

高级机器学习

教材合订版

单位： 东北大学

日期： 2026年 9 月 5 日

目录

第一章 机器学习数学基础与表示学习	1
1.1 机器学习的数学基础	2
1.1.1 张量代数与线性对象	2
1.1.2 高级矩阵微积分与计算图	5
1.1.3 凸优化理论与训练目标	9
1.1.4 高维数据的表示与计算	12
1.2 机器学习机制与优化范式	14
1.2.1 监督学习、无监督学习与自监督学习	14
1.2.2 机器学习建模机制	17
1.2.3 优化算法与训练过程	22
1.2.4 模型评估与模型选择	25
1.2.5 正则化机制	29
1.3 深度神经网络基础	32
1.3.1 多层感知机机制	32
1.3.2 自注意力机制	34
1.3.3 Encoder/Decoder 结构	38
1.4 表示学习机制	40
1.4.1 分布式表示	41
1.4.2 特征映射与表示空间	42
1.4.3 预训练与表示迁移	44
1.5 本章小节	50
第二章 强化学习基础	53
2.1 强化学习建模方法	54
2.1.1 马尔可夫决策过程	55
2.1.2 关键元素介绍	56
2.1.3 优化目标	57
2.2 强化学习的基本优化算法	57
2.2.1 基于价值函数的方法	58
2.2.2 基于策略优化的方法	60

2.2.3	Actor-Critic 框架	63
2.3	强化学习的进阶优化算法	67
2.3.1	近端策略优化方法	67
2.3.2	离线强化学习方法	71
2.4	强化学习的应用设计方法	72
2.4.1	任务边界与转移定义	72
2.4.2	状态表示与动作空间	73
2.4.3	奖励与约束	73
2.4.4	训练与评估过程设计	75
2.5	强化学习方法的挑战	76
2.5.1	面向大规模决策的挑战	76
2.5.2	强化学习训练的不稳定性和泛化性问题	79
2.5.3	强化学习方法的奖励构建问题	82
2.6	本章小结	84
第三章	大模型预训练与微调方法	87
3.1	大模型预训练技术	88
3.1.1	语言建模方法	88
3.1.2	大语言模型架构	89
3.1.3	大语言模型预训练方法	91
3.1.4	缩放定律	93
3.2	提示学习方法	96
3.2.1	Zero/One/Few-Shot 提示方法	96
3.2.2	进阶提示方法	99
3.2.3	学习提示	104
3.3	大模型微调技术	106
3.3.1	微调概述	106
3.3.2	有监督微调方法	109
3.3.3	高效微调方法	112
3.4	大模型偏好对齐技术	116
3.4.1	对齐概述	116
3.4.2	奖励模型构建方法	119
3.4.3	基于 PPO 的强化学习人类反馈训练	122
3.4.4	直接偏好优化	124

3.4.5	偏好数据自动生成	126
3.5	大语言模型推理增强方法	127
3.5.1	推理概述	127
3.5.2	测试时缩放	128
3.5.3	基于可校验奖励的强化学习方法	131
3.5.4	从推理大模型到智能体	134
3.6	本章小结	137
第四章	深度学习中的连续动力学机制	139
4.1	连续动力学的数学基础	139
4.1.1	从离散更新到连续演化	140
4.1.2	常微分方程	141
4.1.3	常微分方程的求解：积分表示与数值方法	145
4.2	表示空间中的连续动力学	151
4.2.1	Transformer 的离散动力学解释	152
4.2.2	数值积分视角下的网络结构设计	153
4.2.3	表示动力学的稳定性与刚性	156
4.2.4	从离散深度到神经常微分方程	158
4.2.5	神经常微分方程的训练与伴随灵敏度法	161
4.3	确定性概率动力学	165
4.3.1	从样本轨迹到概率输运	166
4.3.2	流映射与连续性方程	168
4.3.3	连续归一化流	169
4.3.4	Flow Matching	172
4.3.5	概率路径设计	174
4.4	随机概率动力学	176
4.4.1	随机微分方程	177
4.4.2	正向扩散过程	178
4.4.3	反向 SDE 与分数函数	180
4.4.4	去噪分数匹配	182
4.4.5	概率流 ODE	183
4.4.6	数值采样与高效生成	185
4.5	离散状态上的连续时间动力学	190
4.5.1	令牌序列生成	190

4.5.2	连续空间中的扩散：嵌入空间扩散	193
4.5.3	词元空间中的扩散：离散扩散	198
4.5.4	离散扩散的评注	207
4.5.5	本章小结	215
第五章	多模态机器学习方法与模型	216
5.1	多模态表示学习与跨模态对齐	217
5.1.1	单模态表示学习：从原始信号到向量	217
5.1.2	自监督学习与预训练表示	221
5.1.3	跨模态对齐	224
5.2	多模态基础模型结构与训练方法	227
5.2.1	多模态基础模型的通用结构	227
5.2.2	视觉语言模型：图像理解中的多模态建模	230
5.2.3	语音语言模型：语音理解中的序列建模	232
5.2.4	多模态指令微调与对齐	235
5.2.5	图像—语音—文本联合建模	236
5.3	条件生成建模与多模态内容生成	236
5.3.1	条件生成模型基础	237
5.3.2	文本生成图像	238
5.3.3	图像编辑：约束条件下的生成	240
5.3.4	语音生成：连续信号的条件生成	241
5.3.5	音频生成：声音事件与声景的条件生成	242
5.3.6	跨模态交互生成	244
5.4	本章小结	245

第一章 机器学习数学基础与表示学习

作者：李天园、杨曦涵、肖桐

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 (1.1) 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 1.1 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。

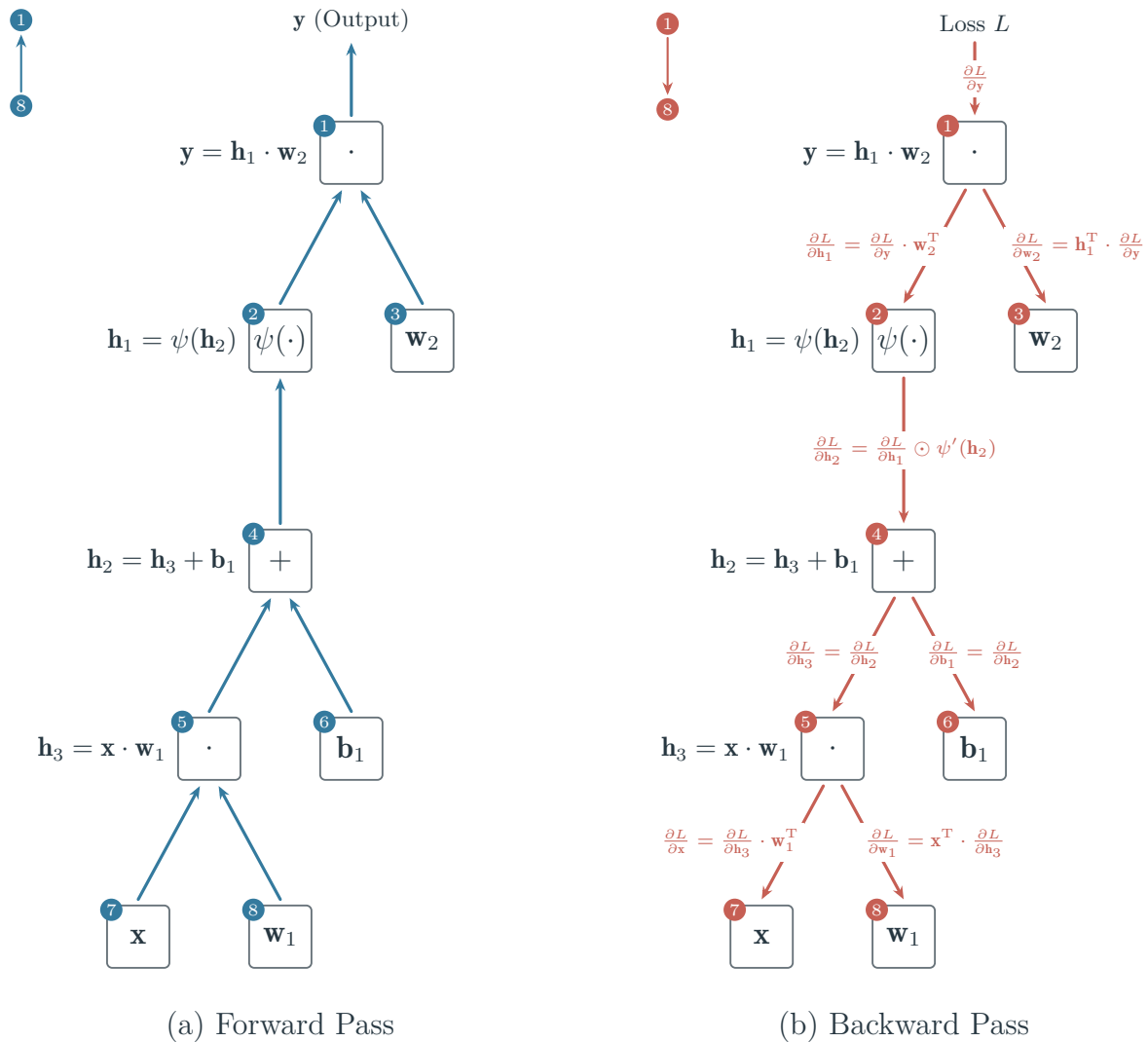


图 1.1: 前向传播与反向传播示例。前向传播计算中间变量和输出，反向传播沿计算图反方向逐层计算梯度。

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

$$\mathbf{p}_{\boldsymbol{\theta}}(\mathbf{x}) = [p_{\boldsymbol{\theta},1}(\mathbf{x}) \quad p_{\boldsymbol{\theta},2}(\mathbf{x}) \quad \cdots \quad p_{\boldsymbol{\theta},C}(\mathbf{x})],$$

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

$$\mathbf{y} = [y_1 \quad y_2 \quad \cdots \quad y_C],$$

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 1.2 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

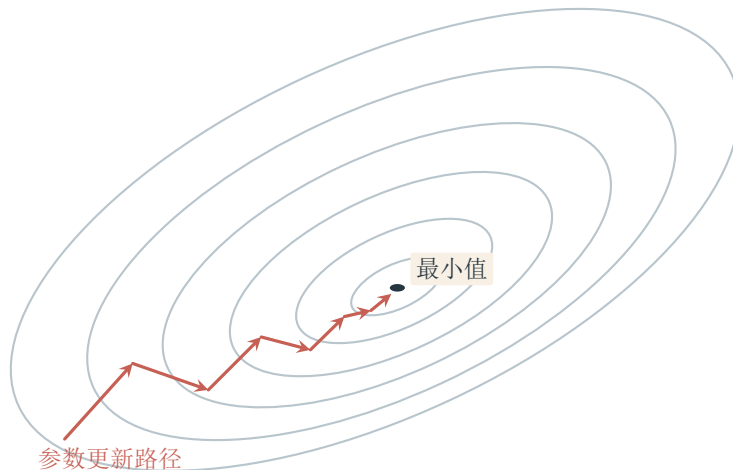


图 1.2: 梯度下降示意图。参数沿负梯度方向逐步移动，使目标函数值不断降低。

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。

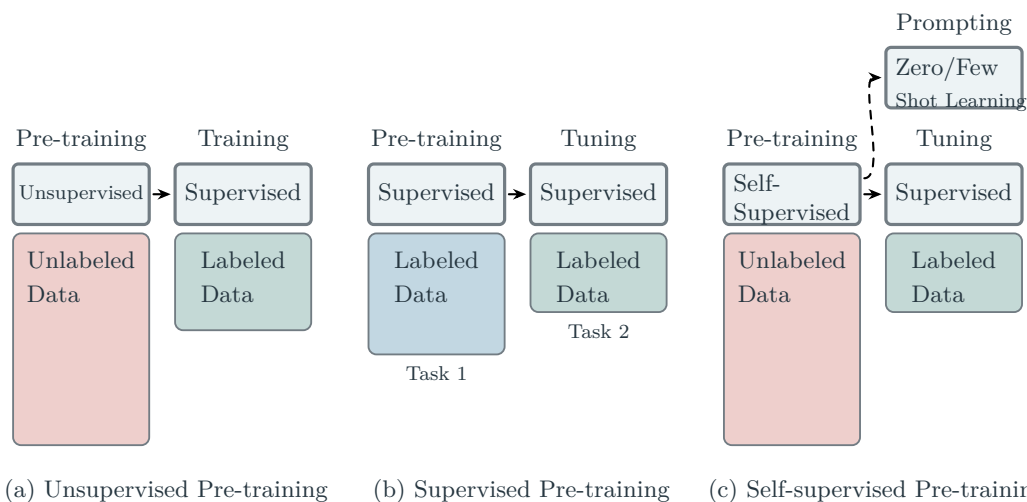


图 1.4: 无监督、有监督与自监督预训练范式示意。不同范式的核心差别在于训练信号的来源。

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 1.5 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。

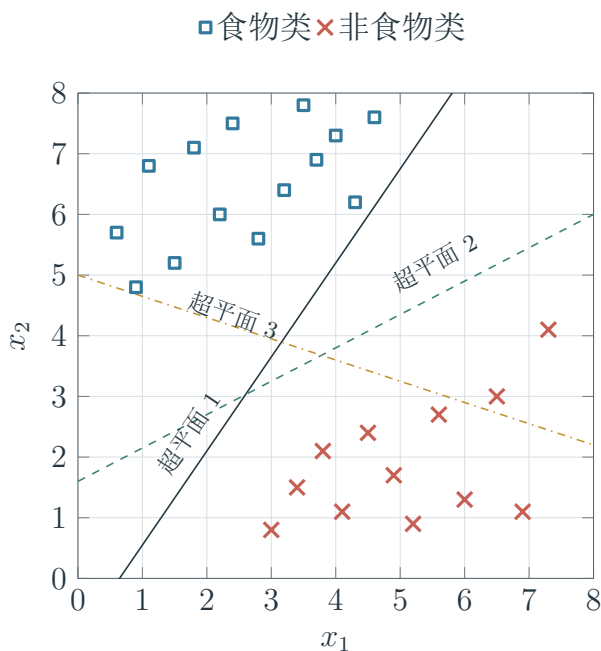


图 1.5: 二维特征空间中的线性决策边界。不同超平面可以具有相同的训练误差，却未必具有相同的泛化性能。

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

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

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 1.6 展示了学习率过小、合适和过大时的典型优化路径。

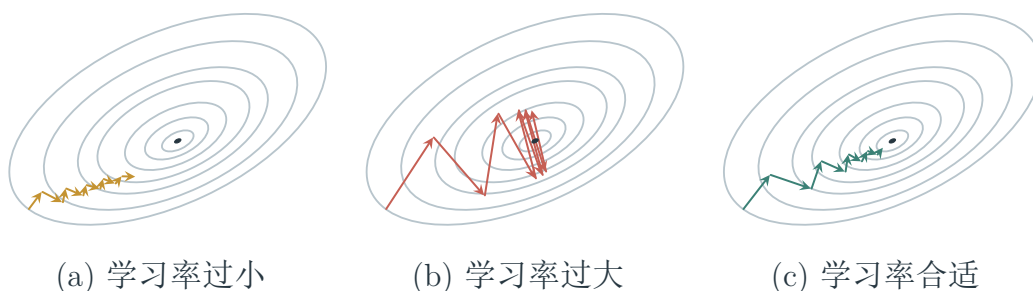


图 1.6: 不同学习率下的优化路径示意。学习率过小收敛慢, 过大则容易在最优点附近震荡。

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 1.7 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。

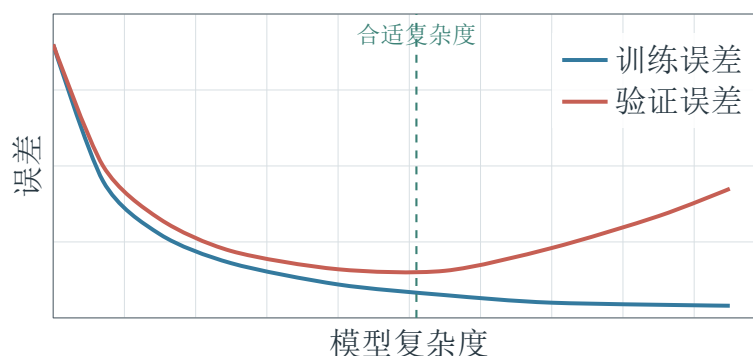


图 1.7: 训练误差与验证误差随模型复杂度变化的示意。训练误差下降并不必然带来更好的泛化性能。

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 1.8 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。

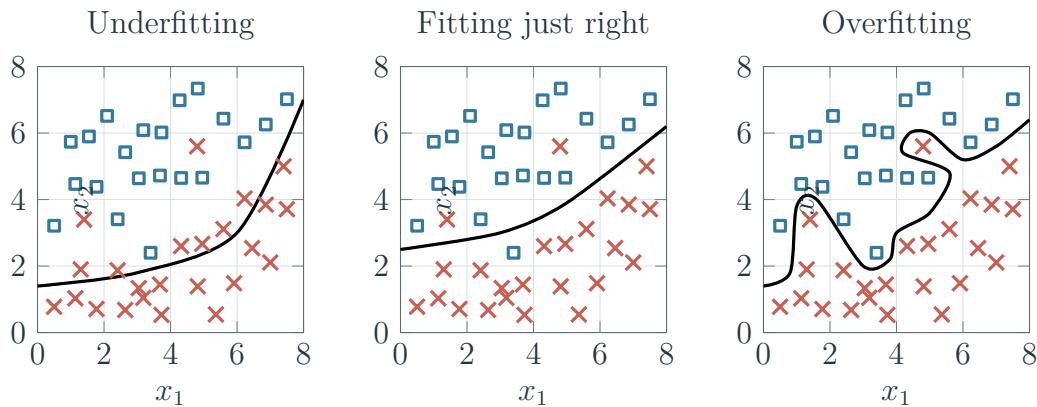


图 1.8: 欠拟合、合适拟合与过拟合示意。模型复杂度过低或过高都会影响泛化能力。

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 1.9 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。

