深度学习现象导论

作者:

许志钦　　张耀宇

（笔画数大小排序）

参与学生:

王志伟　白志威　张众望

杭良慨　周章辰　赵佳杰　姚俊杰

（笔画数大小排序）

联系方式:

Github: `https://github.com/xuzhiqin1990/understanding_dl`

Email: xuzhiqin@sjtu.edu.cn, zhyy.sjtu@sjtu.edu.cn

2025 年 5 月 15 日

目录

1 深度学习介绍 **14**

1.1 数据拟合 16
1.2 神经网络简介 19
1.2.1 单个神经元如何感知信息 19
1.2.2 单层神经网络 21
1.2.3 多层神经网络 23
1.3 常用的损失函数 25
1.3.1 均方误差损失 25
1.3.2 绝对误差损失 26
1.3.3 交叉熵 26
1.4 损失景观 27
1.5 优化方法 29
1.5.1 梯度的计算——反向传播 29
1.5.2 梯度下降法 30
1.5.3 带随机的优化方法 32
1.5.4 带动量的梯度下降方法 33
1.5.5 自适应优化方法 34
1.6 参数的初始化 36
1.7 没有免费的午餐 37
1.8 深度学习的理解 38
1.8.1 深度学习的基本要素 38
1.8.2 深度学习理论 40
1.8.3 神经网络的泛化之谜与隐式偏好 40
1.8.4 研究手段：现象驱动的理论研究 41

1.9 习题 44

2 数据与神经网络结构 45

2.1 全连接网络 48
2.2 残差神经网络 50
2.3 卷积神经网络 52
2.3.1 图像数据集的特征 52
2.3.2 初级视皮层的图像处理结构 58
2.3.3 卷积神经网络 59
2.4 循环神经网络 70
2.4.1 循环神经网络基本单元 72
2.4.2 Encoder-Decoder 架构的循环神经网络 73
2.4.3 使用 BPTT 算法训练循环神经网络 73
2.4.4 长短程记忆网络 74
2.5 Transformer 77
2.5.1 Transformer 的基本原理 79
2.5.2 分词与特殊字符 80
2.5.3 Embedding 82
2.5.4 注意力层 83
2.5.5 前馈神经网络层 86
2.5.6 输出层 87
2.5.7 Transformer 做推断的详细流程 87
2.6 生成模型 89
2.6.1 自编码器 90
2.6.2 变分自编码器 91
2.7 习题 93

3 维数灾难 95

3.1 高维空间的特点 97
3.1.1 高维空间中数据的稀疏性 97
3.1.2 体积集中在表面的特性 99
3.1.3 距离的集中效应与正交性 101
3.1.4 高斯环带效应 104
3.1.5 随机投影降维 106

3.1.6 数据的线性可分性 . . . 107
3.2 维数灾难的例子 . . . 108
3.2.1 高维数值积分 . . . 108
3.2.2 高维偏微分方程 . . . 109
3.2.3 高维函数逼近 . . . 110
3.3 克服维数灾难的方法 . . . 111
3.3.1 蒙特卡洛方法 . . . 111
3.3.2 神经网络方法* . . . 113
3.4 习题 . . . 117

4 频率原则 **118**
4.1 频率原则的一维实验 . . . 120
4.1.1 神经网络的“光滑”偏好 . . . 120
4.1.2 频率和傅里叶变换 . . . 123
4.1.3 频率原则 . . . 124
4.2 从频率原则理解神经网络 . . . 126
4.2.1 实验理解频率原则的必要性 . . . 126
4.2.2 Early-stopping 的频率角度理解 . . . 128
4.2.3 神经网络的优势与局限 . . . 130
4.3 频率原则的高维实验* . . . 132
4.3.1 图像中的频率 . . . 132
4.3.2 分类问题中的响应频率 . . . 133
4.3.3 高维傅里叶变换的困难 . . . 134
4.3.4 投影法 . . . 135
4.3.5 滤波法 . . . 136
4.4 习题 . . . 138

5 基于频率原则设计高效神经网络 **140**
5.1 多尺度神经网络结构 . . . 141
5.1.1 结构介绍 . . . 142
5.1.2 基于子空间分解的神经网络 . . . 143
5.2 神经辐射场 . . . 145
5.3 Fourier 特征网络 . . . 146
5.4 更多 . . . 147

6 频率原则的机制分析 148
6.1 频率原则的影响因素 . . . 149
6.1.1 初始化权重大小的影响 . . . 149
6.1.2 不同激活函数的影响 . . . 150
6.1.3 损失函数形式的影响 . . . 151
6.2 频率原则的简单分析 . . . 152
6.3 线性频率原则模型* . . . 154
6.4 线性频率原则的等价变分问题** . . . 156
6.4.1 高频的收敛极限 . . . 160
6.4.2 线性区域的特征分析 . . . 163
6.4.3 一般神经网络的频率原则理论 . . . 164
6.5 习题 . . . 166
6.6 附录: 频率原则理论相关实验 . . . 167
6.6.1 LFP 模型数值实验 . . . 167
6.6.2 高频收敛极限数值实验 . . . 169

7 相图分析 170
7.1 神经网络在不同初始化下的表现 . . . 172
7.2 神经网络的线性与非线性行为 . . . 173
7.2.1 参数的演化分析 . . . 173
7.2.2 线性行为与非线性行为的界定 . . . 174
7.3 线性区域与非线性区域的划分 . . . 176
7.3.1 状态量的定义与动力学相变 . . . 176
7.3.2 实验上相图的获取 . . . 178
7.3.3 临界和凝聚区域 . . . 180
7.4 相图的理论分析 . . . 182
7.4.1 尺度分析方法* . . . 182
7.4.2 两层 ReLU 神经网络的理论结果* . . . 184
7.4.3 三层 ReLU 神经网络的经验相图 . . . 185
7.5 习题 . . . 186

8 凝聚现象 188
8.1 引入 . . . 188
8.2 凝聚现象的实验 . . . 191

8.2.1 凝聚的过程 . . . 191
8.2.2 全连接网络的凝聚现象 . . . 193
8.2.3 卷积神经网络的凝聚现象 . . . 194
8.2.4 残差神经网络的凝聚现象 . . . 195
8.3 凝聚现象的探讨 . . . 197
8.3.1 凝聚现象的定义 . . . 197
8.3.2 对于凝聚现象的理解 . . . 198
8.4 初始凝聚 . . . 199
8.4.1 初始凝聚的实验 . . . 200
8.4.2 初始凝聚的机制探索* . . . 202
8.4.3 初始凝聚的静态分析** . . . 205
8.4.4 初始凝聚的动态分析** . . . 206
8.5 习题 . . . 209

9 损失景观的嵌入原则 211
9.1 宽度相似性与嵌入原则 . . . 214
9.1.1 损失停滞点的结构相似性 . . . 214
9.1.2 理论框架：嵌入原则 . . . 215
9.1.3 嵌入原则和凝聚现象的关系 . . . 217
9.1.4 嵌入原则和频率原则的关系 . . . 217
9.2 嵌入原则的深入分析 . . . 219
9.2.1 损失函数停滞现象的频谱分析 . . . 219
9.2.2 临界点嵌入后 Hessian 矩阵的特征值分析 . . . 221
9.2.3 简化神经网络的规模 . . . 223
9.3 嵌入原则的扩展与推广 . . . 224
9.3.1 临界函数嵌入* . . . 224
9.3.2 一般临界嵌入与临界流形的维数* . . . 225
9.3.3 神经网络深度上的嵌入原则 . . . 226
9.4 习题 . . . 228

10 乐观估计 230
10.1 非线性模型在回归任务中的实验现象 . . . 232
10.1.1 简单的非线性回归模型 . . . 233
10.1.2 矩阵分解模型 . . . 233

10.1.3 神经网络模型 . . . 235
10.1.4 超参数调节在非线性模型中的作用 . . . 237
10.2 量化模型恢复目标函数所需的最小样本量：模型秩 . . . 238
10.2.1 恢复目标函数的参数集合：目标集 . . . 239
10.2.2 乐观初始化下的样本量估计 . . . 240
10.2.3 乐观估计的重要意义 . . . 242
10.3 乐观样本量和实际实验表现的对比 . . . 243
10.3.1 简单的非线性回归模型 . . . 243
10.3.2 矩阵分解模型 . . . 244
10.3.3 神经网络模型 . . . 245
10.4 神经网络架构设计的分析：乐观样本量是否增加 . . . 246
10.5 多层神经网络的乐观样本量* . . . 248
10.6 两层神经网络全局极小点的几何结构** . . . 249
10.6.1 符号和假设* . . . 250
10.6.2 一个两神经元模型的例子* . . . 250
10.6.3 一般的理论结果** . . . 251
10.6.4 两神经元模型的例子解释* . . . 252
10.7 习题 . . . 254

11 Dropout 的隐式正则化 **256**
11.1 Dropout . . . 257
11.2 Dropout 促进神经元凝聚 . . . 258
11.2.1 一维实验 . . . 258
11.2.2 高维实验 . . . 261
11.3 Dropout 及其隐式正则化的显式表达 . . . 262
11.4 正则项对凝聚的影响 . . . 263
11.5 Dropout 对非凝聚极小值的正则化效果 . . . 265
11.6 Dropout 与乐观估计的关系 . . . 266
11.7 SGD 对凝聚没有明显的引导作用 . . . 266
11.8 Dropout 对解的平坦性的影响 . . . 267
11.9 正则化项引导凝聚的理论解释** . . . 269
11.10习题 . . . 271

12 锚函数：研究语言模型的一类简单函数 **273**
12.1 深度学习模型处理语言的简介 . . . 275
12.2 研究基于 Transformer 的语言模型面临的挑战 . . . 277
12.2.1 未知的任务 . . . 277
12.2.2 高昂的计算和内存需求 . . . 278
12.2.3 推理机制的难可解释性 . . . 278
12.3 语言任务的特点 . . . 279
12.4 研究思路 . . . 279
12.5 锚函数与类语言任务 . . . 280
12.5.1 锚函数 . . . 281
12.5.2 类语言任务 . . . 282
12.6 数据划分与模型泛化 . . . 283
12.6.1 训练集和测试集的划分 . . . 283
12.6.2 数据泛化与任务泛化 . . . 286
12.7 实验结果与讨论 . . . 287
12.7.1 恒等学习任务 . . . 287
12.7.2 阅读理解任务 . . . 288
12.7.3 分类任务 . . . 288
12.7.4 复合任务 . . . 289
12.7.5 工作记忆任务 . . . 290
12.7.6 近义词任务 . . . 291
12.7.7 前向-后向背诵任务 . . . 292
12.7.8 统计输出任务 . . . 293
12.7.9 多锚点任务 . . . 294
12.8 恒等学习任务的机制研究* . . . 295
12.8.1 两层模型的简要解释 . . . 295
12.8.2 简化的两层模型的机制 . . . 296
12.8.3 Llama2-7B 中的移位和广播 . . . 297
12.8.4 移位和广播操作的讨论 . . . 298
12.9 实验设置 . . . 298
12.9.1 模型结构 . . . 299
12.9.2 损失函数 . . . 299
12.9.3 超参数设置 . . . 299
12.10Transformer 在各类类语言任务上的训练误差和预测准确率 . . . 300

12.11习题 300

13 参数初始值对模型推理能力的影响 **302**

13.1 定义 304
13.1.1 双锚点复合函数 (Two-anchor composite function) 304
13.1.2 数据生成 (Data Generation) 305
13.1.3 泛化 (Generalization) 306
13.1.4 模型架构和基本实验设置 (Model Architecture and Basic Experimental Setups) 306
13.1.5 初始化和正则化参数 (Initialization and Regularization Parameters) . . 307
13.2 复合函数解的阶段划分 (Phases of Solutions for Composite Functions) 307
13.3 通过注意力掩蔽策略分析不同阶段的机制 (Mechanism Analysis Across Different Phases Through Attention Masking Strategies) 309
13.3.1 通过掩蔽关键 token 进行机制分析 (Mechanism Analysis via Key Token Masking) 311
13.3.2 通过掩蔽第二个锚进行机制分析 (Mechanism Analysis via Second Anchor Masking) 311
13.4 模型复杂度: 相变的关键因素 (Model Complexity: A Key Factor in Phase Transitions) 312
13.4.1 输入权重的凝聚 (Condensation of Input Weights) 312
13.4.2 词嵌入矩阵的结构化组织 (Structured Organization of the Word Embedding Matrix) 313
13.4.3 通过稳定秩的统一复杂度视角 (Unified Complexity Perspective via Stable Rank) 314
13.5 在现实任务上的进一步验证 (Further Verification on Realistic Tasks) 314
13.6 对不同解决方案背后机制的预测 (Prediction of Mechanisms underlying Different Solutions) 318
13.7 习题 319

14 深度神经网络的一些重要现象 **320**

14.1 引入 320
14.2 泛化之谜 322
14.3 缩放定律 (Scaling law) 324
14.3.1 模型参数规模（Model Size） 324

14.3.2 数据量（Dataset Size） . . . 324
14.3.3 计算量（Compute Budget） . . . 324
14.4 稳定边缘现象 (Edge of Stability) . . . 325
14.5 平坦解与尖锐解 . . . 327
14.6 随机梯度下降的噪音结构对深度学习泛化性能的影响 . . . 329
14.6.1 随机梯度下降噪音结构的重要性 . . . 329
14.6.2 随机梯度下降噪音与损失函数平坦度的关系 . . . 330
14.6.3 随机梯度下降隐式正则化的理论分析 . . . 332
14.7 幸运彩票现象 . . . 332
14.8 神经网络中的 Double Descent 现象 . . . 333
14.9 神经网络中的顿悟 (grokking) 现象 . . . 333
14.10神经塌缩现象 . . . 334
14.11Mode connectivity 现象 . . . 335
14.12大型语言模型的上下文内学习 . . . 336
14.13大语言模型中的思维链 (Chain of Thought) . . . 337

15 神经网络求解微分方程 340

15.1 引言 . . . 340
15.2 优势与不足 . . . 343
15.2.1 优势 . . . 343
15.2.2 不足 . . . 344
15.3 参数化解的方法 . . . 345
15.3.1 最小二乘法 . . . 345
15.3.2 变分方法 . . . 346
15.3.3 弱解求解法 . . . 347
15.4 参数化算子的方法 . . . 348
15.5 传统算法和神经网络对于低频的不同偏好 . . . 348
15.5.1 多尺度神经网络解 PDE . . . 350
15.5.2 小结 . . . 351
15.6 高阶内容* . . . 351
15.6.1 传统方法对高频问题快速收敛的原因 . . . 351
15.7 习题 . . . 353

16 实验手册：频率原则的验证 **354**
16.1 神经网络基本概念 354
16.2 实验背景 355
16.3 实验目标 356
16.4 理论基础 356
16.4.1 离散傅里叶变换（DFT） 356
16.4.2 频率原则 358
16.5 实验步骤 358
16.5.1 数据生成与预处理 358
16.5.2 构建模型 359
16.5.3 模型训练及测试过程 359
16.5.4 相对误差的计算 360
16.5.5 实验结果分析与可视化 361
16.6 任务 363

17 实验手册：参数凝聚现象的验证 **365**
17.1 实验简介 365
17.2 实验目标 365
17.3 理论基础 366
17.4 实验原理 366
17.4.1 神经网络结构 366
17.4.2 参数初始化 367
17.4.3 激活函数 368
17.5 实验任务 369
17.5.1 任务一：不同初始化下神经网络参数演化情况 369
17.5.2 任务二：高维函数的初始参数凝聚实验 370
17.6 课后任务和思考 372

18 实验手册：乐观估计的验证 **373**
18.1 实验背景：过参数化下的泛化 373
18.2 实验目标 373
18.3 理论基础 374
18.4 实验步骤 375
18.4.1 理论计算阶段 375

18.4.2 实验验证阶段 . . . 375
18.5 简单的回归模型实验 . . . 376
18.5.1 理论计算 . . . 376
18.5.2 实验验证 . . . 377
18.6 矩阵分解模型实验 . . . 379
18.6.1 理论计算 . . . 379
18.6.2 实验验证 . . . 381
18.7 神经网络模型实验 . . . 384
18.7.1 理论计算 . . . 384
18.7.2 实验验证 . . . 384
18.8 实验总结 . . . 386
18.9 作业 . . . 387

19 实验手册：语言模型实现多步推理的机制探究 388
19.1 实验背景 . . . 388
19.2 实验目标 . . . 388
19.3 实验准备 . . . 389
19.3.1 多步推理数据集 . . . 389
19.3.2 Transformer 模型 . . . 389
19.3.3 数据集的划分 . . . 390
19.4 实验核心代码 . . . 390
19.4.1 数据生成 . . . 390
19.4.2 初始化模型、损失函数与优化器 . . . 391
19.5 实验结果 . . . 392
19.5.1 数据生成与模型训练 . . . 392
19.5.2 模型信息流可视化 . . . 392
19.6 作业 . . . 395

20 参数初始值对推理能力影响实验手册 396
20.1 实验目的 . . . 396
20.2 任务描述 . . . 396
20.3 实验设定 . . . 397
20.3.1 双锚点复合函数 . . . 397
20.3.2 数据生成 . . . 398

20.3.3 锚对的映射类型 399
20.3.4 泛化 399
20.4 实验步骤 399
20.4.1 初始化尺度设定 399
20.4.2 训练 Transformer 模型 399
20.4.3 不同初始化模型准确率测试 401

Chapter 1

深度学习介绍

深度学习, 这个听起来有些 “高大上” 的名词, 正在悄然地改变我们的生活。不信? 让我们看看身边的例子。假如你是一位手机视频应用的重度用户, 那么你一定对视频中人脸特效、动态贴纸的神奇效果印象深刻。无论是变身呆萌的兔子, 还是画上复杂的妆容, 这些都得益于深度学习在计算机视觉领域取得的突破性进展。通过对海量人脸数据的学习, 深度学习模型可以精准地识别人脸特征, 实现实时的人脸检测、跟踪和特效合成。

如果你经常使用智能音箱并与其对话, 你会发现它们的语音识别和自然语言理解能力越来越强。它能够准确地将你的语音指令转化为文本, 并根据语义给出合适的反馈, 这背后也凝聚着深度学习的力量。深度学习通过学习大量的语音和文本数据, 构建声学模型和语言模型, 让机器 “懂” 得人类的语言。

你有没有被手机相册中自动生成的影集、视频集锦所惊喜到? 这些看似智能的操作, 其实是深度学习在多媒体内容理解中的应用。通过对图像、视频内容的语义进行分析, 深度学习算法能自动为照片打上标签, 识别出人物、场景、物体等, 进而智能生成相关的影集。

围棋爱好者也许听说过 AlphaGo 战胜人类顶尖棋手的事迹。这标志着深度学习在博弈领域的里程碑式突破。通过深度强化学习算法, AlphaGo 从海量棋谱数据中自主学习, 掌握了高超的棋艺, 展现出非凡的决策能力, 开创了人工智能在棋类游戏上的新纪元。

这些只是深度学习应用的冰山一角。在医疗、金融、安防、自动驾驶等众多领域, 深度学习正展现出强大的能力, 推动着人工智能飞速发展。那么, 深度学习究竟是什么? 它为何能取得如此惊艳的成果? 通过本书的学习, 我们将一起揭开它的神秘面纱。

在过去的十年里, 以深度学习为代表的人工智能算法的发展如同一场革命, 极大地提升了机器的智能水平。从谷歌的 AlphaGo 在围棋比赛中战胜世界冠军李世石, 到 DeepMind 的 AlphaFold 在长期困扰生物学领域的蛋白质折叠问题上取得了重大突破, 再到 OpenAI 的通用

大语言模型 GPT4、视频生成模型 Sora 的问世，这些成就不仅是技术的突破，更是人类对机器智能潜力重新认识的开始。这些令人瞩目的进展，得益于深度学习技术的强大推动力。

深度学习，这一受人脑结构启发的算法，已成为当今人工智能研究和应用的核心。其本质在于神经网络能够通过拟合海量数据来学习和提取其中复杂的模式和特征。这种方法的强大之处在于其能够自适应地从数据中学习到解决问题所需的关键信息，而无需人工进行特征提取和选择，这提供了解决先前难以攻克问题的新途径。

随着深度学习的快速发展和应用，我们见证了从视觉识别、语言理解到游戏策略等多个领域的惊人成就。然而，深度学习并非一蹴而就的技术，它的成功建立在对神经网络框架深入的理解和不断的探索上。

科普篇

什么是数据拟合？

数据拟合是一种在观测数据中寻找规律的方法。想象你有一堆散点数据, 数据拟合就像是在这些点之间画一条最佳的线或曲线。这条线或曲线应该尽可能地靠近所有的数据点, 以便能够最好地描述这些数据的整体趋势。这个过程就像是在解一个拼图, 我们试图找到一个模型或函数, 能够最好地匹配和解释我们看到的数据。这个模型不仅能描述已有的数据, 还能用来预测新的、未见过的数据。在实际应用中, 数据拟合被广泛用于各种领域, 从简单的线性回归到复杂的机器学习算法, 都是在尝试从数据中提取有意义的模式和关系。

什么是神经网络？它与数据拟合之间有什么联系？

神经网络是一种受人脑结构启发的数学模型，用于拟合复杂的数据关系。它由多层相互连接的”神经元”组成, 每一层都将信息传递到下一层。

神经网络的核心在于它能够通过大量数据的”训练”来自动学习复杂的模式。每个神经元都有自己的权重和偏置, 这些参数在训练过程中不断调整, 使得整个网络能够更好地拟合输入数据和预期输出之间的关系。简单的神经网络可能只有输入层、隐藏层和输出层, 而更复杂的网络可能包含多个隐藏层, 甚至是不同的结构，每一层都专注于提取数据的不同特征。这种层次结构使得神经网络能够处理从简单到非常复杂的问题, 如图像识别、语音处理和自然语言处理等。

神经网络与数据拟合有着密切的联系。实际上，神经网络可以被看作是一种高度灵活和强大的数据拟合工具。当我们使用神经网络进行数据拟合时，网络的目标是学习一个复杂的函数，该函数能够最好地描述输入数据和输出结果之间的关系。

神经网络有哪些值得关注的部分？

在本章中我们列出以下四个神经网络从数据中学习模式的过程中非常重要的部分：

损失函数 损失函数用于度量模型预测和真实值的差异。

损失景观 损失景观描述了模型的误差随模型参数改变的全貌，是优化过程中的地形图。

优化方法 神经网络在复杂的损失景观中找到最优解的方法。

参数初始化 参数初始化则关乎如何在损失景观中选取合适的初始点来启动训练过程，以帮助模型提高训练速度以及找到更优的解，它与模型的泛化能力有很强的关联。

神经网络是万能的么？

既然神经网络可以完成很多复杂的任务，那么随便给一个数据拟合的问题，神经网络都可以很好的完成任务么？答案是否定的。

没有免费午餐定理 该定理的核心思想是在所有可能的问题分布上，所有算法的平均性能是相同的。这一定理说明神经网络并不是万能的，无法在所有任务上达到最优效果。

在这一章中，我们简要的讲解深度学习的一些基础知识，希望激发读者对深度学习的好奇心和理解深度学习的热情。深度学习虽有其直观的魅力，但要真正建立理解，还是需要下一番功夫观察实验现象，掌握理论基础。本书的后续章节将帮助大家更深入地理解深度学习的底层现象和机制。

1.1 数据拟合

在我们生活的世界里，随处可见各种各样的数据。从日常生活到科学研究，我们时刻都在接触和分析各种数据。例如，我们可以通过观察一个城市不同时间的温度变化数据，来研究该城市的气候特点；又比如说，一位医生会收集大量病人的症状和体征数据，借此来判断疾病的种类和严重程度。这些例子都说明了数据对我们认识世界、解决问题有着重要的意义。

然而，单纯地收集数据并不能直接让我们洞悉数据背后隐藏的规律。我们需要运用一定的数学工具和计算方法，来挖掘数据所蕴含的有价值信息，也就是我们通常所说的**数据拟合 (data fitting)**。数据拟合旨在从数据中学到一个数学模型，使其能很好地逼近产生数据的客观规律。我们将这个规律的函数表达称之为**目标函数 (target function)**。举例来说，对于图

像分类问题，其一个参考性的目标函数可以看作由我们大脑中的生物神经网络给出。例如，在构建如 ImageNet 这样的图像分类数据集时，在收集大量的图像数据之外，需要动用大量的人力对它们进行标注，从而形成一个以人脑为参考的数据集。对于一千类别的 ImageNet 数据集，当前最好的深度学习算法已经达到了惊人的约 90% 的测试准确率。值得注意的是，这还不能说明学到的神经网络分类函数已经足够接近人脑分类函数。我们经常观察到，一个肉眼不可察觉的扰动可以对神经网络分类的结果带来巨大的改变。比如，在一个自然图像上泛化好的神经网络对于有噪音干扰的图像输入，并不能很好的完成预测。寻求更好的深度学习算法，使其更好地逼近人脑分类函数，特别是对于有噪音干扰的图像输入，仍然是深度学习算法研究的一个重要问题。

形式上，我们将数据记为 $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$，其中 $\mathcal{X} \subset \mathbb{R}^d$ 被称为样本集（sample set），$\mathcal{Y} \subset \mathbb{R}^{d_o}$ 被称为标签集（label set）。其中 d 为数据维度，d_o 是输出维数，n 是样本的数量。为了表述的方便，我们假设输出维数 $d_o = 1$。目标函数通常被记作 $f^* : \mathcal{X} \to \mathcal{Y}$。在本书中，我们默认数据从目标函数中采出，即 $y_i = f^*(\boldsymbol{x}_i)$，其中 $i = 1, \cdots, n$。在这里，$\mathcal{X}$ 表示输入空间，$\mathcal{Y}$ 表示输出空间，而 $\boldsymbol{x}_i$ 和 y_i 分别代表第 i 个观测的输入和输出。我们使用的机器学习模型记作 $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$，其中 $\boldsymbol{\theta}$ 为模型的参数。当 $\boldsymbol{\theta}$ 变化时，$\{f_{\boldsymbol{\theta}}\}_{\boldsymbol{\theta}}$ 就形成了一个函数空间。当给定一组数据的时候，我们希望从这个函数空间中找到一个最佳函数来逼近这组数据背后的目标函数。为了实现这一点，我们通常会定义一个**损失函数 (loss function)** 来衡量模型预测值和真实值之间的差距，然后通过优化算法调整 $\boldsymbol{\theta}$，以最小化损失函数。这个过程就是我们通常说的模型训练。通过这种方法，我们希望最终得到的模型 $f_{\boldsymbol{\theta}}$ 能够在整个输入空间，特别是在未见输入上，与目标函数 f^* 足够接近。

注 1. 线性拟合是机器学习中一种基础的算法，其模型为 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}$，其中参数 $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{b})$。

注 2. 深度学习是当前最重要的机器学习算法，它使用神经网络模型拟合数据。一个简单的两层全连接神经网络模型可以表示为 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^m a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j)$，其中参数 $\boldsymbol{\theta} = \{(a_j, \boldsymbol{w}_j, b_j)\}_{j=1}^m$，$\sigma$ 是一个非线性函数，常被称为激活函数。

为了刻画模型的预测效果，我们需要选择合理的损失函数来度量模型与真实函数的距离。损失函数通常记为 $\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$，它度量预测标签与真实标签之间的差距。一个直观且使用较为广泛的损失函数为 L^2 损失函数，或称为均方差（Mean Squared Error, MSE），其表达式为：

$$\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), f^*(\boldsymbol{x})) = (f_{\boldsymbol{\theta}}(\boldsymbol{x}) - f^*(\boldsymbol{x}))^2.$$

通常情况下，目标函数并不是已知的，只有训练数据（或称训练集）是可以获得的。模型在训练集上逼近的好坏由训练集在损失函数上的误差，即训练误差 (training error，或叫经验

风险，empirical risk) 来度量，其定义为:

$$R_S = \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i). \tag{1.1}$$

实际中，我们通过最小化训练误差来获得拟合模型，即

$$\min_{\boldsymbol{\theta}} R_S.$$

这就是广为人知的经验风险最小化问题 (Empirical Risk Minimization，ERM)。

对于一个一般的机器学习方法，比如神经网络方法，评估拟合得到的函数与目标函数之间的差距对于评价算法的好坏至关重要。理论上，对于输入服从分布 $\mathcal{D}$ 的数据，这种差距通常由拟合函数的群体风险 (population risk) 或者泛化误差 (generalization error) 度量

$$R_{\mathcal{D}} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), f^*(\boldsymbol{x})),$$

其中 $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}$ 表示关于输入数据的分布求期望。

注 3. 值得注意的是，如果输入分布 $\mathcal{D}$ 集中在一个低维流形上，低泛化误差并不表明拟合函数在全空间上接近目标函数。一个简单的例子是数据分布在一个二维空间的单位圆 $(x_1^2+x_2^2=1)$ 他是一个嵌入二维空间的一维流形，那么在这些数据上泛化误差比较小并不意味着在整个二维空间上都比较拟合的比较好。比如天气预报中，虽然模型可能在历史数据上显示出很低的泛化误差，表明其能准确预测大多数日常天气，但这并不意味着它能在极端气候条件下 (如罕见的暴风雪或极端高温) 准确预测，因为这些情况在数据中很少出现，模型对这部分的拟合能力可能并不足。又比如图像分类问题，尽管我们利用深度学习已经在无噪音干扰的自然图像分布上取得了较低泛化误差，而自然图像分布在一个特定的流形上，但是就整个图像空间而言，特别是在那些经过特定噪声干扰的自然图像输入上，这些神经网络分类函数仍与目标函数差异巨大。

在实践中，由于目标函数一般无法获得，我们无法精确计算以上期望。通常的做法是通过采集另外一组测试数据 $\{(\boldsymbol{x}_j^{test}, y_j^{test})\}_{i=1}^{n'}$ 并且计算拟合函数在这组数据上的平均损失来估算上述泛化误差。这一测试数据上的损失值被称为测试误差（test loss)：

$$R_{test} = \frac{1}{n'}\sum_{j=1}^{n'} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_j^{test}), f^*(\boldsymbol{x}_j^{test})).$$

通常我们把泛化误差和训练误差的差值定义为泛化差距 (generalization gap)，即

$$R_{gap} = R_{\mathcal{D}} - R_S.$$

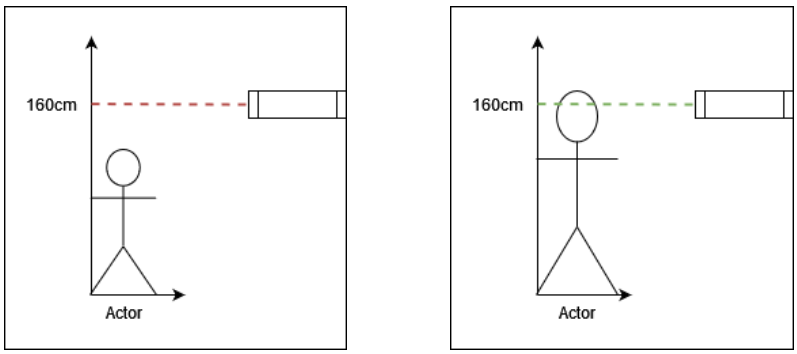
在实际训练中，我们可以通过增加训练样本量来减小泛化差距。这样，当模型通过训练取得了较小的训练误差 R_S 时，其泛化误差 R_D 也会比较小，即拟合函数与目标函数在分布 $\mathcal{D}$ 上足够接近。

为方便机器学习理论的学习与发展，我们倡议在表示机器学习概念时使用统一的规范化的符号，有关机器学习中更多常用符号的规范化描述可以参考附录。

1.2 神经网络简介

1.2.1 单个神经元如何感知信息

感知信息的基本条件是对不同的输入能够区分。为了区分不同的信息，首先要能够从信号中提取有意义的内容。举个例子，假若我们需要征兵，身高超过 160cm 才能满足初试的条件。于是我们设计一个身高信息的提取器。



(a) 未达到 160cm　　(b) 达到 160cm

图 1.1: 身高提取器

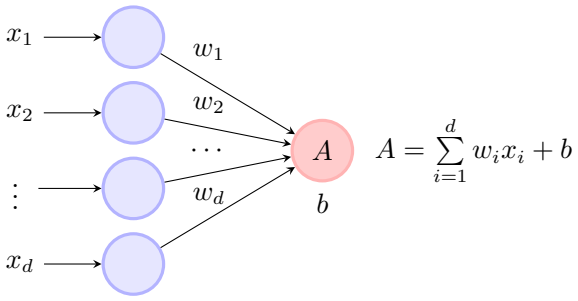
如图 1.1，我们在 160cm 高的位置放一个激光器和接收器。它不断地发出信号，若应征者的身高达到 160cm，则信号会被反射回来，接收器在接收到信号以后，会显示合格，否则不合格。注意到，尽管身高的完整信息是一个具体数字，但我们实际上不需要完整的信息，而是只需要知道是否大于 160cm，只需要输出 0 或者 1，一个字节的信息。因此这套装备的作用可以描述为如下的阈值函数

$$\gamma(x) = \begin{cases} 1 & x \geq T, \\ 0 & x < T. \end{cases} \tag{1.2}$$

一个神经元所执行的基本功能也通常简化成类似的函数，也就是，当外界的信息的强度大

于某个阈值 T 后，神经元会输出一个脉冲信号，即 1，否则为 0。这里的关键是信息怎么被传递给神经元。

我们来举一个例子。假设 A 有两个朋友，B 和 C。这两个朋友每天会给 A 关于股票的信息，该信息是介于 0 和 1 之间的数，表示他们认为股票上升的概率。他们给的信息分别记为 b 和 c。即使 B 和 C 没有给任何信息，A 自己也有一个判断值 a_0。如果 B 是一个经济学家，C 是一个普通人，A 会认为 B 更可信，于是 A 给 B 和 C 各分配了一个权重 λ_1 和 λ_2，其中 $\lambda_1 > \lambda_2 > 0$。最终，A 接收的信息为 $a = \lambda_1 b + \lambda_2 c + a_0$。我们可以把这类传递信息的方式推广到一般的形式。假设有 d 个信息源，$x_1, \cdots, x_d$，神经元对各个信息的权重为 $w_1, \cdots, w_d$，神经元本身的偏置信息为 b，如下图所示



最终神经元得到的总输入为

$$w_1 x_1 + \cdots + w_d x_d + b, \tag{1.3}$$

$$= (w_1, \cdots, w_d) \cdot (x_1, \cdots, x_d)^T + b, \tag{1.4}$$

$$= \boldsymbol{w}^T \boldsymbol{x} + b. \tag{1.5}$$

$(\boldsymbol{w}, b)$ 是神经元的参数，体现神经元的性质。我们再从投影的角度来理解一下神经元处理信息的方式，$\boldsymbol{w}^T \boldsymbol{x}$ 是两个向量的内积，可以理解为信息 $\boldsymbol{x}$ 在神经元特征 $\boldsymbol{w}$ 上的投影。

为了进一步提高神经网络的非线性拟合能力，我们在输出上嵌套一个非线性函数 $\sigma(\cdot)$，于是单个神经元的输出为 $\sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$，这一非线性函数通常被称为激活函数。

注 4. *激活函数是神经网络中不可或缺的一部分，理论上只要激活函数取非多项式形式的非线性函数，就可以使得神经网络具有近似任意阶连续函数的能力。如图1.2所示，常见的激活函数有以下几种：*

- *ReLU*: $\mathrm{ReLU}(x) = max(x, 0)$
- *Tanh:* $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- *Sigmoid:* $\mathrm{Sigmoid}(x) = \frac{1}{1+e^{-x}}$

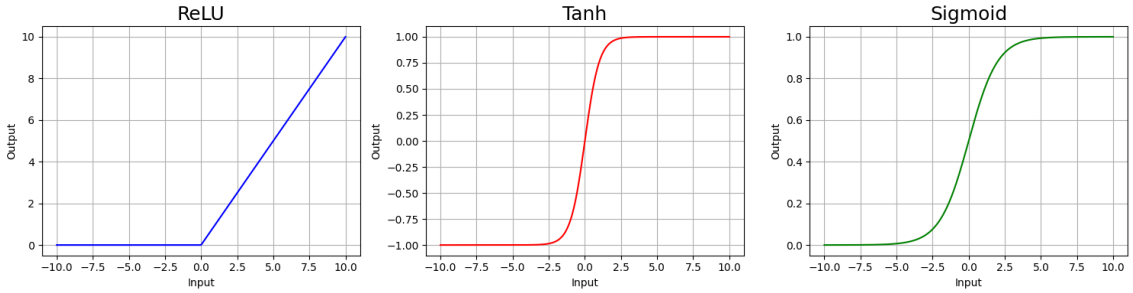
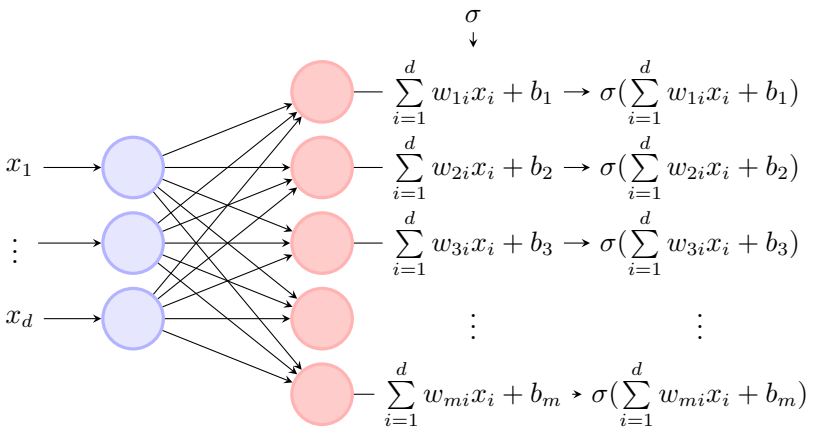


图 1.2: 常见的激活函数

1.2.2 单层神经网络

简单地把多个神经元合在一起，我们就可以得到单层神经网络 (输入不计入层数)。



数学上，我们可以把参数写成一般的形式

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{md} \end{pmatrix}. \tag{1.6}$$

$$\boldsymbol{b} = (b_1, b_2, \cdots, b_m)^T. \tag{1.7}$$

网络的输出为

$$y = \sigma \circ (W\boldsymbol{x} + \boldsymbol{b}), \tag{1.8}$$

其中 $\circ$ 表示对每个元素作用上激活函数。例如，$\sigma \circ ([0.1, 0.3, 0.5]^T) = [\sigma(0.1), \sigma(0.3), \sigma(0.5)]^T$。在神经网络发展的初期，由于计算量不足，主要是单层神经网络起到重要作用。

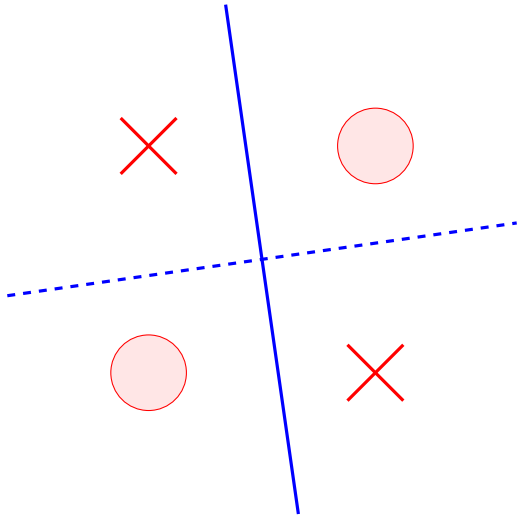


图 1.3: XOR 问题示例。圆圈处的函数值为 1，叉号处的函数值为 0。可见无论何种线性划分方式（蓝色实线和虚线）均无法将两类数据完全划分。

从纯设计的角度，所有神经元的参数都是人为给定，单个神经元可以完成很多逻辑运算。比如，AND 运算，考虑输入是两维的，$x_1, x_2 \in \{0,1\}^2$，也就是每个维度只取 0 或者 1。取阈值 $T = 2$，神经元的输出

$$\sigma(x) = \begin{cases} 1 & x_1 + x_2 \geq T, \\ 0 & x_1 + x_2 < T. \end{cases} \tag{1.9}$$

只有两个维度同时为 1 时，输出才为 1，否则为 0，此即为 AND 运算。若取 $T = 1$，则该神经元在执行 OR 运算。联系到，现代的计算机的基础运算单元正是逻辑运算，所以尽管尚未考虑神经元参数的学习规则，人们已经意识到神经网络可以完成很多任务。

随着使用的深入，人们发现单层神经元只能完成线性可分的问题 Minsky & Papert (1988)，对于如图 1.3 所示的 XOR 问题（XOR 问题是逻辑运算中的一种，当对二维问题的二进制输入执行时，对组合 $\{0,1\}, \{1,0\}$ 产生输出 0，而对组合 $\{0,0\}, \{1,1\}$ 产生输出 1），由于它不是线性可分的，单层神经网络无法做好，至少需要两层的网络才能做到。而多层网络难以训练的情况在当时也劝退了一大批研究人员，带来了神经网络的第一次低潮。

1.2.3 多层神经网络

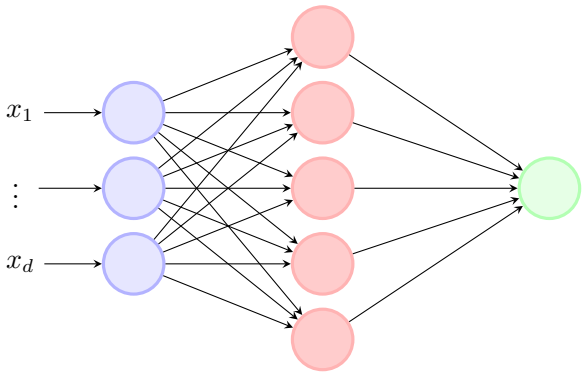


图 1.4: 两层神经网络

多层神经网络是在一层神经网络的基础上堆叠起来的。以两层网络为例，如图1.4，从输入层到第一层的运算和单层网络完全一样，然后以第一层为输入，从第一层到第二层是另一个单层网络。这里的第一层的输出对于使用者来说是看不见的，它是用在内部处理信息的，因此也被称为隐藏层（hidden layer）。

形式上，两层神经网络被表示为：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma \circ (\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j), \tag{1.10}$$

其中 m 为隐藏层的神经元数量，σ 为激活函数（activation function），$\boldsymbol{w}_j$ 为输入权重（input weight），a_j 为输出权重（output weight），b_j 为偏置项（bias）。我们把参数集记为 $\boldsymbol{\theta} = (a_1, \cdots, a_m, \boldsymbol{w}_1, \cdots, \boldsymbol{w}_m, b_1, \cdots, b_m)$.

更一般地，对于一个多层神经网络，假设层数为 L，该神经网络可以被表示为：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]}\sigma \circ (\boldsymbol{W}^{[L-2]}\sigma \circ (\cdots (\boldsymbol{W}^{[1]}\sigma \circ (\boldsymbol{W}^{[0]}\boldsymbol{x} + \boldsymbol{b}^{[0]}) + \boldsymbol{b}^{[1]}) \cdots) + \boldsymbol{b}^{[L-2]}) + \boldsymbol{b}^{[L-1]}, \tag{1.11}$$

其中 $\boldsymbol{W}^{[l]} \in \mathbb{R}^{m_{l+1} \times m_l}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{m_{l+1}}, m_0 = d_{in} = d, m_L = d_o$，$\sigma$ 是一个向量函数，“$\circ$”的意思是对应元素的运算（entry-wise operation），例如对应元素相乘。注意一下在计算网络层数时，输入层不计入（即层数为隐藏层 + 输出层的层数）。我们把参数集记为

$$\boldsymbol{\theta} = (\boldsymbol{W}^{[0]}, \boldsymbol{W}^{[1]}, \cdots, \boldsymbol{W}^{[L-1]}, \boldsymbol{b}^{[0]}, \boldsymbol{b}^{[1]}, \cdots, \boldsymbol{b}^{[L-1]}),$$

把 $\boldsymbol{W}^{[l]}$ 中的元素记为 $\boldsymbol{W}_{ij}^{[l]}$。也可以用递归的方法定义这个网络：

$$f_{\boldsymbol{\theta}}^{[0]}(\boldsymbol{x}) = \boldsymbol{x} \tag{1.12}$$

$$f_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x}) = \sigma \circ (\boldsymbol{W}^{[l-1]} f_{\boldsymbol{\theta}}^{[l-1]}(\boldsymbol{x}) + \boldsymbol{b}^{[l-1]}), \quad 1 \leq l \leq L-1 \tag{1.13}$$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}}^{[L]}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]} f_{\boldsymbol{\theta}}^{[L-1]}(\boldsymbol{x}) + \boldsymbol{b}^{[L-1]}. \tag{1.14}$$

理解神经网络的方式有很多，以下列几种。

线性-非线性变换。当收到一个数据点时，最直接的办法是做一个线性变换再加一个偏置项，也就是仿射变换。但线性函数满足不了实际当中大量非线性的问题。于是，我们给线性变换后的结果套上一个非线性激活函数。可只套一个不够，那就做多次线性变换和套非线性激活函数，再这些套完的结果作为输入，继续做线性-非线性变换。这种形式在生物大脑中也是常见的，比如初级视皮层就发现有很多类似的线性-非线性变换。

投影到高维空间。数据的输入维度是问题给定的。在给定的空间中，拟合或者分类问题可能比较困难，于是，通过非线性激活函数配上不同的参数，将数据映射到另一个高维空间，维数就是隐藏层神经元的个数，然后再做线性拟合。如果一次投影不够，可以做多次投影。这类似于传统方法中的核方法，只是神经网络模型中，核也是一个可学习对象。

函数空间的 Monte-Carlo 采样。对于两层网络，如果我们在公式1.10右边配上一个 $1/m$，这个形式类似于对 m 个函数做加权平均，或者是按某一个概率分布取经验期望，是一个连续期望的离散逼近。

两层及更多层的神经网络具有非常强大的表示能力，我们这里构造一个两层神经网络解 XOR 问题的例子。

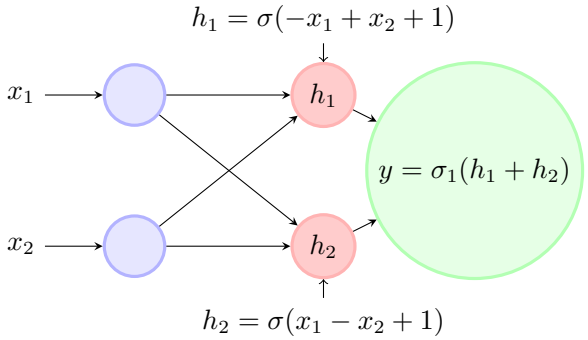


图 1.5: 两层网络解 XOR 问题

如图 1.5所示，取阈值为 1.5 的激活函数 σ，当 $x_1 = x_2 = 0$ 或者 $x_1 = x_2 = 1$ 时，$h_1 = h_2 = \sigma(1)$, 而当 $x_1 = 1, x_2 = 0$ 或 $x_1 = 0, x_2 = 1$ 时，$h_1 + h_2 = \sigma(0) + \sigma(2)$，这样就完

成 XOR 问题的分类了。

进一步，在 1980 年代末，一系列的工作证明当激活函数是非线性且非多项式的时候，对任意一个从有限维空间 $\mathcal{X}$ 到有限维空间 $\mathcal{Y}$ 的连续函数，当神经网络的隐藏层足够宽时，两层神经网络可以以任意精度逼近该函数，这也就是万有逼近定理（universal approximation theorem）(Cybenko 1989, Hornik et al. 1989, Leshno et al. 1993)。

万有逼近定理展现了神经网络具有很强的逼近能力，给神经网络的使用提供了一个重要的理论支撑。但同时，我们也应该注意到万有逼近定理所构造的理论解在实际训练过程中可能并不能达到。因此，一个神经网络有能力逼近只是一个好模型的必要条件，是否能真正拟合好一个给定函数还需要看训练过程的很多因素。这个理论也与实际使用的神经网络有很大的区别，比如尽管两层网络的表达能力已经很强，那为什么实际使用中一般用的是更多层的网络？神经元的数目会不会多到我们实际训练无法承受？定理证明中构造的解是否在实际训练中能被找到？我们需要用什么样的训练方法，需要多少数据？很多类似的问题是万有逼近定理无法回答的。

事实上，万有逼近定理并不能区分神经网络模型的拟合和多项式的拟合。多项式函数在连续函数空间也是稠密的，也就是说多项式函数同样能够以任意精度逼近连续函数，这就是著名但没那么出圈的 Weierstrass 逼近定理。在人工智能这个领域，“命名法”是一个非常常用的方法，对一个新的研究成果，取一个好的名字非常重要。如果万有逼近定理换成一个人名的定理，可能就不容易出圈。当然，“命名法”在每个领域都很重要。

1.3 常用的损失函数

神经网络训练用到的损失函数根据任务不同而往往不同。一般来说，设计符合目标且容易训练的损失函数是一个机器学习任务中最重要的子任务之一。对于监督学习的任务，最简单的和最常用的损失函数是样本标签和神经网络输出之间的均方差，但对于类似对抗生成网络、解微分方程和强化学习等没有显式标签的问题，如何设计损失函数并不直接。我们将在后面的介绍中针对具体的问题讨论损失函数的设计。本节中，我们主要介绍均方误差损失、绝对误差损失和交叉熵。

1.3.1 均方误差损失

均方误差 (Mean Squared Error，MSE) 损失是回归问题中最常用的一种损失函数，也被称为 L^2 损失。考虑数据集 $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$，以及模型 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, 其中 $\boldsymbol{\theta}$ 是可学习的参数。MSE 可以表达成：

$$R_{MSE}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\|f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i\|_2^2. \tag{1.15}$$

1.3.2 绝对误差损失

绝对误差 (Mean Absolute Error，MAE) 损失是回归问题中另一种常用的损失函数，也被称为 L^1 损失。MAE 可以表达成：

$$R_{MAE}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\|f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i\|_1. \tag{1.16}$$

1.3.3 交叉熵

在分类问题中，交叉熵 (cross-entropy) 比 MSE 或者 MAE 都更加常用，主要的原因是在分类问题中，模型输出的解释往往基于概率分布。交叉熵用于衡量两个概率分布之间的差异，其中一个表示真实标签的分布，另一个表示预测输出的分布。对于二分类问题，交叉熵损失函数的形式通常表示为：

$$H(y,p) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i\log(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + (1-y_i)\log(1-f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\right], \tag{1.17}$$

其中，y_i 是样本 i 的真实标签（0 或 1），$f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ 是模型预测样本 i 为类别 1 的概率。对于多分类问题，交叉熵损失可以扩展为：

$$H(y,p) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)_c), \tag{1.18}$$

其中，C 是类别总数，$y_{i,c}$ 是一个指示变量，如果样本 i 属于类别 c 则为 1，否则为 0；$f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)_c$ 是模型预测样本 i 属于类别 c 的概率。

交叉熵损失函数可以通过最大似然估计（Maximum Likelihood Estimation，MLE）推导得到。最大似然估计是一种估计统计模型参数的方法，它通过最大化观测数据的似然函数来找到模型参数的最佳估计。在分类问题中，假设我们有一个概率模型来预测每个类别的概率，我们希望选择参数使得观测到的数据分布出现的概率最大。

对于二分类问题，假设 p_i 是模型预测第 i 个样本为正类的概率，$1-p_i$ 是预测为负类的概率。如果真实标签 $y_i = 1$，观测数据的似然是 p_i；如果 $y_i = 0$，观测数据的似然是 $1-p_i$。因此，给定所有样本，整个数据集的似然函数为：

$$L = \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{(1-y_i)}. \tag{1.19}$$

取对数似然函数，得到：

$$\log L = \sum_{i=1}^{N}\left[y_i\log(p_i) + (1-y_i)\log(1-p_i)\right]. \tag{1.20}$$

最大化对数似然等价于最小化其负值，因此，最大化对数似然函数与最小化交叉熵损失函数是等价的。这就是交叉熵损失函数由最大似然估计推导得到的过程。在多分类问题中，推导过程类似，只不过需要考虑每个样本对每个类别的概率。

softmax。在使用交叉熵损失函数做多分类问题时，要求模型的输出向量的每个值处于 0 到 1 之间，且总和为 1。为实现这一点，通常会在模型输出后加一个额外的 softmax 操作。假设模型输出为 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = (z_1, z_2, \cdots, z_C)$，softmax 操作使模型输出变成：

$$\text{softmax}(f_\theta(x)) = \left(\frac{e^{z_1}}{\sum_{i=1}^{C} e^{z_i}}, \cdots, \frac{e^{z_C}}{\sum_{i=1}^{C} e^{z_i}} \right). \tag{1.21}$$

值得一提的是，尽管交叉熵在分类问题中得到了广泛应用并且有好的概率解释，这只是驱动模型输出和数据标签一致的一种损失函数形式。此前介绍的均方误差和绝对误差损失也都可以实现同样的功能。此外，尽管我们常常将经过 softmax 变换后的模型输出解释为模型对不同类别概率分布的推断，但是这一理解并没有坚实的统计基础，即我们不能依据模型预测某一类别的概率几乎为 1 而推断输入有极高概率属于该类别。因为在实际训练后，输出值只表示模型基于训练数据对输入样本的一个预测而非实际后验概率。

困惑度（Perplexity）。在自然语言中，常常会用困惑度（Perplexity）来表征学习的效果，其定义为

$$PPL = 2^{H(y,p)}. \tag{1.22}$$

这是信息论中的一个概念，其可以用来衡量一个随机变量的不确定性，也可以用来衡量模型训练的好坏程度。通常情况下，一个随机变量的 Perplexity 数值越高，代表其不确定性也越高；一个模型推理时的 Perplexity 数值越高，代表模型表现越差，反之亦然。

直观上，我们可以理解成是猜多次可以猜中目标。举一个例子，假设现在字典中有十个单词，目标输出是第一个单词，当前的学习状态时，前五个单词的输出概率均为 1/5，其它输出为 0。可以计算到交叉熵为 $\log_2 5$, 困惑度为 5。也就是要猜测当前输出具体是哪值，要从五个词里挑，模型不确定是哪一个。

1.4 损失景观

神经网络的经验风险函数（损失函数）$R_S(\boldsymbol{\theta})$，特别是损失值随参数 $\boldsymbol{\theta}$ 变化的几何形态，常被称为损失景观 (Loss Landscape)。我们用 “景观” 这样的称呼来强调理解 $R_S(\boldsymbol{\theta})$ 这样的对象与理解地貌景观的一致性：（1）山峰（局部极大值点）和山谷（局部极小值点）很重要，我们经常需要寻找合适的路径来到达特定的山峰或山谷，比如最低谷（全局最低点）。（2）凸性很重要，凸的景观可以保证任意点都能直接 “看” 到最低谷，因此找到最低谷（优化）很容易，而非凸的景观通常就要困难很多；（3）对于非凸的景观来说，鞍点很重要。鞍点是一个理想的

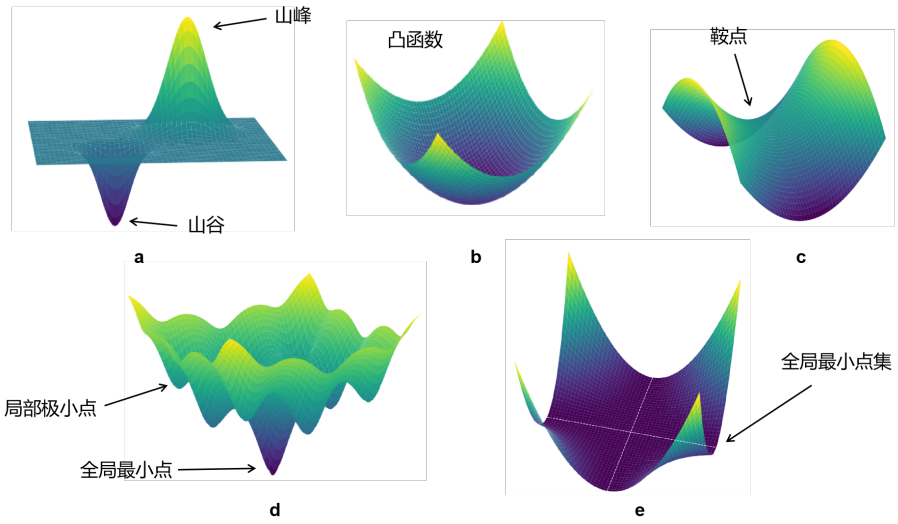


图 1.6: 损失景观中存在的一些结构

“歇脚点”，常常也是从一个山谷到另一个山谷的最低翻越点。借助这种直观的类比，我们可以将神经网络的训练过程理解成一个探险家在一个蜿蜒曲折的复杂地貌景观上寻找最低谷的过程。图 1.6展示了损失景观中存在的一些常见结构。

对于深度学习算法，研究其损失景观的结构有特殊的意义：（1）尽管损失景观高度非凸，我们发现规模足够大的神经网络通常不难收敛到全局最小，阻碍训练的坏的局部极小点并不常见。举一个形象的例子来说明坏的局部极小点，一个盲人下山，在某一个半山腰上用拐杖探路，发现四周比自己的脚下都高，他会仅仅停在半山腰，无法最终走到山脚。但实际训练中的神经网络常常最后都可以到山脚，这提示我们神经网络的损失景观一定有特殊的结构支撑其良好的优化性质。（2）神经网络的损失景观结构不仅影响优化性能，更影响泛化性能。由于实践中我们经常使用过参数化的神经网络，其损失景观的最低谷通常不是一个点，而是一个无穷点组成的高维流形。这个流形上不同的全局极小点泛化性各不相同，大多泛化误差较大。神奇的是，神经网络的损失景观似乎有某种特殊的结构，使得在常见初始化下，各类优化算法都能被引导至其中泛化较好的点。

以上两点告诉我们，不论是研究神经网络的优化还是泛化性能，损失景观都是绕不开的关键研究对象。可以想见，在未来的深度学习理论中，损失景观理论一定占有一个重要的地位。

1.5 优化方法

在前面的内容中，我们讨论了如何使用损失函数来衡量深度学习模型的性能，也看到了深度学习的损失景观呈现出高维非凸的复杂结构。因此，在这样的损失景观中寻找最优解是一个充满挑战的任务。为了高效地优化模型参数，深度学习广泛采用基于梯度的优化方法来降低损失函数值。基于梯度的优化方法的核心思想是利用目标函数的梯度信息来指导优化过程。

在各种优化方法中, 梯度下降是一种最基本的方法。举一个简单的例子来理解。假设你是一位登山爱好者, 你的目标是到达山谷的最低点。但是你身处山顶, 四周都是迷雾, 看不清下山的路。你该如何下山呢? 一个直观的方法是: 环顾四周, 找到最陡峭的下坡方向, 然后朝这个方向走一步。接着, 你再次环顾四周, 找到最陡峭的下坡方向, 再走一步。重复这个过程, 直到你到达山谷的最低点。这个过程, 就是梯度下降的基本思想。在深度学习中, 高维复杂的损失景观难以想象, 我们无法直接“看到”最陡峭的下降方向。这时, 梯度的概念就派上了用场。梯度是一个向量, 它指向损失函数上升最快的方向。那么, 我们只需要沿着梯度的反方向, 就可以朝着损失函数下降最快的方向“走一步”。这就是梯度下降的核心思路。

优化方法是深度学习的重要组成部分。有效的优化算法可以加速模型的训练过程，提高学习效率，同时还有助于找到更优的解。当前深度学习主要使用基于梯度的一阶优化方法，其核心是损失函数关于参数的梯度的计算。在本节中，我们将简单地介绍深度学习中计算梯度所遵循的反向传播模式，梯度下降法与带随机的梯度下降法，以及为提高收敛速度或模型性能而提出的一些梯度下降法的变种。

1.5.1 梯度的计算——反向传播

在深度神经网络中，梯度的计算并非是一件简单的事情。神经网络通常由多个层次的非线性变换组成，每一层的参数更新都依赖于后面层的梯度信息。在深度学习领域中，梯度的计算也被称为反向传播（Back Propagation，BP），因为它遵循着从网络输出到输入的反向传播过程，递归地计算每一层的梯度。这里再一次体现“命名法”的重要性，虽然本质上这就是计算导数时的链式法则，但换一个名字以后，它就变得更有丰满和深度，听起来也更像一个新的东西。

下面我们用两层神经网络 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j)$ 为例。我们同样考虑均方损失函数 (为了求导后的表达式更简洁，我们将标准的 MSE 损失添上系数 1/2):

$$R_S = \frac{1}{2n} \sum_{i=1}^{n} (y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^2. \tag{1.23}$$

对于最外层参数 a_j，当我们使用梯度下降更新参数时，有：

$$\frac{\partial R_S}{\partial a_j} = \frac{1}{n}\sum_{i=1}^{n}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)\sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}_i + b_j). \tag{1.24}$$

对于 $\boldsymbol{w}_j$，由于它不是最外层的参数，我们需要多次用到链式求导法则：

$$\begin{aligned}
\frac{\partial R_S}{\partial \boldsymbol{w}_j} &= \frac{1}{n}\sum_{i=1}^{n}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}{\partial \boldsymbol{w}_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)\frac{\partial(\sum_{t=1}^{m} a_t\sigma(\boldsymbol{w}_t x_i + b_t))}{\partial \boldsymbol{w}_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)a_j\frac{\partial(\sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}_i + b_j))}{\partial \boldsymbol{w}_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)a_j\frac{\partial\sigma(x)}{\partial x}|_{x=\boldsymbol{w}_j\cdot\boldsymbol{x}_i+b_j}\frac{\partial(\boldsymbol{w}_j x_i)}{\partial \boldsymbol{w}_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i)a_j\frac{\partial\sigma(x)}{\partial x}|_{x=\boldsymbol{w}_j\cdot\boldsymbol{x}_i+b_j}\boldsymbol{x}_i.
\end{aligned}$$

因为链式法则，梯度下降的计算顺序是 $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \to a_j \to \sigma \to \boldsymbol{x}_i$，而信息流 (information flow) 的顺序是 $\boldsymbol{x}_i \to \sigma \to a_j \to f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$，这两者的顺序相反，因此称其为反向传播。

注 5. 由于链式法则，在求损失函数关于参数的导数时，会得到一串表达式相乘的情况。对于很深的网络，可能会出现这些相乘后的数很大导致训练不稳定（梯度爆炸），或者数很小导致无法训练（梯度消失）。我们在后边讨论神经网络结构的章节中会提到这些问题的解决方案。

注 6. 许多人把向后传播算法的发现归结于 *D. Rumelhart*、*G. Hinton* 和 *R. Wiliams* 三人在 *1986* 年的文章。但事实上，这篇 *1986* 年的文章明确指出它不是最先发现的，*David Parker* 和 *Yann LeCun* 等人在之前就是发现这个算法。*1986* 年这篇文章优秀的地方在于它把算法描述得很清楚，并且通过复杂的问题论述这个算法的有效性。这里不得不说这个领域中另一个非常重要的研究方法，“考古法”。众所周知，深度学习领域数度起落，每一波浪潮都产生了很多很好的想法，但因为各种原因并没有发扬光大，在算力、数据、训练算法等合适的时候，这些被埋藏的算法可能再度影响学术界，比如用神经网络解微分方程，二十世纪九十年代就已经产生了很多很好的想法，直到 *2016* 年被重新发现以后，才在各个领域广为传播。

1.5.2 梯度下降法

有了梯度信息，最基本的优化算法就是梯度下降法（Gradient Descent，GD）。梯度下降法以负梯度方向为搜索方向，以固定的步长（即学习率）来更新参数。其具体算法如下：

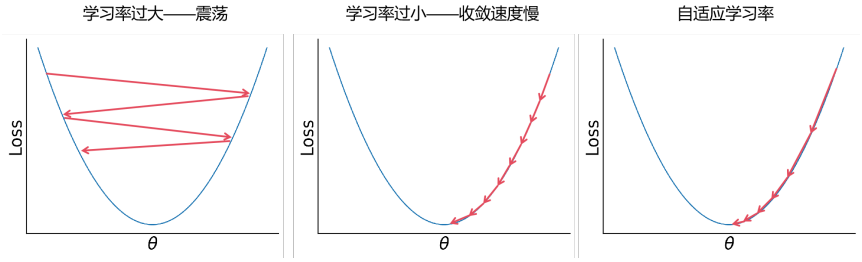


图 1.7: 不同学习率对训练带来的影响

$$g_t = \nabla_\theta R_S(\theta_{t-1}), \tag{1.25}$$

$$\theta_t = \theta_{t-1} - \eta g_t. \tag{1.26}$$

其中，η 是学习率（learning rate，或叫步长，step size）。

我们用泰勒展开 (Taylor expansion) 来解释为什么梯度下降法在步长较小时能够保证损失函数下降。

在 θ_{t-1} 处，损失函数 $R_S(\theta)$ 的一阶泰勒展开式为:

$$R_S(\theta_t) \approx R_S(\theta_{t-1}) + (\theta_t - \theta_{t-1})^T \nabla R_S(\theta_{t-1}).$$

将梯度下降的更新公式代入，得到:

$$R_S(\theta_t) \approx R_S(\theta_{t-1}) + (-\eta \nabla R_S(\theta_{t-1}))^T \nabla R_S(\theta_{t-1}) \tag{1.27}$$

$$= R_S(\theta_{t-1}) - \eta \|\nabla R_S(\theta_{t-1})\|^2 \tag{1.28}$$

从上式可以看出:

$$R_S(\theta_t) \approx R_S(\theta_{t-1}) - \eta \|\nabla R_S(\theta_{t-1})\|^2 \leq R_S(\theta_{t-1}). \tag{1.29}$$

这意味着，在梯度不为零的情况下，只要学习率充分小到可以使用一阶 Taylor 展开逼近损失函数，那么损失函数在更新后会严格减小。这就解释了为什么梯度下降法能够降低损失函数。

在训练神经网络的过程中，学习率是一个至关重要的超参数，它控制着我们沿着梯度下降方向前进的步长大小，只有选取合适的学习率才能顺利达到比较好的训练效果。对梯度下降算法为例，如果学习率过大，则会如图1.7左图所示，神经网络参数就会震荡，难以收敛，给神

经网络训练带来不稳定性；而当学习率过小时，则会如图1.7中图所示，神经网络需要很多个 epoch (epoch 是指神经网络的一步训练循环) 才能够达到极小点，这会造成计算资源的浪费。

我们从数学的角度来理解这种不稳定性，以二次函数

$$L = \frac{1}{2}\lambda\theta^2$$

为例，在高维空间中 λ 为损失函数 Hessian 矩阵（高维函数对自变量求二阶导得到的矩阵）的特征值，η 为学习率，根据梯度下降更新参数

$$\theta_{t+1} = \theta_t - \eta\frac{\partial L}{\partial \theta},$$

要保持损失函数单调下降的线性稳定要求

$$|\theta_{t+1}| < |\theta_t|,$$

即

$$|1 - \lambda\eta| < 1,$$

则我们有学习率 η 与 λ 的关系：

$$\eta < \frac{2}{\lambda}. \tag{1.30}$$

这说明只有学习率小于 $\frac{2}{\lambda}$ 时，才能保证神经网络的训练过程在梯度下降的方法下稳定，关于随机梯度下降的稳定性分析，可以参考Wu et al. (2018)。

1.5.3 带随机的优化方法

随着大数据时代的到来，深度学习面临着前所未有的机遇和挑战。海量的训练数据为模型提供了更为丰富的学习素材，有助于提升模型的泛化能力和鲁棒性。然而，另一方面，超大规模数据也对传统的优化方法提出了严峻的考验。当数据量远超过计算和存储资源的承载能力时，梯度下降等经典优化算法就显得力不从心了。

在这样的背景下，随机优化方法应运而生，成为了深度学习优化的一股清流。随机优化方法的核心思想是通过在每次迭代中只使用一小部分数据来更新模型参数。这种随机优化方法大大降低了每次参数更新的内存开销，使得训练过程可以高效地进行。

随机优化方法中最具代表性的算法就是随机梯度下降(Stochastic Gradient Descent, SGD)。随机梯度下降法的原理是随机地从所有样本中选取 $|B|$ 个样本来估计损失函数，即

$$R_B = \frac{1}{n}\sum_{i=i_1}^{i_{|B|}}(f_{\boldsymbol{\theta}}(x_i) - y_i)^2, \tag{1.31}$$

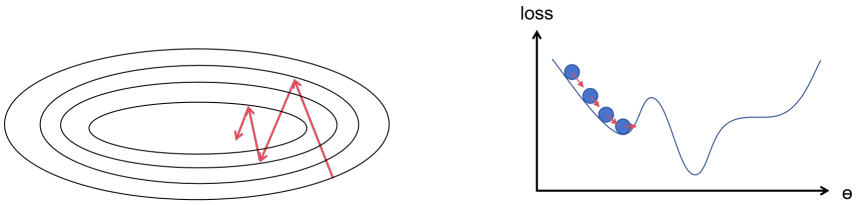


图 1.8: 梯度下降法的震荡行为与带动量的梯度下降法原理

其中 $\{i_1, i_2, \cdots, i_{|B|}\}$ 在每一步可以是随机选取，也可以是提前固定分配好的。随机梯度下降使用 L_B 作为损失函数进行梯度下降，由于没有用到所有的样本，因此计算出来的梯度并不是全局的，所以被称为随机梯度下降法。

实际训练中，通常会把样本分成多份，每份的数据点数量为 $|B|$，每一份称为一个 batch，在每一个 epoch 中，按顺序一步取一个 batch 进行梯度下降，直到所有的 batch 都被使用过一次。完成一个 epoch 后，常见的做法是将数据打乱，以避免模型对数据顺序产生依赖，从而提高模型的泛化能力。然而，在一些特定情况下，如时间序列分析或某些类型的递归神经网络训练，数据的顺序包含了重要的上下文信息，这时可能选择不打乱数据直接进行下一个 epoch，以保持数据的时间连续性和上下文完整性。

随机优化方法的影响是深远的。它不仅改变了深度学习的训练范式，还为其他大规模优化问题提供了新的思路。在机器学习、运筹优化、金融工程等领域，随机梯度类方法正越来越受到重视，成为解决复杂优化问题的利器。尽管 SGD 的引入最初的想法只是为了解决计算量和内存的问题，但后来的实践发现，大多数情况下，SGD 在泛化性上比使用全数据的梯度下降也会更有优势。因此，对 SGD 的理论研究也成为重要且热门的方向。

1.5.4 带动量的梯度下降方法

在前面的小节中，我们介绍了梯度下降法及其随机版本，它们利用了梯度信息来指导优化过程。然而，在某些情况下，单纯的梯度下降可能会表现得不尽如人意。

首先，如图 1.8左侧所示，当目标函数的等高线呈现出狭长的椭圆形时，梯度下降法容易出现“之字形”的震荡行为。这是因为在椭圆的长轴方向上，梯度的分量较小，更新步长有限；而在短轴方向上，梯度的分量较大，更新步长过大，导致来回振荡。这种情况下，优化过程会变得非常缓慢。

其次，当损失景观中存在大量的局部最优或鞍点时，梯度下降法很容易陷入其中，难以逃离。这是因为在局部最优或鞍点附近，梯度的量级会变得非常小，导致优化过程几乎停滞不前。

为了克服这些困难，研究者们引入了动量（momentum）的概念。动量法的核心思想是在每次更新时，不仅考虑当前的梯度方向，还要继承之前更新的方向。如图 1.8右侧所示，红色

箭头为带动量的梯度方向，小球即使在局部最小值时也具有梯度，带动量的梯度下降法使得优化过程就有了一定的“惯性”，能够跨越狭长的山谷和局部最优点，朝着更优的方向前进。

形象地说，动量法就像一个小球在山谷中滚动。当小球下滑时，它不仅受到当前坡度（梯度）的影响，还保持着之前运动的方向和速度。这种“惯性”使得小球能够在陡峭的斜坡上快速下降，并在到达谷底后继续向前滚动，直到新的力量阻止它。

一个最基本的带动量的梯度更新方式如下：

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.32}$$

$$m_t = \mu m_{t-1} + g_t, \tag{1.33}$$

$$\theta_t = \theta_{t-1} - \eta m_t. \tag{1.34}$$

μ 是一个超参数, m_0 可以设为 0。

Nesterov 动量法最早在Polyak (1964) 提出，而深度学习中广泛应用的 Nesterov 加速梯度（Nesterov Accelerated Gradient, NAG）算法 (Nesterov 1983) 是在传统动量法的基础上进行改进，先沿着之前的方向更新一步，再计算梯度并校正方向。这就好像我们爬山的时候可以提前预测如果按照原计划时后一步落脚的情况，从而重新考虑当前状态的决策。这种“向前看”的策略使得 NAG 能够更快地收敛到最优解。

$$g_t = \nabla_\theta R(\theta_{t-1} - \eta\mu m_{t-1}), \tag{1.35}$$

$$m_t = \mu m_{t-1} + g_t, \tag{1.36}$$

$$\theta_t = \theta_{t-1} - \eta m_t. \tag{1.37}$$

1.5.5 自适应优化方法

自适应优化算法是一类在训练机器学习模型过程中能够根据数据特征自动调整学习率的优化算法，与传统优化算法（如梯度下降法）相比，自适应优化算法能够动态调整每个参数的学习率，从而在训练过程中更有效地应对不同参数的梯度变化。这些算法通过对梯度信息进行移动平均或对过去的更新进行积累来计算适应性学习率，常见的自适应优化算法包括 AdaGrad(Duchi et al. 2011)、Adam(Kingma & Ba 2014) 等。自适应优化算法的一个关键优势在于它们能够处理稀疏数据、高维空间以及具有非平稳梯度的任务，提升模型的收敛速度和稳定性。由于其灵活性，自适应优化算法已成为现代深度学习中广泛使用的工具，尤其在训练复杂的神经网络时表现尤为突出。

Adagrad AdaGrad 的关键思想是通过累积过去的梯度平方值为每个参数调整其学习率，以达到自适应修改学习率的目的。这一方法解决了在高维稀疏数据上训练模型时，传统梯度下降

法中固定学习率的问题。它能够自动降低被频繁更新的参数的学习率，而对稀疏更新的参数保持相对较高的学习率，从而更好地处理不平衡数据或特征。其具体形式如下：

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.38}$$

$$n_t = n_{t-1} + g_t^2, \tag{1.39}$$

$$\theta_t = \theta_{t-1} - \eta g_t / \sqrt{\epsilon + n_t}. \tag{1.40}$$

但是这个方法会带来新的问题，也就是在训练末期，由于 n_t 太大，导致学习太慢。ϵ 目的是让分母不为零。

Adadelta 在 Adagrad 中，n_t 的更新是对梯度进行求和累计，会导致训练后期 n_t 太大，学习过慢。因此，Adadelta 对 n_t 的更新采用指数加权移动平均，使得 n_t 增长平滑。

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.41}$$

$$n_t = \nu n_{t-1} + (1-\nu) g_t^2, \tag{1.42}$$

$$\theta_t = \theta_{t-1} - \eta g_t / \sqrt{\epsilon + n_t}. \tag{1.43}$$

ν 是一个超参。**RMSProp** 是 Adadelta 中 $\nu = 0.5$ 的情况。

Adam Adam 方法则是融合了动量方法和自适应学习率方法，进一步提升了优化性能，并在图像识别、语言模型等任务中取得了最优的结果。

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.44}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \tag{1.45}$$

$$n_t = \beta_2 n_{t-1} + (1-\beta_2) g_t^2, \tag{1.46}$$

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^t}, \tag{1.47}$$

$$\hat{n}_t = \frac{n_t}{1-(\beta_2)^t}, \tag{1.48}$$

$$\theta_t = \theta_{t-1} - \eta \hat{m}_t / \sqrt{\epsilon + \hat{n}_t}. \tag{1.49}$$

β_1, β_2, η 为超参数。

注 7. *Reddi et al. (2018)* 对 *Adam* 方法的收敛性建立了理论分析，指出 *Adam* 并不能保证收敛，甚至在一些简单的凸问题中都无法收敛到最优解。*Zhang, Chen, Shi, Sun & Luo (2022)* 发现了之前理论工作的假设和现实应用场景之间的一点小不同之处，过去的收敛性理论是先固定了 β_1 和 β_2，再去解问题。而现实场景中，却是根据某个具体问题来调整 β_1 和 β_2。该论文

假设，当 β_2 足够大，并且 $\beta_1 < \sqrt{\beta_2} < 1$ 时，*Adam* 方法可以收敛到；临界点 *(critical point*，即使得 $\nabla g(\theta) = 0$ 的 θ*)* 的邻域中。

现在还有一些新发展的算法，比如：Adan(Xie et al. 2024)、Sophia(Liu et al. 2023) 等等，用于更好的优化复杂的模型。

1.6 参数的初始化

参数初始化，顾名思义，就是为模型的参数设定初始值的过程。它看似简单，却对模型的训练和收敛有着至关重要的影响。不同的初始化导致不同的训练动力学，也对模型最终的泛化性有影响。当我们采用比较小的初始化时，神经网络在训练的初期就处于鞍点，会产生训练过慢甚至无法训练的情况；当我们采用较大初始化的时候，神经网络又会产生泛化性较差的问题。

一般而言，我们希望初始参数满足以下性质:

- 打破对称性: 初始参数不能都为相同的值，否则网络的每一层每个神经元都会学到相同的特征，丧失了多层结构的意义。
- 保持适度的方差: 初始参数的方差不能过大或过小，否则会导致梯度爆炸或消失，影响训练的稳定性。
- 考虑非线性: 针对不同的非线性激活函数，初始参数应该有对应不同的分布，以确保信息在网络中的有效传播。

我们以全连接网络为例介绍几个常见的初始化方式，m_{in} 和 m_{out} 分别表示输入当前层的参数维度以及输出当前层的宽度，即一个隐藏层对应的参数为 $\boldsymbol{\theta} \in \mathbb{R}^{m_{out} \times m_{in}}$，则有下列一些比较常见的初始化：

Lecun Initialization (LeCun et al. 2012)

- 满足正态分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{N}(0, \frac{1}{m_{in}})$。
- 满足均匀分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{U}(-\sqrt{\frac{3}{m_{in}}}, \sqrt{\frac{3}{m_{in}}})$。

Xavier Initialization (Glorot & Bengio 2010)

- 满足正态分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{N}(0, \frac{2}{m_{in}+m_{out}})$。
- 满足均匀分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{U}(-\sqrt{\frac{6}{m_{in}+m_{out}}}, \sqrt{\frac{6}{m_{in}+m_{out}}})$。

Kaiming Initialization (He et al. 2015)

- 满足正态分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{N}(0, \frac{2}{m_{in}})$。
- 满足均匀分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{U}(-\sqrt{\frac{6}{m_{in}}}, \sqrt{\frac{6}{m_{in}}})$。

注意到以上三种常见的初始化不论是正态分布还是均匀分布的形式，都存在随着网络宽度增加而减小的性质，并且都是以 $1/\sqrt{m}$ 的尺度（高斯分布要考虑标准差）。这主要是因为在设计这些初始化时需要满足计算神经网络的信号 $\boldsymbol{x}$ 在 $l-1$ 和 l 层之间流动时强度（即神经网络每一层输出）在层与层之间传递不会出现爆炸或者消失，

$$\boldsymbol{x}^{[l]} = \boldsymbol{W}^{[l]}\sigma(\boldsymbol{x}^{[l-1]}) + \boldsymbol{b}^{[l]}, \tag{1.50}$$

即如果我们用(1.50)表示信号在两层中流动，则我们需要满足 $\mathrm{Var}(\boldsymbol{x}_j^{[l]}) = \mathrm{Var}(\boldsymbol{x}_i^{[l-1]})$。直观地想，每一个后层神经元都要把前一层所有神经元的输出求和配上一个新的参数，假设前层每个神经元都是独立，且方差都是 $1/m$，高斯分布的求和仍然是高斯分布，且方差是所有高斯分布的方差的和，因此，前层神经元求和的方差为 1，由于配上的参数方差是 $1/m$，因此，这层的每个神经元的方差和前一层一样都是 $1/m$。这就意味着神经网络的参数需随网络宽度增加而减小。同时这些参数在随机的初始化时几乎不会有相同的值。

如果想要了解更多的初始化，可以参考Narkhede et al. (2022)。此外不同的初始化往往会导致神经网络拥有不同的训练动力学，这将在后续的章节中详细说明。

特别需要说明的是，方差是 $1/m$ 量级的很多初始化方法已经被封装到常用的代码框架内部，如 PyTorch, Tensorflow 等。因此，许会深度学习的用户很少会听说需要调节初始化。然而，当前大模型的训练中，调整初始化分布的方差却是一件很重要和常见的事。在后续的章节中，我们还将发现初始化分布的方差会影响大语言模型的记忆和推理能力。

1.7 没有免费的午餐

没有免费的午餐定理（No Free Lunch Theorem）(Wolpert & Macready 1997) 是一个在机器学习领域非常著名的定理，它的核心思想是在所有可能的问题分布上，所有算法的平均性能是相同的。这意味着没有一个算法能够在所有可能的问题上都表现最优。换句话说，一个算法在某些特定问题上的优势，必然以在其他问题上的劣势为代价。具体定理描述如下：

定理 1 (没有免费的午餐)**.** 给定一个有限的数据集合 X 和一个有限的标签集合 Y，从数据到标签的全部映射关系集合记作 Y^X，要在集合 X 上优化从 Y^X 中随机挑选的一个映射函数 $f : X \to Y$，那么没有比盲搜索更好的方法了。

这一定理最初由 David H. Wolpert 和 William G. Macready 于 1997 年提出。他们在研究优化算法的过程中，发现无法设计出一个普适的算法，能够在所有问题上都取得最好的效果。这表明，如果考虑所有可能的问题，每一个算法解决问题的平均性能是一样的。这个定理有多个变体，但它们的基本观点是相同的：不存在绝对意义上的“最佳”算法。对详细的证明感兴趣的读者可以参考 (Wolpert & Macready 1997)。

这个定理强调了选择与特定问题相匹配的算法的重要性。它告诉我们，应该基于问题的具体特性和需求来选择算法，而不是寻找一个万能的解决方案或者脱离数据集特征来讨论算法的性能。以深度学习算法为例，研究表明神经网络通常难以学习和泛化奇偶函数 (parity function) 与傅里叶变换 (Shalev-Shwartz et al. 2017, Nye & Saxe 2018)。这两类任务的目标函数往往比较震荡，而神经网络在学习这种类型的函数比较困难，在后续的章节中，我们也将更深入地介绍神经网络难以学好的一大类函数以及背后的机制。但是，如果我们将视角转移到真实世界的数据集上，神经网络却展现出了非凡的能力。在图像识别、语音识别、自然语言处理等领域，神经网络模型不断刷新着性能记录，甚至在某些任务上超越了人类的表现。造成这种差异的原因，正是不同问题的数据背后目标函数的特征不同。

类似的例子还有很多。例如，朴素贝叶斯算法在文本分类任务上曾长期保持着最优性能的地位，但在许多其他问题上却表现平平。再如，决策树算法在某些结构化数据上非常有效，但在图像、语音等非结构化数据上却难以发挥作用。这些例子都说明，算法的性能评估必须建立在具体问题和数据的基础上。一个算法在某个数据集上的失败，并不意味着它在所有问题上都不可用。反之，一个算法在某些数据上的成功，也不能保证它在其他问题上同样有效。

“没有免费的午餐”定理是机器学习领域的一个基石。它提醒我们，在机器学习的世界里，没有一劳永逸的“万能灵药”，只有因地制宜的“对症下药”。认清这一点，我们才能在使用和发展机器学习算法的道路上行稳致远。

1.8 深度学习的理解

1.8.1 深度学习的基本要素

理解一个机器学习算法本质上需要理解模型、数据、优化三个要素对算法拟合目标函数性能（比如泛化误差等）的影响。对于一些传统的机器学习算法，比如核方法，我们对这些要素的影响有明确且定量的认识。简单来说，核方法与数据的适配程度由目标函数的核范数度量。核范数是一个度量，用来评估在核方法中，函数的复杂度。在机器学习中，核范数可以被视为模型复杂度的一个指标，它基于核函数的特性来衡量。核函数是一种衡量数据点间相似性的函数，而核范数则是这些函数权重的一种度量形式。核范数越小，表明模型越简单，通常意味着在足够多的数据支持下，模型的泛化能力越强。这是因为较小的核范数倾向于对数据的噪声和

异常值具有更好的抗干扰性，从而提高了模型在未见数据上的性能。然而对于深度学习算法，这些要素的组成与对拟合性能的影响要复杂得多，建立理论理解的挑战也大得多。下面，我们首先对于深度学习算法的三要素及其对拟合性能的影响做一些框架性的介绍。

深度学习算法的根本特征在于模型选用神经网络这类特殊的、受大脑结构启发的非线性模型族。许多复杂的任务，比如图像识别、自然语言处理、围棋等，一旦使用合适的神经网络模型，就可以从根本上得到解决。这并不是说围绕特定任务的其他要素，比如任务的建模、数据收集与清洗、优化技巧等，以及工程方法不重要，而是我们认为必须清楚地认识到神经网络模型在整个解决方案中的核心地位，就如同发动机之于飞机的重要性。进一步来看，神经网络模型包括架构、规模、激活函数等要素。其中架构，比如全连接、卷积、残差、LSTM、transformer等，是神经网络模型的关键。比如 OpenAI 的 Radford 在训练语言模型预测 Amazon 用户评论的下一个字符时发现，一经使用 transformer（替代 LSTM），两周的进展就超过了过去两年。架构之外，足够大的规模（包括足够的深度、宽度、参数量）是神经网络模型取得优异性能的重要保障。OpenAI 在Kaplan et al. (2020) 中提出的模型参数量、数据量和模型计算量的幂律关系称为 Scaling Law。关于 Scaling Law 的研究表明，即便在相对较少的训练数据上，模型规模变大一般也不会显著影响性能。这与传统机器学习方法模型规模越大越容易过拟合的基本性质形成了鲜明的反差。这种神奇的现象被称为显著的悖论、神经网络的过参数化谜团、或神经网络的泛化之谜，是深度学习理论研究的关键问题之一。此外，尽管不如以上两个要素那么关键，激活函数对模型的优化和泛化也有重要的调制作用。

在数据方面，深度学习算法的产生使得我们有能力解决一系列此前无法解决的复杂任务，比如围棋、图像识别、自然语言处理等。这些问题背后的目标函数通常非常复杂，没有明确的表达式。不过它们也有一些常见的特性，即人通常也能做好它们。许多人无法做好的事，比如大数的乘法，当前最强的大语言模型也难以做好。目标函数之外，数据量也是数据集的关键要素。Scaling Law 的实验表明，没有足够的训练数据量，许多复杂任务，比如说 In-context Learning，是无法做好的。此外，数据分布也对拟合误差有显著的影响，比如在干净的自然图像数据上训练的神经网络很难在有噪声干扰的图像上取得良好的测试误差。一般来说，在输入空间的低维流形上训练的神经网络通常难以在全空间良好地逼近目标函数。

在优化方面，初始化，即神经网络权重初值的选取，不仅对训练速度影响重大，更会显著影响神经网络的泛化性能。这主要是由于我们常常使用过参数化的神经网络来拟合训练数据。此时损失景观中会有无穷的泛化误差各不相同的全局极小点，在传统机器学习理论中，这种设定会导致模型过拟合，但是在神经网络的实践中这种情况往往不会发生。在常见的初始化下，神经网络通常会收敛到泛化较好的全局极小点，但同时不合理的初始化也很容易导致过拟合。此外，神经网络训练策略也是一个重要要素。合适的训练策略常常能在保证泛化性能的同时提高训练效率，比如在大语言模型的训练中，在大量的实验中可以观察到 AdamW 可以达到最优的结果。此外，dropout、批归一化以及其他优化技巧也可以带来训练效率或泛化能力的提升。

1.8.2 深度学习理论

神经网络的理论经历了数十年的发展，至今仍然不完善。当前的理论尚不能很好地解释数据、模型、优化对拟合性能的影响。这导致我们在使用和发展深度学习算法时，主要依赖经验和大量的试错来取得进步。造成这种情况的原因是多方面的，我们认为主要原因有以下两点：(1) 神经网络及其训练过程在某种意义上可以看作一个高维且高度非线性的复杂系统。尽管可以获取神经网络中每一个神经元的激活和每个权重的大小，神经网络的训练过程可以精确记录，但由于它们之间复杂的非线性相互作用，观察和分析整个神经网络训练和预测过程的统计规律非常困难，它本质上成为一个“黑盒子”。(2) 我们很难分析维数高、结构复杂的真实数据，并且无法描述这些数据背后的目标函数。根据“没有免费的午餐定理”，决定神经网络性能的核心是算法与特定数据的适配性。原则上我们可以通过分析那些神经网络取得优异性能的数据来理解神经网络的“喜好”。然而，这些数据，如图像分类和蛋白质折叠，我们对其结构和背后的目标函数理解极为有限，本质上也是“黑盒子”。

从机器学习的角度来说，打开真实数据和神经网络间任意一个“黑盒子”都是极为重要的研究目标。然而在实际应用中，这两个“黑盒子”常常紧密耦合在一起，这给我们观察和分析其中任何一个“黑盒子”的内在结构和规律增添了极大的困难。面对这种挑战，我们研究的一个基本策略是先利用人造数据拆解神经网络“黑盒子”的规律和偏好，再利用神经网络的规律和偏好解析真实数据“黑盒子”的结构。从另一个角度出发，由于深度神经网络模型对于解决许多复杂问题起到了“发动机”的作用，拆解其本身也具有特别重要的意义。本书的后续章节将深入介绍当前研究在打开神经网络“黑盒子”方面取得的长足进展。我们也期待随着对其研究的逐步深入，我们能开始着手拆解真实数据“黑盒子”。希望通过大家共同的努力，未来我们不再用“炼丹”来形容深度学习算法的设计、使用和改进。

1.8.3 神经网络的泛化之谜与隐式偏好

数据量与模型参数量之间的关系是机器学习中一个关键的考量点。对于线性模型中，模型参数多于数据量会导致过拟合，而少于数据量会导致欠拟合。然而，对于神经网络模型，这种数据量与模型参数量之间的权衡在实践中很少发生。在现代的深度学习实践中，我们发现即便用特别大规模的神经网络来拟合相对简单的训练数据，也能获得良好的泛化性能。我们把这种现象称为神经网络的泛化之谜，是神经网络模型的一个关键特性。这种增加模型的表达能力不会损害泛化性能的新特性，使深度学习在一定程度上跳出了以往数据量与参数量相互制约的窠臼，极大地降低设计神经网络模型的难度。在实际训练中，通常当我们难以训练到低训练误差时，就可以通到增大网络规模来改善训练结果。

从概念上来说，许多研究者将神经网络的泛化之谜归结为神经网络算法具有某种隐式偏好（又称隐式正则化）。之所以说隐式，是因为我们观察到，在过参数化条件下，即便没有在

损失函数中加任何额外的正则化项，神经网络在常见的初始化下也会从无穷的全局极小点中挑选具有特定性质、常常泛化良好的点。仿佛训练过程背后有某种隐藏的偏好（正则化）在起作用。从没有免费的午餐定理来看，理解这种隐式偏好将直接帮助我们搞清楚神经网络与什么样的目标函数适配度高。

本书前半部分的研究重点就聚焦于神经网络的一般性的**隐式偏好（implicit bias，又称归纳偏置，inductive bias）**：神经网络这一特殊的模型通常偏好具有什么性质的函数？后半部分则侧重特定架构神经网络的独特偏好：一个特定架构的神经网络偏好其函数空间中的哪些函数？偏好程度和机制如何？

1.8.4 研究手段：现象驱动的理论研究

近年来，深度学习系统的复杂度不断提升，其内部运作机制也变得日益难以捉摸。以 GPT 系列语言模型为例，其参数规模之大，训练成本之高，令人咋舌。即便发现一个小小的缺陷，重新训练的代价也是天文数字。面对如此高度复杂的系统，我们该如何探寻其内在规律，又该如何指引其未来的发展方向？

科学研究告诉我们，对复杂系统的探索，现象观察与理论分析往往是并驾齐驱、相辅相成的。通常，正是丰富多样的现象推动了理论的进步，成熟的理论又进一步解释更为丰富的现象。

以生物学为例，随着生物进化，生物大脑的结构日趋复杂。为了理解大脑的工作原理，科学家们在现象观察和理论研究上都投入了大量心血。他们一方面通过解剖、显微镜观察、脑成像等手段，获取大脑结构和活动的第一手资料；另一方面，他们又尝试建立数学模型，模拟神经元的信息传递和处理过程。正是在大量现象积累的基础上，神经科学的理论体系才得以逐步构建和完善。Hubel 和 Wiesel 通过系统的电生理实验，发现了视皮层神经元对视觉刺激的选择性响应特性，奠定了视觉信息处理的理论基石。他们的工作不仅揭示了视觉系统的工作原理，也为卷积神经网络的设计提供了重要启发。

纵观科学发展史，现象驱动理论的案例比比皆是。天文学家第谷孜孜不倦地记录天体运动数据，为开普勒发现行星运动定律提供了坚实的观测基础。开普勒通过分析第谷数据，总结出行星运动三大定律。尽管这些定律本质上是对观测现象的经验总结，但它们却启发了牛顿建立经典力学体系，实现了对天体运动的深刻理解和精确预测。

在电磁学领域，法拉第、安培等科学家通过大量实验，发现了电流、磁场间的多种联系。法拉第发现电流能够产生磁场，磁场的变化又能够产生电流，揭示了电磁感应现象。这些实验事实极大地丰富了人们对电磁相互作用的认识。麦克斯韦正是在前人实验的基础上，通过深刻的理论思考，提出了著名的麦克斯韦方程组，实现了电磁理论的大统一。麦克斯韦方程不仅能够解释已有的电磁现象，还预言了电磁波的存在，为无线电通信开辟了道路。

热力学的发展历程同样印证了现象对理论的重要推动作用。科学家们通过大量实验观察，

发现了热机效率等一系列关键现象, 并提炼出相应的经验定律。这些现象和定律构成了热力学理论的基石。卡诺通过分析理想热机循环, 得出了热机效率的上限, 为热力学第二定律的提出奠定了基础。后来, 克劳修斯、开尔文等人在实验事实的基础上, 系统地建立起热力学理论体系, 使之成为物理学的重要分支。

综上所述, 科学发展的一个重要范式, 就是先有现象的观察和积累, 再有理论的提出和发展。现象观察为理论研究指明方向, 提供素材和依据。没有第谷和开普勒的观测数据, 就难以想象牛顿力学的诞生; 没有法拉第和安培的实验发现, 麦克斯韦也难以构建起电磁理论大厦。与此同时, 成熟的理论又能反过来解释已有现象, 并预测新现象。牛顿力学不仅能够解释开普勒定律, 还预言了许多新的天文现象, 如潮汐、彗星运动等; 麦克斯韦方程不仅统一了电磁现象, 还预言了电磁波的存在, 开启了无线电通信的新纪元。可见, 现象积累和理论研究如同车之两轮、鸟之双翼, 共同推动了科学的进步。

相比纯粹的理论研究, 现象驱动的研究模式有独特优势。首先, 它能帮助我们快速锁定复杂系统的关键影响因素。纯理论研究往往需要引入大量假设和前提, 推导过程也常常繁琐费时。而通过巧妙设计实验、控制变量, 我们可以更高效地探索复杂系统的内在规律。这就如同研究黑箱系统, 与其妄加揣测其内部结构, 不如先观察其外部输入输出关系。其次, 现象驱动的研究可以为理论分析提供更广阔的思路和更多灵感。当我们过于专注于特定理论模型时, 往往容易陷入思维定势, 忽视问题的其他方面。而广泛观察各种现象, 则有助于我们跳出固有的理论框架, 发现新的研究方向和突破口。

不过, 并非所有的观察结果都能上升到“现象”的高度。真正的科学现象, 应当是在特定条件下稳定出现的、具有一定规律性的自然事实。因此, 从观察到现象的提炼, 需要严谨缜密的实验设计。我们需要精心控制实验条件, 排除各种干扰因素, 确保实验结果的可重复性; 我们需要系统地记录和分析实验数据, 提炼出关键的统计规律; 我们还需要在不同的实验设置下考察现象的稳定性和普适性。只有经过严格的、系统的实验论证, 一个观察结果才能真正成为指导理论研究的科学现象。

在当前深度学习的研究中, 由于模型和数据都呈现出高度的复杂性, 通过精心设计实验来观察现象, 就显得尤为重要和必要。一方面, 我们对真实世界数据的结构和分布缺乏全面了解, 很难直接从理论上进行分析和预测。但我们可以通过构造特定结构的合成数据, 来考察模型的学习能力和泛化性能。例如, 我们可以设计不同频率特性的目标函数, 生成相应的训练数据, 然后观察模型的拟合情况, 由此来验证模型是否具有频率偏好等特性。另一方面, 尽管目前我们对模型内部的工作机制还缺乏透彻理解, 但我们可以通过系统观察模型在不同任务上的表现, 来推测其处理信息的一般规律。例如, 我们可以比较模型在各种视觉基准测试中的性能, 分析其对不同类型特征的敏感性, 进而对模型的不同架构进行探究。

现象研究和数学理论研究都是基础研究中至关重要的组成部分。通常意义上的（数学）理论研究, 应该同时包含现象总结和数学定理证明, 而不应狭隘地等同于定理证明。近年来, 有观

点认为基础研究发展严重滞后于应用研究, 对应用研究帮助不大。事实上, 这是一种非常错误的看法。恰恰相反, 正是基础研究推动了每一次重要应用的进步。以 Scaling Law 为例, 正是由于观察到这一现象,OpenAI 才坚定地通过扩大模型和数据规模, 最终引领了大语言模型的发展。Scaling Law 这一现象的发现过程, 体现了浓厚的科学精神: 通过观察实验, 总结经验公式, 然后进行预测, 再通过实验验证, 不断重复, 最终得到一系列可靠结论, 并对应用产生深远影响。

再如 AlphaGo, 随着落子数增加, 棋局的可能性呈指数级增长, 通过穷举采样来训练神经网络评估棋局是不可能的。AlphaGo 的成功很大程度上在于解决了高维采样问题, 其核心想法就是观察到并非所有棋局都是等价的, 深度神经网络可以快速辅助判断哪些棋局是没有价值的, 往后就不必再搜索，而高质量的数据又能提升神经网络的精度，这就是蒙特卡洛树搜索结合深度学习的精髓。还有很多类似的例子可以从本书中找到。

之所以会出现上述对深度学习基础研究误导性的观点, 主要是因为人们对基础研究或理论研究缺乏正确认识和合理期待。基础研究并非要告诉你每个参数该如何调节, 而是可以告诉你哪个方向是行不通的, 哪个方向更有希望; 模型擅长什么, 又有何局限。比如本书重点介绍的频率原则, 告诉我们模型擅长解决低频问题, 面对高频问题则力有不逮。在一般网络上调参是困难的, 但通过将高频函数拉伸为低频函数, 调参会更容易。至于如何拉伸, 不同文章提出了不同方法, 尽管各有差异, 但核心思想和达到的功能是一致的, 即让调参更加容易。传统科学中也有类似情况。热力学第一定律指出能量守恒, 第一类永动机不可能, 避免浪费精力。热力学第二定律指出不可能从单一热源吸收能量使之完全转化为有用功而不产生其他影响, 第二类永动机不可能, 避免浪费精力。化学理论指出炼金术把铜变成金是不可能的, 避免浪费精力。把木碳在高温高压下变成钻石虽然可能, 但工艺难度极大, 经过长期发展, 现在人造钻石已经比较普遍。

一个优秀的基础（理论）研究, 可以从以下几个方面来判断:

基本条件:

1. 问题重要: 发现或研究有趣的现象或理论问题。
2. 可分析: 简单到可以分析, 但又复杂到足以对关心的现象或问题提供理解。
3. 可验证: 可以通过实验加以验证。

高级条件:

1. 能够引出更多有意思的现象或理论问题, 帮助我们更深入理解。
2. 有助于指导实际算法的设计。
3. 有助于培养理论或应用人才。

现象观察和理论研究相辅相成, 共同推动了深度学习乃至整个科学的发展。在深度学习日益复杂的今天, 从丰富的现象入手, 系统总结规律, 建立理论框架, 将是探索深度学习奥秘、指

引其未来发展的重要途径。让我们携手并进, 在现象与理论的映照中, 共同开创深度学习的美好明天。

1.9 习题

- 什么是回归问题? 什么是分类问题? 回归问题和分类问题有什么区别?
- 误差可以分成哪几类? 它们各自有什么作用?
- 有哪些常用的激活函数? 为什么不使用多项式激活函数?
- 神经网络是如何学习数据的?
- 一层网络有什么限制?
- 什么是万有逼近定理? 什么样的网络具有万有逼近能力?
- 损失景观的定义是什么? 训练集和测试集的损失景观一样吗?
- 什么是没有免费的午餐定理? 它带给我们什么启示?
- 什么是维数灾难?

Chapter 2

数据与神经网络结构

数据的一种定义是未经加工和整理的信息素材，是信息的原始形式。从生物在地球上诞生的第一天起，生物体便面临在恶劣的、充满不确定性的环境下生存繁衍的艰巨挑战。而收集其生存环境中的各种信息以更好地趋利弊害对于生物体的存活至关重要。比如眼睛这样的感觉器官的出现使得生物体能够有效地收集周围环境的光信息。眼睛就像一台摄像机，可以帮助生物体持续地采集视频数据。然而光有数据还不够，生物体必须能够从这些数据当中提取与生存相关的关键信息才行，比如眼前的东西是什么，能不能吃，有没有危险等等。动物的神经系统就是负责处理数据、提取关键信息的系统。只有生物体从数据中提取到的信息的价值超过生物体感知和处理数据所消耗的能量，神经系统才能在进化的过程中不断地“升级”，形成强大的生物智能。可以这么说，蕴藏着丰富的生存信息的数据是生物智能赖以成长的根本动力。

感官捕捉的各种数据中蕴含丰富的生存信息并不是一件显然的事。比如在海洋深处极其黑暗的环境中，即便光线没有完全消失，当中蕴含的有效信息也是极为稀少的，就如同在极弱光线下拍一张照片，很难从噪点中看出物体的模样。这种情况下，很多生物的眼睛就退化了。没有视觉数据的驱动，其神经系统的智能程度也更难进化。幸运的是，在人类日常生存的环境中，我们的感官捕捉到的视觉、听觉、嗅觉、味觉、触觉等数据的信息丰富。将生存信息从这些数据中有效地提取出来的需求不断驱动人类大脑的进化、智能的提升。可以说，以神经系统为载体的生物智能算法就是为了做好这样的一大类信息提取任务而进化出来的。

那么如何设计机器学习算法来完成这类复杂的从数据中提取信息的任务呢? 在当前这个大脑能力进化的速度远远跟不上数据爆炸（指数增长）的时代，这个问题显得尤为重要。仅仅依靠人的大脑从这些数据中提取信息，无异于徒手挖矿，是极其低效的。根据没有免费的午餐定理，我们设计的算法只有与这些数据背后任务的特征相适配，才能取得良好的性能。这方面，生物神经网络给我们提供了一个重要的参考。毫无疑问，生物神经网络的结构及其学习算法与

我们关注的各种图像和语言任务高度适配。那么，模仿生物神经网络来设计一种人工神经网络算法是一种自然的想法。当前，我们清楚地看到，在突破数据和算力两个瓶颈后，基于人工神经网络的深度学习方法取得了空前的成功。比如，一个基于 transformer 神经网络的大语言模型从人类几乎全部的文字数据中提取我们关心的信息。这种信息提取的能力是早些年完全无法想象的。

和生物神经系统的进化类似，深度学习算法的进化来自对于数据集的提取信息能力（比如用测试准确率来衡量）的优胜劣汰。深度学习算法在这些年主要的进步来自神经网络架构的演进，比如从全连接网络、卷积网络、长短程记忆网络、残差网络再到 Transformer。经验上，我们注意到一个架构与所处理数据的内在结构越一致，就越能取得更好的性能。

本章将探讨数据特征与神经网络架构之间的紧密联系。我们将看到，不同的数据特征如何启发了不同的网络设计；而不同的网络架构又如何体现了对数据特征的不同先验假设。通过这样的讨论，我们希望能够建立起数据、任务和模型之间的桥梁，为读者理解和设计神经网络架构提供一个全面的视角。

科普篇

数据与神经网络架构存在什么联系？

神经网络的架构设计与所处理的数据类型密切相关，这种联系可以通过没有免费午餐定理来解释。没有免费午餐定理指出，对于任何优化算法，其在所有可能的问题空间上的平均表现是相同的。这意味着没有一种通用的算法能够在所有情况下都优于其他算法。因此，特定的数据类型需要特定的网络架构来最优化其性能。例如，

- **图像数据**：图像数据具有局部相关性和空间结构特征。卷积神经网络（Convolutional neural network, CNN）通过使用卷积层，可以捕捉图像中的局部特征如边缘和纹理，并通过池化层减少计算量，同时保留重要的空间信息。CNN 的权重共享和局部连接特点使其特别适合处理图像数据。
- **语言数据**：语言数据是时序数据，具有上下文依赖性。Transformer 架构通过自注意力机制，能够在处理过程中同时关注整个序列中的所有词语，捕捉远距离依赖关系。这使得 Transformer 在处理语言任务（如机器翻译、文本生成和语义分析）时表现优异。相较于递归神经网络（Recurrent neural network，RNN），Transformer 能够更好地处理长距离依赖，并行计算也提高了训练效率。

没有免费午餐定理提醒我们，不同类型的数据需要不同的架构来充分利用其特点，从而达到最佳的性能。

残差神经网络是什么？它解决了什么问题？

残差神经网络（Residual Neural Networks，ResNets）是由 He et al. 在 2015 年提出的一种改进的深度神经网络结构，其核心思想是通过引入残差连接（skip connections）来解决深层神经网络中的退化问题，即随着网络层数的增加，深度神经网络可能会出现训练误差增加的问题，即退化现象。这并不是因为过拟合，而是因为梯度消失或梯度爆炸，使得训练变得困难。

残差连接通过直接将某一层的输入添加到该层的输出，使得网络可以学习残差函数（即实际输出与输入的差值），而不是直接学习输入到输出的映射。这一设计极大地缓解了深层网络中的梯度消失问题。

数学上，假设某一层的输入为 x，对应的理想映射为 $H(x)$。传统网络直接学习 $H(x)$，而 ResNet 通过引入残差连接，学习残差函数 $F(x) = H(x) - x$。因此，该层的输出为 $H(x) = F(x) + x$。

主要优点有：

- **缓解梯度消失问题**：残差连接使得梯度能够更容易地反向传播至前面的层，从而缓解了梯度消失问题。
- **提高训练效率**：由于每个残差块都包含直接的跳跃连接，网络可以更快地收敛。
- **支持更深的网络结构**：通过引入残差块，ResNet 能够训练出非常深的网络（如 100 层以上），并且仍然具有良好的性能。

这些特点使得 ResNet 在图像分类和目标检测等任务中取得了显著的成功，被广泛应用于各种深度学习应用中。

图像数据有什么特点？为什么卷积神经网络更加适配图像数据？

图像数据的特点包括：

- **局部相关性**：图像中的像素通常与其邻近像素有很强的相关性。例如，一张照片中的一部分可能代表了一个特定的物体，而相邻的像素共同定义了这个物体的特征。
- **空间不变性**：图像的模式（如边缘、角点等）在不同位置出现时具有相似的特征。

卷积神经网络（CNNs）更加适配图像数据的原因如下：

- **卷积操作**：卷积层通过在输入图像上滑动滤波器（或称为卷积核），能够检测到局部的图像特征，如边缘和纹理。滤波器参数是通过训练学习的，能够自适应地提取最有利于任务的特征。

- **权重共享**：同一卷积核在整个图像上共享参数，这大大减少了模型的参数数量，降低了计算复杂度和内存需求，同时增强了模型的泛化能力。
- **池化层**：池化操作（如最大池化和平均池化）通过下采样来减小特征图的尺寸，从而减少计算量，同时保留主要特征。这有助于实现一定程度的平移不变性，即物体在图像中移动时仍能被识别。

这些特点使得 CNN 在处理图像数据时能够有效提取并利用空间信息，从而在图像分类、目标检测等任务中表现出色。

语言数据有什么特点？为什么 Transformer 更加擅长处理语言任务？

语言数据的特点包括：

- **序列性**：语言数据具有自然的顺序，词语的顺序对句子意义有着重要影响。
- **上下文依赖性**：语言中的词语往往需要依赖前后文才能确定其确切含义。

Transformer 在处理语言任务时表现优异的原因如下：

- **自注意力机制**：Transformer 通过自注意力机制，能够在处理每个词语时同时关注整个句子中的其他词语。这种机制允许模型捕捉长距离的依赖关系，而不受序列长度的限制。
- **并行计算**：与传统的 RNN 不同，Transformer 能够并行处理序列中的所有位置，从而显著提高训练速度和效率。这是因为 RNN 需要依次处理每个时间步的输入，而 Transformer 可以一次性处理整个序列。
- **层次结构**：Transformer 由多个编码器和解码器层堆叠而成，每一层都包括多头自注意力机制和前馈神经网络。多头自注意力机制使模型能够关注输入序列中不同位置的不同信息，从而增强了模型的表示能力。

这些特点使得 Transformer 在处理语言任务（如机器翻译、文本生成和语义分析）时具有显著优势，特别是在需要捕捉复杂的上下文依赖和长距离关系的情况下。

2.1 全连接网络

全连接网络是一种经典而又基础的神经网络结构，通常由输入层、隐藏层和输出层组成，每一层都包含若干个神经元节点。除了输入层外，网络中每个神经元都与上一层的所有神经元相连接，形成了“全连接”的结构。这些连接都具有可学习的权重参数，使得网络能够通过训练来适应不同的任务。

在全连接网络的前向传播过程中, 每个隐藏层神经元会对来自上一层的输入信号进行加权求和, 然后通过一个非线性激活函数 (如 ReLU、Sigmoid 等) 进行变换, 以增强网络的表示能力。这种逐层的非线性变换使得全连接网络能够拟合复杂的函数关系, 在许多领域都有广泛的应用。

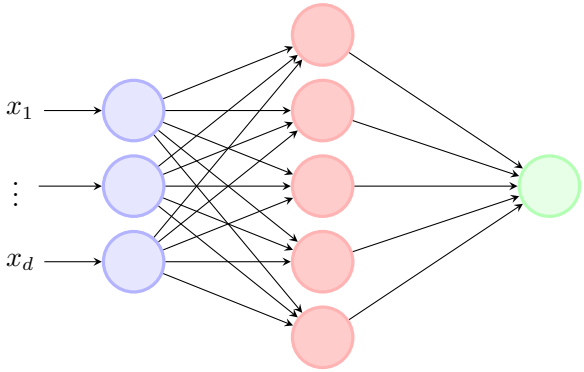


图 2.1: 两层神经网络

图2.1展示了一个简单的两层全连接神经网络结构示意图。从数学角度来看, 对于一个 L 层的全连接网络, 其前向传播过程可以用如下公式来表示:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]}\sigma \circ (\boldsymbol{W}^{L-2}\sigma \circ (\cdots (\boldsymbol{W}^{[1]}\sigma \circ (\boldsymbol{W}^{[0]}\boldsymbol{x} + \boldsymbol{b}^{[0]}) + \boldsymbol{b}^{[1]}) \cdots) + \boldsymbol{b}^{[L-2]}) + \boldsymbol{b}^{[L-1]}, \quad (2.1)$$

其中 $\boldsymbol{W}^{[l]} \in \mathbb{R}^{m_{l+1} \times m_l}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{m_{l+1}}, m_0 = d_{in} = d, m_L = d_o$ 分别表示第 l 层的权重矩阵和偏置向量,σ 是一个非线性激活函数, 它对输入向量中的每个元素分别作用。符号 “$\circ$” 表示对应元素的运算 (entry-wise operation)。需要注意的是, 在描述网络层数时, 我们通常不将输入层计入其中, 即网络的层数等于隐藏层与输出层之和。

为了方便表示, 我们把网络中的所有参数记为

$$\boldsymbol{\theta} = (\boldsymbol{W}^{[0]}, \boldsymbol{W}^{[1]}, \cdots, \boldsymbol{W}^{[L-1]}, \boldsymbol{b}^{[0]}, \boldsymbol{b}^{[1]}, \cdots, \boldsymbol{b}^{[L-1]}),$$

其中 $\boldsymbol{W}_{ij}^{[l]}$ 表示 $\boldsymbol{W}^{[l]}$ 中的第 i 行第 j 列元素。

除了上述前向传播公式外, 我们还可以用递归的方式来定义全连接网络:

$$f_{\boldsymbol{\theta}}^{[0]}(\boldsymbol{x}) = \boldsymbol{x} \quad (2.2)$$

$$f_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x}) = \sigma \circ (\boldsymbol{W}^{[l-1]} f_{\boldsymbol{\theta}}^{[l-1]}(\boldsymbol{x}) + \boldsymbol{b}^{[l-1]}), \quad 1 \le l \le L-1, \quad (2.3)$$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}}^{[L]}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]} f_{\boldsymbol{\theta}}^{[L-1]}(\boldsymbol{x}) + \boldsymbol{b}^{[L-1]}. \quad (2.4)$$

这种递归定义的方式更加简洁, 突出了全连接网络的层次结构特点。

全连接网络中的每个连接都有其独立的权重参数, 不存在参数共享的情况。因此, 当输入维度较高时, 网络的参数量会呈现出组合爆炸式的增长。这一特点使得全连接网络在处理高维数据时面临着参数优化困难、计算复杂度高等挑战。

尽管如此, 全连接网络仍然是深度学习中一个重要的基础结构。它主要用于处理经过向量化的结构化数据, 具有很强的函数拟合能力。在实践中, 全连接网络常作为其他更复杂网络结构的组件, 例如卷积神经网络（CNN）中的分类器部分。

但是, 由于全连接网络缺乏对数据先验知识的利用, 因此在直接处理原始图像、序列等高维非结构化数据时, 其泛化性能往往不够理想。为了克服这一局限性, 研究者们发展出了卷积神经网络、循环神经网络等更适合处理特定类型数据的网络结构。它们通过引入一些先验假设（如局部连接、权重共享等）, 在提高参数效率的同时, 也增强了模型对数据特征的捕捉能力。

2.2 残差神经网络

随着深度学习的发展，研究者们发现, 增加网络的深度可以提高模型的表达能力, 从而在图像分类、目标检测等任务上取得更好的性能。然而, 随着网络层数的增加, 训练深层网络变得越来越困难。这主要有两个原因: 一是梯度消失或爆炸问题, 导致深层网络难以训练; 二是退化问题 (degradation problem), 即网络深度增加时, 性能出现饱和甚至下降的现象 (He et al. 2016)。

为了解决这些问题, 何恺明等人 (He et al. 2016) 在 2015 年提出了残差神经网络 (Residual Neural Network, ResNet)。ResNet 的核心思想是引入了一种称为“残差连接”(Residual Connection) 的结构, 使得网络可以学习残差函数, 而不是直接学习目标函数。这种设计大大缓解了深层网络训练的难度, 使得研究者们可以训练数百层甚至上千层的网络, 并在多个任务上取得了当时的最佳效果。

这里我们介绍一种最简单最基本的残差连接方式:

$$F_{l+1} = \sigma(W_l F_l + b_l) + F_l,$$

其中，F_l 是第 l 层的输出。这种情况在 F_l 和 F_{l+1} 的维度一样的时候适用。当它们的维度不一样时，可以用下面的形式

$$F_{l+1} = \sigma(W_l F_l + b_l) + W_s F_l,$$

其中 W_s 是一个可训练的矩阵，其目的是将维度转化为 $l+1$ 层的维度。

残差的关键在于, 我们将输入 x 直接加到了该层的输出上, 形成了一个“旁路”或“捷径”连接。这个连接被称为残差连接。

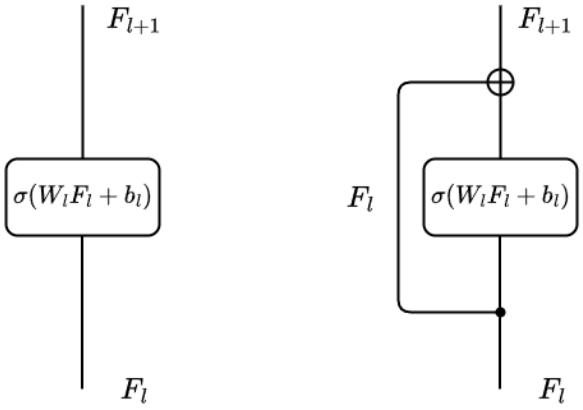


图 2.2: 全连接网络架构与带残差连接的网络架构

图2.2对比了传统网络和残差网络的结构差异。可以看到, 在残差网络中, 输入 F_l 通过一个旁路直接连接到了输出, 与经过一个全连接层作用后的输出相加。这个设计的优点是, 即使全连接层学习的映射函数不理想, 网络仍然可以通过旁路连接来传递输入信息, 确保网络性能不会恶化。换句话说, 残差连接使得网络更容易学习恒等映射, 从而缓解了深层网络的优化难题。

从另一个角度看, 残差连接提供了一种融合不同层特征的方式。通过将浅层特征直接传递到深层, 网络可以灵活地选择使用哪些特征, 从而增强了模型的表达能力。此外, 残差连接还起到了类似于" 梯度高速公路" 的作用, 使得梯度可以更顺畅地在深层网络中传播, 缓解了梯度消失的问题。

残差网络的提出是深度学习发展历程中的一个里程碑。它不仅在 ImageNet 图像分类任务上取得了当时的最佳成绩, 而且开创了一种全新的网络设计范式。此后, 许多经典的网络结构, 如 Inception-ResNet(Szegedy et al. 2017)、DenseNet(Huang et al. 2017)、ResNeXt(Xie et al. 2017) 等, 都借鉴了残差连接的思想, 将其扩展到了更广泛的应用场景中。

残差网络的影响不仅限于计算机视觉领域, 在自然语言处理、语音识别等领域也得到了广泛应用。例如, 谷歌的机器翻译系统 (Wu et al. 2016) 就采用了残差连接来构建深层的编码器-解码器网络, 大大提高了翻译质量。微软的语音识别系统也使用了残差网络来提取语音特征 (Xiong et al. 2018)。

残差网络的提出解决了深层网络训练的关键难题, 促进了神经网络的进一步发展。它不仅在学术界产生了深远影响, 也在工业界得到了广泛应用, 推动了人工智能技术的进步。可以说, 残差网络是现代深度学习的基石之一, 为后续的研究工作奠定了坚实的基础。

2.3 卷积神经网络

卷积神经网络 (Convolutional Neural Network, CNN) 是深度学习中处理图像数据的利器。自上世纪 80 年代起,CNNs 就开始在图像识别领域崭露头角。1989 年,LeCun 等人 (LeCun et al. 1989) 首次将 CNN 应用于手写数字识别, 取得了里程碑式的成果。此后,CNNs 在计算机视觉领域大放异彩, 在图像分类 (Krizhevsky et al. 2012)、目标检测 (Girshick et al. 2014)、语义分割 (Long et al. 2015) 等任务上不断刷新性能记录, 远远超越了传统机器学习方法。2012 年,Krizhevsky 等人 (Krizhevsky et al. 2012) 提出的 AlexNet 在 ImageNet 图像分类挑战赛中以显著优势夺冠, 引爆了深度学习的研究热潮, 也奠定了 CNN 在计算机视觉领域的霸主地位。

CNN 在图像领域的优越性能, 归功于其独特的网络结构设计。与传统的全连接神经网络不同,CNN 引入了一系列针对图像特点的创新结构, 如局部感受野、权值共享、池化等。这些设计使得 CNN 能够高效地提取图像的局部特征, 并逐层组合形成更加抽象和鲁棒的表示。可以说, 正是得益于对图像特征的充分利用,CNN 才有了如此出众的性能。

要真正理解 CNN 的精妙构思, 我们需要先从一个更本质的问题出发: 图像数据集具有什么样的特征? 唯有洞悉图像数据的本质属性, 我们才能领会 CNNs 背后的设计思路, 进而更好地应用并对该工具进行创新。同时我们也需要关注人和其他生物如何处理图像数据, 生物的视皮层对 CNN 的构造具有哪些启发。因此在本节中, 我们将首先介绍图像数据集的特征, 之后介绍对卷积神经网络设计具有重大启发意义的生物初级视皮层的图像处理结构, 最后将系统介绍 CNN 的核心结构, 揭示其内在的工作机制。

2.3.1 图像数据集的特征

视觉信息在人类接收的信息中占据了至关重要的地位。一些研究估计, 视觉信息可能占人类获取信息总量的 80% 以上。尽管这一比例难以精确量化, 但它揭示了一个事实: 人类是一个" 视觉动物", 视觉在我们的感知系统中扮演着主导角色。因此, 要研究视觉人工智能, 我们必须深入理解自然图像数据的特征。

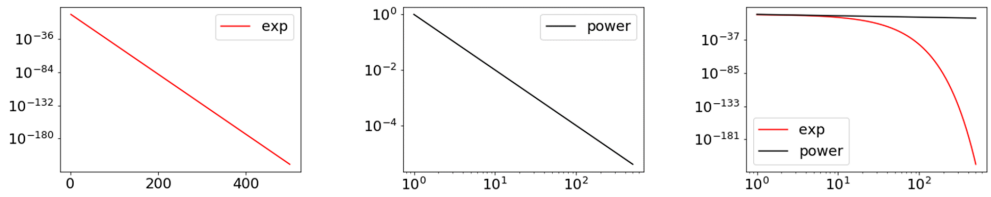
大量研究和经验指出, 关联性和不变性是图像数据的两个关键特征。关联性指的是图像中的像素并非独立, 而是存在着强烈的相关性。这种相关性体现在时域 (相邻像素倾向于取相似的值) 和频域 (图像的频谱通常集中在低频部分)。这启示我们, 图像压缩、去噪等任务可以利用这种关联性。不变性指的是图像的语义内容在一定程度上对变换保持不变。例如, 即使一张猫的照片经过了平移、旋转、放缩, 我们仍然能识别出它是一只猫。这启示我们, 视觉智能系统需要对这些变换有一定的鲁棒性。

关联性

很多具有空间或者时间结构的数据都有局部关联的特点，也就是随着距离增长，两个空间点上的数据的关联强度不断下降。这种局部关联性在自然图像中尤为明显。图像中的像素并非独立分布，而是存在着强烈的空间相关性。相邻像素通常具有相似的灰度值或颜色，这源于现实世界中物体表面的连续性和光照的平滑变化。理解和利用这种关联性，对于图像处理、压缩和特征提取等任务至关重要。

为了刻画关联性的衰减特征，我们首先介绍两种常见的衰减模型：指数衰减和幂级数衰减。指数衰减的特征为具有一定的衰减周期，初始值在经过一个衰减周期之后会变为原来的一半，我们称这个衰减周期为半衰期。例如，对于一个形如 $y = e^{-\frac{x}{\tau}}$ 的指数衰减模型，它的半衰期为 $\tau \ln 2$。指数衰减在许多物理过程中都有体现，如放射性元素的衰变、电容器的放电等。

幂级数衰减没有衰减周期，如图2.3(c)，它比指数衰减要慢地多，例如 $y = 1/x$，当 x 越大的时候，其衰减就越慢。另一种理解幂级数衰减的办法是通过指数衰减，幂级数衰减可以理解为有无穷多个半衰期的指数衰减的和，例如，考察 $\int_0^\infty \exp(-\alpha x) d\alpha = \frac{1}{x}$，其左式为周期各不相同的指数衰减的积分，右侧为幂级数衰减。幂级数衰减在描述许多自然现象时非常有用，如引力势能与距离的关系、城市人口分布等。

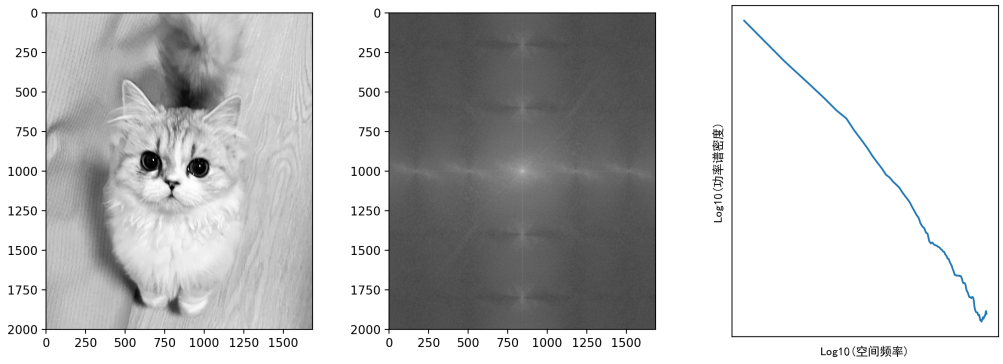


(a) 指数衰减　(b) 幂级数衰减　(c) 指数衰减和幂级数衰减速度对比

图 2.3: 数据衰减

为了验证自然图像的局部关联特性，我们可以计算两个常用的量：功率谱和关联函数。事实上，这两个量可以通过傅里叶变换相互转换。

图像的功率谱反映了图像在不同频率下的能量分布。通过对图像进行傅里叶变换，然后计算每个频率分量的模长平方，我们就得到了功率谱。如图2.4所示，在频谱图中，亮度越高表示该频率分量的能量越大。对大量自然图像的功率谱进行分析，我们发现它们普遍呈现出幂律分布的特点，即低频分量占据主导，高频分量所占比重较小，但不可忽略。这意味着图像的能量大部分集中在较低的频率范围内，这与图像的局部平滑性和缓变特性是一致的。同时，高频分量的存在则反映了图像中的细节、边缘、纹理等结构性信息。功率谱的幂律特性，为图像处理和分析提供了重要的先验知识。



(a) 图像　　(b) 二维频谱图　　(c) 径向平均频谱图

图 2.4: 图像的二维频谱图与径向平均频谱图

除了频域分析，我们还可以直接在空间域研究图像的关联性。一个常用的度量是相关函数，它描述了图像中两个像素点之间的相关程度与它们距离的关系。对于两个像素点 ϕ_1 和 ϕ_2，它们的相关性可以用互信息 I 来度量：

$$I = \iint p_2\left(\phi_1, \phi_2\right) \log \frac{p_2\left(\phi_1, \phi_2\right)}{p_1\left(\phi_1\right) p_1\left(\phi_2\right)} \mathrm{d}\phi_1 \mathrm{d}\phi_2, \tag{2.5}$$

其中，$p_2(\phi_1, \phi_2)$ 是两个像素点的联合概率分布，$p_1(\phi_1)$ 和 $p_1(\phi_2)$ 分别是两个像素点的边缘概率分布。当两个像素点相关时，$I > 0$；当它们独立时，$p_2\left(\phi_1, \phi_2\right) = p_1\left(\phi_1\right) p_1\left(\phi_2\right)$，因此 $I = 0$。

通过计算不同距离下像素点对的平均互信息，我们可以得到图像的相关函数。相关函数定量刻画了图像的局部相关性随距离的衰减趋势。如图 2.5所示，实验和研究表明，对于自然图像，相关函数通常呈现出类似幂律衰减的形式，即近距离像素点的相关性很强，而远距离像素点的相关性较弱，但并非完全无关。这种空间相关性的非局域性，启发我们在处理图像时，不仅要考虑局部邻域的信息，还要适当利用一些远程依赖关系。

图像数据呈现出明显的空间关联性，这既是机遇也是挑战。一方面，我们可以利用这些先验知识来设计更高效、更鲁棒的图像处理和分析算法；另一方面，我们也需要开发出能够充分利用图像关联性的机器学习模型和训练策略，从海量数据中挖掘出有价值的信息。深度学习模型，如卷积神经网络，就是一个很好的例子，它通过局部连接和权值共享，很好地利用了图像的空间相关性，在计算机视觉领域取得了巨大的成功。

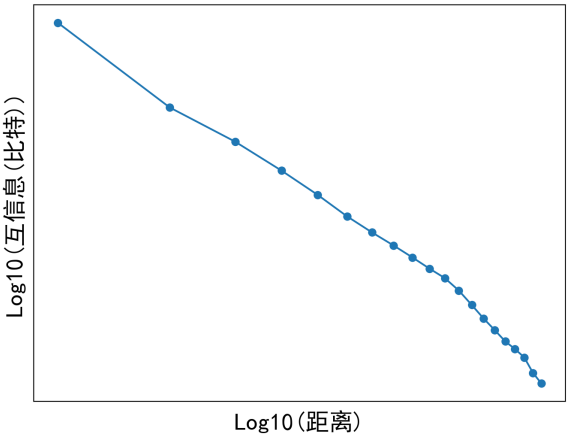


图 2.5: 自然图像像素间的相关性与像素间距离呈现幂律衰减形式

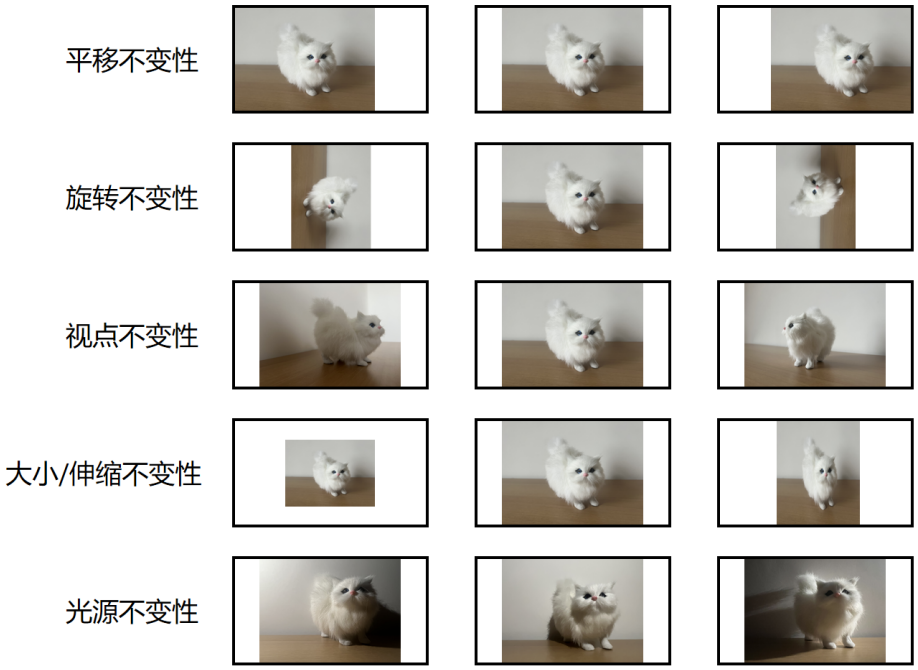


图 2.6: 图像的各种不变性

不变性

图像数据的不变性是指图像的语义内容在某些变换下保持不变的特性。这些变换包括平移、旋转、缩放、亮度变化等，图2.6中展示了各种不变性。不变性是视觉智能的一个关键要求，因为在现实世界中，同一个物体可能会以不同的姿态、角度、大小出现，但我们仍然希望智能系统能够准确地识别它。下面我们从数学的角度来详细理解图像数据的不变性。

平移不变性 平移不变性是指图像在空间上的移动不影响其内容。假设一个图像可以表示为一个二维函数 $f(x,y)$，其中 (x,y) 表示像素的坐标。图像 f 在水平方向上向右平移 a 个单位，在垂直方向上向上平移 b 个单位，可以表示为:

$$f'(x,y) = f(x-a, y-b).$$

如果一个图像处理算法或特征对平移不变，那么它在原图像 f 和平移后的图像 f' 上的输出应该是一样的。

旋转不变性 旋转不变性是指图像在旋转后，其内容不发生改变。对一个图像 $f(x,y)$，如果我们将其逆时针旋转 θ 度，旋转后的图像 $f'(x,y)$ 可以表示为:

$$f'(x,y) = f(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta).$$

如果一个算法对旋转不变，那么它在 f 和 f' 上的输出应该相同。

缩放不变性 缩放不变性是指图像在大小上的缩放不影响其内容。如果我们将图像 $f(x,y)$ 在水平和垂直方向上分别缩放 α 和 β 倍，缩放后的图像 $f'(x,y)$ 可以表示为:

$$f'(x,y) = f(\frac{x}{\alpha}, \frac{y}{\beta}).$$

如果一个算法对缩放不变，那么它在 f 和 f' 上的输出应该一致。

亮度不变性 亮度不变性是指图像在亮度上的变化不影响其内容。如果我们将图像 $f(x,y)$ 的亮度增加或减少一个常数 c，变化后的图像 $f'(x,y)$ 可以表示为:

$$f'(x,y) = f(x,y) \pm c.$$

如果一个算法对亮度不变，那么它在 f 和 f' 上的输出应该相同。

光源不变性 光源不变性是指图像的内容在光照条件变化下应该保持不变。无论光源来自图像的左侧还是右侧，图像的语义内容应该不受影响。

假设一个图像 $f(x,y)$ 在光照条件 L_1 下被拍摄，得到的图像为 $f_{L_1}(x,y)$。如果我们改变光源的位置和角度，得到另一个光照条件 L_2，在该光照条件下拍摄的图像为 $f_{L_2}(x,y)$。如果一个图像识别算法对光源不变，那么它在 f_{L_1} 和 f_{L_2} 上的识别结果应该是一致的。

从数学上讲，我们可以将光照变化建模为一个函数 g，它将原始光照条件下的图像映射到新的光照条件下的图像：

$$f_{L_2}(x,y) = g(f_{L_1}(x,y)).$$

如果一个识别算法 R 对光源不变，那么它应该满足：

$$R(f_{L_1}(x,y)) = R(f_{L_2}(x,y)) = R(g(f_{L_1}(x,y))).$$

也就是说，无论图像是在光照条件 L_1 下还是在光照条件 L_2 下拍摄的，识别算法都应该给出相同的结果。这个性质对于在现实世界中部署视觉智能系统非常重要，因为现实环境中的光照条件是多变的，我们希望我们的算法能够在不同光照下稳定工作。

统计不变性 统计不变性是指图像的统计特性在不同区域应该是一致的。这启示我们, 可以使用相同的方法从图像的不同部分提取特征。

举个例子, 假设我们要从图像中检测边缘。直观地说, 图像中的物体边缘对应着图像像素值快速变化的区域。我们可以设计一个算子, 它在图像的每个位置计算其左边像素值减右边像素值，这样的算子就可以提取竖直的边缘。如果统计不变性成立, 那么这个算子在图像的不同区域应该表现一致, 因为边缘的统计特性是相同的。

另一个例子是图像的自相关函数。自相关函数衡量了图像中不同位置像素之间的相关性, 它只取决于这些位置之间的相对距离, 而不依赖于它们的绝对位置。形式化地, 对于一个充分大的自然数据集 $\mathcal{S}$ 中的所有图像 $f(x,y)$, 它的自相关函数 $C_{x,y}(a,b)$ 定义为:

$$C_{x,y}(a,b) = \mathbb{E}_f f(x,y) f(x-a, y-b).$$

如果统计不变性成立, 那么 $C_{x,y}(a,b)$ 应该与 (a,b) 的选择无关, 只依赖于距离 $\sqrt{a^2+b^2}$。这是因为图像的统计特性在平移下是不变的。

统计不变性反映了自然图像的一个基本性质: 它们在不同区域具有相似的统计特征。这个性质在纹理分析、图像合成等任务中有着广泛的应用。它也为我们理解和设计图像处理算法提供了重要的先验知识: 由于统计不变性, 我们可以在整个图像上重复使用相同的特征提取算法, 从而大大减少参数量。

在深度学习中，我们通常通过以下方式来实现或利用图像数据的不变性:

1. 数据增强：在训练过程中，我们对图像进行随机的平移、旋转、缩放、亮度变化等操作，生成更多的训练样本。这样可以提高模型的不变性和鲁棒性。

2. 特征设计：一些传统的图像特征，如 SIFT（尺度不变特征变换）、HOG（方向梯度直方图）等，通过特殊的数学设计，对平移、旋转、缩放等变换具有不变性。这些特征可以作为深度学习的输入，或者作为设计深度网络结构的灵感来源。

3. 卷积神经网络（CNN）：CNN 通过局部感受野、权值共享、池化等操作，充分利用了图像数据的平移不变性。此外，通过数据增强和特殊的网络结构设计，CNN 也可以获得一定的旋转和缩放不变性。这也是我们接下来要重点介绍的内容。

图像数据的不变性是视觉智能的重要基础。通过数学分析和算法设计，我们可以更好地理解和利用这一特性，构建更加鲁棒、泛化能力更强的视觉智能系统。在深度学习时代，不变性的实现和应用也在不断更新迭代，为计算机视觉的发展注入了新的活力。

2.3.2 初级视皮层的图像处理结构

在上一节中，我们探讨了图像数据的关键特征，如关联性、不变性等。这些特征反映了视觉世界的内在规律，也为视觉智能的发展指明了方向。同时我们也需要注意到，大自然在漫长的进化过程中已经创造出了一个非凡的视觉系统——生物视觉。那么，生物视觉系统是如何处理图像信息的呢？它又为人工神经网络的设计带来了哪些启示？

要回答这些问题，我们需要深入了解视觉皮层的结构和功能。20 世纪 50、60 年代，神经科学家 David Hubel 和 Torsten Wiesel 在猫和猴子的视觉皮层进行了开创性的研究，揭示了视觉信息在大脑中的加工机制。他们的工作不仅加深了我们对视觉认知的理解，也为后来的卷积神经网络提供了重要的生物学基础。

Hubel 和 Wiesel 发现，初级视皮层 (V1) 中存在两类功能不同的细胞：简单细胞和复杂细胞。简单细胞对视野中特定位置和方向的线条或边缘非常敏感。当视野中出现与其优选方向一致的刺激时，简单细胞就会产生强烈的电生理响应。不同的简单细胞偏好不同方向的刺激，这意味着，初级视皮层实际上在提取图像中的基本特征，如边缘和线条。

图2.7展示了一个简单细胞的理想模型。该细胞的感受野由一系列 on 区 (兴奋) 和 off 区 (抑制) 组成，呈现一种规则的空间排列。当视觉刺激与该细胞的感受野结构相匹配时，即亮区落在 on 区、暗区落在 off 区，该细胞就会被激活。这种精细的空间整合使得简单细胞能选择性地响应特定方向的边缘或线条。

进一步的研究发现，复杂细胞可以整合来自多个简单细胞的输入，对更复杂的特征产生选择性响应。例如，一个复杂细胞可能偏好某个方向的边缘，但它对该边缘在视野中的精确位置并不敏感。这种特性被称为位置不变性或位置容差。复杂细胞的发现表明，视觉皮层以分层递阶的方式逐步提取越来越抽象和复杂的特征。

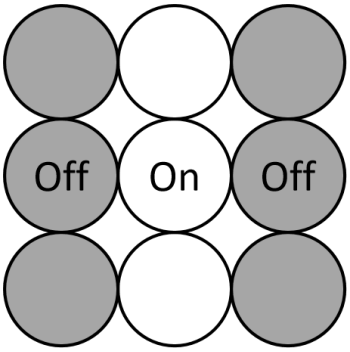


图 2.7: V1 区简单细胞的感受野示例

这些神经科学的发现对人工神经网络的设计具有重要的启发意义。我们可以设想, 在全连接网络的基础上, 可以引入一种类似于视觉皮层简单细胞的特征提取器。这种提取器应该能够在不同位置检测同一种特征, 具备一定的位置不变性。事实上, 这就是卷积神经网络中的核心组件——卷积核的雏形。同时, 通过组合这些基本特征提取器, 我们可以构建出对更复杂模式敏感的单元, 犹如视觉皮层中的复杂细胞。此外, 视觉皮层的分层结构启示我们, 通过逐层堆叠这些特征提取单元, 神经网络能够学习到层次化的特征表示, 从而对输入图像形成深入的理解。

Hubel 和 Wiesel 的开创性工作揭示了视觉皮层的基本计算原理, 为视觉信息处理的研究开辟了一个新的时代。它不仅加深了我们对生物视觉的认识, 也为人工神经网络, 特别是卷积神经网络的发展奠定了基础。尽管我们尚未明确提出卷积神经网络的概念, 但从视觉皮层的结构中, 我们已经可以看到其雏形。在接下来的章节中, 我们将进一步探讨如何借鉴这些生物学洞见, 构建出强大的视觉智能模型。

2.3.3 卷积神经网络

在上一节中, 我们探讨了生物视觉系统, 特别是初级视皮层在处理视觉信息方面的惊人能力。简单细胞和复杂细胞的发现揭示了视觉皮层的一些基本计算原理, 如局部特征提取、方向选择性和位置不变性。这些原理不仅加深了我们对生物视觉的理解, 也为设计高效的人工视觉系统提供了重要的启示。

受到视觉皮层结构的启发, 研究者提出了一种称为卷积神经网络 (CNN) 的人工神经网络模型。CNN 的设计巧妙地利用了图像数据的两大特征: 局部相关性和平移不变性。通过引入卷积操作和池化操作,CNN 能够有效地提取图像的局部特征, 并对物体的位置变化具有较强的鲁棒性。

一个标准的 CNN 通常由以下几个模块组成:

- 卷积层 (convolutional layer): 这一层的核心操作是卷积。卷积层中的卷积核可以看作一种局部特征提取器, 与视觉皮层中的简单细胞类似。每个卷积核在图像上滑动, 提取不同位置的局部特征。多个卷积核可以提取不同类型的特征, 如边缘、纹理等。卷积操作具有平移不变性, 即无论目标特征出现在图像的什么位置, 都能被同一个卷积核检测到。这与图像和视觉皮层的平移不变性相呼应。

- 池化层 (pooling layer): 池化层通常紧跟在卷积层之后, 其作用是压缩空间维度。常见的池化操作包括最大池化和平均池化。池化层可以提取图像的轮廓信息, 并进一步增强模型的平移不变性。以最大池化为例, 在每个局部区域内, 最大池化总是输出该区域数据的最大值, 无论特征在这一局部区域内如何平移, 最大值总是不变的。这种不变性有助于模型抓住图像的关键特征, 而忽略细枝末节。

- 全连接层 (fully-connected layer): 在若干卷积-池化层之后,CNN 通常会将得到的二维特征图展平为一维向量，并接入一到两个全连接层。这一部分与传统的全连接网络结构类似, 可以看作是对提取到的高层特征进行处理和预测。

- 输出层 (output layer): 输出层的神经元数量取决于任务的类型。例如, 在图像分类问题中, 输出层的神经元数对应分类的类别数; 在目标检测问题中, 输出层需要预测目标的位置和类别; 在图像生成问题中, 输出层可能是一个高维的图像向量。输出层的设计与任务密切相关。

CNN 通过卷积层和池化层的层层堆叠, 逐步提取图像的局部特征并抽象出高层表示, 实现了从像素到语义的映射。这种分层递进的结构在一定程度上模拟了视觉皮层的信息处理过程。CNN 的巧妙设计使其在图像识别、目标检测等任务上取得了巨大成功, 也使其在很长一段时间里成为计算机视觉领域的主流模型。

在接下来的小节中, 我们将深入探讨 CNN 的关键组件, 如卷积核、填充和跨步、多通道、池化等, 揭示其内在的工作原理，使读者对 CNN 有更加全面的认识。

卷积操作

卷积 (convolution) 是卷积神经网络的核心操作，它在图像处理和计算机视觉领域有着悠久的历史。最早可以追溯到 1980 年，Kunihiko Fukushima 提出了 Neurocognitron，这是一种受生物神经网络启发的模型，首次在机器学习中引入了类似卷积的概念，用于模拟人类视觉系统中的特征提取过程Fukushima (1980)。随后，Yann LeCun 于 1989 年将卷积的思想与反向传播算法结合，提出了卷积神经网络（CNN），并在手写数字识别任务中取得了突破性成果LeCun et al. (1989)。这些早期的工作不仅证明了卷积在图像处理中提取特征的有效性，也为后续深度学习的蓬勃发展奠定了基础。

如今，卷积已经成为深度学习中不可或缺的工具，其重要性怎么强调都不为过。在现代的卷积神经网络中，卷积操作能够通过局部感受野的设计来有效捕捉图像中的空间特征，从而大幅提高了计算效率和特征提取的质量。这使得卷积神经网络在图像分类、目标检测、语义分割等任务中展现出强大的性能，并在多个计算机视觉应用中得到广泛应用。

那么, 卷积到底是什么呢? 简单来说, 卷积是一种数学运算, 它描述了两个函数之间的相互作用。在图像处理中, 这两个函数通常是输入图像和卷积核 (也称为过滤器)。卷积的过程可以看作是卷积核在输入图像上滑动, 并在每个位置计算加权和。这个过程可以提取图像的局部特征, 如边缘、纹理等。

卷积的数学定义 为了更好地理解卷积神经网络中的卷积操作, 我们首先给出其数学定义。设 $f(t)$ 和 $g(t)$ 是实数域上的两个可积函数，定义二者的卷积 $(f*g)(t)$ 为如下的积分变换：

$$(f*g)(t)=\int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,\mathrm{d}\tau. \tag{2.6}$$

因为图像是二维的，所以这里我们考虑二维卷积的情况, 其中 $f(x,y)$ 表示输入的特征图,$g(x,y)$ 表示卷积核函数。此时连续版本的卷积公式为：

$$(f*g)(t_1,t_2)=\int_{-\infty}^{\infty} f(\tau_1,\tau_2)g(t_1-\tau_1,t_2-\tau_2)\,\mathrm{d}\tau_1\,\mathrm{d}\tau_2. \tag{2.7}$$

