

# CS229 课程讲义

Andrew Ng (吴恩达) and Tengyu Ma (马腾宇)

由 Namoe 翻译

June 2025

# 目录

<b>第一部分 监督学习</b>	<b>5</b>
<b>第 1 章 线性回归</b>	<b>8</b>
1.1 最小均方算法 . . . . .	9
1.2 正规方程 . . . . .	12
1.2.1 矩阵导数 . . . . .	12
1.2.2 再探最小二乘法 . . . . .	13
1.3 概率解释 . . . . .	15
1.4 局部加权线性回归（选读） . . . . .	16
<b>第 2 章 分类与逻辑回归</b>	<b>19</b>
2.1 逻辑回归 . . . . .	19
2.2 离题：感知器学习算法 . . . . .	22
2.3 多类别分类 . . . . .	23
2.4 最大化 $\ell(\theta)$ 的另一种算法 . . . . .	25
<b>第 3 章 广义线性模型</b>	<b>27</b>
3.1 指数族 . . . . .	27
3.2 构造广义线性模型 . . . . .	29
3.2.1 普通最小二乘 . . . . .	30
3.2.2 逻辑回归 . . . . .	30
<b>第 4 章 生成式学习算法</b>	<b>32</b>
4.1 高斯判别分析 . . . . .	33
4.1.1 多元正态分布 . . . . .	33
4.1.2 高斯判别分析模型 . . . . .	35
4.1.3 讨论：GDA 与逻辑回归 . . . . .	36
4.2 朴素贝叶斯（选读） . . . . .	37

4.2.1 拉普拉斯平滑 . . . . .	40
4.2.2 文本分类的事件模型 . . . . .	41
<b>第 5 章 核方法</b>	<b>44</b>
5.1 特征映射 . . . . .	44
5.2 带特征的最小均方 . . . . .	45
5.3 带核技巧的最小均方 . . . . .	45
5.4 核的性质 . . . . .	48
<b>第 6 章 支持向量机</b>	<b>53</b>
6.1 间隔：直觉 . . . . .	53
6.2 符号（选读） . . . . .	54
6.3 函数间隔与几何间隔（选读） . . . . .	55
6.4 最优间隔分类器（选读） . . . . .	56
6.5 拉格朗日对偶性（选读） . . . . .	58
6.6 最优间隔分类器：对偶形式（选读） . . . . .	61
6.7 正则化与非线性可分情况（选读） . . . . .	63
6.8 SMO 算法（选读） . . . . .	65
6.8.1 坐标上升 . . . . .	65
6.8.2 SMO . . . . .	66
<b>第二部分 深度学习</b>	<b>69</b>
<b>第 7 章 深度学习</b>	<b>70</b>
7.1 使用非线性模型的监督学习 . . . . .	70
7.2 神经网络 . . . . .	73
7.3 现代神经网络的模块 . . . . .	81
7.4 反向传播 . . . . .	85
7.4.1 偏导数初步 . . . . .	86
7.4.2 反向传播的通用策略 . . . . .	88
7.4.3 基本模块的后向函数 . . . . .	91
7.4.4 MLP 的反向传播 . . . . .	92
7.4.5 训练样本的向量化 . . . . .	94

<b>第三部分 泛化与正则化</b>	<b>97</b>
<b>第 8 章 泛化</b>	<b>98</b>
8.1 偏差-方差均衡 . . . . .	99
8.1.1 (对于回归问题的) 数学分解 . . . . .	103
8.2 双下降现象 . . . . .	105
8.3 样本复杂度边界 (选读) . . . . .	108
8.3.1 预备知识 . . . . .	108
8.3.2 有限 $\mathcal{H}$ 的情况 . . . . .	110
8.3.3 无限 $\mathcal{H}$ 的情况 . . . . .	112
<b>第 9 章 正则化与模型选择</b>	<b>116</b>
9.1 正则化 . . . . .	116
9.2 隐式正则化效应 (选读) . . . . .	117
9.3 通过交叉验证选择模型 . . . . .	119
9.4 贝叶斯统计与正则化 . . . . .	121
<b>第四部分 无监督学习</b>	<b>124</b>
<b>第 10 章 聚类与 k-means 算法</b>	<b>125</b>
<b>第 11 章 EM 算法</b>	<b>128</b>
11.1 面向高斯混合模型的 EM 算法 . . . . .	128
11.2 Jensen 不等式 . . . . .	130
11.3 广义 EM 算法 . . . . .	131
11.3.1 ELBO 的另一个解释 . . . . .	136
11.4 回顾高斯混合模型 . . . . .	136
11.5 变分推断与变分自编码器 (选读) . . . . .	138
<b>第 12 章 主成分分析 (PCA)</b>	<b>142</b>
<b>第 13 章 独立成分分析 (ICA)</b>	<b>147</b>
13.1 ICA 的不确定性 . . . . .	148
13.2 密度与线性变换 . . . . .	149
13.3 ICA 算法 . . . . .	150

<b>第 14 章 自监督学习与基础模型</b>	<b>152</b>
14.1 预训练与适配 . . . . .	152
14.2 计算机视觉中的预训练方法 . . . . .	154
14.3 预训练的大语言模型 . . . . .	155
14.3.1 零样本学习与语境学习 . . . . .	157
<b>第五部分 强化学习与控制</b>	<b>159</b>
<b>第 15 章 强化学习</b>	<b>160</b>
15.1 马尔可夫决策过程 . . . . .	160
15.2 价值迭代与策略迭代 . . . . .	163
15.3 学习 MDP 的模型 . . . . .	164
15.4 连续状态 MDP . . . . .	166
15.4.1 离散化 . . . . .	166
15.4.2 价值函数近似 . . . . .	169
15.5 策略与价值的联系 (选读) . . . . .	173
<b>第 16 章 LQR, DDP 与 LQG</b>	<b>175</b>
16.1 有限时间范围的 MDP . . . . .	175
16.2 线性二次调节器 (LQR) . . . . .	178
16.3 从非线性动态过程到 LQR . . . . .	180
16.3.1 动态过程的线性化 . . . . .	181
16.3.2 微分动态规划 (DDP) . . . . .	181
16.4 线性二次高斯 (LQG) . . . . .	183
<b>第 17 章 策略梯度 (REINFORCE)</b>	<b>186</b>

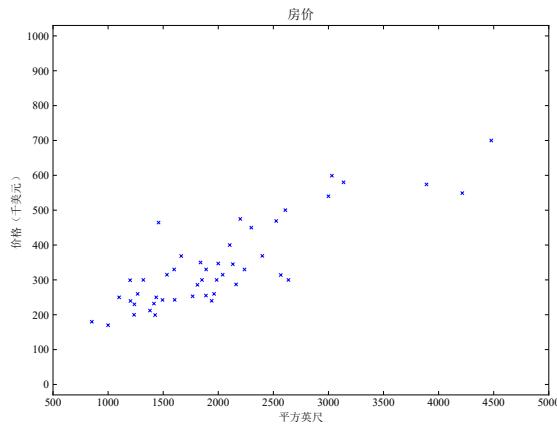
# **第一部分**

## **监督学习**

不妨先从监督学习的几个例子谈起。假设有一个记录了俄勒冈州波特兰市 47 套房屋的居住面积和价格的数据集：

居住面积 (平方英尺)	价格 (1000\$)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

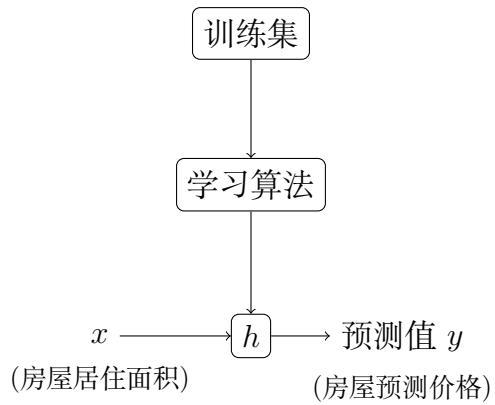
将这些数据绘制出来：



有了这些数据之后，该怎样根据波特兰其他房屋的居住面积来预测其价格呢？

为了后续使用的方便，在这里做如下约定。约定用  $x^{(i)}$  表示“输入”变量（示例中是居住面积），也称作输入**特征 (features)**；用  $y^{(i)}$  表示要预测的“输出”或**目标 (target)** 变量（价格）。一对  $(x^{(i)}, y^{(i)})$  称为一个**训练样本 (training example)**，而用于学习的数据集——由  $n$  个训练样本组成的列表  $\{(x^{(i)}, y^{(i)}); i = 1, \dots, n\}$ ——则称为**训练集 (training set)**。注意，此处的上标 “ $i$ ” 仅表示训练集中的索引，而不表示指数运算。此外，用  $\mathcal{X}$  表示输入的取值空间， $\mathcal{Y}$  表示输出的取值空间。在本例中，有  $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ 。

监督学习问题可以更加形式化地表述为：给定一个训练集，目标是学习一个函数  $h : \mathcal{X} \mapsto \mathcal{Y}$ ，该函数能够对输入  $x$  进行预测，使其输出  $h(x)$  与“很好地”预测相应的真实值  $y$ 。出于历史原因，函数  $h$  被称为**假设 (hypothesis)**。整个过程如下图所示：



当预测的目标变量是连续值时（例如预测房价），称这类学习问题为**回归 (regression)** 问题。当  $y$  只能取有限个离散值时（例如根据居住面积预测住宅是房屋还是公寓），则称为**分类 (classification)** 问题。

# 第 1 章 线性回归

为了让上面的房屋示例更有趣，不妨考虑一个更为丰富的数据集。除了居住面积外，该数据集还包括了每栋房屋的卧室数量：

居住面积 (平方英尺)	卧室数	价格 (1000\$)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
:	:	:

此处， $x$  是  $\mathbb{R}^2$  中的二维向量。对于训练集中第  $i$  栋房屋， $x_1^{(i)}$  是其居住面积，而  $x_2^{(i)}$  是其卧室数量。（在设计学习问题时，特征的选择通常取决于你的具体需求。例如，在收集波特兰的房屋数据时，除了居住面积和卧室数量，还可以考虑纳入壁炉、浴室数量等其他特征。关于特征选择的深入讨论将在后续展开，目前先基于当前给定的两个特征进行分析。）

在进行监督学习时，需要明确如何在计算机中表示假设函数  $h$ 。不妨先尝试用  $x$  的线性函数来近似  $y$ ：

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

在此处， $\theta_i$  是该模型的参数 (parameters)，亦称权重 (weights)。它们参数化了从特征空间  $\mathcal{X}$  到目标空间  $\mathcal{Y}$  的线性函数。在不引起混淆的前提下，可以省略  $h_{\theta}(x)$  中的下标  $\theta$ ，直接写作  $h(x)$ 。为了进一步简化表示，我们引入约定：令  $x_0 = 1$ 。这个  $x_0$  对应的系数  $\theta_0$  通常被称为截距项 (intercept term)。这样就有

$$h(x) = \sum_{i=0}^d \theta_i x_i = \theta^T x,$$

其中  $\theta$  和  $x$  视为向量，而  $d$  则是输入变量的数量（不计入  $x_0$ ）。

现在，对于给定的训练集，我们应该如何选择或学习参数  $\theta$  呢？一个直观且合理的方法是，使假设函数  $h(x)$  对于训练样本的输出  $h_\theta(x^{(i)})$  尽可能地接近其对应的真实值  $y^{(i)}$ 。为了形式化地表述这个接近程度，我们定义一个函数，用于衡量对于任意给定的参数值  $\theta$ ，预测值  $h_\theta(x^{(i)})$  与实际值  $y^{(i)}$  之间的差异。这个函数被称为**代价函数**（cost function）：

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_\theta(x^{(i)}) - y^{(i)})^2.$$

熟悉线性回归的读者可能会发现，此处定义的函数即为**普通最小二乘**（ordinary least squares）回归模型所使用的最小二乘代价函数。但本讲义不要求读者具备相关背景知识，后文将对此进行详细阐述，并最终指出这仅是更广泛算法族中的一个特例。

## 1.1 最小均方算法

我们的目标是找到能够最小化代价函数  $J(\theta)$  的参数  $\theta$ 。为此，我们可以考虑一种搜索算法：该算法从对  $\theta$  的某个“初始猜测”开始，然后不断地调整  $\theta$  的值，使其沿着使  $J(\theta)$  减小的方向移动，直到最终收敛到最小化  $J(\theta)$  的  $\theta$  值。具体而言，我们考虑使用**梯度下降**（gradient descent）算法。该算法从某个初始的  $\theta$  值开始，并重复执行以下更新步骤：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

（上述更新操作同时应用于  $j = 0, \dots, d$  的所有参数  $\theta_j$ 。）这里的  $\alpha$  被称为**学习率**（learning rate）。这是一种非常自然的算法：它每一步都沿着代价函数  $J$  下降最快的方向进行更新。

为了实现上述算法，需要计算公式右侧的偏导数项。可以先考虑只有一个训练样本  $(x, y)$  的情况，这样就可以暂时忽略代价函数  $J$  定义中的求和操作。在这种情况下，偏导数计算如下：

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left( \sum_{i=0}^d \theta_i x_i - y \right) \\
&= (h_\theta(x) - y) x_j
\end{aligned}$$

上式给出了针对单个训练样本的更新规则：<sup>1</sup>

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}.$$

这个规则被称为**最小均方 (Least mean squares, LMS)** 更新规则，也称为**Widrow-Hoff** 学习规则。该规则具有一些自然而直观的特性。例如，更新的幅度与误差 (error) 项 ( $y^{(i)} - h_\theta(x^{(i)})$ ) 成正比；因此，如果对于一个训练样本，其预测值几乎等于  $y^{(i)}$  的实际值，那么参数就几乎不需要调整；反之，如果预测的  $h_\theta(x^{(i)})$  有很大的误差 (即与  $y^{(i)}$  相差甚远)，则需要对参数进行更大的调整。

所推导的 LMS 规则是针对只有一个训练样本的情况。要将其应用于包含多个样本的训练集，有两种常见的方法。第一种方法是将算法修改为以下形式：

重复直到收敛 {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^n (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}, \text{(对于每个 } j \text{)} \quad (1.1)$$

}

将逐位置的更新向量化到  $\theta$ ，可以稍微简化 (1.1)：

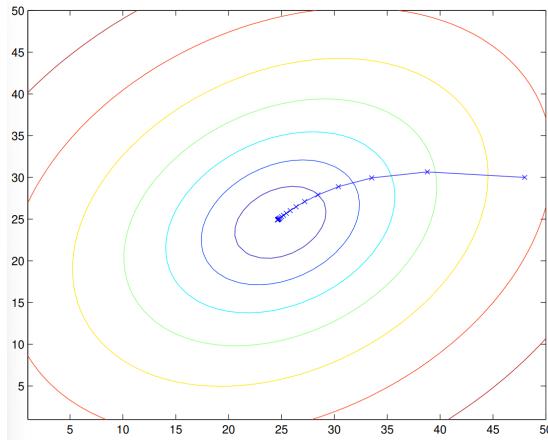
$$\theta := \theta + \alpha \sum_{i=1}^n (y^{(i)} - h_\theta(x^{(i)})) x^{(i)}$$

读者不难验证，上述更新规则中求和项所表示的量，恰好对应于我们先前定义的成本函数的偏导数  $\partial J(\theta)/\partial \theta_j$ 。因此，这个更新规则实际上就是在原始成本函数  $J$  上进行梯度下降。这种方法在每一步都利用了整个训练集的所有样本，因此被称为**批量梯度下降 (batch gradient descent)**。值得注意的是，尽管梯度下降算法在一般情况下可能收敛到局部最优解，但对于我们的线性回归问题，

---

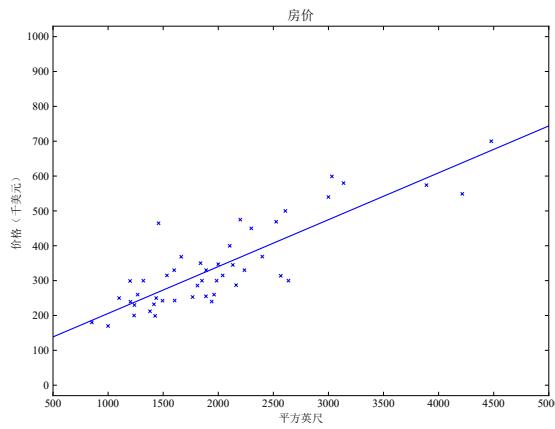
<sup>1</sup> 符号 “ $a := b$ ” 用于表示 (计算机程序中的) 一个操作，其中变量  $a$  的值被设置为  $b$ 。换句话说，这个操作用  $b$  的值覆盖了  $a$  的值。反之，如果需要断言  $a$  的值等于  $b$  的值，会写作 “ $a = b$ ”。

其优化目标函数  $J$  具有良好的性质：它是一个凸二次函数。这意味着它只有一个全局最小值，没有其他局部最优解。因此，在合适的学习率  $\alpha$  下，梯度下降算法能够保证收敛到全局最优解。下面是一个用梯度下降最小化一个二次函数的图例。



上面的椭圆是二次函数的等高线。图中还显示了梯度下降的轨迹，其初始化参数是  $(48,30)$ ，而由直线连接的叉号  $x$  则是梯度下降所经过的一系列参数  $\theta$  值。

在之前的数据集上应用批量梯度下降算法来拟合参数  $\theta$ ，以学习根据居住面积预测房价的函数，最终得到的参数值为  $\theta_0 = 71.27$  和  $\theta_1 = 0.1345$ 。将学习到的函数  $h_\theta(x)$ ，作为输入变量  $x$ （表示居住面积）的函数，与训练数据一同绘制，结果如下图所示：



如果把卧室数量也当作输入特征，最终得到的参数值为  $\theta_0 = 89.60, \theta_1 = 0.1392, \theta_2 = -8.738$ 。

上述结果是通过批量梯度下降算法得到的。除此之外，还有一种很好的替代算法：

循环 {

对于  $i = 1$  到  $n$ , {

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}, (\text{对于每个 } j) \quad (1.2)$$

}

}

将逐位置的更新向量化到  $\theta$ , 可以稍微简化 (1.2):

$$\theta := \theta + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$$

这种算法会重复遍历训练集, 每次遇到一个训练样本时, 仅针对该单个样本计算误差梯度并更新参数。这种方法被称为**随机梯度下降 (stochastic gradient descent)**, 有时也称为**增量梯度下降 (incremental gradient descent)**。与批量梯度下降不同, 批量梯度下降在执行单次更新前需要扫描整个训练集 (当训练集规模  $n$  很大时, 这是一项昂贵的操作), 而随机梯度下降可以立即开始并对每个样本都取得进展。通常情况下, 随机梯度下降能更快地使参数  $\theta$  “接近” 最小值。(然而, 需要注意的是, 它可能不会完全“收敛”到最小值, 参数  $\theta$  可能在目标函数  $J(\theta)$  的最小值附近持续振荡。但在实际应用中, 接近最小值的大多数值都足以作为真实最小值的良好近似。<sup>2</sup>) 因此, 特别是在训练集很大时, 通常更倾向于使用随机梯度下降而非批量梯度下降。

## 1.2 正规方程

梯度下降提供了一种最小化目标函数  $J$  的迭代方法。接下来, 我们将探讨另一种无需迭代的最小化  $J$  的方法。具体而言, 我们将通过明确地计算  $J$  关于每个参数  $\theta_j$  的偏导数, 并将这些导数置为零来求解最小值。为了避免繁琐的代数运算和大量的导数矩阵书写, 我们将在下文引入一些矩阵微积分的记号。

### 1.2.1 矩阵导数

对于一个将  $n \times d$  矩阵映射到实数的函数  $f : \mathbb{R}^{n \times d} \mapsto \mathbb{R}$ , 我们定义  $f$  对  $A$  的导数:

---

<sup>2</sup>通过在算法运行过程中缓慢地减小学习率  $\alpha$  至零, 可以确保参数收敛到全局最小值, 而不仅仅是在最小值附近振荡。

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{nd}} \end{bmatrix}$$

因此，梯度  $\nabla_A f(A)$  本身是一个  $n \times d$  矩阵，其  $(i, j)$  元素是  $\partial f / \partial A_{ij}$ 。例如，假设  $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$  是一个  $2 \times 2$  矩阵，并且函数  $f : \mathbb{R}^{2 \times 2} \mapsto \mathbb{R}$  由下式给出

$$f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}.$$

这里， $A_{ij}$  表示矩阵  $A$  在  $(i, j)$  位置上的元素。则可以得到

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}.$$

### 1.2.2 再探最小二乘法

掌握了矩阵导数的工具后，我们现在可以着手求解使目标函数  $J(\theta)$  最小化的  $\theta$  的闭式解。首先，我们用矩阵向量符号重写  $J$ 。

给定一个训练集，我们定义**设计矩阵** (design matrix)  $X$ 。这是一个  $n \times d$  矩阵（如果包含截距项，则为  $n \times (d + 1)$  矩阵），其每一行对应一个训练样本的输入特征向量：

$$X = \begin{bmatrix} -(x^{(1)})^T - \\ -(x^{(2)})^T - \\ \vdots \\ -(x^{(n)})^T - \end{bmatrix}.$$

进一步地，我们定义向量  $\vec{y}$ ，它是一个  $n$  维列向量，其分量依次为训练集中各个样本的目标值：

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

根据  $h_\theta(x^{(i)}) = (x^{(i)})^T \theta$ , 不难验证

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T \theta \\ \vdots \\ (x^{(n)})^T \theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(n)}) - y^{(n)} \end{bmatrix}. \end{aligned}$$

利用向量  $z$  满足  $z^T z = \sum_i z_i^2$  这一性质, 可以得到

$$\begin{aligned} \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2} \sum_{i=1}^n (h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

最后, 为了最小化  $J$ , 对其关于  $\theta$  求导, 得到:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\ &= \frac{1}{2} \nabla_\theta ((X\theta)^T X\theta - (X\theta)^T \vec{y} - \vec{y}^T (X\theta) + \vec{y}^T \vec{y}) \\ &= \frac{1}{2} \nabla_\theta (\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta) \\ &= \frac{1}{2} \nabla_\theta (\theta^T X^T X\theta - 2(X^T \vec{y})^T \theta) \\ &= \frac{1}{2} (2X^T X\theta - 2X^T \vec{y}) \\ &= X^T X\theta - X^T \vec{y} \end{aligned}$$

在上述推导中, 第三步利用了向量内积的交换律  $a^T b = b^T a$ ; 第五步则利用了向量求导公式  $\nabla_x b^T x = b$  以及对于对称矩阵  $A$ ,  $\nabla_x x^T A x = 2Ax$  (详细推导可参考第 4.3 节的“线性代数回顾与参考”)。为了最小化  $J$ , 我们将上述导数设为零, 从而得到**正规方程 (normal equations)**:

$$X^T X\theta = X^T \vec{y}$$

因此, 使  $J(\theta)$  最小的  $\theta$  的闭式解是

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$


---

<sup>3</sup>需要注意的是, 上述推导隐式假设了  $X^T X$  是一个可逆矩阵。在计算其逆矩阵之前, 应先进行可逆性检查。当线性独立样本的数量少于特征数量, 或者特征之间存在线性相关性时,  $X^T X$  将是不可逆的。即使在这种情况下, 也可以通过其他技术来“修复”, 但为了保持简洁, 此处省略。

### 1.3 概率解释

在面对回归问题时，我们通常会采用线性回归模型，并以最小化平方误差和（即最小二乘代价函数  $J$ ）作为学习目标。本节旨在解释为什么这种方法是合理的，并将阐明在哪些特定的概率假设下，最小二乘回归可以自然地从概率模型中推导出来。

假设目标变量与输入变量之间的关系可以通过以下方程进行建模：

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)},$$

此处  $\epsilon^{(i)}$  表示一个误差项，它包含了模型中未建模的因素（例如，一些对预测结果有显著影响但未纳入模型的特征）以及固有的随机噪声。我们进一步假定这些误差项  $\epsilon^{(i)}$  是独立同分布 (IID) 的，并且都服从均值为零、方差为  $\sigma^2$  的高斯分布（也称为正态分布），写作 “ $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$ ”。由此， $\epsilon^{(i)}$  的概率密度函数可以写为：

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right).$$

这意味着

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right).$$

符号 “ $p(y^{(i)}|x^{(i)}; \theta)$ ” 表示这是给定  $x^{(i)}$  的条件下，由  $\theta$  参数化的  $y^{(i)}$  的分布。需要注意的是，我们不应该将  $\theta$  作为条件（即写成 “ $p(y^{(i)}|x^{(i)}, \theta)$ ”），因为这里  $\theta$  被视为一个未知但固定的值，而非一个随机变量。此外，也可以将  $y^{(i)}$  的分布写成  $y^{(i)}|x^{(i)}; \theta \sim \mathcal{N}(\theta^T x^{(i)}, \sigma^2)$ 。

给定设计矩阵  $X$ （包含所有输入向量  $x^{(i)}$ ）和参数  $\theta$ ，我们就可以确定每个观测值  $y^{(i)}$  的条件分布。因此，整个数据集的概率可以表示为  $p(\vec{y}|X; \theta)$ 。当我们把  $\theta$  视为固定值时，这个概率是关于观测数据  $\vec{y}$ （可能还有  $X$ ）的函数。然而，在推断模型参数时，我们更关注的是在给定观测数据  $\vec{y}$  和输入数据  $X$  的情况下，不同参数值  $\theta$  的可能性。此时，我们将  $p(\vec{y}|X; \theta)$  视为关于  $\theta$  的函数，并称之为似然函数 (likelihood function)：

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X; \theta).$$

需要注意的是，根据  $\epsilon^{(i)}$  的独立性假设（这隐含了在给定  $x^{(i)}$  条件下  $y^{(i)}$  的独立性），上式也可以写成

$$\begin{aligned} L(\theta) &= \prod_{i=1}^n p(y^{(i)}|x^{(i)}; \theta) \\ &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right). \end{aligned}$$

现在，给定这个描述  $y^{(i)}$  和  $x^{(i)}$  之间关系的概率模型，我们应该怎么得到最优的参数  $\theta$ ? **最大似然 (maximum likelihood)** 原理表明，应该选能使观测数据出现的概率尽可能高的参数  $\theta$ 。换句话说，我们应该选择能够最大化似然函数  $L(\theta)$  的  $\theta$  值。

最大化  $L(\theta)$  等价于最大化  $L(\theta)$  的任何严格递增函数。特别地，如果选择最大化对数似然 (log likelihood)  $\ell(\theta)$ ，推导过程会更加简化：

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^n \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^n \left( \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \right) \\ &= n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2.\end{aligned}$$

因此，最大化  $\ell(\theta)$  与最小化

$$\frac{1}{2} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2,$$

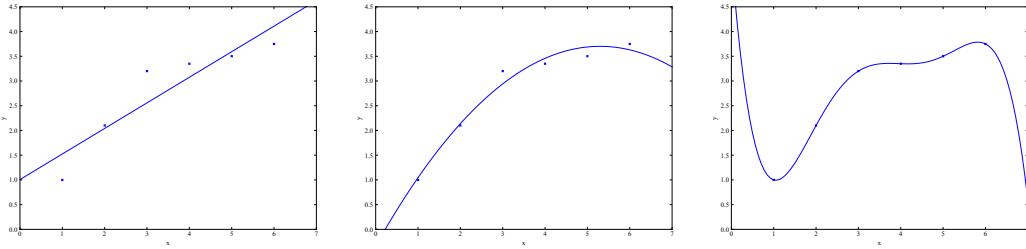
结果相同，而这正是  $J(\theta)$ ，也即最初的小二乘代价函数。

总结来说，在我们先前对数据分布做出的概率假设下，最小二乘回归实际上对应于求解参数  $\theta$  的最大似然估计。因此，这些概率假设构成了一组能够证明最小二乘回归是一种非常自然的最大似然估计方法的条件。(然而需要注意的是，这些概率假设并非使得最小二乘回归成为一个合理有效方法的必要条件，事实上也存在其他自然的假设能够证明其合理性。)

另外需要指出的是，在之前的推导中，对参数  $\theta$  的最终选择并不依赖于  $\sigma^2$  的具体数值，即使  $\sigma^2$  未知，我们也能得到相同的结果。这一特性在后续讨论指数族和广义线性模型时将会再次得到利用。

## 1.4 局部加权线性回归（选读）

考虑从  $x \in \mathbb{R}$  预测  $y$  的问题。下图最左边的图显示了将  $y = \theta_0 + \theta_1 x$  拟合到数据集的结果。从图中可以看出，这些数据点并未完全落在一条直线上，因此拟合效果并不理想。



作为对比，如果我们添加一个额外的特征  $x^2$ ，然后拟合模型  $y = \theta_0 + \theta_1 x + \theta_2 x^2$ ，那么对数据的拟合效果可能会有所改善（参见中间图）。有人可能会简单地认为添加的特征越多越好。然而，过度添加特征也存在风险：最右边的图展示了拟合一个五阶多项式  $y = \sum_{j=0}^5 \theta_j x^j$  的结果。尽管这条拟合曲线完美地穿过了所有数据点，我们也不能期望它能很好地预测不同居住区域 ( $x$ ) 的房价 ( $y$ )。非正式地借用一下拟合的术语，可以说左边的图是欠拟合 (underfitting) 的一个例子——模型未能捕捉到数据中明显的结构——而右边的图则是一个过拟合 (overfitting) 的例子。（在课程后续的学习理论部分，我们将正式定义这些概念，并更严谨地探讨判断一个假设优劣的标准。）

正如先前讨论的，特征的选择对于确保学习算法的良好性能至关重要。（在后续关于模型选择的讨论中，我们也会介绍一些能够自动选择合适特征的算法。）在本节中，我们将简要介绍局部加权线性回归 (locally weighted linear regression, LWR) 算法。该算法假设有足够的训练数据，使得特征的选择不那么关键。鉴于读者将在作业中自行探索 LWR 算法的一些特性，本节的讲解将较为简略。

在原始的线性回归算法中，为了使用输入  $x$  进行预测（即计算  $h(x)$  的值），通常需要执行以下步骤：

1. 拟合  $\theta$  以最小化  $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$ 。
2. 输出  $\theta^T x$ 。

相比之下，局部加权线性回归算法执行以下步骤：

1. 拟合  $\theta$  以最小化  $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$ 。
2. 输出  $\theta^T x$ 。

这里， $w^{(i)}$  是非负的权重 (weights)。直观上，对于特定的训练样本  $i$ ，如果  $w^{(i)}$  较大，则在确定参数  $\theta$  时，模型会更倾向于使  $(y^{(i)} - \theta^T x^{(i)})^2$  误差项尽可能小。反之，如果  $w^{(i)}$  较小，则该误差项在拟合过程中基本上会被忽略。

一种常用的权重选择方法是<sup>4</sup>

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right).$$

需要注意的是，权重取决于要评估的特定点  $x$ 。此外，如果  $|x^{(i)} - x|$  的值很小，则  $w^{(i)}$  接近 1；如果  $|x^{(i)} - x|$  的值很大，则  $w^{(i)}$  会很小。因此，在选择  $\theta$  时，靠近查询点  $x$  的训练样本（即其误差项  $y^{(i)} - \theta^T x^{(i)}$ ）会被赋予更高的“权重”。（另外需要说明的是，尽管权重的表达式形式上类似于高斯分布的概率密度函数，但  $w^{(i)}$  与高斯分布并没有直接联系，特别是  $w^{(i)}$  并非随机变量，无论是正态分布还是其他分布。）参数  $\tau$  控制着训练样本的权重随其与查询点  $x$  距离衰减的速度； $\tau$  被称为**带宽 (bandwidth)** 参数，这也是在作业中需要进行实验的内容。

局部加权线性回归是我们遇到的第一个**非参数 (non-parametric)** 算法示例。之前讨论的（无权重）线性回归算法被称为**参数化 (parametric)** 学习算法，因为它通过固定数量的参数 ( $\theta_i$ ) 来拟合数据。一旦这些参数  $\theta_i$  被确定并存储下来，就不再需要保留训练数据来进行后续的预测。相比之下，为了使用局部加权线性回归进行预测，必须保留整个训练集。术语“非参数”（大致）反映了这样一个事实：表示假设函数  $h$  所需存储的数据量与训练集的大小呈线性关系。

---

<sup>4</sup>如果  $x$  是向量，则推广为  $w^{(i)} = \exp(-(x^{(i)} - x)^T(x^{(i)} - x)/(2\tau^2))$ ，或者  $w^{(i)} = \exp(-(x^{(i)} - x)^T\Sigma^{-1}(x^{(i)} - x)/(2\tau^2))$ ，其中  $\tau$  和  $\Sigma$  需要选择合适的值。

# 第 2 章 分类与逻辑回归

现在我们转向分类问题。这与回归问题类似，主要区别在于需要预测的目标变量  $y$  只能取有限的离散值。目前，我们将重点讨论二元分类 (binary classification) 问题，其中  $y$  的取值仅限于 0 和 1。（这里讨论的大部分内容也适用于多类别分类情况。）例如，在构建垃圾邮件分类器时， $x^{(i)}$  可以代表一封电子邮件的某些特征，而  $y$  则表示该邮件是否为垃圾邮件（垃圾邮件为 1，非垃圾邮件为 0）。通常，0 被称为**负类 (negative class)**，1 被称为**正类 (positive class)**，有时也用符号“-”和“+”表示。对于给定的输入特征  $x^{(i)}$ ，其对应的  $y^{(i)}$  被称为该训练样本的**标签 (label)**。

## 2.1 逻辑回归

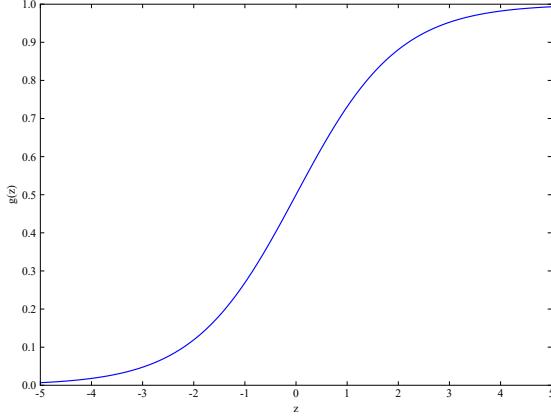
在处理分类问题时，我们可以暂时忽略目标变量  $y$  是离散值这一特性，并沿用之前的线性回归算法来尝试预测给定输入  $x$  的  $y$  值。然而，很容易构造出这种方法表现极差的例子。直觉上，由于  $y$  只能取  $\{0, 1\}$  中的值，模型的输出  $h_\theta(x)$  取大于 1 或小于 0 的值是没有意义的。为了解决这一问题，我们需要改变假设函数  $h_\theta(x)$  的形式。具体而言，我们将选择：

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

其中

$$g(z) = \frac{1}{1 + e^{-z}}.$$

称为**逻辑函数 (logistic function)** 或**S 形函数 (sigmoid function)**。下面是  $g(z)$  的图像：



注意到  $g(z)$  在  $z \rightarrow \infty$  时趋于 1，在  $z \rightarrow -\infty$  时趋于 0。此外， $g(z)$ ，因此  $h(x)$ ，始终介于 0 和 1 之间。和之前一样，这里约定  $x_0 = 1$ ，从而有  $\theta^T x = \theta_0 + \sum_{j=1}^d \theta_j x_j$ 。

现在先将  $g$  的形式视为一个已知条件。其他从 0 平滑增加到 1 的函数也可以使用，但出于一些原因（稍后讨论广义线性模型 (GLMs) 和生成学习算法时会看到），选择 sigmoid 函数是相当自然的。在进一步展开之前，这里给出 sigmoid 函数导数的一个有用性质，记为  $g'$ ：

$$\begin{aligned} g'(z) &= \frac{d}{dz} \frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2} (e^{-z}) \\ &= \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

那么，给定这样的逻辑回归模型，我们如何为其拟合  $\theta$  呢？参照我们之前所见，最小二乘回归在一定假设下可以作为最大似然估计的一种形式。因此，我们也将为分类模型设定一组概率假设，然后通过最大似然估计的方式来拟合参数。

假设

$$\begin{aligned} P(y=1 | x; \theta) &= h_\theta(x) \\ P(y=0 | x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

注意到上述两个概率表达式可以合并为一个更紧凑的形式

$$p(y | x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

假设  $n$  个训练样本是独立生成的，那么参数的似然可以写成

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^n p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^n (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

和之前一样，最大化对数似然会更容易推导：

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^n y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \quad (2.1)$$

那么，如何最大化这个似然函数呢呢？类似于线性回归的推导过程，我们可以采用梯度上升法。以向量形式表示，参数的更新规则为  $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$ 。（注意更新公式中的正号，因为现在是在最大化函数，而不是最小化函数。）接下来，我们将从一个训练样本  $(x, y)$  出发，推导随机梯度上升规则的导数：

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x))x_j \\ &= (y - g(\theta^T x))x_j \end{aligned} \quad (2.2)$$

上面的推导利用了  $g'(z) = g(z)(1 - g(z))$  这一点。这给出了随机梯度上升规则：

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

如果将推导出的逻辑回归更新规则与最小均方更新规则进行比较，会发现它们在形式上是相同的；但这并不是同一个算法，因为这里的  $h_\theta(x^{(i)})$  是  $\theta^T x^{(i)}$  的非线性函数。尽管如此，对于一个截然不同的算法和学习问题，却得到了相同的更新规则，这确实有些令人惊讶。这仅仅是巧合吗？抑或是背后存在更深层的原因？我们将在讨论广义线性模型（GLM）时解答这个问题。

**备注 2.1.1.** 同一个损失函数的另一种表示方式也很有用，特别是在第 7.1 节研究非线性模型时。

设逻辑损失函数  $\ell_{\text{logistic}} : \mathbb{R} \times \{0, 1\} \rightarrow \mathbb{R}_{\geq 0}$  定义为

$$\ell_{\text{logistic}}(t, y) \triangleq y \log(1 + \exp(-t)) + (1 - y) \log(1 + \exp(t)). \quad (2.3)$$

通过代入  $h_\theta(x) = 1/(1 + e^{-\theta^T x})$ , 可以验证负对数似然 (方程 (2.1) 中  $\ell(\theta)$  的负值) 可以改写为

$$-\ell(\theta) = \ell_{\text{logistic}}(\theta^T x, y). \quad (2.4)$$

通常  $\theta^T x$  或  $t$  称为 *logit*。稍作运算可得

$$\frac{\partial \ell_{\text{logistic}}(t, y)}{\partial t} = y \frac{-\exp(-t)}{1 + \exp(-t)} + (1 - y) \frac{1}{1 + \exp(-t)} \quad (2.5)$$

$$= \frac{1}{1 + \exp(-t)} - y. \quad (2.6)$$

然后, 使用链式法则, 得到

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = -\frac{\partial \ell_{\text{logistic}}(t, y)}{\partial t} \cdot \frac{\partial t}{\partial \theta_j} \quad (2.7)$$

$$= (y - 1/(1 + \exp(-t))) \cdot x_j = (y - h_\theta(x))x_j, \quad (2.8)$$

这与方程 (2.2) 的推导是一致的。在第 7.1 节中, 会看到这种观点可以扩展到非线性模型。

## 2.2 离题：感知器学习算法

现在, 我们将简要讨论一个具有历史意义的算法, 该算法在后续讨论学习理论时也将再次被提及。考虑对逻辑回归方法进行修改, “强制” 其输出值为 0 或 1。为此, 一个自然而然的想法是将函数  $g$  的定义改为阈值函数:

$$g(z) = \begin{cases} 1 & \text{若 } z \geq 0 \\ 0 & \text{若 } z < 0 \end{cases}$$

如果像之前一样令  $h_\theta(x) = g(\theta^T x)$ , 但使用上述修改后的  $g$  定义, 并且使用更新规则

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}.$$

那么就得到了**感知机学习算法 (perceptron learning algorithm)**。

在 20 世纪 60 年代, 有人认为“感知机”是脑中单个神经元如何工作的一个粗略模型。考虑到该算法的简单性, 在讨论学习理论时, 它也将为分析提供一个起点。然而, 请注意, 尽管感知机看起来与其他讨论过的算法相似, 但它实际上与逻辑回归和最小二乘线性回归是完全不同类型的算法; 特别是, 很难从概率角度解释感知机的预测结果, 或者将感知机推导为最大似然估计算法。

## 2.3 多类别分类

考虑一个分类问题，其中响应变量  $y$  可以取  $k$  个值中的任意一个，即  $y \in \{1, 2, \dots, k\}$ 。例如，除了将电子邮件分为垃圾邮件或非垃圾邮件这两类（这是一个二元分类问题），也可能希望将其分为三类，例如垃圾邮件、个人邮件和工作相关邮件。标签/响应变量仍然是离散的，但现在可以取超过两个值。因此，将它建模为服从多项分布。

在这种情况下， $p(y | x; \theta)$  是关于  $k$  个可能的离散结果的分布，因此是多项分布。回想一下，多项分布涉及  $k$  个数  $\phi_1, \dots, \phi_k$ ，它们指定了每个结果的概率。注意，这些数必须满足  $\sum_{i=1}^k \phi_i = 1$ 。将设计一个参数化模型，该模型输出满足此约束的  $\phi_1, \dots, \phi_k$ ，给定输入  $x$ 。

引入  $k$  组参数  $\theta_1, \dots, \theta_k$ ，每组参数都是  $\mathbb{R}^d$  中的一个向量。直观地，希望使用  $\theta_1^T x, \dots, \theta_k^T x$  来表示  $\phi_1, \dots, \phi_k$ ，即概率  $P(y = 1 | x; \theta), \dots, P(y = k | x; \theta)$ 。然而，这种直接方法存在两个问题。首先， $\theta_j^T x$  不一定在  $[0, 1]$  范围内。其次， $\theta_j^T x$  的总和不一定为 1。因此，将使用 softmax 函数将  $(\theta_1^T x, \dots, \theta_k^T x)$  转换为一个非负且总和为 1 的概率向量。

定义函数  $\text{softmax} : \mathbb{R}^k \rightarrow \mathbb{R}^k$  如下：

$$\text{softmax}(t_1, \dots, t_k) = \begin{bmatrix} \frac{\exp(t_1)}{\sum_{j=1}^k \exp(t_j)} \\ \vdots \\ \frac{\exp(t_k)}{\sum_{j=1}^k \exp(t_j)} \end{bmatrix}. \quad (2.9)$$

softmax 函数的输入，即这里的向量  $t$ ，通常被称为 *logits*。注意，根据定义，softmax 函数的输出始终是一个概率向量，其分量非负且总和为 1。

令  $(t_1, \dots, t_k) = (\theta_1^T x, \dots, \theta_k^T x)$ 。将 softmax 函数应用于  $(t_1, \dots, t_k)$ ，并将输出用作概率  $P(y = 1 | x; \theta), \dots, P(y = k | x; \theta)$ 。得到以下概率模型：

$$\begin{bmatrix} P(y = 1 | x; \theta) \\ \vdots \\ P(y = k | x; \theta) \end{bmatrix} = \text{softmax}(t_1, \dots, t_k) = \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \\ \vdots \\ \frac{\exp(\theta_k^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \end{bmatrix}. \quad (2.10)$$

为了符号上的方便，令  $\phi_i = \frac{\exp(t_i)}{\sum_{j=1}^k \exp(t_j)}$ 。更简洁地，上面的方程可以写成：

$$P(y = i | x; \theta) = \phi_i = \frac{\exp(t_i)}{\sum_{j=1}^k \exp(t_j)} = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)}. \quad (2.11)$$

接下来，计算单个样本  $(x, y)$  的负对数似然。

$$-\log p(y | x, \theta) = -\log \left( \frac{\exp(t_y)}{\sum_{j=1}^k \exp(t_j)} \right) = -\log \left( \frac{\exp(\theta_y^T x)}{\sum_{j=1}^k \exp(\theta_j^T x)} \right). \quad (2.12)$$

因此，损失函数，即训练数据的负对数似然，由下式给出：

$$\ell(\theta) = \sum_{i=1}^n -\log \left( \frac{\exp(\theta_{y^{(i)}}^T x^{(i)})}{\sum_{j=1}^k \exp(\theta_j^T x^{(i)})} \right). \quad (2.13)$$

定义交叉熵损失  $\ell_{\text{ce}} : \mathbb{R}^k \times \{1, \dots, k\} \rightarrow \mathbb{R}_{\geq 0}$  是很方便的，它将上述复杂的方程模块化为：<sup>1</sup>

$$\ell_{\text{ce}}((t_1, \dots, t_k), y) = -\log \left( \frac{\exp(t_y)}{\sum_{j=1}^k \exp(t_j)} \right). \quad (2.14)$$

使用此记号，方程 (2.13) 可以简写为：

$$\ell(\theta) = \sum_{i=1}^n \ell_{\text{ce}}((\theta_1^T x^{(i)}, \dots, \theta_k^T x^{(i)}), y^{(i)}). \quad (2.15)$$

此外，交叉熵损失的梯度也很便于推导。令  $t = (t_1, \dots, t_k)$ ，根据  $\phi_i = \frac{\exp(t_i)}{\sum_{j=1}^k \exp(t_j)}$ 。可以推导出：

$$\frac{\partial \ell_{\text{ce}}(t, y)}{\partial t_i} = \phi_i - 1\{y = i\}, \quad (2.16)$$

其中  $1\{\cdot\}$  是指示函数，即当  $y = i$  时  $1\{y = i\} = 1$ ，当  $y \neq i$  时  $1\{y = i\} = 0$ 。向量化形式如下，这对于第 7 章将很有用：

$$\frac{\partial \ell_{\text{ce}}(t, y)}{\partial t} = \phi - e_y, \quad (2.17)$$

其中  $e_s \in \mathbb{R}^k$  是第  $s$  个标准基向量（其中第  $s$  个分量是 1，其余所有分量都是零）。使用链式法则，有：

$$\frac{\partial \ell_{\text{ce}}((\theta_1^T x, \dots, \theta_k^T x), y)}{\partial \theta_i} = \frac{\partial \ell(t, y)}{\partial t_i} \frac{\partial t_i}{\partial \theta_i} = (\phi_i - 1\{y = i\}) \cdot x. \quad (2.18)$$

因此，损失函数关于参数  $\theta_i$  的梯度为：

$$\frac{\partial \ell(\theta)}{\partial \theta_i} = \sum_{j=1}^n (\phi_i^{(j)} - 1\{y^{(j)} = i\}) \cdot x^{(j)}, \quad (2.19)$$

其中  $\phi_i^{(j)} = \frac{\exp(\theta_i^T x^{(j)})}{\sum_{s=1}^k \exp(\theta_s^T x^{(j)})}$  是模型预测样本  $x^{(j)}$  属于类别  $i$  的概率。利用上述梯度，可以使用（随机）梯度下降来最小化损失函数  $\ell(\theta)$ 。

---

<sup>1</sup>这里命名存在一些歧义。有些人将交叉熵损失定义为将概率向量（在本讲义中用  $\phi$  表示）和标签  $y$  映射到实数的函数，并将本讲义中的交叉熵损失称为 softmax-交叉熵损失。本讲义选择当前的命名约定是因为它与大多数现代深度学习库（如 PyTorch 和 Jax）的命名一致。

## 2.4 最大化 $\ell(\theta)$ 的另一种算法

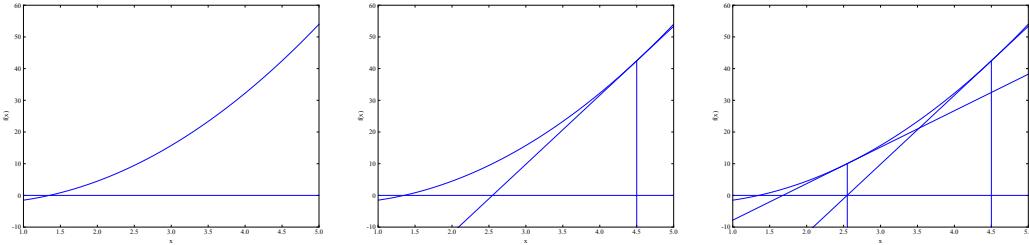
回到以 sigmoid 为  $g(z)$  函数的逻辑回归，现在讨论一种不同的最大化  $\ell(\theta)$  的算法。

首先，考虑牛顿法用于寻找函数零点。具体来说，假设有一个函数  $f : \mathbb{R} \rightarrow \mathbb{R}$ ，并且希望找到一个  $\theta$  值使得  $f(\theta) = 0$ 。这里  $\theta \in \mathbb{R}$  是一个实数。牛顿法执行以下更新：

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}.$$

这种方法有一种自然的解释：将函数  $f$  通过其在当前猜测值  $\theta$  处的切线进行近似，然后求解该切线等于零的点，并将该点作为  $\theta$  的下一个猜测值。

以下是牛顿法实际应用的图示：



在最左边的图中，可以看到函数  $f$  与直线  $y = 0$ 。尝试找到一个  $\theta$  使得  $f(\theta) = 0$ ；实现这一点的  $\theta$  值大约是 1.3。假设初始化的算法的  $\theta$  值为 4.5。然后牛顿法拟合一条在  $\theta = 4.5$  处与  $f$  相切的直线，并求解该直线等于 0 的点（中间图）。这给出了  $\theta$  的下一个猜测值，大约是 2.6。最右边的图显示了再进行一次迭代的结果，更新后的  $\theta$  大约是 1.6。再经过几次迭代后，将迅速接近  $\theta = 1.3$ 。<sup>\*</sup>

牛顿法提供了一种求解  $f(\theta) = 0$  的方法。如果希望最大化某个函数  $\ell$  呢？ $\ell$  的最大值对应于其一阶导数  $\ell'(\theta)$  为零的点。因此，令  $f(\theta) = \ell'(\theta)$ ，可以使用相同的算法来最大化  $\ell$ ，并得到更新规则：

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}.$$

(思考题：如果希望使用牛顿法来最小化而不是最大化一个函数，这会如何改变？)

最后，在逻辑回归中， $\theta$  是向量，因此需要将牛顿法推广到此情况。牛顿法在此多维设置中的推广（也称为牛顿-拉普森法）由下式给出

$$\theta := \theta - H^{-1} \nabla_{\theta} \ell(\theta).$$

---

<sup>\*</sup>译者注：由于不知道原书所用函数的解析形式，这里译者画的图与原图稍有偏差， $\theta$  的猜测值也稍有变化。

这里， $\nabla_{\theta}\ell(\theta)$  是  $\ell(\theta)$  对  $\theta_i$  的偏导数向量，而  $H$  是一个  $d \times d$  矩阵（实际上，如果包含截距项，则是  $(d + 1) \times (d + 1)$  矩阵），称为 **Hessian** 矩阵，其元素由下式给出

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}.$$

牛顿法通常比（批量）梯度下降收敛更快，并且需要更少的迭代次数即可非常接近最小值。然而，牛顿法迭代一次比梯度下降迭代一次更昂贵，因为它需要找到一个  $d \times d$  的 Hessian 矩阵并求逆；但只要  $d$  不太大，通常总体上会快得多。当牛顿法应用于最大化逻辑回归对数似然函数  $\ell(\theta)$  时，所得方法也称为 **Fisher scoring**。

# 第 3 章 广义线性模型

到目前为止，我们已经讨论了一个回归示例和一个分类示例。在回归示例中，有  $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$ ，在分类示例中，有  $y|x; \theta \sim \text{Bernoulli}(\phi)$ ，其中  $\mu$  和  $\phi$  是  $x$  和  $\theta$  的函数。在本章中，将展示这两种方法都是更广泛的模型族——称为广义线性模型 (GLMs) ——的特例。<sup>1</sup> 我们还将展示广义线性模型族中的其他模型如何推导并应用于其他分类和回归问题。

## 3.1 指数族

为了逐步了解广义线性模型，首先定义指数族分布。如果一类分布可以写成以下形式，则称其为指数族：

$$p(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta)) \quad (3.1)$$

这里， $\eta$  称为自然参数 (natural parameter) (也称为典范参数 (canonical parameter))； $T(y)$  是充分统计量 (sufficient statistic) (对于所考虑的分布，通常有  $T(y) = y$ )；而  $a(\eta)$  是对数配分函数 (log partition function)。量  $e^{-a(\eta)}$  实际上起着归一化常数的作用，确保分布  $p(y; \eta)$  在  $y$  上的和或积分等于 1。

固定的  $T$ ,  $a$  和  $b$  定义了一个由  $\eta$  参数化的族 (family) (或分布集)；随着  $\eta$  的变化，将得到该族中的不同分布。

现在展示伯努利分布和高斯分布是指数族分布的示例。具有均值  $\phi$  的伯努利分布，记为  $\text{Bernoulli}(\phi)$ ，指定了在  $y \in \{0, 1\}$  上的分布，使得  $p(y = 1; \phi) = \phi$ ;  $p(y = 0; \phi) = 1 - \phi$ 。随着  $\phi$  的变化，可以得到具有不同均值的伯努利分布。现在我们推导改变  $\phi$  得到的这些伯努利分布属于指数族；也就是说，存在一种  $T, a, b$  的选择，使得公式 (3.1) 恰好成为伯努利分布。

---

<sup>1</sup>本章材料受到 Michael I. Jordan 的 *Learning in graphical models* (未出版的书稿) 以及 McCullagh 和 Nelder 的 *Generalized Linear Models* (2nd ed.) 的启发。

将伯努利分布写为：

$$\begin{aligned}
 p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\
 &= \exp(y \log \phi + (1 - y) \log(1 - \phi)) \\
 &= \exp\left(\left(\log\left(\frac{\phi}{1-\phi}\right)\right)y + \log(1 - \phi)\right).
 \end{aligned}$$

因此，自然参数由  $\eta = \log(\phi/(1 - \phi))$  给出。有趣的是，如果通过求解  $\phi$  关于  $\eta$  的表达式来反转这个定义，得到  $\phi = 1/(1 + e^{-\eta})$ 。这是熟悉的 sigmoid 函数！在将逻辑回归推导为广义线性模型时，这将再次出现。为了完成伯努利分布作为指数族分布的形式化，还需要以下各项：

$$\begin{aligned}
 T(y) &= y \\
 a(\eta) &= -\log(1 - \phi) \\
 &= \log(1 + e^\eta) \\
 b(y) &= 1
 \end{aligned}$$

这表明伯努利分布可以通过选择适当的  $T, a, b$  从而写成公式 (3.1) 的形式。

接下来考虑高斯分布。回想一下，在线性回归推导中， $\sigma^2$  的值对最终选择的  $\theta$  和  $h_\theta(x)$  没有影响。因此，可以在不改变任何内容的情况下选择任意的  $\sigma^2$  值。为了简化下面的推导，令  $\sigma^2 = 1$ 。<sup>2</sup> 有：

$$\begin{aligned}
 p(y; \mu) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - \mu)^2\right) \\
 &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2 + \mu y - \frac{1}{2}\mu^2\right) \\
 &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \cdot \exp\left(\mu y - \frac{1}{2}\mu^2\right).
 \end{aligned}$$

因此，高斯分布属于指数族，其中

$$\begin{aligned}
 \eta &= \mu \\
 T(y) &= y \\
 a(\eta) &= \mu^2/2 \\
 &= \eta^2/2 \\
 b(y) &= (1/\sqrt{2\pi}) \exp(-y^2/2).
 \end{aligned}$$

---

<sup>2</sup>如果将  $\sigma^2$  作为一个变量，高斯分布也可以显示为指数族，其中  $\eta \in \mathbb{R}^2$  现在是一个二维向量，它取决于  $\mu$  和  $\sigma$ 。然而，出于广义线性模型的目的， $\sigma^2$  参数也可以通过考虑指数族的一个更一般的定义来处理： $p(y; \eta, \tau) = b(a, \tau) \exp((\eta^T T(y) - a(\eta))/c(\tau))$ 。这里， $\tau$  称为**散布参数 (dispersion parameter)**，对于高斯分布， $c(\tau) = \sigma^2$ ；但考虑到上面的简化，这里不需要更一般的定义。

还有许多其他分布也属于指数族：多项分布（稍后将看到）、泊松分布（用于建模计数数据；另请参阅问题集）、伽马分布和指数分布（用于建模连续非负随机变量，例如时间间隔）、Beta 分布和 Dirichlet 分布（用于概率分布）等等。在下一节中，将描述构建模型的一般“配方”，其中  $y$ （给定  $x$  和  $\theta$ ）来自于这些分布中的任何一个。

## 3.2 构造广义线性模型

假设希望构建一个模型来估计在给定小时内到达商店的顾客数量（或网站的页面浏览量  $y$ ），基于某些特征  $x$ ，例如商店促销、近期广告、天气、星期几等等。已知泊松分布通常能很好地模拟访客数量。知道了这一点，如何为问题构建模型？幸运的是，泊松分布是指数族分布，因此可以应用广义线性模型。在本节中，将描述一种构建用于解决此类问题的广义线性模型的方法。

更一般地，考虑一个分类或回归问题，希望预测某个随机变量  $y$  作为  $x$  的函数的值。为了推导针对此问题的广义线性模型，将对给定  $x$  的  $y$  的条件分布和模型做出以下三个假设：

1.  $y|x; \theta \sim \text{ExponentialFamily}(\eta)$ 。也就是说，给定  $x$  和  $\theta$ ， $y$  的分布遵循参数为  $\eta$  的某个指数族分布；
2. 给定  $x$ ，目标是预测  $T(y)$  的期望值。在大多数示例中， $T(y) = y$ ，因此这意味着希望学习到的假设  $h$  的输出预测  $h(x)$  满足  $h(x) = E[y|x]$ 。（注意，这个假设在逻辑回归和线性回归中对于  $h_\theta(x)$  的选择是满足的。例如，在逻辑回归中， $h_\theta(x) = p(y=1|x; \theta) = 0 \cdot p(y=0|x; \theta) + 1 \cdot p(y=1|x; \theta) = E[y|x; \theta]$ 。）
3. 自然参数  $\eta$  和输入  $x$  线性相关： $\eta = \theta^T x$ 。（或者，如果  $\eta$  是向量值，则  $\eta_i = \theta_i^T x$ 。）

第三个假设可能看起来最不合理，最好将其视为设计广义线性模型的“设计选择”，而不是一个固有的假设。这三个假设/设计选择将允许推导出非常优雅的一类学习算法，即广义线性模型，它们具有许多理想的特性，例如易于学习。此外，由此产生的模型对于建模不同类型的  $y$  分布非常有效；例如，很快将展示逻辑回归和普通最小二乘都可以作为广义线性模型推导出来。

### 3.2.1 普通最小二乘

为了展示普通最小二乘是广义线性模型族的一个特例，考虑目标变量  $y$ （在广义线性模型术语中也称为**响应变量 (response variable)**）是连续的情况，并将给定  $x$  的  $y$  的条件分布建模为高斯分布  $N(\mu, \sigma^2)$ 。（这里， $\mu$  可能取决于  $x$ 。）因此，令上面的  $\text{ExponentialFamily}(\eta)$  分布为高斯分布。如前所述，在将高斯分布公式化为指数族分布时，有  $\mu = \eta$ 。因此，有

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= \mu \\ &= \eta \\ &= \theta^T x. \end{aligned}$$

第一个等号来自于上面的假设 2；第二个等号来自于  $y|x; \theta \sim N(\mu, \sigma^2)$ ，因此其期望值由  $\mu$  给出；第三个等号来自于假设 1（以及之前推导中表明在将高斯分布公式化为指数族分布时  $\mu = \eta$ ）；最后一个等号来自于假设 3。

