逆時間 $\bar{t} = T - t$ において、厳密な時間反転とモデルはそれぞれ次で与えられる。

$$\begin{aligned} d\mathbf{x} &= \left[-\mathbf{f} + g^2\, \nabla \log p_t\right] d\bar{t} + g\, d\mathbf{B}_{\bar{t}}, \\ d\mathbf{x} &= \left[-\mathbf{f} + g^2\, \mathbf{s}_\theta(\mathbf{x}, t)\right] d\bar{t} + g\, d\mathbf{B}_{\bar{t}}, \end{aligned} \tag{38.91}$$

   ただし右辺は $t = T - \bar{t}$ で評価する。第 1 式は周辺分布 p_t を持ち、事前分布 π（$\bar{t} = 0$ の

とき）から出発した第 2 式は周辺分布 q_t を持つ。すべての密度は滑らかかつ正であり、必要なすべての部分積分が成り立つほど速く減衰すると仮定する。

1. $\bar{t}$ に関する両方のフォッカー・プランク方程式を $\partial_{\bar{t}}\rho = -\nabla\cdot(\rho\,\mathbf{u})$ の形で書き、速度場 $\mathbf{u}_p$ と $\mathbf{u}_q$ を求めよ。
2. $\frac{d}{d\bar{t}} D_{\mathrm{KL}}(p_t\,\|\,q_t) = \mathbb{E}_{p_t}\left[(\mathbf{u}_p - \mathbf{u}_q)^{\top}\nabla \log(p_t/q_t)\right]$ を示せ。
3. 右辺を $\frac{1}{2}g^2\,\mathbb{E}_{p_t}\|\mathbf{s}_\theta - \nabla\log p_t\|^2$ で上から評価し、$\bar{t}$ について積分することで次を得よ。

$$D_{\mathrm{KL}}(p_0\,\|\,q_0) \le D_{\mathrm{KL}}(p_T\,\|\,\pi) + \frac{1}{2}\int_0^T g(t)^2\,\mathbb{E}_{p_t}\left\|\mathbf{s}_\theta(\mathbf{x},t) - \nabla\log p_t(\mathbf{x})\right\|^2 dt. \tag{38.92}$$

   q_t がどのような条件を満たすとき、この導関数の上界は等号を達成するか。

4. スコアの代わりに $\mathbf{s}_\theta$ を用いた確率フロー ODE に対して (a) と (b) を繰り返せ。(c) を可能にしたどの項が欠けており、なぜスコア損失だけではもはや $D_{\mathrm{KL}}(p_0\,\|\,q_0)$ を上から抑えられないのかを説明せよ。

8. [extended] **学習されたノイズ予測。**●●`fit_eps` を定義しているセルでは、17 個の固定特徴量を用い、50 個の各レベルで個別にノイズ予測を学習した。本来の拡散モデルは全レベルに対して 1 つのネットワークを持つ。同じデータ（非対称な標準化混合分布からの抽出 `draw(size)`）に対し、DDPM スケジュール `beta` と `alpha_bar` のもとでそれを構築せよ。

   1. 使用しているタブのフレームワークにおいて、ノイズ予測ネットワーク $\epsilon_\theta(x_t, t)$（入力 $(x_t, t/1000)$、64 個の tanh ユニットからなる 2 つの隠れ層）を、学習率 3×10^{-3} の Adam を用いて毎回 256 個の新規抽出からなるバッチ上で 4,000 ステップ訓練せよ。損失には t を 1,000 個のレベルから一様に抽出する単純損失を用いること。厳密なノイズ予測 `eps_hat(xt, a, ws)` に対する二乗平均平方根誤差をいくつかのレベルにおける p_t のサンプル（複数の t）上で計算し、本文で当てはめた特徴量モデルについても当てはめた各レベルで同様に計算せよ。それぞれの誤差はどこで最大になるか。
   2. `ddim` を用いてノイズ `z` から $K = 10$ および $K = 50$ でサンプリングし、さらに 38.6.3.2 節の祖先ステップ（誘導スコアをネットワークのスコアに置き換えた `cfg_sample` のループ）でもサンプリングせよ。得られた各サンプル集合を `x_exact`、`x_fit`、`x_true` と、軽いモードの質量・2 つのモードの幅・`ks` によって比較せよ。(a) の誤差に最も敏感なサンプラはどれであり、そのサンプラのどのステップで誤差が入り込むか。
   3. $w = 1$ および尤度重み付け $w = \beta$ に対応するレベル別の重み、すなわちレベル t におけるノイズ損失に $w(t)/(1-\bar{\alpha}_t)$ を掛けた損失で再訓練し、(a) と (b) を繰り返せ。演習問題 4 は各重み付けがどのレベルを重視するかを予測しているが、実際の誤差はその予測と一致するか。

   提出物は、各重み付けと各サンプラに対する (b) の統計量をまとめた表、および (a) の誤差を t に対してプロットした図である。

38.7 フローマッチングと最適輸送

前節のサンプラは 1 ステップにつき 1 回スコアを呼び出し、逆方向の連鎖では 1,000 回、その決定論的なストライドでも数十回呼び出す。これほど多くのステップを要するのは、ノイズからデータへ至る経路が曲がっており、ソルバは曲がった経路を短い区間に刻まなければ追跡できない

からである。そうした経路は誰かが設計したものではなく、ノイズ付加過程に付随して生じたものにすぎない。ノイズからデータへの経路を自ら選び、ソルバが大きなステップを踏めるような経路にすることはできないだろうか。

-2 と $+2$ にある 2 つのデータ点と、最終的にそこへ到達しなければならないガウスノイズの雲を思い浮かべよう。もしすべての粒子がそれぞれの目的地へ向かって一定速度で直線上を移動するなら、長さ 1 のオイラーステップ 1 回で正確にそこへ運べるはずである。しかし、速度場は位置と時刻に対する 1 つの関数にすぎない。異なる点へ向かう粒子が同じ場所を通過しても、そこでの速度場は 1 つの向きしか指せないのである。

フローマッチング (Lipman et al. 2023) と Rectified Flow (Liu et al. 2023) は実際に経路を選ぶ。すなわち、経路をあらかじめ指定し、それに沿って質量を運ぶ速度だけを学習する。どの経路が最も低コストかという問いははるかに古く、**最適輸送**（optimal transport）の問題として知られる。1781 年に Gaspard Monge が土の山を穴へ運ぶ問題として提起したものであり、二乗距離のもとで最も低コストな運動は、まさに先ほど望んだとおり、すべての粒子が一定速度で直線上を進む運動となることがわかっている (McCann 1997; Benamou and Brenier 2000)。

単純な要素を組み合わせて作った経路であれば、どのような経路でも 1 回の回帰によってその速度を訓練できる。速度は粒子にわたる平均であり、最小二乗法（least squares）は平均を学習するからである（38.7.1 節）。それでも、直線の要素を組み合わせただけではフローは曲がってしまう。ノイズとデータのペアを組み直せばフローは真っ直ぐになり（38.7.2 節）、その再ペアリングが何を目指しているかは最適輸送が明らかにする（38.7.3 節）。なお、2 つの箇所は再読時に回してもよい。1 つは 38.7.1.3 節でガウス経路（Gaussian path）の定義に続いて述べるスコア・ノイズ・速度の相互変換であり、もう 1 つは 38.7.4 節におけるソルバの比較である。

> **!** 1 つの時計
>
> 時間は前節と同じく、$t = 0$ のデータから、今度は $t = 1$ に置かれたノイズへと進む。これはスコアでも速度でも同様である。訓練では時間を順方向に進めてデータにノイズを加え、どのサンプラも $t = 1$ から $t = 0$ へと逆向きに積分する。
> 本節で引用するフローマッチングの諸論文およびそのコードでは、時間を逆向きに、すなわち 0 のノイズから 1 のデータへと進めている（Figure 38.26）。本書の式をそれらの文献と比較するには、$1 - t$ を論文の t に代入し、論文のノイズ $\mathbf{x}_0$ を本書の ϵ、論文のデータ $\mathbf{x}_1$ を本書の $\mathbf{x}_0$ と読み替え、すべての速度の符号を反転させればよい。

データは $\mathbf{x}_0 \sim p_0$ とし、p_t は p_0 からノイズ $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ へ至る密度の経路とする。ノイズ側からのサンプルには 2 つの記号を使い分ける。前節のようにガウスノイズを表すときは ϵ と書き、任意の 2 つの端点分布に対して成り立つペアリングに関する命題では $\mathbf{x}_1$ と書く。

38.7.1 フローマッチング

38.7.1.1 確率経路と速度場

拡散モデル（diffusion model）はノイズ付加過程から経路を得ていたため、その過程を逆向きに走らせるにはスコアを学習しなければならなかった。ここでは経路自体を自由に選べる。**確率経路**（probability path）とは、密度の族 $(p_t)_{t\in[0,1]}$ であって、p_0 がデータ、p_1 がノイズとなるものである。速度場（velocity field）$\mathbf{u}_t$ がこの経路を**実現する**とは、両者の組が 38.5.4 節の連続の式（continuity equation）を満たすことをいう。連続の式は、速度場によって運ばれる密度が質量の正味の流入分だけ変化することを表す次の式である。

$$\partial_t p_t + \nabla \cdot (p_t \, \mathbf{u}_t) = 0.$$

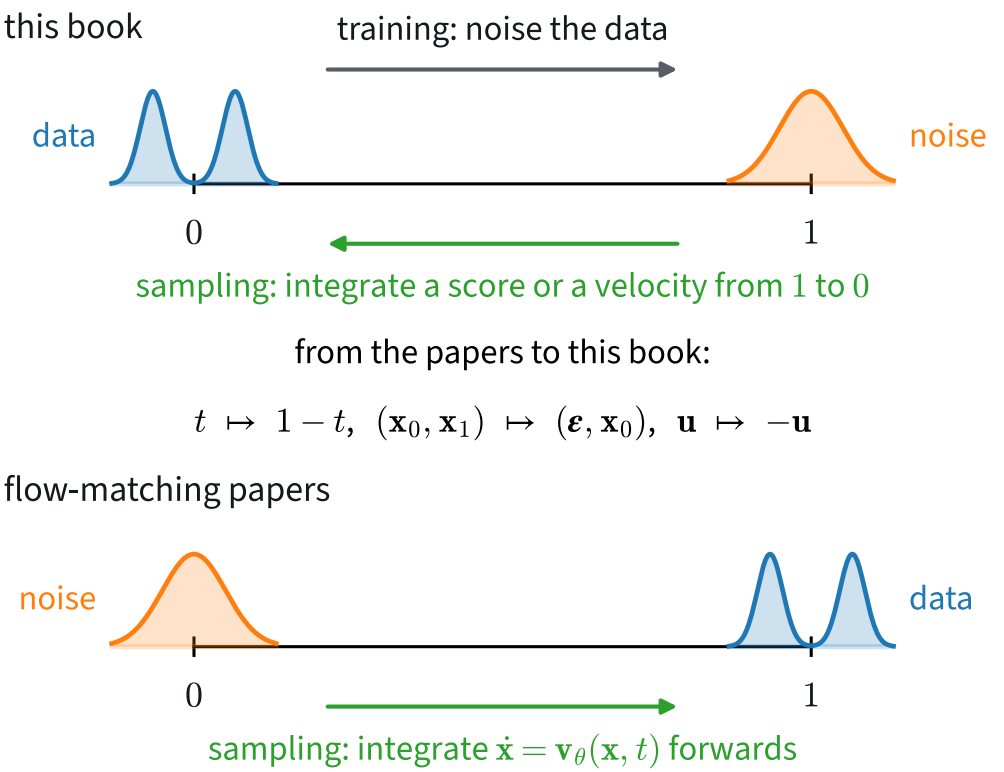


Figure 38.26: 1 つの時計と論文の時計。本書（上）はデータを $t = 0$、ノイズを $t = 1$ に置き、時間に関して順方向にデータへノイズを加え、スコアまたは速度を 1 から 0 へ積分してサンプリングする。フローマッチングの諸論文（下）はノイズを 0、データを 1 に置き、順方向にサンプリングする。両端の密度は同じであり、変数変換 $t \mapsto 1 - t$ によって一方の時計は他方の時計へと変換され、すべての速度の符号が反転する。

このとき、速度場がリプシッツ連続であれば、$\dot{\mathbf{x}} = \mathbf{u}_t(\mathbf{x})$ を p_1 のサンプルから $t=1$ より $t=0$ まで積分することで、各時刻で p_t のサンプルが、そして $t=0$ でデータのサンプルが得られる。向きに注意しよう。速度場はデータからノイズへ向かうノイズ付加の方向を指しており、サンプラはそれに逆らって進む。ネットワーク $\mathbf{u}_\theta(\mathbf{x}, t)$ が $\mathbf{u}_t$ を近似するなら、生成はノイズから常微分方程式（ODE）を解くことになる。このようなモデルは時間を逆向きに走らせた連続正規化フロー（continuous normalizing flow、38.3.3 節）であり、$t=1$ から $t=0$ へのフロー写像（flow map）が生成器（generator）となる。フローマッチングは、この ODE を 1 度も解くことなくモデルを訓練する。自然な目的関数は、次の**フローマッチング損失**（flow-matching loss）である。

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}[0,1],\, \mathbf{x} \sim p_t} \left[\left\| \mathbf{u}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x}) \right\|^2 \right]. \tag{38.93}$$

これは真の速度に対するネットワークの平均二乗誤差を、時間と雲全体にわたって平均したものである。しかし、このままでは役に立たない。$\mathbf{u}_t$ はデータ分布（data distribution）全体に依存するため、評価できない場を回帰のターゲットにしているからである。

そこで、具体的に書き下せる要素から経路を組み立てる。条件付け変数 $\mathbf{z}$ を分布 q から引くものとし、それをデータ点、あるいはノイズ点とデータ点のペアとする。$\mathbf{z}$ の各値に対し、移動する小さな分布の塊（ブロブ）$p_t(\mathbf{x} \mid \mathbf{z})$ として、$t=0$ でデータ点の上にあり、$t=1$ でノイズに行き着くものを選ぶ（$\mathbf{z}$ がデータ点単独ならノイズ全体に広がり、$\mathbf{z}$ がペアなら対応するノイズ点の上に集まる）。各ブロブには閉じた形でわかる速度 $\mathbf{u}_t(\mathbf{x} \mid \mathbf{z})$ が備わっている。経路全体はこれらブロブの混合分布 $p_t(\mathbf{x}) = \int p_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})\, d\mathbf{z}$ であり、その粒子を $\mathbf{x}_t$ と書く。すなわち、$\mathbf{z}$ を q から引き、続いて時刻 t におけるそのブロブから点を引くのである。では、この混合分布を動かす速度は何だろうか。本節のすべてはその答えの上に成り立っている。

命題（周辺速度は事後平均である）。 各 $\mathbf{z}$ に対して密度 $p_t(\cdot \mid \mathbf{z})$ が正かつ $(\mathbf{x}, t)$ について連続微分可能であり、連続微分可能な速度のもとで連続の式 $\partial_t p_t(\mathbf{x} \mid \mathbf{z}) = -\nabla \cdot \left(p_t(\mathbf{x} \mid \mathbf{z})\, \mathbf{u}_t(\mathbf{x} \mid \mathbf{z}) \right)$ を満たし、さらに以下の積分で積分記号下の微分が可能であるとする。$\{p_t > 0\}$ 上で次式により周辺速度（marginal velocity）を定義する。

$$\mathbf{u}_t(\mathbf{x}) = \mathbb{E}\left[\mathbf{u}_t(\mathbf{x} \mid \mathbf{z}) \mid \mathbf{x}_t = \mathbf{x} \right] = \int \mathbf{u}_t(\mathbf{x} \mid \mathbf{z})\, \frac{p_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})}{p_t(\mathbf{x})}\, d\mathbf{z}. \tag{38.94}$$

このとき $(p_t, \mathbf{u}_t)$ は連続の式を満たす。

位置 $\mathbf{x}$ における雲の速度は、$\mathbf{x}$ に粒子を送り込み得た各ブロブの速度を、実際にそうした尤もらしさを重みとして平均したものである。

証明。 混合分布を積分記号下で微分し、条件付きの連続の式を代入して、発散を積分の外へ出す。

$$\begin{aligned} \partial_t p_t(\mathbf{x}) &= \int \partial_t p_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})\, d\mathbf{z} \\ &= -\nabla \cdot \int p_t(\mathbf{x} \mid \mathbf{z})\, \mathbf{u}_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})\, d\mathbf{z} = -\nabla \cdot \left(p_t(\mathbf{x})\, \mathbf{u}_t(\mathbf{x}) \right). \end{aligned}$$

最後の変形では $p_t(\mathbf{x})$ を掛けて割り、Equation 38.94 の形を用いている。$\blacksquare$

1 つのブロブしか通過しない場所では、周辺速度はそのブロブ自身の速度に一致する。複数のブロブが通過する場所ではそれらの平均となり、この平均化こそがフローを曲げる原因となる。

38.7.1.2 条件付きフローマッチング定理

Equation 38.93 のターゲットは、サンプルごとに計算できる量の事後平均（posterior mean）Equation 38.94 である。回帰補題（38.6.1.2 節）により、サンプルごとのターゲットに対する最小二乗法はその条件付き平均に適合する。したがってサンプルごとの量に対して直接回帰すればよく、それを行うのが次の**条件付きフローマッチング**（conditional flow matching、CFM）損失である。

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,\,\mathbf{z}\sim q,\,\mathbf{x}\sim p_t(\cdot\mid\mathbf{z})}\left[\,\left\|\mathbf{u}_\theta(\mathbf{x},t) - \mathbf{u}_t(\mathbf{x}\mid\mathbf{z})\right\|^2\right]. \tag{38.95}$$

この損失に必要なのは、$(t,\mathbf{z},\mathbf{x})$ のサンプルと閉じた形の条件付き速度（conditional velocity）だけである。

定理（CFM は周辺速度場を訓練する）。$\mathbf{u}_t$ を周辺速度 Equation 38.94 とし、$\mathbb{E}\|\mathbf{u}_t(\mathbf{x}\mid\mathbf{z})\|^2 < \infty$ と仮定する。このとき次が成り立つ。

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathcal{L}_{\mathrm{FM}}(\theta) + C$$

ここで C は θ に依存しない定数である。特に、これら 2 つの目的関数は同じ勾配と同じ最小点（minimizer）を持つ (Lipman et al. 2023; Tong et al. 2024)。

証明。 $X=(\mathbf{x},t)$（ただし $\mathbf{x}\sim p_t(\cdot\mid\mathbf{z})$）と、仮定より二乗可積分な $Y=\mathbf{u}_t(\mathbf{x}\mid\mathbf{z})$ に対して回帰補題を適用する。X の分布は $t\sim\mathcal{U}[0,1]$ かつ $\mathbf{x}\sim p_t$ であり、Y の X が与えられたもとでの条件付き平均は、定義 Equation 38.94 より周辺速度である。補題により $\mathcal{L}_{\mathrm{CFM}}$ は $\mathbb{E}\|\mathbf{u}_\theta(X)-\mathbf{u}_t(\mathbf{x})\|^2 = \mathcal{L}_{\mathrm{FM}}$ と $C=\mathbb{E}\|Y-\mathbf{u}_t(\mathbf{x})\|^2$（$\theta$ を含まない）とに分解される。■

補題が本質的な役割を果たすため証明は 1 行で済み、この定理自体は二乗可積分なターゲットという以外に経路への仮定を必要としない。定理が述べているのは、回帰が正しい最小点を持つということだけである。この最小点が p_1 を p_0 へ（$t=1$ から $t=0$ まで積分したときに）運ぶことは、直前の命題が保証している。定数 C はターゲットの条件付き分散（conditional variance）である。これは 1 つの点を通過するサンプルごとの速度がどれだけ食い違っているかを測る量であり、したがって完璧な速度場であっても CFM 損失は 0 ではなく C に等しくなる。この C を損失の**下限**（floor）と呼ぶ。ノイズ除去スコアマッチング（denoising score matching）も同じ構造をしており、ノイズ付加カーネルのスコアがサンプルごとの量、周辺スコアがその事後平均にあたる。

この損失にはソルバが現れず、訓練中に軌跡（trajectory）を積分することもない。38.3.1 節で 10 回のオイラーステップを通して訓練したモデルは、10 ステップではターゲットに一致したものの、より細かく積分すると外れてしまった。10 ステップのプログラムそのものに適合していたからである。これに対し、今回の回帰が適合させる対象は速度場そのものであり、ステップ数はあとから自由に選べる。

38.7.1.3 スコア、ノイズ、速度

周辺速度の命題を使うには速度が既知のブロブが必要だが、ここまではまだ具体例を挙げていなかった。最初の族はすでに手元にある。前節のガウス経路はこの種の確率経路であり、その上では速度は新しい対象ではない。拡散モデルのように、ノイズを予測するよう訓練されたネットワークがあるとする。速度を得るために別の訓練をやり直す必要があるだろうか。その必要はない。スコア、ノイズ、クリーンな点、そして速度はいずれも同一の事後平均のアフィン関数であり、ノイズを学習したネットワークは残り 3 つもすでに学習しているからである。

データ点 $\mathbf{z}=\mathbf{x}_0$ で条件付けを行い、次の経路をとる。

$$\mathbf{x}_t = \alpha_t\,\mathbf{x}_0 + \sigma_t\,\epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{38.96}$$

ここで (α_t, σ_t) は滑らかな係数であり、$(\alpha_0, \sigma_0) = (1, 0)$ かつ $(\alpha_1, \sigma_1) = (0, 1)$ を満たす。$\alpha_t \mathbf{x}_0$ の周りのブロブは幅 σ_t を持ち、$t = 0$ ではデータ点そのもの、$t = 1$ ではノイズとなる。これは前節のノイズ付加カーネルにおいて、ノイズ側の端点を $t = 1$ に置いたものにほかならない。

最小の例として、1 次元のガウスデータ $x_0 \sim \mathcal{N}(0, s_0^2)$ とコサイン経路 $\alpha_t = \cos\frac{\pi t}{2}$、$\sigma_t = \sin\frac{\pi t}{2}$ から始めよう。周辺分布は $\mathcal{N}(0, s_t^2)$（ただし $s_t^2 = \alpha_t^2 s_0^2 + \sigma_t^2$）となり、スコアは $-x/s_t^2$、速度は $u_t(x) = (\dot{s}_t/s_t)\,x$ となる。つまりガウス分布の雲は引き伸ばされるか縮むかしか起こらない（演習問題 3）。$s_0 = 2$ とすると $t = \frac{1}{2}$ で $s_t^2 = 2.5$ となり、位置 x にある粒子の速度は $-0.94\,x$ である。速度場が内側を向いているのは、ノイズ付加によってこの雲が幅 2（$t = 0$）から幅 1（$t = 1$）へと縮むためである。サンプラはこれに逆らって進み、雲を再び幅 2 へと引き伸ばす。$s_0 = 1$ のときはすべての p_t が標準ガウス分布となり、雲はまったく動かない（$u \equiv 0$）。

命題（スコア・速度恒等式）。ガウス経路 Equation 38.96 に対し、任意の時刻 t（ただし $\alpha_t, \sigma_t > 0$）および $\{p_t > 0\}$ 上で、周辺速度は周辺スコアのアフィン関数となる。

$$\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\alpha}_t}{\alpha_t}\,\mathbf{x} - \left(\sigma_t\dot{\sigma}_t - \sigma_t^2\,\frac{\dot{\alpha}_t}{\alpha_t}\right)\nabla \log p_t(\mathbf{x}). \tag{38.97}$$

速度は位置の定数倍とスコアの定数倍の和であり、その両方の係数はスケジュールだけで決まる。データはスコアを通してのみ現れる。スコアの前の括弧内は $\sigma_t^2\,\frac{d}{dt}\log(\sigma_t/\alpha_t)$ に等しく、本書で用いるすべての経路のように信号に対するノイズの比が増加している限り正となる。したがってこの恒等式をスコアについて解くこともでき、一方の場が他方の場を一意に定める。

証明。 両辺が同一の事後平均 $\hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$ のアフィン関数であることを示す。経路を微分すると条件付き速度 $\dot{\mathbf{x}}_t = \dot{\alpha}_t\mathbf{x}_0 + \dot{\sigma}_t\epsilon$ が得られ、また $\epsilon = (\mathbf{x} - \alpha_t\mathbf{x}_0)/\sigma_t$ である。Equation 38.94 に従って事後平均をとると、次式を得る。

$$\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\sigma}_t}{\sigma_t}\,\mathbf{x} + \left(\dot{\alpha}_t - \alpha_t\,\frac{\dot{\sigma}_t}{\sigma_t}\right)\hat{\mathbf{x}}_0.$$

周辺スコアは各カーネルのスコアの事後平均であり、これは前節で扱った Tweedie の公式にほかならない。カーネル $\mathcal{N}(\alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I})$ のスコアは $(\alpha_t\mathbf{x}_0 - \mathbf{x})/\sigma_t^2$ であるから、次が成り立つ。

$$\nabla \log p_t(\mathbf{x}) = \mathbb{E}\left[\frac{\alpha_t\mathbf{x}_0 - \mathbf{x}}{\sigma_t^2}\,\middle|\,\mathbf{x}_t = \mathbf{x}\right] = \frac{\alpha_t\hat{\mathbf{x}}_0 - \mathbf{x}}{\sigma_t^2},$$

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x} + \sigma_t^2\,\nabla \log p_t(\mathbf{x})}{\alpha_t}.$$

この $\hat{\mathbf{x}}_0$ を速度の式に代入し、$\mathbf{x}$ と $\nabla \log p_t$ の項を整理すれば Equation 38.97 が得られる。■

2 つの特別な場合を具体的に書き下しておこう。分散保持（variance-preserving）経路では $\alpha_t^2 + \sigma_t^2 = 1$ であるため $\sigma_t\dot{\sigma}_t = -\alpha_t\dot{\alpha}_t$ となり、スコアの係数は $\dot{\alpha}_t/\alpha_t$ に簡約されて $\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\alpha}_t}{\alpha_t}(\mathbf{x} + \nabla \log p_t(\mathbf{x}))$ となる。前節の分散保持型の確率微分方程式（SDE）では $\alpha_t = \exp\big(-\frac{1}{2}\int_0^t \beta(s)\,ds\big)$ であるから $\dot{\alpha}_t/\alpha_t = -\frac{1}{2}\beta(t)$ となり、速度は $-\frac{1}{2}\beta(t)\big(\mathbf{x} + \nabla \log p_t(\mathbf{x})\big)$ となる。これは確率フロー速度 $\mathbf{f} - \frac{1}{2}g^2\nabla \log p_t$（38.5.4 節）において $\mathbf{f} = -\frac{1}{2}\beta\,\mathbf{x}$ かつ $g^2 = \beta$ と

したものに一致する。この経路上では、フローマッチングの速度と確率フロー ODE（probability-flow ODE）は同じ時計の上でまったく同じ場になる。先のガウス分布の例で用いたコサイン経路も分散保持であるため、この式を確かめられる。実際、$\nabla \log p_t = -x/s_t^2$ を代入すると $\frac{\dot\alpha_t}{\alpha_t}(1 - 1/s_t^2)\,x = \alpha_t \dot\alpha_t (s_0^2 - 1)\,x/s_t^2$ となり、上で求めた伸縮率 $\dot s_t / s_t$ が得られる。

本節の残りで用いる直線経路 $\alpha_t = 1 - t$、$\sigma_t = t$ では、この恒等式は $\mathbf{u}_t(\mathbf{x}) = -\big(\mathbf{x} + t\,\nabla \log p_t(\mathbf{x})\big)/(1-t)$ となる。

実際の訓練でネットワークのターゲットとして使われる諸量も、すべてこの同じ事後平均に基づいている（Table 38.2）。

Table 38.2: ガウス経路上におけるネットワークのターゲット。各行は、ネットワークが予測する量、それが推定する事後平均、およびスコアからの変換式を示す。$\mathbf{v}$ 予測という名称は既存文献に由来する。コサイン経路上では、そのターゲット $\alpha_t\epsilon - \sigma_t \mathbf{x}_0$ は速度ターゲットの $2/\pi$ 倍である。

ネットワークの予測対象	事後平均	スコア $\mathbf{s} = \nabla \log p_t$ からの変換
スコア $\mathbf{s}_\theta$	$\nabla \log p_t(\mathbf{x})$	$\mathbf{s}$
ノイズ $\hat\epsilon$	$\mathbb{E}[\epsilon \mid \mathbf{x}_t = \mathbf{x}]$	$-\sigma_t\,\mathbf{s}$
クリーンな点 $\hat{\mathbf{x}}_0$	$\mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$	$(\mathbf{x} + \sigma_t^2\,\mathbf{s})/\alpha_t$
速度 $\mathbf{u}_\theta$	$\mathbb{E}[\dot{\mathbf{x}}_t \mid \mathbf{x}_t = \mathbf{x}]$	Equation 38.97 の恒等式
$\mathbf{v}$ 予測 $\hat{\mathbf{v}}$	$\mathbb{E}[\alpha_t\epsilon - \sigma_t\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$	$-\frac{\sigma_t}{\alpha_t}\,\mathbf{x} - \sigma_t\,\frac{\alpha_t^2+\sigma_t^2}{\alpha_t}\,\mathbf{s}$

どの行も同一の事後平均に関する t 依存のアフィン関数であるため、いずれか 1 つの行で訓練されたネットワークは他のすべてを定める。どの行が最も学習しやすいかは別の問題であり、経路の両端における訓練ターゲットの分散に関わる（演習問題 6）。したがってノイズ予測器もまた 1 つの速度場であり、Table 38.2 の変換を施せば本節のサンプラをそのまま適用できる。

38.7.1.4 例：1 次元上の 2 点

経路の選び方を議論する前に、1 つの例を最後まで計算してみよう。冒頭で挙げた 2 点のデータセットでは、ここまで登場したすべての対象が閉じた形で求まり、38.7.2 節が修復することになる失敗の原因もその式にはっきり現れる。データは $x_0 = \pm a$（それぞれ確率 $\frac{1}{2}$、$a = 2$）とする。これは本章を通じて用いている混合分布の 2 つのモードから幅を取り除いたものである。ノイズは $\epsilon \sim \mathcal{N}(0,1)$ とし、x_0 とは独立に引く。ここで $\alpha_t = 1 - t$、$\sigma_t = t$ のガウス経路をとると、各データ点はそのノイズ点と一定速度の直線で結ばれる。すなわち $x_t = (1-t)\,x_0 + t\,\epsilon$ であり、速度は $\dot x_t = \epsilon - x_0$ となる。

データ点を条件付け変数 $\mathbf{z} = x_0$ としよう。x_0 が与えられたとき、位置 x_t は $\mathcal{N}\big((1-t)\,x_0, t^2\big)$ に従い、$t > 0$ で正の密度を持つため周辺速度の命題が適用できる。$\epsilon = (x - (1-t)\,x_0)/t$ を消去すると、速度は現在位置の関数として $u_t(x \mid x_0) = (x - x_0)/t$ と表される。時間を逆向きに走らせると、粒子は $t = 0$ でちょうどデータ点に到着する速さで目的地へ向かう。

この速度を平均するには、x_0 の位置 x における事後分布（posterior distribution）が必要である。$+a$ 対 $-a$ の位置 x における事後オッズは、2 つのガウス密度の比になる。指数部では x^2 と $(1-t)^2 a^2$ の項が打ち消し合って交差項だけが残るため、オッズは $\exp\big(2(1-t)\,ax/t^2\big)$ となり、事後平均は $\mathbb{E}[x_0 \mid x_t = x] = a \tanh\big((1-t)\,ax/t^2\big)$ となる。したがって周辺速度は次式で与えられる。

$$u_t(x) = \frac{x - a\tanh\!\big(\frac{(1-t)\,ax}{t^2}\big)}{t}. \tag{38.98}$$

時間を逆向きに走らせると、位置 x にある雲は、x から見てそれぞれのターゲットがどれだけ尤もらしいかで重み付けした平均点へ向かって、時間どおりに到着する速さで進む。$(t, x) = (0.5, 0.5)$ では速度は -2.86 であり、$(0.7, -1)$ では 0.97 である。Equation 38.98 からは 3 つの性質が読み取れ、Figure 38.27 はそのうちの最初の 2 つを描いている。

- **時刻 $t = 1$ では場は $u_1(x) = x$ となる。** ノイズサンプルはターゲットに関する情報をまったく持たないため、事後平均はデータの平均である 0 になる。したがって $t = 1$ から $t = 0$ への 1 回のオイラーステップ $x \mapsto x - u_1(x)$ は、**すべての**サンプルを 0 へ送ってしまう。これは独立ペアリングのもとでは任意のデータに対して成り立つ。すなわち $\mathbf{u}_1(\mathbf{x}) = \mathbb{E}[\epsilon - \mathbf{x}_0 \mid \epsilon = \mathbf{x}] = \mathbf{x} - \mathbb{E}[\mathbf{x}_0]$ であるため、厳密な速度場による 1 ステップは必ずデータ平均の上に着地する。
- **原点は最初引き寄せ、のちに反発する。** t が減少するときサンプラは u_t に逆らって動くため、原点での正の傾きは粒子を原点へ引き寄せ、負の傾きは原点から押し出す。傾きは $\partial_x u_t(0) = \big(1 - (1-t)\,a^2/t^2\big)/t$ であり、$+0.56$ $(t = 0.9)$、0 $(t^* = 2\sqrt{2} - 2 \approx 0.83)$、すなわち $(1-t)\,a^2 = t^2$ の根)、-0.31 $(t = 0.8)$、-14 $(t = 0.5)$ と変化する。0 の近くにある粒子は、ドリフトによって 0 へ向かい続け、t が t^* まで下がってからようやくいずれかのモードへ向けて離れていくため、フローがモードを決める時期は遅い。原点から離れた場所では tanh が飽和し、$u_t \approx (x \mp a)/t$ となって条件付き速度そのものに近づく。
- **損失の下限はノイズの近くで生じる。** ターゲット $\epsilon - x_0$ の二乗平均は $a^2 + 1 = 5$ である。このうち $C = \mathbb{E}_t \mathbb{E}_x \mathrm{Var}(x_0 \mid x_t)/t^2 \approx 2.0$ が損失の下限であり（ここで $\mathrm{Var}(x_0 \mid x_t = x) = a^2\,\mathrm{sech}^2\big((1-t)\,ax/t^2\big)$）、残りの 3.0 が周辺速度の二乗平均である。$C$ のほとんどすべては $t > 0.5$ の領域から生じる。そこでは 2 つのブロブが重なり合っており、位置を見ただけでは粒子がどちらのデータ点へ向かっているのか判別できないからである。$t = 0.4$ を下回ると行き先が判別できるようになり、条件付き速度 $(x - x_0)/t$ が限りなく大きくなるにもかかわらず、そこで生じる下限への寄与は全体の約 1,000 分の 1 にすぎない（演習問題 2）。

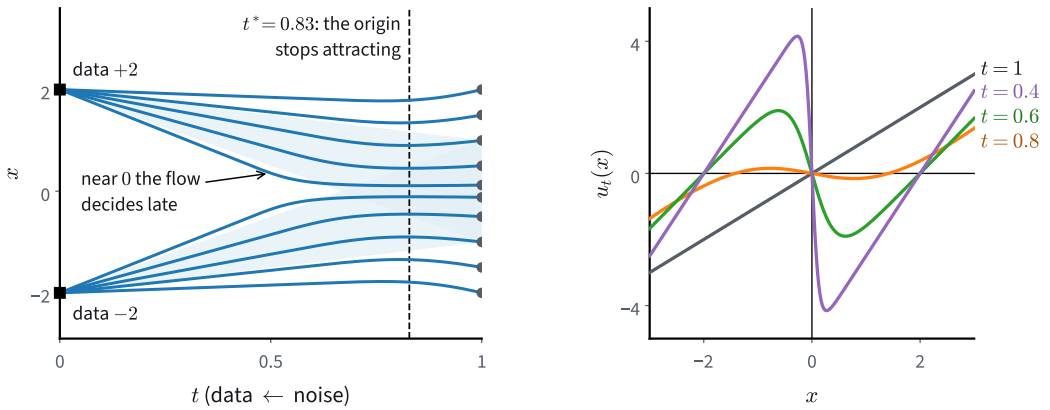


Figure 38.27: ± 2 にある 2 つのデータ点と、独立ペアリングのもとでの標準ガウスノイズ。サンプリングは $t = 1$ のノイズ（右）から $t = 0$ のデータ（左）へと進む。(a) 2 つの条件付きブロブの 1 標準偏差帯の上に重ねた厳密な周辺フローの軌跡。t が $t^* = 0.83$ まで下がる間はすべての軌跡が 0 に向かって動き、0 の近くから出発した軌跡は $t \approx 0.6$ を下回ってようやくモードへ向けて離れる。(b) 速度 $u_t(x)$ は x（$t = 1$ のとき）であり、t が下がるにつれて原点での傾きは負かつ急峻になり、原点から離れると $(x \mp 2)/t$ に漸近する。

次のコードセルでは、2 番目の性質で挙げた 3 つの時刻において原点での傾きを有限差分（finite difference）で求める。続いて 1 番目の性質を検証し、さらにステップ数を増やすとどうなるか

を調べる。ノイズ分布全体を出発点として、厳密な速度場を $t = 1$ から $t = 0$ まで 1、2、4、8 回の等間隔オイラーステップで積分し、データ点から 0.1 以内に着地する質量の割合と終端での $|x|$ の平均を表示する。ノイズ分布はガウス密度で重み付けした細かい開始点グリッドとして表しているため、出力にランダム性は一切ない。

```
a = 2.0                                    # data at -a and +a, noise N(0, 1)
def u_two_point(x, t):                     # closed form of the marginal velocity
    return (x - a * np.tanh((1 - t) * a * x / t**2)) / t

ts = np.array([0.9, 0.8, 0.5])
print('slope at the origin at t = 0.9, 0.8, 0.5:',
      ((u_two_point(1e-6, ts) - u_two_point(-1e-6, ts)) / 2e-6).round(2),
      '   t* =', round(2 * np.sqrt(2) - 2, 4))
x1 = np.linspace(-6, 6, 4001)              # the noise law N(0, 1), as a weighted grid
w = np.exp(-x1**2 / 2) / np.exp(-x1**2 / 2).sum()
for K in (1, 2, 4, 8):                     # K equal Euler steps from t = 1 to t = 0
    x, ts = x1, np.linspace(1, 0, K + 1)
    for t, t_next in zip(ts[:-1], ts[1:]):
        x = x - (t - t_next) * u_two_point(x, t)
    print(f'{K} Euler step(s): mass within 0.1 of +-a {w @ (abs(abs(x) - a) < 0.1):.3f}, '
          f'mean |x| {w @ abs(x):.2f}')
```

```
slope at the origin at t = 0.9, 0.8, 0.5: [  0.56  -0.31 -14.  ]    t* = 0.8284
1 Euler step(s): mass within 0.1 of +-a 0.000, mean |x| 0.00
2 Euler step(s): mass within 0.1 of +-a 0.359, mean |x| 1.48
4 Euler step(s): mass within 0.1 of +-a 0.987, mean |x| 1.99
8 Euler step(s): mass within 0.1 of +-a 0.999, mean |x| 2.00
```

傾きは式どおりの値となる。1 回のオイラーステップのあとではデータ点の近くに到達する質量は皆無であり、$|x|$ の平均は 0 となる。すなわち、すべてのサンプルがデータ平均の上に集まってしまう。こうして冒頭の願いは最初のテストで潰える。個々の条件付き経路はすべて直線であるにもかかわらず、周辺フローは曲率を持ち、1 回の大きなステップで済ませようとすると全サンプルが平均へと潰れてしまうのである。

ステップ数を増やせば、この破綻は急速に修復される。2 ステップで質量の 3 分の 1 強がデータ点に届き、4 ステップで約 1% を除くほぼすべてが、8 ステップで事実上すべてが到達する。これほど少ないステップ数で足りるのは、どのようなグリッドでも最後の 1 ステップが事後平均 $a\tanh(\cdot)$ の上に着地し、それ以前のステップで粒子が原点から十分遠くへ動いてさえいればその値が $\pm a$ になるからである（演習問題 2）。しかし、そもそも節約したかったのはそのステップ数である。フローそのものが真っ直ぐになり、1 ステップで済むようにノイズとデータをペアリングする方法が 38.7.2 節の主題である。

38.7.2 Rectified Flow

38.7.2.1 直線経路とその交差

2 点の例では直線の条件付き経路を用いた。d 次元でも同じ構成により、データサンプル $\mathbf{x}_0$ と独立に引いたノイズサンプル ϵ をペアにし、一定速度で進む直線で両者を結ぶ。

$$\mathbf{x}_t = (1-t)\,\mathbf{x}_0 + t\,\epsilon, \qquad \mathbf{u}_t(\mathbf{x}_t \mid \mathbf{z}) = \dot{\mathbf{x}}_t = \epsilon - \mathbf{x}_0, \qquad \mathbf{z} = (\mathbf{x}_0, \epsilon). \tag{38.99}$$

条件付き速度は t に依存せず、CFM 損失 Equation 38.95 は次の **Rectified Flow** の目的関数とな

る (Liu et al. 2023; Lipman et al. 2023)。

$$\mathcal{L}_{\mathrm{RF}}(\theta) = \mathbb{E}_{t,\, \mathbf{x}_0 \sim p_0,\, \epsilon \sim p_1} \left[\left\| \mathbf{u}_\theta\big((1-t)\mathbf{x}_0 + t\epsilon,\, t\big) - (\epsilon - \mathbf{x}_0) \right\|^2 \right]. \tag{38.100}$$

訓練ではデータとノイズを引き、両者を補間して、その差に対して回帰する。アルゴリズムはこれだけである。この名称はその先を見据えている。**rectified** は「真っ直ぐにされた（整流された）」という意味であり、ペアリングが修正されたあとのフローはまさにそのとおり真っ直ぐになる。しかし先ほど見たように、独立ペアからなるフローは真っ直ぐではない。

技術的な点を 1 つ片付けておく必要がある。ガウスノイズであれば、2 点の例のように $\mathbf{x}_0$ だけで条件付けでき、周辺速度の命題がそのまま適用できる。あとでペアリングを変更できるようにするにはペアで条件付けする必要があるが、ペアを固定すると直線上を滑る点は密度を持たないため、命題はそのままの形では適用できない。それでも結論自体は、期待値 $\mathbb{E}\,\varphi(\mathbf{x}_t)$（ただし φ は滑らかなテスト関数）に対して同じ計算を行うことで成り立つ（演習問題 7）。

なぜフローは曲がるのだろうか。$(\mathbf{x}, t)$ で交差する 2 本の線分は、事後平均 Equation 38.94 に 2 つの異なる向きを与える（Figure 38.28）。関数は各点で 1 つの値しか持てないため、速度場はそれらを平均する。一方、速度場が $\mathbf{x}$ についてリプシッツ連続である常微分方程式は、各点を通る軌跡をただ 1 つしか持たないため（38.1.1.3 節）、学習されたフローの 2 本の軌跡が決して交わることはない。線分が交差する場所では、軌跡は交わる代わりに互いを避けるように曲がるのである。

目指すのは、すべての軌跡が一定速度で進む直線となるような速度場である。そのような場は各軌跡に沿って一定となるため、任意の長さのオイラーステップ 1 回で厳密になる。独立ペアリングはこの理想からほど遠い。損失は正しい速度場によって最小化されており、個々の要素もこれ以上ないほど真っ直ぐであるから、変更できる残された自由度は、どのノイズ点をどのデータ点と結ぶかという点だけである。

38.7.2.2 例：Two Moons

2 点の例には解析解があり、1 ステップでは失敗することがその式からわかった。では解析解を持たないデータに対して、訓練されたネットワークは何ステップを必要とするのだろうか。平面上で 2 つの三日月が噛み合う two-moons データを取り上げ、小さなニューラルネットワークを訓練してステップ数を数えてみよう。数えるにはサンプルの良さを測るスコアが必要であり、ここでは有限の一次モーメント（first moment）を持つ分布 P と Q の間の**二乗エネルギー距離**（squared energy distance）(Székely and Rizzo 2013) を用いる。

$$\mathcal{E}^2(P, Q) = 2\,\mathbb{E}\|X - Y\| - \mathbb{E}\|X - X'\| - \mathbb{E}\|Y - Y'\|,$$

ここで $X, X' \sim P$ と $Y, Y' \sim Q$ は互いに独立である。この量は、P の点と Q の点の間の平均距離を、それぞれの分布内部での平均距離と比較する。Székely と Rizzo はこの式そのものをエネルギー距離と呼んでいるが、距離の公理を満たすのはその平方根であるため、本書では二乗を付けて呼ぶ。これは二乗 MMD（37.4.1 節）の一種であり、$P = Q$ の場合に限り 0 となる。2 組のサンプルから不偏 U 統計量によって推定し、以下で示すスコアはすべてこの二乗距離である。

次のコードセルでは、座標ごとに平均 0・分散 1 に標準化したデータ、2,048 点からなるホールドアウトデータ（held-out data）のサンプル、および以降のすべての比較で再利用する 1 つのノイズ配列 `z_eval` を作成する。続いてホールドアウトサンプルに対する 2 つの基準スコアを計算する。1 つはデータから新たに引いたサンプルであり、もう 1 つは厳密な速度場のオイラーステップ 1 回が生成するデータ平均上の点質量である。

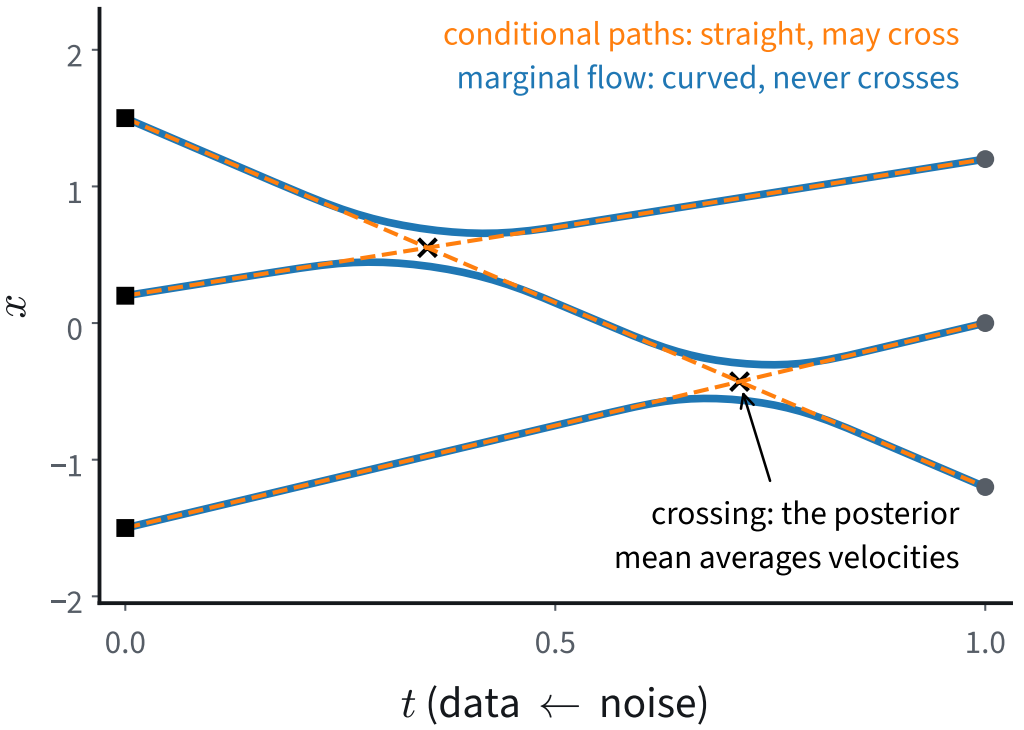


Figure 38.28: 直線分（破線）で結ばれた 1 次元の 3 組のデータ・ノイズペア。うち 2 組が交差している（×）。線分が交差する場所では周辺速度がそれらの向きを平均するため、$t = 1$ のノイズから $t = 0$ のデータへと走らせたフローの軌跡（実線）は曲がり、決して交わらない。各軌跡は元の線分とは異なるデータ点に到達し、出発した順序を保ったまま到着する。

```
def two_moons(n, rng, noise=0.07):
    t = rng.uniform(0.0, np.pi, n)
    m = rng.integers(0, 2, n)                       # which moon
    X = np.stack([np.cos(t) * (1 - 2 * m) + m,
                  np.sin(t) * (1 - 2 * m) + 0.5 * m], axis=1)
    return X + noise * rng.standard_normal((n, 2))

rng = np.random.default_rng()
raw = two_moons(8192, rng)
mu, sd = raw.mean(0), raw.std(0)
moons = (raw - mu) / sd                             # standardized training data
held_out = (two_moons(2048, rng) - mu) / sd
z_eval = rng.standard_normal((2048, 2))             # noise for every comparison

def energy_distance(a, b):                          # squared energy distance, U-statistic
    d = lambda u, v: np.linalg.norm(u[:, None, :] - v[None, :, :], axis=-1)
    n, m = len(a), len(b)
    return (2 * d(a, b).mean() - d(a, a).sum() / (n * (n - 1))
            - d(b, b).sum() / (m * (m - 1)))

print(f'fresh data sample vs held-out: {energy_distance(moons[:2048], held_out):+.4f}')
print(f'point mass at the data mean vs held-out: '
      f'{energy_distance(np.zeros((2048, 2)), held_out):.3f}')
```

```
fresh data sample vs held-out: +0.0018
point mass at the data mean vs held-out: 0.911
```

同じ分布からの 2 つのサンプルのスコアは 0 から 0.001 の数倍以内に収まり、これがこの標本サイズにおけるこの指標の分解能である。点質量のスコアは約 0.9 となる。

モデルは $(\mathbf{x}, t)$ を受け取って速度を返す小さな tanh ネットワークであり、`field` がそれを評価する。あとでペアリングを変更できるよう、関数 `train_rf` はペアのサンプラを引数にとる。単一のバッチでは値がばらつくため最後の 500 バッチにわたり、さらに 2 つの座標にわたって平均した訓練損失をネットワークとともに返す。したがって表示される値は $\mathcal{L}_{\mathrm{RF}}/2$ である。

```
def make_params(key, sizes=(3, 256, 256, 2)):  # Glorot-uniform weights, zero biases
    glorot = jax.nn.initializers.glorot_uniform()
    keys = jax.random.split(key, len(sizes) - 1)
    return [(glorot(k, (i, o)), jnp.zeros(o))
            for k, i, o in zip(keys, sizes[:-1], sizes[1:])]

def mlp(params, x):
    for W, b in params[:-1]:
        x = jnp.tanh(x @ W + b)
    W, b = params[-1]
    return x @ W + b

def field(params, x, t):                # the time enters centered, as 2t - 1
    return mlp(params, jnp.concatenate([x, 2 * t - 1], 1))

def train_rf(sample_pairs, steps=10000, batch=2048, tail=500):
    """Regress v(x_t, t) on eps - x0 (papers: -(x1 - x0) at time 1 - t).
```

```
    Return the parameters and the mean tail loss; sample_pairs(key, n) draws
    n (data, noise) pairs."""
    opt = optax.adam(optax.cosine_decay_schedule(3e-3, steps))   # 3e-3 -> 0
    def loss_fn(params, key):
        k_pairs, k_t = jax.random.split(key)
        x0, eps = sample_pairs(k_pairs, batch)     # data, noise
        t = jax.random.uniform(k_t, (batch, 1))
        xt = (1 - t) * x0 + t * eps                # a point on each segment
        return ((field(params, xt, t) - (eps - x0))**2).mean()
    def step(carry, key):
        params, state = carry
        loss, grads = jax.value_and_grad(loss_fn)(params, key)
        updates, state = opt.update(grads, state)
        return (optax.apply_updates(params, updates), state), loss
    k_init, k_steps = jax.random.split(jax.random.key(int(rng.integers(2**31))))
    params = make_params(k_init)
    (params, _), losses = jax.lax.scan(step, (params, opt.init(params)),
                                       jax.random.split(k_steps, steps))
    return params, float(losses[-tail:].mean())
```

最初の実行ではノイズとデータを独立にペアリングする。CFM 定理により損失は 0 に到達できず、損失が下げ止まる位置は線分がどれだけ交差しているかを測る指標となる。下限 C は、$\epsilon - \mathbf{x}_0$ の $\mathbf{x}_t$ が与えられたもとでの条件付き分散を t にわたって平均したものである。訓練を始める前にこの値を知ることはできるだろうか。有限の訓練セットであれば、ネットワークを使わずに計算できる。$\mathbf{x}_t$ が与えられたとき、$\mathbf{x}_0$ の事後分布は各訓練点 $\mathbf{x}_0'$ に $\exp\left(-\|\mathbf{x}_t - (1-t)\,\mathbf{x}_0'\|^2/2t^2\right)$ に比例する重みを割り当てる。ターゲットは $\epsilon - \mathbf{x}_0 = (\mathbf{x}_t - \mathbf{x}_0)/t$ であるから、その条件付き分散は $\mathbf{x}_0$ の事後分散（posterior variance）を t^2 で割ったものになる。関数 `pairing_floor` は $4{,}000$ 回抽出した $(t, \mathbf{x}_0, \epsilon)$ にわたってこの量を平均し、`train_rf` が座標あたりの損失を報告するのに合わせて値を半分にする。

```
def pairing_floor(data, rng, m=4000, chunk=500):
    """Floor C of independent pairing, per coordinate: E Var(eps - x0 | x_t) / 2."""
    t = rng.uniform(0, 1, (m, 1))
    xt = (1 - t) * data[rng.integers(0, len(data), m)] + t * rng.standard_normal((m, 2))
    var = []
    for i in range(0, m, chunk):              # posterior of x0 over all training points
        ti, xi = t[i:i + chunk], xt[i:i + chunk]
        d2 = ((xi**2).sum(1, keepdims=True) - 2 * (1 - ti) * xi @ data.T
              + (1 - ti)**2 * (data**2).sum(1))          # |x_t - (1 - t) x0'|^2
        logw = -d2 / (2 * ti**2)
        r = np.exp(logw - logw.max(1, keepdims=True))
        r /= r.sum(1, keepdims=True)
        v = r @ (data**2).sum(1) - ((r @ data)**2).sum(1)  # Var(x0 | x_t), summed
        var.append(v / ti[:, 0]**2)           # eps - x0 = (x_t - x0) / t
    return np.concatenate(var).mean() / 2

print(f'floor of the independent pairing, per coordinate: {pairing_floor(moons, rng):.2f}')
```

```
floor of the independent pairing, per coordinate: 1.37
```

下限は座標あたり約 1.37 であり、推定値は抽出ごとに小数第 2 位が変動する。完璧なネットワークであればここで下げ止まる。今回用いるネットワークは、隠れ層 2 層（各 256 ユニット）を持

ち、10,000 ステップ（各ステップ 2,048 ペアのミニバッチ）の訓練によってこの下限に近づく。

```
data = jnp.asarray(moons, dtype=jnp.float32)
def independent_pairs(key, n):                  # random data rows, fresh noise
    k_rows, k_noise = jax.random.split(key)
    return (data[jax.random.randint(k_rows, (n,), 0, len(data))],
            jax.random.normal(k_noise, (n, 2)))

net, plateau = train_rf(independent_pairs)
print(f'CFM loss, mean of the last 500 batches: {plateau:.2f}')
```

```
CFM loss, mean of the last 500 batches: 1.40
```

損失は座標あたり約 1.4 で落ち着き、下限との差は 0.01 の数倍以内に収まる。したがって、損失が 1.4 であることは当てはまりが悪いことを意味しない。そのほとんどすべては交差する線分間の食い違いであり、いかなるネットワークも取り除くことはできない。CFM 定理により、速度場の誤差にあたるのは下限を超える過剰分だけである。

もっとも、その 0.01 の数倍という差は見かけほど小さくはない。ターゲットの座標あたりの二乗平均（二次モーメント）は 2（ノイズと標準化データから 1 ずつ）であり、そのうち 1.37 が下限である。したがって速度場自体の二乗平均は約 0.6 にすぎず、誤差はその数パーセントに相当する。この誤差には、あとで生成サンプルの中で再び出会うことになる。

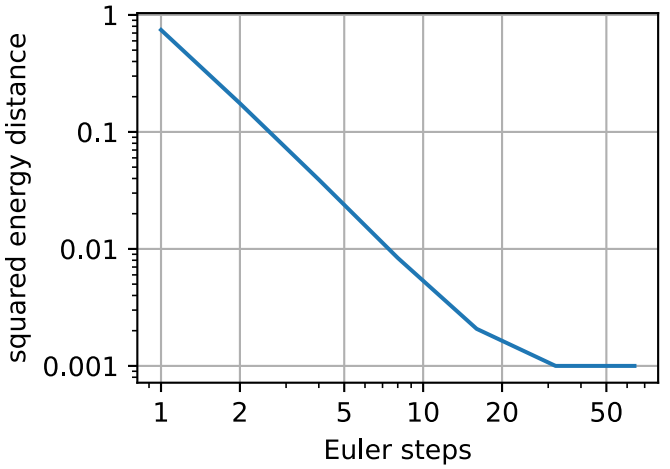
サンプリングでは、$\dot{\mathbf{x}} = \mathbf{u}_\theta(\mathbf{x}, t)$ を $t = 1$ から 0 までオイラー法（38.2 節）で積分する。すなわち K 回のステップ（ステップ幅 $1/K$）を行い、各ステップでは $\mathbf{u}_\theta(\mathbf{x}, t)/K$ を $t = 1, 1-1/K, \dots, 1/K$ において引く。関数 `euler_path` は形状 `(steps + 1, n, 2)` の軌跡全体を返し、その最後の行が生成サンプルとなる。

```
def euler_path(params, z, steps):
    def step(q, t):
        q = q - field(params, q, jnp.full((len(z), 1), t)) / steps
        return q, q
    q1 = jnp.asarray(z, dtype=jnp.float32)              # noise at t = 1
    _, path = jax.lax.scan(step, q1, 1 - jnp.arange(steps) / steps)
    return np.concatenate([np.asarray(q1)[None], np.asarray(path)])
```

厳密な速度場の 1 ステップは、スコアが 0.9 であった点質量を与えるはずである。訓練された速度場について、次のコードセルはノイズ配列 `z_eval` を $K = 1, 2, 4, \dots, 64$ ステップで積分し、ホールドアウトデータとの二乗エネルギー距離を両対数軸にプロットする。推定量は不偏であるため指標の分解能付近では負になり得るが、プロットでは 10^{-3} でクリップしている。

```
steps_list = [1, 2, 4, 8, 16, 32, 64]
eds = [energy_distance(euler_path(net, z_eval, K)[-1], held_out) for K in steps_list]
print('  '.join(f'{K}: {e:.3f}' for K, e in zip(steps_list, eds)))
d2l.plot(steps_list, np.maximum(eds, 1e-3), 'Euler steps',
         'squared energy distance', xscale='log', yscale='log')
```

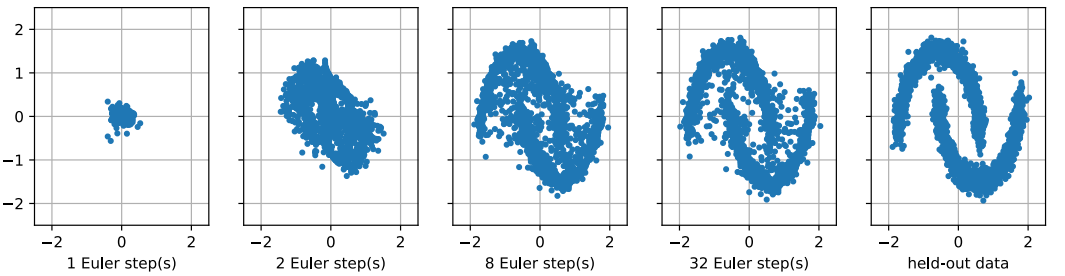
```
1: 0.745  2: 0.175  4: 0.039  8: 0.008  16: 0.002  32: 0.001  64: 0.000
```



1 ステップのスコアは 0.7〜0.75 となり、平均上の点質量（0.9）と同程度である。つまりネットワークは、厳密な速度場が $t = 1$ で引き起こす崩壊を不完全ながら学習している。2 ステップでは 0.16〜0.19、4 ステップでは 0.03〜0.04、8 ステップでは約 0.01 以下となり、16 ステップ以降ではスコアは 0 から 0.001 の数倍以内に留まる。この減少分はオイラー法の離散化誤差である。あとに残るのは訓練された速度場自体の誤差であり、2,048 点でこの指標が分解できる限界まで小さくなっている。ソルバを介さずに適合させた速度場であるから、ステップ数を増やすほど精度が向上するのは当然である。対照的に、38.3.1 節で 10 回のオイラーステップを通して訓練した場は、ステップを細かくするとかえって悪化したのであった。

スコアは 1 つの数値にすぎないため、それが何を要約しているのかを実際に見ておく価値がある。次のコードセルでは、1、2、8、32 ステップの生成サンプルと、その横にホールドアウトデータを描画する。

```
panels = [(f'{K} Euler step(s)', euler_path(net, z_eval, K)[-1]) for K in (1, 2, 8, 32)]
fig, axes = d2l.plt.subplots(1, 5, figsize=(12, 2.6), sharex=True, sharey=True)
for ax, (title, s) in zip(axes, panels + [('held-out data', held_out)]):
    d2l.plot(s[:, 0], s[:, 1], title, fmts=('.',),
             xlim=(-2.5, 2.5), ylim=(-2.5, 2.5), axes=ax)
```



1 ステップでは平均周りのブロブとなり、2 ステップでは斜めの帯になる。8 ステップで雲は 2 つの三日月形へと広がるが、その間にはまだ多くの点が散らばっている。32 ステップになるとそれらの大半が消え、最後のパネルのデータのように 2 つの分離した三日月が描かれる。もっとも、データと並べて見比べると三日月はわずかに太く、隙間にも少数の点が残っている。ステップ数をこれ以上増やしても消えないため、これらの欠陥はソルバではなく速度場自体に属するもので

ある。損失とその下限との間にあった 0.01 の数倍という差は、サンプル上ではこのような姿となって現れる。

スコアはこうした細部の欠陥をほとんど捉えない。点間の平均距離を比較する指標であるため、輪郭がどれほど鋭いかよりも先に質量がどこにあるかを反映し、指標の分解能に達したスコアであってもデータと同等の鋭さまで保証するわけではないからである。ステップ数を増やすことで確実に取り除かれるのはソルバの誤差であり、スコアで見る限りそれにはサンプルあたり 16〜32 回のネットワーク呼び出しを要する。これが、独立ペアリングによってフローに組み込まれた曲率のコストである。

38.7.2.3 Reflow

フローを曲げている原因は独立ペアリングにあり、訓練済みのモデルはより良いペアリングを提供してくれる。新たなノイズからモデルの ODE を走らせ、各ノイズ点 ϵ をその到達端点 $\hat{\mathbf{x}}_0(\epsilon)$ とペアにするのである。単一のフローの軌跡は決して交差しないため、こうして作ったペアは独立ペアよりもはるかに交差しにくい。この新しいペアの上で同じアーキテクチャを再訓練する手続きを **Reflow** と呼ぶ (Liu et al. 2023)。

ペアを組み直すことで何か悪影響は生じないだろうか。端点の分布が崩れたり、以前より遠い点どうしがペアになったりする懸念があるが、実際にはそのどちらも起こらない。ペアリングが点をどれだけ遠くへ動かすかは、**輸送コスト**（transport cost）$\mathbb{E}\,c(\mathbf{x}_1-\mathbf{x}_0)$、すなわち変位の凸関数 c（二乗長さなど）をペア全体で平均した値で測られる。以下の命題では大文字でランダムな経路を表す。$\mathbf{X}_t$ はペアを結ぶ直線分であり、$\mathbf{Z}_t$ は同じノイズ点から出発するフローの軌跡である。

命題（Reflow は周辺分布を保ち輸送コストを下げる）。 **データ点とノイズ点のペア $(\mathbf{X}_0,\mathbf{X}_1)$ が有限の二次モーメントを持つとし、$\mathbf{X}_t=(1-t)\mathbf{X}_0+t\mathbf{X}_1$ とおき、$\mathbf{u}_t$ をこの直線経路の周辺速度とし、$\mathbf{Z}_t$ をノイズ側の端点から時間を逆向きに $\dot{\mathbf{Z}}_t=\mathbf{u}_t(\mathbf{Z}_t)$ を解いた解（ただし $\mathbf{Z}_1=\mathbf{X}_1$）とする。このフローが存在するよう、$\mathbf{X}_t$ は任意の $t>0$ で正の密度を持ち、$\mathbf{u}_t$ は連続微分可能かつ $\mathbf{x}$ について（t に依らない定数で）リプシッツ連続であると仮定する。このとき、(i) $\mathbf{Z}_t$ は $\mathbf{X}_t$ と同じ分布に任意の t で従い、特に $\mathbf{Z}_0\sim p_0$ となる。(ii) 任意の凸関数 c に対して次が成り立つ。**

$$\mathbb{E}\,c(\mathbf{Z}_1-\mathbf{Z}_0)\;\le\;\mathbb{E}\,c(\mathbf{X}_1-\mathbf{X}_0). \tag{38.101}$$

証明。 (i) $\mathbf{X}_t$ の分布は $\mathbf{u}_t$ とともに連続の式を満たす。これは $\mathbf{z}=(\mathbf{X}_0,\mathbf{X}_1)$ として点質量も扱える弱形式の周辺速度命題（演習問題 7）を適用した結果である。$\mathbf{Z}_t$ の分布もまた、$\mathbf{Z}_t$ が速度場 $\mathbf{u}_t$ に従って動くため同じ連続の式を満たす。両者はともに p_1 に $t=1$ で一致し、このような速度場を持つ連続の式の 2 つの解はある時刻で一致すればすべての時刻で一致する。この一意性だけは完全には証明しないステップである。38.5.4.3 節では滑らかな正の密度に対して各軌跡上で両方の対数密度が変化率 $-\nabla\cdot\mathbf{u}_t$ で変化することからこれを示したが、ここでは弱形式での再証明は省いてそのまま用いる。(ii) 変位を時間積分として書き、イェンセンの不等式（Jensen's inequality）を 2 回適用する。

$$\begin{aligned}\mathbb{E}\,c(\mathbf{Z}_1-\mathbf{Z}_0)&=\mathbb{E}\,c\Big(\int_0^1\mathbf{u}_t(\mathbf{Z}_t)\,dt\Big)\le\int_0^1\mathbb{E}\,c(\mathbf{u}_t(\mathbf{Z}_t))\,dt\\&=\int_0^1\mathbb{E}\,c(\mathbb{E}[\mathbf{X}_1-\mathbf{X}_0\mid\mathbf{X}_t])\,dt\le\mathbb{E}\,c(\mathbf{X}_1-\mathbf{X}_0).\end{aligned}$$

1 つ目の不等号は時間平均に対するイェンセンの不等式である。等号では (i) を用いて $\mathbf{Z}_t$ を $\mathbf{X}_t$ に置き換え、続いて周辺速度の定義を用いている。最後の不等号は条件付きイェンセンの不等式である。■

$c = \|\cdot\|^2$ とすると、最後のステップでちょうど条件付き分散の分だけ減少するため、コストは少なくとも元のペアリングの下限の分だけ下がる。すなわち $\mathbb{E}\|\mathbf{Z}_1 - \mathbf{Z}_0\|^2 \le \mathbb{E}\|\mathbf{X}_1 - \mathbf{X}_0\|^2 - C$ である。訓練時には厄介者であった下限 C が、1 ラウンドの Reflow によって保証されるコスト削減量となる。線分がまったく交差しない場合には下限は 0 であり、再ペアリングで得られる改善はない。

この命題の仮定は、ここでの具体例が満たす条件よりも強い。2 点の例や、ガウスノイズと独立にペアリングした任意の有限訓練セットでは、周辺速度場は任意の $[\delta, 1]$（各 $\delta > 0$）上ではリプシッツ連続だが、$t = 0$ まではそうならない。実際、2 点の例の速度場における傾き -14（$t = 0.5$）は、$t \to 0$ で限りなく大きくなる。そのため上の証明がそのままカバーするのは時刻 δ までのフローに限られ、(i) は $t \ge \delta$ に対して、(ii) は $\mathbb{E}\, c\big((\mathbf{Z}_1 - \mathbf{Z}_\delta)/(1-\delta)\big) \le \mathbb{E}\, c(\mathbf{X}_1 - \mathbf{X}_0)$ の形で得られる。極限 $\delta \to 0$ の移行は証明しないため、その意味で以下の例は命題の適用範囲を一歩越えている。

2 点の例では結論を直接確かめられる。1 次元のフローは順序を保つため、1 ラウンドの Reflow は各ノイズ点 ϵ を $a\,\mathrm{sign}(\epsilon)$ とペアにする（演習問題 4）。このペアリングのコストは $a^2 - 2a\sqrt{2/\pi} + 1 = 1.81$ と、独立ペアリングの $a^2 + 1 = 5$ より小さく、しかも線分が決して交差しないため 2 ラウンド目では $C = 0$ となり、その速度場は 1 回のオイラーステップで厳密に積分される。

two-moons データに対しては、次のコードセルで新たに 8,192 個のノイズ点を引き、訓練済みモデルを 32 ステップ積分してその端点を求め、この固定されたペアの上で `train_rf` により新しいネットワークを訓練する。

```
eps_pairs = rng.standard_normal((8192, 2))            # noise endpoints, kept
x0_pairs = euler_path(net, eps_pairs, 32)[-1]         # the model's own endpoints
X0, Eps = jnp.asarray(x0_pairs, jnp.float32), jnp.asarray(eps_pairs, jnp.float32)
def model_pairs(key, n):                              # the same pairs, never re-paired
    i = jax.random.randint(key, (n,), 0, len(X0))
    return X0[i], Eps[i]

net2, plateau2 = train_rf(model_pairs)
print(f'reflow loss, mean of the last 500 batches: {plateau2:.4f}')
```

```
reflow loss, mean of the last 500 batches: 0.0006
```

最初の損失が 1.4 で下げ止まっていたのに対し、Reflow の損失は 1,000 分の 1 未満にまで下がっており、新しい線分はほとんど交差していない。交差のない速度場は軌跡に沿ってほぼ一定となるため、少数のオイラーステップで十分なはずである。次のコードセルでは、同じノイズ配列から両方のモデルを 1、2、32 ステップでサンプリングする。

あわせて、命題が上界を与えるコストである旧ペアと新ペアの平均二乗長さも表示する。旧ペアのコストは 4 である。ノイズと標準化データはそれぞれ 2 つの座標の各々で分散 1 を持ち、独立な抽出では分散がそのまま足し合わされるからである。独立ペアリングの下限を 2 座標分合計すると、先に表示した値の 2 倍、すなわち約 2.75 となる。したがって厳密な周辺フローであれば、命題は新しいペアのコストが高々約 1.25 になることを保証する。

```
for K in (1, 2, 32):
    e1 = energy_distance(euler_path(net, z_eval, K)[-1], held_out)
    e2 = energy_distance(euler_path(net2, z_eval, K)[-1], held_out)
    print(f'{K:2d} Euler step(s): squared energy distance  original {e1:.3f}'
          f'  reflowed {e2:.3f}')
```

```
    x0_indep = moons[rng.integers(0, len(moons), len(eps_pairs))]
    print(f'mean squared pair length: independent {np.mean(np.sum((eps_pairs - x0_indep)**2,
    ↪  1)):.2f}'
          f', reflowed {np.mean(np.sum((eps_pairs - x0_pairs)**2, 1)):.2f}')
```

```
 1 Euler step(s): squared energy distance  original 0.745  reflowed 0.001
 2 Euler step(s): squared energy distance  original 0.175  reflowed 0.001
32 Euler step(s): squared energy distance  original 0.001  reflowed 0.000
mean squared pair length: independent 4.04, reflowed 0.30
```

Reflow 後のモデルのオイラーステップ 1 回は、元のモデルの 32 ステップと同じスコア、すなわち高々 1,000 分の 1〜2 という指標の分解能限界を達成する（元のモデルの 1 ステップは約 0.7 であった）。これ以上良くなりようはない。Reflow 後のネットワークは元のモデルの 32 ステップ端点を再現するよう訓練されているため、最初のモデルのサンプルと一緒にその誤差も受け継ぐからである。

ペアの平均二乗長さは 4 から約 0.3 へと低下し、上界が保証する 1.25 を大きく下回る。この余裕は証明の最初のステップから生じている。最初のフローの曲がった軌跡上では、時間平均に対するイェンセンの不等式が等号からほど遠いのである。

各 32 ステップ軌跡と、その両端を結ぶ弦との平均距離を計算すれば、軌跡の直線性も測ることができる。

```
    def chord_gap(P):                                   # P: (steps + 1, n, 2) trajectory
        chord = P[0] + (P[-1] - P[0]) * np.linspace(0, 1, len(P))[:, None, None]
        return np.linalg.norm(P - chord, axis=-1).mean()

    print(f'mean distance from the chord: original {chord_gap(euler_path(net, z_eval, 32)):.3f}'
          f', reflowed {chord_gap(euler_path(net2, z_eval, 32)):.3f}')
```

```
mean distance from the chord: original 0.246, reflowed 0.004
```

