第二章　数学基础

在介绍神经网络模型之前，我们先介绍一些数学基础知识，包括线性代数、概率论和优化中的基本概念。

2.1　线性代数

2.1.1　向量

在线性代数中，**标量**（Scalar）是一个实数，而**向量**（Vector）是指 n 个实数组成的有序数组，称为 n 维向量。如果没有特别说明，一个 n 维向量一般表示列向量，即大小为 $n \times 1$ 的矩阵。

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \tag{2.1}$$

其中，a_i 称为向量 $\mathbf{a}$ 的第 i 个分量，或第 i 维。

为简化书写、方便排版起见，有时会以加上**转置**符号 T 的行向量（大小为 $1 \times n$ 的矩阵）表示列向量。

$$\mathbf{a} = [a_1, a_2, \cdots, a_n]^\top \tag{2.2}$$

向量符号一般用黑体小写字母 $\mathbf{a}, \mathbf{b}, \mathbf{c}$，或小写希腊字母 α, β, γ 等来表示。

向量的模

向量 $\mathbf{a}$ 的模 $\|\mathbf{a}\|$ 为

$$\|\mathbf{a}\| = \sqrt{\sum_{i=1}^{n} a_i^2}. \tag{2.3}$$

向量的范数

在线性代数中，**范数**（norm）是一个表示“长度”概念的函数，为向量空间内的所有向量赋予非零的正长度或大小。对于一个 n 维的向量 $\mathbf{x}$，其常见的范数有：

L_1 范数：

$$|\mathbf{x}|_1 = \sum_{i=1}^{n} |x_i|. \tag{2.4}$$

L_2 范数：

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}. \tag{2.5}$$

常见的向量

全 1 向量指所有值为 1 的向量。用 $\underline{1}_n$ 表示，n 表示向量的维数。$\underline{1}_K = [1, \cdots, 1]^{\top}_{K\times 1}$ 是 K 维的全 1 向量。

one-hot 向量表示一个 n 维向量，其中只有一维为 1，其余元素都为 0。在数字电路中，one-hot 是一种状态编码，指对任意给定的状态，状态寄存器中只有 1 位为 1，其余位都为 0。

2.1.2 矩阵

一个大小为 $m \times n$ 的**矩阵**（Matrix）是一个由 m 行 n 列元素排列成的矩形阵列。矩阵里的元素可以是数字、符号或数学式。这里，矩阵我们一般默认指数字矩阵。

一个矩阵 A 从左上角数起的第 i 行第 j 列上的元素称为第 i, j 项，通常记为 $A_{i,j}$ 或 A_{ij}。

一个n维向量可以看作是$n \times 1$的矩阵。

矩阵的基本运算

如果A和B都为$m \times n$的矩阵，则A和B的加减也是$m \times n$的矩阵，其每个元素是A和B相应元素相加或相减。

$$(A + B)_{ij} = A_{ij} + B_{ij}, \tag{2.6}$$

$$(A - B)_{ij} = A_{ij} - B_{ij}. \tag{2.7}$$

A和B的点乘$A \odot B \in \mathbb{R}^{m \times n}$为$A$和$B$相应元素相乘：

$$(A \odot B)_{ij} = A_{ij} B_{ij}. \tag{2.8}$$

一个标量c与矩阵A乘积为A的每个元素是A的相应元素与c的乘积

$$(cA)_{ij} = cA_{ij}. \tag{2.9}$$

两个矩阵的乘积仅当第一个矩阵A的列数和另一个矩阵B的行数相等时才能定义。如A是$m \times p$矩阵和B是$p \times n$矩阵，则乘积AB是一个$m \times n$的矩阵

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj} \tag{2.10}$$

矩阵的乘法满足**结合律**和**分配律**：

- 结合律：$(AB)C = A(BC)$,
- 分配律：$(A + B)C = AC + BC$，$C(A + B) = CA + CB$.

$m \times n$矩阵A的**转置**（Transposition）是一个$n \times m$的矩阵，记为$A^\top$，$A^\top$第i行第j列的元素是原矩阵A第j行第i列的元素，

$$(A^\top)_{ij} = A_{ji}. \tag{2.11}$$

矩阵的**向量化**是将矩阵表示为一个列向量。这里，**vec**是向量化算子。设$A = [a_{ij}]_{m \times n}$，则

$$\mathbf{vec}(A) = [a_{11}, a_{21}, \cdots, a_{m1}, a_{12}, a_{22}, \cdots, a_{m2}, \cdots, a_{1n}, a_{2n}, \cdots, a_{mn}]^\top.$$

常见的矩阵

对称矩阵指其转置等于自己的矩阵，即满足 $A = A^\top$。

对角矩阵（Diagonal Matrix）是一个主对角线之外的元素皆为0的矩阵。对角线上的元素可以为0或其他值。一个 $n \times n$ 的对角矩阵矩阵 A 满足：

$$A_{ij} = 0 \text{ if } i \neq j \qquad \forall i, j \in \{1, \cdots, n\} \tag{2.12}$$

对角矩阵 A 也可以记为 $\mathbf{diag}(\mathbf{a})$，$\mathbf{a}$ 为一个 n 维向量，并满足

$$A_{ii} = a_i. \tag{2.13}$$

$n \times n$ 的对角矩阵矩阵 $A = \mathbf{diag}(\mathbf{a})$ 和 n 维向量 $\mathbf{b}$ 的乘积为一个 n 维向量

$$A\mathbf{b} = \mathbf{diag}(\mathbf{a})\mathbf{b} = \mathbf{a} \odot \mathbf{b}, \tag{2.14}$$

其中 $\odot$ 表示点乘，即 $(\mathbf{a} \odot \mathbf{b})_i = a_i b_i$。

单位矩阵是一种特殊的的对角矩阵，其主对角线元素为1，其余元素为0。n 阶单位矩阵 I_n，是一个 $n \times n$ 的方形矩阵。可以记为 $I_n = \mathbf{diag}(1, 1, ..., 1)$。

一个矩阵和单位矩阵的乘积等于其本身。

$$AI = IA = A \tag{2.15}$$

矩阵的范数

矩阵的范数有很多种形式，这里我们定义其p-范数为

$$\|A\|_p = \Big(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^p \Big)^{1/p}. \tag{2.16}$$