### 3.2.2 逻辑回归

现在考虑逻辑回归。这里感兴趣的是二分类问题，因此  $y \in \{0, 1\}$ 。鉴于  $y$  是二值变量，选择伯努利分布族来建模给定  $x$  的  $y$  的条件分布是很自然的。在将伯努利分布公式化为指数族分布时，有  $\phi = 1/(1 + e^{-\eta})$ 。此外，注意如果  $y|x; \theta \sim \text{Bernoulli}(\phi)$ ，则  $E[y|x; \theta] = \phi$ 。因此，按照与普通最小二乘类似的推导，得到：

$$\begin{aligned} h_\theta(x) &= E[y|x; \theta] \\ &= \phi \\ &= 1/(1 + e^{-\eta}) \\ &= 1/(1 + e^{-\theta^T x}) \end{aligned}$$

因此，这给出了形式为  $h_\theta(x) = 1/(1 + e^{-\theta^T x})$  的假设函数。如果之前曾想知道如何得到逻辑函数  $1/(1 + e^{-z})$  的形式，这就是一个答案：一旦假设给定  $x$  的  $y$  服从伯努利分布，它就作为广义线性模型和指数族分布定义的必然结果出现了。

这里引入一些额外的术语，将自然参数映射到分布均值的函数  $g$  ( $g(\eta) = E[T(y); \eta]$ )，称为**典范响应函数 (canonical response function)**。其逆函数  $g^{-1}$  称为**典范连接函数 (canonical link function)**。因此，高斯族分布的典范响应函数就是恒等函数；伯努利分布的典范响应函数是逻辑函数。<sup>3</sup>

---

<sup>3</sup>许多文献使用  $g$  表示连接函数， $g^{-1}$  表示响应函数；但这里使用的符号继承自早期的机器学

---

习文献，将与课程其余部分使用的符号更一致。

# 第 4 章 生成式学习算法

到目前为止，我们讨论的主要是一类算法，这些算法建模给定  $x$  的情况下  $y$  的条件分布  $p(y|x; \theta)$ 。例如，逻辑回归将  $p(y|x; \theta)$  建模为  $h_\theta(x) = g(\theta^T x)$ ，其中  $g$  是 sigmoid 函数。在本章中，将讨论一种不同类型的学习算法。

考虑一个分类问题，其中希望根据动物的一些特征来区分大象 ( $y = 1$ ) 和狗 ( $y = 0$ )。给定一个训练集，像逻辑回归或感知机算法（本质上）试图找到一条直线——也就是一个决策边界——来分隔大象和狗。然后，为了将新动物分类为大象或狗，检查其落在决策边界的哪一侧，并据此做出预测。

这里介绍一种不同的方法。首先，观察大象，可以建立一个关于大象外观的模型。然后，观察狗，可以建立一个关于狗外观的独立模型。最后，为了对新动物进行分类，可以将新动物与大象模型进行匹配，并将其与狗模型进行匹配，以查看新动物是否更像在训练集中看到的大象或狗。

试图直接学习  $p(y|x)$  的算法（如逻辑回归），或试图直接学习从输入空间  $\mathcal{X}$  到标签  $\{0, 1\}$  的映射的算法（如感知机算法）称为**判别式 (discriminative)** 学习算法。这里，将讨论那些试图建模  $p(x|y)$ （和  $p(y)$ ）的算法。这些算法称为**生成式 (generative)** 学习算法。例如，如果  $y$  表示一个样本是狗 (0) 还是大象 (1)，则  $p(x|y = 0)$  建模狗的特征分布， $p(x|y = 1)$  建模大象的特征分布。

在建模  $p(y)$ （称为**类先验 (class priors)**）和  $p(x|y)$  之后，算法可以利用贝叶斯定理推导出给定  $x$  时  $y$  的后验分布：

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}.$$

这里，分母由  $p(x) = p(x|y = 1)p(y = 1) + p(x|y = 0)p(y = 0)$  给出（根据概率的标准性质，可以验证这一点），因此也可以用学到的  $p(x|y)$  和  $p(y)$  来表示。实际上，如果计算  $p(y|x)$  是为了进行预测，那么并不需要计算分母，因为

$$\begin{aligned} \arg \max_y p(y|x) &= \arg \max_y \frac{p(x|y)p(y)}{p(x)} \\ &= \arg \max_y p(x|y)p(y). \end{aligned}$$

## 4.1 高斯判别分析

将要介绍的第一个生成式学习算法是高斯判别分析 (GDA)。在这个模型中，假设  $p(x|y)$  服从多元正态分布。在介绍 GDA 模型本身之前，先简要讨论一下多元正态分布的性质。

### 4.1.1 多元正态分布

$d$  维的多元正态分布，也称为多元高斯分布，由**均值向量 (mean vector)**  $\mu \in \mathbb{R}^d$  和**协方差矩阵 (covariance matrix)**  $\Sigma \in \mathbb{R}^{d \times d}$  参数化，其中  $\Sigma \geq 0$  是对称正半定矩阵。其密度函数写为  $\mathcal{N}(\mu, \Sigma)$ ，形式如下：

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right).$$

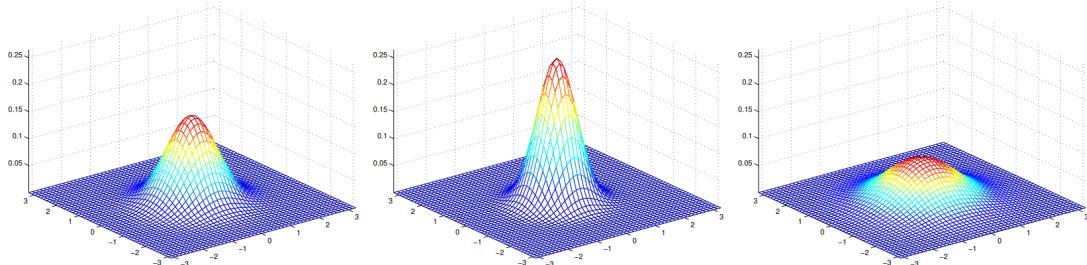
在上面的方程中， $|\Sigma|$  表示矩阵  $\Sigma$  的行列式。对于服从  $\mathcal{N}(\mu, \Sigma)$  分布的随机变量  $X$ ，其均值（毫不意外地）由  $\mu$  给出：

$$\mathbb{E}[X] = \int_x xp(x; \mu, \Sigma) dx = \mu$$

向量值随机变量  $Z$  的**协方差**定义为  $\text{Cov}(Z) = \mathbb{E}[(Z - \mathbb{E}[Z])(Z - \mathbb{E}[Z])^T]$ 。这推广了实值随机变量的方差概念。协方差也可以定义为  $\text{Cov}(Z) = \mathbb{E}[ZZ^T] - (\mathbb{E}[Z])(\mathbb{E}[Z])^T$ 。（可以自行证明这两个定义是等价的。）如果  $X \sim \mathcal{N}(\mu, \Sigma)$ ，则

$$\text{Cov}(X) = \Sigma.$$

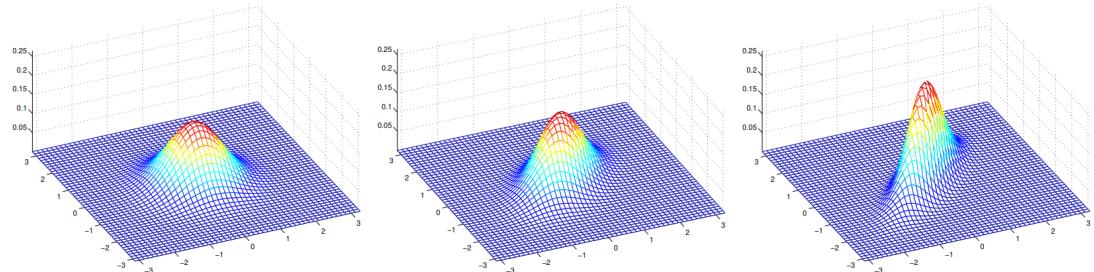
以下是一些高斯分布密度函数的示例：



最左边的图显示的是均值为零（即  $2 \times 1$  零向量）、协方差矩阵为  $\Sigma = I$ （即  $2 \times 2$  单位矩阵）的高斯分布。均值为零、协方差为单位矩阵的高斯分布也称为**标准正态分布**。中间的图显示的是均值为零、 $\Sigma = 0.6I$  的高斯分布的密度；最右

边的图显示的是  $\Sigma = 2I$  的高斯分布的密度。可以看到，随着  $\Sigma$  变大，高斯分布变得更加“分散”，而随着  $\Sigma$  变小，分布变得更加“紧凑”。

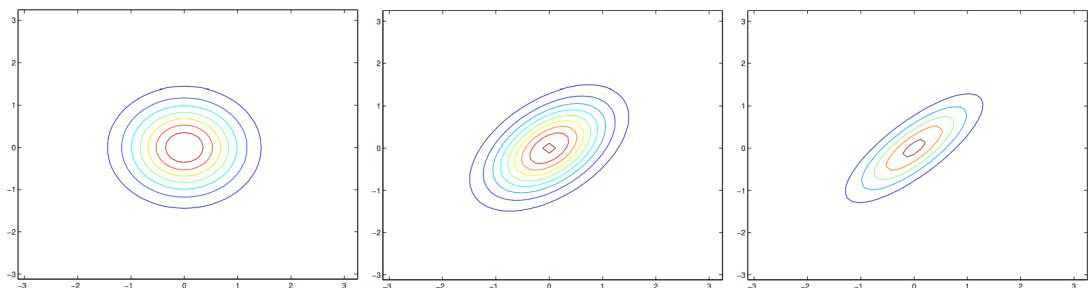
接下来看一些更多的例子。



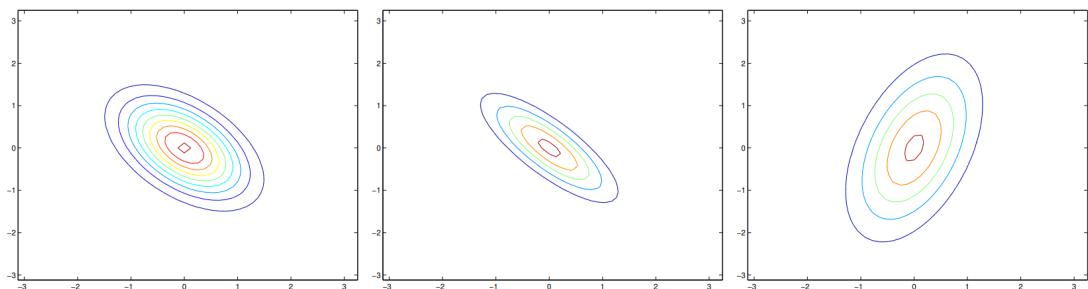
上面的图显示了均值为 0、协方差矩阵分别为

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

最左边的图显示的是熟悉的标准正态分布，并且可以看到，随着  $\Sigma$  中非对角线元素的增加，密度函数变得更加“压缩”到  $45^\circ$  线（由  $x_1 = x_2$  给出）。当观察这三个密度函数的等高线时，可以更清楚地看到这一点：



下面是另外一组由不同的  $\Sigma$  生成的例子：

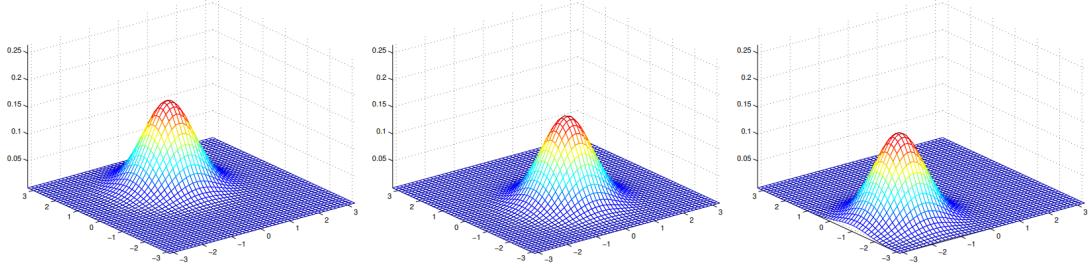


上面的图分别使用了

$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}.$$

从最左边和中间的图可以看到，通过减小协方差矩阵的非对角线元素，密度函数再次变得“压缩”，但方向相反。最后值得一提的是，当改变参数时，等高线通常会形成椭圆（最右边的图显示了一个例子）。

作为最后一组例子，通过固定  $\Sigma = I$ ，并改变  $\mu$ ，也可以移动密度函数的均值。



上面的图是使用  $\Sigma = I$  生成的，并且  $\mu$  分别为

$$\mu = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \mu = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix}; \mu = \begin{bmatrix} -1 \\ -1.5 \end{bmatrix}.$$

#### 4.1.2 高斯判别分析模型

当遇到输入特征  $x$  是连续随机变量的分类问题时，可以使用高斯判别分析模型，该模型利用多元正态分布对  $p(x|y)$  进行建模。模型如下：

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ x|y=0 &\sim \mathcal{N}(\mu_0, \Sigma) \\ x|y=1 &\sim \mathcal{N}(\mu_1, \Sigma) \end{aligned}$$

写出分布形式为：

$$\begin{aligned} p(y) &= \phi^y(1-\phi)^{1-y} \\ p(x|y=0) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1} (x-\mu_0)\right) \\ p(x|y=1) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1} (x-\mu_1)\right) \end{aligned}$$

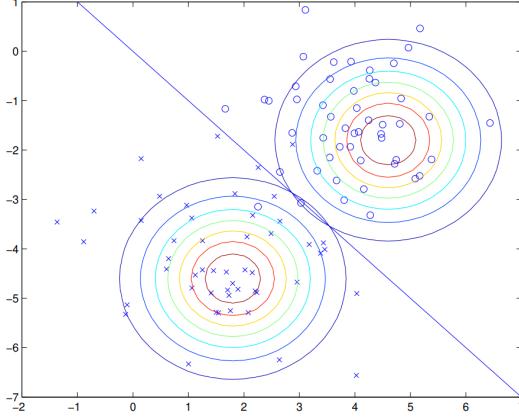
模型中的参数为  $\phi, \Sigma, \mu_0$  和  $\mu_1$ 。（注意，尽管有两个不同的均值向量  $\mu_0$  和  $\mu_1$ ，但该模型通常只使用一个协方差矩阵  $\Sigma$ 。）数据的对数似然函数如下：

$$\begin{aligned} \ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^n p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^n p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi). \end{aligned}$$

通过最大化  $\ell$  对参数进行估计，可以得到参数的最大似然估计（参见习题集 1）：

$$\begin{aligned}\phi &= \frac{1}{n} \sum_{i=1}^n 1\{y^{(i)} = 1\} \\ \mu_0 &= \frac{\sum_{i=1}^n 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)} = 0\}} \\ \mu_1 &= \frac{\sum_{i=1}^n 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^n 1\{y^{(i)} = 1\}} \\ \Sigma &= \frac{1}{n} \sum_{i=1}^n (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T.\end{aligned}$$

从直观上看，算法的操作可以表示如下：



图中展示了训练集以及拟合到两个类别数据的两个高斯分布的等高线。注意到由于共享协方差矩阵  $\Sigma$ ，这两个高斯分布的等高线形状和方向相同，但它们的均值  $\mu_0$  和  $\mu_1$  不同。图中还展示了  $p(y = 1|x) = 0.5$  的决策边界直线。在边界的一侧，预测  $y = 1$  是最可能的结果，而在另一侧，则更多地预测  $y = 0$ 。

#### 4.1.3 讨论：GDA 与逻辑回归

GDA 模型与逻辑回归有着有趣的关系。如果将  $p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma)$  视为  $x$  的函数，会发现它可以表示为以下形式：

$$p(y = 1|x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-\theta^T x)},$$

其中  $\theta$  是  $\phi, \mu_0, \mu_1, \Sigma$  的适当函数。<sup>1</sup> 这正是逻辑回归（一种判别式算法）用来建模  $p(y = 1|x)$  的形式。

---

<sup>1</sup>这里采用了重新定义右侧  $x^{(i)}$  为  $(d+1)$  维向量的约定，即添加额外的坐标  $x_0^{(i)} = 1$ ；参见习题集 1。

何时应该优先选择其中一个模型，何时又该选择另一个？GDA 和逻辑回归在同一数据集上训练时，通常会给出不同的决策边界，如何判断哪个会更好？

前文论述了，如果  $p(x|y)$  是多元高斯分布（且共享  $\Sigma$ ），那么  $p(y|x)$  必然是逻辑函数。反之则不成立，即  $p(y|x)$  是逻辑函数并不意味着  $p(x|y)$  是多元高斯分布。这表明 GDA 对数据做出了比逻辑回归更强的建模假设。事实证明，当这些建模假设正确时，GDA 会更好地拟合数据，并且是一个更好的模型。具体来说，当  $p(x|y)$  确实是高斯分布（且共享  $\Sigma$ ）时，GDA 具有渐近效率 (asymptotically efficient)。非正式地讲，这意味着在训练集非常大 ( $n$  很大) 的极限情况下，没有哪个算法能比 GDA 更好（例如，在估计  $p(y|x)$  的准确性方面）。特别地，在这种情况下，可以证明 GDA 是比逻辑回归更好的算法；更一般地，即使训练集较小，通常也期望 GDA 表现更好。

相比之下，通过做出显著较弱的假设，逻辑回归也更鲁棒 (Robust)，对不正确的建模假设更不敏感。有许多不同的假设集合可以导致  $p(y|x)$  呈现逻辑函数的形式。例如，如果  $x|y=0 \sim \text{Poisson}(\lambda_0)$  且  $x|y=1 \sim \text{Poisson}(\lambda_1)$ ，那么  $p(y|x)$  将是逻辑函数。逻辑回归在这种泊松分布数据上也能很好地工作。但是，如果对非高斯数据使用 GDA 并拟合高斯分布，那么结果将不太可预测，并且 GDA 可能表现不佳。

总结来说：GDA 做出了更强的建模假设，并且在建模假设正确或至少近似正确时，数据效率更高（即，需要较少的训练数据来“很好地”学习）。逻辑回归做出了较弱的假设，并且对建模假设的错误更鲁棒。具体来说，当数据确实是非高斯分布时，在大数据集的极限情况下，逻辑回归几乎总是比 GDA 表现更好。因此，在实践中，逻辑回归比 GDA 使用更频繁。（关于判别式模型与生成式模型的某些相关考虑也适用于接下来讨论的朴素贝叶斯算法，但朴素贝叶斯算法仍然被认为是一个非常好的，并且当然也是一个非常流行的分类算法。）

## 4.2 朴素贝叶斯（选读）

在 GDA 中，特征向量  $x$  是连续的实值向量。接下来讨论一种不同的学习算法，其中  $x_j$  是离散值。

以构建电子邮件垃圾邮件过滤器为例，考虑使用机器学习。希望根据消息是否为未经请求的商业（垃圾邮件）或非垃圾邮件来对其进行分类。学习完成后，可以使邮件阅读器自动过滤掉垃圾邮件，并将其放入单独的文件夹中。对电子邮件进行分类是更广泛的问题集合（称为文本分类 (text classification)）的一个示例。

假设有一个训练集（一组标记为垃圾邮件或非垃圾邮件的电子邮件）。首先通过指定用于表示电子邮件的特征  $x_j$  来构建垃圾邮件过滤器。

将通过一个特征向量来表示一封电子邮件，该向量的长度等于字典中的单词数量。具体来说，如果一封电子邮件包含字典中的第  $j$  个单词，则设置  $x_j = 1$ ；否则，设置  $x_j = 0$ 。例如，以下向量：

$$x = \begin{bmatrix} 1 & a \\ 0 & aardvark \\ 0 & aardwolf \\ \vdots & \vdots \\ 1 & buy \\ \vdots & \vdots \\ 0 & zygmurgy \end{bmatrix}$$

表示一封包含单词“a”和“buy”但不包含“aardvark”、“aardwolf”或“zygmurgy”的电子邮件。<sup>2</sup> 用于编码特征向量的单词集合称为**词汇表 (vocabulary)**，因此  $x$  的维度等于词汇表的大小。

在选择了特征向量之后，现在希望构建一个生成模型。因此，需要对  $p(x|y)$  进行建模。但是，如果词汇表有 50000 个单词，那么  $x \in \{0, 1\}^{50000}$  ( $x$  是一个由 0 和 1 组成的 50000 维向量)，如果对  $x$  使用  $2^{50000}$  种可能结果的多项分布进行显式建模，最终会得到一个  $(2^{50000} - 1)$  维参数向量。参数数量显然太多了。

因此，为了对  $p(x|y)$  进行建模，将做出一个非常强的假设。假设给定  $y$  时， $x_i$  是条件独立的。这个假设称为**朴素贝叶斯假设 (Naive Bayes (NB) assumption)**，由此产生的算法称为**朴素贝叶斯分类器 (Naive Bayes classifier)**。例如，如果  $y = 1$  表示垃圾邮件，“buy”是第 2087 个单词，“price”是第 39831 个单词；那么假设如果知道  $y = 1$ （这封特定的电子邮件是垃圾邮件），那么关于  $x_{2087}$  的知识（消息中是否出现“buy”）对关于  $x_{39831}$  的值（是否出现“price”）的信念没有影响。更正式地讲，这可以写成  $p(x_{2087}|y) = p(x_{2087}|y, x_{39831})$ 。（注意，这与  $x_{2087}$  和  $x_{39831}$  是独立的说法不同，独立的说法会写成  $p(x_{2087}) = p(x_{2087}|x_{39831})$ ；相反，这里只假设给定  $y$  时， $x_{2087}$  和  $x_{39831}$  是条件独立的。）

---

<sup>2</sup>实际上，在实践中会查看训练集并仅将至少出现一次的单词编码到特征向量中，而不是查看所有英文单词的列表。除了减少建模的单词数量，从而降低计算和空间需求外，这还有一个优点，即允许对可能出现在电子邮件中（例如“cs229”）但在字典中找不到的许多单词进行建模/包含作为特征。有时（在作业中）还会排除非常高频的单词（例如“the”、“of”和“and”）；这些高频的“无内容”词被称为**停用词 (stop words)**，因为它们在许多文档中都会出现，并且很少能表明电子邮件是垃圾邮件还是非垃圾邮件。

现在有：

$$\begin{aligned}
 p(x_1, \dots, x_{50000} | y) &= p(x_1 | y) p(x_2 | y, x_1) p(x_3 | y, x_1, x_2) \cdots p(x_{50000} | y, x_1, \dots, x_{49999}) \\
 &= p(x_1 | y) p(x_2 | y) p(x_3 | y) \cdots p(x_{50000} | y) \\
 &= \prod_{j=1}^d p(x_j | y)
 \end{aligned}$$

第一个等号仅遵循概率的通常性质，第二个等号使用了朴素贝叶斯假设。注意，尽管朴素贝叶斯假设是一个非常强的假设，但由此产生的算法在许多问题上效果很好。

模型由  $\phi_{j|y=1} = p(x_j = 1 | y = 1)$ 、 $\phi_{j|y=0} = p(x_j = 1 | y = 0)$  和  $\phi_y = p(y = 1)$  参数化。像往常一样，给定训练集  $\{(x^{(i)}, y^{(i)}); i = 1, \dots, n\}$ ，可以写出数据的联合似然：

$$\mathcal{L}(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}) = \prod_{i=1}^n p(x^{(i)}, y^{(i)}).$$

关于  $\phi_y$ 、 $\phi_{j|y=0}$  和  $\phi_{j|y=1}$  最大化此式，得到最大似然估计：

$$\begin{aligned}
 \phi_{j|y=1} &= \frac{\sum_{i=1}^n 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^n 1\{y^{(i)} = 1\}} \\
 \phi_{j|y=0} &= \frac{\sum_{i=1}^n 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^n 1\{y^{(i)} = 0\}} \\
 \phi_y &= \frac{\sum_{i=1}^n 1\{y^{(i)} = 1\}}{n}
 \end{aligned}$$

在上面的公式中，“ $\wedge$ ” 符号表示“逻辑且”。参数具有非常自然的解释。例如， $\phi_{j|y=1}$  是单词  $j$  出现在垃圾邮件 ( $y = 1$ ) 中的比例。

在拟合了所有这些参数之后，为了对具有特征  $x$  的新示例进行预测，只需计算：

$$\begin{aligned}
 p(y = 1 | x) &= \frac{p(x | y = 1) p(y = 1)}{p(x)} \\
 &= \frac{\left( \prod_{j=1}^d p(x_j | y = 1) \right) p(y = 1)}{\left( \prod_{j=1}^d p(x_j | y = 1) \right) p(y = 1) + \left( \prod_{j=1}^d p(x_j | y = 0) \right) p(y = 0)},
 \end{aligned}$$

并选择后验概率较高的类别。

最后，注意，虽然主要针对特征  $x_j$  是二值的问题开发了朴素贝叶斯算法，但可以很直接地将其推广到  $x_j$  可以取  $\{1, 2, \dots, k_j\}$  中的值，只需将  $p(x_j | y)$  建模为多项分布而不是伯努利分布。实际上，即使某些原始输入属性（例如，之前示

例中的房屋居住面积) 是连续值, 通常也会将其离散化 (discretize)——即将其转换为一小组离散值, 然后应用朴素贝叶斯。例如, 如果使用某个特征  $x_j$  来表示居住面积, 可以将连续值离散化如下:

居住面积 (平方英尺)	< 400	400-800	800-1200	1200-1600	> 1600
$x_i$	1	2	3	4	5

因此, 对于居住面积为 890 平方英尺的房屋, 可以将相应的特征  $x_j$  的值设置为 3。然后可以应用朴素贝叶斯算法, 并如前所述, 使用多项分布对  $p(x_j|y)$  进行建模。当原始的连续值属性无法通过多元正态分布很好地建模时, 离散化特征并使用朴素贝叶斯 (而不是 GDA) 通常会产生更好的分类器。

### 4.2.1 拉普拉斯平滑

如前所述, 朴素贝叶斯算法在许多问题上效果相当不错, 但有一个简单的修改可以使其效果更好, 尤其是在文本分类方面。简要讨论一下该算法当前形式存在的问题, 然后讨论如何解决它。

考虑垃圾邮件/电子邮件分类, 假设于 20xx 年, 在您 CS229 结课并出色地完成了项目后, 您决定在 20xx 年 5 月左右提交您的工作以在 NeurIPS 会议上发表。<sup>3</sup> 因为会在电子邮件中讨论该会议, 所以您也开始收到包含 “neurips” 一词的消息。但这是您的第一篇 NeurIPS 论文, 在此之前, 您从未见过包含 “neurips” 一词的电子邮件; 特别是在垃圾邮件/非垃圾邮件训练集中, “neurips” 从未出现过。假设 “neurips” 是字典中的第 35000 个单词, 您的朴素贝叶斯垃圾邮件过滤器会选择其最大似然估计参数  $\phi_{35000|y}$  为:

$$\phi_{35000|y=1} = \frac{\sum_{i=1}^n 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^n 1\{y^{(i)} = 1\}} = 0$$

$$\phi_{35000|y=0} = \frac{\sum_{i=1}^n 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^n 1\{y^{(i)} = 0\}} = 0$$

也就是说, 由于它从未在垃圾邮件或非垃圾邮件训练示例中见过 “neurips”, 它认为在任何类型的电子邮件中看到它的概率为零。因此, 当尝试判断这些包含 “neurips” 的消息是否是垃圾邮件时, 它会计算类别后验概率, 并得到

$$p(y=1|x) = \frac{\prod_{j=1}^d p(x_j|y=1)p(y=1)}{\prod_{j=1}^d p(x_j|y=1)p(y=1) + \prod_{j=1}^d p(x_j|y=0)p(y=0)} = \frac{0}{0}.$$

---

<sup>3</sup>NeurIPS 是顶级机器学习会议之一。提交论文的截止日期通常在五月至六月。

这是因为  $\prod_{j=1}^d p(x_j|y)$  的每一项都包含一个  $p(x_{35000}|y) = 0$  的项，该项与乘积相乘。因此，算法得到 0/0，无法做出预测。

更广泛地说，仅仅因为在有限的训练集中没有看到某个事件，就将其概率估计为零，这在统计上是一个糟糕的主意。考虑估计一个多项随机变量  $z$  取值为  $\{1, \dots, k\}$  的问题。可以用  $\phi_j = p(z = j)$  参数化多项分布。给定一组  $n$  个独立观测值  $\{z^{(1)}, \dots, z^{(n)}\}$ ，最大似然估计由下式给出：

$$\phi_j = \frac{\sum_{i=1}^n 1\{z^{(i)} = j\}}{n}.$$

正如之前看到的，如果使用这些最大似然估计，那么一些  $\phi_j$  可能会变为零，这是一个问题。为了避免这种情况，可以使用**拉普拉斯平滑 (Laplace smoothing)**，它将上述估计替换为：

$$\phi_j = \frac{1 + \sum_{i=1}^n 1\{z^{(i)} = j\}}{k + n}.$$

在这里，在分子中加了 1，在分母中加了  $k$ 。注意， $\sum_{j=1}^k \phi_j = 1$  仍然成立（自己验证一下！），这是一个理想的属性，因为  $\phi_j$  是概率的估计，而概率之和必须为 1。此外，对于所有  $j$  的值， $\phi_j \neq 0$ ，解决了概率估计为零的问题。在某些（可以说相当强的）条件下，可以证明拉普拉斯平滑实际上给出了  $\phi_j$  的最优估计。

回到朴素贝叶斯分类器，使用拉普拉斯平滑，因此得到以下参数估计：

$$\begin{aligned}\phi_{j|y=1} &= \frac{1 + \sum_{i=1}^n 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{2 + \sum_{i=1}^n 1\{y^{(i)} = 1\}} \\ \phi_{j|y=0} &= \frac{1 + \sum_{i=1}^n 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{2 + \sum_{i=1}^n 1\{y^{(i)} = 0\}}\end{aligned}$$

(在实践中，是否对  $\phi_y$  应用拉普拉斯平滑通常无关紧要，因为通常垃圾邮件和非垃圾邮件的比例都相当可观，所以  $\phi_y$  将是  $p(y = 1)$  的合理估计，并且无论如何都会远离 0。)

### 4.2.2 文本分类的事件模型

为了结束关于生成学习算法的讨论，下面讨论一个专门用于文本分类的模型。虽然前面介绍的朴素贝叶斯在许多分类问题上效果很好，但对于文本分类，有一个相关的模型效果更好。

在文本分类的具体背景下，前面介绍的朴素贝叶斯使用了**伯努利事件模型 (Bernoulli event model)**（或有时称为**多元伯努利事件模型 (multi-variate Bernoulli event model)**）。在该模型中，假设生成电子邮件的过程是：首先随

机确定（根据类别先验  $p(y)$ ）发送者是垃圾邮件发送者还是非垃圾邮件发送者。然后，发送电子邮件的人遍历字典，独立地决定是否根据概率  $p(x_j = 1|y) = \phi_{j|y}$  在该电子邮件中包含每个单词  $j$ 。因此，消息的概率由  $p(y) \prod_{j=1}^d p(x_j|y)$  给出。

这里有一个不同的模型，称为**多项事件模型 (Multinomial event model)**。为了描述该模型，将使用不同的表示法和特征集来表示电子邮件。令  $x_j$  表示电子邮件中第  $j$  个单词的标识。因此， $x_j$  现在是一个整数，取值范围为  $\{1, \dots, |V|\}$ ，其中  $|V|$  是词汇表（字典）的大小。包含  $d$  个单词的电子邮件现在由一个长度为  $d$  的向量  $(x_1, x_2, \dots, x_d)$  表示；注意  $d$  可以因不同的文档而异。例如，如果一封电子邮件以“A NeurIPS ...”开头，那么  $x_1 = 1$ （“a”是字典中的第一个单词）， $x_2 = 35000$ （如果“neurips”是字典中的第 35000 个单词）。

在多项事件模型中，假设生成电子邮件的过程是一个随机过程，其中首先确定（根据  $p(y)$ ）是垃圾邮件还是非垃圾邮件，这与之前相同。然后，发送电子邮件的人通过首先从某个多项分布  $p(x_1|y)$  生成  $x_1$  来编写电子邮件。接下来，第二个单词  $x_2$  独立于  $x_1$  但来自相同的多项分布中选择，对于  $x_3, x_4$  等也是如此，直到生成了电子邮件中的所有  $d$  个单词。因此，消息的总概率由  $p(y) \prod_{j=1}^d p(x_j|y)$  给出。注意，该公式看起来与之前在伯努利事件模型下消息概率的公式相同，但公式中的项现在表示完全不同的事物。特别是  $p(x_j|y)$  现在是一个多项分布而非伯努利分布。

新模型的参数是  $\phi_y = p(y)$  和  $\phi_{k|y=1} = p(x_j = k|y = 1)$ （对于任何  $j$ ）以及  $\phi_{k|y=0} = p(x_j = k|y = 0)$ 。注意，假设  $p(x_j|y)$  对于所有  $j$  的值都是相同的（即，生成单词的分布不依赖于它在电子邮件中的位置  $j$ ）。

如果给定训练集  $\{(x^{(i)}, y^{(i)}); i = 1, \dots, n\}$ ，其中  $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_{d_i}^{(i)})$ （此处， $d_i$  是第  $i$  个训练样本中的单词数），则数据的似然由下式给出：

$$\begin{aligned}\mathcal{L}(\phi_y, \phi_{k|y=0}, \phi_{k|y=1}) &= \prod_{i=1}^n p(x^{(i)}, y^{(i)}) \\ &= \prod_{i=1}^n \left( \prod_{j=1}^{d_i} p(x_j^{(i)}|y^{(i)}; \phi_{k|y=0}, \phi_{k|y=1}) \right) p(y^{(i)}; \phi_y).\end{aligned}$$

最大化此似然得到参数的最大似然估计：

$$\begin{aligned}\phi_{k|y=1} &= \frac{\sum_{i=1}^n \sum_{j=1}^{d_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\}}{\sum_{i=1}^n 1\{y^{(i)} = 1\} d_i} \\ \phi_{k|y=0} &= \frac{\sum_{i=1}^n \sum_{j=1}^{d_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\}}{\sum_{i=1}^n 1\{y^{(i)} = 0\} d_i} \\ \phi_y &= \frac{\sum_{i=1}^n 1\{y^{(i)} = 1\}}{n}.\end{aligned}$$

如果对估计  $\phi_{k|y=0}$  和  $\phi_{k|y=1}$  应用拉普拉斯平滑（在实践中为了获得良好性能是必需的），在分子中加 1，在分母中加  $|V|$ ，得到：

$$\begin{aligned}\phi_{k|y=1} &= \frac{1 + \sum_{i=1}^n \sum_{j=1}^{d_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\}}{|V| + \sum_{i=1}^n 1\{y^{(i)} = 1\} d_i} \\ \phi_{k|y=0} &= \frac{1 + \sum_{i=1}^n \sum_{j=1}^{d_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\}}{|V| + \sum_{i=1}^n 1\{y^{(i)} = 0\} d_i}.\end{aligned}$$

虽然朴素贝叶斯分类器不一定是最好的分类算法，但它通常效果出奇地好。考虑到其简单性和易于实现性，它通常也是一个非常好的“第一步”尝试。

# 第 5 章 核方法

## 5.1 特征映射

回顾在线性回归的讨论中，考虑了根据房屋的居住面积（记为  $x$ ）预测房屋价格（记为  $y$ ）的问题，并将  $x$  的线性函数拟合到训练数据。如果价格  $y$  可以更准确地表示为  $x$  的非线性 (*non-linear*) 函数呢？在这种情况下，需要一个比线性模型更具表现力的模型族。

首先考虑拟合三次函数  $y = \theta_3x^3 + \theta_2x^2 + \theta_1x + \theta_0$ 。结果表明，可以将三次函数视为在不同特征变量集（定义如下）上的线性函数。具体来说，令函数  $\phi : \mathbb{R} \rightarrow \mathbb{R}^4$  定义为

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4. \quad (5.1)$$

令  $\theta \in \mathbb{R}^4$  是包含  $\theta_0, \theta_1, \theta_2, \theta_3$  作为元素的向量。那么可以将三次函数写成  $x$  的形式：

$$\theta_3x^3 + \theta_2x^2 + \theta_1x + \theta_0 = \theta^T\phi(x).$$

因此，变量  $x$  的三次函数可以视为变量  $\phi(x)$  上的线性函数。为了区分这两组变量，在核方法的背景下，将问题的“原始”输入值称为输入属性 (**attributes**)（在本例中为  $x$ ，即居住面积）。当原始输入被映射到一组新的量  $\phi(x)$  时，将这些新的量称为特征 (**features**) 变量。（不幸的是，不同的作者在不同的语境下使用不同的术语来描述这两者。）将  $\phi$  称为特征映射 (**feature map**)，它将属性映射到特征。

## 5.2 带特征的最小均方

现在我们推导拟合模型  $\theta^T \phi(x)$  的梯度下降算法。首先回顾一下，对于普通的最小二乘问题，拟合  $\theta^T x$  的批量梯度下降更新（其推导参见讲义第一章）为：

$$\begin{aligned}\theta &:= \theta + \alpha \sum_{i=1}^n (y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \\ &:= \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})x^{(i)}.\end{aligned}\quad (5.2)$$

令  $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^p$  是一个将属性  $x$ （在  $\mathbb{R}^d$  中）映射到  $\mathbb{R}^p$  中的特征  $\phi(x)$  的特征映射。（在前面小节的示例中， $d = 1$  且  $p = 4$ 。）现在目标是拟合函数  $\theta^T \phi(x)$ ，其中  $\theta$  是  $\mathbb{R}^p$  中的向量而不是  $\mathbb{R}^d$  中的向量。可以将上面算法中  $x^{(i)}$  的所有出现替换为  $\phi(x^{(i)})$ ，得到新的更新：

$$\theta := \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)}))\phi(x^{(i)}).\quad (5.3)$$

类似地，相应的随机梯度下降更新规则是

$$\theta := \theta + \alpha(y^{(i)} - \theta^T \phi(x^{(i)}))\phi(x^{(i)}).\quad (5.4)$$

## 5.3 带核技巧的最小均方

当特征  $\phi(x)$  是高维时，上面的梯度下降或随机梯度下降更新在计算上变得昂贵。例如，考虑将方程 (5.1) 中的特征映射直接扩展到高维输入  $x$ ：假设  $x \in \mathbb{R}^d$ ，

令  $\phi(x)$  是包含所有  $x$  的次数  $\leq 3$  的项的向量

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_1^2 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_2 x_1 \\ \vdots \\ x_1^3 \\ x_1^2 x_2 \\ \vdots \\ x_2 x_3 x_1 \\ \vdots \end{bmatrix}. \quad (5.5)$$

特征  $\phi(x)$  的维度大约是  $d^3$  量级。<sup>1</sup> 对于计算来说，这是一个非常长的向量——当  $d = 1000$  时，每次更新需要至少计算和存储一个  $1000^3 = 10^9$  维向量，这比普通最小二乘更新规则 (5.2) 慢  $10^6$  倍。

乍一看，每次更新  $d^3$  的运行时长和内存使用似乎是不可避免的，因为向量  $\theta$  本身的维度是  $p \approx d^3$ ，并且可能需要更新和存储  $\theta$  的每一个元素。然而，将引入核技巧，通过它不需要显式存储  $\theta$ ，并且运行时长可以显著改善。

为简单起见，假设初始值  $\theta = 0$ ，并且关注迭代更新 (5.3)。主要观察是，在任何时候， $\theta$  都可以表示为向量  $\phi(x^{(1)}), \dots, \phi(x^{(n)})$  的线性组合。实际上，可以通过归纳法证明如下。在初始化时， $\theta = 0 = \sum_{i=1}^n 0 \cdot \phi(x^{(i)})$ 。假设在某个时刻， $\theta$  可以表示为

$$\theta = \sum_{i=1}^n \beta_i \phi(x^{(i)}). \quad (5.6)$$

其中  $\beta_1, \dots, \beta_n \in \mathbb{R}$ 。然后，断言在下一轮中， $\theta$  仍然是  $\phi(x^{(1)}), \dots, \phi(x^{(n)})$  的线

---

<sup>1</sup> 此处，为简单起见，包含所有重复的单项式（因此，例如  $x_1 x_2 x_3$  和  $x_2 x_3 x_1$  都出现在  $\phi(x)$  中）。因此， $\phi(x)$  中共有  $1 + d + d^2 + d^3$  个元素。

性组合，因为

$$\begin{aligned}
\theta &:= \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) \\
&= \sum_{i=1}^n \beta_i \phi(x^{(i)}) + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) \\
&= \sum_{i=1}^n \underbrace{(\beta_i + \alpha(y^{(i)} - \theta^T \phi(x^{(i)})))}_{\text{新的 } \beta_i} \phi(x^{(i)}). \tag{5.7}
\end{aligned}$$

可以意识到，一般策略是通过一组系数  $\beta_1, \dots, \beta_n$  隐式表示  $p$  维向量  $\theta$ 。为了做到这一点，推导系数  $\beta_i$  的更新规则。使用上面的方程，可以看到新的  $\beta_i$  依赖于旧的  $\beta_i$

$$\beta_i := \beta_i + \alpha(y^{(i)} - \theta^T \phi(x^{(i)})). \tag{5.8}$$

这里，方程的右侧仍然有旧的  $\theta$ 。将  $\theta$  替换为  $\theta = \sum_{j=1}^n \beta_j \phi(x^{(j)})$  得到

$$\forall i \in \{1, \dots, n\}, \beta_i := \beta_i + \alpha \left( y^{(i)} - \sum_{j=1}^n \beta_j \phi(x^{(j)})^T \phi(x^{(i)}) \right).$$

通常将  $\phi(x^{(j)})^T \phi(x^{(i)})$  重写为  $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$  以强调它是两个特征向量的内积。通过将  $\beta_i$  视为  $\theta$  的新表示，已经成功地将批梯度下降算法转化为一个迭代更新  $\beta$  值的新算法。它可能看起来在每次迭代时，仍然需要计算所有  $i, j$  对的  $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$  的值，其中每个计算可能需要大约  $O(p)$  的操作。然而，有两个重要的特性可以解决这个问题：

1. 在循环开始之前，可以预先计算所有  $i, j$  对的成对内积  $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$ 。
2. 对于定义在 (5.5) 中的特征映射  $\phi$ （或许多其他有趣的特征映射），计算  $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$  可以是高效的并且不必显式计算  $\phi(x^{(i)})$ 。这是因为：

$$\begin{aligned}
\langle \phi(x), \phi(z) \rangle &= 1 + \sum_{i=1}^d x_i z_i + \sum_{i,j \in \{1, \dots, d\}} x_i x_j z_i z_j + \sum_{i,j,k \in \{1, \dots, d\}} x_i x_j x_k z_i z_j z_k \\
&= 1 + \sum_{i=1}^d x_i z_i + \left( \sum_{i=1}^d x_i z_i \right)^2 + \left( \sum_{i=1}^d x_i z_i \right)^3 \\
&= 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3 \tag{5.9}
\end{aligned}$$

因此，要计算  $\langle \phi(x), \phi(z) \rangle$ ，可以首先用  $O(d)$  的时间计算  $\langle x, z \rangle$ ，然后进行一些常数次操作来计算  $1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3$ 。

正如将看到的，特征  $\phi(x), \phi(z)$  之间的内积在这里至关重要。将与特征映射  $\phi$  对应的核 (Kernel) 定义为一个  $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  的函数，满足：<sup>2</sup>

$$K(x, z) \triangleq \langle \phi(x), \phi(z) \rangle \quad (5.10)$$

最终算法总结如下：

1. 用方程 (5.9) 对所有  $i, j \in \{1, \dots, n\}$  计算  $K(x^{(i)}, x^{(j)}) \triangleq \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ 。  
设置  $\beta := 0$ 。

## 2. 循环：

$$\forall i \in \{1, \dots, n\}, \beta_i := \beta_i + \alpha \left( y^{(i)} - \sum_{j=1}^n \beta_j K(x^{(i)}, x^{(j)}) \right). \quad (5.11)$$

或者用向量表示法，令  $K$  是一个  $n \times n$  矩阵，其中  $K_{ij} = K(x^{(i)}, x^{(j)})$ ，有

$$\beta := \beta + \alpha(\vec{y} - K\beta)$$

通过上面的算法，可以在每次更新时以  $O(n)$  的时间高效地更新向量  $\theta$  的表示  $\beta$ 。最后，需要证明表示  $\beta$  的知识足以计算预测  $\theta^T \phi(x)$ 。实际上，有

$$\theta^T \phi(x) = \sum_{i=1}^n \beta_i \phi(x^{(i)})^T \phi(x) = \sum_{i=1}^n \beta_i K(x^{(i)}, x). \quad (5.12)$$

可以意识到，本质上所有需要知道的关于特征映射  $\phi(\cdot)$  的信息都封装在相应的核函数  $K(\cdot, \cdot)$  中。将在下一节中对此进行详细阐述。

## 5.4 核的性质

在上一小节中，从一个显式定义的特征映射  $\phi$  开始，导出了核函数  $K(x, z) \triangleq \langle \phi(x), \phi(z) \rangle$ 。然后，看到核函数是如此本质，只要核函数被定义，整个训练算法就可以完全用核方法的语言编写，而无需引用特征映射  $\phi$ ，因此对于测试示例  $x$  的预测（方程 (5.12)）也是如此。

因此，可以尝试定义其他核函数  $K(\cdot, \cdot)$  并运行算法 (5.11)。请注意，算法 (5.11) 不需要显式访问特征映射  $\phi$ ，因此只需要确保特征映射  $\phi$  的存在，但不一定需要能够显式写下  $\phi$ 。

---

<sup>2</sup>回想一下， $\mathcal{X}$  是输入  $x$  的取值空间。在当前示例中， $\mathcal{X} = \mathbb{R}^d$ 。

哪些类型的函数  $K(\cdot, \cdot)$  可以对应于某个特征映射  $\phi$ ? 换句话说, 如果存在某个特征映射  $\phi$  使得对于所有  $x, z$  都有  $K(x, z) = \phi(x)^T \phi(z)$ , 能否判断出来?

如果可以通过给出有效核函数的精确表征来回答这个问题, 那么就可以完全改变选择核函数  $K$  的接口, 而不是选择特征映射  $\phi$  的接口。具体来说, 可以选取一个函数  $K$ , 验证它满足该表征 (从而存在一个与  $K$  对应的特征映射  $\phi$ ), 然后就可以运行更新规则 (5.11)。这里的好处是, 不需要能够计算或解析地写下  $\phi$ , 只需要知道它的存在性。在本小节的末尾, 在通过几个具体的核示例之后, 将回答这个问题。

假设  $x, z \in \mathbb{R}^d$ , 首先考虑函数  $K(\cdot, \cdot)$  定义为:

$$K(x, z) = (x^T z)^2.$$

也可以将其写为

$$K(x, z) = \left( \sum_{i=1}^d x_i z_i \right) \left( \sum_{j=1}^d x_j z_j \right) \quad (5.13)$$

$$= \sum_{i=1}^d \sum_{j=1}^d x_i x_j z_i z_j \quad (5.14)$$

$$= \sum_{i,j=1}^d (x_i x_j)(z_i z_j) \quad (5.15)$$

因此, 可以看到  $K(x, z) = \langle \phi(x), \phi(z) \rangle$  是与特征映射  $\phi$  对应的核函数 (这里以  $d = 3$  的情况为例) 由下式给出

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}.$$

请回想一下核的计算效率, 注意, 虽然计算高维的  $\phi(x)$  需要  $O(d^2)$  的时间, 但找到  $K(x, z)$  只需  $O(d)$  的时间——与输入属性的维度呈线性关系。

对于另一个相关的例子, 也考虑由下式定义的  $K(\cdot, \cdot)$ :

$$K(x, z) = (x^T z + c)^2$$

$$= \sum_{i,j=1}^d (x_i x_j)(z_i z_j) + \sum_{i=1}^d (\sqrt{2c} x_i)(\sqrt{2c} z_i) + c^2.$$

(请自行验证) 这个函数  $K$  是一个核函数, 它对应到特征映射 (再次以  $d = 3$  为例)

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \\ \sqrt{2c} x_1 \\ \sqrt{2c} x_2 \\ \sqrt{2c} x_3 \\ c \end{bmatrix},$$

其中参数  $c$  控制  $x_i$  (一阶) 项和  $x_i x_j$  (二阶) 项之间的相对权重。

更广泛地说, 核  $K(x, z) = (x^T z + c)^k$  对应于一个特征空间, 包含所有次数不超过  $k$  的一元多项式  $x_{i_1} x_{i_2} \dots x_{i_k}$ 。然而尽管在这个  $O(d^k)$  维的高维空间中工作, 计算  $K(x, z)$  仍然只需要  $O(d)$  的时间, 因此即使在如此高维的特征空间中, 也永远不需要显式表示出特征向量。

## 核作为相似性度量

现在, 讨论一下核的另一种视角。直观地 (尽管这种直觉有一些问题, 但先忽略), 如果  $\phi(x)$  和  $\phi(z)$  彼此接近, 那么可以期望  $K(x, z) = \phi(x)^T \phi(z)$  很大。反过来, 如果  $\phi(x)$  和  $\phi(z)$  相距很远——例如, 几乎相互正交——那么  $K(x, z) = \phi(x)^T \phi(z)$  将很小。因此, 可以将  $K(x, z)$  看作是  $\phi(x)$  和  $\phi(z)$  之间相似性的一种度量, 或者说是  $x$  和  $z$  之间相似性的一种度量。

有了这种直觉, 假设对于正在研究的某个学习问题, 想出了一个函数  $K(x, z)$ , 认为它可能是衡量  $x$  和  $z$  相似性的合理度量。例如, 可能选择了

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right).$$

这是衡量  $x$  和  $z$  相似性的合理度量，当  $x$  和  $z$  接近时接近 1，当  $x$  和  $z$  相距很远时接近 0。是否存在一个特征映射  $\phi$  使得上面定义的核  $K$  满足  $K(x, z) = \phi(x)^T \phi(z)$ ? 在这个特殊的例子中，答案是肯定的。这个核被称为高斯核 (Gaussian kernel)，并且对应于一个无限维的特征映射  $\phi$ 。下面，我们将精确地描述一个函数  $K$  需要满足哪些性质才能成为一个有效的核函数，即对应于某个特征映射  $\phi$ 。

## 有效核的必要条件

现在假设  $K$  确实是一个有效的核，对应于某个特征映射  $\phi$ ，首先看看它满足哪些性质。考虑一个包含  $n$  个点（不一定是训练集）的有限集合  $\{x^{(1)}, \dots, x^{(n)}\}$ ，并定义一个  $n \times n$  的方阵  $K$ ，其  $(i, j)$  项由  $K_{ij} = K(x^{(i)}, x^{(j)})$  给出。这个矩阵称为核矩阵 (kernel matrix)。请注意，我们为了方便使用了  $K$  来表示核函数  $K(x, z)$  和核矩阵  $K$ ，因为它们之间有明显的密切关系。

如果  $K$  是一个有效的核，那么有  $K_{ij} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = \phi(x^{(j)})^T \phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{ji}$ ，因此  $K$  必须是对称的。此外，令  $\phi_k(x)$  表示向量  $\phi(x)$  的第  $k$  个坐标，对于任意向量  $z$ ，有

$$\begin{aligned} z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\ &= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\ &= \sum_i \sum_j z_i \left( \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) \right) z_j \\ &= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\ &= \sum_k \left( \sum_i z_i \phi_k(x^{(i)}) \right) \left( \sum_j z_j \phi_k(x^{(j)}) \right) \\ &= \sum_k \left( \sum_i z_i \phi_k(x^{(i)}) \right)^2 \\ &\geq 0. \end{aligned}$$

倒数第二步利用了  $\sum_{i,j} a_i a_j = (\sum_i a_i)^2$ ，其中  $a_i = z_i \phi_k(x^{(i)})$ 。由于  $z$  是任意的，这表明  $K$  是半正定的 ( $K \geq 0$ )。

因此，证明出，如果  $K$  是一个有效的核（即，它对应于某个特征映射  $\phi$ ），那么相应的核矩阵  $K \in \mathbb{R}^{n \times n}$  是对称半正定的。

## 有效核的充分条件

更一般地，上面的条件不仅是必要的，而且也是  $K$  成为有效核（也称为 Mercer 核）的充分条件。以下结果归功于 Mercer。<sup>3</sup>

**定理 (Mercer)**: 设  $K : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$  已知。则  $K$  是一个有效核的充要条件是，对于任意有限集合  $\{x^{(1)}, \dots, x^{(n)}\}$  ( $n < \infty$ )，相应的核矩阵是对称半正定的。

给定一个函数  $K$ ，除了尝试找到一个与之对应的特征映射  $\phi$  之外，这个定理因此提供了另一种测试它是否是有效核的方法。在问题集 2 中也将有机会进一步探索这些想法。

课上还简要讨论了几个其他核的例子。例如，考虑手写数字识别问题，给定一个手写数字 (0-9) 的图像 (16x16 像素)，需要确定它是哪个数字。使用简单的多项式核  $K(x, z) = (x^T z)^k$  或高斯核，SVM 能够在此问题上获得非常好的性能。这尤其令人惊讶，因为输入属性  $x$  仅仅是图像像素值的 256 维向量，系统没有关于视觉的先验知识，甚至没有关于哪些像素相邻的知识。在课堂上简要讨论的另一个例子是，如果试图对字符串进行分类（例如， $x$  是由氨基酸串成的蛋白质），那么构建一个合理、“小”的特征集对于大多数学习算法来说似乎很困难，特别是如果不同的字符串长度不同。然而，考虑让  $\phi(x)$  是一个特征向量，它计算  $x$  中每个长度为  $k$  的子字符串的出现次数。如果考虑英文字母组成的字符串，那么有  $26^k$  个这样的字符串。因此， $\phi(x)$  是一个  $26^k$  维向量；即使对于中等大小的  $k$ ，这可能也太大了，无法有效处理（例如， $26^4 \approx 460000$ ）。但是，使用（类似动态规划的）字符串匹配算法，可以有效地计算  $K(x, z) = \phi(x)^T \phi(z)$ ，这样就可以在这个  $26^k$  维特征空间中隐式地工作，而无需显式计算特征向量。

## 核方法的应用

已经看到了核方法在线性回归中的应用。在下一部分，将介绍支持向量机，核方法可以直接应用于其中。这里不再赘述。实际上，核方法的思想比线性回归和 SVM 具有更广泛的适用性。具体来说，如果有一个学习算法，可以完全用输入属性向量之间的内积  $\langle x, z \rangle$  来表达，那么通过将其替换为核  $K(x, z)$ （其中  $K$  是一个核），就可以“神奇地”让算法在对应于  $K$  的高维特征空间中高效工作。例如，这个核技巧可以应用于感知机，推导出核感知机算法。本课程后面将看到的许多算法也将适用于这种方法，这种方法被称为“核技巧”。

---

<sup>3</sup>许多文献以涉及  $L^2$  函数的稍微更复杂的形式给出 Mercer 定理，但当输入属性取值为  $\mathbb{R}^d$  时，这里给出的版本是等价的。

# 第 6 章 支持向量机

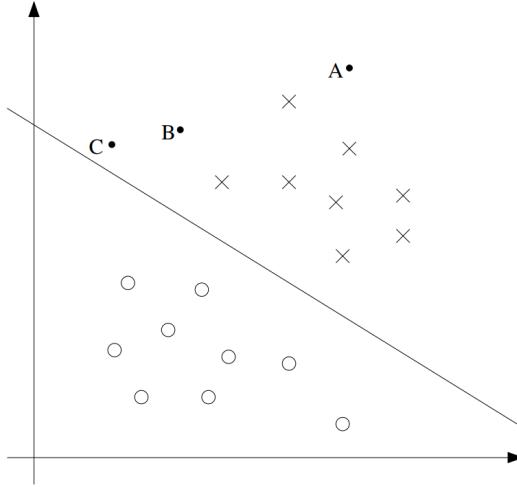
本章将介绍支持向量机 (SVM) 学习算法。SVM 是最好的（许多人甚至认为是最好的）“开箱即用”的监督学习算法之一。为了讲述 SVM 的故事，首先需要讨论间隔 (margin) 以及用大间隔分隔数据的思想。接下来，将讨论最优间隔分类器，这将引出对拉格朗日对偶性的讨论。还将看到核，它允许在非常高维（例如无限维）特征空间中高效地应用 SVM，最后会以 SMO 算法作为结束，该算法提供了 SVM 的高效实现。

## 6.1 间隔：直觉

我们将从讨论间隔开始讲述 SVM 的故事。本节将给出关于间隔以及预测的“置信度”的直觉；这些思想将在第 6.3 节中正式化。

考虑逻辑回归，其中概率  $p(y = 1|x; \theta)$  由  $h_\theta(x) = g(\theta^T x)$  建模。当且仅当  $h_\theta(x) \geq 0.5$  时，或者等价地，当且仅当  $\theta^T x \geq 0$  时，预测输入  $x$  的标签为 “1”。考虑一个正训练样本 ( $y = 1$ )。 $\theta^T x$  越大， $h_\theta(x) = p(y = 1|x; \theta)$  也越大，因此预测标签为 1 的“置信度”也越高。因此，可以非正式地认为，如果  $\theta^T x \gg 0$ ，则预测非常确信标签为 1。类似地，如果  $\theta^T x \ll 0$ ，则认为逻辑回归非常确信预测标签为 0。给定一个训练集，再次非正式地看来，如果能找到  $\theta$  使得当  $y^{(i)} = 1$  时  $\theta^T x^{(i)} \gg 0$ ，并且当  $y^{(i)} = 0$  时  $\theta^T x^{(i)} \ll 0$ ，这将反映出一个非常确信（且正确）的分类集，其适用于所有训练样本。这似乎是一个很好的目标，很快将使用函数间隔的概念来形式化这个想法。

为了获得另一种直觉，考虑下面的图，其中  $x$  表示正训练样本， $o$  表示负训练样本，还显示了决策边界（这是由方程  $\theta^T x = 0$  给出的直线，也称为**分隔超平面 (separating hyperplane)**），并且还标记了三个点 A、B 和 C。



注意到点 A 离决策边界非常远。如果要求预测 A 点的  $y$  值，似乎应该非常确信那里  $y = 1$ 。相反，点 C 离决策边界非常近，尽管它在决策边界预测  $y = 1$  的一侧，但决策边界的微小变化似乎很容易导致预测结果为  $y = 0$ 。因此，在 A 点的预测比在 C 点更确信。点 B 介于这两种情况之间，更广泛地说，可以看出，如果一个点离分隔超平面很远，那么预测可能会更确信。同样，非正式地认为，如果给定一个训练集，能够找到一个决策边界，使得在训练样本上做出所有正确且确信（即远离决策边界）的预测，那将是很好的。稍后将使用几何间隔的概念来形式化这个想法。

## 6.2 符号 (选读)

为了更容易讨论 SVM，首先需要引入一种新的符号来讨论分类。将考虑一个二元分类问题的线性分类器，其标签为  $y$ ，特征为  $x$ 。从现在开始，使用  $y \in \{-1, 1\}$ （而不是  $\{0, 1\}$ ）来表示类别标签。此外，不再使用向量  $\theta$  来参数化线性分类器，而是使用参数  $w, b$ ，并将分类器写为

$$h_{w,b}(x) = g(w^T x + b).$$

这里， $g(z) = 1$  如果  $z \geq 0$ ，否则  $g(z) = -1$ 。这种“ $w, b$ ”符号允许将截距项  $b$  与其他参数分开处理（还放弃了之前让  $x_0 = 1$  成为输入特征向量中的额外坐标的约定）。因此， $b$  扮演着之前  $\theta_0$  的角色，而  $w$  扮演着  $[\theta_1 \dots \theta_d]^T$  的角色。

还注意到，根据上面  $g$  的定义，分类器将直接预测 1 或 -1（参见感知器算法），而无需先经过估计  $p(y = 1)$  的中间步骤（这是逻辑回归所做的）。

### 6.3 函数间隔与几何间隔 (选读)

现在将函数间隔和几何间隔的概念形式化。给定一个训练样本  $(x^{(i)}, y^{(i)})$ , 定义  $(w, b)$  相对于训练样本的**函数间隔 (functional margin)** 为