元の軌跡は弦から平均して約 4 分の 1 単位ほど外れる。最初はオイラーステップ 1 回が着地した平均の方向へ向かって出発し、あとになってようやく三日月の方へと向きを変えるためである。一方、Reflow 後の軌跡は弦から 0.005 以内に収まっており、約 50 倍も直線に近い。1 ステップで十分な理由はここにある。Reflow はフローを真っ直ぐにするが、最適にするとは限らない。2 次元以上では、フローが真っ直ぐで交差が一切なくても、必要以上に遠くまで質量を運んでしまうことがあり得るからである。

38.7.3 最適輸送と直線性

38.7.3.1 カップリングとワッサースタイン距離

Reflow は輸送コストを下げた。コストはどこまで下げられるのか、そして最も低コストなペアリングのフローは真っ直ぐなのだろうか。37.4.2 節の用語を改めて確認しておこう。**カップリング** π（p_0 と p_1 のカップリング）とは、これらを周辺分布に持つペアの同時分布であり、各始点から各終点へどれだけの質量を運ぶかを定める輸送計画である。**2 次ワッサースタイン距離**（2-Wasserstein distance）は、二乗距離のもとで最も低コストな計画のコストを表す次の量である。

$$W_2^2(p_0, p_1) = \min_{\pi \in \Pi(p_0, p_1)} \mathbb{E}_{(\mathbf{x}_0, \mathbf{x}_1) \sim \pi} \left[\|\mathbf{x}_1 - \mathbf{x}_0\|^2 \right], \tag{38.102}$$

ここで $\Pi(p_0, p_1)$ はカップリング全体の集合である。すなわち W_2^2 は、一方の分布を他方の分布へ変えるのに必要な最小の平均二乗変位である。

最も低コストな計画はどのような姿をしているのだろうか。2 つの分布の一方が密度を持ち、両方が有限の二次モーメントを持つとき、その計画は一意に定まり、質量を分割しない。すなわち、密度を持つ側の分布からの写像となる。ここでの設定ではその分布はノイズ p_1 であり、ノイズからデータへの写像は凸関数の勾配、すなわち **Brenier 写像**（Brenier map）となる。これが Brenier の定理 (Brenier 1991) であり、本書では証明なしに引用する。データ側は密度を持つ必要はない。2 点の例も密度を持たないし、先ほどの 8,192 点の two-moons のような有限の訓練セットも密度を持たない。直線（1 次元）上では、この定理は視覚的に理解しやすい。1 変数凸関数の導関数は単調非減少であるため、最適写像は点の順序を保つのである。

最小の例は 1 次元のガウス分布であり、データを $\mathcal{N}(m, s^2)$（$s > 0$）、ノイズを $\mathcal{N}(0,1)$ とする。ソート済みペアリングは各分位点（quantile）を同じ分位点へと送り（$x_0 = m + s\epsilon$）、コストは $\mathbb{E}(m + s\epsilon - \epsilon)^2 = m^2 + (s-1)^2$ となる。一方、独立ペアリングのコストは $\mathbb{E}(\epsilon - x_0)^2 = m^2 + 1 + s^2$ である。1 次元では任意の凸コストに対してソート済みペアリングが最適となる。2 組のペアが交差しているとき、互いの行き先を入れ替えてもコストは増えないからである（演習問題 2d、37.4 節）。したがって次式が成り立つ。

$$W_2^2\big(\mathcal{N}(0,1), \mathcal{N}(m, s^2)\big) = m^2 + (s-1)^2. \tag{38.103}$$

この距離は、平均を平行移動させるコストと幅を伸縮させるコストとに分離される。$\mathcal{N}(0,4)$ と $\mathcal{N}(0,1)$ の間ではその値は 1 であるのに対し、独立ペアでは 5 となる（Figure 38.29 のパネル (a) と (b)）。独立ペアリングは、コストの 5 分の 4 を他の質量とすれ違わせる無駄な移動に費やしている。

これにより、Figure 38.28 の軌跡がどこに行き着いたかが説明できる。1 次元ではフローの軌跡は交差できないためフローは順序を保ち、順序を保つペアリングこそが最適なペアリングとなる。したがって直線上では、1 ラウンドの Reflow で最適輸送に到達する（演習問題 4）。

2 次元では、交差がないというだけでは不十分である。標準ガウス分布の雲を原点の周りに単位時間で 4 分の 1 回転させる場、すなわち点 (ξ_1, ξ_2) に速度 $\frac{\pi}{2}(-\xi_2, \xi_1)$ を割り当てる速度場を考えよう。どの 2 本の軌跡も交差せず、雲は出発した場所と同じ分布に行き着くため $W_2^2(p_0, p_1) = 0$ である。それにもかかわらず、すべての粒子が速さ $\frac{\pi}{2}\|\mathbf{x}\|$ で動く。これはいわばメリーゴーランドであり、全員が移動しているのに群衆全体としてはどこへも進まないのである（Figure 38.29 のパネル (c)）。

円弧を直線に伸ばしてもこれは解決しない。各 $\mathbf{x}_0$ をその 4 分の 1 回転 $\mathbf{R}\mathbf{x}_0$ とペアにし、両者を線分で結んでみよう。線分 $\big((1-t)\mathbf{I} + t\mathbf{R}\big)\mathbf{x}_0$ は決して交差しない（この行列が任意の t で正則であるため）うえ、各線分は一定速度でたどられる。したがって $C = 0$ であり Reflow はこのペアリングを変化させないが、それでもコストは $\mathbb{E}\|\mathbf{R}\mathbf{x}_0 - \mathbf{x}_0\|^2 = 4$ かかり、$W_2^2 = 0$ とは一致しない。直線上では運動を無駄にする原因は交差だけだが、平面上ではペアリングが直線かつ非交差であっても最適でないことが起こり得る。このペアリングに何が欠けているかは Brenier の定理が教えてくれる。最適写像は凸関数の勾配であり、4 分の 1 回転はいかなる関数の勾配でもない。勾配の導関数は対称行列（symmetric matrix）になるが、$\mathbf{R}$ は対称ではないからである。

ペアリングが記録するのは各粒子が最終的にどこへ着くかだけである。途中の運動にはどのようにコストを課すべきだろうか。1 個の粒子が $\mathbf{x}_1$ から $\mathbf{x}_0$ まで単位時間で何らかの経路 $\mathbf{x}_t$ に沿って移動するとしよう。時間平均に対するイェンセンの不等式より、次が成り立つ。

$$\int_0^1 \|\dot{\mathbf{x}}_t\|^2 \, dt \;\geq\; \Big\| \int_0^1 \dot{\mathbf{x}}_t \, dt \Big\|^2 = \|\mathbf{x}_1 - \mathbf{x}_0\|^2,$$

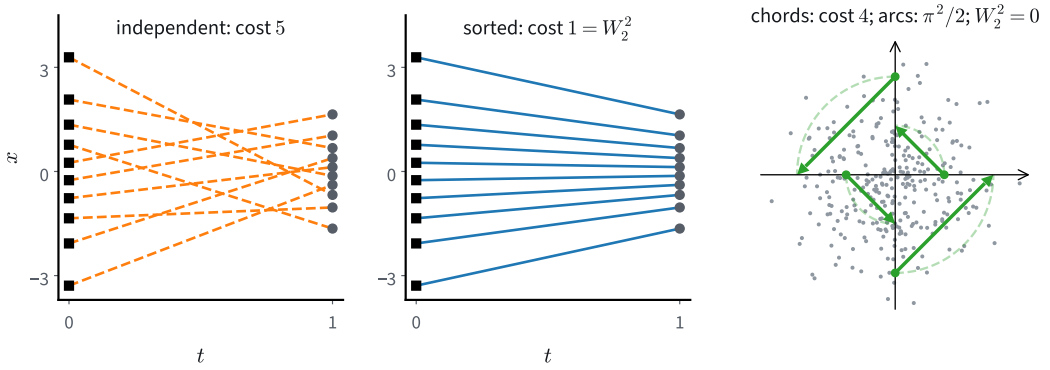


Figure 38.29: ガウス分布を動かす 3 つの方法。(a) データ $\mathcal{N}(0,4)$（$t=0$）とノイズ $\mathcal{N}(0,1)$（$t=1$）の独立ペアリングを、等間隔な 10 個の分位点水準（$0.05, 0.15, \dots, 0.95$）について描いたもの。線分は交差し、平均二乗変位は 5 となる。(b) ソート済みペアリング $x_0 = 2\epsilon$ は決して交差せず、コストは二乗ワッサースタイン距離である 1 となる。(c) 平面内の標準ガウス分布の雲を 4 分の 1 回転させたもの。回転の円弧（破線）の運動エネルギーは $\pi^2/2$ である。各点をその 4 分の 1 回転先と結ぶ弦（矢印）は直線で決して交差せず、コストは 4 である。雲は出発した場所に戻るため $W_2^2 = 0$ であり、2 次元では直線かつ非交差なペアリングであっても最適とは限らない。

等号が成り立つのは速度が一定の場合に限られる。速さの二乗の積分は等速直線運動のときに最小となり、あらゆる迂回や速度変化に対してコストを課す。これを粒子全体にわたって平均したものがフローの**運動エネルギー**（kinetic energy）$\int_0^1 \mathbb{E}\|\mathbf{u}_t(\mathbf{x}_t)\|^2\,dt$ である。先ほどのメリーゴーランドの運動エネルギーは $(\pi/2)^2\,\mathbb{E}\|\mathbf{x}\|^2 = \pi^2/2 \approx 4.9$ であり、そのすべてが無駄に費やされている。

運動エネルギーは、同じ端点を持つフローどうしの順序付けにも使える。データ $\mathcal{N}(0,4)$ とノイズ $\mathcal{N}(0,1)$ の間の 3 つのフロー、すなわちソート済みペアリングの直線フロー、独立ペアリングのもとでの直線分の周辺フロー、および 38.7.1.3 節のコサイン経路を比べよう。どれが最も低コストであり、直線分から組み立てたフローはそうでないフローに勝るのだろうか。3 つともガウス周辺分布 $\mathcal{N}(0, s_t^2)$ を持つため、速度は $(\dot{s}_t/s_t)\,x$、運動エネルギーは $\int_0^1 \dot{s}_t^2\,dt$ となる（演習問題 3）。次のコードセルではまずサンプルをソートすることで、2 つのペアリングコスト 5 と 1 を確かめる。

```
n = 200_000
x0, eps = 2 * rng.standard_normal(n), rng.standard_normal(n)
print(f'data N(0,4), noise N(0,1): independent pairing {np.mean((eps - x0)**2):.2f}, '
      f'sorted pairing {np.mean((np.sort(eps) - np.sort(x0))**2):.3f}')

def kinetic_energy(s, t):                    # marginals N(0, s_t^2): u_t(x) = (s'/s) x
    return np.sum(np.diff(s)**2 / np.diff(t))  # the integral of (ds/dt)^2

t = np.linspace(0, 1, 100_001)
for name, s in [('sorted, straight', 2 - t),
                ('independent, straight', np.sqrt(4 * (1 - t)**2 + t**2)),
                ('cosine path', np.sqrt(1 + 3 * np.cos(np.pi * t / 2)**2))]:
    print(f'{name:22s} kinetic energy {kinetic_energy(s, t):.3f}')
```

```
data N(0,4), noise N(0,1): independent pairing 5.01, sorted pairing 1.001
```

```
sorted, straight       kinetic energy 1.000
independent, straight  kinetic energy 1.858
cosine path            kinetic energy 1.234
```

ソート済みの直線フローが最も低コストであり、その運動エネルギーは $W_2^2 = 1$ となる。驚くべきは残り 2 つの順序である。独立ペアリングの直線分から生じる周辺フローのコストは 1.86 であり、コサイン経路の 1.23 よりも大きい。その分散 $s_t^2 = 4(1-t)^2 + t^2$ は、t が 1 から 0.8 へ下がる間にいったん縮小してから増大に転じるため、粒子は一度内側へ動いてから再び外側へ押し出される。誤ったペアリングのもとで条件付き線分を直線にしても、直線フローにはならないのである。

38.7.3.2 Benamou–Brenier の定理

3 つのフローのうち最も低コストなもののコストは、ちょうど最も低コストなペアリングのコスト W_2^2 に一致した。これはガウス分布に特有の偶然ではない。2 つの分布間の最も低コストなフローは、常に最も低コストなペアリングと同じコストを持ち、しかもそのペアリングの各ペアを一定速度で線分に沿って運ぶフローとなる。これが本節における最も深い事実である。端点をペアにする静的な問題 Equation 38.102 と、フローマッチングが訓練する対象であるフローとが、これによって結びつく。

定理（Benamou–Brenier）。p_0 と p_1 が有限の二次モーメントを持ち、その一方（ここではノイズ p_1）が密度を持つとする。組 $(p_t, \mathbf{u}_t)$ について、$\mathbf{u}_t$ が連続微分可能かつ $\mathbf{x}$ に関して（t に依らない定数で）リプシッツ連続であり、この組が端点 p_0 と p_1 を持つ連続の式を満たすとき、この組を許容的（admissible）と呼ぶ。(i) すべての許容的な組は少なくとも $W_2^2(p_0, p_1)$ の運動エネルギーを持つ。(ii) 最適カップリングのペアを直線分で結ぶ。この経路の周辺速度が許容的であれば、その運動エネルギーは $W_2^2(p_0, p_1)$ に等しく、したがって許容的な組全体にわたる最小値として次式が成り立つ (Benamou and Brenier 2000)。

$$W_2^2(p_0, p_1) = \min_{(p_t, \mathbf{u}_t)} \int_0^1 \int \|\mathbf{u}_t(\mathbf{x})\|^2 \, p_t(\mathbf{x}) \, d\mathbf{x} \, dt. \tag{38.104}$$

(ii) の条件は、たとえば Brenier 写像が滑らかな微分同相写像であり、その導関数と逆写像の導関数がともに有界であるときに満たされる。この条件がない場合でも、二乗可積分な速度を持つ連続の式の弱解全体にわたって最小値をとれば同じ等式が成り立つ。この一般形については証明を省くが、ここでの例にはそれが必要となる。2 点の例における最適速度場は傾き $1/t$ を持ち、上の意味では許容的ではないからである。

証明。最小運動エネルギーを W_2^2 によって下と上から評価する。ステップ II が下界 (i) であり、ステップ III が (ii) を与える。

I. すべてのフローはカップリングを定める。 $\mathbf{Z}_t$ を、$\dot{\mathbf{Z}}_t = \mathbf{u}_t(\mathbf{Z}_t)$ を時間を逆向きに $\mathbf{Z}_1 \sim p_1$ から解いた解とする。Reflow の命題で用いた一意性により、$\mathbf{Z}_t \sim p_t$ が任意の t に対して成り立つため、$(\mathbf{Z}_0, \mathbf{Z}_1)$ は p_0 と p_1 のカップリングである。

II. 下界。このカップリングのコストは少なくとも W_2^2 であり、時間平均に対するイェンセンの不等式によって運動エネルギーで上から抑えられる。

$$\begin{aligned} W_2^2(p_0, p_1) &\le \mathbb{E}\,\|\mathbf{Z}_1 - \mathbf{Z}_0\|^2 = \mathbb{E}\left\| \int_0^1 \mathbf{u}_t(\mathbf{Z}_t)\, dt \right\|^2 \\ &\le \int_0^1 \mathbb{E}\,\|\mathbf{u}_t(\mathbf{Z}_t)\|^2 \, dt = \int_0^1 \int \|\mathbf{u}_t\|^2 \, p_t \, d\mathbf{x}\, dt. \end{aligned}$$

III. **上界**。最適カップリング $\pi^\star$ とその線形補間 $\mathbf{X}_t = (1-t)\mathbf{X}_0 + t\mathbf{X}_1$ をとり、周辺速度を $\mathbf{u}_t(\mathbf{x}) = \mathbb{E}[\mathbf{X}_1 - \mathbf{X}_0 \mid \mathbf{X}_t = \mathbf{x}]$ とする。弱形式の周辺速度命題により、この組は正しい端点を持つ連続の式を満たし、条件付きイェンセンの不等式によりその運動エネルギーは次のようになる。

$$\int_0^1 \mathbb{E}\big\|\mathbb{E}[\mathbf{X}_1 - \mathbf{X}_0 \mid \mathbf{X}_t]\big\|^2 \, dt \le \mathbb{E}\|\mathbf{X}_1 - \mathbf{X}_0\|^2 = W_2^2(p_0, p_1).$$

上下の評価が一致した。■

この定理はモデルについて何を語っているだろうか。最適なペアリングの上で Rectified Flow を訓練すれば、冒頭の願いが叶う。すなわち損失の下限は 0 となり、厳密な速度場は各軌跡に沿って一定となり、長さ 1 のオイラーステップ 1 回で誤差なく積分できる。これら 2 つの性質はいずれも、Reflow の命題にも登場した証明中の 2 つの不等式における等号成立条件にあたる。最適カップリングのフローが許容的であれば下界を達成するため、III の不等式は等号となり、条件付き分散は 0 になる。つまり最適カップリングの線分は交差しない。また II のイェンセンの不等式が等号となるのは各粒子が一定速度で動く場合に限られるため、あらゆる最小点は粒子を一定速度で直線上を動かす。その逆は成り立たない。4 分の 1 回転の例が示したとおりである。フローが真っ直ぐで交差がなく、1 ステップで厳密であっても最も低コストであるとは限らず、W_2^2 を達成するのはそのペアリング自体も最適である場合に限られる（演習問題 7）。

結局のところ目標は最適カップリングであり、実務ではそれに対する 2 つの近似手法が用いられる。1 つはモデルを使って再ペアリングする Reflow である。もう 1 つの**ミニバッチ最適輸送**（minibatch optimal-transport）ペアリング (Tong et al. 2024; Pooladian et al. 2023) は訓練済みモデルを必要とせず、各訓練ステップの前にバッチ内のノイズ点とデータ点を二乗距離の総和が最小となる割り当てによって組み直す。演習問題 5 ではこのペアリングで two-moons を訓練し、損失のプラトー、1 ステップのスコア、およびペアの長さへの効果を測定する。いずれの手法も 2 つの分布間の真の最適カップリングを見つけるわけではない。バッチごとの計画はバッチの選び方に依存する。また Reflow は、4 分の 1 回転の例が示したように最適であるか否かにかかわらず交差のない任意のペアリングで停止するうえ、最初のモデルの誤差も受け継ぐ。

38.7.4 学習された速度場によるサンプリング

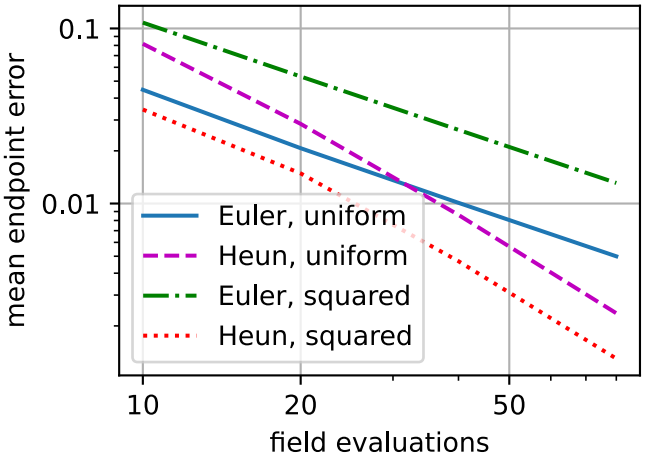
フローを真っ直ぐにすることは、ネットワーク呼び出しを節約する 1 つの方法である。もう 1 つの方法は、高次のソルバを用いたり経路が大きく動く場所にステップを配置したりして、呼び出しをより有効に使うことである。この工夫は学習された速度だけでなく学習されたスコアにも使える。38.6.3 節の確率フロー ODE はスコアを速度場へと変換し、とりわけ分散保持経路上ではフローマッチングが学習するのとまさに同じ場へと変換するからである。サンプルの生成にはソルバの 1 段あたり 1 回の速度場評価というコストがかかり、問題は限られた評価予算をどう配分するかである。ここでは学習誤差のない厳密な速度場の上で 1 つ実験を行い、現れる誤差がすべてソルバに由来するようにして調べる。

ホイン法（Heun's method）は 2 次精度だが、1 ステップあたり速度場を 2 回評価する。2 回分の評価を費やす価値はあるのだろうか。本章で繰り返し用いている混合分布（± 2 に分散 0.25 の成分を持つ分布）の**厳密な**スコアを用いた確率フロー ODE を、$\beta(t)$ が 0.1 から 20 まで線形に増加する分散保持スケジュールのもとで考える。次のコードセルでは、オイラー法とホイン法（38.2 節）を用いて $t=1$ から $t=0$ まで積分し、8,000 個の出発点の到達端点を 800 ステップのホイン法の解と比較する。両手法には 10〜80 回の速度場評価（出力される表の NFE の行、すなわち関数評価回数「number of function evaluations」）という同一の予算を与え、2 種類の時間グリッド上で比較する。1 つは t に関する等間隔ステップであり、もう 1 つは二**乗**グリッド $t_k = (1 - k/K)^2$（$k = 0, \dots, K$、データ側の端点 $t=0$ に向かってステップ幅が縮むグリッ

ド）である。ホイン法の誤差はステップ幅の二乗に比例して減り、オイラー法の誤差はステップ幅に比例して減るため、評価回数が十分多ければ必ずホイン法が勝つ。では 10 回で十分だろうか。

```
bmin, bmax = 0.1, 20.0
beta_fn = lambda t: bmin + t * (bmax - bmin)            # VP noise schedule
abar_fn = lambda t: np.exp(-(bmin * t + 0.5 * (bmax - bmin) * t**2))
def score_vp(q, t, means=(-2.0, 2.0), var=0.25):        # exact mixture score
    a = abar_fn(t)
    v = a * var + (1 - a)
    w = np.stack([np.exp(-(q - np.sqrt(a) * m)**2 / (2 * v)) for m in means])
    return (w * np.stack([(np.sqrt(a) * m - q) / v
                          for m in means])).sum(0) / w.sum(0)
def pf_ode(q, t):                                       # probability-flow ODE
    return -0.5 * beta_fn(t) * (q + score_vp(q, t))
def solve(q, ts, heun=False):                           # steps along grid ts
    for t1, t2 in zip(ts[:-1], ts[1:]):
        h, d1 = t2 - t1, pf_ode(q, t1)
        q = q + 0.5 * h * (d1 + pf_ode(q + h * d1, t2)) if heun else q + h * d1
    return q
grids = {'uniform': lambda K: np.linspace(1, 0, K + 1),     # equal steps in t
         'squared': lambda K: np.linspace(1, 0, K + 1)**2}  # dense near t = 0
z = rng.standard_normal(8000)
ref = solve(z, grids['uniform'](800), heun=True)        # fine reference
nfes = [10, 20, 40, 80]                                 # Heun: n // 2 steps
errs = {f'{m}, {g}': [np.abs(solve(z, grid(n // k), k == 2) - ref).mean()
                      for n in nfes]
        for g, grid in grids.items() for m, k in (('Euler', 1), ('Heun', 2))}
print(f'{"NFE":16s}' + ''.join(f'{n:8d}' for n in nfes))
for name, e in errs.items():
    print(f'{name:16s}' + ''.join(f'{v:8.4f}' for v in e))
d2l.plot(nfes, list(errs.values()), 'field evaluations', 'mean endpoint error',
         legend=list(errs), xscale='log', yscale='log')
```

```
NFE                 10      20      40      80
Euler, uniform    0.0446  0.0207  0.0101  0.0050
Heun, uniform     0.0818  0.0285  0.0086  0.0024
Euler, squared    0.1080  0.0532  0.0263  0.0131
Heun, squared     0.0344  0.0149  0.0047  0.0013
```



一様グリッド上では、各手法の次数が減少率にそのまま現れる。予算を 2 倍にするごとに、オイラー法の誤差は約 2 分の 1 になり、ホイン法の誤差は約 3 分の 1 から 4 分の 1 へと近づく倍率で縮む。ところが同じコストで比べると、ホイン法は評価回数 10（約 0.08 に対しオイラー法は 0.04）および 20（約 0.03 に対し 0.02）で敗れ、40 以上でようやく優位に立つ。一方、二乗グリッド上ではすべての予算でホイン法が勝ち、評価回数 10 での誤差（約 0.03）はいずれのグリッド上のオイラー法よりも小さい。オイラー法は逆の動きを示し、二乗グリッドにすると誤差が 2 倍以上に悪化する。この速度場では、予算が小さいときどちらのソルバが勝つかを決めるのはグリッドなのである。

なぜ 1 次法が 2 次法に勝てるのだろうか。終端誤差は符号付きのステップ誤差の和であり、符号付きの誤差どうしは打ち消し合うことがある。次のコードセルではその度合いを測定する。厳密な経路上の点から出発した k 番目の粗いステップの誤差をとり、その時刻での変位に対する終端の感度（厳密なフローの近傍軌跡から読み取った値）を掛ける。その積 c_k が、終端誤差に対するそのステップの 1 次近似での寄与である。コードセルは 2 つの表を表示する。和をとったあとに残る割合 $\left|\sum_k c_k\right| / \sum_k |c_k|$ と、打ち消し合いが起こる前の総量 $\sum_k |c_k|$ を、それぞれ出発点にわたって平均したものである。最後の行には、各経路の長さのうち $t = 1$ から $t = 0.5$ の間に含まれる割合を表示する。

```
def exact_path(q, ts):                  # fine Heun path, and the gain dx(0)/dx(t)
    qs = [np.stack([q, q + 1e-6])]      # the path and a close neighbor
    for t1, t2 in zip(ts[:-1], ts[1:]):
        qs.append(solve(qs[-1], [t1, t2], heun=True))
    qs = np.array(qs)
    return qs[:, 0], (qs[-1, 1] - qs[-1, 0]) / (qs[:, 1] - qs[:, 0])

def step_errors(ts, qs, gain, steps, heun):   # c_k, shape (steps, starting points)
    j = np.arange(0, len(ts), (len(ts) - 1) // steps)    # coarse nodes
    return np.array([gain[b] * (solve(qs[a], ts[[a, b]], heun) - qs[b])
                     for a, b in zip(j[:-1], j[1:])])   # step error, carried to 0

paths = {g: exact_path(z, grid(400)) for g, grid in grids.items()}   # 400 fine steps
cs = {f'{m}, {g}': [step_errors(grids[g](400), qs, gain, n // k, k == 2) for n in nfes]
      for g, (qs, gain) in paths.items() for m, k in (('Euler', 1), ('Heun', 2))}
```

```
    stats = {'surviving share': (lambda c: np.abs(c.sum(0)).mean() / np.abs(c).sum(0).mean(),
    ↪  '.2f'),
             'total of |c_k|': (lambda c: np.abs(c).sum(0).mean(), '.4f')}   # sum_k |c_k|
    for title, (stat, fmt) in stats.items():
        print(f'{title:16s}' + ''.join(f'{n:8d}' for n in nfes))
        for name, c in cs.items():
            print(f'{name:16s}' + ''.join(f'{stat(c_n):8{fmt}}' for c_n in c))
    d = np.abs(np.diff(paths['uniform'][0], axis=0))    # |fine step| along each path
    share = (d[:200].sum(0) / d.sum(0)).mean()           # first 200 steps: t from 1 to 0.5
    print(f'share of the path length covered by t = 0.5: {share:.2f}')
```

```
surviving share       10      20      40      80
Euler, uniform      0.12    0.12    0.12    0.12
Heun, uniform       0.59    0.66    0.71    0.73
Euler, squared      0.30    0.32    0.34    0.35
Heun, squared       0.26    0.36    0.41    0.43
total of |c_k|        10      20      40      80
Euler, uniform    0.2856  0.1497  0.0767  0.0388
Heun, uniform     0.1324  0.0415  0.0120  0.0032
Euler, squared    0.2466  0.1324  0.0689  0.0352
Heun, squared     0.1257  0.0397  0.0113  0.0030
share of the path length covered by t = 0.5: 0.09
```

$t = 0.5$ までに経路が進む距離は全長の 10 分の 1 未満である。その時間帯の大半ではノイズを加えた分布が $\mathcal{N}(0, 1)$ に近く、そのスコアがドリフトを打ち消すからである。t に関する等間隔ステップはその 10 分の 1 の移動に予算の半分を費やしてしまうため、そもそもステップを偏らせたグリッドを試す動機がそこにある。

残存割合を見ればオイラー法が優位に立った理由がわかる。等間隔ステップでは、どの予算でもオイラー法のステップ誤差のうち終端まで残るのは約 8 分の 1 にすぎないのに対し、ホイン法では半分以上が残る。打ち消し合う前のステップ誤差自体はオイラー法の方が大きく、2 つ目の表の総量はどの予算でもホイン法の 2 倍を超えている。もしホイン法と同じ割合だけ誤差が残っていたなら、オイラー法はすべてで敗れていたはずである。オイラー法の優位は誤差の相殺によるものであり、それは手法自体の性質ではなく、このグリッド上でのこの速度場に固有の性質にすぎない。二乗グリッドはこの相殺を奪い去り、オイラー法のステップ誤差は約 3 分の 1 が残るようになる一方、ホイン法では 4 分の 1〜5 分の 2 となる。

驚くべきことに、二乗グリッドが変えたのはほぼこの点だけである。絶対誤差の総量はグリッドの違いをほとんど受けない。ホイン法の評価回数 10 における総量は等間隔ステップで 0.13、二乗グリッドで 0.125 であり、どの予算でも両者の差は数パーセント以内に収まる。オイラー法の総量に至っては二乗グリッド上の方がむしろ小さく、評価回数 10 で約 7 分の 1、80 で約 10 分の 1 減っているにもかかわらず、終端誤差は 2 倍以上に増えている。つまりステップをデータ側に密集させても、ステップ誤差の総量はほんのわずかしか減らなかったのである。変わったのはどれだけ誤差が打ち消し合うかであり、それがホイン法には有利に、オイラー法には不利に働いた。二乗グリッドは見かけよりも粗く、時刻 t におけるステップ幅は等間隔ステップの約 $2\sqrt{t}$ 倍であるため、$t = 1/4$ 未満では短くなるがそれより上では長くなる。

画像モデルのサンプラも同様にデータ側へステップを密集させており、その度合いは実測によって調整される (Karras et al. 2022)（詳しくは 17.5 節で述べる）。なぜ実測せざるを得ないのかは上の表が物語っている。ある 1 つのグリッド上でのソルバの比較は、そのグリッドについての主張でしかないのである。

2 つの問いが未解決のまま残っている。高次元において最適なペアリングを低コストに見つける

方法は知られておらず、Reflow やバッチ内の再ペアリングはそれに近づくことしかできない。また、本章の冒頭で掲げた「モデルはどこまで少ないステップ数で済ませられるか」という問いも、依然として新しい速度場ごとに試行錯誤で答えが出されているのが実情であり、ステップ数とその配置は導出されるのではなく調整される。第 17 章では、画像を対象にこれら両方の問いを取り上げる。

38.7.5 まとめ

ノイズからデータへの経路は自由に選ぶことができる。その経路を実現する速度はサンプルごとの経路の速度の事後平均であるため、条件付きフローマッチングは最小二乗法によってそれを訓練し、経路どうしの食い違いの大きさを測る損失の下限 C を伴う。本節と前節のすべてのモデルはこの方式で訓練される。すなわち、閉じた形で既知であるサンプルごとの量の条件付き平均に適合させ、ソルバを用いてサンプリングする。ガウス経路上ではスコア、ノイズ、速度がすべて同一の事後平均のアフィン関数となるため、1 つのネットワークをそのいずれとしても解釈できる。条件付き経路を直線にしても線分が交差する場所ではフローが曲がるが、Reflow はノイズをフロー自身の到達端点と組み直すことで輸送コストを増やさずに交差を減らし、two-moons ではオイラーステップ 1 回で 32 ステップに匹敵する結果をもたらす。輸送コストは両端を結ぶあらゆるフローの最小運動エネルギーである二乗ワッサースタイン距離を下回ることはできず、等速直線運動がこの最小値を達成するのは最適なペアリングのもとでのみである。速度場が必要とするステップ数はステップの配置場所にも依存する。グリッドがステップ誤差の相殺量を左右するからである。本章の混合分布の厳密な速度場では、ホイン法は一様グリッド上では評価回数約 40 未満でオイラー法に敗れるが、データ側にステップを密集させたグリッド上ではすべての予算で勝利する。

38.7.6 演習問題

1. **確率的補間。**●Albergo et al. (2023) は、本節と前節の経路を $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1 + c_t\mathbf{w}$ と表している。ここで $(\mathbf{x}_0, \mathbf{x}_1)$ は p_0 と p_1 のカップリングから引かれ、$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ はその両方と独立であり、滑らかなスケジュールは $a_0 = b_1 = 1$ および $a_1 = b_0 = c_0 = c_1 = 0$ を満たす。ガウス分布の p_1 に対しては、拡散経路や Rectified Flow の経路は片側の場合 $c \equiv 0$ にあたり、潜在項 $c_t\mathbf{w}$ は任意の p_0 と p_1 を結ぶ両側経路の役割を果たす。
   1. $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{w})$ を周辺速度 Equation 38.94 にとれ。それが $\mathbb{E}[\dot a_t\mathbf{x}_0 + \dot b_t\mathbf{x}_1 + \dot c_t\mathbf{w} \mid \mathbf{x}_t = \mathbf{x}]$ となることを示し、対応する CFM 損失 Equation 38.95 を書き下せ。
   2. Rectified Flow Equation 38.99 と、データ係数およびノイズ係数が $\cos\frac{\pi t}{2}$ と $\sin\frac{\pi t}{2}$ であるコサイン経路（いずれも独立な端点かつ $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ とする）を復元するスケジュールを挙げよ。
   3. $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$、独立な端点、および $c_t > 0$ のもとで、この補間式が $c \equiv 0$ かつ $\sqrt{b_t^2 + c_t^2}$ で b_t を置き換えた補間式と同じ周辺速度を持つことを示せ。
   4. $c_t > 0$ のとき、$\nabla \log p_t(\mathbf{x}) = -\mathbb{E}[\mathbf{w} \mid \mathbf{x}_t = \mathbf{x}]/c_t$ を示せ。p_1 がガウス分布ではなく任意の分布であるとき、途中の時刻で加わるノイズは何をもたらすか。
2. **遅い意思決定。**データ $\pm a$ を持つ 2 点の例を考える。
   1. 点 $+a$ に重み w、点 $-a$ に重み $1-w$ を与える。Equation 38.98 の導出を繰り返し、事後平均が $a\tanh\big((1-t)\,ax/t^2 + \frac{1}{2}\log\frac{w}{1-w}\big)$ となることを示せ。このとき $t = 1$ から $t = 0$ への 1 回のオイラーステップはどこに着地するか。
   2. 重みが等しいとき、$\partial_x u_t(0) = \big(1 - (1-t)\,a^2/t^2\big)/t$ となること、および t が減少するにつれて原点が $t^* = \frac{1}{2}\big(\sqrt{a^4 + 4a^2} - a^2\big)$（値は $2\sqrt{2} - 2 \approx 0.83$、ただし $a = 2$ のとき）で吸引から反発へと切り替わることを示せ。t^* は $a \to \infty$ および $a \to 0$ のときどのように振る舞うか。両方の極限を解釈せよ。
   3. 原点から離れた場所では傾き $\partial_x u_t$ が $1/t$ に漸近すること、および $t = h$ から $t = 0$

へのオイラーステップが厳密に $\pm a$ へ（tanh が飽和している任意の点から出発したとき）着地することを示せ。それでは、固定ステップ幅ソルバの律速となるのはフローのどの部分であり、どの時間帯か。

4. 損失の下限は $C = \mathbb{E}_t \mathbb{E}_x \mathrm{Var}(x_0 \mid x_t)/t^2$ であり、ここで $\mathrm{Var}(x_0 \mid x_t = x) = a^2 \operatorname{sech}^2\left((1-t)\,ax/t^2\right)$ である。$C \le a^2 + 1$ を示せ。さらに、$t = 0$ 付近の時間帯が C にほとんど寄与しない理由を（そこでは条件付き速度 $(x - x_0)/t$ が非有界であるにもかかわらず）説明せよ。ただし x_t の 0 付近における密度のオーダーが $e^{-(1-t)^2 a^2/2t^2}/t$ である事実を用いよ。

3. **ガウス経路の運動エネルギー**。
   1. 1 次元において $p_t = \mathcal{N}(0, s_t^2)$ であるとき、速度 $u_t(x) = (\dot{s}_t/s_t)\,x$ が連続の式を満たし、その運動エネルギーが $\int_0^1 \dot{s}_t^2\,dt$ となることを示せ。ガウス分布の雲に他の速度が不要なのはなぜか。
   2. データ $\mathcal{N}(0, 4)$ とノイズ $\mathcal{N}(0, 1)$ に対し、本文のコードセルで扱った 3 つのフロー（ソート済みペアリングの直線フロー、独立ペアリングのもとでの直線分の周辺フロー、およびコサイン経路）について s_t を書き下せ。最初の 2 つの運動エネルギーが厳密に 1 と $5 - \pi$ になることを示せ。
   3. データ $\mathcal{N}(0, 1)$ に対して独立ペアリングを同様に計算せよ。その周辺フローが運動エネルギー $2 - \pi/2 \approx 0.43$ を持つ（$W_2^2 = 0$ であるにもかかわらず）ことを示せ。コストが 0 になるのはどのペアリングであり、そのフローはどのようなものか。

4. [code] **2 点に対する 1 ラウンドの Reflow**。データ $\pm a$、ノイズ $\mathcal{N}(0, 1)$、独立ペアリング、および 2 点の例の周辺フロー Equation 38.98 を考える。
   1. 38.3.1.2 節の順序保存の命題と対称性を用いて、$t = 1$ から $t = 0$ へのフロー写像が $\epsilon \mapsto a\,\mathrm{sign}(\epsilon)$ であり、したがって 1 ラウンドの Reflow がソート済みペアリングを生成することを示せ。
   2. そのコストが $a^2 - 2a\sqrt{2/\pi} + 1$（約 1.81、ただし $a = 2$ のとき。これに対し独立ペアリングでは $a^2 + 1 = 5$）であることを示し、なぜこのコストが W_2^2 に等しいのかを説明せよ。
   3. Reflow 後の線分 $(1-t)\,a\,\mathrm{sign}(\epsilon) + t\,\epsilon$ が決して交差せず、したがって 2 ラウンド目の下限が $C = 0$ となることを示し、その速度場を書き下せ。その速度場に対する 1 回のオイラーステップは厳密か。
   4. 閉じた形の速度場を、20,000 個のガウスサンプルから $t = 1$ より $t = 10^{-3}$ まで RK4 で積分し（速度場の傾きが $1/t$ のオーダーで増大するため 0 の手前で止める）、到達端点とコストを確認せよ。
   5. 独立ペアリングの下限は $C \approx 2.03$（$a = 2$ のとき）である。演習問題 2 の (d) の式から数値積分によってこれを計算し、その 99% が集まる境目となる時刻を求めよ。Reflow の命題の上界 $\mathbb{E}\|\mathbf{Z}_1 - \mathbf{Z}_0\|^2 \le \mathbb{E}\|\mathbf{X}_1 - \mathbf{X}_0\|^2 - C$ を確認せよ。ここではどれだけの余裕（スラック）が残っているか。

5. [code] **ミニバッチ最適輸送ペアリング**。`independent_pairs` を、256 個のノイズ点と 256 個のデータ点からなるブロック単位でバッチを引き、各ブロック内を二乗距離の総和が最小となる割り当て（`scipy.optimize.linear_sum_assignment` を 256×256 のコスト行列に適用）によって組み直すサンプラに置き換え、`train_rf` で訓練せよ。
   1. 損失のプラトー、1、2、8、32 ステップにおける二乗エネルギー距離、および再ペアリング前後のペアの平均二乗長さを報告せよ。独立ペアリングおよび Reflow と比較せよ。
   2. 再ペアリングされたノイズ点が依然として標準ガウス分布からのサンプルであり、再ペアリングされたデータ点も依然としてデータからのサンプルであるため、周辺速度場が引き続き p_1 を p_0 へ運ぶことを示せ。
   3. バッチ内のペアリングが 2 つの分布の最適カップリングにはならない理由と、ブロックサイズを大きくするとどうなるかを説明せよ。2 つの再ペアリング手法のうち、訓

練済みモデルを必要とするのはどちらであり、その誤差を受け継ぐのはどちらか。

6. **予測ターゲットの条件性。**ガウス経路 Equation 38.96 において、1 次元データ $x_0 \sim \mathcal{N}(0,1)$ とコサイン経路をとる。
   1. t の関数として、Table 38.2 に挙げたサンプリングされた各訓練ターゲットの分散を計算せよ。対象は条件付きスコア $-\epsilon/\sigma_t$、ノイズ ϵ、クリーンな点 x_0、速度 $\dot\alpha_t x_0 + \dot\sigma_t \epsilon$、および $\mathbf{v}$ ターゲット $\alpha_t\epsilon - \sigma_t x_0$ である。発散するのはどれであり、経路のどちらの端で発散するか。
   2. ノイズに対して訓練されたネットワークが、誤差を伴う出力 $\hat\epsilon$（誤差 δ）を返すとする。これから導かれるクリーンな点 $(x - \sigma_t\hat\epsilon)/\alpha_t$ が誤差 $\sigma_t\delta/\alpha_t$ を伴い、逆に誤差 δ を持つクリーンな点の予測から導かれるノイズが誤差 $\alpha_t\delta/\sigma_t$ を伴うことを示せ。それぞれ経路のどちらの端で増幅されるか。
   3. コサイン経路上では速度ターゲットが $\pi/2$ 倍の $\mathbf{v}$ ターゲットに等しく、したがって $\mathbf{v}$ 予測が速度予測になること、さらにノイズ予測から導かれる速度が誤差 $\pi\delta/(2\alpha_t)$ を伴うことを示せ。両端で誤差を有界に保つパラメータ化はどれか。
   4. Table 38.2 の変換式は任意のデータに対して成り立つ。混合分布 $\frac{1}{2}\mathcal{N}(-2, 0.25) + \frac{1}{2}\mathcal{N}(2, 0.25)$ とコサイン経路の $t = 0.4$ において、Equation 38.97 を数値的に確認せよ。点 x のグリッド上で、周辺速度を一度は各成分にその事後確率で重みを付けた $\dot\alpha_t x_0 + \dot\sigma_t\epsilon$ の事後平均として計算し、もう一度はノイズを加えた混合分布のスコアから計算せよ。両者は丸め誤差の範囲で一致するはずである。混合比の重みを 0.8 と 0.2 に変えたとき、計算中のどの数値が変化し、どの数値が変化しないか。
7. **Reflow は周辺分布を保つが停滞し得る。**
   1. 条件付き分布が点質量となる直線経路に対して、Reflow の命題の (i) を詳細に証明せよ。コンパクトな台を持つ滑らかなテスト関数 φ に対し、弱形式の連続の式である $\frac{d}{dt}\mathbb{E}\,\varphi(\mathbf{X}_t) = \mathbb{E}\big[\nabla\varphi(\mathbf{X}_t)\cdot\mathbf{u}_t(\mathbf{X}_t)\big]$ を示せ。条件付き期待値の塔の性質はどこで使われ、リプシッツ条件の仮定はどこで使われるか。
   2. π_k を k ラウンド後のペアリングとする。$\mathbb{E}_{\pi_k}\|\mathbf{x}_1 - \mathbf{x}_0\|^2$ が単調非増加かつ W_2^2 で下から有界であり、したがって収束することを示せ。
   3. 1 ラウンドで何も変化しない（$\pi_{k+1} = \pi_k$）ならば、$C = 0$ が π_k に対して成り立つことを示せ。1 次元における不動点を特定せよ。
   4. 2 次元において、各 $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ を $\mathbf{R}\mathbf{x}_0$（ただし $\mathbf{R}$ は 4 分の 1 回転行列）とペアにする。線分 $\big((1-t)\mathbf{I} + t\mathbf{R}\big)\mathbf{x}_0$ が決して交差せず（この行列は任意の t で正則である。固有値が $(1-t) \pm \mathrm{i}\,t$ となるため）、したがって $C = 0$ となって Reflow が何も変化させない一方、コストが $\mathbb{E}\|\mathbf{R}\mathbf{x}_0 - \mathbf{x}_0\|^2 = 4$（これに対し $W_2^2 = 0$）となることを示せ。どのような行列 $\mathbf{A}$ に対して線形変換 $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ は凸関数の勾配となり、したがって $\mathcal{N}(\mathbf{0}, \mathbf{I})$ とその像との最適ペアリングを与えるか。
8. [code] **フローマッチングモデルに対する確率的サンプラ。●**
   1. 直線経路 $\alpha_t = 1 - t$、$\sigma_t = t$ 上で Equation 38.97 を逆に解き、$\nabla \log p_t(\mathbf{x}) = -\big(\mathbf{x} + (1-t)\,\mathbf{u}_t(\mathbf{x})\big)/t$ を導け。
   2. 任意の $g \geq 0$ に対し、確率微分方程式 $d\mathbf{X} = \big[\mathbf{u}_t(\mathbf{X}) - \frac{1}{2}g^2\nabla\log p_t(\mathbf{X})\big]dt + g\,d\bar{\mathbf{W}}$ を $t = 1$ から $t = 0$ まで（時間を逆向きに走るブラウン運動 $\bar{\mathbf{W}}$ を用いて）積分した解が周辺分布 p_t を持つことを示せ。ただし $\tau = 1 - t$ と置換し、τ に関するフォッカー・プランク方程式を書いて $\nabla\cdot(p\nabla\log p) = \Delta p$ を用いよ。ここでスコア項が負号を伴って現れるのはなぜか。以上により、訓練済みの速度場は確率的サンプラの族を定める。
   3. 訓練済みの two-moons 速度場に対し、$g \in \{0, \frac{1}{2}, 1\}$ および $K \in \{8, 32, 128\}$ ステップでオイラー・丸山法を用いてこのサンプラを実装せよ。スコアは $t = 1, 1 - 1/K, \ldots, 1/K$ のみで評価し（$t = 0$ で特異になるため）、各実行の二乗エネルギー距離を報告せよ。与えられた品質を達成するのに最も少ないステップ数で済むのはどの g であり、$K = 8$ においてノイズは有効か。

4. Reflow 後の速度場に対して同じ実験を繰り返せ。その軌跡の直線性から、1 ステップの確率的サンプルが決して 1 ステップの決定論的サンプルに匹敵し得ない理由を説明せよ。

38.8 関連文献

これまでの 7 つの節では、1 つの点の軌跡から出発してノイズをデータへと変換するフローに至るまでをたどってきたが、その途上では数多くの問いに立ち止まることなく通り過ぎてきた。物理系におけるノイズを記述するのはどの確率積分（stochastic integral）か。モデル自体が SDE であるとき、随伴系はどうなるのか。以下の各項目では、そうした問いを 1 つずつ取り上げ、その答えがどこにあるかを示す。

拡散モデル（diffusion model）の実践は第 17 章とその関連文献（17.10 節）の守備範囲であるため、そこで扱われる手法については参照先を示すにとどめる。残る話題は数学である。まずは次の 3 つの項目から読むことを勧める。拡散サンプラが少数のステップで済む理由の 1 つである指数積分器（exponential integrator）、拡散する密度が何を下っていくのかを述べるワッサースタイン勾配フロー（Wasserstein gradient flow）、そして最適なペアリングの確率版にあたるシュレーディンガーブリッジ（Schrödinger bridge）である。残りの項目は、読者によって抱くかもしれない疑問に答えるものである。38.7 節で読み終えたとしても、本書の残りの部分で用いる内容を何ひとつ見落とすことはない。

38.8.1 ソルバ

38.8.1.1 指数積分器

学習されたスコアを用いた VP 過程の確率フロー ODE(38.6.3 節)である $\dot{\mathbf{x}} = -\frac{1}{2}\beta\,\mathbf{x} - \frac{1}{2}\beta\,\mathbf{s}_\theta(\mathbf{x}, t)$ は、閉じた形で既知の線形項と学習された部分の和からなる半線形（semilinear）方程式である。ルンゲ・クッタ法のステップは両方を離散化するため、線形項も誤差に寄与するうえ、学習された部分がどれほど滑らかであっても 38.2.3 節の安定性限界によってステップ幅が制約される。線形項は厳密に解ける以上、これは無駄である。指数積分器は、学習された部分を駆動線形方程式（38.1.2.3 節）への入力として扱う。すなわち、定数変化法（variation of constants）の公式によって線形項による状態の時間発展を厳密に運び、学習された部分の積分だけを近似するのである。

ステップの間、学習された部分を一定に保てば、ネットワークの出力を入力とする同項のゼロ次ホールドになる。その出力が変化しなければこのステップは厳密になるが、実際には変化するため 1 次精度となる。実は本章でも、この名前こそ出さなかったものの同じステップを一度用いている。$\sqrt{\bar{\alpha}_t}$ で割る操作（38.6.4 節）によって線形項が取り除かれ、DDIM は 1 ストライドにわたってノイズ予測を一定に保つ。つまり DDIM は 1 次の指数積分器である。

この系統のサンプラは、どの出力（スコア、ノイズ予測、またはクリーンな点の予測）を一定に保つか、そしてどの時計を用いるかという点で異なる。より高次の手法では、定数の代わりに複数の予測値を通る多項式を用いる。その理論は Hochbruck and Ostermann (2010) が概観している。これに基づく拡散サンプラには DEIS (Zhang and Chen 2023) や DPM-Solver (Lu, Zhou, et al. 2022) がある。後者は演習問題 3（17.5 節）の題材であり、その後継手法は 17.10 節に挙げられている。SDE に対しても同じ考え方から、演習問題 7（38.4 節）の指数オイラー・丸山法のステップが得られる。

38.8.1.2 幾何学的数値積分とリープフロッグ法

摩擦のない振動子に対して前進オイラー法はステップごとにエネルギーを増加させ、後退オイラー法はエネルギーを減少させるため、長時間実行すると前者は発散し、後者は静止してしまう。よ

り高次の手法を用いても、これを先延ばしにできるにすぎない。**幾何学的数値積分**（geometric integration）が手法に求めるのは別の性質、すなわち厳密なフローを精度よく追跡するかどうかとは無関係に、厳密なフローが持つ構造を保存することである。シンプレクティックオイラー法やリープフロッグ法のような**シンプレクティック**手法は、厳密なフローと同様に、位置と運動量の平面における面積を保存する（座標が増えれば面積は体積になる）。

さらに後退誤差解析（backward error analysis）によれば、このような手法は $1/h$ に関して指数関数的に小さい誤差を除いて、わずかに異なるエネルギーに対する厳密なフローを追跡する。したがって真のエネルギーにおける誤差は、ホライズンとともに増大するのではなく、非常に長い時間にわたって有界にとどまる。演習問題 4（38.2 節）では、振動子に対するシンプレクティックオイラー法についてこの異なるエネルギーを求める。その等位集合は、真のエネルギーが描く円に近い楕円になる。勾配降下法にもエネルギーを損失に置き換えた同じ解析が成り立ち、損失に加わる項は Barrett and Dherin (2021) の暗黙的勾配正則化項（implicit gradient regularizer）である（同節の演習問題 5）。この理論は Hairer et al. (2006) が展開している。

リープフロッグ法は、ハミルトニアンモンテカルロ法 (Duane et al. 1987; Neal 2011) の内部で用いられる数値積分器でもある。このサンプラは、目標分布の負の対数密度をポテンシャルエネルギーとして持つ摩擦のない粒子をシミュレーションすることで遠くへの移動を提案し、メトロポリス則（36.8.3 節）によって採択または棄却する。リープフロッグ法のエネルギー誤差が有界にとどまるため、遠くへの移動であっても採択され得る。さらに、リープフロッグ法のステップは可逆で体積を保存するため、この判定則は 2 つのエネルギーを比較するだけでよく、ヤコビ行列を必要としない。

38.8.2 連続時間モデル

38.8.2.1 行列式を安価に計算できるフロー

離散的な正規化フロー（normalizing flow）は、ヤコビ行列の行列式を安価に計算できるよう各層に制約を課す。38.3.3 節では通常の制約として三角ヤコビ行列を挙げたが、層がどのようにしてそれを得るかは述べなかった。カップリング層 (Dinh et al. 2014; Dinh, Sohl-Dickstein, et al. 2017) は、座標の半分をそのまま残し、もう半分を前半の座標の関数によってシフトおよびスケーリングする。そのヤコビ行列は三角行列となり、逆変換は減算と除算だけで計算できる。1 つの層では状態の半分しか変化しないため、多数の層を積み重ね、層ごとに役割を入れ替える。

可逆残差ネットワーク（invertible residual network）(Behrmann et al. 2019) は別の対象に制約を課す。残差ブロックはオイラー法の 1 ステップであり（38.3.1 節）、分岐のリプシッツ定数を 1 未満に抑えれば、リプシッツ連続なベクトル場のフローと同様に（38.1.1.4 節）、そのステップは可逆になる。ヤコビ行列は一般の形状を保ち、その対数行列式は演習問題 4（38.3 節）のべき級数によって推定される。Papamakarios et al. (2021) はこれらのモデル族と、それぞれが何を犠牲にしているかを概観している。

38.8.2.2 可逆ソルバと正則化フロー

38.3.2 節の随伴法（adjoint method）は、逆方向の計算中に順方向の状態を再構成することでメモリを節約するが、縮小するフローではその再構成に伴う丸め誤差が増幅される。再構成には縮小がなくても生じる第 2 の誤差がある。ソルバの逆方向ステップは順方向ステップの逆写像に近いものの、厳密な逆写像ではない。そのため逆向きにたどって得られる状態は順方向に送り出した状態と一致せず、得られる勾配も実際に実行した計算の勾配ではなくなる。可逆ソルバ（reversible solver）は各ステップを代数的に元へ戻せるように構成されており、この第 2 の誤差を取り除く。ただし丸め誤差は依然として増幅されるため、硬い（stiff）ベクトル場に対してはチェックポイント保存（checkpointing）の方が引き続き安全な選択肢である。両者の比較は Kidger (2022)（第 5 章）にある。

尤度によって訓練されたフローには、数値積分が安価になるべき理由がない。尤度は各点が最終的にどこへ到達するかだけで決まり、たどった経路には依存しないからである。そこで Finlay et al. (2020) は、訓練中に運動エネルギー $\int \mathbb{E}\|\mathbf{f}_\theta\|^2\, dt$ にペナルティを課す。固定された 2 端点間では等速の直線経路が最小の運動エネルギーを持つため、このペナルティによって経路は直線に近づき、ソルバは直線経路を少数のステップで走破できる。フローマッチングは、直線的な回帰目標とノイズ・データのペアリング（38.7.3 節）を通じて同じ目標を追求する。

38.8.2.3 潜在微分方程式と制御微分方程式

本章のモデルは初期状態を最終状態へと写像する。しかし患者の診療記録のような時系列データは不規則な時刻に到着するため、そのモデルには各観測が得られた時点でそれを順次取り込むことが求められる。潜在 ODE（latent ODE）(Rubanova et al. 2019) はその半分を実現する。すなわち、観測列を 1 つの初期状態へエンコードし、任意の時刻集合における軌跡をデコードするため、後から到着した観測が進路を変えることはできない。

ニューラル制御微分方程式（neural controlled differential equation）(Kidger et al. 2020) では、すべての観測が状態に作用できる。この手法は補間されたデータ経路 $\mathbf{X}_t$ そのものによって状態を駆動し、$d\mathbf{z} = \mathbf{f}_\theta(\mathbf{z})\, d\mathbf{X}_t$（ただし $\mathbf{f}_\theta$ は行列値関数）と表される。この方程式はブラウン経路をデータに置き換えた SDE の形をしているが、補間された経路は滑らかであるため、積分は通常の積分になる。そのオイラーステップ $\mathbf{z}_{n+1} = \mathbf{z}_n + \mathbf{f}_\theta(\mathbf{z}_n)(\mathbf{X}_{n+1} - \mathbf{X}_n)$ は、データの増分を読み取るリカレントネットワークにほかならない。

38.8.2.4 相互作用する粒子としての Transformer

残差ネットワークはオイラーステップによって 1 つの状態を動かす（38.3.1 節）。一方、Transformer は系列内のすべてのトークンを同時に動かし、アテンションによって各トークンの速度が他のトークンの位置に依存するようになる。つまりトークンは相互作用する粒子である。Geshkovski et al. (2023) は、層正規化によってすべてのトークンが単位球面上に保たれる理想化されたモデルでこの系を解析し、深さが増すにつれてトークンがクラスタへと集まることを示している。本章で扱った系との唯一の違いは、ベクトル場が自ら動かす密度に依存するため、連続の式（continuity equation）が密度に関してもはや線形ではなくなる点である。

38.8.3 ノイズ

38.8.3.1 ワッサースタイン勾配フロー

勾配フロー上の点は損失を下っていく（38.2.4 節）。拡散過程の密度もまた、フォッカー・プランク方程式に従って決定論的に時間発展する。では、密度も何かを下っているのだろうか。ドリフトが勾配 $\mathbf{f} = -\nabla V$ であり、かつ g が定数であるときには、確かに下っている。このとき密度は自由エネルギー（free energy）$\mathcal{F}(p) = \int V p\, d\mathbf{x} + \frac{g^2}{2}\int p \log p\, d\mathbf{x}$ を下る。第 1 項は平均ポテンシャルエネルギーであり、すべての質量が V の底に集まるとき最小になる。第 2 項は負のエントロピーであり、質量が広がっているとき最小になる。ドリフトは第 1 項に、ノイズは第 2 項に働き、両者は e^{-2V/g^2} に比例する定常密度において釣り合う。

最急降下方向は距離の測り方に依存し、ここで 2 つの密度間の距離として用いなければならないのは、一方を他方へ輸送するコストを測る 2 次ワッサースタイン距離である。距離がどこに現れるかは 38.2.4 節の後退オイラーステップを見ればわかる。凸な損失に対して、このステップは $\mathcal{L}(\mathbf{x}) + \frac{1}{2\eta}\|\mathbf{x} - \mathbf{x}_k\|^2$ を最小化する。JKO スキーム (Jordan et al. 1998) は、このステップの点を密度に、二乗距離を輸送コストに置き換えた $p_{k+1} = \arg\min_p \ \mathcal{F}(p) + \frac{1}{2\eta} W_2^2(p, p_k)$ である。$\eta \to 0$ のとき、そのステップ列はフォッカー・プランク方程式の解へと収束する。

とりわけ記憶にとどめておくべき例がランジュバン動力学である。そのドリフトは目標分布のスコアの半分であり、ノイズは $g = 1$ を持つ（38.6.2 節）。目標分布を p_∞ と書き、粒子群の密度を表す p と区別しよう。すると $V = -\frac{1}{2}\log p_\infty$ となり、自由エネルギーは $\frac{1}{2}D_{\mathrm{KL}}(p \,\|\, p_\infty)$ となる。すなわちランジュバン粒子の群れは、輸送によって距離を測ったもとで、目標分布に対する KL ダイバージェンス上の勾配降下法を実行しているのである。連続の式からベナムー・ブレニエ (Benamou and Brenier 2000) に至るまで、本章で輸送が繰り返し姿を現す理由はここにある。

38.8.3.2 物理ノイズとウォン・ザカイの定理

38.4.2.2 節では左端点を用いて伊藤積分を定義し、ストラトノビッチ積分を定義する中点則では通常の連鎖律が保たれることを述べた。では、物理系を記述するのはこの 2 つのうちどちらか。物理系におけるノイズはブラウン経路ではない。短い記憶と滑らかな経路を持っており、滑らかな経路に対しては通常の積分ただ 1 つしか存在しない。Wong and Zakai (1965) は、そのような滑らかな経路がブラウン経路に近づくとき、通常の微分方程式の解がストラトノビッチ解へと収束することを証明した。1 次元において、$dX = f\,dt + g\,dW$ をストラトノビッチの意味で解釈した方程式は、ドリフトを $f + \frac{1}{2}g\,\partial_x g$ とした伊藤方程式に等しい。

拡散モデルの加法的ノイズ（additive noise）に対しては、この 2 つの解釈は一致する。乗法的ノイズ（multiplicative noise）に対しては両者は異なり、どちらを選ぶべきかをこの定理が教えてくれる。滑らかな物理ノイズを理想化した方程式はストラトノビッチとして解釈する。一方、離散時間で構築され、係数が自らの掛ける増分を先回りして見ることのできないモデルは伊藤として解釈する。確率的勾配降下法のステップはこの後者の種類に属する。

38.8.3.3 SDE としての確率的勾配降下法

勾配降下法は勾配フローに対する前進オイラー法である（38.2.4 節）。では、確率的勾配降下法は何の離散化なのだろうか。ミニバッチ勾配は全勾配にノイズを加えたものであり、その共分散 Σ は $1/B$ に比例してバッチサイズ B とともに縮小する。Q. Li et al. (2017) は SDE として $d\theta = -\nabla\mathcal{L}\,dt + \sqrt{\eta}\,\Sigma^{1/2}d\mathbf{W}$ を提案している。長さ η のオイラー・丸山法のステップをとってみよう。ノイズ係数にさらに $\sqrt{\eta}$ が掛かり、結果として学習率 η の確率的勾配降下法の 1 ステップが得られる。

したがって学習率は、ステップ幅として、そしてノイズの内部に、計 2 回現れる。Σ がいたるところ単位行列の同じ定数倍 $c\,\mathbf{I}$ であるとき、先ほどの自由エネルギーが $V = \mathcal{L}$ および $g^2 = \eta c$ としてそのまま当てはまり、損失に対してエントロピーの重みを定める。この意味において、学習率は温度である。二次損失の上では、反復列は OU 過程（38.4.3 節）の定常ガウス分布へと落ち着き、その分散は比 η/B とともに増大する。Mandt et al. (2017) はこの性質を利用し、確率的勾配降下法を近似的な事後分布サンプラとして動かしている。ただしこのモデル化には小さな学習率とほぼガウス分布に従う勾配ノイズが必要であり、実用上はその両方の仮定が破れることがある。

38.8.3.4 ニューラル SDE

学習されるモデル自体が、学習されるドリフトと学習されるノイズ振幅 g を持つ SDE であるとき、随伴系はどうなるのか。その場合、随伴系は同じブラウン経路で駆動される後向き SDE（backward SDE）となるため、そのブラウン経路を保存しておくか、乱数シードから再生成しなければならない (X. Li et al. 2020)。損失もまた変化する。このようなモデルの出力は確率的であるため、1 つの端点を 1 つの目標値と直接比較することはできない。訓練信号は、経路全体の法則（確率分布）間のダイバージェンス、あるいは経路同士を比較する識別器になる (Kidger 2022)。

ギルサノフの定理（Girsanov's theorem）(Øksendal 2003) は、同じ g とドリフト $\mathbf{f}$ および $\mathbf{f}'$ を持つ 2 つの過程に対して、このダイバージェンスを閉じた形で与える。両者の経路法則間の KL

ダイバージェンスは、初期法則間の KL ダイバージェンスに $\frac{1}{2}\int \mathbb{E}\|\mathbf{f}-\mathbf{f}'\|^2/g^2\,dt$（期待値は第 1 の過程の経路に関するもの）を加えた値に等しい。ここで g が一致するという条件は外せない。異なる g を持つ 2 つの過程は、たった 1 本の経路からでもその二次変分（38.4.2.1 節）によって見分けられるため、経路法則間の KL ダイバージェンスは無限大になる。

37.3.3.3 節のデータ処理不等式（data-processing inequality）により、端点の法則間の差は経路の法則間の差を超えられない。したがって、拡散モデルの厳密な逆ドリフトと学習された逆ドリフトにギルサノフの公式を適用すれば、演習問題 7（38.6 節）で経路法則を用いずに導出した尤度境界が得られる。

38.8.4 学習される生成器

この方向で本章の先に続く内容の大半は第 17 章で扱う。予測対象、損失の重み付け、およびスケジュールについては 17.6 節で解説しており、そこでは信号対雑音比（signal-to-noise ratio）の対数である $-2\log\bar{\sigma}_t$（38.6.4 節の記法）によって各選択肢が整理される (Kingma et al. 2021)。1 ステップまたは少数ステップでの生成は 17.8 節で扱う。以下では、両章の範囲を超える 4 つのトピックを取り上げる。

38.8.4.1 スライススコアマッチング

38.6.1 節ではトレースの計算コストを理由に陰的スコアマッチング（implicit score matching）を脇に置き、そこでは採用しなかった抜け道として、ランダムな方向ごとに 1 回のベクトル・ヤコビ行列積で済む Hutchinson の推定値に言及した。この推定値を用いて訓練する手法はスライススコアマッチング（sliced score matching）と呼ばれる (Song et al. 2019)。この手法はノイズレベルを選ぶ必要がなく、データそのもののスコアを学習する。もっとも、拡散モデルにはこの利点の使い道がない。拡散モデルが必要とするのはあらゆるノイズレベルにおけるノイズ付きデータのスコアであり、ノイズ除去スコアマッチングはトレースなしでそれを提供するため、ノイズ除去が標準となったのである。

38.8.4.2 スコア誤差とサンプリング誤差

本章のほぼすべてのサンプラは厳密なスコアを用いて動かしたが、訓練されたスコアはどこにおいても厳密ではない。では、どれほど正確であればよいのだろうか。平均的に良ければ十分であり、それ以上は必要ない。Sitan Chen et al. (2023) は、データ分布と離散化された逆 SDE の出力との間の全変動（total variation）を 3 つの項の和で上から抑えており、その各項はすでに本章で個別に登場している。p_T とサンプラの出発点であるガウス分布との距離は e^{-T} のオーダーで減衰し、離散化誤差は次元とともに増大しステップ幅とともに減少する。この 2 つは 38.5.5 節で測定した。第 3 の項はノイズ付きデータのもとでのスコアの二乗平均平方根誤差であり、これはノイズ除去損失がその下限からの超過分として測っている量にほかならない。この定理はそれ以外の項が一切入らないことを示しており、仮定もわずかである。すなわち、データの有限な二次モーメントと、ノイズ付き密度のスコアがリプシッツ連続であることだけを要求する。単峰性は不要であるため、38.6.2 節でランジュバン動力学を阻んだ 2 つのモードも対象に含まれる。

38.8.4.3 シュレーディンガーブリッジとエントロピー輸送

1931 年、Schrödinger は粒子に関するある問いを立てた。自由に拡散する粒子群が、現在ある形状で観測され、後の時刻に別の起こりそうにない形状で観測されたとき、そこへ至る最も確からしい道筋はどのようなものか、という問いである。独立な多数の粒子からなる群れでは、自由拡散とは異なる経路法則に従って群れが動くのを目にする確率は指数関数的に減少し、その減衰率は自由拡散に対するその経路法則の KL ダイバージェンスで与えられる。したがって答えは、与

えられた 2 つの周辺分布を持つ経路法則のうち、KL ダイバージェンスの意味で自由拡散に最も近いものとなる。これを**シュレーディンガーブリッジ**と呼ぶ。

このブリッジを 38.7.3 節のミニバッチペアリングと並べてみよう。ミニバッチペアリングは各バッチ上での厳密な割り当てであり、訓練される生成器は ODE である。これに対しブリッジは、37.4.2.4 節のシンクホーン反復によって計算されるエントロピー正則化最適輸送の計画 (Cuturi 2013; Peyré and Cuturi 2019) に従って両端を対応付け、その生成器は SDE となる。自由拡散のノイズが小さくなるにつれて、この輸送計画は最適輸送計画へと近づく。

拡散モデルと並べると、ブリッジの方が難しい問題である。拡散モデルは順過程を固定して逆過程だけを学習し、事前分布には近似的にしか到達しない。一方ブリッジは与えられた 2 つの法則を有限時間で結ぶため、両方向を学習しなければならない。De Bortoli et al. (2021) は順方向と逆方向の拡散を交互に適合させており、その交互反復の最初の 1 巡は拡散モデルの訓練に一致する。Albergo et al. (2023) はブリッジ問題を解くことなく、ODE と SDE の両方の生成器に到達している。任意の 2 つの分布からのサンプル $\mathbf{x}_0$、$\mathbf{x}_1$ と、独立なガウス変数 $\mathbf{w}$ をとろう。経路 $a_t\mathbf{x}_0 + b_t\mathbf{x}_1 + c_t\mathbf{w}$（ただし両端で c_t はゼロ）が持つ周辺分布は、1 つの ODE または SDE の族によって生成できる（演習問題 1、38.7 節）。

38.8.4.4 直線的カップリングと最適カップリング

Reflow によって変化しないペアリングは交差しない直線経路を持つが、2 次元以上ではそれが最適であるとは限らない。38.7.3 節の例では、ガウス分布に従う点群の各点を、それを 4 分の 1 回転させた点と対応付けている。線分は決して交差しないが、コストが 4 かかるのに対し、本来は $W_2^2 = 0$ である。このペアリングが持ち、最適なペアリングが持たないものは回転である。最適なフローの速度場は勾配であるが (Benamou and Brenier 2000)、点群を原点のまわりに回転させるベクトル場は勾配ではない。

そこで Liu (2022) は速度場を勾配 $\nabla\phi_t$ に制限し、この制限付き Reflow によって変化しないペアリングが最適なペアリングになると論じた。しかし、この主張が成り立つにはそこで置かれた仮定以上の条件が必要である。Hertrich et al. (2025) は、移動中の点群があらゆる時刻で空間全体を覆っている場合にこれを証明する一方、点群が複数の離れた塊に分かれている場合の反例を与えている。したがってこの制限だけでは、38.7.3 節が残した問い、すなわち訓練のみによって最適なペアリングをいかにして見つけるかという問いは決着しない。

38.8.5 ダイナミクスと生成フロー全般の参考資料

以下の文献を開く前に 1 点注意しておく。MIT の講義ノートと講義、Flow Matching Guide、およびフローマッチングに関する 2 編の論文では、時刻を 0 のノイズから 1 のデータへと進めるのに対し、本章ではその逆向きに進めている（Figure 38.26）。

書籍と講義ノート

- 『Nonlinear Dynamics and Chaos』（Steven Strogatz 著）：フロー、不動点とその安定性に関する幾何学的な入門書であり、38.1 節の内容を扱っている。
- 『On Neural Differential Equations』（Patrick Kidger 著、2022 年）：ニューラル ODE、制御微分方程式、確率微分方程式を書籍規模で体系的に論じた文献であり、38.3 節の随伴法の仕組みが余すところなく展開されているほか、38.2 節のソルバの選び方に関する助言も収められている。
- 『Applied Stochastic Differential Equations』（Särkkä、Solin 著）：38.4 節および 38.5 節に対応する伊藤解析、SDE、フォッカー・プランク方程式を応用重視の視点から解説した書籍であり、著者らによって全文の PDF が公開されている。

- 『Stochastic Differential Equations: An Introduction with Applications』（Bernt Øksendal 著）：38.4 節の背景にあるブラウン運動、伊藤積分、および解の存在理論に関する厳密な基本文献である。
- 『Numerical Solution of Stochastic Differential Equations』（Kloeden、Platen 著）：38.4 節のシミュレーション部分に対応する、オイラー・丸山法、ミルスタイン法、より高次のスキーム、およびそれらの強収束次数・弱収束次数に関する基本文献である。
- 「An Introduction to Flow Matching and Diffusion Models (MIT 6.S184 lecture notes)」（Holderrieth、Erives 著）：SDE と連続の式から拡散モデルとフローマッチングを導出する自己完結した講義ノートであり、38.6 節と 38.7 節の副読本となる。

講義とチュートリアル

- 「MIT 6.S184: Generative AI with Stochastic Differential Equations」：38.5 節から 38.7 節への道筋に沿って第一原理から拡散モデルを構築する講義、スライド、および演習ラボであり、オープンライセンスで公開されている。
- 「Understanding Diffusion Models: A Unified Perspective」（Calvin Luo 著、2022 年）：38.6 節に関連して、拡散モデルの変分的な見方とスコアベースの見方を並べて導出している。
- 「Flow Matching Guide and Code」（Lipman ら著、2024 年）：38.7 節向けのフローマッチングに関する詳細な解説文献であり、PyTorch ライブラリ `facebookresearch/flow_matching` と対になっている。

