

# 深度学习

2017 年 2 月 7 日

DRAFT

# 目录

致谢	xvi
数学符号	xvii
<b>第一章 前言</b>	<b>1</b>
1.1 本书面向的读者 . . . . .	10
1.2 深度学习的历史趋势 . . . . .	11
1.2.1 神经网络的众多名称和命运变迁 . . . . .	12
1.2.2 与日俱增的数据量 . . . . .	17
1.2.3 与日俱增的模型规模 . . . . .	18
1.2.4 与日俱增的精度、复杂度和对现实世界的冲击 . . . . .	22
<b>第一部分 应用数学与机器学习基础</b>	<b>25</b>
<b>第二章 线性代数</b>	<b>27</b>
2.1 标量、向量、矩阵和张量 . . . . .	27
2.2 矩阵和向量相乘 . . . . .	30
2.3 单位矩阵和逆矩阵 . . . . .	31
2.4 线性相关和生成子空间 . . . . .	33
2.5 范数 . . . . .	34
2.6 特殊类型的矩阵和向量 . . . . .	36
2.7 特征分解 . . . . .	37
2.8 奇异值分解 . . . . .	39
2.9 Moore-Penrose 伪逆 . . . . .	40

2.10	迹运算 . . . . .	41
2.11	行列式 . . . . .	42
2.12	实例：主成分分析 . . . . .	42
<b>第三章</b>	<b>概率与信息论</b>	<b>47</b>
3.1	为什么要用概率? . . . . .	47
3.2	随机变量 . . . . .	49
3.3	概率分布 . . . . .	50
3.3.1	离散型变量和概率分布律函数 . . . . .	50
3.3.2	连续型变量和概率密度函数 . . . . .	51
3.4	边缘概率 . . . . .	52
3.5	条件概率 . . . . .	52
3.6	条件概率的链式法则 . . . . .	53
3.7	独立性和条件独立性 . . . . .	53
3.8	期望, 方差和协方差 . . . . .	54
3.9	常用概率分布 . . . . .	55
3.9.1	Bernoulli 分布 . . . . .	55
3.9.2	Multinoulli 分布 . . . . .	56
3.9.3	高斯分布 . . . . .	56
3.9.4	指数分布和 Laplace 分布 . . . . .	58
3.9.5	Dirac 分布和经验分布 . . . . .	58
3.9.6	分布的混合 . . . . .	59
3.10	常用函数的一些性质 . . . . .	60
3.11	贝叶斯规则 . . . . .	63
3.12	连续型变量的技术细节 . . . . .	63
3.13	信息论 . . . . .	65
3.14	结构化概率模型 . . . . .	68
<b>第四章</b>	<b>数值计算</b>	<b>72</b>
4.1	上溢和下溢 . . . . .	72
4.2	病态条件数 . . . . .	73
4.3	基于梯度的优化方法 . . . . .	74
4.3.1	梯度之上： Jacobian 和 Hessian 矩阵 . . . . .	77

4.4	约束优化 . . . . .	82
4.5	实例：线性最小二乘 . . . . .	85
<b>第五章</b>	<b>机器学习基础</b>	<b>87</b>
5.1	学习算法 . . . . .	87
5.1.1	任务 $T$ . . . . .	88
5.1.2	性能度量 $P$ . . . . .	91
5.1.3	经验 $E$ . . . . .	92
5.1.4	实例：线性回归 . . . . .	94
5.2	容量、过拟合和欠拟合 . . . . .	97
5.2.1	没有免费午餐定理 . . . . .	102
5.2.2	正则化 . . . . .	102
5.3	超参数和验证集 . . . . .	105
5.3.1	交叉验证 . . . . .	106
5.4	估计、偏差和方差 . . . . .	107
5.4.1	点估计 . . . . .	107
5.4.2	偏差 . . . . .	109
5.4.3	方差和标准误差 . . . . .	111
5.4.4	权衡偏值和方差以最小化均方误差 . . . . .	113
5.4.5	一致性 . . . . .	114
5.5	最大似然估计 . . . . .	115
5.5.1	条件对数似然和均方误差 . . . . .	116
5.5.2	最大似然的性质 . . . . .	117
5.6	贝叶斯统计 . . . . .	118
5.6.1	最大后验 (MAP) 估计 . . . . .	121
5.7	监督学习算法 . . . . .	122
5.7.1	概率监督学习 . . . . .	122
5.7.2	支持向量机 . . . . .	123
5.7.3	其他简单的监督学习算法 . . . . .	125
5.8	无监督学习算法 . . . . .	128
5.8.1	主成分分析 . . . . .	128
5.8.2	$k$ -均值聚类 . . . . .	131
5.9	随机梯度下降 . . . . .	132

5.10 构建机器学习算法 . . . . .	133
5.11 深度学习的动机与挑战 . . . . .	134
5.11.1 维数灾难 . . . . .	135
5.11.2 局部不变性和平滑正则化 . . . . .	135
5.11.3 流形学习 . . . . .	139
<b>第二部分 深层网络：现代实践 . . . . .</b>	<b>143</b>
<b>第六章 深度前馈网络 . . . . .</b>	<b>145</b>
6.1 实例：学习 XOR . . . . .	147
6.2 基于梯度的学习 . . . . .	152
6.2.1 代价函数 . . . . .	153
6.2.1.1 用最大似然学习条件分布 . . . . .	154
6.2.1.2 学习条件统计量 . . . . .	155
6.2.2 输出单元 . . . . .	156
6.2.2.1 用于高斯输出分布的线性单元 . . . . .	156
6.2.2.2 用于 Bernoulli 输出分布的 sigmoid 单元 . . . . .	157
6.2.2.3 用于 Multinoulli 输出分布的 softmax 单元 . . . . .	159
6.2.2.4 其他的输出类型 . . . . .	161
6.3 隐藏单元 . . . . .	165
6.3.1 整流线性单元及其扩展 . . . . .	166
6.3.2 logistic sigmoid 与双曲正切函数 . . . . .	168
6.3.3 其他隐藏单元 . . . . .	168
6.4 结构设计 . . . . .	170
6.4.1 通用近似性质和深度 . . . . .	170
6.4.2 其他结构上的考虑 . . . . .	173
6.5 反向传播和其他的微分算法 . . . . .	175
6.5.1 计算图 . . . . .	176
6.5.2 微积分中的链式法则 . . . . .	176
6.5.3 递归地使用链式法则来实现 BP . . . . .	178
6.5.4 全连接 MLP 中 BP 的计算 . . . . .	180
6.5.5 符号到符号的导数 . . . . .	181

6.5.6	一般化的 BP . . . . .	183
6.5.7	实例：用于 MLP 训练的 BP . . . . .	188
6.5.8	复杂化 . . . . .	190
6.5.9	深度学习界以外的微分 . . . . .	191
6.5.10	高阶微分 . . . . .	192
6.6	历史小记 . . . . .	193
<b>第七章</b>	<b>深度学习中的正则化技术</b>	<b>196</b>
7.1	参数范数惩罚 . . . . .	197
7.1.1	$L^2$ 参数正则化 . . . . .	198
7.1.2	$L^1$ 参数正则化 . . . . .	201
7.2	作为约束的范数惩罚 . . . . .	203
7.3	正则化和欠约束问题 . . . . .	205
7.4	数据集增强 . . . . .	206
7.5	噪声鲁棒性 . . . . .	207
7.5.1	向输出目标注入噪声 . . . . .	208
7.6	半监督学习 . . . . .	208
7.7	多任务学习 . . . . .	209
7.8	提前终止 . . . . .	210
7.9	参数绑定和参数共享 . . . . .	216
7.9.1	卷积神经网络 . . . . .	217
7.10	稀疏表示 . . . . .	217
7.11	Bagging 和其他集成方法 . . . . .	219
7.12	Dropout . . . . .	221
7.13	对抗训练 . . . . .	229
7.14	切面距离、正切传播和流形正切分类器 . . . . .	231
<b>第八章</b>	<b>深度模型中的优化</b>	<b>234</b>
8.1	学习和纯优化有什么不同 . . . . .	234
8.1.1	经验风险最小化 . . . . .	235
8.1.2	替代损失函数和提前终止 . . . . .	236
8.1.3	批算法和minibatch 算法 . . . . .	236

8.2	神经网络优化中的挑战 . . . . .	240
8.2.1	病态 . . . . .	241
8.2.2	局部极小值 . . . . .	241
8.2.3	高原、鞍点和其他平坦区域 . . . . .	243
8.2.4	悬崖和梯度爆炸 . . . . .	245
8.2.5	长期依赖 . . . . .	246
8.2.6	非精确梯度 . . . . .	247
8.2.7	局部和全局结构间的弱对应 . . . . .	247
8.2.8	优化的理论限制 . . . . .	249
8.3	基本算法 . . . . .	250
8.3.1	随机梯度下降 . . . . .	250
8.3.2	动量 . . . . .	252
8.3.3	Nesterov 动量 . . . . .	255
8.4	参数初始化策略 . . . . .	255
8.5	具有自适应学习速率的算法 . . . . .	260
8.5.1	AdaGrad . . . . .	260
8.5.2	RMSProp . . . . .	261
8.5.3	Adam . . . . .	261
8.5.4	选择正确的优化算法 . . . . .	262
8.6	二阶近似方法 . . . . .	264
8.6.1	牛顿法 . . . . .	265
8.6.2	共轭梯度 . . . . .	266
8.6.3	BFGS . . . . .	269
8.7	优化策略和元算法 . . . . .	270
8.7.1	batch normalization . . . . .	270
8.7.2	坐标下降 . . . . .	273
8.7.3	Polyak 平均 . . . . .	273
8.7.4	监督预训练 . . . . .	274
8.7.5	设计有助于优化的模型 . . . . .	276
8.7.6	连续方法和课程学习 . . . . .	277
<b>第九章</b>	<b>卷积神经网络</b>	<b>280</b>
9.1	卷积运算 . . . . .	281

9.2	动机 . . . . .	283
9.3	池化 . . . . .	289
9.4	卷积与池化作为一种无限强的先验 . . . . .	294
9.5	基本卷积函数的变体 . . . . .	295
9.6	结构化输出 . . . . .	305
9.7	数据类型 . . . . .	306
9.8	高效的卷积算法 . . . . .	308
9.9	随机或无监督的特征 . . . . .	309
9.10	卷积神经网络的神经科学基础 . . . . .	310
9.11	卷积神经网络与深度学习的历史 . . . . .	316
<b>第十章</b>	<b>序列建模：循环和递归网络</b>	<b>318</b>
10.1	展开计算图 . . . . .	319
10.2	循环神经网络 . . . . .	322
10.2.1	Teacher Forcing 和输出循环网络 . . . . .	325
10.2.2	计算循环神经网络的梯度 . . . . .	327
10.2.3	作为有向图模型的循环网络 . . . . .	328
10.2.4	基于上下文的RNN 序列建模 . . . . .	332
10.3	双向RNN . . . . .	334
10.4	基于编码-解码的序列到序列架构 . . . . .	336
10.5	深度循环网络 . . . . .	338
10.6	递归神经网络 . . . . .	339
10.7	长期依赖的挑战 . . . . .	341
10.8	回声状态网络 . . . . .	343
10.9	渗漏单元和其他多时间尺度的策略 . . . . .	345
10.9.1	时间维度的跳跃连接 . . . . .	345
10.9.2	渗漏单元和一系列不同时间尺度 . . . . .	345
10.9.3	删除连接 . . . . .	346
10.10	长短期记忆和其他门控 RNN . . . . .	347
10.10.1	LSTM . . . . .	347
10.10.2	其他门控 RNN . . . . .	349
10.11	优化长期依赖 . . . . .	350
10.11.1	截断梯度 . . . . .	351

10.11.2 引导信息流的正则化 . . . . .	353
10.12 外显记忆 . . . . .	353
<b>第十一章 实用方法</b>	<b>357</b>
11.1 性能度量 . . . . .	358
11.2 默认的基准模型 . . . . .	360
11.3 是否收集更多数据 . . . . .	361
11.4 选择超参数 . . . . .	362
11.4.1 手动调整超参数 . . . . .	362
11.4.2 自动超参数优化算法 . . . . .	365
11.4.3 网格搜索 . . . . .	366
11.4.4 随机搜索 . . . . .	367
11.4.5 基于模型的超参数优化 . . . . .	368
11.5 调试策略 . . . . .	369
11.6 示例：多位数字识别 . . . . .	372
<b>第十二章 应用</b>	<b>375</b>
12.1 大规模深度学习 . . . . .	375
12.1.1 快速的 CPU 实现 . . . . .	375
12.1.2 GPU 实现 . . . . .	376
12.1.3 大规模的分布式实现 . . . . .	378
12.1.4 模型压缩 . . . . .	379
12.1.5 动态结构 . . . . .	379
12.1.6 深度网络的专用硬件实现 . . . . .	381
12.2 计算机视觉 . . . . .	383
12.2.1 预处理 . . . . .	383
12.2.1.1 对比度归一化 . . . . .	384
12.2.2 数据集增强 . . . . .	387
12.3 语音识别 . . . . .	388
12.4 自然语言处理 . . . . .	390
12.4.1 $n$ -gram . . . . .	390
12.4.2 神经语言模型 . . . . .	392

12.4.3 高维输出 . . . . .	394
12.4.3.1 使用短列表 . . . . .	394
12.4.3.2 分层 Softmax . . . . .	395
12.4.3.3 重要采样 . . . . .	397
12.4.3.4 噪声对比估计和排名损失 . . . . .	399
12.4.4 结合 $n$ -gram 和神经语言模型 . . . . .	399
12.4.5 神经机器翻译 . . . . .	400
12.4.5.1 使用注意机制并对齐数据片段 . . . . .	402
12.4.6 历史观点 . . . . .	404
12.5 其他应用 . . . . .	405
12.5.1 推荐系统 . . . . .	405
12.5.1.1 探索与开发 . . . . .	407
12.5.2 知识表示、推理和回答 . . . . .	408
12.5.2.1 知识、联系和回答 . . . . .	408
<b>第三部分 深度学习研究</b>	<b>412</b>
<b>第十三章 线性因子模型</b>	<b>415</b>
13.1 概率 PCA 和因子分析 . . . . .	416
13.2 独立分量分析 . . . . .	417
13.3 慢特征分析 . . . . .	419
13.4 稀疏编码 . . . . .	421
13.5 PCA 的流形解释 . . . . .	424
<b>第十四章 自编码器</b>	<b>427</b>
14.1 欠完备自编码器 . . . . .	428
14.2 正则自编码器 . . . . .	429
14.2.1 稀疏自编码器 . . . . .	429
14.2.2 去噪自编码器 . . . . .	431
14.2.3 惩罚导数作为正则 . . . . .	432
14.3 表示能力、层的大小和深度 . . . . .	432
14.4 随机编码器和解码器 . . . . .	433

14.5 去噪自编码器 . . . . .	434
14.5.1 得分估计 . . . . .	435
14.5.2 历史观点 . . . . .	438
14.6 使用自编码器学习流形 . . . . .	438
14.7 收缩自编码器 . . . . .	443
14.8 预测稀疏分解 . . . . .	445
14.9 自编码器的应用 . . . . .	446
<b>第十五章 表示学习</b>	<b>447</b>
15.1 贪心逐层无监督预训练 . . . . .	448
15.1.1 何时以及为何无监督预训练有效? . . . . .	450
15.2 迁移学习和领域自适应 . . . . .	455
15.3 半监督解释因果关系 . . . . .	459
15.4 分布式表示 . . . . .	464
15.5 得益于深度的指数增益 . . . . .	469
15.6 提供发现潜在原因的线索 . . . . .	470
<b>第十六章 深度学习中的结构化概率模型</b>	<b>473</b>
16.1 非结构化建模的挑战 . . . . .	474
16.2 使用图来描述模型结构 . . . . .	477
16.2.1 有向模型 . . . . .	478
16.2.2 无向模型 . . . . .	480
16.2.3 配分函数 . . . . .	482
16.2.4 基于能量的模型 . . . . .	483
16.2.5 分离和 d-分离 . . . . .	485
16.2.6 在有向模型和无向模型中转换 . . . . .	487
16.2.7 因子图 . . . . .	488
16.3 从图模型中采样 . . . . .	494
16.4 结构化建模的优势 . . . . .	495
16.5 学习依赖性关系 . . . . .	495
16.6 推断和近似推断 . . . . .	496
16.7 结构化概率模型的深度学习方法 . . . . .	497
16.7.1 实例：受限玻尔兹曼机 . . . . .	499

<b>第十七章 蒙特卡罗方法</b>	<b>502</b>
17.1 采样和蒙特卡罗方法 . . . . .	502
17.1.1 为什么需要采样? . . . . .	502
17.1.2 蒙特卡罗采样的基础 . . . . .	503
17.2 重要采样 . . . . .	504
17.3 马尔可夫链蒙特卡罗方法 . . . . .	506
17.4 Gibbs 采样 . . . . .	510
17.5 不同的峰值之间的混合挑战 . . . . .	511
17.5.1 不同峰值之间通过回火来混合 . . . . .	513
17.5.2 深度也许会有助于混合 . . . . .	514
<b>第十八章 面对配分函数</b>	<b>516</b>
18.1 对数似然梯度 . . . . .	516
18.2 随机最大似然和对比散度 . . . . .	518
18.3 伪似然 . . . . .	524
18.4 得分匹配和比率匹配 . . . . .	526
18.5 去噪得分匹配 . . . . .	528
18.6 噪扰对比估计 . . . . .	529
18.7 估计配分函数 . . . . .	531
18.7.1 退火重要采样 . . . . .	533
18.7.2 桥式采样 . . . . .	536
<b>第十九章 近似推断</b>	<b>538</b>
19.1 推断是一个优化问题 . . . . .	539
19.2 期望最大化 . . . . .	541
19.3 最大后验推断和稀疏编码 . . . . .	542
19.4 变分推断和学习 . . . . .	544
19.4.1 离散型潜变量 . . . . .	545
19.4.2 变分法 . . . . .	551
19.4.3 连续型潜变量 . . . . .	554
19.4.4 学习和推断之间的相互作用 . . . . .	556
19.5 learned 近似推断 . . . . .	556
19.5.1 wake sleep . . . . .	557

19.5.2 learned 推断的其他形式 . . . . .	557
<b>第二十章 深度生成模型</b>	<b>559</b>
20.1 玻尔兹曼机 . . . . .	559
20.2 受限玻尔兹曼机 . . . . .	561
20.2.1 条件分布 . . . . .	562
20.2.2 训练受限玻尔兹曼机 . . . . .	563
20.3 深度信念网络 . . . . .	564
20.4 深度玻尔兹曼机 . . . . .	566
20.4.1 有趣的性质 . . . . .	568
20.4.2 DBM 均匀场推断 . . . . .	569
20.4.3 DBM 的参数学习 . . . . .	571
20.4.4 逐层预训练 . . . . .	572
20.4.5 联合训练深度玻尔兹曼机 . . . . .	574
20.5 实值数据上的玻尔兹曼机 . . . . .	578
20.5.1 Gaussian-Bernoulli RBM . . . . .	578
20.5.2 条件协方差的无向模型 . . . . .	579
20.6 卷积玻尔兹曼机 . . . . .	583
20.7 用于结构化或序列输出的玻尔兹曼机 . . . . .	585
20.8 其他玻尔兹曼机 . . . . .	586
20.9 通过随机操作的反向传播 . . . . .	587
20.9.1 通过离散随机操作的反向传播 . . . . .	588
20.10 有向生成网络 . . . . .	591
20.10.1 sigmoid 信念网络 . . . . .	591
20.10.2 可微生成器网络 . . . . .	592
20.10.3 变分自编码器 . . . . .	594
20.10.4 生成式对抗网络 . . . . .	597
20.10.5 生成矩匹配网络 . . . . .	600
20.10.6 卷积生成网络 . . . . .	601
20.10.7 自回归网络 . . . . .	602
20.10.8 线性自回归网络 . . . . .	602
20.10.9 神经自回归网络 . . . . .	603
20.10.10 NADE . . . . .	604

20.11 从自编码器采样 . . . . .	606
20.11.1 与任意去噪自编码器相关的马尔可夫链 . . . . .	607
20.11.2 夹合与条件采样 . . . . .	607
20.11.3 回退训练过程 . . . . .	608
20.12 生成随机网络 . . . . .	609
20.12.1 判别GSN . . . . .	610
20.13 其他生成方案 . . . . .	611
20.14 评估生成模型 . . . . .	612
20.15 结论 . . . . .	614
<b>参考文献</b>	<b>615</b>
<b>术语</b>	<b>679</b>

# 致谢

TODO

# 数学符号

本节简要介绍本书所使用的数学符号。我们在第二章至第四章中描述大多数数学概念，如果你不熟悉任何相应的数学概念，可以参考对应的章节。

## 数和数组

$a$	标量 (整数或实数)
$\mathbf{a}$	向量
$\mathbf{A}$	矩阵
$\mathbf{A}$	张量
$I_n$	$n$ 行 $n$ 列的单位矩阵
$I$	维度蕴含于上下文的单位矩阵
$e^{(i)}$	标准基向量 $[0, \dots, 0, 1, 0, \dots, 0]$ , 其中索引 $i$ 处值为 1
$\text{diag}(\mathbf{a})$	对角方阵, 其中对角元素由 $\mathbf{a}$ 给定
$a$	标量随机变量
$\mathbf{a}$	向量随机变量
$\mathbf{A}$	矩阵随机变量

## 集合和图

$\mathbb{A}$  集合

$\mathbb{R}$  实数集

$\{0, 1\}$  包含 0 和 1 的集合

$\{0, 1, \dots, n\}$  包含 0 和  $n$  之间所有整数的集合

$[a, b]$  包含  $a$  和  $b$  的实数区间

$(a, b]$  不包含  $a$  但包含  $b$  的实数区间

$\mathbb{A} \setminus \mathbb{B}$  差集，即其元素包含于  $\mathbb{A}$  但不包含于  $\mathbb{B}$

$\mathcal{G}$  图

$Pa_{\mathcal{G}}(x_i)$  图  $\mathcal{G}$  中  $x_i$  的父节点

## 索引

$a_i$  向量  $\mathbf{a}$  的第  $i$  个元素，其中索引从 1 开始

$a_{-i}$  除了第  $i$  个元素， $\mathbf{a}$  的所有元素

$A_{i,j}$  矩阵  $\mathbf{A}$  的  $i, j$  元素

$\mathbf{A}_{i,:}$  矩阵  $\mathbf{A}$  的第  $i$  行

$\mathbf{A}_{:,i}$  矩阵  $\mathbf{A}$  的第  $i$  列

$A_{i,j,k}$  3 维张量  $\mathbf{A}$  的  $(i, j, k)$  元素

$\mathbf{A}_{:,:,i}$  3 维张量的 2 维切片

$a_i$  随机向量  $\mathbf{a}$  的第  $i$  个元素

## 线性代数中的操作

 $\mathbf{A}^\top$  矩阵  $\mathbf{A}$  的转置 $\mathbf{A}^+$   $\mathbf{A}$  的Moore-Penrose 伪逆 $\mathbf{A} \odot \mathbf{B}$   $\mathbf{A}$  和  $\mathbf{B}$  的逐元素乘积 (Hadamard 乘积) $\det(\mathbf{A})$   $\mathbf{A}$  的行列式

## 微积分

 $\frac{dy}{dx}$   $y$  关于  $x$  的导数 $\frac{\partial y}{\partial x}$   $y$  关于  $x$  的偏导 $\nabla_{\mathbf{x}}y$   $y$  关于  $\mathbf{x}$  的梯度 $\nabla_{\mathbf{X}}y$   $y$  关于  $\mathbf{X}$  的矩阵导数 $\nabla_{\mathbf{X}}y$   $y$  关于  $\mathbf{X}$  求导后的张量 $\frac{\partial f}{\partial \mathbf{x}}$   $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  的Jacobian矩阵  $\mathbf{J} \in \mathbb{R}^{m \times n}$  $\nabla_x^2 f(\mathbf{x})$  or  $\mathbf{H}(f)(\mathbf{x})$   $f$  在点  $\mathbf{x}$  处的Hessian矩阵 $\int f(\mathbf{x}) d\mathbf{x}$   $\mathbf{x}$  整个域上的定积分 $\int_{\mathbb{S}} f(\mathbf{x}) d\mathbf{x}$  集合  $\mathbb{S}$  上关于  $\mathbf{x}$  的定积分

## 概率和信息论

$a \perp b$	$a$ 和 $b$ 相互独立的随机变量
$a \perp b   c$	给定 $c$ 后条件独立
$P(a)$	离散变量上的概率分布
$p(a)$	连续变量（或变量类型未指定时）上的概率分布
$a \sim P$	具有分布 $P$ 的随机变量 $a$
$\mathbb{E}_{x \sim P}[f(x)]$ or $\mathbb{E} f(x)$	$f(x)$ 关于 $P(x)$ 的期望
$\text{Var}(f(x))$	$f(x)$ 在分布 $P(x)$ 下的方差
$\text{Cov}(f(x), g(x))$	$f(x)$ 和 $g(x)$ 在分布 $P(x)$ 下的协方差
$H(x)$	随机变量 $x$ 的香农熵
$D_{\text{KL}}(P \  Q)$	$P$ 和 $Q$ 的KL 散度
$\mathcal{N}(x; \mu, \Sigma)$	均值为 $\mu$ 协方差为 $\Sigma$ , $x$ 上的高斯分布

## 函数

$f : \mathbb{A} \rightarrow \mathbb{B}$  定义域为  $\mathbb{A}$  值域为  $\mathbb{B}$  的函数  $f$

$f \circ g$   $f$  和  $g$  的组合

$f(\mathbf{x}; \boldsymbol{\theta})$  由  $\boldsymbol{\theta}$  参数化, 关于  $\mathbf{x}$  的函数 (有时为简化表示, 我们忽略  $\boldsymbol{\theta}$  记为  $f(\mathbf{x})$ )

$\log x$   $x$  的自然对数

$\sigma(x)$  Logistic sigmoid,  $\frac{1}{1 + \exp(-x)}$

$\zeta(x)$  Softplus,  $\log(1 + \exp(x))$

$\|\mathbf{x}\|_p$   $\mathbf{x}$  的  $L^p$  范数

$\|\mathbf{x}\|$   $\mathbf{x}$  的  $L^2$  范数

$x^+$   $x$  的正数部分, 即  $\max(0, x)$

$\mathbf{1}_{\text{condition}}$  如果条件为真则为 1, 否则为 0

有时候我们使用函数  $f$ , 它的参数是一个标量, 但应用到一个向量、矩阵或张量:  $f(\mathbf{x})$ ,  $f(\mathbf{X})$ , or  $f(\mathbf{X})$ 。这表示逐元素地将  $f$  应用于数组。例如,  $\mathbf{C} = \sigma(\mathbf{X})$ , 则对于所有合法的  $i$ 、 $j$  和  $k$ ,  $C_{i,j,k} = \sigma(X_{i,j,k})$ 。

## 数据集和分布

$p_{\text{data}}$  数据生成分布

$\hat{p}_{\text{train}}$  由训练集定义的经验分布

$\mathbb{X}$  训练样本的集合

$\mathbf{x}^{(i)}$  数据集的第  $i$  个样本 (输入)

$y^{(i)}$  or  $\mathbf{y}^{(i)}$  监督学习中与  $\mathbf{x}^{(i)}$  关联的目标

$\mathbf{X}$   $m \times n$  的矩阵, 其中行  $\mathbf{X}_{i,:}$  为输入样本  $\mathbf{x}^{(i)}$

DRAFT

# 第一章 前言

远在古希腊时期，发明家就梦想着创造能思考的机器。神话人物皮格马利翁 (Pygmalion)、代达罗斯 (Daedalus) 和赫淮斯托斯 (Hephaestus) 可以被看作传说中的发明家，而加拉蒂亚 (Galatea)、塔洛斯 (Talos) 和潘多拉 (Pandora) 则可以被视为人造生命 (Ovid and Martin, 2004; Sparkes, 1996; Tandy, 1997)。

当人类第一次构思可编程计算机时，就已经在思考计算机能否变得智能（尽管这距造出第一台计算机还有一百多年）(Lovelace, 1842)。如今，**人工智能** (artificial intelligence, AI) 已经成为一个具有众多实际应用和活跃研究课题的领域，并且正在蓬勃发展。我们期望通过智能软件自动地处理常规劳动、理解语音或图像、帮助医学诊断和支持基础科学的研究。

在人工智能的早期，那些对人类智力来说非常困难、但对计算机来说相对简单的问题得到迅速解决，比如，那些可以通过一系列形式化的数学规则来描述的问题。人工智能的真正挑战在于解决那些对人来说很容易执行、但很难形式化描述的任务，如识别人们所说的话或图像中的脸。对于这些问题，我们人类往往可以凭直觉轻易地解决。

针对这些比较直观的问题，本书讨论一种解决方案。该方案可以让计算机从经验中学习，并根据层次化的概念体系来理解世界，而每个概念则通过与某些相对简单的概念之间的关系来定义。让计算机从经验获取知识，可以避免由人类来给计算机形式化地指定它需要的所有知识。层次化的概念让计算机构建较简单的概念来学习复杂概念。如果绘制出这些概念如何建立在彼此之上的图，我们将得到一张“深”（层次很多）的图。基于这个原因，我们称这种方法为**AI深度学习** (deep learning)。

AI许多早期的成功发生在相对朴素且形式化的环境中，而且不要求计算机具备很多关于世界的知识。例如，IBM 的深蓝 (Deep Blue) 国际象棋系统在 1997 年

击败了世界冠军Garry Kasparov(Hsu, 2002)。显然国际象棋是一个非常简单的领域，因为它仅含有 64 个位置并只能以严格限制的方式移动 32 个棋子。设计一种成功的国际象棋策略是巨大的成就，但向计算机描述棋子及其允许的走法并不是挑战的困难所在。国际象棋完全可以由一个非常简短的、完全形式化的规则列表来描述，并可以容易地由程序员事先准备好。

讽刺的是，抽象和形式化的任务对人类而言是最困难的脑力任务之一，但对计算机而言却属于最容易的。计算机早就能够打败人类最好的象棋选手，但直到最近计算机才在识别对象或语音任务中达到人类平均水平。一个人的日常生活需要关于世界的巨量知识。很多这方面的知识是主观的、直观的，因此很难通过形式化的方式表达清楚。计算机需要获取同样的知识才能表现出智能。人工智能的一个关键挑战就是如何将这些非形式化的知识传达给计算机。

一些人工智能项目力求将关于世界的知识用形式化的语言进行硬编码 (hard-code)。计算机可以使用逻辑推理规则来自动地理解这些形式化语言中的申明。这就是众所周知的人工智能的知识库 (knowledge base) 方法。这些项目没有导致重大的成功。其中最著名的项目是 Cyc (Lenat and Guha, 1989)。Cyc 包括一个推断引擎和一个使用 CycL 语言描述的声明数据库。这些声明是由人类监督者输入的。这是一个笨拙的过程。人们设法设计出足够复杂的形式化规则来精确地描述世界。例如，Cyc 不能理解一个关于名为Fred的人在早上剃须的故事 (Linde, 1992)。它的推理引擎检测到故事中的不一致性：它知道人没有电气零件，但由于Fred正拿着一个电动剃须刀，它认为实体“正在剃须的 Fred”(“FredWhileShaving”) 含有电气部件。因此它产生了这样的疑问——Fred在刮胡子的时候是否仍然是一个人。

依靠硬编码的知识体系面对的困难表明，AI系统需要具备自己获取知识的能力，即从原始数据中提取模式的能力。这种能力被称为机器学习 (machine learning)。引入机器学习使计算机能够解决涉及现实世界知识的问题，并能作出看似主观的决策。比如，一个被称为逻辑回归 (logistic regression) 的简单机器学习算法可以决定是否建议剖腹产 (Mor-Yosef *et al.*, 1990)。而同样是简单机器学习算法的朴素贝叶斯 (naive Bayes) 则可以区分垃圾电子邮件和合法电子邮件。

这些简单的机器学习算法的性能在很大程度上依赖于给定数据的表示 (representation)。例如，当逻辑回归被用于推荐剖腹产时，AI系统不直接检查患者。相反，医生需要告诉系统几条相关的信息，诸如子宫疤痕是否存在。表示患者的每条信息被称为一个特征。逻辑回归学习病人的这些特征如何与各种结果相关联。然而，它丝毫不影响该特征定义的方式。如果将病人的 MRI 扫描作为逻辑回归的输入，而

不是医生正式的报告，它将无法作出有用的预测。MRI 扫描的单一像素与分娩过程中并发症之间的相关性微乎其微。

在整个计算机科学乃至日常生活中，对表示的依赖都是一个普遍现象。在计算机科学中，如果数据集合被精巧地结构化并被智能地索引，那么诸如搜索之类的操作的处理速度就可以成指数级地加快。人们可以很容易地在阿拉伯数字的表示下进行算术运算，但在罗马数字的表示下运算会比较耗时。因此，毫不奇怪，表示的选择会对机器学习算法的性能产生巨大的影响。图1.1展示了一个简单的可视化例子。

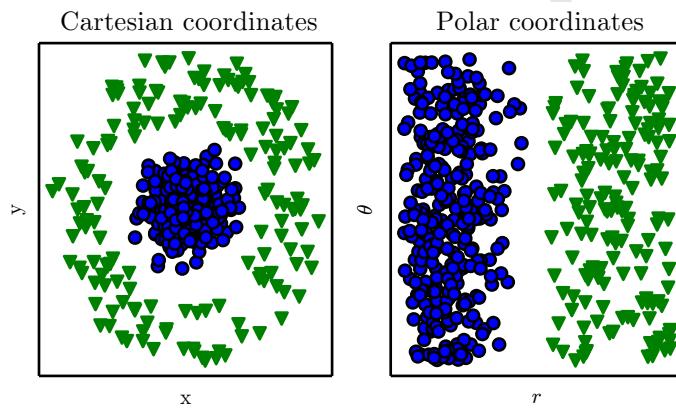


图 1.1: 不同表示的例子：假设我们想在散点图中画一条线来分隔两类数据。在左图，我们使用笛卡尔坐标表示数据，这个任务是不可能的。右图中，我们用极坐标表示数据，可以用垂直线简单地解决这个任务。（与 David Warde-Farley 合作画出此图。）

许多人工智能任务都可以通过以下方式解决：先提取一个合适的特征集，然后将这些特征提供给简单的机器学习算法。例如，对于通过声音鉴别说话者的任务来说，一个有用的特征是对其声道大小的估计。这个特征为判断说话者是男性、女性还是儿童提供了有力线索。

然而，对于许多任务来说，我们很难知道应该提取哪些特征。例如，假设我们想编写一个程序来检测照片中的车。我们知道，汽车有轮子，所以我们可能会想用车轮的存在与否作为特征。不幸的是，我们难以准确地根据像素值来描述车轮看上去像什么。虽然车轮具有简单的几何形状，但它的图像可能会因场景而异，如落在车轮上的阴影、太阳照亮的车轮的金属零件、汽车的挡泥板或者遮挡的车轮一部分的前景物体等等。

解决这个问题的途径之一是使用机器学习来发掘表示本身，而不仅仅把表示映

射到输出。这种方法我们称之为表示学习 (representation learning)。学习到的表示往往比手动设计的表示表现得更好。并且它们只需最少的人工干预，就能让AI系统迅速适应新的任务。表示学习算法只需几分钟就可以为简单的任务发现一个很好的特征集，对于复杂任务则需要几小时到几个月。手动为一个复杂的任务设计特征需要耗费大量的人工时间和精力；甚至需要花费整个社群研究人员几十年的时间。

表示学习算法的典型例子是自编码器 (autoencoder)。自编码器由一个编码器 (encoder) 函数和一个解码器 (decoder) 函数组合而成。编码器函数将输入数据转换为一种不同的表示，而解码器函数则将这个新的表示转换到原来的形式。我们期望当输入数据经过编码器和解码器之后尽可能多地保留信息，同时希望新的表示有各种好的特性，这也是自编码器的训练目标。为了实现不同的特性，我们可以设计不同形式的自编码器。

当设计特征或设计用于学习特征的算法时，我们的目标通常是分离出能解释观察数据的变差因素 (factors of variation)。在此背景下，“因素”这个词仅指代影响的不同来源；因素通常不是乘性组合。这些因素通常是不能被直接观察到的量。相反，它们可能是现实世界中观察不到的物体或者不可观测的力，但会影响可观测的量。为了对观察到的数据提供有用的简化解释或推断其原因，它们还可能以概念的形式存在于人类的思维中。它们可以被看作数据的概念或者抽象，帮助我们了解这些数据的丰富多样性。当分析语音记录时，变差因素包括说话者的年龄、性别、他们的口音和他们正在说的词语。当分析汽车的图像时，变差因素包括汽车的位置、它的颜色、太阳的角度和亮度。

在许多现实的人工智能应用中，困难主要源于很多变差因素影响着我们能够观察到的每一个数据。比如，在一张包含红色汽车的图片中，其单个像素在夜间可能会非常接近黑色。汽车轮廓的形状取决于视角。大多数应用需要我们理清变差因素并忽略我们不关心的因素。

显然，从原始数据中提取如此高层次、抽象的特征是非常困难的。许多诸如说话口音这样的变差因素，只能通过对数据进行复杂的、接近人类水平的理解来辨识。这几乎与获得原问题的表示一样困难，因此，乍一看，表示学习似乎并不能帮助我们。

深度学习 (deep learning) 通过其他较简单的表示来表达复杂表示，解决了表示学习中的核心问题。

深度学习让计算机通过较简单概念构建复杂的概念。图1.2展示了深度学习系统如何通过组合较简单的概念（例如转角和轮廓，它们转而由边线定义）来表示图

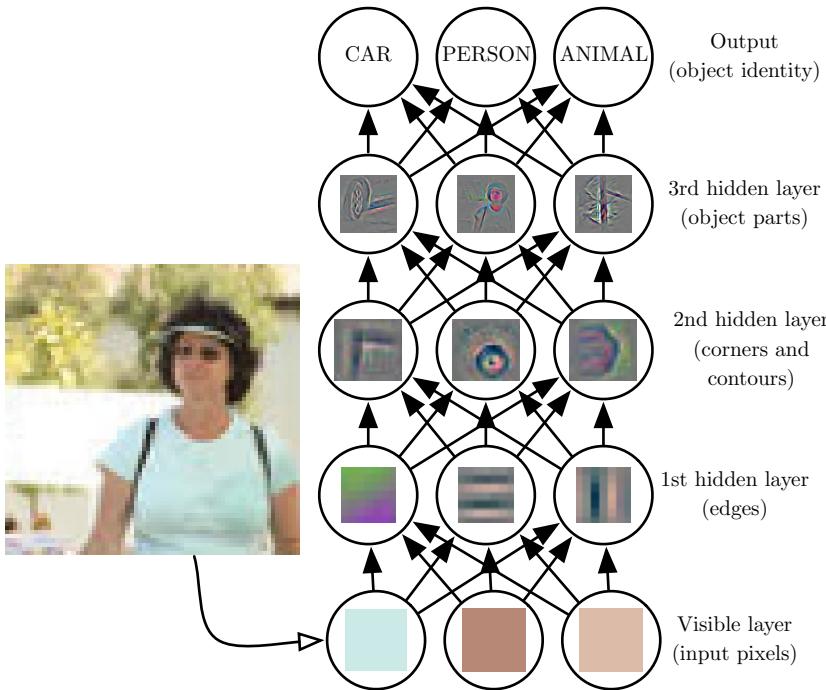


图 1.2: 深度学习模型的示意图。计算机难以理解原始感观输入数据的含义，如表示为像素值集合的图像。将一组像素映射到对象标识的函数非常复杂。如果直接处理，学习或评估此映射似乎是不可能的。深度学习将所需的复杂映射分解为一系列嵌套的简单映射（每个由模型的不同层描述）来解决这一难题。输入展示在可见层 (visible layer)，这样命名的原因是因为它包含我们能观察到的变量。然后是一系列从图像中提取越来越多抽象特征的隐藏层 (hidden layer)。因为它们的值不在数据中给出，所以将这些层称为“隐藏”；模型必须确定哪些概念有利于解释观察数据中的关系。这里的图像是每个隐藏单元表示的特征的可视化。给定像素，第一层可以轻易地通过比较相邻像素的亮度来识别边缘。有了第一隐藏层描述的边缘，第二隐藏层可以容易地搜索可识别为角和扩展轮廓的边集合。给定第二隐藏层中关于角和轮廓的图像描述，第三隐藏层可以找到轮廓和角的特定集合来检测特定对象的整个部分。最后，根据图像描述中包含的对象部分，可以识别图像中存在的对象。经Zeiler and Fergus (2014) 许可转载此图。

像中人的概念。深度学习模型的典型例子是前馈深度网络或多层感知机 (multilayer perceptron, MLP)。多层感知机仅仅是一个将一组输入值映射到输出值的数学函数。该函数由许多较简单的函数复合而成。我们可以认为不同数学函数的每一次应用都为输入提供了新的表示。

学习数据的正确表示的想法是解释深度学习的一个视角。另一个视角是深度促

使计算机学习一个多步骤的计算机程序。每一层表示都可以被认为是并行执行另一组指令之后计算机的存储器状态。更深的网络可以按顺序执行更多的指令。顺序指令提供了极大的能力，因为后面的指令可以参考早期指令的结果。从这个角度上看，在某层激活函数里，并非所有信息都蕴涵着解释输入的变差因素。表示还存储着状态信息，用于帮助程序理解输入。这里的状态信息类似于传统计算机程序中的计数器或指针。它与具体的输入内容无关，但有助于模型组织其处理过程。

目前主要有两种度量模型深度的方式。第一个观点是基于评估架构所需执行的顺序指令的数目。假设我们将模型表示为给定输入后，计算对应输出的流程图，则可以将这张流程图中的最长路径视为模型的深度。正如两个使用不同语言编写的等价程序将具有不同的长度；相同的函数可以被绘制为具有不同深度的流程图，其深度取决于我们可以用来作为一个步骤的函数。图1.3说明了语言的选择如何给相同的架构两个不同的衡量。

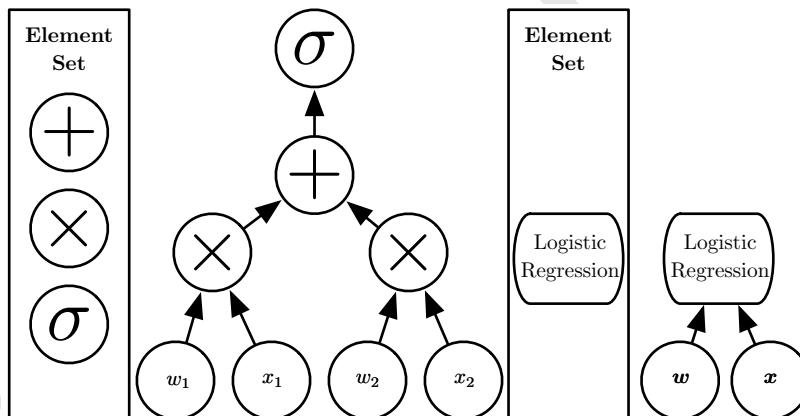


图 1.3: 将输入映射到输出的计算图表的示意图，其中每个节点执行一个操作。深度是从输入到输出的最长路径的长度，但这取决于可能的计算步骤的定义。这些图中所示的计算是逻辑回归模型的输出， $\sigma(w^T x)$ ，其中  $\sigma$  是 logistic sigmoid 函数。如果我们使用加法、乘法和 logistic sigmoid 作为我们计算机语言的元素，那么这个模型深度为三。如果我们将逻辑回归视为元素本身，那么这个模型深度为一。

另一种是在深度概率模型中使用的方法，它不是将计算图的深度视为模型深度，而是将描述概念彼此如何关联的图的深度视为模型深度。在这种情况下，计算每个概念表示的计算流程图的深度可能比概念本身的图更深。这是因为系统对较简单概念的理解在给出更复杂概念的信息后可以进一步精细化。例如，一个AI系统观察其中一只眼睛在阴影中的脸部图像时，它最初可能只看到一只眼睛。但当检测到脸部

的存在后，系统可以推断第二只眼睛也可能是存在的。在这种情况下，概念的图仅包括两层（关于眼睛的层和关于脸的层），但如果我们根据每个概念给出的其他  $n$  次估计进行细化，计算的图将包括  $2n$  层。

由于并不总是清楚计算图的深度或概率模型图的深度哪一个是最重要的，并且由于不同的人选择不同的最小元素集来构建相应的图，因此就像计算机程序的长度不存在单一的正确值一样，架构的深度也不存在单一的正确值。另外，也不存在模型多么深才能被修饰为“深”的共识。但相比传统机器学习，深度学习研究的模型涉及更多学到功能或学到概念的组合，这点毋庸置疑。

总之，这本书的主题——深度学习是通向人工智能的途径之一。具体来说，它是机器学习的一种，一种能够使计算机系统从经验和数据中得到提高的技术。我们坚信机器学习可以构建出在复杂实际环境下运行的AI系统，并且是唯一切实可行的方法。深度学习是一种特定类型的机器学习，具有强大的能力和灵活性，它将大千世界表示为嵌套的层次概念体系（由较简单概念间的联系定义复杂概念、从一般抽象概括到高级抽象表示）。图1.4说明了这些不同的AI学科之间的关系。图1.5展示了每个学科如何工作的高层次原理。

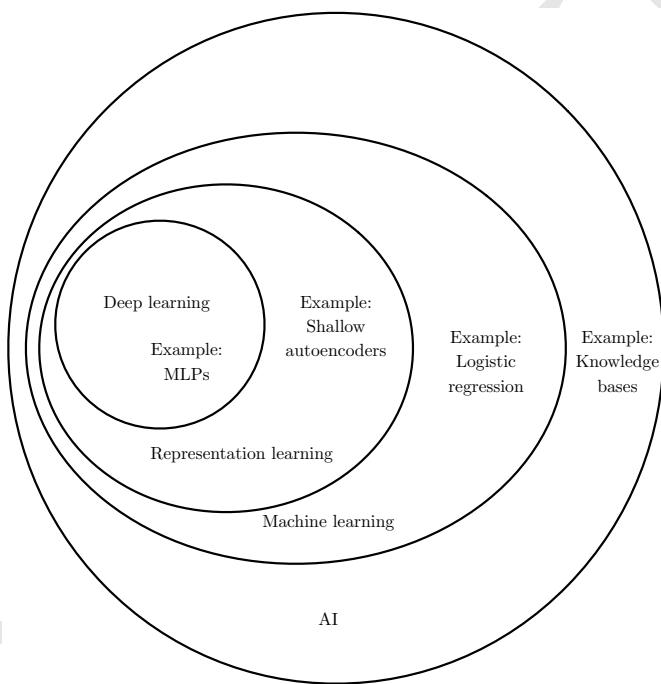


图 1.4: 维恩图展示了深度学习是一种表示学习, 也是一种机器学习, 可以用于许多 (但不是全部) AI方法。维恩图的每个部分包括一个AI技术的示例。

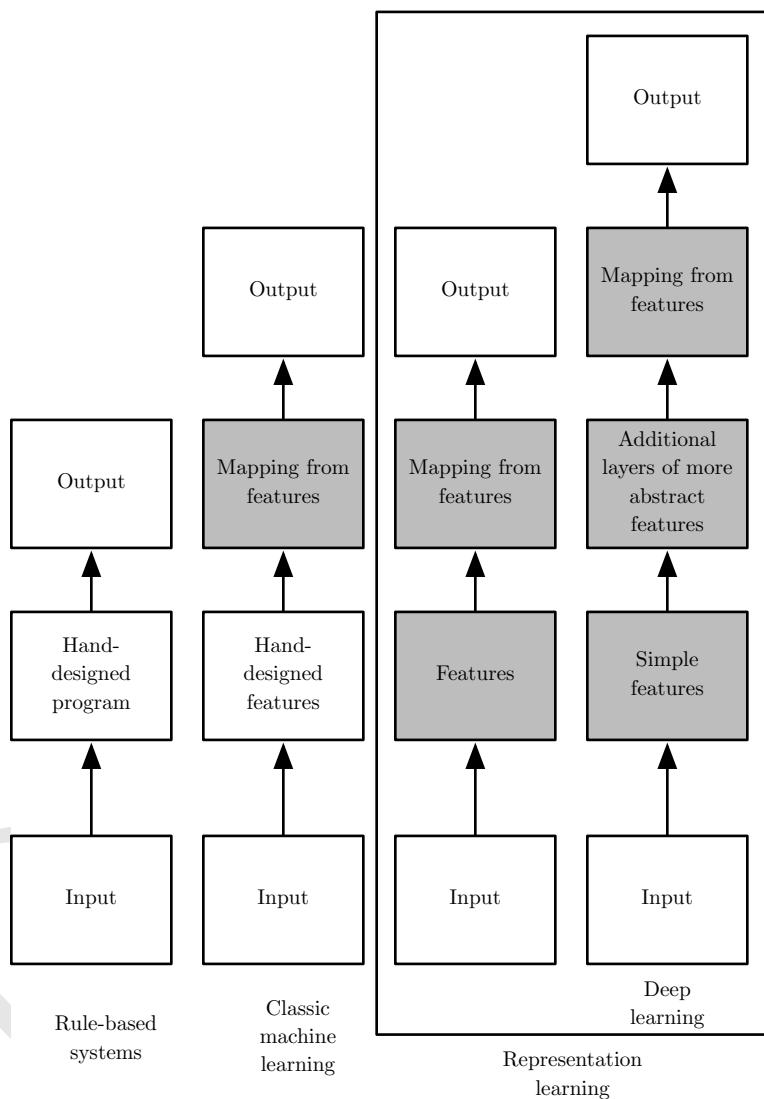


图 1.5: 流程图展示了AI系统的不同部分如何在不同的AI学科中彼此相关。阴影框表示能从数据中学习的组件。

## 1.1 本书面向的读者

这本书对各类读者都有一定用处，但我们主要是为两类受众对象而写的。其中一类受众对象是学习机器学习的大学生（本科或研究生），包括那些已经开始职业生涯的深度学习和人工智能研究者。另一类受众对象是没有机器学习或统计背景但希望能快速地掌握这方面知识并在他们的产品或平台中使用深度学习的软件工程师。深度学习在许多软件领域都已被证明是有用的，包括计算机视觉、语音和音频处理、自然语言处理、机器人技术、生物信息学和化学、电子游戏、搜索引擎、网络广告和金融。

为了最好地服务各类读者，这本书被组织为三个部分。第一部分介绍基本的数学工具和机器学习的概念。第二部分介绍本质上已解决的技术和最成熟的深度学习算法。第三部分讨论某些具有展望性的想法，它们被广泛地认为是深度学习未来的研究重点。

读者可以随意跳过不感兴趣或与自己背景不相关的部分。熟悉线性代数、概率和基本机器学习概念的读者可以跳过第一部分，例如，当读者只是想实现一个能工作的系统则不需要阅读超出第二部分的内容。为了帮助读者选择章节，图1.6展示了这本书的高层组织结构的流程图。

我们假设所有读者都具备计算机科学背景。也假设读者熟悉编程，并且对计算的性能问题、复杂性理论、入门级微积分和一些图论术语有基本的了解。

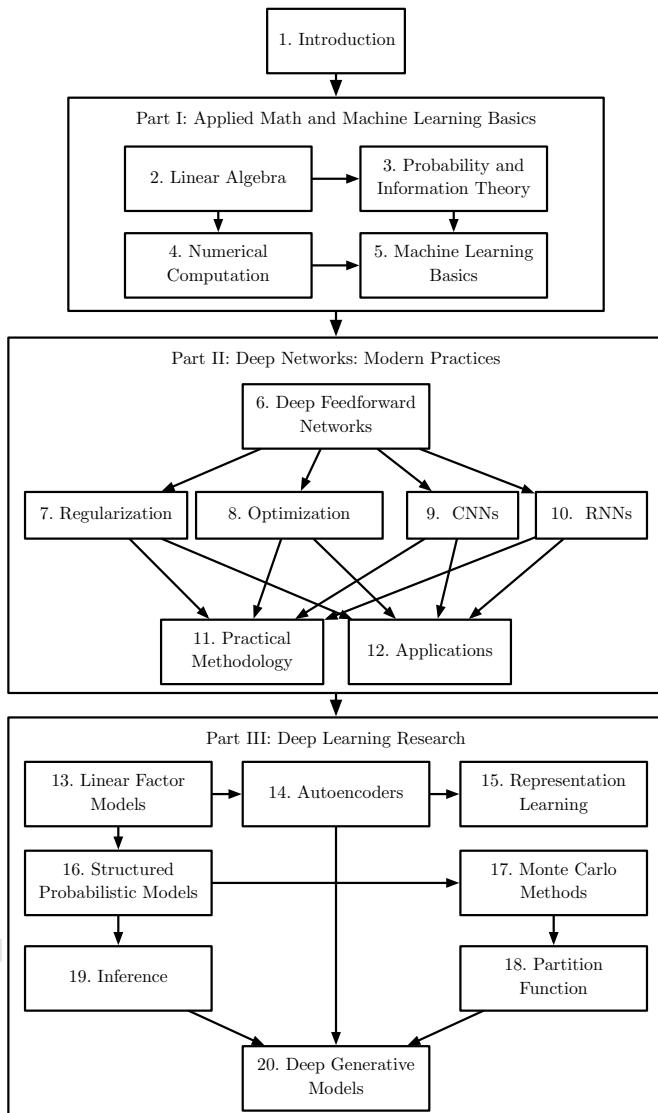


图 1.6: 本书的高层组织。从一章到另一章的箭头表示前一章是理解后一章的必备内容。

## 1.2 深度学习的历史趋势

通过历史背景了解深度学习是最简单的方式。这里我们仅指出深度学习的几个关键趋势，而不是提供其详细的历史：

- 深度学习有着悠久而丰富的历史，但随着许多不同哲学观点的渐渐消逝，与之对应的名称也渐渐尘封。
- 随着可用的训练数据量不断增加，深度学习变得更加有用。
- 随着时间的推移，针对深度学习的计算机软硬件基础设施都有所改善，深度学习模型的规模也随之增长。
- 随着时间的推移，深度学习已经解决日益复杂的应用，并且精度不断提高。

### 1.2.1 神经网络的众多名称和命运变迁

我们期待这本书的许多读者都听说过深度学习这一激动人心的新技术，并对一本书提及一个新兴领域的“历史”而感到惊讶。事实上，深度学习的历史可以追溯到 20 世纪 40 年代。深度学习看似是一个全新的领域，只不过因为在目前流行的前几年它是相对冷门的，同时也因为它被赋予了许多不同的名称（其中大部分已经不再使用），最近才成为众所周知的“深度学习”。这个领域已经更换了很多名称，它反映了不同的研究人员和不同观点的影响。

全面地讲述深度学习的历史超出了本书的范围。然而，一些基本的背景对理解深度学习是有用的。一般来说，目前为止深度学习已经经历了三次发展浪潮：20 世纪 40 年代到 60 年代深度学习的雏形出现在控制论 (cybernetics) 中，20 世纪 80 年代到 90 年代深度学习表现为联结主义 (connectionism)，直到 2006 年，才真正以深度学习之名复兴。图1.7给出了定量的展示。

我们今天知道的一些最早的学习算法，是旨在模拟生物学习的计算模型，即大脑怎样学习或为什么能学习的模型。其结果是深度学习以人工神经网络 (artificial neural network, ANN) 之名而淡去。彼时，深度学习模型被认为是受生物大脑（无论人类大脑或其他动物的大脑）所启发而设计出来的系统。尽管有些机器学习的神经网络有时被用来理解大脑功能 (Hinton and Shallice, 1991)，但它们一般都没有被设计成生物功能的真实模型。深度学习的神经观点受两个主要思想启发。一个想法是大脑作为例子证明智能行为是可能的，因此，概念上，建立智能的直接途径是逆向大脑背后的计算原理，并复制其功能。另一种看法是，理解大脑和人类智能背后的原理也非常有趣，因此机器学习模型除了解决工程应用的能力，如果能让人类对这些基本的科学问题有进一步的认识也将会有用。

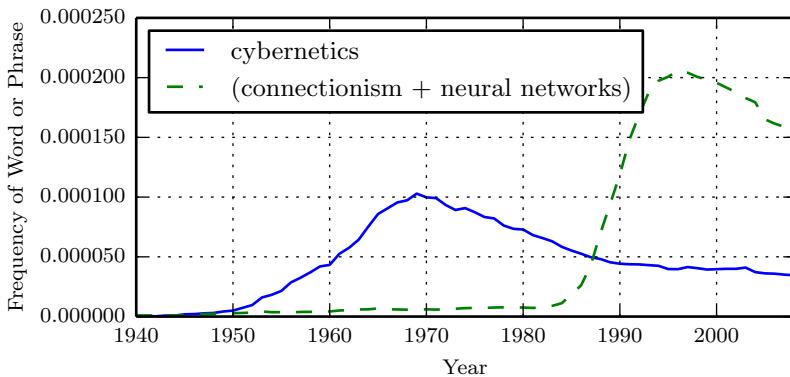


图 1.7: 根据 Google 图书中短语“控制论”、“联结主义”或“神经网络”频率衡量的人工神经网络研究的历史浪潮（图中展示了三次浪潮的前两次，第三次最近才出现）。第一次浪潮开始于 20 世纪 40 年代到 20 世纪 60 年代的控制论，随着生物学习理论的发展 (McCulloch and Pitts, 1943; Hebb, 1949) 和第一个模型的实现（如感知机 (Rosenblatt, 1958)），能实现单个神经元的训练。第二次浪潮开始于 1980-1995 年间的联结主义方法，可以使用反向传播 (Rumelhart *et al.*, 1986a) 训练具有一个或两个隐藏层的神经网络。当前第三次浪潮，也就是深度学习，大约始于 2006 年 (Hinton *et al.*, 2006a; Bengio *et al.*, 2007a; Ranzato *et al.*, 2007a)，并且现在在 2016 年以书的形式出现。另外两次浪潮类似地出现在书中的时间比相应的科学活动晚得多。

现代术语“深度学习”超越了目前机器学习模型的神经科学观点。学习多层次组合这一更普遍的原则更加吸引人，这可以应用于机器学习框架而不必受神经系统启发。

现代深度学习的最早前身是从神经科学的角度出发的简单线性模型。这些模型被设计为使用一组  $n$  个输入  $x_1, \dots, x_n$  并将它们与一个输出  $y$  相关联。这些模型希望学习一组权重  $w_1, \dots, w_n$ ，并计算它们的输出  $f(\mathbf{x}, \mathbf{w}) = x_1 w_1 + \dots + x_n w_n$ 。如图1.7所示，这第一波神经网络研究浪潮被称为控制论。

McCulloch-Pitts神经元 (McCulloch and Pitts, 1943) 是脑功能的早期模型。该线性模型通过检验函数  $f(\mathbf{x}, \mathbf{w})$  的正负来识别两种不同类别的输入。显然，模型的权重需要正确设置后才能使模型的输出对应于期望的类别。这些权重可以由操作人员设定。在 20 世纪 50 年代，感知机 (Rosenblatt, 1956, 1958) 成为第一个能根据每个类别的输入样本来学习权重的模型。约在同一时期，自适应线性单元 (adaptive linear element, ADALINE) 简单地返回函数  $f(\mathbf{x})$  本身的值来预测一个实数 (Widrow and Hoff, 1960)，并且它还可以学习从数据预测这些数。