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b).$$

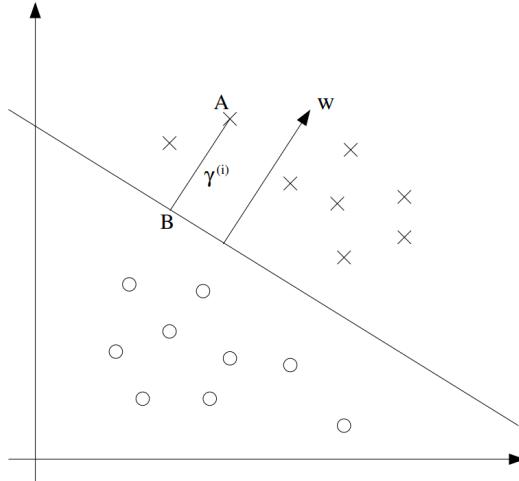
注意, 如果  $y^{(i)} = 1$ , 那么为了使函数间隔大 (即, 预测是确信且正确的), 需要  $w^T x^{(i)} + b$  是一个大的正数。相反, 如果  $y^{(i)} = -1$ , 那么为了使函数间隔大, 需要  $w^T x^{(i)} + b$  是一个大的负数。此外, 如果  $y^{(i)}(w^T x^{(i)} + b) > 0$ , 那么对这个样本的预测是正确的。(自己验证一下。) 因此, 大的函数间隔表示确信且正确的预测。

对于上面给出的  $g$  选择的线性分类器 (取值在  $\{-1, 1\}$  中), 然而函数间隔有一个性质使得它不是一个非常好的置信度度量。考虑  $g$  的选择, 注意到如果将  $w$  替换为  $2w$ , 将  $b$  替换为  $2b$ , 那么由于  $g(w^T x + b) = g(2w^T x + 2b)$ , 这根本不会改变  $h_{w,b}(x)$ 。也就是说,  $g$ , 以及因此  $h_{w,b}(x)$ , 仅取决于  $w^T x + b$  的符号, 而不是大小。然而, 将  $(w, b)$  替换为  $(2w, 2b)$  也会导致函数间隔乘以 2。因此, 似乎通过利用缩放  $w$  和  $b$  的自由度, 可以在不改变任何有意义的东西的情况下使函数间隔任意大。直观地说, 因此施加某种归一化条件 (例如  $\|w\|_2 = 1$ ) 可能是有意义的; 也就是说, 可以将  $(w, b)$  替换为  $(w/\|w\|_2, b/\|w\|_2)$ , 并改为考虑  $(w/\|w\|_2, b/\|w\|_2)$  的函数间隔。稍后将回到这一点。

给定一个训练集  $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, n\}$ , 还定义  $(w, b)$  相对于  $S$  的函数间隔为各个训练样本的函数间隔中的最小值。用  $\hat{\gamma}$  表示, 因此可以写为:

$$\hat{\gamma} = \min_{i=1, \dots, n} \hat{\gamma}^{(i)}.$$

接下来, 讨论**几何间隔 (geometric margins)**。考虑下面的图:



图中显示了与  $(w, b)$  对应的决策边界以及向量  $w$ 。注意  $w$  与分隔超平面正交（呈  $90^\circ$  角）（您应该说服自己确实如此）。考虑 A 点，它代表标记为  $y^{(i)} = 1$  的某个训练样本的输入  $x^{(i)}$ 。它到决策边界的距离  $\gamma^{(i)}$  由线段 AB 给出。

如何找到  $\gamma^{(i)}$  的值？ $w/\|w\|$  是一个单位长度向量，方向与  $w$  相同。由于 A 代表  $x^{(i)}$ ，因此可以发现点 B 由  $x^{(i)} - \gamma^{(i)} \cdot w/\|w\|$  给出。但这个点位于决策边界上，而决策边界上的所有点  $x$  都满足方程  $w^T x + b = 0$ 。因此，

$$w^T \left( x^{(i)} - \gamma^{(i)} \frac{w}{\|w\|} \right) + b = 0.$$

解出  $\gamma^{(i)}$  得到

$$\gamma^{(i)} = \frac{w^T x^{(i)} + b}{\|w\|} = \left( \frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|}.$$

这是针对图中 A 点处正训练样本的情况推导的，其中位于决策边界的“正”侧是好的。更一般地，定义  $(w, b)$  相对于训练样本  $(x^{(i)}, y^{(i)})$  的几何间隔为

$$\gamma^{(i)} = y^{(i)} \left( \left( \frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \right).$$

注意，如果  $\|w\| = 1$ ，那么函数间隔等于几何间隔——这提供了一种关联这两种不同间隔概念的方式。此外，几何间隔对于参数的缩放是不变的；也就是说，如果用  $2w$  和  $2b$  替换  $w$  和  $b$ ，那么几何间隔不会改变。这在后面会很有用。具体来说，由于参数缩放的这种不变性，在试图拟合  $w$  和  $b$  到训练数据时，可以在不改变任何重要内容的情况下对  $w$  施加任意缩放约束；例如，可以要求  $\|w\| = 1$ ，或者  $|w_1| = 5$ ，或者  $|w_1 + b| + |w_2| = 2$ ，并且这些都可以通过缩放  $w$  和  $b$  简单地满足。

最后，给定一个训练集  $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, n\}$ ，还定义  $(w, b)$  相对于  $S$  的几何间隔为各个训练样本的几何间隔中的最小值：

$$\gamma = \min_{i=1, \dots, n} \gamma^{(i)}.$$

## 6.4 最优间隔分类器 (选读)

给定一个训练集，从之前的讨论来看，一个自然期望是找到一个决策边界，使其几何间隔最大化，因为这将反映一组在训练集上非常确信的预测，并且其能很好地“拟合”训练数据。具体来说，这将产生一个分类器，它用一个“间隔”（几何间隔）来分隔正训练样本和负训练样本。

目前，假设给定一个线性可分的训练集；也就是说，可以通过某个分隔超平面来分隔正样本和负样本。如何找到能实现最大几何间隔的那个呢？可以提出以下优化问题：

$$\begin{aligned} \max_{\gamma, w, b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \dots, n \\ & \|w\| = 1. \end{aligned}$$

也就是说，希望最大化  $\gamma$ ，约束条件是每个训练样本的函数间隔至少为  $\gamma$ 。此外， $\|w\| = 1$  约束确保了函数间隔等于几何间隔，因此也保证了所有几何间隔至少为  $\gamma$ 。因此，解决这个问题将得到  $(w, b)$ ，使其相对于训练集具有最大的可能几何间隔。

如果能解决上面的优化问题，任务就完成了。但是，“ $\|w\| = 1$ ” 是一个令人讨厌的（非凸）约束，并且这种问题肯定不是标准优化软件可以解决的格式。因此，尝试将问题转换为更友好的形式。考虑：

$$\begin{aligned} \max_{\gamma, w, b} \quad & \frac{\hat{\gamma}}{\|w\|} \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \dots, n \end{aligned}$$

这里，要最大化  $\hat{\gamma}/\|w\|$ ，约束条件是所有函数间隔至少为  $\hat{\gamma}$ 。由于几何间隔和函数间隔通过  $\gamma = \hat{\gamma}/\|w\|$  相关，这将得到想要的答案。此外，摆脱了不喜欢的  $\|w\| = 1$  约束。缺点是现在有一个令人讨厌的（还是非凸的）目标函数  $\frac{\hat{\gamma}}{\|w\|}$ ；而且，仍然没有现成的软件可以解决这种形式的优化问题。

再想想。回想之前关于可以在不改变任何内容的情况下对  $w$  和  $b$  添加任意缩放约束的讨论。这是现在要使用的关键思想。引入缩放约束，使得  $(w, b)$  相对于训练集的函数间隔必须为 1：

$$\hat{\gamma} = 1.$$

由于将  $w$  和  $b$  乘以某个常数会导致函数间隔也乘以相同的常数，这确实是一个缩放约束，并且可以通过缩放  $w, b$  来满足。将其代入上面的问题，并注意到最大化  $\hat{\gamma}/\|w\| = 1/\|w\|$  等价于最小化  $\|w\|^2$ ，现在有了以下优化问题：

$$\begin{aligned} \min_{w, b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \dots, n \end{aligned}$$

现在已经将问题转换成可以有效求解的形式。上述问题是一个具有凸二次目标函数和线性约束的优化问题。其解给出了**最优间隔分类器 (optimal margin classifier)**。可以使用商用二次规划 (QP) 代码来解决此优化问题。<sup>1</sup>

虽然可以称此问题已解决，但下文会进一步讨论拉格朗日对偶性。这将导出优化问题的对偶形式，使得我们可以应用核函数让最优间隔分类器在极高维空间中高效工作。对偶形式还将允许推导出一种求解上述优化问题的高效算法，该算法通常比通用 QP 软件表现更好。

## 6.5 拉格朗日对偶性 (选读)

暂时搁置支持向量机和最大间隔分类器，讨论如何解决带约束的优化问题。

考虑以下形式的问题：

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & h_i(w) = 0, \quad i = 1, \dots, l. \end{aligned}$$

一些读者可能还记得如何使用拉格朗日乘数法来解决此问题。（如果以前没有见过，不用担心。）在此方法中，定义**拉格朗日函数 (Lagrangian)** 为

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^l \beta_i h_i(w)$$

这里， $\beta_i$  称为**拉格朗日乘数 (Lagrange multipliers)**。然后找到  $\mathcal{L}$  的偏导数并将其设为零：

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0; \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0,$$

并求解  $w$  和  $\beta$ 。

在本节中，将此推广到可能包含不等式约束和等式约束的优化问题。由于时间限制，无法在本课程中充分论述拉格朗日对偶性理论，<sup>2</sup> 但将给出主要思想和结果，并将其应用于最优间隔分类器的优化问题。

考虑以下问题，将其称为**原始 (primal)** 优化问题：

---

<sup>1</sup>您可能熟悉线性规划，它解决目标函数和约束均是线性的优化问题。QP 软件也广泛可用，它允许凸二次目标函数和线性约束。

<sup>2</sup>对这部分感兴趣的读者，建议阅读，例如，R. T. Rockafeller (1970), Convex Analysis, Princeton University Press.

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \dots, k \\ & h_i(w) = 0, \quad i = 1, \dots, l. \end{aligned}$$

为了解决它，首先定义广义拉格朗日函数 (generalized Lagrangian)：

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^k \alpha_i g_i(w) + \sum_{i=1}^l \beta_i h_i(w).$$

这里， $\alpha_i$  和  $\beta_i$  是拉格朗日乘数。考虑以下量

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta).$$

这里，下标 “ $\mathcal{P}$ ” 代表 “原始”。给定某个  $w$ 。如果  $w$  违反任何原始约束（即，对于某个  $i$ ，要么  $g_i(w) > 0$  要么  $h_i(w) \neq 0$ ），那么应该能够验证

$$\begin{aligned} \theta_{\mathcal{P}}(w) &= \max_{\alpha, \beta: \alpha_i \geq 0} f(w) + \sum_{i=1}^k \alpha_i g_i(w) + \sum_{i=1}^l \beta_i h_i(w) \\ &= \infty. \end{aligned} \tag{6.1}$$

反之，如果约束对于某个特定的  $w$  值确实满足，那么  $\theta_{\mathcal{P}}(w) = f(w)$ 。因此，

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{若 } w \text{ 满足原始约束} \\ \infty & \text{其他情况.} \end{cases}$$

因此，对于满足原始约束的所有  $w$  值， $\theta_{\mathcal{P}}$  取与问题中目标函数相同的值，如果违反约束，则为正无穷大。因此，如果考虑最小化问题

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta),$$

可以看出它与原始问题相同（即，具有与原始问题相同的解）。为了后续使用，还将目标函数的最优值定义为  $p^* = \min_w \theta_{\mathcal{P}}(w)$ ；将其称为原始问题的值 (value)。

现在，考察一个稍微不同的问题。定义

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta).$$

这里，下标 “ $\mathcal{D}$ ” 代表 “对偶”。注意，在  $\theta_{\mathcal{P}}$  的定义中，关于  $\alpha, \beta$  进行优化（最大化），而这里关于  $w$  进行最小化。

现在可以提出对偶 (dual) 优化问题:

$$\max_{\alpha, \beta: \alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta).$$

这与上面显示的原始问题完全相同，只是“max”和“min”的顺序现在交换了。将对偶问题目标函数的最优值定义为  $d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \theta_{\mathcal{D}}(w)$ 。

原始问题和对偶问题如何关联？很容易证明

$$d^* = \max_{\alpha, \beta: \alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \min_w \max_{\alpha, \beta: \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^*.$$

(您应该自己证明这一点；这源自函数“max min”总是小于或等于“min max”。) 然而，在某些条件下，将有

$$d^* = p^*,$$

这样就可以求解对偶问题来代替原始问题。来看看这些条件是什么。

假设  $f$  和  $g_i$  是凸函数，<sup>3</sup> 并且  $h_i$  是仿射函数。<sup>4</sup> 进一步假设约束  $g_i$  是（严格）可行的；这意味着存在某个  $w$  使得对于所有  $i$ ，都有  $g_i(w) < 0$ 。

在上述假设下，必然存在  $w^*, \alpha^*, \beta^*$ ，使得  $w^*$  是原始问题的解， $\alpha^*, \beta^*$  是对偶问题的解，并且  $p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$ 。此外， $w^*, \alpha^*$  和  $\beta^*$  满足 **Karush-Kuhn-Tucker (KKT) 条件**，如下：

$$\frac{\partial}{\partial w_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \dots, d \tag{6.3}$$

$$\frac{\partial}{\partial \beta_i} \mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \quad i = 1, \dots, l \tag{6.4}$$

$$\alpha_i^* g_i(w^*) = 0, \quad i = 1, \dots, k \tag{6.5}$$

$$g_i(w^*) \leq 0, \quad i = 1, \dots, k \tag{6.6}$$

$$\alpha^* \geq 0, \quad i = 1, \dots, k \tag{6.7}$$

此外，如果某个  $w^*, \alpha^*, \beta^*$  满足 KKT 条件，那么它也是原始问题和对偶问题的解。

请注意方程 (6.5)，它被称为 **KKT 对偶互补性条件 (dual complementarity)**。具体来说，这意味着如果  $\alpha_i^* > 0$ ，那么  $g_i(w^*) = 0$ 。（即，“ $g_i(w) \leq 0$ ”约

---

<sup>3</sup>当  $f$  具有 Hessian 矩阵时，当且仅当 Hessian 矩阵是半正定时，它是凸的；类似地，所有线性（和仿射）函数也是凸的。（函数  $f$  也可以在不可微的情况下是凸的，但这里不需要那些更一般的凸性定义。）

<sup>4</sup>即，存在  $a_i, b_i$ ，使得  $h_i(w) = a_i^T w + b_i$ 。“仿射”的意思与线性相同，只是允许额外的截距项  $b_i$ 。

束是活跃 (active) 的，意味着它以等式而不是不等式成立。) 稍后，这将是证明 SVM 只有少量“支持向量”的关键；KKT 对偶互补性条件也将为讨论 SMO 算法时提供收敛性测试。

## 6.6 最优间隔分类器：对偶形式（选读）

**注：**优化问题 (6.8) 与优化问题 (6.12) 的等价性，以及方程 (6.10) 中原始变量和对偶变量之间的关系是本节最重要的收获。

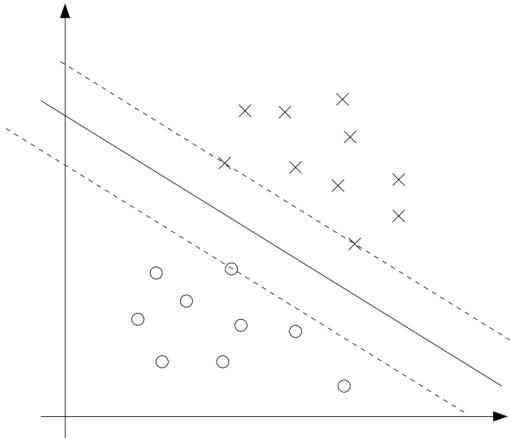
前面，提出了以下寻找最优间隔分类器的问题（原始）优化问题：

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \dots, n \end{aligned} \quad (6.8)$$

可以将约束写成

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0.$$

每个训练样本都有一个这样的约束。注意，根据 KKT 对偶互补性条件，只有当训练样本的功能间隔恰好等于 1 时（即，对应于满足等式  $g_i(w) = 0$  的约束）， $\alpha_i$  才大于 0。考虑下图中，其中实线表示最大间隔分离超平面。



间隔最小的点恰好是离决策边界最近的点；在这里，这三个点（一个负样本和两个正样本）位于平行于决策边界的虚线上。因此，只有这三个  $\alpha_i$ （即，对应于这三个训练样本的  $\alpha_i$ ）在最优解处非零。这三个点被称为该问题中的**支持向量 (support vectors)**。支持向量的数量可以远小于训练集的大小，这一事实将在稍后有用。

接下来，在推导对偶形式问题时，需要注意的一个关键思想是，我们将尝试仅使用内积  $\langle x^{(i)}, x^{(j)} \rangle$ （将其视为输入特征空间中点之间的  $(x^{(i)})^T x^{(j)}$ ）来编写算法。算法能够用这些内积表达的事实是应用核技巧的关键。

构建优化问题的拉格朗日函数为：

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]. \quad (6.9)$$

注意，只有“ $\alpha_i$ ”而没有“ $\beta_i$ ”拉格朗日乘子，因为问题只有不等式约束。

继续推导问题的对偶形式。为此，需要首先对固定的  $\alpha$  最小化  $\mathcal{L}(w, b, \alpha)$  关于  $w$  和  $b$ ，以得到  $\theta_D$ 。方法是将  $\mathcal{L}$  关于  $w$  和  $b$  的导数设为零。得到：

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^n \alpha_i y^{(i)} x^{(i)} = 0$$

这意味着

$$w = \sum_{i=1}^n \alpha_i y^{(i)} x^{(i)}. \quad (6.10)$$

关于  $b$  的导数，得到

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^n \alpha_i y^{(i)} = 0. \quad (6.11)$$

将式 (6.10) 中  $w$  的定义代入拉格朗日函数 (6.9)，并简化，得到

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^n \alpha_i y^{(i)}.$$

但根据式 (6.11)，最后一项必须为零，所以得到

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle.$$

回想一下，通过对  $\mathcal{L}$  关于  $w$  和  $b$  进行最小化，得到了上面的方程。将其与约束  $\alpha_i \geq 0$ （始终存在）和约束 (6.11) 结合，得到以下对偶优化问题：

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0. \end{aligned} \quad (6.12)$$

还应该能够验证  $p^* = d^*$  所需的条件以及 KKT 条件（式 (6.3)-(6.7)）在我们的优化问题里确实满足。因此，可以通过求解对偶问题来求解原问题。具体而言，在上述对偶问题中，这是一个最大化问题，其中的参数是  $\alpha_i$ 。稍后将讨论用于求解对偶问题的具体算法，但如果确实能够求解它（即找到使  $W(\alpha)$  最大化且满足约束的  $\alpha$ ），那么就可以使用式 (6.10) 回过头来找到最优的  $w$  作为  $\alpha$  的函数。找到  $w^*$  后，通过考虑原问题，也很容易找到截距项  $b$  的最优值，如下所示：

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}. \quad (6.13)$$

（可以自行验证一下此处的正确性。）

继续之前，再仔细看看式 (6.10)，它给出了  $w$  的最优值（以最优  $\alpha$  表示）。假设已经用训练集拟合了模型参数，现在想对新的输入点  $x$  进行预测。然后计算  $w^T x + b$ ，并且当且仅当该值大于零时预测  $y = 1$ 。但是使用 (6.10)，该值也可以写成：

$$w^T x + b = \left( \sum_{i=1}^n \alpha_i y^{(i)} x^{(i)} \right)^T x + b \quad (6.14)$$

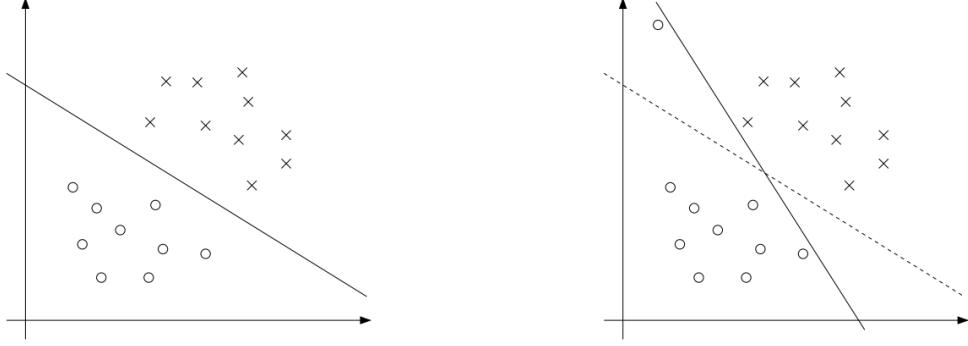
$$= \sum_{i=1}^n \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \quad (6.15)$$

因此，如果找到了  $\alpha_i$ ，为了进行预测，必须计算一个仅取决于  $x$  与训练集中的点之间的内积的值。此外，之前已经看到，除了支持向量对应的  $\alpha_i$  外，其他所有  $\alpha_i$  都将为零。因此，上面求和中的许多项将为零，并且实际上只需要计算  $x$  与支持向量（通常数量很少）之间的内积，以便计算 (6.15) 并进行预测。

通过研究优化问题的对偶形式，对问题的结构获得了重要的见解，并且还能够仅以内积形式编写整个算法。在下一节中，将利用这一特性将核函数应用于分类问题。由此产生的算法，**支持向量机 (support vector machines)**，将能够在非常高维的空间中高效学习。

## 6.7 正则化与非线性可分情况 (选读)

目前为止介绍的支持向量机推导假设数据是线性可分的。虽然通过  $\phi$  将数据映射到高维特征空间通常会增加数据可分的可能性，但不能保证总是如此。此外，在某些情况下，找到一个分离超平面并非完全符合期望，因为它可能对离群点敏感。例如，下面左图显示了一个最优间隔分类器，当在左上方区域（右图）添加一个离群点时，决策边界会发生剧烈变化，并且得到的分类器的间隔会小得多。



为了使算法也适用于非线性可分数据集并降低对离群点的敏感度，我们重新构建了优化问题（使用  $\ell_1$  正则化 (regularization)），如下所示：

$$\begin{aligned} \min_{\gamma, w, b} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \dots, n \\ & \xi_i \geq 0, \quad i = 1, \dots, n. \end{aligned}$$

因此，现在允许存在（函数）间隔小于 1 的样本，如果一个样本的函数间隔为  $1 - \xi_i$  (其中  $\xi_i > 0$ )，那么目标函数会增加  $C\xi_i$  的代价。参数  $C$  控制了两个目标之间的相对权重：使  $\|w\|^2$  小（如前面所述，这会使间隔大）以及确保大多数样本的函数间隔至少为 1。

与之前一样，可以构建拉格朗日函数：

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i] - \sum_{i=1}^n r_i \xi_i.$$

这里， $\alpha_i$  和  $r_i$  是拉格朗日乘子（约束为  $\geq 0$ ）。将不再详细推导对偶问题，但在像之前一样将关于  $w$  和  $b$  的导数设为零，然后代入并简化后，得到问题的对偶形式：

$$\begin{aligned} \max \alpha \quad & W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y^{(i)} = 0. \end{aligned}$$

与之前一样，同样有  $w$  可以表示为  $\alpha_i$  的函数，如式 (6.10) 所示，因此在求解对偶问题后，可以继续使用式 (6.15) 进行预测。注意，令人惊讶的是，在添加

$\ell_1$  正则化后，对偶问题唯一的改变是原先的约束  $0 \leq \alpha_i$  现在变成了  $0 \leq \alpha_i \leq C$ 。对  $b^*$  的计算也必须进行修改（式 (6.13) 不再有效）；请参阅下一节中的注释或 Platt 的论文。

此外，KKT 对偶互补条件（在下一节中将用于测试 SMO 算法的收敛性）为：

$$\alpha_i = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad (6.16)$$

$$\alpha_i = C \Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1 \quad (6.17)$$

$$0 < \alpha_i < C \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1. \quad (6.18)$$

现在，剩下的就是给出一个实际求解对偶问题的算法，这将在下一节中进行。

## 6.8 SMO 算法 (选读)

序列最小化 (sequential minimal optimization, SMO) 算法，由 John Platt 提出，提供了一种有效求解由 SVM 推导得到的对偶问题的方法。部分是为了引出 SMO 算法，部分是因为它本身很有趣，让我们先再进行一次岔开讨论，谈谈坐标上升算法。

### 6.8.1 坐标上升

考虑尝试求解无约束优化问题

$$\max_{\alpha} W(\alpha_1, \alpha_2, \dots, \alpha_n).$$

这里，将  $W$  视为参数  $\alpha_i$  的某个函数，暂时忽略这个问题与支持向量机之间的任何关系。之前已经见过两种优化算法：梯度上升和牛顿法。这里要考慮的新算法称为**坐标上升 (coordinate ascent)**：

循环直到收敛：{

对于  $i = 1, \dots, n$ , {

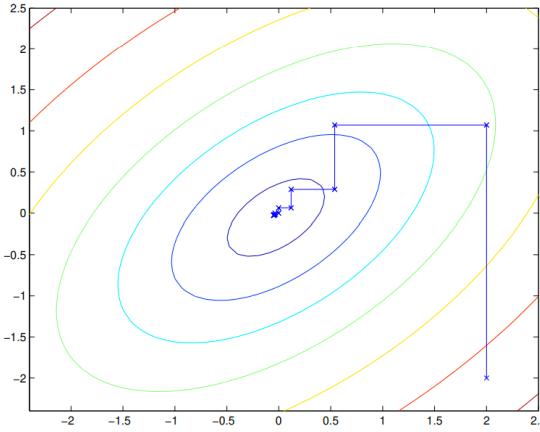
$$\alpha_i := \arg \max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_n).$$

}

}

因此，在该算法的最内层循环中，将固定除某个  $\alpha_i$  之外的所有变量，并仅针对参数  $\alpha_i$  重新优化  $W$ 。然后内层循环按  $\alpha_1, \alpha_2, \dots, \alpha_n, \alpha_1, \alpha_2, \dots$  的顺序重新优化变量。（更复杂的版本可以选择其他顺序；例如，可以根据期望哪个变量能使  $W(\alpha)$  增加最大来选择下一个要更新的变量。）

当函数  $W$  的形式恰好使得内层循环中的“ $\arg \max$ ”可以高效执行时，坐标上升可以是一个相当高效的算法。这里是坐标上升算法的一个图例：



图中的椭圆是我们想要优化的二次函数的等高线。坐标上升从  $(2, -2)$  初始化，图中绘制了它通往全局最大值的路径。注意到在每一步，坐标上升都沿着与一个坐标轴平行的方向前进，因为每次只优化一个变量。

### 6.8.2 SMO

通过概述 SMO 算法的推导来结束对支持向量机的讨论。

这是我们想要解决的（对偶）优化问题：

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \quad (6.19)$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \quad (6.20)$$

$$\sum_{i=1}^n \alpha_i y^{(i)} = 0. \quad (6.21)$$

假设我们有一组满足约束 ((6.20)-(6.21)) 的  $\alpha_i$ 。现在，假设固定  $\alpha_2, \dots, \alpha_n$ ，并进行坐标上升步骤，关于  $\alpha_1$  重新优化目标函数。能取得任何进展吗？答案是否定的，因为约束 (6.21) 确保

$$\alpha_1 y^{(1)} = - \sum_{i=2}^n \alpha_i y^{(i)}.$$

或者，通过将两边乘以  $y^{(1)}$ ，我们等价地得到

$$\alpha_1 = -y^{(1)} \sum_{i=2}^n \alpha_i y^{(i)}.$$

(这一步使用了  $y^{(1)} \in \{-1, 1\}$  的事实, 因此  $(y^{(1)})^2 = 1$ 。) 因此,  $\alpha_1$  完全由其他  $\alpha_i$  确定, 如果固定  $\alpha_2, \dots, \alpha_n$ , 那么在不违反优化问题中的约束 (6.21) 的情况下, 无法对  $\alpha_1$  进行任何改变。

因此, 如果想要更新一些  $\alpha_i$ , 必须同时更新至少两个, 以便保持约束得到满足。这促使了 SMO 算法, 其简单来说就是以下步骤:

重复直到收敛 {

1. 选择一对  $\alpha_i$  和  $\alpha_j$  进行下一次更新 (使用启发式方法来选择能够最大程度接近全局最大值的两个变量)。
2. 在保持所有其他  $\alpha_k$  ( $k \neq i, j$ ) 固定的情况下, 重新优化  $W(\alpha)$  关于  $\alpha_i$  和  $\alpha_j$ 。

}

为了测试该算法的收敛性, 可以检查 KKT 条件 (式 (6.16)-(6.18)) 是否在某个容差 ( $tol$ ) 内得到满足。这里, 容差是收敛容差参数, 通常设置为 0.01 到 0.001。(详细信息请参阅论文和伪代码。)

SMO 算法高效的关键原因在于  $\alpha_i, \alpha_j$  的更新可以非常高效地计算。现在简要概述推导高效更新的主要思路。

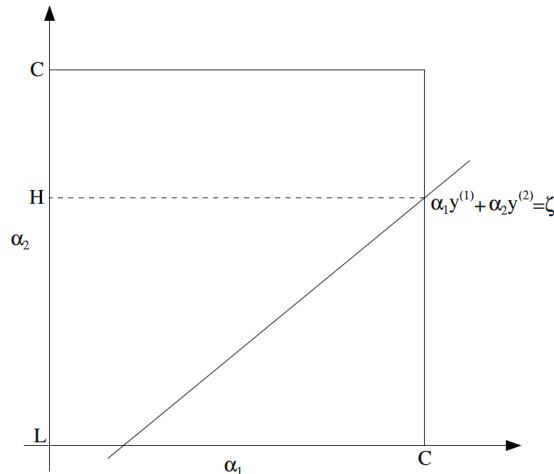
假设当前有一组满足约束 ((6.20)-(6.21)) 的  $\alpha_i$ , 并且固定  $\alpha_3, \dots, \alpha_n$ , 然后重新优化  $W(\alpha_1, \alpha_2, \dots, \alpha_n)$  关于  $\alpha_1$  和  $\alpha_2$  (受约束限制)。根据 (6.21), 我们要求

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^n \alpha_i y^{(i)}.$$

由于右侧是固定的 (因为已经固定了  $\alpha_3, \dots, \alpha_n$ ), 可以将其记为某个常数  $\zeta$ :

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta. \quad (6.22)$$

因此, 可以将  $\alpha_1$  和  $\alpha_2$  的约束可视化如下:



从约束 (6.20) 中，我们知道  $\alpha_1$  和  $\alpha_2$  必须位于所示的盒子  $[0, C] \times [0, C]$  内。还绘制了直线  $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$ ，我们知道  $\alpha_1$  和  $\alpha_2$  必须位于该直线上。另外注意，从这些约束中，我们知道  $L \leq \alpha_2 \leq H$ ；否则， $(\alpha_1, \alpha_2)$  无法同时满足盒子约束和直线约束。在此示例中， $L = 0$ 。但这取决于直线  $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$  的样子，情况并非总是如此；更普遍地，对于  $\alpha_2$  的允许值，会有一个下界  $L$  和一个上界  $H$ ，以确保  $\alpha_1, \alpha_2$  位于盒子  $[0, C] \times [0, C]$  内。

使用方程 (6.22)，我们还可以将  $\alpha_1$  写成  $\alpha_2$  的函数：

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}.$$

(请自行验证此推导；我们再次使用了  $y^{(1)} \in \{-1, 1\}$  的事实，因此  $(y^{(1)})^2 = 1$ 。) 因此，目标函数  $W(\alpha)$  可以写成

$$W(\alpha_1, \alpha_2, \dots, \alpha_n) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \dots, \alpha_n).$$

将  $\alpha_3, \dots, \alpha_n$  视为常数，您应该能够验证这是关于  $\alpha_2$  的二次函数。也就是说，这也可以表示为  $a\alpha_2^2 + b\alpha_2 + c$  的形式，其中  $a, b, c$  是适当的常数。如果忽略“盒子”约束 (6.20)（或等价地，忽略  $L \leq \alpha_2 \leq H$ ），那么我们可以通过将其导数设为零并求解来轻松最大化此二次函数。我们将令  $\alpha_2^{\text{new,unclipped}}$  表示由此产生的  $\alpha_2$  值。您还应该能够说服自己，如果改为最大化关于  $\alpha_2$  的  $W$ ，但受盒子约束限制，那么可以通过简单地取  $\alpha_2^{\text{new,unclipped}}$  并将其“剪裁”到  $[L, H]$  区间内，得到

$$\alpha_2^{\text{new}} = \begin{cases} H & \text{若 } \alpha_2^{\text{new,unclipped}} > H \\ \alpha_2^{\text{new,unclipped}} & \text{若 } L \leq \alpha_2^{\text{new,unclipped}} \leq H \\ L & \text{若 } \alpha_2^{\text{new,unclipped}} < L \end{cases}$$

最后，找到  $\alpha_2^{\text{new}}$  后，可以使用方程 (6.22) 返回并找到  $\alpha_1^{\text{new}}$  的最优值。

还有一些细节非常简单，但我们将留给自己在 Platt 的论文中阅读：一个是用于选择下一个要更新的  $\alpha_i, \alpha_j$  的启发式方法的选择；另一个是 SMO 算法运行时如何更新  $b$ 。

## **第二部分**

### **深度学习**

# 第 7 章 深度学习

现在让我们开始学习深度学习。在本部分中，将概述神经网络，讨论向量化，并讨论使用反向传播训练神经网络。

## 7.1 使用非线性模型的监督学习

在监督学习设置中（从输入  $x$  预测  $y$ ），假设我们的模型/假设是  $h_\theta(x)$ 。过去的章节里考虑了  $h_\theta(x) = \theta^T x$ （线性回归）或  $h_\theta(x) = \theta^T \phi(x)$ （其中  $\phi(x)$  是特征映射）的情况。这两个模型的共同点是它们在参数  $\theta$  中是线性的。接下来，我们将考虑学习在参数  $\theta$  和输入  $x$  中都**非线性**（non-linear）的一般模型族。最常见的非线性模型是神经网络，我们将在下一节开始定义。对于本节，将  $h_\theta(x)$  视为抽象的非线性模型即可。<sup>1</sup>

假设  $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$  是训练样本。我们将定义非线性模型及其学习的损失/成本函数。

### 回归问题

为简单起见，我们从输出是实数的情况开始，即  $y^{(i)} \in \mathbb{R}$ ，因此模型  $h_\theta(x)$  也输出实数  $h_\theta(x) \in \mathbb{R}$ 。将学习第  $i$  个样本  $(x^{(i)}, y^{(i)})$  的最小二乘成本函数定义为

$$J^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2, \quad (7.1)$$

数据集的均方成本函数定义为

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n J^{(i)}(\theta), \quad (7.2)$$

这与线性回归中的定义相同，只是为了与约定一致，我们在成本函数前引入了常数  $1/n$ 。注意，将成本函数乘以一个标量不会改变成本函数的局部最小点或全局

---

<sup>1</sup>如果需要一个具体的例子，可以考虑模型  $h_\theta(x) = \theta_1^2 x_1^2 + \theta_2^2 x_2^2 + \dots + \theta_d^2 x_d^2$ ，尽管它不是神经网络。

最小点。另请注意，尽管成本函数的形式也是均方损失，但  $h_\theta(x)$  的参数化与线性回归的情况不同。在后文中，“损失”和“成本”这两个词会互换使用。

## 二元分类

接下来定义二元分类的模型和损失函数。假设输入  $x \in \mathbb{R}^d$ 。令  $\bar{h}_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$  是一个参数化模型（逻辑线性回归中  $\theta^T x$  的类比）。将输出  $\bar{h}_\theta(x) \in \mathbb{R}$  称为 logit。类似于第 2.1 节，使用 logistic 函数  $g(\cdot)$  将 logit  $\bar{h}_\theta(x)$  转换为概率  $h_\theta(x) \in [0, 1]$ ：

$$h_\theta(x) = g(\bar{h}_\theta(x)) = 1/(1 + \exp(-\bar{h}_\theta(x))). \quad (7.3)$$

使用以下方式建模给定  $x$  和  $\theta$  的  $y$  的条件分布：

$$\begin{aligned} P(y = 1 | x; \theta) &= h_\theta(x) \\ P(y = 0 | x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

按照第 2.1 节中的相同推导并使用备注 2.1.1 中的推导，负对数似然损失函数等于：

$$J^{(i)}(\theta) = -\log p(y^{(i)} | x^{(i)}; \theta) = \ell_{\text{logistic}}(\bar{h}_\theta(x^{(i)}), y^{(i)}) \quad (7.4)$$

如公式 (7.2) 中所示，总损失函数也定义为对单个训练样本的损失函数的平均值， $J(\theta) = \frac{1}{n} \sum_{i=1}^n J^{(i)}(\theta)$ 。

## 多类别分类

按照第 2.3 节，考虑响应变量  $y$  可以取  $k$  个值之一的分类问题，即  $y \in \{1, 2, \dots, k\}$ 。令  $\bar{h}_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^k$  是一个参数化模型。将输出  $\bar{h}_\theta(x) \in \mathbb{R}^k$  称为 logits。每个 logit 对应于  $k$  个类别之一的预测。类似于第 2.3 节，使用 softmax 函数将 logits  $\bar{h}_\theta(x)$  转换为一个非负且和为 1 的概率向量：

$$P(y = j | x; \theta) = \frac{\exp(\bar{h}_\theta(x)_j)}{\sum_{s=1}^k \exp(\bar{h}_\theta(x)_s)}, \quad (7.5)$$

其中  $\bar{h}_\theta(x)_s$  表示  $\bar{h}_\theta(x)$  的第  $s$  个坐标。类似于第 2.3 节，单个训练样本  $(x^{(i)}, y^{(i)})$  的损失函数是其负对数似然：

$$J^{(i)}(\theta) = -\log p(y^{(i)} | x^{(i)}; \theta) = -\log \left( \frac{\exp(\bar{h}_\theta(x^{(i)})_{y^{(i)}})}{\sum_{s=1}^k \exp(\bar{h}_\theta(x^{(i)})_s)} \right). \quad (7.6)$$

使用第 2.3 节的符号，可以简单地抽象地写成：

$$J^{(i)}(\theta) = \ell_{\text{ce}}(\bar{h}_\theta(x^{(i)}), y^{(i)}). \quad (7.7)$$

损失函数也定义为单个训练样本的损失函数的平均值,  $J(\theta) = \frac{1}{n} \sum_{i=1}^n J^{(i)}(\theta)$ 。

还注意到, 上述方法也可以推广到任何条件概率模型, 其中有一个关于  $y$  的指数族分布  $\text{Exponential-family}(y; \eta)$ , 其中  $\eta = \bar{h}_\theta(x)$  是一个参数化的非线性函数  $x$ 。然而, 最广泛的情况还是上面讨论的三种情况。

## 优化器 (SGD)

通常使用梯度下降 (GD)、随机梯度下降 (SGD) 或其变体来优化损失函数  $J(\theta)$ 。GD 的更新规则可以写为<sup>2</sup>

$$\theta := \theta - \alpha \nabla_\theta J(\theta) \quad (7.8)$$

其中  $\alpha > 0$  通常称为学习率或步长。接下来, 介绍一个新的 SGD (算法 1), 它与前面所讲的略有不同。

---

### Algorithm 1: 随机梯度下降

---

- 1 超参数: 学习率  $\alpha$ , 总迭代数  $n_{\text{iter}}$
- 2 随机初始化  $\theta$
- 3 **for**  $i = 1$  **to**  $n_{\text{iter}}$  **do**
- 4     从  $\{1, \dots, n\}$  中均匀采样一个  $j$ , 然后更新  $\theta$

$$\theta := \theta - \alpha \nabla_\theta J^{(j)}(\theta) \quad (7.9)$$

---

通常, 因为硬件并行化, 同时计算  $B$  个样本关于参数  $\theta$  的梯度比单独计算  $B$  个梯度要快。因此, 深度学习中更常用的是小批量随机梯度下降, 如算法 2 所示。还有一些 SGD 或小批量 SGD 的变体, 它们使用略微不同的采样方案。

---

<sup>2</sup>回顾一下, 如之前所定义, 使用符号 “ $a := b$ ” 表示一个操作 (在计算机程序中), 其中将变量  $a$  的值设置为  $b$ 。换句话说, 此操作用  $b$  的值覆盖  $a$ 。相比之下, 当断言事实陈述  $a$  的值等于  $b$  的值时, 将写成 “ $a = b$ ”。

---

**Algorithm 2:** 小批量随机梯度下降
 

---

- 1 超参数: 学习率  $\alpha$ , 总迭代数  $n_{iter}$
- 2 随机初始化  $\theta$
- 3 **for**  $i = 1$  **to**  $n_{iter}$  **do**
- 4     从  $\{1, \dots, n\}$  中无放回地均匀采样  $B$  个样本  $j_1, \dots, j_B$ , 然后更新  $\theta$

$$\theta := \theta - \frac{\alpha}{B} \sum_{k=1}^B \nabla_{\theta} J^{(j_k)}(\theta) \quad (7.10)$$

---

使用这些通用算法, 典型的深度学习模型通过以下步骤进行学习。1. 定义神经网络参数化  $h_{\theta}(x)$ , 将在第 7.2 节中介绍, 2. 编写反向传播算法以有效计算损失函数  $J^{(j)}(\theta)$  的梯度, 这将在第 7.4 节中介绍, 以及 3. 使用损失函数  $J(\theta)$  运行 SGD 或小批量 SGD (或其他基于梯度的优化器)。

## 7.2 神经网络

神经网络是指一类广泛的非线性模型/参数化方法, 涉及矩阵乘法和其它逐元素非线性运算的组合。为了统一处理回归问题和分类问题, 此处将  $\bar{h}_{\theta}(x)$  视为神经网络的输出。对于回归问题, 最终预测  $h_{\theta}(x) = \bar{h}_{\theta}(x)$ ; 对于分类问题,  $\bar{h}_{\theta}(x)$  是 logits, 二分类的预测概率是  $h_{\theta}(x) = 1/(1 + \exp(-\bar{h}_{\theta}(x)))$  (参见公式 (7.3)), 多分类的则是  $h_{\theta}(x) = \text{softmax}(\bar{h}_{\theta}(x))$  用于 (参见公式 (7.5))。

接下来将从小到大逐步构建一个神经网络。

### 单神经元神经网络

回顾之前的房价预测问题: 给定房屋大小, 预测价格。将在本小节中以此为例进行说明。

之前, 将直线拟合到房屋大小与房价的关系图上。现在, 不拟合直线, 希望通过将绝对最低价格设置为零来防止负房价。这在图 7.1 中产生一个“拐点”。如何用具有单个拐点且参数未知的  $\bar{h}_{\theta}(x)$  来表示这样的函数? (完成此操作后, 可以调用第 7.1 节中的机制。)

定义一个参数化的函数  $\bar{h}_{\theta}(x)$ , 以  $x$  为输入, 由  $\theta$  参数化, 输出房屋价格  $y$ 。形式上,  $\bar{h}_{\theta} : x \rightarrow y$ 。最简单的参数化也许是

$$\bar{h}_{\theta}(x) = \max(wx + b, 0), \text{其中 } \theta = (w, b) \in \mathbb{R}^2 \quad (7.11)$$

此处  $\bar{h}_\theta(x)$  返回一个值:  $(wx + b)$  或 0 的较大者。在神经网络的上下文中, 函数  $\max\{t, 0\}$  被称为 ReLU (读作 “ray-lu” ), 或修正线性单元 (Rectified Linear Unit), 通常记作  $\text{ReLU}(t) \triangleq \max\{t, 0\}$ 。

通常, 将  $\mathbb{R}$  映射到  $\mathbb{R}$  的一维非线性函数 (例如 ReLU) 通常被称为**激活函数 (activation function)**。模型  $\bar{h}_\theta(x)$  被称为具有单个神经元, 部分原因是它具有单个非线性激活函数。(稍后将更详细地讨论为什么非线性激活函数被称为神经元。)

当输入  $x \in \mathbb{R}^d$  具有多个维度时, 具有单个神经元的神经网络可以写为

$$\bar{h}_\theta(x) = \text{ReLU}(w^\top x + b), \text{其中 } w \in \mathbb{R}^d, b \in \mathbb{R}, \text{且 } \theta = (w, b) \quad (7.12)$$

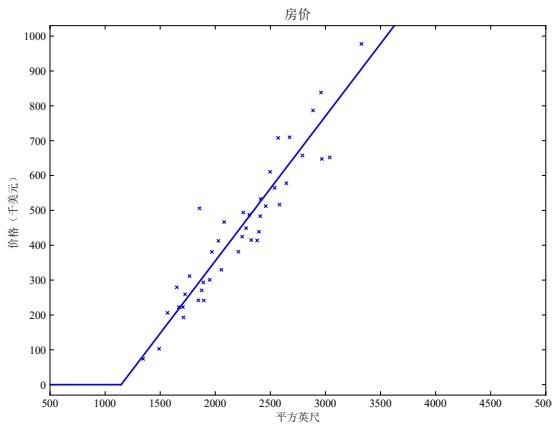


图 7.1: 带拐点的房价拟合

项  $b$  通常被称为“偏置”, 向量  $w$  被称为权重向量。这样的神经网络称之为 1 层。(后续将定义多层的含义。)

## 堆叠神经元

更复杂的神经网络可以将上述单个神经元“堆叠”在一起, 使得一个神经元的输出作为下一个神经元的输入, 从而产生更复杂的函数。现在来深化房价预测的例子。除了房屋大小, 假设还知道卧室数量、邮政编码和社区财富。构建神经网络类似于乐高积木: 将单个积木堆叠在一起以构建复杂的结构。这同样适用于神经网络: 将单个神经元堆叠在一起以创建复杂的神经网络。给定这些特征 (大小、卧室数量、邮政编码和财富), 可能会决定房屋价格取决于其可容纳的最大家庭人数。假设家庭人数是房屋大小和卧室数量的函数 (参见图 7.2)。邮政编码可以提供额外信息, 例如社区的步行便利程度 (即, 是否可以步行到杂货店或需

要开车)。结合邮政编码和社区财富可以预测当地小学的质量。可以认为，房屋价格最终取决于这三个特征。

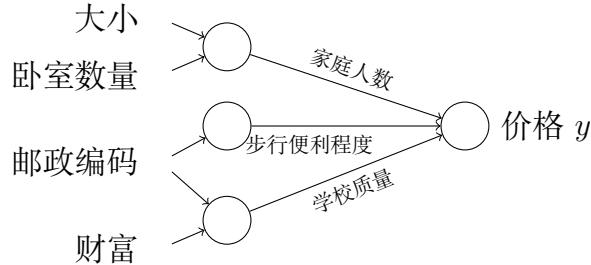


图 7.2: 预测房屋价格的小型神经网络示意图

形式上，神经网络的输入是一组输入特征  $x_1, x_2, x_3, x_4$ 。将“家庭人数”、“步行便利程度”和“学校质量”的中间变量记为  $a_1, a_2, a_3$ (这些  $a_i$  通常被称为“隐藏单元”或“隐藏神经元”)。将每个  $a_i$  表示为以  $x_1, \dots, x_4$  的子集为输入的单神经元神经网络。然后，如在图 7.1 中，将有以下参数化：

$$\begin{aligned} a_1 &= \text{ReLU}(\theta_1 x_1 + \theta_2 x_2 + \theta_3) \\ a_2 &= \text{ReLU}(\theta_4 x_3 + \theta_5) \\ a_3 &= \text{ReLU}(\theta_6 x_3 + \theta_7 x_4 + \theta_8) \end{aligned}$$

其中  $(\theta_1, \dots, \theta_8)$  是参数。现在将最终输出  $\bar{h}_\theta(x)$  表示为以  $a_1, a_2, a_3$  为输入的另一个线性函数，得到<sup>3</sup>

$$\bar{h}_\theta(x) = \theta_9 a_1 + \theta_{10} a_2 + \theta_{11} a_3 + \theta_{12} \quad (7.13)$$

其中  $\theta$  包含所有参数  $(\theta_1, \dots, \theta_{12})$ 。

现在将输出表示为以参数  $\theta$  为输入的相当复杂的函数  $x$ 。然后可以使用第 7.1 节中的机制来学习参数  $\theta$ 。

## 生物神经网络的启发

顾名思义，人工神经网络受到生物神经网络的启发。隐藏单元  $a_1, \dots, a_m$  对应于生物神经网络中的神经元，参数  $\theta_i$  对应于突触。然而，现代深度人工神经网络与生物神经网络的相似程度尚不清楚。例如，也许没有多少神经科学家认为

---

<sup>3</sup>通常，对于多层神经网络，在末端，靠近输出处，不会应用 ReLU，特别是当输出不一定是正数时。

生物神经网络可以有 1000 层，而一些现代人工神经网络则不然（将在后续中详细阐述层的概念）。此外，人类大脑更新神经网络的方式是否与计算机科学家学习人工神经网络的方式（使用反向传播，将在下一节介绍）相似，这是一个悬而未决的问题。

## 两层全连接神经网络

通过利用关于“家庭人数”、“步行便利程度”和“学校质量”如何由输入决定的重要先验知识/信念，构建了公式 (7.13) 中的神经网络。隐含地假设知道家庭人数是一个重要的数量，并且它只能由“大小”和“卧室数量”确定。这样的先验知识可能不适用于其他应用。更灵活和通用的方法是为中间变量  $a_i$  编写一个通用的参数化，作为所有  $x_1, \dots, x_4$  的函数：

$$\begin{aligned} a_1 &= \text{ReLU}(w_1^\top x + b_1), \text{ 其中 } w_1 \in \mathbb{R}^4 \text{ 且 } b_1 \in \mathbb{R} \\ a_2 &= \text{ReLU}(w_2^\top x + b_2), \text{ 其中 } w_2 \in \mathbb{R}^4 \text{ 且 } b_2 \in \mathbb{R} \\ a_3 &= \text{ReLU}(w_3^\top x + b_3), \text{ 其中 } w_3 \in \mathbb{R}^4 \text{ 且 } b_3 \in \mathbb{R} \end{aligned} \quad (7.14)$$

仍然使用上面定义的  $a_1, a_2, a_3$  来定义  $\bar{h}_\theta(x)$ 。这样就得到了一个所谓的**全连接神经网络 (fully-connected neural network)**，因为所有中间变量  $a_i$  都依赖于所有输入  $x_i$ 。

为了通用性，具有  $m$  个隐藏单元和  $d$  维输入  $x \in \mathbb{R}^d$  的两层全连接神经网络定义为

$$\forall j \in \{1, \dots, m\}, \quad z_j = w_j^{[1]\top} x + b_j^{[1]} \quad \text{其中 } w_j^{[1]} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R} \quad (7.15)$$

$$\begin{aligned} a_j &= \text{ReLU}(z_j), \\ a &= [a_1, \dots, a_m]^\top \in \mathbb{R}^m \end{aligned}$$

$$\bar{h}_\theta(x) = w^{[2]\top} a + b^{[2]} \quad \text{其中 } w^{[2]} \in \mathbb{R}^m, b^{[2]} \in \mathbb{R} \quad (7.16)$$

请注意，默认情况下， $\mathbb{R}^d$  中的向量被视为列向量，特别是  $a$  是一个分量为  $a_1, a_2, \dots, a_m$  的列向量。索引 <sup>[1]</sup> 和 <sup>[2]</sup> 用于区分两组参数： $w_j^{[1]}$ （每个都是  $\mathbb{R}^d$  中的向量）和  $w^{[2]}$ （它是  $\mathbb{R}^m$  中的向量）。稍后将更详细地讨论这些。

## 向量化

在引入具有更多层和更复杂结构的神经网络之前，将使用更多的矩阵和向量符号简化神经网络的表达式。向量化的另一个重要动机是实现中的速度。为了有

效地实现神经网络，在使用循环时必须小心。在代码中实现公式 (7.15) 的最自然方法可能是使用 for 循环。在实践中，当输入和隐藏单元的维度很高时，使用循环会导致代码运行非常慢。利用 GPU 的并行性对于深度学习的发展至关重要。

这催生了向量化。向量化不是使用循环，而是利用矩阵代数和高度优化的数值线性代数库（例如 BLAS）来快速运行神经网络计算。在深度学习时代之前，for 循环可能足以处理较小的数据集，但现代深度网络和最先进的数据集使用 for 循环将不可行。

将下面的两层全连接神经网络向量化。将权重矩阵  $W^{[1]} \in \mathbb{R}^{m \times d}$  定义为所有向量  $w_j^{[1]}$  的串联，如下所示：

$$W^{[1]} = \begin{bmatrix} -w_1^{[1]\top} - \\ -w_2^{[1]\top} - \\ \vdots \\ -w_m^{[1]\top} - \end{bmatrix} \in \mathbb{R}^{m \times d} \quad (7.17)$$

现在根据矩阵向量乘法的定义，可以将  $z = [z_1, \dots, z_m]^\top \in \mathbb{R}^m$  写为

$$\underbrace{\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}}_{z \in \mathbb{R}^{m \times 1}} = \underbrace{\begin{bmatrix} -w_1^{[1]\top} - \\ -w_2^{[1]\top} - \\ \vdots \\ -w_m^{[1]\top} - \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m \times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x \in \mathbb{R}^{d \times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{m \times 1}} \quad (7.18)$$

或者简明地写成

$$z = W^{[1]}x + b^{[1]} \quad (7.19)$$

再次强调，在本书中，根据先前建立的约定， $\mathbb{R}^d$  中的向量自动视为列向量，并且可以看作是一个  $d \times 1$  维矩阵。（注意，这与 numpy 不同，在 numpy 中，向量在广播中被视为行向量。）

根据  $z \in \mathbb{R}^m$  计算激活  $a \in \mathbb{R}^m$  涉及 ReLU 函数的逐元素应用，这可以高效地并行计算。重载 ReLU 以进行逐元素应用（也就是说，对于向量  $t \in \mathbb{R}^d$ ， $\text{ReLU}(t)$  是一个向量，使得  $\text{ReLU}(t)_i = \text{ReLU}(t_i)$ ），所以，我们有

$$a = \text{ReLU}(z) \quad (7.20)$$

类似地定义  $W^{[2]} = [w^{[2]}]^\top \in \mathbb{R}^{1 \times m}$ 。那么，公式 (7.16) 中的模型可以概括为

$$\begin{aligned} a &= \text{ReLU}(W^{[1]}x + b^{[1]}) \\ \bar{h}_\theta(x) &= W^{[2]}a + b^{[2]} \end{aligned} \quad (7.21)$$

其中  $\theta$  由  $W^{[1]}, W^{[2]}$  (通常称为权重矩阵) 和  $b^{[1]}, b^{[2]}$  (称为偏置) 组成。 $W^{[1]}, b^{[1]}$  的集合称为第一层,  $W^{[2]}, b^{[2]}$  称为第二层。激活  $a$  称为隐藏层。两层神经网络也称为单隐藏层神经网络。

## 多层全连接神经网络

有了这种简洁的表示法, 就可以堆叠更多层以获得更深的全连接神经网络。令  $r$  为层数 (权重矩阵数)。令  $W^{[1]}, \dots, W^{[r]}, b^{[1]}, \dots, b^{[r]}$  为所有层的权重矩阵和偏置。那么多层神经网络可以写成

$$\begin{aligned} a^{[1]} &= \text{ReLU}(W^{[1]}x + b^{[1]}) \\ a^{[2]} &= \text{ReLU}(W^{[2]}a^{[1]} + b^{[2]}) \\ &\dots \\ a^{[r-1]} &= \text{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]}) \\ \bar{h}_\theta(x) &= W^{[r]}a^{[r-1]} + b^{[r]} \end{aligned} \tag{7.22}$$

注意到, 为了使上述方程有意义, 权重矩阵和偏置需要具有兼容的维度。如果  $a^{[k]}$  的维度为  $m_k$ , 则权重矩阵  $W^{[k]}$  的维度应为  $m_k \times m_{k-1}$ , 偏置  $b^{[k]} \in \mathbb{R}^{m_k}$ 。此外,  $W^{[1]} \in \mathbb{R}^{m_1 \times d}$  且  $W^{[r]} \in \mathbb{R}^{1 \times m_{r-1}}$ 。

网络中的神经元总数为  $m_1 + \dots + m_r$ , 该网络中的参数总数为  $(d+1)m_1 + (m_1+1)m_2 + \dots + (m_{r-1}+1)m_r$ 。

有时为了符号一致性, 也记  $a^{[0]} = x$  且  $a^{[r]} = \bar{h}_\theta(x)$ 。那么有简单的递归关系

$$a^{[k]} = \text{ReLU}(W^{[k]}a^{[k-1]} + b^{[k]}), \forall k = 1, \dots, r-1 \tag{7.23}$$

注意, 如果在公式 (7.22) 中有一个额外的 ReLU, 则对于  $k=r$  这也成立, 但人们通常喜欢使最后一层为线性层 (即没有 ReLU), 这样可以得到负输出, 并且更容易将最后一层解释为线性模型。(关于可解释性的更多内容请参见本节的“与核方法的联系”段落。)

## 其他激活函数

激活函数 ReLU 可以被许多其他将  $\mathbb{R}$  映射到  $\mathbb{R}$  的非线性函数  $\sigma(\cdot)$  替换，例如

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (\text{sigmoid}) \quad (7.24)$$

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (\tanh) \quad (7.25)$$

$$\sigma(z) = \max\{z, \gamma z\}, \gamma \in (0, 1) \quad (\text{leaky ReLU}) \quad (7.26)$$

$$\sigma(z) = \frac{z}{2} \left[ 1 + \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right) \right] \quad (\text{GELU}) \quad (7.27)$$

$$\sigma(z) = \frac{1}{\beta} \log(1 + \exp(\beta z)), \beta > 0 \quad (\text{Softplus}) \quad (7.28)$$

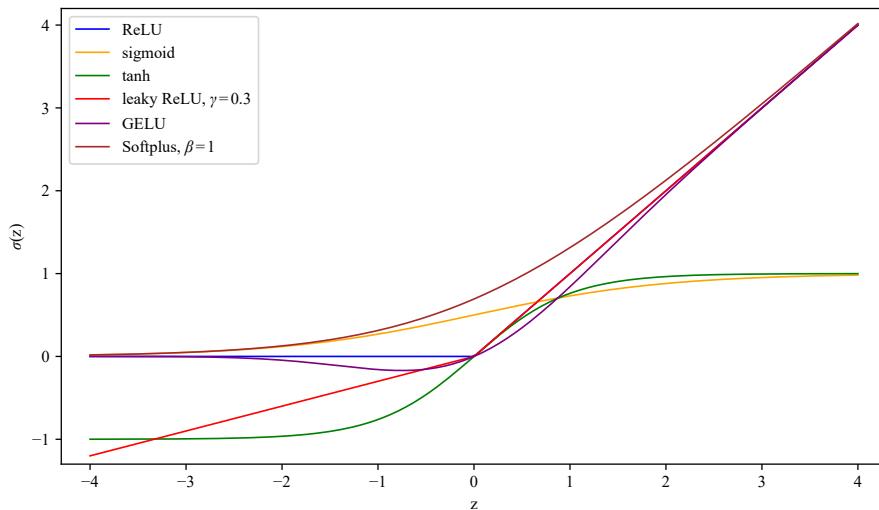


图 7.3: 深度学习中的激活函数

激活函数在图 7.3 中绘制。Sigmoid 和 tanh 现在使用得越来越少，部分原因是它们在两侧都有界，并且当  $z$  趋于正负无穷时，它们的梯度会消失（而所有其他激活函数在输入趋于正无穷时仍然有梯度）。Softplus 可以看作是 ReLU 的平滑版本，它具有适当的二阶导数，不过在实践中也不常用。GELU 和 leaky ReLU 都是 ReLU 的变体，但即使输入为负，它们也具有非零梯度。GELU（或其变体）用于 NLP 模型，例如 BERT 和 GPT（这将在第 14 章讨论）。