2.1.3 向量的导数

导数（Derivative）是微积分学中重要的基础概念。

对于定义域和值域都是实数域的函数 $y = f(x)$，若 $f(x)$ 在点 x_0 的某个邻域 Δx 内，极限

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \tag{2.17}$$

存在，则称函数 $y = f(x)$ 在点 x_0 处可导，并导数为 $f'(x_0)$。

若函数 $f(x)$ 在其定义域包含的某区间内每一个点都可导，那么也可以说函数 $f(x)$ 在这个区间内可导。这样，我们可以定义函数 $f'(x)$ 为函数 $f(x)$ 的**导函数**，通常也称为**导数**。

函数 $f(x)$ 的**导数** $f'(x)$ 也可记作 $\nabla_x f(x)$，$\frac{\partial f(x)}{\partial x}$ 或 $\frac{\partial}{\partial x} f(x)$。

对于一个 p 维向量 $\mathbf{x} \in \mathbb{R}^p$，函数 $y = f(\mathbf{x}) = f(x_1, \cdots, x_p) \in \mathbb{R}$，则 y 关于 $\mathbf{x}$ 的导数为

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^p. \tag{2.18}$$

对于一个 p 维向量 $\mathbf{x} \in \mathbb{R}^p$，函数 $\mathbf{y} = f(\mathbf{x}) = f(x_1, \cdots, x_p) \in \mathbb{R}^q$，则 $\mathbf{y}$ 关于 $\mathbf{x}$ 的导数为

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_q(\mathbf{x})}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_1(\mathbf{x})}{\partial x_p} & \cdots & \frac{\partial f_q(\mathbf{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}. \tag{2.19}$$

导数法则

导数满足如下法则：

加（减）法则 $\mathbf{y} = f(\mathbf{x}), \mathbf{z} = g(\mathbf{x})$ 则

$$\frac{\partial(\mathbf{y} + \mathbf{z})}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \tag{2.20}$$

乘法法则 （1）若 $\mathbf{x} \in \mathbb{R}^p$，$\mathbf{y} = f(\mathbf{x}) \in \mathbb{R}^q$，$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^q$，则

$$\frac{\partial \mathbf{y}^\top \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \mathbf{z} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \mathbf{y} \tag{2.21}$$

（2）若 $\mathbf{x} \in \mathbb{R}^p$，$\mathbf{y} = f(\mathbf{x}) \in \mathbb{R}^s$，$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^t$，$A \in \mathbb{R}^{s \times t}$ 和 $\mathbf{x}$ 无关，则

$$\frac{\partial \mathbf{y}^\top A \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} A \mathbf{z} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} A^\top \mathbf{y} \tag{2.22}$$

（3）若 $\mathbf{x} \in \mathbb{R}^p$，$y = f(\mathbf{x}) \in \mathbb{R}$，$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^p$，则

$$\frac{\partial y \mathbf{z}}{\partial \mathbf{x}} = y \frac{\partial \mathbf{z}}{\partial \mathbf{x}} + \frac{\partial y}{\partial \mathbf{x}} \mathbf{z}^\top \tag{2.23}$$

链式法则 链式法则（chain rule），是求复合函数导数的一个法则，是在微积分中计算导数的一种常用方法。

（1）若 $\mathbf{x} \in \mathbb{R}^p$，$\mathbf{y} = g(\mathbf{x}) \in \mathbb{R}^s$，$\mathbf{z} = f(\mathbf{y}) \in \mathbb{R}^t$，则

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \tag{2.24}$$

（2）若 $X \in \mathbb{R}^{p \times q}$ 为矩阵，$Y = g(X) \in \mathbb{R}^{s \times t}$，$z = f(Y) \in \mathbb{R}$，则

$$\frac{\partial z}{\partial X_{ij}} = \mathbf{tr}\left((\frac{\partial z}{\partial Y})^\top \frac{\partial Y}{\partial X_{ij}} \right) \tag{2.25}$$

（3）若 $X \in \mathbb{R}^{p \times q}$ 为矩阵，$\mathbf{y} = g(X) \in \mathbb{R}^s$，$z = f(\mathbf{y}) \in \mathbb{R}$，则

$$\frac{\partial z}{\partial X_{ij}} = (\frac{\partial z}{\partial \mathbf{y}})^\top \frac{\partial \mathbf{y}}{\partial X_{ij}} \tag{2.26}$$

2.2 数学优化

数学优化（Mathematical Optimization）问题，也叫**最优化问题**，是指在一定约束条件下，求解一个目标函数的最大值（或最小值）问题。

数值优化问题的定义为：给定一个目标函数（也叫代价函数）$f : A \to \mathbb{R}$，寻找一个变量（也叫参数）$\mathbf{x}^* \in A$，使得对于所有 A 中的 $\mathbf{x}$，$f(\mathbf{x}^*) \leq f(\mathbf{x})$（最小化）；或者 $f(\mathbf{x}^*) \geq f(\mathbf{x})$（最大化）。这里，$A$ 为变量 $\mathbf{x}$ 的**约束集**，也叫**可行域**；A 中的变量被称为是**可行解**。

2.2.1 数学优化的类型

离散优化和连续优化

根据输入变量 $\mathbf{x}$ 的值域是否为实数域，数值优化问题可以分为离散优化问题和连续优化问题。

离散优化问题 离散优化（Discrete Optimization）问题是目标函数的输入变量为离散变量，比如为整数或有限集合中的元素。离散优化问题主要有两个分支：

1. **组合优化**（Combinatorial Optimization）：其目标是从一个有限集合中找出使得目标函数最优的元素。在一般的组合优化问题中，集合中的元素之间存在一定的关联，可以表示为图结构。典型的组合优化问题有旅行商问题、最小生成树问题、图着色问题等。很多机器学习问题都是组合优化问题，比如特征选择、聚类问题、超参数优化问题以及结构化学习（Structured Learning）中标签预测问题等。

2. **整数规划**（Integer Programming ）：输入变量$\mathbf{x} \in \mathbb{Z}^d$为整数。一般常见的整数规划问题为**整数线性规划**（Integer Linear Programming，ILP）。整数线性规划的一种最直接的求解方法是：1）去掉输入为整数的限制，得到一个就成为一个一般的线性规划问题，这个线性规划问题为原整数线性规划问题的松弛问题；2）求得相应松弛问题的解；3）把松弛问题的解四舍五入到最接近的整数。但是这种方法得到的解一般都不是最优的，因此原问题的最优解不一定在松弛问题最优解的附近。另外，这种方法得到的解也不一定满足约束条件。

离散优化问题的求解一般都是比较困难，优化算法的复杂度都比较高。

连续优化问题 连续优化（Continuous Optimization）问题是目标函数的输入变量为连续变量$\mathbf{x} \in \mathbb{R}^d$，即目标函数为实函数。