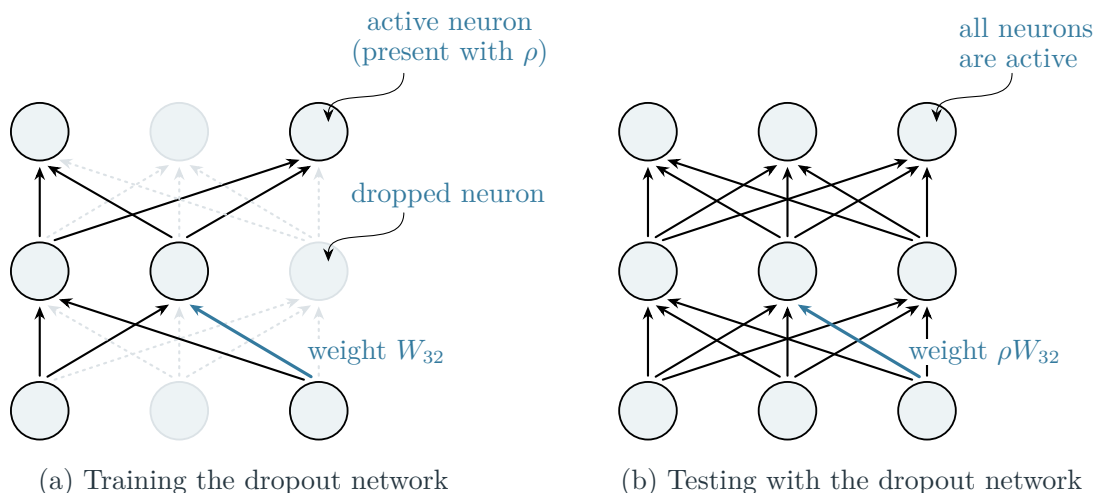


图 1.9: Dropout 的训练与测试过程。训练阶段随机关闭部分神经元，测试阶段使用完整网络。

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。

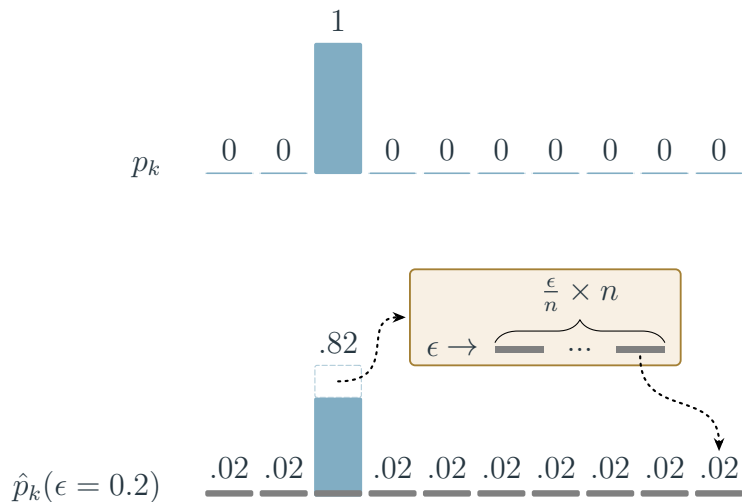


图 1.10: 输出概率平滑示意。平滑会把部分概率质量分配给非标准类别，降低过度自信。

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：

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

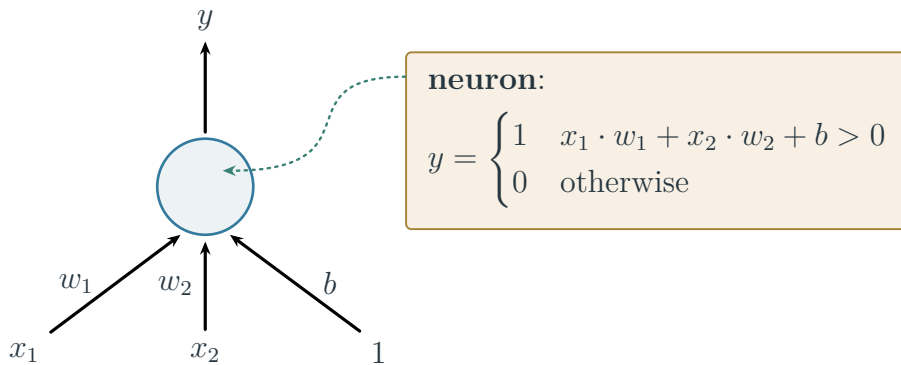


图 1.11: 单个感知机神经元示意。输入特征经线性加权后通过阈值函数输出类别。

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

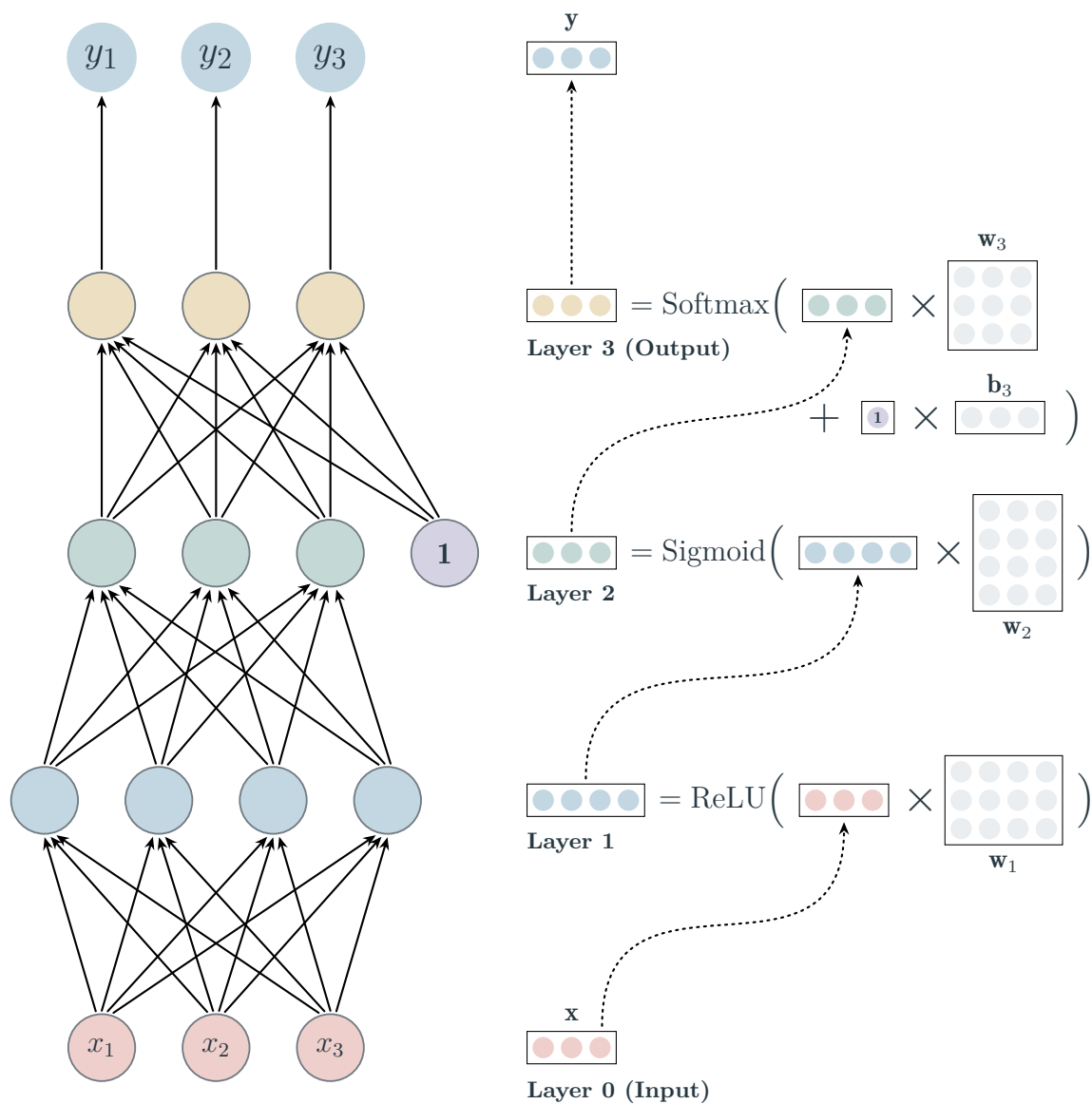


图 1.12: 多层神经网络结构示意。隐藏层逐层变换输入表示，并最终产生输出分布。

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。

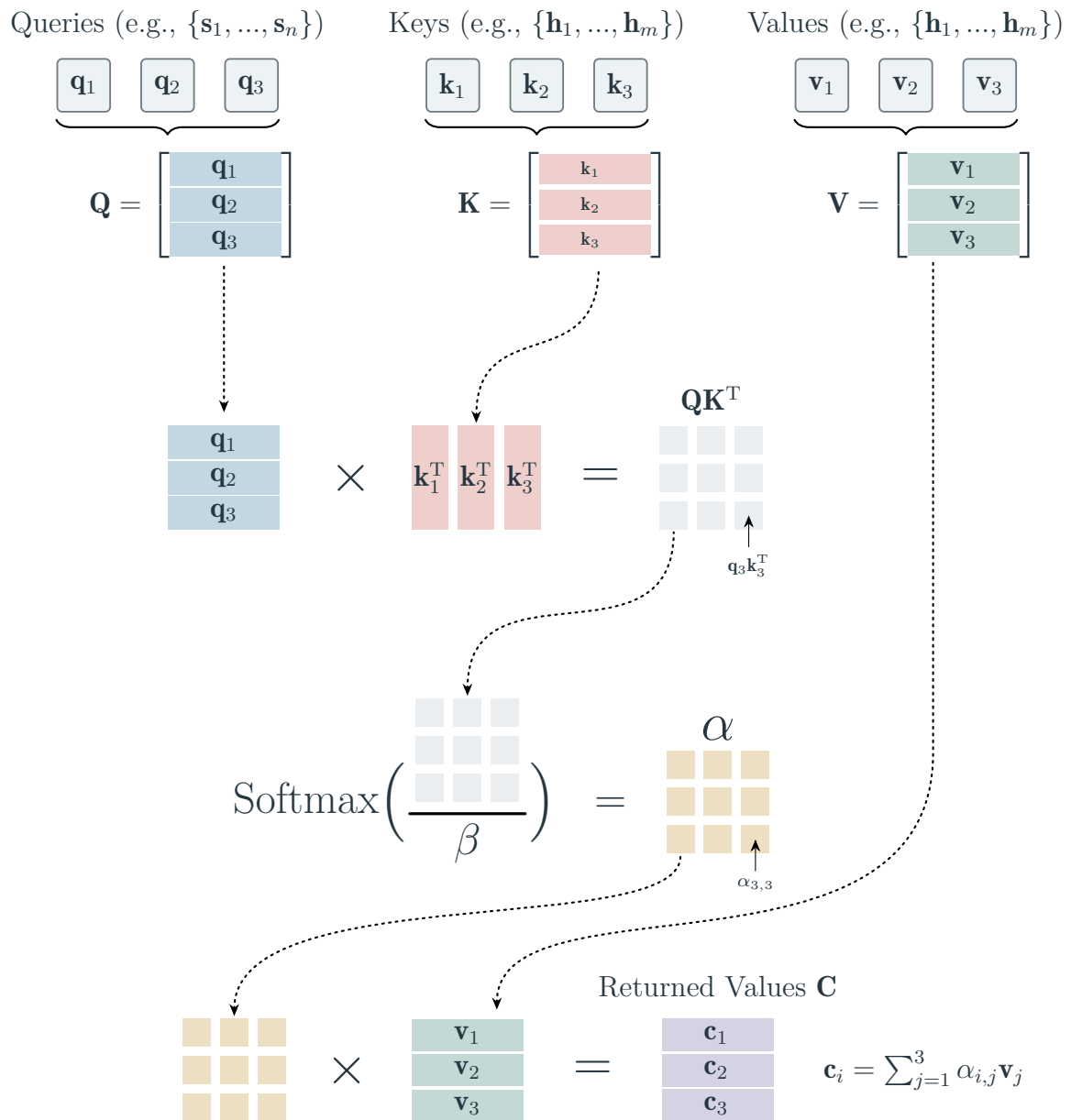


图 1.13: QKV 注意力机制示意。query 与 key 计算权重，再对 value 加权求和。

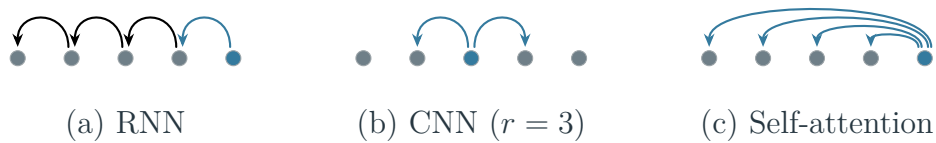


图 1.14: RNN、CNN 与自注意力中的信息流比较。自注意力可以让任意两个位置直接交互。

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

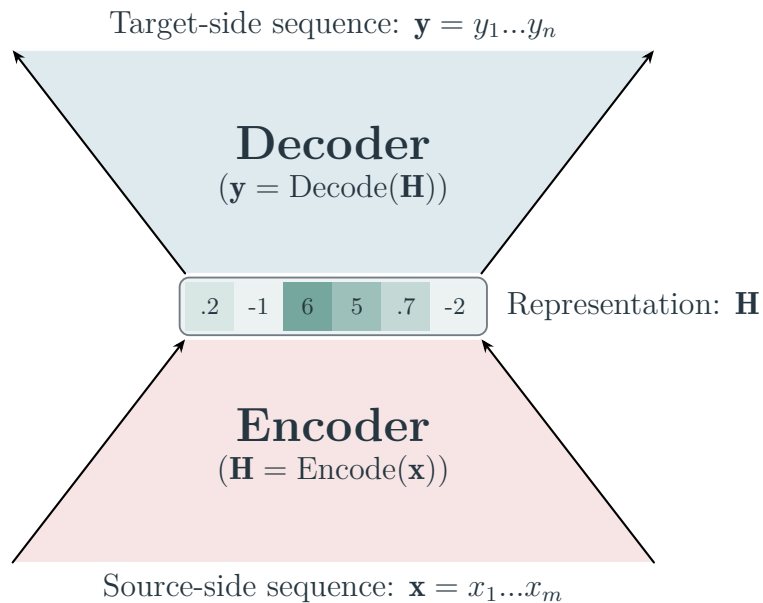


图 1.15: Encoder/Decoder 架构示意。编码器把源端序列映射为表示，解码器基于该表示生成目标端序列。

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 1.16 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。

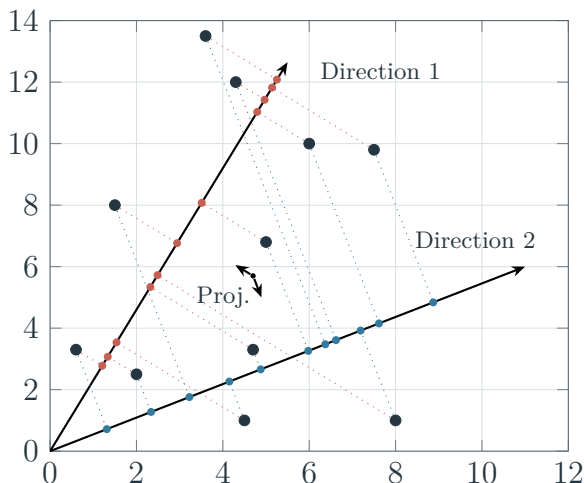


图 1.16: PCA 将二维数据投影到一维主方向的示意。不同投影方向会保留不同程度的方差信息。

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 1.17 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

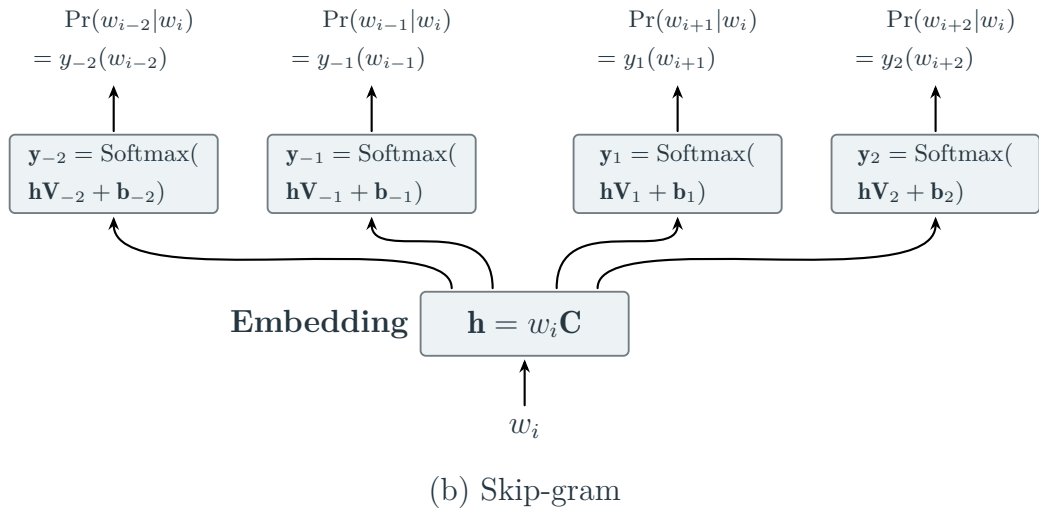
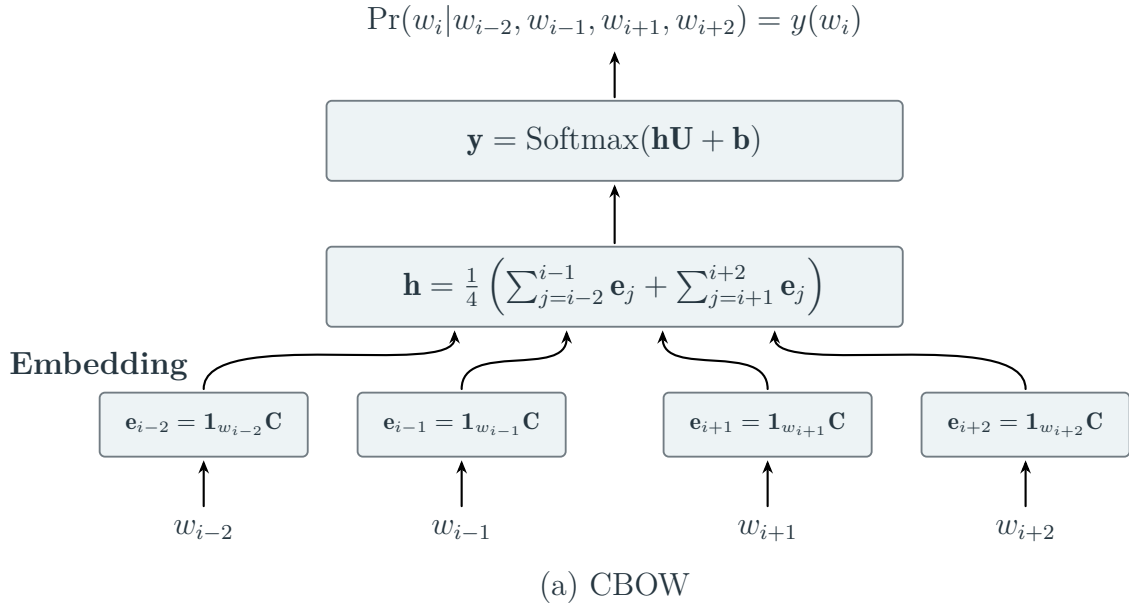


图 1.17: Word2Vec 中 CBOW 与 skip-gram 的训练结构示意图。二者分别由上下文预测中心词、由中心词预测上下文。

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 1.18 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。

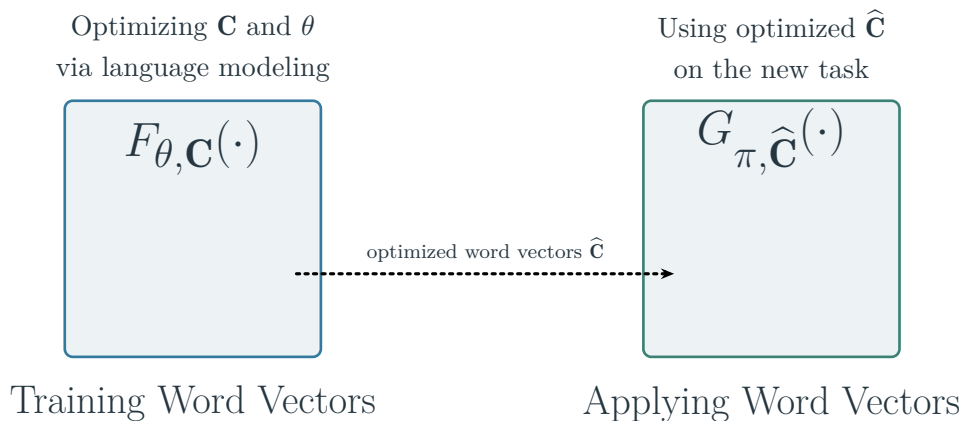


图 1.18: 通过神经语言模型预训练词向量并迁移到新任务的示意。

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 1.19 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 1.20 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 $\rightarrow$ 数据.

遮蔽语言模型可以恢复被遮蔽内容：

模型 [MASK] 数据表示 $\rightarrow$ 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 $\rightarrow$ 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。

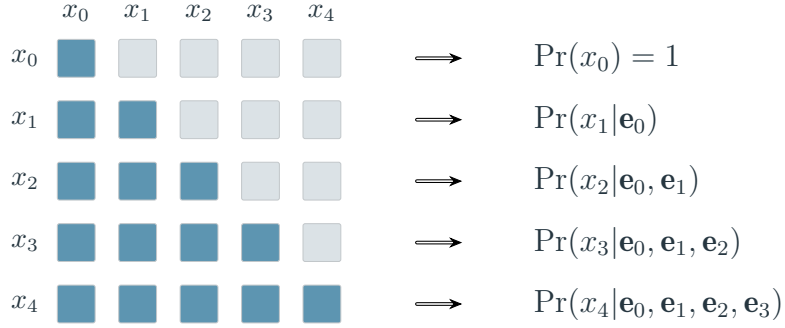
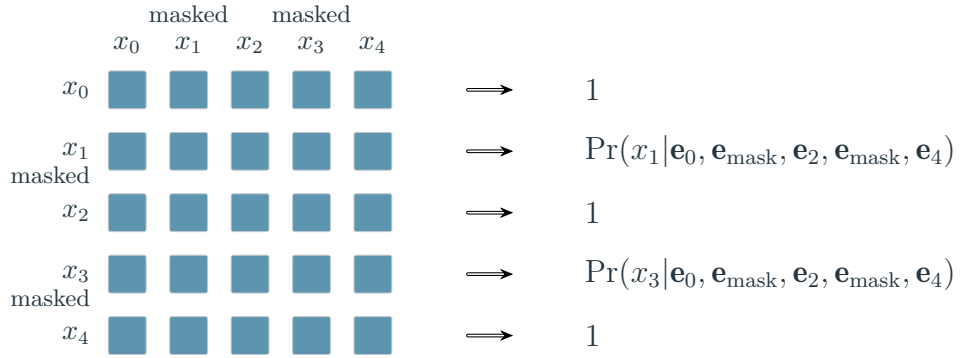
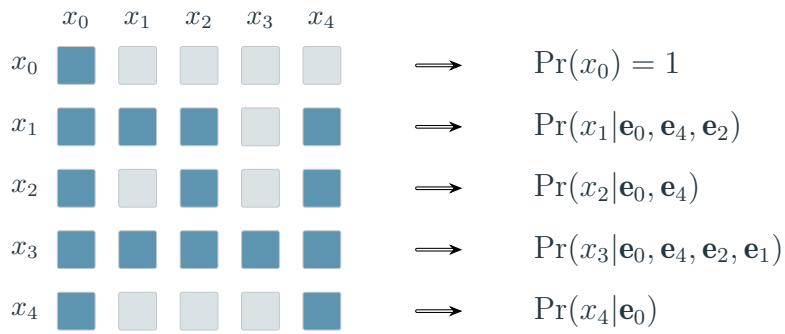
(a) Causal Language Modeling (order: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$)(b) Masked Language Modeling (order: $x_0, [\text{MASK}], x_2, [\text{MASK}], x_4 \rightarrow x_1, x_3$)(c) Permuted Language Modeling (order: $x_0 \rightarrow x_4 \rightarrow x_2 \rightarrow x_1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