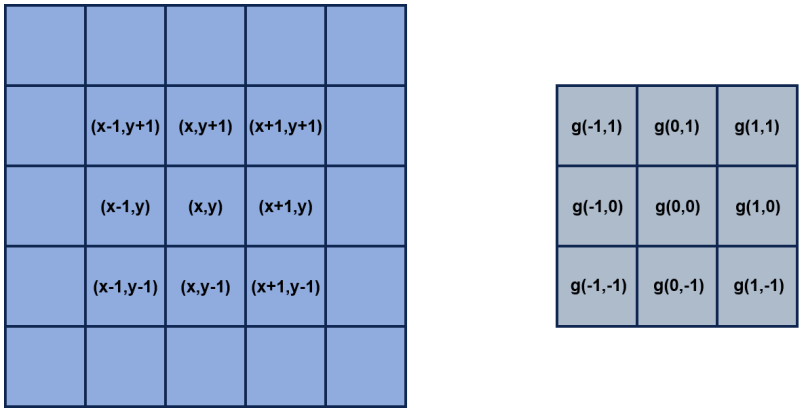


图 2.8: 卷积运算示意图

图2.8展示了卷积运算过程, 左图为输入特征图 $f(x,y)$, 右图为卷积核 $g(x,y)$。卷积运算可以看作是卷积核在输入特征图上滑动, 并计算加权和的过程。数学上, 二维卷积运算可以表示为:

$$
\begin{aligned}
F(x,y) = {} & f(x-1,y+1)g(-1,1) + f(x,y+1)g(0,1) + f(x+1,y+1)g(1,1) \\
& + f(x-1,y)g(-1,0) + f(x,y)g(0,0) + f(x+1,y)g(1,0) \\
& + f(x-1,y-1)g(-1,-1) + f(x,y-1)g(0,-1) + f(x+1,y-1)g(1,-1)
\end{aligned}
$$

这个公式表明, 卷积运算是通过将卷积核与输入特征图的局部区域进行逐元素相乘并求和来计算输出特征图上每个位置的值。

卷积核的特征提取 卷积核在卷积神经网络中扮演着至关重要的角色, 它决定了网络能够提取什么样的特征。通过设计不同的卷积核, 我们可以使网络对输入图像进行各种转换和滤波操作, 从而获得丰富多样的特征图。这些特征图蕴含了图像在不同尺度、方向和抽象层次上的信息, 为后续的识别和分类任务提供了有力支持。

让我们来看一些具体的例子。图2.9展示了一张输入图像经过不同卷积核处理后得到的特征图。其中, 模糊 (Blur) 卷积核可以平滑图像, 消除高频噪声; 锐化 (Sharpen) 卷积核则可以增强图像的细节和边缘。这两种操作在传统的图像处理中非常常见, 而卷积神经网络则可以通过学习来自动优化这些卷积核的参数, 使其适应不同的任务需求。这与视觉皮层中简单细胞的特征提取功能非常类似。

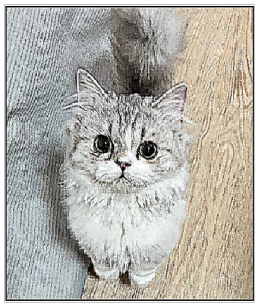
锐化	方框模糊 (归一化版)	高斯模糊 (概率形式)	边缘检测
$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$	$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$
			

图 2.9: 由不同卷积核得到的特征图

填充和跨步 在实际应用中, 我们往往需要调整卷积操作的一些细节, 以满足不同的需求。其中, 填充 (Padding) 和跨步 (Stride) 是两个常用的技巧。

我们观察如下卷积操作 (图 2.10)，如果只是用 3×3 的卷积核与 5×5 的图像做卷积，我们最终得到一个 3×3 的输出，相较原始图像尺寸变小了。

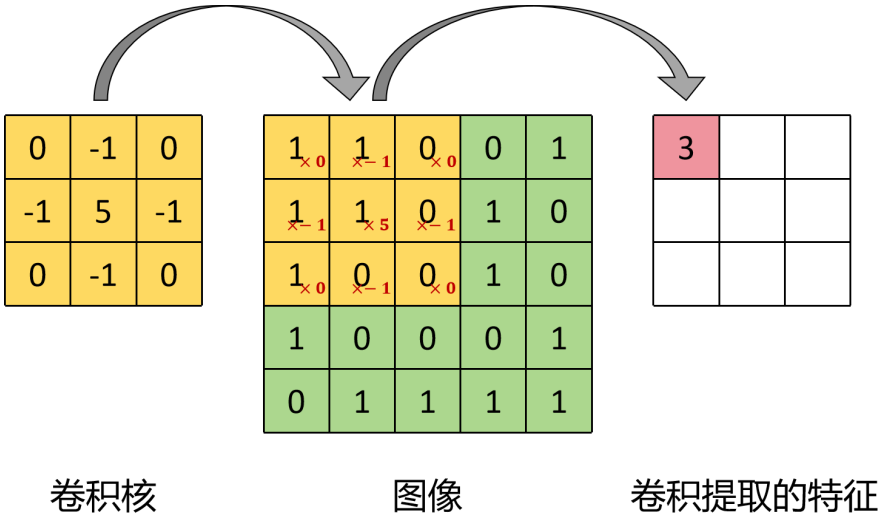


图 2.10: 卷积对图像的特征提取

有时我们并不希望图像在经过卷积后“缩水”。为了解决这个问题，我们引入 Zero Padding 操作，即在做卷积之前在原始图像周围补一些 0，来确保通过卷积以后输出大小尽量与原图像大小保持一致。图 2.11展示了一个 Zero Padding 的例子，原始图像为 4×4 的图像，与 3×3 的卷积核做卷积运算最终得到一个 2×2 的输出。在加入一层 zero padding（图中虚线边框部分）之后，图像由 4×4 变为 6×6，卷积运算的输出结果为 4×4。在加入两层 zero padding 之后，图像变为 8×8，卷积运算的输出结果为 6×6。这种做法有两个好处: 一是可以保留图像边缘的信息; 二是可以使网络的深度和感受野更容易控制。

注 8. *感受野（Receptive Field）指的是网络中某个神经元在输入图像中所能“看到”的区域或范围。它描述了输入图像的哪一部分会影响该神经元的激活。随着卷积层的叠加，感受野通常会增大，也就是说，网络的深层神经元可以“看到”输入图像中更大范围的信息。*

跨步（stride）则是指卷积核在输入特征图上滑动的步长。通过调整跨步, 我们可以控制卷积后输出特征图的尺寸, 同时减少计算量。较大的跨步可以使卷积核快速地扫过输入特征图, 提取更全局的信息; 较小的跨步则可以捕捉更多的局部细节。跨步的引入使卷积神经网络具有更大的灵活性, 能够适应不同尺度和分辨率的输入。图 2.12展示了一个跨步操作示例，5×5 的

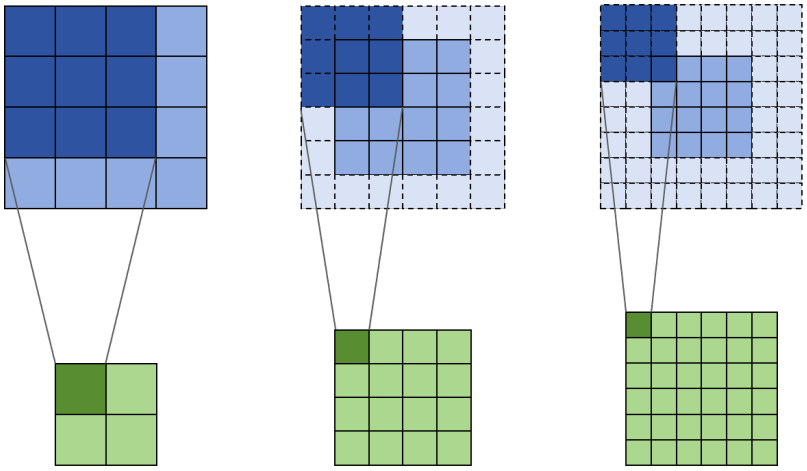


图 2.11: Zero padding 操作示例

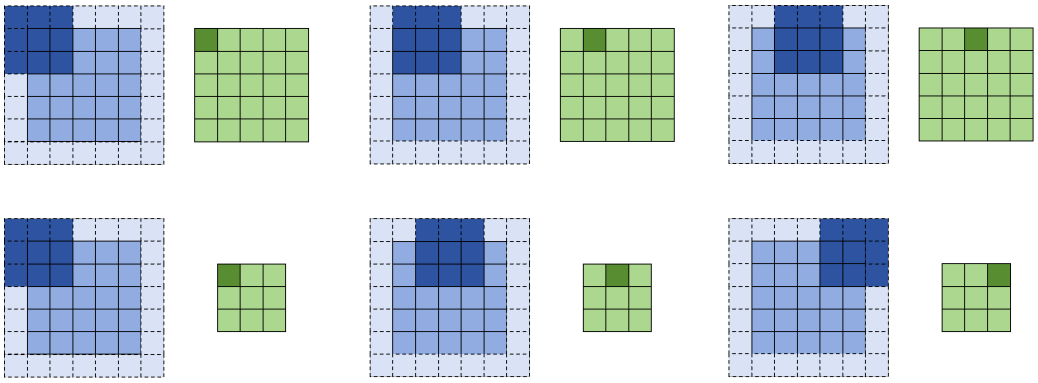


图 2.12: Stride 操作示例

原始图像通过 zero padding 操作变为 7×7 的图像, 用一个 3×3 的卷积核与它做卷积。第一行没有使用 stride，因此得到一个 5×5 的输出；第二行使用 stride(2,2)，即横向纵向跨步步长均为 2, 最终得到一个 3×3 的输出。

多通道卷积 在现实世界中, 图像往往不是单通道的灰度图, 而是多通道的彩色图。例如,RGB 彩色图像有红、绿、蓝三个通道。为了处理这种情况, 卷积神经网络引入了多通道卷积 (Multi-channel Convolution) 的概念。

多通道卷积的基本思想是, 对于每个输入通道, 使用一组独立的卷积核进行卷积运算, 然后将这些卷积结果求和，再加上一个偏置项 (Bias), 得到最终的输出特征图。这种设计使得网络能够提取输入图像中不同通道的特征, 并自动学习它们之间的相互作用。多通道卷积极大地丰

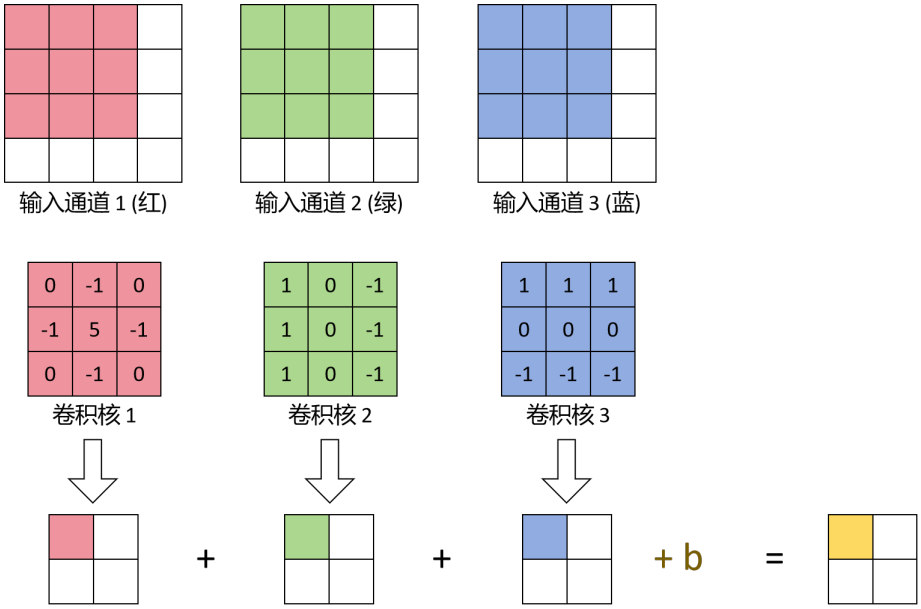


图 2.13: 多通道卷积示例图

富了卷积神经网络的表达能力, 使其能够处理更加复杂和多样化的视觉任务。在图 2.13所示的示例中, 分别用三个 3×3 的卷积核与它们分别做卷积, 在每个点上获得的数值相加再加上一个共同的 bias 项, 就得到了一个输出。因为我们每个神经元共享参数, 所以我们的偏置项 (bias) 是不变的, 该例中 bias 恒定为 b。在该卷积运算中,3 个卷积核提供了 $3\times3\times3=27$ 个参数, 加上 bias 项, 一共是 28 个参数。

卷积操作是卷积神经网络的核心, 它巧妙地利用了图像的局部相关性和平移不变性, 通过卷积核实现了高效的特征提取。填充、跨步和多通道卷积等技巧进一步增强了卷积操作的灵活性和适应性, 使卷积神经网络能够处理各种复杂的视觉任务。这些设计不仅源自传统的图像处理方法, 更借鉴了生物视觉系统的信息处理机制。可以说, 卷积操作是连接图像、生物视觉和人工智能的一座桥梁, 它为计算机视觉的发展开辟了一条充满希望的道路。

为什么要引入卷积？ 图像的空间相关性不仅影响到图像处理算法的设计, 也给机器学习模型的训练带来挑战。假设我们要学习一个判别模型, 来预测两个像素点是否相关。如果两个像素点是独立的, 根据大数定律, 我们需要大量的样本数据才能准确估计它们的独立性。具体来说, 假设两个像素点 x 和 y 相互独立, 且均值为 0, 方差为 1, 我们可以推导出它们的协方差的统计特性。

首先, 协方差的定义为:

$$\mathrm{cov}(x, y) = E((x - E(x))(y - E(y))) \tag{2.8}$$

由于 x 和 y 的均值为 0, 因此:

$$\mathrm{cov}(x, y) = E(xy) \tag{2.9}$$

对于 n 个样本, 协方差的估计为:

$$\mathrm{cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n} x_i y_i \tag{2.10}$$

由于 x 和 y 相互独立, 因此 $E(xy) = E(x)E(y) = 0$, 所以:

$$\mathbb{E}(\mathrm{cov}(x, y)) = 0 \tag{2.11}$$

协方差的方差为:

$$\begin{aligned}
\mathrm{Var}(\mathrm{cov}(x, y)) &= E((\mathrm{cov}(x, y) - E(\mathrm{cov}(x, y)))^2) \\
&= E((\frac{1}{n}\sum_{i=1}^{n} x_i y_i)^2) = \frac{1}{n^2} E((\sum_{i=1}^{n} x_i y_i)^2) \\
&= \frac{1}{n^2}\sum_{i=1}^{n} E(x_i^2 y_i^2) = \frac{1}{n^2}\sum_{i=1}^{n} E(x_i^2)E(y_i^2) \\
&= \frac{1}{n^2}\sum_{i=1}^{n} 1 \cdot 1 = \frac{1}{n}
\end{aligned}$$

因此, 协方差的标准差为:

$$\sqrt{\mathrm{Var}(\mathrm{cov}(x, y))} = \frac{1}{\sqrt{n}} \tag{2.12}$$

可以看出, 协方差的均值为 0, 而标准差与样本数的平方根成反比。这意味着, 假如我们要用算法把两个像素点协方差的标准差识别到 10^{-3} 以下来说明其没有关联, 那么需要 10^6 量级的数据点才能做到。这对于数据的采集和模型的训练都提出了很高的要求。

为了进一步说明卷积的重要性, 我们做了一个识别局部关联的实验。实验中, 输入 $\mathbf{x}$ 是一个 100 维的向量, 每个维度的值都从 $[0, 1]$ 中均匀采样, 输出 y 是 $\mathbf{x}$ 前 10 个维度的求和:

$$y = \sum_{i=1}^{10} x_i, \tag{2.13}$$

这样设计的目的是使得输出 y 只与输入 x 的一个局部有关, 使数据集满足局部关联假设。

假设在训练数据集中存在一点噪音，即 $y_{\text{train}} = \sum_{i=1}^{10} x_i + \mathcal{N}(0, 0.01)$。我们分别用全连接网络（DNN）和在全连接层前加了一个大小和跨度均为 5 的一维卷积的神经网络（conv+DNN）来拟合这个函数。对于带有卷积层的神经网络，100 维的输入向量在经过卷积后会变成一个 20 维的向量，再经过全连接网络来输出结果。在本次实验中，我们重点关注数据量不充分且带有噪音的情形，因此训练集大小设置为 50。图2.14展示了实验结果。可以看出，当数据量不充分且带有噪音时，全连接网络会使用另外 90 个无关的数据维度去拟合训练数据中的噪音，因此在测试集上没有泛化性。但对于带有卷积层的神经网络，得益于卷积可以提取局部信息，因此神经网络几乎没有学到噪音，在测试集上也有很好的泛化性。

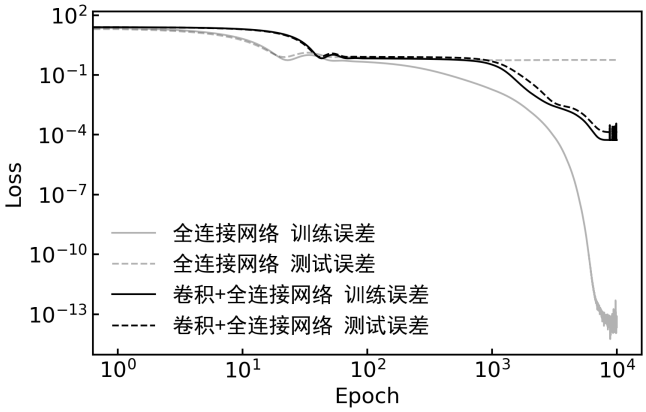


图 2.14: 全连接网络与带卷积的神经网络拟合具有局部关联性质且带噪音的数据集的表现

这个实验直观地展示了卷积在处理具有局部相关性的数据时的优势。在图像识别任务中，卷积神经网络之所以能取得巨大成功，很大程度上就是因为卷积层可以有效地利用图像的空间相关性，从而大大减少了模型的参数数量和计算复杂度。

此外，卷积还赋予了网络对于平移等变的性质，即输入与输出以同样的方式改变。假设图像为 $f(x, y)$，g 是平移变换：$g(f(x, y)) = f(x - a, y - b)$。conv 是跨度为 1 的卷积运算，则有：

$$\text{conv}(g(f(x, y))) = g(\text{conv}(f(x, y))). \tag{2.14}$$

这个等式表明，如果我们先对图像进行平移变换，再做卷积运算，其结果与先做卷积运算再进行平移变换是相同的。这一性质源自卷积的权值共享 (Weight Sharing) 机制：卷积核在图像上滑动，以相同的方式提取不同位置的局部特征，因此无论物体出现在图像的哪个位置，都能被有效地捕捉到。

这种平移等变性使得卷积神经网络在图像识别等任务上表现出色。传统的全连接网络需要单独学习每个像素位置的权重，这不仅参数量巨大，而且难以泛化到新的位置。相比之下，卷

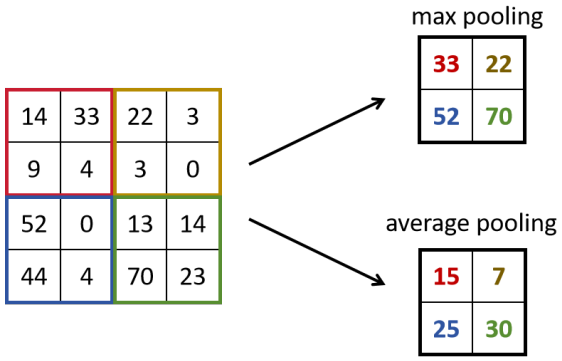


图 2.15: 池化操作示意图

积神经网络通过权值共享, 大大减少了参数数量, 提高了学习效率; 同时, 由于具备平移等变性, 模型可以很好地适应物体位置的变化, 从而获得更强的泛化能力。

综上所述, 卷积赋予了神经网络重要的先验知识——局部连接、权值共享、平移等变性等, 使其能够高效地学习视觉模式, 在图像相关任务上取得了巨大的成功。这些先验知识减轻了数据的学习负担, 加速了训练过程, 提升了模型的泛化性能。

池化 Pooling

对于分辨率很高的图片，在完成卷积运算后，得到的输出对于计算机而言可能还是很大，为了提升计算的效率, 我们可以做适当的下采样, 这里我们引入一个新的操作: 池化 (pooling)。池化分为最大值池化（max pooling）和平均值池化（average pooling）两种，最大值池化将选中区域中最大值提取出来作为输出，平均值池化将选中区域中所有值的平均作为输出。如图 2.15所示，输入为 4×4 图像，将其划分为 4 个 2×2 的小块，max pooling 将每一块中的最大值作为输出，average pooling 将每一块中的平均值作为输出，两种 pooling 方式均得到 2×2 的输出。

为了计算需要，我们人为定义池化按照如下方式进行梯度计算:

- 对于最大值池化，我们记录最大值点的位置，在梯度反向传播的过程中，我们把梯度全部传递到该位置，即仅对该位置做梯度下降，其他点保持不变。
- 对于平均值池化，在梯度反向传播的过程中，梯度会平均分配到每一个单元来进行梯度下降。

图 2.16详细描述了该过程。对于 max pooling（左），图中最大值点位置在右上角，因此反向传播的梯度将全部用来更新右上角处的参数；对于 average pooling（右），每个单元将分配

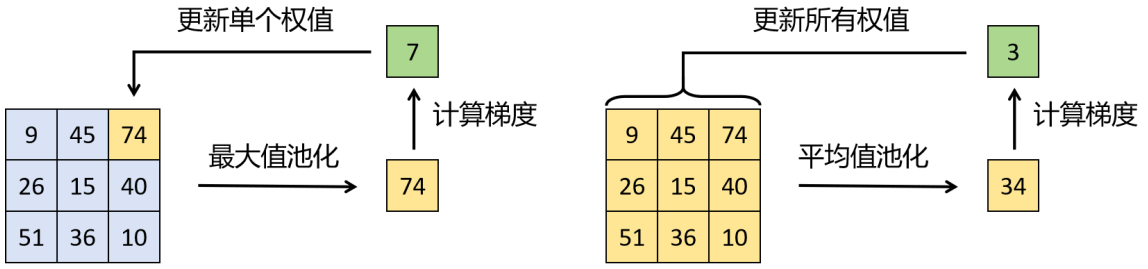


图 2.16: 池化操作的梯度传播方式

到反向传播梯度的 1/9。

卷积神经网络的整体架构

有了卷积和池化这两个核心操作, 我们就可以搭建出一个完整的卷积神经网络了。CNN 的整体架构可以看作是卷积层、池化层和全连接层的有序组合, 再加上一些非线性激活函数 (如 ReLU) 和归一化操作 (如 Softmax)。

让我们以一个具体的例子来说明 CNN 的工作流程, 如图2.17所示。假设我们的输入是一张 RGB 彩色图像, 它有红、绿、蓝三个通道。

首先, 我们用三个卷积核分别对三个输入通道进行卷积操作, 然后将结果相加并加上一个偏置项, 得到第一个卷积层的一个输出通道。接着, 我们再用另外三个卷积核对原始输入做卷积, 得到第一个卷积层的第二个输出通道。以此类推, 我们最终得到第一个卷积层的一个多通道输出。

然后, 这个多通道输出逐元素经过一个非线性激活函数 (如 ReLU), 再输入到第一个池化层。池化操作不改变输出的通道数, 但会减小每个通道的尺寸。假设第一个池化层的输出有 n 个通道。

接下来, 我们用 m 组卷积核 (每组 n 个) 对第一个池化层的输出进行卷积, 得到第二个卷积层的输出, 共 m 个通道。这个输出再经过 ReLU 和第二个池化层, 得到一个通道数为 m、尺寸更小的特征图。

我们重复这种“卷积-ReLU-池化”的模式多次, 得到一系列尺寸逐渐减小、通道数逐渐增多的特征图。这种层次化的结构使 CNN 能够逐步提取图像的特征, 从低级的边缘、纹理到高级的物体部件和场景, 形成一个由局部到全局、由简单到复杂的特征层次。同时, 卷积和池化操作的引入也大大减少了网络的参数数量和计算复杂度, 使得 CNN 能够处理大尺寸、高分辨率的图像输入。

最后, 我们将最后一个池化层的输出“展平”(Flatten), 转化为一个一维向量, 输入到一个或多个全连接层。全连接层可以看作是一个传统的全连接网络, 它对卷积层提取的特征进行非

线性组合, 生成更加紧凑和抽象的表示。

在 CNN 的输出端, 我们通常使用一个 Softmax 层将全连接层的输出转化为一个概率分布。这个分布表示输入图像属于各个类别的可能性, 其中概率最大的类别即为 CNN 的预测结果。

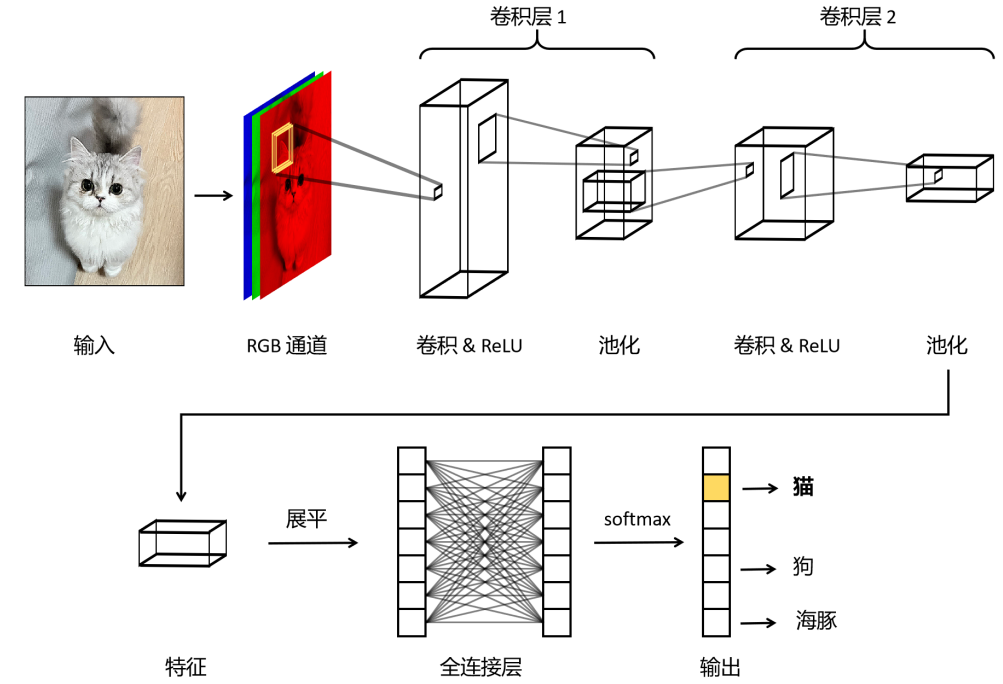


图 2.17: 卷积神经网络的架构

CNN 的成功证明了局部连接、权值共享、池化降采样等结构的有效性, 它们不仅大大降低了网络的复杂度, 也与视觉系统的特点高度吻合。可以说,CNN 是深度学习和计算机视觉的一次完美结合, 它开启了人工智能的新纪元, 让机器拥有了”看”的能力。随着 CNN 的不断发展, 我们有理由相信, 计算机视觉将在越来越多的领域发挥重要作用, 并最终达到甚至超越人类的水平。

2.4 循环神经网络

语言任务是人工智能的一个重要研究领域, 其目标是让机器能够理解、生成和处理人类语言。相比于图像处理任务, 语言任务具有其独特的特点和挑战。以下列举了语言任务的一些特点:

- 语言的构成具有复杂的层次结构。以词汇为例，英语单词“unbelievable”可以被分解为词根“believe”、前缀“un-”和后缀“-able”，分别表示“相信”、“不”和“可……的”等含义。这些语素按照一定规则组合，共同构成了该词所要表达的完整语义。在更高层面上，词汇组成短语，短语构成句子，句子进一步组成段落和篇章。每一个层级都有其独特的组织规律和约束条件。

- 语言遵循严格的语法规则。例如，一个完整的陈述句必须包含主语和谓语，二者缺一不可。此外，句子往往还存在宾语、状语、表语等。诸如词序、句式等语法规则决定了句子成分的排列方式和组合规律，使得语言表达遵循相对固定的模式，保证了交流的顺畅和意义的明确。

- 除了基本句式，复杂句的构成也体现出语言的递归性。所谓递归，指的是某一规则可以重复应用于自身输出的过程。例如，名词从句可以充当另一个句子的主语、宾语或表语，构成更为复杂的句式。“我相信他昨天工作到很晚。”一句中，“他昨天工作到很晚”就是一个作宾语的名词从句，而该从句本身也是一个主谓结构完整的句子。理论上，名词从句可以无限嵌套，生成无穷多的递归结构，这体现了语言的生成能力和表现力。

- 语言数据是离散的符号序列，而非连续的像素值。每个符号 (如单词或字符) 都属于一个大小有限的词汇表，但它们的组合方式却是几乎无穷无尽的。这种组合爆炸性使得语言模型需要具备强大的泛化能力，能够处理未曾见过的新句子。

- 语言具有长程依赖关系。句子中的某个词可能与相隔很远的另一个词存在依赖关系，这种长程依赖给语言建模带来了挑战。传统的前馈神经网络和卷积神经网络难以捕捉这种长程交互。

- 语言还具有强烈的上下文相关性。同一个词在不同的上下文中可能具有完全不同的意思。例如，”交通”在”我爱上海交通大学”和”我不爱上海的交通”中的含义显然不同。这就要求语言模型能够动态地根据上下文来理解词义和句意。

传统的全连接神经网络难以很好地应对这些语言特性带来的挑战。它们将所有的词向量拼接成一个超长向量，再经过多层的矩阵乘法，这种结构无法捕捉词与词之间的交互和顺序信息。此外，传统的全连接神经网络对输入的数据具有形状一致性的要求，然而语言序列的长度是可变的，将所有词向量拼接成定长向量也存在信息损失的问题。

为了应对这些语言特性带来的挑战，研究者提出了循环神经网络 (Recurrent Neural Network, RNN)。通过引入循环连接的方式，RNN 能够在神经网络内部构建状态记忆，并在计算每个时间步的输出时，融合来自当前输入和前一状态的信息，从而自然地捕捉和利用了序列数据所蕴含的动态行为和一定程度的长程依赖关系，展现出卓越的序列建模能力。

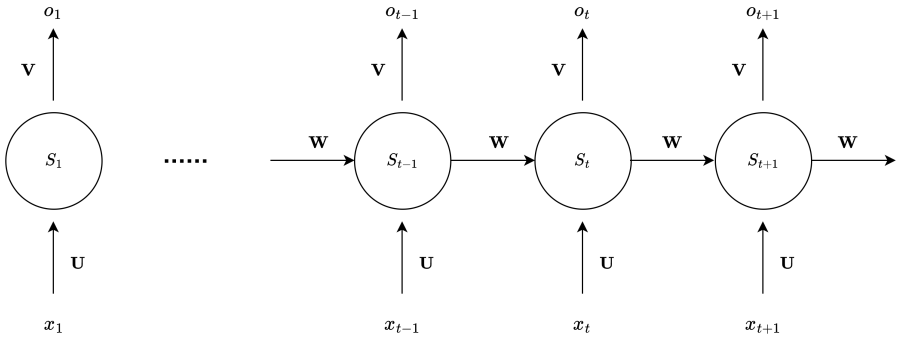


图 2.18: 循环神经网络的基本结构

2.4.1 循环神经网络基本单元

如图2.18所示, 这是一个基本的循环神经网络单元, 下面我们详细介绍一下这个单元是如何工作的。我们的输入为时序序列 $x_1, x_2, \cdots, x_{t-1}, x_t, x_{t+1}, \cdots, x_n$, 并通过公式(2.15),(2.16)和(2.17)的方式计算得到对应的输出序列 $o_1, o_2, \cdots, o_{t-1}, o_t, o_{t+1}, \cdots, o_n$。

$$S_0 = 0, \tag{2.15}$$

$$S_t = f(\mathbf{U} \cdot x_t + \mathbf{W} \cdot S_{t-1} + b), \quad t = 1, 2, 3, \cdots, n, \tag{2.16}$$

$$o_t = g(\mathbf{V} \cdot S_t) \tag{2.17}$$

其中 f 是激活函数，通常使用 Tanh 函数，$g = \text{Softmax}(x)$ 是 softmax 函数。S_t 是状态记忆，$\mathbf{U}$，$\mathbf{W}$，$\mathbf{V}$ 和 b 分别是输入权重矩阵、循环权重矩阵、输出权重矩阵和偏置项。

对于一个多层循环神经网络，假设共有 L 层，每一层的隐藏状态和输出会作为下一层的输入进行计算，具体的数学表达式如下：

$$S_t^{(1)} = f(\mathbf{U}^{(1)} \cdot x_t + \mathbf{W}^{(1)} \cdot S_{t-1}^{(1)} + b^{(1)}), \quad t = 1, 2, 3, \cdots, n, \tag{2.18}$$

$$o_t^{(1)} = g(\mathbf{V}^{(1)} \cdot S_t^{(1)}), \tag{2.19}$$

第 l 层的输入是前一层的输出 $o_1^{(l-1)}, o_2^{(l-1)}, \cdots, o_n^{(l-1)}$，并计算该层的隐藏状态和输出：

$$S_t^{(l)} = f(\mathbf{U}^{(l)} \cdot o_t^{(l-1)} + \mathbf{W}^{(l)} \cdot S_{t-1}^{(l)} + b^{(l)}), \quad t = 1, 2, 3, \cdots, n, \tag{2.20}$$

$$o_t^{(l)} = g(\mathbf{V}^{(l)} \cdot S_t^{(l)}), \tag{2.21}$$

其中 $S_0^{(l)} = 0$, $l = 1, \cdots, L$，最后一层的输出序列即为最终的输出 $o_1^{(L)}, o_2^{(L)}, \cdots, o_n^{(L)}$。

以上为最基本的循环神经网络架构，可以看到，它的输入输出序列长度是相同的，这种结构比较适合输入输出长度一致的任务，例如视频帧的逐帧分类、时间序列的预测等。

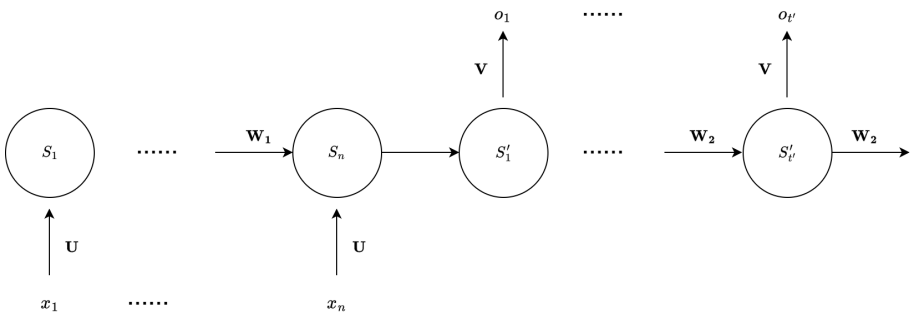


图 2.19: encoder-decoder 架构的循环神经网络

2.4.2 Encoder-Decoder 架构的循环神经网络

前文我们介绍的循环神经网络架构的输入和输出是等长的，这也带来了诸多限制。通常而言，对于像机器翻译这样输入和输出不等长的任务，使用更加广泛的是它带有编码-解码结构的变体：

$$S_0 = 0, \tag{2.22}$$

$$S_t = f_1(\mathbf{U} \cdot x_t + \mathbf{W}_1 \cdot S_{t-1} + b), t = 1, 2, 3, \cdots, n, \tag{2.23}$$

$$S'_0 = S_t \tag{2.24}$$

$$S'_t = f_2(\mathbf{W}_2 \cdot S'_{t-1} + b), \quad t = 1, 2, 3, \cdots, n', \tag{2.25}$$

$$o_t = g(\mathbf{V} \cdot S'_t) \tag{2.26}$$

以一个翻译任务为例: 我们需要将“我喜欢吃苹果”翻译为“I like eating apples”。我们通过编码方式将“我喜欢吃苹果”逐字编码为 $x_1 \sim x_6$, 将“I like eating apples”逐词编码为 $o_1^* \sim o_4^*$。我们会预先指定一个输出最大句长，假设这里最大句长为 7，于是我们的模型输入为 $x_1, x_2, x_3, x_4, x_5, x_6$，标签为 $o_1^*, o_2^*, o_3^*, o_4^*, 0, 0, 0$（标签不足 7 则补 0)。通过上述基本单元的计算, 我们可以得到 $o_1, o_2, o_3, o_4, o_5, o_6, o_7$, 并计算其与标签的交叉熵损失，然后通过反向传播方法更新权重。

注 9. *循环神经网络也可以做回归任务，此时不需要加 Softmax 函数 g。*

2.4.3 使用 BPTT 算法训练循环神经网络

循环神经网络同样使用梯度下降法和反向传播进行训练，但因为其形式比较特殊，o_t 的计算涉及到前 $t-1$ 步的信息，所以计算梯度时也会涉及到多个时间步，我们将其称为随时间反

向传播 BPTT（BackPropagation Through Time, BPTT），具体而言，以循环神经网络基本架构为例，给定一个损失函数，

$$L = \sum_{t=1}^{n} L_t(o_t, o_t^*),$$

BPTT 首先将损失 L 对当前时间步 t 的状态 S_t、输出 o_t 等变量的梯度通过链式法则逐步传递回前面每一个时间步，从而更新所有时间步上的权重。

对于第 t 步的损失 L_t，我们可以通过链式法则依次计算其关于权重 $\mathbf{U}$、$\mathbf{W}$、$\mathbf{V}$ 和偏置 b 的梯度。因为 S_t 依赖于前一个时间步的状态 S_{t-1}，这意味着要通过反向传播回传每个时间步的误差。

具体来说，梯度计算公式如下：

$$\frac{\partial L_t}{\partial \mathbf{V}} = \frac{\partial L_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial \mathbf{V}} = \delta_t \cdot S_t^T$$

$$\frac{\partial L_t}{\partial S_t} = \delta_t \cdot \mathbf{V}$$

$$\frac{\partial L_t}{\partial \mathbf{W}} = \sum_{k=1}^{t} \left(\frac{\partial L_t}{\partial S_t} \cdot \frac{\partial S_t}{\partial S_{t-k}} \right) \cdot \frac{\partial S_{t-k}}{\partial \mathbf{W}}$$

$$\frac{\partial L_t}{\partial \mathbf{U}} = \sum_{k=1}^{t} \left(\frac{\partial L_t}{\partial S_t} \cdot \frac{\partial S_t}{\partial S_{t-k}} \right) \cdot \frac{\partial x_k}{\partial \mathbf{U}}$$

其中，$\delta_t = \frac{\partial L_t}{\partial o_t} \cdot g'(\mathbf{V} \cdot S_t)$。

但由于循环神经网络的递归结构，状态 S_t 的梯度要经过多个时间步的链式计算，导致梯度在长时间依赖时逐渐衰减或急剧增大，导致梯度消失和梯度爆炸，尤其在较长序列中更为严重，导致模型难以学习到长期依赖。并且 BPTT 需要在每个时间步都反向传播梯度，并且每个时间步的状态依赖于前面的状态，因此每个参数更新的计算开销较大，尤其在处理长序列时，训练变得更加缓慢。

为了解决这些问题，研究者提出了长短时记忆网络（LSTM）和门控循环单元（GRU），它们通过引入门控机制有效缓解了梯度消失问题，从而能够更好地处理长时间依赖的任务。

2.4.4 长短程记忆网络

长短期记忆网络（Long short-term memory, LSTM）是一类特殊的 RNN，由 Schmidhuber 和 Hochreiter 在 1997 年提出，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。相比普通的循环神经网络，LSTM 能够在更长的序列中有更好的表现。

人类在阅读文章的时候并不会每分每秒都重新从头思考，看到的知识并不会立即遗忘而是逐渐加深记忆过程，可见记忆信息在人类思考与反馈过程中非常重要，循环神经网络通过传递

隐藏状态来储存这些记忆信息。但是人类的记忆并非一成不变，而是在阅读新内容时候，不断更新新知识遗忘旧知识，因此记忆信息的获取和遗忘是人类有效进行学习的重要因素。但是如公式 (2.16)所示，循环神经网络的每层都会使用参数矩阵 **U** 和 **W** 来处理新的输入信息和记忆信息，但是这样的结构过于简单，缺乏长程信息之间的交互，也没有遗忘信息的过程。

下面我们正式介绍 LSTM 的主要架构，如同循环神经网络那样，LSTM 选择了将相同的子模块堆积形成整个网络结构，只不过与循环神经网络的简单模块不同，LSTM 的子模块（称为门控记忆元）结构较为复杂。它主要由三个不同的门控构成：输入门、遗忘门和输出门，同时除了与循环神经网络相同的隐状态 S_t 之外增加了一个记忆状态的通道 C_t (cell state)，构成了信息流动的双路并行，而三个门控将用于保护和利用记忆状态以补充隐状态。图 2.20是 LSTM 的一个示意图，接下来讲详细介绍这三类门的作用。我们约定本节中出现的 σ 为 sigmoid 函数。

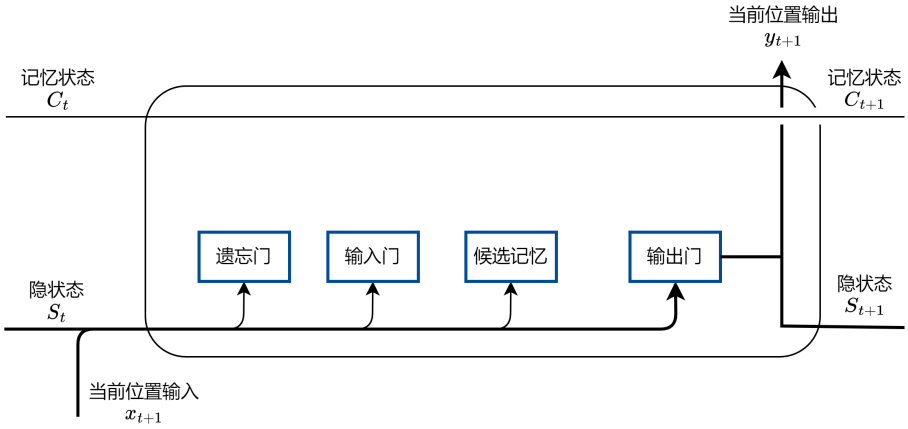


图 2.20: 单个门控神经元主要部件和并行隐藏状态示意图

门控神经元-遗忘门

在信息输入到 LSTM 的第一步过程中，网络需要判断 C_t 中哪些记忆需要遗忘。遗忘门是一个 sigmoid 网络并同时接受隐状态 S_t 和当前位置输入 x_{t+1} 的信息。由于 sigmoid 的返回值是介于 0 到 1 之间的数字，因此其含义是表示记忆状态 C_t 中多大部分应该被保留下来。如果定义 f_t 作为这一步的记忆保存比例，有以下公式成立：

$$f_t = \sigma(W_{Sf}S_t + W_{xf}x_{t+1} + b_f). \tag{2.27}$$

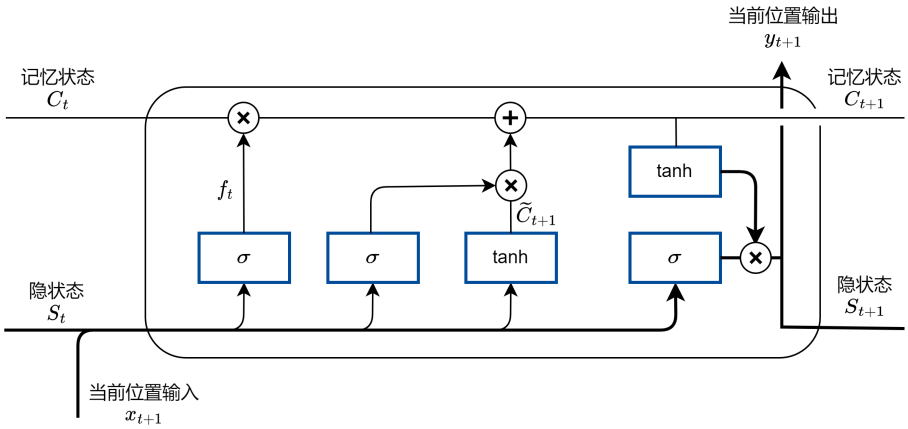


图 2.21: 单个门控神经元结构示意图

门控神经元-输入门

第二步过程中，门控神经元需要判断输入的哪一部分是需要记忆的，并在记忆状态中更新。这一步同时要分为两个部分，第一部分由输入门决定更新哪些记忆值，第二步由候选记忆提供一个新的记忆状态 $\tilde{C}_t$，并接受输入门的控制。最后将整个候选记忆直接添加到记忆状态中。对应的公式如下：

$$i_t = \sigma(W_{Si}S_t + W_{xi}x_{t+1} + b_i), \tag{2.28}$$

$$\tilde{C}_{t+1} = \tanh(W_{SC}S_t + W_{xC}x_{t+1} + b_C), \tag{2.29}$$

$$C_{t+1} = f_t * C_t + i_t * \tilde{C}_{t+1}, \tag{2.30}$$

其中，$*$ 表示按元素相乘。

门控神经元-输出门

最后一步我们需要决定当前的输出，而此时我们的记忆状态已经完全更新为了最新的状态。通过上一步的隐状态和当前输入，经过一个 sigmoid 层决定输出记忆中的哪些部分，记忆状态通过一个 tanh 同样归一化到 -1 至 1 并乘以 sigmoid 层的输出作为最后的当前位置输出 y_{t+1}。对应的公式如下：

$$o_t = \sigma(W_{So}S_t + W_{xo}x_{t+1} + b_o), \tag{2.31}$$

$$y_{t+1} = S_{t+1} = o_t * \tanh(C_{t+1}). \tag{2.32}$$

LSTM 解决了 RNN 中部分梯度消失问题

从数学角度来看，LSTM 能够解决 RNN 中的梯度爆炸或消失问题，关键在于它通过记忆状态 C_t 和门控机制设计来稳定梯度的传播。

在 RNN 中，梯度通过时间步长反向传播，更新时依赖于每个时间步的 Jacobian 矩阵。假设损失函数 L 对于第 t 时刻的隐状态 S_t 的梯度为 $\frac{\partial L}{\partial S_t}$，RNN 的链式法则会导致该梯度的计算为：

$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial S_{t+1}} \cdot \frac{\partial S_{t+1}}{\partial S_t}$$

由于 S_{t+1} 是通过某个激活函数（例如 tanh 或 sigmoid）与权重矩阵计算得到的，Jacobian 矩阵 $\frac{\partial S_{t+1}}{\partial S_t}$ 的特征值可能小于 1（导致梯度消失）或大于 1（导致梯度爆炸）。随着时间步数增加，链式相乘的梯度会迅速衰减或膨胀。

LSTM 引入了一个记忆状态 C_t，并且通过门控机制来控制信息的传递。根据记忆状态的更新公式：

$$C_{t+1} = f_t * C_t + i_t * \tilde{C}_{t+1},$$

我们有：

$$\frac{\partial L}{\partial C_t} = \frac{\partial L}{\partial C_{t+1}} \cdot f_{t+1}.$$

由于遗忘门 f_t 的输出值通常在 0 和 1 之间，因此通过门控机制，梯度会被控制在合理的范围内，不会像 RNN 那样指数级增长或衰减。这种常量误差流使得梯度在多个时间步长上传递时，衰减得较慢，从而缓解了梯度消失的问题。

2.5 Transformer

在前文中我们提到,RNN 存在诸多局限性, 尽管后来在 RNN 框架的基础上又发展了许多具有创新性的算法, 但仍然无法完全克服 RNN 的缺陷以及语言问题本身的困难。在此背景下,Transformer 应运而生。Transformer 的诞生极大地推动了自然语言处理领域的发展, 其中最具代表性的开篇工作是论文《Attention is all you need》Vaswani et al. (2017)。这篇论文首次提出了 Transformer 模型, 它抛弃了 RNN 的思想, 完全依靠注意力机制 (Attention Mechanism) 来建模序列。注意力机制最早由 Bahdanau 等人Bahdanau et al. (2014) 在机器翻译任务中引入, 后来被广泛应用于各种自然语言处理任务。在此基础上, 林洲汉等人Lin et al. (2017) 进一步提出了结构化的自注意力机制 (Self-Attention), 允许序列中的任意两个位置直接发生交互, 无论它们之间的距离有多远。Transformer 模型正是建立在这种结构化自注意力机制之上的。这种结构天然地具有捕捉长程依赖的能力。同时, 自注意力计算也是高度并行的, 大大提高了模型的训练速度。

最初的 Transformer 由若干个编码器 (Encoder) 和解码器 (Decoder) 组成, 编码器是对输入序列进行特征提取，从而转换为一个固定长度的内部表示；解码器基于编码器的内部表示和其他已知内容生成目标（预测）内容, 每个编码器和解码器内部都使用了自注意力和前馈神经网络。值得注意的是, 编码器在计算自注意力时没有使用掩码 (Mask), 因此能够看到整个输入序列的全局信息和因果关系。这些全局信息被编码到上下文向量中, 并传递给解码器。解码器则根据上下文向量和之前生成的输出序列, 预测下一个输出词。编码器捕捉输入序列的全局语义, 解码器利用这些语义信息生成输出, 两者通力合作, 完成端到端的序列转换任务。Transformer 的这种编码器-解码器架构最初主要应用于机器翻译任务, 其中输入是源语言序列（原文）, 输出是目标语言序列（译文）, 两者是不对称的, 因此很自然地契合编码器-解码器的设计理念。

Transformer 的输入和输出长度是可变的, 不需要固定。以机器翻译任务为例, 源语言句子和目标语言句子的长度通常是不同的,Transformer 能够很好地处理这种情况。这得益于其独特的位置编码 (Positional Encoding) 机制, 它为每个位置赋予一个独特的向量, 使得模型能够区分不同位置的词, 即使输入长度发生变化也不会影响其表示能力。

然而, 随着研究的深入, 人们发现 Transformer 的潜力远不止于机器翻译。在许多其他的自然语言处理任务中, 如语言模型、文本摘要、对话生成等, 输入和输出并非完全不对称。以语言模型为例, 它的目标是根据之前的词预测下一个词, 输入和输出包含相同的语言种类。因此研究者们提出了仅解码器 (Decoder-Only) 的 Transformer 变体。在这种架构下, 模型只有解码器, 没有编码器。

现有的大多数大型语言模型, 如 GPT 系列, 其核心架构都是采用了如图2.22所示的 Decoder-Only 模型结构。在这种 Decoder-Only 架构中, 模型的输入首先会通过嵌入层映射到嵌入空间（embedding space）, 获得嵌入表示。之后, 这些嵌入表示将被输入到由多个解码器（Decoder layer）组成的 Transformer 模块中。在上述计算过程中, 残差连接和层归一化也被广泛应用, 以提升模型的训练稳定性。最终,Transformer 会将最后一层解码器的输出映射回词汇空间, 生成对应的输出序列概率分布。通过这种全新的结构设计,Transformer 架构展现出了卓越的并行能力和长程关联的建模能力, 从而在诸多序列相关任务中取得了突破性的进展, 推动了人工智能的发展。据说，OpenAI 在使用 Transformer 的架构后，两个礼拜取得的进展超过了用 RNN 两年的研究进展。一些理论工作也指出，Transformer 对于序列形式的数据具有强大的拟合能力 (Wang et al. 2024)，面对长记忆工作不会像 RNN 那样面临记忆灾难。

Transformer 的提出是自然语言处理领域的一个里程碑。它为后续的语言模型如 BERT(Devlin et al. 2018) 和 GPT(Radford et al. 2018) 奠定了基础。这些模型在大规模语料库上进行预训练, 学习到了丰富的语言知识, 在下游任务上表现出了惊人的性能, 甚至在某些任务上超越了人类。可以说,Transformer 引领了自然语言处理的一场革命, 使得语言理解和生成能力大幅提升。

在本节中, 我们将以一个 Decoder-Only 模型为例, 按照模块来详细介绍数据在 Transformer 中是如何被处理和计算的。

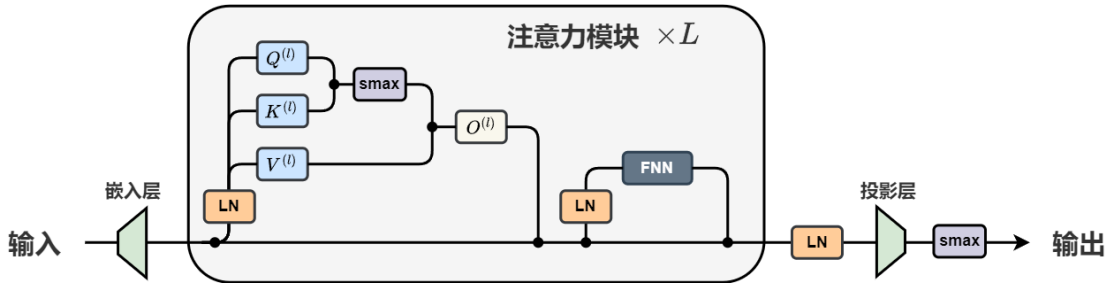


图 2.22: Decoder-Only Transformer 模型示意图

2.5.1 Transformer 的基本原理

在深入探讨 Transformer 模型的内部结构之前, 我们先来了解一下 Transformer 的基本原理。Transformer 模型本质上是一个自回归语言模型, 其基本任务是根据前 n 个词预测第 $n+1$ 个词, 这也被称为 next token prediction。

具体来说, 假设我们有一个长度为 n 的输入序列 $X^{\text{in}} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)^{\mathsf{T}} \in \mathbb{R}^{n\times d}$, 其中 $\boldsymbol{x}_i \in \mathbb{R}^d$ 表示第 i 个词的 d 维 one-hot 表征向量，d 为字典大小，该向量仅在该单词在字典中对应的位置为 1，其他位置为 0。Transformer 模型的目标是根据 X^{in} 预测下一个词 $\boldsymbol{x}_{n+1}$。

注 10. *某种意义上 one-hot 向量也可以理解为一个概率分布，即以 100% 的概率输出字典中某个对应的词，以 0 概率输出其他词。*

模型的输出也是一个 $n \times d$ 维的矩阵 $X^{\text{out}} \in \mathbb{R}^{n\times d}$, 其中第 i 行 X_i^{out} 的目标是预测输入的第 $i+1$ 个词。对于训练好的模型，应当有 $X_i^{\text{out}} \approx \boldsymbol{x}_{i+1}$。我们可以将 X_i^{out} 通过一个线性变换和 softmax 函数, 转化为在词汇表上的概率分布, 然后选择概率最大的词作为第 $i+1$ 个位置的预测结果。

举个例子, 假设我们的输入序列是“我爱小猫”, 对应的输入矩阵为:

$$X^{\text{in}} = \begin{bmatrix} \mathbf{x}_{\text{我}} \\ \mathbf{x}_{\text{爱}} \\ \mathbf{x}_{\text{小}} \\ \mathbf{x}_{\text{猫}} \end{bmatrix} \in \mathbb{R}^{4\times d}.$$

那么对于 next token prediction，我们期望 Transformer 模型的输出是:

$$X^{\text{out}} = \begin{bmatrix} \mathbf{x}_{\text{爱}} \\ \mathbf{x}_{\text{小}} \\ \mathbf{x}_{\text{猫}} \\ \mathbf{x}_{!} \end{bmatrix} \in \mathbb{R}^{4\times d},$$

其中 $\mathbf{x}_{!}$ 表示模型预测的下一个词是感叹号"!"。

在推断过程中，Transformer 模型根据输入的信息做 next token prediction，每次生成一个新的 token 并将其跟输入信息合并在一起继续放到模型中做 next token prediction，直到生成一个代表终止符的向量 $\mathbf{x}_{\text{EOS}}$ 时终止。此时得到的所有输出即为 Transformer 模型的推断结果。在我们了解 Transformer 模型具体架构后，我们将在2.5.7小节给出一个更为具体的例子来详细阐述 Transformer 模型做推断的流程。

在训练过程中,Transformer 模型对整个句子的每个位置分别计算交叉熵损失，然后取平均值作为最终的损失函数。具体而言，假设输入 $X^{\text{in}} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)^{\mathsf{T}}$ 的目标输出序列为 $X^{\text{out}} = (\boldsymbol{x}_2, \boldsymbol{x}_3, ..., \boldsymbol{x}_{n+1})^{\mathsf{T}}$，模型的预测输出为 $\hat{X}^{\text{out}} = (\hat{\boldsymbol{x}}_2, \hat{\boldsymbol{x}}_3, ..., \hat{\boldsymbol{x}}_{n+1})^{\mathsf{T}}$，其中 $\hat{\boldsymbol{x}}_{i+1}$ 是模型在第 i 个位置的预测概率分布。那么，整个序列的交叉熵损失可以表示为:

$$\mathcal{L}(X, \hat{X}) = -\frac{1}{n}\sum_{i=2}^{n+1}\sum_{j=1}^{d} \boldsymbol{x}_{ij} \log \hat{\boldsymbol{x}}_{ij},$$

其中 $\boldsymbol{x}_{ij}$ 是 $\boldsymbol{x}_i$ 的第 j 个元素 (0 或 1),$\hat{\boldsymbol{x}}_{ij}$ 是 $\hat{\boldsymbol{x}}_i$ 的第 j 个元素 (预测概率)。

模型的目标是最小化这个损失函数，从而使预测结果尽可能接近真实结果。这种对整个序列的每个位置分别计算损失的方法，可以更准确地衡量模型在每个位置的预测质量，并为模型提供更细粒度的监督信号。

注 11. *Transformer 模型这种对每个位置分别计算交叉熵损失的思想，可以看作对一个语料同时做了多次 next token prediction，例如，"我爱小猫!" 这个语料可以看作是 ("我") 预测 "爱"; ("我", "爱") 预测 "小"; ("我", "爱", "小") 预测 "猫"; ("我", "爱", "小", "猫") 预测 "!"。这样，相当于一个语料被使用了多次，极大地提高了训练效率和语料的利用率。*

以上就是 Transformer 模型的基本框架思想。在接下来的小节中，我们将详细介绍 Transformer 模型的内部结构，看看它是如何实现 next token prediction 预测任务的。

2.5.2 分词与特殊字符

在介绍 Transformer 模型的输入表示之前，我们需要先对输入的文本数据进行一些必要的预处理操作，其中最重要的就是分词 (tokenization) 和特殊字符的处理。这些预处理操作可以将原始的文本数据转化为模型可以直接处理的格式，并为后续的 Embedding 层做好准备。

自然语言是一种非结构化的数据, 它由一串串的字符组成, 没有明确的边界和结构。然而, 大多数自然语言处理模型 (包括 Transformer 以及前文提到的循环神经网络) 都不直接对字符进行操作, 而是需要先将文本切分成一个个有意义的单元, 如单词或子词。这个过程就是分词。

分词的目的是将连续的文本转化为离散的标记 (token) 序列, 以便于模型的处理。一个好的分词方法应该能够准确地反映语言的语法和语义结构, 将属于同一个语义或语法单元的字符聚合在一起。常见的分词方法包括:

- 基于规则的分词: 这种方法通常使用一些预定义的规则来切分文本, 如根据空格或标点符号进行切分。这种方法简单直观, 但往往无法处理一些复杂的情况, 如缩写、人名等。
- 基于统计的分词: 这种方法通过统计语言中单词或子词的频率, 然后使用一些算法 (如最大匹配算法、条件随机场等) 来决定切分的边界。常见的统计分词算法包括 BPE (Byte Pair Encoding) 和 WordPiece 等。这些算法可以自动地从语料库中学习词汇表和分词规则, 因此更加灵活和鲁棒。

例如, 考虑下面这个句子:

“Transformer is a powerful tool for natural language processing.”

如果我们用空格来作为划分依据（基于规则的分词方法）, 我们会得到以下的标记序列:

[“Transformer”, “is”, “a”, “powerful”, “tool”, “for”, “natural”, “language”, “processing”, “.”]

但是如果我们使用基于统计的分词方法, 如 BPE 算法, 我们可能会得到一个不同的标记序列:

[“Trans”, “former”, “is”, “a”, “power”, “ful”, “tool”, “for”, “natural”, “language”, “process”, “ing”, “.”]

可以看到,BPE 算法将 “Transformer” 这个单词分成了两个子词 “Trans” 和 “former”, 将 “powerful” 分成了 “power” 和 “ful”, 将 “processing” 分成了 “process” 和 “ing”。这是因为在训练语料中,“Trans”、“former”、“power”、“ful”、“process”、“ing” 这些子词可能更加常见, 因此 BPE 算法倾向于将它们作为独立的标记。

选择合适的分词方法对模型的性能有重要影响。一个好的分词方法应该能够在词汇量和切分粒度之间取得平衡。词汇量过大会增加模型的复杂度, 但词汇量过小又可能导致未登录词 (Out-of-Vocabulary, OOV) 问题。因此, 在实践中, 我们通常会根据任务的特点和语言的属性, 来选择最适合的分词方法。

除了分词, 我们在处理输入序列时, 还会引入一些特殊的标记 (Special Token)。这些特殊标记并不对应实际的词汇, 而是用来表示一些特殊的语义或结构。常见的特殊标记包括:

CLS ：表示句子的开始, 通常用于分类任务中作为整个句子的表示。

SEP：表示句子的结束, 或者在多句输入的情况下, 用于分隔不同的句子。

MASK：表示被掩盖的单词, 用于训练过程中的掩码语言模型 (Masked Language Model, MLM) 任务。

PAD：表示填充符号, 用于将不同长度的句子补齐到相同长度。

UNK：表示未登录词, 即那些在训练集中没有出现过的词。

这些特殊标记为模型提供了额外的结构信息, 帮助模型更好地理解和处理输入序列。例如,[CLS] 标记可以用来表示整个句子的语义,[SEP] 标记可以用来区分不同的句子,[MASK] 标记可以用来训练模型预测被掩盖的单词。需要注意的是, 这些特殊标记也需要像普通的词一样, 在 Embedding 层中学习对应的嵌入向量。模型会通过训练, 自动地学习到这些特殊标记的语义表示, 从而更好地利用它们所携带的结构信息。

分词和特殊字符的处理是 Transformer 模型输入流水线中非常重要的一步。通过恰当的分词和特殊字符的引入, 我们可以将原始的文本数据转化为模型可以直接处理的标记序列, 并为模型提供额外的结构信息。这为后续的 Embedding 层和 Self-Attention 层的计算打下了基础。

2.5.3 Embedding

在处理语言任务时, 我们需要将一个句子 (序列) 中的每个单词用一个向量表示，最简单的表示方法就是表示为一个长度为 d 的 one-hot 向量，其中 d 是字典的大小，向量中只有该单词对应的位置为 1, 其他位置全为 0。这种表示简单直观, 但是向量之间是正交的, 并且所有向量间的欧氏距离均为 $\sqrt{2}$，因此无法刻画单词之间的相似性。并且这种表示下所有词向量构成的矩阵为 $d \times d$ 维，难以存储和计算。为了让模型能够更好地理解和利用单词之间的语义信息, 我们可以使用一个线性变换将这些词向量映射到一个连续的、低维的向量空间中, 这就是 Embedding 层的作用。

具体来说, 我们首先将输入序列表示为一个 one-hot 矩阵 $X^{\text{in}} \in \mathbb{R}^{n \times d}$, 其中 n 是序列长度。为了克服 one-hot 表示的局限性, 我们引入了 Embedding 层。Embedding 层包含一个可学习的嵌入矩阵 $W^{\text{em}} \in \mathbb{R}^{d \times d_m}$, 其中 d_m 是嵌入向量的维度, 通常远小于字典大小 d。通过嵌入矩阵, 我们可以将每个 one-hot 向量映射为一个稠密的嵌入向量:

$$X^{\text{em}} = X^{\text{in}} W^{\text{em}} \in \mathbb{R}^{n \times d_m}.$$

在训练过程中, 嵌入矩阵 W^{em} 会被随机初始化, 并与模型的其他部分一起学习。通过学习, 语义相近的单词将被映射到嵌入空间中的相近位置, 而语义不同的单词则被映射到相距较远的位置。这种连续的、高维的表示能够很好地刻画单词之间的语义关系, 为后续的模型学习提供了更好的特征。

由于 Transformer 模型本身没有记录序列位置信息的能力, 我们还需要为每个位置显式编码一个位置向量:$X^{\text{pos}} \in \mathbb{R}^{n\times d_m}$。位置编码的目的是为每个位置引入一个独特的、有意义的表示, 以帮助模型区分不同位置的词。一种简单而有效的位置编码方式是使用正弦和余弦函数:

$$X^{\text{pos}}[k,2i] = \sin(\frac{k}{10000^{2i/d_m}}),$$
$$X^{\text{pos}}[k,2i+1] = \cos(\frac{k}{10000^{2i/d_m}}),$$

其中,k 表示位置 (取值范围为 $[1,n]$),i 表示维度 (取值范围为 $[0,d_m/2-1]$)。这种位置编码满足 $X^{\text{pos}}[k]\cdot X^{\text{pos}}[k+dk]$ 只与 dk 有关，而与位置 k 无关，有助于模型学习到位置之间的相对关系。此外该位置编码能够将编码值域限制在 [-1,1]，从而有助于泛化到各种复杂序列。

最终, 我们将嵌入表示和位置编码相加, 作为 Transformer 模块的输入:

$$X^{(1)} = X^{\text{em}} + X^{\text{pos}}.$$

这个相加的过程, 可以看作是将词汇信息和位置信息融合在一起。通过这样的处理, 原始的单词序列被转化为一个携带语义和位置信息的连续向量序列, 为后续的自注意力计算和特征提取做好了准备。

注 12. 对于嵌入表示和位置编码，也存在其他融合方式，如直接将二者拼接。现在主流大模型仍采用相加的方式，这样做的好处是节省了隐藏空间维度。

注 13. 近年来，旋转位置编码 *RoPE (Rotary Position Embedding) (Su et al. 2024)* 横空出世，成为大模型位置编码的新宠。*RoPE* 编码通过引入二维旋转矩阵，在保持相对位置关系的基础上，允许模型（相比正文提到的正余弦编码方式）更有效地捕捉位置信息，从而增强处理序列任务的能力。

2.5.4 注意力层

单头注意力机制

在 Transformer 模型中, 自注意力机制扮演着核心角色。它允许模型在处理每个位置时, 都能够考虑到序列中其他位置的信息, 从而能够捕捉到单词之间的长程依赖关系。

自注意力机制的计算过程可以分为三步: 第一步是计算查询 (Query)、键 (Key) 和值 (Value) 矩阵; 第二步是计算注意力分数矩阵; 第三步是计算注意力输出。下面我们逐步讨论。

第一步, 对于第 l 层的输入 $X^{(l)} \in \mathbb{R}^{n\times d_m}$, 我们通过三个不同的可学习矩阵 $W^{Q(l)}, W^{K(l)}, W^{V(l)}$, 将其转化为查询矩阵 $Q^{(l)}$、键矩阵 $K^{(l)}$ 和值矩阵 $V^{(l)}$:

$$
\begin{aligned}
Q^{(l)}(X^{(l)}) &= X^{(l)}W^{Q(l)}, \quad W^{Q(l)} \in \mathbb{R}^{d_m \times d_q}, \\
K^{(l)}(X^{(l)}) &= X^{(l)}W^{K(l)}, \quad W^{K(l)} \in \mathbb{R}^{d_m \times d_k}, \\
V^{(l)}(X^{(l)}) &= X^{(l)}W^{V(l)}, \quad W^{V(l)} \in \mathbb{R}^{d_m \times d_v}.
\end{aligned}
$$

直观地理解, 查询矩阵 $Q^{(l)}$ 表示了当前位置对其他位置的查询, 键矩阵 $K^{(l)}$ 表示了每个位置的关键特征, 值矩阵 $V^{(l)}$ 则表示了每个位置的具体信息。通过这样的转换, 我们将原始的词向量映射到了三个不同的语义空间, 为后续的注意力计算做准备。

注 14. 通常为了计算方便，会设置 $d_q = d_k = d_v$。

注 15. 在神经网络中，要使用一个数据，我们一般不会直接用，而是先做一个线性变换，这种操作是十分常见的，目的是提取特征。

第二步, 我们根据查询矩阵和键矩阵, 计算注意力分数矩阵 $\mathrm{Attn}^{(l)}(X^{(l)}) \in \mathbb{R}^{n \times n}$:

$$
\mathrm{Attn}^{(l)}(X^{(l)}) = \mathrm{softmax}\left(\frac{\mathrm{mask}(Q^{(l)}(X^{(l)})K^{(l)}(X^{(l)})^T)}{\sqrt{d_k}}\right).
$$

注意力分数矩阵的每一行表示了当前位置对其他所有位置的注意力分布。具体地,$\mathrm{Attn}_{ij}^{(l)}$ 表示了第 i 个位置对第 j 个位置的注意力分数, 它是通过计算 $Q_i^{(l)}$ 和 $K_j^{(l)}$ 的点积 (内积), 并除以 $\sqrt{d_k}$ 得到的。这里除以 $\sqrt{d_k}$ 是为了缓解点积结果的方差过大问题, 有助于模型的训练。

值得注意的是, 在 Decoder 中, 为了保持自回归属性, 我们通常会在注意力分数矩阵上应用一个上三角形的掩码 (mask) 矩阵。这个掩码矩阵将注意力分数矩阵的上三角位置 $(i, j), i < j$ 上的值设为负无穷大, 这样在 softmax 操作后, 这些位置的注意力分数就会变成 0。这种掩码操作确保了在生成每个词时, 模型只能看到当前词和左边的词, 而看不到右边的词, 从而保持了语言模型的自回归属性。

softmax 函数是按行做的，即对一个矩阵 $A = \{a_{ij}\}_{n \times m}$，有

$$
\mathrm{softmax}(A)_{ij} = \frac{\mathrm{e}^{a_{ij}}}{\sum_{j=1}^{m} \mathrm{e}^{a_{ij}}}, \tag{2.33}
$$

其作用是将一组实数转化为一个概率分布, 它保证了每一行的注意力分数总和为 1, 并且每个分数都在 0 到 1 之间。这使得注意力分数具有了概率的意义, 表示了当前位置对其他位置的重要性。

第三步, 我们根据注意力分数矩阵和值矩阵, 计算注意力输出 $X^{\mathrm{qkv}(l)}$:

$$
X^{\mathrm{qkv}(l)} = \mathrm{Attn}^{(l)}(X^{(l)})V^{(l)}(X^{(l)}) \in \mathbb{R}^{n \times d_v}.
$$

注意力输出的每一行是值矩阵的加权平均, 权重就是注意力分数。直观地理解, 模型通过注意力分数有选择性地聚焦到序列中的不同位置, 并从这些位置提取相关信息, 形成当前位置的新表示。

为了确保注意力输出的维度与输入维度一致, 我们通过另一个可学习的矩阵 $W^{O(l)}$, 将 $X^{\mathrm{qkv}(l)}$ 映射回 d_m 维:

$$X^{\mathrm{pr}(l)} = X^{\mathrm{qkv}(l)} W^{O(l)}, \quad W^{O(l)} \in \mathbb{R}^{d_v \times d_m}.$$

最后, 我们将注意力层的输出 $X^{\mathrm{pr}(l)}$ 与输入 $X^{(l)}$ 相加, 并对其归一化:

$$X^{\mathrm{ao}(l)} = \mathrm{LayerNorm}(X^{(l)} + X^{\mathrm{pr}(l)}).$$

这里的残差连接 (residual connection) 和层归一化 (Layer Normalization, LN) 是 Transformer 模型的两个重要组件。残差连接帮助网络更容易学习恒等映射, 缓解了深度网络中的梯度消失问题; 层归一化则有助于稳定网络的训练, 加速收敛。它通过将每一层每一个位置的 d_m 维输出归一化到均值为 0、方差为 1 的分布, 缓解了不同层之间的协方差偏移 (covariate shift) 问题。

多头注意力机制

尽管单头注意力已经能够捕捉序列中的依赖关系, 但它仍然有一些局限性。直观地说, 单头注意力可以看作是从一个角度去审视序列, 这可能无法全面地理解序列的内部结构。

为了克服这一问题,Transformer 引入了多头注意力机制。多头注意力的基本思想是, 同时学习多组不同的查询、键、值矩阵, 然后并行地计算多个注意力函数, 最后将不同头的输出拼接起来。形式上, 对于第 l 层的第 i 个头, 我们有:

$$\begin{aligned}
Q_i^{(l)}(X^{(l)}) &= X^{(l)} W_i^{Q(l)}, \quad W_i^{Q(l)} \in \mathbb{R}^{d_m \times d_k}, \\
K_i^{(l)}(X^{(l)}) &= X^{(l)} W_i^{K(l)}, \quad W_i^{K(l)} \in \mathbb{R}^{d_m \times d_k}, \\
V_i^{(l)}(X^{(l)}) &= X^{(l)} W_i^{V(l)}, \quad W_i^{V(l)} \in \mathbb{R}^{d_m \times d_v}.
\end{aligned}$$

每个头的注意力分数矩阵和输出计算方式与单头注意力类似:

$$\begin{aligned}
\mathrm{Attn}_i^{(l)}(X^{(l)}) &= \mathrm{softmax}\left(\frac{Q_i^{(l)}(X^{(l)}) K_i^{(l)}(X^{(l)})^T}{\sqrt{d_k}} \right), \\
\mathrm{head}_i^{(l)} &= \mathrm{Attn}_i^{(l)}(X^{(l)}) V_i^{(l)}(X^{(l)}).
\end{aligned}$$

接下来, 我们将所有头的输出拼接起来, 再经过一次线性变换, 得到多头注意力的输出:

$$X^{\mathrm{pr}(l)} = \mathrm{Concat}(\mathrm{head}_1^{(l)}, ..., \mathrm{head}_h^{(l)}) W^{O(l)}, \quad W^{O(l)} \in \mathbb{R}^{h d_v \times d_m}$$

其中,h 是头的数量，Concat 按列拼接。最后, 与单头注意力类似, 我们对多头注意力的输出进行残差连接和层归一化:

$$X^{\mathrm{ao}(l)} = \mathrm{LayerNorm}(X^{(l)} + X^{\mathrm{pr}(l)}).$$

多头注意力机制允许模型在不同的子空间中学习序列的不同表示。直观地说, 每个头可以看作是从不同的角度去理解序列, 捕捉不同的依赖模式。例如, 一些头可能更关注短距离的语法关系, 另一些头则可能更关注长程的语义关系。通过将这些不同的表示合并在一起, 模型可以得到一个更全面、更鲁棒的序列理解。

多头注意力的另一个优势在于, 它增加了模型的表达能力, 而不需要增加计算复杂度。如果我们将所有的参数集中在一个头上, 模型的表达能力就会受到限制。但是通过将参数分配到多个头上, 模型可以学习到更丰富的表示, 而且多头并行运行，提升计算效率。

实践证明, 多头注意力显著提升了 Transformer 在各种任务上的性能。它已经成为 Transformer 模型的标配, 被广泛应用于自然语言处理的各个领域。

2.5.5 前馈神经网络层

除了自注意力层,Transformer 模型的另一个重要组成部分是前馈神经网络层(Feed-Forward Network, FFN)。前馈神经网络层对自注意力层的输出进行进一步的非线性变换, 增强了模型的表达能力。从结构上看, 前馈神经网络层其实就是我们前边介绍的两层全连接网络, 中间使用 ReLU 激活函数, 并在最后使用残差连接和层归一化。具体地, 对于第 l 层的输入 $X^{\mathrm{ao}(l)}$, 前馈神经网络层的计算过程如下:

$$X^{\mathrm{do}(l)} := \mathrm{FNN}(X^{\mathrm{ao}(l)}) = \mathrm{LayerNorm}\left(X^{\mathrm{ao}(l)} + \sigma(X^{\mathrm{ao}(l)}W^{l,1})W^{l,2}\right),$$

其中 $\sigma(\cdot)$ 代表 ReLU 激活函数, 而 $W^{l,1} \in \mathbb{R}^{d_m \times d_{ff}}$ 和 $W^{l,2} \in \mathbb{R}^{d_{ff} \times d_m}$ 是两个可学习的权重矩阵。这里的 d_{ff} 通常大于 d_m, 例如 $d_{ff} = 4d_m$。这里实质上是先经历一个两层全连接网络，然后再经历残差连接，然后是层归一化。

值得注意的是，这里的前馈神经网络是对逐个位置上的词做的，即把 n 个 d_m 维向量分别通过前馈网络计算，再重新组合成 $n \times d_m$ 维的句子矩阵。这种结构增加了模型的非线性。因为自注意力层本质上是一个线性变换 (忽略 softmax 的非线性), 前馈神经网络层的引入可以增强模型的非线性表达能力。并且前馈神经网络层对句长维度 n 没有要求，因此可以方便地插入到模型的各个部分，也可以处理不同长度的句子，兼容性好。

前馈神经网络层虽然结构简单, 但在 Transformer 模型中发挥了重要作用。它与自注意力层一起, 构成了 Transformer 的基本计算单元。经过前馈神经网络之后得到的 $n \times d_m$ 矩阵作为下一层的输入进入下一个计算单元中的自注意力层。即

$$X^{(l+1)} = X^{\mathrm{do}(l)}. \tag{2.34}$$

2.5.6 输出层

在 Transformer 模型的最后, 我们需要将最后一层的输出 $X^{\mathrm{do}(L)}$ 转化为具体的预测结果。这通常涉及两个步骤: 第一步是将 $X^{\mathrm{do}(L)}$ 映射到目标词汇的维度; 第二步是根据映射后的结果, 选择最可能的输出。

第一步的映射操作通过一个线性变换完成, 我们称之为投影层:

$$X^{\mathrm{out}} = X^{\mathrm{do}(L)} W^{\mathrm{proj}} + b^{\mathrm{proj}}, \quad W^{\mathrm{proj}} \in \mathbb{R}^{d_m \times d}, \quad b^{\mathrm{proj}} \in \mathbb{R}^{d},$$

其中 W^{proj} 和 b^{proj} 分别是投影矩阵和偏置向量。这个线性变换将 d_m 维的隐藏状态映射到 d 维的输出空间, 每个维度对应词汇表中的一个单词。

在得到投影后的结果 X^{out} 后, 我们需要根据 X^{out} 选择最可能的输出单词。这通常有两种方式: 贪婪解码 (greedy decoding) 和采样解码 (sampling decoding)。

贪婪解码的思路是, 在每个位置上, 都选择概率最大的单词作为输出:

$$\mathrm{Output} = \mathrm{argmax}(\mathrm{softmax}(X^{\mathrm{out}})).$$

这里的 argmax 的意思是返回最大值对应的指标，操作是在词汇维度上进行的，它返回概率最大的单词的索引。贪婪解码是一种确定性的解码策略，它总是选择局部最优的结果，但并不保证全局最优。

与贪婪解码不同，采样解码则是一种随机性的解码策略。在采样解码中，我们首先将 X^{out} 通过 softmax 函数转化为一个概率分布，然后根据这个分布随机抽样出一个单词:

$$\mathrm{Output} \sim \mathrm{softmax}(X^{\mathrm{out}}).$$

采样解码允许模型生成多样化的结果，但也可能生成一些不太合理的结果。在实践中，我们通常会引入一些技巧，如温度（temperature）、前 k 次采样（top-k sampling）等，来平衡输出的多样性（diversity）和质量（quality）。

输出层的设计与任务类型和评估指标密切相关。例如，在机器翻译任务中，我们通常使用集束搜索（beam search）来寻找全局最优的翻译结果，其在生成序列时会保持多个候选结果，而不仅仅是单一的最优路径；而在对话生成任务中，我们可能更关注生成结果的多样性和自然性。因此，输出层的选择需要根据具体的任务和需求来权衡。

2.5.7 Transformer 做推断的详细流程

在前面的小节中, 我们详细介绍了 Transformer 模型的内部结构和工作原理。现在, 让我们通过一个具体的例子, 来看看 Transformer 模型是如何进行预测的。

假设我们有一个训练好的 Transformer 模型, 我们现在想用它来回答一个问题: “我的名字是迈克, 我叫什么?”。经过分词, 这个问题可以表示为:

[“[CLS]” ,“我的” ,“名字是” ,“迈克” ,“我” ,“叫” ,“什么” ,“[SEP]”].

我们可以通过以下步骤来生成答案:

第一步, 我们将分词后的问题转化为模型可以处理的输入格式。假设经过编码, 这个问题对应的输入矩阵为:

$$X_0 = \begin{bmatrix} \mathbf{x}_{[\text{CLS}]} \\ \mathbf{x}_{\text{我的}} \\ \mathbf{x}_{\text{名字是}} \\ \mathbf{x}_{\text{迈克}} \\ \mathbf{x}_{\text{我}} \\ \mathbf{x}_{\text{叫}} \\ \mathbf{x}_{\text{什么}} \\ \mathbf{x}_{[\text{SEP}]} \end{bmatrix} \in \mathbb{R}^{8\times d}.$$

第二步, 我们将 X_0 输入到 Transformer 模型中, 经过多层的计算, 得到输出矩阵 X_0^{out}。我们将 X_0^{out} 的最后一行通过 softmax 转化为概率分布, 并选择概率最大的分词” 你的” 作为第一个生成的分词。

第三步, 我们将” 你的” 的嵌入向量 $\mathbf{x}_{\text{你的}}$ 附加到 X_0 的末尾, 形成新的输入矩阵:

$$X_1 = \begin{bmatrix} \mathbf{x}_{[\text{CLS}]} \\ \vdots \\ \mathbf{x}_{[\text{SEP}]} \\ \mathbf{x}_{\text{你的}} \end{bmatrix} \in \mathbb{R}^{9\times d}.$$

注 16. 这里我们需要注意 X_1 与前文提到的第 *1* 层的输出 $X^{(1)}$ 是两个完全不同的概念, 前者表示输入句子 X_0 完全通过 *Transformer* 模型之后得到的输出词向量 $\mathbf{x}_{\text{你的}}$ 拼接到 X_0 之后得到的 $9\times d$ 维矩阵; 后者是一个输入矩阵 X 经过 *Transformer* 第 *1* 个 *Decoder* 层后得到的输出, 如输入矩阵为 $X = X_0$, 则 $X^{(1)} \in \mathbb{R}^{8\times d_m}$。

然后, 我们将 X_1 再次输入到 Transformer 模型中, 得到新的输出矩阵 X_1^{out}。我们选择 X_1^{out} 的最后一行对应的分词” 名字是” 作为第二个生成的分词。

第四步, 我们重复第三步, 将” 名字是” 的嵌入向量 $\mathbf{x}_{\text{名字是}}$ 附加到 X_1 的末尾, 形成新的输入矩阵 X_2, 然后输入到 Transformer 模型中, 得到 X_2^{out}。我们选择 X_2^{out} 的最后一行对应的分词” 迈克” 作为第三个生成的分词。

第五步, 我们继续重复第三步, 将” 迈克” 的嵌入向量附加到 X_2 的末尾, 形成 X_3, 然后输入到 Transformer 模型中, 得到 X_3^{out}。我们发现 X_3^{out} 的最后一行对应的是特殊的终止符”[SEP]”, 表示生成结束。

至此, 我们得到了完整的答案:

[“你的” ,“名字是” ,“迈克” ,“[SEP]”].

可以看到,Transformer 模型通过不断地将前面生成的分词反馈到输入中, 一步步地生成出完整的答案。这个过程就像是一个自回归的过程, 模型在每一步都根据之前的输出来预测下一个分词, 直到生成终止符为止。这个例子展示了 Transformer 模型在实际应用中的预测过程。当然, 在实践中, 我们通常会使用一些更高效的解码策略, 如 Beam Search, 以得到更优质的生成结果。但无论使用何种解码策略, 其基本思路都是相同的, 即通过不断地将前面的输出反馈到输入中, 来实现自回归的生成过程。这也是 Transformer 模型的一个重要特点: 它可以根据前文的信息, 动态地调整每一步的预测。这使得 Transformer 模型具有很强的语境理解和生成能力, 可以应对各种复杂的自然语言处理任务。

以上就是 Transformer 模型的整体结构和工作流程。从输入的词嵌入, 到自注意力层和前馈神经网络层的交替计算, 再到最后的输出层, 共同构建了一个强大的语言序列处理模型。Transformer 模型的设计充分考虑了自然语言的特点, 如长程依赖、语义复杂性等, 同时也兼顾了模型的计算效率和可扩展性。当然,Transformer 模型也并非完美无缺。其巨大的模型尺寸和计算开销, 对硬件和能源提出了很高的要求; 其黑盒性质, 也给模型的解释和调试带来了挑战。此外, 如何将 Transformer 模型应用到更广泛的场景, 如多模态学习、强化学习等, 也是一个值得探索的方向。尽管如此,Transformer 模型已经成为当前自然语言处理领域的主流范式, 其影响力还在不断扩大。未来, 随着人们对语言理解的不断深入, 以及人工智能技术的不断发展,Transformer 模型必将在更多领域大放异彩, 为我们开启通向智能时代的大门。

2.6 生成模型

生成模型主要用于解决生成任务，即给定一批真实数据 $\{\boldsymbol{x}_i\}_{i=1}^N$，但我们并不知道这些数据背后的分布 μ。为了模拟这种未知分布，生成模型从一个已知的分布中随机采样一个隐变量 $\boldsymbol{z}$，然后通过解码器 $\boldsymbol{g}$ 进行映射，生成样本 $\boldsymbol{x}$，使其也服从分布 μ。

2.6.1 自编码器

然而，直接这样做会面临一个问题—— 对于一个图像 x，我们事先并不知道它对应的隐空间变量。但是，我们希望它是一个平滑的空间，能被图像连续插值，即当图像 x_1 与图像 x_2 接近 ($x_1 - x_2$ 的某种空间范数比较小) 的时候，对应的隐变量 z_1 和 z_2 也足够近 ($z_1 - z_2$ 的某种空间范数比较小)，否则 z 到 x 的映射为一个高频映射，根据频率原则，$\boldsymbol{g}$ 会非常难以拟合。于是，我们可以假设 z 是 x 的连续映射，这样就保证了 $\boldsymbol{g}$ 一定不是一个高频映射。编码器 $\boldsymbol{f}$ 正是用于保证这一点的工具，这样的直接由编码器和解码器两部分组成的结构也被称为自编码器。

自编码器是一种特殊类型的神经网络。它属于无监督学习算法，其主要目的是通过训练学习输入数据的有效表示（编码），然后再重构出与原始输入尽可能相似的输出（解码）。这种网络由两个主要部分组成：编码器和解码器。

- 编码器 $\boldsymbol{f}$：将高维的输入数据 $\boldsymbol{x} \in \mathrm{R}^n$ 压缩为一个较低维的隐变量 $\boldsymbol{z} \in \mathrm{R}^d$，$d \ll n$。
- 解码器 $\boldsymbol{g}$：将这个低维的隐变量 $\boldsymbol{z}$ 重构为原始的高维的输入数据 $\boldsymbol{x}$ 形式。

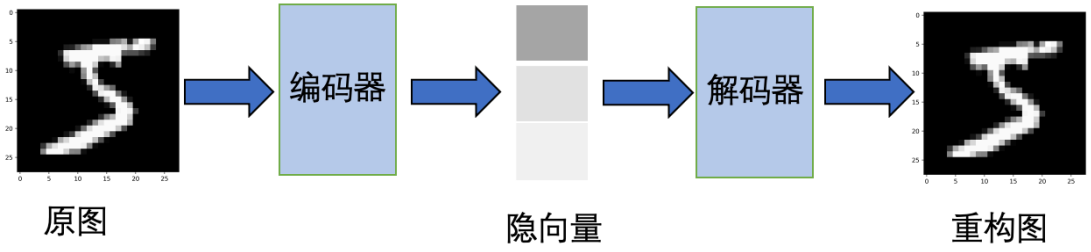


图 2.23: 自编码器示意图

通过使用神经网络 $\boldsymbol{f}_\theta$ 和 $\boldsymbol{g}_\theta$ 分别参数化编码器 $\boldsymbol{f}$ 和解码器 $\boldsymbol{g}$，再最小化如下损失函数：

$$L = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{g}_\theta(\boldsymbol{f}_\theta(\boldsymbol{x}_i)\|^2, \tag{2.35}$$

其中 x_i 为输入数据。最终目标就是通过 $\boldsymbol{f}_\theta$ 将高维数据的内在结构和规律压缩到一个低维的隐藏空间，并且通过解码器 $\boldsymbol{g}_\theta$ 完成生成任务（给定一个隐变量 $\boldsymbol{z}$，解码器返回一个高维数据 $\boldsymbol{g}_\theta(\boldsymbol{z})$）。

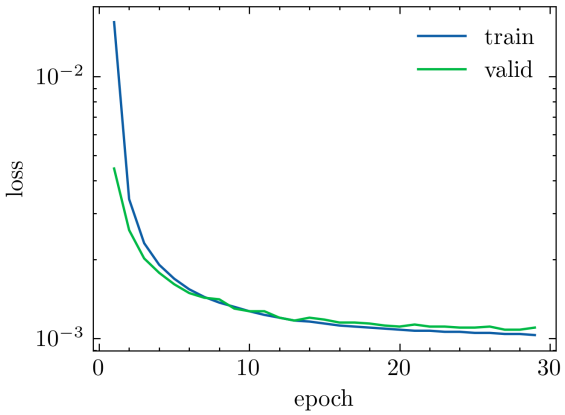
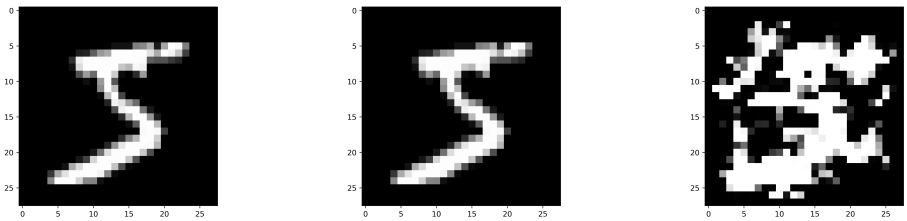


图 2.24: 自编码器的损失历史

2.6.2 变分自编码器

尽管自编码器的构想非常吸引人，但在实际应用中，所拥有的数据相对于其所在的实际空间往往是极为稀疏的。因此，自编码器在数据压缩与恢复方面表现较好，但在生成新数据时却未必理想。这主要是因为自编码器无法有效填补数据空间中的空隙，导致生成的内容往往无法满足预期。为了更直观地说明这一点，我们在 Fashion-MNIST 数据集上训练了一个自编码器。训练损失曲线如图 2.24 所示，模型在训练集和验证集上的重构效果都相当不错。图 2.25(a) 展示了 Fashion-MNIST 中的一张手写数字 5 的图像，图 2.25(b) 则是自编码器对其进行重构后的结果。然而，当我们对图 2.25(a) 经过自编码器的编码器得到的隐向量加上均值为 0，标准差为 5 的高斯噪声后，隐向量已不再位于原始数据分布内，此时重构出的图像也不再是手写数字或者是有意义的照片。



(a) 原图　(b) 自编码器重构图　(c) 扰动图

图 2.25: 自编码器生成和重构效果展示

为了解决上述问题，我们可以转变思路，不再将图片简单地映射为单一的“数值编码”，而是

将其映射为“分布”。由于是“分布”映射，我们将编码器和解码器分别记为 $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ 和 $p_\theta(\boldsymbol{z}|\boldsymbol{x})$。对于编码器来说，其自然的正向过程是将图片变换为隐变量，而对于解码器来说，其自然的正向过程是反过来，将隐变量变为图片。因此，在变分自编码器（Variational Auto-Encoder, VAE）Kingma (2013) 论文中，做了如下三点假设：

- 编码器 $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ 假设为一个高斯分布，均值和方差为 μ_θ 和 σ_θ 是网络预测的结果，$\boldsymbol{z}$ 从 $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$ 采样。
- 解码器 $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ 假设为一个高斯分布。
- 先验分布: $p(\boldsymbol{z})$ 为一个标准高斯分布。

回顾一下我们的目标：我们希望在标准正态分布 $p(\boldsymbol{z})$ 当中随机采样一个样本，并通过经过解码器 $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ 返回一张有意义的图像，且编码器是一个训练解码器的必需品。关键的问题是，如何同时训练 $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ 和 $p_\theta(\boldsymbol{x}|\boldsymbol{z})$，使得解码器 $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ 具有我们想要的生成能力？为了解决这一问题，我们需要考虑以下两个方面：

- 训练的核心目标是最大化对数似然 $\log(p_\theta)$，但显然这个概率无法直接从我们的三个假设中获得。于是，我们通过条件概率公式, 有

  $$p_\theta(\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})},$$

  但由于 $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ 未知，我们无法直接计算这个积分。因此，在 VAE 中通过构造一个下界来近似该对数似然。构造的想法是要凑一些熵或者相对熵的形式，因为这些形式可以保证恒大于 0，因此可以放缩。推导如下：

  $$\log p_\theta(\boldsymbol{x}) = \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})} \tag{2.36}$$

  注意到 $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ 是不可知的, 只有 $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ 和 $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ 是可知的。因此, 我们要利用 $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ 来构造一个恒大于 0 的量。可以考虑构造关于 $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ 和 $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ 的相对熵。

$$
\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{x}) \\
&= \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})} \\
&= \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} \log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})}{q_\theta(\boldsymbol{z}|\boldsymbol{x})} \frac{q_\theta(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})} p(\boldsymbol{z}) \\
&= \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})}{q_\theta(\boldsymbol{z}|\boldsymbol{x})}] + \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log \frac{q_\theta(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})}] + \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log p(\boldsymbol{z})] \\
&= \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})}{q_\theta(\boldsymbol{z}|\boldsymbol{x})}] + KL[q_\theta(\boldsymbol{z}|\boldsymbol{x})|p_\theta(\boldsymbol{z}|\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log p(\boldsymbol{z})] \\
&\geq \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})}{q_\theta(\boldsymbol{z}|\boldsymbol{x})}] + \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log p(\boldsymbol{z})],
\end{aligned}
$$

通过最大化这个下界，我们可以间接优化目标函数，训练解码器以提高其生成能力。于是，得到了最终的损失函数,

$$
\begin{aligned}
L &= \mathbb{E}_{\boldsymbol{x}} \left[-\mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})}{q_\theta(\boldsymbol{z}|\boldsymbol{x})}] - \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} [\log p(\boldsymbol{z})] \right] \\
&= \mathbb{E}_{\boldsymbol{x}} \left[KL(q_\theta(\boldsymbol{z}|\boldsymbol{x})|p(\boldsymbol{z})) - \mathbb{E}_{\boldsymbol{z}\sim q_\theta(\boldsymbol{z}|\boldsymbol{x})} p_\theta(\boldsymbol{x}|\boldsymbol{z}) \right].
\end{aligned}
\tag{2.37}
$$

不妨设 $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ 的均值和方差分别为 $\boldsymbol{\mu_{\theta_1}}(\boldsymbol{z}) = (\mu^1_{\theta_1}, ..., \mu^K_{\theta_1})$，$\boldsymbol{\Sigma^2_{\theta_1}}(\boldsymbol{z}) = (\sigma^1_{\theta_1}, ..., \sigma^K_{\theta_1})$，$q_{\theta_2}(\boldsymbol{z}|\boldsymbol{x})$ 的均值和方差分别为 $\boldsymbol{\mu_{\theta_2}}(\boldsymbol{z}) = (\mu^1_{\theta_2}, ..., \mu^M_{\theta_2})$，$\boldsymbol{\Sigma^2_{\theta_2}}(\boldsymbol{x}) = (\sigma^1_{\theta_2}, ..., \sigma^M_{\theta_2})$。于是，

$$
\begin{aligned}
L \simeq & \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} \left(-1 + \sigma^m_{\theta_2}(\boldsymbol{x}_n) + \mu^m_{\theta_2}(\boldsymbol{x}_n) - \log \sigma^m_{\theta_2}(\boldsymbol{x}_n) \right) \\
& - \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{j=1}^{P_n} \sum_{k=1}^{K} \frac{(\boldsymbol{x}_n - \mu^k_{\theta_1}(\boldsymbol{z}_j))^2}{\sigma^k_{\theta_1}(\boldsymbol{z}_j)} \right).
\end{aligned}
\tag{2.38}
$$

注 17. 实际上，细心的同学可能已经注意到，损失函数中的 $\boldsymbol{z}$ 是通过编码器 $q_\theta(\boldsymbol{z}|\boldsymbol{x})$ 采样得到的，而直接从该分布中采样的过程是不可微的。为了解决这个问题，VAE 中采用了重参数化技巧。具体做法是，首先从标准正态分布中采样一个噪声 η，然后通过 $\boldsymbol{z} = \mu_\theta + \sigma_\theta \eta$ 进行变换，从而由链式法则 $\frac{\partial}{\partial \theta} = \frac{\partial}{\partial z}(\frac{\partial \mu_\theta}{\partial \theta} + \eta \frac{\partial \sigma_\theta}{\partial \theta})$。

2.7 习题

- 全连接网络能处理任意结构的数据, 但在处理图像、序列等结构化数据时, 其参数量往往过大。在处理高维数据时, 我们可以采取哪些措施来减少参数量?

- 残差网络中的跳跃连接 (Skip Connection) 对缓解梯度消失问题有重要作用。请推导残差网络中梯度反向传播的公式，并说明该公式如何体现出残差连接有助于梯度传播。
- 卷积运算具有平移等变性, 即 $f(g(x,y)) = g(f(x,y))$, 其中 f 为卷积运算,g 为平移操作。请证明这一性质。并思考为什么平移不变性对于图像识别任务是有益的?
- 给定一个二维卷积层——输出通道为 16、卷积核大小为 3、填充为 1、步长为 5。假设输入尺寸为 26×26 的 RGB 图片，卷积层的输出形状是什么？该卷积层一共有多少参数?
- 请从梯度传播的角度说明为什么 RNN 容易出现梯度消失或梯度爆炸问题。
- 分词对于文本数据处理有什么意义？对分词方法及其使用的自然语言类型进行举例说明, 请查阅有关资料作答。
- Transformer 所使用的正余弦位置编码有什么特点？大模型还有哪些其他的编码方式？请查阅有关资料作答。
- 请阐述 Transformer 的多头自注意力相较于单头自注意力的优势及其原因。
- 自注意力机制可以捕捉序列中任意两个位置之间的依赖关系。对于一个长度为 n 的序列, 自注意力的计算量是多少?Transformer 的计算量是多少?
- Transformer 的解码器使用了自回归 (Auto-Regressive) 的生成方式, 即每次生成一个新的词, 再将其加入到输入序列中, 直到生成终止符。这种生成方式有什么优点和缺点?
- Transformer 在处理非常长的序列时, 其内存消耗和计算开销会变得很大。请提出一种改进的 Transformer 结构, 能够更高效地处理长序列。你的设计如何权衡计算效率和建模能力?
- 层归一化（Layer Normalization）在 Transformer 中的作用是什么? 它与批归一化（Batch Normalization）有何不同?

Chapter 3

维数灾难

维数灾难是机器学习和数据科学领域中一个根本性的挑战。当我们处理的数据维度增加时，很多算法的性能会显著下降，计算复杂度呈指数级增长，需要的样本量急剧增加。这种现象不仅仅是一个技术难题，而是高维数据分析中的基本限制，影响着从图像识别到自然语言处理的众多应用领域。本章将带领读者深入探索维数灾难的本质、影响以及克服方法。

为了更好地理解高维问题的特殊性，本章 3.1节将介绍一些在高维空间中具有反直观性的几何和统计特性。例如，高维单位球的体积随维度增加迅速减少至接近零；在高维单位球中随机选取的两个点几乎必定相距一个几乎恒定的距离，并且极有可能是正交的；数据点几乎全部集中在球的表面附近，而非中间区域等等。在 3.2节中，我们将通过一些实例进一步探讨在实际问题中遇到的维数灾难。

近年来，深度学习因其卓越的表示能力和灵活的结构，为克服维数灾难带来了新希望。实验结果显示，深度神经网络在处理高维数据（如自然图像、自然语言、围棋等）方面远超过传统方法，成功解决了多个具有挑战性的高维问题。这无疑让人对深度学习的潜力充满期待。但深度学习到底是如何胜任这些任务的呢？在何种意义下它能克服维数灾难？在 3.3节中，我们将围绕“克服维数灾难的策略”这一主题，探讨对蒙特卡洛方法和神经网络方法克服维数灾难的理解。

科普篇

什么是维数灾难？

维数灾难（Curse of Dimensionality，简称 CoD）是指当数据的维度（例如特征数量）增加时，许多机器学习和统计方法所需的参数量或样本量随维数指数级上升的现象。这个术语最

早由 Richard Bellman 在 20 世纪 50 年代提出，用来描述动态规划中维度增加导致计算复杂度指数上升的问题。随着数据科学的发展，维数灾难被广泛应用于机器学习、优化和数值分析等领域。

为什么要关注维数灾难？

在实际应用中，数据集常常具有高维特性。图像和自然语言数据通常位于非常高维的空间。例如，一张 256×256 的灰度图像可以看作是一个 65536 维的向量，而一个自然语言句子，如果每个单词使用 1000 维的词向量表示，那么 100 个单词的句子可以看成一个 100000 维的向量。当维度增加时，数据点变得稀疏，计算复杂度增加，很多传统方法难以有效处理。这种高维空间的稀疏性和复杂性会对机器学习模型的训练和预测造成巨大挑战，因此理解和克服维数灾难是数据科学的重要课题。

机器学习为什么会遇到维数灾难？

机器学习的一项核心问题是学习一个未知的函数 f^*。这通常描述为一个回归问题：给定训练数据集 $\{(x_i, y_i)\}_{i=1}^n$，其中 $y_i = f^*(x_i)$，我们的目标是基于这些数据找到一个函数 $\hat{f}$，使得 $\hat{f}$ 尽可能精确地逼近 f^*。然而，当我们处理的数据具有高维特性时，这个看似简单的任务会变得异常困难。

直观来讲，在高维空间中，数据点通常非常稀疏。简单来说，如果我们把一个区间切成多个等分，那么在高维空间中，这些小区间的数量会随着维度增加呈指数级增长。举个例子，在二维空间中，如果把一个区间分成 10 等分，我们会有 100 个小区间；而在三维空间中会有 1000 个小区间。当维数很高时，哪怕有数百万的数据点，这些数据点在高维空间中也显得非常稀疏，无法覆盖所有小区间。这样，高维空间中的数据点分布非常稀少，使得逼近一个高维函数变得异常困难。经典的函数逼近理论 (DeVore 1998) 正是刻画了这一点：在高维空间中，逼近一个目标函数所需的样本量会随着维度的增加而指数上升，因此会遇到维数灾难。

克服维数灾难的方法有哪些？

克服维数灾难的经典方法之一是蒙特卡洛方法。考虑一个高维数值积分的例子，在高维情况下，如果我们使用传统的打网格方法求解，网格数目随维数指数增加，因此将会遇到维数灾难，而使用蒙特卡洛方法随机采样，积分精度能够与维数 d 无关，因此克服了维数灾难。

近年来，深度神经网络在处理高维数据（如自然图像、自然语言、围棋等）方面表现出了卓越的能力，成功解决了多个具有挑战性的高维问题。这无疑让人对深度学习的潜力充满期待。但深度学习到底是如何胜任这些任务的呢？最近在关于 Barron 空间的研究中，Barron 空

间为我们提供了一个理解神经网络如何克服维数灾难的视角。传统的机器学习方法在高维数据上表现不佳，主要是因为其参数复杂度或样本复杂度随维数呈指数增长，而神经网络在某些情况下能避开这一诅咒，关键在于所要逼近的函数空间具有某些特殊的性质。Barron 空间就是这样的一个特殊例子。对于属于 Barron 空间的函数，其可以有一个积分表示，借助蒙特卡洛方法可以克服数值积分维数灾难的思想，一些工作证明了单隐层神经网络能够以较低的参数复杂度和较低的样本复杂度实现良好的逼近和泛化，其逼近和泛化误差均与维数 d 无关，因此不受维数灾难的影响。这给了我们一个重要的启示：如果我们能够找到更适合刻画真实世界数据的函数空间，并设计出能够有效逼近这些函数空间的神经网络结构，那么我们就有希望在高维情形下获得良好的学习效果。

3.1 高维空间的特点

高维空间有许多反直觉的特点，与我们习惯的低维空间有很大不同。本节我们将介绍几个高维空间的重要性质，帮助读者建立关于高维空间的直观感受。

3.1.1 高维空间中数据的稀疏性

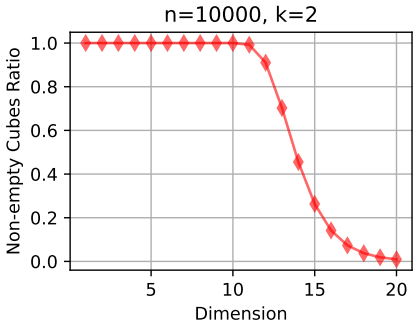
在高维空间中，数据点的分布通常是极其稀疏的。想象在一个 d 维的单位超立方体 $[0,1]^d$ 内随机撒下 n 个点，其中每个点的坐标是在区间 $[0,1]$ 上独立且均匀分布的。如果我们将这个区间划分为 k 个等长的子区间，那么整个超立方体就会被分割成 k^d 个小立方体。随着维数 d 的增加，哪怕是数百万的数据点也不足以覆盖 k^d 个小立方体中的每一个。换句话说，大多数立方体将是空的，这种情况在高维空间中是普遍存在的。

为了确保每个小立方体至少有一个数据点，我们需要的样本量 n 需要满足：

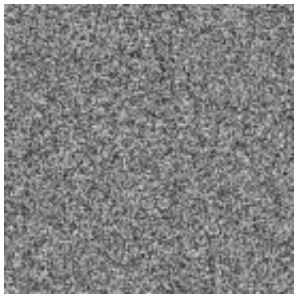
$$n \geq k^d. \tag{3.1}$$

这意味着样本量 n 需要随维数 d 指数级增长。在现实应用中，数据的维数 d 可能非常高（例如一张 256×256 的灰度图像其维数就高达 65536），而可用的样本量 n 通常远小于 k^d，因此高维空间中数据的稀疏性是一个不可忽视的问题。

为了直观地呈现数据的稀疏性，我们考虑下面这个实验：在 d 维的单位超立方体 $[0,1]^d$ 内随机采样 $n = 10000$ 个点，其中每个点的坐标是在区间 $[0,1]$ 上独立且均匀分布的。将这个区间划分为 $k = 2$ 个等长的子区间，整个超立方体被分割成 2^d 个小立方体，我们统计非空小立方体所占的比例。从图 3.1(a) 中可以看出，随着维数的增加，非空小立方体所占的比例迅速下降，这表明数据在高维空间中变得越来越稀疏。具体而言，在 10 维空间中，10000 个数据点显得还较为密集，但在 20 维空间中，非空超立方体所占的比例几乎已经接近于零。这意味



(a) 非空立方体的比例



(b) 随机采样的图像

图 3.1: 高维空间的数据稀疏性实验

着，在 20 维空间中，10000 个数据点的分布极度稀疏。而如果我们进一步考虑大约 15 万维的 ImageNet 图像空间，$2^{150000} \approx 10^{45154}$。这是一个无比庞大的数字! 为了给读者提供一个参照，宇宙中原子的数量级大约为 10^{80}。即使我们每秒钟采集一张图像，日夜不停地进行采集，完成 2^{150000} 次采集所需的时间仍然远远超过宇宙的年龄（约 138 亿年）。换句话说，**高维空间几乎是“空的”**：一旦空间的维数变得足够高，在实际应用中采集到的数据点几乎可以忽略不计。

图像和自然语言数据通常位于非常高维的空间。例如, 一张 256×256 的灰度图像可以看作是一个 65536 维的向量, 而一个自然语言句子, 如果每个单词使用 1000 维的词向量表示, 那么 100 个单词的句子可以看成一个 100000 维的向量。在如此高维的空间中, 数据点天然是非常稀疏的。图 3.1(b) 展示了在高维图像空间中随机采样的结果, 生成的图像看起来像随机噪声, 即“雪花图”, 与自然图像差异巨大。这表明在高维图像空间中, 绝大多数区域对应的都是非自然的、随机噪声般的图像, 自然图像只占据了一个非常小的子空间。因此, 在非常高维的图像空间生成一张自然图像理论上是极其困难的, 需要非常特殊的算法和模型设计, 例如生成对抗网络 (GAN) 和扩散模型 (Diffusion Model)。

同样地, 在高维的自然语言空间中随机采样, 生成的文本通常缺乏语法结构和语义连贯性, 与自然语言文本有很大不同。虽然著名的“无限猴子定理”从理论上探讨了无限时间内随机敲击键盘可以最终生成莎士比亚作品的可能性，然而，实际情况是，高维空间中有意义的自然语言文本非常稀疏，在有限时间内的随机采样几乎不可能碰巧生成连贯、有意义的文本。这揭示了高维空间的一个基本特性：自然的、有意义的对象只占据了整个空间的一个极小子集，随机采样很难碰巧采到这些对象。这也是为什么我们需要专门设计的生成模型（如语言模型 GPT）来生成高质量的自然语言文本。

高维空间中数据的稀疏性对机器学习提出了重大挑战。然而，我们也应该注意到，高维空间中的数据尽管稀疏，但通常带有某种结构。以图像数据和自然语言处理为例，图像中的像素

并不是随机分布的，而是具有局部相关性，这样的结构允许我们用卷积神经网络（CNN）来捕捉图像中的平移不变性和局部特征。类似地，自然语言文本具有长程依赖性，即一个句子中的不同单词之间存在着依赖关系。因此，Transformer 模型使用自注意机制实现对句子中任意词对之间关系的建模。例如，在句子“猫坐在垫子上”中，“猫”和“垫子”之间存在着直接的信息关联，而这种关联可以被自注意机制有效捕捉和利用，从而生成更准确的文本表示。

3.1.2 体积集中在表面的特性

在高维空间中, 一个引人注目的现象是大部分体积都集中在对象的表面附近。为了理解这一点, 让我们考虑一个位于 d 维欧氏空间 $\mathbb{R}^d$ 中的紧致对象 Ω, 这里的“紧致”意味着它是封闭且有限大小的。如果我们将对象 Ω 的每个维度缩小一个微小的比例 ϵ, 得到一个新的内部对象 $(1-\epsilon)\Omega$, 那么这两个对象的体积之比满足:

$$\frac{V((1-\epsilon)\Omega)}{V(\Omega)} = (1-\epsilon)^d. \tag{3.2}$$

这个关系可以通过将对象 Ω 划分为无数个无穷小的立方体来理解。如果我们将 Ω 中每个立方体的边长缩小为原来的 $1-\epsilon$ 倍, 那么每个立方体的体积就会缩小为原来的 $(1-\epsilon)^d$ 倍, 从而整个对象的体积也会缩小 $(1-\epsilon)^d$ 倍。

利用不等式 $1-x \leq e^{-x}$ $(x>0)$, 我们可以得到:

$$\frac{V((1-\epsilon)\Omega)}{V(\Omega)} = (1-\epsilon)^d \leq e^{-\epsilon d}. \tag{3.3}$$

当我们固定 ϵ 并让维数 d 趋于无穷大时, 上述体积比会迅速趋近于零:

$$\lim_{d\to\infty}(1-\epsilon)^d \approx \lim_{d\to\infty} e^{-\epsilon d} = 0. \tag{3.4}$$

这意味着, 当维数 d 足够大时, 几乎所有的体积都集中在 Ω 的表面附近的薄层区域 $\Omega \setminus (1-\epsilon)\Omega$ 内, 而内部对象 $(1-\epsilon)\Omega$ 的体积相对于整个对象 Ω 的体积而言几乎可以忽略不计。

为了更具体地说明这一点, 让我们考虑一个 d 维单位球 $\boldsymbol{B}^d$。d 维球体是在三维空间中球体的推广。半径为 r 的 d 维球体可以视为原点为中心，所有点到原点的直线距离不大于 r 的一个几何体。考虑半径为 1 的情况。1 维球体是线段 $[-1,1]$。2 维球体是由单位圆界定的圆盘，其方程为 $x^2+y^2 \leq 1$。3 维球体（我们通常称之为“球体”）的方程是 $x^2+y^2+z^2 \leq 1$。我们可以将这个定义扩展到任何维度:

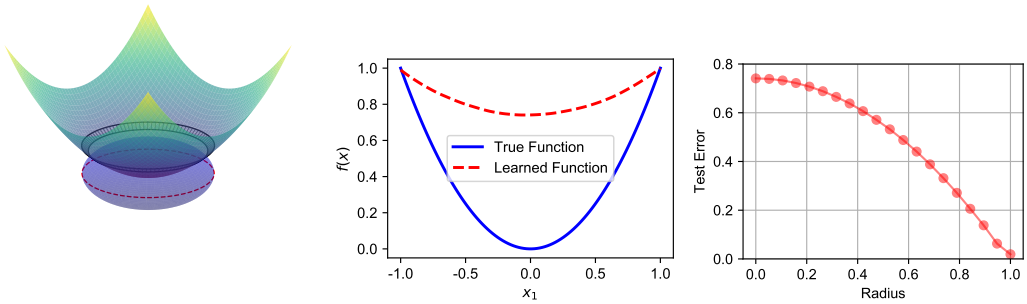
$$\boldsymbol{B}^d = \left\{ (x_1,\ldots,x_d) \in \mathbb{R}^d \middle| \sum_{i=1}^{d} x_i^2 \leq 1 \right\}.$$

维数 d	1	2	3	5	10	100	1000
内球体积比例	99.00%	98.01%	97.03%	95.10%	90.44%	36.60%	$< 0.01\%$

表 3.1: 随维数 d 增加, 内部球 $(1-\epsilon)\boldsymbol{B}^d$ 所占单位球体积的比例迅速下降。

根据我们上述的分析, 当 d 很大时, d 维单位球 $\boldsymbol{B}^d$ 大部分体积都集中在距离球心 $1-\epsilon$ 以外的薄球壳内。表 3.1 列出了当 $\epsilon = 0.01$ 时, 对于不同维数 d, 内部球 $(1-\epsilon)\boldsymbol{B}^d$ 的体积相对于整个单位球体积的比例。

从表 3.1 中可以看到, 随着维数 d 的增加, 内部球的体积所占比例迅速下降。在 1000 维空间中, 半径为 0.99 的内部球仅占单位球体积的不到 0.01%, 这意味着 99.99% 的体积都集中在厚度为 0.01 的薄球壳中。



(a) 高维空间的二次函数示意图　(b) 网络学到的函数和真实的函数对比　(c) 不同半径上的测试误差

图 3.2: 拟合高维空间中的二次函数实验

为了展示这种体积集中在表面的现象对实际机器学习的影响, 我们设计了一个拟合高维二次函数的实验。在这个实验中, 我们在 100 维单位球内随机采样 10000 个数据点 $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{10000}$, 其中 $y_i = \|\boldsymbol{x}_i\|_2^2$, 并用一个两层的 ReLU 神经网络 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ 来拟合这些数据点。

图 3.2(a) 展示了这个实验的二维示意图。半透明的蓝色区域表示采样数据点所在的区域, 即单位球。在神经网络训练完成之后（训练损失值小于 10^{-6}）, 我们分别从两个角度评估模型的表现:

1. 如图 3.2(b) 所示, 我们固定其他 99 个坐标为 0, 只沿第一个坐标轴从 -1 到 1 采样 500 个点, 并在这些点上比较神经网络的输出 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ 和真实的二次函数值 $y = x_1^2$。可以看到, 在边界附近 (-1 和 1), 神经网络的预测与真实值几乎完全重合, 而在中心区域二者存在明显偏差。这是因为高维球内均匀采样的绝大部分点都位于边界附近, 所以模型在这些区

域学习得很好, 而在数据稀疏的内部区域, 模型学到的函数与真实函数相去甚远。

2. 如图 3.2(c) 所示, 我们在不同半径 r 的球面上随机采样 1000 个点 $\boldsymbol{x}$, 并计算神经网络在这些点上的平均绝对误差 $\frac{1}{1000}\sum_{i=1}^{1000}|f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \|\boldsymbol{x}_i\|_2^2|$。可以看到, 随着半径 r 的增大, 平均误差逐渐减小, 并在 $r=1$ 时达到最低。这再次验证了我们的理论: 靠近球心的区域数据很少, 神经网络的泛化误差大; 而在靠近表面的区域数据较多, 神经网络的泛化误差小。

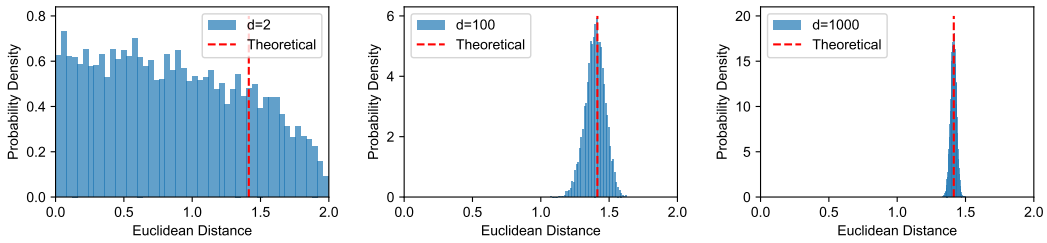
这个实验清晰地展示了高维空间中体积集中现象对机器学习的影响。在实际应用中, 如果我们从高维空间中随机采样数据点, 那么大部分数据点都会分布在边界附近的薄层区域内, 而内部区域的数据点则非常稀疏。这种数据分布的特点会影响机器学习模型的训练和泛化。如果我们不仅关心模型在大部分体积处的泛化表现, 还关心模型在体积很小的内部的泛化表现, 我们可以采取“重要性采样”的策略, 即在我们关心的区域人为地多采样数据, 来缓解高维空间带来的这一问题。

3.1.3 距离的集中效应与正交性

在高维空间中, 数据点之间的距离表现出一种“集中效应”, 即这些距离倾向于在某个特定值附近高度集中。同时, 随机选取的两个向量也倾向于彼此正交。为了直观地理解这些现象, 我们考虑在 d 维单位球内随机采样两个点 $\boldsymbol{x}$ 和 $\boldsymbol{y}$, 并通过实验分析它们之间的欧氏距离和夹角的分布。

首先, 我们来看距离的集中效应。在 d 维单位球 $\boldsymbol{B}^d$ 内随机采样两个点 $\boldsymbol{x}$ 和 $\boldsymbol{y}$, 它们之间的欧氏距离可以表示为:

$$\|\boldsymbol{x}-\boldsymbol{y}\|_2 = \left(\sum_{i=1}^{d}(x_i - y_i)^2\right)^{\frac{1}{2}}.$$



(a) d=2 (b) d=100 (c) d=1000

图 3.3: 高维单位球内随机采样点对之间欧氏距离的分布图, 红色虚线表示距离为 $\sqrt{2}$

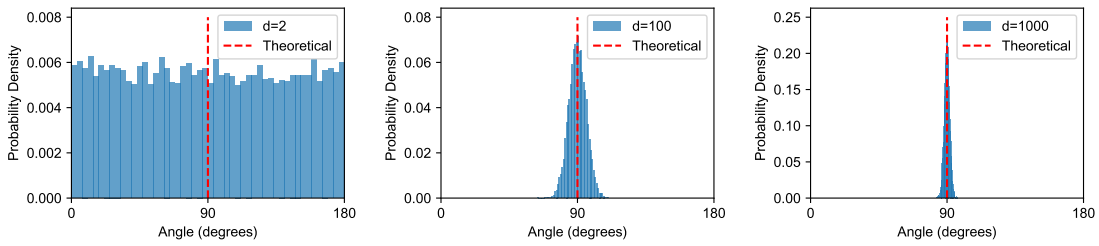
图 3.3展示了在 2 维、100 维和 1000 维单位球内随机采样 10000 对点, 并计算它们之间欧氏距离的分布情况。从图 3.3 可以看出, 随着维数 d 的增加, 随机采样点对之间的欧氏距离分布会越来越集中在 $\sqrt{2} \approx 1.414$ （图 3.3中红色虚线所示）附近, 其方差也会越来越小。这是高维空间中距离的集中效应的体现。

考虑传统的机器学习算法，例如经典的最近邻算法（kNN），依赖于距离来进行分类和回归操作。然而，正如图 3.3 所示，当维度增加时，最近邻和最远邻的距离差异越来越小。这意味着在高维空间中，所有点几乎都“同等遥远”，距离的概念变得模糊且失去区分性。换句话说，在高维空间中“**最近邻概念消失了**”：在二维或三维空间中，邻居的概念是清晰且直观的；而在高维空间中，最近的邻居和最远的邻居之间的距离几乎相同。这使得 kNN 算法和其他依赖距离的算法失去了有效性。

接下来, 我们分析随机采样向量之间的夹角。在 d 维单位球内随机采样两个点 $\boldsymbol{x}$ 和 $\boldsymbol{y}$, 它们之间的夹角 θ 可以通过点积计算:

$$\cos\theta = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}$$

图 3.4展示了在二维、100 维和 1000 维单位球内随机采样 10000 对点，并计算它们之间夹角的分布情况。从图 3.4 可以看出，随着维数 d 的增加，随机采样向量之间的夹角分布会越来越集中在 90°（图 3.4 中红色虚线所示）附近，其方差也会越来越小。这现象被称为高维空间中向量的角度集中效应。

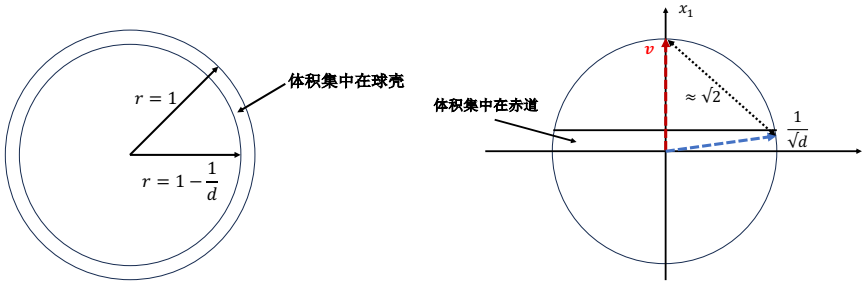


(a) d=2　　(b) d=100　　(c) d=1000

图 3.4: 高维单位球内随机采样向量之间夹角的分布，红色虚线表示夹角为 90°。

类似于最近邻算法在高维空间中遇到的问题，向量角度集中效应在某些机器学习算法中也会导致挑战。例如，在文本处理和信息检索领域中，常见的方法是用余弦相似度来衡量两个文档或向量之间的相似性。余弦相似度实际上是通过计算向量之间的夹角来评估相似性，角度越小表示两个向量越相似。然而，如图 3.4所示，在高维空间中，大多数向量之间的夹角都趋向于 90°，这意味着它们的余弦相似度接近零，导致所有向量看起来都不相似。这对依赖余弦相似度的算法带来了挑战。

下面让我们分析一下为什么会有距离和角度上的集中效应。首先，我们知道单位球 S 在 d 维空间中定义为与原点距离小于等于 1 的所有点的集合。根据上述体积集中在表面的观察，至少有 $1-e^{-\epsilon d}$ 比例的体积集中在单位球的边缘一小块厚度为 ϵ 的球壳中。特别地，如图 3.5(a) 所示，高维单位球的大部分体积集中在边界附近厚度为 $\frac{1}{d}$ 的球壳中。如果球的半径为 r，则球壳的厚度大约为 $\frac{r}{d}$。



(a) 体积集中在球壳　　(b) 体积集中在赤道

图 3.5: 单位球的体积集中在球壳和赤道的示意图

根据体积集中在表面的特性分析，如果我们在高维单位球内进行均匀的随机采样，有非常大的概率采到厚度大约为 $\frac{1}{d}$ 的球壳上。此外，另一个有趣的事实是，在高维空间中，单位球的体积不仅集中在表面，而且也集中在“赤道”附近。这里的“赤道”是指与任意选定的单位向量 $\boldsymbol{v}$ 正交的超平面与球壳的交集。具体来说，对于单位球面上的任何向量 $\boldsymbol{v}$，单位球中与该向量点积很小的点构成的集合占据了单位球的绝大部分体积。为了说明这一点，如图 3.5(b) 所示，不妨假设 $\boldsymbol{v}=[1,0,\cdots,0]^\top$，我们可以证明单位球中满足第一个分量 $|x_1|\leq 1/\sqrt{d}$ 的点占据了大部分体积，也即单位球的大部分体积位于与 $\boldsymbol{v}$ 的点积大小大约为 $1/\sqrt{d}$ 的薄层中（具体的数学推导此处省略，感兴趣的读者可以参考 Blum et al. (2020)）。

这个分析的一个直接结果是，如果我们从单位球中随机抽取两个点，例如图 3.5(b) 中的红色和蓝色虚线表示的向量，它们很可能是几乎正交的。具体而言，根据之前体积集中在球壳的分析，这两个点很可能都接近表面，并且长度为 $1-1/d$。如果我们将第一个点的方向定义为“北”，那么第二个点在这个方向上的投影只有 $\pm 1/\sqrt{d}$ 的概率很高，因此它们的点积将大约是 $\pm 1/\sqrt{d}$，随着维数 d 增大，这个点积将逐渐趋于 0。这意味着在高维的空间中，两个向量之间的距离几乎是 $\sqrt{2}$，而角度几乎是 $90°$。因此，在高维空间中，随机抽取的点不仅几乎都位于单位球的边缘，而且它们之间几乎正交。这也解释了高维空间中距离和角度的集中效应。

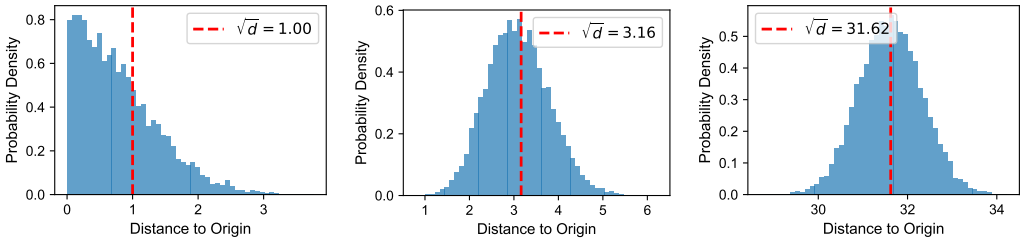
3.1.4 高斯环带效应

考虑一个具有单位协方差的 d 维高斯分布 $N(0, I_d)$，其概率密度函数为：

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{\|\boldsymbol{x}\|_2^2}{2}\right).$$

为了直观地展示高斯分布在不同维度下的特性，我们来看图 3.6。该图展示了从 1 维、10 维和 1000 维高斯分布中采样的 10000 个数据点距离原点的距离分布。

在 1 维情况下，如图 3.6(a) 所示，采样的数据点主要集中在原点附近，这符合我们的直观理解：大多数点都离原点不远。而在 10 和 100 维空间中，我们看到如图3.6(b, c) 所示的现象：大部分数据点集中在距离原点大约 $\sqrt{d}$ 的位置，并且这种分布形成了一个宽度几乎固定为常数 (随维度继续增加，宽度保持不变) 的环带区域。



(a) d=1　　(b) d=10　　(c) d=1000

图 3.6: 不同维数的高斯分布采样数据点到原点的距离分布图

图 3.6中的实验意味着随着维数的增加，数据点不再集中在原点附近，而是被“推”向一个特定的距离范围。这种现象被称为“高斯环带效应”。用更加直观的方式来说，高维空间中的大多数数据点更像是“围绕”原点的而不是“聚集”在原点。

下面我们尝试对这个现象进行一个直观的解释。首先，让我们考虑一下高维空间中半径为 r 的球体的体积。半径为 r 的 d 维球体的体积 $V(d)$ 可以用以下公式表示：

$$V(d) = \frac{2\pi^{\frac{d}{2}}}{d\Gamma\left(\frac{d}{2}\right)} r^d.$$

这里伽马函数 $\Gamma(d)$ 是是阶乘概念在实数和复数上的推广。直观地说，r^d 和 $\pi^{\frac{d}{2}}$ 是关于 d 和 $\frac{d}{2}$ 的一个快速增长的指数函数，而 $\Gamma\left(\frac{d}{2}\right)$ 随维数 d 变大，其增长速度与阶乘函数类似，因此二者相比，体积 $V(d)$ 随着维数 d 的增加将迅速趋近于 0：

$$\lim_{d\to\infty} V(d) = 0. \tag{3.5}$$

这表明，随着维数的增加，d 维球体的体积趋于 0，这对所有半径 r 都成立。

接下来，我们来直观地解释一下为什么从高维高斯分布采样的数据点会集中在一个环带：

- 在高维空间中，概率密度函数在原点（即 $\boldsymbol{x} = \mathbf{0}$）达到最大值，但原点附近包含的体积极小。虽然在原点处概率密度最高，但根据式 (3.5)，单位球的体积在高维空间中迅速减小，即使在原点附近的概率密度最大，整体上积分得到的概率几乎为零。
- 为了获得更显著的概率，我们需要增大积分的区域，考虑半径大约为 $\sqrt{d}$ 的球壳附近。在此球壳区域，虽然概率密度有所减小，但球体的体积变得相对较大，因此积分之后的总概率变得显著。
- 当 $\|\boldsymbol{x}\|_2^2$ 显著大于 d 时，由于指数衰减的影响，概率密度 $p(\boldsymbol{x})$ 会迅速趋近于零。这意味着，虽然球体的体积随着半径的增加而增加，但概率的提高却非常有限，因为概率密度的减小速度超过了体积的增加速度。

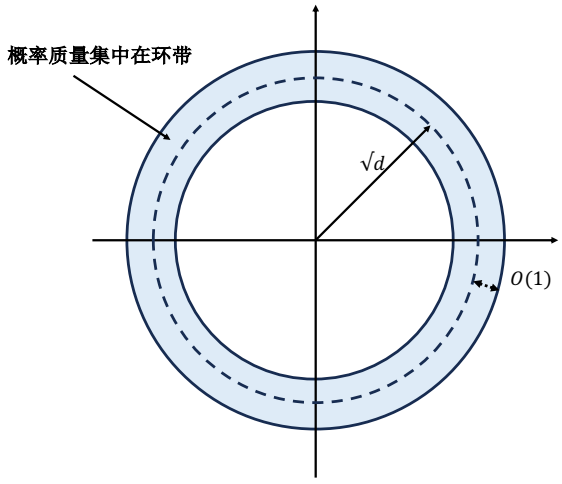


图 3.7: 高斯环带定理示意图

数学上，高斯环带定理（Gaussian Annulus Theorem）正式表述了这种集中趋势 (具体细节可以参考 Blum et al. (2020))。如图 3.7所示，对于每个方向上方差为 1 的 d 维高斯分布，这个定理断言几乎所有的概率都集中在半径为 $\sqrt{d}$、宽度为一个固定常数的环带中。

高斯环带效应对我们理解高维噪声有重要意义。例如，在扩散模型中，我们需要对一张 256×256 的自然图像加随机噪声。这个噪声图像的每个像素是独立的，服从 $d = 65536$ 维的高斯分布。高斯环带定理保证了加完高维随机采样的噪声后的图像与原来自然图像的距离几乎都是一个确定的常数（即 $\sqrt{d}$）。此外，在扩散模型中，需要添加多步噪声。因为高斯分布的多

步叠加依然是一个高斯分布, 加噪后的图像距离原始图像依然有一个确定的正距离。因此, 不论我们如何加噪, 我们几乎都不用担心噪声之间会相互抵消, 从而回到原来的自然图像。但是对于一维问题，在多步叠加高斯分布噪音的过程中, 会有显著的概率返回原点附近。

此外，在大规模文本分类任务中，文档通常被表示为高维向量，其中每个维度对应一个词汇项。由于词汇量庞大，这些向量处于高维空间中。高斯朴素贝叶斯算法假设每个类别的文档特征按照高维高斯分布进行建模。高斯环带效应意味着，尽管文本向量可能处于高维空间中，它们的距离在各类内部会集中在特定范围内。因此，使用高斯分布来建模文档类别间的差异，能够有效区分不同类别。例如，假设有两个类别的文档，它们在高维空间中分别形成两个不同的高斯分布。高斯环带效应确保每个分类内的文档向量在一个确定的距离范围内，这使得分类器在这种环境下仍然有效。

3.1.5 随机投影降维

在现代机器学习, 尤其是深度学习中, 我们经常需要处理高维数据, 例如图像数据和自然语言数据。假设我们有一个包含 n 个数据点的数据集, 其中每个数据点都位于 d 维空间 $\mathbb{R}^d$ 中。随着数据维数 d 的增加, 直接在原始高维空间中对数据进行分析和处理会带来较高的存储和计算负担。这就需要用到降维技术。

在高维空间中，有一个重要的特点是，我们可以通过随机投影降维来保持点与点之间的距离关系。一个经典的定理——Johnson-Lindenstrauss 引理, 利用高斯环带效应证明了可以通过随机线性映射将原始数据嵌入到一个通常远低于原始维数 d 的 k 维空间（k 依赖于 $\log n$）中, 且点与点之间的距离可以近似得以保持 (具体细节可以参考 Blum et al. (2020))。

该引理中的随机映射 f 非常简单，可以用一个 $k \times d$ 的随机矩阵 $\boldsymbol{R}$ 来实现:

$$\boldsymbol{R} = \frac{1}{\sqrt{k}} \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1d} \\ r_{21} & r_{22} & \cdots & r_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ r_{k1} & r_{k2} & \cdots & r_{kd} \end{pmatrix} \in \mathbb{R}^{k \times d}.$$

其中 r_{ij} 是独立同分布的高斯随机变量, 均值为 0, 方差为 1。这样对于任意 d 维向量 $\boldsymbol{x}$, 其随机投影 $\boldsymbol{f}(\boldsymbol{x})$ 就可以表示为 $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{R}\boldsymbol{x} \in \mathbb{R}^k$。

Johnson-Lindenstrauss 引理表明, 高维空间中的 n 个点可以嵌入到维数约为 $O(\log n)$ 的低维空间中, 且点与点之间的距离可以近似得以保持，并且这种嵌入只需要通过一个简单的随机投影映射就可以实现。在实践中, 我们通常取 $\log n$ 量级的维数就可以得到较好的嵌入效果。

在机器学习领域, 随机投影引理启发我们可以通过嵌入或投影, 将高维数据压缩到一个较低维的空间来处理，这能够降低存储和计算资源的消耗。例如在自然语言处理任务中, 我们通

常将每个词表示为一个 one-hot 高维词向量，维数取决于词表的大小 n，通常可以达到数十万。随机投影引理告诉我们，可以使用随机投影将这些词向量嵌入到一个 $O(\log n)$ 维的更低维空间中，而词之间的语义相似性（通常用向量之间的距离来度量）依然能够得到很好的保持。在图像领域，考虑我们有一组高清图片，每张图片都是一系列像素值的高维数字表示。传统处理高维图像数据不仅费时而且费力。通过随机投影降维，我们可以将每张图片投影到一个低维空间中，从而减少计算的复杂性。更重要的是，即便在降维之后，这些图片之间的相似性（通常用向量之间的距离来度量）仍然能够得到较好的保持，使我们能够利用降维后的数据进行高效的图像分类和搜索。

3.1.6 数据的线性可分性

尽管在机器学习中处理高维数据时往往会面临诸多挑战，但高维空间也带来了一些独特的优势。其中一个显著优势是增强了数据的线性可分性，使得诸如传统机器学习中核方法（Kernel Methods）这样的技术变得更为有效。

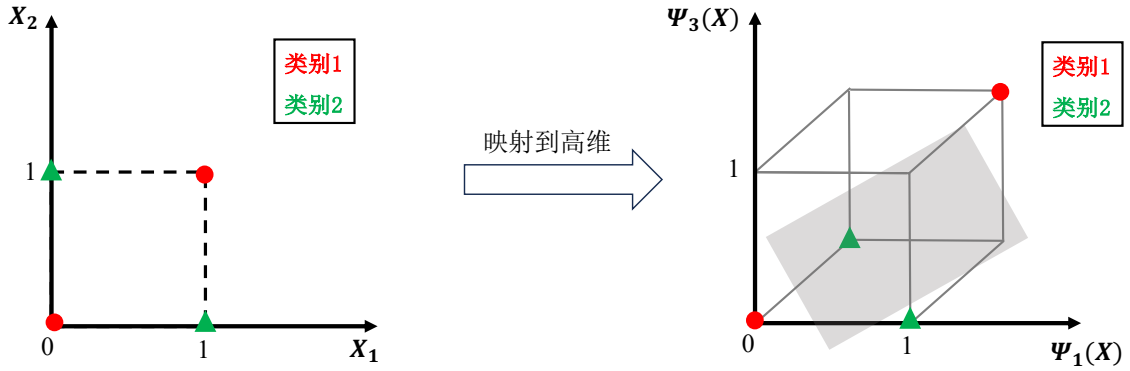


图 3.8: XOR 问题的二维和三维数据分布示例图

线性可分性指的是在空间中存在一个超平面（在二维中是一条直线，在三维中是一个平面），能够将不同类别的数据点完全分开。我们以经典的 XOR 问题为例来探讨这一现象。如图 3.8 所示，在二维空间中，我们有两类点，红色和绿色。这些点彼此交错，无法用一条直线将它们分开。然而，神奇的事情发生了：当我们将数据映射到更高的三维空间时，这些点变得线性可分。这是因为在高维空间中有更多的“自由度”或“维度”来找到一个可以分开不同类别数据的超平面。

核方法就是充分利用高维空间线性可分性的一种经典技术。核方法通过将原始低维数据进行非线性映射，使其嵌入到一个高维空间中。在这个高维空间里，数据通常会变得线性可分，然后我们可以在高维空间中应用线性分类器，如支持向量机（SVM）。这种方法背后的原理是，

通过增加新的维度，我们为数据点提供了额外的“活动空间”，使得在新空间中找一个平面来分隔这些点变得可行。这种技术不仅限于 XOR 问题，在许多实际应用中，我们常常利用类似的方法来解决线性不可分的问题。

核方法通过引入核函数，实现了这种非线性映射，使得原本在低维空间中难以分离的数据在高维空间中变得可分。神经网络模型也可以理解为一种利用非线性变换来增强数据线性可分性的技术。事实上，可以将神经网络看作是一系列复杂的非线性变换堆叠在一起，其目标是在网络的倒数第二层将数据变得线性可分，然后通过最后一层的线性分类器进行分类。

3.2 维数灾难的例子

维数灾难是指当数据的维度增高时, 为了保持模型的精度, 算法所需的参数量或数据量会随维数呈指数级增长。维数灾难是高维空间的一个普遍现象, 它对许多领域的问题都造成了显著的影响。本节我们将通过一些具体的例子来展示维数灾难的表现和后果, 让读者对这一概念有更加直观和深刻的理解。

3.2.1 高维数值积分

数值积分是应用数学中的一个重要工具, 它允许我们近似计算定义在某区域上的函数的积分。在一维空间中, 有许多高效的数值积分方法, 例如梯形法则、辛普森法则和高斯求积法。这些方法通过将积分区间划分为若干个小区间, 然后在每个小区间上用简单函数 (如多项式) 来近似原函数, 从而将连续的积分转化为离散的求和。

然而, 当我们尝试将这些方法推广到高维空间时, 我们会遇到严重的问题。考虑一个简单的例子, 我们希望在 d 维单位超立方体 $[0,1]^d$ 上计算一个函数 $f(\boldsymbol{x})$ 的积分, 积分可以写为:

$$I = \int_{[0,1]^d} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}.$$

在一维情况下, 我们可以使用 n 个等距节点的网格来近似这个积分, 每个节点的间距为 $1/n$。这个方法被称为复合梯形法则。通过将 $[0,1]$ 区间划分为 n 个小区间, 复合梯形法可以将积分近似为:

$$I \approx \frac{1}{n}\left(\frac{f(x_0)+f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right),$$

其中 $x_i = i/n$ 是第 i 个节点的坐标。对于光滑的被积函数 $f(x)$, 复合梯形法则的误差通常与 n^{-2} 成正比。这意味着, 如果我们将节点数 n 增加 10 倍, 积分的近似误差就会减小大约 100 倍。

现在让我们考虑将复合梯形法则推广到 d 维空间。一种自然的推广方式是, 在每个坐标轴上均匀放置 n 个节点, 从而得到一个 d 维的网格。这个网格将整个积分区域 $[0,1]^d$ 划分为 n^d 个小的超立方体, 每个超立方体的边长为 $1/n$ 。如果我们在每个超立方体上应用一维的复合梯形法则, 就可以得到整个积分的一个近似:

$$I \approx \left(\frac{1}{n}\right)^d \sum_{i_1=0}^{n} \cdots \sum_{i_d=0}^{n} w(i_1,\ldots,i_d) f(x_{i_1},\ldots,x_{i_d}),$$

其中 $w(i_1,\ldots,i_d)$ 是权重函数, $x_{i_j} = i_j/n$ 是第 j 维上的第 i_j 个节点的坐标。

乍一看, 这似乎是一个合理的推广。但是, 让我们仔细看看这个方法需要多少个节点。在一维情况下, 我们需要 n 个节点。在二维情况下, 我们需要在一个 $n \times n$ 的网格上采样, 总共需要 n^2 个节点。在 d 维的情况下, 我们需要在一个 $n \times n \times \cdots \times n$ 的网格上采样, 总共需要 n^d 个节点。这就是维数灾难的直观体现。节点数量, 也就是计算积分所需的函数评估次数, 随着维数 d 的增加呈指数级增长。即使对于相对较小的 d 和 n , 计算量也会迅速变得巨大, 计算变得不可行。举个具体的例子, 假设我们要计算一个 10 维空间上的积分, 如果我们在每个维度上使用 10 个节点, 那么总共需要 $10^{10} = 10,000,000,000$ 个节点。这意味着我们需要评估函数 $f(\boldsymbol{x})$ 100 亿次! 即使每次评估只需要 1 纳秒, 完成所有评估也需要 10 秒钟。而如果维度增加到 20, 同样使用每维 10 个节点, 那么我们需要 10^{20} 次函数评估, 这在实际中是完全不可行的。

这个简单的例子揭示了高维问题的一个基本困难: 计算复杂性随维数呈指数级增长, 即所谓的 “维数灾难”。许多在低维空间中行之有效的算法, 在高维空间中都会遇到指数爆炸的障碍。这也是许多传统的算法在处理高维数据时经常遇到困难的一个重要原因。

3.2.2 高维偏微分方程

高维偏微分方程在现代科学和工程领域有着广泛而重要的应用, 但同时也面临着计算复杂度的巨大挑战。让我们通过一些具体的例子来深入了解这些应用和挑战:

量子物理中的薛定谔方程 在量子物理中, 描述由 d 个粒子组成的量子系统的波函数满足薛定谔方程, 而这个方程恰好是一个定义在 $\mathbb{R}^{3d}$ 高维空间上的偏微分方程。也就是说, 即使对于一个只含 $d = 10$ 个粒子的量子系统, 其薛定谔方程所对应的空间维数也已经高达 30 维。随着粒子数 d 的进一步增加, 薛定谔方程的维数将变得更加庞大。这给利用经典数值方法求解薛定谔方程带来了极大的困难。

金融工程中的 Black-Scholes 方程 在金融工程领域, 期权定价是一个核心问题, 而 Black Scholes 偏微分方程是解决期权定价问题的重要工具。然而, Black-Scholes 方程的空间维数正好对应于所考虑的金融资产的数量。以一个由 100 种资产构成的投资组合为例, 其对应的

Black-Scholes 方程就是一个 100 维的偏微分方程。可以想见，对于一个更加复杂的金融衍生产品，所涉及的资产数量可能远远超过 100 种，由此导出的 Black-Scholes 方程维数也将变得极高。这对于金融工程师利用数值方法进行期权定价带来了很大挑战。

为了从直观上理解高维偏微分方程中的维数灾难，让我们考虑求解 d 维单位立方体 $[0,1]^d$ 上的一个简单的 Poisson 方程：

$$\begin{cases} -\Delta u = f, & x \in \Omega, \\ u = 0, & x \in \partial\Omega. \end{cases}$$

这里 Ω 表示单位立方体的内部，$\partial\Omega$ 表示单位立方体的边界。

如果用有限差分方法离散这个方程，在每个坐标方向上均匀剖分 n 个网格，那么离散后方程的未知量 u 将有 $N = n^d$ 个自由度。可以看到，离散方程的规模 N 会随着空间维数 d 呈指数增长。这意味着当维数 d 较大时，即使 n 取一个较小的值，N 也将是一个天文数字。这导致即使对于中等规模的高维问题，经典的数值方法也可能变得在计算资源上不可承受。

高维偏微分方程在许多领域都有重要应用，但求解这些方程必须克服“维数灾难”带来的挑战。经典数值方法在高维问题中可能完全失效，因此亟需发展新的理论和方法。我们将在后面的章节讨论用神经网络解决高维问题的方法，以期为这一难题提供新的思路。

3.2.3 高维函数逼近

在许多机器学习任务中，如回归和分类，我们的目标是从样本数据中学习一个从高维输入空间到输出空间的映射函数。输出空间可以是实数（对于回归问题），也可以是一个有限的类别集合（对于分类问题）。函数逼近理论研究的是，如何找到一个函数，使其在给定的数据上能够尽可能接近目标函数。

经典的函数逼近理论告诉我们，在高维空间中，逼近一个目标函数的难度会随着维度的增加而指数上升。换句话说，随着维度 d 的增加，样本数量 n 必须以指数级的速度增长，才能保持相同的逼近精度。具体来说，如果我们有一个目标函数 f^*，并且我们通过采样 n 个样本点来逼近它。那么在高维空间中，我们的逼近误差 ε（即我们找到的逼近函数和真实目标函数之间的差距）大致遵循如下规律 (DeVore 1998)：

$$\varepsilon \approx n^{-s/d} \tag{3.6}$$

这里，s 是目标函数的光滑程度，d 是数据的维度，n 是样本数量。随着维度 d 的增加，样本数量 n 必须指数增加，才能保持相同的逼近误差。高维函数逼近面临的一个主要问题是所谓的“维数灾难”。即使目标函数非常光滑 (即 s 很大)，要在高维空间中逼近它，我们仍需要大量的样本。这使得在实际应用中高效地学习和逼近高维函数变得非常困难。

为了更直观地理解高维函数逼近问题，我们通过一个具体的例子来说明光滑性和维数对函数逼近的影响。首先，我们来看目标函数的光滑性如何影响我们的逼近效果。如果目标函数 f^* 是一个线性函数，由于线性函数变化平滑，意味着我们需要的样本点数量相对较少。通过少数几个点，我们就能很好地逼近这个函数。但是，如果目标函数的光滑性很差，函数值在不同区域变化剧烈。为了准确地逼近这样的函数，我们需要采样更多的样本点，尤其是集中在变化剧烈的区域，以确保逼近误差小。

接下来我们考虑维数升高带来的影响。假设我们的输入空间是一个 d 维单位立方体，即每个维度都在 $[0,1]$ 范围内。我们希望用 n 个样本点来逼近一个在这个空间中的目标函数。当 $d=1$ 时，我们可以在 $[0,1]$ 区间内均匀分布 n 个样本点。这样，两个相邻样本点之间的距离大约是 $1/n$。由于目标函数是光滑的，我们可以较好地逼近它，在样本点之间差距不大的情况下，逼近误差会比较小，大约是 n^{-s}。然而，当维度 d 增加时，情况变得非常复杂。在 d 维空间中，每个维度上的样本点大约为 $n^{1/d}$，因此样本点之间的距离大约是 $n^{-1/d}$。随着维度 d 增加，每一维上的样本点数目急剧减小，样本点之间的距离变得很大，使得我们难以捕捉到目标函数的变化。因此，逼近误差变得很大，大约是 $n^{-s/d}$，维数越高，逼近误差越大。

高维函数逼近问题的维数灾难告诉我们，在高维情形下，即使目标函数非常光滑，我们也需要随着维数指数级增长的样本数量才能保证一定的逼近精度。这对机器学习算法提出了很大的挑战，因为实际获得的训练样本数量往往远小于理论上所需的样本数量。然而，尽管一般情况下高维函数逼近的样本复杂度很高，但对于某些特定的函数空间，神经网络可以在较少的样本上达到良好的逼近效果，我们将在后面的部分讨论关于神经网络克服维数灾难的理解。

3.3 克服维数灾难的方法

3.3.1 蒙特卡洛方法

蒙特卡洛 (Monte Carlo) 方法是一类基于随机抽样的数值方法，它利用概率和统计的思想来解决高维问题，从而克服维数灾难。与传统的确定性算法不同，蒙特卡洛方法引入了随机性，通过在问题的解空间中随机采样，并对这些随机样本进行统计平均，来估计所要求的量。

蒙特卡洛方法最初由乌拉姆 (Stanislaw Ulam)、冯 · 诺伊曼 (John von Neumann) 等人在研究原子弹设计时提出，其名称来源于著名的赌城蒙特卡洛，象征着其中的随机性和博弈性质。自 20 世纪 40 年代诞生以来，蒙特卡洛方法得到了蓬勃发展，并在物理、化学、生物、工程、金融等诸多领域得到了广泛应用。

蒙特卡洛方法为何能克服维数灾难呢? 我们通过一个简单的例子来说明。考虑计算 d 维单位立方体 $[0,1]^d$ 上的积分

$$I = \int_{[0,1]^d} f(\boldsymbol{x})\, dx. \tag{3.7}$$

如果用均匀采样的蒙特卡洛方法来估计这个积分，即从 $[0,1]^d$ 中独立地抽取 n 个随机样本点 $\{x_i\}_{i=1}^n$，然后用样本均值来近似积分：

$$I_n = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}_i). \tag{3.8}$$

不难验证，I_n 是 I 的无偏估计，即 $\mathbb{E}[I_n] = I$。而 I_n 的方差为

$$\mathrm{Var}(I_n) = \frac{\mathrm{Var}(f)}{n}, \tag{3.9}$$

其中 $\mathrm{Var}(f)$ 是被积函数 f 在 $[0,1]^d$ 上的方差，定义为：

$$\mathrm{Var}(f) = \int_{[0,1]^d} \left(f(\boldsymbol{x}) - \mathbb{E}[f(\boldsymbol{x})]\right)^2 d\boldsymbol{x}. \tag{3.10}$$

因此，蒙特卡洛估计的均方误差为

$$\mathbb{E}\left[(I - I_n)^2\right] = \frac{\mathrm{Var}(f)}{n}. \tag{3.11}$$

可见，这个误差仅依赖于样本数 n 和被积函数的方差，而与维数 d 无关。换句话说，不管问题的维数有多高，蒙特卡洛方法都能以 $O(n^{-1/2})$ 的收敛速度逼近积分。这种维数无关性，正是蒙特卡洛方法克服维数灾难的根本原因。

直观地说，蒙特卡洛方法之所以能避免维数灾难，是因为它并不试图用一个规则的网格来填充整个高维空间，而是随机地在这个空间中撒下一些采样点。在高维空间中，绝大部分的体积都集中在边界附近的一个狭小区域内。蒙特卡洛方法恰恰能自适应地将更多的样本点放置在这些区域中，而无需在对积分贡献很小的内部区域浪费计算资源。这种自适应性使得蒙特卡洛方法能高效地进行数值积分的计算，避免了维数灾难。

但是我们也必须指明，蒙特卡洛方法尽管克服了维数灾难，但是其采样效率在一些具体的问题中仍然存在不足。例如在我们前面考虑用神经网络逼近一个高维二次函数的实验中（图 3.2），蒙特卡洛方法自适应地在单位球的球壳附近采了大量的样本点，导致模型在积分意义下的泛化误差确实很小，但是当我们关注单位球内部靠近球心的那些测试点时，其泛化误差特别大。因此，对于蒙特卡洛采样效率高低的判断需要具体问题具体分析。在实际应用中，人们发展了许多改进的蒙特卡洛采样效率方法，如重要性采样、分层采样、马尔可夫链蒙特卡洛等，以进一步提高其采样效率。此外，蒙特卡洛的收敛率 $O(n^{-1/2})$ 虽然与维数无关，但是若要将误差降低一个数量级，所需的样本数仍然面临 100 倍的增加。这样的收敛率在低维问题中通常没有竞争力。

在围棋这样一个具体的高维问题中，随着落子数增加，棋局的可能性呈指数级增长，采样面临维数灾难。因此，通过穷举采样来训练算法评估棋局是不现实的。AlphaGo 的成功很大程

度上在于解决了高维采样困难。其核心思想首先是通过蒙特卡洛树搜索来避免维数灾难。其次是观察到每一步并非所有可下的位置都有同样的发生概率。可以用深度神经网络来学一个给定棋局下的采样概率分布，快速排除几乎不可能发生的无价值位置，专注于对有显著价值位置的随机采样和搜索。这也就是重要性采样的思想。通过这种方式，AlphaGo 实现了在庞大的状态空间中的高效采样和最优策略近似。

总的来说，蒙特卡洛方法是一类强大而通用的数值方法，尤其擅长处理高维问题。它通过随机抽样和统计平均的思想，巧妙地规避了维数灾难。在当前大数据时代，随着问题规模和复杂度的不断增加，蒙特卡洛方法及其思想正在发挥越来越重要的作用。

3.3.2 神经网络方法*

人工神经网络, 特别是深度学习模型, 在处理高维复杂任务时（如图像和自然语言任务）展现出了超越经典机器学习方法的优异表现。下面我们将介绍对于一类特殊的函数空间（Barron 空间）神经网络能够克服灾难的理论理解。

万有逼近定理 早在上世纪 80 年代末,Cybenko (Cybenko 1989) 和 Hornik (Hornik et al. 1989) 等人就发现了全连接神经网络的一个重要性质: 具有一个隐藏层的神经网络, 只要激活函数非多项式, 就能以任意精度逼近定义在紧集上的任意连续函数。这就是著名的**万有逼近定理**。

具体而言, 对任意给定的连续函数 $f : \mathbb{R}^d \to \mathbb{R}$ 和容许误差 $\epsilon > 0$, 都存在一个具有合适参数 $\boldsymbol{\theta}$ 的单隐藏层神经网络 $f_{\boldsymbol{\theta}}$, 使得

$$\sup_{x \in K} |f(\boldsymbol{x}) - f_{\boldsymbol{\theta}}(\boldsymbol{x})| < \epsilon, \tag{3.12}$$

其中 $K \subset \mathbb{R}^d$ 为任意紧集。

万有逼近定理揭示了即便是浅层神经网络, 原则上也具有学习任意复杂模式的能力。这一理论为神经网络的应用奠定了基础。然而, 它并没有回答这些更关键的问题: 为达到给定精度, 网络规模需要多大? 是指数增长的吗? 网络的逼近效率如何?