本节后面的内容主要以连续优化为主。

无约束优化和约束优化

在连续优化问题中，根据是否有变量的约束条件，可以将优化问题分为无约束优化问题和约束优化问题。

无约束优化问题（Unconstrained Optimization）的可行域为整个实数域$A = \mathbb{R}^d$，可以写为

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) \tag{2.27}$$

其中，$\mathbf{x} \in \mathbb{R}^d$为输入变量；$f : \mathbb{R}^d \to \mathbb{R}$为目标函数。

最优化问题一般可以表示为求最小值问题。求$f(\mathbf{x})$最大值等价于求$-f(\mathbf{x})$的最小值。

约束优化问题（Constrained Optimization）中变量$\mathbf{x}$需要满足一些等式或

不等式的约束，可写为

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & c_i(\mathbf{x}) = 0, \quad i = 1, \ldots, m \\ & c_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, n \end{aligned} \tag{2.28}$$

其中，$c_i(x)$ 为等式约束函数；$c_j(x)$ 为不等式约束函数。

有约束条件的约束问题常常可以通过拉格朗日乘数转化为非约束问题。

线性优化和非线性优化

如果在公式2.28中，目标函数和所有的约束函数都为线性函数，则该问题为**线性规划问题**（Linear Programming）。相反，如果目标函数或任何一个约束函数为非线性函数，则该问题为**非线性规划问题**（Nonlinear Programming）。

在非线性优化问题中，有一类比较特殊的问题是**凸优化问题**（Convex Programming）。在凸优化问题中，变量 $\mathbf{x}$ 的可行域为凸集，即对于集合中任意两点，它们的连线全部位于在集合内部。目标函数 f 也必须为凸函数，即满足

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}), \forall \alpha \in [0, 1]. \tag{2.29}$$

凸优化问题是一种特殊的约束优化问题，需满足目标函数为凸函数，并且等式约束函数为线性函数，不等式约束函数为凹函数。

2.2.2 优化算法

优化问题一般都是通过迭代的方式来求解：通过猜测一个初始的估计 $\mathbf{x}_0$，然后不断迭代产生新的估计 $\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_t$，希望 $\mathbf{x}_t$ 最终收敛到期望的最优解 $\mathbf{x}^*$。一个好的优化算法应该是在一定的时间或空间复杂度下能够快速准确地找到最优解。同时，好的优化算法受初始猜测点的影响较小，通过迭代能稳定地找到最优解 $\mathbf{x}^*$ 的邻域，然后迅速收敛于 $\mathbf{x}^*$。

优化算法中常用的迭代方法有线性搜索和置信域方法等。线性搜索的策略是寻找方向和步长，具体算法有梯度下降法、牛顿法、共轭梯度法等。

全局最优和局部最优

对于很多非线性优化问题，会存在若干个局部的极小值。**局部最小值**，或**局部最优解** $\mathbf{x}^*$ 定义为：存在一个 $\delta > 0$，对于所有的满足 $\|\mathbf{x} - \mathbf{x}^*\| \leq \delta$ 的 $\mathbf{x}$，公式 $f(\mathbf{x}^*) \leq f(\mathbf{x})$ 成立。也就是说，在 $\mathbf{x}^*$ 的附近区域内，所有的函数值都大于或者等于 $\mathbf{x}^*$。

所对于所有的 $\mathbf{x} \in A$，都有 $f(\mathbf{x}^*) \leq f(\mathbf{x})$ 成立，则 $\mathbf{x}^*$ 为**全局最小值**，或**全局最优解**。

一般的，求局部最优解是容易的，但很难保证其为全局最优解。对于线性规划或凸优化问题，局部最优解就是全局最优解。

要确认一个点 $\mathbf{x}^*$ 是否为局部最优解，通过比较它的邻域内有没有更小的函数值是不现实的。如果函数 $f(\mathbf{x})$ 是二次连续可微的，我们可以通过检查目标函数在点 $\mathbf{x}^*$ 的梯度 $\nabla f(\mathbf{x}^*)$ 和 Hessian 矩阵 $\nabla^2 f(\mathbf{x}^*)$ 来判断。

定理 2.1 – 局部最小值的一阶必要条件： 如果 $\mathbf{x}^*$ 为局部最优解并且函数 f 在 $\mathbf{x}^*$ 的邻域内一阶可微，则在 $\nabla f(\mathbf{x}^*) = 0$。

证明. 如果函数 $f(\mathbf{x})$ 是连续可微的，根据泰勒展开公式（Taylor's Formula），函数 $f(\mathbf{x})$ 的一阶展开可以近似为

$$f(\mathbf{x}^* + \triangle\mathbf{x}) = f(\mathbf{x}^*) + \triangle\mathbf{x}^\top \nabla f(\mathbf{x}^*), \tag{2.30}$$

假设 $\nabla f(\mathbf{x}^*) \neq 0$，则可以找到一个 $\triangle\mathbf{x}$（比如 $\triangle\mathbf{x} = -\alpha\nabla f(\mathbf{x}^*)$，$\alpha$ 为很小的正数），使得

$$f(\mathbf{x}^* + \triangle\mathbf{x}) - f(\mathbf{x}^*) = \triangle\mathbf{x}^\top \nabla f(\mathbf{x}^*) \leq 0. \tag{2.31}$$

这和局部最优的定义矛盾。 □

定理 2.2 – 局部最优解的二阶必要条件： 如果 $\mathbf{x}^*$ 为局部最优解并且函数 f 在 $\mathbf{x}^*$ 的邻域内二阶可微，则在 $\nabla f(\mathbf{x}^*) = 0$，$\nabla^f(\mathbf{x}^*)$ 为半正定矩阵。

证明. 如果函数 $f(\mathbf{x})$ 是二次连续可微的，函数 $f(\mathbf{x})$ 的二阶展开可以近似为

$$f(\mathbf{x}^* + \triangle\mathbf{x}) = f(\mathbf{x}^*) + \triangle\mathbf{x}^\top \nabla f(\mathbf{x}^*) + \frac{1}{2}\triangle\mathbf{x}^\top(\nabla^2 f(\mathbf{x}^*))\triangle\mathbf{x}. \quad (2.32)$$

有一阶必要性定理可知 $\nabla f(\mathbf{x}^*)$ 则

$$f(\mathbf{x}^* + \triangle\mathbf{x}) - f(\mathbf{x}^*) = \frac{1}{2}\triangle\mathbf{x}^\top(\nabla^2 f(\mathbf{x}^*))\triangle\mathbf{x} \geq 0. \quad (2.33)$$