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。

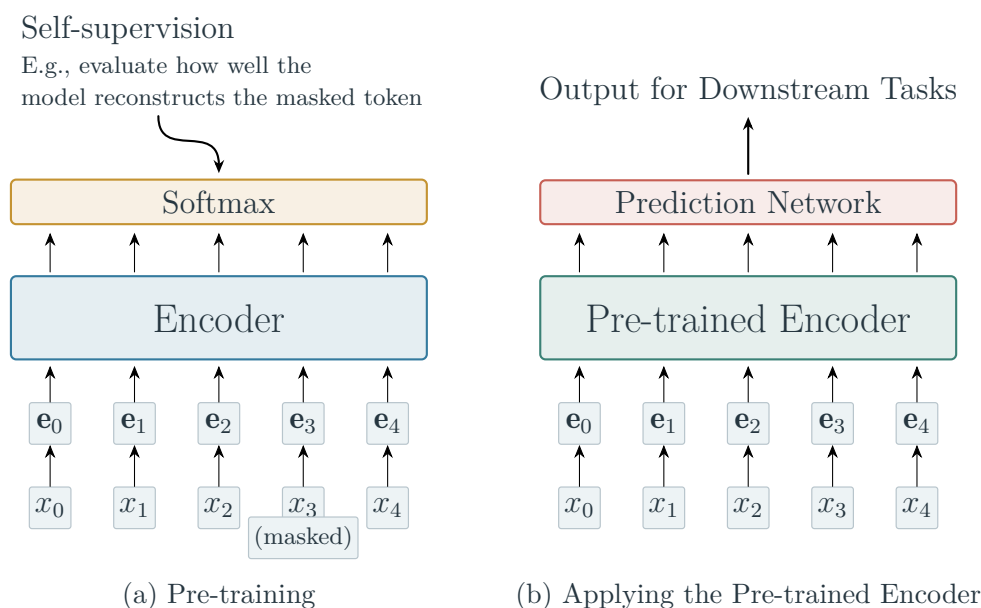


图 1.20: 预训练编码器及其下游应用示意。预训练得到的编码器可以迁移到分类、标注等任务中。

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 1.1 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。

第二章 强化学习基础

作者：朱梓铭、徐欣怡、王成龙

许多智能任务并不是对一个静态输入给出一次答案，而是连续作出选择。机器人当前的转向会改变下一时刻的位置，调度系统当前分配的资源会影响后续任务的等待时间，推荐系统展示的内容也可能改变用户之后的行为。此类任务中的动作既影响即时结果，也改变未来能够看到的状态和可以采取的动作。强化学习研究的正是这种**智能体通过与环境交互，依据反馈学习长期决策策略**的问题。

强化学习并非随着大模型才出现。它吸收了行为心理学中的试错学习、动态规划与最优控制等思想，早期主要用于控制、机器人、资源调度和博弈等序列决策问题。与监督学习依赖“输入—标签”样本不同，强化学习通常只得到奖励或代价：智能体必须探索不同动作，判断延迟出现的结果应当归因于此前哪些决策，并在利用已有经验与尝试未知行为之间作出权衡。价值函数、时序差分学习和策略优化等方法，都是围绕这些困难逐步发展起来的。

深度学习的发展进一步扩大了强化学习能够处理的状态规模。深度神经网络可以直接从图像等高维输入中学习表示，DQN 在一组 Atari 游戏中展示了端到端学习决策策略的能力；随后，深度强化学习与搜索、自我对弈结合，在围棋等复杂博弈中取得重要进展。强化学习的应用也由相对低维的控制问题，扩展到高维感知、复杂策略和大规模函数近似场景。

大模型出现后，强化学习获得了新的角色。预训练通过预测海量语料中的下一个词元学习通用知识和语言能力，但“更像训练语料”并不自动等于更符合用户意图。大模型后训练可以把模型看作策略，把生成过程看作一串动作，再利用人类偏好、奖励模型、规则检查器或可自动验证的任务结果提供反馈。基于人类反馈的强化学习（Reinforcement Learning from Human Feedback, RLHF）已被用于改善指令遵循和人类偏好对齐；在答案能够可靠验证的数学、编程等任务中，强化学习还可用于强化多步推理行为。

因此，在当下的大模型技术体系中，强化学习是连接“已有模型能力”与“目标行为”的重要后训练手段，也是训练可交互智能体和自主决策系统的基础工具。它并不替代预训练、监督微调、搜索或工具调用，也不是所有任务都必须采用的方案：只有当目标能够转化为可信反馈、决策会影响后续过程，并且训练数据与探索成本可以控制时，强化学习的优势才更容易发挥。奖励若不能代表真实目标，策略反而可能更有效地利用奖励漏洞；反馈昂贵或不能在线试错时，也需要离线强化学习、模仿学习或其他方法共同解决。

本章沿着三个问题展开：怎样把持续交互的任务写成一个明确的数学模型；怎样

利用有限的交互数据判断动作的长期价值并改进策略；当算法进入真实任务时，怎样检查状态、动作、奖励和评估方案是否设计正确。为使这些抽象问题始终落到同一任务上，下面引入五子棋作为贯穿案例。

设想一个刚开始训练的五子棋程序。它在一局棋中落下几十枚棋子，直到对局结束才得到胜负结果。如果程序输了，究竟是哪一步出了问题？最后一次漏掉防守当然可疑，但更早的一次落子也可能已经让局面无法挽回。反过来，一步当时没有得分的落子，也可能为十几步后的胜利创造条件。程序的每次选择都会改变后续局面，训练数据也会随着策略改变而变化，因而不能只用一次静态的“输入-标签”预测描述这一过程。

五子棋在本章中的作用是说明方法，而不是把某一种算法包装成完整的现成解法。除非另有说明，后文采用同一套设定：棋盘大小为 15×15 ，智能体执黑棋，对手及棋盘规则属于环境。智能体观察完整棋盘，在每个空位中选择一次落子；环境完成合法性检查、更新棋盘并执行对手回应，然后返回智能体下一次决策时看到的局面。任一方形成五子连线或棋盘填满时，对局自然终止。胜、负、平的核心奖励分别为 1、-1 和 0。本章中的数值计算只用于解释算法更新，不代表一套经过训练验证的五子棋系统。

学习目标

- 能够把一个序列决策任务表示为马尔可夫决策过程，并区分环境状态、智能体观测、即时奖励和长期回报
- 能够解释 Q-learning、策略梯度、Actor-Critic 和 PPO 的更新依据，以及这些方法各自试图解决的问题
- 能够根据任务目标设计状态、动作、奖励、约束和评估方案，并识别离线数据、训练稳定性、泛化和奖励偏差带来的风险

2.1 强化学习建模方法

一个强化学习任务首先要说清楚两件事：谁在作决定，以及一次决定会怎样改变后续处境。如图 2.1 所示，作决定的部分称为智能体；其余能够影响决策结果、但不由智能体直接控制的部分统称为环境。智能体接收信息、选择动作，环境执行动作并返回新的信息和奖励。下一次决策建立在这次交互产生的结果之上。

图中的边界并非总是唯一的。训练五子棋程序时，可以把对手看作环境的一部分，也可以把双方都看作独立智能体。两种划分会产生不同的数据和训练方法。因此，建模不是给既定对象贴上几个名称，而是决定模型能够观察什么、控制什么，以及环境负责完成什么。

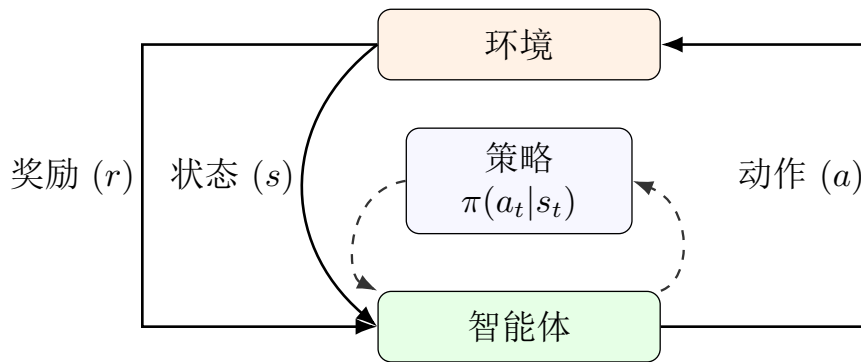


图 2.1: 强化学习的基本建模框架

2.1.1 马尔可夫决策过程

马尔可夫决策过程（Markov Decision Process, MDP）是描述上述交互的一种标准模型。它的关键要求不是“任务没有历史”，而是当前状态已经概括了预测未来所需的历史信息。给定当前状态和动作后，更早的状态、动作不应再改变下一状态的条件分布：

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = P(s_{t+1} | s_t, a_t) \quad (2.1)$$

其中， s_t 是环境在时刻 t 的状态， a_t 是随后执行的动作。这个等式是对状态表示的要求，而不是可以无条件接受的事实。如果机器人只输入一张静止图像，它可能无法判断前方物体正在靠近还是远离；此时单帧图像就不是充分状态。可以加入连续观测、速度信息或模型的记忆状态，重新构造更充分的决策信息。

一个 MDP 记为五元组：

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma) \quad (2.2)$$

其中， $\mathcal{S}$ 是状态空间， $\mathcal{A}$ 是动作空间， P 是状态转移分布， R 是奖励函数， $\gamma \in [0, 1]$ 是折扣因子。转移分布可写为

$$P(s' | s, a) = \Pr(S_{t+1} = s' | S_t = s, A_t = a) \quad (2.3)$$

奖励函数则描述一次转移产生的即时反馈：

$$R(s, a, s') = \mathbb{E}[R_t | S_t = s, A_t = a, S_{t+1} = s'] \quad (2.4)$$

正文后续用小写 r_t 表示一次实际交互得到的奖励样本。奖励函数可以采用其他等价定义，例如只依赖 (s, a) ；关键是全章使用同一时间约定：智能体在 s_t 执行 a_t ，随后收到 r_t 并进入 s_{t+1} 。

一次从初始状态到终止状态的完整交互称为轨迹：

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, a_{T-1}, r_{T-1}, s_T) \quad (2.5)$$

这里 s_T 是终止状态，最后一次动作和奖励分别是 a_{T-1} 与 r_{T-1} 。固定这一约定可以避免后续回报和 TD 目标出现一位下标偏差。

重点提示

马尔可夫性取决于状态如何构造。任务具有历史依赖，并不意味着它一定无法使用 MDP；真正的问题是当前状态是否已经保存了影响未来的历史信息。

2.1.2 关键元素介绍

表 2.1 将 MDP 中的主要对象与五子棋任务对应起来。这里把对手划入环境，并规定智能体每次只控制自己的落子。

表 2.1: 强化学习基本元素及其在五子棋中的对应关系

对象	数学记号	五子棋中的含义
智能体	—	需要学习落子策略的棋手
环境	—	棋盘规则、胜负判断以及对手行为
环境状态	s_t	完整棋盘、当前行动方及规则所需信息
智能体观测	o_t	实际提供给策略模型的信息；完全可观测时可与 s_t 相同
动作	$a_t \in \mathcal{A}$	在一个合法空位落子
即时奖励	r_t	本次落子及随后转移产生的反馈
策略	$\pi(a s)$	给定局面时各合法落子的选择规则或概率
轨迹	τ	从开局到胜、负或平局的完整交互序列

状态与观测的区别值得单独保留。环境状态是对真实决策过程的描述，观测则是智能体实际拿到的输入。五子棋棋盘通常完全可见，因此二者可以相同；机器人被遮挡的视野、推荐系统中未被记录的用户意图则使二者产生差异。后文为了简化公式，在完全可观测的讨论中使用 s_t 作为策略输入；涉及信息缺失时再显式使用 o_t 。

策略可以是确定性的，也可以是随机的。确定性策略为每个状态指定一个动作；随机策略写成条件分布

$$\pi(a | s) = \Pr(A_t = a | S_t = s) \quad (2.6)$$

训练的目标不是让策略复现某一条固定轨迹，而是让它在可能遇到的状态中作出长期效果更好的选择。

2.1.3 优化目标

一步动作的即时奖励不能完整说明它是否合理。从时刻 t 开始,智能体关心的是随后整段交互的折扣回报

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \quad (2.7)$$

折扣因子 γ 决定未来奖励在当前评价中的权重。较小的 γ 强调近期结果;接近1的 γ 保留更多长期影响。在有限回合任务中可以取 $\gamma = 1$;对于可能无限持续的任务,则需要保证回报及其期望有良好定义。

整条轨迹从初始时刻获得的回报记为 $R(\tau) = G_0$ 。参数化策略 π_θ 会诱导出轨迹分布 $p_\theta(\tau)$:策略改变后,智能体访问的状态和收集到的数据也随之改变。策略的性能定义为轨迹回报的期望:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[R(\tau)] \quad (2.8)$$

训练希望找到使该期望尽可能大的参数:

$$\theta^* = \arg \max_{\theta} J(\theta) \quad (2.9)$$

这个目标有两层难点。第一,不能枚举所有轨迹,只能用采样数据估计策略表现;第二,策略一旦改变,数据分布也会改变。后续算法的差别,主要在于它们怎样估计动作的长期影响,以及怎样利用这种估计更新策略。

2.2 强化学习的基本优化算法

在前一节中,强化学习问题被形式化为长期累积奖励最大化问题,其核心目标是寻找能够使性能函数 $J(\theta)$ 最大化的最优策略。然而,在实际任务中,环境的状态转移规律通常难以完全获得,所有可能的交互轨迹也无法被逐一枚举,因此很难直接计算性能函数并求解最优策略。强化学习算法需要利用智能体与环境交互产生的有限样本,对不同状态和动作所带来的长期收益进行估计,并依据估计结果不断改进决策策略。

根据策略改进方式的不同,强化学习的基本优化算法可以分为基于价值函数的方法和基于策略优化的方法。基于价值函数的方法首先估计某一状态或状态与动作组合能够获得的长期回报,再根据价值估计结果选择动作,将策略优化问题转化为价值函数的迭代更新问题。基于策略优化的方法则直接构建参数化策略,通过调整策略参数,提高高回报动作被选择的概率,从而实现对长期累积奖励的优化。两类方法在优化对象、动作选择方式和适用场景等方面具有不同特点。

在此基础上,Actor-Critic框架将价值估计与策略优化结合起来。其中,Actor负责根据当前状态选择动作并更新策略,Critic负责评估当前策略或动作的长期收益,

并为 Actor 的更新提供指导。该框架既保留了策略优化方法处理随机策略和连续动作空间的能力，也利用价值函数降低策略更新过程中的估计波动，成为许多现代强化学习算法的基础。

2.2.1 基于价值函数的方法

五子棋中的多数落子不会立即产生胜负奖励，却会改变后续局面。价值方法要解决的问题正是：当即时奖励不足以评价动作时，怎样估计它对未来回报的影响？

在策略 π 下，状态价值函数定义为

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t \mid S_t = s_t] \quad (2.10)$$

它评价的是“处于这个状态，并继续按照 π 行动”的期望回报。动作价值函数进一步固定当前动作：

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t \mid S_t = s_t, A_t = a_t] \quad (2.11)$$

因此， V^π 比较局面， Q^π 比较同一局面中的候选动作。两者都依赖后续策略：同一个棋盘局面由强策略继续下和由随机策略继续下，其价值一般不同。

图 2.2 给出了某一棋盘局面下不同候选落子位置的动作价值，以及智能体依据动作价值进行落子选择的过程。

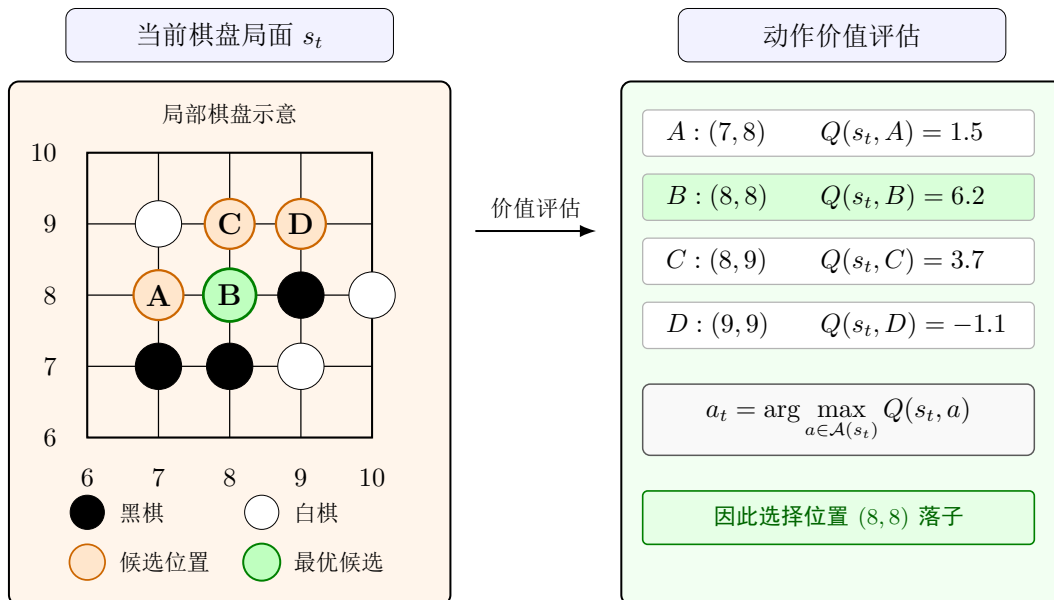


图 2.2: 基于动作价值的五子棋落子选择

图中位置 (8,8) 的估计价值最高。若这些估计足够准确，贪心策略会在合法动作集合 $\mathcal{A}(s_t)$ 中选择

$$a_t = \arg \max_{a \in \mathcal{A}(s_t)} Q(s_t, a) \quad (2.12)$$

这里最难的部分不是取最大值，而是得到可信的 Q 值。回报满足

$$G_t = r_t + \gamma G_{t+1} \quad (2.13)$$

所以长期问题可以拆成“本次奖励”和“下一状态之后的回报”。对最优动作价值函数，这一递推成为贝尔曼最优方程：

$$Q^*(s_t, a_t) = \mathbb{E} \left[r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q^*(s_{t+1}, a') \mid s_t, a_t \right] \quad (2.14)$$

期望来自环境转移的不确定性；最大值表示下一状态采用当前认为最优的动作。若 s_{t+1} 是自然终止状态，后续回报为零，最大值项也应置零。

Q-learning 不要求提前知道转移概率。每观察到一次转移 (s_t, a_t, r_t, s_{t+1}) ，它就用单步目标

$$y_t = r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a') \quad (2.15)$$

修正旧估计：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [y_t - Q(s_t, a_t)] \quad (2.16)$$

其中 α 是学习率，差值 $y_t - Q(s_t, a_t)$ 是这次样本带来的时序差分误差。更新目标使用下一状态的最大动作价值，而采集样本时不必始终执行贪心动作；这正是 Q-learning 的离策略特征。

以图中的位置 (8,8) 为例。设旧估计为 2.0，落子后的即时奖励为 0，下一决策状态的最大动作价值为 6.0，并取 $\gamma = 0.9$ 、 $\alpha = 0.5$ 。图 2.3 展示了这次更新。

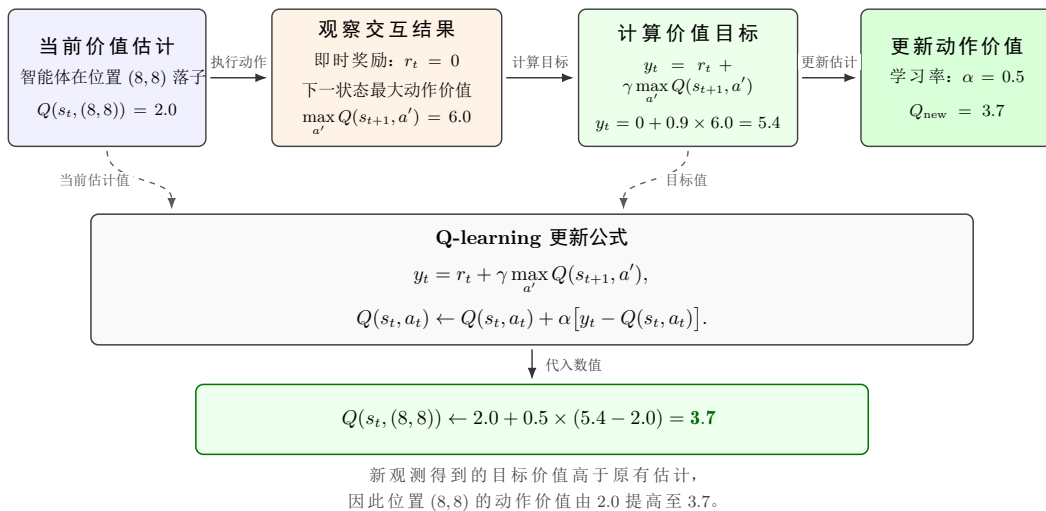


图 2.3: Q-learning 动作价值更新过程示意

单步目标为

$$y_t = 0 + 0.9 \times 6.0 = 5.4 \quad (2.17)$$

因此

$$Q(s_t, (8, 8)) \leftarrow 2.0 + 0.5 \times (5.4 - 2.0) = 3.7 \quad (2.18)$$

目标值高于原估计，说明这次观察到的后续局面比原先预期更好。一次更新只把估计向 5.4 推进，并不会立刻把它改成目标值。反复采样让终局奖励逐步传回更早的状态，但传递速度和估计质量仍取决于数据覆盖、学习率与函数近似方式。

