

高级机器学习

第一部分：机器学习数学基础

将封面图命名为 `cover.jpg` 并放入 `figures/` 目录后，会自动显示在这里。

教师： 授课教师
学期： 2026
单位： 学校 / 机构名称
日期： 2026 年 8 月 17 日

目录

第一讲 机器学习数学基础与表示学习

机器学习课程常常从模型、算法和实验开始，但真正支撑这些内容的是一套共同的数学语言。无论是线性分类器、神经网络、注意力机制，还是后续章节中的大语言模型，它们都需要把现实对象转化为向量、矩阵和概率分布，再通过损失函数与优化算法进行学习。因此，本章并不是孤立地复习线性代数、概率统计或优化理论，而是希望把这些工具放回机器学习任务本身：数据如何表示，模型如何计算，误差如何度量，参数如何更新，最终结果又如何在新样本上接受检验。

在自然语言处理场景中，这种联系尤其直观。一个词、一句话或一篇文档原本是离散符号，计算机并不能直接理解其中的语义关系。词袋模型把文本表示为计数向量，TF-IDF 进一步调整词的重要性，Word2Vec 和 GloVe 将词映射到稠密向量空间，Transformer 则在上下文中动态生成表示 [? ? ? ?]。看似不同的方法，本质上都在回答同一个问题：怎样把语言对象放入一个适合计算、比较和学习的空间中。

本章的内容可以看作后续深度学习与表示学习章节的入口。前半部分从向量、矩阵、范数、计算图和梯度下降开始，建立模型训练所需的基本语言；中间部分讨论监督学习、无监督学习、自监督学习、模型评估和正则化，说明机器学习系统如何从数据中获得可泛化的规律；最后部分转向多层感知机、自注意力、Encoder/Decoder 结构以及预训练表示，为理解现代 NLP 模型奠定基础。阅读本章时，建议始终把公式和具体任务联系起来：每一个符号背后都应当对应某类数据、某个模型部件或某种训练目标。

学习目标

- 建立机器学习所需的线性代数、概率统计、矩阵微积分和优化语言。
- 理解监督学习、无监督学习、自监督学习、模型评估和正则化机制。
- 掌握多层感知机、计算图、神经网络训练和 Encoder/Decoder 的基本思想。
- 从特征映射、分布式表示和预训练的角度理解表示学习机制。

1.1 机器学习的数学基础

1.1.1 张量代数与线性对象

向量与矩阵

标量、向量和矩阵是机器学习中最常用的线性对象。标量是单个数值，例如温度、距离或一个样本的损失值；向量是一组有序数值，常用于表示一个样本的特征；矩阵则是二维数值表，常用于表示一批样本、线性变换或模型参数。在本书中，我们使用斜体字母表示标量，例如 a 、 b 、 x ；使用粗体字母表示向量和矩阵。

一个 n 维向量可以写为

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}, \quad (1.1)$$

其中 $\{a_1, a_2, \dots, a_n\}$ 是该向量的元素。若所有元素都是实数，则记为 $\mathbf{a} \in \mathbb{R}^n$ 。类似地，一个 $m \times n$ 的矩阵可以写为

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}, \quad (1.2)$$

其中 m 是矩阵的行数， n 是矩阵的列数， A_{ij} 表示第 (i, j) 个位置的元素。实值矩阵记为 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 。在机器学习中，若一共有 m 个样本、每个样本有 n 个特征，就可以把整个数据集写成一个 $m \times n$ 的矩阵。

常用的特殊矩阵包括零矩阵 $\mathbf{0}$ 和单位矩阵 $\mathbf{I}$ 。单位矩阵是一个方阵，对角线元素为 1，其他元素为 0。向量也可以看作一种特殊的矩阵，例如式 (??) 中的向量就是一个只有一行的矩阵。

矩阵转置

矩阵 $\mathbf{A} \in \mathbb{R}^{m \times n}$ 的转置记为 $\mathbf{A}^\top \in \mathbb{R}^{n \times m}$ ，其含义是交换行和列。也就是说，若 $\mathbf{B} = \mathbf{A}^\top$ ，则 $B_{ji} = A_{ij}$ 。例如，对于向量

$$\mathbf{a} = \begin{bmatrix} 2 & -4 & 6 & 8 \end{bmatrix} \quad (1.3)$$

它的转置为

$$\mathbf{a}^\top = \begin{bmatrix} 2 \\ -4 \\ 6 \\ 8 \end{bmatrix} \quad (1.4)$$

只有一行的向量称为行向量，只有一列的向量称为列向量。实际推导中需要特别留意向量方向，因为它会影响矩阵乘法是否合法。

矩阵的逐元素运算

形状相同的矩阵可以做逐元素加法、减法、乘法和除法。设 $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$ ，则 $\mathbf{A} + \mathbf{B}$ 的每个元素都是两个矩阵对应位置元素之和。下面是一个例子。

$$\begin{aligned} \mathbf{A} + \mathbf{B} &= \begin{bmatrix} 4 & -2 & 5 \\ 1 & 7 & 3 \end{bmatrix} + \begin{bmatrix} 2 & 6 & -1 \\ 8 & 0 & 4 \end{bmatrix} \\ &= \begin{bmatrix} 6 & 4 & 4 \\ 9 & 7 & 7 \end{bmatrix}. \end{aligned} \quad (1.5)$$

逐元素乘法记为 $\mathbf{A} \odot \mathbf{B}$ ，逐元素除法记为 $\mathbf{A} \oslash \mathbf{B}$ 。标量乘法也是常见操作，即用一个标量 k 乘以矩阵 $\mathbf{A}$ 。下面给出 $k = 3$ 且 $\mathbf{A} = \begin{bmatrix} 4 & -2 & 5 \\ 1 & 7 & 3 \end{bmatrix}$ 时的例子。

$$\begin{aligned} k\mathbf{A} &= 3 \begin{bmatrix} 4 & -2 & 5 \\ 1 & 7 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 12 & -6 & 15 \\ 3 & 21 & 9 \end{bmatrix}. \end{aligned} \quad (1.6)$$

这些运算满足常见的交换律、结合律和分配律。对机器学习而言，更重要的是确认参与运算的对象形状是否一致；形状不匹配时，逐元素运算没有定义。

点积

两个同维向量 $\mathbf{a} = [a_1 \ a_2 \ \cdots \ a_n]$ 和 $\mathbf{b} = [b_1 \ b_2 \ \cdots \ b_n]$ 的点积定义为：

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= \sum_{i=1}^n a_i b_i \\ &= a_1 b_1 + a_2 b_2 + \cdots + a_n b_n. \end{aligned} \quad (1.7)$$

在几何意义上，一个实值向量 $\mathbf{a}$ 可以看作一个既有大小（记为 $|\mathbf{a}|$ ）又有方向的几何对象。 $\mathbf{a}$ 和 $\mathbf{b}$ 的点积也可以定义为：

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \times |\mathbf{b}| \times \cos(\theta), \quad (1.8)$$

其中 θ 是 $\mathbf{a}$ 和 $\mathbf{b}$ 之间的夹角。

矩阵乘法

给定矩阵 $\mathbf{A} \in \mathbb{R}^{m \times p}$ 和矩阵 $\mathbf{B} \in \mathbb{R}^{p \times n}$ ，矩阵乘法 $\mathbf{AB}$ 会得到一个矩阵 $\mathbf{C} \in \mathbb{R}^{m \times n}$ ，其元素定义为：

$$\begin{aligned} C_{ij} &= \sum_{k=1}^p A_{ik} \times B_{kj} \\ &= A_{i1} \times B_{1j} + A_{i2} \times B_{2j} + \cdots + A_{ip} \times B_{pj}. \end{aligned} \quad (1.9)$$

矩阵乘法要求 $\mathbf{A}$ 的列数必须等于 $\mathbf{B}$ 的行数。它满足结合律和分配律，并且有 $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$ 。但矩阵乘法通常不满足交换律，也就是说一般不能把 $\mathbf{AB}$ 写成 $\mathbf{BA}$ 。神经网络中的线性层 $\mathbf{xW} + \mathbf{b}$ 正是矩阵乘法、向量加法和非线性函数的组合。

范数

在线性空间中，范数是用于度量向量“长度”的函数。对于向量 $\mathbf{a} \in \mathbb{R}^n$ ， p -范数（或 l_p 范数）定义为：

$$\|\mathbf{a}\|_p = \left(\sum_{i=1}^n |a_i|^p \right)^{1/p}. \quad (1.10)$$

最常用的 p -范数对应 $p = 1, 2$ 和 ∞ ：

$$\|\mathbf{a}\|_1 = \sum_{i=1}^n |a_i| \quad (1.11)$$

$$\|\mathbf{a}\|_2 = \sqrt{\sum_{i=1}^n |a_i|^2} \quad (1.12)$$

$$\|\mathbf{a}\|_\infty = \max\{|a_1|, |a_2|, \dots, |a_n|\}. \quad (1.13)$$

范数也可以用于度量两个点之间的距离。令 $\mathbf{b} \in \mathbb{R}^n$ 为另一个向量，则 $\mathbf{a}$ 和 $\mathbf{b}$ 之间的 p -范数距离为 $\|\mathbf{a} - \mathbf{b}\|_p$ 。

例如，在词袋模型中，特征对应单词的出现频次。设一个很小的词表为

$$V = \{\text{"a"}, \text{"and"}, \text{"as"}, \text{"Bonner"}, \text{"honor"}, \text{"I"}, \\ \text{"met"}, \text{"pig"}, \text{"to"}, \text{"went"}, \text{"wig"}, \text{"without"}\}.$$

句子 “As I went to Bonner” 可以表示为一个 $|V|$ 维向量，其中出现在句子中的词对应位置为 1，未出现的词对应位置为 0。若另一句 “I met a pig” 也被表示成同一词表下的向量，则两个句子的点积会统计它们共享了多少词。这个例子说明，向量并不只是抽象的数学对象；在文本分类中，一篇文档、一条评论甚至一个句子都可以通过向量进入后续的矩阵运算。

1.1.2 高级矩阵微积分与计算图

神经网络可以看成由许多简单函数复合而成的复杂函数。一个深层模型通常不是一次性完成从输入到输出的映射，而是先进行线性变换，再经过非线性激活函数，然后继续进入下一层。为了清楚地描述这种计算过程，我们常使用计算图。

计算图由节点和边组成。节点可以表示变量，也可以表示数学运算；边表示数据从一个节点流向另一个节点。因此，计算图本质上是一种有向图。它可以把复杂的函数拆解成一系列简单操作，使模型的前向计算和梯度计算都更加清晰。

例如，下面三个函数都可以表示为计算图：

$$\mathbf{y} = \mathbf{x} + \mathbf{w}, \quad (1.14)$$

$$\mathbf{y} = \text{Softmax}(\mathbf{x}\mathbf{W} + \mathbf{b}), \quad (1.15)$$

$$\mathbf{y} = \text{Sigmoid}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1) - \text{ReLU}(\mathbf{x}\mathbf{W}_2). \quad (1.16)$$

其中 $\mathbf{x}$ 表示输入向量， $\mathbf{y}$ 表示输出向量， $\mathbf{W}$ 、 $\mathbf{W}_1$ 和 $\mathbf{W}_2$ 表示权重矩阵， $\mathbf{b}$ 和 $\mathbf{b}_1$ 表示偏置向量。Softmax、Sigmoid 和 ReLU 都是常见的非线性函数。

从表达式的角度看，神经网络就是由这些基本运算不断组合而成的数学表达式。从计算图的角度看，每个中间结果都可以看成一个节点，每个运算都对应一次数据变换。

前向传播

计算图最直接的作用是执行模型的前向计算。给定输入之后，模型会按照计算图中边的方向，从输入节点开始逐步计算每个中间节点，直到得到最终输出。这个过程称为前向传播。

例如，考虑如下函数：

$$\mathbf{y} = \psi(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2. \quad (1.17)$$

其中 $\mathbf{x}$ 是输入向量, $\mathbf{W}_1$ 和 $\mathbf{W}_2$ 是权重矩阵, $\mathbf{b}_1$ 是偏置向量, $\psi(\cdot)$ 是激活函数, $\mathbf{y}$ 是输出向量。

为了更清楚地表示这个复合函数, 我们可以引入中间变量:

$$\mathbf{h}_3 = \mathbf{x}\mathbf{W}_1, \quad (1.18)$$

$$\mathbf{h}_2 = \mathbf{h}_3 + \mathbf{b}_1, \quad (1.19)$$

$$\mathbf{h}_1 = \psi(\mathbf{h}_2), \quad (1.20)$$

$$\mathbf{y} = \mathbf{h}_1\mathbf{W}_2. \quad (1.21)$$

这里 $\mathbf{h}_3$ 表示线性变换后的结果, $\mathbf{h}_2$ 表示加入偏置后的结果, $\mathbf{h}_1$ 表示经过激活函数后的隐藏表示, $\mathbf{y}$ 表示最终输出。

这样一来, 原本看起来比较复杂的函数就被拆成了几个简单步骤。前向传播就是依次计算

$$\mathbf{x} \rightarrow \mathbf{h}_3 \rightarrow \mathbf{h}_2 \rightarrow \mathbf{h}_1 \rightarrow \mathbf{y}.$$

这种分解方式不仅有助于理解神经网络的计算过程, 也方便程序自动执行模型推理。

链式法则与反向传播

训练神经网络时, 我们不仅需要计算模型输出, 还需要知道如何调整参数, 使损失函数变小。设损失函数为 $\mathcal{L}$, 它衡量模型输出 $\mathbf{y}$ 和真实答案之间的差异。为了更新参数, 我们需要计算损失函数对参数的偏导数, 例如

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{W}_2}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1}.$$

由于神经网络是复合函数, 梯度计算需要使用链式法则。对于一个简单的复合函数

$$y = p(q(x)),$$

链式法则给出

$$\frac{\partial y}{\partial x} = \frac{\partial p}{\partial q} \frac{\partial q}{\partial x}.$$

也就是说, 如果一个变量通过若干中间变量影响最终输出, 那么它对最终输出的影响可以沿着计算路径逐层相乘。

在神经网络中, 反向传播就是链式法则在计算图上的系统化应用。前向传播从输入走向输出, 而反向传播则从损失函数出发, 沿着计算图反方向传播梯度 [? ?]。

仍然考虑前面的计算过程：

$$\mathbf{h}_3 = \mathbf{x}\mathbf{W}_1, \quad (1.22)$$

$$\mathbf{h}_2 = \mathbf{h}_3 + \mathbf{b}_1, \quad (1.23)$$

$$\mathbf{h}_1 = \psi(\mathbf{h}_2), \quad (1.24)$$

$$\mathbf{y} = \mathbf{h}_1\mathbf{W}_2. \quad (1.25)$$

设

$$\delta_{\mathbf{y}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}},$$

其中 $\delta_{\mathbf{y}}$ 表示损失函数对模型输出的梯度。若使用平方损失，例如

$$\mathcal{L} = \frac{1}{2} \|\mathbf{y} - \mathbf{y}_{\text{gold}}\|_2^2,$$

其中 $\mathbf{y}_{\text{gold}}$ 是真实输出向量，则有

$$\delta_{\mathbf{y}} = \mathbf{y} - \mathbf{y}_{\text{gold}}.$$

接下来，梯度可以沿计算图反向传播。首先，由于

$$\mathbf{y} = \mathbf{h}_1\mathbf{W}_2,$$

所以有

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \mathbf{W}_2^\top, \quad (1.26)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_2} = \mathbf{h}_1^\top \frac{\partial \mathcal{L}}{\partial \mathbf{y}}. \quad (1.27)$$

这里 $\mathbf{W}_2^\top$ 是 $\mathbf{W}_2$ 的转置， $\mathbf{h}_1^\top$ 是 $\mathbf{h}_1$ 的转置。第一式表示损失对隐藏表示 $\mathbf{h}_1$ 的梯度，第二式表示损失对参数矩阵 $\mathbf{W}_2$ 的梯度。

由于

$$\mathbf{h}_1 = \psi(\mathbf{h}_2),$$

根据链式法则，有

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_1} \odot \psi'(\mathbf{h}_2).$$

其中 $\psi'(\mathbf{h}_2)$ 表示激活函数在 $\mathbf{h}_2$ 处的导数， $\odot$ 表示逐元素乘法。这个公式说明：经过激活函数反传时，梯度需要乘上激活函数在前向传播中留下的局部变化率。

为了便于理解，可以把前向传播和反向传播看作同一张计算图上的两次遍历。前向传播从输入 $\mathbf{x}$ 与参数开始，逐步得到输出 $\mathbf{y}$ ；反向传播则从损失 L 开始，沿相反方

向把梯度传回每个参数。图 ?? 给出了一个典型例子。图中的蓝色编号表示前向计算顺序，红色编号表示反向求导顺序。这样，复杂神经网络的求导过程就可以被拆成一系列局部规则。