即 $\nabla^2 f(\mathbf{x}^*)$ 为半正定矩阵。 □

梯度下降法

梯度下降法（Gradient Descent Method），也叫最速下降法（Steepest Descend Method），经常用来求解无约束优化的极值问题。

对于函数 $f(\mathbf{x})$，如果 $f(\mathbf{x})$ 在点 $\mathbf{x}_t$ 附近是连续可微的，那么 $f(\mathbf{x})$ 下降最快的方向是 $f(\mathbf{x})$ 在 $\mathbf{x}_t$ 点的梯度方法的反方向。

根据泰勒一阶展开公式，

$$f(\mathbf{x}_{t+1}) = f(\mathbf{x}_t + \triangle\mathbf{x}) \approx f(\mathbf{x}_t) + \triangle\mathbf{x}^\top \nabla f(\mathbf{x}_t). \quad (2.34)$$

要使得 $f(\mathbf{x}_{t+1}) < f(\mathbf{x}_t)$，就得使 $\triangle\mathbf{x}^\top \nabla f(\mathbf{x}_t) < 0$。我们取 $\triangle\mathbf{x} = -\alpha\nabla f(\mathbf{x}_t)$。如果 $\alpha > 0$ 为一个够小数值时，那么 $f(\mathbf{x}_{t+1}) < f(\mathbf{x}_t)$ 成立。

这样，我们可以从一个初始估计 $\mathbf{x}_0$ 出发，通过迭代公式

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t),\ t \geq 0. \quad (2.35)$$

生成序列 $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ 使得

$$f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq f(\mathbf{x}_2) \geq \cdots \quad (2.36)$$

如果顺利的话序列，$(\mathbf{x}_n)$ 收敛到局部最优解 $\mathbf{x}^*$。注意每次迭代步长 α 可以改变，其取值必须合适，如果过大就不会收敛，如果过小则收敛速度太慢。

梯度下降法的过程如图2.1所示。曲线是等高线（水平集），即函数 f 为不同常数的集合构成的曲线。红色的箭头指向该点梯度的反方向（梯度方向与通过该点的等高线垂直）。沿着梯度下降方向，将最终到达函数 f 值的局部最优解。

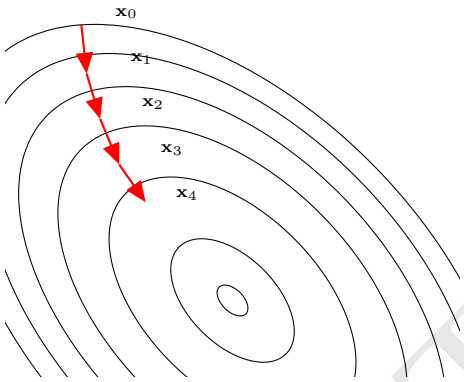


图 2.1: 梯度下降法

梯度下降法为一阶收敛算法，当靠近极小值时梯度变小，收敛速度会变慢，并且可能以"之字形"的方式下降。如果目标函数为二阶连续可微，我们可以采用牛顿法。**牛顿法**为二阶收敛算法，收敛速度更快，但是每次迭代需要计算Hessian矩阵的逆矩阵，复杂较高。

2.3 概率论基础

概率论是研究大量随机现象中数量规律的数学分支。

2.3.1 样本空间

样本空间是一个实验或随机试验所有可能结果的集合，而随机试验中的每个可能结果称为样本点。通常用Ω表示。例如，如果抛掷一枚硬币，那么样本空间就是集合{正面，反面}。如果投掷一个骰子，那么样本空间就是$\{1,2,3,4,5,6\}$。

有些实验有两个或多个可能的样本空间。例如，从52张扑克牌中随机抽出一张，一个可能的样本空间是数字（A到K），另外一个可能的样本空间是花色（黑桃，红桃，梅花，方块）。如果要完整地描述一张牌，就需要同时给出数字和花色，这时的样本空间可以通过构建上述两个样本空间的笛卡儿乘积来得到。

数学小知识 | 笛卡儿乘积

在数学中，两个集合 X 和 Y 的笛卡儿乘积（Cartesian product），又称直积，在集合论中表示为X Œ Y，是所有可能的有序对组成的集合，其中有序对的第一个对象是X的成员，第二个对象是Y的成员。

$$X \times Y = \{\langle x, y\rangle \mid x \in X \wedge y \in Y\}.$$

举个实例，如果集合X是13个元素的点数集合{A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2}，而集合Y是4个元素的花色集合{, , , }，则这两个集合的笛卡儿积是有52个元素的标准扑克牌的集合{(A,), (K,), ..., (2,), (A,), ..., (3,), (2,)}。

2.3.2 事件和概率

随机事件（或简称事件）指的是一个被赋与机率的事物集合，也就是样本空间中的一个子集。**概率**表示对一个随机事件发生的可能性大小，为0到1之间的一个非负实数。

比如，一个0.5的概率表示一个事件有50%的可能性发生。

对于一个机会均等的抛硬币动作来说，其样本空间为“正面”或“反面”。下面各个随机事件的概率分别为：

- {正面}，其概率为0.5；
- {反面}，其概率为0.5；
- { }=∅，不是正面也不是反面，其概率为0；
- {正面，反面}，不是正面就是反面，这是，其概率为1。

2.3.3 随机变量

在随机实验中，很多问题是与数值发生关系，试验的结果可以用一个数 X 来表示，这个数 X 是随着试验结果的不同而变化的，是样本点的一个函数。我们把这种数称为**随机变量**。

例如，随机掷一个骰子，得到的点数就可以看成一个随机变量X，X的取值为$\{1,2,3,4,5,6\}$。

如果随机掷两个骰子，整个事件空间可以由36个元素组成：

$$\Omega = \{(i,j)|i = 1,\ldots,6; j = 1,\ldots,6\} \tag{2.37}$$

这里可以构成多个随机变量，比如随机变量X（获得的两个骰子的点数和）或者随机变量Y （获得的两个骰子的点数差），随机变量X可以有11个整数值，而随机变量Y只有6个。

$$X(i,j) := i+j, \quad x = 2,3,\ldots,12 \tag{2.38}$$

$$Y(i,j) := |\, i-j \,|, \quad y = 0,1,2,3,4,5. \tag{2.39}$$

又比如，在一次扔硬币事件中，如果把获得的背面的次数作为随机变量X，则X可以取两个值，分别是0和1。

离散随机变量

如果**随机变量**X所可能取的值为有限可列举的，假设有n个有限取值$\{x_1,\cdots,x_n\}$。X称为**离散随机变量**。我们一般用大写的字母表示一个随机变量，用小字字母表示该变量的某一个具体的取值。