如果始终选择当前估计最高的动作，智能体可能永远收集不到其他动作的数据。常用的 ε -greedy 行为策略是

$$a_t = \begin{cases} \text{从 } \mathcal{A}(s_t) \text{ 中随机选择动作,} & \text{概率为 } \varepsilon, \\ \arg \max_{a \in \mathcal{A}(s_t)} Q(s_t, a), & \text{概率为 } 1 - \varepsilon. \end{cases} \quad (2.19)$$

当 $\varepsilon = 0.1$ 时，约 10% 的决策在合法动作中随机探索。训练后期常逐渐减小 ε ，但不能只按固定日程机械衰减：如果重要状态仍未被覆盖，过早停止探索只会把当前误差固化下来。

重点提示

Q-learning 的示例说明了价值如何从下一状态传回当前动作，并不意味着表格型 Q-learning 能够直接解决完整五子棋。即使不考虑棋子顺序， 15×15 棋盘的合法局面数量也极其庞大，绝大多数状态不可能被重复访问到足够多次。

小规模任务可以用表格保存每个状态—动作对的估计；大规模任务则要借助神经网络等函数近似器，让相似状态共享统计信息。深度 Q 网络属于后一类方法，但它还需要经验回放、目标网络等稳定化设计，本章不在这里展开。价值方法也更自然地适用于能够枚举或有效比较候选动作的场景。下一节换一个角度：不先为每个动作学习一个价值，再取最大值，而是直接调整动作的选择概率。

2.2.2 基于策略优化的方法

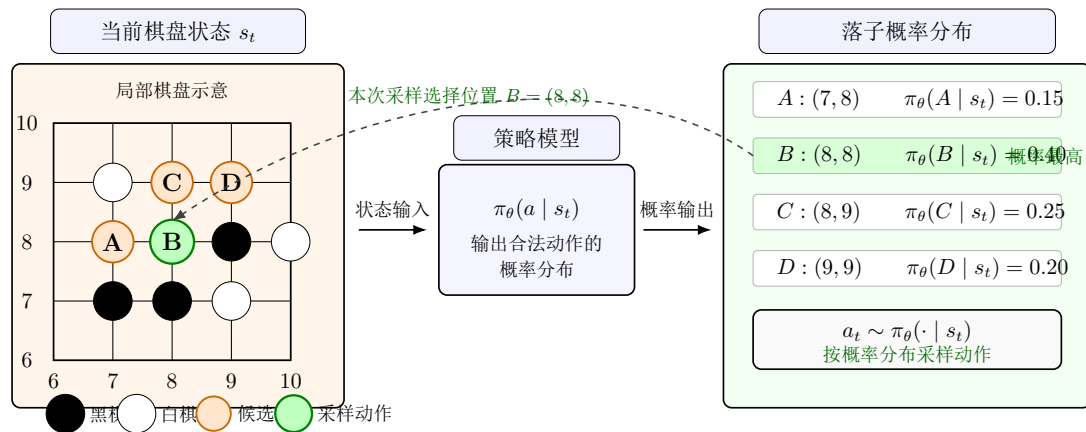
价值方法先回答“每个动作值多少”，再选择价值最高的动作。策略方法跳过这个中间决策规则，直接学习“在当前状态下，各动作应当以多大概率被选择”。这使它自然表示随机策略，也不必在连续动作空间中枚举所有候选动作。

设参数化策略为：

$$\pi_{\theta}(a_t | s_t) \quad (2.20)$$

其中 θ 是策略参数。离散动作策略输出概率质量函数；连续动作策略通常输出高斯分布等概率分布的参数。两种情况下，训练期间都可以从分布中采样动作。

在五子棋中，策略网络接收棋盘状态，输出各位置的 logits。合法动作掩码排除已经落子的格点，归一化后得到空位上的概率分布。图 2.4 给出一个简化示例。



图中四个候选位置的概率分别为 0.15、0.40、0.25 和 0.20。位置 $B = (8, 8)$ 最可能被选中，但并非必然被选中：

$$a_t \sim \pi_\theta(\cdot | s_t) \quad (2.21)$$

这种随机性保留了探索机会。评估时是否继续采样要由评估目标决定；如果只取最大概率动作，测得的是确定性部署规则的表现，而不是训练时随机策略的完整表现。

策略应怎样利用对弈结果改变这些概率？设 π_θ 生成一条轨迹

$$\tau = (s_0, a_0, r_0, s_1, \dots, a_{T-1}, r_{T-1}, s_T) \quad (2.22)$$

其目标仍是最大化期望轨迹回报：

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [R(\tau)] \quad (2.23)$$

利用对数导数技巧，可以把无法直接求和的轨迹期望转化为可用采样估计的策略梯度：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T-1} G_t \nabla_\theta \log \pi_\theta(a_t | s_t) \right] \quad (2.24)$$

其中

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \quad (2.25)$$

梯度中的 $\nabla_\theta \log \pi_\theta(a_t | s_t)$ 指向提高已采样动作对数概率的方向， G_t 决定这个方向的权重和符号。这里不能简单说“回报较低就降低概率”：只要回报仍为正，原始 REINFORCE 更新仍可能提高该动作概率。要判断一次结果是好于还是差于当前局面的正常水平，需要引入基线。

直接以蒙特卡洛回报估计上述梯度的经典算法称为 REINFORCE。用当前策略

采样 N 条轨迹后，可最小化

$$\mathcal{L}_{\text{PG}}(\theta) = -\frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T_i-1} G_t^{(i)} \log \pi_{\theta} \left(a_t^{(i)} \mid s_t^{(i)} \right) \quad (2.26)$$

负号把回报最大化写成损失最小化。REINFORCE 的优点是梯度估计在常见条件下无偏，代价是必须等到后续奖励出现才能计算回报，而且不同轨迹的结果可能相差很大。

仅依靠完整对局的最终结果更新策略，也会带来一些问题。五子棋的一局对弈通常包含多个动作，最终胜负是双方多次落子共同作用的结果。智能体可能在前期连续作出合理决策，却因为后期的一次失误输掉整局。如果将负回报直接作用于此前所有动作，一些原本合理的落子也会受到错误抑制。类似地，获胜对局中也可能包含效果较差的动作，但这些动作可能因为最终获胜而获得正向更新。

这就是时间上的信用分配问题。同一局面还可能因为对手行动和后续采样不同而得到相反结果，所以单条轨迹的 G_t 是噪声很大的估计。增加样本能缓解波动，却不能让模型更早获得反馈。

为了减小回报波动对策略更新的影响，可以引入基线 $b(s_t)$ ，用动作回报与基线之间的差值代替原始回报：

$$\mathcal{L}_{\text{PG}}(\theta) = -\mathbb{E} \left[\sum_{t=0}^{T-1} (G_t - b(s_t)) \log \pi_{\theta}(a_t \mid s_t) \right] \quad (2.27)$$

只要基线不依赖当前采样动作，从策略梯度中减去它不会改变梯度的期望，却能降低方差。状态相关基线可以理解为当前局面的正常回报水平：更新关注的从此不再是“这次回报是否为正”，而是“这次结果是否好于在该局面下的通常表现”。

较为常用的状态相关基线是状态价值函数：

$$b(s_t) = V^{\pi}(s_t) \quad (2.28)$$

此时，回报与基线之间的差值可以写为：

$$\hat{A}_t = G_t - V^{\pi}(s_t) \quad (2.29)$$

这给出优势函数 $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$ 的蒙特卡洛估计。正优势提高已采样动作的概率，负优势降低它；这种相对比较比回报的绝对正负更有意义。

仍以前文的五子棋局面为例。假设智能体在状态 s_t 下选择了位置 $B = (8, 8)$ ，当前选择概率为 $\pi_{\theta}(B \mid s_t) = 0.40$ 。图 2.5 展示了该动作在两种不同回报情况下的更新结果。

在图 2.5 的第一种情况下，动作 B 获得的实际回报为 8，而当前状态的基线为 6，

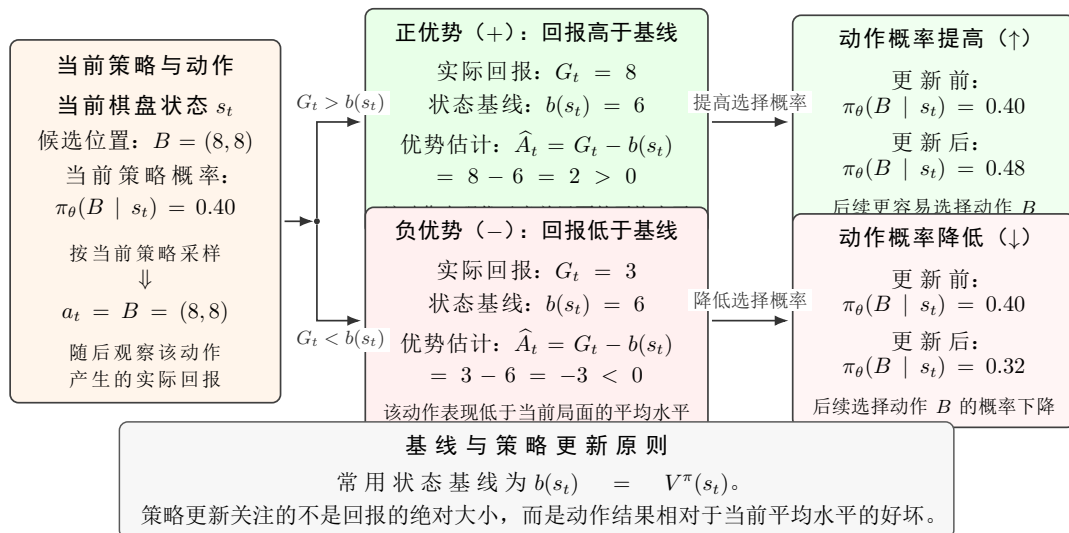


图 2.5: 回报与基线对落子概率更新的影响

因此优势估计为:

$$\hat{A}_t = 8 - 6 = 2 > 0 \quad (2.30)$$

这一结果表明, 位置 B 的实际表现高于当前局面的平均水平, 策略会提高该动作的选择概率。

在第二种情况下, 动作 B 获得的实际回报为 3, 低于基线 6, 因此有:

$$\hat{A}_t = 3 - 6 = -3 < 0 \quad (2.31)$$

此时, 策略会降低位置 B 的选择概率。可见, 优势估计关注的是动作表现与当前状态一般水平之间的差异, 而不是回报的绝对大小。

重点提示

原始 REINFORCE 用采样回报给动作加权, 并不是简单执行“正奖励鼓励、低奖励惩罚”。引入状态价值基线后, $G_t - V^\pi(s_t)$ 才显式比较这次动作与当前状态正常水平之间的差异。基线降低方差, 但不会自动解决长序列中的信用分配问题。

如果不想等到整局结束, 可以用一个学习得到的价值模型近似后续回报。这样会引入估计偏差, 却能更及时地更新策略。Actor-Critic 正是沿着这一取舍建立的。

2.2.3 Actor-Critic 框架

REINFORCE 用实际采样到的后续回报评价动作。这个评价不依赖一个可能出错的价值模型, 却来得晚、波动大。Actor-Critic 用学习得到的价值估计替代一部分尚未发生的回报, 在更早更新和更小方差之间换取一定偏差。

框架包含两个可训练部分: Actor 是参数为 θ 的策略 $\pi_\theta(a | s)$, 负责行动; Critic

是参数为 ϕ 的价值近似器 $V_\phi(s)$ ，负责预测当前策略下的期望回报。二者可以使用独立网络，也可以共享特征提取层，不能仅凭“两个角色”推断实现中一定存在两张完全独立的网络。

在五子棋任务中，一条样本的产生和使用过程如下。Actor 接收 s_t 并输出

$$\pi_\theta(a_t | s_t) \quad (2.32)$$

并采样 a_t 。环境执行动作和约定范围内的对手回应，返回 r_t 、 s_{t+1} 与终止标记。Critic 随后用相邻状态的价值构造训练目标。图 2.6 展示了这条数据流。

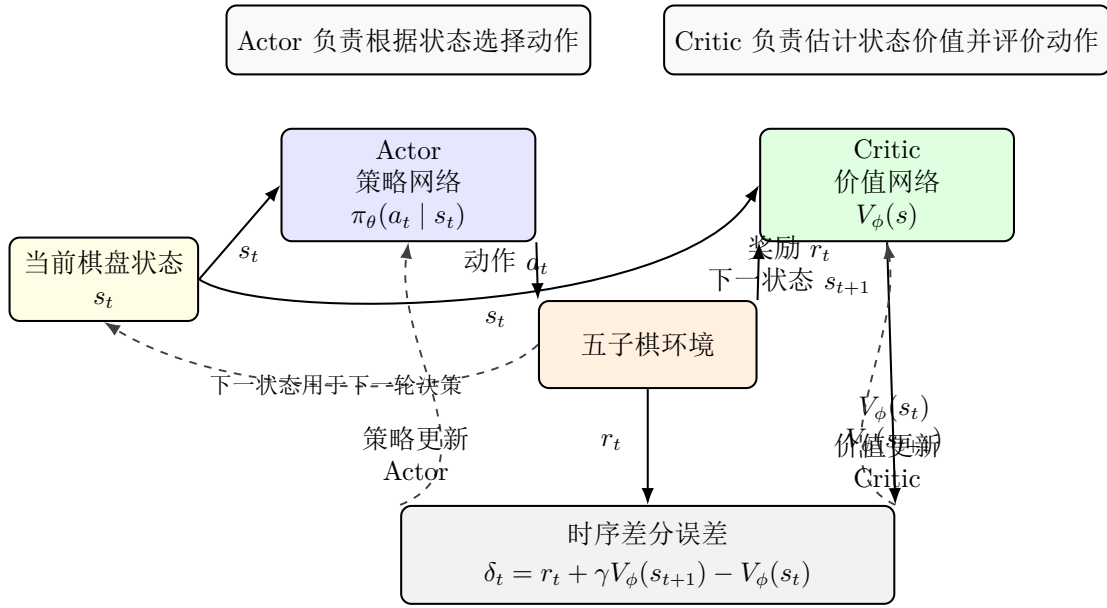


图 2.6: Actor-Critic 框架中的策略选择与价值评价过程

Actor 回答“下一步按什么分布行动”，Critic 回答“从当前状态继续执行该策略，预计能得到多少回报”。动作相对于当前状态平均水平的好坏由优势函数表示：

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \quad (2.33)$$

正优势表示该动作优于当前策略在这个状态下的平均选择，负优势表示它更差。Actor 分别提高或降低相应动作的对数概率。

真实的 $Q^\pi(s_t, a_t)$ 未知时，可以用一步奖励加下一状态价值构造 TD 目标：

$$y_t = r_t + \gamma(1 - d_t)V_{\bar{\phi}}(s_{t+1}) \quad (2.34)$$

其中 $d_t = 1$ 表示任务在本次转移后自然终止，否则为 0。记号 $\bar{\phi}$ 强调计算目标时不通过下一状态价值反向传播梯度；实现中可以对这一项停止梯度，也可以使用单独维

护的目标网络。TD 误差为

$$\delta_t = r_t + \gamma(1 - d_t)V_{\phi}(s_{t+1}) - V_{\phi}(s_t) \quad (2.35)$$

在最简单的单步 Actor-Critic 中, 可用 δ_t 近似优势:

$$\hat{A}_t = \delta_t \quad (2.36)$$

仍以前文的棋盘为例。Actor 以 0.40 的概率选择位置 $B = (8, 8)$ 。该动作没有自然终止对局, 所以 $d_t = 0$, 并设

$$r_t = 0 \quad (2.37)$$

$$V_{\phi}(s_t) = 4.0 \quad (2.38)$$

$$V_{\phi}(s_{t+1}) = 6.0 \quad (2.39)$$

当 $\gamma = 0.9$ 时,

$$\delta_t = 0 + 0.9 \times 6.0 - 4.0 = 1.4 \quad (2.40)$$

图 2.7 给出了该评价信号对 Actor 和 Critic 的作用。

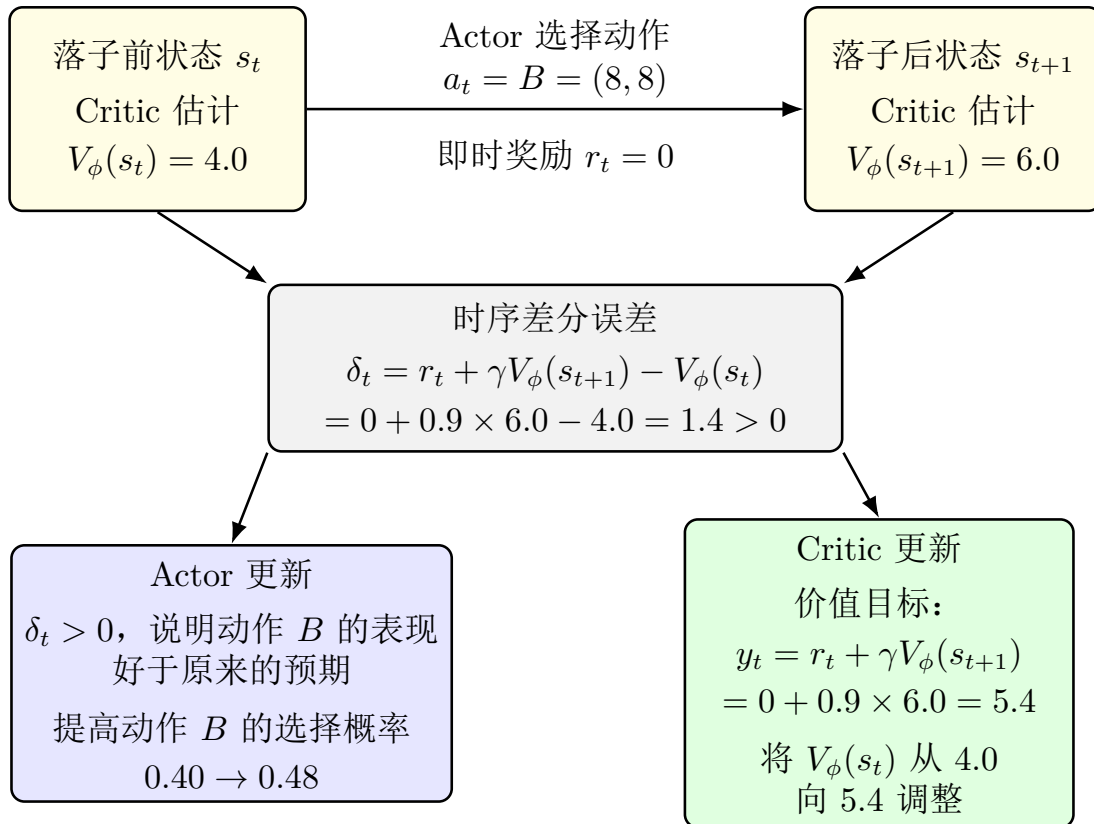


图 2.7: 时序差分误差对 Actor 与 Critic 更新的作用

这个样本给出的 TD 目标是 5.4，高于 Critic 对 s_t 的旧估计 4.0，所以 $\delta_t = 1.4$ 。Actor 把它当作正优势，提高动作 B 的对数概率；Critic 则把 $V_\phi(s_t)$ 向 5.4 调整。需要注意，这只是当前 Critic 提供的学习信号，并不能证明动作 B 客观上一定是好棋。

Actor 使用优势估计更新策略，其损失函数可以写为：

$$\mathcal{L}_{\text{actor}}(\theta) = -\mathbb{E} \left[\hat{A}_t \log \pi_\theta(a_t | s_t) \right] \quad (2.41)$$

计算 Actor 梯度时， $\hat{A}_t$ 应被视为固定目标，不让梯度通过优势估计回流到 Critic。正负优势决定更新方向，绝对值影响单个样本的权重。

Critic 的任务是使当前状态价值接近 TD 目标：

$$y_t = r_t + \gamma(1 - d_t)V_\phi(s_{t+1}) \quad (2.42)$$

在上述例子中：

$$y_t = 0 + 0.9 \times 6.0 = 5.4 \quad (2.43)$$

Critic 原先对状态 s_t 的估计为 4.0，而新的目标值为 5.4，因此需要适当提高 $V_\phi(s_t)$ 。Critic 可以使用平方误差进行训练：

$$\mathcal{L}_{\text{critic}}(\phi) = \frac{1}{2} \mathbb{E} [(y_t - V_\phi(s_t))^2] \quad (2.44)$$

若下一状态的估计价值只有 2.0，则

$$V_\phi(s_t) = 4.0, \quad V_\phi(s_{t+1}) = 2.0 \quad (2.45)$$

在即时奖励仍为 0、折扣因子为 0.9 时，有：

$$\delta_t = 0 + 0.9 \times 2.0 - 4.0 = -2.2 \quad (2.46)$$

Actor 会降低该动作的概率，Critic 会把当前状态估计向 1.8 调整。若落子直接结束对局， $d_t = 1$ ，目标只剩最终奖励。时间上限造成的截断不一定等于自然终止：如果任务在时间上限之后本可继续，是否保留 bootstrap 项要根据环境语义处理。