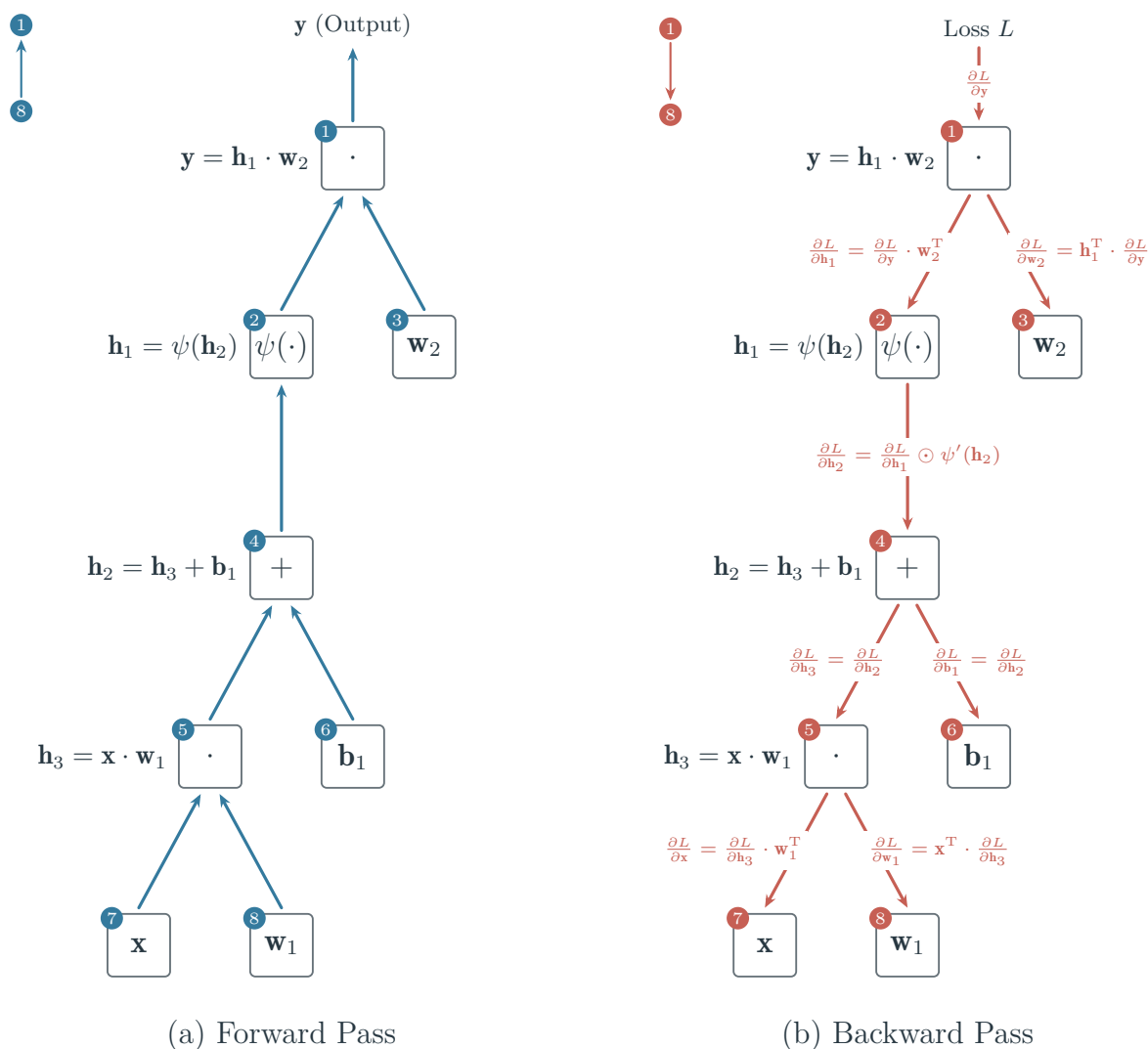


图 1.1: 前向传播与反向传播示例。前向传播计算中间变量和输出，反向传播沿计算图反方向逐层计算梯度。

重点提示

计算图的重点是“局部计算”和“局部求导”。前向传播时，每个节点只需要根据输入计算自己的输出；反向传播时，每个节点只需要知道上游梯度和本节点的局部导数，再把梯度传给前一层。难点在链式法则的方向：前向传播按数据流方向计算数值，反向传播按相反方向累积梯度。理解这一点后，多层神经网络的训练就不再是一大团公式，而是一组局部规则的重复应用。

1.1.3 凸优化理论与训练目标

优化问题的目标是在给定的可行域中寻找一组变量，使目标函数取得最小值或最大值。在机器学习中，这些变量通常是模型参数，目标函数通常由训练损失和正则化项组成。凸优化是一类具有特殊结构的优化问题。设参数的可行域为凸集 $\mathcal{C}$ 。如果对于任意 $\theta_1, \theta_2 \in \mathcal{C}$ 和任意 $\alpha \in [0, 1]$ ，都有

$$\alpha\theta_1 + (1 - \alpha)\theta_2 \in \mathcal{C},$$

则称 $\mathcal{C}$ 为凸集。直观上，凸集中任意两点之间的线段都完全位于该集合内部。凸优化的重要性质是：任意局部最优解都是全局最优解。因此，优化算法不容易被非全局的局部极小值困住，得到的解也更容易分析和验证 [?]。

训练目标的优化形式

在机器学习中，模型训练通常可以写成一个优化问题。设训练数据集为

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N,$$

其中 N 表示训练样本的数量， $\mathbf{x}^{(i)}$ 表示第 i 个输入样本的特征向量， $\mathbf{y}^{(i)}$ 表示第 i 个样本对应的真实输出。这里使用粗体字母表示向量。若输出只是一个标量标签，也可以将 $\mathbf{y}^{(i)}$ 简化写为 $y^{(i)}$ 。

设模型为 f_{θ} ，其中 θ 表示模型参数。对于输入样本 $\mathbf{x}^{(i)}$ ，模型的预测输出为

$$\hat{\mathbf{y}}^{(i)} = f_{\theta}(\mathbf{x}^{(i)}),$$

其中 $\hat{\mathbf{y}}^{(i)}$ 表示模型对第 i 个样本的预测结果。模型训练的目标，就是选择一组参数 θ ，使预测结果 $\hat{\mathbf{y}}^{(i)}$ 尽可能接近真实输出 $\mathbf{y}^{(i)}$ 。

为了衡量模型预测和真实答案之间的差异，我们需要定义损失函数。损失函数通常记为

$$\mathcal{L}(f_{\theta}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}),$$

其中 $\mathcal{L}$ 表示单个样本上的预测误差。损失越小，说明模型对该样本的预测越接近真实答案。

对于整个训练集，我们关心的不是某一个样本上的误差，而是所有训练样本上的平均误差。因此，可以定义经验风险：

$$\hat{\mathcal{L}}(\theta) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f_{\theta}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}).$$

其中 $\hat{\mathcal{L}}(\theta)$ 表示模型在训练集上的平均损失，也称为经验风险。机器学习的训练目标

通常就是最小化经验风险：

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \hat{\mathcal{L}}(\boldsymbol{\theta}).$$

这里 $\hat{\boldsymbol{\theta}}$ 表示通过训练得到的最优参数。这个公式表达了一个非常核心的思想：训练模型，就是在参数空间中寻找一组参数，使训练集上的平均损失最小。

不同的机器学习模型可能有不同的参数形式，不同任务也可能使用不同的损失函数，但许多训练过程都可以归结为类似的优化问题。

例如，在回归任务中，模型输出通常是连续值。设真实输出为向量 $\mathbf{y}$ ，模型预测输出为

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}),$$

其中 $\mathbf{x}$ 是输入特征向量， $\hat{\mathbf{y}}$ 是模型预测值。常用的平方损失可以写为

$$\mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) = \|f_{\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{y}\|_2^2.$$

其中 $\|\cdot\|_2$ 表示向量的 L_2 范数。平方损失会对较大的预测误差给予更大的惩罚，因此常用于线性回归和许多连续值预测问题。

在分类任务中，模型通常输出每个类别的概率。设共有 C 个类别，模型输出的概率向量为

$$\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}) = \begin{bmatrix} p_{\boldsymbol{\theta},1}(\mathbf{x}) & p_{\boldsymbol{\theta},2}(\mathbf{x}) & \cdots & p_{\boldsymbol{\theta},C}(\mathbf{x}) \end{bmatrix},$$

其中 $p_{\boldsymbol{\theta},c}(\mathbf{x})$ 表示模型认为输入 $\mathbf{x}$ 属于第 c 类的概率。真实标签可以表示为 one-hot 向量

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_C \end{bmatrix},$$

其中只有真实类别对应的位置为 1，其他位置为 0。此时常用的交叉熵损失为

$$\mathcal{L}(\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}) = - \sum_{c=1}^C y_c \log p_{\boldsymbol{\theta},c}(\mathbf{x}).$$

如果模型给真实类别分配的概率越大，损失就越小；如果模型对真实类别很不确定，甚至给出很小的概率，损失就会变大。因此，交叉熵损失非常适合用于分类模型的训练。

正则化目标

只让训练误差尽可能小并不总是最好的策略。一个模型如果过度贴合训练数据，可能会记住训练集中的细节甚至噪声，却无法很好地处理新的样本。这种现象称为过拟合。

为了缓解过拟合，训练目标中通常会加入正则化项。正则化的作用是限制模型复杂度，使模型不要为了降低训练误差而变得过于复杂。加入正则化之后，训练目标可

以写成

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \left[\hat{\mathcal{L}}(\boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}) \right],$$

其中 $\Omega(\boldsymbol{\theta})$ 是正则化项，用来刻画模型参数的复杂程度； λ 是正则化系数，用来控制正则化的强度。当 λ 较大时，模型会更重视控制参数复杂度；当 λ 较小时，模型会更重视降低训练误差。

常见的正则化方式包括 L_2 正则化和 L_1 正则化。 L_2 正则化通常写为

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2.$$

其中 $\|\boldsymbol{\theta}\|_2$ 表示参数向量 $\boldsymbol{\theta}$ 的 L_2 范数。 L_2 正则化会惩罚过大的参数，使模型参数整体保持较小的规模。在神经网络和线性模型中， L_2 正则化也常被称为权重衰减。直观地说，如果一个模型必须依赖非常大的参数才能拟合训练数据，那么它可能对数据中的细微变化过于敏感；而限制参数大小可以让模型更加平滑，也更容易泛化到新的数据。

L_1 正则化通常写为

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1.$$

其中 $\|\boldsymbol{\theta}\|_1$ 表示参数向量 $\boldsymbol{\theta}$ 的 L_1 范数。 L_1 正则化倾向于让一部分参数变为零，因此常用于稀疏建模和特征选择。也就是说， L_1 正则化会鼓励模型只保留少数真正重要的特征，而忽略不重要的特征。

比如设房价预测模型使用面积、房龄、地铁距离等特征进行线性回归。若只最小化平方误差，模型可能给某个偶然相关的特征分配很大的权重。加入 L_2 正则化后，目标函数可写为

$$\min_{\mathbf{w}, b} \frac{1}{N} \sum_{i=1}^N (\mathbf{x}^{(i)} \mathbf{w} + b - y^{(i)})^2 + \lambda \|\mathbf{w}\|_2^2.$$

这样做会抑制过大的权重，使模型对单个特征的依赖更平滑。

正则化目标体现了机器学习中的一个重要平衡：模型既要能够拟合训练数据，又不能过度复杂。经验风险 $\hat{\mathcal{L}}(\boldsymbol{\theta})$ 关注训练误差，正则化项 $\Omega(\boldsymbol{\theta})$ 关注模型复杂度，而正则化系数 λ 决定二者之间的权衡 [?]。

凸优化与梯度下降

当训练目标被写成优化问题之后，一个自然的问题是：我们如何找到使目标函数最小的参数？凸优化理论为这个问题提供了重要的数学工具。

设目标函数为 $J(\boldsymbol{\theta})$ ，其中 $\boldsymbol{\theta}$ 表示模型参数向量， $J(\boldsymbol{\theta})$ 表示在参数 $\boldsymbol{\theta}$ 下需要最小化的目标值。这个目标函数可以是经验风险 $\hat{\mathcal{L}}(\boldsymbol{\theta})$ ，也可以是加入正则化之后的目标函数

$$J(\boldsymbol{\theta}) = \hat{\mathcal{L}}(\boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}).$$

如果对任意两个参数向量 θ_1 、 θ_2 ，以及任意 $\alpha \in [0, 1]$ ，都有

$$J(\alpha\theta_1 + (1 - \alpha)\theta_2) \leq \alpha J(\theta_1) + (1 - \alpha)J(\theta_2),$$

则称 $J(\theta)$ 是凸函数。

凸函数有一个非常好的性质：局部最优解也是全局最优解。也就是说，如果我们在某一点附近找不到更好的解，那么这个点就是整个参数空间中的最优解。正因为如此，凸优化问题通常比非凸优化问题更容易分析，也更容易得到稳定可靠的解。

当目标函数可微时，最常用的优化方法之一是梯度下降。梯度表示目标函数上升最快的方向，因此如果我们希望减小目标函数，就可以沿着负梯度方向更新参数。设第 t 步的参数为 θ_t ，学习率为 η ，则梯度下降的更新规则为

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} J(\theta_t).$$

其中 $\nabla_{\theta} J(\theta_t)$ 表示目标函数 J 在当前参数 θ_t 处对参数的梯度，学习率 η 控制每次更新的步长。

如果学习率太小，模型训练会非常缓慢；如果学习率太大，参数可能在最优点附近来回震荡，甚至导致训练不稳定。因此，学习率是模型训练中非常重要的超参数。

在实际训练中，尤其是当训练数据很多时，我们通常不会每次都使用全部训练样本计算梯度，而是使用随机梯度下降或小批量梯度下降。小批量梯度下降每次只使用一部分样本估计梯度，在计算效率和梯度稳定性之间取得平衡，因此成为现代机器学习和深度学习中最常用的训练方式之一。

例如，若训练目标是一个碗形曲面，参数点一开始可能位于曲面边缘。每一次梯度下降都会沿着当前最陡下降方向移动一小步。图 ?? 用等高线表示损失曲面：红色箭头并不直接跳到最低点，而是通过多次局部更新逐渐接近 minimum。这个过程也解释了为什么学习率需要合适：步长太小会走得很慢，步长太大则可能越过最低点。

重点提示

优化部分需要区分三个概念：目标函数、优化算法和超参数。目标函数说明“什么样的模型算好”，梯度下降说明“怎样朝更好的方向移动”，学习率、批量大小和正则化系数则控制移动方式。凸优化中的局部最优等于全局最优，但深度神经网络通常是非凸问题，因此实际训练更关注稳定下降、泛化表现和验证集效果，而不是严格证明找到全局最优点。

1.1.4 高维数据的表示与计算

前面介绍的向量、矩阵和优化方法，最终都要用于真实数据。现实中的一个样本往往包含许多特征，例如图像由大量像素组成，工业设备同时记录多种传感器信号，表格数据也可能包含许多字段。把这些特征组织成向量后，样本就成为高维空间中的

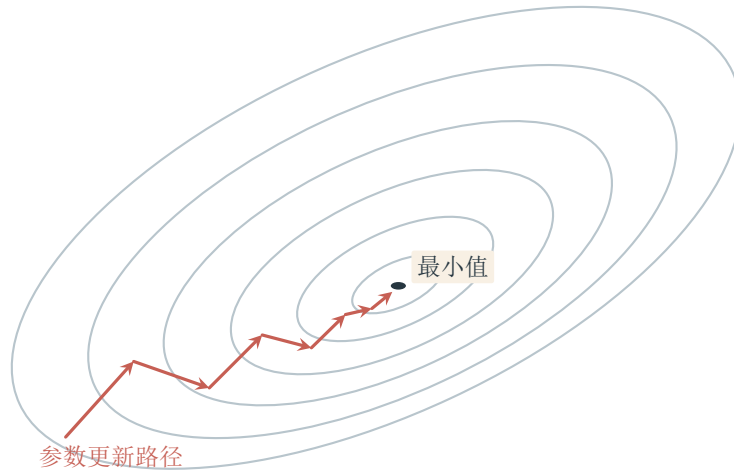


图 1.2: 梯度下降示意图。参数沿负梯度方向逐步移动，使目标函数值不断降低。

一个点；机器学习模型则在这些高维向量上进行计算。

从样本向量到数据矩阵

设数据集包含 N 个样本，每个样本有 d 个特征，可以写成 $\mathbf{x}^{(i)} \in \mathbb{R}^d$ 。将所有样本按行排列，得到数据矩阵

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(N)})^T \end{bmatrix} \in \mathbb{R}^{N \times d}. \quad (1.28)$$

其中，每一行对应一个样本，每一列对应一个特征。例如，一张彩色图像可以由所有像素值组成高维向量；一篇文档也可以由词频组成向量。这些对象的形式虽然不同，进入模型后都可以统一表示为向量或矩阵。

高维数据上的模型计算

模型利用前面介绍的线性代数运算处理高维数据。以线性模型为例，单个样本的预测分数为

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b, \quad (1.29)$$

其中， $\mathbf{w}$ 为每个特征分配一个权重。对全部样本进行预测时，可以写成矩阵形式

$$\mathbf{s} = \mathbf{X}\mathbf{w} + b\mathbf{1}. \quad (1.30)$$

因此，向量点积对应单个样本的计算，矩阵乘法则可以同时处理一批样本。随着特征维度和样本数量增加，需要保存的数据和完成的运算也会增多，这正是高维计算首先需要考虑的问题。

稀疏性与维度控制

高维向量不一定每个位置都有有效信息。以词袋表示为例，假设词表包含数万个词，一篇短文档通常只出现其中几十个词，因此它的词袋向量虽然维度很高，但大部分分量为零。对于这类数据，可以只保存非零值及其位置，并在计算时跳过其余分量，这就是稀疏表示与稀疏计算。

有些高维数据还包含重复、相关或与任务无关的特征。过多的无关信息不仅增加计算量，也可能使模型更难从有限样本中找到稳定规律。特征选择可以保留较重要的原始特征，PCA可以把数据线性投影到较低维空间，自编码器则可以用神经网络学习压缩表示。这些方法都可以概括为

$$\mathbf{z} = \phi(\mathbf{x}), \quad \mathbf{z} \in \mathbb{R}^r, \quad r < d, \quad (1.31)$$

其中， $\mathbf{x}$ 是原始高维向量， $\mathbf{z}$ 是更紧凑的表示。降维并不是简单地让维度越少越好，而是尽量删除冗余信息，同时保留对后续任务有用的结构。

重点提示

高维数据处理可以概括为三个步骤：先把每个样本表示为特征向量，再用向量和矩阵运算完成模型计算，最后根据数据特点选择稀疏计算、特征选择或降维。更具体的特征映射和表示学习方法将在后文继续讨论。

至此，前面介绍的向量、矩阵和优化方法已经与实际数据连接起来。下一节将进一步讨论模型从这些数据中学习时，监督信号可以来自哪里，以及不同学习范式之间有什么区别。

1.2 机器学习机制与优化范式

1.2.1 监督学习、无监督学习与自监督学习

监督学习

有监督学习处理带标注样本，此处的含义是输入 $\mathbf{x}$ 对应一个输出 y 。给定一组输入输出配对 $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(K)}, y^{(K)})\}$ ，本场景的任务是学习一个函数 $f(\cdot)$ ，将每个 $\mathbf{x}^{(k)}$ 映射至 $y^{(k)}$ ：

$$y^{(k)} = f(\mathbf{x}^{(k)}). \quad (1.32)$$

该过程被称为有监督学习，因为函数 $f(\cdot)$ 的学习过程由 $\mathbf{x}^{(k)}$ 对应的人工标注标准答案 $y^{(k)}$ 提供引导。一般来说， $y^{(k)}$ 称作 $\mathbf{x}^{(k)}$ 的真值或标准标注。在此之后，当新输入 $\mathbf{x}_{\text{new}}$ 出现时，我们使用训练得到的函数 $f(\cdot)$ 进行输出预测。若预测结果 $f(\mathbf{x}_{\text{new}})$ 与真值 y_{new} 保持一致，则称本次有监督学习取得成功。