这些简单的学习算法大大影响了机器学习的现代景象。用于调节 ADALINE 权重的训练算法是被称为随机梯度下降 (stochastic gradient descent) 的一种特例。稍加改进后的随机梯度下降算法仍然是当今深度学习的主要训练算法。

基于感知机和 ADALINE 中使用的函数  $f(\mathbf{x}, \mathbf{w})$  的模型被称为线性模型 (linear model)。尽管在许多情况下，这些模型以不同于原始模型的方式进行训练，但仍是目前最广泛使用的机器学习模型。

线性模型有很多局限性。最著名的是，它们无法学习异或 (XOR) 函数，即  $f([0, 1], \mathbf{w}) = 1$  和  $f([1, 0], \mathbf{w}) = 1$ ，但  $f([1, 1], \mathbf{w}) = 0$  和  $f([0, 0], \mathbf{w}) = 0$ 。观察到线性模型这个缺陷的批评者对受生物学启发的学习普遍地产生了抵触 (Minsky and Papert, 1969)。这导致了神经网络热潮的第一次大衰退。

现在，神经科学被视为深度学习研究的一个重要灵感来源，但它已不再是该领域的主要指导。

如今神经科学在深度学习研究中的作用被削弱，主要原因是我们根本没有足够的关于大脑的信息来作为指导去使用它。要获得对被大脑实际使用算法的深刻理解，我们需要有能力同时监测（至少是）数千相连神经元的活动。我们不能够做到这一点，所以我们甚至连大脑最简单、最深入研究的部分都还远远没有理解 (Olshausen and Field, 2005)。

神经科学已经给了我们依靠单一深度学习算法解决许多不同任务的理由。神经学家们发现，如果将雪貂的大脑重新连接，使视觉信号传送到听觉区域，它们可以学会用大脑的听觉处理区域去“看”(Von Melchner *et al.*, 2000)。这暗示着大多数哺乳动物的大脑能够使用单一的算法就可以解决其大脑可以解决的大部分不同任务。在这个假设之前，机器学习研究是比较分散的，研究人员在不同的社群研究自然语言处理、计算机视觉、运动规划和语音识别。如今，这些应用社群仍然是独立的，但是对于深度学习研究团体来说，同时研究许多或甚至所有这些应用领域是很常见的。

我们能够从神经科学得到一些粗略的指南。仅通过计算单元之间的相互作用而变得智能的基本思想是受大脑启发的。新认知机 (Fukushima, 1980) 受哺乳动物视觉系统的结构启发，引入了一个处理图片的强大模型架构，它后来成为了现代卷积网络的基础 (LeCun *et al.*, 1998b)（我们将会在第9.10节看到）。目前大多数神经网络是基于一个称为整流线性单元 (rectified linear unit) 的神经单元模型。原始认知机 (Fukushima, 1975) 受我们关于大脑功能知识的启发，引入了一个更复杂的版本。简化的现代版通过吸收来自不同观点的思想而形成，Nair and Hinton (2010b)

和Glorot *et al.* (2011a) 援引神经科学作为影响, Jarrett *et al.* (2009a) 援引更多面向工程的影响。虽然神经科学是灵感的重要来源, 但它不需要被视为刚性指导。我们知道, 真实的神经元计算着与现代整流线性单元非常不同的函数, 但更接近真实神经网络的系统并没有导致机器学习性能的提升。此外, 虽然神经科学已经成功地启发了一些神经网络架构, 但我们对用于神经科学的生物学习还没有足够多的了解, 因此也就不能为训练这些架构用的学习算法提供太多的借鉴。

媒体报道经常强调深度学习与大脑的相似性。的确, 深度学习研究者比其他机器学习领域(如核方法或贝叶斯统计)的研究者更可能地引用大脑作为影响, 但是大家不应该认为深度学习在尝试模拟大脑。现代深度学习从许多领域获取灵感, 特别是应用数学的基本内容如线性代数、概率论、信息论和数值优化。尽管一些深度学习的研究人员引用神经科学作为灵感的重要来源, 然而其他学者完全不关心神经科学。

值得注意的是, 了解大脑是如何在算法层面上工作的尝试确实存在且发展良好。这项尝试主要被称为“计算神经科学”, 并且是独立于深度学习的领域。研究人员在两个领域之间来回研究是很常见的。深度学习领域主要关注如何构建计算机系统, 从而成功解决需要智能才能解决的任务, 而计算神经科学领域主要关注构建大脑如何真实工作的比较精确的模型。

在 20 世纪 80 年代, 神经网络研究的第二次浪潮在很大程度上是伴随一个被称为**联结主义**(connectionism)或**并行分布处理**(parallel distributed processing)潮流而出现的(Rumelhart *et al.*, 1986d; McClelland *et al.*, 1995)。联结主义是在认知科学的背景下出现的。认知科学是理解思维的跨学科途径, 即它融合多个不同的分析层次。在 20 世纪 80 年代初期, 大多数认知科学家研究符号推理模型。尽管这很流行, 但符号模型很难解释大脑如何真正使用神经元实现推理功能。联结主义者开始研究真正基于神经系统实现的认知模型(Touretzky and Minton, 1985), 其中很多复苏的想法可以追溯到心理学家Donald Hebb在 20 世纪 40 年代的工作(Hebb, 1949)。

联结主义的中心思想是, 当网络将大量简单的计算单元连接在一起时可以实现智能行为。这种见解同样适用于生物神经系统中的神经元, 因为它和计算模型中隐藏单元起着类似的作用。

在上世纪 80 年代的联结主义期间形成的几个关键概念在今天的深度学习中仍然是非常重要的。

其中一个概念是**分布式表示**(distributed representation)(Hinton *et al.*, 1986)。

其思想是：系统的每一个输入都应该由多个特征表示，并且每一个特征都应该参与到多个可能输入的表示。例如，假设我们有一个能够识别红色、绿色、或蓝色的汽车、卡车和鸟类的视觉系统，表示这些输入的其中一个方法是将九个可能的组合：红卡车，红汽车，红鸟，绿卡车等等使用单独的神经元或隐藏单元激活。这需要九个不同的神经元，并且每个神经必须独立地学习颜色和对象身份的概念。改善这种情况的方法之一是使用分布式表示，即用三个神经元描述颜色，三个神经元描述对象身份。这仅仅需要 6 个神经元而不是 9 个，并且描述红色的神经元能够从汽车、卡车和鸟类的图像中学习红色，而不仅仅是从一个特定类别的图像中学习。分布式表示的概念是本书的核心，我们将在第十五章中更加详细地描述。

联结主义潮流的另一个重要成就是反向传播在训练具有内部表示的深度神经网络中的成功使用以及反向传播算法的普及 (Rumelhart *et al.*, 1986c; LeCun, 1987)。这个算法虽然曾黯然失色不再流行，但截至写书之时，它仍是训练深度模型的主导方法。

在 20 世纪 90 年代，研究人员在使用神经网络进行序列建模的方面取得了重要进展。Hochreiter (1991b) 和Bengio *et al.* (1994a) 指出了对长序列进行建模的一些根本性数学难题，这将在第10.7节中描述。Hochreiter and Schmidhuber (1997) 引入长短期记忆 (long short-term memory, LSTM) 网络来解决这些难题。如今，LSTM 在许多序列建模任务中广泛应用，包括 Google 的许多自然语言处理任务。

神经网络研究的第二次浪潮一直持续到上世纪 90 年代中期。基于神经网络和其他AI技术的创业公司开始寻求投资，其做法野心勃勃但不切实际。当AI研究不能实现这些不合理的期望时，投资者感到失望。同时，机器学习的其他领域取得了进步。比如，核方法 (Boser *et al.*, 1992; Cortes and Vapnik, 1995; Schölkopf *et al.*, 1999) 和图模型 (Jordan, 1998) 都在很多重要任务上实现了很好的效果。这两个因素导致了神经网络热潮的第二次衰退，并一直持续到 2007 年。

在此期间，神经网络继续在某些任务上获得令人印象深刻的表现 (LeCun *et al.*, 1998b; Bengio *et al.*, 2001a)。加拿大高级研究所 (CIFAR) 通过其神经计算和自适应感知 (NCAP) 研究计划帮助维持神经网络研究。该计划联合了分别由Geoffrey Hinton、Yoshua Bengio和Yann LeCun领导的多伦多大学、蒙特利尔大学和纽约大学的机器学习研究小组。这个多学科的 CIFAR NCAP 研究计划还囊括了神经科学家、人类和计算机视觉专家。

在那个时候，人们普遍认为深度网络是难以训练的。现在我们知道，20 世纪 80

年代就存在的算法能工作得非常好，但是直到在 2006 年前后都没有体现出来。这可能仅仅由于其计算代价太高，而以当时可用的硬件难以进行足够的实验。

神经网络研究的第三次浪潮始于 2006 年的突破。Geoffrey Hinton表明名为深度信念网络的神经网络可以使用一种称为贪婪逐层预训练的策略来有效地训练 (Hinton *et al.*, 2006a)，我们将在第15.1节中更详细地描述。其他 CIFAR 附属研究小组很快表明，同样的策略可以被用来训练许多其他类型的深度网络 (Bengio and LeCun, 2007a; Ranzato *et al.*, 2007b)，并能系统地帮助提高在测试样例上的泛化能力。神经网络研究的这一次浪潮普及了“深度学习”这一术语的使用，强调研究者现在有能力训练以前不可能训练的比较深的神经网络，并着力于深度的理论重要性上 (Bengio and LeCun, 2007b; Delalleau and Bengio, 2011; Pascanu *et al.*, 2014a; Montufar *et al.*, 2014)。此时，深度神经网络已经优于与之竞争的基于其他机器学习技术以及手工设计功能的AI系统。在写这本书的时候，神经网络的第三次发展浪潮仍在继续，尽管深度学习的研究重点在这一段时间内发生了巨大变化。第三次浪潮已开始着眼于新的无监督学习技术和深度模型在小数据集的泛化能力，但目前更多的兴趣点仍是比較传统的监督学习算法和深度模型充分利用大型标注数据集的能力。

### 1.2.2 与日俱增的数据量

人们可能想问，既然人工神经网络的第一个实验在 20 世纪 50 年代就完成了，但为什么深度学习直到最近才被认为是关键技术。自 20 世纪 90 年代以来，深度学习就已经成功用于商业应用，但通常被视为是一种艺术而不是一种技术，且只有专家可以使用的艺术，这种观点持续到最近。确实，要从一个深度学习算法获得良好的性能需要一些技巧。幸运的是，随着训练数据的增加，所需的技巧正在减少。目前在复杂的任务达到人类水平的学习算法，与 20 世纪 80 年代努力解决玩具问题 (*toy problem*) 的学习算法几乎是一样的，尽管我们使用这些算法训练的模型经历了变革，即简化了极深架构的训练。最重要的新进展是现在我们有了这些算法得以成功训练所需的资源。图1.8展示了基准数据集的大小如何随着时间的推移而显著增加。这种趋势是由社会日益数字化驱动的。由于我们的活动越来越多发生在计算机上，我们做什么也越来越多地被记录。由于我们的计算机越来越多地联网在一起，这些记录变得更容易集中管理，并更容易将它们整理成适于机器学习应用的数据集。因为统计估计的主要负担（观察少量数据以在新数据上泛化）已经减轻，“大数据”时代

使机器学习更加容易。截至 2016 年，一个粗略的经验法则是，监督深度学习算法在每类给定约 5000 个标注样本情况下一般将达到可以接受的性能，当至少有 1000 万个标注样本的数据集用于训练时，它将达到或超过人类表现。此外，在更小的数据集上获得成功是一个重要的研究领域，为此我们应特别侧重于如何通过无监督或半监督学习充分利用大量的未标注样本。

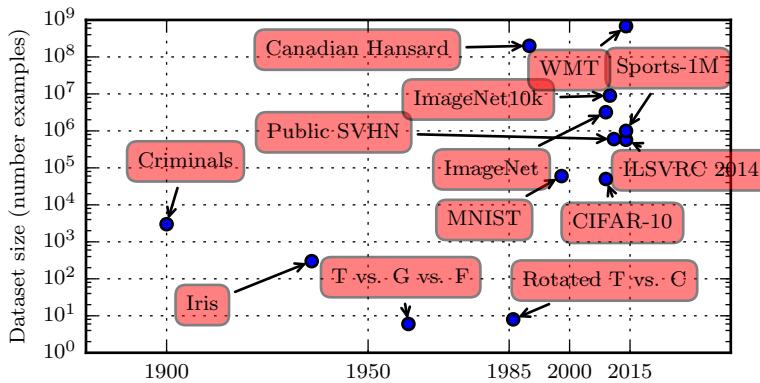


图 1.8: 与日俱增的数据量。20 世纪初，统计学家使用数百或数千的手工制作的度量来研究数据集 (Garson, 1900; Gosset, 1908; Anderson, 1935; Fisher, 1936)。20 世纪 50 年代到 80 年代，受生物启发的机器学习开拓者通常使用小的合成数据集，如低分辨率的字母位图，设计为在低计算成本下表明神经网络能够学习特定功能 (Widrow and Hoff, 1960; Rumelhart *et al.*, 1986b)。20 世纪 80 年代和 90 年代，机器学习变得更加统计，并开始利用包含成千上万个样本的更大数据集，如手写扫描数字的 MNIST 数据集 (如图 1.9 所示 (LeCun *et al.*, 1998b))。在 21 世纪初的第一个十年，相同大小更复杂的数据集持续出现，如 CIFAR-10 数据集 (Krizhevsky and Hinton, 2009)。在这十年结束和下五年，明显更大的数据集（包含数万到数千万的样例）完全改变了深度学习的可能实现的事。这些数据集包括公共 Street View House Numbers 数据集 (Netzer *et al.*, 2011)、各种版本的 ImageNet 数据集 (Deng *et al.*, 2009, 2010a; Russakovsky *et al.*, 2014a) 以及 Sports-1M 数据集 (Karpathy *et al.*, 2014)。在图顶部，我们看到翻译句子的数据集通常远大于其他数据集，如根据 Canadian Hansard 制作的 IBM 数据集 (Brown *et al.*, 1990) 和 WMT 2014 英法数据集 (Schwenk, 2014)。

### 1.2.3 与日俱增的模型规模

20 世纪 80 年代，神经网络只能取得相对较小的成功，而现在神经网络非常成功的另一个重要原因是我们现在拥有的计算资源可以运行更大的模型。联结主义的

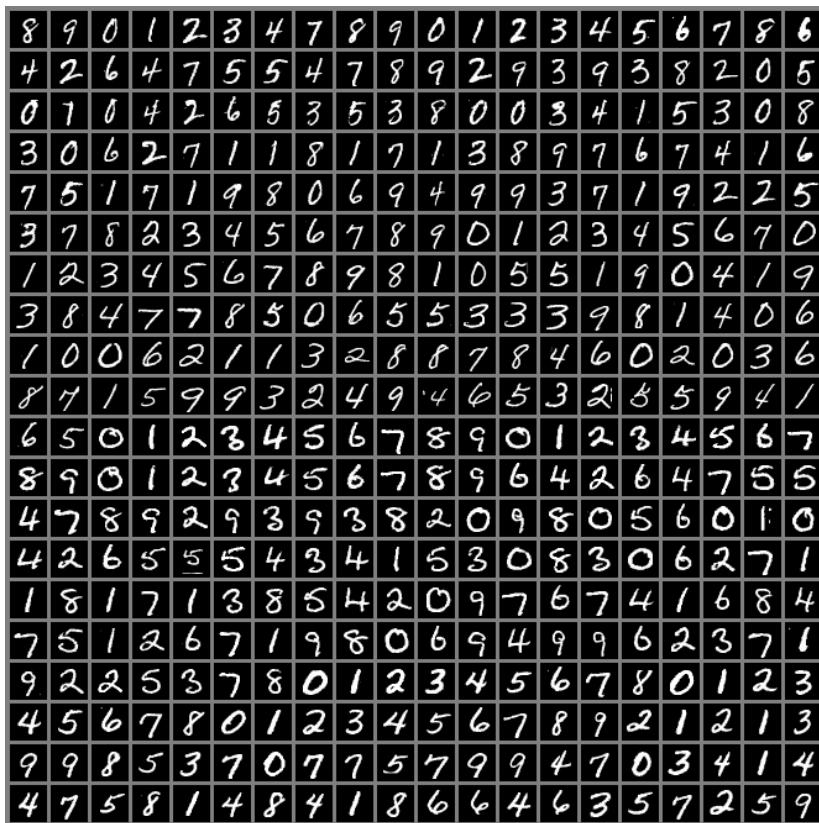


图 1.9: MNIST 数据集的输入样例。“NIST”代表国家标准和技术研究所 (National Institute of Standards and Technology)，是最初收集这些数据的机构。“M”代表“修改的 (Modified)”，为更容易地与机器学习算法一起使用，数据已经过预处理。MNIST 数据集包括手写数字的扫描和相关标签 (描述每个图像中包含 0-9 中哪个数字)。这个简单的分类问题是深度学习研究中最简单和最广泛使用的测试之一。尽管现代技术很容易解决这个问题，它仍然很受欢迎。Geoffrey Hinton 将其描述为“机器学习的果蝇”，这意味着机器学习研究人员可以在受控的实验室条件下研究他们的算法，就像生物学家经常研究果蝇一样。

主要见解之一是，当动物的许多神经元一起工作时会变得聪明。单独神经元或小集合的神经元不是特别有用。

生物神经元不是特别稠密地连接在一起。如图1.10所示，几十年来，我们的机器学习模型中每个神经元的连接数量已经与哺乳动物的大脑在同一数量级上。

如图1.11所示，就神经元的总数目而言，直到最近神经网络都是惊人的小。自

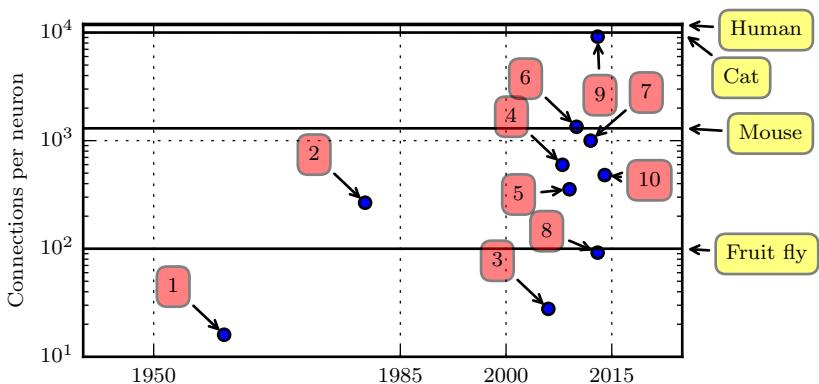


图 1.10: 与日俱增的每神经元连接数。最初，人工神经网络中神经元之间的连接数受限于硬件能力。而现在，神经元之间的连接数大多是出于设计考虑。一些人工神经网络中每个神经元的连接数与猫一样多，并且对于其他神经网络来说，每个神经元的连接与较小哺乳动物（如小鼠）一样多是非常普遍的。甚至人类大脑每个神经元的连接也没有过高的数量。生物神经网络规模来自Wikipedia (2015)。

1. 自适应线性单元 (Widrow and Hoff, 1960)
2. 神经认知机 (Fukushima, 1980)
3. GPU-加速 卷积网络 (Chellapilla *et al.*, 2006)
4. 深度玻尔兹曼机 (Salakhutdinov and Hinton, 2009a)
5. 无监督卷积网络 (Jarrett *et al.*, 2009b)
6. GPU-加速 多层感知机 (Ciresan *et al.*, 2010)
7. 分布式自编码器 (Le *et al.*, 2012)
8. Multi-GPU 卷积网络 (Krizhevsky *et al.*, 2012a)
9. COTS HPC 无监督卷积网络 (Coates *et al.*, 2013)
10. GoogLeNet (Szegedy *et al.*, 2014a)

从隐藏单元引入以来，人工神经网络的规模大约每 2.4 年扩大一倍。这种增长是由更大内存、更快的计算机和更大的可用数据集驱动的。更大的网络能够在更复杂的任务中实现更高的精度。这种趋势看起来将持续数十年。除非有能力迅速扩展的新技术，否则至少要到 21 世纪 50 年代，人工神经网络将才能具备与人脑相同数量级的神经元。生物神经元表示的功能可能比目前的人工神经元所表示的更复杂，因此生物神经网络可能比图中描绘的甚至要更大。

现在看来，其神经元比一个水蛭还少的神经网络不能解决复杂的人工智能问题是不足为奇的。即使现在的网络，从计算系统角度来看它可能相当大的，但实际上

## 1.2 深度学习的历史趋势

21

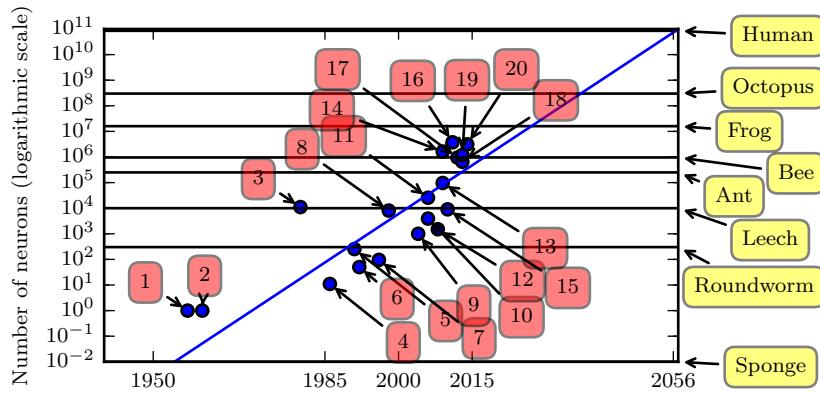


图 1.11: 与日俱增的神经网络规模。自从引入隐藏单元，人工神经网络的大小大约每 2.4 年翻一倍。生物神经网络规模来自 Wikipedia (2015)。

1. 感知机 (Rosenblatt, 1958, 1962)
2. 自适应线性单元 (Widrow and Hoff, 1960)
3. 神经认知机 (Fukushima, 1980)
4. 早期后向传播网络 (Rumelhart *et al.*, 1986b)
5. 用于语音识别的循环神经网络 (Robinson and Fallside, 1991)
6. 用于语音识别的多层感知机 (Bengio *et al.*, 1991)
7. 均匀场sigmoid信念网络 (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. 回声状态网络 (Jaeger and Haas, 2004)
10. 深度信念网络 (Hinton *et al.*, 2006a)
11. GPU-加速卷积网络 (Chellapilla *et al.*, 2006)
12. 深度玻尔兹曼机 (Salakhutdinov and Hinton, 2009a)
13. GPU-加速深度信念网络 (Raina *et al.*, 2009a)
14. 无监督卷积网络 (Jarrett *et al.*, 2009b)
15. GPU-加速多层感知机 (Ciresan *et al.*, 2010)
16. OMP-1 网络 (Coates and Ng, 2011)
17. 分布式自编码器 (Le *et al.*, 2012)
18. Multi-GPU卷积网络 (Krizhevsky *et al.*, 2012a)
19. COTS HPC 无监督卷积网络 (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

它比相对原始的脊椎动物如青蛙的神经系统还要小。

由于更快的 CPU、通用 GPU 的出现（在第12.1.2节中讨论）、更快的网络连接和更好的分布式计算的软件基础设施，模型规模随着时间的推移不断增加是深度学习历史中最重要的趋势之一。普遍预计这种趋势将很好地持续到未来。

### 1.2.4 与日俱增的精度、复杂度和对现实世界的冲击

20世纪80年代以来，深度学习提供精确识别和预测的能力一直在提高。而且，深度学习持续成功地被应用于越来越广泛的实际问题中。

最早的深度模型被用来识别裁剪紧凑且非常小的图像中的单个对象 (Rumelhart *et al.*, 1986d)。此后，神经网络可以处理的图像尺寸逐渐增加。现代对象识别网络能处理丰富的高分辨率照片，并且不需要在被识别的对象附近进行裁剪 (Krizhevsky *et al.*, 2012b)。类似地，最早的网络只能识别两种对象（或在某些情况下，单类对象的存在与否），而这些现代网络通常能够识别至少1000个不同类别的对象。对象识别中最大的比赛是每年举行的 ImageNet 大型视觉识别挑战 (ILSVRC)。深度学习迅速崛起的激动人心的一幕是卷积网络第一次大幅赢得这一挑战，它将最高水准的前 5 错误率从26.1%降到15.3% (Krizhevsky *et al.*, 2012b)，这意味着该卷积网络针对每个图像的可能类别生成一个顺序列表，除了 15.3% 的测试样本，其他测试样本的正确类标都出现在此列表中的前 5 项里。此后，深度卷积网络连续地赢得这些比赛，截至写本书时，深度学习的最新结果将这个比赛中的前 5 错误率降到了3.6%，如图1.12所示。

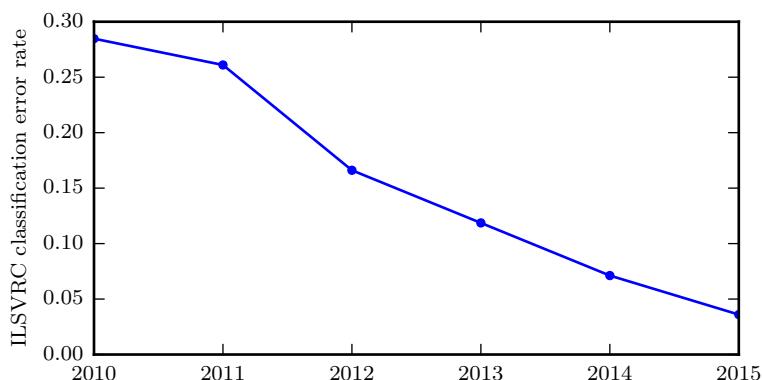


图 1.12: 日益降低的错误率。由于深度网络达到了在 ImageNet 大规模视觉识别挑战中竞争所必需的规模，它们每年都能赢得胜利，并且产生越来越低的错误率。数据来源于 Russakovsky *et al.* (2014b) 和 He *et al.* (2015)。

深度学习也对语音识别产生了巨大影响。语音识别在 20 世纪 90 年代得到提高后，直到约 2000 年都停滞不前。深度学习的引入 (Dahl *et al.*, 2010; Deng *et al.*, 2010b; Seide *et al.*, 2011; Hinton *et al.*, 2012a) 使得语音识别错误率陡然下降，有些

错误率甚至降低了一半。我们将在第12.3节更详细地探讨这个历史。

深度网络在行人检测和图像分割中也取得了引人注目的成功 (Sermanet *et al.*, 2013; Farabet *et al.*, 2013; Couprie *et al.*, 2013)，并且在交通标志分类上取得了超越人类的表现 (Ciresan *et al.*, 2012)。

在深度网络的规模和精度有所提高的同时，它们可以解决的任务也日益复杂。Goodfellow *et al.* (2014d) 表明，神经网络可以学习输出描述图像的整个字符序列，而不是仅仅识别单个对象。此前，人们普遍认为，这种学习需要对序列中的单个元素进行标注 (Gulcehre and Bengio, 2013)。循环神经网络，如之前提到的LSTM序列模型，现在用于对序列和其他序列之间的关系进行建模，而不是仅仅固定输入之间的关系。这种序列到序列的学习似乎引领着另一个应用的颠覆性发展，即机器翻译 (Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015)。

这种复杂性日益增加的趋势已将其推向逻辑结论，即神经图灵机 (Graves *et al.*, 2014) 的引入，它能学习读取存储单元和向存储单元写入任意内容。这样的神经网络可以从期望行为的样本中学习简单的程序。例如，从杂乱和排好序的样本中学习对一系列数进行排序。这种自我编程技术正处于起步阶段，但原则上未来可以适用于几乎所有的任务。

深度学习的另一个最大的成就是其在强化学习 (reinforcement learning) 领域的扩展。在强化学习中，一个自主的智能体必须在没有人类操作者指导的情况下，通过试错来学习执行任务。DeepMind 表明，基于深度学习的强化学习系统能够学会玩 Atari 视频游戏，并在多种任务中可与人类匹敌 (Mnih *et al.*, 2015)。深度学习也显著改善了机器人强化学习的性能 (Finn *et al.*, 2015)。

许多深度学习应用都是高利润的。现在深度学习被许多顶级的技术公司使用，包括 Google、Microsoft、Facebook、IBM、Baidu、Apple、Adobe、Netflix、NVIDIA 和 NEC 等。

深度学习的进步也严重依赖于软件基础架构的进展。软件库如 Theano(Bergstra *et al.*, 2010a; Bastien *et al.*, 2012a)、PyLearn2(Goodfellow *et al.*, 2013e)、Torch(Collobert *et al.*, 2011b)、DistBelief(Dean *et al.*, 2012)、Caffe(Jia, 2013)、MXNet(Chen *et al.*, 2015) 和 TensorFlow(Abadi *et al.*, 2015) 都能支持重要的研究项目或商业产品。

深度学习也为其他科学做出了贡献。用于对象识别的现代卷积网络为神经科学家们提供了可以研究的视觉处理模型 (DiCarlo, 2013)。深度学习也为处理海量数据以及在科学领域作出有效的预测提供了非常有用的工具。它已成功地用于预

测分子如何相互作用从而帮助制药公司设计新的药物 (Dahl *et al.*, 2014)，搜索亚原子粒子 (Baldi *et al.*, 2014)，以及自动解析用于构建人脑三维图的显微镜图像 (Knowles-Barley *et al.*, 2014) 等。我们期待深度学习未来能够出现在越来越多的科学领域中。

总之，深度学习是机器学习的一种方法。在过去几十年的发展中，它大量借鉴了我们关于人脑、统计学和应用数学的知识。近年来，得益于更强大的计算机、更大的数据集和能够训练更深网络的技术，深度学习的普及性和实用性都有了极大的发展。未来几年充满了进一步提高深度学习并将它带到新领域的挑战和机遇。

## 第一部分

# 应用数学与机器学习基础

本书这一部分将介绍理解深度学习所需的基本数学概念。我们从应用数学的一般概念开始，这能使我们定义许多变量的函数，找到这些函数的最高和最低点，并量化信念度。

接着，我们描述机器学习的基本目标，并描述如何实现这些目标。我们需要指定代表某些信念的模型、设计衡量这些信念与现实对应程度的代价函数以及使用训练算法最小化这个代价函数。

这个基本框架是广泛多样的机器学习算法的基础，包括非深度的机器学习方法。在本书的后续部分，我们将在这个框架下开发深度学习算法。

## 第二章 线性代数

线性代数作为数学的一个分支，广泛用于科学和工程中。然而，因为线性代数主要是面向连续数学，而非离散数学，所以很多计算机科学家很少接触它。掌握好线性代数对于理解和从事机器学习算法相关工作是很有必要的，尤其对于深度学习而言。因此，在我们开始介绍深度学习之前，我们集中探讨一些必备的线性代数知识。

如果你已经很熟悉线性代数，那么你可以轻松地跳过本章。如果你先前接触过本章的内容，但是需要一份索引表来回顾一些重要公式，那么我们推荐 *The Matrix Cookbook* (Petersen and Pedersen, 2006)。如果你没有接触过线性代数，那么本章将告诉你本书所需的线性代数知识，不过我们仍然非常建议你参考其他专注于讲解线性代数的文献，例如Shilov (1977)。最后，本章跳过了很多重要但是对于理解深度学习非必需的线性代数知识。

### 2.1 标量、向量、矩阵和张量

学习线性代数，会涉及以下几个数学概念：

- **标量 (scalar)**: 一个标量就是一个单独的数，不同于线性代数中大多数概念会涉及到多个数。我们用斜体表示标量。标量通常赋予小写的变量名称。当我们介绍标量时，会明确它们是哪种类型的数。比如，在定义实数标量时，我们可能会说“让  $s \in \mathbb{R}$  表示一条线的斜率”；在定义自然数标量时，我们可能会说“让  $n \in \mathbb{N}$  表示元素的数目”。
- **向量 (vector)**: 一个向量是一列数。这些数是有序排列的。通过次序中的索引，我们可以确定每个单独的数。通常我们赋予向量粗体的小写变量名称，比如  $\mathbf{x}$ 。向量中的元素可以通过带脚标的斜体表示。向量  $\mathbf{x}$  的第一个元素是  $x_1$ ，第二个

元素是  $x_2$ , 等等。我们也会注明存储在向量中的元素是什么类型的。如果每个元素都属于  $\mathbb{R}$ , 并且该向量有  $n$  个元素, 那么该向量属于实数集  $\mathbb{R}$  笛卡尔乘积  $n$  次, 表示为  $\mathbb{R}^n$ 。当我们需要明确表示向量中的元素时, 我们会将元素排列成一个方括号包围的纵柱:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \quad (2.1)$$

我们可以把向量看作空间中的点, 每个元素是不同的坐标轴上的坐标。

有时我们需要指定向量中某个集合的元素。在这种情况下, 我们定义一个包含这些索引的集合, 然后将该集合写在脚标处。比如, 指定  $x_1$ ,  $x_3$  和  $x_6$ , 我们定义集合  $S = \{1, 3, 6\}$ , 然后写作  $\mathbf{x}_S$ 。我们用符号  $-$  表示集合的补集中的索引。比如  $\mathbf{x}_{-1}$  表示  $\mathbf{x}$  中除  $x_1$  外的所有元素,  $\mathbf{x}_{-S}$  表示  $\mathbf{x}$  中除  $x_1$ ,  $x_3$ ,  $x_6$  外所有元素构成的向量。

- **矩阵 (matrix):** 矩阵是二维数组, 其中的每一个元素被两个索引而非一个所确定。我们通常会赋予矩阵粗体的大写变量名称, 比如  $\mathbf{A}$ 。如果一个实数矩阵高度为  $m$ , 宽度为  $n$ , 那么我们说  $\mathbf{A} \in \mathbb{R}^{m \times n}$ 。我们在表示矩阵中的元素时, 通常使用其名称以不加粗的斜体形式, 索引用逗号间隔。比如,  $A_{1,1}$  表示  $\mathbf{A}$  左上的元素,  $A_{m,n}$  表示  $\mathbf{A}$  右下的元素。我们表示垂直坐标  $i$  中的所有元素时, 用 “ $:$ ” 表示水平坐标。比如,  $\mathbf{A}_{:,i}$  表示  $\mathbf{A}$  中垂直坐标  $i$  上的一横排元素。这也被称为  $\mathbf{A}$  的第  $i$  行 (row)。同样地,  $\mathbf{A}_{:,i}$  表示  $\mathbf{A}$  的第  $i$  列 (column)。当我们需要明确表示矩阵中的元素时, 我们将它们写在用方括号包围起来的数组中:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}. \quad (2.2)$$

有时我们需要矩阵值表达式的索引, 而不是单个元素。在这种情况下, 我们在表达式后面接下标, 但不需要写作小写字母。比如,  $f(\mathbf{A})_{i,j}$  表示函数  $f$  作用在  $\mathbf{A}$  上输出的矩阵的第  $(i, j)$  个元素。

- **张量 (tensor):** 在某些情况下, 我们会讨论不只二维坐标的数组。一般地, 一组数组中的元素分布在若干维坐标的规则网格中, 我们将其称之为张量。我们使用这种字体  $\mathbf{A}$  来表示张量 “ $A$ ”。张量  $\mathbf{A}$  中坐标为  $(i, j, k)$  的元素记作  $A_{i,j,k}$ 。

转置 (transpose) 是矩阵的重要操作之一。矩阵的转置是以对角线为轴的镜像，这条从左上角到右下角的对角线被称为主对角线 (main diagonal)。图2.1显示了这个操作。我们将矩阵  $\mathbf{A}$  的转置表示为  $\mathbf{A}^\top$ ，定义如下

$$(\mathbf{A}^\top)_{i,j} = A_{j,i}. \quad (2.3)$$

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \mathbf{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

图 2.1: 矩阵的转置可以看成是以主对角线为轴的一个镜像。

向量可以看作是只有一列的矩阵。对应地，向量的转置可以看作是只有一行的矩阵。有时，我们将向量表示成行矩阵的转置，写在行中，然后使用转置将其变为标准的列向量，比如  $\mathbf{x} = [x_1, x_2, x_3]^\top$ .

标量可以看作是只有一个元素的矩阵。因此，标量的转置等于它本身， $a = a^\top$ 。

只要矩阵的形状一样，我们可以把两个矩阵相加。两个矩阵相加是指对应位置的元素相加，比如  $\mathbf{C} = \mathbf{A} + \mathbf{B}$ ，其中  $C_{i,j} = A_{i,j} + B_{i,j}$ 。

标量和矩阵相乘，或是和矩阵相加时，我们将其与矩阵的每个元素相乘或相加，比如  $\mathbf{D} = a \cdot \mathbf{B} + c$ ，其中  $D_{i,j} = a \cdot B_{i,j} + c$ 。

在深度学习中，我们也使用一些不那么常规的符号。我们允许矩阵和向量相加，产生另一个矩阵： $\mathbf{C} = \mathbf{A} + \mathbf{b}$ ，其中  $C_{i,j} = A_{i,j} + b_j$ 。换言之，向量  $\mathbf{b}$  和矩阵  $\mathbf{A}$  的每一行相加。这个速记方法使我们无需在加法操作前定义复制向量  $\mathbf{b}$  到矩阵的每一行。这种隐式地复制向量  $\mathbf{b}$  到很多位置的方式，被称为广播 (broadcasting)。

## 2.2 矩阵和向量相乘

矩阵乘法是矩阵运算中最重要的操作之一。两个矩阵  $A$  和  $B$  的矩阵乘积 (matrix product) 是第三个矩阵  $C$ 。为了使乘法定义良好，矩阵  $A$  的列数必须和矩阵  $B$  的行数相等。如果矩阵  $A$  的形状是  $m \times n$ , 矩阵  $B$  的形状是  $n \times p$ , 那么矩阵  $C$  的形状是  $m \times p$ 。矩阵乘积可以作用于两个或多个并在一起的矩阵，例如

$$C = AB. \quad (2.4)$$

具体地，该乘法操作定义为

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}. \quad (2.5)$$

需要注意的是，两个矩阵的标准乘积不是指两个矩阵中对应元素的乘积。不过，那样的矩阵操作确实是存在的，被称为元素对应乘积 (element-wise product) 或者 Hadamard 乘积 (Hadamard product)，表示为  $A \odot B$ 。

两个相同维数的向量  $x$  和  $y$  的点积 (dot product) 可看作是矩阵乘积  $x^\top y$ 。我们可以把矩阵乘积  $C = AB$  中计算  $C_{i,j}$  的步骤看作是  $A$  的第  $i$  行和  $B$  的第  $j$  列之间的点积。

矩阵乘积运算有许多有用的性质，从而使矩阵的数学分析更加方便。比如，矩阵乘积服从分配律：

$$A(B + C) = AB + AC. \quad (2.6)$$

矩阵乘积也服从结合律：

$$A(BC) = (AB)C. \quad (2.7)$$

不同于标量乘积，矩阵乘积并不满足交换律 ( $AB = BA$  的情况并非总是满足)。然而，两个向量的点积 (dot product) 满足交换律：

$$x^\top y = y^\top x. \quad (2.8)$$

矩阵乘积的转置有着简单的形式：

$$(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top. \quad (2.9)$$

利用向量乘积是标量，标量转置是自身的事，我们可以证明式(2.8)：

$$\mathbf{x}^\top \mathbf{y} = (\mathbf{x}^\top \mathbf{y})^\top = \mathbf{y}^\top \mathbf{x}. \quad (2.10)$$

由于本书的重点不是线性代数，我们并不试图展示矩阵乘积的所有重要性质，但读者应该知道矩阵乘积还有很多有用的性质。

现在我们已经知道了足够多的线性代数符号，可以表达下列线性方程组：

$$\mathbf{Ax} = \mathbf{b} \quad (2.11)$$

其中  $\mathbf{A} \in \mathbb{R}^{m \times n}$  是一个已知矩阵， $\mathbf{b}$  是一个已知向量， $\mathbf{x}$  是一个我们要求解的未知向量。向量  $\mathbf{x}$  的每一个元素  $x_i$  都是未知的。矩阵  $\mathbf{A}$  的每一行和  $\mathbf{b}$  中对应的元素构成一个约束。我们可以把式(2.11)重写为

$$\mathbf{A}_{1,:} \mathbf{x} = b_1 \quad (2.12)$$

$$\mathbf{A}_{2,:} \mathbf{x} = b_2 \quad (2.13)$$

$$\dots \quad (2.14)$$

$$\mathbf{A}_{m,:} \mathbf{x} = b_m \quad (2.15)$$

或者，更明确地，写作

$$\mathbf{A}_{1,1}x_1 + \mathbf{A}_{1,2}x_2 + \dots + \mathbf{A}_{1,n}x_n = b_1 \quad (2.16)$$

$$\mathbf{A}_{2,1}x_1 + \mathbf{A}_{2,2}x_2 + \dots + \mathbf{A}_{2,n}x_n = b_2 \quad (2.17)$$

$$\dots \quad (2.18)$$

$$\mathbf{A}_{m,1}x_1 + \mathbf{A}_{m,2}x_2 + \dots + \mathbf{A}_{m,n}x_n = b_m. \quad (2.19)$$

矩阵向量乘积符号为这种形式的等式提供了更紧凑的表示。

## 2.3 单位矩阵和逆矩阵

线性代数提供了被称为矩阵逆 (matrix inverse) 的强大工具，使我们能够解析地求解具有多值矩阵  $\mathbf{A}$  的式(2.11)。

为了描述矩阵逆，我们首先需要定义单位矩阵 (identity matrix) 的概念。任意向量和单位矩阵相乘，都不会改变。我们将保持  $n$  维向量不变的单位矩阵记作  $\mathbf{I}_n$ 。形式上， $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ ，

$$\forall \mathbf{x} \in \mathbb{R}^n, \mathbf{I}_n \mathbf{x} = \mathbf{x}. \quad (2.20)$$

单位矩阵的结构很简单：所有沿主对角线的元素都是 1，而所有其他位置的元素都是 0。如图2.2所示。

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.21)$$

图 2.2: 单位矩阵的一个样例：这是  $\mathbf{I}_3$ 。

矩阵  $\mathbf{A}$  的矩阵逆 (matrix inverse) 记作  $\mathbf{A}^{-1}$ ，定义为如下的矩阵

$$\mathbf{A}^{-1} \mathbf{A} = \mathbf{I}_n. \quad (2.22)$$

现在我们可以通过以下步骤求解式(2.11)：

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (2.23)$$

$$\mathbf{A}^{-1} \mathbf{A}\mathbf{x} = \mathbf{A}^{-1} \mathbf{b} \quad (2.24)$$

$$\mathbf{I}_n \mathbf{x} = \mathbf{A}^{-1} \mathbf{b} \quad (2.25)$$

$$\mathbf{x} = \mathbf{A}^{-1} \mathbf{b}. \quad (2.26)$$

当然，这取决于能否找到一个逆矩阵  $\mathbf{A}^{-1}$ 。在接下来的章节中，我们会讨论逆矩阵  $\mathbf{A}^{-1}$  存在的条件。

当逆矩阵  $\mathbf{A}^{-1}$  存在时，有几种不同的算法都能找到它的闭解形式。理论上，相同的逆矩阵可用于多次求解不同向量  $\mathbf{b}$  的方程。然而，逆矩阵  $\mathbf{A}^{-1}$  主要是作为理论工具使用的，并不会在实际中使用于大多数软件应用程序中。这是因为逆矩阵  $\mathbf{A}^{-1}$  在数字计算机上只能表现出有限的精度，有效使用向量  $\mathbf{b}$  的算法通常可以得到更精确的  $\mathbf{x}$ 。

## 2.4 线性相关和生成子空间

如果逆矩阵  $\mathbf{A}^{-1}$  存在，那么式(2.11)肯定对于每一个向量  $\mathbf{b}$  恰好存在一个解。对系统方程而言，对于某些  $\mathbf{b}$  的值，有可能不存在解，或者存在无限多个解。然而，存在多于一个解但是少于无限多个解的情况是不可能发生的；因为如果  $\mathbf{x}$  和  $\mathbf{y}$  都是某系统方程的解，则

$$\mathbf{z} = \alpha \mathbf{x} + (1 - \alpha) \mathbf{y} \quad (2.27)$$

(其中  $\alpha$  取任意实数) 也是该系统方程的解。

分析方程有多少个解，我们可以将  $\mathbf{A}$  的列向量看作是从原点 (origin) (元素都是零的向量) 出发的不同方向，确定有多少种方法可以到达向量  $\mathbf{b}$ 。在这个观点中，向量  $\mathbf{x}$  中的每个元素表示我们应该沿着这些方向走多远，即  $x_i$  表示我们需要沿着第  $i$  个向量的方向走多远：

$$\mathbf{Ax} = \sum_i x_i \mathbf{A}_{:,i}. \quad (2.28)$$

一般而言，这种操作被称为线性组合 (linear combination)。形式上，某个集合中向量的线性组合，是指每个向量乘以对应系数之后的和，即：

$$\sum_i c_i \mathbf{v}^{(i)}. \quad (2.29)$$

一组向量的生成子空间 (span) 是原始向量线性组合后所能抵达的点的集合。

确定  $\mathbf{Ax} = \mathbf{b}$  是否有解相当于确定向量  $\mathbf{b}$  是否在  $\mathbf{A}$  列向量的生成子空间中。这个特殊的生成子空间被称为  $\mathbf{A}$  的列空间 (column space) 或者  $\mathbf{A}$  的值域 (range)。

为了让方程  $\mathbf{Ax} = \mathbf{b}$  对于任意的向量  $\mathbf{b} \in \mathbb{R}^m$  都有解的话，我们要求  $\mathbf{A}$  的列空间构成整个  $\mathbb{R}^m$ 。如果  $\mathbb{R}^m$  中的某个点不在  $\mathbf{A}$  的列空间中，那么该点对应的  $\mathbf{b}$  会使得该方程没有解。矩阵  $\mathbf{A}$  的列空间是整个  $\mathbb{R}^m$  的要求，意味着  $\mathbf{A}$  至少有  $m$  列，即  $n \geq m$ 。否则， $\mathbf{A}$  列空间的维数会小于  $m$ 。例如，假设  $\mathbf{A}$  是一个  $3 \times 2$  的矩阵。目标  $\mathbf{b}$  是 3 维的，但是  $\mathbf{x}$  只有 2 维。所以无论如何修改  $\mathbf{x}$  的值，也只能描绘出  $\mathbb{R}^3$  空间中的二维平面。当且仅当向量  $\mathbf{b}$  在该二维平面中时，该方程有解。

不等式  $n \geq m$  仅是方程对每一点都有解的必要条件。这不是一个充分条件，因为有些列向量可能是冗余的。假设有一个  $\mathbb{R}^{2 \times 2}$  中的矩阵，它的两个列向量是相同的。那么它的列空间和它的一个列向量作为矩阵的列空间是一样的。换言之，虽然该矩阵有 2 列，但是它的列空间仍然只是一条线，不能涵盖整个  $\mathbb{R}^2$  空间。

正式地，这种冗余被称为线性相关 (linear dependence)。如果一组向量中的任意一个向量都不能表示成其他向量的线性组合，那么这组向量被称为线性无关 (linearly independent)。如果某个向量是一组向量中某些向量的线性组合，那么我们将这个向量加入到这组向量后不会增加这组向量的生成子空间。这意味着，如果一个矩阵的列空间涵盖整个  $\mathbb{R}^m$ ，那么该矩阵必须包含至少一组  $m$  个线性无关的向量。这是式(2.11)对于每一个向量  $b$  的取值都有解的充分必要条件。值得注意的是，这个条件是说该向量集恰好有  $m$  个线性无关的列向量，而不是至少  $m$  个。不存在一个  $m$  维向量的集合具有多于  $m$  个彼此线性不相关的列向量，但是一个有多于  $m$  个列向量的矩阵却有可能拥有不止一个大小为  $m$  的线性无关向量集。

要想使矩阵可逆，我们还需要保证式(2.11)对于每一个  $b$  值至多有一个解。为此，我们需要确保该矩阵至多有  $m$  个列向量。否则，该方程会有不止一个解。

综上所述，这意味着该矩阵必须是一个方阵 (square)，即  $m = n$ ，并且所有列向量都是线性无关的。一个列向量线性相关的方阵被称为奇异的 (singular)。

如果矩阵  $A$  不是一个方阵或者是一个奇异的方阵，该方程仍然可能有解。但是我们不能使用逆矩阵去求解。

目前为止，我们已经讨论了逆矩阵左乘。我们也可以定义逆矩阵右乘：

$$AA^{-1} = I. \quad (2.30)$$

对于方阵而言，它的左逆和右逆是相等的。

## 2.5 范数

有时我们需要衡量一个向量的大小。在机器学习中，我们经常使用被称为范数 (norm) 的函数衡量向量大小。形式上， $L^p$  范数定义如下

$$\|x\|_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}} \quad (2.31)$$

其中  $p \in \mathbb{R}$ ,  $p \geq 1$ 。

范数（包括  $L^p$  范数）是将向量映射到非负值的函数。直观上来说，向量  $x$  的范数是衡量从原点到点  $x$  的距离。更严格地说，范数是满足下列性质的任意函数：

- $f(x) = 0 \Rightarrow x = \mathbf{0}$

- $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$  ( 三角不等式 (triangle inequality) )
- $\forall \alpha \in \mathbb{R}, f(\alpha \mathbf{x}) = |\alpha| f(\mathbf{x})$

当  $p = 2$  时,  $L^2$  被称为欧几里得范数 (Euclidean norm)。它表示从原点出发到向量  $\mathbf{x}$  确定的点的欧几里得距离。 $L^2$  范数十分频繁地出现在机器学习中, 经常简化表示为  $\|\mathbf{x}\|$ , 略去了下标 2。平方  $L^2$  范数也经常用来衡量向量的大小, 可以简单地通过点积  $\mathbf{x}^\top \mathbf{x}$  计算。

平方  $L^2$  范数在数学和计算上都比  $L^2$  范数本身更方便。例如, 平方  $L^2$  范数对  $\mathbf{x}$  中每个元素的导数只取决于对应的元素, 而  $L^2$  范数对每个元素的导数却和整个向量相关。但是在很多情况下, 平方  $L^2$  范数也可能不受欢迎, 因为它在原点附近增长得十分缓慢。在某些机器学习应用中, 区分元素值恰好是零还是非零小值是很重要的。在这些情况下, 我们更倾向于使用在各个位置斜率相同, 同时保持简单的数学形式的函数:  $L^1$  范数。 $L^1$  范数可以简化如下:

$$\|\mathbf{x}\|_1 = \sum_i |x_i|. \quad (2.32)$$

当机器学习问题中零和非零元素之间的差异非常重要时, 通常会使用  $L^1$  范数。每当  $\mathbf{x}$  中某个元素从 0 增加  $\epsilon$ , 对应的  $L^1$  范数也会增加  $\epsilon$ 。

有时候我们会统计向量中非零元素的个数来衡量向量的大小。有些作者将这种函数称为 “ $L^0$  范数”, 但是这个术语在数学意义上是不对的。向量的非零元素的数目不是范数, 因为对标量放缩  $\alpha$  倍不会改变该向量非零的数目。因此,  $L^1$  范数经常作为表示非零元素数目的替代函数。

另外一个经常在机器学习中出现的范数是  $L^\infty$  范数, 也被称为 max 范数 (max norm)。这个范数表示向量中具有最大幅度的元素的绝对值:

$$\|\mathbf{x}\|_\infty = \max_i |x_i|. \quad (2.33)$$

有时候我们可能也需要衡量矩阵的大小。在深度学习中, 最常见的做法是使用 Frobenius 范数 (Frobenius norm),

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}, \quad (2.34)$$

类似于向量的  $L^2$  范数。

两个向量的点积 (dot product) 可以用范数来表示。具体地，

$$\mathbf{x}^\top \mathbf{y} = \|\mathbf{x}\|_2 \|\mathbf{y}\|_2 \cos \theta \quad (2.35)$$

其中  $\theta$  表示  $\mathbf{x}$  和  $\mathbf{y}$  之间的夹角。

## 2.6 特殊类型的矩阵和向量

有些特殊类型的矩阵和向量是特别有用的。

**对角矩阵** (diagonal matrix) 只在主对角线上含有非零元素，其他位置都是零。形式上，矩阵  $\mathbf{D}$  是对角矩阵，当且仅当对于所有的  $i \neq j$ ,  $D_{i,j} = 0$ 。我们已经看到过一个对角矩阵：单位矩阵，对角元素全部是 1。我们用  $\text{diag}(\mathbf{v})$  表示一个对角元素由向量  $\mathbf{v}$  中元素给定的对角方阵。对角矩阵受到关注，部分原因是对角矩阵的乘法计算很高效。计算乘法  $\text{diag}(\mathbf{v})\mathbf{x}$ ，我们只需要将  $\mathbf{x}$  中的每个元素  $x_i$  放大  $v_i$  倍。换言之， $\text{diag}(\mathbf{v})\mathbf{x} = \mathbf{v} \odot \mathbf{x}$ 。计算对角矩阵的逆矩阵也很高效。对角矩阵的逆矩阵存在，当且仅当对角元素都是非零值，在这种情况下， $\text{diag}(\mathbf{v})^{-1} = \text{diag}([1/v_1, \dots, 1/v_n]^\top)$ 。在很多情况，我们可以根据任意矩阵导出一些通用的机器学习算法；但通过将一些矩阵限制为对角矩阵，我们可以得到计算代价较低的（并且描述语言较少的）算法。

不是所有的对角矩阵都是方阵。长方形的矩阵也有可能是对角矩阵。长方形对角矩阵没有逆矩阵，但我们仍然可以很快地计算它们的乘法。对于一个长方形对角矩阵  $\mathbf{D}$  而言，乘法  $\mathbf{D}\mathbf{x}$  会涉及到  $\mathbf{x}$  中每个元素的放缩，如果  $\mathbf{D}$  是瘦长型矩阵，那么放缩后末尾添加一些零；如果  $\mathbf{D}$  是胖宽型矩阵，那么放缩后去掉最后一些元素。

**对称** (symmetric) 矩阵是转置和自己相等的矩阵：

$$\mathbf{A} = \mathbf{A}^\top. \quad (2.36)$$

当某些不依赖参数顺序的双参数函数生成元素时，对称矩阵经常会出现。例如，如果  $\mathbf{A}$  是一个表示距离的矩阵， $A_{i,j}$  表示点  $i$  到点  $j$  的距离，那么  $A_{i,j} = A_{j,i}$ ，因为距离函数是对称的。

**单位向量** (unit vector) 是具有单位范数 (unit norm) 的向量：

$$\|\mathbf{x}\|_2 = 1. \quad (2.37)$$

如果  $\mathbf{x}^\top \mathbf{y} = 0$ ，那么向量  $\mathbf{x}$  和向量  $\mathbf{y}$  互相正交 (orthogonal)。如果两个向量都有非零范数，那么这两个向量之间的夹角是 90 度。在  $\mathbb{R}^n$  中，至多有  $n$  个范数非零

向量互相正交。如果这些向量不仅互相正交，并且范数都为 1，那么我们称它们是标准正交 (orthonormal)。

正交矩阵 (orthogonal matrix) 是指行向量是标准正交的，列向量是标准正交的方阵：

$$\mathbf{A}^\top \mathbf{A} = \mathbf{A} \mathbf{A}^\top = \mathbf{I}. \quad (2.38)$$

这意味着

$$\mathbf{A}^{-1} = \mathbf{A}^\top, \quad (2.39)$$

所以正交矩阵受到关注是因为求逆计算代价小。需要注意正交矩阵的定义。反直觉地，正交矩阵的行向量不仅是正交的，还是标准正交的。对于行向量或列向量互相正交但不是标准正交的矩阵没有对应的专有术语。

## 2.7 特征分解

许多数学对象可以通过将它们分解成多个组成部分，或者找到它们的一些属性而更好地理解，这些属性是通用的，而不是由我们选择表示它们的方式引起的。

例如，整数可以分解为质数。我们可以用十进制或二进制等不同方式表示整数 12，但质因数分解永远是对的  $12 = 2 \times 3 \times 3$ 。从这个表示中我们可以获得一些有用的信息，比如 12 不能被 5 整除，或者 12 的倍数可以被 3 整除。

正如我们可以通过分解质因数来发现整数的一些内在性质，我们也可以通过分解矩阵来发现矩阵表示成数组元素时不明显的函数性质。

特征分解 (eigendecomposition) 是使用最广的矩阵分解之一，即我们将矩阵分解成一组特征向量和特征值。

方阵  $\mathbf{A}$  的特征向量 (eigenvector) 是指与  $\mathbf{A}$  相乘后相当于对该向量进行放缩的非零向量  $\mathbf{v}$ :

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}. \quad (2.40)$$

标量  $\lambda$  被称为这个特征向量对应的特征值 (eigenvalue)。（类似地，我们也可以定义左奇异向量 (left singular vector)  $\mathbf{v}^\top \mathbf{A} = \lambda\mathbf{v}^\top$ ，但是通常我们更关注右奇异向量 (right singular vector)）。

如果  $\mathbf{v}$  是  $\mathbf{A}$  的特征向量，那么任何放缩后的向量  $s\mathbf{v}(s \in \mathbb{R}, s \neq 0)$  也是  $\mathbf{A}$  的特征向量。此外， $s\mathbf{v}$  和  $\mathbf{v}$  有相同的特征值。基于这个原因，通常我们只考虑单位特

征向量。

假设矩阵  $\mathbf{A}$  有  $n$  个线性无关的特征向量  $\{\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}\}$ , 对应着特征值  $\{\lambda_1, \dots, \lambda_n\}$ 。我们将特征向量连接一个矩阵, 使得每一列是一个特征向量:  $V = [\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}]$ . 类似地, 我们也可以将特征值连接成一个向量  $\boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_n]^\top$ 。因此  $\mathbf{A}$  的特征分解 (eigendecomposition) 可以记作

$$\mathbf{A} = \mathbf{V}\text{diag}(\boldsymbol{\lambda})\mathbf{V}^{-1}. \quad (2.41)$$

我们已经看到了构建具有特定特征值和特征向量的矩阵, 能够使我们在目标方向上延伸空间。然而, 我们也常常希望将矩阵分解 (decompose) 成特征值和特征向量。这样可以帮助我们分析矩阵的特定性质, 就像质因数分解有助于我们理解整数。

不是每一个矩阵都可以分解成特征值和特征向量。在某些情况下, 特征分解会涉及到复数, 而非实数。幸运的是, 在本书中我们通常只需要探讨一类有简单分解的矩阵。具体地, 每个实对称矩阵都可以分解成实特征向量和实特征值:

$$\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top. \quad (2.42)$$

其中  $\mathbf{Q}$  是  $\mathbf{A}$  的特征向量组成的正交矩阵,  $\boldsymbol{\Lambda}$  是对角矩阵。特征值  $\Lambda_{i,i}$  对应的特征向量是矩阵  $\mathbf{Q}$  的第  $i$  列, 记作  $\mathbf{Q}_{:,i}$ 。因为  $\mathbf{Q}$  是正交矩阵, 我们可以将  $\mathbf{A}$  看作是沿方向  $\mathbf{v}^{(i)}$  延展  $\lambda_i$  倍的空间。如图2.3所示。