要了解X的统计规律，就必须知道它取每种可能值x_i的概率，即

$$P(X = x_i) = p(x_i), i = 1\cdots,n \tag{2.40}$$

$p(x_i), i = 1\cdots,n$称为离散型随机变量X的**概率分布**或**分布**，并且满足

$$\sum_{i=1}^{n} P(x_i) = 1 \tag{2.41}$$

$$P(x_i) \geq 0, \quad i = 1\cdots,n \tag{2.42}$$

下面看些常见的离散随机变量以及其概率分布的例子。

[**伯努利分布**] 在一次试验中，事件A出现的概率为p，不出现的概率为$q = 1-p$。若用变量X表示事件A出现的次数，则X的取值为0和1，其相应的分布为：$P(X=0) = q, P(X=1) = p$。这个分布称为伯努利分布（Bernoulli distribution），又名两点分布或者0-1分布。

数学小知识 | 排列组合

排列组合是组合学最基本的概念。所谓**排列**，就是指从给定个数的元素中取出指定个数的元素进行排序。**组合**则是指从给定个数的元素中仅仅取出指定个数的元素，不考虑排序。排列组合的中心问题是研究给定要求的排列和组合可能出现的情况总数。

排列的任务是确定 n 个不同的元素的排序的可能性。n 个不同的元素可以有 $n!$ 种不同的排列方式，即 n 的阶乘。

$$n! = n \times (n-1) \times \cdots \times 3 \times 2 \times 1.$$

如果从 n 个元素中取出 k 个元素，这 k 个元素的排列总数为

$$P_n^k = n \times (n-1) \times \cdots \times (n-k+1) = \frac{n!}{(n-k)!}.$$

从 n 个元素中取出 k 个元素，这 k 个元素可能出现的组合数为

$$C_n^k = \binom{n}{k} = \frac{P_n^k}{k!} = \frac{n!}{k!(n-k)!}$$

区分排列与组合的关键是"有序"与"无序"。

[**二项分布**] 在 n 次伯努利分布中，若以变量 X 表示事件 A 出现的次数，则 X 的取值为 $\{0, \cdots, n\}$，其相应的分布为：

$$P(X=k) = \binom{n}{k} p^k q^{n-k}, \quad k = 1 \cdots, n \tag{2.43}$$

这里，$\binom{n}{k}$ 为二项式系数（这就是二项分布的名称的由来），表示从 n 个元素中取出 k 个元素而不考虑其顺序的**组合**的总数。

连续随机变量

与离散随机变量不同，一些随机变量 X 的取值是不可列举的，由全部实数或者由一部分区间组成，比如

$$X = \{x | a \leq x \leq b\}, \quad -\infty < a < b < \infty$$

则称 X 为**连续随机变量**，连续随机变量的值是不可数及无穷尽的。

对于连续随机变量 X，它取一个具体值 x_i 的概率为0，这个离散随机变量截然不同。因此用列举连续随机变量取某个值的概率来描述这种随机变量不但做不到，也毫无意义。

我们一般用**密度函数** $p(x)$ 来描述连续随机变量 X 的概率分布。$p(x)$ 为可积函数，并满足

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \tag{2.44}$$

$$p(x) \geq 0. \tag{2.45}$$

给定密度函数 $p(x)$，便可以计算出随机变量落入某一个区间的概率，而 $p(x)$ 本身反映了随机变量取落入 x 的非常小的邻近区间中的概率大小。

下面看些常见的连续随机变量以及其概率分布的例子。

[**均匀分布**] 若 a, b 为有限数，有下面密度函数定义的分布称为 $[a, b]$ 上的均匀分布（Uniform Distribution）。

$$p(x) = \begin{cases} \frac{1}{b-a} & , \quad a \leq x \leq b \\ 0 & , \quad x < a或x > b \end{cases} \tag{2.46}$$

[**正态分布**] 正态分布（Normal Distribution），又名高斯分布，是自然界最常见的一种分布，并且具有很多良好的性质，在很多领域都有非常重要的影响力，其概率密度函数为

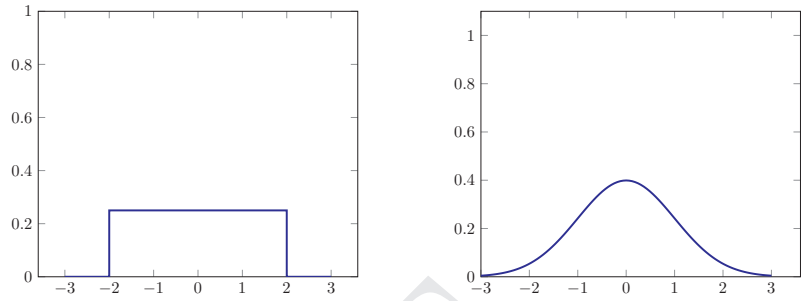
$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2.47}$$

其中，$\sigma > 0$，μ 和 σ 均为常数。若随机变量 X 服从一个位置参数为 μ 和 σ 的概率分布，简记为

$$X \sim N(\mu, \sigma^2). \tag{2.48}$$

当 $\mu = 0$，$\sigma = 1$ 时，称为**标准正态分布**。

图2.2a和2.2b分别显示了均匀分布和正态分布的密度函数。



(a) 均匀分布 (b) 正态分布

图 2.2: 连续随机变量的密度函数

2.3.4 随机向量

随机向量是指一组随机变量构成的向量。如果 $X_1, X_2, \cdots, X_n$ 为 n 个随机变量, 那么称 $(X_1, X_2, \cdots, X_n)$ 为一个 n 维随机向量。一维随机向量称为随机变量。

随机向量也分为离散随机向量和连续随机向量。

一般，离散随机向量的（**联合**）**概率分布**为

$$P(X_1 = x_{i_1}, X_2 = x_{i_2}, \cdots, X_n = x_{i_n}) = p(x_{i_1}, x_{i_2}, \cdots, x_{i_n}), \quad x_{i_1}, x_{i_2}, \cdots, x_{i_n} = 1, 2, \cdots \tag{2.49}$$

和离散随机变量类似，离散随机向量的概率分布满足

$$p(x_{i_1}, x_{i_2}, \cdots, x_{i_n}) \geq 0, \quad x_{i_1}, x_{i_2}, \cdots, x_{i_n} = 1, 2, \cdots \tag{2.50}$$

$$\sum_{i_1} \sum_{i_2} \cdots \sum_{i_n} p(x_{i_1}, x_{i_2}, \cdots, x_{i_n}) = 1. \tag{2.51}$$