绝大多数自然语言处理任务都可以建模为有监督学习问题。为文档分配类别无疑是最简单的场景之一。其他自然语言处理任务包括但不限于：生成标签序列、一段文本、句法树或图结构。

例如在有监督学习的情感分类任务中，训练数据可以由“这部电影节奏很紧凑”及其标签“正面”、“剧情拖沓且人物单薄”及其标签“负面”组成。模型学习的是从文本到情感标签的映射。类似地，命名实体识别需要为句子中的每个词标注人名、地名、机构名等标签；机器翻译则把源语言句子映射为目标语言句子。

Tokens :	Most	are	expected	to	fall	below	previous-	levels	.
							month		
POS tags :	JJS	VBP	VBN	TO	VB	IN	JJ	NNS	.
Chunk tags :	<u>B-NP</u>	<u>B-VPI-VP</u>	<u>I-VPI-VP</u>	<u>B-PP</u>	<u>B-NP</u>	<u>I-NP</u>	O		
	NP	VP	VP	PP	NP	NP			

图 1.3: 词性标注与短语块识别示例。序列标注任务需要为输入序列中的每个位置预测标签。

无监督学习

与有监督学习相对，无监督学习处理无标注样本；换言之，对于每个样本，我们仅拥有输入 $\mathbf{x}$ ，不存在对应的正确输出 y 。这种情况下，我们需要一种算法，从无标注数据 $\{\mathbf{x}^{(k)}\}$ 中学习由 $\mathbf{x}^{(k)}$ 映射至某一输出的函数。由于不存在人为干预定义函数输出，这类算法需要挖掘 $\{\mathbf{x}^{(k)}\}$ 中存在的内在模式，并依据某些准则优化 $\{\mathbf{x}^{(k)}\}$ 的表示方式与函数输出。这些准则通常借鉴人类先验知识，使最终学习得到的函数能够输出符合我们预期的结果。

无监督学习的一个常见示例是词聚类。该方法对词汇集合分组，将语义相近的词语划分至同一簇。基于这一准则，我们可以约束函数 $f(\cdot)$ ，让相近词语输出同一簇编号。难度更高的场景是无监督双语词典自动构建。该方法无需平行语料，即可学习两种语言之间的词汇级映射关系。解决该问题的常用思路之一，是利用不同语言词表征空间的同构特性。

半监督学习

介于有监督学习与无监督学习二者之间的是半监督学习。该范式适用于仅少量输入数据带有标注、绝大部分数据无标注的场景，因此能够同时吸收两种范式的优势。例如在机器翻译任务中，我们会拥有一定规模的平行标注数据，同时还有数量高出数个数量级的单语无标注数据。常规做法是先基于平行数据以有监督方式训练基础模型；随后可利用大规模单语数据对该基础模型的组件进行优化。实现方式分为两种：一是将双语数据训练得到的翻译模型与目标语言单语数据训练得到的语言模型相结合；二是先用单语数据预训练模型部分模块，再基于双语标注数据对整个模型微调。

广义而言，所有学习算法都需要监督信号。这个结论看似违背直觉，因为无监督学习似乎不存在任何真值数据作为信号。但从通用机器学习视角来看，监督信号不应局限于人工提供的标注。即便是无监督学习任务，学习过程依然受先验知识与输入数据 $\{\mathbf{x}^{(k)}\}$ 内部隐藏分布模式的监督约束。从这个角度来讲，无监督学习并非严格意义上“无监督的学习”。

以这种广义的“监督”概念为基准，机器学习衍生出更多范式，无法简单归入传统有监督或无监督分类，强化学习便是一例。强化学习建模智能体在环境中执行连续决策的过程。智能体执行动作后会从环境中获得反馈（或称奖励）。强化学习的目标是学习一套决策策略，使智能体每一步行动累积得到的总奖励最大化[?]。多数场景下，完整奖励仅在智能体抵达终止状态时才会产生，例如游戏胜利或失败。正因如此，强化学习擅长解决具备延迟奖励（或称稀疏奖励）特征的问题，这也是其与标准有监督学习的核心区别：标准有监督学习依靠预先定义、每一步即时生效的监督信号。这种延迟奖励特性适配大量自然语言处理任务。例如文本生成系统从左至右逐词输出文本，但往往需要完整生成全文并完成评估后，才能判断单个词汇选择是否恰当。

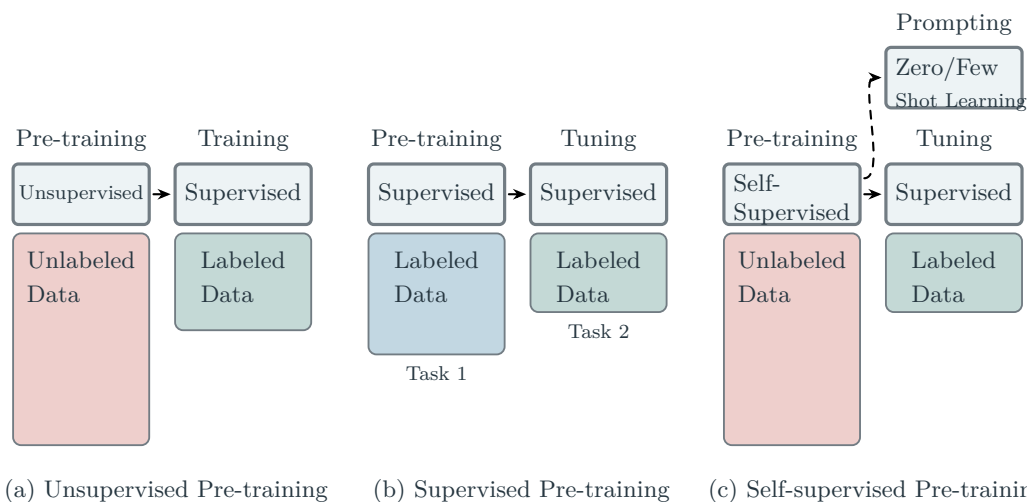


图 1.4: 无监督、有监督与自监督预训练范式示意。不同范式的核心差别在于训练信号的来源。

自监督学习

另一类范式是自监督学习。该范式以有监督学习的思路解决无监督学习问题。核心思想是将无监督学习任务转化为辅助任务，借助该辅助任务来求解原始目标任务。在辅助任务中，真值可由输入数据自身生成。因此模型学习过程依靠自主生成的信号完成监督，而非人工标注标签 [?]。

语言模型预测下一个词是最经典的自监督任务之一。例如给定“模型需要学习数据中的”，模型要预测后续可能出现的词。遮蔽语言模型也是常见例子：把句子中的一部分词替换为 [MASK]，要求模型根据上下文恢复原词。图像领域也有类似思想，例如把一张图片切成若干块，遮住其中一块后让模型重建缺失区域；语音模型则可以遮蔽一小段声学特征，再要求模型恢复它 [???]。

可以用同一批影评文本区分这几种学习方式。若每条影评后面都有“正面”或“负面”标签，模型学习的是从文本到标签的监督映射；若只有影评本身，算法可以尝试把它们自动聚成若干主题簇，如“剧情讨论”“演员讨论”“视听效果讨论”；若把句子“这部电影的镜头很漂亮”改成“这部电影的镜头很 [MASK]”，并要求模型恢复“漂亮”，则人工标签并未出现，监督信号来自文本自身。这样的例子可以帮助我们看到：所谓学习范式的差别，关键不在数据是不是文本，而在训练信号如何被构造出来。

自监督学习在自然语言处理领域取得了巨大成功。预训练或许是推动近年技术发展最具代表性的应用。预训练流程中，基础模型（如语言模型）通过自监督学习在海量语料上完成训练，随后将学习得到的参数迁移至下游任务模型（如情感分析模型）。该方案具备两大核心优势：第一，自监督模型无需人工标注，可基于近乎无限规模的文本数据进行训练；第二，预训练阶段学到的表征泛化能力极强，能够适配各类下游任务。在预训练章节中，我们将结合多个实例介绍自监督学习在现代自然语言处理模型中的应用。

重点提示

学习范式的区分标准不是“数据长什么样”，而是“监督信号从哪里来”。有监督学习依赖人工标注，无监督学习依赖数据内部结构，自监督学习则从输入本身构造预测目标，例如预测下一个词或恢复被遮蔽的词。难点在于：自监督学习虽然不需要人工标签，但它仍然有明确的训练目标，因此并不是没有监督信号；它只是把监督信号从人工标注转移到了数据自身。

1.2.2 机器学习建模机制

机器学习建模的核心，是把现实任务转化为定义明确的数学问题。完整的建模过程至少要回答五个相互衔接的问题：输入和输出分别是什么，原始对象如何表示，选用怎样的模型族，依据什么目标学习参数，以及如何把模型输出转化为最终预测。模型族规定了可选择函数的范围，训练算法则在这个范围内寻找参数；二者相关，但不是同一个概念。

下面以文本分类作为贯穿例子。垃圾邮件过滤、情感分析和主题识别虽然标签含义不同，但都要求模型根据一篇文档输出一个类别。以“判断文档是否与食物有关”为例，一条带标注样本由原始文档及其标签组成。这一任务足够简单，可以清楚展示从任务定义到预测的完整链条；其中的表示、模型族、学习目标和预测规则也能推广到回归、序列标注等其他任务。

任务形式化

设 $\mathcal{X}_{\text{raw}}$ 为原始文档空间， $d \in \mathcal{X}_{\text{raw}}$ 表示一篇文档， $C = \{c_1, \dots, c_K\}$ 为有限标签集合， $y \in C$ 为文档的真实标签。训练集可写为

$$\mathcal{D}_{\text{train}} = \{(d^{(i)}, y^{(i)})\}_{i=1}^N. \quad (1.33)$$

单标签分类的目标是学习预测函数 $h: \mathcal{X}_{\text{raw}} \rightarrow C$ ，使它在未见文档上也能给出正确标签。 $K=2$ 时是二分类， $K>2$ 时是多类分类。多标签分类则允许一个样本同时属于多个类别，其输出通常写成 $\{0,1\}^K$ 中的向量；它与“从 K 个类别中选一个”的单标签分类不是同一个输出空间。先明确输出空间，才能选择合适的模型和损失函数。

任务定义还要区分真实规律与模型。通常假设文档及其标签来自某个未知的联合分布 $p(d, c)$ ，其中条件分布 $p(c|d)$ 描述给定文档时标签的不确定性。训练集只是该联合分布的有限样本，模型只能借助这些样本构造近似规则，而不能直接获得真实分布。

输入表示

现实任务中的原始输入可以是文档、图像、音频或表格记录。它们具有不同的形式和含义，不能直接参与分类模型中的点积、矩阵乘法等数值运算。因此，在送入模型之前，需要先把原始输入转换为由数值组成的向量，这个过程称为输入表示或向量化。设原始输入为 s ，向量化方法为 $\phi(\cdot)$ ，则

$$\mathbf{x} = \phi(s), \quad \mathbf{x} \in \mathbb{R}^n. \quad (1.34)$$

其中， $\mathbf{x} = [x_1, \dots, x_n]^\top$ 是模型实际读取的向量，每个分量 x_i 表示输入的一个数值特征。

不同类型的数据可以采用不同的向量化方法。表格数据通常将连续字段直接作为数值特征，并对类别字段进行 one-hot 编码；图像可以由像素值组成，并进一步通过图像特征提取器得到向量；音频可以先转换为采样值或时频谱，再提取声学特征向量；文本则可以使用词频、TF-IDF、词向量或文档向量表示。具体方法虽然不同，目的都是得到模型能够计算的数值向量 $\mathbf{x}$ 。

下面以文本分类为例。设原始输入 d 是一篇文档，词袋模型通过统计词语出现次数把它转换为向量。给定词表 $V = \{V_1, \dots, V_n\}$ ，文档向量 $\mathbf{x} = \phi(d)$ 的第 i 个分量定

义为

$$x_i = \text{count}(V_i, d), \quad i = 1, \dots, n. \quad (1.35)$$

例如，设词表为 $V = \{\text{食物}, \text{餐厅}, \text{好吃}\}$ ，文档为“这家餐厅的食物很好吃”，那么对应的词袋向量为 $\mathbf{x} = [1, 1, 1]^\top$ 。词袋向量保留了词语的出现次数，但没有保留词序。其他向量化方法可以改变特征的定义，例如 TF-IDF 使用加权词频，词向量和文档向量则用连续数值表示语义信息。无论采用哪种方法，输入表示的作用都是把文档 d 转换为后续模型可以处理的向量 $\mathbf{x}$ 。

模型族与预测规则

得到 $\mathbf{x}$ 之后，需要规定哪些输入输出关系可以被模型表达，这一集合称为模型族或假设空间。分类模型常见的输出形式有两种。一种是为每个类别计算实数分数 $s_{\theta}(\mathbf{x}, c)$ ，再选择分数最高的类别：

$$\hat{c} = \arg \max_{c \in C} s_{\theta}(\mathbf{x}, c). \quad (1.36)$$

另一种是输出条件概率模型 $q_{\theta}(c|\mathbf{x})$ ，其中 $q_{\theta}(c|\mathbf{x}) \geq 0$ 且 $\sum_{c \in C} q_{\theta}(c|\mathbf{x}) = 1$ 。此时也可用最大后验概率规则进行预测。概率模型能够表达预测的不确定性，而一般的分数模型不一定给出经过校准的概率。

多类线性分类器为每个类别 c 设置权重向量 $\mathbf{w}_c \in \mathbb{R}^n$ 和偏置 $b_c \in \mathbb{R}$ ，并定义

$$s_{\theta}(\mathbf{x}, c) = \mathbf{w}_c^\top \mathbf{x} + b_c, \quad (1.37)$$

其中 $\theta = \{\mathbf{w}_c, b_c\}_{c \in C}$ 。将所有类别的分数组成向量

$$\mathbf{s}_{\theta}(\mathbf{x}) = [s_{\theta}(\mathbf{x}, c_1), \dots, s_{\theta}(\mathbf{x}, c_K)]^\top. \quad (1.38)$$

这些分数可以是任意实数，不能直接解释为概率。softmax 函数及其第 c 个分量定义为

$$\mathbf{q}_{\theta}(\mathbf{x}) = \text{Softmax}(\mathbf{s}_{\theta}(\mathbf{x})), \quad (1.39)$$

$$\begin{aligned} q_{\theta}(c|\mathbf{x}) &= [\text{Softmax}(\mathbf{s}_{\theta}(\mathbf{x}))]_c \\ &= \frac{\exp(s_{\theta}(\mathbf{x}, c))}{\sum_{c' \in C} \exp(s_{\theta}(\mathbf{x}, c'))}. \end{aligned} \quad (1.40)$$

例如，二分类模型的分数为 2 和 1 时，

$$\text{Softmax}\left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right) = \frac{1}{e^2 + e} \begin{bmatrix} e^2 \\ e \end{bmatrix} \approx \begin{bmatrix} 0.73 \\ 0.27 \end{bmatrix}. \quad (1.41)$$

softmax 的每个输出均为正，并且全部输出之和为 1。它不改变类别分数的大小次序，因此按最大概率和按最大分数预测会得到同一类别。

对于二分类 $C = \{c_a, c_b\}$ ，决策边界由两个类别分数相等的位置给出：

$$(\mathbf{w}_a - \mathbf{w}_b)^\top \mathbf{x} + (b_a - b_b) = 0. \quad (1.42)$$

它在 n 维特征空间中是一个超平面。以食物主题识别为例，可令 x_1 表示食物词计数， x_2 表示餐厅、口味等上下文词计数。图 ?? 中，超平面 1 和 2 都正确划分了图中的训练样本，超平面 3 则产生了误分类。仅凭训练样本无法断定前两个边界哪个在新数据上更好，这还取决于学习目标、正则化和验证结果。

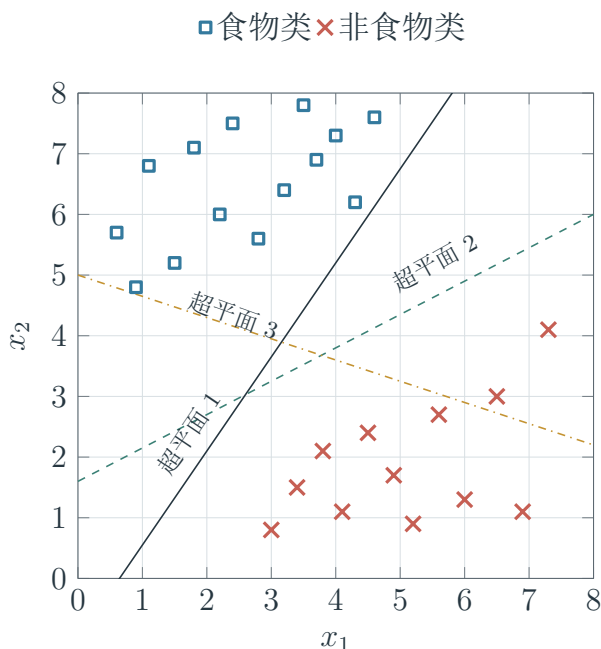


图 1.5: 二维特征空间中的线性决策边界。不同超平面可以具有相同的训练误差，却未必具有相同的泛化性能。

线性模型只能在当前特征空间中表达线性决策边界。若任务需要非线性边界，可以改变表示函数 ϕ ，使用核方法，或直接采用含非线性变换的模型 [?]。

两种分类思路：判别式与生成式

文档已经表示为向量 $\mathbf{x}$ ，接下来要判断它属于“食物类”还是“非食物类”。判别式与生成式是解决这一问题的两种不同思路。它们最终都输出类别，区别在于模型学习过程中首先关注什么。