虽然任意一个实对称矩阵  $\mathbf{A}$  都有特征分解, 但是特征分解可能并不唯一。如果两个或多个特征向量拥有相同的特征值, 那么这组特征向量生成子空间中, 任意一组正交向量都是该特征值对应的特征向量。因此, 我们可以使用任意一组正交向量作为  $\mathbf{Q}$  的特征向量。按照惯例, 我们通常按降序排列  $\boldsymbol{\Lambda}$  的条目。在该约定下, 特征分解唯一当且仅当所有的特征值都是唯一的。

矩阵的特征分解给了我们很多关于矩阵的有用信息。矩阵是奇异的当且仅当含有零特征值。实对称矩阵的特征分解也可以用于优化二次方程  $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$ , 其中限制  $\|\mathbf{x}\|_2 = 1$ 。当  $\mathbf{x}$  等于  $\mathbf{A}$  的某个特征向量时,  $f$  将返回对应的特征值。在限制条件下, 函数  $f$  的最大值是最大特征值, 最小值是最小特征值。

所有特征值都是正数的矩阵被称为正定 (positive definite); 所有特征值都是非负数的矩阵被称为半正定 (positive semidefinite)。同样地, 所有特征值都是负数的矩阵被称为负定 (negative definite); 所有特征值都是非正数的矩阵被称为半负定 (negative semidefinite)。半正定矩阵受到关注是因为它们保证  $\forall \mathbf{x}, \mathbf{x}^\top \mathbf{A} \mathbf{x} \geq 0$ 。此外, 正定矩阵还保证  $\mathbf{x}^\top \mathbf{A} \mathbf{x} = 0 \Rightarrow \mathbf{x} = \mathbf{0}$ 。

Effect of eigenvectors and eigenvalues

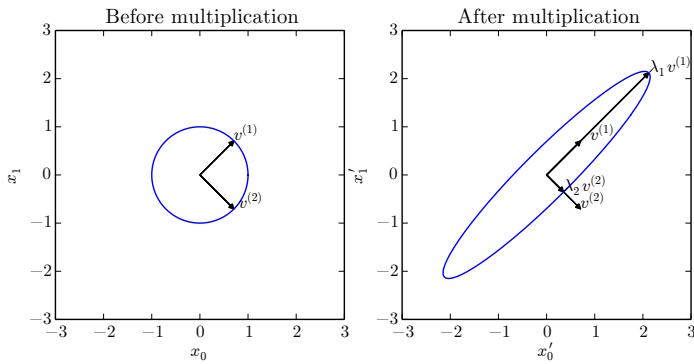


图 2.3: 特征向量和特征值的作用效果。特征向量和特征值的作用效果的一个实例。在这里，矩阵  $A$  有两个标准正交的特征向量，对应特征值为  $\lambda_1$  的  $v^{(1)}$  以及对应特征值为  $\lambda_2$  的  $v^{(2)}$ 。（左）我们画出了所有的单位向量  $u \in \mathbb{R}^2$  的集合，构成一个单位圆。（右）我们画出了所有的  $Au$  点的集合。通过观察  $A$  拉伸单位圆的方式，我们可以看到它能够将  $v^{(2)}$  方向的空间拉伸到  $\lambda_i$ 。

## 2.8 奇异值分解

在第2.7节，我们探讨了如何将矩阵分解成特征向量和特征值。还有另一种分解矩阵的方法，被称为奇异值分解 (singular value decomposition, SVD)，将矩阵分解为奇异向量 (singular vector) 和奇异值 (singular value)。通过奇异值分解，我们会得到一些类似特征分解的信息。然而，奇异值分解有更广泛的应用。每个实数矩阵都有一个奇异值分解，但不一定都有特征分解。例如，非方阵的矩阵没有特征分解，这时我们只能使用奇异值分解。

回想一下，我们使用特征分解去分析矩阵  $A$  时，得到特征向量构成的矩阵  $V$  和特征值构成的向量  $\lambda$ ，我们可以重新将  $A$  写作

$$A = V \text{diag}(\lambda) V^{-1}. \quad (2.43)$$

奇异值分解是类似的，只不过这回我们将矩阵  $A$  分解成三个矩阵的乘积：

$$A = UDV^\top. \quad (2.44)$$

假设  $A$  是一个  $m \times n$  的矩阵，那么  $U$  是一个  $m \times m$  的矩阵， $D$  是一个  $m \times n$  的矩阵， $V$  是一个  $n \times n$  矩阵。

这些矩阵每一个都拥有特殊的结构。矩阵  $U$  和  $V$  都是正交矩阵，矩阵  $D$  是对角矩阵。注意，矩阵  $D$  不一定是方阵。

对角矩阵  $D$  对角线上的元素被称为矩阵  $A$  的奇异值。矩阵  $U$  的列向量被称为左奇异向量 (left singular vector)，矩阵  $V$  的列向量被称为右奇异向量 (right singular vector)。

事实上，我们可以用与  $A$  相关的特征分解去解释  $A$  的奇异值分解。 $A$  的左奇异向量 (left singular vector) 是  $AA^\top$  的特征向量。 $A$  的右奇异向量 (right singular vector) 是  $A^\top A$  的特征向量。 $A$  的非零奇异值是  $A^\top A$  特征值的平方根，同时也是  $AA^\top$  特征值的平方根。

SVD最有用的一个性质可能是拓展矩阵求逆到非方矩阵上。我们将在下一节中探讨。

## 2.9 Moore-Penrose 伪逆

对于非方矩阵而言，其逆矩阵没有定义。假设在下面的问题中，我们希望通过矩阵  $A$  的左逆  $B$  来求解线性方程，

$$Ax = y \quad (2.45)$$

等式两边左乘左逆  $B$  后，我们得到

$$x = By. \quad (2.46)$$

是否存在一个唯一的映射，将  $A$  映射到  $B$ ，取决于问题的形式。

如果矩阵  $A$  的行数大于列数，那么上述方程可能没有解。如果矩阵  $A$  的行数小于列数，那么上述矩阵可能有多个解。

Moore-Penrose 伪逆 (Moore-Penrose pseudoinverse) 使我们能够解决这类问题。矩阵  $\mathbf{A}$  的伪逆定义为：

$$\mathbf{A}^+ = \lim_{\alpha \searrow 0} (\mathbf{A}^\top \mathbf{A} + \alpha \mathbf{I})^{-1} \mathbf{A}^\top. \quad (2.47)$$

计算伪逆的实际算法没有基于这个式子，而是使用下面的公式：

$$\mathbf{A}^+ = \mathbf{V} \mathbf{D}^+ \mathbf{U}^\top. \quad (2.48)$$

其中，矩阵  $\mathbf{U}$ ,  $\mathbf{D}$  和  $\mathbf{V}$  是矩阵  $\mathbf{A}$  奇异值分解后得到的矩阵。对角矩阵  $\mathbf{D}$  的伪逆  $\mathbf{D}^+$  是其非零元素取倒数之后再转置得到的。

当矩阵  $\mathbf{A}$  的列数多于行数时，使用伪逆求解线性方程是众多可能解法中的一种。具体地， $\mathbf{x} = \mathbf{A}^+ \mathbf{y}$  是方程所有可行解中欧几里得范数  $\|\mathbf{x}\|_2$  最小的一个。

当矩阵  $\mathbf{A}$  的行数多于列数时，可能没有解。在这种情况下，通过伪逆得到的  $\mathbf{x}$  是使得  $\mathbf{Ax}$  和  $\mathbf{y}$  的欧几里得距离  $\|\mathbf{Ax} - \mathbf{y}\|_2$  最小的解。

## 2.10 迹运算

迹运算返回的是矩阵对角元素的和：

$$\text{Tr}(\mathbf{A}) = \sum_i \mathbf{A}_{i,i}. \quad (2.49)$$

迹运算因为很多原因而受到关注。若不使用求和符号，有些矩阵运算很难描述，而通过矩阵乘法和迹运算符号，可以清楚地表示。例如，迹运算提供了另一种描述矩阵Frobenius 范数的方式：

$$\|\mathbf{A}\|_F = \sqrt{\text{Tr}(\mathbf{A} \mathbf{A}^\top)}. \quad (2.50)$$

用迹运算表示表达式，我们可以使用很多有用的等式来操纵表达式。例如，迹运算在转置运算下是不变的：

$$\text{Tr}(\mathbf{A}) = \text{Tr}(\mathbf{A}^\top). \quad (2.51)$$

多个矩阵乘积的迹，和将这些矩阵中最后一个挪到最前面之后乘积的迹是相同的。当然，我们需要考虑挪动之后矩阵乘积依然定义良好：

$$\text{Tr}(\mathbf{ABC}) = \text{Tr}(\mathbf{CAB}) = \text{Tr}(\mathbf{BCA}). \quad (2.52)$$

或者更一般地，

$$\mathrm{Tr}\left(\prod_{i=1}^n \mathbf{F}^{(i)}\right) = \mathrm{Tr}(\mathbf{F}^{(n)} \prod_{i=1}^{n-1} \mathbf{F}^{(i)}). \quad (2.53)$$

即使循环置换后矩阵乘积得到的矩阵形状变了，迹运算的结果依然不变。例如，假设矩阵  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , 矩阵  $\mathbf{B} \in \mathbb{R}^{n \times m}$ , 我们可以得到

$$\mathrm{Tr}(\mathbf{AB}) = \mathrm{Tr}(\mathbf{BA}) \quad (2.54)$$

即使  $\mathbf{AB} \in \mathbb{R}^{m \times m}$  和  $\mathbf{BA} \in \mathbb{R}^{n \times n}$ 。

另一个有用的事是标量在迹运算后仍然是它自己： $a = \mathrm{Tr}(a)$ 。

## 2.11 行列式

行列式，记作  $\det(\mathbf{A})$ ，是一个将方阵  $\mathbf{A}$  映射到实数的函数。行列式等于矩阵特征值的乘积。行列式的绝对值可以用来衡量矩阵相乘后空间扩大或者缩小了多少。如果行列式是 0，那么空间至少沿着某一维完全收缩了，使其失去了所有的体积。如果行列式是 1，那么矩阵相乘没有改变空间体积。

## 2.12 实例：主成分分析

主成分分析 (principal components analysis, PCA) 是一个简单的机器学习算法，可以通过基础的线性代数知识推导。

假设在  $\mathbb{R}^n$  空间中我们有  $m$  个点  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ ，我们希望对这些点进行有损压缩。有损压缩表示我们使用更少的内存，但损失一些精度去存储这些点。我们希望损失的精度尽可能少。

一种编码这些点的方式是用低维表示。对于每个点  $\mathbf{x}^{(i)} \in \mathbb{R}^n$ ，会有一个对应的编码向量  $\mathbf{c}^{(i)} \in \mathbb{R}^l$ 。如果  $l$  比  $n$  小，那么我们便使用了更少的内存来存储原来的数据。我们希望找到一个编码函数，根据输入返回编码， $f(\mathbf{x}) = \mathbf{c}$ ；我们也希望找到一个解码函数，给定编码重构输入， $\mathbf{x} \approx g(f(\mathbf{x}))$ 。

PCA由我们选择的解码函数而定。具体地，为了简化解码器，我们使用矩阵乘法将编码映射回  $\mathbb{R}^n$ ，即  $g(\mathbf{c}) = \mathbf{D}\mathbf{c}$ ，其中  $\mathbf{D} \in \mathbb{R}^{n \times l}$  是解码矩阵。

计算这个解码器的最优编码可能是一个困难的问题。为了使编码问题简单一些，PCA限制  $D$  的列向量为彼此正交的（注意，除非  $l = n$ ，否则严格上  $D$  不是一个正交矩阵）。

目前为止所描述的问题，可能会有多个解。因为如果我们按比例地缩小所有点对应的编码向量  $c_i$ ，那么我们只需按比例放大  $D_{:,i}$ ，即可保持结果不变。为了让问题有唯一解，我们限制  $D$  中所有列向量都有单位范数。

为了将这个基本想法放进我们能够实现的算法，首先我们需要明确如何根据每一个输入  $x$  得到一个最优编码  $c^*$ 。一种方法是最小化原始输入向量  $x$  和重构向量  $g(c^*)$  之间的距离。我们使用范数来衡量它们之间的距离。在PCA算法中，我们使用  $L^2$  范数：

$$c^* = \arg \min_c \|x - g(c)\|_2. \quad (2.55)$$

我们可以用平方  $L^2$  范数替代  $L^2$  范数，因为两者在相同的值  $c$  上取得最小值。这是因为  $L^2$  范数是非负的，并且平方运算在非负值上是单调递增的。

$$c^* = \arg \min_c \|x - g(c)\|_2^2. \quad (2.56)$$

该最小化函数可以简化成

$$(x - g(c))^\top (x - g(c)) \quad (2.57)$$

(式(2.31)中  $L^2$  范数的定义)

$$= x^\top x - x^\top g(c) - g(c)^\top x + g(c)^\top g(c) \quad (2.58)$$

(分配律)

$$= x^\top x - 2x^\top g(c) + g(c)^\top g(c) \quad (2.59)$$

(因为标量  $g(c)^\top x$  的转置等于自己)

因为第一项  $x^\top x$  不依赖于  $c$ ，所以我们可以忽略它，得到如下的优化目标：

$$c^* = \arg \min_c - 2x^\top g(c) + g(c)^\top g(c). \quad (2.60)$$

更进一步，我们代入  $g(c)$  的定义：

$$c^* = \arg \min_c - 2x^\top Dc + c^\top D^\top Dc \quad (2.61)$$

$$= \arg \min_c -2\mathbf{x}^\top \mathbf{D}\mathbf{c} + \mathbf{c}^\top \mathbf{I}_l \mathbf{c} \quad (2.62)$$

(矩阵  $\mathbf{D}$  有正交和单位范数约束)

$$= \arg \min_c -2\mathbf{x}^\top \mathbf{D}\mathbf{c} + \mathbf{c}^\top \mathbf{c} \quad (2.63)$$

我们可以通过线性代数微积分来求解这个最优化问题（如果你不清楚怎么做，请参考第4.3节）

$$\nabla_c (-2\mathbf{x}^\top \mathbf{D}\mathbf{c} + \mathbf{c}^\top \mathbf{c}) = 0 \quad (2.64)$$

$$-2\mathbf{D}^\top \mathbf{x} + 2\mathbf{c} = 0 \quad (2.65)$$

$$\mathbf{c} = \mathbf{D}^\top \mathbf{x}. \quad (2.66)$$

这使得算法很高效：最优编码  $\mathbf{x}$  只需要一个矩阵-向量乘法操作。我们使用该编码函数去编码向量：

$$f(\mathbf{x}) = \mathbf{D}^\top \mathbf{x}. \quad (2.67)$$

进一步使用矩阵乘法，我们也可以定义PCA重构操作：

$$r(\mathbf{x}) = g(f(\mathbf{x})) = \mathbf{D}\mathbf{D}^\top \mathbf{x}. \quad (2.68)$$

接下来，我们需要挑选编码矩阵  $\mathbf{D}$ 。要做到这一点，我们回顾最小化输入和重构之间  $L^2$  距离的这个想法。因为我们用相同的矩阵  $\mathbf{D}$  对所有点进行解码，我们不能再孤立地看待每个点。反之，我们必须最小化所有维数和所有点上的误差矩阵的Frobenius 范数：

$$\mathbf{D}^* = \arg \min_{\mathbf{D}} \sqrt{\sum_{i,j} \left( \mathbf{x}_j^{(i)} - r(\mathbf{x}^{(i)})_j \right)^2} \text{ 限制在 } \mathbf{D}^\top \mathbf{D} = \mathbf{I}_l. \quad (2.69)$$

为了推导用于寻求  $\mathbf{D}^*$  的算法，我们首先考虑  $l = 1$  的情况。在这种情况下， $\mathbf{D}$  是一个单一向量  $\mathbf{d}$ 。将式(2.70)代入式(2.71)，简化  $\mathbf{D}$  为  $\mathbf{d}$ ，问题化为

$$\mathbf{d}^* = \arg \min_{\mathbf{d}} \sum_i \left\| \mathbf{x}^{(i)} - \mathbf{d}\mathbf{d}^\top \mathbf{x}^{(i)} \right\|_2^2 \text{ 限制在 } \|\mathbf{d}\|_2 = 1. \quad (2.70)$$

上述公式是直接代入得到的，但不是文体表述最舒服的方式。在上述公式中，我们将标量  $\mathbf{d}^\top \mathbf{x}^{(i)}$  放在向量  $\mathbf{d}$  的右边。将该标量放在左边的写法更为传统。于是我们

## 2.12 实例：主成分分析

通常写作如下：

$$\mathbf{d}^* = \arg \min_{\mathbf{d}} \sum_i \left\| \mathbf{x}^{(i)} - \mathbf{d}^\top \mathbf{x}^{(i)} \mathbf{d} \right\|_2^2 \text{ 限制在 } \|\mathbf{d}\|_2 = 1, \quad (2.71)$$

或者，考虑到标量的转置和自身相等，我们也可以写作：

$$\mathbf{d}^* = \arg \min_{\mathbf{d}} \sum_i \left\| \mathbf{x}^{(i)} - \mathbf{x}^{(i)\top} \mathbf{d} \mathbf{d} \right\|_2^2 \text{ 限制在 } \|\mathbf{d}\|_2 = 1. \quad (2.72)$$

读者应该对这些重排写法慢慢熟悉起来。

此时，使用单一矩阵来重述问题，比将问题写成求和形式更有帮助。这有助于我们使用更紧凑的符号。让  $\mathbf{X} \in \mathbb{R}^{m \times n}$  是将描述点的向量堆叠在一起的矩阵，例如  $\mathbf{X}_{i,:} = \mathbf{x}^{(i)\top}$ 。原问题可以重新表述为：

$$\mathbf{d}^* = \arg \min_{\mathbf{d}} \left\| \mathbf{X} - \mathbf{X} \mathbf{d} \mathbf{d}^\top \right\|_F^2 \text{ 限制在 } \mathbf{d}^\top \mathbf{d} = 1. \quad (2.73)$$

暂时不考虑约束，我们可以将Frobenius 范数简化成下面的形式：

$$\arg \min_{\mathbf{d}} \left\| \mathbf{X} - \mathbf{X} \mathbf{d} \mathbf{d}^\top \right\|_F^2 \quad (2.74)$$

$$= \arg \min_{\mathbf{d}} \text{Tr} \left( (\mathbf{X} - \mathbf{X} \mathbf{d} \mathbf{d}^\top)^\top (\mathbf{X} - \mathbf{X} \mathbf{d} \mathbf{d}^\top) \right) \quad (2.75)$$

( 式(2.50) )

$$= \arg \min_{\mathbf{d}} \text{Tr} \left( \mathbf{X}^\top \mathbf{X} - \mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top - \mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X} + \mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top \right) \quad (2.76)$$

$$= \arg \min_{\mathbf{d}} \text{Tr}(\mathbf{X}^\top \mathbf{X}) - \text{Tr}(\mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) - \text{Tr}(\mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X}) + \text{Tr}(\mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) \quad (2.77)$$

$$= \arg \min_{\mathbf{d}} - \text{Tr}(\mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) - \text{Tr}(\mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X}) + \text{Tr}(\mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) \quad (2.78)$$

( 因为与  $\mathbf{d}$  无关的项不影响  $\arg \min$  )

$$= \arg \min_{\mathbf{d}} - 2\text{Tr}(\mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) + \text{Tr}(\mathbf{d} \mathbf{d}^\top \mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) \quad (2.79)$$

( 因为循环改变迹运算中相乘矩阵的顺序不影响结果，如式(2.55)所示 )

$$= \arg \min_{\mathbf{d}} - 2\text{Tr}(\mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top) + \text{Tr}(\mathbf{X}^\top \mathbf{X} \mathbf{d} \mathbf{d}^\top \mathbf{d} \mathbf{d}^\top) \quad (2.80)$$

( 再次使用上述性质 )

此时，我们再来考虑约束条件：

$$= \arg \min_d - 2\text{Tr}(\mathbf{X}^\top \mathbf{X} d d^\top) + \text{Tr}(\mathbf{X}^\top \mathbf{X} d d^\top) \text{ 限制在 } d^\top d = 1 \quad (2.81)$$

$$= \arg \min_d - 2\text{Tr}(\mathbf{X}^\top \mathbf{X} d d^\top) + \text{Tr}(\mathbf{X}^\top \mathbf{X} d d^\top) \text{ 限制在 } d^\top d = 1 \quad (2.82)$$

(因为约束条件)

$$= \arg \min_d - \text{Tr}(\mathbf{X}^\top \mathbf{X} d d^\top) \text{ 限制在 } d^\top d = 1 \quad (2.83)$$

$$= \arg \max_d \text{Tr}(\mathbf{X}^\top \mathbf{X} d d^\top) \text{ 限制在 } d^\top d = 1 \quad (2.84)$$

$$= \arg \max_d \text{Tr}(d^\top \mathbf{X}^\top \mathbf{X} d) \text{ 限制在 } d^\top d = 1 \quad (2.85)$$

这个优化问题可以通过特征分解来求解。具体地，最优的  $\mathbf{d}$  是  $\mathbf{X}^\top \mathbf{X}$  最大特征值对应的特征向量。

以上推导特定于  $l = 1$  的情况，仅得到了第一个主成分。更一般地，当我们希望得到主成分的基时，矩阵  $\mathbf{D}$  由几个最大的特征值对应的  $l$  个特征向量组成。这个结论可以通过归纳法证明，我们建议将此证明作为练习。

线性代数是学习深度学习所必须掌握的基础数学学科之一。另一门在机器学习中无处不在的重要数学学科是概率学，我们将在下章探讨。

# 第三章 概率与信息论

本章我们讨论概率论和信息论。

概率论是用于表示不确定性陈述(statement) 的数学框架。它不仅提供了量化不确定性的方法，也提供了用于导出新的不确定性陈述的公理。在人工智能领域，我们主要以两种方式来使用概率论。首先，概率法则告诉我们AI系统应该如何推理，所以我们设计一些算法来计算或者近似由概率论导出的表达式。其次，我们可以用概率和统计从理论上分析我们提出的AI系统的行为。

概率论是众多科学和工程学科的基本工具。我们提供这一章是为了保证那些背景是软件工程而较少接触概率论的读者也可以理解本书的内容。

概率论使我们能够作出不确定的陈述以及在不确定性存在的情况下推理，而信息论使我们能够量化概率分布中的不确定性总量。

如果你已经对概率论和信息论很熟悉了，那么你可能会希望跳过除了3.14节以外的整章内容，而在这一节中我们会介绍用来描述机器学习中结构化概率模型的图。如果你对这些主题完全没有任何的先验知识，本章对于完成深度学习的研究项目来说已经足够，但我们还是建议你能够参考一些额外的资料，例如Jaynes (2003)。

## 3.1 为什么要用概率？

计算机科学的许多分支处理的大部分都是完全确定的实体。程序员通常可以安全地假定 CPU 将完美地执行每个机器指令。硬件错误确实会发生，但它们足够罕见，以至于大部分软件应用并不需要被设计为考虑这些因素的影响。鉴于很多计算机科学家和软件工程师在一个相对干净和确定的环境中工作，机器学习对于概率论的大量使用不得不令人吃惊。

这是因为机器学习必须始终处理不确定量，有时也可能需要处理随机(非确定性)量。不确定性和随机性可能来自多个方面。研究人员至少从20世纪80年代开始就对使用概率论来量化不确定性提出了令人信服的论据。这里提出的许多论点都是根据Pearl (1988) 总结或启发得到的。

几乎所有的活动都需要能够在不确定性存在时进行推理。事实上，除了那些被定义为真的数学陈述，我们很难认定某个命题是千真万确的或者确保某件事一定会发生。

不确定性有三种可能的来源：

1. 被建模系统内在的随机性。例如，大多数量子力学的解释，都将亚原子(subatomic) 粒子的动力学描述为概率性的。我们还可以假定有随机动力学，并为此建立理论方案，例如一个假想的纸牌游戏，我们假设纸牌真正混洗成随机顺序。
2. 不完全观测。即使是确定的系统，当我们不能观测到所有驱动系统行为的变量时，该系统也会呈现随机性。例如，在Monty Hall提出的问题中，一个游戏节目的参赛者被要求在三个门之间选择并且赢得放置在选中门后的奖金。两扇门通向山羊，第三扇门通向一辆汽车。选手选择所导致的结果是确定的，但是站在选手的角度，结果是不确定的。
3. 不完全建模。当我们使用一些必须舍弃某些观测信息的模型时，舍弃的信息会导致模型的预测出现不确定性。例如，假设我们建立了一个机器人，它可以准确地观察周围每一个对象的位置。如果预测这些对象将来的位置时机器人采用的是离散化的空间，那么离散化使得机器人立即变得不确定对象的精确位置：每个对象都可能处于它被观察到占据的离散单元的任何位置。

在很多情况下，使用一些简单而不确定的原则要比复杂而确定的原则更为实用，即使真实的规则是确定的并且我们的模型系统对适应复杂规则具有很好的逼真度(fidelity)。例如，简单的原则“多数鸟儿都会飞”可以很容易应用并且使用广泛，而正式的规则，“除了那些非常小的还没学会飞翔的幼鸟，生病或是受伤失去了飞翔能力的鸟，不会飞的鸟类包括食火鸟(cassowary)、鸵鸟(ostrich)、几维(kiwi，一种新西兰产的无翼鸟)……等等，鸟儿会飞”，很难应用、维持和沟通，毕竟这方面的努力还是很脆弱，容易失败。

尽管我们明确需要一种表示和推理不确定性的方法，但是概率论能够提供所有我们想要的人工智能领域的工具并不是那么显然。概率论最初的发展是为了分析事件发生的频率。可以很容易地看出概率论，对于像在扑克牌游戏中抽出一手特定的牌这种事件的研究中，是如何使用的。这类事件往往是重复的。当我们说一个结果发生的概率为  $p$ ，这意味着如果我们反复实验（例如，抽取一手牌）无限次，有  $p$  的比例会导致这样的结果。这种推理似乎并不立即适用于那些不重复的命题。如果一个医生诊断了病人，并说该病人患流感的几率为 40%，这意味着非常不同的事情——我们既不能让病人有无穷多的副本，也没有任何理由去相信病人的不同副本在具有不同的潜在情况下表现出相同的症状。在医生诊断病人的情况下，我们用概率来表示一种信任度 (degree of belief)，其中 1 表示非常肯定病人患有流感而 0 表示非常肯定病人没有流感。前面一种概率，直接与事件发生的频率相联系，被称为频率概率 (frequentist probability)；而后者，涉及到确定性水平，被称为贝叶斯概率 (Bayesian probability)。

如果我们要列出一些，我们期望关于不确定性的常识推理具有的性质，那么满足这些属性的唯一一点就是将贝叶斯概率和频率概率视为等同的。例如，如果我们要在扑克牌游戏中根据玩家手上的牌计算她能够获胜的概率，我们和医生情境使用完全相同的公式，就是我们依据病人的某些症状计算她是否患病的概率。有关为什么一些小的常识假设能够导出相同的公理的细节必须深入了解这两种概率，参见 Ramsey (1926)。

概率可以被看作是用于处理不确定性的逻辑扩展。逻辑提供了一套形式化的规则在给定某些命题是真或假的假设下，判断另外一些命题是真的还是假的。概率论提供了一套形式化的规则在给定一些命题的似然 (likelihood) 后，计算其他命题为真的似然。

## 3.2 随机变量

随机变量 (random variable) 是可以随机地取不同值的变量。我们通常用打印机体的小写字母来表示随机变量本身，而用脚本字体中的小写字母来表示随机变量能够取到的值。例如， $x_1$  和  $x_2$  都是随机变量  $x$  可能的取值。对于向量值变量，我们会将随机变量写成  $\mathbf{x}$ ，它的一个值为  $\mathbf{x}$ 。就其本身而言，一个随机变量只是对可能的状态的描述；它必须伴随着一个概率分布来指定每个状态的可能性。

随机变量可以是离散的或者连续的。离散型随机变量拥有有限或者可数无限多的状态。注意这些状态不一定非要是整数；它们也可能只是一些被命名的状态并且没有数值。连续型随机变量伴随着实数值。

## 3.3 概率分布

概率分布 (probability distribution) 用来描述随机变量或一簇随机变量在每一个可能取到的状态的可能性大小。我们描述概率分布的方式取决于随机变量是离散的还是连续的。

### 3.3.1 离散型变量和概率分布律函数

离散型变量的概率分布可以用概率分布律函数 (probability mass function, PMF)<sup>1</sup> 来描述。我们通常用大写字母  $P$  来表示概率分布律函数。通常每一个随机变量都会有一个不同的概率分布律函数，并且读者必须根据随机变量来推断所使用的PMF，而不是根据函数的名称来推断；例如， $P(x)$  通常和  $P(y)$  不一样。

概率分布律函数将随机变量能够取得的每个状态映射到随机变量取得该状态的概率。 $x = x$  的概率用  $P(x)$  来表示，概率为 1 表示  $x = x$  是确定的，概率为 0 表示  $x = x$  是不可能发生的。有时为了使得PMF的使用不相互混淆，我们会明确写出随机变量的名称： $P(x = x)$ 。有时我们会先定义一个随机变量，然后用  $\sim$  符号来说明它遵循的分布： $x \sim P(x)$ 。

概率分布律函数可以同时作用于多个随机变量。这种多个变量的概率分布被称为联合概率分布 (joint probability distribution)。 $P(x = x, y = y)$  表示  $x = x$  和  $y = y$  同时发生的概率。我们也可以简写为  $P(x, y)$ 。

一个函数  $P$  如果想要成为随机变量  $x$  的PMF，必须满足下面这几个条件：

- $P$  的定义域必须是  $x$  所有可能状态的集合。
- $\forall x \in x, 0 \leq P(x) \leq 1$ . 不可能发生的事件概率为 0，并且没有比这概率更低的状态了。类似的，能够确保一定发生的事件概率为 1，而且没有比这概率更高的状态了。

<sup>1</sup>译者注：国内有些教材也将它翻译成概率分布律。

- $\sum_{x \in \mathcal{X}} P(x) = 1$ . 我们把这条性质称之为归一性 (normalized)。如果没有这条性质，当我们计算很多事件其中之一发生的概率时可能会得到大于 1 的概率。

例如，考虑一个离散型随机变量  $x$  有  $k$  个不同的状态。我们可以假设  $x$  是均匀分布 (uniform distribution) 的——也就是将它的每个状态视为等可能的——通过将它的PMF设为

$$P(x = x_i) = \frac{1}{k} \quad (3.1)$$

对于所有的  $i$  都成立。可以看出这满足上述成为概率分布律函数的条件。 $\frac{1}{k}$  是正的，因为  $k$  是一个正整数。我们也可以看出

$$\sum_i P(x = x_i) = \sum_i \frac{1}{k} = \frac{k}{k} = 1, \quad (3.2)$$

所以分布也满足归一化条件。

### 3.3.2 连续型变量和概率密度函数

当我们研究的对象是连续型随机变量时，我们用概率密度函数 (probability density function, PDF) 而不是概率分布律函数来描述它的概率分布。一个函数  $p$  如果想要成为概率密度函数，必须满足下面这几个条件：

- $p$  的定义域必须是  $x$  所有可能状态的集合。
- $\forall x \in \mathcal{X}, p(x) \geq 0$ . 注意，我们并不要求  $p(x) \leq 1$ 。
- $\int p(x) dx = 1$ .

概率密度函数  $p(x)$  并没有直接对特定的状态给出概率，相对的，它给出了落在面积为  $\delta x$  的无限小的区域内的概率为  $p(x)\delta x$ 。

我们可以对概率密度函数求积分来获得点集的真实分布律。特别地， $x$  落在集合  $\mathbb{S}$  中的概率可以通过  $p(x)$  对这个集合求积分来得到。在单变量的例子中， $x$  落在区间  $[a, b]$  的概率是  $\int_{[a,b]} p(x) dx$ 。

为了给出一个连续型随机变量的PDF的例子，考虑实数区间上的均匀分布。我们可以通过函数  $u(x; a, b)$  来实现，其中  $a$  和  $b$  是区间的端点满足  $b > a$ 。符号 “;” 表示“以什么为参数”；我们把  $x$  作为函数的自变量， $a$  和  $b$  作为定义函数的参数。

为了确保区间外没有概率，我们对所有的  $x \notin [a, b]$  令  $u(x; a, b) = 0$ 。在  $[a, b]$  内，有  $u(x; a, b) = \frac{1}{b-a}$ 。可以看出任何一点都非负。另外，它的积分为 1。我们通常用  $x \sim U(a, b)$  表示  $x$  在  $[a, b]$  上是均匀分布的。

## 3.4 边缘概率

有时候，我们知道了一组变量的联合概率分布，想要了解其中一个子集的概率分布。这种定义在子集上的概率分布被称为边缘概率分布 (marginal probability distribution)。

例如，假设有离散型随机变量  $x$  和  $y$ ，并且我们知道  $P(x, y)$ 。我们可以依据下面的求和法则 (sum rule) 来计算  $P(x)$ :

$$\forall x \in X, P(x = x) = \sum_y P(x = x, y = y). \quad (3.3)$$

“边缘概率”的名称来源于手算边缘概率的计算过程。当  $P(x, y)$  的每个值被写在由每行表示不同的  $x$  值，每列表示不同的  $y$  值形成的网格中时，对网格中的每行求和是很自然的事情，然后将求和的结果  $P(x)$  写在每行右边的纸的边缘处。

对于连续型变量，我们需要用积分替代求和：

$$p(x) = \int p(x, y) dy. \quad (3.4)$$

## 3.5 条件概率

在很多情况下，我们感兴趣的是某个事件，在给定其他事件发生时，出现的概率。这种概率叫做条件概率。我们将给定  $x = x$  时  $y = y$  发生的条件概率记为  $P(y = y | x = x)$ 。这个条件概率可以通过下面的公式计算：

$$P(y = y | x = x) = \frac{P(y = y, x = x)}{P(x = x)} \quad (3.5)$$

条件概率只在  $P(x = x) > 0$  时有定义。我们不能计算给定在永远不会发生的事件上的条件概率。

这里需要注意的是，不要把条件概率和计算当采用某个动作后会发生什么相混淆。假定某个人说德语，那么他可能是德国人的条件概率是非常高的，但是如果

随机选择的一个人会说德语，他的国籍是不变的。计算一个行动的后果被称为干预查询 (intervention query)。干预查询属于因果模型 (causal modeling) 的范畴，我们不在本书中讨论。

## 3.6 条件概率的链式法则

任何多维随机变量的联合概率分布，都可以分解成只有一个变量的条件概率相乘的形式：

$$P(x^{(1)}, \dots, x^{(n)}) = P(x^{(1)}) \prod_{i=2}^n P(x^{(i)} | x^{(1)}, \dots, x^{(i-1)}) \quad (3.6)$$

这个规则被称为概率的链式法则 (chain rule) 或者乘法法则 (product rule)。它可以直接从公式3.5条件概率的定义中得到。例如，使用两次定义可以得到

$$\begin{aligned} P(a, b, c) &= P(a | b, c)P(b, c) \\ P(b, c) &= P(b | c)P(c) \\ P(a, b, c) &= P(a | b, c)P(b | c)P(c). \end{aligned}$$

## 3.7 独立性和条件独立性

两个随机变量  $x$  和  $y$ ，如果它们的概率分布可以表示成两个因子的乘积形式，并且一个因子只包含  $x$  另一个因子只包含  $y$ ，我们就称这两个随机变量是相互独立的 (independent)：

$$\forall x \in \mathcal{X}, y \in \mathcal{Y}, p(x = x, y = y) = p(x = x)p(y = y). \quad (3.7)$$

两个随机变量  $x$  和  $y$  在给定随机变量  $z$  是条件独立的 (conditionally independent)，如果关于  $x$  和  $y$  的条件概率分布对于  $z$  的每一个值都可以写成乘积的形式：

$$\forall x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z}, p(x = x, y = y | z = z) = p(x = x | z = z)p(y = y | z = z). \quad (3.8)$$

我们可以采用一种简化形式来表示独立性和条件独立性： $x \perp y$  表示  $x$  和  $y$  相互独立， $x \perp y | z$  表示  $x$  和  $y$  在给定  $z$  时条件独立。

## 3.8 期望，方差和协方差

函数  $f(x)$  关于某分布  $P(x)$  的期望 (expectation) 或者期望值 (expected value) 是指，当  $x$  由  $P$  产生时， $f$  作用于  $x$  的平均值。对于离散型随机变量，可以通过求和得到：

$$\mathbb{E}_{x \sim P}[f(x)] = \sum_x P(x)f(x), \quad (3.9)$$

对于连续型随机变量可以通过求积分得到：

$$\mathbb{E}_{x \sim p}[f(x)] = \int p(x)f(x)dx. \quad (3.10)$$

当概率分布在上下文中指明时，我们可以只写出期望作用的随机变量的名字来进行简化，例如  $\mathbb{E}_x[f(x)]$ 。如果期望作用的随机变量也很明确，我们可以完全不写脚标，就像  $\mathbb{E}[f(x)]$ 。默认地，我们假设  $\mathbb{E}[\cdot]$  表示对方括号内的所有随机变量的值求平均。类似的，当没有歧义时，我们还可以省略方括号。

期望是线性的，例如，

$$\mathbb{E}_x[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_x[f(x)] + \beta \mathbb{E}_x[g(x)], \quad (3.11)$$

当  $\alpha$  和  $\beta$  不依赖于  $x$  时。

**方差** (variance) 衡量的是当我们对  $x$  依据它的概率分布进行采样时，随机变量  $x$  的函数值会呈现多大的差异：

$$\text{Var}(f(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]. \quad (3.12)$$

当方差很小时， $f(x)$  的值形成的簇比较接近它们的期望值。方差的平方根被称为**标准差** (standard deviation)。

**协方差** (covariance) 在某种意义上给出了两个变量线性相关性的强度以及这些变量的尺度：

$$\text{Cov}(f(x), g(y)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])(g(y) - \mathbb{E}[g(y)])]. \quad (3.13)$$

协方差的绝对值如果很大则意味着变量值变化很大并且它们同时距离各自的均值很远。如果协方差是正的，那么两个变量都倾向于同时取得相对较大的值。如果协方差是负的，那么其中一个变量倾向于取得相对较大的值的同时，另一个变量倾向于

取得相对较小的值，反之亦然。其他的衡量指标如相关系数 (correlation) 将每个变量的贡献归一化，为了只衡量变量的相关性，而不受变量大小的分别影响。

协方差和相关性是有联系的，但实际上不同的概念。它们是有联系的，因为两个变量如果相互独立那么它们的协方差为零，如果两个变量的协方差不为零那么它们一定是相关的。然而，独立性又是和协方差完全不同的性质。两个变量如果协方差为零，它们之间一定没有线性关系。独立性是大于零协方差的更强的要求，因为独立性还排除了非线性的关系。两个变量相互依赖但是具有零协方差是可能的。例如，假设我们首先从区间  $[-1, 1]$  上的均匀分布中采样出一个实数  $x$ 。然后我们对一个随机变量  $s$  进行采样。 $s$  以  $\frac{1}{2}$  的概率值为 1，否则为 -1。我们可以通过令  $y = sx$  来生成一个随机变量  $y$ 。显然， $x$  和  $y$  不是相互独立的，因为  $x$  完全决定了  $y$  的尺度。然而， $\text{Cov}(x, y) = 0$ 。

随机向量  $\mathbf{x} \in \mathbb{R}^n$  的协方差矩阵 (covariance matrix) 是一个  $n \times n$  的矩阵，并且满足

$$\text{Cov}(\mathbf{x})_{i,j} = \text{Cov}(\mathbf{x}_i, \mathbf{x}_j). \quad (3.14)$$

协方差矩阵的对角元是方差：

$$\text{Cov}(\mathbf{x}_i, \mathbf{x}_i) = \text{Var}(\mathbf{x}_i). \quad (3.15)$$

## 3.9 常用概率分布

许多简单的概率分布在机器学习的众多领域中都是有用的。

### 3.9.1 Bernoulli 分布

**Bernoulli 分布** (Bernoulli distribution) 是单个二值型随机变量的分布。它由单个参数  $\phi \in [0, 1]$  控制， $\phi$  给出了随机变量等于 1 的概率。它具有如下的一些性质：

$$P(\mathbf{x} = 1) = \phi \quad (3.16)$$

$$P(\mathbf{x} = 0) = 1 - \phi \quad (3.17)$$

$$P(\mathbf{x} = x) = \phi^x(1 - \phi)^{1-x} \quad (3.18)$$

$$\mathbb{E}_{\mathbf{x}}[\mathbf{x}] = \phi \quad (3.19)$$

$$\text{Var}_{\mathbf{x}}(\mathbf{x}) = \phi(1 - \phi) \quad (3.20)$$

### 3.9.2 Multinoulli 分布

Multinoulli 分布 (multinoulli distribution) 或者分类分布 (categorical distribution) 是指在具有  $k$  个不同状态的单个离散型随机变量上的分布,  $k$  是有限的。<sup>2</sup> Multinoulli 分布的参数是向量  $\mathbf{p} \in [0, 1]^{k-1}$ , 每一个分量  $p_i$  表示第  $i$  个状态的概率。最后的第  $k$  个状态的概率通过  $1 - \mathbf{1}^\top \mathbf{p}$  给出。注意到我们必须限制  $\mathbf{1}^\top \mathbf{p} \leq 1$ 。Multinoulli 分布经常用来表示对象分类的分布, 所以我们很少假设状态 1 具有数值 1 之类的。因此, 我们通常不需要去计算 Multinoulli 分布的随机变量的期望和方差。

Bernoulli 分布和 Multinoulli 分布足够用来描述在它们领域内的任意分布。它们能够描述这些分布, 不是因为它们特别强大, 而是因为它们的领域很简单; 它们可以对那些, 能够将所有的状态进行标号的离散型随机变量, 进行建模。当处理的是连续型随机变量时, 会有不可数无限多的状态, 所以任何通过少量参数描述的概率分布都必须在分布上采用严格的极限。

### 3.9.3 高斯分布

对于实数上的分布最常用的就是正态分布 (normal distribution), 也称为高斯分布 (Gaussian distribution):

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right). \quad (3.21)$$

图3.1画出了正态分布的概率密度函数。

正态分布被两个参数控制,  $\mu \in \mathbb{R}$  和  $\sigma \in (0, \infty)$ 。参数  $\mu$  给出了中心峰值的坐标, 这也是分布的均值:  $E[x] = \mu$ 。分布的标准误差用  $\sigma$  表示, 方差用  $\sigma^2$  表示。

当我们需要对概率密度函数求值时, 我们需要对  $\sigma$  平方并且取倒数。当我们需要经常对不同参数下的概率密度函数求值时, 一种更高效的使用参数描述分布的方式是使用参数  $\beta \in (0, \infty)$ , 来控制分布的精度 (precision) 或者方差的倒数:

$$\mathcal{N}(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x - \mu)^2\right). \quad (3.22)$$

<sup>2</sup>“multinoulli”这个术语是最近被 Gustavo Lacerdo 发明、被 Murphy (2012) 推广的。Multinoulli 分布是多项分布 (multinomial distribution) 的一个特例。多项分布是  $\{0, \dots, n\}^k$  中的向量的分布, 用于表示当对 Multinoulli 分布采样  $n$  次时  $k$  个类中的每一个被访问的次数。很多文章使用“多项分布”而实际上说的是 Multinoulli 分布, 但是他们并没有说是对  $n = 1$  的情况, 这点需要注意。

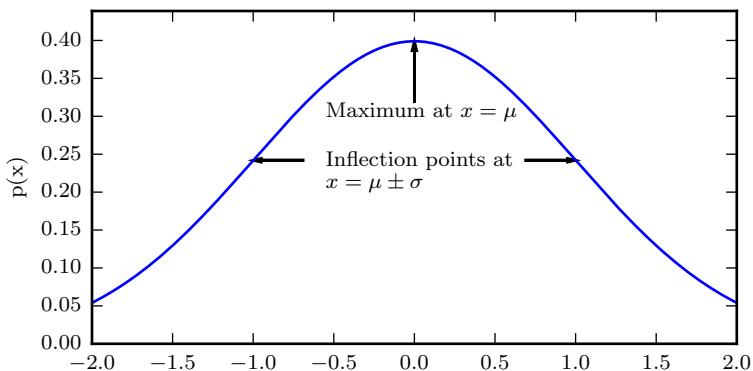


图 3.1: 正态分布。正态分布  $\mathcal{N}(x; \mu, \sigma^2)$  呈现经典的“钟形曲线”的形状，其中中心峰的  $x$  坐标由  $\mu$  给出，峰的宽度受  $\sigma$  控制。在这个示例中，我们展示的是标准正态分布 (standard normal distribution)，其中  $\mu = 0, \sigma = 1$ 。

采用正态分布在很多应用中都是一个明智的选择。当我们缺乏对于某个实数上分布的先验知识而不知道该选择怎样的形式时，正态分布是默认的比较好的选择，这里有两个原因。

第一，我们想要建模的很多分布真实情况是比较接近正态分布的。**中心极限定理** (central limit theorem) 说明很多独立随机变量的和近似服从正态分布。这意味着在实际中，很多复杂系统都可以被成功建模成正态分布的噪声，即使系统可以被分解成具有更多结构化行为的各个部分。

第二，在具有相同方差的所有可能的概率分布中，正态分布在实数上具有最大的不确定性。我们因此可以认为正态分布是对模型加入的先验知识量最少的分布。充分利用和证明这个想法需要更多的数学工具，我们推迟到 19.4.2 节进行讲解。

正态分布可以推广到  $\mathbb{R}^n$  空间，这种情况下被称为**多维正态分布** (multivariate normal distribution)。它的参数是一个正定对称矩阵  $\Sigma$ :

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sqrt{\frac{1}{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right). \quad (3.23)$$

参数  $\boldsymbol{\mu}$  仍然表示分布的均值，只不过现在是向量值的。参数  $\boldsymbol{\Sigma}$  给出了分布的协方差矩阵。和单变量的情况类似，当我们希望对很多不同参数下的概率密度函数多次求值时，协方差矩阵并不是一个很高效的用参数描述分布的方法，因为对概率密

度函数求值时需要对  $\Sigma$  求逆。我们可以用一个精度矩阵 (precision matrix)  $\beta$  进行替代：

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\beta}^{-1}) = \sqrt{\frac{\det(\boldsymbol{\beta})}{(2\pi)^n}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\beta}(\mathbf{x} - \boldsymbol{\mu})\right). \quad (3.24)$$

我们常常把协方差矩阵固定成一个对角阵。一个更简单的版本是各向同性 (isotropic) 高斯分布，它的协方差矩阵是一个标量乘以单位阵。

### 3.9.4 指数分布和 Laplace 分布

在深度学习中，我们经常会需要一个在  $x = 0$  点处取得边界点 (sharp point) 的分布。为了实现这一目的，我们可以使用指数分布 (exponential distribution)：

$$p(x; \lambda) = \lambda \mathbf{1}_{x \geq 0} \exp(-\lambda x). \quad (3.25)$$

指数分布使用指示函数(indicator function)  $\mathbf{1}_{x \geq 0}$  来使得当  $x$  取负值时的概率为零。

一个非常相关的概率分布是Laplace 分布 (Laplace distribution)，它允许我们在任意一点  $\mu$  处设置概率分布的峰值

$$\text{Laplace}(x; \mu, \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x - \mu|}{\gamma}\right). \quad (3.26)$$

### 3.9.5 Dirac 分布和经验分布

在一些情况下，我们想要所有的概率都集中在一个点上。这可以通过Dirac delta 函数 (Dirac delta function)  $\delta(x)$  定义概率密度函数来实现：

$$p(x) = \delta(x - \mu). \quad (3.27)$$

Dirac delta 函数被定义成除了 0 以外的其他点的值都为 0，但是积分为 1。Dirac delta 函数不像普通函数一样对  $x$  的每一个值都有一个实数值的输出，它是一种不同类型的数学对象，被称为广义函数 (generalized function)，广义函数是依据积分性质定义的数学对象。我们可以把Dirac delta 函数想成一系列函数的极限点，这一系列函数把除  $\mu$  以外的所有点的概率密度越变越小。

通过把  $p(x)$  定义成  $\delta$  函数左移  $-\mu$  个单位，我们得到了一个在  $x = \mu$  处具有无限窄也无限高的峰值的概率密度函数。

Dirac 分布经常作为经验分布 (empirical distribution) 的一个组成部分出现：

$$\hat{p}(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^m \delta(\mathbf{x} - \mathbf{x}^{(i)}) \quad (3.28)$$

经验分布将概率密度  $\frac{1}{m}$  赋给  $m$  个点  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$  中的每一个，这些点是给定的数据集或者采样的集合。Dirac delta 函数只对定义连续型随机变量的经验分布是必要的。对于离散型随机变量，情况更加简单：经验分布可以被定义成一个 Multinoulli 分布，对于每一个可能的输入，其概率可以简单地设为在训练集上那个输入值的经验频率 (empirical frequency)。

当我们在训练集上训练模型时，我们可以认为从这个训练集上得到的经验分布指明了我们采样来源的分布。关于经验分布另外一种重要的观点是，它是训练数据的似然最大的那个概率密度函数 (见 5.5 节)。

### 3.9.6 分布的混合

通过组合一些简单的概率分布来定义新的概率分布也是很常见的。一种通用的组合方法是构造混合分布 (mixture distribution)。混合分布由一些组件 (component) 分布构成。每次实验，样本是由哪个组件分布产生的取决于从一个 Multinoulli 分布中采样的结果：

$$P(\mathbf{x}) = \sum_i P(\mathbf{c} = i) P(\mathbf{x} | \mathbf{c} = i), \quad (3.29)$$

这里  $P(\mathbf{c})$  是对各组件的一个 Multinoulli 分布。

我们已经看过一个混合分布的例子了：实值变量的经验分布对于每一个训练实例来说，就是以 Dirac 分布为组件的混合分布。

混合模型是组合简单概率分布来生成更丰富的分布的一种简单策略。在第十六章中，我们探讨从简单概率分布构建复杂模型的更详细的技术。

混合模型使我们能够一瞥以后会用到的一个非常重要的概念——潜变量 (latent variable)。潜变量是我们不能直接观测到的随机变量。混合模型的组件变量  $\mathbf{c}$  就是其中一个例子。潜变量在联合分布中可能和  $\mathbf{x}$  有关，在这种情况下， $P(\mathbf{x}, \mathbf{c}) = P(\mathbf{x} | \mathbf{c})P(\mathbf{c})$ 。潜变量的分布  $P(\mathbf{c})$  以及关联潜变量和观测变量的条件分布  $P(\mathbf{x} | \mathbf{c})$ ，共同决定了分布  $P(\mathbf{x})$  的形状，尽管描述  $P(\mathbf{x})$  时可能并不需要潜变量。潜变量会在 16.5 一节中深入讨论。

一个非常强大且常见的混合模型是高斯混合模型 (Gaussian Mixture Model)，它的组件  $p(x | c = i)$  是高斯分布。每个组件都有各自的参数，均值  $\mu^{(i)}$  和协方差矩阵  $\Sigma^{(i)}$ 。有一些混合可以有更多的限制。例如，协方差矩阵可以通过  $\Sigma^{(i)} = \Sigma, \forall i$  的形式在组件之间共享参数。和单个高斯分布一样，高斯混合模型有时会限制每个组件的协方差矩阵是对角的或者各向同性的 (标量乘以单位矩阵)。

除了均值和协方差以外，高斯混合模型的参数指明了给每个组件  $i$  的先验概率 (prior probability)  $\alpha_i = P(c = i)$ 。“先验”一词表明了在观测到  $x$  之前传递给模型关于  $c$  的信念。作为对比， $P(c | x)$  是后验概率 (posterior probability)，因为它是是在观测到  $x$  之后进行计算的。高斯混合模型是概率密度的通用逼近器 (universal approximator)，在这种意义上，任何平滑的概率密度都可以用具有足够多组件的高斯混合模型以任意精度来逼近。

图3.2演示了某个高斯混合模型生成的样例。

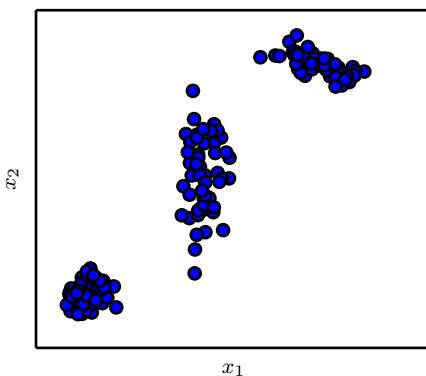


图 3.2: 来自高斯混合模型的样本。在这个示例中，有三个组件。从左到右，第一个组件具有各向同性的协方差矩阵，这意味着它在每个方向上具有相同的方差。第二个组件具有对角的协方差矩阵，这意味着它可以沿着每个轴的对齐方向单独控制方差。该示例中，沿着  $x_2$  轴的方差要比沿着  $x_1$  轴的方差大。第三个组件具有满秩的协方差矩阵，使它能够沿着任意基的方向单独地控制方差。

### 3.10 常用函数的一些性质

某些函数在处理概率分布时经常会出现，尤其是深度学习的模型中用到的概率分布。

其中一个函数是logistic sigmoid函数：

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \quad (3.30)$$

logistic sigmoid函数通常用来产生Bernoulli 分布中的参数  $\phi$ , 因为它的范围是  $(0, 1)$ , 处在  $\phi$  的有效取值范围内。图3.3给出了 sigmoid 函数的图示。sigmoid 函数在变量取绝对值非常大的正值或负值时会出现饱和 (saturate) 现象, 意味着函数会变得很平, 并且对输入的微小改变会变得不敏感。

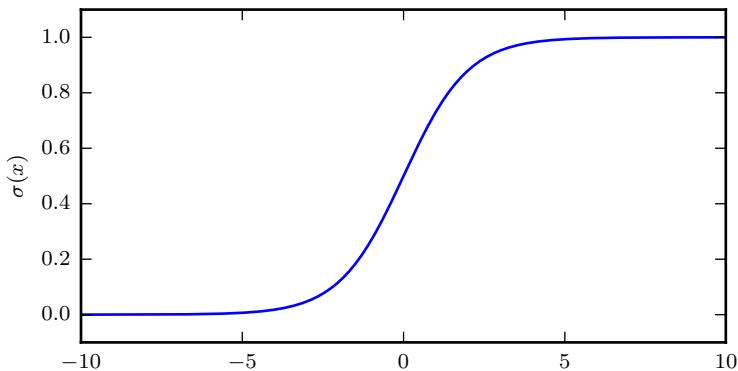


图 3.3: logistic sigmoid函数。

另外一个经常遇到的函数是softplus 函数 (softplus function)(Dugas *et al.*, 2001a):

$$\zeta(x) = \log(1 + \exp(x)). \quad (3.31)$$

softplus 函数可以用来产生正态分布的  $\beta$  和  $\sigma$  参数, 因为它的范围是  $(0, \infty)$ 。当处理包含 sigmoid 函数的表达式时它也经常出现。softplus 函数名来源于它是另外一个函数的平滑形式, 这个函数是

$$x^+ = \max(0, x). \quad (3.32)$$

图3.4给出了softplus 函数的图示。

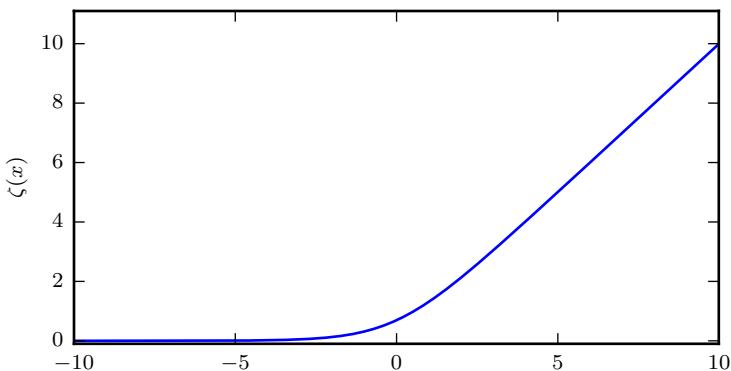


图 3.4: softplus 函数。

下面一些性质非常有用，你可能会希望记下来：

$$\sigma(x) = \frac{\exp(x)}{\exp(x) + \exp(0)} \quad (3.33)$$

$$\frac{d}{dx} \sigma(x) = \sigma(x)(1 - \sigma(x)) \quad (3.34)$$

$$1 - \sigma(x) = \sigma(-x) \quad (3.35)$$

$$\log \sigma(x) = -\zeta(-x) \quad (3.36)$$

$$\frac{d}{dx} \zeta(x) = \sigma(x) \quad (3.37)$$

$$\forall x \in (0, 1), \sigma^{-1}(x) = \log \left( \frac{x}{1-x} \right) \quad (3.38)$$

$$\forall x > 0, \zeta^{-1}(x) = \log(\exp(x) - 1) \quad (3.39)$$

$$\zeta(x) = \int_{-\infty}^x \sigma(y) dy \quad (3.40)$$

$$\zeta(x) - \zeta(-x) = x \quad (3.41)$$

函数  $\sigma^{-1}(x)$  在统计学中被称为分对数 (logit)，但这个函数在机器学习中很少用到。

公式3.41为函数名“softplus”提供了其他的正当理由。softplus 函数被设计成正部函数 (positive part function) 的平滑版本，这个正部函数是指  $x^+ = \max\{0, x\}$ 。与正部函数相对的是负部函数 (negative part function)  $x^- = \max\{0, -x\}$ 。为了获得类似负部函数的一个平滑函数，我们可以使用  $\zeta(-x)$ 。就像  $x$  可以用它的正部和负部通过等式  $x^+ - x^- = x$  恢复一样，我们也可以用同样的方式对  $\zeta(x)$  和  $\zeta(-x)$  进

行操作，就像公式3.41中那样。

## 3.11 贝叶斯规则

我们经常会需要在已知  $P(y | x)$  时计算  $P(x | y)$ 。幸运的是，如果还知道  $P(x)$ ，我们可以用贝叶斯规则 (Bayes' rule) 来实现这一目的：

$$P(x | y) = \frac{P(x)P(y | x)}{P(y)}. \quad (3.42)$$

注意到  $P(y)$  出现在上面的公式中，它通常使用  $P(y) = \sum_x P(y | x)P(x)$  来计算，所以我们并不需要事先知道  $P(y)$  的信息。

贝叶斯规则可以从条件概率的定义直接推导得出，但我们最好记住这个公式的名字，因为很多文献通过名字来引用这个公式。这个公式是以 Reverend Thomas Bayes 来命名的，他是第一个发现这个公式的特例的人。这里介绍的一般形式由 Pierre-Simon Laplace 独立发现。