基礎論文

- 「Neural Ordinary Differential Equations」（Chen ら著、NeurIPS 2018）：38.3 節と同様に、ネットワークのダイナミクスを ODE としてパラメータ化し、随伴法を用いて訓練する。
- 「Generative Modeling by Estimating Gradients of the Data Distribution」（Song、Ermon 著、NeurIPS 2019）：38.6 節に対応する、ノイズ除去スコアマッチングとアニーリング付きランジュバンサンプリングによるスコアベース生成を提案している。
- 「Denoising Diffusion Probabilistic Models」（Ho、Jain、Abbeel 著、NeurIPS 2020）：38.6 節で離散化された SDE として解釈した、離散時間の DDPM 定式化を与えている。
- 「Score-Based Generative Modeling through SDEs」（Song ら著、ICLR 2021）：スコアマッチングと拡散モデルを順方向・逆方向の SDE として統一し、38.5 節の確率フロー ODE を導入している。
- 「Denoising Diffusion Implicit Models」（Song、Meng、Ermon 著、ICLR 2021）：訓練済みの DDPM を再利用してより少ないステップ数で生成を行う、38.6 節の決定論的サンプラである。
- 「Flow Matching for Generative Modeling」（Lipman ら著、ICLR 2023）：条件付きベクトル場の回帰によって連続フローをシミュレーションなしで訓練する、38.7 節の手法である。
- 「Flow Straight and Fast: Rectified Flow」（Liu、Gong、Liu 著、ICLR 2023）：38.7 節に対応する、少数ステップサンプリングのための直線補間経路と Reflow を提案している。

ブログと視覚的解説

- 「Generative Modeling by Estimating Gradients of the Data Distribution」（Yang Song 著）：スコアマッチング、ランジュバン動力学、および 38.6 節の SDE による描像を図とともに結びつける著者自身の解説記事である。
- 「What are Diffusion Models?」（Lilian Weng 著）：DDPM からスコア SDE やガイダンスに至る拡散モデルのサーベイ記事であり、2024 年まで更新されている（38.6 節に対応）。
- 「Perspectives on Diffusion」（Sander Dieleman 著）：拡散モデルがいかにしてノイズ除去器、スコア推定器、逆 SDE ソルバ、そしてフローという顔を併せ持つかを示す記事であり、38.7.1.3 節の対応辞書にあたる。

- 「The Annotated Diffusion Model」（Hugging Face）：38.6 節の数学を実行可能なコードへと結びつける、PyTorch による DDPM の 1 行ごとの実装解説である。

深層学習のためのツール

ノートパソコン上で動作するノートブックも、モデル、バッチ、あるいはコンテキストが大きくなった途端に失敗することがある。次に下すべき判断は、公称スループットが最も高いアクセラレータを無条件に購入することではない。まずは、ワークロードがメモリに収まるか、計算とデータ移動のどちらが律速しているか、そして実験を完遂するのにどれだけのコストがかかるかを見極める必要がある。さらに、訓練とサービングとでは課される制約が異なる。分散訓練ではアクセラレータ間の通信コストが発生するのに対し、サービングではモデルとキャッシュの永続的な状態を保持しつつ、レイテンシとスループットの目標も満たさなければならない。

本パートでは、単一のノートブックからデプロイされたシステムに至るまで、これらの判断を順に展開していく。自身の課題に関連する節を探す際には、次の表を参照するとよい。

タスク	節	中心となる問い
ローカルでの本書の実行と確認	39.1 節	再現可能な反復作業のために、コード・状態・出力をどう整理すべきか
ローカルへのインストールなしでの開始	39.2 節	どのホスト型環境が必要なフレームワークとアクセラレータを提供するか
計算リソースのレンタル	39.3 節	セットアップやデータ転送を含め、1 回の実行完遂にかかる総コストはいくらか
ハードウェアの選定	39.4 節	ワークロードが収まるか、また計算・メモリ帯域幅・インターコネクトのどれが律速するか
モデル・データセット・エビデンスの探索	39.5 節	どのアーティファクト・実装・ベンチマークが再現可能で信頼できるか
訓練のスケーリング	39.6 節	パラメータ・活性化値・データ・通信をどう分割すべきか
モデルのサービング	39.7 節	バッチ化・レイテンシ・スループット・キャッシュメモリはどう相互作用するか
本書への貢献	39.8 節	変更をどうテストし、4 つのフレームワーク間で一貫性を保つか
本書のユーティリティの参照	オンライン付録「ユーティリティ関数とクラス」とオンライン付録「d21 API リファレンス」	クラスや関数はどこで定義され、ドキュメント化されているか

システムを構築する読者は、この表を上から順にたどるとよい。運用上の具体的な疑問がある読者は、対応する節を直接参照できる。ハードウェアの名称、価格、クラウドのインターフェース、

ライブラリのバージョンは移り変わりが速いため、リンク先の各節ではそうした事実に時点を明記し、製品が変わっても引き続き役立つ測定を重視している。ユーティリティと `d2l` API のページは、学習順序の一部ではなく、検索可能なリファレンスである。

Chapter 39

深層学習のためのツール

大半の読者はまず、ノート PC 上やブラウザのタブ内で本書のコードを実行する。ノートブックを開き、セルを実行し、学習率を変更して再実行するといった具合である。しかし遅かれ早かれ、手元の環境には収まらない事態が生じる。モデルがノート PC の搭載量を超えるメモリを必要としたり、訓練の実行に 1 週間かかったり、週末のプロジェクト用にダウンロードしたオープンな言語モデルが毎秒 3 語しか書き出せなかったりするのである。これらはいずれも、どの製品を使うべきかという問いに見える。だがその根底では、同じ少数の問いが繰り返し現れる。ジョブはメモリに収まるか、何がその速度を制限しているか、そして完了した結果を得るのにどれだけのコストがかかるか。

製品は数か月のうちに移り変わるが、その答えはより長く通用する少数の原則に基づいている。そもそも実行できるかどうかを決めるのはメモリであるため、高速化を図る前にまずジョブがメモリに収まるかを確認する。言語モデルがテキストを生成する際には、新しいトークンごとに、使用するすべての重みをメモリから 1 回読み出すため、メモリ帯域幅（memory bandwidth）が生成速度を決める。時間あたりの料金が最も安いマシンであっても、ジョブの実行が遅かったり途中で失敗したりすれば最も安上がりな実験にはならないため、完了した結果あたりのコストを比較する。さらに、記録された内容から再実行できない結果は、検証することもその上に積み上げることもできないため、再現性（reproducibility）は最初のセルからワークフローに組み込まれていなければならない。第 13 章では、プログラムは算術演算、メモリ帯域幅、あるいは演算を発行するオーバーヘッドによって律速されていた。以上の原則は、何を実行し、借り、購入し、デプロイするかという判断にそのモデルを適用したものである。

本章では、単一の対話型ノートブックから複数マシンでの訓練、そして本番サービングへと段階的に進む。まずは日常的にコードを実行する環境から始める。結果が隠れた実行履歴に左右されないよう Jupyter カーネルの状態とローカルの uv 環境を管理する方法（39.1 節）、ローカルの環境構築なしで Colab や Kaggle 上でノートブックを実行する方法（39.2 節）、そしてレンタルしたクラウドアクセラレータの料金を完了した結果あたりの総コストで見積もる方法（39.3 節）を扱う。次に、ハードウェアと外部アーティファクトに目を向け、メモリ容量・メモリ帯域幅・演算性能の上限に基づいて購入すべき機材を選ぶ方法（39.4 節）と、公開ハブにある事前学習済みモデルやデータセットを探索・評価・ピン留め（pinning）し、ライセンスを確認する方法（39.5 節）を解説する。最後に、複数のマシンや貢献者へと規模を広げ、複数のアクセラレータに訓練を分割する手法（39.6 節）、目標レイテンシの下で同時接続ユーザーにサービングする手法（39.7 節）、そしてシングルソースの Markdown ファイルから本書自体がどのようにビルド・保守されているか（39.8 節）を見ていく。

価格、製品名、クォータ、ライブラリのバージョンは急速に変化する。これらを記載している各

節では時点を明記しているため、利用前に最新の情報を確認されたい。

生成されたユーティリティおよび d21 API のドキュメントは、引き続き検索可能な HTML リファレンスページとして提供されている。これらは通読する学習順序や PDF 版には含まれない。

参考資料と関連文献

以下の資料は、1 台のマシンでの作業から大規模な訓練とサービングに至るまで、本章の各節に対応している。いずれもオンラインで無料公開されている。

システムとスケーリング

- How to Scale Your Model — Austin et al., Google DeepMind (2025)：実測の前に性能を予測する方法を解説したオンライン書籍であり、ルーフライン、TPU と GPU の内部構造、並列性、および推論の算術を、39.4 節、39.6 節、39.7 節に対応する具体的な数値とともに展開している。
- The Ultra-Scale Playbook — Hugging Face (2025)：最大 512 基の GPU を用いた 4,000 回以上の実測ランに基づく GPU クラスタでの訓練ガイドであり、詳細なメモリバジェットと並列性のトレードオフによって 39.6 節の内容を掘り下げている。
- Stanford CS336: Language Modeling from Scratch：トークン化から分散訓練、推論に至るまで、言語モデルをエンドツーエンドで構築する YouTube 上の講義と GitHub 上の課題であり、39.6 節および 39.7 節の副教材となる講義コースである。
- GPU MODE lectures：CUDA の基礎から FlashAttention、量子化までを網羅した動画、スライド、およびノートブックであり、39.4 節で限界を見積もる対象となるカーネルを自ら書きたい読者に適している。

計算性能の考え方

- Making Deep Learning Go Brrrr From First Principles — Horace He (2022)：プログラムの実行時間を演算、メモリ帯域幅、およびオーバーヘッドに切り分ける短い論考であり、39.4 節の土台となる考え方である。
- Transformer Inference Arithmetic — kipply (2022)：KV キャッシュのサイズとデコーディングの帯域幅上限を手計算で導出しており、39.4 節および 39.7 節における見積もりの基礎となっている。
- Accelerating Generative AI with PyTorch II: GPT, Fast — PyTorch team (2023)：gpt-fast のコードを伴う解説記事であり、コンパイル、量子化、投機的デコーディング、テンソル並列を用いて、1,000 行未満の PyTorch コードで 10 倍近い推論の高速化を段階的に構築している。
- Which GPU for Deep Learning? — Tim Dettmers (2023)：最終更新が 2023 年 1 月であるためハードウェアの具体例は古くなっているが、ワークロードから GPU を選ぶ手法は 39.4 節を補完する。

サーベイ論文

- A Survey on Efficient Inference for Large Language Models — Zhou et al. (2024)：データ・モデル・システムの各レベルにおける推論の最適化手法を体系的に分類し、代表的手法の比較実験をまとめたサーベイであり、39.7 節の背景知識となる。
- A Survey on Large Language Model Acceleration Based on KV Cache Management — Li et al. (2024)：KV キャッシュ（39.7 節で導入）の保持、圧縮、および退避の手法を、精選された論文リストとともに解説している。

最新動向の把握

- r/LocalLLaMA と llama.cpp discussions：リリース報告、量子化の検証結果、ハードウェアの実測値、および再現の試みが集まる場であり、こうした情報源をどう吟味するかは 39.5 節で解説する。
- Pro Git — Chacon and Straub：無料の書籍であり、その第 1〜3 章と第 6 章では 39.8 節が前提とする Git とプルリクエストのワークフローを解説している。

39.1 ノートブック

ここまでに読んできたすべての節では、本文のすぐそばにコードが並び、各コードセルの下にその出力が表示されていた。読み進めるなかで、セルの一部を書き換えてみたいと思ったことがあるかもしれない。学習率を小さくしたり、層の幅を広げたり、訓練データを半分にしたりして、何が起きるか確かめるといった試みである。本書はまさにそのために書かれている。各節は 1 つの Jupyter ノートブックであり、文章・数式・コード・実行結果を 1 つのドキュメントに収め、1 セルずつ編集して実行できるようになっている。

セルを 1 つずつ実行すると、スクリプトでは決して生じない疑問が浮かび上がる。画面上の出力を生み出したのは、どのコードなのかという問題である。ノートブックは実行したすべての内容をその実行順に記憶するが、その順序はページ上の並びと一致するとは限らない。したがってノートブックには、再起動したが最後、作成者本人を含め誰にも再現できない結果が表示されうる。その対策は、結果を信用したり共有したりする前に、ノートブックを再起動して先頭から実行し直すことである。

以下では、ノートブックがどこに状態を保持し、それをどう確認するかを説明したうえで、手元のマシンに本書のノートブック環境を構築し、エディタやリモート GPU から操作する方法を解説する。何もインストールしたくない場合は、39.2 節の手順で同じノートブックを Colab や Kaggle 上で無料実行できる。ノート PC 以上のハードウェアが必要な場合は、39.3 節を参照されたい。

39.1.1 ノートブックの実行モデル

通常の Python スクリプトは毎回先頭から再実行されるため、データセットの読み込みや前処理のような時間のかかる処理が実験のたびに繰り返される。これに対しノートブックは、実行と実行のあいだも稼働し続ける Python プロセス（**カーネル**）を保持する。データを一度読み込めば、それをメモリに置いたまま何度でもモデルを試行錯誤できる。インラインの図表と解説文を織り交ぜられる点も相まって、この実行モデルは教育と研究の双方で威力を発揮する。

しかし同じメモリは、そこに値を書き込んだコードよりも長く生き残る。変数を代入したセルを編集したり削除したりしても、変数の値はそのまま残るのである。したがって、この**状態**（state）がどこに存在し、いつまで持続するのかを把握しておく必要がある。

39.1.1.1 ドキュメントとカーネル

ノートブックは 2 つの要素からなる。**ドキュメント**はセルと保存された出力を格納するものであり、読み書きや共有、コミットの対象となる。**カーネル**は稼働中の状態を保持するものであり、インポート済みモジュール、変数、乱数生成器（random number generator）、開いているファイル、コンパイル済みプログラム、およびアクセラレータのメモリがそこに含まれる。ドキュメントはセルを上から下へと並べるが、カーネルが知っているのはセルを**実行した**順序だけである。この 2 つの順序の混同が、ノートブックにおける多くのバグを引き起こす。

Figure 39.1 に典型的な失敗例を示す。編集セッションの途中で、まずデータ読み込みセルを実行し、あとからその**上**にモデル定義セルを書いたとする。カーネルはすでに `data` を保持しているため、その場では問題なく動作する。しかし、このドキュメントは実際には起きなかった経緯を語

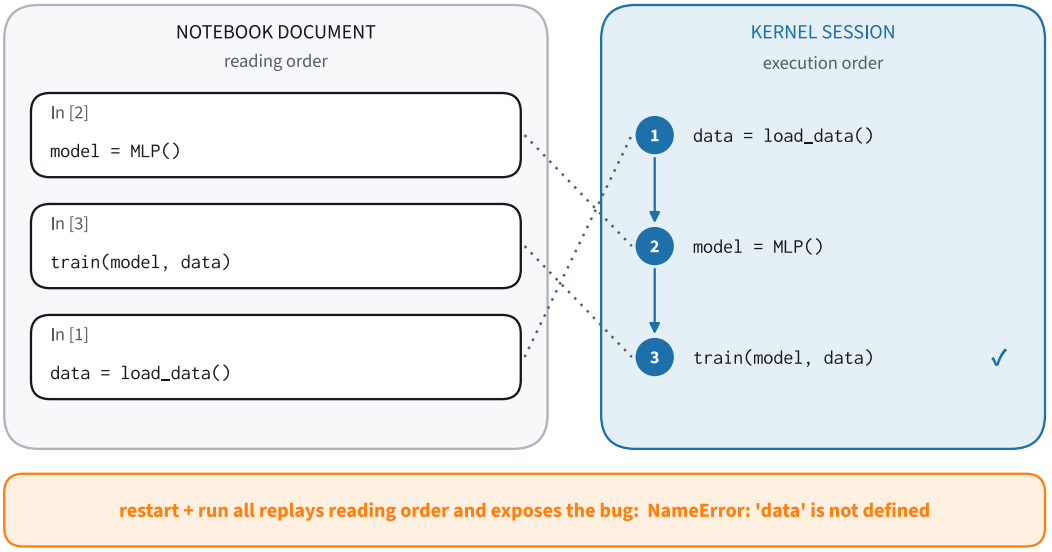


Figure 39.1: ドキュメントはセルを上から下へ並べ、カーネルは実行順に状態を蓄積する。カーネルの再起動と全セル実行は、ドキュメントの順序（読者が再現できる唯一の順序）を再現し、両者の不一致を暴き出す。

っていることになる。表示された順序どおりにセルを実行する読者や、翌日再起動したあとの作成者自身は、`NameError` に直面する。

実行カウンタ（`In [3]`、`In [1]`、…）は、カーネル起動以降のセルの実行回数を順に番号付けしたものであり、過去の履歴の手がかりではあっても依存関係グラフではない。次の 2 つのセルは、ドキュメントの順序に沿った無害な状態の例である。2 つ目のセルは、1 つ目のセルが実行されて初めて動作する。

```
import numpy as np

rng = np.random.default_rng(7)
samples = rng.normal(size=5)
```

```
float(samples.mean())
```

```
-0.2638849576850524
```

この依存関係は読み進める順序に従っているため問題ない。しかし、それより下にある代入や、すでに削除したセルに依存するセルは問題である。セルは小さく保ち、関数定義と実験の実行は分け、多数のセルにまたがってグローバル変数を書き換えるのではなくデータを明示的に受け渡すべきである。

39.1.1.2 カーネルの再起動と全セル実行

カーネルを再起動してすべてのセルを実行（Restart Kernel and Run All Cells）は、カーネルに蓄積された状態を破棄し、白紙の状態からドキュメントを読み進める順序どおりに再実行する操作である。これにより、ノートブックを共有する前にカーネルの隠れた状態や実行順序への依存

を検出できる。ノートブックの保存・共有・コミットの前には、必ずこれを実行する。本書のビルドでも同じテストを適用しており、すべてのノートブックを新しいカーネルで先頭から末尾まで実行し、ウェブサイト上の出力はその実行結果から生成している。

39.1.2 ローカル環境での本書の実行

39.1.2.1 セットアップ

パッケージのバージョンが変わると数値結果や API さえも変化するため、本書では環境をピン留めしている。インストール で説明した各フレームワーク用のノートブックアーカイブには、ノートブック一式、`d2l` パッケージ、および CPU 環境用と GPU 環境用の `uv` ロックファイルが含まれている。展開したディレクトリで一度だけ環境を作成し、そのなかで JupyterLab を起動する。

```
uv venv --python 3.12
uv pip sync pylock.cpu.toml
uv run --no-sync jupyter lab
```

NVIDIA GPU を搭載した Linux 環境では、代わりに `pylock.gpu.toml` を同期する。`--no-sync` オプションは、ロックファイルで確定した環境を uv が書き換えないようにするための指定である。GitHub リポジトリをクローンした場合は、代わりに `make` を使って環境を構築する（設定手順は 39.8 節で解説する）。

ノートブック内で選択する**カーネル**は、パッケージをインストールした環境に属していなければならない。1 つの Jupyter サーバーから複数のカーネルを参照できるため、サーバー自身の Python プロセスがどのカーネルでノートブックを実行するかを決めるわけではない。

39.1.2.2 簡単な事前確認

コードを変更する前に、実行環境の簡潔な識別情報を記録しておこう。

```
import os
import platform
import sys

{
    "python": sys.executable.replace(os.path.expanduser("~"), "~"),
    "version": platform.python_version(),
    "working_directory": os.path.basename(os.getcwd()),
}
```

```
{'python': '~/d2l-neu/.venv-pytorch/bin/python',
 'version': '3.12.3',
 'working_directory': 'chapter_appendix-tools-for-deep-learning'}
```

これにより、環境構築でよくある 2 つのミスを 1 つのセルで検出できる。別の環境のカーネルを選んでしまうミス（`sys.executable` が想定外の場所を指す）と、データへの相対パスが解決できないディレクトリからノートブックを開いてしまうミスである。このセルはホームディレクトリを `~` に省略し、作業ディレクトリの末尾の要素だけを表示するため、保存された出力から個人固有のパスが漏れることはない。

39.1.3 エディタでの作業

39.1.3.1 JupyterLab

JupyterLab は、ファイルブラウザ、ノートブックエディタ、ターミナル、テキストエディタ、デバッガ機能、および実行中カーネルの一覧表示を統合した環境である。インターフェースが進化しても、基本操作は変わらない。

- `Shift+Enter` でセルを実行し、`uv` や Git の操作、ファイルの確認にはターミナルタブを使う。
- 環境名でカーネルを選択し、続いて `sys.executable` を確認する。
- 長時間の計算は、再起動する前にまず割り込み（Interrupt）で停止させる。再起動すると Python の状態が解放され、通常はそのプロセスが保持していたアクセラレータのメモリも解放される。
- 他者に渡す結果を保存する前には、**カーネルを再起動してすべてのセルを実行**を使う。
- **実行中**（Running）パネルを確認し、使わなくなったカーネルを停止する。ブラウザのタブを閉じてもカーネルは停止せず、放置された孤立カーネルが何ギガバイトもの GPU メモリを占有し続けることがある。

39.1.3.2 VS Code

Visual Studio Code では、カーネル選択、変数インスペクタ、セル単位のデバッグ、ノートブック構造を認識する差分表示を備えており、`.ipynb` ファイルの編集と実行ができる。展開したディレクトリをフォルダとして開き、先ほど作成した環境のインタプリタを選択したうえで、そのインタプリタをノートブックのカーネルに指定する。

ノートブックが 1 つのプロジェクトへと成長した段階でも、VS Code なら定義への移動、テストの実行、コード整形、Git の差分確認をこなせる。テスト可能なロジックは `.py` モジュールに置き、ノートブックは解説と実験記録のために使うとよい。また、このリポジトリには本書の執筆ワークフロー（フレームワーク表示の切り替え、スライドのプレビュー、編集内容のソースへの同期）を支援する VS Code 拡張機能も含まれており、詳細は 39.8 節で解説する。

39.1.3.3 デバッグと実行時間計測

例外が発生しても終了するのはそのセルだけであり、カーネルは終了せずすべての変数とともにトレースバックを保持している。したがって、次のセルで `%debug` を実行すれば、例外が送出されたフレームでデバッガを開ける。カーネルが対応していればノートブックのデバッガを使い、あるいは失敗する小さな呼び出しをモジュールに切り出してターミナルでデバッグするとよい。「failed」と表示するだけの大雑把な `try/except` ブロックは避けるべきである。原因の診断に必要な例外の型と発生箇所が失われてしまうからである。手軽な時間計測には、IPython のマジックコマンドが標準で用意されている。

```
values = np.arange(100_000, dtype=np.float64)
%timeit values @ values
```

```
9.77 µs ± 113 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

アクセラレータの実行時間を計測する際には 1 つ注意点がある。GPU の演算は非同期（asynchronous）であるため、ホスト側のタイマは処理の完了ではなく発行（ディスパッチ）だけを測定してしまうことがある。明示的に同期する（またはフレームワークのベンチマーク用ユーティリティを使う）こと、コンパイルされるカーネルを事前にウォームアップすること、そして測定値とあわせて形状・数値精度・デバイス・ライブラリのバージョンを記録することが不可欠である。

39.1.4 リモートマシン

エディタ、ノートブックサーバー、カーネル、およびアクセラレータは、同じマシン上にある必要はない。GPU のあるマシン側で JupyterLab を動かし、手元のノート PC から編集する構成はよく使われる。

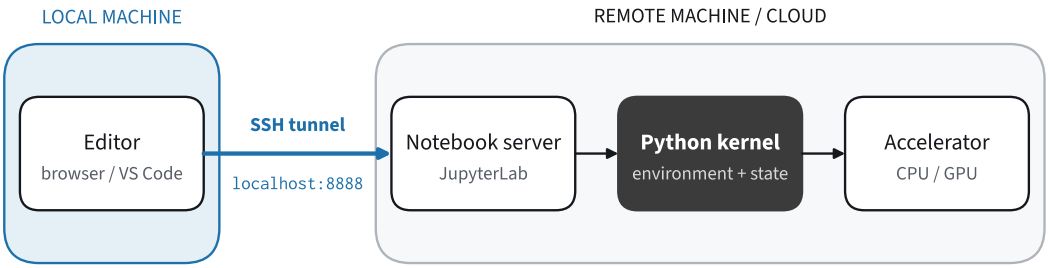


Figure 39.2: ローカルのエディタは SSH トンネル経由でリモートの Jupyter サーバーとカーネルに接続し、アクセラレータはリモートカーネル側に接続されたままとなる。

リモートサーバーをループバックアドレスにバインドし、SSH でポートフォワーディングする。

```
# Remote machine
uv run --no-sync jupyter lab --no-browser --ip 127.0.0.1 --port 8888
```

```
# Local machine
ssh -N -L 8888:127.0.0.1:8888 myserver
```

続いて、トークン付きの `http://127.0.0.1:8888/...` の URL をローカルで開く。認証なしの Jupyter サーバーを決してすべてのネットワークインターフェースにバインドしてはならない。そのポートに到達できる誰もが、起動したユーザー自身の権限でコードを実行できてしまうからである。SSH トンネルが暗号化とアクセス制御を担い、Jupyter のトークンは有効な第 2 の防御境界となる。

VS Code の Remote SSH はサーバー側に小さなコンポーネントをインストールするため、エディタ UI はローカルで動作しつつ、ターミナル・拡張機能・ファイル・カーネルはすべてリモートマシン上に置かれる。ローカルのカーネルからはリモートの GPU を使えないため、Python 環境を選択する前にステータスバーに **SSH** と表示されていることを確認する。いずれの構成でも、リモート計算環境は使い捨てのものとして扱うべきである。再起動やプリエンプション（強制終了）、削除が起きる前に、コードをコミットし、チェックポイントを永続ストレージへコピーしておく必要がある（39.3 節）。

ノートブックを共有またはコミットする前には、再現性（reproducibility）とプライバシーの最終確認を実施する。カーネルを再起動してすべてのセルを実行すること、セットアップが個人固有のパスや記録されていないダウンロードに依存していないか確かめること、出力からシークレットや非公開データを除去すること、説明や検証に役立つ出力だけを残して冗長な進捗ログを削除すること、そしてあらゆる測定値の解釈に必要な環境とハードウェアを記録しておくことである。

39.1.5 まとめ

ノートブックは文章・コード・実行結果を織り交ぜつつ、計算コストの高い状態を実験間でメモリに保持するため、受動的に読むだけでなく自ら手を動かして編集することで真価を発揮する。保存されたドキュメントと稼働中のカーネルは異なる順序で状態を記録するため、両者が一致しているかを確かめるには、カーネルを再起動して先頭からすべてのセルを実行する検証が欠かせない。ローカルでの実行を再現可能に保つには、本書の `uv` ロックファイルから同期した環境内で JupyterLab を起動し、`sys.executable` によって有効なカーネルを確認する。JupyterLab はノートブック・ターミナル・カーネル管理を 1 つのブラウザ上のワークスペースに統合し、VS Code はプロジェクトの成長に伴ってモジュール間の移動・テスト・差分確認・リモート開発の機能を提供する。リモートのアクセラレータ上で作業する際は、認証なしのノートブックサーバーをネットワークに公開するのではなく、SSH トンネルまたは VS Code Remote SSH を介して接続する。

39.1.6 演習問題

1. [code] **順序外の実行**。新しいノートブックで Figure 39.1 の失敗を再現せよ。すなわち、`data` を定義するセルを実行し、その上に `data` を使用するセルを挿入して、新しいセルを実行せよ。その後、カーネルを再起動してすべてのセルを実行すると `NameError` で停止する。
   1. エラーを送出する代わりに値を変化させるような順序外実行のセルの組を作成せよ（たとえば、下の方のセルで学習率を再設定したあと、上の訓練セルを再実行する例など）。このノートブックでカーネルを再起動してすべてのセルを実行すると、どのような結果になるか。2 つの失敗のうち、より危険なのはどちらか。また、それはなぜか。
   2. 保存されたノートブックで、実行カウンタ $1, 2, \ldots, n$ が n 個のコードセルにドキュメント順で表示されているとする。このことから、出力の生成過程について何が保証されるか。Pimentel et al. (2019) は、隠れた状態を 3 種類（再実行されたセル、実行後に編集されたセル、削除されたセル）に分類している。このようなノートブックにもなお含まれうるのは、このうちどれか。カーネルを再起動するとカウンタは 1 にリセットされるため、再起動の可能性も考慮せよ。
   3. 別のノートブックでは、カウンタ $3, 8, 9$ が 3 つのコードセルに表示されているとする。番号の飛びから、ドキュメントにはもはや残っていない実行について何が読み取れるか。また、この飛び番号は出力が隠れた状態に依存していることの証明になるか。
2. **クリーン実行の限界**。カーネルの再起動と全セル実行は、新しいカーネル上でドキュメントを再実行するが、実行先は同じマシンのままである。作成者のマシン上でこのテストに合格する次の 5 つのノートブックを考えよう。
   (i) あるセルはファイルが存在すれば `features.npy` を読み込み、存在しなければそれを計算するが、計算側の分岐にはバグがあり、`features.npy` はリポジトリに含まれていない。
   (ii) すでに削除されたセルからパッケージがインストールされていた。
   (iii) あるセルが `/home/alice/data/train.csv` を読み込む。
   (iv) あるセルが、JupyterLab を起動したシェルで設定されていた環境変数を読み込む。
   (v) シード未設定の乱数生成器を使ってセルが計算した正解率を、本文の段落が引用している。
   1. それぞれのノートブックについて、それをダウンロードした読者が経験する事象を説明せよ。
   2. 代わりに、別の新しいマシン上で、リポジトリを新規にクローンし、ロックファイルから構築した新しい環境のなかでノートブックを先頭から末尾まで実行するとする。この検証によって 5 つの失敗のうちどれが検出されるか。残る失敗については、それを検出する方法を挙げよ。
3. [code] **定義前参照の検出器**。●カーネルの再起動と全セル実行は、ノートブックを実際に実

行することによってのみ順序外の依存関係を見つけ出す。これに対し静的チェックは、実行せずにドキュメントを読み取って検査する。

1. `.ipynb` ファイルを読み込み、あるコードセルが参照している名前のうち、それより前のどのセルでも組み込み（ビルトイン）でも束縛されておらず、かつ後続のセルで束縛されている名前をすべて報告する関数を作成せよ。`%timeit` などのマジックコマンドを `IPython.core.inputtransformer2.TransformerManager` で Python コードに変換してから、セルを `ast` で構文解析すること。作成した関数を演習問題 1 のノートブックでテストせよ。
2. 関数の本体は、関数が定義されたときではなく呼び出されたときに実行される。この理由により検出器が誤検知（フォルスアラーム）を起こすノートブックと、実際の失敗を見逃すノートブックをそれぞれ 1 つずつ構成せよ。さらに、その誤検知を修正せよ。
3. 次の 2 つの警告を追加せよ。1 つは前のセルが参照したあとに後続のセルで再束縛される名前に対する警告、もう 1 つは同じ名前を参照して再束縛するセル（`pixels = pixels / 255` など）に対する警告である。それぞれの警告は、演習問題 1a の失敗と、インプレースなセルの反復実行による失敗のどちらを対象としているか。また、いずれかの警告だけで無害なケースと有害なケースを見分けることはできるか。

4. [code] **ワーカプロセス。**●Python の `multiprocessing` は、3 つの方式のいずれかでワーカを起動する。`fork` は親プロセスをコピーし、`spawn` は各ワーカを新しいインタプリタで起動し、`forkserver` はそれ自体が新規に起動されたサーバープロセスから各ワーカを fork する。どの方式でも、プールは呼び出す関数を含む各タスクを `pickle` でワーカへ送信する。ワーカプロセスを用いるデータローダも同じ仕組みに依存している。
   1. ノートブックのセルで関数 `square(i)`（戻り値は `i * i`）を定義せよ。別のセルで、システムが提供するすべての方式について `p.map(square, range(4))` を `with multiprocessing.get_context(method).Pool(2) as p:` の内側で実行せよ。セルの処理が戻らない場合は、カーネルに割り込みをかけよ。各方式で何が起き、エラーメッセージはどこに表示されるか。
   2. 関数に対して `pickle` が何を保存するか、およびノートブックがどのモジュール内に関数を定義するかという観点から、この挙動を説明せよ。
   3. 本書の環境で使用している Python 3.12 は、Linux では `fork`、macOS と Windows では `spawn` でワーカを起動する。Python 3.14 では Linux のデフォルトが `forkserver` に変更された。すべての起動方式でプールが動作するようノートブックを再構成せよ。
5. [code] **非同期アクセラレータの時間計測。**●GPU 上での 2 つの 8192×8192 行列の 16 ビット形式による積を考えよう。GPU が対応していれば bf16 を使い、bf16 演算器を持たない Colab や Kaggle の無料枠の T4 および P100（セットアップ手順は 39.2 節で解説する）では fp16 を使う。手元のマシンに GPU がない場合は、これらの無料 GPU のいずれかを使用せよ。
   1. 使用する GPU のデータシートに記載された該当形式の密行列スループット（RTX 4090 の場合は Table 13.2 の該当項目）に基づき、この積に必要な時間を計算せよ。
   2. `time.perf_counter()` を用いて 1 回の呼び出しの実行時間を、タイマ停止前のデバイス同期ありの場合となしの場合でそれぞれ計測せよ。いずれの計測でも、タイマ開始前に同期し、最初の 1 回目の呼び出しは除外すること。(a) と整合するのはどちらの計測値か。もう一方の計測値はどれほどのスループットを意味することになるか。
   3. 最初の呼び出し単独の実行時間を計測し、2 回目以降の呼び出しと異なる理由を説明せよ。
   4. 同期なしの積を、多数回連続で実行する `%timeit` で計測せよ。その結果は (b) の 2 つの計測値と比べてどの位置に収まるか。また、それはなぜか。
6. **共有 GPU サーバー。**●1 台のリモートサーバーを利用する複数のユーザーが、それぞれ `uv run --no-sync jupyter lab --no-browser --ip 127.0.0.1 --port 8888` で JupyterLab を起動し、各自のノート PC から `ssh -N -L 8888:127.0.0.1:8888 myserver` で接続するとする。

1. そのマシンの `127.0.0.1` で待ち受けるサーバーには誰が接続できるか。それらの利用者から自分のカーネルを守っているものは何か。実行中のサーバーがディスク上のどこにトークンを記録しているか、またどのユーザーがそのファイルを読み取れるかを調べよ。
2. 自分のサーバーがポート 8888 を保持している状態で、2 人目のユーザーが同じコマンドを実行したとする。Jupyter はデフォルトで近傍の最大 50 個の別ポートを順に試す。2 人目のユーザーのトンネルはどこへつながり、ブラウザには何が表示されるか。また、2 人目のユーザーは自分のサーバーが待ち受けているポートをどうやって確認できるか。
3. トンネルを開く際、手元のノート PC 上で自分が使っているローカルの JupyterLab がすでにポート 8888 で待ち受けていたとする。デフォルトでは、`ssh` はローカルポートをバインドできなくても警告を出すだけで実行を継続する。このとき、ブラウザで `http://127.0.0.1:8888` を開くとどこに接続されるか。セル `#interactive-development-identity` は、ノート PC 上でもサーバー上でも同様にホームディレクトリを `~` と表示する。2 つのマシンを見分けるにはどの項目を確認すればよいか。
4. (b) と (c) のそれぞれのポート競合を、確実に気づけるエラーへと変える接続手順を作成せよ。

39.2 Colab と Kaggle

前節では、手元のマシン上に本書のノートブック環境を構築した。しかし、GPU を搭載していないノート PC、ソフトウェアのインストールが禁じられた学校のコンピュータ、あるいはタブレットなど、適切なマシンが手元にない読者も多い。そうした読者の役に立つのが Google Colab と Kaggle である。いずれもセッションの間だけ貸し出される仮想マシン上のノートブックをブラウザで開く仕組みになっており、Python と主要なフレームワークがあらかじめインストールされているほか、利用上限の範囲内で空きがあれば無料の GPU も使える。

借り物のマシンでは、ローカルマシンにはない問いが生じる。明日になっても残っているものは何か、という問いである。パッケージをインストールし、Fashion-MNIST をダウンロードして、分類器を 1 時間訓練したとしよう。プロバイダは接続時にマシンを作成し、一定時間のアイドル状態や利用上限への到達によってセッションが終了すると、そのマシンを削除する。翌朝になっても、ノートブック本体と永続ストレージ（durable storage）に保存したファイルは残っている。しかし、インストールしたパッケージ、データセット、およびローカルディスクに書き出したチェックポイントは消えている。

その対策は、ランタイムを使い捨て可能なものとして扱うことである。環境を再構築する手順を記録し、残すべきデータを保存するノートブックであれば、Colab でも Kaggle でもローカルカーネルでも同じように動作し、プロバイダがソフトウェアイメージを更新した後でも問題なく動く。以下では、本書の起動ボタンが 2 つのプロバイダへどのようにつながるか、両プロバイダにどのような違いがあるか、そして新しいマシンに変わってもそのまま動くセットアップセルをどう書くかを見ていく。

Figure 39.3 にこのライフサイクルを示す。ランタイムは、無操作状態が続いた後に停止したり、セッション上限に達したり、別のソフトウェアイメージで再起動したりすることがある。ランタイムの再構築に必要な手順がすべてノートブックに記録されていれば、セットアップセルを再実行するコストはわずかで済む。本書のノートブックはそのように書かれている。

39.2.1 本書のノートブックの起動

本書のコードを含むすべてのページには **Run notebook** コントロールがあり、**Colab**、**Kaggle**、**Download** の 3 つの操作を選べる。このコントロールは選択中のフレームワークタブに連動し、

Figure 39.3: ホスト型ランタイムは一時的なものであり、ノートブック、データ、チェックポイントは明示的に保存したときに限り永続化される。

そのタブに対応する PyTorch、TensorFlow、または JAX のノートブックを提供する。MXNet のようにホスト型ノートブックがないタブでは、**Notebook unavailable** と表示される。フレームワークを使わないコードのみのページでは、代わりに単一の NumPy ノートブックが提供される。これらのノートブックは公開 GitHub ブランチ上の固定パスに配置されており、ビルド元のソースリビジョンを記録している。`d2l` をインポートするノートブックの先頭には、自動生成されたセットアップセルが入っている。このセルは、プロバイダにプリインストールされたパッケージに互換性があればそれをそのまま使い、互換性がなければ本書で動作確認済みのバージョンをインストールした上で、そのリビジョンに対応する `d2l` の補助モジュールを GitHub からダウンロードする。

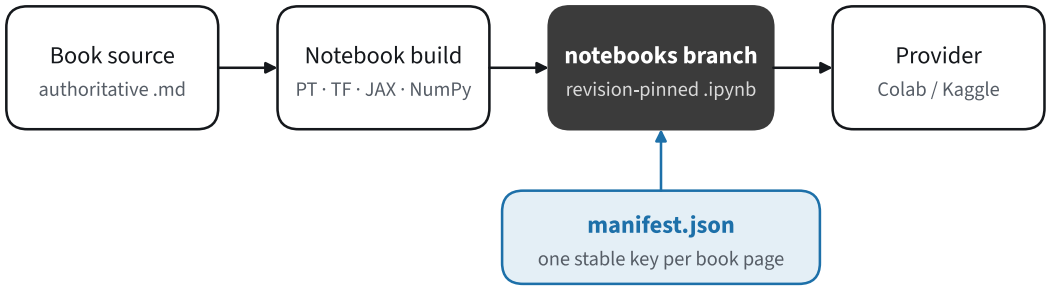


Figure 39.4: ソースページ、生成されたノートブックブランチ、およびプロバイダでの起動は独立した段階に分かれており、ページとノートブックは固定された対応関係で結ばれている。

2 つの起動操作では、作業内容の保存先が異なる。

- **Colab は GitHub へのポインタを開く。**URL は `colab.research.google.com/github/<org>/<repo>/blob/<` という形式になっており、Colab は開くたびに GitHub からノートブックを取得する。常に最新の公開版が表示されるが、編集内容は書き戻されない。変更を残すには、**File →Save a copy in Drive** を使うか、`.ipynb` をダウンロードする。
- **Kaggle はコピーをインポートする。**本書はノートブックの raw URL を Kaggle のインポータ（`kaggle.com/kernels/welcome?src=...`）に渡し、これによって自身の Kaggle アカウント内に新しい非公開ノートブックが作成されるため、利用にはアカウントが必要である。このコピーは本書から独立しており、本書が更新されても自動では更新されず、Kaggle 上でバージョンを保存することで変更内容を残せる。

いずれの場合も、起動リンクは公開パス上の閲覧可能なノートブックを参照しているため、実行前に中身を確認できる。インターネット上のどのような「Open in Colab」バッジに対しても同

様に、自分のアカウント権限で他人のコードを実行する前に、参照元のリポジトリを確認し、セットアップセルを読んでおく必要がある。

39.2.2 Colab

39.2.2.1 無料枠のリソース

Colab の無料枠では GPU と TPU が提供されるが、公式の FAQ ではハードウェアも上限も保証されていない。2026 年 11 月時点の記載によれば、利用可能なアクセラレータの種類は時期によって変わり、利用上限も変動する。ノートブックの実行時間は空き状況や自身の利用状況に応じて最大 12 時間であり、アイドル状態のランタイムはタイムアウトする。無料枠のユーザーには 16 GB のメモリを備えた T4 が割り当てられることが多いが、これもサービスとして保証されているわけではない。

メニューに表示されるのは要求した構成であって、実際に割り当てられた構成ではないため、ランタイムはコード内で確認する。

```
import os
import sys
from pathlib import Path

resource = {
    "cpu_count_visible": os.cpu_count(),
    "colab": "google.colab" in sys.modules,
    "kaggle": Path("/kaggle/working").exists(),
}
resource
```

```
{'cpu_count_visible': 64, 'colab': False, 'kaggle': False}
```

このセルは、起動時に読み込まれる `google.colab` モジュールによって Colab を判別し、作業ディレクトリによって Kaggle を判別する。アクセラレータについては、PyTorch では `torch.cuda.is_available()` と `torch.cuda.get_device_properties(0)`、TensorFlow では `tf.config.list_physical_devices("GPU")`、JAX では `jax.devices()` で確認できる。ただし、アクセラレータが検出されたとしても、ワークロードがそのメモリに収まる保証はない。

無料枠の上位として、Colab は Pay As You Go および Pro、Pro+ サブスクリプションを通じて**コンピューティングユニット**（compute unit）を販売している。残高を保有していればアクセラレータを利用しやすくなるが、特定のハードウェアが保証されるわけではない。単価や消費レートは変動するため、コストを見積もる前に購入ページを確認する必要がある。

39.2.2.2 ランタイムバージョンとコマンドライン

Colab はソフトウェアイメージを定期的に更新するため、先月動いていたノートブックが動かなくなることがある。2025 年 9 月以降、ノートブックでは **Runtime** →**Change runtime type** から日付付きの**ランタイムバージョン**をピン留めできるようになった。2026 年 11 月時点では、過去のバージョンはリリース後 1 年間利用可能であり、接続には 2〜3 分かかる。Google は、大半の作業では最新のランタイム上でライブラリのバージョンをピン留めする方法を推奨しており、講義資料のように多くの人と共有するノートブックではランタイム自体をピン留めすることを推奨している。

2026 年 6 月に Linux および macOS 向けにリリースされた Colab CLI を使えば、ターミナルやコーディングエージェントから同じランタイムを操作できる。`colab new` でランタイムを確保し、

`colab exec` でその上のスクリプトやノートブックを実行し、`colab stop` で解放する。

39.2.2.3 作業内容の保存

Colab は作業内容を Google Drive に保存する。**Save a copy in Drive** でノートブック本体が保存され、データセットやチェックポイントをセッションをまたいで保持するには Drive のマウント（`drive.mount`）を使うのが一般的である。GitHub から開いたノートブックが自動的に書き戻されることは決してない。また Colab は、手元のマシン上の Jupyter サーバーをブラウザページから操作する**ローカルランタイム**にも接続できる。ただし、これによってセキュリティ境界が移動する。ノートブックはローカルの Jupyter プロセスの権限で実行されるようになり、Drive のマウントも機能しなくなる。信頼できるノートブックだけを接続すべきである。

39.2.3 Kaggle

39.2.3.1 入力、出力、バージョン

Kaggle はデータセットとコンペティションを中心に設計されており、そのノートブックもこれに沿った仕組みになっている。Kaggle ノートブックにおけるファイルの扱い方は、次の 3 種類のディレクトリによって決まる。

- `/kaggle/input/` には、紐付けられたデータセット、モデル、および他のノートブックの出力が格納され、バージョン管理された上で**読み取り専用**としてマウントされる。紐付けられたデータセットはノートブック内でダウンロードする必要がない。
- `/kaggle/working/` は書き込み可能な出力ディレクトリであり、2026 年 11 月時点で 20 GB の容量がある。その中身は、ノートブックの**保存済みバージョン**に対してのみ永続化される。
- それ以外の場所はすべて、セッション終了とともに消えるスクラッチ領域である。

第 2 の大きな特徴はバージョン管理である。**Save & Run All** は、新規セッションでノートブック全体を上から下まで実行し、その入力、コード、出力、およびログをまとめた不変（immutable）なバージョンとして結果を保存する。これは、カーネルを再起動してすべてのセルを実行する習慣（39.1 節）をプラットフォーム自体に組み込んだものである。

39.2.3.2 クォータとハードウェア

2026 年 11 月時点では、各アカウントには P100 1 基または T4 2 基の GPU で週あたり 30 時間（需要に余裕があるときはそれ以上になることもある）の GPU クォータが付与され、これとは別に週あたりの TPU クォータも付与される。1 回のセッションの実行時間は、CPU または GPU で最大 12 時間、TPU で最大 9 時間である。ノートブックのインターフェースにアカウントの現在の値が表示されるため、実行を計画する前に確認しておくとよい。また Kaggle では、アクセラレータやインターネット接続を有効化する前に電話番号認証を求められる場合がある。

インターネット接続はノートブックごとの設定項目であり、一部のコンペティションでは無効化されるため、行儀のよいノートブックでは実行途中にデータをダウンロードするのではなく入力としてデータを紐付ける。Kaggle CLI を使えば、ターミナルからノートブックやデータセットのプッシュ、実行、プルを行える。そのトークンは必ずプロバイダのシークレットストアに保管し、決してセル内に書いてはならない。

39.2.4 ポータブルなノートブック

39.2.4.1 プロバイダの選択

どちらのプロバイダでも本書のノートブックを実行できる。両者の違いは、ファイルや結果がどこに保存されるか、そして利用上限がどの程度予測可能かという点にあり、その比較を Table 39.1 にまとめる。

Table 39.1: ホスト型ノートブックのトレードオフ（2026 年 11 月時点）

要件	Colab	Kaggle
GitHub ノートブックを直接開く	可（開くたびに取得）	インポートにより自身のコピーを作成
無料 GPU	通常は T4（上限は非公開）	P100 または 2×T4（週 30 時間と公開）
予測可能なクォータ	なし	あり
ファイルの永続化	Google Drive	バージョン管理されたデータセットと出力
再現可能な実行	手動での再起動と全セル実行、ランタイムバージョンのピン留め	Save & Run All によるバージョン保存
コンペティション連携	外部	ネイティブ対応
有料アップグレード	コンピューティングユニット（Pay As You Go、Pro、Pro+）	公開クォータの利用には不要

本書の 1 つの節を手早く動かすには、GitHub から直接開ける Colab が最短の選択肢である。一方、共有や再実行を前提としたデータセット中心の実験には、入力と出力がバージョン管理される Kaggle の方が適している。

39.2.4.2 セットアップセル

ランタイムは使い捨て可能であるため、ノートブック自身が環境構築の手順を備えていなければならない。優れたセットアップセルは短く、べき等（idempotent）かつ明示的である。すなわち、バージョンをピン留めし、不足しているものだけをインストールし、シークレットを一切含まない。

```
import importlib.util
import platform

required = ["numpy", "matplotlib"]
missing = [name for name in required
           if importlib.util.find_spec(name) is None]
{
    "python": platform.python_version(),
    "missing": missing,
    "reconstructible": not missing,
}
```

```
{'python': '3.12.3', 'missing': [], 'reconstructible': True}
```

ノートブックの先頭で無条件の `pip install --upgrade ...` を実行するのは避けるべきである。これを実行すると、検証済みのプロバイダ環境が壊れ、起動のたびに時間がかかり、昨日動いていたノートブックで今日になると異なるパッケージが解決される原因になる。代わりに、不足して

いる特定のパッケージだけをピン留めしたバージョンでインストールする。また、ノートブックを一切変更していなくてもプロバイダ側のイメージが変わることがあるため、簡潔な環境フィンガープリントを出力しておけば、結果の解釈やバグ報告に役立つ。

```
import json
import numpy as np

fingerprint = {
    "python": platform.python_version(),
    "numpy": np.__version__,
    "machine": platform.machine(),
}
print(json.dumps(fingerprint, indent=2))
```

```
{
  "python": "3.12.3",
  "numpy": "2.4.4",
  "machine": "x86_64"
}
```

39.2.4.3 パスとシークレット

ポータブルなノートブックでは、プロバイダ固有のパスのハードコーディングも避ける。小さなアダプタで作業ディレクトリを選択し、残りのファイル操作は `pathlib` に任せる。

```
from pathlib import Path

if Path("/kaggle/working").exists():
    work = Path("/kaggle/working")
elif Path("/content").exists():
    work = Path("/content")
else:
    work = Path.cwd()
work
```

```
PosixPath('/home/smola/d2l-neu/_notebooks/pytorch/chapter_appendix-tools-for-deep-learning')
```

どちらのプロバイダも API トークン用のシークレットマネージャを提供しており、ノートブックはそこからトークンを読み込むべきである。トークンを表示したり、セル出力に残したり、保存するコピーにコミットしたりしてはならない。公開ノートブックは実行可能な出版物であり、そのセル出力も公開内容の一部となるからである。

こうした習慣によって、ノートブックは実行先のマシンに依存しなくなる。ただし、それでセッション時間が延びたりアクセラレータが保証されたりするわけではない。長時間の実行や保証されたハードウェアが必要な場合は、39.3 節で扱うように、自分で管理できるマシンをレンタルする。

39.2.5 まとめ

Colab と Kaggle は、上限付きで GPU や TPU を利用できる無料の一時的なノートブックランタイムを貸し出しているが、ハードウェアの割り当ては保証されないため、必ずコード内で確認する必要がある。本書の 2 つの起動ボタンは作業内容の保存方法が異なる。Colab は GitHub へのライブポインタを開くため Drive にコピーしない限り編集内容は保存されないのに対し、Kaggle

は独立したスナップショットを自身のアカウントにインポートする。Colab の無料枠の上限は需要に応じて変動し、優先アクセスのための有料のコンピューティングユニットやコンテナイメージを固定するランタイムバージョンのピン留めが用意されている一方、Kaggle は週あたりのクォータを公開しており、Save & Run All の実行ごとに不変なバージョンを記録する。どのホスト型マシンもアイドルタイムアウトやセッション上限で消滅し得るため、べき等なセットアップセル、永続ストレージへの明示的なチェックポイント保存、およびプロバイダのシークレットマネージャによる認証情報の管理を行うことで、プロバイダや時間の経過に左右されないポータブルなノートブックを実現できる。

39.2.6 演習問題

1. **セットアップセルの監査**。「Open in Colab」バッジから開いたノートブックの先頭に、4 行のセットアップセルがある。このセルは順に `!pip install git+https://github.com/someone/helpers`、`!curl -fsSL https://example.org/setup.sh | bash`、`from google.colab import drive`、そして最後に `drive.mount("/content/drive")` を実行する。
   1. 各行について、どのようなコードを実行するか、あるいは自身のどのデータをノートブックから読み取り可能にするかを述べよ。そのコードのうち、セルを実行する前に中身を確認できるものはどれか。
   2. 最初の 2 行を書き換え、読んだ時点で実行されるコードが固定されるようにせよ。
   3. 次に、同じノートブックをローカルランタイムに接続し、ブラウザが手元のマシン上の Jupyter サーバーを操作するようにしたとする。影響が最も大きく変わるのはどの行か、またそれはなぜか。
2. [code] **インストール後の再起動**。ホスト型ノートブック、または新規作成したローカル仮想環境の Jupyter カーネルで次の実験を行え。まず `"tabulate" in sys.modules` が `False` であることを確認せよ。軽量なパッケージ `tabulate` のバージョン 0.8.10 をインストールし、インポートせよ。同じカーネルセッションの後続のセルで、再起動せずにバージョン 0.9.0 をインストールせよ。
   1. 2 回目のインストールの後に `tabulate.__version__` と `importlib.metadata.version("tabulate")` を出力せよ。2 つの値が異なるのはなぜか。実際に動作するコードを表しているのはどちらか。
   2. 本節で `missing` を計算したセルは `importlib.util.find_spec` に依拠している。この確認方法では、今観察した不一致を見逃してしまうのはなぜか。
   3. 関数 `ensure(dist, module, version)` を書け。この関数は、インストール済みディストリビューションのバージョンが異なる場合に限り `dist==version` をインストールし、インストール前に `module` がインポート済みであったためにカーネルの再起動が必要かどうかを返すものとする。セットアップセルからこれを 2 回続けて呼び出し、2 回目の呼び出しでは何もインストールされないことを確認せよ。
3. [code] **フィンガープリントの拡張**。`fingerprint` を構築するセルは、Python のバージョン、NumPy のバージョン、およびマシンのアーキテクチャを記録している。
   1. このセルで、本書が使用するインストール済みのすべての深層学習フレームワークのバージョンも記録したい。このセルでバージョンを調べるためにすべてのフレームワークをインポートするのが不適切なのはなぜか。
   2. これらのフレームワークのバージョン、`nvidia-smi` が報告する認識可能な各 NVIDIA GPU の名前とメモリ容量、および名前が `COLAB_` または `KAGGLE_` で始まるプロバイダ固有の環境変数を追加して、このセルを拡張せよ。セルは CPU のみのランタイムでも動作し、フレームワークのバージョンをパッケージメタデータから読み取る必要がある。フレームワークがすでにインポートされている場合は、その `__version__` も記録せよ（2 つの値が異なり得る理由は演習 2 に示した通りである）。いずれか 1 つのプロバイダ上でこのセルを実行せよ。記録された変数のうち、プロバイダがソフトウェアイメージを変更したことを明らかにするのはどれか。

   3. 最初のセルで `pip install --upgrade transformers` を実行するノートブックが、月曜日には動作していたのに、金曜日には `transformers` 内部のインポートエラーで失敗したとする。第 1 の仮想的なケースでは、金曜日のフィンガープリントにおけるプロバイダイメージと NumPy のバージョンは月曜日と同じであり、`transformers` が 4.56.1 から 4.57.0 に変わっていた。第 2 のケースでは、プロバイダイメージが変わり、NumPy が 2.0.2 から 2.3.1 に変わった一方、`transformers` は 4.56.1 のままであった。それぞれのケースで失敗の原因となった変更はどれか。また、それぞれのケースで再発を防ぐにはノートブックをどう修正すればよいか。
4. [code] **永続パスとスクラッチパス**。`work` を設定するセルは、Kaggle では `/kaggle/working`、Colab では `/content`、それ以外ではカレントディレクトリを選択する。
   1. 3 つのランタイムのうち、`work` に書き出したチェックポイントがセッション終了後も残るのはどれか、またどのような形で残るか。
   2. このセルを関数 `locations(root)` に置き換えよ。この関数は、読み取り専用の入力ディレクトリである `inputs`、セッション終了後も残るチェックポイント用ディレクトリである `durable`、およびセッションとともに消え得る領域である `scratch` の 3 つのエントリを持つ辞書を返す。引数 `root` はファイルシステムのルートを表す。ホスト型ノートブックでは `locations(Path("/"))` を呼び出し、関数内では `root / "kaggle" / "working"` を使って判定し、`/kaggle/working` を直接は参照しない。Colab における入力ディレクトリのように、そのランタイムに存在しない場所についてはすべて `None` を返せ。Colab では Google Drive が `/content/drive` にマウントされている場合に限り永続ディレクトリが存在するため、それ以外の場合に `None` を返すようにしておけば、チェックポイントの保存処理はセッション終了とともに消えるディレクトリへの保存を拒否できる。
   3. 手元のマシン上で、4 つの一時ディレクトリを `root` としてこの関数をテストせよ。すなわち、`kaggle/input` と `kaggle/working` を含むもの、`content` のみを含むもの、`content/drive` を含むもの、および空のディレクトリの 4 つである。
5. **無料クォータでの事前学習**。●BERT-base は約 1.1 億個のパラメータを持つ。Devlin et al. (2018) は 256 系列のバッチで 1,000,000 ステップにわたりこれを事前学習しており、全ステップの 90% では 128 トークンの系列を、残りでは 512 トークンの系列を用いている。NVIDIA の仕様では、T4 の FP16 ピーク性能は 65 TFLOP/s である。1 基の T4 上での訓練がこのピーク性能の 30% を維持すると仮定せよ。
   1. N 個のパラメータを持つモデルを D トークンで訓練するには、約 $C \approx 6ND$ 回の浮動小数点演算というコストがかかる。内訳は、パラメータおよびトークンあたり順伝播で 2 回、逆伝播で 4 回である（11.6 節）。この方法で BERT-base の訓練コストを見積もれ。T4 で何時間かかるか。また、それは 12 時間セッションで何回分、週 30 時間のクォータで何週間分に相当するか。
   2. BERT の事前学習におけるコストのうち、$6ND$ が見落としているものはどれで、過大評価しているものはどれか。両方の系列長におけるアテンションスコア（補正項は 11.6 節にある）、全位置の 15% のみを予測するマスク言語モデルヘッド、および埋め込みテーブルについて考察せよ。それぞれを補正すると見積もりは上がるか下がるか、またおおよそどの程度変わるか。
   3. Hoffmann et al. (2022) は、パラメータあたりの訓練トークン数を約 20 とする計算量最適な配分を見出した。この配分の下では、1 回の 12 時間 T4 セッションでどのモデルサイズとトークン数を訓練できるか。これに BERT-base のパラメータあたりのトークン数を比較せよ。
6. [code] **チェックポイント間隔**。●●アイドル切断やマシンの回収といったホスト型セッションの中断を、中断間の平均時間が M のポアソン過程としてモデル化せよ。訓練では τ 秒の計算ごとにチェックポイントを書き出し、その書き出しには δ 秒かかる。中断が起きた後、再接続して最新のチェックポイントを復元するのに R 秒かかり、そのチェックポイント以降の計算は失われる。また、復元中にも中断が起き得るものとする。

1. $\delta \ll \tau \ll M$ と仮定せよ。1 次近似において、実経過時間のうちチェックポイント保存と失われた計算に費やされる期待割合が最小になるのは次の間隔であることを示せ。

$$\tau^* = \sqrt{2\delta M}, \tag{39.1}$$

これは Young (1974) の間隔である。
2. τ 秒の計算とそれに続くチェックポイント保存からなる 1 区間を完了するまでの期待実経過時間が次式となることを示せ。

$$\mathbb{E}[T] = Me^{R/M}\left(e^{(\tau+\delta)/M} - 1\right). \tag{39.2}$$

$\mathbb{E}[T]/\tau$ を、$\delta/M \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$ のそれぞれについて数値的に最小化せよ。τ^* は厳密な最適値からどれだけずれるか。また、τ^* における期待実経過時間は最小値をどれだけ上回るか。これら 2 つのずれの大きさがこれほど異なる理由を説明せよ。
3. この過程のモンテカルロシミュレーションを書き、$\mathbb{E}[T]$ の式を δ/M と R/M のある 1 組の設定の下で検証せよ。
4. Kaggle のセッションはランダムな時刻ではなく、固定された 12 時間の上限で終了する。中断の時刻があらかじめわかっている場合、チェックポイントのスケジュールはどう変更すべきか。さらにランダムなアイドル切断も起こり得る場合、上の解析のどの部分が依然として成り立つか。

39.3 クラウドコンピューティング

大半の読者は、本書のノートブックをノート PC、デスクトップ GPU、あるいは Colab のようなホスト型サービス（39.2 節）上で実行しているだろう。しかし遅かれ早かれ、プロジェクトはこの 3 つのいずれにも収まらなくなる。前処理がホスト型セッションの制限時間を超えてしまったり、モデルが 80 GB のアクセラレータメモリを必要としたり、実験に複数の GPU が同時に必要になったりするためである。これに対する通常の解決策はマシンを時間単位で借りることだが、そこでは 2 つの問いが生じる。どのマシンを借りるべきであり、完了した結果を得るのにいくらかかるのか。そして、いつ消えてもおかしくないマシン上でどのように作業すればよいのか。

料金は毎月変動するため、以下に示す数値は時点を明記した例であり、価格カタログではない。だが、価格が変わっても通用する考え方が 2 つある。マシンは時間あたりの料金ではなく完了した結果あたりのコストで選ぶこと、そしてレンタルしたインスタンスは使い捨てとして扱い、重要なものはすべて永続ストレージにコピーしてからインスタンスを削除することである。なお、本書自身のノートブックが必要とするハードウェアは控えめである。いずれも 24 GB の RTX 4090 カード上で実行されており、大半は 8 GB 未満しか使用しないため、24 GB のレンタル機があれば単一 GPU のノートブックはすべて動作する。複数のカードを必要とするのは、第 13 章のマルチ GPU に関する各節のみである。

39.3.1 レンタル市場

39.3.1.1 プロバイダの 3 つのティア

GPU のレンタルは 3 種類の事業者によって提供されており、それらの違いはハードウェアそのものよりも、それを取り巻く周辺サービスと運営主体にある（Figure 39.5）。

- AWS、Google Cloud、Microsoft Azure などの**ハイパースケーラ**は、ID 管理、仮想ネットワーク、マネージドストレージをはじめとするあらゆる周辺サービスを備えた総合クラウドの一部として GPU を販売している。オンデマンドの GPU 料金は最も高く、最大規模

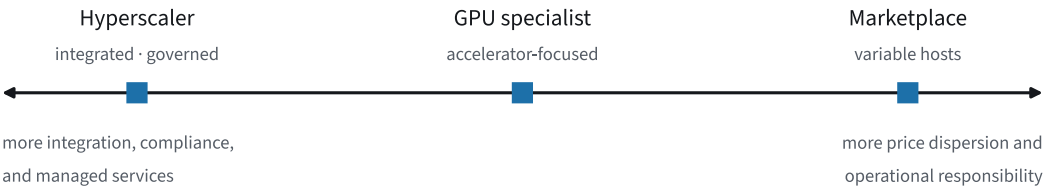


Figure 39.5: クラウドの選択肢は、マネージドな統合環境やガバナンスと引き換えに、価格のばらつきや運用上の責任を受け入れる関係にある。

のインスタンスではクォータの引き上げ申請や営業窓口への問い合わせが必要になることも多い。データがすでにそのクラウド上にある業務や、コンプライアンス要件の下にある業務、より大規模なマネージドシステムの一部として組み込まれる訓練に適している。また Google は本書の JAX コードを実行できる TPU も貸し出しており、2026 年 11 月時点で v5e チップ 1 基の料金は 1 時間あたり $1.20 である。

- Lambda、Nebius、Voltage Park、CoreWeave などの **GPU スペシャリスト**は、高速なインターコネクトと機械学習用イメージを備えたアクセラレータを貸し出しており、それ以外の周辺サービスはほとんど持たない。2026 年 11 月時点で、このティアにおける H100 の料金は GPU 時間あたり約 $2〜6 であり、Lambda、Nebius、Voltage Park では新規アカウントから営業担当を介さずに起動できる。ハイパースケーラの他のサービスを必要としない本格的な訓練の実行に適している。
- Vast.ai や RunPod などの**マーケットプレイス**は、大手クラウドが扱わないコンシューマ向け GPU も含め、多数の独立した運営者からマシンを集約している。時間あたりの料金は最も安く、2026 年 11 月時点で RTX 4090 が 1 時間あたり $0.30〜0.50、H100 が約 $1.80 から出品されている。品質は出品枠ごとにばらつきがあり、ホストの信頼性、ディスク速度、ネットワーク品質はホストによって異なるうえ、コードは見ず知らずの第三者のマシン上で動作する。講義の課題や公開データであれば問題ないが、機密性の高い用途には、RunPod の Secure Cloud のような審査済みのデータセンターティアでジョブを実行しない限り不向きである。

39.3.1.2 特定時点の価格スナップショット

料金はリージョン、空き状況、契約期間、ホストによって異なる。Table 39.2 にティア間の価格差の例を示すが、利用前には最新の見積もりを確認されたい。

Table 39.2: オンデマンドの GPU レンタル価格スナップショット（GPU 1 基あたり、2026 年 11 月）

プロバイダ（ティア）	低価格 GPU の選択肢	≈$/hr	1× H100 80 GB ≈$/hr
Vast.ai（マーケットプレイス）	RTX 4090 24 GB	0.30〜0.50	1.80〜2.40
RunPod（マーケットプレイス）	RTX 4090 24 GB	0.34〜0.89	1.99〜3.99
Lambda（スペシャリスト）	A100 40 GB	1.99	3.29〜4.29
Nebius（スペシャリスト）	RTX PRO 6000 96 GB	1.80	4.50
Voltage Park（スペシャリスト）	H100 80 GB（唯一の掲載価格）	1.99	1.99

プロバイダ（ティア）	低価格 GPU の選択肢	≈$/hr	1× H100 80 GB ≈$/hr
AWS（ハイパースケーラ）	L4 24 GB (g6.xlarge)	0.80	6.88 (p5)
Azure（ハイパースケーラ）	T4 16 GB (NC4as T4 v3)	0.53	6.98 (NC40ads H100 v5, 94 GB)
Google Cloud（ハイパースケーラ）	L4 24 GB (g2-standard-4)	0.71	≈11（8 GPU の A3）

RunPod の価格幅はコミュニティホストからデータセンターティアまでにわたる。Azure の単一 GPU 版 H100 インスタンスは 94 GB の H100 NVL を搭載している。Google Cloud では、オンデマンドの H100 は 8 GPU マシンとしてのみ販売されており、表の数値は GPU 1 基あたりに換算した料金である。

ここで 2 つの但し書きが重要になる。第 1 に、マルチ GPU ノードの料金はおおむね GPU 数に比例するが、インターコネクトはそうではない。NVLink を備えた 8×H100 マシンは PCIe 接続のカード 8 枚とは別物の道具であり、通信負荷の高い訓練（39.6 節）ではその差が顕著に現れる。第 2 に、新しい世代ほど仕事量あたりのコストが安いとは限らない。2026 年 11 月時点で B200 インスタンスは GPU 時間あたり約 $7〜14 かかるため、より大きなメモリ容量や FP8/FP4 のスループットをジョブが活かせるときに限り元が取れる。

39.3.1.3 完了した結果あたりのコスト

時間あたりの料金は総額のなかで最も目につきやすい項目にすぎず、実際の総額にはセットアップ時間、デバッグ中のアイドル時間、実行後も課金され続けるストレージ、データエグレス、そしてエンジニアの作業時間も含まれる。したがって、より高速で高価な GPU を使った方が、**完了した**実験あたりのコストはかえって下がることがある。以下のモデルでは、最も遅い GPU で 8 時間かかるジョブについて、マーケットプレイスの 3 種類の GPU のうちどれが最も安くジョブを完了できるかを計算する。自身の前提条件に書き換えて再実行してみてほしい。

```
import numpy as np

gpu = ["RTX 4090", "A100 80GB", "H100"]
gpu_per_hour = np.array([0.40, 1.20, 2.50])   # marketplace, November 2026
relative_speed = np.array([1.0, 1.6, 3.0])    # measure for your workload!
setup_hours = np.array([0.5, 0.5, 0.5])
storage_and_egress = np.array([2.0, 2.0, 2.0])
engineer_per_hour = 60.0

wall_hours = 8.0 / relative_speed
invoice = (wall_hours + setup_hours) * gpu_per_hour + storage_and_egress
complete = invoice + setup_hours * engineer_per_hour
for row in zip(gpu, np.round(wall_hours, 1), np.round(invoice, 2),
               np.round(complete, 2)):
    print(row)
```

```
('RTX 4090', np.float64(8.0), np.float64(5.4), np.float64(35.4))
('A100 80GB', np.float64(5.0), np.float64(8.6), np.float64(38.6))
('H100', np.float64(2.7), np.float64(9.92), np.float64(39.92))
```

クラウドの請求額は $5.40、$8.60、$9.92 となり、請求額だけを見れば安価なカードが有利である。ここにエンジニアの作業時間 0.5 時間分を加えると総額は $35.40、$38.60、$39.92 となり、差はずっと縮まるものの順位は変わらない。しかし、遅いカードで高速なカードより 30 分のデバ

ッグ作業が 1 回多く必要になれば、その総額は $65 を超えて順位が逆転するため、試行錯誤の回数も時間単価と同様にコストモデルへ組み込むべきである。ワークロードの規模はさらに重要である。7B モデルの LoRA ファインチューニングであれば 1 基のコンシューマ向け GPU で数時間で済むかもしれないが、小規模な言語モデルの事前学習（pretraining）には数千 GPU 時間を要し得る。時間単価のわずかな差を比較する前に、まずワークロードを見積もる必要がある。

見落としやすいコストが 2 つある。

- **スポットおよび中断可能な容量**は半額以下に割引されることが多く、チェックポイント保存と再開を確実に行えるジョブ（39.6 節）に向いており、対話型セッションやチェックポイントなしの実行には向かない。AWS は 2 分前に終了を通知し、Google Cloud はデフォルトで最大 30 秒のシャットダウン猶予を与えるが、Vast.ai の中断可能インスタンスはより高い入札が入った瞬間にプロセスを強制終了して停止する。復旧手順は長時間の実行の最中ではなく、開始前にテストしておくべきである。
- **エグレス料金**。2026 年 11 月時点で、AWS と Azure はインターネットへのデータ送信に対し、月間最初の 100 GB を超えた分に GB あたり約 $0.09 を課金する。140 GB のチェックポイントを 1 か月間毎日ダウンロードすると約 $370 かかり、マーケットプレイスの RTX 4090 を 24 時間稼働させ続ける料金（$0.44 で 720 時間、約 $320）を上回る。データと計算機は同一リージョンに置き、大きなアーティファクトを頻繁に移動するワークフローでは、RunPod をはじめエグレス料金を課さないプロバイダを選ぶとよい。

39.3.2 レンタルマシンでの作業

39.3.2.1 インスタンスのセットアップと事前確認

起動したばかりのインスタンスは未知のマシンであり、最初の数分間の確認が長時間の実行を安心して任せられるかを左右する。まずはドライバ、CUDA、コンテナランタイムがプリインストールされたプロバイダ最新の深層学習用イメージから始めること。素の OS イメージから始めると、後から再現しなければならない環境構築の手間が増える。接続には、どのプロバイダでも同じ手順で使える 39.1 節の SSH トンネル、または VS Code Remote SSH を用いる。SSH の切断で訓練が強制終了しないよう長時間ジョブは `tmux` 上で実行し、最初にマシンの状態を確認する。