对连续型随机向量，其（**联合**）**密度函数**满足

$$p(x_1, \cdots, x_n) \geq 0, \tag{2.52}$$

$$\int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} p(x_1, \cdots, x_n) dx_1 \cdots dx_n = 1. \tag{2.53}$$

[**多元正态分布**] 若 n 维随机向量 $X = [X_1, \ldots, X_N]^T$ 服从 n 元正态分布，其密度函数为

$$p(x_1, \ldots, x_n) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right), \tag{2.54}$$

其中，Σ 为正定对称矩阵；Σ^{-1} 表示 Σ 的逆阵；$|\Sigma|$ 表示 Σ 的行列式；向量 $\mu = [\mu_1, \ldots, \mu_N]^T$ 为列向量。

多项分布 在 n 次伯努利分布中，若以变量 X 表示事件 A 出现的次数，则 X 的取值为 $\{0, \cdots, n\}$，其相应的分布为：

$$P(X = k) = \binom{n}{k} p^k q^{n-k}, \quad k = 1 \cdots, n \tag{2.55}$$

这里，$\binom{n}{k}$ 为二项系数（这就是二项分布的名称的由来），表示从 n 个元素中取出 k 个元素而不考虑其顺序的**组合**的总数。

把二项分布推广到随机向量，就得到了多项分布。假设一个袋子中装了很多球，总共有 k 个不同的颜色。我们从袋子中取出 n 个球。每次取出一个球时，就在袋子中放入一个同样颜色的球。这样保证每次取到同颜色的球的概率是相等的。我们定义一个 k 维随机向量 X，X_i 定义为取出的 n 个球中颜色为 i 的球的数量 $(i = 1, \cdots, k)$。θ_i 定义为每次抽取的球颜色为 i 的概率。X 服从多项分布，其概率分布为：

$$\begin{aligned} p(x_1, \ldots, x_k|\boldsymbol{\theta}) &= P(X_1 = x_1, \cdots, X_k = x_k|\theta_1, \cdots, \theta_k) & (2.56)\\ &= \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}, & (2.57) \end{aligned}$$

其中，$x_1, \cdots, x_k$ 为非负整数，并且满足 $\sum_{i=1}^k x_i = n$。

多项分布的概率分布也可以用 gamma 函数表示：

$$p(x_1, \cdots, x_k|\boldsymbol{\theta}) = \frac{\Gamma(\sum_i x_i + 1)}{\prod_i \Gamma(x_i + 1)} \prod_{i=1}^k \theta_i^{x_i}. \tag{2.58}$$

从这种表示形式和 Dirichlet 分布类似，而 Dirichlet 分布可以作为多项分布的共轭先验。

2.3.5 边际分布

为了方便起见，我们下面以二维随机向量进行讨论，多维时这些结论依然成立。对于二维离散随机向量(X,Y)，假设X取值为$x_1,x_2,\cdots$；Y取值为$y_1,y_2,\cdots$。其联合概率分布满足

$$p(x_i,y_j)\geq 0,\qquad \sum_{i,j}p(x_i,y_j)=1. \tag{2.59}$$

对于联合概率分布$p(x_i,y_j)$，我们可以分别对i和j进行求和。

（1）对于固定的i，

$$\sum_j p(x_i,y_j)=P(X=x_i)=p(x_i). \tag{2.60}$$

（2）对于固定的j，

$$\sum_i p(x_i,y_j)=P(Y=y_j)=p(y_j). \tag{2.61}$$

也就是说，由离散随机向量(X,Y)的联合概率分布，对Y的所有取值进行求和得到X的概率分布；而对X的所有取值进行求和得到Y的概率分布。这里$p(x_i)$和$p(y_j)$就称为$p(x_i,y_j)$的**边际分布**。

对于二维连续随机向量(X,Y)，其边际分布为：

$$p(x)=\int_{-\infty}^{+\infty}p(x,y)dy \tag{2.62}$$

$$p(y)=\int_{-\infty}^{+\infty}p(x,y)dx \tag{2.63}$$

对于二元正态分布，其边际分布仍为正态分布。

2.3.6 条件分布

对于离散随机向量(X,Y)，已知$X=x_i$的条件下，随机变量$Y=y_j$的**条件概率**为：

$$P(Y=y_j|X=x_i)=\frac{P(X=x_i,Y=y_j)}{P(X=x_i)}=\frac{p(x_i,y_j)}{p(x_i)}. \tag{2.64}$$

这个公式定义了随机变量Y关于随机变量X的**条件分布**。

对于二维连续随机向量(X,Y)，已知$X=x$的条件下，随机变量$Y=y$的**条件密度函数**为

$$p(y|x)=\frac{p(x,y)}{p(x)}. \tag{2.65}$$

同理，已知$Y=y$的条件下，随机变量$X=x$的**条件密度函数**为

$$p(x|y)=\frac{p(x,y)}{p(y)}. \tag{2.66}$$

通过公式2.65和2.65，我们可以得到两个条件概率$p(y|x)$和$p(x|y)$之间的关系。

$$p(y|x)=\frac{p(y|x)p(y)}{p(x)}=\frac{p(y|x)p(y)}{\sum_y p(y|x)p(y)} \tag{2.67}$$

这个公式称为**贝叶斯公式**

2.3.7 随机变量的独立性

对于离散随机向量(X,Y)，如果其联合概率$p(x_i,y_j)$满足

$$p(x_i,y_j)=p(x_i)p(y_j),\quad i,j=1,2,\cdots \tag{2.68}$$

则称X和Y是**相互独立**的。

对于二维连续随机向量(X,Y)，如果其联合分布概率密度函数$p(x,y)$满足

$$p(x,y)=p(x)p(y), \tag{2.69}$$

则称X和Y是**相互独立**的。

2.3.8 均值和方差

对于离散变量X，取值为$x_1,x_2,\cdots$，对应的概率为$p(x_1),p(x_2),\cdots$，如果级数

$$\sum_{i=1}^{\infty}x_ip(x_i) \tag{2.70}$$

绝对收敛，则把它称为X的**数学期望**或**均值**，记为$E(X)$。

如果$\sum_{i=1}^{\infty}x_ip(x_i)$发散时，则说$X$的均值不存在。

若 $E((X-E(X))^2)$ 存在，则称它为随机变量 X 的**方差**，记为 $D(X)$，$\sqrt{D(X)}$ 则称为 X 的**根方差**或**标准差**。

2.4 信息论

2.4.1 熵

熵的概念最早起源于物理学，用于度量一个热力学系统的无序程度。在信息论里面，熵是对不确定性的测量。但是在信息世界，熵越高，则能传输越多的信息，熵越低，则意味着传输的信息越少。