## 3.12 连续型变量的技术细节

连续型随机变量和概率密度函数的深入理解需要用到数学分支测度论 (measure theory) 的相关内容来扩展概率论。测度论超出了本书的范畴，但我们可以简要勾勒一些测度论用来解决的问题。

在3.3.2节中，我们已经看到连续型向量值随机变量  $\mathbf{x}$  落在某个集合  $S$  中的概率是通过  $p(\mathbf{x})$  对集合  $S$  积分得到的。对于集合  $S$  的一些选择可能会引起悖论。例如，构造两个集合  $S_1$  和  $S_2$  使得  $p(\mathbf{x} \in S_1) + p(\mathbf{x} \in S_2) > 1$  并且  $S_1 \cap S_2 = \emptyset$  是可能的。这些集合通常是大量使用了实数的无限精度来构造的，例如通过构造分形形状 (fractal-shaped) 的集合或者是通过有理数相关集合的平移来定义的集合。<sup>3</sup> 测度论的一个重要贡献就是提供了一些集合的特征使得我们在计算概率时不会遇到悖论。在本书中，我们只对描述相对简单的集合进行积分，所以测度论的这个方面不会成为一个相关考虑。

对于我们的目的，测度论更多的是用来描述那些适用于  $\mathbb{R}^n$  上的大多数点的定理的，而不是只适用于一些小的情况。测度论提供了一种严格的方式来描述那些非

<sup>3</sup>Banach-Tarski 定理给出了这类集合的一个有趣的例子。

常微小的点集。这种集合被称为“零测度 (measure zero)”的。我们不会在本书中给出这个概念的正式定义。然而，直观地理解这个概念是有用的，我们可以认为零测度集在我们的度量空间中不占有任何的体积。例如，在  $\mathbb{R}^2$  空间中，一条直线的测度为零，而填充的多边形具有正的测度。类似的，一个单独的点的测度为零。可数多个零测度集的并仍然是零测度的 (所以所有有理数构成的集合测度为零)。

另外一个有用的测度论中的术语是“几乎处处 (almost everywhere)”。某个性质如果是几乎处处都成立的，那么它在整个空间中除了测度为零的集合以外都是成立的。因为这些例外只在空间中占有极其微小的量，它们在多数应用中都可以被放心地忽略。概率论中的一些重要结果对于离散值成立但对于连续值只能是“几乎处处”成立。

连续型随机变量的另一技术细节，涉及到处理那种相互之间是彼此的确定性函数的连续型变量。假设我们有两个随机变量  $\mathbf{x}$  和  $\mathbf{y}$  满足  $\mathbf{y} = g(\mathbf{x})$ ，其中  $g$  是可逆的、连续可微的函数。可能有人会想  $p_y(\mathbf{y}) = p_x(g^{-1}(\mathbf{y}))$ 。但实际上这并不对。

举一个简单的例子，假设我们有两个标量值随机变量  $x$  和  $y$ ，并且满足  $y = \frac{x}{2}$  以及  $x \sim U(0, 1)$ 。如果我们使用  $p_y(y) = p_x(2y)$ ，那么  $p_y$  除了区间  $[0, \frac{1}{2}]$  以外都为 0，并且在这个区间上的值为 1。这意味着

$$\int p_y(y) dy = \frac{1}{2}, \quad (3.43)$$

而这违背了概率密度的定义 (积分为 1)。

这个常见错误之所以错是因为它没有考虑到引入函数  $g$  后造成的空间变形。回忆一下， $\mathbf{x}$  落在无穷小的体积为  $\delta\mathbf{x}$  的区域内的概率为  $p(\mathbf{x})\delta\mathbf{x}$ 。因为  $g$  可能会扩展或者压缩空间，在  $\mathbf{x}$  空间内的包围着  $\mathbf{x}$  的无穷小体积在  $\mathbf{y}$  空间中可能有不同的体积。

为了看出如何改正这个问题，我们回到标量值的情况。我们需要保持下面这个性质：

$$|p_y(g(x))dy| = |p_x(x)dx|. \quad (3.44)$$

求解上式，我们得到

$$p_y(y) = p_x(g^{-1}(y)) \left| \frac{\partial x}{\partial y} \right| \quad (3.45)$$

或者等价地，

$$p_x(x) = p_y(g(x)) \left| \frac{\partial g(x)}{\partial x} \right|. \quad (3.46)$$

在高维空间中，微分运算扩展为**Jacobi 矩阵** (Jacobian matrix) 的行列式——矩阵的每个元素为  $J_{i,j} = \frac{\partial x_i}{\partial y_j}$ 。因此，对于实数值的向量  $\mathbf{x}$  和  $\mathbf{y}$ ，

$$p_x(\mathbf{x}) = p_y(g(\mathbf{x})) \left| \det \left( \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} \right) \right|. \quad (3.47)$$

## 3.13 信息论

信息论是应用数学的一个分支，主要研究的是对一个信号能够提供信息的多少进行量化。它最初被发明是用来研究在一个含有噪声的信道上用离散的字母表来发送消息，例如通过无线电传输来通信。在这种情况下，信息论告诉我们如何设计最优编码，以及计算从一个特定的概率分布上采样得到、使用多种不同的编码机制的消息的期望长度。在机器学习中，我们也可以把信息论应用在连续型变量上，而信息论中一些消息长度的解释不怎么使用。信息论是电子工程和计算机科学的许多领域的基础。在本书中，我们主要使用信息论的一些关键思想来描述概率分布或者量化概率分布之间的相似性。有关信息论的更多细节，参见Cover and Thomas (2006) 或者MacKay (2003)。

信息论的基本想法是一个不太可能的事件居然发生了，要比一个非常可能的事件发生，能提供更多的信息。消息说：“今天早上太阳升起”信息量是如此之少以至于没有必要发送，但一条消息说：“今天早上有日食”信息量就很丰富。

我们想要通过这种基本想法来量化信息。特别地，

- 非常可能发生的事件信息量要比较少，并且极端情况下，确保能够发生的事件应该没有信息量。
- 更不可能发生的事件要具有更高的信息量。
- 独立事件应具有增量的信息。例如，投掷的硬币两次正面朝上传递的信息量，应该是投掷一次硬币正面朝上的信息量的两倍。

为了满足上述三个性质，我们定义一个事件  $x = x$  的**自信息** (self-information) 为

$$I(x) = -\log P(x). \quad (3.48)$$

在本书中，我们总是用  $\log$  来表示自然对数，底数为  $e$ 。因此我们定义的  $I(x)$  单位是**奈特** (nats)。一奈特是以  $\frac{1}{e}$  的概率观测到一个事件时获得的信息量。其他的材料

中使用底数为 2 的对数，单位是比特 (bit) 或者香农 (shannons)；通过比特度量的信息只是通过奈特度量信息的常数倍。

当  $x$  是连续的，我们使用类似的关于信息的定义，但有些来源于离散形式的性质就丢失了。例如，一个具有单位密度的事件信息量仍然为 0，但是不能保证它一定发生。

自信息只处理单个的输出。我们可以用香农熵 (Shannon entropy) 来对整个概率分布中的不确定性总量进行量化：

$$H(x) = \mathbb{E}_{x \sim P}[I(x)] = -\mathbb{E}_{x \sim P}[\log P(x)], \quad (3.49)$$

也记作  $H(P)$ 。换言之，一个分布的香农熵是指遵循这个分布的事件所产生的期望信息总量。它给出了对，依据概率分布  $P$  生成的符号，进行编码所需的比特数的平均意义上的下界 (如果对数的底是 2 的话，否则单位有所不同)。那些接近确定性的分布 (输出几乎可以确定) 具有较低的熵；那些接近均匀分布的概率分布具有较高的熵。图3.5给出了一个说明。当  $x$  是连续的，香农熵被称为微分熵 (differential entropy)。

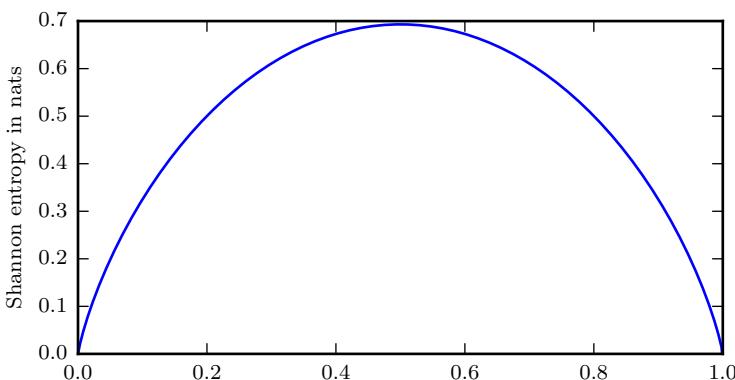


图 3.5: 二值型随机变量的香农熵。该图说明了更接近确定性的分布是如何具有较低的香农熵，而更接近均匀分布的分布是如何具有较高的香农熵。水平轴是  $p$ ，表示二值型随机变量等于 1 的概率。熵由  $(p - 1) \log(1 - p) - p \log p$  给出。当  $p$  接近 0 时，分布几乎是确定的，因为随机变量几乎总是 0。当  $p$  接近 1 时，分布也几乎是确定的，因为随机变量几乎总是 1。当  $p = 0.5$  时，熵是最大的，因为分布在两个结果 (0 和 1) 上是均匀的。

如果我们对于同一个随机变量  $x$  有两个单独的概率分布  $P(x)$  和  $Q(x)$ ，我们可

以使用**KL 散度** (Kullback-Leibler (KL) divergence) 来衡量这两个分布的差异：

$$D_{\text{KL}}(P||Q) = \mathbb{E}_{x \sim P} \left[ \log \frac{P(x)}{Q(x)} \right] = \mathbb{E}_{x \sim P} [\log P(x) - \log Q(x)]. \quad (3.50)$$

在离散型变量的情况下，KL 散度衡量的是，当我们使用一种被设计成能够使得概率分布  $Q$  产生的消息的长度最小的编码时，发送包含由概率分布  $P$  产生的符号的消息时，所需要的额外信息量 (如果我们使用底数为 2 的对数时信息量用比特衡量，但在机器学习中，我们通常用奈特和自然对数。)

KL 散度有很多有用的性质，最重要的是它是非负的。KL 散度为 0 当且仅当  $P$  和  $Q$  在离散型变量的情况下是相同的分布，或者在连续型变量的情况下是“几乎处处”相同的。因为KL 散度是非负的并且衡量的是两个分布之间的差异，它经常被用作分布之间的某种距离。然而，它并不是真的距离因为它不是对称的：对于某些  $P$  和  $Q$ ， $D_{\text{KL}}(P||Q) \neq D_{\text{KL}}(Q||P)$ 。这种非对称性意味着选择  $D_{\text{KL}}(P||Q)$  还是  $D_{\text{KL}}(Q||P)$  影响很大。更多细节可以看图3.6。

一个和KL 散度密切联系的量是交叉熵 (cross-entropy)  $H(P, Q) = H(P) + D_{\text{KL}}(P||Q)$ ，它和KL 散度很像但是缺少左边一项：

$$H(P, Q) = -\mathbb{E}_{x \sim P} \log Q(x). \quad (3.51)$$

针对  $Q$  最小化互信息等价于最小化KL 散度，因为  $Q$  并不参与被省略的那一项。

当计算这些量时，经常会遇到  $0 \log 0$  这个表达式。按照惯例，在信息论中，我们对于这个表达式这样处理  $\lim_{x \rightarrow 0} x \log x = 0$ 。

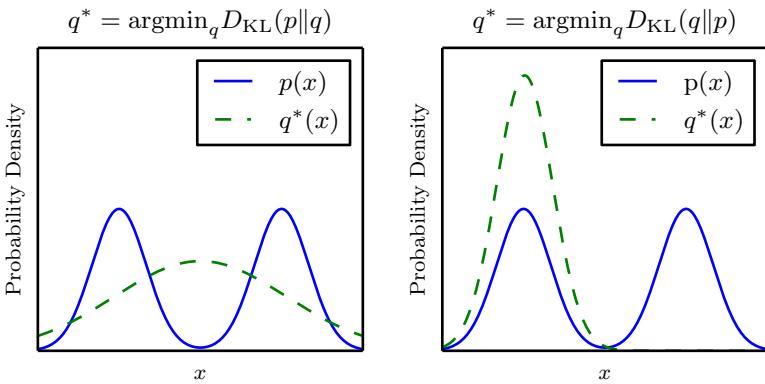


图 3.6: KL 散度是不对称的。假设我们有一个分布  $p(x)$ , 并且希望用另一个分布  $q(x)$  来近似它。我们可以选择最小化  $D_{\text{KL}}(p\|q)$  或最小化  $D_{\text{KL}}(q\|p)$ 。为了说明每种选择的效果, 我们令  $p$  是两个高斯分布的混合, 令  $q$  为单个高斯分布。选择使用KL 散度的哪个方向是取决于问题的。一些应用需要这个近似分布  $q$  在真实分布  $p$  放置高概率的所有地方都放置高概率, 而其他应用需要这个近似分布  $q$  在真实分布  $p$  放置低概率的所有地方都很少放置高概率。KL 散度方向的选择反映了对于每种应用, 优先考虑哪一种选择。(左) 最小化  $D_{\text{KL}}(p\|q)$  的效果。在这种情况下, 我们选择一个  $q$  使得它在  $p$  具有高概率的地方具有高概率。当  $p$  具有两个峰时,  $q$  选择将这些峰模糊到一起, 以便将高概率质量放到所有峰上。(右) 最小化  $D_{\text{KL}}(q\|p)$  的效果。在这种情况下, 我们选择一个  $q$  使得它在  $p$  具有低概率的地方具有低概率。当  $p$  具有两个峰并且这些峰间隔很宽时, 如该图所示, 最小化KL 散度会选择单个峰, 以避免将概率质量放置在  $p$  的多个峰之间的低概率区域中。这里, 我们说明当  $q$  被选择成强调左边峰时的结果。我们也可以通过选择右边峰来得到KL 散度相同的值。如果这些峰没有被足够强的低概率区域分离, 那么KL 散度的这个方向仍然可能选择模糊这些峰。

### 3.14 结构化概率模型

机器学习的算法经常会涉及到在非常多的随机变量上的概率分布。通常, 这些概率分布涉及到的直接相互作用都是介于非常少的变量之间的。使用单个函数来描述整个联合概率分布是非常低效的 (无论是计算还是统计)。

代替使用单一的函数来表示概率分布, 我们可以把概率分布分割成许多因子的乘积形式。例如, 假设我们有三个随机变量  $a, b$  和  $c$ , 并且  $a$  影响  $b$  的取值,  $b$  影响  $c$  的取值, 但是  $a$  和  $c$  在给定  $b$  时是条件独立的。我们可以把全部三个变量的概

率分布重新表示为两个变量的概率分布的连乘形式：

$$p(a, b, c) = p(a)p(b | a)p(c | b). \quad (3.52)$$

这种因子分解可以极大地减少用来描述一个分布的参数的数量。每个因子使用的参数数目是它的变量数目的指数倍。这意味着，如果我们能够找到一种使每个因子分布具有更少变量的因子分解方法，我们就能极大地降低表示联合分布的成本。

我们可以用图来描述这种因子分解。这里我们使用的是图论中的“图”的概念：由一些可以通过边互相连接的顶点的集合构成。当我们用图来表示这种概率分布的因子分解，我们把它称为**结构化概率模型** (structured probabilistic model) 或者**图模型** (graphical model)。

有两种主要的结构化概率模型：有向的和无向的。两种图模型使用图  $\mathcal{G}$ ，其中图的每个节点对应着一个随机变量，连接两个随机变量的边意味着概率分布可以表示成这两个随机变量之间的直接作用。

**有向** (directed) 模型使用带有有向边的图，它们用条件概率分布来表示因子分解，就像上面的例子。特别地，有向模型对于分布中的每一个随机变量  $x_i$  都包含着一个影响因子，这个组成  $x_i$  条件概率的影响因子被称为  $x_i$  的双亲，记为  $Pa_{\mathcal{G}}(x_i)$ ：

$$p(\mathbf{x}) = \prod_i p(x_i | Pa_{\mathcal{G}}(x_i)). \quad (3.53)$$

图3.7给出了一个有向图的例子以及它表示的概率分布的因子分解。

**无向** (undirected) 模型使用带有无向边的图，它们将因子分解表示成一堆函数；不像有向模型那样，这些函数通常不是任何类型的概率分布。 $\mathcal{G}$  中任何全部相连的节点构成的集合被称为团。无向模型中的每个团  $\mathcal{C}^{(i)}$  都伴随着一个因子  $\phi^{(i)}(\mathcal{C}^{(i)})$ 。这些因子仅仅是函数，并不是概率分布。每个因子的输出都必须是非负的，但是并没有像概率分布中那样要求因子的和或者积分为 1。

随机变量的联合概率和所有这些因子的乘积成比例 (proportional)——意味着因子的值越大则可能性越大。当然，不能保证这种乘积的求和为 1。所以我们需要除以一个归一化常数  $Z$  来得到归一化的概率分布，归一化常数  $Z$  被定义为  $\phi$  函数乘积的所有状态的求和或积分。概率分布为：

$$p(\mathbf{x}) = \frac{1}{Z} \prod_i \phi^{(i)}(\mathcal{C}^{(i)}). \quad (3.55)$$

图3.8给出了一个无向图的例子以及它表示的概率分布的因子分解。

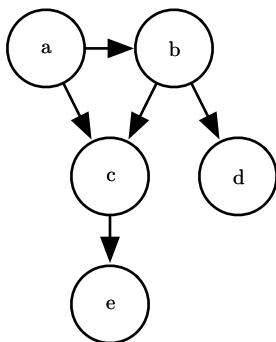


图 3.7: 关于随机变量 a, b, c, d 和 e 的有向图模型。这幅图对应的概率分布可以分解为

$$p(a, b, c, d, e) = p(a)p(b | a)p(c | a, b)p(d | b)p(e | c). \quad (3.54)$$

该图模型使我们能够快速看出此分布的一些性质。例如，a 和 c 直接相互影响，但 a 和 e 只有通过 c 间接相互影响。

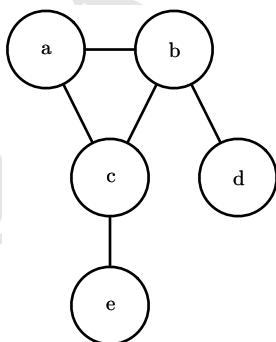


图 3.8: 关于随机变量 a, b, c, d 和 e 的无向图模型。这幅图对应的概率分布可以分解为

$$p(a, b, c, d, e) = \frac{1}{Z} \phi^{(1)}(a, b, c) \phi^{(2)}(b, d) \phi^{(3)}(c, e). \quad (3.56)$$

该图模型使我们能够快速看出此分布的一些性质。例如，a 和 c 直接相互影响，但 a 和 e 只有通过 c 间接相互影响。

请记住，这些图模型表示的因子分解仅仅是描述概率分布的一种语言。它们不是互相排斥的概率分布族。有向或者无向不是概率分布的特性；它是概率分布的一种特殊描述 (description) 所具有的特性，但是任何的概率分布都可以用两种方式进行描述。

在本书第一部分和第二部分中，我们使用结构化概率模型仅仅是作为一门语言，来描述不同的机器学习算法选择表示的直接的概率关系。一直到研究课题的讨论之前，不会需要用到结构化概率模型的深入理解。在第三部分的研究课题中，我们会更为详尽地探讨结构化概率模型。

本章复习了概率论中与深度学习最为相关的一些基本概念。还剩下一些基本的数学工具需要讨论：数值方法。

# 第四章 数值计算

机器学习算法通常需要大量的数值计算。这通常是指通过迭代地更新解来解决数学问题的算法，而不是解析地提供正确解的符号表达。常见的操作包括优化（找到最小化或最大化函数值的参数）和线性方程组的求解。对数字计算机来说实数无法在有限内存下精确表示，因此仅仅计算涉及实数的函数也是困难的。

## 4.1 上溢和下溢

在数字计算机上实现连续数学的根本困难是，我们需要通过有限数量的位模式来表示无限多的实数。这意味着我们在计算机中表示实数时，几乎总会引入一些近似误差。在许多情况下，这仅仅是舍入误差。如果在理论上可行的算法没有被设计为最小化舍入误差的累积，可能就会在实践中失效，因此舍入误差会导致一些问题（特别是许多操作复合时）。

一种特别的毁灭性舍入误差是下溢 (underflow)。当接近零的数被四舍五入为零时发生下溢。许多函数在其参数为零而不是一个很小的正数时才会表现出质的不同。例如，我们通常要避免被零除（一些软件环境将在这种情况下抛出异常，有些会返回一个非数字 (not-a-number) 的占位符）或避免取零的对数（这通常被视为  $-\infty$ ，进一步的算术运算会使其变成非数字）。

另一个极具破坏力的数值错误形式是上溢 (overflow)。当大量级的数被近似为  $\infty$  或  $-\infty$  时发生上溢。进一步的运算通常导致这些无限值变为非数字。

必须对上溢和下溢进行数值稳定的一个例子是softmax 函数 (softmax func-

tion)。softmax 函数经常用于预测与 Multinoulli 分布相关联的概率，定义为

$$\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)}. \quad (4.1)$$

考虑一下当所有  $x_i$  都等于某个常数  $c$  时会发生什么。从理论分析上说，我们可以发现所有的输出都应该为  $\frac{1}{n}$ 。从数值计算上说，当  $c$  量级很大时，这可能不会发生。如果  $c$  是很小的负数， $\exp(c)$  就会下溢。这意味着 softmax 函数的分母会变成 0，所以最后的结果是未定义的。当  $c$  是非常大的正数时， $\exp(c)$  的上溢再次导致整个表达式未定义。这两个困难能通过计算  $\text{softmax}(\mathbf{z})$  同时解决，其中  $\mathbf{z} = \mathbf{x} - \max_i x_i$ 。简单的代数计算表明，softmax 解析上的函数值不会因为从输入向量减去或加上标量而改变。减去  $\max_i x_i$  导致  $\exp$  的最大参数为 0，这排除了上溢的可能性。同样地，分母中至少有一个值为 1 的项，这就排除了因分母下溢而导致被零除的可能性。

还有一个小问题。分子中的下溢仍可以导致整体表达式被计算为零。这意味着，如果我们在计算  $\log \text{softmax}(\mathbf{x})$  时先计算 softmax 再把结果传给 log 函数，会错误地得到  $-\infty$ 。相反，我们必须实现一个单独的函数，并以数值稳定的方式计算  $\log \text{softmax}$ 。我们可以使用相同的技巧稳定  $\log \text{softmax}$  函数。

在大多数情况下，我们没有明确地对本书描述的各种算法所涉及的数值考虑进行详细说明。底层库的开发者在实现深度学习算法时应该牢记数值问题。本书的大多数读者可以简单地依赖保证数值稳定的底层库。在某些情况下，有可能在实现一个新的算法时自动保持数值稳定。Theano(Bergstra et al., 2010a; Bastien et al., 2012a) 就是这样软件包的一个例子，它能自动检测并稳定深度学习中许多常见的数值不稳定的表达式。

## 4.2 病态条件数

条件数表明函数相对于输入的微小变化而变化的快慢程度。输入被轻微扰动而迅速改变的函数对于科学计算来说是可能是有问题的，因为输入中的舍入误差可能导致输出的巨大变化。

考虑函数  $f(\mathbf{x}) = \mathbf{A}^{-1}\mathbf{x}$ 。当  $\mathbf{A} \in \mathbb{R}^{n \times n}$  具有特征值分解时，其条件数为

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|. \quad (4.2)$$

这是最大和最小特征值的模之比。当该数很大时，矩阵求逆对输入的误差特别敏感。

这种敏感性是矩阵本身的固有特性，而不是矩阵求逆期间舍入误差的结果。即使我们乘以完全正确的矩阵逆，病态条件数的矩阵也会放大预先存在的误差。在实践中，该错误将与求逆过程本身的数值误差进一步复合。

### 4.3 基于梯度的优化方法

大多数深度学习算法涉及某种形式的优化。优化指的是改变  $\mathbf{x}$  以最小化或最大化某个函数  $f(\mathbf{x})$  的任务。我们通常以最小化  $f(\mathbf{x})$  指代大多数最优化问题。最大化可经由最小化算法最小化  $-f(\mathbf{x})$  来实现。

我们把要最小化或最大化的函数称为目标函数 (objective function) 或准则 (criterion)。当我们对其进行最小化时，我们也把它称为代价函数 (cost function)、损失函数 (loss function) 或误差函数 (error function)。虽然有些机器学习著作赋予这些名称特殊的意义，但在这本书中我们交替使用这些术语。

我们通常使用一个上标 \* 表示最小化或最大化函数的  $\mathbf{x}$  值。如我们记  $\mathbf{x}^* = \arg \min f(\mathbf{x})$ 。

我们假设读者已经熟悉微积分，这里简要回顾微积分概念如何与优化联系。

假设我们有一个函数  $y = f(x)$ ，其中  $x$  和  $y$  是实数。这个函数的导数 (derivative) 记为  $f'(x)$  或  $\frac{dy}{dx}$ 。导数  $f'(x)$  代表  $f(x)$  在点  $x$  处的斜率。换句话说，它表明如何缩放输入的小变化才能在输出获得相应的变化： $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$ 。

因此导数对于最小化一个函数很有用，因为它告诉我们如何更改  $x$  来略微地改善  $y$ 。例如，我们知道对于足够小的  $\epsilon$  来说， $f(x - \epsilon \text{sign}(f'(x)))$  是比  $f(x)$  小的。因此我们可以将  $x$  往导数的反方向移动一小步来减小  $f(x)$ 。这种技术被称为梯度下降 (gradient descent)(Cauchy, 1847)。图4.1展示了一个例子。

当  $f'(x) = 0$ ，导数无法提供往哪个方向移动的信息。 $f'(x) = 0$  的点称为临界点 (critical point) 或驻点 (stationary point)。一个局部极小点 (local minimum) 意味着这个点的  $f(x)$  小于所有邻近点，因此不可能通过移动无穷小的步长来减小  $f(x)$ 。一个局部极大点 (local maximum) 是  $f(x)$  意味着这个点的  $f(x)$  大于所有邻近点，因此不可能通过移动无穷小的步长来增大  $f(x)$ 。有些临界点既不是最小点也不是最大点。这些点被称为鞍点 (saddle point)。见图4.2给出的各种临界点的例子。

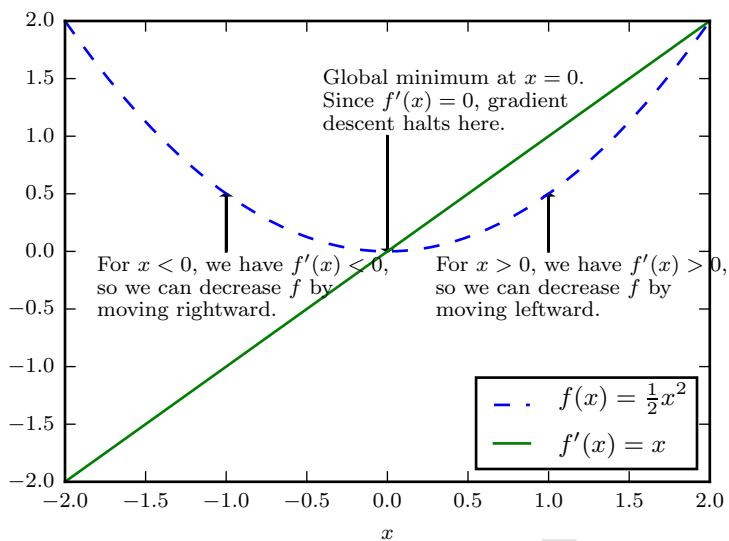


图 4.1: 梯度下降。梯度下降算法如何使用函数导数的示意图, 即沿着函数的下坡方向 (导数反向) 直到最小。

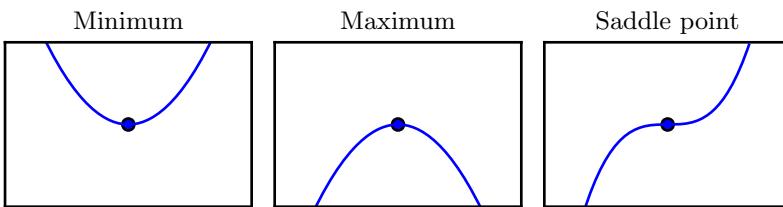


图 4.2: 临界点的类型。一维情况下, 三种临界点的示例。临界点是斜率为零的点。这样的点可以是局部极小点 (local minimum), 其值低于相邻点; 局部极大点 (local maximum), 其值高于相邻点; 或鞍点, 同时存在更高和更低的相邻点。

使  $f(x)$  取得绝对的最小值 (相对所有其他值) 的点是全局最小点 (global minimum)。函数可能只有一个全局最小点或存在多个全局最小点, 还可能存在不是

全局最优的局部极小点。在深度学习的背景下，我们优化的函数可能含有许多不是最优的局部极小点，或许多被非常平坦的区域包围的鞍点。尤其是当输入是多维的时候，所有这些都将使优化变得困难。因此，我们通常寻找  $f$  非常小的值，但在任何形式意义下并不一定是最小。见图4.3的例子。

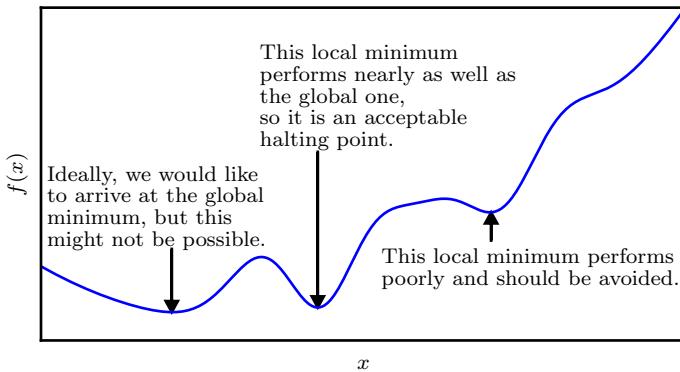


图 4.3: 近似最小化。当存在多个局部极小点或平坦区域时，优化算法可能无法找到全局最小点。在深度学习的背景下，即使找到的解不是真正最小的，但只要它们对应于代价函数的显著低的值，我们通常就能接受这样的解。

我们经常最小化具有多维输入的函数： $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 。为了使“最小化”的概念有意义，输出必须是一维的（标量）。

针对具有多维输入的函数，我们需要用到偏导数 (partial derivative) 的概念。偏导数  $\frac{\partial}{\partial x_i} f(\mathbf{x})$  衡量点  $\mathbf{x}$  处只有  $x_i$  增加时  $f(\mathbf{x})$  如何变化。梯度 (gradient) 是相对一个向量求导的导数： $f$  的导数是包含所有偏导数的向量，记为  $\nabla_{\mathbf{x}} f(\mathbf{x})$ 。梯度的第  $i$  个元素是  $f$  关于  $x_i$  的偏导数。在多维情况下，临界点是梯度中所有元素都为零的点。

在  $\mathbf{u}$  (单位向量) 方向的方向导数 (directional derivative) 是函数  $f$  在  $\mathbf{u}$  方向的斜率。换句话说，方向导数是函数  $f(\mathbf{x} + \alpha \mathbf{u})$  关于  $\alpha$  的导数 (在  $\alpha = 0$  时取得)。使用链式法则，我们可以看到当  $\alpha = 0$  时， $\frac{\partial}{\partial \alpha} f(\mathbf{x} + \alpha \mathbf{u}) = \mathbf{u}^\top \nabla_{\mathbf{x}} f(\mathbf{x})$ 。

为了最小化  $f$ ，我们希望找到使  $f$  下降得最快的方向。计算方向导数：

$$\min_{\mathbf{u}, \mathbf{u}^\top \mathbf{u}=1} \mathbf{u}^\top \nabla_{\mathbf{x}} f(\mathbf{x}) \quad (4.3)$$

$$= \min_{\mathbf{u}, \mathbf{u}^\top \mathbf{u}=1} \|\mathbf{u}\|_2 \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2 \cos \theta \quad (4.4)$$

其中  $\theta$  是  $\mathbf{u}$  与梯度的夹角。将  $\|\mathbf{u}\|_2 = 1$  代入，并忽略与  $\mathbf{u}$  无关的项，就能简化得到  $\min \cos \theta$ 。这在  $\mathbf{u}$  与梯度方向相同时取得最小。换句话说，梯度向量指向上坡，负梯度向量指向下坡。我们在负梯度方向上移动可以减小  $f$ 。这被称为最速下降法 (method of steepest descent) 或梯度下降 (gradient descent)。

最速下降建议新的点为

$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}) \quad (4.5)$$

其中  $\epsilon$  为学习速率 (learning rate)，是一个确定步长大小的正标量。我们可以通过几种不同的方式选择  $\epsilon$ 。普遍的方式是选择一个小常数。有时我们通过计算，选择使方向导数消失的步长。还有一种方法是根据几个  $\epsilon$  计算  $f(\mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}))$ ，并选择其中能产生最小目标函数值的  $\epsilon$ 。这种策略被称为线搜索。

最速下降在梯度的每一个元素为零时收敛（或在实践中，很接近零时）。在某些情况下，我们也许能够避免运行该迭代算法，并通过解方程  $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$  直接跳到临界点。

虽然梯度下降被限制在连续空间中的优化问题，但不断向更好的情况移动一小步（即近似最佳的小移动）的一般概念可以推广到离散空间。递增带有离散参数的目标函数被称为爬山 (hill climbing) 算法 (Russel and Norvig, 2003)。

### 4.3.1 梯度之上：Jacobian 和 Hessian 矩阵

有时我们需要计算输入和输出都为向量的函数的所有偏导数。包含所有这样的偏导数的矩阵被称为**Jacobian** (Jacobian) 矩阵。具体来说，如果我们有一个函数： $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ ， $f$  的 Jacobian 矩阵  $\mathbf{J} \in \mathbb{R}^{n \times m}$  定义为  $J_{i,j} = \frac{\partial}{\partial x_j} f(\mathbf{x})_i$ 。

有时，我们也对导数的导数感兴趣，即二阶导数 (second derivative)。例如，有一个函数  $f: \mathbb{R}^m \rightarrow \mathbb{R}$ ， $f$  的一阶导数 (关于  $x_j$ ) 关于  $x_i$  的导数记为  $\frac{\partial^2}{\partial x_i \partial x_j} f$ 。在一维情况下，我们可以将  $\frac{\partial^2}{\partial x^2} f$  为  $f''(x)$ 。二阶导数告诉我们的一阶导数将如何随着输入的变化而改变。它表示只基于梯度信息的梯度下降步骤是否会产生如我们预期的那样大的改善，因此是重要的。我们可以认为，二阶导数是对曲率的衡量。假设我们有一个二次函数（虽然很多实践中的函数都不是二次，但至少在局部可以很好地用二次近似）。如果这样的函数具有零二阶导数，那就没有曲率。也就是一条完全平坦的线，仅用梯度就可以预测它的值。我们使用沿负梯度方向大小为  $\epsilon$  的下降步，当该梯度是 1 时，代价函数将下降  $\epsilon$ 。如果二阶导数是负的，函数曲线向下凹陷 (向上

凸出)，因此代价函数将下降的比  $\epsilon$  多。如果二阶导数是正的，函数曲线是向上凹陷(向下凸出)，因此代价函数将下降的比  $\epsilon$  少。从图4.4可以看出不同形式的曲率如何影响基于梯度的预测值与真实的代价函数值的关系。

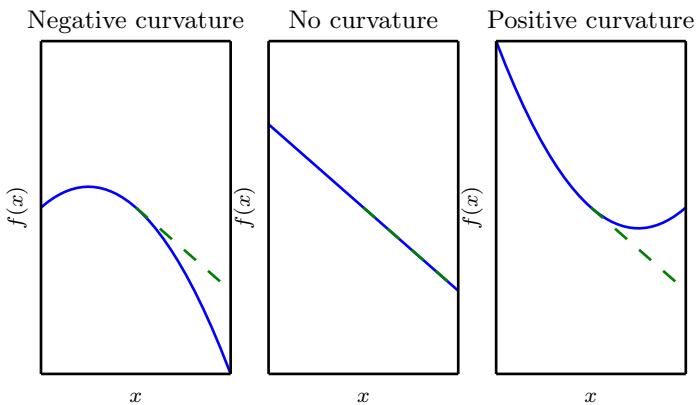


图 4.4: 二阶导数确定函数的曲率。这里我们展示具有各种曲率的二次函数。虚线表示我们仅根据梯度信息进行梯度下降后预期的代价函数值。对于负曲率，代价函数实际上比梯度预测下降得更快。没有曲率时，梯度正确预测下降值。对于正曲率，函数比预期下降得更慢，并且最终会开始增加，因此太大的步骤实际上可能会无意地增加函数值。

当我们的函数具有多维输入时，二阶导数也有很多。可以将这些导数合并成一个矩阵，称为**Hessian** (Hessian) 矩阵。Hessian矩阵  $\mathbf{H}(f)(\mathbf{x})$  定义为

$$\mathbf{H}(f)(\mathbf{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}). \quad (4.6)$$

Hessian等价于梯度的Jacobian矩阵。

微分算子在任何二阶偏导连续的点处可交换，也就是它们的顺序可以互换：

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\mathbf{x}). \quad (4.7)$$

这意味着  $H_{i,j} = H_{j,i}$ ，因此Hessian矩阵在这些点上是对称的。在深度学习背景下，我们遇到的大多数函数的Hessian几乎处处都是对称的。因为Hessian矩阵是实对称的，我们可以将其分解成一组实特征值和特征向量的正交。在特定方向  $\mathbf{d}$  上的二阶导数可以写成  $\mathbf{d}^\top \mathbf{H} \mathbf{d}$ 。当  $\mathbf{d}$  是  $\mathbf{H}$  的一个特征向量时，这个方向的二阶导数就是对应的特征值。对于其他的方向  $\mathbf{d}$ ，方向二阶导数是所有特征值的加权平均，权重在 0

和 1 之间，且与  $\mathbf{d}$  夹角越小的特征向量有更大的权重。最大特征值确定最大二阶导数，最小特征值确定最小二阶导数。

我们可以通过（方向）二阶导数预期一个梯度下降步骤能表现得多好。我们在当前点  $\mathbf{x}^{(0)}$  处作函数  $f(\mathbf{x})$  的近似二阶泰勒级数：

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{g} + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{H}(\mathbf{x} - \mathbf{x}^{(0)}), \quad (4.8)$$

其中  $\mathbf{g}$  是梯度， $\mathbf{H}$  是  $\mathbf{x}^{(0)}$  点的Hessian。如果我们使用学习速率  $\epsilon$ ，那么新的点  $\mathbf{x}$  将会是  $\mathbf{x}^{(0)} - \epsilon\mathbf{g}$ 。代入上述的近似，可得

$$f(\mathbf{x}^{(0)} - \epsilon\mathbf{g}) \approx f(\mathbf{x}^{(0)}) - \epsilon\mathbf{g}^\top \mathbf{g} + \frac{1}{2}\epsilon^2\mathbf{g}^\top \mathbf{H}\mathbf{g}. \quad (4.9)$$

其中有 3 项：函数的原始值、函数斜率导致的预期改善、函数曲率导致的校正。当这最后一项太大时，梯度下降实际上是可能向上移动的。当  $\mathbf{g}^\top \mathbf{H}\mathbf{g}$  为零或负时，近似的泰勒级数表明增加  $\epsilon$  将永远导致  $f$  的下降。在实践中，泰勒级数不会在  $\epsilon$  大的时候也保持准确，因此在这种情况下我们必须采取更启发式的选择。当  $\mathbf{g}^\top \mathbf{H}\mathbf{g}$  为正时，通过计算可得，使近似泰勒级数下降最多的最优步长为

$$\epsilon^* = \frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{H}\mathbf{g}}. \quad (4.10)$$

最坏的情况下， $\mathbf{g}$  与  $\mathbf{H}$  最大特征值  $\lambda_{\max}$  对应的特征向量对齐，则最优步长是  $\frac{1}{\lambda_{\max}}$ 。我们要最小化的函数能用二次函数很好地近似的情况下，Hessian的特征值决定了学习速率的量级。

二阶导数还可以被用于确定一个临界点是否是局部极大点、局部极小点或鞍点。回想一下，在临界点处  $f'(x) = 0$ 。而  $f''(x) > 0$  意味着  $f'(x)$  会随着我们移向右边而增加，移向左边而减小，也就是  $f'(x - \epsilon) < 0$  和  $f'(x + \epsilon) > 0$  对足够小的  $\epsilon$  成立。换句话说，当我们移向右边，斜率开始指向右边的上坡，当我们移向左边，斜率开始指向左边的上坡。因此我们得出结论，当  $f'(x) = 0$  且  $f''(x) > 0$  时， $\mathbf{x}$  是一个局部极小点。同样，当  $f'(x) = 0$  且  $f''(x) < 0$  时， $\mathbf{x}$  是一个局部极大点。这就是所谓的二阶导数测试 (second derivative test)。不幸的是，当  $f''(x) = 0$  时测试是不确定的。在这种情况下， $\mathbf{x}$  可以是一个鞍点或平坦区域的一部分。

在多维情况下，我们需要检测函数的所有二阶导数。利用Hessian的特征值分解，我们可以将二阶导数测试扩展到多维情况。在临界点处 ( $\nabla_{\mathbf{x}}f(\mathbf{x}) = 0$ )，我们通过检测Hessian的特征值来判断该临界点是一个局部极大点、局部极小点还是鞍点。

当Hessian是正定的（所有特征值都是正的），则该临界点是局部极小点。因为方向二阶导数在任意方向都是正的，参考单变量的二阶导数测试就能得出此结论。同样的，当Hessian是负定的（所有特征值都是负的），这个点就是局部极大点。在多维情况下，实际上可以找到确定该点是否为鞍点的积极迹象（某些情况下）。如果Hessian的特征值中至少一个是正的且至少一个是负的，那么  $\mathbf{x}$  是  $f$  某个横截面的局部极大点，却是另一个横截面的局部极小点。见图4.5中的例子。最后，多维二阶导数测试可能像单变量版本那样是不确定的。当所有非零特征值是同号的且至少有一个特征值是0时，这个检测就是不确定的。这是因为单变量的二阶导数测试在零特征值对应的横截面上是不确定的。

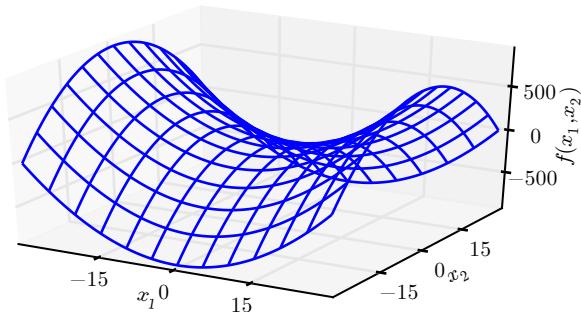


图 4.5: 既有正曲率又有负曲率的鞍点。示例中的函数是  $f(\mathbf{x}) = x_1^2 - x_2^2$ 。函数沿  $x_1$  轴向上弯曲。 $x_1$  轴是Hessian的一个特征向量，并且具有正特征值。函数沿  $x_2$  轴向下弯曲。该方向对应于Hessian负特征值的特征向量。名称“鞍点”源自该处函数的鞍状形状。这是具有鞍点函数的典型示例。维度多于一个时，鞍点不一定要具有 0 特征值：仅需要同时具有正特征值和负特征值。我们可以想象这样一个鞍点（具有正负特征值）在一个横截面内是局部极大点，另一个横截面内是局部极小点。

多维情况下，单个点处每个方向上的二阶导数是不同。Hessian的条件数衡量这些二阶导数的变化范围。当Hessian的条件数很差时，梯度下降法也会表现得很差。这是因为一个方向上的导数增加得很快，而在另一个方向上增加得很慢。梯度下降不知道导数的这种变化，所以它不知道应该优先探索导数长期为负的方向。病态条件数也导致很难选择合适的步长。步长必须足够小，以免冲过最小而向具有较强的正曲率方向上升。这通常意味着步长太小，以致于在其他较小曲率的方向上进展不明显。见图4.6的例子。

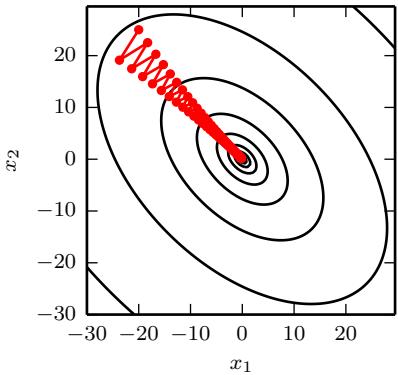


图 4.6: 梯度下降无法利用包含在Hessian矩阵中的曲率信息。这里我们使用梯度下降来最小化Hessian矩阵条件数为 5 的二次函数  $f(\mathbf{x})$ 。这意味着最大曲率方向具有比最小曲率方向多五倍的曲率。在这种情况下, 最大曲率在  $[1, 1]^\top$  方向上, 最小曲率在  $[1, -1]^\top$  方向上。红线表示梯度下降的路径。这个非常细长的二次函数类似一个长峡谷。梯度下降把时间浪费于在峡谷壁反复下降, 因为它们是最陡峭的特征。由于步长有点大, 有超过函数底部的趋势, 因此需要在下一次迭代时在对面的峡谷壁下降。与指向该方向的特征向量对应的Hessian的大的正特征值表示该方向上的导数快速增加, 因此基于Hessian的优化算法可以预测, 在此情况下最陡峭方向实际上不是有前途的搜索方向。

使用Hessian矩阵的信息指导搜索可以解决这个问题。其中最简单的方法是牛顿法 (Newton's method)。牛顿法基于一个二阶泰勒展开来近似  $\mathbf{x}^{(0)}$  附近的  $f(\mathbf{x})$ :

$$f(\mathbf{x}) \approx f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^\top \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)}) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^{(0)})^\top \mathbf{H}(f)(\mathbf{x}^{(0)}) (\mathbf{x} - \mathbf{x}^{(0)}). \quad (4.11)$$

接着通过计算, 我们可以得到这个函数的临界点:

$$\mathbf{x}^* = \mathbf{x}^{(0)} - \mathbf{H}(f)(\mathbf{x}^{(0)})^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}^{(0)}). \quad (4.12)$$

当  $f$  是一个正定二次函数时, 牛顿法只要应用一次式(4.12)就能直接跳到函数的最小点。如果  $f$  不是一个真正二次但能在局部近似为正定二次, 牛顿法则需要多次迭代应用式(4.12)。迭代地更新近似函数和跳到近似函数的最小点可以比梯度下降更快地到达临界点。这在接近局部极小点时是一个特别有用的性质, 但是在鞍点附近是有害的。如式(8.2.3)所讨论的, 当附近的临界点是最小点 (Hessian的所有特征值都是正的) 时牛顿法才适用, 而梯度下降不会被吸引到鞍点(除非梯度指向鞍点)。

仅使用梯度信息的优化算法被称为一阶优化算法 (first-order optimization algo-

rithms)，如梯度下降。使用Hessian矩阵的优化算法被称为**二阶最优化算法** (second-order optimization algorithms)(Nocedal and Wright, 2006)，如牛顿法。

在本书大多数上下文中使用的优化算法适用于各种各样的函数，但几乎都没有保证。因为在深度学习中使用的函数族是相当复杂的，所以深度学习算法往往缺乏保证。在许多其他领域，优化的主要方法是为有限的函数族设计优化算法。

在深度学习的背景下，限制函数满足**Lipschitz 连续** (Lipschitz continuous) 或其导数Lipschitz连续可以获得一些保证。Lipschitz连续函数的变化速度以**Lipschitz 常数** (Lipschitz constant)  $\mathcal{L}$  为界：

$$\forall \mathbf{x}, \forall \mathbf{y}, |f(\mathbf{x}) - f(\mathbf{y})| \leq \mathcal{L} \|\mathbf{x} - \mathbf{y}\|_2. \quad (4.13)$$

这个属性允许我们量化我们的假设——梯度下降等算法导致的输入的微小变化将使输出只产生微小变化，因此是很有用的。Lipschitz连续性也是相当弱的约束，并且深度学习中很多优化问题经过相对较小的修改后就能变得Lipschitz 连续。

最成功的特定优化领域或许是**凸优化** (Convex optimization)。凸优化通过更强的限制提供更多的保证。凸优化算法只对凸函数适用——即Hessian处处半正定的函数。因为这些函数没有鞍点而且其所有局部极小点必然是全局最小点，所以表现很好。然而，深度学习中的大多数问题都难以表示成凸优化的形式。凸优化仅用作的一些深度学习算法的子程序。凸优化中的分析思路对证明深度学习算法的收敛性非常有用，然而一般来说，深度学习背景下的凸优化的重要性大大减少。有关凸优化的详细信息，见Boyd and Vandenberghe (2004) 或Rockafellar (1997)。

## 4.4 约束优化

有时候，在  $\mathbf{x}$  的所有可能值下最大化或最小化一个函数  $f(\mathbf{x})$  不是我们所希望的。相反，我们可能希望在  $\mathbf{x}$  的某些集合  $S$  中找  $f(\mathbf{x})$  的最大值或最小值。这被称为**约束优化** (constrained optimization)。在约束优化术语中，集合  $S$  内的点  $\mathbf{x}$  被称为**可行** (feasible) 点。

我们常常希望找到在某种意义上小的解。针对这种情况下的常见方法是强加一个范数约束，如  $\|\mathbf{x}\| \leq 1$ 。

约束优化的一个简单方法是将约束考虑在内后简单地对梯度下降进行修改。如果我们使用一个小的恒定步长  $\epsilon$ ，我们可以先取梯度下降的单步结果，然后将结果投

影回  $\mathbb{S}$ 。如果我们使用线搜索，我们只能在步长为  $\epsilon$  范围内搜索可行的新  $\mathbf{x}$  点，或者我们可以将线上的每个点投影到约束区域。如果可能的话，在梯度下降或线搜索前将梯度投影到可行域的切空间会更高效 (Rosen, 1960)。

一个更复杂的方法是设计一个不同的、无约束的优化问题，其解可以转化成原始约束优化问题的解。例如，我们要在  $\mathbf{x} \in \mathbb{R}^2$  中最小化  $f(\mathbf{x})$ ，其中  $\mathbf{x}$  约束为具有单位  $L^2$  范数。我们可以关于  $\theta$  最小化  $g(\theta) = f([\cos \theta, \sin \theta]^\top)$ ，最后返回  $[\cos \theta, \sin \theta]$  作为原问题的解。这种方法需要创造性；优化问题之间的转换必须专门根据我们遇到的每一种情况进行设计。

**Karush-Kuhn-Tucker (KKT) 方法<sup>1</sup>**是针对约束优化非常通用的解决方案。为介绍KKT方法，我们引入一个称为广义 Lagrangian (generalized Lagrangian) 或广义 Lagrange 函数 (generalized Lagrange function) 的新函数。

为了定义Lagrangian，我们先要通过等式和不等式的形式描述  $\mathbb{S}$ 。我们希望通过  $m$  个函数  $g^{(i)}$  和  $n$  个函数  $h^{(j)}$  描述  $\mathbb{S}$ ，那么  $\mathbb{S}$  可以表示为  $\mathbb{S} = \{\mathbf{x} \mid \forall i, g^{(i)}(\mathbf{x}) = 0 \text{ and } \forall j, h^{(j)}(\mathbf{x}) \leq 0\}$ 。其中涉及  $g^{(i)}$  的等式称为等式约束 (equality constraint)，涉及  $h^{(j)}$  的不等式称为不等式约束 (inequality constraint)。

我们为每个约束引入新的变量  $\lambda_i$  和  $\alpha_j$ ，这些新变量被称为KKT乘子。广义 Lagrangian可以如下定义：

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_i \lambda_i g^{(i)}(\mathbf{x}) + \sum_j \alpha_j h^{(j)}(\mathbf{x}). \quad (4.14)$$

现在，我们可以通过优化无约束的广义 Lagrangian解决约束最小化问题。只要存在至少一个可行点且  $f(\mathbf{x})$  不允许取  $\infty$ ，那么

$$\min_{\mathbf{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \quad (4.15)$$

与如下函数有相同的最优目标函数值和最优点集  $\mathbf{x}$

$$\min_{\mathbf{x} \in \mathbb{S}} f(\mathbf{x}). \quad (4.16)$$

这是因为当约束满足时，

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\mathbf{x}), \quad (4.17)$$

---

<sup>1</sup>KKT方法是 Lagrange 乘子法（只允许等式约束）的推广

而违反任意约束时，

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \infty. \quad (4.18)$$

这些性质保证不可行点不会是最佳的，并且可行点范围内的最优点不变。

要解决约束最大化问题，我们可以构造  $-f(\boldsymbol{x})$  的广义 Lagrange 函数，从而导致以下优化问题：

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \quad (4.19)$$

我们也可将其转换为在外层最大化的一个问题：

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) - \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \quad (4.20)$$

等式约束对应项的符号并不重要；因为优化可以自由选择每个  $\lambda_i$  的符号，我们可以随意将其定义为加法或减法。

不等式约束特别有趣。如果  $h^{(i)}(\boldsymbol{x}^*) = 0$ ，我们就说这个约束  $h^{(i)}(\boldsymbol{x})$  是活跃 (active) 的。如果约束不是活跃的，则有该约束的问题的解与去掉该约束的问题的解至少存在一个相同的局部解。一个不活跃约束有可能排除其他解。例如，整个区域（代价相等的宽平区域）都是全局最优点的凸问题可能因约束消去其中的某个子区域，或在非凸问题的情况下，收敛时不活跃的约束可能排除了较好的局部驻点。然而，无论不活跃的约束是否被包括在内，收敛时找到的点仍然是一个驻点。因为一个不活跃的约束  $h^{(i)}$  必有负值，那么  $\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$  中的  $\alpha_i = 0$ 。因此，我们可以观察到在该解中  $\boldsymbol{\alpha} \odot \mathbf{h}(\boldsymbol{x}) = 0$ 。换句话说，对于所有的  $i$ ,  $\alpha_i \geq 0$  或  $h^{(j)}(\boldsymbol{x}) \leq 0$  在收敛时必有一个是活跃的。为了获得关于这个想法的一些直观解释，我们可以说这个解是由不等式强加的边界，我们必须通过对应的 KKT 乘子影响  $\boldsymbol{x}$  的解，或者不等式对解没有影响，我们则归零 KKT 乘子。

可以使用一组简单性质描述约束优化问题的最优点。这些性质称为 **Karush-Kuhn-Tucker (KKT)** 条件 (Karush, 1939; Kuhn and Tucker, 1951)。这些是确定一个点是最优点的必要条件，但不一定是充分条件。这些条件是：

- 广义 Lagrangian 的梯度为零。
- 所有关于  $\boldsymbol{x}$  和 KKT 乘子的约束都满足。

- 不等式约束显示的“互补松弛性”： $\alpha \odot h(\mathbf{x}) = 0$ 。

有关KKT方法的详细信息，请参阅Nocedal and Wright (2006)。

## 4.5 实例：线性最小二乘

假设我们希望找到最小化下式的  $\mathbf{x}$  值

$$f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2. \quad (4.21)$$

专门线性代数算法能够高效地解决这个问题；但是，我们也可以探索如何使用基于梯度的优化来解决这个问题，这可以作为这些技术是如何工作的一个简单例子。

首先，我们计算梯度：

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \mathbf{A}^\top (\mathbf{Ax} - \mathbf{b}) = \mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b}. \quad (4.22)$$

然后，我们可以采用小的步长，按照这个梯度下降。见算法4.1中的详细信息。

---

**算法 4.1** 从任意点  $\mathbf{x}$  开始，使用梯度下降关于  $\mathbf{x}$  最小化  $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|_2^2$  的算法。

---

将步长 ( $\epsilon$ ) 和容差 ( $\delta$ ) 设为小的正数。

```

while  $\|\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b}\|_2 > \delta$  do
     $\mathbf{x} \leftarrow \mathbf{x} - \epsilon (\mathbf{A}^\top \mathbf{Ax} - \mathbf{A}^\top \mathbf{b})$ 
end while

```

---

我们也可以使用牛顿法解决这个问题。因为在这个情况下真正的函数是二次的，牛顿法所用的二次近似是精确的，该算法会在一步后收敛到全局最小点。

现在假设我们希望最小化同样的函数，但受  $\mathbf{x}^\top \mathbf{x} \leq 1$  的约束。要做到这一点，我们引入Lagrangian

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda(\mathbf{x}^\top \mathbf{x} - 1). \quad (4.23)$$

现在，我们解决以下问题

$$\min_{\mathbf{x}} \max_{\lambda, \lambda \geq 0} L(\mathbf{x}, \lambda). \quad (4.24)$$

可以用Moore-Penrose伪逆： $\mathbf{x} = \mathbf{A}^+ \mathbf{b}$  找到无约束最小二乘问题的最小范数解。如果这一点是可行，那么这也是约束问题的解。否则，我们必须找到约束是活跃的解。关于  $\mathbf{x}$  对Lagrangian微分，我们得到方程

$$\mathbf{A}^\top \mathbf{A} \mathbf{x} - \mathbf{A}^\top \mathbf{b} + 2\lambda \mathbf{x} = 0. \quad (4.25)$$

这就告诉我们，该解的形式将会是

$$\mathbf{x} = (\mathbf{A}^\top \mathbf{A} + 2\lambda \mathbf{I})^{-1} \mathbf{A}^\top \mathbf{b}. \quad (4.26)$$

$\lambda$  的选择必须使结果服从约束。我们可以关于  $\lambda$  进行梯度上升找到这个值。为了做到这一点，观察

$$\frac{\partial}{\partial \lambda} L(\mathbf{x}, \lambda) = \mathbf{x}^\top \mathbf{x} - 1. \quad (4.27)$$

当  $\mathbf{x}$  的范数超过 1 时，该导数是正的，所以为了跟随导数上坡并相对  $\lambda$  增加Lagrangian，我们需要增加  $\lambda$ 。因为  $\mathbf{x}^\top \mathbf{x}$  的惩罚系数增加了，求解关于  $\mathbf{x}$  的线性方程现在将得到具有较小范数的解。求解线性方程和调整  $\lambda$  的过程一直持续到  $\mathbf{x}$  具有正确的范数并且关于  $\lambda$  的导数是 0。

本章总结了开发机器学习算法所需的数学基础。现在，我们已经准备好建立和分析一些成熟学习系统。

# 第五章 机器学习基础

深度学习是机器学习的一个特定分支。要想学好深度学习，必须对机器学习的基本原理有深刻的理解。本章将探讨贯穿本书其余部分的一些机器学习重要原理。我们建议新手读者或是希望更全面了解的读者参考一些更全面覆盖基础知识的机器学习参考书，例如Murphy (2012) 或者Bishop (2006)。如果你已经熟知机器学习，可以跳过前面的部分，前往第5.11节。第5.11节涵盖了一些传统机器学习技术观点，这些技术对深度学习的发展有着深远影响。