判别式建模：直接判断类别。 判别式模型直接学习输入向量与类别之间的关系。对于食物主题分类，它会学习哪些特征更支持“食物类”，哪些特征更支持“非食物类”，再根据这些特征计算类别分数或条件概率 $q_\theta(c|\mathbf{x})$ 。前面介绍的 softmax 线性分

类器就是判别式模型：它直接比较各类别的分数，并把分数最高的类别作为预测结果。logistic regression 和支持向量机也属于这一类方法。

生成式建模：先描述每一类文档，再进行判断。生成式模型先分别描述“食物类文档通常是什么样”和“非食物类文档通常是什么样”。具体来说，它学习类别出现的概率 $p(c)$ ，以及已知类别后出现某种文档向量的概率 $p(\mathbf{x}|c)$ 。面对一篇新文档时，模型会分别假设它属于每个候选类别，再比较哪种假设更能解释这篇文档。根据 Bayes 公式，比较各类别的后验概率等价于比较下面的乘积：

$$\hat{c} = \arg \max_{c \in C} p(c)p(\mathbf{x}|c). \quad (1.43)$$

其中， $p(c)$ 反映某个类别本身有多常见， $p(\mathbf{x}|c)$ 反映该类别下出现当前文档向量的可能性。

朴素 Bayes 是典型的生成式分类器。训练时，它统计每个类别包含多少篇文档，以及不同词在各类别中出现的频率。例如，“餐厅”和“好吃”在食物类文档中经常出现，那么包含这些词的新文档在“食物类”假设下就会得到较高概率。若某个词在某一类别的训练文档中从未出现，通常使用加性平滑避免整个文档概率变为零 [?]。这里的“生成式”表示模型描述了每个类别产生何种输入的概率规律，并不特指生成文本的模型。

重点提示

判别式模型问：“给定这篇文档，它属于哪个类别？”生成式模型问：“如果它属于某个类别，出现这篇文档是否合理？”两种方法都能完成分类，但建模的出发点不同。

生成式与判别式也不同于线性与非线性。前者区分模型是直接学习分类关系，还是先学习各类别下的输入分布；后者描述决策函数的形状。一个判别式模型可以是线性的，也可以是非线性的。选择哪种建模思路，需要结合数据规模、特征表示和任务需求判断。

参数学习

模型族确定后，训练数据用于选择其中的具体参数。沿用前文定义，常见的正则化经验风险最小化形式为

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \left[\hat{\mathcal{L}}(\boldsymbol{\theta}) + \lambda \Omega(\boldsymbol{\theta}) \right]. \quad (1.44)$$

对于 softmax 分类器，在不计正则化时，最小化交叉熵等价于最大化训练标签的条件似然；对于朴素 Bayes，无平滑的最大似然估计可由类别数和词频计数直接得到，加性平滑则在这些计数估计上作进一步修正。由此也能看出，参数学习不一定都依赖梯

度下降：有些模型存在闭式解或计数形式，有些模型则需要数值优化。

完成训练后，新文档 d_{new} 先通过同一个表示函数得到 $\mathbf{x}_{\text{new}} = \phi(d_{\text{new}})$ ，再由模型计算类别分数或后验概率，最后依据预测规则输出 $\hat{c}$ 。因此，本例的逻辑链条可以概括为

任务定义 $\rightarrow$ 输入表示 $\rightarrow$ 模型族与建模路线 $\rightarrow$ 参数学习 $\rightarrow$ 预测。

回归、序列标注和结构化预测会采用不同的输出空间与预测规则，但仍需依次回答这几类问题。下一节将在模型族已经确定的前提下，进一步讨论损失函数和优化算法如何求得参数。

1.2.3 优化算法与训练过程

上一节已经确定了模型 f_{θ} 及其参数 θ ，训练要做的就是根据数据不断调整这些参数 [?]。一次基本训练迭代可以概括为

输入 $\rightarrow$ 前向传播得到预测 $\rightarrow$ 计算损失 $\rightarrow$ 反向传播计算梯度 $\rightarrow$ 更新参数。

前向传播根据当前参数得到预测，损失函数衡量预测与标准答案之间的差异；反向传播再利用链式法则计算损失对各层参数的梯度，SGD、Adam 等优化器根据这些梯度给出参数更新量 [??]。需要注意，反向传播负责“算出梯度”，优化器负责“利用梯度更新参数”，二者是训练过程中的两个不同步骤。

训练阶段无法直接优化模型在未知测试数据上的真实性能，因此需要在训练数据上定义一个可计算的损失作为替代目标。本节先介绍损失如何度量预测误差，再说明梯度、SGD、Adam 和小批量训练如何共同完成参数更新。

损失函数与训练目标

设模型输出为 $\mathbf{y}_{\theta} = f_{\theta}(\mathbf{x})$ ，标准答案为 $\mathbf{y}_{\text{gold}}$ 。单个样本的损失 $L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}})$ 用来度量模型输出和标准答案之间的差异，多个样本损失的平均值构成训练目标 $\mathcal{L}(\theta)$ 。常见损失包括 0-1 损失、边际损失、排序损失和对比损失。

0-1 损失直接统计预测标签是否等于真实标签。令

$$c_{\theta} = \arg \max_c y_{\theta}(c), \quad c_{\text{gold}} = \arg \max_c y_{\text{gold}}(c),$$

则

$$L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}}) = \begin{cases} 1, & c_{\theta} \neq c_{\text{gold}}, \\ 0, & c_{\theta} = c_{\text{gold}}. \end{cases} \quad (1.45)$$

这个损失符合分类直觉，但不可微，直接优化比较困难。因此实践中常使用交叉熵、负对数似然、hinge loss 等可优化的替代损失。

边际损失关注正确标签与错误标签之间的分数差距。令 c_{gold} 为正确标签， c 为错误标签，则边际可写为

$$\text{margin}(c, c_{\text{gold}}) = y_{\theta}(c_{\text{gold}}) - y_{\theta}(c). \quad (1.46)$$

大间隔训练希望正确类别的分数显著高于错误类别。若 $\Delta(c, c_{\text{gold}})$ 表示把 c_{gold} 误判为 c 的代价，则一种边际损失为 [?]]

$$L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}}) = \max_c (0, y_{\theta}(c) - y_{\theta}(c_{\text{gold}}) + \Delta(c, c_{\text{gold}})). \quad (1.47)$$

排序损失用于需要对一组候选对象排序的任务，例如信息检索、分类排序和度量学习。其基本思想是惩罚预测排序与标准排序之间不一致的候选对。对比损失则常用于对比学习：给定一个样本，模型需要拉近正样本表示，同时推远负样本表示 [??]。若 $\mathbf{y}^+$ 表示正样本输出， $\mathbf{Y}^-$ 表示负样本集合，一种对比目标可以写为

$$L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}}) = \log \frac{D(\mathbf{y}_{\theta}, \mathbf{y}^+)}{\sum_{\mathbf{y}^- \in \mathbf{Y}^-} D(\mathbf{y}_{\theta}, \mathbf{y}^-)}, \quad (1.48)$$

其中 $D(\alpha, \beta)$ 是 α 与 β 之间的距离或相似度度量。如何构造正样本和负样本，是对比学习中的关键设计问题。

另一种改进训练目标的思路，是把模型输出 $\mathbf{y}_{\theta}$ 看成从概率分布中采样得到的随机变量。此时单个损失也变为随机变量，可以在所有可能预测上取期望：

$$\begin{aligned} L(\mathbf{x}, \mathbf{y}_{\text{gold}}) &= \mathbb{E}_{\mathbf{y}_{\theta} \sim \Pr(\cdot|\mathbf{x})} [L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}})] \\ &= \sum_{\mathbf{y}_{\theta}} L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}}) \cdot \Pr(\mathbf{y}_{\theta}|\mathbf{x}). \end{aligned} \quad (1.49)$$

这种方法把所有可能预测纳入损失估计，常称为期望损失、期望风险或 Bayesian risk。

反向传播与基本更新规则

定义可微的训练目标后，反向传播计算它对当前参数的梯度。设第 t 次迭代使用的训练目标为 $\mathcal{L}_t$ ，当前参数为 $\boldsymbol{\theta}_t$ ，则梯度记为

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta}_t). \quad (1.50)$$

向量 $\mathbf{g}_t$ 与参数 $\boldsymbol{\theta}_t$ 形状相同，表示每个参数变化时损失如何变化。最基本的梯度下降更新为

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{g}_t, \quad (1.51)$$

其中， t 是迭代步数， $\eta > 0$ 是学习率。负梯度方向是当前使损失下降最快的局部方向，因此每次更新都沿 $-\mathbf{g}_t$ 移动一小步。

SGD 与小批量训练

设训练集共有 N 个样本。训练时没有必要在每次更新中都计算整个训练集的损失。令 $\mathcal{S}_t$ 表示第 t 次迭代抽取的样本集合， $B = |\mathcal{S}_t|$ 表示其中的样本数，则当前小批量的平均损失为

$$\mathcal{L}_t(\boldsymbol{\theta}) = \frac{1}{B} \sum_{(\mathbf{x}, \mathbf{y}_{\text{gold}}) \in \mathcal{S}_t} L(f_{\boldsymbol{\theta}}(\mathbf{x}), \mathbf{y}_{\text{gold}}). \quad (1.52)$$

若每次只抽取一个样本，即 $B = 1$ ，就是随机梯度下降（Stochastic Gradient Descent, SGD）；若 $1 < B < N$ ，就是实践中更常用的 mini-batch SGD；若每次使用全部 N 个训练样本，则是全批量梯度下降。SGD 中的“随机”表示每一步使用随机抽取的训练样本来估计整体梯度。

小批量中的输入和标准答案可以分别堆叠成矩阵，一次前向传播得到整批预测，再对这 B 个样本的损失取平均。反向传播由这个平均损失得到 $\mathbf{g}_t$ ，优化器随后更新参数。这样既能利用 GPU 的矩阵计算能力，也能使梯度比单样本估计更稳定。

动量方法与 Adam

基本 SGD 只使用当前梯度。动量方法还保留一部分历史更新方向：

$$\mathbf{u}_t = \mu \mathbf{u}_{t-1} - \eta \mathbf{g}_t, \quad (1.53)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{u}_t. \quad (1.54)$$

其中， $\mathbf{u}_t$ 是第 t 步的更新方向， $0 \leq \mu < 1$ 是动量系数。历史方向较一致时，动量会加快移动；梯度来回变化时，历史平均则可以减弱震荡 [?]。

AdaGrad、RMSProp 等自适应方法会根据历史梯度大小为不同参数调整更新幅度 [?]。Adam（Adaptive Moment Estimation）把动量思想和自适应更新结合起来 [?]。它首先用当前梯度 $\mathbf{g}_t$ 更新两个向量：

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t, \quad (1.55)$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\mathbf{g}_t \odot \mathbf{g}_t). \quad (1.56)$$

其中, $\mathbf{m}_t$ 是梯度的指数移动平均, 用来估计整体更新方向; $\mathbf{v}_t$ 是梯度平方的指数移动平均, 用来估计各参数梯度的尺度; $\beta_1, \beta_2 \in [0, 1)$ 是控制历史信息保留程度的衰减系数; $\odot$ 表示逐元素乘法。

由于 $\mathbf{m}_0$ 和 $\mathbf{v}_0$ 通常初始化为零, 训练初期的估计会偏向零, 因此 Adam 使用

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \quad (1.57)$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t} \quad (1.58)$$

进行偏置修正, 最终按照

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \quad (1.59)$$

更新参数。这里的平方根和除法都按元素进行, $\epsilon > 0$ 是防止分母为零的稳定项。简单来说, $\mathbf{m}_t$ 决定主要更新方向, $\mathbf{v}_t$ 根据近期梯度大小调节每个参数的步幅。

无论使用 SGD、动量还是 Adam, 学习率 η 都控制更新的基本步长。图 ?? 展示了学习率过小、合适和过大时的典型优化路径。

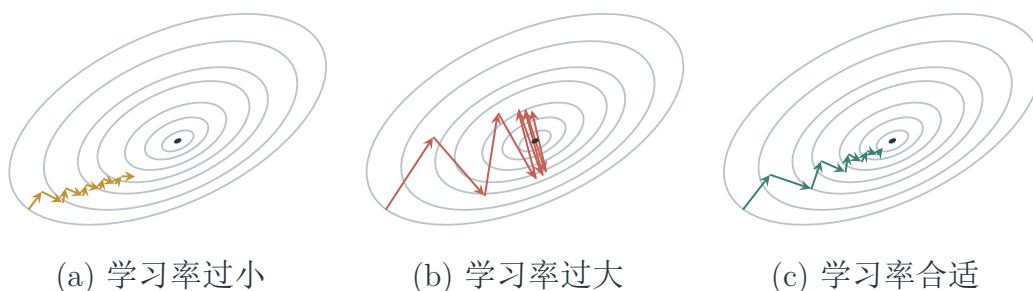


图 1.6: 不同学习率下的优化路径示意。学习率过小收敛慢, 过大则容易在最优点附近震荡。

例如, 在情感分类中, 一个 mini-batch 可以包含 32 条评论。模型并不是先完整学会第一条评论, 再学习第二条评论, 而是把 32 条评论组成矩阵 $\mathbf{X}$, 一次前向传播得到 32 个预测分布, 再把 32 个交叉熵损失取平均。这样得到的梯度代表这一小批样本共同给出的更新方向。若 batch 太小, 更新方向可能受个别样本影响而波动较大; 若 batch 太大, 每一步计算成本又会上升。实际训练中选择 batch size, 本质上是在梯度噪声、计算效率和硬件容量之间折中。

1.2.4 模型评估与模型选择

上一节讨论了如何通过优化降低训练目标, 但训练损失下降只说明模型越来越适合训练数据, 并不能直接说明它能处理未见样本。训练完成后还要回答两个问题: 候

选模型中应该选择哪一个，以及选定模型在新数据上究竟表现如何 [?]。本节按照一次完整的评估流程，依次讨论模型选择、数据划分、泛化误差、模型复杂度、评价指标和显著性检验。

模型选择与模型评估

模型选择和模型评估服务于不同阶段。模型选择发生在系统开发过程中，目标是从不同模型结构、超参数或训练检查点中选出较好的方案。例如，学习率、正则化强度和训练轮数都可以依据验证集表现确定。模型评估则发生在模型确定之后，目标是估计该模型在未见数据上的最终性能。

两者必须分开。如果一边查看测试结果，一边据此调整模型，那么测试集实际上参与了模型选择，最终结果就会过于乐观。因此，机器学习实验通常把可用数据划分为训练集、验证集和测试集，让每部分数据只承担一种主要职责。

训练集、验证集与测试集

训练集用于计算训练目标并更新模型参数；验证集用于比较候选模型、调节超参数和决定何时停止训练；测试集只在模型及其设置确定后使用，用于报告最终性能。常见划分比例包括 60%/20%/20% 或 80%/10%/10%，但具体比例应根据数据规模和任务特点决定。

这种划分能够估计泛化性能的前提，是各部分数据能够代表同一个目标分布。对于普通独立样本，可以随机划分；对于时间序列、同一用户产生的多条记录或同一文档的多个片段，则要按照时间、用户或文档分组，避免高度相关的样本同时进入训练集和测试集。

数据量较小时，单次划分的结果可能受抽样偶然性影响。 k 折交叉验证把开发数据划分为 k 个互斥子集，每轮使用其中 $k - 1$ 份训练、剩余一份验证，最后汇总各轮结果。即使采用交叉验证，独立测试集仍应保留到模型选择完成之后。

重点提示

训练集用于学习参数，验证集用于选择模型，测试集用于最终评估。反复根据测试集结果修改模型会造成测试信息泄漏，使测试集不再代表真正的未见数据。

训练误差与泛化误差

设训练集包含 N 个样本，单样本损失为 $\mathcal{L}$ 。模型在训练集上的平均误差为

$$\text{Err}_{\text{train}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \mathbf{y}^{(i)}) . \quad (1.60)$$

真实的泛化误差是模型在目标数据分布上的期望损失，但该分布通常未知，因此无法直接计算。验证误差和测试误差都是使用未参与参数更新的数据对泛化误差作出的有限样本估计：验证误差用于模型选择，测试误差用于最终报告。

训练误差降低不保证验证误差同步降低。图 ?? 展示了模型复杂度增加时的典型变化：训练误差持续下降，验证误差却往往先下降后上升。模型选择应关注验证误差最低的位置，而不是训练误差最低的位置。

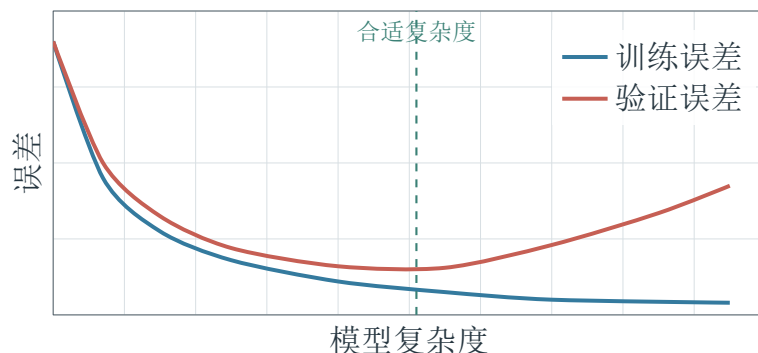


图 1.7: 训练误差与验证误差随模型复杂度变化的示意。训练误差下降并不必然带来更好的泛化性能。

欠拟合、过拟合与模型复杂度

模型复杂度可以直观理解为模型能够表达多少种输入输出关系。增加特征数量、参数数量、网络宽度或深度，通常会提高表达能力，但也会增加模型记忆训练数据偶然细节的可能性。模型选择的关键不是让模型尽可能复杂，而是找到与数据规模和任务难度相匹配的复杂度。

图 ?? 从决策边界的角度比较了欠拟合、合适拟合和过拟合。过于简单的边界无法捕捉数据中的主要规律；复杂度合适的边界能够表达稳定模式；过于曲折的边界则开始追随个别训练样本和噪声。

欠拟合时，模型没有充分学到训练数据中的规律，因此训练误差和验证误差通常都较高。过拟合时，模型在训练集上表现很好，验证误差却明显较高。例如，文本分类器可能记住训练集中频繁出现的专有名词，而没有学到真正能够迁移到新文档的语义线索。