神经网络逼近 Barron 空间里的函数没有维数灾难 在实践中，我们观察到神经网络在许多高维问题上表现出色。那么，是否可以严格地证明神经网络在某些情况下可以避免维数灾难呢？为了从理论上理解神经网络为何能够避开维数灾难，我们引入一个关键概念：Barron 空间。Barron 空间是这样一类特别为神经网络量身定制的函数空间。它最早由 Barron 在 1993 年的开创性工作中引入 (Barron 1993), 并在论文 E et al. (2019*a*) 中进一步发展和完善。具体来说，论文 E et al. (2019*a*) 证明了对于属于 Barron 空间的函数，单隐层神经网络能够以较低的参数复杂度和较低的样本复杂度实现良好的逼近和泛化效果，而不受维数灾难的影响。这从

理论上明确了神经网络在处理一类高维问题时的优越性。在接下来的部分中，我们会详细讨论具体的理论细节，以进一步说明这一结果。

Barron 空间 形式化地，记 $\mathbb{S}^d := \{\boldsymbol{w} \mid \|\boldsymbol{w}\|_1 = 1\}$ 表示单位球面，$\mathcal{B}(X)$ 表示具有以下积分表示的函数集合：

$$f(\boldsymbol{x}) = \int_{\mathbb{S}^d} a(\boldsymbol{w})\sigma(\boldsymbol{w}^\top \boldsymbol{x}) d\pi(\boldsymbol{w}) \quad \forall \boldsymbol{x} \in X, \tag{3.13}$$

其中 σ 为满足齐次性 $\sigma(\alpha t) = \alpha\sigma(t), \forall \alpha \geq 0$ 的非线性激活函数，例如 ReLU 和 Leaky ReLU，$\boldsymbol{w}$ 表示单位球面上的一个方向，$\pi(\boldsymbol{w})$ 表示概率分布，$a(\boldsymbol{w})$ 描述了 $\boldsymbol{w}$ 这个方向上的幅值。

对于任意 $f \in \mathcal{B}(X)$ 和 $p \geq 1$，我们定义以下范数（称为 **Barron 范数**）：

$$\gamma_p(f) := \inf_{(a,\pi)\in\Theta_f} \left(\int_{\mathbb{S}^d} |a(\boldsymbol{w})|^p \, \mathrm{d}\pi(\boldsymbol{w}) \right)^{1/p}, \tag{3.14}$$

其中

$$\Theta_f = \left\{ (a, \pi) \mid f(\boldsymbol{x}) = \int_{\mathbb{S}^d} a(\boldsymbol{w})\sigma(\boldsymbol{w}^\top \boldsymbol{x}) \, \mathrm{d}\pi(\boldsymbol{w}) \right\}. \tag{3.15}$$

借助 γ_p 范数, Barron 空间 $\mathcal{B}_p(X)$ 可如下形式化定义。

定义 1 (Barron 空间).

$$\mathcal{B}_p(X) := \{f \in \mathcal{B}(X) \mid \gamma_p(f) < \infty\}.$$

特别地, 当 $p = 2$ 时, 记 $\mathcal{B}_2(X)$ 为 L^2 型的 *Barron* 空间。

让我们以一种更直观的方式来理解一下 Barron 空间。假设 $f \in \mathcal{B}(\mathbb{R}^d)$，这意味着函数 f 可以表示成以下形式：

$$f(\boldsymbol{x}) = \int_{\mathbb{S}^d} a(\boldsymbol{w})\sigma(\boldsymbol{w}^\top \boldsymbol{x}) \, d\pi(\boldsymbol{w}). \tag{3.16}$$

从神经网络的角度来看，这种积分表示相当于一个无限宽的单隐层神经网络。每个隐层神经元对应着球面 $\mathbb{S}^d$ 上的一个方向 $\boldsymbol{w}$，并具有权重 $a(\boldsymbol{w})$。我们可以把这些权重想象成是神经元在不同方向上的 “贡献度”。不仅如此，实际上有限宽的两层神经网络也可以看作是 Barron 空间的一部分。具体来说，一个有限宽的两层神经网络表示可以写成：

$$f(\boldsymbol{x}; \theta) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}), \tag{3.17}$$

它对应的概率分布可以表示为：

$$\pi(\boldsymbol{w}) = \frac{1}{m} \sum_{k=1}^{m} \delta(\boldsymbol{w} - \boldsymbol{w}_k), \tag{3.18}$$

其中 $\delta(\boldsymbol{x})$ 是狄拉克 δ 函数。这意味着，任何有限宽的两层神经网络表示的函数也属于 Barron 空间 $\mathcal{B}_p(X)$。

万有逼近定理告诉我们，连续函数可以用有限宽的两层神经网络逼近。因此，Barron 空间 $\mathcal{B}_p(X)$ 在连续函数空间 $C(X)$ 中是稠密的，这意味着它能够逼近几乎所有的连续函数。简而言之，Barron 空间 $\mathcal{B}_p(X)$ 是一个非常大的函数空间。下面我们将看到神经网络逼近这一大类函数时均没有维数灾难。

Barron 空间的逼近理论 现在我们给出用两层神经网络逼近 Barron 空间函数的一个核心结果 (E et al. 2019*b*):

定理 2. *对任意的 $f \in \mathcal{B}_2(\mathbb{R}^d)$, 存在一个宽度为 m 的两层神经网络 $f(\cdot;\tilde{\boldsymbol{\theta}})$, 使得*

$$\begin{aligned} \mathbb{E}_{\boldsymbol{x}}\left[(f(\boldsymbol{x}) - f(\boldsymbol{x};\tilde{\boldsymbol{\theta}}))^2\right] &\leq \frac{3\gamma_2^2(f)}{m}, \\ \|\tilde{\boldsymbol{\theta}}\|_{\mathcal{P}} &\leq 2\gamma_2(f). \end{aligned} \tag{3.19}$$

其中 $\mathbb{E}_{\boldsymbol{x}}$ 表示对输入分布的期望, $\gamma_2(\cdot)$ 范数如前定义，而 $\|\boldsymbol{\theta}\|_{\mathcal{P}} := \sum_{k=1}^m |a_k| \|\boldsymbol{w}_k\|_1$ 是路径范数，它度量了从输入到输出的所有权重绝对值之和, 体现了网络的某种复杂度。

定理 2表明, 宽度为 m 的两层神经网络能以 $O(m^{-1/2})$ 的收敛速度逼近任意 $f \in \mathcal{B}_2(\mathbb{R}^d)$, 且所构造的网络参数在路径范数意义下是有界的。特别地, 这里逼近阶 $O(m^{-1/2})$ 与输入维数 d 无关, 这正是克服维数灾难的关键。

定理 2的证明巧妙地利用了 Barron 空间函数的积分表示和**蒙特卡洛随机抽样**的思想。具体而言, 对于任意的 $f \in \mathcal{B}_2(\mathbb{R}^d)$, 其 Barron 表示为

$$f(\boldsymbol{x}) = \int_{\mathbb{S}^d} a(\boldsymbol{w})\sigma(\boldsymbol{w}^\top \boldsymbol{x}) d\pi(\boldsymbol{w}). \tag{3.20}$$

若从分布 π 中独立抽取 m 个样本 $\{\boldsymbol{w}_k\}_{k=1}^m$, 则 f 可被如下两层神经网络逼近:

$$f(\boldsymbol{x};\tilde{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{k=1}^m a(\boldsymbol{w}_k)\sigma(\boldsymbol{w}_k^\top \boldsymbol{x}). \tag{3.21}$$

容易验证 $\mathbb{E}_{\boldsymbol{w}}[f(\boldsymbol{x};\tilde{\boldsymbol{\theta}})] = f(\boldsymbol{x})$, 即 $f(\boldsymbol{x};\tilde{\boldsymbol{\theta}})$ 的期望等于 $f(\boldsymbol{x})$。进一步我们也可以控制方差 $\mathbb{E}_{\boldsymbol{x}}[(f(\boldsymbol{x}) - f(\boldsymbol{x};\tilde{\boldsymbol{\theta}}))^2]$ 的上界，从而得到定理 2 的结论，具体细节可以参考 (E et al. 2019*b*)。

定理 2的意义在于: 它揭示了有限宽度的两层神经网络对 Barron 空间的逼近能力, 而这种逼近能力在高维情形下并不会退化，即参数复杂度与维数 d 无关，因此避免了维数灾难。

Barron 空间的泛化理论 在上一节中, 我们基于 Barron 空间刻画了两层神经网络的逼近能力。自然而然的问题是: 这种逼近能力能否转化为网络的泛化能力, 即从有限训练样本出发学习未知目标函数的能力? 本节将围绕这一问题展开讨论。

考虑如下的无噪声学习问题。给定训练数据

$$y_i = f^*(\boldsymbol{x}_i), \quad i = 1, \ldots, n,$$

其中 $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ 为从某一未知分布 P 中独立抽取的样本, $f^* \in \mathcal{B}_2(X)$ 为我们希望学习的目标函数。我们的目标是利用观测数据 $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ 来估计 f^*。

为了理论分析方便, 设 $\hat{\boldsymbol{\theta}}_{n,\lambda}$ 为如下加了路径范数正则化后的经验风险最小化 (Empirical Risk Minimization, ERM) 的解:

$$\hat{\boldsymbol{\theta}}_{n,\lambda} = \arg\min_{\theta} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i; \boldsymbol{\theta}))^2 + \lambda \|\boldsymbol{\theta}\|_{\mathcal{P}} \right\}, \tag{3.22}$$

其中 $\lambda > 0$ 为正则化参数,$\|\cdot\|_{\mathcal{P}}$ 为路径范数, 定义如前。记 $f(\cdot; \hat{\boldsymbol{\theta}}_{n,\lambda})$ 为 ERM 得到的两层神经网络。

为刻画 $f(\cdot; \hat{\boldsymbol{\theta}}_{n,\lambda})$ 的泛化能力, 我们引入以下记号。令 $\hat{\gamma}_p(f) = \max\{1, \gamma_p(f)\}$, 其中 $\gamma_p(\cdot)$ 为 Barron 范数。取正则化参数 $\lambda_n = 4\sqrt{\frac{2\ln(2d)}{n}}$, 这里我们假设 $\ln(2d) \geq 1$。

有了这些准备, 我们给出两层神经网络泛化能力的一个主要定理 (E et al. 2019*b*):

定理 3. *假定目标函数 $f^* \in \mathcal{B}_2(X)$, 并且 $\lambda \geq \lambda_n$, 则对任意 $\delta > 0$, 以至少 $1 - \delta$ 的概率,ERM 得到的两层神经网络 $f(\cdot; \hat{\boldsymbol{\theta}}_{n,\lambda})$ 满足*

$$\mathbb{E}_{\boldsymbol{x}} \left| f\left(\boldsymbol{x}; \hat{\boldsymbol{\theta}}_{n,\lambda}\right) - f^*(\boldsymbol{x}) \right|^2 \lesssim \frac{\hat{\gamma}_2^2(f^*)}{m} + \lambda \hat{\gamma}_2(f^*) + \frac{1}{\sqrt{n}} \left(\hat{\gamma}_2(f^*) + \sqrt{\ln(n/\delta)} \right), \tag{3.23}$$

其中 $\lesssim$ 表示存在与 n, d, δ 无关的正常数使不等式成立。

定理 3给出了两层神经网络学习 Barron 空间函数的泛化误差界。不等式 (3.23)右端的三项分别对应逼近误差、正则化误差和样本复杂度。我们逐一讨论之。首先, 第一项逼近误差 $\frac{\hat{\gamma}_2^2(f^*)}{m}$ 度量了宽度为 m 的两层网络对目标函数 f^* 的逼近能力。这一项可以利用定理 2的结果直接控制。其次, 第二项中 $\hat{\gamma}_2(f^*)$ 反映了目标函数 f^* 的复杂度, 由 f^* 在 Barron 空间中的范数刻画。最后, 样本复杂度 $\frac{1}{\sqrt{n}} \left(\hat{\gamma}_2(f^*) + \sqrt{\ln(n/\delta)} \right)$ 刻画了泛化误差随训练样本数 n 的衰减速度, 忽略对数因子, 当 n 充分大时, 这一项反映了整体的收敛速度为 $O(n^{-1/4})$。

定理 3表明, ERM 所得的两层神经网络能以 $O(n^{-1/4})$ 的速度学习 Barron 空间中的目标函数, 这里的样本复杂度与维数 d 无关, 从而从理论上证明了两层神经网络的泛化能力, 揭示了其在高维数据学习中的独特优势。

Barron 空间为我们提供了一个理解神经网络如何克服维数灾难的视角。传统的机器学习方法之所以在高维数据上表现失败, 核心在于其参数复杂度或者样本复杂度随维数呈指数增长, 这通常是不可接受的。而神经网络在某些情况下能避开这一诅咒, 关键在于所要逼近的函数空间具有某些特殊的性质, Barron 空间就是这样的一个特殊例子。对于属于 Barron 空间的函数, 单隐层神经网络能够以较低的参数复杂度和较低的样本复杂度实现良好的逼近和泛化, 而不受维数灾难的影响。这给了我们一个重要的启示: 如果我们能够找到更适合刻画真实世界数据的函数空间, 并设计出能够有效逼近这些函数空间的神经网络结构, 那么我们就有希望在高维情形下获得良好的学习效果。

深度学习为人工智能从低维感知走向高维认知开辟了一条充满希望的道路。随着数据维数和复杂性的不断提高, 对高维数据的处理已成为现代人工智能的核心课题。而以深度神经网络为代表的新一代学习范式, 无疑将在这一重大挑战中扮演越来越重要的角色。尽管目前对深度学习的理论认识尚不完备, 但大量实践应用已经展现了其惊人的潜力。未来, 相信深度学习必将在更多高维复杂场景大放异彩。

3.4 习题

1. 什么是维数灾难? 请举例说明维数灾难可能带来的问题。

2. 证明 d 维单位球的表面积 $A(d)$ 和体积 $V(d)$ 公式分别为:

$$A(d) = \frac{2\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}\right)} \quad \text{and} \quad V(d) = \frac{2\pi^{\frac{d}{2}}}{d\Gamma\left(\frac{d}{2}\right)}$$

   其中 $\Gamma(x)$ 为伽马函数。(提示: 可以考虑使用球坐标系)

3. 请绘制出 $d = 5, 10, 20$ 时 d 维球体体积 $V(d)$ 随半径 $r \in [0, 1]$ 的变化曲线, 分析维数 d 增大时的影响。

4. 随机在 d 维空间的单位立方体 $[0, 1]^d$ 中采样, 请计算采样点落在单位球内部的概率。

5. 证明蒙特卡洛方法的收敛阶是 $O(n^{-1/2})$, 与维数 d 无关。此外, 请思考即使蒙特卡洛采样的数据点个数 n 已经很大了, 但我们在使用神经网络学习一个高维二次函数时在某些点的泛化误差依然很大, 这是为什么?

Chapter 4

频率原则

在前面的章节中, 我们介绍了深度学习的一些基本知识, 也指出了理解神经网络这样一个复杂系统的挑战性。因此, 找到一个合适的切入点去分析神经网络, 对于剖析这个” 黑盒子” 的内部机制尤为重要。在科学研究中, 人们对复杂系统的探索往往起源于对基本现象的观察和分析。例如, 在遗传学研究中, 孟德尔选择了相对简单的豌豆作为研究对象, 通过观察豌豆性状的分离和组合规律, 总结出了遗传的基本定律。这些定律后来被应用于解释更加复杂的生物遗传现象。再如, 在电磁学研究中, 安培和法拉第等科学家首先研究了相对简单的电场和磁场现象, 发现了电磁感应定律等基本规律。在此基础上, 麦克斯韦进一步将这些规律统一为麦克斯韦方程, 解释了更复杂的电磁现象, 预言了电磁波的存在。

受此启发, 我们在研究神经网络时, 也可以先从一些非常简单的问题入手, 通过在简单模型中观察、分析、总结规律, 再尝试将这些认识推广到更加复杂的神经网络中。当然, 这种推广需要谨慎, 并非所有从简单模型中得到的结论都能直接应用于复杂模型。但这种研究方法可以帮助我们抓住问题的关键, 为进一步探索复杂网络提供有益的思路和启发。基于上述考虑, 在神经网络的实验探究过程中, 我们选择从最简单的拟合一维函数的问题开始进行研究。一方面, 一维问题结构简单、直观, 便于观察分析网络的特性; 另一方面, 很多神经网络的基本特性, 如学习过程的非线性、在常规设定下不易过拟合等, 在一维问题中就已经能够得到充分的体现。通过对一维问题的研究, 我们可以初步认识神经网络的一些重要特点, 为进一步探索更加复杂的网络打下基础。因此, 在本节中, 我们将基于一维问题展开讨论, 通过具体的实例来向读者展示神经网络在学习过程中的一般规律。

科普篇

什么是神经网络的频率原则？

神经网络的频率原则（Frequency Principle，F-Principle），也常被称为谱偏差（spectral bias），是近年来深度学习领域的一个重要发现。该原则描述了神经网络在学习过程中呈现出的一种特定模式：倾向于先学习数据中的低频信息，随后逐步过渡到高频信息的学习。这一原则为我们理解神经网络的优化过程和性能提供了新的视角。

什么是频率？

在信号处理和数据分析中，频率指的是信号或数据在时间或空间维度上变化的快慢程度。从数学角度来看，低频成分代表变化缓慢的整体趋势或模式，而高频成分则对应变化迅速的细节和局部特征。例如，一个平滑的曲线主要包含低频成分，而具有尖锐变化的波形则包含较多的高频成分。

频率原则的内涵是什么？

频率原则揭示了神经网络在训练过程中的一个重要特性：网络倾向于先学习和拟合数据中的低频信息，随后逐渐捕捉和适应高频信息。这一过程可以分为以下几个阶段：

- 初始阶段：在训练的早期，神经网络主要学习数据中的低频成分，即整体趋势和主要模式。此时，网络参数对数据的宏观变化较为敏感。
- 中间阶段：随着训练的进行，网络开始捕捉更多的细节信息，这些信息通常包含在中频成分中。此阶段，网络的预测能力逐渐提高，表现出更精细的特征识别能力。
- 后期阶段：在训练的后期，网络能够学习到高频成分，这些成分可能包含数据中的微小变化和噪声。此时，网络在处理高频信息时表现出对细微模式和异常点的敏感性。

频率原则有何重要性？

频率原则的发现对深度学习研究和应用具有多方面的重要意义：

- 理解泛化能力：通过先学习低频信息，神经网络可以构建一个相对稳健的基础模型，有助于减少过拟合风险。过拟合是指模型在训练数据上表现优异，但在新数据上泛化性能较差的现象。
- 深化学习机制理解：频率原则为研究人员提供了一个新的角度来解析神经网络的学习机制，有助于改进现有的训练方法和模型架构设计，以提升高频学习的效率。

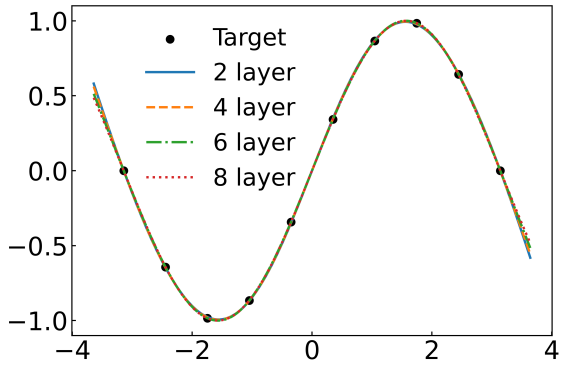


图 4.1: 不同规模的深度神经网络（DNN）学习到的函数输出

频率原则对于实际有何指导？

在实际应用中，对频率原则的理解可以帮助我们：

- 优化训练策略，如提前停止等。
- 设计更有效的特征提取方法，兼顾低频和高频信息的捕捉。

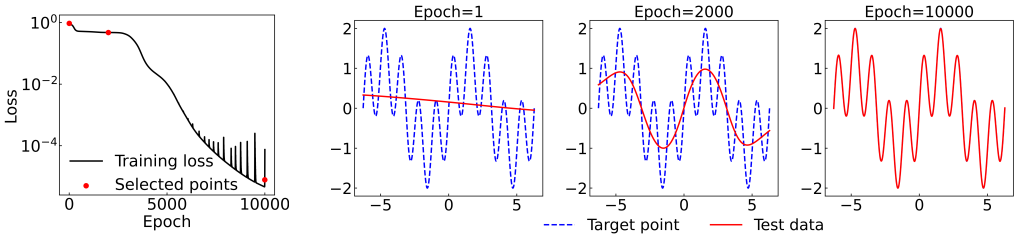
4.1 频率原则的一维实验

4.1.1 神经网络的“光滑”偏好

一维问题的一个优点在于，我们可以直接观察神经网络输出函数（即其“输入-输出”映射）的图像，直观认识输出函数的整体性质，如光滑度、拟合误差。我们观察了不同深度网络模型的学习结果，如图4.1所示，黑色点表示从目标函数 $\sin(x)$ 中采样的训练数据。不同颜色的线代表不同规模神经网络的函数输出，它们的一个显著特点是在训练数据点间形成光滑连线。这种神经网络对于光滑函数的“隐式”偏好引发了我们的好奇：为什么在拟合过程中，不同规模的神经网络都会倾向于学到光滑的函数?

为了探究神经网络对训练点的光滑拟合是怎么产生的，我们对其训练的全过程进行追踪。我们构造了一个由光滑的低频函数 $\sin(x)$ 和振荡剧烈的高频函数 $\sin(5x)$ 叠加而成的目标函数，并在目标函数中采点进行拟合。如图4.2所示, 蓝色虚线表示目标函数 $\sin(x)+\sin(5x)$，红色实线表示网络在该训练步数（$t = 1, 2000, 10000$）时神经网络的输出。在训练的前期（2000步时），神经网络学到的曲线基本与目标函数中 $\sin(x)$ 分量非常靠近, 而没有拟合剧烈振荡部

分。由此，我们对神经网络的学习过程产生了以下直观感受: 神经网络往往先快速捕捉到训练集的整体轮廓, 而后经过较长时间的训练, 才开始逐步拟合更多的细节信息。同时，在 $\sin(x)$ 分量拟合好的位置，在 loss 图中可以看到明显的损失停滞现象。

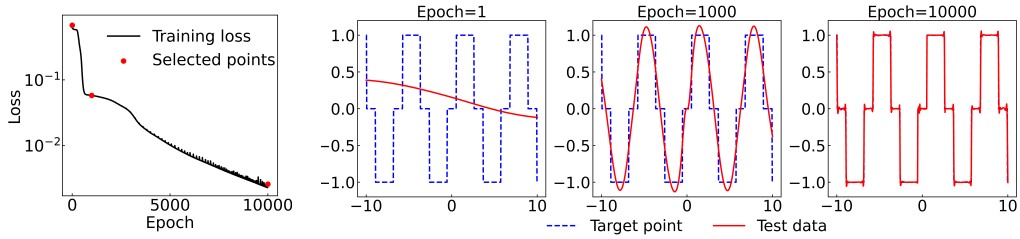


(a) loss　　(b) model output

图 4.2: 深度神经网络（DNN）训练过程的示意图

在分类问题中, 目标函数通常是离散的，在不同的自变量取值范围内函数值为一个常数。为了研究神经网络拟合这类函数的过程, 我们构造了如图4.3中红色实线所示的目标函数。其中蓝色虚线表示目标函数，红色实线表示网络在该训练步数（$t = 0, 1000, 10000$）时神经网络的输出。可以发现神经网络首先捕捉了低频成分，得到了形如三角函数的输出，然后开始捕捉到目标函数的高频特性，逐渐学到目标函数中的“台阶”部分。这个函数是一个周期性的跳跃函数, 其输出只有三个离散值, 可以类比为三个不同的类别标签。为了简化问题, 我们没有使用常见的交叉熵损失函数, 而是采用了均方误差作为损失函数，将问题当作回归任务来处理。为了更清晰地观察神经网络拟合（高频）跳跃的过程, 我们使用了非常密集的数据点, 以确保训练数据能尽可能保留目标函数中的高频信息。从图中可以看出, 神经网络在训练过程中呈现出有规律的学习模式。首先, 网络快速学习并捕捉到了数据的整体轮廓。随后, 网络开始逐步学习数据的细节信息, 慢慢捕捉到那些跳跃变化的部分。最终, 网络输出逐渐逼近了真实的目标函数。这个实验再次展示了神经网络先整体后局部的学习规律。尽管目标函数是非连续的跳跃函数, 但网络仍然能够通过回归的方式来逼近它。事实上，当时选择这类非连续的跳跃函数是一种偶然。但回顾来看，利用这种非连续跳跃的目标函数来观察神经网络的训练过程却很有好处：在频率空间，这个函数由多个频率不同的谐波构成，容易观察不同频率成分收敛速率的差异。

在二维拟合任务中, 我们同样可以观察到神经网络优先学习整体轮廓的特性。以拟合二维图像为例, 一张黑白照片实际上可以看作一个二维函数, 其自变量为像素坐标, 因变量为对应位置的灰度值。如果一个神经网络完全记住了这张图片, 我们可以将每个像素坐标依次输入到网络中, 得到对应位置的灰度值输出, 最后将它们拼合起来就可以还原出原图。对于彩色图片, 由于有红、绿、蓝三个通道 (RGB), 我们可以使用一个具有两维输入和三维输出的神经网络来拟



(a) loss (b) model output

图 4.3: 深度神经网络（DNN）训练过程的示意图。

合。

图4.4展示了使用全连接网络拟合一张图片的训练过程, 其中训练点为原图中奇数列像素点, 并观察模型在奇数列（训练数据）以及偶数列（测试数据）上的模型输出。我们发现, 在训练步数为 20000 时, 网络只能捕捉到图中猫的大致位置信息; 当步数达到 50000 时, 面部等轮廓信息已经基本还原; 直到训练步数达到 1000000, 网络才能够捕捉到毛发等细节, 对整张图片有了较为完整的还原。通过观察这个训练过程, 我们可以看出神经网络在学习二维图像时并非先记住某一部分再去记另一部分, 而是先学习到一个模糊的整体轮廓, 再逐步添加细节信息, 最终得到清晰完整的图像。这表明, 神经网络的学习过程并非毫无章法的 “死记硬背”，而是遵循先整体后局部的原则。

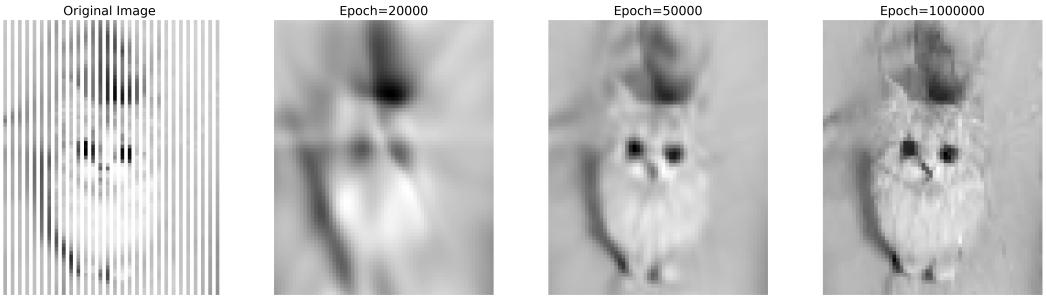


图 4.4: 神经网络拟合图像的过程

进一步，我们也可以用神经网络来重构三维物体。比如给一堆点云数据，也就是一个物体的三维坐标和强度值，神经网络的拟合过程与上述例子相似。它往往能快速拟合物体的轮廓，但在较少的训练轮次内很难重构细节部分。

神经网络的这种先整体后局部的学习模式, 与我们人类认识陌生事物的过程颇为类似。当

我们接触一个新事物时, 往往也是先掌握其大致轮廓, 再逐步了解更多细节。这种从简单变复杂的拟合训练数据的过程, 为我们理解神经网络提供了一个有趣的视角。然而, 要想更加精细、定量地分析这个过程，我们还需要将轮廓和细节这样的直观描述转化为可实验观测的量。

频率原则指出神经网络在学习低频的速度相比高频快很多，既表明它在学习低频的优势，也指出其在学习高频时的困难, 比如求解微分方程或者重构高清图像时, 高频细节常常很重要。对于这些问题，一般的神经网络学习就很困难。在频率原则的启发下，许多相关的算法被提出来提升高频的学习，并得到广泛的应用，比如 PhaseDNN Cai et al. (2020)，多尺度神经网络（MscaleDNN）Liu et al. (2020)，傅里叶特征网络 Tancik et al. (2020)，神经辐射场 Mildenhall et al. (2020)，隐式神经网络表示 Sitzmann et al. (2020) 等。后面的章节会有进一步的介绍。

4.1.2 频率和傅里叶变换

通过前面的分析, 我们大致总结出神经网络学习的一般规律, 即先学习轮廓，再学习细节。为了定量地比较模型学习轮廓和细节的速度, 我们需要引入一个量来刻画神经网络在学习过程中对轮廓和细节的捕捉程度。在数学上, 描述 “轮廓” 与 “细节” 的一种常用方法就是频率。直观地说, 一个函数如果变化平缓, 我们称之为低频函数; 反之, 如果函数变化剧烈, 我们称之为高频函数。

为了定量刻画一个函数的频率特性, 我们通常采用傅里叶变换。傅里叶变换的核心思想是, 绝大多数函数都可以表示为不同频率的正弦函数和余弦函数的叠加。这里简单介绍傅里叶变换（更多的介绍请见附录）：

连续傅里叶变换 (Continuous Fourier Transform, CFT)： 考虑函数 $g(x)$ 从时域 x 空间到频率 ξ 空间的变换：

$$\mathcal{F}[g(x)](\xi) = \int_{-\infty}^{\infty} g(x)\mathrm{e}^{-2\pi\mathrm{i}\xi x}\mathrm{d}x, \tag{4.1}$$

这实际上是一种内积的形式。注意到两个函数的内积，$< u(x), v(x) >= \int u\bar{v}\mathrm{d}x$，后一个函数 $v(x)$ 需要取共轭，因此，傅里叶变换是将 $g(x)$ 投影到 $\mathrm{e}^{2\pi\mathrm{i}\xi x}$ 的方向上, 也就是分解到了基函数 sin 和 cos 上（因为 $\mathrm{e}^{2\pi\mathrm{i}\xi x} = \cos(2\pi\xi x) + \mathrm{i}\sin(2\pi\xi x)$）。如果 $g(x)$ 是一个低频平坦的函数, 那么其在低频方向上的投影分量将占主导; 反之, 如果 $g(x)$ 是一个高频振荡的函数, 那么其在高频方向上的投影分量将更为显著。

与之对应，我们也给出傅里叶逆变换的公式：

连续傅里叶逆变换 (Continuous Inverse Fourier Transform, CIFT)： 考虑函数 $\hat{g}(\xi)$ 从频率 ξ 空间到时域 x 空间的变换：

$$\mathcal{F}^{-1}[\hat{g}(\xi)](x) = \int_{-\infty}^{\infty} \hat{g}(\xi)\mathrm{e}^{2\pi\mathrm{i}\xi x}\mathrm{d}\xi. \tag{4.2}$$

为了直观展示傅里叶变换的效果, 我们以一张猫的图像为例。通过对图像进行傅里叶变换, 我们可以将其分解为不同频率的成分, 从而有效地区分出图像中的高频和低频信息。如图4.5所示, 傅里叶变换后的低频成分刻画了图像的整体轮廓和基本形状, 体现了图像的整体结构; 而高频成分则对应着图像中的细节信息, 如猫的毛发、眼睛等纹理和边缘特征。通过傅里叶变换, 我们可以将一个函数分解为不同频率的成分, 从而深入理解其在频域上的特性。这一工具为分析神经网络的学习行为提供了新的视角。

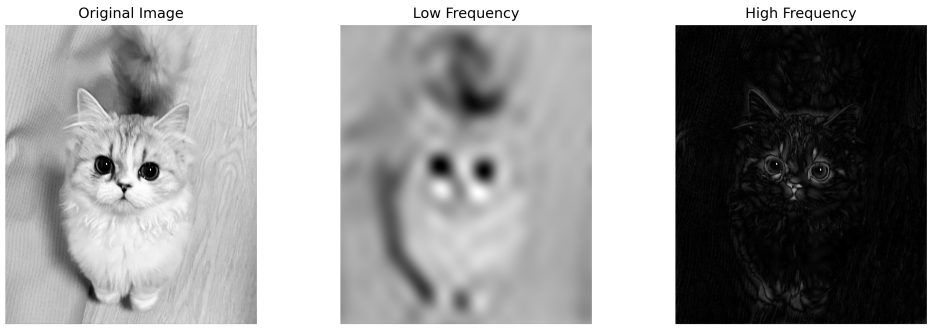


图 4.5: 图像傅里叶变换后分离出不同的频率成分

下面, 我们将进一步利用频率这一视角, 观察模型学习高频和低频成分的相对速度差异。通过比较模型在不同频率上的学习速度, 我们可以对神经网络在训练过程中的 “光滑” 偏好开展定量的实验研究。

4.1.3 频率原则

在上一节中, 我们引入了傅里叶变换这一强大的数学工具, 使我们能够定量地研究神经网络在学习过程中对不同频率成分的拟合速度。在本章中, 我们将展示神经网络在频域上的训练过程。从中，我们归纳出其训练过程的一个重要特征, 即:

频率原则: 神经网络常常会以从低频到高频的顺序拟合训练数据。

如图4.6所示，我们对图4.3中对应步数的目标函数及模型输出进行傅里叶变换，观察模型在频域空间的学习过程。蓝色的实线为目标函数的频率分布，红色的虚线为这个时刻神经网络拟合结果的频率分布，从左到右训练步数逐渐增加。从频域上的结果可以更直观地看出神经网络首先学习了数据的低频成分，然后随着训练步数的增多，学到了数据的高频成分。通过分析变换结果, 我们发现了一个有趣的现象: 深度神经网络 (DNN) 在拟合函数时, 往往首先学习到低频成分, 而高频成分则需要更长的训练时间才能被有效捕捉。随着训练的进行, 神经网络逐渐学习到越来越多的高频细节, 最终实现对整个函数的精确拟合。我们将这一现象总结为神经

网络训练过程的“频率原则”。

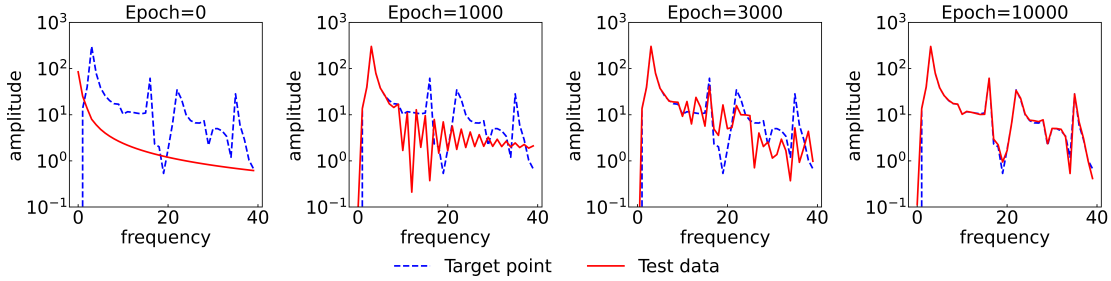


图 4.6: 神经网络输出的频率分布随时间的变化过程

根据频率原则，神经网络常常会以从低频到高频的顺序拟合训练数据。值得注意的是，当训练数据的所有频率成分都被捕捉时，训练误差将降为零，训练终止。因此，神经网络的拟合结果所包含的最高频率一般不会超过训练数据的最高频率。这也直观解释了我们此前的实验具体的图标记出来，即神经网络总是用比较光滑的函数拟合训练数据。

然而，细心的读者可能会发现，上述实验还存在一些缺陷，不足以确定不同成分的训练速度差异是由其频率高低引起的。具体来说，在图4.6中，我们观察到随着频率的增加，相应的幅值也在下降。因此，对上述现象另一种可能的解释是幅值大小决定了不同频率成分的训练速度。为了确定频率和幅值谁才是影响训练速度的关键因素，

我们针对性地设计了一个控制变量实验，观察神经网络拟合具有同幅值不同频率成分的目标函数的训练过程。我们选择目标函数 $f(x)=\sin(x)+\sin(3x)+\sin(5x)$，其中各频率成分的幅值均为 1。实验结果如图4.7所示，左图展示了目标函数，中图展示了目标函数及模型输出在频域空间分布，右图展示了不同频率成分下神经网络拟合结果的相对误差 $\Delta F(k)=\left|\hat{h}_k-\hat{f}_k\right|/\left|\hat{f}_k\right|$ 随训练步数的演化（其中，$\hat{h}_k$ 和 $\hat{f}_k$ 分别代表模型输出和真实值）。如中间图像所示，我们只考虑频率分布中的峰值位置，以避免数值计算导致的频率振幅的不准确。从图中可以看出，即使在幅值相同的情况下，低频成分依然表现出快得多的收敛速度。这帮助我们确定了频率是影响收敛速度的核心要素。

以上多个实验结果验证了频率原则是一种广泛存在的现象。事实上，这种低频优先的定性特征对神经网络具体的结构、激活函数、损失函数等因素并不敏感，是神经网络训练过程的一般规律。这一现象为我们理解神经网络的训练和泛化提供了新的视角。下面，我们将对此进行深入探讨。

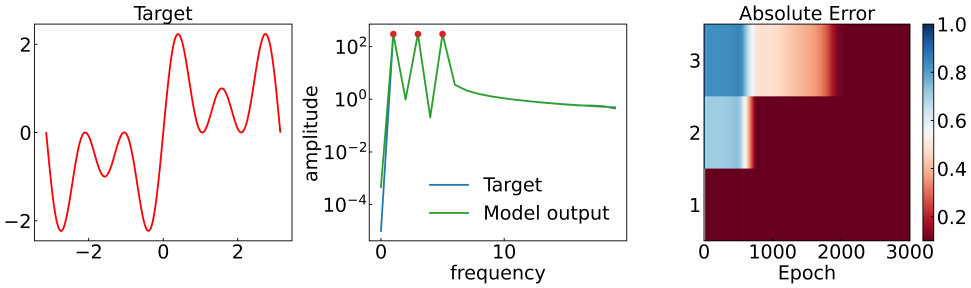


图 4.7: 控制目标函数幅值相同的一维数据实验

4.2 从频率原则理解神经网络

首先需要指出的是, 任何算法不可能在所有数据集上都有好的泛化能力。这就是前文介绍的没有免费午餐定理所揭示的事实。因此, 深入理解模型的学习过程, 并设法使其符合任务需求, 是机器学习研究中的重要课题。频率原则为我们提供了一个新的角度来审视神经网络的学习行为。基于频率原则, 我们可以更好地理解神经网络在不同任务中的表现。在接下来的章节中, 我们将进一步探讨频率原则在神经网络优化、泛化理论以及新模型设计等方面的应用, 展示其在深度学习研究中的重要价值。

4.2.1 实验理解频率原则的必要性

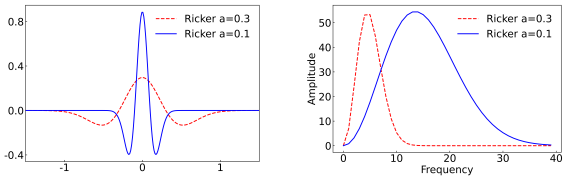
一维实验

为了直观理解频率原则的重要性, 我们考虑用一个全连接神经网络来拟合函数 $\sin(x) + \sin(3x) + \sin(5x)$。在这个例子中, 我们选择带参数 a 的 Ricker 函数 (如图4.8) 作为激活函数:

$$\sigma(x) = \frac{1}{15a}\pi^{1/4}\left(1 - \left(\frac{x}{a}\right)^2\right)\exp\left(-\frac{1}{2}\left(\frac{x}{a}\right)^2\right). \tag{4.3}$$

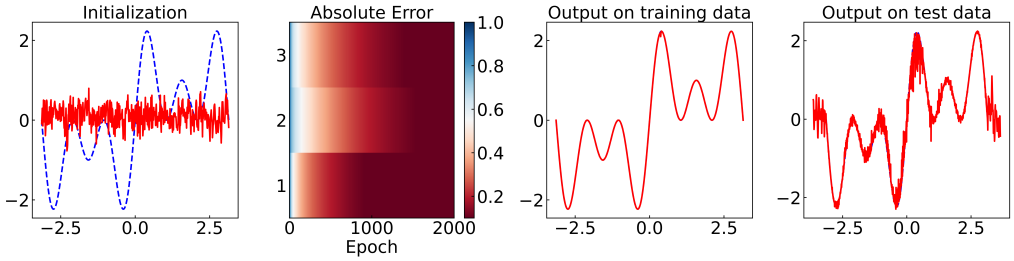
通过调节参数 a 的大小, 我们可以控制神经网络对高频和低频成分的拟合速度。a 越大，网络对低频成分的拟合速度越快。

图4.9展示了不同 a 值下神经网络的拟合效果。在图4.9第一行, 我们选择一个较小的 a 值, 使得高频和低频成分的拟合速度相近, 即频率原则不再显著。此时, 尽管网络能够完美地拟合训练点, 但在测试点上的输出出现了明显的振荡, 与真实值差异较大。这表明网络过度拟合了

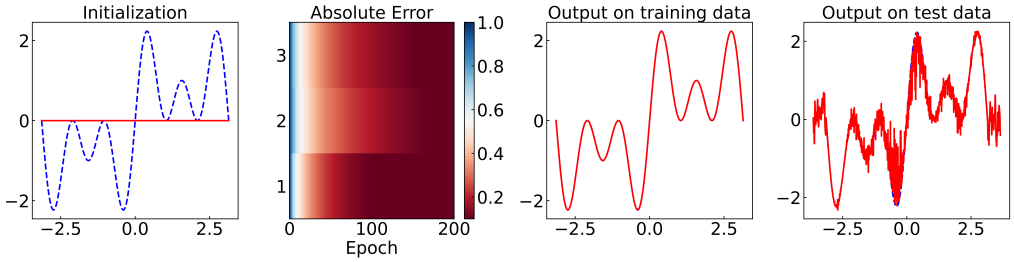


(a) spatial domain (b) frequency domain

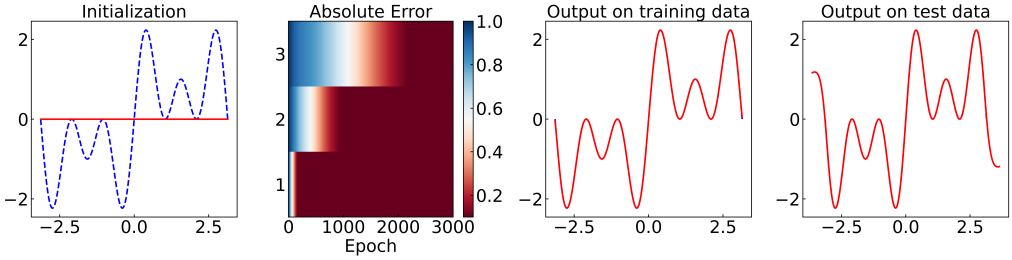
图 4.8: Ricker 激活函数示意图



(a) a=0.1, without ASI trick



(b) a=0.1, with ASI trick



(c) a=0.3, with ASI trick

图 4.9: 不同的 Ricker 激活函数下神经网络拟合情况

训练数据, 导致泛化能力下降。然而, 我们可以观察到, 在初始化阶段, 模型已经有了一定的高频成分, 这有可能对拟合结果中的高频振荡有直接的贡献。因此, 我们目前还不能确定过拟合和频率原则不显著之间的关系。为了排除振荡的初始函数对训练结果的影响, 我们通过一个常用技巧——antisymmetrical initialization (ASI) Zhang et al. (2020) 来控制初始输出函数为 0 函数。具体而言, ASI 通过生成两个独立且初始化参数完全一致的子网络, 并将两个网络的差作为模型的输出。这个技巧可以有效地确保使用不同激活函数的模型初始频率分布的一致性。在图4.9第二行, 我们选择一个较小的 a 值的同时, 通过 ASI 控制模型输出为 0, 容易观察到, 此时模型的拟合过程仍然不遵循频率原则, 且模型的泛化能力依然较差。而在图4.9的第三行, 我们选取一个较大的 a 值, 使得网络对低频成分的拟合速度更快。在这种情况下, 尽管网络在训练点上的拟合并不完美, 但在测试点上的输出与真实值非常接近。这说明网络很好地捕捉到了函数的整体结构, 实现了良好的泛化性能。同理, 对于激活函数 $\sigma(x) = sin(ax)$, 我们可以通过修改 a 的值来改变激活函数的频率分布。如图4.10所示, 对于较大的 a, 频率原则会被削弱甚至消失, 此时模型将会表现出差的泛化性。

从这两个简单的例子中, 我们已经可以认识到频率原则的重要性。当频率原则不成立时, 即网络没有表现出明显的低频偏好时, 拟合得到的模型输出往往包含大量高频振荡成分, 这通常意味着较差的泛化性能。

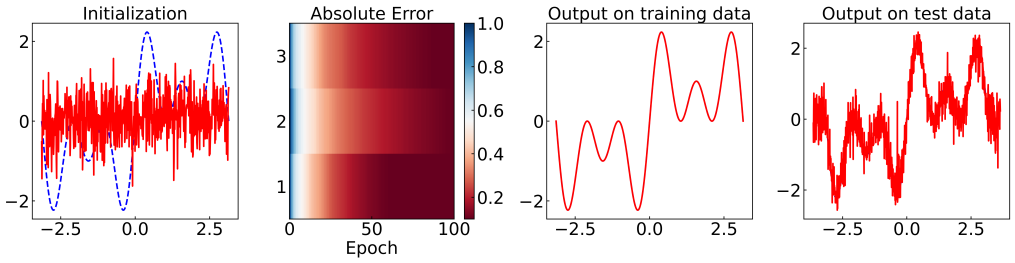
二维实验

对于二维甚至真实世界的数据, 同样也存在类似的现象。在二维实验中, 我们采用大初始化来破坏频率原则。如图4.11所示, 当我们只使用图像的奇数列像素点, 并采用较大的初始化权重进行训练时, 虽然在训练集上能够很好地拟合这些像素点, 但是当我们把所有像素点都输入到训练好的网络中进行测试时, 结果却变得面目全非。将测试输出沿着 x 方向绘制成曲线, 可以发现其中包含了大量的高频振荡。产生这种现象的原因与前面的一维实验类似, 即高频拟合速率增大, 使得模型在拟合低频的同时, 学习到很多高频噪声。

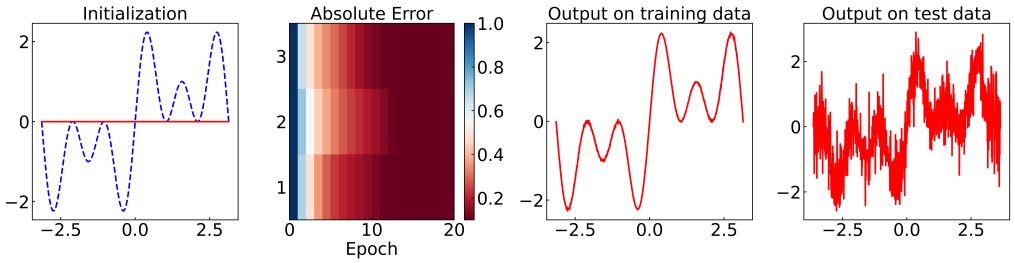
4.2.2 Early-stopping 的频率角度理解

在实际应用神经网络时, 人们经常发现将训练误差降到最低并不一定是最优策略。在训练过程中, 我们通常观察到训练误差会持续下降, 但测试误差却呈现先下降后上升的趋势。图4.12展示了一个典型的例子, 其中我们用神经网络拟合带有噪声的蓝色虚线（图 b）。图（a）显示, 随着训练的进行, 训练误差 (蓝色曲线) 不断下降, 但测试误差 (红色曲线) 先下降后上升。显然, 在这个例子中, 我们需要在测试误差最低点 (黑色虚线处) 提前停止训练, 以避免过拟合。

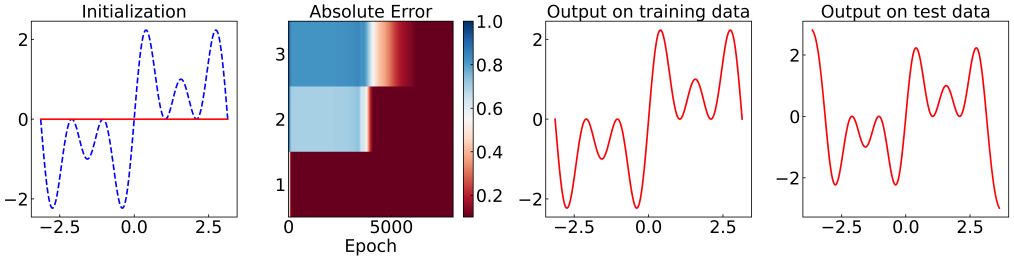
为了理解 early-stopping 在这个例子中的有效性, 我们首先观察在黑色虚线处神经网络的



(a) a=10, without ASI trick



(b) a=10, with ASI trick

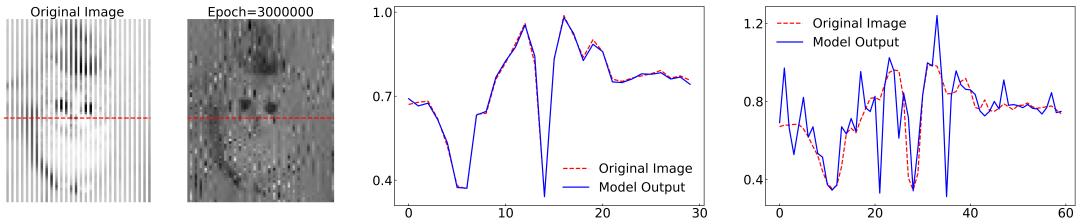


(c) a=1, with ASI trick

图 4.10: 激活函数 $\sigma(x) = \sin(ax)$ 下神经网络拟合情况

输出情况。从图（b）的蓝色曲线可以看出，此时神经网络的输出是一条平滑的正弦曲线。事实上，我们的数据正是从 $\sin(x)$ 函数加上噪声采样得到的。而在训练末态，如图（c）所示，神经网络拟合到高频噪声信息，模型产生过拟合。接下来，我们在频域中检验此时神经网络的拟合效果。

图4.12(d) 展示了没有噪声的真实数据在频域中的幅度谱（蓝色虚线）。现在我们观察神经网络在训练集和测试集上的频域表现。选取图（a）中黑色虚线所示的时刻（图（d）中红色实线），我们发现此时神经网络输出的低频部分与真实数据吻合得很好，而高频部分的幅度仍然较小。这得益于神经网络的频率原则，即网络倾向于先学习低频成分。在这个时刻，网络尚未



(a) output (b) output on training data (c) output on all data

图 4.11: 初始化权重较大的二维图片拟合实验

拟合被噪声严重影响的高频部分, 因此没有出现明显的过拟合现象。而在训练末端（图（d）中绿色点线），则对高频成分进行拟合，从而发生过拟合。

在许多真实数据中, 噪声对高频成分的影响往往更大。对于这类数据, 频率原则使得神经网络优先学习信噪比高的低频信号, 这恰恰是数据的主要特征。采用 early-stopping 技巧可以有效防止神经网络过度拟合噪声主导的高频成分, 从而避免严重的过拟合问题。

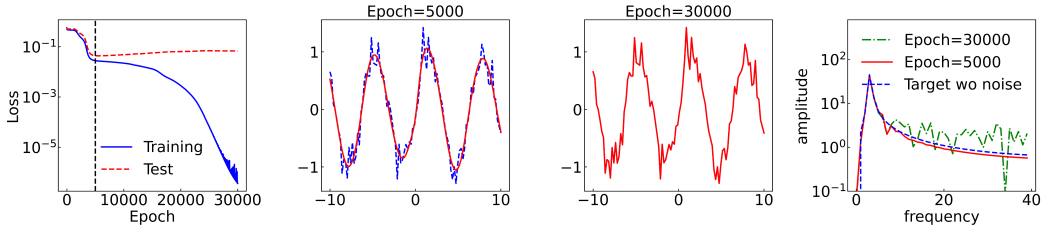


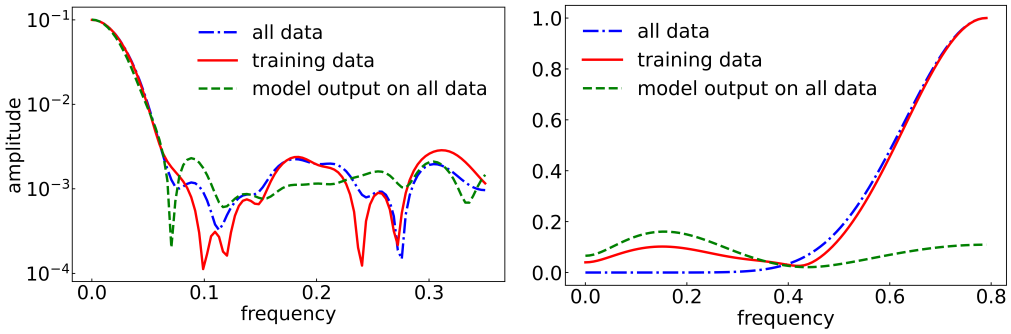
图 4.12: 神经网络拟合带噪声的数据实验

4.2.3 神经网络的优势与局限

之前在探讨泛化问题时, 我们提到过参数化的神经网络不容易发生过拟合这一著名的泛化谜团。事实上, 根据没有免费午餐定理, 这个泛化谜团是针对某类特定的数据, 如图像分类识别等任务。从频率原则可知, 神经网络具有低频偏好, 那么这些可以被很好泛化的数据是否本身也有低频主导的特性呢? 我们是否可以找到一些高频主导的数据, 证明神经网络在这类数据中无法很好地泛化? 在这一小节中, 我们将通过对几类数据的频域分析来验证这些猜想。

图4.13展示了 CIFAR10 主成分方向上进行傅里叶变换的结果（具体做法请见下一节）。红色线由训练集计算得到, 蓝色虚线是测试集加上训练集的结果, 绿色虚线是神经网络输出在训

练点和测试点上做傅里叶变换的结果, 这里选择的是真实数据集中的主成分方向。可以发现, 在低频占优的数据集上, 神经网络拟合结果对低频信息拟合很好, 高频成分的幅值很小, 所以即便高频没有拟合好也不会造成特别大的误差。



(a) CIFAR10　　(b) 奇偶函数

图 4.13: 神经网络拟合真实数据

现在考虑一个高频占优的数据集: 定义一个 d 维空间中的函数, 对于每个维度只取 1 或 -1, 即 $\boldsymbol{x} = (x_1, \cdots, x_d) \in \{-1, 1\}^d$, 函数为 $f(\boldsymbol{x}) = \prod_{j=1}^{d} x_j$。当所有维度中取了偶数个 -1 时,$f(\boldsymbol{x})$ 的值为 1; 如果取了奇数个 -1, 函数值为 -1。这个函数被称为奇偶函数, 经常被用来做智商测试。一旦发现输出只依赖于 -1 的数目, 就很容易预测其它点的值。以 $d = 10$ 为例, 这个函数只定义在 $2^{10} = 1024$ 个点上。如果我们只给其中 200 个数据点作为训练集, 并将其余数据点作为测试集。神经网络可以在训练集上准确地预测每一个点的输出, 或者说能够记住所有训练点对应的输出, 但在测试点上, 它的准确率却始终在 50% 左右。可见在这个例子中, 人工智能表现得是多么不智能! 同样地, 我们用频率空间来分析一下这个例子。

对这个函数做离散傅里叶变换:

$$\frac{1}{2^d} \sum_{x \in \Omega} \prod_{j=1}^{d} x_j \mathrm{e}^{-\mathrm{i}2\pi \boldsymbol{k} \cdot \boldsymbol{x}} = (-\mathrm{i})^d \prod_{j=1}^{d} \sin 2\pi k_j. \tag{4.4}$$

我们将这个变换的曲线展示在图4.13(b) 的蓝线上。当频率 k 变大时，它的幅度也慢慢增长，说明这是一组高频占优的数据。当我们采样不完整时，训练集做离散傅里叶变换的结果由红线表示。显然，在幅度较小的低频部分，它与真实频谱有明显区别。事实上，由于采样不足，会导致训练数据产生一些假的低频（频率混叠效应）。

现在用神经网络对这个函数进行拟合，可以发现神经网络会产生很多假的低频。也就是说，对于未知标签的数据，神经网络倾向于用低频来拟合，导致神经网络学习到的高频比真实数据的高频要少很多。因此，神经网络的频谱（绿色虚线）与真实数据的频谱完全不一样，这也解释了为什么神经网络完全无法在奇偶函数上做出好的预测。

4.3 频率原则的高维实验*

在前面的章节中, 我们通过一维函数的例子介绍了频率原则, 揭示了神经网络倾向于以从低频到高频的顺序拟合训练数据的特点。然而, 在实际应用中, 我们往往面临着更加高维和复杂的问题。以图像识别任务为例, 输入数据通常是高维的图像矩阵, 而输出则是对应的类别标签。在这种情况下, 频率的概念和傅里叶变换的应用都变得更加抽象和难以想象。以 CIFAR10 图像分类任务为例, 每张输入图片的大小为 $32 \times 32 \times 3$, 其中 "3" 对应着 RGB 三个颜色通道。将一张图片输入到神经网络中, 我们需要处理 3072 个像素值, 而网络的输出则是 10 个类别中每个类别的概率。也就是说, 在 CIFAR10 的分类问题中, 神经网络需要学习一个从 3072 维输入空间到 10 维输出空间的高维映射函数。面对如此高维的问题, 我们很难直观地想象其在频域上的表示。然而, 通过引入适当的频率度量和分析工具, 我们仍然可以研究神经网络在学习过程中对不同频率成分的拟合情况。

在接下来的部分, 我们首先区分图片分类任务中两种常见的频率概念——图像频率和响应频率。对于图像分类任务中，我们更关心的是响应频率。然后为了克服高维傅里叶变换的复杂性，我们介绍高维傅里叶变换的两种替代方式。

4.3.1 图像中的频率

在图像拟合的问题中，我们考虑的是图像频率, 频率的概念与像素值变化的剧烈程度密切相关。当我们说一张图片包含高频成分时, 通常意味着图像中存在着像素值变化很快的区域, 如物体的边缘、纹理细节等。相反, 如果一个区域的像素值变化平缓, 我们则认为其主要包含低频成分。

以图4.14所示的照片为例。在这张图像中, 毛发的边缘区域像素值变化非常剧烈, 呈现出明显的高频特征。这些高频成分携带了图像中的细节信息, 使我们能够清晰地感知毛发的纹理和形状。而在背景区域, 像素值的变化相对平缓, 因此这部分主要由低频成分主导。在极端情况下, 如果一张图片只包含一种纯色, 那么它就相当于一个只含有零频的常值函数, 其像素值在整个图像范围内保持不变。

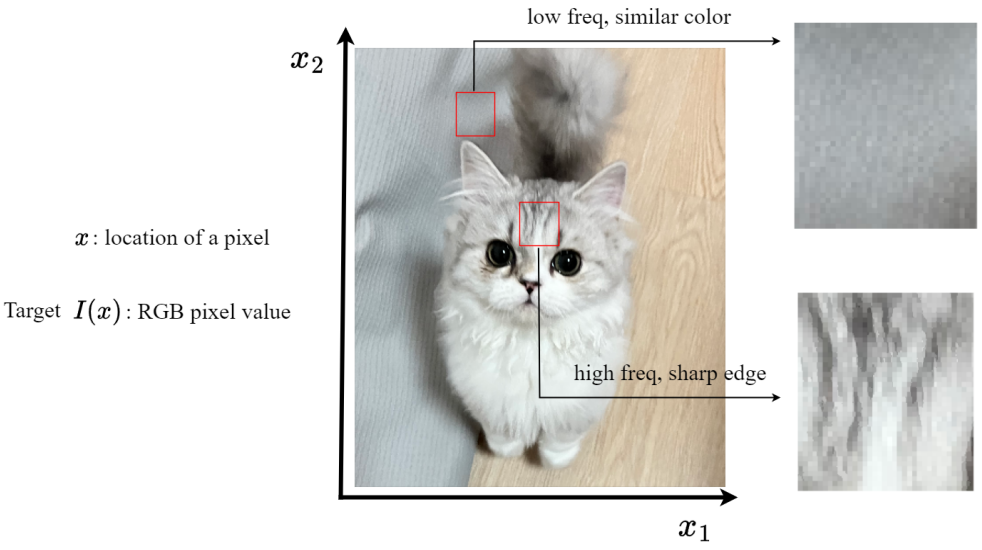


图 4.14: 图像频率的示例

4.3.2 分类问题中的响应频率

在高维函数中, 频率的概念可能变得相对抽象和难以想象。在这种情况下, 我们更关注的是从输入到输出的响应频率, 而不是输入信号自身的频率特性。响应频率描述了函数输出对输入变化的敏感程度。

对于高维函数中的某一点, 沿着不同的方向, 其主导的频率成分可能有很大差异。一个典型的例子是图像分类问题中的对抗样本 (adversarial examples)。如图4.15所示, 我们有一张猫的图片，它作为训练集中的一张图片，对应着高维输入空间（所有训练集）中的一个点。如果我们沿着某个特定方向对这个点进行微小的扰动, 就可以得到一张与原始图片在视觉上几乎一模一样的新图片。然而, 这个细微的扰动却可能导致神经网络的输出发生剧烈变化, 使得原本预测为“猫” 的图片被错误地分类为“飞机”。这个现象说明, 在该方向上, 分类函数包含着高强度的高频成分, 因为微小的输入变化引起了巨大的输出变化。

这个实验结果引出一个有趣的问题: 为什么在满足频率原则的前提下, 模型学习到的函数依旧会包含如此高频的成分, 以至于一个微小的扰动便会使模型输出产生极大的差异? 一个可能的解释是, 虽然高频的拟合速度远远小于低频, 但模型学习到的函数中仍包含少量高频成分, 而这种高频成分会在对抗样本实验中被特定选择。

需要特别指出的是, 在考虑图像分类问题中的频率时, 很多人往往只关注图像频率, 而忽略分类函数的响应频率。这可能导致对频率原则本质的误解。当然, 这并不意味着图像内部的频

率特性对分类问题无关紧要。事实上, 研究图像中低频和高频成分对分类准确度和稳定性的影响, 也是一个非常有趣和重要的课题。图像的频率组成可能影响神经网络的特征提取和学习过程, 进而影响分类的正确率。

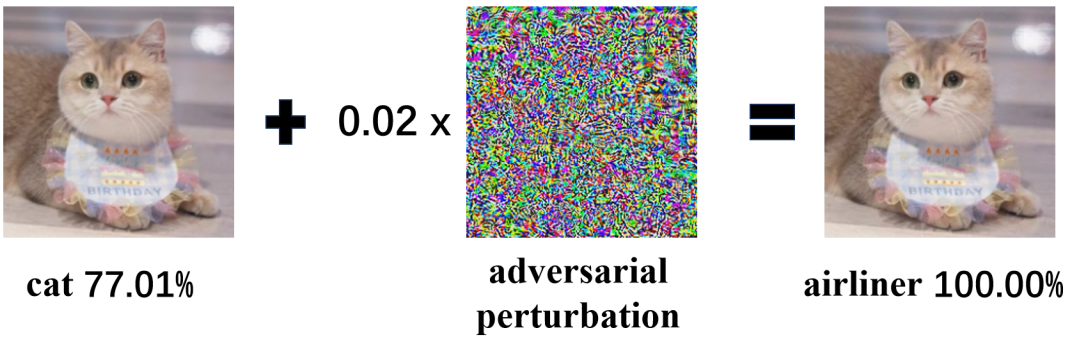


图 4.15: 对抗样本示例

4.3.3 高维傅里叶变换的困难

在高维数据集上进行傅里叶变换时, 我们可以将其看作连续傅里叶变换的一种离散蒙特卡洛近似形式。对于一个训练数据集 $\{(\boldsymbol{x}_j, \boldsymbol{y}_j)\}_{i=1}^n$, 其中$\boldsymbol{x}_j \in \mathbb{R}^d, \boldsymbol{y}_j \in \mathbb{R}^{d_o}$, 我们可以使用非均匀离散傅里叶变换 (Nonuniform Discrete Fourier Transform, NUDFT) 来估计数据在响应频率 $\boldsymbol{k}$ 处的幅值:

$$\hat{y}_{\mathbf{k}} = \frac{1}{n} \sum_{j=1}^{n} \boldsymbol{y}_j \mathrm{e}^{-\mathrm{i}2\pi \mathbf{k} \cdot \mathbf{x}_j}, \tag{4.5}$$

其中 $\mathbf{k}$ 的维数为 d, 与图像输入的维数相同。例如, 对于 MNIST 数据集, 输入 $\boldsymbol{x}$ 是一个 784 维的灰度值向量, 那么相应的频率 $\mathbf{k}$ 也是 784 维的向量。值得注意的是, 公式中的求和是对所有样本进行的, 因此这里计算的频率是考虑了所有样本和所有输出的整体频率特性。

然而, 直接对高维数据进行傅里叶变换往往面临着维数灾难 (curse of dimensionality) 的问题。以 MNIST 数据集为例, 输入 $\boldsymbol{x}$ 是 784 维的向量。如果我们在频率 $\boldsymbol{k}$ 空间的每个维度上只考虑两个频率取值（比如 $\pm k_0$）, 那么就会产生 2^{784}（$\sim 10^{236}$）个频率点。对于当前的计算能力而言, 计算所有这些频率点的幅值是一个不可能完成的任务, 更不用说比较它们在训练过程中的收敛速度！因此, 我们需要在计算效率和信息完整性之间进行权衡, 寻找一种折中的方法来研究高维函数在频域空间的特征。下面, 我们介绍两种可行的检验高维函数问题中神经网络频率原则的方法。

4.3.4 投影法

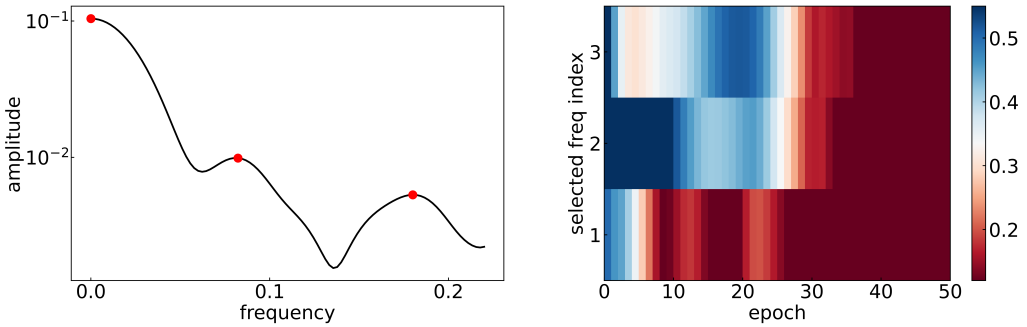
一种方法称为**投影法 (Projection Method)**, 其基本思想是只计算函数在高维频率空间中某一特定方向上的频率分量。首先, 我们选定一个投影方向 $\mathbf{p}_1$, 并将频率向量 $\mathbf{k}$ 表示为该方向上的标量乘积形式, 即 $\mathbf{k} = k\mathbf{p}_1$。将其代入非均匀离散傅里叶变换 (NUDFT) 的定义式中, 可以得到:

$$\hat{y}_{\mathbf{k}} = \frac{1}{n}\sum_{j=1}^{n} y_j \mathrm{e}^{-\mathrm{i}2\pi k(\mathbf{p}_1\cdot\mathbf{x}_j)}. \tag{4.6}$$

理论上, 投影方向 $\mathbf{p}_1$ 可以任意选取。不过选择信息较多，特别是包含不同频率成分的投影方向有助于我们检验频率原则。因此，在下面的实验中, 我们选择了数据的主成分方向作为投影方向进行研究。将数据在该方向上的投影记为 v_j, 即 $v_j = \mathbf{p}_1 \cdot \mathbf{x}_j \in \boldsymbol{R}$, 那么 NUDFT 可以进一步化简为:

$$\hat{y}_{\mathbf{k}} = \frac{1}{n}\sum_{j=1}^{n} y_j \mathrm{e}^{-\mathrm{i}2\pi k v_j}. \tag{4.7}$$

这实际上将高维傅里叶变换转化为一维情况, 大大降低了计算复杂度。



(a) fourier domain

(b) relative error

图 4.16: CIFAR10 在主成分方向的傅里叶变换及其主频率下的收敛速度

图4.16展示了在 CIFAR10 数据集上应用投影法的实验结果。图 (a) 给出了 CIFAR10 在主成分方向上的傅里叶变换结果。可以发现, 对于真实数据, 低频分量通常占主导地位, 即低频处的幅值往往更大。我们选取其中几个幅值较大的峰值点对应的频率进行考察, 结果如图 (b) 所示。颜色越偏向深红, 表示在某个频率上幅值的相对误差 $\Delta_F(k) = \left|\hat{h}_k - \hat{y}_k\right| / |\hat{y}_k|$ 越小, 即该频率成分拟合越好。从单个投影方向的角度来看，我们观察到了低频分量比高频分量更早收敛的现象。这为频率原则在高维问题中的适用性提供了初步的支持。当然, 由于投影法只考虑

了单一方向的频率成分, 还不足帮助我们全面了解神经网络的频率偏好。我们下面将进一步探索一种能够综合不同方向频率信息的分析方法。

4.3.5 滤波法

尽管从投影法可以从单个投影方向的角度观察到了低频分量比高频分量更早收敛的现象, 但有时我们并不知道感兴趣的信号处于哪个特定方向。为此, 我们引入另一种方法——滤波法 (Filtering Method), 其基本思想是将频率空间分为低频和高频两个部分 (如图4.17(a) 所示), 然后分别检验低频部分和高频部分的收敛情况。理论上来说，我们可以通过傅里叶变换获取频率信息，然后舍去高频（蓝色部分），仅对其低频（红色部分）做逆变换, 即可获得低频成分。具体来说，我们可以把要检验的函数的傅里叶变换乘以一个示性函数 (图4.17(b))，这个示性函数在频率强度小于等于 k_0 的范围内为 1，其余为 0，

$$\mathbf{1}_{|\boldsymbol{k}|\leq k_0} = \begin{cases} 1, & |\boldsymbol{k}| \leq k_0 \\ 0, & |\boldsymbol{k}| > k_0 \end{cases} . \tag{4.8}$$

随后再通过傅里叶逆变换得到低频成分。然而，由于高维傅里叶变换有维数灾难，这种方法实际并不可行。为了避免做高维傅里叶变换, 我们利用卷积定理来解决这个问题。根据卷积定理, 两个频域函数 $\hat{g}_1$ 和 $\hat{g}_2$ 的乘积做傅里叶逆变换, 等价于它们各自的逆变换在时域空间做卷积, 即:

$$F^{-1}\left[\hat{g}_1 \cdot \hat{g}_2\right](x) = F^{-1}\left(\hat{g}_1\right) * F^{-1}\left(\hat{g}_2\right). \tag{4.9}$$

尽管利用了卷积定理, 我们仍然需要计算高维示性函数的傅里叶逆变换。为了克服这个困难, 我们采用高斯函数 (如图4.17(c) 所示) 来近似示性函数。这是因为:1) 高斯函数具有指数衰减的性质, 可以作为理想示性函数的良好近似;2) 高斯函数的傅里叶逆变换仍然是高斯函数。以一维高斯函数 $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$ 为例, 对其做傅里叶变换可得:

$$F(\omega) = \int_{-\infty}^{+\infty} e^{-ax^2} e^{-j\omega x} dx = \int_{-\infty}^{+\infty} e^{-ax^2 - j\omega x} dx = e^{-\frac{\omega^2}{4a}} \int_{-\infty}^{+\infty} e^{-\left(\sqrt{a}x + \frac{j\omega}{2\sqrt{a}}\right)^2} dx = \sqrt{\frac{\pi}{a}} e^{-\frac{\omega^2}{4a}}. \tag{4.10}$$

这表明高斯函数在傅里叶变换前后保持了 $f(x) = e^{-ax^2}$ 的形式, 即高斯函数的傅里叶 (逆) 变换仍是高斯函数。对于时域上标准差为 δ 的高斯函数, 其在频域上的标准差大约为 $1/\delta$, 这个标准差近似对应于示性函数的宽度。因此, 频域上的截断操作转化为时域上函数与高斯函数的卷积，即添加高斯窗的方法，从而近似地分离出函数的高频成分和低频成分。低频成分可近似表示为:

$$\boldsymbol{y}_i^{\text{low},\delta} = \frac{1}{C_i} \sum_{j=0}^{n-1} \boldsymbol{y}_j G^{\delta}(\boldsymbol{x}_i - \boldsymbol{x}_j), \tag{4.11}$$

其中 $C_i = \sum_{j=0}^{n-1} G^\delta(\boldsymbol{x}_i - \boldsymbol{x}_j)$ 是归一化因子，

$$G^\delta(\boldsymbol{x}_i - \boldsymbol{x}_j) = \exp\left(-|\boldsymbol{x}_i - \boldsymbol{x}_j|^2/(2\delta)\right). \tag{4.12}$$

高频部分可以近似为 $\boldsymbol{y}_i^{\text{high},\delta} \triangleq \boldsymbol{y}_i - \boldsymbol{y}_i^{\text{low},\delta}$。对于神经网络的每个输出 $\boldsymbol{h}_i$, 我们也可以进行类似的处理, 得到 $\boldsymbol{h}_i^{\text{low},\delta}$ 和 $\boldsymbol{h}_i^{\text{high},\delta}$。然后, 我们可以计算低频和高频的相对误差:

$$e_{\text{low}} = \left(\frac{\sum_i |\boldsymbol{y}_i^{\text{low},\delta} - \boldsymbol{h}_i^{\text{low},\delta}|^2}{\sum_i |\boldsymbol{y}_i^{\text{low},\delta}|^2}\right)^{\frac{1}{2}}, \tag{4.13}$$

$$e_{\text{high}} = \left(\frac{\sum_i |\boldsymbol{y}_i^{\text{high},\delta} - \boldsymbol{h}_i^{\text{high},\delta}|^2}{\sum_i |\boldsymbol{y}_i^{\text{high},\delta}|^2}\right)^{\frac{1}{2}}, \tag{4.14}$$

这里 h 和 y 分别代表神经网络输出和真实标签。

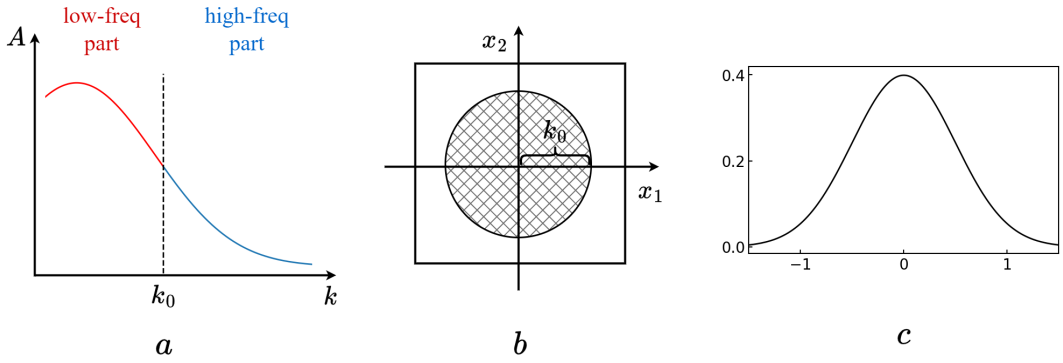
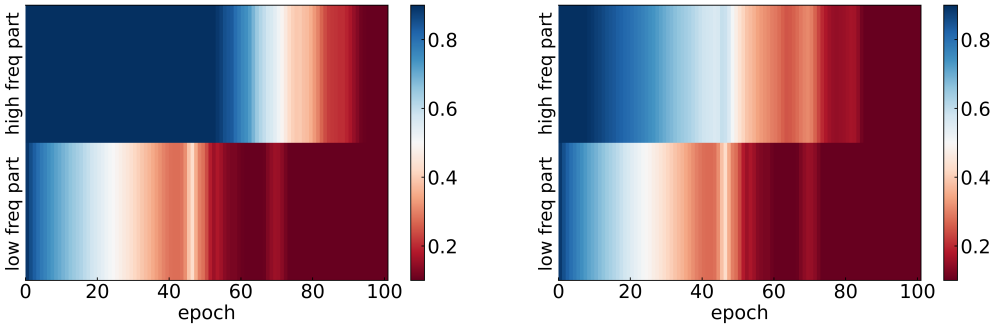


图 4.17: 滤波法相关概念示意图

我们在 CIFAR10 数据集上使用卷积神经网络 (CNN)，并采用交叉熵损失函数进行训练。实验结果如图4.18所示。其中 δ 表征高斯函数的宽度，也就是研究的低频到底有多低，高频到底有多高。δ 越大表示高斯函数在频率空间的带宽越窄，低频成分的范围越小。热图表示神经网络拟的高频和低频部分拟合结果随训练进行的相对误差，横坐标表示训练 epoch。红色代表神经网络拟合结果的相对误差很小，蓝色代表相对误差较大。通过对图4.18中低频部分与高频部分的收敛速度进行比较, 我们发现模型呈现出了低频成分先于高频成分收敛的现象。同时基于子图 (a, b) 可以看出，当高斯函数的标准差变化时, 这一结论仍然成立。

对于大量非梯度下降的优化方法，例如共轭梯度下降、牛顿法、粒子群算法等，频率原则仍然可以被观察到Ma et al. (2021)。



(a) δ=1　　(b) δ=5

图 4.18: 不同 δ 下滤波法在高/低频部分的收敛速度

4.4 习题

1. 频率原则是神经网络学习过程中的一般规律。请回答以下问题：

   (a) 什么是频率原则？请给出其定义。

   (b) 频率原则对于理解神经网络的训练和泛化有什么启示？

   (c) 设计一个实验来验证频率原则。描述实验的设置、使用的数据集、网络结构以及如何分析实验结果。

2. 在本章的实验中，我们观察到破坏频率原则会导致神经网络的泛化性能下降。

   (a) 请举例说明如何通过改变激活函数或初始化方式来破坏频率原则。

   (b) 在你的例子中，神经网络的输出有什么特点？这与频率原则的预期相符吗？

   (c) 破坏频率原则的神经网络在训练集上能取得很低的损失吗？它在测试集上的表现如何？请解释原因。

3. Early-stopping 是一种常用的正则化技巧，可以防止神经网络过拟合噪声。

   (a) 在本章给出的例子中，Early-stopping 是如何起作用的？请从频率的角度进行解释。

   (b) 噪声通常会对函数的高频部分产生更大影响。这个事实与频率原则有什么关系？它们共同说明了 Early-stopping 的有效性吗？

4. 没有免费午餐定理告诉我们，任何算法都不可能在所有问题上都取得良好的泛化性能。

(a) 请举例说明一类高频主导的函数。对于这类函数, 标准的神经网络是否能够很好地拟合和泛化?

(b) 结合频率原则对上述问题进行解释。这体现了神经网络的优势和局限性吗?

(c) 对于高频主导的数据, 我们是否可以通过一些技巧使神经网络在其上也能取得好的泛化? 请给出你的分析和想法。

5. 对于二维图像，如何理解其高频成分和低频成分？它们各自对应了图像的哪些特征?

6. 在图像分类任务中，我们更关注的是模型从输入到输出的响应频率。

   (a) 图像频率与响应频率有什么区别？哪个与分类问题的泛化性能更加相关?

   (b) 对抗样本的存在说明了什么问题？它与响应频率的概念有什么联系?

7. 对高维数据进行傅里叶变换通常面临维数灾难的问题。请回答:

   (a) 维数灾难具体指什么？它的根本原因是什么?

   (b) 投影法是如何缓解这一问题的？请简要说明其基本思想。

   (c) 滤波法在解决维数灾难问题上有什么独特的优势？它与投影法相比有什么不同?

Chapter 5

基于频率原则设计高效神经网络

在许多科学和工程问题中，频域上的多尺度现象是十分常见的。例如，在图像处理中，一张图像往往既包含物体内部色彩变化平滑的低频区域，也包含物体边缘、细节、纹理这一类变化突兀的高频成分；在微分方程求解问题中，许多方程的解是由不同频率尺度的周期函数复合得到的。而根据前面章节提到的频率原则，一般的神经网络结构在训练过程中对低频信息的收敛速度较快，而对高频信息的收敛速度相对较慢，这使得其在真实问题的应用中往往面临着高频训练困难的问题。因此，我们有必要设计合适的网络架构来提升神经网络的高频收敛速度，以应对真实应用场景中的多尺度问题。

近年来，频率视角在神经网络优化中得到了越来越多的关注。研究人员从模型架构、求解策略、特征表示等多个层面入手，提出了一系列基于频率分析的改进方法，显著提升了神经网络处理多尺度问题的能力。例如，在计算机视觉领域，通过引入多尺度特征提取模块，网络能够更好地捕捉不同尺度下的物体信息，在图像分类、目标检测等任务上取得了显著的性能提升。在科学计算领域，频率分析视角下的神经网络模型能够更高效地求解多尺度偏微分方程，极大地拓展了其应用范围。可以预见，频率视角与神经网络的结合仍大有可为，必将在更广泛的领域展现出令人瞩目的应用前景。

本章将聚焦于一些重要的基于频率原则设计的高效神经网络，如多尺度神经网络、神经辐射场、Fourier 特征网络等。我们希望通过介绍这些模型的思想原理、方法和测试效果，进一步说明频率原则的重要性，启发读者有更多相关的思考。

科普篇

频率原则会给神经网络训练带来哪些影响？

神经网络在训练过程中对低频信息的收敛速度通常较快，对高频信息（如图像中的细节和边缘）的收敛速度较慢，导致模型在捕捉复杂特征和细节时表现较差。

在实际问题中存在哪些频域多尺度现象？

在实际的科学和工程问题中，频域上的多尺度现象非常普遍。在图像处理中，一张图片通常包含不同频率的成分：低频成分包括图像中的大面积平滑区域，如天空或墙壁，表现为缓慢变化的色彩；高频成分包括图像中的细节和边缘，如物体的轮廓和纹理，表现为快速变化的像素值。在气候模型中，多尺度现象涵盖了从几天到几百年的不同时间尺度，其中季节变化和年际变化属于低频变化，而天气变化和短期气象事件则属于高频变化。在声音信号处理中，低频成分通常对应声音的基音和大部分能量所在的区域，如音乐中的低音；高频成分则对应声音的细节和质感，如乐器的谐波和人声的清晰度。生物医学信号如心电图（ECG）或脑电图（EEG）也显示出多尺度现象：低频成分反映心脏或大脑的整体活动趋势；高频成分反映快速变化的电生理事件，如心跳或神经放电。通过这些例子可以看出，理解和处理频域上的多尺度现象对分析和解决实际问题至关重要。

在已有的工作中，研究人员通过哪些方法缓解神经网络的“高频困难”？

研究人员从模型架构、求解策略、特征表示等多个层面入手，提出了一系列基于频率分析的改进方法，显著提升了神经网络处理多尺度问题的能力。例如，在计算机视觉领域，通过引入多尺度特征提取模块，网络能够更好地捕捉不同尺度下的物体信息，在图像分类、目标检测等任务上取得了显著的性能提升。在科学计算领域，频率分析视角下的神经网络模型能够更高效地求解多尺度偏微分方程，极大地拓展了其应用范围。

5.1 多尺度神经网络结构

基于神经网络在低频收敛快而高频收敛慢的特点，一种很自然的想法是将高频数据转换为低频数据，以加快训练速度。基于这个想法，PhaseDNN Cai et al. (2020) 在各个维度上频率打网格，然后把每个网格上的频率变换到低频。但通过各个维度的操作会面临维数灾难，使其对高维问题难以处理。那么，如何在高维问题实现这种转换呢？我们可以观察图5.1中的两张子图，其中左图的震荡明显比右图剧烈。要将左图转换为右图，只需对坐标进行简单的拉伸即

可。这个简单想法的合理性也能由傅里叶变换的放缩性质得到验证：

$$\mathcal{F}(f(kx))(\zeta) = \frac{1}{|k|}\widehat{f}(\frac{\zeta}{k}). \tag{5.1}$$

在一般问题中，我们通常无法预先知道原始数据的频率信息，也无法确定究竟要做多少尺度的拉伸。为了解决这个问题，我们可以提供多种尺度通道，让神经网络自主选择适合的频率变换。这样，即使我们不知道数据的具体频率，神经网络也能自适应地调整，以适应不同频率的信息。基于这样的想法，Liu et al. (2020) 设计了多尺度神经网络，通过对网络输入作伸缩变换，缓解网络的高频学习困难。

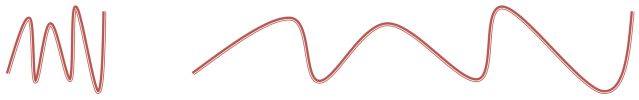
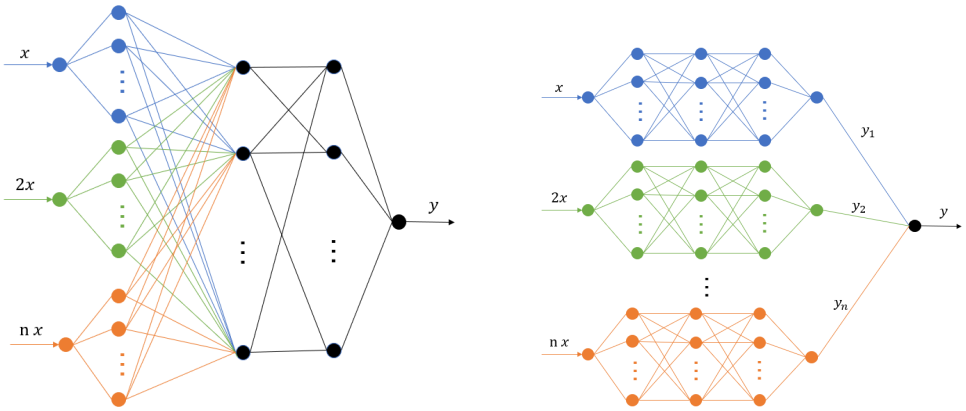


图 5.1: 通过拉伸将高频变为低频。

5.1.1 结构介绍



(a) MscaleDNN-1　　(b) MscaleDNN-2

图 5.2: 两类多尺度神经网络结构。摘自Liu et al. (2020)。

为了实现多尺度神经网络，我们将第一个隐藏层的神经元分解为 A 个部分。第 i 部分的神经元的输入是 $i\boldsymbol{x}$，也就是说它对应的输出是 $\sigma(i\boldsymbol{w}\cdot\boldsymbol{x}+b)$, 这里 $\boldsymbol{w}$、$\boldsymbol{x}$、$\boldsymbol{b}$ 分别对应着权重，

输入和偏差项。完整的 MscaleDNNs 是如下定义的，

$$h(\boldsymbol{x}) = \boldsymbol{W}_L\sigma \circ (\boldsymbol{W}_{L-1}\sigma \circ (\cdots \sigma \circ (\boldsymbol{W}_1\sigma \circ (\boldsymbol{K} \odot \boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}_0) + \boldsymbol{b}_1)\cdots) + \boldsymbol{b}_{L-1}), \tag{5.2}$$

这里 $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{W}_l \in \mathbb{R}^{n_{l+1}\times n_l}$, n_l 是第 l 个隐藏层的神经元数目, $n_0 = d$, $b_l \in \mathbb{R}^{n_{l+1}}$, σ 表示网络的激活函数，$\circ$ 代表逐元素操作, $\odot$ 指哈达玛乘积，

$$\boldsymbol{K} = (\underbrace{1,1,\cdots,1}_{\text{1st part}},\underbrace{1,1,\cdots,1}_{\text{2nd part}},2,\cdots,i-1,\underbrace{i,i,\cdots,i}_{\text{(i+1)th part}},i+1,\cdots,\underbrace{A-1,A-1,\cdots,A-1}_{\text{Ath part}})^T. \tag{5.3}$$

图5.2展示了两种常见的多尺度网络结构。结构 (a) 将普通全连接网络的第一个隐藏层的每一个神经元加一个系数。第二种结构由不同尺度的小网络组成，由于子网络之间没有相互连接，因此，参数数目远比第一种网络要少。

注意，如果只用一个非常大的尺度把函数拉到非常低的频率，看起来好像变低频了，但实际上神经网络仍然很难学好，一方面是因为神经网络可能有学习最快的特定频率，太低频反而学不快。另一方面，这种统一的拉伸，会使问题仍然是有多尺度的，不同尺度的学习会互相竞争，使学习速度变慢。我们提供给网络多种尺度，在梯度下降等算法中，期待网络会选择最快学习的一条路径来更新输出，达到最优的速度。下一子节的例子在实验上，验证了不同拉伸的部分确实学习到不同频率的部分。

接下来，我们看一下多孔二维的例子来验证多尺度网络的可行性，来源于Liu et al. (2020)。考虑带有如下源项的泊松方程：

$$f(\boldsymbol{x}) = 2\mu^2 \sin\mu x_1 \ \sin\mu x_2, \mu = 7\pi. \tag{5.4}$$

精确解为

$$u(\boldsymbol{x}) = \sin\mu x_1 \sin\mu x_2. \tag{5.5}$$

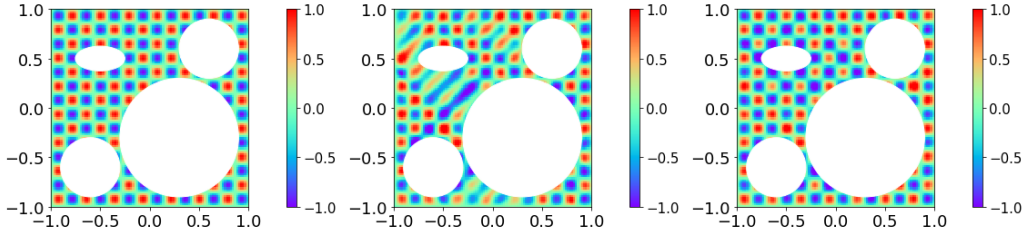
我们同样利用精确解来设定边界条件。采样和神经网络结构和前面的一样。通过这样的结构去参数化方程，损失共有两部分，第一部分为方程残差项 L_{eq}，采用方程的能量作为该部分损失；第二部分 L_{bd} 为方程的边界损失。损失函数形式如下：

$$L_{sum} = L_{eq} + L_{bd}, \tag{5.6}$$

具体可以参考 A13 部分关于神经网络解偏微分方程的内容。如图5.3所示，多尺度神经网络可以准确地抓住精确解中的振荡。

5.1.2 基于子空间分解的神经网络

Li et al. (2023) 提出了基于子空间分解的神经网络 (Subspace Decomposition based DNN , SD^{2}DNN) 架构，如图 5.4所示。该架构由两部分组成：第一部分是一个用于捕获多尺度解中



(a) 精确解　(b) 正常神经网络　(c) MscaleDNN

图 5.3: 多孔二维的例子。图片来源于Liu et al. (2020)。

平滑分量的低频 MscaleDNN 子模块或一个普通的全连接神经网络；第二部分为一个或多个用于捕获多尺度解中振荡分量的高频 MscaleDNN 子模块。为了分离解的平滑和振荡分量，损失函数中增加了一个正交惩罚项，强制不同尺度子空间之间保持正交性。这个工作使用了如下形式的激活函数 $\sigma(z) = s[\cos(z), \sin(z)]$，其中，$s \in (0, 1]$ 是一个用于控制输出范围的松弛参数，文中 s 设置为 0.5。

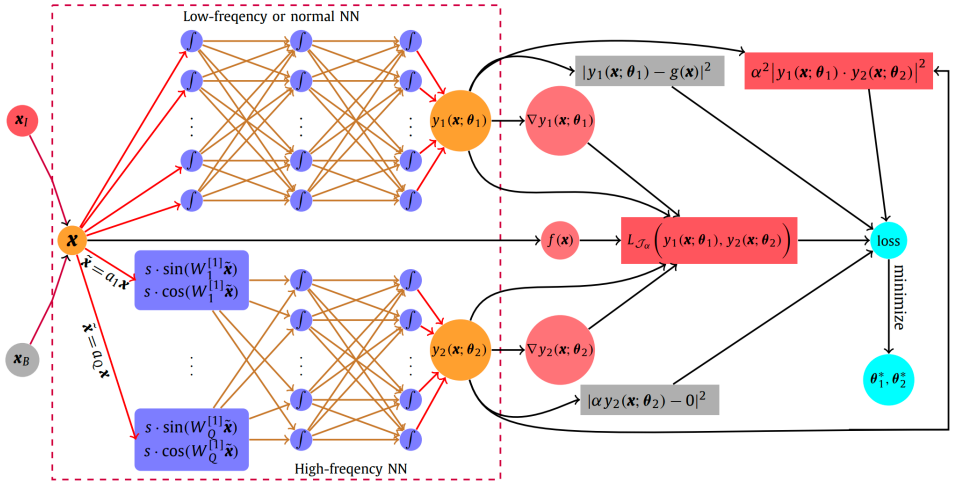


图 5.4: SD^{2}DNN 架构图，引用自Li et al. (2023)

为了验证所提出架构的有效性，文章给出了如下一维椭圆方程的例子：

$$\begin{cases} -\operatorname{div}\left(A(x)\nabla u(x)\right) = f(x), \quad x \in [0,1], \\ u(0) = u(1) = 0, \end{cases} \tag{5.7}$$

其中，系数 $A(x)$ 具有三种尺度 $\mathcal{O}(1), \mathcal{O}(\varepsilon_1), \mathcal{O}(\varepsilon_2)$，$1 \gg \varepsilon_1 = 0.1 \gg \varepsilon_2 = 0.01$，具体形式为：

$$A(x) = \left(2 + \cos\left(2\pi \frac{x}{\epsilon_1}\right)\right)\left(2 + \cos\left(2\pi \frac{x}{\epsilon_2}\right)\right), \tag{5.8}$$

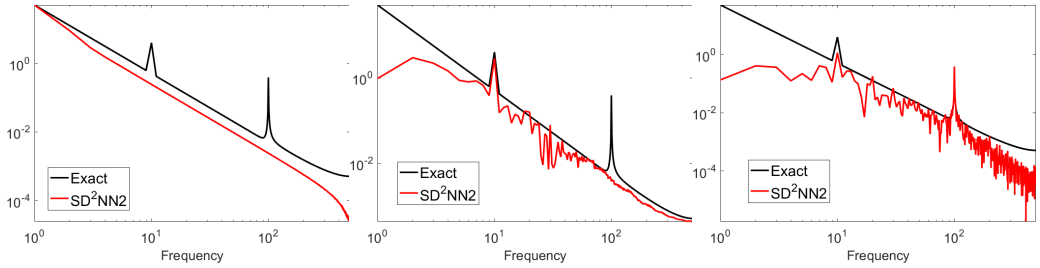
此时，方程(5.7)的解析解为:

$$u(x) = x - x^2 + \frac{\epsilon_1}{4\pi} \sin\left(2\pi \frac{x}{\epsilon_1}\right) + \frac{\epsilon_2}{4\pi} \sin\left(2\pi \frac{x}{\epsilon_2}\right). \tag{5.9}$$

文章在这个例子上采用了一种记为 SD^{2}DNN2 的架构,其中第一部分采用一个低频 MscaleDNN 模块（没有添加额外的拉伸尺度），第二部分由一个中频 MscaleDNN 和一个高频 MscaleDNN 组成（中频添加的拉伸尺度数值小于高频）。三个网络的输出和用来逼近目标函数。运用这样的架构去参数化方程，损失函数共有三部分，第一部分是方程残差项 L_{eq}，采用方程的能量作为该部分损失，让神经网络输出满足控制方程；第二部分 L_{orth} 为正交约束的损失，让不同频率模块的输出尽量正交；第三部分 L_{bd} 是来自边界的损失，让神经网络满足边界条件。形式如下：

$$L_{sum} = L_{eq} + L_{orth} + L_{bd}, \tag{5.10}$$

具体可以参考 A12 部分关于神经网络解 PDE 的内容。图5.5展示了 SD^{2}DNN2 的三个子模块学习到的结果。可以看出，各个子模块成功捕获了对应尺度的频率成分，与解析解吻合得很好，不同尺度部分均得到了很好的重构。



(a) SD^{2}DNN2 低频模块结果　(b) SD^{2}DNN2 中频模块结果　(c) SD^{2}DNN2 高频模块结果

图 5.5: 三尺度一维的例子。图片来源于Li et al. (2023)。

5.2 神经辐射场

神经辐射场 (Neural Radiance Fields, NeRF)Mildenhall et al. (2021) 能够从一组图像中学习并生成高质量的新视图，其视觉质量已达到了当时的最先进 (state-of-the-art) 水平。NeRF 的演示令人印象深刻，启发了许多源自这种新方法的后续研究。在随后的发展中,NeRF 在三维

重建、视图合成和虚拟现实等领域得到了广泛应用。NeRF 的论文在提出的四年内引用数已超过 6400 次（谷歌学术）。

(a) 真解　(b) 有位置编码预测结果　(c) 无位置编码预测结果

图 5.6: 移除位置编码比较图。图片来源于Mildenhall et al. (2021)。

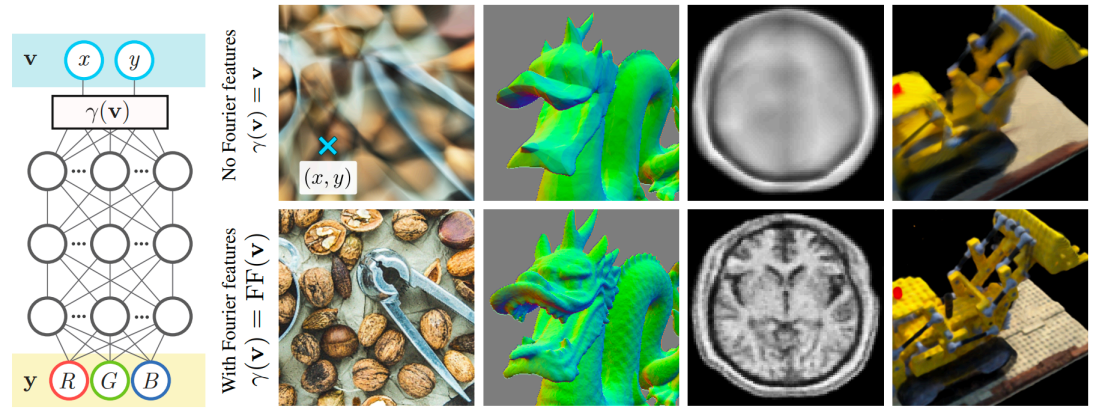
NeRF 的核心思想是使用神经网络来表示连续的三维空间，并通过体积渲染的方式生成新视图。在 NeRF 中，位置编码扮演了至关重要的角色。它实际上是采用正余弦函数作为第一层隐藏层的激活函数。在 NeRF 的原始文章中，作者使用多层全连接神经网络来拟合神经辐射场。对于输入 $\boldsymbol{x}$，NeRF 先将每个维度映射到更高维空间，映射函数为 $\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p))$，其中 p 是 $\mathbf{x}$ 的某一维度，L 是一个超参数。这是一种特殊的多尺度结构，把所有的可训练参数都设为常数，主要是让伸缩尺度起到关键作用。为了直观展示位置编码的效果, 图5.6给出了 NeRF 有无位置编码时的渲染结果对比。可以看到, 使用位置编码后,NeRF 生成的图像包含了更多的高频细节, 视觉质量明显提升。这说明了位置编码对 NeRF 性能的重要性。

5.3 Fourier 特征网络

Tancik et al. (2020) 给出了更一般的 Fourier 特征映射 γ，对于 $\mathbf{v} \in \mathbb{R}^d$，有:

$$\gamma(\mathbf{v}) = \left[a_1 \cos\left(2\pi \mathbf{b}_1^\top \mathbf{v}\right), a_1 \sin\left(2\pi \mathbf{b}_1^\top \mathbf{v}\right), \ldots, a_m \cos\left(2\pi \mathbf{b}_m^\top \mathbf{v}\right), a_m \sin\left(2\pi \mathbf{b}_m^\top \mathbf{v}\right)\right]^\top, \tag{5.11}$$

其中 $\{\mathbf{b}_i\}$ 是从某个概率分布中采样得到的频率向量。图 5.7中描述了 Fourier 特征的模型架构，以及有无 Fourier 特征的影响，通过对比两行的结果可以发现, 加入 Fourier 特征映射后, 神经网络生成的图像包含了更丰富的高频细节, 视觉质量得到明显提升。这直观地说明了 Fourier 特征映射能够增强神经网络对高频信号的建模能力。



(a) Coordinate-based MLP (b) Image regression $(x,y) \rightarrow$ RGB (c) 3D shape regression $(x,y,z) \rightarrow$ occupancy (d) MRI reconstruction $(x,y,z) \rightarrow$ density (e) Inverse rendering $(x,y,z) \rightarrow$ RGB, density

图 5.7: Fourier feature 架构图以及有无 Fourier feature 对模型预测的影响，引用自Li et al. (2023)

5.4 更多

有许多神经网络算法的工作设计了相应的方法以提升高频的学习，我们这里列一些例子。

为了提升高频的学习速度，设计新的激活函数也是一种常见的操作，例如自适应激活函数Jagtap & Karniadakis (2023) 或者混合激活函数 Jagtap et al. (2022)，以及变化比较剧烈的隐藏单元Agarwal et al. (2020)。

将深度神经网络与传统迭代方法相结合，可以分别加速计算问题中低频和高频的收敛速度Xu et al. (2020), Huang et al. (2020)。例如对于解方程，首先，使用 DNN 通过一定的优化步骤 (或迭代次数) 求解。然后，以训练好的 DNN 获得的初始值为起点，进行传统算法迭代，利用迭代方法的平滑特性消除高频误差，但这种方法可能只能处理比较低维的问题，对于具有复杂边界的问题，无网络的神经网络方法仍然是有优势的。

用于求解偏微分方程的随机特征方法 (randon feature method, RFM)Chen et al. (2022) 将求解空间划分为多个子空间后，把每个子空间的中心都移到中心，然后配上一个多尺度系数，例如

$$\frac{\boldsymbol{x} - c_i}{r_i}, \tag{5.12}$$

，其中 c_i 是第 i 个子空间的中心，r_i 是一个预先给定的随机数，起到多尺度网络中调节尺度的作用。

Chapter 6

频率原则的机制分析

在第四章中，我们介绍了频率原则的概念，并通过实验验证了其有效性。同时，我们也讨论了频率原则对神经网络训练时产生现象的一些解释。频率原则指出，神经网络通常会以从低频到高频的顺序拟合训练数据。同时我们观察到，对于不满足频率原则的模型，其往往具有较差的泛化能力。因此，系统地刻画频率原则成立的条件、理解频率原则成立的原因是重要的。

在本章中，我们深入探讨了频率原则的底层机制。我们先详细分析了影响频率原则的主要因素，包括初始化权重大小、激活函数的选择以及损失函数的形式。通过实验和理论分析，我们揭示了这些因素如何影响神经网络在训练过程中的频率收敛行为。进一步，我们引入了线性频率原则（LFP）模型，并通过理论推导和实验验证，揭示了神经网络频率收敛背后的动力学机制和等价变分问题。我们还讨论了线性区域中的神经切线核（NTK）理论及其对频率原则的解释，分析了频率原则在更高维度和更复杂神经网络中的表现。此外，我们探讨了频率收敛极限问题，研究了在不同数据维度下，神经网络频率收敛的最小可能衰减速率。我们通过数值实验验证了理论分析，并讨论了这些结果对神经网络训练和泛化的影响。通过这些分析和实验，我们不仅加深了对频率原则的理解，也为神经网络在实际应用中的初始化和训练提供了理论依据和指导。

科普篇

频率原则是什么？哪些因素可能影响频率原则？

频率原则是指神经网络在训练过程中通常会先拟合数据中的低频分量，然后再逐渐拟合高频分量。影响这一现象的主要因素包括初始化权重的大小、激活函数的选择以及损失函数的形式等多种因素。

首先，**初始化权重的大小**对频率原则有显著影响。当初始化权重较大时，神经元输出中会包含更多的高频成分，导致高频拟合速度增加，从而削弱甚至破坏频率原则。因此，合理选择初始化权重的范围非常重要，以确保频率原则的有效性。

其次，**激活函数的选择**也会影响频率原则。常见的激活函数如 tanh 和 ReLU 在频率空间中是单调下降的，容易观察到频率原则。然而，如果使用在频率空间中先上升后下降的函数，如 Ricker 函数，则可能导致频率原则失效。

最后，**损失函数的形式**对频率收敛速度有显著影响。如果在损失函数中对某些特定频率赋予较大的权重，这些频率分量的收敛速度会加快。例如，添加导数项后的损失函数可以显著加快高频分量的收敛速度。

线性频率原则模型是什么？

为了深入理解频率原则的底层机制，我们提出了线性频率原则（LFP）模型。假设我们使用两层神经网络拟合数据集，通过分析神经网络输出的动力学方程，并将其近似成线性模型。我们发现，神经网络的输出频率在训练过程中是从低频到高频逐渐收敛的。

通过数值实验验证，我们发现 LFP 模型能够很好地解释神经网络的频率原则。该模型不仅加深了我们对频率原则的理解，也为神经网络在实际应用中的初始化和训练提供了理论依据和指导。

6.1 频率原则的影响因素

在从理论上证明频率原则之前，首先对频率原则的影响因素进行讨论，这将辅助理解频率原则的成立条件。直观上，神经网络初始化权重大小将可能对频率原则产生影响，下文即从此出发展开研究。

6.1.1 初始化权重大小的影响

初始化权重规模在频率原则中具有重要作用。大初始化设定下，频率原则会变弱甚至消失。从时域的角度来看, 我们考虑一个经简化的 tanh 神经元的输出 $\tanh(wx)$。我们取 $w=1$ 和 $w=100$ 分别代表相同激活函数下小初始化和大初始化设定。如图 6.1所示, 当初始化较大时, 简化神经元的输出在 $x=0$ 附近的变化更加剧烈, 这意味着其输出中会包含更多的高频成分。当神经网络由这样的激活函数组合而成时，模型的高频拟合速度会增加，从而使得模型的频率原则减弱甚至失效。

如图6.2所示, 红色实线为初始化模型输出，蓝色虚线为被拟合的曲线。当初始化权重较小时, 由于神经网络不会去拟合超出数据集最高频率的信息。但是如果初始化权重设置得太大,

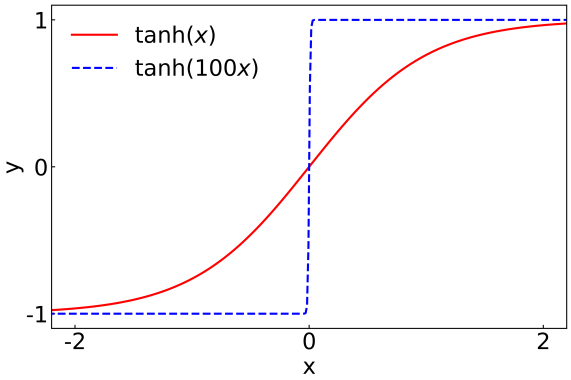
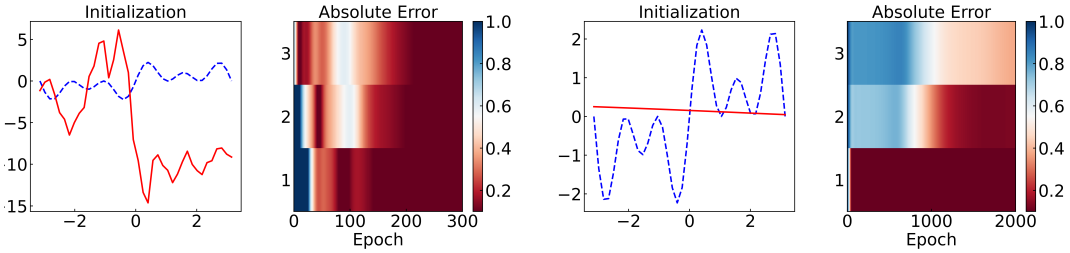


图 6.1: 小初始化和大初始化设定下 tanh 神经元的输出

就会在初始状态下引入很多高频信息, 其中有些甚至比数据集中的高频信息还要高。在这种情况下，模型的频率原则减弱甚至失效。同时，神经网络不会学习比数据集频率更高的信息, 这也就意味着训练过程无法完全抵消初始化带来的高频干扰。这可以通过相对误差图像得到证实，大初始化权重在训练初始阶段的高频拟合速度大于低频，且有大幅波动。小初始化权重从低频到高频其拟合速度依次递减，有助于网络平稳地学习数据的主要特征，避免过度拟合。



(a) large initialization

(b) small initialization

图 6.2: 初始化权重较大与初始化权重较小的情况下拟合一个函数的结果

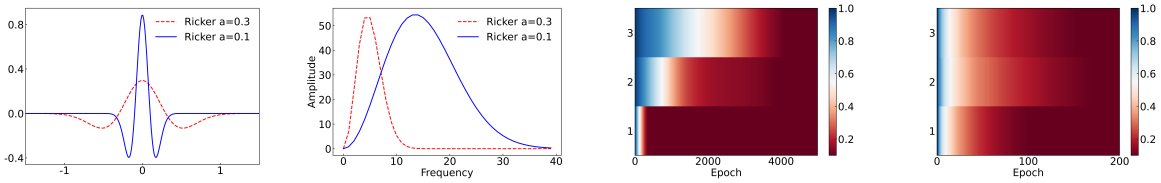
6.1.2 不同激活函数的影响

除了初始化权重外，我们发现不同激活函数也会影响频率收敛速度。对于大部分的激活函数, 例如 tanh 和 ReLU, 它们在频率空间中是单调下降的。因此, 在这些情况下, 我们可以容易

地观察到频率原则。然而，如果我们设计一类在频率空间中不是单调下降的函数，例如这类函数在频率空间先单调上升直到某个高频，然后再单调下降，那么我们可能会观察到频率原则失效。为了验证这一点，我们尝试用带参数 a 的 Ricker 函数作为激活函数：

$$\frac{1}{15a}\pi^{1/4}\left(1-\left(\frac{x}{a}\right)^2\right)\exp\left(-\frac{1}{2}\left(\frac{x}{a}\right)^2\right). \tag{6.1}$$

当 a 比较小时，Ricker 函数从一个更高的频率开始衰减。我们用一个全连接网络来拟合 $\sin(x)+\sin(3x)+\sin(5x)$。如图6.3所示，对于 $a=0.3$ 的情况，当激活函数从一个比较低的频率开始衰减时，我们可以明显地观察到低频分量收敛得更快；然而，当我们调节 $a=0.1$ 时，此时激活函数从一个比较高的频率开始衰减时，我们无法观察到明显的频率收敛顺序。可以看到，激活函数在频率空间的行为会显著影响神经网络学习不同频率分量的顺序，激活函数的频率特性是影响频率原则的一个重要因素。



(a) spatial domain (b) frequency domain (c) $a=0.3$ (d) $a=0.1$

图 6.3: Ricker 激活函数在时域与频域空间的函数输出，及不同参数 a 对高低频成分拟合速度的影响

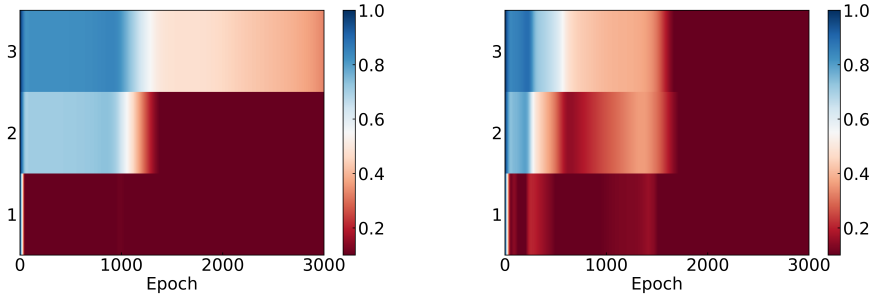
6.1.3 损失函数形式的影响

除了上述因素外，损失函数的形式也可以显著地影响频率的收敛。例如，如果我们在损失函数中显式地给一些特定的频率加入很大的权重，那么这些频率分量的收敛就可能会加快。为了验证这一点，我们可以考虑两种损失函数：一种是普通的均方损失 L_{nongrad}，另一种是在 L_{nongrad} 的基础上额外添加一个导数项的损失 L_{grad}，

$$L_{\text{nongrad}}=\sum_{i=1}^{n}(f_\theta(x_i)-f^*(x_i))^2/n, \tag{6.2}$$

$$L_{\text{grad}}=L_{\text{nongrad}}+\sum_{i=1}^{n}(\nabla_x f_\theta(x_i)-\nabla_x f^*(x_i))^2/n. \tag{6.3}$$

同样, 我们用神经网络来拟合 $\sin(x)+\sin(3x)+\sin(5x)$。如图6.4所示, 当损失函数中添加导数项后, 高频分量的收敛速度明显加快。这背后的关键原因是, 在频率空间中, 导函数的傅里叶变换等于原函数的傅里叶变换乘以对应的频率。这实际上相当于对高频分量施加了更大的权重。因此, 在一些特殊设定的任务中, 例如求解微分方程, 由于损失函数经常包含梯度信息, 频率原则可能会被弱化或者消失。



(a) L_{nongrad} (b) L_{grad}

图 6.4: 在损失函数 L_{nongrad} 和 L_{grad} 下，各个主要频率的收敛速度

6.2 频率原则的简单分析

首先，让我们考虑如下的网络结构：使用 $\sigma(x)=\tanh(x)$ 作为激活函数的两层深度神经网络（DNN），用于拟合一维目标函数 $f(x)$。选择 $\tanh(x)$ 作为激活函数是因为它的傅里叶变换容易计算。网络的输入和输出均为一维，表示如下：

$$h(x)=\sum_{j=1}^{m} a_j\sigma\left(w_jx+b_j\right). \tag{6.4}$$

对激活函数 $\sigma(w_jx+b_j)$ 进行傅里叶变换，我们得到：

$$\hat{\sigma(w_jx+b_j)}(k)=\frac{2\pi i}{|w_j|}\exp\left(\frac{ib_jk}{w_j}\right)\frac{1}{\exp(-\frac{\pi k}{2w_j})-\exp(\frac{\pi k}{2w_j})}. \tag{6.5}$$

从上式可以看出，随着频率的增加，$\tanh(x)$ 在频率空间呈指数衰减。这种快速衰减的主要原因是 $\tanh(x)$ 在时域空间是一个光滑、无穷阶可导的函数。对 $h(x)$ 进行连续傅里叶变换，我

们得到：

$$\hat{h}(k) = \sum_{j=1}^{m} \frac{2\pi i a_j}{|w_j|} \exp\left(\frac{ib_j k}{w_j}\right) \frac{1}{\exp(-\frac{\pi k}{2w_j}) - \exp(\frac{\pi k}{2w_j})} \tag{6.6}$$

$$\approx \sum_{j=1}^{m} a_j \exp\left(\frac{ib_j k}{w_j}\right) \exp\left(-\left|\frac{\pi k}{2w_i}\right|\right). \tag{6.7}$$

注意，本节是一个定性的分析。第一行等号右端在频率 $k = 0$ 的时候会有奇性，但我们在此不深入分析。第二行的约等式是在频率比较大的时候近似成立。定义某个频率上的损失函数为：

$$L(k) = \frac{1}{2}|\hat{h}(k) - \hat{f}(k)|^2. \tag{6.8}$$

根据 Parseval 定理，频域下的总损失函数等于时域上的总损失函数，即：

$$L = \int L(k)dk = \int \frac{1}{2}|f(x) - h(x)|^2 dx. \tag{6.9}$$

使用梯度下降法对网络进行训练，参数更新公式为：

$$\theta^{(n+1)} \leftarrow \theta^{(n)} - \eta \sum \frac{\partial L(k)}{\partial \theta}, \tag{6.10}$$

其中 η 是学习率。接下来，我们考察 $\frac{\partial L(k)}{\partial \theta}$ 的表达式：

$$\left|\frac{\partial L(k)}{\partial \theta}\right| \approx |\hat{h}(k) - \hat{f}(k)| \exp\left(-\left|\frac{\pi k}{2w_i}\right|\right) G(\theta, k), \tag{6.11}$$

其中 $G(\theta, k)$ 是一个 $O(1)$ 的函数。从上式可以看出，如果低频成分尚未收敛，即 $A(k) = |\hat{h}(k) - \hat{f}(k)| > 0$，且参数 w_i 较小时，$\exp(-|\pi k/2w_i|)$ 项起主导作用。这意味着低频成分对梯度的贡献远大于高频成分，梯度下降的方向实际上就是更接近低频收敛的方向。因此，从这个理想模型中，我们可以得到一个重要的认识：深度学习对低频的偏向性源于激活函数的光滑性和正则性，以及基于梯度下降的训练方法。类似地，对于 ReLU(x) 激活函数，也可以证明其在频率空间呈多项式衰减。

根据公式6.11, 我们可以看出频率原则成立与否与上一节讨论的三个因素之间的联系。其中 $|\hat{h}(k) - \hat{f}(k)|$ 受损失函数形式的影响，$\exp\left(-\left|\frac{\pi k}{2w_i}\right|\right)$ 的形式得益于激活函数（上文推导中以 tanh 为例）的形式，同时这一项的大小受 w_i, 也就是参数大小的影响。

若我们将上面的理想模型写成严格的数学表述，就有

定理 4. *考虑一个以 $\sigma(x) = \tanh(x)$ 作为激活函数的两层的神经网络，任取两个频率 k_1, k_2，使得 $\left|\hat{f}(k_1)\right| > 0, \left|\hat{f}(k_2)\right| > 0$，$|k_2| > |k_1| > 0$，一定存在正的常数 c 和 C，使得对足够小的 δ，有*

$$\frac{\mu\left(\left\{W : \left|\frac{\partial L(k_1)}{\partial \theta_{lj}}\right| > \left|\frac{\partial L(k_2)}{\partial \theta_{lj}}\right| \quad for\ all \quad l, j\right\} \cap B_\delta\right)}{\mu(B_\delta)} \geq 1 - C\exp(-c/\delta) \tag{6.12}$$

其中 $B_\delta \subset \mathbb{R}^m$ 是一个球心在原点，半径为 δ 的圆球，$\mu(\cdot)$ 是勒贝格测度。

在上述定理中，对于任意两个都没收敛的频率 k_1,k_2，可以证明只要我们的参数在某个球内，当球的半径很小，也就是参数比较小的情况下，低频梯度大于高频梯度部分的测度比上整个空间的测度以指数形式趋近于 1。也就是说，当 $\delta \to 0$ 时，低频对应的梯度几乎处处大于高频对应的梯度。

这部分理论不仅给我们呈现了频率原则产生的原因，同时也告诉我们在哪些情况下频率原则会不明显、甚至不成立，这与上文的实验现象是一致的。

6.3 线性频率原则模型*

在前文的描述中，我们大致建立了对频率原则的理论理解。然而，我们仅仅考虑了低频和高频成分哪个对损失函数梯度影响大，但并没有明确地写出神经网络的输出是如何随训练演化的。本节中，我们将提出了一个有效的模型来对较大初始化下，以 ReLU 为激活函数的两层宽神经网络的动力学行为进行建模，将这一模型称之为线性频率原则（Linear Frequency Principle，LFP）。

假设我们使用两层神经网络 $f(\boldsymbol{x},\boldsymbol{\theta}(t)) = \sum_{j=1}^m a_j\sigma(\boldsymbol{w}_j\cdot\boldsymbol{x}+b_j)$ 来拟合 n 个数据点构成的数据集 $\{(\boldsymbol{x}_i,y_i)\}_{i=1}^n$。损失函数选用均方损失函数。因此，神经网络参数更新满足以下动力学形式：

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\theta} = -\nabla_{\boldsymbol{\theta}} R_S(\boldsymbol{\theta}) = -\sum_{i=1}^n \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i,\boldsymbol{\theta})(f(\boldsymbol{x}_i,\boldsymbol{\theta})-y_i), \tag{6.13}$$

$$\boldsymbol{\theta}(0) = \boldsymbol{\theta}_0. \tag{6.14}$$

因此我们可以推导出神经网络输出的动力学方程如下（为方便起见，以下含时参数 $\boldsymbol{\theta}(t)$ 简记为 $\boldsymbol{\theta}$）：

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}(f(\boldsymbol{x},\boldsymbol{\theta})-f^*(\boldsymbol{x})) &= \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta})\cdot\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\theta} = -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta})\cdot\nabla_{\boldsymbol{\theta}} R_S(\boldsymbol{\theta})\\ &= -\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta})\cdot\sum_{i=1}^n \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i,\boldsymbol{\theta})(f(\boldsymbol{x}_i,\boldsymbol{\theta})-y_i)\\ &= -\sum_{i=1}^n K_m(\boldsymbol{x},\boldsymbol{x}_i)(t)(f(\boldsymbol{x}_i,\boldsymbol{\theta})-y_i),\end{aligned}$$

其中 $K_m(\boldsymbol{x},\boldsymbol{x}')(t) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}(t))\cdot\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}',\boldsymbol{\theta}(t))$。

设 $u(\boldsymbol{x},t) = f(\boldsymbol{x},\boldsymbol{\theta}(t)) - f^*(\boldsymbol{x})$ 为神经网络输出与目标函数之间的差值，则有：

$$\frac{\mathrm{d}}{\mathrm{d}t}u(\boldsymbol{x},t) = -\sum_{i=1}^n K_m(\boldsymbol{x},\boldsymbol{x}_i)(t)u(\boldsymbol{x}_i,t). \tag{6.15}$$

我们要计算神经网络的动力学方程，需要考察 $K_m(\boldsymbol{x}, \boldsymbol{x}_i)(t)$ 的性质。已有的一些理论工作 (Jacot et al. 2018, Lee et al. 2019) 表明，如果神经网络每层神经元个数足够多，给定比较大的参数初始化，那么对任何 $t \geq 0$，有 $\|\boldsymbol{\theta}(t) - \boldsymbol{\theta}(0)\| \ll 1$，这里 $\boldsymbol{\theta}(t)$ 表示 t 时刻神经网络的参数。这里我们可以对其有一个直观地理解，对于充分宽并且参数较大的情形，神经网络已经有了拟合大多数目标函数的能力，因此每个参数只需要改变很小一点，就可以使得模型输出快速朝目标函数靠拢。更加严格的讨论，如参数初始化与网络宽度的关系等，将放在下一章相图分析中详细展开。

因为参数变化相当小，神经网络 t 时刻的输出 $f(\boldsymbol{x}, \boldsymbol{\theta}(t))$ 可以被 $\boldsymbol{\theta}_0 = \boldsymbol{\theta}(0)$ 处的一阶泰勒展开逼近，这意味着神经网络的输出近似可以被表示为：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{\theta}(t)) \approx f(\boldsymbol{x}, \boldsymbol{\theta}_0) + \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} \cdot (\boldsymbol{\theta}(t) - \boldsymbol{\theta}_0). \tag{6.16}$$

注意到上述方程中，带有参数 $\boldsymbol{\theta}$ 的只有 $(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$，于是两边对参数 $\boldsymbol{\theta}$ 求导可得：

$$\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{\theta}_0). \tag{6.17}$$

因此我们有：

$$K_m(\boldsymbol{x}, \boldsymbol{x}')(t) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}(t)) \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}', \boldsymbol{\theta}(t)) \approx \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta}(0)) \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}', \boldsymbol{\theta}(0)) = K_m(\boldsymbol{x}, \boldsymbol{x}')(0), \tag{6.18}$$

于是，

$$\frac{\mathrm{d}}{\mathrm{d}t} u(\boldsymbol{x}, t) = -\sum_{i=1}^{n} K_m(\boldsymbol{x}, \boldsymbol{x}_i)(0) u(\boldsymbol{x}_i, t). \tag{6.19}$$

到这里我们就将动力学右端项的系数与时间解耦了，得到了网络输出与目标函数之间差值的变化与网络在训练数据集上表现的线性关系。为得到频率原则的动力学解释，我们只需将上式两端做傅里叶变换。虽然我们考虑的是两层神经网络，但傅里叶变换过程仍比较复杂，详细推导过程可以在相关论文 (Zhang et al. 2019, Luo et al. 2020) 中查阅。我们推导出来的由线性频率原则（LFP）表征的神经网络动力学模型如下：

$$\partial_t \mathcal{F}[u](\boldsymbol{\xi}, t) = -\mathcal{L}[\mathcal{F}[u_\rho]], \tag{6.20}$$

$$\begin{aligned}\mathcal{L}[\mathcal{F}[u_\rho]] = &\frac{\Gamma(d/2)}{2\sqrt{2}\pi^{(d+1)/2}\sigma_b\|\boldsymbol{\xi}\|^{d-1}} \mathbb{E}_{a(0), r(0)} \left[\frac{1}{r(0)} \mathcal{F}[\boldsymbol{g}_1]\left(\frac{\|\boldsymbol{\xi}\|}{r(0)}\right) \cdot \mathcal{F}[\boldsymbol{g}_1]\left(\frac{-\|\boldsymbol{\xi}\|}{r(0)}\right)\right] \mathcal{F}[u_\rho](\boldsymbol{\xi}) \\ &- \frac{\Gamma(d/2)}{2\sqrt{2}\pi^{(d+1)/2}\sigma_b} \nabla \cdot \left(\mathbb{E}_{a(0), r(0)} \left[\frac{1}{r(0)\|\boldsymbol{\xi}\|^{d-1}} \mathcal{F}[g_2]\left(\frac{\|\boldsymbol{\xi}\|}{r(0)}\right) \mathcal{F}[g_2]\left(-\frac{\|\boldsymbol{\xi}\|}{r(0)}\right)\right] \nabla \mathcal{F}[u_\rho](\boldsymbol{\xi})\right),\end{aligned} \tag{6.21}$$

其中 $r(0) = |\boldsymbol{w}(0)|$，并且初始化参数 $a(0)$ 和 $\boldsymbol{w}(0)$ 是按照给定的分布随机生成的。$\boldsymbol{g}_1(z) := (\sigma(z), a\sigma'(z))^\intercal$，$g_2(z) := a\sigma'(z)$，$u_\rho(\boldsymbol{x}, t) = u(\boldsymbol{x}, t)\rho(\boldsymbol{x})$，$\rho(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta(\boldsymbol{x} - \boldsymbol{x}_i)$。我们通常

不考虑右端的高阶导数项，只关注第一项。并且可以通过一个合适的时间尺度放缩来忽略常数系数。因此，对于以 ReLU 为激活函数的神经网络，我们的 LFP 模型可以有以下相对简单的形式：

$$\partial_t \mathcal{F}[u](\boldsymbol{\xi}, t) = -(\gamma(\boldsymbol{\xi}))^2 \mathcal{F}[u_\rho](\boldsymbol{\xi}), \tag{6.22}$$

$$(\gamma(\boldsymbol{\xi}))^2 = \mathbb{E}_{a(0),r(0)} \left[\frac{r(0)^3}{16\pi^4 \|\boldsymbol{\xi}\|^{d+3}} + \frac{a(0)^2 r(0)}{4\pi^2 \|\boldsymbol{\xi}\|^{d+1}} \right]. \tag{6.23}$$

公式6.22中所示的动力学模型表明，低频部分要比高频部分收敛速度要快。因此神经网络更加倾向于获得低频系数较大的函数。

注 18. 注意到公式 *6.23*中 $\|\boldsymbol{\xi}\|$ 有两个指数，分别是 $-(d+3)$ 和 $-(d+1)$. 第一项是因为对 ReLU（即 a 求导留下的部分）做傅里叶变换产生的，第二项是因为对 ReLU 的导数（即 w, b 求导留下的部分）做傅里叶变换产生的。

注 19. 注意到，这个动力学方程的左右两端的 u 差一个采样密度。左端的 u 是整个神经网络函数，而右端因为乘以了采样密度，使得其只受训练点的影响。这符合神经网络的训练过程，即训练目标着眼于将训练点拟合好。

注 20. 对于以 tanh 为激活函数的神经网络，其 *LFP* 模型中的 $\gamma(\boldsymbol{\xi})^2$ 为

$$\gamma(\boldsymbol{\xi})^2 = \frac{1}{\|\boldsymbol{\xi}\|^{d-1}} \mathbb{E}_{a,r} \left[\frac{\pi^2}{r} \operatorname{csch}^2 \left(\frac{\pi^2 \|\boldsymbol{\xi}\|}{r} \right) + \frac{4\pi^4 a^2 \|\boldsymbol{\xi}\|^2}{r^3} \operatorname{csch}^2 \left(\frac{\pi^2 \|\boldsymbol{\xi}\|}{r} \right) \right], \tag{6.24}$$

其中 csch 为双曲余割函数。

6.4 线性频率原则的等价变分问题**

在论文Zhang et al. (2019), Luo et al. (2020) 中，还推导了与 LFP 等价的变分模型：

$$\min_f \int_{\mathbb{R}^d} (\gamma(\boldsymbol{\xi}))^{-2} |\mathcal{F}[f](\boldsymbol{\xi}) - \mathcal{F}[f_{\text{ini}}](\boldsymbol{\xi})|^2 \, \mathrm{d}\boldsymbol{\xi}, \tag{6.25}$$

$$\text{s.t.} f(\boldsymbol{x}_i) = y_i, i = 1, \ldots, n \tag{6.26}$$

当频率 $\boldsymbol{\xi}$ 增大时，权重项 $(\gamma(\boldsymbol{\xi}))^{-2}$ 也会随之增大，这意味着对于 $f(\boldsymbol{x}) - f_{\text{ini}}(\boldsymbol{x})$ 中的高频项施加了更大的惩罚。根据这个分析也可以得出 $f(\boldsymbol{x}) - f_{\text{ini}}(\boldsymbol{x})$ 将会是一个低频函数。但是我们注意到如果 $f_{\text{ini}}(\boldsymbol{x})$ 不是 0 的话，那么 $f(\boldsymbol{x})$ 将会含有 $f_{\text{ini}}(\boldsymbol{x})$ 中的高频成分。论文Zhang et al. (2020) 提出一种反对称初始化的技术（图 6.5），可以使神经网络的初始输出 $f_{\text{ini}}(\boldsymbol{x})$ 为 0。其方法为将网络定义为两个相同初始化的全连接神经网络之差：

$$f_{\text{ASI}}(\boldsymbol{x}, \Theta(t)) = \frac{\sqrt{2}}{2} f(\boldsymbol{x}, \boldsymbol{\theta}(t)) - \frac{\sqrt{2}}{2} f(\boldsymbol{x}, \boldsymbol{\theta}'(t)), \tag{6.27}$$

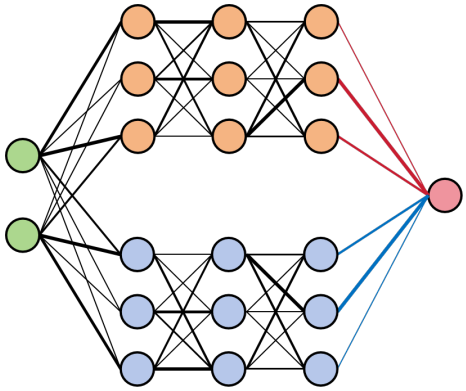


图 6.5: 反对称初始化技术示意图

其中，$\Theta(t) = (\boldsymbol{\theta}(t), \boldsymbol{\theta}'(t))$ 是模型的全部参数，且 $\boldsymbol{\theta}(0) = \boldsymbol{\theta}'(0)$，这也是反对称这一名称的由来。在图 6.5中，上下两个全连接神经网络的参数初始值完全相同，最终输出 $f_{\text{ASI}}(\boldsymbol{x}, \Theta(t))$ 为两个神经网络之差。这种反对称技巧可以使得 $f_{\text{ASI,ini}} = 0$，并且保持神经网络的动力学仍满足 LFP 动力学公式 6.19。这样，神经网络最后的输出将是一个低频函数，从另一个角度得出了频率原则的结论。

另外，我们注意到，上述变分模型的右端实际上可以看成 $f - f_{\text{ini}}$ 的一种半范数，现在我们正式在离散傅里叶变换下定义这个半范数，称为 FP-范数，定义为：

$$\|f\|_\gamma = \Big(\sum_{\boldsymbol{\xi}\in\mathbb{Z}^d} (\gamma(\boldsymbol{\xi}))^{-2} |\mathcal{F}[f](\boldsymbol{\xi})|^2\Big)^{\frac{1}{2}}. \tag{6.28}$$

这里我们约定 $\gamma(\mathbf{0})^{-1} = 0$，因此 FP-范数是半范数。因此变分问题也可以解读为在所有满足约束条件 $f(\boldsymbol{x}_i) = y_i, i = 1, \dots, n$ 的函数中，寻找 $\|f - f_{\text{ini}}\|_\gamma$ 最小的解 f。

我们还可以利用这个范数定义一个 FP-空间，定义如下：

$$F_\gamma(\Omega) = \{f \in L^2(\Omega) : \|f\|_\gamma < \infty\}, \tag{6.29}$$

其中 Ω 为数据点 x 所处的空间。由于我们考虑的是最小化 $\|f - f_{\text{ini}}\|_\gamma$ 的问题，因此很自然地应该满足 $f - f_{\text{ini}} \in F_\gamma$。

之所以要定义 FP-范数和 FP-空间，是因为我们想通过它们建立一个泛化误差的先验估计，详细的推导可以参考论文Zhang et al. (2019), Luo et al. (2020)。这里我们不加证明地给出泛化误差的估计定理：

定理 5. *假设目标函数 $f^* \in F_\gamma$，函数 f 是以下问题的解*

$$\min_{f - f_{\text{ini}} \in F_\gamma} \|f - f_{\text{ini}}\|_\gamma, \quad \text{s.t. } f(\boldsymbol{x}_i) = y_i, i = 1, \dots, n. \tag{6.30}$$

那么对任何映射 $\gamma : \mathbb{Z}^d \to f^+$，$\delta \in (0,1)$，对于蒙特卡洛采样生成的数据集 $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$。至少以 $1-\delta$ 的概率满足以下公式：

$$L(f) \leq \|f^* - f_{\text{ini}}\|_\gamma \|\gamma\|_{\ell^2} \left(\frac{2}{\sqrt{n}} + 4\sqrt{\frac{2\log(4/\delta)}{n}} \right) \tag{6.31}$$

从以上定理我们可以看到，泛化误差随着训练数据量 n 增大而降低，并且衰减速率为 $O(1/\sqrt{n})$，这也是神经网络能够克服维数灾难的重要原因。从右端项还可以看出，假设我们通过一定的技术（如之前提到的反对称初始化）使得 $f_{\text{ini}} = 0$，那么先验泛化误差估计还依赖于 $\|f^*\|_\gamma$。对于神经网络对应的 FP-范数，其会对高频部分施加更大的权重。于是，f^* 的高频项越多，那么 $\|f^*\|_\gamma$ 就越大，因此，泛化误差的上界也越大；反之，如果 f^* 主要是由低频部分组成，那么 $\|f^*\|_\gamma$ 就会相对较小，因此泛化误差上界也会相应变小。这一项也很好地体现出之前章节中没有免费午餐定理提到的算法与目标函数的匹配关系这一话题，$\|\cdot\|_\gamma$ 由算法提供，f^* 是目标函数，二者耦合在一起共同影响了泛化误差上界的估计。

我们通过一个简单的数值实验来对我们提出的泛化误差估计定理做一个验证，我们使用以 ReLU 为激活函数的两层神经网络来拟合区间 $[0,1]$ 上的目标函数 $f^*(x) = \sin(2\pi k x)$，其中 k 是目标函数的频率。我们使用相同的实验设定来拟合不同的 k 对应的目标函数，这样即可保证公式 6.31右边只有 $\|f^* - f_{\text{ini}}\|_\gamma$ 这一项在变化。图 6.6展示了泛化误差随目标函数频率的变化情况，与我们公式推导的结果相吻合。

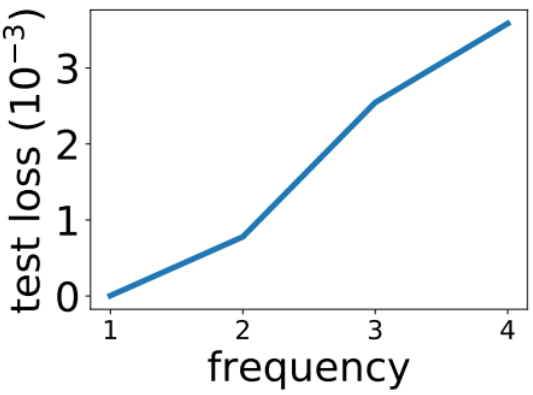
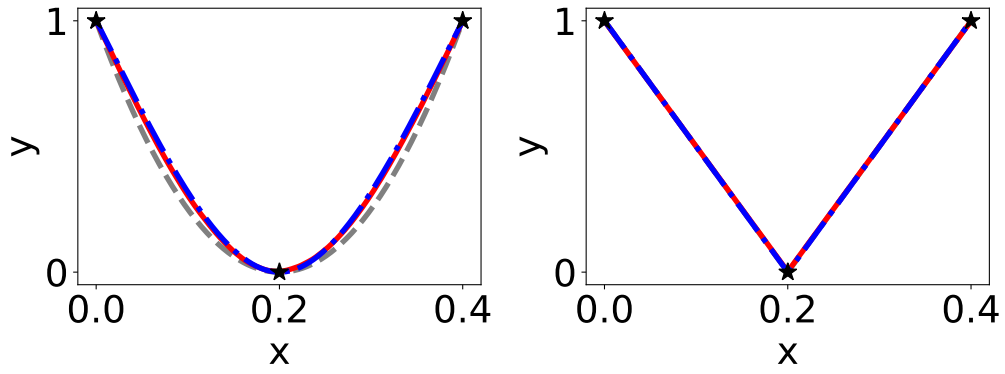


图 6.6: 使用神经网络拟合不同频率 k 对应的目标函数 $f^*(x) = \sin(2\pi k x)$ 时，测试误差随目标函数频率的变化

上述理论对于理解神经网络的过参数化谜团具有重要帮助，对于过参数化的神经网络，通过 LFP 动力学模型我们可以得出它的输出实际上是在解变分问题 6.30。根据定理 5，我们可以通过公式 5对其泛化误差上界做一个估计。对于低频占有的目标函数 f^*，公式右端项较小。



(a) (b)

图 6.7: 神经网络模型、LFP 变分模型和样条插值效果对比

因此，可以得出神经网络不太会在这样的目标函数上过拟合。由此我们可以总结出神经网络的几条特性：

- **偏好**：神经网络偏好输出低频函数。
- **优势**：神经网络在拟合低频占有的目标函数时往往具有良好的泛化性。
- **缺点**：神经网络在拟合高频目标函数时可能会过拟合。

我们还通过数值实验进一步验证了 LFP 等价变分模型是两层充分宽的神经网络模型的良好近似。为了验证公式6.22中两个加和项的准确性，我们考察宽度为 10000 的两层 ReLU 神经网络，我们分别设置两组超参数 a，$\boldsymbol{w}$，b 分别使得 $1/\xi^4$ 和 $1/\xi^2$ 项占主导。具体实验方法可以看附录 6.6。

- **通过设置 $\mathbb{E}_r[r^2] \gg \mathbb{E}_a[a^2]$，使得 $1/\xi^4$ 项占主导地位**：如图 6.7(a) 所示，神经网络通过平滑函数（实线）对训练数据（星号）进行插值，这与 LFP 模型的预测（点线）和三次样条插值（虚线）几乎重叠。
- **通过设置 $\mathbb{E}_r[r^2] \ll \mathbb{E}_a[a^2]$，使得 $1/\xi^2$ 项占主导地位**：如图 6.7(b) 所示，神经网络通过函数（实线）对训练数据（星号）进行插值，这与 LFP 模型的预测（点线）和线性样条插值（虚线）几乎重叠。

注 21. 图中还展示了一条代表样条插值 *(McKinley & Levine 1998)* 的灰色虚线。事实上，对一维区间 $[a,b]$ 上问题，三次样条插值也可以解释为最小化以下泛函的变分问题：

$$\min_f \int_a^b |f''|^2 \,\mathrm{d}x. \tag{6.32}$$

傅里叶变换具有一条很重要的性质：$\mathcal{F}[f'](\xi) = -2\pi j\xi\mathcal{F}[f](\xi)$，于是 $|\mathcal{F}[f']| = C|\xi||\mathcal{F}[f]|$。这里我们假设所有的常数都用 C 表示，对于图 *6.7(a)*，我们 *LFP* 模型对应的变分问题为

$$\min_f \int_R C|\xi|^4\mathcal{F}[f]^2 \,\mathrm{d}\xi = \int_R C\mathcal{F}[f'']^2 \,\mathrm{d}\xi. \tag{6.33}$$

因此这近似跟一个三次样条插值问题等价。同理，对于 $1/\xi^2$ 项占优的情况，可以推出它近似跟一个二次样条插值问题等价。

6.4.1 高频的收敛极限

通过 LFP 模型的变分形式 6.26我们可以看出，频率的收敛速度依赖于不同激活函数或网络框架计算出来的 $\gamma(\boldsymbol{\xi})^{-2}$。通常 $\gamma(\boldsymbol{\xi})^{-2}$ 随频率递增，这意味着对高频施加了较大的惩罚项，导致最后的解是低频占主导。但有时我们可能并不希望高频部分衰减太快，因为这时可能难以拟合好数据本身存在的较高频率信息。那么，一方面，我们得知大多数神经网络都具备频率原则，于是最后的输出是低频占有；另一方面，我们又希望设计激活函数或网络架构使得最后输出随频率衰减不要太快。于是我们不禁好奇：神经网络最后输出随频率衰减最慢可以是多少?