Actor-Critic 框架中，Actor 负责输出动作概率并更新策略，Critic 负责估计状态价值。时序差分误差

$$\delta_t = r_t + \gamma(1 - d_t)V_\phi(s_{t+1}) - V_\phi(s_t)$$

可以作为单步优势估计，也给出 Critic 当前预测与 TD 目标的差异。它是带有函数近似误差的训练信号，不是动作质量的真实标签。

一条基本更新链可以概括为：Actor 采样动作，环境产生转移，Critic 构造 TD 目

标并学习价值，Actor 使用停止梯度的优势信号更新策略。它比纯蒙特卡洛更新更及时、方差通常更小，代价是 Actor 会继承 Critic 的偏差。

二者的学习速度失衡时，这个问题尤其明显。Critic 学得太慢，优势长期带有系统误差；Actor 变化太快，Critic 刚拟合的数据分布已经过时；Critic 在有限样本上过拟合，也会向 Actor 提供过度自信的方向。后续的 PPO 仍采用 Actor-Critic 结构，它重点限制的是策略一次更新的幅度，而不是消除 Critic 的估计误差。

2.3 强化学习的进阶优化算法

Actor-Critic 给出了策略与价值函数协同更新的框架，实际训练还要面对两个具体限制。第一，策略一旦变化太快，旧策略采集的优势估计便不能可靠地描述新策略；第二，有些任务根本不允许当前策略不断进入环境试错。PPO 处理前一种更新尺度问题，离线强化学习处理后一种固定数据问题。二者解决的困难不同，不能因为都“减少训练风险”而混为一谈。

2.3.1 近端策略优化方法

Actor-Critic 有了评价动作的方向，却没有回答一次应当走多远。设旧策略在某个棋盘状态下以 0.40 的概率选择位置 B 。正优势说明这个概率应当提高，但从 0.40 调到 0.46 和一步调到 0.80 完全不是同一件事。后者会显著改变后续访问的局面，而用于计算优势的样本仍来自概率为 0.40 的旧策略。

单纯减小学习率不能直接解决这个问题。学习率约束参数移动的距离，同样大小的参数变化却可能在不同状态下造成完全不同的概率变化。近端策略优化（Proximal Policy Optimization, PPO）因此直接比较新旧策略对已采样动作给出的概率，并让过于激进的变化不再从代理目标中继续获益。

PPO 仍采用 Actor-Critic 结构。Critic 提供优势估计，主要变化发生在 Actor 的目标函数。“近端”描述的是希望新策略留在采样策略附近，并不是一个由裁剪严格保证的硬约束。

在每一轮 PPO 训练开始时，先固定当前策略，并将其记为旧策略：

$$\pi_{\theta_{\text{old}}}(a_t | s_t) \quad (2.47)$$

使用旧策略与环境交互后，可以得到一批状态、动作和奖励数据。更新 Actor 时，新策略与旧策略可能对同一动作给出不同的选择概率。为衡量这一变化，定义概率比：

$$\rho_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \quad (2.48)$$

概率比反映新旧策略对已采样动作 a_t 的偏好变化。 $\rho_t(\theta) > 1$ 表示概率提高，

$\rho_t(\theta) < 1$ 表示概率降低。它只比较样本中实际执行的动作，并不直接约束该状态下整个动作分布。

例如，旧策略选择位置 B 的概率为：

$$\pi_{\theta_{\text{old}}}(B | s_t) = 0.40$$

如果新策略将该概率提高到 0.48，则有：

$$\rho_t(\theta) = \frac{0.48}{0.40} = 1.20 \quad (2.49)$$

这表示新策略选择位置 B 的概率是旧策略的 1.20 倍。概率比比参数距离更直接地反映这条样本上的行为变化。

若暂时不限制策略变化，可以利用概率比与优势估计构造代理目标：

$$L^{\text{PG}}(\theta) = \mathbb{E} [\rho_t(\theta) \hat{A}_t] \quad (2.50)$$

其中， $\hat{A}_t$ 表示 Critic 给出的优势估计。当 $\hat{A}_t > 0$ 时，动作 a_t 的表现高于当前策略的一般水平，提高该动作的概率会增大代理目标；当 $\hat{A}_t < 0$ 时，该动作的表现低于一般水平，降低其选择概率会增大代理目标。

如果对同一批样本反复优化，Actor 可能持续增大正优势动作的概率，或持续减小负优势动作的概率。PPO 为概率比设置裁剪范围：

$$[1 - \epsilon, 1 + \epsilon]$$

其中， ϵ 表示裁剪阈值。PPO 的裁剪代理目标写为：

$$L^{\text{CLIP}}(\theta) = \mathbb{E} \left[\min \left(\rho_t(\theta) \hat{A}_t, \text{clip}(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (2.51)$$

式中的 $\min$ 不能忽略：PPO 取原始项与裁剪项中更保守的一个。裁剪只在变化会继续改善代理目标的方向上截断收益，并不是把所有概率比强制投影到区间内。 ϵ 是需要验证的超参数，不能脱离批量大小、更新轮数和学习率单独判断。

裁剪机制对正优势动作和负优势动作的作用可以通过五子棋中的概率变化进行说明。图 2.8 给出了旧策略概率为 0.40、裁剪阈值为 $\epsilon = 0.2$ 时的两种情况。

在图 2.8 中，裁剪区间为：

$$[1 - \epsilon, 1 + \epsilon] = [0.8, 1.2]$$

当位置 B 的优势估计为正时，PPO 允许适当提高该动作的选择概率。若新策略将概率从 0.40 提高到 0.48，概率比值为：

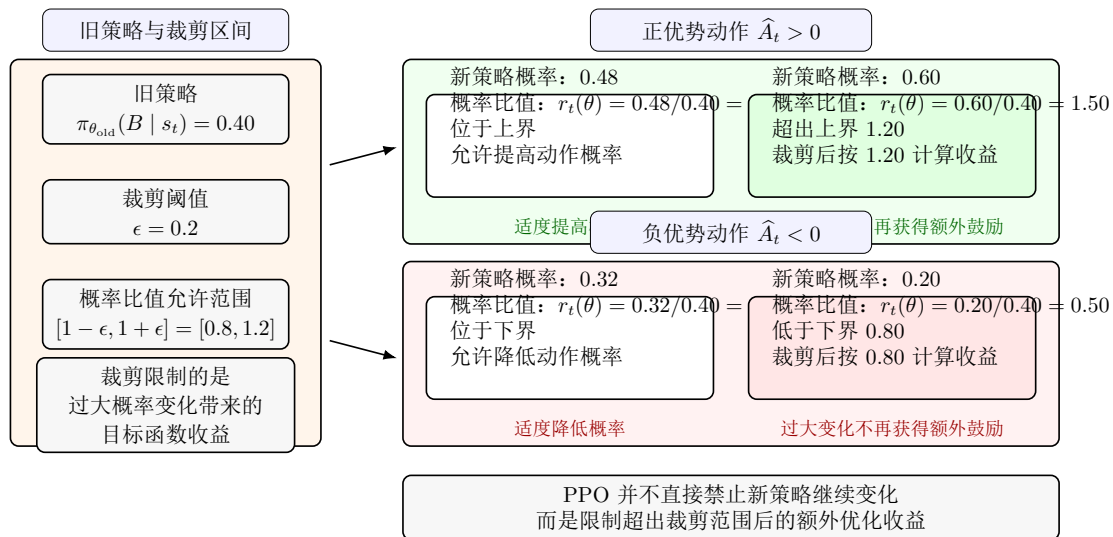


图 2.8: PPO 裁剪机制对策略概率更新的限制

$$\rho_t(\theta) = \frac{0.48}{0.40} = 1.2$$

该比值位于裁剪上界，概率增加仍然能够反映在代理目标中。若新策略进一步将概率提高到 0.60，概率比值变为：

$$\rho_t(\theta) = \frac{0.60}{0.40} = 1.5$$

此时概率比值已经超过上界。对于正优势动作，超过上界的部分不会继续增加裁剪目标，因而优化过程不再强烈推动该动作概率继续增大。

当某一动作的优势估计为负时，PPO 会降低该动作的选择概率。如果新策略将概率从 0.40 降低到 0.32，概率比值为：

$$\rho_t(\theta) = \frac{0.32}{0.40} = 0.8$$

该比值位于裁剪下界。如果概率继续降低到 0.20，概率比值变为：

$$\rho_t(\theta) = \frac{0.20}{0.40} = 0.5$$

对于负优势动作，低于裁剪下界的变化不会继续带来更多优化收益，策略因而缺少进一步大幅降低该动作概率的动力。

裁剪机制并不会将新策略的实际概率强制固定在某个区间内。它限制的是过大概率变化能够从代理目标中获得的收益。策略仍然可以发生变化，但当变化超过一定范围后，继续扩大变化不再受到目标函数的鼓励。

这种做法与信任域思想相近。信任域可以理解为旧策略附近的一定范围，策略更

新尽量在这一范围内完成。PPO 不需要直接求解带有严格约束的复杂优化问题，而是通过裁剪代理目标近似控制更新幅度。在实际训练中，还可以监测新旧策略之间的 KL 散度。当二者差异超过设定范围时，可以提前结束当前批次的策略更新。

除了限制策略变化，PPO 还会在有限范围内重复使用同一批交互数据。它先用旧策略采集一批样本，再将数据划分为小批次，对 Actor 和 Critic 进行多轮训练。裁剪机制减弱了策略在这些更新中偏离采样策略过远的动机，但并不保证偏离一定不会发生。

图 2.9 展示了 PPO 一次训练迭代的主要过程。

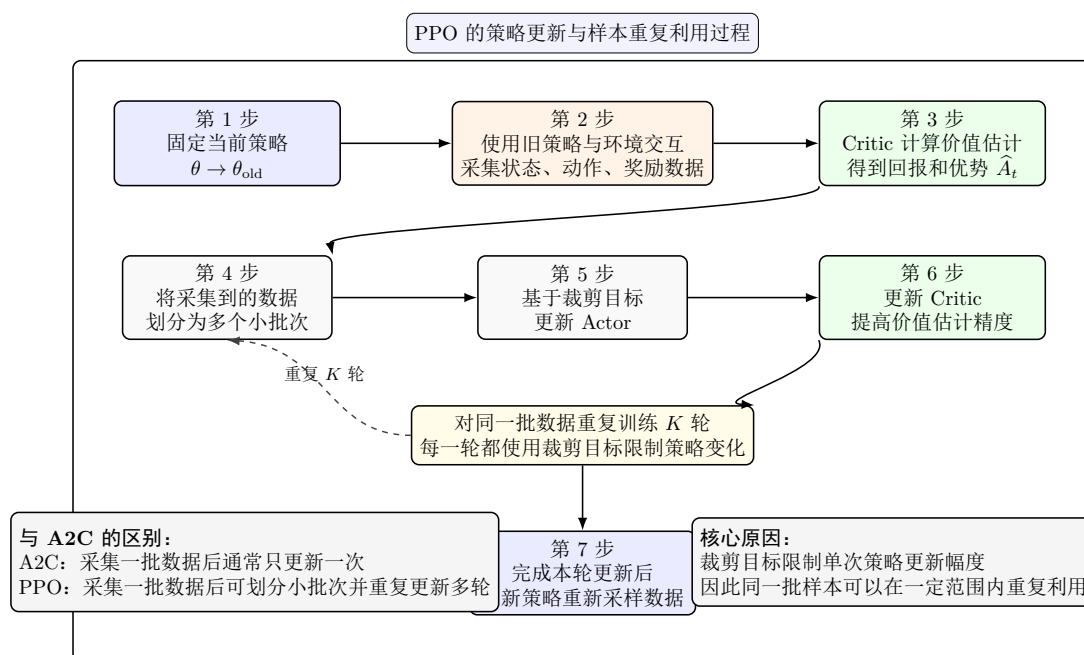


图 2.9: PPO 的一次采样与更新过程

实际实现通常还把策略目标、价值误差和熵正则合并训练。价值损失用于拟合回报或优势计算所需的基线，熵项用于延缓策略过早变得确定。三个分量的尺度不同，因此应分别记录，而不能只观察相加后的总损失。若价值误差持续增大，单靠缩小 PPO 的裁剪范围通常不能解决问题；若近似 KL 很大或大量样本触发裁剪，则说明同一批数据上的更新可能过多。

重点提示

PPO 的裁剪目标限制的是过大策略变化继续获得的优化收益，并不把新策略严格约束在某个概率区间，也不保证每次更新后性能都提高。

2.3.2 离线强化学习方法

在线强化学习可以用当前策略继续采样，离线强化学习则只使用预先收集并固定的数据集

$$\mathcal{D} = \{(s_i, a_i, r_i, s'_i, d_i)\}_{i=1}^N \quad (2.52)$$

其中 d_i 记录转移是否因自然终止而结束。数据可能来自人工操作、旧策略或多个不同版本的系统。训练期间不能让目标策略进入环境补充它最缺少的样本，这是离线学习与经验回放之间的关键区别。

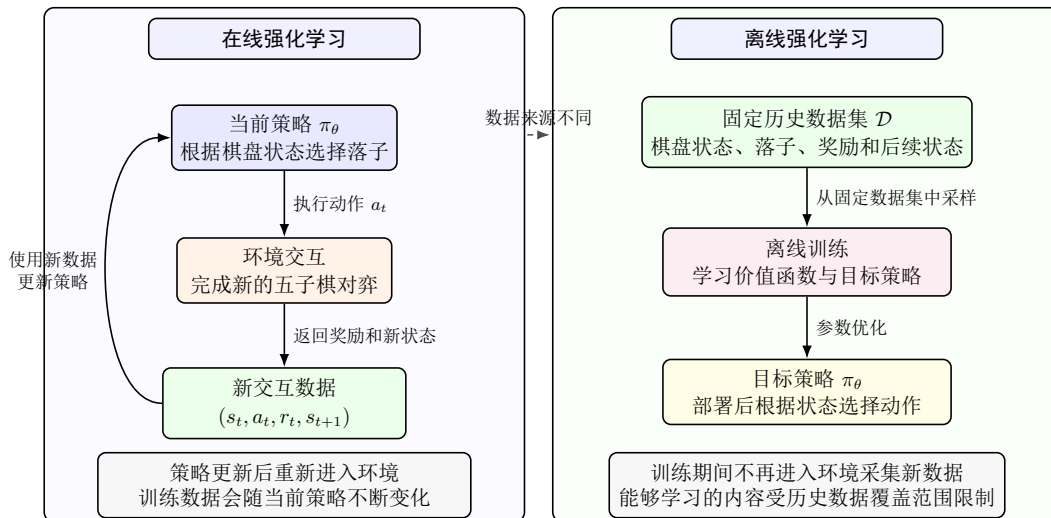


图 2.10: 在线强化学习与离线强化学习的数据来源

困难不只在数据有限，还在于学习得到的策略可能主动选择数据中罕见的动作。价值网络必须在这些位置外推，而最大化操作又偏好估计偏高的动作。一个偶然的高估会使策略更常选择该动作，但固定数据无法提供新的后果来纠正它，于是形成“分布偏移—价值高估—策略进一步偏移”的循环。

不同离线算法以不同方式限制这种外推。BCQ 和 BEAR 约束目标策略不要轻易离开行为数据覆盖的动作范围；CQL 对缺少数据支持却被估计得很高的动作施加保守惩罚；IQL 主要在数据中的动作上学习价值关系，再用优势加权方式拟合策略。这些方法共同承认一个事实：没有记录的动作不一定差，但现有数据不足以证明它好。约束越强，外推风险通常越小，同时也越难发现数据之外的改进。

方法选择以前，先要弄清数据由哪些策略产生、关键状态和动作是否出现、奖励规则是否一致，以及终止与截断是否被正确记录。同一条轨迹不应拆散到训练集和验证集两边，否则相邻转移会造成泄漏。行为克隆提供了重要基线：如果复杂离线算法甚至不能稳定超过对历史动作的模仿，就很难把性能提升归因于长期价值学习。

离线指标也有边界。保留数据上的 Bellman 误差较低，只说明模型能拟合选定的自举目标，不能直接证明新策略有效。离线策略评估、模拟器测试和重要性采样各自依赖额外假设；目标策略离行为数据越远，结论通常越不可靠。高风险任务仍需经过受控测试和可回退的小范围部署。

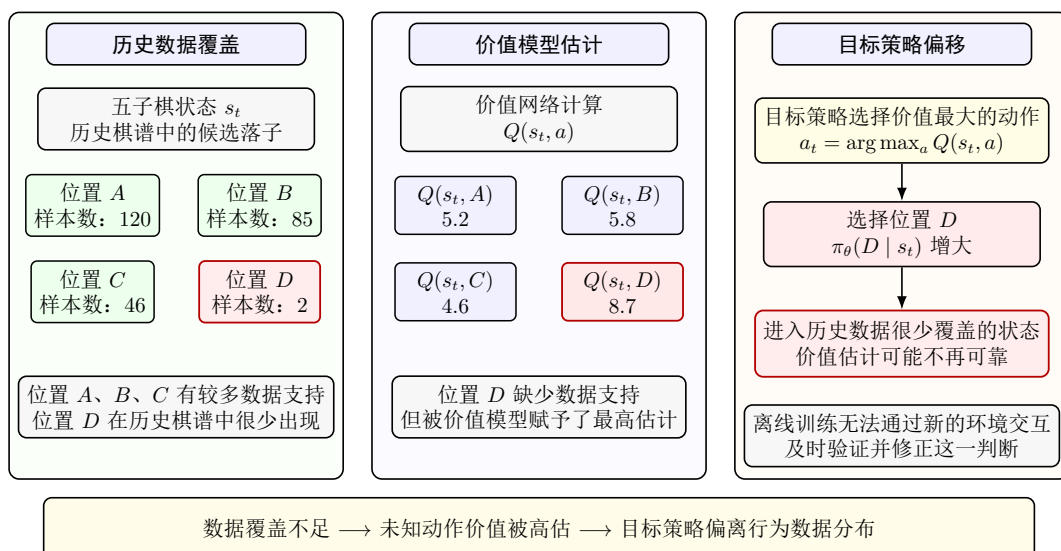


图 2.11: 离线强化学习中的分布偏移与价值高估

重点提示

离线强化学习的核心问题不是样本总数，而是目标策略的选择是否有数据支持。重复很多次相似轨迹，并不能补上从未出现过的关键状态和动作。

2.4 强化学习的应用设计方法

选定算法以前，任务定义已经决定模型能看到什么、能改变什么，以及什么结果会被当成成功。下面继续使用本章开头的五子棋设定，把这些决定落实为可以执行和评估的系统。

2.4.1 任务边界与转移定义

本章把智能体定义为执黑的一方，对手和规则属于环境。一次转移从黑方决策前开始：策略选择一个空位，环境检查合法性、落下黑子；若对局尚未结束，环境再执行白方回应，随后返回新的黑方决策局面。这样，数据中的 (s_t, a_t, r_t, s_{t+1}) 与算法使用的时间步保持一致。

如果改成黑白双方各自都是智能体，或者让同一个策略在每一个落子回合都被调用，状态中的行动方、奖励归属和轨迹结构都要随之改变。这并非实现细节，而是另一个任务定义。比较算法时应冻结环境版本、对手来源、开局分布和终止规则，否则性能差异无法只归因于算法。

最小规模实验很适合暴露边界错误。随机策略若能落到已占位置，问题在动作执行或掩码；相同局面得到互相矛盾的输入，通常意味着行动方或坐标编码不一致；训练回报上升而独立胜率不变，则应先核对奖励和评估目标。只有任务定义能够稳定地

产生正确轨迹，扩大训练规模才有意义。

2.4.2 状态表示与动作空间

环境状态不一定等于策略能够观察到的信息。本章的棋盘完全可见，且策略只在黑方回合调用，因此可以令 $o_t = B_t$ 。若同一模型控制双方，还需加入当前行动方 p_t ，否则同一棋盘在黑方和白方回合会得到相同输入。

棋盘可以用单个整数矩阵表示

$$B_t \in \{-1, 0, 1\}^{15 \times 15} \quad (2.53)$$

也可以使用两个二值通道分别记录双方棋子

$$B_t \in \{0, 1\}^{2 \times 15 \times 15} \quad (2.54)$$

前者紧凑，适合作为最小基线；后者把类别含义分开，更便于卷积网络处理。表示方式不应包含对局结束后才能得到的信息，归一化统计量也只能由训练数据计算。

动作空间对应棋盘的 225 个位置。策略网络可以为每个位置输出一个得分 $z_\theta(s_t, a)$ ，再用合法动作集合 $\mathcal{A}(s_t)$ 构造掩码后的分布

$$\pi_\theta(a \mid s_t) = \frac{\exp z_\theta(s_t, a)}{\sum_{a' \in \mathcal{A}(s_t)} \exp z_\theta(s_t, a')} \quad (2.55)$$

已经有棋子的格点由规则确定为非法，可以直接屏蔽；一手棋即使很差，只要符合规则就仍是合法动作。把棋力启发式写入掩码，会使策略失去发现其他解法的机会。

动作编码还要与环境坐标严格一致。把二维位置展平时，可以固定使用 $a = 15i + j$ ，并用边角位置和往返转换做单元测试。训练阶段若允许随机策略直接产生未掩码动作，环境也应给出明确错误，而不是静默地改成另一个位置。