```
nvidia-smi                # driver ok? expected GPU? memory free?
df -h                     # scratch disk has room for data + checkpoints?
python -c "import torch; print(torch.cuda.get_device_name(0))"
```

モデルに 1 バッチを通し、チェックポイントを 1 回書き出して読み込み直す短いスモークテストを実行すれば、プロバイダのヘルス指標では見逃される問題を検出できる。壊れたドライバ、空き容量不足のディスク、読み取り専用のマウント、そしてストリーミングが遅すぎて GPU を稼働させ続けられないデータセットなどである。

39.3.2.2 使い捨てのインスタンスと永続的な結果

レンタルしたインスタンスは、プリエンプトされたり、クラッシュしたり、誤って削除されたりする可能性があり、その上の作業内容はこれら 3 つのいずれが起きても失われないようにしなければならない。

Figure 39.6 のワークフローは、マシンの寿命と作業の寿命を切り離している。コードは Git に保存し、インスタンス上へクローンする。データとチェックポイントは、ジョブ側で制御するスケジュールに従って永続オブジェクトストレージ（S3、GCS、またはプロバイダのボリューム製品）にコピーする。こうしておけば、インスタンスを失った際の損失は高々直近のチェックポイント以降の作業分にとどまり、代替マシンがストレージから処理を再開できる。

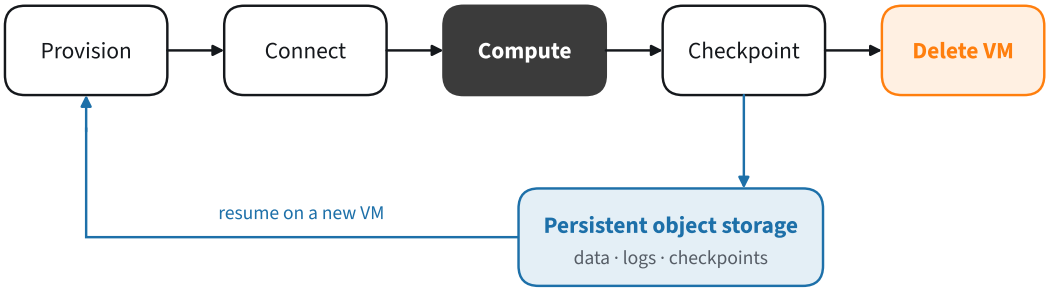


Figure 39.6: 計算機をプロビジョニングして安全に接続し、永続ストレージへチェックポイントを保存したうえで VM を削除する。代替マシンではそのストレージから処理を再開する。

実験が終わったら、インスタンス、ディスクやスナップショット、予約済み IP アドレス、不要になったバケットなど、課金対象となるものをすべて削除する。初日に請求アラートを設定しておくこと。アイドル状態の GPU も稼働中の GPU と同じ料金で課金される。また、アラートの通知は実際の利用より遅れて届くため、認証情報が漏洩した際の損失に上限を設けるのはアカウントの GPU クォータである。

以上で、完了した結果あたりのコストに基づいてレンタルマシンを選び、マシンに障害が起きても作業内容を安全に保てるようになった。1 年の大半の日数で稼働し続ける GPU は、やがて借りるよりも購入する方が安くなる。39.4 節では、どのハードウェアを購入すべきかを検討する。

39.3.3 まとめ

本書のノートブックの大半は必要な GPU メモリが 8 GB 未満であり、単一 GPU のノートブックはすべて 24 GB のカードに収まるため、多額のレンタル費用が発生するのは主に言語モデルの規模からである。クラウド GPU 市場はハイパースケーラ、GPU スペシャリスト、マーケットプレイスにまたがり、同一の半導体でも数倍の価格差があるとともに、マネージドな統合機能、信頼性、運用上の信頼水準も大きく異なる。マシンを比較する際は、時間あたりの表示価格ではなく、セットアップ、デバッグの試行回数、永続ストレージ、データエグレス、およびエンジニアの作業時間を織り込んだ完了結果あたりの総コストで常に判断する。スポット容量はチェックポイント保存と再起動が可能なジョブの計算料金を削減できる一方、対話的な作業やチェックポイントなしの実行には不向きであり、またクラウドをまたぐエグレス料金は大容量チェックポイントの反復転送に対して急速に負担となる。レンタルしたインスタンスはすべて使い捨てとして扱い、構築済みイメージから起動し、`tmux` を用いて SSH トンネル経由で接続し、チェックポイントを永続ストレージへ順次保存して、実行が終わり次第すべての課金対象リソースを削除する。

39.3.4 演習問題

1. [code] **ピークメモリとバッチサイズ。**本書のノートブックの大半は GPU メモリの使用量が 8 GB 未満にとどまるため、Table 39.2 にある 24 GB カードではより大きなバッチをとる余裕がある。19.2 節（英語版） のエンコーダのファインチューニングを GPU 上（たとえば 39.2 節のホスト型ノートブック）で、元のバッチサイズと半分のバッチサイズでそれぞれ 1 回ずつ実行せよ。両方の実行について、Figure 5.16 における GPU メモリの 3 つのビューそれぞれのピーク値を記録せよ。

    1. 3 つのビューのうち、実行がカードに収まるかどうかを決めるのはどれか。また、それはなぜか。

   2. そのビューのピーク値をバッチサイズの一次関数として当てはめ、24 GB カードおよび 80 GB カードに収まる最大のバッチサイズを予測せよ。切片はメモリのどの部分を表すか。
   3. 2 回の実行の間のバッチサイズ（24 GB カードが利用できる場合はその予測値のバッチサイズでもよい）で実行し、実測されたピーク値を線形予測と比較せよ。

2. **信頼境界**。あるファインチューニングのジョブが、インスタンス上に保存されたアクセスキーを用いてオブジェクトストレージから訓練データを読み込み、インスタンスのローカルディスクにチェックポイントを書き出すとする。

   1. データは公開データセットであり、ジョブはマーケットプレイスのホスト上で実行されるとする。ジョブの実行中、ホスト運営者はデータセット以外に何を読み取ったり書き換えたりできるか。データが公開されていても問題になるのはそのうちどれか。また、インスタンスのディスクを暗号化してもホスト運営者からそれらを守れないのはなぜか。
   2. データが病院の医療記録であり、ジョブをハイパースケーラへ移すとする。ジョブの実行中、依然としてデータやチェックポイントを読み取れる立場にあるのはどの主体か。また、技術的な統制以外に、記録をアップロードする前に整えておかなければならないものは何か。

3. [code] **コストモデルにおけるエンジニア工数**。`#cloud-instances-cost-model` では、`complete` の列は請求額に `setup_hours * engineer_per_hour` を加算している。

   1. `engineer_per_hour` の値がいくらであっても、`setup_hours` が 3 つの GPU すべてで同じである限り、この項によって最安となる GPU が変わらない理由を説明せよ。
   2. 結果を待ったり実行状況を確認したりするエンジニアが、`wall_hours` のうちの割合 f をそれに費やすとする。この時間に `engineer_per_hour` を掛けた額を `complete` に加え、H100 と RTX 4090 の総コストが等しくなる f の値を求めよ。この値は、RTX 4090 での 8 時間の実行中に何分間の対応時間が生じることに相当するか。
   3. エンジニア工数の項を両方とも除外せよ。請求額だけで H100 が RTX 4090 と並ぶには、いくらの `relative_speed` が必要か。
   4. Table 39.2 に挙げたマーケットプレイスの 1 つから 3 つの GPU の本日のオンデマンド価格を取得し、日付を記録したうえで損益分岐点となる割合 f を再計算せよ。その値は 2 倍を超えて変化するか。

4. [code] **スモークテスト用スクリプト**。長時間の実行に入る前にレンタルマシンが通過すべきスクリプトを作成せよ。このスクリプトは、任意の小さなモデルを `d2l.FashionMNIST` 上で訓練し、最初に失敗したチェック項目で明確なメッセージを出して終了するものとする。

   1. 次の 4 つのチェックを実装せよ。コマンドラインで指定されたアクセラレータが認識されること、チェックポイント用ディレクトリに指定バイト数以上の空き容量があること、アクセラレータ上で訓練ステップが 1 回実行できること、そして `d2l.save_checkpoint` で書き出して `d2l.load_checkpoint` で読み込み直したチェックポイントがすべてのパラメータとオプティマイザのテンソルを厳密に再現することである。
   2. 5 つ目のチェックとして、データイテレータが供給する毎秒バッチ数と、すでにデバイス上にあるバッチに対してモデルが維持できる毎秒訓練ステップ数とを比較する処理を追加せよ。なぜこれら 2 つの速度を別々に測定しなければならないのか。
   3. 手元のマシンで各障害を意図的に発生させよ。ディスクの実際の空きを超える容量を要求し、チェックポイント用ディレクトリを読み取り専用にし、バッチごとに `time.sleep` を入れてイテレータを遅延させよ。root はディレクトリの権限を無視するため、読み取り専用テストは非 root ユーザーで実行すること。このスクリプトがなければ、1 時間ごとにチェックポイントを書き出す 8 時間の実行において、それぞれの障害はいつ発覚していたか。また、実行を決して停止させない障害はどれか。

5. **NVLink か PCIe か。**●2 つのプロバイダが GPU 時間あたり同じ料金で 8 基の H100 を提供している。一方のノードは GPU 間を NVLink で接続し、もう一方は PCIe 5.0 のみで接続している。データ並列ジョブでパラメータ数 N の Transformer を BF16 勾配で訓練すると、通信時間には $S = 2N$ バイトが関与する（13.5 節）。各 GPU はステップあたり b トークンを処理し、これには $6Nb$ FLOPs を要する（11.6 節）。バス帯域幅 $1/\beta_{\text{link}}$ を片方向の公称速度（NVLink は 450 GB/s、PCIe は 64 GB/s）とし、両ノードとも H100 の密な BF16 演算性能 989 TFLOPS の 40% を実効維持するものとする。また、通信と計算はオーバーラップしないと仮定する。

   1. $N = 1.3 \times 10^9$ のとき、$b = 16{,}384$ および $b = 4{,}096$ における各ノードのステップ時間を計算せよ。トークンあたりのコストで NVLink ノードと並ぶには、PCIe ノードの価格をどの割合だけ引き下げる必要があるか。
   2. 通信時間と計算時間の比は、両方の時間が N に比例して増大するため、このパラメータ数に依存しない（13.2 節）。各ノードにおいて、通信時間と計算時間が等しくなる GPU・ステップあたりのトークン数を求めよ。
   3. 勾配累積（13.4.4 節）では、$N_{\text{accumulation}}$ マイクロバッチごとに 1 回だけ通信する。これにより上で求めた閾値はどう変化し、また訓練の何が変化するか。

6. **スポット容量での再実行。**●あるジョブが T 時間の計算を必要とする。スポット容量上では、中断は到着間隔の平均時間 M のポアソン過程に従って発生する。各中断の後、代替インスタンスの起動に R 時間かかり、ジョブはゼロから再開される。スポット容量の料金はオンデマンド価格の割合 q であり、再起動にかかる時間も課金対象となる。代替インスタンスの起動中には中断は発生しないものとし、完了までの実経過時間の期待値は次の式で与えられる。

$$\mathbb{E}[T_{\text{wall}}] = \left(e^{T/M} - 1\right)(M + R). \tag{39.3}$$

   問題 6b（39.2 節）では、復旧作業中にも中断が発生し得る場合に対応する式を導出している。そこでは因子 $M + R$ が $M e^{R/M}$ に置き換わる。

   1. $T = 8\,\text{h}$、$R = 0.25\,\text{h}$、$q = 0.5$ のとき、$M = 24$、8、4 h のそれぞれについて、期待スポットコストがオンデマンドコストに占める割合を計算せよ。
   2. $R \ll M$ かつ $q = 0.5$ とする。スポット容量の方が期待値において安くなるのは、T が M のある定数倍未満のとき、かつそのときに限ることを示し、その倍数を求めよ。
   3. 今度は、ジョブが計算時間 $\tau = \sqrt{2\delta M}$ 時間（Young (1974) による問題 6a の間隔）ごとにチェックポイントを書き出すとし、1 回の書き出しに δ 時間かかるとする。中断によって失われるのは直近のチェックポイント以降の作業のみである。$\mathbb{E}[T_{\text{wall}}]$ の式を長さ $\tau + \delta$ の 1 区間に適用し、スポット容量の方が期待値において安くなる最大の q を、$\delta = 2\,\text{min}$、$R = 0.25\,\text{h}$、および $M = 24$、8、4、2、1 h のそれぞれについて求めよ。

39.4 ハードウェア

大半の読者は、ノート PC、ホスト型ノートブック、レンタルしたカードなど、手元にある GPU で本書のノートブックを実行している。だが遅かれ早かれ、自前のマシンを購入すべきか、そしてどれを選ぶべきかという疑問に行き着く。たとえば、80 億パラメータの言語モデルを独自の文書でファインチューニングしたいとしよう。スペック表には何十もの数値が並んでいるが、最も高速に見えるカードが、そもそも目的のモデルを実行できるとは限らない。

こうした選択のほぼすべてを決めるのは、メモリ容量（memory capacity）、メモリ帯域幅（memory bandwidth）、演算スループット（compute throughput）の 3 つの量である。13.2 節

ではこれらがマシンの挙動をどのように規定するかを説明したが、本節では何を購入すべきかを問う。その手法とは、自身のワークロードを律速している量を見極め、その量がより大きい機材を購入することである。以下に示すマシンと価格は 2026 年 11 月時点のスナップショットであり今後変化するが、選ぶための手法自体は変わらない。

容量は、そもそも何が実行できるかを決める。訓練では、重み、その勾配、およびオプティマイザ状態をメモリ上に保持するため、モデルサイズの数倍のメモリを要する。これらは 39.6 節で見るように、Distributed Data Parallel（DDP）の下で複数の GPU に複製されるか、Fully Sharded Data Parallel（FSDP）および Zero Redundancy Optimizer（ZeRO）の下で複数の GPU にシャーディングされる。メモリに収まらないモデルは、どれほど高速なカードであっても実行できない。**帯域幅は、モデルがどれだけ速くテキストを生成するかを決める。**新しいトークンを 1 つ生成するたびに、そのトークンが使用するすべての重みに加え、キー・値（KV）キャッシュ、すなわちアテンションが過去の全トークンについて保持しているキーと値を読み出す必要がある（11.3 節、およびサービング規模については 39.7 節を参照）。したがって、次の関係が成り立つ。

$$\text{decode tokens/s} \lesssim \frac{\text{memory bandwidth}}{\text{bytes of active weights and KV cache}}. \tag{39.4}$$

8B モデルをファインチューニングし、その重みを 4 ビットに量子化（quantization）すると、短いコンテキストでは 1 トークンあたり約 4.5 GB を読み出すため、900 GB/s でデータを流し込めるカードが生成できるのは多くても毎秒約 200 トークンとなる。一方、プロンプトの処理（**プレフィル**）や訓練では一度に多数のトークンを扱い、読み出した各重みを再利用するため、今度は演算性能が律速となる。マシンはそれぞれの軸で 1 桁もの差があり、しかも互いに逆方向の特性をもつことが多いため、最適なマシンはワークロードによって異なる。

39.4.1 ワークロードの要件

機材選びはカードからではなくワークロードから出発する。訓練において問うべきは、ピークメモリ（Figure 39.7）がメモリに収まるかどうかである。

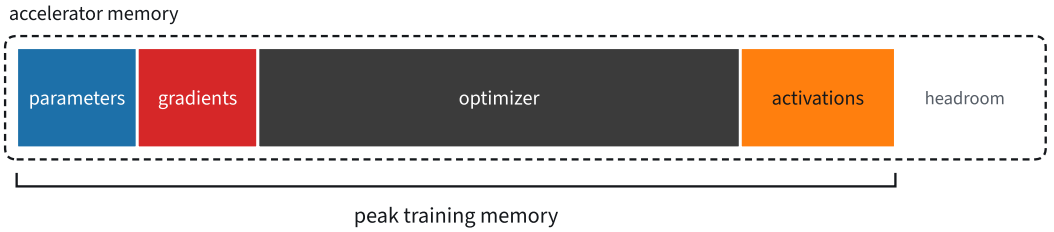


Figure 39.7: Adam を用いて BF16 で訓練する 7B モデルのピーク訓練メモリ（概ね実寸比）。パラメータの上に勾配、オプティマイザ状態、活性化値が積み重なり、その合計が余裕を残してアクセラレータのメモリに収まらなければならない。

```
def training_memory_gib(parameters_billion, weight_bytes=2,
                        gradient_bytes=2, optimizer_bytes=8,
                        activation_gib=6, workspace_gib=2):
    parameters = parameters_billion * 1e9
    state = parameters * (weight_bytes + gradient_bytes + optimizer_bytes)
    return state / 2**30 + activation_gib + workspace_gib
```

```
for b in [0.1, 1, 7]:
    print(f"{b:>4}B params: {training_memory_gib(b):6.1f} GiB")
```

```
 0.1B params:    9.1 GiB
   1B params:   19.2 GiB
   7B params:   86.2 GiB
```

7B モデルのフルファインチューニングには約 86 GiB が必要であり、どのコンシューマ向けカードの容量も超えてしまう。QLoRA (Dettmers et al. 2023) は重みを凍結して 4 ビットで保持し、その脇で小さなアダプタ行列を訓練する手法である。これにより同じジョブの訓練状態が 78 GiB から約 4 GiB に削減されるため、7B モデルのアダプタによるファインチューニングの一部は 16 GB のカードに収まる。一方、推論については、本節で扱う各マシンにデコード上限を適用してみよう。

```
bandwidth_gbs = {"RTX 5070 Ti": 896, "RTX 3090 (used)": 936,
                 "RTX 5090": 1792, "RTX PRO 6000": 1792,
                 "DGX Spark": 273, "Strix Halo": 256,
                 "M5 Max": 614, "M5 Ultra": 1200, "H100 SXM": 3350}
active_gb = 4.5   # 8B model at 4-bit, including cache traffic
for device, bw in sorted(bandwidth_gbs.items(), key=lambda kv: -kv[1]):
    print(f"{device:>16s}: <= {bw / active_gb:5.0f} tok/s")
```

```
       H100 SXM: <=   744 tok/s
       RTX 5090: <=   398 tok/s
   RTX PRO 6000: <=   398 tok/s
       M5 Ultra: <=   267 tok/s
RTX 3090 (used): <=   208 tok/s
    RTX 5070 Ti: <=   199 tok/s
         M5 Max: <=   136 tok/s
      DGX Spark: <=    61 tok/s
     Strix Halo: <=    57 tok/s
```

実効速度はカーネルに左右されるが、この上限は多くの場合マシンの順位を正しく予測する。DGX Spark と Strix Halo マシンはほぼ同じ帯域幅をもち、同程度の速度でデコードする（39.4.3 節）。Spark の価格が約 2 倍で、演算性能がはるかに高いにもかかわらずである。

Figure 39.8 は、これら 2 つの軸上に代表的なマシンを配置したものである。マシンは大きく 2 つの系統に分かれる。高速だが容量の小さいディスクリート GPU と、大容量だが低速なユニファイドメモリシステムである。ワークステーションクラスの RTX PRO 6000 は、高価格ながらこの両者の間を橋渡しする。

39.4.2 訓練向けのディスクリート GPU

多くの訓練やファインチューニングのワークロードにおいて、NVIDIA のディスクリート GPU は高いメモリ帯域幅と幅広いソフトウェアサポートを備えている。本書のフレームワークをはじめ、FlashAttention、量子化カーネル、そしてレンタルマシンで一般的な CUDA スタックがそのまま利用できる。

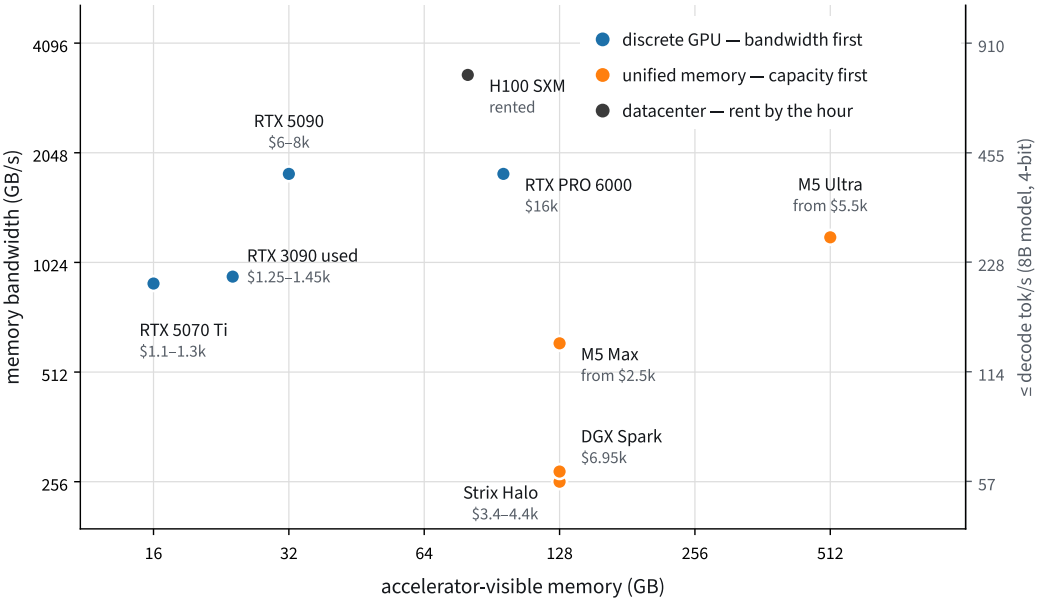


Figure 39.8: ディスクリート GPU は控えめな容量と非常に高い帯域幅を組み合わせ、ユニファイドメモリマシンはそのトレードオフを逆転させる。レンタルのデータセンタ GPU（H100）が帯域幅軸の頂点に位置し、容量と帯域幅がともに高い右上隅に到達する選択肢は存在しない。

39.4.2.1 RTX 5070 Ti と中古の RTX 3090

最初の訓練用マシンでは、メモリ容量が最低条件を決める。RTX 5070 Ti（16 GB GDDR7、896 GB/s、300 W）は発売時の定価が $749 で、2026 年 11 月時点の実売価格は約 $1,100〜1,300 であった。16 GB あれば、最大でもピーク時 13 GiB 近くにとどまる本書のすべての単一 GPU ノートブックを実行でき、7〜8B モデルの QLoRA ファインチューニングも収まる。これに対し 12 GB のカードでは、はるかに多くの構成が除外されてしまう。メモリの価格はデスクトップ全体の構成にも影響し、64 GB の DDR5 は約 $1,000〜1,200 と、カード本体と同程度の費用がかかる。中古の RTX 3090（24 GB、936 GB/s、約 $1,250〜1,450）はより大きなメモリと同程度の帯域幅を備えるが、消費電力が 350 W あり、FP8 および FP4 のテンソルコアをもたず、保証も付かないうえ、その中古価格は 2026 年を通じて上昇した。

39.4.2.2 RTX 5090 ワークステーション

16 GB や 24 GB では足りない場合、デスクトップにおける次の段階は 32 GB である。RTX 5090（32 GB GDDR7、1.79 TB/s、575 W）は定価 $1,999 であったが、供給が逼迫していた 2026 年 11 月時点では新品が約 $6,000〜8,000 で販売されていた。これはコンシューマ向けとして最速のカードであり、H100 の約半分の帯域幅、FP8 および FP4 のテンソルコア、そして約 2B パラメータまでのフルファインチューニングと約 30B までの QLoRA ファインチューニングに十分なメモリを備えている。NVIDIA はシステム電源として 1000 W を指定しており、筐体は 575 W の熱を排出し、電源回路はシステム全体の負荷に耐えなければならない。バランスの取れた構成（Figure 39.9）では、ここに 16 コア CPU、64〜128 GB の RAM、高速な NVMe ストレージを組み合わせ、総額は約 $8,500〜12,500 となる。

中古の 3090 を 2〜4 枚組み合わせて構築されることが多いマルチ GPU のコンシューマシステムは、メモリ容量を合算できる。データ並列は PCIe 経由でも機能するが、コンシューマ向けマザー

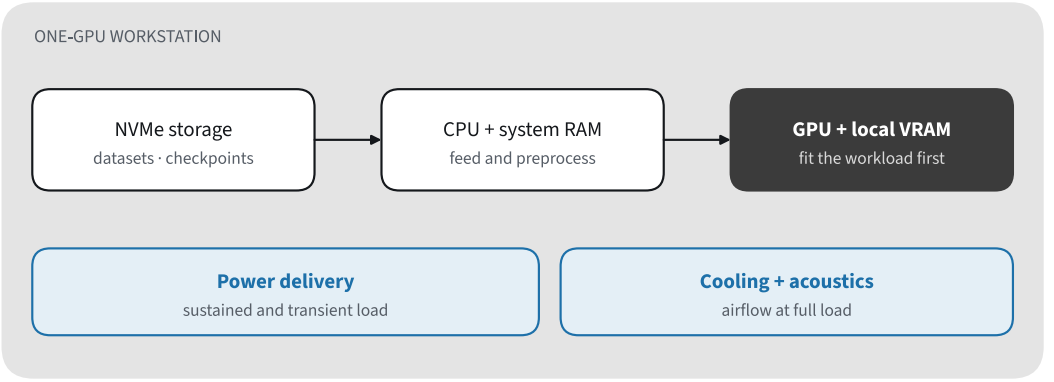


Figure 39.9: バランスの取れたワークステーションは、GPU にデータを供給し、データとチェックポイントを保存し、高負荷時の電力供給と冷却を維持する。

ボードが備えるフル帯域幅のスロットは 1 本だけであるため、自作する際は PCIe レーン数を確保するために Threadripper や EPYC プラットフォームを用い、電源ユニットを 2 基搭載する構成も珍しくない。3090 以降のコンシューマ向けカードには NVLink がないため、通信量の多いテンソル並列はうまくスケールしない。購入に踏み切る前にマルチ GPU ノード（39.3 節）を短時間レンタルしてみれば、インターコネクトの帯域幅がワークロードを制限するかどうかを確認できる。

39.4.3 推論向けのユニファイドメモリマシン

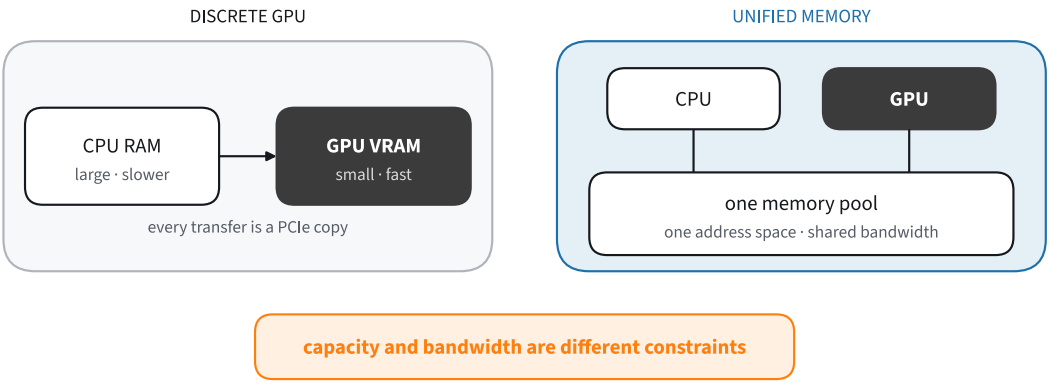


Figure 39.10: ディスクリート GPU はシステム RAM と VRAM の間でデータをコピーする。ユニファイドメモリシステムは単一のアドレス空間を共有するが、容量と帯域幅は依然として別個の制約である。

第 2 の系統のマシンは、これとは異なる問いに答える。どれだけ速く訓練できるかではなく、どれだけ大きなモデルを自宅で**実行**できるかという問いである。これらのシステムでは、ノート PC やスマートフォンで使われるメモリである LPDDR5X の大容量プールに GPU が直接アクセスでき、GDDR7 や HBM の数分の 1 の帯域幅と引き換えに大きな容量が得られる（Figure 39.10）。そのトレードオフはデコード上限が予測するとおりであり、大規模モデルはメモリに収まるもののデコードは遅くなる。混合エキスパート（mixture-of-experts、MoE）モデルは各トークンに対

して一部のエキスパートネットワークのみを実行するため、アクティブパラメータだけを読み出せばよく、総サイズが同程度の密なモデルよりも高速にデコードできる。

Table 39.3: ローカル推論向けのユニファイドメモリマシン（2026 年 11 月時点）

マシン	メモリ	帯域幅	概算価格	備考
AMD Strix Halo ミニ PC（Ryzen AI Max+ 395）	128 GB	256 GB/s	$3,400〜4,400	Framework、GMKtec、Minisforum、Beelink ほか。Linux 上の ROCm または Vulkan
NVIDIA DGX Spark（GB10）	64 または 128 GB	273 GB/s	$5,000（64 GB、OEM パートナ製）、$6,950（128 GB）	CUDA ネイティブの Blackwell。強力なプレフィル。ConnectX-7 経由で最大 4 台をクラスタ接続可能
Mac Studio M5 Max	最大 128 GB	614 GB/s	$2,500〜	MLX エコシステム。ベースモデルは 36 GB・460 GB/s
Mac Studio M5 Ultra	96〜512 GB	1.2 TB/s	$5,500〜	このクラスで最大のメモリ容量と帯域幅
MacBook Pro M5 Max	最大 128 GB	614 GB/s	$3,600〜	ノート PC に搭載された M5 Max。ベースモデルは 36 GB・460 GB/s

コミュニティによる実測値も、おおむね帯域幅の順序に従っている。MoE モデルの gpt-oss-120b は 117B のパラメータをもつが、1 トークンあたりに使用するのはそのうち約 5B である。llama.cpp のベンチマークでは、DGX Spark で約 60 tokens/s、Strix Halo で約 50 tokens/s を生成し、M5 Ultra のレビュアは Ollama や MLX を用いて約 115〜120 tokens/s と報告している。一方、4 ビットの密な 70B モデルは 1 トークンあたり約 40 GB を読み出すため、デコード上限だけでも Spark と Strix Halo では 7 tokens/s 未満、M5 Ultra でも 30 tokens/s 未満に抑えられる。マシン間の差がより大きく表れるのはプレフィルであり、同じ llama.cpp のベンチマークにおいて Spark のテンソルコアは 2,048 トークンのプロンプトを Strix Halo より約 2.5 倍高速に処理する。ソフトウェアスタック（Spark の CUDA、Strix Halo の ROCm や Vulkan、Apple マシンの MLX）も影響するため、実際に使いたいモデルで計測する必要がある。

39.4.4 ワークステーション GPU

たとえば自宅で 70B モデルをファインチューニングするなど、容量と帯域幅の両方を必要とする購入者はワークステーションクラスに行き着く。RTX PRO 6000 Blackwell は、5090 と同じ 1.79 TB/s の帯域幅で 96 GB の GDDR7 を搭載する。2025 年に $8,565 で発売され、2026 年 11 月時点では NVIDIA のストアで $16,000 で掲載されていた。同じメモリ容量をもつ 300 W の Max-Q モデルであれば、1 つの筐体に 4 枚のカードを収められる。2 枚で 192 GB となり、非量子化の 70B 推論や 70B の LoRA ファインチューニングに十分な容量となる。4 枚では 384 GB となり、数千億パラメータ規模の 4 ビット MoE モデルも収まる。ワークステーション向けカードは Ampere 世代以降 NVLink を失ったため、カード間の通信は片方向あたり約 64 GB/s の PCIe 5.0 を経由す

ることになり、各カード自身のメモリより 30 倍近く遅い。データ並列やパイプライン並列のワークロードはこの帯域幅でも耐えられるが、テンソル並列は減速する。

これほどの予算をかけるなら、代替手段のコストも計算すべきである。カード 4 枚分の $64,000 があれば、1 時間あたり $2 のマーケットプレイス料金（Table 39.2）で約 32,000 H100 時間分をレンタルできる。購入の元が取れるのは、レンタル費用が購入価格と等しくなるだけの実稼働時間に達した時点である。

```
purchase = 10500.0                  # an RTX 5090 workstation, Nov. 2026
rental_per_productive_hour = 2.00   # H100 marketplace rate
print(f"{purchase / rental_per_productive_hour:.0f} "
      "productive hours to break even")
```

```
5250 productive hours to break even
```

1 日 8 時間の実稼働とすると、5,250 時間には約 1.8 年かかり、しかも自宅のマシンは持ち主が思う以上に遊休していることが多い。電気代、再販価値、そしてマシン間の速度差を加味するとこの時間数はさらに変わる（4）。その一方で、購入する側にはプライバシー、待ち行列がないこと、そして自ら分解して触れられるハードウェアならではの価値がある。

39.4.5 最新数値の情報源

上記の価格は今後変化するし、新しい実測値であっても裏付け確認は欠かせない。自作構成の助言や実売価格については r/LocalLLaMA（ハードウェアのメガスレッド）や Level1Techs フォーラム（マルチ GPU の構築ログや Strix Halo の検証ノート）を、Strix Halo マシン・DGX Spark・中古サーバーのレビューについては ServeTheHome を参照するとよい。また、横並びで比較できる数値については、多数のマシン上で同一のモデルファイルに対して `llama-bench` を実行している llama.cpp のディスカッション内のベンチマークスレッドが参考になる。Tim Dettmers のガイド（最終更新は 2023 年）は、現行パーツの推奨ではなくワークロードに基づく選定手法を提示している。GPU に関する検索結果の多くは、推測値や捏造されたベンチマークを引用する機械生成ページである。信頼すべきはデータシート、実名のレビュア、そしてデコード上限の検証を通過する数値である。

39.4.6 まとめ

ハードウェアの選定は、律速となっている物理的制約を見極めることから始まる。メモリ容量はそもそも何が実行できるかを決め、メモリ帯域幅は自己回帰デコード速度（トークン生成速度を 2 倍以内の精度で予測できる）を決め、演算スループットは訓練およびプロンプトのプレフィル速度を決める。2026 年 11 月時点では、16 GB の RTX 5070 Ti（約 $1,100〜1,300）が入門用のディスクリート訓練カードとなり、32 GB の RTX 5090（約 $6,000〜8,000）が大きな電力・冷却要件を伴うハイエンドデスクトップの中核を担い、中古の RTX 3090 は保証や新しいテンソル形式と引き換えに 24 GB の容量を提供する。ユニファイドメモリシステム（Strix Halo、DGX Spark、Apple Silicon）はこのトレードオフを逆転させ、低めの帯域幅と引き換えに大容量を提供するため、スパースな混合エキスパートモデルが特に魅力的な選択肢となる一方、同程度の帯域幅をもつマシン間ではプロンプトのプレフィル性能が差を生む。96 GB の RTX PRO 6000 クラスのワークステーションカードは 5 桁ドル台の価格で両系統の間を橋渡しするが、同等のクラウドレンタル時間と常に比較検討すべきである。依拠する価格には必ず時点を明記し、実測値は機械生成の検索ページではなくデータシートやコミュニティの一次ベンチマークと照合して裏付けを取る必要がある。

39.4.7 演習問題

1. [code] **訓練メモリ**。`training_memory_gib` は各パラメータに対して重みに 2 バイト、勾配に 2 バイト、オプティマイザ状態に 8 バイトを計上し、さらに活性化値とワークスペースの固定量を加算する。
   1. Adam を用いた混合精度訓練では、Rajbhandari et al. (2020) の算出にあるとおり、パラメータあたり 16 バイトを保持する。`training_memory_gib` のデフォルト設定が省いているテンソルはどれか。また、それを加えると 7B モデルの見積もりはどう変わるか。両方の算出法のもとで、カードの GB を GiB とみなしたとき、16、24、32、96 GB のカードにフルファインチューニングが収まる最大のモデルサイズを求めよ。
   2. LoRA (Hu et al. 2021) は事前学習済みの重みを凍結し、その脇で小さなアダプタ行列を訓練する。QLoRA (Dettmers et al. 2023) はさらに凍結した重みを 4 ビットで保存し、そこへ量子化定数としてパラメータあたり 0.127 ビットを加える。`training_memory_gib` に訓練可能なパラメータの割合を表す引数を追加せよ。凍結パラメータは保存バイト数のみを消費し、訓練可能パラメータは従来どおり重み・勾配・オプティマイザ状態を消費するものとする。訓練可能パラメータを 1% とし、デフォルトの活性化値およびワークスペース項を用いた 8B モデルについて、凍結 bf16 重みを用いる LoRA と QLoRA のそれぞれでこの関数を評価せよ。2 つのファインチューニングのうち 16 GB カードに収まるのはどちらか。また、収まる方において最大の項はどれか。
   3. 活性化値の項は定数ではなく、ステップあたりのトークン数とともに増大する。ブロック単位の活性化チェックポイント（13.4.3 節）を用いると、モデルが保持するのは各ブロックの入力（トークンあたり d 個の bf16 値）と、再計算中の 1 ブロック分の活性化値（この小問では無視する）のみとなる。第 2 小問の QLoRA ファインチューニングに Llama 3.1 8B (Grattafiori et al. 2024) の形状、すなわち幅 $d = 4096$ の 32 ブロックと 128,256 トークンの語彙を与えよ。16 GB カードが活性化値用に残しているメモリには、ステップあたり何トークンが収まるか。
   4. 出力層は損失計算のため、語彙エントリおよびトークンごとに 1 個の bf16 ロジットも保持する。このロジットも計上したとき、ステップあたり何トークンが収まるか。トークンあたりのロジットのバイト数を、トークンあたりのブロック入力のバイト数と比較せよ。
2. **コンテキスト長とデコード**。各デコードステップは重みだけでなく過去の全トークンの KV キャッシュも読み出す。`bandwidth_gbs` を定義するコードが 4 ビットの 8B モデルにトークンあたり `active_gb = 4.5` GB を計上しているのはそのためである。L 個の層と H_{kv} 個のキー・値ヘッド（次元 d_h）をもつモデルは、コンテキスト 1 トークンあたり $2LH_{\text{kv}}d_h$ 個の値（各層・各ヘッドにつきキー 1 つと値 1 つ）を保存する（11.3 節）。Llama 3.1 8B (Grattafiori et al. 2024) は 32 個の層と 8 個のキー・値ヘッド（次元 128）をもち、キャッシュを bf16 で保持する。
   1. トークンあたりのキャッシュのバイト数を計算せよ。`active_gb` に含まれる 0.5 GB のキャッシュ転送量は、どのコンテキスト長に対応するか。
   2. コンテキスト長が 4K、32K、128K トークンのそれぞれについて、RTX 5070 Ti と M5 Ultra におけるデコード速度の上限を求め、トークンあたりの読み出しバイト数に占めるキャッシュの割合を示せ。これらのコンテキスト長のうち、重みとキャッシュが 16 GB カードになお収まるのはどれか。
   3. 重みを bf16 から 4 ビットへ量子化すると、重みのバイト数は 4 分の 1 になる。これによりデコード上限は 4K トークン時で何倍になり、128K トークン時では何倍になるか。長いコンテキストを用いるワークロードをもつ購入者にとって、この結果は何を意味するか。
3. **混合エキスパートのデコード**。混合エキスパート（MoE）モデルは多数のエキスパートネットワークを保持するが、各トークンに対してはそのうち少数のみを実行するため、デコードステップが読み出すのは**アクティブ**なパラメータのみである。300 億個のパラメータのう

ちトークンあたり 30 億個がアクティブな 30B-A3B モデルと、密な 30B モデルを考えよう（いずれも 4 ビット重みとし、KV キャッシュは無視する）。

1. Equation 39.4 を用いて、Table 39.3 のすべてのマシンにおける両モデルのデコード速度の上限を求めよ。対話利用の一般的な目安である 20 tokens/s より速く各モデルをデコードできるマシンはどれか。
2. 4 ビットの密な 3B モデルがトークンあたりに読み出すバイト数は、この MoE モデルと同じである。密な 3B モデルにはない、MoE モデルならではの購入者側のコストは何か。また、`bandwidth_gbs` に挙げたカードのうち、この MoE モデルを実行できるのはどれか。
3. gpt-oss-120b のモデルカード (OpenAI 2025) によると、36 個の MoE 層のエキスパートに 114.71B パラメータ、アテンションに 0.96B パラメータ、そして重み共有されていない同サイズの埋め込み行列と出力行列に合わせて 1.16B パラメータが含まれる。各層には 128 個のエキスパートがあり、1 トークンはそのうち 4 個を使用する。エキスパートはパラメータあたり 4.25 ビットの MXFP4 で保存され、それ以外の重みはすべて bf16 で保存される。埋め込み参照が読み出すのは 1 行のみであるとして、1 デコードステップが読み出すバイト数を計算せよ。DGX Spark と Strix Halo におけるデコード速度の上限を求め、39.4.3 節で報告した llama.cpp の速度と比較せよ。
4. DGX Spark での実測速度は、上で求めた上限を上回る。この計測に使われた llama.cpp のモデルファイルは、116.83B パラメータに対して 59.02 GiB を占めている。このファイルサイズは、エキスパート以外の重みの形式について何を示唆するか。1 デコードステップあたりのバイト数と、実測速度が各上限に対して達成している割合を再計算せよ。DGX Spark と Strix Halo の比較から、デコード速度の予測指標としての帯域幅について何が言えるか。

4. [code] **運用コストを含めた損益分岐点**。損益分岐点を計算するコードでは、`purchase` を `rental_per_productive_hour`（所有マシンが 1 実稼働時間に行う仕事量に相当するレンタル料金）で割っている。
   1. 所有マシンは実稼働時間ごとに電力を消費し、また後で売却することもできる。購入価格 C、再販価値 R、負荷時のコンセント消費電力 p、1 kWh あたりの電気料金 e、1 実稼働時間あたりのレンタル料金 r に対する損益分岐条件を書き下し、実稼働時間数 H について解け。
   2. H を次の条件で評価せよ。39.4.2 節の RTX 5070 Ti デスクトップの価格を $3,000、3 年後の再販価値を $800、コンセント消費電力を 450 W とし、Table 39.2 のマーケットプレイスの RTX 4090（1 時間あたり $0.35）と比較する。電気料金は 1 kWh あたり $0.20 とし、5070 Ti が 1 時間あたりに完了する仕事量は 4090 の 70% であると仮定する。1 日 8 時間の実稼働としたとき、H は何年分に相当するか。
   3. 本文のコードは 5090 マシンを H100 クラスのレンタル料金と比較しており、どちらのマシンでも 1 時間を同じ仕事量とみなしている。Table 13.2 を用いて、計算律速の訓練と帯域幅律速のデコードのそれぞれにおいて、H100 が 1 時間あたり何倍の仕事を行うかを見積もれ。そのうえで、両方のワークロードについて H を再計算せよ（39.4.2 節の 5090 構成の価格帯の中央値を用い、再販なし、コンセント消費電力 900 W とする）。
5. **プレフィル対デコード**。●リクエストはまずプロンプトを処理し（プレフィル）、続いて回答を 1 トークンずつ生成する（デコード）。39.4.3 節によれば、gpt-oss-120b のデコード速度は DGX Spark で約 60 tokens/s、Strix Halo で約 50 tokens/s である。プレフィルが DGX Spark では 2,400 プロンプト tokens/s、Strix Halo では 1,000 プロンプト tokens/s で動作し、すべての回答が 500 トークンであると仮定する。
   1. 500 トークンのプロンプトをもつチャットリクエストと、30,000 トークンのプロンプトをもつコーディングエージェントのリクエストについて、両マシンにおける最初のトークンまでの時間と合計時間を計算せよ。各リクエストにおいて DGX Spark は何倍高速か。128 GB の Spark は Strix Halo マシンの約 2 倍の価格である（Table 39.3）。

高い価格を正当化しやすいのはどちらのワークロードか。
2. gpt-oss-120b (OpenAI 2025) の 5.13B 個のアクティブパラメータについて、アクティブパラメータおよびトークンあたり 2 FLOPs と数え、アテンションスコアの FLOPs は無視する。両マシンのプレフィル速度を実効 FLOP/s に換算せよ。
3. プロンプトを 512 トークンずつのチャンクで処理するプレフィルでは、チャンク内のいずれかのトークンが使用する重みをすべて、チャンクごとに 1 回読み出す。各トークンは他のトークンと独立に、各層の 128 個のエキスパートから一様ランダムに 4 個を選ぶと仮定する。1 つのチャンクは、1 層あたり平均して何個の異なるエキスパートを読み出すか。問題 3c の算出法を用いて 1 チャンクが読み出すバイト数を計算し、各マシンのプレフィル速度の上限をその帯域幅から求めよ。512 トークンのチャンクで仮定したプレフィル速度に到達できるのはどちらのマシンか。この結果は、DGX Spark の計測におけるチャンクサイズについて何を示唆するか。
4. チャンクサイズを 2,048 トークンにすると、DGX Spark のプレフィルに対する帯域幅上限はどう変化するか。同じアクティブパラメータ数をもつ密なモデルと比べて、MoE モデルのプレフィルが計算律速になるまでにより長いチャンクを必要とするのはなぜか。

6. [extended] **デコード上限の較正。●●**まとめでは、Equation 39.4 が生成速度を 2 倍以内の精度で予測すると述べた。手元で利用できるハードウェア上でこの主張を検証し、単純な上限を較正済みモデルに置き換えよ。llama.cpp は CPU 上でも動作するため、ノート PC の CPU で十分である。
1. 大規模なストリーミングコピーにおいて手元のマシンが持続できる帯域幅を計測し、データシート記載の帯域幅と比較せよ。
2. llama.cpp の `llama-bench` を用い、モデルサイズと重み形式にまたがる少なくとも 6 通りの構成（たとえば 4 ビットおよび 8 ビットの 1B・3B・8B パラメータの密モデルと、1 つの MoE モデル）についてデコード速度を計測せよ。各構成のパラメータ数と形式からトークンあたりに読み出すバイト数を計算し、上限に対して実測速度をプロットせよ。
3. モデル $1/\text{rate} = t_0 + \text{bytes}/(\eta\beta)$（ただし β は第 1 小問の帯域幅）を当てはめ、デコードが達成する帯域幅の割合 η とトークンあたりの固定時間 t_0 を推定せよ。当てはめからの乖離が最も大きい構成はどれか。また、それはなぜか。
4. 当てはめに使用しなかった 1 つの構成、およびコンテキスト長 16K トークンにおける 1 つの構成（問題 2 のキャッシュバイト数を加算する）についてデコード速度を予測せよ。両方を実測し、予測誤差をまとめで述べた 2 倍という目安、および 39.4.5 節の情報源から得られる手元のハードウェアの公表値と比較せよ。

39.5 エコシステム

本書で扱ってきたモデルの大半は、あらゆるステップが見えるように、Fashion-MNIST や『The Time Machine』を用いてゼロから訓練したものであった。しかし、実際のプロジェクトがそのように始まることはめったにない。事前学習済みのチェックポイント、公開データセット、論文とともに公開されたコードといった、誰かの先行成果から出発するのである。たとえば、小規模な企業のサポートチケットを振り分けるモデルが欲しいとする。Hugging Face Hub でテキスト分類器を検索すると 10 万件以上の候補が返され、それぞれにダウンロード数があり、その多くにはモデルカードとベンチマークスコアが添えられている。最初にどれを試すべきだろうか。報告されているスコアは手元のチケットでも維持されるだろうか。そして来年読み込むモデルは、今日評価したものと同一だろうか。

これらは訓練そのものではなく、根拠と同一性に関する問いである。これに答えるとは、ソフトウェアエンジニアが依存関係（dependency）を扱うのと同じように外部アーティファクトを扱うことである。すなわち、コミュニティが保管している場所で探し、ランキングだけでなく自身

のタスクでの実測によって評価し、どのバージョンをどのライセンスの下で使ったかを記録する。その記録がなければ、結果を再現することはできない。また、善意でダウンロードしたチェックポイントであっても、事前にファイル形式を確認しなければ手元のマシン上でコードを実行してしまうおそれがある。モデルの選択とマシンの選択は互いに制約し合うため、本節はハードウェアの各節に続けて配置されている。以下ではまず、モデル・データセット・論文がどこで見つかるかを見ていき、次にベンチマークとリーダーボードの読み方、実務家が最新情報を追うためのいくつかの情報源を取り上げ、最後にダウンロードしたものをピン留め（pinning）し、検査し、ライセンスを確認する方法を解説する。

39.5.1 モデル、データ、論文の入手先

39.5.1.1 モデル

Hugging Face Hub には、2026 年 11 月時点で 300 万件以上の公開モデルが掲載されている。各モデルは Git リポジトリに格納されており、モデルカードと本書で使用する各ライブラリとの連携機能を備えている。人気があるからといって用途に適しているとは限らないため、タスクで検索し、公開・更新時期とダウンロード数を確認したうえで、モデルカードを読む必要がある。Hub を補完する他の入手先もいくつかある。

- Alibaba のハブである ModelScope は、中国語圏におけるオープンモデルのエコシステムの大部分を担っており、2026 年 11 月時点で 280,000 件以上のモデルをホストしている。
- Kaggle Models は、精選された重みをホストしており、Kaggle のコンペティションおよびノートブック基盤と直結している（39.2 節）。
- Ollama ライブラリは、ローカルランタイム向けにパッケージ化された精選モデルのコンパクトな一覧である（39.7 節）。
- `timm` は幅広いビジョンバックボーンのコレクションを提供しており、Civitai はコミュニティによる画像生成のチェックポイントや LoRA アダプタをホストしている。

39.5.1.2 データセット

Hugging Face Datasets（2026 年 11 月時点で 100 万件以上の公開データセット）と Kaggle で、教師あり学習における大半の用途はまかなえる。事前学習（pretraining）規模のテキストについては、生の Common Crawl からフィルタリング済みの派生データセットへと系譜が連なっている。FineWeb は 2024 年の公開時点で 15 兆トークン、新しいクロールデータを追加した後のバージョンでは 18.5 兆トークン以上を収録しており、多言語版の FineWeb2 は同じパイプラインを 1,000 以上の言語に適用している。FineWeb は現在、2019 年に T5 とともに公開された C4 がかつて担っていたような、事前学習における一般的なオープンベースラインとなっている。画像・テキストペアについては、DataComp が統制されたデータセット構築と評価に重点を置いた LAION の代替を提供している。古いコーパスや極めて大規模なコーパスは、Academic Torrents を通じてのみ入手できる場合がある。入手先がどこであれ、データセットにもモデルと同様にバージョン、ライセンス、および文書化された失敗モードが存在する。データセットを採用する前には、そのドキュメントを読んでおく必要がある。

39.5.1.3 論文とコード

新しい研究はまず arXiv に登場する。Hugging Face Papers は日替わりのフロントページをキュレーションし、論文とコードを紐づけている。長年、論文からコードやベンチマークへの索引として機能してきた Papers with Code が 2025 年 7 月に閉鎖された際、Hugging Face Papers がその役割を引き継いだ。現在、旧アドレスはそこへリダイレクトされており、過去のデータは凍結されたアーカイブとして残っている。この出来事は、プロジェクトを単一の外部インデックスに依存させてはならない理由を示している。検索や議論には Semantic Scholar や alphaXiv が役

立つ。公開コードは通常 GitHub 上にあり、そこでの Issue の履歴、最近のコミット、報告されている再現実験は、論文そのものを超える判断材料となる。

39.5.2 ベンチマークとリーダーボード

リーダーボードはある 1 つの測定基準でモデルをランク付けするが、プロジェクトが必要とするのは別の基準、すなわちプロジェクト固有のタスクで高い性能を発揮するモデルである。また、静的なベンチマークは経年劣化する。モデルがスコアを飽和させるか、問題が訓練データにリークするためである。2 つのベンチマークの廃止がその両方を物語っている。1 万件以上のオープンモデルに対して固定ベンチマークを実行していた Open LLM Leaderboard は、モデルの飽和を受けて 2024 年にベンチマークを入れ替えた後、2025 年に廃止された。2026 年 2 月には OpenAI が、コーディングエージェント向けの実際の GitHub Issue 集である SWE-bench Verified について、データ汚染（contamination）と正しい修正を却下してしまうテストの存在を理由にスコアの報告を取りやめ、代わりにより難度の高い SWE-bench Pro を推奨した。したがって、モデル選択（model selection）では複数の種類の根拠を組み合わせる必要がある。

- Arena（旧 LMArena および Chatbot Arena）は、ブラインドでの一対比較による人間の選好（preference）に基づいてモデルをランク付けする。ここで測られるのはサイト利用者が平均して何を好むかであり、対象タスクが必要とするものとは異なる場合がある。
- LiveBench は、新しい問題を定期的に公開しつつ最新の問題を非公開に保つことで、データ汚染を抑えている。
- コーディングエージェント向けの SWE-bench のような能力別ベンチマークは、能力ごとにベンチマークを設ける形式をとっており、数学、長文脈（long context）、安全性にもそれぞれ専用のベンチマークがある。
- Artificial Analysis は、モデルおよびプロバイダ横断で品質、価格、レイテンシを測定している。
- OpenRouter のランキングは、顧客が送信したトークン数によってモデルを並べており、品質リーダーボードと順位が食い違う箇所で最も示唆に富む顕示選好（revealed preference）となっている。

候補を絞り込んだら、対象タスクから小規模な評価セットを作成し、各候補をその上で実行する。対象タスクにおいては、リーダーボード上で隣り合っているモデル同士がリーダーボードのスコア差よりはるかに大きく開くことも珍しくなく、順位すら入れ替わることがある。本書全体を通じて徹底してきた評価の規律（ホールドアウトデータ、意味のあるベースライン、誤り分析）は、モデルを訓練するときとまったく同様に、モデルを**選ぶ**際にも当てはまる。

39.5.3 情報源

新しいモデル、論文、ツールは毎日登場するため、すべての発表に目を通すことは誰にもできない。絶え間なく流れてくる情報を追い続けるよりも、意図的に絞った少数の情報源を持つ方が判断しやすい。2026 年 11 月時点で実用に適した組み合わせの一例を以下に挙げる。

- r/LocalLLaMA：オープンウェイトモデルのリリース、量子化、ハードウェアの実測値、および再現の試みが報告されるコミュニティである。
- arXiv に毎日投稿される数百本の論文を直接追う代わりに、Hugging Face Papers の日替わりセレクションを活用する。
- Andrew Ng の **The Batch**、Jack Clark の **Import AI**、あるいは Sebastian Raschka の **Ahead of AI** といったニュースレターを 1 つ購読する。
- システム分野については GPU MODE の講義とコミュニティ、オープンな研究については EleutherAI コミュニティが参考になる。

ソーシャルメディアは研究機関が成果を発表する場である。発見のきっかけとして利用するにと

どめ、引用には論文またはモデルカードを用いること。まとまった解説資料や書籍については、本章の導入部（第 39 章）に一覧を掲載している。

39.5.4 外部アーティファクトの管理

39.5.4.1 リビジョンのピン留め

モデルとは単なる重みだけではない。トークナイザ、前処理、設定、ライセンス、そしてリビジョンのすべてが、結果を再現できるかどうかを左右する（Figure 39.11）。

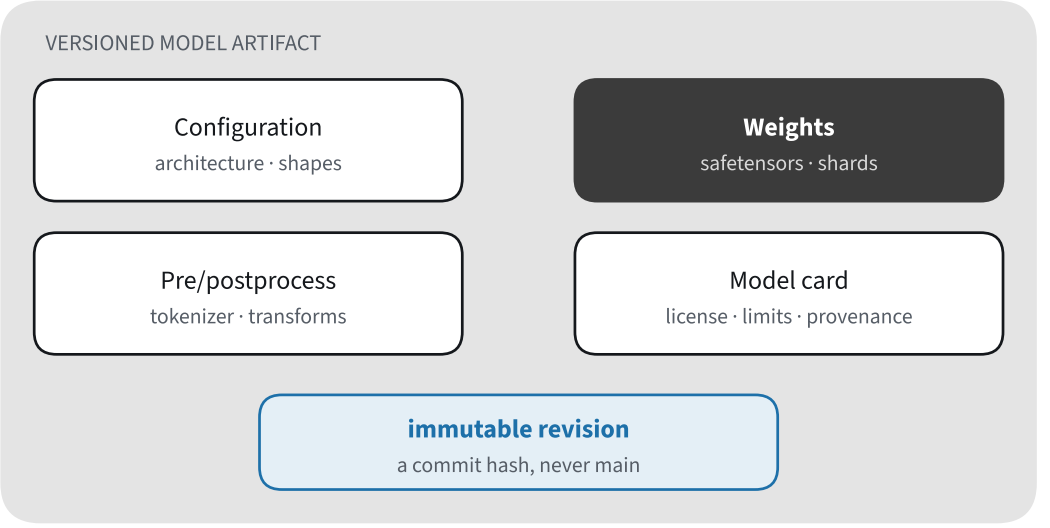


Figure 39.11: 再利用可能なモデルアーティファクトは、設定、重み、前処理、ドキュメント、および不変のリビジョンを組み合わせたものである。

Hub のリポジトリは Git リポジトリであり、所有者がコミットをプッシュすれば `main` ブランチは移動する。明日も同じファイルを読み込むには、特定のコミットを指定してダウンロードする。

```
from huggingface_hub import snapshot_download

path = snapshot_download(
    repo_id="organization/model-name",
    revision="0123456789abcdef",          # an immutable commit, not main
    allow_patterns=["*.json", "*.safetensors"],
)
```

小さなマニフェストを用意しておけば、モデルの同一性を明示的に記録できる。本番環境では、さらにファイルのハッシュ値、ライブラリのバージョン、および評価記録を追加する。

```
from dataclasses import asdict, dataclass
import json

@dataclass(frozen=True)
class Artifact:
```

```
      repository: str
      revision: str
      task: str
      license: str
      parent: str | None = None

  artifact = Artifact(repository="organization/model-name",
                      revision="0123456789abcdef",
                      task="text-generation", license="apache-2.0")
  print(json.dumps(asdict(artifact), indent=2))
```

```
{
  "repository": "organization/model-name",
  "revision": "0123456789abcdef",
  "task": "text-generation",
  "license": "apache-2.0",
  "parent": null
}
```

同じピン留めは**派生**アーティファクトにも当てはまる。ベースモデルのリビジョンを欠いた LoRA アダプタは不完全であり、量子化による変換結果（GGUF や AWQ、39.7 節参照）は元のモデルに対する数値的な忠実度（fidelity）を検証すべき新たなアーティファクトである。`parent` フィールド（`Artifact` クラス内）は、派生アーティファクトの元となったアーティファクトを記録する。

39.5.4.2 セキュリティとライセンス

ダウンロードしたモデルはソフトウェアサプライチェーンへの入力であり、他のあらゆるサードパーティ製ライブラリと同じセキュリティ上およびライセンス上の確認事項を伴う。

- テンソルのみを保持するコンテナ形式である **safetensors** を優先する。従来の pickle ベースのチェックポイントは、読み込み時に任意のコードを実行しうる。Hub は既知の悪意あるパターンについてこうしたファイルをスキャンしており、それによってリスクは低減するものの完全にはなくならない。
- `trust_remote_code=True` を指定すると、リポジトリ内の Python コードが手元のマシン上で実行される。そのコードを読み通してピン留めし、アーキテクチャ上どうしても必要な場合に限ってこのフラグを使用する。
- 「オープン」という言葉は幅広いライセンス形態にまたがっている。寛容な Apache や MIT ライセンスの重み、商用利用に上限を設けた利用規定（acceptable-use）ライセンス、研究用途限定の公開、アカウントごとに規約への同意を求めるゲート付きモデルなどがある。リポジトリはリビジョン間でライセンスを変更することがあるため、ピン留めしたリビジョン時点のライセンスを記録しておく。
- Hub クライアントはダウンロードしたすべてのファイルをキャッシュするため、`~/.cache/huggingface` が数十〜数百 GB に膨らむことも珍しくない。`hf cache ls` はキャッシュの一覧を表示し、`hf cache rm` は特定のリポジトリやリビジョンを削除し、`hf cache prune` はどのブランチやタグからも参照されなくなったリビジョンを未完了のダウンロードとともに削除する。

こうした習慣を身につければ、ハブ上で見つけたモデルは、選択理由を説明でき、再現でき、監査できる依存関係となる。ただし、これだけではそのモデルが自身のタスクに十分かどうかは決まらない。それに答えられるのは手元のデータによる評価だけであり、2 つのモデルを見分けるためにどれだけの規模の評価が必要になるかは、以下の演習問題で確認する。

39.5.5 まとめ

事前学習済みモデル、データセット、および論文は、主に Hugging Face Hub とそれを補完するサービス、そして arXiv で入手できる。選好ランキング、データ汚染に強いベンチマーク、能力別ベンチマーク、価格・レイテンシのデータはそれぞれ異なる側面を測定しており、またベンチマークはモデルの飽和や丸暗記に伴って廃止されていくため、単一のリーダーボードだけでモデルの選択が決まることはない。モデルの選択は、自身のタスクから構築したエラーバー付きの小規模な評価に基づくべきである。あらゆる外部アーティファクトはサプライチェーン上の依存関係にほかならない。リビジョンをピン留めし、safetensors を優先し、リモートコードを精査し、ライセンスと派生元（parent）を記録することが重要である。

39.5.6 演習問題

1. **Arena におけるレーティング差**。Arena（旧 LMArena）は、15.3 節の Bradley–Terry モデルを一対比較の投票データに当てはめ (Chiang et al. 2024)、そのスコアを Elo スケールで報告している。引き分けを無視すると、モデル A がモデル B に勝つ確率は次の式で与えられる。

$$P(A \succ B) = \frac{1}{1 + 10^{-(R_A - R_B)/400}}, \tag{39.5}$$

   ここで、R_A と R_B は報告されているスコアである。

   1. スコア差 $R_A - R_B$ が 10、20、50、100 ポイントのとき、この確率を計算せよ。
   2. $P(A \succ B)$ を $R_A = R_B$ の周りで線形化し、スコア差 $R_A - R_B$ を実力の近い 2 つのモデル間の独立な v 票から推定したとき、その標準誤差が約 $347/\sqrt{v}$ ポイントになることを示せ。20 ポイントの差が 0 から標準誤差の 2 倍以上離れるには、何票が必要か。
   3. 全投票ではモデル A がモデル B を 20 ポイント上回っているとする。対象タスクに似たプロンプトに投じられた 10% の票では、B が 60% 勝利している。残りの票における A の勝率はいくらか。この結果は A と B の選択に対して何を示唆するか。

2. **リビジョンがピン留めするもの**。Hugging Face のリポジトリは、`main` ブランチ、`v1.0` のようなタグ、`refs/pr/3` のようなプルリクエストの ref、または 40 文字のコミットハッシュによって指定できる。

   1. これらの識別子のうち、常に同じファイルを指すものはどれで、明日には別のファイルを指しうるものはどれか。それぞれを動かせるのは誰か。データセットのリポジトリも同じ方法で指定され、`HuggingFaceFW/fineweb` は過去の各バージョンを `v1.3.0` のようにそのバージョン名のブランチに保持している。訓練マニフェストには、このデータセットのどの識別子を記録すべきか。
   2. あるリポジトリのピン留めされたコミットに `config.json` が含まれており、その `auto_map` エントリが `"AutoModelForCausalLM": "other-org/shared-code--modeling.MyModel"` となっているとする。`trust_remote_code=True` のとき、`transformers` は `modeling.py` をリポジトリ `other-org/shared-code` からインポートする。`get_class_from_dynamic_module`（`transformers/dynamic_module_utils.py` 内）を読め。`modeling.py` のどのリビジョンが、手元のコミットを指定した `revision` のみを渡したときに実行されるか。したがって、マニフェストには何を記録しなければならないか。
   3. 関与するすべてのリポジトリをコミットにピン留めしていてもなお、読み込んだモデルの出力を変化させる、リポジトリ外の入力を 3 つ挙げよ。

3. [code] **読み込み時のコード実行**。Python の `pickle` は、オブジェクトの `__reduce__` メソッドが返す呼び出し可能オブジェクト（callable）を、同じくそれが返す引数で呼び出すことによってオブジェクトを再構築する。

   1. `__reduce__` が `(print, ("code ran during load",))` を返すクラスを書け。そのインスタンスをテンソルも保持する辞書に入れて `torch.save` で保存し、`torch.load` でデフォルト引数の場合と `weights_only=False` を指定した場合の 2 回読み込め。それぞれの場合に何が起こるか。リポジトリ `openai-community/gpt2`（`model.safetensors`、`tf_model.h5`、`flax_model.msgpack` も格納されている）にあるファイルのうち、これと同じ unpickle 機構で読み込まれるのはどのファイルか。
   2. `weights_only=True` を指定しても、`torch.load` は依然として pickle のオペコードストリームを解釈し、クラスと関数の許可リストに含まれるものだけを認める仕組みをとっている。PyTorch 2.5.1 以前ではこの制限を回避できたため、細工されたファイルを読み込むと任意のコードが実行された（CVE-2025-32434）。safetensors のローダにこのような許可リストが不要なのはなぜか。信頼していないリポジトリから safetensors のモデルを読み込む際になお残るリスクは何か。

4. [extended] **タスク特化型のモデル選択。**●自身の専門分野におけるサポート質問への回答や、日常的に扱う文書からの項目抽出など、よく知っているタスクを 1 つ選べ。

   1. Arena または LiveBench、そのタスクが必要とする能力に対応するベンチマーク 1 つ、および Artificial Analysis を用いて、モデルを 3 つに絞り込め。ランキングはどこで食い違っているか。食い違っている各リーダーボードが何を測定しているかに基づいて、それぞれの不一致を説明せよ。
   2. 自身のタスクから評価セットを構築せよ。選択を左右する最小の正解率の差に対して、36.5.2.6 節の対応のある比較における標準誤差の解析を用いて評価セットのサイズを決めよ。ランダムなベースラインと、最頻出の回答やキーワード一致などの単純なヒューリスティックベースラインを追加せよ。
   3. 絞り込んだ候補モデルと各ベースラインを評価セット上で実行せよ。各スコアをその標準誤差とともに、また 2 つのモデル間の各差分を対応のある標準誤差とともに報告せよ。どの差が統計的に判別でき、得られたランキングはリーダーボードと一致するか。

   Chip Huyen の Machine Learning Systems Design「Model selection」より改変。

5. [code] **ゼロから作るテンソルコンテナ。**●safetensors ファイルは、8 バイトの符号なしリトルエンディアン整数 N、長さ N バイトの UTF-8 JSON ヘッダ、およびバイトバッファから構成される。ヘッダは各テンソル名を、`dtype`（`"F32"` や `"I64"` など）、`shape`、および `data_offsets`（バッファ先頭からの相対バイト位置のペア `[begin, end)`）に対応付ける。任意のキー `__metadata__` は文字列を文字列に対応付ける。テンソルはリトルエンディアンのバイト順かつ行優先レイアウトで格納される。

   1. NumPy 配列の辞書に対する `save_tensors(path, tensors, metadata)` と `load_tensors(path)` を、`json`、`struct`、`numpy` のみを用いて書け。保存して読み込み直す往復処理で元の配列が再現されることを確認せよ。`safetensors` パッケージがインストールされている場合は、`safetensors.numpy.load_file` で作成したファイルを読み込めることも確認せよ。
   2. `load_tensors` に引数を追加し、バッファの残りの部分を読み込むことなく `np.memmap` を介して指定した名前の 1 つのテンソルだけを読み込めるようにせよ。
   3. `model.safetensors` のヘッダのみを、リビジョン `607a30d783dfa663caf39e06633721c8d4cfcd7e`（リポジトリ `openai-community/gpt2`）から 2 回の HTTP レンジリクエストを使って読み取り、形状から格納されている値の総数を数えよ。この総数が GPT-2 small のパラメータ数である 1 億 2400 万個を超えるのはなぜか。
   4. 未知の提供元から得たヘッダを信頼する前にローダが実行すべき検査項目を列挙せよ。各検査項目に違反する不正な形式のファイルを 1 つずつ作成し、実装したローダがそれらすべてを拒否することを確認せよ。

39.6 分散モデル訓練

第 13 章のマルチ GPU に関する各節では、1 台のマシン上の 4 基の GPU それぞれにモデルのコピーを持たせることで ResNet と GPT を訓練し、さらに訓練状態を GPU 間で分割することで単一の GPU には収まらないモデルのための領域をどう確保するかを導出した。実務でファインチューニングされるモデルは、それよりさらに大きいことも珍しくない。70 億パラメータの言語モデルを Adam で訓練する場合、活性化値を 1 つも保存する前の時点で、重み、勾配、およびオプティマイザ状態（optimizer state）だけで約 112 GB を必要とする。これは H100 の 80 GB を上回っており、このようなモデルを 1 基の GPU 上でゼロから訓練しようとすれば何年もかかるだろう。

1 回の訓練実行における状態と計算を、場合によっては複数のマシンにまたがる多数のアクセラレータに分散させ、同じ結果をより短時間で計算するにはどうすればよいだろうか。どの解決策も何かを分割する。すなわち、バッチ、訓練状態、層、あるいは系列である。分割を増やすたびに通信が加わり、ワーカーを 1 つ追加するたびに実行が失敗しうる箇所が 1 つ増える。したがって、実測済みの単一デバイスのベースラインから出発し、観測されたメモリまたはスループットの制約を取り除く最も単純な手法を追加していく。本節では、各手法を取り除く制約の順に並べ、2026 年 11 月時点でそれらを実装しているライブラリに対応づけ、最後に数週間に及ぶ実行を途切れさせないための実践手法を扱う。

39.6.1 並列性の形態

39.6.1.1 スケーリングラダー

以下の各手法は、1 つ前の手法の限界を取り除く代わりに通信を増やすため、**スケーリングラダー**（Figure 39.12）の段を上るように順番に採用していく。

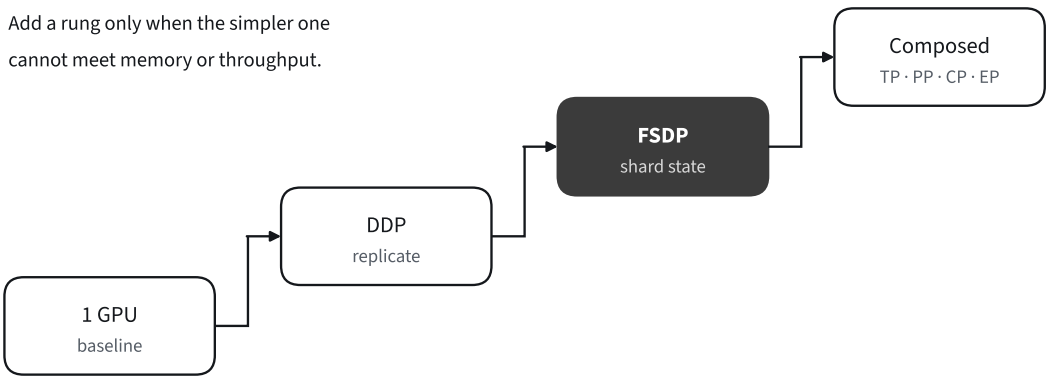


Figure 39.12: 実用的なスケーリングラダーは、単一のアクセラレータから複製、シャーディング、複合的な並列性へと段階的に進む。

1. **単一のアクセラレータ**で基準を確立する。具体的には、損失曲線、毎秒トークン数または毎秒データ例数、ピークメモリ、そして信頼できるチェックポイントである。
2. **データ並列**（data parallelism、DDP）は、すべての GPU 上にモデルを複製し、それぞれにバッチの異なるスライスを入力して、各逆伝播の後に勾配を平均する。モデル、勾配、オプティマイザ状態が 1 つのデバイスに収まる限り、常にこれが適切な選択肢となる。
3. **完全シャーディングデータ並列**（fully sharded data parallelism、FSDP）は ZeRO ファミリのステージ 3 にあたり、パラメータ、勾配、オプティマイザ状態を複数の GPU にシャーディングし、各層が使用されている間だけ集約する（13.7.1 節）。これによりデバイスあたりの状態メモリは GPU 数分の 1 になり、通信量は DDP の約 1.5 倍となる。

4. **テンソル並列、パイプライン並列、コンテキスト並列、エキスパート並列**（tensor, pipeline, context, and expert parallelism）はそれぞれ、個々の層、連続する層のグループ、系列次元、および混合エキスパート（mixture of experts、11.7 節）の各エキスパートを分割する。単一の軸だけでは十分にスケールしないため、大規模システムではこれらのいくつかを組み合わせる。

PyTorch では、DDP と FSDP の実行にランチャとラッパが必要である。

```
torchrun --standalone --nproc-per-node=4 train.py
```

`train.py` の内部では、`DistributedDataParallel(model)` が DDP を適用し、`fully_shard(model)`（`torch.distributed.fsdp` で提供）が FSDP を適用する。FSDP2 と呼ばれるこの API は、シャーディングされた各パラメータを `DTensor` として保持するものであり、PyTorch は従来の `FullyShardedDataParallel` ラッパよりもこちらを推奨している。JAX では、配列がデバイスメッシュ上の `NamedSharding` を持ち、それらの配置が意味する集団通信（collective）を `jax.jit` がコンパイルする（13.6 節）。

テンソル並列はすべての層の内部で活性化値を交換するのに対し、パイプライン並列はステージの境界でのみ活性化値を送信する。Hugging Face の **Ultra-Scale Playbook**（4,000 回を超える実行を最大 512 基の GPU 上で行った調査に基づく）によると、テンソル並列を 1 台の 8 GPU ノードから 2 台へ広げると通常スループットの約 43% が失われたのに対し、各ステップのマイクロバッチ数が少ない条件下でパイプライン並列を同様に広げた場合の低下は 14% であった。テンソル並列は NVLink がトラフィックを運ぶノード内に留め、通信量が最大の集団通信を最も高速なリンクに割り当てるべきである。

どの段においても 1 つの恒等式が成り立つ。オプティマイザから見たバッチサイズは次の式で与えられる。

$$B_{\text{global}} = B_{\text{device}} \cdot N_{\text{ranks}} \cdot N_{\text{accumulation}}, \tag{39.6}$$

すなわち、デバイスあたりのバッチサイズに、データ並列のランク数と、オプティマイザの 1 ステップあたりに累積されるマイクロバッチ数を掛けた値である。したがって、デバイスあたりのバッチサイズや学習率を調整せずに GPU 数を変更すると、最適化問題そのものが変わってしまう。データに関してはさらに 2 つの原則がある。分散サンプラはデータ例を重複させてはならず、評価時には正しい分母を用いて検証指標を集約しなければならない。いずれかに違反することが、マルチ GPU 実行で異なる正解率が報告される主な原因である。

39.6.1.2 訓練メモリ

大半のジョブが 1 基の GPU に収まらなくなる原因は訓練状態が入りきらないことにあるため、最初に行う計算はメモリバジェットの見積もりである（Figure 39.13）。

ピーク時の訓練メモリは、パラメータ、勾配、オプティマイザ状態、活性化値、通信バッファ、およびアロケータのオーバーヘッドの合計である。70 億パラメータのモデルを BF16 と Adam で訓練する場合、パラメータと勾配がそれぞれ 14 GB、Adam の 2 つの FP32 モーメント（一次モーメントと二次モーメント）が 56 GB を占める。混合精度訓練が保持するパラメータの FP32 マスターコピーがさらに 28 GB 加わり、合計 112 GB、すなわち 1 パラメータあたり 16 バイトとなる。以下の簡易モデルではその 28 GB のマスターコピーを省略し（GB の数値をそのまま GiB として流用すると、モデル状態は $14 + 14 + 56 = 84$ GiB 残る）、18 GiB の活性化値と 5 GiB の一時バッファを加えている（合計 107 GiB）。大まかな計算ではあるが、各戦略の順位を正しく捉えており、GPU なしで実行できる。

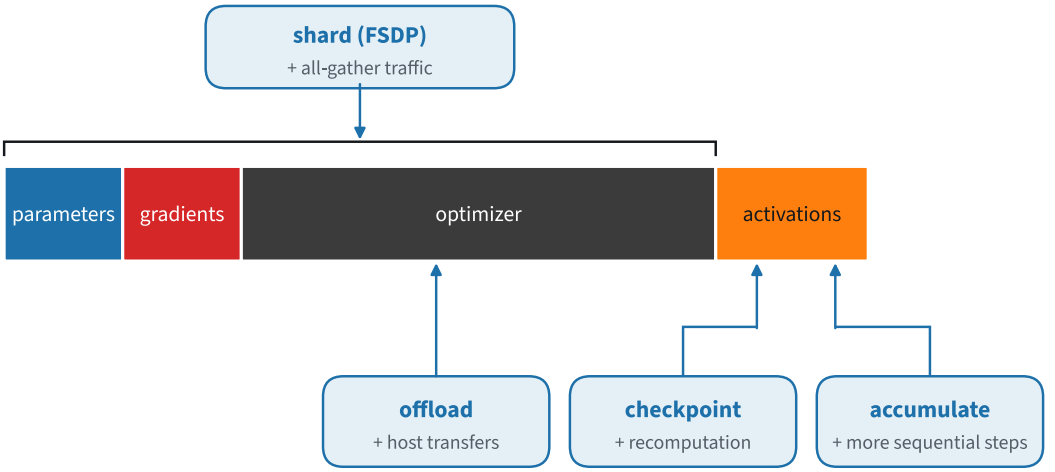


Figure 39.13: シャーディング、チェックポイント保存、勾配累積、オフロードはそれぞれ異なるメモリ項目を削減し、異なるコストをもたらす。