我们首先脱离神经网络和变分问题来思考一个极端情形，是否存在某个函数各频率振幅几乎不随频率衰减，同时还满足 $f(x_i) = y_i, \quad i = 1, \cdots, n$? 我们注意到高斯核函数有希望做到这点，因为对于一个高斯核 $k(x, x_i) = e^{-|x-x_i|^2/2\sigma^2}$，其傅里叶变换后为 $\mathcal{F}[k] = C\sigma \mathrm{e}^{-2\pi^2\sigma^2|\xi|^2}$，$C$ 是常数。由此可见，只要 σ 足够小，这个高斯核就对频率不敏感。我们使用核方法去拟合两个数据点 $\{(0.4, 1), (0.6, 1)\}$，当设置 $\sigma = 0.01$ 时，我们会得到如图 6.8所示的退化解（即在训练点之外迅速退化到 0）。通过这个例子我们发现，能以光滑曲线拟合好数据点的函数，其频率振幅是需要以一定速率随频率衰减的。

为了研究这个问题，受 ReLU 对应 $\gamma(\boldsymbol{\xi})^{-2}$ 的启发（公式 6.23），我们可以假设 $\gamma(\boldsymbol{\xi})^{-2}$ 为一个关于 $\|\boldsymbol{\xi}\|$ 的多项式函数，从而抽象出一个新的监督学习算法框架：

$$\min_{f\in\mathcal{H}} \int_{\mathbb{R}^d} \langle\boldsymbol{\xi}\rangle^\alpha \left|\mathcal{F}[f](\boldsymbol{\xi})\right|^2 \mathrm{d}\boldsymbol{\xi}, \tag{6.34}$$

$$\text{s.t.} \quad f(\boldsymbol{x}_i) = y_i, \quad i = 1, \cdots, n, \tag{6.35}$$

其中 $\langle\boldsymbol{\xi}\rangle = (1 + \|\boldsymbol{\xi}\|^2)^{\frac{1}{2}}$，$\mathcal{H} = \{f(x) | \int_{\mathbb{R}^d} \langle\boldsymbol{\xi}\rangle^\alpha \left|\mathcal{F}[f](\boldsymbol{\xi})\right|^2 \mathrm{d}\boldsymbol{\xi} < \infty\}$。

那么现在我们缓解高频惩罚的方式就变成了减小指数 α。但是我们在实验中发现，当 α 过小时该问题的解会退化为平凡解，即仅在数据点上有值，而在其他位置为零。而 α 较大时该

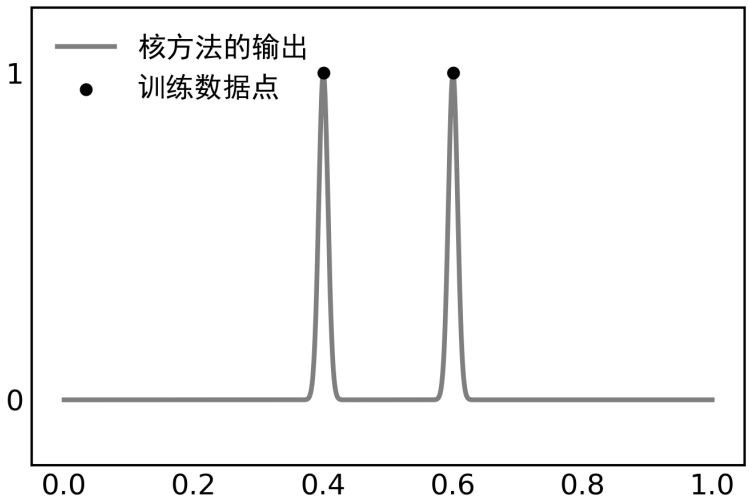


图 6.8: 高斯核方法拟合数据点

问题的解才具有光滑性。于是我们可以将寻找神经网络输出随频率衰减的下限问题转化为寻找能使得输出解光滑的最小 α 的问题。

Luo, Ma, Wang, Xu & Zhang (2021) 详细研究了这个临界问题。研究发现，当数据维度为 d 时，问题 (6.26) 中的 $\alpha = d$ 是一个临界情形。当 $\alpha > d$ 时，拟合的结果是一个比较连续光滑的解，而当 $\alpha < d$ 时，拟合的解退化，即输出一种类似 Dirac 函数的形状：在训练点其数值与真实值一样，而在非训练点其值为 0。

注 22. 这里我们构造一个例子来对 $\alpha < d$ 时解退化做一个直观解释。我们构造如下的 f_σ:

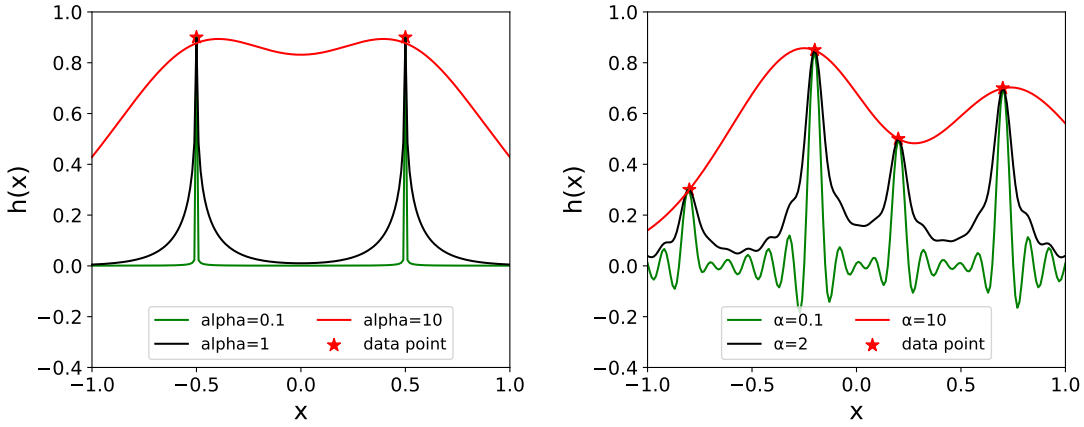
$$f_\sigma(\boldsymbol{\xi}) = \sum_{i=1}^{n} y_i e^{-\|\boldsymbol{x}-\boldsymbol{x}_i\|^2/2\sigma^2}, \tag{6.36}$$

其中 σ 充分小，于是 $\lim_{\sigma\to 0} f_\sigma(\boldsymbol{\xi}) = y_i$，$i = 1, 2, \cdot, n$。另一方面，我们有

$$\mathcal{F}[f_\sigma] = \sum_{i=1}^{n} y_i (2\pi)^{\frac{d}{2}} \sigma^d \mathrm{e}^{-2\pi^2\sigma^2\|\boldsymbol{\xi}\|^2}. \tag{6.37}$$

在忽略掉所有常数项后，变分问题的积分部分大致可以表述为：

$$\int_{\mathbb{R}^d} \|\boldsymbol{\xi}\|^\alpha \mathcal{F}[f_\sigma]^2 \,\mathrm{d}\boldsymbol{\xi} = \int_{\mathbb{R}^d} \|\boldsymbol{\xi}\|^\alpha \sigma^d \mathrm{e}^{-\sigma^2\|\boldsymbol{\xi}\|^2} \,\mathrm{d}\boldsymbol{\xi} = \sigma^{\mathrm{d}-\alpha} \int_{\mathbb{R}^{\mathrm{d}}} \|\sigma\boldsymbol{\xi}\|^\alpha \mathrm{e}^{-\|\sigma\boldsymbol{\xi}\|^2} \,\mathrm{d}(\sigma\boldsymbol{\xi}) = \mathrm{C}\sigma^{\mathrm{d}-\alpha}, \tag{6.38}$$



(a) 拟合 2 维空间中的 2 个数据点　　(b) 拟合 3 维空间中的 20 个数据点

图 6.9: 用不同的衰减率 α 拟合数据

其中 C 是一个常数。因此当 $\alpha < d$ 时，我们可以通过一系列序列 $\{f_\sigma\}_{\sigma\to 0}$ 来逼近训练点，同时使得变分问题的最小化目标为 0。于是这些函数的极限就是我们这个模型最终输出的函数 f。此外，这些函数具有一个非常鲜明的特点：在训练点 $\boldsymbol{x}_i$ 处的值为 y_i，但在其他点迅速退化至 0。因此最终输出会退化。

下面展示两个数值实验例子来验证这个临界现象。具体实验设定可以参见附录 6.6。如图6.9所示，通过改变 α，我们都能够看到在拟合训练数据点（星星）时，当 $\alpha < d$ 时，插值的解几乎没有意义，而当 $\alpha > d$ 时，插值的解看起来是有意义的解，因为它具有一定的光滑性。

这些理论表明，对于一个有意义的插值，α 不能太小，也就是最优解关于频率的衰减速度不能太慢，而是有一个依赖数据维度的上限。因此，当数据维度 d 越来越大时，最优解的频率衰减速度也会越来越大，这就产生了高频难以收敛的灾难。

神经网络的 LFP 模型同样适用于这一套算法分析框架，从上一节中我们得知，以 ReLU 为激活函数的神经网络背后的变分问题中，对频率的惩罚项系数为 $d+1$，因此，神经网络可以以一个光滑函数来拟合训练数据。同时，从公式 6.21和公式 6.26我们也应该意识到，对于两层神经网络，无论怎样设计激活函数，我们仍旧处于算法框架 6.35之下，因此神经网络的输出关于频率的衰减速度指标 α 至少为 d（这里我们的讨论均是在忽略掉公式 6.21右端高阶导数项前提下）。因此，这从另一个角度说明了神经网络偏好输出低频函数并且难以拟合高频信息是其固有特征。

6.4.2 线性区域的特征分析

在前文关于 LFP 模型的推导中，我们对神经网络动力学的切入点是将其近似成一个线性模型，即神经网络满足以下演化公式（公式 6.19）：

$$\frac{\mathrm{d}}{\mathrm{d}t}u(\boldsymbol{x},t) = -\sum_{i=1}^{n} K_m(\boldsymbol{x},\boldsymbol{x}_i)(0)u(\boldsymbol{x}_i,t), \tag{6.39}$$

其中 $K_m(\boldsymbol{x},\boldsymbol{x}_i)(0) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x},\boldsymbol{\theta}(0)) \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_i,\boldsymbol{\theta}(0))$ 是一个不随时间 t 演化的常数，这一思想被称为神经正切核（Neural Tangent Kernel, NTK）理论 (Jacot et al. 2018)。

我们从另一个角度来理解这个公式，假设我们的训练数据点为 $\boldsymbol{X} := (\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n)^T \in \mathbb{R}^{n\times d}$, $\boldsymbol{Y} := (\boldsymbol{y}_1,\cdots,\boldsymbol{y}_n)^T \in \mathbb{R}^n$。记 $\boldsymbol{u}(\boldsymbol{X},t) := (u(\boldsymbol{x}_1,t),\cdots,u(\boldsymbol{x}_n,t)) \in \mathbb{R}^n, \nabla_{\boldsymbol{\theta}} f(\boldsymbol{X},\boldsymbol{\theta}(t)) := (\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_1,\boldsymbol{\theta}(t)),\cdots,\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}_n,\boldsymbol{\theta}(t))) \in \mathbb{R}^{n\times M}$，$M$ 为训练参数数目。定义矩阵 $K^* \in \mathbb{R}^{n\times n}$ 满足 $K^*_{ij} = K_m(\boldsymbol{x}_i,\boldsymbol{x}_j)(0)$。于是，我们有以下动力学方程：

$$\frac{\mathrm{d}\boldsymbol{u}(\boldsymbol{X},t)}{\mathrm{d}t} = -K^*\boldsymbol{u}(\boldsymbol{X},t). \tag{6.40}$$

于是根据常微分方程理论，我们可以有以下解：

$$\boldsymbol{u}(\boldsymbol{X},t) = \mathrm{e}^{-K^*t}\boldsymbol{u}(\boldsymbol{X},0). \tag{6.41}$$

考虑对公式 6.40中的 K^* 做特征分解，设

$$K^* = P^T\Sigma P,$$

其中 $P = (\boldsymbol{v}_1,\boldsymbol{v}_2,\cdots,\boldsymbol{v}_n)^T \in \mathbb{R}^{n\times n}$, $\Sigma \in \mathbb{R}^{n\times n}$ 是一个对角矩阵。记 λ_i 为特征方向 $\boldsymbol{v}_i$ 对应的特征值。因此，K^* 可以写成

$$K^* = \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^T.$$

用 $\{\boldsymbol{v}_i\}$ 构成的基来表示 $\boldsymbol{u}(\boldsymbol{X},t)$,

$$\boldsymbol{u}(\boldsymbol{X},t) = \sum_{i=1}^{n} \alpha_i(t)\boldsymbol{v}_i.$$

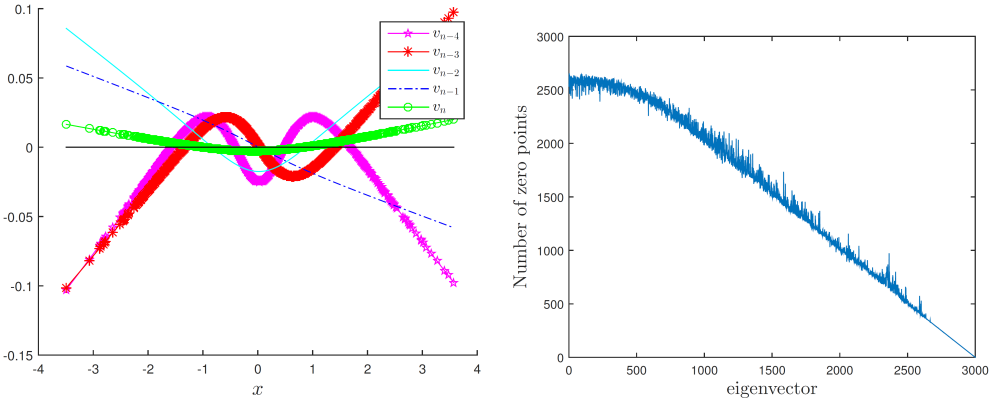
我们把上式代入公式 6.40，得：

$$\sum_{i=1}^{n} \frac{\mathrm{d}\alpha_i(t)}{\mathrm{d}t}\boldsymbol{v}_i = -\sum_{i=1}^{n} \alpha_i(t)K\boldsymbol{v}_i = -\sum_{i=1}^{n} \alpha_i\lambda_i(t)\boldsymbol{v}_i. \tag{6.42}$$

对每个分量可以得到

$$\frac{\mathrm{d}\alpha_i(t)}{\mathrm{d}t} = -\lambda_i\alpha_i(t). \tag{6.43}$$

这个方程有显示的解 $\alpha_i(t) = \exp(-\lambda_i t)\alpha_i(0)$。这些分析可以提供大量的信息。首先，在符合线性假设下，误差 $\boldsymbol{u}$ 是指数衰减的，收敛的快慢最终被最小特征值的大小所控制。一旦假设最小特征值是一个大于某个正数的情况，那误差就显然是指数收敛到全局最小值。有一系列的工作把这些分析做了严格化。其次，特征向量的信息提供了哪些信息可以快速收敛或者很慢收敛。我们可以将所有的点对 $\{(\boldsymbol{x}_j, \boldsymbol{v}_{i,j})\}_{j=1}^n$ 画在一张图上。图 6.10展示了一个以 ReLU 为激活函数的例子，在线性区域的设定下，我们使用 3000 个训练点对一维目标函数进行训练。其中最大特征值对应的特征方向为带圆圈的曲线，随着特征值变小，红色星星曲线明显更加振荡。可以看到，大特征值对应的特征方向的频率会比较低，这也说明了低频为什么会先收敛。为进一步定量刻画特征方向的振荡，我们数其穿过零点的次数，如图 6.10(b) 所示，我们设定特征值随着指标 (横坐标) 变大而变大，可以看出特征值越大，其穿过零点的次数越少，也可以理解为振荡越少，频率越低。这也符合我们前面谈到频率原则，即低频先收敛。



(a) (b)

图 6.10: K^* 矩阵的特征向量及其震荡情况

在我们研究频率原则的同时, 一些并行工作也从不同角度分析了线性假设下的频率原则 (或称为谱偏差)。这些工作主要关注在线性区域中, 两层宽度很大的 ReLU 网络的频率收敛行为。感兴趣的读者可以进行拓展阅读 (Ronen et al. 2019, Cao et al. 2019, Basri et al. 2020, Bordelon et al. 2020, Arora et al. 2019, Yang & Salman 2019)。

6.4.3 一般神经网络的频率原则理论

在6.2小节中, 我们讨论了一维情况下频率原则的理论。然而, 当我们试图将这些理论推广到高维情况时, 会遇到一些困难。首先, 在高维空间中进行傅里叶变换可能会变得非常复杂。其次, 对于一般的激活函数, 其傅里叶变换的解析表达式可能难以精确写出。

其次, 在6.3中讨论的 LFP 模型以及其他在神经切线核 (NTK) 区域研究频率收敛速度的理论, 又过于依赖网络的线性化假设。这些理论对于一般的中度过参数化的网络并不适用。

再次，以上的讨论均是对于两层神经网络进行的。对于多层神经网络，其内部涉及到非常复杂的函数复合过程，我们无法写出其傅里叶变换公式。

基于以上的种种原因, 对于一般的多层神经网络建立直接的频率原则理论分析是非常困难的。面对这些困难, 我们可以退而求其次, 进行定性的分析。基本的思路是利用函数的正则性和其傅里叶系数的衰减之间的关系。这里的正则性可以理解为函数可以被求导的次数。我们用到的主要分析工具就是以下反映函数与其导数在频率空间关系的公式:

$$\mathcal{F}[\partial f/\partial x](\xi) = -2\pi j\xi\mathcal{F}[f](\xi). \tag{6.44}$$

下面我们看几类常见的函数以及它们经过傅里叶变换后，傅里叶系数有什么性质。

Dirac 函数 Dirac 函数的定义为

$$\delta(x) = \begin{cases} 0, & x \neq 0 \\ \infty, & x = 0 \end{cases}. \tag{6.45}$$

且

$$\int_{-\infty}^{\infty} \delta(x)dx = 1. \tag{6.46}$$

Dirac 函数的形式给它赋予了一个特殊的性质，对任意函数，有

$$f(x) = \int f(y)\delta(y-x)dy. \tag{6.47}$$

为计算其傅里叶变换，我们只需考虑函数 $f(y) = \exp(-2\pi jy\xi)$,

$$1 = \exp(-2\pi jx\xi)|_{x=0} = \int \exp(-2\pi jy\xi)\delta(y)dy. \tag{6.48}$$

上式的右端实际上就是 Dirac 函数的傅里叶变换。因此，Diract 的傅里叶变换是一个常数。它在频率空间的幅度不随频率衰减。

注 23. *Dirac 函数的傅里叶变换性质和量子力中学的不确定原理有很深的联系。在时域空间, Dirac 函数只在一个点有值，可以理解它的分辨率是无穷细的，而它在傅里叶空间则是处处一样，完全无法通过函数值来区分其对应的频率，所以没有任何分辨率。*

Heaviside 函数 Heaviside 函数是由 Dirac 函数积分得到的，它的表达式为

$$H(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}, \tag{6.49}$$

根据导数关系 $\delta(x) = H'(x)$ 以及公式6.44，我们有

$$\mathcal{F}[\delta(x)](\xi) = -2\pi j\xi \mathcal{F}[H(x)](\xi), \tag{6.50}$$

于是有

$$\|\mathcal{F}[H(x)](\xi)\| = \frac{C}{\|\xi\|}, \tag{6.51}$$

我们用 C 表示常数，为了简便，不同常数，我们用同一个 C 表示。

ReLU 函数 如果我们对 Heaviside 函数再求一次积分，就得到了我们熟悉的 ReLU 函数，$\mathrm{ReLU}(x) = \max\{0, x\}$。

根据公式6.44，类似地，我们可以得到

$$\|\mathcal{F}[\mathrm{ReLU}(x)](\xi)\| = \frac{C}{\|\xi\|^2}. \tag{6.52}$$

依此类推，我们可以得到当一个函数可以求导的次数越高（正则性），它在频率空间的幅度随频率增长衰减得越快。如果一个函数无穷可导，那这将是指数衰减。

在定性上，我们可以只关注频率幅度的衰减速率，利用梯度下降，这些衰减会对梯度产生影响，从而对损失函数的不同频率成分的收敛产生影响。对于函数的复合，我们只需要估计复合函数的正则性即可。Luo et al. (2019) 严格发展了这些理论，并且将损失函数的类型推广到 L^p 损失函数 ($p \geq 2$)，感兴趣的读者可以自行查阅。

6.5 习题

- 频率原则的成立与否受到哪些因素的影响？请列举并解释其中至少三个因素。
- 在本章的实验中，我们观察到初始化权重大小会影响频率原则的成立。

  (a) 过大的初始化权重会对频率原则产生什么影响？为什么？

  (b) 在实际应用中，如何选择合适的初始化权重范围以确保频率原则的有效性？
- 激活函数的形式如何影响频率原则？

  (a) 常见的激活函数如 ReLU 和 tanh 在频域的衰减特性如何？这对频率原则意味着什么？

  (b) 我们尝试使用 Ricker 函数作为激活函数，并通过调节参数 a 来改变其频域特性。请解释不同 a 值下频率原则成立与否的原因。

- 本章引入了包含梯度信息的损失函数 L_{grad}。

  (a) L_{grad} 相比普通的均方损失 L_{nongrad} 有何不同？为什么它会加快高频分量的收敛速度？

  (b) 在频域中，导函数的傅里叶变换与原函数的傅里叶变换满足什么关系？这一关系如何影响不同频率分量的权重？

  (c) 你能否想到一种损失函数的设计方式，使得频率原则更加显著？提示：考虑在损失函数中引入频率相关的权重项。

- 在本章的简单分析部分，我们以两层 tanh 网络为例推导了频率原则成立的条件。

  (a) 为什么选择 tanh 作为激活函数？如果换成其他常见的激活函数（如 ReLU），结论是否仍然成立？

  (b) 基于公式 6.11，尝试解释大初始化权重、损失函数形式等因素与频率原则之间的联系。

- 线性频率原则 (LFP) 模型是如何推导出来的？它从哪些方面揭示了神经网络频率原则的本质？

- 基于 LFP 模型，我们可以得到一个神经网络输出的等价变分模型。请描述这个变分模型，并分析它与频率原则的关系。进一步，这个变分模型对于理解神经网络的泛化有什么启示？

- 在线性区域中，神经网络的谱偏差 (频率原则) 可以用神经切线核 (NTK) 的特征值和特征向量来解释。请分析 NTK 的特征值、特征向量与频率的关系。这一分析对于理解神经网络训练和泛化有什么意义？

- 对于 $f(x) = \sin 2\pi k x$，其中 $k \in \mathbb{Z}$。计算其 FP-范数。

- 对于一般的神经网络，函数的正则性与其傅里叶变换的衰减速度有什么关系？这一关系对于分析神经网络的频率原则有什么启示？

6.6 附录：频率原则理论相关实验

6.6.1 LFP 模型数值实验

下面我们将通过数值实验来进一步说明 LFP 模型是两层充分宽的神经网络模型的良好近似。在数值上，我们无法考察一个函数的全部频率，因此我们只能对频率做截断，即假设函数

$f - f_{\text{ini}}$ 仅有有限频率，设：

$$f - f_{\text{ini}} = \sum_{j \in I} \left[a_j \sin\left(2\pi j x_i\right) + b_j \cos\left(2\pi j x_i\right) \right], \tag{6.53}$$

其中，$I = \left\{ \frac{0}{2L}, \frac{1}{2L}, \cdots, \frac{K-1}{2L} \right\}$ 为频率集合，L 是训练集的取值范围宽度，$K = 100$ 远大于我们所使用的训练集样本数量。并且我们有

$$|\mathcal{F}[f](j) - \mathcal{F}[f_{\text{ini}}](2\pi j)|^2 = a_j^2 + b_j^2. \tag{6.54}$$

在数值上，我们可以通过岭回归方法（Tikhonov 正则化方法）来求解 LFP 模型 [1]。具体而言，我们可以改为计算如下无约束优化问题：

$$\min_{a_j, b_j} \sum_{i=1}^{n} \left(\sum_{j \in I} \left[a_j \sin\left(2\pi j x_i\right) + b_j \cos\left(2\pi j x_i\right) \right] - y_i \right)^2 + \varepsilon \sum_{j \in I} \gamma (2\pi j)^{-2} \left(a_j^2 + b_j^2 \right), \tag{6.55}$$

其中，$\varepsilon = 10^{-6}$。

记 $M_I = \frac{K-1}{2L}$。我们可以将上述问题改写为向量形式：

$$\min_{\boldsymbol{a}} \left(\boldsymbol{E}\boldsymbol{a} - \boldsymbol{Y} \right)^{\intercal} \left(\boldsymbol{E}\boldsymbol{a} - \boldsymbol{Y} \right) + \varepsilon \boldsymbol{a}^{\intercal} \boldsymbol{W}^{-1} \boldsymbol{a}, \tag{6.56}$$

其中

$$\boldsymbol{a} = [a_0, \cdots, a_{M_I}, b_0, \cdots, b_{M_I}]^{\intercal},$$

$$\boldsymbol{E} = \left[\sin\left(2\pi 0 \boldsymbol{X}\right), \cdots, \sin\left(2\pi M_I \boldsymbol{X}\right), \cos\left(2\pi 0 \boldsymbol{X}\right), \cdots, \cos\left(2\pi M_I \boldsymbol{X}\right) \right],$$

$$\boldsymbol{X} = [x_1, \cdots, x_n]^{\intercal}, \quad \boldsymbol{Y} = [y_1, \cdots, y_n]^{\intercal},$$

$$\boldsymbol{W}^{-1} = \operatorname{diag}\left\{ \gamma(2\pi 0)^{-2}, \cdots, \gamma(2\pi M_I)^{-2} \right\}.$$

上述问题的解应满足对 $\boldsymbol{a}$ 求导等于 0，即

$$\boldsymbol{E}^{\intercal} \left(\boldsymbol{E}\boldsymbol{a} - \boldsymbol{Y} \right) + \varepsilon \boldsymbol{W}^{-1} \boldsymbol{a} = 0. \tag{6.57}$$

求解 $\boldsymbol{a}$ 我们有，

$$\boldsymbol{a} = \left[\boldsymbol{E}^{\intercal} \boldsymbol{E} + \varepsilon \boldsymbol{W}^{-1} \right]^{-1} \boldsymbol{E}^{\intercal} \boldsymbol{Y}. \tag{6.58}$$

在数值实验中，我们使用具有 10000 隐藏神经元和不同初始化的 ReLU 神经网络。我们取三个数据点 $\{(0,1),(0.2,0),(0.4,1)\}$ 作为训练数据集，所有参数的初始化均从均值为零的均匀分布中采样。为了验证公式6.22中两个加和项的准确性，我们分别设置两组超参数 a，$\boldsymbol{w}$，b 使得 $1/\xi^4$ 和 $1/\xi^2$ 项占主导。

[1] 代码可见 `https://github.com/xuzhiqin1990/LFP`

- 为了使 $1/\xi^4$ 项占主导地位，我们设置 $\boldsymbol{w}(0) \in [-3,3]$，$a(0) \in [-0.01, 0.01]$，偏置项 $b(0) \in [-3,3]$。如图 6.7(a) 所示，神经网络通过平滑函数（红色实线）对训练数据进行插值，这与 LFP 模型的预测（蓝色虚线）和三次样条插值（灰色虚线）几乎重叠。
- 为了使 $1/\xi^2$ 项占主导地位，我们设置 $\boldsymbol{w}(0) \in [-0.1, 0.1]$，$a(0) \in [-2,2]$，偏置项 $b(0) \in [-2,2]$。如图 6.7(b) 所示，神经网络通过函数对训练数据进行插值，这与 LFP 模型的预测和线性样条插值几乎重叠。

6.6.2 高频收敛极限数值实验

假设有 n 个数据点：$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$，每个数据点是 d 维的，$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})^\intercal$。数据对应的标签记为 $(y_1, y_2, \ldots, y_n)^\intercal$。为了符号简单，我们记 $(j_1, j_2, \cdots, j_d)^\intercal$ 为 $\boldsymbol{J}_{j_1 \ldots j_d}$。优化模型可以变为

$$\min_{\phi \in \mathbb{R}^{(2M)^d}} \sum_{j_1,\ldots,j_d=-M}^{M} (1 + \|\boldsymbol{J}_{j_1\ldots j_d}\|^2 \Delta\xi^2)^{\frac{\alpha}{2}} \left|\phi_{j_1\ldots j_d}\right|^2, \tag{6.59}$$

$$\text{s.t.} \sum_{j_1,\ldots,j_d=-M}^{M} \phi_{j_1\ldots j_d} \mathrm{e}^{2\pi \mathrm{i} \Delta\xi \boldsymbol{J}_{j_1\ldots j_d}^\intercal \boldsymbol{x}_k} = y_k, \quad k = 1, 2, \ldots, d \tag{6.60}$$

这个问题同样可以通过岭回归求解。在图6.9中，对一维的例子，我们设置 $M = 1000$，对二维的例子，我们设置 $M = 100$。

Chapter 7

相图分析

在我们之前章节的讨论中，我们揭示了神经网络模型的一种有趣的现象，即在训练过程满足**频率原则**。这一原则指出，神经网络在学习过程中有一种内在的、隐式的偏好，即首先学习低频部分然后过渡到高频。因为训练并没有显式地要求频率收敛有差异，因此，这种现象可以被视为一种隐式发生的正则化形式。它从一个更宏观的函数空间视角，为我们揭示了神经网络如何逐渐把握和构建对数据的理解。这种洞察不仅加深了我们对神经网络训练过程的认识，而且可以指导我们设计出性能更好的神经网络架构。

尽管如此，频率原则仍然只能提供一个相对粗糙的理解框架。它使我们能够从一个较高的层次把握神经网络的学习倾向——即优先学习低频然后过渡到高频——但却未能为我们提供一个精确的工具，去定量化研究网络参数随着学习过程的具体演化。更进一步，仅凭这一原则，我们难以深入探究神经网络模型超越传统核方法的独特之处，或是区分不同神经网络架构之间的差异，因为它们都表现出类似的从低频到高频的学习进程。

为了更精细地描绘和理解这些差异，我们需要深入挖掘模型的细节层面。在本章节中，我们将详细探讨全连接神经网络在不同超参数配置下的表现，特别关注那些影响参数初始化的超参数。

科普篇

神经网络的相图分析是一种研究神经网络在不同初始化条件下行为表现的方法。具体来说，相图分析可以帮助我们了解神经网络在训练过程中是表现出线性行为还是非线性行为。这里所谓的线性和非线性是指模型关于参数是线性还是非线性的。实际上，模型关于参数是否线性决定了问题的复杂程度。

如果模型关于参数是线性的（例如线性回归模型、随机特征模型、核方法），那么损失函数（如均方误差，MSE）通常是一个凸函数。在这种情况下，优化问题会相对简单，因为损失函数具有一个唯一的全局极小点，没有局部极小点。然而，如果模型关于参数是非线性的，损失函数则是非凸的，从而使得优化过程变得更加复杂。

为什么我们要研究神经网络的相图分析？

- **现象驱动：**实验表明，参数初始化的尺度大小对模型最终学到的函数有显著影响。然而，神经网络的频率原则只能提供一个粗略的理解框架，难以定量分析网络参数在学习过程中的具体演化。为了更精细地描绘和理解这些差异，我们需要更深入地探讨模型在参数层面的细节。

- **理解学习机制：**区分神经网络的线性行为和非线性行为非常重要。如果神经网络的行为近似于线性模型，我们就难以解释为什么其他线性模型（如核方法）没有像神经网络那样取得巨大的成功。

神经网络的相图具体是怎么样的？跟初始化的关系是什么？

在热力学系统中，相态的描述需要识别合适的状态量，比如水的温度和压强，这些状态量能够独立变化并清晰地区分不同的相态。受此启发，我们提出一个问题：是否能为神经网络定义相似的状态量来判定其所处的状态——是线性状态还是非线性状态？更具体地说，在适当定义的状态量下，我们是否能够绘制出类似于水相图（水的三种状态：固体、液体、气体变化图）的神经网络相图？

答案是肯定的。神经网络的相图就是一个展示网络在不同初始化条件下行为表现的图表。在两层神经网络相图中，我们用两个状态量参数（γ 和 γ'）来描述初始化条件，并展示在这些初始化条件下网络的训练是线性还是非线性行为。这里状态量参数 γ 表示初始化时输出函数的幅值大小，状态量参数 γ' 表示初始化时内外层权重的大小差异。

初始化条件对神经网络的训练过程有重要影响。具体来说，初始化的大小决定了网络参数的初始值，这些初始值会影响网络在训练过程中的演化。

- **线性区域：**当初始化较大时，神经网络在初始化附近就能拟合好数据，参数的相对变化范围非常小，因此模型在初始化附近可以用一阶泰勒展开近似，网络表现出线性行为。

- **凝聚区域：**当初始化较小时，神经网络在初始化附近不能拟合好数据，网络参数的相对变化范围较大，因此网络不能用一个线性模型近似，表现出高度的非线性行为。实验中，我们发现非线性行为和神经元的“凝聚现象”（即许多神经元表现趋同）高度相关，因此我们将其称为凝聚区域，这也启发了后续关于凝聚现象的一系列研究。

- **临界区域：**介于线性和凝聚区域之间，网络表现出中等程度的非线性行为。

通过实验和理论分析，我们可以绘制出一个相图，展示不同初始化条件下网络的表现。例如，图中的一部分区域表示网络在这些条件下表现为线性行为，而另一部分区域表示网络在这些条件下表现为凝聚或者临界两种行为，这就是神经网络的相图。

神经网络的相图分析有什么用？

- **理解和优化训练过程：**相图分析可以帮助我们理解和优化神经网络的训练过程。通过相图，我们可以确定在不同初始化条件下，网络是表现为线性还是非线性，从而选择我们想要的初始化条件，提高模型的训练效率和性能。
- **启发后续研究：**相图分析可以启发一系列后续研究。例如，我们可以进一步探讨为什么在神经网络的非线性区域会出现凝聚现象？凝聚现象与特征学习之间的关系是什么？凝聚现象对理解神经网络的优化和泛化有什么影响?

7.1 神经网络在不同初始化下的表现

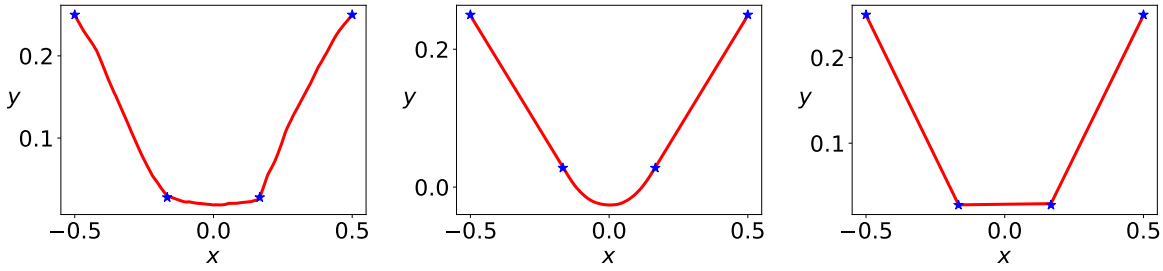
为了观察神经网络在不同初始化尺度下的表现，让我们从一个简单的实验入手。如图 7.1 所示，我们使用一个简单的两层神经网络来拟合一组简单的数据点（图 7.1中的蓝色星星点）。该网络的表达式为：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}),$$

这里神经网络隐藏层的宽度 $m = 1000$, 我们设定 a_k 和 $\boldsymbol{w}_k$ 的初始化服从服从均值为 0，不同方差的高斯分布，本文称**高斯分布方差的大小为参数的初始化大小**。

在这个实验中，我们观察到参数初始化的尺度大小对模型最终学到的函数有显著影响。从图 7.1(a) 可以看到，当参数初始化较大时，神经网络学到的函数表现出轻微的锯齿状波动。而在图 7.1(b) 中，随着参数初始化的减小，神经网络学习到的函数逐渐变得更加平滑，类似于三次样条函数。最终，在图 7.1(c) 中，我们观察到当参数初始化进一步减小时，模型倾向于学习到一个几乎是线性插值的函数，这反映了参数初始化大小对学习结果的微妙而关键的作用。

图 7.1所示的实验虽然简单，但是却提示我们神经网络在不同超参数配置下的表现可能会不同，特别是那些影响参数初始化的超参数。本章的后面内容我们将分析这些超参数如何影响网络在训练过程中的表现以及最终的学习结果。在 7.2节中，我们将介绍神经网络的线性与非线性行为，并展示非线性行为的一个重要现象——凝聚。在 7.3节中，我们定义划分线性区域与非线性区域的状态量，并得到实验上的相图。在 7.4节中，我们将仔细分析两层 ReLU 神经网络理论上相图的获取方法，并将两层 ReLU 神经网络的相图推广至三层。



(a) 初始化大小: $\frac{1}{\sqrt{m}}$　(b) 初始化大小: $\frac{1}{m}$　(c) 初始化大小: $\frac{1}{m^{1.75}}$

图 7.1: 不同初始化大小的两层 ReLU 神经网络（宽度 $m = 1000$）学习四个数据点的结果图

7.2 神经网络的线性与非线性行为

7.2.1 参数的演化分析

之前章节提到的频率原则着重于刻画函数空间的演化，但是神经网络训练的本质却是在参数空间中的演化。因此，为了更细致地理解神经网络的内在工作机制，我们将聚焦于参数的演化过程。考虑一个具有两层结构和 ReLU 激活函数的神经网络，其数学表达式可以写为：

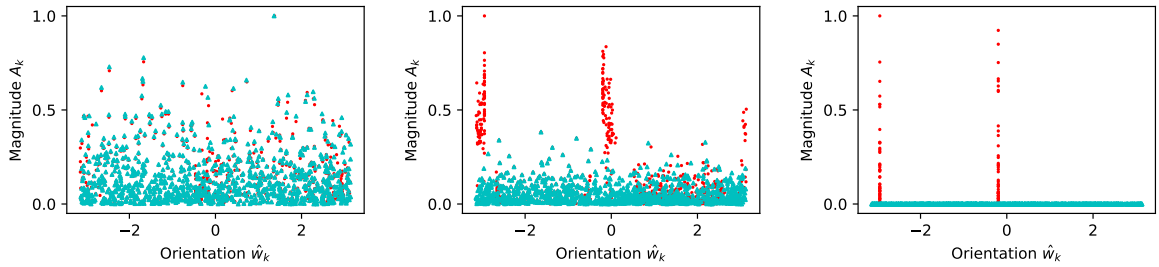
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \mathrm{ReLU}(\boldsymbol{w}_k^\top \boldsymbol{x}), \tag{7.1}$$

其中 $\boldsymbol{x} \in \mathbb{R}^d$ 代表模型的输入向量。网络的参数集合 $\boldsymbol{\theta}$ 可通过向量化操作 $\mathrm{vec}(\cdot)$ 表示为 $\boldsymbol{\theta} = \mathrm{vec}(\boldsymbol{\theta}_a, \boldsymbol{\theta}_{\boldsymbol{w}})$，这里 $\boldsymbol{\theta}_a = \mathrm{vec}(\{a_k\}_{k=1}^m)$ 和 $\boldsymbol{\theta}_{\boldsymbol{w}} = \mathrm{vec}(\{\boldsymbol{w}_k\}_{k=1}^m)$ 分别代表输出层权重和输入层权重，它们的初始值 $(a_k^0, \boldsymbol{w}_k^0)$ 通过高斯分布 $a_k^0 \sim \mathcal{N}(0, \beta_1^2)$ 和 $\boldsymbol{w}_k^0 \sim \mathcal{N}(0, \beta_2^2 \mathbf{I}_d)$ 进行初始化。在式 7.1 中，可以通过将输入 $\boldsymbol{x}$ 和权重 $\boldsymbol{w}_k$ 扩展为 $(\boldsymbol{x}^\top, 1)^\top$ 和 $(\boldsymbol{w}_k^\top, b_k)^\top$ 来使偏置项 b_k 整合进模型。

为了探索图 7.1 中不同初始化条件下网络参数在空间中的演化细节，我们注意到 ReLU 激活函数具有正齐次性，也即 $\sigma(ax) = a\sigma(x), \forall a > 0$，因此我们可以将每个神经元的参数 $(a_k, \boldsymbol{w}_k)$ 分解为单位方向向量 $\hat{\boldsymbol{w}}_k = \boldsymbol{w}_k / \|\boldsymbol{w}_k\|_2$ 和一个表示其对输出贡献的振幅 $A_k = |a_k| \|\boldsymbol{w}_k\|_2$，即 $(\hat{\boldsymbol{w}}_k, A_k)$。

在图 7.1 所示的实验中，我们用不同初始化大小（均值为 0，不同方差的高斯分布）的两层 ReLU 神经网络（宽度 $m = 1000$）学习四个数据点。这里的输入是一维的，且模型包含偏置项，所以权重向量 $\boldsymbol{w}_k := (w_k, b_k)^\top$ 实际上是二维的。因此，我们可以通过每个 $\hat{\boldsymbol{w}}$ 相对于 x 轴的 $[-\pi, \pi)$ 范围内的角度来表示其方向。基于 $(\hat{\boldsymbol{w}}_k, A_k\}_{k=1}^m$ 这一表示方法，我们展示了不同初始化条件下模型最终学到的参数（红色圆点）与模型初始参数（青色三角）的对比，如图 7.2所

示。每张图根据图内所有点的最大幅度做归一化，使得最大幅度为 1。例如图 7.2(c) 中，末态的幅度远大于初态，所以归一化后，初态的幅度几乎靠近 0。



(a) 初始化大小: $\frac{1}{\sqrt{m}}$　(b) 初始化大小: $\frac{1}{m}$　(c) 初始化大小: $\frac{1}{m^{1.75}}$

图 7.2: 不同初始化大小的神经网络学习四个数据点对应的网络初始态 (青色三角) 和末态 (红色圆点) 的散点图

从图 7.2 中，我们可以观察到两个显著的现象：

(i) 当初始化较大时，网络学习到的参数与初始参数相差无几（图 7.2(a) 中青色三角与红色圆点点几乎重合），而随着初始化的减小，网络学习到的参数与初始参数的差异逐渐增大（图 7.2(b) 和图 7.2(c) 中青色三角与红色圆点的分布有明显的区分）。

(ii) 在较小的初始参数设置下，那些振幅显著的活跃神经元倾向于向特定方向集中，这种现象在图 7.2(b) 中表现为中等程度的“凝聚”，并且随着初始化参数的进一步减小，图 7.2(c) 展示了强烈的“凝聚”行为。这里图 7.2(c) 中两个凝聚的方向刚好对应图 7.1(c) 中两个 ReLU 组合形成的输出函数。

这些观察结果为我们揭示了参数初始化大小对神经网络学习行为非常重要，我们将在下面的部分仔细研究这些行为。

7.2.2 线性行为与非线性行为的界定

在探讨神经网络的动力学行为时，理解网络的线性和非线性行为至关重要。尽管神经网络模型 $f_{\boldsymbol{\theta}}$ 关于其参数 $\boldsymbol{\theta}$ 是非线性的，但在某些情况下，当参数 $\boldsymbol{\theta}$ 的变化量极小，我们可以用一个线性模型来近似这种非线性的模型。具体地，对于任意训练时刻 t，如果参数 $\boldsymbol{\theta}$ 的变化量足够小，那么神经网络模型 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ 可以被以下线性模型 $f_{\boldsymbol{\theta}}^{\text{lin}}(\boldsymbol{x})$ 很好地近似：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx f_{\boldsymbol{\theta}}^{\text{lin}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}(0)}(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}(0)}(\boldsymbol{x}) \cdot (\boldsymbol{\theta}(t) - \boldsymbol{\theta}(0)). \tag{7.2}$$

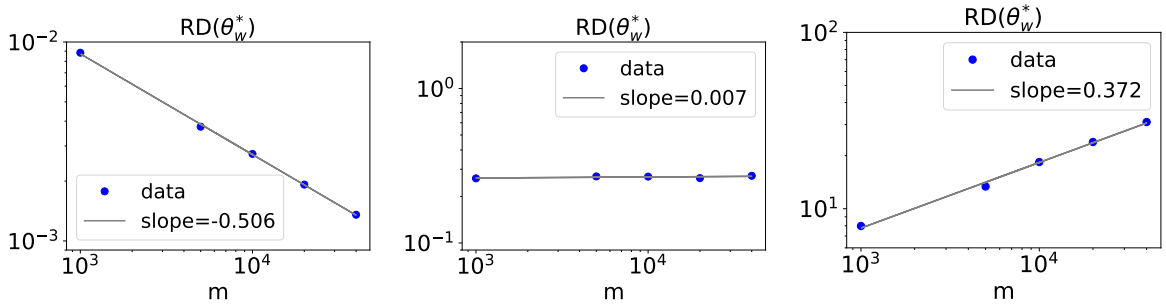
这种线性近似是基于一阶泰勒展开，通常只在 $\boldsymbol{\theta}(t)$ 保持在 $\boldsymbol{\theta}(0)$ 的一个小邻域内时有效。对于两层神经网络，由于模型 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k \cdot \boldsymbol{x})$ 对于输出层参数 a_k 总是线性的，这个小邻域的要求主要适用于输入权重 $\boldsymbol{\theta}_{\boldsymbol{w}}(t)$，即 $\boldsymbol{\theta}_{\boldsymbol{w}}(t)$ 需要保持在 $\boldsymbol{\theta}_{\boldsymbol{w}}(0)$ 的一个小邻域内。因此，我们将这个线性近似有效的区域称为 **“线性区域”**。

是否可以用线性模型准确地逼近一个神经网络，取决于输入权重参数 $\boldsymbol{\theta}_{\boldsymbol{w}}(t)$ 在训练后的变化程度。为了量化这种变化，我们可以定义**相对距离** (Relative Distance, RD) 来度量 $\boldsymbol{\theta}_{\boldsymbol{w}}(t)$ 在训练期间偏离其初始值 $\boldsymbol{\theta}_{\boldsymbol{w}}(0)$ 的程度：

$$\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}(t)) = \frac{\|\boldsymbol{\theta}_{\boldsymbol{w}}(t) - \boldsymbol{\theta}_{\boldsymbol{w}}(0)\|_2}{\|\boldsymbol{\theta}_{\boldsymbol{w}}(0)\|_2}. \tag{7.3}$$

具体来说，我们关注 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$，它代表了在整个训练过程中 $\boldsymbol{\theta}_{\boldsymbol{w}}(t)$ 从初始值到最终学习状态的相对偏移距离，其中 $\boldsymbol{\theta}^*_{\boldsymbol{w}} := \boldsymbol{\theta}_{\boldsymbol{w}}(\infty)$ 代表训练结束时的参数值。

使用这个指标，我们可以定量地描述图 7.2 实验中观察到的参数相对变化距离。在实践中我们经常使用不同宽度的网络来学习数据，而这些网络的初始化大小通常与网络宽度成幂律比例关系。因此对于图 7.2的数据，我们绘制了在不同初始化大小下 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$ 与网络宽度 m 之间的关系，如图 7.3 所示。随着网络宽度 $m \to \infty$, 相对距离 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$ 与宽度 m 的关系呈现不同的趋势。对于图 7.3(a-c)，五个蓝点分别表示隐层神经元数为 $1000, 5000, 10000, 20000, 40000$ 的神经网络上的 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$ 的计算结果。灰线是数据点的线性拟合并在标签中给出斜率标注。



(a) 初始化大小: $\frac{1}{\sqrt{m}}$　(b) 初始化大小: $\frac{1}{m}$　(c) 初始化大小: $\frac{1}{m^{1.75}}$

图 7.3: 随着网络宽度 $m \to \infty$, 相对距离 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$ 与宽度 m 的关系图

从图 7.3 中，我们可以明显看出，一旦初始化尺度确定，相对距离 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$ 与网络宽度 m 之间近似遵循一种幂律关系。随着 m 趋向无穷大，若 $\mathrm{RD}(\boldsymbol{\theta}^*_{\boldsymbol{w}})$ 趋于零，这表明 $\boldsymbol{\theta}$ 的变化很小，基于前面的泰勒一阶线性展开，神经网络的行为可以被一个线性模型很好地近似，此时，网络的训练动力学主要表现为线性行为，我们将这个区域成为 **“线性区域”（linear regime）**。相

反，如果这个相对距离 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 接近常数 $O(1)$ 或趋向于无穷大，网络的训练动力学则显现出显著的非线性行为。

特别地，当 $\boldsymbol{\theta}_{\boldsymbol{w}}$ 与其初始值偏离非常远，即 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 趋向于无穷大时，这对应高度非线性的动力学，而这时我们在参数空间中观察到了一种显著的凝聚行为（如图 7.2(c) 所示）。这种现象表明了参数在训练过程中向特定方向的集中趋势，因此我们将这个区域称为 **“凝聚区域”（condensed regime）**。而对于 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 趋于常数 $O(1)$ 的情况，神经网络表现出中等程度的非线性，这个区域我们称之为 **“临界区域”（critical regime）**。

总的来说，通过对参数初始化大小的分析，我们发现网络在相对较大的初始化下表现出线性行为，而在较小的初始化下则表现出高度的非线性行为。非线性行为的一个显著特征是**神经元的凝聚**，这与网络的**特征学习**能力紧密相关，即模型能够从数据中学习到某些特定的特征——那些参数凝聚的方向。这种特征学习被普遍认为是神经网络取得显著成功的关键因素。如果神经网络的行为近似于一个线性模型，我们就难以解释为什么其他的线性模型，例如核方法，没有能够像神经网络那样取得巨大的成功。因此，神经网络的能力来自于其内在的**非线性**特性，这是其区别于传统线性模型的关键所在。

7.3 线性区域与非线性区域的划分

在之前的实验中，我们观察到神经网络在不同的初始化条件下可能呈现线性或非线性的行为。为了更精确地区分线性行为和非线性行为，本节将尝试对线性区域和非线性区域进行划分。

7.3.1 状态量的定义与动力学相变

我们发现网络在相对较大的初始化下表现出线性行为，而在相对较小的初始化下表现出非线性行为，这类似于物理学中的相变。在统计力学中，为了观察到相变现象，研究者通常会将粒子数目扩展到无限。在图 7.3 所示的实验中，随着网络宽度 m 的增加，我们观察到 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 可能趋向于 $0, O(1), +\infty$。因此为了使这一分界更加明确，我们同样可以考虑当 $m \to \infty$ 时的极限情况，即**无限宽神经网络的动力学相变**。

在热力学系统中，相态的描述需要识别合适的状态量，比如水的温度和压强，这些状态量能够独立变化并清晰地区分不同的相态。受此启发，我们提出一个问题：是否能为神经网络定义相似的状态量来判定其所处的状态——是线性状态还是非线性状态？更具体地说，在适当定义的状态量下，我们是否能够绘制出类似于水相图 (水的三种状态: 固体，液体，气体的变化图) 的神经网络相图？

我们还是以两层 ReLU 神经网络作为研究对象。考虑给定的数据集 $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$，其

中 $\boldsymbol{x}_i \in \mathbb{R}^d$，$i \in [n]$，网络宽度为 m，激活函数 $\sigma = \mathrm{ReLU}$。这里我们引入一个缩放参数 $1/\alpha$，用于控制求和后的整体输出大小，这使得我们能够统一分析两层无限宽神经网络的常见初始化方法：

$$f_{\boldsymbol{\theta}}^{\alpha}(\boldsymbol{x}) = \frac{1}{\alpha}\sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^{\top}\boldsymbol{x}), \tag{7.4}$$

这里 a_k 和 $\boldsymbol{w}_k$ 分别按照不同的尺度 β_1 和 β_2 初始化：

$$a_k^0 \sim N(0, \beta_1^2), \quad \boldsymbol{w}_k^0 \sim N(0, \beta_2^2 \boldsymbol{I}_d). \tag{7.5}$$

考虑如下的均方误差损失函数：

$$R_S(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}^{\alpha}(\boldsymbol{x}_i) - y_i\right)^2. \tag{7.6}$$

基于梯度下降的优化策略，在连续时间极限下，神经网络参数 $\boldsymbol{\theta}$ 遵循如下的梯度流动力学方程：

$$\frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = -\nabla_{\boldsymbol{\theta}} R_S(\boldsymbol{\theta}). \tag{7.7}$$

这一方程描述了参数随时间变化的轨迹，指向损失函数下降最快的方向。具体到我们的两层 ReLU 神经网络模型，参数 $\boldsymbol{\theta}$ 可以被表示为 $\boldsymbol{\theta} = \mathrm{vec}(\{\boldsymbol{q}_k\}_{k=1}^m)$，其中 $\boldsymbol{q}_k = (a_k, \boldsymbol{w}_k^{\top})^{\top}$，$k \in [m]$。这样，参数的时间演化由下列方程描述：

$$\begin{aligned}
\frac{\mathrm{d}a_k}{\mathrm{d}t} &= -\frac{1}{n}\sum_{i=1}^{n} \frac{1}{\alpha}\sigma(\boldsymbol{w}_k^{\top}\boldsymbol{x}_i)\left(\frac{1}{\alpha}\sum_{k'=1}^{m} a_{k'}\sigma(\boldsymbol{w}_{k'}^{\top}\boldsymbol{x}_i) - y_i\right) \\
\frac{\mathrm{d}\boldsymbol{w}_k}{\mathrm{d}t} &= -\frac{1}{n}\sum_{i=1}^{n} \frac{1}{\alpha}a_k\sigma'(\boldsymbol{w}_k^{\top}\boldsymbol{x}_i)\boldsymbol{x}_i\left(\frac{1}{\alpha}\sum_{k'=1}^{m} a_{k'}\sigma(\boldsymbol{w}_{k'}^{\top}\boldsymbol{x}_i) - y_i\right).
\end{aligned}$$

为了消除时间以及参数大小等尺度的影响，我们执行了以下的尺度放缩：

$$\bar{a}_k = \beta_1^{-1} a_k, \quad \bar{\boldsymbol{w}}_k = \beta_2^{-1}\boldsymbol{w}_k, \quad \bar{t} = \frac{1}{\beta_1\beta_2}t, \tag{7.8}$$

这种尺度缩放后的模型允许我们在不同的初始化大小和训练动力学之间建立一种规范化的比较基准。从而，我们得到了归一化的动力学方程：

$$\begin{aligned}
\frac{\mathrm{d}\bar{a}_k}{\mathrm{d}\bar{t}} &= -\frac{\beta_2}{\beta_1}\frac{1}{n}\sum_{i=1}^{n} \frac{\beta_1\beta_2}{\alpha}\sigma(\bar{\boldsymbol{w}}_k^{\top}\boldsymbol{x}_i)\left(\frac{\beta_1\beta_2}{\alpha}\sum_{k'=1}^{m} \bar{a}_{k'}\sigma(\bar{\boldsymbol{w}}_{k'}^{\top}\boldsymbol{x}_i) - y_i\right), \\
\frac{\mathrm{d}\bar{\boldsymbol{w}}_k}{\mathrm{d}\bar{t}} &= -\frac{\beta_1}{\beta_2}\frac{1}{n}\sum_{i=1}^{n} \frac{\beta_1\beta_2}{\alpha}\bar{a}_k\sigma'(\bar{\boldsymbol{w}}_k^{\top}\boldsymbol{x}_i)\boldsymbol{x}_i\left(\frac{\beta_1\beta_2}{\alpha}\sum_{k'=1}^{m} \bar{a}_{k'}\sigma(\bar{\boldsymbol{w}}_{k'}^{\top}\boldsymbol{x}_i) - y_i\right).
\end{aligned}$$

在这里，我们注意到只有两个独立的量与梯度流动力学的时间尺度无关，这促使我们引入两个缩放参数：

$$\kappa := \frac{\beta_1\beta_2}{\alpha}, \quad \kappa' := \frac{\beta_1}{\beta_2}, \tag{7.9}$$

其中 κ 和 κ' 分别称为能量缩放参数和动力学缩放参数。κ 可以理解为平均意义下每个神经元初始化时的输出幅度，κ' 则体现 a_k 和 $\boldsymbol{w}_k$ 之间的幅度差异，或者从它们的演化动力学来看，κ' 体现它们的演化速度差异。最终，归一化的动力学方程可以被简洁地表达为：

$$\frac{\mathrm{d}\bar{a}_k}{\mathrm{d}\bar{t}} = -\frac{1}{\kappa'}\frac{1}{n}\sum_{i=1}^{n}\kappa\sigma(\bar{\boldsymbol{w}}_k^\top \boldsymbol{x}_i)\left(\kappa\sum_{k'=1}^{m}\bar{a}_{k'}\sigma(\bar{\boldsymbol{w}}_{k'}^\top \boldsymbol{x}_i) - y_i\right), \tag{7.10}$$

$$\frac{\mathrm{d}\bar{\boldsymbol{w}}_k}{\mathrm{d}\bar{t}} = -\kappa'\frac{1}{n}\sum_{i=1}^{n}\kappa\bar{a}_k\sigma'(\bar{\boldsymbol{w}}_k^\top \boldsymbol{x}_i)\boldsymbol{x}_i\left(\kappa\sum_{k'=1}^{m}\bar{a}_{k'}\sigma(\bar{\boldsymbol{w}}_{k'}^\top \boldsymbol{x}_i) - y_i\right). \tag{7.11}$$

在后续的讨论中，我们将专注于经过无量纲化处理的模型，该模型由方程 (7.10), (7.11)定义，并为了简洁起见，我们将省略 $\bar{a}_k$、$\bar{\boldsymbol{w}}_k$、$\bar{t}$ 中的 “$\bar{\cdot}$”。

当网络宽度 m 趋向于无限大时，常见的初始化尺度大小都与网络宽度 m 存在幂律关系。为了捕捉这种关系，我们定义 κ 和 κ' 与 m 之间的对数对数（$\log-\log$）关系如下：

$$\gamma = \lim_{m\to\infty} -\frac{\log\kappa}{\log m}, \quad \gamma' = \lim_{m\to\infty} -\frac{\log\kappa'}{\log m}. \tag{7.12}$$

这两个参数 γ 和 γ' 是与时间尺度无关的独立参量，也是下文我们划分线性区域和非线性区域的状态量。不同的初始化方法会导致这些参数取不同的值，从而影响模型的训练动力学和最终学到的结果。

为了展示这一点，我们列出了一些在文献中常见的初始化方法及其相关工作，以及它们对应的缩放参数。这些信息被总结在表 7.1中。

7.3.2 实验上相图的获取

有了划分线性区域和非线性区域的状态量之后，我们可以设计实验来获取不同相态之间转变的相图，特别是用以区分线性区域和凝聚区域的界限。实验的核心是估计网络权重更新的相对距离 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$。

虽然理想情况下我们希望在无限大的模型宽度 m 下进行实验，但这在实际操作中是不可行的。因此，我们转而量化 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 随着模型宽度 m 增加而发生的变化。如图 7.3的子图 (a) 到 (c) 所示，这种变化近似遵循一个幂律关系。基于此，我们定义权重更新的相对差异的**标度指数** $S_{\boldsymbol{w}}$ 如下：

$$S_{\boldsymbol{w}} = \lim_{m\to\infty}\frac{\log\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)}{\log m}, \tag{7.13}$$

超参数 (相关工作)	α	β_1	β_2	κ $\left(\frac{\beta_1\beta_2}{\alpha}\right)$	κ' $\left(\frac{\beta_1}{\beta_2}\right)$	γ $\left(\lim_{m\to\infty}\frac{\log 1/\kappa}{\log m}\right)$	γ' $\left(\lim_{m\to\infty}\frac{\log 1/\kappa'}{\log m}\right)$
LeCun (LeCun et al. 2012)	1	$\sqrt{\frac{1}{m}}$	$\sqrt{\frac{1}{d}}$	$\sqrt{\frac{1}{md}}$	$\sqrt{\frac{d}{m}}$	$\frac{1}{2}$	$\frac{1}{2}$
He (He et al. 2015)	1	$\sqrt{\frac{2}{m}}$	$\sqrt{\frac{2}{d}}$	$\sqrt{\frac{4}{md}}$	$\sqrt{\frac{d}{m}}$	$\frac{1}{2}$	$\frac{1}{2}$
Xavier (Glorot & Bengio 2010)	1	$\sqrt{\frac{2}{m+1}}$	$\sqrt{\frac{2}{m+d}}$	$\sqrt{\frac{4}{(m+1)(m+d)}}$	$\sqrt{\frac{m+d}{m+1}}$	1	0
NTK (Jacot et al. 2018)	$\sqrt{m}$	1	1	$\sqrt{\frac{1}{m}}$	1	$\frac{1}{2}$	0
Mean-field (Mei et al. 2018) (Sirignano & Spiliopoulos 2020) (Rotskoff & Vanden-Eijnden 2018)	m	1	1	$\frac{1}{m}$	1	1	0
E et al. (E et al. 2020)	1	β	1	β	β	$\lim_{m\to\infty}\frac{\log 1/\beta}{\log m}$	$\lim_{m\to\infty}\frac{\log 1/\beta}{\log m}$

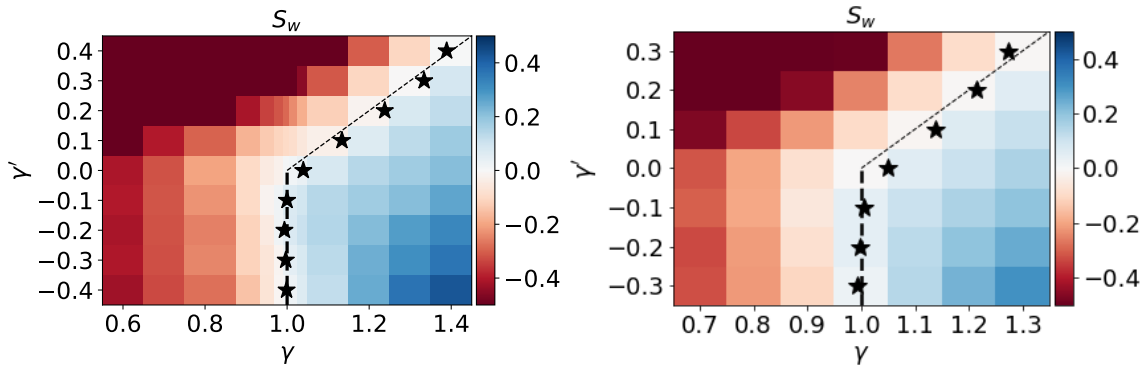
表 7.1: 常见的初始化方法及其缩放参数

该指数可以通过估计图 7.3中的对数坐标图的斜率来获得, 从而实证地确定 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 随着 $m \to \infty$ 的变化趋势。如果斜率 $S_{\boldsymbol{w}} < 0$, 则表明 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 随着 m 的增大趋于零, 对应线性区域; 反之, 如果斜率 $S_{\boldsymbol{w}} > 0$, 则意味着 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 随着 m 的增大而趋于无限大, 对应非线性区域。

接下来, 我们实验上可以通过在相空间中扫描 $S_{\boldsymbol{w}}$ 来经验地获得相图。例如对于图7.1的一维拟合问题, 我们在图 7.4(a) 中展示了其实验上获得的相图, 其中 $S_{\boldsymbol{w}}$ 的估计是在具有 1000, 5000, 10000, 20000 和 40000 个隐层单元的两层 ReLU 神经网络上进行的。在图 7.4(a) 中, 红色区域代表 $S_{\boldsymbol{w}}$ 值小于零, 这表明 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 随 $m \to \infty$ 趋向于零, 刻画了线性区域。相反, 在蓝色区域, $S_{\boldsymbol{w}}$ 值大于零, $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 随 $m \to \infty$ 增长至无穷大, 这表明了高度非线性的区域。

这两个区域的分界线可以通过插值方法来近似确定, 我们在图 7.4(a) 中以星号标出, 这是 $\mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}^*)$ 的量级约为 $O(1)$ 的区域。这样的插值不仅为我们提供了一个直观的区域划分, 而且为进一步的理论分析和实验设计提供了一个明确的基准。

为了验证在图 7.4(a) 中得到的相图是否也适用于真实数据集, 我们采用了一个两层 ReLU 激活函数的神经网络对 MNIST 数据集进行了训练, 其中损失函数为均方误差 (Mean Squared Error, MSE)。$S_{\boldsymbol{w}}$ 的估计是在具有 1000, 10000, 50000, 250000 和 400000 个隐层单元的两层 ReLU 神经网络上进行的。图中的星号标记了在固定 γ' 的条件下, 通过线性插值得到的零交叉点。虚线代表通过理论分析确定的分界线 (参见后续的理论分析 7.4节)。在此实验设置中, 输入是一个 784 维的向量, 代表了图像的像素值, 输出则是一个一维值, 用以表示图像的标签



(a) 一维数据　　(b) MNIST 数据

图 7.4: 一维数据和 MNIST 数据上得到的实验上的相图

(其值从 0 到 9)。如图 7.4(b) 所展示的，我们观察到从一维数据中获得的相图与从实际的高维 MNIST 数据集中得到的结果具有高度的一致性。

通过这些实验，我们成功地划分了线性区域和非线性区域，并据此构建了一个经验性的相图。在后面的内容中，我们将展示实际训练过程中神经网络在临界和凝聚区域的行为以及如何利用理论分析来精确刻画线性区域与非线性区域之间的界限。

7.3.3 临界和凝聚区域

在前一节通过图 7.2的分析中，我们注意到了神经网络在参数空间中的表示 $\{(\hat{\boldsymbol{w}}_k, A_k)\}_{k=1}^m$ 的凝聚现象，这是神经网络非线性训练动力学的一个突出特点。在有了实验上相图的获取后，我们也可以关注在相图中随网络宽度 m 趋向于无穷大时的凝聚行为，即当 $\boldsymbol{\theta}_w$ 的相对变化量趋于无穷大时。我们采用与图 7.2中同样的一维数据集，观察 $(\hat{\boldsymbol{w}}_k, A_k)$ 的分布模式。通过分析三种不同宽度的双层网络 $m = 10^3, 10^4, 10^6$，我们探索了 $m \to +\infty$ 的趋势。结果如图 7.5所示，每个子图中的颜色代表对应 γ 和 γ' 的神经网络上的 $\{(\hat{\boldsymbol{w}}_k, A_k)\}_{k=1}^m$ 的散点分布。隐层神经元数量分别为 $m = 10^3$（蓝色）、10^4（红色）、10^6（黄色）。横坐标表示 γ，纵坐标表示 γ'。可以清晰地看到，如蓝色框所示的边界右侧，在非线性区域，随着 $m \to \infty$，凝聚态的形成越发显著。这与我们的预期相符，即 $\boldsymbol{\theta}_{\boldsymbol{w}}$ 偏离初始状态越远，网络的非线性特性也就越强。如前文所述，我们将这个区域定义为凝聚区域。在线性区域与凝聚区域之间的临界区域，随着 $m \to \infty$，凝聚的程度似乎保持不变，这表现出了类似平均场理论的行为。目前而言，凝聚机制及其隐式的正则化效应尚未被完全理解，这仍是未来研究的一个重要课题。

进一步，我们也研究了在 MNIST 数据集上的神经网络凝聚行为。由于这是一个高维数据

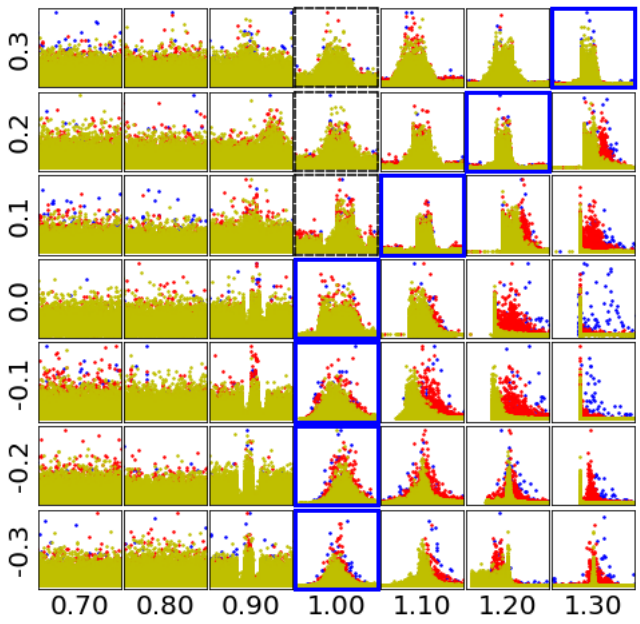


图 7.5: 一维合成数据上的凝聚态分布图

集，无法直接可视化高维特征空间中的分布，因此我们采用了一种投影方法：将每个 $\hat{\boldsymbol{w}}$ 投影到一个参考方向 $\boldsymbol{p}$ 上，并绘制 A_k 与 $I_{\hat{\boldsymbol{w}}} = \hat{\boldsymbol{w}} \cdot \boldsymbol{p}$ 之间的关系图。值得注意的是，参考方向的选择是任意的，不会影响我们的结论。不失一般性，我们选择了方向 $\boldsymbol{p} = \mathbf{1}_n/\sqrt{n}$，其中 $\mathbf{1}_n$ 代表分量全为 1 的 n 维向量。如果神经元确实在高维特征空间中的某些方向上发生凝聚，那么它们的一维投影也应当在某些点上表现出凝聚。如图 7.6所示，每个框中的每个颜色表示对应 γ 和 γ' 的神经网络上的 $\{(I_{\hat{\boldsymbol{w}}}, A_k)\}_{k=1}^m$ 的散点分布。隐层神经元数量为:$m = 10^3$(蓝色), 10^4(红色), 2.5×10^5(黄色)。横坐标为 γ, 纵坐标为 γ'。与一维情况相似，我们可以观察到在凝聚区域中存在凝聚现象。随着网络宽度增加，凝聚现象变得更加明显。

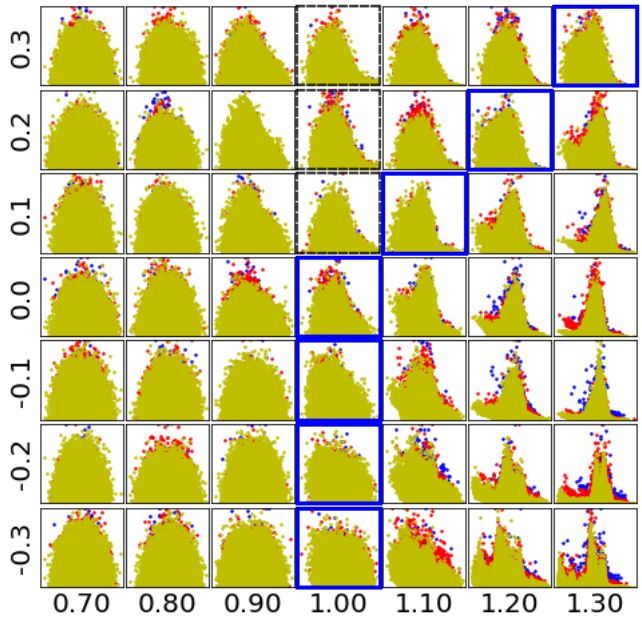


图 7.6: MNIST 数据集的凝聚态分布图

7.4 相图的理论分析

7.4.1 尺度分析方法*

在我们深入到严格的理论证明之前，初步的尺度分析其实已经能够让我们对线性区域和凝聚区域进行有效的区分。图 7.7 展示了通过严格的理论分析得到的两层无限宽 ReLU 神经网络的相图，图中用符号标出的例子可以在表格 (7.1) 找到对应。下面我们将看到它和我们进行直观的尺度分析获得的区分边界完全一致。

首先，我们来看两层 ReLU 神经网络的公式：

$$f_{\boldsymbol{\theta}}^{\alpha}(\boldsymbol{x}) = \frac{1}{\alpha}\sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^{\top}\boldsymbol{x}),$$

这是一个由 m 个神经元构成的模型，其中每个神经元参数包括一个输出权重 a_k 和一个输入权重向量 $\boldsymbol{w}_k$。

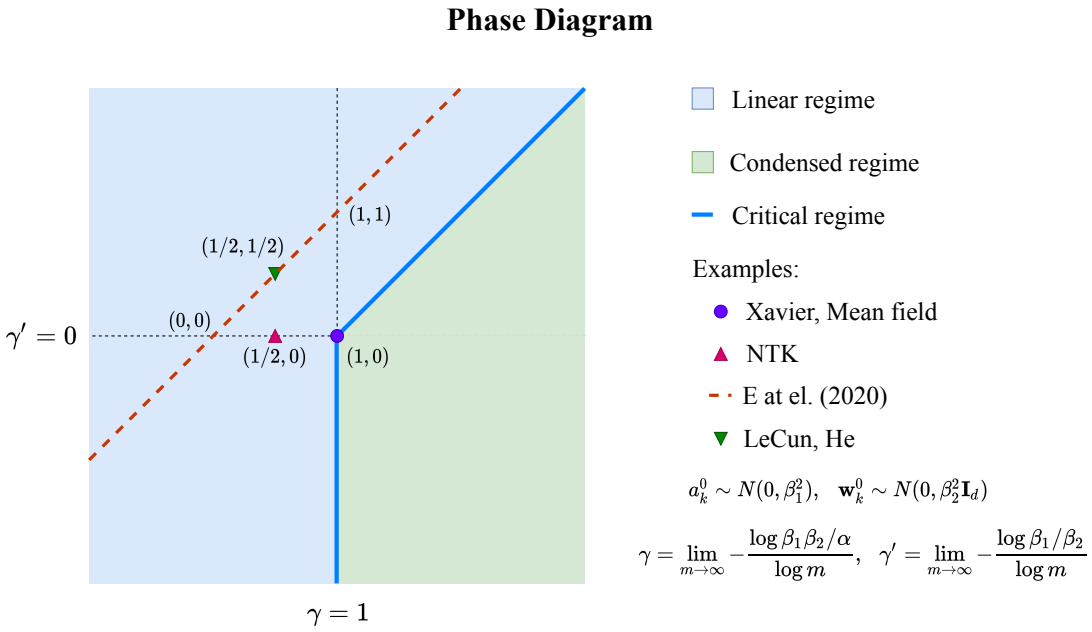


图 7.7: 两层无穷宽 ReLU 神经网络的相图

接着，我们对 a_k 和 $\boldsymbol{w}_k$ 进行初始化：

$$a_k^0 \sim N(0, \beta_1^2), \quad \boldsymbol{w}_k^0 \sim N(0, \beta_2^2 \boldsymbol{I}_d).$$

这里，a_k^0 和 $\boldsymbol{w}_k^0$ 分别是初始权重和向量参数，它们都是随机选择的，其中 $N(0, \beta_1^2)$ 和 $N(0, \beta_2^2 \boldsymbol{I}_d)$ 表示这些参数遵循均值为 0，方差分别为 β_1^2 和 β_2^2 的高斯（正态）分布。初始化的尺度 β_1 和 β_2 决定了参数的分布宽度，即初始化大小。

然后，我们考察这个两层 ReLU 神经网络在初始化时的拟合能力。由于 ReLU 函数具有齐次性，我们可以估计模型能表达的函数 $f_{\boldsymbol{\theta}}^{\alpha}(\boldsymbol{x})$ 大致的量级（即输出值的大小范围）为：

$$C = \frac{m\beta_1\beta_2}{\alpha} = m\kappa.$$

这里，$\kappa = \frac{\beta_1\beta_2}{\alpha}$ 就是我们上文定义的能量缩放因子。

我们希望比较模型输出的量级（C）与目标函数输出的量级（设为 $O(1)$，即大致为常数级别）。如果 C 大致等于或超过 $O(1)$，这意味着网络有足够的能力在初始化时就比较好地拟合目标函数。而神经网络处于线性区域的一种情况是其具有在初始化附近拟合目标函数的能力，于是我们可以得到：

$$\kappa \gtrsim 1/m,$$

回忆在式 (7.12)中，$\gamma = \lim_{m\to\infty} -\frac{\log\kappa}{\log m}$，这意味着当 $m \to \infty$ 时，我们有 $\gamma \leq 1$，这对应图 7.7中左半边区域。

此外，我们还注意到，模型相对于输出层参数 $\boldsymbol{\theta}_a$ 始终是线性的。因此，即使输出权重 $\boldsymbol{\theta}_a$ 发生显著变化，只要输入层权重 $\boldsymbol{\theta}_{\boldsymbol{w}}$ 保持在初始化附近，动力学可能仍可被线性化。因此，还存在下面这种情况，即如果以下两个条件满足，模型可能仍处于线性区域:

(i) 初始化时 $\kappa' \ll 1$（即 $\beta_1 \ll \beta_2$），a 的值较小而 $\boldsymbol{w}$ 的值较大，此时 $\gamma' > 0$，在训练过程中 a 的变化可能占主导，模型有可能保持线性。

(ii) 如果 a 的量级——例如，用期望值 $\mathbb{E}(|a|)$ 表示——在整个训练过程中满足 $\mathbb{E}(|a|) \ll \beta_2$，这种情况是可能的。此时训练结束时，模型能表达的函数 $f^{\alpha}_{\boldsymbol{\theta}}(\boldsymbol{x})$ 的量级估计为

$$C = \frac{m\beta_2\mathbb{E}(|a|)}{\alpha} \ll \frac{m\beta_2^2}{\alpha} = \frac{m\kappa}{\kappa'}. \tag{7.14}$$

为了使模型处于线性区域，我们需要 $C \gtrsim O(1)$，从而得出

$$\frac{1}{\kappa'} \gg \frac{1}{m\kappa}, \tag{7.15}$$

这在 $m \to \infty$ 时导致条件 $\gamma' > \gamma - 1(\gamma' > 0)$。这对应图 7.7右上角的区域。

综上所述，直观的尺度分析揭示了神经网络处于线性区域的必要条件，即 $\gamma \leq 1$ 或者 $\gamma' > 0$ 且 $\gamma' > \gamma - 1$。相反，如果 $\gamma' < \gamma - 1$ 且 $\gamma > 1$，即当 $m \to \infty$ 时，$m\kappa \ll 1$ 且 $m\kappa/\kappa' \ll 1$，则神经网络在 $\boldsymbol{\theta}_{\boldsymbol{w}}$ 保持在初始化附近时，无法拟合 $O(1)$ 量级的目标函数。此时，神经网络需要经历显著的参数演化才能拟合数据，这是非线性区域的特征。这种尺度分析方法得到的结果与我们下文通过严格的理论分析得到的相图（图 7.7）中的分界线完全一致。

我们注意到，在图 7.7中，实践中常用的初始化策略，如 Xavier 初始化和 Kaiming He 初始化，在无限宽度网络的相图中都位于线性或临界区域，而非我们所关注的非线性区域。然而，这并不意味着这些初始化方法在有限宽度网络中的表现也处于线性区域。如图 7.3(a) 所示，当网络宽度 m 增加至无限大时，在线性区域初始化策略下，网络参数表现出的相对距离趋于零，这使得我们将这种行为归类为线性行为。但值得强调的是，只要网络宽度是有限宽，这些线性初始化策略对应的参数变化相对距离依然是显著非零的，这表明即使在无限宽度下表现为线性的初始化策略，在有限宽度下网络也会展现出非线性的学习特性。此外，在实践中我们使用的正则化技术，例如后面章节中要讨论的 dropout，也会促进神经元的凝聚，从而加强网络的非线性学习行为。

7.4.2 两层 ReLU 神经网络的理论结果*

直观的尺度分析和实验结果展示出一致的边界来区分线性区域和凝聚区域。一个自然的问题是——是否存在一个理论来使直观的尺度分析成为严格的，并将上述在简单实验上获取的经

验相图推广到一般高维数据在两层 ReLU 神经网络上的情况。实际上我们可以建立严格的理论来证明尺度分析的有效性。我们提供两个非正式的定理，更严格的定理陈述和证明，请有兴趣的读者参考 Luo, Xu, Ma & Zhang (2021)。

定理 * 1. *如果 $\gamma < 1$ 或 $\gamma' > \gamma - 1$, 那么对于 $\boldsymbol{\theta}^0$ 的选择, 我们有较高的概率*

$$\lim_{m\to+\infty} \sup_{t\in[0,+\infty)} \mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}(t)) = 0. \tag{7.16}$$

定理 * 2. *如果 $\gamma > 1$ 并且 $\gamma' < \gamma - 1$, 那么对于 $\boldsymbol{\theta}^0$ 的选择, 我们有较高的概率*

$$\lim_{m\to+\infty} \sup_{t\in[0,+\infty)} \mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}(t)) = +\infty. \tag{7.17}$$

注 24. $\lim_{m\to+\infty} \sup_{t\in[0,+\infty)} \mathrm{RD}(\boldsymbol{\theta}_{\boldsymbol{w}}(t))$ *就像统计力学中相变分析的一个序参量，它对于区域分离至关重要，并在边界上呈现不连续性。*

在定理 1中，我们专注于线性区域，通过证明训练动力学的核（kernel）在整个动力学过程中没有显著变化，从而本质上证明了 $\boldsymbol{w}$ 的相对变化可以忽略不计。然而，在凝聚区域，训练动力学的核可能变得难以控制。这一困难使得定理 2的结果是非平凡的。以上两个定理完善了两层 ReLU 神经网络的相图，根据 $\lim_{m\to+\infty} \sup_{t\in[0,+\infty)} \mathrm{RD}\left(\boldsymbol{\theta}_{\boldsymbol{w}}(t)\right)$ 可以将不同的动力学区域区分开来。

线性区域内的神经网络行为，例如损失的指数衰减、以可再生核希尔伯特空间 (reproducing kernel Hilbert space, RKHS) 范数形式的隐式正则化等，已经被非常充分地研究了。然而，临界区域和凝聚区域的理解还很有限。在下一章节的研究中，我们将进一步探索在 $m \to \infty$ 时的一个标志性非线性行为——**凝聚现象**，这为后续的理论研究提供了启示。

7.4.3 三层 ReLU 神经网络的经验相图

在对两层神经网络相图的深入研究之后，我们可以进一步探索更为复杂的三层神经网络的相图。这一步是必要的，因为三层网络具有多层结构，每一层实际上可以处于不同的线性或非线性状态。

考虑一个每层宽度均为 m 的三层 ReLU 激活神经网络，其数学表达式为：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{\alpha}\boldsymbol{a}^T\sigma\left(\boldsymbol{W}^{[2]}\sigma\left(\boldsymbol{W}^{[1]}\boldsymbol{x}\right)\right),$$

其中输入 $\boldsymbol{x} \in \mathbb{R}^d$，$\alpha$ 为缩放因子，m 表示隐藏层的宽度。网络参数 $\boldsymbol{\theta} = \mathrm{vec}\left\{\boldsymbol{a}, \boldsymbol{W}^{[2]}, \boldsymbol{W}^{[1]}\right\}$，包括 $\boldsymbol{a} \in \mathbb{R}^m$ 和两个权重矩阵 $\boldsymbol{W}^{[2]} \in \mathbb{R}^{m\times m}, \boldsymbol{W}^{[1]} \in \mathbb{R}^{m\times d}$，这三组参数各自通过独立同分

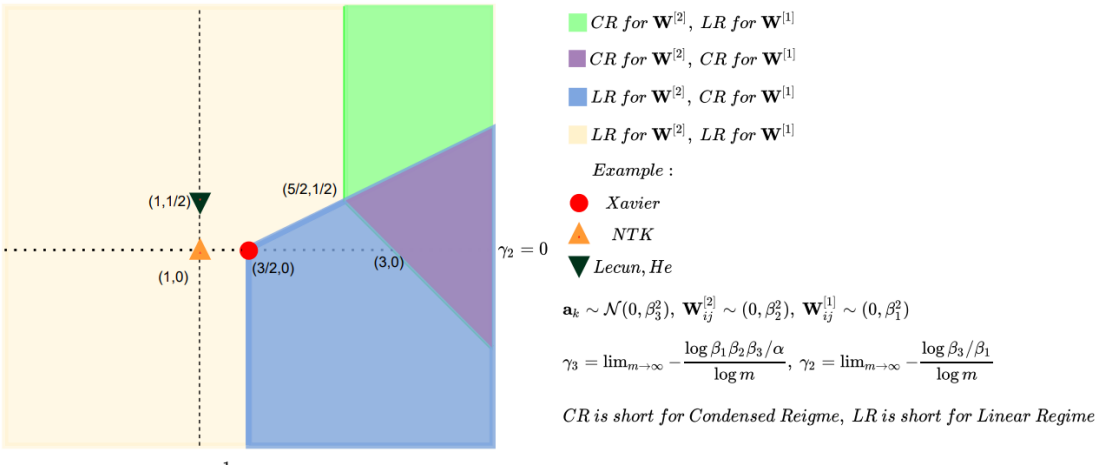


图 7.8: 三层无限宽 ReLU 神经网络实验上获取的相图

布的高斯分布初始化，具体为 $a_k \sim \mathcal{N}(0, \beta_3^2)$，$W_{kk'}^{[2]} \sim \mathcal{N}(0, \beta_2^2)$ 和 $W_{kk'}^{[1]} \sim \mathcal{N}(0, \beta_1^2)$。偏置项 $\boldsymbol{b}^{[1]}$ 可以通过将 $\boldsymbol{x}$ 和 $\boldsymbol{W}^{[1]}$ 扩展为 $\left(\boldsymbol{x}^\top, 1\right)^\top$ 和 $\left(\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}\right)^\top$ 来合并，而 $\boldsymbol{b}^{[2]}$ 可以通过将 $\overline{\boldsymbol{x}} = \sigma\left(\boldsymbol{W}^{[1]}\boldsymbol{x}\right)$ 和 $\boldsymbol{W}^{[2]}$ 扩展为 $\left(\overline{\boldsymbol{x}}^\top, 1\right)^\top$ 和 $\left(\boldsymbol{W}^{[2]}, \boldsymbol{b}^{[2]}\right)^\top$ 来合并。

和两层神经网络归一化的动力学一样，我们可以引入如下四个缩放参数:

$$\kappa_1 = \frac{\beta_3}{\beta_2}, \kappa_2 = \frac{\beta_3}{\beta_1}, \kappa_3 = \frac{\beta_1\beta_2\beta_3}{\alpha}, \bar{t} = \left(\alpha \Pi_{i=1}^3 \kappa_i\right)^{-\frac{2}{3}} t.$$

由于模型关于最外层参数 $\boldsymbol{a}$ 是线性的，所以我们只关心模型中的参数 $\boldsymbol{W}^{[1]}$ 和 $\boldsymbol{W}^{[2]}$ 的动力学演化。此外，在实际训练中，参数 $\boldsymbol{a}$ 和 $\boldsymbol{W}^{[2]}$ 的初始化往往是一样的，或者仅仅差一个常数倍，也即 $\beta_2 = B\beta_3$. 因此，我们可以用如下两个状态量来刻画三层无限宽 ReLU 神经网络的相图:

$$\gamma_2 = \lim_{m\to\infty} -\frac{\log \kappa_2}{\log m}, \gamma_3 = \lim_{m\to\infty} -\frac{\log \kappa_3}{\log m}.$$

如图 7.8所示，我们展示了三层 ReLU 网络经验上的相图，具体细节可以参考 Zhou, Zhou, Jin, Luo, Zhang & Xu (2022)。值得注意，和两层网络不一样的是，在同一时刻，不同的层可以处在不同的状态，这依赖于它们每层各自的初始化大小。

7.5 习题

1. 网络的初始化尺度大小如何影响模型训练的动力学过程？更大的初始化尺度是否倾向于使模型表现出更线性的特性，还是更小的初始化尺度会导致这种倾向？

2. 在分析两层神经网络 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k \cdot \boldsymbol{x})$ 的线性与非线性行为时，我们应该更加关注参数 $\boldsymbol{\theta}_a$ 还是参数 $\boldsymbol{\theta}_{\boldsymbol{w}}$ 在训练过程中的变化情况？为什么？

3. 如何在实验中定量地衡量参数在训练过程中的变化程度？

4. 对于两层 ReLU 神经网络 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k \cdot \boldsymbol{x})$，非线性区域的凝聚是什么意思？该如何刻画？

5. 考虑两层 ReLU 神经网络 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k \cdot \boldsymbol{x})$，对常见的初始化方法，例如 Kaiming He 初始化和 Xavier 初始化，计算它们在两层无限宽 ReLU 网络中对应的状态量参数 γ 和 γ'。

6. 若某种初始化方法在无限宽度网络的相图中位于线性区域，那么在有限宽度的网络中也同样位于线性区域吗？为什么？

7. 考虑一个两层 ReLU 神经网络：$f_{\boldsymbol{\theta}}^{\alpha}(\boldsymbol{x}) = \frac{1}{\alpha} \sum_{k=1}^{m} a_k \sigma\left(\boldsymbol{w}_k^{\top} \boldsymbol{x}\right), a_k^0 \sim N\left(0, \beta_1^2\right), \boldsymbol{w}_k^0 \sim N\left(0, \beta_2^2 \boldsymbol{I}_d\right)$，当 $m \to \infty$ 时，确定以下超参数组合对应的动力学状态，并在两层 ReLU 相图中进行标注：

   (i) $\alpha, \beta_1, \beta_2 = 1$;

   (ii) $\alpha = m^2, \beta_1, \beta_2 = 1$;

   (iii) $\alpha, \beta_1 = 1, \beta_2 = \frac{1}{m}$;

8. 考虑一个两层 ReLU 神经网络：$f_{\boldsymbol{\theta}}^{\alpha}(\boldsymbol{x}) = \frac{1}{\alpha} \sum_{k=1}^{m} a_k \sigma\left(\boldsymbol{w}_k^{\top} \boldsymbol{x}\right) \quad a_k^0 \sim N\left(0, \beta_1^2\right), \boldsymbol{w}_k^0 \sim N\left(0, \beta_2^2 \boldsymbol{I}_d\right)$，设计控制变量实验来研究网络宽度 m 对模型泛化能力的影响：

   (i) 固定 $\alpha, \beta_1, \beta_2 = 1$ 进行比较，是否能得出具有代表性的结论？

   (ii) 固定训练步数（例如 10000 步），这样的设置是否合理？

   (iii) 固定学习率（例如 0.1），这样的设置是否合理？

Chapter 8

凝聚现象

8.1 引入

在之前探索深度学习的过程中，我们经常会遇到一些重要但彼此关系并不明确的概念，比如非线性、频率原则、特征学习、损失景观、泛化以及复杂度等等。这些概念在初看之下可能显得孤立，然而，如果我们深入研究，就会发现它们其实都与一个核心现象紧密相关，这个现象就是“凝聚现象”。

我们对凝聚现象的关注源于对相图分析和频率原则的深入研究和理解。在相图中，神经网络的训练过程可以划分为线性态和非线性态两种相态。在线性态下，神经网络的学习过程等价于一个核方法（模型关于参数是线性的方法），可以很好地被频率原则刻画。然而，非线性态下的神经网络行为虽然整体也符合频率原则，但是它有更多复杂的特征。

在对非线性态的探索中，我们发现凝聚现象是一个普遍存在的现象。在神经网络中，凝聚现象是指同层神经元趋于一致的现象。神经元的一致性本质上是不同神经元的函数（指网络输入到该神经元的输出的映射）相同。

本章的内容将会通过一系列的实验和理论分析来说明，凝聚现象在非线性训练过程中的普遍性，比如有以下四点：第一，凝聚现象不依赖于网络的宽度、深度。无论是在无穷宽的神经网络中，还是在有限宽度的常用神经网络中，我们都可以观察到凝聚现象的发生。这意味着，凝聚现象不受网络规模的限制。第二，凝聚现象不依赖于网络的结构。全连接网络、卷积网络、残差连接网络等都会出现凝聚现象。第三，凝聚现象对数据量不敏感。无论网络是过参数化还是欠参数化，凝聚现象都可以发生。第四，凝聚现象对损失函数和优化算法不敏感。

凝聚现象的普遍性背后隐藏着诸多引导凝聚的结构。事实上，在神经网络训练的各个阶段，都有引导凝聚的机制。在训练的初始阶段，如果采用小初始化，那么各个神经元在遵循相

同的动力学方程的同时，神经元的初始点又非常靠近，因此不同神经元容易趋于一致。在训练的中间阶段，神经网络常经历某些鞍点附近。在这些鞍点附近，可以观察到神经网络的输出函数和某个等效的小网络的输出函数很相似，这也是一种引导凝聚的方式。这种现象我们会在下一章详细分析。在训练的末态，网络的全局极小具有特定的几何结构，这种几何结构会促进凝聚的发生。因此，从训练的初始到末态，神经网络的损失景观中都有引导凝聚的结构。此外，还有一些正则化的技术也可以引导凝聚，比如 dropout。

凝聚现象和泛化有密切的联系，它是一种有效的控制复杂度的方式。通过让神经元趋同，凝聚使得宽网络等价于某个窄网络。这使得模型的有效复杂度被降低，有助于防止模型过拟合训练数据，从而提高模型的泛化能力。

凝聚现象在特征学习中扮演着重要的角色。特征学习是深度学习的关键优势，它允许模型自动从原始数据中提取有价值的特征。神经网络中的每个神经元可以视为特征提取器，其学习到的权重决定了它提取的特征。当凝聚现象发生时，同层神经元的函数映射趋于一致，意味着它们学习到了相似的特征，这会用尽量少的关键特征来捕捉数据当中的重要信息。这样既能够拟合训练数据，又能够保证好的泛化。此外，凝聚现象有利于增强特征学习的鲁棒性，即使部分神经元受到噪声影响，其他神经元仍能提供相似特征，从而维持模型性能，有利于提高模型的鲁棒性和泛化能力。凝聚现象可能使神经网络学习到更有效、更鲁棒的特征，提高模型性能。但凝聚现象与特征学习的深入关系仍是一个活跃的研究领域，需要更多的理论和实验研究来进一步探索。

在本章中，我们将首先从实验层面展示凝聚现象的存在。然后，我们将介绍实现凝聚的一个机制——初始凝聚，并对其进行详细的分析和探索。

科普篇

什么是神经网络的凝聚现象？

凝聚现象是指在神经网络的训练过程中，同层神经元会有趋同的现象。例如，多个神经元最终可能对同一个输入产生非常相似的输出。尽管神经元一开始是随机初始化的，但通过训练，它们有趋于一致的倾向。这种现象揭示了神经网络内部有隐式地控制模型有效复杂度的机制，有助于我们更好地理解神经网络的运作方式。

凝聚现象的普遍性

凝聚现象在神经网络的训练过程中普遍出现，具体表现为以下四点：

- 凝聚现象不依赖于网络的宽度、深度。

- 凝聚现象不依赖于网络的结构。
- 凝聚现象对数据量不敏感。
- 凝聚现象对损失函数和优化算法不敏感。

为什么凝聚现象很重要？

研究凝聚现象可以帮助我们理解许多神经网络中的重要方面，如非线性、特征学习、和泛化。

- **理解非线性态的行为：**在研究神经网络的过程中，我们发现训练过程大致分为两个部分：线性态和非线性态。在线性态下，神经网络的学习过程很像传统的线性方法（核方法），可以很好地被频率原则刻画。然而，在非线性态下，神经网络的行为更加复杂，频率原则只能定性描述，但总体上依然遵循一定规律，例如这时会出现凝聚现象。
- **理解神经网络的泛化能力：**凝聚现象和泛化有密切的联系，它是一种有效的控制复杂度的方式。通过让神经元趋同，凝聚使得宽网络等价于某个窄网络。这使得模型的有效复杂度被降低，有助于防止模型过拟合训练数据，从而提高模型的泛化能力。
- **理解神经网络的特征学习：**神经网络中的每个神经元可以视为特征提取器，其学习到的权重决定了它提取的特征。当凝聚现象发生时，同层神经元的函数映射趋于一致，意味着它们学习到了相似的特征，这会用尽量少的关键特征来捕捉数据当中的重要信息。

初始凝聚

在小初始化下，损失函数在训练刚开始时会出现停滞。在这个停滞阶段，神经元的输入权重方向会趋于一致，即使在初始化时神经元的输入权重方向是完全随机的。我们把这种现象叫做初始凝聚。

- **实验验证：**我们以一个两层的神经网络为例。假设我们有如下的网络结构：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}), \tag{8.1}$$

其中 $\sigma(\cdot)$ 是激活函数，$\boldsymbol{w}_j$ 是输入权重。我们选择多种激活函数（如 $\tanh(x)$、$x\tanh(x)$、$x^2\tanh(x)$），并采用小初始化，训练网络拟合某个目标函数，并记录每个神经元在训练早期的行为。通过余弦相似度热力图可视化神经网络是否发生凝聚，以及凝聚方向的个数（见图 8.10）。

- **理论结果：**(i)：在小初始化条件下，神经网络模型可以用其泰勒展开的主导项来近似。激活函数在零点附近的泰勒展开决定了神经网络的行为特性。我们引入了重数的概念，即激活函数在零点的主导项的阶数。我们理论证明了初始凝聚的方向个数为激活函数重数的两倍。(ii)：理论预测了初始凝聚发生的时间 T 与网络宽度 m 的关系。在某些区域，T 与 $\log(m)$ 呈线性正相关。在其他区域，T 随 m 的增加而快速减小。这一结果能够帮助我们预判初始凝聚现象的发生时间。

8.2 凝聚现象的实验

在本节中，我们将深入探索并展示凝聚现象的各种实验结果。这些实验涵盖了不同类型的神经网络结构，包括全连接网络、卷积神经网络以及残差网络。我们将从实验结果出发，探索凝聚现象是如何在神经网络训练过程中发生的。

8.2.1 凝聚的过程

我们先用一个一维的例子展示神经网络训练过程中凝聚是如何发生的。我们选取目标函数(如图8.1所示) 为

$$f(x) = -\mathrm{ReLU}(x) + \mathrm{ReLU}(2\times(x+0.3)) - \mathrm{ReLU}(1.5\times(x-0.4)) + \mathrm{ReLU}(0.5\times(x-0.8)), \quad (8.2)$$

其中 $\mathrm{ReLU}(x) = \max\{0, x\}$。我们采用一个两层的 ReLU 神经网络来拟合这个函数。神经网

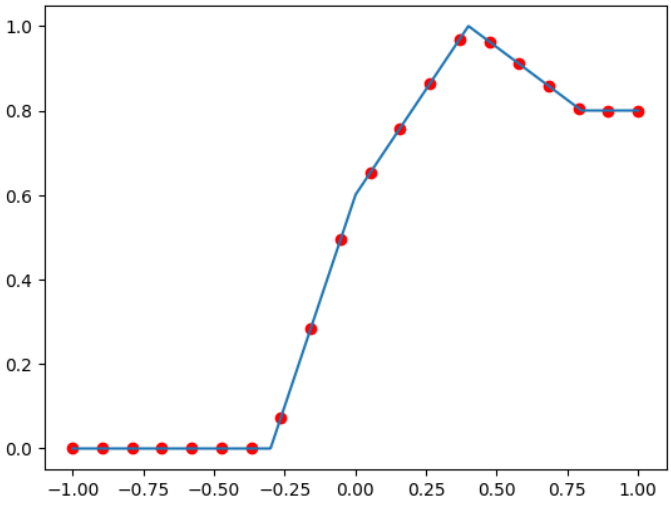


图 8.1: (8.2)的函数图像

络的宽度为 $m = 100$。该神经网络的模型如下：

$$f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \mathrm{ReLU}(\boldsymbol{w}_k \cdot \boldsymbol{x}) = \sum_{k=1}^{m} a_k \mathrm{ReLU}(w_k x + b_k), \tag{8.3}$$

其中 $\boldsymbol{w}_k = (w_k, b_k)$，$\boldsymbol{x} = (x, 1)$，$\boldsymbol{w}_k \cdot \boldsymbol{x} = w_k x + b_k$。

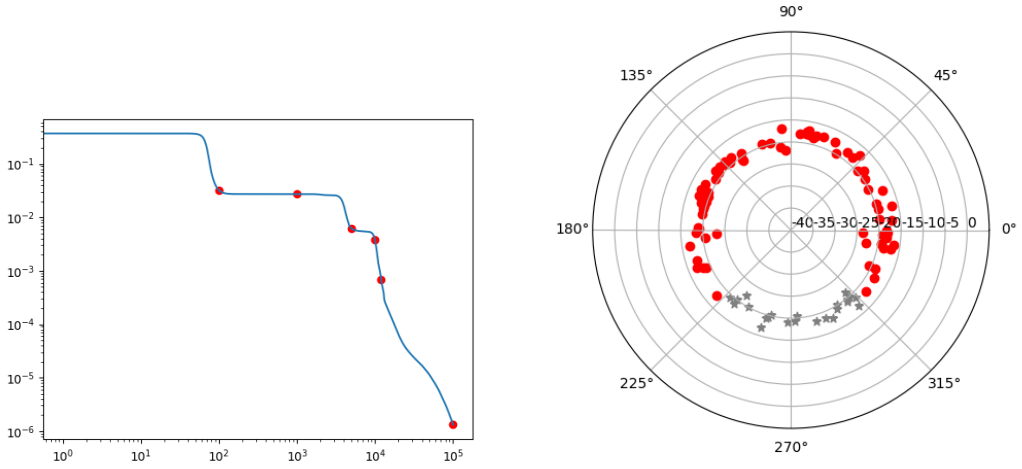
值得注意的是，对于 ReLU 激活函数，每个神经元的参数对 $(a_k, \boldsymbol{w}_k)$ 可以分离为一个单位方向特征 $\hat{\boldsymbol{w}}_k = \boldsymbol{w}_k / \|\boldsymbol{w}_k\|_2$ 和一个表示其对输出贡献的振幅 $A_k = |a_k| \|\boldsymbol{w}_k\|_2$，即 $(A_k, \hat{\boldsymbol{w}}_k)$。对于一维输入，由于加入了偏置项，$\boldsymbol{w}_k$ 是二维的。因此，我们使用每个 $\hat{\boldsymbol{w}}_k$ 相对于 x 轴的 $[-\pi, \pi)$ 内的角度 $\hat{\Omega}_k$ 来表示其方向。即 $\hat{\Omega}_k = \arctan(\frac{b_k}{w_k})$。

首先，我们来看一下训练过程中损失函数的变化（见图8.2(a)）。我们发现损失函数出现了多次停滞，这表明训练过程呈现出非常强的非线性。如果模型关于参数是线性的话，例如模型是 $wx + b$，那均方损失函数关于 w 和 b 都是二次函数，有唯一的最小值，是个凸问题。传统的核方法就是这类凸问题，已经研究得比较清楚了；对于非线性的问题，一般有比较复杂的性质。

为了更细致地研究非线性训练过程中发生的细节，我们选择了几个时刻，来进一步观察参数变化的细节。我们选择了初始时刻以及图8.2(a) 中红色圆点对应的六个训练过程中的时刻，将其训练轨迹展示在图8.2(b) 和图8.3中画出。这些图展示了各个训练阶段的 $(\hat{\Omega}_k, A_k)_{k=1}^m$ 的极坐标散点图。极坐标图中的极径代表 $\log_{10}(A_k)$ 的大小，极角 $\hat{\Omega}_k$ 则代表 $\hat{\boldsymbol{w}}_k$ 的方向。其中 A_k 如同之前定义，$\hat{\Omega}_k = \mathrm{acrtan}(\frac{b_k}{w_k})$。圆点表示 $(\hat{\Omega}_k, A_k)$ 在当前时刻的分布，实线为对应神经元模长随时间演化的轨迹。这些时刻对应的神经网络输出在图8.4中展示，其中蓝色五角星表示真实的训练点数据，红色实线表示模型输出的函数。

如图8.2(b)，在 epoch $= 0$ 时，由于输入权重的初始化是随机的，神经元输入权重的朝向也各不相同。然而，在图8.3(a) 中 epoch $= 100$ 时，幅值较大的那些神经元的输入权重都已经凝聚到了某个特定的方向。从图 (a) 到图 (f) 中，我们可以观察到，随着训练的进行，神经元凝聚的方向逐渐增多，每个方向上的神经元的模长也逐步增大。形象地说，神经元逐渐“长”出了凝聚的方向。并且长出新的方向的神经元有两种情况。一种是从幅值很小的那部分神经元长出来，另一种是从已经凝聚到某个方向且幅值较大的神经元转移过来。训练结束时，如图 (f) 所示，神经元一共“长”出了六个方向。

需要注意的是，当对所有输入 x，$wx + b < 0$ 时恒成立时，$\mathrm{ReLU}(wx + b)$ 在整个训练过程都保持为 0，因为梯度恒为 0，参数不会被训练。在该实验设定下，图8.2中标记为五角星的神经元对所有训练数据都满足 $wx + b < 0$，因此这些神经元的输出始终是 0。这些神经元的参数在训练过程中不会发生变化，不参与到凝聚的过程中。



(a) 损失函数　　(b) epoch= 0

图 8.2: 左图为两层 ReLU 神经网络拟合函数(8.2)的训练过程中损失函数的图像，右图为训练初始时刻 $(\hat{\Omega}_k, A_k)$ 的分布

8.2.2 全连接网络的凝聚现象

在全连接神经网络结构中, 第 $L+1$ 层的 m_{L+1} 维输出 $\boldsymbol{h}_{L+1}$ 可表示为:

$$\boldsymbol{h}_{L+1}(\boldsymbol{x}) = \sigma(\boldsymbol{W}^{[L]}\boldsymbol{h}_L(\boldsymbol{x}) + \boldsymbol{b}^{[L]}),$$

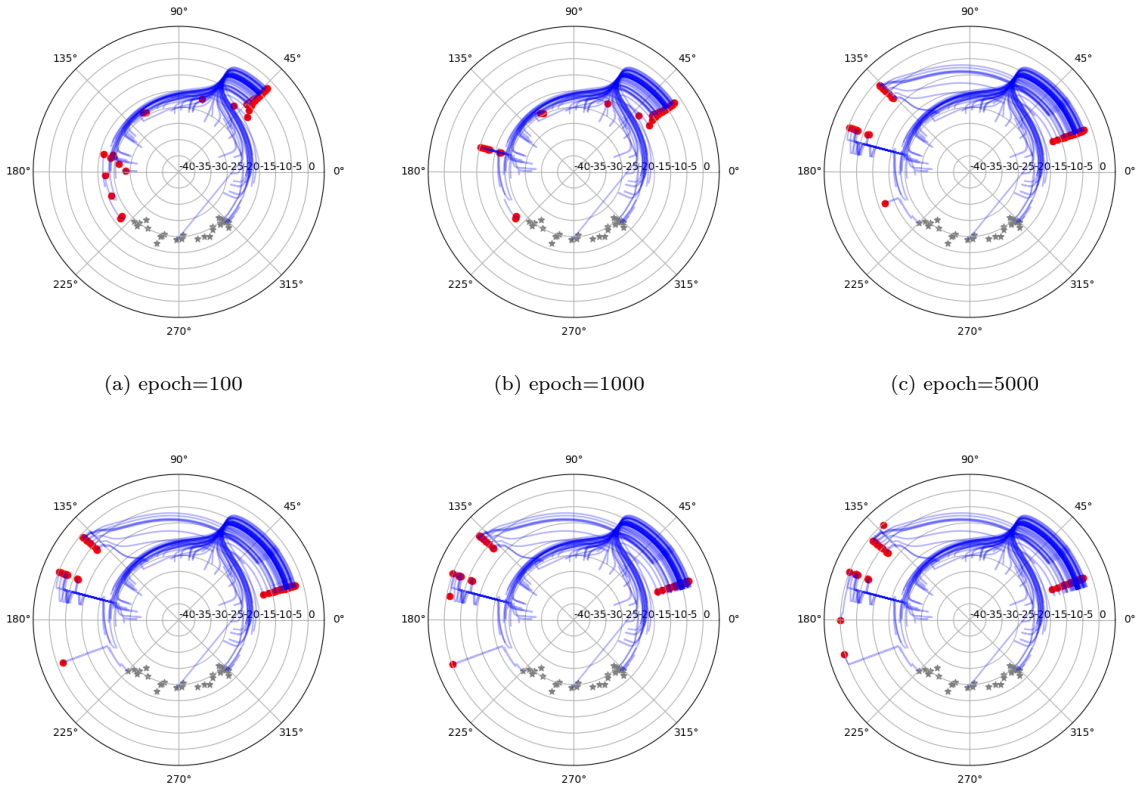
其中,$\boldsymbol{W}^{[L]} \in \mathbb{R}^{m_{L+1}\times m_L}$ 为权重矩阵，$\boldsymbol{b}^{[L]} \in \mathbb{R}^{m_{L+1}\times 1}$ 为偏置向量，σ 为激活函数。我们可将 $\boldsymbol{W}^{[L]}$ 和 $\boldsymbol{b}^{[L]}$ 合并为扩展权重矩阵 $\bar{\boldsymbol{W}}^{[L]} \in \mathbb{R}^{m_{L+1}\times(m_L+1)} := (\boldsymbol{W}^{[L]}, \boldsymbol{b}^{[L]})$，那么 $\bar{\boldsymbol{W}}^{[L]}$ 包含 m_{L+1} 个 m_L+1 维向量, 我们要观察这些向量是否发生凝聚现象。在一些简单案例中, 例如相图分析章节的两层 ReLU 网络凝聚以及8.2.1节所展示的凝聚过程, 由于输入为一维, 可通过观察神经元方向特征 $\hat{\boldsymbol{w}}_k$ 在极坐标系中的分布来判断是否发生凝聚。

对于高维数据, 无法使用极坐标系。在这种情况下, 我们可以利用余弦相似度来衡量两个向量之间夹角的大小。

余弦相似度: 两个向量 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 的余弦相似度定义为

$$D(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^\intercal \boldsymbol{v}}{(\boldsymbol{u}^\intercal \boldsymbol{u})^{1/2}(\boldsymbol{v}^\intercal \boldsymbol{v})^{1/2}}. \tag{8.4}$$

对于一组向量, 我们可以计算出它们之间的余弦相似度矩阵。得到这个矩阵后, 我们按照其最大特征值对应的特征向量不同分量的大小, 对神经元进行排序。然后, 按照这个顺序绘制向量之间的余弦相似度热力图。通过观察热力图, 我们可以发现向量之间是否存在凝聚现象。



(a) epoch=100　(b) epoch=1000　(c) epoch=5000

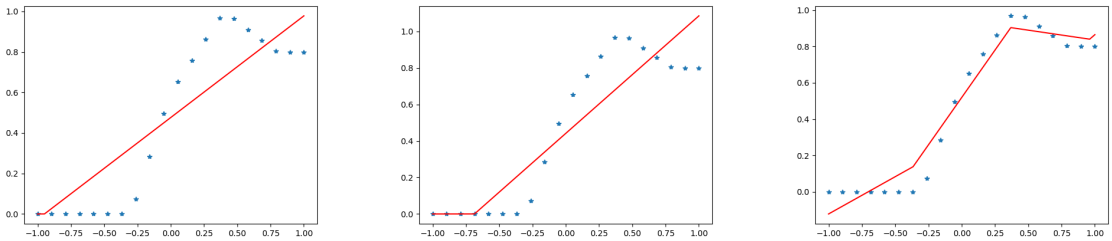
(d) epoch=10000　(e) epoch=12000　(f) epoch=100000

图 8.3: 两层 ReLU 神经网络拟合(8.2)时神经元随时间演化的 $(\hat{\Omega}_k, A_k)$ 分布，模长的坐标尺度为对数尺度

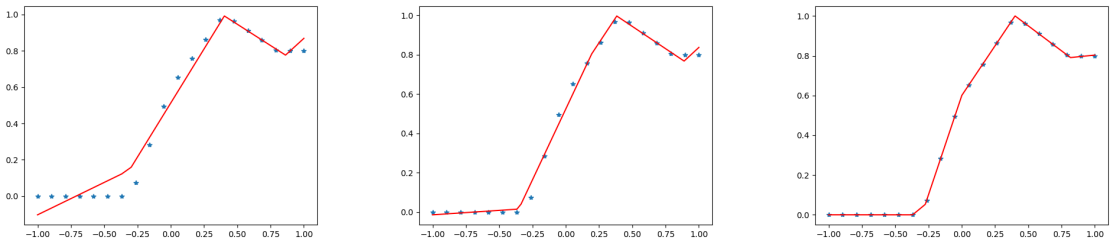
8.2.3 卷积神经网络的凝聚现象

如图8.5所示, 我们训练了一个仅有一个卷积层加一个全连接层的卷积神经网络，训练数据是 MNIST（一个常用的小型图像数据集），并采用交叉熵损失作为损失函数，激活函数为 tanh(x)。热力图中的颜色表示不同卷积核的 $D(\boldsymbol{u}, \boldsymbol{v})$，输出层使用 softmax，优化器为 Adam，full batch 训练。卷积核大小 $m = 3$，学习率为 2×10^{-4}。训练到训练集准确率达到 100% 结束，此时测试集准确率为 97.62%。

图8.5(a) 和 (d) 分别是训练过程的误差和准确率。图8.5(b) 和 (e) 分别是训练初始和训练准确率达到 100% 时，卷积核的余弦相似度热力图，其中卷积层共有 32 个通道，卷积核的尺寸为 3×3，因此是 32 个 9 维向量两两之间的余弦相似度。图8.5(c) 和 (f) 则分别是训练集和



(a) epoch=100　(b) epoch=1000　(c) epoch=5000



(d) epoch=10000　(e) epoch=12000　(f) epoch=100000

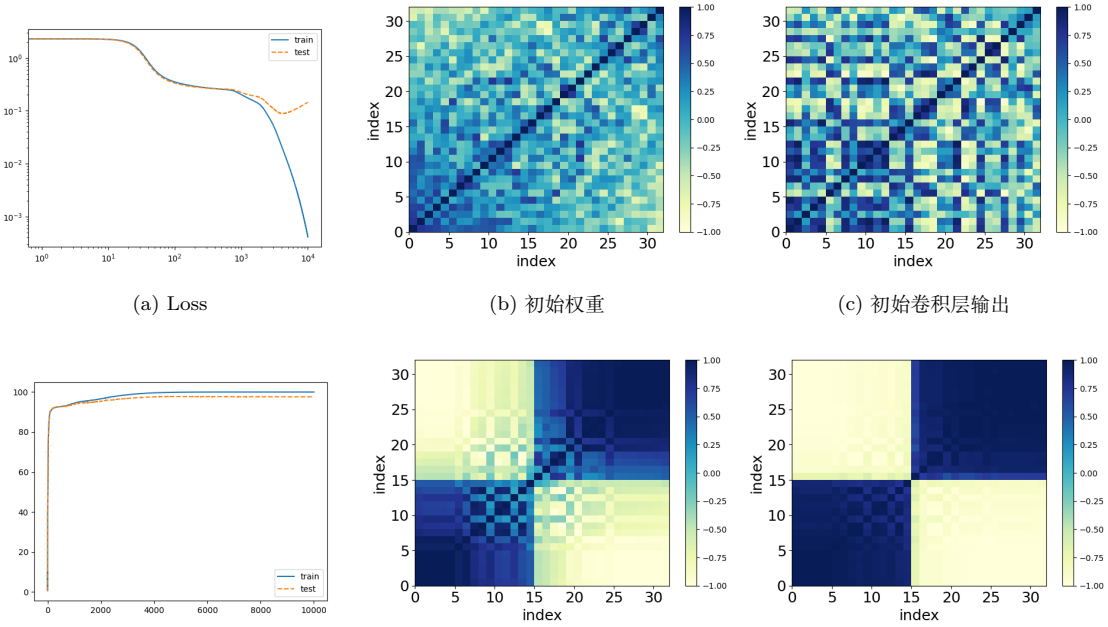
图 8.4: 图8.3对应时刻神经网络的输出

测试集一共 70000 个数据经过该卷积层得到 $70000 \times 32 \times 28 \times 28$ 的四维张量。我们固定第二个维度并将其余维度展平得到 32 个 $70000 \times 28 \times 28$ 的神经网络输出向量之间的余弦相似度。

从图中可以看出，第一，在训练初始阶段向量之间没有凝聚关系。而在训练完成后，卷积层及训练数据经过卷积层后均出现了块状结构，发生了明显的凝聚现象，大约凝聚到了两个方向相反的方向上。第二，图（f）中的块状结构比图（e）中的块状结构更清晰，这表明末态卷积层输出的凝聚程度强于末态权重的凝聚程度。

8.2.4 残差神经网络的凝聚现象

凝聚现象在残差神经网络中也会发生。我们以深度学习网络模型 ResNet18 为例，展示它的训练过程中的凝聚现象。ResNet18 是一种应用于视觉任务的卷积神经网络，擅长处理图像和视频等数据。该网络由 18 个可学习参数层 (17 个卷积层, 1 个线性层) 和批归一化层 (Batch Normalization)、池化层等组成。这些层通过特定的结构——残差块，被有序地组织起来。尽管 ResNet18 在深度学习模型中规模较小，但它能够在 ImageNet 数据集上达到 top1 准确率 73.16%，top5 准确率 91.03%(数据来源`https://huggingface.co/timm/resnet18.a1_in1k`)。



(a) Loss (b) 初始权重 (c) 初始卷积层输出

(d) Accuracy (e) 末态权重 (f) 末态卷积层输出

图 8.5: 小初始化 (卷积层和全连接层初始均服从 $\mathcal{N}(0, 96^{-8})$) 下单层 CNN 训练后的最终阶段的凝聚

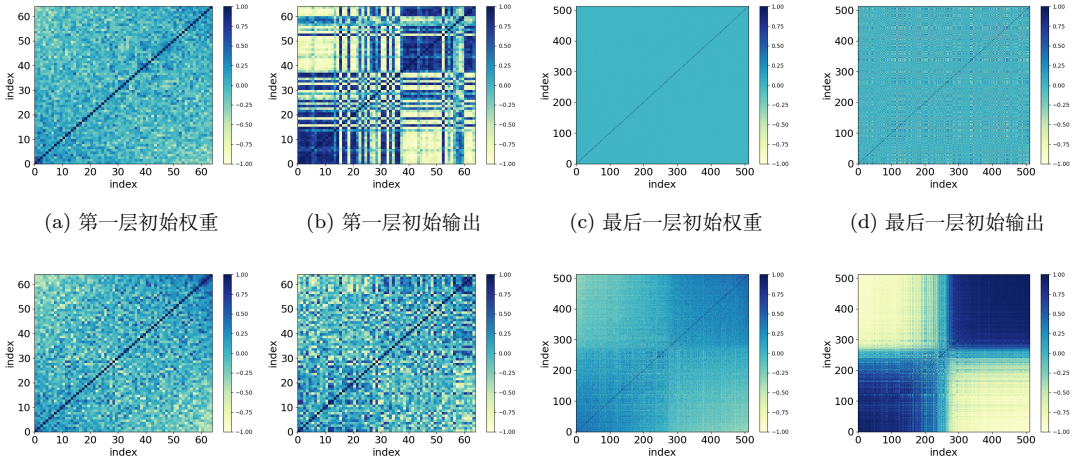
在残差神经网络中，我们采用与卷积神经网络类似的方式处理卷积核，唯一的区别在于对于彩色图片的多通道卷积核需要在通道以及卷积核尺寸上全部展平。而对于神经网络的输出，我们随机选取 256 张训练图片和 256 张测试图片组成一个 512 张图片的批次 (batch)，在这个批次中进行类似于卷积神经网络的处理来观察各向量之间的凝聚情况。

图8.6中 (a)(e) 和 (c)(g) 的分别为 Resnet18 中第一个卷积层和最后一个卷积层的权重按输出通道展开得到。输出为一个批次的训练数据和测试数据合并通过 Resnet18 卷积层得到的特征按照通道展开得到。损失函数为交叉熵，优化器为 Adam，训练集批次的大小为 128，测试集批次的大小为 100，学习率为 10^{-3}。训练结束时训练集准确率为 99.7%，测试集准确率为 92.5%。

我们观察到在训练结束后的最后一层权重和输出，如图8.6(g) 和 (h) 所示，与初始时刻如图8.6(c) 和 (d) 相比，出现了比较明显的块状结构。这表明神经元在该层展现出一定的凝聚现象。然而，并非每一个卷积层都会呈现显著的凝聚现象。如图8.6(e) 和 (f) 所示，第一层的权重和输出在经过训练后没有出现明显的凝聚现象。

值得注意的是，在 ImageNet 数据集上预训练的 ResNet-18 模型中，我们也观察到类似的

情况。如图8.7(b) 和 (d) 所示, 最后一个卷积层的权重和输出均出现了凝聚现象, 而第一层的权重和输出 (如图8.7(a) 和 (c)) 则没有表现出那么明显的凝聚现象。



(a) 第一层初始权重　(b) 第一层初始输出　(c) 最后一层初始权重　(d) 最后一层初始输出

(e) 第一层末态权重　(f) 第一层末态输出　(g) 最后一层末态权重　(h) 最后一层末态输出

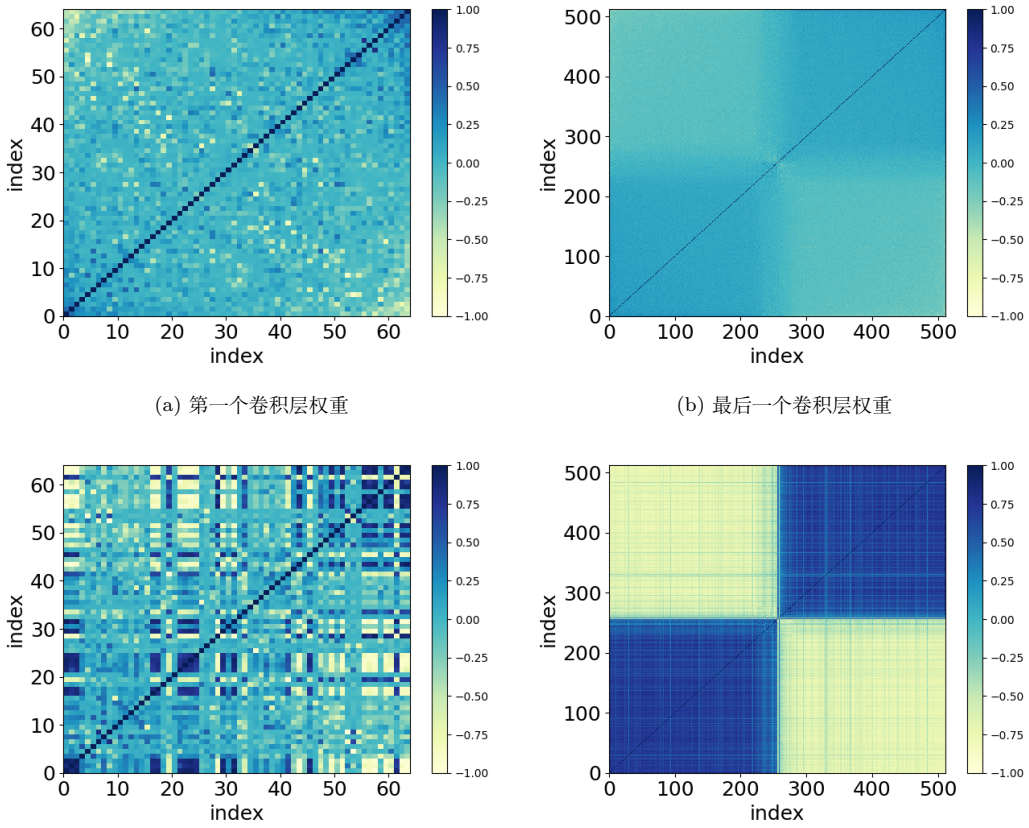
图 8.6: 默认初始化下利用 ResNet-18 训练 CIFAR10 数据集过程中的凝聚现象

8.3 凝聚现象的探讨

8.3.1 凝聚现象的定义

在上一节的内容中，我们观察到了一个非常重要的现象：在神经网络的训练过程中，同层神经元的行为表现出了一种明显的趋同性。这种趋同可以表现在神经元的输出上，即神经元对输入的响应趋于一致，也可以表现在神经元的权重上，即它们的权重向量在训练过程中趋于一致。这种同层神经元趋于一致的现象，我们将其命名为“凝聚现象”。在多层网络当中，即使输入权重不一样，神经元的映射也可以一样，这也是一种常见的趋同现象。

我们给出一个理想的凝聚的示意图。如图17.1所示，虽然在初始化时，各个神经元的输入权重差异很大 (示意图中连线的粗细与线的类型不同)，但是在经过一段时间训练后，中间隐藏神经元分成了两类，前两个神经元是一类，后三个神经元是另一类。在每一类中，不同神经元的输入权重是完全一样的（其对应连线的类型和粗细均相同），因此，它们的输出也是一样的。这就表明发生了理想的凝聚。需要强调的一点是，凝聚是一个过程。即使最终没有达成理想的凝聚，但是只要训练过程中神经元有趋同性，它也是凝聚。



(a) 第一个卷积层权重

(b) 最后一个卷积层权重

(c) 第一个卷积层输出

(d) 最后一个卷积层输出

图 8.7: 利用 ImageNet 预训练的 ResNet-18 模型的凝聚现象。模型参数来源：`https://huggingface.co/timm/resnet18.a1_in1k`

8.3.2 对于凝聚现象的理解

在确认凝聚现象是一个普遍存在的现象后，我们可以从两个方向进一步探索：一是从动力学或损失景观的角度分析凝聚现象的产生原因；二是探讨我们能否利用这种凝聚现象来更深入地理解神经网络的泛化性能。

对于凝聚现象的产生原因，我们可以从动力学的角度进行深入研究。神经网络的训练过程中，神经元的状态会随着时间的推移而变化。这种变化可以被视为一个动力学系统，其中的稳定性和收敛性质可能会影响神经元是否会发生凝聚。例如，如果神经元的动力学系统存在某种特殊的稳定点或吸引子，那么神经元可能会向这些稳定点趋近，从而发生凝聚。另一方面，我们也可以从损失景观的角度进行分析。在神经网络的训练过程中，我们通常希望最小化某种损

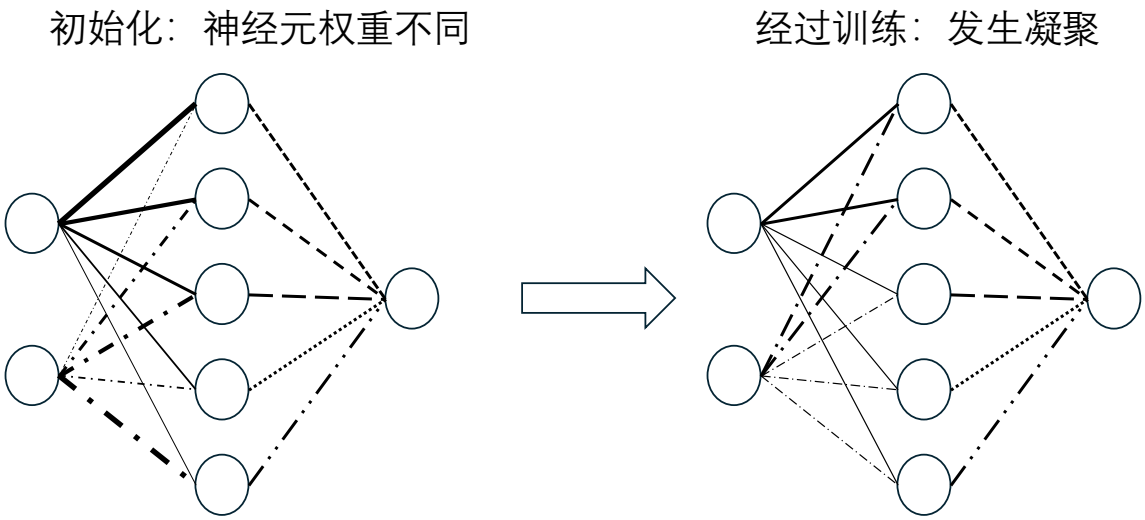


图 8.8: 理想的凝聚现象

失函数，这个损失函数定义了一个损失景观，神经元的状态会在这个景观中进行搜索。如果损失景观存在某些特殊的吸引区域，那么神经元可能会被吸引到这些区域，从而发生凝聚。

对于凝聚现象对神经网络泛化性能的影响，我们可以从两个方面进行探讨。首先，凝聚意味着神经网络可以被一个规模更小的神经网络表示。这意味着它的有效复杂度低于它的模型参数量。如果数据本身不复杂，可以通过凝聚降低复杂度。另一方面，如果神经网络的有效复杂度过低，可以通过长出新的方向增加模型的表达能力。因此凝聚提供了一种自适应的控制拟合复杂度的机制，能促进神经网络学到数据的低复杂度拟合。

我们的研究表明，凝聚现象是神经网络非线性动力学的关键现象。通过研究凝聚现象，我们可以更深入地理解神经网络的行为和性能。

8.4 初始凝聚

初始训练时的动力学行为对整个训练过程至关重要，因为它在很大程度上决定了神经网络的训练动态以及其最终停留的区域，这会影响神经网络在训练的最终阶段的特性。因此，单独研究初始训练阶段的行为本身是一个重要的研究方向。

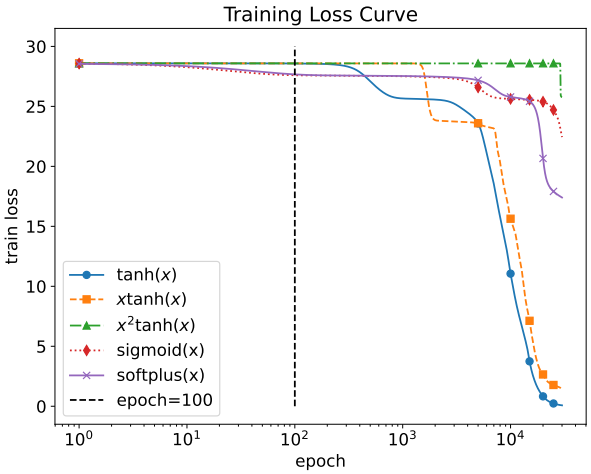
在相图的研究中，我们发现当采用小初始化的时候，网络的训练过程呈现出高度的非线性，

有很强的凝聚特征。这种凝聚在训练的初始阶段就表现得非常强烈。这种初始阶段的凝聚可以通过将不同的神经元凝聚到一些特定的特征，使得神经网络对初始化中的随机性的敏感度降低，即不同初始化的随机种子可以更容易达到相同的效果。在这一章节中，我们将观察神经网络在小初始化下的初始凝聚现象，并探索初始凝聚现象发生的机制 (Zhou, Zhou, Luo, Zhang & Xu 2022)。

8.4.1 初始凝聚的实验

在实验中，我们观察到了一个有趣的现象。尽管参数初始化是完全随机的，但在训练的初始阶段，神经网络都会经历一种我们称之为“初始凝聚”的行为。具体来说，在小初始化下，这种现象会使得某些神经元的输入权重方向趋于一致，即使在初始阶段它们的权重方向是完全随机的。在本节中，我们将从实验结果出发，仔细观察这种现象。

初始凝聚发生在训练的初始阶段。那什么是训练的初始阶段呢？我们知道，参数全为 0 的点是两层（多层）神经网络的鞍点。在小初始化下，神经网络的初始值很接近 0，因此也就很接近鞍点，所以损失函数在训练刚开始时会经历停滞，我们把这个停滞叫做训练的初始阶段。如图8.9所示，epoch= 100 之前可大致认为是训练的初始阶段。



(a)

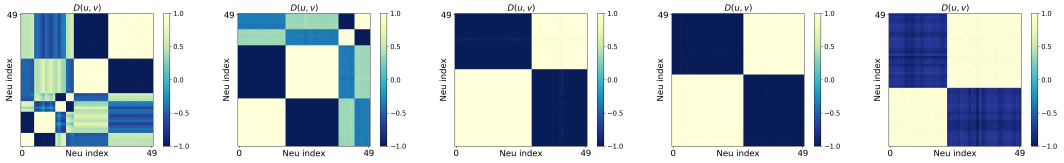
图 8.9: 图8.10的损失函数图像

两层神经网络的初始凝聚现象

图8.10展示了训练初始阶段的凝聚现象。我们使用宽度依次为 5-50-1 的两层全连接神经网络来拟合从五维函数 $\sum_{k=1}^{5} 3.5\sin(5x_k+1)$ 中采样的 $n=80$ 个训练数据，其中 $\boldsymbol{x}=(x_1,x_2,\cdots,x_5)^{\intercal}\in\mathbb{R}^5$，每个 x_k 均匀随机地从 $[-4,2]$ 中采样。在 epoch $=100$ 时，我们画出了热力图，来可视化不同神经元的输入权重之间的余弦相似度。热力图的颜色深浅代表余弦相似度的大小，横纵坐标代表神经元的索引。图 (a) 到 (e) 使用了不同的激活函数，在图片的字标题中分别给出。颜色表示 epoch 100 时两个隐藏层神经元输入权重的 $D(\boldsymbol{u},\boldsymbol{v})$。子图 (a,b,d),(c) 和 (e) 的学习率分别为 $\mathrm{lr}=10^{-3},8\times10^{-4},2.5\times10^{-4}$。

图中揭示了几个重要现象。第一，无论我们采用何种激活函数，图 (a) 到 (e) 的热力图都呈现了明显的块状结构，这表明神经元凝聚到了特定的方向上。第二，当改变激活函数时，神经元凝聚的方向个数也可能有改变。具体来说，在图 (c)、(d)、(e) 中，神经网络凝聚在两个相反的方向上，即一条线上。在图 (b) 中，神经元凝聚到了两条线上，共四个凝聚方向。而在图 (a) 中，神经元凝聚到了三条线上，共六个凝聚方向。

总结一下，我们观察到了，在训练初始阶段神经元会凝聚到特定的方向上，并且凝聚方向的个数与网络的激活函数有关。



(a) $x^2\tanh(x)$ (b) $x\tanh(x)$ (c) $\mathrm{sigmoid}(x)$ (d) $\tanh(x)$ (e) $\mathrm{softplus}(x)$

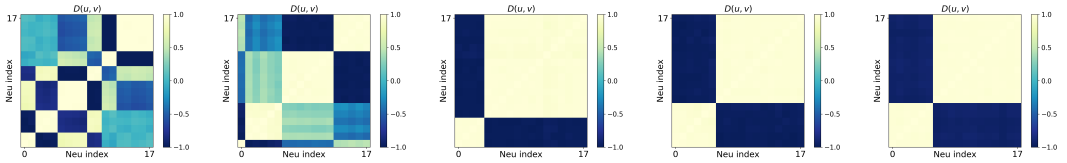
图 8.10: 两层神经网络的初始凝聚

多层神经网络的初始凝聚现象

在多层全连接网络和多层残差网络中，都可以发生凝聚。我们这里拿残差网络作为例子展示凝聚现象。在深度学习中，经常引入残差连接来克服梯度消失。残差的结构是 $\boldsymbol{h}_{l+1}(\boldsymbol{x})=\sigma(\boldsymbol{W}_l\boldsymbol{h}_l(\boldsymbol{x})+\boldsymbol{b}_l)+\boldsymbol{h}_l(\boldsymbol{x})$，其中 $\boldsymbol{h}_l(\boldsymbol{x})$ 是第 l 层的输出。我们对具有残差连接的六层神经网络进行实验。训练集为从 $\sum_{k=1}^{d} 4\sin(12x_k+1)$ 采样的 80 个点，其中每个 x_k 在 $[-4,2]$ 上均匀采样。$n=80$,$d=3$,$m=18$,$d_{\text{out}}=1$,$\mathrm{var}=0.01^2$,$\mathrm{lr}=4\times10^{-5}$。为了展示各种激活函数的不同，我们分别将隐藏层 1 到隐藏层 5 的激活函数设置为 $x^2\tanh(x)$, $x\tanh(x)$, $\mathrm{sigmoid}(x)$, $\tanh(x)$ 和 $\mathrm{softplus}(x)$。我们画出了各个层内，神经元输入权重的余弦相似度的热力图。实验结果如

图8.11所示。子图中的步数分别为 epoch 1000、900、900、1400 和 1400。

第一, 在图8.11中, 输入权重在隐藏层 1 到隐藏层 5 分别凝聚在三条、两条、一条、一条和一条线上。这表明在神经网络的各个层均发生了凝聚现象。第二, 将图8.11和图8.10对比, 我们注意到, 尽管网络结构不同、数据点不同、层位置不同, 等等, 但是只要激活函数相同, 那么该层的神经元的凝聚方向的个数就相同。这表明凝聚的方向个数和激活函数的关系十分紧密。



(a) layer 1 (b) layer 2 (c) layer 3 (d) layer 4 (e) layer 5

图 8.11: 带残差连接的六层神经网络的初始凝聚

8.4.2 初始凝聚的机制探索*

本节我们将初步探索初始凝聚的机制, 以及初始凝聚的方向个数和激活函数的关系。我们从两层网络开始研究。一个两层神经网络为

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}), \tag{8.5}$$

其中 $\sigma(\cdot)$ 是激活函数, $\boldsymbol{w}_j = (\bar{\boldsymbol{w}}_j, \boldsymbol{b}_j) \in \mathbb{R}^{d+1}$ 是包含输入权重和偏置项的神经元特征, $\boldsymbol{x} = (\bar{\boldsymbol{x}}, 1) \in \mathbb{R}^{d+1}$ 是输入样本和标量 1 的拼接,$\boldsymbol{\theta}$ 是所有参数的集合, 即 $\{a_j, \boldsymbol{w}_j\}_{j=1}^m$。为简单起见, 我们称 $\boldsymbol{w}_j$ 为（输入）权重, 称 $\boldsymbol{x}$ 为输入样本。

接下来, 我们来看一下神经元的权重是怎么演化的。注意到, 同一层的神经元, 它们仅仅拥有不同的标号以及不同的初始值, 它们的动力学形式应该是完全一样的。也就是, 我们一旦得它们的统一的动力学演化方程后, 把它们的值代入方程就可以得到对应神经元演化的方向和速度。下面, 我们来推导一下它的演化方程, 这类似于推导梯度场下粒子群的运动方程。

目标函数由 $f^*(\boldsymbol{x})$ 表示。训练损失函数为均方误差

$$R(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} (f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i))^2. \tag{8.6}$$

在理论分析中, 不失一般性, 我们假设输出是一维的, 因为对于高维情况, 我们只需要直接求和不同分量即可。求和不会影响我们理论结果。我们考虑梯度流下的训练过程

$$\dot{\boldsymbol{\theta}} = -\nabla_{\boldsymbol{\theta}} R(\boldsymbol{\theta}). \tag{8.7}$$

输入权重 $\boldsymbol{w}_j := (w_j, b_j)$ 的方向场沿着梯度流动,

$$\dot{\boldsymbol{w}}_j = -\frac{a_j}{n}\sum_{i=1}^{n} e_i \sigma'(\boldsymbol{w}_j \cdot \boldsymbol{x}_i)\boldsymbol{x}_i,$$

其中 $e_i := f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)$。由于我们只关心 $\boldsymbol{w}_j$ 的方向, 且 a_j 在每个 epoch 为标量, 我们可以通过 $\dot{\boldsymbol{w}}_j/a_j$ 来可视化 $\dot{\boldsymbol{w}}_j$。为简洁起见, 如果无歧义, 我们不区分 $\dot{\boldsymbol{w}}_j/a_j$ 和 $\dot{\boldsymbol{w}}_j$。在计算不同 j 对应的 $\dot{\boldsymbol{w}}_j$ 时,$e_i\boldsymbol{x}_i$(对 $i = 1, \cdots, n$) 与 j 无关。因此, 在每个 epoch 中, 对于一组 $\{e_i, \boldsymbol{x}_i\}_{i=1}^n$, 我们可以考虑以下方向场

$$\dot{\boldsymbol{\omega}} = -\frac{1}{n}\sum_{i=1}^{n} e_i \boldsymbol{x}_i \sigma'(\boldsymbol{\omega} \cdot \boldsymbol{x}_i).$$