## 为什么不使用恒等函数作为 $\sigma(z)$ ?

也即，为什么不使用  $\sigma(z) = z$ ? 为了论证方便，假设  $b^{[1]}$  和  $b^{[2]}$  都为零。假设  $\sigma(z) = z$ ，那么对于两层神经网络，有

$$\bar{h}_\theta(x) = W^{[2]}a^{[1]} \quad (7.29)$$

$$= W^{[2]}\sigma(z^{[1]}) \quad \text{根据定义} \quad (7.30)$$

$$= W^{[2]}z^{[1]} \quad \text{因为 } \sigma(z) = z \quad (7.31)$$

$$= W^{[2]}W^{[1]}x \quad \text{根据公式 (7.18)} \quad (7.32)$$

$$= \tilde{W}x \quad \text{其中 } \tilde{W} = W^{[2]}W^{[1]} \quad (7.33)$$

注意  $W^{[2]}W^{[1]}$  如何合并成  $\tilde{W}$ 。

这是因为将一个线性函数应用于另一个线性函数会得到一个关于原始输入的线性函数(即，可以构造一个  $\tilde{W}$  使得  $\tilde{W}x = W^{[2]}W^{[1]}x$ )。这会损失神经网络的许多表达能力，因为通常要预测的输出与输入之间存在非线性关系。如果没有非线性激活函数，神经网络将只会进行简单的线性回归。

## 与核方法的联系

在之前的课程中，讨论了特征映射的概念。回忆一下，特征映射的主要动机是通过  $\theta^\top \phi(x)$  来表示输入  $x$  的非线性函数，其中  $\theta$  是参数， $\phi(x)$  是特征映射，是一个手工设计的关于原始输入  $x$  的非线性函数。学习算法的性能很大程度上取决于特征映射  $\phi(x)$  的选择。通常人们会利用领域知识来设计特征映射  $\phi(x)$  以便适合特定的应用。选择特征映射的过程通常被称为**特征工程 (feature engineering)**。

可以将深度学习看作是一种自动学习正确特征映射(有时也称为“表示”)的方法，如下所示。假设用  $\beta$  表示全连接神经网络(除了最后一层)的参数集合(公式(7.22))。然后可以将  $a^{[r-1]}$  抽象为输入  $x$  和参数  $\beta$  的函数： $\beta : a^{[r-1]} = \phi_\beta(x)$ 。现在可以将模型写成

$$\bar{h}_\theta(x) = W^{[r]}\phi_\beta(x) + b^{[r]} \quad (7.34)$$

当  $\beta$  固定时， $\phi_\beta(\cdot)$  可以看作是特征映射，因此  $\bar{h}_\theta(x)$  是关于特征  $\phi_\beta(x)$  的线性模型。然而，训练神经网络时，参数  $\beta$  和参数  $W^{[r]}, b^{[r]}$  都会被优化，因此我们不仅在特征空间中学习线性模型，还在学习一个好的特征映射  $\phi_\beta(\cdot)$  本身，以便在特征映射之上使用线性模型能够准确地进行预测。因此，深度学习倾向于较少依赖特定应用的领域知识，并且通常需要较少的特征工程。倒数第二层  $a^{[r]}$  通常被(非正式地)称为深度学习背景下的学习特征或表示。

在房价预测的例子中，全连接神经网络不需要指定中间量，例如“家庭规模”，并且可以自动发现倒数第二层（即激活  $a^{[r-1]}$ ）中的一些有用特征，并使用它们来线性预测房价。从一个数据集获得的特征映射或表示（即函数  $\phi_\beta(\cdot)$ ）也可以用于其他数据集，这表明它们包含关于数据的基本信息。然而，通常神经网络会发现复杂的特征，这些特征对于预测输出非常有用，但对于人类来说可能难以理解或解释。这就是为什么有些人将神经网络称为黑箱 (*black box*)，因为很难理解它发现的特征。

### 7.3 现代神经网络的模块

第 7.2 节公式 (7.22) 中介绍的多层神经网络现在通常被称为多层次感知机 (MLP)。现代神经网络在实践中通常更复杂，由多个构建块或多层构建块组成。在本节中，将介绍一些其他的构建块并讨论可能的组合方式。

首先，每个矩阵乘法可以看作是一个构建块。考虑一个带有参数  $(W, b)$  的矩阵乘法运算，其中  $W$  是权重矩阵， $b$  是偏置向量，作用于输入  $z$ ，

$$\text{MM}_{W,b}(z) = Wz + b. \quad (7.35)$$

注意，这里隐含地假设所有维度都兼容。当在上下文中清晰或对讨论不重要时，也会省略 MM 的下标用以简化。

然后，MLP 可以写成多个矩阵乘法模块和非线性激活模块（也可以看作是构建块）的组合：

$$\text{MLP}(x) = \text{MM}_{W^{[r]}, b^{[r]}}(\sigma(\text{MM}_{W^{[r-1]}, b^{[r-1]}}(\sigma(\cdots \text{MM}_{W^{[1]}, b^{[1]}}(x))))). \quad (7.36)$$

或者，当省略表示参数的下标时，可以写成

$$\text{MLP}(x) = \text{MM}(\sigma(\text{MM}(\sigma(\cdots \text{MM}(x))))). \quad (7.37)$$

注意，在本讲义中，默认情况下，所有模块都有不同的参数集，并且参数的维度是有意义的。

较大的模块也可以通过较小的模块定义，例如，一个激活层  $\sigma$  和一个矩阵乘法层 MM 经常组合在一起，在许多论文中被称为“层”。人们通常通过在图中指示这些模块之间的依赖关系来绘制架构。例如，参见图 7.4 左侧的 MLP 示意图。

### 残差连接

一种非常有影响力的用于视觉应用的神经网络架构是 ResNet，它使用了残差连接，这些连接现在几乎用于所有大规模深度学习架构中。使用上面介绍的符

号，一个非常简化的残差块可以定义为

$$\text{Res}(z) = z + \sigma(\text{MM}(\sigma(\text{MM}(z)))) \quad (7.38)$$

一个简化版 ResNet 是许多残差块的组合，后接一个矩阵乘法，

$$\text{ResNet-S}(x) = \text{MM}(\text{Res}(\dots \text{Res}(x))) \quad (7.39)$$

这些模块的依赖关系也绘制在图 7.4 右侧。

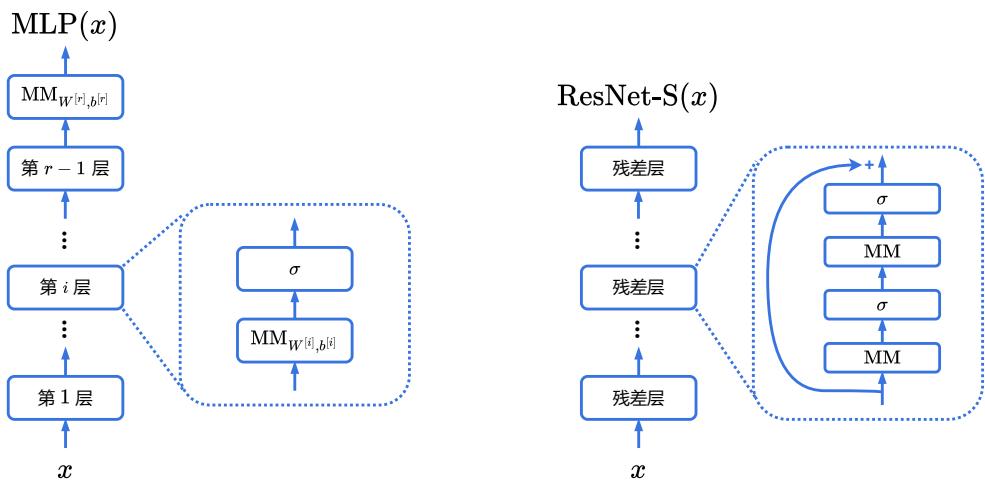


图 7.4: 网络架构示意图。左图:  $r$  层的 MLP。右图: 残差网络。

注意，ResNet-S 与经典论文 [He et al. 2016] 中介绍的 ResNet 架构仍不完全相同，因为 ResNet 使用卷积层而不是普通的矩阵乘法，并在卷积和激活之间添加了批量归一化。下面将介绍卷积层和批量归一化的一些变体。ResNet-S 和层归一化是 Transformer 架构的一部分，它们在现代大型语言模型中被广泛使用。

## 层归一化

层归一化，本文中记为 LN，是一个将向量  $z \in \mathbb{R}^m$  映射到规范化的向量  $\text{LN}(z) \in \mathbb{R}^m$  的模块。它通常在非线性激活之后使用。

首先定义层归一化的一个子模块，记为 LN-S。

$$\text{LN-S}(z) = \begin{bmatrix} \frac{z_1 - \hat{\mu}}{\hat{\sigma}} \\ \frac{z_2 - \hat{\mu}}{\hat{\sigma}} \\ \vdots \\ \frac{z_m - \hat{\mu}}{\hat{\sigma}} \end{bmatrix}, \quad (7.40)$$

其中  $\hat{\mu} = \frac{\sum_{i=1}^m z_i}{m}$  是向量  $z$  的实际均值,  $\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^m (z_i - \hat{\mu})^2}{m}}$  是  $z$  中元素的实际标准差。<sup>4</sup> 直观上看,  $\text{LN-S}(z)$  是一个经过归一化处理的向量, 其实际均值为零, 实际标准差为 1。

通常零均值和标准差 1 并非最理想的归一化方案, 因此层归一化引入了可学习的标量参数  $\beta$  和  $\gamma$  作为期望的均值和标准差, 并使用仿射变换将  $\text{LN-S}(z)$  的输出转换为均值为  $\beta$ 、标准差为  $\gamma$  的向量。

$$\text{LN}(z) = \beta + \gamma \cdot \text{LN-S}(z) = \begin{bmatrix} \beta + \gamma \left( \frac{z_1 - \hat{\mu}}{\hat{\sigma}} \right) \\ \beta + \gamma \left( \frac{z_2 - \hat{\mu}}{\hat{\sigma}} \right) \\ \vdots \\ \beta + \gamma \left( \frac{z_m - \hat{\mu}}{\hat{\sigma}} \right) \end{bmatrix}. \quad (7.41)$$

这里, 第一个  $\beta$  技术上应该解释为一个所有元素都是标量  $\beta$  的向量。还需要注意,  $\hat{\mu}$  和  $\hat{\sigma}$  也是  $z$  的函数, 在计算层归一化的导数时不应将其视为常数。此外,  $\beta$  和  $\gamma$  是可学习参数, 因此层归一化是一个带参数的模块 (与不带任何参数的激活层不同)。

**尺度不变性:** 层归一化一个重要的性质是它使得模型对参数的缩放具有不变性, 具体如下。假设考虑将  $\text{LN}$  与  $\text{MM}_{W,b}(z)$  组合, 得到一个子网络  $\text{LN}(\text{MM}_{W,b}(z))$ 。那么, 当  $\text{MM}_{W,b}$  中的参数被缩放时, 这个子网络的输出不会改变:

$$\text{LN}(\text{MM}_{\alpha W, \alpha b}(z)) = \text{LN}(\text{MM}_{W,b}(z)), \forall \alpha > 0. \quad (7.42)$$

为了说明这一点, 首先知道  $\text{LN-S}(\cdot)$  是尺度不变的:

$$\text{LN-S}(\alpha z) = \begin{bmatrix} \frac{\alpha z_1 - \alpha \hat{\mu}}{\alpha \hat{\sigma}} \\ \frac{\alpha z_2 - \alpha \hat{\mu}}{\alpha \hat{\sigma}} \\ \vdots \\ \frac{\alpha z_m - \alpha \hat{\mu}}{\alpha \hat{\sigma}} \end{bmatrix} = \begin{bmatrix} \frac{z_1 - \hat{\mu}}{\hat{\sigma}} \\ \frac{z_2 - \hat{\mu}}{\hat{\sigma}} \\ \vdots \\ \frac{z_m - \hat{\mu}}{\hat{\sigma}} \end{bmatrix} = \text{LN-S}(z). \quad (7.43)$$

然后有

$$\text{LN}(\text{MM}_{\alpha W, \alpha b}(z)) = \beta + \gamma \text{LN-S}(\text{MM}_{\alpha W, \alpha b}(z)) \quad (7.44)$$

$$= \beta + \gamma \text{LN-S}(\alpha \text{MM}_{W,b}(z)) \quad (7.45)$$

$$= \beta + \gamma \text{LN-S}(\text{MM}_{W,b}(z)) \quad (7.46)$$

$$= \text{LN}(\text{MM}_{W,b}(z)). \quad (7.47)$$

---

<sup>4</sup>注意, 这里的实际标准差是除以  $m$  而不是  $m - 1$ , 因为所感兴趣的是使  $\text{LN-S}(z)$  的输出的平方和等于 1 (与统计学中估计标准差不同)。

由于这一性质，大多数用于大规模计算机视觉和语言应用的现代深度学习架构都具有以下尺度不变性，即对于除最后一层权重  $W_{\text{last}}$  以外的所有权重  $W$ ，网络  $f$  具有  $f_{W_{\text{last}}, \alpha W}(x) = f_{W_{\text{last}}, W}(x)$  对于任意  $\alpha > 0$  成立。这里，最后一层的权重是特殊的，因为在最后一层权重之后通常没有层归一化或批量归一化。

**其他归一化层：**还有几种其他归一化层，旨在将神经网络的中间层归一化到更固定和可控的尺度，例如批量归一化 [?] 和组归一化 [?]。批量归一化和组归一化更常用于计算机视觉应用，而层归一化更常用于语言应用。

## 卷积层

卷积神经网络是由卷积层（以及许多其他模块）组成的神经网络，特别适用于计算机视觉应用。为了简化说明，本文重点介绍 1-D 卷积，并在本小节末尾非正式地简要提及 2-D 卷积。（2-D 卷积更适合于具有两个维度的图像。1-D 卷积也用于自然语言处理。）

首先介绍一个简化版本的 1-D 卷积层，记为  $\text{Conv1D-S}(\cdot)$ ，它是一种具有特殊结构的矩阵乘法层。 $\text{Conv1D-S}$  的参数是一个滤波器向量  $w \in \mathbb{R}^k$ ，其中  $k$  称为滤波器大小（通常  $k \ll m$ ），以及一个标量偏置  $b$ 。滤波器有时也称为核（但与核方法中的核无关）。为简单起见，假设  $k = 2\ell + 1$  是一个奇数。首先在输入向量  $z$  的两端填充零，即  $z_{1-\ell} = z_{1-\ell+1} = \dots = z_0 = 0$  和  $z_{m+1} = z_{m+2} = \dots = z_{m+\ell} = 0$ ，并将  $z$  看作一个  $(m + 2\ell)$  维向量。 $\text{Conv1D-S}$  输出一个维度为  $\mathbb{R}^m$  的向量，其中每个输出维度是  $z$  中子集元素的线性组合，系数来自  $w$ ，

$$\text{Conv1D-S}(z)_i = w_1 z_{i-\ell} + w_2 z_{i-\ell+1} + \dots + w_{2\ell+1} z_{i+\ell} = \sum_{j=1}^{2\ell+1} w_j z_{i-\ell+(j-1)}. \quad (7.48)$$

因此，可以将  $\text{Conv1D-S}$  视为具有共享参数的矩阵乘法： $\text{Conv1D-S}(z) = Qz$ ，其中

$$Q = \begin{bmatrix} w_{\ell+1} & \cdots & w_{2\ell+1} \\ \vdots & \ddots & \ddots & w_{2\ell+1} \\ w_1 & \ddots & \ddots & \ddots & \ddots \\ w_1 & \ddots & \ddots & \ddots & w_{2\ell+1} \\ \ddots & \ddots & \ddots & \ddots & \vdots \\ w_1 & \cdots & w_{\ell+1} \end{bmatrix}. \quad (7.49)$$

注意， $Q_{i,j} = Q_{i-1,j-1}$  对于所有  $i, j \in \{2, \dots, m\}$  都成立，因此卷积是一种带有参数共享的矩阵乘法。还注意到，计算卷积只需要  $O(km)$  的时间，而计算一般

的矩阵乘法需要  $O(m^2)$  的时间。卷积有  $k$  个参数，而一般的矩阵乘法有  $m^2$  个参数。因此，卷积应该比一般的矩阵乘法效率高得多（只要施加的附加结构不损害模型的灵活性以适应数据）。

还注意到，在实践中存在许多卷积层的变体，与这里定义的有所不同，例如，填充零的方式或有时卷积层输出的维度可能与输入的维度不同。为了简化，这里省略了一些这些细节。

实践中使用的卷积层也有许多“通道”，上面的简化版本对应于 1 通道版本。形式上，Conv1D 接受  $C$  个向量  $z_1, \dots, z_C \in \mathbb{R}^m$  作为输入，其中  $C$  称为通道数。换句话说，更一般的版本 Conv1D 接受一个矩阵作为输入，它是  $z_1, \dots, z_C$  的拼接，维度为  $m \times C$ 。它可以输出  $C'$  个维度为  $m$  的向量，记为  $\text{Conv1D}(z)_1, \dots, \text{Conv1D}(z)_{C'}$ ，其中  $C'$  称为输出通道，或等效地一个维度为  $m \times C'$  的矩阵。每个输出是应用于不同通道的简化卷积之和。

$$\forall i \in [C'], \text{Conv1D}(z)_i = \sum_{j=1}^C \text{Conv1D-S}_{i,j}(z_j). \quad (7.50)$$

注意，每个  $\text{Conv1D-S}_{i,j}$  都是具有不同参数的模块，因此总参数数量是  $k \times C \times C'$  ( $\text{Conv1D-S}$  的参数数量  $\times \text{Conv1D-S}_{i,j}$  的数量)  $= kCC'$ 。相比之下，从  $\mathbb{R}^{m \times C}$  到  $\mathbb{R}^{m \times C'}$  的一般线性映射具有  $m^2CC'$  个参数。卷积的参数也可以表示为维度为  $k \times C \times C'$  的三维张量。

*2-D 卷积 (简述):* 单通道的 2-D 卷积，记为 Conv2D-S，类似于 Conv1D-S，但接受一个 2 维输入  $z \in \mathbb{R}^{m \times m}$  并应用一个  $k \times k$  大小的滤波器，输出  $\text{Conv2D-S}(z) \in \mathbb{R}^{m \times m}$ 。完整的 2-D 卷积层，记为 Conv2D，接受一系列矩阵  $z_1, \dots, z_C \in \mathbb{R}^{m \times m}$  作为输入，或者等效地一个 3-D 张量  $z = (z_1, \dots, z_C) \in \mathbb{R}^{m \times m \times C}$ ，并输出一系列矩阵  $\text{Conv2D}(z)_1, \dots, \text{Conv2D}(z)_{C'} \in \mathbb{R}^{m \times m}$ ，也可以看作一个维度为  $\mathbb{R}^{m \times m \times C'}$  的 3D 张量。每个输出通道是应用于所有输入通道的 Conv2D-S 层结果的总和。

$$\forall i \in [C'], \text{Conv2D}(z)_i = \sum_{j=1}^C \text{Conv2D-S}_{i,j}(z_j). \quad (7.51)$$

因为有  $CC'$  个 Conv2D-S 模块，并且每个 Conv2D-S 模块有  $k^2$  个参数，所以总参数数量是  $CC'k^2$ 。其参数也可以看作一个维度为  $C \times C' \times k \times k$  的四维张量。

## 7.4 反向传播

本节将介绍反向传播，或称自动微分，它能高效地计算损失函数的梯度  $\nabla J(\theta)$ 。首先，从一个非正式定理开始，该定理指出，只要一个实值函数 (*real-*

*valued function*)  $f$  可以通过一个可微网络或电路高效地计算/评估，那么它的梯度也可以在相似的时间内高效地计算。然后，将展示如何在神经网络中具体实现这一点。

由于通用定理的严格性并非本节的主要重点，将使用非正式定义来介绍术语。可微电路或可微网络是指一系列可微算术运算（加法、减法、乘法、除法等）和基本可微函数（ReLU、 $\exp$ 、 $\log$ 、 $\sin$ 、 $\cos$  等）的组合。电路的大小定义为这些运算和基本函数的总数。假设每个运算和函数及其导数或偏导数都可以在  $O(1)$  时间内计算。

#### 定理 7.4.1. [反向传播，或称自动微分的非正式表述]

假设有一个大小为  $N$  的可微电路计算一个实值函数  $f: \mathbb{R}^\ell \rightarrow \mathbb{R}$ 。那么，梯度  $\nabla f$  可以通过一个大小为  $O(N)$  的电路在  $O(N)$  的时间内计算。<sup>5</sup>

注意到，对于第  $j$  个样本的损失函数  $J^{(j)}(\theta)$  可以通过包含加法、减法、乘法和非线性激活等运算和函数的序列计算。因此，定理表明，应该能够在与计算  $J^{(j)}(\theta)$  本身相似的时间内计算  $\nabla J^{(j)}(\theta)$ 。这不仅适用于第 7.2 节介绍的全连接神经网络，也适用于许多其他使用更高级模块的神经网络。

注意到，自动微分或反向传播已经集成到所有深度学习库中，例如 tensorflow 和 pytorch，因此在实践中，大多数情况下研究人员无需编写自己的反向传播算法。然而，理解其原理对于深入理解深度学习的工作原理非常有帮助。

本节的其余部分组织如下。在第 7.4.1 节中，将从一个新的角度回顾基本链式法则，这对于理解反向传播特别有用。第 7.4.2 节将介绍反向传播的通用策略。第 7.4.2 节将讨论如何计算神经网络中使用的基本模块的所谓反向函数，第 7.4.4 节将把所有内容组合起来，得到一个用于 MLP 的具体反向传播算法。

### 7.4.1 偏导数初步

假设标量变量  $J$  依赖于某些变量  $z$ （可以是标量、矩阵或高阶张量），记  $\frac{\partial J}{\partial z}$  为  $J$  关于变量  $z$  的偏导数。这里的惯例是  $\frac{\partial J}{\partial z}$  与  $z$  本身具有完全相同的维度。例如，如果  $z \in \mathbb{R}^{m \times n}$ ，则  $\frac{\partial J}{\partial z} \in \mathbb{R}^{m \times n}$ ，并且  $\frac{\partial J}{\partial z}$  的  $(i, j)$  元素等于  $\frac{\partial J}{\partial z_{ij}}$ 。

**备注 7.4.1.** 当  $J$  和  $z$  都不是标量时， $J$  关于  $z$  的偏导数变成矩阵或张量，并且符号变得有些棘手。除了处理数学或符号上的挑战之外，这些多元函数的偏导数不仅计算和存储成本高昂，而且实际上也很少显式构造这些偏导数。作者的经验

---

<sup>5</sup> 注意到，如果函数  $f$  的输出不依赖于某些输入的分量，则默认将关于这些分量的梯度设为零。在本节的计算方案中，将梯度设为零不计入总运行时间。因此，当  $N \leq \ell$  时，可以在  $O(N)$  时间内计算梯度，这可能甚至小于  $\ell$ 。

表明，只考虑标量函数对向量、矩阵或张量的导数通常更有效。故在本讲义中，不会讨论多元函数的导数。

## 链式法则

下面回顾微积分中的链式法则，但我们的视角和符号更侧重于自动微分。

考虑一个标量变量  $J$ ，它由函数  $f$  和  $g$  在某个变量  $z$  上的复合得到，

$$\begin{aligned} z &\in \mathbb{R}^m \\ u &= g(z) \in \mathbb{R}^n \\ J &= f(u) \in \mathbb{R}. \end{aligned} \tag{7.52}$$

下面的推导可以很容易地扩展到  $z$  和  $u$  是矩阵或张量的情况；但需要强调的是，最终变量  $J$  是一个标量（参见备注 7.4.1）。令  $u = (u_1, \dots, u_n)$ ，并令  $g(z) = (g_1(z), \dots, g_n(z))$ 。那么，根据标准的链式法则，有

$$\forall i \in \{1, \dots, m\}, \quad \frac{\partial J}{\partial z_i} = \sum_{j=1}^n \frac{\partial J}{\partial u_j} \cdot \frac{\partial g_j}{\partial z_i}. \tag{7.53}$$

或者，当  $z$  和  $u$  都是向量时，采用向量化符号：

$$\frac{\partial J}{\partial z} = \begin{bmatrix} \frac{\partial g_1}{\partial z_1} & \cdots & \frac{\partial g_n}{\partial z_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_1}{\partial z_m} & \cdots & \frac{\partial g_n}{\partial z_m} \end{bmatrix} \cdot \frac{\partial J}{\partial u}. \tag{7.54}$$

换句话说，反向函数总是从  $\frac{\partial J}{\partial u}$  到  $\frac{\partial J}{\partial z}$  的线性映射，但请注意，映射本身可以以复杂的方式依赖于  $z$ 。公式 (7.54) 右侧的矩阵实际上是函数  $g$  的雅可比矩阵的转置。然而，不会深入讨论雅可比矩阵的应用，以避免复杂性。部分原因在于，当  $z$  是矩阵（或张量）时，要写出类似公式 (7.54) 的形式，需要将  $z$  展平为向量或引入张量-矩阵乘法的额外符号。从这个意义上说，公式 (7.53) 在所有情况下都更方便有效。例如，当  $z \in \mathbb{R}^{r \times s}$  是一个矩阵时，可以很容易地重写公式 (7.53) 得到

$$\forall i, k, \quad \frac{\partial J}{\partial z_{ik}} = \sum_{j=1}^n \frac{\partial J}{\partial u_j} \cdot \frac{\partial g_j}{\partial z_{ik}}, \tag{7.55}$$

这将在第 7.4.3 节的一些推导中用到。

链式法则的关键解释：可以将上面的公式（公式 (7.53) 或 (7.54)）看作是从  $\frac{\partial J}{\partial u}$  计算  $\frac{\partial J}{\partial z}$  的一种方法。考虑以下抽象问题：假设  $J$  通过公式 (7.52) 中定义的  $u$  依

赖于  $z$ 。然而，假设函数  $f$  未知或函数  $f$  很复杂，但已知  $\frac{\partial J}{\partial u}$  的值。那么，公式 (7.54) 提供了一种从  $\frac{\partial J}{\partial u}$  计算  $\frac{\partial J}{\partial z}$  的方法。

$$\frac{\partial J}{\partial u} \xrightarrow[\text{只需要关于 } g(\cdot) \text{ 和 } z \text{ 的信息}]{\text{利用链式法则和公式 (7.54)}} \frac{\partial J}{\partial z}. \quad (7.56)$$

此外，此公式仅涉及关于  $g$  的知识（更准确地说， $\frac{\partial g_j}{\partial z_i}$ ）。将反复利用这一点来处理  $g$  是复杂网络  $f$  的构建模块的情况。

根据经验，可以将公式 (7.53) 或 (7.54) 中的映射模块视为一个黑盒，并且定义一个数学符号也便于后续讨论。<sup>6</sup> 使用  $\mathcal{B}[g, z]$  定义将  $\frac{\partial J}{\partial u}$  映射到  $\frac{\partial J}{\partial z}$  的函数，并记作

$$\frac{\partial J}{\partial z} = \mathcal{B}[g, z] \left( \frac{\partial J}{\partial u} \right). \quad (7.57)$$

称  $\mathcal{B}[g, z]$  为模块  $g$  的**反向函数 (backward function)**。注意，当  $z$  固定时， $\mathcal{B}[g, z]$  仅是从  $\mathbb{R}^n$  到  $\mathbb{R}^m$  的线性映射。使用公式 (7.53)，有

$$(\mathcal{B}[g, z](v))_i = \sum_{j=1}^m \frac{\partial g_j}{\partial z_i} \cdot v_j. \quad (7.58)$$

或者采用向量化符号，使用公式 (7.54)，有

$$\mathcal{B}[g, z](v) = \begin{bmatrix} \frac{\partial g_1}{\partial z_1} & \cdots & \frac{\partial g_n}{\partial z_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_1}{\partial z_m} & \cdots & \frac{\partial g_n}{\partial z_m} \end{bmatrix} \cdot v. \quad (7.59)$$

因此  $\mathcal{B}[g, z]$  可以看作一个矩阵。然而，实际上， $z$  将不断变化，因此当  $g$  固定时，反向映射必须针对不同的  $z$  重新计算。因此，从经验上看，反向函数  $\mathcal{B}[g, z](v)$  通常被视为一个函数，它接受  $z$  ( $g$  的输入) 和  $v$  ( $J$  对某个变量的梯度，该变量被认为是  $g$  的输出) 作为输入，并输出一个向量，该向量被认为是  $J$  对  $z$  的梯度。

#### 7.4.2 反向传播的通用策略

本节讨论自动微分的通用策略，以建立高层次的理解。然后，将把这种方法实例化到具体的神经网络中。采用的观点是，神经网络是由小的构建块组成的复杂组合，例如第 7.3 节中定义的 MM、 $\sigma$ 、Conv2D、LN 等。注意，损失函数（例

---

<sup>6</sup>该函数也是 pytorch 中模块的 `.backward()` 方法。

如，均方误差损失或交叉熵损失）也可以抽象地视为附加模块。因此，可以将损失函数  $J$ （针对单个样本  $(x, y)$ ）抽象地写成许多模块的组合：<sup>7</sup>

$$J = M_k(M_{k-1}(\cdots M_1(x))). \quad (7.60)$$

例如，对于具有 MLP  $\bar{h}_\theta(x)$ （在公式 (7.36) 和 (7.37) 中定义）的二分类问题，损失函数可以写成公式 (7.60) 的形式，其中  $M_1 = \text{MM}_{W^{[1]}, b^{[1]}}$ ， $M_2 = \sigma$ ， $M_3 = \text{MM}_{W^{[2]}, b^{[2]}}$ ，…，以及  $M_{k-1} = \text{MM}_{W^{[r]}, b^{[r]}}$ ， $M_k = \ell_{\text{logistic}}$ 。

从这个例子可以看出，有些模块涉及参数，而有些模块可能只涉及固定的操作集。为了通用性，假设每个  $M_i$  都涉及一组参数  $\theta^{[i]}$ ，尽管当  $M_i$  是像非线性激活这样的固定操作时， $\theta^{[i]}$  可能是一个空集。之后将更详细地讨论模块化的粒度，但目前假设所有模块  $M_i$  都足够简单。

引入用于公式 (7.60) 中计算的中间变量：

$$\begin{aligned} u^{[0]} &= x \\ u^{[1]} &= M_1(u^{[0]}) \\ u^{[2]} &= M_2(u^{[1]}) \\ &\vdots \\ J &= u^{[k]} = M_k(u^{[k-1]}). \end{aligned} \quad (\text{F})$$

反向传播由两个过程组成：前向传播和反向传播。在前向传播中，算法根据定义 (F) 按顺序计算  $u^{[1]}, \dots, u^{[k]}$ ，并将所有中间变量  $u^{[i]}$  保存在内存中。

在反向传播中，首先按反向顺序计算  $J$  对中间变量的导数，即  $\frac{\partial J}{\partial u^{[k]}}, \dots, \frac{\partial J}{\partial u^{[1]}}$ ，然后从  $\frac{\partial J}{\partial u^{[i]}}$  和  $u^{[i-1]}$  计算参数  $\theta^{[i]}$  的导数  $\frac{\partial J}{\partial \theta^{[i]}}$ 。这两类计算可以相互交织，因为  $\frac{\partial J}{\partial \theta^{[i]}}$  仅依赖于  $\frac{\partial J}{\partial u^{[i]}}$  和  $u^{[i-1]}$ ，而不依赖于任何  $k < i$  的  $\frac{\partial J}{\partial u^{[k]}}$ 。

首先通过引用第 7.4.1 节关于链式法则的讨论来理解为什么  $\frac{\partial J}{\partial u^{[i-1]}}$  可以从  $\frac{\partial J}{\partial u^{[i]}}$  和  $u^{[i-1]}$  有效计算。通过设置  $u = u^{[i]}$  和  $z = u^{[i-1]}$ ，以及  $f(u) = M_k(M_{k-1}(\cdots M_{i+1}(u)))$ ，以及  $g(\cdot) = M_i(\cdot)$ ，来实例化讨论。注意， $f$  非常复杂，但不需要关于  $f$  的任何具体信息。那么，结论性公式 (7.56) 对应于

$$\frac{\partial J}{\partial u^{[i]}} \xrightarrow[\text{只需要关于 } M_i(\cdot) \text{ 和 } u^{[i-1]} \text{ 的信息}]{\text{链式法则}} \frac{\partial J}{\partial u^{[i-1]}}. \quad (7.61)$$

更准确地说，根据公式 (7.57)，可以写成

$$\frac{\partial J}{\partial u^{[i-1]}} = \mathcal{B}[M_i, u^{[i-1]}] \left( \frac{\partial J}{\partial u^{[i]}} \right). \quad (\text{B1})$$

---

<sup>7</sup>严格来说，应该写成  $J = M_k(M_{k-1}(\cdots M_1(x)), y)$ 。然而，为了计算相对于参数的导数，将  $y$  视为常数，因此为了符号的简洁性，可以将其视为  $M_k$  的一部分。

将链式法则实例化为  $z = \theta^{[i]}$  和  $u = u^{[i]}$ , 也有

$$\frac{\partial J}{\partial \theta^{[i]}} = \mathcal{B}[M_i, \theta^{[i]}] \left( \frac{\partial J}{\partial u^{[i]}} \right). \quad (\text{B2})$$

有关算法的说明, 请参见图 7.5。

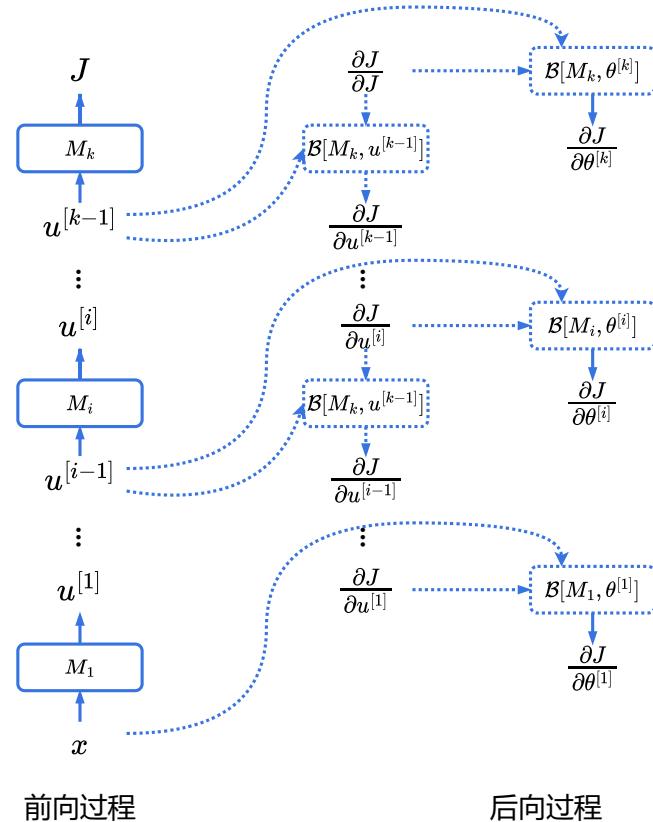


图 7.5: 反向传播

#### 备注 7.4.2. [计算效率和模块的粒度]

将复杂网络视为小型模块组合的主要根本目的是，小型模块往往具有可高效实现的后向函数。实际上，所有原子模块（如加法、乘法和 ReLU）的后向函数都可以像评估这些模块一样高效地计算（最多相差一个乘法常数因子）。利用这一事实，可以通过将神经网络视为许多原子操作的组合，并调用上面讨论的反向传播来证明定理 7.4.1。然而，在实践中，会更常使用矩阵乘法、层归一化这类模块来模块化网络。正如后文所述，这些操作的后向函数的朴素实现也具有与这些函数的评估相同的运行时间。

### 7.4.3 基本模块的后向函数

利用第 7.4.2 节的通用策略，只需计算网络中使用的所有模块  $M_i$  的后向函数即可。本节计算基本模块 MM、激活函数  $\sigma$  和损失函数的后向函数。

$MM$  的后向函数：假设  $MM_{W,b}(z) = Wz + b$  是一个矩阵乘法模块，其中  $z \in \mathbb{R}^m$  且  $W \in \mathbb{R}^{n \times m}$ 。那么，对于  $v \in \mathbb{R}^n$ ，使用公式 (7.59)，有

$$\mathcal{B}[MM, z](v) = \begin{bmatrix} \frac{\partial(Wz+b)_1}{\partial z_1} & \dots & \frac{\partial(Wz+b)_n}{\partial z_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial(Wz+b)_1}{\partial z_m} & \dots & \frac{\partial(Wz+b)_n}{\partial z_m} \end{bmatrix} v. \quad (7.62)$$

由于对于所有  $i \in [m], j \in [n]$ ， $\frac{\partial(Wz+b)_j}{\partial z_i} = \frac{\partial b_j + \sum_{k=1}^m W_{jk} z_k}{\partial z_i} = W_{ji}$ ，因此有

$$\mathcal{B}[MM, z](v) = W^\top v \in \mathbb{R}^m. \quad (7.63)$$

在上述推导中，将 MM 视为  $z$  的函数。如果将 MM 视为  $W$  和  $b$  的函数，那么也可以计算参数变量  $W$  和  $b$  的后向函数。使用公式 (7.59) 不太方便，因为变量  $W$  是一个矩阵，而公式 (7.59) 中的矩阵将是一个四阶张量，有书写难度。因此，使用 (7.58)：

$$(\mathcal{B}[MM, W](v))_{ij} = \sum_{k=1}^m \frac{\partial(Wz+b)_k}{\partial W_{ij}} \cdot v_k = \sum_{k=1}^m \frac{\partial \sum_{s=1}^m W_{ks} z_s}{\partial W_{ij}} \cdot v_k = v_i z_j. \quad (7.64)$$

在向量化表示中，有

$$\mathcal{B}[MM, W](v) = v z^\top \in \mathbb{R}^{n \times m}. \quad (7.65)$$

对于变量  $b$ ，使用公式 (7.59)，有

$$\mathcal{B}[MM, b](v) = \begin{bmatrix} \frac{\partial(Wz+b)_1}{\partial b_1} & \dots & \frac{\partial(Wz+b)_n}{\partial b_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial(Wz+b)_1}{\partial b_n} & \dots & \frac{\partial(Wz+b)_n}{\partial b_n} \end{bmatrix} v = v. \quad (7.66)$$

这里利用了当  $i \neq j$  时  $\frac{\partial(Wz+b)_i}{\partial b_j} = 0$ , 当  $i = j$  时  $\frac{\partial(Wz+b)_j}{\partial b_i} = 1$ 。

计算后向函数的计算效率是  $O(mn)$ , 与计算矩阵乘法结果的效率相同 (相差一个常数因子)。

激活函数的后向函数。假设  $M(z) = \sigma(z)$ , 其中  $\sigma$  是一个逐元素的激活函数, 且  $z \in \mathbb{R}^m$ 。那么, 使用公式 (7.59), 有

$$\mathcal{B}[\sigma, z](v) = \begin{bmatrix} \frac{\partial \sigma(z_1)}{\partial z_1} & \dots & \frac{\partial \sigma(z_m)}{\partial z_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \sigma(z_1)}{\partial z_m} & \dots & \frac{\partial \sigma(z_m)}{\partial z_m} \end{bmatrix} v \quad (7.67)$$

$$= \text{diag}(\sigma'(z_1), \dots, \sigma'(z_m))v \quad (7.68)$$

$$= \sigma'(z) \odot v \in \mathbb{R}^m. \quad (7.69)$$

这里利用了当  $j \neq i$  时  $\frac{\partial \sigma(z_j)}{\partial z_i} = 0$ ,  $\text{diag}(\lambda_1, \dots, \lambda_m)$  表示对角线上为  $\lambda_1, \dots, \lambda_m$  的对角矩阵,  $\odot$  表示两个同维度向量的逐元素乘积,  $\sigma'(\cdot)$  是激活函数  $\sigma$  的导数 (对向量逐元素应用)。关于计算效率, 注意到乍一看, 公式 (7.67) 似乎表明后向函数需要  $O(m^2)$  的时间, 但公式 (7.69) 表明它可以在  $O(m)$  的时间内实现 (这与评估函数的时间相同)。如果使用更小的模块, 即把向量到向量的非线性激活视为  $m$  个标量到标量的非线性激活, 那么公式 (7.67) 到 (7.69) 的简化可能性不应感到惊讶, 这时后向传播应该具有与前向传播相似的时间。

损失函数的后向函数。当模块  $M$  接受一个向量  $z$  并输出一个标量时, 根据公式 (7.59), 后向函数接受一个标量  $v$  并输出一个向量, 其分量为  $(\mathcal{B}[M, z](v))_i = \frac{\partial M}{\partial z_i} v$ 。因此, 在向量化表示中,  $\mathcal{B}[M, z](v) = \frac{\partial M}{\partial z} \cdot v$ 。

回想一下, 平方损失  $\ell_{\text{MSE}}(z, y) = \frac{1}{2}(z-y)^2$ 。因此,  $\mathcal{B}[\ell_{\text{MSE}}, z](v) = \frac{\partial \frac{1}{2}(z-y)^2}{\partial z} \cdot v = (z-y) \cdot v$ 。

对于逻辑损失, 根据公式 (2.6), 有

$$\mathcal{B}[\ell_{\text{logistic}}, t](v) = \frac{\partial \ell_{\text{logistic}}(t, y)}{\partial t} \cdot v = (1/(1 + \exp(-t)) - y) \cdot v. \quad (7.70)$$

对于交叉熵损失, 根据公式 (2.17), 有

$$\mathcal{B}[\ell_{\text{ce}}, t](v) = \frac{\partial \ell_{\text{ce}}(t, y)}{\partial t} \cdot v = (\phi - e_y) \cdot v, \quad (7.71)$$

其中  $\phi = \text{softmax}(t)$ 。

#### 7.4.4 MLP 的反向传播

给定评估 MLP 损失所需的每个模块的后向函数, 根据第 7.4.2 节的策略计算损失相对于隐藏激活和参数的梯度。考虑一个具有逻辑损失的  $r$  层 MLP。损

失函数可以通过一系列操作计算（即前向传播），

$$\begin{aligned}
z^{[1]} &= \text{MM}_{W^{[1]}, b^{[1]}}(x), \\
a^{[1]} &= \sigma(z^{[1]}), \\
z^{[2]} &= \text{MM}_{W^{[2]}, b^{[2]}}(a^{[1]}), \\
a^{[2]} &= \sigma(z^{[2]}), \\
&\vdots \\
z^{[r]} &= \text{MM}_{W^{[r]}, b^{[r]}}(a^{[r-1]}), \\
J &= \ell_{\text{logistic}}(z^{[r]}, y).
\end{aligned} \tag{7.72}$$

按后向顺序依次应用后向函数。首先，有

$$\frac{\partial J}{\partial z^{[r]}} = \mathcal{B}[\ell_{\text{logistic}}, z^{[r]}] \left( \frac{\partial J}{\partial J} \right) = \mathcal{B}[\ell_{\text{logistic}}, z^{[r]}](1). \tag{7.73}$$

然后，通过重复调用链式法则（公式 (7.58)），迭代计算  $\frac{\partial J}{\partial a^{[i]}}$  和  $\frac{\partial J}{\partial z^{[i]}}$ ：

$$\begin{aligned}
\frac{\partial J}{\partial a^{[r-1]}} &= \mathcal{B}[\text{MM}, a^{[r-1]}] \left( \frac{\partial J}{\partial z^{[r]}} \right) \\
\frac{\partial J}{\partial z^{[r-1]}} &= \mathcal{B}[\sigma, z^{[r-1]}] \left( \frac{\partial J}{\partial a^{[r-1]}} \right) \\
&\vdots \\
\frac{\partial J}{\partial z^{[1]}} &= \mathcal{B}[\sigma, z^{[1]}] \left( \frac{\partial J}{\partial a^{[1]}} \right).
\end{aligned} \tag{7.74}$$

数值上，通过重复调用公式 (7.69) 和 (7.63)，并选择不同的变量，计算这些量。注意到中间值  $a^{[i]}$  和  $z^{[i]}$  在反向传播（公式 (7.74)）中使用，因此这些值需要在前向传播后存储在内存中。

接下来，通过调用公式 (7.65) 和 (7.66)，计算参数的梯度：

$$\begin{aligned}
\frac{\partial J}{\partial W^{[r]}} &= \mathcal{B}[\text{MM}, W^{[r]}] \left( \frac{\partial J}{\partial z^{[r]}} \right) \\
\frac{\partial J}{\partial b^{[r]}} &= \mathcal{B}[\text{MM}, b^{[r]}] \left( \frac{\partial J}{\partial z^{[r]}} \right) \\
&\vdots \\
\frac{\partial J}{\partial W^{[1]}} &= \mathcal{B}[\text{MM}, W^{[1]}] \left( \frac{\partial J}{\partial z^{[1]}} \right) \\
\frac{\partial J}{\partial b^{[1]}} &= \mathcal{B}[\text{MM}, b^{[1]}] \left( \frac{\partial J}{\partial z^{[1]}} \right).
\end{aligned} \tag{7.75}$$

还注意到，公式 (7.75) 中的计算块可以与公式 (7.74) 中的计算块交织进行，因为一旦计算出  $\frac{\partial J}{\partial z^{[i]}}$ ，就可以计算出  $\frac{\partial J}{\partial W^{[i]}}$  和  $\frac{\partial J}{\partial b^{[i]}}$ 。

将所有这些放在一起，并调用公式 (7.72)、(7.74) 和 (7.75)，得到以下算法（算法 3）：

---

**Algorithm 3:** 多层神经网络的反向传播算法

---

1 前向过程：使用公式 (7.72) 计算出  $a^{[k]}$ ,  $z^{[k]}$ , and  $J$  的值并存储下来。

2 反向过程：计算  $J$  相对于  $z^{[r]}$  的梯度：

$$\frac{\partial J}{\partial z^{[r]}} = \mathcal{B}[\ell_{\text{logistic}}, z^{[r]}](1) = (1/(1 + \exp(-z^{[r]})) - y). \quad (7.76)$$

3 for  $k = r - 1$  to 0 do

4 计算相对于参数  $W^{[k+1]}$  和  $b^{[k+1]}$  的梯度：

$$\begin{aligned} \frac{\partial J}{\partial W^{[k+1]}} &= \mathcal{B}[\text{MM}, W^{[k+1]}] \left( \frac{\partial J}{\partial z^{[k+1]}} \right) \\ &= \frac{\partial J}{\partial z^{[k+1]}} a^{[k]\top}. \end{aligned} \quad (7.77)$$

$$\begin{aligned} \frac{\partial J}{\partial b^{[k+1]}} &= \mathcal{B}[\text{MM}, b^{[k+1]}] \left( \frac{\partial J}{\partial z^{[k+1]}} \right) \\ &= \frac{\partial J}{\partial z^{[k+1]}}. \end{aligned} \quad (7.78)$$

5 若  $k \geq 1$ , 计算相对于  $z^{[k]}$  和  $a^{[k]}$  的梯度：

$$\begin{aligned} \frac{\partial J}{\partial a^{[k]}} &= \mathcal{B}[\sigma, a^{[k]}] \left( \frac{\partial J}{\partial z^{[k+1]}} \right) \\ &= W^{[k+1]\top} \frac{\partial J}{\partial z^{[k+1]}}. \end{aligned} \quad (7.79)$$

$$\begin{aligned} \frac{\partial J}{\partial z^{[k]}} &= \mathcal{B}[\sigma, z^{[k]}] \left( \frac{\partial J}{\partial a^{[k]}} \right) \\ &= \sigma'(z^{[k]}) \odot \frac{\partial J}{\partial a^{[k]}}. \end{aligned} \quad (7.80)$$


---

#### 7.4.5 训练样本的向量化

正如在第 7.1 节中讨论的，在实现神经网络时，会利用多个样本之间的并行性。这意味着需要为以矩阵形式表示的多个训练样本编写神经网络的前向过程（输出的评估）和后向过程（反向传播）。

## 基本思想

基本思想很简单。假设有一个包含三个训练样本  $x^{(1)}, x^{(2)}, x^{(3)}$  的训练集。每个样本的第一层激活如下：

$$\begin{aligned} z^{[1](1)} &= W^{[1]}x^{(1)} + b^{[1]} \\ z^{[1](2)} &= W^{[1]}x^{(2)} + b^{[1]} \\ z^{[1](3)} &= W^{[1]}x^{(3)} + b^{[1]} \end{aligned}$$

注意方括号  $[ \cdot ]$  和圆括号  $( \cdot )$  的区别，前者指代层数，后者指代训练样本编号。直观上，可以使用循环来实现这一点。结果表明，也可以向量化这些操作。首先，定义：

$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} \in \mathbb{R}^{d \times 3} \quad (7.81)$$

注意，这里将训练样本按列堆叠而不是按行。然后可以合并为一个统一的公式：

$$Z^{[1]} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = W^{[1]}X + b^{[1]} \quad (7.82)$$

可能会注意到，这里将  $b^{[1]} \in \mathbb{R}^{4 \times 1}$  和  $W^{[1]}X \in \mathbb{R}^{4 \times 3}$  相加。在严格的线性代数中是不允许这样操作的。然而实践中，这种加法是使用广播 (*broadcast*) 执行的。创建一个中间的  $\tilde{b}^{[1]} \in \mathbb{R}^{4 \times 3}$ ：

$$\tilde{b}^{[1]} = \begin{bmatrix} | & | & | \\ b^{[1]} & b^{[1]} & b^{[1]} \\ | & | & | \end{bmatrix} \quad (7.83)$$

然后可以执行计算： $Z^{[1]} = W^{[1]}X + \tilde{b}^{[1]}$ 。通常没有必要显式地构造出  $\tilde{b}^{[1]}$ 。通过检查公式 (7.82) 中的维度，可以假定  $b^{[1]} \in \mathbb{R}^{4 \times 1}$  正确地广播到  $W^{[1]}X \in \mathbb{R}^{4 \times 3}$ 。

上述矩阵化方法可以很容易地推广到多层，但有一个细微之处需要注意，如下所述。

## 实现中的复杂性/细微之处

所有深度学习软件包或实现都将数据点放在数据矩阵的行中。（如果数据点本身是矩阵或张量，则将数据沿着第 0 维堆叠。）然而，大多数深度学习论文使

用与本讲义类似的表示法，其中数据点被视为列向量。<sup>8</sup>有一个简单的转换来处理这种不匹配：在实现中，所有列向量变成行向量，行向量变成列向量，所有矩阵都被转置，并且矩阵乘法的顺序被颠倒。在上面的例子中，使用行优先约定，数据矩阵是  $X \in \mathbb{R}^{3 \times d}$ ，第一层权重矩阵的维度是  $d \times m$ （而不是两层神经网络部分中的  $m \times d$ ），偏置向量  $b^{[1]} \in \mathbb{R}^{1 \times m}$ 。隐藏层激活的计算变为

$$Z^{[1]} = XW^{[1]} + b^{[1]} \in \mathbb{R}^{3 \times m} \quad (7.84)$$

---

<sup>8</sup>笔者猜测这主要是因为在数学中，习惯对向量左乘矩阵。

# **第三部分**

## **泛化与正则化**

# 第 8 章 泛化

本章将讨论理解和分析机器学习模型泛化能力的工具，即这些模型在未见过的测试样本上的表现。回想一下，对于监督学习问题，给定训练数据集  $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ ，通常通过最小化损失/成本函数  $J(\theta)$  来学习模型  $h_\theta$ ，这鼓励  $h_\theta$  拟合数据。例如，当损失函数是最小二乘损失（也称为均方误差）时，有  $J(\theta) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - h_\theta(x^{(i)}))^2$  这个用于训练的损失函数通常被称为**训练 (training)** 损失/误差/代价。

然而，最小化训练损失并非最终目标，它只是实现学习预测模型这一目标的途径。模型最重要的评估指标是在未见过的测试样本上的损失，通常被称为测试误差。形式上，从所谓的测试分布  $\mathcal{D}$  中抽取一个测试样本  $(x, y)$ ，并衡量模型在其上的误差，例如均方误差  $(h_\theta(x) - y)^2$ 。测试样本随机性下的期望损失/误差称为测试损失/误差。<sup>1</sup>

$$L(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[(y - h_\theta(x))^2] \quad (8.1)$$

请注意，误差的测量涉及计算期望，在实践中，通过对许多抽取的测试样本计算平均误差来近似，这些样本被称为测试数据集。这里训练集和测试集之间的关键区别在于测试样本是未见过的 (*unseen*)，因为训练过程没有使用这些测试样本。在经典的统计学习设置中，训练样本和测试样本都从与测试分布  $\mathcal{D}$  相同的分布中抽取，但测试样本对于学习过程来说仍然是未见过的，而训练样本是已见过的。<sup>2</sup>

由于训练集和测试集之间的这种关键区别，即使它们都从相同的分布  $\mathcal{D}$  中抽取，测试误差也不一定总是接近训练误差。<sup>3</sup> 因此，成功最小化训练误差可能并不总是导致测试误差很小。如果模型在训练集上准确预测数据，但在其他测试

<sup>1</sup> 在理论和统计文献中，通常将训练集  $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$  上的均匀分布称为经验分布，记为  $\mathcal{D}$ ，将总体分布称为  $\mathcal{D}$ 。因此（这是一部分原因），训练损失也称为经验损失/风险/误差，测试损失也称为总体损失/风险/误差。

<sup>2</sup> 近年来，研究人员越来越关注“域偏移”的情况，即训练分布和测试分布不同。

<sup>3</sup> 训练误差和测试误差之间的差异通常称为泛化差距。泛化误差在一些文献中指测试误差，在其他一些文献中指泛化差距。

样本上泛化能力不好，通常说模型**过拟合 (overfits)** 数据，即训练误差小而测试误差大。如果训练误差相对较大<sup>4</sup>（在这种情况下，测试误差通常也相对较大），则说模型**欠拟合 (underfits)** 数据。

本章研究学习过程如何影响测试误差，特别是模型参数化的选择。将测试误差分解为“偏差”和“方差”项，并研究模型参数化的选择及其权衡如何影响它们。利用偏差-方差权衡，将讨论何时发生过拟合和欠拟合以及如何避免。还将讨论 8.2 节中的双下降现象和 8.3 节中的一些经典理论结果。

## 8.1 偏差-方差均衡

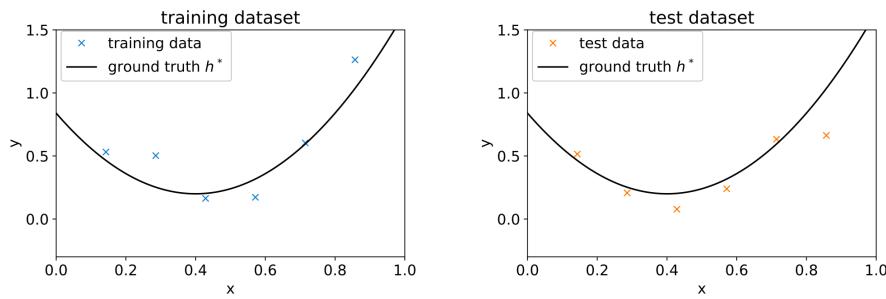


图 8.1: 用于本节的一个训练数据集和测试数据集的示例

作为说明性示例，考虑图 8.1 所示的训练数据集和测试数据集。训练输入  $x^{(i)}$  是随机选择的，输出  $y^{(i)}$  由  $y^{(i)} = h^*(x^{(i)}) + \xi^{(i)}$  生成，其中函数  $h^*(\cdot)$  是一个二次函数，在图 8.1 中以实线显示，而  $\xi^{(i)}$  是假定从  $\sim N(0, \sigma^2)$  生成的观测噪声。测试样本  $(x, y)$  也有相同的输入-输出关系  $y = h^*(x) + \xi$ ，其中  $\xi \sim N(0, \sigma^2)$ 。预测噪声  $\xi$  是不可能的，因此本质上我们的目标是恢复函数  $h^*(\cdot)$ 。

将考虑学习各种类型模型的测试误差。在讨论线性回归时，讨论了拟合“简单”模型，例如线性模型  $y = \theta_0 + \theta_1 x$ ，还是更“复杂”的模型，例如多项式模型  $y = \theta_0 + \theta_1 x + \dots + \theta_5 x^5$  的问题。

从拟合线性模型开始，如图 8.2 所示。即使在训练数据集上，最佳拟合线性模型也无法准确预测  $y$  与  $x$  的关系，更不用说在测试数据集上了。这是因为  $y$  和  $x$  之间的关系不是线性的——任何线性模型都远离真实函数  $h^*(\cdot)$ 。因此，训练误差很大，这是欠拟合的典型情况。

<sup>4</sup>例如，大于回归问题中数据的内在噪声水平。

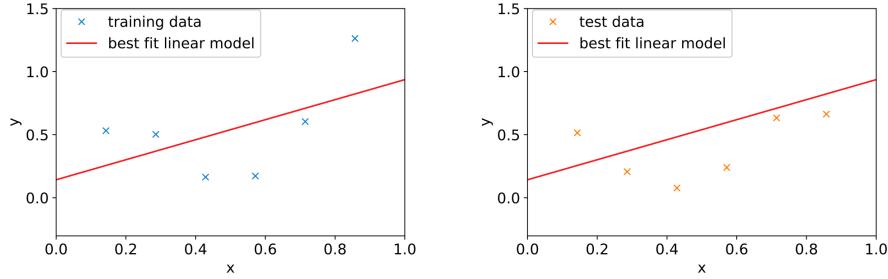


图 8.2: 最好的线性拟合模型也有着巨大的训练和测试误差。

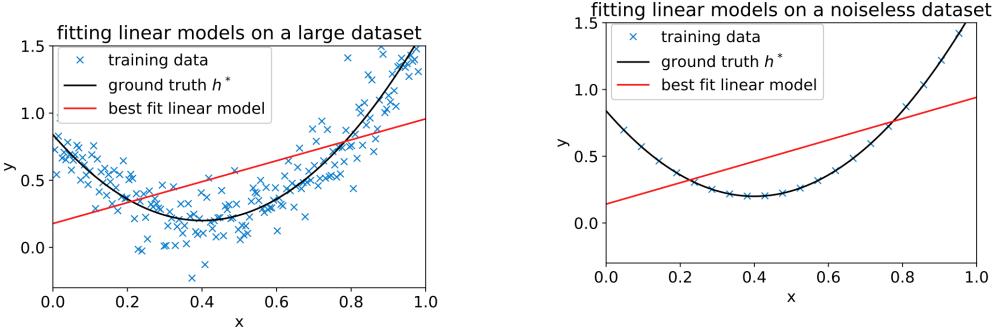


图 8.3: 最好的线性拟合模型在超大训练集上也有着巨大的训练误差。

图 8.4: 最好的线性拟合模型在没有噪声的训练集上也有着巨大的训练和测试误差。

这个问题不能通过增加训练样本来缓解——即使有非常大量的，甚至无限的训练样本，最佳拟合的线性模型仍然不准确，并且无法捕捉数据的结构（图 8.3）。即使训练数据中不存在噪声，问题仍然存在（图 8.4）。因此，这里的根本瓶颈在于线性模型族无法捕捉数据中的结构——线性模型无法表示真实的二次函数  $h^*$ ——而不是缺乏数据。非正式地，我们将模型的偏差（bias）定义为即使我们将其拟合到非常大（例如，无限大）的训练数据集时的测试误差。因此，在这种情况下，线性模型具有较大的偏差，并且欠拟合（即无法捕捉数据所表现出的结构）。