```
terms_gib = {"parameters": 14.0, "gradients": 14.0, "optimizer": 56.0,
             "activations": 18.0, "temporary": 5.0}

def per_device_gib(world_size=1, shard_state=False, ckpt_activations=False):
    divisor = world_size if shard_state else 1
    state = sum(terms_gib[k] / divisor
                for k in ("parameters", "gradients", "optimizer"))
    activations = terms_gib["activations"] * (0.35 if ckpt_activations else 1)
    return state + activations + terms_gib["temporary"]

for label, cfg in [("1 GPU, plain", (1, False, False)),
                   ("8 GPU DDP + ckpt", (8, False, True)),
                   ("8 GPU FSDP + ckpt", (8, True, True))]:
    print(f"{label:>20s}: {per_device_gib(*cfg):6.1f} GiB/device")
```

```
      1 GPU, plain:  107.0 GiB/device
 8 GPU DDP + ckpt:   95.3 GiB/device
8 GPU FSDP + ckpt:   21.8 GiB/device
```

通常の訓練には 107 GiB が必要であり、80 GB の H100 の容量を超える。活性化チェックポイント（activation checkpointing）を併用した DDP でも、各 GPU 上になお 95 GiB が必要となる。84 GiB の状態が複製され、縮小するのは活性化値だけだからである。8 基の GPU 上でチェックポイント保存を併用した FSDP ならば、状態が 8 分割されるため、デバイスあたり 22 GiB で済む。

標準的なメモリ削減手法は、それぞれ異なるコストと引き換えに合計の異なる項目を変化させる。**混合精度**（mixed precision）は、選択されたテンソルと演算が使用するバイト数を減らす。**勾配累積**（gradient accumulation）は、Equation 39.6 のグローバルバッチを維持したままステップあたりのマイクロバッチを小さくして実行することで、モデル状態は変わらないものの活性化メモリ（activation memory）を削減する。**活性化チェックポイント**は、誤差逆伝播法（backpropagation）の最中に選択された活性化値を再計算する手法であり、メモリの節約量と計算オーバーヘッドはどの活性化値を保持するかに依存する。**CPU または NVMe オフロード**（DeepSpeed の ZeRO-Offload など）は、状態をデバイスメモリから移動させ、毎ステップにホ

スト・デバイス間の転送を追加する。**LoRA と QLoRA** は全重みの代わりに小さな低ランクアダプタを訓練することで、大半の勾配とオプティマイザ状態を不要にする。さらに QLoRA は凍結した重みを 4 ビットで保持するため、7B モデルを 1 枚の 16 GB カード上でファインチューニングできる。複数の手法は慎重に組み合わせる必要がある。個々には妥当な最適化であっても、コンパイル、通信と計算のオーバーラップ、あるいは数値精度と干渉することがあるからである。

39.6.2 ソフトウェアライブラリ

1 ノードを超える規模では、これらの手法を手書きする人はほとんどいない。各手法はライブラリによって実装されており、どれを選ぶかは主にジョブの規模によって決まる。以下のプロジェクトは 2026 年 11 月時点で保守され広く使われていたものである。この領域は変化が速いため、採用する前に各プロジェクトの最新リリースを確認されたい。

PyTorch の組み込み機能。DDP と FSDP2 のほかにも、PyTorch には `DTensor` 上に構築されたテンソル並列およびコンテキスト並列の API と、パイプライン並列の API である `torch.distributed.pipelining` がある。ドキュメント上では依然として実験的（experimental）またはアルファ版と位置づけられているが、以下で紹介する上位ツールの大半はこれらのプリミティブへの呼び出しを生成する。

Hugging Face スタック。`accelerate` はコードを変更することなく設定ファイルの切り替えだけで同じスクリプトを DDP、FSDP2、または DeepSpeed 上で起動する。その上に `transformers.Trainer` が載り、`peft` が LoRA と QLoRA を実装し、`trl` が教師ありファインチューニング、DPO、および GRPO 型の強化学習向けのトレーナを提供する。1 ノードまたは小規模クラスタにおける大半のファインチューニングジョブは、このスタックで賄える。

DeepSpeed。ZeRO のステージ 1〜3（オプティマイザ状態、次いで勾配、さらにパラメータを順にシャーディングする）に加え、CPU および NVMe オフロードを提供する。現在 FSDP2 がネイティブに備えているシャーディングを導入したのは ZeRO であり、FSDP2 も状態を CPU メモリへオフロードできる。それでも、ジョブが NVMe へのオフロードを行う場合や、他のフレームワークが基盤として利用している場合には、依然として DeepSpeed が定番の選択肢である。

Megatron Core と TorchTitan。大規模な事前学習（pretraining）向けのライブラリである。NVIDIA の Megatron Core は FP8 および FP4 訓練を伴うテンソル並列、パイプライン並列、コンテキスト並列、エキスパート並列を実装しており、Megatron-LM はそのリファレンス訓練スタックにあたる。TorchTitan はこれに相当する PyTorch ネイティブの実装であり、FSDP2 とテンソル並列、パイプライン並列、コンテキスト並列、エキスパート並列を組み合わせるコンパクトなリファレンススタックである。Llama 3 405B や DeepSeek-V3 を含むモデル定義を備え、最大 512 基の GPU での実測値も公開されている。いずれも複数の形態の並列性を同時に必要とするモデルに適している。

ファインチューニング用フロントエンド。Unsloth は融合カーネルによる単一 GPU 上の高速な LoRA および QLoRA から始まり、現在ではフルファインチューニング、GRPO、およびマルチ GPU にも対応している。その高速化の度合いはモデル、系列長、ハードウェアに依存する。Axolotl は FSDP2 と DeepSpeed をバックエンドとして、1 つまたは複数のノード上でのフルファインチューニングやパラメータ効率的ファインチューニングを 1 つの YAML ファイルから駆動する。LLaMA-Factory も Web インターフェースを備えて同様の範囲をカバーしている。なお、PyTorch チームの `torchtune` は 2025 年に活発な開発を終了した。

JAX。MaxText は TPU および GPU 上での事前学習と事後学習（post-training）のためのリファレンススタックである。Levanter を用いて訓練する Marin は、生データから最終モデルに至るまでの訓練実行を失敗した実験も含めて記録し、第三者が再現できるようにコード、データパイプライン、およびログを公開している。

強化学習による事後学習。veRL は FSDP、FSDP2、または Megatron による訓練と、ロールアウト生成用の vLLM または SGLang とを組み合わせる。OpenRLHF は Ray、DeepSpeed、および vLLM の上に構築されている。

39.6.2.1 規模別のツール

Table 39.4: 規模別の訓練ツール（2026 年 11 月時点）

規模	典型的なジョブ	選ぶべきツール
1 GPU、≥8 GB	8B 以下の LoRA/QLoRA ファインチューニング	Unsloth、または PEFT + Trainer
1 ノード、2〜8 GPU	最大約 30B のフルファインチューニング、DDP/FSDP2	Accelerate または Axolotl
数ノード	大規模ファインチューニング、小規模事前学習	FSDP2 + torchrun、DeepSpeed
多数ノード	大規模事前学習、MoE	Megatron Core、TorchTitan、MaxText
強化学習による事後学習	GRPO/DPO パイプライン	TRL（小規模）、veRL/OpenRLHF（大規模）

表中のモデルサイズは、フルファインチューニングにおける 1 パラメータあたり 16 バイトという値から導かれる。8 基の 80 GB GPU があれば、活性化値のための余裕を残しつつ約 30B パラメータの状態を保持できる。表の下の行ほど多くの設定が必要になるため、実測した制約を満たす最も単純な行を選ぶとよい。

39.6.3 長時間実行における耐障害性

10 日目にクラッシュして再開できない実行は、何も生み出さなかったのと同じである。ワーカー数が増えジョブの実行期間が長くなるにつれて、ハードウェア、ネットワーク、ソフトウェアの障害は日常茶飯事になるため、復旧の仕組みも訓練の他の部分と同様に設計・テストしておかなければならない。

- **完全かつアトミックにチェックポイントを保存する**。再開可能なチェックポイントには、モデル状態とオプティマイザ状態、学習率スケジュールとステップ数、データローダの位置、および乱数生成器（RNG）の状態が含まれる。書き込み途中に訓練プロセスがクラッシュしても直前のチェックポイントを破壊しないよう、一時パスに書き出してから完了時にリネームし、復元がテストされるまでは複数の世代を保持しておく。シャーディングされた（ランクごとの）チェックポイントを使えば、単一ホストのメモリを超える大きさのモデルを 1 か所に集約せずに済む。
- **復旧の予行演習をしておく**。意図的にジョブを強制終了させ、新しいワーカー上で復元し、損失、ステップ数、およびデータの読み出し位置が何事もなかったかのように継続することを確認する。スポット価格のインスタンス（39.3 節）は中断のたびに再開できてこそコスト削減につながるため、それに頼る前に復旧をテストしておく必要がある。
- **アクセラレータへのデータ供給を絶やさない**。ストレージからの読み出し、デコードとデータ拡張、トークン化、およびホストからデバイスへの転送といったデータの読み込み処理を、演算とは切り離してプロファイリングする。入力待ちが生じる GPU はスループットが低下し、データ律速のジョブに GPU を追加しても競合が増えるだけで速度は上がらない。
- **速度だけでなく収束も監視する**。トークン処理速度が高くても、損失が下がっているとは限らない。その両方に加えて、集団通信とチェックポイント保存に費やされた時間もログに記

録する。分散実行の障害をデバッグする際は、すべてのログ行にランクとホスト名を出力したうえで、まず 1 プロセス、次に 1 ノード、最後に多数ノードの順で問題を再現させる。

以上の実践によって実行は再開可能になるが、チェックポイント保存と再起動のたびになお時間のコストがかかる。演習問題では、与えられた故障率のもとでそのコストから最適なチェックポイント間隔がどう決まるかを導出する。

39.6.4 まとめ

分散訓練は、バッチ、訓練状態、層、または系列をワーカー間で分割する。実測した単一 GPU のベースラインから出発し、モデル状態がメモリに収まる間は DDP を使い、収まらなくなれば FSDP または ZeRO へ移行し、モデルの規模がそれを必要とする場合に限ってテンソル並列、パイプライン並列、コンテキスト並列、またはエキスパート並列を追加する。その際、通信量の大きいテンソル並列は NVLink ノード内に留める。ピークメモリは異なる項目の和であるため、シャーディング、活性化チェックポイント、勾配累積、ホストオフロード、および LoRA は、通信、再計算、または転送時間という予測可能なコストと引き換えに、それぞれ特定の項目を縮小する。2026 年 11 月時点での一般的な選択肢は、単一 GPU では Unsloth または PEFT、単一ノードでは FSDP2 を用いた Accelerate または Axolotl、マルチノードの事前学習では Megatron Core、TorchTitan、または MaxText、強化学習による事後学習では TRL または veRL である。チェックポイントが完全かつアトミックに書き出され、復旧の予行演習が事前に実施され、データパイプラインが演算に追随し、スループットと並んで収束がログに記録されていれば、数週間にわたる実行も確実に完遂できる。

39.6.5 演習問題

1. **グローバルバッチサイズ。**ある事前学習ジョブでは、長さ 4,096 トークンの系列と、$2^{22} \approx 4.2$ 百万トークン、すなわちオプティマイザの 1 ステップあたり 1,024 系列のグローバルバッチを用いる。各 GPU はマイクロバッチあたり最大 4 系列を保持できる。
   1. $N_{\text{ranks}} \in \{32, 64, 128, 256\}$ に対し、B_{device} と $N_{\text{accumulation}}$（Equation 39.6）を、グローバルバッチが 1,024 系列のまま累積ステップ数が最小になるように選べ。データ並列のみでこのグローバルバッチを訓練できる最大のワールドサイズはいくつか。
   2. 4 系列のマイクロバッチ 1 個にかかる時間を t_{mb} とし、各オプティマイザステップの最後に行う勾配の all-reduce にかかる時間をオーバーラップなしで t_{ar} とする。オプティマイザ 1 ステップあたりの時間と GPU あたりの毎秒トークン数を N_{ranks} の関数として書き下せ。$t_{\text{ar}} = 2t_{\text{mb}}$ のとき、ワールドサイズが 32 から 64 へ、また 128 から 256 へと倍増すると、一定数のトークンを訓練する時間はそれぞれどの程度の比率で短縮されるか。ワールドサイズが大きくなるにつれて高速化を制限するものは何か。
   3. 同僚がジョブを 64 基から 128 基の GPU に移行し、B_{device} と $N_{\text{accumulation}}$ をそのままにしたとする。一定数のトークンに対するグローバルバッチとオプティマイザステップ数はどう変化するか。Adam のような適応的手法に対して、9.8 節では学習率をバッチサイズの平方根に比例して $\eta(b) = \sqrt{b/b_0}\,\eta_0$ とスケーリングする（ここで η_0 はバッチサイズ b_0 で調整された学習率である）。拡大後のバッチをそのまま使う場合、この規則はどの学習率を示唆するか。また、ステップ数で指定されたウォームアップには何が起きるか。
2. [code] **パディングされた評価シャード。**10,001 個のデータ例からなる評価セットを 8 つのランクで評価し、ランク r は `DistributedSampler(dataset, num_replicas=8, rank=r, shuffle=False)` を反復処理するものとする。`num_replicas` と `rank` が明示的に渡されるため、プロセスグループを起動しなくても 1 つのプロセスで全ランクのサンプラを構築できる。
   1. 8 つのサンプラすべてのインデックスを `dataset = list(range(10001))` に対して収集せよ。インデックスは合計いくつあるか、どのデータ例が 2 回現れるか、また重複によって測定される正解率は最大で何パーセントポイントずれるか。

   2. `drop_last=True` として同じ実験を繰り返せ。どのデータ例が一度も評価されないか。
   3. ランク間で 10,001 個のデータ例をパディングなしで分割し、各ランクが自身の正解予測数とデータ例数を数え、`torch.distributed.all_reduce` でそれらのカウントを集約する厳密な評価処理を実装せよ。ランダムな予測を用いて 8 つのランクをシミュレートし、all-reduce をランク間の和に置き換えることで、1 つのプロセス内でこの集約をテストせよ。これらのシャードにおいて、8 つのランクごとの正解率の平均が厳密な値にならないのはなぜか。
3. [code] **再開時の一致性**。再開された実行が正しいといえるのは、中断なしの実行と同じパラメータを計算するときである。CPU 上で小規模な実験を構築せよ。ドロップアウト付きの 2 層の多層パーセプトロンを、20 個の特徴量を持つ 1,024 個のランダムなデータ例上で、`LambdaLR` のウォームアップスケジュールを伴う Adam によって訓練する。データ例は `DataLoader`（バッチサイズ 32、`shuffle=True`）から供給し、そのシャッフルにはエポックごとに専用の `torch.Generator` でシードを与えるものとする。
   1. 中断なしで 64 ステップ訓練せよ。2 回目の実行では、40 ステップ訓練してチェックポイントを保存し、異なるシードからモデル、オプティマイザ、スケジューラ、データローダを再構築したうえでチェックポイントを復元し、残りの 24 ステップを訓練せよ。両方の実行の最終パラメータをビット単位で一致させよ。39.6.3 節に挙げられた各状態は、この実験のどのオブジェクトが保持しているか。また、復元処理はどのようにしてエポックの途中からデータローダを再開させるか。
   2. 復元対象から各状態を 1 つずつ順番に省き、64 ステップ後のパラメータの最大差を報告せよ。損失曲線に現れるのはどの状態の欠落であり、気づかれずに見過ごされるのはどの状態の欠落か。
   3. チェックポイントのバイト列の前半半分だけを最終パスに書き出すことで `torch.save` の最中のクラッシュをシミュレートし、そのファイルを読み込んでみよ。次に、各チェックポイントを同じディレクトリ内の一時ファイルに書き出し、`os.replace` で所定の位置へ移動させよ。2 番目の手順の最中に訓練プロセスがクラッシュしたとき、最終パスに必ず完全なチェックポイント（直前のものまたは新しいもの）が残るのはなぜか。停電が起きると、ファイルの中身より先にリネーム結果だけがディスクに残ることがある。停電にも耐えられるようにするには、この手順にどの呼び出しが必要か。
4. **オプティマイザのオフロード**。●ZeRO-Offload (Ren et al. 2021) は、順伝播と逆伝播のために 16 ビットのパラメータを GPU 上に保持し、FP32 のマスターパラメータと Adam の 2 つのモーメントをホストメモリ上に保持して CPU でパラメータ更新を実行する。各ステップでは、16 ビットの勾配をホストへ送り、更新された 16 ビットのパラメータを送り返す。元論文は BF16 と同じサイズの FP16 を用いている。実効 400 TFLOP/s を維持する 1 基の GPU 上の 7B モデルを考えよう。この GPU は双方向それぞれ 50 GB/s の PCIe で、メモリ帯域幅 200 GB/s のホストに接続されているものとする。
   1. 2 回の転送を続けて実行したときの 1 ステップあたりの転送時間を計算せよ。
   2. CPU 上での更新はメモリトラフィックによって律速される。この配置において Adam の更新が読み書きする 1 パラメータあたりのバイト数を数え、更新にかかる時間を推定せよ。
   3. 順伝播と逆伝播には、トークンあたり約 $6N$ FLOPs（パラメータ数 N のモデルの場合、11.6 節）の計算コストがかかる。何もオーバーラップしない場合、転送と更新の合計時間がステップ時間の 20% 未満に収まるには、1 ステップに少なくとも何トークン含まれていなければならないか。
   4. ZeRO-Offload の 1 ステップ遅延パラメータ更新では、ステップ k の勾配からの更新を CPU 上で計算するのと並行して、GPU がステップ $k+1$ の順伝播と逆伝播を（ステップ $k-1$ の勾配から生成されたパラメータを用いて）実行する。このとき、更新の実質的な所要時間がゼロになるのはどの条件のもとでか。また、この遅延によって最適化の何が変化するか。
5. **少ないワーカー数での再開**。●あるジョブが 64 ランクの FSDP で $B_{\text{device}} = 4$ 系列、

$N_{\text{accumulation}} = 4$ として訓練し、シャーディングされたチェックポイントを書き出していたが、2 ノードを失ったため 48 ランクで再開しなければならなくなったとする。

1. 39.6.3 節に挙げられた再開可能なチェックポイントの内容を、すべてのランクで同一の状態と、ランク間で異なる状態とに分類せよ。ランクごとに異なる各項について、ランク数が変化した際に復元処理が何をしなければならないかを述べよ。
2. 48 ランク上では B_{device} と $N_{\text{accumulation}}$ をどのように選んでも、1,024 系列のグローバルバッチを再現できないことを示せ。異なるグローバルバッチを採用する方法と、ランク間で不均等なマイクロバッチを用いる方法という 2 つの対処法について、それぞれが最適化の何を変化させ、損失の正規化に何を要求するかを比較せよ。
3. 再開後の実行が、任意のランク数に対してそのエポックの残りのデータ例をちょうど 1 回ずつ処理するようなデータ順序を設計せよ。各ランクの読み出し位置をそれぞれのシャードに保存するだけでは不十分なのはなぜか。

6. [extended] **パイプラインスケジュール**。●●p ステージのパイプラインで、オプティマイザの 1 ステップあたり m 個のマイクロバッチを訓練する。1 個のマイクロバッチが 1 つのステージを通過する順伝播には時間 t_f がかかり、逆伝播には $t_b = 2t_f$ がかかる。あるステージは、前のステージがマイクロバッチ j の順伝播を終えればその順伝播を実行でき、次のステージが j の逆伝播を終えれば自身の j の逆伝播を実行できる。各ステージは、順伝播から逆伝播までの間、j の活性化値を保持する。GPipe (Huang et al. 2019) はすべての逆伝播の前にすべての順伝播を実行するのに対し、Narayanan et al. (2021) の 1F1B スケジュールはウォームアップとして数回の順伝播を実行した後、1 回の順伝播と 1 回の逆伝播を交互に繰り返す。スケジュールのバブル率は、ステップ時間から理想時間 $m(t_f + t_b)$ を引いた値を理想時間で割ったものである。
   1. 各ステージでの演算の順序を受け取り、ステージが空きかつ入力が揃い次第ただちに各演算を実行して、ステップ時間と、各ステージが同時に活性化値を保持するマイクロバッチの最大数とを返すシミュレータを作成せよ。
   2. 任意の p と m に対して GPipe と 1F1B の演算順序を生成せよ。その際、1F1B の各ステージが必要とするウォームアップ順伝播の回数を導出せよ。これらを $p \in \{2, 4, 8\}$ および $m \in \{1, 4, 8, 32\}$ に対して実行し、その結果から各スケジュールのバブル率と活性化メモリを p と m の関数として導出せよ。
   3. Narayanan et al. (2021) のインターリーブ割り当てでは、各デバイスが連続しない v 個の層のチャンクを保持する。このとき、1 つのマイクロバッチは pv 個のステージを通過し、すべてのデバイスを v 回ずつ、それぞれ $1/v$ のコストで訪問する。すべての順伝播を先に行う方式で、$v \in \{1, 2, 4\}$ についてこれをシミュレートせよ。バブル率と 1 ステップあたりのポイントツーポイント転送回数は v にどう依存するか。
   4. パイプラインの外側に並列度 d のデータ並列を組み合わせ、マイクロバッチを b 系列とすると、1,024 系列のグローバルバッチ（演習問題 1）のもとで $m = 1024/(db)$ となる。$p = 8$、$b = 1$ のとき、$d = 8$、16、64 における 1F1B のバブル率を計算せよ。グローバルバッチを固定したとき、パイプライン化されたジョブのデータ並列度を制限するものは何か。

39.7 モデルサービング

本書で扱ってきた言語モデルは、セルを実行した 1 人のために、ノートブック内でプロンプトを 1 つずつ処理してテキストを生成していた。では代わりに、100 人の同僚が同じ言語モデルへ好きなタイミングでリクエストを送るとしよう。もし GPU がそれらに 1 件あたり 2 秒ずつ順番に応答すれば、最後の同僚は 3 分以上待つことになる。モデルを呼び出すこととサービングすることの間では、何を変えなければならないのだろうか。

サービングとは、1 基のアクセラレータに多数のリクエストを同時に処理させる仕組みである。1 トークンを生成するには、重みとキー・値キャッシュをメモリから読み出す（39.4 節）。すべての

リクエストは重みを共有するため、メモリがアクティブな全リクエストのキャッシュを保持できる限り、多数のリクエストを同じステップで進めるエンジンは全員分の重みを 1 回読み出すだけで済む。本節ではまずユーザーが体感するレイテンシの数値から始め、2026 年 11 月時点におけるローカル用およびサーバー用のエンジンを概観したうえで、エンジンを高速化する技法を解説する。

39.7.1 ワークロードとレイテンシ指標

チャットアシスタントのユーザーは、最初の単語が現れるまでの待ち時間と、それに続く単語が表示される速度という 2 つの遅れに気づく。1 つ目はリクエストの到着から計測される**最初のトークンまでの時間**（time to first token、TTFT）である。アイドル状態のサーバーでは、これはすべてのプロンプトトークンを並列に処理し計算律速（compute-bound）となる**プレフィル**に支配される。2 つ目は**出力トークンあたりの時間**（time per output token、TPOT）であり、系列ごと・ステップごとに 1 トークンを生成しメモリ帯域幅律速（memory-bandwidth bound）となる**デコード**に支配される。全ユーザーについて合計した毎秒トークン数であるスループットは、これら両方とトレードオフの関係にある。

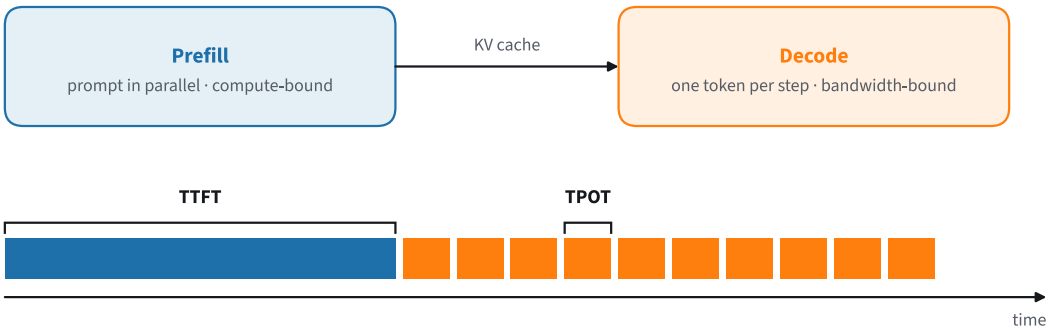


Figure 39.14: プロンプトのプレフィルは並列処理であり計算負荷が高くなりやすい。一方、自己回帰デコードは系列ごとに 1 トークンずつ進み、メモリ転送に負荷をかける。

どの数値が重要になるかはワークロード次第であり、ワークロードによって選ぶべきツールが決まる。

Table 39.5: サービングのワークロードと出発点の例（2026 年 11 月時点）

ワークロード	最適化対象	出発点
手元のマシン上の個人用アシスタント	簡便さ、プライバシー	Ollama / LM Studio（llama.cpp）、Mac では MLX
オフラインのバッチスコアリング	ドルあたりのスループット	vLLM オフラインモード
単一アプリケーションの API	運用性	軽量な認証プロキシ配下の vLLM
マルチユーザー / エージェント型サービス	SLO 下でのグッドプット	vLLM または SGLang、大規模化時は NVIDIA Dynamo
固定された NVIDIA 環境での最大スループット	GPU あたりのスループット	TensorRT-LLM

サービスレベル目標（service level objective、SLO）は、TTFT を 1 s 未満にするといったレイテンシ目標を定める。目標も満たした完了リクエストのレートである**グッドプット**こそが、本番環境における評価指標である。過負荷状態では、リクエストがキューに滞留して目標を逃すようになるため、生のスループットが上がってもグッドプットは崩壊しうる。

39.7.2 ローカルサービング

ユーザーが 1 人のノート PC やワークステーションでは、モデルがメモリに収まるか、そしてどれだけ速く応答するかが問題となる。多くのローカル推論ツールは **llama.cpp** とその **GGUF** ファイル形式を採用しており、コミュニティのリポジトリではオープンウェイトモデルの GGUF 変換版が広く公開されている。ユーザーはラッパを介して llama.cpp を利用することが多い。

- **Ollama** は llama.cpp を基盤とし、Apple シリコン上では MLX も利用する。`ollama run qwen3:8b` を実行するとモデルのダウンロード、キャッシュ、サービングが一括で行われ、`http://localhost:11434/v1` で OpenAI 互換 API が提供される。
- **LM Studio** は同じエンジンをグラフィカルインターフェース経由で提供し、モデルのどの程度を GPU にオフロードするかを制御できる。
- **MLX** は Apple の配列フレームワークである。パッケージ `mlx-lm` によって Apple シリコン上でモデルの実行とファインチューニングが可能であり、Hugging Face 上の `mlx-community` オーガニゼーションが数千もの変換済みモデルを公開している。

代表的な GGUF の量子化形式には、`Q4_K_M`（重みあたり約 4.9 ビット）、`Q5_K_M`（約 5.7 ビット）、`Q8_0`（約 8.5 ビット）がある。Qwen3-8B の 82 億個のパラメータに対して `Q4_K_M` を用いると $8.2 \times 10^9 \times 4.9/8 \approx 5.0$ GB となり、Ollama の `qwen3:8b` のダウンロードサイズである 5.2 GB に近い。この 4.9 ビットという値は Llama 3.1 8B で計測されたものであり、`Q4_K_M` はモデルに応じた割合のテンソルをより高い精度で保持する。最大 32,768 トークンのコンテキスト全体を bf16 でキャッシュすると、ほぼ同程度の容量がさらに加わる。一般にビット数の多い形式ほど品質はよく保たれるため、自身のタスクで影響を計測するとよい。デコード速度は高々、メモリ帯域幅をトークンあたりの読み出しバイト数で割った値になるため、ビット数を増やすごとにメモリだけでなく速度も犠牲になる。定石は、キャッシュ用の余裕を残して量子化後の重みがメモリに収まる最大のモデルを選び、次にその範囲内で収まる最もビット数の高い形式を選ぶことである。

39.7.3 サーバーエンジン

39.7.3.1 オープンソースエンジンと NVIDIA エンジン

共有サーバーには、複数のリクエストをまとめてスケジューリングするエンジンが必要である。オープンソースの GPU サーバー向けには、**vLLM** が連続バッチング（continuous batching）、ページド KV キャッシュ管理、プレフィックスキャッシュ、テンソル並列（tensor parallelism）、量子化モデルのサポート、および OpenAI 互換サーバーを 1 つのコマンドで提供する。

```
vllm serve Qwen/Qwen3-8B --max-model-len 32768
```

SGLang はこれに最も近い競合エンジンである。その **RadixAttention** (Zheng et al. 2024) は、キャッシュ済みのキーと値をリクエスト間で共有される基数木（radix tree）に保持するため、システムプロンプトを伴うチャット、同じ文書群に対する検索、あるいは木状に広がるエージェント呼び出しのように、多数のリクエストが長いプレフィックスを共有する場面で威力を発揮する。両プロジェクトとも急速に進化しており、複数のアクセラレータバックエンドをサポートしているため、プロンプト長と同時接続数の実際の分布で両者をベンチマークしてから選ぶべきである。

NVIDIA のツール群は、移植性と引き換えにハードウェア固有の最適化を追求する。**TensorRT-LLM** は NVIDIA がチューニングしたカーネルを用いて PyTorch 上でモデルを実行し、セットアップの手間が増える代わりに定常状態のスループットを高められる。**Dynamo** は vLLM、SGLang、または TensorRT-LLM をバックエンドとして多数の GPU とノードを協調動作させる。プレフィルとデコードを別々の GPU プール上で実行し、ワーカーの負荷やプレフィックスがすでにどれだけキャッシュされているかに応じて各リクエストをルーティングする。これらのツールが重要

になるのはデータセンタ規模においてであり、数基の GPU であれば vLLM や SGLang 単体のほうが簡潔である。

39.7.3.2 共通のクライアント API

ノート PC からサーバーへ移行する際に、アプリケーションの書き直しが必要になるべきではない。これらのツールの多くは OpenAI の Chat Completions API を実装しているため、ベース URL とモデル名を変更するだけで、同じクライアントコードからローカルの Ollama、自前の vLLM サーバー、あるいは商用プロバイダを利用できる。

```
from openai import OpenAI

client = OpenAI(base_url="http://127.0.0.1:8000/v1", api_key="local")
response = client.chat.completions.create(
    model="Qwen/Qwen3-8B",
    messages=[{"role": "user", "content": "Explain KV caching briefly."}],
    temperature=0,
)
print(response.choices[0].message.content)
```

もっとも、同じリクエストを受け付けるサーバーであっても応答の挙動が同じとは限らない。トークン化、サンプリングのデフォルト設定、構造化出力のサポート、ストリーミングの挙動、トークン数の集計方法はいずれもサーバーごとに異なる。小規模な適合性テスト（conformance test）を用意しておき、エンジンを入れ替える前に再実行するとよい。

39.7.4 エンジンの技法

組み合わせ可能な 4 つの技法によって、これらのエンジンの速さの大部分を説明できる。

連続バッチング。静的バッチは最も遅いリクエストの完了を待つ。これに対し連続バッチングは、デコードステップの合間に新しいリクエストを受け入れ、完了したリクエストを退出させることで、バッチをほぼ満杯に保つ。

以下の簡易スケジューラは、この仕組みを数行で表現したものである。演習問題 4 では、これに到着時刻とメモリバジェットを加えて拡張する。

```
from collections import deque

requests = deque([5, 2, 7, 3])  # output tokens requested
capacity = 2
active, timeline = [], []
while requests or active:
    while requests and len(active) < capacity:
        active.append(requests.popleft())
    timeline.append(tuple(active))
    active = [remaining - 1 for remaining in active if remaining > 1]

timeline
```

```
[(5, 2), (4, 1), (3, 7), (2, 6), (1, 5), (4, 3), (3, 2), (2, 1), (1,)]
```

ページド KV キャッシュ。アクティブな各系列のキー・値キャッシュは、系列長とともに増大する。vLLM の PagedAttention (Kwon et al. 2023) は仮想メモリと同様にこれを固定サイズのブ

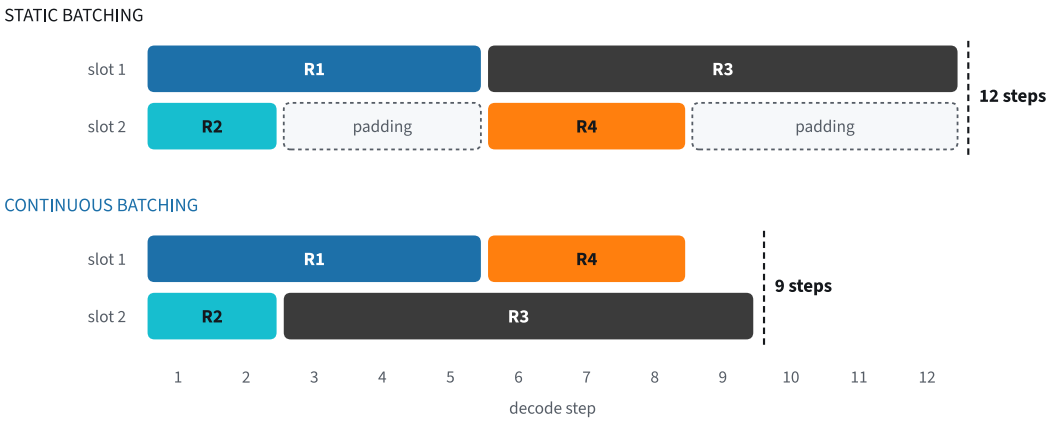


Figure 39.15: 下の簡易スケジューラと同様に、出力トークン数が 5、2、7、3 のリクエストを 2 つのバッチスロットで処理する例。静的バッチ化は各バッチを最長の要素に合わせてパディングするが、連続バッチングは完了したスロットを次のステップで補充し、12 ステップではなく 9 ステップで完了する。

ロック単位で割り当てることで、かつてバッチサイズの上限となっていた断片化をほぼ解消する。ここから容量設計の原則が導かれる。すなわち、GPU メモリには重みに**加えて**同時実行される全系列のキャッシュが収まらなければならない。したがって、1 件のリクエストなら収まるモデルであっても、高い同時接続数の下ではメモリを超過しうる。

プレフィックスキャッシュ。2 つのリクエストがトークン化後のプレフィックス（同じシステムプロンプトや同じ文書）を共有している場合、そのキャッシュされたキーと値は 1 回計算するだけで再利用できる。

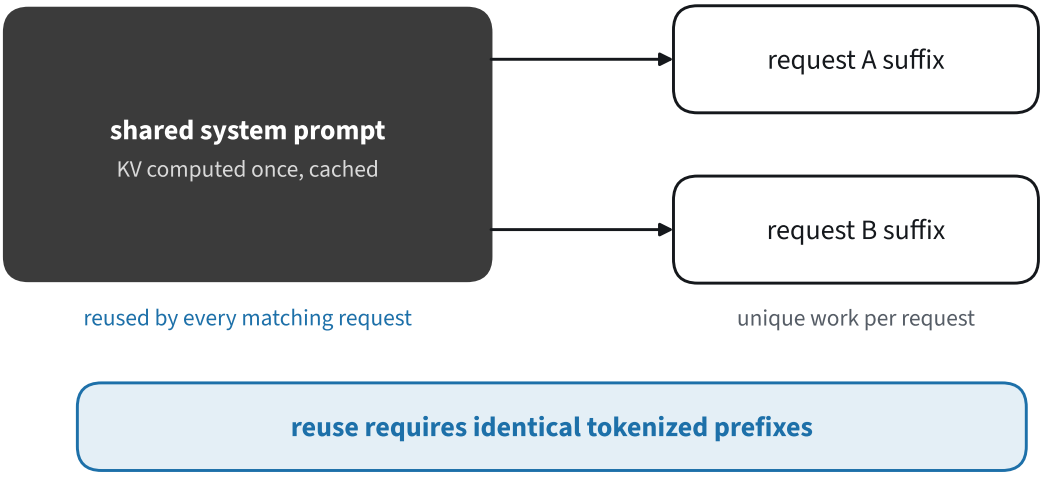


Figure 39.16: プレフィックスキャッシュはトークン化後のプレフィックスが一致する場合にのみ KV 状態を再利用し、その後で各リクエスト固有のサフィックスを計算する。

商用 API のなかには、プロバイダごとに異なる規則の下で、キャッシュされた入力トークンの料金を割り引くものがある。そのため、プロンプトの可変部分より前に共有される固定的な内容を

配置すれば、プレフィル時間だけでなく費用も節約できる。

投機的デコーディングと量子化。投機的デコーディング（speculative decoding）では、小規模なドラフトモデルが複数のトークンを提案し、ターゲットモデルが 1 回の並列パスでそれらを検証する。提案の大半が受理される場合には、ターゲット分布を保ったままデコーディングを高速化できる。一方、量子化はデコード時に読み出すバイト数を削減する。重みあたりのバイト数が減れば毎秒トークン数は増えるが、極端に低いビット幅では品質が劣化する。近年の NVIDIA テンソルコアは FP8（Hopper および Ada 世代以降）と NVFP4（Blackwell 以降）をサポートしており、vLLM や SGLang は 4 ビットの AWQ 重みも読み込める。ローカル環境では GGUF 形式が同じ役割を果たす。

39.7.5 運用

ポートを待ち受けているだけのエンジンには、共有サービスに必要な制御機構がまだ備わっていない。エンジンの前段にプロキシを置いて認証、TLS、レート制限を担わせ、エンジン自体のポートは外部に公開しないようにする。受付制御（admission control）では、メモリ不足（out-of-memory）エラーが起きた後ではなく、メモリが枯渇する**前**にリクエストを拒否またはキューイングすべきである。クライアントからのキャンセルを伝播させ、放棄された生成処理が計算資源を消費し続けないようにする。TTFT、TPOT、およびエンドツーエンドのレイテンシは、閉ループのクライアント群による平均値ではなく、現実的な到着パターンの下での p50 と p95 として計測する。デフォルトでは生のプロンプトをログに残さないようにする。さらに、モデル、トークナイザ、量子化、エンジンのバージョンを一括でピン留めする。より高速でも異なる応答を返すサーバーは別物のシステムであるため、ロールアウトの可否はレイテンシだけでなくタスク品質でも判定しなければならない。

ここまでで、モデルとユーザーのキャッシュが 1 枚のカードに収まるか、そして最初にどのエンジンを試すべきかを見積もれるようになった。負荷の下でサーバーがどのようにリクエストを受け入れ、順序付けるべきかという問いはまだ残されており、演習問題のシミュレータでこれを探究する。

39.7.6 まとめ

サービングのレイテンシは、最初のトークンまでの時間（TTFT）を決める計算律速のプレフィルと、出力トークンあたりの時間（TPOT）を決める帯域幅律速のデコードに分かれる。ローカルでの Ollama、LM Studio、MLX から、共有サービス向けの vLLM や SGLang、さらには大規模な NVIDIA 環境にわたる TensorRT-LLM と Dynamo の組み合わせまで、ワークロードに合わせてエンジンを選ぶ必要がある。ローカルの GGUF 形式やサーバーの AWQ、FP8、NVFP4 による量子化は、極端に低いビット幅で品質が劣化するまでは、削減した重みのバイト数をそのまま高速なトークン生成へと変える。現代のエンジンは連続バッチング、ページド KV キャッシュ、プレフィックスキャッシュ、投機的デコーディングを組み合わせており、クライアント側のプロンプトの並べ方によって再利用できるプレフィックス状態の量が決まる。OpenAI 互換の Chat Completions API によりローカルとクラウドのサーバー間でクライアントコードの移植性は保たれるものの、細かな挙動の違いがあるため、エンジンを切り替える前には小規模な適合性テストを行うべきである。最後に、メモリ容量にはモデルの重みと同時接続する全ユーザーの KV キャッシュがともに収まらなければならず、本番環境での運用には受付制御、クライアントのキャンセル伝播、テールレイテンシの監視、およびモデルとエンジンのバージョンの一括ピン留めが不可欠である。

39.7.7 演習問題

1. **バッチサイズの上限。** コマンド `vllm serve Qwen/Qwen3-8B` は、$L = 36$ 層、次元 $d_h = 128$ のキー・値ヘッド $H_{\text{kv}} = 8$ 個をもち、bf16 の重みが $B_w = 16.4$ GB を占めるモデルをサービングする。キャッシュも同様に bf16 で保存される。これをメモリ帯域幅が $\beta = 3.35$ TB/s、密な bf16 演算レートが 989 TFLOP/s（Table 13.2）である H100 SXM 上でサービングせよ。1 回のデコードステップでは、バッチ全体に対して重みを 1 回読み出し、アクティブな全系列のキャッシュを読み出す。それぞれが n 個のキャッシュ済みトークンを保持する b 本の系列に対し、このステップ（すなわち各系列の出力トークンあたりの時間）にかかる時間はおよそ次の式で与えられる。

$$t(b) = \frac{B_w + b\,n\,B_{\text{kv}}}{\beta}, \tag{39.7}$$

ここで、B_{kv} はトークンあたりのキャッシュサイズである。

   1. 各層はキー・値ヘッドごと、トークンごとに 1 つのキーベクトルと 1 つの値ベクトルを保存するため、$B_{\text{kv}} = 2 \cdot L \cdot H_{\text{kv}} \cdot d_h \cdot (\text{bytes per element})$ となる（11.3 節）。B_{kv} を計算せよ。次に、TPOT とスループット $b/t(b)$ を、$n = 2048$ のもとで $b \in \{1, 8, 64, 256\}$ について評価せよ。b が大きくなるときスループットはどの値に近づき、$t(b)$ のどの項がそれを決めるか。また、82 億個のパラメータに対してパラメータ・トークンあたり 2 FLOPs として計算し、bf16 演算レートと照らし合わせて、$b = 256$ におけるステップが依然としてメモリ転送によって律速されていることを確かめよ。
   2. ある対話型サービスが TPOT を最大 20 ms 以内に抑えると約束している。実行できる最大のバッチサイズはいくらか。これに対し、Table 39.5 のオフラインスコアリングジョブであればどのバッチサイズを選ぶか。また、それはなぜか。
   3. エンジンがカードのメモリのうち 72 GB を重みとキャッシュに使用し、各系列が平均 2,048 トークンを保持するとする。ページドキャッシュ以前は、サーバーはリクエストごとにその最大長分の連続領域を 1 つ予約していた。各系列が `--max-model-len` である 32,768 トークン分を予約するとき、何本の系列が収まるか。またそのとき、予約されたキャッシュのうち実際にトークン状態を保持している割合はいくらか。Kwon et al. (2023) はそのようなシステムで 20.4%〜38.2% という値を計測しているが、ここで求めた割合がそれより低くなるのはなぜか。さらに、キャッシュを 16 トークンのブロック単位で割り当てる場合には何本の系列が収まるか。部分的にしか埋まっていないブロックが無駄にするメモリ量の上界を求めよ。
   4. 2 つの割り当て方式のそれぞれにおいて、対話型サービスのバッチサイズを決める上限はメモリと TPOT 目標のどちらか。

2. [code] **適合性テスト。** 39.7.3.2 節では、エンジンを入れ替える前に小規模な適合性テストを行うことを勧めている。

   1. 関数 `conformance(base_url, model)` を 39.7.3.2 節の `openai` クライアントを用いて作成し、4 つのチェックを実行してどれが合格したかを報告するようにせよ。すべてのチェックにおいて、`content` が空でないことも要件とする。第 1 に、ソフトマックス温度 0 での 2 回の同一リクエストが同じテキストを返すこと。第 2 に、`max_tokens=8` を指定したリクエストが `finish_reason` を `"length"` として終了し、`usage` 内で報告される生成トークン数が多くても 8 個であること。第 3 に、1 から 20 まで数えるようモデルに求めるリクエストを `stop=["7"]` 付きで送ったとき、数字 7 を含まないテキストを返すこと。第 4 に、温度 0 かつ `stream_options={"include_usage": True}` を指定したストリーミングリクエストで得られるチャンクについて、連結した内容と最終的な `usage` がストリーミングなしの同一リクエストと一致すること。CPU 上で動作する `qwen3:0.6b` などの小規模モデルを用いた Ollama と、任意に選んだもう 1 つの

OpenAI 互換サーバーに対してこのテストを実行せよ。どのチェックが失敗し、それはなぜか。

2. vLLM サーバーでは、第 1 のチェックはサーバーがアイドル状態の間は合格するが、他のクライアントがリクエストを送っている間はときおり失敗する。温度 0 ではサンプリングは行われない。この失敗の理由を説明せよ。
3. 両方のサーバーが同じモデルと同じトークナイザを読み込み、同じメッセージを受け取っているにもかかわらず、`usage.prompt_tokens` として数トークン異なる値を報告することがある。考えられる原因を 2 つ挙げよ。また、そのような差異が課金以外の面でクライアントにとって問題となるのはなぜか。

3. [code] **エージェント向けのプレフィックスキャッシュ。**あるコーディングエージェントがモデルを 30 回呼び出す。最初のプロンプトは 20,000 トークンのシステムプロンプトとリポジトリコンテキストを保持している。以降のすべてのプロンプトは直前のプロンプトをそのまま繰り返し、モデルの前回の回答とツールの実行結果からなる 1,000 トークンを末尾に追加する。

   1. 30 回の呼び出し全体でエンジンがプレフィルしなければならないプロンプトトークン数は、プレフィックスキャッシュなしの場合と、過去のすべてのプロンプトを保持するキャッシュがある場合とでそれぞれいくつになるか。あるプロバイダがキャッシュ済みトークンに対して通常の入力料金の 10 分の 1 を課金するとき、キャッシュなしの場合の入力料金に対して支払額はどの割合になるか。
   2. vLLM は完全に埋まったブロックのみをキャッシュし、先行ブロックのキーとそのブロック自身のトークン ID 列のハッシュによって各ブロックのキーを作る。この方式を 16 トークンのブロックで関数 `prefill_tokens(cache, tokens)` として実装し、計算すべきトークン数を返して新しい完全なブロックを集合 `cache` に追加するようにせよ。ランダムなトークン ID を用いてエージェントをシミュレーションし、第 1 小問で求めた値と結果を比較せよ。ブロックのキーに先行ブロックのキーを含めるのはなぜか。
   3. エージェントフレームワークが秒単位の現在時刻をシステムプロンプトの 1 行目に書き込むとする。シミュレーションを再実行せよ。タイムスタンプは代わりにどこへ配置すべきか。
   4. ある検索アプリケーションが、共有のシステムプロンプトの後に 50 文書のうち 3 文書を関連度順に配置する。2 つのリクエストが同じ 3 文書を異なる順序で検索したとき、2 つ目のリクエストはどのブロックを再利用するか。再利用を高める並べ方を提案し、それによって何が犠牲になるかを述べよ。

4. [code] **リクエストの到着とメモリバジェット。**●`timeline` を構築するセルを拡張し、時間がデコードステップ単位で進むシミュレータを作成せよ。リクエスト i はステップ a_i に到着し、P_i トークンのプロンプトをもち、受け入れられたステップ以降、毎ステップ 1 トークンずつ合計 T_i トークンを生成する。サーバーは最大 64 本の系列を実行し、最大 32,768 トークン分のキャッシュを保持し、各系列はそのプロンプトとそれまでに生成されたトークンを保持する。到着はステップあたり λ 件のポアソン過程から抽出し、P_i は 100 から 1,000 までの一様分布から抽出し、T_i は 1 に平均 200 の指数確率変数を加えて小数点以下を切り捨て、`max_tokens = 1024` を上限とした値とせよ。最初のトークンまでの時間を、到着から最初に生成されたトークンまでのステップ数として計測せよ。

   1. リクエストが必要としうる最大量であるプロンプト長＋ `max_tokens` をキャッシュに収められるときに、先着順でリクエストを受け入れよ。すでに 50 件のリクエストが待機している場合は、到着したリクエストを拒否せよ。$\lambda \in \{0.05, 0.1, 0.15, 0.2\}$ について、ステップあたりの生成トークン数、TTFT の p50 と p95、および拒否率を報告せよ。
   2. 代わりに、プロンプト長＋ 1 トークンをキャッシュに収められるならいつでもリクエストを受け入れよ。アクティブな系列が成長してキャッシュを超過したときは、最も

新しく受け入れられた系列をプリエンプトしてそのキャッシュを解放し、再受け入れ時にキャッシュを再計算するようキューの先頭に戻せ。同じ計測を繰り返せ。このシミュレータではプレフィルの所要時間を 0 としている。実際のサーバーでは、再計算はどの資源を消費し、代わりにキャッシュを CPU メモリへスワップする場合はどの資源を消費するか。プレフィルの所要時間を考慮に入れると、第 1 小問との比較結果はどのように変わるか。

3. 残りトークン数の少ない順にキューを並べ替え、$\lambda = 0.2$ で両方の受け入れ規則を実行せよ。全リクエストの p95 TTFT と、出力が最も長い上位 5% のリクエストの p95 TTFT について、この順序付けを先着順と比較せよ。
4. サーバーは `max_tokens` は知っているが T_i は知らない。代わりに何によってキューを順序付けできるか。また、第 3 小問で得られた改善効果のうちどの程度を維持できると予想されるか。

5. **投機的サンプリング。**●Leviathan et al. (2023) の投機的デコーディングでは、次トークン分布 q をもつドラフトモデルがトークン $x \sim q$ を提案し、次トークン分布 p をもつターゲットモデルがそれを確率 $\min(1, p(x)/q(x))$ で受理する。棄却された場合、トークンは次の残差分布から代わりに抽出される。

$$p'(x) = \frac{\max(0, p(x) - q(x))}{\sum_{x'} \max(0, p(x') - q(x'))}. \tag{39.8}$$

   1. 出力されるトークンが p に従うこと、および受理確率が $\beta = \sum_x \min(p(x), q(x))$ となることを示せ。
   2. 各ラウンドにおいて、ドラフトモデルは γ 個のトークンを順次に提案し、ターゲットモデルの 1 回のパスがそれらすべてにスコアを付ける。ターゲットモデルは最初の棄却までの受理されたトークンを出力し、続けて自身のトークンを 1 つ出力する（棄却後は p' から抽出し、p からは全 γ トークンが受理されたときに抽出する）。α を、第 1 小問の受理確率 β を系列の各位置にわたって平均した値とし、各ドラフトトークンが独立に確率 α で受理されると仮定する。1 ラウンドで出力されるトークン数の期待値が $(1 - \alpha^{\gamma+1})/(1 - \alpha)$ となることを示せ。ドラフトの 1 ステップにはターゲットの 1 ステップの c 倍の時間がかかり、スコアリングのパスにはターゲットの 1 ステップ分の時間がかかるとする。通常のデコーディングに対する期待高速化率を導出し、最適な γ を $c = 0.05$ および $\alpha \in \{0.6, 0.8, 0.9\}$ について求めよ。
   3. スコアリングのパスがターゲットの 1 ステップ分の時間で済むのは、メモリ転送によって律速されている間だけである。サーバーが b 本の系列をまとめてデコードし、各系列がラウンドあたり $\gamma + 1$ 個の位置にスコアを付けるとする。キャッシュは無視し、重みは bf16 で保存し、パラメータ・トークンあたり 2 FLOPs として数えよ。b がいくつを超えると、演習問題 1 の H100 上で $\gamma = 4$ のときにこのパスが計算律速になるか。このことは、単一ユーザーの場合と比べて、高負荷サーバーにおける投機的デコーディングに何を意味するか。

6. **[extended] 負荷下でのグッドプット。**●●演習問題 4 のシミュレータを、時計がデコードステップではなく秒単位で進むものへと拡張せよ。対象は、演習問題 1 の H100（重みとページドキャッシュ用に 72 GB を使用）上で稼働する Qwen3-8B とする。1 ステップでは、p 個のプロンプトトークンと、デコード中の b 本の系列それぞれの 1 トークンを処理し、これらの系列は合わせて K 個のキャッシュ済みトークンを保持している。パラメータ数 N、密な bf16 演算レート F、および演習問題 1 の記法を用いると、このステップにかかる時間は次の式となる。

$$t_{\text{step}} = \max\left(\frac{2N(p + b)}{F}, \frac{B_w + K B_{\text{kv}}}{\beta} \right), \tag{39.9}$$

ただし、ここではアテンションスコアの FLOPs は無視している。リクエストは毎秒 λ 件のポアソン過程として到着し、プロンプト長は 500 から 4,000 トークンの一様分布に従い、出力長は 1 に平均 250 の指数確率変数を加えて小数点以下を切り捨て、2,048 を上限とした値とする。リクエストの TTFT が 1 s 以下であり、かつ出力トークン間の平均時間である TPOT が 50 ms 以下であるとき、そのリクエストをグッドプットに算入する。

1. 受け入れられたすべてのプロンプトを 1 ステップ内で完全にプレフィルせよ。シミュレーションを行う前に、t_{step} から飽和レートを見積もれ。次に、λ を軽負荷から飽和を十分に超える領域までスイープし、スループットとグッドプットを毎秒リクエスト数でプロットせよ。グッドプットがピークに達するのはどのレートであり、事前の見積もりと比べてどうなっているか。
2. 長いプロンプトを完全にプレフィルすると、デコード中のすべての系列で次のトークンが遅延する。ステップあたりのトークン数（デコードトークンを含む）をバジェット以下に制限し、1 つのプロンプトを複数ステップに分割するチャンク化プレフィル (Agrawal et al. 2024) を実装せよ。512、2,048、8,192 トークンの各バジェットについてスイープを繰り返せ。グッドプットを最大化するバジェットはどれか。また、その答えは 2 つの SLO にどのように依存するか。
3. TTFT の目標をもはや満たせないと判明した時点でリクエストを拒否する受付制御を追加せよ。飽和を超えた領域でグッドプットはどう変化するか。受付制御がない場合に、スループットが高いままグッドプットが崩壊する理由を説明せよ。
4. すべてのクライアントが 10 s 後にリクエストを放棄して 1 回だけ再送するものとせよ。放棄されたリクエストをサーバーが生成し続ける場合とキャンセルする場合とで、グッドプットを比較せよ。
5. 拒否されたリクエストも未達成として数えたとき、到着する全リクエストの少なくとも 90% が両方の SLO を満たす最大のリクエストレートと、それを達成する設定を述べよ。

39.8 貢献者ガイド

ここまで本書のノートブックを数多く実行してくるなかで、意味の取りにくい文や、数式中で入れ替わった添字、あるいは新しいバージョンのライブラリで失敗するセルを見つけたことがあるかもしれない。修正は、その文章やコードが現れるすべての場所、すなわちウェブサイト、PDF、各フレームワークのノートブック、そして同じセルを引用するスライドのすべてに反映されるべきである。では、たった 1 行の修正がどのようにしてこれらすべての場所に行き渡り、何も壊していないことをどうやって確認できるのだろうか。

本書はオープンソースソフトウェアと同じ方法で保守されている。すべてのページはバージョン管理下の 1 つの Markdown ファイルを元にしており、ビルドシステムがそれらのファイルから他のすべてを生成する。変更が関係するノートブックが正常に実行され、サイトがレンダリングできれば、レビューの準備が整う。本節では、ビルドの仕組み、変更を提案するための Git ワークフロー、任意で利用できるコーディングエージェント、そして変更が通過すべきチェックについて解説する。同様の課題は、ドキュメント自体が実行可能でなければならないあらゆる機械学習リポジトリで生じる。

39.8.1 ビルドの仕組み

39.8.1.1 1 つのソースと多数のアーティファクト

各節は単一の Markdown ファイルに格納されており、本節のファイルは `chapter_appendix-tools-for-deep-learning/developers-guide.md` である。ビルドでは、これらのソースから他のすべてを派生させる。

```
chapter_*/*.md              source of truth (edit these)
 ├──── chapter_*/*.qmd        generated Quarto input → HTML site and PDFs
 ├──── _notebooks/<fw>/...    generated Jupyter notebooks, one set per framework
 ├──── _slides/<fw>/...       generated lecture slides
 └──── d2l/*.py               generated d2l library, from the #@save blocks
outputs/<fw>/...            executed outputs, captured from _notebooks/, committed
```

この設計からは、実用上 3 つの帰結が得られる。

- **生成されたファイルは決して編集しない。** .qmd や .ipynb への編集は次のビルドで消去されてしまう。ソースである .md を編集して再生成すること。
- **フレームワークのタブは 1 つのファイルに同居する。** %%tab pytorch のタグが付いたコードブロックは PyTorch ビューにのみ現れ、タグのない Python ブロックはそのファイルから生成されるすべてのノートブックに現れる。ファイル内に少なくとも 1 つのブロックでタグ付けされたフレームワークごとに 1 つのノートブックが生成され、タグ付きブロックを持たないファイルからは PyTorch のノートブックのみが生成される。したがって、1 つのソースファイルから最大 4 つのノートブックが生成され、そのすべてが正常に実行できなければならない。「基礎」、「画像モデル」、「深層学習のための数学」、「深層学習のためのツール」の各パートの章は 4 つのフレームワークすべてに対応しており、「発展」パートの章は PyTorch と JAX に（ただし第 14 章は PyTorch のみ）、「言語モデル」パートは PyTorch に対応している。
- **セルの識別子は不変である。** すべての Python ブロックは（tools/add_cell_ids.py によって割り当てられる）永続的な ID を持つため、ファイル名の変更や移動があっても、スライド、実行済み出力、およびフレームワークごとのビューから同一のセルを参照できる。

ソースがノートブックではなく Markdown である理由は、バージョン管理にある。.ipynb ファイルはコード、出力、実行回数、エディタのメタデータが混在する JSON であるため、1 語を修正して再実行するだけで、その語以外の多くの行まで変化してしまう。一方、Markdown の diff には変更された本文とコードしか現れない。実行済み出力は別個に保存され、それを生成したコード、そのコードが使用するライブラリのコード、あるいはフレームワークのバージョンが変更されたときにのみ再取り込みされる。

本書全体を通じてインポートしてきた d2l ライブラリも同じ仕組みで生成されている。make lib は #@save が付いたすべてのブロックを抽出し、フレームワークごとのモジュールである d2l/torch.py、d2l/tensorflow.py、d2l/jax.py、d2l/mxnet.py に書き出す。関数を解説している箇所で加えた修正は、再ビルドによって後続のすべての章に行き渡る。これらのファイルの先頭に直接編集しないよう記されたバナーがあるのはそのためである。39.2 節の Colab および Kaggle 用ノートブックも同様に、独立した notebooks ブランチ上に生成される。このブランチはソースから再ビルドされるものであり、手作業で編集することは決してない。

39.8.1.2 リポジトリの構成

リポジトリのトップレベルには、ソース、ビルドのエントリポイント、およびそれらを説明するドキュメントが置かれている。

```
d2l-neu/
 ├──── chapter_*/          source Markdown, one directory per chapter
 ├──── img/                figures; illustrative SVGs come from tools/gen_*_figures.py
 ├──── d2l/                the generated d2l library
 ├──── outputs/            committed store of executed notebook outputs
 ├──── tools/              build scripts (preprocessing, notebooks, slides, audits)
```

```
├──── docs/                build-system documentation, writing and code guides
├──── Makefile, make/      entry point: make html, make lib, make capture-outputs ...
├──── bootstrap.sh         one-time setup of a fresh clone
└──── CLAUDE.md            build summary, written for coding agents
```

新規クローン直後に `./bootstrap.sh` を実行すると、コミット済み出力の画像を保持する Git LFS のインストールとその画像の取得が行われ、Quarto を含む環境 `.venv-build` が作成される。その後は、GPU を持たないノート PC 上でも `make html` によってコミット済み出力からサイト全体をレンダリングできる。ノートブックの実行はこれとは別のステップである。Makefile の実行には GNU Make 4.3 以降が必要であり、macOS では追加インストール（たとえば MacPorts の `gmake`）としてのみ提供されている。ターゲットの一覧は `make help` で確認でき、詳細は `docs/build-system.md` で解説されている。

39.8.2 Git の基本

本書へのあらゆる変更は、各自が作成したリポジトリのコピー上のブランチにおける Git コミットとして扱われる。日常的に Git を使っている読者は先へ進んでよい。まず、そのコピーを作成する。リポジトリページで **Fork** をクリックし、フォークしたリポジトリをクローンして変更用のブランチを作成する。

```
git clone https://github.com/<your-username>/d2l-neu.git
cd d2l-neu
git switch -c fix-typo-in-attention    # a branch per change
```

ソースの `.md` ファイルを編集したら、変更を記録して公開する。

```
git status                  # what changed?
git diff                    # show the exact edits
git add chapter_attention/attention-scoring.md
git commit -m "Fix transposed subscript in attention equation"
git push -u origin fix-typo-in-attention
```

すると GitHub 上で**プルリクエスト**（マージを提案するブランチであり、メンテナがレビューやコメントを行える diff を伴う）の作成が案内される（`gh` コマンドラインツールでも `gh pr create` で同じ操作ができる）。複数の章にまたがる変更よりも、小さく焦点を絞ったプルリクエストのほうがレビューしやすい。

39.8.3 コーディングエージェント

コーディングエージェントとは、リポジトリを読み取り、ファイルを編集し、コマンドを実行する言語モデルである。関数がどこで定義されているかを探すこと、機械的な変更をフレームワークごとの実装へ波及させること、あるいはチェックを実行して失敗箇所を読み取ることといった、範囲の明確な作業で力を発揮する。その利用は任意であり、エージェントによる編集にも人間の編集とまったく同じレビューが必要である。本リポジトリの 3 つの特性により、こうした支援が実用的なものとなっている。

- **ビルド手順が文書化されている。** `CLAUDE.md` と `docs/` には、アーティファクトの再生成・実行・レンダリングの方法がエージェント向けに記されており、同じ手順書が人間の貢献者にも役立つ。

- **検証が実行可能である**。編集が正しく機能しているとは、ノートブックが対応するすべてのフレームワークで先頭から末尾まで実行でき、コミット済み出力の監査を通過し、ページがレンダリングできることを意味する。編集、再生成、実行、失敗箇所の調査というループはエージェントが回せるが、最終的な diff とチェック結果の根拠の両方をレビューするのは貢献者である。
- **退屈な作業は機械的である**。フレームワーク間の実装の同期、すべての `:numref:` が解決されることの確認、図の再生成はいずれも複数ファイルにまたがる作業であり、エージェントが実行して人間がレビューするのに適している。

有益な依頼とは、たとえば「モメンタムの節の等高線プロットで軸ラベルが誤っているため、ソースを修正し、影響を受けるノートブックを再実行せよ」のように、変更内容とその検証方法を明記したものである。レビューでは 2 つの失敗モードに注意する。第 1 に、プロジェクトのドキュメントで警告していない限り（本リポジトリでは警告している）、エージェントは初心者と同様に生成ファイルを編集してしまうことがある。第 2 に、失敗しているチェックを通すよう求められたエージェントは、原因を直す代わりにチェックやコード自体を弱めてしまうことがある。ノートブックが再び実行できるようになっても、本来教えるべき内容を教えなくなっているかもしれないのである。

39.8.4 変更の提出

貢献の内容は 1 行の修正から新しいコードの追加まで多岐にわたり、変更が大きくなるにつれて必要なチェックも増える。1 行の修正であれば、ローカルへのチェックアウトすら必要ない。GitHub 上でソースファイルを開き、編集（鉛筆）ボタンを押せばよい。GitHub がリポジトリのフォークを提案し、編集後はコミットからプルリクエストの作成まで、すべてブラウザ上で案内してくれる。

コードや本文の変更では、リポジトリをクローンしてブランチを作成し、ソースの `.md` を編集する。編集したファイルは `tools/lint_source.py` で検査する。このスクリプトは、対応が取れていないタブやスライドのマーカ、重複したセル ID、数式に正しく紐づかない数式ラベルを検出し、`--corpus` を付けるとすべての相互参照が実在するラベルを指しているかも検査する。続いて、編集が影響するすべてのノートブックを再実行し、その出力をコミット対象のストアにコピーしてサイトをレンダリングする。