将 $\boldsymbol{\omega}$ 设为 $\boldsymbol{w}_j$, 我们可以得到 $\dot{\boldsymbol{w}}_j$。如图8.12所示, 我们画出了初始训练阶段的参数轨迹以及 0 附近的方向场。箭头是该点处方向场的方向。图的第一行的激活函数是 $\tanh(x)$，第二行的激活函数是 $x\tanh(x)$，第三行的激活函数是 $x^2\tanh(x)$，step 代表训练次数。在原点附近, 对于 $\tanh(x)$、$x\tanh(x)$ 和 $x^2\tanh(x)$, 该方向场分别有 1、2、3 条稳定线, 权重会保持其在这些线上的方向。我们还用绿点显示每个权重的值, 以及用橙色箭头显示相应的方向。从图中我们看到，神经元的输入权重凝聚到了特定的方向上。在这些凝聚的方向上，即使不同神经元离原点的距离不同，但是不同神经元所处位置的方向场的方向都相同，并且与凝聚的方向平行。

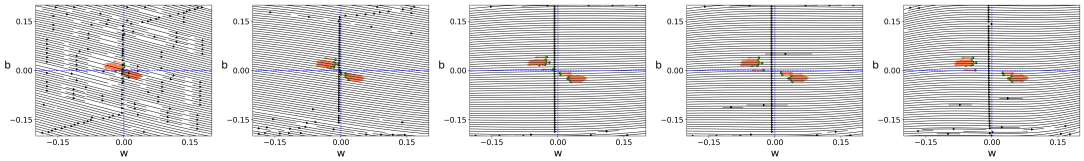
这个实验提示我们，初始凝聚现象或许可以被 **0** 附近的方向场解释，所以我们应该研究神经网络在参数为零的附近的动力学行为。最直观的想法是，我们把神经网络模型在参数为零的位置做泰勒展开，主导项 (leading-order term) 应该能非常好地近似真实模型函数。本质上，我们需要对激活函数进行泰勒展开。注意，这里的主导项并不一定是一阶线性项，比如如果激活函数是 $\tanh(x)$，那展开的主导项是线性的，但如果激活函数是 $x\tanh(x)$，其主导项就是二阶的。

为了刻画一个函数在零点处泰勒展开的主导项的阶数，我们定义一个称为“重数”（multiplicity）的概念。

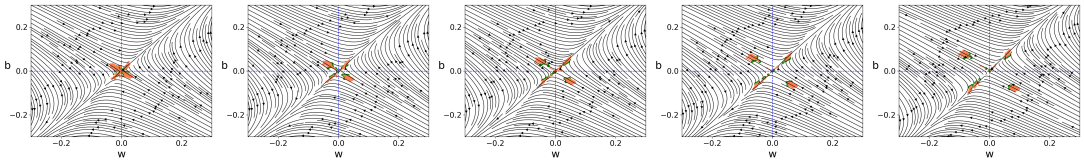
定义 2 (重数 p)**.** *假设 $\sigma(x)$ 满足以下条件, 存在 $p \in \mathbb{N}^*$, 使得 $\sigma(x)$ 在 $x = 0$ 处的 s 阶导数 $\sigma^{(s)}(0) = 0$, 当 $s = 1, 2, \cdots, p-1$, 且 $\sigma^{(p)}(0) \neq 0$, 则称 $\sigma(x)$ 的重数为 p。*

注 25. *列举一些例子。$\tanh(x)$、$\mathrm{sigmoid}(x)$ 和 $\mathrm{softplus}(x)$ 的重数为 $p = 1$。$x\tanh(x)$ 的重数为 $p = 2$。*

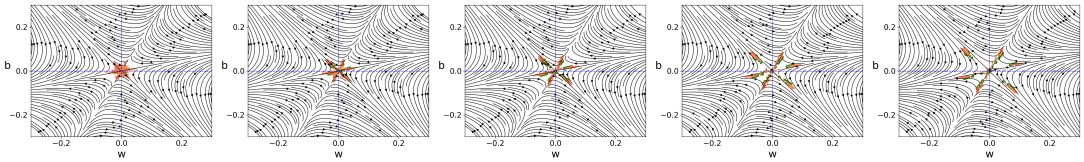
我们注意到，在图8.10、图8.11和图8.12中，激活函数为 $x^2\tanh(x)$ 时 ($p = 3$)，初始凝聚的方向有 6 个。激活函数为 $x\tanh(x)$ 时 ($p = 2$)，初始凝聚的方向有 4 个。而对于 $\tanh(x)$, sigmoid(x), softplus(x), $p = 1$，初始凝聚的方向有 2 个。因此，一个合理的推测是, 初始凝聚的方向数量是所采用的激活函数的重数的两倍。



(a) Step 40 (b) Step 80 (c) Step 120 (d) Step 160 (e) Step 200

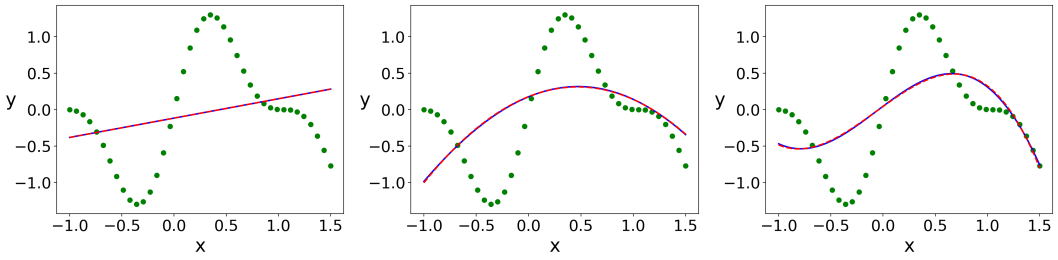


(f) Step 40 (g) Step 80 (h) Step 120 (i) Step 160 (j) Step 200



(k) Step 40 (l) Step 80 (m) Step 120 (n) Step 160 (o) Step 200

图 8.12: 动力学 (8.4.2) 中输入权重 $\boldsymbol{w} := (w, b)$ 的方向场, 所有设置与图8.13相同



(a) $\tanh(x)$ (b) $x\tanh(x)$ (c) $x^2\tanh(x)$

图 8.13: 采用激活函数 $\tanh(x)$、$x\tanh(x)$ 和 $x^2\tanh(x)$ 的两层神经网络在 epoch 1000 时的输出

另外，我们发现，初始阶段网络的复杂度与激活函数也有关系。图8.13是在如下设定下进行：训练集为 40 个在 x $[-1, 1.5]$ 上均匀采样的 $\sin(3x) + \sin(6x)/2$ 的点, 用绿点表示。蓝色虚线是神经网络在测试点上的输出, 而红色实线则分别是 (a,b,c) 中测试点的一阶、二阶和三阶多项式拟合。训练参数为 $n = 40$,$d = 1$,$m = 100$,$d_{\text{out}} = 1$,var $= 0.005^2$,lr $= 5\times10^{-4}$。由于参数幅

度很小, 重数为 p 的激活函数可以很好地被 p 阶多项式逼近, 因此神经网络输出也可以用 p 阶多项式逼近。如图8.13所示, 在训练初始阶段, 具有激活函数 $\tanh(x)$、$x\tanh(x)$ 和 $x^2\tanh(x)$ 的神经网络输出与线性、二次和三次多项式曲线的拟合线分别重叠得很好。这个实验表明，神经网络在初始训练阶段学习到的函数的复杂度与激活函数有关。

我们可以通过初始阶段学习到的函数，来理解初始凝聚。如果激活函数重数是 p，那么在初始阶段，网络会用 p 阶多项式来拟合数据点。当 p 越大时，求解梯度流的稳定点的方程的阶数也相应越大，因此其零点的数目也就越多。而这些零点就是初始凝聚的方向。

8.4.3 初始凝聚的静态分析**

在本节中, 我们将提出一个初步分析, 以了解激活函数的重数如何影响初始凝聚。我们递归地定义 L 层神经网络, 即将前一层的输出设为当前隐层的输入, 即

$$\begin{gathered}\boldsymbol{x}^{[0]}=(\boldsymbol{x},1),\quad \boldsymbol{x}^{[1]}=(\sigma(\boldsymbol{W}^{[1]}\boldsymbol{x}^{[0]}),1),\quad \boldsymbol{x}^{[l]}=(\sigma(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l-1]}),1),\ \text{for } l\in\{2,3,...,L\}\\ f(\boldsymbol{\theta},\boldsymbol{x})=\boldsymbol{a}^{\intercal}\boldsymbol{x}^{[L]}\triangleq f_{\boldsymbol{\theta}}(\boldsymbol{x}),\end{gathered}\tag{8.8}$$

其中 $\boldsymbol{W}^{[l]}=(\bar{\boldsymbol{W}}^{[l]},\boldsymbol{b}^{[l]})\in\mathbb{R}^{m_l\times(m_{l-1}+1)}$, 且 m_l 表示第 l 隐藏层的维度。为简洁起见, 我们也称 $\boldsymbol{W}^{[l]}$ 的每行为（输入）权重, 并将 $\boldsymbol{x}^{[l-1]}$ 称为第 l 隐藏层的输入。考虑具有 L 个隐藏层的网络, 我们使用行向量 $\boldsymbol{W}_j^{[k]}$ 表示从第 $k-1$ 层到第 k 层第 j 个神经元的权重。在初始训练阶段，由于参数的幅值很小，因此神经网络模型的函数可以被泰勒展开的主导项逼近。我们做以下假设:

假设 1. *小初始化意味着对所有 k 和 j 都有 $\boldsymbol{W}_j^{[k]}\boldsymbol{x}^{[k-1]}\sim o(1)$。*

假设 2. *在凝聚的初始阶段, 每个激活神经元的泰勒展开成立, 即*

$$\sigma'(\boldsymbol{W}_j^{[k]}\boldsymbol{x}^{[k-1]})=\sigma'(0)+\cdots+\frac{\sigma^{(\gamma)}(0)}{(\gamma-1)!}(\boldsymbol{W}_j^{[k]}\boldsymbol{x}^{[k-1]})^{\gamma-1}+O((\boldsymbol{W}_j^{[k]}\boldsymbol{x}^{[k-1]})^{\gamma})\tag{8.9}$$

假设激活函数的重数为 p，即对 $s=1,2,\cdots,p-1$, 有 $\sigma^{(s)}(0)=0$, 且 $\sigma^{(p)}(0)\neq 0$。则结合假设2, 我们有

$$\sigma'(\boldsymbol{W}_j^{[k]}\boldsymbol{x}^{[k-1]})\approx\frac{\sigma^{(p)}(0)}{(p-1)!}(\boldsymbol{W}_j^{[k]}\boldsymbol{x}^{[k-1]})^{p-1}.\tag{8.10}$$

对每个 k 和 j,$\boldsymbol{W}_j^{[k]}$ 满足以下动力学:

$$\dot{r}=\boldsymbol{u}\cdot\dot{\boldsymbol{w}},\quad \dot{\boldsymbol{u}}=\frac{\dot{\boldsymbol{w}}-(\dot{\boldsymbol{w}}\cdot\boldsymbol{u})\boldsymbol{u}}{r}.\tag{8.11}$$

其中 $\boldsymbol{w}$ 可以表示所有 k 和 j 的 $\boldsymbol{W}_j^{[k]\intercal}$,$r=\|\boldsymbol{w}\|_2$ 是振幅,$\boldsymbol{u}=\boldsymbol{w}/r$。为方便起见, 我们定义一个算子 $\mathcal{P}$, 满足 $\mathcal{P}\boldsymbol{w}:=\dot{\boldsymbol{w}}-\boldsymbol{u}(\dot{\boldsymbol{w}}\cdot\boldsymbol{u})$ 为了在理论分析中明确凝聚, 我们做以下定义:

凝聚: 权重演化向一个在方向场中不会改变的方向, 定义如下

$$\dot{\boldsymbol{u}} = 0 \;\Leftrightarrow\; \mathcal{P}\boldsymbol{w} := \dot{\boldsymbol{w}} - \boldsymbol{u}(\dot{\boldsymbol{w}} \cdot \boldsymbol{u}) = 0. \tag{8.12}$$

由于 $\dot{\boldsymbol{w}} \cdot \boldsymbol{u}$ 是一个标量,$\dot{\boldsymbol{w}}$ 与 $\boldsymbol{u}$ 平行。$\boldsymbol{u}$ 是一个单位向量, 因此, 我们有 $\boldsymbol{u} = \pm\dot{\boldsymbol{w}}/\|\dot{\boldsymbol{w}}\|_2$。

注 26 (参数凝聚的直观解释)**.** 在本节中, 我们考虑参数足够小的神经网络。假设 $r = \|\boldsymbol{w}\|_2 \sim O(\epsilon)$, 其中 ϵ 是一个小量, 则动力学(8.11)会显示 $O(\dot{r}) \sim O(\dot{\boldsymbol{w}})$ 和 $O(\dot{\boldsymbol{u}}) \sim O(\dot{r})/O(\epsilon)$。这里 $O(\dot{r}) \sim O(\dot{\boldsymbol{w}})$ 是指 $\dot{r}$ 的演化阶和 $\dot{\boldsymbol{w}}$ 的每个分量的阶相同, 对 $O(\dot{\boldsymbol{u}}) \sim O(\dot{r})/O(\epsilon)$ 也是如此。因此, 方向 $\boldsymbol{u}$ 的变化会比振幅 r 快得多。通过 $\boldsymbol{w}$ 的动力学和重数为 p 的 *Taylor* 逼近(8.10), 很容易看出式(8.12) 的解是有限的。综合起来, 方向 $\boldsymbol{u}$ 会快速收敛到某些方向, 导致凝聚。

我们细致地研究了 (i) $p = 1$ 和 (ii) $m_{k-1} = 1$(第 k 层的输入维度等于 1) 的情况, 并得到以下非正式命题:

命题 1. 假设*1* 和 *2*成立。考虑(8.12)在初始化趋于零时的主要项泰勒展开。如果 $p = 1$ 或 $m_{k-1} = 1$ 成立, 则(8.12)的根的最大数量是激活函数的重数的两倍。

具体的证明可以在文献Zhou, Zhou, Luo, Zhang & Xu (2022) 找到。

8.4.4 初始凝聚的动态分析**

以上关于初始凝聚的分析是一种静态的分析，即我们的分析是针对在某一时刻进行泰勒展开得到，而我们最终期望从参数的动力学出发得到关于参数凝聚的动态分析，Chen et al. (2023) 给出了一类激活函数下两层神经网络的动态分析。

梯度流的动态分析

我们使用与 Luo, Xu, Ma & Zhang (2021) 几乎相同的设置, 从原始模型开始

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) := \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^{\intercal}\boldsymbol{x}), \tag{8.13}$$

其中 $\boldsymbol{x} \in \mathbb{R}^d$, 且其参数 $\boldsymbol{\theta}^0 := \mathrm{vec}(\boldsymbol{\theta}_a^0, \boldsymbol{\theta}_{\boldsymbol{w}}^0)$ 被初始化为

$$a_k^0 \sim \mathcal{N}(0, \nu^2), \quad \boldsymbol{w}_k^0 \sim \mathcal{N}(\boldsymbol{0}, \varepsilon^2 \boldsymbol{I}_d), \tag{8.14}$$

损失函数为

$$R_S(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\right)^2. \tag{8.15}$$

则基于梯度下降 (GD) 的连续极限处的训练动力学精确地遵循以下梯度流: 对 $k \in [m]$,

$$\begin{aligned}\frac{\mathrm{d}a_k}{\mathrm{d}t} &= -\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k'=1}^{m} a_{k'}\sigma(\boldsymbol{w}_{k'}^{\intercal}\boldsymbol{x}_i) - y_i\right)\sigma(\boldsymbol{w}_k^{\intercal}\boldsymbol{x}_i),\\ \frac{\mathrm{d}\boldsymbol{w}_k}{\mathrm{d}t} &= -\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k'=1}^{m} a_{k'}\sigma(\boldsymbol{w}_{k'}^{\intercal}\boldsymbol{x}_i) - y_i\right)a_k\sigma^{(1)}(\boldsymbol{w}_k^{\intercal}\boldsymbol{x}_i)\boldsymbol{x}_i.\end{aligned} \tag{8.16}$$

为了证明定理6，我们需要以下主要假设:

- 激活函数重数为 1: $\sigma(0) = 0$, $\sigma^{(1)}(0) = 1$
- 数据非退化: $\frac{1}{n}\sum_{i=1}^{n} y_i\boldsymbol{x}_i \neq \boldsymbol{0}$
- 以下极限存在: $\gamma_1 := \lim\limits_{m\to\infty} -\frac{\log\nu}{\log m}$, $\quad \gamma_2 := \lim\limits_{m\to\infty} -\frac{\log\varepsilon}{\log m}$

这里我们省略了一些其他的较为温和的假设。由定义有,

$$\gamma = \lim_{m\to\infty} -\frac{\log\nu\varepsilon}{\log m} = \gamma_1 + \gamma_2, \quad \gamma' = \lim_{m\to\infty} -\frac{\log\frac{\nu}{\varepsilon}}{\log m} = \gamma_1 - \gamma_2$$

此外，我们定义 $\boldsymbol{z} := \frac{1}{n}\sum_{i=1}^{n} y_i\boldsymbol{x}_i$, $\hat{\boldsymbol{z}} := \frac{\sum_{i=1}^{n} y_i\boldsymbol{x}_i}{\|\sum_{i=1}^{n} y_i\boldsymbol{x}_i\|_2}$。在上述假设下，我们有以下关于初始凝聚的定理:

定理 6 (凝聚)**.** *对任意 $\delta \in (0,1)$, 如果 $\gamma > 1$ 且 $\gamma' < \gamma - 1$, 则以概率至少 $1-\delta$ 取 $\boldsymbol{\theta}^0$ 时, 存在 $T > 0$, 使得*

$$\lim_{m\to\infty}\sup_{t\in[0,T]}\frac{\|\boldsymbol{\theta}_{\boldsymbol{w}}(t) - \boldsymbol{\theta}_{\boldsymbol{w}}(0)\|_2}{\|\boldsymbol{\theta}_{\boldsymbol{w}}(0)\|_2} = +\infty, \tag{8.17}$$

且

$$\lim_{m\to\infty}\sup_{t\in[0,T]}\frac{\|\boldsymbol{\theta}_{\boldsymbol{w},\boldsymbol{z}}(t)\|_2}{\|\boldsymbol{\theta}_{\boldsymbol{w}}(t)\|_2} = 1, \tag{8.18}$$

其中 $\boldsymbol{\theta}_{\boldsymbol{w},\boldsymbol{z}}(t) := [\langle\boldsymbol{w}_1,\hat{\boldsymbol{z}}\rangle, \langle\boldsymbol{w}_2,\hat{\boldsymbol{z}}\rangle, \cdots, \langle\boldsymbol{w}_m,\hat{\boldsymbol{z}}\rangle]^{\intercal}$。

在定理6中，(8.17)说明两层神经网络在发生凝聚现象时参数与初始时的相对距离非常大，(8.18)则说明在极限状态下，参数会凝聚到 $\hat{\boldsymbol{z}}$ 的方向上。具体证明过程可见 Chen et al. (2023)。

初始凝聚的时间

我们另一个关心的问题是，初始凝聚需要多长时间发生。我们将相图分析中的凝聚区域细分为了两个子区域，分别是 w-lag 和 a-lag，如图8.14所示。这两个区域中，凝聚现象发生的时间有差异。用 T 表示初始凝聚发生的时间，用 m 表示网络宽度。关于 T 有以下理论估计：

- w-lag 区域中，$T \geq \log(\frac{1}{4}) + \frac{\gamma - \gamma' - 1}{8} \log(m)$
- a-lag 区域中，存在 $\alpha > 0$，使得 $m^{-\alpha} \leq T \leq 2, m^{-\alpha}$

其中 γ 和 γ' 分别是相图中的横、纵坐标。在 w-lag 区域中，当 $m \to \infty$ 时，$T \to \infty$，因此发生凝聚所需的时间会长一些。在 a-lag 区域中，当 $m \to \infty$ 时，$T \to 0$，因此凝聚会在训练的初始阶段观察到。我们也在实验上测量了凝聚现象发生的时间（见图8.15和图8.16，其中蓝点是两层 tanh 神经网络的实验结果。横坐标是神经网络的宽度（对数尺度），纵坐标是凝聚发生的时间。(a)、(b)、(c) 中 γ 取不同值，但 γ' 固定为 0。黑线是线性拟合的结果，R^2 是决定系数，提供了线性回归拟合优度的信息。R^2 越接近 1，模型对数据的拟合就越好。）。可以看到，实验结果与理论预测完全吻合。

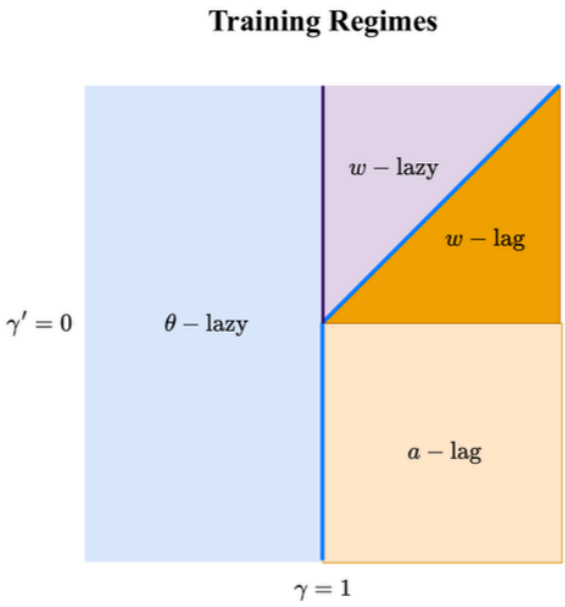
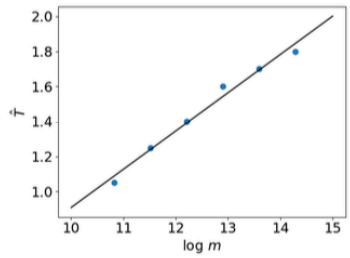
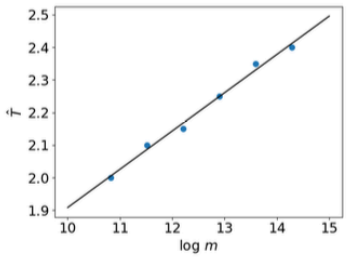


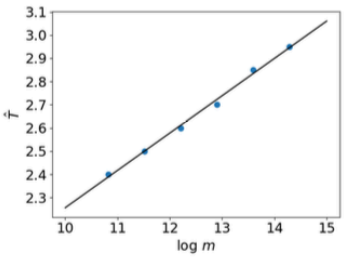
图 8.14: 两层 Tanh 神经网络的相图。$\boldsymbol{\theta}$-lazy 和 w-lazy 含于线性区域；w-lag 和 a-lag 含于凝聚区域



(a) $\gamma = 1.4,\quad R^2 = 0.983$

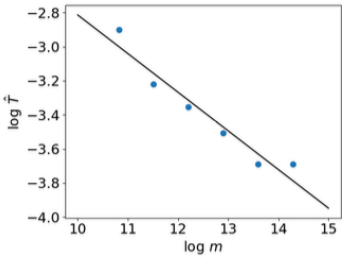


(b) $\gamma = 1.6,\quad R^2 = 0.991$

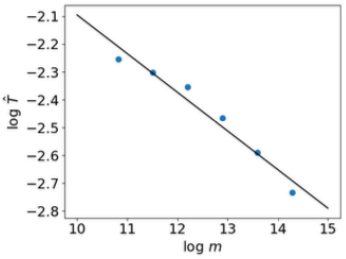


(c) $\gamma = 1.8,\quad R^2 = 0.995$

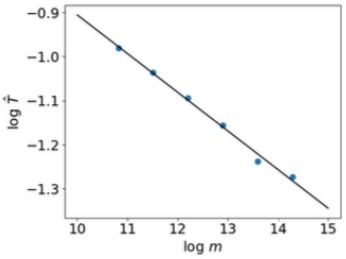
图 8.15: w-lag 区域实验结果



(a) $\gamma = 1.2,\quad R^2 = 0.936$



(b) $\gamma = 1.4,\quad R^2 = 0.977$



(c) $\gamma = 1.6,\quad R^2 = 0.993$

图 8.16: a-lag 区域实验结果

8.5 习题

1. 什么是凝聚现象?

2. 对初始化很小的网络，考虑训练初始阶段，在分析的技巧上有什么好处?

3. 凝聚现象的实验中，为什么网络参数的初始化要使用小初始化?

4. 对于一个 m 宽的两层 ReLU 神经网络 $f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \mathrm{ReLU}(w_k x + b_k)$，其中 $\boldsymbol{\theta} = (a_k, w_k, b_k)_{k=1}^{n}, \mathrm{ReLU}(x) = max\{0, x\}$。用该网络来拟合 n 个数据点 $(x_k, y_k)_{k=1}^{n}$。请问为什么在训练过程中，图8.3中的灰色神经元的权重不改变? 请证明它。

5. 我们有一个宽为 m 的两层 tanh 神经网络 $f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \tanh(w_k x + b_k)$，其中 $\boldsymbol{\theta} = (a_k, w_k, b_k)_{k=1}^{m}, \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$。用该网络来拟合 n 个数据点 $(x_k, y_k)_{k=1}^{n}$，其中 $x_k, y_k \in \mathbb{R}$。假设网络的宽度 m 很大，a_k, w_k, b_k 的初始化为均值为 0 的高斯分布，标准差为 $\frac{1}{m^5}$。请问：

   (i)：为什么在训练的初始阶段，神经网络会学到线性函数?

   (ii)：学到这个线性函数的过程中，是否有凝聚发生? 如果有，它跟我们所说的初始凝聚在机制上有什么不同?

6. 对于一个 m 宽的两层 ReLU 神经网络 $f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \mathrm{ReLU}(w_k x + b_k)$，其中 $\boldsymbol{\theta} = (a_k, w_k, b_k)_{k=1}^{n}, \mathrm{ReLU}(x) = max\{0, x\}$。用该网络来拟合一个数据点 (x_0, y_0)，其中 $x_0, y_0 \in \mathbb{R}$。a_k, w_k, b_k 的初始化采用小初始化。请完成：

   (i)：推导凝聚现象发生的原因

   (ii)：神经网络最终学到的函数是什么?

7. 对于一个 m 宽的两层 ReLU 神经网络 $f_{\boldsymbol{\theta},x} = \sum_{k=1}^{m} a_k \mathrm{ReLU}(w_k x + b_k)$，其中 $\boldsymbol{\theta} = (a_k, w_k, b_k)_{k=1}^{n}, \mathrm{ReLU}(x) = max\{0, x\}$。用该网络来拟合两个数据点 $\{(1, 4), (-2, 2)\}$。其中 a_k, w_k, b_k 的初始化采用小初始化。请问：

   (i)：神经网络最终学到的函数是什么? 请做实验来探究一下。

   (ii)：给出（i）中的实验结果的数学证明。

Chapter 9

损失景观的嵌入原则

在深入探究神经网络的复杂性和学习过程时，我们遇到了一个引人入胜的现象——**神经元的凝聚**。这意味着一层内不同的神经元在训练过程中有趋同的倾向。正如前一章所讨论的，这种凝聚现象揭示了神经元在非线性训练过程中所表现出的集体行为模式。这一发现为我们提供了一个窗口，可以引领我们对一系列深刻的问题进行探讨。

首先，我们注意到在神经网络训练过程中，由于凝聚现象的发生，有效的神经元数量常常远低于网络的总规模。这引发了一个直观的疑问：如果小规模网络足以捕捉目标函数的复杂性，为何我们不直接训练一个更小的模型呢？在大网络和小网络都能够充分表达目标函数的前提下，它们之间存在哪些本质的共同点和差异？此外，为何在训练过程中，我们总能观察到凝聚方向数量的单调增长？

通过对神经网络非线性动力学的实验研究，我们进一步发现，凝聚现象通常与损失函数的**停滞现象**相伴随。损失函数的停滞指的是在训练的某些阶段，损失值下降变得异常缓慢，这通常意味着训练轨迹出现在临界点附近。所谓临界点是指梯度为零的点，也叫驻点，包括鞍点、局部极小点和全局极小点等。

在一个两层 ReLU 神经网络的一维函数拟合问题中，当我们将不同宽度的网络训练所得的损失曲线并置对比时，便观察到了一种令人惊奇的规律。如图 9.1所示，我们发现在小初始化条件下，尽管网络宽度 m 不同，它们的损失值却在几乎相同的点上发生了停滞，揭示出一种**宽度相似性**。这种现象启发我们去思考：不同宽度网络的临界点之间，可能存在着某种本质上的联系。

然而，我们必须认识到一个关键问题：不同宽度的网络其参数维度亦不同。在这些不同维度的空间中对比临界点，便是我们面临的挑战。如何在高维参数空间中寻找和定义这些网络之间的相似性？具体而言，图 9.1是否意味着不同宽度网络中始终存在损失值相同的临界点？

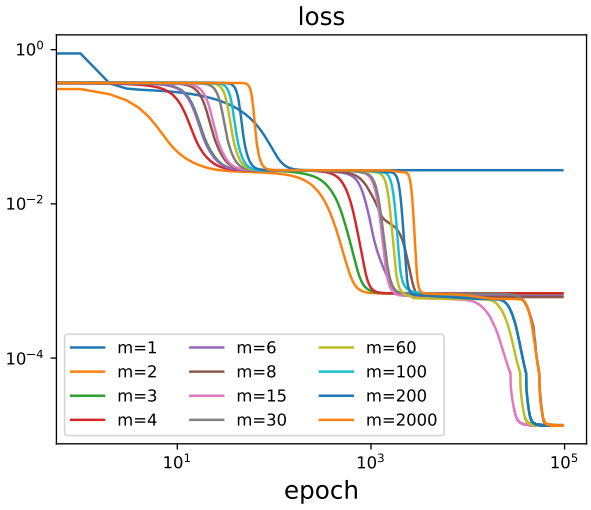


图 9.1: 不同宽度全连接神经网络的损失曲线图

在接下来的部分中，我们将深入探讨并试图解答上述问题。在 9.1节中，我们将从宽度相似性的现象出发，探索网络宽度如何影响学习动力学和损失景观的结构，建立损失景观的嵌入原则。在 9.2节中，我们将深入分析嵌入原则如何影响网络的学习过程和泛化性能，以及探讨网络规模简化的方法。在 9.3节中，我们将对嵌入原则做进一步的扩展与推广，建立临界函数嵌入和神经网络深度上的嵌入原则。

科普篇

什么是神经网络的嵌入原则？

神经网络的嵌入原则是指一个网络的损失景观会“包含”所有比它更窄的网络的临界点。这里的临界点是指梯度为零的点，会对训练过程产生显著影响。这里的“包含”并非传统意义上的几何包含，而是一种函数上的包含，即通过特定的嵌入操作，任何较窄网络的临界点都可以被嵌入到较宽网络的损失景观中，同时保持输出函数不变。

嵌入原则如何运作？

- **实验验证：**在实验中，我们使用不同宽度的神经网络进行训练时发现，它们的训练损失曲线表现出一致性。具体来说，较宽网络的损失曲线会在训练过程中停滞在与较窄网络

相同的值上，同时输出函数保持一致。这表明宽网络经历了由窄网络嵌入的临界点。

- **理论支持：**嵌入原则的理论支持依赖于神经元的“分裂”操作。通过分裂操作，可以将一个网络嵌入到更宽的网络中，同时保持输出函数和临界点的特性。具体来说，可以将一个隐藏层神经元分裂为两个，新神经元的输入权重与原神经元相同，而输出权重则按比例分配。这种操作逐步地将较窄的网络嵌入到较宽的网络中，保持其输出一致性和临界点的极值特性。

为什么嵌入原则重要？

- **揭示内在联系：**嵌入原则揭示了不同宽度神经网络之间的内在关系，帮助我们理解神经网络结构的关键特性。它从理论上说明，通过适当的嵌入操作，较窄网络的临界点可以在较宽网络的损失景观中存在。这个原则具有高度的一般性，其不依赖于具体的训练数据或损失函数选择，而是与网络结构本身的层结构密切相关。

- **描绘简单偏好：**嵌入原则与频率原则和凝聚现象密切相关，共同描绘了神经网络的偏好简单模式。相比于频率原则，其更加细致，刻画了训练过程中遇到的临界点等行为；相比于凝聚现象，嵌入原则刻画了一种理想的凝聚状态，虽然理想状态在实验中不一定完全达到，但可以在训练过程中通过凝聚趋近于这样的状态。

- **拓展到深度上：**嵌入原则可以推广到深度上，即更深的神经网络的损失景观也包含了所有较浅网络的临界点。实验上表明，即使是非常深的网络，在训练过程中也会停滞在与较浅网络相同的损失值上。这表明了深网络经历了由浅网络嵌入的临界点。深度上的嵌入原则与之前介绍的宽度上的嵌入原则相结合，提供了关于神经网络损失景观内在层次结构的完整图景。这一图景有力地支撑了实验中观察到的不同规模大小神经网络之间在训练和泛化方面的相似性。

嵌入原则有什么用？

嵌入原则的应用有多方面：

1. **优化训练过程：**嵌入原则帮助我们理解较宽网络在训练过程中如何经历与较窄网络的全局最小点具有相同输出的鞍点，以及鞍点处 Hessian 矩阵的特性。这种理解可以优化训练过程，提升模型性能。

2. **模型压缩和凝聚现象：**嵌入原则和网络的凝聚现象密切相关，即大网络可以压缩到小网络而不改变其输出函数。这对于模型压缩和高效部署深度学习模型具有指导意义。

3. **多种网络架构的通用性：**嵌入原则适用于多种常见的神经网络架构，如卷积神经网络（CNN）、Transformer、ResNet 等，具有高度的普遍性。

9.1 宽度相似性与嵌入原则

9.1.1 损失停滞点的结构相似性

在观察到图 9.1所揭示的停滞现象后，我们不禁产生了一个疑问：不同宽度的网络在损失停滞点具有相同的损失值，这是否预示着它们在结构层面上存在着某种程度的相似性?

为了探索这一问题，我们将深入研究不同宽度神经网络在特定损失停滞点的**输出行为**和**内部表示**。对于输出行为，我们通过比较不同宽度网络在同一停滞点的输出函数，来揭示网络的外在表现。如图 9.2(a) 所示，我们选定了一个特定的损失值——宽度为 2 的神经网络在损失函数中的最小值，然后比较了不同宽度网络在此损失值处对应的输出函数。

对于内部表示，我们可以用和之前章节中类似的方法来可视化。具体来说，对于一维输入的两层神经网络，如公式 (9.1) 所示:

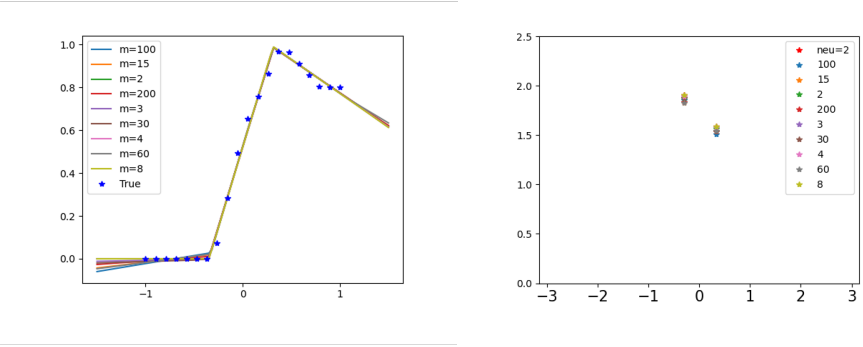
$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j^\top \boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(w_j x + b_j), \tag{9.1}$$

我们可以通过考察每个神经元的参数对 $(a_k, \boldsymbol{w}_k)$ 来探究网络的内部表示。特别是，对于 ReLU 激活函数，我们将每个神经元的参数分解为单位方向特征 $\hat{\boldsymbol{w}} = \boldsymbol{w}/\|\boldsymbol{w}\|_2$ 和代表其对输出贡献的振幅 $A = |a|\|\boldsymbol{w}\|_2$，即 $(A, \hat{\boldsymbol{w}})$。由于输入是一维的，$\boldsymbol{w}$ 实际上是包含偏置项的二维向量。因此，我们采用每个 $\hat{\boldsymbol{w}}$ 相对于 x 轴的 $[-\pi, \pi)$ 范围内的角度来描述其方向，用 A 表示该方向上的振幅。基于这一表示方法，我们可以将方向相同的神经元的振幅累加，从而形成一个 “等效神经元”，同时忽略那些振幅接近零的等效神经元。如图 9.2(b) 所示，这样的可视化揭示了网络内部结构的集中趋势。

通过对图 9.2的分析，我们揭示了两个关键发现：

(i) 不同宽度的网络在特定损失值的停滞点达到了**输出函数的高度一致性**。如图 9.2(a) 所示，不同宽度网络的输出函数几乎完美重合，这表明损失值相同的停滞点不仅仅是表面的巧合，而是在输出函数层面展现出了深层次的一致性。

(ii) 独立于网络宽度，所有网络在这些特定的损失值停滞点均显示出了内部参数的显著**凝聚性**。如图 9.2(b) 所展示的，不同宽度网络的神经元在两个特定方向上凝聚，并且它们在这些方向上的振幅也显示出惊人的相似性。这种凝聚性表明，在训练的某些阶段，即便是宽度较大的网络，其有效的输出表现也可以通过一种内部的参数简化过程来实现，从而呈现出**低复杂度**的特性。



(a) 网络的输出函数　　(b) 网络的内部表示

图 9.2: 不同宽度的神经网络在宽度为 2 的神经网络最小值点处所对应的输出函数和参数内部表示图

这些观察结果表明，宽神经网络可能与窄神经网络存在输出函数一致的临界点，而这些临界点通常具有低复杂度的特性。临界点是损失景观中非常重要的结构，它们在优化方面有着重要研究价值：我们通常希望训练过程能够避开局部最小点，找到全局最小点，并且在众多可能的全局最小点中，找到那些具有最佳泛化能力的解。此外，临界点在训练过程中也起着核心作用，例如鞍点——损失景观中的一种关键结构——尽管可能会使训练过程在其附近停滞，降低训练效率，但它们在某些维度上的极小性质也能吸引训练轨迹，在一定程度上起到了导航的作用，引导着训练轨迹向着损失函数更低的方向移动，从而在更宏观的层面上对网络的学习路径和最终的泛化能力产生积极的影响。

此外，图 9.2(b) 表明这些输出函数一致的临界点与我们之前关于凝聚现象的讨论遥相呼应：凝聚现象指出在适当的条件下，即使是宽度较大的网络，也能通过某种内在的凝聚机制，在输出行为上近似于更小规模的网络。这些发现促使我们进一步深入挖掘这种相互关联的本质。

9.1.2 理论框架：嵌入原则

当我们将这些发现放在损失景观的宏观视角下考虑时，就可以得到一个深度神经网络中的嵌入原则 (详细证明证明参见 Zhang, Zhang, Luo & Xu (2021))：

嵌入原则：一个神经网络的损失景观中会“包含”所有比它更窄的网络的临界点。

这里“更窄的网络”指的是深度相同但每层宽度均不大于目标神经网络的网络。这里的“包含”并非指传统意义上的几何包含，因为不同宽度的神经网络的参数空间是不同的。然而，这种包含关系在某种意义上是合理的，因为通过一类特定的嵌入操作，任何较窄网络的临界点

都可以被嵌入到较宽网络的损失景观中，同时保持输出函数不变。这种特殊的嵌入操作，我们称之为**临界嵌入**（critical embedding），因为它保持了网络在临界点的特性。

我们将这一概念称为“原则”，是因为它表达了深度神经网络的一种内在属性，这种属性不依赖于具体的训练数据或损失函数的选择，而是与网络的**层结构**紧密相关。此外，后面的实验结果显示，这一原则与深度神经网络的实际训练过程紧密相连：在实际训练中，较宽的网络会经历一些与较窄网络的全局最小点具有相同输出的鞍点，这可以视作较窄网络的全局最小点在较宽网络中的嵌入变体。

在我们正式证明嵌入原则之前，让我们先直观地了解证明的关键要素——神经元的“分裂”操作。通过分裂操作，我们可以将一个网络嵌入到一个更宽的网络中，同时在保持输出函数不变的前提下，也保持了临界点的特性。

我们首先考虑一步嵌入。如图 9.3所示，我们可以将任何隐藏层神经元（比如图 9.3左侧网络中的黑色神经元）分裂为两个神经元（图 9.3右侧网络中的蓝色和紫色神经元），这两个新神经元的输入权重与原神经元相同，而输出权重则按比例 α 和 $(1-\alpha)$（$\alpha \in \mathbb{R}$ 为一个超参数）分配。多步嵌入是多个一步嵌入的组合。由于每个一步嵌入可以向选择的层添加一个神经元，因此通过多步嵌入，任何较窄的网络都可以被嵌入到任何宽度更大的网络中，且这一过程不仅保持了输出函数的一致性，还保持了临界点的极值特性。

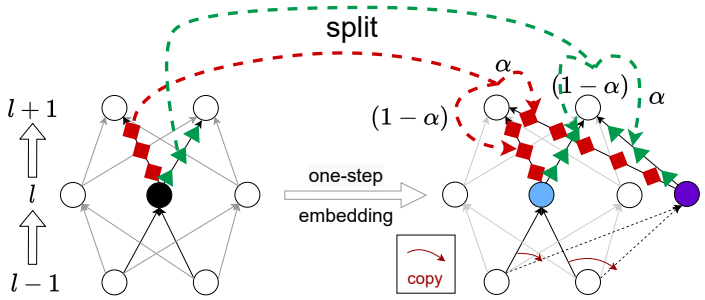


图 9.3: 一步嵌入的示意图

实际上，分裂操作并不仅限于全连接神经网络，具有层结构并且神经元指标具有可交换性的网络都可以通过分裂操作证明相应的嵌入原则，从而该原则可以推广并应用于多种网络架构中。在卷积神经网络（CNN）中，分裂操作可以被理解为对卷积核的复制和变换，从而在保持感受野和功能性不变的情况下增加网络的宽度。对于 Transformer 架构，分裂操作可以类比为增加注意力头的数量，这样做可以扩展模型的能力，让它在处理复杂的序列关系时表现得更为多样化。除了分裂操作，确实也存在其他方式来实现网络的嵌入。在后续的内容中，我们将探讨更多的嵌入操作，这些操作不仅在理论上扩展了我们对神经网络嵌入原则的理解，也为实际应用中网络设计和训练提供了新的视角和工具。

9.1.3 嵌入原则和凝聚现象的关系

上面提出的分裂操作和神经网络训练中的凝聚现象高度相关。凝聚现象描述了训练过程中神经元的趋同，特别地，不同神经元在学习过程中可能会发展出相似或一致的权重方向。这种现象与图 9.3所示的分裂操作直接相关，分裂操作意味着通过复制一个神经元，我们可以得到两个具有完全相同输入权重的神经元。

嵌入原则为我们理解神经网络结构的关键特性提供了一个强有力的理论框架。它从存在性的角度阐述了不同宽度的神经网络如何通过临界嵌入操作相互关联。而凝聚现象，从实验的视角，展现了这些理论关系在实际训练中的具体体现。这种理论与实践的结合揭示了一个深刻的洞见：嵌入原则不仅是一个高层次的理论概念，它还能够与神经网络的实际训练过程形成紧密的联系。这种联系为我们提供了宝贵的洞察力，帮助我们更好地理解和优化深度学习模型的训练动态。

此外，从嵌入原则的角度出发，我们可以对凝聚提供进一步的理解。如果一个较大的网络可以通过对一个较小的网络进行临界嵌入操作而生成，那么我们可以认为这个较大的网络处于一种**理想凝聚状态**。虽然在实验中难以完全达到，但是神经网络在训练过程中可以经过凝聚趋近于这样的状态。在这种理解下，凝聚不仅意味着神经元的趋同性，而且还意味着网络的可压缩性。处于理想凝聚状态的大网络可以被压缩到较小的网络而不改变其输出函数。嵌入原则的发现为建立凝聚现象的理论理解提供了新的思路：可以通过发掘更多的临界嵌入操作，增进对理想凝聚状态的认识，以帮助设计更好的指标在实验中识别凝聚的发生。

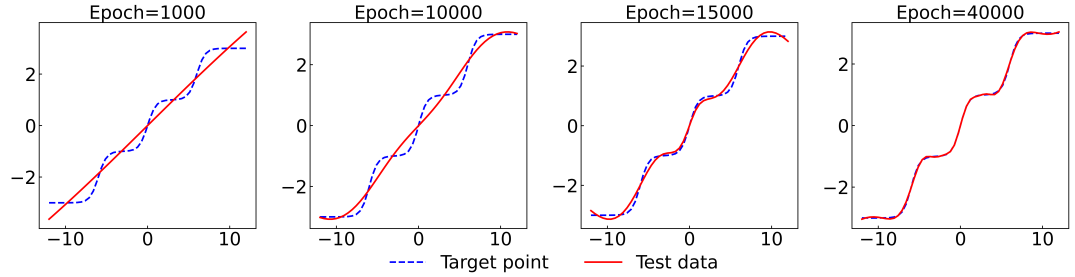
9.1.4 嵌入原则和频率原则的关系

我们已经探讨了两个深刻的神经网络原则——频率原则和嵌入原则。频率原则阐述了神经网络在学习过程中倾向于首先捕捉目标函数的低频分量，随后逐渐学习高频细节。这一原则体现了神经网络的频谱学习动态，即从简单（低频）模式开始，逐步过渡到复杂（高频）模式。而嵌入原则则从参数空间的角度，揭示了宽度不同的神经网络之间临界点的继承关系，说明了一个较窄的网络可以在其损失景观中找到的临界点，同样可以在更宽网络的损失景观中找到对应的临界点。

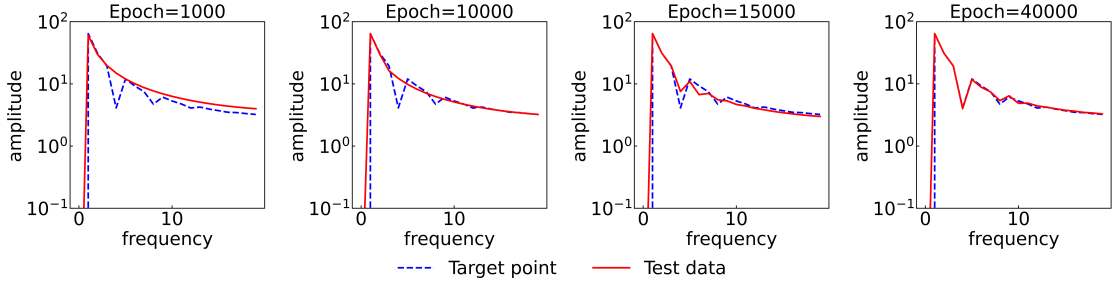
这两个原则都具有高度的普遍性，它们不特定于某种数据分布，且适用于包括卷积神经网络（CNN）、Transformer、ResNet 等在内的多种常见架构。为了深入理解这两个原则之间的联系，我们进行了一个实验，使用一个两层、宽度为 500、以 tanh 为激活函数的神经网络来拟合一个一维目标函数（如图 9.4(b) 中的蓝色虚线所示）。

我们记录了在不同训练时期（Epoch）的输出函数及其频率谱。如图 9.4所展示的，我们从输出函数（图 9.4(a)）和频率域（图 9.4(b)）两个视角观察了神经网络的学习过程。从图 9.4(b) 可以看出，在训练初期，神经网络的输出在低频区域与目标函数有较好的吻合，随着训练的进

展，高频部分逐渐与目标函数对齐。在特定的 Epoch，如图 9.4(a) 所示，网络学习到的函数逐步演进，直至在 Epoch=10000 时，网络先是学到了一个与单个 tanh 神经元对应的输出函数（图 9.4(a) 从左数第二张图），这实际上是一个较窄网络的全局极小值点，而这个点可以通过嵌入原则被嵌入到更宽网络的临界点中。最终，在 Epoch=40000 时，网络成功地拟合了目标函数（对应 3 个 tanh 神经元的全局极小点）。



(a) 输出函数视角



(b) 频率视角

图 9.4: 两层、宽度为 500 的 tanh 神经网络拟合一维函数时在输出函数和频率两种视角下神经网络的学习过程图

这个实验不仅展示了神经网络如何逐步逼近复杂函数的过程，也体现了嵌入原则在实际训练中的作用：宽网络在训练过程中可能会经历那些由窄网络嵌入而来的临界点，从而控制其输出函数的复杂度。根据频率原则，我们将那些由低频分量主导的函数视为训练网络时隐式倾向于选择的“简单”函数。而在这里，“简单”函数通过嵌入原则得到定义，即那些属于较窄网络的临界点对应的函数。神经网络这种先学到由窄网络嵌入的临界函数再学到由更宽网络嵌入的临界函数现象，与频率原则中的从低频到高频的学习过程是一致的。

9.2 嵌入原则的深入分析

9.2.1 损失函数停滞现象的频谱分析

嵌入原则提供了一个理论框架，为我们揭示了不同宽度的神经网络在理论上可以共享输出函数的临界点。通过图 9.1和图 9.2，我们观察到宽网络在实际训练过程中确实可能经历这些窄网络所定义的临界点，这引发了对该现象成因及其普遍性的进一步探讨。特别是，我们想要了解这种宽窄网络之间停滞点的相似性是普遍存在的，还是仅仅因为特定实验设置的结果。

为了探讨宽网络在与窄网络在相同损失值的临界点停滞的普遍性，我们采取了一种系统的实验方法。具体来说，我们对每个宽度的神经网络进行了 100 次独立的实验，每次实验都采用不同随机种子下的初始化。这样的重复实验设计旨在排除随机性对结果的影响，确保我们观察到的现象是由网络结构本身的特性所决定的，而非偶然因素的结果。根据上一章中对相图分析的讨论，我们这里选择了一种特殊的参数初始化策略，确保所有网络的参数初始化都位于凝聚区域的相同位置。具体来说，这意味着网络的初始状态被设置为具有相似的动力学性质，从而确保每个网络都从非线性训练区域起步。这种初始化方法的选择是基于对凝聚现象的先前观察，以及对动力学系统行为的理解。

在每次实验中，我们密切关注损失函数的演化轨迹，特别是记录损失函数在各个小区间内停留的步数。如果损失函数在某个小区间内停留了异常多的步数，这将在损失函数的演化图中形成一个明显的“平台”，正如图 9.1所示。通过将不同实验中训练损失函数进行合并统计，我们构建了一个损失值分布图，如图 9.5所展示。这个分布图的每一行代表一个特定宽度的网络对应的损失值分布，其中颜色的深浅表示损失值落在特定区间的频率的对数值的大小。

图 9.5的形式类似于物理学中的能谱图，它能够为我们提供损失函数的关键信息。例如，能谱图中的谱线离散性和有限性能够帮助我们分析损失函数在哪些特定值上发生停滞，以及不同宽度网络之间的相互关系。在这个能谱图中，纵坐标代表网络宽度的变化，横坐标则是 $\log_{10}(\text{loss})$。我们在区间 $[-5, 0]$ 上等距取 200 个小区间，并记录每个宽度下 100 次实验中损失值落在每个区间的频数 p_i。这样，我们得到了集合 $\{p_i\}_{i=1}^{200}$，即在 100 次实验中损失值在每个取值区间的经历次数。为了在图中清晰地展示频数较小的谱线，我们采用了对数尺度来设置颜色，即使用 $\log_{10} \frac{p_i}{\sum_{i=1}^{200} p_i}$ 来表示颜色条的强度。直观上，这意味着在 100 次实验中频繁出现的区间（即损失函数停滞现象发生的区间）会在图中以更亮的颜色标出。

统计分析的结果揭示了一些关键的发现，这些发现增强了我们对不同宽度网络训练过程中损失函数停滞现象的理解：

(i) 对于特定宽度的网络，不同初始化随机种子下的损失函数展现了显著的一致性。具体而言，不同实验的损失函数倾向于在特定的损失值上停留较长时间（如图 9.5所示，同一行中的特定损失值呈现高亮）。这种一致性表明，尽管初始化条件各异，网络的训练动力学

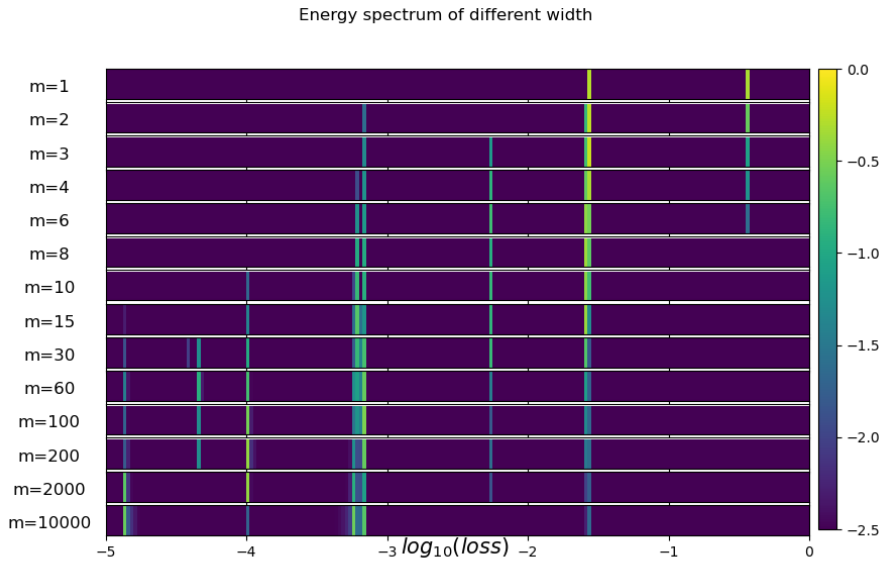


图 9.5: 不同宽的两层神经网络，100 次训练得到的损失谱图

在某种程度上是可预测的，而且某些损失值似乎是训练过程中的关键“吸引子”。

(ii) 不同宽度的网络在损失函数的停滞模式上显示出相似之处（图 9.5中不同行的高亮位置相似），但宽网络未必经历所有窄网络嵌入的临界点（图 9.5中显示了一些窄网络高亮而宽网络未高亮的情况）。因此，尽管网络的规模不同，宽网络确实会经历窄网络中嵌入的临界点。但具体经历哪些临界点，以及为何经历依然值得我们继续探索。

(iii) 随着网络宽度的增加，损失函数在较小数值上停滞的概率也越大。这一点从图 9.5中可以明显看出，宽度较大的网络在训练结束时的损失值更小。这个观察结果揭示了宽网络可能在优化过程中具有固有的优势，暗示了更宽的网络可能更容易逃离鞍点，以达到更小的损失函数值。

这些观察结果为我们提供了对神经网络训练过程中损失函数停滞现象的新视角，同时也提示了网络宽度如何影响训练效率和最终性能。

图 9.6展示了上述概念的形象化解释。想象一个旅人从起点出发，目标是到达遥远的城堡，途中充满了各种挑战和危险。在这个形象化比喻中，由单个神经元构成的网络中的全局最小点对应的临界点像是一名交警，指引旅人到达第一个里程碑——我们称之为第一个驿站。随后，一个包含两个神经元的网络中的全局最小点对应的临界点扮演了第二名交警的角色，继续引导

旅人前往第二个驿站。在实际的训练过程中，旅人可能会跳过某些中间驿站，例如，他可能会直接被一个包含三个神经元的网络的全局最小点对应的临界点所吸引，从而直接前往城堡。这个过程中的临界点及指引，构成了一条带有结构性的特殊路径，指引旅人穿越复杂且危险的损失景观。

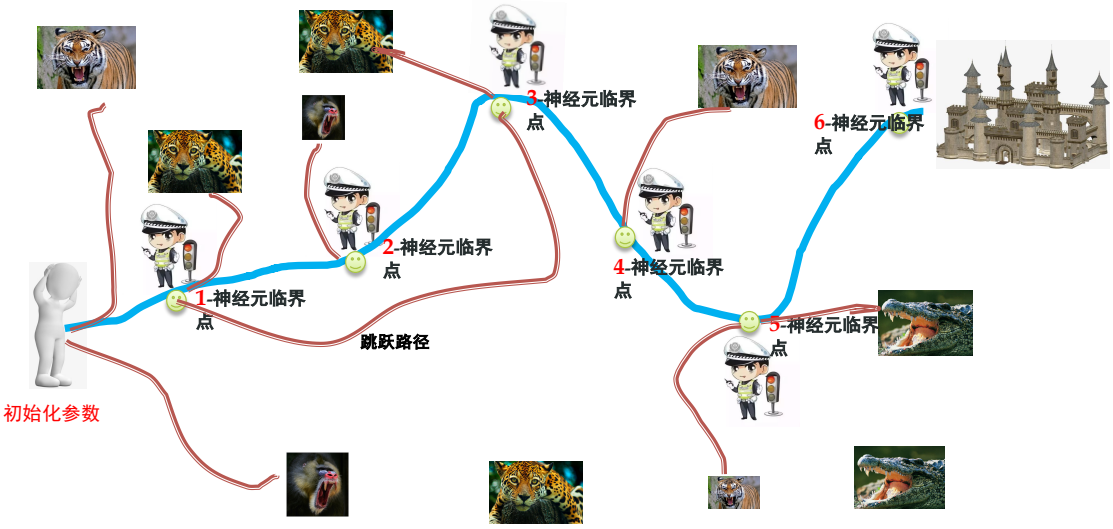


图 9.6: 优化路径在损失景观中的直观示意图

虽然神经网络的损失景观极其复杂，但其层结构带来的特殊性质允许网络在训练过程中逐渐增加复杂度。这种对参数空间的理解与频率原则对函数空间的理解相一致，即低频部分更容易被网络捕捉和优化，而高频部分则在训练过程的后期才逐渐被学习。

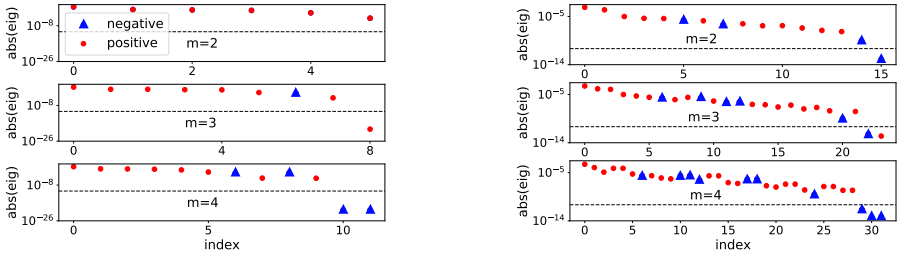
9.2.2 临界点嵌入后 Hessian 矩阵的特征值分析

嵌入原则保证了任何网络的损失景观都“包含”了所有较窄网络的临界点，在图 9.5中我们也观察到不同网络在训练过程中会在相同的损失值对应的临界点停滞。进一步，我们可以研究嵌入前后的临界点之间有什么差异。具体来说，我们有两个很直接的切入点，一是研究它们的下降方向的个数，也就是 Hessian 矩阵的负特征值个数，显然，下降的方向越多，对优化越容易；其次，我们可以研究它们的退化程度，也就是 Hessian 矩阵的零特征值个数。

首先，通过实验我们观察了嵌入原则作用下特征值如何变化。我们训练了一个宽度为 $m_{\text{small}} = 2$ 的两层神经网络，目标是学习一组从一维函数采样的数据（见图 9.7(a)）或是 Iris 数据集（见图 9.7(b)）。在损失函数下降缓慢到几乎停滞时，我们认为网络接近临界点，并检查此时的梯度数值。在图 9.7(a) 中，临界点处的损失函数梯度的 L_1 范数大约为 7.15×10^{-15}，

而在图 9.7(b) 中为 3.72×10^{-13}，这些极小的值使我们有理由认为这些点是临界点。

接下来，我们通过一步或两步嵌入将这些临界点转移到宽度为 $m = 3$ 和 $m = 4$ 的网络中。如图 9.7所示，每个子图中的辅助虚线是 $y = 10^{-11}$。在这些子图中，我们将处于临界点的宽度为 2 的两层神经网络中的一个神经元等分成 k 个神经元（$k = 2, 3$），其输入权重保持不变，但输出权重变为原神经元的 $1/k$。如图 9.7所示，每一步嵌入都在 Hessian 矩阵中引入了额外的零特征值。更为重要的是，在图 9.7(a) 中，$m = 2$ 的网络所有特征值均为正（红色原点），表明训练得到的临界点是局部或全局最小值。嵌入后，出现的负特征值（蓝色三角）将这些点转变为鞍点。在图 9.7(a) 和 (b) 中，我们观察到负特征值的数量随着网络宽度的增加而增加，例如观察在辅助虚线以上的部分，(a) 中从 0 增加到 1，再到 2；(b) 中从 3 增加到 5，再到 7。这个实验说明在嵌入后，Hessian 矩阵的特征值中出现了负特征值，这表明了更宽的神经网络中相应临界点的变化，以及从这些临界点中逃脱的难度降低，支持了我们在频谱分析中的观察。



(a) 一维合成数据　　(b) Iris 数据

图 9.7: 一维合成数据和 Iris 数据集中的临界点所对应的神经网络的 Hessian 矩阵特征值图

实际上，论文 Zhang, Li, Zhang, Luo & Xu (2021) 证明了对于一般的临界嵌入操作，大网络的 Hessian 矩阵的零特征值（退化方向），正特征值（上升方向）和负特征值（下降方向）的个数都不会减少。我们知道只要存在负特征值方向，一个临界点就不可能是局部最小点。因此，对于一个小网络的局部最小点，当它被嵌入到更宽的网络中时，由于很可能增加下降方向，它变成鞍点的可能性很大，而且下降方向的数目越多，训练过程逃离该鞍点就越容易。这种理解与我们在损失频谱中观察到的现象是一致的，即更宽的网络更容易学习到更小的损失值。

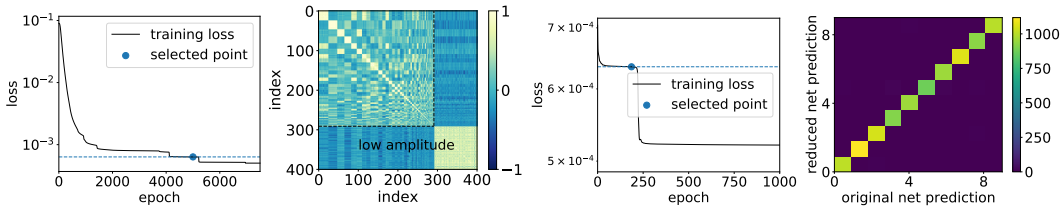
我们的实验和理论分析揭示了神经网络训练过程中的关键现象：较小网络的临界点通过嵌入操作变换到更宽网络中时，新的临界点会展现出更多的退化方向和下降方向。这意味着在宽网络的损失景观中，这些嵌入得到的临界点由于退化方向的增加而更有可能吸引训练轨迹。同时，更多的下降方向也意味着相较于小网络，宽网络更容易将损失函数优化到更低的值。

9.2.3 简化神经网络的规模

在实际的训练中，神经网络往往不会完全处于理论上的凝聚区域，导致我们难以观察到绝对的凝聚现象。尽管如此，我们仍可以观察到同层神经元的趋同行为，即不同的神经元趋于彼此相似，这种趋同可以作为一种控制模型复杂度的机制。

为了探究当神经网络趋近于一个由较小网络嵌入而来的临界点时，我们是否能够有效地缩减网络规模，我们进行了以下实验。根据嵌入原则，我们预期在较窄网络嵌入得到的临界点附近，凝聚的神经元组可以被单个神经元替换而不损失网络性能。

这一预测在图 9.8的实验中得到了验证。我们在 MNIST 数据集的 1000 个样本上训练了一个宽度为 400 的两层 ReLU 神经网络 $\boldsymbol{f}_{\boldsymbol{\theta}} = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^T \tilde{\boldsymbol{x}})$ $(\tilde{\boldsymbol{x}} = [\boldsymbol{x}^\intercal, 1]^\intercal)$。在网络训练过程中，我们发现在图 9.8(a) 中标记为蓝点的位置，损失下降非常缓慢，这表明网络可能接近一个鞍点。为了分析神经元之间的相似性，我们计算了每对神经元输入权重向量的归一化内积。



(a) 损失曲线　(b) 方向相似度　(c) 简化后的损失曲线　(d) 预测相似度

图 9.8: MNIST 训练过程图

如图 9.8(b) 所示，我们画出了不同神经元的输入权重的归一化内积。横坐标和纵坐标表示神经元序号。位于“低振幅”区域的神经元幅度远低于其他神经元，因此被移除。我们识别出 58 个神经元组，这些组内的神经元具有至少 0.9 的输入权重向量相似度。对于每个这样的相似性组 S_{similar}，我们随机选择一个神经元 j，并将其输出权重调整为组内所有神经元输出权重的加权和，权重由输入权重向量的模长决定。随后，我们移除了该组内的其他神经元。这样，神经网络的宽度从 400 减少到 58。

我们从调整后的参数 $\boldsymbol{\theta}_{\text{redu}}$ 开始，继续训练简化后的网络。如图 9.8所示，简化后的网络在几个训练步骤后，损失值停滞在与原始网络在图 9.8(a) 中标记的蓝点相同的值，由图中的蓝色虚线标记，并在图 9.8(c) 中以蓝点表示。进一步地，我们在 10000 个测试样本上比较了原始网络和简化网络在相应蓝点处的预测表现，如图 9.8(d) 所示。在该图中，横轴代表原始的宽网络在 MNIST 中 $0-9$ 10 个数字上的预测，纵轴代表简化后的小网络在 10 个数字上的预测。每个格点的颜色代表了两个模型预测结果相同的频率。对角线上的亮点表示两个模型之间的高预测一致性，总体一致性大约为 98.5%。

这个实验结果清楚地证明了我们的嵌入原则与真实数据集训练的相关性。简化后的宽度为 58 的神经网络在临界点的表现与原始宽度为 400 的神经网络高度一致，这表明在某些条件下，我们可以通过减少神经元数量来简化网络的规模而不牺牲太多性能。这为实现更高效的网络提供了一种可能的途径，并为网络结构优化提供了新的理论支持。

9.3 嵌入原则的扩展与推广

9.3.1 临界函数嵌入*

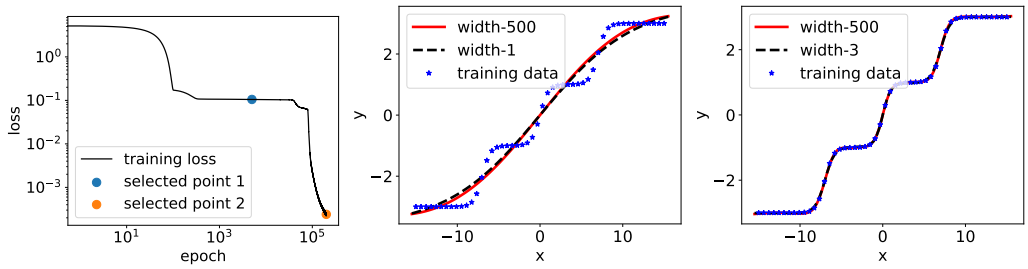
嵌入原则的数学基础是嵌入操作，该操作在保持输出函数和临界性（网络在参数空间的临界点）不变的同时，允许我们将一个较窄的网络嵌入到一个更宽的网络中。尽管传统上，大家倾向于从优化的视角来审视神经网络的损失景观，重点探讨参数空间中的性质，例如是否存在泛化能力较差的局部最小值，局部最小值是否全局最小，以及所有鞍点是否都是严格的鞍点等。但我们也必须认识到，损失景观对于网络的隐式正则化和泛化性能同样具有重要影响。因此，从函数空间的角度来研究损失景观，无疑是一项至关重要的研究工作。

在损失景观中，我们特别感兴趣的是那些有能力吸引训练轨迹的临界函数（临界点处的输出函数）。更具体地说，我们想要探索在宽度较大的网络中，是否存在那种能够隐式地引导训练过程以避免过拟合的“简单”临界函数。以图 9.9(a) 所示的实验为例，我们可以清晰地看到一个宽度为 500 的两层 tanh 网络在拟合 50 个数据点（蓝色星星）时，似乎是首先向一个“简单”的函数模型靠拢，该模型与一个宽度仅为 1 的 tanh 网络能够产生的最佳拟合输出函数相似（图 9.9(b)），并在该状态下停留了一段时间，这个状态看起来像是一个鞍点。随后，网络进入第二阶段，收敛至一个复杂度更高、类似于宽度为 3 的 tanh 网络能够产生的最佳拟合输出函数（图 9.9(c) 的黑色虚线），这个函数能够插值所有的数据点。这个观察结果表明，尽管宽度为 500 的网络具备过拟合的潜力，但其训练过程最终仍然趋向于一种非过拟合的数据插值解。在我们深入探索这种与泛化能力直接相关的训练行为的普遍性之前，理论上需要解答的关键问题是，在非常宽的网络中，是否总是存在这样的“简单”临界函数。

通过对嵌入原则的明确陈述和验证，我们对于前述关于网络训练过程中“简单”临界函数的存在性问题给出了肯定的答复：记 $\mathcal{F}^{\mathrm{c}}_{\mathrm{narr}} = \{\boldsymbol{f}_{\boldsymbol{\theta}} \mid \boldsymbol{\theta} \in \boldsymbol{\Theta}^{\mathrm{c}}_{\mathrm{narr}}\}$ 和 $\mathcal{F}^{\mathrm{c}}_{\mathrm{wide}} = \{\boldsymbol{f}_{\boldsymbol{\theta}} \mid \boldsymbol{\theta} \in \boldsymbol{\Theta}^{\mathrm{c}}_{\mathrm{wide}}\}$ 分别为窄网络和宽网络的临界函数集合，它们有如下的包含关系：

$$\mathcal{F}^{\mathrm{c}}_{\mathrm{narr}} \subset \mathcal{F}^{\mathrm{c}}_{\mathrm{wide}}. \tag{9.2}$$

此外，我们进一步阐明了所谓的“简单”临界函数的本质——它们实际上对应于较窄网络的临界函数。正如图 9.10中的（a）和（b）所展示的，无论是两层宽度为 m 的网络还是三层宽度为 $\{m_1, m_2\}$ 的网络，其临界函数均可以通过嵌入原则得到预测。



(a) 损失曲线　(b) 蓝点处的输出函数　(c) 橙点处的输出函数

图 9.9: 两层 tanh 神经网络的训练过程图

图 9.9中所示的宽度为 500 的 tanh 网络在训练过程中遇到的临界函数，在图 9.10（c）中得到了具体展示。根据频率原则，我们将那些由低频分量主导的函数视为训练网络时隐式倾向于选择的“简单”函数。而在这里，“简单”函数通过嵌入原则得到定义，即那些属于较窄网络的临界函数，它们揭示了训练数据在不同复杂性层次上的拟合情况，而这种复杂性是由临界函数所属的最窄网络的宽度所决定的。

结合频率原则和对凝聚现象的经验观察，我们可以推测非线性训练网络在其训练轨迹上隐式地偏好这些“简单”临界函数。显然，这一猜想的进一步研究对于未来的工作至关重要，它不仅能够加深我们对神经网络训练动力学的理解，还可能对解释神经网络的泛化之谜产生重要影响。

9.3.2 一般临界嵌入与临界流形的维数*

在研究临界嵌入时，我们的一个发现是，临界嵌入并非是点到点的映射，其嵌入方式也并不唯一。实际上，对于任何一个神经网络，都存在多种方式将其临界点嵌入到一个更宽网络的临界点集合中。这种多样性的存在暗示了一个更宽泛的临界流形（即一些临界点的集合），其中原有网络的每个临界点都能扩展到更高维度的临界子空间。

在上面的“分裂”嵌入的讨论中，我们发现通过连续进行 K 步的复合嵌入，每一步的嵌入都引入了由一个参数 α 描述的自由度。这种操作将窄网络参数空间中的单一临界点扩展为宽网络参数空间中一个 K 维的临界仿射子空间。这为临界点嵌入的退化（degeneracy）维数提供了一个理论上的下界估计。然而，神经元分裂操作仅仅只是一种临界嵌入操作，实际上还存在其他临界嵌入方式，而发现尽可能多的临界嵌入方式是至关重要的，因为这有助于我们更加全面地理解神经网络的临界子流形的维数和结构。

论文 Zhang, Li, Zhang, Luo & Xu (2021) 介绍了一类更为广泛的一般兼容性临界嵌入，这不仅包括了前文提及的神经元分裂嵌入，以及最直观的零神经元嵌入——即加入一个新的神

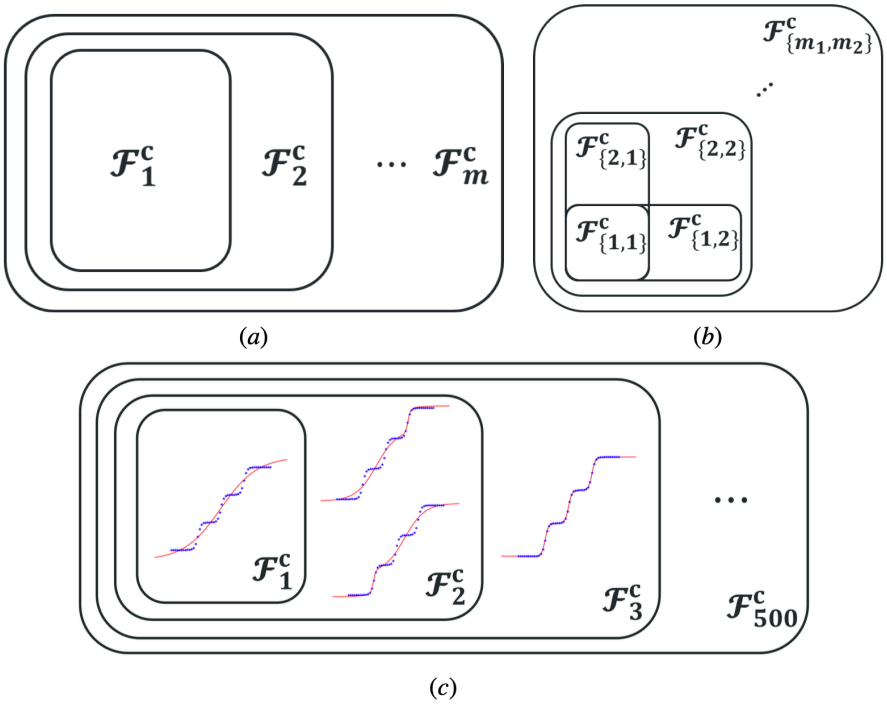


图 9.10: 嵌入原则对临界函数的示意图

经元，但其输入和输出权重为零，还包括了一大类满足兼容性条件的临界嵌入算子，例如在网络的前一层中如果两个神经元一致，那么在后一层的临界嵌入就可以拥有更多的自由度。通过对一般临界嵌入的研究，我们可以对临界流形的维数——对于任何给定的较窄神经网络中的临界函数，其对应的宽网络中临界流形维度——得到一个更为精确的估计。从直观上讲，临界嵌入中的自由度直接影响了临界子流形的维度。因此在考虑了一般临界嵌入之后，我们得到的退化维数估计实际上要比之前 K 维的简单案例更为复杂和高维。具体来说，论文 Zhang, Li, Zhang, Luo & Xu (2021) 估计了这个维度大约为 $K + \sum_{l\in[L]} K_l K_{l-1}$，其中 K_l 表示第 l 层额外增加的宽度，即 $K_l := m_l' - m_l$；而 K 表示总共增加的神经元数目，即 $K := \sum_{l\in[L-1]} K_l$。这一维度为我们提供了对临界流形高度退化的理解。

9.3.3 神经网络深度上的嵌入原则

在现代深度学习架构中，除了网络宽度，深度同样是一个关键维度。在详细讨论了宽度上的嵌入原则之后，深入理解神经网络在深度层面的嵌入效应显得尤为重要。论文 Bai et al. (2022) 通过一系列实验和理论分析，进一步阐明了深度嵌入原则，即一个更深的神经网络的损

失景观也包含了所有较浅网络的临界点。

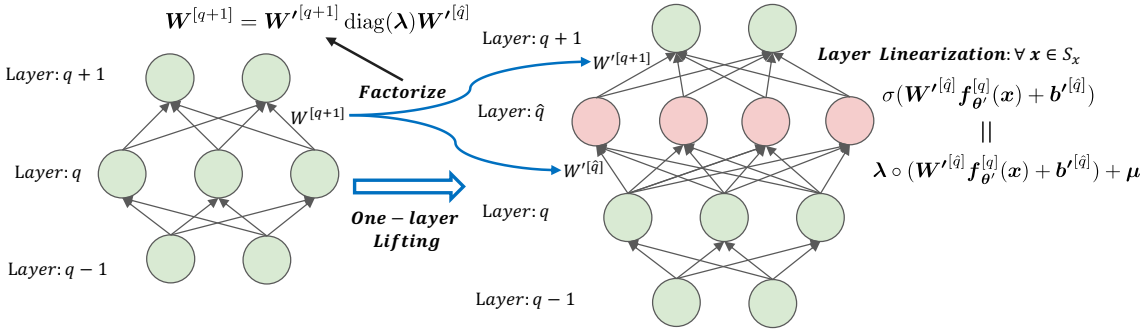


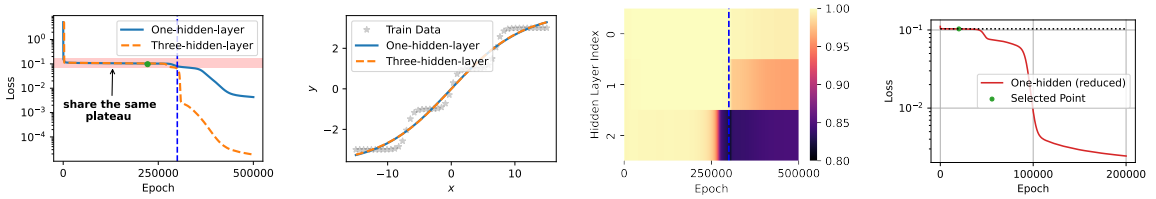
图 9.11: 一层提升的示意图

深度嵌入原则的关键在于所谓的层线性化操作。如图 9.11所示，我们可以通过在网络中插入一个额外的隐藏层实现所谓的一层提升（或称为层嵌入），如图中右侧网络的粉红色层（$\hat{q}$ 层）所展现。对于一个较浅网络的临界点，通过一层提升，我们可以将其转换为更深网络参数空间中的一个临界流形。这个临界流形由满足如下条件的所有参数向量构成：

(i) 插入层的权重 $\boldsymbol{W}'^{[\hat{q}]}$ 和偏置 $\boldsymbol{b}'^{[\hat{q}]}$ 满足层线性化条件，以确保在训练集输入 $S_{\boldsymbol{x}} = \{\boldsymbol{x}_i\}_{i=1}^{n}$ 上，新插入的层表现得如同线性层一般。

(ii) 第 $(q+1)$ 层的权重 $\boldsymbol{W}'^{[q+1]}$ 和偏置 $\boldsymbol{b}'^{[q+1]}$ 满足输出函数保持条件，确保网络中的 $\hat{q}$ 层与第 $(q+1)$ 层的复合在功能上等同于原始浅网络中的第 $(q+1)$ 层。

论文 Bai et al. (2022) 证明了当使用 ReLU，Leaky ReLU 等分段线性激活函数时，总存在一组参数严格满足上述的层线性化和输出函数保持条件。因此，对于任何较浅网络的临界点，我们也总能在深网络中找到输出函数一致的临界点。



(a) 损失曲线　(b) 平台处的输出函数　(c) 皮尔逊相关系数的演化　(d) 合并线性化层后的损失曲线

图 9.12: 深度神经网络在训练过程中经历浅网络嵌入的临界点实验图

此外，论文 Bai et al. (2022) 也进一步通过实验验证了在深网络的训练过程中，这些从浅网络中嵌入的临界点是可以实际经历的。如图 9.12(a) 所示，三隐藏层神经网络首先在与单隐藏层网络相同的损失值处发生停滞，在图 9.12(b) 中展示出几乎与单隐层网络相同的输出函数。我们可以使用皮尔逊相关系数来描述一层的线性化程度，皮尔逊相关系数越接近 1 代表该层越接近线性层。根据图 9.12(c)，前两个隐藏层在停滞期间显示出强烈的层线性特征。在图 9.12(d) 中，我们观察到，将线性化的层权重合并到前面的层，刚好对应到单隐藏层网络的一个临界点。

总的来说，深度上的嵌入原则与之前介绍的宽度上的嵌入原则相结合，为我们提供了关于神经网络损失景观内在层次结构的完整图景。这一图景有力地支撑了实验中观察到的不同规模大小神经网络之间在训练和泛化方面的相似性。嵌入原则的建立可以为我们理解不同规模网络的行为提供坚实的基础。

9.4 习题

1. 当不同大小的神经网络在相同的损失值上停滞时，这通常表明训练过程很可能遭遇了什么样的点？

2. 当我们说 “一个大网络的损失景观包含了一个小网络的临界点” 时，这里的 “包含” 具体指的是什么含义？

3. 临界嵌入是指什么？它有什么特性？

4. 什么是临界函数？对于一个任意宽度、任意深度的神经网络，给定任意数据，该网络的损失景观中是否总存在临界函数？

5. 临界流形是什么？它在神经网络优化中扮演什么角色？

6. 临界点处 Hessian 矩阵的特征值与神经网络的优化过程有何关联？

7. 考虑目标函数 $f^*(x) = \tanh(x-7) + \tanh(x) + \tanh(x+7)$，并给定从该目标函数中采样得到的 n 个数据点集合 $S = \{(x_i, f^*(x_i))\}_{i=1}^n$。考虑一个两层神经网络模型：

$$f_{\boldsymbol{\theta}}(x) = \sum_{j=1}^{m} a_j \tanh\left(w_j x + b_j\right),$$

其中参数集合为 $\boldsymbol{\theta} = [a_j, w_j, b_j]_{j=1}^m$。定义损失景观为：

$$R_S(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(x_i) - f^*(x_i)\right)^2.$$

当 $m = 3$ 时，

(i) 损失景观 $R_S(\boldsymbol{\theta})$ 是多少维的函数?

(ii) 是否存在全局最小点使得 $R_S(\boldsymbol{\theta}) = 0$?

(iii) 损失景观 $R_S(\boldsymbol{\theta})$ 是否为凸函数?

(iv) 是否存在非零损失值的临界点，即满足 $R_S(\boldsymbol{\theta}) \neq 0$? 如果存在，有多少个?

(v) 存在多少个临界函数 $\mathcal{F}^c := \{f_{\boldsymbol{\theta}}(\cdot) \mid \nabla R_S(\boldsymbol{\theta}) = 0\}$? 它们分别是什么?

8. 在问题 7 的设定下，对于任意宽度 m 的两层神经网络:

(i) 存在哪些临界函数 $\mathcal{F}^c := \{f_{\boldsymbol{\theta}}(\cdot) \mid \nabla R_S(\boldsymbol{\theta}) = 0\}$?

(ii) 每个临界函数对应的临界流形是什么? 其维度有多少?

(iii) 随着网络宽度 m 的增加，这些临界流形及其维数会如何变化?

(iv) 随着网络宽度 m 的增加，临界点处 Hessian 矩阵的特征值将如何变化?

Chapter 10

乐观估计

在深度学习领域，过参数化是一种常见的做法，它指模型的参数数量超过了训练样本的数量。一般而言，参数越多，模型的复杂度就越高。根据传统的学习理论，如果模型的参数量超过了样本量，那么模型很可能会发生过拟合。然而，深度学习颠覆了这一传统观念。神经网络即使在参数数量远超样本数量的情况下，仍然能够展现出卓越的泛化性能。这种现象被称为**“泛化之谜”**。这个谜团引发了我们的思考：我们应该如何理解这种现象？我们又应该如何构建泛化理论以解释这种现象呢？

在讨论泛化的时候，我们需要综合考虑模型和数据的特性。通常，数据的特性是相当复杂的。然而，无论数据的特性如何，我们使用的模型始终是神经网络。因此，我们的首要任务是研究神经网络的拟合特性，然后通过这些特性来理解数据。抽象地说，数据和神经网络就像两个黑盒子，我们需要先解开神经网络的黑盒子，然后再解开数据的黑盒子。

一个完整的泛化理论应该包括三个要素：数据、优化方法和模型。这个理论需要清楚地解释这三者如何共同影响泛化性能。由于数据是一个黑盒子，我们需要尽可能地简化和明确数据，以便更清楚地看到神经网络模型的影响。因此，在实验和理论研究中，我们选择的数据是从能够被某个小型网络表示的目标函数中采样得到的。这样做的目的是减少数据的复杂性，使我们能够更直接地观察和理解神经网络模型的效果。

在神经网络的训练过程中，我们常观察到凝聚现象，凝聚现象是非线性系统所特有的，我们预期它与神经网络的泛化能力有着密切的关系。然而，这种关系并没有被完全明确，我们还不清楚凝聚现象在泛化中的优势具体体现在哪里。从直观的角度来看，如果在训练过程中出现了凝聚现象，那么神经网络在训练动力学过程中可以被视为一个更小的网络。如果目标函数能被这个小型网络所表示，那么我们就有可能仅用很少的样本就能恢复出这个目标函数。凝聚现象的强度越大，恢复目标函数所需的样本数量就越小。因此，凝聚现象可以帮助神经网络从更

少的数据中学习到简单的目标函数，从而提升其泛化能力。

在本章中，我们进行了一系列的实验，来探索神经网络学到简单函数的可能性。接着我们提出了乐观估计的概念，来估计学到简单函数所需的样本量。

科普篇

在机器学习领域，传统的学习理论强调为了达到良好的泛化效果，模型的复杂度不应该太高。特别地，如果我们增大模型的规模（即复杂度），我们应该额外增加更多的样本以防止过拟合。然而，在深度学习中实际情况往往与理论背道而驰。即使在模型的参数数量远超样本数量的情况下，深度神经网络仍能展现出卓越的泛化能力，我们称这一现象为“泛化之谜”。本章将发展新的“乐观估计”理论框架来探索这种现象背后的深层次机制。

什么是乐观估计？

乐观估计是一种理论工具，用于估算神经网络拟合目标函数所需的最少样本量。它的核心思想是：考虑最好的情况，将参数的初始化选择在一个理想的点附近，从而最大限度地提升神经网络的恢复能力。在参数理想点附近，我们通过对模型进行线性近似，得到恢复目标函数所需的最小样本量，从而作为恢复目标函数所需样本量的下界。尽管乐观样本量是从一种不太实际的乐观初始化推导出的，但是我们的实验结果表明，无需乐观初始化，只需要小初始化加好的学习率超参选择，这种由理论推导得到的乐观样本量在实际模型中可以完全达到或者接近。

启发乐观估计的实验现象是什么？

通过一系列实验，我们发现，在矩阵分解模型和神经网络中，恢复目标函数所需的样本量显著小于模型的参数量。特别地，我们发现增大模型的规模并不需要额外增加同样的样本量来保持泛化。实验还显示了在不同初始化条件下，神经网络的恢复能力。例如，使用较小的初始化可以减少恢复所需的样本量，核心的原因就是我们之前章节中提到的凝聚现象。

乐观估计的理论基础是什么？

我们定义了模型秩 $R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}')$：

$$R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}') := \dim\left(\operatorname{span}\left\{\partial_{\theta_i} f(\cdot;\boldsymbol{\theta}')\right\}_{i=1}^{M}\right). \tag{10.1}$$

它表示在给定参数初始化 $\boldsymbol{\theta}'$ 时，模型 $f_{\boldsymbol{\theta}}$ 的线性化后的自由度。理论结果表明，模型秩等于线性恢复目标函数所需的最少样本量。

乐观估计的应用有哪些？

矩阵分解

我们考虑一个经典的问题：从部分观察数据中恢复一个矩阵。对这个问题的分析得出，恢复一个矩阵所需的样本量与其秩有关。实验表明，低秩矩阵在过参数化情况下恢复所需的样本量等于模型秩 $R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}')$。

$$R_{\boldsymbol{f_\theta}}(\boldsymbol{M}^*) = 2r_{\boldsymbol{M}^*}d - r_{\boldsymbol{M}^*}^2, \tag{10.2}$$

其中，$r_{\boldsymbol{M}^*}$ 为目标矩阵 $\boldsymbol{M}^*$ 的矩阵秩。

神经网络

在神经网络模型上，乐观估计展示了出色的适用性。我们发现，对于一个给定的目标函数，乐观样本量仅与目标函数的复杂度（内在宽度）相关，而与神经网络的实际宽度无关。

定义 3. *内在宽度 $k(f^*)$：定义为可以表示 f^* 的神经网络的最小宽度，即 f^* 可以由宽度为 $k(f^*)$ 的神经网络表示，但不能由任何更窄的神经网络表示。*

根据计算，神经网络的乐观样本量为：

$$R_{\mathrm{NN}_m}(f^*) = k(f^*)(d+1), \tag{10.3}$$

其中，$k(f^*)$ 表示目标函数 f^* 的内在宽度，d 为输入维度。实验结果表明，在超参数调节得当时，恢复目标函数所需的最小样本量接近乐观样本量。

乐观估计的潜力与挑战

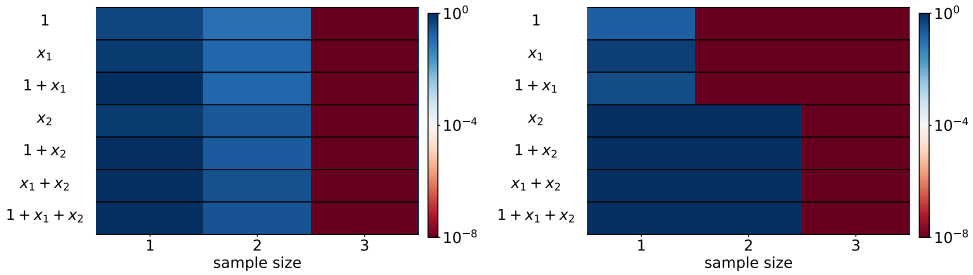
尽管乐观估计为神经网络的泛化之谜提供了合理的解释，其推广和应用仍面临诸多挑战。首先，如何在实际训练中有效地实现乐观初始化，使得训练过程逼近最佳情况？其次，对于更复杂的多层神经网络和高级模型（如 Transformer），如何准确估算其乐观样本量？

10.1 非线性模型在回归任务中的实验现象

我们进行了一系列非线性模型的实验，实验中我们将重点观察非线性模型能否学到目标函数。当模型的输出函数等于该目标函数时，我们称模型“恢复”了目标函数，简称“恢复”。此外，基于神经网络模型在小初始化的训练过程中会发生凝聚，我们猜测，凝聚越强，恢复目标函数所需要的样本数可能越少。受此启发，我们用小初始化来诱导非线性模型出现强的非线性的动力学，然后看恢复目标函数的情况。

10.1.1 简单的非线性回归模型

首先，我们来看一个基本的线性模型 $f_{\mathrm{L}}(\boldsymbol{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$，该模型具有 $M = 3$ 个参数。我们对这个线性模型进行重新参数化为 $f_{\mathrm{NL}}(\boldsymbol{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2\theta_3 x_2$ 。此时，虽然函数空间仍然保持为 $\mathcal{F}_{\mathrm{NL}} = \mathcal{F}_{\mathrm{L}}$，但模型在参数上变得非线性。



(a) $f_{\mathrm{L}}(\boldsymbol{x};\boldsymbol{\theta})$　　(b) $f_{\mathrm{NL}}(\boldsymbol{x};\boldsymbol{\theta})$

图 10.1: 不同目标函数（纵坐标）的平均测试误差（颜色）与训练样本数量（横坐标）的关系。

在图 10.1 中，我们以数据点的均方损失作为损失函数，使用小的初始化进行梯度下降，来训练这两个模型，模型的参数使用均值为 0 的高斯分布来初始化，高斯分布的方差为 10^{-8}。训练数据是从目标函数中随机抽取的一个、两个或三个数据点，纵坐标表示不同的目标函数，横坐标表示采样的个数。小方块的颜色表示平均测试（泛化）误差，是在随机初始化的 100 次试验中取平均得到的。对于线性模型 f_{L} 来说，每个目标函数都需要三个样本才能达到接近 0 的泛化误差。相反，对于非线性模型 f_{NL} 来说，只需要两个样本就足以拟合目标函数 1、x_1 和 $1 + x_1$。

虽然线性模型和非线性模型的函数空间完全相同，但是在“恢复”所需的样本量上，两个模型差异很明显。对于线性模型 $f_{\mathrm{L}}(\boldsymbol{x};\boldsymbol{\theta})$，需要 3 个样本才能恢复目标函数。对于非线性模型 $f_{\mathrm{NL}}(\boldsymbol{x};\boldsymbol{\theta})$，如果目标函数不含 x_2 项，那么 2 个样本就能恢复目标函数，如果目标函数含 x_2 项，那么要 3 个样本才能恢复目标函数。

10.1.2 矩阵分解模型

我们考虑一个 $\boldsymbol{f_\theta} = \boldsymbol{AB}$ 的非线性矩阵分解模型，这个模型在矩阵补全任务中有应用，我们的目标是从 n 个观察到的元素 $S = \left\{\left((i_s, j_s), \boldsymbol{M}^*_{i_s j_s}\right)\right\}_{s=1}^{n}$ 中恢复目标矩阵 $\boldsymbol{M}^*$，其中 (i_s, j_s) 表示矩阵 M^* 的行和列的索引。换句话说，我们已知目标矩阵 $\boldsymbol{M}^*$ 中某些位置的值，我们想还原 $\boldsymbol{M}^*$ 其他位置的值。

考虑一个非常简单的低秩矩阵补全任务：

$$\boldsymbol{M} = \begin{bmatrix} 1 & 2 \\ 3 & \star \end{bmatrix} \tag{10.4}$$

如果我们直接设定一个关于参数线性的模型 $f_{\boldsymbol{\theta}} = \boldsymbol{W} \in \mathbb{R}^{d\times d}$，并从零附近初始化用梯度下降并最小化以下目标：

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{s=1}^{n} \left([\boldsymbol{W}]_{i_s j_s} - \boldsymbol{M}^*_{i_s j_s} \right)^2, \tag{10.5}$$

$\boldsymbol{W}$ 的第二行第二列的元素拿不到梯度，所以不会被训练，因此一般学不到低秩解。

但是如果我们考虑一个 $\boldsymbol{f_\theta} = \boldsymbol{AB}, \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$ 的非线性矩阵分解模型，采用梯度下降并最小化以下目标：

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{s=1}^{n} \left([\boldsymbol{AB}]_{i_s j_s} - \boldsymbol{M}^*_{i_s j_s} \right)^2, \tag{10.6}$$

其中 $\boldsymbol{M}^* \in \mathbb{R}^{d\times d}$，$\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{B})$，以及 $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$，$[\boldsymbol{AB}]_{i_s j_s}$ 代表取出矩阵 $\boldsymbol{W} = \boldsymbol{AB}$ 第 i_s 行，第 j_s 列的数据。在这个非线性模型中，$\boldsymbol{A}$ 的第二行和 $\boldsymbol{B}$ 的第二列都会被训练，所以模型有可能学到低秩解，从而在只采样 3 个样本的时候就恢复这个秩 1 矩阵。

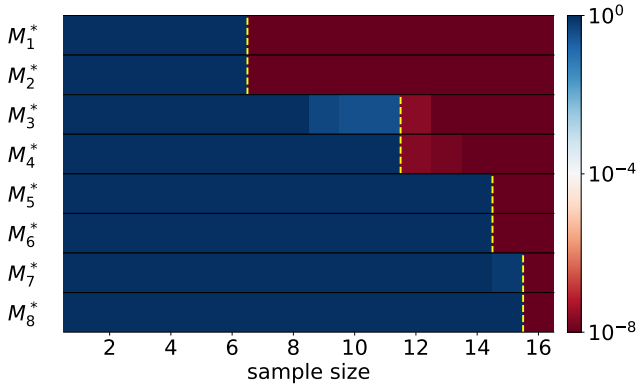


图 10.2: 平均测试误差（颜色）与训练样本数量（横坐标）在不同目标函数（纵坐标）上的关系。

为了研究这个模型在过参数化下是否真的能学到低秩解，我们对 4×4 的矩阵补全任务做了实验。通过实验我们发现，如果目标矩阵 $\boldsymbol{M}^*$ 的矩阵秩相同，那么恢复它所需的最小样本量就相同（矩阵秩是矩阵列空间的维数）。图 10.2 展示了使用从 1 到 16 的样本量对 8 个目标矩阵进行拟合的误差。图的横坐标是样本数量，纵坐标是不同的目标矩阵。$\mathrm{rank}(\boldsymbol{M}^*_{2k-1}) = \mathrm{rank}(\boldsymbol{M}^*_{2k}) = k$，$k = 1, 2, 3, 4$。其中 rank 表示矩阵的秩，所有的 $\boldsymbol{M}^*_i$ 都是 4×4 的方阵。颜

色表示代表平均测试误差（训练得到的矩阵和目标矩阵的欧式距离），每个测试误差是对随机初始化的 50 次实验进行平均得到的。图中的虚线表示平均测试误差突变的位置。

从图中我们发现，当目标矩阵秩为 1、2、3、4 时，恢复目标矩阵所需的样本量为 7、12、15、16。我们发现，该样本量刚好对应着秩 r 矩阵的自由度个数，也就是说，秩为 1、2、3、4 的 4×4 矩阵，它的自由度个数是 7、12、15、16。我们拿秩为 1 的 4×4 的矩阵举个例子。想要确定该矩阵，我们需要先确定第一行的值（4 个自由度），再确定其他三行同第一行的比值（3 个自由度），因此该矩阵一共有 7 个自由度。推广的结论是，对于一个秩为 r 的 $d\times d$ 矩阵，它的自由度个数是 $2rd-r^2$。

10.1.3 神经网络模型

在图 10.3 中，我们训练了各种宽度的两层 tanh-NN 和 tanh-CNN。它们的表达式如下：

定义 4. 对于一个两层 tanh-*NN* $f_{\boldsymbol{\theta}}$，我们有

$$f_{\boldsymbol{\theta}}(\boldsymbol{x})=\sum_{i=1}^{m}a_i\tanh(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}+b_i)$$

其中 $\boldsymbol{w}_i\in\mathbb{R}^d$，$b_i\in\mathbb{R}$，$a_i\in\mathbb{R}$，$\boldsymbol{\theta}=(\boldsymbol{w}_i,b_i,a_i)_{i=1}^m$ 是网络的参数，$\boldsymbol{x}\in\mathbb{R}^d$ 是网络的输入。

定义 5. 对于一个带权重共享的 tanh-*CNN* $f_{\boldsymbol{\theta}}$，我们有

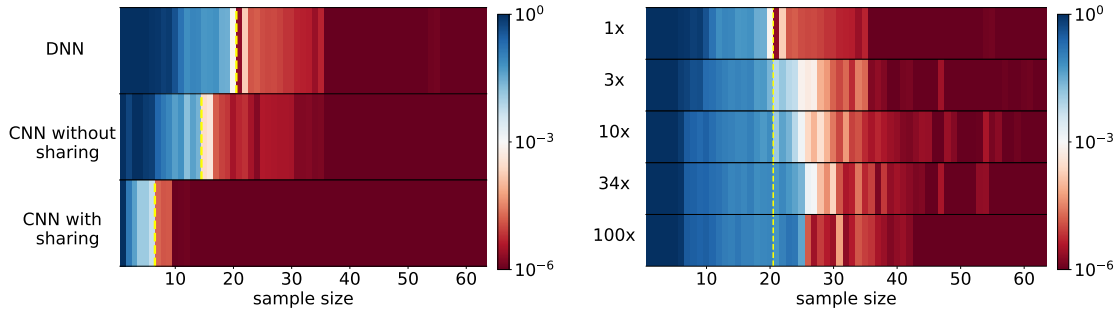
$$f_{\boldsymbol{\theta}}(\boldsymbol{I})=\sum_{l=1}^{m}\sum_{i,j=1}^{d+1-s}a_{lij}\tanh\left(\sum_{\alpha,\beta}I_{i+s-\alpha,j+s-\beta}K_{l;\alpha,\beta}\right),\boldsymbol{I}\in\mathbb{R}^{d\times d},$$

其中，m 表示卷积核的数量，d 表示输入图像的维度，s 表示卷积核的尺寸。a_{lij} 是卷积层的输出权重，$I_{i+s-\alpha,j+s-\beta}$ 是输入图像 $\boldsymbol{I}$ 的像素值，$K_{l;\alpha,\beta}$ 是第 l 个卷积核的权重。整个表达式描述了使用 *tanh* 激活函数的带权重共享的步长为 1 的卷积神经网络的输出。

定义 6. 对于一个不带权重共享的 tanh *CNN* $f_{\boldsymbol{\theta}}$，我们有

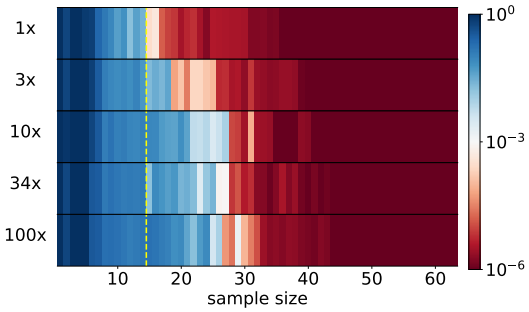
$$f_{\boldsymbol{\theta}}(\boldsymbol{I})=\sum_{l=1}^{m}\sum_{i,j=1}^{d+1-s}a_{lij}\tanh\left(\sum_{\alpha,\beta}I_{i+s-\alpha,j+s-\beta}K_{lij;\alpha,\beta}\right),\boldsymbol{I}\in\mathbb{R}^{d\times d},$$

其中，m 表示卷积核的数量，d 表示输入图像的维度，s 表示卷积核的尺寸。$\boldsymbol{K}_{lij}$ 是针对每个位置 (i,j) 的唯一卷积核矩阵，a_{lij} 是对应的输出权重，$I_{i+s-\alpha,j+s-\beta}$ 是输入图像 $\boldsymbol{I}$ 的像素值，$K_{lij;\alpha,\beta}$ 是对应于第 l 个卷积核在位置 (i,j) 的权重。整个表达式描述了使用 tanh 激活函数的不带权重共享的步长为 1 的卷积神经网络的输出。

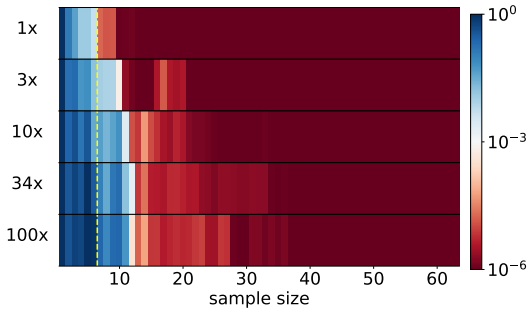


(a) 不同网络类型

(b) DNN



(c) CNN 无权重共享



(d) CNN 有权重共享

图 10.3: 不同宽度的神经网络（纵坐标）和样本大小（横坐标）在拟合目标函数公式(10.7)时的平均测试误差（颜色）。

图 10.3 展示了三种不同架构的神经网络在拟合同一目标函数时的平均测试误差。图中的纵坐标表示网络宽度，横坐标表示样本数量。所使用的目标函数为：

$$f^*(\boldsymbol{x}) = \boldsymbol{W}^{*[2]} \tanh(\boldsymbol{W}^{*[1]}\boldsymbol{x}), \tag{10.7}$$

其中 $\boldsymbol{W}^{*[2]} = [1,1,1]$ 和 $\boldsymbol{W}^{*[1]} = \begin{bmatrix} 0.6 & 0.8 & 1 & 0 & 0 \\ 0 & 0.6 & 0.8 & 1 & 0 \\ 0 & 0 & 0.6 & 0.8 & 1 \end{bmatrix}$，$\boldsymbol{x} \in \mathbb{R}^4$。图 10.3 包含四个子图，展示了不同架构的神经网络实验结果。

- **(a)**：对比三种神经网络架构的表现：两层 1 卷积核的 tanh-CNN；两层 1 卷积核但无权重共享的 tanh-CNN；以及两层、宽度为 3 的全连接 tanh-NN。在 (b)-(d) 中，这些网络被称为各自架构的 1x。

- **(b)**：两层、宽度为 $3N$ 的全连接 tanh-NN，其中 $N = 1, 3, 10, 34, 100$，并以 Nx 标记。
- **(c)**：两层、N 个卷积核但无权重共享的 tanh-CNN。
- **(d)**：两层、N 个卷积核的 tanh-CNN。

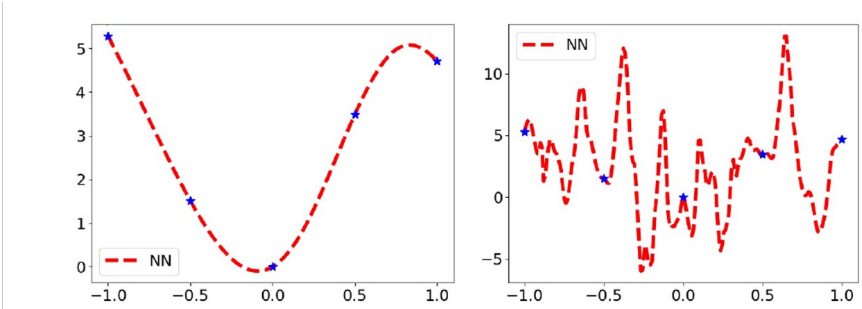
在所有实验中，网络参数均从均值为 0、方差为 10^{-20} 的正态分布中初始化，并使用梯度下降优化，同时对学习率进行微调。训练数据集和测试数据集的输入数据均从标准正态分布中采样，并使用目标函数计算输出值。训练数据集的大小可变，而测试数据集的大小固定为 1000。每个实验设定中，学习率在 0.05 到 0.5 之间调整，以优化泛化性能。

图中有许多现象。第一，在（a）中，我们看到, 在恢复目标函 (10.7)时，权重共享的 tanh-CNN 所需样本最少，其次是非权重共享的 tan-CNN，所需样本最多的是 tanh-NN。第二，在（b）、（c）、（d）图中，我们发现，神经网络具有很强的过参数化下恢复目标函数的能力。比如（b）图的 100x 的那一行，网络一共有 700 个参数，但是恢复目标函数仅需约 25 个样本。第三，我们发现，在（b）、（c）、（d）图中，无论网络有多宽，恢复目标函数所需的样本量都在虚线附近，似乎该样本量与网络的实际宽度关系不大。

10.1.4 超参数调节在非线性模型中的作用

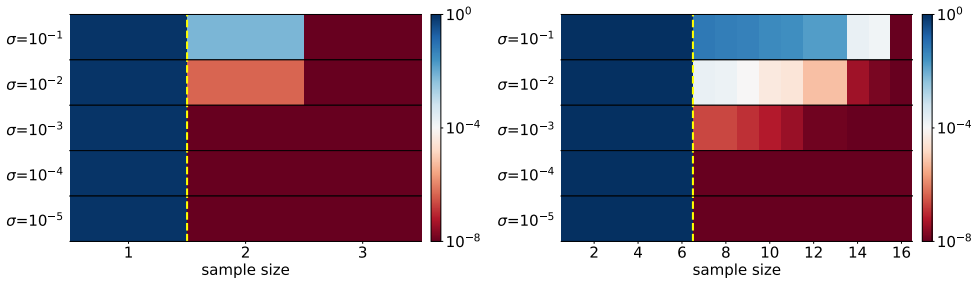
神经网络有许多超参数可以调节，比如初始化的高斯分布的方差、学习率等等。一方面，如果这些超参数选择得很不好，网络的泛化误差可能会很差。比如在图 10.4中，我们使用四个隐藏层的神经网络拟合蓝色星点数据, 网络的每层有 500 个神经元, 激活函数为 Tanh。两张图的差别在于参数初始化的高斯分布的方差不一致。左图的初始化方差较小，而右图初始化的方差较大。我们发现，如果初始化的方差很大，那么学到的解会高度震荡，导致泛化误差较差。另一方面，即使超参数调得特别好，恢复能力也总是有一个上限。比如在前面的矩阵分解问题中，想要恢复一个秩为 1 的 4×4 矩阵，无论超参数选得多好，也至少需要 7 个样本才能恢复目标矩阵。在前面神经网络的实验中，即使初始化非常小，凝聚非常强，我们也需要一定的样本量才能恢复目标函数。

为了更细致地说明超参数如何影响模型的恢复能力，我们在图 10.5中，展示了本节的简单任务和矩阵分解任务，在不同的初始化标准差大小（纵坐标）下，均值测试误差（颜色）和训练样本数量（横坐标）的关系。图 (a) 是用 $f_{\mathrm{NL}}(x, \theta)$ 拟合 $1 + x$ 的实验结果。图 (b) 是用矩阵分解模型拟合一个秩为 1 的矩阵。黄色虚线表示过渡到乐观样本量的位置。每个测试误差是对随机初始化的 50 次试验进行平均得到的。我们发现在图 (a) 中，如果初始化的标准差大于 10^{-2}，那么恢复目标函数需要 3 个样本点。但是如果初始化的标准差小于 10^{-3}，恢复目标函数只需要 2 个样本点。图 (b) 中也有类似的现象，即小初始化时，恢复所需的样本量会更少。但是即使初始化很小，恢复所需的样本量也有一个不能超过的下界。



(a) 小初始化。　(b) 大初始化。

图 10.4: 一维拟合例子的结果。



(a) 用 f_{NL} 来拟合 $1 + x_1$　(b) 矩阵分解，目标函数的秩为 1

图 10.5: 平均测试误差（颜色）与训练样本数量（横坐标）在不同的初始化标准差大小（纵坐标）上的关系。

10.2 量化模型恢复目标函数所需的最小样本量：模型秩

从上面的实验中，我们看到，要重构目标函数，好像有一个最小的样本量。我们想知道这个最小样本量怎么确定。与此同时，我们发现，即使很努力地调参数，在恢复问题中，所需样本量依旧有一个无法突破的下界，这个下界依赖目标函数和模型。比如在矩阵分解模型中，这个下界应该依赖于目标函数的矩阵秩。为了解释之前的实验现象，我们提出了一个新的理论框架—**乐观估计**。它可以从理论上分析恢复目标函数最少所需的样本量。

在回归中，我们使用模型 $f_{\boldsymbol{\theta}}$ 拟合从目标函数 f^* 采样的 n 个数据点，目标是恢复 f^*。为了使“恢复”可行，我们考虑那些能被模型 $f_{\boldsymbol{\theta}}$ 表示的目标函数，即 $f^* \in \mathcal{F} := \{f(\cdot;\boldsymbol{\theta}) | \boldsymbol{\theta} \in \mathbb{R}^M\}$。

默认情况下，我们均使用梯度下降来解决回归问题：

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_i; \boldsymbol{\theta}\right) - f^*(\boldsymbol{x}_i) \right)^2 . \tag{10.8}$$

梯度下降收敛到的解被记为 $f_{\boldsymbol{\theta}_\infty}$，其中 $\boldsymbol{\theta}_\infty$ 表示收敛时的模型参数。

10.2.1 恢复目标函数的参数集合：目标集

接下来我们定义一个重要的概念，**目标（参数）集（target set）**：

定义 7. 目标集 $\boldsymbol{\Theta}_{f^*} := \left\{ \boldsymbol{\theta} \mid f_{\boldsymbol{\theta}} = f^*, \boldsymbol{\theta} \in \mathbb{R}^M \right\}$

换句话说，目标集就是使得模型的输出函数等于目标函数的那些参数的集合。当梯度下降收敛的参数 $\boldsymbol{\theta}_\infty$ 属于目标集 $\boldsymbol{\Theta}_{f^*}$ 时，模型就恢复了目标函数，这恰恰是我们最期待发生的情况。而这一章我们要探究的就是何时模型能够收敛到目标集中呢？可以想象如果我们采样的样本点数目大于参数量，那么能够拟合样本点的所有全局极小值只会有 f^* 一种情况，这时模型收敛到目标集是不足为奇的。但是如果采样的样本量小于参数量，是否模型依然可以收敛到目标集中呢？在 10.1 节中，我们看到的一些实验现象表明这种情况在非线性模型中是很有可能发生的，那么随之而来的问题就是**这个最少采样的样本量是多少呢？**

为了理解这个问题，我们先来举几个目标集的例子。

例 1. 对于如下固定偏置的两个神经元的网络 $f_{\boldsymbol{\theta}}(x) = a_1 \tanh(\boldsymbol{w}_1 \cdot \boldsymbol{x} + 1) + a_2 \tanh(\boldsymbol{w}_2 \cdot \boldsymbol{x} + 1)$，其中参数 $\boldsymbol{\theta} = (a_1, \boldsymbol{w}_1, a_2, \boldsymbol{w}_2), \boldsymbol{w}_i \in \mathbb{R}^2, a_i \in \mathbb{R}, \boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$。设目标函数为 $f^*(x) = \bar{a} \tanh(\bar{\boldsymbol{w}} \cdot \boldsymbol{x} + 1)$。我们考虑能够恢复目标函数的目标集，一共有三种情况：
(1) 两个神经元的权重都取 $\bar{\boldsymbol{w}}$，前面的系数 a_1, a_2 满足 $a_1 + a_2 = \bar{a}$，即这时的目标集为 *1* 维的空间：$Q_1 = \{(a, \bar{\boldsymbol{w}}, \bar{a} - a, \bar{\boldsymbol{w}}) : a \in \mathbb{R}\}$.
(2) 第一个神经元的权重 $\boldsymbol{w}_1$ 取 $\bar{\boldsymbol{w}}$，前面的系数 a_1 取 $\bar{a}$，另一个神经元的权重 $\boldsymbol{w}_2$ 任意，前面的系数 a_2 取 0. 即这时的目标集为 *2* 维的空间：$Q_2 = \{(\bar{a}, \bar{\boldsymbol{w}}, 0, \boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^2\}$.
(3) 第二个神经元的权重 $\boldsymbol{w}_2$ 取 $\bar{\boldsymbol{w}}$，前面的系数 a_2 取 $\bar{a}$，另一个神经元的权重 $\boldsymbol{w}_1$ 任意，前面的系数 a_1 取 0. 即这时的目标集为 *2* 维的空间：$Q_3 = \{(0, \boldsymbol{w}, \bar{a}, \bar{\boldsymbol{w}}) : \boldsymbol{w} \in \mathbb{R}^2\}$.

那么整体的目标集 $\boldsymbol{\Theta}_{f^*} = Q_1 \cup Q_2 \cup Q_3$。在 Q^1 上，两个神经元发生理想凝聚，输入权重 $\boldsymbol{w}$ 一模一样。在 Q^2 和 Q^3 上，有一个神经元的输出权重 a 为 0。从几何上看，Q_1, Q_2 和 Q_3 看起来像三个不同维数的“分支”，Q_1 是一维的直线，Q_2, Q_3 是二维的平面，Q_1, Q_2 在 $(\bar{a}, \bar{\boldsymbol{w}}, 0, \bar{\boldsymbol{w}})$ 处相交，Q_1, Q_3 在 $(0, \bar{\boldsymbol{w}}, \bar{a}, \bar{\boldsymbol{w}})$ 处相交。参见图 *10.6*。

例 2. 在矩阵分解问题中，也有目标集的概念。比如设目标矩阵为 $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$，模型 $f_{\boldsymbol{\theta}} = \boldsymbol{AB}$，

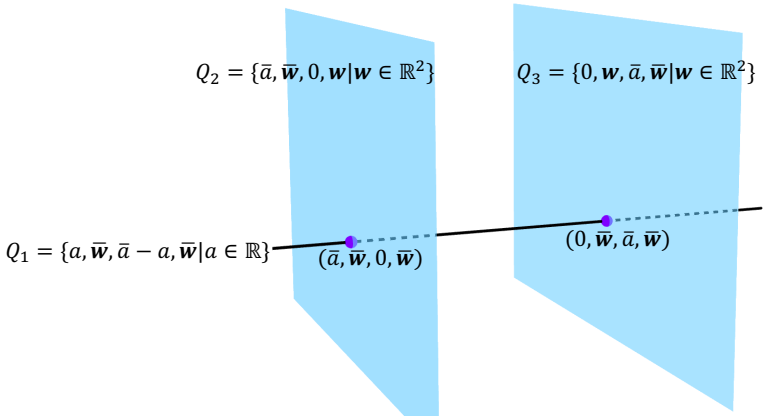


图 10.6: $\boldsymbol{\Theta}_{f^*}$ 的插图。

其中 $\boldsymbol{A} = (a, b)^T \in \mathbb{R}^{2\times 1}, \boldsymbol{B} = (c, d) \in \mathbb{R}^{1\times 2}, \boldsymbol{\theta} = (a, b, c, d)$。那么目标集 $\boldsymbol{\Theta}_{f^*} = \{(a, b, c, d) : ac = 1, b = d = 0\}$，它是 $\mathbb{R}^4$ 中的双曲线。

10.2.2 乐观初始化下的样本量估计

在过参数化下，有很多解使得模型的训练误差为 0，但是其中大部分解的泛化很差，所以在最差情况下，我们就不能恢复目标函数。但是我们之前的实验表明，非线性模型具有过参数化下恢复目标函数的能力，考虑最坏情况的理论不能解释我们的实验现象。但是，如果我们考虑最好的情况，将参数的初始化选择在一个很好的点，我们可能可以在过参数化下恢复目标函数。

首先，一个平凡的最优情况是，参数的初始化在目标集 $\boldsymbol{\Theta}_{f^*}$ 中，此时样本量为 0 就可以恢复目标函数。但是，第一，这样的样本量的估计不给我们带来任何有用的信息，实践中 0 个样本量显然是不能恢复目标函数的。第二，目标集 $\boldsymbol{\Theta}_{f^*}$ 的体积一般为 0。在均匀随机采样时，取到这种初始化的概率为 0。所以这种初始化是不可行的。一个次优情形（称之为乐观初始化）是参数的初始化在目标集附近，定义如下：

乐观初始化: 在目标集 $\boldsymbol{\Theta}_{f^*}$ 的一个“最优”点 $\boldsymbol{\theta}^*$ 的邻域中初始化。

我们注意到，目标集 $\boldsymbol{\Theta}_{f^*}$ 有不同的几何结构（见图 10.6），在不同的分支附近，恢复目标函数需要的样本量可能不同。我们的做法是，在目标集 $\boldsymbol{\Theta}_{f^*}$ 中挑选一个恢复目标函数所需样本量最小的 $\boldsymbol{\theta}^*$，并在该 $\boldsymbol{\theta}^*$ 附近初始化。这就是乐观初始化定义中“最优”的含义。此外，以

上乐观初始化导致的最佳的拟合场景是可能的，因为任何邻域 $N(\boldsymbol{\theta}^*,\epsilon)\subset\mathbb{R}^M$ 都有正的体积，在均匀采样时也就有正的概率取到该邻域。

我们使用线性近似 $f(\cdot;\boldsymbol{\theta})\approx f^{\text{lin}}(\cdot;\boldsymbol{\theta}):=f^*(\cdot)+\nabla f_{\boldsymbol{\theta}'}(\cdot)^{\mathrm{T}}(\boldsymbol{\theta}-\boldsymbol{\theta}')$ 来分析在小邻域 $N(\boldsymbol{\theta}',\epsilon)$ 中拟合所需的样本量，其中 $\boldsymbol{\theta}'\in\boldsymbol{\Theta}_{f^*}$。然后，回归问题 (18.1)变为

$$\min_{\boldsymbol{\theta}\in N(\boldsymbol{\theta}',\epsilon)}\frac{1}{n}\sum_{i=1}^{n}\left(f^{\text{lin}}(\boldsymbol{x}_i;\boldsymbol{\theta})-f^*(\boldsymbol{x}_i)\right)^2. \tag{10.9}$$

这是一个关于参数 $\boldsymbol{\theta}$ 线性的问题，$f^{\text{lin}}(\boldsymbol{x}_i;\boldsymbol{\theta})$ 作为 $\boldsymbol{\theta}$ 的函数位于一个线性空间中，该线性空间就是由模型 $f(\boldsymbol{x};\boldsymbol{\theta})$ 的切函数张成的空间，其维数是 $\operatorname{rank}(\nabla f_{\boldsymbol{\theta}'}(\cdot)^{\mathrm{T}})=\dim\left(\operatorname{span}\{\partial_{\theta_i}f(\cdot;\boldsymbol{\theta}')\}_{i=1}^{M}\right)$。对于这类线性模型（例如经典的线性回归），一个有一个广为人知的结果：我们需要

$$n=\dim\left(\operatorname{span}\{\partial_{\theta_i}f(\cdot;\boldsymbol{\theta}')\}_{i=1}^{M}\right)$$

个样本来恢复 f^*。受微分拓扑学中秩的启发，我们将这个量称为**模型秩**，对于任何 $\boldsymbol{\theta}'\in\mathbb{R}^M$ 都定义为

$$R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}'):=\dim\left(\operatorname{span}\{\partial_{\theta_i}f(\cdot;\boldsymbol{\theta}')\}_{i=1}^{M}\right). \tag{10.10}$$

这个定义看起来有点抽象，但实际上它的计算并不复杂，可以很容易地理解，我们来看两个具体的例子。

例 3. 第一个是经典的线性回归模型 $f(\boldsymbol{x};\boldsymbol{\theta})=a_1x_1+\ldots+a_kx_k$，其中 $\boldsymbol{\theta}=(a_1,\ldots,a_k)\in\mathbb{R}^k$。假设目标函数为 $f^*(\boldsymbol{x})=x_1$，我们来算一下模型秩。$\forall\boldsymbol{\theta}\in\mathbb{R}^k$，我们有 $f(\boldsymbol{x};\boldsymbol{\theta})$ 关于每个参数 θ_i 的偏导数 $\partial_{\theta_i}f(\cdot;\boldsymbol{\theta})=x_i$，而 $\{x_1,x_2,\cdots,x_k\}$ 作为 k 个函数是线性无关的。此处线性无关的定义是经典的，即 $\sum_{i=1}^{k}\alpha_ix_i=0\implies\alpha_j=0,\forall 1\le j\le k.$

根据定义，模型秩就是模型关于参数求导之后得到切函数张成空间的维数，所以模型秩 $R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}'):=\dim\left(\operatorname{span}\{\partial_{\theta_i}f(\cdot;\boldsymbol{\theta})\}_{i=1}^{M}\right)=k$。我们发现，在线性模型中，模型秩是一个常数，不随 $\boldsymbol{\theta}'$ 变化。

例 4. 第二个是前文提到的固定偏置的两个神经元的网络 $f_{\boldsymbol{\theta}}(x)=a_1\tanh(\boldsymbol{w}_1\cdot\boldsymbol{x}+1)+a_2\tanh(\boldsymbol{w}_2\cdot\boldsymbol{x}+1)$，目标函数为 $f^*(x)=\bar{a}\tanh(\bar{\boldsymbol{w}}\cdot\boldsymbol{x}+1)$，其中 $\boldsymbol{x}\in\mathbb{R}^2$。我们有目标集 $\boldsymbol{\Theta}_{f^*}=Q_1\cup Q_2\cup Q_3$。经过简单的计算得到，如图 *10.7* 所示，在 Q_1 和 Q_2,Q_3 中，模型秩 $R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}')$ 分别为 3、4、4（留作习题）。

我们看到，对于非线性模型，在目标集中，不同的点的模型秩不一样，我们想从目标集中挑一个模型秩 $R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}')$ 最小的 $\boldsymbol{\theta}'$，并在 $\boldsymbol{\theta}'$ 附近初始化。所以，对于任何函数 $f'\in\mathcal{F}$，模型秩定义为 $\boldsymbol{\Theta}_{f'}$ 中的最小秩，即

$$R_{f_{\boldsymbol{\theta}}}(f'):=\min_{\boldsymbol{\theta}'\in\boldsymbol{\Theta}_{f'}}R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}'). \tag{10.11}$$

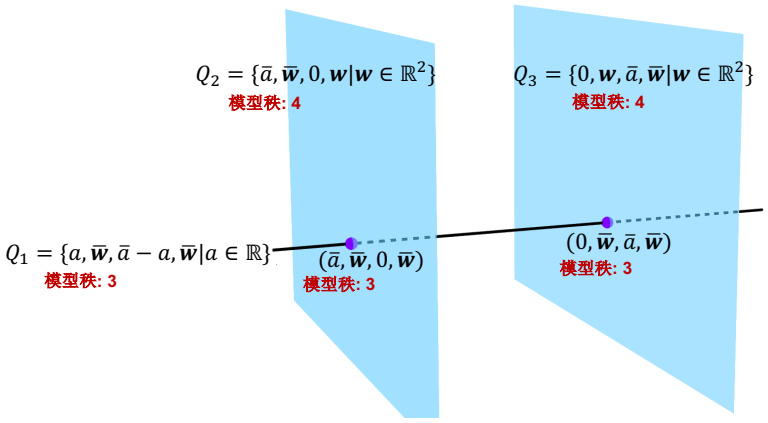


图 10.7: $\boldsymbol{\Theta}_{f^*}$ 的插图。

注意，为了方便，我们稍微重复使用了 $R_{f_{\boldsymbol{\theta}}}$ 的记号。它是关于函数的秩还是参数点的秩，取决于它的输入。在乐观初始化下，拟合 f^* 所需的样本量为 $R_{f_{\boldsymbol{\theta}}}(f^*)$，我们将其称为 f^* 的乐观样本量，它决定了使用模型 $f_{\boldsymbol{\theta}}$ 拟合 f^* 所需的最小样本量。

10.2.3 乐观估计的重要意义

(1) 乐观样本量的上下界： $0 \leqslant R_{f_{\boldsymbol{\theta}}}(f^*) \leqslant M$。对于关于参数 $\boldsymbol{\theta}$ 线性的模型 $f(\boldsymbol{x}; \boldsymbol{\theta})$，例 3 表明其乐观样本量是一个常数，并且就等于参数量 M. 这和我们通常的直观符合，对于线性模型，我们必须采样 M 个样本才能保证模型恢复目标函数。如果我们一味地增大模型的参数量，而不相应地增加同样的样本量，模型的泛化就会变得越来越差。但是，对于非线性模型，如神经网络，例 4 表明其乐观样本量可以小于模型参数量 M。这和我们在图 10.3 中看到的实验现象类似，如果我们增大模型的参数量，恢复目标函数所需的样本量并不需要随模型参数量的增加而同等程度地增加，而是显著小于参数量。

(2)：乐观样本量同时受模型架构 $f_{\boldsymbol{\theta}}$ 和目标函数 f^* 的影响。这与”没有免费午餐”定理相符。该定理指出，不存在一种单一的算法能有效学习所有可能的目标函数，也即一个目标函数的复杂度必须与尝试学习它的特定模型相关联。在一个模型下看似复杂的函数，在另一个模型下可能就很简单。模型秩量化了这种架构-目标函数关系，为评估架构选择提供了数学框架。从机器学习的角度来看，我们的一个目标就是为某类特定类别任务设计最适合的架构。例如，如果目标函数就是具有局部结构的函数（例如自然图像），那么如图 10.3 所示，卷积神经网络（CNNs）相比全连接网络，其样本复杂度（通过模型秩计算）就会显著降低。

(3) 乐观估计给出了有效参数的度量，能够精确地描述非线性回归中的过参数化。传统的过参数化定义（样本量 $n < M$）可能会产生误导。例如考虑用 $f_{\boldsymbol{\theta}} = (\theta_1 + \theta_2)x$ 拟合一个线性函数

$f^*(x) = x$：尽管有 $M = 2$ 个参数，但这个模型只需要一个样本就能拟合。为了解决这个定义的局限性，我们借助模型秩将过参数化定义为 $n < M_{\mathrm{I}}$ 的情况，其中：

$$M_{\mathrm{I}} := \max_{\boldsymbol{\theta}^* \in \mathbb{R}^M} R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}^*) \tag{10.12}$$

这里，M_{I} 代表模型 $f_{\boldsymbol{\theta}}$ 的最大有效参数数量。当对所有 $f^* \in \mathcal{F}$，$R_{f_{\boldsymbol{\theta}}}(f^*) = M_{\mathrm{I}}$ 时，模型 $f_{\boldsymbol{\theta}}$ 在过参数化条件下无法实现拟合。对于关于参数线性的模型，很容易验证模型秩保持恒定（为 M_{I}），这排除了它们在过参数化条件下恢复目标函数的能力。相比之下，非线性模型——如矩阵分解模型和神经网络——能够在过参数化条件下实现恢复目标函数，展示了非线性所带来的根本优势。

注 27. 虽然基于乐观初始化导出的乐观样本量看起来有点不切实际，因为在做一个机器学习任务之前，目标函数 f^* 和目标集 $\boldsymbol{\Theta}_{f^*} := \left\{\boldsymbol{\theta} \mid f_{\boldsymbol{\theta}} = f^*, \boldsymbol{\theta} \in \mathbb{R}^M\right\}$ 都是事先未知的，我们不能在目标集中的 $\boldsymbol{\theta}^*$ 附近初始化。但是我们要强调的是，基于乐观初始化发展出的乐观估计框架为我们**理解**非线性模型的行为提供了重要见解，该框架描述了**为什么一个模型在过参数化下恢复目标函数是可能的**，这在理解上是一个重要的突破，希望读者可以体会到这一点的非平凡性。

注 28. 基于乐观初始化导出的乐观样本量建立了所需样本量的下界。这个界限的重要性体现在两个方面：
(1) 它提供了一个基本限制——如果在乐观初始化下模型都不能恢复目标函数，那么在任何其他更差的初始化方案下也大概率失败。我们将尝试在之后的工作严格证明这一点。
(2) 更令人惊讶的是，我们下一节的的计算和实验结果表明，常规的随机小初始化，无需乐观初始化，加上学习率的合适调节，通常就能达到接近理论界限的性能。具体而言，简单的架构如矩阵分解模型能刚好达到这个乐观样本量界限，而更复杂的模型，如神经网络模型实际上也能非常接近这个界限。通过理论推导的界限与实验观察到的性能之间的这种对应关系验证了乐观估计框架的有效性，并表明乐观样本量不仅仅是一个停留在纸面的理论构造，而是在实践中可以预测模型行为的有意义指标。

10.3 乐观样本量和实际实验表现的对比

10.3.1 简单的非线性回归模型

首先，我们来看一个基本的线性模型 $f_{\mathrm{L}}(\boldsymbol{x}; \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$，该模型具有 $M = 3$ 个参数。考虑任意的 $\boldsymbol{\theta}' = [\theta_0', \theta_1', \theta_2']^T \in \mathbb{R}^3$，有 $R_{f_{\mathrm{L}}}(\boldsymbol{\theta}') = \dim\left(\mathrm{span}\left\{1, x_1, x_2\right\}\right) = 3$。因此，对于任何 $f' \in \mathcal{F}_{\mathrm{L}} := \{f_{\mathrm{L}}(\cdot; \boldsymbol{\theta}) | \boldsymbol{\theta} \in \mathbb{R}^3\}$，我们有 $R_{f_{\mathrm{L}}}(f') = M_{\mathrm{I}} = 3$。因此，使用模型 $f_{\boldsymbol{\theta}}(x)$ 恢复

任何函数都至少要 3 个样本，所以 f_{L} 无法在过参数化的情况下恢复目标函数。同样，我们可以证明所有的线性模型都具有固定的乐观样本量。因此，所有线性模型都无法在过参数化的情况下恢复目标函数。这与线性回归的理论是一致的。

当我们对线性模型进行重新参数化为 $f_{\mathrm{NL}}(\boldsymbol{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2\theta_3 x_2$ 时，乐观样本量就会发生变化。虽然函数空间仍然保持为 $\mathcal{F}_{\mathrm{NL}} = \mathcal{F}_{\mathrm{L}}$，但模型在参数上变得非线性。模型的乐观样本量估计如下。作为参数空间 $\mathbb{R}^4$ 上的函数的模型秩为：

$$R_{f_{\mathrm{NL}}}(\boldsymbol{\theta}') = \dim\left(\mathrm{span}\{1, x_1, \theta_3' x_2, \theta_2' x_2\}\right) = \begin{cases} 2, & \theta_2' = \theta_3' = 0, \\ 3, & \text{others.} \end{cases} \tag{10.13}$$

所以模型的有效参数量 $M_{\mathrm{I}} = 3$。通过解决最小化问题 (18.5)，我们得到模型秩作为函数空间 $\mathcal{F}$ 上的函数为

$$R_{f_{\mathrm{NL}}}(f^*) = \begin{cases} 2, & f^* \in \{a_0 + a_1 x_1 | a_0, a_1 \in \mathbb{R}\}, \\ 3, & f^* \in \{a_0 + a_1 x_1 + a_2 x_2 | a_2 \neq 0, a_0, a_1, a_2 \in \mathbb{R}\}. \end{cases} \tag{10.14}$$

上述估计表明，非线性模型 f_{NL} 可以使用两个样本在过参数化的情况下恢复 $f^* \in \{a_0 + a_1 x_1 | a_0, a_1 \in \mathbb{R}\}$，但不能对 $f^* \in \{a_0 + a_1 x_1 + a_2 x_2 | a_2 \neq 0, a_0, a_1, a_2 \in \mathbb{R}\}$ 做到过参数化下的恢复。我们看到，**乐观样本量分析对图 10.1 中的实验现象做到了完美的预测**。在线性模型 f_{L} 的情况下，每个目标函数都需要三个样本才能达到接近 0 的泛化误差。相反，对于非线性模型 f_{NL}，只需要两个样本就足以恢复目标函数 1、x_1 和 $1 + x_1$。这表明，在这个简单的模型上，乐观样本量作为“恢复”所需的样本量的下界，是可以达到的。

10.3.2 矩阵分解模型

我们将我们的乐观估计应用于图 10.5 中的 $\boldsymbol{f}_{\boldsymbol{\theta}} = \boldsymbol{A}\boldsymbol{B}$ 的非线性矩阵分解模型。回忆一下，我们的目标是从 n 个观察到的元素 $S = \left\{\left((i_s, j_s), \boldsymbol{M}^*_{i_s j_s}\right)\right\}_{s=1}^n$ 中恢复目标矩阵 $\boldsymbol{M}^*$。我们采用梯度下降并最小化以下损失函数：

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{s=1}^{n} \left([\boldsymbol{A}\boldsymbol{B}]_{i_s j_s} - \boldsymbol{M}^*_{i_s j_s}\right)^2, \tag{10.15}$$

其中 $\boldsymbol{M}^* \in \mathbb{R}^{d\times d}$，$\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{B})$，以及 $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$。通过采用标准的乐观估计过程，我们确定任何 $\boldsymbol{M}^* \in \mathbb{R}^{d\times d}$ 的乐观样本量为：

$$R_{\boldsymbol{f}_{\boldsymbol{\theta}}}(\boldsymbol{M}^*) = 2 r_{\boldsymbol{M}^*} d - r_{\boldsymbol{M}^*}^2, \tag{10.16}$$

其中 $r_{\boldsymbol{M}^*} = \mathrm{rank}(\boldsymbol{M}^*)$ 是 $\boldsymbol{M}^*$ 的矩阵秩（证明见Zhang, Zhang, Zhang, Bai, Luo & Xu (2023)）。这个结果刚好跟我们前面分析的 $d \times d$ 秩 r 矩阵的自由度 $2rd - r^2$ 是一致的。值得注意的是，

对于固定的 d，乐观样本量随 $r_{\boldsymbol{M}^*}$ 增加而增加。特别是，对于大的 d，秩为 r 的目标矩阵的乐观样本量为 $2rd-r^2$，这比有效参数量 $M_{\mathrm{I}}=d^2$ 要小得多。因此，矩阵分解模型可能在过参数化的情况下恢复低秩矩阵。

在图 10.5 的实验测试中，我们发现上述推导出的乐观样本量是可以达到的。参照方程 (18.12)，一个 4×4 的目标矩阵 $\boldsymbol{M}^*$，其秩为 1，2，3 或 4，对应的乐观样本量分别为 7，12，15 或 16。当使用方差很小，均值为 0 的高斯分布进行初始化时，**恢复目标矩阵所需的样本量与黄色虚线指示的乐观样本量完全一致**。

10.3.3 神经网络模型

首先，我们考虑一个具有 m 个神经元的两层全连接神经网络，其激活函数为 $\tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$，神经网络由 $f_{\boldsymbol{\theta}}(\boldsymbol{x})=\sum_{i=1}^m a_i\tanh(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x})$ 表示，其中 $\boldsymbol{x}\in\mathbb{R}^d,\boldsymbol{\theta}=(a_i\in\mathbb{R},\boldsymbol{w}_i\in\mathbb{R}^d)_{i=1}^m$。网络总共有 $M=m(d+1)$ 个参数。经过计算得到，乐观样本量与目标函数 f^* 的内在宽度 $k(f^*)$ 有关。

定义 8. *内在宽度 $k(f^*)$：定义为可以表示 f^* 的神经网络的最小宽度，即 f^* 可以由宽度为 $k(f^*)$ 的神经网络表示，但不能由任何更窄的神经网络表示。*

对于一个可以由宽度为 m 的神经网络表示的 f^*，其内在宽度 $0\leqslant k(f^*)\leqslant m$。$f^*$ 的乐观样本量为

$$R_{\mathrm{NN}_m}(f^*)=k(f^*)(d+1), \tag{10.17}$$

实现该乐观样本量的 $\boldsymbol{\theta}'$ 的取法是，先选取 $k(f^*)$ 个神经元来表示 f^*，然后令其他的神经元的 a 和 $\boldsymbol{w}$ 全为 0，此时 $\boldsymbol{\theta}'$ 对应的模型秩就是 $k(f^*)(d+1)$。乐观样本量 $R_{\mathrm{NN}_m}(f^*)$ 随着内在宽度的增加线性增加。因此，对于一个内在宽度 $k(f^*)\ll m$ 的函数，当参数的初始化为乐观初始化时，我们可以在样本量小于参数量的情况下恢复它。

然后，我们考虑一个具有权重共享的两层 tanh-CNN（见图 10.8），表示为

$$f_{\boldsymbol{\theta}}(\boldsymbol{x})=\sum_{i=1}^{m_{\mathrm{C}}}\sum_{j=1}^{d+1-s}a_{ij}\tanh\left(\sum_{\alpha=1}^{s}x_{j+s-\alpha}K_{i;\alpha}\right), \tag{10.18}$$

其中 $\boldsymbol{x}=[x_1,\cdots,x_d]^{\mathrm{T}}\in\mathbb{R}^d$，$K_i$ 是卷积核（也称作神经元），s 是核的大小，a_{ij} 是输出权重，卷积方式为一维卷积。对于每个 i，K_i 有 s 个参数，$(a_{ij})_{i=1}^{d+1-s}$ 共有 $d+1-s$ 个参数，所以对于每个 i，表达式 $\sum_{j=1}^{d+1-s}a_{ij}\tanh\left(\sum_{\alpha=1}^{s}x_{j+s-\alpha}K_{i;\alpha}\right)$ 中有 $d+1$ 个参数。所以模型一共有 $m_{\mathrm{C}}(d+1)$ 个参数。其中 m_{C} 是卷积核的数量，类似于全连接神经网络的宽度。类似于全连接中定义的内在宽度，我们定义内在核数为：

定义 9. *内在核数 $k_{\mathrm{C}}(f^*)$：定义为可以表示 f^* 的卷积网络的卷积核数量的最小值。即 f^* 可以由卷积核数量为 $k_{\mathrm{C}}(f^*)$ 的卷积网络表示，但不能由任何卷积核更少的卷积网络表示。*

注意，如果卷积网络是权重共享的，那么那些权重共享的神经元被看成一个卷积核。也就是说，一个卷积核里面，可以包含多个权重相同的神经元。如果权重不共享，那么一个卷积核里面只有一个神经元。类似与全连接情况，乐观样本量是目标函数 f^* 的内在核数 $k_{\mathrm{C}}(f^*)$ 的函数。具体来说，对于任何可以由 m_{C}-核 CNN 表示的 f^*，其乐观样本量为

$$R_{\mathrm{CNN}_{m_{\mathrm{C}}}}(f^*) = k_{\mathrm{C}}(f^*)(d+1). \tag{10.19}$$

因此，一个内在核数 $k_{\mathrm{C}}(f^*) \ll m_{\mathrm{C}}$ 的函数可以在过参数化的情况下被卷积网络恢复。

我们现在回头看一下前面的神经网络的实验（见图 10.3）。图中的红蓝交界处是“恢复”所需的样本量，而黄线则是计算得到的乐观样本量。我们看到，在超参数调整得当的情况下，**神经网络“恢复”所需的样本量很接近乐观样本量**。注意，在这个实验中，参数的初始化是在 **0** 附近的小初始化，并不是乐观初始化。也就是说，即使初始化不在目标集附近，当样本量在乐观样本量附近时，神经网络也能恢复目标函数。这表明，网络的实际性能与乐观初始化下的网络性能较为接近。

10.4 神经网络架构设计的分析：乐观样本量是否增加

从乐观样本量的角度来分析神经网络，我们看到两条重要的性质。

第一个性质是**宽度的免费表达能力**。在全连接网络和卷积网络中，目标函数的乐观样本量只与目标函数的内在宽度（或内在核数）有关，与网络的宽度无关。这表明，对于两层层全连接网络和卷积网络，增加宽度 (或核数) 可以增加他们的表达能力，却不会增加乐观样本量。由于网络实际“恢复”所需的样本量和乐观样本量接近，所以我们可以认为网络“恢复”所需的样本量几乎不变。

第二个性质是**连接的昂贵表达能力**。在表格 10.1中，我们展示了同一个目标函数在不同架构的神经网络下的乐观样本量。CNN、无权重共享 CNN、全连接网络的表达式见定义5、6、4。表格中的 $m_{\mathrm{null}} = |\{(l,i,j)|a^*_{lij} = 0\}|$ 表示目标函数中输出权重为零的数量，$\mathcal{F}_k^{\mathrm{CNN}}$ 表示能被具有 k 个卷积核的权重共享的 CNN 表示的目标函数的全体。我们来举个例子算一下不同架构的乐观样本量的差别。假设目标函数 f^* 满足 $f \in \mathcal{F}_k^{\mathrm{CNN}} \backslash \mathcal{F}_{k-1}^{\mathrm{CNN}}$，并且 f^* 的输出权重均不为 0，即 $m_{null} = 0$。卷积核大小 $s = 3$，输入的图片大小为 28×28，即 $d = 28$。此时，f^* 在 CNN、CNN（无权重共享）、全连接网络中的乐观样本量分别是 $709k$、$6760k$、$530660k$。我们看到，在不同架构下恢复同一个目标函数所需的样本量差别巨大。当目标函数能被 CNN 表示时，CNN 网络的乐观样本量远小于其他两种架构。图 10.8展示了这三种网络的架构，图中从左到右依次为：两层全连接网络、无权重共享的卷积网络、有权重共享的卷积网络。该示意图中的卷积为一维卷积。在图中可以看到，从权重共享的卷积网络到全连接网络，增加了许多神经元之间的连接。当目标函数能被权重共享的 CNN 表示时，这些新增加的连接是不必要的。

这些不必要的连接尽管增强了神经网络的表示能力，但是它会以牺牲样本效率为代价。这种特性称就是**连接的昂贵表达能力**。

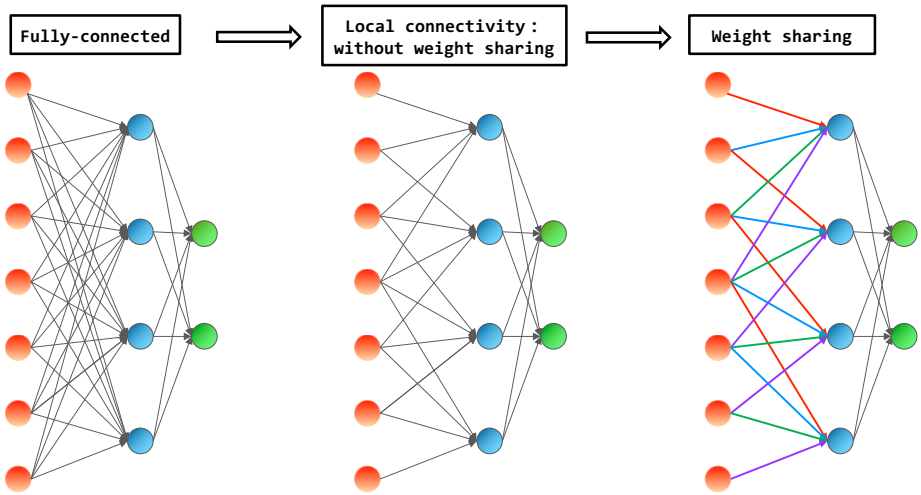


图 10.8: 不同网络结构的示意图。

f^*	CNN	CNN (无权重共享)	全连接网络
$\{0\}$	0	0	0
$\mathcal{F}_1^{\text{CNN}} \backslash \{0\}$	$s^2 + (d+1-s)^2$	$(s^2+1)(d+1-s)^2 - (s^2+1)m_{\text{null}}$	$(d^2+1)(d+1-s)^2 - (d^2+1)m_{\text{null}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{F}_k^{\text{CNN}} \backslash \mathcal{F}_{k-1}^{\text{CNN}}$	$k(s^2 + (d+1-s)^2)$	$k(s^2+1)(d+1-s)^2 - (s^2+1)m_{\text{null}}$	$k(d^2+1)(d+1-s)^2 - (d^2+1)m_{\text{null}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{F}_m^{\text{CNN}} \backslash \mathcal{F}_{m-1}^{\text{CNN}}$	$m(s^2 + (d+1-s)^2)$	$m(s^2+1)(d+1-s)^2 - (s^2+1)m_{\text{null}}$	$m(d^2+1)(d+1-s)^2 - (d^2+1)m_{\text{null}}$