偏差-方差权衡为这一现象提供了另一种解释。在平方损失下，期望预测误差可分为偏差平方、方差和不可约噪声。过于简单的模型通常偏差较高，过于复杂的模型则更容易对训练数据变化产生较大波动，即方差较高 [?]。下一节介绍的正则化和 early stopping，正是控制模型复杂度、缓解过拟合的常用方法。

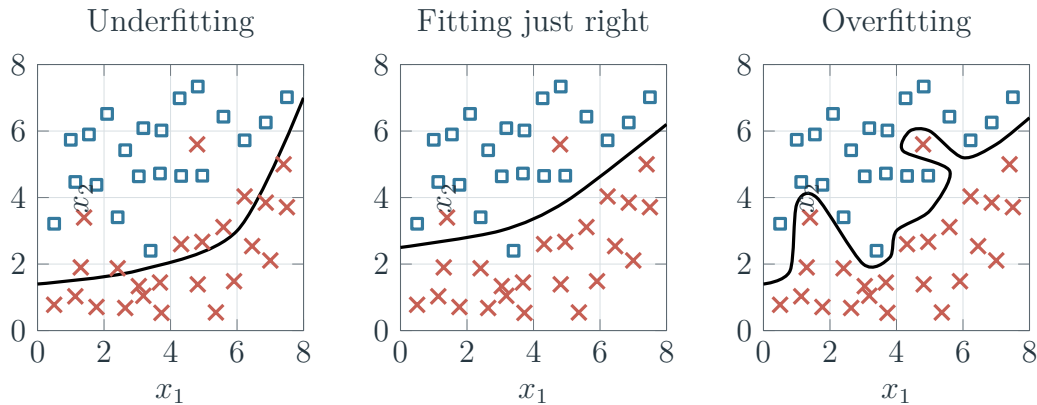


图 1.8: 欠拟合、合适拟合与过拟合示意。模型复杂度过低或过高都会影响泛化能力。

评价指标与任务目标

确定数据划分和候选模型之后，还要选择与任务目标一致的评价指标。分类任务常用 precision、recall 和 F_1 。若 TP 、 FP 和 FN 分别表示真正例、假正例和假负例，则

$$\text{precision} = \frac{TP}{TP + FP}, \quad (1.61)$$

$$\text{recall} = \frac{TP}{TP + FN}, \quad (1.62)$$

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \quad (1.63)$$

precision 衡量模型给出的正类预测有多少是正确的，recall 衡量所有真实正类中有多少被模型找出， F_1 则综合二者。

例如，在命名实体识别中，测试集共有 40 个真实人名，系统预测出 50 个片段，其中 30 个确实是人名。此时 precision 为 $30/50 = 0.60$ ，recall 为 $30/40 = 0.75$ ， F_1 约为 0.67。precision 高但 recall 低的系统较为谨慎，recall 高但 precision 低的系统则更倾向于多报。不同应用对错误类型的容忍度不同，因此不能只凭单一指标判断所有任务。

不同任务需要不同指标。例如，机器翻译常使用 BLEU [?]，生成任务也可能使用人工评价或基于模型的评价。训练损失和最终指标的作用也不同：训练损失需要便于优化，评价指标则要尽可能反映任务目标，二者不一定完全一致。

性能差异与显著性检验

当系统 A 的评价指标略高于系统 B 时，还要判断这一差异是否可能来自测试样本或随机训练过程的偶然波动。显著性检验把“两个系统没有可靠差异”设为原假设，

并计算在原假设成立时观察到当前差异或更极端差异的概率。例如，可以写为

$$H_0: \text{系统} A \text{ 不优于系统} B, \quad H_1: \text{系统} A \text{ 优于系统} B.$$

其中 H_0 是原假设, H_1 是备择假设。若 p -value 小于预先设定的显著性水平 α , 则拒绝原假设。需要注意, p -value 不是“原假设为真的概率”, 统计显著也不等于提升在实际应用中足够重要 [? ? ?]。NLP 实验常使用配对检验、bootstrap 或 approximate randomization 等方法, 因为两个系统通常在同一批测试样本上进行比较 [? ? ?]。

完整的模型开发流程至此形成闭环: 训练集用于学习参数, 验证集用于选择模型, 测试集和合适的评价指标用于报告性能, 显著性检验用于判断模型差异是否可靠。如果评估结果显示模型过拟合, 下一步就需要通过正则化等方法限制模型复杂度。

1.2.5 正则化机制

正则化是控制模型复杂度、缓解过拟合的重要方法。前面已经看到, 可以在训练目标中加入正则项:

$$\hat{\theta} = \arg \min_{\theta} L(\mathbf{y}_{\theta}, \mathbf{y}_{\text{gold}}) + \alpha \cdot R, \quad (1.64)$$

其中 R 是惩罚模型复杂度的函数, 例如惩罚参数幅度或非零参数数量; α 是控制正则化强度的超参数。

范数惩罚是最常见的正则化方法之一。设 $\mathbf{W}$ 是神经网络参数矩阵, 可以使用 L_1 范数或 L_2 范数构造惩罚项:

$$R_1(\mathbf{W}) = \|\mathbf{W}\|_1, \quad (1.65)$$

$$R_2(\mathbf{W}) = \|\mathbf{W}\|_2^2. \quad (1.66)$$

L_2 正则化会限制参数整体幅度, 使模型函数更平滑; L_1 正则化倾向于产生稀疏参数, 使一部分权重接近或等于零。二者都可以看作向训练目标注入先验: 偏好更简单、更稳定的模型。

dropout 是神经网络中非常常用的正则化方法 [? ?]。它在训练时随机屏蔽部分神经元连接, 使网络不能过度依赖某些特定隐藏单元。对某一层

$$\mathbf{y} = \psi(\mathbf{x} \cdot \mathbf{W} + \mathbf{B}),$$

dropout 可引入一个掩码向量 $\mathbf{M}_{\text{drop}}$:

$$\mathbf{y} = \mathbf{M}_{\text{drop}} \odot \psi(\mathbf{x} \cdot \mathbf{W} + \mathbf{B}). \quad (1.67)$$

掩码中的每个位置以概率 ρ 保留, 以概率 $1 - \rho$ 置零。这样, 每次训练更新时, 模型

都像是在训练一个由原网络随机抽取出的子网络。测试时通常使用完整网络，并通过缩放保持激活的期望一致。

图 ?? 展示了 dropout 的直观过程。训练时，一部分神经元和连接被随机关闭，网络必须在不完整的子网络上完成预测；测试时，完整网络重新启用，并通过缩放保持激活期望一致。这个机制迫使不同隐藏单元承担更稳健的表示功能，而不是让模型把某个训练样本的判断完全寄托在少数连接上。

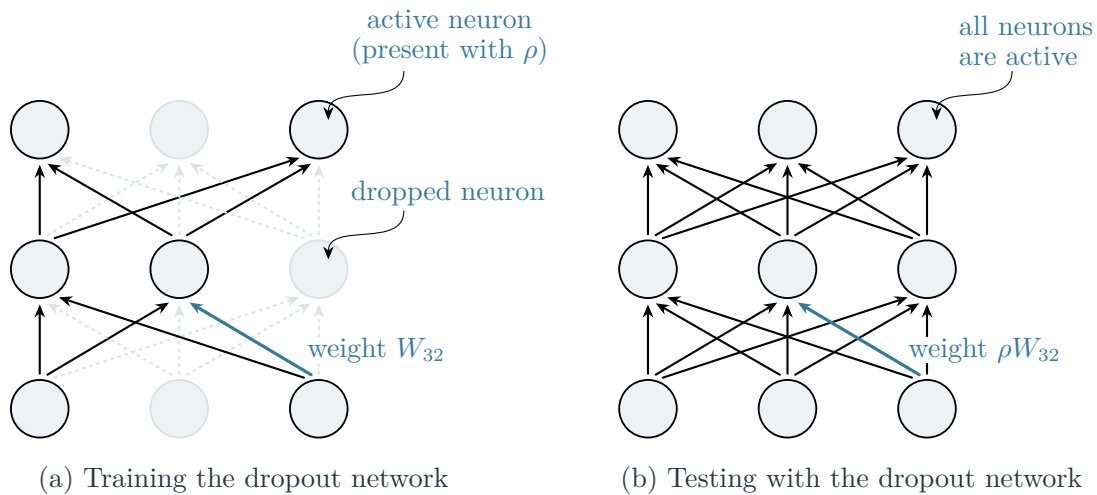


图 1.9: Dropout 的训练与测试过程。训练阶段随机关闭部分神经元，测试阶段使用完整网络。

early stopping 通过监控验证集上的模型状态来决定何时停止训练。它可以被看作一种模型选择方法：在训练不足和过拟合之间寻找平衡 [??]。实践中，验证误差曲线并不总是理想的 U 型曲线，而是常常会波动并出现多个局部最优点。因此，常见停止条件包括：若若干训练步内性能变化低于阈值，则停止；若参数变化低于阈值，则停止；若一段时间内平均性能开始下降，则停止；或若一段时间内的最好性能开始退化，则停止。实际系统中也可以保存多个 checkpoint，并对最后若干 checkpoint 做参数平均或模型集成。

输出概率平滑也是一种正则化思想。在统计学中，平滑指降低极端数据点的影响，并把概率质量重新分配给普通数据点或未观察事件。给定分布 $\mathbf{p} = [p_1, \dots, p_n]$ ，一种简单方法是把它与均匀分布线性插值：

$$\hat{p}_k = (1 - \epsilon) \cdot p_k + \frac{\epsilon}{n}. \quad (1.68)$$

若 $\mathbf{p}$ 是 one-hot 向量，该操作会把原本集中在单一类别上的概率质量缩小，并给所有维度分配 $\frac{\epsilon}{n}$ 的概率。label smoothing 正是采用类似形式来软化类别标签，避免模型对训练答案过度自信 [?]。

例如，在三分类任务中，硬标签 $[1, 0, 0]$ 表示第一个类别绝对正确。若使用 $\epsilon = 0.15$ 的 label smoothing，目标分布可变为近似 $[0.90, 0.05, 0.05]$ 。这并不是说其他两个类别

也真的正确，而是在训练时告诉模型：不要把概率全部压到单一类别上。对于含噪标注或类别边界模糊的文本任务，这种软化常常能改善泛化。

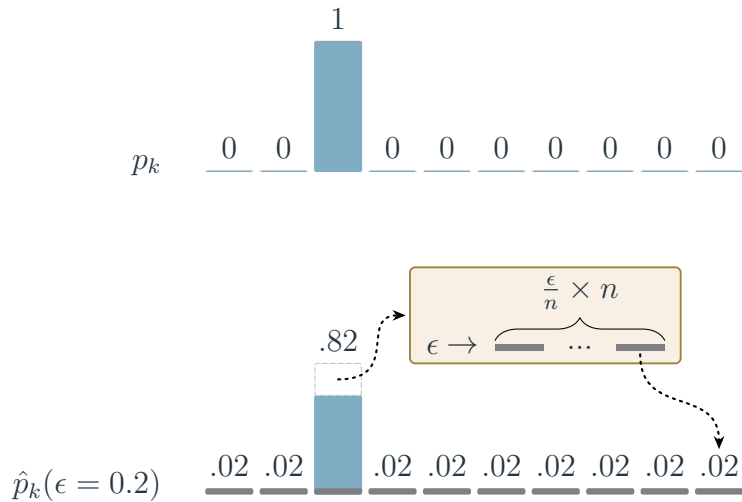


图 1.10: 输出概率平滑示意。平滑会把部分概率质量分配给非标准类别，降低过度自信。

噪声训练基于这样一种思想：模型应当学会在非理想条件下工作，从而避免精确记忆训练样本中的极端点。噪声可以注入训练目标，也可以注入输入、中间隐藏状态、输出、参数或梯度 [? ? ? ? ?]。一个常见选择是高斯噪声。对激活向量 $\mathbf{x} \in \mathbb{R}^n$ ，可写为

$$\mathbf{x}_{\text{noise}} = \mathbf{x} + \mathbf{g}, \quad (1.69)$$

其中 $\mathbf{g}$ 的各个元素独立采样自高斯分布。若把噪声加入层输入，则有

$$\mathbf{y} = \psi((\mathbf{x} + \mathbf{g}_{\text{input}}) \cdot \mathbf{W} + \mathbf{B}). \quad (1.70)$$

噪声只在训练阶段出现，测试时模型通常不再加入噪声。dropout 也可以被理解为一种噪声训练：它随机屏蔽激活，使每个神经元都在带扰动的环境中学习。

噪声训练还有一个额外好处：随机扰动会为同一样本产生不同训练版本，相当于扩大训练样本空间。这与数据增强思想相连。数据增强通过从已有样本构造新样本来扩大训练分布覆盖范围。NLP 中的例子包括同义词替换、词序交换、删除或插入词 [? ? ?]，也包括机器翻译中的回译：先训练目标语言到源语言的反向翻译系统，再用它把额外目标语言单语数据翻译成源语言，从而构造新的双语训练数据 [?]。更进一步，还可以构造对抗样本训练模型，使其在容易误导模型的输入上也能保持鲁棒 [? ?]。

1.3 神经网络基础

本节在多层感知机和神经网络训练机制的基础上，进一步引入序列建模中最重要两类结构：注意力机制与 Encoder/Decoder 架构。前者使模型能够显式建模序列内部或两个序列之间的关联，后者则为机器翻译、文本摘要、问答和预训练模型提供了统一的建模框架。

1.3.1 多层感知机机制

单层感知机是最简单的神经网络之一 [?]。设输入特征向量为 $\mathbf{x}$ ，权重为 $\mathbf{w}$ ，偏置为 b ，仿射变换为

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b. \quad (1.71)$$

标准感知机通过阶跃激活函数输出二值标签：

$$y = \begin{cases} 1, & f(\mathbf{x}) > 0, \\ 0, & \text{其他情况}. \end{cases} \quad (1.72)$$

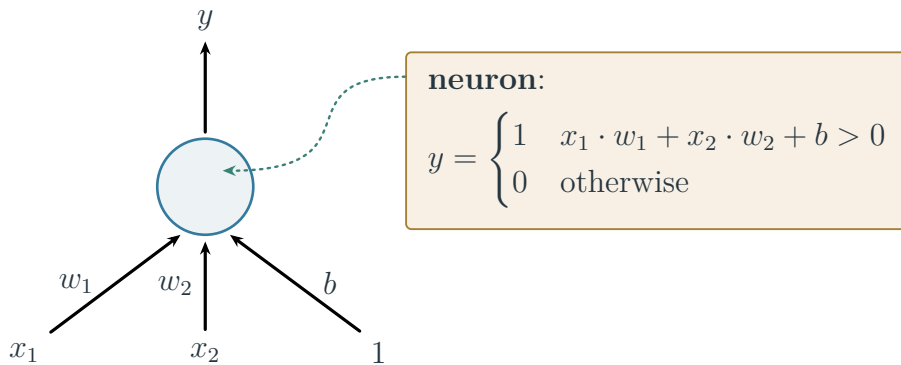


图 1.11: 单个感知机神经元示意。输入特征经线性加权后通过阈值函数输出类别。

当多个神经元组成一层时，单层网络可写为

$$\mathbf{y} = \psi(f(\mathbf{x})), \quad (1.73)$$

$$f(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}, \quad (1.74)$$

其中 $\psi(\cdot)$ 是激活函数。常见激活函数包括 sigmoid、tanh、ReLU、softmax 等。多层感知机把多个层堆叠起来，形成函数复合 [?]：

$$\mathbf{y} = \text{Softmax}(\text{Sigmoid}(\text{ReLU}(\mathbf{x}\mathbf{W}_1)\mathbf{W}_2) \mathbf{W}_3 + \mathbf{b}_3). \quad (1.75)$$

多层网络的深度通常指层数，宽度指每层神经元数量。增加深度和宽度会提升表

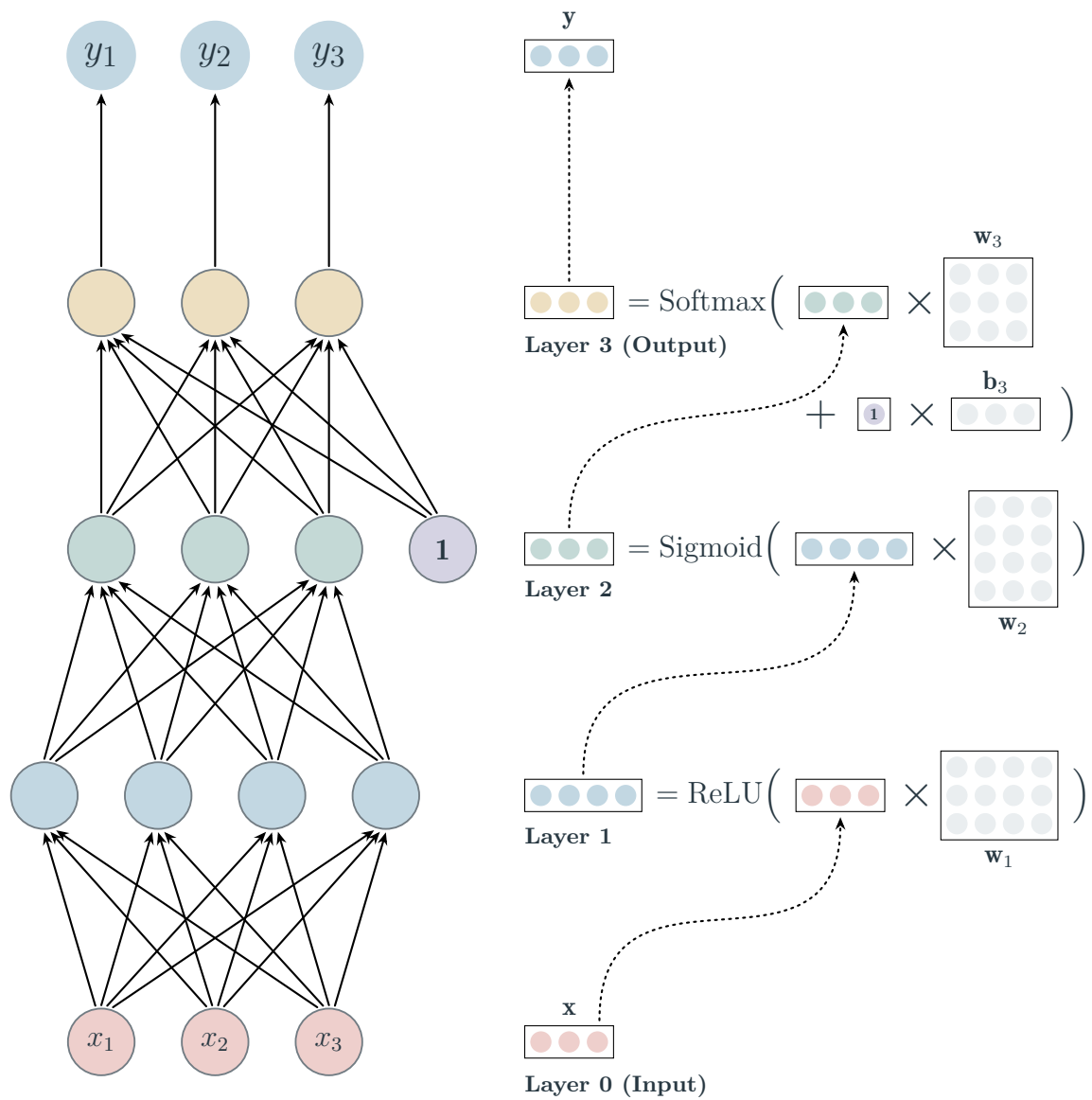


图 1.12: 多层神经网络结构示意。隐藏层逐层变换输入表示, 并最终产生输出分布。

示能力，但也会增加优化难度和过拟合风险。

神经语言模型是多层网络在 NLP 中的经典案例 [??]。给定词序列 $w_1, \dots, w_m$ ，语言模型分解为

$$\Pr(w_1, \dots, w_m) = \prod_{i=1}^m \Pr(w_i | w_1, \dots, w_{i-1}). \quad (1.76)$$