首先，我们将介绍学习算法的定义，并介绍一个简单的示例：线性回归算法。接下来，我们会探讨拟合训练数据和泛化到新数据之间有哪些不同的挑战。大部分机器学习算法都有超参数（必须在学习算法外设定）；我们将讨论如何使用额外的数据设置超参数。机器学习本质上属于应用统计学，更多关注于如何用计算机统计地估计复杂函数，不太关注为这些函数提供置信区间；因此我们会探讨两种统计学的主要方法：频率估计和贝叶斯推断。大部分机器学习算法可以分成监督学习和无监督学习两类；我们将探讨不同的分类，并为每类提供一些简单的机器学习算法作为示例。大部分深度学习算法都基于随机梯度下降求解。我们将介绍如何组合不同的算法部分，例如优化算法、代价函数、模型和数据集，来建立一个机器学习算法。最后在第5.11节，我们会介绍一些限制传统机器学习泛化能力的因素。这些挑战促进了解决这些问题的深度学习算法的发展。

## 5.1 学习算法

机器学习算法是一种可以从数据中学习的算法。然而，我们所谓的“学习”是什么意思呢？Mitchell (1997) 提供了一个简洁的定义：“对于某类任务  $T$  和性能度量  $P$ ，一个计算机程序被认为可以从经验  $E$  中学习是指，通过经验  $E$  改进后，它在任

务  $T$  上由性能度量  $P$  衡量的性能有所提升。”经验  $E$ ，任务  $T$  和性能度量  $P$  的定义范围非常宽广，在本书中我们并不会去试图解释这些定义的具体意义。相反，我们会在接下来的章节中提供直观的解释和示例来介绍不同的任务、性能度量和经验，这些将被用来构建机器学习算法。

### 5.1.1 任务 $T$

机器学习可以让我们解决一些人为设计和实现固定程序很难解决的问题。从科学和哲学的角度来看，机器学习受到关注是因为提高我们对机器学习的认识需要提高我们对智能背后原理的理解。

如果考虑“任务”比较正式的定义，那么学习的过程并不是任务。

在相对正式的“任务”定义中，学习过程本身并不是任务。学习是我们所谓的获取完成任务的能力。例如，我们的目标是使机器人能够行走，那么行走便是任务。我们可以编程让机器人学会如何行走，或者可以编写特定的指令，人工指导机器人如何行走。

通常机器学习任务定义为机器学习系统该如何处理样本 (example)。样本是指我们从某些希望机器学习系统处理的对象或事件中收集到的已经量化的特征 (feature) 的集合。我们通常会将样本表示成一个向量  $\mathbf{x} \in \mathbb{R}^n$ ，其中向量的每一个元素  $x_i$  是一个特征。例如，一张图片的特征通常是指这张图片的像素值。

机器学习可以解决很多类型的任务。一些非常常见的机器学习任务列举如下：

- **分类：**在这类任务中，计算机程序需要指定某些输入属于  $k$  类中的哪一类。为了完成这个任务，学习算法通常会返回一个函数  $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$ 。当  $y = f(\mathbf{x})$  时，模型为向量  $\mathbf{x}$  所代表的输入指定数码  $y$  所代表的类别。还有一些其他的分类问题，例如， $f$  输出的是不同类别的概率分布。分类任务中有一类是对象识别，输入是图片（通常用一组像素亮度值表示），输出是表示图片物体的数码。例如 Willow Garage PR2 机器人像服务员一样识别不同饮料，并送给点餐的顾客 (Goodfellow et al., 2010)。目前，最好的对象识别工作正是基于深度学习(Krizhevsky et al., 2012a; Ioffe and Szegedy, 2015)。对象识别同时也是计算机识别人脸的基本技术，可用于标记相片集中的人脸 (Taigman et al., 2014)，有助于计算机更自然地和用户交互。
- **输入缺失分类：**当输入向量的每个度量不被保证的时候，分类问题将会变得更

有挑战性。为了解决分类任务，学习算法只需要定义一个从输入向量映射到输出类别的函数。当一些输入可能丢失时，学习算法必须学习一组函数，而不是单个分类函数。每个函数对应着分类具有不同缺失输入子集的  $\mathbf{x}$ 。这种情况在医疗诊断中经常出现，因为很多类型的医学测试是昂贵的，对身体有害的。有效地定义这样一个大集合函数的方法是学习所有相关变量的概率分布，然后通过边缘化缺失变量来解决分类任务。使用  $n$  个输入变量，我们现在可以获得每个可能的缺失输入集合所需的所有  $2^n$  个不同的分类函数，但是计算机程序仅需要学习一个描述联合概率分布的函数。参见Goodfellow *et al.* (2013d) 了解以这种方式将深度概率模型应用于这样任务的示例。本节中描述的许多其他任务也可以推广到缺失输入的情况；缺失输入分类只是机器学习能够解决的问题的一个示例。

- **回归：**这类任务中，计算机程序会对给定输入预测数值。为了解决这个问题，学习算法会输出函数  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 。除了返回结果的形式不一样外，这类问题和分类问题是很像的。这类任务的一个示例是预测投保人的索赔金额（用于设置保险费），或者预测证券未来的价格。这类预测也用在算法交易中。
- **转录：**这类任务中，机器学习系统观测一些相对非结构化表示的数据，并转录信息为离散的文本形式。例如，光学字符识别要求计算机程序根据文本图片返回文字序列（ASCII 码或者 Unicode 码）。谷歌街景以这种方式使用深度学习处理街道编号 (Goodfellow *et al.*, 2014d)。另一个例子是语音识别，计算机程序输入一段音频波形，输出一序列音频记录中所说的字符或单词 ID 的编码。深度学习是现代语音识别系统的重要组成部分，广泛用于各大公司，包括微软，IBM 和谷歌 (Hinton *et al.*, 2012a)。
- **机器翻译：**在机器翻译任务中，输入是一种语言的符号序列，计算机程序必须将其转化成另一种语言的符号序列。这通常适用于自然语言，如将英语译成法语。最近，深度学习已经开始在这个任务上产生重要影响 (Sutskever *et al.*, 2014; Bahdanau *et al.*, 2015)。
- **结构化输出：**结构化输出任务的输出是向量或者其他包含多个值的数据结构，并且构成输出的这些不同元素间具有重要关系。这是一个很大的范畴，包括上面转录任务和翻译任务在内的很多其他任务。例如语法分析——映射自然语言句子到语法结构树，并标记树的节点为动词，名词，副词等等。参考Collobert (2011) 应用深度学习到语法分析。另一个例子是图像的像素级分割，将每一

一个像素分配到特定类别。例如，深度学习可用于标注航拍照片中的道路位置 (Mnih and Hinton, 2010)。在这些标注型的任务中，输出的结构形式不需要和输入尽可能相似。例如，在图片标题中，计算机程序观察到一幅图，输出描述这幅图的自然语言句子 (Kiros *et al.*, 2014a,b; Mao *et al.*, 2014; Vinyals *et al.*, 2015b; Donahue *et al.*, 2014; Karpathy and Li, 2015; Fang *et al.*, 2015; Xu *et al.*, 2015)。这类任务被称为结构化输出任务是因为输出值之间内部紧密相关。例如，图片标题程序输出的单词必须组合成一个通顺的句子。

- **异常检测：**这类任务中，计算机程序在一组事件或对象中筛选，并标记不正常或非典型的个体。异常检测任务的一个例子是信用卡欺诈检测。通过对你的购买习惯建模，信用卡公司可以检测到你的卡是否被滥用。如果窃贼窃取你的信用卡或信用卡信息，窃贼采购物品的分布通常和你的不同。当该卡发生了不正常的购买行为时，信用卡公司可以尽快冻结该卡以防欺诈。参考Chandola *et al.* (2009) 了解欺诈检测方法。
- **合成和采样：**这类任务中，机器学习程序生成一些和训练数据相似的新样本。通过机器学习，合成和采样可能在媒体应用中非常有用，可以避免艺术家大量昂贵或者乏味费时的手动工作。例如，视频游戏可以自动生成大型物体或风景的纹理，而不是让艺术家手动标记每个像素 (Luo *et al.*, 2013)。在某些情况下，我们希望采样或合成过程可以根据给定的输入生成一些特定类型的输出。例如，在语音合成任务中，我们提供书写的句子，要求程序输出这个句子语音的音频波形。这是一类结构化输出任务，但是多了每个输入并非只有一个正确输出的条件，我们明确希望输出有很大的偏差，使结果看上去更加自然和真实。
- **缺失值填补：**这类任务中，机器学习算法给定一个新样本  $\mathbf{x} \in \mathbb{R}^n$ ， $\mathbf{x}$  中某些元素  $x_i$  缺失。算法必须填补这些缺失值。
- **去噪：**这类任务中，机器学习算法的输入是，由未知破坏过程从干净样本  $\mathbf{x} \in \mathbb{R}^n$  得到的污染样本  $\tilde{\mathbf{x}} \in \mathbb{R}^n$ 。算法根据污染后的样本  $\tilde{\mathbf{x}}$  预测干净的样本  $\mathbf{x}$ ，或者更一般地预测条件概率分布  $P(\mathbf{x} | \tilde{\mathbf{x}})$ 。
- **密度估计或概率分布律函数估计：**在密度估计问题中，机器学习算法学习函数  $p_{\text{model}} : \mathbb{R}^n \rightarrow \mathbb{R}$ ，其中  $p_{\text{model}}(\mathbf{x})$  可以解释成样本采样空间的概率密度函数（如果  $\mathbf{x}$  是连续的）或者概率分布律函数（如果  $\mathbf{x}$  是离散的）。要做好这样的任务（当我们讨论性能度量  $P$  时，我们会明确定义任务是什么），算法需要学习观测

到的数据的结构。算法必须知道什么情况下样本聚堆出现，什么情况下不太可能出现。以上描述的大多数任务都要求学习算法至少能隐式地抓住概率分布的结构。密度估计可以让我们显式地抓住该分布。原则上，我们可以在该分布上计算以便解决其他任务。例如，如果我们通过密度估计得到了概率分布  $p(\mathbf{x})$ ，我们可以用该分布解决缺失值填补任务。如果  $x_i$  的值是缺失的，但是其他的变量值  $\mathbf{x}_{-i}$  已知，那么我们可以得到条件概率分布  $p(x_i | \mathbf{x}_{-i})$ 。现实中，密度估计并非能够解决所有这类问题，因为在很多情况下  $p(\mathbf{x})$  是难以计算的。

当然，还有很多其他同类型或其他类型的任务。这里我们列举的任务类型只是用来介绍机器学习可以做哪些任务，并非严格地定义机器学习任务分类。

### 5.1.2 性能度量 $P$

为了评估机器学习算法的能力，我们必须设计其性能的定量度量。通常性能度量  $P$  是特定于系统执行的任务  $T$  而言的。

对于诸如分类，缺失输入分类和转录任务，我们通常度量模型的准确率 (accuracy)。准确率是指该模型输出正确结果的样本比例。我们也可以通过错误率 (error rate) 得到相同的信息。错误率是指该模型输出错误结果的样本比例。我们通常把错误率称作 0-1 损失的期望。在一个特定的样本上，如果结果是对的，那么 0-1 损失是 0；否则是 1。但是对于密度估计这类任务而言，度量准确率，错误率或者其他类型的 0-1 损失是没有意义的。反之，我们必须使用不同的性能度量，使模型对每个样本都输出一个连续数值的得分。最常用的方法是输出模型在一些样本上概率对数的平均值。

通常，我们会更加关注机器学习算法在未观测数据上的性能如何，因为这将决定其在现实生活中的性能如何。因此，我们使用测试数据来评估系统性能，同训练机器学习系统的数据分开。

性能度量的选择或许看上去简单且客观，但是选择一个与系统理想表现对应的性能度量通常是很难的。

在某些情况下，这是因为很难决定应该度量什么。例如，在执行转录任务时，我们是应该度量系统转录整个序列的准确率，还是应该用一个更细粒度的指标，对序列中部分元素正确的以正面评价？在执行回归任务时，我们应该更多地惩罚频繁犯一些中等错误的系统，还是较少犯错但是犯很大错误的系统？这些设计的选择取决

于应用。

还有一些情况，我们知道应该度量哪些数值，但是度量它们不太现实。这种情况经常出现在密度估计中。很多最好的概率模型只能隐式地表示概率分布。在许多这类模型中，计算空间中特定点的概率是不可行的。在这些情况下，我们必须设计一个仍然对应于设计对象的替代标准，或者设计一个理想标准的良好近似。

### 5.1.3 经验 $E$

根据学习过程中的不同经验，机器学习算法可以大致分类为无监督 (unsupervised) 和监督 (supervised)。

本书中的大部分学习算法可以理解成在整个数据集 (dataset) 上获取经验。数据集是指很多样本组成的集合，如第5.1.1节的定义。有时我们也将样本称为数据点 (data point)。

Iris (鸢尾花卉) 数据集(Fisher, 1936) 是统计学家和机器学习研究者使用很久的数据集。它是 150 个鸢尾花卉植物不同部分测量结果的集合。每个单独的植物对应一个样本。每个样本的特征是该植物不同部分的测量结果：萼片长度，萼片宽度，花瓣长度和花瓣宽度。这个数据集记录了每个植物属于什么品种，其中共有三个不同的品种。

无监督学习算法 (unsupervised learning algorithm) 训练含有很多特征的数据集，然后学习出这个数据集上有用的结构性质。在深度学习中，我们通常要学习生成数据集的整个概率分布，显式地，比如密度估计，或是隐式地，比如合成或去噪。还有一些其他类型的无监督学习任务，例如聚类，将数据集分成相似样本的集合。

监督学习算法 (supervised learning algorithm) 训练含有很多特征的数据集，不过数据集中的样本都有一个标签 (label) 或目标 (target)。例如，Iris数据集注明了每个鸢尾花卉样本属于什么品种。监督学习算法通过研究 Iris数据集，学习如何根据测量结果将样本划分到三个不同品种。

大致说来，无监督学习涉及到观察随机向量  $\mathbf{x}$  的好几个样本，试图隐式或显式地学习出概率分布  $p(\mathbf{x})$ ，或者是该分布一些有意思的性质；而监督学习包含观察随机向量  $\mathbf{x}$  及其相关联的值或向量  $\mathbf{y}$ ，然后从  $\mathbf{x}$  预测  $\mathbf{y}$ ，通常是估计  $p(\mathbf{y} | \mathbf{x})$ 。术语监督 (supervised) 源自这样一个视角，教员或者老师提供目标  $\mathbf{y}$  给机器学习系统，指导其应该做什么。在无监督学习中，没有教员或者老师，算法必须学会在没有指导

的情况下让数据有意义。

无监督学习和监督学习不是严格定义的术语。它们之间界线通常是模糊的。很多机器学习技术可以用于这两个任务。例如，概率的链式法则表明对于向量  $\mathbf{x} \in \mathbb{R}^n$ ，联合分布可以分解成

$$p(\mathbf{x}) = \prod_{i=1}^n p(\mathbf{x}_i \mid \mathbf{x}_1, \dots, \mathbf{x}_{i-1}). \quad (5.1)$$

该分解意味着我们可以将其拆分成  $n$  个监督学习，来解决表面上的无监督学习  $p(\mathbf{x})$ 。另外，我们求解监督学习问题  $p(y \mid \mathbf{x})$  时，也可以使用传统的无监督学习策略学习联合分布  $p(\mathbf{x}, y)$ ，然后推断

$$p(y \mid \mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')}. \quad (5.2)$$

尽管无监督学习和监督学习并非完全没有交集的正式概念，它们确实有助于粗略分类我们研究机器学习算法时遇到的问题。传统地，人们将回归，分类，或者结构化输出问题称为监督学习。支持其他任务的密度估计通常被称为无监督学习。

学习范式的其他变种也是有可能的。例如，半监督学习中，一些样本有监督目标，但其他的没有。在多实例学习中，样本的整个集合被标记为含有或者不含有该类的样本，但是集合中单独的样本是没有标记的。参考Kotzias *et al.* (2015) 了解最近深度模型进行多实例学习的示例。

有些机器学习算法并不是训练于一个固定的数据集上。例如，强化学习 (reinforcement learning) 算法会和环境进行交互，所以学习系统和它的训练过程会有反馈回路。这类算法超出了本书的范畴。请参考Sutton and Barto (1998) 或Bertsekas and Tsitsiklis (1996) 了解强化学习相关知识，Mnih and Kavukcuoglu (2013) 介绍了强化学习方向的深度学习方法。

大部分机器学习算法简单地训练于一个数据集上。数据集可以用很多不同方式来表示。在所有的情况下，数据集都是样本的集合，而样本是特征的集合。

表示数据集的常用方法是设计矩阵 (design matrix)。设计矩阵的每一行包含一个不同的样本。每一列对应不同的特征。例如，Iris数据集包含 150 个样本，每个样本有 4 个特征。这意味着我们将该数据集表示成设计矩阵  $\mathbf{X} \in \mathbb{R}^{150 \times 4}$ ，其中  $\mathbf{X}_{i,1}$  表示第  $i$  个植物的萼片长度， $\mathbf{X}_{i,2}$  表示第  $i$  个植物的萼片宽度，等等。我们在本书中描述的大部分学习算法都是讲述它们是如何运行在设计矩阵数据集上的。

当然，将一个数据集表示成设计矩阵，必须是可以将每一个样本表示成向量，并且这些向量的大小相同。这一点并非永远可能。例如，你有不同宽度和高度的照片

的集合，那么不同的照片将会包含不同数量的像素。因此不是所有的照片都可以表示成相同长度的向量。第9.7节和第十章将会介绍如何处理这类异质问题的不同类型。在上述的这类情况下，我们不会将数据集表示成  $m$  行的矩阵，而是表示成  $m$  个元素的结合： $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}\}$ 。这种表示方式并非意味着样本向量  $\mathbf{x}^{(i)}$  和  $\mathbf{x}^{(j)}$  有相同的大小。

在监督学习中，样本包含一个标签或目标和一组特征。例如，我们希望使用学习算法从照片中识别物体。我们需要明确哪些物体会出现在每张照片中。我们或许会用数字编码表示，如 0 表示人，1 表示车，2 表示猫，等等。通常当工作在包含观测特征的设计矩阵  $\mathbf{X}$  的数据集时，我们也会提供一个标签向量  $\mathbf{y}$ ，其中  $y_i$  表示样本  $i$  的标签。

当然，有时标签可能不止一个数。例如，如果我们想要训练语音模型转录整个句子，那么每个句子样本的标签是一个单词序列。

正如监督学习和无监督学习没有正式的定义，数据集或者经验也没有严格的区分。这里介绍的结构涵盖了大多数情况，但始终有可能为新的应用设计出新的结构。

#### 5.1.4 实例：线性回归

我们将机器学习算法定义为，通过经验以提高计算机程序在某些任务上性能的算法。这个定义有点抽象。为了使这个定义更具体点，我们展示一个简单的机器学习实例：**线性回归** (linear regression)。当我们介绍更多有助于理解机器学习特性的概念时，我们会反复回顾这个实例。

顾名思义，线性回归解决回归问题。换言之，我们的目标是建立一个系统，将向量  $\mathbf{x} \in \mathbb{R}^n$  作为输入，预测标量  $y \in \mathbb{R}$  作为输出。线性回归的输出是其输入的线性函数。让  $\hat{y}$  表示模型预测  $y$  应该取的值。我们定义输出为

$$\hat{y} = \mathbf{w}^\top \mathbf{x}, \quad (5.3)$$

其中  $\mathbf{w} \in \mathbb{R}^n$  是参数 (parameter) 向量。

参数是控制系统行为的值。在这种情况下， $w_i$  是系数，会和特征  $x_i$  相乘之后全部相加起来。我们可以将  $\mathbf{w}$  看作是一组决定每个特征如何影响预测的权重 (weight)。如果特征  $x_i$  对应的权重  $w_i$  是正的，那么特征值增加，我们的预测值  $\hat{y}$  也会增加。如果特征  $x_i$  对应的权重  $w_i$  是负的，那么特征值减少，我们的预测值  $\hat{y}$  也会减少。

如果特征权重的大小很大，那么它对预测有很大的影响；如果特征权重的大小是零，那么它对预测没有影响。

因此，我们可以定义任务  $T$ ：通过输出  $\hat{y} = \mathbf{w}^\top \mathbf{x}$  从  $\mathbf{x}$  预测  $y$ 。接下来我们需要定义性能度量， $P$ 。

假设我们有  $m$  个输入样本组成的设计矩阵，我们不用它来训练模型，而是评估模型性能如何。我们也有每个样本对应的正确值  $y$  组成的回归目标向量。因为这个数据集只是用来评估性能，我们称之为测试集。我们将输入的设计矩阵记作  $\mathbf{X}^{(\text{test})}$ ，回归目标向量记作  $\mathbf{y}^{(\text{test})}$ 。

度量模型性能的一种方法是计算模型在测试集上的均方误差 (mean squared error)。如果  $\hat{\mathbf{y}}^{(\text{test})}$  表示模型在测试集上的预测值，那么均方误差表示为：

$$\text{MSE}_{\text{test}} = \frac{1}{m} \sum_i (\hat{\mathbf{y}}^{(\text{test})} - \mathbf{y}^{(\text{test})})_i^2. \quad (5.4)$$

直观上，当  $\hat{\mathbf{y}}^{(\text{test})} = \mathbf{y}^{(\text{test})}$  时，我们会发现误差降为 0。我们也可以表示为

$$\text{MSE}_{\text{test}} = \frac{1}{m} \left\| \hat{\mathbf{y}}^{(\text{test})} - \mathbf{y}^{(\text{test})} \right\|_2^2, \quad (5.5)$$

所以当预测值和目标值之间的欧几里得距离增加时，误差也会增加。

构建一个机器学习算法，我们需要设计一个算法，通过观察训练集  $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$  获得经验，减少  $\text{MSE}_{\text{test}}$  以改进权重  $\mathbf{w}$ 。一种直观方式（我们将在后续的第5.5.1节说明其合法性）是最小化训练集上的均方误差， $\text{MSE}_{\text{train}}$ 。

最小化  $\text{MSE}_{\text{train}}$ ，我们可以简单地求解其导数为  $\mathbf{0}$  的情况：

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0 \quad (5.6)$$

$$\Rightarrow \nabla_{\mathbf{w}} \frac{1}{m} \left\| \hat{\mathbf{y}}^{(\text{train})} - \mathbf{y}^{(\text{train})} \right\|_2^2 = 0 \quad (5.7)$$

$$\Rightarrow \frac{1}{m} \nabla_{\mathbf{w}} \left\| \mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})} \right\|_2^2 = 0 \quad (5.8)$$

$$\Rightarrow \nabla_{\mathbf{w}} \left( \mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})} \right)^\top \left( \mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})} \right) = 0 \quad (5.9)$$

$$\Rightarrow \nabla_{\mathbf{w}} \left( \mathbf{w}^\top \mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})} \mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})} + \mathbf{y}^{(\text{train})\top} \mathbf{y}^{(\text{train})} \right) = 0 \quad (5.10)$$

$$\Rightarrow 2\mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})} \mathbf{w} - 2\mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})} = 0 \quad (5.11)$$

$$\Rightarrow \mathbf{w} = \left( \mathbf{X}^{(\text{train})\top} \mathbf{X}^{(\text{train})} \right)^{-1} \mathbf{X}^{(\text{train})\top} \mathbf{y}^{(\text{train})} \quad (5.12)$$

通过式(5.12)给出解的系统方程被称为正规方程 (normal equation)。计算式(5.12)构成了一个简单的机器学习算法。参看图5.1，线性回归算法在使用中的示例。

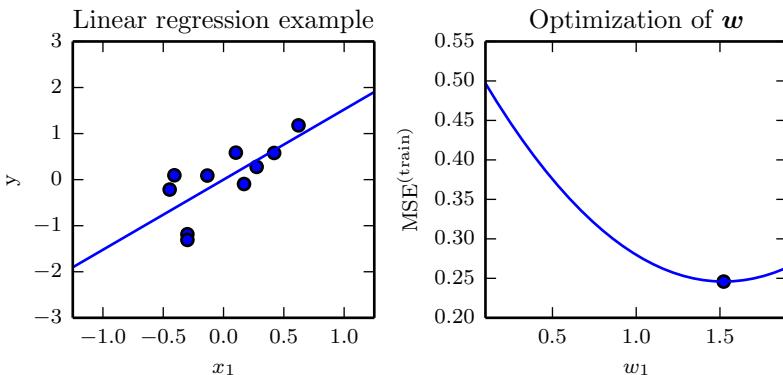


图 5.1: 一个线性回归问题，训练集包括了十个数据点，每个数据点包含了一个特征。因为只有一个特征，权重向量  $w$  也只有一个要学习的参数  $w_1$ 。(左) 我们可以观察到线性回归学习  $w_1$ ，从而使得直线  $y = w_1 x$  能够尽量接近穿过所有的训练点。(右) 标注的点表示由正规方程学习到的  $w_1$  的值，我们发现它可以最小化训练集上的均方误差。

值得注意的是，术语线性回归通常用来指稍微复杂一些，附加额外参数（截距项  $b$ ）的模型。在这个模型中，

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b, \quad (5.13)$$

因此从参数到预测的映射仍是一个线性函数，而从特征到预测的映射是一个仿射函数。如此扩展到仿射函数意味着模型预测的曲线仍然看起来像是一条直线，只是这条直线没必要经过原点。不通过添加偏置参数  $b$ ，我们仍然可以使用仅含权重的模型，但是  $\mathbf{x}$  需要增加一项永远为 1 的元素。对应于额外 1 的权重起到了偏置参数的作用。当我们在本书中提到仿射函数时，我们会经常使用术语“线性”。

截距项  $b$  通常被称为仿射变换的偏置 (bias) 参数。这个术语的命名源自该变换的输出在没有任何输入时会偏移  $b$ 。它和统计偏差中指代统计估计算法的某个量的期望估计偏离真实值的意思是不一样的。

线性回归当然是一个极其简单且有局限的学习算法，但是它提供了一个说明学习算法如何工作的例子。在接下来的小节中，我们将会介绍一些设计学习算法的基本原则，并说明如何使用这些原则来构建更复杂的学习算法。

## 5.2 容量、过拟合和欠拟合

机器学习的主要挑战是我们的算法必须能够在先前未观测的新输入上表现良好，而不只是在训练集上效果好。在先前未观测到的输入上表现良好的能力被称为泛化 (generalization)。

通常情况下，当我们训练机器学习模型时，我们可以访问训练集，在训练集上计算一些度量误差，被称为训练误差 (training error)，并且我们会降低训练误差。目前为止，我们讨论的是一个简单的优化问题。机器学习和优化不同的地方在于，我们也希望泛化误差 (generalization error)，也被称为测试误差 (test error)，很低。泛化误差被定义为新输入的误差期望。这里，期望取值自我们期望系统在现实中从输入分布中采样得到的不同可能值。

通常，我们度量模型在训练集中分出来的测试集 (test set) 样本上的性能，来评估机器学习模型的泛化误差。

在我们的线性回归实例中，我们通过最小化训练误差来训练模型，

$$\frac{1}{m^{(\text{train})}} \left\| \mathbf{X}^{(\text{train})} \mathbf{w} - \mathbf{y}^{(\text{train})} \right\|_2^2, \quad (5.14)$$

但是我们真正关注的是测试误差， $\frac{1}{m^{(\text{test})}} \left\| \mathbf{X}^{(\text{test})} \mathbf{w} - \mathbf{y}^{(\text{test})} \right\|_2^2$ 。

当我们只能观测到训练集时，我们如何才能影响测试集的性能呢？统计学习理论 (statistical learning theory) 提供了一些答案。如果训练集和测试集的数据是任意收集的，那么我们能够做的确实很有限。如果我们可以对训练集和测试集数据的收集方式有些假设，那么我们能够对算法做些改进。

训练集和测试集数据通过数据集上被称为数据生成过程 (data generating process) 的概率分布生成。通常，我们会做一系列假设，被统称为独立同分布假设 (i.i.d. assumption)。该假设是说，每个数据集中的样本都是彼此相互独立的 (independent)，并且训练集和测试集是同分布的 (identically distributed)，采样自相同的分布。这个假设使我们能够在单个样本上用概率分布描述数据生成过程。然后相同的分布可以用来生成每一个训练样本和每一个测试样本。我们将这个共享的潜在分布称为数据生成分布 (data generating distribution)，记作  $p_{\text{data}}$ 。这个概率框架和独立同分布假设允许我们数学地研究训练误差和测试误差之间的关系。

我们能观察到训练误差和测试误差之间的直接联系是，随机模型训练误差的期望和该模型测试误差的期望是一样的。假设我们有概率分布  $p(\mathbf{x}, y)$ ，从中重复采样

生成训练集和测试集。对于某个固定的  $w$ ，训练集误差的期望恰好和测试集误差的期望一样，这是因为这两个期望的计算都是用相同的数据集生成过程。这两种情况的唯一区别是数据集的名字不同。

当然，当我们使用机器学习算法时，我们不会提前固定参数，然后从数据集中采样。我们会在训练集上采样，然后挑选参数去降低训练集误差，然后再在测试集上采样。在这个过程中，测试误差期望会大于或等于训练误差期望。以下是决定机器学习算法效果是否好的因素：

1. 降低训练误差
2. 缩小训练误差和测试误差的差距

这两个因素对应机器学习的两个主要挑战：欠拟合 (underfitting) 和过拟合 (overfitting)。欠拟合发生于模型不能在训练集上获得足够低的误差。过拟合发生于训练误差和测试误差之间的差距太大。

通过调整模型的容量 (capacity)，我们可以控制模型是否偏向于过拟合或者欠拟合。通俗地，模型的容量是指其拟合各种函数的能力。容量低的模型可能很难拟合训练集。容量高的模型可能会过拟合，因为记住了不适用于测试集的训练集性质。

一种控制训练算法容量的方法是选择假设空间 (hypothesis space)，即能够选为解决方案的学习算法函数集。例如，线性回归函数将关于其输入的所有线性函数作为假设空间。广义线性回归的假设空间包括多项式函数，而非仅有线性函数。这样增加了模型的容量。

一次多项式提供了我们已经熟悉的线性回归模型，其预测如下：

$$\hat{y} = b + wx. \quad (5.15)$$

通过引入  $x^2$  作为线性回归模型的另一个特征，我们能够学习关于  $x$  的二次函数模型：

$$\hat{y} = b + w_1x + w_2x^2. \quad (5.16)$$

尽管该模型是输入的二次函数，但输出仍是参数的线性函数。因此我们仍然可以用正规方程得到模型的闭解。我们可以继续添加  $x$  的更高幂作为额外特征，例如下面的 9 次多项式：

$$\hat{y} = b + \sum_{i=1}^9 w_i x^i. \quad (5.17)$$

当机器学习算法的容量适合于所执行任务的复杂度和所提供数据的数量时，算法效果会最佳。容量不足的模型不能解决复杂任务。容量高的模型能够解决复杂的任务，但是当其容量高于任务时，有可能会过拟合。

图5.2展示了这个原理在使用中的情况。我们比较了线性，二次和9次预测器拟合二次真实函数的效果。线性函数无法刻画真实函数的曲率，所以欠拟合。9次函数能够表示正确的函数，但是因为训练参数比训练样本还多，所以它也能够表示无限多个刚好穿越训练样本点的很多其他函数。我们不太可能从这很多不同的解中选出一个泛化良好的。在这个问题中，二次模型非常符合任务的真实结构，因此它可以很好地泛化到新数据上。

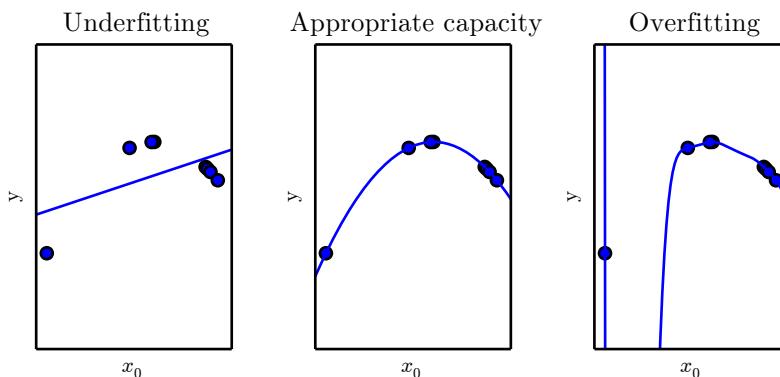


图 5.2: 我们用三个模型拟合了这个训练集的样本。训练数据是通过随机抽取  $x$  然后用二次函数确定性地生成  $y$  来合成的。(左) 用一个线性函数拟合数据会导致欠拟合——它无法捕捉数据中的曲率信息。(中) 用二次函数拟合数据在未观察到的点上泛化得很好。这并不会导致明显的欠拟合或者过拟合。(右) 一个 9 阶的多项式拟合数据会导致过拟合。在这里我们使用 Moore-Penrose 伪逆来解这个欠定的正规方程。得出的解能够精确的穿过所有的训练点，但不幸的是我们无法提取有效的结构信息。在两个数据点之间它有一个真实的函数所不包含的深谷。在数据的左侧，它也会急剧增长，在这一区域真实的函数却是下降的。

目前为止，我们探讨了通过改变输入特征的数目，和加入这些特征对应的参数，改变模型的容量。事实上，还有很多方法可以改变模型的容量。容量不仅取决于模型的选择。模型规定了调整参数降低训练对象时，学习算法可以从哪些函数族中选择函数。这被称为模型的表示容量 (representational capacity)。在很多情况下，从这些函数中挑选出最优函数是非常困难的优化问题。实际中，学习算法不会真的找到最优函数，而仅是找到一个可以降低训练误差很多的函数。额外的限制因素，比如

优化算法的不完美，意味着学习算法的有效容量 (effective capacity) 可能小于模型族的表示容量。

提高机器学习模型泛化的现代思想可以追溯到早在托勒密时期的哲学家的思想。许多早期的学者提出一个简约原则，现在被广泛称为奥卡姆剃刀 (Occam's razor) (c. 1287-1387)。该原则指出，在同样能够解释已知观测现象的假设中，我们应该挑选“最简单”的那一个。这个想法是在 20 世纪，由统计学习理论创始人提出来并精确化的 (Vapnik and Chervonenkis, 1971; Vapnik, 1982; Blumer *et al.*, 1989; Vapnik, 1995)。

统计学习理论提供了量化模型容量的不同方法。在这些中，最有名的是**Vapnik-Chervonenkis 维度** (Vapnik-Chervonenkis dimension, VC)。VC维度量二元分类器的容量。VC维定义为该分类器能够分类的训练样本的最大数目。假设存在  $m$  个不同  $x$  点的训练集，分类器可以任意地标记该  $m$  个不同的  $x$  点，VC维被定义为  $m$  的最大可能值。

量化模型的容量使得统计学习理论可以进行量化预测。统计学习理论中最重要的结论阐述了训练误差和泛化误差之间差异的上界随着模型容量增长而增长，但随着训练样本增多而下降 (Vapnik and Chervonenkis, 1971; Vapnik, 1982; Blumer *et al.*, 1989; Vapnik, 1995)。这些边界为机器学习算法可以有效解决问题提供了理论验证，但是它们很少应用于实际中的深度学习算法。一部分原因是边界太松，另一部分原因是很难确定深度学习算法的容量。确定深度学习模型容量的问题特别困难是由于有效容量受限于优化算法的能力。对于深度学习中的一般非凸优化问题，我们只有很少的理论分析。

我们必须记住虽然更简单的函数更可能泛化（训练误差和测试误差的差距小），但我们仍然需要选择一个充分复杂的假设以达到低的训练误差。通常，当模型容量上升时，训练误差会下降，直到其渐近最小可能误差（假设误差度量有最小值）。通常，泛化误差是一个关于模型容量的 U 形曲线函数。如图5.3所示。

考虑任意高容量的极端情况，我们介绍非参数 (non-parametric) 模型的概念。至此，我们只探讨过参数模型，例如线性回归。参数模型学习到的函数在观测新数据前，参数是有限且固定的向量。非参数模型没有这些限制。

有时，非参数模型仅是一些不能实际实现的理论抽象（比如搜索所有可能概率分布的算法）。然而，我们也可以设计一些实用的非参数模型，使它们的复杂度和训练集大小有关。这种算法的一个实例是最近邻回归 (nearest neighbor regression)。不

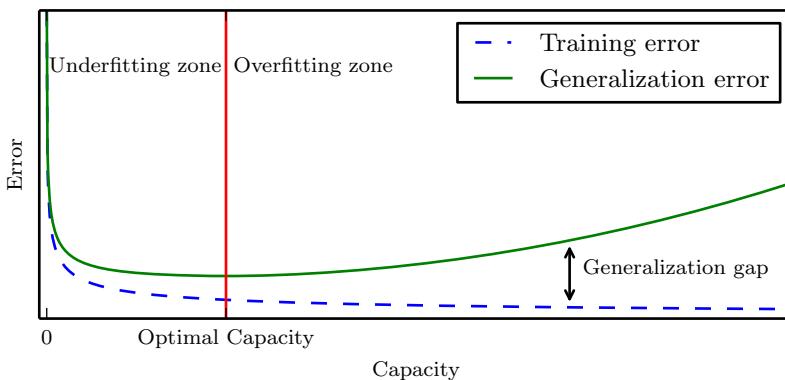


图 5.3: 容量和误差之间的典型关系。训练误差和测试误差表现得非常不同。在图的左端，训练误差和泛化误差都非常高。这是欠拟合期 (underfitting regime)。当我们增加容量时，训练误差减小，但是训练误差和泛化误差之间的间距却不断扩大。最终，这个间距的大小超过了训练误差的下降，我们进入到了过拟合期 (overfitting regime)，其中容量过大，超过了最佳容量 (optimal capacity)。

像线性回归有固定长度的向量作为权重，最近邻回归模型存储了训练集中所有的  $\mathbf{X}$  和  $\mathbf{y}$ 。当需要为测试点  $\mathbf{x}$  分类时，模型会查询训练集中离该点最近的点，并返回相关的回归目标。换言之， $\hat{y} = y_i$  其中  $i = \arg \min \| \mathbf{X}_{i,:} - \mathbf{x} \|^2_2$ 。该算法也可以扩展成  $L^2$  范数以外的距离度量，例如学习距离度量 (Goldberger et al., 2005)。如果该算法通过平均  $\mathbf{X}_{i,:}$  中所有最近的向量对应的  $y_i$  来打破平局，那么该算法会在任意回归数据集上达到最小可能的训练误差（如果存在两个相同的输入对应不同的输出，那么训练误差可能会大于零）。

最后，我们也可以将参数学习算法嵌入另一个依所需增加参数数目的算法来创建非参数学习算法。例如，我们可以想象一个算法，外层循环调整多项式的次数，内存循环通过线性回归学习模型。

理想模型假设我们能够预先知道生成数据的真实概率分布。然而这样的模型仍然会在很多问题上发生一些错误，因为分布中仍然会有一些噪扰。在监督学习中，从  $\mathbf{x}$  到  $\mathbf{y}$  的映射可能内在是随机的，或者  $\mathbf{y}$  可能是包括  $\mathbf{x}$  在内还有其他变量的确定性函数。从预先知道的真实分布  $p(\mathbf{x}, \mathbf{y})$  预测而出现的误差被称为贝叶斯误差 (Bayes error)。

训练误差和泛化误差会随训练集的大小发生变化。泛化误差的期望不会随着训练样本数目的增加而增加。对于非参数模型而言，更多的数据会得到更好的泛化能

力，直到达到最佳可能的泛化误差。任何模型容量小于最优容量的固定参数模型会渐近到大于贝叶斯误差的误差值。如图5.4所示。值得注意的是，具有最优容量的模型仍然有可能在训练误差和泛化误差之间存在很大的差距。在这种情况下，我们可以通过收集更多的训练样本来缩小差距。

### 5.2.1 没有免费午餐定理

学习理论表明机器学习算法能够从有限个训练集样本中很好地泛化。这似乎违背一些基本的逻辑原则。归纳推理，或是从一组有限的样本中推断一般的规则，在逻辑上不是很有效。逻辑地推断一个规则去描述集合中的元素，我们必须具有集合中每个元素的信息。

在一定程度上，机器学习仅通过概率法则就可以避免这个问题，而无需使用纯逻辑推理整个确定性法则。机器学习保证找到一个关注的大多数样本可能正确的规则。

不幸的是，即使这样也不能解决整个问题。机器学习的没有免费午餐定理 (no free lunch theorem) 表明，在所有可能的数据生成分布上平均，每一个分类算法在未事先观测的点上都有相同的错误率。换言之，在某种意义上，没有一个机器学习算法总是比其他的要好。我们能够设想的最先进的算法和简单地将每一个点归为同一类的简单算法有着相同的平均性能（在所有可能的任务上）。

幸运的是，这些结论仅在我们考虑所有可能的数据生成分布时才成立。在现实世界的应用中，如果我们对遇到的概率分布进行假设的话，那么我们可以设计在这些分布上效果良好的学习算法。

这意味着机器学习研究的目标不是找一个通用学习算法或是绝对最好的学习算法。反之，我们的目标是理解什么样的分布和人工智能获取经验的“真实世界”相关，什么样的学习算法在我们关注的数据生成分布上效果最好。

### 5.2.2 正则化

没有免费午餐定理暗示我们必须在特定任务上设计性能良好的机器学习算法。我们建立一组学习算法的偏好来达到这个要求。当这些偏好和我们希望算法解决的学习问题相吻合时，性能会更好。

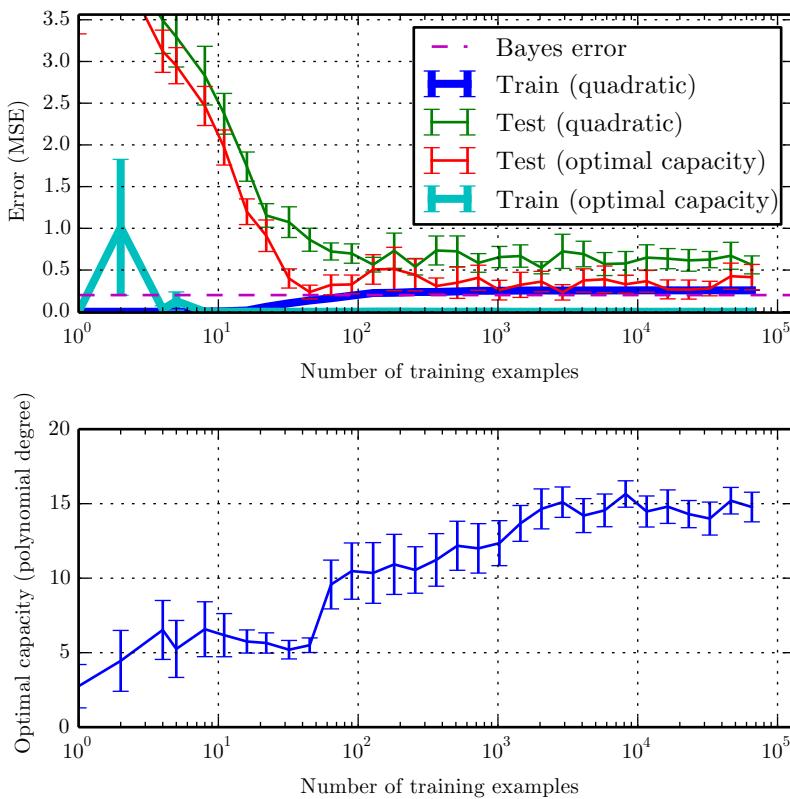


图 5.4: 训练集大小对训练误差, 测试误差以及最佳容量的影响。通过给一个 5 阶多项式添加适当大小的噪声, 我们构造了一个合成的回归问题, 生成单个测试集, 然后生成一些不同尺寸的训练集。为了描述%95 置信区间的误差条, 对于每一个尺寸, 我们生成了 40 个不同的训练集。(上) 两个不同的模型上训练集和测试集的 MSE, 一个二次模型, 另一个模型的阶数通过最小化测试误差来选择。两个模型都是用闭式解来拟合。对于二次模型来说, 当训练集增加时训练误差也随之增大。这是由于越大的数据集越难以拟合。同时, 测试误差随之减小, 因为关于训练数据的不正确的假设越来越少。二次模型的容量并不足以解决这个问题, 所以它的测试误差趋近于一个较高的值。最佳容量点处的测试误差趋近于贝叶斯误差。训练误差可以低于贝叶斯误差, 因为训练算法有能力记住训练集中特定的样本。当训练集趋向于无穷大时, 任何固定容量的模型(在这里指的是二次模型)的训练误差都至少增至贝叶斯误差。(下) 当训练集大小增大时, 最佳容量(在这里是用最优多项式回归器的阶数衡量的)也会随之增大。最佳容量在达到足够捕捉模型复杂度之后就不再增长了。

至此，我们具体讨论修改学习算法的方法只有，通过增加或减少学习算法可选假设空间的函数来增加或减少模型的容量。我们列举的一个具体实例是线性回归增加或减少多项式的次数。目前为止讨论的观点都是过度简化的。

算法的效果不仅受影响于假设空间的函数数量，也取决于这些函数的具体形式。我们已经讨论的学习算法，线性回归，具有包含其输入的线性函数集的假设空间。对于输入和输出确实接近线性相关的问题，这些线性函数是很有用的。对于完全非线性的问题它们不太有效。例如，我们用线性回归，从  $x$  预测  $\sin(x)$ ，效果不会好。我们控制算法的性能，可以通过控制允许采样的函数种类的方式，也可以通过控制这些函数的数量的方式。

在假设空间中，相比于某一个学习算法，我们可能更偏好另一个学习算法。这意味着两个函数都是符合条件的，但是我们更偏好其中一个。只有非偏好函数比偏好函数在训练数据集上效果明显好很多时，我们才会考虑非偏好函数。

例如，我们可以加入权重衰减 (weight decay) 来修改线性回归的训练标准。带权重衰减的线性回归最小化，训练集上的均方误差和正则项的和  $J(\mathbf{w})$ ，偏好于平方  $L^2$  范数较小的权重。具体如下：

$$J(\mathbf{w}) = \text{MSE}_{\text{train}} + \lambda \mathbf{w}^\top \mathbf{w}, \quad (5.18)$$

其中  $\lambda$  是提前挑选的值，控制我们偏好小范数权重的程度。当  $\lambda = 0$ ，我们没有任何偏好。越大的  $\lambda$  偏好范数越小的权重。最小化  $J(\mathbf{w})$  可以看作是拟合训练数据和偏好小权重范数之间的权衡。这会使得解决方案的斜率较小，或是将权重放在较少的特征上。我们可以训练具有不同  $\lambda$  值的高次多项式，来举例说明如何通过权重衰减控制模型欠拟合或过拟合的趋势。如图5.5所示。

更一般地，正则化一个学习函数  $f(\mathbf{x}; \boldsymbol{\theta})$  的模型，我们可以给代价函数添加被称为正则化项 (regularizer) 的惩罚。在权重衰减的例子中，正则化项是  $\Omega(\mathbf{w}) = \mathbf{w}^\top \mathbf{w}$ 。在第七章，我们将看到很多其他可能的正则化项。

表示对函数的偏好是比增减假设空间的成员函数更一般的去控制模型容量的方法。我们可以将去掉假设空间中的某个函数看作是对不赞成这个函数的无限偏好。

在我们权重衰减的示例中，通过在最小化的目标中额外增加一项，我们明确地表示了偏好权重较小的线性函数。有很多其他方法隐式地或显式地表示对不同解决方法的偏好。总而言之，这些不同的方法都被称为正则化 (regularization)。正则化是指我们对学习算法所做的降低泛化误差而非训练误差的修改。正则化是机器学习领

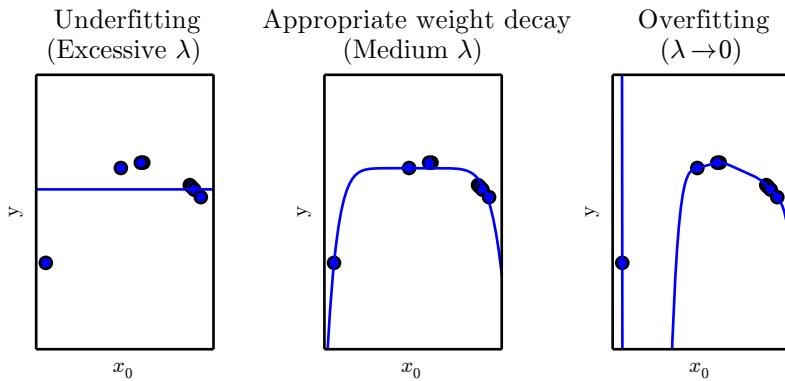


图 5.5: 我们使用高阶多项式回归模型来拟合图5.2中训练样本。真实函数是二次的，但是在这里我们只使用 9 阶多项式。我们通过改变权重衰减的量来避免高阶模型的过拟合问题。(左) 当  $\lambda$  非常大时，我们可以强迫模型学习到了一个没有斜率的函数。由于它只能表示一个常数函数，所以会导致欠拟合。(中) 取一个适当的  $\lambda$  时，学习算法能够用一个正常的形状来恢复曲率。即使模型能够用更复杂的形状来表示函数，权重衰减鼓励用一个带有更小参数的更简单的模型来描述它。(右) 当权重衰减趋近于 0 (即，使用Moore-Penrose 伪逆来解这个带有最小正则化的欠定问题) 时，这个 9 阶多项式会导致严重的过拟合，这和我们在图5.2中看到的一样。

域的中心问题之一，能够和其重要性媲美的只有优化。

没有免费午餐定理已经清楚阐述了没有最优的学习算法，特别地，没有最优的正则化形式。反之，我们必须挑选一个非常适合于我们要解决任务的正则形式。深度学习中普遍的，特别是本书的，理念是大量任务（例如所有人类能做的智能任务）也许都可以使用非常通用的正则化项来有效解决。

## 5.3 超参数和验证集

大多数机器学习算法都有设置超参数，可以用来控制算法行为。超参数的值不是通过学习算法本身学习出来的（尽管我们可以设计一个嵌套的学习过程，一个学习算法为另一个学习算法学出最优超参数）。

在图5.2所示的多项式回归实例中，有一个超参数：多项式的次数，作为容量超参数。控制权重衰减程度的  $\lambda$  是另一个超参数。

有时一个情景被设为学习算法不用学习的超参数，是因为它太难优化了。更多

的情况是，该设定必须是超参数，因为它不适合在训练集上学习。这适用于控制模型容量的所有超参数。如果在训练集上学习超参数，这些超参数总是趋向于最大可能的模型容量，导致过拟合（参考图5.3）。例如，相较低次多项式和正的权重衰减设定，更高次的多项式和权重衰减参数设定  $\lambda = 0$  总能在训练集上更好地拟合。

为了解决这个问题，我们需要训练算法观测不到的验证集（validation set）样本。

早先我们讨论过和训练数据相同分布的样本组成的测试集可以用来估计学习过程完成之后的学习器的泛化误差。其重点在于测试样本不能以任何形式参与到模型的选择，包括设定超参数。基于这个原因，测试集中的样本不能用于验证集。因此，我们总是从训练数据中构建验证集。特别地，我们将训练数据分成两个不相交的子集。其中一个用于学习参数。另一个作为验证集，用于估计训练中或训练后的泛化误差，更新超参数。用于学习参数的数据子集通常仍被称为训练集，尽管这会和整个训练过程用到的更大的数据集相混。用于挑选超参数的数据子集被称为验证集。通常，80% 的训练数据用于训练，20% 用于验证。由于验证集是用来“训练”超参数的，尽管验证集的误差通常会比训练集误差小，验证集会低估泛化误差。所有超参数优化完成之后，泛化误差可能会通过测试集来估计。

在实际中，当相同的测试集已在很多年中重复地用于评估不同算法的性能，并且考虑学术界在该测试集上的各种尝试，我们最后可能也会对测试集有着乐观的估计。基准会因之变得陈旧，而不能反映系统的真实性能。值得庆幸的是，学术界往往会移到新的（通常会更具大更具挑战性的）基准数据集上。

### 5.3.1 交叉验证

将数据集分成固定的训练集和固定的测试集后，若测试集的误差很小，这将是有问题的。一个小规模的测试集意味着平均测试误差估计的统计不确定性，使得很难判断算法 A 是否比算法 B 在给定的任务上做得更好。

当数据集有十万计或者更多的样本时，这不会是一个严重的问题。当数据集太小时，也有替代方法允许我们使用所有的样本估计平均测试误差，代价是增加了计算量。这些过程是基于在原始数据上随机采样或分离出的不同数据集上重复训练和测试的想法。最常见的是  $k$ -折交叉验证过程，如算法5.1所示，将数据集分成  $k$  个不重合的子集。测试误差可以估计为  $k$  次计算后的平均测试误差。在第  $i$  次测试时，数据的第  $i$  个子集用于测试集，其他的数据用于训练集。带来的一个问题是不存在平均误差方差的无偏估计（Bengio and Grandvalet, 2004），但是我们通常会使用近

似来解决。

## 5.4 估计、偏差和方差

统计领域为我们提供了很多工具用于实现机器学习目标，不仅可以解决训练集上的任务，还可以泛化。基本的概念，例如参数估计，偏差和方差，对于形式化刻画泛化，欠拟合和过拟合都非常有帮助。

### 5.4.1 点估计

点估计试图为一些感兴趣的量提供单个“最优”预测。一般地，感兴趣的量可以是单个参数，或是某些参数模型中的一个向量参数，例如第5.1.4节线性回归中的权重，但是也有可能是整个函数。

为了区分参数估计和真实值，我们习惯表示参数  $\theta$  的点估计为  $\hat{\theta}$ 。

让  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  是  $m$  个独立同分布 (i.i.d.) 的数据点。点估计 (point estimator) 或统计量 (statistics) 是这些数据的任意函数：

$$\hat{\theta}_m = g(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}). \quad (5.19)$$

这个定义不要求  $g$  返回一个接近真实值  $\theta$  的值，或者  $g$  的值域恰好是  $\theta$  的允许取值范围。点估计的定义非常宽泛，给了估计量的设计者极大的灵活性。虽然几乎所有的函数都可以称为估计量，但是一个好的估计量的输出会接近生成训练数据的真实参数  $\theta$ 。

现在，我们采取频率派在统计上的观点。换言之，我们假设真实参数  $\theta$  是固定但未知的，而点估计  $\hat{\theta}$  是数据的函数。由于数据是随机过程采样出来的，数据的任何函数都是随机的。因此  $\hat{\theta}$  是一个随机变量。

点估计也可以指输入和目标变量之间关系的估计。我们将这类点估计称为函数估计。

**函数估计** 有时我们会关注函数估计（或函数近似）。这时我们试图从输入向量  $\mathbf{x}$  预测变量  $y$ 。我们假设有一个函数  $f(\mathbf{x})$  表示  $y$  和  $\mathbf{x}$  之间的近似关系。例如，我们可能假设  $y = f(\mathbf{x}) + \epsilon$ ，其中  $\epsilon$  是  $y$  中未能从  $\mathbf{x}$  预测的一部分。在函数估计中，我们感兴趣的是用模型估计去近似  $f$ ，或者估计  $\hat{f}$ 。函数估计和估计参数  $\theta$  是一样的；

**算法 5.1**  $k$ -折交叉验证算法。当给定数据集  $\mathbb{D}$  对于简单的训练/测试或训练/验证分割而言太小难以产生泛化误差的准确估计时（因为在小的测试集上， $L$  可能具有过高的方差）， $k$ -折交叉验证算法可以用于估计学习算法  $A$  的泛化误差。数据集  $\mathbb{D}$  包含的元素是抽象的样本  $\mathbf{z}^{(i)}$ （对于第  $i$  个样本），在监督学习的情况代表（输入，目标）对  $\mathbf{z}^{(i)} = (\mathbf{x}^{(i)}, y^{(i)})$ ，或者无监督学习的情况下仅用于输入  $\mathbf{z}^{(i)} = \mathbf{x}^{(i)}$ 。该算法返回  $\mathbb{D}$  中每个示例的误差向量  $\mathbf{e}$ ，其均值是估计的泛化误差。单个样本上的误差可用于计算平均值周围的置信区间（式(5.47)）。虽然这些置信区间在使用交叉验证之后不能很好地证明，但是通常的做法是只有当算法  $A$  误差的置信区间低于并且不与算法  $B$  的置信区间相交时，我们采声明算法  $A$  比算法  $B$  更好。

**Define** KFoldXV( $\mathbb{D}, A, L, k$ ):

**Require:**  $\mathbb{D}$  为给定数据集，其中元素为  $\mathbf{z}^{(i)}$

**Require:**  $A$  为学习算法，可视为一个函数（使用数据集作为输入，输出一个学好的函数）

**Require:**  $L$  为损失函数，可视为来自学好的函数  $f$ ，将样本  $\mathbf{z}^{(i)} \in \mathbb{D}$  映射到标量  $\in \mathbb{R}$  的函数

**Require:**  $k$  为折数

将  $\mathbb{D}$  分为  $k$  个互斥子集  $\mathbb{D}_i$ ，它们的并为  $\mathbb{D}$

**for**  $i$  from 1 to  $k$  **do**

$f_i = A(\mathbb{D} \setminus \mathbb{D}_i)$

**for**  $\mathbf{z}^{(j)}$  in  $\mathbb{D}_i$  **do**

$e_j = L(f_i, \mathbf{z}^{(j)})$

**end for**

**end for**

**Return**  $\mathbf{e}$

函数估计  $\hat{f}$  是函数空间中的一个点估计。线性回归实例（第5.1.4节中讨论的）和多项式回归实例（第5.2节中讨论的）都既可以解释为估计参数  $w$ ，又可以解释为估计从  $x$  到  $y$  的函数映射  $\hat{f}$ 。

现在我们回顾点估计最常研究的性质，并探讨这些性质说明了估计的什么性质。

## 5.4.2 偏差

估计的偏差被定义为：

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta, \quad (5.20)$$

其中期望作用在所有数据（看作是从随机变量采样得到的）上， $\theta$  是用于定义数据生成分布的  $\theta$  的真实值。如果  $\text{bias}(\hat{\theta}_m) = 0$ ，那么估计量  $\hat{\theta}_m$  被称为是无偏 (unbiased)，这意味着  $\mathbb{E}(\hat{\theta}_m) = \theta$ 。如果  $\lim_{m \rightarrow \infty} \text{bias}(\hat{\theta}_m) = 0$ ，那么估计量  $\hat{\theta}_m$  称为是渐近无偏 (asymptotically unbiased)，这意味着  $\lim_{m \rightarrow \infty} \mathbb{E}(\hat{\theta}_m) = \theta$ 。

**实例：伯努利分布** 考虑一组服从均值为  $\theta$  的伯努利分布的独立同分布采样  $\{x^{(1)}, \dots, x^{(m)}\}$ ：

$$P(x^{(i)}; \theta) = \theta^{x^{(i)}} (1 - \theta)^{(1-x^{(i)})}. \quad (5.21)$$