接下来，我们将一个 5 次多项式拟合到数据。图 8.5 表明它也未能学习到一个好的模型。然而，其失败模式与线性模型的情况不同。具体来说，尽管学到的 5 次多项式在预测训练样本的  $y^{(i)}$  与  $x^{(i)}$  时表现非常好，但在测试样本上效果不佳（图 8.5）。换句话说，从训练集学习到的模型无法很好地泛化（generalize）到其他测试样本——测试误差很高。与线性模型的行为相反，5 次多项式的偏差较小——如果我们将 5 次多项式拟合到非常大的数据集，得到的模型将接近二次函数并且也很准确（图 8.6）。这是因为 5 次多项式族包含所有二次函数（将

$\theta_5 = \theta_4 = \theta_3 = 0$  设置为零即可得到二次函数)，因此，原则上 5 次多项式能够捕捉数据的结构。

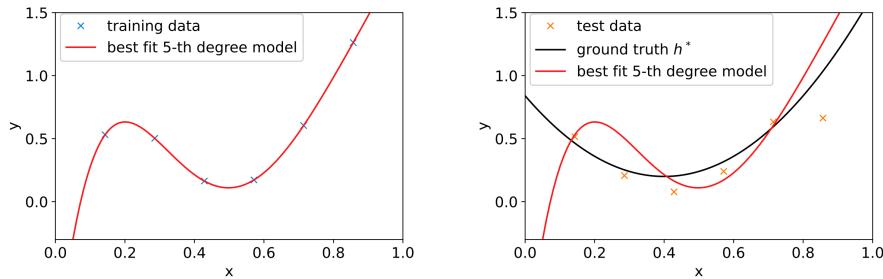


图 8.5：最佳的 5 次多项式拟合模型训练误差为零，但测试误差仍然很大，并且未能恢复真实情况。这是经典的过拟合情形。

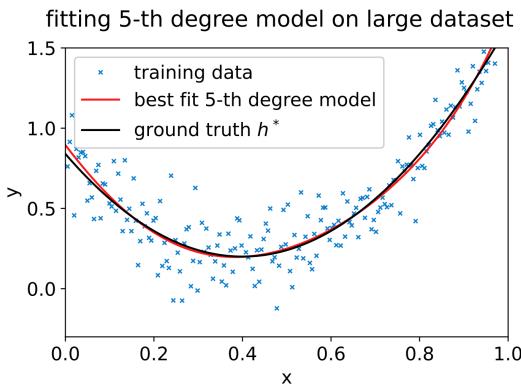


图 8.6：在大型数据集上拟合的最佳 5 次多项式几乎恢复了真实情况——这表明图 8.5 中的问题是方差（或数据不足）而不是偏差。

拟合 5 次多项式的失败可以用测试误差的另一个组成部分来解释，称为模型拟合过程的**方差 (variance)**。具体来说，如图 8.7 所示，在拟合 5 次多项式时，存在很大的风险，即我们拟合了数据中恰好存在于我们小而有限 (*small, finite*) 的训练集中的模式，但这些模式并不能反映  $x$  和  $y$  之间关系的更广泛模式。训练集中的这些“虚假”模式（大部分）是由于观测噪声  $\xi^{(i)}$  引起的，拟合这些虚假模式会导致模型具有较大的测试误差。在这种情况下，我们称模型具有较大的方差。

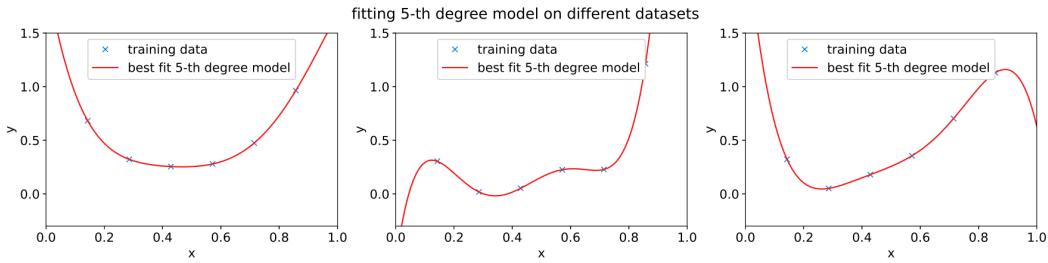


图 8.7: 在三个从同一分布产生的不同数据集上拟合的最佳 5 次多项式表现出极大不同，揭示其存在的巨大方差。

方差可以直观地（以及数学上证明，如第 8.1.1 节所示）通过在多个不同的训练数据集（从相同的潜在分布中抽取）上学习到的模型之间的变化量来表征。“虚假模式”是特定于特定数据集中的噪声（和输入）的随机性，因此在多个训练数据集之间是不同的。因此，对多个数据集的“虚假模式”过拟合应该会导致非常不同的模型。实际上，如图 8.7 所示，在三个不同训练数据集上学习到的模型差异很大，对每个数据集的“虚假模式”都存在过拟合。通常，偏差和方差之间存在权衡。如果我们的模型过于“简单”且参数很少，那么它可能具有较大的偏差（但方差较小），并且通常会遭受欠拟合。如果它过于“复杂”且参数很多，那么它可能遭受较大的方差（但偏差较小），因此会过拟合。图 8.8 展示了偏差和方差之间典型的权衡关系。

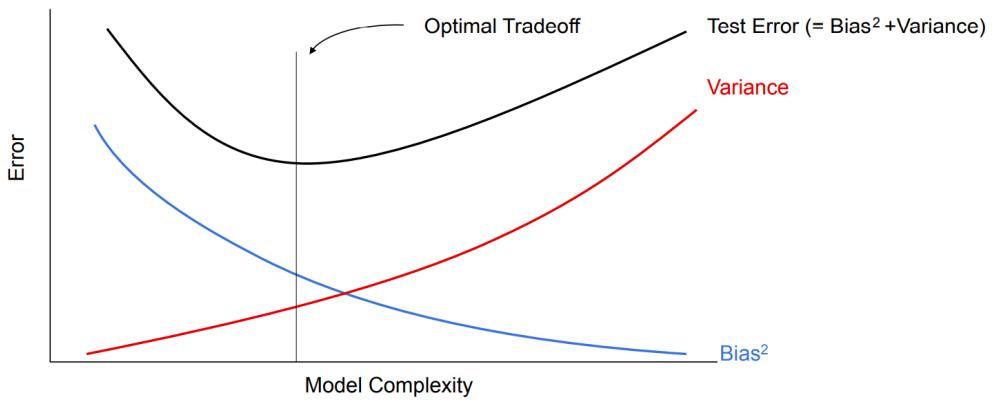


图 8.8: 典型的偏差-方差权衡示意图。

正如我们将在第 8.1.1 节中正式看到的，测试误差可以分解为偏差和方差之和。这意味着随着模型复杂度的增加，测试误差将呈现凸曲线，在实践中我们应该调整模型复杂度以达到最佳权衡。例如，在上面的例子中，拟合二次函数比拟合一次或五次多项式效果更好，如图 8.9 所示。

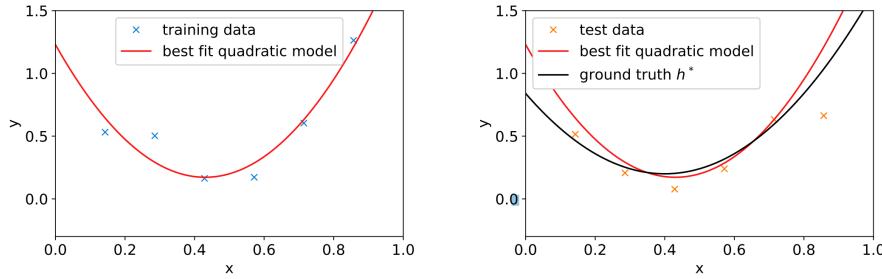


图 8.9: 最佳的二次拟合模型的训练误差和测试误差都很小, 因为二次模型达到了良好的偏差-方差权衡。

有趣的是, 偏差-方差权衡曲线或测试误差曲线并不普遍遵循图 8.8 中的形状, 至少当模型复杂度仅通过参数数量衡量时并非如此。(我们将在 8.2 节中讨论所谓的双下降现象。) 尽管如此, 偏差-方差权衡原理在分析和预测测试误差的行为时, 可能仍然是首选方法。

### 8.1.1 (对于回归问题的) 数学分解

为了形式化回归问题的偏差-方差权衡, 我们考虑如下设置 (这是 8.1 节开头段落的扩展)

- 抽取一个训练数据集  $S = \{x^{(i)}, y^{(i)}\}_{i=1}^n$ , 其中  $y^{(i)} = h^*(x^{(i)}) + \xi^{(i)}$  且  $\xi^{(i)} \in N(0, \sigma^2)$ 。
- 在数据集  $S$  上训练一个模型, 记为  $\hat{h}_S$ 。
- 取一个测试样本  $(x, y)$ , 使得  $y = h^*(x) + \xi$  且  $\xi \sim N(0, \sigma^2)$ , 并测量测试误差的期望 (在随机抽取的训练集  $S$  和随机的  $\xi$  上进行平均)。<sup>56</sup>

$$\text{MSE}(x) = \mathbb{E}_{S, \xi}[(y - \hat{h}_S(x))^2] \quad (8.2)$$

我们将把 MSE 分解为偏差项和方差项。我们从一个简单的数学工具开始, 该工具将在下面使用两次。

**断言 8.1.1.** 假设  $A$  和  $B$  是两个独立的实随机变量, 且  $\mathbb{E}[A] = 0$ 。则  $\mathbb{E}[(A+B)^2] = \mathbb{E}[A^2] + \mathbb{E}[B^2]$ 。

**推论:** 因为随机变量  $A$  与常数  $c$  独立, 当  $\mathbb{E}[A] = 0$  时, 我们有  $\mathbb{E}[(A+c)^2] = \mathbb{E}[A^2] + c^2$ 。

<sup>5</sup>为简单起见, 这里测试输入  $x$  被视为固定, 但在对  $x$  进行平均时, 同样的思想仍然成立。

<sup>6</sup>期望符号下的下标是为了强调在期望运算中被视为随机的变量。

断言的证明通过展开平方项得出:  $\mathbb{E}[(A + B)^2] = \mathbb{E}[A^2 + B^2 + 2AB] = \mathbb{E}[A^2] + \mathbb{E}[B^2] + 2\mathbb{E}[AB] = \mathbb{E}[A^2] + \mathbb{E}[B^2]$ 。这里我们使用了独立性来证明  $\mathbb{E}[AB] = \mathbb{E}[A]\mathbb{E}[B] = 0$ 。使用断言 8.1.1, 令  $A = \xi$  且  $B = h^*(x) - \hat{h}_S(x)$ , 我们有

$$\text{MSE}(x) = \mathbb{E}[(y - \hat{h}_S(x))^2] = \mathbb{E}[(\xi + (h^*(x) - \hat{h}_S(x)))^2] \quad (8.3)$$

$$= \mathbb{E}[\xi^2] + \mathbb{E}[(h^*(x) - \hat{h}_S(x))^2] \quad (\text{根据断言 8.1.1})$$

$$= \sigma^2 + \mathbb{E}[(h^*(x) - \hat{h}_S(x))^2] \quad (8.4)$$

然后, 我们定义  $h_{\text{avg}}(x) = \mathbb{E}_S[\hat{h}_S(x)]$  为“平均模型”——通过抽取无限多个数据集, 并在其上进行训练, 然后对它们在  $x$  上的预测进行平均而获得的模型。注意,  $h_{\text{avg}}$  是一个用于分析目的的假设模型, 它在现实中无法获得 (因为我们无法拥有无限多个数据集)。结果表明, 对于许多情况,  $h_{\text{avg}}$  (近似) 等于在具有无限样本的单个数据集上训练得到的模型。因此, 我们也可以直观地解释  $h_{\text{avg}}$ , 这与我们在上一小节中对偏差的直观定义一致。

我们可以通过令  $c = h^*(x) - h_{\text{avg}}(x)$  (这是一个不依赖于  $S$  选择的常数) 和  $A = h_{\text{avg}}(x) - \hat{h}_S(x)$  来进一步分解  $\text{MSE}(x)$ , 这符合断言 8.1.1 的推论:

$$\text{MSE}(x) = \sigma^2 + \mathbb{E}[(h^*(x) - \hat{h}_S(x))^2] \quad (8.5)$$

$$= \sigma^2 + \mathbb{E}[(h^*(x) - h_{\text{avg}}(x) + h_{\text{avg}}(x) - \hat{h}_S(x))^2] \quad (8.6)$$

$$= \underbrace{\sigma^2}_{\text{不可避免}} + \underbrace{(h^*(x) - h_{\text{avg}}(x))^2}_{\text{偏差}^2} + \underbrace{\text{var}(\hat{h}_S(x))}_{\text{方差}} \quad (8.7)$$

我们将第二项称为偏差 (平方), 第三项称为方差。如前所述, 偏差捕捉了由于模型表达能力不足而引入的误差部分。回想一下,  $h_{\text{avg}}$  可以被认为是即使在无限数据下学习到的最佳可能模型。因此, 偏差并非根本上由数据不足引起, 而是由模型族本身无法很好地近似  $h^*$  造成的。例如, 在图 8.2 所示的例子中, 由于任何线性模型都无法近似真实的二次函数  $h^*$ , 因此  $h_{\text{avg}}$  也不能, 从而偏差项很大。方差项捕捉了有限数据集的随机性如何引入学习模型中的误差。它衡量了学习模型对数据集中随机性的敏感度。随着数据集大小的增加, 方差通常会减小。对于第一项  $\sigma^2$ , 我们无能为力, 因为根据定义, 我们无法预测噪声  $\xi$ 。最后, 我们注意到, 分类问题的偏差-方差分解远不如回归问题清晰。已经有一些提案, 但对于什么是“正确”和/或最有用的形式还没有达成一致。

## 8.2 双下降现象

### 模型层面的双下降现象

最近的研究表明，在包括线性模型和深度神经网络在内的一系列机器学习模型中，测试误差会呈现出一种“双下降”现象。<sup>7</sup> 正如 8.1 节中所讨论的传统观点是，随着模型复杂度的增加，测试误差先下降然后上升，如图 8.8 所示。然而，在许多情况下，我们经验性地观察到测试误差可以有第二次下降——它先下降，然后在大到足以很好地拟合所有训练数据时达到峰值附近，然后在所谓的过参数化区域再次下降，其中参数数量大于数据点数量。图 8.10 展示了测试误差随模型复杂度（由参数数量衡量）变化的典型曲线。在某种程度上，过参数化区域的第二次下降被认为是机器学习领域的发现——部分原因是轻度正则化的过参数化模型在深度学习时代得到了广泛应用。这种现象的一个实际意义是，不应回避扩大模型规模和尝试过参数化模型，因为测试误差很可能再次下降到比之前的最低点更低的水平。实际上，在许多情况下，更大的过参数化模型总是能带来更好的测试性能（这意味着第二次下降之后不会出现第二次上升）。

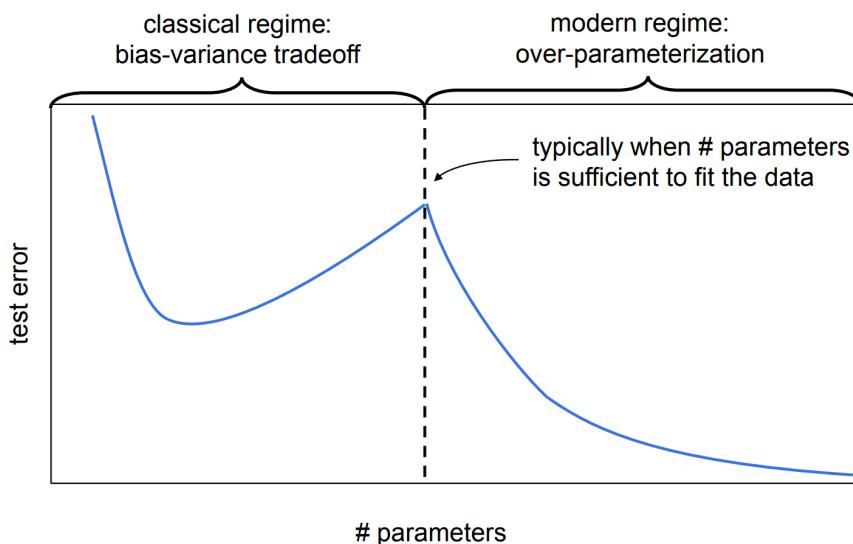


图 8.10: 典型的模型层面双下降现象。随着参数数量的增加，当参数数量小于训练数据时，测试误差先下降。然后在过参数化区域，测试误差再次下降。

<sup>7</sup>该现象的发现可能可以追溯到 Opper 1995; Opper 2001，最近由 Belkin, Hsu, and Xu 2020; Hastie et al. 2022 推广。

## 样本层面的双下降现象

从先验知识来看，我们期望更多的训练样本总是能带来更小的测试误差——更多的样本为算法提供了更严格的信息来学习。然而，最近的研究 [Nakkiran 2019] 观察到，随着样本数量的增加，测试误差并非单调递减。相反，如图 8.11 所示，测试误差先下降，然后在样本数量（记为  $n$ ）与参数数量（记为  $d$ ）接近时增加并达到峰值，然后再次下降。我们将此称为样本层面的双下降现象。在某种程度上，样本层面的双下降和模型层面的双下降本质上描述的是类似的现象——测试误差在  $n \approx d$  时达到峰值。

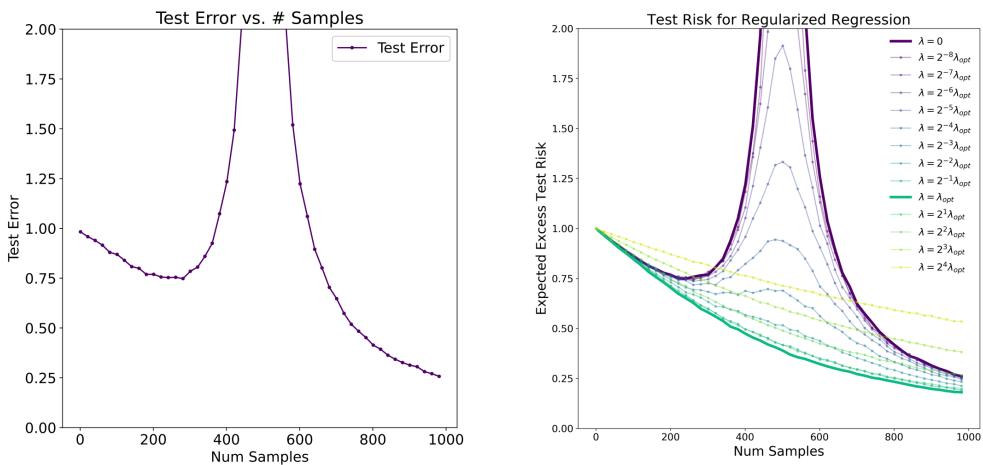


图 8.11：左：线性模型在样本层面的双下降现象。右：不同正则化强度下线性模型在样本层面的双下降现象。使用最优正则化参数（对每个  $n$  都调到最优，以绿色实线显示）缓解双下降。设置： $(x, y)$  的数据分布为  $x \sim \mathcal{N}(0, I_d)$  且  $y \sim x^\top \beta + \mathcal{N}(0, \sigma^2)$ ，其中  $d = 500$ ,  $\sigma = 0.5$  且  $\|\beta\|_2 = 1$ .<sup>8</sup>

## 解释和缓解策略

样本层面的双下降，特别是测试误差在  $n \approx d$  处的峰值，表明在这些实验中评估的现有训练算法在  $n \approx d$  时远非最优。我们可以通过丢弃一些样本，使用较小的样本量运行算法来避免峰值。换句话说，原则上存在其他算法可以在  $n \approx d$  时实现更小的测试误差，但这些实验中评估的算法未能做到。学习过程的次优性似乎是样本层面和模型层面双下降峰值的罪魁祸首。实际上，通过最优调整的正则化（将在第 9 章中更详细讨论），在  $n \approx d$  区域的测试误差可以显著改

<sup>8</sup>此图依照 Nakkiran et al. 2020 的图 1 复现，类似的现象也可以在 Hastie et al. 2022; Mei, and Montanari 2022 中观察到。

善，并且模型层面和样本层面的双下降现象都得到了缓解。参见图 8.11。上述直觉只解释了模型层面和样本层面双下降的峰值，但没有解释模型层面双下降的第二次下降——为什么过参数化模型能够很好地泛化。对过参数化模型的理论理解是一个活跃的研究领域，最近取得了许多进展。一个典型的解释是，常用的优化器（如梯度下降）提供了隐式正则化效应（将在第 9.2 节中更详细讨论）。换句话说，即使在过参数化区域且使用了非正则化的损失函数，模型仍然隐式地进行了正则化，因此表现出比拟合数据的任意解更好的测试性能。例如，对于线性模型，当  $n \ll d$  时，使用零初始化进行梯度下降的优化器会找到拟合数据的最小范数 (*minimum norm*) 解（而不是拟合数据的任意解），而最小范数正则化器对于过参数化区域来说是一个足够好的正则化器（但在  $n \approx d$  时不是一个好的正则化器，导致测试误差达到峰值）。

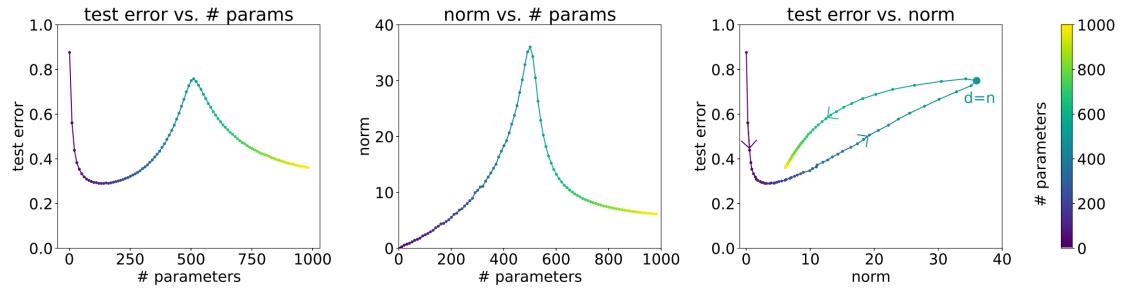


图 8.12：左：双下降现象，其中模型复杂度用参数数量来衡量。中：学习到的模型的范数在  $n \approx d$  附近达到峰值。右：测试误差与学习到的模型范数的关系。颜色条表示参数数量，箭头表示模型尺寸增加的方向。它们的关系更接近于传统认知，而不是双下降。**设置：**我们考虑一个样本量固定为  $n = 500$  的线性回归。输入  $x$  是 Fashion-MNIST 上的随机 ReLU 特征，输出  $y \in \mathbb{R}^{10}$  是 one-hot 标签。这与 Nakkiran et al. 2020 第 5.2 节中的设置相同。

最后，我们还要指出，双下降现象主要在模型的复杂度用参数数量来衡量时被观察到。参数数量是否以及何时是衡量模型复杂度的最佳指标尚不清楚。例如，在许多情况下，模型的范数被用作复杂度度量。如图 8.12 右图所示，对于一个特定的线性情况，如果我们绘制测试误差与学习到的模型范数的关系，双下降现象就不再发生。这部分是因为学习到的模型的范数在  $n \approx d$  附近也达到峰值（参见图 8.12（中）或 Belkin, Hsu, Ma, et al. 2019、Mei, and Montanari 2022，以及 James et al. 2021 在第 10.8 节中的讨论）。对于深度神经网络，正确的复杂度度量更加难以捉摸。双下降现象的研究仍是一个活跃的研究课题。

## 8.3 样本复杂度边界 (选读)

### 8.3.1 预备知识

在本节中，我们将开始探索学习理论。除了其本身具有的趣味性和启发性之外，本节的讨论还将帮助我们锻炼直觉，并推导如何在不同情境下最优地应用学习算法的经验法则。我们还将尝试回答几个关键问题：首先，我们能否形式化地描述刚刚讨论的偏差/方差权衡？这将最终引导我们讨论模型选择方法，例如，它可以自动决定将多项式拟合到训练集的阶数。其次，在机器学习中，我们真正关心的是泛化误差，但大多数学习算法是根据训练集来拟合模型的。为什么在训练集上表现良好能说明泛化误差方面的问题呢？具体来说，我们能否将训练集上的误差与泛化误差联系起来？第三也是最后一个问题是，是否存在一些条件，在这些条件下我们可以实际证明学习算法是有效的？

我们先两个简单但非常有用的引理开始。

**引理.** (并集上界) 设  $A_1, A_2, \dots, A_k$  是  $k$  个不同的事件 (它们可能不相互独立)。那么

$$P(A_1 \cup \dots \cup A_k) \leq P(A_1) + \dots + P(A_k).$$

在概率论中，并集上界通常被视为公理（因此我们不尝试证明它），但它也具有直观意义： $k$  个事件中任意一个发生的概率至多是这  $k$  个不同事件的概率之和。

**引理.** (Hoeffding 不等式) 设  $Z_1, \dots, Z_n$  是从  $\text{Bernoulli}(\phi)$  分布中独立同分布 (iid) 抽取的随机变量。即  $P(Z_i = 1) = \phi$ ，且  $P(Z_i = 0) = 1 - \phi$ 。设  $\hat{\phi} = (1/n) \sum_{i=1}^n Z_i$  是这些随机变量的均值，并任取  $\gamma > 0$  并固定住。那么

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2 \exp(-2\gamma^2 n).$$

这个引理（在学习理论中也称为 **Chernoff 界 (Chernoff bound)**）表明，如果我们取  $\hat{\phi}$  作为  $n$  个  $\text{Bernoulli}(\phi)$  随机变量的平均值来估计  $\phi$  的真值，那么我们的估计值与真值相差很大的概率会很小，只要  $n$  足够大。换句话说，如果你有一个抛出正面的概率为  $\phi$  的有偏硬币，那么如果你抛掷  $n$  次，计算出现正面的比例。当  $n$  很大的时候，这大概率是  $\phi$  的一个很好的估计值。

利用这两个引理，我们将能够证明学习理论中一些深刻且重要的结果。

为了简化，我们将重点放在标签  $y \in \{0, 1\}$  的二分类问题上。我们在这里讨论的内容可以推广到其他问题，包括回归和多分类问题。

假设我们有一个大小为  $n$  的训练集  $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, n\}$ , 其中训练样本  $(x^{(i)}, y^{(i)})$  是从概率分布  $\mathcal{D}$  中独立同分布地抽取的。对于一个假设  $h$ , 我们将学习理论中的**训练误差 (training error)** (也称为**经验风险 (empirical risk)** 或**经验误差 (empirical error)**) 定义为

$$\hat{\varepsilon}(h) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}\{h(x^{(i)}) \neq y^{(i)}\}.$$

这是  $h$  错误分类的训练样本的比例。当我们要明确指出  $\hat{\varepsilon}(h)$  对训练集  $S$  的依赖性时, 也可以将其写成  $\hat{\varepsilon}_S(h)$ 。我们还定义泛化误差为

$$\varepsilon(h) = P_{(x,y) \sim \mathcal{D}}(h(x) \neq y).$$

也就是说, 这是如果我们从分布  $\mathcal{D}$  中抽取一个新的样本  $(x, y)$ ,  $h$  将其错误分类的概率。

请注意, 我们假设训练数据是从我们将用于评估我们的假设的相同 (*same*) 分布  $\mathcal{D}$  中抽取的 (在泛化误差的定义中)。这有时也被视为 **PAC** 假设中的一条。<sup>9</sup>

考虑线性分类的设置, 并令  $h_\theta(x) = \mathbb{1}\{\theta^\top x \geq 0\}$ 。拟合参数  $\theta$  的合理方法是什么? 一种方法是尝试最小化训练误差, 并选择

$$\hat{\theta} = \arg \min_{\theta} \hat{\varepsilon}(h_\theta).$$

我们将这个过程称为**经验风险最小化 (empirical risk minimization, ERM)**, 学习算法的输出假设为  $\hat{h} = h_{\hat{\theta}}$ 。我们将 ERM 视为最“基本”的学习算法, 并且这个算法也是本节的重点。(逻辑回归等算法也可以看作是经验风险最小化的近似。)

在学习理论的研究中, 将假设的具体参数化以及决策边界是否线性的问题抽象出来将会很有帮助。我们定义学习算法使用的**假设类 (hypothesis class)**  $\mathcal{H}$  为其考虑的所有分类器集合。对于线性分类,  $\mathcal{H} = \{h_\theta : h_\theta(x) = \mathbb{1}\{\theta^\top x \geq 0\}, \theta \in \mathbb{R}^{d+1}\}$ , 也就是所有在  $\mathcal{X}$  (输入的域) 上决策边界为线性的分类器的集合。更广泛地说, 如果我们研究的是神经网络, 那么我们可以令  $\mathcal{H}$  是由某种神经网络结构表示的所有分类器的集合。

经验风险最小化现在可以被视为在函数空间  $\mathcal{H}$  上的最小化问题, 其中学习算法选择的假设是

$$\hat{h} = \arg \min_{h \in \mathcal{H}} \hat{\varepsilon}(h).$$

---

<sup>9</sup>PAC (Probably Approximately Correct) 代表“可能近似正确”, 它是学习理论中大量结果得以证明的框架和假设集合。其中, 训练和测试样本来自相同分布的假设以及训练样本独立抽取的假设是最重要的。

### 8.3.2 有限 $\mathcal{H}$ 的情况

让我们首先考虑一个学习问题，其有一个有限的假设类  $\mathcal{H} = \{h_1, \dots, h_k\}$ ，包含  $k$  个假设。因此， $\mathcal{H}$  是从  $\mathcal{X}$  到  $\{0, 1\}$  的  $k$  个函数的集合，经验风险最小化选择  $\hat{h}$  为这些函数中训练误差最小的那个。

我们希望对  $\hat{h}$  的泛化误差给出保证。我们的策略将分为两部分：首先，我们将证明  $\hat{\varepsilon}(h)$  是对所有  $h$  的  $\varepsilon(h)$  的可靠估计。其次，我们将证明这隐含了  $\hat{h}$  的泛化误差的上界。

取任意一个固定的  $h_i \in \mathcal{H}$ 。考虑一个伯努利随机变量  $Z$ ，其分布定义如下。我们将从分布  $\mathcal{D}$  中抽取样本  $(x, y)$ 。然后，我们设置  $Z = 1\{h_i(x) \neq y\}$ 。也就是说，我们将抽取一个样本，并让  $Z$  表示  $h_i$  是否将其错误分类。类似地，我们还定义  $Z_j = 1\{h_i(x^{(j)}) \neq y^{(j)}\}$ 。由于我们的训练集是从  $\mathcal{D}$  中独立同分布地抽取的， $Z$  和  $Z_j$  具有相同的分布。

我们看到，随机抽取样本被错误分类的概率——即  $\varepsilon(h_i)$ ——恰好是  $Z$ （以及  $Z_j$ ）的期望值。此外，训练误差可以写成

$$\hat{\varepsilon}(h_i) = \frac{1}{n} \sum_{j=1}^n Z_j.$$

因此， $\hat{\varepsilon}(h_i)$  恰好是从均值为  $\varepsilon(h_i)$  的伯努利分布中独立同分布地抽取的  $n$  个随机变量  $Z_j$  的均值。因此，我们可以应用 Hoeffding 不等式，得到

$$P(|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) \leq 2 \exp(-2\gamma^2 n).$$

这表明，对于我们特定的  $h_i$ ，训练误差将依概率收敛到泛化误差，假设  $n$  很大。但我们不只是想保证  $\hat{\varepsilon}(h_i)$  将依概率收敛到  $\varepsilon(h_i)$ ，对于某一个特定的  $h_i$ 。我们希望证明这对所有  $h \in \mathcal{H}$  同时成立。为此，令  $A_i$  表示事件  $|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma$ 。我们已经证明，对于任意特定的  $A_i$ ，有  $P(A_i) \leq 2 \exp(-2\gamma^2 n)$ 。因此，利用并集上界，我们有

$$\begin{aligned} P(\exists h \in \mathcal{H}. |\varepsilon(h) - \hat{\varepsilon}(h)| > \gamma) &= P(A_1 \cup \dots \cup A_k) \\ &\leq \sum_{i=1}^k P(A_i) \\ &\leq \sum_{i=1}^k 2 \exp(-2\gamma^2 n) \\ &= 2k \exp(-2\gamma^2 n). \end{aligned}$$

如果用 1 中减去不等式两侧，可以发现

$$\begin{aligned} P(\neg \exists h \in \mathcal{H}. |\varepsilon(h) - \hat{\varepsilon}(h)| > \gamma) &= P(\forall h \in \mathcal{H}. |\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma) \\ &\geq 1 - 2k \exp(-2\gamma^2 n). \end{aligned}$$

(“ $\neg$ ” 符号表示“逻辑非”。) 因此可以保证，对于所有  $h \in \mathcal{H}$ ,  $\hat{\varepsilon}(h)$  在  $\varepsilon(h)$  的  $\gamma$  范围内的概率不低于  $1 - 2k \exp(-2\gamma^2 n)$ 。这被称为一致收敛 (*uniform convergence*) 结果，因为这是一个对所有（而不是单独一个） $h \in \mathcal{H}$  同时成立的界。

在上面的讨论中，我们所做的，是对于特定的  $n$  和  $\gamma$  值，给出了某些  $h \in \mathcal{H}$  使得  $|\varepsilon(h) - \hat{\varepsilon}(h)| > \gamma$  的概率的界。这里有三个感兴趣的量： $n$ 、 $\gamma$  和误差概率；我们可以用另外两个量来界定其中任何一个。

例如，给定  $\gamma$  和某个  $\delta > 0$ ，以至少  $1 - \delta$  的概率保证训练误差将在泛化误差的  $\gamma$  范围内时， $n$  需要多大？令  $\delta = 2k \exp(-2\gamma^2 n)$  并解出  $n$ ，可以发现当

$$n \geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta},$$

时，对于所有  $h \in \mathcal{H}$ ，得到  $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$  的概率至少是  $1 - \delta$ （等价地，这表明对于某个  $h \in \mathcal{H}$  使得  $|\varepsilon(h) - \hat{\varepsilon}(h)| > \gamma$  的概率最多为  $\delta$ 。）这个界告诉我们为了做出保证需要多少训练样本。算法达到一定性能所需的训练集大小  $n$  也被称为算法的**样本复杂度** (**sample complexity**)。

上述界的关键性质是，这个性能保证所需的训练样本数量与假设类  $\mathcal{H}$  中假设的数量  $k$  成对数 (*logarithmic*) 关系。这一点在后面会很重要。

类似地，我们也可以固定  $n$  和  $\delta$ ，解出  $\gamma$ ，并证明对于所有  $h \in \mathcal{H}$ ，我们以  $1 - \delta$  的概率有

$$|\hat{\varepsilon}(h) - \varepsilon(h)| \leq \sqrt{\frac{1}{2n} \log \frac{2k}{\delta}}.$$

现在，假设一致收敛成立，即对于所有  $h \in \mathcal{H}$ ,  $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$ 。关于我们的学习算法选择的  $\hat{h} = \arg \min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$  的泛化误差，我们可以证明什么？

定义  $h^* = \arg \min_{h \in \mathcal{H}} \varepsilon(h)$  为  $\mathcal{H}$  中最好的假设。注意， $h^*$  是给定我们正在使用  $\mathcal{H}$  的情况下，我们可能做到的最好的结果，因此将我们的性能与  $h^*$  的性能进行比较是有意义的。我们有：

$$\begin{aligned} \varepsilon(\hat{h}) &\leq \hat{\varepsilon}(\hat{h}) + \gamma \\ &\leq \hat{\varepsilon}(h^*) + \gamma \\ &\leq \varepsilon(h^*) + 2\gamma \end{aligned}$$

第一行使用了  $|\varepsilon(\hat{h}) - \hat{\varepsilon}(\hat{h})| \leq \gamma$  (根据我们的一致收敛假设)。第二行使用了  $\hat{h}$  被选择为最小化  $\hat{\varepsilon}(h)$ ，因此对于所有  $h$ ,  $\hat{\varepsilon}(\hat{h}) \leq \hat{\varepsilon}(h)$ ，特别地， $\hat{\varepsilon}(\hat{h}) \leq \hat{\varepsilon}(h^*)$ 。第三

行再次使用了一致收敛假设，表明  $\hat{\varepsilon}(h^*) \leq \varepsilon(h^*) + \gamma$ 。因此，我们已经证明了以下结论：如果一致收敛成立，那么  $\hat{h}$  的泛化误差最多比  $\mathcal{H}$  中最好的可能假设差  $2\gamma$ 。

让我们把这些归纳为一个定理。

**定理.** 设  $|\mathcal{H}| = k$ ，任取  $n, \delta$  并固定住。那么以至少  $1 - \delta$  的概率有

$$\varepsilon(\hat{h}) \leq \left( \min_{h \in \mathcal{H}} \varepsilon(h) \right) + 2 \sqrt{\frac{1}{2n} \log \frac{2k}{\delta}}.$$

这个定理的证明是通过令  $\gamma$  等于根号项，利用我们之前关于一致收敛以至少  $1 - \delta$  的概率发生的论证，并注意到一致收敛意味着  $\varepsilon(\hat{h})$  最多比  $\varepsilon(h^*) = \min_{h \in \mathcal{H}} \varepsilon(h)$  高  $2\gamma$ （如我们之前所示）。

这也量化了我们之前关于模型选择中的偏差/方差权衡的说法。具体来说，假设我们有一个假设类  $\mathcal{H}$ ，并且正在考虑切换到某个更大的假设类  $\mathcal{H}' \supseteq \mathcal{H}$ 。如果我们切换到  $\mathcal{H}'$ ，那么第一项  $\min_{h \in \mathcal{H}} \varepsilon(h)$  只能减小（因为我们将在更大的函数集合上取最小值）。因此，通过使用更大的假设类进行学习，我们的“偏差”只会减小。然而，如果  $k$  增加，那么第二个  $2\sqrt{\cdot}$  项也会增加。这种增加对应于当我们使用更大的假设类时，“方差”的增加。

通过固定  $\gamma$  和  $\delta$  并像之前一样解出  $n$ ，我们也可以得到以下样本复杂度界：

**推论.** 设  $|\mathcal{H}| = k$ ，任取  $\gamma, \delta$  并固定住。那么为了使  $\varepsilon(\hat{h}) \leq \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$  以至少  $1 - \delta$  的概率成立，只需

$$\begin{aligned} n &\geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta} \\ &= O\left(\frac{1}{\gamma^2} \log \frac{k}{\delta}\right) \end{aligned}$$

### 8.3.3 无限 $\mathcal{H}$ 的情况

我们已经证明了一些关于有限假设类的有用定理。但是有许多假设类，包括任何由实数参数化的假设类（例如线性分类），实际上包含无限多的函数。我们能否在这种情况下证明类似的结果？

让我们从一个不 (*not*) “正确”的论证开始。存在更好、更一般的论证 (*Better and more general arguments exist*)，但现在的论证可以培养我们对该领域的直觉。

假设我们有一个由  $d$  个实数参数化的  $\mathcal{H}$ 。由于我们使用计算机来表示实数，并且 IEEE 双精度浮点数（C 语言中的 `double`）使用 64 位来表示浮点数，这意味着，如果使用的是双精度浮点数，我们的学习算法实际上由  $64d$  位参数化。因

此，我们的假设类实际上最多包含  $k = 2^{64d}$  个不同的假设。从上一节末尾的推论中，我们发现，为了保证  $\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma$  以至少  $1 - \delta$  的概率成立，只需

$$n \geq O\left(\frac{1}{\gamma^2} \log \frac{2^{64d}}{\delta}\right) = O_{\gamma, \delta}\left(\frac{d}{\gamma^2} \log \frac{1}{\delta}\right).$$

(下标  $\gamma, \delta$  表示最后一个大  $O$  隐藏了可能依赖于  $\gamma$  和  $\delta$  的常数。) 因此，所需的训练样本数量最多与模型的参数数量呈线性 (*linear*) 关系。

依赖于 64 位浮点数这一点并不完全令人满意，但结论大致是正确的：如果我们的目标是最小化训练误差，那么为了使具有  $d$  个参数的假设类“很好地”学习，我们通常需要与  $d$  呈线性关系的训练样本数量。

(值得注意的是，这些结果是针对使用经验风险最小化的算法证明的。因此，虽然样本复杂度对  $d$  的线性依赖性通常适用于大多数旨在最小化训练误差或其近似值的判别式学习算法，但这些结论并不总是直接适用于非判别式学习算法。为许多非 ERM 学习算法提供良好的理论保证仍然是一个活跃的研究领域。)

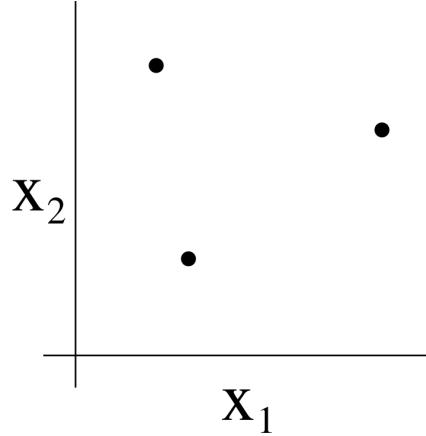
之前论证中另一个稍微令人不满意的部分是它依赖于  $\mathcal{H}$  的参数化。直观上，这似乎不应该很重要：我们曾将线性分类器的类写成  $h_\theta(x) = 1\{\theta_0 + \theta_1x_1 + \dots + \theta_dx_d \geq 0\}$ ，其具有  $n+1$  个参数  $\theta_0, \dots, \theta_d$ 。但它也可以写成  $h_{u,v}(x) = 1\{(u_0^2 - v_0^2) + (u_1^2 - v_1^2)x_1 + \dots + (u_d^2 - v_d^2)x_d \geq 0\}$ ，这样就有  $2d+2$  个参数  $u_i, v_i$ 。然而，这两种方式都定义了相同的假设类  $\mathcal{H}$ :  $d$  维空间中的线性分类器集合。

为了推导出更令人满意的论证，让我们对更多的概念进行定义。

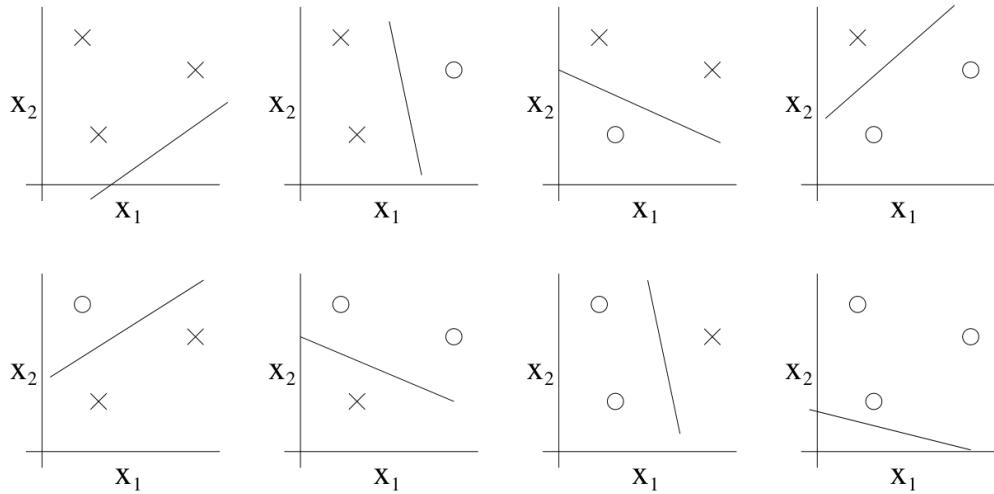
给定一个点集  $S = \{x^{(i)}, \dots, x^{(D)}\}$  (与训练集无关)，其中  $x^{(i)} \in \mathcal{X}$ ，如果  $\mathcal{H}$  可以在  $S$  上实现任何标记，那么就称  $\mathcal{H}$  打散 (*shatters*) 了  $S$ 。也就是说，对于任何标记集  $\{y^{(1)}, \dots, y^{(D)}\}$ ，存在某个  $h \in \mathcal{H}$  使得  $h(x^{(i)}) = y^{(i)}$  对所有  $i = 1, \dots, D$  成立。

给定一个假设类  $\mathcal{H}$ ，我们定义其 **Vapnik-Chervonenkis 维数** (**Vapnik-Chervonenkis dimension**)，记作  $\text{VC}(\mathcal{H})$  为  $\mathcal{H}$  可以打散的最大集合的大小。(如果  $\mathcal{H}$  可以打散任意大的集合，那么  $\text{VC}(\mathcal{H}) = \infty$ 。)

例如，考虑下面这个包含三个点的集合：

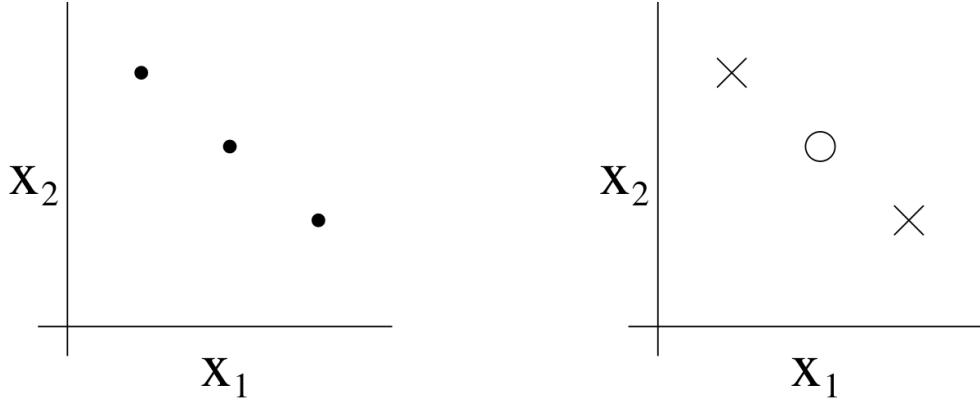


二维线性分类器组成的假设类  $\mathcal{H}$  ( $h(x) = 1\{\theta_0 + \theta_1x_1 + \theta_2x_2 \geq 0\}$ ) 能否打散上述集合？答案是肯定的。具体来说，我们看到，对于这八种可能的标记方式中的任何一种，我们都可以找到一个线性分类器，使得其在这三点上的“训练误差”为零：



此外，可以证明不存在任何大小为 4 的点集是该假设类可以打散的。因此， $\mathcal{H}$  可以打散的最大集合的大小为 3，所以  $VC(\mathcal{H}) = 3$ 。

注意，这里的  $\mathcal{H}$  的 VC 维数为 3，即使可能存在一些大小为 3 的集合它无法打散。例如，如果我们有三个点共线（左图），那么对于下面右图所示的三个点的标记方式，无法找到一个线性分隔：



换句话说，根据 VC 维数的定义，为了证明  $\text{VC}(\mathcal{H})$  至少为  $\mathbf{D}$ ，我们只需要证明存在至少一个 (*one*) 大小为  $\mathbf{D}$  的集合是  $\mathcal{H}$  可以打散的。

下面的定理由 Vapnik 证明。(许多人会认为这是学习理论中最重要的定理。)

**定理.** 给定假设类  $\mathcal{H}$ ，令  $\mathbf{D} = \text{VC}(\mathcal{H})$ 。那么对于所有  $h \in \mathcal{H}$ ，以至少  $1 - \delta$  的概率有：

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq O\left(\sqrt{\frac{\mathbf{D}}{n} \log \frac{n}{\mathbf{D}}} + \frac{1}{n} \log \frac{1}{\delta}\right).$$

因此，以至少  $1 - \delta$  的概率，也有：

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + O\left(\sqrt{\frac{\mathbf{D}}{n} \log \frac{n}{\mathbf{D}}} + \frac{1}{n} \log \frac{1}{\delta}\right).$$

换句话说，如果一个假设类具有有限的 VC 维数，那么当  $n$  变大时，就会产生一致收敛。像之前一样，这使得我们能够给出  $\varepsilon(\hat{h})$  的一个界限，该界限以  $\varepsilon(h^*)$  表示。我们还有下面的推论：

**推论.** 对于所有  $h \in \mathcal{H}$ ，要使  $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$  以至少  $1 - \delta$  的概率成立（因此  $\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma$ ），只需  $n = O_{\gamma, \delta}(\mathbf{D})$ 。

换句话说，使  $\mathcal{H}$  学习得“很好”，所需的训练样本数量与  $\mathcal{H}$  的 VC 维数呈线性关系。结果表明，对于“大多数”假设类，VC 维数（假设“合理地”进行参数化）也与参数数量大致呈线性关系。将这些结合起来，我们得出结论：对于给定的假设类  $\mathcal{H}$ （以及最小化训练误差的算法），达到接近最优分类器泛化误差所需的训练样本数量通常与  $\mathcal{H}$  的参数数量大致呈线性关系。

# 第 9 章 正则化与模型选择

## 9.1 正则化

回顾第 8.1 节中讨论的，过拟合通常是由于使用了过于复杂的模型导致的，我们需要选择适当的模型复杂度以达到最优的偏差-方差权衡。当模型复杂度用参数数量衡量时，我们可以改变模型的规模（例如，神经网络的宽度）。然而，衡量模型复杂度的正确且有信息量的方式可以是参数的函数（例如，参数的  $\ell_2$  范数），这可能不一定取决于参数的数量。在这种情况下，我们将使用正则化，这是一种重要的机器学习技术，用于控制模型复杂度并防止过拟合。

正则化通常涉及在训练损失/代价函数中添加一个附加项，称为正则项，这里用  $R(\theta)$  表示：

$$J_\lambda(\theta) = J(\theta) + \lambda R(\theta) \quad (9.1)$$

这里， $J_\lambda$  通常被称为正则化损失， $\lambda \geq 0$  被称为正则化参数。正则项  $R(\theta)$ （在几乎所有情况下）是一个非负函数。在经典方法中， $R(\theta)$  纯粹是参数  $\theta$  的函数，但一些现代方法允许  $R(\theta)$  依赖于训练数据集。<sup>1</sup>

正则项  $R(\theta)$  通常被选择为衡量模型  $\theta$  复杂度的某种度量。因此，在使用正则化损失时，我们的目标是找到一个既能很好地拟合数据（即具有小的损失  $J(\theta)$ ）又具有小的模型复杂度（即小的  $R(\theta)$ ）的模型。这两个目标之间的平衡由正则化参数  $\lambda$  控制。当  $\lambda = 0$  时，正则化损失等同于原始损失。当  $\lambda$  是一个足够小的正数时，最小化正则化损失也可以有效地最小化原始损失，同时利用正则项来区分那些原始损失相同的解。当正则项非常大时，原始损失不再有效（并且模型很可能具有很大的偏差）。

最常用的正则化可能是  $\ell_2$  正则化，其中  $R(\theta) = \frac{1}{2} \|\theta\|_2^2$ 。它鼓励优化器找到一个  $\ell_2$  范数较小的模型。在深度学习中，这通常被称为权重衰减（weight decay），因为在正则化损失上进行学习率为  $\eta$  的梯度下降，等价于将  $\theta$  乘以一个标量因

---

<sup>1</sup>为了简洁起见，这里的记号省略了对训练数据集的依赖性—— $J(\theta)$  显然需要依赖于训练数据集。

子  $1 - \eta\lambda$  收缩/衰减后，使用标准梯度：

$$\begin{aligned}\theta &\leftarrow \theta - \eta \nabla J_\lambda(\theta) = \theta - \eta \lambda \theta - \eta \nabla J(\theta) \\ &= \underbrace{(1 - \eta \lambda)\theta}_{\text{权重衰减}} - \eta \nabla J(\theta)\end{aligned}\tag{9.2}$$

除了鼓励更简单的模型之外，正则化还可以对模型参数施加归纳偏置或结构。例如，假设我们先验地认为真实模型参数中非零项的数量很少<sup>2</sup>——这通常被称为模型的稀疏性——就可以对  $\theta$  的非零项数量施加正则化，记作  $\|\theta\|_0$ ，从而利用这种先验。施加额外的参数结构会缩小我们的搜索空间，并使模型族的复杂度变小——例如，稀疏模型族可以被认为比所有模型族具有更低的复杂度——因此往往会有更好的泛化能力。另一方面，强烈施加额外的结构可能会增加偏差的风险。例如，如果强烈地对稀疏性进行正则化，但是稀疏模型不能准确地预测，我们将会遭受很大的偏差（类似于第 8.1 节中使用线性模型去学习由二次函数表示的数据的情况）。

参数的稀疏性不是参数的连续函数，因此无法通过（随机）梯度下降对其进行优化。一个常见的松弛方法是使用  $R(\theta) = \|\theta\|_1$  作为替代。<sup>3</sup>

对于线性模型， $R(\theta) = \|\theta\|_1$ （也称为 LASSO）和  $R(\theta) = \frac{1}{2}\|\theta\|_2^2$  可能是最常用的正则化项。其他范数和范数的幂有时也会被使用。 $\ell_2$  范数正则化在核方法中更为常用，因为  $\ell_1$  正则化通常与核技巧不兼容（最优解不能写成特征内积的函数）。

在深度学习中，最常用的正则化项是  $\ell_2$  正则化或权重衰减。其他常见的正则化方法包括 dropout、数据增强、对权重矩阵的谱范数进行正则化以及对模型的 Lipschitz 性进行正则化等。深度学习中的正则化是一个活跃的研究领域，并且我们还知道还有一个隐式的正则化来源，这将在下一节中讨论。

## 9.2 隐式正则化效应（选读）

优化器的隐式正则化效应，或者说隐式偏置或算法正则化，是在深度学习时代观察到的一个新概念/现象。它主要指的是优化器可以在正则化损失所施加的结构之外，隐式地对参数施加结构。

---

<sup>2</sup>对于线性模型，这意味着模型只使用输入的少数几个坐标来做出准确的预测。

<sup>3</sup>存在丰富的理论工作解释了为什么  $\|\theta\|_1$  是稀疏性的良好替代，但这超出了本课程的范围。直观地说：假设参数在单位球面上，具有最小  $\ell_1$  范数的参数也碰巧是只有 1 个非零坐标的最稀疏参数。因此，在某种程度上，稀疏性和  $\ell_1$  范数给出了相同的极值点。

在大多数经典设置中，损失或正则化损失具有唯一的全局最小值，因此任何合理的优化器都应该收敛到该全局最小值，并且不施加任何额外的偏好。然而，在深度学习中，通常损失或正则化损失具有多个（近似）全局最小值，不同的优化器可能收敛到不同的全局最小值。尽管这些全局最小值具有相同或相似的训练损失，但它们的性质可能不同，并且泛化性能也可能显著不同。参见图 9.1 和 9.2 及其说明文字，以了解插图和一些实验结果。例如，某个全局最小值可能比其他全局最小值产生更具 Lipschitz 性或更稀疏的模型，从而具有更好的测试误差。结果表明，许多常用的优化器（或其组成部分）倾向于或偏向于寻找具有某些特性的全局最小值，从而带来更好的测试性能。

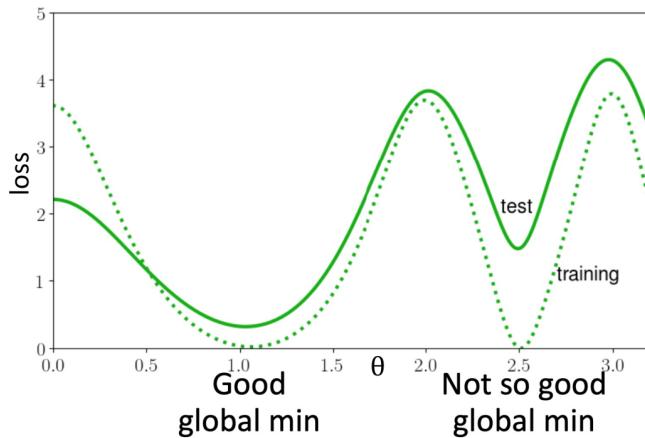


图 9.1: 在训练损失上的全局最小点的测试性能不相同的图示。

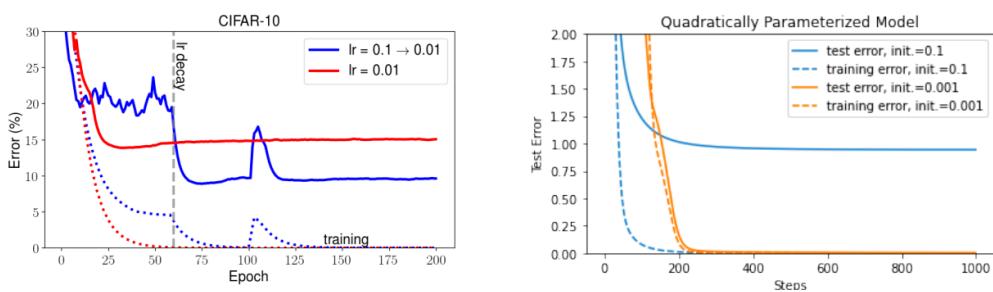


图 9.2: 左：在 CIFAR-10 数据集上，使用两种不同学习率调度方案训练的神经网络的性能。尽管两个实验使用了完全相同的正则化损失，并且优化器完美地拟合了训练数据，但模型的泛化性能差异很大。右：在另一个合成数据集上，使用不同初始化的优化器具有相同的训练误差，但泛化性能不同。<sup>4</sup>

<sup>4</sup>设置与 Woodworth et al. 2020 和 HaoChen et al. 2020 一致。

总而言之，所要传达的关键信息是，优化器的选择不仅影响训练损失的最小化，还施加了隐式正则化并影响模型的泛化能力。即使当前的优化器已经完美地收敛到较小的训练误差，也可能仍然需要调整优化器以获得更好的泛化性能。

人们可能会想知道优化器的哪些组成部分偏向于哪种类型的全局最小值，以及哪种类型的全局最小值可能具有更好的泛化能力。这些是研究人员正在积极探索的开放性问题。经验和理论研究已经提供了一些线索和启发式方法。在许多（但不是全部）情况下，在那些优化能够成功最小化训练损失的设置中，使用较大的初始学习率、较小的初始化、较小的批量大小和动量似乎有助于偏向于更具泛化能力的解。一个猜想（在某些简化情况下可以证明）是，优化过程中的随机性有助于优化器找到更平坦的全局最小值（损失曲率较小的全局最小值），而平坦的全局最小值往往会产生更具 Lipschitz 性和更好泛化能力的模型。形式化地刻画隐式正则化效应仍然是一个具有挑战性的开放性研究问题。

### 9.3 通过交叉验证选择模型

假设我们正在为一个学习问题选择几种不同的模型。例如，我们可能正在使用多项式回归模型  $h_\theta(x) = g(\theta_0 + \theta_1x + \theta_2x^2 + \dots + \theta_kx^k)$ ，并希望决定  $k$  应该取 0, 1, ..., 或 10 中的那个值。我们如何自动选择一个模型，使其在偏差和方差这对孪生罪恶之间取得良好权衡？<sup>5</sup> 另外，假设我们想自动选择局部加权回归的带宽参数  $\tau$ ，或者  $\ell_1$  正则化 SVM 的参数  $C$ ，该如何做？

为了具体起见，在这些笔记中，我们假设我们有一些有限的模型集合  $\mathcal{M} = \{M_1, \dots, M_d\}$ ，我们正尝试从中进行选择。例如，在上面的第一个例子中，模型  $M_i$  将是一个  $i$  次多项式回归模型。（推广到无限  $\mathcal{M}$  并不难。<sup>6</sup>）另外，如果我们正在决定是使用 SVM、神经网络还是逻辑回归，那么  $\mathcal{M}$  可能包含这些模型。