神经语言模型用连续向量表示词和上下文，再用神经网络预测下一个词的概率。

例如，给定短语“自然语言处理很”，语言模型需要预测下一个词。传统 n -gram 模型主要依靠离散词串计数；神经语言模型则先把“自然”“语言”“处理”“很”映射成词向量，再由多层网络得到上下文表示，最后输出词表上的概率分布。若模型给“重要”的概率最高，就可以生成“自然语言处理很重要”。这个例子说明，多层感知机中的隐藏层可以把离散词序列转化为连续上下文表示，再用于预测。

1.3.2 自注意力机制

注意力机制最初在序列到序列建模中被广泛使用 [?]。早期 Encoder/Decoder 模型常把整个源端序列压缩成一个固定长度向量，再由解码器生成目标序列。这个做法容易在长序列上失效，因为一个固定长度向量很难承载所有细粒度信息。注意力机制的思想是：在每个解码位置，模型都可以根据当前状态从源端序列中自适应地选择更重要的信息。

设编码器把源端序列表示为 $\mathbf{h}_1, \dots, \mathbf{h}_m$ ，解码器在第 i 个位置的状态为 $\mathbf{s}_i$ 。注意力模型为该位置构造一个上下文向量：

$$\mathbf{c}_i = \sum_{j=1}^m \alpha_{i,j} \mathbf{h}_j, \quad (1.77)$$

其中 $\alpha_{i,j}$ 表示在计算 $\mathbf{c}_i$ 时模型对 $\mathbf{h}_j$ 的依赖程度。常见做法是先计算对齐分数 $a(\mathbf{s}_i, \mathbf{h}_j)$ ，再通过 softmax 归一化：

$$\alpha_{i,j} = \frac{\exp(a(\mathbf{s}_i, \mathbf{h}_j))}{\sum_{j'=1}^m \exp(a(\mathbf{s}_i, \mathbf{h}_{j'}))}. \quad (1.78)$$

对齐分数可以有多种形式 [???]。最简单的是点积注意力：

$$a(\mathbf{s}_i, \mathbf{h}_j) = \mathbf{s}_i \mathbf{h}_j^\top. \quad (1.79)$$

scaled dot-product attention 则在点积上加入缩放因子 [?]：

$$a(\mathbf{s}_i, \mathbf{h}_j) = \frac{\mathbf{s}_i \mathbf{h}_j^\top}{\beta}, \quad (1.80)$$

其中 β 通常与向量维度有关。在 Transformer 中，常用 $\beta = \sqrt{d}$ ，这样可以避免维度

较大时点积值过大，导致 softmax 分布过于尖锐。

更一般地，注意力可以表述为 Query-Key-Value (QKV) 模型。设有一组 query $\mathbf{Q}$ 、key $\mathbf{K}$ 和 value $\mathbf{V}$ 。模型先用 query 与 key 的相似度确定权重，再对 value 加权求和：

$$\text{Att}_{\text{qkv}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right) \mathbf{V}. \quad (1.81)$$

这里 $\text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}})$ 是逐行归一化的注意力权重矩阵。每一行表示某个 query 对所有 value 位置的偏好分布。

重点提示

QKV 注意力的新概念可以用检索来理解：query 表示“当前位置想找什么信息”，key 表示“每个位置提供什么索引”，value 表示“真正被取出的内容”。难点在于不要把注意力权重误解成最终表示本身。权重只决定从哪些 value 中取信息，最终输出是 value 的加权和。缩放因子 $\sqrt{d}$ 的作用也不是改变模型结构，而是避免高维点积过大，使 softmax 不至于过早变成极端分布。

自注意力是 QKV 注意力的一种特殊情况。给定输入序列表示矩阵 $\mathbf{H} \in \mathbb{R}^{m \times d}$ ，模型用三个可学习线性变换生成 query、key 和 value：

$$\mathbf{H}^q = \mathbf{H}\mathbf{W}^q, \quad (1.82)$$

$$\mathbf{H}^k = \mathbf{H}\mathbf{W}^k, \quad (1.83)$$

$$\mathbf{H}^v = \mathbf{H}\mathbf{W}^v. \quad (1.84)$$

于是自注意力输出为

$$\begin{aligned} \mathbf{C} &= \text{Att}_{\text{self}}(\mathbf{H}) \\ &= \text{Softmax}\left(\frac{\mathbf{H}^q[\mathbf{H}^k]^\top}{\sqrt{d}}\right) \mathbf{H}^v. \end{aligned} \quad (1.85)$$

输出 $\mathbf{C}$ 与输入 $\mathbf{H}$ 形状相同，可以看作每个位置融合全局上下文之后的新表示。与 RNN 和 CNN 相比，自注意力的一个重要优势是缩短了任意两个位置之间的信息传递距离：每个位置可以直接访问序列中的任意其他位置，而不必通过递归步或卷积层逐步传播。

multi-head attention 是自注意力的进一步扩展 [?]。它不是只在一个表示空间中做注意力，而是把输入投影到多个低维子空间，并在每个子空间中独立执行注意力。

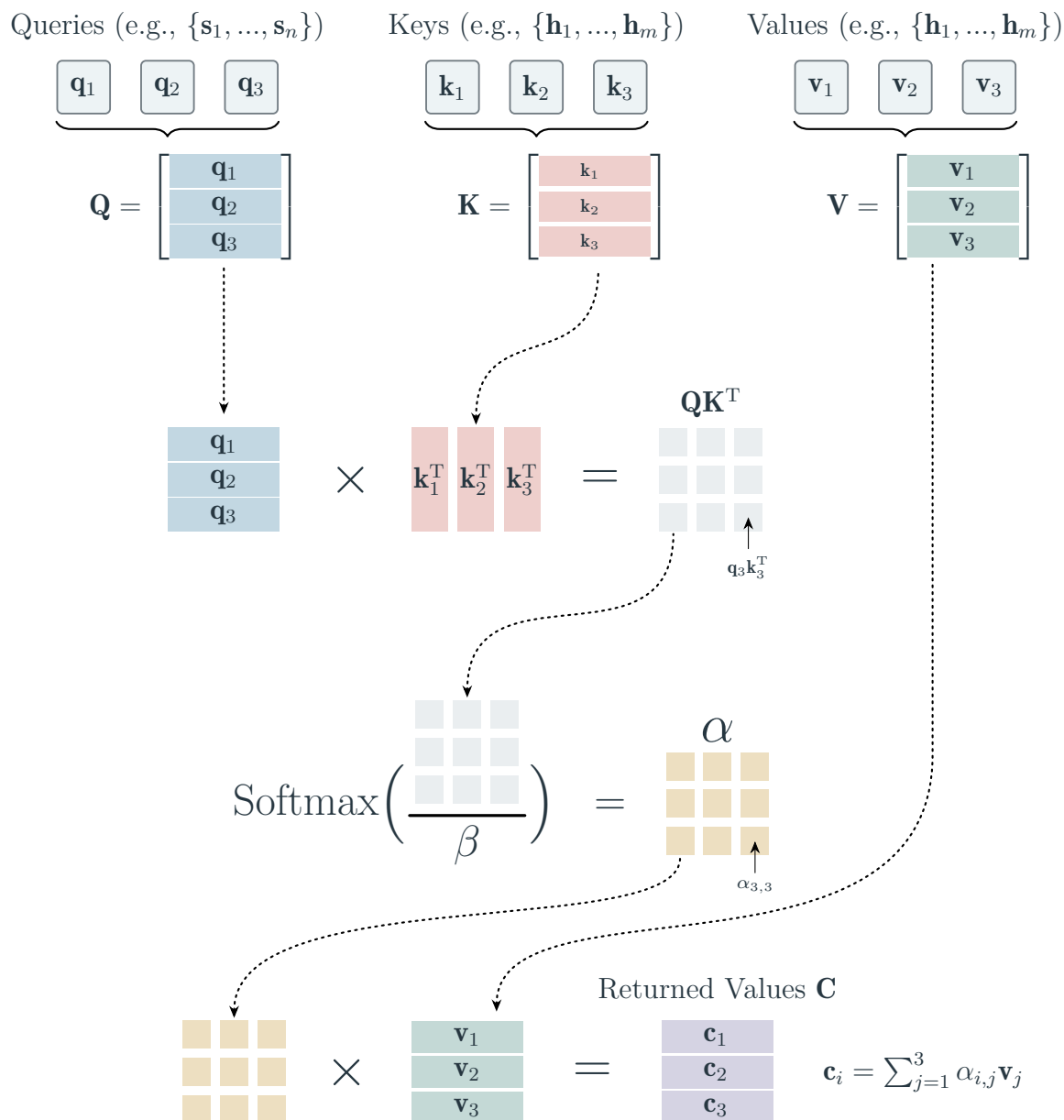


图 1.13: QKV 注意力机制示意。query 与 key 计算权重，再对 value 加权求和。

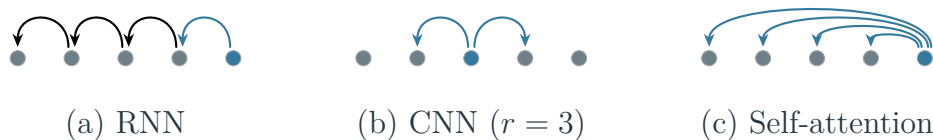


图 1.14: RNN、CNN 与自注意力中的信息流比较。自注意力可以让任意两个位置直接交互。

设共有 τ 个 head。对第 h 个 head，有

$$\mathbf{C}_h^{\text{head}} = \text{Softmax} \left(\frac{\mathbf{H}_h^q [\mathbf{H}_h^k]^\top}{\sqrt{d}} \right) \mathbf{H}_h^v, \quad (1.86)$$

$$\mathbf{H}_h^q = \mathbf{H} \mathbf{W}_h^q, \quad (1.87)$$

$$\mathbf{H}_h^k = \mathbf{H} \mathbf{W}_h^k, \quad (1.88)$$

$$\mathbf{H}_h^v = \mathbf{H} \mathbf{W}_h^v. \quad (1.89)$$

各个 head 的输出再拼接并线性变换：

$$\mathbf{C} = \text{Merge}(\mathbf{C}_1^{\text{head}}, \dots, \mathbf{C}_\tau^{\text{head}}) \mathbf{W}_c. \quad (1.90)$$

多头机制可以理解为让模型从多个角度观察同一序列：不同 head 可以关注不同位置关系、语法线索或语义关联。由于各个 head 结构相同且可以并行计算，这一机制在现代深度学习框架中实现也很高效。

Transformer 中的自注意力还需要处理位置信息。普通注意力和 FFN 本身并不天然感知词序，因此 Transformer 会把词向量与位置编码相加 [?]。对位置 j 的词 x_j ，其输入表示可写为

$$\mathbf{e}_j = \mathbf{x}_j + \text{PE}(j), \quad (1.91)$$

其中 $\mathbf{x}_j$ 是词向量， $\text{PE}(j)$ 是位置表示。原始 Transformer 使用正弦和余弦函数构造绝对位置编码：

$$\text{PE}(i, 2k) = \sin \left(i \cdot \frac{1}{10000^{2k/d}} \right), \quad (1.92)$$

$$\text{PE}(i, 2k+1) = \cos \left(i \cdot \frac{1}{10000^{2k/d}} \right). \quad (1.93)$$

这样，每个位置都能得到一个连续向量，模型可以在自注意力中区分不同 token 的先后顺序。

例如，在句子“学生阅读论文”中，当模型更新“阅读”这个位置的表示时，它可能同时关注动作发出者“学生”和动作对象“论文”。一种可能的注意力分布为

query: 阅读	学生	阅读	论文
α	0.35	0.10	0.55

于是“阅读”的新表示会更多融合“论文”的信息，也会保留一部分“学生”的信息。若换成另一个 head，它可能更关注主谓关系，从而提高“学生”的权重。多头注意力的直观作用，就是让模型从不同关系角度同时理解同一句话。

1.3.3 Encoder/Decoder 结构

Encoder/Decoder 架构最适合描述序列到序列问题 [? ? ?]。给定源端序列 $\mathbf{x}$ 和目标端序列 $\mathbf{y}$ ，我们希望学习从 $\mathbf{x}$ 到 $\mathbf{y}$ 的映射。直接学习离散序列之间的映射通常会遇到高维数据和维度灾难问题，因此一种自然做法是引入中间表示 $\mathbf{H}$ ：先把 $\mathbf{x}$ 编码为 $\mathbf{H}$ ，再由 $\mathbf{H}$ 解码出 $\mathbf{y}$ 。

形式化地，编码器把源端序列映射为表示：

$$\mathbf{H} = \text{Encode}(\mathbf{x}), \quad (1.94)$$

解码器再从该表示生成目标端序列：

$$\mathbf{y} = \text{Decode}(\mathbf{H}). \quad (1.95)$$

这个架构广泛用于现代 sequence-to-sequence 系统。解码器也可以被重新定义为概率函数：

$$\begin{aligned} \Pr(\cdot|\mathbf{H}) &= \text{Decode}(\mathbf{H}) \\ &= \text{Decode}(\text{Encode}(\mathbf{x})). \end{aligned} \quad (1.96)$$

于是给定源端序列 $\mathbf{x}$ ，目标端序列 $\mathbf{y}$ 的概率为

$$\Pr(\mathbf{y}|\mathbf{x}) = \Pr(\mathbf{y}|\mathbf{H}). \quad (1.97)$$

预测时，系统需要在候选目标序列中寻找概率最大的 $\hat{\mathbf{y}}$ 。因此，Encoder/Decoder 不仅是表示学习问题，也引出了搜索和推理问题。

神经机器翻译是 Encoder/Decoder 的典型应用。设源端词表为 V_x ，每个源端词 x_j 先被映射为词向量：

$$\mathbf{x}_j^e = \text{Embed}_s(x_j). \quad (1.98)$$

若编码器使用 RNN，则它逐步读入词向量序列，并产生状态向量：

$$\mathbf{h}_j = \text{RNN}(\mathbf{h}_{j-1}, \mathbf{x}_j^e). \quad (1.99)$$

最简单的做法是把最后一个状态 $\mathbf{h}_m$ 作为整个源端序列的压缩表示：

$$\mathbf{h}_m = \text{Encode}(x_1 \dots x_m). \quad (1.100)$$

解码器则是另一个语言模型式的网络，它在每个位置基于已经生成的目标端词和源端

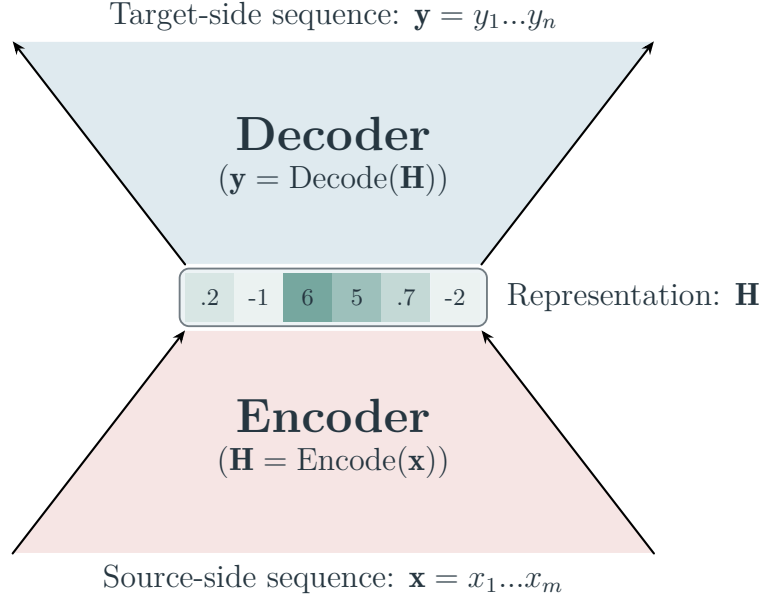


图 1.15: Encoder/Decoder 架构示意。编码器把源端序列映射为表示，解码器基于该表示生成目标端序列。

表示预测下一个词。其条件概率常写为

$$\Pr(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^n \Pr(y_i | y_0 \dots y_{i-1}, x_1 \dots x_m), \quad (1.101)$$

其中 y_0 通常是起始符号 (SOS)。

Transformer 也遵循 Encoder/Decoder 框架，但它用自注意力、前馈网络、残差连接和层归一化替代传统 RNN [? ? ?]。一个 Transformer encoder 由多个编码层堆叠而成，每个编码层包含 self-attention 子层和 FFN 子层。若输入为 $\mathbf{H}$ ，self-attention 子层可写为

$$\text{Layer}_{\text{self}}(\mathbf{H}) = \text{LNorm}(\text{Att}_{\text{self}}(\mathbf{H}) + \mathbf{H}), \quad (1.102)$$