2.4.3 奖励与约束

在固定设定中，胜、负、平分别得到 1、-1 和 0，非终止步骤奖励为 0。这种稀疏奖励与真实目标接近，却让早期落子的评价依赖整局结果。加入棋型分数等过程反馈能够降低学习难度，但也改变了算法实际优化的目标。

密集反馈缩短了动作与奖励之间的距离，也把设计者的棋型偏好写进了优化目标。若“形成活三”可重复计分，策略可能维持局部棋型来刷取奖励，而不是尽快获胜。过程奖励因此不是免费的训练加速器。

将任务奖励和多个辅助奖励组合时，可以写为：

$$r_t = r_t^{\text{task}} + \sum_{i=1}^K \lambda_i r_t^{(i)} \quad (2.56)$$

其中 r_t^{task} 是核心奖励, $r_t^{(i)}$ 是辅助项。判断 λ_i 不能只比较单步数值, 还要估算每项在一个回合中的累计上限和出现频率。

奖励权重的大小还会影响不同目标之间的取舍。例如, 机器人控制任务可能同时希望提高移动速度、降低能量消耗并保持动作平稳。总奖励可以由多个部分组成:

$$r_t = \lambda_1 r_t^{\text{progress}} - \lambda_2 r_t^{\text{energy}} - \lambda_3 r_t^{\text{instability}} \quad (2.57)$$

权重会改变策略的取舍: 能耗惩罚过大可能让机器人静止, 每步惩罚过大可能让处于劣势的棋手主动寻求快速失败。奖励尺度还影响价值目标和梯度数值。加入任何新项后, 都要分别记录各项回报以及独立任务指标; 总回报上升无法说明究竟是哪一项被优化。

约束是否进入奖励, 取决于它能否被权衡。机械极限、非法落子和不可突破的安全边界应用掩码、动作范围或安全控制器保证; 能耗、平稳性和延迟等允许折中的要求才适合写成软成本:

$$r'_t = r_t - \lambda c_t \quad (2.58)$$

其中 c_t 是成本, λ 表示任务愿意为降低成本牺牲多少核心回报。若需求方无法接受任何违反, 就不存在这个权衡, 也不应仅靠把 λ 调得很大来模拟硬约束。

巨大负奖励不能替代硬约束。随机策略仍可能先执行危险动作再收到惩罚, 而真实系统可能没有承受这次探索的余地。

奖励信号还可能被智能体以设计者没有预料到的方式利用。智能体只会优化实际给出的奖励, 而不会自动理解设计者的真实意图。当奖励函数存在漏洞时, 策略可能获得很高回报, 却没有完成预期任务。这种现象通常称为奖励投机或奖励漏洞。

例如, 如果只根据连续棋子数量奖励五子棋智能体, 策略可能偏好不断扩展已有棋型, 而忽略对手已经形成的直接威胁。如果机器人任务只奖励向目标方向的瞬时速度, 机器人可能通过剧烈摆动获得较高速度信号, 却没有稳定到达目标位置。

奖励漏洞要靠行为审计和独立指标发现。五子棋至少应检查真实胜率、先后手表现、不同对手下的结果及典型失败棋局。如果辅助奖励上升而胜率下降, 应回滚该奖励版本, 而不是继续调高训练步数。

新奖励项加入后, 应单独记录并做消融。除了比较总回报, 还要检查辅助项在整回合尺度上是否压过核心奖励, 以及高回报轨迹的行为是否符合预期。

奖励版本确定后, 还要建立彼此隔离的训练、验证和测试流程。

2.4.4 训练与评估过程设计

训练、验证和测试承担不同职责，混用会让结果失去解释性。

表 2.2: 训练、验证与最终测试的职责

阶段	可以做什么	不能做什么
训练	采集或读取训练数据、探索、更新参数、监控优化指标	用训练回报宣称最终任务性能
验证	比较检查点、选择超参数、触发早停、诊断失败	把反复调参后的验证结果当成无偏测试结果
最终测试	在冻结方案上报告预先定义的任务指标与波动	继续选模型、改奖励、调对手或回流测试数据

采样量决定梯度方差和数据新鲜度。只用一局五子棋更新，胜负偶然性可能支配梯度；等待过多对局，策略又长期使用旧数据。PPO 对同一批样本更新过多轮还会使当前策略偏离采样策略。批量大小、更新轮数和学习率必须联合报告，并结合 KL、裁剪比例、价值误差和策略熵诊断。

探索规则属于实验配置。始终取最大概率动作容易固化早期偏好，随机性过强又会掩盖已经学到的能力。训练可以采样并调节熵，验证和测试则要预先规定使用采样策略还是确定性策略；两种评估回答的问题不同，不能在看到结果后择优报告。

五子棋训练还需要设计对手来源。如果智能体始终与一个固定的弱对手进行比赛，它可能很快取得较高胜率，但这种胜率并不代表策略具有普遍能力。智能体可能只是发现了该对手的某些固定漏洞。

自我对弈是一种常见的数据生成方式。当前策略同时控制对弈双方，使对手能力随着训练同步提高。这样可以不断产生与当前策略水平接近的比赛，减少固定对手过强或过弱的问题。

单纯使用最新策略进行自我对弈也存在局限。如果新策略暂时形成某一种棋路，双方可能反复围绕这一棋路训练，逐渐遗忘对旧策略有效的行为。为增加对手多样性，可以定期保存历史策略，并从不同版本中随机选择对手。还可以加入规则策略、随机策略和外部棋谱策略，使训练数据覆盖不同水平和不同风格的对局。

对手池中的角色应预先定义：随机策略检查基本能力，规则策略提供稳定锚点，历史策略检查遗忘，当前策略提供相近难度。训练对手可以参与数据生成，测试对手不得因为测试结果而被加入当前模型的训练循环。

五子棋验证应使用独立种子、开局和对手，并交换先后手。只报合并胜率会掩盖先手优势，因此还应分别报告先手、后手及各对手上的结果。

设评估总局数为 N_{eval} ，获胜局数为 N_{win} ，则评估胜率可以写为：

$$\hat{p}_{\text{win}} = \frac{N_{\text{win}}}{N_{\text{eval}}} \quad (2.59)$$

训练指标用于解释优化过程：策略损失、价值误差、熵、KL、回合长度和各奖励分项都属于诊断信息。任务指标用于评价结果：五子棋的胜率、平局率和对手分项才回答棋力是否提高。两类曲线相关时也不能互相替代。

如果使用动作掩码，还应检查非法动作在掩码前是否获得过高得分。如果模型长期为大量非法位置分配较高原始得分，只是依靠掩码将其删除，说明策略网络对棋盘状态的理解可能仍然不足。掩码保证了动作合法，但不能代替模型学习合理的动作偏好。

网络初始化、环境随机性、动作采样和小批次顺序都会改变训练轨迹。应使用多个训练随机种子，在固定测试协议下报告均值与离散程度，不能从多次实验中只挑最好的一次。

检查点应包含策略、Critic、优化器、训练步数、归一化统计量和环境版本。验证集负责选择检查点和触发早停；最终测试只对冻结的候选方案运行。短期胜率波动不应触发即时早停，平滑窗口和耐心周期应在实验前设定。

在分析训练失败原因时，应区分任务设计问题和优化问题。如果策略损失剧烈变化、价值估计不断发散，问题可能来自学习率、更新频率或算法实现；如果训练过程稳定但策略始终选择无效行为，问题可能来自状态、动作或奖励设计；如果训练表现很好而独立评估较差，则需要检查训练场景是否过于单一。

训练集用于更新参数，验证集用于选择模型和调整设计，最终测试用于报告冻结方案。测试结果一旦反过来影响奖励、超参数、对手或检查点选择，这部分数据就已经参与了开发过程，需要重新建立未使用过的测试协议。

即使流程隔离正确，规模、训练波动和环境变化仍会带来新的困难。下一节从失败现象出发讨论这些问题。

2.5 强化学习方法的挑战

一条最终上升的训练曲线可能掩盖不同问题。有些重要局面从未进入训练数据，有些结果换一个随机种子便无法复现，还有些策略只适用于训练对手或当前奖励。下面从这些现象追查原因。

2.5.1 面向大规模决策的挑战

“任务规模大”可能指完全不同的瓶颈。高维状态首先造成覆盖困难，模型不得不对未见局面作出外推；动作候选太多时，计算和探索会分散在大量选择上；回合过长则拉开动作与结果的距离，使信用分配更加困难。扩大网络主要增加函数近似能力，并不会自动解决动作搜索或长时域问题。

继续以五子棋任务为例。一个 15×15 棋盘包含 225 个格点。若暂时不考虑棋局规则，每个格点可能为空、黑子或白子，棋盘状态数量的上界为：

$$|\mathcal{S}| \leq 3^{225} \quad (2.60)$$

实际合法棋局的数量远小于这一上界，但仍然十分庞大。智能体不可能遍历全部棋盘状态，也无法为每个状态单独保存一个精确价值。训练只能利用有限对局学习不同棋型之间的共同规律，并将已学到的判断推广到没有出现过的局面。

这说明，表格形式的价值函数只适合状态数量较少的任务。在大规模问题中，通常需要使用神经网络近似价值函数或策略函数：

$$V_{\phi}(s) \approx V^{\pi}(s) \quad (2.61)$$

$$\pi_{\theta}(a | s) \approx \pi(a | s) \quad (2.62)$$

神经网络能够利用相似状态之间的共享结构。例如，五子棋中的连续三子、两端开放和对手威胁等局部棋型，可能出现在棋盘的不同位置。卷积结构可以提取这些局部特征，使模型不必分别记忆每一个具体棋盘。

不过，函数近似并没有消除状态空间过大的问题。模型只能根据训练数据学习状态之间的相似性。如果某类棋型在训练中很少出现，网络对这些局面的判断仍可能不准确。随着状态维度增加，覆盖具有代表性的环境情况需要更多数据，模型训练和存储成本也会随之上升。

除了状态空间，动作空间也会带来明显挑战。五子棋在空棋盘状态下最多有 225 个候选落子位置，策略模型需要为这些位置分别给出动作得分。随着棋盘逐渐被填满，合法动作数量会减少，但训练早期仍需要在大量候选位置之间进行比较。

在推荐、调度和组合优化任务中，动作规模可能远大于五子棋。例如，推荐系统可能需要从大量候选物品中选择一个或多个结果，生产调度需要同时决定任务顺序、执行设备和开始时间。若将所有可能组合直接视为独立动作，动作数量会迅速超过模型能够逐一评价的范围。

假设每一步平均有 b 个候选动作，一段决策过程包含 H 步，则可能形成的动作序列数量约为：

$$N_{\text{seq}} \approx b^H \quad (2.63)$$

即使每一步的动作数量并不特别大，随着决策长度增加，完整动作序列仍会呈指数增长。智能体无法逐一尝试所有序列，只能根据有限经验判断哪些早期动作更可能带来较好的长期结果。

较大的动作空间还会增加探索难度。训练初期，策略对不同动作的判断较为接近，

有限的交互次数会被分散到大量候选动作上。许多动作只能获得少量样本，价值估计因而难以稳定。

在五子棋中，真正值得重点考虑的落子位置通常集中在已有棋子附近、己方棋型延伸位置和对对手威胁位置。如果策略在训练初期对所有空白格点进行近似均匀探索，大量对局会消耗在与当前棋局关系较小的位置上。

处理大规模动作空间的一种方法是先缩小候选范围，再由强化学习策略进行精细选择。候选集合可以由任务规则、启发式方法或单独训练的候选生成模型产生。设原始动作空间为 $\mathcal{A}$ ，候选生成过程得到状态相关集合：

$$\mathcal{A}_{\text{cand}}(s_t) \subseteq \mathcal{A} \quad (2.64)$$

策略只需要在候选集合中计算动作概率：

$$\pi_{\theta}(a \mid s_t, a \in \mathcal{A}_{\text{cand}}(s_t)) \quad (2.65)$$

这种方式可以显著降低单次决策的计算量，但候选生成过程本身也可能引入偏差。如果真正的最优动作被提前排除，后续策略无论如何训练都无法选择它。因此，候选集合既要足够小，又需要保留较高的有效动作覆盖率。

除了候选筛选，还可以将复杂动作拆分为多个较小决策。例如，在调度任务中，可以先选择待处理任务，再选择执行设备，最后确定开始时间。原本需要一次完成的组合动作被分解为多个阶段，每个阶段只处理其中一部分选择。

分解动作能够降低单次输出规模，但也会延长决策过程，并使后续选择依赖前面已经作出的决定。如果早期阶段选错了任务，后续阶段即使选择合理，也难以得到好的最终结果。因此，动作分解需要在单步决策难度和整体序列长度之间进行权衡。

分层强化学习采用相近思路，将长任务划分为不同层次。高层策略负责选择阶段目标，低层策略负责完成具体操作。例如，机器人导航可以先由高层策略选择需要到达的区域，再由低层策略控制移动方向和速度。这样可以减少高层策略需要处理的决策步数，也使低层策略能够重复用于不同目标。

长决策序列本身也是大规模任务的重要难点。在五子棋中，一步失误可能要经过多轮对弈后才表现为最终失败。随着回合长度增加，早期动作和最终奖励之间的距离越来越远，智能体更难判断每一步对结果的影响。

如果回合长度为 T ，时刻 t 的动作需要考虑之后的全部奖励：

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \quad (2.66)$$

当 T 较大时，回报会受到许多后续随机事件影响。同一个早期动作在不同轨迹中可能得到差异较大的回报，使价值估计和策略梯度具有较高方差。

折扣因子 γ 可以降低远期奖励对当前动作的影响，但过小的折扣因子会使智能体过度关注短期收益。在五子棋中，如果策略只关注少数几步后的局面，可能会偏好立即形成局部棋型，却忽略更长时间后的防守风险。因此，长序列任务不能简单依靠减小折扣因子解决，还需要更准确的价值估计、合理的过程反馈或分层任务结构。

大规模问题还会提高环境交互成本。状态和动作覆盖不足时，通常需要采集更多轨迹；模型规模增加后，每次前向计算和参数更新也会消耗更多资源。在仿真环境中，可以通过并行运行多个环境提高数据采集速度：

$$\mathcal{D} = \bigcup_{i=1}^M \mathcal{D}_i \quad (2.67)$$

其中， M 表示并行环境数量， $\mathcal{D}_i$ 表示第 i 个环境采集的数据。

并行采样能够缩短收集相同数量数据所需的时间，但不能减少任务本身的样本需求。在真实机器人、工业设备和人工交互任务中，同时运行大量环境往往并不现实。此时可以利用历史数据、模拟环境或已有策略减少真实交互次数。

模拟环境能够低成本产生大量数据，但模拟过程与真实环境之间可能存在差异。若策略过度依赖模拟环境中特有的状态特征或动态规律，部署到真实系统后可能无法保持原有性能。因此，模拟训练通常还需要配合环境随机化、少量真实数据调整和独立测试。

在大规模任务中，课程学习也可以降低初始训练难度。课程学习先让智能体处理较简单的状态和任务，再逐渐增加环境复杂度。例如，五子棋训练可以先在较小棋盘或预设中期局面中学习基本进攻和防守，再逐步过渡到完整棋盘和完整对局。

课程设置必须保持前后任务之间的连续性。如果早期任务与最终任务差异过大，智能体在简单环境中学到的行为可能无法迁移到复杂环境。课程学习的作用是调整学习顺序，而不是替代对最终任务的训练。

这些方法都在改变困难的位置，而不是消灭困难。函数近似把表格存储变成分布外预测；候选筛选用较小计算量换取漏掉关键动作的风险；动作分解降低单步输出规模，却增加阶段依赖；模拟器扩大数据量，也引入模拟—现实偏差；课程学习降低早期难度，但课程断层会阻止能力迁移。

选择扩展方法时，要先找到实际瓶颈。候选动作方法需要报告候选召回率，函数近似需要检查分布外状态，动作分解则要评估早期错误能否恢复。并行采样能够缩短训练时间，却没有改变任务所需的样本数量。

规模问题解决后，仍需区分“同样条件能否复现”和“条件变化后是否有效”。

2.5.2 强化学习训练的不稳定性和泛化性问题

训练数据由策略产生，Critic又用这些数据评价策略，于是形成反馈环：策略改变访问分布，访问分布改变价值估计，价值估计再改变策略。早期一次偶然胜局因此

可能被后续自我对弈反复放大。

本节区分三个容易混用的概念。

表 2.3: 稳定性、泛化性与鲁棒性的区别

性质	问题	检查方式
训练稳定性	相同配置重复训练，结果是否接近？训练中是否突然退化？	多随机种子、学习曲线、梯度和价值诊断
泛化性	规则不变但状态、初始条件或对手未见时，性能是否保持？	冻结策略，在预先划分的未见场景测试
鲁棒性	观测噪声、参数扰动或有限对抗变化下，性能是否平稳退化？	分级扰动和压力测试

训练曲线平滑只提供第一类问题的部分证据，不能证明后两类性质。

图中的波动可以从四个位置诊断：样本是否过少而放大偶然轨迹；Actor是否移动过快；Critic 是否在变化的数据上产生系统偏差；探索是否过早消失或长期过强。它们可能同时发生，不能仅凭“回报下降”判断超参数方向。

在 Actor-Critic 框架中，时序差分误差为：

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \quad (2.68)$$

TD 目标中的当前价值与下一状态价值都来自正在变化的 Critic，下一状态误差会通过 bootstrap 向前传播。Actor 过快会让 Critic 追逐不断移动的数据分布，Critic 在小批次上过拟合又会向 Actor 提供过度自信的优势。PPO 裁剪只缓解策略目标的一部分，不能修正错误价值目标。

诊断时应成组观察采样规模、近似 KL、裁剪比例、策略熵、解释方差、价值误差和梯度范数。单独把学习率调小可能只是让错误方向走得更慢。梯度裁剪可以拦住个别异常更新：

$$g \leftarrow g \cdot \min \left(1, \frac{c}{\|g\|} \right) \quad (2.69)$$

其中， g 表示当前梯度， c 表示允许的最大梯度范数。梯度裁剪可以避免个别异常样本造成过大的参数更新，但不能解决价值目标错误或奖励设计不合理等根本问题。

多次重复实验也是判断训练稳定性的重要手段。设使用 M 个随机种子得到的评估结果分别为 $J_1, J_2, \dots, J_M$ ，可以计算平均性能：

$$\bar{J} = \frac{1}{M} \sum_{i=1}^M J_i \quad (2.70)$$

同时还应报告不同实验之间的波动。如果一种方法只有少数实验能够取得较高性能，而多数实验表现较差，仅报告最好结果会高估方法的可靠性。

多随机种子均值与离散程度用于回答训练能否复现，仍不能回答未见场景是否有效。

在五子棋任务中，如果智能体始终与同一个规则对手进行训练，它可能学会专门利用该对手的固定弱点。当评估对手改变棋路时，策略的胜率可能迅速下降。此时，训练过程本身可能十分稳定，训练胜率也持续提高，但模型学习到的是针对单一对手的行为，而不是普遍适用的棋局判断能力。

泛化问题常与训练数据覆盖范围有关。如果训练过程中只出现少量初始状态，策略便可能记住这些状态下的有效动作，而没有学会更一般的决策规律。五子棋若始终从空棋盘开始，并且对手开局方式十分固定，模型可能只熟悉少数常见开局。

为了检查这种情况，可以在评估时使用不同的开局状态，例如随机放置少量合法棋子后开始比赛，或从历史棋谱的中间局面继续对弈。若策略只能在标准空棋盘开局中取得较好结果，说明其局面适应能力仍然有限。

环境中的无关特征也可能被策略错误利用。假设训练环境中某类棋盘显示方式总是与某个对手绑定，模型可能根据显示差异识别对手，而不是根据棋盘局面判断动作。当显示方式改变后，性能便会下降。

在机器人和自动驾驶任务中，类似问题可能来自光照、背景、传感器噪声或模拟环境中的固定纹理。策略可能依赖这些与任务目标无关的特征，导致环境外观稍有变化时无法正常决策。

提高泛化能力的一种方法是增加训练环境的多样性。五子棋可以使用不同水平、不同棋风和不同历史版本的策略组成对手池，使智能体不能只依赖某个对手的固定行为。训练中还可以随机交换先后手、改变初始棋盘状态，并加入不同类型的规则策略。

对于连续控制任务，可以在合理范围内随机改变物体质量、摩擦系数、传感器误差和环境布局。策略只有在多种条件下都取得较好回报，才更可能学到稳定的控制规律。这种做法通常称为环境随机化或域随机化。

状态数据增强也可以利用任务中的对称结构。五子棋棋盘经过旋转或翻转后，棋局的基本规则保持不变。训练时可以对同一棋盘进行旋转和镜像变换，并对动作位置作出相应调整，使模型接触更多等价局面。