#### 交叉验证

假设我们像往常一样，给定一个训练集  $S$ 。考虑到我们对经验风险最小化的了解，以下是基于经验风险最小化进行模型选择的一种初步算法：

---

<sup>5</sup> 考虑到我们在之前的笔记中说过偏差和方差是截然不同的两种“野兽”，有些读者可能会想知道我们在这里是否应该称它们为“孪生”罪恶。也许最好将它们视为非同卵双胞胎。不过“偏差和方差这对异卵孪生罪恶”这个短语听起来不太顺口。

<sup>6</sup> 如果我们正在从无限的模型集合中进行选择，例如对应于带宽  $\tau \in \mathbb{R}^+$  的所有可能值，我们可以对  $\tau$  进行离散化，只考虑其有限个可能值。更普遍地，这里描述的大多数算法都可以看作是在模型空间中执行优化搜索，而且我们也可以在无限模型类别上执行这种搜索。

1. 在  $S$  上训练每个模型  $M_i$ , 得到假设  $h_i$ 。
2. 选择训练误差最小的假设。

这个算法不 (*not*) 起作用。考虑选择多项式的幂次。多项式的幂次越高, 它就越能更好地拟合训练集  $S$ , 从而训练误差越低。因此, 这种方法总是会选择高方差、高幂次的多项式模型, 正如之前所述, 这通常是一个糟糕的选择。

这里有一个效果更好的算法: 叫做留出交叉验证 (**hold-out cross validation**) (也称为简单交叉验证 (**simple cross validation**)) 中, 算法执行以下步骤:

1. 随机将  $S$  分割成  $S_{\text{train}}$  (例如, 70% 的数据) 和  $S_{\text{cv}}$  (剩余的 30%)。这里的  $S_{\text{cv}}$  称为留出交叉验证集。
2. 仅在  $S_{\text{train}}$  上训练每个模型  $M_i$ , 得到一些假设  $h_i$ 。
3. 选择并输出在留出交叉验证集上误差  $\hat{\varepsilon}_{S_{\text{cv}}}(h_i)$  最小的假设  $h_i$ 。(这里  $\hat{\varepsilon}_{S_{\text{cv}}}(h)$  表示假设  $h$  在  $S_{\text{cv}}$  中的例子上的平均误差。) 留出集上的误差也称为验证误差。

通过在模型未训练过的例子集合  $S_{\text{cv}}$  上进行测试/验证, 我们获得了每个假设  $h_i$  的真实泛化/测试误差的更好估计。因此, 这种方法本质上是选择具有最小估计泛化/测试误差的模型。验证集的大小取决于可用例子的总数。通常, 留出交叉验证中使用的验证集数据量占总数据量的 1/4 到 1/3, 其中 30% 是一个典型的选择。然而, 当总数据集非常大时, 只要验证例子的绝对数量足够多, 验证集可以占总例子的一小部分。例如, ImageNet 数据集有大约 1M 训练图像, 验证集有时设置为 50K 图像, 仅占总例子的约 5%。

可选地, 算法中的步骤 3 也可以替换为根据  $\arg \min_i \hat{\varepsilon}_{S_{\text{cv}}}(h_i)$  选择模型  $M_i$ , 然后使用整个训练集  $S$  重新训练  $M_i$ 。(这通常是一个好主意, 除了一个例外, 即对初始条件和/或数据的扰动很敏感的学习算法。对于这些方法, 模型  $M_i$  在  $S_{\text{train}}$  上表现良好并不一定意味着它在  $S_{\text{cv}}$  上也会表现良好, 并且最好放弃重新训练这一步。)

使用留出交叉验证的缺点是它“浪费”了大约 30% 的数据。就算我们选择在整个训练集上重新训练, 但如果我们试图寻找一个性能良好的模型, 却只有  $0.7n$  个训练样本而不是完整的  $n$  个训练样本, 那么这种“浪费”依然存在。虽然在数据丰富廉价的情况下这很好, 但在数据稀缺的情况下 (比如  $n = 20$  时), 我们希望采取更有效的方法。

这里有一种称为 *k* 折交叉验证 (*k*-fold cross validation) 的方法，它每次保留更少的数据：

1. 随机将  $S$  分割成  $k$  个不相交的子集，每个子集包含  $m/k$  个训练样本。我们将这些子集称为  $S_1, \dots, S_k$ 。
2. 对于每个模型  $M_i$ ，我们按如下方式评估它：

对于  $j = 1, \dots, k$

在  $S_1 \cup \dots \cup S_{j-1} \cup S_{j+1} \cup \dots \cup S_k$  上训练模型  $M_i$ （即，在除  $S_j$  外的所有数据上训练），得到一些假设  $h_{ij}$ 。

在  $S_j$  上测试假设  $h_{ij}$ ，得到  $\hat{\varepsilon}_{S_j}(h_{ij})$ 。

然后，模型  $M_i$  的估计泛化误差计算为  $\hat{\varepsilon}_{S_j}(h_{ij})$  的平均值（对  $j$  取平均）。

3. 选择估计泛化误差最低的模型  $M_i$ ，并在整个训练集  $S$  上重新训练该模型。得到的假设作为最终输出。

这里通常选择的折数是  $k = 10$ 。每次保留的数据比例现在是  $1/k$ ，比之前小得多——这个过程也可能比留出交叉验证的计算成本更高，因为现在需要对每个模型训练  $k$  次。

虽然  $k = 10$  是一个常用的选择，但在数据非常稀缺的问题中，有时我们会使用极端的选择  $k = m$ ，以便每次保留尽可能少的数据。在这种设置下，我们会在  $S$  中除了一个训练例子之外的所有例子上重复训练，并在那个保留的例子上进行测试。然后将得到的  $m = k$  个误差平均起来，以获得模型泛化误差的估计。这种方法也有自己的名字，由于我们每次只保留一个训练样本，这种方法称为留一交叉验证 (leave-one-out cross validation)。

最后，虽然我们把这些交叉验证当作选择模型的方法，它们也可以用于评估单个 (*single*) 模型或算法。例如，如果您实现了一些学习算法并想估计它在您的应用程序中的表现如何（或者如果您发明了一种新的学习算法并想在技术论文中报告它在各种测试集上的表现），交叉验证是一种合理的方法。

## 9.4 贝叶斯统计与正则化

本节我们将讨论对抗过拟合的另一个工具。

之前我们讨论了使用最大似然估计 (MLE) 进行参数拟合，并根据以下公式选择参数：

$$\theta_{\text{MLE}} = \arg \max_{\theta} \prod_{i=1}^n p(y^{(i)}|x^{(i)}; \theta).$$

在随后的讨论中，我们将  $\theta$  视为一个未知参数。这种将  $\theta$  视为常数但未知 (*constant-valued but unknown*) 的观点在**频率派 (frequentist)** 统计学中被采用。在频率派观点中， $\theta$  不是随机的——它只是未知的——而我们的任务是利用统计程序（例如最大似然）来估计这个参数。

解决参数估计问题的另一种方法是采用**贝叶斯 (Bayesian)** 的世界观，并将  $\theta$  视为一个随机变量 (*random variable*)，其值是未知的。在这种方法中，我们会指定一个**先验分布 (prior distribution)**  $p(\theta)$ ，它表达了我们对参数的“先验观念”。给定一个训练集  $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ ，当被要求对一个新的  $x$  值进行预测时，我们可以计算参数的后验分布

$$\begin{aligned} p(\theta|S) &= \frac{p(S|\theta)p(\theta)}{p(S)} \\ &= \frac{(\prod_{i=1}^n p(y^{(i)}|x^{(i)}, \theta))p(\theta)}{\int_{\theta} (\prod_{i=1}^n p(y^{(i)}|x^{(i)}, \theta))p(\theta)d\theta} \end{aligned} \quad (9.3)$$

在上述公式 (9.3) 中， $p(y^{(i)}|x^{(i)}, \theta)$  可以来自所研究的学习问题所采用的任何模型。例如，如果使用贝叶斯逻辑回归，那么可能会选择  $p(y^{(i)}|x^{(i)}, \theta) = h_{\theta}(x^{(i)})^{y^{(i)}}(1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}}$ ，其中  $h_{\theta}(x^{(i)}) = 1/(1 + \exp(-\theta^T x^{(i)}))$ 。<sup>7</sup>

当给定一个新的测试样本  $x$  并要求对其进行预测时，可以使用关于  $\theta$  的后验分布计算类别标签上的后验分布：

$$p(y|x, S) = \int_{\theta} p(y|x, \theta)p(\theta|S)d\theta \quad (9.4)$$

在上述公式 (9.4) 中， $p(\theta|S)$  来自公式 (9.3)。因此，例如，如果目标是预测给定  $x$  时  $y$  的期望值，那么有<sup>8</sup>

$$E[y|x, S] = \int_y y p(y|x, S) dy$$

所概述的这个过程可以被认为是进行“完全贝叶斯”预测，其中模型的预测是通过对  $\theta$  的后验分布  $p(\theta|S)$  求平均来计算的。不幸的是，通常情况下计算这

---

<sup>7</sup>因为现在  $\theta$  被视为一个随机变量，所以将它作为条件是可以的，写作  $p(y|x, \theta)$  而不是  $p(y|x; \theta)$ 。

<sup>8</sup>如果  $y$  是离散值，下面的积分将被求和代替。

个后验分布的计算成本非常高。这是因为这需要对公式 (9.3) 中的（通常是高维的） $\theta$  进行积分，而这通常无法得到闭式解。

因此，实践中将转而近似  $\theta$  的后验分布。一种常见的近似方法是用一个单点估计来代替  $\theta$  的后验分布（如公式 (9.4) 中所示）。 $\theta$  的最大后验 (**maximum a posteriori, MAP**) 估计由下式给出：

$$\theta_{\text{MAP}} = \arg \max_{\theta} \prod_{i=1}^n p(y^{(i)}|x^{(i)}, \theta)p(\theta). \quad (9.5)$$

注意，这与  $\theta$  的最大似然估计 (MLE) 的公式相同，只是在末尾多了一个先验项  $p(\theta)$ 。

在实际应用中，先验  $p(\theta)$  的一个常见选择是假设  $\theta \sim \mathcal{N}(0, \tau^2 I)$ 。使用这种先验选择，拟合的参数  $\theta_{\text{MAP}}$  的范数将小于通过最大似然选择的范数。在实践中，这使得贝叶斯 MAP 估计对过拟合不那么敏感，优于参数的最大似然估计。例如，贝叶斯逻辑回归被证明是文本分类的有效算法，即使在文本分类中通常有  $d \gg n$ 。

# **第四部分**

## **无监督学习**

# 第 10 章 聚类与 k-means 算法

在聚类问题中，给定一个训练集  $\{x^{(1)}, \dots, x^{(n)}\}$ ，我们希望将数据分组到一些具有汇聚效果的“簇”中。这里， $x^{(i)} \in \mathbb{R}^d$  像往常一样，但没有给出标签  $y^{(i)}$ 。因此，这是一个无监督学习问题。

$k$ -means 聚类算法如下：

1. 随机初始化簇中心 (cluster centroids)  $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^d$ 。
2. 重复直到收敛：{

对于每个  $i$ ，设置

$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

对于每个  $j$ ，设置

$$\mu_j := \frac{\sum_{i=1}^n 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^n 1\{c^{(i)} = j\}}.$$

}

在上述算法中， $k$ （算法的一个参数）是我们想要找到的簇的数量；而簇中心  $\mu_j$  代表我们对簇中心位置的当前猜测。为了初始化簇中心（在上述算法的步骤 1 中），我们可以随机选择  $k$  个训练样本，并将簇中心设置为这些  $k$  个样本的值。（也可以采用其他初始化方法。）

算法的内循环重复执行两个步骤：(i) 将每个训练样本  $x^{(i)}$  “分配”给最近的簇中心  $\mu_j$ ；(ii) 将每个簇中心  $\mu_j$  移动到分配给它的点的均值。图 10.1 展示了  $k$ -means 算法运行的示意图。

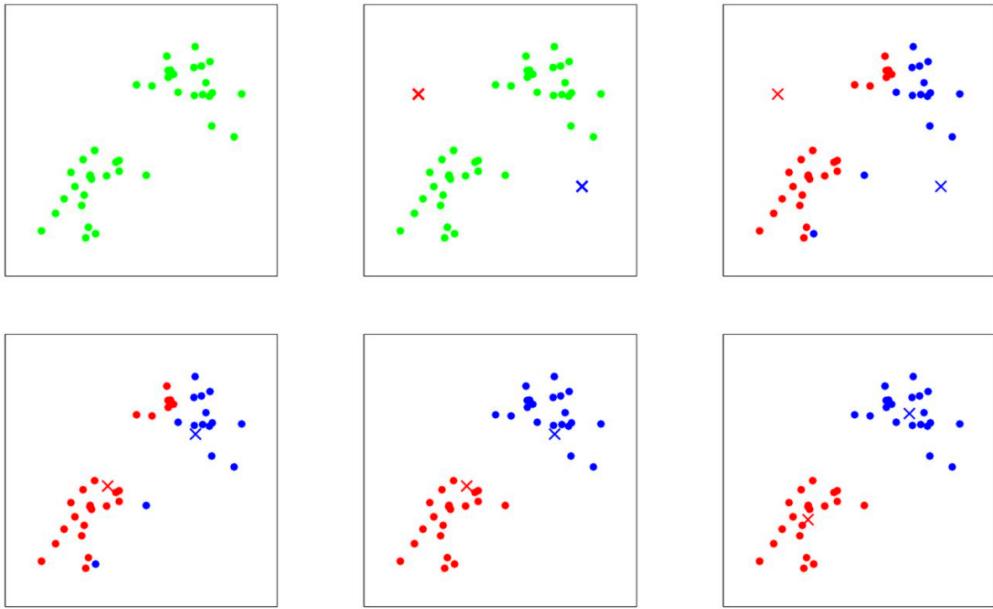


图 10.1: K-means 算法。训练样本显示为点，簇中心显示为叉。(a) 原始数据集。(b) 随机初始簇中心（在此示例中，不是选择了两个训练样本作为簇中心）。(c-f) 演示运行两次 k-means 迭代。每次迭代将每个训练样本分配给最近的簇中心（通过将训练样本“涂成”与其分配到的簇中心相同颜色来显示）；然后我们将每个簇中心移动到分配给它的点的均值。（彩色观看效果最佳。）图片由 Michael Jordan 提供。

*k*-means 算法是否保证收敛？是的，在某种意义上。特别地，定义失真函数（distortion function）为：

$$J(c, \mu) = \sum_{i=1}^n \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

因此， $J$  度量了每个训练样本  $x^{(i)}$  与其被分配到的簇中心  $\mu_{c^{(i)}}$  之间的平方距离之和。可以证明，*k*-means 正是  $J$  上的坐标下降。具体来说，*k*-means 的内循环在固定  $\mu$  的同时重复最小化  $J$  关于  $c$ ，然后在固定  $c$  的同时最小化  $J$  关于  $\mu$ 。因此， $J$  必须单调递减，并且  $J$  的值必须收敛。（通常，这意味着  $c$  和  $\mu$  也会收敛。理论上，*k*-means 算法可能会在几个具有完全相同的  $J$  值的不同聚类之间振荡，即  $c$  和/或  $\mu$  具有几个不同的值，但这在实践中几乎从未发生过。）失真函数  $J$  是非凸的，因此在  $J$  上的坐标下降不保证收敛到全局最小值。换句话说，*k*-means 容易受到局部最优的影响。尽管如此，*k*-means 通常工作良好并能得到很好的聚类结果。但是，如果担心陷入糟糕的局部极小值，常见的做法是多次运行 *k*-means（使用不同的随机初始簇中心  $\mu_j$ ）。然后，在所有找到的不同聚类中，

选择失真  $J(c, \mu)$  最低的那一个。

# 第 11 章 EM 算法

本章将讨论用于密度估计的 EM（期望最大化）算法。

## 11.1 面向高斯混合模型的 EM 算法

假设像往常一样给定一个训练集  $\{x^{(1)}, \dots, x^{(n)}\}$ 。由于处于无监督学习设置中，这些点没有标签。

我们希望通过指定联合分布  $p(x^{(i)}, z^{(i)}) = p(x^{(i)}|z^{(i)})p(z^{(i)})$  来建模数据。这里， $z^{(i)} \sim \text{Multinomial}(\phi)$  (其中  $\phi_j \geq 0, \sum_{j=1}^k \phi_j = 1$ , 参数  $\phi_j$  代表了  $p(z^{(i)} = j)$  的值)，并且  $x^{(i)}|z^{(i)} = j \sim \mathcal{N}(\mu_j, \Sigma_j)$ 。令  $k$  表示  $z^{(i)}$  可以取的值的数量。因此，我们的模型假设每个  $x^{(i)}$  是通过从  $\{1, \dots, k\}$  中随机选择  $z^{(i)}$  生成的，然后  $x^{(i)}$  是根据  $z^{(i)}$  从  $k$  个高斯分布中的一个中抽取的。这被称为**高斯混合 (mixture of Gaussians)** 模型。另请注意  $z^{(i)}$  是**隐 (latent)** 变量，这意味着它们是隐藏/未观察到的。这使得我们的估计问题变得困难。

因此，模型的参数是  $\phi, \mu$  和  $\Sigma$ 。为了估计它们，可以写出数据的似然函数：

$$\begin{aligned}\ell(\phi, \mu, \Sigma) &= \sum_{i=1}^n \log p(x^{(i)}; \phi, \mu, \Sigma) \\ &= \sum_{i=1}^n \log \sum_{z^{(i)}=1}^k p(x^{(i)}|z^{(i)}; \mu, \Sigma)p(z^{(i)}; \phi).\end{aligned}$$

然而，如果我们将此公式里的参数的导数设为 0 并尝试求解，会发现无法找到参数的闭式最大似然估计。(可以自行尝试。)

随机变量  $z^{(i)}$  表示每个  $x^{(i)}$  来自  $k$  个高斯分布中的哪一个。注意，如果已知  $z^{(i)}$  的值，最大似然问题是很容易的。具体来说，可以写出似然函数为

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^n \log p(x^{(i)}|z^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi).$$

对上式关于  $\phi, \mu$  和  $\Sigma$  求最大值，得到参数：

$$\begin{aligned}\phi_j &= \frac{1}{n} \sum_{i=1}^n 1\{z^{(i)} = j\}, \\ \mu_j &= \frac{\sum_{i=1}^n 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^n 1\{z^{(i)} = j\}}, \\ \Sigma_j &= \frac{\sum_{i=1}^n 1\{z^{(i)} = j\} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^n 1\{z^{(i)} = j\}}.\end{aligned}$$

确实，可以看到，如果已知  $z^{(i)}$  的值，那么最大似然估计几乎与估计高斯判别分析模型的参数完全相同，不同之处在于这里  $z^{(i)}$  扮演着类别标签的角色。<sup>1</sup>

但是在密度估计问题中， $z^{(i)}$  的值是未知的。该怎么办？

EM 算法是一种迭代算法，它有两个主要步骤。在步骤 E 中，它尝试“猜测” $z^{(i)}$  的值。在步骤 M 中，它根据我们的猜测更新模型的参数。由于在步骤 M 中假装第一部分的猜测是正确的，最大化就变得容易了。以下是算法：

重复直到收敛：{

(步骤 E) 对于每个  $i, j$ ，设置

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(步骤 M) 更新参数：

$$\begin{aligned}\phi_j &:= \frac{1}{n} \sum_{i=1}^n w_j^{(i)}, \\ \mu_j &:= \frac{\sum_{i=1}^n w_j^{(i)} x^{(i)}}{\sum_{i=1}^n w_j^{(i)}}, \\ \Sigma_j &:= \frac{\sum_{i=1}^n w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^n w_j^{(i)}}\end{aligned}$$

}

在步骤 E 中，根据给定的  $x^{(i)}$  和当前参数设置，计算参数  $z^{(i)}$  的后验概率。也就是说，使用贝叶斯法则，得到：

$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^k p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

---

<sup>1</sup>这里的公式与在习题集 1 中高斯判别分析的公式有一些细微差别，首先是因为我们将  $z^{(i)}$  推广为多项分布而不是伯努利分布，其次是因为这里我们对每个高斯分布使用不同的  $\Sigma_j$ 。

其中,  $p(x^{(i)}|z^{(i)} = j; \mu, \Sigma)$  是在  $x^{(i)}$  处计算均值为  $\mu_j$ 、协方差为  $\Sigma_j$  的高斯分布的密度函数;  $p(z^{(i)} = j; \phi)$  由  $\phi_j$  给出, 依此类推。在步骤 E 中计算的  $w_j^{(i)}$  值代表了对  $z^{(i)}$  值的“软”猜测。<sup>2</sup>

此外, 应该将步骤 M 中的更新公式与已知  $z^{(i)}$  值时的公式进行对比。它们是相同的, 只不过前面使用指示函数  $1\{z^{(i)} = j\}$  表示每个数据点来自哪个高斯分布, 而这里使用  $w_j^{(i)}$ 。

EM 算法也让人联想到 K-means 聚类算法, 不同之处在于 K-means 使用“硬”聚类分配  $c(i)$ , 而这里使用“软”分配  $w_j^{(i)}$ 。与 K-means 类似, EM 算法也容易陷入局部最优, 因此使用几个不同的初始参数进行重新初始化可能是个好主意。

显然, EM 算法对重复猜测未知  $z^{(i)}$  值具有非常自然的解释; 但是它是如何得出的? 我们能对其做出任何保证吗, 例如关于其收敛性? 后面我们将更一般地描述 EM 算法, 使得该算法更容易应用于其他包含隐变量的估计问题, 并使其收敛性得到保证。

## 11.2 Jensen 不等式

我们从一个非常有用的结果开始讨论, 即 **Jensen 不等式 (Jensen's inequality)**。

设  $f$  是一个定义域为实数集的函数。回想一下, 当  $f''(x) \geq 0$  时  $f$  是一个凸函数 (对于所有  $x \in \mathbb{R}$ )。对于取向量值输入的函数, 这推广为其 Hessian 矩阵  $H$  是半正定 ( $H \geq 0$ ) 的条件。如果对于所有  $x$ , 都有  $f''(x) > 0$ , 则称  $f$  为**严格 (strictly)** 凸函数 (在向量值情况下, 相应的表述是  $H$  必须是正定的, 记作  $H > 0$ )。Jensen 不等式可以表述如下:

**定理.** 设  $f$  是一个凸函数, 且  $X$  是一个随机变量。则:

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}X).$$

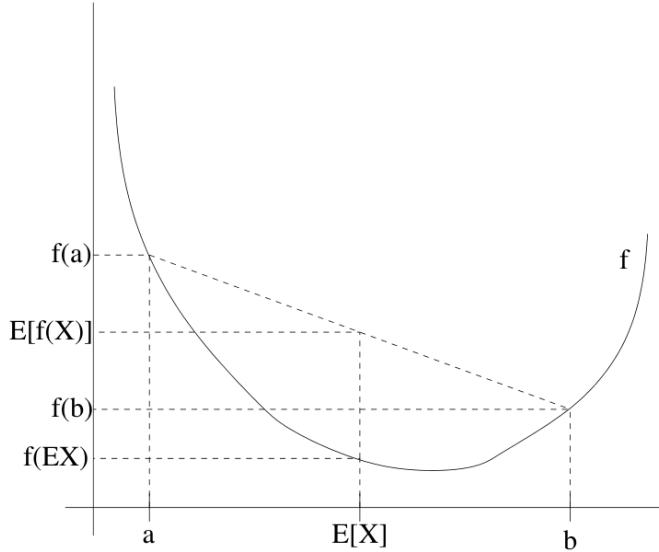
此外, 如果  $f$  是严格凸函数, 则  $\mathbb{E}[f(X)] = f(\mathbb{E}X)$  当且仅当  $X = \mathbb{E}X$  以概率 1 成立 (即  $X$  是一个常数)。

回想一下, 在书写期望时, 有时会省略括号, 因此在上述定理中,  $f(\mathbb{E}X) = f(\mathbb{E}[X])$ 。

---

<sup>2</sup> “软”指的是猜测是概率且取值在  $[0, 1]$  中; 作为对比, “硬”表示单一的最佳猜测 (例如取值在  $\{0, 1\}$  或  $\{1, \dots, k\}$  中)。

对于该定理的解释，请参考下图。



此处， $f$  是由实线表示的凸函数。 $X$  是一个随机变量，其取值  $a$  的概率为 0.5，取值  $b$  的概率为 0.5（在  $x$  轴上标示）。因此， $X$  的期望值是  $a$  和  $b$  的中点。

还可以在  $y$  轴上看到  $f(a)$ 、 $f(b)$  和  $f(E[X])$  的值。此外，值  $E[f(X)]$  现在是  $y$  轴上  $f(a)$  和  $f(b)$  之间的中点。从示例中，我们看到因为  $f$  是凸函数，所以必然有  $E[f(X)] \geq f(E[X])$ 。

顺便说一句，很多人在记忆不等式的方向时会遇到困难，记住这个图是一个快速得出答案的好方法。

**备注.** 回想一下， $f$  是（严格）凹函数当且仅当  $-f$  是（严格）凸函数（即  $f''(x) \leq 0$  或  $H \leq 0$ ）。Jensen 不等式对于凹函数  $f$  也成立，但所有不等号的方向都反转了（如  $E[f(X)] \leq f(E[X])$ ）。

### 11.3 广义 EM 算法

假设有一个估计问题，其包含一个有  $n$  个独立样本的训练集  $\{x^{(1)}, \dots, x^{(n)}\}$ 。我们有一个隐变量模型  $p(x, z; \theta)$ ，其中  $z$  是隐变量（为简单起见，假设其取有限个值）。 $x$  的密度可以通过对隐变量  $z$  进行边缘化得到：

$$p(x; \theta) = \sum_z p(x, z; \theta) \quad (11.1)$$

我们希望通过最大化数据的对数似然来拟合参数  $\theta$ ，对数似然定义为：

$$\ell(\theta) = \sum_{i=1}^n \log p(x^{(i)}; \theta) \quad (11.2)$$

可以将目标函数写成联合密度  $p(x, z; \theta)$  的形式：

$$\ell(\theta) = \sum_{i=1}^n \log p(x^{(i)}; \theta) \quad (11.3)$$

$$= \sum_{i=1}^n \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta). \quad (11.4)$$

但是，显式地找到参数  $\theta$  的最大似然估计可能很困难，因为它会导致困难的非凸优化问题。<sup>3</sup>这里， $z^{(i)}$  是隐随机变量；通常情况下，如果  $z^{(i)}$  被观测到，那么最大似然估计就会变得容易。

在这种设置下，EM 算法提供了一种有效的方法。显式地最大化  $\ell(\theta)$  可能很困难，因此我们的策略是反复构造一个  $\ell(\theta)$  的下界（步骤 E），然后优化该下界（步骤 M）。<sup>4</sup>

事实证明，这里的求和  $\sum_{i=1}^n$  并非本质的。为了更简单地阐述 EM 算法，首先考虑对单个样本 (a single example)  $x$  的似然  $\log p(x)$  进行优化。在推导出优化  $\log p(x)$  的算法后，我们将通过将相关方程相加来将其转换为适用于  $n$  个样本的算法。因此，现在我们要优化  $\log p(x; \theta)$ ，它可以重写为

$$\log p(x; \theta) = \log \sum_z p(x, z; \theta) \quad (11.5)$$

令  $Q$  是  $z$  可能取值上的一个分布。也就是说， $\sum_z Q(z) = 1$ ，且  $Q(z) \geq 0$ 。

考虑以下推导：<sup>5</sup>

$$\begin{aligned} \log p(x; \theta) &= \log \sum_z p(x, z; \theta) \\ &= \log \sum_z Q(z) \frac{p(x, z; \theta)}{Q(z)} \end{aligned} \quad (11.6)$$

$$\geq \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)} \quad (11.7)$$

上述推导的最后一步使用了 Jensen 不等式。具体来说， $f(x) = \log x$  是一个凹函数，因为在其定义域  $x \in \mathbb{R}^+$  上  $f''(x) = -1/x^2 < 0$ 。此外，求和项

$$\sum_z Q(z) \left[ \frac{p(x, z; \theta)}{Q(z)} \right]$$

<sup>3</sup>这个优化问题难以优化主要是一种经验性的观察。

<sup>4</sup>根据经验，步骤 E 和步骤 M 通常比直接优化函数  $\ell(\cdot)$  更高效。然而，这并不一定意味着交替执行这两个步骤总能收敛到  $\ell(\cdot)$  的全局最优解。即使对于高斯混合模型，EM 算法也可能收敛到全局最优解或陷入局部最优，这取决于训练数据的性质。根据经验，对于实际数据，EM 通常可以收敛到具有相对高似然的解（即使不是最优解），其背后的理论很大程度上尚未被完全理解。

<sup>5</sup>如果  $z$  是连续变量，则  $Q$  是一个概率密度函数，对  $z$  的求和也变为对  $z$  的积分。

是  $[p(x, z; \theta)/Q(z)]$  关于从分布  $Q$  抽取的  $z$  的期望。<sup>6</sup> 根据 Jensen 不等式，有

$$f\left(\mathbb{E}_{z \sim Q}\left[\frac{p(x, z; \theta)}{Q(z)}\right]\right) \geq \mathbb{E}_{z \sim Q}\left[f\left(\frac{p(x, z; \theta)}{Q(z)}\right)\right],$$

其中下标标 “ $z \sim Q$ ” 表示期望是关于从  $Q$  中抽取的  $z$  计算的。这使得公式 (11.6) 可以推导到公式 (11.7)。

现在，对于任何分布  $Q$ ，公式 (11.7) 给出了  $\log p(x; \theta)$  的一个下界。对于  $Q$  有许多可能的选择，应该选择哪一个呢？如果对参数  $\theta$  有当前的猜测值，那么自然会尝试使该下界在  $\theta$  的该值处尽可能紧。也就是说，希望在  $\theta$  的特定值处使上述不等式取等号。为了上述推导中涉及 Jensen 不等式的步骤取等号，需要期望作用在一个“常数值”随机变量上。也就是说，要求对于某个不依赖于  $z$  的常数  $c$ ，有

$$\frac{p(x, z; \theta)}{Q(z)} = c$$

这很容易通过选择

$$Q(z) \propto p(x, z; \theta)$$

来实现。实际上，由于  $\sum_z Q(z) = 1$ （因为它是一个分布），这进一步揭示了

$$\begin{aligned} Q(z) &= \frac{p(x, z; \theta)}{\sum_z p(x, z; \theta)} \\ &= \frac{p(x, z; \theta)}{p(x; \theta)} \\ &= p(z|x; \theta) \end{aligned} \tag{11.8}$$

因此，将  $Q$  设置为给定  $x$  和参数  $\theta$  下  $z$  的后验分布。

事实上，当  $Q(z) = p(z|x; \theta)$  时，可以直接验证公式 (11.7) 是一个等式，因为

$$\begin{aligned} \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)} &= \sum_z p(z|x; \theta) \log \frac{p(x, z; \theta)}{p(z|x; \theta)} \\ &= \sum_z p(z|x; \theta) \log \frac{p(x, z; \theta)p(x; \theta)}{p(z|x; \theta)p(x; \theta)} \\ &= \sum_z p(z|x; \theta) \log p(x; \theta) \\ &= \log p(x; \theta) \sum_z p(z|x; \theta) \\ &= \log p(x; \theta) \quad (\text{因为 } \sum_z p(z|x; \theta) = 1) \end{aligned}$$

---

<sup>6</sup>注意到，当  $p(x, z; \theta) \neq 0$  时，表达式  $\frac{p(x, z; \theta)}{Q(z)}$  只有在  $Q(z) \neq 0$  时才有意义。在此，隐含地假设只考虑满足这一性质的  $Q$ 。

为方便起见，将公式 (11.7) 中的表达式称为 **证据下界 (evidence lower bound, ELBO)**，并用下式表示：

$$\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)} \quad (11.9)$$

根据这个等式，可以将公式 (11.7) 重写为

$$\forall Q, \theta, x, \quad \log p(x; \theta) \geq \text{ELBO}(x; Q, \theta) \quad (11.10)$$

直观上，EM 算法通过以下方式交替更新  $Q$  和  $\theta$ : a) 根据公式 (11.8) 设置  $Q(z) = p(z|x; \theta)$ ，从而使  $\text{ELBO}(x; Q, \theta) = \log p(x; \theta)$  对于  $x$  和当前的  $\theta$  成立；b) 在固定  $Q$  的选择下，最大化  $\text{ELBO}(x; Q, \theta)$  关于  $\theta$  的值。

回想一下，上述所有讨论都是在优化单个样本  $x$  的对数似然  $\log p(x; \theta)$  的假设下进行的。事实证明，对于多个训练样本，基本思想是相同的，只需要在相关位置对样本进行求和即可。接下来将构建针对多个训练样本的证据下界，并使 EM 算法形式化。

我们有一个训练集  $\{x^{(1)}, \dots, x^{(n)}\}$ 。注意， $Q$  的最优选择是  $p(z|x; \theta)$ ，它取决于特定的样本  $x$ 。因此，在这里将对于每个样本  $x^{(i)}$  引入  $n$  个分布  $Q_1, \dots, Q_n$ ，从而构建证据下界

$$\log p(x^{(i)}; \theta) \geq \text{ELBO}(x^{(i)}; Q_i, \theta) = \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

对所有样本求和，可以得到对数似然的下界

$$\begin{aligned} \ell(\theta) &\geq \sum_i \text{ELBO}(x^{(i)}; Q_i, \theta) \\ &= \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \end{aligned} \quad (11.11)$$

对于任意 (*any*) 一组分布  $Q_1, \dots, Q_n$ ，公式 (11.11) 给出了  $\ell(\theta)$  的一个下界，并且与公式 (11.8) 的论证类似，达到等号的  $Q_i$  满足

$$Q_i(z^{(i)}) = p(z^{(i)}|x^{(i)}; \theta)$$

因此，将  $Q_i$  设置为给定  $x^{(i)}$  和当前参数  $\theta$  设置下  $z^{(i)}$  的后验分布。

现在，对于  $Q_i$  的这种选择，公式 (11.11) 给出了试图最大化的对数似然  $\ell$  的下界。这是步骤 E。算法的步骤 M 将最大化公式 (11.11) 中关于参数的表达式，从而获得  $\theta$  的新设置。重复执行这两个步骤就得到了 EM 算法，其过程如下：

重复直到收敛 {

(步骤 E) 对于每个  $i$ , 设置

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)}; \theta).$$

(步骤 M) 设置

$$\begin{aligned} \theta &:= \arg \max_{\theta} \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i, \theta) \\ &= \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log(p(x^{(i)}, z^{(i)}; \theta)/Q_i(z^{(i)})). \end{aligned} \quad (11.12)$$

}

如何知道这个算法的收敛性呢? 假设  $\theta^{(t)}$  和  $\theta^{(t+1)}$  是 EM 的连续两次迭代的参数。现在将证明  $\ell(\theta^{(t)}) \leq \ell(\theta^{(t+1)})$ , 这表明 EM 总是单调地改进对数似然。证明此结果的关键在于对  $Q_i$  的选择。对  $Q_i$  的选择。具体来说, 在 EM 的迭代中, 如果参数从  $\theta^{(t)}$  开始, 就选择  $Q_i^{(t)}(z^{(i)}) := p(z^{(i)}|x^{(i)}; \theta^{(t)})$ 。之前看到过, 这使得了 Jensen 不等式 (得到公式 (11.11) 的) 的等号成立, 因此

$$\ell(\theta^{(t)}) = \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t)}) \quad (11.13)$$

然后通过最大化上述等式的右侧来获得参数  $\theta^{(t+1)}$ 。因此,

$$\begin{aligned} \ell(\theta^{(t+1)}) &\geq \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t+1)}) \quad (\text{不等式 (11.11) 对所有 } Q \text{ 和 } \theta \text{ 成立}) \\ &\geq \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t)}) \quad (\text{见下文}) \\ &= \ell(\theta^{(t)}) \quad (\text{根据公式 (11.13)}) \end{aligned}$$

其中最后一个不等号源于  $\theta^{(t+1)}$  被明确地设置为

$$\arg \max_{\theta} \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i^{(t)}, \theta)$$

因此, EM 算法可以使似然单调收敛。EM 算法需要运行直到收敛。考虑到刚刚的结果, 一个合理的收敛检验是检查连续迭代之间  $\ell(\theta)$  的增加是否小于某个容差参数, 如果 EM 算法改进  $\ell(\theta)$  的速度很慢, 就声明已经收敛。

**备注.** 如果定义 (通过重载  $\text{ELBO}(\cdot)$ )

$$\text{ELBO}(Q, \theta) = \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i, \theta) = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (11.14)$$

那么从之前的推导可知  $\ell(\theta) \geq \text{ELBO}(Q, \theta)$ 。EM 也可以看作是  $\text{ELBO}(Q, \theta)$  上的交替最大化算法，其中步骤 E 关于  $Q$  最大化（可以自行验证），步骤 M 关于  $\theta$  最大化。

### 11.3.1 ELBO 的另一个解释

如公式 (11.9) 中所定义的，令  $\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)}$ 。ELBO 还有几种其他形式。首先，可以按如下方式重写

$$\begin{aligned} \text{ELBO}(x; Q, \theta) &= \mathbb{E}_{z \sim Q} [\log p(x, z; \theta)] - \mathbb{E}_{z \sim Q} [\log Q(z)] \\ &= \mathbb{E}_{z \sim Q} [\log p(x|z; \theta)] - D_{KL}(Q||p_z) \end{aligned} \quad (11.15)$$

其中  $p_z$  表示  $z$  的边际分布（在分布  $p(x, z; \theta)$  下）， $D_{KL}(\cdot||\cdot)$  表示 KL 散度

$$D_{KL}(Q||p_z) = \sum_z Q(z) \log \frac{Q(z)}{p(z)} \quad (11.16)$$

在许多情况下， $z$  的边际分布不依赖于参数  $\theta$ 。这种情况下，可以看到，最大化关于  $\theta$  的 ELBO 等价于最大化公式 (11.15) 中的第一项。这对应于最大化给定  $z$  的  $x$  的条件似然，这通常比原始问题更简单。

ELBO( $\cdot$ ) 的另一种形式是（请自行验证）

$$\text{ELBO}(x; Q, \theta) = \log p(x) - D_{KL}(Q||p_{z|x}) \quad (11.17)$$

其中  $p_{z|x}$  是参数  $\theta$  下给定  $x$  的  $z$  的条件分布。这些形式表明，当  $Q = p_{z|x}$  时，可以获得关于  $Q$  的  $\text{ELBO}(Q, \theta)$  的最大值，这和之前的公式 (11.8) 一致。

## 11.4 回顾高斯混合模型

有了 EM 算法的一般定义，让我们回到之前高斯混合模型中拟合参数  $\phi, \mu$  和  $\Sigma$  的例子。为了简洁起见，我们只推导  $\phi$  和  $\mu_j$  的步骤 M 更新公式，并将  $\Sigma_j$  的更新留作读者的练习。

步骤 E 很简单。按照上面算法的推导，只需计算

$$w_j^{(i)} = Q_i(z^{(i)} = j) = P(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma).$$

这里，“ $Q_i(z^{(i)} = j)$ ” 表示在分布  $Q_i$  下， $z^{(i)}$  取值  $j$  的概率。

接下来，步骤 M 需要最大化关于参数  $\phi, \mu, \Sigma$  的以下量：

$$\begin{aligned} & \sum_{i=1}^n \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)}{Q_i(z^{(i)})} \\ &= \sum_{i=1}^n \sum_{j=1}^k Q_i(z^{(i)} = j) \log \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{Q_i(z^{(i)} = j)} \\ &= \sum_{i=1}^n \sum_{j=1}^k w_j^{(i)} \log \frac{\frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)) \cdot \phi_j}{w_j^{(i)}} \end{aligned}$$

关于  $\mu_l$  最大化此表达式。如果对  $\mu_l$  求导，得到

$$\begin{aligned} & \nabla_{\mu_l} \sum_{i=1}^n \sum_{j=1}^k w_j^{(i)} \log \frac{\frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)) \cdot \phi_j}{w_j^{(i)}} \\ &= -\nabla_{\mu_l} \sum_{i=1}^n \sum_{j=1}^k w_j^{(i)} \frac{1}{2} (x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j) \\ &= \frac{1}{2} \sum_{i=1}^n w_l^{(i)} \nabla_{\mu_l} 2\mu_l^T \Sigma_l^{-1} x^{(i)} - \mu_l^T \Sigma_l^{-1} \mu_l \\ &= \sum_{i=1}^n w_l^{(i)} (\Sigma_l^{-1} x^{(i)} - \Sigma_l^{-1} \mu_l) \end{aligned}$$

令其为零并求解  $\mu_l$ ，得到更新规则

$$\mu_l := \frac{\sum_{i=1}^n w_l^{(i)} x^{(i)}}{\sum_{i=1}^n w_l^{(i)}},$$

这与之前讲义中的结果一致。

再看一个例子，推导参数  $\phi_j$  的步骤 M 更新。将仅依赖于  $\phi_j$  的项组合在一起，可以发现需要最大化

$$\sum_{i=1}^n \sum_{j=1}^k w_j^{(i)} \log \phi_j.$$

然而，由于  $\phi_j$  表示概率  $\phi_j = p(z^{(i)} = j; \phi)$ ，它们需要满足  $\sum_j \phi_j = 1$  的额外约束。为了处理这个约束，构建拉格朗日函数：

$$\mathcal{L}(\phi) = \sum_{i=1}^n \sum_{j=1}^k w_j^{(i)} \log \phi_j + \beta \left( \sum_{j=1}^k \phi_j - 1 \right),$$

其中  $\beta$  是拉格朗日乘子。<sup>7</sup>求导，得到

$$\frac{\partial}{\partial \phi_j} \mathcal{L}(\phi) = \sum_{i=1}^n \frac{w_j^{(i)}}{\phi_j} + \beta.$$

---

<sup>7</sup>无需担心  $\phi_j \geq 0$  的约束，很快会看到，从这个推导中得到的解将自动满足该约束。

令其为零并求解，得到

$$\phi_j = \frac{\sum_{i=1}^n w_j^{(i)}}{-\beta}.$$

即  $\phi_j \propto \sum_{i=1}^n w_j^{(i)}$ 。利用约束  $\sum_j \phi_j = 1$ ，可以得到  $-\beta = \sum_{j=1}^k \sum_{i=1}^n w_j^{(i)} = \sum_{i=1}^n \sum_{j=1}^k w_j^{(i)} = \sum_{i=1}^n 1 = n$ 。（这里利用了  $w_j^{(i)} = Q_i(z^{(i)} = j)$  且概率和为 1，即  $\sum_j w_j^{(i)} = 1$ 。）这样就得到了参数  $\phi_j$  的步骤 M 更新公式：

$$\phi_j := \frac{1}{n} \sum_{i=1}^n w_j^{(i)}.$$

对于  $\Sigma_j$  的步骤 M 更新推导也非常直接。

## 11.5 变分推断与变分自编码器（选读）

广义上讲，变分自编码器（Kingma, and Welling 2013）通常指的是一类将 EM 算法扩展到由神经网络参数化的更复杂模型的算法。它扩展了变分推断技术，并附加了将在下文介绍的“重参数化技巧”。变分自编码器在许多数据集上可能无法提供最佳性能，但它包含了一些关于如何将 EM 算法扩展到具有非线性模型的高维连续隐变量的核心思想。理解它可能会帮助读者理解各种相关的最新论文。

作为示例，我们将考虑通过神经网络对  $p(x, z; \theta)$  进行如下参数化。设  $\theta$  是神经网络  $g(z; \theta)$  的权重集合，该网络将  $z \in \mathbb{R}^k$  映射到  $\mathbb{R}^d$ 。令

$$z \sim \mathcal{N}(0, I_{k \times k}) \tag{11.18}$$

$$x|z \sim \mathcal{N}(g(z; \theta), \sigma^2 I_{d \times d}) \tag{11.19}$$

这里  $I_{k \times k}$  表示维度为  $k \times k$  的单位矩阵， $\sigma$  是一个标量，为了简单起见，我们假设它是已知的。对于第 11.4 节中的高斯混合模型，对于固定的  $\theta$ ， $Q(z) = p(z|x; \theta)$  是  $z$  的后验分布，并且可以解析计算。在许多更复杂的模型（如模型 (11.19)）中，很难精确计算后验分布  $p(z|x; \theta)$ 。

回顾公式 (11.10)，ELBO 总是任何  $Q$  的下界，因此也可以寻求对真实后验分布进行近似（approximation）。通常，人们会使用某种特定形式的  $Q$  来近似真实后验分布。设  $\mathcal{Q}$  是我们正在考虑的  $Q$  的族，目标是在  $\mathcal{Q}$  族中找到一个最接近真实后验分布的  $Q$ 。为了形式化，回顾公式 (11.14) 中定义的作为  $Q$  和  $\theta$  函数的 ELBO 下界：

$$\text{ELBO}(Q, \theta) = \sum_{i=1}^n \text{ELBO}(x^{(i)}; Q_i, \theta) = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

回想一下，EM 算法可以看作是  $\text{ELBO}(Q, \theta)$  的交替最大化。这里，我们转而优化

$$\max_{Q \in \mathcal{Q}} \max_{\theta} \text{ELBO}(Q, \theta) \quad (11.20)$$

现在下一个问题是，什么样的  $Q$  形式（或者对  $Q$  做什么样的结构假设）能够有效地最大化上述目标函数。当隐变量  $z$  是高维离散变量时，一个流行的假设是**平均场假设 (mean field assumption)**，它假设  $Q_i(z)$  可以分解成独立坐标的乘积，换句话说， $Q_i(z) = Q_i^1(z_1) \cdots Q_i^k(z_k)$ 。对于平均场假设使用在面向离散隐变量的学习生成模型中有大量的应用，可以参考 Blei, Kucukelbir, and McAuliffe 2017 的综述，里面介绍了这些模型以及它们在广泛的应用领域产生的影响，包括计算生物学、计算神经科学和社会科学。本节不再深入探讨离散隐变量的情况，主要关注处理连续隐变量，这不仅需要平均场假设，还需要一些额外的技术。

当  $z \in \mathbb{R}^k$  是连续隐变量时，为了优化 (11.20)，需要做出几个决策。首先，由于分布  $Q_i$  涉及无限多个点，需要给出其简洁的表示。一个自然的选取是假设  $Q_i$  是具有一定均值和方差的高斯分布。还希望对所有样本的  $Q_i$  的均值有简洁的表示。注意到  $Q_i(z^{(i)})$  应该近似于  $p(z^{(i)}|x^{(i)}; \theta)$ ，因此将所有  $Q_i$  的均值设为  $x^{(i)}$  的某个函数是合理的。具体来说，设  $q(\cdot; \phi), v(\cdot; \psi)$  是两个将维度  $d$  映射到  $k$  的函数，它们由  $\phi$  和  $\psi$  参数化，假设

$$Q_i = \mathcal{N}(q(x^{(i)}; \phi), \text{diag}(v(x^{(i)}; \psi))^2) \quad (11.21)$$

这里  $\text{diag}(w)$  表示一个  $k \times k$  矩阵，其对角线上的元素是  $w \in \mathbb{R}^k$  的分量，非对角线元素为零。换句话说，分布  $Q_i$  被假定为具有独立坐标的高斯分布，其均值和标准差由  $q$  和  $v$  控制。在变分自编码器中， $q$  和  $v$  通常被选为神经网络。<sup>8</sup> 在最近的深度学习文献中， $q, v$  通常被称为**编码器 (encoder)**（意为将数据编码为隐变量），而  $g(z; \theta)$  通常被称为**解码器 (decoder)**。

值得注意的是，在许多情况下，这种形式的  $Q_i$  与真实的后验分布相去甚远。不过，为了优化的可行性，还是做了这些近似。事实上， $Q_i$  的形式需要满足其他要求（有时候 (11.21) 也能满足这些要求）。

在优化 ELBO 之前，首先验证，对于形为 (11.21) 的固定值  $Q$  和固定值  $\theta$ ，是否可以有效地计算 ELBO 的值。将 ELBO 重写为  $\phi, \psi, \theta$  的函数：

$$\text{ELBO}(\phi, \psi, \theta) = \sum_{i=1}^n \mathbb{E}_{z^{(i)} \sim Q_i} \left[ \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right], \quad (11.22)$$

其中  $Q_i = \mathcal{N}(q(x^{(i)}; \phi), \text{diag}(v(x^{(i)}; \psi))^2)$ .

---

<sup>8</sup>  $q$  和  $v$  也可以共享参数。讲义在这里不想涉及过多细节。

注意，评估期望中的  $Q_i(z^{(i)})$  需要能对  $Q_i$  计算密度 (compute the density)。为了估计期望  $\mathbb{E}_{z^{(i)} \sim Q_i}$ ，需要能采样于分布 (sample from distribution)  $Q_i$ ，以便用样本构建经验估计器。高斯分布  $Q_i = \mathcal{N}(q(x^{(i)}; \phi), \text{diag}(v(x^{(i)}; \psi))^2)$  能使这两者都高效地进行。

现在来优化 ELBO。结果表明，可以对  $\phi, \psi, \theta$  进行梯度上升，而不是交替最大化。没有强烈的必要以更大的代价计算每个变量上的最大值。（对于第 11.4 节中的高斯混合模型，计算最大值是解析可行的且相对便宜，因此我们进行了交替最大化。）数学地，设  $\eta$  为学习率，梯度上升步骤为

$$\begin{aligned}\theta &:= \theta + \eta \nabla_{\theta} \text{ELBO}(\phi, \psi, \theta) \\ \phi &:= \phi + \eta \nabla_{\phi} \text{ELBO}(\phi, \psi, \theta) \\ \psi &:= \psi + \eta \nabla_{\psi} \text{ELBO}(\phi, \psi, \theta)\end{aligned}$$

计算  $\theta$  的梯度较为简单，因为

$$\begin{aligned}\nabla_{\theta} \text{ELBO}(\phi, \psi, \theta) &= \nabla_{\theta} \sum_{i=1}^n \mathbb{E}_{z^{(i)} \sim Q_i} \left[ \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \\ &= \nabla_{\theta} \sum_{i=1}^n \mathbb{E}_{z^{(i)} \sim Q_i} [\log p(x^{(i)}, z^{(i)}; \theta)] \\ &= \sum_{i=1}^n \mathbb{E}_{z^{(i)} \sim Q_i} [\nabla_{\theta} \log p(x^{(i)}, z^{(i)}; \theta)],\end{aligned}\tag{11.23}$$

然而，计算关于  $\phi$  和  $\psi$  的梯度很棘手，因为采样分布  $Q_i$  依赖于  $\phi$  和  $\psi$ 。（抽象地说，所面临的问题可以简化为计算关于变量  $\phi$  的期望  $\mathbb{E}_{z \sim Q_{\phi}}[f(\phi)]$  的梯度。通常情况下， $\nabla \mathbb{E}_{z \sim Q_{\phi}}[f(\phi)] \neq \mathbb{E}_{z \sim Q_{\phi}}[\nabla f(\phi)]$ ，因为  $Q_{\phi}$  对  $\phi$  的依赖性也必须考虑在内。）

解决这个问题的方式是所谓的重参数化技巧 (re-parameterization trick)：将  $z^{(i)} \sim Q_i = \mathcal{N}(q(x^{(i)}; \phi), \text{diag}(v(x^{(i)}; \psi))^2)$  重写为等价的形式：

$$z^{(i)} = q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)} \quad \text{其中 } \xi^{(i)} \sim \mathcal{N}(0, I_{k \times k})\tag{11.24}$$

这里的  $\odot$  表示两个维度相同的向量的逐元素乘积。这里利用了  $x \sim \mathcal{N}(\mu, \sigma^2)$  等价于  $x = \mu + \xi\sigma$ ，其中  $\xi \sim \mathcal{N}(0, 1)$  这一点。这里是在随机变量  $z^{(i)} \sim Q_i$  的每个维度上同时应用了这一点。

通过重参数化，可以得到

$$\begin{aligned} & \mathbb{E}_{z^{(i)} \sim Q_i} \left[ \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \\ &= \mathbb{E}_{\xi^{(i)} \sim \mathcal{N}(0, I_{k \times k})} \left[ \log \frac{p(x^{(i)}, q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)}; \theta)}{Q_i(q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)})} \right] \end{aligned} \quad (11.25)$$

因此

$$\begin{aligned} & \nabla_{\phi} \mathbb{E}_{z^{(i)} \sim Q_i} \left[ \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \\ &= \nabla_{\phi} \mathbb{E}_{\xi^{(i)} \sim \mathcal{N}(0, I_{k \times k})} \left[ \log \frac{p(x^{(i)}, q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)}; \theta)}{Q_i(q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)})} \right] \\ &= \mathbb{E}_{\xi^{(i)} \sim \mathcal{N}(0, I_{k \times k})} \left[ \nabla_{\phi} \log \frac{p(x^{(i)}, q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)}; \theta)}{Q_i(q(x^{(i)}; \phi) + v(x^{(i)}; \psi) \odot \xi^{(i)})} \right] \end{aligned}$$

现在可以采样多个  $\xi^{(i)}$  来估计上式右侧的期望。<sup>9</sup> 同样，可以类似地估计关于  $\psi$  的梯度，并利用这些梯度实现优化 ELBO 的梯度上升算法。

已知的高维分布中，具有解析可计算密度函数且可重参数化的并不多。对于可以替代高斯分布的其他几种选择，可以参考 Kingma, and Welling 2013。

---

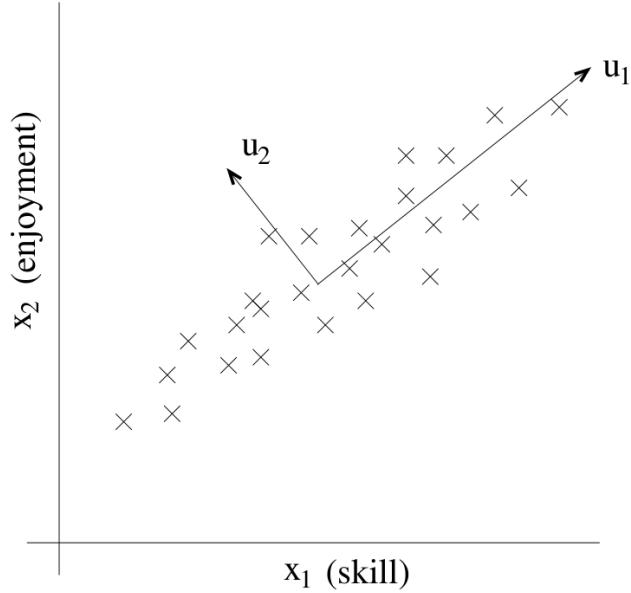
<sup>9</sup>在实践中，为了最大化计算效率，有时仅使用一个样本来估计它。

# 第 12 章 主成分分析 (PCA)

本章将介绍主成分分析 (Principal Components Analysis, PCA) 方法，该方法旨在识别数据大致所在的子空间。PCA 的计算效率很高：它只需要进行特征向量计算（在 Matlab 中使用 `eig` 函数即可轻松完成）。

假设给定一个包含  $n$  种不同类型汽车属性的数据集  $\{x^{(i)}; i = 1, \dots, n\}$ ，属性可能是最大速度、转弯半径等。令每个  $x^{(i)} \in \mathbb{R}^d$ （其中  $d \ll n$ ）。但我们不知道的是，其中两个不同的属性——某个  $x_i$  和  $x_j$ ——分别给出汽车的以英里/小时为单位的最大速度和以公里/小时为单位的最大速度。因此，这两个属性几乎呈线性相关，仅存在由于四舍五入到最接近的英里/小时或公里/小时而引入的微小差异。因此，数据实际上近似位于一个  $n - 1$  维子空间中。该如何自动检测并消除这种冗余？

举一个不那么牵强的例子，考虑一个对遥控直升机飞行员进行调查得到的数据集，其中  $x_1^{(i)}$  是飞行员  $i$  的飞行技能度量，而  $x_2^{(i)}$  衡量他/她对飞行的喜爱程度。由于遥控直升机非常难飞，只有最投入、真正喜欢飞行的学生才能成为优秀的飞行员。因此，属性  $x_1$  和  $x_2$  强相关。事实上，我们可以假设数据实际上大致沿着某个对角线轴（即  $u_1$  方向，捕捉了个人内在的飞行“天赋”）分布，只有少量噪声偏离该轴。（见下图。）我们如何自动计算这个  $u_1$  方向？



接下来将介绍 PCA 算法。但在正式运行 PCA 之前，通常首先通过归一化每个特征来预处理数据，使其均值为 0，方差为 1。具体做法是减去均值并除以经验标准差：

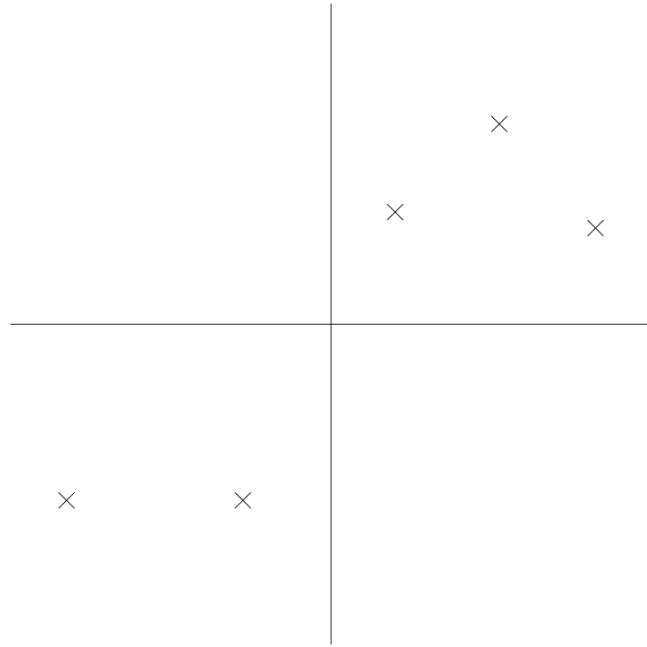
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

其中  $\mu_j = \frac{1}{n} \sum_{i=1}^n x_j^{(i)}$  和  $\sigma_j^2 = \frac{1}{n} \sum_{i=1}^n (x_j^{(i)} - \mu_j)^2$  分别是特征  $j$  的均值和方差。

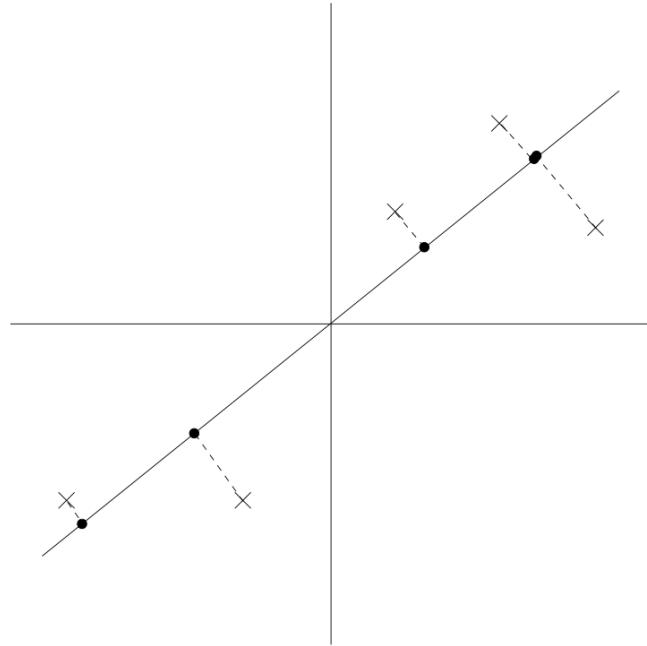
减去  $\mu_j$  会使均值变为零，对于已知均值为零的数据（例如，对应于语音或其他声学信号的时间序列）可以省略这一步。除以标准差  $\sigma_j$  会将每个坐标重新缩放，使其具有单位方差，这确保了不同属性被视为同一“尺度”上的数据。例如，如果  $x_1$  是汽车的最大速度（取值在几十到几百之间），而  $x_2$  是座位数量（取值在 2-4 之间），那么这种重新归一化会重新缩放不同的属性，使它们更具可比性。如果已知不同属性都在同一尺度上，则可以省略这种缩放。一个例子是，如果每个数据点代表一个灰度图像，并且每个  $x_j^{(i)}$  取值在  $\{0, 1, \dots, 255\}$  之间，对应于图像  $i$  中像素  $j$  的强度值。