这个分布中参数  $\theta$  的常用估计量是训练样本的均值：

$$\hat{\theta}_m = \frac{1}{m} \sum_{i=1}^m x^{(i)}. \quad (5.22)$$

判断这个估计量是否有偏，我们将式(5.22)代入式(5.20)：

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}[\hat{\theta}_m] - \theta \quad (5.23)$$

$$= \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m x^{(i)}\right] - \theta \quad (5.24)$$

$$= \frac{1}{m} \sum_{i=1}^m \mathbb{E}[x^{(i)}] - \theta \quad (5.25)$$

$$= \frac{1}{m} \sum_{i=1}^m \sum_{x^{(i)}=0}^1 \left( x^{(i)} \theta^{x^{(i)}} (1 - \theta)^{(1-x^{(i)})} \right) - \theta \quad (5.26)$$

$$= \frac{1}{m} \sum_{i=1}^m (\theta) - \theta \quad (5.27)$$

$$= \theta - \theta = 0 \quad (5.28)$$

因为  $\text{bias}(\hat{\theta}) = 0$ ，我们称估计  $\hat{\theta}$  是无偏的。

**实例：均值的高斯分布估计** 现在，考虑一组独立同分布的样本  $\{x^{(1)}, \dots, x^{(m)}\}$  服从高斯分布  $p(x^{(i)}) = \mathcal{N}(x^{(i)}; \mu, \sigma^2)$ ，其中  $i \in \{1, \dots, m\}$ 。回顾高斯概率密度函数如下：

$$p(x^{(i)}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)} - \mu)^2}{\sigma^2}\right). \quad (5.29)$$

高斯均值参数的常用估计量被称为**样本均值** (sample mean)：

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad (5.30)$$

判断样本均值是否有偏，我们再次计算它的期望：

$$\text{bias}(\hat{\mu}_m) = \mathbb{E}[\hat{\mu}_m] - \mu \quad (5.31)$$

$$= \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m x^{(i)}\right] - \mu \quad (5.32)$$

$$= \left(\frac{1}{m} \sum_{i=1}^m \mathbb{E}[x^{(i)}]\right) - \mu \quad (5.33)$$

$$= \left(\frac{1}{m} \sum_{i=1}^m \mu\right) - \mu \quad (5.34)$$

$$= \mu - \mu = 0 \quad (5.35)$$

因此我们发现样本均值是高斯均值参数的无偏估计量。

**实例：高斯分布方差估计** 本例中，我们比较高斯分布方差参数  $\sigma^2$  的两个不同估计。我们探讨是否有一个是有偏的。

我们考虑的第一个方差估计被称为**样本方差** (sample variance)：

$$\hat{\sigma}_m^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2, \quad (5.36)$$

其中  $\hat{\mu}_m$  是样本均值。更形式地，我们感兴趣计算

$$\text{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2. \quad (5.37)$$

我们首先估计项  $\mathbb{E}[\hat{\sigma}_m^2]$ :

$$\mathbb{E}[\hat{\sigma}_m^2] = \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2\right] \quad (5.38)$$

$$= \frac{m-1}{m} \sigma^2 \quad (5.39)$$

回到式(5.37)，我们可以得出  $\hat{\sigma}_m^2$  的偏差是  $-\sigma^2/m$ 。因此样本方差是有偏估计。

**无偏样本方差 (unbiased sample variance) 估计**

$$\tilde{\sigma}_m^2 = \frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2 \quad (5.40)$$

提供了另一种可选方法。正如名字所言，这个估计是无偏的。换言之，我们会发现  $\mathbb{E}[\tilde{\sigma}_m^2] = \sigma^2$ :

$$\mathbb{E}[\tilde{\sigma}_m^2] = \mathbb{E}\left[\frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu}_m)^2\right] \quad (5.41)$$

$$= \frac{m}{m-1} \mathbb{E}[\hat{\sigma}_m^2] \quad (5.42)$$

$$= \frac{m}{m-1} \left(\frac{m-1}{m} \sigma^2\right) \quad (5.43)$$

$$= \sigma^2. \quad (5.44)$$

我们有两个估计量：一个是有偏的，另一个是无偏的。尽管无偏估计显然是可取的，但它并不总是“最好”的估计。我们将看到，经常会使用其他具有重要性质的有偏估计。

### 5.4.3 方差和标准误差

我们有时会考虑估计量的另一个性质，数据样本函数的变化程度。正如我们可以计算估计量的期望来决定它的偏差，我们也可以计算它的方差。估计量的方差 (variance) 就是一个方差

$$\text{Var}(\hat{\theta}) \quad (5.45)$$

其中随机变量是训练集。另外，方差的平方根被称为标准误差 (standard error)，记作  $\text{SE}(\hat{\theta})$ 。

估计量的方差或标准误差告诉我们，当独立地从潜在的数据生成过程中重采样数据集时，如何期望估计的变化。正如我们希望估计的偏差较小，我们也希望其方差较小。

当我们使用有限的样本计算任何统计量时，真实参数的估计都是不确定的，在这个意义上，从相同的分布得到其他样本时，它们的统计量会不一样。任何方差估计量的期望程度是我们想量化的误差的来源。

均值的标准误差被记作

$$\text{SE}(\hat{\mu}_m) = \sqrt{\text{Var}\left[\frac{1}{m} \sum_{i=1}^m x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}, \quad (5.46)$$

其中  $\sigma^2$  是样本  $x^{(i)}$  的真实方差。标准误差通常被记作  $\sigma$ 。不幸的是，样本方差的平方根和方差无偏估计的平方根都不是标准差的无偏估计。这两种计算方法都倾向于低估真实的标准差，但仍用于实际中。相较而言，方差无偏估计的平方根较少被低估。对于较大的  $m$ ，这种近似非常合理。

均值的标准误差在机器学习实验中非常有用。我们通常用测试集样本的误差均值来估计泛化误差。测试集中样本的数量决定了这个估计的精确度。中心极限定理告诉我们均值会接近一个高斯分布，我们可以用标准误差计算出真实期望落在选定区间的概率。例如，以均值  $\hat{\mu}_m$  为中心的 95% 置信区间是

$$(\hat{\mu}_m - 1.96\text{SE}(\hat{\mu}_m), \hat{\mu}_m + 1.96\text{SE}(\hat{\mu}_m)), \quad (5.47)$$

以上区间是基于均值  $\hat{\mu}_m$  和方差  $\text{SE}(\hat{\mu}_m)^2$  的高斯分布。在机器学习实验中，我们通常说算法 A 比算法 B 好，是指算法 A 的误差的 95% 置信区间的上界小于算法 B 的误差的 95% 置信区间的下界。

**实例：伯努利分布** 我们再次考虑从伯努利分布（回顾  $P(x^{(i)}; \theta) = \theta^{x^{(i)}}(1 - \theta)^{1-x^{(i)}}$ ）中独立同分布采样出来的一组样本  $\{x^{(1)}, \dots, x^{(m)}\}$ 。这次我们关注估计

$\hat{\theta}_m = \frac{1}{m} \sum_{i=1}^m x^{(i)}$  的方差：

$$\text{Var}(\hat{\theta}_m) = \text{Var}\left(\frac{1}{m} \sum_{i=1}^m x^{(i)}\right) \quad (5.48)$$

$$= \frac{1}{m^2} \sum_{i=1}^m \text{Var}(x^{(i)}) \quad (5.49)$$

$$= \frac{1}{m^2} \sum_{i=1}^m \theta(1 - \theta) \quad (5.50)$$

$$= \frac{1}{m^2} m\theta(1 - \theta) \quad (5.51)$$

$$= \frac{1}{m} \theta(1 - \theta) \quad (5.52)$$

估计量方差的下降速率是关于数据集样本数目  $m$  的函数。这是常见估计量的普遍性质，在探讨一致性（参考第5.4.5节）时，我们会继续讨论。

#### 5.4.4 权衡偏值和方差以最小化均方误差

偏差和方差度量着估计量的两个不同误差来源。偏差度量着离真实函数或参数的误差期望。而方差度量着数据上任意特定采样可能导致的估计期望的偏差。

当可以选择一个偏差更大的估计和一个方差更大的估计时，会发生什么呢？我们该如何选择？例如，想象我们希望近似图5.2中的函数，我们只可以选择一个偏差较大的估计或一个方差较大的估计，我们该如何选择呢？

判断这种权衡最常用的方法是交叉验证。经验上，交叉验证在许多真实世界的任务中都非常成功。另外，我们也可以比较这些估计的均方误差 (mean squared error, MSE)：

$$\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2] \quad (5.53)$$

$$= \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m) \quad (5.54)$$

MSE度量着估计和真实参数  $\theta$  之间平方误差的总体期望偏差。如式(5.54)所示，MSE估计包含了偏差和方差。理想的估计量具有较小的MSE或是在检查中会稍微约束它们的偏差和方差。

偏差和方差的关系和机器学习容量，欠拟合和过拟合的概念紧密相联。用MSE度量泛化误差（偏差和方差对于泛化误差都是有意义的）时，增加容量会增加方差，降

低偏差。如图5.6所示，我们再次在关于容量的函数中，看到泛化误差的U形曲线。

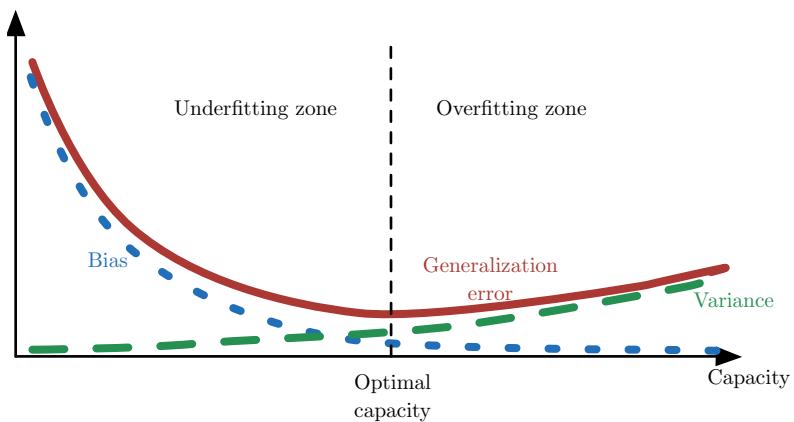


图 5.6: 当容量增大 ( $x$  轴) 时, 偏差 (用点表示) 随之减小, 而方差 (虚线) 随之增大, 使得泛化误差 (加粗曲线) 产生了另一种 U 形。如果我们沿着轴改变容量, 会发现最佳容量, 当容量小于最佳容量会呈现欠拟合, 大于时导致过拟合。这种关系与第5.2节以及图5.3中讨论的容量, 欠拟合和过拟合之间的关系类似。

#### 5.4.5 一致性

目前我们已经探讨了, 固定大小训练集下不同估计量的性质。通常, 我们也会关注训练数据增多后估计量的效果。特别地, 我们希望当数据集中数据点的数量  $m$  增加时, 点估计会收敛到对应参数的真实值。更形式地, 我们想要

$$\text{plim}_{m \rightarrow \infty} \hat{\theta}_m = \theta. \quad (5.55)$$

符号  $\text{plim}$  表示依概率收敛, 即对于任意的  $\epsilon > 0$ , 当  $m \rightarrow \infty$  时, 有  $P(|\hat{\theta}_m - \theta| > \epsilon) \rightarrow 0$ 。式(5.55)表示的条件被称为**一致性** (consistency)。有时它是指弱一致性, 强一致性是指几乎必然 (almost sure) 从  $\hat{\theta}$  收敛到  $\theta$ 。几乎必然收敛 (almost sure convergence) 是指当  $p(\lim_{m \rightarrow \infty} \mathbf{x}^{(m)} = \mathbf{x}) = 1$  时, 随机变量序列  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots$  收敛到  $\mathbf{x}$ 。

一致性保证了估计量的偏差会随数据样本数目的增多而减少。然而, 反过来是不正确的——渐近无偏并不意味着一致性。例如, 考虑用包含  $m$  个样本的数据集  $\{x^{(1)}, \dots, x^{(m)}\}$  估计正态分布  $\mathcal{N}(x; \mu, \sigma^2)$  的均值参数  $\mu$ 。我们可以使用数据集的第

一个样本  $x^{(1)}$  作为无偏估计量： $\hat{\theta} = x^{(1)}$ 。在该情况下， $\mathbb{E}(\hat{\theta}_m) = \theta$ ，所以不管观测到多少数据点，该估计量都是无偏的。然而，这不是一个一致估计，因为它不满足当  $m \rightarrow \infty$  时， $\hat{\theta}_m \rightarrow \theta$ 。

## 5.5 最大似然估计

之前，我们已经看过常用估计的定义，并分析了它们的性质。但是这些估计是从哪里来的呢？并非猜测某些函数可能是好的估计，然后分析其偏差和方差，我们希望有些准则可以让我们从不同模型中得到特定函数作为好的估计。

最常用的准则是最大似然估计。

考虑一组含有  $m$  个样本的数据集  $\mathbb{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ ，独立地由真正但未知的数据生成分布  $p_{\text{data}}(\mathbf{x})$  生成。

让  $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$  是一族由  $\boldsymbol{\theta}$  确定在相同空间上的概率分布。换言之， $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$  将任意输入  $\mathbf{x}$  映射到实数去估计真实概率  $p_{\text{data}}(\mathbf{x})$ 。

$\boldsymbol{\theta}$  的最大后验估计被定义为：

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}), \quad (5.56)$$

$$= \arg \max_{\boldsymbol{\theta}} \prod_{i=1}^m p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\theta}). \quad (5.57)$$

很多概率的乘积会因很多原因不方便。例如，很容易出现数值下溢。为了得到一个更方便但是等价的优化问题，我们观察到似然对数不会改变其  $\arg \max$  但是便利地将乘积转化成了和：

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} \sum_{i=1}^m \log p_{\text{model}}(\mathbf{x}^{(i)}; \boldsymbol{\theta}). \quad (5.58)$$

因为当我们重新缩放损失函数时  $\arg \max$  不会改变，我们可以除以  $m$  得到和训练数据经验分布  $\hat{p}_{\text{data}}$  相关的期望作为准则：

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta}). \quad (5.59)$$

解释最大似然估计的一种观点是将它看作最小化训练集上的经验分布  $\hat{p}_{\text{data}}$  和

模型分布之间的差异，两者之间的差异程度可以通过 KL 散度度量。KL 散度定义为

$$D_{\text{KL}}(\hat{p}_{\text{data}} \| p_{\text{model}}) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(\mathbf{x}) - \log p_{\text{model}}(\mathbf{x})]. \quad (5.60)$$

左边一项仅涉及到数据生成过程，和模型无关。这意味着当我们训练模型最小化 KL 散度时，我们只需要最小化

$$-\mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log p_{\text{model}}(\mathbf{x})], \quad (5.61)$$

当然，这和式(5.59)中最大化是相同的。

最小化 KL 散度其实就是在最小化分布之间的交叉熵。许多作者使用术语“交叉熵”特定表示伯努利或 softmax 分布的负对数似然，但那是用词不当的。任何一个由负对数似然组成的损失都是定义在训练集上的经验分布和定义在模型上的概率分布之间的交叉熵。例如，均方误差是经验分布和高斯模型之间的交叉熵。

我们可以将最大似然看作是使模型分布尽可能和经验分布  $\hat{p}_{\text{data}}$  相匹配的尝试。理想情况下，我们希望匹配真实的数据生成分布  $p_{\text{data}}$ ，但我们没法直接知道这个分布。

虽然最优  $\theta$  在最大化似然或是最小化 KL 散度时是相同的，但是目标函数值是不一样的。在软件中，我们通常将两者都称为最小化损失函数。因此最大化似然变成了最小化负对数似然 (NLL)，或者等价的是最小化交叉熵。将最大化似然看作最小化 KL 散度的视角在这个情况下是有帮助的，因为已知 KL 散度最小值是零。当  $x$  取实数时，负对数似然是负值。

### 5.5.1 条件对数似然和均方误差

最大似然估计很容易扩展到估计条件概率  $P(\mathbf{y} | \mathbf{x}; \theta)$ ，给定  $\mathbf{x}$  预测  $\mathbf{y}$ 。实际上这是最常见的情况，因为这构成了大多数监督学习的基础。如果  $\mathbf{X}$  表示所有的输入， $\mathbf{Y}$  表示我们观测到的目标，那么条件最大似然估计是

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\theta} P(\mathbf{Y} | \mathbf{X}; \theta). \quad (5.62)$$

如果假设样本是独立同分布的，那么这可以分解成

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\theta} \sum_{i=1}^m \log P(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}; \theta). \quad (5.63)$$

**实例：线性回归作为最大似然** 第5.1.4节介绍的线性回归，可以被看作是最大似然过程。之前，我们将线性回归作为学习从输入  $\mathbf{x}$  映射到输出  $\hat{y}$  的算法。从  $\mathbf{x}$  到  $\hat{y}$  的映射选自最小化均方误差（我们或多或少介绍的一个标准）。现在，我们以最大似然估计的角度重新审视线性回归。不只是得到一个单独的预测  $\hat{y}$ ，我们现在希望模型能够得到条件概率  $p(y | \mathbf{x})$ 。想象下有一个无限大的训练集，我们可能会观测到几个训练样本有相同的输入  $\mathbf{x}$  但是不同的  $y$ 。现在学习算法的目标是拟合分布  $p(y | \mathbf{x})$  到和  $\mathbf{x}$  相匹配的不同的  $y$ 。为了得到我们之前推导出的相同的线性回归算法，我们定义  $p(y | \mathbf{x}) = \mathcal{N}(y; \hat{y}(\mathbf{x}; \mathbf{w}), \sigma^2)$ 。函数  $\hat{y}(\mathbf{x}; \mathbf{w})$  预测高斯的均值。在这个例子中，我们假设方差是用户固定的某个常量  $\sigma^2$ 。这种函数形式  $p(y | \mathbf{x})$  会使得最大似然估计得出之前得到的相同学习算法。由于假设样本是独立同分布的，条件对数似然（式(5.63)）如下

$$\sum_{i=1}^m \log p(y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}) \quad (5.64)$$

$$= -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^m \frac{\|\hat{y}^{(i)} - y^{(i)}\|^2}{2\sigma^2}, \quad (5.65)$$

其中  $\hat{y}^{(i)}$  是线性回归在第  $i$  个输入  $\mathbf{x}^{(i)}$  上的输出， $m$  是训练样本的数目。对比于均方误差的对数似然，

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^m \|\hat{y}^{(i)} - y^{(i)}\|^2, \quad (5.66)$$

我们立刻可以看出最大化关于  $\mathbf{w}$  的对数似然和最小化均方误差会得到相同的参数估计  $\mathbf{w}$ 。但是对于相同的最优  $\mathbf{w}$ ，这两个准则有着不同的值。这验证了 MSE 可以用于最大似然估计。正如我们将看到的，最大似然估计有几个理想的性质。

### 5.5.2 最大似然的性质

最大似然估计最吸引人的地方在于，它被证明是当样本数目  $m \rightarrow \infty$  时，就收敛率而言最好的渐近估计。

在合适的条件下，最大似然估计具有一致性（参考第5.4.5节），意味着训练样本数目趋向于无限大时，参数的最大似然估计收敛到参数的真实值。这些条件是：

- 真实分布  $p_{\text{data}}$  必须在模型族  $p_{\text{model}}(\cdot; \boldsymbol{\theta})$  中。否则，没有估计可以表示  $p_{\text{data}}$ 。

- 真实分布  $p_{\text{data}}$  必须刚好对应一个  $\theta$  值。否则，最大似然学习恢复出真实分布  $p_{\text{data}}$  后，也不能决定数据生成过程使用哪个  $\theta$ 。

除了最大似然估计，还有其他的归纳准则，其中许多共享一致估计的性质。然而，一致估计的统计效率 (statistic efficiency) 可能区别很大。某些一致估计可能会在固定数目的样本上获得一个较低的泛化误差，或者等价地，可能只需要较少的样本就能达到一个固定程度的泛化误差。

通常，统计效率研究于有参情况 (parametric case) (例如线性回归)。有参情况下我们的目标是估计参数值 (假设有可能确定真实参数)，而不是函数值。一种度量我们和真实参数相差多少的方法是计算均方误差期望，即计算  $m$  个从数据生成分布中出来的训练样本上的估计参数和真实参数之间差值的平方。有参均方误差估计随着  $m$  的增加而减少，当  $m$  较大时，Cramér-Rao 下界 (Rao, 1945; Cramér, 1946) 表明不存在均方误差低于最大似然学习的一致估计。

因为这些原因 (一致性和统计效率)，最大似然通常是机器学习中的首选估计。当样本数目小到会过拟合时，正则化策略如权重衰减可用于获得训练数据有限时方差较小的最大似然有偏版本。

## 5.6 贝叶斯统计

至此我们已经讨论了频率统计 (frequentist statistics) 方法和基于估计单一值  $\theta$  的方法，然后基于该估计作所有的预测。另一种方法是在做预测时会考虑所有可能  $\theta$ 。后者属于贝叶斯统计 (Bayesian statistics) 的范畴。

正如第5.4.1节中讨论的，频率派的视角是真实参数  $\theta$  是未知的定值，而点估计  $\hat{\theta}$  是考虑数据集上函数（可以看作是随机的）的随机变量。

贝叶斯统计的视角完全不同。贝叶斯用概率反映知识状态的确定性程度。数据集能够直接观测到，因此不是随机的。另一方面，真实参数  $\theta$  是未知或不确定的，因此可以表示成随机变量。

在观察到数据前，我们将  $\theta$  的已知知识表示成先验概率分布 (prior probability distribution)， $p(\theta)$  (有时简单地称为“先验”)。一般而言，机器学习实践者会选择一个相当宽泛的（即，高熵的）先验分布，反映在观测到任何数据前参数  $\theta$  的高度不确定性。例如，我们可能会假设先验  $\theta$  在有限区间中均匀分布。许多先验偏好于

“更简单”的解决方法（如小幅度的系数，或是接近常数的函数）。

现在假设有一组数据样本  $\{x^{(1)}, \dots, x^{(m)}\}$ 。通过贝叶斯法则结合数据似然  $p(x^{(1)}, \dots, x^{(m)} | \boldsymbol{\theta})$  和先验，我们可以恢复数据对我们关于  $\boldsymbol{\theta}$  信念的影响：

$$p(\boldsymbol{\theta} | x^{(1)}, \dots, x^{(m)}) = \frac{p(x^{(1)}, \dots, x^{(m)} | \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(x^{(1)}, \dots, x^{(m)})} \quad (5.67)$$

在贝叶斯估计通常使用的情况下，先验开始是相对均匀的分布或高熵的高斯分布，观测数据通常会使后验的熵下降，并集中在参数的几个可能性很高的值。

相对于最大似然估计，贝叶斯估计有两个重要区别。第一，不像最大似然方法预测时使用  $\boldsymbol{\theta}$  的点估计，贝叶斯方法预测  $\boldsymbol{\theta}$  的全分布。例如，在观测到  $m$  个样本后，下一个数据样本， $x^{(m+1)}$ ，的预测分布如下：

$$p(x^{(m+1)} | x^{(1)}, \dots, x^{(m)}) = \int p(x^{(m+1)} | \boldsymbol{\theta})p(\boldsymbol{\theta} | x^{(1)}, \dots, x^{(m)})d\boldsymbol{\theta}. \quad (5.68)$$

这里，具有正概率密度的  $\boldsymbol{\theta}$  的每个值有助于下一个样本的预测，其中贡献由后验密度本身加权。在观测到数据集  $\{x^{(1)}, \dots, x^{(m)}\}$  之后，如果我们仍然非常不确定  $\boldsymbol{\theta}$  的值，那么这个不确定性会直接包含在我们所做的任何预测中。

在第5.4节中，我们已经探讨频率派方法解决给定点估计  $\boldsymbol{\theta}$  不确定性的方法是评估方差，估计的方差评估了观测数据重新从观测数据中采样后，估计可能如何变化。对于如何处理估计不确定性的这个问题，贝叶斯派的答案是积分，这往往会防止过拟合。积分当然是概率法则的应用，使贝叶斯方法容易验证，而频率派机器学习基于相当特别的决定构建了一个估计，将数据集里的所有信息归纳到一个单独的点估计。

贝叶斯方法和最大似然方法的第二个最大区别是由贝叶斯先验分布造成的。先验能够影响概率质量密度朝参数空间中偏好先验的区域偏移。实践中，先验通常表现为偏好更简单或更光滑的模型。对贝叶斯方法的批判认为先验是人为主观判断影响预测的来源。

当训练数据很有限时，贝叶斯方法通常泛化得更好，但是当训练样本数目很大时，通常会有很高的计算代价。

**实例：贝叶斯线性回归** 我们使用贝叶斯估计方法学习线性回归参数。在线性回归中，我们学习从输入向量  $\mathbf{x} \in \mathbb{R}^n$  预测标量  $y \in \mathbb{R}$  的映射。该预测参数化为向量  $\mathbf{w} \in \mathbb{R}^n$ ：

$$\hat{y} = \mathbf{w}^\top \mathbf{x}. \quad (5.69)$$

给定一组  $m$  个训练样本  $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$ ，我们可以表示整个训练集对  $y$  的预测为：

$$\hat{\mathbf{y}}^{(\text{train})} = \mathbf{X}^{(\text{train})}\mathbf{w}. \quad (5.70)$$

表示为  $\mathbf{y}^{(\text{train})}$  上的高斯条件分布，我们得到

$$p(\mathbf{y}^{(\text{train})} | \mathbf{X}^{(\text{train})}, \mathbf{w}) = \mathcal{N}(\mathbf{y}^{(\text{train})}; \mathbf{X}^{(\text{train})}\mathbf{w}, \mathbf{I}) \quad (5.71)$$

$$\propto \exp\left(-\frac{1}{2}(\mathbf{y}^{(\text{train})} - \mathbf{X}^{(\text{train})}\mathbf{w})^\top(\mathbf{y}^{(\text{train})} - \mathbf{X}^{(\text{train})}\mathbf{w})\right), \quad (5.72)$$

其中，我们根据标准的 MSE 公式假设  $y$  上的高斯方差为 1。在下文中，为减少符号负担，我们将  $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$  简单表示为  $(\mathbf{X}, \mathbf{y})$ 。

确定模型参数向量  $\mathbf{w}$  的后验分布，我们首先需要指定一个先验分布。先验应该反映我们对这些参数取值的信念。虽然有时很难或很不自然将我们的先验信念表示为模型的参数，在实践中我们通常假设一个相当广泛的分布来表示  $\theta$  的高度不确定性。实数值参数通常使用高斯作为先验分布：

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0) \propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Lambda}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right), \quad (5.73)$$

其中， $\boldsymbol{\mu}_0$  和  $\boldsymbol{\Lambda}_0$  分别是先验分布的均值向量和协方差矩阵。<sup>1</sup>

确定好先验后，我们现在可以继续确定模型参数的后验分布。

$$p(\mathbf{w} | \mathbf{X}, \mathbf{y}) \propto p(\mathbf{y} | \mathbf{X}, \mathbf{w})p(\mathbf{w}) \quad (5.74)$$

$$\propto \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})\right) \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Lambda}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right) \quad (5.75)$$

$$\propto \exp\left(\frac{1}{2}(-2\mathbf{y}^\top \mathbf{X}\mathbf{w} + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} + \mathbf{w}^\top \boldsymbol{\Lambda}_0^{-1}\mathbf{w} - 2\boldsymbol{\mu}_0^\top \boldsymbol{\Lambda}_0^{-1}\mathbf{w})\right). \quad (5.76)$$

现在我们定义  $\boldsymbol{\Lambda}_m = (\mathbf{X}^\top \mathbf{X} + \boldsymbol{\Lambda}_0^{-1})^{-1}$  和  $\boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m(\mathbf{X}^\top \mathbf{y} + \boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)$ 。使用这些新的变量，我们发现后验可改写为高斯分布：

$$p(\mathbf{w} | \mathbf{X}, \mathbf{y}) \propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_m)^\top \boldsymbol{\Lambda}_m^{-1}(\mathbf{w} - \boldsymbol{\mu}_m) + \frac{1}{2}\boldsymbol{\mu}_m^\top \boldsymbol{\Lambda}_m^{-1}\boldsymbol{\mu}_m\right) \quad (5.77)$$

$$\propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_m)^\top \boldsymbol{\Lambda}_m^{-1}(\mathbf{w} - \boldsymbol{\mu}_m)\right). \quad (5.78)$$

<sup>1</sup>除非有理由使用协方差矩阵的特定结构，我们通常假设其为对角协方差矩阵  $\boldsymbol{\Lambda}_0 = \text{diag}(\boldsymbol{\lambda}_0)$ 。

所有不包括的参数向量  $\mathbf{w}$  的项都已经被删去了；它们意味着分布的积分必须归一这个事实。式(3.23)显示了如何标准化多元高斯分布。

检查此后的验分布可以让我们获得贝叶斯推断效果的一些直觉。大多数情况下，我们设置  $\mu_0 = 0$ 。如果我们设置  $\Lambda_0 = \frac{1}{\alpha} \mathbf{I}$ ，那么  $\mu_m$  对  $\mathbf{w}$  的估计就和频率派带权重衰减惩罚  $\alpha \mathbf{w}^\top \mathbf{w}$  的线性回归的估计一样。一个区别是若  $\alpha$  设为 0 则贝叶斯估计是未定义的——不能初始化贝叶斯学习过程为一个无限宽的  $\mathbf{w}$  先验。更重要的区别是贝叶斯估计会给出一个协方差矩阵，表示  $\mathbf{w}$  所有不同值的可能范围，而不仅是估计  $\mu_m$ 。

### 5.6.1 最大后验 (MAP) 估计

虽然使用完整的贝叶斯后验分布进行参数  $\boldsymbol{\theta}$  预测是非常合理的，但仍常常希望能够进行单点估计。希望点估计的一个常见原因是，对于非常有趣的模型而言，大部分涉及到贝叶斯后验的操作是非常棘手的，点估计提供了一个可解的近似。并非简单地回归到最大似然学习，我们仍然可以通过先验影响点估计的选择而获取贝叶斯方法的优点。一种能够做到这一点的合理方式是选择最大后验 (Maximum A Posteriori, MAP) 点估计。MAP 估计选择后验概率最大的点（或在  $\boldsymbol{\theta}$  是连续值的更常见情况下，概率密度最大的点）：

$$\boldsymbol{\theta}_{\text{MAP}} = \arg \max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} | \mathbf{x}) = \arg \max_{\boldsymbol{\theta}} \log p(\mathbf{x} | \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \quad (5.79)$$

我们可以认出上式右边的  $\log p(\mathbf{x} | \boldsymbol{\theta})$  对应着标准的对数似然项， $\log p(\boldsymbol{\theta})$  对应着先验分布。

例如，考虑具有高斯先验权重  $\mathbf{w}$  的线性回归模型。如果先验是  $\mathcal{N}(\mathbf{w}; \mathbf{0}, \frac{1}{\lambda} I^2)$ ，那么式(5.79)的对数先验项正比于熟悉的权重衰减惩罚  $\lambda \mathbf{w}^\top \mathbf{w}$ ，加上一个不依赖于  $\mathbf{w}$  也不会影响学习过程的项。因此，具有高斯先验权重的最大后验贝叶斯推断对应着权重衰减。

正如全贝叶斯推断，MAP 贝叶斯推断具有训练数据没有的，先验带来的信息利用优势。该附加信息有助于减少最大后验点估计的方差（相比于 ML 估计）。然而，这个优点的代价是增加了偏差。

许多正规化估计方法，例如权重衰减正则化的最大似然学习，可以被解释为贝叶斯推断的 MAP 近似。这个解释产生于正则化时加到目标函数的附加项对应着  $\log p(\boldsymbol{\theta})$ 。并非所有的正规化惩罚都对应于贝叶斯推断。例如，有些正则化项可能不

是一个概率分布的对数。还有些正则化项依赖于数据，当然也不会是一个先验概率分布。

MAP贝叶斯推断提供了一个直观的方法去设计复杂但可解释的正则化项。例如，更复杂的惩罚项可以通过混合高斯作为先验得到，而不是一个单独的高斯分布(Nowlan and Hinton, 1992)。

## 5.7 监督学习算法

回顾第5.1.3节，监督学习算法，粗略地说，是给定一组输入  $\mathbf{x}$  和输出  $\mathbf{y}$  的训练集，学习如何关联输入和输出。在许多情况下，输出  $\mathbf{y}$  很难自动收集，必须由人来提供“管理”，不过该术语仍然适用于训练集目标可以被自动收集的情况。

### 5.7.1 概率监督学习

本书的大部分监督学习算法都是基于估计概率分布  $p(y | \mathbf{x})$ 。我们可以使用最大似然估计找到对于有参分布族  $p(y | \mathbf{x}; \boldsymbol{\theta})$  最好的参数向量  $\boldsymbol{\theta}$ 。

我们已经看到，线性回归对应于分布族

$$p(y | \mathbf{x}; \boldsymbol{\theta}) = \mathcal{N}(y; \boldsymbol{\theta}^\top \mathbf{x}, \mathbf{I}). \quad (5.80)$$

通过定义一族不同的概率分布，我们可以将线性回归扩展到分类情况中。如果我们有两个类，类 0 和类 1，那么我们只需要指定这两类之一的概率。类 1 的概率决定了类 0 的概率，因为这两个值加起来必须等于 1。

我们用于线性回归的实数正态分布是用均值参数化的。我们提供这个均值的任何值都是有效的。二元变量上的的分布稍微复杂些，因为它的均值必须始终在 0 和 1 之间。解决这个问题的一种方法是使用 logistic sigmoid 函数将线性函数的输出压缩进区间 (0, 1)。该值可以解释为概率：

$$p(y = 1 | \mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \mathbf{x}). \quad (5.81)$$

这个方法被称为逻辑回归 (logistic regression)，名字有点奇怪，因为该模型用于分类，而非回归。

线性回归中，我们能够通过求解正规方程以找到最佳权重。相比而言，逻辑回

归会更困难些。其最佳权重没有闭解。反之，我们必须最大化对数似然来搜索最优解。我们可以通过梯度下降最小化负对数似然达到这一点。

通过确定正确的输入和输出变量上的有参条件概率分布族，相同的策略基本上可以用于任何监督学习问题。

### 5.7.2 支持向量机

**支持向量机** (support vector machine, SVM) 是监督学习中最有影响力的方法之一 (Boser *et al.*, 1992; Cortes and Vapnik, 1995)。类似于逻辑回归，这个模型也是基于线性函数  $\mathbf{w}^\top \mathbf{x} + b$ 。不同于逻辑回归的是，支持向量机不输出概率，只输出类别。当  $\mathbf{w}^\top \mathbf{x} + b$  为正时，支持向量机预测属于正类。类似地，当  $\mathbf{w}^\top \mathbf{x} + b$  为负时，支持向量机预测属于负类。

支持向量机的一个重要创新是核技巧 (kernel trick)。核策略观察到许多机器学习算法都可以写成样本间点积的形式。例如，支持向量机中的线性函数可以重写为

$$\mathbf{w}^\top \mathbf{x} + b = b + \sum_{i=1}^m \alpha_i \mathbf{x}^\top \mathbf{x}^{(i)}, \quad (5.82)$$

其中， $\mathbf{x}^{(i)}$  是训练样本， $\boldsymbol{\alpha}$  是系数向量。学习算法重写为这种形式允许我们将  $\mathbf{x}$  替换为特征函数  $\phi(\mathbf{x})$  的输出，点积替换为被称为核函数 (kernel function) 的函数  $k(\mathbf{x}, \mathbf{x}^{(i)}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}^{(i)})$ 。· 操作表示类似于  $\phi(\mathbf{x})^\top \phi(\mathbf{x}^{(i)})$  的点积。对于某些特征空间，我们可能不会书面地使用向量内积。在某些无限维空间中，我们需要使用其他类型的内积，如基于积分而非总和的内积。这种类型内积的完整介绍超出了本书的范围。

核估计替换点积之后，我们可以使用如下函数进行预测

$$f(\mathbf{x}) = b + \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)}). \quad (5.83)$$

这个函数关于  $\mathbf{x}$  是非线性的，关于  $\phi(\mathbf{x})$  是线性的。 $\boldsymbol{\alpha}$  和  $f(\mathbf{x})$  之间的关系也是线性的。核函数完全等价于用  $\phi(\mathbf{x})$  预处理所有的输入，然后在新的转换空间学习线性模型。

核策略十分强大有两个原因。首先，它使我们能够使用保证有效收敛的凸优化技术来学习作为  $\mathbf{x}$  的函数的非线性模型。这是可能的，因为我们可以认为  $\phi$  是固定

的，仅优化  $\alpha$ ，即优化算法可以将决策函数视为不同空间中的线性函数。其二，核函数  $k$  的实现方法通常有比直接构建  $\phi(\mathbf{x})$  再算点积高效很多。

在某些情况下， $\phi(\mathbf{x})$  甚至可以是无限维的，对于普通的显式方法而言，这将是无限的计算代价。在很多情况下，即使  $\phi(\mathbf{x})$  是难算的， $k(\mathbf{x}, \mathbf{x}')$  却会是一个关于  $\mathbf{x}$  非线性的，易算的函数。举个无限维空间易解的核的例子，我们构建一个作用于非负整数  $x$  上的特征映射  $\phi(x)$ 。假设这个映射返回一个由开头  $x$  个 1，随后时无限个 0 的向量。我们可以写一个核函数  $k(x, x^{(i)}) = \min(x, x^{(i)})$ ，完全等价于对应的无限维点积。

最常用的核函数是高斯核 (Gaussian kernel)，

$$k(\mathbf{u}, \mathbf{v}) = \mathcal{N}(\mathbf{u} - \mathbf{v}; \mathbf{0}, \sigma^2 I), \quad (5.84)$$

其中  $\mathcal{N}(x; \mu, \Sigma)$  是标准正态密度。这个核也被称为径向基函数 (radial basis function, RBF) 核，因为其值沿  $\mathbf{v}$  中从  $\mathbf{u}$  向外辐射的方向减小。高斯核对应于无限维空间中的点积，但是该空间的推导没有整数上的 min 核实例直观。

我们可以认为高斯核在执行一种模板匹配。训练标签  $y$  相关的训练样本  $\mathbf{x}$  变成了类别  $y$  的模版。当测试点  $\mathbf{x}'$  到  $\mathbf{x}$  的欧几里得距离很小时，对应的高斯核很大，表明  $\mathbf{x}'$  和模版  $\mathbf{x}$  非常相似。该模型进而会赋予相对应的训练标签  $y$  较大的权重。总的来说，预测将会组合很多这种通过训练样本相似性加权的训练标签。

支持向量机不是唯一可以使用核策略来增强的算法。许多其他的线性模型可以通过这种方式来增强。使用核策略的算法类别被称为核机器 (kernel machine) 或核方法 (kernel method) (Williams and Rasmussen, 1996; Schölkopf *et al.*, 1999)。

核机器的一个主要缺点是计算决策函数的成本关于训练样本的数目是线性的。因为第  $i$  个样本贡献  $\alpha_i k(\mathbf{x}, \mathbf{x}^{(i)})$  到决策函数。支持向量机能够通过学习主要包含零的向量  $\boldsymbol{\alpha}$ ，以缓和这个缺点。那么判断新样本的类别仅需要计算非零  $\alpha_i$  对应的训练样本的核函数。这些训练样本被称为支持向量 (support vector)。

当数据集很大时，核机器的计算量也会很大。我们将会在第5.9节回顾这个想法。带通用核的核机器会泛化得更好。我们将在第5.11节解释原因。现代深度学习的设计旨在克服核机器的这些限制。当前深度学习的复兴始于Hinton *et al.* (2006b) 表明神经网络能够在 MNIST 基准数据上胜过 RBF 核的支持向量机。

### 5.7.3 其他简单的监督学习算法

我们已经简要介绍过另一个非概率监督学习算法，近邻回归。更一般地， $k$ -最近邻是一类可用于分类或回归的技术。作为一个非参数学习算法， $k$ -最近邻并不局限于固定数目的参数。我们通常认为  $k$ -最近邻算法没有任何参数，而是使用训练数据的简单函数。事实上，甚至也没有一个真正的训练阶段或学习过程。反之，在测试阶段我们希望在新的测试输入  $\mathbf{x}$  上产生  $y$ ，我们需要在训练数据  $\mathbf{X}$  上找到  $\mathbf{x}$  的  $k$ -最近邻。然后我们返回训练集上对应的  $y$  值的平均值。这几乎适用于任何类型可以确定  $y$  值平均值的监督学习。在分类情况中，我们可以关于one-hot编码向量  $\mathbf{c}$  求平均，其中  $c_y = 1$ ，其他的  $i$  值取  $c_i = 0$ 。然后，我们可以解释这些one-hot编码的均值为类别的概率分布。作为一个非参数学习算法， $k$ -近邻能达到非常高的容量。例如，假设我们有一个用 0-1 误差度量性能的多分类任务。在此设定中，当训练样本数目趋向于无限大时，1-最近邻收敛到两倍贝叶斯误差。超出贝叶斯误差的原因是它会随机从等距离的临近点中随机挑一个。当有无限的训练数据时，所有测试点  $\mathbf{x}$  周围距离为零的邻近点有无限多个。如果我们使用所有这些临近点投票的决策方式，而不是随机挑选一个，那么该过程将会收敛到贝叶斯错误率。 $k$ -最近邻的高容量使其在训练样本数目大时能够获取较高的精度。然而，它的计算成本很高，另外在训练集较小时泛化能力很差。 $k$ -最近邻的一个弱点是它不能学习出哪一个特征比其他更具识别力。例如，假设我们在做从各向同性的高斯分布中抽取  $\mathbf{x} \in \mathbb{R}^{100}$  的回归任务，但是只有一个变量  $x_1$  和结果相关。进一步假设该特征直接决定了输出，即在所有情况下  $y = x_1$ 。最近邻回归不能检测到这个简单模式。大多数点  $\mathbf{x}$  的最近邻将取决于  $x_2$  到  $x_{100}$  的大多数特征，而不是单独取决于特征  $x_1$ 。因此，小训练集上的输出将会非常随机。

决策树 (decision tree)，及其变种是一类将输入空间分成不同的区域，每个区域有独立的参数的算法 (Breiman *et al.*, 1984)。如图5.7所示，决策树的每个节点都与输入空间的一个区域相关联，并且内部节点继续将区域分成子节点下的子区域（通常使用坐标轴拆分区域）。空间由此细分成不重叠的区域，叶节点和输入区域之间形成一一对应的关系。每个叶结点将其输入区域的每个点映射到相同的输出。决策树通常有特定的训练算法，超出了本书的范围。如果允许学习任意大小的决策树，那么可以被视作非参数算法。然而实践中通常有大小限制作为正则化项将其转变成有参模型。由于决策树通常使用坐标轴相关的拆分，并且每个子节点关联到常数输出，因此有时解决一些对于逻辑回归很简单的问题很费力。例如，假设有一个二分类问题，当  $x_2 > x_1$  时分为正类，则决策树的分界不是坐标轴对齐的。决策树将需要许

多节点近似决策边界，坐标轴对齐使其算法步骤将不断来回穿梭于真正的决策函数。

正如我们已经看到的，最近邻预测和决策树都有很多的局限性。尽管如此，在计算资源受限制时，它们都是很有用的学习算法。通过思考复杂算法和  $k$ -最近邻或决策树之间的相似性和差异，我们可以建立对更复杂学习算法的直觉。

参考Murphy (2012); Bishop (2006); Hastie *et al.* (2001) 或其他机器学习教科书了解更多的传统监督学习算法。

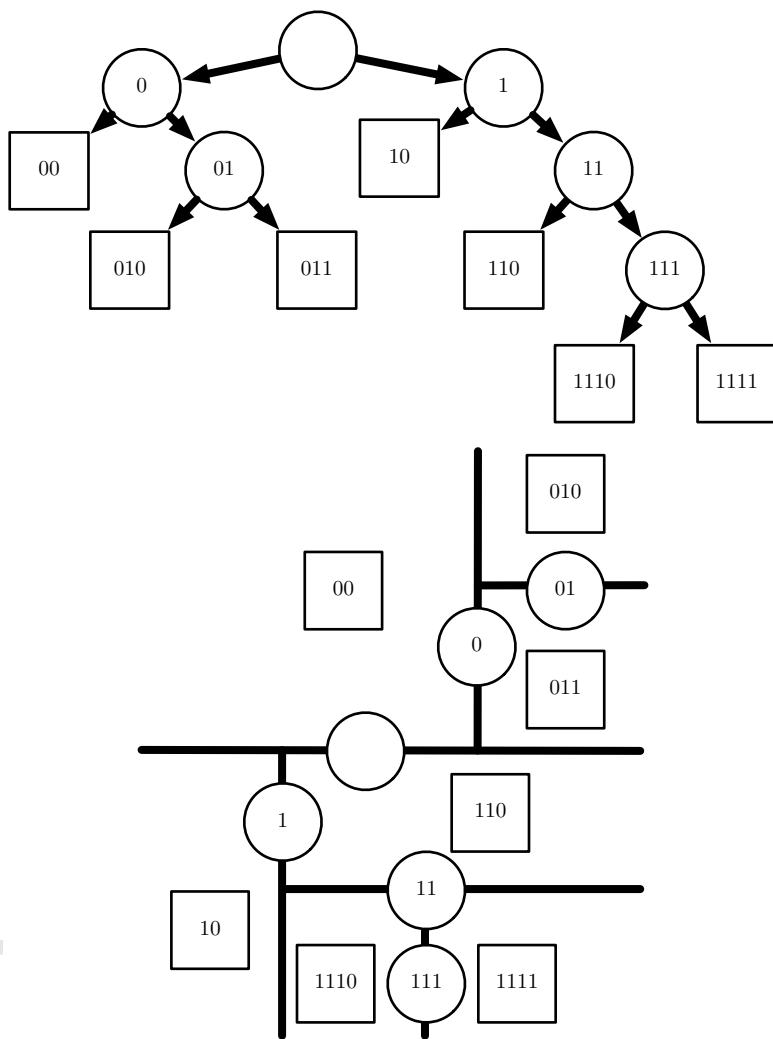


图 5.7: 描述一个决策树如何工作的图。(上) 树中每个节点都选择将输入样本送到左子节点 (0) 或者右子节点 (1)。内部的节点用圆圈表示, 叶节点用方块表示。每一个节点可以用一个二值的字符串识别并对应树中的位置, 这个字符串是通过给起父节点的字符串添加一个 bit 位来实现的 (0 表示选择左或者上, 1 表示选择右或者下)。(下) 这个树将空间分为区域。这个二维平面说明决策树可以分割  $\text{Set}R^2$ 。平面中画了树的节点, 每个内部点穿过分割线并用来给样本分类, 叶节点画在样本所属区域的中心。结果是一个分块常数函数, 每一个叶节点一个区域。每个叶需要至少一个训练样本来定义, 所以决策树不可能用来学习一个拥有比训练样本数量还多的局部极大值的函数。

## 5.8 无监督学习算法

回顾第5.1.3节，无监督算法只处理“特征”，不操作监督信号。监督和无监督算法之间的区别没有规范，严格的定义，因为没有客观的判断来区分监督者提供的值是特征还是目标。通俗地说，无监督学习是指从不需要人为注释样本的分布中抽取信息的大多数尝试。该术语通常与密度估计相关，学习从分布中采样，学习从分布中去噪，需要数据分布的流形，或是将数据中相关的样本聚类。

一个经典的无监督学习任务是找到数据的“最佳”表示。“最佳”可以是不同的表示，但是一般来说，是指该表示在比本身表示的信息更简单或更易访问而受到一些惩罚或限制的情况下，尽可能保存关于  $x$  更多的信息。

有很多方式定义较简单的表示。最常见的三种包括低维表示，稀疏表示，独立表示。低维表示尝试将  $x$  中的信息尽可能压缩在一个较小的表示中。稀疏表示将数据集嵌入到输入项大多数为零的表示中 (Barlow, 1989; Olshausen and Field, 1996; Hinton and Ghahramani, 1997)。稀疏表示通常用于需要增加表示维数的情况，使得大部分为零的表示不会丢失很多信息。这会使得表示的整体结构倾向于将数据分布在表示空间的坐标轴上。独立表示试图解开数据分布中变动的来源，使得表示的维度是统计独立的。

当然这三个标准并非相互排斥的。低维表示通常会产生比原始的高维数据具有较少或较弱依赖关系的元素。这是因为减少表示大小的一种方式是找到并消除冗余。识别并去除更多的冗余使得降维算法在丢失更少信息的同时显现更大的压缩。

表示的概念是深度学习核心主题之一，因此也是本书的核心主题之一。本节会介绍表示学习算法中的一些简单实例。总的来说，这些实例算法会说明如何实施上面的三个标准。剩余的大部分章节会介绍其他表示学习算法以不同方式处理这三个标准或是介绍其他标准。

### 5.8.1 主成分分析

在第2.12节中，我们看到PCA算法提供了一种压缩数据的方式。我们也可以将PCA视为学习数据表示的无监督学习算法。这种表示基于上述简单表示的两个标准。PCA学习一种比原始输入低维的表示。它也学习了一种元素之间彼此没有线性相关的表示。这是学习表示中元素统计独立标准的第一步。要实现完全独立性，表示学习算法必须也去掉变量间的非线性关系。

如图5.8所示，PCA将输入  $\mathbf{x}$  投影表示成  $\mathbf{z}$ ，学习数据的正交，线性变换。在第2.12节中，我们看到了如何学习重建原始数据的最佳一维表示（就均方误差而言），这种表示其实对应着数据的第一个主要成分。因此，我们可以用PCA作为保留数据尽可能多信息的降维方法（再次是就最小重构误差平方而言）。在下文中，我们将研究PCA表示如何使原始数据表示  $\mathbf{X}$  去相关的。

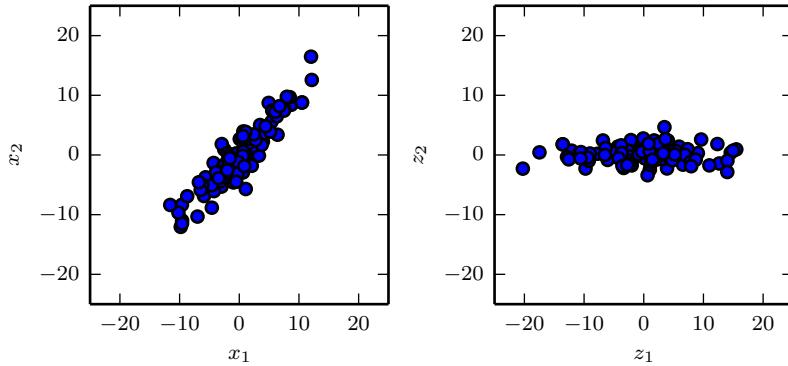


图 5.8: PCA学习了一种线性投影， $\mathbf{x}$  投影的方向对齐了新的空间最大方差的轴。（左）原始数据包含了  $\mathbf{x}$  的样本。在这个空间中，方差的方向与轴的方向并不是对齐的。（右）变换过的数据  $\mathbf{z} = \mathbf{x}^\top \mathbf{W}$  在轴  $z_1$  的方向上有最大的变化。第二大变化方差的方向沿着轴  $z_2$ 。

假设有一个  $m \times n$  的设计矩阵  $\mathbf{X}$ ，数据的均值为零， $\mathbb{E}[\mathbf{x}] = 0$ 。若非如此，通过预处理步骤所有样本减去均值，数据可以很容易地中心化。

$\mathbf{X}$  对应的无偏样本协方差矩阵给定如下

$$\text{Var}[\mathbf{x}] = \frac{1}{m-1} \mathbf{X}^\top \mathbf{X}. \quad (5.85)$$

PCA会找到一个  $\text{Var}[\mathbf{z}]$  是对角矩阵的表示（通过线性变换） $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$ 。

在第2.12节，我们看到设计矩阵  $\mathbf{X}$  的主成分由  $\mathbf{X}^\top \mathbf{X}$  的特征向量给定。从这个角度，我们有

$$\mathbf{X}^\top \mathbf{X} = \mathbf{W} \Lambda \mathbf{W}^\top. \quad (5.86)$$

本节中，我们会探索主成分分析的另一种推导。主成分也可以通过奇异值分解得到。具体地，它们是  $\mathbf{X}$  的右奇异向量。为了说明这点，假设  $\mathbf{W}$  是奇异值分解  $\mathbf{X} = \mathbf{U} \Sigma \mathbf{W}^\top$  的右奇异向量。以  $\mathbf{W}$  作为特征向量基，我们可以得到原来的特征向量

方程：

$$\mathbf{X}^\top \mathbf{X} = (\mathbf{U}\boldsymbol{\Sigma}\mathbf{W}^\top)^\top \mathbf{U}\boldsymbol{\Sigma}\mathbf{W}^\top = \mathbf{W}\boldsymbol{\Sigma}^2\mathbf{W}^\top. \quad (5.87)$$

SVD有助于说明PCA后的  $\text{Var}[\mathbf{z}]$  是对角的。使用  $\mathbf{X}$  的SVD分解， $\mathbf{X}$  的方差可以表示为

$$\text{Var}[\mathbf{x}] = \frac{1}{m-1} \mathbf{X}^\top \mathbf{X} \quad (5.88)$$

$$= \frac{1}{m-1} (\mathbf{U}\boldsymbol{\Sigma}\mathbf{W}^\top)^\top \mathbf{U}\boldsymbol{\Sigma}\mathbf{W}^\top \quad (5.89)$$

$$= \frac{1}{m-1} \mathbf{W}\boldsymbol{\Sigma}^\top \mathbf{U}^\top \mathbf{U}\boldsymbol{\Sigma}\mathbf{W}^\top \quad (5.90)$$

$$= \frac{1}{m-1} \mathbf{W}\boldsymbol{\Sigma}^2\mathbf{W}^\top, \quad (5.91)$$

其中，我们使用  $\mathbf{U}^\top \mathbf{U} = \mathbf{I}$ ，因为根据奇异值的定义矩阵  $\mathbf{U}$  是正交的。这表明  $\mathbf{z}$  的协方差满足对角的要求：

$$\text{Var}[\mathbf{z}] = \frac{1}{m-1} \mathbf{Z}^\top \mathbf{Z} \quad (5.92)$$

$$= \frac{1}{m-1} \mathbf{W}^\top \mathbf{X}^\top \mathbf{X}^\top \mathbf{W} \quad (5.93)$$

$$= \frac{1}{m-1} \mathbf{W}^\top \mathbf{W}\boldsymbol{\Sigma}^2\mathbf{W}^\top \mathbf{W} \quad (5.94)$$

$$= \frac{1}{m-1} \boldsymbol{\Sigma}^2, \quad (5.95)$$

其中，再次使用SVD的定义有  $\mathbf{W}^\top \mathbf{W} = \mathbf{I}$ 。

以上分析指明当我们通过线性变换  $\mathbf{W}$  将数据  $\mathbf{x}$  投影到  $\mathbf{z}$  时，得到的数据表示的协方差矩阵是对角的 ( $\boldsymbol{\Sigma}^2$ )，即说明  $\mathbf{z}$  中的元素是彼此无关的。

PCA这种将数据变换为元素之间彼此不相关表示的能力是PCA的一个重要性质。它是消除数据中未知变动因素的简单表示实例。在PCA中，这个消除是通过寻找输入空间的一个旋转（由  $\mathbf{W}$  确定），使得方差的主坐标和  $\mathbf{z}$  相关的新表示空间的基对齐。

虽然相关性是数据元素间依赖关系的一个重要范畴，但我们对于能够消除特征依赖更复杂形式的表示学习也很有兴趣。对此，我们需要比简单线性变换能做到更多的工具。

### 5.8.2 $k$ -均值聚类

另外一个简单的表示学习算法是  $k$ -均值聚类。 $k$ -均值聚类算法将训练集分成  $k$  个靠近彼此的不同样本聚类。因此我们可以认为该算法提供了  $k$ -维的one-hot编码向量  $\mathbf{h}$  以表示输入  $\mathbf{x}$ 。当  $\mathbf{x}$  属于聚类  $i$  时，有  $h_i = 1$ ， $\mathbf{h}$  的其他项为零。

$k$ -均值聚类提供的one-hot编码也是一种稀疏表示，因为每个输入的对应表示大部分元素为零。之后，我们会介绍能够学习更灵活的稀疏表示的一些其他算法（表示中每个输入  $\mathbf{x}$  不只一个非零项）。one-hot编码是稀疏表示的一个极端实例，丢失了很多分布式表示的优点。one-hot编码仍然有一些统计优点（自然地传达了相同聚类中的样本彼此相似的观点），也具有计算上的优势，因为整个表示可以用一个单独的整数表示。

$k$ -均值聚类初始化  $k$  个不同的中心点  $\{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(k)}\}$ ，然后迭代交换两个不同的步骤直到收敛。步骤一，每个训练样本分配到最近的中心点  $\boldsymbol{\mu}^{(i)}$  所代表的聚类  $i$ 。步骤二，每一个中心点  $\boldsymbol{\mu}^{(i)}$  更新为聚类  $i$  中所有训练样本  $\mathbf{x}^{(j)}$  的均值。

关于聚类的一个问题是聚类问题本身是病态的。这是说没有单一的标准去度量聚类的数据对应真实世界有多好。我们可以度量聚类的性质，例如每个聚类的元素到该类中心点的平均欧几里得距离。这使我们可以判断能够多好地从聚类分配中重建训练数据。然而我们不知道聚类的性质多好地对应于真实世界的性质。此外，可能有许多不同的聚类都能很好地对应到现实世界的某些属性。我们可能希望找到和一个特征相关的聚类，但是得到了一个和任务无关的不同的，同样是合理的聚类。例如，假设我们在包含红色卡车图片，红色汽车图片，灰色卡车图片和灰色汽车图片的数据集上运行两个聚类算法。如果每个聚类算法聚两类，那么可能一个算法将汽车和卡车各聚一类，另一个根据红色和灰色各聚一类。假设我们还运行了第三个聚类算法，用来决定类别的数目。这有可能聚成了四类，红色卡车，红色汽车，灰色卡车和灰色汽车。现在这个新的聚类至少抓住了属性的信息，但是损失掉了相似性信息。红色汽车和灰色汽车在不同的类中，正如红色汽车和灰色卡车也在不同的类中。该聚类算法没有告诉我们灰色汽车比灰色卡车和红色汽车更相似。我们只知道它们是不同的。

这些问题说明了一些我们可能更偏好于分布式表示（相对于one-hot表示而言）的原因。分布式表示可以对每个车辆赋予两个属性——一个表示它颜色，一个表示它是汽车还是卡车。目前仍然不清楚什么是最优的分布式表示（学习算法如何知道我们关心的两个属性是颜色和是否汽车或卡车，而不是制造商和车龄？），但是多个

属性减少了算法去猜我们关心哪一个属性的负担，允许我们通过比较很多属性而非测试一个单一属性来细粒度地度量相似性。

## 5.9 随机梯度下降

几乎所有的深度学习算法都用到了一个非常重要的算法：随机梯度下降 (stochastic gradient descent, SGD)。随机梯度下降是第4.3节介绍的梯度下降算法的一个扩展。

机器学习中的一个循环问题是大的数据集是好的泛化所必要的，但大的训练集的计算代价也更大。

机器学习算法中的损失函数通常可以分解成每个样本损失函数的总和。例如，训练数据的负条件对数似然可以写成

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim p_{\text{data}}} L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}), \quad (5.96)$$

