2.6 特殊类型的矩阵和向量

有些特殊类型的矩阵和向量是特别有用的。

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

不是所有的对角矩阵都是方阵。长方形的矩阵也有可能是对角矩阵。非方阵的对角矩阵没有逆矩阵，但我们仍然可以高效地计算它们的乘法。对于一个长方形对角矩阵 $\boldsymbol{D}$ 而言，乘法 $\boldsymbol{D}\boldsymbol{x}$ 会涉及到 $\boldsymbol{x}$ 中每个元素的缩放，如果 $\boldsymbol{D}$ 是瘦长型矩阵，那么在缩放后的末尾添加一些零；如果 $\boldsymbol{D}$ 是胖宽型矩阵，那么在缩放后去掉最后一些元素。

对称（symmetric）矩阵是转置和自己相等的矩阵：

$$\boldsymbol{A} = \boldsymbol{A}^\top. \tag{2.35}$$

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

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

$$\|\boldsymbol{x}\|_2 = 1. \tag{2.36}$$

如果 $\boldsymbol{x}^\top \boldsymbol{y} = 0$，那么向量 $\boldsymbol{x}$ 和向量 $\boldsymbol{y}$ 互相 **正交**（orthogonal）。如果两个向量都有非零范数，那么这两个向量之间的夹角是 90 度。在 $\mathbb{R}^n$ 中，至多有 n 个范数非零向量互相正交。如果这些向量不仅互相正交，并且范数都为 1，那么我们称它们是 **标准正交**（orthonormal）。

上的下界 (当对数底数不是 2 时，单位将有所不同)。那些接近确定性的分布 (输出几乎可以确定) 具有较低的熵；那些接近均匀分布的概率分布具有较高的熵。图 3.5 给出了一个说明。当 x 是连续的，香农熵被称为 **微分熵**（differential entropy）。

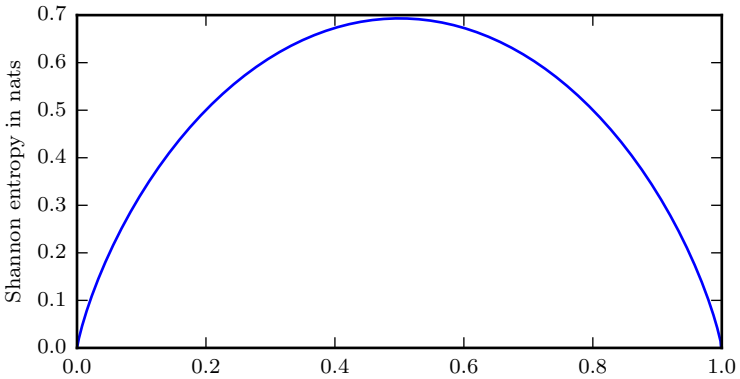

图 3.5: 二值随机变量的香农熵。该图说明了更接近确定性的分布是如何具有较低的香农熵，而更接近均匀分布的分布是如何具有较高的香农熵。水平轴是 p，表示二值随机变量等于 1 的概率。熵由 $(p-1)\log(1-p) - p\log p$ 给出。当 p 接近 0 时，分布几乎是确定的，因为随机变量几乎总是 0。当 p 接近 1 时，分布也几乎是确定的，因为随机变量几乎总是 1。当 $p = 0.5$ 时，熵是最大的，因为分布在两个结果（0 和 1）上是均匀的。

如果我们对于同一个随机变量 x 有两个单独的概率分布 $P(\mathrm{x})$ 和 $Q(\mathrm{x})$，我们可以使用 **KL 散度**（Kullback-Leibler (KL) divergence）来衡量这两个分布的差异：

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

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

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

$D_{\mathrm{KL}}(Q||P)$ 影响很大。更多细节可以看图 3.6 。

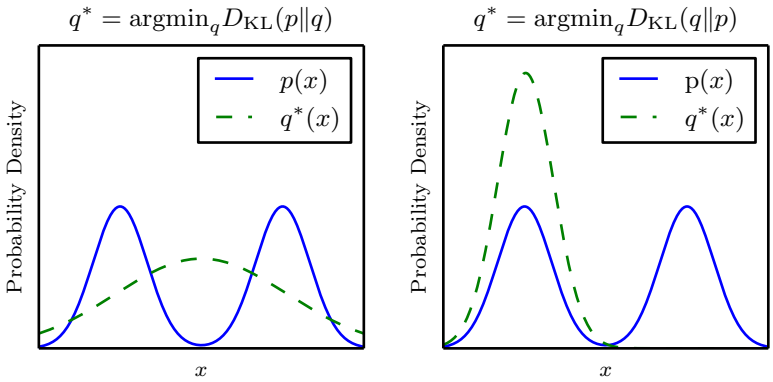

图 3.6: KL 散度是不对称的。假设我们有一个分布 $p(x)$，并且希望用另一个分布 $q(x)$ 来近似它。我们可以选择最小化 $D_{\mathrm{KL}}(p||q)$ 或最小化 $D_{\mathrm{KL}}(q||p)$。为了说明每种选择的效果，我们令 p 是两个高斯分布的混合，令 q 为单个高斯分布。选择使用 KL 散度的哪个方向是取决于问题的。一些应用需要这个近似分布 q 在真实分布 p 放置高概率的所有地方都放置高概率，而其他应用需要这个近似分布 q 在真实分布 p 放置低概率的所有地方都很少放置高概率。KL 散度方向的选择反映了对于每种应用，优先考虑哪一种选择。*(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