设状态变换为 T_s ，对应的动作变换为 T_a 。如果变换不改变任务含义，则策略应当满足近似一致性：

$$\pi_{\theta}(T_a(a) \mid T_s(s)) \approx \pi_{\theta}(a \mid s) \quad (2.71)$$

这种一致性要求可以通过数据增强自然学习，也可以通过额外损失进行约束。它能够减少模型对棋盘绝对方向的依赖，提高对等价局面的利用效率。

模型结构也会影响泛化能力。与任务结构相匹配的网络通常更容易学习可复用特

征。五子棋使用卷积网络能够在不同棋盘位置共享局部棋型检测参数，比为每个格点独立设置大量参数更有利于位置泛化。

正则化方法可以减少模型对训练数据的过度拟合，例如限制网络规模、使用权重衰减或在状态输入中加入适量噪声。不过，强化学习中的泛化问题不能只通过监督学习中的正则化解决，因为训练数据分布还取决于策略本身。更重要的是扩大训练场景和评估场景的覆盖范围。

评估变化需要分级，否则无法解释性能下降。第一层保持训练分布，检查任务是否学会；第二层保持规则不变，只更换合法开局、随机种子和未见对手，检查泛化；第三层加入观测噪声或合理参数扰动，检查鲁棒性。规则和目标都发生改变时，测到的是任务迁移，不应继续称为同一任务的泛化。

五子棋应分别测试训练对手、冻结的未见对手、合法中间局面、先后手和旋转翻转等价局面。失败类型决定修正方向：未见对手下降指向对手覆盖，中间局面下降指向状态覆盖，对称变换下降则说明模型没有学到规则本身具有的对称性。

稳定与泛化并不总是同步。单一训练分布可能让策略平滑地收敛到一个对手特化解；训练波动较大时，不同检查点又可能表现出不同的泛化结果。因此，多种子重复训练回答“能否复现”，未见场景测试回答“能否迁移到同一任务的新条件”，扰动测试回答“性能怎样退化”。一条平滑曲线不能代替后两种实验。

最后一种失败更隐蔽：训练可复现、未见场景也表现稳定，但策略优化的根本不是设计者真正关心的目标。

2.5.3 强化学习方法的奖励构建问题

前文讨论了怎样设置奖励，这里进一步看一种更棘手的情况：奖励已经可以计算，训练也能收敛，结果仍然可能不符合预期。训练回报上升而真实指标不变，往往说明可计算奖励只是一个不完整的代理目标；终局结果明确而早期动作更新混乱，问题来自反馈延迟；策略反复利用某条规则刷取高分，则属于奖励投机。任务同时追求速度、安全和能耗时，困难又不只是计算奖励，而是这些目标之间本来就需要人为取舍。

若可计算奖励为 R ，算法求解的是

$$\pi_R^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}) \right] \quad (2.72)$$

这定义了奖励最优策略 π_R^* 。设计者真正关心的任务效用若记为 U ，一般没有理由保证最大化 R 同时最大化 U 。五子棋胜率接近真实目标，但活三数量只是代理指标；暂时放弃局部棋型可能反而是长期好棋。代理指标越容易反复获取，错位越容易被优化放大。

当奖励只在任务结束时给出时，还会出现明显的信用分配问题。信用分配关注的是最终结果应当归因于此此前哪些动作。五子棋智能体在一局对弈结束后得到了失败奖

励，但这次失败可能来自最后一步没有阻挡，也可能来自更早之前破坏了自身棋型，还可能是连续多步选择共同造成的结果。

设一条轨迹包含动作序列：

$$a_0, a_1, \dots, a_{T-1} \quad (2.73)$$

如果只有终止时刻得到奖励 r_{T-1} ，那么较早动作与最终奖励之间相隔较远。强化学习需要依靠回报和价值估计，将最终结果逐步传递到此前状态：

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k \quad (2.74)$$

当决策序列较长时，早期动作的回报会受到大量后续动作和环境变化影响。同一个动作在不同轨迹中可能获得完全不同的结果，使模型难以准确判断它对最终目标的贡献。

过程奖励缩短信号延迟，也把人工判断写进目标。为“形成活三”加分无法表达“此刻必须先防守”的局面依赖。更详细的规则未必更准确，只可能让策略更快学会设计者已经写出的启发式偏好。加入塑形奖励后必须重新检查最优行为是否发生变化。

奖励投机比一般的目标近似误差更具体：策略找到了一条规则允许、但设计者没有预料到的高回报路径。同一棋型被重复计分、机器人靠摆动刷取瞬时速度、推荐系统靠诱导性内容刷点击都属于这种情况。测试不能只验证“预期好行为能否得高分”，还要反向搜索“还有哪些行为能得到同样高分”。

复杂任务通常还包含多个目标。例如，一个控制系统既需要完成任务，又需要降低能耗、减少时间并保证安全。常见方法是将多个奖励项进行加权组合：

$$r_t = \sum_{i=1}^K \lambda_i r_t^{(i)} \quad (2.75)$$

其中 λ_i 不是纯技术超参数，而是对速度、能耗、安全等目标作出的偏好声明。完成更快但能耗更高，与更慢但更安全的策略之间可能不存在唯一最优解。调权重不能替代需求方作出取舍，而且相同权重也不代表相同影响：各项的尺度和出现频率不同。多智能体协作还会叠加个体贡献分配问题，本章不在此展开。

难以手写奖励时，可以借助专家示范、逆强化学习或成对偏好。它们改变的是监督来源，不会消除目标错位：示范只覆盖专家到达的状态，评价者可能存在分歧，学习到的奖励模型仍会在训练分布外外推。策略持续优化这个模型时，同样可能发现模型漏洞。

无论奖励来自规则还是学习模型，都要保留与它不同源的评估信号。五子棋用独立对局胜率和失败棋局审计；控制任务可以使用任务完成率、安全事件和资源消耗。如果训练回报提高而这些指标不变，继续增加训练时间只会更充分地优化错误目标。

奖励审查应以轨迹为单位。分别抽取高回报、低回报、指标冲突和约束边界附近的轨迹，检查策略怎样获得每一项分数；再做奖励消融和权重敏感性实验，观察行为是否发生预期变化。只看奖励公式和均值曲线，无法发现策略实际利用的路径。

高回报只说明策略善于优化当前奖励。代理指标是否错位、反馈是否过度延迟、规则是否存在可利用路径，还需要通过独立指标、轨迹审计和消融实验判断。

几类挑战会相互放大：覆盖不足使奖励模型在数据外失真，训练波动改变策略利用奖励的方式，单一环境又让这些问题在训练指标中隐藏起来。因此，算法诊断、奖励审计和独立评估不能彼此替代。

2.6 本章小结

强化学习关注的是带有反馈和时间关联的序列决策：当前动作不仅产生即时结果，还会改变后续状态、数据分布和可选动作。从早期的控制、调度和博弈，到深度强化学习处理高维状态，再到大模型利用人类偏好或可验证结果进行后训练，应用对象不断变化，但核心问题始终是怎样用有限反馈改进长期行为。强化学习在大模型时代是连接已有能力与目标行为的重要工具，却不能替代预训练、监督微调和可靠评估。

五子棋把这一问题具体化：程序在终局才知道输赢，它怎样评价几十步以前的落子？回报把当前动作与未来结果联系起来，价值函数把这种联系写成可以递推估计的形式，策略梯度则直接改变动作概率。Actor-Critic用正在学习的价值模型提供更及时的评价，因此减小了蒙特卡洛回报的部分波动，也把 Critic 的偏差带进了策略更新。PPO 试图让这种更新不要一次走得太远，但裁剪不是稳定性的保证。

当训练只能依靠历史数据时，问题从“怎样探索”变成“哪些判断有数据支持”。离线强化学习可以在行为数据之上改进策略，却无法现场验证陌生动作。进入具体任务后，算法只是整个系统的一部分：观测是否充分、动作能否执行、奖励是否代表真实目标，以及测试数据是否独立，都会改变最终结论。

因此，训练曲线上升之后仍要回到任务本身：策略究竟依据了哪些信息，数据怎样随策略改变，最后报告的指标是否真的对应任务目标。只说明实验采用了哪一种算法，还不足以回答这些问题。

训练回报高只说明当前奖励被有效优化；PPO 裁剪不保证训练稳定；离线数据量大不等于关键动作得到覆盖；输入更多信息也可能造成泄漏；密集奖励未必更接近真实目标；最后一个检查点未必优于验证集选出的检查点。这些反例指向同一个原则：算法输出只能在任务定义、数据来源和评估协议给定的范围内解释。

练习

课后练习

1. **基础 · 价值递推** 设智能体从时刻 t 开始获得的折扣回报为

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$$

请先证明

$$G_t = r_t + \gamma G_{t+1}$$

再根据这一递推关系，写出策略 π 下动作价值函数 $Q^\pi(s_t, a_t)$ 的贝尔曼期望方程，并说明最优动作价值函数中为什么会出现 $\max_{a'} Q^*(s_{t+1}, a')$ 。

2. **基础 · Actor-Critic** 在某一五子棋状态 s_t 下，Actor 选择位置 B 落子，且该转移没有自然终止。已知即时奖励为 $r_t = 0$ ，折扣因子为 $\gamma = 0.9$ ，Critic 对落子前后状态的价值估计分别为

$$V_\phi(s_t) = 3.5, \quad V_\phi(s_{t+1}) = 5.0$$

计算时序差分目标 y_t 和时序差分误差 δ_t 。根据计算结果，说明 Actor 应提高还是降低位置 B 的选择概率，并说明 Critic 应如何调整对 $V_\phi(s_t)$ 的估计。

3. **进阶 · PPO 裁剪** 在一次 PPO 更新中，旧策略选择动作 a_t 的概率为 0.40，裁剪阈值为 $\epsilon = 0.2$ 。分别考虑以下两种情况：

$$\text{情况一: } \hat{A}_t = 2, \quad \pi_\theta(a_t | s_t) = 0.56$$

$$\text{情况二: } \hat{A}_t = -3, \quad \pi_\theta(a_t | s_t) = 0.20$$

对每种情况计算概率比 $\rho_t(\theta)$ 、裁剪后的概率比，以及 PPO 裁剪目标中的两个候选项。判断最终采用哪一项，并解释裁剪机制如何限制策略发生过大变化。

4. **进阶 · 离线强化学习** 判断下列说法是否正确，并简要说明理由：“离线强化学习使用固定历史数据进行训练，因此只要数据量足够大，就可以像在线强化学习一样放心地选择数据集中从未出现过的动作。”

回答时应说明行为策略与目标策略的区别，并结合分布偏移、外推误差和保守价值估计解释离线强化学习为什么需要对未知动作保持谨慎。

5. **设计 · 任务建模** 现需要设计一个仓库移动机器人强化学习任务。机器人需

要从起点到达指定货架，同时尽量缩短时间、降低能耗，并避免与障碍物发生碰撞。请完成以下设计：

- (a) 给出一种状态表示和动作空间
- (b) 区分任务中的核心奖励、辅助奖励和惩罚项
- (c) 判断“不能穿过障碍物”和“尽量减少能耗”分别应采用硬约束还是软约束
- (d) 说明至少两个用于独立评估策略性能的指标

并简要分析奖励权重设置不合理时，机器人可能出现的非预期行为。

6. **设计·实验诊断** 某五子棋智能体在训练对手上的胜率达到 90%，但更换为未参与训练的规则策略后，胜率下降到 45%。同时，使用不同随机种子重复训练时，最终胜率波动较大。请分别从训练不稳定性和泛化性两个方面分析可能原因，并提出改进方案。

回答中至少应涉及以下内容中的四项：数据采样规模、Actor 与 Critic 的更新速度、策略熵、对手池、随机开局、棋盘旋转与翻转增强、多随机种子评估、独立测试环境。

第三章 大模型预训练与微调方法

作者：吴钰璋、王骏鑫、王成龙

参考：Foundations of Large Language Models

近年来，大语言模型（LLMs）的蓬勃发展成为自然语言处理（NLP）领域最重要的进展之一。这一进展推进生成了可模拟人类认知、实现自然语言理解与生成的系统。这些系统甚至具备逻辑推理能力，而推理任务曾一直被视为人工智能领域极具挑战性的难题。得益于这些技术革新，NLP 取得了长足的进步，正式步入全新研究阶段，以往诸多研究难题，例如搭建可与人流畅沟通的对话交互系统等，正逐步迎刃而解。

在大规模数据上训练语言模型已将 NLP 研究带入了一个全新发展的阶段。长期以来，语言建模仅被视为底层基础技术，与通用人工智能的研发目标相差甚远，但如今该方向催生出具备涌现智能的系统：模型通过持续预测文本序列中的词汇，学习到一定程度的通用知识。研究表明，一个经过良好训练的单一 LLM 能够处理海量任务，并以极低的适配成本泛化至全新任务 [?]。这一成果代表着人工智能向高阶通用形态迈出关键一步，也推动学界进一步探索，致力于将更强大的语言模型开发为基础模型。

本章将系统介绍大语言模型的预训练与微调全流程技术体系。首先，我们将阐述大语言模型的基础预训练技术，包括语言建模方法、主流模型架构、预训练策略以及指导模型规模扩张的缩放定律。在此基础上，展开提示学习相关方法研究，包含零样本、少样本提示、思维链提示及其各类优化手段。随后，重点阐述大模型微调技术，包括有监督微调和各类高效微调方法。进一步，讨论模型偏好对齐技术，涉及奖励模型构建、对齐策略、直接偏好优化及偏好数据自动生成方案。最后，探讨大语言模型的推理增强方法，包括测试时缩放策略、基于可校验奖励的强化学习以及从推理模型到智能体的演进路径。通过对上述内容的系统梳理，本章旨在为读者构建大模型从预训练到应用部署的完整技术认知体系。

学习目标

- 理解大语言模型预训练的基本原理，包括语言建模的目标、模型架构以及模型规模与训练数据之间的扩展规律，能够解释模型性能与计算资源之间的关系。
- 掌握通过提示设计和有监督微调来适配预训练模型的方法，能够区分不同提示策略和高效微调技术的适用场景。
- 了解模型偏好对齐与推理增强的核心思路，能够说明如何利用人类反馈优化模型行为，以及如何通过扩展推理时的计算来提升复杂任务的解决能力。

3.1 大模型预训练技术

3.1.1 语言建模方法

在引入严谨的定义之前，我们不妨先通过一个例子来理解语言建模在做什么。想象你正在读一句话的前半部分：“今天天气真”。即便后续尚未给出，你大概已经能猜到接下来会是“好”“不错”或“热”之类的词，而几乎不可能是“宪法”“光合作用”或“平方根”。语言模型本质上就是在做这件“猜词”的事：给定上文，它估计每个候选词出现的概率，概率高的就是“合理”的续写，概率低的则不太可能出现在当前上下文之后。

更形式化地来看，语言建模的目标是计算一个 token 序列整体出现的概率。设 $\{x_0, x_1, \dots, x_m\}$ 为一个 token 序列，其中 x_0 为起始符号 $\langle s \rangle$ （或记为 $\langle \text{SOS} \rangle$ ）¹。根据链式法则，该序列的联合概率可分解为一系列条件概率的乘积：

$$\Pr(x_0, \dots, x_m) = \prod_{i=0}^m \Pr(x_i | x_0, \dots, x_{i-1}) \quad (3.1)$$

这一定义的直觉很简单：先把开头 x_0 出现的概率算出来，再算“有了 x_0 之后出现 x_1 的概率”，接着算“在 x_0 和 x_1 已经有了的前提下出现 x_2 的概率”……依此类推，整个句子的概率就是每一步条件概率的乘积²。

在深度学习时代，语言建模的一种典型方法是用深度神经网络来估计这些条件概率：模型接收历史 token 序列 $x_0, \dots, x_{i-1}$ 作为输入，输出词汇表 $\mathcal{V}$ 上的概率分布（记为 $\Pr(\cdot | x_0, \dots, x_{i-1})$ ）。其中， $\Pr(x_i | x_0, \dots, x_{i-1})$ 即为该分布中对应 token x_i 的概率值。

有了训练好的语言模型后，最常见的操作是根据已有上下文预测下一个最可能出现的 token：

$$\hat{x}_i = \arg \max_{x_i \in \mathcal{V}} \Pr(x_i | x_0, \dots, x_{i-1}) \quad (3.2)$$

将这一过程反复迭代，就构成了自回归生成：每次选出当前最优的下一个 token，把它拼到上下文中，再以更新后的完整序列预测再下一个 token，如此递推。这一机制恰好对应了公式 (3.1) 中链式法则的逐项展开。

为了直观说明上述过程，考虑一个简单的续写场景。假设模型已见过前缀 $\langle s \rangle$ The（起始符后的第一个词是“The”），接下来需要生成三个词。每一步的决策如下：

经过这三步，模型生成了序列 The cat sat on，而整句“The cat sat on”的联合概率正是各步条件概率的乘积。当然，基于同样的前缀，模型也可能在第一步选中 dog、第二步选中 ran，从而生成另一条合理的续写路径——这正是语言模型作为一个概率系统的灵活之处。

接下来，我们将深入探讨 LLM 的构建、训练与应用。

¹在 BERT 模型中，起始符号也可以是 [CLS]。

²为处理 $i = 0$ 的边界情况，约定 $\Pr(x_0 | x_0, \dots, x_{i-1})|_{i=0} = \Pr(x_0) = 1$ ，因此 $\Pr(x_0, \dots, x_m) = \Pr(x_1, \dots, x_m | x_0)$ 。

当前上下文	预测	解读
$\langle s \rangle$ The	cat	模型评估 $\Pr(\cdot \langle s \rangle \text{ The})$ 分布后, cat 得分最高——“The” 后面常接名词。
$\langle s \rangle$ The cat	sat	给定 “The cat”, 模型判断最合理的下一个词是动词 sat。
$\langle s \rangle$ The cat sat	on	有了 “The cat sat” 后, on 成为最高概率的续写 (“sat on” 是常见搭配)。

表 3.1: 语言模型基于前缀 $\langle s \rangle$ The 依次生成 cat、sat、on 的自回归过程。每一步中, 模型根据当前上下文计算词汇表上的条件概率分布, 并选出概率最大的词追加到上下文中; 更新后的序列随即成为下一轮预测的输入。每一步的决策对应于链式分解中新增的一个条件概率因子。

3.1.2 大语言模型架构

上一节从概率的角度介绍了语言建模的目标。本节我们来回答一个更具体的问题: 什么样的神经网络架构能够实现这一目标?

图 3.1 展示了当前主流 LLM 所采用的仅解码器 (Decoder-only) Transformer 架构。尽管不同 LLM 在细节上各有所异, 但它们共享同一个信息处理流水线:

1. **嵌入:** 输入 token 序列 $\{x_0, \dots, x_{m-1}\}$ 被转换为一组 d 维向量 $\{\mathbf{e}_0, \dots, \mathbf{e}_{m-1}\}$, 每个向量是 token 嵌入与位置嵌入之和。
2. **Transformer 块堆叠:** 这些向量依次通过 L 个结构相同的 Transformer 块。每经过一个块, 序列中每个位置的表示都会吸收更多来自上下文的语义信息。记第 ℓ 个块的输出为 $\mathbf{H}^\ell \in \mathbb{R}^{m \times d}$, 其第 i 行就是位置 i 在当前深度下的上下文表示。
3. **输出层:** 最后一个块的输出 $\mathbf{H}^L$ 通过一个 Softmax 层映射为词汇表上的概率分布:

$$\begin{bmatrix} \Pr(\cdot | x_0, \dots, x_{m-1}) \\ \vdots \\ \Pr(\cdot | x_0, x_1) \\ \Pr(\cdot | x_0) \end{bmatrix} = \text{Softmax}(\mathbf{H}^L \mathbf{W}^o) \quad (3.3)$$

其中 $\mathbf{W}^o \in \mathbb{R}^{d \times |V|}$ 为输出投影矩阵。这恰好就是公式 (3.1) 中所需的每一步条件概率分布。

每个 Transformer 块由两个核心子层构成:

- **掩码多头自注意力:** 让每个 token 能够聚合其前方所有 token 的信息。“掩码”

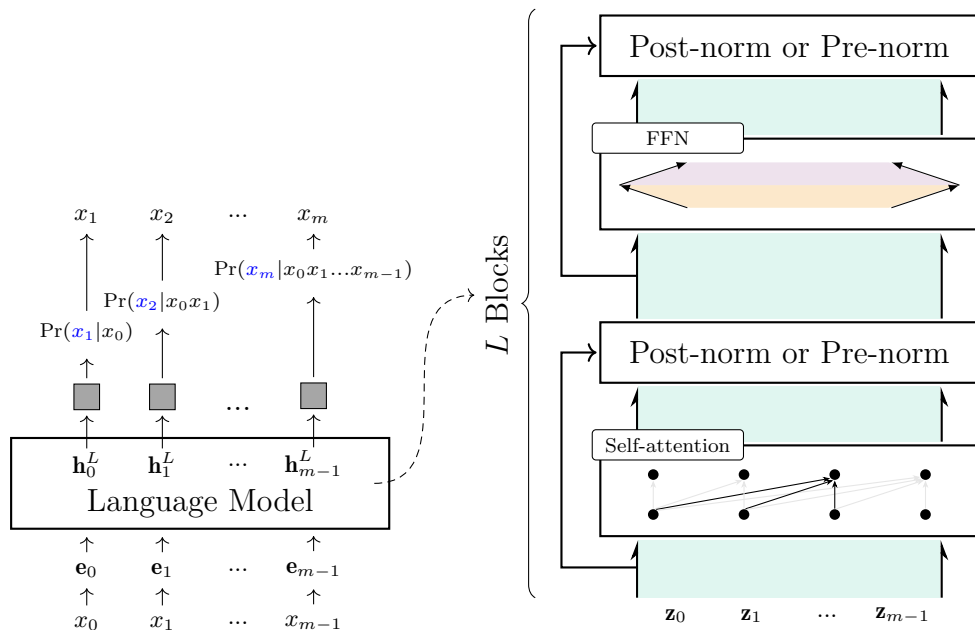


图 3.1: 用于语言建模的 Transformer 解码器架构。核心组件为 L 个堆叠的 Transformer 块，每个块包含一个掩码多头自注意力子层和一个 FFN 子层。输出层将最后一个块的上下文表示映射为下一个 token 的概率分布。