其中加号表示残差连接， $\text{LNorm}(\cdot)$ 表示层归一化。FFN 子层形式类似：

$$\text{Layer}_{\text{ffn}}(\mathbf{H}_{\text{self}}) = \text{LNorm}(\text{FFN}(\mathbf{H}_{\text{self}}) + \mathbf{H}_{\text{self}}). \quad (1.103)$$

堆叠 L 层后， $\mathbf{H}^L$ 就是编码器输出，可被看作输入序列的上下文文化表示。

Transformer decoder 与 encoder 类似，但每个解码层多出一个 encoder-decoder attention，也称 cross-attention。若 $\mathbf{H}^L$ 是编码器最终输出， $\mathbf{S}^l$ 是第 l 层解码器输出，

则一个解码层可以抽象为

$$\mathbf{S}^l = \text{Layer}_{\text{ffn}}(\mathbf{S}_{\text{cross}}^l), \quad (1.104)$$

$$\mathbf{S}_{\text{cross}}^l = \text{Layer}_{\text{cross}}(\mathbf{H}^L, \mathbf{S}_{\text{self}}^l), \quad (1.105)$$

$$\mathbf{S}_{\text{self}}^l = \text{Layer}_{\text{self}}(\mathbf{S}^{l-1}). \quad (1.106)$$

decoder self-attention 用于建模目标端前缀内部的依赖，cross-attention 用于读取源端表示。为了保持自回归生成，decoder self-attention 通常需要 mask，使当前位置不能看到未来 token。最后一层 decoder 输出 $\mathbf{S}^L$ 经过线性变换和 softmax，得到每个目标位置上的词分布。

从架构角度看，Transformer 可以退化为多种常见预训练模型。只使用 encoder，可得到 encoder-only 架构，适合文本编码和理解任务；只使用 decoder，可得到 decoder-only 架构，适合语言建模和文本生成；同时使用 encoder 与 decoder，则适合机器翻译、摘要、问答等输入输出皆为序列的任务。

以机器翻译为例，源句 “I love NLP” 经过 encoder 后得到一组上下文化表示 $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ 。decoder 生成中文译文时，会从起始符号开始逐步预测：

步骤	已生成前缀	预测下一个词
1	⟨SOS⟩	我
2	我	喜欢
3	我喜欢	NLP
4	我喜欢 NLP	⟨EOS⟩

其中 self-attention 负责建模已经生成的目标端前缀，cross-attention 负责读取源端表示。例如生成 “喜欢” 时，decoder 应当更多关注源端的 “love”；生成 “NLP” 时，则应当关注源端最后一个 token。这个例子展示了 Encoder/Decoder 架构如何把 “理解输入” 和 “逐步生成输出” 分成两个相互配合的过程。

1.4 表示学习机制

机器学习模型不能直接处理 “商品” “文档” “图像类别” 或 “用户兴趣” 等抽象对象，而要对数值向量或张量进行计算。表示学习研究的就是如何把现实对象转换为数值表示，并让表示保留对任务有用的信息 [?]。一个好的表示不仅应当支持矩阵运算，还应使相似对象在表示空间中具有相近结构，使后续模型更容易学习分类边界、回归关系或生成规律。本节从分布式表示出发，进一步讨论特征映射和预训练，形成 “对象如何表示、表示如何学习、学到的表示如何迁移” 的完整线索。

1.4.1 分布式表示

从现实对象到数值表示

设现实对象为 s ，表示函数 ϕ 把它转换为 d 维向量：

$$\phi(s) = \mathbf{z} \in \mathbb{R}^d. \quad (1.107)$$

对象 s 可以是一种商品、一个用户、一篇文档或一张图像， $\mathbf{z}$ 则是模型实际读取的表示。表示方式决定模型能够看到哪些信息。例如，将颜色类别只编码成编号 1, 2, 3，会人为引入大小顺序；将其编码成 one-hot 向量，则只表达类别身份，不引入不存在的顺序关系。

one-hot 表示及其局限

one-hot 是表示离散对象的基本方法。设对象集合为

$$V = \{\text{猫}, \text{狗}, \text{汽车}\},$$

则三个对象可以分别表示为

$$\mathbf{x}_{\text{猫}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{x}_{\text{狗}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{x}_{\text{汽车}} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

这种表示能够准确区分对象，却没有表达对象之间的相似性。任意两个不同对象的点积都为 0，欧氏距离都为 $\sqrt{2}$ ，所以在 one-hot 空间中，“猫”和“狗”并不比“猫”和“汽车”更接近。此外，当对象数量很大时，向量维度和后续参数矩阵会随之增长。稀疏存储可以避免保存大量的零，却不能自动建立对象之间的关系。更重要的是，不同对象之间不能共享统计信息：模型从“猫”上学到的规律不会自然帮助它处理“狗”。

分布式表示解决什么问题

分布式表示用多个共享的连续特征共同描述一个对象。设共有 $|V|$ 个离散对象，one-hot 向量为 $\mathbf{x}_s \in \mathbb{R}^{|V|}$ ，嵌入矩阵为 $\mathbf{E} \in \mathbb{R}^{|V| \times d}$ ，则对象 s 的低维表示为

$$\mathbf{e}_s = \mathbf{E}^T \mathbf{x}_s \in \mathbb{R}^d, \quad d \ll |V|. \quad (1.108)$$

这个乘法等价于从嵌入矩阵中取出对象 s 对应的一行。矩阵 $\mathbf{E}$ 可以由数据学习，使经常出现在相似情境中、具有相似作用或对任务产生相似影响的对象获得相近向量。因此，分布式表示主要解决三个问题：用较低维的稠密向量替代高维稀疏编码；用距离或方向表达对象之间的关系；通过共享特征把一个对象上学到的规律迁移到相似对

象。

仍以“猫”“狗”“汽车”为例，模型可以学习到

对象	e_1	e_2
猫	0.81	0.62
狗	0.76	0.68
汽车	-0.40	0.91

这样的低维向量。虽然单个维度未必有明确的人类解释，但“猫”和“狗”的向量更接近，说明多个维度共同编码了二者的相似性。同样的思想也适用于用户表示、商品表示和图节点表示，并不局限于词向量。

在文本处理中，词向量是一种典型的分布式表示。如果两个词经常出现在相似上下文中，训练过程通常会使它们的向量接近。但这种几何关系不是天然存在的，而是由训练数据和学习目标共同决定的 [? ? ? ?]。静态词向量为一个词分配固定向量；上下文文化模型则根据具体输入生成表示，例如英文单词 *table* 在“餐桌”和“数据表格”两个语境中可以得到不同向量 [?]。

重点提示

“分布式”不是指概率分布，而是指一个对象的信息分散在多个向量维度中，同时每个维度也被许多对象共享。表示是否真正有用，最终要由下游任务上的泛化表现检验。

1.4.2 特征映射与表示空间

上一节比较了 one-hot 向量和低维嵌入，它们其实是两种不同的特征映射。特征映射要回答的核心问题是：模型应该用哪些数值描述输入，才能保留与预测目标有关的信息，并减少无关变化带来的干扰。

原始输入与特征空间

原始输入是数据被模型处理之前的观测形式。例如，一条房屋记录可能由“城市、面积、建造年份”等混合类型字段组成，一篇文档是字符或词的序列，一张图像则是像素数组。为了送入模型，需要用映射 ϕ 把原始对象 s 转换为特征向量：

$$s \xrightarrow{\phi} \mathbf{z} = \phi(s) \in \mathbb{R}^d. \quad (1.109)$$

映射后的 $\mathbb{R}^d$ 称为特征空间。所谓“更适合任务的空间”，并不是任务预先规定了一个神秘空间，而是希望特征空间中的坐标、距离或方向与预测目标更相关。例如，在房价预测中，可以把城市做 one-hot 编码，把面积标准化，并将建造年份转换为房龄；得到的数值向量比原始混合字段更适合线性回归。在图像分类中，原始像素空间对平

移和光照变化很敏感，而神经网络中间层可以把边缘、纹理和形状组合成更稳定的特征。

好的特征映射通常承担三种作用。第一，把非数值对象转换为模型可计算的向量或张量；第二，突出与任务有关的信息并抑制噪声；第三，通过压缩、共享或不变性，使有限数据上的学习更容易泛化。映射可能由人工规则给定，也可能从数据中估计，还可以与预测模型一起端到端学习。

从计数到任务相关的权重

文本向量化可以直观展示固定特征映射的作用。词袋模型先建立词表，再把文档映射成词频向量。它完成了“文档到数值向量”的转换，但把所有词的出现次数同等看待。TF-IDF在词频基础上加入逆文档频率，使在少数文档中出现的词获得更高权重 [?]：

$$\text{tfidf}(a, d, D) = \text{tf}(a, d) \cdot \text{idf}(a, D), \quad (1.110)$$

其中一种常见的 IDF 形式为

$$\text{idf}(a, D) = \log \frac{|D|}{\text{df}(a, D)}. \quad (1.111)$$

其中， $\text{df}(a, D)$ 表示数据集 D 中包含词 a 的文档数。假设一个数据集包含 1000 篇文档，“模型”出现在 900 篇中，“过拟合”只出现在 20 篇中。如果二者在某篇文档里都出现 3 次，它们的词频相同，但“过拟合”的 IDF 更大。于是 TF-IDF 会降低常见词的影响，提高更有区分度的词的权重。这个例子说明，特征映射不仅把输入变成数字，还把“哪些信息可能对任务更有用”编码进表示。

压缩与可学习映射

固定映射得到的向量可能非常高维。例如，词袋向量的维度等于词表大小，图像展开后的像素向量也可能包含数十万个分量。降维方法尝试在减少维度的同时保留主要结构。PCA 是一种代表性线性方法 [?]。设数据矩阵为 $\mathbf{M} \in \mathbb{R}^{N \times d}$ ，PCA 选择 r 个主要方向组成矩阵 $\mathbf{C} \in \mathbb{R}^{d \times r}$ ，得到

$$\mathbf{N} = \mathbf{MC}, \quad r < d. \quad (1.112)$$

这里每个样本从 d 维变为 r 维， $\mathbf{C}$ 选择的是数据方差较大的方向。图 ?? 展示了二维数据投影到一维方向的过程。PCA 保留的是总体变化较大的结构，但这些结构未必恰好是具体预测任务最需要的信息，因此降维结果仍要通过任务性能评估。

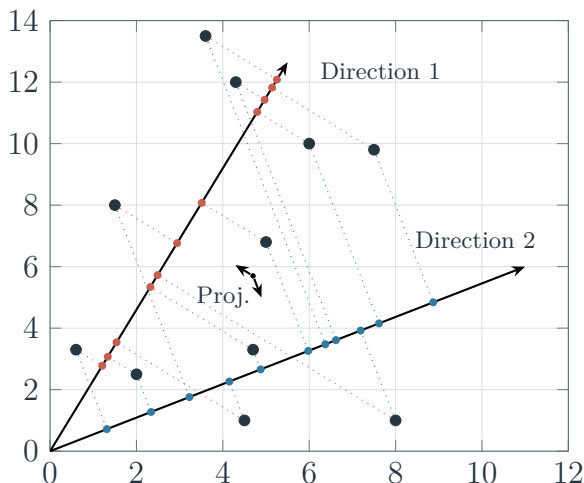


图 1.16: PCA 将二维数据投影到一维主方向的示意。不同投影方向会保留不同程度的方差信息。

神经网络进一步把映射本身写成带参数的函数 ϕ_θ ，并通过数据学习参数：

$$\mathbf{z} = \phi_\theta(s). \quad (1.113)$$

参数 θ 可以只根据输入数据学习，也可以和下游预测器一起依据任务损失更新。以文本处理为例，Word2Vec 不直接规定每个词向量应该取什么值，而是设计一个局部预测任务 [?]。CBOW 根据上下文词预测中心词；skip-gram 根据中心词预测上下文词。以 CBOW 为例，模型先聚合上下文词向量得到 $\mathbf{h}$ ，再预测中心词 w_i ：

$$\Pr(w_i | w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}) = \text{Softmax}(\mathbf{W}\mathbf{h} + \mathbf{b})_{w_i}. \quad (1.114)$$

为了提高中心词预测的准确率，模型必须调整嵌入矩阵，使具有相似上下文的词产生相似表示。图 ?? 对比了这两种训练方向。这里真正重要的不是记住某个词向量，而是理解“用一个可优化的辅助任务学习特征映射”。

从固定规则到可学习映射，模型对人工设计特征的依赖逐渐减少。词袋和 TF-IDF 由人定义坐标及权重，PCA 从数据中估计线性投影，神经网络则能够学习多层非线性映射。无论使用哪一种方法，都应明确三个问题：原始输入是什么，映射后保留了什么信息，以及这种信息是否有助于目标任务。当单个任务的数据不足以学习稳定的复杂映射时，就需要先在更大规模数据上学习表示，再把它迁移到目标任务，这正是预训练的出发点。

1.4.3 预训练与表示迁移

上一节说明，特征映射可以由神经网络从数据中学习。但深层模型通常参数很多，而单个任务的标注数据往往有限，直接从随机参数开始训练容易出现优化困难或过拟合。预训练先利用规模更大、来源更广的数据学习一个通用参数起点，再把学到的表

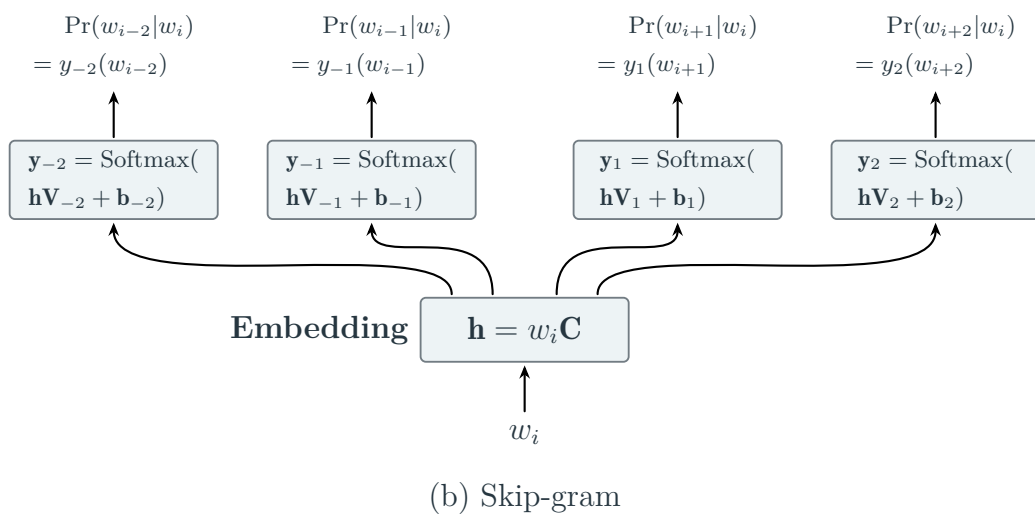
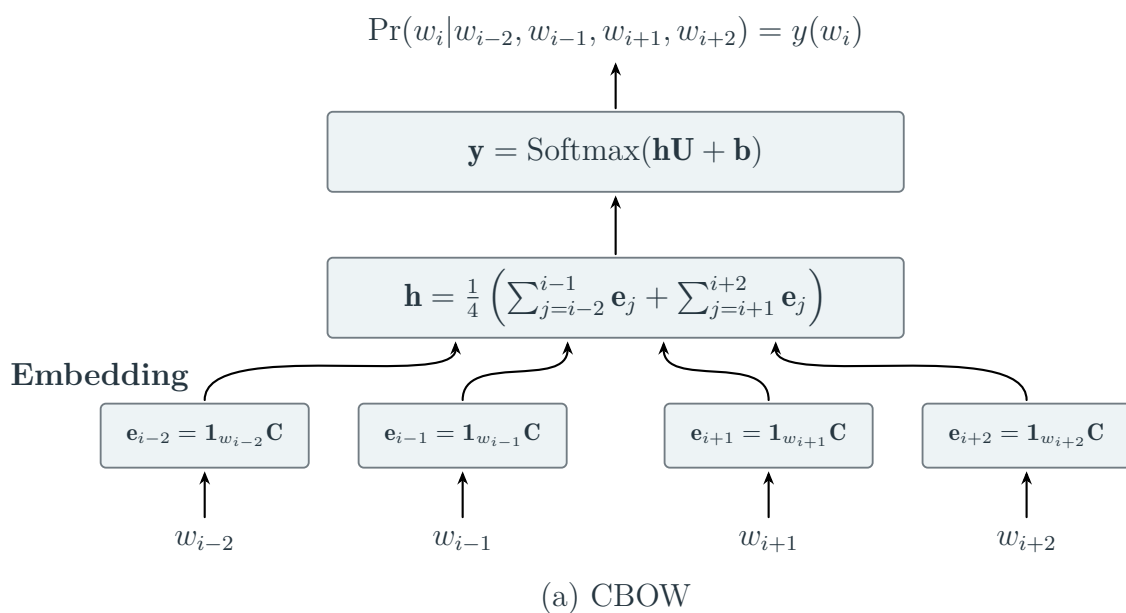


图 1.17: Word2Vec 中 CBOW 与 skip-gram 的训练结构示意图。二者分别由上下文预测中心词、由中心词预测上下文。

示和参数迁移到目标任务 [? ? ?]。

为什么需要预训练

设预训练数据为 $\mathcal{D}_{\text{pre}}$ ，目标任务数据为 $\mathcal{D}_{\text{task}}$ 。预训练先求得

$$\theta_{\text{pre}} = \arg \min_{\theta} \mathcal{L}_{\text{pre}}(\theta; \mathcal{D}_{\text{pre}}), \quad (1.115)$$

再以 θ_{pre} 为起点学习目标任务：

$$\theta_{\text{task}} \approx \arg \min_{\theta} \mathcal{L}_{\text{task}}(\theta; \mathcal{D}_{\text{task}}), \quad \theta^{(0)} = \theta_{\text{pre}}. \quad (1.116)$$