其中  $L$  是每个样本的损失函数  $L(\mathbf{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \mathbf{x}; \boldsymbol{\theta})$ 。

对于这些相加的损失函数，梯度下降需要计算

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m \nabla_{\boldsymbol{\theta}} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}). \quad (5.97)$$

这个运算的计算代价是  $O(m)$ 。随着训练集规模增长为数十亿的样本，计算一步梯度也会消耗相当长的时间。

随机梯度下降的核心是，梯度是期望。期望可使用小规模的样本近似估计。具体而言，在算法的每一步，我们从训练集中均匀抽出一**minibatch** (minibatch) 样本  $\mathbb{B} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m')}\}$ 。minibatch的数目  $m'$  通常是一个相对较小的数，从一到几百。重要的是，当训练集大小  $m$  增长时， $m'$  通常是固定的。我们可能在拟合几十亿的样本时，每次更新计算只用到几百个样本。

梯度的估计可以表示成

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \quad (5.98)$$

使用来自minibatch  $\mathbb{B}$  的样本。然后，随机梯度下降算法使用如下的梯度下降估计：

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g}, \quad (5.99)$$

其中， $\epsilon$  是学习速率。

梯度下降往往被视为慢的或不可靠的。以前，将梯度下降应用到非凸优化问题被视为鲁莽的或无原则的。现在，我们知道梯度下降用于训练第二部分中工作效果很好。优化算法可能不能保证在合理的时间内达到一个局部最小值，但它通常能足够快地找到损失函数的一个可以用的非常低的值。

随机梯度下降在深度学习之外有很多重要的应用。它是在大规模数据上训练大型线性模型的主要方法。对于固定大小的模型，每一步随机梯度下降更新的计算量不取决于训练集的大小  $m$ 。在实践中，当训练集大小增长时，我们通常会使用一个更大的模型，但这并非强制的。达到收敛所需的更新次数通常会随训练集规模增大而增加。然而，当  $m$  趋向于无限大时，该模型最终会在随机梯度下降抽样训练集上的每个样本前收敛到可能的最优测试误差。继续增加  $m$  不会延长达到模型可能的最优测试误差的时间。从这点来看，我们可以认为用SGD训练模型的渐近代价是关于  $m$  的函数的  $O(1)$  级别。

在深度学习之前，学习非线性模型的主要方法是结合核策略的线性模型。很多核学习算法需要构建一个  $m \times m$  的矩阵  $G_{i,j} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ 。构建这个矩阵的计算量是  $O(m^2)$ 。当数据集是几十亿个样本时，这个计算量是不能接受的。在学术界，深度学习从 2006 年开始收到关注的原因是，在数以万计样本的中等规模数据集上，深度学习在新样本上比当时很多热门算法泛化得更好。不久后，深度学习在工业界受到了更多的关注，因为其提供了一种可扩展的方式训练大数据集上的非线性模型。

随机梯度下降及其很多强化方法将会在第八章继续探讨。

## 5.10 构建机器学习算法

几乎所有的深学习算法可以被描述为一个相当简单的配方：特定的数据集，损失函数，优化过程和模型。

例如，线性回归算法的组成成分有  $\mathbf{X}$  和  $\mathbf{y}$  构成的数据集，损失函数

$$J(\mathbf{w}, b) = -\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim p_{\text{data}}} \log p_{\text{model}}(\mathbf{y} \mid \mathbf{x}), \quad (5.100)$$

模型是  $p_{\text{model}}(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y}; \mathbf{x}^\top \mathbf{w} + b, 1)$ ，在大多数情况下，优化算法可以定义为求解损失函数梯度为零的正规方程。

意识到我们可以替换独立于其他成分的大多数成分，因此我们能得到很多不同

的算法。

损失函数通常包括使学习过程执行统计估计的至少一项。最常见的损失函数是负对数似然，最小化损失函数导致的最大似然估计。

损失函数也可能含有附加项，如正则化项。例如，我们可以将权重衰减加到线性回归的损失函数中

$$J(\mathbf{w}, b) = \lambda \|\mathbf{w}\|_2^2 - \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(y | \mathbf{x}). \quad (5.101)$$

该优化仍然有闭解。

如果我们将该模型变成非线性的，那么大多数损失函数不再有优化闭解。这就要求我们选择一个迭代数值优化过程，如梯度下降等。

组合模型，损失函数和优化算法来构建学习算法的配方同时适用于监督学习和无监督学习。线性回归实例说明了如何适用于监督学习的。无监督学习时，我们需要定义一个只包含  $\mathbf{X}$  的数据集，一个合适的无监督损失函数和一个模型。例如，通过指定如下损失函数可以得到PCA的第一个主向量

$$J(\mathbf{w}) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \|\mathbf{x} - r(\mathbf{x}; \mathbf{w})\|_2^2 \quad (5.102)$$

模型定义为重建函数  $r(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \mathbf{w}$ ，并且  $\mathbf{w}$  有范数为 1 的限制。

在某些情况下，由于计算原因，我们不能实际计算损失函数。在这种情况下，只要我们有近似其梯度的方法，那么我们仍然可以使用迭代数值优化近似最小化目标。

大多数学习算法都用到了上述配方，尽管有时候不显然。如果一个机器学习算法看上去特别独特或是手动设计的，那么通常使用特殊的优化方法进行求解。有些模型，如决策树或  $k$ -均值，需要特殊的优化，因为它们的损失函数有平坦的区域，使其不适合通过基于梯度的优化去最小化。认识到大部分机器学习算法可以使用上述配方描述，有助于将不同算法视为出于相同原因解决相关问题的一类方法，而不是一长串各个不同的算法。

## 5.11 深度学习的动机与挑战

本章描述的简单机器学习算法在很多不同的重要问题上都效果良好。但是他们不能成功解决人工智能中的核心问题，如语音识别或者对象识别。

深度学习发展动机的一部分原因是传统学习算法在这类人工智能问题上泛化能力不行。

本节介绍为何处理高维数据时在新样本上泛化特别困难，以及为何传统机器学习中实现泛化的机制不适合学习高维空间中复杂的函数。这些空间经常涉及巨大的计算代价。深度学习旨在克服这些，以及一些其他难题。

### 5.11.1 维数灾难

当数据的维数很高时，很多机器学习问题变得相当困难。这种现象被称为维数灾难 (curse of dimensionality)。特别值得注意的是，一组变量不同的可能配置数量会随着变量数目的增加而指数级增长。

维数灾难发生在计算机科学的许多地方，在机器学习中尤其如此。

由维数灾难带来的一个挑战是统计挑战。如图5.9所示，统计挑战产生于  $x$  的可能配置数目远大于训练样本的数目。为了充分理解这个问题，我们假设输入空间如图所示被分成网格。低维时我们可以用由数据占据的少量网格去描述这个空间。泛化到新数据点时，通过检测和新输入在相同网格中的训练样本，我们可以判断如何处理新数据点。例如，如果要估计某点  $x$  处的概率密度，我们可以返回  $x$  处单位体积内训练样本的数目除以训练样本的总数。如果我们希望对一个样本进行分类，我们可以返回相同网格中训练样本最多的类别。如果我们是做回归分析，我们可以平均该网格中样本对应的目标值。但是，如果该网格中没有样本，该怎么办呢？因为在高维空间中参数配置数目远大于样本数目，大部分配置没有相关的样本。我们如何能在这些新配置中找到一些有意义的东西？许多传统机器学习算法只是简单地假设在一个新点的输出应大致和最接近的训练点的输出相同。然而在高维空间中，这个假设是不够的。

### 5.11.2 局部不变性和平滑正则化

为了更好地泛化，机器学习算法需要由先验信念引导应该学习什么类型的函数。此前，我们已经看到过由模型参数的概率分布形成的先验。通俗地讲，我们也可以先验信念直接影响函数本身，而仅仅通过它们对函数的影响来间接改变参数。此外，我们还能通俗地说，先验信念还间接地体现在选择一些偏好某类函数的算法，尽管这些偏好并没有通过我们对不同函数置信程度的概率分布表现出来（也许根本没法

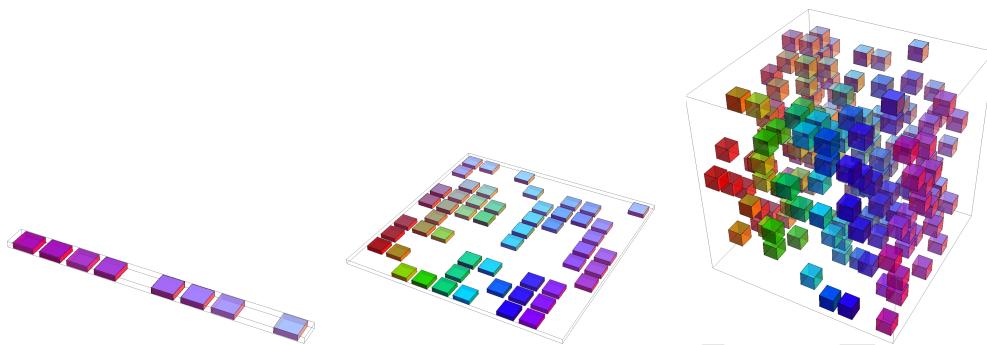


图 5.9：当数据的相关维度增大时（从左向右），我们感兴趣的 configuration 的数目会随之指数性地增长。（左）在这个一维的例子中，我们用一个变量来区分所感兴趣的仅仅 10 个区域。当每个区域都有足够的样本数时（图中每个样本对应了一个细胞），学习算法能够轻易地泛化得很好。泛化的一个直接方法是估计目标函数在每个区域的值（`<bad>` 可能是相邻区域的插值）。（中）在二维情况下，对每个变量区分 10 个不同的值更加困难。我们需要追踪  $10 \times 10 = 100$  个区域，至少需要很多样本来覆盖所有的区域。（右）三维情况下，区域数量增加到了  $10^3 = 1000$ ，至少需要更多的样本。对于需要区分的  $d$  维以及  $v$  个值来说，我们需要  $O(v^d)$  个区域和样本。这就是维数灾难的一个实例。感谢由 Nicolas Chapados 提供的图片。

表现)。

其中最广泛使用的隐式“先验”是平滑先验 (smoothness prior)，或局部不变性先验 (local constancy prior)。这个先验表明我们学习的函数不应在小区域内发生很大的变化。

许多简单算法完全依赖于此先验达到良好的泛化，其结果是不能推广去解决人工智能级别任务中的统计挑战。本书中，我们将介绍深度学习如何引入额外的（显示的和隐式的）先验去降低复杂任务中的泛化误差。这里，我们解释为什么单是平滑先验不足以应对这类任务。

有许多不同的方法来隐式地或显式地表示学习函数应该是光滑或局部不变的先验。所有这些不同的方法都旨在鼓励学习过程能够学习出函数  $f^*$  对于大多数设置  $x$  和小变动  $\epsilon$ ，都满足条件

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \boldsymbol{\epsilon}). \quad (5.103)$$

换言之，如果我们知道对应输入  $\mathbf{x}$  的答案（例如， $\mathbf{x}$  是个有标签的训练样本），那么该答案对于  $\mathbf{x}$  的邻域应该也适用。如果在有些邻域中我们有几个好答案，那么我们可以组合它们（通过某种形式的平均或插值法）以产生一个尽可能和大多数输入一

致的答案。

局部不变方法的一个极端例子是  $k$  最近邻系列的学习算法。当一个区域里的所有点  $\mathbf{x}$  在训练集中的  $k$  个最近邻是一样的，那么对这些点的预测也是一样的。当  $k = 1$  时，不同区域的数目不会比训练样本还多。

虽然  $k$ -最近邻算法复制了附近训练样本的输出，大部分核机器也是在和附近训练样本相关的训练集输出上插值。一类重要的核函数是局部核 (local kernel)，其核函数  $k(\mathbf{u}, \mathbf{v})$  在  $\mathbf{u} = \mathbf{v}$  时很大，当  $\mathbf{u}$  和  $\mathbf{v}$  距离拉大时而减小。局部核可以看作是执行模版匹配的相似函数，用于度量测试样本  $\mathbf{x}$  和每个训练样本  $\mathbf{x}^{(i)}$  有多么相似。近年来深度学习的很多推动力源自研究局部模版匹配的局限性，以及深度学习如何克服这些局限性 (Bengio *et al.*, 2006a)。

决策树也有平滑学习的局限性，因为它将输入空间分成和叶节点一样多的区间，并在每个区间使用单独的参数（或者有些决策树的拓展有多个参数）。如果目标函数需要至少拥有  $n$  个叶节点的树才能精确表示，那么至少需要  $n$  个训练样本去拟合。需要几倍于  $n$  的样本去达到预测输出上的某种统计置信度。

总的来说，区分输入空间中  $O(k)$  个区间，所有的这些方法需要  $O(k)$  个样本。通常会有  $O(k)$  个参数， $O(1)$  参数对应于  $O(k)$  区间之一。最近邻算法中，每个训练样本至多用于定义一个区间，如图5.10所示。

有没有什么方法能表示区间数目比训练样本数目还多的复杂函数？显然，只是假设函数的平滑性不能做到这点。例如，想象目标函数作用在西洋跳棋盘上。棋盘包含许多变化，但只有一个简单的结构。想象一下，如果训练样本数目远小于棋盘上的黑白方块数目，那么会发生什么。基于局部泛化和平滑性或局部不变性先验，如果新点和某个训练样本位于相同的棋盘方块中，那么我们能够保证正确地预测新点的颜色。但如果新点所在的方块没有训练样本，学习器不一定能举一反三。如果仅依靠这个先验，一个样本只能告诉我们它所在的方块的颜色。获得整个棋盘颜色的唯一方法是其上的每个方块至少要有一个样本。

只要在要学习的真实函数的峰值和谷值处有足够的样本，那么平滑性假设和相关的无参数学习算法的效果都非常好。当要学习的函数足够平滑，并且只在少数几维变动，这样做一般没问题。在高维空间中，即使是非常平滑的函数，也会在不同维度上有不同的平滑变动程度。如果函数在不同的区间中表现不一样，那么就非常难用一组训练样本去刻画函数。如果函数是复杂的（我们想区分多于训练样本数目的大量区间），有希望很好地泛化么？

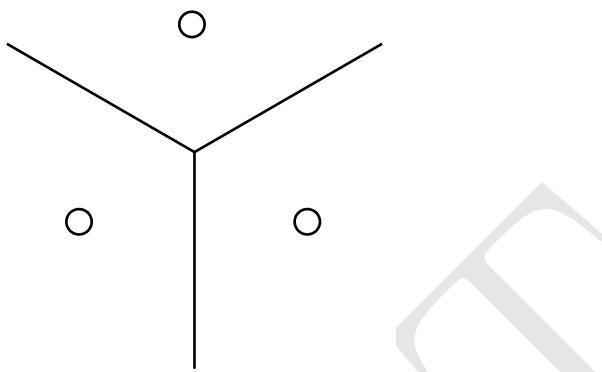


图 5.10: 最近邻算法如何将输入空间分区域的一个例子。每个区域内的一个样本（这里用圆圈表示）定义了区域边界（这里用线表示）。每个样本相关的  $y$  值定义了对应区域内所有数据点的输出。由最近邻定义并且匹配几何模式的区域被叫做 Voronoi 图。这些连续区域的数量不能比训练样本的数量增加的更快。尽管此图具体说明了最近邻算法的表现，其他的单纯依赖局部光滑先验的机器学习算法也表现出了类似的泛化能力：每个训练样本仅仅能告诉学习者如何在其周围的相邻区域泛化。

回答这些问题——是否可以有效地表示复杂的函数，以及所估计的函数是否可以很好地泛化到新的输入——答案是有。关键观点是，只要我们通过额外假设生成数据的分布来建立区域间的依赖关系，那么  $O(k)$  个样本足以描述多如  $O(2^k)$  的大量区间。通过这种方式，我们确实能做到非局部的泛化 (Bengio and Monperrus, 2005; Bengio *et al.*, 2006b)。为了利用这些优势，许多不同的深度学习算法都提出了一些适用于多种AI任务的或隐或显的假设。

许多不同的深度学习算法提出隐式或显式的适用于大范围人工智能问题的合理假设，使其可以利用这些优势。

一些其他的机器学习方法往往会提出更强的，针对特定问题的假设。例如，假设目标函数是周期性的，我们很容易解决棋盘问题。通常，神经网络不会包含这些很强的，针对特定任务的假设，因此神经网络可以泛化到更广泛的各种结构中。人工智能任务的结构非常复杂，很难限制到简单的，人工手动指定的性质，如周期性，因此我们希望学习算法具有更通用的假设。深度学习的核心思想是假设数据由因素或特征组合产生，这些因素或特征可能来自一个层次结构的多个层级。许多其他类似的通用假设进一步提高了深度学习算法。这些很温和的假设允许了样本数目和可区分区间数目之间的指数增益。这类指数增益将在第6.4.1节，第15.4节和第15.5节中被

更详尽地介绍。深度的分布式表示带来的指数增益有效解决了维数灾难带来的挑战。

### 5.11.3 流形学习

流形是一个机器学习中很多想法内在的重要概念。

流形 (manifold) 指连接在一起的区域。数学上，它是指一组点，且每个点都有其邻域。给定一个任意的点，其流形局部看起来像是欧几里得空间。日常生活中，我们将地球视为二维平面，但实际上它是三维空间中的球状流形。

每个点周围邻域的定义暗示着存在变换能够从一个位置移动到其邻域位置。例如在地球表面这个流形中，我们可以朝东南西北走。

尽管术语“流形”有正式的数学定义，但是机器学习倾向于更松散地定义一组点，只需要考虑少数嵌入在高维空间中的自由度或维数就能很好地近似。每一维都对应着局部的变动方向。如图5.11所示，训练数据位于二维空间中的一维流形中。在机器学习中，我们允许流形的维数从一个点到另一个点有所变化。这经常发生于流形和自身相交的情况下。例如，数字“8”形状的流形在大多数位置只有一维，但在中心的相交处有两维。

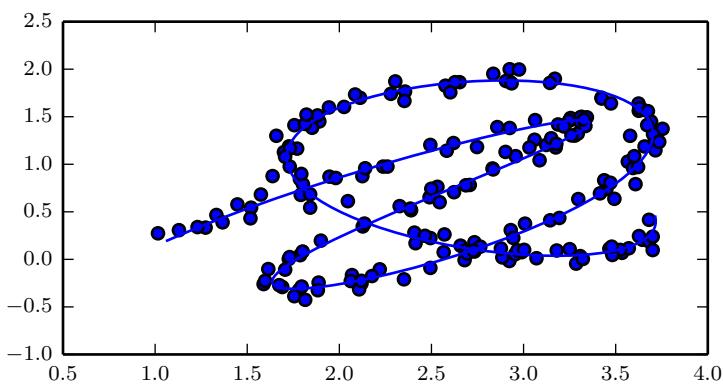


图 5.11：从一个聚集在一维流形的二维空间的分布中抽取的数据样本，像一个缠绕的带子一样。实线代表了学习者想要推断的隐含的流形。

如果我们希望机器学习算法学习  $\mathbb{R}^n$  上的所有感兴趣的函数，那么很多机器学习问题看上去都是不可解的。流形学习 (manifold learning) 算法通过一个假设来克服这个障碍，该假设认为  $\mathbb{R}^n$  中大部分区域都是无效的输入，感兴趣的输入只分布

在包含少量点的子集构成的一组流形中，而学习函数中感兴趣的变动只位于流形中的方向，或者感兴趣的变动只发生在我们从一个流形移动到另一个流形的时候。流形学习是在连续数值数据和无监督学习的设定下被引入的，尽管这个概率集中的想法也能够泛化到离散数据和监督学习的设定下：关键假设仍然是概率质量高度集中。

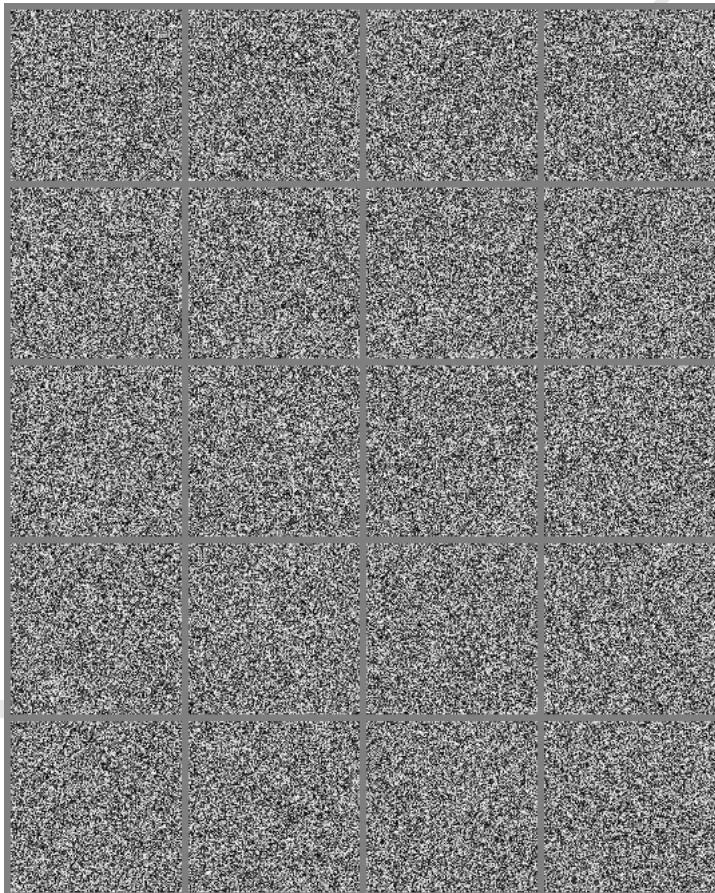


图 5.12: 随机地均匀抽取图像（根据均匀分布随机地选择每一个像素）会得到噪音图像。尽管在人工智能应用中生成一个脸或者其他物体的图像是非零概率的，但是实践中我们从来没有观察到这个现象。这也意味着人工智能应用中遇到的图像在所有图像空间中的占比是忽略不计的。

数据位于低维流形的假设并不总是对的或者有用的。我们认为在人工智能的一些场景中，如涉及到处理图像，声音或者文本，流形假设至少是近似对的。这个假设的支持证据包含两类观察结果。

第一个支持流形假设 (manifold hypothesis) 的观察是现实生活中的图像，文本，声音的概率分布都是高度集中的。均匀的噪扰从来没有和这类领域的结构化输入相似过。图5.12显示均匀采样的点看上去像是没有信号时模拟电视上的静态模式。同样，如果我们均匀地随机抽取字母来生成文件，能有多大的概率得到一个有意义的英语文档？几乎是零。因为大部分字母长序列不对应着自然语言序列：自然语言序列的分布只占了字母序列的总空间里非常小的一部分。

当然，集中的概率分布不足以说明数据位于一个相当小的流形中。我们还必须确定，我们遇到的样本和其他样本相互连接，每个样本被其他高度相似的样本包围，可以通过变换来遍历该流形。支持流形假设的第二个论点是，我们至少能够非正式地想象这些邻域和变换。在图像中，我们当然会认为有很多可能的变换允许我们描绘出图片空间的流形：我们可以逐渐变暗或变亮光泽，逐步移动或旋转图中对象，逐渐改变对象表面的颜色，等等。在大多数应用中很有可能会涉及多个流形。例如，人脸图像的流形不太可能连接到猫脸图像的流形。

这些支持流形假设的思维试验传递了一些支持它的直观理由。更严格的实验 (Cayton, 2005; Narayanan and Mitter, 2010; Schölkopf *et al.*, 1998a; Roweis and Saul, 2000; Tenenbaum *et al.*, 2000; Brand, 2003a; Belkin and Niyogi, 2003b; Donoho and Grimes, 2003; Weinberger and Saul, 2004a) 在人工智能中受关注的一大类数据集上支持了这个假设。

当数据位于低维流形中时，使用流形中的坐标，而非  $\mathbb{R}^n$  中的坐标表示机器学习数据更为自然。日常生活中，我们可以认为道路是嵌入在三维空间的一维流形。我们用一维道路中的地址号码确定地址，而非三维空间中的坐标。提取这些流形中的坐标是非常具有挑战性的，但是很有希望改进许多机器学习算法。这个一般性原则能够用在很多情况下。图5.13展示了包含脸的数据集的流形结构。在本书的最后，我们会介绍一些学习这样的流形结构的必备方法。在图20.6中，我们将看到机器学习算法如何成功完成这个目标。

第一部分介绍了数学和机器学习中的基本概念，这将用于本书其他章节中。至此，我们已经完成了开始学习深度学习的准备。



图 5.13: QMUL Multiview Face 数据集中训练样本 (Gong *et al.*, 2000), 其中的物体是移动的从而覆盖对应两个旋转角度的二维流形。我们希望学习算法能够发现并且解决这些流形坐标。图20.6提供了这样一个例子。

## 第二部分

### 深层网络：现代实践

本书这一部分总结现代深度学习用于解决实际应用的现状。

深度学习有着悠久的历史和许多愿景。数种提出的方法尚未完全结出果实。数个雄心勃勃的目标尚未实现。这些较不发达的深度学习分支将出现在本书的最后部分。

这一部分仅关注那些基本上已在工业中大量使用的技术方法。

现代深度学习为监督学习提供了一个强大的框架。通过添加更多层以及向层内添加更多单元，深度网络可以表示复杂性不断增加的函数。给定足够大的模型和足够大的标注训练数据集，我们可以通过深度学习将输入向量映射到输出向量，完成大多数对人来说能迅速处理的任务。其他任务，比如不能被描述为将一个向量与另一个相关联的任务，或者对于一个人来说足够困难并需要时间思考和反复琢磨才能完成的任务，现在仍然超出了深度学习的能力范围。

本书这一部分描述参数化函数近似技术的核心，几乎所有现代实际应用的深度学习背后都用到了这一技术。首先，我们描述用于表示这些函数的前馈深度网络模型。接着，我们提出正则化和优化这种模型的高级技术。将这些模型扩展到大输入（如高分辨率图像或长时间序列）需要专门化。我们将会介绍扩展到大图像的卷积网络和用于处理时间序列的循环神经网络。最后，我们提出实用方法的一般准则，有助于设计、构建和配置一些涉及深度学习的应用，并回顾其中一些应用。

这些章节对于从业者来说是最重要的，也就是现在想开始实现和使用深度学习算法解决现实问题的人需要阅读这些章节。

# 第六章 深度前馈网络

深度前馈网络 (deep feedforward network)，也叫作前馈神经网络 (feedforward neural network) 或者多层感知机 (multilayer perceptron, MLP)，是典型的深度学习模型。前馈网络的目标是近似某个函数  $f^*$ 。例如，对于分类器， $y = f^*(\mathbf{x})$  将输入  $\mathbf{x}$  映射到一个类别  $y$ 。前馈网络定义了一个映射  $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ ，并且学习参数  $\boldsymbol{\theta}$  的值，使它能够得到最佳的函数近似。

这种模型被称为前向 (feedforward) 的，是因为信息流过  $\mathbf{x}$  的函数，流经用于定义  $f$  的中间计算过程，最终到达输出  $\mathbf{y}$ 。在模型的输出和模型本身之间没有反馈 (feedback) 连接。当前馈神经网络被扩展成包含反馈连接时，它们被称为循环神经网络 (recurrent neural network)，在第十章介绍。

前馈网络对于机器学习的从业者是极其重要的。它们是许多重要商业应用的基础。例如，用于对照片中的对象进行识别的卷积神经网络就是一种专门的前馈网络。前馈网络是向循环网络前进中的概念奠基石，后者在自然语言的许多应用中发挥着巨大作用。

前馈神经网络被称作网络 (network) 是因为它们通常用许多不同函数复合在一起表示。该模型与一个有向无环图相关联，而图描述了函数是如何复合在一起的。例如，我们有三个函数  $f^{(1)}$ ,  $f^{(2)}$  和  $f^{(3)}$  连接在一个链上以形成  $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ 。这些链式结构是神经网络中最常用的结构。在这种情况下， $f^{(1)}$  被称为网络的第一层 (first layer)， $f^{(2)}$  被称为第二层 (second layer)，以此类推。链的全长称为模型的深度 (depth)。正是因为这个术语才出现了“深度学习”这个名字。前馈网络的最后一层被称为输出层 (output layer)。在神经网络训练的过程中，我们让  $f(\mathbf{x})$  去匹配  $f^*(\mathbf{x})$  的值。训练数据为我们提供了在不同训练点上取值的、含有噪声的  $f^*(\mathbf{x})$  的近似实例。每个样例  $\mathbf{x}$  都伴随着一个标签  $y \approx f^*(\mathbf{x})$ 。训练样例直接指明了输出层在每一点  $\mathbf{x}$  上必须做什么；它必须产生一个接近  $y$  的值。

但是训练数据并没有直接指明其他层应该怎么做。学习算法必须决定如何使用这些层来产生想要的输出，但是训练数据并没有说每个单独的层应该做什么。相反，学习算法必须决定如何使用这些层来最好地实现  $f^*$  的近似。因为训练数据并没有给出这些层中的每一层所需的输出，所以这些层被称为隐藏层 (hidden layer)。

最后，这些网络被称为神经网络是因为它们或多或少地受到神经科学的启发。网络中的每个隐藏层通常都是向量值的。这些隐藏层的维数决定了模型的宽度 (width)。向量的每个元素都可以被视为起到类似一个神经元的作用。除了将层想象成向量到向量的单个函数，我们也可以把层想象成由许多并行操作的单元 (unit) 组成，每个单元表示一个向量到标量的函数。每个单元在某种意义上类似一个神经元，它接收的输入来源于许多其他的单元，并且计算它自己的激活值。使用多层向量值表示的想法来源于神经科学。用于计算这些表示的函数  $f^{(i)}(\mathbf{x})$  的选择，也或多或少地受到神经科学观测的指引，这些观测是关于生物神经元计算功能的。然而，现代的神经网络研究受到更多的是来自许多数学和工程学科的指引，并且神经网络的目标并不是完美地给大脑建模。我们最好将前馈神经网络想成是为了实现统计泛化而设计出的函数近似机器，它偶尔从我们了解的大脑中提取灵感但是并不是大脑功能的模型。

一种理解前馈网络的方式是从线性模型开始，并考虑如何克服它的局限性。线性模型，例如逻辑回归和线性回归，是非常吸引人的，因为无论是封闭形式还是使用凸优化，它们都能高效而可靠地拟合。线性模型也有明显的缺陷，那就是该模型的能力被局限在线性函数里，所以它无法理解任何两个输入变量间的相互作用。

为了扩展线性模型来表示  $\mathbf{x}$  的非线性函数，我们可以把线性模型不用在  $\mathbf{x}$  本身，而是用在一个变换后的输入  $\phi(\mathbf{x})$  上，这里  $\phi$  是一个非线性变换。等价地，我们可以使用5.7.2节中描述的核技巧，来得到一个基于隐含地使用  $\phi$  映射的非线性学习算法。我们可以认为  $\phi$  提供了一组描述  $\mathbf{x}$  的特征，或者认为它提供了  $\mathbf{x}$  的一个新的表示。

剩下的问题就是如何选择映射  $\phi$ 。

1. 其中一种选择是使用一个通用的  $\phi$ ，例如无限维的  $\phi$ ，它隐含地用在基于RBF核的核机器上。如果  $\phi(\mathbf{x})$  具有足够高的维数，我们总是有足够的能力来拟合训练集，但是对于测试集的泛化往往不佳。非常通用的特征映射通常只基于局部平滑的原则，并没有将足够的先验信息进行编码来解决高级问题。
2. 另一种选择是手动地设计  $\phi$ 。在深度学习出现以前，这都是主流的方法。这种方法对于每个单独的任务都需要人们数十年的努力，其中包括不同领域的（如

语音识别或计算机视觉)专家以及不同领域间微小的迁移(transfer)。

3. 深度学习的策略是去学习  $\phi$ 。在这种方法中，我们有一个模型  $y = f(\mathbf{x}; \theta, \mathbf{w}) = \phi(\mathbf{x}; \theta)^\top \mathbf{w}$ 。我们现在有两种参数：用于从一大类函数中学习  $\phi$  的参数  $\theta$ ，以及用于将  $\phi(\mathbf{x})$  映射到所需的输出的参数  $\mathbf{w}$ 。这是深度前馈网络的一个例子，其中  $\phi$  定义了一个隐藏层。这是三种方法中唯一一种放弃训练问题的凸性的方法，但是利大于弊。在这种方法中，我们将表示参数化为  $\phi(\mathbf{x}; \theta)$ ，并且使用优化算法来寻找  $\theta$ ，使它能够得到一个好的表示。如果我们希望，这种方法也可以通过使它变得高度通用以获得第一种方法的优点——我们只需使用一个非常广泛的函数族  $\phi(\mathbf{x}; \theta)$ 。这种方法也可以获得第二种方法的优点。人类专家可以将他们的知识编码进网络来帮助泛化，他们只需要设计那些他们期望能够表现优异的函数族  $\phi(\mathbf{x}; \theta)$  即可。这种方法的优点是人类设计者只需要寻找正确的函数族即可，而不需要去寻找精确的函数。

这种通过学习特征来改善模型的一般化原则不止适用于本章描述的前馈神经网络。它是深度学习中反复出现的主题，适用于全书描述的所有种类的模型。前馈神经网络是这个原则的应用，它学习从  $\mathbf{x}$  到  $\mathbf{y}$  的确定性映射并且没有反馈连接。后面出现的其他模型会把这些原则应用到学习随机映射、学习带有反馈的函数以及学习单个向量的概率分布。

本章我们先从前馈网络的一个简单例子说起。接着，我们讨论部署一个前馈网络所需的每个设计决策。首先，训练一个前馈网络至少需要做和线性模型同样多的设计决策：选择一个优化模型，代价函数以及输出单元的形式。我们先回顾这些基于梯度学习的基本知识，然后去面对那些只出现在前馈网络中的设计决策。前馈网络已经引入了隐藏层的概念，这需要我们去选择用于计算隐藏层值的激活函数(activation function)。我们还必须设计网络的结构，包括网络应该包含多少层、这些层应该如何连接，以及每一层包含多少单元。在深度神经网络的学习中需要计算复杂函数的梯度。我们给出反向传播(back propagation)算法和它的现代推广，它们可以用来高效地计算这些梯度。最后，我们以某些历史观点来结束这一章。

## 6.1 实例：学习 XOR

为了使前馈网络的想法更加具体，我们首先从前馈网络充分发挥作用的一个简单例子说起：学习 XOR 函数。

XOR 函数（“异或”逻辑）是两个二进制值  $x_1$  和  $x_2$  的运算。当这些二进制值中恰好有一个为 1 时，XOR 函数返回值为 1。其余情况下返回值为 0。XOR 函数提供了我们想要学习的目标函数  $y = f^*(\mathbf{x})$ 。我们的模型给出了一个函数  $y = f(\mathbf{x}; \boldsymbol{\theta})$  并且我们的学习算法会不断调整参数  $\boldsymbol{\theta}$  来使得  $f$  尽可能接近  $f^*$ 。

在这个简单的例子中，我们不会关心统计泛化。我们希望网络在这四个点  $\mathbb{X} = \{[0, 0]^\top, [0, 1]^\top, [1, 0]^\top, [1, 1]^\top\}$  上表现正确。我们会用全部这四个点来训练我们的网络，唯一的挑战是拟合训练集。

我们可以把这个问题当作是回归问题，并使用均方误差损失函数。我们选择这个损失函数是为了尽可能简化本例中用到的数学。在应用领域，对于二进制数据建模时，MSE 通常并不是一个合适的损失函数。更加合适的方法将在 6.2.2.2 节中讨论。

为了对在整个训练集上的表现进行评估，MSE 损失函数为

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\mathbf{x} \in \mathbb{X}} (f^*(\mathbf{x}) - f(\mathbf{x}; \boldsymbol{\theta}))^2. \quad (6.1)$$

我们现在必须要选择我们模型  $f(\mathbf{x}; \boldsymbol{\theta})$  的形式。假设我们选择一个线性模型， $\boldsymbol{\theta}$  包含  $\mathbf{w}$  和  $b$ ，那么我们的模型被定义成

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^\top \mathbf{w} + b. \quad (6.2)$$

我们可以用正规方程对  $\mathbf{w}$  和  $b$  最小化  $J(\boldsymbol{\theta})$ ，来得到一个封闭形式的解。

解正规方程以后，我们得到  $\mathbf{w} = 0$  以及  $b = \frac{1}{2}$ 。线性模型仅仅是在任意一点都输出 0.5。为什么会发生这种事？图 6.1 演示了线性模型为什么不能用来表示 XOR 函数。解决这个问题的其中一种方法是使用一个模型来学习一个不同的特征空间，在这个空间上线性模型能够表示这个解。

具体来说，我们这里引入一个非常简单的前馈神经网络，它有一层隐藏层并且隐藏层中包含两个单元。参见图 6.2 中对该模型的解释。这个前馈网络有一个通过函数  $f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c})$  计算得到的隐藏单元的向量  $\mathbf{h}$ 。这些隐藏单元的值随后被用作第二层的输入。第二层就是这个网络的输出层。输出层仍然只是一个线性回归模型，只不过现在它作用于  $\mathbf{h}$  而不是  $\mathbf{x}$ 。网络现在包含链接在一起的两个函数： $\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c})$  和  $y = f^{(2)}(\mathbf{h}; \mathbf{w}, b)$ ，完整的模型是  $f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = f^{(2)}(f^{(1)}(\mathbf{x}))$ 。

$f^{(1)}$  应该是哪种函数？线性模型到目前为止都表现不错，让  $f^{(1)}$  也是线性的似乎很有诱惑力。不幸的是，如果  $f^{(1)}$  是线性的，那么前馈网络作为一个整体对于输入

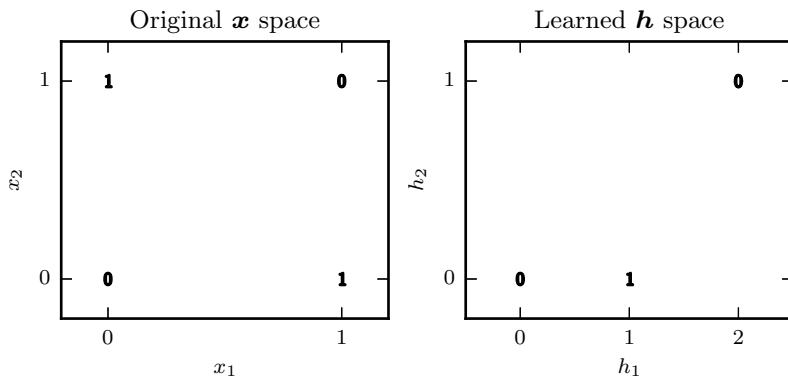


图 6.1: 通过学习一个表示来解决 XOR 问题。图上的粗体数字标明了学得的函数必须在每个点输出的值。(左) 直接应用于原始输入的线性模型不能实现 XOR 函数。当  $x_1 = 0$  时, 模型的输出必须随着  $x_2$  的增大而增大。当  $x_1 = 1$  时, 模型的输出必须随着  $x_2$  的增大而减小。线性模型必须对  $x_2$  使用固定的系数  $w_2$ 。因此, 线性模型不能使用  $x_1$  的值来改变  $x_2$  的系数, 从而不能解决这个问题。(右) 在由神经网络提取的特征表示的变换空间中, 线性模型现在可以解决这个问题了。在我们的示例解决方案中, 输出必须为 1 的两个点折叠到了特征空间中的单个点。换句话说, 非线性特征将  $\mathbf{x} = [1, 0]^\top$  和  $\mathbf{x} = [0, 1]^\top$  都映射到了特征空间中的单个点  $\mathbf{h} = [1, 0]^\top$ 。线性模型现在可以将函数描述为  $h_1$  增大和  $h_2$  减小。在该示例中, 学习特征空间的动机仅仅是使得模型的能力更大, 使得它可以拟合训练集。在更现实的应用中, 学习的表示也可以帮助模型泛化。

仍然是线性的。暂时忽略截距项, 假设  $f^{(1)}(\mathbf{x}) = \mathbf{W}^\top \mathbf{x}$  并且  $f^{(2)}(\mathbf{h}) = \mathbf{h}^\top \mathbf{w}$ , 那么  $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{W}^\top \mathbf{x}$ 。我们可以将这个函数重新表示成  $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}'$  其中  $\mathbf{w}' = \mathbf{W}\mathbf{w}$ 。

显然, 我们必须用非线性函数来描述这些特征。大多数神经网络通过仿射变换之后紧跟着一个被称为激活函数的固定非线性函数来实现这个目标, 其中仿射变换由学得的参数控制。我们这里使用这种策略, 定义  $\mathbf{h} = g(\mathbf{W}^\top \mathbf{x} + \mathbf{c})$ , 其中  $\mathbf{W}$  是线性变换的权重矩阵,  $\mathbf{c}$  是偏置。先前, 为了描述线性回归模型, 我们使用权重向量和一个标量的偏置参数来描述从输入向量到输出标量的仿射变换。现在, 因为我们描述的是向量  $\mathbf{x}$  到向量  $\mathbf{h}$  的仿射变换, 所以我们需要一整个向量的偏置参数。激活函数  $g$  通常选择对每个元素分别起作用的函数, 有  $h_i = g(\mathbf{x}^\top \mathbf{W}_{:,i} + c_i)$ 。在现代神经网络中, 默认的推荐是使用由激活函数  $g(z) = \max\{0, z\}$  定义的整流线性单元 (rectified linear unit) 或者称为ReLU(Jarrett et al., 2009b; Nair and Hinton, 2010a; Glorot et al., 2011a), 如图6.3所示。

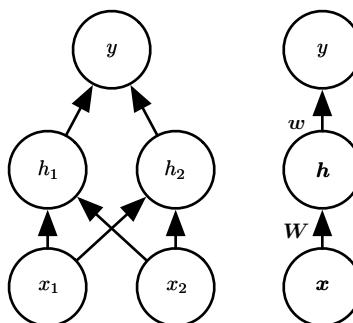


图 6.2: 使用两种不同样式的前馈网络的示例。具体来说，这是我们用来解决 XOR 问题的前馈网络。它有单个隐藏层，包含两个单元。(左) 在这种样式中，我们将每个单元绘制为图中的一个节点。这种风格是清楚而明确的，但对于比这个例子更大的网络，它可能会消耗太多的空间。(右) 在这种样式中，我们将表示每一层激活的整个向量绘制为图中的一个节点。这种样式更加紧凑。有时，我们对图中的边使用参数名进行注释，这些参数是用来描述两层之间的关系的。这里，我们用矩阵  $W$  描述从  $x$  到  $h$  的映射，用向量  $w$  描述从  $h$  到  $y$  的映射。当标记这种图时，我们通常省略与每个层相关联的截距参数。

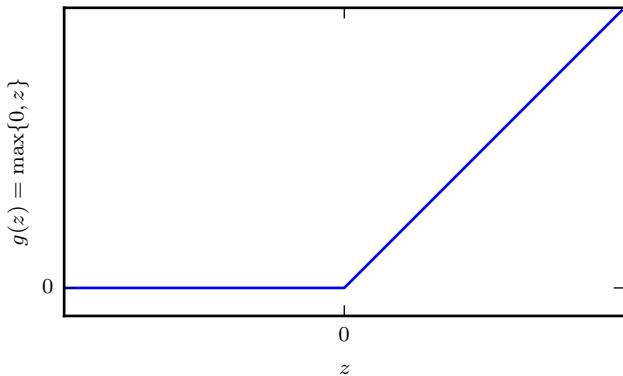


图 6.3: 整流线性激活函数。该激活函数是被推荐用于大多数前馈神经网络的默认激活函数。将此函数用于线性变换的输出将产生非线性变换。然而，函数仍然非常接近线性，在这种意义上它是具有两个线性部分的分段线性函数。由于整流线性单元几乎是线性的，因此它们保留了许多使得线性模型易于使用基于梯度的方法进行优化的属性。它们还保留了许多使得线性模型能够泛化良好的属性。计算机科学的一个公共原则是，我们可以从最小的组件构建复杂的系统。就像图灵机的内存只需要能够存储 0 或 1 的状态，我们可以从整流线性函数构建一个通用函数逼近器。

我们现在可以指明我们的整个网络是

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^\top \max\{0, \mathbf{W}^\top \mathbf{x} + \mathbf{c}\} + b. \quad (6.3)$$

我们现在可以给出 XOR 问题的一个解。令

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad (6.4)$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad (6.5)$$

$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad (6.6)$$

以及  $b = 0$ 。

我们现在可以了解这个模型如何处理一批输入。令  $\mathbf{X}$  表示设计矩阵，它包含二进制输入空间中全部的四个点，每个样例占一行，那么矩阵表示为：

$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}. \quad (6.7)$$

神经网络的第一步是将输入矩阵乘以第一层的权重矩阵：

$$\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}. \quad (6.8)$$

然后，我们加上偏置向量  $\mathbf{c}$ ，得到

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \quad (6.9)$$

在这个空间中，所有的样例都处在一条斜率为 1 的直线上。当我们沿着这条直线移动时，输出需要从 0 升到 1，然后再降回 0。线性模型不能实现这样一种函数。为了

用  $h$  对每个样例求值，我们使用整流线性变换：

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}. \quad (6.10)$$

这个变换改变了样例间的关系。它们不再处于同一条直线上了。如图6.3所示，它们现在处在一个可以用线性模型解决的空间上。

我们最后乘以一个权重向量  $w$ ：

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}. \quad (6.11)$$

神经网络对这一批次中的每个样例都给出了正确的结果。

在这个例子中，我们简单地指明了答案，然后说明它得到的误差为零。在实际情况中，可能会有数十亿的模型参数以及数十亿的训练样本，所以不能像我们这里做的那样进行简单地猜解。与之相对的，基于梯度的优化算法可以找到一些参数使得产生的误差非常小。我们这里给出的 XOR 问题的解处在损失函数的全局最小点，所以梯度下降算法可以收敛到这一点。梯度下降算法还可以找到 XOR 问题一些其他的等价解。梯度下降算法的收敛点取决于参数的初始值。在实践中，梯度下降通常不会找到像我们这里给出的那种干净的、容易理解的、整数值的解。

## 6.2 基于梯度的学习

设计和训练神经网络与使用梯度下降训练其他任何机器学习模型并没有太大不同。在5.10节中，我们描述了如何通过指明一个优化过程、代价函数和一个模型族来构建一个机器学习算法。

我们到目前为止看到的线性模型和神经网络的最大区别，在于神经网络的非线性导致大多数我们感兴趣的损失函数都成为了非凸的。这意味着神经网络的训练通常使用的迭代的、基于梯度的优化，仅仅使得代价函数达到一个非常小的值；而不是像用于训练线性回归模型的线性方程求解器，或者用于训练逻辑回归或SVM的凸优

化算法那样具有全局的收敛保证。凸优化从任何一种初始参数出发都会收敛（理论上如此——在实践中也很鲁棒但可能会遇到数值问题）。用于非凸损失函数的随机梯度下降没有这种收敛性保证，并且对参数的初始值很敏感。对于前馈神经网络，将所有的权重值初始化为小随机数是很重要的。偏置可以初始化为零或者小的正值。这种用于训练前馈神经网络以及几乎所有深度模型的迭代的基于梯度的优化算法会在第八章详细介绍，参数初始化会在8.4节中具体说明。就目前而言，只需要懂得，训练算法几乎总是基于使用梯度来使得代价函数下降的各种方法即可。一些特别的算法是对梯度下降思想的改进和提纯，在4.3节中介绍，还有一些更特别的，大多数是对随机梯度下降算法的改进，在5.9节中介绍。

我们当然也可以用梯度下降来训练诸如线性回归和支持向量机之类的模型，并且事实上当训练集相当大时这是很常用的。从这点来看，训练神经网络和训练其他任何模型并没有太大区别。计算梯度对于神经网络会略微复杂一些，但仍然可以很高效而精确地实现。6.5节将会介绍如何用反向传播算法以及它的现代扩展算法来求得梯度。

和其他的机器学习模型一样，为了使用基于梯度的学习方法我们必须选择一个代价函数，并且我们必须选择如何表示模型的输出。现在，我们重温这些设计上的考虑，并且特别强调神经网络的情景。

### 6.2.1 代价函数

深度神经网络设计中的一个重要方面是代价函数的选择。幸运的是，神经网络的代价函数或多或少是和其他的参数模型例如线性模型的代价函数相同的。

在大多数情况下，我们的参数模型定义了一个分布  $p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta})$  并且我们简单地使用最大似然原理。这意味着我们使用训练数据和模型预测间的交叉熵作为代价函数。

有时，我们使用一个更简单的方法，不是预测  $\mathbf{y}$  的完整概率分布，而是仅仅预测在给定  $\mathbf{x}$  的条件下  $\mathbf{y}$  的某种统计量。某些专门的损失函数允许我们来训练这些估计量的预测器。

用于训练神经网络的完整的代价函数，通常在我们这里描述的基本代价函数的基础上结合一个正则项。我们已经在5.2.2节中看到正则化应用到线性模型中的一些简单的例子。用于线性模型的权值衰减方法也直接适用于深度神经网络，而且是最

流行的正则化策略之一。用于神经网络的更高级的正则化策略会在第七章中讨论。

### 6.2.1.1 用最大似然学习条件分布

大多数现代的神经网络使用最大似然来训练。这意味着代价函数就是负的对数似然，它与训练数据和模型分布间的交叉熵等价。这个代价函数表示为

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{y} | \mathbf{x}). \quad (6.12)$$

代价函数的具体形式随着模型而改变，取决于  $\log p_{\text{model}}$  的具体形式。上述方程的展开形式通常会有一些项不依赖于模型的参数，我们可以舍去。例如，正如我们在5.1.1节中看到的，如果  $p_{\text{model}}(\mathbf{y} | \mathbf{x}) = \mathcal{N}(\mathbf{y}; f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{I})$ ，那么我们恢复均方误差代价，

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} \|\mathbf{y} - f(\mathbf{x}; \boldsymbol{\theta})\|^2 + \text{const}, \quad (6.13)$$

至少系数  $\frac{1}{2}$  和常数项不依赖于  $\boldsymbol{\theta}$ 。舍弃的常数是基于高斯分布的方差，在这种情况下我们选择不把它参数化。先前，我们看到了对输出分布的最大似然估计和对线性模型均方误差的最小化之间的等价性，但事实上，这种等价性并不要求  $f(\mathbf{x}; \boldsymbol{\theta})$  用于预测高斯分布的均值。

使用最大似然来导出代价函数的方法的一个优势是，它减轻了为每个模型设计代价函数的负担。明确一个模型  $p(\mathbf{y} | \mathbf{x})$  则自动地确定了一个代价函数  $\log p(\mathbf{y} | \mathbf{x})$ 。

贯穿神经网络设计的一个反复出现的主题是代价函数的梯度必须足够的大和具有足够的预测性，来为学习算法提供一个好的指引。饱和（变得非常平）的函数破坏了这一目标，因为它们把梯度变得非常小。这在很多情况下都会发生，因为用于产生隐藏单元或者输出单元的输出的激活函数会饱和。负的对数似然帮助我们在很多模型中避免这个问题。很多输出单元都会包含一个指数函数，这在它的变量取绝对值非常大的负值时会造成饱和。负对数似然代价函数中的对数函数消除了某些输出单元中的指数效果。我们将会在6.2.2节中讨论代价函数和输出单元的选择间的交互关系。

用于实现最大似然估计的交叉熵代价函数有一个不同寻常的特性，那就是当它被应用于实践中经常遇到的模型时，它通常没有最小值。对于离散型输出变量，大多数模型以一种特殊的形式来参数化，即它们不能表示概率零和一，但是可以无限接近。逻辑回归是其中一个例子。对于实值的输出变量，如果模型可以控制输出分布的密度（例如，通过学习高斯输出分布的方差参数），那么它可能对正确的训练集

输出赋予极其高的密度，这将导致交叉熵趋向负无穷。第七章中描述的正则化技术提供了一些不同的方法来修正学习问题，所以模型不会通过这种方式来获得无限制的收益。

### 6.2.1.2 学习条件统计量

有时我们并不是想学习一个完整的概率分布  $p(\mathbf{y} \mid \mathbf{x}; \theta)$ ，而仅仅是想学习在给定  $\mathbf{x}$  时  $\mathbf{y}$  的某个条件统计量。

例如，我们可能有一个预测器  $f(\mathbf{x}; \theta)$ ，我们想用它来预测  $\mathbf{y}$  的均值。如果我们使用一个足够强大的神经网络，我们可以认为这个神经网络能够表示一大类函数中的任何一个函数  $f$ ，这个类仅仅被一些特征所限制，例如连续性和有界，而不是具有特殊的参数形式。从这个角度来看，我们可以把代价函数看作是一个泛函 (functional) 而不仅仅是一个函数。泛函是函数到实数的映射。我们因此可以将学习看作是选择一个函数而不仅仅是选择一组参数。我们可以设计代价泛函在我们想要的某些特殊函数处取得最小值。例如，我们可以设计一个代价泛函，使它的最小值处于一个特殊的函数上，这个函数将  $\mathbf{x}$  映射到给定  $\mathbf{x}$  时  $\mathbf{y}$  的期望值。对函数求解优化问题需要用到变分法 (calculus of variations) 这个数学工具，我们将在19.4.2节中讨论。理解变分法对于理解本章的内容不是必要的。目前，只需要知道变分法可以被用来导出下面的两个结果。

我们使用变分法导出的第一个结果是解优化问题

$$f^* = \arg \min_f \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim p_{\text{data}}} \|\mathbf{y} - f(\mathbf{x})\|^2 \quad (6.14)$$

得到

$$f^*(\mathbf{x}) = \mathbb{E}_{\mathbf{y} \sim p_{\text{data}}(\mathbf{y} \mid \mathbf{x})} [\mathbf{y}], \quad (6.15)$$

要求这个函数处在我们要优化的类里。换句话说，如果我们能够用无穷多的、来源于真实的数据生成分布的样例进行训练，最小化均方误差代价函数将得到一个函数，它可以用来对每个  $\mathbf{x}$  的值预测出  $\mathbf{y}$  的均值。

不同的代价函数给出不同的统计量。第二个使用变分法得到的结果是

$$f^* = \arg \min_f \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim p_{\text{data}}} \|\mathbf{y} - f(\mathbf{x})\|_1 \quad (6.16)$$

将得到一个函数可以对每个  $\mathbf{x}$  预测  $\mathbf{y}$  取值的中位数，只要这个函数在我们要优化的函数族里。这个代价函数通常被称为平均绝对误差 (mean absolute error)。

不幸的是，均方误差和平均绝对误差在使用基于梯度的优化方法时往往会导致糟糕的结果。一些饱和的输出单元当结合这些代价函数时会产生非常小的梯度。这就是为什么交叉熵代价函数比均方误差或者平均绝对误差更受欢迎的原因之一了，即使是在没必要估计整个  $p(\mathbf{y} | \mathbf{x})$  分布时。

## 6.2.2 输出单元

代价函数的选择与输出单元的选择紧密相关。大多数时候，我们简单地使用数据分布和模型分布间的交叉熵。选择怎样表示输出决定了交叉熵函数的形式。

任何种类的可以被用作输出的神经网络单元，也可以被用作隐藏单元。这里，我们关注把这些单元用作模型的输出，但是原则上它们也可以在内部使用。我们将在6.3节中重温这些单元并且给出当它们被用作隐藏单元时一些额外的细节。

在本节中，我们假设前馈网络提供了一组定义为  $\mathbf{h} = f(\mathbf{x}; \boldsymbol{\theta})$  的隐藏特征。输出层的作用是随后对这些特征进行一些额外的变换来完成整个网络必须完成的任务。

### 6.2.2.1 用于高斯输出分布的线性单元

一种简单的输出单元是基于仿射变换的输出单元，仿射变换不带有非线性。这些单元往往被直接称为线性单元。

给定特征  $\mathbf{h}$ ，线性输出单元层产生一个向量  $\hat{\mathbf{y}} = \mathbf{W}^\top \mathbf{h} + \mathbf{b}$ 。

线性输出层经常被用来产生条件高斯分布的均值：

$$p(\mathbf{y} | \mathbf{x}) = \mathcal{N}(\mathbf{y}; \hat{\mathbf{y}}, \mathbf{I}). \quad (6.17)$$

最大化对数似然此时等价于最小化均方误差。

最大化似然的框架使它也可以很直观的来学习高斯分布的协方差矩阵，或者使得高斯分布的协方差是输入的函数。然而，对于所有输入，协方差矩阵都必须被限制成一个正定的矩阵。用线性输出层来满足这种限制是困难的，所以通常使用其他的输出单元来对协方差参数化。对协方差建模的方法在6.2.2.4节中简要介绍。

因为线性模型不会饱和，所以它们对基于梯度的优化算法没有任何困难并且可以被用在相当广泛的优化算法中。

### 6.2.2.2 用于 Bernoulli 输出分布的 sigmoid 单元

许多任务需要预测二值型变量  $y$  的值。具有两个类的分类问题可以归结为这种形式。

此时最大似然的方法是定义  $y$  在  $\mathbf{x}$  条件下的Bernoulli 分布。

Bernoulli 分布仅需单个参数来定义。神经网络只需要预测  $P(y = 1 | \mathbf{x})$  即可。为了使这个数是有效的概率，它必须处在区间  $[0, 1]$  中。

满足这个限制需要一些细致的设计工作。假设我们打算使用线性单元，并且通过阈值来限制它成为一个有效的概率：

$$P(y = 1 | \mathbf{x}) = \max \{0, \min\{1, \mathbf{w}^\top \mathbf{h} + b\}\}. \quad (6.18)$$

这的確定义了一个有效的条件概率分布，但我们并不能使用梯度下降来高效地训练它。任何时候当  $\mathbf{w}^\top \mathbf{h} + b$  处于单位区间外时，模型的输出对它的参数的梯度都将为 0。梯度为 0 通常是有问题的，因为学习算法对于如何提高相应的参数没有了指导。

与之相对的，最好是使用一种不同的方法来保证无论何时模型给出了错误的答案时总能有一个很强的梯度。这种方法是基于使用sigmoid输出单元结合最大似然来实现的。

sigmoid输出单元定义为

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{h} + b), \quad (6.19)$$

这里  $\sigma$  是第3.10节中介绍的logistic sigmoid函数。

我们可以认为sigmoid输出单元具有两个部分。首先，它使用一个线性层来计算  $z = \mathbf{w}^\top \mathbf{h} + b$ 。接着，它使用sigmoid激活函数将  $z$  转化成概率。

我们暂时忽略对于  $\mathbf{x}$  的依赖性，只讨论如何用  $z$  的值来定义  $y$  的概率分布。sigmoid可以通过构造一个非归一化（和不为 1）的概率分布  $\tilde{P}(y)$  来得到。我们可以随后除以一个合适的常数来得到有效的概率分布。如果我们假定非归一化的对数概率对  $y$  和  $z$  是线性的，可以对它取指数来得到非归一化的概率。我们然后对它归