```
make venv-pytorch                                      # once per framework
make -B _notebooks/pytorch/<chapter>/<file>.executed  # repeat per framework
make capture-outputs FILES=<chapter>/<file>.md
make html
```

`#@save` が付いたコードを編集した場合は、ノートブックを実行する前に `make lib` でライブラリを再ビルドする。すると、編集したシンボルを使用している他の節のノートブックも陳腐化（stale）するため、`make audit-outputs` でそれらを一覧表示できる。GPU を持たないノート PC では GPU ノートブックを再実行できないため、`make html` はそのようなノートブックに対してビルドを停止せず警告を出す。プルリクエストには、GPU での実行がまだ必要なノートブックを明記しておくべきである。新しい本文は `docs/writing-guide.md` に、新しいコードは `docs/code-guide.md` に従う。

生の Markdown よりもノートブックを編集したい場合は、`.vscode-extension/` にある VS Code 拡張機能（ビルドとインストールの手順はその README に記載されている）を使うとよい。この拡張機能は、現在の節の生成済みノートブックを適切なカーネルが選択された状態で開き、フレームワークごとのビューを切り替え、スライドデッキをライブプレビューし、1 つのコマンドでノートブックの再実行と出力の取り込みまでこなす。ノートブック上で行った編集は Markdown

のソースに書き戻されるため、生成されたファイル自体を直接編集したことには決してならない。この拡張機能が使用する Jupyter カーネルは `make kernels` で登録できる。

どの手順をとるにせよ、変更が関係するすべてのフレームワークでまっさらな状態から問題なく実行でき、かつ変更されたのが生成ファイルではなくソースであるときに、変更の準備が整ったといえる。

39.8.5 まとめ

各節につき単一の Markdown ソースから、ウェブサイト、PDF、フレームワークごとのノートブック、スライド、および `d2l` パッケージが生成される一方、実行済み出力は別個にバージョン管理され、生成されたファイルを手作業で編集することは決してない。本文とコードをプレーンな Markdown に保つことで行単位の見通しのよい Git diff が得られるため、600 以上のノートブックからなる書籍であっても容易にレビュー・保守し続けられる。すべての貢献は同じループをたどる。すなわち、リポジトリをフォークし、目的を絞ったブランチを作成し、`.md` ソースを編集し、リント・影響を受けるノートブックの再実行・出力の取り込み・レンダリングによって検証したうえで、小さなプルリクエストを作成する。ビルドルールが文書化されており検証も実行可能であるため、機械的なマルチフレームワーク作業の多くはコーディングエージェントで自動化でき、人間の貢献者はその結果生じる変更をレビューして責任を持つことができる。

39.8.6 演習問題

1. **フレームワークタブ**。あるソースファイルに、次の順序でちょうど 5 個のコードブロックが含まれているとする。(1) `%%tab pytorch` のタグが付き、`import torch` と `x = torch.arange(6.0)` を実行するブロック、(2) `%%tab jax` のタグが付き、`import jax.numpy as jnp` と `x = jnp.arange(6.0)` を実行するブロック、(3) タグなしのブロック `y = x * 2`、(4) `%%tab pytorch` のタグが付き、`x[0] = 10.0` を実行するブロック、(5) タグなしのブロック `print(y.sum(), x.sum())`。
   1. ビルドはこのファイルからどのノートブックを生成し、それぞれにどのブロックが含まれ、それぞれ何を表示するか。
   2. ある貢献者がブロック (4) のタグを削除し、すべてのフレームワークで実行されるようにしたとする。このとき失敗するのはどのノートブックであり、どのようなエラーが発生するか。両方のノートブックが同じ値を表示するよう、ブロック (4) をタグ付きブロックに置き換えよ。
   3. 生成された PyTorch ノートブック上でブロック (3) を編集し、VS Code 拡張機能がその編集をソースファイルに書き戻したとする。この編集によって変更されるのはどのノートブックであり、提出前に実行しなければならないのはそのうちのどれか。
2. **陳腐化した出力**。コミット対象のストアは、ノートブックの出力をそれを生成したセルの ID（キー）ごとに整理して `outputs/<fw>/<chapter>/<stem>.json` に保持する。各出力とともにフィンガープリントとして、各行の末尾空白を除去したうえでそのセルおよびそれより前のすべてのコードセルのコードを求めたハッシュが保存される。またノートブック全体に対して、ローカルビルドタグを除いたインストール済みフレームワークのバージョン（`torch==2.11.0` であり、`torch==2.11.0+cu128` ではない）と、その節のコードが言及するすべてのシンボル `d2l.X` のライブラリブロック（およびそれらのシンボルがさらに使用するブロック）のハッシュも保存される。これらの値のいずれかが現在のソースや環境と一致しなくなったとき、出力は陳腐化していると判定される。監査スクリプト `tools/audit_outputs.py`（サイトをレンダリングする前に `make html` が実行する）は、ノートブックがホストの搭載数を超える GPU を必要とする場合を除き、陳腐化した画像を警告として報告し、陳腐化したテキスト出力があればビルドを停止する。
   1. ある節に 10 個のコードセルがあり、そのすべてのフレームワーク版が `d2l.Trainer` を呼び出しているとする。次の各編集を行った後、どのフレームワークのどの出力

が陳腐化するか。(i) 段落内の誤字を修正した場合、(ii) タグのない 3 番目のコードセル内のコメントの誤字を修正した場合、(iii) すべての行から末尾の空白を削除した場合、(iv) 5 番目と 6 番目のコードセルを入れ替えた場合、(v) 別の節にある `Trainer` メソッドの docstring を（タグなしの `#@save` セル内で）変更した場合、(vi) PyTorch を 2.11.0 から 2.12.0 にアップグレードした場合、(vii) Linux 向け CUDA ビルドの代わりに macOS 向け CPU ビルドから PyTorch 2.11.0 をインストールした場合。

2. (a) の陳腐化判定のうち、実際には変化しえない出力にフラグを立てているのはどれか。また、(vii) が見逃してしまう実際の出力変化にはどのようなものがあるか。ルールがこの両種の誤りを許容しているのはなぜか。
3. 新たに実行した結果が保存済みの出力と異なる表示になったとしても、ビルドがそれをもって出力を陳腐化と判定しないのはなぜか。

3. [code] **重複する定義**。`make lib` はすべての節の `#@save` ブロックを書籍の順序で収集する。複数のブロックが同じ関数やメソッドを定義している場合、ライブラリはそのうちの 1 つだけを保持する。メソッド `Trainer.prepare_batch` は、すべてのフレームワークで実行されるタグなしのセル `#linear-regression-scratch-training-1`（2.4 節）で定義され、さらにデバイス対応を備えた形でセル `#gpus-devices-memory-the-trainer-now-with-devices-1`（5.7 節）の各フレームワーク版で再定義されている。
   1. タグなしの定義の本体と、デバイス対応版の PyTorch 実装の本体の両方にコメント行を追加し、`make lib` を実行したうえで、`d2l/torch.py` 内で両方のコメントを検索せよ。どちらのコメントが書き出され、そこからどのようなルールがわかるか。
   2. `d2l.Trainer.prepare_batch` は、2.4 節の PyTorch ノートブックの開始時点ではどちらの定義を保持しており、そのセル `#linear-regression-scratch-training-1` の実行後にはどちらの定義を保持しているか。
   3. ビルドが最初の定義ではなく最後の定義を保持するのはなぜか。
   4. ある貢献者がデバイス対応の PyTorch 版 `prepare_batch` を変更し、すべてのバッチを 16 ビット浮動小数点数にも変換するようにしたとする。異なる結果を表示しうるのはどの節の PyTorch ノートブックか。書籍の順序で 5.7 節より前の節か、後の節か、あるいはその両方か。
4. [extended] **データフローに基づく陳腐化判定**。●●演習問題 2 のフィンガープリントは、最初に編集されたセル以降のすべてのセルを陳腐化とマークする。これに対しデータフロールールでは、編集されたセルが束縛（bind）する名前を直接、あるいは他のセルを介して読み取るセルのみをマークする。演習問題 3（39.1 節）の検出器は、各セルが束縛する名前と読み取る名前をすでに抽出している。
   1. `make notebooks-pytorch` で PyTorch ノートブックを生成し、すべてのコードセルについて束縛する名前と読み取る名前を抽出せよ。単一セルのあらゆる編集について、データフロールールが陳腐化とマークするセルと、それらを再計算するために実行しなければならないセル（陳腐化したセルが依存するそれ以前のすべてのセルを含む）を求めよ。両ルールの下で、陳腐化とマークされるセルの割合の平均と、実行しなければならないセルの割合の平均を章ごとに報告せよ。
   2. たとえば乱数生成器の状態を介する場合のように、後続のセルが読み取る名前を束縛することなく、編集によって後続の出力を変化させてしまう本書中のセルを見つけよ。本書のノートブックでは、この種の他のどのようなチャネルが使われているか。
   3. Macke et al. (2021) は、ソースから依存関係を読み取る代わりに、ノートブックの実行中にそのような依存関係を記録する。実行時トレーサが観測できるのは (b) のチャネルのうちどれであり、観測できないまま残るのはどれか。
   4. ビルドのプレフィックスルールをこのルールに置き換えるべきか。(a) での節約量と (b) での破綻例に基づいて答えよ。
5. [code] **ノイズを伴う二分探索**。●●`git bisect` は二分探索によって不具合混入元のコミットを特定する。すなわち、正常と判明しているコミットと不良と判明しているコミットの中間のコミットをチェックアウトし、そのコミットをテストすることで探索範囲を半分に絞る。

しかし、モデルを訓練するテストにはノイズが伴う。ノートブックのチェック「検証正解率が 0.9 以上」が、各実行で独立に、確率 $p = 0.9$ で最初の不良コミット k より前のすべてのコミットにおいて通過し、確率 $q = 0.3$ で k 以降のすべてのコミットにおいて通過するとする。コミット 0 は正常、コミット 64 は不良であることがわかっており、k は $\{1, \dots, 64\}$ 上の一様分布に従うとする。

1. 1 回のテストにつき 1 回だけ実行するとき、`git bisect` が正しく k を特定する確率はいくらか。
2. 各テストを m 回繰り返し、通過回数がしきい値以上に達したときにそのコミットを正常と判定するとする。誤ったコミットを特定する確率が多くても 5% となる最小の m とそのしきい値を求めよ。この探索には合計何回の実行を要するか。
3. 確率的二分法（probabilistic bisection）アルゴリズムは、不具合の位置に関する事後分布を保持し、その中央値でテストを実行して、実行ごとにベイズの定理で事後分布を更新する (Frazier et al. 2019)。k に関する事後分布と、p および q で与えられる尤度を用いて、事後分布が 1 つのコミットに 0.95 の確率を割り当てるまでこれをシミュレーションせよ。離散的なコミットの集合上では、中央値でのテストが進行しなくなりうるのはなぜか。これを解決するようテスト対象コミットの選び方を変更し、平均実行回数と誤り率を (b) と比較せよ。
4. 1 回の実行を $\{\text{good}, \text{bad}\}$ から $\{\text{pass}, \text{fail}\}$ への通信路とみなし、その通信路容量 $C = \max_{p(x)} I(X;Y)$、すなわちすべての入力分布にわたる入出力間の相互情報量の最大値（37.6 節）を計算せよ。テストを適応的に選んだとしても、各実行が明らかにする量は高々 C ビット（k に関する情報量）である。ファノの不等式によれば、実行結果から計算される推定量 $\hat{k}$ は、誤り確率を $P_e = P(\hat{k} \neq k)$ とするとき、$H_b(P_e) + P_e \log_2 63 \geq H(k \mid \text{runs})$ を満たす（ここで H_b は二値エントロピーである）。これを用いて、k を少なくとも 0.95 の確率で特定する任意の戦略における平均実行回数の下界を求めよ。(c) はこの下界にどこまで迫っているか。

参考文献

Abadi, Martín, Paul Barham, Jianmin Chen, et al. 2016. “TensorFlow: A System for Large-Scale Machine Learning.” **12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)**, 265–83. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

Achiam, Joshua. 2017. **Advanced Policy Gradient Methods**. Lecture, UC Berkeley CS 294-112: Deep Reinforcement Learning, https://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf.

Achiam, Joshua. 2018. **Spinning up in Deep RL**. Online, OpenAI; PyTorch rewrite Jan 2020, https://spinningup.openai.com/.

Agarwal, Alekh, Nan Jiang, Sham M Kakade, and Wen Sun. 2019. **Reinforcement Learning: Theory and Algorithms**. Working draft monograph, continuously revised, https://rltheorybook.github.io/.

Agarwal, Alekh, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. 2021. “On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift.” **Journal of Machine Learning Research** 22 (98): 1–76.

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. 2021. “Deep Reinforcement Learning at the Edge of the Statistical Precipice.” **Advances in Neural Information Processing Systems** 34. https://arxiv.org/abs/2108.13264.

Agarwal, Rishabh, Nino Vieillard, Yongchao Zhou, et al. 2024. “On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes.” **International Conference on Learning Representations**. https://arxiv.org/abs/2306.13649.

Agrawal, Amey, Nitin Kedia, Ashish Panwar, et al. 2024. “Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve.” **18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)**, 117–34.

Agrawal, Shipra, and Navin Goyal. 2012. “Analysis of Thompson Sampling for the Multi-Armed Bandit Problem.” **Proceedings of the 25th Annual Conference on Learning Theory**, Proceedings of machine learning research, vol. 23: 39.1–26. https://proceedings.mlr.press/v23/agrawal12.html.

Ahmadian, Arash, Chris Cremer, Matthias Gallé, et al. 2024. “Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs.” **Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics**. https://arxiv.org/abs/2402.14740.

Ahn, Kwangjun, Byron Xu, Natalie Abreu, et al. 2025. “Dion: Distributed Orthonormalized Updates.” **arXiv Preprint arXiv:2504.05295**.

Ainslie, Joshua, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. “GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints.” **Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing**, 4895–901.

Ainsworth, Samuel K., Jonathan Hayase, and Siddhartha Srinivasa. 2023. “Git Re-Basin: Merging Models Modulo Permutation Symmetries.” **International Conference on Learning Representations**.

Akyürek, Ekin, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2023. “What Learning Algorithm Is in-Context Learning? Investigations with Linear Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2211.15661.

Alayrac, Jean-Baptiste, Jeff Donahue, Pauline Luc, et al. 2022. “Flamingo: A Visual Language Model for Few-Shot Learning.” **ArXiv:2204.14198**. https://arxiv.org/abs/2204.14198.

Albergo, Michael S., Nicholas M. Boffi, and Eric Vanden-Eijnden. 2023. “Stochastic Interpolants: A Unifying Framework for Flows and Diffusions.” **arXiv Preprint arXiv:2303.08797**.

Alemi, Alexander A., Ian Fischer, Joshua V. Dillon, and Kevin Murphy. 2017. “Deep Variational Information Bottleneck.” **International Conference on Learning Representations**.

Alsallakh, Bilal, Narine Kokhlikyan, Vivek Miglani, Jun Yuan, and Orion Reblitz-Richardson. 2020. “Mind the PAD – CNNs Can Develop Blind Spots.” **ArXiv:2010.02178**. https://arxiv.org/abs/2010.02178.

Amari, Shun-ichi. 1998. “Natural Gradient Works Efficiently in Learning.” **Neural Computation** 10 (2): 251–76.

Amari, Shun-ichi. 2016. **Information Geometry and Its Applications**. Springer.

Amdahl, Gene M. 1967. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.” **Proceedings of the AFIPS Spring Joint Computer Conference** 30: 483–85.

Ameisen, Emmanuel, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, et al. 2025. **Circuit Tracing: Revealing Computational Graphs in Language Models**. Transformer Circuits Thread, https://transformer-circuits.pub/2025/attribution-graphs/met

hods.html.

Amsel, Noah, David Persson, Christopher Musco, and Robert M. Gower. 2026. “The Polar Express: Optimal Matrix Sign Methods and Their Application to the Muon Algorithm.” **International Conference on Learning Representations**.

Anderson, Brian D. O. 1982. “Reverse-Time Diffusion Equation Models.” **Stochastic Processes and Their Applications** 12 (3): 313–26.

Anderson, Brian D. O., and John B. Moore. 2007. **Optimal Control: Linear Quadratic Methods**. Dover Publications.

Andrychowicz, Marcin, Anton Raichuk, Piotr Stańczyk, et al. 2021. “What Matters for on-Policy Deep Actor-Critic Methods? A Large-Scale Study.” **International Conference on Learning Representations**. https://arxiv.org/abs/2006.05990.

Angelopoulos, Anastasios N., and Stephen Bates. 2023. “Conformal Prediction: A Gentle Introduction.” **Foundations and Trends in Machine Learning** 16 (4): 494–591. https://doi.org/10.1561/2200000101.

Ansel, Jason, Edward Yang, Horace He, et al. 2024. “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation.” **29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)**, 929–47. https://doi.org/10.1145/3620665.3640366.

Arıkan, Erdal. 2009. “Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels.” **IEEE Transactions on Information Theory** 55 (7): 3051–73. https://doi.org/10.1109/TIT.2009.2021379.

Arjevani, Yossi, Yair Carmon, John C. Duchi, Dylan J. Foster, Nathan Srebro, and Blake Woodworth. 2023. “Lower Bounds for Non-Convex Stochastic Optimization.” **Mathematical Programming** 199: 165–214.

Arjovsky, Martin, and Léon Bottou. 2017. “Towards Principled Methods for Training Generative Adversarial Networks.” **International Conference on Learning Representations**.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. “Wasserstein Generative Adversarial Networks.” **Proceedings of the 34th International Conference on Machine Learning**, 214–23.

Armijo, Larry. 1966. “Minimization of Functions Having Lipschitz Continuous First Partial Derivatives.” **Pacific Journal of Mathematics** 16 (1): 1–3.

Arora, Sanjeev, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. 2017. “Generalization and Equilibrium in Generative Adversarial Nets (GANs).” **International Conference on Machine Learning**, 224–32.

Arora, Sanjeev, Zhiyuan Li, and Kaifeng Lyu. 2019. “Theoretical Analysis of Auto Rate-Tuning by Batch Normalization.” **International Conference on Learning Representations**. https://arxiv.org/abs/1812.03981.

Arora, Simran, Sabri Eyuboglu, Aman Timalsina, et al. 2024. “Zoology: Measuring and Improving Recall in Efficient Language Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2312.04927.

Arora, Simran, Sabri Eyuboglu, Michael Zhang, et al. 2024. “Simple Linear Attention Language Models Balance the Recall-Throughput Tradeoff.” **International Conference on Machine Learning**.

Arpit, Devansh, Stanisław Jastrzębski, Nicolas Ballas, et al. 2017. “A Closer Look at Memorization in Deep Networks.” **International Conference on Machine Learning**, 233–42.

Arriola, Marianne, Aaron Gokaslan, Justin T. Chiu, et al. 2025. “Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2503.09573.

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. “Finite-Time Analysis of the Multiarmed Bandit Problem.” **Machine Learning** 47 (2-3): 235–56.

Austin, Jacob, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. “Structured Denoising Diffusion Models in Discrete State-Spaces.” **Advances in Neural Information Processing Systems** 34.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. “Layer Normalization.” **ArXiv:1607.06450**. https://arxiv.org/abs/1607.06450.

Bach, Francis, and Eric Moulines. 2013. “Non-Strongly-Convex Smooth Stochastic Approximation with Convergence Rate $O(1/n)$.” **Advances in Neural Information Processing Systems** 26.

Bachlechner, Thomas, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W. Cottrell, and Julian McAuley. 2021. “ReZero Is All You Need: Fast Convergence at Large Depth.” **Uncertainty in Artificial Intelligence**. https://arxiv.org/abs/2003.04887.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. “Neural Machine Translation by Jointly Learning to Align and Translate.” **International Conference on Learning Representations**. https://doi.org/10.48550/arXiv.1409.0473.

Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. 2019. “Deep Equilibrium Models.” **Advances in Neural Information Processing Systems** 32.

Baik, Jinho, and Jack W. Silverstein. 2006. “Eigenvalues of Large Sample Covariance Matrices of Spiked Population Models.” **Journal of Multivariate Analysis** 97 (6): 1382–408. https://doi.org/10.1016/j.jmva.2005.08.003.

Baird, Leemon. 1995. “Residual Algorithms: Reinforcement Learning with Function Approximation.” **International Conference on Machine Learning**, 30–37.

Bakker, Bram. 2001. “Reinforcement Learning with Long Short-Term Memory.” **Advances in Neural Information Processing Systems** 14.

Baldi, Pierre, and Kurt Hornik. 1989. “Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima.” **Neural Networks** 2 (1): 53–58.

Balduzzi, David, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. 2017. “The Shattered Gradients Problem: If ResNets Are the Answer, Then What Is the Question?” **Proceedings of the 34th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 70: 342–50.

Ball, Philip J., Laura Smith, Ilya Kostrikov, and Sergey Levine. 2023. “Efficient Online Reinforcement Learning with Offline Data.” **International Conference on Machine Learning**, 1577–94.

Ballé, Johannes, Valero Laparra, and Eero P. Simoncelli. 2017. “End-to-End Optimized Image Compression.” **International Conference on Learning Representations**.

Ballé, Johannes, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. “Variational Image Compression with a Scale Hyperprior.” **International Conference on Learning Representations**.

Bao, Fan, Chongxuan Li, Jun Zhu, and Bo Zhang. 2022. “Analytic-DPM: An Analytic Estimate of the Optimal Reverse Variance in Diffusion Probabilistic Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2201.06503.

Barber, David, and Felix Agakov. 2003. “The IM Algorithm: A Variational Approach to Information Maximization.” **Advances in Neural Information Processing Systems** 16.

Barbero, Federico, Álvaro Arroyo, Xiangming Gu, et al. 2025. “Why Do LLMs Attend to the First Token?” **Conference on Language Modeling (COLM)**. https://arxiv.org/abs/2504.02732.

Barbero, Federico, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. 2025. “Round and Round We Go! What Makes Rotary Positional Encodings Useful?” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.06205.

Barrett, David G. T., and Benoit Dherin. 2021. “Implicit Gradient Regularization.” **International Conference on Learning Representations**.

Bartlett, Peter L., Philip M. Long, Gábor Lugosi, and Alexander Tsigler. 2020. “Benign Overfitting in Linear Regression.” **Proceedings of the National Academy of Sciences** 117 (48): 30063–70.

Bartlett, Peter L., and Shahar Mendelson. 2002. “Rademacher and Gaussian Complexities: Risk Bounds and Structural Results.” **Journal of Machine Learning Research** 3: 463–82.

Barto, Andrew G, Richard S Sutton, and Charles W Anderson. 1983. “Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems.” **IEEE Transactions on Systems, Man, and Cybernetics**, no. 5: 834–46.

Baur, Walter, and Volker Strassen. 1983. “The Complexity of Partial Derivatives.” **Theoret-**

ical Computer Science 22 (3): 317–30.

Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. 2006. "SURF: Speeded up Robust Features." **European Conference on Computer Vision**, 404–17. https://doi.org/10.1007/11744023_32.

Baydin, Atılım Güneş, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. 2022. "Gradients Without Backpropagation." **arXiv Preprint arXiv:2202.08587**.

Bayes, Thomas. 1763. "An Essay Towards Solving a Problem in the Doctrine of Chances." **Philosophical Transactions of the Royal Society of London** 53: 370–418.

Beck, Amir, and Marc Teboulle. 2003. "Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization." **Operations Research Letters** 31 (3): 167–75. https://doi.org/10.1016/S0167-6377(02)00231-6.

Beck, Amir, and Marc Teboulle. 2009. "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems." **SIAM Journal on Imaging Sciences** 2 (1): 183–202.

Beck, Amir, and Luba Tetruashvili. 2013. "On the Convergence of Block Coordinate Descent Type Methods." **SIAM Journal on Optimization** 23 (4): 2037–60.

Beck, Maximilian, Korbinian Pöppel, Phillip Lippe, et al. 2025. "xLSTM 7B: A Recurrent LLM for Fast and Efficient Inference." **arXiv Preprint arXiv:2503.13427**.

Beck, Maximilian, Korbinian Pöppel, Markus Spanring, et al. 2024. "xLSTM: Extended Long Short-Term Memory." **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2405.04517.

Behrmann, Jens, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. 2019. "Invertible Residual Networks." **International Conference on Machine Learning**, 573–82.

Behrouz, Ali, Peilin Zhong, and Vahab Mirrokni. 2025. "Titans: Learning to Memorize at Test Time." **arXiv Preprint arXiv:2501.00663**.

Belghazi, Mohamed Ishmael, Aristide Baratin, Sai Rajeshwar, et al. 2018. "Mutual Information Neural Estimation." **Proceedings of the 35th International Conference on Machine Learning**, 531–40.

Belkin, Mikhail, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. "Reconciling Modern Machine-Learning Practice and the Classical Bias–Variance Trade-Off." **Proceedings of the National Academy of Sciences** 116 (32): 15849–54.

Bellemare, Marc G, Will Dabney, and Rémi Munos. 2017. "A Distributional Perspective on Reinforcement Learning." **International Conference on Machine Learning**, 449–58.

Bellemare, Marc G., Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Rémi Munos. 2016. "Unifying Count-Based Exploration and Intrinsic Motivation." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/1606.01868.

Bellman, Richard. 1952. “On the Theory of Dynamic Programming.” **Proceedings of the National Academy of Sciences** 38 (8): 716–19. https://doi.org/10.1073/pnas.38.8.716.

Bellman, Richard. 1957a. “A Markovian Decision Process.” **Journal of Mathematics and Mechanics** 6 (5): 679–84. http://www.jstor.org/stable/24900506.

Bellman, Richard. 1957b. **Dynamic Programming**. Princeton University Press.

Beltagy, Iz, Matthew E Peters, and Arman Cohan. 2020. “Longformer: The Long-Document Transformer.” **ArXiv:2004.05150**. https://arxiv.org/abs/2004.05150.

Benamou, Jean-David, and Yann Brenier. 2000. “A Computational Fluid Mechanics Solution to the Monge–Kantorovich Mass Transfer Problem.” **Numerische Mathematik** 84 (3): 375–93.

Benaych-Georges, Florent, and Raj Rao Nadakuditi. 2012. “The Singular Values and Vectors of Low Rank Perturbations of Large Rectangular Random Matrices.” **Journal of Multivariate Analysis** 111: 120–35. https://doi.org/10.1016/j.jmva.2012.04.019.

Bengio, Samy, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks.” **Advances in Neural Information Processing Systems** 28. https://arxiv.org/abs/1506.03099.

Bengio, Yoshua, Aaron Courville, and Pascal Vincent. 2013. “Representation Learning: A Review and New Perspectives.” **IEEE Transactions on Pattern Analysis and Machine Intelligence** 35 (8): 1798–828.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. “A Neural Probabilistic Language Model.” **Journal of Machine Learning Research** 3 (Feb): 1137–55. https://jmlr.org/papers/v3/bengio03a.html.

Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. 2013. “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.” **arXiv Preprint arXiv:1308.3432**.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. “Learning Long-Term Dependencies with Gradient Descent Is Difficult.” **IEEE Transactions on Neural Networks** 5 (2): 157–66. https://doi.org/10.1109/72.279181.

Benjamini, Yoav, and Yosef Hochberg. 1995. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing.” **Journal of the Royal Statistical Society: Series B (Methodological)** 57 (1): 289–300.

Bergsma, Shane, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. 2025. “Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-Training.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2505.13738.

Bergstra, James, Olivier Breuleux, Frédéric Bastien, et al. 2010. “Theano: A CPU and GPU Math Compiler in Python.” **Proc. 9th Python in Science Conference** 1: 3–10. https:

//www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf.

Bernstein, Jeremy. 2025. **Deriving Muon**. Https://jeremybernste.in/writing/deriving-muon.

Bernstein, Jeremy, and Laker Newhouse. 2024. "Old Optimizer, New Norm: An Anatomy." **arXiv Preprint arXiv:2409.20325**.

Bertoin, David, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. 2021. "Numerical Influence of ReLU'(0) on Backpropagation." **Advances in Neural Information Processing Systems** 34: 468–79. https://arxiv.org/abs/2106.12915.

Bertrand, Quentin, Anne Gagneux, Mathurin Massias, and Rémi Emonet. 2025. "On the Closed-Form of Flow Matching: Generalization Does Not Arise from Target Stochasticity." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2506.03719.

Bertsekas, Dimitri P. 2025. **A Course in Reinforcement Learning**. 2nd ed. Athena Scientific.

Besiroglu, Tamay, Ege Erdil, Matthew Barnett, and Josh You. 2024. "Chinchilla Scaling: A Replication Attempt." **arXiv Preprint arXiv:2404.10102**.

Beyer, Lucas, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. 2020. "Are We Done with ImageNet?" **ArXiv:2006.07159**. https://arxiv.org/abs/2006.07159.

Bhatt, Aditya, Daniel Palenicek, Boris Belousov, et al. 2024. "CrossQ: Batch Normalization in Deep Reinforcement Learning for Greater Sample Efficiency and Simplicity." **International Conference on Learning Representations**.

Bick, Aviv, Kevin Y. Li, Eric P. Xing, J. Zico Kolter, and Albert Gu. 2024. "Transformers to SSMs: Distilling Quadratic Knowledge to Subquadratic Models." **Advances in Neural Information Processing Systems**.

Billingsley, Patrick. 1995. **Probability and Measure**. 3rd ed. Wiley.

Bińkowski, Mikołaj, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. 2018. "Demystifying MMD GANs." **International Conference on Learning Representations**. https://arxiv.org/abs/1801.01401.

Biroli, Giulio, Tony Bonnaire, Valentin de Bortoli, and Marc Mézard. 2024. "Dynamical Regimes of Diffusion Models." **Nature Communications** 15: 9957. https://arxiv.org/abs/2402.18491.

Bishop, Chris M. 1995. "Training with Noise Is Equivalent to Tikhonov Regularization." **Neural Computation** 7 (1): 108–16. https://doi.org/10.1162/neco.1995.7.1.108.

Bishop, Christopher M. 2006. **Pattern Recognition and Machine Learning**. Springer. https://doi.org/10.1007/978-0-387-45528-0.

Black, Fischer, and Myron Scholes. 1973. “The Pricing of Options and Corporate Liabilities.” **Journal of Political Economy** 81: 637–54. https://doi.org/10.1086/260062.

Black Forest Labs. 2024. **FLUX.1 [Schnell] Model Card**. Https://huggingface.co/black-forest-labs/FLUX.1-schnell.

Black, Kevin, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. 2024. “Training Diffusion Models with Reinforcement Learning.” **International Conference on Learning Representations**. https://arxiv.org/abs/2305.13301.

Blattmann, Andreas, Tim Dockhorn, Sumith Kulal, et al. 2023. “Stable Video Diffusion: Scaling Latent Video Diffusion Models to Large Datasets.” **arXiv Preprint arXiv:2311.15127**. https://arxiv.org/abs/2311.15127.

Blau, Yochai, and Tomer Michaeli. 2018. “The Perception-Distortion Tradeoff.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 6228–37.

Blelloch, Guy E. 1990. **Prefix Sums and Their Applications**. CMU-CS-90-190. School of Computer Science, Carnegie Mellon University. https://www.cs.cmu.edu/~scandal/papers/CMU-CS-90-190.html.

Blier, Léonard, and Yann Ollivier. 2018. “The Description Length of Deep Learning Models.” **Advances in Neural Information Processing Systems** 31. https://arxiv.org/abs/1802.07044.

Blum, Avrim, and Moritz Hardt. 2015. “The Ladder: A Reliable Leaderboard for Machine Learning Competitions.” **Proceedings of the 32nd International Conference on Machine Learning**, Proceedings of machine learning research, vol. 37: 1006–14.

Boffi, Nicholas M., Michael S. Albergo, and Eric Vanden-Eijnden. 2025. “How to Build a Consistency Model: Learning Flow Maps via Self-Distillation.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2505.18825.

Bogachev, Vladimir I., Nicolai V. Krylov, Michael Röckner, and Stanislav V. Shaposhnikov. 2015. **Fokker–Planck–Kolmogorov Equations**. Vol. 207. Mathematical Surveys and Monographs. American Mathematical Society.

Bollobás, B. 1999. **Linear Analysis**. Cambridge University Press. https://doi.org/10.1017/CBO9780511626296.

Bolte, Jérôme, and Edouard Pauwels. 2021. “Conservative Set Valued Fields, Automatic Differentiation, Stochastic Gradient Methods and Deep Learning.” **Mathematical Programming** 188 (1): 19–51.

Bonferroni, Carlo E. 1936. “Teoria Statistica Delle Classi e Calcolo Delle Probabilità.” **Pubblicazioni Del R. Istituto Superiore Di Scienze Economiche e Commerciali Di Firenze** 8: 3–62.

Bonneel, Nicolas, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. 2015. “Sliced and Radon Wasserstein Barycenters of Measures.” **Journal of Mathematical Imaging and**

Vision 51 (1): 22–45. https://doi.org/10.1007/s10851-014-0506-3.

Bordelon, Blake, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. 2024. "Depthwise Hyperparameter Transfer in Residual Networks: Dynamics and Scaling Limit." **International Conference on Learning Representations**. https://arxiv.org/abs/2309.16620.

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. "A Training Algorithm for Optimal Margin Classifiers." **Proceedings of the Fifth Annual Workshop on Computational Learning Theory**, 144–52.

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal. 2018. "Optimization Methods for Large-Scale Machine Learning." **SIAM Review** 60 (2): 223–311.

Boucheron, Stéphane, Olivier Bousquet, and Gábor Lugosi. 2005. "Theory of Classification: A Survey of Some Recent Advances." **ESAIM: Probability and Statistics** 9: 323–75. https://doi.org/10.1214/154957805100000046.

Boucheron, Stéphane, Gábor Lugosi, and Pascal Massart. 2013. **Concentration Inequalities: A Nonasymptotic Theory of Independence**. Oxford University Press.

Bousquet, Olivier, and André Elisseeff. 2002. "Stability and Generalization." **Journal of Machine Learning Research** 2: 499–526. https://www.jmlr.org/papers/v2/bousquet02a.html.

Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." **Foundations and Trends in Machine Learning** 3 (1): 1–122. https://doi.org/10.1561/2200000016.

Boyd, Stephen, and Lieven Vandenberghe. 2004. **Convex Optimization**. Cambridge University Press. https://web.stanford.edu/~boyd/cvxbook/.

Bradley, Ralph Allan, and Milton E Terry. 1952. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." **Biometrika** 39 (3/4): 324–45. https://doi.org/10.1093/biomet/39.3-4.324.

Brandfonbrener, David, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. 2022. "When Does Return-Conditioned Supervised Learning Work for Offline Reinforcement Learning?" **Advances in Neural Information Processing Systems** 35.

Bray, Alan J., and David S. Dean. 2007. "Statistics of Critical Points of Gaussian Fields on Large-Dimensional Spaces." **Physical Review Letters** 98 (15): 150201.

Brenier, Yann. 1991. "Polar Factorization and Monotone Rearrangement of Vector-Valued Functions." **Communications on Pure and Applied Mathematics** 44 (4): 375–417.

Bretagnolle, Jean, and Catherine Huber. 1978. "Estimation Des Densités: Risque Minimax." In **Séminaire de Probabilités XII**, vol. 649. Lecture Notes in Mathematics. Springer.

Bricken, Trenton, Adly Templeton, Joshua Batson, et al. 2023. **Towards Monosemanticity: Decomposing Language Models with Dictionary Learning**. Transformer Circuits Thread, https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Brin, Sergey, and Lawrence Page. 1998. "The Anatomy of a Large-Scale Hypertextual Web Search Engine." **Computer Networks and ISDN Systems** 30 (1–7): 107–17.

Brock, Andrew, Soham De, Samuel L. Smith, and Karen Simonyan. 2021. "High-Performance Large-Scale Image Recognition Without Normalization." **International Conference on Machine Learning**. https://arxiv.org/abs/2102.06171.

Brock, Andrew, Jeff Donahue, and Karen Simonyan. 2019. "Large Scale GAN Training for High Fidelity Natural Image Synthesis." **International Conference on Learning Representations**. https://arxiv.org/abs/1809.11096.

Brown, Noam, and Tuomas Sandholm. 2017. "Libratus: The Superhuman AI for No-Limit Poker." **IJCAI**, 5226–28. https://doi.org/10.24963/ijcai.2017/772.

Brown, Tom, Benjamin Mann, Nick Ryder, et al. 2020. "Language Models Are Few-Shot Learners." **Advances in Neural Information Processing Systems** 33: 1877–901. https://arxiv.org/abs/2005.14165.

Bubeck, Sébastien, and Nicolò Cesa-Bianchi. 2012. "Regret Analysis of Stochastic and Nonstochastic Multi-Armed Bandit Problems." **Foundations and Trends in Machine Learning** 5 (1): 1–122. https://doi.org/10.1561/2200000024.

Burda, Yuri, Harrison Edwards, Amos Storkey, and Oleg Klimov. 2019. "Exploration by Random Network Distillation." **International Conference on Learning Representations**. https://arxiv.org/abs/1810.12894.

Burda, Yuri, Roger Grosse, and Ruslan Salakhutdinov. 2016. "Importance Weighted Autoencoders." **International Conference on Learning Representations**. https://arxiv.org/abs/1509.00519.

Byrd, Jonathon, and Zachary C. Lipton. 2019. "What Is the Effect of Importance Weighting in Deep Learning?" **Proceedings of the 36th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 97: 872–81.

Campbell, Andrew, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. 2022. "A Continuous Time Framework for Discrete Denoising Models." **Advances in Neural Information Processing Systems** 35. https://arxiv.org/abs/2205.14987.

Campbell, Andrew, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. 2024. "Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design." **International Conference on Machine Learning**.

Candès, Emmanuel J., and Benjamin Recht. 2009. "Exact Matrix Completion via Convex Optimization." **Foundations of Computational Mathematics** 9 (6): 717–72.

Canny, John. 1987. “A Computational Approach to Edge Detection.” In **Readings in Computer Vision**. Elsevier. https://doi.org/10.1109/tpami.1986.4767851.

Canonne, Clément L. 2022. “A Short Note on an Inequality Between KL and TV.” **arXiv Preprint arXiv:2202.07198**. https://arxiv.org/abs/2202.07198.

Carson, Erin, and Nicholas J. Higham. 2018. “Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions.” **SIAM Journal on Scientific Computing** 40 (2): A817–47.

Čencov, N. N. 1982. **Statistical Decision Rules and Optimal Inference**. Vol. 53. Translations of Mathematical Monographs. American Mathematical Society.

Cesista, Franz Louis, Jiacheng You, and Keller Jordan. 2025. **Squeezing 1–2% Efficiency Gains Out of Muon by Optimizing the Newton–Schulz Coefficients**. Http://leloykun.github.io/ponder/muon-opt-coeffs/.

Chang, Huiwen, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. 2022. “MaskGIT: Masked Generative Image Transformer.” **IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 11315–25. https://arxiv.org/abs/2202.04200.

Chebyshev, Pafnuty L. 1867. “Des Valeurs Moyennes.” **Journal de Mathématiques Pures Et Appliquées** 12 (2): 177–84.

Chechik, Gal, Amir Globerson, Naftali Tishby, and Yair Weiss. 2005. “Information Bottleneck for Gaussian Variables.” **Journal of Machine Learning Research** 6: 165–88.

Chen, Charlie, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. “Accelerating Large Language Model Decoding with Speculative Sampling.” **arXiv Preprint arXiv:2302.01318**.

Chen, Junyu, Han Cai, Junsong Chen, et al. 2025. “Deep Compression Autoencoder for Efficient High-Resolution Diffusion Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.10733.

Chen, Lei, and Joan Bruna. 2023. “Beyond the Edge of Stability via Two-Step Gradient Updates.” **International Conference on Machine Learning**, 4330–91.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, et al. 2021. “Decision Transformer: Reinforcement Learning via Sequence Modeling.” **Advances in Neural Information Processing Systems** 34.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, et al. 2021. “Decision Transformer: Reinforcement Learning via Sequence Modeling.” **Advances in Neural Information Processing Systems** 34: 15084–97. https://arxiv.org/abs/2106.01345.

Chen, Mark, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. 2021. “Evaluating Large Language Models Trained on Code.” **arXiv Preprint arXiv:2107.03374**.

Chen, Ricky T. Q., Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. “Neural Ordinary Differential Equations.” **Advances in Neural Information Processing Systems** 31.

Chen, Shouyuan, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. “Extending Context Window of Large Language Models via Positional Interpolation.” **arXiv Preprint arXiv:2306.15595**.

Chen, Sitan, Sinho Chewi, Jerry Li, Yuanzhi Li, Adil Salim, and Anru R. Zhang. 2023. “Sampling Is as Easy as Learning the Score: Theory for Diffusion Models with Minimal Data Assumptions.” **International Conference on Learning Representations**.

Chen, Stanley F., and Joshua Goodman. 1999. “An Empirical Study of Smoothing Techniques for Language Modeling.” **Computer Speech & Language** 13 (4): 359–93. https://doi.org/10.1006/csla.1999.0128.

Chen, Tianqi, Mu Li, Yutian Li, et al. 2015. “MXNET: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems.” **ArXiv:1512.01274**. https://arxiv.org/abs/1512.01274.

Chen, Tianqi, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. “Training Deep Nets with Sublinear Memory Cost.” **arXiv Preprint arXiv:1604.06174**.

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. “A Simple Framework for Contrastive Learning of Visual Representations.” **Proceedings of the 37th International Conference on Machine Learning**, 1597–607.

Chen, Xinyue, Che Wang, Zijian Zhou, and Keith Ross. 2021. “Randomized Ensembled Double Q-Learning: Learning Fast Without a Model.” **International Conference on Learning Representations**. https://arxiv.org/abs/2101.05982.

Cheng, Jianpeng, Li Dong, and Mirella Lapata. 2016. “Long Short-Term Memory-Networks for Machine Reading.” **Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing**, 551–61. https://doi.org/10.18653/v1/d16-1053.

Cheng, Pengyu, Weituo Hao, Shuyang Dai, Jiachang Liu, Zhe Gan, and Lawrence Carin. 2020. “CLUB: A Contrastive Log-Ratio Upper Bound of Mutual Information.” **Proceedings of the 37th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 119: 1779–88.

Chi, Cheng, Siyuan Feng, Yilun Du, et al. 2023. “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion.” **Robotics: Science and Systems**. https://arxiv.org/abs/2303.04137.

Chi, Ta-Chung, Ting-Han Fan, Alexander I. Rudnicky, and Peter J. Ramadge. 2023. “Dissecting Transformer Length Extrapolation via the Lens of Receptive Field Analysis.” **Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics**. https://arxiv.org/abs/2212.10356.

Chiang, Wei-Lin, Lianmin Zheng, Ying Sheng, et al. 2024. “Chatbot Arena: An Open Plat-

form for Evaluating LLMs by Human Preference." **International Conference on Machine Learning**, 8359–88. https://arxiv.org/abs/2403.04132.

Child, Rewon, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. "Generating Long Sequences with Sparse Transformers." **ArXiv:1904.10509**. https://arxiv.org/abs/1904.10509.

Cho, Kyunghyun, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches." **ArXiv:1409.1259**. https://arxiv.org/abs/1409.1259.

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, et al. 2014. "Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation." **Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing**, 1724–34. https://arxiv.org/abs/1406.1078.

Chollet, François. 2017. "Xception: Deep Learning with Depthwise Separable Convolutions." **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. https://arxiv.org/abs/1610.02357.

Chong, Min Jin, and David Forsyth. 2020. "Effectively Unbiased FID and Inception Score and Where to Find Them." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 6070–79. https://arxiv.org/abs/1911.07023.

Choromanski, Krzysztof, Valerii Likhosherstov, David Dohan, et al. 2021. "Rethinking Attention with Performers." **International Conference on Learning Representations**.

Chowdhery, Aakanksha, Sharan Narang, Jacob Devlin, et al. 2022. "PaLM: Scaling Language Modeling with Pathways." **ArXiv:2204.02311**. https://arxiv.org/abs/2204.02311.

Christiano, Paul, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. "Deep Reinforcement Learning from Human Preferences." **Advances in Neural Information Processing Systems** 30.

Christodoulou, Petros. 2019. "Soft Actor-Critic for Discrete Action Settings." **arXiv Preprint arXiv:1910.07207**.

Chu, Casey, Andrey Zhmoginov, and Mark Sandler. 2017. "CycleGAN, a Master of Steganography." **arXiv Preprint arXiv:1712.02950**. https://arxiv.org/abs/1712.02950.

Chu, Xiangxiang, Liang Li, and Bo Zhang. 2024. "Make RepVGG Greater Again: A Quantization-Aware Approach." **AAAI Conference on Artificial Intelligence**. https://arxiv.org/abs/2212.01593.

Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. "Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models." **Advances in Neural Information Processing Systems** 31.

Chung, Hyungjin, Jeongsol Kim, Michael T. Mccann, Marc L. Klasky, and Jong Chul Ye. 2023. "Diffusion Posterior Sampling for General Noisy Inverse Problems." **International**

Conference on Learning Representations. https://arxiv.org/abs/2209.14687.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." **ArXiv:1412.3555**. https://arxiv.org/abs/1412.3555.

Church, Kenneth Ward, and Patrick Hanks. 1990. "Word Association Norms, Mutual Information, and Lexicography." **Computational Linguistics** 16 (1): 22–29.

Church, Kenneth W., and William A. Gale. 1995. "Poisson Mixtures." **Natural Language Engineering** 1 (2): 163–90. https://doi.org/10.1017/S1351324900000139.

Chwialkowski, Kacper, Heiko Strathmann, and Arthur Gretton. 2016. "A Kernel Test of Goodness of Fit." **Proceedings of the 33rd International Conference on Machine Learning**, 2606–15.

Coddington, Earl A., and Norman Levinson. 1955. **Theory of Ordinary Differential Equations**. McGraw-Hill.

Cohen, Jeremy M., Behrooz Ghorbani, Shankar Krishnan, et al. 2022. "Adaptive Gradient Methods at the Edge of Stability." **arXiv Preprint arXiv:2207.14484**. https://arxiv.org/abs/2207.14484.

Cohen, Jeremy M., Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. 2021. "Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability." **International Conference on Learning Representations**.

Cohen, Joel E., and Charles M. Newman. 1984. "The Stability of Large Random Matrices and Their Products." **The Annals of Probability** 12 (2): 283–310. https://doi.org/10.1214/aop/1176993291.

Conmy, Arthur, Augustine N. Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. "Towards Automated Circuit Discovery for Mechanistic Interpretability." **Advances in Neural Information Processing Systems** 36. https://arxiv.org/abs/2304.14997.

Cover, Thomas M., and Peter E. Hart. 1967. "Nearest Neighbor Pattern Classification." **IEEE Transactions on Information Theory** 13 (1): 21–27.

Cover, T, and JM Thomas. 1999. **Elements of Information Theory**. John Wiley & Sons. https://doi.org/10.1002/047174882X.

Cramér, H. 1946. **Mathematical Methods of Statistics.** Princeton University Press.

Crowson, Katherine, Stefan Andreas Baumann, Alex Birch, Tanishq Mathew Abraham, Daniel Z. Kaplan, and Enrico Shippole. 2024. "Scalable High-Resolution Pixel-Space Image Synthesis with Hourglass Diffusion Transformers." **Proceedings of the 41st International Conference on Machine Learning**, Proceedings of machine learning research, vol. 235: 9550–75.

Csiszár, Imre. 1967. “Information-Type Measures of Difference of Probability Distributions and Indirect Observations.” **Studia Scientiarum Mathematicarum Hungarica** 2: 299–318.

Csiszár, Imre. 2008. “Axiomatic Characterizations of Information Measures.” **Entropy** 10 (3): 261–73. https://doi.org/10.3390/e10030261.

Cubuk, Ekin D., Barret Zoph, Jonathon Shlens, and Quoc V. Le. 2020. “RandAugment: Practical Automated Data Augmentation with a Reduced Search Space.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops**. https://arxiv.org/abs/1909.13719.

Cunningham, Hoagy, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. 2024. “Sparse Autoencoders Find Highly Interpretable Features in Language Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2309.08600.

Cuturi, Marco. 2013. “Sinkhorn Distances: Lightspeed Computation of Optimal Transport.” **Advances in Neural Information Processing Systems** 26.

Cybenko, George. 1989. “Approximation by Superpositions of a Sigmoidal Function.” **Mathematics of Control, Signals and Systems** 2 (4): 303–14. https://doi.org/10.1007/BF02551274.

D’Angelo, Francesco, Maksym Andriushchenko, Aditya Varre, and Nicolas Flammarion. 2024. “Why Do We Need Weight Decay in Modern Deep Learning?” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2310.04415.

D’Oro, Pierluca, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G. Bellemare, and Aaron Courville. 2023. “Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier.” **International Conference on Learning Representations**.

Dahl, George E., Frank Schneider, Zachary Nado, et al. 2023. “Benchmarking Neural Network Training Algorithms.” **arXiv Preprint arXiv:2306.07179**.

Dahlquist, Germund G. 1963. “A Special Stability Problem for Linear Multistep Methods.” **BIT Numerical Mathematics** 3: 27–43.

Dai, Damai, Chengqi Deng, Chenggang Zhao, et al. 2024. “DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models.” **arXiv Preprint arXiv:2401.06066**.

Dalal, Navneet, and Bill Triggs. 2005. “Histograms of Oriented Gradients for Human Detection.” **2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)** 1: 886–93. https://doi.org/10.1109/cvpr.2005.177.

Damian, Alex, Eshaan Nichani, and Jason D. Lee. 2023. “Self-Stabilization: The Implicit Bias of Gradient Descent at the Edge of Stability.” **International Conference on Learning Representations**.

Dao, Tri. 2024. “FlashAttention-2: Faster Attention with Better Parallelism and Work Parti-

tioning." **International Conference on Learning Representations**. https://arxiv.org/abs/2307.08691.

Dao, Tri, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness." **Advances in Neural Information Processing Systems**.

Dao, Tri, and Albert Gu. 2024. "Transformers Are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality." **International Conference on Machine Learning**, 10041–71. https://arxiv.org/abs/2405.21060.

Darmois, Georges. 1935. "Sur Les Lois de Probabilité à Estimation Exhaustive." **Comptes Rendus de l'Académie Des Sciences** 200: 1265–66.

Dasgupta, Sanjoy, and Anupam Gupta. 2003. "An Elementary Proof of a Theorem of Johnson and Lindenstrauss." **Random Structures & Algorithms** 22 (1): 60–65.

Daskalakis, Constantinos, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. 2018. "Training GANs with Optimism." **International Conference on Learning Representations**.

Dauphin, Yann N., Angela Fan, Michael Auli, and David Grangier. 2017. "Language Modeling with Gated Convolutional Networks." **International Conference on Machine Learning**, 933–41.

Dauphin, Yann N., Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. "Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization." **Advances in Neural Information Processing Systems** 27: 2933–41.

Daxberger, Erik, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. 2021. "Laplace Redux – Effortless Bayesian Deep Learning." **Advances in Neural Information Processing Systems** 34. https://arxiv.org/abs/2106.14806.

De Bortoli, Valentin, James Thornton, Jeremy Heng, and Arnaud Doucet. 2021. "Diffusion Schrödinger Bridge with Applications to Score-Based Generative Modeling." **Advances in Neural Information Processing Systems** 34: 17695–709.

De Cock, Dean. 2011. "Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project." **Journal of Statistics Education** 19 (3). https://doi.org/10.1080/10691898.2011.11889627.

De, Soham, and Samuel L. Smith. 2020. "Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks." **Advances in Neural Information Processing Systems** 33. https://arxiv.org/abs/2002.10444.

De, Soham, Samuel L. Smith, Anushan Fernando, et al. 2024. "Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models." **arXiv Preprint arXiv:2402.19427**.

Dean, David S., and Satya N. Majumdar. 2006. “Large Deviations of Extreme Eigenvalues of Random Matrices.” **Physical Review Letters** 97 (16): 160201.

Dean, Jeffrey, Greg S Corrado, Rajat Monga, et al. 2012. “Large Scale Distributed Deep Networks.” **Proceedings of the 25th International Conference on Neural Information Processing Systems, Volume 1**, 1223–31. https://doi.org/10.5555/2999134.2999271.

DeepSeek-AI. 2024. “DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model.” **arXiv Preprint arXiv:2405.04434**.

DeepSeek-AI. 2025. “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.” **Nature**, ahead of print. https://doi.org/10.1038/s41586-025-09422-z.

DeepSeek-AI, Xiao Bi, Deli Chen, Guanting Chen, et al. 2024. “DeepSeek LLM: Scaling Open-Source Language Models with Longtermism.” **arXiv Preprint arXiv:2401.02954**.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, et al. 2024. “DeepSeek-V3 Technical Report.” **arXiv Preprint arXiv:2412.19437**.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, et al. 2025. “DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models.” **arXiv Preprint arXiv:2512.02556**.

Defazio, Aaron, Francis Bach, and Simon Lacoste-Julien. 2014. “SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives.” **Advances in Neural Information Processing Systems** 27.

Defazio, Aaron, Ashok Cutkosky, Harsh Mehta, and Konstantin Mishchenko. 2023. “Optimal Linear Decay Learning Rate Schedules and Further Refinements.” **arXiv Preprint arXiv:2310.07831**.

Defazio, Aaron, Xingyu Alice Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. 2024. “The Road Less Scheduled.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2405.15682.

Dehghani, Mostafa, Josip Djolonga, Basil Mustafa, et al. 2023. “Scaling Vision Transformers to 22 Billion Parameters.” **International Conference on Machine Learning**, 7480–512.

Delétang, Grégoire, Anian Ruoss, Paul-Ambroise Duquenne, et al. 2023. “Language Modeling Is Compression.” **ArXiv:2309.10668**. https://arxiv.org/abs/2309.10668.

Dempster, Arthur P, Nan M Laird, and Donald B Rubin. 1977. “Maximum Likelihood from Incomplete Data via the EM Algorithm.” **Journal of the Royal Statistical Society: Series B** 39 (1): 1–38.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. “Imagenet: A Large-Scale Hierarchical Image Database.” **2009 IEEE Conference on Computer Vision and Pattern Recognition**, 248–55. https://doi.org/10.1109/cvpr.2009.5206848.

Der Kiureghian, Armen, and Ove Ditlevsen. 2009. “Aleatory or Epistemic? Does It Matter?”

Structural Safety 31 (2): 105–12. https://doi.org/10.1016/j.strusafe.2008.06.020.

Deshpande, Ishan, Ziyu Zhang, and Alexander G. Schwing. 2018. "Generative Modeling Using the Sliced Wasserstein Distance." **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**.

Dettmers, Tim, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. "8-Bit Optimizers via Block-Wise Quantization." **International Conference on Learning Representations**. https://arxiv.org/abs/2110.02861.

Dettmers, Tim, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. "QLoRA: Efficient Finetuning of Quantized LLMs." **Advances in Neural Information Processing Systems** 36.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." **ArXiv:1810.04805**. https://arxiv.org/abs/1810.04805.

Dey, Nolan, Quentin Anthony, and Joel Hestness. 2024. **The Practitioner's Guide to the Maximal Update Parameterization**. Https://blog.eleuther.ai/mutransfer/.

Dey, Nolan, Gurpreet Gosal, Zhiming Chen, et al. 2023. "Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster." **arXiv Preprint arXiv:2304.03208**.

Dey, Nolan, Bin Claire Zhang, Lorenzo Noci, et al. 2025. "Don't Be Lazy: CompleteP Enables Compute-Efficient Deep Transformers." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2505.01618.

Dhariwal, Prafulla, and Alex Nichol. 2021. "Diffusion Models Beat GANs on Image Synthesis." **Advances in Neural Information Processing Systems** 34.

DiCiccio, Thomas J., and Bradley Efron. 1996. "Bootstrap Confidence Intervals." **Statistical Science** 11 (3): 189–228. https://doi.org/10.1214/ss/1032280214.

Dieleman, Sander. 2024. **Diffusion Is Spectral Autoregression**. Blog post, 2 September 2024, https://sander.ai/2024/09/02/spectral-autoregression.html.

Dieleman, Sander. 2025. **Generative Modelling in Latent Space**. Blog post, 15 April 2025, https://sander.ai/2025/04/15/latents.html.

Dieleman, Sander. 2026a. **Continuous Diffusion Language Models**. Blog post, 24 August 2026, https://sander.ai/2026/08/24/continuous-dlms.html.

Dieleman, Sander. 2026b. **Learning the Integral of a Diffusion Model**. Blog post, 6 May 2026, https://sander.ai/2026/05/06/flow-maps.html.

Ding, Ming, Zhuoyi Yang, Wenyi Hong, et al. 2021. "CogView: Mastering Text-to-Image Generation via Transformers." **Advances in Neural Information Processing Systems** 34.

Ding, Xiaohan, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. 2021. “RepVGG: Making VGG-Style ConvNets Great Again.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. https://arxiv.org/abs/2101.03697.

Ding, Xiaohan, Xiangyu Zhang, Yizhuang Zhou, Jungong Han, Guiguang Ding, and Jian Sun. 2022. “Scaling up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. https://arxiv.org/abs/2203.06717.

Ding, Yiran, Li Lyna Zhang, Chengruidong Zhang, et al. 2024. “LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens.” **International Conference on Machine Learning**, Proceedings of machine learning research, vol. 235: 11091–104. https://arxiv.org/abs/2402.13753.

Dinh, Laurent, David Krueger, and Yoshua Bengio. 2014. “NICE: Non-Linear Independent Components Estimation.” **ArXiv:1410.8516**. https://arxiv.org/abs/1410.8516.

Dinh, Laurent, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. 2017. “Sharp Minima Can Generalize for Deep Nets.” **Proceedings of the 34th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 70: 1019–28. https://proceedings.mlr.press/v70/dinh17b.html.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio. 2017. “Density Estimation Using Real NVP.” **International Conference on Learning Representations**. https://openreview.net/forum?id=HkpbnH9lx.

Domingos, Pedro, and Michael Pazzani. 1997. “On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss.” **Machine Learning** 29: 103–30.

Dong, Juechu, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. 2025. “Flex Attention: A Programming Model for Generating Optimized Attention Kernels.” **Proceedings of Machine Learning and Systems (MLSys)**. https://arxiv.org/abs/2412.05496.

Donsker, Monroe D., and S. R. Srinivasa Varadhan. 1983. “Asymptotic Evaluation of Certain Markov Process Expectations for Large Time. IV.” **Communications on Pure and Applied Mathematics** 36 (2): 183–212.

Dormand, John R., and Peter J. Prince. 1980. “A Family of Embedded Runge–Kutta Formulae.” **Journal of Computational and Applied Mathematics** 6 (1): 19–26.

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, et al. 2021. “An Image Is Worth 16 x 16 Words: Transformers for Image Recognition at Scale.” **International Conference on Learning Representations**. https://openreview.net/forum?id=YicbFdNTTy.

Dowson, D. C., and B. V. Landau. 1982. “The Fréchet Distance Between Multivariate Normal Distributions.” **Journal of Multivariate Analysis** 12 (3): 450–55.

Draxler, Felix, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. 2018. “Essentially No Barriers in Neural Network Energy Landscape.” **Proceedings of the 35th Inter-**

national Conference on Machine Learning, Proceedings of machine learning research, vol. 80: 1309–18.

Drori, Yoel, and Marc Teboulle. 2014. "Performance of First-Order Methods for Smooth Convex Minimization: A Novel Approach." **Mathematical Programming** 145 (1–2): 451–82. https://doi.org/10.1007/s10107-013-0653-0.

Du, Yilun, and Igor Mordatch. 2019. "Implicit Generation and Modeling with Energy Based Models." **Advances in Neural Information Processing Systems** 32. https://arxiv.org/abs/1903.08689.

Duane, Simon, A. D. Kennedy, Brian J. Pendleton, and Duncan Roweth. 1987. "Hybrid Monte Carlo." **Physics Letters B** 195 (2): 216–22.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." **Journal of Machine Learning Research** 12: 2121–59. https://jmlr.org/papers/v12/duchi11a.html.

Duchi, John, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. "Efficient Projections onto the ℓ_1-Ball for Learning in High Dimensions." **Proceedings of the 25th International Conference on Machine Learning (ICML)**, 272–79.

Duda, Jarek. 2013. **Asymmetric Numeral Systems: Entropy Coding Combining Speed of Huffman Coding with Compression Rate of Arithmetic Coding**. arXiv:1311.2540. https://arxiv.org/abs/1311.2540.

Dupont, Emilien, Arnaud Doucet, and Yee Whye Teh. 2019. "Augmented Neural ODEs." **Advances in Neural Information Processing Systems** 32.

Dwork, Cynthia, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. 2015. "Preserving Statistical Validity in Adaptive Data Analysis." **ACM Symposium on Theory of Computing**, 117–26.

Dziugaite, Gintare Karolina, and Daniel M. Roy. 2017. "Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters Than Training Data." **Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)**. https://arxiv.org/abs/1703.11008.

Eckart, Carl, and Gale Young. 1936. "The Approximation of One Matrix by Another of Lower Rank." **Psychometrika** 1 (3): 211–18.

Efron, Bradley. 1979. "Bootstrap Methods: Another Look at the Jackknife." **Annals of Statistics** 7 (1): 1–26.

Efron, Bradley. 2011. "Tweedie's Formula and Selection Bias." **Journal of the American Statistical Association** 106 (496): 1602–14.

Elhage, Nelson, Tristan Hume, Catherine Olsson, et al. 2022. "Toy Models of Superposition." **Transformer Circuits Thread**.

Elhage, Nelson, Neel Nanda, Catherine Olsson, et al. 2021. “A Mathematical Framework for Transformer Circuits.” **Transformer Circuits Thread**.

Elman, Jeffrey L. 1990. “Finding Structure in Time.” **Cognitive Science** 14 (2): 179–211. https://doi.org/10.1016/0364-0213(90)90002-E.

Elsayed, Mohamed, Gautham Vasan, and A Rupam Mahmood. 2024. “Streaming Deep Reinforcement Learning Finally Works.” **arXiv Preprint arXiv:2410.14606**.

Emmons, Scott, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. 2022. “RvS: What Is Essential for Offline RL via Supervised Learning?” **International Conference on Learning Representations**.

Endres, Dominik M., and Johannes E. Schindelin. 2003. “A New Metric for Probability Distributions.” **IEEE Transactions on Information Theory** 49 (7): 1858–60.

Engstrom, Logan, Andrew Ilyas, Shibani Santurkar, et al. 2020. “Implementation Matters in Deep RL: A Case Study on PPO and TRPO.” **International Conference on Learning Representations**.

Entezari, Rahim, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. 2022. “The Role of Permutation Invariance in Linear Mode Connectivity of Neural Networks.” **International Conference on Learning Representations**.

Erven, Tim van, and Peter Harremoës. 2014. “Rényi Divergence and Kullback–Leibler Divergence.” **IEEE Transactions on Information Theory** 60 (7): 3797–820.

Espeholt, Lasse, Hubert Soyer, Rémi Munos, et al. 2018. “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.” **International Conference on Machine Learning**, 1407–16.

Esser, Patrick, Sumith Kulal, Andreas Blattmann, et al. 2024. “Scaling Rectified Flow Transformers for High-Resolution Image Synthesis.” **Proceedings of the 41st International Conference on Machine Learning**, Proceedings of machine learning research, vol. 235: 12606–33. https://arxiv.org/abs/2403.03206.

Esser, Patrick, Robin Rombach, and Björn Ommer. 2021. “Taming Transformers for High-Resolution Image Synthesis.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**.

Even-Dar, Eyal, and Yishay Mansour. 2003. “Learning Rates for Q-Learning.” **Journal of Machine Learning Research** 5: 1–25.

Falcon, William, and The PyTorch Lightning team. 2019. **PyTorch Lightning**. https://doi.org/10.5281/zenodo.3828935.

Fan, Angela, Mike Lewis, and Yann Dauphin. 2018. “Hierarchical Neural Story Generation.” **Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics**, 889–98.

Farebrother, Jesse, Jordi Orbay, Quan Vuong, et al. 2024. “Stop Regressing: Training Value Functions via Classification for Scalable Deep RL.” **International Conference on Machine Learning**. https://arxiv.org/abs/2403.03950.

Fedus, William, Barret Zoph, and Noam Shazeer. 2022. “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.” **Journal of Machine Learning Research** 23 (120): 1–39. https://arxiv.org/abs/2101.03961.

Feng, Leo, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadeghi. 2024. “Were RNNs All We Needed?” **arXiv Preprint arXiv:2410.01201**.

Feydy, Jean, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouvé, and Gabriel Peyré. 2019. “Interpolating Between Optimal Transport and MMD Using Sinkhorn Divergences.” **Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics**, 2681–90.

Field, David J. 1987. “Relations Between the Statistics of Natural Images and the Response Properties of Cortical Cells.” **JOSA A** 4 (12): 2379–94. https://doi.org/10.1364/josaa.4.002379.

Finlay, Chris, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman. 2020. “How to Train Your Neural ODE: The World of Jacobian and Kinetic Regularization.” **International Conference on Machine Learning**.

Fisher, Ronald A. 1925. “Theory of Statistical Estimation.” **Mathematical Proceedings of the Cambridge Philosophical Society** 22 (5): 700–725.

Fisher, Ronald A. 1935. **The Design of Experiments**. Oliver; Boyd.

Fisher, Ronald A. 1936. “The Use of Multiple Measurements in Taxonomic Problems.” **Annals of Eugenics** 7 (2): 179–88. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.

Fokker, Adriaan D. 1914. “Die Mittlere Energie Rotierender Elektrischer Dipole Im Strahlungsfeld.” **Annalen Der Physik** 348 (5): 810–20.

Folland, Gerald B. 1999. **Real Analysis: Modern Techniques and Their Applications**. 2nd ed. Wiley.

Foret, Pierre, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. “Sharpness-Aware Minimization for Efficiently Improving Generalization.” **International Conference on Learning Representations**. https://arxiv.org/abs/2010.01412.

Frank, Marguerite, and Philip Wolfe. 1956. “An Algorithm for Quadratic Programming.” **Naval Research Logistics Quarterly** 3 (1–2): 95–110. https://doi.org/10.1002/nav.3800030109.

Frans, Kevin, Danijar Hafner, Sergey Levine, and Pieter Abbeel. 2025. “One Step Diffusion via Shortcut Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.12557.

Frazier, Peter I., Shane G. Henderson, and Rolf Waeber. 2019. “Probabilistic Bisection Converges Almost as Quickly as Stochastic Approximation.” **Mathematics of Operations Research** 44 (2): 651–67. https://doi.org/10.1287/moor.2018.0938.

Frey, Brendan J., and Geoffrey E. Hinton. 1997. “Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model.” **The Computer Journal** 40 (2/3): 157–65.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. “Regularization Paths for Generalized Linear Models via Coordinate Descent.” **Journal of Statistical Software** 33 (1): 1–22.

Frostig, Roy, Matthew James Johnson, and Chris Leary. 2018. “Compiling Machine Learning Programs via High-Level Tracing.” In **Proceedings of Systems for Machine Learning**. https://mlsys.org/Conferences/2018/doc/19.pdf.

Fu, Daniel Y., Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. 2023. “Hungry Hungry Hippos: Towards Language Modeling with State Space Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2212.14052.

Fujimoto, Scott, and Shixiang Shane Gu. 2021. “A Minimalist Approach to Offline Reinforcement Learning.” **Advances in Neural Information Processing Systems** 34.

Fujimoto, Scott, Herke van Hoof, and David Meger. 2018. “Addressing Function Approximation Error in Actor-Critic Methods.” **International Conference on Machine Learning**, 1587–96.

Fujimoto, Scott, David Meger, and Doina Precup. 2019. “Off-Policy Deep Reinforcement Learning Without Exploration.” **International Conference on Machine Learning**, 2052–62.

Fukushima, Kunihiko. 1980. “Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.” **Biological Cybernetics** 36 (4): 193–202. https://doi.org/10.1007/BF00344251.

Furstenberg, Harry, and Harry Kesten. 1960. “Products of Random Matrices.” **The Annals of Mathematical Statistics** 31 (2): 457–69.

Gage, Philip. 1994. “A New Algorithm for Data Compression.” **The C Users Journal** 12 (2): 23–38.

Gal, Yarin, and Zoubin Ghahramani. 2016. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning.” **International Conference on Machine Learning**, 1050–59. https://arxiv.org/abs/1506.02142.

Gallici, Matteo, Mattie Fellows, Benjamin Ellis, et al. 2025. “Simplifying Deep Temporal Difference Learning.” **International Conference on Learning Representations**. https://arxiv.org/abs/2407.04811.

Gao, Leo, John Schulman, and Jacob Hilton. 2023. “Scaling Laws for Reward Model Overoptimization.” **International Conference on Machine Learning**. https://arxiv.org/abs/2210.10760.

Gao, Ruiqi, Emiel Hoogeboom, Jonathan Heek, Valentin De Bortoli, Kevin P. Murphy, and Tim Salimans. 2024. **Diffusion Meets Flow Matching: Two Sides of the Same Coin**. Blog post, https://diffusionflow.github.io/.

Garg, Shivam, Dimitris Tsipras, Percy Liang, and Gregory Valiant. 2022. “What Can Transformers Learn in-Context? A Case Study of Simple Function Classes.” **Advances in Neural Information Processing Systems** 35. https://arxiv.org/abs/2208.01066.

Garipov, Timur, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. 2018. “Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs.” **Advances in Neural Information Processing Systems** 31: 8789–98.

Gat, Itai, Tal Remez, Neta Shaul, et al. 2024. “Discrete Flow Matching.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2407.15595.

Gauss, Carl Friedrich. 1809. “Theoria Motus Corporum Coelestium.” In **Werke**. Königlich Preussische Akademie der Wissenschaften. https://doi.org/10.1007/978-3-642-92478-1.

Gavish, Matan, and David L. Donoho. 2014. “The Optimal Hard Threshold for Singular Values Is $4/\sqrt{3}$.” **IEEE Transactions on Information Theory** 60 (8): 5040–53.

Geman, Stuart. 1986. “The Spectral Radius of Large Random Matrices.” **The Annals of Probability** 14 (4): 1318–28.

Geman, Stuart, Elie Bienenstock, and René Doursat. 1992. “Neural Networks and the Bias/Variance Dilemma.” **Neural Computation** 4 (1): 1–58.

Geman, Stuart, and Donald Geman. 1984. “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.” **IEEE Transactions on Pattern Analysis and Machine Intelligence** 6 (6): 721–41.

Gemma Team. 2024. “Gemma 2: Improving Open Language Models at a Practical Size.” **arXiv Preprint arXiv:2408.00118**.

Gemma Team. 2025. “Gemma 3 Technical Report.” **arXiv Preprint arXiv:2503.19786**.

Genevay, Aude, Gabriel Peyré, and Marco Cuturi. 2018. “Learning Generative Models with Sinkhorn Divergences.” **Proceedings of the 21st International Conference on Artificial Intelligence and Statistics**.

Geng, Zhengyang, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. 2025. “Mean Flows for One-Step Generative Modeling.” **arXiv Preprint arXiv:2505.13447**.

Geng, Zhengyang, Yiyang Lu, Zongze Wu, Eli Shechtman, J. Zico Kolter, and Kaiming He. 2026. “Improved Mean Flows: On the Challenges of Fastforward Generative Models.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recogni-**

tion. https://arxiv.org/abs/2512.02012.

Geng, Zhengyang, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. 2021. "On Training Implicit Models." **Advances in Neural Information Processing Systems** 34. https://arxiv.org/abs/2111.05177.

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. 2000. "Learning to Forget: Continual Prediction with LSTM." **Neural Computation** 12 (10): 2451–71. https://doi.org/10.1162/089976600300015015.

Gershgorin, Semyon A. 1931. "Über Die Abgrenzung Der Eigenwerte Einer Matrix." **Izvestiya Akademii Nauk SSSR, Seriya Matematicheskaya**, no. 6: 749–54.

Geshkovski, Borjan, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. 2023. "A Mathematical Perspective on Transformers." **arXiv Preprint arXiv:2312.10794**.

Geweke, John. 1989. "Bayesian Inference in Econometric Models Using Monte Carlo Integration." **Econometrica** 57 (6): 1317–39. https://doi.org/10.2307/1913710.

Ghadimi, Saeed, and Guanghui Lan. 2013. "Stochastic First- and Zeroth-Order Methods for Nonconvex Stochastic Programming." **SIAM Journal on Optimization** 23 (4): 2341–68. https://arxiv.org/abs/1309.5549.

Gholami, Amir, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer. 2024. "AI and Memory Wall." **IEEE Micro** 44 (3): 33–39. https://doi.org/10.1109/MM.2024.3373763.

Ghorbani, Behrooz, Shankar Krishnan, and Ying Xiao. 2019. "An Investigation into Neural Net Optimization via Hessian Eigenvalue Density." **Proceedings of the 36th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 97: 2232–41.

Ghosh, Dhruba, Hanna Hajishirzi, and Ludwig Schmidt. 2023. "GenEval: An Object-Focused Framework for Evaluating Text-to-Image Alignment." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2310.11513.

Gibbs, Josiah Willard. 1902. **Elementary Principles of Statistical Mechanics**. Scribner's.

Gidel, Gauthier, Reyhane Askari Hemmat, Mohammad Pezeshki, et al. 2019. "Negative Momentum for Improved Game Dynamics." **Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics**, Proceedings of machine learning research, vol. 89: 1802–11.

Giles, Michael B. 2008. "Multilevel Monte Carlo Path Simulation." **Operations Research** 56 (3): 607–17.

Giles, Mike B. 2008. **An Extended Collection of Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation**. NA-08/01. Oxford University Computing Laboratory.

Ginibre, Jean. 1965. “Statistical Ensembles of Complex, Quaternion, and Real Matrices.” **Journal of Mathematical Physics** 6 (3): 440–49. https://doi.org/10.1063/1.1704292.

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 580–87. https://doi.org/10.1109/CVPR.2014.81.

Givens, Clark R., and Rae Michael Shortt. 1984. “A Class of Wasserstein Metrics for Probability Distributions.” **Michigan Mathematical Journal** 31 (2): 231–40.

Glorot, Xavier, and Yoshua Bengio. 2010. “Understanding the Difficulty of Training Deep Feedforward Neural Networks.” **Proceedings of the 13th International Conference on Artificial Intelligence and Statistics**, 249–56. https://proceedings.mlr.press/v9/glorot10a.html.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. 2011. “Deep Sparse Rectifier Neural Networks.” **Proceedings of the 14th International Conference on Artificial Intelligence and Statistics**, 315–23. https://proceedings.mlr.press/v15/glorot11a.html.

Gnedenko, B. V., and A. N. Kolmogorov. 1954. **Limit Distributions for Sums of Independent Random Variables**. Addison-Wesley.

Gneiting, Tilmann, Fadoua Balabdaoui, and Adrian E. Raftery. 2007. “Probabilistic Forecasts, Calibration and Sharpness.” **Journal of the Royal Statistical Society Series B: Statistical Methodology** 69 (2): 243–68. https://doi.org/10.1111/j.1467-9868.2007.00587.x.

Gneiting, Tilmann, and Adrian E. Raftery. 2007. “Strictly Proper Scoring Rules, Prediction, and Estimation.” **Journal of the American Statistical Association** 102 (477): 359–78. https://doi.org/10.1198/016214506000001437.

Godbole, Varun, George E. Dahl, Justin Gilmer, Christopher J. Shallue, and Zachary Nado. 2023. **Deep Learning Tuning Playbook**. Https://github.com/google-research/tuning_playbook.

Goel, Karan, Albert Gu, Chris Donahue, and Christopher Ré. 2022. “It’s Raw! Audio Generation with State-Space Models.” **International Conference on Machine Learning**. https://arxiv.org/abs/2202.09729.

Goldberg, David. 1991. “What Every Computer Scientist Should Know about Floating-Point Arithmetic.” **ACM Computing Surveys** 23 (1): 5–48.

Goldfeld, Ziv, Ewout van den Berg, Kristjan Greenewald, et al. 2019. “Estimating Information Flow in Deep Neural Networks.” **Proceedings of the 36th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 97: 2299–308.

Golub, Gene H, and Charles F Van Loan. 1996. **Matrix Computations**. Johns Hopkins University Press. https://jhupbooks.press.jhu.edu/title/matrix-computations.

Goodfellow, Ian. 2016. “NIPS 2016 Tutorial: Generative Adversarial Networks.” **arXiv**

Preprint arXiv:1701.00160. https://arxiv.org/abs/1701.00160.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. **Deep Learning**. MIT Press.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, et al. 2014. "Generative Adversarial Nets." **Advances in Neural Information Processing Systems**, 2672–80. https://doi.org/10.5555/2969033.2969125.

Goyal, Priya, Piotr Dollár, Ross Girshick, et al. 2017. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour." **ArXiv:1706.02677**. https://arxiv.org/abs/1706.02677.

Grathwohl, Will, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. 2019. "FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models." **International Conference on Learning Representations**.

Grattafiori, Aaron, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. "The Llama 3 Herd of Models." **arXiv Preprint arXiv:2407.21783**.

Graves, Alex. 2013. "Generating Sequences with Recurrent Neural Networks." **ArXiv:1308.0850**. https://arxiv.org/abs/1308.0850.

Grazzi, Riccardo, Julien Siems, Jörg K. H. Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. 2025. "Unlocking State-Tracking in Linear RNNs Through Negative Eigenvalues." **International Conference on Learning Representations**. https://arxiv.org/abs/2411.12537.

Greensmith, Evan, Peter L Bartlett, and Jonathan Baxter. 2004. "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning." **Journal of Machine Learning Research** 5: 1471–530.

Gretton, Arthur, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. "A Kernel Two-Sample Test." **Journal of Machine Learning Research** 13: 723–73.

Griewank, Andreas. 1992. "Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation." **Optimization Methods and Software** 1 (1): 35–54.

Griewank, Andreas, and Andrea Walther. 2008. **Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation**. 2nd ed. Society for Industrial; Applied Mathematics (SIAM). https://doi.org/10.1137/1.9780898717761.

Grinsztajn, Léo, Edouard Oyallon, and Gaël Varoquaux. 2022. "Why Do Tree-Based Models Still Outperform Deep Learning on Tabular Data?" **Advances in Neural Information Processing Systems** 35: 507–20. https://arxiv.org/abs/2207.08815.

Grobman, David M. 1959. "Homeomorphism of Systems of Differential Equations." **Doklady Akademii Nauk SSSR** 128: 880–81.

Grünwald, Peter D. 2007. **The Minimum Description Length Principle**. MIT Press. https:

//doi.org/10.7551/mitpress/4643.001.0001.

Gu, Albert. 2023. “Modeling Sequences with Structured State Spaces.” PhD thesis, Stanford University.