香农熵值 （希腊字母 Eta）定义如下，其值域为 x1, ..., xn：

$$\mathrm{H} = \mathrm{E}[\mathrm{I}(X)] = \mathrm{E}[-\ln(\mathrm{P}(X))]$$

。

当取自有限的样本时，熵的公式可以表示为：

$$\mathrm{H}(X) = \sum_i \mathrm{P}(x_i)\,\mathrm{I}(x_i) = -\sum_i \mathrm{P}(x_i)\log_b \mathrm{P}(x_i),$$

在这里 b 是对数所使用的底，通常是 2, 自然常数 e，或是 10。当 b = 2，熵的单位是 bit；当 b = e，熵的单位是 nat；而当 b = 10, 熵的单位是 Hart。pi = 0 时，对于一些 i 值，对应的被加数 0 logb 0 的值将会是 0，这与极限一致。

$$\lim_{p\to 0+} p\log p = 0$$

。

还可以定义事件 X 与 Y 分别取 xi 和 yj 时的条件熵为

$$\mathrm{H}(X|Y) = \sum_{i,j} p(x_i, y_j)\log\frac{p(y_j)}{p(x_i, y_j)} \tag{2.71}$$

$$= -\sum_{i,j} p(x_i, y_j)\log p(x_i|y_j) \tag{2.72}$$

其中 p(xi, yj) 为 X = xi 且 Y = yj 时的概率。这个量应当理解为你知道 Y 的值前提下随机变量 X 的随机性的量。

2.4.2 交叉熵

交叉熵（Cross Entropy）The cross entropy for the distributions p and q over a given set is defined as follows:

$$H(p,q) = \mathrm{E}_p[-\log q] = H(p) + D_{\mathrm{KL}}(p\|q), \tag{2.73}$$

2.4.3 互信息

正式地，两个离散随机变量 X 和 Y 的互信息可以定义为：

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right), \tag{2.74}$$

直观上，互信息度量 X 和 Y 共享的信息：它度量知道这两个变量其中一个，对另一个不确定度减少的程度。例如，如果 X 和 Y 相互独立，则知道 X 不对 Y 提供任何信息，反之亦然，所以它们的互信息为零。在另一个极端，如果 X 是 Y 的一个确定性函数，且 Y 也是 X 的一个确定性函数，那么传递的所有信息被 X 和 Y 共享：知道 X 决定 Y 的值，反之亦然。因此，在此情形互信息与 Y（或 X）单独包含的不确定度相同，称作 Y（或 X）的熵。而且，这个互信息与 X 的熵和 Y 的熵相同。（这种情形的一个非常特殊的情况是当 X 和 Y 为相同随机变量时。）

2.5 常用函数及其导数

这里我们介绍几个本书中常用的函数。

2.5.1 标量函数及其导数

指示函数 指示函数 $I(x=c)$ 为

$$I(x=c) == \begin{cases} 1 & \text{if} \quad x=c, \\ 0 & \text{else} \quad 0. \end{cases} \tag{2.75}$$

指示函数 $I(x=c)$ 除了在 c 外，其导数为 0。

多项式函数 如果 $f(x) = x^r$，其中 r 是非零实数，那么导数

$$\frac{\partial x^r}{\partial x} = rx^{r-1}. \tag{2.76}$$

当 $r = 0$ 时，常函数的导数是 0。

指数函数 底数为 e 的指数函数 $f(x) = \exp(x) = e^x$ 的导数是它本身。

$$\frac{\partial \exp(x)}{\partial x} = \exp(x). \tag{2.77}$$

对数函数 底数为 e 对数函数 $\log(x)$ 的导数则是 x^{-1}。

$$\frac{\partial \log(x)}{\partial x} = \frac{1}{x}. \tag{2.78}$$

2.5.2 向量函数及其导数

$$\frac{\partial \mathbf{x}}{\partial \mathbf{x}} = I, \tag{2.79}$$

$$\frac{\partial A\mathbf{x}}{\partial \mathbf{x}} = A^\top, \tag{2.80}$$

$$\frac{\partial \mathbf{x}^\top A}{\partial \mathbf{x}} = A \tag{2.81}$$

2.5.3 按位计算的向量函数及其导数

假设一个函数 $f(x)$ 的输入是标量 x。对于一组 K 个标量 $x_1, \cdots, x_K$，我们可以通过 $f(x)$ 得到另外一组 K 个标量 $z_1, \cdots, z_K$，

$$z_k = f(x_k), \forall k = 1, \cdots, K \tag{2.82}$$

为了简便起见，我们定义 $\mathbf{x} = [x_1, \cdots, x_K]^\top$，$\mathbf{z} = [z_1, \cdots, z_K]^\top$，

$$\mathbf{z} = f(\mathbf{x}), \tag{2.83}$$

其中，$f(\mathbf{x})$ 是按位运算的，即 $(f(\mathbf{x}))_i = f(x_i)$。

当x为标量时，$f(x)$的导数记为$f'(x)$。当输入为K维向量$\mathbf{x} = [x_1, \cdots, x_K]^\top$时，其导数为一个对角矩阵。

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = [\frac{\partial f(x_j)}{\partial x_i}]_{K\times K} \tag{2.84}$$

$$= \begin{bmatrix} f'(x_1) & 0 & \cdots & 0 \\ 0 & f'(x_2) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & f'(x_K) \end{bmatrix} \tag{2.85}$$

$$= \mathbf{diag}(f'(\mathbf{x})). \tag{2.86}$$

2.5.4 Logistic 函数

Logistic 函数是一种常用的S形函数，是比利时数学家 Pierre François Verhulst 在1844-1845年研究种群数量的增长模型时提出的命名的，最初作为一种生态学模型。

Logistic 函数定义为：

$$\text{logistic}(x) = \frac{L}{1 + \exp(-k(x - x_0))}, \tag{2.87}$$

这里 exp() 函数表示自然对数，x_0是中心点，L是最大值，k是曲线的倾斜度。图2.3给出了几种不同参数的logistic函数曲线。当x趋向于$-\infty$时，$\sigma(x)$接近于0；当x趋向于$+\infty$时，$\sigma(x)$接近于L.