一化，可以发现这服从Bernoulli 分布，它受  $z$  的sigmoid变换控制：

$$\log \tilde{P}(y) = yz, \quad (6.20)$$

$$\tilde{P}(y) = \exp(yz), \quad (6.21)$$

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^1 \exp(y'z)}, \quad (6.22)$$

$$P(y) = \sigma((2y - 1)z). \quad (6.23)$$

基于指数和归一化的概率分布在统计建模的文献中很常见。用于定义这种二值型变量分布的变量  $z$  被称为分对数 (logit)。

这种在对数空间里预测概率的方法可以很自然地使用最大似然学习。因为用于最大似然的代价函数是  $-\log P(y | \mathbf{x})$ ，代价函数中的  $\log$  抵消了sigmoid中的  $\exp$ 。如果没有这个效果，sigmoid的饱和性会阻止基于梯度的学习做出好的改进。我们使用最大似然来学习一个由sigmoid参数化的Bernoulli 分布，它的损失函数为

$$J(\boldsymbol{\theta}) = -\log P(y | \mathbf{x}) \quad (6.24)$$

$$= -\log \sigma((2y - 1)z) \quad (6.25)$$

$$= \zeta((1 - 2y)z). \quad (6.26)$$

这个推导使用了3.10节中的一些性质。通过将损失函数写成softplus 函数的形式，我们可以看到它仅仅在  $(1 - 2y)z$  取绝对值非常大的负值时才会饱和。因此饱和只会出现在模型已经得到正确答案时——当  $y = 1$  且  $z$  取非常大的正值时，或者  $y = 0$  且  $z$  取非常小的负值时。当  $z$  的符号错误时，softplus 函数的变量  $(1 - 2y)z$  可以简化为  $|z|$ 。当  $|z|$  变得很大并且  $z$  的符号错误时，softplus 函数渐进地趋向于它的变量  $|z|$ 。对  $z$  求导则渐进地趋向于  $\text{sign}(z)$ ，所以，对于极限情况下极度不正确的  $z$ ，softplus 函数完全不会收缩梯度。这个性质很有用，因为它意味着基于梯度的学习可以很快地改正错误的  $z$ 。

当我们使用其他的损失函数，例如均方误差之类的，损失函数会在任何  $\sigma(z)$  饱和时饱和。sigmoid激活函数在  $z$  取非常小的负值时会饱和到 0，当  $z$  取非常大的正值时会饱和到 1。这种情况一旦发生，梯度会变得非常小以至于不能用来学习，无论此时模型给出的是正确还是错误的答案。因为这个原因，最大似然几乎总是训练sigmoid输出单元的优选方法。

理论上，sigmoid的对数总是确定和有限的，因为sigmoid的返回值总是被限制在开区间  $(0, 1)$  上，而不是使用整个闭区间  $[0, 1]$  的有效概率。在软件实现时，为了

避免数值问题，最好将负的对数似然写作  $z$  的函数，而不是  $\hat{y} = \sigma(z)$  的函数。如果sigmoid函数下溢到零，那么之后对  $\hat{y}$  取对数会得到负无穷。

### 6.2.2.3 用于 Multinoulli 输出分布的 softmax 单元

任何时候当我们想要表示一个具有  $n$  个可能取值的离散型随机变量的分布时，我们都可以使用 softmax 函数。它可以看作是sigmoid函数的扩展，sigmoid函数用来表示二值型变量的分布。

softmax 函数最常用作分类器的输出，来表示  $n$  个不同类上的概率分布。比较少见的是，softmax 函数可以在模型内部使用，例如如果我们想要在某个内部变量的  $n$  个不同选项中进行选择。

在二值型变量的情况下，我们希望计算一个单独的数

$$\hat{y} = P(y = 1 | \mathbf{x}). \quad (6.27)$$

因为这个数需要处在 0 和 1 之间，并且我们想要让这个数的对数可以很好地用于对数似然的基于梯度的优化，我们选择去预测另外一个数  $z = \log \hat{P}(y = 1 | \mathbf{x})$ 。对其指数化和归一化，我们就得到了一个由sigmoid函数控制的Bernoulli 分布。

为了推广到具有  $n$  个值的离散型变量的情况，我们现在需要创造一个向量  $\hat{\mathbf{y}}$ ，它的每个元素是  $\hat{y}_i = P(y = i | \mathbf{x})$ 。我们不仅要求每个  $\hat{y}_i$  元素介于 0 和 1 之间，还要使得整个向量的和为 1 使得它表示一个有效的概率分布。用于Bernoulli 分布的方法同样可以推广到Multinoulli 分布。首先，线性层预测了非标准化的对数概率：

$$\mathbf{z} = \mathbf{W}^\top \mathbf{h} + \mathbf{b}, \quad (6.28)$$

其中  $z_i = \log \hat{P}(y = i | \mathbf{x})$ 。softmax 函数然后可以对  $z$  指数化和归一化来获得需要的  $\hat{y}$ 。最终，softmax 函数的形式为

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \quad (6.29)$$

和logistic sigmoid一样，当使用最大化对数似然训练 softmax 来输出目标值  $y$  时，使用指数函数工作地非常好。这种情况下，我们想要最大化  $\log P(y = i; \mathbf{z}) = \log \text{softmax}(\mathbf{z})_i$ 。将 softmax 定义成指数的形式是很自然的因为对数似然中的 log 可以抵消 softmax 中的 exp：

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j). \quad (6.30)$$

公式6.30中的第一项表示输入  $z_i$  总是对代价函数有直接的贡献。因为这一项不会饱和，所以即使  $z_i$  对公式6.30的第二项的贡献很小，学习依然可以进行。当最大化对数似然时，第一项鼓励  $z_i$  被推高，而第二项则鼓励所有的  $z$  被压低。为了对第二项  $\log \sum_j \exp(z_j)$  有一个直观的理解，注意到这一项可以大致近似为  $\max_j z_j$ 。这种近似是基于对任何明显小于  $\max_j z_j$  的  $z_k$ ， $\exp(z_k)$  都是不重要的。我们能从这种近似中得到的直觉是，负对数似然代价函数总是强烈地惩罚最活跃的不正确预测。如果正确答案已经具有了 softmax 的最大输入，那么  $-z_i$  项和  $\log \sum_j \exp(z_j) \approx \max_j z_j = z_i$  项将大致抵消。这个样本对于整体训练代价贡献很小，将由其他未被正确分类的样本支配。

到目前为止我们只讨论了一个例子。总体来说，未正则化的最大似然会驱动模型去学习一些参数，而这些参数会驱动 softmax 函数来预测在训练集中观察到的每个结果的比率：

$$\text{softmax}(\mathbf{z}(\mathbf{x}; \boldsymbol{\theta}))_i \approx \frac{\sum_{j=1}^m \mathbf{1}_{y^{(j)}=i, \mathbf{x}^{(j)}=\mathbf{x}}}{\sum_{j=1}^m \mathbf{1}_{\mathbf{x}^{(j)}=\mathbf{x}}} \quad (6.31)$$

因为最大似然是一致的估计量，所以只要模型族能够表示训练的分布，这就能保证发生。在实践中，有限的模型能力和不完美的优化将意味着模型只能近似这些比率。

除了对数似然之外的许多目标函数对 softmax 函数不起作用。具体来说，那些不使用对数来抵消 softmax 中的指数的目标函数，当指数函数的变量取非常小的负值时会造成梯度消失，从而无法学习。特别是，平方误差对于 softmax 单元来说是一个很差的损失函数，即使模型做出高度可信的不正确预测，也不能训练模型改变其输出 (Bridle, 1990)。要理解为什么这些损失函数可能失败，我们需要检查 softmax 函数本身。

像sigmoid一样，softmax 激活函数可能会饱和。sigmoid函数具有单个输出，当它的输入极端负或者极端正时会饱和。对于 softmax 的情况，它有多个输出值。当输入值之间的差异变得极端时，这些输出值可能饱和。当 softmax 饱和时，基于 softmax 的许多代价函数也饱和，除非它们能够转化饱和的激活函数。

为了说明 softmax 函数对于输入之间差异的响应，观察到当对所有的输入都加上一个相同常数时 softmax 的输出不变：

$$\text{softmax}(\mathbf{z}) = \text{softmax}(\mathbf{z} + c). \quad (6.32)$$

使用这个性质，我们可以导出一个数值方法稳定的 softmax 函数的变体：

$$\text{softmax}(\mathbf{z}) = \text{softmax}(\mathbf{z} - \max_i z_i). \quad (6.33)$$

变换后的形式允许我们在对 softmax 函数求值时只有很小的数值误差，即使是当  $z$  包含极正或者极负的数时。观察 softmax 数值稳定的变体，可以看到 softmax 函数由它的变量偏离  $\max_i z_i$  的量来驱动。

当其中一个输入是最大 ( $z_i = \max_i z_i$ ) 并且  $z_i$  远大于其他的输入时，相应的输出  $\text{softmax}(z)_i$  会饱和到 1。当  $z_i$  不是最大值并且最大值非常大时，相应的输出  $\text{softmax}(z)_i$  也会饱和到 0。这是 sigmoid 单元饱和方式的一般化，并且如果损失函数不被设计成对其进行补偿，那么也会造成类似的学习困难。

softmax 函数的变量  $z$  可以用两种方式产生。最常见的是简单地使神经网络较早的层输出  $z$  的每个元素，就像先前描述的使用线性层  $z = W^\top h + b$ 。虽然很直观，但这种方法是对分布的过度参数化。 $n$  个输出总和必须为 1 的约束意味着只有  $n - 1$  个参数是必要的；第  $n$  个概率值可以通过 1 减去前面  $n - 1$  个概率来获得。因此，我们可以强制要求  $z$  的一个元素是固定的。例如，我们可以要求  $z_n = 0$ 。事实上，这正是 sigmoid 单元所做的。定义  $P(y=1 | x) = \sigma(z)$  等价于用二维的  $z$  以及  $z_1 = 0$  来定义  $P(y=1 | x) = \text{softmax}(z)_1$ 。无论是  $n - 1$  个变量还是  $n$  个变量的方法，都描述了相同的概率分布，但会产生不同的学习机制。在实践中，无论是过度参数化的版本还是限制的版本都很少有差别，并且实现过度参数化的版本更为简单。

从神经科学的角度看，有趣的是认为 softmax 是一种在参与其中的单元之间形成竞争的方式：softmax 输出总是和为 1，所以一个单元的值增加必然对应着其他单元值的减少。这与被认为存在于皮质中相邻神经元间的侧抑制类似。在极端情况下（当最大的  $a_i$  和其他的在幅度上差异很大时），它变成了赢者通吃 (winner-take-all) 的形式（其中一个输出接近 1，其他的接近 0）。

“softmax”的名称可能会让人产生困惑。这个函数更接近于 argmax 函数而不是 max 函数。“soft”这个术语来源于 softmax 函数是连续可微的。“argmax”函数的结果表示为一个 one-hot 向量（只有一个元素为 1，其余元素都为 0 的向量），不是连续和可微的。softmax 函数因此提供了 argmax 的“软化”版本。max 函数相应的软化版本是  $\text{softmax}(z)^\top z$ 。可能最好是把 softmax 函数称为“softargmax”，但当前名称是一个根深蒂固的习惯了。

#### 6.2.2.4 其他的输出类型

先前描述的线性、sigmoid 和 softmax 输出单元是最常见的。神经网络可以推广到我们希望的几乎任何种类的输出层。最大似然原则为如何为几乎任何种类的输出

层设计一个好的代价函数提供了指导。

一般的，如果我们定义了一个条件分布  $p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta})$ ，最大似然原则建议我们使用  $-\log p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta})$  作为代价函数。

一般来说，我们可以认为神经网络表示函数  $f(\mathbf{x}; \boldsymbol{\theta})$ 。这个函数的输出不是对  $\mathbf{y}$  值的直接预测。相反， $f(\mathbf{x}; \boldsymbol{\theta}) = \boldsymbol{\omega}$  提供了  $\mathbf{y}$  分布的参数。我们的损失函数就可以表示成  $-\log p(\mathbf{y}; \boldsymbol{\omega}(\mathbf{x}))$ 。

例如，我们想要学习在给定  $\mathbf{x}$  时  $\mathbf{y}$  的条件高斯分布的方差。简单情况下，方差  $\sigma^2$  是一个常数，此时有一个闭合的表达式，这是因为方差的最大似然估计量仅仅是观测值  $\mathbf{y}$  与它们的期望值的差值的平方平均。一种计算上更加昂贵但是不需要写特殊情况代码的方法是简单地将方差作为分布  $p(\mathbf{y} | \mathbf{x})$  的其中一个属性，这个分布受  $\boldsymbol{\omega} = f(\mathbf{x}; \boldsymbol{\theta})$  控制。负对数似然  $-\log p(\mathbf{y}; \boldsymbol{\omega}(\mathbf{x}))$  将为代价函数提供一个必要的合适项来使我们的优化过程可以逐渐地学到方差。在标准差不依赖于输入的简单情况下，我们可以在网络中创建一个直接复制到  $\boldsymbol{\omega}$  中的新参数。这个新参数可以是  $\sigma$  本身，或者可以是表示  $\sigma^2$  的参数  $v$ ，或者可以是表示  $\frac{1}{\sigma^2}$  的参数  $\beta$ ，取决于我们怎样对分布参数化。我们可能希望模型对不同的  $\mathbf{x}$  值预测出  $\mathbf{y}$  不同的方差。这被称为异方差 (heteroscedastic) 模型。在异方差情况下，我们简单地把方差指定为  $f(\mathbf{x}; \boldsymbol{\theta})$  其中一个输出值。实现它的典型方法是使用精度而不是方差来表示高斯分布，就像公式3.22所描述的。在多维变量的情况下，最常见的是使用一个对角精度矩阵

$$\text{diag}(\boldsymbol{\beta}). \quad (6.34)$$

这个公式适用于梯度下降，因为由  $\beta$  参数化的高斯分布的对数似然的公式仅涉及  $\beta_i$  的乘法和  $\log \beta_i$  的加法。乘法、加法和对数运算的梯度表现良好。相比之下，如果我们用方差来参数化输出，我们需要用到除法。除法函数在零附近会变得任意陡峭。虽然大梯度可以帮助学习，但任意大的梯度通常导致不稳定。如果我们用标准差来参数化输出，对数似然仍然会涉及除法，并且还将涉及平方。通过平方运算的梯度可能在零附近消失，这使得学习被平方的参数变得困难。无论我们使用的是标准差，方差还是精度，我们必须确保高斯分布的协方差矩阵是正定的。因为精度矩阵的特征值是协方差矩阵特征值的倒数，所以这等价于确保精度矩阵是正定的。如果我们使用对角矩阵，或者是一个常数乘以单位矩阵<sup>1</sup>，那么我们需要对模型输出强加的唯一条件是它的元素都为正。如果我们假设  $\mathbf{a}$  是用于确定对角精度的模型的原始激活，

<sup>1</sup>译者注：这里原文是 “If we use a diagonal matrix, or a scalar times the diagonal matrix...” 即 “如果我们使用对角矩阵，或者是一个标量乘以对角矩阵...”，但一个标量乘以对角矩阵和对角矩阵没区别，结合上下文可以看出，这里原作者误把 “identity” 写成了 “diagonal matrix”，因此这里采用 “常数乘以单位矩阵”的译法。

那么可以用 softplus 函数来获得正的精度向量： $\beta = \zeta(\mathbf{a})$ 。这种相同的策略对于方差或标准差同样适用，也适用于常数乘以单位阵的情况。

学习一个比对角矩阵具有更丰富结构的协方差或者精度矩阵是很少见的。如果协方差矩阵是满的和有条件的，那么参数化的选择就必须要保证预测的协方差矩阵是正定的。这可以通过写成  $\Sigma(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{B}^\top(\mathbf{x})$  来实现，这里  $\mathbf{B}$  是一个无约束的方阵。如果矩阵是满秩的，那么一个实际问题是计算似然是很昂贵的，计算一个  $d \times d$  的矩阵的行列式或者  $\Sigma(\mathbf{x})$  的逆（或者等价地并且更常用地，对它特征值分解或者  $\mathbf{B}(\mathbf{x})$  的特征值分解）需要  $O(d^3)$  的计算量。

我们经常想要执行多峰回归 (multimodal regression)，即预测条件分布  $p(\mathbf{y} | \mathbf{x})$  的实值，该条件分布对于相同的  $\mathbf{x}$  值在  $\mathbf{y}$  空间中有多个不同的峰值。在这种情况下，Gaussian 混合是输出的自然表示 (Jacobs *et al.*, 1991; Bishop, 1994)。带有将 Gaussian 混合作为其输出的神经网络通常被称为混合密度网络 (mixture density network)。具有  $n$  个分量的 Gaussian 混合输出由下面的条件分布定义

$$p(\mathbf{y} | \mathbf{x}) = \sum_{i=1}^n p(c=i | \mathbf{x}) \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}^{(i)}(\mathbf{x}), \boldsymbol{\Sigma}^{(i)}(\mathbf{x})). \quad (6.35)$$

神经网络必须有三个输出：定义  $p(c=i | \mathbf{x})$  的向量，对所有的  $i$  给出  $\boldsymbol{\mu}^{(i)}(\mathbf{x})$  的矩阵，以及对所有的  $i$  给出  $\boldsymbol{\Sigma}^{(i)}(\mathbf{x})$  的张量。这些输出必须满足不同的约束：

1. 混合组件  $p(c=i | \mathbf{x})$ : 它们由潜变量<sup>2</sup>  $c$  关联着，在  $n$  个不同组件上形成 Multinoulli 分布。这个分布通常可以由  $n$  维向量的 softmax 来获得，以确保这些输出是正的并且和为 1。
2. 均值  $\boldsymbol{\mu}^{(i)}(\mathbf{x})$ : 它们指明了与第  $i$  个 Gaussian 组件相关联的中心或者均值，并且是无约束的（通常对于这些输出单元完全没有非线性）。如果  $\mathbf{y}$  是一个  $d$  维向量，那么网络必须输出一个由  $n$  个这种  $d$  维向量组成的  $n \times d$  的矩阵。用最大似然来学习这些均值要比学习只有一个输出模式的分布的均值稍稍复杂一些。我们只想更新那个真正产生观测数据的组件的均值。在实践中，我们并不知道是那个组件产生了观测数据。负对数似然的表达式对每个样例对于损失函数的贡献关于每个组件赋予相应的权重，权重值的大小由相应的组件产生这个样例的概率来决定。

<sup>2</sup> 我们之所以认为  $c$  是潜在的，是因为我们不能直接在数据中观测到它：给定输入  $\mathbf{x}$  和目标  $\mathbf{y}$ ，不可能确切地知道是哪个 Gaussian 组件产生  $\mathbf{y}$ ，但我们可以想象  $\mathbf{y}$  是通过选择其中一个来产生的，并且将那个未被观测到的选择作为随机变量。

3. 协方差  $\Sigma^{(i)}(\mathbf{x})$ : 它们指明了每个组件  $i$  的协方差矩阵。当学习单个的 Gaussian 组件时，我们通常使用对角矩阵来避免计算行列式。当学习混合均值时，最大似然是很复杂的，它需要将每个点的部分责任分配给每个混合组件。如果给定了混合模型的正确的负对数似然，梯度下降将自动地遵循正确的过程。

有报道说基于梯度的优化方法对于混合条件 Gaussian（作为神经网络的输出）可能是不可靠的，部分是因为涉及到除法（除以方差）可能是数值不稳定的（当某个方差对于特定的实例变得非常小时，会导致非常大的梯度）。一种解决方法是梯度截断 (clip gradient) (见10.11.1节)，另外一种是启发式梯度放缩 (Murray and Larochelle, 2014)。

Gaussian 混合输出在语音生成模型 (Schuster, 1999) 和物理运动 (Graves, 2013) 中特别有效。混合密度策略为网络提供了一种方法来表示多种输出模式，并且控制输出的方差，这对于在这些实数域中获得高质量的结果是至关重要的。混合密度网络的一个实例如图6.4所示。

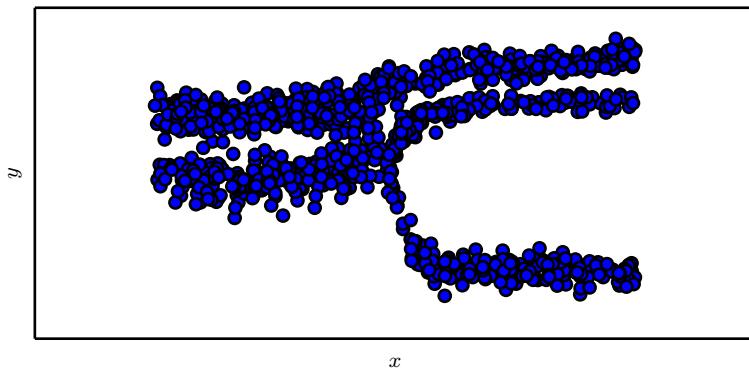


图 6.4: 从具有混合密度输出层的神经网络中抽取的样本。输入  $x$  从均匀分布中采样，输出  $y$  从  $p_{\text{model}}(y | x)$  中采样。神经网络能够学习从输入到输出分布的参数的非线性映射。这些参数包括控制三个组件中的哪一个将产生输出的概率，以及每个组件各自的参数。每个混合组件都是高斯分布，具有预测的均值和方差。输出分布的这些方面都能够相对输入  $x$  变化，并且以非线性的方式改变。

一般的，我们可能希望继续对包含更多变量的更大的向量  $\mathbf{y}$  来建模，并在这些输出变量上施加更多更丰富的结构。例如，我们可能希望神经网络输出字符序列形成一个句子。在这些情况下，我们可以继续使用最大似然原理应用到我们的模

型  $p(\mathbf{y}; \omega(\mathbf{x}))$  上，但我们用来描述  $\mathbf{y}$  的模型会变得非常复杂，超出了本章的范畴。第十章描述了如何用循环神经网络来定义这种序列上的模型，第三部分描述了对任意概率分布进行建模的高级技术。

## 6.3 隐藏单元

到目前为止我们集中讨论了神经网络的设计选择，这对于使用基于梯度的优化方法来训练的大多数参数化机器学习模型都是通用的。现在我们转向一个前馈神经网络独有的问题：该如何选择隐藏单元的类型，这些隐藏单元用在模型的隐藏层中。

隐藏单元的设计是一个非常活跃的研究领域，并且还没有许多明确的指导性理论原则。

整流线性单元是隐藏单元极好的默认选择。许多其他类型的隐藏单元也是可用的。决定何时使用哪种类型的隐藏单元是困难的事（尽管整流线性单元通常是一个可接受的选择）。我们这里描述对于每种隐藏单元的一些基本直觉。这些直觉可以用来建议我们何时来尝试一些单元。通常不可能预先预测出哪种隐藏单元工作得最好。设计过程充满了试验和错误，先直觉认为某种隐藏单元可能表现良好，然后用它组成神经网络进行训练，最后用校验集来评估它的性能。

这里列出的一些隐藏单元可能并不是在所有的输入点上都是可微的。例如，整流线性单元  $g(z) = \max\{0, z\}$  在  $z = 0$  处不可微。这似乎使得  $g$  对于基于梯度的学习算法无效。在实践中，梯度下降对这些机器学习模型仍然表现得足够好。部分原因是神经网络训练算法通常不会达到代价函数的局部最小值，而是仅仅显著地减小它的值，如图4.3所示。这些想法会在第八章中进一步描述。因为我们不再期望训练能够实际到达梯度为  $\mathbf{0}$  的点，所以代价函数的最小值对应于梯度未定义的点是可以接受的。不可微的隐藏单元通常只在少数点上不可微。一般来说，函数  $g(z)$  具有左导数和右导数，左导数定义为紧邻在  $z$  左边的函数的斜率，右导数定义为紧邻在  $z$  右边的函数的斜率。只有当函数在  $z$  处的左导数和右导数都有定义并且相等时，函数在  $z$  点处才是可微的。神经网络中用到的函数通常对左导数和右导数都有定义。在  $g(z) = \max\{0, z\}$  的情况下，在  $z = 0$  处的左导数是 0，右导数是 1。神经网络训练的软件实现通常返回左导数或右导数的其中一个，而不是报告导数未定义或产生一个错误。这可以通过观察到在数字计算机上基于梯度的优化总是会受到数值误差的影响来启发式地给出理由。当一个函数被要求计算  $g(0)$  时，底层值真正为 0 是不太

可能的。相对的，它可能是被舍入为 0 的一个小量  $\epsilon$ 。在某些情况下，理论上更好的理由可以使用，但这些通常对神经网络训练并不适用。重要的是，在实践中，我们可以放心地忽略下面描述的隐藏单元激活函数的不可微性。

除非另有说明，大多数的隐藏单元都可以描述为接受输入向量  $\mathbf{x}$ ，计算仿射变换  $\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$ ，然后使用一个作用于每个元素的非线性函数  $g(\mathbf{z})$ 。大多数隐藏单元的区别仅仅在于激活函数  $g(\mathbf{z})$  的形式。

### 6.3.1 整流线性单元及其扩展

整流线性单元使用激活函数  $g(z) = \max\{0, z\}$ 。

整流线性单元易于优化，因为它们和线性单元非常类似。线性单元和整流线性单元的唯一区别在于整流线性单元在其一半的定义域上输出为零。这使得只要整流线性单元处于激活状态它的导数都能保持较大。它的梯度不仅大而且一致。整流操作的二阶导数几乎处处为 0，并且在整流线性单元处于激活状态时它的一阶导数处处为 1。这意味着它的梯度方向对于学习来说更加有用，相比于引入二阶效应的激活函数。

整流线性单元通常用于仿射变换之上：

$$\mathbf{h} = g(\mathbf{W}^\top \mathbf{x} + \mathbf{b}). \quad (6.36)$$

当初始化仿射变换的参数时，可以将  $\mathbf{b}$  的所有元素设置成一个小的正值，例如 0.1。这使得整流线性单元很可能初始时就对训练集中的大多数输入呈现激活状态，并且允许导数通过。

有很多整流线性单元的扩展存在。大多数这些扩展的表现比得上整流线性单元，并且偶尔表现得更好。

整流线性单元的一个缺陷是它们不能通过基于梯度的方法学习那些使它们激活为零的样例。整流线性单元的各种扩展保证了它们能在各个位置都接收到梯度。

整流线性单元的三个扩展基于当  $z_i < 0$  时使用一个非零的斜率  $\alpha_i$ :  $h_i = g(\mathbf{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$ 。绝对值整流 (absolute value rectification) 固定  $\alpha_i = 1$  来得到  $g(z) = |z|$ 。它用于图像中的对象识别 (Jarrett *et al.*, 2009a)，其中寻找在输入照明极性反转下不变的特征是有意义的。整流线性单元的其他扩展更广泛地适用。渗漏整流线性单元 (Leaky ReLU)(Maas *et al.*, 2013) 将  $\alpha_i$  固定成一个类似

0.01 的小值，参数化整流线性单元 (parametric ReLU) 或者 PReLU 将  $\alpha_i$  作为学习的参数 (He *et al.*, 2015)。

**maxout 单元** (maxout unit) (Goodfellow *et al.*, 2013a) 进一步扩展了整流线性单元。并不是使用作用于每个元素的函数  $g(z)$ ，maxout 单元将  $z$  划分为具有  $k$  个值的组。每个 maxout 单元然后输出其中一组的最大元素：

$$g(\mathbf{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j \quad (6.37)$$

这里  $\mathbb{G}^{(i)}$  是组  $i$  的输入指标集  $\{(i-1)k+1, \dots, ik\}$ 。这提供了一种方法来学习对输入  $\mathbf{x}$  空间中多个方向响应的分段线性函数。

maxout 单元可以学习具有多达  $k$  段的分段线性的凸函数。maxout 单元因此可以视为学习激活函数本身而不仅仅是单元之间的关系。使用足够大的  $k$ ，maxout 单元可以以任意的逼真度来近似任何凸函数。特别地，具有每组两块的 maxout 层可以学习实现和传统层相同的输入  $\mathbf{x}$  的函数，包括整流线性激活函数、绝对值整流、渗漏整流线性单元 或 参数化整流线性单元、或者可以学习实现与这些都不同的函数。maxout 层的参数化当然也将与这些层不同，所以即使是 maxout 学习去实现和其他种类的层相同的  $\mathbf{x}$  的函数这种情况下，学习的机理也是不一样的。

每个 maxout 单元现在由  $k$  个权重向量来参数化，而不仅仅是一个，所以 maxout 单元通常比整流线性单元需要更多的正则化。如果训练集很大并且每个单元分得的块数保持很低的话，它们可以在没有正则化的情况下正常工作 (Cai *et al.*, 2013)。

maxout 单元还有一些其他的优点。在某些情况下，需要更少的参数可以获得一些统计和计算上的优点。具体来说，如果由  $n$  个不同的线性过滤器描述的特征可以在不损失信息的情况下，用每一组  $k$  个特征的最大值来概括的话，那么下一层可以获得  $k$  倍更少的权重数。

因为每个单元由多个过滤器驱动，maxout 单元具有一些冗余来帮助它们抵抗一种被称为灾难遗忘 (catastrophic forgetting) 的现象，这个现象是说神经网络忘记了如何执行它们过去训练的任务 (Goodfellow *et al.*, 2014a)。

整流线性单元和它们的这些扩展都是基于一个原则，那就是如果它们的行为更接近线性，那么模型更容易优化。使用线性行为更容易优化的一般性原则同样也适用在除了深度线性网络以外的内容。循环网络可以从序列中学习并产生状态和输出的序列。当训练它们时，需要通过一些时间步长来传播信息，当其中包含一些线性计算（具有大小接近 1 的某些方向导数）时，这会更容易。作为性能最好的循环网

络结构之一，LSTM 通过求和来在时间上传播信息，这是一种特别直观的线性激活。它将在10.10节中进一步讨论。

### 6.3.2 logistic sigmoid与双曲正切函数

在引入整流线性单元之前，大多数神经网络使用logistic sigmoid激活函数

$$g(z) = \sigma(z) \quad (6.38)$$

或者是双曲正切激活函数

$$g(z) = \tanh(z). \quad (6.39)$$

这些激活函数紧密相关，因为  $\tanh(z) = 2\sigma(2z) - 1$ 。

我们已经看过sigmoid单元作为输出单元用来预测二值型变量取值为 1 的概率。与分段线性单元不同，sigmoid单元在其大部分定义域内都饱和——当  $z$  取绝对值很大的正值时，它们饱和到一个高值，当  $z$  取绝对值很大的负值时，它们饱和到一个低值，并且仅仅当  $z$  接近 0 时它们才对输入强烈敏感。sigmoid单元的广泛饱和性会使得基于梯度的学习变得非常困难。因为这个原因，现在不鼓励将它们用作前馈网络中的隐藏单元。它们作为输出单元可以与基于梯度的学习相兼容，如果使用了一个合适的代价函数来抵消sigmoid的饱和性的话。

当必须要使用sigmoid激活函数时，双曲正切激活函数通常要比logistic sigmoid函数表现更好。在  $\tanh(0) = 0$  而  $\sigma(0) = \frac{1}{2}$  的意义上，它更像是单位函数。因为  $\tanh$  在 0 附近与单位函数类似，训练深层神经网络  $\hat{y} = \mathbf{w}^\top \tanh(\mathbf{U}^\top \tanh(\mathbf{V}^\top \mathbf{x}))$  类似于训练一个线性模型  $\hat{y} = \mathbf{w}^\top \mathbf{U}^\top \mathbf{V}^\top \mathbf{x}$ ，只要网络的激活能够被保持地很小。这使得训练  $\tanh$  网络更加容易。

sigmoid激活函数在除了前馈网络以外的配置中更为常见。循环网络、许多概率模型以及一些自编码器有一些额外的要求使得它们不能使用分段线性激活函数，并且使得sigmoid单元更具有吸引力，尽管它存在饱和性的问题。

### 6.3.3 其他隐藏单元

也存在许多其他种类的隐藏单元，但它们并不常用。

一般来说，很多种类的可微函数都表现得很好。许多未发布的激活函数与流行的激活函数表现得一样好。为了提供一个具体的例子，作者在 MNIST 数据集上使

用  $h = \cos(\mathbf{W}\mathbf{x} + \mathbf{b})$  测试了一个前馈网络，并获得了小于 1% 的误差率，这可以与更为传统的激活函数获得的结果相媲美。在新技术的研究和开发期间，通常会测试许多不同的激活函数，并且会发现许多标准方法的变体表现非常好。这意味着，通常新的隐藏单元类型只有在被明确证明能够提供显著改进时才会被发布。新的隐藏单元类型如果与已有的隐藏单元表现大致相当的话，那么它们是非常常见的，不会引起别人的兴趣。

想要列出文献中出现的所有隐藏单元类型是不切实际的。我们只对一些特别有用和独特的类型进行强调。

其中一种是完全没有激活函数  $g(z)$ 。也可以认为这是使用单位函数作为激活函数。我们已经看过线性单元可以用作神经网络的输出。它也可以用作隐藏单元。如果神经网络的每一层都仅由线性变换组成，那么网络作为一个整体也将是线性的。然而，神经网络的一些层是纯线性也是可以接受的。考虑具有  $n$  个输入和  $p$  个输出的神经网络层  $\mathbf{h} = g(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$ 。我们可以用两层来代替它，一层使用权重矩阵  $\mathbf{U}$ ，另一层使用权重矩阵  $\mathbf{V}$ 。如果第一层没有激活函数，那么我们对基于  $\mathbf{W}$  的原始层的权重矩阵进行因式分解。分解方法是计算  $\mathbf{h} = g(\mathbf{V}^\top \mathbf{U}^\top \mathbf{x} + \mathbf{b})$ 。如果  $\mathbf{U}$  产生了  $q$  个输出，那么  $\mathbf{U}$  和  $\mathbf{V}$  一起仅包含  $(n+p)q$  个参数，而  $\mathbf{W}$  包含  $np$  个参数。如果  $q$  很小，这可以在很大程度上节省参数。这是以将线性变换约束为低秩的代价来实现的，但这些低秩关系往往是足够的。线性隐藏单元因此提供了一种减少网络中参数数量的有效方法。

softmax 单元是另外一种经常用作输出的单元（如第6.2.2.3节中所描述的），但有时也可以用作隐藏单元。softmax 单元很自然地表示具有  $k$  个可能值的离散型随机变量的概率分布，所以它们可以用作一种开关。这些类型的隐藏单元通常仅用于明确地学习操作内存的高级结构中，将在10.12节中描述。

其他一些常见的隐藏单元类型包括：

- **径向基函数** (radial basis function, RBF):  $h_i = \exp\left(-\frac{1}{\sigma_i^2} \|\mathbf{W}_{:,i} - \mathbf{x}\|^2\right)$ 。这个函数在  $\mathbf{x}$  接近模板  $\mathbf{W}_{:,i}$  时更加活跃。因为它对大部分  $\mathbf{x}$  都饱和到 0，因此很难优化。
- **softplus**函数:  $g(a) = \zeta(a) = \log(1 + e^a)$ 。这是整流线性单元的平滑版本，由Dugas *et al.* (2001b) 引入用于函数近似，由Nair and Hinton (2010a) 引入用于无向概率模型的条件分布。Glorot *et al.* (2011a) 比较了 softplus 和整流线性单元，发现后者的结果更好。通常不鼓励使用softplus 函数。softplus 表明隐藏

单元类型的性能可能是非常反直觉的——因为它处处可导或者因为它不完全饱和，人们可能希望它具有优于整流线性单元的点，但根据经验来看，它并没有。

- **硬双曲正切函数 (hard tanh):** 它的形状和 tanh 以及整流线性单元类似，但是不同于后者，它是有界的， $g(a) = \max(-1, \min(1, a))$ 。它由Collobert (2004) 引入。

隐藏单元的设计仍然是一个活跃的研究领域，许多有用的隐藏单元类型仍有待发现。

## 6.4 结构设计

神经网络设计的另一个关键点是确定它的结构。结构 (architecture) 一词是指网络的整体结构：它应该具有多少单元，以及这些单元应该如何连接。

大多数神经网络被组织成称为层的单元组。大多数神经网络结构将这些层布置成链式结构，其中每一层都是前一层的函数。在这种结构中，第一层由下式给出：

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)\top} \mathbf{x} + \mathbf{b}^{(1)}); \quad (6.40)$$

第二层由

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)\top} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}); \quad (6.41)$$

给出，以此类推。

在这些链式结构中，主要的结构考虑是选择网络的深度和每一层的宽度。我们将会看到，即使只有一个隐藏层的网络也足够适应训练集。更深层的网络通常能够对每一层使用更少的单元数和更少的参数，并且经常推广到测试集，但是通常也更难以优化。对于一个具体的任务，理想的网络结构必须通过实验，观测在验证集上的错误来找到。

### 6.4.1 通用近似性质和深度

线性模型，通过矩阵乘法将特征映射到输出，顾名思义，仅能表示线性函数。它具有易于训练的优点，因为当使用线性模型时，许多损失函数会导出凸优化问题。不幸的是，我们经常想要学习非线性函数。

乍一看，我们可能认为学习非线性函数需要为我们想要学习的那种非线性专门设计一类模型族。幸运的是，具有隐藏层的前馈网络提供了一种通用近似框架。具体来说，**通用近似定理** (universal approximation theorem)(Hornik *et al.*, 1989; Cybenko, 1989) 表明，一个前馈神经网络如果具有线性输出层和至少一层具有任何一种“挤压”性质的激活函数（例如logistic sigmoid激活函数）的隐藏层，只要给予网络足够数量的隐藏单元，它可以以任意的精度来近似任何从一个有限维空间到另一个有限维空间的 Borel 可测函数。前馈网络的导数也可以任意好地来近似函数的导数 (Hornik *et al.*, 1990)。Borel 可测的概念超出了本书的范畴；对于我们想要实现的目标，只需要知道定义在  $\mathbb{R}^n$  的有界闭集上的任意连续函数是 Borel 可测的，因此可以用神经网络来近似。神经网络也可以近似从任何有限维离散空间映射到另一个的任意函数。虽然原始定理最初以具有特殊激活函数的单元的形式来描述，这个激活函数当变量取绝对值非常大的正值和负值时都会饱和，通用近似定理也已经被证明对于更广泛类别的激活函数也是适用的，其中就包括现在常用的整流线性单元(Leshno *et al.*, 1993)。

通用近似定理意味着无论我们试图学习什么函数，我们知道一个大的 MLP 一定能够表示这个函数。然而，我们不能保证训练算法能够学得这个函数。即使 MLP 能够表示该函数，学习也可能因两个不同的原因而失败。首先，用于训练的优化算法可能找不到用于期望函数的参数值。其次，训练算法可能由于过拟合选择了错误的函数。回忆5.2.1节中的“没有免费的午餐”定理说明了没有普遍优越的机器学习算法。前馈网络提供了表示函数的通用系统，在这种意义上，给定一个函数，存在一个前馈网络能够近似该函数。不存在通用的过程既能够验证训练集上的特殊样例，又能够选择一个函数来扩展到训练集上没有的点。

通用近似定理说明了存在一个足够大的网络能够达到我们所希望的任意精度，但是定理并没有说这个网络有多大。Barron (1993) 提供了单层网络近似一大类函数所需大小的一些界。不幸的是，在最坏情况下，可能需要指数数量的隐藏单元（可能一个隐藏单元对应着一个需要区分的输入配置）。这在二进制情况下很容易看到：向量  $v \in \{0, 1\}^n$  上的可能的二进制函数的数量是  $2^{2^n}$  并且选择一个这样的函数需要  $2^n$  位，这通常需要  $O(2^n)$  的自由度。

总之，具有单层的前馈网络足以表示任何函数，但是网络层可能不可实现得大并且可能无法正确地学习和泛化。在很多情况下，使用更深的模型能够减少表示期望函数所需的单元的数量，并且可以减少泛化误差。

存在一些函数族能够在网络的深度大于某个值  $d$  时被高效地近似，而当深度被

限制到小于或等于  $d$  时需要一个远远大于之前的模型。在很多情况下，浅层模型所需的隐藏单元的数量是  $n$  的指数级。这个结果最初被证明是在那些不与连续可微的神经网络类似的机器学习模型中出现，但现在已经扩展到了这些模型。第一个结果是关于逻辑门电路的 (Håstad, 1986)。后来的工作将这些结果扩展到了具有非负权重的线性阈值单元 (Håstad and Goldmann, 1991; Hajnal *et al.*, 1993)，然后扩展到了具有连续值激活的网络 (Maass, 1992; Maass *et al.*, 1994)。许多现代神经网络使用整流线性单元。Leshno *et al.* (1993) 证明带有一大类非多项式激活函数族的浅层网络，包括整流线性单元，具有通用的近似性质，但是这些结果并没有强调深度或效率的问题——它们仅指出足够宽的整流网络能够表示任意函数。Montufar *et al.* (2014) 指出一些用深度整流网络表示的函数可能需要浅层网络（一个隐藏层）指数级的隐藏单元才能表示。更确切的说，他们说明分段线性网络（可以通过整流非线性或maxout 单元获得）可以表示区域的数量是网络深度的指数级的函数。图6.5解释了带有绝对值整流的网络是如何创建函数的镜像图像的，这些函数在某些隐藏单元的顶部计算，作用于隐藏单元的输入。每个隐藏单元指定在哪里折叠输入空间，来创造镜像响应（在绝对值非线性的两侧）。通过组合这些折叠操作，我们获得指数级的分段线性区域，他们可以概括所有种类的规则模式（例如，重复）。

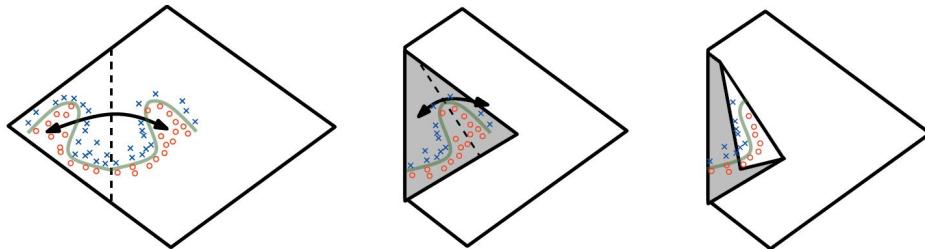


图 6.5: 关于更深的整流网络具有指数优势的一个直观的几何解释，来自 Montufar *et al.* (2014)。(左) 绝对值整流单元对其输入中的每对镜像点有相同的输出。镜像的对称轴由单元的权重和偏置定义的超平面给出。在该单元顶部计算的函数（绿色决策面）将是横跨该对称轴的更简单模式的一个镜像。(中) 该函数可以通过折叠对称轴周围的空间来得到。(右) 另一个重复模式可以在第一个的顶部折叠（由另一个下游单元）以获得另外的对称性（现在重复四次，使用了两个隐藏层）。图重置得到了 Montufar *et al.* (2014) 的许可。

更确切的说，Montufar *et al.* (2014) 的主要定理指出，具有  $d$  个输入，深度为

$l$ , 每个隐藏层有  $n$  个单元的深度整流网络可以描述的线性区域的数量是

$$O\left(\binom{n}{d}^{d(l-1)} n^d\right), \quad (6.42)$$

这意味着, 是深度  $l$  的指数级。在每个单元具有  $k$  个过滤器的 maxout 网络中, 线性区域的数量是

$$O(k^{(l-1)+d}). \quad (6.43)$$

当然, 我们不能保证在机器学习 (特别是 AI) 的应用中我们想要学得的函数类型享有这样的属性。

我们还可能出于统计原因来选择深度模型。任何时候, 当我们选择一个特定的机器学习算法时, 我们隐含地陈述了一些先验, 这些先验是关于算法应该学得什么样的函数的。选择深度模型默许了一个非常普遍的信念, 那就是我们想要学得的函数应该涉及几个更加简单的函数的组合。这可以从表示学习的观点来解释, 我们相信学习的问题包含发现一组潜在的变化因素, 它们可以根据其他更简单的潜在的变化因素来描述。或者, 我们可以将深度结构的使用解释为另一种信念, 那就是我们想要学得的函数是包含多个步骤的计算机程序, 其中每个步骤使用前一步骤的输出。这些中间输出不一定是变化的因素, 而是可以类似于网络用来组织其内部处理的计数器或指针。根据经验, 更深的模型似乎确实会更适用于广泛的任务 (Bengio *et al.*, 2007b; Erhan *et al.*, 2009; Bengio, 2009; Mesnil *et al.*, 2011; Ciresan *et al.*, 2012; Krizhevsky *et al.*, 2012a; Sermanet *et al.*, 2013; Farabet *et al.*, 2013; Couprie *et al.*, 2013; Kahou *et al.*, 2013; Goodfellow *et al.*, 2014d; Szegedy *et al.*, 2014a)。图6.6和图6.7展示了一些实验结果的例子。这表明使用深层结构确实在模型学习的函数空间上表示了一个有用的先验。

## 6.4.2 其他结构上的考虑

到目前为止, 我们都将神经网络描述成简单的层的链式结构, 主要考虑因素是网络的深度和每层的宽度。在实践中, 神经网络显示出相当的多样性。

许多神经网络结构已经被开发用于特定的任务。用于计算机视觉的卷积神经网络的特殊结构将在第九章中介绍。前馈网络也可以推广到用于序列处理的循环神经网络, 但有它们自己的结构考虑, 将在第十章中介绍。

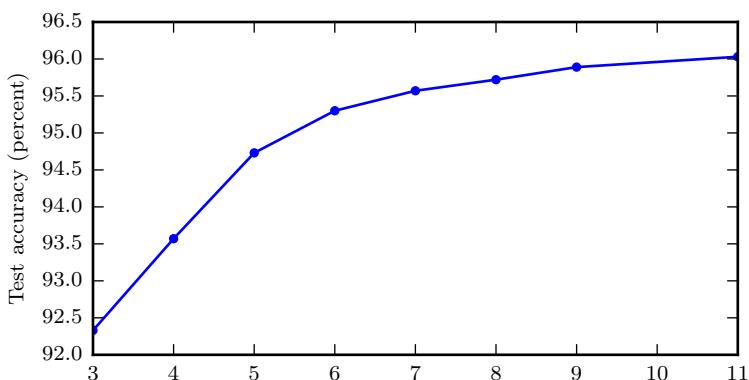


图 6.6: 深度的影响。实验结果表明, 当从地址照片转录多位数字时, 更深层的网络能够更好地泛化。数据来自Goodfellow *et al.* (2014d)。测试集上的准确率随着深度的增加而不断增加。图6.7给出了一个对照实验, 它说明了对模型尺寸其他方面的增加并不能产生相同的效果。

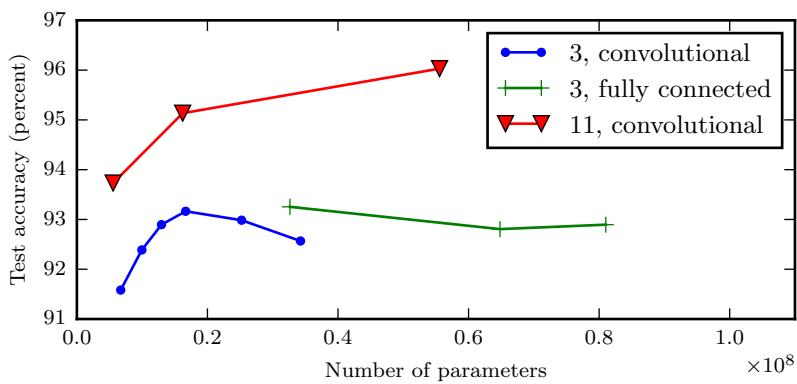


图 6.7: 参数数量的影响。更深的模型往往表现更好。这不仅仅是因为模型更大。Goodfellow *et al.* (2014d) 的这项实验表明, 增加卷积网络层中参数的数量, 但是不增加它们的深度, 在提升测试集性能方面几乎没有效果, 如此图所示。图例标明了用于画出每条曲线的网络深度, 以及曲线表示的是卷积层还是全连接层的大小变化。我们可以观察到, 在这种情况下, 浅层模型在参数数量达到 2000 万时就过拟合, 而深层模型在参数数量超过 6000 万时仍然表现良好。这表明, 使用深层模型表达出了对模型可以学习的函数空间的有用偏好。具体来说, 它表达了一种信念, 即该函数应该由许多更简单的函数复合在一起而得到。这可能导致学习由更简单的表示所组成的表示 (例如, 由边所定义的角) 或者学习具有顺序依赖步骤的程序 (例如, 首先定位一组对象, 然后分割它们, 之后识别它们)。

一般的，层不需要连接在链中，尽管这是最常见的做法。许多结构构建了一个主链，但随后又添加了额外的结构属性，例如从层  $i$  到层  $i+2$  或者更高层的跳动连接。这些跳动连接使得梯度更容易从输出层流向更接近输入的层。

结构设计考虑的另外一个关键点是如何将层与层之间连接起来。默认的神经网络层采用矩阵  $\mathbf{W}$  描述的线性变换，每个输入单元连接到每个输出单元。在之后章节中的许多专用网络具有较少的连接，使得输入层中的每个单元仅连接到输出层单元的一个小子集。这些用于减少连接数量的策略减少了参数的数量以及用于评估网络的计算量，但通常高度依赖于问题。例如，第九章描述的卷积神经网络使用对于计算机视觉问题非常有效的稀疏连接的专用模式。在这一章中，很难对通用神经网络的结构给出更多具体的建议。随后的章节研发了一些特殊的结构策略，可以在不同的领域工作良好。

## 6.5 反向传播和其他的微分算法

当我们使用前馈神经网络接收输入  $\mathbf{x}$  并产生输出  $\hat{\mathbf{y}}$  时，信息通过网络向前流动。输入  $\mathbf{x}$  提供初始信息，然后传播到每一层的隐藏单元，最终产生输出  $\hat{\mathbf{y}}$ 。这称之为前向传播 (forward propagation)。在训练过程中，前向传播可以持续向前直到它产生一个标量代价函数  $J(\theta)$ 。反向传播 (back propagation) 算法 (Rumelhart *et al.*, 1986c)，经常简称为**backprop**，允许来自代价函数的信息通过网络向后流动，以便计算梯度。

计算梯度的解析表达式是很直观的，但是数值化地求解这样的表达式在计算上可能是昂贵的。反向传播算法使用简单和廉价的程序来实现这个目标。

反向传播这个术语经常被误解为用于多层神经网络的整个学习算法。实际上，反向传播仅指用于计算梯度的方法，而另一种算法，例如随机梯度下降，使用该梯度来进行学习。此外，反向传播经常被误解为仅适用于多层神经网络，但是原则上它可以计算任何函数的导数（对于一些函数，正确的响应是报告函数的导数是未定义的）。特别地，我们会描述如何计算一个任意函数  $f$  的梯度  $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y})$ ，其中  $\mathbf{x}$  是一组变量，我们需要它们的导数，而  $\mathbf{y}$  是另外一组函数的输入变量，但我们并不需要它们的导数。在学习算法中，我们最常需要的梯度是成本函数关于参数的梯度，即  $\nabla_{\theta} J(\theta)$ 。许多机器学习任务涉及计算其他导数，作为学习过程的一部分，或者用来分析学习的模型。反向传播算法也适用于这些任务，并且不限于计算成本函数关于参

数的梯度。通过网络传播信息来计算导数的想法是非常通用的，并且可以用于计算诸如具有多个输出的函数  $f$  的 Jacobi 矩阵的值。我们这里描述的是最常用的情况， $f$  只有单个输出。

### 6.5.1 计算图

到目前为止我们已经用相对非正式的图形语言讨论了神经网络。为了更精确地描述反向传播算法，使用更精确的计算图 (computational graph) 语言是很有帮助的。

将计算形式化为图形的方法有很多。

这里，我们使用图中的每一个节点来表示一个变量。变量可以是标量、向量、矩阵、张量、或者甚至是另一类型的变量。

为了形式化我们的图形，我们还需要引入操作 (operation)。操作是一个或多个变量的简单函数。我们的图形语言伴随着一组被允许的操作。可以通过将多个操作组合在一起描述比该组中的操作更复杂的函数。

不失一般性，我们定义一个操作仅返回单个输出变量。这并没有失去一般性，因为输出变量可以有多个条目，例如向量。反向传播的软件实现通常支持具有多个输出的操作，但是我们在描述中避免这种情况，因为它引入了对概念理解不重要的许多额外细节。

如果变量  $y$  是变量  $x$  通过一个操作计算得到的，那么我们画一条从  $x$  到  $y$  的有向边。我们有时用操作的名称来注释输出的节点，当上下文很明确时有时也会省略这个标注。

计算图的实例可以参见图6.8。

### 6.5.2 微积分中的链式法则

微积分中的链式法则（为了不与概率中的链式法则相混淆）用于计算复合函数的导数。反向传播是一种计算链式法则的算法，使用高效的具体运算顺序。

设  $x$  是实数， $f$  和  $g$  是从实数映射到实数的函数。假设  $y = g(x)$  并且  $z = f(g(x)) = f(y)$ 。那么链式法则是说

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}. \quad (6.44)$$

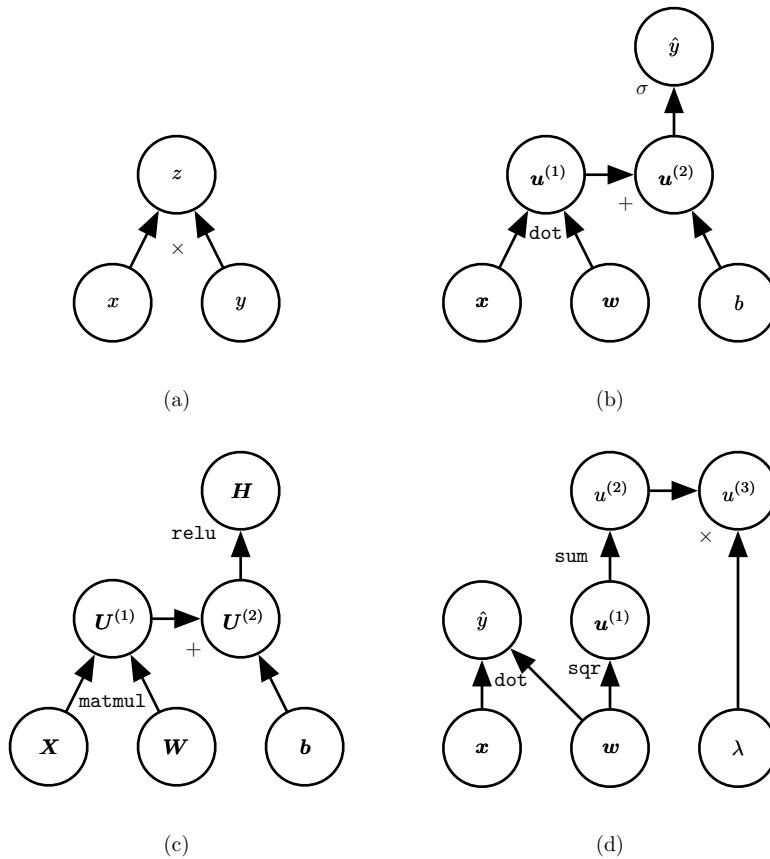


图 6.8: 一些计算图的示例。(a) 使用  $\times$  操作计算  $z = xy$  的图。(b) 用于逻辑回归预测  $\hat{y} = \sigma(x^\top w + b)$  的图。一些中间表达式在代数表达式中没有名称，但在图形中却需要。我们简单地将第  $i$  个这样的变量命名为  $u^{(i)}$ 。(c) 表达式  $H = \max\{0, \mathbf{X}\mathbf{W} + b\}$  的计算图，在给定包含小批量输入数据的设计矩阵  $\mathbf{X}$  时，它计算整流线性单元激活的设计矩阵  $\mathbf{H}$ 。(d) 示例 a-c 对每个变量最多只实施一个操作，但是对变量实施多个操作也是可能的。这里我们展示一个计算图，它对线性回归模型的权重  $w$  实施多个操作。这个权重不仅用于预测  $\hat{y}$ ，也用于权重衰减罚项  $\lambda \sum_i w_i^2$ 。

我们可以将这种标量情况进行扩展。假设  $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$ ,  $g$  是从  $\mathbb{R}^m$  到  $\mathbb{R}^n$  的映射,  $f$  是从  $\mathbb{R}^n$  到  $\mathbb{R}$  的映射。如果  $\mathbf{y} = g(\mathbf{x})$  并且  $z = f(\mathbf{y})$ , 那么

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}. \quad (6.45)$$

使用向量记法，可以等价地写成

$$\nabla_{\mathbf{x}} z = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^{\top} \nabla_{\mathbf{y}} z, \quad (6.46)$$

这里  $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$  是  $g$  的  $n \times m$  的 Jacobi 矩阵。

从这里我们看到变量  $\mathbf{x}$  的梯度可以通过将 Jacobi 矩阵  $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$  是  $g$  的  $n \times m$  乘以梯度  $\nabla_{\mathbf{y}} z$  来得到。反向传播算法由图中每一个这样的 Jacobi 梯度的乘积操作所组成。

通常我们不将反向传播算法仅用于向量，而是应用于任意维度的张量。从概念上讲，这与使用向量的反向传播完全相同。唯一的区别是如何将数字排列成网格以形成张量。我们可以想象，在我们运行反向传播之前，将每个张量变平为一个向量，计算一个向量值梯度，然后将该梯度重新构造成一个张量。从这种重新排列的观点上看，反向传播仍然只是将 Jacobi 矩阵乘以梯度。

为了表示值  $z$  对张量  $\mathbf{X}$  的梯度，我们记为  $\nabla_{\mathbf{X}} z$ ，就像  $\mathbf{X}$  是向量一样。 $\mathbf{X}$  的索引现在有多个坐标——例如，一个 3 维的张量由三个坐标索引。我们可以通过使用单个变量  $i$  来表示完整的索引元组，从而完全抽象出来。对所有可能的元组  $i$ ， $(\nabla_{\mathbf{X}} z)_i$  给出  $\frac{\partial z}{\partial X_i}$ 。这与向量中索引的方式完全一致， $(\nabla_{\mathbf{x}} z)_i$  给出  $\frac{\partial z}{\partial x_i}$ 。使用这种记法，我们可以写出适用于张量的链式法则。如果  $\mathbf{Y} = g(\mathbf{X})$  并且  $z = f(\mathbf{Y})$ ，那么

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}. \quad (6.47)$$

### 6.5.3 递归地使用链式法则来实现 BP

使用链式规则，可以直接写出某个标量对于计算图中任何产生该标量的节点的梯度的代数表达式。然而，实际在计算机中计算该表达式时会引入一些额外的考虑。

具体来说，许多子表达式可能在梯度的整个表达式中重复若干次。任何计算梯度的程序都需要选择是存储这些子表达式还是重新计算它们几次。图6.9给出了一个例子来说明这些重复的子表达式是如何出现的。在某些情况下，计算两次相同的子表达式纯粹是浪费。在复杂图中，可能存在指数多的这种计算上的浪费，使得简单的链式法则不可实现。在其他情况下，计算两次相同的子表达式可能是以较高的运行时间为代价来减少内存开销的有效手段。

我们首先给出一个版本的反向传播算法，它指明了梯度的直接计算方式（算法6.2以及相关的正向计算的算法6.1），按照它实际完成的顺序并且递归地使用链式