现在，数据已经归一化，如何计算“主要变化轴”  $u$ ，即数据近似所在的那个方向？一种方法是将这个问题视为寻找单位向量  $u$ ，使得当数据被投影到对应于  $u$  的方向上时，投影数据的方差最大化。直观地说，数据本身具有一定的方差/信息量。我们希望选择一个方向  $u$ ，使得如果我们将数据近似为位于对应于  $u$  的方向/子空间中，则尽可能多地保留这些方差。

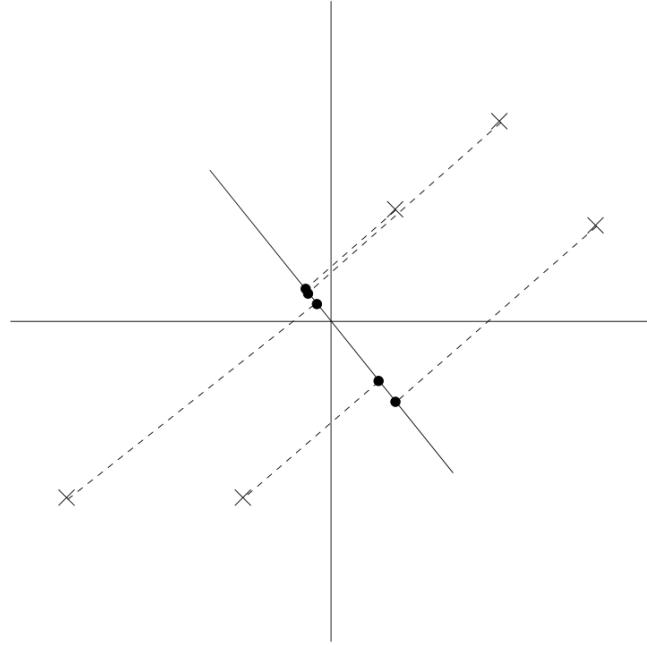
考虑以下已经归一化的数据集：



现在，假设选择的  $u$  对应于下图所示的方向。圆圈表示原始数据点在该直线上的投影。



可以看到，投影后的数据仍然具有相当大的方差，并且数据点倾向于远离零点。相比之下，如果选择以下方向：



在此方向上，投影的方差显著减小，并且更接近原点。

我们希望能够自动选择对应于上面所示两个图中的第一个图的方向  $u$ 。为了形式化这一点，请注意，给定一个单位向量  $u$  和一个点  $x$ ， $x$  在  $u$  上的投影长度由  $x^T u$  给出。也就是说，如果  $x^{(i)}$  是数据集中的一个点（图中交叉点之一），那么它在  $u$  上的投影（图中对应的圆圈）到原点的距离为  $x^{(i)T} u$ 。因此，为了最大化投影的方差，需要选择一个单位长度向量  $u$  来最大化：

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n (x^{(i)T} u)^2 &= \frac{1}{n} \sum_{i=1}^n u^T x^{(i)} x^{(i)T} u \\ &= u^T \left( \frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)T} \right) u. \end{aligned}$$

容易看出，在约束  $\|u\|_2 = 1$  下最大化这个表达式会得到  $\Sigma = \frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)T}$  的主特征向量，这正是数据的经验协方差矩阵（假设数据均值为零）。<sup>1</sup>

总而言之，如果希望找到一个一维子空间来近似数据，应该选择  $u$  作为  $\Sigma$  的主特征向量。更一般地，如果希望将数据投影到  $k$  维子空间 ( $k < d$ )，应该选择  $u_1, \dots, u_k$  作为  $\Sigma$  的前  $k$  个特征向量。这些  $u_i$  现在构成数据的一组新的正交基。<sup>2</sup>

<sup>1</sup>可以尝试使用拉格朗日乘子法来最大化  $u^T \Sigma u$ ，约束条件为  $u^T u = 1$ 。能够证明  $\Sigma u = \lambda u$  对于某个  $\lambda$  成立，这意味着  $u$  是  $\Sigma$  的特征向量， $\lambda$  是对应的特征值。

<sup>2</sup>由于  $\Sigma$  是对称的，所以  $u_i$  是（或总是可以选择是）彼此正交的。

然后，为了在该基下表示  $x^{(i)}$ ，只需要计算相应的向量

$$y^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \in \mathbb{R}^k.$$

因此，尽管  $x^{(i)} \in \mathbb{R}^d$ ，我们能用向量  $y^{(i)}$  作为  $x^{(i)}$  的低维  $k$  维近似/表示。PCA 也因此被称为一种降维 (dimensionality reduction) 算法。向量  $u_1, \dots, u_k$  被称为数据的前  $k$  个主成分 (principal components)。

**备注.** 尽管形式上只对  $k = 1$  的情况进行了推导，但利用特征向量的已知性质，很容易证明在所有可能的正交基  $u_1, \dots, u_k$  中，我们选择的基最大化  $\sum_i \|y^{(i)}\|_2^2$ 。因此，我们选择的基保留了原始数据中尽可能多的变异性。

PCA 也可以通过选择最小化将数据投影到由基向量张成的  $k$  维子空间所产生的近似误差的基来推导（详见习题集）。

PCA 有许多应用，下面将通过几个例子来结束讨论。首先，压缩——用较低维度的  $y^{(i)}$  表示  $x^{(i)}$ ——是一个显而易见的应用。如果将高维数据降至  $k = 2$  或 3 维，那么还可以绘制  $y^{(i)}$  来可视化数据。例如，如果将汽车数据降至 2 维，就可以绘制数据点（图中的一个点将对应一辆汽车，例如某种车型），从而了解哪些汽车彼此相似，以及哪些汽车群体可能聚类在一起。

另一个标准应用是在运行监督学习算法之前对数据集进行预处理，以降低其维度，将  $x^{(i)}$  作为输入。除了计算上的好处外，降低数据的维度还可以降低所考虑的假设类的复杂度，并有助于避免过拟合（例如，低维输入空间上的线性分类器将具有较小的 VC 维）。

最后，在我们遥控直升机飞行员的例子中，也可以将 PCA 视为一种降噪算法。它估计了飞行技能和乐趣的内在“飞行业力”，而排除了嘈杂的测量结果。在课堂上，我们还看到了将这种思想应用于人脸图像的应用，产生了**特征脸 (eigenfaces)** 方法。在这里，每个点  $x^{(i)} \in \mathbb{R}^{100 \times 100}$  是一个 10000 维向量，每个坐标对应于人脸图像中 100x100 像素的像素强度值。使用 PCA，我们将每个图像  $x^{(i)}$  表示为一个低得多维度的  $y^{(i)}$ 。这样做，我们希望主成分能够保留人脸真正有趣、系统的变化，而不会保留由微小的光照变化、略微不同的成像条件等引入的图像中的“噪声”。然后，通过在降维空间中计算  $\|y^{(i)} - y^{(j)}\|_2$  来测量人脸  $i$  和  $j$  之间的距离。这在人脸匹配和检索算法中取得了令人惊讶的好结果。

# 第 13 章 独立成分分析 (ICA)

下一个主题是独立成分分析 (Independent Components Analysis, ICA)。与 PCA 类似，ICA 也旨在找到一组新的基来表示数据，但其目标非常不同。

作为一个启发性例子，考虑“鸡尾酒会问题”。假设有  $d$  位说话者在派对上同时说话，房间里的任何一个麦克风都只记录了这  $d$  位说话者声音的叠加组合。但是假设在房间里放置了  $d$  个不同的麦克风，并且由于每个麦克风与每个说话者的距离不同，它记录了说话者声音的不同组合。使用这些麦克风记录，能否分离出原始的  $d$  位说话者的语音信号？

为了形式化这个问题，假设存在一些数据  $s \in \mathbb{R}^d$ ，这些数据是由  $d$  个独立的源生成的。我们观察到的是

$$x = As,$$

其中  $A$  是一个未知的方阵，称为**混合矩阵 (mixing matrix)**。一系列观察结果构成了数据集  $\{x^{(i)}; i = 1, \dots, n\}$ ，我们的目标是恢复生成源数据  $s^{(i)}$ （其中  $x^{(i)} = As^{(i)}$ ）。

在鸡尾酒会问题中， $s^{(i)}$  是一个  $d$  维向量， $s_j^{(i)}$  是说话者  $j$  在时刻  $i$  发出的声音。此外， $x^{(i)}$  是一个  $d$  维向量， $x_j^{(i)}$  是麦克风  $j$  在时刻  $i$  记录的声学读数。

令  $W = A^{-1}$  为**解混矩阵 (unmixing matrix)**。我们的目标是找到  $W$ ，以便给定麦克风记录  $x^{(i)}$ ，可以通过计算  $s^{(i)} = Wx^{(i)}$  来恢复源数据。为了方便记号，也令  $w_i^T$  表示  $W$  的第  $i$  行，以便

$$W = \begin{bmatrix} -w_1^T - \\ -\vdots - \\ -w_d^T - \end{bmatrix}.$$

因此， $w_i \in \mathbb{R}^d$ ，并且第  $j$  个源可以恢复为  $s_j^{(i)} = w_j^T x^{(i)}$ 。

### 13.1 ICA 的不确定性

$W = A^{-1}$  可以在多大程度上被恢复？如果我们对源和混合矩阵一无所知，很容易看出在仅给定  $x^{(i)}$  的情况下，存在一些固有的  $A$  中无法恢复的不确定性。

具体来说，令  $P$  为任意  $d \times d$  置换矩阵。这意味着  $P$  的每一行和每一列都恰好有一个“1”。以下是一些置换矩阵的例子：

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

如果  $z$  是一个向量，那么  $Pz$  是另一个向量，它包含  $z$  的坐标的置换版本。仅给定  $x^{(i)}$  是无法区分  $W$  和  $PW$  的。具体来说，原始源的置换是不确定的，这不足为奇。幸运的是，这对于大多数应用来说并不重要。

此外，也无法恢复  $w_i$  的正确比例。例如，如果  $A$  被  $2A$  替换，并且每个  $s^{(i)}$  都被  $(0.5)s^{(i)}$  替换，那么我们观察到的  $x^{(i)} = 2A \cdot (0.5)s^{(i)}$  仍然相同。进一步地，如果  $A$  的某一列被因子  $\alpha$  缩放，而相应的源被因子  $1/\alpha$  缩放，那么仅给定  $x^{(i)}$ ，就无法确定发生了这种情况。因此，无法恢复源的“正确”比例。然而，对于我们关心的应用——包括鸡尾酒会问题——这种不确定性同样不重要。具体来说，将说话者  $j$  的语音信号  $s_j^{(i)}$  乘以某个正因子  $\alpha$  只会影响该语音的音量。符号变化也不重要， $s_j^{(i)}$  和  $-s_j^{(i)}$  在扬声器上播放时听起来是相同的。因此，如果算法找到的  $w_i$  被任何非零实数缩放，相应的恢复源  $s_i = w_i^T x$  也将缩放同样的因子；但这通常不重要。（这些评论也适用于我们在课堂上讨论的脑/MEG 数据的 ICA。）

这些是 ICA 中唯一的不确定性来源吗？事实证明如果源  $s_i$  是非高斯 (*non-Gaussian*) 分布的，这些就确实是仅有的不确定性来源。为了了解高斯数据的困难所在，考虑一个例子，其中  $n = 2$ ，且  $s \sim N(0, I)$ 。这里， $I$  是  $2 \times 2$  单位矩阵。注意，标准正态分布  $N(0, I)$  的密度等高线是中心位于原点的圆，并且密度是旋转对称的。

现在，假设观察到一些  $x = As$ ，其中  $A$  是混合矩阵。那么， $x$  的分布将是高斯分布， $x \sim N(0, AA^T)$ ，因为

$$\mathbb{E}_{s \sim N(0, I)}[x] = \mathbb{E}[As] = A\mathbb{E}[s] = 0$$

$$\text{Cov}[x] = \mathbb{E}_{s \sim N(0, I)}[xx^T] = \mathbb{E}[Ass^TA^T] = A\mathbb{E}[ss^T]A^T = A \cdot \text{Cov}[s] \cdot A^T = AA^T$$

现在，令  $R$  为任意正交矩阵（不太正式地说，为旋转或反射矩阵），使得  $RR^T = R^TR = I$ ，并令  $A' = AR$ 。那么，如果数据是按照  $A'$  而不是  $A$  混合的，那么我

们将观察到  $x' = A's$ 。 $x'$  的分布也是高斯分布， $x' \sim N(0, AA^T)$ ，因为

$$\mathbb{E}_{s \sim N(0, I)}[x'(x')^T] = \mathbb{E}[A'ss^T(A')^T] = \mathbb{E}[ARss^T(AR)^T] = ARR^TA^T = AA^T.$$

因此，无论混合矩阵是  $A$  还是  $A'$ ，我们都会观察到服从  $N(0, AA^T)$  分布的数据。因此，无法判断源是使用  $A$  还是  $A'$  混合的。混合矩阵中存在一个任意的旋转分量，无法从数据中确定，并且无法恢复原始源。

上面的论证是基于多元标准正态分布是旋转对称的事实。尽管高斯数据的 ICA 前景黯淡，但事实证明，只要数据不是高斯分布的，就有可能在给定足够数据的情况下恢复  $d$  个独立源。

## 13.2 密度与线性变换

在继续推导 ICA 算法之前，先简要介绍一下线性变换对概率密度的影响。

假设随机变量  $s$  服从密度函数  $p_s(s)$ 。为简单起见，暂时假设  $s \in \mathbb{R}$  是一个实数。现在，令随机变量  $x$  定义为  $x = As$ （这里  $x \in \mathbb{R}, A \in \mathbb{R}$ ）。令  $p_x$  为  $x$  的密度。 $p_x$  是什么？

令  $W = A^{-1}$ 。计算特定值  $x$  的“概率”时，很容易想到计算  $s = Wx$ ，然后评估  $p_s$  在该点的值，并得出结论  $p_x(x) = p_s(Wx)$ 。然而，这是不正确的。例如，令  $s \sim \text{Uniform}[0, 1]$ ，所以  $p_s(s) = 1\{0 \leq s \leq 1\}$ 。现在，令  $A = 2$ ，所以  $x = 2s$ 。显然， $x$  在区间  $[0, 2]$  上均匀分布。因此，其密度为  $p_x(x) = (0.5)1\{0 \leq x \leq 2\}$ 。这不等于  $p_s(Wx)$ ，其中  $W = 0.5 = A^{-1}$ 。正确的公式是  $p_x(x) = p_s(Wx)|W|$ 。

更一般地，如果  $s$  是一个具有密度  $p_s$  的向量值分布，并且  $x = As$  对于一个方阵、可逆矩阵  $A$ ，那么  $x$  的密度由下式给出：

$$p_x(x) = p_s(Wx) \cdot |W|,$$

其中  $W = A^{-1}$ 。

**备注.** 如果你见过矩阵  $A$  将  $[0, 1]^d$  映射到一个体积为  $|A|$  的集合的结果，那么这里有一个另一种记忆上述  $p_x$  公式的方法，它也推广了我们之前的一维例子。具体来说，令  $A \in \mathbb{R}^{d \times d}$ ，并且像往常一样令  $W = A^{-1}$ 。也令  $C_1 = [0, 1]^d$  为  $d$  维超立方体，并定义  $C_2 = \{As : s \in C_1\} \subseteq \mathbb{R}^d$  为  $C_1$  在由  $A$  给定的映射下的像。那么，这是一个线性代数中的标准结果（实际上，也是定义行列式的一种方式），即  $C_2$  的体积由  $|A|$  给出。现在，假设  $s$  在  $[0, 1]^d$  上均匀分布，所以其密度为  $p_s(s) = 1\{s \in C_1\}$ 。那么显然  $x$  将在  $C_2$  中均匀分布。因此，可以找到其密度为  $p_x(x) = 1\{x \in C_2\}/\text{vol}(C_2)$ （因为在  $C_2$  上的积分必须为 1）。但是利用矩阵逆的

行列式就是行列式的倒数这一事实，我们有  $1/\text{vol}(C_2) = 1/|A| = |A^{-1}| = |W|$ 。因此， $p_x(x) = 1\{x \in C_2\}|W| = 1\{Wx \in C_1\}|W| = p_s(Wx)|W|$ 。

### 13.3 ICA 算法

现在准备推导 ICA 算法，一个由 Bell 和 Sejnowski 提出的算法，并将其解释为最大似然估计方法。（这与他们原始的解释不同，后者涉及一个复杂的概念，称为 infomax 原理，鉴于对 ICA 的现代理解，这已不再必要。）

假设每个源  $s_j$  的分布由密度  $p_s$  给出，并且源  $s$  的联合分布由下式给出：

$$p(s) = \prod_{j=1}^d p_s(s_j).$$

注意，通过将联合分布建模为边缘分布的乘积，满足了源是独立的假设。利用上一节的公式，得到  $x = As = W^{-1}s$  的密度如下：

$$p(x) = \prod_{j=1}^d p_s(w_j^T x) \cdot |W|.$$

剩下的就是指定各个源的密度  $p_s$ 。

回想一下，给定一个实值随机变量  $z$ ，其累积分布函数 (cumulative distribution function, cdf)  $F$  定义为  $F(z_0) = P(z \leq z_0) = \int_{-\infty}^{z_0} p_z(z) dz$ ，并且密度  $p_z(z) = F'(z)$  是 cdf 的导数。

因此，要指定  $s_i$  的密度，只需要为其指定一个 cdf。cdf 必须是一个从零单调递增到一的函数。根据之前的讨论，不能选择高斯 cdf，因为 ICA 不适用于高斯数据。我们将选择一个合理的“默认”cdf，它从零缓慢增加到一，即 sigmoid 函数  $g(s) = 1/(1 + e^{-s})$ 。因此， $p_s(s) = g'(s)$ 。<sup>1</sup>

方阵  $W$  是模型中的参数。给定训练集  $\{x^{(i)}; i = 1, \dots, n\}$ ，对数似然由下式给出：

$$\ell(W) = \sum_{i=1}^n \left( \sum_{j=1}^d \log g'(w_j^T x^{(i)}) + \log |W| \right).$$

---

<sup>1</sup>如果你事先知道源的密度具有某种形式，那么将其代入此处是个好主意。但在缺乏此类知识的情况下，sigmoid 函数可以被视为一个合理的默认设置，它在许多问题上效果很好。此外，这里的演示假设数据  $x^{(i)}$  已被预处理为零均值，或者自然地预期为零均值（例如声学信号）。这是必要的，因为我们的假设  $p_s(s) = g'(s)$  (logistic 函数的导数是一个对称函数，因此给出了对应于零均值随机变量的密度) 意味着  $\mathbb{E}[s] = 0$ ，这意味着  $\mathbb{E}[x] = \mathbb{E}[As] = 0$ 。

希望最大化关于  $W$  的对数似然。通过求导并利用（第一章的） $\nabla_W|W| = |W|(W^{-1})^T$  这一事实，可以很容易地推导出随机梯度上升学习规则。对于一个训练样本  $x^{(i)}$ ，更新规则为：

$$W := W + \alpha \begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_d^T x^{(i)}) \end{pmatrix} x^{(i)T} + (W^T)^{-1},$$

其中  $\alpha$  是学习率。

算法收敛后，计算  $s^{(i)} = Wx^{(i)}$  以恢复原始源。

**备注.** 在写数据的似然时，隐式地假设了  $x^{(i)}$  彼此独立（对于不同的  $i$ ；注意这个问题与  $x^{(i)}$  的不同坐标是否独立不同），因此训练集的似然由  $\prod_i p(x^{(i)}; W)$  给出。对于语音数据和  $x^{(i)}$  有相关性的其他时间序列，这个假设显然是不正确的，但可以证明，如果数据充足，有相关性的训练样本不会损害算法的性能。然而，对于连续训练样本相关的问题，在实现随机梯度上升时，如果以随机排列的顺序访问训练样本，有时有助于加速收敛。（即，在随机打乱的训练集副本上运行随机梯度上升。）

# 第 14 章 自监督学习与基础模型

尽管取得了巨大成功，但基于神经网络的监督学习通常依赖于大规模的标注数据集，收集成本很高。最近，随着一系列模型（例如，BERT [Devlin et al. 2019] 和 GPT-3 [Brown et al. 2020]）的兴起，人工智能和机器学习正在经历一场范式转变。这些模型在大规模广域数据上进行预训练，并且适用于各种下游任务。这些模型，被 Bommasani et al. 2021 称为基础模型，通常利用海量无标注数据，因此下游任务所需的标注数据大大减少。此外，尽管基础模型基于标准的深度学习和迁移学习，但其规模带来了新的涌现能力。这些模型通常通过自监督学习方法进行（预）训练，其中监督/标注就来自输入数据的一部分。

本章将介绍基础模型的范式和相关的基本概念。

## 14.1 预训练与适配

基础模型范式包含两个阶段：预训练（或简称为训练）和适配。首先，在一个大规模无标注数据集（例如，数十亿张无标注图像）上预训练一个大型模型。<sup>1</sup>然后，将预训练模型适配到下游任务（例如，从扫描图像中检测癌症）。这些下游任务通常是预测任务，其标注数据有限或者甚至没有标注数据。其直觉是，预训练模型学习到能够捕捉数据内在语义结构/信息的良好表示 (*representations*)，而适配阶段则根据特定下游任务对模型进行定制，例如，检索与其相关的信息。例如，在一个大规模无标注图像数据上预训练的模型可能会学习到良好的通用视觉表示/特征，然后我们将这些表示适配于解决生物医学图像任务。

下面我们形式化这两个阶段。

### 预训练

假设有一个无标注预训练数据集  $\{x^{(1)}, x^{(2)}, \dots, x^{(n)}\}$ ，包含  $n$  个位于  $\mathbb{R}^d$  中的样本。设  $\phi_\theta$  是一个模型，由  $\theta$  参数化，并将输入  $x$  映射到某个  $m$  维表示

---

<sup>1</sup>有时候预训练也包含了大规模带标注数据，例如 ImageNet 数据集。

$\phi_\theta(x)$ 。（人们也将  $\phi_\theta(x) \in \mathbb{R}^m$  称为样本  $x$  的嵌入或特征。）我们使用预训练损失  $L_{\text{pre}}(\theta)$  对模型  $\theta$  进行预训练，该损失通常是所有样本上损失函数的平均值： $L_{\text{pre}}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell_{\text{pre}}(\theta, x^{(i)})$ 。这里的  $\ell_{\text{pre}}$  是一种所谓的单一样本  $x^{(i)}$  上的自监督损失，因为监督信息来自数据点  $x^{(i)}$  本身，正如后面在第 14.3 节中所示。预训练损失也可以是单个样本损失的总和。我们将在第 14.2 节和第 14.3 节中讨论两种预训练损失。

我们用一些优化器（可能是 SGD 或 ADAM [Kingma, and Ba 2014]）来最小化  $L_{\text{pre}}(\theta)$ 。将得到的预训练模型记为  $\hat{\theta}$ 。

## 适配

对于下游任务，通常有一个标注数据集  $\{(x_{\text{task}}^{(1)}, y_{\text{task}}^{(1)}), \dots, (x_{\text{task}}^{(n_{\text{task}})}, y_{\text{task}}^{(n_{\text{task}})})\}$ ，包含  $n_{\text{task}}$  个样本。当  $n_{\text{task}} = 0$  时，称为零样本学习——即下游任务没有任何标注样本。当  $n_{\text{task}}$  相对较小（例如，在 1 到 50 之间）时，称为少样本学习。较大的  $n_{\text{task}}$ （数量级从数百到数万）也非常常见。

适配算法通常以下游数据集和预训练模型  $\hat{\theta}$  作为输入，并输出解决下游任务的  $\hat{\theta}$  的变体。下面将讨论两种流行且通用的适配方法：线性探测 (linear probe) 和微调 (finetune)。此外，还将在 14.3.1 节中介绍另外两种针对语言问题的方法。

线性探测方法在表示层之上使用一个线性头来预测下游标签。数学上，适配后的模型输出  $w^T \phi_{\hat{\theta}}(x)$ ，其中  $w \in \mathbb{R}^m$  是待学习的参数，而  $\hat{\theta}$  是固定的预训练模型。可以使用 SGD（或其他优化器）在下游任务损失上训练  $w$  以预测任务标签：

$$\min_{w \in \mathbb{R}^m} \frac{1}{n_{\text{task}}} \sum_{i=1}^{n_{\text{task}}} \ell_{\text{task}}(y_{\text{task}}^{(i)}, w^T \phi_{\hat{\theta}}(x_{\text{task}}^{(i)})) \quad (14.1)$$

例如，若下游任务是回归问题，则  $\ell_{\text{task}}(y_{\text{task}}^{(i)}, w^T \phi_{\hat{\theta}}(x_{\text{task}}^{(i)})) = (y_{\text{task}}^{(i)} - w^T \phi_{\hat{\theta}}(x_{\text{task}}^{(i)}))^2$ 。

微调算法对下游预测模型使用类似的结构，但会进一步微调预训练模型（而不是固定它）。具体来说，预测模型是  $w^T \phi_\theta(x)$ ，参数为  $w$  和  $\theta$ 。我们优化  $w$  和  $\theta$  以适配下游数据，但用预训练模型  $\hat{\theta}$  初始化  $\theta$ 。线性头  $w$  通常随机初始化。

$$\underset{w, \theta}{\text{minimize}} \frac{1}{n_{\text{task}}} \sum_{i=1}^{n_{\text{task}}} \ell_{\text{task}}(y_{\text{task}}^{(i)}, w^T \phi_\theta(x_{\text{task}}^{(i)})) \quad (14.2)$$

$$\text{初始化 } w \leftarrow \text{随机向量} \quad (14.3)$$

$$\theta \leftarrow \hat{\theta} \quad (14.4)$$

存在各种其他适配方法，有时是针对特定的预训练方法而专门设计的。我们将在第 14.3.1 节中讨论其中一种。

## 14.2 计算机视觉中的预训练方法

本节介绍两种计算机视觉中具体的预训练方法：监督预训练和对比学习。

### 监督预训练

在这里，预训练数据集是一个大规模标注 (*labeled*) 数据集（例如，ImageNet），而预训练模型是一个使用普通监督学习训练的神经网络（去掉了最后一层）。具体来说，假设将学习到的神经网络表示为  $U\phi_{\hat{\theta}}(x)$ ，其中  $U$  是最后一层（全连接层）的参数， $\hat{\theta}$  对应于所有其他层的参数，而  $\phi_{\hat{\theta}}(x)$  是倒数第二层的激活（用作表示）。我们丢弃  $U$ ，并将  $\phi_{\hat{\theta}}(x)$  作为预训练模型。

### 对比学习

对比学习是一种仅使用无标注数据的自监督预训练方法。其主要直觉是，一个好的表示函数  $\phi_{\theta}(\cdot)$  应该将语义相似的图像映射到相似的表示，而随机的图像对通常应该具有不同的表示。例如，我们可能希望将两只哈士奇的图像映射到相似的表示，而将哈士奇和大象的图像映射到不同的表示。一种相似性的定义是同一类别的图像是相似的。使用这个定义将产生所谓的监督对比学习算法，这些算法在有标注的预训练数据集上表现良好。

在没有标注数据的情况下，可以使用数据增强来生成一对“相似”的增强图像，给定原始图像  $x$ 。数据增强通常意味着对原始图像  $x$  应用随机 (*random*) 裁剪、翻转和/或颜色变换来生成变体。可以对同一原始图像  $x$  进行两次随机增强，分别记为  $\hat{x}$  和  $\tilde{x}$ ，并将它们称为正样本对。可以观察到正样本对的图像通常在语义上相关，因为它们是同一图像的增强。我们将设计一个损失函数，使得  $\theta$  能够使正样本对的表示  $\phi_{\theta}(\hat{x}), \phi_{\theta}(\tilde{x})$  尽可能接近。

另一方面，还可以从预训练数据集中获取另一个随机图像  $z$ ，并从  $z$  生成一个增强  $\hat{z}$ 。注意， $(\hat{x}, \hat{z})$  来自不同的图像；因此，很有可能它们在语义上不相关。 $(\hat{x}, \hat{z})$  被称为负样本对或随机样本对。<sup>2</sup> 我们将设计一个损失来推开随机样本对的表示  $\phi_{\theta}(\hat{x}), \phi_{\theta}(\hat{z})$ 。

有许多基于对比学习原理的最新算法，这里以 SIMCLR [Chen et al. 2020] 为例。损失函数定义在大小为  $B$  的一批样本  $(x^{(1)}, \dots, x^{(B)})$  上。算法对批次中的每个样本  $x^{(i)}$  施加两个随机增强，分别记为  $\hat{x}^{(i)}$  和  $\tilde{x}^{(i)}$ 。这样就得到大小为  $2B$

---

<sup>2</sup>随机样本对可能更准确，因为  $x$  和  $z$  仍然有可能（尽管概率很低）在语义上相关， $\hat{x}$  和  $\hat{z}$  也是如此。但在文献中，负样本对这个术语也很常见。

的增广批次： $\hat{x}^{(1)}, \dots, \hat{x}^{(B)}, \tilde{x}^{(1)}, \dots, \tilde{x}^{(B)}$ 。SIMCLR 损失定义为<sup>3</sup>

$$L_{\text{pre}}(\theta) = - \sum_{i=1}^B \log \frac{\exp(\phi_\theta(\hat{x}^{(i)})^T \phi_\theta(\tilde{x}^{(i)}))}{\exp(\phi_\theta(\hat{x}^{(i)})^T \phi_\theta(\tilde{x}^{(i)})) + \sum_{j \neq i} \exp(\phi_\theta(\hat{x}^{(i)})^T \phi_\theta(\tilde{x}^{(j)}))}.$$

这里的直觉是，损失随着  $\phi_\theta(\hat{x}^{(i)})^T \phi_\theta(\tilde{x}^{(j)})$  增加而增加，因此最小化损失会鼓励  $\phi_\theta(\hat{x}^{(i)})^T \phi_\theta(\tilde{x}^{(j)})$  变小，从而使  $\phi_\theta(\hat{x}^{(i)})$  远离  $\phi_\theta(\tilde{x}^{(j)})$ 。另一方面，损失随着  $\phi_\theta(\hat{x}^{(i)})^T \phi_\theta(\tilde{x}^{(i)})$  增加而减少，因此最小化损失会鼓励  $\phi_\theta(\hat{x}^{(i)})$  和  $\phi_\theta(\tilde{x}^{(i)})$  变得接近。<sup>4</sup>

### 14.3 预训练的大语言模型

自然语言处理是预训练模型特别成功的另一个领域。在语言问题中，一个例子通常对应于一个文档或通常是一个词序列/片段，<sup>5</sup> 表示为  $x = (x_1, \dots, x_T)$ ，其中  $T$  是文档/序列的长度， $x_i \in \{1, 2, \dots, V\}$  是文档中的词， $V$  是词汇表大小<sup>6</sup>。  
<sup>6</sup>

语言模型是一个概率模型，表示文档的概率，记为  $p(x_1, \dots, x_T)$ 。这个概率分布非常复杂，因为它的支撑集大小是  $V^T$ ——关于文档长度呈指数级增长。与其直接建模文档的分布，不如应用条件概率的链式法则将其分解如下：

$$p(x_1, \dots, x_T) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \dots, x_{T-1}). \quad (14.5)$$

现在，每个条件概率  $p(x_t|x_1, \dots, x_{t-1})$  的支撑集大小都是  $V$ 。

将条件概率  $p(x_t|x_1, \dots, x_{t-1})$  建模为  $x_1, \dots, x_{t-1}$  的函数，并用  $\theta$  参数化。

参数化模型接受数值输入，因此先引入词的嵌入或表示。设  $e_i \in \mathbb{R}^d$  是词  $i \in \{1, 2, \dots, V\}$  的嵌入。 $[e_1, \dots, e_V] \in \mathbb{R}^{d \times V}$  被称为嵌入矩阵。

最常用的模型是 Transformer [Vaswani et al. 2017]。本节将介绍 Transformer 的输入输出接口，但将 Transformer 中的中间计算视为黑箱。有关更多详细信息，请参阅 Transformer 论文或更高级的课程。如图 14.1 所示，给定一个文档  $(x_1, \dots, x_T)$ ，首先将离散变量序列转换为相应的词嵌入  $(e_{x_1}, \dots, e_{x_T})$ 。并且词

<sup>3</sup>这是原始损失的一个变体和简化，它不改变本质（但可能会稍微改变效率）。

<sup>4</sup>为了理解这一点，可以验证函数  $-\log \frac{p}{p+q}$  在  $p$  增大时减小，而在  $q$  增大时增大，前提是  $p, q > 0$ 。

<sup>5</sup>在实际实现中，通常将所有数据按某种顺序串联成一个单一序列，每个样本通常对应于一个连续词的子序列，该子序列可能对应于文档的一部分或跨越多个文档。

<sup>6</sup>从技术上讲，词可以分解为 token，可以是词或子词（字母组合），但这里略去这一技术细节。事实上，大多数常见词本身就是一个 token。

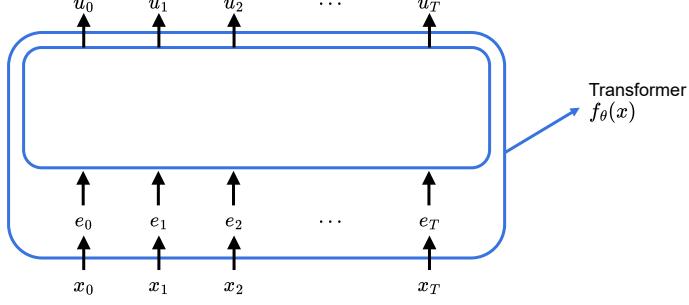


图 14.1: Transformer 模型的输入输出。

汇表中引入一个固定的特殊标记  $x_0 = \perp$ , 其对应的嵌入  $e_{x_0}$  用于标记文档的开始。然后, 将词嵌入输入到 Transformer 模型中, 该模型接收一个向量序列  $(e_{x_0}, e_{x_1}, \dots, e_{x_T})$ , 并输出一个向量序列  $(u_1, u_2, \dots, u_{T+1})$ , 其中  $u_t \in \mathbb{R}^V$  将被解释为下一个词概率分布的 logits。这里使用 Transformer 的自回归版本, 其设计确保  $u_t$  仅依赖于  $x_1, \dots, x_{t-1}$  (注意, 在掩码语言模型 [Devlin et al. 2019] 中, 损失函数也不同, 这一点不成立)。在本节中, 我们将从  $x$  到  $u$  的整个映射视为一个黑箱, 并将其称为 Transformer, 记为  $f_\theta$ , 其中  $\theta$  包括 Transformer 中的参数和输入嵌入。记  $u_t = f_\theta(x_0, x_1, \dots, x_{t-1})$ , 其中  $f_\theta$  表示从输入到输出的映射。

条件概率  $p(x_t|x_1, \dots, x_{t-1})$  是 logits 的 softmax:

$$\begin{bmatrix} p(x_t = 1|x_1, \dots, x_{t-1}) \\ p(x_t = 2|x_1, \dots, x_{t-1}) \\ \vdots \\ p(x_t = V|x_1, \dots, x_{t-1}) \end{bmatrix} = \text{softmax}(u_t) \in \mathbb{R}^V \quad (14.6)$$

$$= \text{softmax}(f_\theta(x_0, \dots, x_{t-1})) \quad (14.7)$$

通过最小化在由  $\theta$  定义的概率模型下看到数据的负对数似然, 从而训练 Transformer 参数  $\theta$ 。这等价于最小化 logits 上的交叉熵损失。

$$\begin{aligned} \text{loss}(\theta) &= \frac{1}{T} \sum_{t=1}^T -\log(p_\theta(x_t|x_1, \dots, x_{t-1})) \\ &= \frac{1}{T} \sum_{t=1}^T \ell_{\text{ce}}(f_\theta(x_0, x_1, \dots, x_{t-1}), x_t) \\ &= \frac{1}{T} \sum_{t=1}^T -\log(\text{softmax}(f_\theta(x_0, x_1, \dots, x_{t-1})))_{x_t}. \end{aligned} \quad (14.8)$$

## 自回归文本解码 / 生成

给定一个自回归 Transformer，可以直接从中按顺序采样文本。给定前缀  $x_1, \dots, x_t$ ，利用条件分布按顺序生成文本补全  $x_{t+1}, \dots, x_T$ ：

$$x_{t+1} \sim \text{softmax}(f_\theta(x_0, x_1, \dots, x_t)) \quad (14.9)$$

$$x_{t+2} \sim \text{softmax}(f_\theta(x_0, x_1, \dots, x_{t+1})) \quad (14.10)$$

$$\dots \quad (14.11)$$

$$x_T \sim \text{softmax}(f_\theta(x_0, x_1, \dots, x_{T-1})). \quad (14.12)$$

注意，生成的每一个 token 都被用作生成后续 token 时模型的输入。在实践中，为了进一步调整生成分布的熵/尖锐度，通常会引入一个名为 温度 (*temperature*) 的参数  $\tau > 0$ 。当  $\tau = 1$  时，文本从模型定义的原始条件概率中采样。随着  $\tau$  的减小，生成的文本逐渐变得更加“确定性”。当  $\tau \rightarrow 0$  时，这退化为贪婪解码，即从条件概率中生成概率最高的下一个 token。

### 14.3.1 零样本学习与语境学习

对于语言模型，有多种方法可以将预训练模型适配到下游任务。本小节将讨论其中的三种：微调、零样本学习和上下文学习。

**微调 (Finetuning)** 对于自回归语言模型并不常见，但在其他变体（如掩码语言模型）中更为常见，这些变体具有类似的输入-输出接口，但预训练方式不同 [Devlin et al. 2019]。微调方法与第 14.1 节中介绍的通用方法相同——唯一的问题是如何定义预测任务并添加一个额外的线性头部。一种选择是将  $c_{T+1} = \phi_\theta(x_1, \dots, x_T)$  视为表示，并使用  $w^T c_{T+1} = w^T \phi_\theta(x_1, \dots, x_T)$  来预测任务标签。如第 14.1 节所述，将  $\theta$  初始化为预训练模型  $\hat{\theta}$ ，然后同时优化  $w$  和  $\theta$ 。

**零样本 (Zero-shot)** 适配或零样本学习是指下游任务没有输入-输出对的情景。对于语言问题任务，通常将任务格式化为问题或通过自然语言的完形填空形式。例如：

$$x_{\text{task}} = (x_{\text{task},1}, \dots, x_{\text{task},T}) = \text{“光速是常数吗？”}$$

然后，计算此问题下，语言模型所预测的最有可能的下一个词，即计算下式  $\text{argmax}_{x_{T+1}} p(x_{T+1} | x_{\text{task},1}, \dots, x_{\text{task},T})$ 。在这种情况下，如果最有可能的下一个词  $x_{T+1}$  是“否”，则任务完成。（光速仅在真空中是常数。）注意，有许多方法可以从语言模型中解码答案，例如，除了计算 argmax，也可以使用语言模型直接生成几个词作为答案。寻找利用语言模型的最佳方法是一个活跃的研究问题。

**语境学习 (In-context learning)** 主要用于少样本情景，此时我们有少量带标签的例子  $(x_{\text{task}}^{(1)}, y_{\text{task}}^{(1)}), \dots, (x_{\text{task}}^{(n_{\text{task}})}, y_{\text{task}}^{(n_{\text{task}})})$ 。给定一个测试例子  $x_{\text{test}}$ ，通过以某种格式将带标签的例子和测试例子连接起来，构建一个文档  $(x_1, \dots, x_T)$ ，这在上下文中通常称为“提示 (prompt)”。例如，可以按如下方式构建提示：

$$\begin{aligned}
 x_1, \dots, x_T &= \text{“Q: } 2 \sim 3 = ? \quad x_{\text{task}}^{(1)} \\
 &\quad \text{A: } 5 \quad y_{\text{task}}^{(1)} \\
 &\quad \text{Q: } 6 \sim 7 = ? \quad x_{\text{task}}^{(2)} \\
 &\quad \text{A: } 13 \quad y_{\text{task}}^{(2)} \\
 &\quad \dots \\
 &\quad \text{Q: } 15 \sim 2 = ?” \quad x_{\text{test}}
 \end{aligned}$$

然后，让预训练模型生成最有可能的  $x_{T+1}, x_{T+2}, \dots$ 。在这种情况下，如果模型能从少量例子中“学会”符号  $\sim$  代表加法，将得到以下结果，表明答案是 17。

$$x_{T+1}, x_{T+2}, \dots = \text{“A: } 17” .$$

基础模型领域是一个非常新且快速发展的领域。这里仅试图在概念层面介绍这些模型，并进行了大量简化。关于更多细节，请参阅其他材料，例如 Bommasani et al. 2021。

# **第五部分**

## **强化学习与控制**

# 第 15 章 强化学习

现在开始学习强化学习和自适应控制。

监督学习中的算法试图使其输出模仿训练集中的标签  $y$ 。在这种设置下，对于每个输入  $x$ ，标签都给出了明确的“正确答案”。然而，对于许多序列决策和控制问题，很难为学习算法提供这种显式监督。例如，如果我们构建了一个四足机器人并试图对其进行编程以使其行走，那么我们其实不知道什么是使其行走的“正确”动作，因此也不知道如何为学习算法提供显式监督以供其模仿。

强化学习将转而为算法提供一个奖励函数，该函数指示学习智能体何时表现良好，何时表现不佳。在四足行走示例中，奖励函数可以对机器人向前移动给予正奖励，而对向后移动或摔倒给予负奖励。然后，学习算法的任务就是随着时间推移确定如何选择动作以获得高奖励。

强化学习已成功应用于各种领域，包括自主直升机飞行、机器人腿部运动、蜂窝网络路由、营销策略选择、工厂控制和高效网页索引。对强化学习的研究将从**定义马尔可夫决策过程 (Markov decision processes, MDP)**开始，它提供了表述强化学习问题的一般化形式化框架。

## 15.1 马尔可夫决策过程

马尔可夫决策过程是一个元组  $(S, A, \{P_{sa}\}, \gamma, R)$ ，其中：

- $S$  是一个**状态 (states)** 集合。(例如，在自主直升机飞行中， $S$  可以是直升机所有可能的位置和方向的集合。)
- $A$  是一个**动作 (actions)** 集合。(例如，可以推动直升机控制杆的所有可能方向的集合。)
- $P_{sa}$  是状态转移概率 (state transition probabilities)。对于每个状态  $s \in S$  和动作  $a \in A$ ， $P_{sa}$  是状态空间上的一个分布。稍后会详细介绍，但简而言之， $P_{sa}$  给出了如果在状态  $s$  采取动作  $a$  后将转移到哪些状态的分布。

- $\gamma \in [0, 1]$  称为**折扣因子 (discount factor)**。
- $R : S \times A \mapsto \mathbb{R}$  是**奖励函数 (reward function)**。(奖励有时也写成仅关于状态  $S$  的函数，在这种情况下有  $R : S \mapsto \mathbb{R}$ )。

MDP 的动态过程如下：我们从某个状态  $s_0$  开始，然后在 MDP 中选择一个动作  $a_0 \in A$  执行。然后 MDP 的状态随机转移到某个后继状态  $s_1$ ，根据  $s_1 \sim P_{s_0 a_0}$  抽取。然后选择另一个动作  $a_1$ 。由于这个动作，状态再次转移，现在转移到某个  $s_2 \sim P_{s_1 a_1}$ ，依此类推。可以将这个过程表示为：

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

以动作序列  $a_0, a_1, \dots$  遍历状态序列  $s_0, s_1, \dots$  后，总收益由下式给出

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots$$

或者，将奖励写成仅关于状态的函数时，则变为

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

在大部分推导中，将使用更简单的状态奖励  $R(s)$ ，尽管推广到状态-动作奖励  $R(s, a)$  也不会有特别的困难。

在强化学习中，目标是随着时间推移选择动作以最大化总收益的期望值：

$$\mathbb{E} [R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots]$$

注意，时间步  $t$  的奖励被  $\gamma^t$  **折扣 (discount)**。因此，为了使这个期望值最大化，希望尽快累积正奖励（并尽可能推迟负奖励）。在经济应用中，如果  $R(\cdot)$  代表赚取的金额，那么  $\gamma$  也有一个自然的利率解释（今天的一美元比明天的一美元更值钱）。

**策略 (policy)** 是一个函数  $\pi : S \mapsto A$ ，它将状态映射到动作。当处于状态  $s$  时，如果**执行 (executing)** 某个策略  $\pi$ ，则采取动作  $a = \pi(s)$ 。同时定义策略  $\pi$  的**价值函数 (value function)** 为

$$V^\pi(s) = \mathbb{E} [R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \mid s_0 = s, \pi].$$

$V^\pi(s)$  表示从状态  $s$  开始并按照策略  $\pi$  采取动作所获得的折扣奖励的期望总和。

1

---

<sup>1</sup>请注意，这里以  $\pi$  为条件的写法并不完全正确，因为  $\pi$  不是随机变量，但这在文献中是相当标准的用法。

给定一个固定的策略  $\pi$ , 其价值函数  $V^\pi$  满足**贝尔曼方程 (Bellman equation)**:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

这表明从状态  $s$  开始的折扣奖励期望总和  $V^\pi(s)$  由两部分组成: 第一部分是从状态  $s$  开始即刻获得的**即时奖励 (immediate reward)**  $R(s)$ ; 第二部分是未来折扣奖励的期望总和。仔细考察第二项, 可以看到上面的求和项可以重写为  $E_{s' \sim P_{s\pi(s)}}[V^\pi(s')]$ 。这是从状态  $s'$  开始的折扣奖励的期望总和, 其中  $s'$  的分布由  $P_{s\pi(s)}$  给出, 也就是在 MDP 中从状态  $s$  执行第一个动作  $\pi(s)$  后将到达的状态分布。因此, 上面的第二项给出的是在 MDP 中执行第一步后获得的折扣奖励的期望总和。

贝尔曼方程可以有效地用于求解  $V^\pi$ 。具体来说, 在一个有限状态 MDP ( $|S| < \infty$ ) 中, 可以为每个状态  $s$  写出一个关于  $V^\pi(s)$  的方程。这给出了  $|S|$  个线性方程组, 其中包含  $|S|$  个变量 (未知的  $V^\pi(s)$ ), 可以有效地求解这些变量。

同样地, 定义**最优价值函数 (optimal value function)** 为

$$V^*(s) = \max_{\pi} V^\pi(s). \quad (15.1)$$

换句话说, 这是使用任何策略可以达到的最佳期望折扣奖励总和。对于最优价值函数, 也有一个贝尔曼方程:

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.2)$$

上面的第一项是即时奖励。第二项是在执行动作  $a$  之后获得的期望未来折扣奖励总和在所有动作  $a$  上的最大值。应该确保理解这个方程及其合理性。

同时定义策略  $\pi^* : S \mapsto A$  如下:

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s'). \quad (15.3)$$

注意,  $\pi^*(s)$  给出了在方程(15.2)中的”max” 中达到最大值的动作  $a$ 。

事实证明, 对于每一个状态  $s$  和每一个策略  $\pi$ , 有

$$V^*(s) = V^{\pi^*}(s) \geq V^\pi(s).$$

第一个等号表示, 对于每个状态  $s$ , 策略  $\pi^*$  的价值函数  $V^{\pi^*}$  都等于最优价值函数  $V^*$ 。此外, 不等号表示  $\pi^*$  的价值至少与任何其他策略的价值一样大。换句话说, 方程 (15.3) 所定义的  $\pi^*$  是**最优策略 (optimal policy)**。

注意， $\pi^*$  具有一个有趣的性质，即它是所有 (all) 状态  $s$  的最优策略。也就是说，不会是这样：从某个状态  $s$  开始的最优策略，与从另一个状态  $s'$  开始的最优策略不同。同一个策略  $\pi^*$  在所有 (all) 状态  $s$  下都达到了方程 (15.1) 的最大值。这意味着无论 MDP 的初始状态是什么，都可以使用相同的策略  $\pi^*$ 。

## 15.2 价值迭代与策略迭代

现在介绍两种求解有限状态 MDP 的高效算法。目前，只考虑具有有限状态空间和动作空间（即  $|S| < \infty, |A| < \infty$ ）的 MDP。在本节中，还假设已知状态转移概率  $\{P_{sa}\}$  和奖励函数  $R$ 。

第一个算法是**价值迭代** (value iteration)，如下所示：

---

**Algorithm 4:** 价值迭代

---

- 1 对于每个状态  $s$ ，初始化  $V(s) := 0$ 。
- 2 **for** 直到收敛 **do**
- 3     对于每个状态，更新

$$V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s'). \quad (15.4)$$


---

这个算法可以看作是重复使用贝尔曼方程(15.2)来更新估计的价值函数。

算法的内循环中更新  $V(s)$  的方式有两种方式。第一种是先计算每个状态  $s$  的新  $V(s)$  值，然后用新值覆盖所有旧值。这称为**同步** (synchronous) 更新。在这种情况下，算法可以看作是实现了一个“贝尔曼备份算子”，它接收价值函数的当前估计，并将其映射到一个新的估计。(详请参阅习题集。) 另一种方法是执行**异步** (asynchronous) 更新。在这种情况下，可以按某种顺序逐个状态进行循环，每次更新一个值。

在同步或异步更新下，可以证明价值迭代会使  $V$  收敛到  $V^*$ 。找到  $V^*$  后，可以使用方程(15.3)来找到最优策略。

除了价值迭代，还有另一种算法用于寻找 MDP 的最优策略。**策略迭代** (policy iteration) 算法如下：

---

**Algorithm 5:** 策略迭代
 

---

- 1 随机初始化  $\pi$ 。
- 2 **for** 直到收敛 **do**
- 3   令  $V := V^\pi$ . ▷ 通常用线性系统求解器处理
- 4   对于每个状态  $s$ , 令

$$\pi(s) := \arg \max_{a \in A} \sum_{s'} P_{sa}(s')V(s').$$

---

因此, 内循环重复计算当前策略的价值函数, 然后使用当前价值函数更新策略。(步骤 (b) 中找到的策略也称为关于  $V$  的**贪婪策略 (greedy policy)**。) 注意, 步骤 (a) 可以通过求解贝尔曼方程来实现, 如前所述, 对于一个固定的策略, 这仅仅是关于  $|S|$  个变量的  $|S|$  个线性方程组。经过该算法的有限 (*finite*) 次迭代后,  $V$  将收敛到  $V^*$ , 并且  $\pi$  将收敛到  $\pi^*$ 。<sup>2</sup>

价值迭代和策略迭代都是求解 MDP 的标准算法, 目前对于哪种算法更好还没有普遍共识。对于小型 MDP, 策略迭代通常非常快, 并且在很少的迭代次数内收敛。然而, 对于状态空间较大的 MDP, 显式求解  $V^*$  将涉及求解一个大型线性方程组, 这可能会很困难 (并且注意在策略迭代中需要多次求解线性方程组)。在这些问题中, 价值迭代可能更受欢迎。因此, 在实践中, 价值迭代似乎比策略迭代更常用。关于价值迭代和策略迭代的比较和联系的更多讨论, 请参阅第 15.5 节。

### 15.3 学习 MDP 的模型

到目前为止, 所讨论的 MDP 和求解 MDP 的算法都假设已知状态转移概率和奖励。在许多现实问题中, 我们无法直接获得状态转移概率和奖励, 而是必须从数据中估计它们。(通常, 状态空间  $S$ 、动作空间  $A$  和折扣因子  $\gamma$  是已知的。) 例如, 假设对于倒立摆问题 (参见习题集 4), 我们在 MDP 中进行了一些试验,

---

<sup>2</sup>注意, 价值迭代无法在有限次迭代中达到精确的  $V^*$ , 但策略迭代可以使用精确的线性方程组求解器, 因此可以精确收敛。这是因为当动作空间和策略空间是离散且有限时, 一旦策略迭代中的策略达到最优策略, 它将不再改变。另一方面, 尽管价值迭代会收敛到  $V^*$ , 但学习到的价值函数中总是存在一些非零误差。

过程如下：

$$\begin{aligned} s_0^{(1)} &\xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots \\ s_0^{(2)} &\xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots \\ &\dots \end{aligned}$$

这里， $s_i^{(j)}$  是在第  $j$  次试验的时刻  $i$  时的状态， $a_i^{(j)}$  是在该状态下采取的相应动作。在实践中，上述每一次试验可能会持续到 MDP 终止（例如，倒立摆问题中，摆杆倒下），或者可能运行很长但有限的时间步。

给定由多次试验组成的 MDP 中的这种“经验”，可以很容易地得出状态转移概率的最大似然估计：

$$P_{sa}(s') = \frac{\text{在状态 } s \text{ 采取动作 } a \text{ 到达状态 } s' \text{ 的次数}}{\text{在状态 } s \text{ 采取动作 } a \text{ 的次数}} \quad (15.5)$$

或者，如果上述比率是“0/0”——对应于之前从未在状态  $s$  采取动作  $a$  的情况——我们可以简单地将  $P_{sa}(s')$  估计为  $1/|S|$ 。（即，将  $P_{sa}$  估计为关于所有状态的均匀分布。）

注意，如果得到了更多经验（观察了更多试验），有一种有效的方法可以使用新的经验更新状态转移概率估计。具体来说，如果我们跟踪 (15.5) 中分子和分母的计数，那么当观察更多试验时，可以简单地累加这些计数。计算这些计数的比率即可得到对  $P_{sa}$  的估计。

使用类似的过程，如果  $R$  未知，也可以将状态  $s$  中预期即时奖励  $R(s)$  的估计值取为在状态  $s$  中观察到的平均奖励。

在学习了 MDP 的模型后，可以使用价值迭代或策略迭代来求解 MDP，使用估计的状态转移概率和奖励。例如，将模型学习和价值迭代结合起来，是学习具有未知状态转移概率的 MDP 的一种可能的算法：

1. 随机初始化  $\pi$ 。
2. 重复 {
  - (a) 在 MDP 中执行一定次数的  $\pi$ ，收集经验。
  - (b) 使用上述经验更新状态转移概率  $P_{sa}$  的估计值（如果可行的话，也更新  $R$  的估计值）。
  - (c) 根据估计的状态转移概率和奖励，使用价值迭代更新价值函数  $V$  的估计值。

(d) 更新策略  $\pi$ , 使其成为关于  $V$  的贪婪策略。

}

注意到, 对于该算法, 有一个简单的优化可以使其运行得更快。具体来说, 在算法的内层循环中, 应用价值迭代时, 如果不是将价值迭代初始化为  $V = 0$ , 而是用算法前一次迭代中找到的解进行初始化, 那么这将为价值迭代提供一个更好的起始点, 并使其更快收敛。

## 15.4 连续状态 MDP

之前我们主要关注了有限状态数量的 MDP。现在我们将讨论可能具有无限状态数量的 MDP 的算法。例如, 对于汽车, 我们可以将状态表示为  $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$ , 其中包括其位置  $(x, y)$ ; 方向  $\theta$ ; 在  $x$  和  $y$  方向上的速度  $\dot{x}$  和  $\dot{y}$ ; 以及角速度  $\dot{\theta}$ 。因此,  $S = \mathbb{R}^6$  是一个无限状态集, 因为汽车有无限多的可能位置和方向。<sup>3</sup> 类似地, 在习题集 4 中的倒立摆具有状态  $(x, \theta, \dot{x}, \dot{\theta})$ , 其中  $\theta$  是摆杆的角度。而一架在 3D 空间中飞行的直升机具有  $(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$  形式的状态, 其中横滚角  $\phi$ 、俯仰角  $\theta$  和偏航角  $\psi$  指定了直升机的 3D 方向。

在本节中, 我们将考虑状态空间为  $S = \mathbb{R}^d$  的情况, 并描述解决此类 MDP 的方法。

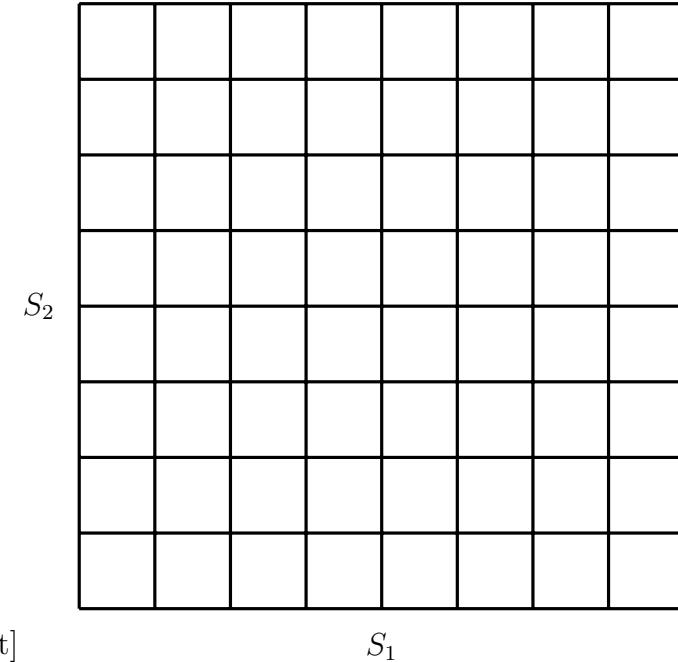
### 15.4.1 离散化

解决连续状态 MDP 的最简单方法可能是离散化状态空间, 然后使用之前描述的价值迭代或策略迭代等算法。

例如, 对于 2D 状态  $(s_1, s_2)$ , 可以使用网格来离散化状态空间:

---

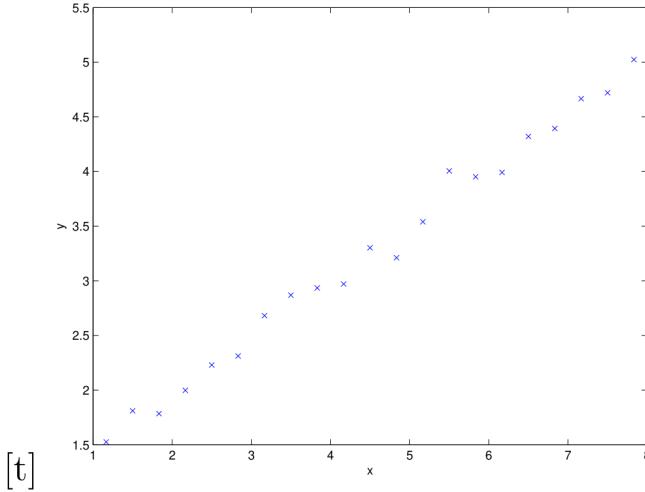
<sup>3</sup>严格来说,  $\theta$  是一个方向, 因此  $\theta$  的范围写成  $\theta \in [-\pi, \pi]$  比  $\theta \in \mathbb{R}$  更合适; 但对我们的目标而言, 这种区别并不重要。