一般使用标准logistic函数（记为$\sigma(x)$），即参数为$(k = 1, x_0 = 0, L = 1)$，

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{2.88}$$

标准logistic函数在机器学习中使用得非常广泛，经常用来将一个实数空间的数映射到$(0, 1)$区间。

标准logistic函数的导数为

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2.89}$$

$$\tag{2.90}$$

当输入为K维向量$\mathbf{x} = [x_1, \cdots, x_K]^\top$时，其导数为

$$\sigma'(\mathbf{x}) = \mathbf{diag}\left(\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))\right). \tag{2.91}$$

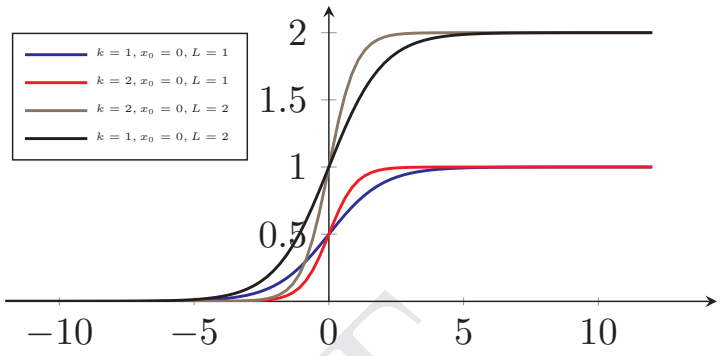


图 2.3: Logistic 函数

2.5.5 softmax 函数

softmax 函数是将多个标量映射为一个概率分布。

对于 K 个标量 $x_1, \cdots, x_K$，softmax 函数定义为

$$z_k = \text{softmax}(x_k) = \frac{\exp(x_k)}{\sum_{i=1}^{K} \exp(x_i)}, \tag{2.92}$$

这样，我们可以将 K 个变量 $x_1, \cdots, x_K$ 转换为一个分布：$z_1, \cdots, z_K$，满足

$$z_k \in [0, 1], \forall k, \qquad \sum_{i=1}^{K} z_k = 1. \tag{2.93}$$

当 softmax 函数的输入为 K 维向量 $\mathbf{x}$ 时，

$$\hat{\mathbf{z}} = \text{softmax}(\mathbf{x}) \tag{2.94}$$

$$= \frac{1}{\sum_{k=1}^{K} \exp(x_k)} \begin{bmatrix} \exp(x_1) \\ \vdots \\ \exp(x_K) \end{bmatrix}$$

$$= \frac{\exp(\mathbf{x})}{\sum_{k=1}^{K} \exp(x_k)}$$

$$= \frac{\exp(\mathbf{x})}{1_K^{\top} \exp(\mathbf{x})}, \tag{2.95}$$

其中，$\underline{1}_K = [1, \cdots, 1]_{K\times 1}$ 是 K 维的全 1 向量。

其导数为

$$\begin{aligned}
\frac{\partial \operatorname{softmax}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial \left(\frac{\exp(\mathbf{x})}{\underline{1}_K^\top \exp(\mathbf{x})}\right)}{\partial \mathbf{x}} \\
&= \frac{1}{\underline{1}_K \exp(\mathbf{x})}\frac{\partial \exp(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \left(\frac{1}{\underline{1}_K^\top \exp(\mathbf{x})}\right)}{\partial \mathbf{x}} (\exp(\mathbf{x}))^\top \\
&= \frac{\mathbf{diag}(\exp(\mathbf{x}))}{\underline{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\underline{1}_K^\top \exp(\mathbf{x}))^2}\right)\frac{\partial \left(\underline{1}_K^\top \exp(\mathbf{x})\right)}{\partial \mathbf{x}} (\exp(\mathbf{x}))^\top \\
&= \frac{\mathbf{diag}(\exp(\mathbf{x}))}{\underline{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\underline{1}_K^\top \exp(\mathbf{x}))^2}\right)\mathbf{diag}(\exp(\mathbf{x}))\,\underline{1}_K (\exp(\mathbf{x}))^\top \\
&= \frac{\mathbf{diag}(\exp(\mathbf{x}))}{\underline{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\underline{1}_K^\top \exp(\mathbf{x}))^2}\right)\exp(\mathbf{x})(\exp(\mathbf{x}))^\top \\
&= \mathbf{diag}\left(\frac{\exp(\mathbf{x})}{\underline{1}_K \exp(\mathbf{x})}\right) - \frac{\exp(\mathbf{x})}{\underline{1}_K^\top \exp(\mathbf{x}))} \cdot \frac{(\exp(\mathbf{x}))^\top}{\underline{1}_K^\top \exp(\mathbf{x}))} \\
&= \mathbf{diag}(\operatorname{softmax}(\mathbf{x})) - \operatorname{softmax}(\mathbf{x})\operatorname{softmax}(\mathbf{x})^\top, \qquad (2.96)
\end{aligned}$$

其中，$\mathbf{diag}(\exp(\mathbf{x}))\,\underline{1}_K = \exp(\mathbf{x})$。

2.6 总结和深入阅读

详细的矩阵偏导数参考https://en.wikipedia.org/wiki/Matrix_calculus。

关于数学优化的内容，可以阅读《Numerical Optimization》[Nocedal and Wright, 2006] 和《Convex Optimization》[Boyd and Vandenberghe, 2004]。

参考文献

James A Anderson and Edward Rosenfeld. *Talking nets: An oral history of neural networks*. MiT Press, 2000.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

Yoshua Bengio, Jean-Sébastien Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS Conference*, 2003.

C.M. Bishop. *Pattern recognition and machine learning*. Springer New York., 2006.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

P.F. Brown, P.V. Desouza, R.L. Mercer, V.J.D. Pietra, and J.C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4): 467–479, 1992.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of*

the 2002 Conference on Empirical Methods in Natural Language Processing, 2002.

Hal Daumé III. A course in machine learning. http://ciml.info/. [Online].

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2001. ISBN 0471056693.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in Neural Information Processing Systems*, pages 472–478, 2001.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

Ian Goodfellow, Aaron Courville, and Yoshua Bengio. Deep learning. Book in preparation for MIT Press, 2015. URL http://goodfeli.github.io/dlbook/.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. In *ICML*, volume 28, pages 1319–1327, 2013.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2001.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

M.I. Jordan. *Learning in Graphical Models*. Kluwer Academic Publishers, 1998.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for*

Computational Linguistics, pages 456–464. Association for Computational Linguistics, 2010.

Marvin Minsky and Papert Seymour. Perceptrons. 1969.

Marvin L Minsky and Seymour A Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press Boston, MA:, 1987.

T.M. Mitchell. *Machine learning*. Burr Ridge, IL: McGraw Hill, 1997.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Albert BJ Novikoff. On convergence proofs for perceptrons. Technical report, DTIC Document, 1963.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

DE Rumelhart GE Hinton RJ Williams and GE Hinton. Learning representations by back-propagating errors. *Nature*, pages 323–533, 1986.

Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

索引