表 10.1: 三种架构的乐观样本量

一般来说，我们将从乐观估计中得到的模型属性称为它们的乐观属性。上面，我们揭示了两层神经网络的两个乐观属性——宽度的自由表达能力和连接的昂贵表达能力。这些属性提出了神经网络的两个架构设计原则：(i) 随意添加神经元/核，和 (ii) 限制连接神经元。这两个原则与常见的做法相符，即人们宁愿扩大宽度（或核数）而不是添加连接来增加神经网络的表达能力。

10.5 多层神经网络的乐观样本量*

在前面的章节，我们讨论了两层神经网络的模型秩。在这一节我们进一步介绍如何估计多层神经网络的模型秩。事实上，准确的估计多层网络的乐观样本量大小是较为困难的，这里我们给出乐观样本量的两个上界 (见定理 7、定理 8、表格 10.2)。

在推论 7 中，我们将模型的参数量作为乐观样本量的上界。由于模型的参数量与目标函数无关，所以我们说这个上界是“通用的”。不过，天下没有免费的午餐。由于这个上界中缺失了目标函数的信息，所以这个上界是相当松的。在定理 8 中，我们利用之前章节中发展出的嵌入原则证明，对于多层神经网络，宽网络的乐观样本量一定不会比窄网络更大。也就是说，我们可以任意增加神经网络的宽度，而不必担心乐观样本量增加，这与两层神经网络中的“宽度的免费表达能力”相符。在表格 10.2 中，我们列出了对于不同的目标函数 f^*，从定理 7 和定理 8 中得到的两个乐观样本量上界。为了简化表述，表格中考虑的是没有偏置项的全连接深度神经网络 (DNN)。网络的参数总数是 $M = dm_1 + m_1m_2 + \cdots + m_{L-2}m_{L-1} + m_{L-1}$。在表格中，$\mathcal{F}_{\{m_i\}_{i=1}^{L-1}}$ 表示隐藏层宽度为 $\{m_i\}_{i=1}^{L-1}$ 的 L 层 DNN 的函数空间。$\mathcal{F}_{\{m_i'\}_{i=1}^{L-1}}$ 表示能被宽度为 $\{m_i'\}_{i=1}^{L-1}$ 的 L 层 DNN 表示的函数全体。可以看到，对于能用较窄网络表示的目标函数，定理 8 得到的乐观样本量上界比定理 7 紧很多。

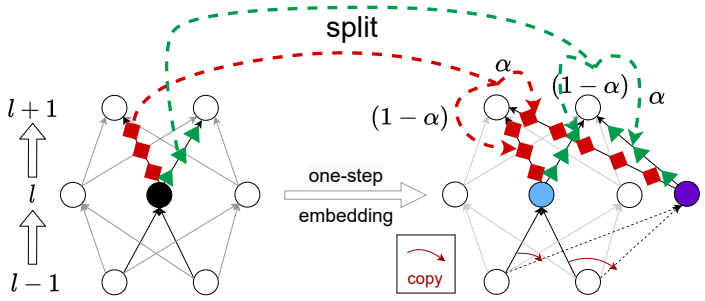


图 10.9: （参考文献 Zhang, Zhang, Luo & Xu (2021) 中的图 2）一步分裂嵌入示意图。

定理 7 (乐观样本量的通用上界)**.** 假设我们有一个可微模型 $f_{\boldsymbol{\theta}}$。对于函数空间 $\mathcal{F}$ 中的任何函数 f^*，我们有 $O_{f_{\boldsymbol{\theta}}}(f^*) \leqslant M$. ($M$ 是 $\boldsymbol{\theta}$ 的维数)

定理 8 (深度神经网络乐观样本量的上界)**.** 对于任意具有 M_{wide} 参数的神经网络，对于较窄神经网络函数空间中的任意函数，以及任意 $f^* \in \mathcal{F}_{\text{narr}}$，我们有 $O_{f_{\boldsymbol{\theta}_{\text{wide}}}}(f^*) \leqslant O_{f_{\boldsymbol{\theta}_{\text{narr}}}}(f^*) \leqslant M_{\text{narr}}$。

证明. 记 $r = O_{f_{\boldsymbol{\theta}_{\text{narr}}}}(f^*)$。由乐观样本量的定义，能找到一个窄网络 $f_{\boldsymbol{\theta}_{narr}}$，它的输出等于 f^*，并且它的切空间维数是 r。接下来我们将该窄网络嵌入为宽网络。图10.9是一步分裂嵌入的示

意图。左侧网络中的黑色神经元在右侧网络中分裂成蓝色和紫色神经元。左侧网络中黑色神经元的红色（绿色）输出权重在右侧网络中分裂成两个红色（绿色）权重，比例分别为 α 和 $(1-\alpha)$。利用图 10.9中的分裂嵌入，将该窄网络嵌入为宽网络 $f_{\boldsymbol{\theta}_{wide}}$。由于嵌入不改变切空间，所以 $f_{\boldsymbol{\theta}_{wide}}$ 的切空间维数也是 r，并且 $f_{\boldsymbol{\theta}_{wide}}$ 的输出也等于 f^*。由乐观乐观样本量的定义，$O_{f_{\boldsymbol{\theta}_{\text{wide}}}}(f^*) \leq r$。 □

model: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L]}\sigma(\cdots\sigma(\boldsymbol{W}^{[1]}\boldsymbol{x})\cdots)$, $\boldsymbol{W}^{[l]} = \mathbb{R}^{m_l \times m_{l-1}}$,$m_L = 1$, $m_0 = d$		
f^*	通用的上界 (推论 7)	嵌入得到的上界 (定理 8)
$\mathcal{F}_{\{1,1,\cdots,1\}}$	M	$d+L-1$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{F}_{\{m_i'\}_{i=1}^{L-1}}, 1 \leq m_i' \leq m_i$	M	$dm_1' + m_1'm_2' + \cdots + m_{L-2}'m_{L-1}' + m_{L-1}'$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{F}_{\{m_i\}_{i=1}^{L-1}}$	M	M

表 10.2: 宽度为 $\{m_i\}_{i=1}^{L-1}$ 的 L 层全连接深度神经网络（DNN）的乐观样本量上界。

10.6 两层神经网络全局极小点的几何结构**

在乐观估计中，我们使用线性近 $f(\cdot;\boldsymbol{\theta}) \approx f^{\text{lin}}(\cdot;\boldsymbol{\theta}) := f^*(\cdot) + \nabla f_{\boldsymbol{\theta}'}(\cdot)^{\mathrm{T}}(\boldsymbol{\theta}-\boldsymbol{\theta}')$ 来分析在小邻域 $N(\boldsymbol{\theta}',\epsilon)$ 中拟合所需的样本量，其中 $\boldsymbol{\theta}' \in \boldsymbol{\Theta}_{f^*}$，然后我们用梯度下降来解决回归问题 $\min_{\boldsymbol{\theta}\in N(\boldsymbol{\theta}',\epsilon)} \frac{1}{n}\sum_{i=1}^n \left(f^{\text{lin}}(\boldsymbol{x}_i;\boldsymbol{\theta}) - f^*(\boldsymbol{x}_i)\right)^2$。当样本量达到乐观样本量时，我们就能恢复 $f^{\text{lin}}(\cdot;\boldsymbol{\theta})$，我们把这种恢复叫做 **“局部线性恢复”**，即模型能够恢复目标函数的线性近似。当然，有时候我们还期待完全恢复目标函数，这就是所谓的 **“局部恢复”**。具体一点，我们现在有 $\boldsymbol{\theta}' \in \boldsymbol{\Theta}_{f^*}$，我们希望在小邻域 $N(\boldsymbol{\theta}',\epsilon)$ 中，采用梯度下降来解决回归问题 $\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{i=1}^n \left(f(\boldsymbol{x}_i;\boldsymbol{\theta}) - f^*(\boldsymbol{x}_i)\right)^2$，最终在训练结束时，模型的输出函数是 $f^*(\boldsymbol{x})$。

显然，局部恢复比局部线性恢复更难，因此，一个合理的推测是，局部恢复需要的样本量大于乐观样本量。进而，我们可以问这样一些问题。**局部恢复的样本量比乐观样本量多多少呢？在过参数化下，有没有可能实现局部恢复？**我们对于两层神经网络的局部恢复进行了理论分析，并发现局部恢复所需的样本量和全局极小的结构密切相关。

10.6.1 符号和假设*

我们关注一个两层神经网络

$$f:\mathbb{R}^{(d+1)m}\times\mathbb{R}^d\to\mathbb{R},\quad f(\boldsymbol{\theta},\boldsymbol{x})=\sum_{k=1}^{m}a_k\sigma(\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}),$$

这里 $m\in\mathbb{N}$ 是固定的，通常被称为 g 的宽度，$\boldsymbol{x}\in\mathbb{R}^d$ 是 g 的输入，模型参数 $\boldsymbol{\theta}\in\mathbb{R}^{(d+1)m}$ 中。我们选取目标函数 f^* 为一个该神经网络能表示的目标函数，即：

$$f^*(\boldsymbol{x})=\sum_{k=1}^{m_0}\bar{a}_k\sigma(\bar{\boldsymbol{w}}_k\cdot\boldsymbol{x}),,m\geq m_0$$

我们定义损失函数为通常的均方损失：在这种情况下，我们有

$$R(\boldsymbol{\theta})=\sum_{i=1}^{n}|f(\boldsymbol{\theta},\boldsymbol{x}_i)-f^*(\boldsymbol{x}_i)|^2=\sum_{i=1}^{n}\left|\sum_{k=1}^{m}a_k\sigma(\boldsymbol{w}_k\cdot\boldsymbol{x}_i)-\sum_{k=1}^{m_0}\bar{a}_k\sigma(\bar{\boldsymbol{w}}_k\cdot\boldsymbol{x}_i)\right|^2. \tag{10.20}$$

注 29. 由于神经网络能表示 f^*，因此存在某个 $\boldsymbol{\theta}\in\mathbb{R}^{(d+1)m}$ 使得 $f(\boldsymbol{\theta},\cdot)=f^*$。我们定义 **目标集** $Q^*:=\{\boldsymbol{\theta}\in\mathbb{R}^{(d+1)m}:f(\boldsymbol{\theta},\cdot)=f^*\}$。显然 $Q^*\subseteq R^{-1}\{0\}$，其中 $R^{-1}\{0\}$ 表示损失函数 R 的零点集，也即能够完美拟合数据的所有全局最小值。

10.6.2 一个两神经元模型的例子*

考虑一个具有指数激活函数和二维输入的两个神经元网络，即 $f(\boldsymbol{\theta},x)=a_1e^{\boldsymbol{w}_1\cdot\boldsymbol{x}}+a_2e^{\boldsymbol{w}_2\cdot\boldsymbol{x}}$，其中 $\boldsymbol{x}\in\mathbb{R}^2$ 和 $\boldsymbol{\theta}=(a_1,\boldsymbol{w}_1,a_2,\boldsymbol{w}_2)\in\mathbb{R}^6$。考虑训练数据 $\{(\boldsymbol{x}_i,y_i)\}_{i=1}^n$，因此损失函数为

$$R(\boldsymbol{\theta})=\sum_{i=1}^{n}\left|f(\boldsymbol{\theta},\boldsymbol{x}_i)-y_i\right|^2.$$

我们选取目标函数 $f^*(\boldsymbol{x})=\bar{a}e^{\bar{\boldsymbol{w}}\cdot\boldsymbol{x}}$ $(\bar{a}\neq0)$。损失函数写为

$$R(\boldsymbol{\theta})=\sum_{i=1}^{n}\left|f(\boldsymbol{\theta},\boldsymbol{x}_i)-\bar{a}e^{\bar{\boldsymbol{w}}\cdot\boldsymbol{x}_i}\right|^2.$$

首先，注意到 $e^{\boldsymbol{w}_1\cdot\boldsymbol{x}}$ 和 $e^{\boldsymbol{w}_2\cdot\boldsymbol{x}}$ 是线性无关的当且仅当 $\boldsymbol{w}_1 \neq \boldsymbol{w}_2$。因此，根据不同的 $\boldsymbol{w}_k$ 的数量，我们有一个目标集的划分 $Q^* = Q_1 \cup Q_2 \cup Q_3$，这和例 1中的划分一样，其中

$$
\begin{aligned}
Q_1 &= \{(a, \bar{\boldsymbol{w}}, \bar{a} - a, \bar{\boldsymbol{w}}) : a \in \mathbb{R}\} \\
Q_2 &= \{(\bar{a}, \bar{\boldsymbol{w}}, 0, w) : \boldsymbol{w} \in \mathbb{R}^2 \setminus \{\bar{\boldsymbol{w}}\}\} \\
Q_3 &= \{(0, \boldsymbol{w}, \bar{a}, \bar{\boldsymbol{w}}) : \boldsymbol{w} \in \mathbb{R}^2 \setminus \{\bar{\boldsymbol{w}}\}\}.
\end{aligned}
$$

注意到 Q_1, Q_2, Q_3 的闭包都是 $\mathbb{R}^6$ 中不同维度的仿射子空间。一般来说，当样本大小 $n \geq 6$ 时，我们预期目标集 Q^* 等于全局极小 $R^{-1}\{0\}$，因此当神经网络收敛到零损失时，它的泛化误差为 0。然而，在过参数化时，即 $n < 6$，通常存在泛化误差不为 0 的全局最小点，这引发了一个问题，**即每个目标集的分支是否被泛化不为 0 的全局最小点所“包裹”**。如果不是，那么在该目标集附近，训练的动力学显然有机会收敛到该目标集，从而实现完美的泛化。受此例启发，我们提出了以下关于 $R^{-1}\{0\}$ 和 Q^* 的问题，以理解实现完美泛化的可能性。

(a) **分支：**我们如何在几何上描述 Q^*？什么决定了它的分支？

(b) **分离：**Q^* 与 $R^{-1}\{0\}$，特别是与泛化不为 0 的全局最小值 $R^{-1}\{0\} \setminus Q^*$ 有什么关系？它可以从 $R^{-1}\{0\} \setminus Q^*$ 中“分离”出来吗？这如何依赖于样本？

(c) **收敛：**我们能对梯度流在 $R^{-1}\{0\}$ 附近的收敛、收敛率、收敛方向等说些什么？此外，对于收敛到 Q^* 中的点的梯度流，有没有什么特殊的性质？

我们在以下的 10.6.3 节中总结了对它们的答案，其中非正式的定理（定理 9，10，和11）分别回答了问题 (a)，(b)，(c)（正式定理见Zhang, Zhang & Luo (2023)）。理解下面的这些理论结果可能稍显困难，不是特别感兴趣的读者可以直接跳到 10.6.4 节，我们通过具体的例子展示了对这些理论结果的理解。

10.6.3 一般的理论结果**

对于 Q^* 的几何结构，我们有以下理论结果：

定理 9 (Q^* 的分支)**.** *设 σ 为一般的激活函数。在较弱的假设下，存在某个 $N \in \mathbb{N}$，使得 $Q^* = \bigcup_{t=1}^{N} Q_t \subseteq R^{-1}\{0\}$，每个“分支” Q_t 都是仿射子空间 $\tilde{Q}_t \subseteq \mathbb{R}^{(d+1)m}$ 减去一些 $\tilde{Q}_t$ 的适当的仿射子空间。所有的 Q_t 都是“内在的”。*

这里说 Q_t 是内在的，我们的意思是它们只依赖于目标 f^* 和模型 g，而不依赖于数据点的选择。例如，这些部分的数量 N 只依赖于 m_0 和 m。进一步地，通过对损失函数 R 的局部秩进行分析，我们有以下结果。

定理 10 (Q^* 中分支的分离)**.** *按照定理 9中的假设和符号，每个分支 Q_t 对应一个样本大小阈值 $N_t \le m(d+1)$（如果 $m > m_0$，我们有 $N_t < m(d+1)$），当样本大小 $n \ge N_t$ 时，Q_t 从不完美的全局最小值中“分离”出来。此外，通过重新排列 Q_t 的指标，有一个划分*

$$Q^* := \bigcup_{t=1}^{N} Q_t = \left(\bigcup_{t=1}^{N'} Q_t\right) \bigcup \left(\bigcup_{t=N'+1}^{N} Q_t\right)$$

使得无论何时 $t \le N'$ 且 $n \ge N_t$，R 在 Q_t 的任何地方都不是 Morse–Bott 的，而对于 $t > N'$ 且 $n \ge N_t$，R 在 Q_t 的几乎每个地方都是 Morse–Bott 的。

这里说 Q_t 被分离，我们的意思是存在一个开集 $U \subseteq \mathbb{R}^{(d+1)m}$，使得 $U \cap R^{-1}\{0\} = U \cap Q_t$。说 f 是 Morse–Bott 函数，我们是指 $\operatorname{Hess} f$ 在 $(\nabla f)^{-1}\{0\}$ 的构成的流形的法向上是非退化的。最后，任何在 $R^{-1}\{0\}$ 附近的梯度流都有以下性质。

定理 11 (全局最小值附近的梯度流)**.** *按照定理 9和10中的假设和符号，任何足够接近 $R^{-1}\{0\}$ 的梯度流都会收敛。另一方面，$R^{-1}\{0\}$ 中的任何点都是某个梯度流的极限。以下结果成立。*

(a) 无论何时 $t \le N'$ 且样本大小 $n \ge N_t$，足够接近 Q_t 的一般梯度流将收敛到 Q_t 中的一个点 θ^*。收敛没有线性速率，曲线是“偏向于” $\ker \operatorname{Hess} R(\theta^*)$。此外，它的任何小扰动仍然收敛到 Q_t。

(b) 给定 $t > N'$。当样本大小 $n \ge N_t$ 时，任何足够接近 Q_t 的梯度流将以线性速率收敛到 Q_t 的点。类似于 (a)，它的任何小扰动仍然收敛到 Q_t。

这三个（非正式的）定理全面展示了在超参数化情况下（即，当 $n < (d+1)m$ 时）全局最小值的几何结构和局部动力学的结构变化，其中完美的全局最小值 Q^* 是其主干。特别是，分支分离和收敛结果保证了两层神经网络的局部恢复能力，即，当在分离的分支附近初始化时，目标函数将被恢复。此外，我们的结果表明了一种在过参数化下的泛化机制：通过适当的通用初始化和超参数调整，可以将梯度动力学全局地引导到 Q^* 的分离分支的邻域，从而在收敛时恢复目标函数。

10.6.4 两神经元模型的例子解释*

下面，我们使用之前的例子中说明以上的结果。使用上述的三个定理以及一些计算，我们现在回答第 10.6.2节中我们的两个神经元模型例子的三个问题。

(a) 从定理 9，Q^* 是若干个 $\mathbb{R}^4$ 的子集的并，其闭包是仿射子空间。具体来说，Q_1 是一维的仿射子空间，而 Q_2, Q_3 都是二维的仿射子空间减去一个点。这与我们上面的观察以及图 18.3一致。

(b) 通过定理 10，每个 Q_t，$1 \le t \le 3$，都对应一个使其分离的样本大小 N_t。具体来说，我们有 $N_1 = 5$ 和 $N_2 = N_3 = 4$。此外，当样本大小 $n \ge N_t$ 时，如果 $t = 1$，R 在 Q_t 的任何地方都不是 Morse–Bott 的，如果 $t = 2, 3$，R 在 Q_t 的几乎每个地方都是 Morse–Bott 的。

(c) 根据定理 11，当样本量 $n \ge 5$ 时，Q^1 附近的梯度流会收敛到 Q_1 中的一个点 θ^*，并且是 “偏向于”$\ker \operatorname{Hess} R(\theta^*)$，但是收敛速率小于线性速率。当 $n \ge 4$，任何收敛到 $Q_2 \cup Q_3$ 中的点的梯度流 γ 都有线性速率。

图 18.3 形象地描绘了 Q^* 和附近的梯度流的这些性质。如 (a) 部分所描述，Q^* 由几个集合组成，其闭包是一维的仿射子空间。由 (b) 部分描述，损失 R 在 Q_1 的任何点附近都不是 Morse–Bott 的，因此由 (c) 部分描述，一个以 Q_1 中的点 (图中的 $\boldsymbol{\theta}_1^*$) 为极限的梯度流通常是 “偏向于”$\ker \operatorname{Hess} R(\boldsymbol{\theta}_1^*)$。另一方面，$R$ 在 Q_2 和 Q_3 的几乎处处都是 Morse–Bott 的，因此一个以 $Q_2 \cup Q_3$ 中的点 (图中的 $\boldsymbol{\theta}_2^*, \boldsymbol{\theta}_3^*$) 为极限的梯度流通常以线性速率收敛。最后，注意 $Q_{12} = (\bar{a}, \bar{\boldsymbol{w}}, 0, \bar{\boldsymbol{w}})$ 和 $Q_{13} = (0, \bar{\boldsymbol{w}}, \bar{a}, \bar{\boldsymbol{w}})$ 分别是交点 $\overline{Q_1} \cap \overline{Q_2}$ 和 $\overline{Q_1} \cap \overline{Q_3}$。

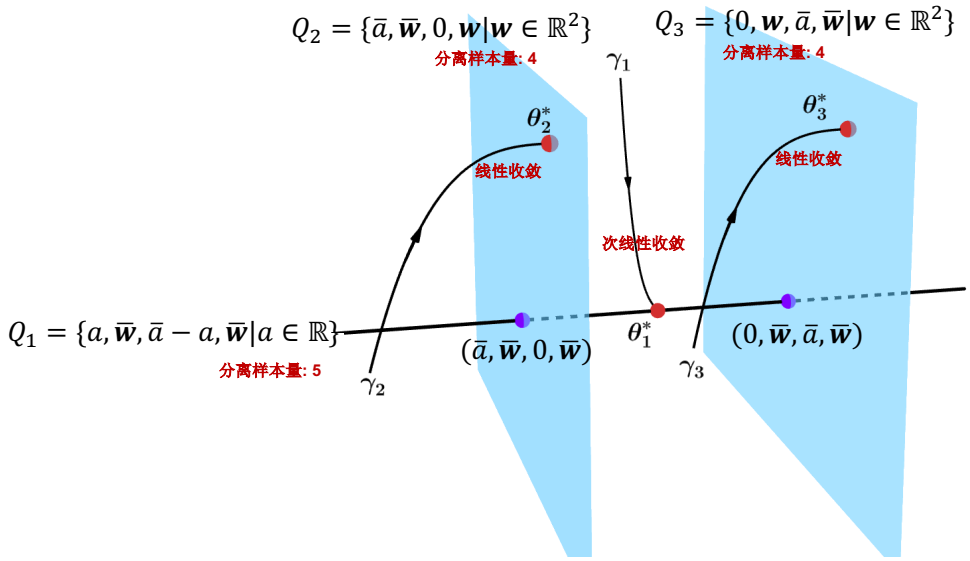


图 10.10: 描绘两神经元模型拟合一个神经元目标函数的例子。

10.7 习题

1. 对以下模型进行秩分析和乐观样本量估计：

   (i) linear model: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{n} a_i x_i$

   (ii) deep model: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{n} a_i b_i x_i$

2. 当凝聚现象发生时神经网络的 model rank 如何变化?

3. (思考题) 凝聚现象、嵌入原则和乐观估计之间有什么关系?

4. 对于固定偏置的两层神经网络 $f_{\boldsymbol{\theta}}(x) = a_1 \tanh(\boldsymbol{w}_1 \cdot \boldsymbol{x} + 1) + a_2 \tanh(\boldsymbol{w}_2 \cdot \boldsymbol{x} + 1)$, 其中 $\boldsymbol{\theta} = (a_1, \boldsymbol{w}_1, a_2, \boldsymbol{w}_2), \boldsymbol{w}_i \in \mathbb{R}^2, a_i \in \mathbb{R}, \boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2$。设目标函数为 $f^*(x) = \bar{a} \tanh(\bar{\boldsymbol{w}} \cdot \boldsymbol{x})$。则有 $\boldsymbol{\Theta}_{f^*} = Q_1 \cup Q_2 \cup Q_3$ 其中

   $$Q_1 = \{(a, \bar{\boldsymbol{w}}, \bar{a} - a, \bar{\boldsymbol{w}}) : a \in \mathbb{R}\}$$
   $$Q_2 = \{(\bar{a}, \bar{\boldsymbol{w}}, 0, w) : w \in \mathbb{R}^2$$
   $$Q_3 = \{(0, w, \bar{a}, \bar{\boldsymbol{w}}) : w \in \mathbb{R}^2$$

   请计算 $\boldsymbol{\theta}' \in Q_1$、$\boldsymbol{\theta}' \in Q_2$、$\boldsymbol{\theta}' \in Q_3$ 时，模型秩 $R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}')$ 的大小。

5. 对于两层神经网络 $f_{\boldsymbol{\theta}}(x) = a_1 \tanh(w_1 x + b_1) + a_2 \tanh(w_2 x + b_2)$，其中 $\boldsymbol{\theta} = (a_1, w_1, b_1, a_2, w_2, b_2) \in \mathbb{R}^6$，$x \in \mathbb{R}$。设目标函数为 $f^*(x) = \tanh(x + 1)$。请探究目标集 $\boldsymbol{\Theta}_{f^*}$ 的结构。

6. 对于两层神经网络 $f_{\boldsymbol{\theta}}(x) = a_1 \tanh(w_1 x + b_1) + a_2 \tanh(w_2 x + b_2)$，其中 $\boldsymbol{\theta} = (a_1, w_1, b_1, a_2, w_2, b_2) \in \mathbb{R}^6$，$x \in \mathbb{R}$。设目标函数为 $f^*(x) = \tanh(x + 1)$，请问：

   (i)：f^* 的乐观样本量是多少?

   (ii)：当样本个数恰为乐观样本量时，并且初始化为乐观初始化时，模型 $f_{\boldsymbol{\theta}}(x)$ 是否能恢复目标函数 f^*? 请做实验探究一下。

   (iii)：当样本个数恰为乐观样本量时，并且初始化为小初始化时，模型 $f_{\boldsymbol{\theta}}(x)$ 是否能恢复目标函数 f^*? 请做实验探究一下。

7. 有一个两层神经网络 $f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \sigma(w_k x + b_k)$，其中 $\boldsymbol{\theta} = (a_k, w_k, b_k)_{k=1}^{n}, \sigma(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$。网络的宽度 $m \geq 100$。用该网络来恢复目标函数 $f^*(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$。请问乐观样本量是多少?

8. 有一个随机特征网络 $f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \tanh(w_k x + b_k)$，其中 $\boldsymbol{\theta} = (a_k)_{k=1}^{n}, \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$。它与神经网络的区别是，神经元的输入权重 $(w_k, b_k)_{k=1}^{m}$ 不训练，它们的值是由初始化决定的。假设该随机特征网络的宽度 $m \geq 100$，并且 $w_1 = b_1 = 1$。用它来恢复目标函数 $f^*(x) = \tanh(x + 1)$。请问乐观样本量是多少?

Chapter 11

Dropout 的隐式正则化

在上文的介绍中，我们提到神经网络的非线性训练过程中存在凝聚现象，即在训练过程中，同层神经元会趋同。我们在相图分析中提到，要想发生凝聚现象，往往需要比较小的权重初始化。在实践中，我们常常不会使用非常小的参数初始化，因为太小的初始化会使训练经历很长的平台期，使训练时间变得很长。实际中，梯度下降法往往会搭配一些正则化技术来使用，以提高模型的泛化性能。正则化是一类通常通过减小模型复杂度来防止模型过拟合的技术。它的原理是通过在训练目标中引入额外的约束项或惩罚项，使模型更倾向于学习到满足某个特定性质的解。具体选择何种正则化方法需要根据任务的特点和需求而定。

在本章中，我们发现一种常用的正则化技术——dropout，可以在高效训练的同时使神经网络在整个训练过程中都趋向凝聚状态。这是 dropout 得以提升模型泛化能力的一种重要因素。在接下来的内容中，我们将首先通过实验论证 dropout 具有促进凝聚的隐式正则化效果，然后尝试从理论上给出解释，并讨论其在引导凝聚的效率、降低乐观样本量等方面的有效性 (Zhang & Xu 2022)。

科普篇

Dropout：神经网络的随机“丢弃”机制

在深度学习领域，Dropout 是一种广泛使用的技术，它通过在训练过程中随机“关闭”一些神经元，同时放大其它神经元的输出来提高模型的泛化能力。这种方法可以比喻为给神经网络中的部分神经元安排“休假”，从而防止模型过拟合，提高模型泛化性。

Dropout 的工作原理是什么？

想象一下，在每次训练迭代中，网络中的每个神经元都有一定概率（通常为 p）被暂时“关闭”。这就像是在一个大公司中，每天都有一部分员工随机休假，剩下的员工要尽量能够也拥有休假员工的能力，也就是员工之间的能力应该类似。这种机制迫使网络不能过度依赖于某些特定的神经元，而是必须学会更加灵活和鲁棒的表示方式。在实际应用中，Dropout 通过以下步骤实现：

在训练过程中，对于每批样本，随机选择一部分神经元（概率为 p）保持激活，其他的则被“关闭”。对于激活的神经元，其输出要乘以 $1/(p)$，以平衡训练和测试时的期望输出。

- 前向传播时，只使用激活的神经元进行计算。
- 反向传播时，只更新激活神经元的权重。
- 在测试阶段，所有神经元都参与计算。

如何从理论层面理解 Dropout？

Dropout 的期望效果可以分解为两部分：原始的损失函数和一个额外的正则化项 $R_1(\boldsymbol{\theta})$

这个正则化项 $R_1(\boldsymbol{\theta})$ 能够捕捉到网络参数之间的相互关系，从而促进模型凝聚并提高模型平坦性。这种理论解释帮助我们更好地理解为什么 Dropout 能够有效地防止过拟合，并提高模型的泛化能力。

11.1 Dropout

Dropout 是一种常用于训练神经网络的技术 (Srivastava et al. 2014), 它能够显著提高模型的泛化能力 (Tan & Le 2019)。Dropout 的基本工作原理是: 对于网络中的每个神经元, 以概率 p 将其输出乘以 $1/p$, 或者以概率 $1-p$ 将其输出置零, 并在每次前向传播过程中独立地进行这一随机操作。

具体来说, 考虑神经网络的第 l 层, 其输出为 $\boldsymbol{f}_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x}) \in \mathbb{R}^{m_l}$。在应用 dropout 时，在训练的每一步开始时，我们首先采样一个长度为 m_l 的随机向量 $\boldsymbol{\eta}$, 其中每个元素 $(\boldsymbol{\eta})_k$ 满足:

$$(\boldsymbol{\eta})_k = \begin{cases} \frac{1-p}{p} & \text{以概率} p \\ -1 & \text{以概率} 1-p \end{cases},$$

这里 $k \in [m_l]$ 是 $\boldsymbol{\eta}$ 的索引。容易验证,$\boldsymbol{\eta}$ 是一个均值为 0 的随机变量。接下来, 我们通过以下计算给第 l 层的输出施加 dropout:

$$\boldsymbol{f}_{\boldsymbol{\theta},\boldsymbol{\eta}}^{[l]}(\boldsymbol{x}) = (\mathbf{1}+\boldsymbol{\eta}) \odot \boldsymbol{f}_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x}),$$

其中 $\odot$ 表示 Hadamard 积, 即两个同维向量或矩阵的逐元素乘积。在后续的计算中, 我们用 $\boldsymbol{f}_{\boldsymbol{\theta},\boldsymbol{\eta}}^{[l]}(\boldsymbol{x})$ 替代原来的 $\boldsymbol{f}_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x})$, 作为第 l 层的输出。

为了简化表示, 我们用 $\boldsymbol{\eta}$ 统一表示网络所有层上的 dropout 向量。对于给定的输入 $\boldsymbol{x}$ 和噪声 $\boldsymbol{\eta}$, 模型 $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$ 在 dropout 作用下的输出记为 $\boldsymbol{f}_{\boldsymbol{\theta},\boldsymbol{\eta}}^{\text{drop}}(\boldsymbol{x})$。相应地, 网络在训练集 $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ 上的损失函数写为:

$$R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta}) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(\boldsymbol{f}_{\boldsymbol{\theta},\boldsymbol{\eta}}^{\text{drop}}\left(\boldsymbol{x}_i\right), \boldsymbol{y}\left(\boldsymbol{x}_i\right)\right) = \mathbb{E}_S \ell\left(\boldsymbol{f}_{\boldsymbol{\theta},\boldsymbol{\eta}}^{\text{drop}}(\boldsymbol{x}), \boldsymbol{y}\right).$$

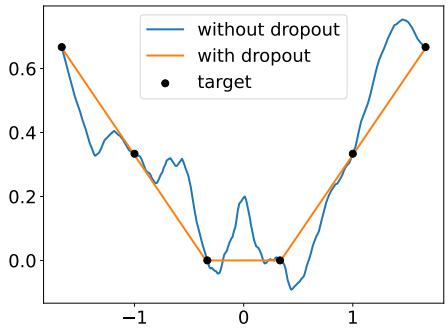


图 11.1: 训练有无 dropout 的两层 ReLU 神经网络的实验结果

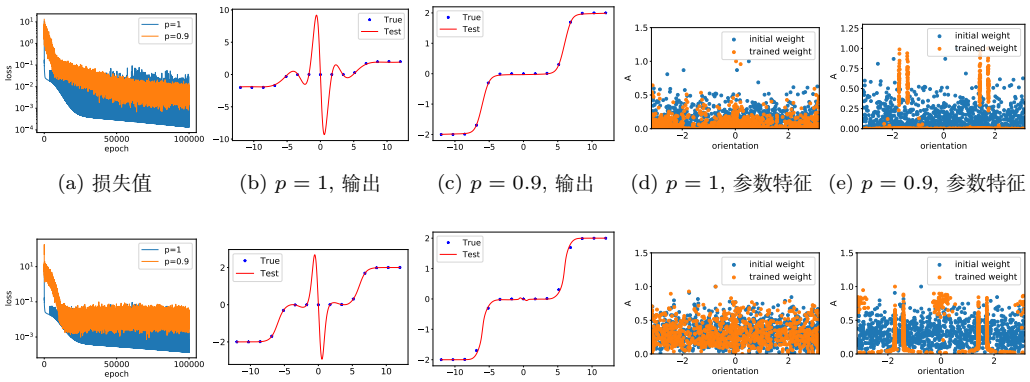
为了直观理解 dropout 对模型训练的影响, 我们首先在一个简单的回归任务上进行实验。图11.1展示了一个大初始化（线性区域的设定 (Jacot et al. 2018)）的神经网络在有无 dropout 时的拟合情况。隐藏层的宽度为 1000，所有实验的学习率均为 1×10^{-3}。图中蓝色曲线和橙色曲线分别是有 dropout 及没有 dropout 的神经网络的输出，其中黑点表示目标点。可以看到, 如果不使用 dropout, 模型倾向于用一个剧烈振荡的曲线去拟合给定的离散点, 这与我们在相图分析中看到的结果一致。而加入 dropout 后, 神经网络学到的曲线则表现出明显的分段特征, 暗示模型内部参数可能发生了凝聚。受此启发，在下一节中，我们观察到 dropout 对凝聚具有促进作用，并以此作为切入点，理解 dropout 的正则化行为。

11.2 Dropout 促进神经元凝聚

11.2.1 一维实验

本节我们将在不同的设定下研究 dropout 对神经元凝聚的促进作用, 并尝试解释其产生凝聚的内在机制。首先, 让我们考虑一个一维函数拟合的简单例子。

如图11.2所示, 我们取目标函数



(a) 损失值 (b) $p=1$, 输出 (c) $p=0.9$, 输出 (d) $p=1$, 参数特征 (e) $p=0.9$, 参数特征

(f) 损失值 (g) $p=1$, 输出 (h) $p=0.9$, 输出 (i) $p=1$, 参数特征 (j) $p=0.9$, 参数特征

图 11.2: 具有 tanh 激活的神经网络在不同 dropout 留存概率下的输出和参数特征

$$f(x) = \sigma(x-6) + \sigma(x+6),$$

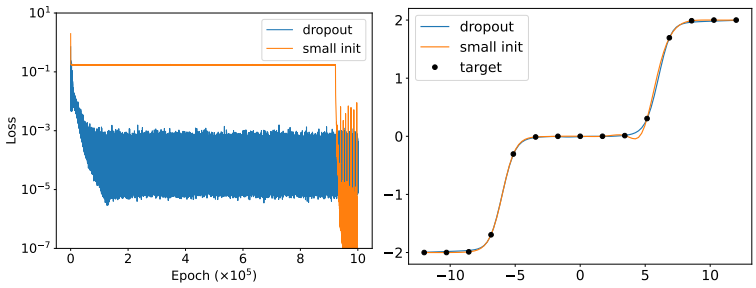
其中 $\sigma(x) = \tanh(x)$。利用由上述一维函数生成的数据, 我们训练隐藏层宽度为 1000 的 tanh 神经网络, 损失函数为均方误差 (MSE)。所有实验的学习率固定为 1×10^{-3}。在 (d,e,i,j) 中, 蓝色和橙色点分别表示训练初期和后期的权重分布。我们均考虑输入层的权重分布。对于两层网络,dropout 层被放置在隐藏层之后 (图11.2上半部分); 而在三层网络中, 两个隐藏层之间和最后一个隐藏层之后都添加了 dropout 层 (图11.2下半部分)。为了更清晰地观察 dropout 对凝聚现象的影响, 我们特意采用了大的参数初始化方式。通过前文的讨论, 我们已经清楚这种初始化本身并不会导致明显凝聚。在相同的初始化条件下, 无论是否使用 dropout, 训练后的网络都能很好地拟合训练数据。

通过比较训练过程, 我们发现使用 dropout 时网络的损失下降并没有出现明显的停滞期 (图11.2(a,f)), 说明参数没有被困在鞍点附近。此外, 从拟合结果可以看出, 不加 dropout 的网络输出存在大量震荡 (图11.2(b,g)), 而使用 dropout 的网络则能学到更加平滑的函数 (图11.2(c,h))。为了更好地理解 dropout 的作用机制, 我们进一步观察了参数空间的分布特征。

借助相图分析中的方法, 我们可以将每个神经元的参数 $(a_j, \boldsymbol{w}_j)$ 分解为单位方向向量 $\hat{\boldsymbol{w}}_j = \boldsymbol{w}_j / \|\boldsymbol{w}_j\|_2$ 和一个表示其对输出贡献的振幅 $A_j = |a_j| \|\boldsymbol{w}_j\|_2$, 即 $\{(\hat{\boldsymbol{w}}_j, A_j)\}_{j=1}^m$。我们对神经网络的参数分布 $\{(\hat{\boldsymbol{w}}_j, A_j)\}_{j=1}^m$ 作散点图 (图 11.2(d,e,i,j)), 其中横轴表示每个 $\hat{\boldsymbol{w}}_j$ 相对于 x 轴的角度, 纵轴表示神经元振幅 A_j。为了便于比较, 每个模型的参数进行归一化, 使其模长最大神经元的模长为 1。可以看到, 不使用 dropout 时, 训练后的权重分布 (橙色) 与初始

分布 (蓝色) 接近; 相比之下, 在使用 dropout 的情况下, 训练后的非零权重向量在某些离散的方向上形成了明显的聚集, 呈现出凝聚的趋势。

下面，我们简单理解一下 dropout 为何引导凝聚。dropout 的基本思想是在每个训练步骤中随机丢弃一部分神经元，并用同层其他神经元进行补偿。举一个简单的例子, 如果在某次训练中, 某层有一半的神经元被随机遮盖, 那这一层的总输出幅值肯定会大幅下降。为了补偿这种损失, 我们会将留存神经元的输出乘以 2, 然后再进行梯度下降训练。如果训练收敛到一个稳定的解, 最理想的情况是那些被遮盖的神经元和留存的神经元在功能上尽量一致, 这样 dropout 的遮盖对网络行为的影响能降到最低。这种具有神经元一致性的网络状态正是我们在实验中所观察到的凝聚现象。



(a) 损失函数　　(b) 模型输出

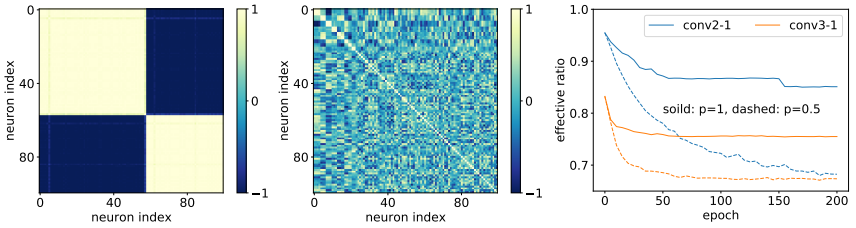
图 11.3: 小初始化下的梯度下降训练 (橙色) 与正常初始化但使用 dropout 的训练 (蓝色) 在损失和输出函数上的对比

一个自然的问题是，通过小初始化引导凝聚的方式与 dropout 有何差异? 在介绍嵌入原则时, 我们观察到, 在小初始化设定下，凝聚的发生往往伴随着损失停滞现象, 即在损失函数关于训练步数的曲线中, 损失值会在某个值附近徘徊很久。从相图分析的视角来看, 我们需要在初始化权重大小上进行权衡。较小的初始化会使凝聚现象更加明显, 但同时会减慢训练速度; 而较大的初始化虽然能加快训练, 但会使神经网络的行为趋近于随机特征模型。

相比小初始化, dropout 为我们提供了一种更理想的训练方式, 在不显著增加训练时间的情况下就能诱导出凝聚现象。例如, 在图11.3(a) 中, 我们对在大初始化中使用 dropout 训练的模型和在凝聚区域初始化的模型的训练过程及网络输出进行比较。我们发现 dropout 训练 (蓝色) 在大约 1×10^5 个 epoch 内就能将损失降至 10^{-5} 量级, 远少于小初始化训练 (橙色) 所需的迭代次数。同时, dropout 训练得到的模型输出也更加平滑 (图11.3(b))。相比之下, 对于具有大初始化的模型, 如果不使用 dropout, 即便参数规模已经足够大, 最终学到的函数依然存在明显的震荡, 且不会出现凝聚现象 (图11.2(b,g))。

11.2.2 高维实验

接下来, 我们将目光转向更高维的数据。这里以教师-学生设定[1]下的两层 tanh 网络为例, 其中教师网络具有 10 维输入和 1 个隐藏单元。学生网络的输入维度、深度与教师保持一致, 但隐藏层宽度远大于 1。为了刻画任意两个隐藏单元之间的相似程度, 我们计算两两神经元归一化输入权重向量的内积, 即余弦相似度。如图11.4(a,b) 所示, 对于有 dropout 的学生网络, 隐藏层神经元在参数空间中只形成了两个方向相反的凝聚方向, 与我们构造的教师网络的凝聚方向一致。而在没有使用 dropout 训练的模型中, 则完全不出现凝聚现象。



(a) $p = 0.5$, 参数分布　(b) $p = 1$, 参数分布　(c) 有效神经元比率

图 11.4: 高维数据上 dropout 对网络稀疏性的影响

在更加复杂的高维数据和网络架构下, 直接可视化参数分布的方法难以清晰地反映凝聚程度随训练进程的变化。为了定量刻画这一动态过程, 我们引入了有效神经元比率这一指标。

定义 10 (有效神经元比率). *对于一个给定的神经网络, 令 $\boldsymbol{\theta}_j^{[l]} \in \mathbb{R}^{m_l}$ 表示第 l 层第 j 个神经元的输入权重向量。定义单位向量集合 $U^{[l]} = \{\boldsymbol{u}_k^{[l]}\}_{k=1}^{m_l'}$, 其中 m_l' 为集合 $U^{[l]}$ 的元素个数, 使得对于任意的 $\boldsymbol{\theta}_j^{[l]}$, 总存在 $\boldsymbol{u} \in U^{[l]}$ 满足 $\boldsymbol{u} \cdot \boldsymbol{\theta}_j^{[l]} / ||\boldsymbol{\theta}_j^{[l]}||_2 > 0.95$。我们称满足条件的 $U^{[l]}$ 中元素个数最少的集合的元素个数为第 l 层的有效神经元数目 m_l^{eff}, 并将 m_l^{eff}/m_l 定义为该层的有效神经元比率, 其中 m_l 为第 l 层宽度。*

图11.4(c) 展示了在 CIFAR-10 图像分类任务上, 使用 ResNet-18(He et al. 2015) 作为模型, 并使用默认的 kaiming uniform(He et al. 2016) 初始化方式, 比较在有无 dropout 的情况下网络各层有效神经元比率的变化情况。图中 Conv2-1 和 Conv3-1 分别表示 ResNet 第 2 个和第 3 个 block 的第一个卷积层。训练中我们使用 Adam 优化器, 学习率为 0.001, batch size 为 128, 并采用 kaiming 初始化。具体地, 对于使用 dropout 的网络, 我们在每个残差块后的 ReLU 激活之后插入一个留存概率为 $p = 0.5$ 的 dropout 层。可以看到, 加入 dropout 后（即 $p = 0.9$）, 各层的有效神经元比率在整个训练过程中始终远低于不使用 dropout 的对照组（即 $p = 1$）, 这表明 dropout 能够持续地促进网络凝聚。

[1]对任意输入数据, 将教师网络的输出作为输入数据的目标值, 学生模型拟合这些教师网络生成的数据。

通过一系列实验, 我们观察到 dropout 在低维和高维数据上都能够有效促进神经元的凝聚现象。这种凝聚在保证训练效率的前提下, 使其输出变得平滑。此外, 实验表明,dropout 诱导的凝聚是伴随训练过程逐渐形成的, 体现了 dropout 作为一种隐式正则化手段的特点。下面，我们将研究该隐式正则化的显式表达，并理解该显式表达与凝聚的关联。

11.3 Dropout 及其隐式正则化的显式表达

本节我们将基于 dropout 所带来的噪声结构, 给出其隐式正则化效应的显式表达形式，并说明 dropout 相当于对每个神经元的输出施加了一个 L2 范数约束。在后面的章节，我们会细致探讨这个显式正则项的意义，特别是它在引导神经网络凝聚方面的贡献。有关该显式表达式的具体推导过程，请参见Zhang & Xu (2022)。

为便于分析, 我们考虑一个 L 层 ($L \geq 2$) 的全连接网络, 并假设 dropout 层只出现在最后一个隐藏层 (即第 $L-1$ 层) 之后。同时, 我们选择均方误差 (mean squared error, MSE) 作为损失函数。在这些假设下, 我们可以将使用 dropout 的模型在数据集 S 上的期望损失 $\mathbb{E}_{\boldsymbol{\eta}}(R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta}))$ 分解为两项之和:

$$\mathbb{E}_{\boldsymbol{\eta}}(R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})) = R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta}),$$

其中 $R_S(\boldsymbol{\theta})$ 为没有 dropout 时的标准经验风险, 即 $R_S(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}(\boldsymbol{x}_i))$, $R_1(\boldsymbol{\theta})$ 则是一个由 dropout 噪声带来的额外正则项, 具体形式为

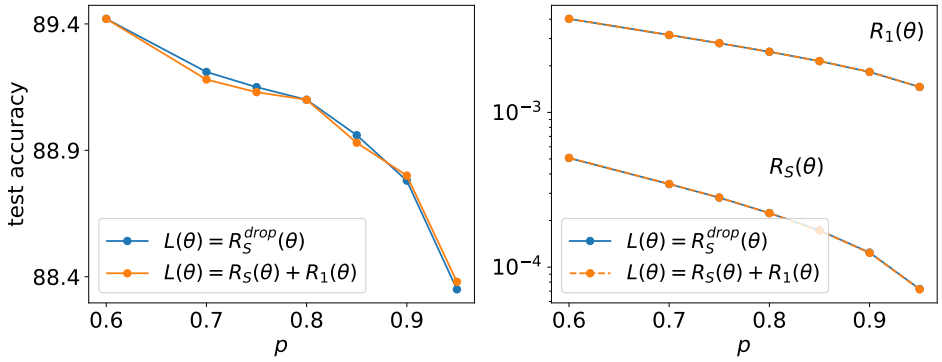
$$R_1(\boldsymbol{\theta}) := \frac{1-p}{2np} \sum_{i=1}^{n} \sum_{j=1}^{m_{L-1}} \sum_{k=1}^{m_L} \left(\boldsymbol{w}_{j,k}^{[L]} f_{\boldsymbol{\theta},j}^{[L-1]}(\boldsymbol{x}_i) \right)^2. \tag{11.1}$$

这里 $\boldsymbol{w}_{j,k}^{[L]}$ 表示最后一层权重矩阵 $\mathbf{W}^{[L]}$ 的第 j 列、第 k 行的元素, 即为最后一层隐藏层第 j 个神经元的输出权重在第 k 维对应的值,$f_{\boldsymbol{\theta},j}^{[L-1]}(\boldsymbol{x}_i)$ 表示最后一个隐藏层输出向量 $\boldsymbol{f}_{\boldsymbol{\theta}}^{[L-1]}(\boldsymbol{x}_i)$ 的第 j 个神经元的输出。

与随机梯度下降 (stochastic gradient descent, SGD) 等优化算法的隐式正则化不同, 正则项 $R_1(\boldsymbol{\theta})$ 是一个与学习率无关的项，这意味着即使在梯度流 (gradient flow) 极限下，即学习率趋于 0 时,dropout 所产生的正则效应也不会消失。从 $R_1(\boldsymbol{\theta})$ 的具体形式可以看出, 它本质上是在惩罚 dropout 所在层每个神经元输出的 L_2 范数的平方和（下一节的简单例子可以容易看出这个关系）。在后续章节中, 我们将展示这种学习率无关的 L_2 约束如何有助于诱导神经元凝聚以及使模型更容易找到平坦的解。在此之前, 我们先通过实验来检验公式(11.1)给出的 $R_1(\boldsymbol{\theta})$ 表达式的准确性。

为验证 $R_1(\boldsymbol{\theta})$ 的正确性, 一种自然的想法是对分别用 $R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})$ 和 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 作为损失函数训练所得模型的性能差异进行比较。鉴于 $R_1(\boldsymbol{\theta})$ 与学习率无关, 我们可以取一个较小

的学习率，以减小损失函数 $R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})$ 动力学的方差，通过梯度流动力学近似模拟 dropout 训练的过程。



(a) (b)

图 11.5: 基于 MNIST 数据集的 dropout 与其隐式正则形式的等价性实验

我们使用具有 1000 个隐藏单元的两层全连接网络作为模型，分别基于 $R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})$ 和 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 两种损失函数进行训练，并固定学习率为 $\varepsilon = 5 \times 10^{-3}$。为了研究不同的 dropout 留存概率 p 对性能的影响，我们进行了一系列实验。图11.5(a) 展示了测试准确率随 dropout 留存概率 p 的变化曲线。这里需要说明的是，对于 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 训练的模型，我们只是将 $R_1(\boldsymbol{\theta})$ 项添加到常用的均方损失函数中，而在前向传播过程并不引入任何 dropout 操作。可以看到，在所有 p 值下，这两类模型的性能几乎完全一致。作为对照，如果单纯基于 $R_S(\boldsymbol{\theta})$ 进行训练，所得模型的分类准确率仅为 79% 左右，明显低于加入 dropout 或 R_1 项之后的结果 (11.5(a) 中均在 88% 以上)。图11.5(b) 进一步比较了 $R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})$ 和 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 训练所得模型在这两个分项上的取值。这里对于两种情况，我们均使用训练完成后的模型参数 $\boldsymbol{\theta}$ 来分别计算 R_S 和 R_1。同样地，我们观察到两种训练方式下，无论 R_S 还是 R_1 的数值都非常接近 (在图中几乎重叠)。

11.4 正则项对凝聚的影响

在上一节中，我们从正则化项 $R_1(\boldsymbol{\theta})$ 可以看出，dropout 正则化对每个神经元的输出施加了额外的 L_2 范数约束。在这一节中我们指出，这一约束与凝聚现象密切相关。为了更直观地理解 $R_1(\boldsymbol{\theta})$ 的作用机制，我们考虑一个简单的两层 ReLU 网络。具体来说，我们使用如下形式

的两层 ReLU 网络来完成一维函数的拟合任务:

$$f_{\boldsymbol{\theta}}(x) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(w_j x + b_j),$$

其中 $\boldsymbol{x} := (x, 1)^\intercal \in \mathbb{R}^2$, $\boldsymbol{w}_j := (w_j, b_j) \in \mathbb{R}^2$, $\sigma(x) = \mathrm{ReLU}(x)$。为简单起见, 我们设置网络宽度 $m = 2$, 并假设网络可以完美拟合由目标函数 $\sigma(\boldsymbol{w}^* \cdot \boldsymbol{x})$ 生成的由 $\boldsymbol{x}_1, \boldsymbol{x}_2$ 两个数据点构成的训练集, 记为 $\boldsymbol{o}^* := (\sigma(\boldsymbol{w}^* \cdot \boldsymbol{x}_1), \sigma(\boldsymbol{w}^* \cdot \boldsymbol{x}_2))$。我们进一步假设 $\boldsymbol{w}^* \cdot \boldsymbol{x}_i > 0, i = 1, 2$。将样本上第 j 个神经元的输出表示为

$$\boldsymbol{o}_j = (a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}_1), a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}_2)).$$

在经过充分的训练后, 网络的输出应当等于训练数据点上的目标值, 即

$$\boldsymbol{o}^* = \boldsymbol{o}_1 + \boldsymbol{o}_2.$$

事实上, 由于神经网络的自由度大于 2，因此有无数种 $\boldsymbol{o}_1$ 和 $\boldsymbol{o}_2$ 的组合都能很好地拟合 $\boldsymbol{o}^*$。然而, 对于使用 dropout 正则化技巧的网络，它们存在额外的 $R_1(\boldsymbol{\theta})$ 项会引导训练过程收敛到特定的解。注意到

$$\|\boldsymbol{o}_j\|_2^2 = \sum_{i=1}^{2} (a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x}_i))^2,$$

$R_1(\boldsymbol{\theta})$ 可以写成

$$R_1(\boldsymbol{\theta}) = \|\boldsymbol{o}_1\|_2^2 + \|\boldsymbol{o}_2\|_2^2,$$

如图11.6所示, 为了最小化 $R_1(\boldsymbol{\theta})$, 当模型训练良好时,$\boldsymbol{o}_j$ 中垂直于 $\boldsymbol{o}^*$ 的分量需要相互抵消。因此，为了让他们的范数和最小，$\boldsymbol{o}_1$ 和 $\boldsymbol{o}_2$ 必须与 $\boldsymbol{o}^*$ 平行, 此时可以得到 $\boldsymbol{w}_1 // \boldsymbol{w}_2 // \boldsymbol{w}^*$, 这正是凝聚现象。对于更广泛设定下的 Dropout 引导凝聚的证明，我们在后续高阶部分进行介绍。

图 11.6: $R_1(\boldsymbol{\theta})$ 项促进凝聚现象发生原因的直观示意图

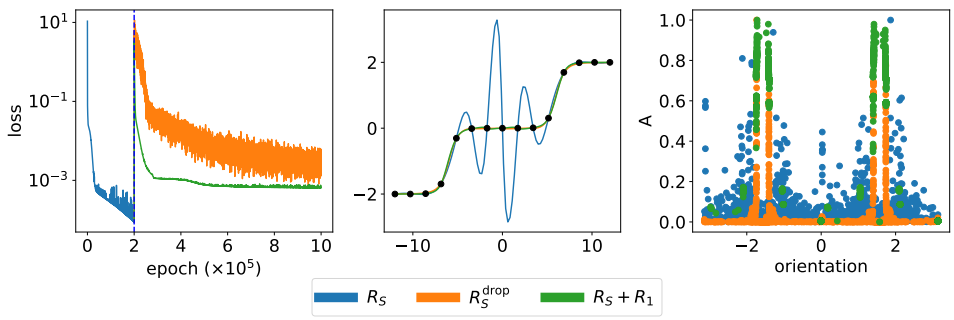


图 11.7: 两层 tanh 网络解在不同设定下的比较

11.5 Dropout 对非凝聚极小值的正则化效果

既然对于大初始的网络, dropout 对其有促进凝聚的作用, 那么如果对于一个正常训练得到的、已经近似拟合好的模型，然后使用 dropout 继续进行训练, 使用 dropout 后模型的凝聚程度是否会得到提升? 为了回答这个问题, 我们研究了使用损失函数 $R_S(\boldsymbol{\theta})$ 训练的模型在两个损失函数 $R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})$ 和 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 下继续训练得到的结果。

如图11.7左图所示, 我们首先使用 $R_S(\boldsymbol{\theta})$ 作为损失函数来训练虚线之前的模型。其中蓝色表示在大初始化以 $R_S(\boldsymbol{\theta})$ 为目标函数训练。橙色表示将以 $R_S(\boldsymbol{\theta})$ 为目标函数训练得到的解作为初始化参数（左图虚线位置），以 $R_S^{\text{drop}}(\boldsymbol{\theta})$ 为损失函数进行训练。绿色表示将以 $R_S(\boldsymbol{\theta})$ 为目标函数训练得到的解作为初始化参数（左图虚线位置），以 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 为损失函数进行训练。当 $R_S(\boldsymbol{\theta})$ 的值已经很小时, 我们用 $R_S^{\text{drop}}(\boldsymbol{\theta}, \boldsymbol{\eta})$ 或 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$ 替换原来的损失函数, 并继续训练。图11.7的中间和右侧子图分别展示了使用这三个损失函数训练所得模型的输出函数和参数特征。其中，中间子图表示以 $R_S(\boldsymbol{\theta})$ 训练的模型（蓝色）以及以 $R_S(\boldsymbol{\theta})$ 得到模型作为初始化，以 $R_S^{\text{drop}}(\boldsymbol{\theta})$（橙色）和 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$（绿色）训练得到的模型的输出，其中黑点是目标点。右侧子图表示以 $R_S(\boldsymbol{\theta})$ 训练的模型（蓝色）以及以 $R_S(\boldsymbol{\theta})$ 得到模型作为初始化，以 $R_S^{\text{drop}}(\boldsymbol{\theta})$（橙色）和 $R_S(\boldsymbol{\theta}) + R_1(\boldsymbol{\theta})$（绿色）训练得到的模型的参数特征分布。

通过比较实验结果, 我们发现加入 dropout(或 $R_1(\boldsymbol{\theta})$ 项) 后, 新得到的解（橙色点）相较于加入 dropout 前 $R_S(\boldsymbol{\theta})$ 在大初始化下获得的解（蓝色点）具有更明显的凝聚。这表明, 即便是对于已经接近极小值的模型,dropout 依然能够发挥正则化作用, 找到凝聚程度更高的解。

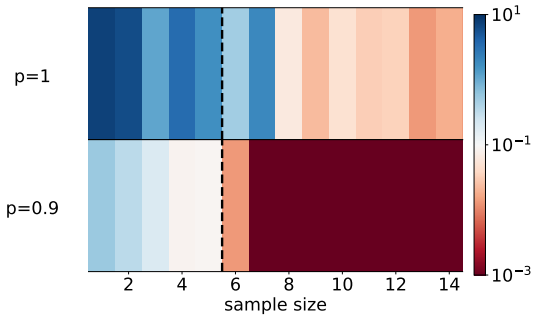


图 11.8: 不同 dropout 留存概率下两层 tanh 神经网络的平均测试误差与样本数量的关系

11.6 Dropout 与乐观估计的关系

在前面的章节中，我们引入了乐观样本量这一概念。乐观样本量是指理想情况下，恢复目标函数所需要的最少样本量。小初始化是一种达到这种理想情况的方式，但往往需要较久的训练步数。Dropout 同样可以引导模型凝聚，我们下面研究其是否可以通过乐观样本量数量重构目标函数。为了验证这一点，我们取目标函数

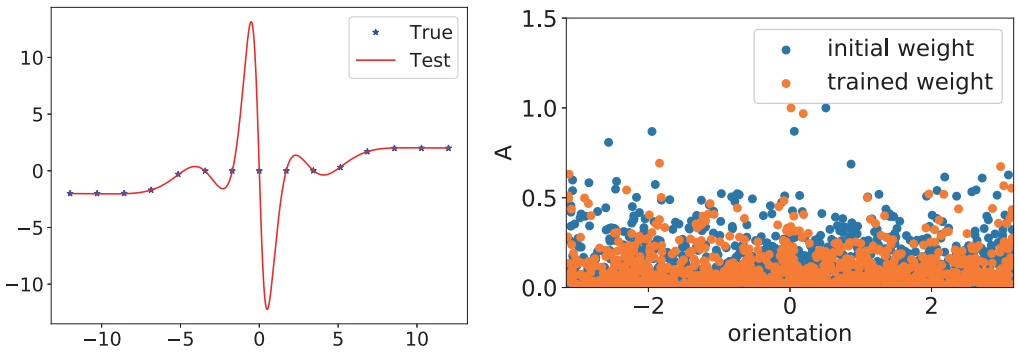
$$f(x) = \sigma(x-6) + \sigma(x+6),$$

其中 $\sigma(x) = \tanh(x)$。我们使用具有 1000 个神经元的两层 tanh 网络来学习这个具有两个神经元的两层 tanh 网络（教师-学生设定）。教师网络中的自由参数数量为 6，通过乐观样本量章节的学习，我们知道恢复目标函数所需要的最少样本量为 6。如图11.8所示，对于所有实验，隐藏层的宽度为 1000，Adam 优化器的学习率为 1×10^{-4}。每个测试误差是随机初始化的 10 次独立试验的平均值。当样本数大于 6 时，带有 dropout 的模型泛化能力变好，而没有 dropout 的模型泛化能力仍然较差。这表明,dropout 可以显著降低实际训练中恢复目标函数所需要的样本量。使其更靠近乐观样本量。因此，dropout 提供了一种简单而有效的方法来降低模型实际训练中恢复目标函数所需要的样本量, 在有限数据量下提高过参数化模型的泛化能力。

11.7 SGD 对凝聚没有明显的引导作用

随机梯度下降（SGD）作为一种常用的随机算法，其在数据上的随机性同样被验证对泛化性具有一定的提升效果。读者可能会好奇，SGD 是否与 dropout 类似，也具有引导凝聚的作用？我们仿照图11.2的实验设定，对 SGD 进行了类似的实验。如图11.9所示，隐藏层的宽度为 1000，学习率为 1×10^{-3}。(b) 中，蓝点和橙点分别表示初始和最终训练阶段的权重特征分布。

即使训练期间存在由数据随机性所带来的噪声，模型也不会发生明显的凝聚。因此，dropout 促进凝聚特性是由其噪声结构的特殊性所导致的。



(a) batch size = 2, output

(b) batch size = 2, feature

图 11.9: 两层 tanh NN 输出和特征

事实上，我们也可以通过实现方式来直观猜测不同正则化方法的作用对象。对于 SGD，由于它是对数据做随机，因此他更倾向于学到使不同批次样本带来的梯度方向尽量靠近的函数 (Smith et al. 2020)。这种特性有助于模型在面对数据的随机扰动时保持稳定性，从而提高模型泛化性。而对于 dropout，由于它是在神经元层面做随机，因此他学到的模型更倾向于使不同神经元具有相似输出，这也就是凝聚现象。下面，我们从另一个视角——损失景观平坦性去探究 dropout 提升泛化性的原因。

11.8 Dropout 对解的平坦性的影响

许多实验工作证实最小值周围损失景观的平坦性与其泛化能力有很高的相关性。在本节中，我们研究了 dropout 得到的损失最小值对应的损失景观平坦性。

为了公平比较不同模型之间的平坦度，我们采用Li et al. (2017) 中使用的方法，如下所示。为了获得具有参数 $\boldsymbol{\theta}$ 的网络方向，我们首先生成一个随机高斯方向向量 $\boldsymbol{d}$，其尺寸与 $\boldsymbol{\theta}$ 一致。然后，我们对 $\boldsymbol{d}$ 中的每个滤波器进行标准化，使其与 $\boldsymbol{\theta}$ 中相应的滤波器具有相同的范数。对于全连接网络来说，每一层都可以看作一个滤波器，归一化过程相当于对层进行归一化，而对于卷积神经网络来说，每个卷积核可能有多个滤波器，每个滤波器都单独进行归一化。因此，我们通过将 $\boldsymbol{d}_{i,j}$ 替换为 $\frac{\boldsymbol{d}_{i,j}}{\|\boldsymbol{d}_{i,j}\|}\|\boldsymbol{\theta}_{i,j}\|$ 来获得归一化方向向量 $\boldsymbol{d}$，其中 $\boldsymbol{d}_{i,j}$ 和 $\boldsymbol{\theta}_{i,j}$ 表示随机方向第 i 层的第 j 个滤波器分别为 $\boldsymbol{d}$ 和网络参数 $\boldsymbol{\theta}$。这里，$\|\cdot\|$ 表示 Frobenius 范数。需要注意的是，j 指的是过滤器索引。我们使用函数 $L(\alpha) = R_S(\boldsymbol{\theta} + \alpha\boldsymbol{d})$ 来表征使用和不使用 dropout

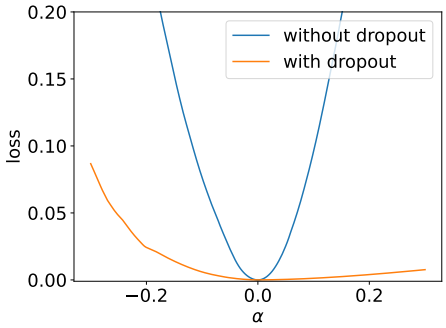


图 11.10: 训练有无 dropout 的两层 ReLU 神经网络的实验结果

层时获得的最小值周围的损失景观。

我们先在一维简单例子上验证 dropout 对平坦性的效果。我们使用宽度为 1000 的 ReLU 全连接神经网络来拟合目标函数，如下所示，

$$f(x) = \frac{1}{2}\sigma(-x - \frac{1}{3}) + \frac{1}{2}\sigma(x - \frac{1}{3}),$$

其中 $\sigma(x) = \text{ReLU}(x)$. 我们使用 Adam 训练网络，学习率为 1×10^{-4}。我们使用大初始化参数，$\boldsymbol{\theta} \sim N\left(0, \frac{1}{m^{0.2}}\right)$，其中 $m = 1000$ 是隐藏层的宽度。

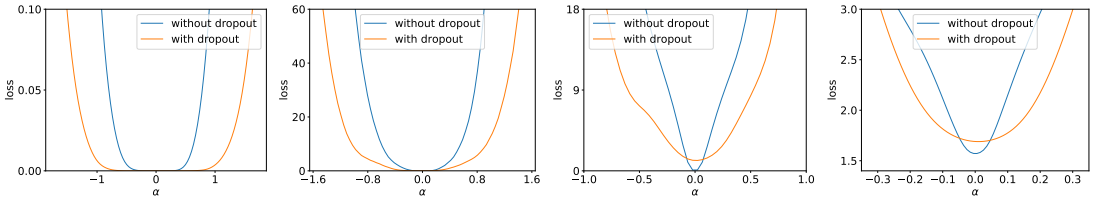
从图 11.10中容易看到，对于使用 dropout 的网络，其最小值点附近的损失景观更加平坦。下面，我们着眼于更复杂的任务，并比较添加与不添加 dropout 层网络的平坦性及其泛化能力。

对于图 11.11 中所示的所有网络结构，dropout 提高了网络的泛化性并找到更平坦的最小值。在图 11.11(a, b) 中，对于有和没有 dropout 层训练的网络，训练损失值都接近于零，但它们的平坦度和泛化能力仍然不同。在图 11.11(c, d) 中，由于数据集（即 CIFAR-100 和 Multi30k）以及网络结构（即 ResNet-20 和 Transformer）的复杂性，具有 dropout 的网络难以实现零训练误差，但带有 dropout 层的网络依旧能找到更平坦的最小值，具有更好的泛化能力。不同网络的准确率如表11.1所示，详细设定如下：图 11.11(a) 表示 FNN 在 MNIST 数据集上进行训练。具有 dropout 层的模型的测试精度为 98.7%，而没有 dropout 层的模型的测试精度为 98.1%。图 11.11(b) 表示使用前 2048 个示例作为训练数据集，在 CIFAR-10 数据集上训练 VGG-9 网络。具有 dropout 层的模型的测试精度为 60.6%，而没有 dropout 层的模型的测试精度为 59.2%。图 11.11(c) 表示使用所有示例作为训练数据集，在 CIFAR-100 数据集上训练 ResNet-20 网络。具有 dropout 层的模型的测试准确度为 54.7%，而没有 dropout 层的模型的测试准确度为 34.1%。图 11.11(d) 表示使用前 2048 个示例作为训练数据集，在 Multi30k 数据集上训练 Transformer。具有 dropout 层的模型的测试准确度为 49.3%，而没有 dropout

表 11.1: Dropout 对模型准确性的影响

网络结构	数据集	有 dropout	没有 dropout
FNN	MNIST	98.7%	98.1%
VGG-9	CIFAR-10	60.6%	59.2%
ResNet-20	CIFAR-100	54.7%	34.1%
Transformer	Multi30k	49.3%	34.7%

层的模型的测试准确度为 34.7%。



(a) FNN 的损失景观 (b) VGG-9 的损失景观 (c) ResNet-20 的损失景观 (d) transformer 的损失景观

图 11.11: 使用或不使用 dropout 层获得的不同网络结构的解的损失景观一维可视化

11.9 正则化项引导凝聚的理论解释**

在本节中, 我们将展示在多个设定下, 最小化 $R_1(\boldsymbol{\theta})$ 项可以导致神经元凝聚。首先, 我们给出一些描述 ReLU 神经元特征的定义 (如图11.12所示)。

定义 11 (ReLU 神经元的方向和截距点). *对于一维 ReLU 神经元 $a_j\sigma(w_jx+b_j)$, 其方向定义为 $\mathrm{sign}(w_j)$, 截距点定义为 $x=-\frac{b_j}{w_j}$。*

通俗来讲, 这里方向的含义即是 ReLU 神经元输出为零的区间相对于非零区间的位置, 而非输出值的正负。

定义 12 (ReLU 神经网络的凸性变化). *考虑分段线性函数 $f(t), t\in\mathbb{R}$, 以及它的线性区间集合 $\{[t_i,t_{i+1}]\}_{i=1}^{T}$。对于任意两个区间 $[t_i,t_{i+2}],[t_{i+1},t_{i+3}], i\in[T-3]$, 如果在其中一个区间上 f 是凸函数, 而在另一个区间上 f 是凹函数, 则称 f 存在一个凸性变化。*

我们证明在两层 ReLU 神经网络和一维输入数据的设定下,$R_1(\boldsymbol{\theta})$ 项的隐式偏好是满足下列两个条件的 “简单” 函数: (i) 它们具有拟合训练点所需的最少凸性变化次数; (ii) 如果神经元的截距点位于同一内部区间, 且神经元具有相同的方向, 则它们的截距点相同。定理内容如下:

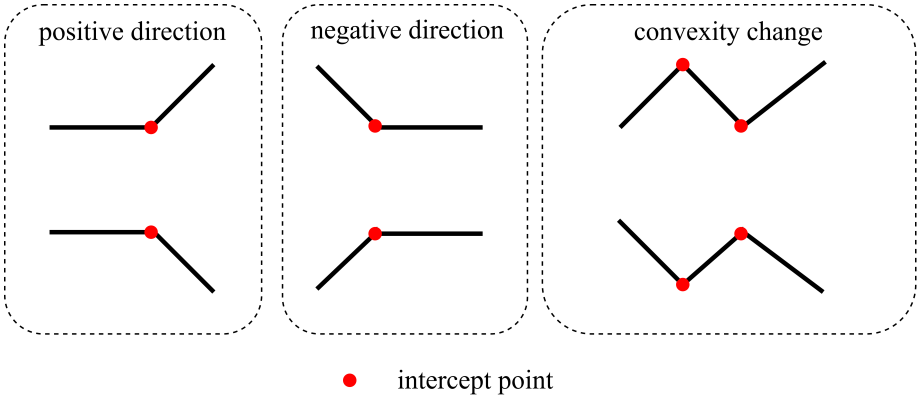


图 11.12: ReLU 神经元相关定义的示意图

定理 12 ($R_1(\boldsymbol{\theta})$ **促进凝聚的效应**)**.** *考虑以下二层 ReLU 神经网络,*

$$f_{\boldsymbol{\theta}}(x) = \sum_{j=1}^{m} a_j \sigma(w_j x + b_j) + ax + b,$$

使用一维数据集 $S = \{(x_i, y_i)\}_{i=1}^n$ 进行训练, 其中 $x_1 < x_2 < \cdots < x_n$。当训练数据的均方误差 $R_S(\boldsymbol{\theta}) = 0$ 时, 如果发生以下两种情况之一:

(i) 神经网络在 (x_1, x_n) 中的凸性变化次数可以在保持 $R_S(\boldsymbol{\theta}) = 0$ 的同时减少;

(ii) 存在两个指标为 $k_1 \neq k_2$ 的神经元, 使得它们具有相同的符号, 即 $\text{sign}(w_{k_1}) = \text{sign}(w_{k_2})$, 且在同一区间内具有不同的截距点, 即对于某个 $i \in [2 : n-2]$, 有 $-b_{k_1}/w_{k_1}, -b_{k_2}/w_{k_2} \in [x_i, x_{i+1}]$, 且 $-b_{k_1}/w_{k_1} \neq -b_{k_2}/w_{k_2}$;

则存在参数 $\boldsymbol{\theta}'$, 它是 $\boldsymbol{\theta}$ 的一个无穷小扰动, 使得:

(i)$R_S(\boldsymbol{\theta}') = 0$;

(ii)$R_1(\boldsymbol{\theta}') < R_1(\boldsymbol{\theta})$。

Remark 11.9.1. *定理12意味着, 对于一个过参数化的、具有许多全局最小值的网络, 如果发生上述两种情况之一, 就会存在一个全局最小值, 其 R_1 值比当前的最小值更小。换句话说, 如果一个全局最小值在所有最小值上最小化了 R_1, 那么定理中的两种情况都不会发生, 即神经网络在 (x_1, x_n) 中的凸性变化次数是最小的, 并且对于任意 $i \in [2 : n-2]$, 模型在任意区间 $[x_i, x_{i+1}]$ 之间最多有 2 个截距点。*

Remark 11.9.2. *定理12中额外的线性函数 $ax+b$ 是为了在研究情况 (i) 中的某些情况时保持零损失而添加的。在该定理中, 我们通过同时向不同方向微调两个神经元的截距点来减小 R_1。由于在某些情况下, 两个神经元具有相反的方向, 即它们的输入权重具有相反的符号, 因此两*

个神经元的截距点的移动将全局影响网络输出 (一个神经元将影响一侧)。为了抵消这种全局影响以保持零损失，必须添加一个全局函数。通过构造证明，我们可以使用线性函数可以实现这一点。由于线性函数的可加性，减少多个凸性变化所需的线性函数可以等价于一个线性函数。需要注意的是，这个线性函数可以由两个 ReLU 组成，这意味着原始网络本质上仍然是一个二层 ReLU 神经网络。

图11.13左侧曲线的例子展示了定理12的情况 (i)，即在保持损失为零的同时可以减少凸性变化。图11.13右侧曲线的例子展示了定理12的情况 (ii)，即在第三个和第四个点之间存在多于两个截距点而没有凸性变化。右侧和左侧曲线都可以在 R_1 减小且训练损失为零的情况下简化为中间曲线。因此，情况 (i) 和情况 (ii) 都不发生的情况可以促进凝聚。同时值得注意的是，凝聚可以发生在任何一种情况下。然而，当任何一种情况简化为中间情况时，有效神经元比率可以进一步降低 (更加凝聚)。

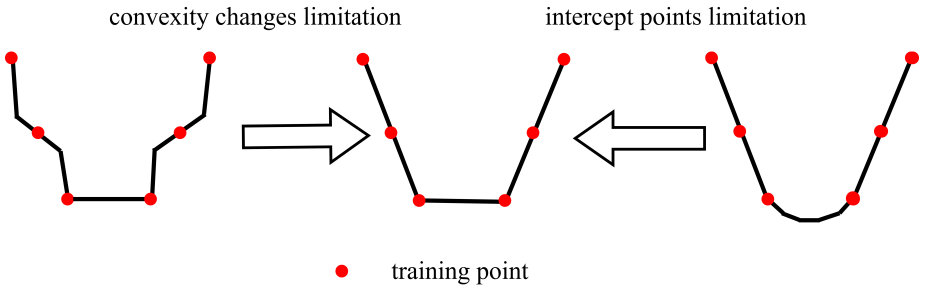


图 11.13: $R_1(\boldsymbol{\theta})$ 项对凸性变化和截距点约束效应的示意图

同时，并非所有使用 dropout 训练的函数都表现出明显的凝聚。例如，当仅使用一个数据点进行训练时，带有 dropout 的函数不会出现凝聚。然而，对于一般的数据集，如图11.1和图11.2所示的例子，神经网络由于凸性变化和截距点的约束 (如图11.13所示) 而达到凝聚解。

11.10 习题

1. 在本章的实验中，我们观察到 dropout 可以显著促进神经元的凝聚现象。

   (a) 神经元凝聚对模型的泛化能力有何意义？为什么我们希望模型出现凝聚？

   (b) 除了实验观察，你能否从直观上解释 dropout 为什么能够引导凝聚？dropout 的随机遮盖机制起到了什么作用？

   (c) 设计一个实验来量化 dropout 对凝聚程度的影响。描述你的实验设置并讨论可能的结果。

2. Dropout 作为隐式正则化技术, 在提升模型泛化性方面有独特的优势。

   (a) 什么是隐式正则化? 它与显式正则化 (如 L2 正则) 有何不同?

   (b) 相比于标准的 SGD 训练, 使用 dropout 的优势体现在哪些方面? 进行类比实验说明。

3. 本章推导了 dropout 隐式正则化的显式表达式 $R_1(\boldsymbol{\theta})$。请回答:

   (a) $R_1(\boldsymbol{\theta})$ 的具体形式是什么? 它对神经元的输出施加了什么样的约束?

   (b) $R_1(\boldsymbol{\theta})$ 是一个与学习率无关的项。这意味着什么? 为什么说这体现了 dropout 作为隐式正则化手段的特点?

   (c) 基于 $R_1(\boldsymbol{\theta})$ 项, 你能否提出一种新的显式正则化方法? 与 dropout 相比, 这种方法有何异同?

4. tanh 函数的多解性:

   (a) 方程 $a\tanh(wx+b) = \tanh(x))$ 有几个解?

   (b) 为何我们的教师网络的构造只有一个方向, 在添加 dropout 的余弦相似度图11.4(a)中有两个方向, 我们却认为学生网络很好地还原了教师网络?

5. 分析 dropout 对神经网络损失景观几何性质（如平坦性）的影响。

   (a) 什么是平坦的最小值? 它与模型泛化能力有何关联?

   (b) 在本章给出的实验中, dropout 是如何改变损失景观的几何性质的? 这种改变对模型泛化有何意义?

   (c) 你能否提出一种定量的指标来度量神经网络解的平坦性?

6. 本章定理 7.1 从理论上证明了 $R_1(\boldsymbol{\theta})$ 项促进凝聚的效应。

   (a) 定理 7.1 的核心结论是什么? 它在什么条件下成立?

   (b) 在定理 7.1 的证明中, 为什么我们要引入一个额外的线性项 $ax+b$? 它起到了什么作用?

   (c) 你能否构造一个具体的例子来说明定理 7.1 的结论? 在你的例子中, $R_1(\boldsymbol{\theta})$ 项是如何引导凝聚的?

7. 请解释为什么当仅使用一个数据点进行训练时, 带有 dropout 的函数不会出现凝聚?

Chapter 12

锚函数：研究语言模型的一类简单函数

近年来，大语言模型（Large Language Model, LLM）取得了巨大的成功，在自然语言处理领域发挥着越来越重要的作用。这些模型在机器翻译、文本摘要、问答系统、情感分析等众多任务中都有广泛的应用。

目前，基于 Transformer 架构 Vaswani et al. (2017) 的语言模型已经成为了一种重要的范式。Transformer 模型通过自注意力机制（self-attention）来捕捉单词之间的长短距离依赖关系，相比传统的循环神经网络（RNN）和长短期记忆网络 (LSTM)，Transformer 结构能更好地支持并行化计算，处理更长的文本序列。

GPT 系列模型 Brown et al. (2020), OpenAI (2023) 展示了语言模型在文本生成、对话、问答等任务中的惊人能力。通过持续扩大模型规模和训练数据量，GPT 等大型语言模型表现出了接近人类水平的语言理解和生成能力，引起了学术界和工业界的广泛关注。

尽管语言模型取得了令人瞩目的成就，但我们对其内部工作机制的理解还相对有限。语言模型究竟如何学习和表征语言知识，如何在下游任务中进行迁移和泛化，这些问题仍有待进一步探索。因此，深入研究 Transformer 等语言模型的理论基础和优化机制，对于推动语言模型的发展和应用具有重要意义。

本章详细探讨了一类研究语言模型的简单函数——锚函数（Anchor Function），并通过多种类语言任务实验分析其对语言模型的影响和潜在应用。具体内容安排如下：第一节简要回顾了深度学习模型处理语言的主要步骤，包括分词、映射、编码、输入模型和输出结果的过程，重点讨论了现代自然语言处理预训练中的 Next Token Prediction 任务。第二节讨论了研究基于 Transformer 的语言模型面临的挑战，尤其是在任务未知、高昂的计算和内存需求，以及推

理机制难解释性方面的难题。第三节阐述了语言任务的特点，重点介绍了自然语言中的逻辑推理关系和“锚点-关键值”对的概念，为研究 Transformer 模型提供了切入点。第四节提出了通过锚函数与类语言任务的研究思路，借鉴生物学和物理学中的实验方法，设计简化的实验环境来探索语言模型的行为特性。第五节详细介绍了锚函数和类语言任务的定义，通过具体示例说明其在研究语言模型中的应用，特别是单锚点和多锚点函数的构建与分析。第六节讨论了数据划分与模型泛化的问题，提出了模-余数数据划分、基于锚点的数据划分和复合锚点划分三种方法，探讨了数据泛化与任务泛化的概念。第七节展示了多种基于锚函数的实验结果与讨论，分析了模型在恒等学习任务、阅读理解任务、分类任务、复合任务、工作记忆任务、近义词任务、前向-后向背诵任务、统计输出任务、多锚点任务上的表现及其机制。第八节研究恒等学习任务的机制，通过分析简化的两层模型，揭示了 Transformer 模型在移位和广播操作上的内部机制，并讨论了其在大语言模型中的普遍性。通过这些内容，本章深入探讨了大语言模型在处理复杂语言任务时的内部工作机制，并提出了锚函数这一新视角，为未来的研究提供了重要参考。

科普篇

当前大模型的研究面临哪些困难？

在研究基于 Transformer 的大语言模型时，研究人员面临着多个挑战。这些困难主要包括未知的任务、高昂的计算和内存需求以及推理机制的难可解释性。大语言模型通常通过 next token prediction 进行训练，尽管这种方式在各种语言任务上表现出色，但其训练目标与具体任务不直接相关，导致我们难以解释模型在具体任务上的表现。此外，训练高质量的大语言模型需要大量的数据和计算资源，这对学术界的研究团队来说是巨大的挑战。最后，理解大语言模型的推理过程十分复杂，即使模型输出了正确的结果，我们也难以解释其决策过程。

什么是锚函数和类语言任务？研究动机是什么？

锚函数：锚函数是从一个语言序列到其标签的一种映射，由句子中出现的锚点决定。根据锚点的数量和组合方式，可以构造出多种不同形式的锚函数。例如，对于句子“我的名字是 xxx，请问我叫什么?”，锚点是“名字是”，关键值是“xxx”，锚函数则是从这个句子到答案“xxx”的映射。锚函数帮助研究人员理解语言模型在处理特定模式下的行为。

类语言任务：类语言任务是根据锚函数设计的简化任务，用于模拟自然语言处理中常见的某些模式。通过这些任务，研究人员可以分析模型在不同难度水平上的表现，考察其泛化能力和内部机制。例如，任务“$3x$ to x”可以模拟通过锚点“3”找到其后面的数字，这样的任务帮助研究人员更好地理解和解释模型的行为。

研究动机：通过研究锚函数和类语言任务，研究人员希望解决以下几个问题：

- 通过简化的实验环境探索模型的行为特性，揭示其内部工作机制。
- 设计合理的实验和数据集划分方案，全面评估模型的泛化能力。
- 提供更好的理解和解释语言模型的推理过程，帮助改进和优化模型在实际应用中的表现。

12.1 深度学习模型处理语言的简介

在第二章中，我们已经详细地介绍了深度学习模型处理语言的主要步骤。在这里我们做一个简短的回顾:

分词 (Tokenization) 将输入的文本按照一定规则切分成一个个基本单元, 这些单元被称为token。常见的分词方式有:

- 按空格或标点符号分割, 每个单词或标点就是一个 token
- 按字符分割, 每个字符就是一个 token
- 按词典中的词条分割, 每个词条作为一个 token
- 更复杂的方式如 Byte Pair Encoding, 根据统计频率合并字符, 生成 token

映射 (Mapping) 将每个 token 映射为一个数字 id, 以便输入模型。常见做法是根据 token 出现频率排序, 频率最高的排最前, 然后依次编号。这个映射关系被称为词表 (Vocabulary)。

编码 (Encoding) 将 token id 转化为模型可以处理的向量形式, 常用的编码方式有:

- One-hot: 词表中的每个 token id 对应一个长度为词表大小的向量, 向量中对应位置为 1, 其余为 0。比如红色:[1,0,0,0], 蓝色:[0,1,0,0]。

- 词嵌入 (Word Embedding): 先将 token id 进行 one-hot 编码, 然后通过一个嵌入矩阵 (Embedding Matrix) 将高维稀疏的 one-hot 向量映射为低维稠密的嵌入向量。嵌入矩阵可以随机初始化, 并在模型训练过程中不断学习优化。设词表大小为 V, 嵌入维度为 d, 则嵌入矩阵形状为 $V \times d$。

输入模型 将词嵌入序列输入深度学习模型。常见的模型有 RNN、CNN、Transformer 等。输入时通常会添加一些特殊 token, 如:

- <bos> 标志句子开头
- <eos> 标志句子结尾
- <pad> 将句子补齐到固定长度

- <unk> 替换词表里没有的生僻 token

输出结果 模型输出一个长度与词表大小相同的向量, 表示下一个 token 的概率分布。这个过程通常采用 softmax 函数实现:

$$P(\text{token}_i|\text{context}) = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}},$$

其中,z_i 是模型输出向量的第 i 个元素,V 为词表大小。softmax 函数将输出向量归一化为一个概率分布, 表示在给定上文 (context) 的情况下, 每个 token 出现的概率。概率最大的 token 就是模型预测的下一个 token。

举个 next token prediction 的例子, 假设要预测 “我爱吃苹果, 我每天都吃一个 [ ]” 中 “[ ]” 的位置应该填什么。

- 分词:[“我” , “爱” , “吃” , “苹果” , “,” , “我” , “每天” , “都” , “吃” , “一个”]
- 映射 (假设): “我” :100, “爱” :205, “吃” :302, “苹果” :408, “,” :500, “每天” :603, “都” :701, “一个” :809
- One-hot 编码 (假设词表大小 10000):
  - 我:[0,0,...,1,0,...,0] (第 100 个位置为 1, 其余为 0);
  - 爱:[0,0,...,1,0,...,0] (第 205 个位置为 1, 其余为 0);
  - 吃:[0,0,...,1,0,...,0] (第 302 个位置为 1, 其余为 0);
  - 苹果:[0,0,...,1,0,...,0] (第 408 个位置为 1, 其余为 0);
  - ,:[0,0,...,1,0,...,0] (第 500 个位置为 1, 其余为 0);
  - 每天:[0,0,...,1,0,...,0] (第 603 个位置为 1, 其余为 0);
  - 都:[0,0,...,1,0,...,0] (第 701 个位置为 1, 其余为 0);
  - 一个:[0,0,...,1,0,...,0] (第 809 个位置为 1, 其余为 0);
- 词嵌入 (假设嵌入维度为 50,V=10000):
  - 我: 矩阵 [100,:]=[0.1,0.2,...,0.5] (嵌入矩阵第 100 行) ;
  - 爱: 矩阵 [205,:]=[0.2,0.1,...,0.3];
  - 吃: 矩阵 [302,:]=[0.3,0.2,...,0.1];
  - 苹果: 矩阵 [408,:]=[0.1,0.4,...,0.2];
  - ,: 矩阵 [500,:]=[0.5,0.1,...,0.2];

- 每天: 矩阵 [603,:]=[0.2,0.3,...,0.4];
- 都: 矩阵 [701,:]=[0.4,0.1,...,0.3];
- 一个: 矩阵 [809,:]=[0.3,0.2,...,0.5];

- 输入: - [[0.1,0.2,...,0.5], [0.2,0.1,...,0.3], [0.3,0.2,...,0.1], [0.1,0.4,...,0.2], [0.5,0.1,...,0.2], [0.1,0.2,...,0.5], [0.2,0.3,...,0.4], [0.4,0.1,...,0.3], [0.3,0.2,...,0.1], [0.3,0.2,...,0.5]]
- 输出 (假设词表大小 10000，只展示前五个):
  - 苹果:0.5
  - 梨:0.2
  - 香蕉:0.1
  - 橘子:0.05
  - 西瓜:0.03
- 预测:"[ ]" 最可能是" 苹果", 填词后得到" 我爱吃苹果, 我每天都吃一个苹果"。

可以看到, 语言模型通过学习上下文信息, 可以比较准确地预测句子中缺失的词。将这一过程递归下去, 就可以生成连贯的文本片段。Next token prediction 是现代自然语言处理预训练的主要任务之一，GPT 等著名模型都是用这种方式进行训练的。

12.2 研究基于 Transformer 的语言模型面临的挑战

在对 Transformer 等大语言模型进行深入研究和理解的过程中，我们面临着诸多挑战。这些挑战大大增加了 Transformer 的研究难度，尤其是对于资源有限的学术研究团队而言。下面我们将详细阐述这些挑战。

12.2.1 未知的任务

在大语言模型的训练过程中，我们通常使用基于 next token prediction 的方式，即利用给定的一串 token 来预测下一个 token。这种训练方式看似与具体的语言任务无关，然而实践证明，经过这种方式训练的语言模型却能够在各种语言任务上取得优异的性能。这一现象引发了我们的思考：为什么 next token prediction 这种看似简单的训练方式，能够让语言模型产生如此强大的语言能力?

传统的机器学习范式通常是基于特定任务的, 即为每个任务设计专门的模型结构和训练目标。例如，为了进行情感分析，我们可能会设计一个专门的分类模型。然而，面对语言这样一

个复杂的认知系统，我们很难穷尽所有可能的任务并为每个任务设计最优的解决方案。相反，next token prediction 提供了一种通用的语言建模方式，让模型在海量语料中自主学习语言的内在规律和表示方法。这种训练方式不依赖于特定的任务，而是试图捕捉语言本身的统计特性和结构模式。

然而，这种通用的训练方式也带来了新的挑战：由于训练目标与具体任务并不直接相关，我们很难通过分析训练过程来解释模型在下游任务上的性能。模型在训练过程中到底学到了什么？哪些语言知识和能力最先被习得？这些问题都难以直接从 next token prediction 的训练过程中得到答案。此外，由于缺乏针对具体任务的评估指标，我们也难以在训练过程中直接优化和改进模型在特定任务上的表现。

next token prediction 这种通用的语言模型训练方式，让模型能够从海量语料中习得语言的内在规律，产生强大的语言理解和生成能力。但与此同时，这种非任务驱动的训练方式也给模型的分析和优化带来了新的挑战。如何更好地理解和引导语言模型的学习过程，是我们在探索语言模型奥秘时不可回避的重要课题。

12.2.2 高昂的计算和内存需求

训练一个高质量的语言模型通常需要海量的数据和极大的计算资源，这对学术界研究团队来说是一个巨大的挑战。

- GPT-3 模型使用了 1750 亿个参数，训练数据量达到了 499 亿个 tokenBrown et al. (2020)。训练如此巨大的模型需要昂贵的硬件设施和漫长的训练时间，这对于资金有限的学术实验室而言是难以承受的。

12.2.3 推理机制的难可解释性

理解语言模型如何完成各种任务以及解释其决策过程是 Transformer 研究中的一大挑战。

- Transformer 模型通常包含数以亿计的参数，其推理过程涉及复杂的非线性变换和高维向量运算。即使模型根据输入内容输出了正确的结果，但是我们也很难准确地解释模型是如何一步步推理出这个结果的。
- 目前已有一些工作尝试通过可视化注意力机制来理解 Transformer 的工作原理。然而，注意力分数并不能完全反映模型的推理过程。我们还需要更加细致入微的分析工具和方法。

12.3 语言任务的特点

语言是人类最重要的交流工具，其所具有的独特结构和特征为研究语言奥秘提供了丰富的线索。大多数语言任务通常都具有某些明确的逻辑结构，例如，

- 在自然语言中，通常存在大量逻辑推理关系。例如，“小明是学校 A 的老师，学校 A 的老师都是教授，那么可以推出小明是教授”。这实际上是一个经典的三段论推理模式，其基本形式为 “所有 A 是 B，C 是 A，因此 C 是 B”。除了三段论，自然语言中还存在更高阶的逻辑推理链，其推理过程涉及了多个环节，需要逐步应用之前得到的结论，形成一条完整的逻辑链条。
- 在自然语言中，还存在大量的 “锚点-关键值”（anchor-key）对。例如，如果你询问一个大语言模型：“我的名字是 xxx，请问我叫什么?”，那么无论你的名字是什么，它都会准确无误地输出出来。在这个任务中，“名字是” 就是一个锚点，它告诉大模型应该关注紧随其后的信息。“xxx” 即为锚点对应的关键值，通常来说，无论其填入什么 token，它都指的是句中 “我” 的 “名字” 这一属性。这种锚点-关键值的对应关系构成了语言组织的基本单元，为理解语言结构提供了重要依据。

这类具有明确结构的语言任务为研究 transformer 提供了一个很好的切入点。我们可以通过分析 transformer 模型的推断方式是否与这些任务的内在逻辑结构相一致，来理解 transformer 的学习和推断机制，揭示其内部工作原理和偏好性。这些认识不仅有助于我们改进语言模型，还可以为 transformer 在其他领域的应用提供有益的启示和参考。尽管语言任务具有上述有利于分析的特点，但由于其本身的复杂性，直接在真实语料上研究 transformer 仍然面临巨大挑战。因此，我们需要另辟蹊径，在简化的环境中探索 transformer 的行为特性。

12.4 研究思路

在科学研究中，当我们试图理解一个复杂现象时，一个常见的思路是先在简单、可控的环境中进行实验和验证，然后再逐步扩展到更加复杂、真实的场景。这种思路在生物学、物理学等领域已经得到了广泛的应用，并取得了巨大的成功。

一个著名的例子是 Hodgkin 和 Huxley 在研究神经元电生理时使用了巨型乌贼（Loligo）的轴突。他们之所以选择巨型乌贼，是因为其轴突直径可达 1 毫米，远大于大多数动物的轴突，这使得插入电极测量电位变化成为可能。通过在这种特殊的模型生物上进行实验，Hodgkin 和 Huxley 成功地总结出了神经元膜电位变化的数学模型（即 HH 模型），为我们理解神经系统的工作机制作出了开创性的贡献。他们的工作也因此获得了诺贝尔生理学或医学奖。

另一个例子是利用小鼠进行糖尿病研究。小鼠之所以成为糖尿病研究的理想模型，有以下几个原因：首先，小鼠与人类具有高度的生理和遗传相似性，这使得在小鼠身上获得的研究结果更有可能转化到人类身上；其次，小鼠的生命周期较短，这让研究人员能够在相对较短的时间内观察疾病的发生和发展过程，加快研究进程；再次，小鼠更容易饲养和管理，这大大降低了研究的成本；最后，小鼠的基因可以通过基因工程技术轻松地进行修改，从而能够创建特定的基因敲除或敲入模型，用于研究特定基因在糖尿病发生中的作用。这些优势使得小鼠成为了糖尿病研究中不可或缺的模型生物。

我们在前面章节中提到的频率原则、凝聚现象等方面的研究，也是使用了这种化繁为简的思想。这些工作都是首先在一些人为构造的简单函数上发现和总结实验现象，然后再推广到真实数据集。通过在受限环境下分析系统行为，我们能够更准确地把握事物的本质规律。

上述案例启示我们，在研究语言模型时，也可以采用类似的方法。与其直接在大规模、复杂的数据集和任务上分析模型的行为，不如先设计一些简单、可控的任务，通过实验来观察模型的特性和规律。然而，之前章节中的简单函数的构造方式主要针对回归问题（对于连续函数的拟合），无法直接应用于离散型的语言建模任务。为了研究语言模型，我们需要另辟蹊径，设计新的简化函数，以观察 Transformer 等语言模型对不同语言特征的敏感性和偏好。通过在可控的实验环境中考察模型行为，我们有望发现语言处理背后的一般规律，从而为理论突破提供切入点。这种从现象出发、通过简单函数来探索未知领域的研究范式，正是我们所倡导的现象驱动的理论研究方法。

12.5 锚函数与类语言任务

在前文中我们已经介绍，语言数据具有很多特点，这里我们重点关注 “锚点-关键值” 对这一个特点。我们根据这一特性，提出了一种名为「锚函数」(Anchor Function) 的类语言任务模式。锚函数是一类特殊设计的函数，用于模拟自然语言处理中常见的某些模式。

我们用一个简单的例子来引出锚函数和类语言任务的概念。对于我们前文中提到的询问名字的任务，我们可以设计一个称为 “$3x$ to x” 的类语言任务：任意给定由 20∼100 中的随机数字和一个 3 组成的定长序列（3 有且仅有一个），如（43，33，23，3，20，89，44，24，56），我们设定其标签是 3 后边的数字，在本例中为 20。在这个任务中，锚点为 3，关键值为 3 后边的数字，锚函数为这段序列到 3 后边的数字的映射。这个类语言任务与询问名字的任务有很好的对应关系，但相比之下，这个任务更加简洁、确定和独立，完全可以在小型 Transformer 网络上进行训练和研究。

在接下来的部分中，我们将详细介绍锚函数、类语言任务的定义以及使用锚函数来研究语言模型的优势。

12.5.1 锚函数

锚函数指的是从一个语言序列到其标签的一种映射，其函数表达式由句中出现的锚点来决定。根据锚点的数量和组合方式，我们可以构造出多种不同形式的锚函数。

单锚点函数

单锚点函数是最简单的一类锚函数，指那些拥有独立运算规则的锚点所对应的函数。例如前文中所提到的“$3x$ to x”任务，这个任务中锚点为“3”，对应的单锚点函数为：

$$f_3(x_1, \ldots, x_n) = x_{i+1}, \quad \text{其中} \quad x_i = 3. \tag{12.1}$$

我们定义的单锚点函数 f_a 是一个从 $\mathbb{R}^{n\times d}$ 到 $\mathbb{R}^d$ 的映射，其中 a 是锚点，n 是句子中 token 的数量，也就是句长，d 是每个 token 的维度，在深度学习中，一般每个 token 都用一个 one-hot 向量表示，即 $\boldsymbol{x} \in \{0,1\}^d$，于是 d 还表示词表的大小。一个长度为 n 的句子用 $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \in \mathbb{R}^{n\times d}$ 表示。在每个句子 X 中，$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n-1}$ 中有且仅有一项等于指定的提示锚点 $\boldsymbol{a} \in \mathbb{R}^d$，假设 $\boldsymbol{x}_i = \boldsymbol{a}$，则单锚点函数满足 $f_a(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \boldsymbol{x}_{i+1}$。更严格地，单锚点函数可以写成以下形式：

$$f_a(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_{i\in\Omega} \boldsymbol{x}_{i+1}\delta(\boldsymbol{x}_i, \boldsymbol{a}), \tag{12.2}$$

其中，Ω 是锚点可能出现的所有位置的集合，$\delta(\boldsymbol{x}_i, \boldsymbol{a})$ 为克罗内克 δ 函数，即当 $\boldsymbol{x}_i = \boldsymbol{a}$ 时为 1，其他情况下为 0。

更一般地，我们也可以对锚函数的输出做进一步运算，如“$3x$ to x+1”，即输出“3”后边的数字加 1。锚点关注的数字也是可变的，如可以让锚点关注后两位或前一位，甚至更多的位置，于是我们可以定义一个更一般的单锚点函数：

$$f_a(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_{i\in\Omega} g_a(X_{I_{\boldsymbol{a},i}})\delta(\boldsymbol{x}_i, \boldsymbol{a}), \tag{12.3}$$

其中 g_a 表示我们对识别出的 token 所做的进一步运算，$I_{\boldsymbol{a},i}$ 表示由锚点和它所处的位置共同决定的关注到的 token 索引构成的指标集，我们称这些被关注到的 token（即 $X_{I_{\boldsymbol{a},i}}$）为**锚点 a 的关键项**。

注 30. 通常锚点的运算对于它所处的位置是不敏感的，如在询问名字的语境里，总是输出“名字是”这个锚点后边跟着的名字，也就是说锚点的操作在一定程度上满足平移不变性。我们在设计锚函数时也重点关注满足平移不变性的锚函数，因此，为体现平移不变性，$I_{\boldsymbol{a},i}$ 可以进一步改为 $I_{\boldsymbol{a}} + i$。

多锚点函数

在单锚点函数的基础上，我们还可以构建多个锚点共同发挥作用的多锚点函数 $f_A(X)$：$\mathbb{R}^{n\times d}\to\mathbb{R}^d$，其中，$A=\{\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_l\}$ 为锚点集合。假设在输入序列 X 中，这些锚点出现的位置集合为 J，则多锚点函数可表示为：

$$f_A(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)=\sum_{J\in\Omega}g_A(X_{I_{A,J}})\delta(X_J,A), \tag{12.4}$$

其中 Ω 为锚点集可能所在的所有位置的集合构成的集族。

注 31. 一个简单的多锚点函数是双锚点复合函数，即每个锚点有自己的运算规则，记锚点 $\boldsymbol{a}_i$ 的运算规则为 $g_{\boldsymbol{a}_i}$，双锚点函数为这些简单锚函数的复合：

$$g_A(X_{I_{A,J}})=g_a(g_b(X_{I_{A,J}})). \tag{12.5}$$

在后续的内容中，我们会重点讨论这一类双锚点复合函数的例子。

与真实语言数据中的锚点及其作用相比，锚函数有明确的定义和输出，我们可以方便地控制输入数据的分布和复杂度。这有助于我们分析模型在不同难度水平上的表现，考察其泛化能力的极限。同时，由于任务目标明确，我们也更容易解释模型的行为。

12.5.2 类语言任务

依据不同的锚函数，我们可以设计多种类语言任务，来模拟一些真实语言情景。图 12.1列举了多个类语言任务，其中，第一行为单锚点函数的任务，第二行为多锚点函数的任务，第三行为对序列做理解的任务以及非确定性映射的锚函数任务。这些任务都有具体的语境意义，如恒等学习（identity learning task）任务即是前文中提到的 “$3x$ to x” 任务，进而对应从前文找答案的场景；复合任务（composite task）是两个锚点构造的多锚点函数任务，对应复杂的思考场景；阅读理解（reading comprehension task）的目标是根据提示词寻找语句前文中出现过的内容；前向-后向背诵任务（forward-backward recitation task）考察了大模型根据前文找后文和根据后文找前文能力的差异..... 这些类语言任务的定义以及它们与真实语境的对应关系将在后文讨论实验结果时详细地阐述。

与真实语言任务相比，这些类语言任务具有许多优势。首先，它们的规则简单明确，易于理解和实现。研究人员可以快速设计和构建符合特定需求的任务，而无需处理真实语言数据中的噪声和不确定性。其次，类语言任务的数据生成过程完全受控，可以精确地控制训练集和测试集的划分，从而更准确地评估模型的泛化能力。此外，由于任务简单，训练时需要的数据量比较小，模型规模也相对较小，对于现象研究来说，所需的计算资源比较小，使得研究人员能够快速迭代和优化模型。

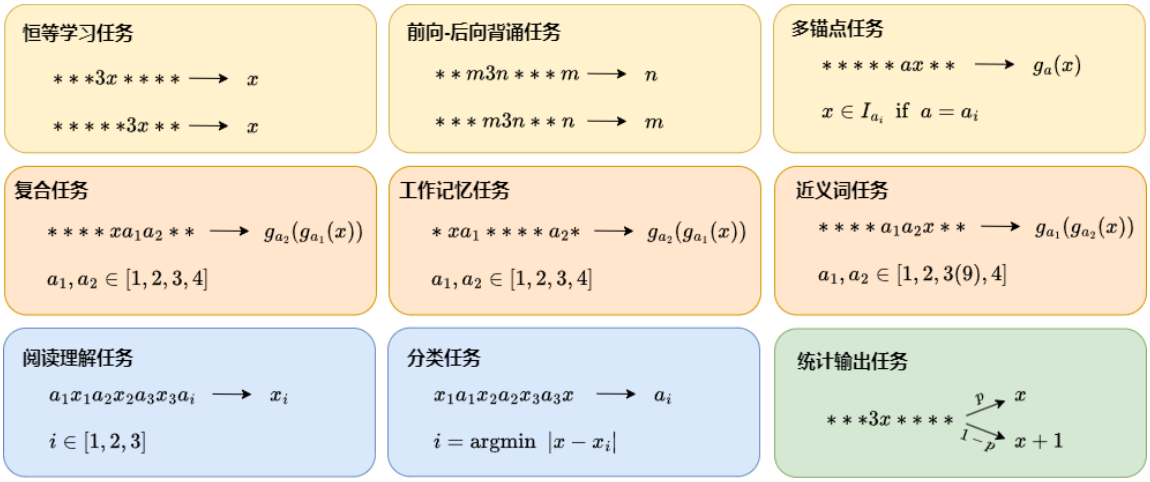


图 12.1: 锚点函数的任务示例

更重要的是，通过类语言任务，我们可以深入理解大语言模型在处理真实语言问题时的内部机制。尽管类语言任务相对简化，但它们仍然捕捉到了真实语言处理中的部分关键特征，如上下文依赖、语义组合、推理等。我们可以通过可视化神经元激活、注意力权重等方式，来解释模型是如何一步步得出输出结果的。这对于理解 Transformer 等深度神经网络的工作机制具有重要意义。这种从简单到复杂、从可控到真实的研究范式，有助于我们逐步揭示大语言模型的本质。

12.6 数据划分与模型泛化

为了充分发挥锚函数与类语言任务的优势，合理的实验设定和数据集划分策略至关重要。在接下来的部分中，我们将详细讨论数据集划分的各种方法，以及对模型泛化性更深层次的理解。

12.6.1 训练集和测试集的划分

在本节中，我们希望提出几种适用于类语言任务的数据划分方式，以期望达到以下两个目标：

- 能够清晰地划分训练集和测试集。
- 希望通过该数据划分方式来进一步验证 Transformer 结构是否学到了语言任务在逻辑结构层面的本质。

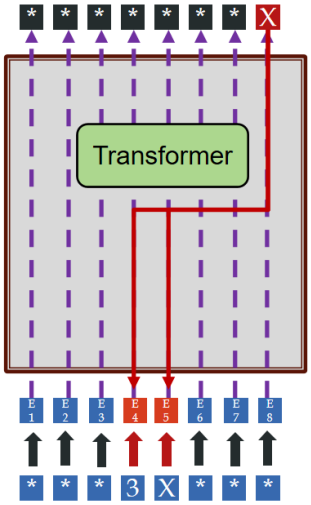


图 12.2: 一个序列传入 Transformer 后参数的更新形式

以 “$3x$ to x” 任务为例，一个自然的想法是依据关键项来进行划分，即训练集中序列的关键项属于某个区间，测试集的属于另一个区间，句中其他 token 均随机选择。但这样划分实际上是存在问题的，图 12.2给了一个直观的理解，一方面，对于一个 “$3x$ to x” 序列，只有锚点 “3” 和关键项 “x” 是与标签 “x” 有关联的，所以在反向传播过程中，“3” 与关键项 “x” 的 embedding 向量会被当做锚点和关键项更新一次，而其他位置的 “x” 会被当做噪音更新一次。如果某个 “x_0” 从来没有在关键项出现过，那么它对应的 embedding 向量永远只被当做噪音来更新；另一方面，负责编码的 embedding 矩阵和负责投影到输出的 projection 矩阵并没有天然的匹配关系，即在未训练状态下，“x” 经过 embedding 和 projection 后不会输出 “x”。这两个原因导致神经网络无法在以这个 “x_0” 为关键项的序列上泛化。因此，我们需要合理地划分训练集与测试集，既要保证两个数据集完全没有交集，又要保证所有的 “$3x$” 组合均在训练集中出现过，其他任务同理。

在本节中，我们提出了三种划分训练集和测试集的方法，如图 12.3 所示。这三种方法各有特点，能够帮助我们更好地评估模型的泛化能力。下面我们将详细介绍每一种划分方法。

模-余数数据划分

为确保所有数据均作为关键项出现过，还能完全区分训练集和测试集，我们设计了一种模-余数数据划分方法。考虑一个输入序列长度为 n 的任务。所有 token x 的取值范围为 I。定义 $\Gamma_i = \{kn + i | k \in \mathbb{Z}\}$ 为所有模 n 余数为 i 的集合。模-余数数据划分方法如下：

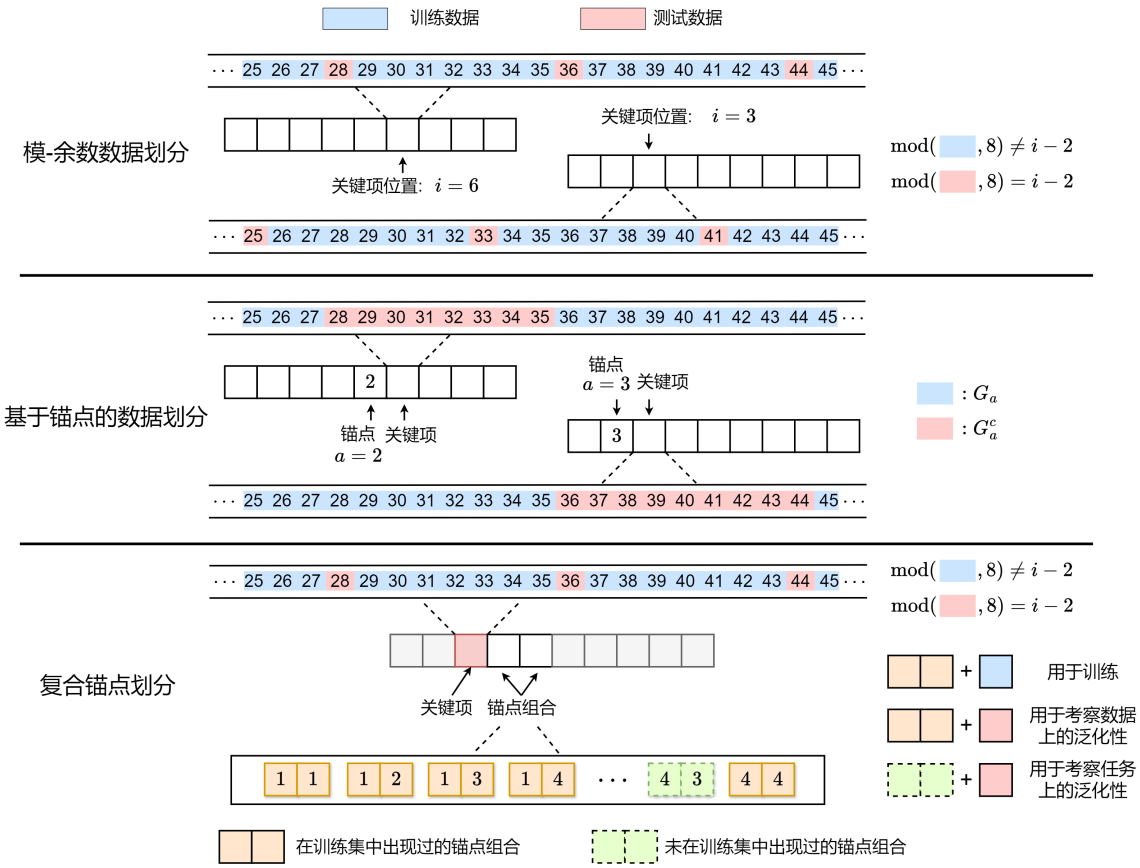


图 12.3: 训练集和测试集划分的三种方式示意图

- 对于训练集的输入序列，如果第 i 个位置的 token 是关键项，那么，这个关键项的值 x 取自 $x \in I \backslash \Gamma_i$，即 x 模 n 不余 i。
- 对于测试集的输入序列，如果第 i 个位置的 token 是关键项，那么，这个关键项的值 x 取自 $x \in I \cap \Gamma_i$，即 x 模 n 余 i。

这种划分方式可以实现完全区分训练集与测试集，在后续训练类语言任务时，以此划分的测试集被作为评价 Transformer 泛化性的一个最基本的指标。但是，评价一个模型泛化性好坏的标准并不是唯一的，为避免模型通过投机的方式（即在真实大模型中不会出现）完全记住所有“$3x$”组合，在下文中，我们提出了基于锚点的数据划分和复合锚点划分这两种更加复杂的数据划分方式来对 Transformer 模型进行全方位评价。

基于锚点的数据划分

假设一个类语言任务数据集由多个单锚点函数构造的语句组成，每个句子中仅出现一个锚点。将锚点集表示为 $A = \{a_j\}_{j=1}^{J}$。这时，可以考虑如下基于锚点的数据划分方法。

将所有 token 的集合表示为 I。在训练集中，锚点 a_j 的关键项属于集合 $G_{a_j} \subsetneq I$，且这些集合满足 $\cup G_{a_j} = I$。在测试序列中，a_j 的关键项属于 $G_{a_j}^c = I \backslash G_{a_j}$。

这里的核心想法是当“x”作为多个锚点的关键项被训练后，模型可以对其作为其他几个关键项时的表现进行准确估计。

复合锚点划分

假设一个类语言任务数据集由多个多锚点函数构造的语句组成。例如设定 [“1”,“2”,“3”,“4”] 为单锚点，它们两两组合形成 16 种复合锚点，任务为 “abx to $g_b(g_a(x))$”（称为复合任务）。那么我们可以考虑使用 15 种复合锚点对应的类语言任务进行训练，在剩下的 1 种复合锚点对应的类语言任务上进行测试。

这种划分方法可以用来考察模型对新的锚点组合的泛化能力，体现了模型在组合推理方面的能力。通过巧妙地设计锚点组合，我们可以构建出难度不同的测试集，全面评估模型的性能。

本节介绍的三种数据集划分方法，为我们提供了多角度研究 Anchor Function 和评估模型泛化能力的工具。在后续的实验中，我们将灵活运用这些方法，深入探究 Transformer 模型的行为特性和内在机制。

12.6.2 数据泛化与任务泛化

数据集的划分自然引出了两种泛化的概念：数据泛化和任务泛化。前者描述了模型对已见任务和未见数据的泛化能力，而后者则与模型对未见任务的泛化能力有关。

数据泛化和任务泛化是评估模型性能的两个重要维度，反映了模型在不同层面上的适应和推广能力。下面我们以复合任务 “abx to $g_b(g_a(x))$” 为例，来讨论这两种泛化的特点和实现方式。在该任务中，锚点 “a” 和锚点 “b” 都取自集合 [“1”, “2”, “3”, “4”]，它们两两组合形成 16 种锚点组合。在这个例子中，我们可以考察对于关键项 “x” 的泛化性，也可以考察在训练其中 15 种锚点组合的任务后，模型在剩下的一种锚点组合任务上的泛化性。

数据泛化。数据泛化主要指的是模型在根据关键项 “x” 划分出测试集上的表现。划分方式可以使用前文提到的模-余数数据划分或者基于锚点的数据划分。对这个测试集，其锚点函数（即任务）已经在训练集中出现过。数据泛化考察模型在面对新的数据样本时，是否能够充分利用训练阶段学到的知识和规律，对未见过的样本做出正确的预测或输出。例如，当模型学会了加法这一操作后，那么对于任何的两个加数，它应当都能够输出正确的加和结果。这要求

模型不仅要记住训练集中的内容，更要掌握任务背后的一般性规律，形成鲁棒的表示和计算能力。

任务泛化。任务泛化主要指的是模型在根据锚点 "ab" 划分出的测试集上的表现。划分方式可以选择一组锚点 "ab" 作为测试集，其他锚点组合作为训练集，即前文提到的复合锚点划分。尽管测试数据中的一些锚点组合是新的（在训练集中不存在），但模型可以利用训练集中现有的复合锚点来推断未见过的复合锚点函数。任务泛化考察模型在面对全新的任务时，是否能够利用已学习的知识和技能，通过组合、类比等方式，快速适应和解决新的问题。例如当模型学习过大量美食、旅行、音乐等情感分析任务后，它应当在影评的情感分析任务中也有较高准确率。这需要模型具备一定的抽象和推理能力，能够在不同任务之间建立联系，提取共性，并灵活地将其应用到新的场景中。

数据泛化和任务泛化这两个概念的提出，为我们全面评估模型的性能提供了重要的视角。通过设计合理的数据集划分方案，构造难度适中、覆盖全面的测试集，我们可以客观地衡量模型在这两个维度上的表现，识别其优势和局限性，为进一步的改进和优化提供依据。

在接下来的章节中，我们将通过一系列基于锚函数类语言任务的实验，深入探究 Transformer 模型在数据泛化和任务泛化方面的特点，揭示其内部工作机制，并讨论可能的改进方向。

12.7 实验结果与讨论

在本节中，我们使用 Transformer 模型对前文图12.1中提到多种锚函数类语言任务进行学习，并探究不同任务下模型的内部机制以及输出偏好。如无特别说明，我们采用一个 4 层的仅有解码器的 Transformer 网络来学习任务。每个解码器层包括 4 个注意力头。详细设置请参考附录 12.9.3。首先，在数据量充分的情况下，Transformer 模型可以更好地学好这些任务，图12.17总结了 Transformer 在学习每个任务时损失和准确率。以下将重点讨论这些类语言任务所模拟的真实对话场景，以及 Transformer 模型在学习这些数据时有什么特点。

12.7.1 恒等学习任务

恒等学习任务即是前文中的 "$3x$ to x" 任务，它模拟了在一句话中找出表示名字的关键词的任务。例如，对于输入 "他来自中国，他的名字是迈克，他喜欢阅读"，在这句话中，"名字是" 是一个锚点，"迈克" 是锚点的关键值（或称为关键项）。

恒等学习任务的锚函数在式 (12.2) 中定义，其中作为提示的锚点项在输入序列中只出现一次。函数的输出是紧跟在特定锚点后面的数字。在我们的实验中，我们使用 "3" 作为锚点项。

为了体现 Transformer 架构的优越性，我们在不同数据量下比较了全连接神经网络（FNN）、

LSTM 和 Transformer 在恒等学习任务中的泛化能力。这三个模型的参数数量分别为 4.16 亿、530 万和 560 万。三种模型的详细设置请参考附录。对于每个实验设置，我们使用 10 个随机种子进行实验。每个实验训练 4000 个 epoch。如图12.4所示，散点是 10 次实验结果的平均值，阴影部分是 10 次独立实验的最大值和最小值。可以看到，Transformer 只需 600 个训练数据就表现出优异的泛化性能，而 LSTM 需要大约 2200 个训练数据才能达到相似的泛化水平。另一方面，FNN 在有限的数据中难以泛化。

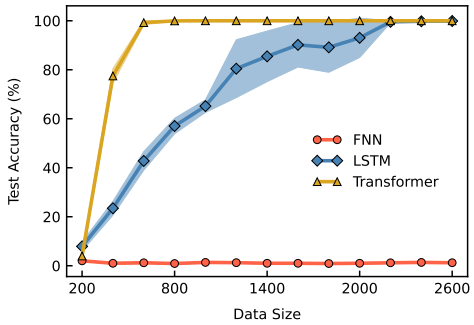


图 12.4: FNN、LSTM 和 Transformer 的性能比较

恒等学习任务是锚函数的一个基本研究任务，我们在后文第12.8节还会详细研究 Transformer 模型在恒等学习任务上具有良好泛化能力的机制。

12.7.2 阅读理解任务

在日常阅读理解中，面对提问，人们通常先在问题中识别关键信息，即锚点，然后在给定文本中搜索与锚点相似的表述。问题的答案通常可以在正文中与锚点相似表述的附近内容中找到。我们设置了以下任务来模拟这个阅读理解任务。

我们定义了不同的锚点值 $a_i \in \{1, 2, \cdots, 8\}$，以及关键项 $x_i \in [20, 100]$。输入序列遵循模式 $(a_1, x_1, a_2, x_2, a_3, x_3, a_4, x_4, a)$，其中 a_i 是锚点，对于 $i \neq j$，我们设定 $a_i \neq a_j$。序列输出的目标值是 x_k，其中索引 k 满足 $a_k = a$。在此任务中，训练集和测试集的划分通过 “模-余数数据划分” 方法完成。通过在总共 1000 个训练数据点上训练 1000 个 epoch，模型便可以在测试集中达到 100% 的准确率。

12.7.3 分类任务

分类任务是阅读理解任务的变体，关注的是与锚点相似而非相同的内容。在这个任务中，我们定义不同的锚点值 $a_i \in \{1, 2, \cdots, 8\}$，以及关键项 $x_i \in [20, 100]$。输入序列遵循模式

$(x_1, a_1, x_2, a_2, x_3, a_3, x_4, a_4, x)$，其中 a_i 是锚点，对于 $i \neq j$，我们规定 $a_i \neq a_j$。输出标签 $a = a_k$，其中 $k = \text{argmin}_{i \in \{1,2,3,4\}} |x_i - x|$。在此任务中，训练集和测试集的划分通过“模-余数数据划分”方法完成。我们使用总共 30000 个数据点进行训练，批量（batch）大小为 50，训练 4000 个 epoch。此时模型在测试集中能够达到 100% 的准确率。

相较于阅读理解任务，分类任务拥有更高的复杂度，这种复杂度主要来源于模型需要对数据之间的位次关系进行编码，以刻画两个任意数据之间的距离。如图12.5所示，我们通过 t-SNE Van der Maaten & Hinton (2008) 可视化了训练后模型关于不同 token 在嵌入空间的向量表示。显然，嵌入向量捕捉到了 token 之间的次序关系，这在分类任务中至关重要。

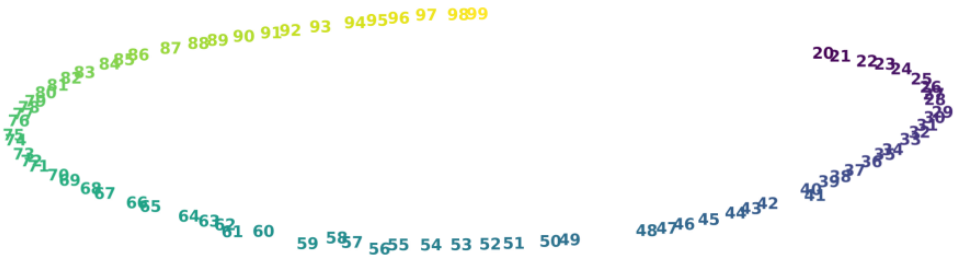


图 12.5: 20 到 99 的数字在训练后的 token 嵌入的 t-SNE 分布

12.7.4 复合任务

复合任务是单锚点任务的自然扩展。在每个输入序列中，总是有两个锚点来指示一个复合函数。一个基本的问题出现了：在复合任务中，模型能否学习构成复合函数的每个单锚点的映射？在这个任务中，我们通过模-余数数据划分和复合锚点划分来分割数据。因此，我们可以研究 Transformer 在数据和任务上的泛化能力。

首先，我们定义四个锚点，每个锚点代表一个特定的映射。然后将这些锚点两两配对，创建 16 个不同的复合锚点函数。选择其中的 15 对来构建训练集。模型的泛化能力在全部 16 种复合锚函数所生成的测试数据集上进行测试。值得注意的是，模型在用于训练的 15 个复合映射上的泛化反映了它在数据上的泛化能力，而在训练集中不存在的复合映射上的泛化反映了它在任务上的泛化能力。

如图12.6左所示，我们用 f_a 表示单锚点 a 的函数，并组合每个单锚点以获得锚点对 $\{a_1, a_2\}$ 的双锚点复合函数 f_{a_1,a_2}。“复合输出”部分中的灰色锚点对没有出现在训练集中。在图12.6右侧，我们通过三个输入示例说明了 10 层模型在训练后的情况。我们发现网络学习了一个特定的复合映射结构。对于第一个锚点 a_1，网络在该锚点位置的输出为 $(f_{a_1} f_{a_1})(x)$，其中 x 是关键项。对于第二个锚点，网络在该锚点位置的输出为 $(f_{a_2} f_{a_1}^{-1})(f_{a_1} f_{a_1})(x)$，它抵消了在锚点 a_1

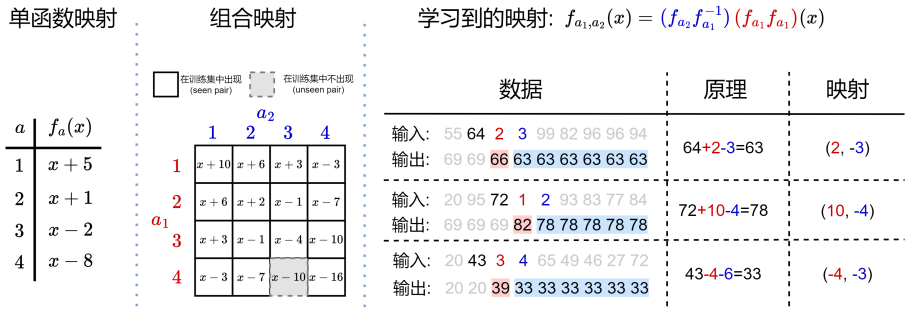


图 12.6: 复合任务设定与模型输出规律示意图

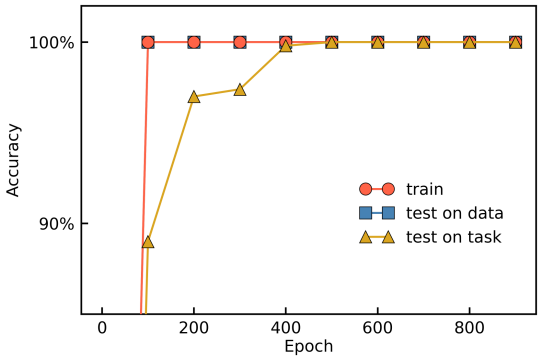


图 12.7: 复合任务的训练准确率和测试准确率

位置的一个额外的 f_{a_1} 的运算。

我们发现模型大小可以显著影响训练后的输出模式。对于 10 层模型，如图12.6右所示，在输入第二个锚点后，网络输出正确答案，并对后续输入 token 持续输出正确答案。而对于层数较少的模型，并不会稳定的出现上述现象，这暗示了模型规模与其稳定性间的联系。

此外我们观察到，训练后的任务测试准确率通常小于数据测试准确率（总是达到 100%）。这表明任务泛化更加困难。为了验证这一点，我们在图12.7中展示了一个学习过程。在该训练过程中，训练数据上的准确率和测试数据上的准确率仅经历 100 个 epoch 便达到 100%，而任务上的泛化性在 400 个 epoch 才达到 100%。模型在数据上的泛化收敛速度远快于在任务上的收敛速度。

12.7.5 工作记忆任务

复合任务展示了模型从复合锚点中学习单个锚点运算的能力。第二个锚点处理来自第一个锚点的输出，而不是直接处理关键项。如果两个锚点之间有一些不相关的项，这种顺序处理需

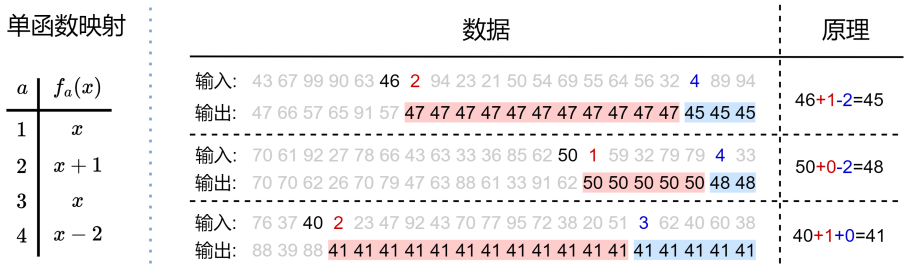


图 12.8: 工作记忆任务设定与模型输出规律示意图

要模型具备工作记忆能力。

具体而言，我们设定第一个锚点 $a_1 \in \{1, 2\}$，第二个锚点 $a_2 \in \{3, 4\}$，它们对应的映射如图 12.8 所示。

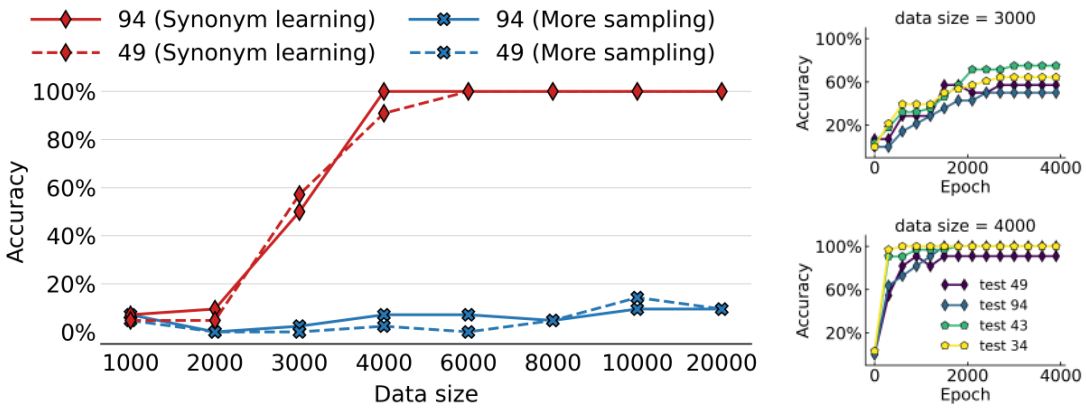
如图 12.8 所示，第一个锚点的输出是通过第一个锚点对关键项运算得到的中间结果。模型通过在不相关 token 的输出中保留这个值来记忆这个中间结果。一旦模型看到第二个锚点，模型将结合中间结果和第二个锚点输出最终正确答案。值得注意的是，由于该问题是个多解问题（对 a_1 增加一个常数 k，a_2 减少一个常数 k，对复合结果没有影响），网络学习到的中间结果可能与我们给定的单个锚点运算不同。

工作记忆任务揭示了 Transformer 模型在处理需要暂时存储和维护信息的复杂任务时的能力。模型不仅能够学习单个锚点的运算，还能在必要时保留中间结果，直到完成最终输出。这种能力对于理解和模拟人类在推理、问题解决等任务中的认知过程具有重要意义。

12.7.6 近义词任务

近义词任务是复合任务的扩展，旨在研究语言模型建模两个锚点之间近义关系的能力。原始数据集中的复合锚点取自：{(1,9), (2,9), (9,1), (9,2), (1,2), (1,4), (2,1), (4,1), (2,4), (4,2)}。每个复合锚点的训练数据集数量相同。然而，如图 12.9(a) 中的叉号点线所示，即使我们不断增大训练数据量，Transformer 网络依然很难学习 (4,9) 和 (9,4) 等复合锚点的映射。

我们设置一个近义锚点 “3”，它执行与锚点 “9” 相同的函数。为了查看 Transformer 网络是否可以通过复合锚点 (4,3) 和 (3,4) 建模锚点 (4,9) 和 (9,4) 的功能，我们将带有近义锚点 “3” 的数据添加到训练集中。具体而言，我们将复合锚点 {(1,3), (3,1), (2,3), (3,2), (4,3), (3,4)} 添加到原始数据集中。在总训练数据量为 M 的情况下，每种包含 “9” 的复合锚点（如 (1,9)）的训练数据数量固定为 100，共 400，而其他复合锚点的训练数据数量为 $(M - 400)/12$。如图 12.9(a) 中的菱形点线所示，只要总数据量大于某个值，网络就可以准确预测 (4,9) 和 (9,4) 的输出。在图 12.9(b) 中，我们展示了两种数据量（分别为 3000 和 4000）情况下的具体学习过



(a) (b)

图 12.9: 比较两种数据增强方法对任务泛化的影响 (a) 以及不同数据量下模型的训练情况 (b)

程。显然，(4,9) 和 (9,4) 的学习略晚于 (4,3) 和 (3,4)。这些结果表明，引入近义词显著影响 Transformer 的任务泛化能力。

近义词任务展示了 Transformer 模型在学习词汇间语义关系方面的潜力。通过引入近义锚点，模型能够将已学习的知识迁移到新的组合中，实现更强的泛化能力。这一发现为利用语言模型处理同义词识别、词义消歧等实际任务提供了新的思路。

12.7.7 前向-后向背诵任务

通常，人们发现背诵一句诗句的下一句诗句（前向任务）比背诵其上一句（后向任务）要容易得多。基于以上经验，我们对 LLM 进行测试，发现其完成前向任务也比后向任务更容易。例如，我们使用 GPT4 测试了包含 20 首诗歌的前向-后向背诵任务。在前向背诵任务中，GPT4 的准确率为 95%，而后向背诵的准确率只有 5%。这与人类语言理解和记忆的认知机制有一定程度上的相似性。基于这一观察，我们设计了一个理想化的前向-后向背诵任务来研究 LLM 的这种不对称性。输入序列遵循模式 “$(*,*,m,3,n,*,*,*,n)$” 或 “$(*,*,m,3,n,*,*,*,m)$”，其中 “3” 是锚点，其他 token 从 $[20, 100]$ 中随机选择，“$*$” 表示该位置的 token 与目标值不相关，“m,3,n” 可以在任何位置。“$(*,*,m,3,n,*,*,*,m)$” 的输出为 “n”（后向任务），而 “$(*,*,m,3,n,*,*,*,n)$” 的输出为 “m”（前向任务）。如图 12.10 所示，Transformer 网络在前向任务中的泛化能力始终优于后向任务。

前向-后向背诵任务揭示了 Transformer 模型在处理顺序信息时的一个有趣偏好。尽管在数据结构上是对称的，但由于 mask 的存在，模型在前向和后向任务上的表现却存在显著差异。通过进一步研究对于这两种数据学习速率与泛化能力的差异，我们可以更好地理解语言模型的

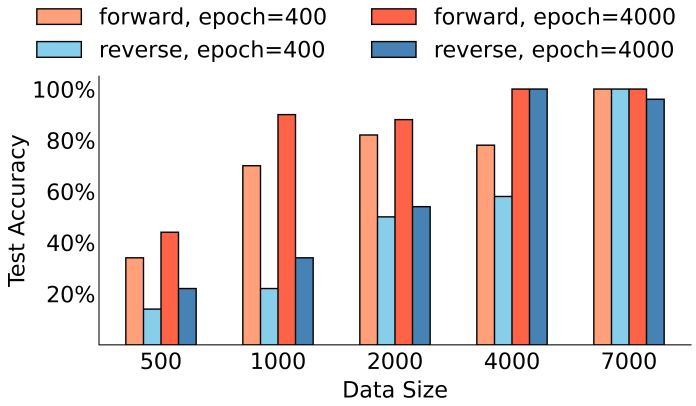


图 12.10: 前向-后向背诵任务中前向预测和后向预测的准确率随训练数据量的变化

内部工作原理，并探索其与人类语言处理的异同。

12.7.8 统计输出任务

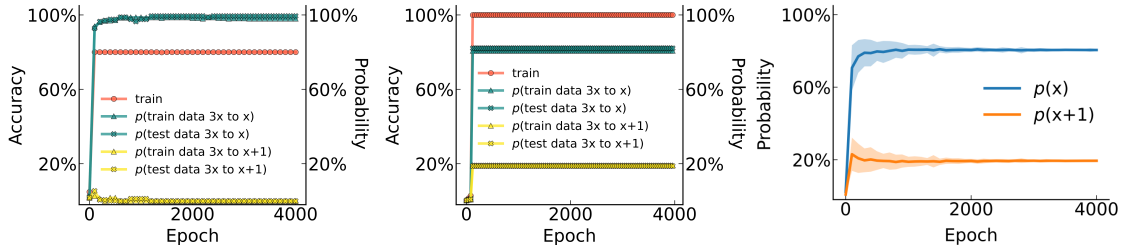
统计输出是一个常见的场景，即序列的输出遵循概率分布。举例来说，我们对一群人的饮食偏好进行统计，假设 80% 的人喜欢吃苹果，20% 的人喜欢吃橙子，那么模型可能更倾向于仅输出苹果，而把少数人的诉求忽视。为了模拟这个场景，我们设计了一个基于恒等学习锚点函数的任务。

为了模拟统计输出的场景，我们设计了两种方法来构建数据集：

- 数据集 S 以与恒等学习任务相同的方式构建，并复制五次，将最后一个副本的标签修改为 $x+1$。这五个数据集合并构成了第一类数据集。
- 数据集 S 以与恒等学习任务相同的方式构建，并随机将 20% 的数据标签改为 $x+1$。这个新获得的数据集称为第二类数据集。

如图 12.11(a) 所示，对于第一类数据，训练准确率只能达到 80%（红色）。这是由于我们数据方式，使得存在相同输入但目标值不同的数据。对于标签为 x 的数据，网络可以达到 100%，而对于标签为 $x+1$ 的数据，网络的准确率为 0%。如图 12.11(b) 所示，对于第二类数据集构造方式，由于每个数据对应的标签是唯一的，所以 Transformer 可以完全学好训练集数据，对于从未出现过的测试集数据，网络有 80% 概率输出 x，20% 概率输出 $x+1$。

我们接下来检查网络是否学习了输出的概率分布。对于输入，我们可以分别记录输出 x 和 $x+1$ 的概率，即输出层的 softmax。如图 12.11(c) 所示，对于第一类数据，随着 epoch 的增加，输出 x 的概率逼近 80%，而 $x+1$ 的概率逼近 20%，两者都满足实验设置。



(a) (b) (c)

图 12.11: Transformer 在统计输出任务中的预测准确率

统计输出任务展示了 Transformer 模型在学习和表示概率分布方面的能力。通过设计合适的数据集，我们可以引导模型学习输入-输出之间的统计关系，而不仅仅是确定性的映射。这种能力对于处理不确定性和生成多样化输出的实际任务（如对话生成、机器翻译等）具有重要意义。

12.7.9 多锚点任务

在模-余数数据划分的情况下，它只确保训练集和测试集中锚点和关键项对（即 $((a, x))$）的位置不同。有人可能会争论说，网络只学习了位移不变性，而没有学习运算。为了验证运算的学习，我们设计了以下多锚点任务。

我们使用十个不同的锚点 $a \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$，并定义锚点函数 $f_a(x) = x + a$，其中 x 是锚点对应的关键项。对于测试数据集，锚点 a 的关键项属于 $G_a^c = [12+8a, 12+8(a+1)-1]$，而对于训练集，锚点 a 的关键项属于集合 $G_a = [20, 100] \backslash G_a^c$，即基于锚点的数据划分。我们的关注点是，在训练后，Transformer 的预测能力是否可以泛化到训练中未见过的锚点和关键项组合的句子。

我们考察了 Transformer 在使用不同锚点构建的数据集上的预测能力。测试集为 $G_a^c = [12+8a, 12+8(a+1)-1]$，训练集为 $G_a = [20, 100] \backslash G_a^c$。如图 12.12 所示，在大多数情况下，模型在基于锚点的数据上表现出良好的泛化能力，即网络学习到了运算而不是位移不变性。对于接近边界的情况（锚点 “1”、“2”、“10”），由于数据集构造方法的原因，锚点处于边界的数据点要比其他的情况要少，这可能是学习边界准确率没有到 100% 的原因。

多锚点任务进一步验证了 Transformer 模型学习运算而非单纯依赖位置信息的特性。通过精心设计的数据划分策略，我们可以考察模型在全新的锚点-关键项组合上的泛化表现，从而更全面地评估其学习成果。同时，这项任务也揭示了边界情况对模型学习的影响，为我们优化数据集构建和模型训练提供了有益的启示。

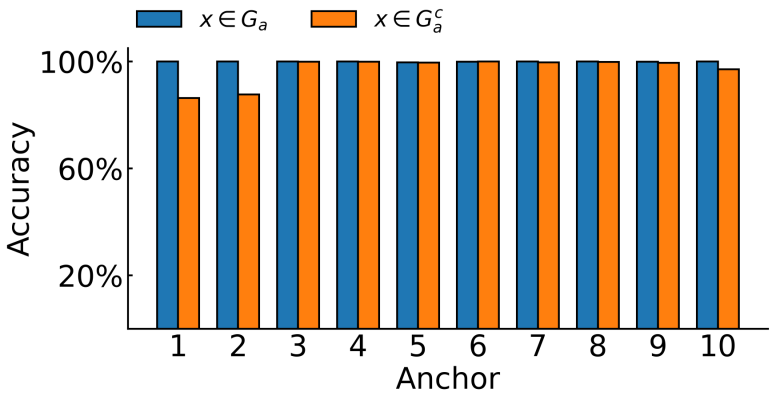
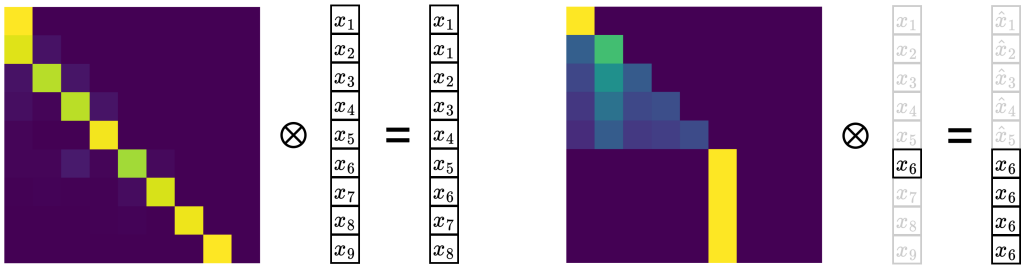


图 12.12: 多锚点任务训练后的准确率

12.8 恒等学习任务的机制研究*

通过前面的实验，我们已经看到 Transformer 模型在各种锚函数任务上展现出了优异的性能。然而，模型的内部工作机制仍然值得进一步探究。在本节中，我们将重点分析一个两层 Transformer 模型在恒等学习任务上的详细机制，以期获得更深入的理解。本节首先简要解释了恒等学习任务在两层模型中的机制，确定了两个关键操作，即移位 (shift) 和广播 (broadcast)。然后，我们找到了一个非常简化的两层模型，该模型可以完成恒等学习任务。移位和广播操作在这个简化模型中也起着关键作用。最后，我们展示了这两个操作在大语言模型 (LLMs) 中的普遍性。

12.8.1 两层模型的简要解释



(a) 第一层, 移位操作　　(b) 第二层, 广播操作

图 12.13: 第一层和第二层的注意力图

在恒等学习任务中, 两层模型的注意力图主要有两个目的: 移位和广播。如图12.13(a) 所示, 第一层的注意力图执行移位操作, 序列 $(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)$ 通过第一层的注意力图移位为 $(x_1,x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8)$。基于残差连接, 移位后的向量和未移位的向量融合, 从而建立锚点项和关键项之间的信息整合。值得注意的是, 该操作是一个位置的移位, 这是因为关键项被设计成锚点的下一个 token。一旦锚点和关键项的信息被融合, 后续的全连接结构将根据这个融合的信息生成正确的标签。

第二层的注意力图执行广播操作。如图12.13(b) 所示, 序列 $(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)$ 被转换为 $(\hat{x_1},\hat{x_2},\hat{x_3},\hat{x_4},\hat{x_5},x_6,x_6,x_6,x_6)$, 其中 $\hat{x_i}$ 是 $x_1,\ldots,x_i$ 的线性组合。这个广播操作将正确的标签传输到最后一个位置, 即输出位置。第一层的注意力图主要基于位置嵌入, 第二层的注意力图主要基于输入序列中锚点的位置。

这个例子说明了注意力结构在语言模型中提供的两个重要操作, 即移位和广播。我们还发现, 这两个操作在大语言模型 (LLMs) 中也很常见 (见后面章节的实验)。因此, 锚点函数可以作为理解基于 transformer 的语言模型的一种基准函数。

12.8.2 简化的两层模型的机制

在本节中, 我们采用了一个简化的两层模型, 该模型移除了包括掩码、层归一化、残差连接、线性变换和全连接网络等结构, 以研究恒等学习任务的机制。我们首先给出简化模型的定义。

令 $X^{(1)} \in \mathbb{R}^{n\times d_m}$ 表示经过词嵌入和位置嵌入后的输入序列, 即 $X^{(1)} = X^{\mathrm{em}} + X^{\mathrm{pos}}$。我们选择 d_m 等于字典大小 d, 以简化 transformer 中的线性算子, 并删除投影层。最终输出的计算如下:

$$\begin{aligned}
\mathrm{Attn}^{(1)} &= \mathrm{softmax}\left(\frac{X^{(1)}W^{Q(1)}W^{K(1)T}X^{(1)T}}{\sqrt{d}}\right), \quad X^{(2)} = \mathrm{Attn}^{(1)}X^{(1)}W^{V(1)},\\
\mathrm{Attn}^{(2)} &= \mathrm{softmax}\left(\frac{X^{(2)}W^{Q(2)}W^{K(2)T}X^{(2)T}}{\sqrt{d}}\right), \quad X^{(\mathrm{out})} = \mathrm{Attn}^{(2)}X^{(2)}W^{V(2)}.
\end{aligned}$$

它可以简单地表示为

$$X^{(\mathrm{out})} = \mathrm{Attn}^{(2)}\mathrm{Attn}^{(1)}X^{(1)}W^{V(1)}W^{V(2)}.$$

值得注意的是,$\mathrm{Attn}^{(l)}$ 融合了来自 $X^{(l)}$ 的信息。因此, 上述公式并不代表一个简单的线性变换。

图12.14描述了该模型的机制解释, 其中第一层注意力将关键项 "x" 的信息移位到锚点项 "3" 的位置, 并将锚点项的信息广播到其他位置。具体而言, 第一层的注意力矩阵使输出向量

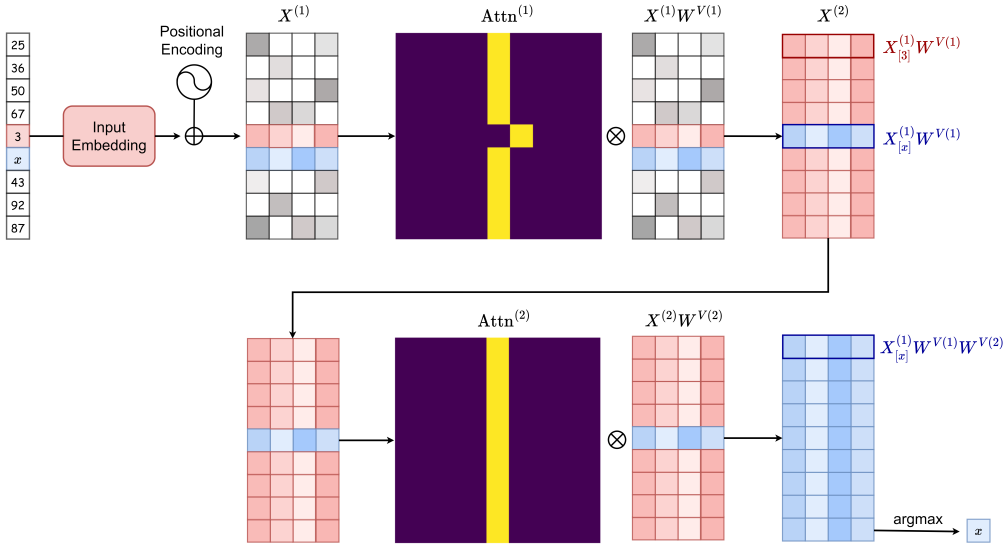


图 12.14: 恒等学习任务中简化的两层 transformer 模型的机制

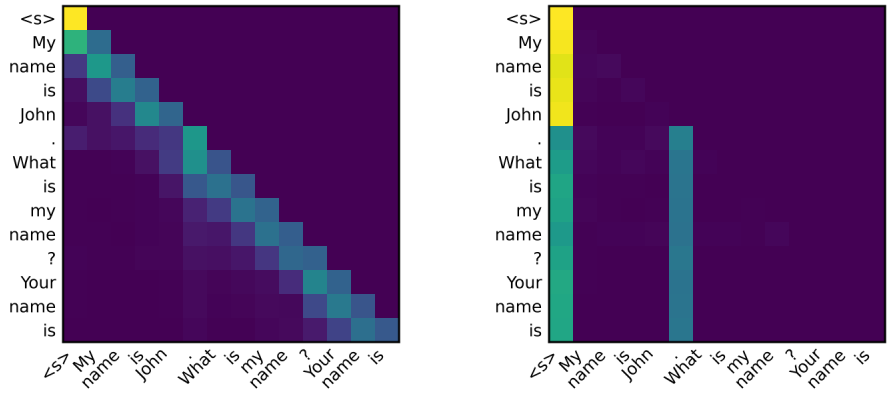
变为 $X^{(1)}_{[3]}W^{V(1)}$ 或 $X^{(1)}_{[x]}W^{V(1)}$, 其中 [3] 和 $[x]$ 表示输入序列中 “3” 和 “x” 的位置。因此, 输入到第二层的矩阵行向量只有两种类型: 包含锚点项信息的向量和包含关键项信息的向量。这种结构也有利于构建第二层注意力图 $\text{Attn}^{(2)}$。注意力矩阵 $\text{Attn}^{(2)}$ 执行广播操作, 即将包含关键项信息的向量广播到所有位置。具体而言, 第二层的注意力矩阵使所有的输出向量变为 $X^{(1)}_{[x]}W^{V(1)}W^{V(2)}$。$W^{V(1)}W^{V(2)}$ 起到分类的作用。最终预测通过函数 $\text{argmax}(\cdot)$ 获得。

在上述简化模型中,$\text{Attn}^{(1)}$ 在信息传递中起着重要作用, 可以看作是移位注意力和广播注意力的组合。

12.8.3 Llama2-7B 中的移位和广播

我们发现, 在学习锚点函数中起重要作用的注意力的移位和广播操作, 在大语言模型 (LLMs) 中也非常常见。我们以 Llama2-7B Touvron et al. (2023) 为例来演示这两种操作。

与恒等学习任务类似, 我们使用 “My name is John. What is my name? Your name is” 作为 Llama2-7B 的输入, 并观察不同层的注意力图。图12.15(a, b) 展示了 Llama2-7B 中的两个头。左边的是第二层中的一个头, 表现出明显的移位操作; 第二个是第 29 层中的一个头, 表现出明显的广播操作。附录中提供了多个头的各种注意力图。这两种操作都相当普遍。同时基于对所有头的观察, 我们注意到, 浅层注意力倾向于移位操作, 而深层注意力倾向于广播操作。



(a) 第二层的第 23 个头　　(b) 第 29 层的第 17 个头

图 12.15: Llama2-7B 的注意力示例

12.8.4 移位和广播操作的讨论

在恒等学习这个类语言任务中，我们发现 Transformer 的注意力结构呈现出移位和广播两种结构。其中，移位操作是依赖于任务而产生的，并且不依赖于输入序列，其形成原因是因为锚点和关键项之间的距离是固定的，注意力结构学习这种位置关系并执行移位操作。而广播操作的呈现形式主要依赖于输入序列，它可以简化网络，例如忽略不相关的 token，为中间结果提供工作内存等。从复杂性的角度来看，广播操作使神经网络能够使用低复杂度的函数来完成任务。

移位和广播只是 Transformer 注意力矩阵基本操作中的两种。注意力矩阵还存在其他操作，例如，归纳头将当前 token 与先前上下文中的类似 token 关联起来Elhage et al. (2021), Olsson et al. (2022)。未来对各种锚点函数的研究将揭示越来越多的注意力结构以及其他组件的基本操作。

12.9 实验设置

本节介绍了我们在各类 Anchor Function 任务上进行实验的具体设置，包括模型结构、损失函数、优化算法和超参数等。通过合理的实验设计，我们可以全面评估模型在不同任务上的性能表现，深入分析其内部机制。

12.9.1 模型结构

我们采用了基于 Transformer 的结构来学习各种任务。图 12.16 展示了 Transformer 网络的一个通用模块。

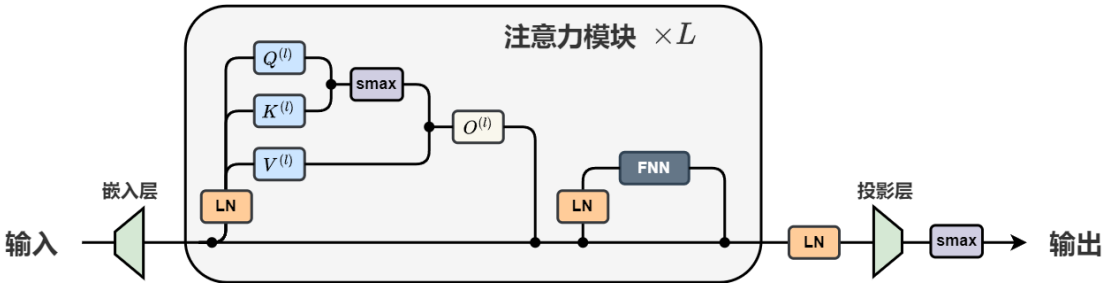


图 12.16: Decoder-Only Transformer 模型示意图

12.9.2 损失函数

在本研究中，我们使用了 Transformer 架构，它接受一个长度为 $n = 9$ 的输入序列，随后输出一个矩阵 $X^{(\text{out})} \in \mathbb{R}^{n \times d}$，其中 d 是字典的大小。对于第 i 个输入序列，损失函数是 softmax 后最后一个 token 输出的交叉熵（用 $\mathbf{x}_n^{(\text{out},i)} \in \mathbb{R}^d$ 表示）。在整个训练数据集上的总损失计算如下：

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{d} \boldsymbol{y}_c^{(i)} \log(\mathbf{x}_{n,c}^{(\text{out},i)}),$$

其中 $\boldsymbol{y}^{(i)}$ 是第 i 个序列的 one-hot 标签，N 表示训练数据集的大小。

12.9.3 超参数设置

如无特殊说明，在后续任务中，我们使用的 Transformer 网络以及作为对比实验的 LSTM 和 DNN 网络训练时的超参数如下：

Transformer：我们使用一个 4 层的仅解码器的 Transformer 网络来学习各种任务。每个解码器层有 4 个头。选择的优化器是 AdamW。批量大小设置为 100。初始学习率为 2e-5。在前 400 个 epoch 中，使用预热学习率更新策略将学习率逐渐增加到 2e-4，然后使用余弦退火策略将学习率衰减回 2e-5。总共训练 4000 个 epoch。

LSTM：我们使用一个 4 层的 LSTM 来学习各种任务。选择的优化器是 AdamW。批量大小设置为 100。初始学习率为 2e-5。在前 400 个 epoch 中，使用预热学习率更新策略将学习率逐渐增加到 2e-4，然后使用余弦退火策略将学习率衰减回 2e-5。总共训练 4000 个 epoch。

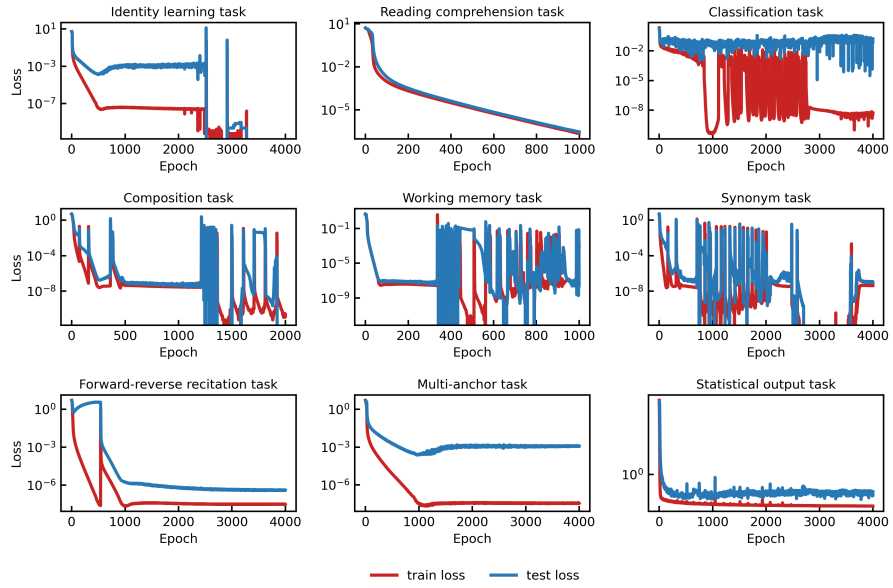
DNN：我们使用一个 4 层的 DNN 来学习各种任务。选择的优化器是 AdamW。批量大小设置为 10。初始学习率为 2e-5。在前 400 个 epoch 中，使用预热学习率更新策略将学习率逐渐增加到 2e-4，然后使用余弦退火策略将学习率衰减回 2e-5。总共训练 4000 个 epoch。

12.10 Transformer 在各类类语言任务上的训练误差和预测准确率

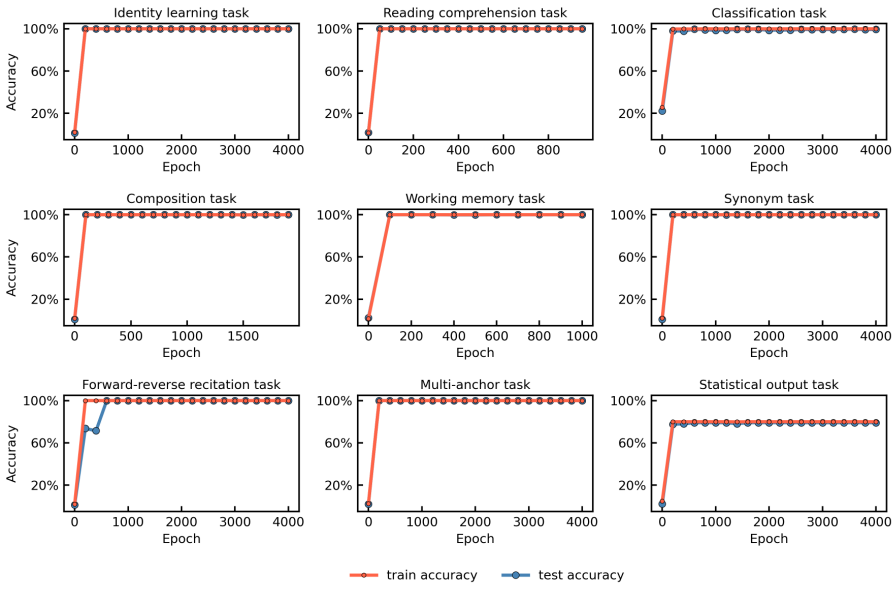
图 12.17展示了 Transformer 在各类类语言任务上的训练误差和预测准确率。图中九条误差曲线和九条预测准确率曲线与正文 9 个任务相对应。

12.11 习题

- 对于图片 12.1所示的 9 种类语言任务，写出它们的锚函数表达式。
- 为什么前向背诵任务比后向背诵任务容易?