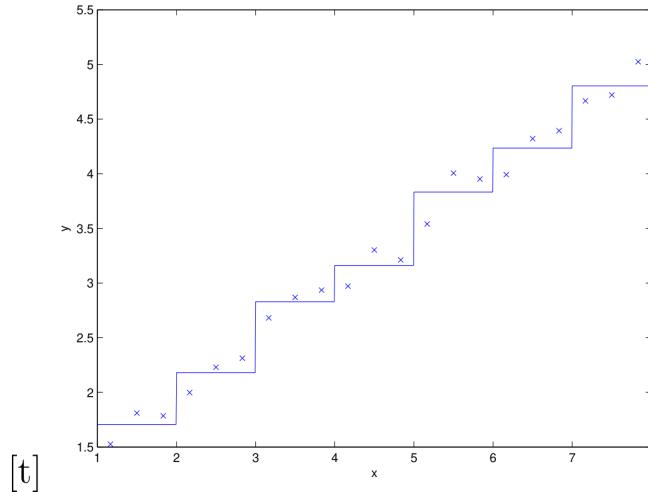
这里，每个网格单元代表一个独立的离散状态  $\bar{s}$ 。然后，可以通过一个离散状态 MDP  $(\bar{S}, A, \{P_{\bar{s}a}\}, \gamma, R)$  来近似连续状态 MDP，其中  $\bar{S}$  是离散状态的集合， $\{P_{\bar{s}a}\}$  是离散状态上的状态转移概率。然后，可以使用价值迭代或策略迭代来解出离散状态 MDP  $(\bar{S}, A, \{P_{\bar{s}a}\}, \gamma, R)$  中的  $V^*(\bar{s})$  和  $\pi^*(\bar{s})$ 。当实际系统处于某个连续值状态  $s \in S$  并且需要选择一个动作来执行时，就计算相应的离散化状态  $\bar{s}$ ，并执行动作  $\pi^*(\bar{s})$ 。

这种离散化方法对于许多问题都能很好地工作。然而，它有两个缺点。首先，它对  $V^*$ （以及  $\pi^*$ ）使用了相当简单的表示。具体来说，它假设价值函数在每个离散化区间上取一个常数值（即，价值函数在每个网格单元中是分段常数）。

为了更好地理解这种表示的局限性，考虑一个监督学习 (*supervised learning*) 问题，将函数拟合到这个数据集：



显然，线性回归可以很好地拟合这个数据集，然而，如果对  $x$  轴离散化，并使用在每个离散区间内为分段常数的表示，那么我们对数据的拟合将如下所示：



这种分段常数表示对于许多平滑函数来说并不是一个好的表示。它导致输入上的平滑性很差，并且在不同网格单元之间没有泛化能力。使用这种表示，我们需要非常精细的离散化（非常小的网格单元）才能获得良好的近似。

这种表示的第二个缺点被称为**维度诅咒 (curse of dimensionality)**。假设  $S = \mathbb{R}^d$ ，并且我们将状态的  $d$  个维度中的每一个都离散化为  $k$  个值。那么我们拥有的离散状态总数为  $k^d$ 。这在状态空间维度  $d$  中呈指数级增长，因此不适用于大型问题。例如，对于一个 10 维状态，如果我们将每个状态变量离散化为 100 个值，我们将有  $100^{10} = 10^{20}$  个离散状态，这对于现代计算机来说也大到无法表示。

根据经验，离散化通常对于 1 维和 2 维问题非常有效（而且简单且快速实

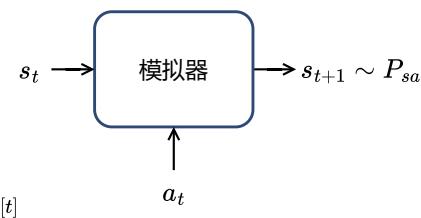
现)。也许用一些取巧的办法，并在选择离散化方法时谨慎一些，它也能适用于 4 维状态的问题。如果你非常聪明且幸运，甚至可能让它适用于某些 6 维问题。但它很少适用于更高维度的问题。

### 15.4.2 价值函数近似

我们现在介绍一种在连续状态 MDP 中寻找策略的替代方法，即直接近似  $V^*$ ，而无需诉诸离散化。这种方法被称为价值函数近似，且已成功应用于许多强化学习问题。

#### 使用模型或模拟器

为了开发值函数近似算法，我们将假设我们拥有一个用于 MDP 的模型 (model) 或模拟器 (simulator)。非正式地，模拟器是一个黑盒，它接收任何 (连续值) 状态  $s_t$  和动作  $a_t$  作为输入，并根据状态转移概率  $P_{s_t a_t}$  输出下一个状态  $s_{t+1}$  的采样。



获取此类模型有多种方法。一种是使用物理模拟。例如，习题集 4 中倒立摆的模拟器是通过使用物理定律计算在给定当前状态  $t$  和所采取的动作  $a$  的情况下，小车/杆在时间  $t + 1$  的位置和方向来获得的，前提是已知系统的所有参数，例如杆的长度、杆的质量等。或者，也可以使用现成的物理模拟软件包，该软件包将机械系统的完整物理描述、当前状态  $s_t$  和动作  $a_t$  作为输入，并在未来一小段时间内计算出系统的状态  $s_{t+1}$ 。<sup>4</sup>

获取模型的另一种方法是从 MDP 中收集的数据中学习一个模型。例如，假设执行  $n$  次试验 (trials)，每次试验重复在 MDP 中采取动作，且持续  $T$  个时间步。可以随机选择动作、执行某些特定策略或通过其他方式选择动作。然后，将

---

<sup>4</sup>Open Dynamics Engine (<http://www.ode.com>) 是一个免费/开源的物理模拟器示例，可用于模拟倒立摆等系统，并且在强化学习研究人员中一直是一个受欢迎的选择。

观察到  $n$  个状态序列，如下所示：

$$\begin{aligned} s_0^{(1)} &\xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} \dots \xrightarrow{a_{T-1}^{(1)}} s_T^{(1)} \\ s_0^{(2)} &\xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} \dots \xrightarrow{a_{T-1}^{(2)}} s_T^{(2)} \\ &\dots \\ s_0^{(n)} &\xrightarrow{a_0^{(n)}} s_1^{(n)} \xrightarrow{a_1^{(n)}} s_2^{(n)} \xrightarrow{a_2^{(n)}} \dots \xrightarrow{a_{T-1}^{(n)}} s_T^{(n)} \end{aligned}$$

然后，可以应用学习算法来预测  $s_{t+1}$  作为  $s_t$  和  $a_t$  的函数。

例如，可以选择学习一个如下形式的线性模型：

$$s_{t+1} = As_t + Ba_t, \quad (15.6)$$

使用类似于线性回归的算法。这里，模型的参数是矩阵  $A$  和  $B$ ，并且可以通过从  $n$  次试验中收集的数据来估计它们：

$$\arg \min_{A,B} \sum_{i=1}^n \sum_{t=0}^{T-1} \|s_{t+1}^{(i)} - (As_t^{(i)} + Ba_t^{(i)})\|^2.$$

也可以使用其他损失函数来学习模型。例如，最近 Luo et al. 2018 的工作发现，使用  $\|\cdot\|_2$  范数（不带平方）在某些情况下可能有所帮助。

在学习了  $A$  和  $B$  之后，一个选项是构建一个**确定性 (deterministic)** 模型，其中给定输入  $s_t$  和  $a_t$ ，输出  $s_{t+1}$  被精确确定。具体来说，总是根据公式 (15.6) 计算  $s_{t+1}$ 。或者，也可以构建一个**随机 (stochastic)** 模型，其中  $s_{t+1}$  是输入的随机函数，通过将其建模为：

$$s_{t+1} = As_t + Ba_t + \epsilon_t,$$

其中  $\epsilon_t$  是一个噪声项，通常建模为  $\epsilon_t \sim \mathcal{N}(0, \Sigma)$ 。（协方差矩阵  $\Sigma$  也可以以直接的方式从数据中估计。）

这里将下一个状态  $s_{t+1}$  写成当前状态和动作的线性函数；当然，这也可以是非线性函数。具体来说，可以学习一个模型  $s_{t+1} = A\phi_s(s_t) + B\phi_a(a_t)$ ，其中  $\phi_s$  和  $\phi_a$  是状态和动作的一些非线性特征映射。或者，也可以使用非线性学习算法，例如局部加权线性回归，来估计  $s_{t+1}$  作为  $s_t$  和  $a_t$  的函数。这些方法都可以用来构建 MDP 的确定性或随机模拟器。

## 拟合价值迭代

现在介绍用于近似连续状态 MDP 价值函数的**拟合价值迭代 (fitted value iteration)** 算法。在下文中，我们将假设问题具有连续状态空间  $S = \mathbb{R}^d$ ，但动

作空间  $A$  较小且离散。<sup>5</sup>

回顾一下，价值迭代执行以下更新：

$$V(s) := R(s) + \gamma \max_a \int_{s'} P_{sa}(s') V(s') ds' \quad (15.7)$$

$$= R(s) + \gamma \max_a \mathbb{E}_{s' \sim P_{sa}} [V(s')] \quad (15.8)$$

(在第 15.2 节中，价值迭代更新写为  $V(s) := R(s) + \gamma \max_a \sum_{s'} P_{sa}(s') V(s')$  而不是状态上的积分；新的符号反映了现在正在处理连续状态而不是离散状态。)

拟合价值迭代的主要思想是，在有限样本状态  $s^{(1)}, \dots, s^{(n)}$  上近似地执行此步骤。具体来说，下面使用监督学习算法——线性回归——来近似状态的价值函数，作为状态的线性或非线性函数：

$$V(s) = \theta^T \phi(s).$$

这里， $\phi$  是状态的某种适当的特征映射。

对于具有的有限样本  $n$  个状态中的每个状态  $s$ ，拟合价值迭代将首先计算一个量  $y^{(i)}$ ，是  $R(s) + \gamma \max_a \mathbb{E}_{s' \sim P_{sa}} [V(s')]$  的近似（公式 (15.8) 的右侧）。然后将应用监督学习算法，尝试使  $V(s)$  接近  $R(s) + \gamma \max_a \mathbb{E}_{s' \sim P_{sa}} [V(s')]$ （换句话说，尝试使  $V(s)$  接近  $y^{(i)}$ ）。

详细来说，该算法如下：

1. 随机采样  $n$  个状态  $s^{(1)}, s^{(2)}, \dots, s^{(n)} \in S$ 。
2. 初始化  $\theta := 0$ 。
3. 重复 {

对于  $i = 1, \dots, n$  {

对于每个动作  $a \in A$  {

采样  $s'_1, \dots, s'_k \sim P_{s^{(i)}a}$ （使用某种 MDP 模型）。

令  $q(a) = \frac{1}{k} \sum_{j=1}^k R(s^{(i)}) + \gamma V(s'_j)$ 。

// 因此， $q(a)$  是  $R(s^i) + \gamma \mathbb{E}_{s' \sim P_{s^{(i)}a}} [V(s')]$  的估计值。

}

令  $y^{(i)} = \max_a q(a)$ 。

<sup>5</sup>在实践中，大多数 MDP 的动作空间远小于状态空间。例如，汽车状态空间有 6 维而动作空间则是 2 维（转向和速度控制）；倒立摆状态空间有 4 维而动作空间只有 1 维；直升机则是 12 维状态空间和 4 维动作空间。因此，离散化动作空间通常比离散化状态空间的问题要小得多。

```

// 因此,  $y^{(i)}$  是  $R(s^i) + \gamma \max_a \mathbb{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$  的估计值。
}

// 原始的 (用于离散状态的) 价值迭代算法
// 根据  $V(s^{(i)}) := y^{(i)}$  更新价值函数。
// 此算法则希望  $V(s^{(i)}) \approx y^{(i)}$ 
// 可以使用监督学习 (线性回归) 达成这点。
令  $\theta := \arg \min_{\theta} \frac{1}{2} \sum_{i=1}^n (\theta^T \phi(s^{(i)}) - y^{(i)})^2$ 。
}

```

如上所述, 我们已经详细阐述了使用线性回归的拟合价值迭代算法, 以使  $V(s^{(i)})$  接近  $y^{(i)}$ 。算法的这一步骤完全类似于标准的监督学习 (回归) 问题, 其中有一个训练集  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})$ , 并且希望学习一个从  $x$  到  $y$  的函数; 唯一的区别是这里  $s$  扮演了  $x$  的角色。尽管上面的描述使用了线性回归, 但显然也可以使用其他回归算法 (例如局部加权线性回归)。

与离散状态集上的价值迭代不同, 拟合价值迭代不能被证明总是收敛的。然而, 它在实践中通常能收敛 (或近似收敛), 并且对许多问题都表现良好。还需要注意的是, 如果使用 MDP 的确定性模拟器/模型, 那么可以通过在算法中设置  $k = 1$  来简化拟合价值迭代。这是因为公式 (15.8) 中的期望变为确定性分布上的期望, 因此一个单一的样本就足以精确计算该期望。否则, 在上述算法中, 需要抽取  $k$  个样本并取平均值来近似该期望 (参见算法伪代码中  $q(a)$  的定义)。

最后, 拟合价值迭代输出  $V$  作为  $V^*$  的近似。这隐式地定义了我们的策略。具体来说, 当系统处于某个状态  $s$ , 需要选择一个动作时, 我们希望选择以下动作:

$$\arg \max_a \mathbb{E}_{s' \sim P_{sa}}[V(s')] \quad (15.9)$$

计算/近似此过程类似于拟合价值迭代的内循环, 其中对于每个动作, 我们从  $P_{sa}$  中采样  $s'_1, \dots, s'_k$  来近似期望。(同样, 如果模拟器是确定性的, 我们可以设置  $k = 1$ 。)

在实践中, 通常还有其他方法来近似此步骤。例如, 一个非常常见的情况是模拟器具有  $s_{t+1} = f(s_t, a_t) + \epsilon_t$  的形式, 其中  $f$  是状态的某个确定性函数 (例如  $f(s_t, a_t) = As_t + Ba_t$ ),  $\epsilon_t$  是零均值高斯噪声。在这种情况下, 我们可以通过以下方式选择动作:

$$\arg \max_a V(f(s, a)).$$

换句话说，这里是设置  $\epsilon_t = 0$ （即，忽略模拟器中的噪声），并设置  $k = 1$ 。等效地，这可以从公式 (15.9) 使用以下近似导出：

$$\mathbb{E}_{s'}[V(s')] \approx V(\mathbb{E}_{s'}[s']) \quad (15.10)$$

$$= V(f(s, a)), \quad (15.11)$$

其中期望是关于随机变量  $s' \sim P_{sa}$  的。只要噪声项  $\epsilon_t$  很小，这通常是一个合理的近似。

然而，对于那些不适合这种近似的问题，为了近似上述期望，需要使用模型对  $k|A|$  个状态进行采样，这在计算上可能非常昂贵。

## 15.5 策略与价值的联系 (选读)

---

### Algorithm 6: 策略迭代的变体

---

- 1 过程  $VE(\pi, k)$  ▷ 用于评估  $V^\pi$
  - 2 选项 1：初始化  $V := 0$ ；选项 2：使用主算法的当前  $V$  进行初始化。
  - 3 **for**  $i = 0$  **to**  $k - 1$  **do**
  - 4     对于每个状态  $s$ ，更新
- $$V(s) := R(s) + \gamma \sum_{s'} P_{s\pi(s)}(s')V(s'). \quad (15.12)$$
- 5 **返回**  $V$
  - 6
  - 7 输入 超参数  $k$ 。
  - 8 随机初始化  $\pi$ 。
  - 9 **for** 直到收敛 **do**
  - 10   令  $V := VE(\pi, k)$ .
  - 11   对于每个状态  $s$ ，令

$$\pi(s) := \arg \max_{a \in A} \sum_{s'} P_{sa}(s')V(s'). \quad (15.13)$$

---

在策略迭代算法 5 的第 3 行，通常用线性系统求解器来计算  $V^\pi$ 。或者也可以使用迭代贝尔曼更新来评估  $V^\pi$ ，这类似于价值迭代，如算法 6 中过程  $VE(\cdot)$  的第 1 行所示。如果我们在过程  $VE$  的第 2 行中选择选项 1，那么过程  $VE$  与价值迭代（算法 4）的不同之处在其第 4 行：过程  $VE$  用  $\pi$  中的动作，而价值迭代

则使用贪婪动作。

用过程 VE 可以构建算法 6，它是策略迭代的一种变体，作为连接策略迭代和价值迭代的中间算法。这里我们在 VE 中选用选项 2，以最大化重用之前所学的知识。可以验证，如果取  $k = 1$  并在算法 6 的第 2 行中选用选项 2，那么算法 6 在语义上等同于价值迭代（算法 4）。换句话说，算法 6 和价值迭代交错更新 (15.13) 和 (15.12)。算法 6 在更新  $k$  步 (15.12) 和一步 (15.13) 之间交替，而价值迭代在更新一步 (15.12) 和一步 (15.13) 之间交替。因此，算法 6 通常不会比价值迭代更快，因为假设更新 (15.12) 和 (15.13) 效用和耗时相同，那么更新频率的最佳平衡可能只是  $k = 1$  或  $k \approx 1$ 。

另一方面，如果更新  $k$  步 (15.12) 可以比更新一步 (15.12)  $k$  次快得多，那么多求几步 (15.12) 可能有用。这就是策略迭代所利用的——线性系统求解器求解  $k = \infty$  的 VE 比求解较大的  $k$  的 VE 会快得多。反之，如果不存在这种加速效果，例如当状态空间很大且线性系统求解器也不快时，价值迭代更可取。

# 第 16 章 LQR, DDP 与 LQG

## 16.1 有限时间范围的 MDP

第 15 章中定义了马尔可夫决策过程，并讨论了简化设置下的价值迭代/策略迭代。具体而言，引入了定义最优策略  $\pi^*$  的最优价值函数  $V^*$  的**最优贝尔曼方程 (optimal Bellman equation)**。

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

回想一下，从最优价值函数中能恢复最优策略  $\pi^*$ ，如下所示：

$$\pi^*(s) = \operatorname{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

在本章中，将采用更一般的设置：

1. 为了同时适用于离散和连续的情况。因此，将使用

$$\begin{aligned} & \mathbb{E}_{s' \sim P_{sa}} [V^{\pi^*}(s')] \quad \text{而非} \\ & \sum_{s' \in S} P_{sa}(s') V^{\pi^*} \end{aligned}$$

这意味着将取下一状态价值函数的期望。在离散情况下，可以将期望改写为对所有状态的求和。在连续情况下，可以将期望改写为对所有状态的积分。符号  $s' \sim P_{sa}$  表示  $s'$  是从分布  $P_{sa}$  中采样的。

2. 假设奖励同时取决于**状态和动作 (both states and actions)**。换句话说， $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ 。这意味着计算最优动作的先前机制变为

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} R(s, a) + \gamma \mathbb{E}_{s' \sim P_{sa}} [V^{\pi^*}(s')]$$

3. 不考虑无限时间范围的 MDP，而是假设有一个**有限时间范围的 MDP (finite horizon MDP)**，定义为一个元组

$$(\mathcal{S}, \mathcal{A}, P_{sa}, T, R)$$

其中  $T > 0$  是**时间范围 (time horizon)**（例如  $T = 100$ ）。在这种设置下，对回报的定义将略有不同：

$$R(s_0, a_0) + R(s_1, a_1) + \cdots + R(s_T, a_T)$$

而不是（无限时间范围情况下的）

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots + \sum_{t=0}^{\infty} R(s_t, a_t) \gamma^t$$

折现因子  $\gamma$  呢？(*what happened to the discount factor  $\gamma$ ?*) 请记住，引入  $\gamma$  (部分原因) 是为了确保无限项求和是有限且良定义的。如果奖励以常数  $\bar{R}$  为界，则回报是以下式为界的

$$\left| \sum_{t=0}^{\infty} R(s_t) \gamma^t \right| \leq \bar{R} \sum_{t=0}^{\infty} \gamma^t$$

发现这里是一个几何级数！在这里，由于回报是有限项求和，折现因子  $\gamma$  不再是必需的。

在这种新设置中，情况表现得相当不同。首先，最优策略  $\pi^*$  可能是非平稳的，这意味着**它随时间变化 (it changes over time)**。换句话说，现在有

$$\pi^{(t)} : \mathcal{S} \rightarrow \mathcal{A}$$

其中上标  $(t)$  表示时间步  $t$  的策略。遵循策略  $\pi^{(t)}$  的有限时间范围 MDP 的动态过程如下：从某个状态  $s_0$  开始，根据时间步 0 的策略  $\pi^{(0)}(s_0)$  采取某个动作  $a_0 := \pi^{(0)}(s_0)$ 。MDP 根据  $P_{s_0 a_0}$  转换到后继状态  $s_1$ 。然后根据时间步 1 的新策略  $\pi^{(1)}(s_1)$  选择另一个动作  $a_1 := \pi^{(1)}(s_1)$ ，依此类推……

为什么最优策略在有限时间范围设置下会是非平稳的？(*why does the optimal policy happen to be non-stationary in the finite-horizon setting?*) 直观地说，由于有有限数量的动作要执行，实际可能希望根据所处的环境位置以及剩余的时间来采取不同的策略。想象一个有 2 个目标的网格，奖励分别为  $+1$  和  $+10$ 。一开始可能希望采取行动以得到  $+10$ 。但如果经过一些步骤，动态过程以某种方式接近了  $+1$ ，并且没有足够的剩余步数来达到  $+10$ ，那么更好的策略将是瞄准  $+1$ ……

#### 4. 这一现象导致了时间依赖的动态过程 (time dependent dynamics)

$$s_{t+1} \sim P_{s_t, a_t}^{(t)}$$

这意味着转移分布  $P_{s_t, a_t}^{(t)}$  随时间变化。关于  $R^{(t)}$  也是一样。请注意，这种设置更好地模拟了现实生活。例如在汽车驾驶时，油箱会变空，交通状况会变化等等。综上，对有限时间范围 MDP 使用以下一般表述

$$(\mathcal{S}, \mathcal{A}, P_{sa}^{(t)}, T, R^{(t)})$$

**备注.** 注意上式等价于将时间加入到状态中。

在时间  $t$  处，策略  $\pi$  的价值函数定义方式与之前相同，即从状态  $s$  开始，遵循策略  $\pi$  生成的轨迹的期望：

$$V_t(s) = \mathbb{E}[R^{(t)}(s_t, a_t) + \dots + R^{(T)}(s_T, a_T) | s_t = s, \pi]$$

现在，问题是

如何在有限时间范围设置下找到最优价值函数

$$V_t^*(s) = \max_{\pi} V_t^{\pi}(s)$$

事实上，贝尔曼方程的价值迭代是为了**动态规划 (Dynamic Programming)** 而设计的。这不足为奇，因为贝尔曼是动态规划的创始人之一，而贝尔曼方程与该领域密切相关。为了理解如何通过采用基于迭代的方法来简化问题，有如下观察：

1. 注意，在博弈结束时（对于时间步  $T$ ），最优值是显而易见的

$$\forall s \in \mathcal{S} : V_T^*(s) := \max_{a \in \mathcal{A}} R^{(T)}(s, a) \quad (16.1)$$

2. 对于另一个时间步  $0 \leq t < T$ ，如果假设已知下一个时间步  $V_{t+1}^*$  的最优价值函数，那么就有

$$\forall t < T, s \in \mathcal{S} : V_t^*(s) := \max_{a \in \mathcal{A}} \left[ R^{(t)}(s, a) + \mathbb{E}_{s' \sim P_{sa}^{(t)}} [V_{t+1}^*(s')] \right] \quad (16.2)$$

考虑上述结果，可以提出一个巧妙的算法来解决最优价值函数：

1. 使用公式 (16.1) 计算  $V_T^*$ 。

2. 对于  $t = T - 1, \dots, 0$ :

使用  $V_{t+1}^*$  和公式 (16.2) 计算  $V_t^*$ 。

**旁注.** 可以将标准价值迭代视作这种一般情况的特例（即不将时间视为状态的一部分）。结果表明，在标准设置中，如果运行价值迭代  $T$  步，将得到最优价值迭代的  $\gamma^T$  近似（几何收敛）。下面结果的证明请参见习题集 4：

**定理.** 设  $B$  是贝尔曼更新算子，然后记  $\|f(x)\|_\infty := \sup_x |f(x)|$ 。令  $V_t$  表示第  $t$  步的价值函数。则

$$\begin{aligned}\|V_{t+1} - V^*\|_\infty &= \|B(V_t) - V^*\|_\infty \\ &\leq \gamma \|V_t - V^*\|_\infty \\ &\leq \gamma^t \|V_1 - V^*\|_\infty\end{aligned}$$

所以贝尔曼更新算子  $B$  是一个  $\gamma$ -收缩算子。

## 16.2 线性二次调节器 (LQR)

本节将讨论第 16.1 节所述的有限时间范围设定中的一个特例，其**精确解 (exact solution)** 是可（容易）处理的。该模型在机器人中被广泛使用。

首先，描述该模型的假设。考虑状态和动作是连续的情况，其中

$$\mathcal{S} = \mathbb{R}^d, \mathcal{A} = \mathbb{R}^d$$

并且假设**线性转移 (linear transitions)**（带噪声）

$$s_{t+1} = A_t s_t + B_t a_t + w_t$$

其中  $A_t \in \mathbb{R}^{d \times d}$ ,  $B_t \in \mathbb{R}^{d \times d}$  是矩阵， $w_t \sim \mathcal{N}(0, \Sigma_t)$  是某种高斯噪声（均值为零 (zero)）。正如后文所示，只要噪声具有零均值，它就不会影响最优策略！

还假设模型具有**二次奖励 (quadratic rewards)**

$$R^{(t)}(s_t, a_t) = -s_t^\top U_t s_t - a_t^\top W_t a_t$$

其中  $U_t \in \mathbb{R}^{d \times n}$ ,  $W_t \in \mathbb{R}^{d \times d}$  是正定矩阵（意味着奖励总是负 (negative) 的）。

**备注.** 请注意，奖励的二次性等价于希望状态接近原点（以获得较高奖励）。例如，如果  $U_t = I_d$ （单位矩阵）和  $W_t = I_d$ ，则  $R_t = -\|s_t\|^2 - \|a_t\|^2$ ，这意味着希望采取平滑的动作 ( $a_t$  的范数小) 以回到原点 ( $s_t$  的范数小)。这可以模拟一辆汽车试图保持在车道中间而不进行冲动性移动……

定义了 LQR 模型的假设之后，接下来介绍 LQR 算法的 2 个步骤。

**步骤 1** 假设不知道矩阵  $A, B, \Sigma$ 。为了估计它们，可以借鉴强化学习一章中价值逼近部分的思想。首先，从任意策略中收集转移。然后，使用线性回归来找到  $\arg \min_{A, B} \sum_{i=1}^n \sum_{t=0}^{T-1} |s_{t+1}^{(i)} - (As_t^{(i)} + Ba_t^{(i)})|^2$ 。最后，使用高斯判别分析一节中学到的技术来学习  $\Sigma$ 。

**步骤 2** 假设模型参数已知（给定或通过步骤 1 估计），可以使用动态规划推导出最优策略。

换句话说，给定

$$\begin{cases} s_{t+1} & = As_t + Ba_t + w_t \quad A_t, B_t, U_t, W_t, \Sigma_t \text{ 已知} \\ R^{(t)}(s_t, a_t) & = -s_t^\top U_t s_t - a_t^\top W_t a_t \end{cases}$$

计算  $V_t^*$ 。根据第 16.1 节，可以应用动态规划，得到

### 1. 初始化

对于最后一个时间步  $T$ ，

$$\begin{aligned} V_T^*(s_T) &= \max_{a_T \in \mathcal{A}} R_T(s_T, a_T) \\ &= \max_{a_T \in \mathcal{A}} -s_T^\top U_T s_T - a_T^\top W_T a_T \\ &= -s_T^\top U_T s_T \quad (\text{最大化时 } a_T = 0) \end{aligned}$$

### 2. 迭代步骤

令  $t < T$ 。假设已知  $V_{t+1}^*$ 。

事实 1：可以证明，如果  $V_{t+1}^*$  是  $s_t$  的二次函数，则  $V_t^*$  也是二次函数。

换句话说，存在某个矩阵  $\Phi$  和某个标量  $\Psi$  使得下式成立

$$\begin{aligned} \text{如果 } V_{t+1}^*(s_{t+1}) &= s_{t+1}^\top \Phi_{t+1} s_{t+1} + \Psi_{t+1} \\ \text{则 } V_t^*(s_t) &= s_t^\top \Phi_t s_t + \Psi_t \end{aligned}$$

对于时间步  $t = T$ ，有  $\Phi_T = -U_T$  且  $\Psi_T = 0$ 。

事实 2：可以证明，最优策略是状态的线性函数。已知  $V_{t+1}^*$  等价于已知  $\Phi_{t+1}$  和  $\Psi_{t+1}$ ，因此只需要解释如何从  $\Phi_{t+1}$  和  $\Psi_{t+1}$  以及问题的其他参数计算  $\Phi_t$  和  $\Psi_t$ 。

$$\begin{aligned} V_t^*(s_t) &= s_t^\top \Phi_t s_t + \Psi_t \\ &= \max_{a_t} \left[ R^{(t)}(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P_{s_t, a_t}^{(t)}} [V_{t+1}^*(s_{t+1})] \right] \\ &= \max_{a_t} \left[ -s_t^\top U_t s_t - a_t^\top V_t a_t + \mathbb{E}_{s_{t+1} \sim \mathcal{N}(A_t s_t + B_t a_t, \Sigma_t)} [s_{t+1}^\top \Phi_{t+1} s_{t+1} + \Psi_{t+1}] \right] \end{aligned}$$

其中第二行是最优值函数的定义，第三行是通过将模型动态过程以及二次假设代入得到的。注意到最后一个表达式是  $a_t$  的二次函数，因此可以（容易地）优化。<sup>1</sup> 得到最优动作  $a_t^*$

$$\begin{aligned} a_t^* &= [(B_t^\top \Phi_{t+1} B_t - W_t)^{-1} B_t^\top \Phi_{t+1} A_t] \cdot s_t \\ &= L_t \cdot s_t \end{aligned}$$

其中

$$L_t := [(B_t^\top \Phi_{t+1} B_t - W_t)^{-1} B_t^\top \Phi_{t+1} A_t]$$

这是一个令人印象深刻的结果：最优策略对  $s_t$  是线性 (linear) 的。给定  $a_t^*$ ，可以求解  $\Phi_t$  和  $\Psi_t$ 。最终得到离散里卡蒂方程 (Discrete Riccati equations)

$$\begin{aligned} \Phi_t &= A_t^\top (\Phi_{t+1} - \Phi_{t+1} B_t (B_t^\top \Phi_{t+1} B_t - W_t)^{-1} B_t^\top \Phi_{t+1}) A_t - U_t \\ \Psi_t &= -\text{tr}(\Sigma_t \Phi_{t+1}) + \Psi_{t+1} \end{aligned}$$

事实 3：注意到  $\Phi_t$  既不依赖于  $\Psi$  也不依赖于噪声  $\Sigma_t$ ！由于  $L_t$  是  $A_t, B_t$  和  $\Phi_{t+1}$  的函数，这意味着最优策略也不依赖于噪声 (does not depend on the noise)！（但是  $\Psi_t$  确实依赖于  $\Sigma_t$ ，这意味着  $V_t^*$  依赖于  $\Sigma_t$ 。）

总结一下，LQR 算法的工作方式如下：

1. (如果需要) 估计参数  $A_t, B_t, \Sigma_t$ 。
2. 初始化  $\Phi_T := -U_T$  和  $\Psi_T := 0$ 。
3. 迭代更新  $t = T-1 \dots 0$  时的  $\Phi_t$  和  $\Psi_t$ ，使用离散里卡蒂方程。如果存在使状态趋于零的策略，则收敛性得到保证！

利用事实 3，可以更巧妙地使算法运行得（稍微）快一些！由于最优策略不依赖于  $\Psi_t$ ，并且  $\Phi_t$  的更新仅依赖于  $\Phi_t$ ，因此仅更新  $\Phi_t$  就足够了！

### 16.3 从非线性动态过程到 LQR

事实证明，就算动态过程是非线性的，许多问题也可以归结为 LQR。虽然 LQR 是一个很好的公式，因为可以得到一个精确的解析解，但它远非通用。以倒立摆为例，状态之间的转换如下：

---

<sup>1</sup>对  $a_t$  求导并令导数等于零。

$$\begin{pmatrix} x_{t+1} \\ \dot{x}_{t+1} \\ \theta_{t+1} \\ \dot{\theta}_{t+1} \end{pmatrix} = F \left( \begin{pmatrix} x_t \\ \dot{x}_t \\ \theta_t \\ \dot{\theta}_t \end{pmatrix}, a_t \right)$$

其中函数  $F$  取决于角度的余弦等。现在，一个问题是：

能把这个系统线性化吗？

### 16.3.1 动态过程的线性化

假设在时间  $t$ ，系统大部分时间处于状态  $\bar{s}_t$ ，并且所执行的动作在  $\bar{a}_t$  附近。对于倒立摆，如果达到了某种最优状态，这意味着：动作很小，且与垂直方向偏离不大。

使用泰勒展开来使动态过程线性化。在状态是一维且转移函数  $F$  不依赖于动作的简单情况下，可以写成：

$$s_{t+1} = F(s_t) \approx F(\bar{s}_t) + F'(\bar{s}_t) \cdot (s_t - \bar{s}_t)$$

更一般的情况用梯度代替了简单的导数：

$$s_{t+1} \approx F(\bar{s}_t, \bar{a}_t) + \nabla_s F(\bar{s}_t, \bar{a}_t) \cdot (s_t - \bar{s}_t) + \nabla_a F(\bar{s}_t, \bar{a}_t) \cdot (a_t - \bar{a}_t) \quad (16.3)$$

现在， $s_{t+1}$  对  $s_t$  和  $a_t$  是线性的，因为可以将方程(16.3)重写为

$$s_{t+1} \approx As_t + Bs_t + \kappa$$

其中  $\kappa$  是某个常数， $A, B$  是矩阵。这种写法与 LQR 的假设非常相似，只需要消除常数项  $\kappa$ ！结果表明，可以通过人为地增加一个维度，将常数项吸收到  $s_t$  中。这与在课程开始时用于线性回归的技巧相同…

### 16.3.2 微分动态规划 (DDP)

之前的方法适用于目标是保持在某个状态  $s^*$  附近的情况（例如倒立摆，或者汽车需要保持在车道中间）。然而，某些情况下，目标可能更复杂。

下面介绍一种方法，其适用于系统需要遵循某个轨迹的情况（例如火箭）。该方法将把轨迹离散化为离散时间步，并创建中间目标，然后就可以使用之前介绍的技术！这种方法称为**微分动态规划 (Differential Dynamic Programming)**。主要步骤是：

**步骤 1** 用朴素的控制器生成一个标称轨迹，该轨迹近似于需要遵循的轨迹。换句话说，控制器能够通过以下方式近似最优轨迹：

$$s_0^*, a_0^* \rightarrow s_1^*, a_1^* \rightarrow \dots$$

**步骤 2** 在每个轨迹点  $s_t^*$  附近进行线性化，即

$$s_{t+1} \approx F(s_t^*, a_t^*) + \nabla_s F(s_t^*, a_t^*) \cdot (s_t - s_t^*) + \nabla_a F(s_t^*, a_t^*) \cdot (a_t - a_t^*)$$

其中  $s_t, a_t$  是当前状态和动作。现在，对这些点进行线性近似后，用上一节的方法并重写作：

$$s_{t+1} = A_t \cdot s_t + B_t \cdot a_t$$

(注意，在这种情况下，使用了开头提到的非平稳动态过程设置)

**注意**，可以对奖励  $R^{(t)}$  作类似推导，使用二阶泰勒展开：

$$\begin{aligned} R(s_t, a_t) \approx & R(s_t^*, a_t^*) + \nabla_s R(s_t^*, a_t^*)(s_t - s_t^*) + \nabla_a R(s_t^*, a_t^*)(a_t - a_t^*) \\ & + \frac{1}{2}(s_t - s_t^*)^\top H_{ss}(s_t - s_t^*) + (s_t - s_t^*)^\top H_{sa}(a_t - a_t^*) \\ & + \frac{1}{2}(a_t - a_t^*)^\top H_{aa}(a_t - a_t^*) \end{aligned}$$

其中  $H_{xy}$  表示  $R$  对  $x$  和  $y$  在  $(s_t^*, a_t^*)$  处求值的 Hessian 矩阵的元素。这个表达式可以改写为：

$$R_t(s_t, a_t) = -s_t^\top U_t s_t - a_t^\top W_t a_t$$

对矩阵  $U_t, W_t$  也用了添加额外维度的方法。为了完成推导，请注意：

$$\begin{pmatrix} 1x \end{pmatrix} \cdot \begin{pmatrix} ab \\ bc \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x \end{pmatrix} = a + 2bx + cx^2$$

**步骤 3** 现在，可以确信问题已经严格地 (strictly) 在 LQR 框架下重写。下面使用 LQR 寻找最优策略  $\pi_t$ 。因此，新的控制器（有望）表现更好！

**注意**，如果 LQR 轨迹与轨迹的线性化近似偏离过大，可能会出现一些问题，但这可以通过奖励整形来解决。

**步骤 4** 得到新的控制器（新的策略  $\pi_t$ ）后，使用它来生成新的轨迹：

$$s_0^*, \pi_0(s_0^*) \rightarrow s_1^*, \pi_1(s_1^*) \rightarrow \dots \rightarrow s_T^*$$

请注意，生成这个新轨迹时，使用真实的  $F$  而不是其线性近似来计算状态转移，这意味着：

$$s_{t+1}^* = F(s_t^*, a_t^*)$$

然后，返回步骤 2 并重复，直到满足某个停止条件。

## 16.4 线性二次高斯 (LQG)

在实际应用中，通常无法得到完整的状态  $s_t$ 。例如，自动驾驶汽车可能从摄像头接收图像，这仅仅是一个观测 (observation)，而非世界的完整状态。到目前为止都假设状态是可得到的。这对于大多数实际问题可能不成立，因此需要一种新的工具来建模这种情况：部分可观测马尔可夫决策过程 (Partially Observable MDPs, POMDP)。

POMDP 是一种带有额外观测层的 MDP。换句话说，引入了一个新的变量  $o_t$ ，它遵循给定当前状态  $s_t$  的某种条件分布：

$$o_t | s_t \sim O(o|s)$$

形式上，有限时间范围的 POMDP 由一个元组定义：

$$(\mathcal{S}, \mathcal{O}, \mathcal{A}, P_{sa}, T, R)$$

在此框架内，一般策略是基于观测  $o_1, \dots, o_t$  维护一个信念状态 (belief state) (状态上的分布)。然后，POMDP 中的策略将这些信念状态映射到动作。

在本节中，将 LQR 扩展到这种新设置。假设观测到  $y_t \in \mathbb{R}^n$ ，其中  $m < n$ ，且满足：

$$\begin{cases} y_t = C \cdot s_t + v_t \\ s_{t+1} = A \cdot s_t + B \cdot a_t + w_t \end{cases}$$

其中  $C \in \mathbb{R}^{n \times d}$  是一个压缩矩阵， $v_t$  是高斯传感器噪声 (与  $w_t$  类似)。请注意，奖励函数  $R^{(t)}$  保持不变，它是一个关于状态 (而非观测) 和动作的函数。此外，由于分布是高斯的，信念状态也将是高斯的。下面将概述在这个新框架下，用于寻找最优策略的策略：

**步骤 1** 首先，根据已有的观测值计算可能状态 (信念状态) 的分布。换句话说，需要计算  $s_{t|t}$  的均值和协方差  $\Sigma_{t|t}$ ：

$$s_t | y_1, \dots, y_t \sim \mathcal{N}(s_{t|t}, \Sigma_{t|t})$$

为了高效地执行跨时间的计算，将使用卡尔曼滤波 (Kalman Filter) 算法 (阿波罗登月舱上使用的算法！)。

**步骤 2** 有了分布之后，使用均值  $s_{t|t}$  作为  $s_t$  的最优近似。

**步骤 3** 然后令动作  $a_t := L_t s_{t|t}$ ，其中  $L_t$  来自常规 LQR 算法。

为了直观地理解其工作原理, 请注意  $s_{t|t}$  是  $s_t$  的带噪声近似 (相当于在 LQR 中添加更多噪声), 但已证明 LQR 不受噪声影响!

步骤 1 需要详细说明。将介绍一个简单的例子, 其中动态过程中没有动作依赖 (但一般情况遵循相同的思想)。假设:

$$\begin{cases} s_{t+1} = A \cdot s_t + w_t, & w_t \sim \mathcal{N}(0, \Sigma_s) \\ y_t = C \cdot s_t + v_t, & v_t \sim \mathcal{N}(0, \Sigma_y) \end{cases}$$

由于噪声是高斯分布, 可以很容易证明联合分布也是高斯分布。

$$\begin{pmatrix} s_1 \\ \vdots \\ s_t \\ y_1 \\ \vdots \\ y_t \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma) \quad \text{对于某些 } \mu, \Sigma$$

然后, 使用高斯分布的边缘分布公式 (参见因子分析), 可以得到:

$$s_t | y_1, \dots, y_t \sim \mathcal{N}(s_{t|t}, \Sigma_{t|t})$$

然而, 使用这些公式计算边缘分布参数的计算成本会非常高昂! 它需要操作形状为  $t \times t$  的矩阵。回想一下, 矩阵求逆的计算复杂度为  $O(t^3)$ , 并且必须在每个时间步重复执行, 导致总成本为  $O(t^4)$ !

**卡尔曼滤波 (Kalman filter)** 算法提供了一种更好的方法, 可以在对  $t$  的常数时间 (constant time) 内更新均值和方差! 卡尔曼滤波器有两个基本步骤。假设已知  $s_t | y_1, \dots, y_t$  的分布:

**预测步骤 (predict step):** 计算  $s_{t+1} | y_1, \dots, y_t$

**更新步骤 (update step):** 计算  $s_{t+1} | y_1, \dots, y_{t+1}$

迭代执行这些时间步! 预测和更新步骤的组合更新了信念状态。换句话说, 该过程如下所示:

$$(s_t | y_1, \dots, y_t) \xrightarrow{\text{预测}} (s_{t+1} | y_1, \dots, y_t) \xrightarrow{\text{更新}} (s_{t+1} | y_1, \dots, y_{t+1}) \xrightarrow{\text{预测}} \dots$$

**预测步骤:** 假设已知分布如下:

$$s_t | y_1, \dots, y_t \sim \mathcal{N}(s_{t|t}, \Sigma_{t|t})$$

那么，关于下一个状态的分布也是高斯分布：

$$s_{t+1}|y_1, \dots, y_t \sim \mathcal{N}(s_{t+1|t}, \Sigma_{t+1|t})$$

其中：

$$\begin{cases} s_{t+1|t} = A \cdot s_{t|t} \\ \Sigma_{t+1|t} = A \cdot \Sigma_{t|t} \cdot A^T + \Sigma_s \end{cases}$$

**更新步骤：**给定  $s_{t+1|t}$  和  $\Sigma_{t+1|t}$ ，使得

$$s_{t+1}|y_1, \dots, y_t \sim \mathcal{N}(s_{t+1|t}, \Sigma_{t+1|t})$$

可以证明

$$s_{t+1}|y_1, \dots, y_{t+1} \sim \mathcal{N}(s_{t+1|t+1}, \Sigma_{t+1|t+1})$$

其中

$$\begin{cases} s_{t+1|t+1} = s_{t+1|t} + K_t(y_{t+1} - Cs_{t+1|t}) \\ \Sigma_{t+1|t+1} = \Sigma_{t+1|t} - K_t \cdot C \cdot \Sigma_{t+1|t} \end{cases}$$

且

$$K_t := \Sigma_{t+1|t} C^T (C \Sigma_{t+1|t} C^T + \Sigma_y)^{-1}$$

矩阵  $K_t$  称为卡尔曼增益 (Kalman gain)。

现在，如果仔细观察这些公式，会发现不需要时间步  $t$  之前的观测值！更新步骤仅依赖于先前的分布。综合来看，该算法首先运行一个前向过程来计算  $K_t$ 、 $\Sigma_{t|t}$  和  $s_{t|t}$  (在文献中有时被称为  $\hat{s}$ )。然后，它运行一个后向过程 (LQR 更新) 来计算量  $\Psi_t$ 、 $\Phi_t$  和  $L_t$ 。最后，通过  $a_t^* = L_t s_{t|t}$  得出最优策略。

# 第 17 章 策略梯度 (REINFORCE)

本章将介绍一种名为 REINFORCE 的无模型 (model-free) 算法, 它不需要价值函数和  $Q$  函数的概念。将 REINFORCE 用于有限时间范围的情况会更方便一点, 因此假设: 使用  $\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T)$  表示一条轨迹, 其中  $T < \infty$  是轨迹的长度。此外, REINFORCE 仅适用于学习随机策略 (randomized policy)。使用  $\pi_\theta(a|s)$  表示策略  $\pi_\theta$  在状态  $s$  下输出动作  $a$  的概率。其他符号将与之前保持一致。

REINFORCE 的优势在于, 只需假设可以从转移概率  $\{P_{sa}\}$  中采样, 并且可以评估状态  $s$  和动作  $a$  下的奖励函数  $R(s, a)$ , 而无需知道转移概率或奖励函数的解析形式。也不需要显式地学习转移概率或奖励函数。<sup>1</sup>

令  $s_0$  从某个分布  $\mu$  中采样。考虑优化策略  $\pi_\theta$  关于参数  $\theta$  的预期总回报, 定义如下:

$$\eta(\theta) \triangleq \mathbb{E} \left[ \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) \right] \quad (17.1)$$

其中有  $s_t \sim P_{s_{t-1} a_{t-1}}$  且  $a_t \sim \pi_\theta(\cdot | s_t)$ 。另请注意, 如果忽略有限和无限时间范围之间的差异, 则  $\eta(\theta) = \mathbb{E}_{s_0 \sim P}[V^{\pi_\theta}(s_0)]$ 。

目标是使用梯度上升来最大化  $\eta(\theta)$ 。这里面临的主要挑战是在不知道奖励函数和转移概率形式的情况下计算 (或估计)  $\eta(\theta)$  的梯度。

令  $P_\theta(\tau)$  表示由策略  $\pi_\theta$  生成的  $\tau$  的分布, 且  $f(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$ 。可以将  $\eta(\theta)$  重写为

$$\eta(\theta) = \mathbb{E}_{\tau \sim P_\theta}[f(\tau)] \quad (17.2)$$

在变分自编码器设置中也面临类似情况, 其中需要对期望下标中的变量求梯度 ( $P_\theta$  分布依赖于  $\theta$ )。在 VAE 中使用了重参数化技巧来解决这个问题。然而, 该技巧不适用于此处, 因为不知道如何计算函数  $f$  的梯度。(计算梯度时, 只能通过观测到的奖励的加权和来评估函数  $f$ , 但不知道奖励函数本身。)

---

<sup>1</sup>本章使用奖励同时依赖于状态和动作的通用设置。

REINFORCE 算法使用另一种方法来估计  $\eta(\theta)$  的梯度。从以下推导开始：

$$\begin{aligned}
 \nabla_{\theta} \mathbb{E}_{\tau \sim P_{\theta}}[f(\tau)] &= \nabla_{\theta} \int P_{\theta}(\tau) f(\tau) d\tau \\
 &= \int \nabla_{\theta}(P_{\theta}(\tau) f(\tau)) d\tau && (\text{交换积分与梯度}) \\
 &= \int (\nabla_{\theta} P_{\theta}(\tau)) f(\tau) d\tau && (\text{因为 } f \text{ 不依赖于 } \theta) \\
 &= \int P_{\theta}(\tau) (\nabla_{\theta} \log P_{\theta}(\tau)) f(\tau) d\tau \\
 &&& (\text{因为 } \nabla_{\theta} \log P_{\theta}(\tau) = \frac{\nabla_{\theta} P_{\theta}(\tau)}{P_{\theta}(\tau)}) \\
 &= \mathbb{E}_{\tau \sim P_{\theta}}[(\nabla_{\theta} \log P_{\theta}(\tau)) f(\tau)] && (17.3)
 \end{aligned}$$

现在，有了  $\nabla_{\theta} \mathbb{E}_{\tau \sim P_{\theta}}[f(\tau)]$  的基于样本的估计。令  $\tau^{(1)}, \dots, \tau^{(n)}$  为  $P_{\theta}$  的  $n$  个经验样本（通过运行  $n$  次策略  $\pi_{\theta}$  获得，每次运行  $T$  步）。可以通过以下方式估计  $\eta(\theta)$  的梯度：

$$\nabla_{\theta} \mathbb{E}_{\tau \sim P_{\theta}}[f(\tau)] = \mathbb{E}_{\tau \sim P_{\theta}}[(\nabla_{\theta} \log P_{\theta}(\tau)) f(\tau)] \quad (17.4)$$

$$\approx \frac{1}{n} \sum_{i=1}^n (\nabla_{\theta} \log P_{\theta}(\tau^{(i)})) f(\tau^{(i)}) \quad (17.5)$$

下一个问题是如何计算  $\log P_{\theta}(\tau)$ 。下面推导  $\log P_{\theta}(\tau)$  的解析公式并计算其关于  $\theta$  的梯度（使用自动微分）。根据  $\tau$  的定义，有

$$P_{\theta}(\tau) = \mu(s_0) \pi_{\theta}(a_0|s_0) P_{s_0 a_0}(s_1) \pi_{\theta}(a_1|s_1) P_{s_1 a_1}(s_2) \cdots P_{s_{T-1} a_{T-1}}(s_T) \quad (17.6)$$

这里  $\mu$  用于表示  $s_0$  的分布密度。因此有

$$\begin{aligned}
 \log P_{\theta}(\tau) &= \log \mu(s_0) + \log \pi_{\theta}(a_0|s_0) + \log P_{s_0 a_0}(s_1) + \log \pi_{\theta}(a_1|s_1) \\
 &\quad + \log P_{s_1 a_1}(s_2) + \cdots + \log P_{s_{T-1} a_{T-1}}(s_T) && (17.7)
 \end{aligned}$$

对  $\theta$  求梯度，得到

$$\nabla_{\theta} \log P_{\theta}(\tau) = \nabla_{\theta} \log \pi_{\theta}(a_0|s_0) + \nabla_{\theta} \log \pi_{\theta}(a_1|s_1) + \cdots + \nabla_{\theta} \log \pi_{\theta}(a_{T-1}|s_{T-1})$$

请注意，许多项消失了，因为它们不依赖于  $\theta$ ，因此梯度为零。（这在某种程度上很重要——我们不知道如何评估像  $\log P_{s_0 a_0}(s_1)$  这样的项，因为无法得知转移概率，但幸运的是这些项的梯度为零！）

将上述方程代入方程(17.4), 得出

$$\begin{aligned}\nabla_{\theta} \eta(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim P_{\theta}}[f(\tau)] = \mathbb{E}_{\tau \sim P_{\theta}} \left[ \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \cdot f(\tau) \right] \\ &= \mathbb{E}_{\tau \sim P_{\theta}} \left[ \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \cdot \left( \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t) \right) \right]\end{aligned}\quad (17.8)$$

通过经验样本轨迹可以估计上述方程的右侧, 并且估计是无偏的。原始 REINFORCE 算法使用估计的梯度通过梯度上升迭代更新参数。

**策略梯度 (policy gradient)** 即 (17.8) 的解释。直观上,  $\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$  这一项是  $\theta$  的变化方向, 这促使轨迹  $\tau$  发生概率变大 (或增加选择动作  $a_0, \dots, a_{t-1}$  的概率), 而  $f(\tau)$  是该轨迹的总回报。因此, 梯度上升直观上会提高所有轨迹的似然, 但对每个  $\tau$  (或对每组动作  $a_0, a_1, \dots, a_{t-1}$ ) 有不同的强调或权重。如果  $\tau$  的回报非常高 (即  $f(\tau)$  很大), 则朝着能够增加轨迹  $\tau$  的概率 (或者增加选择  $a_0, \dots, a_{t-1}$  的概率) 的方向努力增加, 如果  $\tau$  的回报较低, 则以较小的权重进行调整。

从公式 (17.3) 可以得出一个有趣的结论, 即

$$\mathbb{E}_{\tau \sim P_{\theta}} \left[ \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right] = 0 \quad (17.9)$$

为了证明这一点, 令  $f(\tau) = 1$  (即, 奖励始终是一个常数), 那么 (17.8) 的左侧为零, 因为回报始终是固定的常数  $\sum_{t=0}^{T-1} \gamma^t$ 。因此 (17.8) 的右侧也为零, 所以 (17.9) 成立。

实际上, 可以验证对于任何固定的  $t$  和  $s_t$ ,  $\mathbb{E}_{a_t \sim \pi_{\theta}(\cdot | s_t)} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) = 0$ 。<sup>2</sup> 这产生了两个推论。首先, 可以简化公式(17.8) 为

$$\begin{aligned}\nabla_{\theta} \eta(\theta) &= \mathbb{E}_{\tau \sim P_{\theta}} \left[ \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \cdot \left( \sum_{j=0}^{T-1} \gamma^j R(s_j, a_j) \right) \right] \\ &= \mathbb{E}_{\tau \sim P_{\theta}} \left[ \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \cdot \left( \sum_{j \geq t}^{T-1} \gamma^j R(s_j, a_j) \right) \right]\end{aligned}\quad (17.10)$$

---

<sup>2</sup>因为  $\mathbb{E}_{x \sim p_{\theta}} [\nabla_{\theta} \log p_{\theta}(x)] = 0$  通常是成立的。

其中第二个等号来源于

$$\begin{aligned}
 & \mathbb{E}_{\tau \sim P_\theta} \left[ \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot \left( \sum_{0 \leq j < t} \gamma^j R(s_j, a_j) \right) \right] \\
 &= \mathbb{E} \left[ \mathbb{E} [\nabla_\theta \log \pi_\theta(a_t | s_t) | s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t] \cdot \left( \sum_{0 \leq j < t} \gamma^j R(s_j, a_j) \right) \right] \\
 &= 0 \quad (\text{因为 } \mathbb{E}[\nabla_\theta \log \pi_\theta(a_t | s_t) | s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t] = 0)
 \end{aligned}$$

注意，这里使用了全期望定律。上面第二行的外部期望是对  $s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t$  的随机性求期望，而内部期望是对  $a_t$  的随机性求期望（以  $s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t$  为条件）。可以看出，这使得估计稍有简化。

$\mathbb{E}_{a_t \sim \pi_\theta(\cdot | s_t)} \nabla_\theta \log \pi_\theta(a_t | s_t) = 0$  的第二个推论是：对于任何仅依赖于  $s_t$  的值  $B(s_t)$ ，有

$$\begin{aligned}
 & \mathbb{E}_{\tau \sim P_\theta} [\nabla_\theta \log \pi_\theta(a_t | s_t) \cdot B(s_t)] \\
 &= \mathbb{E} [\mathbb{E} [\nabla_\theta \log \pi_\theta(a_t | s_t) | s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t] B(s_t)] \\
 &= 0 \quad (\text{因为 } \mathbb{E}[\nabla_\theta \log \pi_\theta(a_t | s_t) | s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t] = 0)
 \end{aligned}$$

这里再次使用了全期望定律。上面第二行的外部期望是对  $s_0, a_0, \dots, a_{t-1}, s_t$  的随机性求期望，而内部期望是对  $a_t$  的随机性求期望（以  $s_0, a_0, \dots, a_{t-1}, s_t$  为条件）。根据方程 (17.10) 和上述方程，有

$$\begin{aligned}
 \nabla_\theta \eta(\theta) &= \sum_{t=0}^{T-1} \mathbb{E}_{\tau \sim P_\theta} \left[ \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot \left( \sum_{j \geq t}^{T-1} \gamma^j R(s_j, a_j) - \gamma^t B(s_t) \right) \right] \\
 &= \sum_{t=0}^{T-1} \mathbb{E}_{\tau \sim P_\theta} \left[ \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot \gamma^t \left( \sum_{j \geq t}^{T-1} \gamma^{j-t} R(s_j, a_j) - B(s_t) \right) \right] \quad (17.11)
 \end{aligned}$$

因此，通过选择不同的  $B(\cdot)$ ，将得到一个不同的  $\nabla_\theta \eta(\theta)$  估计。引入一个合适的  $B(\cdot)$ ——这通常被称为基线 (baseline)——的好处是它有助于减少估计的方差。<sup>3</sup> 结果表明，一个接近最优的估计是预期的未来回报  $\mathbb{E} \left[ \sum_{j \geq t}^{T-1} \gamma^{j-t} R(s_j, a_j) | s_t \right]$ ，这与价值函数  $V^{\pi_\theta}(s_t)$  几乎相同（忽略有限和无限时间范围之间的差异）。这里

---

<sup>3</sup> 作为一个启发式但具有说明性的例子，假设对于一个固定的  $t$ ，未来奖励  $\sum_{j \geq t}^{T-1} \gamma^{j-t} R(s_j, a_j)$  以相等的概率随机取两个值  $1000 + 1$  和  $1000 - 2$ ，并且  $\nabla_\theta \log \pi_\theta(a_t | s_t)$  的对应值是向量  $z$  和  $-z$ 。（注意，由于  $\mathbb{E}[\nabla_\theta \log \pi_\theta(a_t | s_t)] = 0$ ，如果  $\nabla_\theta \log \pi_\theta(a_t | s_t)$  只能均匀地取两个值，那么这两个值必须是方向相反的两个向量。）在这种情况下，不减去基线时，估计可以取两个值  $(1000 + 1)z$  和  $-(1000 - 2)z$ ；而减去基线  $1000$  后，估计的两个值是  $z$  和  $2z$ 。后者的估计比原始估计具有更低的方差。

可以粗略地估计价值函数  $V^{\pi_\theta}(\cdot)$ ，因为其精确值不影响估计的均值，而只影响方差。将上面总结为算法 7 中所述的带基线的策略梯度算法。<sup>4</sup>

---

**Algorithm 7:** 带基线的策略梯度

---

- 1 **for**  $i = 1, \dots$  **do**
- 2     通过执行当前策略，收集一组轨迹。简记  $\sum_{j \geq t}^{T-1} \gamma^{j-t} R(s_j, a_j)$  为  $R_{\geq t}$ 。
- 3     通过最小化下式，找到一个函数  $B$  以拟合基线

$$\sum_{\tau} \sum_t (R_{\geq t} - B(s_t))^2 \quad (17.12)$$

- 4     根据梯度估计来更新策略参数  $\theta$ :

$$\sum_{\tau} \sum_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \cdot (R_{\geq t} - B(s_t)) \quad (17.13)$$


---

---

<sup>4</sup>需要注意的是，算法中梯度的估计与方程 (17.11) 并不完全匹配。如果在方程 (17.13) 的求和项中乘以  $\gamma^t$ ，那么它们将完全匹配。经验上，移除这些折扣因子效果很好，因为可以提供更大的更新。

# Bibliography

- Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.
- Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2(4):1167–1180, 2020.
- David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.
- Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607, PMLR, 2020.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter*

- of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.
- Jeff Z HaoChen, Colin Wei, Jason D Lee, and Tengyu Ma. Shape Matters: Understanding the Implicit Bias of the Noise Covariance. *arXiv preprint arXiv:2006.08680*, 2020.
- Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *Annals of statistics*, 50(2):949, 2022.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. An introduction to statistical learning, volume 112. Springer, 2021.
- Diederik P Kingma, and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Diederik P Kingma, and Max Welling. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees. In *International Conference on Learning Representations*, 2018.
- Song Mei, and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and the double descent curve. *Communications on Pure and Applied Mathematics*, 75(4):667–766, 2022.
- Preetum Nakkiran. More data can hurt for linear regression: Sample-wise double descent. 2019.
- Preetum Nakkiran, Prayaag Venkat, Sham Kakade, and Tengyu Ma. Optimal regularization can mitigate double descent. 2020.
- Manfred Opper. Statistical mechanics of learning: Generalization, pages 922–925, 1995.
- Manfred Opper. Learning to generalize. *Frontiers of Life*, 3(part 2):763–775, 2001.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. *arXiv preprint arXiv:2002.09277*, 2020.