Gu, Albert, and Tri Dao. 2023. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces.” **arXiv Preprint arXiv:2312.00752**.

Gu, Albert, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. 2020. “HiPPO: Recurrent Memory with Optimal Polynomial Projections.” **Advances in Neural Information Processing Systems** 33: 1474–87.

Gu, Albert, Karan Goel, and Christopher Ré. 2022. “Efficiently Modeling Long Sequences with Structured State Spaces.” **International Conference on Learning Representations**. https://arxiv.org/abs/2111.00396.

Gu, Albert, Ankit Gupta, Karan Goel, and Christopher Ré. 2022. “On the Parameterization and Initialization of Diagonal State Space Models.” **Advances in Neural Information Processing Systems** 35: 35971–83.

Gu, Albert, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré. 2023. “How to Train Your HiPPO: State Space Models with Generalized Orthogonal Basis Projections.” **International Conference on Learning Representations**. https://arxiv.org/abs/2206.12037.

Gu, Xiangming, Tianyu Pang, Chao Du, et al. 2025. “When Attention Sink Emerges in Language Models: An Empirical View.” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.10781.

Gulati, Anmol, James Qin, Chung-Cheng Chiu, et al. 2020. “Conformer: Convolution-Augmented Transformer for Speech Recognition.” **Proc. Interspeech 2020**, 5036–40. https://doi.org/10.21437/interspeech.2020-3015.

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. “Improved Training of Wasserstein GANs.” **Advances in Neural Information Processing Systems** 30.

Gumbel, Emil J. 1954. **Statistical Theory of Extreme Values and Some Practical Applications**. U.S. National Bureau of Standards.

Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. “On Calibration of Modern Neural Networks.” **International Conference on Machine Learning**, 1321–30. https://arxiv.org/abs/1706.04599.

Guo, Han, Songlin Yang, Tarushii Goel, Eric P. Xing, Tri Dao, and Yoon Kim. 2025. “Log-Linear Attention.” **arXiv Preprint arXiv:2506.04761**.

Gupta, Suyog, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. “Deep Learning with Limited Numerical Precision.” **International Conference on Machine Learning**, 1737–46.

Gupta, Vineet, Tomer Koren, and Yoram Singer. 2018. “Shampoo: Preconditioned Stochastic Tensor Optimization.” **Proceedings of the 35th International Conference on Machine Learning (ICML)**, 1842–50.

Gustafson, John L. 1988. “Reevaluating Amdahl’s Law.” **Communications of the ACM** 31 (5): 532–33.

Gutmann, Michael U., and Aapo Hyvärinen. 2012. “Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics.” **Journal of Machine Learning Research** 13: 307–61. https://www.jmlr.org/papers/v13/gutmann12a.html.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.” **International Conference on Machine Learning**, 1861–70.

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, et al. 2018. “Soft Actor-Critic Algorithms and Applications.” **arXiv Preprint arXiv:1812.05905**.

Hafner, Danijar, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. 2025. “Mastering Diverse Control Tasks Through World Models.” **Nature** 640 (8059): 647–53. https://doi.org/10.1038/s41586-025-08744-2.

Hägele, Alexander, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro von Werra, and Martin Jaggi. 2024. “Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2405.18392.

Hairer, Ernst, Christian Lubich, and Gerhard Wanner. 2006. **Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations**. 2nd ed. Vol. 31. Springer Series in Computational Mathematics. Springer.

Hairer, Ernst, Syvert P. Nørsett, and Gerhard Wanner. 1993. **Solving Ordinary Differential Equations I: Nonstiff Problems**. 2nd ed. Springer.

Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. “Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions.” **SIAM Review** 53 (2): 217–88.

Hall, Peter. 1992. **The Bootstrap and Edgeworth Expansion**. Springer Series in Statistics. Springer. https://doi.org/10.1007/978-1-4612-4384-7.

Hanin, Boris, and Mihai Nica. 2020. “Products of Many Large Random Matrices and Gradients in Deep Neural Networks.” **Communications in Mathematical Physics** 376 (1): 287–322.

Hansen, Nicklas, Hao Su, and Xiaolong Wang. 2024. “TD-MPC2: Scalable, Robust World Models for Continuous Control.” **International Conference on Learning Representations**. https://arxiv.org/abs/2310.16828.

Hansen-Estruch, Philippe, Jiahui Chen, Vivek Ramanujan, et al. 2026. “ViTok-V2: Scaling Native Resolution Auto-Encoders to 5 Billion Parameters.” **arXiv Preprint arXiv:2605.05331**.

Hansen-Estruch, Philippe, David Yan, Ching-Yao Chung, et al. 2025. “Learnings from Scaling Visual Tokenizers for Reconstruction and Generation.” **arXiv Preprint arXiv:2501.09755**.

Hardt, Moritz, Ben Recht, and Yoram Singer. 2016. “Train Faster, Generalize Better: Stability of Stochastic Gradient Descent.” **Proceedings of the 33rd International Conference on Machine Learning**, Proceedings of machine learning research, vol. 48: 1225–34. https://proceedings.mlr.press/v48/hardt16.html.

Harris, Charles R., K. Jarrod Millman, Stéfan J. van der Walt, et al. 2020. “Array Programming with NumPy.” **Nature** 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Hartman, Philip. 1960. “A Lemma in the Theory of Structural Stability of Differential Equations.” **Proceedings of the American Mathematical Society** 11 (4): 610–20.

Hasselt, Hado van. 2010. “Double Q-Learning.” **Advances in Neural Information Processing Systems** 23: 2613–21.

Hasselt, Hado van, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. 2018. “Deep Reinforcement Learning and the Deadly Triad.” **arXiv Preprint arXiv:1812.02648**.

Hastings, W. Keith. 1970. “Monte Carlo Sampling Methods Using Markov Chains and Their Applications.” **Biometrika** 57 (1): 97–109.

Haveliwala, Taher H., and Sepandar D. Kamvar. 2003. **The Second Eigenvalue of the Google Matrix**. Stanford University. http://ilpubs.stanford.edu:8090/582/.

Haviv, Adi, Ori Ram, Ofir Press, Peter Izsak, and Omer Levy. 2022. “Transformer Language Models Without Positional Encodings Still Learn Positional Information.” **Findings of the Association for Computational Linguistics: EMNLP 2022**. https://arxiv.org/abs/2203.16634.

He, Horace. 2022. **Making Deep Learning Go Brrrr from First Principles**. Https://horace.io/brrr_intro.html.

He, Kaiming, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. “Masked Autoencoders Are Scalable Vision Learners.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 16000–16009. https://doi.org/10.1109/cvpr52688.2022.01553.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.” **Proceedings of the IEEE International Conference on Computer Vision**, 1026–34. https://doi.org/10.1109/iccv.2015.123.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. “Deep Residual Learning for Image Recognition.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 770–78. https://doi.org/10.1109/cvpr.2016.90.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016b. “Identity Mappings in Deep Residual Networks.” **European Conference on Computer Vision**, 630–45. https://doi.org/10.1007/978-3-319-46493-0_38.

He, Pengcheng, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. “DeBERTa: Decoding-Enhanced BERT with Disentangled Attention.” **International Conference on Learning Representations**.

He, Tong, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2019. “Bag of Tricks for Image Classification with Convolutional Neural Networks.” **IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 558–67. https://arxiv.org/abs/1812.01187.

Hebb, Donald Olding. 1949. **The Organization of Behavior**. Wiley.

Heimersheim, Stefan, and Neel Nanda. 2024. “How to Use and Interpret Activation Patching.” **arXiv Preprint arXiv:2404.15255**.

Held, Michael, Philip Wolfe, and Harlan P. Crowder. 1974. “Validation of Subgradient Optimization.” **Mathematical Programming** 6: 62–88.

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2018. “Deep Reinforcement Learning That Matters.” **Proceedings of the AAAI Conference on Artificial Intelligence** 32: 3207–14.

Hendrycks, Dan, and Kevin Gimpel. 2016. “Gaussian Error Linear Units (GELUs).” **ArXiv:1606.08415**. https://arxiv.org/abs/1606.08415.

Hennessy, John L, and David A Patterson. 2011. **Computer Architecture: A Quantitative Approach**. Elsevier. https://www.elsevier.com/books/computer-architecture/hennessy/978-0-12-383872-8.

Henry, Alex, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. 2020. “Query-Key Normalization for Transformers.” **Findings of the Association for Computational Linguistics: EMNLP 2020**.

Hertrich, Johannes, Antonin Chambolle, and Julie Delon. 2025. “On the Relation Between Rectified Flows and Optimal Transport.” **arXiv Preprint arXiv:2505.19712**.

Hessel, Jack, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. 2021. “CLIPScore: A Reference-Free Evaluation Metric for Image Captioning.” **Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing**. https://arxiv.org/abs/2104.08718.

Hessel, Matteo, Joseph Modayil, Hado van Hasselt, et al. 2018. “Rainbow: Combining Improvements in Deep Reinforcement Learning.” **Proceedings of the AAAI Conference**

on Artificial Intelligence 32: 3215–22.

Hestenes, Magnus R., and Eduard Stiefel. 1952. "Methods of Conjugate Gradients for Solving Linear Systems." **Journal of Research of the National Bureau of Standards** 49 (6): 409–36.

Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." **Advances in Neural Information Processing Systems** 30. https://arxiv.org/abs/1706.08500.

Higham, Nicholas J. 2002. **Accuracy and Stability of Numerical Algorithms**. 2nd ed. SIAM.

Higham, Nicholas J., and Theo Mary. 2019. "A New Approach to Probabilistic Rounding Error Analysis." **SIAM Journal on Scientific Computing** 41 (5): A2815–35. https://doi.org/10.1137/18M1226312.

Hill, Bruce M. 1975. "A Simple General Approach to Inference about the Tail of a Distribution." **The Annals of Statistics** 3 (5): 1163–74. https://doi.org/10.1214/aos/1176343247.

Hillis, W. Daniel, and Guy L. Steele Jr. 1986. "Data Parallel Algorithms." **Communications of the ACM** 29 (12): 1170–83.

Hinton, Geoffrey E. 2002. "Training Products of Experts by Minimizing Contrastive Divergence." **Neural Computation** 14 (8): 1771–800.

Hinton, Geoffrey E., and Drew van Camp. 1993. "Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights." **Proceedings of the Sixth Annual Conference on Computational Learning Theory**, 5–13.

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors." **arXiv Preprint arXiv:1207.0580**.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. "Distilling the Knowledge in a Neural Network." **arXiv Preprint arXiv:1503.02531**.

Hiraoka, Takuya, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. 2022. "Dropout Q-Functions for Doubly Efficient Reinforcement Learning." **International Conference on Learning Representations**.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. 2020. "Denoising Diffusion Probabilistic Models." **Advances in Neural Information Processing Systems** 33: 6840–51. https://arxiv.org/abs/2006.11239.

Ho, Jonathan, and Tim Salimans. 2022. "Classifier-Free Diffusion Guidance." **arXiv Preprint arXiv:2207.12598**.

Ho, Jonathan, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. 2022. "Video Diffusion Models." **Advances in Neural Information Pro-**

cessing Systems 35. https://arxiv.org/abs/2204.03458.

Hochbruck, Marlis, and Alexander Ostermann. 2010. “Exponential Integrators.” **Acta Numerica** 19: 209–86.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long Short-Term Memory.” **Neural Computation** 9 (8): 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735.

Hockney, Roger W. 1994. “The Communication Challenge for MPP: Intel Paragon and Meiko CS-2.” **Parallel Computing** 20 (3): 389–98. https://doi.org/10.1016/S0167-8191(06)80021-9.

Hoeffding, Wassily. 1963. “Probability Inequalities for Sums of Bounded Random Variables.” **Journal of the American Statistical Association** 58 (301): 13–30.

Hoerl, Arthur E, and Robert W Kennard. 1970. “Ridge Regression: Biased Estimation for Nonorthogonal Problems.” **Technometrics** 12 (1): 55–67.

Hoffman, Matthew D., and Andrew Gelman. 2014. “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo.” **Journal of Machine Learning Research** 15: 1593–623. https://jmlr.org/papers/v15/hoffman14a.html.

Hoffmann, Jordan, Sebastian Borgeaud, Arthur Mensch, et al. 2022. “Training Compute-Optimal Large Language Models.” **Advances in Neural Information Processing Systems** 35. https://arxiv.org/abs/2203.15556.

Holderrieth, Peter, and Ezra Erives. 2025. “An Introduction to Flow Matching and Diffusion Models.” **arXiv Preprint arXiv:2506.02070**. https://arxiv.org/abs/2506.02070.

Holtzman, Ari, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. “The Curious Case of Neural Text Degeneration.” **International Conference on Learning Representations**. https://arxiv.org/abs/1904.09751.

Hoogeboom, Emiel, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. 2022. “Autoregressive Diffusion Models.” **International Conference on Learning Representations**.

Hoogeboom, Emiel, Jonathan Heek, and Tim Salimans. 2023. “Simple Diffusion: End-to-End Diffusion for High Resolution Images.” **International Conference on Machine Learning**. https://arxiv.org/abs/2301.11093.

Hoogeboom, Emiel, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. 2021. “Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions.” **Advances in Neural Information Processing Systems** 34. https://arxiv.org/abs/2102.05379.

Horn, Roger A., and Charles R. Johnson. 2012. **Matrix Analysis**. 2nd ed. Cambridge University Press.

Horowitz, Mark. 2014. “1.1 Computing’s Energy Problem (and What We Can Do about It).”

IEEE International Solid-State Circuits Conference (ISSCC), 10–14. https://doi.org/10.1109/ISSCC.2014.6757323.

Howard, Andrew G., Menglong Zhu, Bo Chen, et al. 2017. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." **ArXiv:1704.04861**. https://arxiv.org/abs/1704.04861.

Howard, Ronald A. 1960. **Dynamic Programming and Markov Processes**. The Technology Press of the Massachusetts Institute of Technology; John Wiley & Sons.

Howard, Steven R., Aaditya Ramdas, Jon McAuliffe, and Jasjeet Sekhon. 2021. "Time-Uniform, Nonparametric, Nonasymptotic Confidence Sequences." **The Annals of Statistics** 49 (2): 1055–80. https://doi.org/10.1214/20-AOS1991.

Hu, Edward J., Yelong Shen, Phillip Wallis, et al. 2021. "LoRA: Low-Rank Adaptation of Large Language Models." **arXiv Preprint arXiv:2106.09685**.

Hu, Jie, Li Shen, and Gang Sun. 2018. "Squeeze-and-Excitation Networks." **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 7132–41. https://doi.org/10.1109/cvpr.2018.00745.

Hu, Shengding, Yuge Tu, Xu Han, et al. 2024. "MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies." **ArXiv:2404.06395**. https://arxiv.org/abs/2404.06395.

Huang, Cheng-Zhi Anna, Ashish Vaswani, Jakob Uszkoreit, et al. 2018. "Music Transformer: Generating Music with Long-Term Structure." **International Conference on Learning Representations**. https://arxiv.org/abs/1809.04281.

Huang, Gao, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. "Snapshot Ensembles: Train 1, Get M for Free." **International Conference on Learning Representations**. https://arxiv.org/abs/1704.00109.

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. "Densely Connected Convolutional Networks." **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 4700–4708. https://doi.org/10.1109/cvpr.2017.243.

Huang, Gao, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. 2016. "Deep Networks with Stochastic Depth." **European Conference on Computer Vision**. https://arxiv.org/abs/1603.09382.

Huang, Shengyi, Rousslan F J Dossa, Chang Ye, et al. 2022. "CleanRL: High-Quality Single-File Implementations of Deep Reinforcement Learning Algorithms." **Journal of Machine Learning Research** 23 (274): 1–18.

Huang, Shengyi, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. 2022. **The 37 Implementation Details of Proximal Policy Optimization**. ICLR Blog Track, 25 March 2022, https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Huang, Xun, and Serge Belongie. 2017. "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization." **Proceedings of the IEEE International Conference on Computer Vision**, 1501–10. https://arxiv.org/abs/1703.06868.

Huang, Yanping, Youlong Cheng, Ankur Bapna, et al. 2019. "GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism." **Advances in Neural Information Processing Systems** 32. https://arxiv.org/abs/1811.06965.

Huang, Yiwen, Aaron Gokaslan, Volodymyr Kuleshov, and James Tompkin. 2024. "The GAN Is Dead; Long Live the GAN! A Modern GAN Baseline." **Advances in Neural Information Processing Systems** 37.

Hubel, David H, and Torsten N Wiesel. 1959. "Receptive Fields of Single Neurones in the Cat's Striate Cortex." **Journal of Physiology** 148 (3): 574–91. https://doi.org/10.1113/jphysiol.1959.sp006308.

Hubel, David H, and Torsten N Wiesel. 1962. "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex." **Journal of Physiology** 160 (1): 106–54. https://doi.org/10.1113/jphysiol.1962.sp006837.

Hubel, David H, and Torsten N Wiesel. 1968. "Receptive Fields and Functional Architecture of Monkey Striate Cortex." **Journal of Physiology** 195 (1): 215–43. https://doi.org/10.1113/jphysiol.1968.sp008455.

Huffman, David A. 1952. "A Method for the Construction of Minimum-Redundancy Codes." **Proceedings of the IRE** 40 (9): 1098–101. https://doi.org/10.1109/JRPROC.1952.273898.

Hutchinson, Michael F. 1989. "A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines." **Communications in Statistics—Simulation and Computation** 18 (3): 1059–76.

Hyvärinen, Aapo. 2005. "Estimation of Non-Normalized Statistical Models by Score Matching." **Journal of Machine Learning Research** 6: 695–709.

IEEE. 2019. **IEEE Standard for Floating-Point Arithmetic**. IEEE Std 754-2019.

Ilic, Aleksandar, Frederico Pratas, and Leonel Sousa. 2014. "Cache-Aware Roofline Model: Upgrading the Loft." **IEEE Computer Architecture Letters** 13 (1): 21–24. https://doi.org/10.1109/L-CA.2013.6.

Inan, Hakan, Khashayar Khosravi, and Richard Socher. 2017. "Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling." **International Conference on Learning Representations**.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." **International Conference on Machine Learning**, 448–56. https://arxiv.org/abs/1502.03167.

Isola, Phillip, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. "Image-to-Image Translation with Conditional Adversarial Networks." **Proceedings of the IEEE Conference on**

Computer Vision and Pattern Recognition, 1125–34. https://doi.org/10.1109/cvpr.2017.632.

Itô, Kiyosi. 1944. “Stochastic Integral.” **Proceedings of the Imperial Academy** 20 (8): 519–24.

Ivanov, Andrei, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. 2021. “Data Movement Is All You Need: A Case Study on Optimizing Transformers.” **Proceedings of Machine Learning and Systems** 3. https://arxiv.org/abs/2007.00072.

Izmailov, Pavel, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. “Averaging Weights Leads to Wider Optima and Better Generalization.” **Uncertainty in Artificial Intelligence**, 876–85. https://arxiv.org/abs/1803.05407.

Jaakkola, Tommi, Michael I Jordan, and Satinder P Singh. 1994. “On the Convergence of Stochastic Iterative Dynamic Programming Algorithms.” **Neural Computation** 6 (6): 1185–201.

Jacobs, Robert A., Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. “Adaptive Mixtures of Local Experts.” **Neural Computation** 3 (1): 79–87.

Jacot, Arthur, Franck Gabriel, and Clément Hongler. 2018. “Neural Tangent Kernel: Convergence and Generalization in Neural Networks.” **Advances in Neural Information Processing Systems** 31. https://doi.org/10.1145/3406325.3465355.

Jaeger, Herbert. 2002. **Tutorial on Training Recurrent Neural Networks, Covering BPTT, RTRL, EKF and the “Echo State Network” Approach**. GMD-Forschungszentrum Informationstechnik Bonn.

Jaegle, Andrew, Sebastian Borgeaud, Jean-Baptiste Alayrac, et al. 2022. “Perceiver IO: A General Architecture for Structured Inputs & Outputs.” **International Conference on Learning Representations**.

Jaegle, Andrew, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. 2021. “Perceiver: General Perception with Iterative Attention.” **International Conference on Machine Learning**, 4651–64.

Jaggi, Martin. 2013. “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization.” **Proceedings of the 30th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 28: 427–35.

Jain, Paras, Ajay Jain, Aniruddha Nrusimha, et al. 2020. “Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization.” **Proceedings of Machine Learning and Systems** 2. https://arxiv.org/abs/1910.02653.

Jain, Sarthak, and Byron C. Wallace. 2019. “Attention Is Not Explanation.” **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics**, 3543–56.

Jamba Team. 2024. “Jamba-1.5: Hybrid Transformer-Mamba Models at Scale.” **arXiv**

Preprint arXiv:2408.12570.

James, W., and Charles Stein. 1961. "Estimation with Quadratic Loss." **Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability** 1: 361–79.

Jang, Eric, Shixiang Gu, and Ben Poole. 2017. "Categorical Reparameterization with Gumbel-Softmax." **International Conference on Learning Representations**.

Janner, Michael, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. "When to Trust Your Model: Model-Based Policy Optimization." **Advances in Neural Information Processing Systems** 32.

Jaynes, Edwin T. 1957. "Information Theory and Statistical Mechanics." **Physical Review** 106 (4): 620–30. https://doi.org/10.1103/PhysRev.106.620.

Jaynes, Edwin T. 1968. "Prior Probabilities." **IEEE Transactions on Systems Science and Cybernetics** 4 (3): 227–41. https://doi.org/10.1109/TSSC.1968.300117.

Jelassi, Samy, David Brandfonbrener, Sham M. Kakade, and Eran Malach. 2024. "Repeat After Me: Transformers Are Better Than State Space Models at Copying." **International Conference on Machine Learning**, 21502–21. https://arxiv.org/abs/2402.01032.

Jelinek, Frederick, Robert L. Mercer, Lalit R. Bahl, and James K. Baker. 1977. "Perplexity—a Measure of the Difficulty of Speech Recognition Tasks." **Journal of the Acoustical Society of America** 62 (S1): S63.

Jiang, Albert Q., Alexandre Sablayrolles, Arthur Mensch, et al. 2023. "Mistral 7B." **arXiv Preprint arXiv:2310.06825**.

Jiang, Albert Q., Alexandre Sablayrolles, Antoine Roux, et al. 2024. "Mixtral of Experts." **arXiv Preprint arXiv:2401.04088**.

Jiang, Nan, Alex Kulesza, Satinder Singh, and Richard Lewis. 2015. "The Dependence of Effective Planning Horizon on Model Accuracy." **International Conference on Autonomous Agents and Multiagent Systems**.

Jin, Chi, Zeyuan Allen-Zhu, Sébastien Bubeck, and Michael I. Jordan. 2018. "Is Q-Learning Provably Efficient?" **Advances in Neural Information Processing Systems** 31: 4863–73. https://arxiv.org/abs/1807.03765.

Jin, Ying, Zhuoran Yang, and Zhaoran Wang. 2021. "Is Pessimism Provably Efficient for Offline RL?" **International Conference on Machine Learning**, Proceedings of machine learning research, vol. 139: 5084–96.

Johari, Ramesh, Pete Koomen, Leonid Pekelis, and David Walsh. 2017. "Peeking at A/B Tests: Why It Matters, and What to Do about It." **Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, 1517–25. https://doi.org/10.1145/3097983.3097992.

Johnson, Rie, and Tong Zhang. 2013. "Accelerating Stochastic Gradient Descent Using Pre-

dictive Variance Reduction." **Advances in Neural Information Processing Systems** 26: 315–23.

Johnson, William B., and Joram Lindenstrauss. 1984. "Extensions of Lipschitz Mappings into a Hilbert Space." **Contemporary Mathematics** 26: 189–206.

Jolicoeur-Martineau, Alexia. 2019. "The Relativistic Discriminator: A Key Element Missing from Standard GAN." **International Conference on Learning Representations**.

Jolicoeur-Martineau, Alexia. 2020. "On Relativistic f-Divergences." **Proceedings of the 37th International Conference on Machine Learning**, 4931–39.

Jordan, Keller. 2026. **Modded-NanoGPT Optimization Benchmark**. Https://github.com/KellerJordan/modded-nanogpt/tree/master/records/track_3_optimization.

Jordan, Keller, and contributors. 2024. **Modded-Nanogpt: Speedrunning the NanoGPT Baseline**. Https://github.com/KellerJordan/modded-nanogpt.

Jordan, Keller, Yuchen Jin, Vlado Boza, et al. 2024. **Muon: An Optimizer for Hidden Layers in Neural Networks**. Https://kellerjordan.github.io/posts/muon/.

Jordan, Richard, David Kinderlehrer, and Felix Otto. 1998. "The Variational Formulation of the Fokker–Planck Equation." **SIAM Journal on Mathematical Analysis** 29 (1): 1–17.

Jouppi, Norman P, Cliff Young, Nishant Patil, et al. 2017. "In-Datacenter Performance Analysis of a Tensor Processing Unit." **2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)**, 1–12. https://doi.org/10.1145/3140659.3080246.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. 2015. "An Empirical Exploration of Recurrent Network Architectures." **International Conference on Machine Learning**, 2342–50. https://proceedings.mlr.press/v37/jozefowicz15.html.

Kadkhodaie, Zahra, Florentin Guth, Eero P. Simoncelli, and Stéphane Mallat. 2024. "Generalization in Diffusion Models Arises from Geometry-Adaptive Harmonic Representations." **International Conference on Learning Representations**. https://arxiv.org/abs/2310.02557.

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra. 1998. "Planning and Acting in Partially Observable Stochastic Domains." **Artificial Intelligence** 101 (1–2): 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X.

Kahan, William. 1965. "Further Remarks on Reducing Truncation Errors." **Communications of the ACM** 8 (1): 40.

Kahng, Minsuk, Nikhil Thorat, Duen Horng Chau, Fernanda B. Viégas, and Martin Wattenberg. 2018. "GAN Lab: Understanding Complex Deep Generative Models Using Interactive Visual Experimentation." **IEEE Transactions on Visualization and Computer Graphics** 25 (1): 310–20.

Kakade, Sham, and John Langford. 2002. “Approximately Optimal Approximate Reinforcement Learning.” **International Conference on Machine Learning**, 267–74.

Kalman, Rudolf E. 1960. “A New Approach to Linear Filtering and Prediction Problems.” **Journal of Basic Engineering** 82 (1): 35–45.

Kalra, Dayal Singh, and Maissam Barkeshli. 2024. “Why Warmup the Learning Rate? Underlying Mechanisms and Improvements.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2406.09405.

Kamb, Mason, and Surya Ganguli. 2025. “An Analytic Theory of Creativity in Convolutional Diffusion Models.” **Proceedings of the 42nd International Conference on Machine Learning**, Proceedings of machine learning research, vol. 267: 28795–831.

Kaplan, Jared, Sam McCandlish, Tom Henighan, et al. 2020. “Scaling Laws for Neural Language Models.” **ArXiv:2001.08361**. https://arxiv.org/abs/2001.08361.

Karimi, Hamed, Julie Nutini, and Mark Schmidt. 2016. “Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak–Łojasiewicz Condition.” **Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)**, 795–811.

Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. “Progressive Growing of GANs for Improved Quality, Stability, and Variation.” **International Conference on Learning Representations**. https://arxiv.org/abs/1710.10196.

Karras, Tero, Miika Aittala, Timo Aila, and Samuli Laine. 2022. “Elucidating the Design Space of Diffusion-Based Generative Models.” **Advances in Neural Information Processing Systems** 35.

Karras, Tero, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. 2020. “Training Generative Adversarial Networks with Limited Data.” **Advances in Neural Information Processing Systems** 33.

Karras, Tero, Miika Aittala, Tuomas Kynkäänniemi, Jaakko Lehtinen, Timo Aila, and Samuli Laine. 2024. “Guiding a Diffusion Model with a Bad Version of Itself.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2406.02507.

Karras, Tero, Miika Aittala, Samuli Laine, et al. 2021. “Alias-Free Generative Adversarial Networks.” **Advances in Neural Information Processing Systems** 34: 852–63. https://arxiv.org/abs/2106.12423.

Karras, Tero, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. 2024. “Analyzing and Improving the Training Dynamics of Diffusion Models.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**.

Karras, Tero, Samuli Laine, and Timo Aila. 2019. “A Style-Based Generator Architecture for Generative Adversarial Networks.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 4401–10. https://arxiv.org/abs/1812.04948.

Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. “Analyzing and Improving the Image Quality of StyleGAN.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 8110–19. https://arxiv.org/abs/1912.04958.

Karush, William. 1939. “Minima of Functions of Several Variables with Inequalities as Side Constraints.” Master’s thesis, Department of Mathematics, University of Chicago.

Kasimbeg, Priya, Frank Schneider, Runa Eschenhagen, et al. 2025. “Accelerating Neural Network Training: An Analysis of the AlgoPerf Competition.” **International Conference on Learning Representations**. https://arxiv.org/abs/2502.15015.

Katharopoulos, Angelos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. “Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention.” **International Conference on Machine Learning**.

Kawar, Bahjat, Michael Elad, Stefano Ermon, and Jiaming Song. 2022. “Denoising Diffusion Restoration Models.” **Advances in Neural Information Processing Systems** 35. https://arxiv.org/abs/2201.11793.

Kazemnejad, Amirhossein, Milad Aghajohari, Eva Portelance, et al. 2025. “VinePPO: Refining Credit Assignment in RL Training of LLMs.” **International Conference on Machine Learning**. https://arxiv.org/abs/2410.01679.

Kazemnejad, Amirhossein, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. 2023. “The Impact of Positional Encoding on Length Generalization in Transformers.” **Advances in Neural Information Processing Systems** 36. https://arxiv.org/abs/2305.19466.

Kidambi, Rahul, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. 2020. “MOReL: Model-Based Offline Reinforcement Learning.” **Advances in Neural Information Processing Systems** 33.

Kidger, Patrick. 2022. “On Neural Differential Equations.” **arXiv Preprint arXiv:2202.02435**.

Kidger, Patrick, James Morrill, James Foster, and Terry Lyons. 2020. “Neural Controlled Differential Equations for Irregular Time Series.” **Advances in Neural Information Processing Systems** 33: 6696–707.

Kim, Dongjun, Chieh-Hsin Lai, Wei-Hsiang Liao, et al. 2024. “Consistency Trajectory Models: Learning Probability Flow ODE Trajectory of Diffusion.” **International Conference on Learning Representations**. https://arxiv.org/abs/2310.02279.

Kim, Gyu Yeol, and Min-hwan Oh. 2026. “Convergence of Muon with Newton–Schulz.” **International Conference on Learning Representations**.

Kim, Jaeyeon, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. 2025. “Train for the Worst, Plan for the Best: Understanding Token Ordering in Masked Diffusions.” **International Conference on Machine Learning**.

Kimi Team. 2025a. “Kimi K2: Open Agentic Intelligence.” **arXiv Preprint arXiv:2507.20534**.

Kimi Team. 2025b. “Kimi Linear: An Expressive, Efficient Attention Architecture.” **arXiv Preprint arXiv:2510.26692**.

Kingma, Diederik P, and Jimmy Ba. 2015. “Adam: A Method for Stochastic Optimization.” **International Conference on Learning Representations**. https://arxiv.org/abs/1412.6980.

Kingma, Diederik P., and Ruiqi Gao. 2023. “Understanding Diffusion Objectives as the ELBO with Simple Data Augmentation.” **Advances in Neural Information Processing Systems** 36. https://arxiv.org/abs/2303.00848.

Kingma, Diederik P., Tim Salimans, Ben Poole, and Jonathan Ho. 2021. “Variational Diffusion Models.” **Advances in Neural Information Processing Systems** 34.

Kingma, Diederik P., and Max Welling. 2014. “Auto-Encoding Variational Bayes.” **International Conference on Learning Representations (ICLR)**. https://arxiv.org/abs/1312.6114.

Kitaev, Nikita, Lukasz Kaiser, and Anselm Levskaya. 2020. “Reformer: The Efficient Transformer.” **International Conference on Learning Representations**.

Kloeden, Peter E., and Eckhard Platen. 1992. **Numerical Solution of Stochastic Differential Equations**. Springer.

Kneser, Reinhard, and Hermann Ney. 1995. “Improved Backing-Off for M-Gram Language Modeling.” **International Conference on Acoustics, Speech, and Signal Processing** 1: 181–84. https://doi.org/10.1109/ICASSP.1995.479394.

Kocmi, Tom, Christian Federmann, Roman Grundkiewicz, Marcin Junczys-Dowmunt, Hitokazu Matsushita, and Arul Menezes. 2021. “To Ship or Not to Ship: An Extensive Evaluation of Automatic Metrics for Machine Translation.” **Proceedings of the Sixth Conference on Machine Translation**, 478–94.

Koehler, Frederic, Alexander Heckett, and Andrej Risteski. 2022. “Statistical Efficiency of Score Matching: The View from Isoperimetry.” **arXiv Preprint arXiv:2210.00726**. https://arxiv.org/abs/2210.00726.

Koh, Pang Wei, Shiori Sagawa, Henrik Marklund, et al. 2021. “WILDS: A Benchmark of in-the-Wild Distribution Shifts.” **International Conference on Machine Learning**, 5637–64. https://arxiv.org/abs/2012.07421.

Kohavi, Ron. 1995. “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.” **International Joint Conference on Artificial Intelligence** 14: 1137–45.

Kolda, Tamara G., and Brett W. Bader. 2009. “Tensor Decompositions and Applications.” **SIAM Review** 51 (3): 455–500.

Koller, Daphne, and Nir Friedman. 2009. **Probabilistic Graphical Models: Principles and Techniques**. MIT Press. https://doi.org/10.7551/mitpress/7432.001.0001.

Kolmogorov, Andrey. 1933. **Grundbegriffe Der Wahrscheinlichkeitsrechnung**. Springer. https://doi.org/10.1007/978-3-642-49888-6.

Kolmogorov, Andrey N. 1931. "Über Die Analytischen Methoden in Der Wahrscheinlichkeitsrechnung." **Mathematische Annalen** 104: 415–58.

Komatsuzaki, Aran, Joan Puigcerver, James Lee-Thorp, et al. 2023. "Sparse Upcycling: Training Mixture-of-Experts from Dense Checkpoints." **International Conference on Learning Representations**.

Konda, Vijay R, and John N Tsitsiklis. 2000. "Actor-Critic Algorithms." **Advances in Neural Information Processing Systems** 12.

Kong, Jungil, Jaehyeon Kim, and Jaekyoung Bae. 2020. "HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis." **Advances in Neural Information Processing Systems** 33.

Kong, Zhifeng, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. 2021. "DiffWave: A Versatile Diffusion Model for Audio Synthesis." **International Conference on Learning Representations**. https://arxiv.org/abs/2009.09761.

Kool, Wouter, Herke van Hoof, and Max Welling. 2019. "Buy 4 REINFORCE Samples, Get a Baseline for Free!" **ICLR 2019 Workshop on Deep Reinforcement Learning Meets Structured Prediction**. https://mlanthology.org/iclrw/2019/kool2019iclrw-buy/.

Koopman, Bernard O. 1936. "On Distributions Admitting a Sufficient Statistic." **Transactions of the American Mathematical Society** 39 (3): 399–409.

Korbak, Tomasz, Ethan Perez, and Christopher L Buckley. 2022. "RL with KL Penalties Is Better Viewed as Bayesian Inference." **Findings of the Association for Computational Linguistics: EMNLP 2022**. https://arxiv.org/abs/2205.11275.

Korpelevich, G. M. 1976. "The Extragradient Method for Finding Saddle Points and Other Problems." **Ekonomika i Matematicheskie Metody** 12: 747–56.

Korthikanti, Vijay, Jared Casper, Sangkug Lym, et al. 2022. "Reducing Activation Recomputation in Large Transformer Models." **arXiv Preprint arXiv:2205.05198**.

Kosson, Atli, Bettina Messmer, and Martin Jaggi. 2024. "Rotational Equilibrium: How Weight Decay Balances Learning Across Neural Networks." **International Conference on Machine Learning**. https://arxiv.org/abs/2305.17212.

Kostrikov, Ilya, Ashvin Nair, and Sergey Levine. 2022. "Offline Reinforcement Learning with Implicit Q-Learning." **International Conference on Learning Representations**. https://arxiv.org/abs/2110.06169.

Kovalev, Dmitry. 2025. "Understanding Gradient Orthogonalization for Deep Learning via

Non-Euclidean Trust-Region Optimization." **arXiv Preprint arXiv:2503.12645**.

Krakovna, Victoria, Jonathan Uesato, Vladimir Mikulik, et al. 2020. **Specification Gaming: The Flip Side of AI Ingenuity**. DeepMind blog, 21 April 2020, https://deepmind.google/blog/specification-gaming-the-flip-side-of-ai-ingenuity/.

Kramár, János, Tom Lieberum, Rohin Shah, and Neel Nanda. 2024. "AtP*: An Efficient and Scalable Method for Localizing LLM Behaviour to Components." **arXiv Preprint arXiv:2403.00745**.

Kramers, Hendrik A. 1940. "Brownian Motion in a Field of Force and the Diffusion Model of Chemical Reactions." **Physica** 7 (4): 284–304.

Kraskov, Alexander, Harald Stögbauer, and Peter Grassberger. 2004. "Estimating Mutual Information." **Physical Review E** 69 (6): 066138.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." **Advances in Neural Information Processing Systems**, 1097–105. https://doi.org/10.5555/2999134.2999257.

Kuhn, Harold W., and Albert W. Tucker. 1951. "Nonlinear Programming." **Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability**, 481–92.

Kuleshov, Volodymyr. 2023. **Deep Generative Models: Lecture Slides (Lectures 10–13: Advanced Topics in GANs, Energy-Based Models, Score-Based Generative Models, Diffusion Models)**. Course materials, Cornell Tech. Course website: https://kuleshov-group.github.io/dgm-website/; slides: https://github.com/kuleshov-group/dgm-resources.

Kullback, Solomon. 1959. **Information Theory and Statistics**. John Wiley & Sons.

Kullback, Solomon, and Richard A. Leibler. 1951. "On Information and Sufficiency." **Annals of Mathematical Statistics** 22 (1): 79–86.

Kumar, Aviral, Aurick Zhou, George Tucker, and Sergey Levine. 2020. "Conservative Q-Learning for Offline Reinforcement Learning." **Advances in Neural Information Processing Systems** 33.

Kung, Sun Yuan. 1988. "VLSI Array Processors." **Prentice Hall**.

Kunstner, Frederik, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. 2023. "Noise Is Not the Main Factor Behind the Gap Between SGD and Adam on Transformers, but Sign Descent Might Be." **International Conference on Learning Representations**. https://arxiv.org/abs/2304.13960.

Kunstner, Frederik, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. 2024. "Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models." **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2402.19449.

Kuzovkin, Ilya, Raul Vicente, Mathilde Petton, et al. 2018. "Activations of Deep Convolu-

tional Neural Networks Are Aligned with Gamma Band Activity of Human Visual Cortex." **Communications Biology** 1 (1): 1–12. https://doi.org/10.1038/s42003-018-0110-y.

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, et al. 2023. "Efficient Memory Management for Large Language Model Serving with PagedAttention." **Proceedings of the 29th Symposium on Operating Systems Principles**, 611–26. https://doi.org/10.1145/3600006.3613165.

Kynkäänniemi, Tuomas, Miika Aittala, Tero Karras, Samuli Laine, Timo Aila, and Jaakko Lehtinen. 2024. "Applying Guidance in a Limited Interval Improves Sample and Distribution Quality in Diffusion Models." **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2404.07724.

Kynkäänniemi, Tuomas, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. 2019. "Improved Precision and Recall Metric for Assessing Generative Models." **Advances in Neural Information Processing Systems** 32. https://arxiv.org/abs/1904.06991.

Lahoti, Aakash, Kevin Y. Li, Berlin Chen, et al. 2026. "Mamba-3: Improved Sequence Modeling Using State Space Principles." **arXiv Preprint arXiv:2603.15569**.

Lai, Chieh-Hsin, Yang Song, Dongjun Kim, Yuki Mitsufuji, and Stefano Ermon. 2025. "The Principles of Diffusion Models." **arXiv Preprint arXiv:2510.21890**. https://arxiv.org/abs/2510.21890.

Lai, Tze Leung, and Herbert Robbins. 1985. "Asymptotically Efficient Adaptive Allocation Rules." **Advances in Applied Mathematics** 6 (1): 4–22. https://doi.org/10.1016/0196-8858(85)90002-8.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. 2017. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles." **Advances in Neural Information Processing Systems** 30.

Lambert, Nathan. 2026. **Reinforcement Learning from Human Feedback**. Online book, continuously updated since 2024; arXiv:2504.12501; print edition Manning/Simon & Schuster, https://rlhfbook.com/.

Laplace, Pierre-Simon. 1814. **Essai Philosophique Sur Les Probabilités**. Courcier.

Lattimore, Tor, and Csaba Szepesvári. 2020. **Bandit Algorithms**. Cambridge University Press. https://tor-lattimore.com/downloads/book/book.pdf.

Läuchli, Peter. 1961. "Jordan-Elimination Und Ausgleichung Nach Kleinsten Quadraten." **Numerische Mathematik** 3 (1): 226–40.

LeCun, Yann, Bernhard Boser, John S Denker, et al. 1989. "Backpropagation Applied to Handwritten Zip Code Recognition." **Neural Computation** 1 (4): 541–51. https://doi.org/10.1162/neco.1989.1.4.541.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." **Proceedings of the IEEE** 86 (11): 2278–

324. https://doi.org/10.1109/5.726791.

LeCun, Yann, Leon Bottou, G Orr, and Klaus-Robert Muller. 1998. “Efficient Backprop.” In **Neural Networks: Tricks of the Trade**. Springer. https://doi.org/10.1007/3-540-49430-8_2.

LeCun, Yann, Sumit Chopra, Raia Hadsell, Marc’Aurelio Ranzato, and Fu Jie Huang. 2006. “A Tutorial on Energy-Based Learning.” In **Predicting Structured Data**. MIT Press.

LeCun, Yann, LD Jackel, Leon Bottou, et al. 1995. “Comparison of Learning Algorithms for Handwritten Digit Recognition.” **International Conference on Artificial Neural Networks**, 53–60.

Lee, Jason D., Max Simchowitz, Michael I. Jordan, and Benjamin Recht. 2016. “Gradient Descent Only Converges to Minimizers.” **29th Annual Conference on Learning Theory**, Proceedings of machine learning research, vol. 49: 1246–57.

Lee, Sang-gil, Wei Ping, Boris Ginsburg, Bryan Catanzaro, and Sungroh Yoon. 2023. “BigVGAN: A Universal Neural Vocoder with Large-Scale Training.” **International Conference on Learning Representations**.

Lee, Yongjoon, and Jung-Woo Choi. 2026. “RAF: Relativistic Adversarial Feedback for Universal Speech Synthesis.” **Interspeech**.

Legendre, Adrien Marie. 1805. **Nouvelles méthodes Pour La détermination Des Orbites Des Comètes**. Courcier.

Lepikhin, Dmitry, HyoukJoong Lee, Yuanzhong Xu, et al. 2021. “GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding.” **International Conference on Learning Representations**.

Leshno, Moshe, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. “Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function.” **Neural Networks** 6 (6): 861–67.

Lessard, Laurent, Benjamin Recht, and Andrew Packard. 2016. “Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints.” **SIAM Journal on Optimization** 26 (1): 57–95.

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. 2023. “Fast Inference from Transformers via Speculative Decoding.” **International Conference on Machine Learning**, 19274–86.

Levin, David A., and Yuval Peres. 2017. **Markov Chains and Mixing Times**. 2nd ed. American Mathematical Society.

Levine, Sergey. 2018. “Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review.” **arXiv Preprint arXiv:1805.00909**.

Levine, Sergey, Aviral Kumar, George Tucker, and Justin Fu. 2020. “Offline Reinforce-

ment Learning: Tutorial, Review, and Perspectives on Open Problems." **arXiv Preprint arXiv:2005.01643**.

Levy, Omer, and Yoav Goldberg. 2014. "Neural Word Embedding as Implicit Matrix Factorization." **Advances in Neural Information Processing Systems** 27.

Li, Chun-Liang, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. 2017. "MMD GAN: Towards Deeper Understanding of Moment Matching Network." **Advances in Neural Information Processing Systems** 30.

Li, Gen, Yuting Wei, Yuejie Chi, Yuantao Gu, and Yuxin Chen. 2021. "Softmax Policy Gradient Methods Can Take Exponential Time to Converge." **Proceedings of the 34th Conference on Learning Theory**, Proceedings of machine learning research, vol. 134: 3107–10.

Li, Hao, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. "Visualizing the Loss Landscape of Neural Nets." **Advances in Neural Information Processing Systems** 31: 6389–99.

Li, Ke, and Jitendra Malik. 2018. "Implicit Maximum Likelihood Estimation." **arXiv Preprint arXiv:1809.09087**.

Li, Ming, and Paul Vitányi. 2019. **An Introduction to Kolmogorov Complexity and Its Applications**. 4th ed. Texts in Computer Science. Springer. https://doi.org/10.1007/978-3-030-11298-1.

Li, Mu, David G Andersen, Jun Woo Park, et al. 2014. "Scaling Distributed Machine Learning with the Parameter Server." **11th Symposium on Operating Systems Design and Implementation (OSDI 14)**, 583–98. https://doi.org/10.1145/2640087.2644155.

Li, Qianxiao, Cheng Tai, and Weinan E. 2017. "Stochastic Modified Equations and Adaptive Stochastic Gradient Algorithms." **Proceedings of the 34th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 70: 2101–10.

Li, Shen, Yanli Zhao, Rohan Varma, et al. 2020. "PyTorch Distributed: Experiences on Accelerating Data Parallel Training." **Proceedings of the VLDB Endowment** 13 (12): 3005–18. https://doi.org/10.14778/3415478.3415530.

Li, Tianhong, and Kaiming He. 2026. "Back to Basics: Let Denoising Generative Models Denoise." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 36115–25. https://arxiv.org/abs/2511.13720.

Li, Weiwei, and Emanuel Todorov. 2004. "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems." **Proceedings of the First International Conference on Informatics in Control, Automation and Robotics**, 222–29.

Li, Xiang, Shuo Chen, Xiaolin Hu, and Jian Yang. 2019. "Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 2682–90. https://arxiv.org/abs/1801.05134.

Li, Xuechen, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. 2020. “Scalable Gradients for Stochastic Differential Equations.” **Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics**, Proceedings of machine learning research, vol. 108: 3870–82.

Li, Yujia, Kevin Swersky, and Richard Zemel. 2015. “Generative Moment Matching Networks.” **Proceedings of the 32nd International Conference on Machine Learning**, 1718–27.

Li, Zichong, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. 2026. “NorMuon: Making Muon More Efficient and Scalable.” **International Conference on Machine Learning**.

Lieber, Opher, Barak Lenz, Hofit Bata, et al. 2024. “Jamba: A Hybrid Transformer-Mamba Language Model.” **arXiv Preprint arXiv:2403.19887**.

Lightman, Hunter, Vineet Kosaraju, Yuri Burda, et al. 2024. “Let’s Verify Step by Step.” **International Conference on Learning Representations**.

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, et al. 2016. “Continuous Control with Deep Reinforcement Learning.” **International Conference on Learning Representations**. https://arxiv.org/abs/1509.02971.

Lim, Jae Hyun, and Jong Chul Ye. 2017. “Geometric GAN.” **arXiv Preprint arXiv:1705.02894**.

Lin, Jianhua. 1991. “Divergence Measures Based on the Shannon Entropy.” **IEEE Transactions on Information Theory** 37 (1): 145–51.

Lin, Long-Ji. 1992. “Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching.” **Machine Learning** 8: 293–321.

Lin, Min, Qiang Chen, and Shuicheng Yan. 2013. “Network in Network.” **ArXiv:1312.4400**. https://arxiv.org/abs/1312.4400.

Lin, Shanchuan, Bingchen Liu, Jiashi Li, and Xiao Yang. 2024. “Common Diffusion Noise Schedules and Sample Steps Are Flawed.” **Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision**, 5404–11. https://arxiv.org/abs/2305.08891.

Lin, Shanchuan, Anran Wang, and Xiao Yang. 2024. “SDXL-Lightning: Progressive Adversarial Diffusion Distillation.” **arXiv Preprint arXiv:2402.13929**.

Lin, Shanchuan, Xin Xia, Yuxi Ren, Ceyuan Yang, Xuefeng Xiao, and Lu Jiang. 2025. “Diffusion Adversarial Post-Training for One-Step Video Generation.” **Proceedings of the 42nd International Conference on Machine Learning**.

Lin, Zhouhan, Minwei Feng, Cicero Nogueira dos Santos, et al. 2017. “A Structured Self-Attentive Sentence Embedding.” **ArXiv:1703.03130**. https://arxiv.org/abs/1703.03130.

Linnainmaa, Seppo. 1970. “The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors.” Master’s thesis, University

of Helsinki.

Lipman, Yaron, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. 2023. "Flow Matching for Generative Modeling." **International Conference on Learning Representations**.

Lipman, Yaron, Marton Havasi, Peter Holderrieth, et al. 2024. "Flow Matching Guide and Code." **arXiv Preprint arXiv:2412.06264**. https://arxiv.org/abs/2412.06264.

Lipton, Zachary, Yu-Xiang Wang, and Alexander Smola. 2018. "Detecting and Correcting for Label Shift with Black Box Predictors." **International Conference on Machine Learning**, 3122–30. https://arxiv.org/abs/1802.03916.

Little, John D. C. 1961. "A Proof for the Queuing Formula: $L = \lambda W$." **Operations Research** 9 (3): 383–87. https://doi.org/10.1287/opre.9.3.383.

Liu, Bo, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. 2024. "Longhorn: State Space Models Are Amortized Online Learners." **arXiv Preprint arXiv:2407.14207**.

Liu, Chaoyue, Libin Zhu, and Mikhail Belkin. 2022. "Loss Landscapes and Optimization in over-Parameterized Non-Linear Systems and Neural Networks." **Applied and Computational Harmonic Analysis** 59: 85–116.

Liu, Dong C, and Jorge Nocedal. 1989. "On the Limited Memory BFGS Method for Large Scale Optimization." **Mathematical Programming** 45 (1): 503–28. https://doi.org/10.1007/bf01589116.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang. 2018. "DARTS: Differentiable Architecture Search." **ArXiv:1806.09055**. https://arxiv.org/abs/1806.09055.

Liu, Jie, Gongye Liu, Jiajun Liang, et al. 2025. "Flow-GRPO: Training Flow Matching Models via Online RL." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2505.05470.

Liu, Jingyuan, Jianlin Su, Xingcheng Yao, et al. 2025. "Muon Is Scalable for LLM Training." **arXiv Preprint arXiv:2502.16982**.

Liu, Nelson F., Kevin Lin, John Hewitt, et al. 2024. "Lost in the Middle: How Language Models Use Long Contexts." **Transactions of the Association for Computational Linguistics** 12: 157–73. https://arxiv.org/abs/2307.03172.

Liu, Qiang. 2022. "Rectified Flow: A Marginal Preserving Approach to Optimal Transport." **arXiv Preprint arXiv:2209.14577**.

Liu, Qiang, Jason D. Lee, and Michael I. Jordan. 2016. "A Kernelized Stein Discrepancy for Goodness-of-Fit Tests." **Proceedings of the 33rd International Conference on Machine Learning**, 276–84.

Liu, Qiang, and Dilin Wang. 2016. "Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm." **Advances in Neural Information Processing Systems**

29.

Liu, Shih-Yang, Chien-Yi Wang, Hongxu Yin, et al. 2024. “DoRA: Weight-Decomposed Low-Rank Adaptation.” **International Conference on Machine Learning**.

Liu, Xingchao, Chengyue Gong, and Qiang Liu. 2023. “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow.” **International Conference on Learning Representations**.

Liu, Yinhan, Myle Ott, Naman Goyal, et al. 2019. “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” **ArXiv:1907.11692**. https://arxiv.org/abs/1907.11692.

Liu, Yue, Yunjie Tian, Yuzhong Zhao, et al. 2024. “VMamba: Visual State Space Model.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2401.10166.

Liu, Ze, Yutong Lin, Yue Cao, et al. 2021. “Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows.” **Proceedings of the IEEE/CVF International Conference on Computer Vision**, 10012–22. https://doi.org/10.1109/iccv48922.2021.00986.

Liu, Zhuang, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. “A ConvNet for the 2020s.” **ArXiv:2201.03545**. https://arxiv.org/abs/2201.03545.

Liu, Zichen, Changyu Chen, Wenjun Li, et al. 2025. “Understanding R1-Zero-Like Training: A Critical Perspective.” **Second Conference on Language Modeling**. https://arxiv.org/abs/2503.20783.

Łojasiewicz, Stanisław. 1963. “Une Propriété Topologique Des Sous-Ensembles Analytiques réels.” In **Les Équations Aux dérivées Partielles**. Éditions du CNRS.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell. 2015. “Fully Convolutional Networks for Semantic Segmentation.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 3431–40. https://doi.org/10.1109/CVPR.2015.7298965.

Lopez-Paz, David, and Maxime Oquab. 2017. “Revisiting Classifier Two-Sample Tests.” **International Conference on Learning Representations**.

Loshchilov, Ilya, and Frank Hutter. 2016. “SGDR: Stochastic Gradient Descent with Warm Restarts.” **ArXiv:1608.03983**. https://arxiv.org/abs/1608.03983.

Loshchilov, Ilya, and Frank Hutter. 2019. “Decoupled Weight Decay Regularization.” **International Conference on Learning Representations**. https://arxiv.org/abs/1711.05101.

Lou, Aaron, Chenlin Meng, and Stefano Ermon. 2024. “Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution.” **Proceedings of the 41st International Conference on Machine Learning**, Proceedings of machine learning research, vol. 235: 32819–48. https://arxiv.org/abs/2310.16834.

Lowe, David G. 2004. “Distinctive Image Features from Scale-Invariant Keypoints.” **International Journal of Computer Vision** 60 (2): 91–110. https://doi.org/10.1023/b:visi.0000029664.99615.94.

Lu, Cheng, and Yang Song. 2025. “Simplifying, Stabilizing and Scaling Continuous-Time Consistency Models.” **International Conference on Learning Representations**.

Lu, Cheng, Kaiwen Zheng, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022. “Maximum Likelihood Training for Score-Based Diffusion ODEs by High-Order Denoising Score Matching.” **International Conference on Machine Learning**.

Lu, Cheng, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022. “DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps.” **Advances in Neural Information Processing Systems** 35.

Lu, Cheng, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2025. “DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models.” **Machine Intelligence Research** 22 (4): 730–51. https://doi.org/10.1007/s11633-025-1562-4.

Lu, Enzhe, Zhejun Jiang, Jingyuan Liu, et al. 2025. “MoBA: Mixture of Block Attention for Long-Context LLMs.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2502.13189.

Lucic, Mario, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. 2018. “Are GANs Created Equal? A Large-Scale Study.” **Advances in Neural Information Processing Systems** 31. https://arxiv.org/abs/1711.10337.

Lugmayr, Andreas, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. 2022. “RePaint: Inpainting Using Denoising Diffusion Probabilistic Models.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 11461–71. https://arxiv.org/abs/2201.09865.

Luo, Calvin. 2022. “Understanding Diffusion Models: A Unified Perspective.” **arXiv Preprint arXiv:2208.11970**.

Luo, Ping, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. 2018. “Towards Understanding Regularization in Batch Normalization.” **ArXiv:1809.00846**. https://arxiv.org/abs/1809.00846.

Luo, Weile, Ruibo Fan, Zeyu Li, Dayou Du, Qiang Wang, and Xiaowen Chu. 2024. “Benchmarking and Dissecting the Nvidia Hopper GPU Architecture.” **arXiv Preprint arXiv:2402.13499**.

Luo, Wenjie, Yujia Li, Raquel Urtasun, and Richard Zemel. 2016. “Understanding the Effective Receptive Field in Deep Convolutional Neural Networks.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/1701.04128.

Lyle, Clare, Pablo Samuel Castro, and Marc G. Bellemare. 2019. “A Comparative Analysis of Expected and Distributional Reinforcement Learning.” **Proceedings of the AAAI Conference on Artificial Intelligence** 33: 4504–11.

Ma, Siyuan, Raef Bassily, and Mikhail Belkin. 2018. “The Power of Interpolation: Understanding the Effectiveness of SGD in Modern over-Parametrized Learning.” **Proceedings of the 35th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 80: 3325–34.

Mack, Yue-Pok, and Bernard W Silverman. 1982. “Weak and Strong Uniform Consistency of Kernel Regression Estimates.” **Zeitschrift für Wahrscheinlichkeitstheorie Und Verwandte Gebiete** 61 (3): 405–15. https://doi.org/10.1007/bf00539840.

MacKay, David J. C. 1992. “The Evidence Framework Applied to Classification Networks.” **Neural Computation** 4 (5): 720–36. https://doi.org/10.1162/neco.1992.4.5.720.

Macke, Stephen, Hongpu Gong, Doris Jung-Lin Lee, Andrew Head, Doris Xin, and Aditya Parameswaran. 2021. “Fine-Grained Lineage for Safer Notebook Interactions.” **Proceedings of the VLDB Endowment** 14 (6): 1093–101. https://doi.org/10.14778/3447689.3447712.

Maddison, Chris J., Andriy Mnih, and Yee Whye Teh. 2017. “The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables.” **International Conference on Learning Representations**.

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. “Towards Deep Learning Models Resistant to Adversarial Attacks.” **International Conference on Learning Representations**. https://arxiv.org/abs/1706.06083.

Mahoney, Michael W., and Petros Drineas. 2009. “CUR Matrix Decompositions for Improved Data Analysis.” **Proceedings of the National Academy of Sciences** 106 (3): 697–702.

Makoviychuk, Viktor, Lukasz Wawrzyniak, Yunrong Guo, et al. 2021. “Isaac Gym: High Performance GPU-Based Physics Simulation for Robot Learning.” **Advances in Neural Information Processing Systems, Datasets and Benchmarks Track**. https://arxiv.org/abs/2108.10470.

Malladi, Sadhika, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. 2022. “On the SDEs and Scaling Rules for Adaptive Gradient Algorithms.” **Advances in Neural Information Processing Systems** 35. https://arxiv.org/abs/2205.10287.

Mandt, Stephan, Matthew D. Hoffman, and David M. Blei. 2017. “Stochastic Gradient Descent as Approximate Bayesian Inference.” **Journal of Machine Learning Research** 18 (134): 1–35.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. **Introduction to Information Retrieval**. Cambridge University Press. https://doi.org/10.1017/CBO9780511809071.

Mao, Xudong, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. 2017. “Least Squares Generative Adversarial Networks.” **Proceedings of the IEEE International Conference on Computer Vision**, 2794–802.

Marchenko, Vladimir A., and Leonid A. Pastur. 1967. “Distribution of Eigenvalues for Some Sets of Random Matrices.” **Mathematics of the USSR-Sbornik** 1 (4): 457–83.

Markowitz, Harry. 1952. “Portfolio Selection.” **The Journal of Finance** 7 (1): 77–91.

Maron, Melvin E. 1961. “Automatic Indexing: An Experimental Inquiry.” **Journal of the ACM** 8 (3): 404–17.

Martens, James. 2010. “Deep Learning via Hessian-Free Optimization.” **Proceedings of the 27th International Conference on Machine Learning**, 735–42.

Martens, James. 2020. “New Insights and Perspectives on the Natural Gradient Method.” **Journal of Machine Learning Research** 21 (146).

Martens, James, and Roger Grosse. 2015. “Optimizing Neural Networks with Kronecker-Factored Approximate Curvature.” **Proceedings of the 32nd International Conference on Machine Learning (ICML)**, 2408–17.

Martin, Charles H., and Michael W. Mahoney. 2021. “Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning.” **Journal of Machine Learning Research** 22 (165): 1–73.

Martins, André F. T., and Ramón F. Astudillo. 2016. “From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification.” **Proceedings of the 33rd International Conference on Machine Learning (ICML)**, 1614–23.

Maruyama, Gisiro. 1955. “Continuous Markov Processes and Stochastic Equations.” **Rendiconti Del Circolo Matematico Di Palermo** 4: 48–90.

Massias, Mathurin, and Quentin Bertrand. 2026. **Diffusion and Flow-Matching: From Memorization to Generalization & Beyond**. Tutorial at the International Conference on Machine Learning, https://icml.cc/virtual/2026/75374.

Mayne, David Q. 1966. “A Second-Order Gradient Method for Determining Optimal Trajectories of Non-Linear Discrete-Time Systems.” **International Journal of Control** 3 (1): 85–95. https://doi.org/10.1080/00207176608921369.

Mazumder, Rahul, Trevor Hastie, and Robert Tibshirani. 2010. “Spectral Regularization Algorithms for Learning Large Incomplete Matrices.” **Journal of Machine Learning Research** 11 (80): 2287–322.

McAllester, David A. 1999. “Some PAC-Bayesian Theorems.” **Machine Learning** 37 (3): 355–63. https://doi.org/10.1023/A:1007618624809.

McAllester, David, and Karl Stratos. 2020. “Formal Limitations on the Measurement of Mutual Information.” **International Conference on Artificial Intelligence and Statistics**, 875–84.

McCandlish, Sam, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. “An Empirical Model of Large-Batch Training.” **arXiv Preprint arXiv:1812.06162**.

McCann, Robert J. 1997. “A Convexity Principle for Interacting Gases.” **Advances in Mathematics** 128 (1): 153–79.

McCulloch, Warren S, and Walter Pitts. 1943. “A Logical Calculus of the Ideas Immanent in Nervous Activity.” **Bulletin of Mathematical Biophysics** 5 (4): 115–33. https://doi.org/10.1016/s0092-8240(05)80006-0.

McDiarmid, Colin. 1989. “On the Method of Bounded Differences.” In **Surveys in Combinatorics**. Cambridge University Press.

McFadden, Daniel. 1974. “Conditional Logit Analysis of Qualitative Choice Behavior.” In **Frontiers in Econometrics**, edited by Paul Zarembka. Academic Press.

McKinney, Wes. 2010. “Data Structures for Statistical Computing in Python.” **Proceedings of the 9th Python in Science Conference**, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a.

McMillan, Brockway. 1956. “Two Inequalities Implied by Unique Decipherability.” **IRE Transactions on Information Theory** 2 (4): 115–16.

Mei, Jincheng, Bo Dai, Chenjun Xiao, Csaba Szepesvári, and Dale Schuurmans. 2021. “Understanding the Effect of Stochasticity in Policy Optimization.” **Advances in Neural Information Processing Systems** 34.

Mei, Jincheng, Chenjun Xiao, Csaba Szepesvári, and Dale Schuurmans. 2020. “On the Global Convergence Rates of Softmax Policy Gradient Methods.” **Proceedings of the 37th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 119: 6820–29.

Meng, Chenlin, Yutong He, Yang Song, et al. 2022. “SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations.” **International Conference on Learning Representations**. https://arxiv.org/abs/2108.01073.

Meng, Chenlin, Robin Rombach, Ruiqi Gao, et al. 2023. “On Distillation of Guided Diffusion Models.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 14297–306.

Meng, Fanxu, Zhaohui Wang, and Muhan Zhang. 2024. “PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models.” **Advances in Neural Information Processing Systems**.

Meng, Kevin, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. “Locating and Editing Factual Associations in GPT.” **Advances in Neural Information Processing Systems** 35. https://arxiv.org/abs/2202.05262.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher. 2016. “Pointer Sentinel Mixture Models.” **ArXiv:1609.07843**. https://arxiv.org/abs/1609.07843.

Merrill, William, Shane Arora, Dirk Groeneveld, and Hannaneh Hajishirzi. 2025. “Critical Batch Size Revisited: A Simple Empirical Approach to Large-Batch Language Model

Training." **arXiv Preprint arXiv:2505.23971**.

Merrill, William, Jackson Petty, and Ashish Sabharwal. 2024. "The Illusion of State in State-Space Models." **International Conference on Machine Learning**. https://arxiv.org/abs/2404.08819.

Mescheder, Lars, Andreas Geiger, and Sebastian Nowozin. 2018. "Which Training Methods for GANs Do Actually Converge?" **International Conference on Machine Learning**, 3481–90. https://arxiv.org/abs/1801.04406.

Meta AI. 2025. **The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation**. Https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

Metropolis, Nicholas, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. "Equation of State Calculations by Fast Computing Machines." **The Journal of Chemical Physics** 21 (6): 1087–92.

Metropolis, Nicholas, and Stanislaw Ulam. 1949. "The Monte Carlo Method." **Journal of the American Statistical Association** 44 (247): 335–41.

Michel, Paul, Omer Levy, and Graham Neubig. 2019. "Are Sixteen Heads Really Better Than One?" **Advances in Neural Information Processing Systems** 32.

Micikevicius, Paulius, Sharan Narang, Jonah Alben, et al. 2018. "Mixed Precision Training." **International Conference on Learning Representations**.

Micikevicius, Paulius, Dusan Stosic, Neil Burgess, et al. 2022. "FP8 Formats for Deep Learning." **arXiv Preprint arXiv:2209.05433**.

Mikolov, Tomáš, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2011. "Extensions of Recurrent Neural Network Language Model." **IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**, 5528–31. https://doi.org/10.1109/ICASSP.2011.5947611.

Milakov, Maxim, and Natalia Gimelshein. 2018. "Online Normalizer Calculation for Softmax." **arXiv Preprint arXiv:1805.02867**.

Miller, Evan. 2024. "Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations." **arXiv Preprint arXiv:2411.00640**.

Milstein, Grigori N. 1975. "Approximate Integration of Stochastic Differential Equations." **Theory of Probability and Its Applications** 19 (3): 557–62.

MiniMax. 2025. "MiniMax-01: Scaling Foundation Models with Lightning Attention." **arXiv Preprint arXiv:2501.08313**.

Mironov, Ilya. 2017. "Rényi Differential Privacy." **2017 IEEE 30th Computer Security Foundations Symposium (CSF)**, 263–75.

Mirsky, Leon. 1960. "Symmetric Gauge Functions and Unitarily Invariant Norms." **The**

Quarterly Journal of Mathematics 11 (1): 50–59.

Mirza, Mehdi, and Simon Osindero. 2014. "Conditional Generative Adversarial Nets." **ArXiv:1411.1784**. https://arxiv.org/abs/1411.1784.

Mishchenko, Konstantin, Ahmed Khaled, and Peter Richtárik. 2020. "Random Reshuffling: Simple Analysis with Vast Improvements." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2006.05988.

Mishkin, Dmytro, and Jiri Matas. 2016. "All You Need Is a Good Init." **International Conference on Learning Representations**. https://arxiv.org/abs/1511.06422.

Mistral AI. 2023. **Mistral-7B-Instruct-v0.2 Model Card**. Https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2.

Miyasawa, Koichi. 1961. "An Empirical Bayes Estimator of the Mean of a Normal Population." **Bulletin of the International Statistical Institute** 38 (4): 181–88.

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. "Spectral Normalization for Generative Adversarial Networks." **International Conference on Learning Representations**.

Miyato, Takeru, and Masanori Koyama. 2018. "CGANs with Projection Discriminator." **International Conference on Learning Representations**. https://arxiv.org/abs/1802.05637.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, et al. 2016. "Asynchronous Methods for Deep Reinforcement Learning." **International Conference on Machine Learning**, 1928–37.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, et al. 2013. "Playing Atari with Deep Reinforcement Learning." **ArXiv:1312.5602**.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, et al. 2015. "Human-Level Control Through Deep Reinforcement Learning." **Nature** 518 (7540): 529–33. https://doi.org/10.1038/nature14236.

Mohamed, Shakir, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. 2020. "Monte Carlo Gradient Estimation in Machine Learning." **Journal of Machine Learning Research** 21 (132): 1–62.

Mohan, Jayashree, Amar Phanishayee, Ashish Raniwala, and Vijay Chidambaram. 2021. "Analyzing and Mitigating Data Stalls in DNN Training." **Proceedings of the VLDB Endowment** 14 (5): 771–84. https://doi.org/10.14778/3446095.3446100.

Mokady, Ron, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. 2023. "NULL-Text Inversion for Editing Real Images Using Guided Diffusion Models." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 6038–47.

Moler, Cleve B. 1967. "Iterative Refinement in Floating Point." **Journal of the ACM** 14 (2):

316–21.

Moon, Taesup, Alex Smola, Yi Chang, and Zhaohui Zheng. 2010. “Intervalrank: Isotonic Regression with Listwise and Pairwise Constraints.” **Proceedings of the 3rd ACM International Conference on Web Search and Data Mining**, 151–60. https://doi.org/10.1145/1718487.1718520.

Müller, Alfred. 1997. “Integral Probability Metrics and Their Generating Classes of Functions.” **Advances in Applied Probability** 29 (2): 429–43.

Müller, Rafael, Simon Kornblith, and Geoffrey Hinton. 2019. “When Does Label Smoothing Help?” **Advances in Neural Information Processing Systems** 32. https://arxiv.org/abs/1906.02629.

Murphy, Kevin P. 2025. “Reinforcement Learning: An Overview.” **arXiv Preprint arXiv:2412.05265**.

Nadaraya, Elizbar A. 1964. “On Estimating Regression.” **Theory of Probability & Its Applications** 9 (1): 141–42. https://doi.org/10.1137/1109020.

Nado, Zachary, Justin M. Gilmer, Christopher J. Shallue, Rohan Anil, and George E. Dahl. 2021. “A Large Batch Optimizer Reality Check: Traditional, Generic Optimizers Suffice Across Batch Sizes.” **arXiv Preprint arXiv:2102.06356**.

Nagabandi, Anusha, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. 2018. “Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning.” **IEEE International Conference on Robotics and Automation (ICRA)**, 7559–66. https://doi.org/10.1109/ICRA.2018.8463189.

Nair, Vinod, and Geoffrey E Hinton. 2010. “Rectified Linear Units Improve Restricted Boltzmann Machines.” **ICML**, 807–14. https://dl.acm.org/doi/10.5555/3104322.3104425.

Nakamoto, Mitsuhiko, Yuexiang Zhai, Anikait Singh, et al. 2023. “Cal-QL: Calibrated Offline RL Pre-Training for Efficient Online Fine-Tuning.” **Advances in Neural Information Processing Systems** 36.

Nakkiran, Preetum, Arwen Bradley, Hattie Zhou, and Madhu Advani. 2025. “Step-by-Step Diffusion: An Elementary Tutorial.” **Foundations and Trends in Computer Graphics and Vision** 17 (1): 1–75. https://doi.org/10.1561/0600000114.

Nanda, Neel. 2023. **Attribution Patching: Activation Patching at Industrial Scale**. Blog post, https://www.neelnanda.io/attribution-patching.

Narayanan, Deepak, Mohammad Shoeybi, Jared Casper, et al. 2021. “Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM.” **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**, 1–15. https://doi.org/10.1145/3458817.3476209.

Nauman, Michal, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. 2024. “Bigger, Regularized, Optimistic: Scaling for Compute and Sample-Efficient

Continuous Control." **Advances in Neural Information Processing Systems** 37.

Naumann, Uwe. 2008. "Optimal Jacobian Accumulation Is NP-Complete." **Mathematical Programming** 112 (2): 427–41.

Neal, Radford M. 2011. "MCMC Using Hamiltonian Dynamics." In **Handbook of Markov Chain Monte Carlo**. Chapman; Hall/CRC. https://doi.org/10.1201/b10905-6.

Nemirovski, Arkadi. 2004. "Prox-Method with Rate of Convergence $O(1/t)$ for Variational Inequalities with Lipschitz Continuous Monotone Operators and Smooth Convex-Concave Saddle Point Problems." **SIAM Journal on Optimization** 15 (1): 229–51. https://doi.org/10.1137/S1052623403425629.

Nemirovski, Arkadi, and David Yudin. 1983. **Problem Complexity and Method Efficiency in Optimization**. Wiley.

Nesterov, Yu. 2018. **Lectures on Convex Optimization**. Springer. https://doi.org/10.1007/978-3-319-91578-4.

Nesterov, Yurii. 1983. "A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$." **Soviet Mathematics Doklady** 27 (2): 372–76.

Nesterov, Yurii. 2005. "Smooth Minimization of Non-Smooth Functions." **Mathematical Programming** 103 (1): 127–52. https://doi.org/10.1007/s10107-004-0552-5.

Nesterov, Yurii. 2012. "Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems." **SIAM Journal on Optimization** 22 (2): 341–62.

Ney, Hermann, Ute Essen, and Reinhard Kneser. 1994. "On Structuring Probabilistic Dependences in Stochastic Language Modelling." **Computer Speech & Language** 8 (1): 1–38. https://doi.org/10.1006/csla.1994.1001.

Neyman, Jerzy. 1937. "Outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability." **Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences** 236 (767): 333–80. https://doi.org/10.2307/jj.8501421.24.

Ng, Andrew Y, Daishi Harada, and Stuart Russell. 1999. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping." **International Conference on Machine Learning**, 278–87.

Ng, Andrew Y., and Michael I. Jordan. 2002. "On Discriminative Vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes." **Advances in Neural Information Processing Systems** 14: 841–48.

Nguyen, Eric, Michael Poli, Marjan Faizi, et al. 2023. "HyenaDNA: Long-Range Genomic Sequence Modeling at Single Nucleotide Resolution." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2306.15794.

Nguyen, Minh Nhat, Andrew Baker, Clement Neo, Allen Roush, Andreas Kirsch, and Ravid

Shwartz-Ziv. 2025. “Turning up the Heat: Min-p Sampling for Creative and Coherent LLM Outputs.” **International Conference on Learning Representations**. https://arxiv.org/abs/2407.01082.

Nguyen, XuanLong, Martin J. Wainwright, and Michael I. Jordan. 2009. “On Surrogate Loss Functions and f-Divergences.” **The Annals of Statistics** 37 (2): 876–904.

Nguyen, XuanLong, Martin J. Wainwright, and Michael I. Jordan. 2010. “Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization.” **IEEE Transactions on Information Theory** 56 (11): 5847–61.

Nichol, Alexander Quinn, and Prafulla Dhariwal. 2021. “Improved Denoising Diffusion Probabilistic Models.” **Proceedings of the 38th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 139: 8162–71. https://arxiv.org/abs/2102.09672.

Nie, Shen, Fengqi Zhu, Chao Du, et al. 2025. “Scaling up Masked Diffusion Models on Text.” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.18514.

Nie, Shen, Fengqi Zhu, Zebin You, et al. 2025. “Large Language Diffusion Models.” **Advances in Neural Information Processing Systems** 38. https://arxiv.org/abs/2502.09992.