(a) 损失



(b) 准确率

图 12.17: 各任务的损失和准确率

Chapter 13

参数初始值对模型推理能力的影响

通过前文的介绍，我们已经了解到大规模 Transformer 模型展示出了前所未有的能力，甚至被誉为“通用人工智能的火花”。尽管取得了巨大成功，但它们在推理能力方面仍存在显著差距，特别是在系统性地处理新颖场景时。例如，语言模型常常难以一致地应用逻辑规则或将知识扩展到不熟悉的上下文中，这凸显了它们在系统性推理方面的局限性，而系统性推理是人类认知的关键方面。系统性推理中的一个关键挑战是组合推理（compositional reasoning），即从已知概念泛化到新颖组合的能力。人类在这一领域表现出卓越的能力，例如，一旦孩子学会了“跳”的概念，他们就能轻松理解“向后跳”或“绕着圆锥跳两圈”等扩展。这种组合技能是人类推理不可或缺的一部分，也一直是神经网络领域争论的焦点。虽然现代模型在各种任务上取得了卓越的性能，但它们执行组合推理的能力，尤其是在分布外（Out-of-Distribution, OOD）场景下，仍然有限。

这引发了关于如何准确解释 Transformer 在组合任务上能力的关键开放性问题：Transformer 是真正学习了数据中的组合基元（compositional primitives），还是主要依赖于记忆输入-输出映射？当它们在组合任务上失败时，这些错误是系统性的，还是反映了更深层次的推理结构缺失？解决这些问题对于理解它们的机制和局限性至关重要。此外，理解这些问题可以为制定增强模型泛化（尤其是在 OOD 场景下）的策略提供信息。在锚函数章节中，我们构造了复合任务，并进行了一些初步探索，并发现 Transformer 有学习到未见锚对的能力。

在本章中，我们将进一步分析复合任务中的内在机制。结果显示，初始化对模型的学习方式有很大影响。初始化尺度较小时，模型倾向于推断，即学习“推断解”；较大时，则倾向于记忆，即学习“对称解”。分析表明，对称解直接组合锚信息，复杂性较高；推断解为每个单锚学习单独映射，复杂性较低。特别的，推断解与前文提到的凝聚现象存在密不可分的关联。这些发现强调了初始化尺度在 Transformer 组合任务学习中的关键作用，针对任务类型调整尺度能

够提升性能。

科普篇

- **什么是复杂度控制 (Complexity Control)？**复杂度控制是指在训练神经网络时，通过调整某些设置来影响模型最终学习到的解的“复杂程度”。在本章中，我们主要关注两种方式：
  - **参数初始化尺度 (Parameter Initialization Scale)：**模型训练开始时，参数（如权重）被赋予的初始值的大小。尺度小意味着初始值接近于零，尺度大意味着初始值相对较大。
  - **权重衰减 (Weight Decay)：**一种正则化技术，通过在损失函数中加入一个惩罚项来限制参数的大小，防止模型变得过于复杂。权重衰减系数越大，惩罚越强。

  通过控制这两个因素，我们可以引导模型学习不同类型的解。
- **复杂度控制为什么重要？**模型的复杂度与其学习策略和泛化能力密切相关。复杂度过高可能导致模型仅仅“死记硬背”训练数据（记忆），而难以泛化到未见过的数据（尤其是 OOD 数据）。复杂度过低则可能无法充分学习数据中的复杂模式。合适的复杂度控制有助于模型找到平衡，既能拟合训练数据，又能学习到底层的规律（推理），从而更好地泛化。
- **复杂度控制如何影响模型的学习方式？**本章研究发现：
  - **强复杂度控制（小初始化尺度 + 大权重衰减）：**倾向于引导模型学习“**推理型解 (Reasoning-based Solution)**”。模型会试图找出数据背后的基本规则或运算单元（基元），并通过组合这些基元来解决问题。这种解通常更简单、更结构化，具有更好的 OOD 泛化能力。
  - **弱复杂度控制（大初始化尺度 + 小权重衰减）：**倾向于引导模型学习“**记忆型解 (Memory-based Solution)**”。模型更可能直接记住输入和输出之间的特定映射关系，而不是去理解底层的组合规则。这种解虽然能在训练数据和相似的 ID 测试数据上表现良好，但在 OOD 场景下通常表现较差。
- **什么是推理型解和记忆型解？**
  - **推理型解：**模型学习了任务的基本组成部分（例如，单个算术运算），并在遇到新组合时能灵活地应用这些基本部分进行推理得到结果。这种解体现了模型对任务结构的理解。

- **记忆型解：**模型主要记住了训练中见过的特定输入-输出对。当遇到未见过的新组合时，由于缺乏对基本规则的理解，模型难以正确处理。

- **复杂度控制如何影响模型的泛化能力（尤其是 OOD 泛化）？**由于 OOD 任务通常需要模型理解并应用潜在规则到新的组合上，因此，能够学习推理型解的模型通常具有更好的 OOD 泛化能力。通过采用强复杂度控制（小初始化、大权重衰减），我们可以促使模型学习这种推理型解，从而提升其在 OOD 任务上的表现。

13.1 定义

我们首先介绍将在本章中使用的关键定义。图 13.1 简要描述了我们的定义。

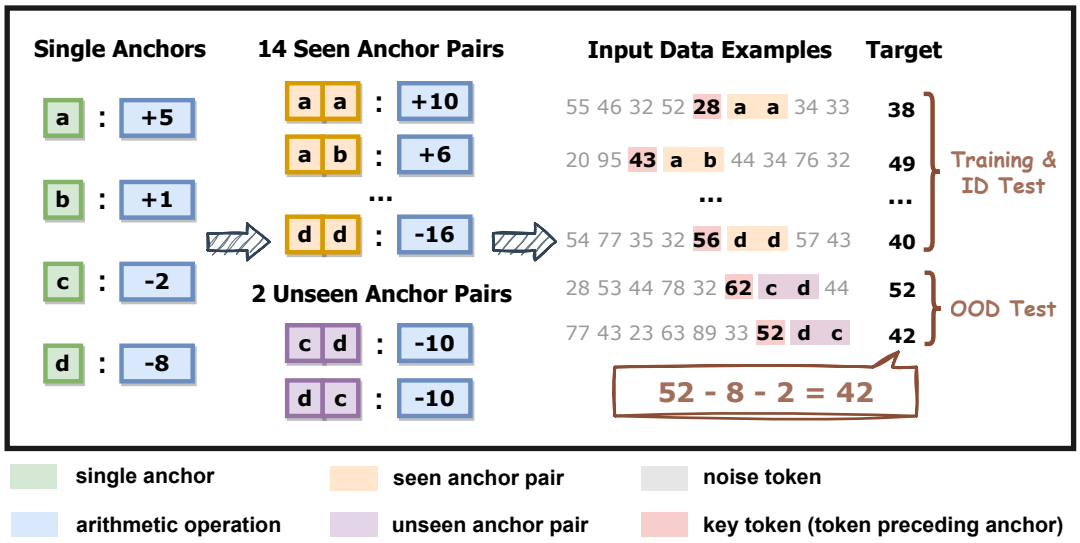


图 13.1: 组合任务的实验设置。左：单锚（即 a, b, c, d）对应特定的算术运算。中：训练期间，16 个可能的锚对中的 14 个出现在训练集中，剩余的锚对（c, d）和（d, c）作为未见任务被保留（训练期间不出现）。右：输入序列包含一个锚对、锚对之前的一个关键 token，以及与目标无关的噪声 token。我们使用来自 14 个已见锚对生成的数据构建互斥的训练集和 ID 测试集，而来自剩余 2 个未见锚对的数据用于形成 OOD 测试集。

13.1.1 双锚点复合函数 (Two-anchor composite function)

双锚点复合函数将两个顺序运算应用于特定 token。这种设置使我们能够剖析模型是将这些运算学习为离散的、可组合的规则，还是仅仅记忆特定的输入-输出映射。一个双锚点复合

函数 $f(X):\mathbb{R}^n \to \mathbb{R}$ 定义为：

$$f(x_1,\ldots,x_n) = g\left(g(x_{i-1};x_i);x_{i+1}\right), \tag{13.1}$$

其中 $x_i, x_{i+1} \in A$。这里，输入序列 $X = (x_1,\ldots,x_n)$ 包含 n 个 token。指定一个锚集合 $A = \{a_1, a_2, \ldots, a_J\}$，其中每个 token $a_k \in A$ 对应一个函数 $g(x;a_k)$。在每个 X 中，有且仅有一对连续元素属于 A，例如 $x_i, x_{i+1} \in A$。我们将紧邻锚对之前的 token 称为 **关键 token (key token)**。为了简化符号，我们将双锚点复合函数记作 $f(x_{i-1};x_i,x_{i+1})$ 以强调锚对 (x_i,x_{i+1}) 和关键 token x_{i-1}。

在本研究中，我们设置锚集合 $A = \{a,b,c,d\}$。每个锚 token 对应一个特定的函数：

$$\begin{aligned} g(x;a) &= x+5, \quad g(x;b) = x+1, \\ g(x;c) &= x-2, \quad g(x;d) = x-8. \end{aligned} \tag{13.2}$$

示例： 假设我们有一个输入序列 $X = (23,a,b,43,46,74,54,44,72)$。在这个序列中，第二个和第三个 token（即 token a 和 b）属于锚集合 A，因此形成了一个锚对 (a,b)。紧邻锚对之前的 token 23 被称为关键 token。

对于这个输入序列 X，双锚点复合函数的计算过程如下：

$$\begin{aligned} f(X) &= f(23;a,b) = g(g(23;a);b) \\ &= g(23+5;b) = g(28;b) = 28+1 = 29. \end{aligned}$$

13.1.2 数据生成 (Data Generation)

输入数据： 在本研究中，我们使用四个锚（即 a,b,c,d）和从 20 到 100 采样的数值 token 构建输入数据集。对于每个序列，选择两个锚（允许重复）形成一个锚对，序列由这个锚对以及其他随机采样的数值 token 组成。紧邻锚对之前的 token 被指定为关键 token，其余 token 被视为与目标无关的 **噪声 token (noise tokens)**。噪声 token 主要用于将训练集与 ID 测试数据集分开。四个锚形成了 16 个锚对，其中 (c,d) 和 (d,c) 被指定为 **未见锚对 (unseen anchor pairs)**。训练数据集基于剩余的 14 个 **已见锚对 (seen anchor pairs)** 构建。

目标： 目标是关键 token 经过双锚点复合函数处理后的输出，即双锚点复合函数对应的输出值。

数据集划分： 我们将数据集分为三类：训练集、ID 测试集和 OOD 测试集。对于训练集和 ID 测试集，锚对选自 14 个已见锚对，不包括 (c,d) 和 (d,c)。对于 OOD 测试集，锚对取自 (c,d) 或 (d,c)。

13.1.3 泛化 (Generalization)

数据集的划分自然引出了以下两种泛化概念：

- **ID 泛化 (In-Distribution Generalization)：**在 ID 测试集上的泛化能力，其中所有锚对都已在训练集中出现过。
- **OOD 泛化 (Out-of-Distribution Generalization)：**在 OOD 测试集上的泛化能力，其中的锚对在训练集中未出现过。

13.1.4 模型架构和基本实验设置 (Model Architecture and Basic Experimental Setups)

为了进行基础分析并简化机制探索，我们首先采用单头 Transformer 模型。随后的实验将这些发现扩展到 GPT-2 的多头架构，以确保我们结论的普适性。以下仅介绍单头注意力模型的架构，多头情况是单头模型的自然扩展。

输入序列表示为一个独热向量 X^{in}。词嵌入 X^{em} 和第一个 Transformer 块的输入 $X^{(1)}$ 计算如下：

$$X^{\text{em}} = X^{\text{in}} W^{\text{em}}, X^{(1)} = X^{\text{em}} + X^{\text{pos}}, \tag{13.3}$$

其中 X^{pos} 是一个可训练的位置向量。对于第 l 层，Q, K, V 定义如下：

$$Q^{(l)} = X^{(l)} W^{Q(l)}, K^{(l)} = X^{(l)} W^{K(l)}, V^{(l)} = X^{(l)} W^{V(l)}. \tag{13.4}$$

第 l 层的注意力矩阵 $\text{Attn}^{(l)}$ 及其后续输出 $X^{\text{qkv}(l)}$ 计算如下：

$$\begin{aligned} \text{Attn}^{(l)} &= \text{softmax}\left(\frac{Q^{(l)} K^{(l)T}}{\sqrt{d_k}}\right) \text{ (带掩码)}, \\ X^{\text{qkv}(l)} &= \text{Attn}^{(l)} V^{(l)}. \end{aligned} \tag{13.5}$$

第 l 层注意力层的输出为：

$$\begin{aligned} X^{\text{ao}(l)} &= \text{LN}(X^{(l)} + X^{\text{qkv}(l)} W^{\text{attn},l}), \\ X^{(l+1)} := X^{\text{do}(l)} &= \text{LN}(\text{MLP}(X^{\text{ao}(l)}) + X^{\text{ao}(l)}). \end{aligned} \tag{13.6}$$

其中 “LN” 指的是层归一化 (Layer Normalization)。最终输出通过一个线性投影层投影最后一层 $X^{\text{do}(L)}$ 的输出，然后通过 softmax 操作和 argmax 获得预测的 token。

我们使用 $f_{\boldsymbol{\theta}}(x; a_1, a_2; \boldsymbol{n})$ 表示神经网络对于关键 token 为 x、锚对为 (a_1, a_2) 以及噪声 token 序列为 $\boldsymbol{n}$ 的序列的输出。此时，输入序列由 $\boldsymbol{n}$ 中的噪声 token 的值和位置信息唯一确定。

对于基本实验设置，我们在序列的最后一个 token 上使用交叉熵损失，并使用带权重衰减的 Adam 优化器优化模型。

13.1.5 初始化和正则化参数 (Initialization and Regularization Parameters)

- **初始化率 (Initialization Rate):** 初始化率，记作 γ，控制模型内参数初始化的尺度。具体来说，模型参数通过从正态分布 $\mathcal{N}\left(0, \left(\frac{1}{d_{\text{in}}^{\gamma}}\right)^2\right)$ 中采样来初始化，其中 d_{in} 表示输入的维度。较高的 γ 导致较小的初始化尺度。

- **权重衰减系数 (Weight Decay Coefficient):** 权重衰减系数是训练期间 L_2 正则化的超参数，记作 λ。它通过向损失函数添加 $\lambda\|W\|_2^2$ 来控制正则化强度，其中 W 是模型的权重。

13.2 复合函数解的阶段划分 (Phases of Solutions for Composite Functions)

在本节中，我们探讨基于 Transformer 的模型（具体为 GPT-2）在使用 Section 13.1.2 定义的 ID 和 OOD 数据集进行组合任务学习时的动态。我们的研究重点是理解初始化尺度和权重衰减系数的变化如何影响模型的泛化能力。

首先，我们评估不同初始化尺度对模型 ID 和 OOD 性能的影响，同时将权重衰减系数固定为 0.01，如图 13.2A 所示。横坐标表示初始化率 γ，纵坐标表示在 ID 数据（蓝色）和 OOD 数据（红色）上达到的准确率。根据 ID 和 OOD 泛化能力，我们将不同的初始化设置分为三个阶段:

- **阶段 1 (差 ID 和差 OOD 泛化):** 模型主要依赖记忆，ID 和 OOD 泛化能力都有限。这对应于非常大或非常小的初始化尺度（γ 值非常小或非常大）。

- **阶段 2 (好 ID 但差 OOD 泛化):** 模型学习了复合锚映射（即特定锚对的整体效果），实现了良好的 ID 泛化，但缺乏对单个锚运算的抽象理解，导致 OOD 泛化较差。这通常出现在中等到较大的初始化尺度（γ 值中等偏小）。

- **阶段 3 (好 ID 且好 OOD 泛化):** 模型学习了单个锚的映射，从更高层次理解了任务结构，使得在未见锚对上具有强大的 OOD 泛化能力。这通常需要较小的初始化尺度（较大的 γ 值）。

为了进一步阐明模型的推理能力，我们检查了其对 **交换性（commutativity）** 的学习情况，这是加法任务中的一个基本属性。具体来说，我们评估当将未见锚对从 (d, c) 交换为 (c, d) 时，保持关键 token 和噪声 token 不变，模型的输出是否保持一致。（注：我们不关注输出与目标值的关系。对于 OOD 准确率高的实例，模型自然满足交换性，因为输出与目标值一致。）

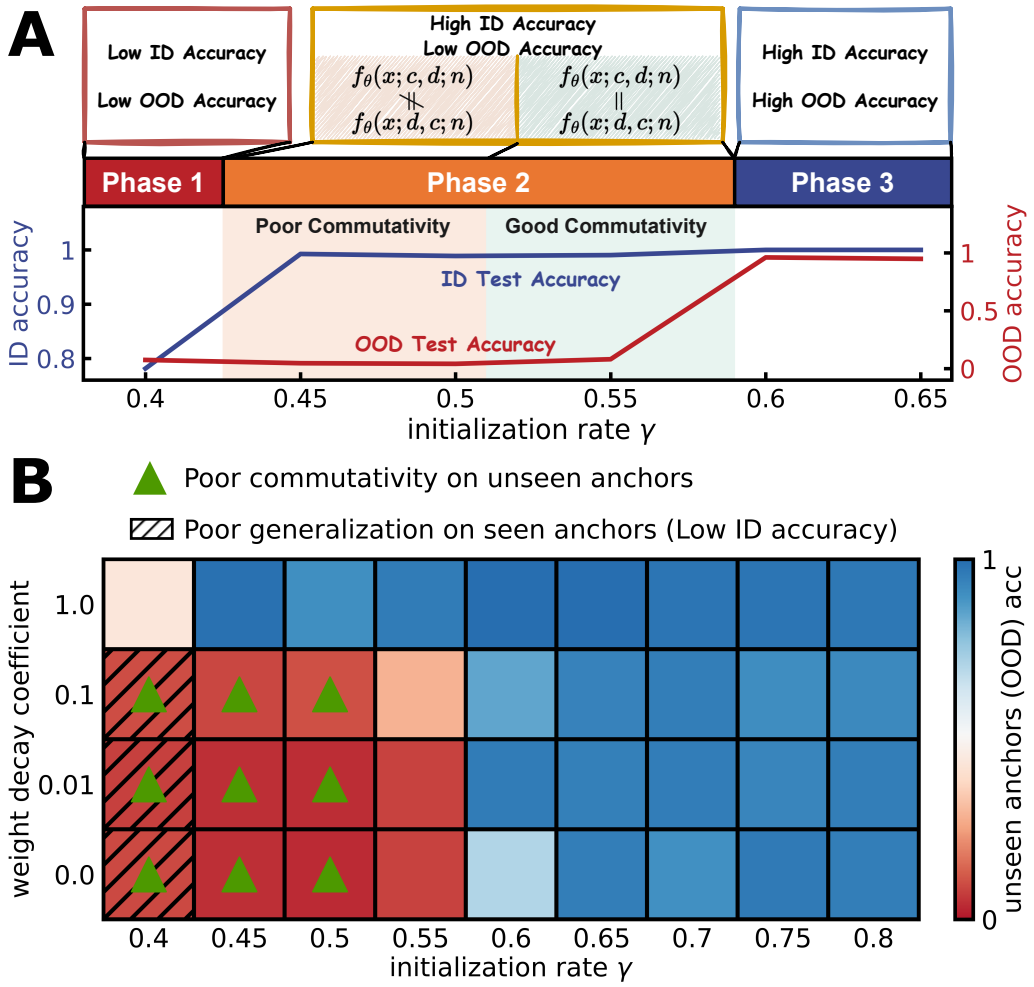


图 13.2: **(A)** 固定权重衰减系数为 0.01 时，GPT-2 模型在组合任务上的 ID 和 OOD 泛化。横坐标表示初始化率 γ，对应于参数初始化所用正态分布的标准差 $(1/d_{\text{in}})^{\gamma}$。纵坐标表示 ID（蓝色）和 OOD（红色）数据的准确率。不同的阶段根据其 ID 和 OOD 泛化能力进行分类。**(B)** 热力图展示了 GPT-2 模型在组合任务上的 OOD 泛化，未见锚对上的准确率由颜色强度表示。横坐标与图 A 相同，纵坐标表示权重衰减系数。每个设置反映了三次独立试验的平均结果。条纹区域表示 ID 泛化较差（ID 准确率 $< 90\%$）。绿色三角形突出显示了在切换未见锚对 (c, d) 和 (d, c) 时，交换性较差的情况（交换性概率 $< 70\%$）。

在阶段 2 内部，交换性表现出依赖于初始化率的变化。对于较大的初始化率 γ（小参数初始化尺度），模型将对称的锚对视为统一实体，因此表现出良好的交换性。形式上，对于大多数关键 token x 和噪声序列 $\boldsymbol{n}$，$f_{\boldsymbol{\theta}}(x; c, d; \boldsymbol{n}) = f_{\boldsymbol{\theta}}(x; d, c; \boldsymbol{n})$ 成立。而对于较小的初始化率 γ（大参数初始化尺度），模型将每个锚对视为独立的映射，导致较差的交换性。

除了初始化尺度，我们还研究了权重衰减系数在塑造模型泛化能力中的作用，如图 13.2B 所示。该分析考察了变化的初始化率 γ 和权重衰减系数对 OOD 性能的综合影响，由热图颜色表示。条纹区域表示 ID 泛化较差（ID 准确率 $< 90\%$），而绿色三角形表示在切换未见锚对从 (d, c) 到 (c, d) 时交换性概率低于 70% 的情况。

这种模式在所有试验中都保持一致，结果是三个随机种子的平均值。我们得出结论，随着初始化率和权重衰减系数逐渐增大（即复杂度控制增强），模型经历了从基本记忆到对任务结构更深层次理解的转变，从而在复杂的泛化任务中增强了推理能力并掌握了底层规则。

我们将初始化率和权重衰减系数的调整视为 **复杂度控制**的统一方面，因为这些参数的值显著影响模型的复杂度特征。因此，具有不同复杂度水平的模型表现出截然不同的内部机制。在后续章节中，我们将进一步探讨不同阶段模型的机制和复杂度差异。

13.3 通过注意力掩蔽策略分析不同阶段的机制 (Mechanism Analysis Across Different Phases Through Attention Masking Strategies)

在本节中，我们通过选择性地 **掩蔽（masking）** 特定信息回路来分析不同阶段背后的机制。这种方法使我们能够隔离和检查关键 token 和锚对的贡献，系统地研究输出空间，并揭示模型在不同阶段的内部机制。具体来说：i) 我们通过掩蔽关键 token 信息来研究不同锚对在输出空间中的分布模式（图 13.3A, 13.3C）。ii) 我们通过掩蔽第二个锚的信息来探索关键 token 与单个锚在输出空间中的分布模式（图 13.3B, 13.3D）。

为此，我们在模型中掩蔽特定回路，以从输出中排除特定 token 的信息（masked activation），使我们能够研究仅存在其他输入信息时模型的行为。掩蔽是通过禁用特定回路来实现的，如图 13.3A 和 13.3B 中的虚线灰色线所示，同时保留未掩蔽 token 的正常回路（实线蓝色线）。为了保留正常回路的结构，掩蔽是在 softmax 操作后的注意力矩阵上应用的。为简化分析，我们使用一个两层、单头的 Transformer 模型。

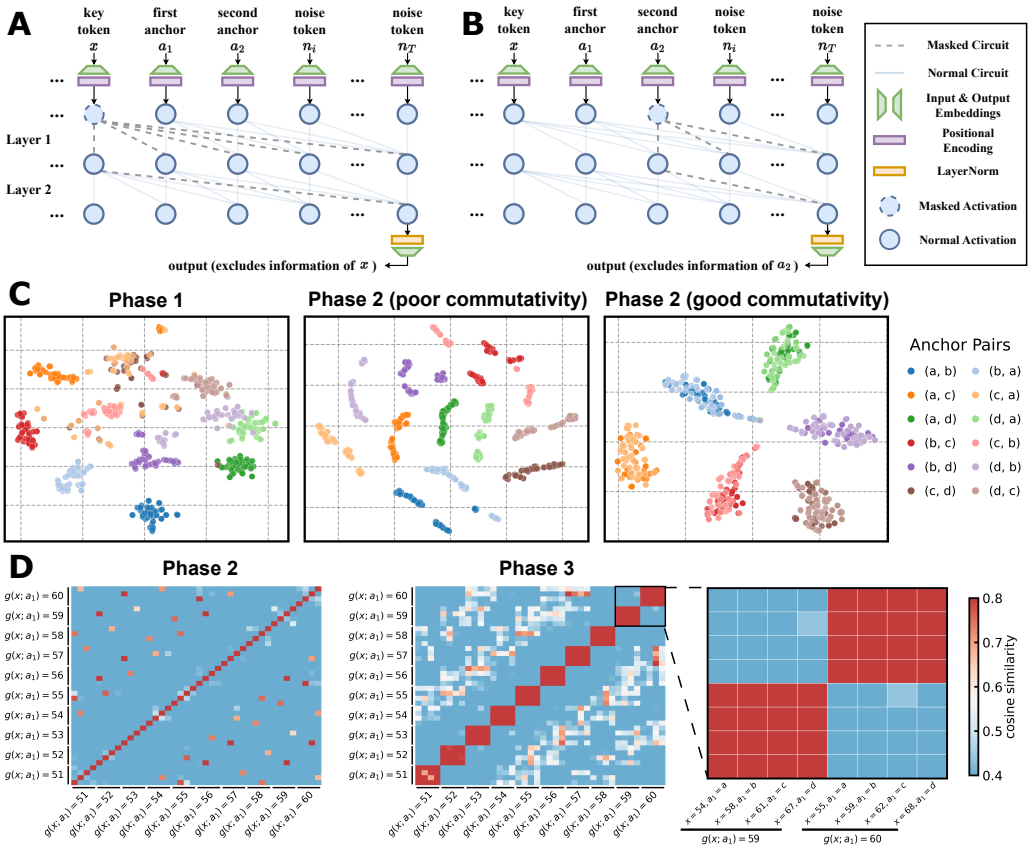


图 13.3: **(A, B)** 应用于 Transformer 模型的掩蔽策略，用于分析特定 token 的贡献。**(A)** 关键 token (x) 被掩蔽，将其信息从模型输出中移除，允许分析受锚对 (a_1, a_2) 和噪声 token 影响的输出空间分布。**(B)** 第二个锚 (a_2) 被掩蔽，以隔离关键 token (x) 和第一个锚 (a_1) 之间的交互。虚线灰色线表示被掩蔽的回路，实线蓝色线表示正常回路。空心节点描绘了被掩蔽的激活，实心节点描绘了正常激活。**(C)** 对模型输出应用主成分分析 (PCA)，以可视化模型对不同锚对的表示。每个锚对采样 50 个数据点。对称锚对（例如 (c, d) 和 (d, c)）以相似颜色但不同深浅显示，以表明它们的等价性。三个阶段通过调整初始化率 (γ) 实现。**(D)** 掩蔽第二个锚 (a_2) 后模型输出之间的余弦相似度矩阵。每组四个块对应于具有相同 $g(x; a_1)$ 值的输入。右侧面板提供了选定块的放大视图，显示了相应输入的 x 和 a_1 的详细值。颜色标度代表余弦相似度值。不同阶段对应于初始化率 γ 的不同值。

13.3.1 通过掩蔽关键 token 进行机制分析 (Mechanism Analysis via Key Token Masking)

基于前面描述的掩蔽策略，我们通过使用 t-SNE 可视化输出嵌入来分析模型不同阶段背后的机制。图 13.3C 显示了模型在不同条件下如何组织关键 token 和锚对的表示。对于此分析，我们为每个锚对随机采样了 50 个数据点，以评估模型区分不同映射的有效性以及它是否遵守任务特定属性（如交换性）。为了增强清晰度，我们使用相似颜色但不同深浅来表示对称锚对（例如 (c, d) 和 (d, c)），表明它们在交换运算下的潜在等价性。

图 13.3C 中观察到的三个阶段是通过调整初始化率 (γ) 实现的，该率控制参数初始化的标准差。在 **阶段 1**（图 13.3C 左面板），不同锚对的表示组织混乱，簇之间存在显著重叠。这表明模型未能捕捉到与每个锚对相关的不同映射，导致表示纠缠和泛化能力差。缺乏清晰结构反映了模型依赖记忆而非学习任务背后的组合规则。因此，模型在 ID 和 OOD 数据上均表现不佳，表明其无法在训练分布内进行泛化。

在 **阶段 2**（图 13.3C 中间和右侧面板），模型开始捕捉关键 token 和锚对之间的结构关系，表现为出现清晰且分离良好的簇。在中间面板中，模型将对称锚对（例如 (c, d) 和 (d, c)）视为独立的簇，未能识别它们的等价性。这种缺乏抽象导致尽管 ID 泛化有所改善，但交换性仍然很差。相比之下，右侧面板展示了一个更高级的模型状态，其中对称对成功合并为单个簇。这种结构上的改进不仅实现了良好的交换性并增强了泛化，而且暗示了向更简单、基于规则的方法来拟合数据的过渡。从无组织到高度结构化的簇的转变，突显了模型在后期阶段倾向于采用更低复杂度和系统性解决方案的趋势。

值得注意的是，阶段 3 是具有良好交换性的解的一种特殊情况。因此，仅从复合锚的角度分析具有良好交换性的阶段 2 和阶段 3 之间的差异是不可行的。在下一小节中，我们将着重于在单锚层面上分析这些差异，以揭示模型学习任务的潜在机制。

13.3.2 通过掩蔽第二个锚进行机制分析 (Mechanism Analysis via Second Anchor Masking)

为了研究单锚操作背后的机制及其与公式 (13.2) 中定义的单锚函数 $g(\cdot;\cdot)$ 的一致性，我们通过掩蔽第二个锚的信息来分析关键 token 与单个锚在输出空间中的分布模式，如图 13.3B 所述。在图 13.3D 中，颜色表示被掩蔽模型（隔离了关键 token 和第一个锚的贡献）输出之间的成对余弦相似度。每组四个块对应于产生相同 $g(x; a_1)$ 值的输入，右侧放大面板提供了这些选定块的 x 和 a_1 的详细输入值。

在图 13.3D 的左面板（对应**阶段 2**）中，具有相同 $g(x; a_1)$ 值的输出嵌入表现出低相关性，如稀疏的红色对角线和普遍较低的余弦相似度所示。这一观察结果表明，即使模型捕捉到了对称关系，它仍然依赖于记忆复合映射来拟合数据。相比之下，中间面板代表 **阶段 3**，显示具有

相同 $g(x; a_1)$ 值的输入之间存在显著更高的相关性，如对角块中更突出的红色区域所示。这表明模型是逐步计算单锚映射的，遵循定义的函数 $g(\cdot;\cdot)$ 来产生最终输出。右侧的放大视图突显了每个块内跨输入值的一致余弦相似度模式，进一步证实了阶段 3 中的分步推理机制。

这些结果说明了两个阶段之间的根本区别：阶段 2 依赖于记忆复合映射，而阶段 3 通过逐步计算单锚映射展示了一种基于推理的策略。这种转变凸显了模型在阶段 3 中利用组合规则泛化 OOD 数据的能力，而不是依赖记忆，并表明模型向更简单、更结构化的数据拟合策略过渡。

13.4 模型复杂度：相变的关键因素 (Model Complexity: A Key Factor in Phase Transitions)

基于前一节对模型机制的分析，我们假设模型的 **复杂度偏好（complexity preferences）** 在驱动阶段转变中起着关键作用。在本节中，我们从两个关键角度考察这一现象：输入权重的 **凝聚（condensation）** 和词嵌入矩阵的 **结构化组织（structured organization）**。

凝聚发生在神经元的输入权重聚集在少数几个孤立方向上时，有效降低了模型的复杂度，并近似为一个较小规模的网络。然而，像词嵌入矩阵这样的参数，需要编码不同输入 token 之间的区别，会抵抗凝聚，并可能形成系统性的、基于规则的结构。我们通过这两个方面来考察模型在不同阶段的复杂性。

13.4.1 输入权重的凝聚 (Condensation of Input Weights)

为了研究参数凝聚现象，我们分析了第一层查询权重矩阵 ($W^{Q(1)}$) 中神经元输入权重之间的余弦相似度。第 i 个和第 j 个神经元之间的相似度计算为：

$$\frac{W^{Q(1)}[i,:] \cdot W^{Q(1)}[j,:]}{\|W^{Q(1)}[i,:]\|_2 \|W^{Q(1)}[j,:]\|_2},$$

结果在图 13.4A 中为每个阶段进行了可视化。

在 **阶段 1**，输入权重没有表现出显著的凝聚，神经元之间的余弦相似度较低。这表明神经元在权重空间中独立取向，对应于高复杂度状态。在 **阶段 2**，我们观察到轻微的凝聚，神经元开始沿着有限数量的方向聚集，导致相似度略有增加。最后，在 **阶段 3**，输入权重强烈地聚集到少数几个孤立的方向上，形成了具有高组内相似度的不同神经元群组。这种明显的凝聚反映了复杂性的显著降低，使模型能够近似一个较低维的结构，同时保持任务性能。

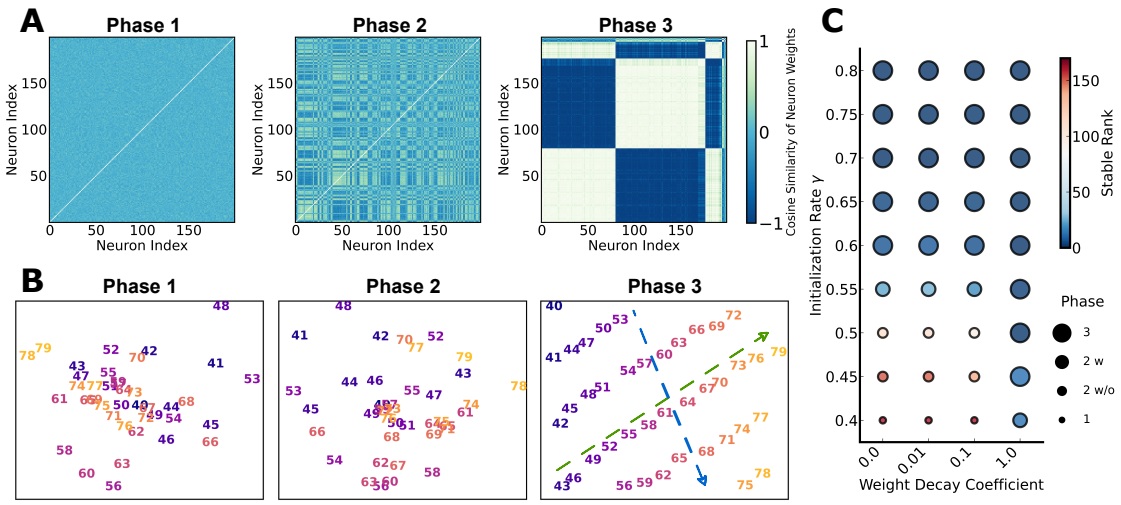


图 13.4: **(A)** 每个阶段中第一层查询权重矩阵 ($W^{Q(1)}$) 中神经元输入权重之间的余弦相似度。横坐标和纵坐标均代表神经元索引。矩阵是在权重衰减系数固定为 0.01，初始化率 (γ) 分别设置为 0.2、0.5 和 0.8 以对应阶段 1、阶段 2 和阶段 3 的设置下计算的。**(B)** 与 (A) 相同设置下词嵌入向量的 PCA 可视化。每个数字对应一个特定的 token，其位置代表通过 PCA 获得的降维嵌入。**(C)** 跨初始化率 (γ) 和权重衰减系数的参数矩阵 ($W^{Q(1)}$) 稳定秩总结的气泡图。每个气泡的大小代表阶段索引（w: 具有交换性；w/o: 不具有交换性），其中阶段 3 和阶段 2 根据 OOD 准确率是否超过 50% 来区分，其他阶段边界与图 X.2 一致。每个气泡的颜色代表稳定秩的值，所有结果均为三次独立试验的平均值。

13.4.2 词嵌入矩阵的结构化组织 (Structured Organization of the Word Embedding Matrix)

为了探索模型词嵌入矩阵的结构特性，我们使用 PCA 可视化嵌入空间，如图 13.4B 所示。每个数字代表对应特定 token 的嵌入向量经 PCA 降维后的位置。

在 **阶段 1** 和 **阶段 2**，嵌入向量没有表现出任何清晰的结构，token 的位置在降维空间中不规则地散布。这种缺乏组织性表明模型没有编码 token 之间的系统性关系。然而，在 **阶段 3**，PCA 可视化揭示了一个高度结构化的模式，token 嵌入以有序和规则的方式排列。这种结构反映了模型倾向于以系统性、基于规则的方式编码关系，使其能够有效泛化，同时保持较低的复杂度。

值得注意的是，阶段 3 中词嵌入矩阵的相对顺序关系并非对应 token 的简单数值大小关系。这种顺序关系可能源于四个单锚的定义，其中任意两个单锚操作之间的差异可以通过基本元素 3 和 4 的加法获得。这种排列与数字在嵌入空间中从两个方向以 3（绿色箭头）和 4（蓝

色箭头）为间隔进行排序是一致的。

13.4.3 通过稳定秩的统一复杂度视角 (Unified Complexity Perspective via Stable Rank)

为了统一观察到的输入权重凝聚和词嵌入结构化组织现象，我们使用 **稳定秩 (stable rank)** 分析了不同阶段参数矩阵的复杂度，这是一种广泛采用的矩阵复杂度度量。稳定秩定义为：

$$R_{\text{stable}}(A) = \frac{\|A\|_F^2}{\|A\|_2^2},$$

其中 $\|A\|_F$ 表示 Frobenius 范数，代表矩阵的总能量，$\|A\|_2$ 表示谱范数，捕捉矩阵的最大奇异值。该比率量化了矩阵的 “有效维度”，提供了一个归一化的复杂度度量。

在图 13.4C 中，我们分析了图 13.2 中介绍的多头 GPT-2 模型的第一层查询权重矩阵。来自所有头的权重被合并为每个配置的一个方阵。水平轴和垂直轴分别代表权重衰减系数和初始化率 (γ)。每个气泡的大小表示给定配置下的阶段索引，其中阶段 3 和阶段 2 根据 OOD 准确率是否超过 50% 来区分，其他阶段边界与图 13.2 一致。每个气泡的颜色反映了该配置下参数矩阵的稳定秩。每个结果由三个随机种子平均得到。

随着模型经历阶段转变，稳定秩随着阶段指数的增加而逐渐降低，表明随着推理偏好变得更加明显，模型复杂度降低。在同一阶段内，稳定秩的值在不同设置下保持相对一致，进一步强调了稳定秩与模型阶段特定复杂度特征之间的一致性。

模型复杂度、内部机制和推理能力之间的相互作用在塑造 Transformer 的组合泛化能力方面至关重要。复杂度控制——通过调整初始化率和权重衰减系数实现——塑造了模型的参数结构，导致神经元输入权重凝聚和结构化的词嵌入矩阵。神经元凝聚简化了输入表示，有助于识别基本模式，而结构化嵌入则使 token 表示与底层的组合规则对齐。这些内部的改进通过使模型能够专注于核心组合基元而不是记忆大量的输入-输出映射，从而增强了模型的推理能力。

13.5 在现实任务上的进一步验证 (Further Verification on Realistic Tasks)

我们在系列组合和推理任务上验证了具有不同初始化尺度和权重衰减设置的模型的性能。下面，我们介绍每个任务及其相应结果。

组合扩散任务：概念图 (Concept Graphs)

Concept Graphs 数据集提供了一个合成且结构化的框架，旨在评估生成模型中的组合泛化。该数据集建立在三个核心概念变量——颜色、形状和大小——的基础上，每个变量都有特

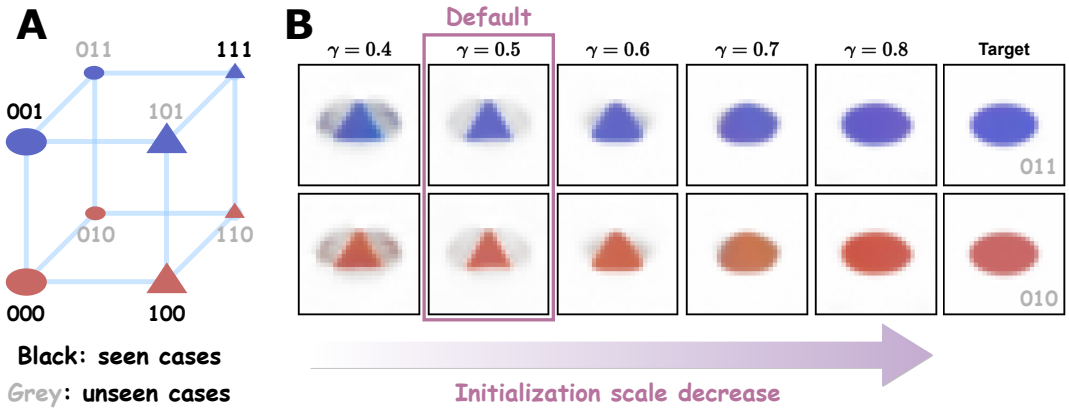


图 13.5: **(A)** Concept Graphs 数据集的结构。该数据集基于三个核心概念变量——颜色、形状和大小——组织成一个立方体框架，每个变量都有特定的值（例如，颜色为红色或蓝色，形状为圆形或三角形，大小为大或小）。黑色节点代表用于识别有趣失败模式的稀疏训练数据子集。**(B)** 不同初始化率对案例 011 和 010 的组合泛化影响。随着初始化率增加（对应于初始化尺度减小），模型从表现出有趣的失败模式过渡到实现稳健的组合泛化。紫色框突出显示了先前工作中使用的默认初始化设置。

定值（例如，颜色为红色或蓝色，形状为圆形或三角形，大小为大或小），以立方体的形式组织了不同概念组合之间的关系，如图 13.5A 所示。先前的工作发现了一个显著的组合泛化失败案例，他们称之为“**有趣的失败模式 (interesting failure mode)**”。当模型在数据的稀疏子集上训练时（具体为图 13.5A 中的黑色节点：000, 100, 001 和 111），会出现这种失败。当在未见元组（如 010，代表小红色圆圈）上测试时，模型错误地生成了一个小三角形而不是预期的小圆圈。这表明模型在这些条件下无法实现组合泛化。

基于我们对初始化对模型学习组合任务能力影响的分析，我们使用不同的初始化尺度训练了用于此任务的模型。我们发现，当初始化尺度较小时，“有趣的失败模式”消失了，模型实现了稳健的组合泛化。如图 13.5B 所示，我们从左到右逐渐增加初始化率（减小初始化尺度）。模型对两个未见概念元组（010 和 011）的输出逐渐从三角形过渡到圆形，与目标形状一致。紫色框突出显示了先前工作中使用的初始化设置，称为默认初始化。值得注意的是，为了精确复现先前工作中在 $\gamma = 0.5$（默认设置）下观察到的行为，我们在此实验中使用了均匀分布进行参数初始化 $\boldsymbol{\theta} \sim \mathcal{U}(-1/d_{\text{in}}^{\gamma}, 1/d_{\text{in}}^{\gamma})$（PyTorch 默认的 `kaiming_uniform` 分布，这也被“先前工作”采用）。当使用正态分布进行初始化时，也观察到了一致的结果。

组合任务：SCAN 和 COGS

SCAN 和 COGS 是经典的、具有更自然语言变化的组合任务。对于 SCAN 数据集，我们选择了“跨基本命令泛化组合 (Generalizing composition across primitive commands)”任务，

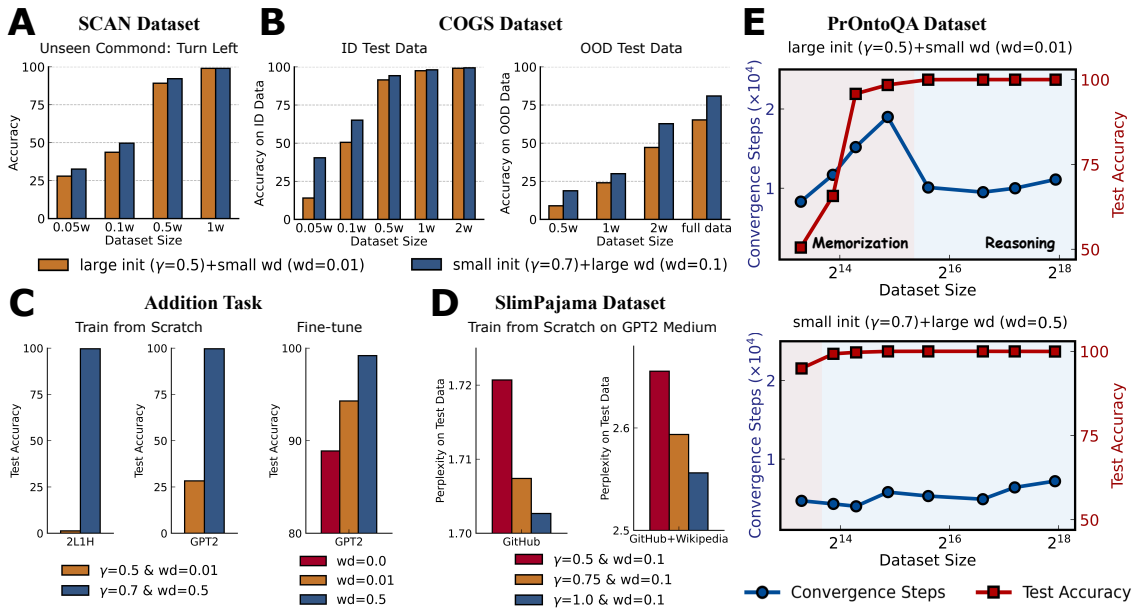


图 13.6: **(A, B)** 不同初始化尺度和权重衰减系数模型在组合任务上的性能比较。**(A)** 对于 SCAN 任务，我们评估包含 “turn left” 命令的复合命令的泛化能力。**(B)** 对于 COGS 任务，我们在相同数据集上训练后评估 (左面板) ID 和 (右面板) OOD 泛化。小初始化和大权重衰减（蓝色）在不同任务和数据规模上始终优于大初始化和小权重衰减（橙色）。参数遵循标准差为 $d_{in}^{-\gamma}$ 的零均值正态分布进行初始化。**(C)** 不同初始化尺度和权重衰减系数模型在加法任务上的性能比较。我们使用基于案例的推理干预实验，测试集为 $a, b \in [400, 600]$，其余数据作为训练集。对于从头训练部分，参数遵循标准差为 $d_{in}^{-\gamma}$ 的零均值正态分布进行初始化。对于微调部分，我们使用 Hugging Face 提供的 GPT-2 模型预训练权重作为微调起点。**(D)** 不同初始化尺度和权重衰减系数模型在 SlimPajama 数据集上的性能比较。对于 SlimPajama 数据集，在 GitHub 和 GitHub+Wikipedia 数据上训练的 GPT-2 Medium 模型，使用小初始化尺度始终获得较低的困惑度。参数遵循标准差为 $d_{in}^{-\gamma}$ 的零均值正态分布进行初始化。**(E)** 不同初始化尺度和权重衰减系数模型在 PrOntoQA 上的性能比较。上面板：大初始化 ($\gamma = 0.5$) 和小权重衰减 (WD $= 0.01$) 的收敛步数和测试准确率。下面板：小初始化 ($\gamma = 0.7$) 和大权重衰减 (WD $= 0.1$) 的收敛步数和测试准确率。参数遵循标准差为 $d_{in}^{-\gamma}$ 的零均值正态分布进行初始化。

其中 “turn left” 命令仅出现在单命令映射中，并与其他复合命令一起训练。我们评估模型在包含 “turn left” 命令的复合命令上的泛化能力。对于 COGS 数据集，我们在同一集合上训练后评估 ID 和 OOD 泛化。ID 测试使用具有相同组合模式但不同基元的数据，而 OOD 测试使用遵循不同组合规则的数据。

如图 13.6A, 13.6B 所示，我们展示了具有不同初始化尺度和权重衰减系数的模型在各种数据规模下的泛化性能。小初始化和大权重衰减（蓝色）在不同的任务类型和数据规模上始终优于大初始化和小权重衰减（橙色）。值得注意的是，在 COGS 任务中，即使两种设置的 ID 泛化（使用 20k 训练数据）都达到 99% 以上，OOD 泛化方面的差异仍然显著。

现实任务：加法任务和 SlimPajama 数据集

与传统的加法任务不同，我们使用 **基于案例的推理干预实验 (case-based reasoning intervention experiment)** 来研究加法任务中规则学习的泛化。具体来说，我们考虑设置：$a + b = c$，其中 $a, b \in [0, 999]$。我们使用 $a, b \in [400, 600]$ 作为测试集，其余数据作为训练集。这种构建方式阻止了模型简单地模仿与测试集相似的训练数据。我们训练了一个简单的 2 层 1 头模型和一个 GPT-2 模型。如图 13.6C 所示，无论模型大小和学习模式如何，小的初始化尺度（或大的权重衰减系数）通常导致良好的规则泛化，而大的初始化尺度（或小的权重衰减系数）则未能完美泛化。

对于 SlimPajama 数据集，我们使用了两种数据组成：GitHub 部分和 GitHub+Wikipedia 部分。我们使用不同的初始化在两个数据集上训练 GPT-2 Medium 模型 40B token。如图 13.6D 所示，对于两种数据组成，较小的初始化尺度始终实现了较低的困惑度。

推理任务：PrOntoQA

PrOntoQA 是一个合成的多步推理数据集，其中每个数据点都指定了对象之间的层次关系，并要求模型确定一个多步推理链是否正确。在测试期间，我们只评估模型判断层次关系的准确性。因此，模型的随机猜测准确率为 50%。

图 13.6E 说明了具有大初始化（和小权重衰减系数，上面板）和小初始化（和大权重衰减系数，下面板）的模型关于数据规模的收敛速度和泛化误差。对于具有大初始化（小权重衰减系数）的模型观察到一个有趣的现象：随着数据量的增加，收敛速度先降低后增加。当数据量较小时，模型倾向于通过记忆来拟合数据。因此，随着数据量的增加，训练难度增加（即训练速度减慢），模型的泛化能力较差。随着数据量进一步增长，受其复杂度限制的模型无法再记住所有数据，因此转向通过推理来拟合数据。这导致拟合速度加快，并产生更好的泛化。相比之下，具有小初始化（大权重衰减系数）的模型天生更倾向于通过推理来拟合数据，导致在相同数据规模下比具有大初始化（小权重衰减系数）的模型收敛更快且泛化更好。

13.6 对不同解决方案背后机制的预测 (Prediction of Mechanisms underlying Different Solutions)

随着数据复杂性的增加，神经网络通常需要更多的训练步骤来拟合数据。基于我们对不同解决方案背后机制的分析，我们可以预测具有不同初始化尺度的模型在学习不同复杂度的数据时会表现出不同程度的困难。为了创建具有不同复杂度水平的数据集，我们故意修改某些复合映射，同时确保具有对称关系的锚对保持一致（即 $f(x;a_i,a_j)=f(x;a_j,a_i)\neq g(g(x;a_i);a_j)$）。我们将 **推理复杂度 (reasoning complexity)** 定义为不满足组合规则的锚对组的数量（注：具有对称关系的两个锚对被认为属于同一组）。这种推理复杂度捕捉了数据集中违反推理规则并需要基于记忆的映射的数据的多样性。高的推理复杂度表明存在更复杂的关系和模式，简单的推理规则无法轻易捕捉。因此，模型必须更多地依赖于记忆特定的数据映射，而不是应用通用的推理原则。

可以预见，小初始化的模型（即倾向于以尽可能低的复杂度拟合数据）应该使用更多的训练步骤来拟合具有较大推理复杂度的数据，而大初始化的模型（即容易记住对称解映射的模型）应该使用相似的训练步骤来拟合具有不同推理复杂度的数据。

为了测量拟合数据所需的训练步骤，我们使用模型训练准确率达到 100% 所需的 epoch 数量。我们为每个设置进行了 9 次随机试验。

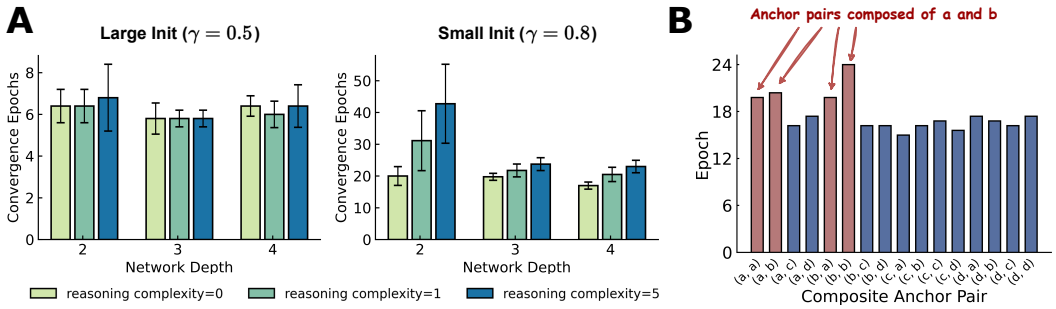


图 13.7: **(A)** 不同初始化尺度在面对不同数据复杂度时的训练速度。横坐标：模型深度。纵坐标：模型训练准确率达到 100% 所需的 epoch 数。颜色：推理复杂度。我们对不同初始化、深度和数据推理复杂度的每个设置进行 9 次随机试验，并取所需 epoch 数的平均值。左面板：大初始化（初始化率 $\gamma=0.5$）。右面板：小初始化（初始化率 $\gamma=0.8$）。**(B)** 不同锚对在其对应训练数据上达到 60% 准确率所需的 epoch 数。锚对 (a,b)、(b,a)、(a,a) 和 (b,b) 以红色突出显示，因为它们受到非推理解决方案 (a,b) 和 (b,a) 的显著影响。

如图 13.7A 所示，倾向于通过记忆解决方案学习数据的模型对具有不同推理复杂度的数据使用大致相同的训练步骤（图 13.7A 左面板），而倾向于通过推理解决方案学习数据的模型

确实对具有较大推理复杂度的数据使用更多的训练步骤（图 13.7A 右面板）。这一预测有力地支持了我们关于不同初始化尺度为何会导致不同解决方案的分析。

我们进一步展示了当推理复杂度为 1 时，不同锚对的拟合速度。在这个实验中，我们同时将锚对 (a, b) 和 (b, a) 设置为违反组合规则的相同映射。如图 13.7B 所示，我们展示了 16 种锚对在其对应训练数据上达到 60% 准确率所需的 epoch 数。我们将四种锚对 (a, b)、(b, a)、(a, a) 和 (b, b) 用红色突出显示（因为 (a, b) 和 (b, a) 违反了组合规则，上述四种锚对会受到显著影响）。可以观察到，突出显示的锚对类型表现出明显较慢的训练速度，进一步验证了违反组合规则的映射的存在会降低小初始化模型的训练速度。

从另一个角度来看，这也使我们能够发现数据集中违反组合规则的映射（通常是噪声数据）。对于一组分布未知的数据，我们可以通过使用小初始化模型研究不同类型数据的收敛速度来进行一定程度的区分。

13.7 习题

1. 参数初始值的尺度对模型训练过程中学习推理解和记忆解的影响分别是什么？如何通过实验验证这种影响？

2. 为什么较小的初始化尺度更容易导致模型学习推理解，而较大的初始化尺度则更容易导致模型学习记忆解？请详细解释背后的机制。

3. 对于一个两层神经网络 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k \cdot \boldsymbol{x})$，如果我们希望该网络能够更好地泛化到分布外（OOD）任务上，应该如何选择参数初始值的尺度？请解释理由。

4. 在训练过程中，如何定量地衡量模型参数的凝聚程度？这种凝聚现象对模型学习推理解和记忆解有什么影响？

5. 参数凝聚现象与词嵌入矩阵的规律性之间存在什么相关性？这两种现象均通过何种方式促进模型学习数据中的规律？还有什么参数表现形式同样有利于模型学习到推断解吗？

Chapter 14

深度神经网络的一些重要现象

14.1 引入

在本书前面的章节中，我们已经重点介绍了频率原则和凝聚等现象。本节将带您探索深度学习中的另外一系列令人着迷的现象。这些现象不仅展示了深度学习的复杂性和潜力，也挑战了我们对学习、智能和优化的传统理解。

科普篇

为什么要关注神经网络的现象？

现象驱动正在成为深度学习中越来越重要的研究范式。其背后的根本原因在于深度神经网络做为一个高度非线性强关联的高维复杂系统，很难只依靠演绎，从其数学形式和动力学方程出发推导出宏观层面的规律。科学发展的历史告诉我们，面对这样复杂的系统，通过实验对其现象进行系统地观察和归纳常常是建立理解的第一步也是关键的一步。从科学史的角度来看，当前深度学习的理论研究仍处于前牛顿时代（斯坦福李飞飞）。在这样一个阶段，系统的实验研究和现象归纳是极为关键的工作。这样的工作也将构成我们迈入牛顿时代的基石。

为了给大家一个直观的理解，我们可以做一个思想实验。假如在热力学诞生前，我们就已经获知理想气体微观的动力学方程，也就是大量的分子在空气中进行无规则运动，遵循牛顿运动定律，以弹性方式相碰撞。当考虑阿伏伽德罗常量（$\sim 6.02 \times 10^{23}$）量级的分子在一个容器中的运动时，我们如何才能从 $O(10^{23})$ 个微分方程中演绎出理想气体的状态方程 $PV = nRT$ 呢？如何才能从这样确定性的方程中演绎出熵增原理呢？我们很难想象这种数学演绎可以在没有以下实验现象指引的情况下发生：温度、压强、体积是重要的宏观量且三者之间有明确的依

赖关系；尽管动力学方程是可逆的，但是实验中限定在半空间中的分子会迅速布满全空间，而充满全空间的分子却几乎不可能再聚集到半空间中。

什么样的现象值得关注？

泛化之谜 在传统机器学习理论中过参数的模型容易发生过拟合现象，但这个现象在神经网络中似乎并不存在，这就是神经网络的泛化之谜。

稳定边缘现象 神经网络的优化过程并不总是平稳，而是处于临界状态，这个状态与学习率有关，学习率太大会导致训练动力学不稳定。

平坦解与尖锐解 平坦解与尖锐解的研究探讨了神经网络优化中不同类型局部最小值的特性及其对模型泛化能力的影响。平坦解通常与更好的泛化性能相关联，而尖锐解则可能导致过拟合与训练不稳定。

随机梯度下降噪声结构的影响 随机梯度下降 (SGD) 的噪声结构研究表明，SGD 引入的噪声不是随机扰动，而是具有特定结构，这种结构化噪声可能有助于模型逃离尖锐局部最小值，倾向于找到更平坦的解，从而提高模型的泛化能力。

幸运彩票现象 幸运彩票现象提出大型神经网络中存在稀疏的子网络，这些子网络在初始化时就具有良好的结构，能够在训练中表现出色。

Double Descent 现象 Double descent 现象描述了在增加模型复杂度或训练数据量时，模型的测试误差可能会出现两次下降的趋势。

顿悟 (Grokking) 现象 顿悟现象指的是神经网络模型在长期训练过程中先达到训练误差很小的状态，经过一段时间后会经历一个突然的 grokking 阶段。在这个阶段，模型的性能会迅速从过拟合状态跃升到良好的泛化状态。

神经塌缩现象 神经塌缩现象指的是深度神经网络在训练后期出现的一种特殊状态，其中同类样本的特征表示会聚集在一起，而不同类别的特征表示会呈现等角分布。

Mode connectivity 现象 Mode connectivity 研究发现，神经网络优化景观中的不同局部最优解通常是相互连接的。

大语言模型上下文学习 上下文内学习是大型语言模型的一个重要特性，它使模型能够仅通过给定的任务描述和少量示例，就能快速适应并执行新的任务，无需额外调整模型权重进行学习。

大语言模型思维链 思维链是指通过引导模型生成中间推理步骤，而不是直接给出问题答案的方法，这种方法能够显著提高模型在复杂任务中的表现。

14.2 泛化之谜

深度神经网络凭借其强大的表达能力和灵活的学习能力, 在各类机器学习任务上取得了令人瞩目的成就。然而, 尽管深度神经网络在实践中展现出了惊人的性能, 但对其泛化能力的理论理解仍然非常有限。

早在 1995 年，Breiman 教授在他的一篇论文 (Breiman 1995) 中就敏锐地指出, 统计学习理论中的很多结论都建立复杂度上。复杂度可以非常粗糙地通过模型参数数量的反映，也可以严格地通过 VC 维度或者 Rademacher 复杂度来刻画。VC 维度是一种衡量模型复杂度的方法，它表示模型能够分类的最大样本数量。Rademacher 复杂度则衡量模型在完全打乱数据集的标签后模型的最高准确率。以线性拟合为例，当模型参数数量超过样本数量时，往往会出现多个可能的解。在这种情况下，某些参数组合可能导致模型的泛化能力显著降低。因此，在选择模型复杂度时，需要在拟合能力和泛化性能之间找到平衡点。如图 14.1所示, 传统的学习理论暗示, 如果模型的复杂度过高, 训练误差和测试误差之间的差距会变大, 导致泛化性能下降。

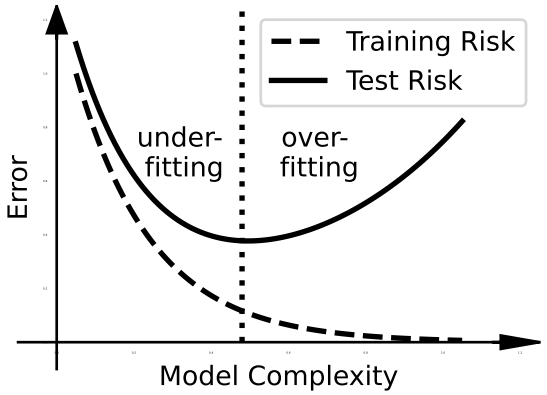


图 14.1: 传统的统计学习理论对应的的 U 形曲线，虚线表示训练误差，实线表示测试误差

但在现实世界中, 很多机器学习模型 (尤其是神经网络模型) 往往都具有远大于训练样本

数目的参数。传统理论认为这种过参数化 (Over-parameterization) 必然会导致过拟合, 使模型完全记住训练数据的特点, 从而丧失泛化到新数据的能力。然而, 实践中过参数化的神经网络不仅能轻松地把训练数据拟合得非常好, 还常常能在测试数据上取得优异的性能。这表明，深度神经网络具有某种神奇的泛化能力，使其免于过拟合的困扰。Breiman 教授由此提出了一个非常好的问题: 究竟是什么赋予了过参数化神经网络良好的泛化性? 这背后的机制是什么? 这个问题后来被形象地称为 “泛化之谜”(Generalization Puzzle)。

尽管泛化之谜早在 1995 年就被提出, 但由于当时深度神经网络还未进入大规模实用阶段, 这个问题并未引起太多关注。直到近年来, 随着深度神经网络在各领域大放异彩, 这个谜团重新引起了人们的兴趣。2017 年, 张驰原等人在 ICLR 的论文 (Zhang et al. 2017), 系统地实证了泛化之谜的存在, 引发了学界的广泛讨论, 并最终荣获当年的最佳论文奖。

Zhang et al. (2017) 采用了一种称为 “随机标签” (random labels) 的技巧。具体而言, 他们用完全随机的标签替换了原始训练集的真实标签, 然后用这些随机标签去训练标准的神经网络。多个常用的神经网络架构都能轻松地把随机标签的训练集学得非常好, 训练误差持续下降直到接近于零。网络的这种强悍拟合能力意味着其模型复杂度非常高，足以 “暴力记忆” 海量的训练样本。然而，这样高复杂度的模型对实际问题的泛化能力仍然非常高。以一个名为 Inception 的网络为例，它有超过 160 万的参数量，当把它用来分类具有 6 万张训练集和 1 万张测试集的 CIFAR10 数据集时，它可以达到将近 90% 的准确度，远比随机猜测的 10% 准确率要高。传统的优化领域，对于这种高复杂度的模型通常会加一些正则化项来寻找泛化好的解。为了控制正则化项带来的不确定性，Zhang et al. (2017) 做了多种实验，包括含有不同正则化项的训练和完全没有正则化项的训练，模型均能表现非常好的泛化能力。这些实验表明经典统计学习理论无法对深度神经网络的泛化行为做出令人满意的解释。

泛化之谜揭示了深度神经网络令人费解但又不可忽视的一面。它不是泛泛地问神经网络为何如此强大, 而是尖锐地指出参数数量与泛化性能之间存在悖论。这个悖论对深度学习的理论基础构成了严峻的挑战:

一方面, 从优化的角度看, 大量的参数赋予了模型极强的表达能力, 使其能够学习到训练数据中蕴含的高度复杂的模式。从这个角度看, 参数越多越好。另一方面, 从泛化的角度看, 过多的参数意味着模型极易过拟合, 丧失学到的模式在新数据上的适用性。从这个角度看, 参数越少越好。

泛化之谜表明模型参数量大和模型泛化性好这两者在深度神经网络身上并不矛盾, 而是以一种微妙的方式达成了平衡。理解并阐明这种平衡机制, 对发展深度学习的理论基础至关重要。它要求我们跳出传统理论的窠臼, 从全新的视角来理解深度学习的内在规律。领域内已经有一系列的理论工作对参数化谜团做了深入的探索。本书中着重讲解的频率原则和凝聚现象说明神经网络在训练过程倾向于用低复杂度的函数拟合数据，希望以此来解释泛化之谜。

14.3 缩放定律 (Scaling law)

大模型的 Scaling Law（缩放定律）指的是随着模型规模、数据量和计算资源的增长，模型性能、训练损失和推理能力如何变化的一系列经验性规律。最早由 OpenAI 在Kaplan et al. (2020) 中提出，Scaling Law 为大模型的设计和训练提供了重要的指导。

具体来说，缩放定律提出了三大变量对于模型性能的影响：

14.3.1 模型参数规模（Model Size）

模型参数规模 N 指的是神经网络的非嵌入层参数数量，通常与模型的深度和宽度相关。研究表明，随着参数数量的增加，模型的损失函数呈现出一种幂律关系：

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N},$$

其中 N_c 是缩放定律的常数。

这种幂律关系意味着，增加模型大小会提升模型性能，在训练集足够大的前提下，参数数量每增加一倍，测试损失就会减少 2^{α_N} 倍。

然而增加模型规模带来的性能收益会受到收益递减的影响。随着模型大小的增大，参数每增加一次，测试损失的提升程度就会变小。这一观察结果表明，虽然增加模型大小是提高性能的有效方法，但它可能不是最有效的方法，尤其是在计算资源有限的情况下。

14.3.2 数据量（Dataset Size）

模型训练所使用的数据规模也对最终表现产生显著影响。在给定模型规模下，数据量越大，模型的损失函数也会呈现出一种幂律关系：

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D},$$

Scaling Law 表明数据量不足时，即使增加模型参数，模型性能的提升也会受限。

这种幂律关系意味着，增加训练数据集的大小会提升模型性能，数据集大小每增加一倍，测试损失就会减少 2^{α_D} 倍。

然而，类似于模型大小，增加数据集大小带来的性能收益会受到收益递减的影响。随着数据集的变大，token 数量每增加一倍，模型性能的提升也会受到限制。

14.3.3 计算量（Compute Budget）

除了模型大小和数据集大小之外，计算量是决定语言模型性能的另一个关键因素。我们以 petaflop/s-day （PF-days）为单位，限制计算量，在最佳大小的模型上训练足够大的数据集，

且 batch size 足够小（这是为了最佳使用计算量），得到 test loss 与计算量的关系:

$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}.$$

这种幂律关系意味着增加计算预算会导致性能的持续改进，预算每增加一倍，测试损失就会减少 2^{α_C} 倍。

同样地，增加计算预算带来的性能收益也会受到收益递减的影响，意味着增加计算预算虽然能提高模型性能，但在资源有限的前提下，很可能并不是最优方法。

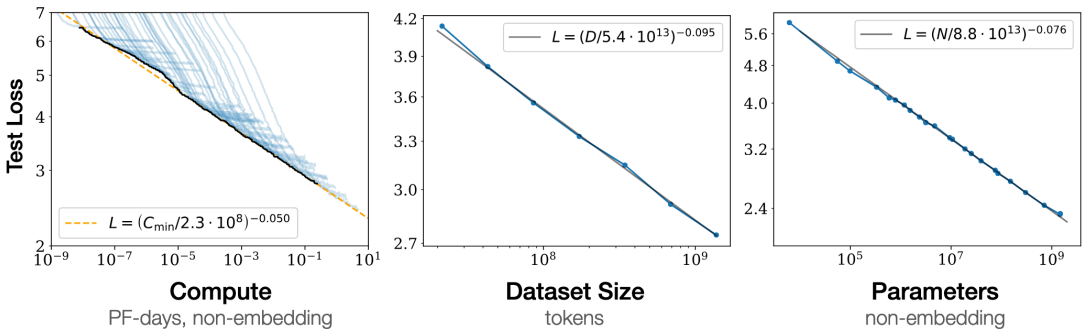


图 14.2: 左图是关于计算量的缩放定律；中图是关于数据量的缩放定律；右图是关于参数量的缩放定律。取自文献Kaplan et al. (2020) 图 1

14.4 稳定边缘现象 (Edge of Stability)

在深度学习的优化过程中，学习率 (learning rate) 是一个关键的超参数，它不仅影响训练的速度，还影响训练的稳定性。为了理解学习率与稳定性的关系，让我们首先考虑一个简单的二次函数:

$$f(x) = \frac{1}{2}\lambda x^2 \tag{14.1}$$

其中 $\lambda > 0$ 是一个常数。如果我们使用梯度下降法 (Gradient Descent, GD) 来优化这个函数，更新规则为:

$$x_{t+1} = x_t - \eta \nabla f(x_t) = (1 - \eta\lambda)x_t \tag{14.2}$$

其中 η 是学习率。为了保证收敛性，我们需要 $|1 - \eta\lambda| < 1$，即:

$$0 < \eta < \frac{2}{\lambda} \tag{14.3}$$

这个不等式给出了 GD 的稳定性条件：学习率不能超过 $2/\lambda$，否则优化过程将发散。

在高维情况下，我们可以用 Hessian 矩阵的最大特征值来刻画函数的 “锐度” (sharpness)，记为 $\lambda_{\max}$，这个量主要用于刻画神经网络局部极小点附近损失景观的平坦程度。类似地，GD 的稳定性条件变为：

$$0 < \eta < \frac{2}{\lambda_{\max}}. \tag{14.4}$$

如果学习率超过这个阈值，优化过程就会变得不稳定，表现出剧烈的震荡。在神经网络的训练中，锐度经常会保持在边缘 $2/\eta$，这种现象被称为 “稳定边缘”(Edge of Stability, EoS)。

吴磊、马超、鄂维南在 2018 年的一项研究 (Wu et al. 2018) 中，首次系统地探讨了深度神经网络训练中的稳定边缘现象。表14.1是文中对不同数据集在不同学习率训练末期锐度的平均值和标准差的实验结果。每个实验重复 5 次, 每次使用独立的随机初始化。表中第四行显示了根据理论预测的最大可能锐度。他们观察到神经网络在训练末期，大部分情况下，学习率和 $2/\lambda_{\max}$ 非常接近，这说明此时优化过程已经处于稳定边缘。

η	0.01	0.05	0.1	0.5	1	5
FashionMNIST	53.5 ± 4.3	39.3 ± 0.5	19.6 ± 0.15	3.9 ± 0.0	1.9 ± 0.0	0.4 ± 0.0
CIFAR10	198.9 ± 0.6	39.8 ± 0.2	19.8 ± 0.1	3.6 ± 0.4	-	-
prediction $2/\eta$	200	40	20	4	2	0.4

表 14.1: 不同学习率下 GD 算法得到的解的锐度，表格来源 Wu et al. (2018)

Cohen 等人在 2020 年的一项研究 (Cohen et al. 2020) 中，进一步揭示了稳定边缘现象的普遍性。他们在多个数据集和网络架构上进行了实验，结果如图14.3所示，横轴为训练步数，纵轴为锐度，虚线标记了 $2/\eta$。在三种不同的架构上, 可以观察到锐度上升到 $2/\eta$(用相应颜色的水平虚线标记), 然后在这个值附近波动, 或者略高于这个值。

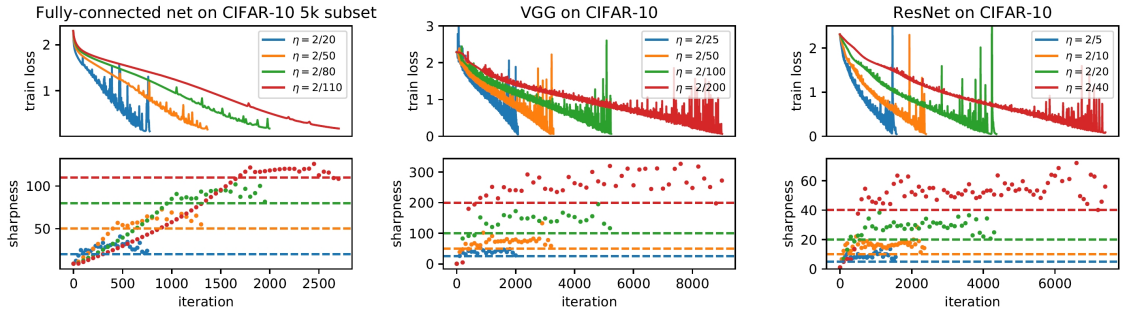


图 14.3: 梯度下降通常发生在稳定性的边缘。取自文献Cohen et al. (2020) 图 1

表 14.2: 在表列出的 6 个网络上, 使用小批量和大批量变体的 ADAM 的性能。来源Keskar et al. (2016)。

网络结构	训练精度		测试精度	
	小批量	大批量	小批量	大批量
F1	99.66% ± 0.05%	99.92% ± 0.01%	98.03% ± 0.07%	97.81% ± 0.07%
F2	99.99% ± 0.03%	98.35% ± 2.08%	64.02% ± 0.2%	59.45% ± 1.05%
C1	99.89% ± 0.02%	99.66% ± 0.2%	80.04% ± 0.12%	77.26% ± 0.42%
C2	99.99% ± 0.04%	99.99% ± 0.01%	89.24% ± 0.12%	87.26% ± 0.07%
C3	99.56% ± 0.44%	99.88% ± 0.30%	49.58% ± 0.39%	46.45% ± 0.43%
C4	99.10% ± 1.23%	99.57% ± 1.84%	63.08% ± 0.5%	57.81% ± 0.17%

稳定边缘现象对深度学习理论和实践都有重要的启示。首先，它表明深度神经网络的优化过程并非总是平稳的，而是经常处于一种动力学临界状态。这种状态既不像传统的凸优化那样单调收敛，也不会完全发散，而是表现出介于二者之间的复杂动力学。其次，稳定边缘现象提醒我们，在选择学习率时要格外小心。一方面，学习率不能太小，否则训练速度会很慢；另一方面，学习率也不能太大，否则会导致不稳定。要想在二者之间取得平衡，就需要对问题的结构有更深入的理解，并根据具体情况进行调参。稳定边缘现象是一个被清晰刻画的现象，揭示了深度学习优化过程的复杂性，是一个值得进一步研究的重要课题。它不仅有助于我们更好地理解深度学习的本质，也有可能为改进优化算法、提高训练效率提供了新的思路。

14.5 平坦解与尖锐解

在训练深度神经网络时, 人们发现使用的随机下降算法中用的批量大小（batch size）不同, 模型的泛化性能存在显著差异。使用较小批量时, 模型往往具有较好的泛化性能, 能够在测试数据上获得较高的准确率; 而当使用较大批量时, 模型在相同的测试数据上的泛化性能则会显著下降。如表14.2中，列出了六种网络结构的结果，F1 和 F2 是全连接网络，剩下的四个是卷积网络。用小批量（批量大小是 256）明显比用大批量（全部数据量的 10%）的泛化性能要好。

令人费解的是, 无论使用大批量还是小批量, 模型在训练集上的性能都很好, 训练损失函数的值相差无几，但在测试集的性能上有明显的差别，Keskar et al. (2016) 对这一现象进行了深入研究和分析。

Keskar et al. (2016) 发现, 批量大的训练方法往往收敛到损失函数的尖锐极小值点, 而小批量训练方法则更有可能收敛到平坦的极小值点。如图14.4，展示了 VGG 网络Simonyan

& Zisserman (2014) 在 CIFAR100Krizhevsky et al. (2010) 的结果。在参数空间，考虑 $\theta = \alpha\theta_L + (1-\alpha)\theta_S$ 对应的损失值和准确率，其中 θ_L 和 θ_S 分别是大批量和小批量的结果。从图中可以看出，小批量的解的邻域看起来更平坦，而大批量的解的邻域明显更尖锐。

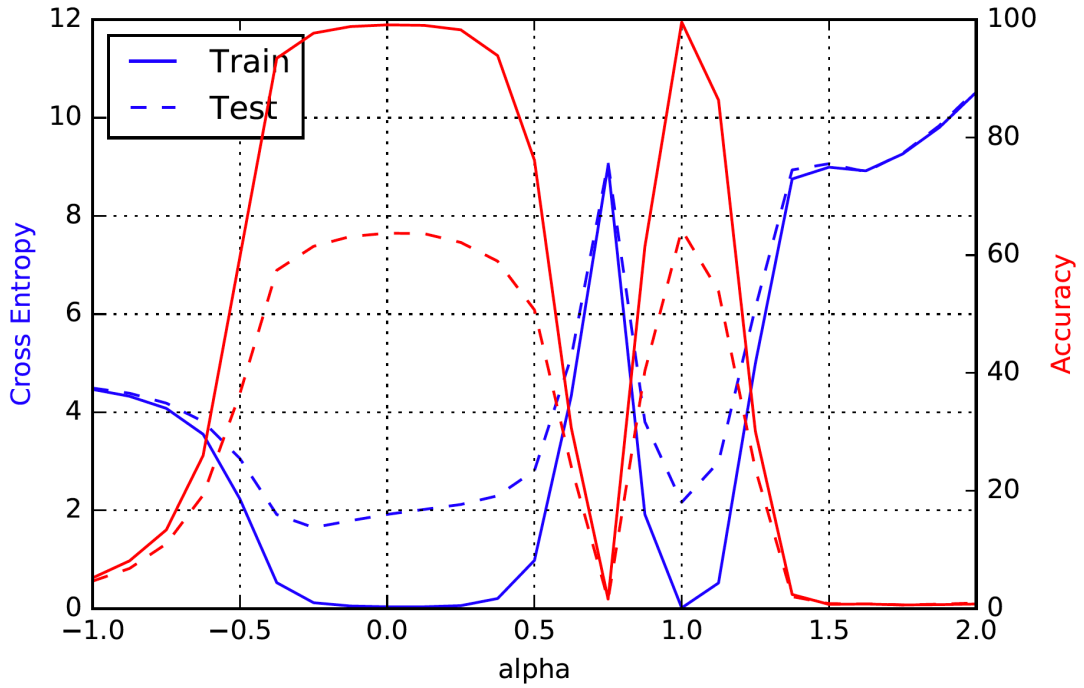


图 14.4: 一维参数曲线图 (左垂直轴对应交叉熵损失, 右垂直轴对应分类精度; 实线表示训练数据集, 虚线表示测试数据集); $\alpha = 0$ 对应小批量极小值解, $\alpha = 1$ 对应大批量极小值解。图片来源Keskar et al. (2016)。

解的平坦性启发了一种理解模型泛化能力的方式。具体来说, 如图14.5所示，当我们只是找到解的附近，比如数据量不足、受到噪音影响或者训练不充分等，此时，对于平坦极小值点，损失函数在该点附近变化较缓, 即函数曲面较平坦，因此，平坦解的泛化能力仍然很好。但对于尖锐极小值点，损失函数在该点附近变化剧烈, 即函数曲面较陡峭，尖锐解对参数的微小变化非常敏感, 因此泛化性能较差。

如何表征高维空间中解的平坦性并不是一件容易的事。有一些量是比较常用的，比如海森（Hessian）矩阵的最大特征值或者迹（trace），或者考虑在固定参数变化范围内，损失函数的变化幅度。

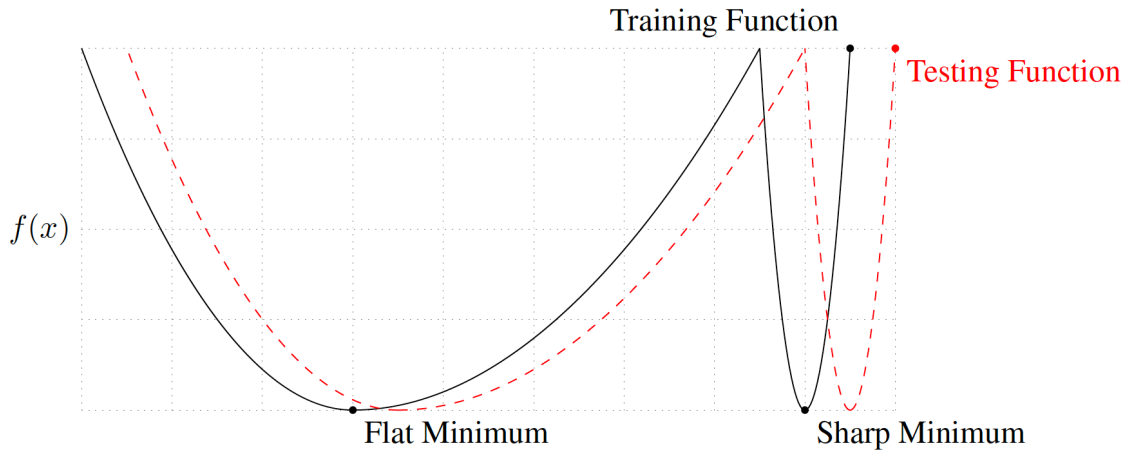


图 14.5: 平坦解与尖锐解对泛化能力影响的概念草图。图片来源Keskar et al. (2016)

14.6 随机梯度下降的噪音结构对深度学习泛化性能的影响

14.6.1 随机梯度下降噪音结构的重要性

在深度学习模型的优化过程中, 随机梯度下降 (Stochastic Gradient Descent, SGD) 是一种被广泛使用的优化算法。随机梯度下降的噪音结构对模型泛化性能有着重要影响。Zhu et al. (2018) 的研究表明, 不同的随机噪音结构会导致模型性能有显著差异，随机梯度下降的噪音具有非常特殊的结构，与损失景观的 Hessian 矩阵具有强相关。

随机梯度下降的更新规则可以写为:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t) + \eta \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \Sigma_{sgd}^t) \tag{14.5}$$

其中 η 为学习率，Σ_{sgd}^t 为 SGD 噪音的协方差矩阵, 具有非各向同性的结构。而梯度 Langevin 动力学 (Gradient Langevin Dynamics, GLD) 可以看作是随机梯度下降的一个变体, 其更新规则为:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t) + \eta \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_t^2 I) \tag{14.6}$$

其中噪音项服从各向同性的高斯分布。

图14.6展示了几种不同的优化方法在 SVHN 和 CIFAR-10 数据集上的性能对比, 这些方法的噪音结构总结于表14.3。其中随机梯度下降表现出了最佳的测试精度, 而其他几种优化方法如 GD、或者含不同噪音结构的 GLD 变种测试精度显著低于随机梯度下降。这说明随机梯度下降噪音的特殊噪音结构是其泛化性能优于其他优化方法的关键。Zhu et al. (2018) 发现 Σ_{sgd}^t 和损失景观的 Hessian 矩阵 H^t 具有一定的对齐，他们的结果显示 $H^t \Sigma_{sgd}^t$ 明显大于 H^t 与一个各向同性的协方差矩阵（特征值为 Σ_{sgd}^t 的特征值的平均值）的乘积。

Dynamics	Noise ϵ_t	Remarks
GLD constant	$\epsilon_t \sim \mathcal{N}(0, \varrho_t^2 I)$	ϱ_t is a tunable constant.
GLD dynamic	$\epsilon_t \sim \mathcal{N}(0, \sigma_t^2 I)$	σ_t shares the same magnitude with Σ_{sgd}^t.
GLD diagonal	$\epsilon_t \sim \mathcal{N}(0, \text{diag}(\Sigma_{sgd}^t))$	$\text{diag}(\Sigma_{sgd}^t)$ is the diagonal of Σ_{sgd}^t

表 14.3: 不同优化方法的噪音结构。表格来源Zhu et al. (2018)

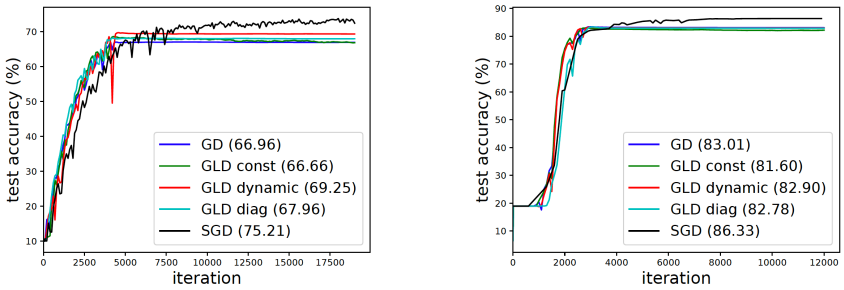


图 14.6: 不同优化动力学方法在 SVHN(左) 和 CIFAR-10(右) 数据集上的性能对比。图片来源于Zhu et al. (2018)

14.6.2 随机梯度下降噪音与损失函数平坦度的关系

为何随机梯度下降的噪音结构如此特殊? Feng & Tu (2021) 进一步研究发现, 随机梯度下降噪音的方差与损失函数的曲率呈现一种反比关系。他们考虑在 $\boldsymbol{\theta}_t$ 处，损失函数在一个固定方向 $\boldsymbol{p}_i$ 的变化，定义

$$L_i(\delta\boldsymbol{\theta}) = L(\boldsymbol{\theta}_t + \delta\boldsymbol{\theta}\boldsymbol{p}_i). \tag{14.7}$$

他们定义了一个描述损失函数景观平坦程度的量 F_i:

$$F_i \equiv \theta_i^r - \theta_i^l, \tag{14.8}$$

其中 $\theta_i^l < 0$ 和 $\theta_i^r > 0$ 分别为损失函数 L_i 沿 $\boldsymbol{p}_i$ 方向上, 左右两侧最近的满足 $L_i(\theta_i^l) = L_i(\theta_i^r) = e \times L_0$ 的点, 其中 e 为自然对数的底, L_0 为损失函数在极小值 $\boldsymbol{\theta}_t$ 处的值。F_i 的值越大, 意味着损失函数在 $\boldsymbol{p}_i$ 方向上越平坦。

取 $\boldsymbol{p}_i$ 为 Σ_{sgd}^t 的第 i 个特征值对应的特征方向，σ_i^2 是刻画噪音在 $\boldsymbol{p}_i$ 方向的方差。如图14.7所示, 该实验在一个简单的全连接神经网络上进行, 该网络有两个隐藏层, 每层包含 50 个神经元, 用于使用 MNIST 数据库进行分类任务。B 是批次大小，α 是学习率，随机梯度下

降噪音的方差 σ_i^2 与损失函数的平坦度 F_i 呈现出一种幂律的反比关系:

$$\sigma_i^2 \sim F_i^{-\psi} \tag{14.9}$$

其中 ψ 在不同的超参数设置下均接近 4。这表明, 在损失函数较为平坦的方向上, 随机梯度下降噪音的方差较小; 而在损失函数较为陡峭的方向, 随机梯度下降噪音的方差则较大。这种随机梯度下降噪音与损失函数曲率的自适应耦合, 使得随机梯度下降能更快地逃离尖锐的极小值, 从而更倾向于收敛到宽阔平坦的极小值, 这可能是其泛化性能优异的原因之一。

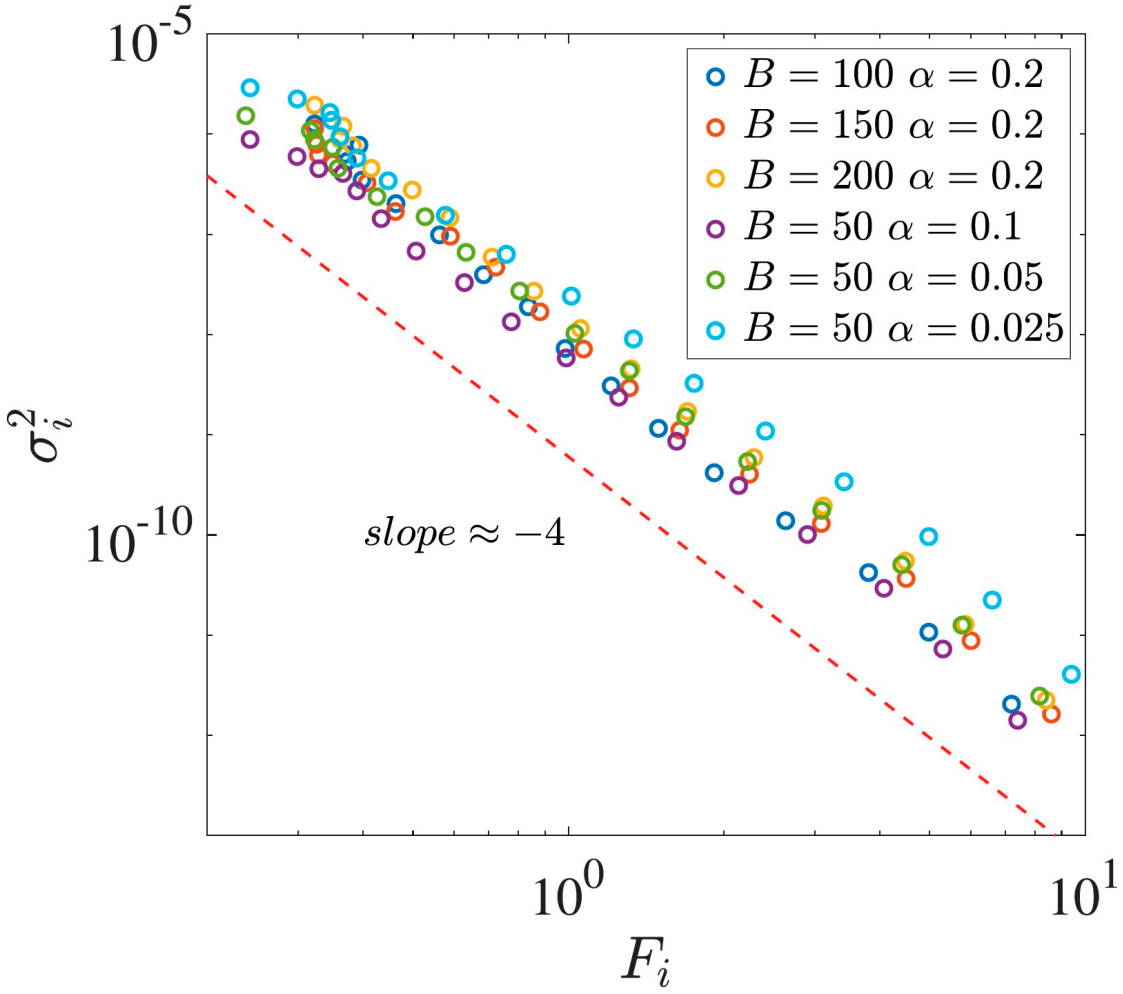


图 14.7: 随机梯度下降噪音的方差与损失函数平坦度的反比关系，图片来源Feng & Tu (2021)

14.6.3 随机梯度下降隐式正则化的理论分析

Smith et al. (2020) 从等效梯度流的角度分析，发现随机梯度下降等效于在梯度流上加了一个正则化项。假设 $L(\boldsymbol{\theta})$ 是原始的损失函数，$\nabla L(\boldsymbol{\theta})$ 为其梯度，η 为学习率，m 为每个 epoch 内的 minibatch 的数量，$\widehat{L}_i(\boldsymbol{\theta})$ 为第 i 个 minibatch 对应的损失函数。新的正则化项可表示为:

$$\frac{\eta}{4m}\sum_{i=0}^{m-1}||\nabla\widehat{L}_i(\boldsymbol{\theta})-\nabla L(\boldsymbol{\theta})||^2. \tag{14.10}$$

由于每个批次都是随机采样的，因此，$\mathbb{E}(\nabla\widehat{L}_i(\boldsymbol{\theta}))=\nabla L(\boldsymbol{\theta})$。这等价于在训练过程中，找到一个解，使得不同 minibatch 之间的梯度的方差最小。

事实上，这和我们前面提到的 Dropout 可以在一个框架下理解。若是在训练过程中随机丢掉一些部分，并用其它部分弥补，那训练过程就会使得弥补的部分和丢掉的部分尽量相同。对于 Dropout，丢掉的是神经元的输出，因此需要用同层神经元的输出进行弥补，所以训练的结果是使同层神经元的输出趋同，也就是凝聚现象。而对于随机梯度下降，丢掉的是部分样本的损失函数的梯度，用其它样本的损失函数的梯度来弥补（通过计算平均值），所以训练结果会让不同批次带来的梯度的差异尽量的小。一个平坦的解是可以做到这样的效果，因此，这个正则化项也为随机梯度下降为什么倾向于找到平坦的解提供了一个解释。

14.7 幸运彩票现象

在 2019 年，Frankle & Carbin (2018) 提出了幸运彩票猜想假设（The Lottery Ticket Hypothesis）：一个随机初始化的密集神经网络包含一个子网络，即原本网络中有一些参数被屏蔽不会使用，使得这个子网络以相同的初始化单独训练时，在相同的迭代次数内，能达到与原网络相当的测试精度。

形式化地，对于一个随机初始化的前馈网络 $f(x;\theta)$，其初始参数为 $\theta=\theta_0\sim D_\theta$。在用随机梯度下降（SGD）训练时，$f$ 在迭代第 j 次时在验证集上达到最小损失 l，此时在测试集上的精度为 a。现在对网络的参数加一个二值掩码 $m\in\{0,1\}^{|\theta|}$，即 m 以概率 p 取 1，概率 $1-p$ 取 0，得到 $f(x;m\odot\theta)$（$\odot$ 表示逐元素乘法），以相同的 θ_0 进行初始化。如果在相同的训练集上用 SGD 训练这个子网络（m 固定），则存在一个 m，使得 f 在第 $j'\le j$ 次迭代时达到验证集最小损失 $l'\le l$，且在测试集上的精度 $a'\ge a$，同时 m 中 1 的个数远小于 $|\theta|$。

实验发现，用一种迭代剪枝（iterative pruning）的方法可以很好地找到这样的子网络（图14.8中每条曲线是 5 次试验的平均值。图例中不同颜色表示剪枝后网络中剩余权重不同的比例。误差线表示任何一次试验的最小值和最大值。）。具体来说，该方法先对整个网络进行训练，然后每次去掉一部分幅度小的权重，再将保留下的权重重置为初始值，反复迭代多轮。

此时如果剪掉过多的权重，性能也会明显下降。与此同时，如果将保留的权重随机重新初始化，其性能会大大下降。这说明初始化对于中奖彩票至关重要。

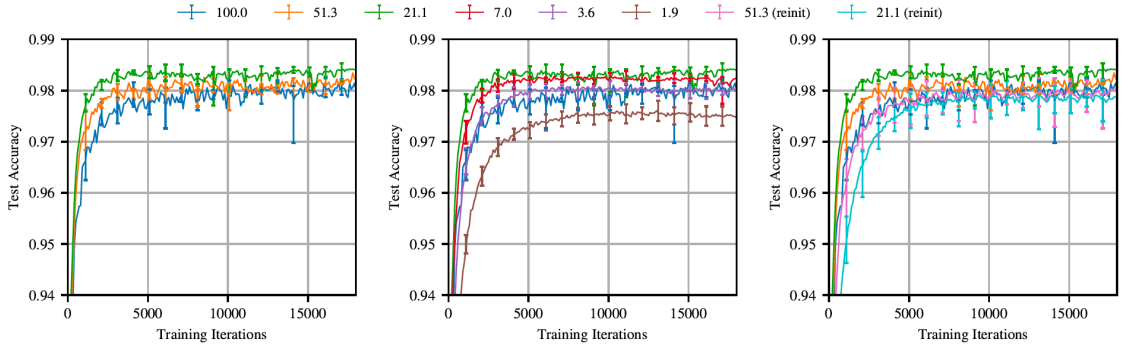


图 14.8: 在 LeNet(迭代剪枝) 中，随着训练的进行，测试精度的变化。（图片来自Frankle & Carbin (2018) 的图 3）

幸运彩票现象表明，尽管深度学习模型参数众多，但其中可能有一小部分参数就对网络性能起决定性作用。幸运彩票现象还引出了一些有趣的问题：为什么神经网络倾向于包含中奖彩票？它们是如何在随机初始化和优化过程中形成的？除了剪枝之外，是否还有更高效的方法来发现中奖彩票？这些都有待进一步的理论和实证研究。

14.8 神经网络中的 Double Descent 现象

Belkin 等人在 2019 年的一项研究 Belkin et al. (2019) 中，使用不同大小的两层神经网络在 MNIST 和 CIFAR 等数据集上进行实验，发现了明显的 double descent 现象，也就是测试误差随着模型增加，会先下降，而后上升，然后再下降。此后，这一现象在更多的神经网络实验中得到了验证。Nakkiran 等人 Nakkiran et al. (2021) 使用了更深的卷积神经网络 ResNet-18，当加入标签噪声后,double descent 现象变得非常明显，如图14.9所示。Nakkiran et al. (2021) 发现 double descent 现象关于优化步数也能被观察到。

14.9 神经网络中的顿悟 (grokking) 现象

2022 年，Power 等人在Power et al. (2022) 中发现，利用仅解码器 (Decoder-Only) 的 Transformer 训练算法数据集 (algorithmic datasets) 时，会出现明显的顿悟现象，如图14.10，即神经网络训练过程中训练误差先下降到很小，再经过一段时间后测试误差才开始减小。此后，

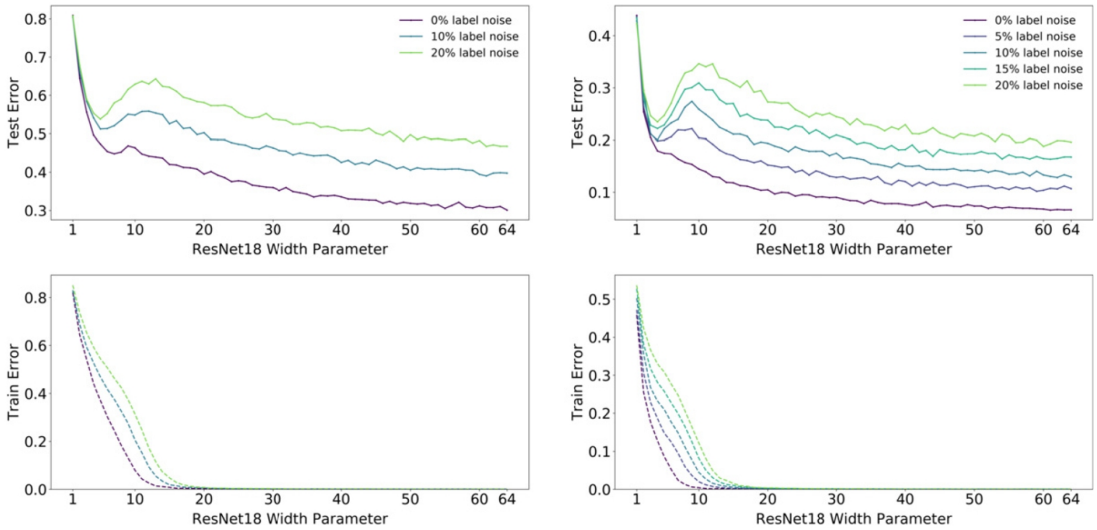


图 14.9: ResNet-18 在 CIFAR-10 和 CIFAR-100 数据集上 (有标签噪声) 的 Double Descent 现象。图片来源于Nakkiran et al. (2021)

Barak et al. (2022) 和 Bhattamishra et al. (2022) 在稀疏奇偶函数 (Sparse Parity Function) 的任务中；Liu et al. (2022) 在 MNIST 数据集、IMDb 数据集以及 QM9 数据集上均观察到了 Grokking 现象。

14.10 神经塌缩现象

2020 年，Papyan et al. (2020) 发现了一种称为神经塌缩现象（“Neural Collapse”）。他们的论文用了四种方式刻画这个现象。本节以其中的两种方式来介绍这个现象。

神经塌缩现象的一个特点是, 随着训练的进行, 同一类别内的样本在特征空间中的可变性逐渐减小并最终塌缩到类中心。形式化地, 我们可以用最后一层特征的类内协方差矩阵 Σ_W 来刻画这一现象:

$$\Sigma_W = \text{Ave}_{i,c}\left[(h_{i,c} - \mu_c)(h_{i,c} - \mu_c)^T\right] \to 0 \tag{14.11}$$

其中 $h_{i,c}$ 表示第 c 类第 i 个样本的最后一层特征，μ_c 表示第 c 类的特征均值，即类中心。Σ_W 收敛到 0 意味着所有类内样本的特征都塌缩到了类中心。

这一现象的直观解释是, 随着训练的进行, 神经网络学到了如何提取最有判别性的特征。这些特征能够很好地区分不同类别, 同时忽略了类内的细微差异。因此, 同一类别的样本在特征空间中变得越来越“紧致”, 最终塌缩成一个点。

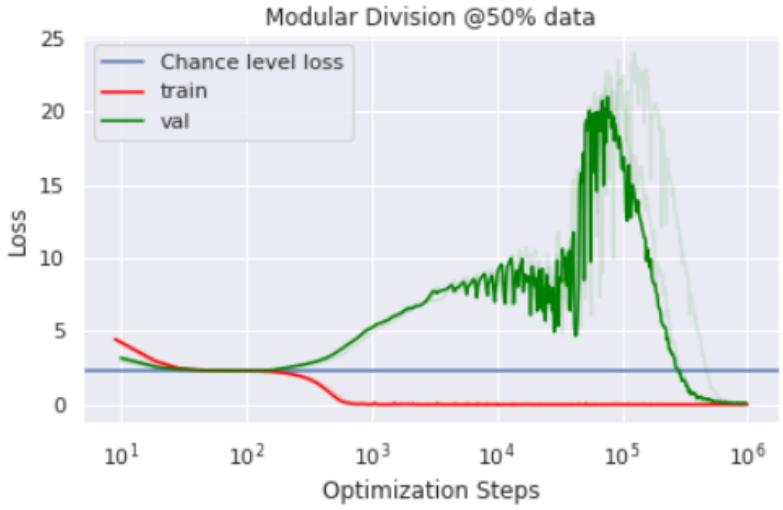


图 14.10: 除法算法数据集上的顿悟现象。图片来源于Power et al. (2022)

神经塌缩现象的另一个特点是, 最后一层特征的类中心会收敛到彼此之间的角度达到最大的点（Equiangular Tight Frame，ETF），例如正四面体的四个顶点就是在三维空间中原点出发的四个向量两两角度最大的点。

14.11 Mode connectivity 现象

在深度学习中, 如果我们针对同一个网络架构和数据集, 用不同的随机初始化训练多次, 就会得到多个不同的权重向量 $w_1, w_2, ..., w_n$, 它们都是损失函数的局部最小值。Garipov et al. (2018) 发现的 mode connectivity 指的是, 存在一条从 w_1 到 w_2 的连续路径, 路径上的每一个点 w 表示的模型在测试集上的精度都很高, 并且接近 w_1 和 w_2 所对应模型的精度。

图14.11展示了一个 mode connectivity 的例子。图中展示了 ResNet-164 在 CIFAR-100 数据集上的交叉熵损失函数的等高线图。横轴和纵轴分别对应两个不同的权重集 w_1 和 w_2, 它们都是独立训练得到的。

从图中中间和右侧的图可以看出，在一些局部极小点附近，即 w_1 和 w_2 之间存在一条弯曲的路径, 沿着路径损失函数的值基本保持不变 (用颜色表示)。这说明, 沿着这条路径从 w_1 到 w_2, 模型的精度基本不变。而 w_1 和 w_2 之间的直线路径 (图中未画出) 上的损失值要高很多。类似的 mode connectivity 现象普遍存在于其他网络架构和数据集中。

Mode connectivity 是深度学习中一个有趣的现象。它表明, 虽然神经网络的权重空间极其复杂, 充满了大量的局部最小值, 但是这些局部最小值并不是孤立的, 它们可以被简单的曲线连

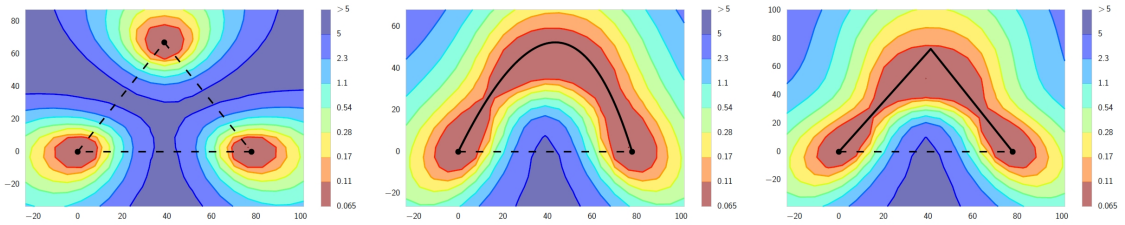


图 14.11: ResNet-164 在 CIFAR-100 上的 mode connectivity。图片来源于Garipov et al. (2018)

接起来, 曲线上的模型具有几乎一样的泛化性能。

14.12 大型语言模型的上下文内学习

在 2020 年，OpenAI 通过具有 1750 亿参数的 GPT3 发现大型语言模型展现了一种令人惊叹的能力, 叫做**上下文学习** (Brown et al. 2020)。上下文学习 (In Context Learning, ICL) 指的是语言模型在只给出几个例子的情况下, 就能够学会完成一项任务的能力, 而不需要对模型进行额外的训练或调整。在本章中, 我们将通过一些例子来直观地解释上下文学习, 并讨论一些研究结果, 展示上下文学习的强大效果。

我们先用一个例子来直观地解释上下文学习。假设我们有一个聊天机器人, 我们希望它能够学会对用户的评论进行情感分析, 即判断一条评论是正面的还是负面的。传统的做法是收集大量的评论数据, 然后让机器人在这些数据上进行训练和学习。但是使用上下文学习, 我们只需要给机器人几个例子, 它就能够学会如何进行情感分析了。

具体来说, 我们可以给机器人提供几个示例, 每个示例包括一条评论和它对应的情感 (正面或负面), 如图14.12所示。机器人通过阅读这些例子, 就可以大致学会如何判断一条评论的情感倾向。当我们给机器人一条新的评论时, 它就可以根据之前看到的例子, 来预测这条新评论的情感是正面还是负面的。

除了情感分析, 上下文学习还可以应用于许多其他任务中。比如在图片分割处理中，如图14.13，给定一段自然语言处理的任务要求，以及给一个示例，然后输入一个待处理的图片，让模型输出该图片的分割结果。

在 OpenAI 的 2020 年的文章中 (Brown et al. 2020)，通过大量实验, 研究者们发现上下文学习在各种自然语言处理任务上都取得了令人印象深刻的效果。例如, 拥有 1750 亿参数的 GPT-3 模型在只给出 32 个例子的情况下, 就在 SuperGLUE 基准测试中取得了接近最先进模型的结果。在 COPA 和 ReCoRD 数据集上,GPT-3 的上下文学习性能也达到了与当时最好的微调模型相当的水平。

从这些例子可以看出, 上下文学习允许语言模型在没有大量训练数据的情况下, 仅通过少

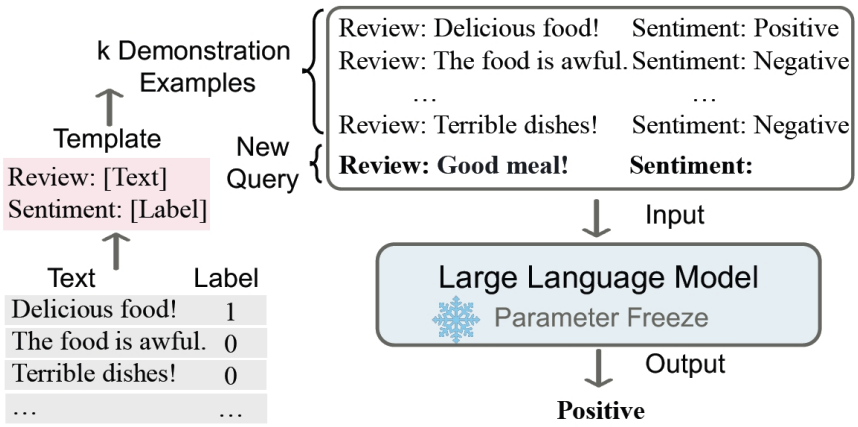


图 14.12: 上下文学习在情感分析任务中的示意图。ICL 需要一段演示上下文, 其中包含几个用自然语言模板编写的示例。将演示和查询作为输入, 大型语言模型负责进行预测。来源于Dong et al. (2022)

量示例就可以学习新的任务。这种学习方式非常类似于人类的学习方式——我们往往也是通过几个例子来学习一项新技能, 而不需要看过大量的例子。为什么会有上下文学习这个现象仍然是一个开放的问题。

14.13 大语言模型中的思维链 (Chain of Thought)

在大语言模型中，思维链 (Chain of Thought) 是一种新的提示方法 (Wei et al. 2022)，它通过让语言模型生成一系列中间推理步骤，来显著提高模型在复杂推理任务上的表现。具体来说，思维链提示是在少量样本中加入一些思维链的示范，从而诱导语言模型产生类似的推理过程。图14.14展示了一个思维链提示的例子。

实验表明，在算术、常识和符号推理等任务上，思维链提示能够大幅提升语言模型的表现。以在 GSM8K 数学应用题数据集上的结果为例，仅用 8 个思维链样本提示的 PaLM 540B 模型，就超越了微调过的 GPT-3 的准确率，刷新了该任务的最好结果。

更有趣的是，思维链推理能力是随着模型规模逐渐增大而涌现出的新能力。思维链提示在小模型上反而会降低性能，只有在足够大的模型 (如上百亿参数) 上才能带来性能提升，这表明思维链推理是大语言模型的一种独特能力。

大语言模型中的思维链展现了提示方法的巨大潜力。通过精心设计的提示，我们可以诱导语言模型展现出类似人类的逐步推理能力。这为开发更加智能、可解释的语言模型应用铺平了

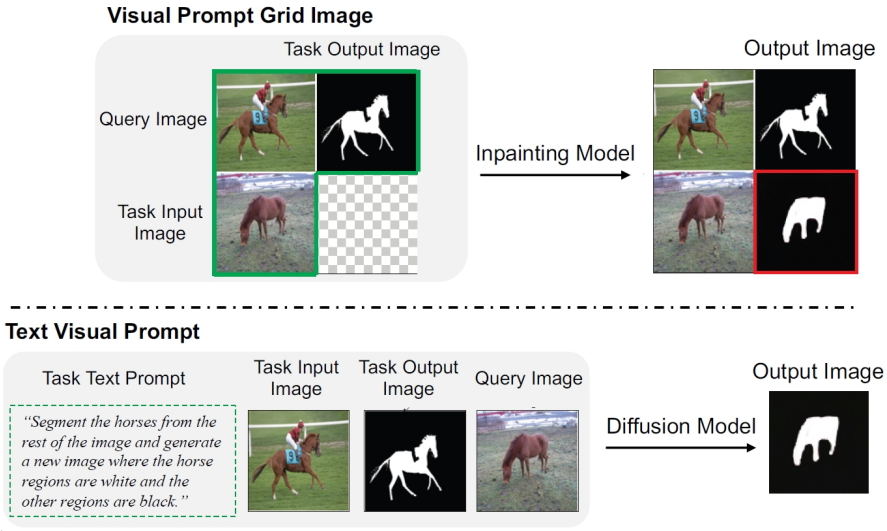


图 14.13: 视觉上下文学习中的纯图像和文本增强提示。来源于Dong et al. (2022)

道路。

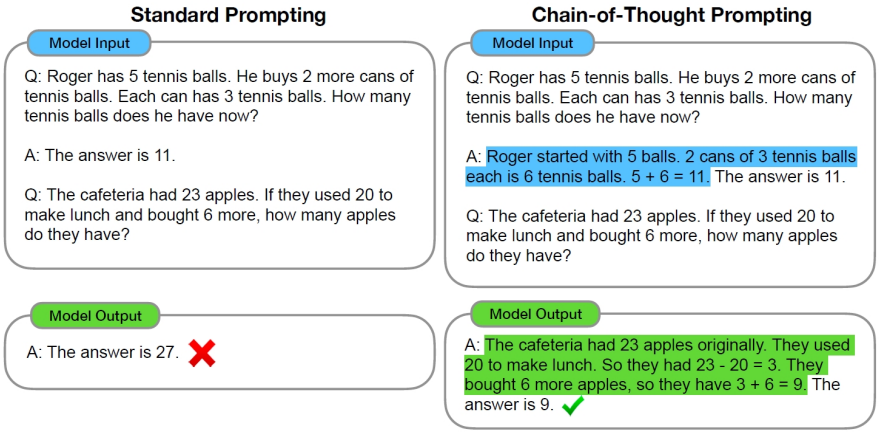


图 14.14: 思维链提示的例子。图中展示了标准提示 (左) 和思维链提示 (右) 在一个数学应用题上的区别

Chapter 15

神经网络求解微分方程

15.1 引言

神经网络技术已经成为解决各种科学和技术问题的关键工具。这些问题范围广泛，从自然语言处理到图像识别，再到游戏策略的制定，神经网络展示了其在捕捉和参数化复杂映射关系中的强大能力。以围棋为例，这一源远流长的策略游戏曾被视为人工智能的终极挑战。但 AlphaGo 通过神经网络模型对棋局策略进行参数化描述，并利用强化学习不断优化，终于突破了人类专家的极限，实现了人机大战的全面胜利。这项超凡成就彰显了神经网络在理解复杂决策过程、提炼优化策略方面的无与伦比的潜能。在自然语言处理领域，大语言模型如 ChatGPT 凭借对海量文本数据的学习，掌握了语言的细微结构和语义信息，能够自如地完成语言理解和生成任务，输出缜密连贯的文本内容。这种全新的语言能力全归功于神经网络在捕获语言本质特征方面的独特魅力。在计算机视觉领域，神经网络也正在重塑计算机视觉的未来。扩散模型通过神经网络参数化将噪声映射为图像生成的逆过程，成功破解了长期困扰图像生成领域的瓶颈，为人工智能创造出高质量、多样化的视觉世界铺平了道路。

神经网络技术不仅在自然语言处理和图像识别等领域取得了巨大成功，其在科学问题上的应用也同样令人瞩目。在许多科学领域，研究人员利用神经网络来模拟和预测复杂系统的行为。例如，在分子动力学模拟中，神经网络被用于建立分子构型与势能函数之间的映射关系。通过学习大量的分子结构数据，神经网络能够准确捕捉到这种复杂的非线性关系，从而加速分子动力学模拟的进行，为研究分子体系的性质和演化提供了有力工具。另一个典型的例子是 AlphaFold，这是一个基于深度学习的蛋白质结构预测系统。了解蛋白质的三维结构对于理解其生物学功能至关重要，但实验测定蛋白质结构通常耗时耗力。AlphaFold 通过训练神经网络学习蛋白质序列与结构之间的关系，成功地预测了大量蛋白质的三维结构，为生命科学领域带

来了革命性的突破。

这些成功案例的背后，是神经网络强大的学习能力。在许多复杂的科学任务中，涉及的映射关系往往呈现出高度非线性和复杂的高维结构。面对这种情况，神经网络能够从大量数据中自动学习和捕获这些复杂的映射关系，因此成为解决这类问题的利器。特别是人工智能在科学领域的应用 (AI for Science) 中，研究人员广泛利用神经网络来对复杂的科学问题进行建模和参数化，这已经成为当前研究的一大热点。通过训练神经网络学习复杂系统的内在规律，科学家们可以更高效、准确地模拟和预测各种物理、化学、生物学过程，加速科学发现的步伐。

在本章中，我们将从深度学习求解微分方程入手，介绍神经网络在科学问题中的一系列应用与挑战，展示深度学习方法的潜力和优势。

微分方程是数学、物理学、工程学等多个学科中描述现象和解决问题的基本工具。它们用于表达变量随时间或空间变化的关系，是理解和预测许多科学和技术领域中系统行为的核心数学语言。从牛顿运动定律到麦克斯韦电磁场方程，微分方程为我们提供了一个强大的框架来描述和分析各种自然和人造系统的动态行为。通过建立微分方程模型，研究者可以深入洞察系统的内在机理，预测其未来的演化，并探索各种因素对系统行为的影响。根据导数的类型，微分方程可分为常微分方程 (Ordinary Differential Equations，ODEs) 和偏微分方程 (Partial Differential Equations，PDEs) 两大类。常微分方程主要处理系统因变量对单一自变量 (通常是时间) 的变化关系。常微分方程在描述集中参数系统、人口动力学、化学反应动力学等领域有广泛应用。与之不同，偏微分方程考虑多个自变量 (如时间和空间) 的导数，这使得它们能够描述更加复杂的分布参数系统的动态行为。偏微分方程在流体动力学、热传导、电磁学、量子力学等领域发挥着关键作用。通过偏微分方程，我们可以研究波的传播、湍流的产生、场的分布等复杂现象。

微分方程的求解是一个重要而有挑战性的问题。对于一些简单的微分方程，我们可以通过解析方法求得精确解。然而，对于大多数实际问题，微分方程往往非常复杂，难以求得解析解。在这种情况下，数值方法成为求解微分方程的重要手段。传统的数值算法，如欧拉法、有限元法和有限体积法等，通过离散化时间和空间，将微分方程转化为代数方程组，再使用计算机进行求解。这些方法在过去几十年中取得了显著进展，推动了计算数学领域的快速发展。然而，尽管取得了巨大的成功，传统数值算法在处理某些类型的问题时仍面临挑战，例如：

1. 对于高度非线性、多尺度的微分方程，传统数值方法往往需要非常精细的网格剖分和极小的时间步长，导致计算成本十分高昂；

2. 对于具有复杂边界的问题，传统算法的网格划分过程需要花费大量的计算资源；

3. 对于反问题，传统算法通常需要求解大量的正问题，严重降低了计算效率；

4. 对于高维偏微分方程，一些传统方法存在维数灾难的问题，即计算复杂度随着维数的增

加呈指数级增长。

这些困难导致某些复杂微分方程问题长期未能找到有效的解决方案。

近年来，随着人工智能技术的发展，特别是深度学习的进步，一种新的思路受到关注，即使用神经网络来解决微分方程问题。这种方法利用神经网络强大的函数拟合能力，通过训练神经网络来近似微分方程的解。与传统数值方法不同，基于神经网络的方法可以自动学习问题的内在结构和规律，无需手动设计精细的网格和离散化方案。详细介绍请参考鄂维南教授的综述文章 (E et al. 2021)。

在人工智能与科学交叉的研究领域中，替代模型的概念尤为重要。首先，我们需要明确所要解决的问题，并将其转化为一个优化问题。通过求解这一优化问题，我们能够直接获得所需的解。例如，在物理学中，能量最小化问题是一个非常常见且自然存在的优化问题，这使得我们的工作变得相对简单。

以求解牛顿运动问题为例，假设一个质量为 m 的物体，它在一个一维管道中受到一个随时间变化的力的作用，$F = \sin(t) + t$，请问它的位置与时间的关系是什么？由于力的形式比较复杂，因此，我们很难直接理论上把公式算出来。于是，我们假设位置 $s(t)$ 近似满足多项式的形式：$s(t) = a * t + b * t^2 + c * t^3 + d * t^4 + e$。这里 a, b, c, d, e 是待定的参数，那要如何确定这些参数呢？根据牛顿第二定律：

$$m\frac{d^2s(t)}{dt^2} = F = \sin(t) + t, \tag{15.1}$$

假设初始状态是 $s(0) = 0$ 和 $\frac{ds(t)}{dt}|_{t=0} = 0$，可以算出 $a = 0, e = 0$。事实上，这就是一个二阶的常微分方程的问题。显然由于力的形式比较复杂，这个多项式形式是无法严格满足上述的微分方程。那我们可以退而求其次，找到一组最靠近的参数就可以。我们希望在每个时间点 t 上，上述等式的误差最小，以平方差为例，即 $|m\frac{d^2s(t)}{dt^2} - (\sin(t) + t)|^2$ 最小。假设我们只关心 $t \in [0, T]$。显然，我们也不能取到所有可能的时间点，一种常用的方法就是在这个时间段时随机采 n 个点，计算误差

$$L(b, c, d) = \sum_{i=1}^{n} |m\frac{d^2s(t)}{dt^2} - (\sin(t) + t)|^2. \tag{15.2}$$

事实上，这就是一个关于 b, c, d 的三维函数，我们要找到一组 b, c, d 使这个函数值最小。梯度下降等算法就可以做到。

对于不能直接找到小化形式的问题，我们可以寻找与问题相关的等式形式，例如演化方程或能量守恒定律。通过将等式的左侧减去右侧并对结果进行平方，我们可以将其转化为一个最小二乘问题。这种方法不仅简化了问题的复杂性，还为我们提供了一种有效的求解途径。像上述的问题，对于初始值的情况，我们也可以把 $t = 0$ 的情况代入，获得初始位置和初始速度的等式，然后得到两个平方差最小的目标函数，再把三个目标函数求和一起优化。

在处理涉及连续积分的问题时，我们可以通过离散化的方式将其转化为求和的近似形式，这使得计算变得更加可行和高效。在这个框架下，上述提到的多项式拟合会带来诸多问题，特别是对高维问题，神经网络的角色变得尤为关键。神经网络可以被用来替代最小化问题中的某些部分，具体来说，神经网络可以直接作为要求解的函数，或者作为要求解函数的一部分，从而帮助我们更好地捕捉复杂的非线性关系。

这些基于神经网络的方法为解决复杂的微分方程提供了新的视角和工具。它们不仅展示了神经网络在捕获高维、非线性映射方面的强大能力，也为计算数学领域带来了新的解决方案。这一章，我们将介绍一些基本的方法，并从现象出发，对这些方法做简单的讨论。

基于神经网络求解微分方程的方法大致可分为两大类。第一类方法是用神经网络参数化解的方法，针对特定微分方程的定制化解决方案。这类方法通过构建一个参数化的神经网络来表示微分方程的解，并通过优化神经网络的参数，使其输出的解尽可能满足微分方程的条件。这类求解方法在二十世纪九十年开始逐步发展，通过方程左端减右端的方式获得最小二乘损失函数 (Dissanayake & Phan-Thien 1994)，但受到深度学习低谷的影响，这类方法逐渐被人淡忘。近年来，随着深度学习重新崛起，这类方法又再次引起科研人员的注意。在 2017 年，该方法被重新命名为物理驱动的神经网络（Physics-informed Neural Network，PINN）(Raissi et al. 2019)。类似地，求解微分方程的各类等价问题均可以用来构造神经网络的损失函数，例如 Ritz 变分的最小化可以用来求解椭圆方程，以 Ritz 变分形式构造损失函数的方法被称为 Deep Ritz 方法 (E & Yu 2018)，以及类似的 Deep Nitsche 方法 (Liao & Ming 2021)。

第二类神经网络求解微分方程的方法是通过参数化算子的方法，这类方法具有更强的通用性，旨在用神经网络对一整类微分方程的算子进行参数化。这类方法不局限于求解单个微分方程，而是试图学习微分方程的广义算子，从而为求解一大类问题提供统一的框架，比如直接求解源项和解在离散点之间的映射关系 (Khoo et al. 2017)，DeepOnet (Lu et al. 2019) 和 Fourier Neural Operator (Li et al. 2020*b*) 等方法。但这类方法要达到预期的泛化能力并不容易。

15.2 优势与不足

15.2.1 优势

融合机理与数据 深度学习方法能够有效融合物理机理与数据驱动的特性。通过引入惩罚项，可以将物理定律、边界条件和初始条件等约束直接嵌入到损失函数中。这种方式不仅提高了模型的预测精度，还确保了求解结果在物理上是合理的，从而增强了模型的可信度。

神经网络逼近高维函数的能力　深度神经网络具备强大的函数逼近能力，能够处理高维空间中的复杂函数。相较于传统数值方法，深度学习能够有效地捕捉高维问题中的非线性特征，提供更精确的解。这一特性对于解决复杂的偏微分方程尤为重要。

无网格化处理高维和复杂边界问题　深度学习方法不依赖于网格划分，这使得它在处理高维和复杂几何边界问题时具有显著优势。传统的数值方法通常需要精细的网格划分，而深度学习方法可以在任意点进行求解，避免了网格生成和优化带来的计算负担。

自动微分的优势　深度学习框架通常内置自动微分功能，能够方便地计算神经网络的导数。这一特性使得深度学习方法特别适合求解微分方程，因为可以直接利用网络输出计算偏微分方程的残差，并通过反向传播算法高效地更新网络参数。

并行计算优势　深度学习模型的训练过程可以充分利用现代 GPU 的并行计算能力。这一特性使得大规模数据集的处理和复杂模型的训练变得更加高效，显著缩短了求解时间。这对于需要实时或近实时解的应用场景（如流体仿真、气候建模等）尤为重要。

快速求解一类问题　参数化算子方法在推理阶段可以快速地求解一类方程。例如，当边界条件发生变化时，训练好的神经网络算子不用再经过训练，可直接快速求出对应的解。

求解反问题　方程中需要被确定的一些参数可以直接作为优化问题求解的变量之一，因此，正反问题的求解可以使用统一的形式。同时，若借助于神经算子求解一类方程，套用传统算法求解反问题的过程，需要求解大量的正问题，可以用神经网络快速求解算法中的正问题。在反问题的求解过程中，方程中需要确定的参数可以直接作为优化问题的变量之一，从而使得正问题和反问题的求解可以以统一的形式进行。此外，通过利用神经算子来求解特定类型的方程，可以有效地应用传统算法来处理反问题。在这一过程中，神经网络能够快速求解正问题，从而避免传统算法需要花大量时间求解大量正问题，提高整体求解效率。

15.2.2　不足

对训练数据的依赖　深度学习方法需要大量的训练数据才能取得好的效果，而获取高质量的训练数据并不容易。特别是对于高维问题，在高维空间获得足够多的高质量的样本点是十分困难的。

缺乏理论保证　深度学习方法的收敛性、稳定性等理论分析还不够完善，难以给出严格的误差估计。

对物理约束的满足　深度学习的解可能无法严格满足微分方程的物理约束，需要采取一些策略来强化约束条件。

计算资源需求大　训练复杂的深度学习模型需要大量的计算资源和时间，对硬件要求较高。

可解释性较弱　深度学习模型通常被视为黑箱，难以解释其内部工作机制，这在一些对可解释性有严格要求的应用中可能成为障碍。

低精度　受到数据质量和训练方法的限制，深度学习方法的求解精度通常低于复杂的传统数值格式。

15.3　参数化解的方法

15.3.1　最小二乘法

本小节将介绍使用神经网络求解偏微分方程的最小二乘法，该方法也被称为物理驱动的神经网络方法 (Physics-informed neural network, PINN) (Dissanayake & Phan-Thien 1994, Raissi et al. 2019)。假设我们要解决的偏微分方程具有如下形式：

$$\mathcal{N}(u(x,t)) = f(x,t), \quad (x,t) \in \Omega \times [0,T], \tag{15.3}$$

其中，$\mathcal{N}$ 是微分算子，$u(x,t)$ 是待求解的未知函数，$f(x,t)$ 是已知的源项，Ω 是空间域，$[0,T]$ 是时间区间。

定义神经网络解为 $u_\theta(x,t)$，用于近似解 $u(x,t)$。损失函数 $L(\theta)$ 可以表示为：

$$L(\theta) = L_{\text{PDE}}(\theta) + L_{\text{IC}}(\theta) + L_{\text{BC}}(\theta) + L_{\text{data}}(\theta), \tag{15.4}$$

其中，L_{PDE} 用于量化模型对偏微分方程控制律的满足程度，L_{IC} 和 L_{BC} 分别表征模型在初始条件与边界条件上的拟合误差，L_{data} 衡量模型在空间域内已知数据点处的预测偏差。各部分的具体形式如下：

$$L_{\text{PDE}}(\theta) = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left| \mathcal{N}(u_\theta(x_i,t_i)) - f(x_i,t_i) \right|^2. \tag{15.5}$$

这里，(x_i,t_i) 是在计算域 $\Omega \times [0,T]$ 中随机选取的点，N_{PDE} 是选取的点的数量。

$$L_{\text{IC}}(\theta) = \frac{1}{N_{\text{IC}}} \sum_{j=1}^{N_{\text{IC}}} \left| u_\theta(x_j,0) - u_0(x_j) \right|^2, \tag{15.6}$$

其中，$u_0(x_j)$ 是初始条件的真实值，$(x_j, 0)$ 是在初始时刻的随机选取点，N_{IC} 是选取的点的数量。

$$L_{\text{BC}}(\theta) = \frac{1}{N_{\text{BC}}} \sum_{k=1}^{N_{\text{BC}}} \left| u_\theta(x_k, t_k) - g(x_k, t_k) \right|^2 . \tag{15.7}$$

这里，$g(x_k, t_k)$ 是边界条件的真实值，(x_k, t_k) 是在边界上的随机选取点，N_{BC} 是选取的点的数量。

$$L_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{m=1}^{N_{\text{data}}} \left| u_\theta(x_m, t_m) - \hat{u}(x_m, t_m) \right|^2 , \tag{15.8}$$

其中，$\hat{u}(x_m, t_m)$ 是观测数据的值，(x_m, t_m) 是观测数据的随机选取点，N_{data} 是观测数据点的数量。PINNs 的关键优势在于损失函数可以在域内的任意点进行评估，无需预定义网格或格点。这使得处理复杂几何形状和高维问题变得高效。

15.3.2 变分方法

Deep Ritz 变分方法 (E & Yu 2018) 的核心思想是利用深度神经网络来近似求解能量泛函的最小化问题，从而获得偏微分方程的解。

以泊松方程为例，考虑以下形式的方程：

$$-\Delta u(x) = f(x), \quad x \in \Omega, \tag{15.9}$$

其中，Δ 是拉普拉斯算子，$u(x)$ 是待求解的未知函数，$f(x)$ 是已知的源项，Ω 是定义域。我们还需要考虑边界条件，例如狄利克雷（Dirichlet）边界条件：

$$u(x) = g(x), \quad x \in \partial\Omega, \tag{15.10}$$

其中，$g(x)$ 是在边界 $\partial\Omega$ 上给定的函数。为了解这个方程，我们可以定义一个能量泛函：

$$E(u) = \frac{1}{2} \int_\Omega |\nabla u|^2 \, dx - \int_\Omega f u \, dx. \tag{15.11}$$

我们的目标是找到使能量泛函 $E(u)$ 最小化的函数 u，同时满足边界条件。

在 Deep Ritz 方法中，我们使用一个深度神经网络 $u_\theta(x)$ 来近似解 $u(x)$。为了确保网络输出满足边界条件，我们可以在损失函数中加入边界条件的约束。通过优化网络参数 θ，我们最小化能量泛函的损失函数：

$$L(\theta) = E(u_\theta) + L_{\text{BC}}(\theta) = E(u_\theta) + \frac{1}{N_{\text{BC}}} \sum_{k=1}^{N_{\text{BC}}} \left| u_\theta(x_k) - g(x_k) \right|^2 . \tag{15.12}$$

在这里，$L_{\text{BC}}(\theta)$ 是边界条件损失，x_k 是在边界上的随机选取点，N_{BC} 是选取的边界点的数量。对于含时问题，同样可以在损失函数中加入关于初始值的损失。如果问题也有提供一些测量数据，也可类似地加到损失函数中。

在实际计算中，通常使用数值积分方法来近似上述积分，例如蒙特卡洛积分或自适应积分技术。

Deep Ritz 方法的优点在于对解的正则性要求会降低，求解需要求解的导数的阶数会更低，对于训练更友好，同时能够有效地处理边界条件。

15.3.3 弱解求解法

弱对抗网络（Weak Adversarial Networks, WAN）(Zang et al. 2020) 利用了偏微分方程的弱解形式。以二阶椭圆型偏微分方程为例，考虑以下形式的方程：

$$-\Delta u(x) = f(x), \quad x \in \Omega, \tag{15.13}$$

其中，Δ 是拉普拉斯算子，$u(x)$ 是待求解的未知函数，$f(x)$ 是已知的源项，Ω 是定义域。为了求解该方程，我们需要考虑其弱形式。

在弱形式中，目标是找到满足以下条件的弱解：

$$\int_{\Omega} \nabla u(x) \cdot \nabla \phi(x)\, dx = \int_{\Omega} f(x)\phi(x)\, dx, \quad \forall \phi \in H_0^1(\Omega). \tag{15.14}$$

其中，ϕ 是测试函数，$H_0^1(\Omega)$ 是适当的希尔伯特空间。

在 WAN 方法中，我们将弱解和测试函数分别参数化为原始网络和对抗网络。具体而言，我们定义两个神经网络：

- 原始网络 $u_\theta(x)$ 用于近似弱解。
- 对抗网络 $\phi_\eta(x)$ 用于近似测试函数。

通过交替更新这两个网络的参数，WAN 能够有效地逼近偏微分方程的弱解。损失函数的构造基于弱形式的条件：

$$L(\theta, \eta) = \left\| \int_{\Omega} \nabla u_\theta(x) \cdot \nabla \phi_\eta(x)\, dx - \int_{\Omega} f(x)\phi_\eta(x)\, dx \right\|^2, \tag{15.15}$$

为了实现这一目标，WAN 方法通过对抗训练的方式，将原始网络和对抗网络的参数进行优化。具体而言，我们可以将问题转化为一个最大最小优化问题：

$$\min_{\theta} \max_{\eta} L(\theta, \eta). \tag{15.16}$$

在这个优化过程中，原始网络的参数 θ 被优化以最小化损失，而对抗网络的参数 η 被优化以最大化损失。这样的对抗训练机制使得网络能够更好地捕捉到问题的复杂性，从而提高解的精度。

15.4 参数化算子的方法

传统方法与直接参数化解的神经网络方法，有一个相同的问题，只能求解单个方程。对于一个新的例子（源项或者边界条件改变，要解一族方程的），如果每次都要重新求解，计算代价就会很大。一种非常直接的思路就是，能否学习函数之间的映射，例如，输入是初始条件的函数，输出则是问题的解函数。

当然，我们面临的首要问题是，怎么把一个函数作为神经网络的输入或者输出呢？最简单的办法就是，在一些固定网格点上采点来表示一个函数。因此，输入就是初始条件函数在选定网格点上的采样，即一个向量；输出要逼近的就是目标函数在选定网格点上的值，也是一个向量。神经网络要学习的就是一个高维向量到高维向量的映射。可以通过传统算法求解很多不同的初始条件获得数据。该方法在 2017 年被发展起来Khoo et al. (2017)。

对于神经网络的输出，我们可以进一步做如下改变。神经网络只输出一个位置的值，这个位置信息和初始条件函数一起作为神经网络的网络。这样的好处是可以获得目标函数的任意点上的值。Deep Operator Network (DeepONet) 采用了这种方式Lu et al. (2019)，并做了一些结构上的创新。

还有另一种常见的神经网络算子的形式是借助于微分方程的格林函数表示，例如，如果一个微分方程的解关于边值条件可以表示成如下形式，

$$u(x) = \int_D G(x, y) f(y) \mathrm{d}y, \tag{15.17}$$

其中 $G(x, y)$ 是格林函数，$f(y)$ 是边值条件函数。那可以学习一个神经网络来表示格林函数，然后通过离散积分的形式快速获得目标函数的近似。傅里叶神经算子正是利用了这个想法Li et al. (2020a)。

前面提到的神经网络算子都是基于数据驱动，并没有用到方程的具体信息。事实上，在损失函数中，也可以加入如 PINN 中使用的控制方程最小化的部分，有助于提升模型的性能，这个探索最早可见于Zhang, Luo, Zhang, E, Xu & Ma (2022)。

15.5 传统算法和神经网络对于低频的不同偏好

在本节中，我们将以一维泊松方程为例，展示传统算法和神经网络对于低频的不同偏好。

我们考虑的一维泊松方程如下：

$$-\Delta u(x) = g(x) \quad x \in \Omega = (-1\ 1), \tag{15.18}$$

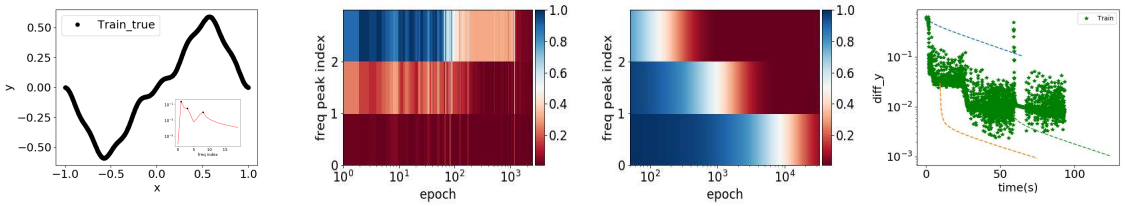


图 15.1: 神经网络与传统算法求解泊松方程的不同频率表现。(a) $u_{\text{ref}}(x)$. 插入图: $|\hat{u}_{\text{ref}}(k)|$ 关于频率 k 的函数图像. 黑色的点对应频率的峰值。(b，c) 针对所选的几个频率，对 DNN(b) 和雅可比方法 (c) 不同 epoch 处得到的训练数据上的 $\Delta_F(k)$. (d) $\|h - u_{\text{ref}}\|_\infty$ 在不同运行时间上的值。

其中 $g(x) = \sin(x) + 4\sin(4x) - 8\sin(8x) + 16\sin(24x)$，且边界条件为：$u(x) = 0 \quad x = -1, 1.$

我们的目标是使用神经网络 $u_\theta(x)$ 来逼近真实解 $u_{ref}(x)$。为此，我们在区间 [0 1] 上均匀采样 1001 个训练点 $\{x_i\}_{i=0}^n$，相邻点之间的距离记为 h。为了避免直接计算高阶导数，我们采用 Deep Ritz 方法，即使用泊松方程的变分形式 (15.19)作为损失函数：

$$I_{\text{emp}} = \sum_{i=1}^{n-1} \left(\frac{1}{2} |\nabla_x u(x_i)|^2 - g(x_i) u(x_i) \right) h + \beta \left(u(x_0)^2 + u(x_n)^2 \right). \tag{15.19}$$

其中，β 是一个固定的权重系数，用于处理边界条件，也被称为惩罚项。通过不断优化这个损失函数，我们期望神经网络 $u_\theta(x)$ 能够逼近真实解 $u_{ref}(x)$。

在训练过程中，我们重点关注 $u_{ref}(x)$ 的傅里叶变换结果中三个峰值的收敛情况（如图 15.1(a)）。图 15.1(b) 清晰地展现了神经网络在训练过程中对不同频率成分的学习偏好：低频成分比高频成分收敛得更快，这与频率原则相一致。

与之相对，传统的数值方法在处理低频情况时表现出完全不同的倾向。以有限差分法为例，我们首先将泊松方程离散化，得到一个线性方程组，然后可以使用各种线性代数方法求解，例如直接法（如高斯消元或 LU 分解）和迭代法（如雅可比迭代或共轭梯度法）。特别地，迭代方法对于解决来源于微分方程离散化的大型稀疏线性系统尤为适用，因为这类方法可逐步改进解，且无需存储完整的矩阵，从而使得计算过程更加高效。但此类方法往往存在低频误差大的缺点，如结果 15.1(c) 所示，传统方法很难消除低频误差。这种频率偏好可以从理论上得到证明，详见本小节的高阶部分。

基于传统方法和神经网络的对于低频成分的不同偏好，一个直观的想法是在训练的前期使用神经网络快速学习低频特征，然后在后期切换为传统方法，以更高效地捕捉学习较高频特征。

在实验中，我们首先使用神经网络来求解公式 Eq.(15.18)) 中的泊松方程，其中 $g(x) =$

$\sin(x)+4\sin(4x)-8\sin(8x)+16\sin(24x)$。为了使神经网络在停止训练时已经很好地拟合低频部分，我们需要合理选择训练步数 M。接下来，我们将神经网络输出的离散函数值作为新的初始值，输入到雅可比迭代方法中，继续进行收敛计算。在实验过程中，我们使用 $\|h-u_{\text{ref}}\|_\infty \triangleq \max_{x\in\Omega}|h(x)-u_{\text{ref}}(x)|$ 来量化误差。

图. 15.1(d) 展示了不同方法的收敛情况，绿色星号表示仅使用神经网络解决方程，可以观察到误差 $\|h-u_{\text{ref}}\|_\infty$ 在迭代几次后出现波动。虚线表示在神经网络训练到一定程度后，将其结果作为雅可比迭代的初始条件，继续进行迭代。如果停止训练的步数 M 过小，即结果主要依赖雅可比迭代法，那么算法需要更长的时间才能收敛，因为雅可比方法对低频部分收敛慢。反之，如果 M 过大 (对应图中右侧的虚线)，结果就主要依赖神经网络，此时较高频部分的收敛较慢。选取合适的 M 值 (对应图中的橙色虚线) 可以兼顾两种方法的优势: 利用神经网络快速拟合低频部分，再借助雅可比方法高效捕捉较高频细节，从而以更快的速度得到收敛的解。

这个例子给我们带来了重要启示: 在解决某些问题时，单独使用 DNN 可能并非最佳选择。这是因为 DNN 在高频分量的学习上存在收敛缓慢的局限性，可能影响求解效率和精度。利用 DNN 和传统方法的优势来设计更快的方案可能是科学计算问题的一个有希望的方向，比如 Huang et al. (2020) 用神经网络作为迭代算法的初始值在很多问题上加速了求解。

15.5.1 多尺度神经网络解 PDE

神经网络对高频误差收敛慢不仅影响计算效率，还可能导致精度问题。高频误差主要包含物理现象中的小尺度特征，如麦克斯韦方程中的高频电磁场。如果这些高频部分没有被准确解析，可能导致求解出的整体物理过程缺失关键的细节信息，进而影响我们对物理现象的理解和应用。

为了克服神经网络学习高频特征慢的问题，第五章提出了一种多尺度神经网络 (Mscale-DNN)，使用这种网络可以有效地学习高频信息。下面我们通过一个多孔介质中的二维问题示例来说明 MscaleDNN 的优势。

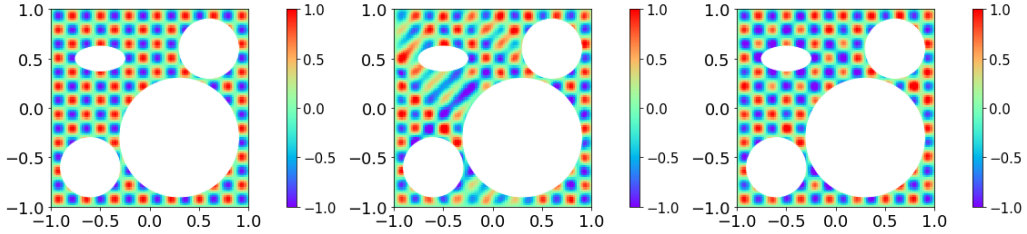
考虑带有如下源项的泊松方程 (15.18)：

$$g(\boldsymbol{x}) = 2\mu^2 \sin\mu x_1 \sin\mu x_2 \quad \mu = 7\pi, \tag{15.20}$$

其精确解为

$$u(\boldsymbol{x}) = \sin\mu x_1 \sin\mu x_2. \tag{15.21}$$

我们使用精确解来给出边界条件。如图 1 所示，MscaleDNN 能够准确地捕捉精确解中的振荡特征，而正常的神经网络则难以准确拟合这些高频震荡。



(a) 精确解　(b) 正常神经网络　(c) MscaleDNN

图 15.2: 多孔二维的例子。图片来源于Liu et al. (2020)。

15.5.2 小结

传统的数值算法在处理高维问题时，常常会遭遇所谓的“维数灾难”——随着问题维度的增加，计算复杂度呈指数级增长，导致数据处理和操作变得极为困难。然而，神经网络在应对高维问题方面展现出了显著优势，特别是在拟合高维、非线性以及具有复杂结构的映射时，神经网络的表现尤为突出。

神经网络强大的高维拟合能力是其主要吸引力之一。当我们需要求解具有复杂边界条件或者特殊性质的偏微分方程时，深度学习可以作为一种强有力的工具，用于捕捉和学习数据中的结构和模式。与传统方法相比，神经网络有可能在简化模型复杂性的同时，保持较高的求解精度和计算效率，为解决偏微分方程提供了一种新颖的策略。

15.6 高阶内容*

15.6.1 传统方法对高频问题快速收敛的原因

我们以 Section 15.5中的泊松方程为例进行分析，

有限差分方法

有限差分方法是一种广泛应用于求解偏微分方程数值解的传统方法。首先，我们将求解区间 $[-1\ 1]$ 离散化，划分为 $n+1$ 个节点，其中各节点的网格间距为 $h=2/n$。随后，利用中心差分格式对泊松方程进行离散化求解，具体表示为:

$$-\Delta u_i = -\frac{u_{i+1}-2u_i+u_{i-1}}{(h)^2} = g(x_i), \quad i = 1\ 2\ \cdots\ n, \tag{15.22}$$

将上述方程组织成线性方程组形式，得到:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{g}, \tag{15.23}$$

其中

$$\boldsymbol{A}=\frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \cdots & & & \vdots \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}_{(n-1)\times(n-1)}, \tag{15.24}$$

$$\boldsymbol{u}=\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1}, \end{pmatrix} \quad \boldsymbol{g}=\begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ g_{n-1}. \end{pmatrix} \quad x_i=-1+2\frac{i}{n}. \tag{15.25}$$

得到离散矩阵 $\boldsymbol{A}$ 和相应的线性方程组后，可使用雅可比迭代法求解该方程组。令 $\boldsymbol{A}=\boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U}$，其中 $\boldsymbol{D}$ 是 $\boldsymbol{A}$ 的对角矩阵，$\boldsymbol{L}$ 和 $\boldsymbol{U}$ 分别是 $-\boldsymbol{A}$ 的严格下三角和上三角部分，则我们有

$$\boldsymbol{u}=\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{u}+\boldsymbol{D}^{-1}\boldsymbol{g}. \tag{15.26}$$

在第 $t\in\mathbb{N}$ 步，雅可比迭代公式如下，

$$\boldsymbol{u}^{t+1}=\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{u}^{t}+\boldsymbol{D}^{-1}\boldsymbol{g}. \tag{15.27}$$

我们对上述迭代过程进行标准误差分析。用 $\boldsymbol{u}^*$ 表示直接对 Eq. (15.23) 中的 $\boldsymbol{A}$ 求逆得到的真解。第 $t+1$ 步的误差是 $\boldsymbol{e}^{t+1}=\boldsymbol{u}^{t+1}-\boldsymbol{u}^*$. 则可以得到 $\boldsymbol{e}^{t+1}=\boldsymbol{R}_J\boldsymbol{e}^t$, 其中 $\boldsymbol{R}_J=\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$. 因此 $\boldsymbol{e}^t$ 的收敛速度是由 $\boldsymbol{R}_J$ 的特征值决定的，也就是，

$$\lambda_k=\lambda_k(\boldsymbol{R}_J)=\cos\frac{k\pi}{n} \qquad k=1,2,\cdots,n-1, \tag{15.28}$$

相应的特征向量 $\boldsymbol{v}_k$ 的分量是

$$v_{k\,i}=\sin\frac{ik\pi}{n}, i=1,2\ \cdots,n-1. \tag{15.29}$$

为了进一步剖析，我们对 $\boldsymbol{e}^t$ 进行分解：

$$\boldsymbol{e}^t=\sum_{k=1}^{n-1}\alpha_k^t\boldsymbol{v}_k, \tag{15.30}$$

其中 α_k^t 为 $\boldsymbol{e}^t$ 在 $\boldsymbol{v}_k$ 方向的投影。则

$$\boldsymbol{e}^{t+1}=\sum_{k=1}^{n-1}\alpha_k^t\boldsymbol{R}_J\boldsymbol{v}_k=\sum_{k=1}^{n-1}\alpha_k^t\lambda_k\boldsymbol{v}_k. \tag{15.31}$$

$$\alpha_k^{t+1} = \lambda_k \alpha_k^t.$$

因此，$\boldsymbol{e}^t$ 在 $\boldsymbol{v}_k$ 方向上的收敛速度是被 λ_k 控制的. 由于

$$\cos\frac{k\pi}{n} = -\cos\frac{(n-k)\pi}{n}, \tag{15.32}$$

频率 k 和 $(n-k)$ 具有相同的收敛速率，当 k 很小或很大时，λ_k 的绝对值接近于 1，此时，从(15.31)可以看出，高频误差和低频误差都很难消除。

因此，在雅可比迭代方法中，低频和高频收敛都会比较慢。针对高频收敛缓慢的问题，可以通过加权雅可比迭代来解决。对于低频收敛缓慢的问题，研究者们提出了多种方法。其中，多重网格算法是一种重要的途径，通过在不同尺度的网格上进行计算，可以有效地加速低频误差的收敛。一系列的研究工作不断发展和完善了多重网格算法，使其成为解决许多实际问题的有力工具。

multigrid 小故事

多重网格算法的历史源远流长，其中出现了很多有趣的小故事，比如 1. 多重网格方法的奠基人之一 Achi Brandt 在上世纪 60 年代获得灵感，提出了这一开创性的方法。据说，他的灵感来源于对棋盘游戏的观察。在国际象棋和围棋中，高手通常先考虑全局的战略布局，然后再逐步细化到具体的落子。这种从整体到局部的思考方式启发了 Brandt ，让他萌生了在不同尺度上解决问题的思路，进而提出了多重网格方法的雏形。2. 多重网格方法另一位重要先驱 Fedorenko 在 1964 年发表了一篇关键论文，系统阐述了多重网格方法的数学原理。但由于当时的历史背景，这篇发表在苏联的论文并没有引起西方学者的广泛关注。直到上世纪 70 年代，Brandt 的工作让多重网格方法重新引起重视，人们才发现 Fedorenko 的论文早已提出了相似的思想。3. 另一位多重网格方法大家 Wolfgang Hackbusch 在总结多重网格理论时，提出了“平滑、递归、插值”三个关键词，形象地概括了多重网格方法的核心要素。这个简洁而精辟的总结被称为 Hackbusch 三元组，成为学习和理解多重网格方法的一个重要框架，广为流传。

15.7 习题

1. 深度学习方法与传统数值方法相比有何异同点?