与随机初始化相比，预训练参数已经编码了数据中的常见结构。目标任务不必从头学习所有低层规律，而可以把有限标注集中在任务特有的决策上。因此，预训练通常能够提高数据利用效率、改善优化起点，并在多个相关任务之间复用一次大规模训练的成本。图 ?? 以神经语言模型为例，展示了先学习表示、再迁移到新任务的过程。

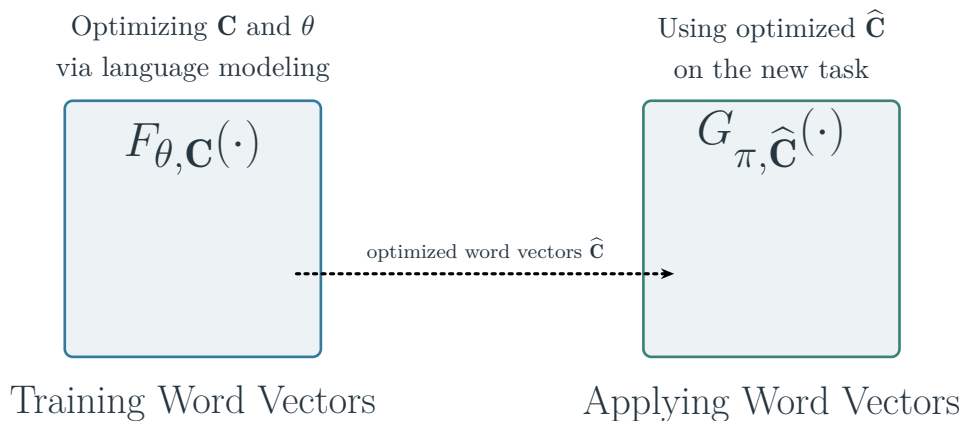


图 1.18: 通过神经语言模型预训练词向量并迁移到新任务的示意。

预训练不只用于语言模型

预训练是一种通用的迁移学习机制，并不属于某一种数据类型或模型结构。在计算机视觉中，卷积网络或视觉 Transformer 可以先通过大规模图像分类、对比学习或遮蔽图像重建学习视觉特征，再迁移到目标检测、医学图像分类等任务。在语音处理中，模型可以先预测被遮蔽的声学片段，再用少量转写数据训练语音识别系统。在图和分子学习中，也可以通过恢复被遮蔽的节点、边或局部结构预训练图网络，再进行分子性质预测 [? ? ?]。

按照监督信号来源，预训练可以采用有监督、自监督或无监督目标。有监督预训练利用来源任务的人工标签；自监督预训练从数据本身构造输入和目标，例如遮蔽后恢复、预测下一个元素或对比不同数据变换；传统无监督预训练则常通过重构输入或

生成数据学习结构。它们的共同点不是具体损失函数，而是先学习可复用规律，再适配目标任务。

为什么大语言模型强调预训练

大语言模型包含大量参数，单个下游任务的数据无法充分约束这些参数。与此同时，文本序列天然提供了大规模自监督信号：给定前文预测下一个 token，或遮蔽部分 token 后恢复原内容，都不需要逐条人工标注。模型可以在大量文本上反复执行这些预测任务，从中学习词法、句法、篇章结构以及数据中反复出现的知识模式。

更重要的是，预训练把昂贵的通用学习集中在一个基础模型中。完成预训练后，同一组参数可以通过微调、特征提取或提示适配多个任务。因此，现代大语言模型强调预训练，并不只是因为训练数据更多，而是因为预训练把大规模自监督学习与多任务迁移连接起来 [? ? ?]。不过，预训练学到的是训练数据中的统计规律，它并不保证输出始终真实、无偏或适合所有目标任务，仍需要下游数据、评估和对齐过程。

重点提示

预训练的核心是“先学习可复用结构，再适配目标任务”。它既可以发生在文本、图像、语音和图数据上，也可以使用有监督或自监督目标。模型结构、预训练目标和下游用途之间存在常见搭配，但不是不可改变的一一对应关系。

语言模型中的三类预训练结构

在文本领域，Transformer常见的三种结构分别对应三类训练方式。下面的对应关系用于说明它们各自擅长利用什么信息，而不是限制一种结构只能使用一种目标。

decoder-only 模型通常采用自回归目标，根据左侧上下文预测下一个 token [?]。对序列 $\mathbf{x} = (x_0, \dots, x_m)$ ，负对数似然损失为

$$\mathcal{L}_{\text{AR}}(\boldsymbol{\theta}) = - \sum_{i=0}^{m-1} \log \Pr_{\boldsymbol{\theta}}(x_{i+1} | x_0, \dots, x_i). \quad (1.117)$$

模型在每个位置都获得一次训练信号，因此一段未标注文本可以自动产生多个输入目标对。这类目标直接训练逐步生成能力，是许多生成式大语言模型的基础。

encoder-only 模型常采用 masked language modeling。训练过程随机遮蔽部分 token，得到 $\bar{\mathbf{x}}$ ，再利用双向上下文恢复原 token。设遮蔽位置集合为 $\mathcal{A}(\mathbf{x})$ ，其损失为

$$\mathcal{L}_{\text{MLM}}(\boldsymbol{\theta}) = - \sum_{i \in \mathcal{A}(\mathbf{x})} \log \Pr_{\boldsymbol{\theta}}(x_i | \bar{\mathbf{x}}). \quad (1.118)$$

由于编码器能够同时利用左右上下文，这类表示常用于分类、检索和序列标注。BERT 是这一结构的经典实例 [?]。训练完成后，可以移除用于 token 预测的输出层，保留

编码器作为下游任务的特征映射。图 ?? 对比了自回归模型与遮蔽语言模型在训练时可见的上下文范围。

encoder-decoder 模型可以把被破坏的输入映射回完整序列。设 $\mathcal{C}(\mathbf{x})$ 表示遮蔽、删除或打乱部分输入的破坏函数，去噪目标可以写为

$$\mathcal{L}_{\text{DAE}}(\boldsymbol{\theta}) = -\log \Pr_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathcal{C}(\mathbf{x})). \quad (1.119)$$

这种训练要求编码器理解受损输入，再由解码器生成目标序列，因而适合摘要、翻译等输入输出均为序列的任务。BART 的去噪训练和 T5 的 span corruption 都可以从这一框架理解 [? ?]。

从预训练模型迁移到具体任务

预训练完成后，有三种常见使用方式。第一种是特征提取：冻结预训练参数，只训练一个较小的任务模型。第二种是 fine-tuning：以预训练参数为起点，用目标任务数据更新部分或全部参数。第三种是 prompting：把任务描述和示例组织成模型输入，在不更新参数或只更新少量参数的情况下引导生成模型完成任务 [? ?]。

以分类任务为例，预训练编码器先把输入映射为表示 $\mathbf{h}$ ，再由分类器输出类别分布：

$$\mathbf{h} = \text{Encode}_{\boldsymbol{\theta}_{\text{pre}}}(\mathbf{x}), \quad (1.120)$$

$$\mathbf{p}(\cdot \mid \mathbf{x}) = \text{Softmax}(\mathbf{W}\mathbf{h} + \mathbf{b}). \quad (1.121)$$

若冻结编码器，只更新 $\mathbf{W}$ 和 $\mathbf{b}$ ，就是特征提取；若编码器参数也参与更新，就是 fine-tuning。图 ?? 展示了同一个预训练编码器如何服务多个下游任务。

以句子“模型学习数据表示”为例，自回归预训练可以根据前缀预测下一个 token：

模型学习 → 数据.

遮蔽语言模型可以恢复被遮蔽内容：

模型 [MASK] 数据表示 → 学习.

去噪 encoder-decoder 模型则可以根据受损输入重建完整序列：

模型 [MASK] 表示 → 模型学习数据表示.

这些训练样本都由原始数据自动构造，不依赖逐条人工标注。它们先迫使模型学习可复用的表示，再由目标任务决定哪些表示真正有用。这条“预训练表示、迁移适配、任务评估”的主线，也把本节与前面讨论的特征映射、优化和泛化连接起来。

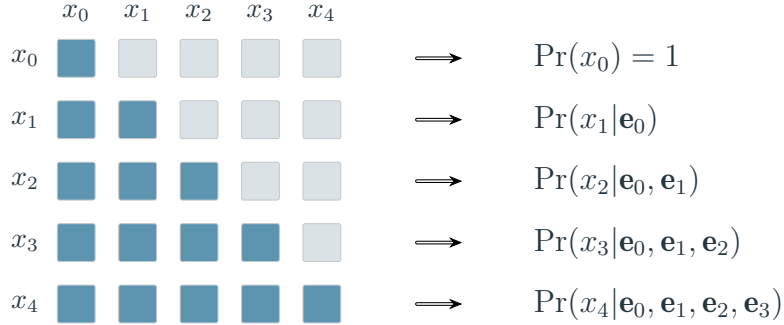
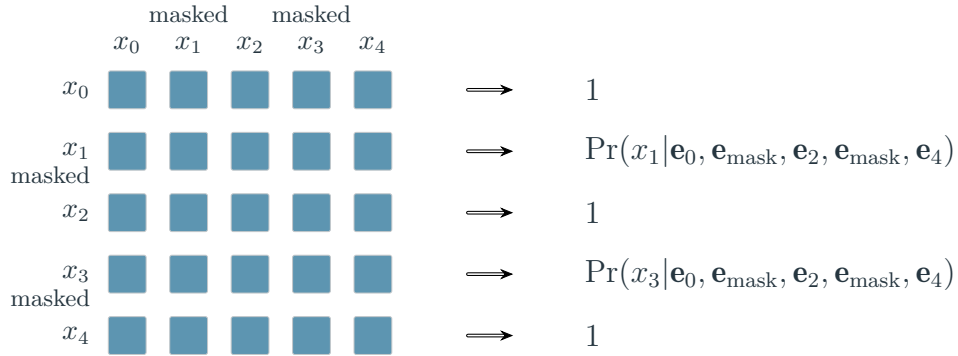
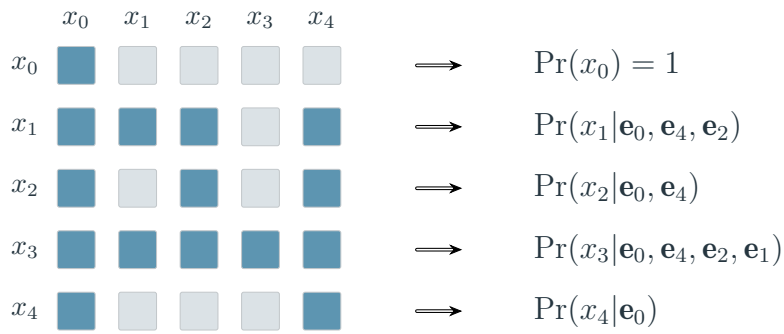
(a) Causal Language Modeling (order: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$)(b) Masked Language Modeling (order: $x_0, [\text{MASK}], x_2, [\text{MASK}], x_4 \rightarrow x_1, x_3$)(c) Permuted Language Modeling (order: $x_0 \rightarrow x_4 \rightarrow x_2 \rightarrow x_1 \rightarrow x_3$)

图 1.19: 不同语言模型中的遮蔽机制示意。遮蔽方式决定模型在预训练时能够看到哪些上下文。

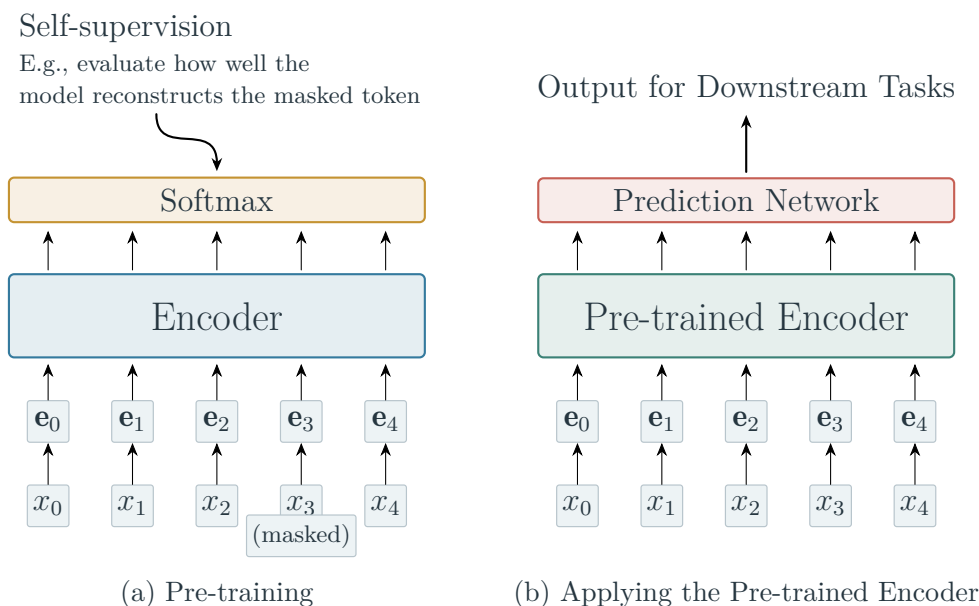


图 1.20: 预训练编码器及其下游应用示意。预训练得到的编码器可以迁移到分类、标注等任务中。

1.5 本章小节

本章围绕机器学习数学基础与表示学习前置概念，建立了一条从“数据对象”到“可训练模型”的完整线索。首先，我们从向量、矩阵、点积、范数和矩阵乘法出发，说明机器学习为什么需要线性代数语言。向量可以表示一个样本、一篇文档、一个词或一个隐藏状态；矩阵可以表示一批样本、一个线性变换或神经网络中的参数。许多看似复杂的模型计算，本质上都是这些线性对象的组合。理解向量方向、矩阵形状和乘法规则，是后续阅读神经网络公式的基础。

在计算图与矩阵微积分部分，我们进一步把神经网络看作由简单运算复合而成的函数。前向传播负责逐层计算中间表示和输出，反向传播则利用链式法则把损失对参数的梯度传回每一层。这里的核心思想不是记住某一个复杂梯度公式，而是理解“局部计算”和“局部求导”：每个节点只处理自己的输入、输出和局部导数，整个网络的训练由这些局部规则连接起来。图 ?? 展示了这一过程如何在同一张计算图上完成。

优化理论为模型训练提供了目标和方法。训练模型通常可以写成最小化经验风险的问题，而正则化项进一步把模型复杂度纳入目标函数。凸优化帮助我们理解局部最优、全局最优和梯度方向等基本概念；梯度下降则给出了参数更新的实际算法。需要注意的是，现代深度网络大多不是凸优化问题，因此训练过程不仅要看损失是否下降，还要结合学习率、批量大小、验证集表现和正则化策略综合判断。

机器学习机制部分从学习范式、建模、优化和评估四个角度展开。监督学习依赖人工标注，无监督学习挖掘数据内部结构，自监督学习则从输入本身构造监督信号。分类任务展示了建模、学习和预测三类问题如何结合：我们先把对象表示为特征向量，再定义打分函数或概率模型，然后通过训练数据估计参数，最后在新样本上输出

预测。模型评估部分强调，训练误差并不能直接代表泛化能力；验证集和测试集的划分，是为了估计模型在未见数据上的表现，避免模型只是在训练集上“背答案”。

深度神经网络部分展示了从线性模型到多层感知机、自注意力和编码器-解码器结构的过渡。多层感知机通过线性变换和非线性激活函数组合，获得比单层线性模型更强的表示能力。自注意力机制进一步突破固定窗口或固定长度向量的限制，使模型能够根据当前位置动态选择序列中相关的信息。QKV 形式把注意力解释为一种可微的检索机制：query 决定要找什么，key 决定匹配程度，value 提供被聚合的信息。编码器-解码器结构则把序列编码与序列生成连接起来，为机器翻译、摘要生成和大语言模型等任务奠定结构基础。

最后，表示学习部分把前面的数学和模型机制应用到不同类型的数据对象上。one-hot、TF-IDF、PCA 和神经网络表示分别展示了离散编码、统计加权、线性压缩和可学习映射。文本只是其中一个直观例子；同样的思想也适用于图像、语音、商品、用户和图结构数据。预训练进一步说明，复杂表示不必在每个任务上从零学习，而可以先从大规模数据中获得可复用结构，再通过特征提取、微调或提示迁移到具体任务。大语言模型中的自回归预测、遮蔽恢复和去噪重构，是这一通用机制在文本领域的三种代表性实现。

从后续课程的角度看，本章提供的是一组基础工具箱。后面讨论预训练、微调、提示学习、偏好对齐和推理增强时，仍会反复回到本章中的概念：输入如何表示，目标函数如何设计，梯度如何更新参数，模型复杂度如何影响泛化，预训练表示如何迁移到下游任务。掌握这些基础之后，读者在面对更复杂的大模型公式和系统结构时，就能把它们拆解为熟悉的组成部分，而不是把每个新模型都看作完全陌生的技术。

课后练习

1. 设一个数据集包含 m 个样本，每个样本有 n 个特征。请写出该数据集的矩阵表示，并说明矩阵的行和列分别对应什么含义。
2. 给定向量 $\mathbf{a} = [1, 2, 0, 3]$ 和 $\mathbf{b} = [0, 1, 4, 3]$ ，计算二者的点积，并解释该点积在词袋表示中可以如何理解。
3. 判断下列说法是否正确，并简要说明理由：训练误差越低，模型在测试集上的表现一定越好。
4. 设训练目标为

$$J(\boldsymbol{\theta}) = \hat{\mathcal{L}}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2.$$

请说明其中两项分别表示什么， λ 增大时模型训练会受到怎样的影响。

5. 用自己的话解释前向传播和反向传播的区别。为什么说反向传播本质上是链式法则在计算图上的应用？

6. 在文本分类任务中，比较词袋表示、TF-IDF 表示和词向量表示的差异。它们分别保留了哪些信息，又可能丢失哪些信息？
7. 请说明监督学习、无监督学习和自监督学习的监督信号分别来自哪里，并各举一个 NLP 任务例子。
8. 对于 QKV 注意力机制，query、key 和 value 分别承担什么角色？为什么注意力输出不是注意力权重本身，而是 value 的加权和？
9. 比较 decoder-only、encoder-only 和 encoder-decoder 三类预训练模型的训练目标。它们分别更适合哪些类型的下游任务？
10. 结合本章内容，设计一个简单的中文新闻分类系统流程。从数据表示、模型选择、训练目标、正则化和评估方式五个方面进行说明。