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

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

针对 Q 最小化交叉熵等价于最小化 KL 散度，因为 Q 并不参与被省略的那一项。

当我们计算这些量时，经常会遇到 $0 \log 0$ 这个表达式。按照惯例，在信息论中，我们将这个表达式处理为 $\lim_{x\to 0} x \log x = 0$。

3.14 结构化概率模型

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

我们可以把概率分布分解成许多因子的乘积形式，而不是使用单一的函数来表示概率分布。例如，假设我们有三个随机变量 a, b 和 c，并且 a 影响 b 的取值，b 影响 c 的取值，但是 a 和 c 在给定 b 时是条件独立的。我们可以把全部三个变量的概率分布重新表示为两个变量的概率分布的连乘形式：

$$p(\mathrm{a}, \mathrm{b}, \mathrm{c}) = p(\mathrm{a})p(\mathrm{b} \mid \mathrm{a})p(\mathrm{c} \mid \mathrm{b}). \tag{3.52}$$

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

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

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

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

$$p(\mathbf{x}) = \prod_i p(\mathrm{x}_i \mid Pa_{\mathcal{G}}(\mathrm{x}_i)). \tag{3.53}$$

图 3.7 给出了一个有向图的例子以及它表示的概率分布的分解。

无向 (undirected) 模型使用带有无向边的图，它们将分解表示成一组函数；不像有向模型那样，这些函数通常不是任何类型的概率分布。$\mathcal{G}$ 中任何满足两两之间有边连接的顶点的集合被称为团。无向模型中的每个团 $\mathcal{C}^{(i)}$ 都伴随着一个因子 $\phi^{(i)}(\mathcal{C}^{(i)})$。这些因子仅仅是函数，并不是概率分布。每个因子的输出都必须是非负

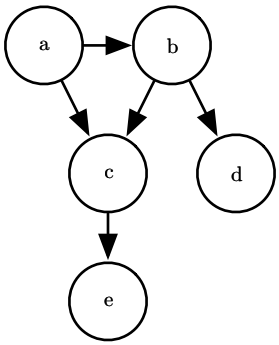

图 3.7: 关于随机变量 a, b, c, d 和 e 的有向图模型。这幅图对应的概率分布可以分解为

$$p(\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}) = p(\mathrm{a})p(\mathrm{b} \mid \mathrm{a})p(\mathrm{c} \mid \mathrm{a}, \mathrm{b})p(\mathrm{d} \mid \mathrm{b})p(\mathrm{e} \mid \mathrm{c}). \tag{3.54}$$

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

的，但是并没有像概率分布中那样要求因子的和或者积分为 1。

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

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

图 3.8 给出了一个无向图的例子以及它表示的概率分布的分解。

请记住，这些图模型表示的分解仅仅是描述概率分布的一种语言。它们不是互相排斥的概率分布族。有向或者无向不是概率分布的特性；它是概率分布的一种特殊 **描述**（description）所具有的特性，而任何概率分布都可以用这两种方式进行描述。

在本书第一部分和第二部分中，我们仅仅将结构化概率模型视作一门语言，来描述不同的机器学习算法选择表示的直接的概率关系。在讨论研究课题之前，读者不需要更深入地理解结构化概率模型。在第三部分的研究课题中，我们将更为详尽地探讨结构化概率模型。

本章复习了概率论中与深度学习最为相关的一些基本概念。我们还剩下一些基

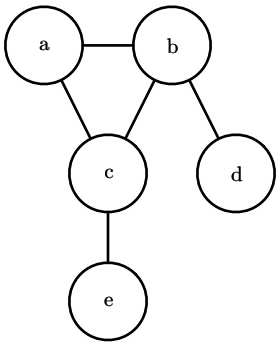

图 3.8: 关于随机变量 $\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}$ 和 e 的无向图模型。这幅图对应的概率分布可以分解为

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

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

本的数学工具需要讨论：数值方法。

第四章　数值计算

机器学习算法通常需要大量的数值计算。这通常是指通过迭代过程更新解的估计值来解决数学问题的算法，而不是通过解析过程推导出公式来提供正确解的方法。常见的操作包括优化（找到最小化或最大化函数值的参数）和线性方程组的求解。对数字计算机来说实数无法在有限内存下精确表示，因此仅仅是计算涉及实数的函数也是困难的。

4.1　上溢和下溢

连续数学在数字计算机上的根本困难是，我们需要通过有限数量的位模式来表示无限多的实数。这意味着我们在计算机中表示实数时，几乎总会引入一些近似误差。在许多情况下，这仅仅是舍入误差。舍入误差会导致一些问题，特别是当许多操作复合时，即使是理论上可行的算法，如果在设计时没有考虑最小化舍入误差的累积，在实践时也可能会导致算法失效。

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

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

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

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

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

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

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

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

4.2 病态条件

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

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

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

这是最大和最小特征值的模之比[1]。当该数很大时，矩阵求逆对输入的误差特别敏感。

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

4.3 基于梯度的优化方法

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

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

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

我们假设读者已经熟悉微积分，这里简要回顾微积分概念如何与优化联系。

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

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

当 $f'(x) = 0$，导数无法提供往哪个方向移动的信息。$f'(x) = 0$ 的点称为 **临界点**（critical point）或 **驻点**（stationary point）。一个 **局部极小点**（local minimum）意味着这个点的 $f(x)$ 小于所有邻近点，因此不可能通过移动无穷小的步长来减小 $f(x)$。一个 **局部极大点**（local maximum）意味着这个点的 $f(x)$ 大于所有邻近点，因此不可能通过移动无穷小的步长来增大 $f(x)$。有些临界点既不是最小点也不是最大

[1] 译者注：与通常的条件数定义有所不同。