2. 对于泊松方程15.18. 请写出其有限差分格式，并说明所得代数方程组的系数矩阵 A 的结构特点。

3. 写出雅可比迭代格式，并分析收敛性。特别地，求出雅可比迭代矩阵的特征值和特征向量。

Chapter 16

实验手册：频率原则的验证

Jupyter 代码可以在 GitHub 找到[1]。

16.1 神经网络基本概念

神经网络是一个有许多参数的机器学习方法。假设我们有一组训练集 $S = \{(x_i, y_i)\}_{i=1}^n, (x_i, y_i) \in \mathbb{R}^2$。考虑一个只有一层隐藏层的神经网络，即两层神经网络，

$$f_\theta(x) = \sum_{j=1}^m a_j \sigma(w_j x + b_j). \tag{16.1}$$

为了方便起见，定义

$$h_i = f_\theta(x_i). \tag{16.2}$$

常见的激活函数 $\sigma(x)$ 为 $\mathrm{ReLU}(x) = \max\{0, x\}$、$\tanh(x)$ 等。我们这里假设 x, w, b, a 都是一维的标量，对于高维的情形，后面会有介绍。我们可以通过（随机）梯度下降来调节参数。当梯度下降用在神经网络模型中时，它有一种特殊的名字：向后传播法。考虑训练的损失函数为均方差:

$$L_S = \frac{1}{2n} \sum_{i=1}^n (y_i - h_i)^2. \tag{16.3}$$

对于参数集 $\boldsymbol{\theta} = (a_1, w_1, b_1, a_2, w_2, \cdots, b_m)$, 我们通过梯度下降法进行训练：

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\partial L_S}{\partial \boldsymbol{\theta}_t}. \tag{16.4}$$

[1] https://github.com/xuzhiqin1990/understanding_dl/tree/main/code/frequency_principle

因为链式法则，梯度下降的计算顺序是 $h_i \to a_j \to \sigma \to x_i$，而信息流 (information flow) 的顺序是 $x_i \to \sigma \to a_j \to h_i$，这两者的顺序相反，因此称其为向后传播。优化后，通过取一些新的采样点计算测试误差来判断拟合的好坏。

下面用矩阵的形式写 $f_\theta(\boldsymbol{x})$:

$$f_\theta(\boldsymbol{x}) = \boldsymbol{W}^{[2]}\sigma \circ (\boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}), \tag{16.5}$$

其中 $\boldsymbol{x} \in \mathbb{R}^{d\times 1}, \boldsymbol{W}^{[1]} \in \mathbb{R}^{m\times d}, \boldsymbol{b}^{[1]} \in \mathbb{R}^{m\times 1}, \boldsymbol{W}^{[2]} \in \mathbb{R}^{d_o\times m}$，“$\circ$” 的意思是对应元素的运算 (entry-wise operation)（例如对应元素相乘）。这里 d 是输入数据的维度，m 是隐藏层神经元的个数，d_o 是输出维度。有时我们会同时计算 n 个样本的值，比如在实际编程计算中，把 $\boldsymbol{x}$ 写成矩阵形式:

$$\boldsymbol{Y} = h(\boldsymbol{X}) = \boldsymbol{W}^{[2]}\sigma \circ (\boldsymbol{W}^{[1]}\boldsymbol{X} + \boldsymbol{B}^{[1]}), \tag{16.6}$$

其中 $\boldsymbol{X} \in \mathbb{R}^{d\times n}, \boldsymbol{Y} \in \mathbb{R}^{[d_o\times n]}, \boldsymbol{B}^{[1]} \in \mathbb{R}^{m\times n}$。$\boldsymbol{B}^{[1]}$ 的每一列都为 $b^{[1]}$, 即 $\boldsymbol{B}^{[1]} = [b^{[1]}, b^{[1]}, \cdots, b^{[1]}]$。用这些符号可以类似得定义一个深度神经网络。

一个 L 层神经网络记为

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]}\sigma \circ (\boldsymbol{W}^{L-2}\sigma \circ (\cdots (\boldsymbol{W}^{[1]}\sigma \circ (\boldsymbol{W}^{[0]}\boldsymbol{x} + \boldsymbol{b}^{[0]}) + \boldsymbol{b}^{[1]}) \cdots) + \boldsymbol{b}^{[L-2]}) + \boldsymbol{b}^{[L-1]}, \tag{16.7}$$

其中 $\boldsymbol{W}^{[l]} \in \mathbb{R}^{m_{l+1}\times m_l}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{m_{l+1}}, m_0 = d_{in} = d, m_L = d_o$，$\sigma$ 是一个向量函数。注意一下在计算网络层数时，输入层不计入（即层数为隐藏层 + 输出层的层数）我们把参数集记为

$$\boldsymbol{\theta} = (\boldsymbol{W}^{[0]}, \boldsymbol{W}^{[1]}, \cdots, \boldsymbol{W}^{[L-1]}, \boldsymbol{b}^{[0]}, \boldsymbol{b}^{[1]}, \cdots, \boldsymbol{b}^{[L-1]}),$$

把 $\boldsymbol{W}^{[l]}$ 中的元素记为 $\boldsymbol{W}^{[l]}_{ij}$。也可以用递归的方法定义这个网络:

$$f^{[0]}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{x} \tag{16.8}$$

$$f^{[l]}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma \circ (\boldsymbol{W}^{[l-1]} f^{[l-1]}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \boldsymbol{b}^{[l-1]}), \quad 1 \le l \le L-1 \tag{16.9}$$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f^{[L]}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]} f^{[L-1]}_{\boldsymbol{\theta}}(\boldsymbol{x}) + \boldsymbol{b}^{[L-1]} \tag{16.10}$$

16.2 实验背景

我们在理论部分已经介绍了深度学习的一些基本知识, 也指出了理解神经网络这样一个复杂系统的挑战性。因此, 找到一个合适的切入点去分析神经网络, 对于剖析这个” 黑盒子” 的内部机制尤为重要。在科学研究中, 人们对复杂系统的探索往往起源于对基本现象的观察和分析。例如, 在遗传学研究中, 孟德尔选择了相对简单的豌豆作为研究对象, 通过观察豌豆性状的分离和组合规律, 总结出了遗传的基本定律。这些定律后来被应用于解释更加复杂的生物遗传现象。

再如, 在电磁学研究中, 安培和法拉第等科学家首先研究了相对简单的电场和磁场现象, 发现了电磁感应定律等基本规律。在此基础上, 麦克斯韦进一步将这些规律统一为麦克斯韦方程，解释了更复杂的电磁现象, 预言了电磁波的存在。

受此启发, 我们在研究神经网络时, 也可以先从一些非常简单的问题入手, 通过在简单模型中观察、分析、总结规律, 再尝试将这些认识推广到更加复杂的神经网络中。当然, 这种推广需要谨慎, 并非所有从简单模型中得到的结论都能直接应用于复杂模型。但这种研究方法可以帮助我们抓住问题的关键, 为进一步探索复杂网络提供有益的思路和启发。基于上述考虑, 在神经网络的实验探究过程中, 我们选择从最简单的拟合一维函数的问题开始进行研究。一方面, 一维问题结构简单、直观, 便于观察分析网络的特性; 另一方面, 很多神经网络的基本特性, 如学习过程的非线性、在常规设定下不易过拟合等, 在一维问题中就已经能够得到充分的体现。通过对一维问题的研究, 我们可以初步认识神经网络的一些重要特点, 为进一步探索更加复杂的网络打下基础。因此, 在本节中, 我们将基于一维问题展开讨论, 通过具体的实例来向读者展示神经网络在学习过程中的一般规律。

16.3 实验目标

我们要验证神经网络训练中频率原则的适用性，即神经网络在训练时先学习低频成分，随后逐步学习高频成分。通过对网络输出和目标函数的频率成分进行分析，我们希望能够清晰地观察到这一现象。

在进行实验设计时，我们需要控制一些关键变量，确保实验结果能够反映频率原则的表现。此外，我们将通过**相对误差**来定量评估神经网络对不同频率成分的学习速度。

16.4 理论基础

16.4.1 离散傅里叶变换（DFT）

离散傅里叶变换（Discrete Fourier Transform, DFT）是傅里叶分析在离散数据上的一种实现方法。它通过将有限的离散时间信号分解为频率成分，从而揭示信号在频域中的特性。对于一个给定的离散信号 $\{x_n\}$，其长度为 N，其 DFT 定义为:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N}, \quad k = 0, 1, \ldots, N-1.$$

其中，X_k 表示信号在第 k 个频率上的幅值。通过 DFT，我们可以得到信号的频谱，从而分析信号中的不同频率成分。

在神经网络的学习过程中，通过 DFT 可以观察不同频率成分的变化情况，进一步分析频率原则的实现过程。然而，在进行频谱分析时，需要注意采样率的选择。如果采样率不足，会产生假频现象，即在频谱中出现不存在的频率成分。因此，在实际操作中，我们需确保足够的采样点来避免假频的出现。

下面的代码展示了一个利用 DFT 分析信号的示例，通过改变采样点数来观察频率分布。

代码片段：

```
from utils import dft_analysis

"""
func: dft_analysis(f, T, N)

    Perform DFT analysis on a given function f with a total time of
        T and N samples.

    Args:
        f (function): The function to be analyzed.
        T (float): The total time of the signal.
        N (int): The number of samples.

    Returns:
        None
"""


def f(t):
    return np.sin(2*np.pi*t) + np.sin(6*np.pi*t)


print('sample size=50: ')
dft_analysis(f=f, T=2, N=50)

print('sample size=8: ')
dft_analysis(f=f, T=2, N=8)
```

16.4.2 频率原则

频率原则指出：在训练过程中，神经网络倾向于先拟合数据的低频成分，而高频成分则需要较长的时间才能被有效学习。这一现象可以通过傅里叶变换来理解。傅里叶变换能够将时间或空间域中的函数分解为一系列正弦或余弦波，这些波按照频率由低到高排列。通过傅里叶分析，我们能够检测到神经网络如何逐步学习这些不同频率的成分。

16.5 实验步骤

16.5.1 数据生成与预处理

目标函数为三项正弦波的叠加，表示为：

$$f(x) = \sin(x) + \sin(3x) + \sin(5x).$$

该函数生成了同时包含低、中、高频成分的信号，便于验证神经网络的频率学习过程。

代码片段：

```
def get_y(x):

    """
    Function to fit.

    Args:
        x (float): input value.

    Returns:
        float: output value.
    """
    alpha=2
    y = np.sin(x)+np.sin(3*x)+np.sin(5*x)
    return y



args.training_input, args.training_target, args.test_input, args.
   test_target = get_dataset(args, get_y)
```

```
print("The training data size: ", args.training_input.shape)


print("The target function:")
plot_target(args)
```

变量控制：

- **数据点数**：我们需要保证训练数据足够密集，特别是在高频区域，确保高频部分被合理采样。

16.5.2 构建模型

模型为多层感知机（MLP），使用自定义的网络结构和激活函数。每层的神经元数量及激活函数可以通过命令行参数控制。

代码片段：

```
act_func = get_act_func(args.act_func_name)

#Initialize the neural network model
model = Model(args.t, args.hidden_layers_width, args.input_dim,
               args.output_dim, act_func).to(args.device)
```

16.5.3 模型训练及测试过程

我们对初始化完成的模型通过梯度下降法进行训练。我们使用均方差损失作为损失函数，并使用 Adam 优化器。

代码片段：

```
# Define the optimizer: determine the gradient descent optimization
    algorithm, the default is Adam.
if args.optimizer=='sgd':
    optimizer = torch.optim.SGD(model.parameters(), lr=args.lr)
else:
    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
```

```
# Define the loss function: the loss function is the mean square
    error.
loss_fn = nn.MSELoss(reduction='mean')
```

对于每个 epoch，我们均进行一步训练及一步测试，并保存模型损失值、模型输出等。

代码片段：

```
for epoch in range(args.epochs+1):
    # Set the model to training mode
    model.train()

    # Train one step
    loss, training_output = train_one_step(
    model, optimizer, loss_fn, args)

    # Test the trained model
    loss_test, output = test(
        model, loss_fn, args)

    # Record the loss and output
    args.loss_training_lst.append(loss)
    args.loss_test_lst.append(loss_test)
    args.training_output.append(training_output.detach().cpu().
       numpy())

    # Print the loss and time
    if epoch % args.plot_epoch == 0:
        print("[%d] loss: %.6f valloss: %.6f time: %.2f s" %
                (epoch + 1, loss, loss_test, (time.time()-t0)))
```

16.5.4 相对误差的计算

为了定量评估神经网络对不同频率成分的学习速度，我们使用**相对误差**来观察频率原则。相对误差衡量了模型输出的频率分量和目标频率分量之间的差异。通过相对误差，我们可以清

楚地看到不同频率成分的拟合进度。为避免 DFT 产生的误差影响，我们选取目标函数频域空间的振幅峰值对应的频率进行观测。

相对误差计算公式

$$\Delta F(k) = \frac{|h_k - f_k|}{|f_k|},$$

其中 h_k 表示模型输出在频率 k 处的幅值，f_k 表示目标函数在频率 k 处的幅值。

代码片段：

```
# create the absolute error array
abs_err = np.zeros([len(idx1), len(args.training_output)])

# calculate the absolute error
tmp1 = y_fft[idx1]
for i in range(len(y_pred_epoch)):
    tmp2 = my_fft(y_pred_epoch[i])[idx1]
    abs_err[:, i] = np.abs(tmp1 - tmp2)/(1e-5 + tmp1)
```

为什么使用相对误差：

- **频率成分差异评估**：相对误差提供了模型拟合的准确性，尤其是在高频成分较难学习时，误差的变化能够直观地展示不同频率成分的收敛速度。
- **避免绝对值误差影响**：高频成分的幅值往往较低，直接使用绝对误差可能会使误差看起来很小。使用相对误差可以平衡不同频率成分的幅值，确保误差计算的公平性。

16.5.5 实验结果分析与可视化

训练过程结束后，我们可以基于上述计算所得的相对误差，绘制相对误差图。通过对比不同频率成分的误差变化曲线，可以验证网络是否遵循频率原则。

代码片段：

```
def plot_abs_err(args, abs_err):

    """
    Plot the heatmap of the relative error for different
       frequencies.
```

```
    Args:
    args: A dictionary containing save path.
    abs_err: The absolute error array.

    Returns:
    None
    """

    # initialize the figure
    plt.figure(figsize=(8, 6))
    ax = plt.gca()

    # plot the heatmap of the relative error
    plt.pcolor(abs_err, cmap='RdBu', vmin=0.1, vmax=1, linewidths
        =0.4)

    # set the colorbar
    plt.colorbar()

    # set the x-axis labels and its fontsize
    plt.xlabel('Epoch', fontsize=22)

    # Set the y-axis ticks and labels to 1, 2, 3
    plt.yticks([0.5, 1.5, 2.5], [1, 2, 3], fontsize=22)

    # Set the y-axis tick parameters to hide the tick marks and set
        the tick label size
    plt.gca().yaxis.set_tick_params(size=0)
    plt.gca().tick_params(axis='y', labelsize=22)
    plt.gca().tick_params(axis='x', labelsize=22)
    plt.title('Absolute Error', fontsize=22)
    plt.tight_layout()
```

```
    # save the figure
    plt.savefig(os.path.join(args.path, 'hot.png'))
    plt.show()
    plt.close()
```

最后实验结果如图16.1所示。左图表示拟合结果。中间图像为目标函数与模型输出在频域空间的变换结果，可以发现目标函数有三个主要的频率成分且对应的幅值都相等。红点表示目标函数频域分布的峰值，也是我们刻画在不同频率模型学习快慢的目标频率。右图热力图表示三个主要频率的神经网络拟合结果随训练进行的相对误差，横坐标表示训练 epoch，左坐标轴 index 数字越小代表频率越低。红色代表神经网络拟合结果的相对误差很小，蓝色代表相对误差较大。在训练的过程中频率最低的成分收敛地最快，几个 epoch 就已经拟合地很好，而高频成分却用了 2000 个 epoch 才达到了较小的相对误差。可以由热图看出，模型先拟合低频成分，再拟合高频。

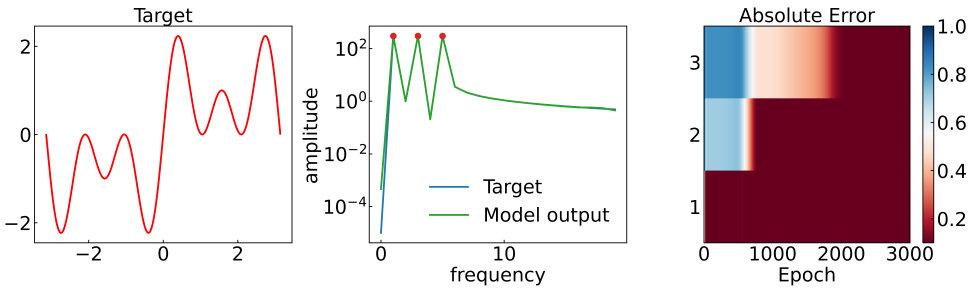


图 16.1: 控制目标函数幅值相同的一维数据实验

16.6 任务

神经网络的拟合过程仍然受到许多因素的影响，如：参数初始化分布、学习率、目标函数、激活函数等。这些因素的变化是否会影响频率原则？可以通过控制变量的方式，研究某些参数对频率原则的影响。

接下来，我们提供了两种实验方案，用于研究目标函数变化对频率原则的影响。这两种方案分别通过改变目标函数中高频部分的频率和幅度，探索其对频率原则的影响。（下述两个任务中的 10% 是可修改的。）

示例 1：研究不同频率幅度如何影响模型的收敛速度。以目标函数 $sin(x) + a * sin(4x)$ 为例，其中 a 为给定的参数。我们可以将横坐标设为 a，纵坐标设为在该目标函数下，满足高频

相对误差 <10% 所需的步数与满足低频相对误差 <10% 所需步数的比值。

示例 2：研究不同高频如何影响模型的收敛速度。以目标函数 $sin(x) + 0.2 * sin(k \cdot x)$ 为例，其中 k 为给定的参数。我们可以将横坐标设为 k，纵坐标设为在该目标函数下，满足高频相对误差 <10% 所需的步数与满足低频相对误差 <10% 所需步数的比值。

Chapter 17

实验手册：参数凝聚现象的验证

Jupyter 代码可以在 GitHub 找到[1]。

17.1 实验简介

在神经网络的训练过程中，不同的初始化对神经网络的训练动力学有非常大的影响。在本实验中，我们将通过一个简单的一维实验观察不同初始化下，神经网络参数的动力学演化情况。其中在小初始化时神经网路会表现出参数凝聚的现象，以下是参数凝聚现象的一个示意图：

这是一个理想的凝聚的示意图。在初始化时，各个神经元的输入权重差异很大，用不同的颜色表示。但是在经过一段时间训练后，中间隐藏神经元分成了两类，前两个神经元是一类，后三个神经元是另一类。在每一类中，不同神经元的输入权重是完全一样的 (颜色相同)，因此，它们的输出也是一样的。

17.2 实验目标

希望通过本实验，可以直观感受不同的初始化下神经网络在参数层面的演化过程，以及其对神经网络输出结果的不同表现；同时也可以比较不同的神经网络激活函数对神经网络初始凝聚的影响。

[1] https://github.com/xuzhiqin1990/understanding_dl/tree/main/code/condense_exp

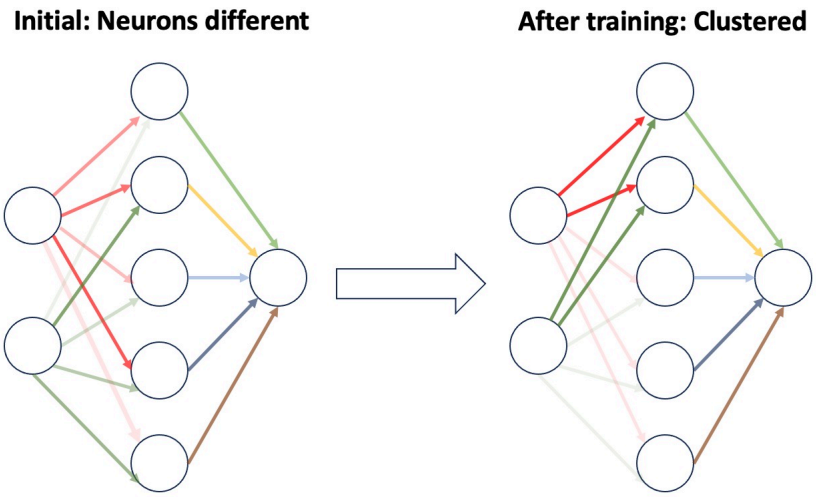


图 17.1: 理想的凝聚现象。

17.3 理论基础

- **线性区域：**当初始化较大时，神经网络在初始化附近就能拟合好数据，参数的相对变化范围非常小，因此模型在初始化附近可以用一阶泰勒展开近似:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx f_{\boldsymbol{\theta}}^{\text{lin}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}(0)}(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}(0)}(\boldsymbol{x}) \cdot (\boldsymbol{\theta}(t) - \boldsymbol{\theta}(0)). \tag{17.1}$$

- **凝聚区域：**当初始化较小时，神经网络在初始化附近不能拟合好数据，网络参数的相对变化范围较大，因此网络不能用一个线性模型近似，表现出高度的非线性行为。
- **临界区域：**介于线性和凝聚区域之间，网络表现出中等程度的非线性行为。

17.4 实验原理

17.4.1 神经网络结构

本实验使用一个简单的前馈神经网络，包含一个输入层、一个隐藏层和一个输出层。网络结构如下:

$$f_{\boldsymbol{\theta}}(x) = \sum_{k=1}^{m} a_k \sigma(\boldsymbol{w}_k^T x + b_k),$$

其中，

- 输入维度 d_{in}
- 隐藏层宽度 m 可调。
- 输出维度 d_{out}

17.4.2 参数初始化

参数初始化对神经网络的训练至关重要。本实验使用以下初始化方法：

- 权重初始化: $\boldsymbol{w}_k \sim \mathcal{N}(0, \varepsilon^2 I_d)$
- 偏置初始化: $b_k \sim \mathcal{N}(0, \varepsilon^2)$
- 输出层初始化: $a_k \sim \mathcal{N}(0, \varepsilon^2)$

其中，$\varepsilon = \frac{1}{m^\gamma}$，$\gamma$ 是一个可调参数，用于控制初始化的方差，从而控制初始化的大小。
下面是代码中用于初始化网络和控制网络初始化的部分：

代码片段：

```
class Linear(nn.Module):
    def __init__(self, t, hidden_layers_width=[100],  input_size
       =20, num_classes: int = 1000, act_layer: nn.Module = nn.ReLU
       ()):
        super(Linear, self).__init__()
        self.num_classes = num_classes
        self.input_size = input_size
        self.hidden_layers_width = hidden_layers_width
        self.t = t
        layers: List[nn.Module] = []
        self.layers_width = [self.input_size]+self.
           hidden_layers_width
        for i in range(len(self.layers_width)-1):
            layers += [nn.Linear(self.layers_width[i],
                                 self.layers_width[i+1]),
                                    act_layer]
        layers += [nn.Linear(self.layers_width[-1], num_classes,
           bias=False)]
```

```
        self.features = nn.Sequential(*layers)
        self._initialize_weights()


    def forward(self, x):

        x = x.view(x.size(0), -1)
        x = self.features(x)
        return x

    def _initialize_weights(self) -> None:

        for obj in self.modules():
            if isinstance(obj, (nn.Linear, nn.Conv2d)):
                nn.init.normal_(obj.weight.data, 0, 1 /
                                self.hidden_layers_width[0]**(self.
                                   gamma))
                if obj.bias is not None:
                    nn.init.normal_(obj.bias.data, 0, 1 /
                                    self.hidden_layers_width[0]**(
                                       self.gamma))
```

17.4.3 激活函数

本实验将比较不同激活函数对网络性能的影响。可选的激活函数包括:

- ReLU: $\max(0, x)$
- tanh: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Sigmoid: $\frac{1}{1+e^{-x}}$
- $x * \tanh(x)$

下面是代码中用于获取激活函数类别的函数

代码片段:

```
def get_act_func(act_func):
    if act_func == 'Tanh':
        return nn.Tanh()
    elif act_func == 'ReLU':
        return nn.ReLU()
    elif act_func == 'Sigmoid':
        return nn.Sigmoid()
    elif act_func == 'xTanh':
        return xtanh()
    else:
        raise NameError('No such act func!')

act_func = get_act_func(args.act_func_name)
```

创建模型的代码

代码片段：

```
model = Linear(args.gamma, args.hidden_layers_width, args.input_dim
    , args.output_dim, act_func).to(args.device)
```

通过上述代码可以创建一个线性网络，同时初始化满足 $\mathcal{N}(0, \frac{1}{m^\gamma})$ 分布。

17.5 实验任务

17.5.1 任务一：不同初始化下神经网络参数演化情况

参考代码：condense.ipynb

实验将使用以下一维函数作为拟合目标:

$$f(x) = 0.2 * \mathrm{ReLU}(x - 1/3) + 0.2 * \mathrm{ReLU}(-x - 1/3)$$

运行代码，观察不同初始化方差 (ε) 对训练过程的影响：

设置不同的 γ 值（如 $0.1, 0.5$ 和 1），运行实验并比较结果，观察神经元特征（方向和幅度）的分布变化。需要注意调整学习率使得学习过程中 loss 曲线不会出现上升。

观测指标

对于一维输入的函数，并采用 ReLU 作为激活函数时，神经元方向和幅值的定义：

每个神经元的参数对 $(a_k, \boldsymbol{w}_k)$ 可以分离为一个单位方向特征 $\hat{\boldsymbol{w}}_k = \boldsymbol{w}_k/\|\boldsymbol{w}_k\|_2$ 和一个表示其对输出贡献的振幅 $A_k = |a_k|\|\boldsymbol{w}_k\|_2$, 即 $(A_k, \hat{\boldsymbol{w}}_k)$。对于一维输入，由于加入了偏置项，$\boldsymbol{w}_k = (w_k, b_k)$ 是二维的。因此，我们使用每个 $\hat{\boldsymbol{w}}_k$ 相对于 x 轴的 $[-\pi, \pi)$ 内的角度 $\hat{\Omega}_k$ 来表示其方向。即 $\hat{\Omega}_k = \arctan(\frac{b_k}{w_k})$。

下面是代码中用于计算某个 checkpoint 中神经元方向和幅值的代码

代码片段：

```
def get_ori_A(checkpoint):
    wei1, bias, wei2 = get_parameter(checkpoint)

    wei1 = wei1.squeeze()
    bias = bias.squeeze()
    wei2 = wei2.squeeze()
    wei = wei1 / (wei1 ** 2 + bias ** 2)**(1/2)

    bia = bias / (wei1 ** 2 + bias ** 2)**(1/2)
    ori = torch.sign(bia) * torch.acos(wei)
    A = wei2 * (wei1 ** 2 + bias ** 2)**(1/2)

    return ori, A
```

预期结果参考图 17.2。

17.5.2 任务二：高维函数的初始参数凝聚实验

参考代码：initial condense.ipynb

实验将采用 5 维输入，1 维输出的函数：

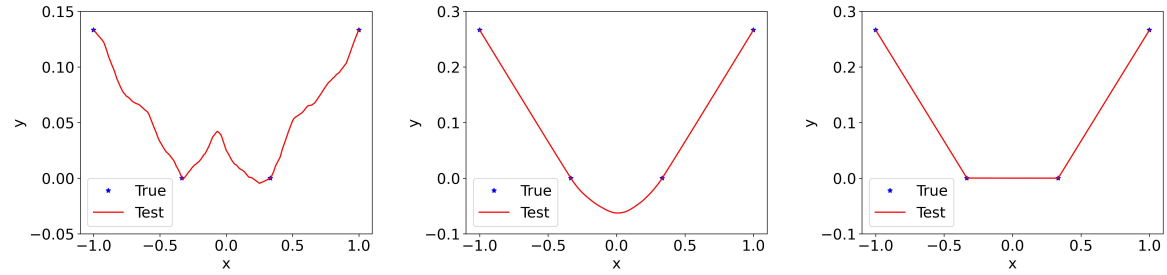
$$f(x) = \sum_{k=1}^{5} 3.5 \sin(5x_k + 1)$$

其中数据 $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)$，从 $[-4, 2]$ 中均匀采样得到。

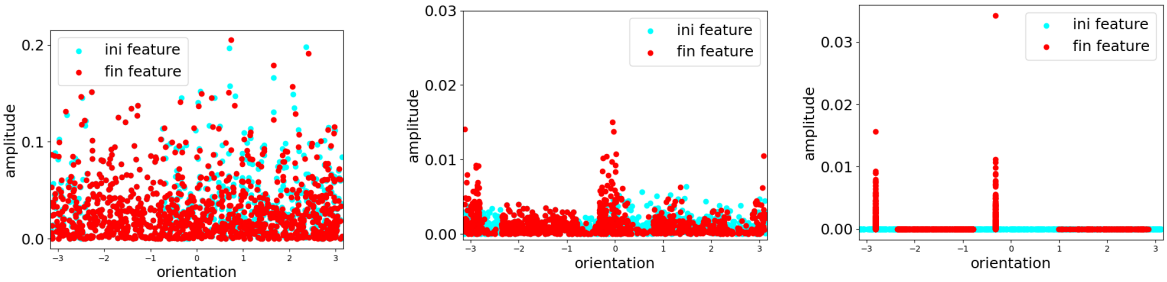
采用 Adam 优化器，以 Tanh 函数作为激活函数，进行实验。

观测指标

对于高维数据, 无法使用极坐标系可视化神经元的特征。在这种情况下, 我们可以利用余弦相似度来衡量两个向量之间夹角的大小。



(a) $\gamma = 0.1$ (b) $\gamma = 0.5$ (c) $\gamma = 1.0$



(d) $\gamma = 0.1$ (e) $\gamma = 0.5$ (f) $\gamma = 1.0$

图 17.2: 不同大小初始化下，两层 ReLU 网络的预期实验结果。

余弦相似度: 两个向量 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 的余弦相似度定义为

$$D(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^\intercal \boldsymbol{v}}{(\boldsymbol{u}^\intercal \boldsymbol{u})^{1/2} (\boldsymbol{v}^\intercal \boldsymbol{v})^{1/2}}. \tag{17.2}$$

如果余弦相似度为 1，则说明两个向量平行

下面是代码中用于计算某个 checkpoint 中神经元余弦相似度的代码

代码片段:

```
def seperate_vectors_by_eigenvector(vector_group):
    mask = np.linalg.norm(vector_group,axis=1) > 0
    vector_group = vector_group[mask]
    similarity_matrix = np.dot(vector_group,vector_group.transpose
        ())
    w,v = np.linalg.eig(similarity_matrix)
    index = np.argmax(w)
    tmpeig = v[:,index]
    order_mask = np.argsort(tmpeig)
```

```
    similarity_matrix = similarity_matrix[order_mask,:]
    similarity_matrix = similarity_matrix[:,order_mask]
    return similarity_matrix,order_mask
```

需要注意的是为了可视化参数凝聚的聚类，我们按照余弦相似度矩阵的最大特征值对应的特征向量对参数向量进行一个排序，以达到同一方向的向量可以在相邻位置。

预期在 $\boldsymbol{w}_k$ 的结果：

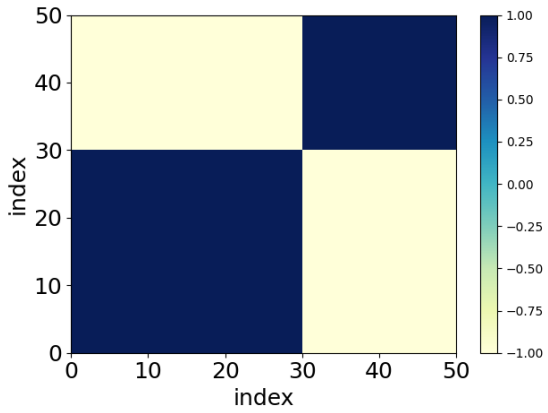


图 17.3: 初始凝聚时余弦相似度热力图结果。

17.6 课后任务和思考

尝试实现以 $x * \tanh(x)$ 为激活函数的两层神经网络的初始凝聚现象，画出其余弦相似度热力图结果。可以思考一下不同激活函数对于初始参数凝聚有什么影响。

Chapter 18

实验手册：乐观估计的验证

Jupyter 代码可以在 GitHub 找到[1]。

18.1 实验背景：过参数化下的泛化

当我们考虑泛化这个问题时，传统机器学习理论强调模型的复杂度应当与训练样本的数量相匹配，以避免过拟合现象的发生。然而，深度学习的实践却往往与这一理论预期形成鲜明对比：即使在模型参数数量远超训练样本数量的情况下，深度神经网络仍然能够展现出优异的泛化性能。这一反直觉的现象被学术界称为" 泛化之谜"（Generalization Puzzle）。本实验旨在通过" 乐观估计"（Optimistic Estimate）理论框架，深入探索这一现象背后的本质机制。乐观估计是一种创新的理论工具，用于估算训练神经网络恢复目标函数（达到零泛化误差）时所需的最小样本量。其核心思想可概括为：

1. 在最优条件下（乐观初始化）考察模型性能，即将参数初始化选择在理想点附近
2. 在该理想点附近对模型进行线性近似
3. 基于线性近似推导出恢复目标函数所需的最小样本量

18.2 实验目标

本实验的核心目标是验证乐观估计理论在一些机器学习模型中的适用性。通过实验，我们将验证理论推导得到的乐观样本量作为恢复目标函数所需样本量下界的有效性。特别地，我们

[1] https://github.com/xuzhiqin1990/understanding_dl/tree/main/code/Optimistic_Estimate

将验证，通过调参，一些简单回归模型和矩阵分解模型恢复目标函数所需样本量就是乐观样本量，一些更复杂的神经网络模型恢复目标函数所需样本量能够接近乐观样本量。

18.3 理论基础

在回归问题中,我们使用模型 $f_{\boldsymbol{\theta}}$ 拟合从目标函数 f^* 采样的 n 个数据点,目标是恢复 f^*。为了使“恢复”可行,我们考虑可以通过模型 $f_{\boldsymbol{\theta}}$ 表示的目标函数,即 $f^* \in \mathcal{F} := \{f(\cdot;\boldsymbol{\theta})|\boldsymbol{\theta} \in \mathbb{R}^M\}$。默认情况下，我们使用梯度下降来解决回归问题

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right) - f^*(\boldsymbol{x}_i)\right)^2. \tag{18.1}$$

解被记为 $f_{\boldsymbol{\theta}_\infty}$，其中 $\boldsymbol{\theta}_\infty$ 表示收敛时的模型参数。接下来我们定义目标集:

Definition 18.3.1. 目标函数对应的参数集合（又叫目标集）$\boldsymbol{\Theta}_{f^*} := \left\{\boldsymbol{\theta} \mid f_{\boldsymbol{\theta}} = f^*, \boldsymbol{\theta} \in \mathbb{R}^M\right\}$

在过参数化下，有很多解使得模型的训练误差为 0，但是其中大部分解的泛化很差，所以在最差情况下，我们就不能恢复目标函数。但是我们之前的实验表明，非线性模型具有过参数化下恢复目标函数的能力，考虑最坏情况的理论不能解释泛化之谜的实验现象。但是，如果我们考虑最好的情况，将参数的初始化选择在一个很好的点，我们可能可以在过参数化下恢复目标函数。

首先，一个平凡的最优情况是，参数的初始化在目标集 $\boldsymbol{\Theta}_{f^*}$ 中，此时样本量为 0 就可以恢复目标函数。但是，第一，这样的样本量的估计不给我们带来任何有用的信息，实践中 0 个样本量显然是不能恢复目标函数的。第二，目标集 $\boldsymbol{\Theta}_{f^*}$ 的体积一般为 0。在均匀随机采样时，取到这种初始化的概率为 0。所以这种初始化是不可行的。一个次优情形（称之为乐观初始化）是参数的初始化在目标集附近，定义如下:

乐观初始化: 在目标集 $\boldsymbol{\Theta}_{f^*}$ 的一个“最优”点 $\boldsymbol{\theta}^*$ 的邻域中初始化。

我们使用线性近似 $f(\cdot;\boldsymbol{\theta}) \approx f^{\text{lin}}\left(\cdot;\boldsymbol{\theta}\right) := f^*(\cdot) + \nabla f_{\boldsymbol{\theta}'}(\cdot)^{\mathrm{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}')$ 来分析在小邻域 $N(\boldsymbol{\theta}', \epsilon)$ 中拟合所需的样本量，其中 $\boldsymbol{\theta}' \in \boldsymbol{\Theta}_{f^*}$。然后，回归问题 (18.1)变为

$$\min_{\boldsymbol{\theta} \in N(\boldsymbol{\theta}', \epsilon)} \frac{1}{n} \sum_{i=1}^{n} \left(f^{\text{lin}}\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right) - f^*(\boldsymbol{x}_i)\right)^2. \tag{18.2}$$

这是一个线性问题，$f^{\text{lin}}\left(\boldsymbol{x}_i;\boldsymbol{\theta}\right)$ 作为 $\boldsymbol{x}$ 的函数位于一个线性空间中，该线性空间的维数是 $\operatorname{rank}(\nabla f_{\boldsymbol{\theta}'}(\cdot)^{\mathrm{T}}) = \dim\left(\operatorname{span}\left\{\partial_{\theta_i} f(\cdot;\boldsymbol{\theta}')\right\}_{i=1}^{M}\right)$。有一个广为人知的结果：我们需要

$$n = \dim\left(\operatorname{span}\left\{\partial_{\theta_i} f(\cdot;\boldsymbol{\theta}')\right\}_{i=1}^{M}\right)$$

个样本来恢复 f^*。受微分拓扑学中秩的启发，我们将这个量称为**模型秩**，即先算模型关于每个参数求导得到一个切函数，然后算这些切函数张成线性空间的维数。

Definition 18.3.2 (**参数点处的模型秩**)**.** 对于参数可微的模型 $f_{\boldsymbol{\theta}}$ 和参数点 $\boldsymbol{\theta}^* \in \mathbb{R}^M$，模型秩定义为：

$$R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}^*) := \dim(\mathrm{span}\{\partial_{\theta_i} f(\cdot;\boldsymbol{\theta}^*)\}_{i=1}^M) \tag{18.3}$$

其中 $\mathrm{span}\{\phi_i(\cdot)\}_{i=1}^M = \{\sum_{i=1}^M a_i\phi_i(\cdot) \mid a_1,\ldots,a_M \in \mathbb{R}\}$ 表示基函数张成的空间 $\dim(\cdot)$ 表示线性函数空间的维数

例 5. 对于一个简单的线性回归模型：$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{x} \in \mathbb{R}^d$，在任何一个参数点 $\boldsymbol{\theta}^*$ 处，模型秩的计算过程为：先关于每个参数 θ_i 求导得到一个切函数 $\partial_{\theta_i} f(\boldsymbol{x};\boldsymbol{\theta}) = x_i$，然后算这些切函数张成线性空间的维数：

$$R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}^*) = \dim(\mathrm{span}\{\partial_{\theta_i} f(\cdot;\boldsymbol{\theta}^*)\}_{i=1}^d) = \dim(\mathrm{span}\{x_i\}_{i=1}^d) = d. \tag{18.4}$$

Definition 18.3.3 (**函数的模型秩**)**.** 对于任意目标函数 $f^* \in \mathcal{F}$，其中 $\mathcal{F} := \{f(\cdot;\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \mathbb{R}^M\}$ 为模型函数空间，令 $\Theta_{f^*} := \boldsymbol{\theta} \mid f(\cdot;\boldsymbol{\theta}) = f^*, \boldsymbol{\theta} \in \mathbb{R}^M$ 表示目标参数集。函数的模型秩定义为：

$$R_{f_{\boldsymbol{\theta}}}(f^*) := \min_{\boldsymbol{\theta}^* \in \Theta_{f^*}} R_{f_{\boldsymbol{\theta}}}(\boldsymbol{\theta}^*) \tag{18.5}$$

接下来，我们将对一些具体的模型，计算出关于目标函数的模型秩，并与实验所需的最小样本量做比较。

18.4 实验步骤

18.4.1 理论计算阶段

1. 对给定模型 $f_{\boldsymbol{\theta}}$ 和目标函数 f^*，根据定义计算模型恢复目标函数所需的乐观样本量 $R_{f_{\boldsymbol{\theta}}}(f^*)$
2. 记录计算过程和理论结果

18.4.2 实验验证阶段

1. 从目标函数中采样，设计递增的样本量序列
2. 对每个样本量的实验：

(a) 使用接近零的小随机值进行参数初始化：过参数化模型在大的随机初始化下很容易出现泛化性能差的情况，我们考虑的是实际更常用的小初始化情况

(b) 采用全批量梯度下降进行训练：实验中的模型和数据相对简单，排除其他随机性的干扰，确保训练稳定性

(c) 仔细调整学习率以确保收敛：学习率的调整对实验结果有重要影响，大的学习率会影响收敛，小的学习率训练太慢

3. 记录每个样本量下模型恢复目标函数的效果，即泛化误差

4. 确定实际所需的最小样本量

5. 与理论计算的乐观样本量进行对比分析

18.5 简单的回归模型实验

18.5.1 理论计算

首先，我们选定一个简单的关于参数线性的模型：

$$f_{\mathrm{L}}(\boldsymbol{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2. \tag{18.6}$$

设定目标函数为 $f^*(\boldsymbol{x}) = 1+x_1$. 该模型具有 $M = 3$ 个参数。考虑任意的 $\boldsymbol{\theta}' = [\theta_0', \theta_1', \theta_2']^T \in \mathbb{R}^3$，有 $R_{f_{\mathrm{L}}}(\boldsymbol{\theta}') = \dim\left(\mathrm{span}\left\{1, x_1, x_2\right\}\right) = 3$。因此，对于任何 $f^* \in \mathcal{F}_{\mathrm{L}} := \{f_{\mathrm{L}}(\cdot;\boldsymbol{\theta}) | \boldsymbol{\theta} \in \mathbb{R}^3\}$，我们有 $R_{f_{\mathrm{L}}}(f^*) = M_{\mathrm{I}} = 3$。

接下来，我们对线性模型进行重新参数化为 $f_{\mathrm{NL}}(\boldsymbol{x};\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2\theta_3 x_2$，函数空间仍然保持为 $\mathcal{F}_{\mathrm{NL}} = \mathcal{F}_{\mathrm{L}}$，但模型在参数上变得非线性。模型的乐观样本量估计如下。作为参数空间 $\mathbb{R}^4$ 上的函数的模型秩为：

$$R_{f_{\mathrm{NL}}}(\boldsymbol{\theta}') = \dim\left(\mathrm{span}\{1, x_1, \theta_3' x_2, \theta_2' x_2\}\right) = \begin{cases} 2, & \theta_2' = \theta_3' = 0, \\ 3, & \text{others.} \end{cases} \tag{18.7}$$

所以模型的有效参数量 $M_{\mathrm{I}} = 3$。通过解决最小化问题 Eq. (18.5)，我们得到模型秩作为函数空间 $\mathcal{F}$ 上的函数为

$$R_{f_{\mathrm{NL}}}(f^*) = \begin{cases} 2, & f^* \in \{a_0 + a_1 x_1 | a_0, a_1 \in \mathbb{R}\}, \\ 3, & f^* \in \{a_0 + a_1 x_1 + a_2 x_2 | a_2 \neq 0, a_0, a_1, a_2 \in \mathbb{R}\}. \end{cases} \tag{18.8}$$

上述估计表明，非线性模型 f_{NL} 恢复 f^* 所需的乐观样本量为 $R_{f_{\mathrm{NL}}}(f^*) = 2$.

18.5.2 实验验证

对于目标函数 $f^*(\boldsymbol{x}) = 1 + x_1$, 我们分别用上述的线性和非线性模型来拟合，记录下每个样本量下模型恢复目标函数的效果。

首先我们定义好目标函数，并设置采样为 $[-1, 1]$ 均匀分布随机采样：

代码片段

```
# 定义目标函数
def f_star(x, a=1.0, b=0.0):
    y = 1 + a * x[:, 0] + b * x[:, 1]
    return y

def generate_data(a, b, num_samples):
    # x = torch.randn(num_samples, 2)
    x = 2 * torch.rand(num_samples, 2) - 1
    y = f_star(x, a, b)
    return x, y

test_size = 1000
x_test, y_test = generate_data(1.0, 0.0, test_size)
test_dataset = torch.utils.data.TensorDataset(x_test, y_test)
```

这个简单模型在 Pytorch 中的实现如下：

```
class NonLinearModel(torch.nn.Module):
    def __init__(self):
        super(NonLinearModel, self).__init__()
        self.theta0 = torch.nn.Parameter(torch.randn(1) * sigma)
        self.theta1 = torch.nn.Parameter(torch.randn(1) * sigma)
        self.theta2 = torch.nn.Parameter(torch.randn(1) * sigma)
        self.theta3 = torch.nn.Parameter(torch.randn(1) * sigma)


    def forward(self, x):
        return self.theta0 + self.theta1 * x[:, 0] + self.theta2 *
           self.theta3 * x[:, 1]
```

训练过程如下：

```
batch_size, num_epochs, lr = 1024, 10000, 1e-1
test_iter = torch.utils.data.DataLoader(test_dataset, batch_size,
    shuffle=True)
train_ls_dict_NL = {}
test_ls_dict_NL = {}
for sample_size in range(1, 4):
    net = NonLinearModel()
    x_train, y_train = generate_data(1.0, 0.0, sample_size)
    train_dataset = torch.utils.data.TensorDataset(x_train, y_train
        )
    train_iter = torch.utils.data.DataLoader(train_dataset,
        batch_size, shuffle=True)
    train_ls_dict_NL[sample_size], test_ls_dict_NL[sample_size] =
        train(net, train_iter, num_epochs, lr, try_gpu(), test_iter=
        test_iter)
```

最终我们可以把线性模型和非线性模型的泛化误差随样本量变化画在一张图热力图上，并标出理论计算出的乐观样本量阈值，与实验结果比对。

```
import seaborn as sns
from matplotlib import cm
from matplotlib.colors import LogNorm, Normalize
set_figsize((3.5, 2.5))

sns.heatmap(np.array([[test_ls_dict[1][-1], test_ls_dict[2][-1],
    test_ls_dict[3][-1]], [test_ls_dict_NL[1][-1], test_ls_dict_NL
    [2][-1], test_ls_dict_NL[3][-1]]]), cmap='RdBu', annot=False,
    norm=LogNorm(vmax=1e0, vmin=1e-8))
plt.vlines(1, 1, 2, colors='yellow', linestyles='dashed')
plt.vlines(2, 0, 1, colors='yellow', linestyles='dashed')

plt.xlabel('Sample size')
plt.ylabel('Model')
plt.xticks([0.5, 1.5, 2.5], ['1', '2', '3'])
```

```
plt.yticks([0.5, 1.5], ['Linear', 'Non-linear'])
plt.show()
```

实验结果如下：

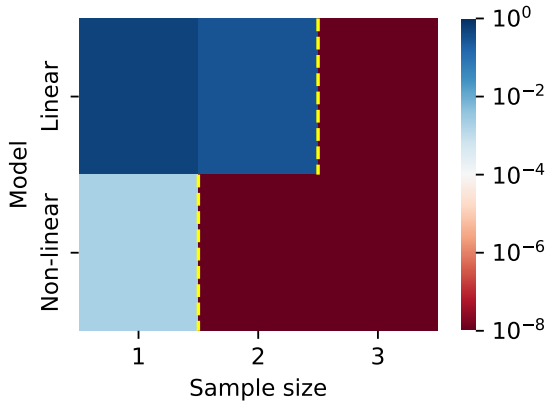


图 18.1: 实验结果

更进一步地，我们可以把线性模型和非线性模型在 1，2，3 样本下的训练过程可视化：

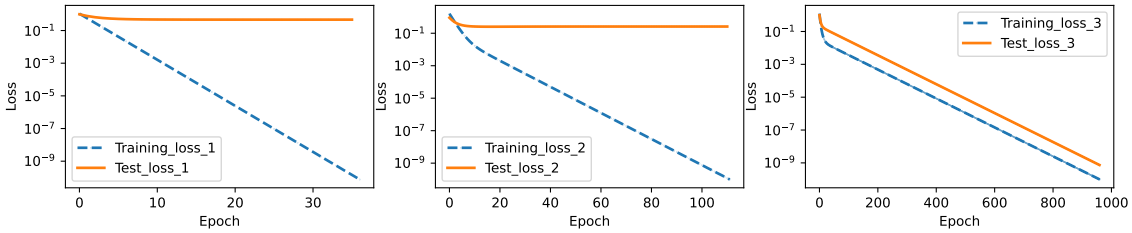


图 18.2: 线性模型。从左到右分别为样本量为 1，2，3 时的训练过程。

18.6 矩阵分解模型实验

18.6.1 理论计算

我们考虑一个 $\boldsymbol{f_\theta} = \boldsymbol{AB}$ 的非线性矩阵分解模型，这个模型在低秩矩阵补全任务中有应用，我们的目标是从 n 个观察到的元素 $S = \left\{\left((i_s, j_s), \boldsymbol{M}^*_{i_s j_s}\right)\right\}_{s=1}^{n}$ 中恢复目标矩阵 $\boldsymbol{M}^*$，其中 (i_s, j_s) 表示矩阵 M^* 的行和列的索引。换句话说，我们已知目标矩阵 $\boldsymbol{M}^*$ 中某些位置的值，我们想还原 $\boldsymbol{M}^*$ 其他位置的值。

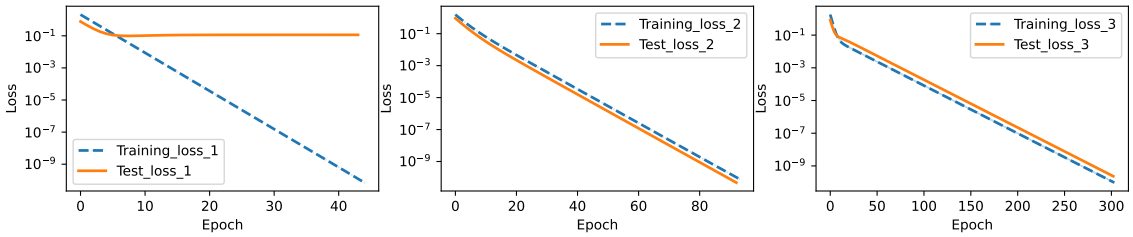


图 18.3: 非线性模型。从左到右分别为样本量为 1，2，3 时的训练过程。

考虑一个非常简单的低秩矩阵补全任务：

$$\boldsymbol{M} = \begin{bmatrix} 1 & 2 \\ 3 & \star \end{bmatrix} \tag{18.9}$$

如果我们直接设定一个关于参数线性的模型 $f_{\boldsymbol{\theta}} = \boldsymbol{W} \in \mathbb{R}^{d\times d}$，并从零附近初始化用梯度下降并最小化以下目标：

$$\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{s=1}^{n} \left([\boldsymbol{W}]_{i_s j_s} - \boldsymbol{M}^*_{i_s j_s}\right)^2, \tag{18.10}$$

$\boldsymbol{W}$ 的第二行第二列的元素拿不到梯度，所以不会被训练，因此一般学不到低秩解。

但是如果我们考虑一个 $\boldsymbol{f_\theta} = \boldsymbol{AB}, \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$ 的非线性矩阵分解模型，采用梯度下降并最小化以下目标：

$$\min_{\boldsymbol{\theta}} \frac{1}{n}\sum_{s=1}^{n} \left([\boldsymbol{AB}]_{i_s j_s} - \boldsymbol{M}^*_{i_s j_s}\right)^2, \tag{18.11}$$

其中 $\boldsymbol{M}^* \in \mathbb{R}^{d\times d}$，$\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{B})$，以及 $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d\times d}$，$[\boldsymbol{AB}]_{i_s j_s}$ 代表取出矩阵 $\boldsymbol{W} = \boldsymbol{AB}$ 第 i_s 行，第 j_s 列的数据。在这个非线性模型中，$\boldsymbol{A}$ 的第二行和 $\boldsymbol{B}$ 的第二列都会被训练，所以模型有可能学到低秩解，从而在只采样 3 个样本的时候就恢复这个秩 1 矩阵。

通过采用标准的乐观估计过程，我们可以确定任何 $\boldsymbol{M}^* \in \mathbb{R}^{d\times d}$ 的乐观样本量为（留作习题）：

$$R_{\boldsymbol{f_\theta}}(\boldsymbol{M}^*) = 2r_{\boldsymbol{M}^*}d - r^2_{\boldsymbol{M}^*}, \tag{18.12}$$

其中 $r_{\boldsymbol{M}^*} = \mathrm{rank}(\boldsymbol{M}^*)$ 是 $\boldsymbol{M}^*$ 的矩阵秩。

实际上，这个结果和秩 r 矩阵的自由度是一样的，例如考虑一个 2×2 的矩阵 $\boldsymbol{A}$，如果要求它的秩为 1，那么它就只能表示为两个向量的外积形式：$\boldsymbol{A} = \boldsymbol{uv}^\top$，其中 $\boldsymbol{u}$ 是一个 2×1 的列向量，$\boldsymbol{v}$ 是一个 2×1 的列向量。因此，矩阵 $\boldsymbol{A}$ 的元素可以用向量 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 的元素表示。具体地说，$\boldsymbol{u}$ 有 2 个分量，$\boldsymbol{v}$ 也有 2 个分量，总共是 4 个分量。然而，它们有一个尺度上的冗余，因为如果我们把 $\boldsymbol{u}$ 扩大某个常数倍，$\boldsymbol{v}$ 缩小同一个常数倍，矩阵 $\boldsymbol{A}$ 的值并不会改变，因此其自由度为 $2rd - r^2 = 2\times 1\times 2 - 1^2 = 3$.

值得注意的是，对于固定的 d，乐观样本量随 $r_{\boldsymbol{M}^*}$ 增加而增加。特别是，对于大的 d，秩为 $O(r)$ 的目标矩阵的乐观样本量为 $O(r^2)$，这比有效参数量 $M_{\mathrm{I}} = d^2$ 要小得多。

我们设定目标矩阵分别为一个 4×4 的秩 1，2，3 矩阵，其乐观样本量 $2rd - r^2$ 计算结果分别为：$7, 12, 15$.

18.6.2 实验验证

对于每个目标矩阵, 我们分别用上述的矩阵分解来拟合，记录下每个样本量下模型恢复目标函数的效果。

首先我们定义好目标矩阵，分别为：

$$\boldsymbol{M}_1^* = \begin{bmatrix} 4 & 0.6 & 1.8 & 0.8 \\ 8 & 1.2 & 3.6 & 1.6 \\ 8 & 1.2 & 3.6 & 1.6 \\ 6 & 0.9 & 2.7 & 1.2 \end{bmatrix}, \boldsymbol{M}_2^* = \begin{bmatrix} 4 & 0.6 & 1.8 & 0.8 \\ 10 & 2.7 & 5.1 & 3.6 \\ 8 & 1.2 & 3.6 & 1.6 \\ 6 & 0.9 & 2.7 & 1.2 \end{bmatrix}, \boldsymbol{M}_3^* = \begin{bmatrix} 4 & 0.6 & 1.8 & 0.8 \\ 8 & 2.7 & 5.1 & 3.6 \\ 8 & 2.2 & 2.6 & 1.6 \\ 6 & 0.9 & 2.7 & 1.2 \end{bmatrix}, \tag{18.13}$$

很容易验证：$\mathrm{rank}(\boldsymbol{M}_1^*) = 1, \mathrm{rank}(\boldsymbol{M}_2^*) = 2, \mathrm{rank}(\boldsymbol{M}_3^*) = 3$.

我们定义采样方式以连通的方式采样，先采第一行第一列，再采第二行第二列，再采第三行第三列，最后采第四行第四列。

模型的实现和训练如下：

代码片段

```
d = 4
num_inputs, num_outputs, num_hiddens = d, d, d
loss = nn.MSELoss()
class MatrixFactorization(nn.Module):
    def __init__(self):
        super(MatrixFactorization, self).__init__()
        self.linear_stack = nn.Sequential(
            nn.Linear(num_inputs, num_hiddens, bias=False),
            nn.Linear(num_hiddens, num_outputs, bias=False)
        )
    def forward(self, x):
        y = self.linear_stack(torch.eye(d, d))
        return y[x.T.numpy().tolist()].reshape(-1, 1), y
```

```
net = MatrixFactorization()
```

训练过程如下：

```
outputs_sample_ls = {}
theta_A_sample_ls = {}
theta_B_sample_ls = {}
batch_size, num_epochs, lr = 1024, 100000, 1e-1
M = M1
test_ls_M1 = {}
for i, ls in enumerate(sample_patterns):
    features_train, labels_train = fix_sample(M, ls=ls)
    train_dataset = torch.utils.data.TensorDataset(features_train,
       labels_train)
    def init_weights(m, sigma=1e-7):
            if type(m) == nn.Linear or type(m) == nn.Conv2d:
                # nn.init.xavier_uniform_(m.weight)
                nn.init.normal_(m.weight, 0, sigma)
                if m.bias:
                    nn.init.normal_(m.bias, 0, sigma)
    net.apply(init_weights)
    train_iter = torch.utils.data.DataLoader(train_dataset,
       batch_size, shuffle=True)
    train_ls, test_ls = train(net, train_iter, num_epochs, lr,
       try_gpu())
    AB = net(features_train)[1].detach().clone()
    A = np.copy(list(net.parameters())[0].detach().T)
    theta_A_sample_ls[i+1]=(A)
    B = np.copy(list(net.parameters())[1].detach().T)
    theta_B_sample_ls[i+1]=(B)
    outputs_sample_ls[i+1]=(AB)
    # frobenius norm
    test_ls_M1[i+1] = np.linalg.norm(AB - M1.numpy(), ord='fro') /
       d ** 2
```

最终我们可以把线性模型和非线性模型的泛化误差随样本量变化画在一张图热力图上，并标出理论计算出的乐观样本量阈值，与实验结果比对。

```
import seaborn as sns
from matplotlib import cm
from matplotlib.colors import LogNorm, Normalize
set_figsize((3.5, 2.5))
test_ls_array = np.array([list(test_ls_M1.values()), list(
    test_ls_M2.values()), list(test_ls_M3.values())])
test_ls_array
sns.heatmap(test_ls_array, cmap='RdBu', annot=False, norm=LogNorm(
    vmax=1e0, vmin=1e-4))
plt.vlines(6, 0, 1, colors='yellow', linestyles='dashed')
plt.vlines(11, 1, 2, colors='yellow', linestyles='dashed')
plt.vlines(14, 2, 3, colors='yellow', linestyles='dashed')
plt.xlabel('Sample size')
plt.xticks(np.array([0, 2, 4, 6, 8, 10, 12, 14])+0.5, [1, 3, 5, 7,
    9, 11, 13, 15], rotation=0)
plt.yticks([0.5, 1.5, 2.5], ['Rank-1', 'Rank-2', 'Rank-3'])
plt.show()
```

实验结果如下，可以看到理论计算得到的乐观样本量和实验所需的最小样本量完全吻合。

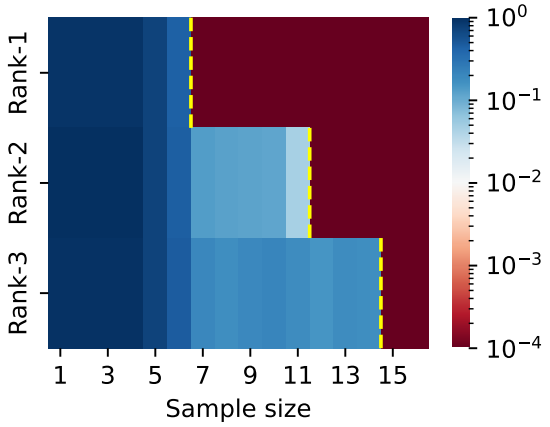


图 18.4: 实验结果。黄线代表理论计算的乐观样本量，colorbar 代表泛化误差。

18.7 神经网络模型实验

18.7.1 理论计算

首先，我们考虑一个具有 m 个神经元的两层全连接神经网络，其激活函数为 $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$，神经网络由 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{i=1}^{m} a_i \tanh(\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{x})$ 表示，其中 $\boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{\theta} = (a_i \in \mathbb{R}, \boldsymbol{w}_i \in \mathbb{R}^d)_{i=1}^m$。网络总共有 $M = m(d+1)$ 个参数。经过计算得到，乐观样本量与目标函数 f^* 的内在宽度 $k(f^*)$ 有关。

Definition 18.7.1. 内在宽度 $k(f^*)$：定义为可以表示 f^* 的神经网络的最小宽度，即 f^* 可以由宽度为 $k(f^*)$ 的神经网络表示，但不能由任何更窄的神经网络表示。

对于一个可以由宽度为 m 的神经网络表示的 f^*，其内在宽度 $0 \leqslant k(f^*) \leqslant m$。$f^*$ 的乐观样本量为

$$R_{\mathrm{NN}_m}(f^*) = k(f^*)(d+1), \tag{18.14}$$

实现该乐观样本量的 $\boldsymbol{\theta}^*$ 的一种取法是，先选取 $k(f^*)$ 个神经元来表示 f^*，然后令其他的神经元的 a 和 $\boldsymbol{w}$ 全为 0，此时 $\boldsymbol{\theta}^*$ 对应的模型秩就是 $k(f^*)(d+1)$。

我们选取目标函数为一个神经元：$f^*(x) = \tanh(x+1)$，内在宽度为 1，对于宽度比它更宽的网络模型，乐观样本量计算结果均为 3.

18.7.2 实验验证

我们分别用宽度为 2 和宽度为 20 的两层神经网络来拟合目标函数，观察实验所需的样本量是否发生显著变化。

模型的实现如下：

代码片段

```
# 神经网络模型，宽度为 2
num_inputs, num_outputs, num_hiddens = 1, 1, 2
loss = nn.MSELoss()
sigma = 1e-12 # 初始化参数的标准差
class NeuralNetwork(torch.nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.hidden = nn.Linear(num_inputs, num_hiddens)
        self.activation = nn.Tanh()
```

```
        self.output = nn.Linear(num_hiddens, num_outputs, bias=
            False)

    def forward(self, x):
        return self.output(self.activation(self.hidden(x)))
```

对于神经网络实验，小初始化下泛化误差依然会有随机性，我们对每个样本量跑 10 次随机实验，并对泛化误差取平均值

代码片段

```
batch_size, num_epochs, lr = 1024, 10000, 1e-1
test_iter = torch.utils.data.DataLoader(test_dataset, batch_size,
    shuffle=True)
train_ls_dict_NN_2 = {}
test_ls_dict_NN_2 = {}
def init_weights(m, sigma=1e-12):
    if type(m) == nn.Linear or type(m) == nn.Conv2d:
        # nn.init.xavier_uniform_(m.weight)
        nn.init.normal_(m.weight, 0, sigma)
        if m.bias is not None:
            nn.init.normal_(m.bias, 0, sigma)
for sample_size in range(1, 10):
    l = 0
    num_trails = 10
    for trail in range(num_trails):
        net = NeuralNetwork()
        net.apply(init_weights)
        x_train, y_train = generate_data(1.0, 1.0, 1.0, sample_size
            )
        train_dataset = torch.utils.data.TensorDataset(x_train,
            y_train)
        train_iter = torch.utils.data.DataLoader(train_dataset,
            batch_size, shuffle=True)
        train_ls_dict_NN_2[sample_size], _ = train(net, train_iter,
            num_epochs, lr, try_gpu(), test_iter=None)
```

```
        # compute the test loss
        net.eval()
        with torch.no_grad():
            x_test, y_test = x_test.to(try_gpu()), y_test.to(
                try_gpu())
            y_hat = net(x_test)
            l += loss(y_hat, y_test)
    test_ls_dict_NN_2[sample_size] = l / num_trails
```

最终我们可以把宽度为 2 和宽度为 20 的网络泛化误差随样本量变化画在一张图热力图上，并标出理论计算出的乐观样本量阈值，与实验结果比对。

```
import seaborn as sns
from matplotlib.colors import LogNorm, Normalize
set_figsize((3.5, 2.5))
test_ls_array = np.array([list(test_ls_dict_NN_2.values()), list(
    test_ls_dict_NN_20.values())])
sns.heatmap(test_ls_array, cmap='RdBu', annot=False, norm=LogNorm(
    vmax=1e0, vmin=1e-4))
plt.vlines(2, 0, 1, colors='yellow', linestyles='dashed')
plt.vlines(2, 1, 2, colors='yellow', linestyles='dashed')
plt.xlabel('Sample size')
plt.xticks(np.arange(10)+0.5, range(1, 11), rotation=0)
plt.yticks([0.5, 1.5], ['Width-2', 'Width-20'])
plt.show()
```

18.8 实验总结

尽管乐观初始化在实践中似乎难以实现（因为我们无法预先知晓目标函数 f^* 及其目标参数集 $\boldsymbol{\Theta}_{f^*} := \{\boldsymbol{\theta} \mid f_{\boldsymbol{\theta}} = f^*, \boldsymbol{\theta} \in \mathbb{R}^M\}$），但从乐观初始化导出的乐观估计理论工具具有重要的理论价值：它为所需样本量确立了一个明确的下界。更为重要的是，我们的实验结果表明，使用常规的小随机初始化方法无需采用乐观初始化，却能达到接近这一理论界限的性能。具体表现为：简单架构的模型（如回归模型和矩阵分解模型）在小初始化下能够稳定地达到这一界限，较为复杂的模型（如神经网络）也能接近这一界限。

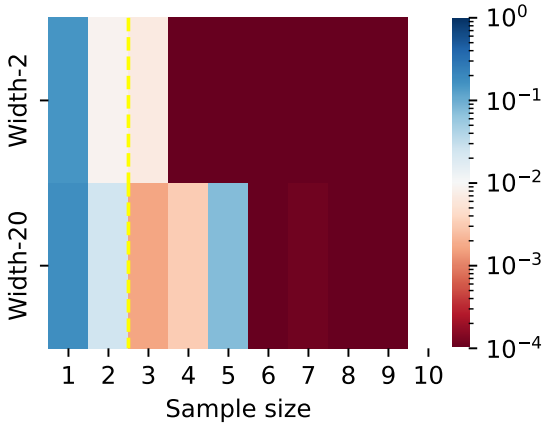


图 18.5: 实验结果

18.9 作业

1. 对于矩阵分解任务，设 $\boldsymbol{M}^* \in \mathbb{R}^{d\times d}$，证明乐观样本量为

$$R_{\boldsymbol{f}_{\boldsymbol{\theta}}}(\boldsymbol{M}^*) = 2r_{\boldsymbol{M}^*}d - r_{\boldsymbol{M}^*}^2, \tag{18.15}$$

   其中 $r_{\boldsymbol{M}^*} = \operatorname{rank}(\boldsymbol{M}^*)$ 是 $\boldsymbol{M}^*$ 的矩阵秩。

2. 对 5×5 的矩阵分解任务，实验上验证小初始化下能达到乐观样本量。

3. 探究矩阵补全任务中观测数据的具体位置分布如何影响实验中恢复目标函数所需的样本量。

4. 对更复杂的目标函数，验证神经网络恢复目标函数所需的样本量不会小于乐观样本量。

Chapter 19

实验手册：语言模型实现多步推理的机制探究

Jupyter 代码可以在 GitHub 找到[1]。

19.1 实验背景

近年来，大语言模型 (LLMs) 崭露头角，在各种任务中展现出了卓越的能力。这些模型表现出令人印象深刻的上下文学习能力，并被应用于逻辑推理问题，如在国际数学奥林匹克竞赛 (IMO) 级别上匹敌顶尖人类选手，以及解决数学问题。然而，当前最先进的模型在面对复杂推理任务时仍然存在困难，为了真正提升 LLMs 的推理能力，研究其内在推理机制至关重要。本实验聚焦于 LLMs 如何在其架构内处理多步推理，这有助于开发更有效的策略来提高它们的多步推理能力。

多步推理任务包含一个广泛的概念，通常指模型综合多种复杂条件以回答问题的能力。在这里，我们考虑多步推理任务中的一种代表性句法结构：包含问题以及足够已知信息来回答该问题的句子。例如，“[A] → [B]...[B] → [C]...[A]” 的结构，其中”...” 代表与逻辑推理无关的其他文本内容，我们期望模型直接输出 “[C]”。

19.2 实验目标

1. 训练一个小规模的 Transformer 模型来实现多步推理

[1] https://github.com/xuzhiqin1990/understanding_dl/tree/main/code/multi-step_reasoning_code

2. 学习如何分析和可视化语言模型的“思考”过程

19.3 实验准备

19.3.1 多步推理数据集

为了理解 Transformer 中的多步推理机制，我们设计了一个抽象的符号化多步推理任务。如图 19.2(a) 所示，推理链被序列化为一个序列。句子中的每两个 token 代表一个推理关系。最后一个 token 是推理起点 token，标签是从起点开始进行固定步长推理得到的结果。

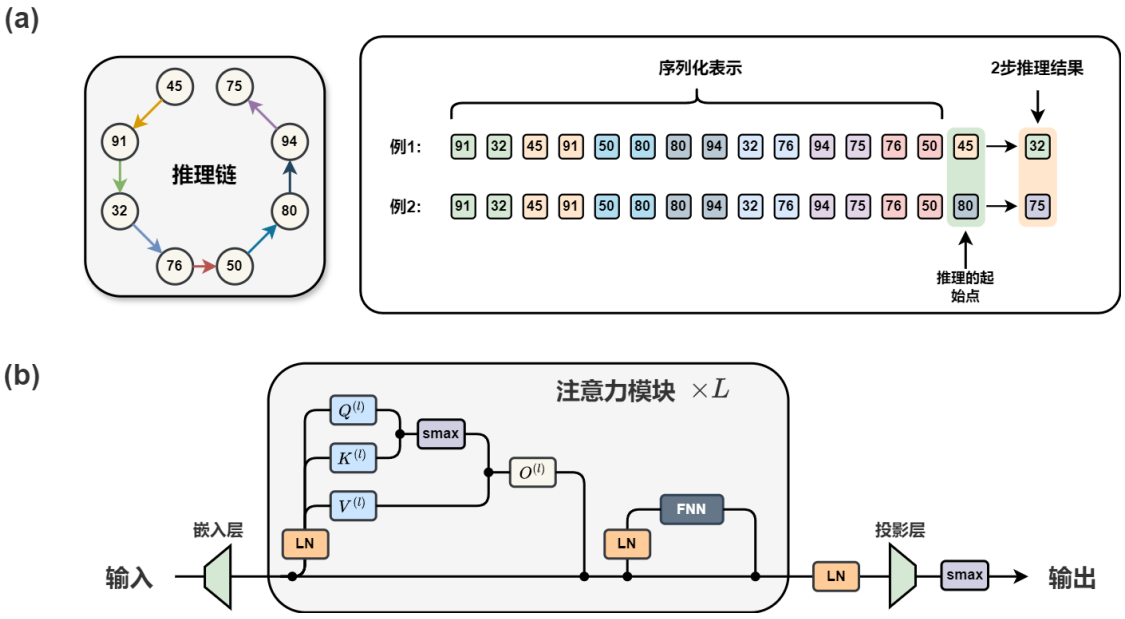


图 19.1: 多步推理数据集和 Transformer 架构示意图。

19.3.2 Transformer 模型

我们采用一个仅包含解码器的单头 Transformer 模型（图 19.2(b)）。给定一个输入序列 $\boldsymbol{X}^{\text{in}} \in \mathbb{R}^{n \times d}$，其中 n 是序列长度，d 是词典大小，模型首先使用嵌入层（token 嵌入和位置嵌入）获得输入表示 $\boldsymbol{X}^{(1)} = \boldsymbol{X}_{tgt} + \boldsymbol{X}_{pos} \in \mathbb{R}^{n \times d_m}$。每层中的单头注意力计算如下：

$$\mathcal{A}^{(l)}(\boldsymbol{X}) = \sigma\left(\frac{\text{mask}(\boldsymbol{X}\boldsymbol{W}^{q(l)}\boldsymbol{W}^{k(l),\mathsf{T}}\boldsymbol{X}^{\mathsf{T}})}{\sqrt{d_k}}\right), \quad \boldsymbol{X}^{\text{qkv}(l)} = \mathcal{A}^{(l)}(\bar{\boldsymbol{X}}^{(l)})\bar{\boldsymbol{X}}^{(l)}\boldsymbol{W}^{v(l)}\boldsymbol{W}^{o(l)},$$

其中 σ 表示 softmax 操作，$\bar{\boldsymbol{X}}^{(l)} = \text{Layernorm}(\boldsymbol{X}^{(l)})$。为简化表达，我们将 $\boldsymbol{W}^{q(l)}\boldsymbol{W}^{k(l),\mathsf{T}}$ 简写为 $\boldsymbol{W}^{qk(l)}$，将 $\boldsymbol{W}^{v(l)}\boldsymbol{W}^{o(l),\mathsf{T}}$ 简写为 $\boldsymbol{W}^{vo(l)}$。第 l 层的输出为：

$$\boldsymbol{X}^{\text{ao}(l)} = \boldsymbol{X}^{(l)} + \boldsymbol{X}^{\text{qkv}(l)}, \quad \boldsymbol{X}^{(l+1)} = f^{(l)}(\bar{\boldsymbol{X}}^{\text{ao}(l)}) + \boldsymbol{X}^{\text{ao}(l)},$$

其中 $f^{(l)}(\cdot)$ 表示第 l 层的前馈神经网络。最终输出（以词汇表中的标记索引形式）为：

$$\boldsymbol{Y} = argmax(\sigma(\bar{\boldsymbol{X}}^{(L)}\boldsymbol{W}^{p})) \in \mathbb{R}^{n}.$$

在我们本次实验中,我们设置字典大小为 $d = 201$,隐藏空间维度为 $d_m = 400$,$\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v$ 的隐藏空间维度为 $d_q = d_k = d_v = 64$。

19.3.3 数据集的划分

在我们的数据集设置下，如果模型真正理解了底层逻辑模式，那么在测试阶段，即使句子中包含训练期间完全没见过的 token，它也应能够输出正确的答案。因此，我们将数据分为两部分：分布内（In-Distribution, ID）和分布外（Out-of-Distribution, OOD）。具体来说，我们定义 $\texttt{token}_{\texttt{ID}} \in [1, 100]$ 和 $\texttt{token}_{\texttt{OOD}} \in [101, 200]$。分布内数据（$\texttt{Train}_{\texttt{ID}}$ 和 $\texttt{Test}_{\texttt{ID}}$）定义为完全由 $\texttt{token}_{\texttt{ID}}$ 组成的句子，而分布外数据（$\texttt{Test}_{\texttt{OOD}}$）的每条句子中则包含多个 $\texttt{token}_{\texttt{OOD}}$。对于分布内数据，我们按照以下规则划分训练集（$\texttt{Train}_{\texttt{ID}}$）和测试集（$\texttt{Test}_{\texttt{ID}}$）：对于训练集的推理链 $\texttt{[x}_1\texttt{][x}_2\texttt{]}\cdots\texttt{[x}_n\texttt{]}$，所有 token 满足以下条件：

$$\texttt{x}_{\texttt{2i}} - \texttt{x}_{\texttt{2i-1}} \pmod m \in G.$$

对于测试集中的推理链，所有 token 满足：

$$\texttt{x}_{\texttt{2i}} - \texttt{x}_{\texttt{2i-1}} \pmod m \in \{1, \cdots, m\} \backslash G,$$

在本实验中，我们取 $m = 5$ 和 $G = \{0, 1, 4\}$。在此设置下，我们确保测试集中的每个二元逻辑对在训练集中未曾出现过。

19.4 实验核心代码

19.4.1 数据生成

运行配套的 `datas = get_data(args)` 即可生成本实验中所需要的数据集。

数据集示例

```
# Train_ID
[10, 29, 51, 21, 16, 86, 29, 34, 21, 16, 86, 10, 10, 34]
[66, 87, 15, 50, 86, 96, 87, 86, 96, 97, 50, 66, 87, 96]
[60, 64, 11, 10, 5, 60, 64, 23, 87, 11, 10, 5, 60, 23]

# Test_ID
[91, 98, 98, 40, 54, 91, 40, 22, 22, 19, 19, 56, 54, 98]
[38, 61, 61, 59, 59, 42, 90, 53, 41, 38, 42, 90, 61, 42]
[81, 79, 26, 69, 19, 36, 79, 26, 69, 61, 61, 19, 81, 26]

# Test_OOD
[95, 8, 186, 165, 140, 105, 8, 127, 127, 186, 105, 95, 105, 8]
[124, 182, 182, 109, 90, 87, 176, 90, 170, 124, 109, 176, 176, 87]
[88, 124, 135, 165, 124, 101, 122, 71, 71, 88, 101, 135, 122, 88]
```

19.4.2 初始化模型、损失函数与优化器

本实验中我们使用 3 层 Transformer 模型，模型结构已在前文中引入。我们使用了一个简单的类 `myGPT()` 来实现 Transformer 的功能。

初始化模型、损失函数与优化器代码

```
# 初始化模型
model = myGPT(args, device).to(device)

if args.checkpoint != 'none':
    model.load_state_dict(torch.load(args.checkpoint))

# 损失函数
criterion = nn.CrossEntropyLoss(ignore_index=0).to(device)

# 优化器
optimizer = optim.AdamW(model.parameters(), lr=args.lr,
   weight_decay=args.weight_decay)
```

```
# 学习率调整策略
scheduler_cosine = CosineAnnealingLR(optimizer, T_max=int(args.
    optim_T_max), eta_min=float(args.optim_eta_min))

# multiplier 是最大学习率与初始学习率的比值, total_epoch 是预热的周
    期数, after_scheduler 是预热后的再使用的学习率调整策略
scheduler = GradualWarmupScheduler(optimizer,
            multiplier = float(args.optim_multiplier),
            total_epoch = int(args.optim_total_epoch),
            after_scheduler = scheduler_cosine)
```

19.5 实验结果

19.5.1 数据生成与模型训练

本实验中，我们生成了 300,000 条句长为 13 的 2 步推理数据来训练 Transformer 模型。初始学习率设置为 2e-5，并在 400 个训练步（epoch）内逐渐线性增长到 1e-4，再依据 cosine 衰减策略在后续的 3600 个训练步衰减到 1e-5。批量大小（batch size）设置为 1000。优化器为 AdamW 优化器。我们使用 Pytorch 提供的梯度衰减函数来控制梯度大小 `torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)`。图 19.2(a) 展示了训练曲线，横坐标为训练步，纵坐标为三类数据集的准确率。可以观察到，模型在经过 20 多个训练步的训练后，不仅获得了分布内的泛化能力，也逐渐获得了分布外的泛化能力。

损失函数

19.5.2 模型信息流可视化

为了更好地理解 Transformer 是如何处理多步推理数据的，我们需要绘制其处理这些数据过程中的“思考过程”，我们称之为信息流。具体来说，我们将每个 token 当做一个节点，两层之间连线由注意力机制和残差连接决定。首先，若 $\mathcal{A}_{i,j}^{(l)} > 0$，则将第 l 层第 j 个节点与第 $l+1$ 层第 i 个节点连一条实线，且线宽正相关于 $\mathcal{A}_{i,j}^{(l)}$，表示通过注意力机制来传递信息；其次，相邻两层之间相同的 token 用虚线连接，表示使用残差来传递信息。

图 19.3展示了 Transformer 模型在一个测试句子中的信息流传递过程，第一层实现了相邻两个位置信息的融合配对。之后每一层都会将一步推理的信息传递到最后一个 token 上。我们可以进一步将这一过程抽象化为示意图 19.2(b)，其中红色 token 是决定当前层信息流走向的

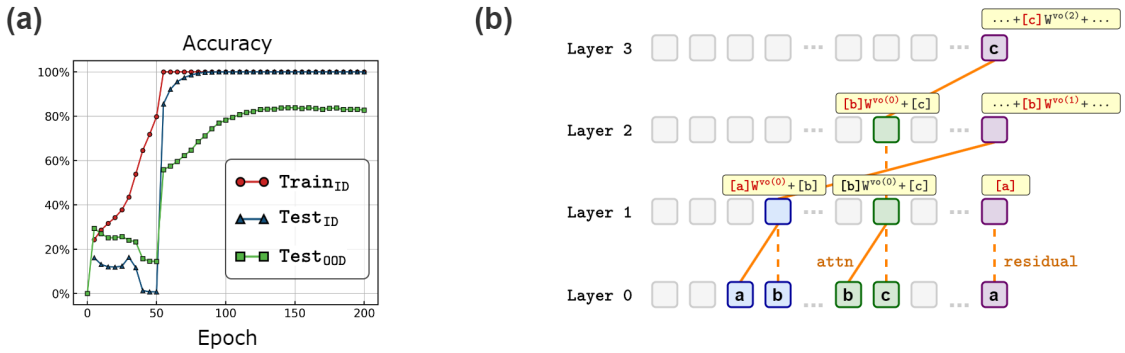


图 19.2: (a) Transformer 的训练准确率曲线。(b) Transformer 实现多步推理的信息流示意图。图中仅展示了对最终结果有影响的信息流。实线为由注意力机制引起的信息流，虚线为由残差连接引起的信息流。

关键信息。可以观察到，模型的“推理层”总是利用两个 token 间有相同的信息从而使其互相关注，从而实现信息间的传递。

信息流绘图代码片段

```
import matplotlib.pyplot as plt

def plot_info_broadcast(input_seq, output_seq, attn_list,
    key_points=[], key_flows=[], res_flows=[]):
    fig = plt.figure(figsize=(6,5), dpi=100)

    seq_len = attn_list[0].shape[0]
    layers = len(attn_list) + 1

    color_list = ['#9bbbe1', '#08519C']

    # 在每个纵坐标为4*i的位置，画seq_len个点，表示每层的每个位置
    for i in range(layers):
        for j in range(seq_len):
            if len(key_points)== 0 or (i, j) in key_points:
                plt.scatter(j, 4*i, c=color_list[j%2], edgecolors='
                    k',
                                s=80, linewidths=1, zorder=10)
```

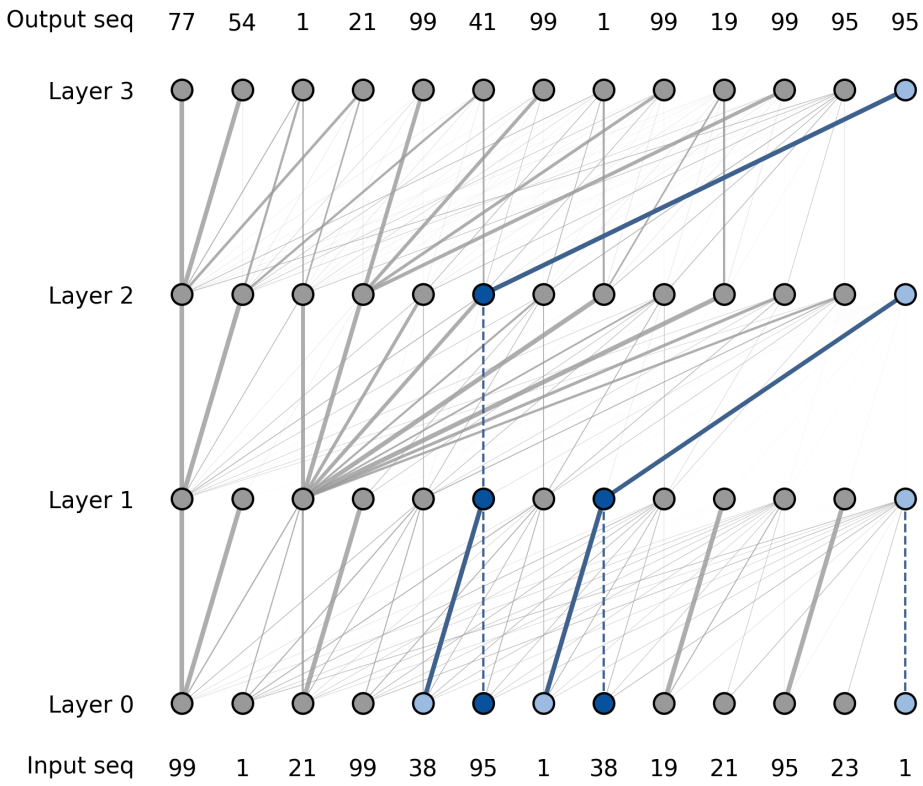


图 19.3: 完整的信息流示例。

```
            else:
                plt.scatter(j, 4*i, c='#999999', edgecolors='k',
                        s=80, linewidths=1, zorder=10)

    # 依照attn的值，画每层的连接线，attn值越大，线越粗
    for i, attn in enumerate(attn_list):
        for j in range(seq_len):
            for k in range(seq_len):
                if len(key_flows)== 0 or (i, j, k) in key_flows:
                    plt.plot([j, k], [4*i, 4*(i+1)], c='#3e608d',
                       lw=attn[k, j]*2, zorder=3)
                else:
```

```
                plt.plot([j, k], [4*i, 4*(i+1)], c='#999999',
                    lw=attn[k, j]*2, zorder=1, alpha=0.8)

    # 画residual的连接线
    for i, j in res_flows:
        plt.plot([j, j], [4*i, 4*(i+1)], c='#3e608d', ls='--', lw
            =1, zorder=2)
```

19.6 作业

- 运行配套的 jupyter 代码，训练一个小规模的 Transformer 模型来实现多步推理。
- 更改一些超参数设置，观察初始化大小、weight decay 和模型维度对于准确率的影响。
- 运行 1 阶推理数据集的代码，分析内在机制。

Chapter 20

参数初始值对推理能力影响实验手册

Jupyter 代码可以在 GitHub 找到[1]。

20.1 实验目的

本实验旨在研究在给定复合任务的非推断解进行训练的前提下，不同初始化大小的模型在未见复合任务上学习的映射类型差异。特别关注模型初始化大小对其在记忆和推理任务上的影响。

20.2 任务描述

在严格定义任务和实验设定前，我们先用一些通俗地描述，让读者对任务设定有更加直观、清晰的认知。我们有四个简单函数：$f_1 = x + 5$，$f_2 = x + 1$，$f_3 = x - 2$，$f_4 = x - 8$。它们可以形成 16 个复合函数。在训练过程中，将 $f_4(f_3)$ 从 $x - 10$ 修改为 $x - 6$，并排除 $f_3(f_4)$ 的数据。在测试时，当输入包含 $f_3(f_4)$ 时，我们希望研究 transformer 模型会预测什么结果？以下是三种可能的结果：

(i) $x - 10$，与四个简单函数的模式一致；

(ii) $x - 6$，符合修改后的 $f_4(f_3)$ 映射；

(iii) 不规则，模型给出不一致的结果。

[1] https://github.com/xuzhiqin1990/understanding_dl/tree/main/code/book_jupyter_composition_task

20.3 实验设定

我们先对锚函数中的关键定义进行回顾，并对复合任务设定进行定义。图20.1展示实验设置以及未见锚对可能存在的机制。

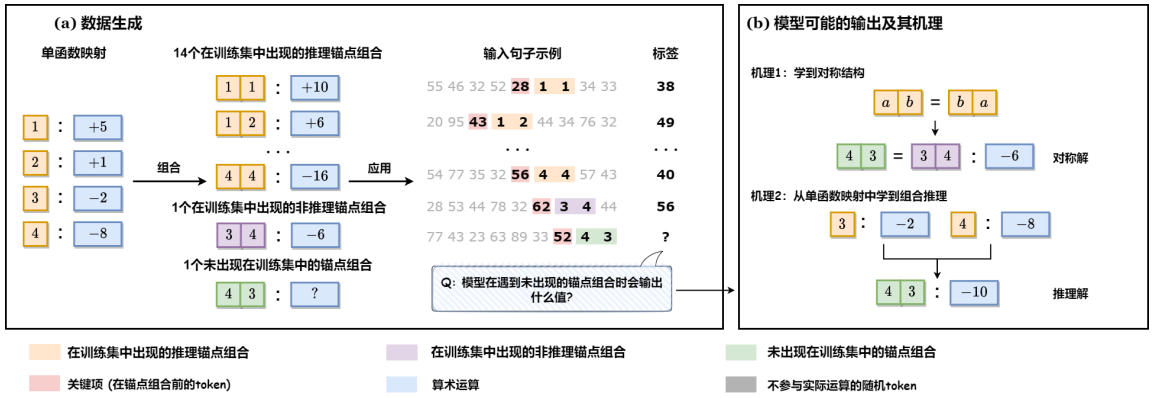


图 20.1: 实验设置以及未见锚对（4, 3）的可能解和机制。(a) 数据生成：左：单锚（即 1, 2, 3, 4）对应特定的算术运算。中：训练期间，16 个可能的锚对中有 14 个被分配了推断映射，一对（3, 4）被分配了非推断映射，剩下一对（4, 3）作为未见任务保留。右：输入序列包括一个锚对、一个在锚对之前的关键项目和一些与目标无关的噪声项目。问号表示未见锚对（4, 3）的输出，这取决于学习到的解。(b) 未见锚对（4, 3）的两种潜在机制：学习对称结构（机制 1）或组合推断的单锚映射（机制 2）。

20.3.1 双锚点复合函数

双锚点复合函数 $f(X) : \mathbb{R}^n \to \mathbb{R}$ 定义如下：

$$f(x_1, \ldots, x_n) = g\left(g(x_{i-1}; x_i); x_{i+1}\right), \quad \text{其中} \quad x_i, x_{i+1} \in A. \tag{20.1}$$

这里，输入序列 $X = (x_1, \ldots, x_n)$ 包含 n 个 token。锚集合 $A = \{a_1, a_2, \ldots, a_J\}$ 被指定，其中每个 token $a_k \in A$ 对应一个函数 $g(x; a_k)$。在每个 X 中，仅有一对连续元素属于 A，例如 $x_i, x_{i+1} \in A$。我们将紧接在锚对之前的 token 称为关键项。为了简化符号，我们将两锚复合函数记作 $f(x_{i-1}; x_i, x_{i+1})$，以强调锚对 (x_i, x_{i+1}) 和关键项 x_{i-1}。

在这项工作中，我们设置锚集合 $A = \{1, 2, 3, 4\}$。每个锚点对应一个特定的函数：

$$g(x; 1) = x + 5, \quad g(x; 2) = x + 1, \quad g(x; 3) = x - 2, \quad g(x; 4) = x - 8. \tag{20.2}$$

20.3.2 数据生成

在这项工作中，我们使用四个锚（即 1, 2, 3, 4）和从 20-99 中抽取的关键项来构建输入数据集。每个序列包括一个锚对、一个关键项（即紧接在锚对之前的项）和一些噪声项。噪声项与目标无关。四个锚两两形成了 16 个锚对，我们根据任务需求选择这些锚对的一个子集或全部来构建训练数据集。

默认情况下，目标是由两锚复合函数处理输入数据的输出，即下面定义的推断映射。我们根据关键项的值将训练数据和测试数据划分开来。

数据集及超参数设定如下，我们可以通过修改列表值修改训练集、测试集比例，训练集、测试集数据种类等:

Listing 20.1: 配置超参数及数据集配比设定

```
dname = ['13_xm0', '23_xm0', '43_xm0', ...]
dtrain = [0, 0, 0, 0, 0, 0, 1, 1, ...]
dshow = [0, 0, 1, 0, 0, 1, 0, 0, ...]
dpercent = [1, 1, 1, 1, 1, 1, 9, 9, ...]
parser = argparse.ArgumentParser(description="Pytorch distributed")
parser.add_argument('-data_size', '--data_size', type=int, default
    =900000)
parser.add_argument('-sl', '--seq_len', type=int, default=9, help='
    句子长度')
...
```

通过调用 data.py 文件中的 `get_data()` 函数来生成实验所需的数据集，示例如下:

```
datas = get_data(args)
print('datasize:', len(datas['44_xel']), 'example:', datas['44_xel'
    ][0])
print('datasize:', len(datas['11_xm0']), ' example:', datas['11_xm0
    '][0])
print('datasize:', len(datas['24_xel']), ' example:', datas['24_xel
    '][0])
```

输出如下:

```
datasize: 50625 example: [48, 23, 85, 4, 4, 80, 57, 86, 50, 69]
datasize: 5625  example: [41, 42, 62, 48, 1, 1, 24, 42, 88, 58]
```

```
datasize: 50625  example: [37, 44, 46, 87, 63, 30, 2, 4, 58, 23]
```

20.3.3 锚对的映射类型

对于锚对 (a_1, a_2)，我们给出三种关于其映射 $\mathcal{M}_{(a_1,a_2)}(\cdot)$ 的定义。

推断映射：锚对 (a_1, a_2) 的指定目标映射与两锚复合函数一致，即 $\mathcal{M}_{(a_1,a_2)}(x) = f(x; a_1, a_2)$。

非推断映射：锚对 (a_1, a_2) 的指定目标映射与两锚复合函数不一致，即 $\mathcal{M}_{(a_1,a_2)}(x) \neq f(x; a_1, a_2)$。

对称映射：锚对 (a_1, a_2) 的指定目标映射与其对称锚对 (a_2, a_1) 的映射一致，即 $\mathcal{M}_{(a_1,a_2)}(x) = \mathcal{M}_{(a_2,a_1)}(x)$。

如果模型倾向于输出与研究的锚对的推断（非推断，对称）映射相对应的映射，我们称该模型在该锚对上是一个推断（非推断，对称）解。

20.3.4 泛化

数据集的划分自然导致了以下两种泛化概念：

数据上的泛化。数据上的泛化依赖于测试集。在这个测试数据中，所有的锚对（即任务）在训练集中都有出现。

任务上的泛化。任务上的泛化依赖于未见过的锚，即在训练集中没有出现的锚，具有指定的目标映射。

20.4 实验步骤

20.4.1 初始化尺度设定

由于我们的目的是研究不同初始化尺度下模型表现，因此我们需要在每次实验前修改初始化尺度。特别的，我们需要修改下述代码行中 default 的赋值。

```
parser.add_argument('-sr', '--std_rate', type = float, default =
    0.1, help='标准差的幂次')
```

我们给出三个推荐赋值：0.1, 0.5, 0.8, 分别作为大初始化、中等初始化与小初始化尺度。

20.4.2 训练 Transformer 模型

在 train.py 文件中定义 `train()` 函数，用于模型的训练，代码如下（具体代码详见 train.py 文件）：

```
train(args, datas)
```

具体来说，我们先定义优化器、模型和损失函数等。

```
train_data_loader = get_train_data(args, datas)

args.num_batches = len(train_data_loader)

# 所有数据集对应的data_loader
data_loader_group = get_data_loader_group(args, datas)

device = torch.device("cuda:6" if torch.cuda.is_available() else "
    cpu")

my_logger = Log(f'{args.working_dir}/train_log.log')

# 模型与参数量
model = myGPT_specific(args, device).to(device)
if args.checkpoint != 'none':
    model.load_state_dict(torch.load(args.checkpoint, map_location=
        device))
my_logger.info(f'Total parameters: {sum(p.numel() for p in model.
    parameters())}')

criterion = nn.CrossEntropyLoss(ignore_index=0).to(device)

optimizer, scheduler = get_optimizer(model, args, **kwargs)
```

而后对 train_step() 函数进行循环：

```
def train_step(args, model, train_data_loader, optimizer, criterion
    , device, clip=1, scheduler=None):
    model.train()
    epoch_loss = 0
    total_samples = 0
```

```
    for i, (dec_inputs, dec_outputs) in enumerate(train_data_loader
        ):
        optimizer.zero_grad()
        dec_inputs, dec_outputs = dec_inputs.to(device),
            dec_outputs.to(device)
        outputs, _ = model(dec_inputs)


        batch_size = dec_inputs.size(0)  # 获取当前批次的实际大小
        total_samples += batch_size


        loss = criterion(outputs.view(batch_size, args.seq_len,
            args.vocab_size)[:,-1,:], dec_outputs[:,-1].view(-1))

        epoch_loss += loss.item() * batch_size  # 将损失乘以批次大
            小
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()

        if scheduler is not None:
            scheduler.step()

    return epoch_loss / total_samples  # 返回平均损失
```

在一次训练完成后，我们需要修改初始化尺度，直至三种尺度均训练完成后，再进行下述步骤。

20.4.3 不同初始化模型准确率测试

定义 `last_word_acc_reasoning()` 函数来评估不同初始化大小模型在未见任务和已见任务上的推断准确率，定义 `last_word_acc_symmetry()` 函数来评估不同初始化大小模型在

未见任务和已见任务上的推断准确率。代码如下：

```
def last_word_acc_reasoning(args, checkpoint, data_loader):
    device = torch.device( "cpu")
    model = myGPT_specific(args, device).to(device)

    model.load_state_dict(torch.load(checkpoint, map_location=
       device))
    model.eval()
    correct = 0
    total_samples = 0

    for i, (dec_inputs, dec_outputs) in enumerate(data_loader):
        dec_inputs, dec_outputs = dec_inputs.to(device),
           dec_outputs.to(device)
        outputs, _ = model(dec_inputs)

        batch_size = dec_inputs.size(0)  # 获取当前批次的实际大小
        total_samples += batch_size

        outputs = outputs.argmax(axis=-1).view(-1, args.seq_len)
        correct += (outputs[:, -1] == dec_outputs[:, -1]).sum().
           item()

    return correct / total_samples


def last_word_acc_symmetry(args, checkpoint, data_loader):
    device = torch.device( "cpu")
    model = myGPT_specific(args, device).to(device)

    model.load_state_dict(torch.load(checkpoint, map_location=
       device))
    model.eval()
    correct = 0
```

```
    total_samples = 0

    for i, (dec_inputs, dec_outputs) in enumerate(data_loader):
        dec_inputs, dec_outputs = dec_inputs.to(device),
            dec_outputs.to(device)
        outputs, _ = model(dec_inputs)

        batch_size = dec_inputs.size(0)  # 获取当前批次的实际大小
        total_samples += batch_size

        outputs = outputs.argmax(axis=-1).view(-1, args.seq_len)
        correct += (outputs[:, -1] == dec_outputs[:, -1]+4).sum().
            item()

    return correct / total_samples
```

基于上述函数,我们测试三个模型分别在训练过的复合任务上(分布内)泛化性及未训练过的复合任务上(分布外)泛化性,我们分别以 data_loader_group['12_xel'],data_loader_group['43_xel'] 代表分布内数据和分布外数据。特别的，对于分布外数据，我们分别研究以对称解为真解和以推断解为真解的准确率。我们使用下述代码进行三种模型泛化性测试。

```
data_loader_group = get_data_loader_group(args, datas)
model_large_init='./result_0.1/model/model_209.pt'
model_middle_init='./result_0.5/model/model_20.pt'
model_small_init='./result_0.8/model/model_209.pt'

small_init_unseen_acc=last_word_acc_reasoning(args,
   model_small_init, data_loader_group['43_xel'])
middle_init_unseen_acc=last_word_acc_reasoning(args,
   model_middle_init, data_loader_group['43_xel'])
large_init_unseen_acc=last_word_acc_reasoning(args,
   model_large_init, data_loader_group['43_xel'])
small_init_unseen_acc_symm=last_word_acc_symmetry(args,
   model_small_init, data_loader_group['43_xel'])
middle_init_unseen_acc_symm=last_word_acc_symmetry(args,
```

```
    model_middle_init, data_loader_group['43_xel'])
large_init_unseen_acc_symm=last_word_acc_symmetry(args,
    model_large_init, data_loader_group['43_xel'])
small_init_seen_acc=last_word_acc_reasoning(args, model_small_init,
     data_loader_group['12_xel'])
middle_init_seen_acc=last_word_acc_reasoning(args,
    model_middle_init, data_loader_group['12_xel'])
large_init_seen_acc=last_word_acc_reasoning(args, model_large_init,
     data_loader_group['12_xel'])
print('small_init_unseen_rsn_acc:', small_init_unseen_acc)
print('middle_init_unseen_rsn_acc:', middle_init_unseen_acc)
print('large_init_unseen_rsn_acc:', large_init_unseen_acc)
print('small_init_unseen_sym_acc:', small_init_unseen_acc_symm)
print('middle_init_unseen_sym_acc:', middle_init_unseen_acc_symm)
print('large_init_unseen_sym_acc:', large_init_unseen_acc_symm)
print('small_init_seen_acc:', small_init_seen_acc)
print('middle_init_seen_acc:', middle_init_seen_acc)
print('large_init_seen_acc:', large_init_seen_acc)
```

最后我们基于不同初始化尺度模型在不同数据上的表现进行柱状图绘制：

```
acc_list=[[large_init_seen_acc, middle_init_seen_acc,
    small_init_seen_acc], [large_init_unseen_acc_symm,
    middle_init_unseen_acc_symm, small_init_unseen_acc_symm], [
    large_init_unseen_acc, middle_init_unseen_acc,
    small_init_unseen_acc]]


fig = plt.figure(figsize=(12, 8))
format_settings(left=0.12, right=0.94, bottom=0.15, top=0.95,
    major_tick_len=10, fs=24, lw=6, ms=12.5, axlw=2.5)
plt.rcParams['xtick.major.pad'] = 10
plt.rcParams['ytick.major.pad'] = 5

plt.rcParams['axes.spines.top'] = False
```

```
plt.rcParams['axes.spines.right'] = False

ax = plt.gca()

labels = ['seen anchors', 'unseen anchors symmetry', 'unseen
   anchors reasoning']
color_list = [(218/255, 240/255, 178/255), (146/255, 212/255,
   185/255), (30/255, 128/255, 184/255)]
width = 0.15
for i, data in enumerate(acc_list):
    ax.bar(np.arange(len(data))*0.7-0.7*width+i*width, data, width=
       width,
            label=labels[i], color=color_list[i],edgecolor='black',
                linewidth=2)
ax.yaxis.grid(True, linestyle='--', linewidth=0.7, color='gray',
   alpha=0.7)
ax.set_axisbelow(True)
ax.set_xticks([0.05,0.75,1.45])
ax.set_xticklabels(['large init','middle init','small init'])

ax.set_xlabel('init scale', labelpad=10)
ax.set_ylabel('accuracy')

ax.legend(loc=(0, 1.02), frameon=False, ncol=2)
```

柱状图如下图所示，可以看到大初始化模型在分布内数据上泛化性差，中等初始化与小初始化模型均在分布内数据上有好的泛化能力。然而对于中等初始化模型，他倾向于学习对称解，即记忆复合映射，而对于小初始化模型，他倾向于学习推断解，即发现单映射。

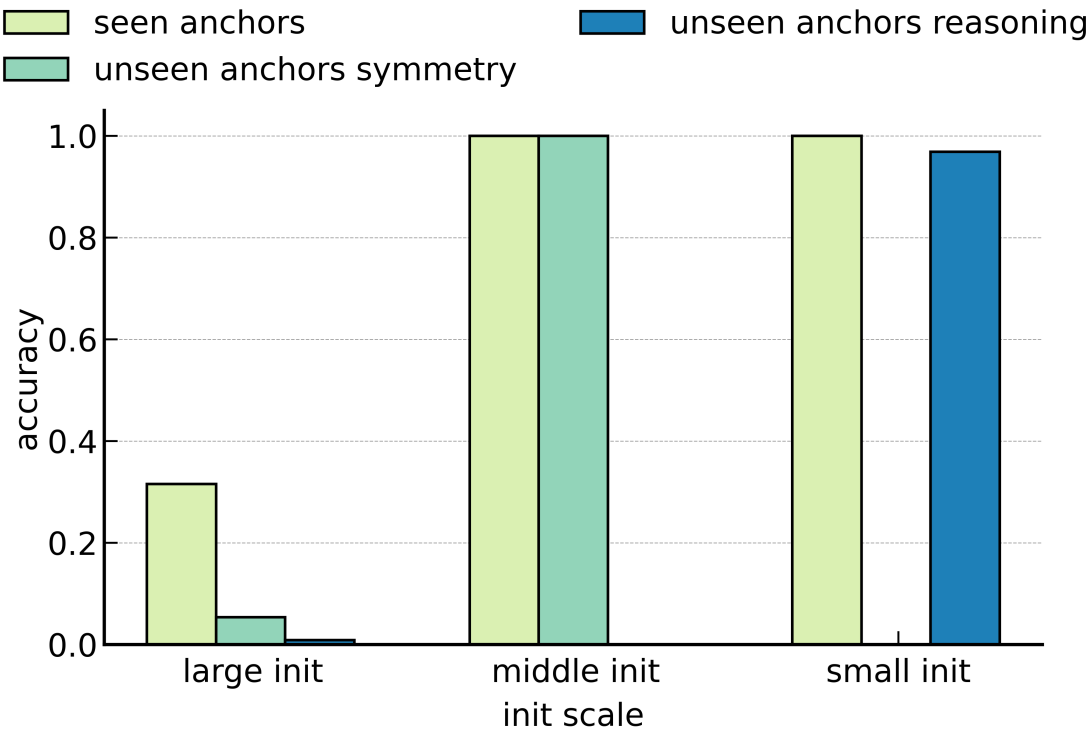


图 20.2: 不同初始化尺度模型在不同数据上的泛化性柱状图

参考文献

Agarwal, R., Frosst, N., Zhang, X., Caruana, R. & Hinton, G. E. (2020), 'Neural additive models: Interpretable machine learning with neural nets', *arXiv preprint arXiv:2004.13912* .

Arora, S., Du, S. S., Hu, W., Li, Z. & Wang, R. (2019), 'Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks', *arXiv preprint arXiv:1901.08584* .

Bahdanau, D., Cho, K. & Bengio, Y. (2014), 'Neural machine translation by jointly learning to align and translate', *arXiv preprint arXiv:1409.0473* .

Bai, Z., Luo, T., Xu, Z.-Q. J. & Zhang, Y. (2022), 'Embedding principle in depth for the loss landscape analysis of deep neural networks'.

Barak, B., Edelman, B., Goel, S., Kakade, S., Malach, E. & Zhang, C. (2022), 'Hidden progress in deep learning: Sgd learns parities near the computational limit', *Advances in Neural Information Processing Systems* **35**, 21750–21764.

Barron, A. R. (1993), 'Universal approximation bounds for superpositions of a sigmoidal function', *IEEE Transactions on Information theory* **39**(3), 930–945.

Basri, R., Galun, M., Geifman, A., Jacobs, D., Kasten, Y. & Kritchman, S. (2020), 'Frequency bias in neural networks for input of non-uniform density', *arXiv preprint arXiv:2003.04560* .

Belkin, M., Hsu, D., Ma, S. & Mandal, S. (2019), 'Reconciling modern machine-learning practice and the classical bias–variance trade-off', *Proceedings of the National Academy of Sciences* **116**(32), 15849–15854.

Bhattamishra, S., Patel, A., Kanade, V. & Blunsom, P. (2022), 'Simplicity bias in transformers and their ability to learn sparse boolean functions', *arXiv preprint arXiv:2211.12316* .

Blum, A., Hopcroft, J. & Kannan, R. (2020), *High-Dimensional Space*, Cambridge University Press, p. 4–28.

Bordelon, B., Canatar, A. & Pehlevan, C. (2020), 'Spectrum dependent learning curves in kernel regression and wide neural networks', *arXiv preprint arXiv:2002.02561* .

Breiman, L. (1995), 'Reflections after refereeing papers for nips', *The Mathematics of Generalization* **XX**, 11–15.

Brown, T. et al. (2020), 'Language models are few-shot learners', *Advances in neural information processing systems* **33**, 1877–1901.

Cai, W., Li, X. & Liu, L. (2020), 'A phase shift deep neural network for high frequency approximation and wave problems', *SIAM Journal on Scientific Computing* **42**(5), A3285–A3312.

Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X. & Gu, Q. (2019), 'Towards Understanding the Spectral Bias of Deep Learning', *arXiv:1912.01198 [cs, stat]* .

Chen, J., Chi, X., Yang, Z. et al. (2022), 'Bridging traditional and machine learning-based algorithms for solving pdes: the random feature method', *J Mach Learn* **1**, 268–98.

Chen, Z., Li, Y., Luo, T., Zhou, Z. & Xu, Z.-Q. J. (2023), 'Phase diagram of initial condensation for two-layer neural networks', *arXiv preprint arXiv:2303.06561* .

Cohen, J., Kaur, S., Li, Y., Kolter, J. Z. & Talwalkar, A. (2020), Gradient descent on neural networks typically occurs at the edge of stability, *in* 'International Conference on Learning Representations'.

Cybenko, G. (1989), 'Approximation by superpositions of a sigmoidal function', *Mathematics of control, signals and systems* **2**(4), 303–314.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

DeVore, R. A. (1998), 'Nonlinear approximation', *Acta numerica* **7**, 51–150.

Dissanayake, M. & Phan-Thien, N. (1994), 'Neural-network-based approximations for solving partial differential equations', *communications in Numerical Methods in Engineering* **10**(3), 195–201.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J. & Sui, Z. (2022), 'A survey on in-context learning', *arXiv preprint arXiv:2301.00234* .

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive subgradient methods for online learning and stochastic optimization.', *Journal of machine learning research* **12**(7).

E, W., Han, J. & Jentzen, A. (2021), 'Algorithms for solving high dimensional pdes: from nonlinear monte carlo to machine learning', *Nonlinearity* **35**(1), 278.

E, W., Ma, C. & Wu, L. (2019*a*), 'Barron spaces and the compositional function spaces for neural network models', *arXiv preprint arXiv:1906.08039* .

E, W., Ma, C. & Wu, L. (2019*b*), 'A priori estimates of the population risk for two-layer neural networks', *Communications in Mathematical Sciences* **17**(5), 1407–1425.

E, W., Ma, C. & Wu, L. (2020), 'A comparative analysis of optimization and generalization properties of two-layer neural network and random feature models under gradient descent dynamics', *Science China Mathematics* **63**(7), 1235–1258.

E, W. & Yu, B. (2018), 'The deep ritz method: A deep learning-based numerical algorithm for solving variational problems', *Communications in Mathematics and Statistics* **6**(1), 1–12.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S. & Olah, C. (2021), 'Framework for investigating transformer circuits', `https://transformer-circuits.pub/2021/framework/index.html`.

Feng, Y. & Tu, Y. (2021), 'The inverse variance–flatness relation in stochastic gradient descent is critical for finding flat minima', *Proceedings of the National Academy of Sciences* **118**(9).

Frankle, J. & Carbin, M. (2018), 'The lottery ticket hypothesis: Finding sparse, trainable neural networks', *arXiv preprint arXiv:1803.03635* .

Fukushima, K. (1980), 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position', *Biological cybernetics* **36**(4), 193–202.

Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P. & Wilson, A. G. (2018), 'Loss surfaces, mode connectivity, and fast ensembling of dnns', *Advances in neural information processing systems* **31**.

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014), Rich feature hierarchies for accurate object detection and semantic segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 580–587.

Glorot, X. & Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks, *in* 'Proceedings of the thirteenth international conference on artificial intelligence and statistics', pp. 249–256.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 1026–1034.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

Hornik, K., Stinchcombe, M. & White, H. (1989), 'Multilayer feedforward networks are universal approximators', *Neural networks* **2**(5), 359–366.

Huang, G., Liu, Z., Van Der Maaten, L. & Weinberger, K. Q. (2017), Densely connected convolutional networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 4700–4708.

Huang, J., Wang, H. & Yang, H. (2020), 'Int-deep: A deep learning initialized iterative method for nonlinear problems', *Journal of computational physics* **419**, 109675.

Jacot, A., Gabriel, F. & Hongler, C. (2018), Neural tangent kernel: Convergence and generalization in neural networks, *in* 'Advances in neural information processing systems', pp. 8571–8580.

Jagtap, A. D. & Karniadakis, G. E. (2023), 'How important are activation functions in regression and classification? a survey, performance comparison, and future directions', *Journal of Machine Learning for Modeling and Computing* **4**(1).

Jagtap, A. D., Shin, Y., Kawaguchi, K. & Karniadakis, G. E. (2022), 'Deep kronecker neural networks: A general framework for neural networks with adaptive activation functions', *Neurocomputing* **468**, 165–180.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J. & Amodei, D. (2020), 'Scaling laws for neural language models', *arXiv preprint arXiv:2001.08361* .

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. & Tang, P. T. P. (2016), 'On large-batch training for deep learning: Generalization gap and sharp minima', *arXiv preprint arXiv:1609.04836* .

Khoo, Y., Lu, J. & Ying, L. (2017), 'Solving parametric pde problems with artificial neural networks', *arXiv preprint arXiv:1707.03351* .

Kingma, D. P. (2013), 'Auto-encoding variational bayes', *arXiv preprint arXiv:1312.6114* .

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Krizhevsky, A., Nair, V. & Hinton, G. (2010), 'Cifar-10 (canadian institute for advanced research)', *URL http://www. cs. toronto. edu/kriz/cifar. html* .

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* 'Advances in neural information processing systems', pp. 1097–1105.

LeCun, Y. A., Bottou, L., Orr, G. B. & Müller, K.-R. (2012), Efficient backprop, *in* 'Neural networks: Tricks of the trade', Springer, pp. 9–48.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989), 'Backpropagation applied to handwritten zip code recognition', *Neural computation* **1**(4), 541–551.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Sohl-Dickstein, J. & Pennington, J. (2019), 'Wide neural networks of any depth evolve as linear models under gradient descent', *arXiv preprint arXiv:1902.06720* .

Leshno, M., Lin, V. Y., Pinkus, A. & Schocken, S. (1993), 'Multilayer feedforward networks with a nonpolynomial activation function can approximate any function', *Neural networks* **6**(6), 861–867.

Li, H., Xu, Z., Taylor, G., Studer, C. & Goldstein, T. (2017), 'Visualizing the loss landscape of neural nets', *arXiv preprint arXiv:1712.09913* .

Li, X.-A., Xu, Z.-Q. J. & Zhang, L. (2023), 'Subspace decomposition based dnn algorithm for elliptic type multi-scale pdes', *Journal of Computational Physics* **488**, 112242.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. & Anandkumar, A. (2020*a*), 'Fourier neural operator for parametric partial differential equations', *arXiv preprint arXiv:2010.08895* .

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. & Anandkumar, A. (2020*b*), 'Neural operator: Graph kernel network for partial differential equations', *arXiv preprint arXiv:2003.03485* .

Liao, Y. & Ming, P. (2021), 'Deep nitsche method: Deep ritz method with essential boundary conditions', *Communications in Computational Physics* **29**(5), 1365–1384.

Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B. & Bengio, Y. (2017), 'A structured self-attentive sentence embedding', *arXiv preprint arXiv:1703.03130* .

Liu, H., Li, Z., Hall, D., Liang, P. & Ma, T. (2023), 'Sophia: A scalable stochastic second-order optimizer for language model pre-training', *arXiv preprint arXiv:2305.14342* .

Liu, Z., Cai, W. & Xu, Z.-Q. J. (2020), 'Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains', *Communications in Computational Physics* **28**(5), 1970–2001.

Liu, Z., Michaud, E. J. & Tegmark, M. (2022), Omnigrok: Grokking beyond algorithmic data, *in* 'The Eleventh International Conference on Learning Representations'.

Long, J., Shelhamer, E. & Darrell, T. (2015), Fully convolutional networks for semantic segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3431–3440.

Lu, L., Jin, P. & Karniadakis, G. E. (2019), 'Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators', *arXiv preprint arXiv:1910.03193* .

Luo, T., Ma, Z., Wang, Z., Xu, Z.-Q. J. & Zhang, Y. (2021), 'An upper limit of decaying rate with respect to frequency in deep neural network', *arXiv preprint arXiv:2105.11675* .

Luo, T., Ma, Z., Xu, Z.-Q. J. & Zhang, Y. (2019), 'Theory of the frequency principle for general deep neural networks', *arXiv preprint arXiv:1906.09235* .

Luo, T., Ma, Z., Xu, Z.-Q. J. & Zhang, Y. (2020), 'On the exact computation of linear frequency principle dynamics and its generalization', *arXiv preprint arXiv:2010.08153* .

Luo, T., Xu, Z.-Q. J., Ma, Z. & Zhang, Y. (2021), 'Phase diagram for two-layer relu neural networks at infinite-width limit', *Journal of Machine Learning Research* **22**(71), 1–47.

Ma, Y., Xu, Z.-Q. J. & Zhang, J. (2021), 'Frequency principle in deep learning beyond gradient-descent-based training', *arXiv preprint arXiv:2101.00747* .

McKinley, S. & Levine, M. (1998), 'Cubic spline interpolation', *College of the Redwoods* **45**(1), 1049–1060.

Mei, S., Montanari, A. & Nguyen, P.-M. (2018), 'A mean field view of the landscape of two-layer neural networks', *Proceedings of the National Academy of Sciences* **115**(33), E7665–E7671.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R. & Ng, R. (2020), Nerf: Representing scenes as neural radiance fields for view synthesis, *in* 'European Conference on Computer Vision', Springer, pp. 405–421.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R. & Ng, R. (2021), 'Nerf: Representing scenes as neural radiance fields for view synthesis', *Communications of the ACM* **65**(1), 99–106.

Minsky, M. L. & Papert, S. A. (1988), 'Perceptrons: expanded edition'.

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B. & Sutskever, I. (2021), 'Deep double descent: Where bigger models and more data hurt', *Journal of Statistical Mechanics: Theory and Experiment* **2021**(12), 124003.

Narkhede, M. V., Bartakke, P. P. & Sutaone, M. S. (2022), 'A review on weight initialization strategies for neural networks', *Artificial intelligence review* **55**(1), 291–322.

Nesterov, Y. (1983), 'A method of solving a convex programming problem with convergence rate o (1/k** 2)', *Doklady Akademii Nauk SSSR* **269**(3), 543.

Nye, M. & Saxe, A. (2018), 'Are efficient deep representations learnable?'.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A. et al. (2022), 'In-context learning and induction heads', *arXiv preprint arXiv:2209.11895* .

OpenAI (2023), 'Gpt-4 technical report'.

Papyan, V., Han, X. & Donoho, D. L. (2020), 'Prevalence of neural collapse during the terminal phase of deep learning training', *Proceedings of the National Academy of Sciences* **117**(40), 24652–24663.

Polyak, B. T. (1964), 'Some methods of speeding up the convergence of iteration methods', *Ussr computational mathematics and mathematical physics* **4**(5), 1–17.

Power, A., Burda, Y., Edwards, H., Babuschkin, I. & Misra, V. (2022), 'Grokking: Generalization beyond overfitting on small algorithmic datasets', *arXiv preprint arXiv:2201.02177* .

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. et al. (2018), 'Improving language understanding by generative pre-training'.

Raissi, M., Perdikaris, P. & Karniadakis, G. E. (2019), 'Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations', *Journal of Computational Physics* **378**, 686–707.

Reddi, S. J., Kale, S. & Kumar, S. (2018), On the convergence of adam and beyond, *in* 'International Conference on Learning Representations'.

Ronen, B., Jacobs, D., Kasten, Y. & Kritchman, S. (2019), 'The convergence rate of neural networks for learned functions of different frequencies', *Advances in Neural Information Processing Systems* **32**, 4761–4771.

Rotskoff, G. & Vanden-Eijnden, E. (2018), Parameters as interacting particles: long time convergence and asymptotic error scaling of neural networks, *in* 'Advances in neural information processing systems', pp. 7146–7155.

Shalev-Shwartz, S., Shamir, O. & Shammah, S. (2017), 'Failures of gradient-based deep learning', *arXiv preprint arXiv:1703.07950* .

Simonyan, K. & Zisserman, A. (2014), 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* .

Sirignano, J. & Spiliopoulos, K. (2020), 'Mean field analysis of neural networks: A central limit theorem', *Stochastic Processes and their Applications* **130**(3), 1820–1852.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D. & Wetzstein, G. (2020), 'Implicit neural representations with periodic activation functions', *Advances in neural information processing systems* **33**, 7462–7473.

Smith, S. L., Dherin, B., Barrett, D. & De, S. (2020), On the origin of implicit regularization in stochastic gradient descent, *in* 'International Conference on Learning Representations'.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *The journal of machine learning research* **15**(1), 1929–1958.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W. & Liu, Y. (2024), 'Roformer: Enhanced transformer with rotary position embedding', *Neurocomputing* **568**, 127063.

Szegedy, C., Ioffe, S., Vanhoucke, V. & Alemi, A. (2017), Inception-v4, inception-resnet and the impact of residual connections on learning, *in* 'Proceedings of the AAAI conference on artificial intelligence', Vol. 31.

Tan, M. & Le, Q. (2019), Efficientnet: Rethinking model scaling for convolutional neural networks, *in* 'International conference on machine learning', PMLR, pp. 6105–6114.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. & Ng, R. (2020), Fourier features let networks learn high frequency functions in low dimensional domains, *in* 'Advances in Neural Information Processing Systems', Vol. 33, Curran Associates, Inc., pp. 7537–7547.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S. et al. (2023), 'Llama 2: Open foundation and fine-tuned chat models', *arXiv preprint arXiv:2307.09288* .

Van der Maaten, L. & Hinton, G. (2008), 'Visualizing data using t-sne.', *Journal of machine learning research* **9**(11).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017), 'Attention is all you need', *Advances in neural information processing systems* **30**.

Wang, M. et al. (2024), 'Understanding the expressive power and mechanisms of transformer for sequence modeling', *arXiv preprint arXiv:2402.00522* .

Wei, J. et al. (2022), 'Chain-of-thought prompting elicits reasoning in large language models', *Advances in neural information processing systems* **35**, 24824–24837.

Wolpert, D. H. & Macready, W. G. (1997), 'No free lunch theorems for optimization', *IEEE transactions on evolutionary computation* **1**(1), 67–82.

Wu, L., Ma, C. et al. (2018), 'How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective', *Advances in Neural Information Processing Systems* **31**.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. et al. (2016), 'Google's neural machine translation system: Bridging the gap between human and machine translation', *arXiv preprint arXiv:1609.08144* .

Xie, S., Girshick, R., Dollár, P., Tu, Z. & He, K. (2017), Aggregated residual transformations for deep neural networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1492–1500.

Xie, X., Zhou, P., Li, H., Lin, Z. & Yan, S. (2024), 'Adan: Adaptive nesterov momentum algorithm for faster optimizing deep models', *IEEE Transactions on Pattern Analysis and Machine Intelligence* .

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X. & Stolcke, A. (2018), The microsoft 2017 conversational speech recognition system, *in* '2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)', IEEE, pp. 5934–5938.

Xu, Z.-Q. J., Zhang, Y., Luo, T., Xiao, Y. & Ma, Z. (2020), 'Frequency principle: Fourier analysis sheds light on deep neural networks', *Communications in Computational Physics* **28**(5), 1746–1767.

Yang, G. & Salman, H. (2019), 'A fine-grained spectral perspective on neural networks', *arXiv preprint arXiv:1907.10599* .

Zang, Y., Bao, G., Ye, X. & Zhou, H. (2020), 'Weak adversarial networks for high-dimensional partial differential equations', *Journal of Computational Physics* **411**, 109409.
URL: *http://dx.doi.org/10.1016/j.jcp.2020.109409*

Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O. (2017), Understanding deep learning requires rethinking generalization, *in* '5th International Conference on Learning Representations'.

Zhang, L., Luo, T., Zhang, Y., E, W., Xu, Z. Q. J. & Ma, Z. (2022), 'MOD-Net: A machine learning approach via model-operator-data network for solving pdes', *Communications in Computational Physics* **32**(2), 299–335.

Zhang, L., Zhang, Y. & Luo, T. (2023), 'Structure and gradient dynamics near global minima of two-layer neural networks', *arXiv preprint arXiv:2309.00508* .

Zhang, Y., Chen, C., Shi, N., Sun, R. & Luo, Z.-Q. (2022), 'Adam can converge without any modification on update rules', *Advances in neural information processing systems* **35**, 28386–28399.

Zhang, Y., Li, Y., Zhang, Z., Luo, T. & Xu, Z.-Q. J. (2021), 'Embedding principle: a hierarchical structure of loss landscape of deep neural networks', *arXiv preprint arXiv:2111.15527* .

Zhang, Y., Xu, Z.-Q. J., Luo, T. & Ma, Z. (2019), 'Explicitizing an Implicit Bias of the Frequency Principle in Two-layer Neural Networks', *arXiv:1905.10264 [cs, stat]* . arXiv: 1905.10264.
URL: *http://arxiv.org/abs/1905.10264*

Zhang, Y., Xu, Z.-Q. J., Luo, T. & Ma, Z. (2020), A type of generalization error induced by initialization in deep neural networks, *in* 'Mathematical and Scientific Machine Learning', PMLR, pp. 144–164.

Zhang, Y., Zhang, Z., Luo, T. & Xu, Z.-Q. J. (2021), 'Embedding principle of loss landscape of deep neural networks', *arXiv preprint arXiv:2105.14573* .

Zhang, Y., Zhang, Z., Zhang, L., Bai, Z., Luo, T. & Xu, Z.-Q. J. (2023), 'Optimistic estimate uncovers the potential of nonlinear models', *arXiv preprint arXiv:2307.08921* .

Zhang, Z. & Xu, Z.-Q. J. (2022), 'Implicit regularization of dropout', *arXiv preprint arXiv:2207.05952* .

Zhou, H., Zhou, Q., Jin, Z., Luo, T., Zhang, Y. & Xu, Z.-Q. J. (2022), 'Empirical phase diagram for three-layer neural networks with infinite width', *Advances in Neural Information Processing Systems* .

Zhou, H., Zhou, Q., Luo, T., Zhang, Y. & Xu, Z.-Q. (2022), 'Towards understanding the condensation of neural networks at initial training', *Advances in Neural Information Processing Systems* **35**, 2184–2196.

Zhu, Z., Wu, J., Yu, B., Wu, L. & Ma, J. (2018), 'The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects', *arXiv preprint arXiv:1803.00195* .