Nigam, Kamal, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. 2000. “Text Classification from Labeled and Unlabeled Documents Using EM.” **Machine Learning** 39 (2–3): 103–34. https://doi.org/10.1023/A:1007692713085.

Nikishin, Evgenii, Max Schwarzer, Pierluca D’Oro, Pierre-Luc Bacon, and Aaron Courville. 2022. “The Primacy Bias in Deep Reinforcement Learning.” **International Conference on Machine Learning**. https://arxiv.org/abs/2205.07802.

Nocedal, Jorge, and Stephen J. Wright. 2006. **Numerical Optimization**. 2nd ed. Springer.

Nota, Chris, and Philip S. Thomas. 2020. “Is the Policy Gradient a Gradient?” **Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems**.

Nowozin, Sebastian, Botond Cseke, and Ryota Tomioka. 2016. “F-GAN: Training Generative Neural Samplers Using Variational Divergence Minimization.” **Advances in Neural Information Processing Systems** 29.

NVIDIA. 2025. “Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models.” **arXiv Preprint arXiv:2504.03624**.

Odena, Augustus, Vincent Dumoulin, and Chris Olah. 2016. “Deconvolution and Checkerboard Artifacts.” **Distill**. http://distill.pub/2016/deconv-checkerboard.

Odena, Augustus, Christopher Olah, and Jonathon Shlens. 2017. “Conditional Image Synthesis with Auxiliary Classifier GANs.” **Proceedings of the 34th International Conference on Machine Learning**, 2642–51. https://arxiv.org/abs/1610.09585.

Øksendal, Bernt. 2003. **Stochastic Differential Equations: An Introduction with Applications**. 6th ed. Springer.

Olsson, Catherine, Nelson Elhage, Neel Nanda, et al. 2022. “In-Context Learning and Induction Heads.” **Transformer Circuits Thread**.

Oord, Aaron van den, Yazhe Li, and Oriol Vinyals. 2018. “Representation Learning with Contrastive Predictive Coding.” **arXiv Preprint arXiv:1807.03748**.

OpenAI. 2025. “Gpt-Oss-120b & Gpt-Oss-20b Model Card.” **arXiv Preprint arXiv:2508.10925**.

Oquab, Maxime, Timothée Darcet, Théo Moutakanni, et al. 2024. “DINOv2: Learning Robust Visual Features Without Supervision.” **Transactions on Machine Learning Research**. https://arxiv.org/abs/2304.07193.

Osband, Ian, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. “Deep Exploration via Bootstrapped DQN.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/1602.04621.

Oseledets, Ivan V. 2011. “Tensor-Train Decomposition.” **SIAM Journal on Scientific Computing** 33 (5): 2295–317.

Oswald, Johannes von, Eyvind Niklasson, Ettore Randazzo, et al. 2023. “Transformers Learn in-Context by Gradient Descent.” **International Conference on Machine Learning**, Proceedings of machine learning research, vol. 202. https://arxiv.org/abs/2212.07677.

Ou, Jingyang, Shen Nie, Kaiwen Xue, et al. 2025. “Your Absorbing Discrete Diffusion Secretly Models the Conditional Distributions of Clean Data.” **International Conference on Learning Representations**. https://arxiv.org/abs/2406.03736.

Ouyang, Long, Jeff Wu, Xu Jiang, et al. 2022. “Training Language Models to Follow Instructions with Human Feedback.” **ArXiv:2203.02155**. https://arxiv.org/abs/2203.02155.

Owen, Art B. 2013. **Monte Carlo Theory, Methods and Examples**.

Paley, Raymond E. A. C., Norbert Wiener, and Antoni Zygmund. 1933. “Notes on Random Functions.” **Mathematische Zeitschrift** 37: 647–68.

Papamakarios, George, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. 2021. “Normalizing Flows for Probabilistic Modeling and Inference.” **Journal of Machine Learning Research** 22 (57): 1–64.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. “BLEU: A Method for Automatic Evaluation of Machine Translation.” **Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics**, 311–18. https://doi.org/10.3115/1073083.1073135.

Pardo, Fabio, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. 2018. “Time Limits in Reinforcement Learning.” **International Conference on Machine Learning**, 4045–54.

https://arxiv.org/abs/1712.00378.

Parikh, Ankur P, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. “A Decomposable Attention Model for Natural Language Inference.” **ArXiv:1606.01933**. https://arxiv.org/abs/1606.01933.

Parikh, Neal, and Stephen Boyd. 2014. “Proximal Algorithms.” **Foundations and Trends in Optimization** 1 (3): 127–239. https://doi.org/10.1561/2400000003.

Park, Taesung, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. 2019. “Semantic Image Synthesis with Spatially-Adaptive Normalization.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 2337–46. https://doi.org/10.1109/cvpr.2019.00244.

Parmar, Gaurav, Richard Zhang, and Jun-Yan Zhu. 2022. “On Aliased Resizing and Surprising Subtleties in GAN Evaluation.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 11410–20. https://arxiv.org/abs/2104.11222.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. “On the Difficulty of Training Recurrent Neural Networks.” **International Conference on Machine Learning**, 1310–18.

Paster, Keiran, Sheila A. McIlraith, and Jimmy Ba. 2022. “You Can’t Count on Luck: Why Decision Transformers and RvS Fail in Stochastic Environments.” **Advances in Neural Information Processing Systems** 35.

Paszke, Adam, Sam Gross, Francisco Massa, et al. 2019. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” **Advances in Neural Information Processing Systems** 32: 8026–37. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Patarasuk, Pitch, and Xin Yuan. 2009. “Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations.” **Journal of Parallel and Distributed Computing** 69 (2): 117–24. https://doi.org/10.1016/j.jpdc.2008.09.002.

Pathak, Deepak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. “Curiosity-Driven Exploration by Self-Supervised Prediction.” **International Conference on Machine Learning**. https://arxiv.org/abs/1705.05363.

Pearlmutter, Barak A. 1994. “Fast Exact Multiplication by the Hessian.” **Neural Computation** 6 (1): 147–60. https://doi.org/10.1162/neco.1994.6.1.147.

Peebles, William, and Saining Xie. 2023. “Scalable Diffusion Models with Transformers.” **Proceedings of the IEEE/CVF International Conference on Computer Vision**, 4195–205. https://arxiv.org/abs/2212.09748.

Peng, Bowen, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. “YaRN: Efficient Context Window Extension of Large Language Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2309.00071.

Peng, Bo, Ruichong Zhang, Daniel Goldstein, et al. 2025. “RWKV-7 ”Goose” with Expressive Dynamic State Evolution.” **arXiv Preprint arXiv:2503.14456**.

Pennington, Jeffrey, Samuel Schoenholz, and Surya Ganguli. 2017. “Resurrecting the Sigmoid in Deep Learning Through Dynamical Isometry: Theory and Practice.” **Advances in Neural Information Processing Systems**, 4785–95. https://proceedings.neurips.cc/paper/2017/hash/a9fc2d3b0c721b5b3f0b3f11bb24a75c-Abstract.html.

Perez, Ethan, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. 2018. “FiLM: Visual Reasoning with a General Conditioning Layer.” **Proceedings of the AAAI Conference on Artificial Intelligence** 32. https://arxiv.org/abs/1709.07871.

Peters, Jonas, Dominik Janzing, and Bernhard Schölkopf. 2017. **Elements of Causal Inference: Foundations and Learning Algorithms**. MIT Press. https://doi.org/10.7551/mitpress/11283.001.0001.

Pethick, Thomas, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. 2025. “Training Deep Learning Models with Norm-Constrained LMOs.” **International Conference on Machine Learning**.

Peyré, Gabriel, and Marco Cuturi. 2019. “Computational Optimal Transport.” **Foundations and Trends in Machine Learning** 11 (5–6): 355–607.

Pimentel, João Felipe, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. “A Large-Scale Study about Quality and Reproducibility of Jupyter Notebooks.” **IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)**, 507–17. https://doi.org/10.1109/MSR.2019.00077.

Pinsker, Mark S. 1964. **Information and Information Stability of Random Variables and Processes**. Holden-Day.

Pitman, Edwin J. G. 1936. “Sufficient Statistics and Intrinsic Accuracy.” **Mathematical Proceedings of the Cambridge Philosophical Society** 32 (4): 567–79.

Planck, Max. 1917. “Über Einen Satz Der Statistischen Dynamik Und Seine Erweiterung in Der Quantentheorie.” **Sitzungsberichte Der Preussischen Akademie Der Wissenschaften**, 324–41.

Platt, John C. 1999. “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods.” In **Advances in Large Margin Classifiers**, edited by Alexander J. Smola, Peter Bartlett, Bernhard Schölkopf, and Dale Schuurmans. MIT Press.

Pleiss, Geoff, Danlu Chen, Gao Huang, Tongcheng Li, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. “Memory-Efficient Implementation of Densenets.” **ArXiv:1707.06990**. https://arxiv.org/abs/1707.06990.

Poli, Michael, Stefano Massaroli, Eric Nguyen, et al. 2023. “Hyena Hierarchy: Towards Larger Convolutional Language Models.” **International Conference on Machine Learning**. https://arxiv.org/abs/2302.10866.

Polyak, Adam, Amit Zohar, Andrew Brown, et al. 2024. “Movie Gen: A Cast of Media Foundation Models.” **arXiv Preprint arXiv:2410.13720**. https://arxiv.org/abs/2410.13720.

Polyak, Boris T. 1964. “Some Methods of Speeding up the Convergence of Iteration Methods.” **USSR Computational Mathematics and Mathematical Physics** 4 (5): 1–17. https://doi.org/10.1016/0041-5553(64)90137-5.

Polyak, Boris T. 1963. “Gradient Methods for the Minimisation of Functionals.” **USSR Computational Mathematics and Mathematical Physics** 3 (4): 864–78.

Polyak, Boris T., and Anatoli B. Juditsky. 1992. “Acceleration of Stochastic Approximation by Averaging.” **SIAM Journal on Control and Optimization** 30 (4): 838–55.

Pomerleau, Dean A. 1989. “ALVINN: An Autonomous Land Vehicle in a Neural Network.” In **Advances in Neural Information Processing Systems 1**, edited by David S Touretzky. Morgan Kaufmann.

Pomerleau, Dean A. 1991. “Efficient Training of Artificial Neural Networks for Autonomous Navigation.” **Neural Computation** 3 (1): 88–97.

Pontryagin, Lev S., Vladimir G. Boltyanskii, Revaz V. Gamkrelidze, and Evgenii F. Mishchenko. 1962. **The Mathematical Theory of Optimal Processes**. Interscience.

Pooladian, Aram-Alexandre, Heli Ben-Hamu, Carles Domingo-Enrich, Brandon Amos, Yaron Lipman, and Ricky T. Q. Chen. 2023. “Multisample Flow Matching: Straightening Flows with Minibatch Couplings.” **Proceedings of the 40th International Conference on Machine Learning**.

Poole, Ben, Sherjil Ozair, Aaron van den Oord, Alexander A. Alemi, and George Tucker. 2019. “On Variational Bounds of Mutual Information.” **Proceedings of the 36th International Conference on Machine Learning**, 5171–80.

Popović, Maja. 2015. “ChrF: Character n-Gram F-Score for Automatic MT Evaluation.” **Proceedings of the Tenth Workshop on Statistical Machine Translation**, 392–95. https://doi.org/10.18653/v1/W15-3049.

Post, Matt. 2018. “A Call for Clarity in Reporting BLEU Scores.” **Proceedings of the Third Conference on Machine Translation: Research Papers**, 186–91. https://doi.org/10.18653/v1/W18-6319.

Power, Alethea, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. 2022. “Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets.” **ArXiv:2201.02177**. https://arxiv.org/abs/2201.02177.

Prabhudesai, Mihir, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. 2025. “Diffusion Beats Autoregressive in Data-Constrained Settings.” **arXiv Preprint arXiv:2507.15857**. https://arxiv.org/abs/2507.15857.

Press, Ofir, Noah A. Smith, and Mike Lewis. 2022. “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation.” **International Conference on**

Learning Representations. https://arxiv.org/abs/2108.12409.

Press, Ofir, and Lior Wolf. 2017. "Using the Output Embedding to Improve Language Models." **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics**, 157–63.

Puterman, Martin L. 1994. **Markov Decision Processes: Discrete Stochastic Dynamic Programming**. John Wiley & Sons. https://doi.org/10.1002/9780470316887.

Qiu, Zihan, Zekun Wang, Bo Zheng, et al. 2025. "Gated Attention for Large Language Models: Non-Linearity, Sparsity, and Attention-Sink-Free." **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/2505.06708.

Qwen Team. 2025. **Qwen3-Next: Towards Ultimate Training and Inference Efficiency**. Model release, https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct.

Rabe, Markus N., and Charles Staats. 2021. "Self-Attention Does Not Need $O(n^2)$ Memory." **arXiv Preprint arXiv:2112.05682**.

Rabiner, Lawrence, and Biing-Hwang Juang. 1993. **Fundamentals of Speech Recognition**. Prentice-Hall.

Rademacher, Hans. 1919. "Über Partielle Und Totale Differenzierbarkeit von Funktionen Mehrerer Variabeln Und über Die Transformation Der Doppelintegrale." **Mathematische Annalen** 79 (4): 340–59.

Radford, Alec, Jong Wook Kim, Chris Hallacy, et al. 2021. "Learning Transferable Visual Models from Natural Language Supervision." **International Conference on Machine Learning**, 8748–63. https://proceedings.mlr.press/v139/radford21a.html.

Radford, Alec, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. "Robust Speech Recognition via Large-Scale Weak Supervision." **International Conference on Machine Learning**. https://arxiv.org/abs/2212.04356.

Radford, Alec, Luke Metz, and Soumith Chintala. 2015. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." **ArXiv:1511.06434**. https://arxiv.org/abs/1511.06434.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. "Improving Language Understanding by Generative Pre-Training." **OpenAI**. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. "Language Models Are Unsupervised Multitask Learners." **OpenAI Blog** 1 (8): 9. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Radhakrishna Rao, C. 1945. "Information and Accuracy Attainable in the Estimation of Statistical Parameters." **Bulletin of the Calcutta Mathematical Society** 37 (3): 81–91. https://doi.org/10.1007/BF02908227.

Radosavovic, Ilija, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. 2019. "On Network Design Spaces for Visual Recognition." **Proceedings of the IEEE/CVF International Conference on Computer Vision**, 1882–90. https://doi.org/10.1109/iccv.2019.00052.

Radosavovic, Ilija, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. "Designing Network Design Spaces." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 10428–36. https://doi.org/10.1109/cvpr42600.2020.01044.

Rafailov, Rafael, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. "Direct Preference Optimization: Your Language Model Is Secretly a Reward Model." **Advances in Neural Information Processing Systems** 36.

Raffel, Colin, Noam Shazeer, Adam Roberts, et al. 2020. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." **Journal of Machine Learning Research** 21: 1–67. https://arxiv.org/abs/1910.10683.

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. "Stable-Baselines3: Reliable Reinforcement Learning Implementations." **Journal of Machine Learning Research** 22 (268): 1–8.

Rahimi, Ali, and Benjamin Recht. 2007. "Random Features for Large-Scale Kernel Machines." **Advances in Neural Information Processing Systems** 20.

Rajbhandari, Samyam, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models." **SC20: International Conference for High Performance Computing, Networking, Storage and Analysis**, 1–16.

Rajbhandari, Samyam, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. "ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning." **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**. https://arxiv.org/abs/2104.07857.

Ramapuram, Jason, Federico Danieli, Eeshan Dhekane, et al. 2025. "Theory, Analysis, and Best Practices for Sigmoid Self-Attention." **International Conference on Learning Representations**. https://arxiv.org/abs/2409.04431.

Ramdas, Aaditya, Sashank J. Reddi, Barnabás Póczos, Aarti Singh, and Larry Wasserman. 2015. "On the Decreasing Power of Kernel and Distance Based Nonparametric Hypothesis Tests in High Dimensions." **Proceedings of the AAAI Conference on Artificial Intelligence** 29.

Ramesh, Aditya, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. "Hierarchical Text-Conditional Image Generation with Clip Latents." **ArXiv:2204.06125**. https://arxiv.org/abs/2204.06125.

Ramsauer, Hubert, Bernhard Schäfl, Johannes Lehner, et al. 2021. "Hopfield Networks Is All You Need." **International Conference on Learning Representations**. https://arxiv.org/abs/2008.02217.

Randløv, Jette, and Preben Alstrøm. 1998. “Learning to Drive a Bicycle Using Reinforcement Learning and Shaping.” **Proceedings of the Fifteenth International Conference on Machine Learning**, 463–71.

Ranzato, Marc’Aurelio, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. “Sequence Level Training with Recurrent Neural Networks.” **International Conference on Learning Representations**. https://arxiv.org/abs/1511.06732.

Rashidinejad, Paria, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. 2021. “Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism.” **Advances in Neural Information Processing Systems** 34.

Rawlings, James B., David Q. Mayne, and Moritz Diehl. 2017. **Model Predictive Control: Theory, Computation, and Design**. 2nd ed. Nob Hill Publishing.

Recht, Benjamin, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. “Do ImageNet Classifiers Generalize to ImageNet?” **International Conference on Machine Learning**, 5389–400. https://arxiv.org/abs/1902.10811.

Reddi, Sashank J, Satyen Kale, and Sanjiv Kumar. 2018. “On the Convergence of Adam and Beyond.” **International Conference on Learning Representations**. https://arxiv.org/abs/1904.09237.

Rei, Ricardo, Craig Stewart, Ana C. Farinha, and Alon Lavie. 2020. “COMET: A Neural Framework for MT Evaluation.” **Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing**, 2685–702. https://doi.org/10.18653/v1/2020.emnlp-main.213.

Ren, Jie, Samyam Rajbhandari, Reza Yazdani Aminabadi, et al. 2021. “ZeRO-Offload: Democratizing Billion-Scale Model Training.” **2021 USENIX Annual Technical Conference (USENIX ATC 21)**, 551–64. https://www.usenix.org/conference/atc21/presentation/ren-jie.

Rényi, Alfréd. 1961. “On Measures of Entropy and Information.” **Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics**, 547–61.

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra. 2014. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models.” **International Conference on Machine Learning**, 1278–86. https://proceedings.mlr.press/v32/rezende14.html.

Rezende, Danilo, and Shakir Mohamed. 2015. “Variational Inference with Normalizing Flows.” **International Conference on Machine Learning**, 1530–38. https://proceedings.mlr.press/v37/rezende15.html.

Richardson, Tom, and Rüdiger Urbanke. 2008. **Modern Coding Theory**. Cambridge University Press. https://doi.org/10.1017/CBO9780511791338.

Riesenhuber, Maximilian, and Tomaso Poggio. 1999. “Hierarchical Models of Object Recog-

nition in Cortex." **Nature Neuroscience** 2 (11): 1019–25. https://doi.org/10.1038/14819.

Risken, Hannes. 1996. **The Fokker–Planck Equation: Methods of Solution and Applications**. 2nd ed. Springer.

Rissanen, Jorma J. 1976. "Generalized Kraft Inequality and Arithmetic Coding." **IBM Journal of Research and Development** 20 (3): 198–203.

Robbins, Herbert. 1952. "Some Aspects of the Sequential Design of Experiments." **Bulletin of the American Mathematical Society** 58 (5): 527–35. https://doi.org/10.1090/S0002-9904-1952-09620-8.

Robbins, Herbert. 1956. "An Empirical Bayes Approach to Statistics." **Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1**, 157–63.

Robbins, Herbert, and Sutton Monro. 1951. "A Stochastic Approximation Method." **The Annals of Mathematical Statistics** 22 (3): 400–407.

Roberts, Gareth O., Andrew Gelman, and Walter R. Gilks. 1997. "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms." **The Annals of Applied Probability** 7 (1): 110–20. https://doi.org/10.1214/aoap/1034625254.

Roberts, Gareth O., and Richard L. Tweedie. 1996. "Exponential Convergence of Langevin Distributions and Their Discrete Approximations." **Bernoulli** 2 (4): 341–63.

Rockafellar, R. T. 1970. **Convex Analysis**. Princeton University Press. https://doi.org/10.1515/9781400873173.

Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. "High-Resolution Image Synthesis with Latent Diffusion Models." **IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 10684–95.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. 2015. "U-Net: Convolutional Networks for Biomedical Image Segmentation." **Medical Image Computing and Computer-Assisted Intervention (MICCAI)**, Lecture notes in computer science, vol. 9351: 234–41. https://arxiv.org/abs/1505.04597.

Ross, Stéphane, and J Andrew Bagnell. 2010. "Efficient Reductions for Imitation Learning." **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**, Proceedings of machine learning research, vol. 9: 661–68.

Ross, Stéphane, Geoffrey J Gordon, and J Andrew Bagnell. 2011. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." **International Conference on Artificial Intelligence and Statistics**, 627–35.

Rouhani, Bita Darvish, Ritchie Zhao, Ankit More, et al. 2023. "Microscaling Data Formats for Deep Learning." **arXiv Preprint arXiv:2310.10537**.

Rubanova, Yulia, Ricky T. Q. Chen, and David Duvenaud. 2019. "Latent Ordinary Differen-

tial Equations for Irregularly-Sampled Time Series." **Advances in Neural Information Processing Systems** 32.

Rubinstein, Reuven Y., and Dirk P. Kroese. 2004. **The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning**. Springer. https://doi.org/10.1007/978-1-4757-4321-0.

Rudin, Nikita, David Hoeller, Philipp Reist, and Marco Hutter. 2022. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning." **Conference on Robot Learning**. https://arxiv.org/abs/2109.11978.

Rudin, Walter. 1976. **Principles of Mathematical Analysis**. 3rd ed. McGraw-Hill.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. "Learning Representations by Back-Propagating Errors." **Nature** 323 (6088): 533–36. https://doi.org/10.1038/323533a0.

Rummery, Gavin A, and Mahesan Niranjan. 1994. **On-Line Q-Learning Using Connectionist Systems**. CUED/F-INFENG/TR 166. Cambridge University Engineering Department.

Russakovsky, Olga, Jia Deng, Hao Su, et al. 2015. "ImageNet Large Scale Visual Recognition Challenge." **International Journal of Computer Vision** 115 (3): 211–52. https://doi.org/10.1007/s11263-015-0816-y.

Sadat, Seyedmorteza, Otmar Hilliges, and Romann M. Weber. 2025. "Eliminating Oversaturation and Artifacts of High Guidance Scales in Diffusion Models." **International Conference on Learning Representations**. https://arxiv.org/abs/2410.02416.

Saerens, Marco, Patrice Latinne, and Christine Decaestecker. 2002. "Adjusting the Outputs of a Classifier to New a Priori Probabilities: A Simple Procedure." **Neural Computation** 14 (1): 21–41. https://doi.org/10.1162/089976602753284446.

Sahami, Mehran, Susan Dumais, David Heckerman, and Eric Horvitz. 1998. "A Bayesian Approach to Filtering Junk e-Mail." **AAAI Workshop on Learning for Text Categorization**.

Saharia, Chitwan, William Chan, Saurabh Saxena, et al. 2022. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding." **ArXiv:2205.11487**. https://arxiv.org/abs/2205.11487.

Sahoo, Subham Sekhar, Marianne Arriola, Yair Schiff, et al. 2024. "Simple and Effective Masked Diffusion Language Models." **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2406.07524.

Sahoo, Subham Sekhar, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. 2025. "The Diffusion Duality." **International Conference on Machine Learning**.

Salimans, Tim, and Jonathan Ho. 2022. "Progressive Distillation for Fast Sampling of Diffusion Models." **International Conference on Learning Representations**.

Samuel, Arthur L. 1959. “Some Studies in Machine Learning Using the Game of Checkers.” **IBM Journal of Research and Development** 3 (3): 210–29. https://doi.org/10.1147/rd.33.0210.

Samuelson, Paul A. 1965. “Rational Theory of Warrant Pricing.” **Industrial Management Review** 6 (2): 13–31.

Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. “MobileNetV2: Inverted Residuals and Linear Bottlenecks.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. https://arxiv.org/abs/1801.04381.

Santurkar, Shibani, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. 2018. “How Does Batch Normalization Help Optimization?” **Advances in Neural Information Processing Systems**, 2483–93. https://doi.org/10.5555/3327345.3327508.

Sardana, Nikhil, Jacob Portes, Sasha Doubov, and Jonathan Frankle. 2024. “Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws.” **International Conference on Machine Learning**.

Särkkä, Simo, and Arno Solin. 2019. **Applied Stochastic Differential Equations**. Cambridge University Press.

Sauer, Axel, Frederic Boesel, Tim Dockhorn, Andreas Blattmann, Patrick Esser, and Robin Rombach. 2024. “Fast High-Resolution Image Synthesis with Latent Adversarial Diffusion Distillation.” **SIGGRAPH Asia 2024 Conference Papers**.

Sauer, Axel, Kashyap Chitta, Jens Müller, and Andreas Geiger. 2021. “Projected GANs Converge Faster.” **Advances in Neural Information Processing Systems** 34.

Sauer, Axel, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. 2023. “StyleGAN-T: Unlocking the Power of GANs for Fast Large-Scale Text-to-Image Synthesis.” **Proceedings of the 40th International Conference on Machine Learning**.

Sauer, Axel, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. 2023. “Adversarial Diffusion Distillation.” **arXiv Preprint arXiv:2311.17042**.

Saxe, Andrew M., Yamini Bansal, Joel Dapello, et al. 2018. “On the Information Bottleneck Theory of Deep Learning.” **International Conference on Learning Representations**.

Saxe, Andrew M., James L. McClelland, and Surya Ganguli. 2014. “Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks.” **International Conference on Learning Representations**. https://arxiv.org/abs/1312.6120.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. 2016. “Prioritized Experience Replay.” **International Conference on Learning Representations**. https://arxiv.org/abs/1511.05952.

Schiff, Yair, Chia-Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. 2024. “Caduceus: Bi-Directional Equivariant Long-Range DNA Sequence Modeling.” **In-**

ternational Conference on Machine Learning. https://arxiv.org/abs/2403.03234.

Schlag, Imanol, Kazuki Irie, and Jürgen Schmidhuber. 2021. "Linear Transformers Are Secretly Fast Weight Programmers." **International Conference on Machine Learning**.

Schmidt, Robin M., Frank Schneider, and Philipp Hennig. 2021. "Descending Through a Crowded Valley — Benchmarking Deep Learning Optimizers." **Proceedings of the 38th International Conference on Machine Learning**, 9367–76. https://arxiv.org/abs/2007.01547.

Schölkopf, Bernhard, and Alexander J Smola. 2002. **Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond**. MIT Press. https://doi.org/10.7551/mitpress/4175.001.0001.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, et al. 2020. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model." **Nature** 588 (7839): 604–9. https://doi.org/10.1038/s41586-020-03051-0.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. "Trust Region Policy Optimization." **International Conference on Machine Learning**, 1889–97.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. "High-Dimensional Continuous Control Using Generalized Advantage Estimation." **International Conference on Learning Representations**.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. "Proximal Policy Optimization Algorithms." **arXiv Preprint arXiv:1707.06347**.

Schuster, Mike, and Kuldip K Paliwal. 1997. "Bidirectional Recurrent Neural Networks." **IEEE Transactions on Signal Processing** 45 (11): 2673–81. https://doi.org/10.1109/78.650093.

Seitzer, Maximilian, Arash Tavakoli, Dimitrije Antic, and Georg Martius. 2022. "On the Pitfalls of Heteroscedastic Uncertainty Estimation with Probabilistic Neural Networks." **International Conference on Learning Representations**. https://arxiv.org/abs/2203.09168.

Semenov, Andrei, Matteo Pagliardini, and Martin Jaggi. 2025. "Benchmarking Optimizers for Large Language Model Pretraining." **arXiv Preprint arXiv:2509.01440**.

Sennrich, Rico, Barry Haddow, and Alexandra Birch. 2016. "Neural Machine Translation of Rare Words with Subword Units." **Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics**, 1715–25. https://doi.org/10.18653/v1/P16-1162.

Sergeev, Alexander, and Mike Del Balso. 2018. "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow." **ArXiv:1802.05799**. https://arxiv.org/abs/1802.05799.

Shah, Ishaan, Anthony M. Polloreno, Karl Stratos, et al. 2025. "Practical Efficiency of Muon

for Pretraining." **arXiv Preprint arXiv:2505.02222**.

Shah, Jay, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. "FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-Precision." **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2407.08608.

Shallue, Christopher J., Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. 2019. "Measuring the Effects of Data Parallelism on Neural Network Training." **Journal of Machine Learning Research** 20 (112): 1–49. https://arxiv.org/abs/1811.03600.

Shannon, Claude E. 1959. "Coding Theorems for a Discrete Source with a Fidelity Criterion." **IRE National Convention Record** 7 (4): 142–63.

Shannon, Claude Elwood. 1948. "A Mathematical Theory of Communication." **The Bell System Technical Journal** 27 (3): 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Shannon, Claude Elwood. 1951. "Prediction and Entropy of Printed English." **The Bell System Technical Journal** 30 (1): 50–64. https://doi.org/10.1002/j.1538-7305.1951.tb01366.x.

Shannon, Matt, Ben Poole, Soroosh Mariooryad, et al. 2020. "Non-Saturating GAN Training as Divergence Minimization." **arXiv Preprint arXiv:2010.08029**.

Shao, Zhihong, Peiyi Wang, Qihao Zhu, et al. 2024. "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models." **arXiv Preprint arXiv:2402.03300**.

Shaw, Peter, Jakob Uszkoreit, and Ashish Vaswani. 2018. "Self-Attention with Relative Position Representations." **ArXiv:1803.02155**. https://arxiv.org/abs/1803.02155.

Shazeer, Noam. 2019. "Fast Transformer Decoding: One Write-Head Is All You Need." **arXiv Preprint arXiv:1911.02150**.

Shazeer, Noam. 2020. "GLU Variants Improve Transformer." **ArXiv:2002.05202**. https://arxiv.org/abs/2002.05202.

Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarz, et al. 2017. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer." **International Conference on Learning Representations**.

Shazeer, Noam, and Mitchell Stern. 2018. "Adafactor: Adaptive Learning Rates with Sublinear Memory Cost." **International Conference on Machine Learning**. https://arxiv.org/abs/1804.04235.

Shen, Lingfeng, Aayush Mishra, and Daniel Khashabi. 2024. "Position: Do Pretrained Transformers Learn in-Context by Gradient Descent?" **International Conference on Machine Learning**, Proceedings of machine learning research, vol. 235: 44712–40. https://arxiv.org/abs/2310.08540.

Shi, Jiaxin, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. 2024. “Simplified and Generalized Masked Diffusion for Discrete Data.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2406.04329.

Shimodaira, Hidetoshi. 2000. “Improving Predictive Inference Under Covariate Shift by Weighting the Log-Likelihood Function.” **Journal of Statistical Planning and Inference** 90 (2): 227–44.

Shoeybi, Mohammad, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.” **ArXiv:1909.08053**. https://arxiv.org/abs/1909.08053.

Shwartz-Ziv, Ravid, and Amitai Armon. 2022. “Tabular Data: Deep Learning Is Not All You Need.” **Information Fusion** 81: 84–90. https://doi.org/10.1016/j.inffus.2021.11.011.

Shwartz-Ziv, Ravid, and Naftali Tishby. 2017. “Opening the Black Box of Deep Neural Networks via Information.” **arXiv Preprint arXiv:1703.00810**.

Siems, Julien, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. 2025. “DeltaProduct: Improving State-Tracking in Linear RNNs via Householder Products.” **arXiv Preprint arXiv:2502.10297**.

Silver, David, Aja Huang, Chris J Maddison, et al. 2016. “Mastering the Game of Go with Deep Neural Networks and Tree Search.” **Nature** 529 (7587): 484–89. https://doi.org/10.1038/nature16961.

Silver, David, Thomas Hubert, Julian Schrittwieser, et al. 2018. “A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go Through Self-Play.” **Science** 362 (6419): 1140–44. https://doi.org/10.1126/science.aar6404.

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. “Deterministic Policy Gradient Algorithms.” **Proceedings of the 31st International Conference on Machine Learning**, Proceedings of machine learning research, vol. 32: 387–95. https://proceedings.mlr.press/v32/silver14.html.

Silver, David, Julian Schrittwieser, Karen Simonyan, et al. 2017. “Mastering the Game of Go Without Human Knowledge.” **Nature** 550 (7676): 354–59. https://doi.org/10.1038/nature24270.

Silverman, B. W. 1986. **Density Estimation for Statistical and Data Analysis**. Chapman; Hall.

Simonyan, Karen, and Andrew Zisserman. 2015. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” **International Conference on Learning Representations**. https://arxiv.org/abs/1409.1556.

Simsekli, Umut, Levent Sagun, and Mert Gurbuzbalaban. 2019. “A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks.” **Proceedings of the 36th International Conference on Machine Learning**, Proceedings of machine learning research, vol. 97: 5827–37. https://proceedings.mlr.press/v97/simsekli19a.html.

Sivic, Josef, and Andrew Zisserman. 2003. “Video Google: A Text Retrieval Approach to Object Matching in Videos.” **Proceedings of the IEEE International Conference on Computer Vision** 3: 1470–77. https://doi.org/10.1109/iccv.2003.1238663.

Slater, Morton. 1950. **Lagrange Multipliers Revisited**. Discussion Paper Mathematics 403. Cowles Commission for Research in Economics.

Smith, Daniel G. A., and Johnnie Gray. 2018. “Opt_einsum: A Python Package for Optimizing Contraction Order for Einsum-Like Expressions.” **Journal of Open Source Software** 3 (26): 753.

Smith, Jimmy T. H., Andrew Warrington, and Scott W. Linderman. 2023. “Simplified State Space Layers for Sequence Modeling.” **International Conference on Learning Representations**. https://arxiv.org/abs/2208.04933.

Smith, Samuel L., Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 2018. “Don’t Decay the Learning Rate, Increase the Batch Size.” **International Conference on Learning Representations**. https://arxiv.org/abs/1711.00489.

Sohl-Dickstein, Jascha, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. “Deep Unsupervised Learning Using Nonequilibrium Thermodynamics.” **International Conference on Machine Learning**, 2256–65. https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Sønderby, Casper Kaae, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. 2017. “Amortised MAP Inference for Image Super-Resolution.” **International Conference on Learning Representations**.

Song, Jiaming, and Stefano Ermon. 2020. “Understanding the Limitations of Variational Mutual Information Estimators.” **International Conference on Learning Representations**.

Song, Jiaming, Chenlin Meng, and Stefano Ermon. 2021. “Denoising Diffusion Implicit Models.” **International Conference on Learning Representations**.

Song, Yang. 2021. **Generative Modeling by Estimating Gradients of the Data Distribution**. Blog post, https://yang-song.net/blog/2021/score/.

Song, Yang, and Prafulla Dhariwal. 2024. “Improved Techniques for Training Consistency Models.” **International Conference on Learning Representations**. https://arxiv.org/abs/2310.14189.

Song, Yang, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. 2023. “Consistency Models.” **Proceedings of the 40th International Conference on Machine Learning**.

Song, Yang, Conor Durkan, Iain Murray, and Stefano Ermon. 2021. “Maximum Likelihood Training of Score-Based Diffusion Models.” **Advances in Neural Information Processing Systems** 34.

Song, Yang, and Stefano Ermon. 2019. “Generative Modeling by Estimating Gradients of

the Data Distribution." **Advances in Neural Information Processing Systems** 32. https://arxiv.org/abs/1907.05600.

Song, Yang, and Stefano Ermon. 2020. "Improved Techniques for Training Score-Based Generative Models." **Advances in Neural Information Processing Systems** 33. https://arxiv.org/abs/2006.09011.

Song, Yang, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. 2019. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation." **Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence**, Proceedings of machine learning research, vol. 115: 574–84. https://arxiv.org/abs/1905.07088.

Song, Yang, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. "Score-Based Generative Modeling Through Stochastic Differential Equations." **International Conference on Learning Representations**. https://arxiv.org/abs/2011.13456.

Soudry, Daniel, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. 2018. "The Implicit Bias of Gradient Descent on Separable Data." **Journal of Machine Learning Research** 19 (70): 1–57.

Squire, William, and George Trapp. 1998. "Using Complex Variables to Estimate Derivatives of Real Functions." **SIAM Review** 40 (1): 110–12. https://doi.org/10.1137/S003614459631241X.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." **Journal of Machine Learning Research** 15 (1): 1929–58. https://doi.org/10.5555/2627435.2670313.

Stein, Charles. 1986. **Approximate Computation of Expectations**. Vol. 7. Institute of Mathematical Statistics Lecture Notes–Monograph Series. Institute of Mathematical Statistics.

Stein, Charles M. 1981. "Estimation of the Mean of a Multivariate Normal Distribution." **Annals of Statistics** 9 (6): 1135–51.

Stein, George, Jesse C. Cresswell, Rasa Hosseinzadeh, et al. 2023. "Exposing Flaws of Generative Model Evaluation Metrics and Their Unfair Treatment of Diffusion Models." **Advances in Neural Information Processing Systems** 36. https://arxiv.org/abs/2306.04675.

Sterbenz, Pat H. 1974. **Floating-Point Computation**. Prentice-Hall.

Stich, Sebastian U. 2019. "Local SGD Converges Fast and Communicates Little." **International Conference on Learning Representations**. https://arxiv.org/abs/1805.09767.

Stratonovich, Ruslan L. 1966. "A New Representation for Stochastic Integrals and Equations." **SIAM Journal on Control** 4 (2): 362–71.

Strehl, Alexander L., and Michael L. Littman. 2008. “An Analysis of Model-Based Interval Estimation for Markov Decision Processes.” **Journal of Computer and System Sciences** 74 (8): 1309–31. https://doi.org/10.1016/j.jcss.2007.08.009.

Student (Gosset, William S.). 1908. “The Probable Error of a Mean.” **Biometrika** 6 (1): 1–25.

Su, Jianlin, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2024. “RoFormer: Enhanced Transformer with Rotary Position Embedding.” **Neurocomputing** 568: 127063. https://arxiv.org/abs/2104.09864.

Sun, Mingjie, Xinlei Chen, J. Zico Kolter, and Zhuang Liu. 2024. “Massive Activations in Large Language Models.” **Conference on Language Modeling (COLM)**. https://arxiv.org/abs/2402.17762.

Sun, Ruoyu, Tiantian Fang, and Alexander Schwing. 2020. “Towards a Better Global Loss Landscape of GANs.” **Advances in Neural Information Processing Systems** 33.

Sun, Yu, Xinhao Li, Karan Dalal, et al. 2024. “Learning to (Learn at Test Time): RNNs with Expressive Hidden States.” **arXiv Preprint arXiv:2407.04620**.

Sun, Yutao, Li Dong, Shaohan Huang, et al. 2023. “Retentive Network: A Successor to Transformer for Large Language Models.” **arXiv Preprint arXiv:2307.08621**.

Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. 2013. “On the Importance of Initialization and Momentum in Deep Learning.” **International Conference on Machine Learning**, 1139–47. https://proceedings.mlr.press/v28/sutskever13.html.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. 2014. “Sequence to Sequence Learning with Neural Networks.” **Advances in Neural Information Processing Systems**, 3104–12. https://doi.org/10.5555/2969033.2969173.

Sutton, Richard S. 1988. “Learning to Predict by the Methods of Temporal Differences.” **Machine Learning** 3 (1): 9–44. https://doi.org/10.1007/BF00115009.

Sutton, Richard S. 1990. “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming.” **Proceedings of the Seventh International Conference on Machine Learning**, 216–24.

Sutton, Richard S, and Andrew G Barto. 2018. **Reinforcement Learning: An Introduction**. 2nd ed. MIT Press. http://incompleteideas.net/book/RLbook2020.pdf.

Sutton, Richard S, David McAllester, Satinder Singh, and Yishay Mansour. 2000. “Policy Gradient Methods for Reinforcement Learning with Function Approximation.” **Advances in Neural Information Processing Systems** 12.

Szegedy, Christian, Wei Liu, Yangqing Jia, et al. 2015. “Going Deeper with Convolutions.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 1–9. https://doi.org/10.1109/cvpr.2015.7298594.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016.

"Rethinking the Inception Architecture for Computer Vision." **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 2818–26. https://doi.org/10.1109/cvpr.2016.308.

Székely, Gábor J., and Maria L. Rizzo. 2013. "Energy Statistics: A Class of Statistics Based on Distances." **Journal of Statistical Planning and Inference** 143 (8): 1249–72.

Szepesvári, Csaba. 2010. **Algorithms for Reinforcement Learning**. Synthesis Lectures on Artificial Intelligence and Machine Learning 9. Morgan; Claypool. https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf.

Tallec, Corentin, and Yann Ollivier. 2018. "Can Recurrent Neural Networks Warp Time?" **International Conference on Learning Representations**. https://arxiv.org/abs/1804.11188.

Tan, Mingxing, and Quoc Le. 2019. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." **International Conference on Machine Learning**, 6105–14. https://proceedings.mlr.press/v97/tan19a.html.

Tan, Mingxing, and Quoc V. Le. 2021. "EfficientNetV2: Smaller Models and Faster Training." **International Conference on Machine Learning**. https://arxiv.org/abs/2104.00298.

Tang, Yunhao, and Rémi Munos. 2025. "On a Few Pitfalls in KL Divergence Gradient Estimation for RL." **arXiv Preprint arXiv:2506.09477**.

Tao, Terence, and Van Vu. 2010. "Random Matrices: Universality of ESDs and the Circular Law." **Annals of Probability** 38 (5): 2023–65.

Tay, Yi, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. "Efficient Transformers: A Survey." **ArXiv:2009.06732**. https://arxiv.org/abs/2009.06732.

Taylor, Adrien B., Julien M. Hendrickx, and François Glineur. 2018. "Exact Worst-Case Convergence Rates of the Proximal Gradient Method for Composite Convex Minimization." **Journal of Optimization Theory and Applications** 178 (2): 455–76.

Team OLMo. 2025. "Olmo 3." **arXiv Preprint arXiv:2512.13961**.

Team OLMo, Pete Walsh, Luca Soldaini, et al. 2025. "2 OLMo 2 Furious." **arXiv Preprint arXiv:2501.00656**.

Tedrake, Russ. 2023. **Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation**. MIT. https://underactuated.csail.mit.edu/.

Telgarsky, Matus. 2016. "Benefits of Depth in Neural Networks." **Conference on Learning Theory**, 1517–39.

Tesauro, Gerald. 1995. "Temporal Difference Learning and TD-Gammon." **Communications of the ACM** 38 (3): 58–68. https://doi.org/10.1145/203330.203343.

Thakur, Rajeev, Rolf Rabenseifner, and William Gropp. 2005. "Optimization of Collec-

tive Communication Operations in MPICH." **The International Journal of High Performance Computing Applications** 19 (1): 49–66. https://doi.org/10.1177/1094342005051521.

Theis, Lucas, Aäron van den Oord, and Matthias Bethge. 2016. "A Note on the Evaluation of Generative Models." **International Conference on Learning Representations**.

Thomas, Philip. 2014. "Bias in Natural Actor-Critic Algorithms." In **Proceedings of the 31st International Conference on Machine Learning**, edited by Eric P. Xing and Tony Jebara, vol. 32. Proceedings of Machine Learning Research 1.

Thompson, William R. 1933. "On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples." **Biometrika** 25 (3/4): 285–94.

Thrun, Sebastian, and Anton Schwartz. 1993. "Issues in Using Function Approximation for Reinforcement Learning." **Proceedings of the 4th Connectionist Models Summer School**, 255–63.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." **Journal of the Royal Statistical Society: Series B** 58 (1): 267–88.

Tieleman, Tijmen. 2008. "Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient." **Proceedings of the 25th International Conference on Machine Learning**, 1064–71.

Tieleman, Tijmen, and Geoffrey Hinton. 2012. "Divide the Gradient by a Running Average of Its Recent Magnitude." In **COURSERA: Neural Networks for Machine Learning, Lecture 6.5-Rmsprop**. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Tikhonov, A. N., and V. Y. Arsenin. 1977. **Solutions of Ill-Posed Problems**. W.H. Winston. https://doi.org/10.1137/1.9780898719741.

Tillet, Philippe, H. T. Kung, and David Cox. 2019. "Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations." **3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)**, 10–19. https://doi.org/10.1145/3315508.3329973.

Tishby, Naftali, Fernando C. Pereira, and William Bialek. 1999. "The Information Bottleneck Method." **Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing**, 368–77.

Tishby, Naftali, and Noga Zaslavsky. 2015. "Deep Learning and the Information Bottleneck Principle." **IEEE Information Theory Workshop (ITW)**, 1–5. https://doi.org/10.1109/ITW.2015.7133169.

Tobin, Josh, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World." **IEEE/RSJ International Conference on Intelligent Robots and Systems**. https://arxiv.org/abs/1703.06907.

Tong, Alexander, Kilian Fatras, Nikolay Malkin, et al. 2024. “Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport.” **Transactions on Machine Learning Research**.

Touvron, Hugo, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. “Training Data-Efficient Image Transformers & Distillation Through Attention.” **International Conference on Machine Learning**, 10347–57. https://proceedings.mlr.press/v139/touvron21a.html.

Touvron, Hugo, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. 2021. “Going Deeper with Image Transformers.” **IEEE/CVF International Conference on Computer Vision**, 32–42. https://arxiv.org/abs/2103.17239.

Towers, Mark, Ariel Kwiatkowski, Jordan Terry, et al. 2024. “Gymnasium: A Standard Interface for Reinforcement Learning Environments.” **arXiv Preprint arXiv:2407.17032**.

Townsend, James, Thomas Bird, and David Barber. 2019. “Practical Lossless Compression with Latent Variables Using Bits Back Coding.” **International Conference on Learning Representations**. https://arxiv.org/abs/1901.04866.

Trefethen, Lloyd N., and David Bau. 1997. **Numerical Linear Algebra**. SIAM.

Trefethen, Lloyd N., and Mark Embree. 2005. **Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators**. Princeton University Press.

Tschannen, Michael, Josip Djolonga, Paul K. Rubenstein, Sylvain Gelly, and Mario Lucic. 2020. “On Mutual Information Maximization for Representation Learning.” **International Conference on Learning Representations**.

Tseng, Paul. 2001. “Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization.” **Journal of Optimization Theory and Applications** 109 (3): 475–94.

Tsitsiklis, John N. 1994. “Asynchronous Stochastic Approximation and Q-Learning.” **Machine Learning** 16: 185–202. https://doi.org/10.1007/BF00993306.

Tsitsiklis, John N, and Benjamin Van Roy. 1997. “An Analysis of Temporal-Difference Learning with Function Approximation.” **IEEE Transactions on Automatic Control** 42 (5): 674–90. https://doi.org/10.1109/9.580874.

Tsoumakas, Grigorios, and Ioannis Katakis. 2007. “Multi-Label Classification: An Overview.” **International Journal of Data Warehousing and Mining** 3 (3): 1–13. https://doi.org/10.4018/jdwm.2007070101.

Tucker, George, Andriy Mnih, Chris J. Maddison, John Lawson, and Jascha Sohl-Dickstein. 2017. “REBAR: Low-Variance, Unbiased Gradient Estimates for Discrete Latent Variable Models.” **Advances in Neural Information Processing Systems** 30.

Turing, Alan. 1950. “Computing Machinery and Intelligence.” **Mind** 59 (236): 433–60. https://doi.org/10.1093/mind/lix.236.433.

Uhlenbeck, George E., and Leonard S. Ornstein. 1930. “On the Theory of the Brownian Motion.” **Physical Review** 36 (5): 823–41.

Uria, Benigno, Iain Murray, and Hugo Larochelle. 2014. “A Deep and Tractable Density Estimator.” **Proceedings of the 31st International Conference on Machine Learning**, Proceedings of machine learning research, vol. 32: 467–75.

Van Hasselt, Hado, Arthur Guez, and David Silver. 2016. “Deep Reinforcement Learning with Double Q-Learning.” **Proceedings of the AAAI Conference on Artificial Intelligence** 30.

Vapnik, V., and A. Chervonenkis. 1971. “On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities.” **Theory Probab. Appl.** 16 (2): 264–81. https://doi.org/10.1137/1116025.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. 2017. “Attention Is All You Need.” **Advances in Neural Information Processing Systems**, 5998–6008. https://doi.org/10.5555/3295222.3295349.

Vaswani, Sharan, Francis Bach, and Mark Schmidt. 2019. “Fast and Faster Convergence of SGD for over-Parameterized Models and an Accelerated Perceptron.” **Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics**, Proceedings of machine learning research, vol. 89: 1195–204.

Veit, Andreas, Michael J Wilber, and Serge Belongie. 2016. “Residual Networks Behave Like Ensembles of Relatively Shallow Networks.” **Advances in Neural Information Processing Systems**. https://arxiv.org/abs/1605.06431.

Vershynin, Roman. 2018. **High-Dimensional Probability: An Introduction with Applications in Data Science**. Cambridge University Press.

Vincent, Pascal. 2011. “A Connection Between Score Matching and Denoising Autoencoders.” **Neural Computation** 23 (7): 1661–74.

Voita, Elena, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. “Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned.” **Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics**. https://arxiv.org/abs/1905.09418.

Volkov, Vasily. 2016. “Understanding Latency Hiding on GPUs.” PhD thesis, University of California, Berkeley.

Vovk, Vladimir, Alexander Gammerman, and Glenn Shafer. 2022. **Algorithmic Learning in a Random World**. 2nd ed. Springer. https://doi.org/10.1007/978-3-031-06649-8.

Vries, Harm de, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron Courville. 2017. “Modulating Early Visual Processing by Language.” **Advances in Neural Information Processing Systems** 30. https://arxiv.org/abs/1707.00683.

Vyas, Nikhil, Depen Morwani, Rosie Zhao, et al. 2024. “SOAP: Improving and Stabilizing

Shampoo Using Adam." **arXiv Preprint arXiv:2409.11321**.

Waibel, Alex, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. 1989. "Phoneme Recognition Using Time-Delay Neural Networks." **IEEE Transactions on Acoustics, Speech, and Signal Processing** 37 (3): 328–39. https://doi.org/10.1109/29.21701.

Waleffe, Roger, Wonmin Byeon, Duncan Riach, et al. 2024. "An Empirical Study of Mamba-Based Language Models." **arXiv Preprint arXiv:2406.07887**.

Wallace, Bram, Meihua Dang, Rafael Rafailov, et al. 2024. "Diffusion Model Alignment Using Direct Preference Optimization." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 8228–38. https://arxiv.org/abs/2311.12908.

Wallace, Bram, Akash Gokul, and Nikhil Naik. 2023. "EDICT: Exact Diffusion Inversion via Coupled Transformations." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**.

Wallace, Chris S. 1990. "Classification by Minimum-Message-Length Inference." **Advances in Computing and Information – ICCI '90**, Lecture notes in computer science, vol. 468: 72–81.

Wang, Feng, and Huaping Liu. 2021. "Understanding the Behaviour of Contrastive Loss." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 2495–504.

Wang, Guanghan, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. 2025. "Remasking Discrete Diffusion Models with Inference-Time Scaling." **Advances in Neural Information Processing Systems**.

Wang, Hongyu, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. 2024. "DeepNet: Scaling Transformers to 1,000 Layers." **IEEE Transactions on Pattern Analysis and Machine Intelligence** 46: 6761–74. https://arxiv.org/abs/2203.00555.

Wang, Jie, Tao Ji, Yuanbin Wu, et al. 2024. "Length Generalization of Causal Transformers Without Position Encoding." **Findings of the Association for Computational Linguistics: ACL 2024**, 14024–40. https://doi.org/10.18653/v1/2024.findings-acl.834.

Wang, Junxiong, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. 2024. "The Mamba in the Llama: Distilling and Accelerating Hybrid Models." **Advances in Neural Information Processing Systems**.

Wang, Ke Alexander, Jiaxin Shi, and Emily B. Fox. 2025. "Test-Time Regression: A Unifying Framework for Designing Sequence Models with Associative Memory." **arXiv Preprint arXiv:2501.12352**.

Wang, Kevin, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. "Interpretability in the Wild: A Circuit for Indirect Object Identification in GPT-2 Small." **International Conference on Learning Representations**. https://arxiv.org/abs/2211.00593.

Wang, Lean, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. 2024. “Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts.” **arXiv Preprint arXiv:2408.15664**.

Wang, Sinong, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. “Linformer: Self-Attention with Linear Complexity.” **arXiv Preprint arXiv:2006.04768**.

Wang, Xi, and Laurence Aitchison. 2025. “How to Set AdamW’s Weight Decay as You Scale Model and Dataset Size.” **International Conference on Machine Learning**. https://arxiv.org/abs/2405.13698.

Wang, Zhendong, Jonathan J. Hunt, and Mingyuan Zhou. 2023. “Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning.” **International Conference on Learning Representations**. https://arxiv.org/abs/2208.06193.

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. “Dueling Network Architectures for Deep Reinforcement Learning.” **International Conference on Machine Learning**, 1995–2003.

Warner, Benjamin, Antoine Chaffin, Benjamin Clavié, et al. 2024. “Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference.” **arXiv Preprint arXiv:2412.13663**.

Wasserman, Larry. 2013. **All of Statistics: A Concise Course in Statistical Inference**. Springer. https://link.springer.com/book/10.1007/978-0-387-21736-9.

Wasserstein, Ronald L, and Nicole A Lazar. 2016. “The ASA Statement on p-Values: Context, Process, and Purpose.” **The American Statistician** 70 (2): 129–33. https://doi.org/10.1080/00031305.2016.1154108.

Watkins, Christopher J C H. 1989. “Learning from Delayed Rewards.” PhD thesis, King’s College, University of Cambridge. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

Watkins, Christopher JCH, and Peter Dayan. 1992. “Q-Learning.” **Machine Learning** 8 (3–4): 279–92. https://doi.org/10.1007/bf00992698.

Watson, Geoffrey S. 1964. “Smooth Regression Analysis.” **Sankhyā: The Indian Journal of Statistics, Series A**, 359–72. https://doi.org/10.1007/bf02868765.

Weed, Jonathan, and Francis Bach. 2019. “Sharp Asymptotic and Finite-Sample Rates of Convergence of Empirical Measures in Wasserstein Distance.” **Bernoulli** 25 (4A): 2620–48.

Weinberger, Kilian Q., and Gerald Tesauro. 2007. “Metric Learning for Kernel Regression.” **Proceedings of the 11th International Conference on Artificial Intelligence and Statistics**, Proceedings of machine learning research, vol. 2: 612–19.

Welch, Lloyd R. 1974. “Lower Bounds on the Maximum Cross Correlation of Signals.” **IEEE Transactions on Information Theory** 20 (3): 397–99. https://doi.org/10.1109/TIT.1974.1055219.

Welford, B. P. 1962. “Note on a Method for Calculating Corrected Sums of Squares and Products.” **Technometrics** 4 (3): 419–20.

Welling, Max, and Yee W Teh. 2011. “Bayesian Learning via Stochastic Gradient Langevin Dynamics.” **Proceedings of the 28th International Conference on Machine Learning (ICML-11)**, 681–88. https://dl.acm.org/doi/10.5555/3104482.3104568.

Wen, Kaiyue, David Hall, Tengyu Ma, and Percy Liang. 2025. “Fantastic Pretraining Optimizers and Where to Find Them.” **arXiv Preprint arXiv:2509.02046**.

Wen, Kaiyue, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. 2024. “Understanding Warmup-Stable-Decay Learning Rates: A River Valley Loss Landscape Perspective.” **arXiv Preprint arXiv:2410.05192**.

Weng, Lilian. 2021. **What Are Diffusion Models?** Blog post, https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

Wengert, Robert Edwin. 1964. “A Simple Automatic Derivative Evaluation Program.” **Communications of the ACM** 7 (8): 463–64. https://doi.org/10.1145/355588.365726.

Wenliang, Li K., and Heishiro Kanagawa. 2020. “Blindness of Score-Based Methods to Isolated Components and Mixing Proportions.” **arXiv Preprint arXiv:2008.10087**. https://arxiv.org/abs/2008.10087.

Werbos, Paul J. 1990. “Backpropagation Through Time: What It Does and How to Do It.” **Proceedings of the IEEE** 78 (10): 1550–60. https://doi.org/10.1109/5.58337.

White, Halbert. 1982. “Maximum Likelihood Estimation of Misspecified Models.” **Econometrica** 50 (1): 1–25.

White, Tom. 2016. “Sampling Generative Networks.” **arXiv Preprint arXiv:1609.04468**.

Widrow, Bernard, and Marcian E. Hoff. 1960. “Adaptive Switching Circuits.” **IRE WESCON Convention Record** 4: 96–104.

Wiegreffe, Sarah, and Yuval Pinter. 2019. “Attention Is Not Not Explanation.” **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing**, 11–20.

Wiener, Norbert. 1923. “Differential-Space.” **Journal of Mathematics and Physics** 2: 131–74.

Wightman, Ross, Hugo Touvron, and Hervé Jégou. 2021. “ResNet Strikes Back: An Improved Training Procedure in Timm.” **ArXiv:2110.00476**. https://arxiv.org/abs/2110.00476.

Wigner, Eugene P. 1958. “On the Distribution of the Roots of Certain Symmetric Matrices.” **Ann. Math.**, 325–27. https://doi.org/10.2307/1970079.

Williams, Ronald J. 1992. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.” **Machine Learning** 8: 229–56.

Williams, Samuel, Andrew Waterman, and David Patterson. 2009. **Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures**. Lawrence Berkeley National Lab. https://doi.org/10.2172/1407078.

Witten, Ian H., Radford M. Neal, and John G. Cleary. 1987. "Arithmetic Coding for Data Compression." **Communications of the ACM** 30 (6): 520–40.

Wolpert, David H. 1996. "The Lack of a Priori Distinctions Between Learning Algorithms." **Neural Computation** 8 (7): 1341–90.

Wong, Eugene, and Moshe Zakai. 1965. "On the Convergence of Ordinary Integrals to Stochastic Integrals." **The Annals of Mathematical Statistics** 36 (5): 1560–64.

Woo, Sanghyun, Shoubhik Debnath, Ronghang Hu, et al. 2023. "ConvNeXt V2: Co-Designing and Scaling ConvNets with Masked Autoencoders." **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. https://arxiv.org/abs/2301.00808.

Woodruff, David P. 2014. "Sketching as a Tool for Numerical Linear Algebra." **Foundations and Trends in Theoretical Computer Science** 10 (1–2): 1–157.

Wortsman, Mitchell, Gabriel Ilharco, Samir Yitzhak Gadre, et al. 2022. "Model Soups: Averaging Weights of Multiple Fine-Tuned Models Improves Accuracy Without Increasing Inference Time." **International Conference on Machine Learning**.

Wu, C. F. Jeff. 1983. "On the Convergence Properties of the EM Algorithm." **Annals of Statistics** 11 (1): 95–103.

Wu, Chengyue, Hao Zhang, Shuchen Xue, et al. 2026. "Fast-dLLM: Training-Free Acceleration of Diffusion LLM by Enabling KV Cache and Parallel Decoding." **International Conference on Learning Representations**.

Wu Fung, Samy, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. 2022. "JFB: Jacobian-Free Backpropagation for Implicit Networks." **Proceedings of the AAAI Conference on Artificial Intelligence** 36: 6648–56. https://doi.org/10.1609/aaai.v36i6.20619.

Wu, Yonghui, Mike Schuster, Zhifeng Chen, et al. 2016. "Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation." **ArXiv:1609.08144**. https://arxiv.org/abs/1609.08144.

Wu, Yuxin, and Kaiming He. 2018. "Group Normalization." **European Conference on Computer Vision**. https://arxiv.org/abs/1803.08494.

Xiao, Guangxuan, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. "Efficient Streaming Language Models with Attention Sinks." **International Conference on Learning Representations**.

Xiao, Han, Kashif Rasul, and Roland Vollgraf. 2017. "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms." **ArXiv:1708.07747**. https:

//arxiv.org/abs/1708.07747.

Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. "Aggregated Residual Transformations for Deep Neural Networks." **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 1492–500. https://doi.org/10.1109/cvpr.2017.634.

Xie, Shuo, and Zhiyuan Li. 2024. "Implicit Bias of AdamW: ℓ_∞ Norm Constrained Optimization." **arXiv Preprint arXiv:2404.04454**.

Xiong, Ruibin, Yunchang Yang, Di He, et al. 2020. "On Layer Normalization in the Transformer Architecture." **International Conference on Machine Learning**, 10524–33. https://proceedings.mlr.press/v119/xiong20b.html.

Xiong, Wayne, Lingfeng Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. 2018. "The Microsoft 2017 Conversational Speech Recognition System." **2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**, 5934–38. https://doi.org/10.1109/TASLP.2018.2876459.

Xiong, Wenhan, Jingyu Liu, Igor Molybog, et al. 2024. "Effective Long-Context Scaling of Foundation Models." **Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics**, 4643–63. https://doi.org/10.18653/v1/2024.naacl-long.260.

Xu, Yilun, Mingyang Deng, Xiang Cheng, Yonglong Tian, Ziming Liu, and Tommi Jaakkola. 2023. "Restart Sampling for Improving Generative Processes." **Advances in Neural Information Processing Systems** 36.

Xu, Yilun, Shengjia Zhao, Jiaming Song, Russell Stewart, and Stefano Ermon. 2020. "A Theory of Usable Information Under Computational Constraints." **International Conference on Learning Representations**.

Xu, Yuanzhong, HyoukJoong Lee, Dehao Chen, et al. 2021. "GSPMD: General and Scalable Parallelization for ML Computation Graphs." **ArXiv:2105.04663**. https://arxiv.org/abs/2105.04663.

Yamaguchi, Kouichi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. 1990. "A Neural Network for Speaker-Independent Isolated Word Recognition." **First International Conference on Spoken Language Processing**. https://doi.org/10.21437/icslp.1990-282.

Yang, An, Anfeng Li, Baosong Yang, et al. 2025. "Qwen3 Technical Report." **arXiv Preprint arXiv:2505.09388**.

Yang, Charlene, Yunsong Wang, Steven Farrell, Thorsten Kurth, and Samuel Williams. 2020. "Hierarchical Roofline Performance Analysis for Deep Learning Applications." **arXiv Preprint arXiv:2009.05257**.

Yang, Greg, and Edward J. Hu. 2021. "Tensor Programs IV: Feature Learning in Infinite-Width Neural Networks." **International Conference on Machine Learning**, 11727–37.

Yang, Greg, Edward J. Hu, Igor Babuschkin, et al. 2022. “Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer.” **arXiv Preprint arXiv:2203.03466**.

Yang, Greg, James B. Simon, and Jeremy Bernstein. 2023. “A Spectral Condition for Feature Learning.” **arXiv Preprint arXiv:2310.17813**.

Yang, Greg, Dingli Yu, Chen Zhu, and Soufiane Hayou. 2024. “Tensor Programs VI: Feature Learning in Infinite-Depth Neural Networks.” **International Conference on Learning Representations**. https://arxiv.org/abs/2310.02244.

Yang, Songlin, Jan Kautz, and Ali Hatamizadeh. 2025. “Gated Delta Networks: Improving Mamba2 with Delta Rule.” **International Conference on Learning Representations**. https://arxiv.org/abs/2412.06464.

Yang, Songlin, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2024. “Gated Linear Attention Transformers with Hardware-Efficient Training.” **International Conference on Machine Learning**, 56501–23. https://arxiv.org/abs/2312.06635.

Yang, Songlin, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. 2024. “Parallelizing Linear Transformers with the Delta Rule over Sequence Length.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2406.06484.

Ye, Jiacheng, Zhihui Xie, Lin Zheng, et al. 2025. “Dream 7B: Diffusion Large Language Models.” **arXiv Preprint arXiv:2508.15487**. https://arxiv.org/abs/2508.15487.

Ye, Tianzhu, Li Dong, Yuqing Xia, et al. 2025. “Differential Transformer.” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.05258.

Yin, Tianwei, Michaël Gharbi, Taesung Park, et al. 2024. “Improved Distribution Matching Distillation for Fast Image Synthesis.” **Advances in Neural Information Processing Systems** 37.

Yin, Tianwei, Michaël Gharbi, Richard Zhang, et al. 2024. “One-Step Diffusion with Distribution Matching Distillation.” **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**, 6613–23. https://arxiv.org/abs/2311.18828.

Yin, Y. Q., Z. D. Bai, and P. R. Krishnaiah. 1988. “On the Limit of the Largest Eigenvalue of the Large Dimensional Sample Covariance Matrix.” **Probability Theory and Related Fields** 78 (4): 509–21.

You, Yang, Igor Gitman, and Boris Ginsburg. 2017. “Large Batch Training of Convolutional Networks.” **ArXiv:1708.03888**. https://arxiv.org/abs/1708.03888.

Young, John W. 1974. “A First Order Approximation to the Optimum Checkpoint Interval.” **Communications of the ACM** 17 (9): 530–31. https://doi.org/10.1145/361147.361115.

Yu, Fisher, and Vladlen Koltun. 2016. “Multi-Scale Context Aggregation by Dilated Convolutions.” **International Conference on Learning Representations**. https://arxiv.org/abs/1511.07122.

Yu, Gyeong-In, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. “Orca: A Distributed Serving System for Transformer-Based Generative Models.” **16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)**, 521–38. https://www.usenix.org/conference/osdi22/presentation/yu.

Yu, Qiying, Zheng Zhang, Ruofei Zhu, et al. 2025. “DAPO: An Open-Source LLM Reinforcement Learning System at Scale.” **arXiv Preprint arXiv:2503.14476**.

Yu, Sihyun, Sangkyung Kwak, Huiwon Jang, et al. 2025. “Representation Alignment for Generation: Training Diffusion Transformers Is Easier Than You Think.” **International Conference on Learning Representations**. https://arxiv.org/abs/2410.06940.

Yu, Tianhe, Garrett Thomas, Lantao Yu, et al. 2020. “MOPO: Model-Based Offline Policy Optimization.” **Advances in Neural Information Processing Systems** 33.

Yuan, Jingyang, Huazuo Gao, Damai Dai, et al. 2025. “Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention.” **Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**, 23078–97. https://doi.org/10.18653/v1/2025.acl-long.1126.

Yun, Sangdoo, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. 2019. “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features.” **Proceedings of the IEEE/CVF International Conference on Computer Vision**. https://arxiv.org/abs/1905.04899.

Zadrozny, Bianca, and Charles Elkan. 2002. “Transforming Classifier Scores into Accurate Multiclass Probability Estimates.” **Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, 694–99. https://doi.org/10.1145/775047.775151.

Zaheer, Manzil, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. 2018. “Adaptive Methods for Nonconvex Optimization.” **Advances in Neural Information Processing Systems**, 9793–803. https://proceedings.neurips.cc/paper/2018/hash/90365351ccc7437a1309dc64e4db32a3-Abstract.html.

Zhang, Biao, and Rico Sennrich. 2019. “Root Mean Square Layer Normalization.” **Advances in Neural Information Processing Systems** 32.

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. “Understanding Deep Learning (Still) Requires Rethinking Generalization.” **Communications of the ACM** 64 (3): 107–15. https://doi.org/10.1145/3446776.

Zhang, Fred, and Neel Nanda. 2024. “Towards Best Practices of Activation Patching in Language Models: Metrics and Methods.” **International Conference on Learning Representations**. https://arxiv.org/abs/2309.16042.

Zhang, Han, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. 2019. “Self-Attention Generative Adversarial Networks.” **Proceedings of the 36th International Conference on Machine Learning**, 7354–63. https://arxiv.org/abs/1805.08318.

Zhang, Hanlin, Depen Morwani, Nikhil Vyas, et al. 2024. “How Does Critical Batch Size Scale in Pre-Training?” **arXiv Preprint arXiv:2410.21676**.

Zhang, Hongyi, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. 2018. “mixup: Beyond Empirical Risk Minimization.” **International Conference on Learning Representations**. https://arxiv.org/abs/1710.09412.

Zhang, Hongyi, Yann N. Dauphin, and Tengyu Ma. 2019. “Fixup Initialization: Residual Learning Without Normalization.” **International Conference on Learning Representations**. https://arxiv.org/abs/1901.09321.

Zhang, Jingzhao, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2020. “Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity.” **International Conference on Learning Representations**.

Zhang, Jingzhao, Sai Praneeth Karimireddy, Andreas Veit, et al. 2020. “Why Are Adaptive Methods Good for Attention Models?” **Advances in Neural Information Processing Systems** 33.

Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala. 2023. “Adding Conditional Control to Text-to-Image Diffusion Models.” **Proceedings of the IEEE/CVF International Conference on Computer Vision**, 3836–47. https://arxiv.org/abs/2302.05543.

Zhang, Qinsheng, and Yongxin Chen. 2023. “Fast Sampling of Diffusion Models with Exponential Integrator.” **International Conference on Learning Representations**. https://arxiv.org/abs/2204.13902.

Zhang, Richard. 2019. “Making Convolutional Networks Shift-Invariant Again.” **International Conference on Machine Learning**. https://arxiv.org/abs/1904.11486.

Zhang, Susan, Stephen Roller, Naman Goyal, et al. 2022. “OPT: Open Pre-Trained Transformer Language Models.” **ArXiv:2205.01068**. https://arxiv.org/abs/2205.01068.

Zhang, Wei, Jun Tanida, Kazuyoshi Itoh, and Yoshiki Ichioka. 1988. “Shift-Invariant Pattern Recognition Neural Network and Its Optical Architecture.” **Proceedings of Annual Conference of the Japan Society of Applied Physics**.

Zhang, Xiangyu, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices.” **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 6848–56.

Zhang, Yushun, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. 2024. “Why Transformers Need Adam: A Hessian Perspective.” **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2402.16788.

Zhao, Shengyu, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. 2020. “Differentiable Augmentation for Data-Efficient GAN Training.” **Advances in Neural Information Processing Systems** 33.

Zhao, Wenliang, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. 2023. “UniPC: A Unified

Predictor-Corrector Framework for Fast Sampling of Diffusion Models." **Advances in Neural Information Processing Systems** 36. https://arxiv.org/abs/2302.04867.

Zhao, Yanli, Andrew Gu, Rohan Varma, et al. 2023. "PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel." **Proceedings of the VLDB Endowment** 16 (12): 3848–60. https://doi.org/10.14778/3611540.3611569.

Zheng, Boyang, Nanye Ma, Shengbang Tong, and Saining Xie. 2026. "Diffusion Transformers with Representation Autoencoders." **International Conference on Learning Representations**. https://arxiv.org/abs/2510.11690.

Zheng, Chujie, Shixuan Liu, Mingze Li, et al. 2025. "Group Sequence Policy Optimization." **arXiv Preprint arXiv:2507.18071**.

Zheng, Kaiwen, Yongxin Chen, Huayu Chen, et al. 2025. "Direct Discriminative Optimization: Your Likelihood-Based Visual Generative Model Is Secretly a GAN Discriminator." **Proceedings of the 42nd International Conference on Machine Learning**.

Zheng, Kaiwen, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. 2025. "Masked Diffusion Models Are Secretly Time-Agnostic Masked Models and Exploit Inaccurate Categorical Sampling." **International Conference on Learning Representations**.

Zheng, Kaiwen, Cheng Lu, Jianfei Chen, and Jun Zhu. 2023. "Improved Techniques for Maximum Likelihood Estimation for Diffusion ODEs." **International Conference on Machine Learning**. https://arxiv.org/abs/2305.03935.

Zheng, Lianmin, Liangsheng Yin, Zhiqiang Xie, et al. 2024. "SGLang: Efficient Execution of Structured Language Model Programs." **Advances in Neural Information Processing Systems** 37. https://arxiv.org/abs/2312.07104.

Zhu, Jun-Yan, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks." **Proceedings of the IEEE International Conference on Computer Vision**, 2223–32. https://doi.org/10.1109/iccv.2017.244.

Zhu, Lianghui, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. 2024. "Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model." **International Conference on Machine Learning**. https://arxiv.org/abs/2401.09417.

Ziebart, Brian D, Andrew Maas, J Andrew Bagnell, and Anind K Dey. 2008. "Maximum Entropy Inverse Reinforcement Learning." **Proceedings of the AAAI Conference on Artificial Intelligence**, 1433–38.

Zinkevich, Martin. 2003. "Online Convex Programming and Generalized Infinitesimal Gradient Ascent." **Proceedings of the 20th International Conference on Machine Learning**, 928–36.

Ziv, Jacob, and Abraham Lempel. 1977. "A Universal Algorithm for Sequential Data Com-

pression." **IEEE Transactions on Information Theory** 23 (3): 337–43. https://doi.org/10.1109/TIT.1977.1055714.

Ziv, Jacob, and Abraham Lempel. 1978. "Compression of Individual Sequences via Variable-Rate Coding." **IEEE Transactions on Information Theory** 24 (5): 530–36. https://doi.org/10.1109/TIT.1978.1055934.

Zoph, Barret, and Quoc V Le. 2016. "Neural Architecture Search with Reinforcement Learning." **ArXiv:1611.01578**. https://arxiv.org/abs/1611.01578.