确保位置 i 只能看到位置 0 到 i 的内容——这正是条件概率 $\Pr(x_i | x_0, \dots, x_{i-1})$ 的“只看上文”要求。

- **前馈神经网络 (FFN)**: 对每个位置的表示独立做非线性变换，为模型提供逐 token 的细粒度加工能力。

两个子层均配备**残差连接**：子层输出与其输入相加后再传递至下一阶段。这一设计使得信号即使在很深的网络中也能顺畅流动，是 LLM 能够堆叠数十乃至上百层的关键。

自注意力层是整个架构的精髓。其核心直觉是：对每个位置 i ，模型生成一个查询向量 (Query)，用它去“询问”所有前方位置 $k \leq i$ ——每个前方位置用一个键向量 (Key) 来回应，匹配度越高，该位置的值向量 (Value) 在最终聚合中的权重就越大。形式化地：

$$\text{Att}_{\text{qkv}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} + \mathbf{Mask}\right) \mathbf{V} \quad (3.4)$$

其中 $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{m \times d}$ ， $\mathbf{Mask}$ 确保 $i < k$ 时注意力权重为 0，除以 $\sqrt{d}$ 则用于稳定训练。

为了使模型能同时关注不同类型的模式（例如一个头关注句法，另一个头关注语义），实际使用的是**多头注意力**：将 $\mathbf{Q}$ 、 $\mathbf{K}$ 、 $\mathbf{V}$ 投影到 τ 个不同的低维子空间中独立

LLM	# of Params	Depth L	Width d	# of Heads (Q/KV)
GPT-2 [?]	1.5B	48	1,600	25/25
GPT-3 [?]	175B	96	12,288	96/96
LLaMA3.1 [?]	8B	32	4,096	32/8
	70B	80	8,192	64/8
	405B	126	16,384	128/8
Qwen3 [?]	0.6B	28	1,024	16/8
	8B	36	4,096	32/8
	235B [†]	94	4,096	64/4
Gemma3 [?]	27B	62	5,376	32/16
DeepSeek-V3 [?]	671B [†]	61	7,168	128/128
Mistral-Large-3 [?]	673B [†]	61	7,168	128/128

表 3.2: 部分大语言模型的架构参数对比。表中列出了各模型的参数量、层数 (L)、隐藏维度 (d) 以及注意力头数 (a/b 表示 a 个 Query 注意力头, b 个 Key/Value 注意力头)。标记 [†] 的模型采用混合专家 (Mixture-of-Experts, MoE) 架构, 表中给出的是总参数量; 推理时每 token 实际激活的参数量分别为 Qwen3-235B ~ 22 B、DeepSeek-V3 ~ 37 B、Mistral-Large-3 ~ 41 B。

计算注意力, 最后拼接:

$$\text{MHA}(\mathbf{H}) = [\text{head}_1 \parallel \cdots \parallel \text{head}_\tau] \mathbf{W}^{\text{head}}, \quad \text{head}_j = \text{Att}_{\text{qkv}}(\mathbf{H}\mathbf{W}_j^q, \mathbf{H}\mathbf{W}_j^k, \mathbf{H}\mathbf{W}_j^v) \quad (3.5)$$

其中 $\mathbf{W}_j^q, \mathbf{W}_j^k, \mathbf{W}_j^v \in \mathbb{R}^{d \times d/\tau}$, $\mathbf{W}^{\text{head}} \in \mathbb{R}^{d \times d}$ 。

模型通过最大化对数似然 $\sum_{i=1}^m \log \Pr(x_i | x_0, \dots, x_{i-1})$ 进行训练。推理时采用自回归方式: 模型输入当前序列, 预测下一个 token $\hat{x}_i$, 将其附加到序列尾部, 再以更新后的序列预测 $\hat{x}_{i+1}$, 反复进行直至生成结束符。

表 3.2 列出了近年来代表性 LLM 的深度 L 、宽度 d 及注意力头数等架构参数, 帮助读者建立对模型规模的直观感受。

3.1.3 大语言模型预训练方法

直观地说, 预训练的本质就是把海量文本“喂”给模型, 让它反复练习“给定上文, 猜下一个词”这项任务。这一朴素的目标——在数学上即最大化训练数据上的对数似然——正是驱动 LLM 获得强大语言能力的核心机制。

预训练数据

预训练首先需要大量的文本数据。如表 3.3 所示, 现代 LLM 在预训练阶段通常消耗数万亿乃至数十万亿个 token, 远超传统 NLP 模型数个数量级。

LLM	Year	# of Tokens
GPT-3-175B [?]	2020	0.5T
LLaMA2-70B [?]	2023	2.0T
LLaMA3-8B [?]	2024	15T
DeepSeek-V3 [?]	2024	14.8T
Qwen3-235B [?]	2025	36T

表 3.3: 部分 LLM 的预训练数据规模。

除了规模之外，预训练数据的质量与构成同样决定了模型最终的能力边界。训练语料通常来自多个渠道：

- **网络爬取数据：**占比最大，但也包含大量低质量、重复甚至有害内容。已有工作表明 [?]，通过系统性的过滤和清洗，可剔除约 90% 的原始爬取数据而不损害模型性能。高质量的过滤流程已成为现代 LLM 预训练数据管线中不可或缺的一环。
- **书籍与学术论文：**提供连贯的长文本和深度知识，有助于模型习得复杂的语言结构和事实性知识。
- **编程代码：**被证明是提升模型推理能力的关键。将代码纳入训练数据不仅能增强代码生成能力，还会显著改善模型在逻辑推理和多步骤任务上的表现，尤其是需要思维链（Chain-of-Thought）提示的复杂场景。
- **多语言语料：**使单一模型能够处理多种语言。但低资源语种的表现仍然受限于对应语料的体量与质量。

需要指出的是，数据的多样性和质量之间需要谨慎权衡——过度过滤可能导致某些知识或语言现象的流失，而过滤不足则引入噪声。此外，训练数据中的偏见问题（如性别刻板印象）和隐私风险（如模型记忆并复现个人身份信息）也是数据准备阶段需要关注的工程挑战，通常通过数据去偏、匿名化处理以及部署阶段的安全防护系统加以缓解。

训练目标

预训练的核心数学目标简洁而自然：最大化模型在所有训练序列上逐 token 预测的对数似然。

设训练集 $\mathcal{D}$ 由 K 个 token 序列组成。对于任意序列 $\mathbf{x} = x_0 \dots x_m$ ，模型在该序列上的对数似然为：

$$\mathcal{L}_\theta(\mathbf{x}) = \sum_{i=1}^m \log \Pr_\theta(x_i \mid x_0, \dots, x_{i-1}) \quad (3.6)$$

其中 θ 表示模型参数, $\Pr_\theta(x_i | x_0, \dots, x_{i-1})$ 正是上一节所定义的、由 Transformer 输出层 Softmax 给出的条件概率。预训练的目标是找到参数 $\hat{\theta}$ 使得整体对数似然最大:

$$\hat{\theta} = \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}_\theta(\mathbf{x}) \quad (3.7)$$

在实际实现中, 最大化对数似然等价于最小化交叉熵损失。具体而言, 在序列的每个位置 i , 模型输出一个概率分布 $\mathbf{p}_{i+1}^\theta = \Pr_\theta(\cdot | x_0, \dots, x_i)$, 将其与真实 token x_{i+1} 的 one-hot 分布 $\mathbf{p}_{i+1}^{\text{gold}}$ 计算交叉熵, 并沿所有位置求和。整个训练集上的优化目标为:

$$\hat{\theta} = \arg \min_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i=0}^{m-1} \text{CrossEntropy}(\mathbf{p}_{i+1}^\theta, \mathbf{p}_{i+1}^{\text{gold}}) \quad (3.8)$$

公式 (3.7) 与 (3.8) 在数学上完全等价, 前者体现概率建模的视角, 后者对应工程实现中的损失函数。通过优化得到的模型即可用于计算任意上下文下的 token 条件概率, 进而支持自回归文本生成。

3.1.4 缩放定律

缩放定律

直观而言, 缩放定律 (Scaling Laws) 回答这样一个问题: 如果我把模型做得更大、喂更多数据、投入更多算力, 模型性能能提升多少?

令 $\mathcal{L}$ 表示模型的测试损失 (如交叉熵), N 为模型参数量, D 为训练数据量 (以 token 计)。大量实验表明, $\mathcal{L}$ 与 N 、 D 之间存在幂律 (power-law) 关系——即损失随资源的增加按幂函数规律下降:

$$\mathcal{L}(N) = aN^b, \quad \mathcal{L}(D) = cD^d \quad (3.9)$$

其中 a, b, c, d 为经验拟合参数, b, d 均为负数 (损失随规模增大而降低)。图 3.2 展示了这一关系的典型形态。

[?] 最早在语言模型中系统地验证了这一规律。他们的核心发现是: 在给定计算预算 C 的条件下, 模型性能主要由 N 和 D 共同决定, 而非单独由其中一方主导。换言之, 单纯增大模型而不配以足够的数据, 边际收益会迅速衰减。

在此基础上, [?] 提出了更为精确的 Chinchilla 缩放定律。他们将测试损失同时建模为 N 和 D 的函数, 并加上一个不可消除的误差下界 ϵ_∞ (不可约误差):

$$\mathcal{L}(N, D) = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + \epsilon_\infty \quad (3.10)$$

其中通过大规模实验拟合得到 $\alpha \approx 0.34$ 、 $\beta \approx 0.28$ 、 $\epsilon_\infty \approx 1.69$ 。

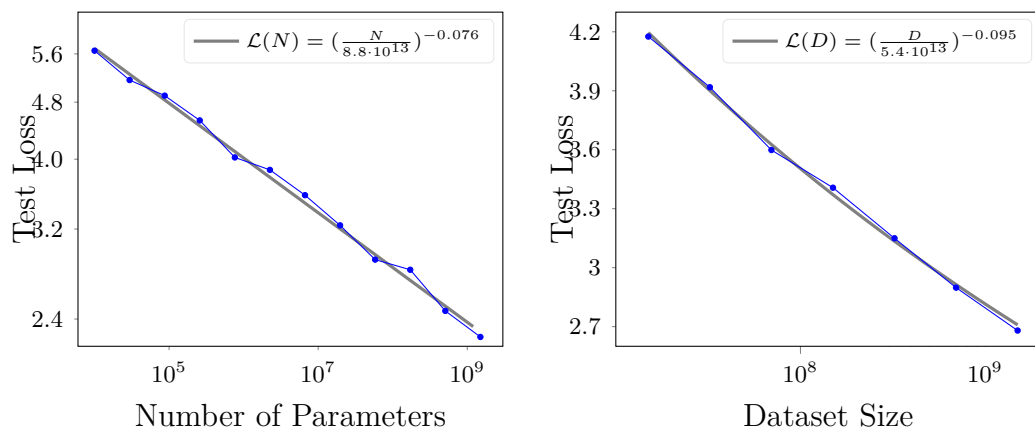


图 3.2: 测试损失与模型参数量 N 及训练数据量 D 的关系。左: $\mathcal{L}(N) = \left(\frac{N}{8.8 \times 10^{13}}\right)^{-0.076}$; 右: $\mathcal{L}(D) = \left(\frac{D}{5.4 \times 10^{13}}\right)^{-0.095}$ 。均基于 [?] 报告的拟合结果。

Chinchilla 缩放定律的核心洞见在于最优配比：对于一个给定的计算预算，模型参数 N 与训练数据 D 应当以相近的速度增长——这被称为“计算最优”（compute-optimal）训练策略。Chinchilla 的研究发现，此前的许多大模型（如 GPT-3 的某些版本）都是“参数过大而数据不足”的，同等算力下若减少参数并增加数据量，反而能获得更低的损失。

重点提示

预训练的核心目标是最小化交叉熵损失，而缩放定律揭示了模型参数量与训练数据量必须同步增长才能获得最优性能。Chinchilla 定律指出，在给定算力下，参数和数据应等比例扩大，避免“大模型小数据”的低效配置。这一结论直接指导了后续模型规模和数据量的设计决策。

缩放定律对于 LLM 研发布局具有重要的指导意义。其一，它使得在训练前即可大致预测给定资源配置下的模型表现，从而优化计算预算的分配。其二，它确认了“持续扩大规模”这一技术路线至今仍处于收益区间内——表 3.3 所示的训练量从 0.5T 飙升至 36T，正是缩放定律驱动的结果。

值得一提的是， $\mathcal{L}$ 的幂律下降还伴随着模型能力在质上的跃迁。[?] 观察到，当模型规模突破某一阈值时，某些能力（如多步推理、上下文理解）会从接近随机水平骤升至显著优于 baseline——这一现象被称为涌现能力（emergent abilities）。涌现能力的存在，为缩放定律的实用价值提供了另一层佐证：不断扩张的模型规模不仅能带来量上的损失降低，更可能触发质的飞跃。

分布式训练

现代 LLM 动辄数百亿乃至数千亿参数，单张 GPU 的显存远不足以装载完整模型，更无法在合理时间内完成训练。因此，分布式并行训练成为支撑 LLM 规模化的关键技术基石。当前主流方案包含三类并行策略，常组合使用（如 Megatron-LM 提

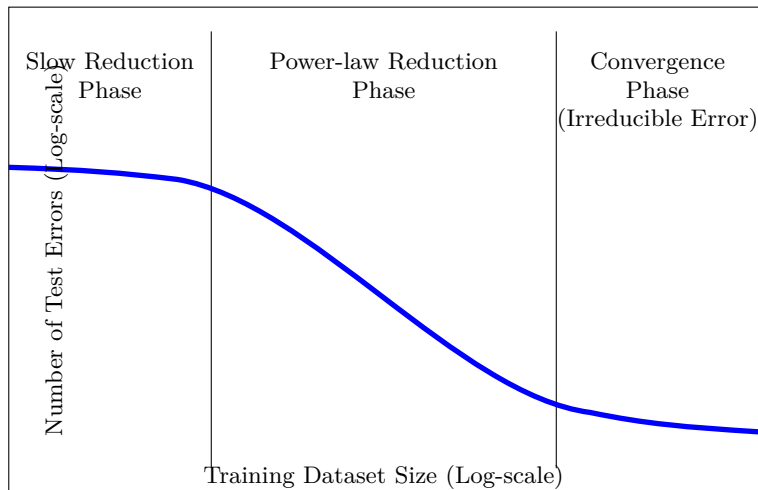


图 3.3: 测试误差随训练数据量变化的缩放曲线 [?]。在大多数实用区间内，误差按幂律衰减；当数据量极大时衰减趋缓，但误差始终存在一个大于零的下界（不可约误差）。

出的三维并行 [?]):

数据并行。数据并行是最基本、最广泛使用的并行策略。其思路简洁直观：将每个训练批次（mini-batch）拆分为 N 份子批次，分发至 N 个工作节点。每个节点持有完整的模型副本，独立完成前向和反向传播并计算局部梯度，随后通过通信操作（如 All-Reduce）聚合各节点的局部梯度，得到全局梯度后更新参数。在通信开销可忽略的理想条件下，数据并行能实现接近 N 倍的训练加速。

然而，数据并行的前提是每个节点能装下完整模型。随着模型参数量突破单卡显存上限，单纯的模型复制策略不再可行——此时需要在模型结构层面进行拆分，即模型层面的并行。

流水线并行。流水线并行将模型的不同层（或层组）分配到不同设备上。一个直观但不实用的做法是让各层串行执行——这会导致任一时刻仅有一个设备在计算，其余全部空闲。流水线并行（如 GPipe [?]）通过引入微批次（micro-batch）机制来解决这一低效问题：将一个 mini-batch 进一步切分为若干个 micro-batch，各设备处理完当前 micro-batch 后立即传递给下游，随即开始处理下一个 micro-batch。这样一来，不同设备上的计算便在时间维度上实现了重叠，大幅减少了空闲时间。micro-batch 数量越多，流水线气泡（bubble）越小；但 batch 过小也会降低 GPU 算力利用率，需在实际部署中权衡。

张量并行。张量并行从另一个维度切入：不拆分模型层，而是拆分单层内部的计算。以 FFN 子层中的矩阵乘法 $\mathbf{h}\mathbf{W}_h$ 为例（ $\mathbf{W}_h \in \mathbb{R}^{d \times d_h}$ ），可将 $\mathbf{W}_h$ 沿列切分为 M 个子矩阵 $\{\mathbf{W}_h^1, \dots, \mathbf{W}_h^M\}$ ，每块大小为 $d \times (d_h/M)$ 。于是：

$$\mathbf{h}\mathbf{W}_h = [\mathbf{h}\mathbf{W}_h^1 \parallel \dots \parallel \mathbf{h}\mathbf{W}_h^M] \quad (3.11)$$

M 个设备分别计算各自对应的子矩阵乘法，最后拼接即可得到与单设备计算完全一致的输出。这一思想同样适用于自注意力层中的 Q 、 K 、 V 投影矩阵。张量并行的优势在于：它将单个 GPU 无法容纳的大矩阵运算拆解为多个 GPU 上可并行执行的小矩阵运算，且数学上等价，不会引入额外的近似误差。

混合精度训练。除并行策略外，降低单步计算开销同样至关重要。混合精度训练是最广泛采用的优化手段之一：在前向和反向传播中使用半精度（如 FP16 或 BF16）以加速计算，同时维护一份全精度（FP32）的主权重副本用于参数更新 [?]。这一技术在显著降低显存占用和计算时间的同时，通过损失缩放（loss scaling）等技巧保证了训练的数值稳定性。

实际挑战。需要指出，分布式训练在工程实践中远不止并行策略选型那么简单。通信开销、节点同步延迟（慢节点拖累整体）、硬件故障容错、网络拓扑设计、计算与通信重叠（overlap）、负载均衡等问题，均对系统设计与调优提出严苛要求。在实际的大模型训练中，上述三类并行策略通常会被组合使用，以在算力、显存和通信三个维度间取得最优平衡。

3.2 提示学习方法

3.2.1 Zero/One/Few-Shot 提示方法

在与大语言模型交互时，我们输入的文本称为提示（prompt），模型据此生成回复。基于同一个预训练模型，只需更换提示即可完成翻译、问答、写作等多种任务，无需重新训练。本节旨在介绍提示设计的基本方法与思路，帮助读者更有效地利用大语言模型完成下游任务。

根据提示中包含的示例数量，可将其大致划分为三类：零样本（zero-shot，不提供示例）、单样本（one-shot，提供一个示例）和少样本（few-shot，提供多个示例）。本节将围绕这三类方法展开讨论。需要说明的是，提示的实际效果与最佳表述方式往往因模型而异，因此本文不针对特定大语言模型，而是总结一些具有通用性的指导原则。

提示的基本概念

在深入介绍这三类提示方法之前，我们先明确提示的基本概念及其构造方式。“提示”一词含义多样，在本章中，我们将其定义为 LLM 的输入文本，记作 $\mathbf{x}$ 。LLM 在给定输入 $\mathbf{x}$ 的条件下，通过最大化概率 $\Pr(\mathbf{y} | \mathbf{x})$ 生成输出文本 $\mathbf{y}$ 。提示作为生成条件，可以包含任何有助于描述问题和提供信息的内容。

提示通常通过提示模板（简称模板）[?] 获得。模板是一段含有占位符或变量的文本，各占位符可填入具体信息。以下展示两个向 LLM 征求周末建议的模板示例：

请给我一些有趣的周末建议。

如果 { *premise* }, 你有什么有趣的周末建议吗?

第一个模板直接请求建议, 因此不含变量; 第二个模板中, 变量 { *premise* } 需由用户指定, 作为给出建议的前提条件。例如, 若输入:

premise = 这个周末天气很好

则可生成如下提示:

如果这个周末天气很好,
你有什么有趣的周末建议吗?

除了自由格式的文本, 另一种常见做法是采用“名称: 内容”的结构组织提示, 明确标识问题与答案的位置。例如, 用“问”和“答”分别表示问题与答案, 可构建如下问答模板:

问: { *question* }
答: _____

这种“问: ... 答: ...”结构既可单独使用, 也可重复多次并填入具体的问答实例, 从而构成后文将要介绍的小样本提示的基本形式。

除了直接描述任务, 许多提示还会包含一段系统信息 (system message), 用于设定模型的角色、能力与行为约束。系统信息帮助 LLM 更好地理解当前任务的背景, 从而生成更符合预期的回复。以下是一个语法纠错任务的提示, 其中第一条语句便是系统信息:

系统 你是一位乐于助人的助手, 擅长语法纠正。

用户 请将下面的句子修改为语法正确的英文:

输入: She don't like going to the park.

输出: _____

上述提示中没有给出任何示例，仅凭指令和系统信息引导模型完成任务。这种不提供示例、直接要求模型解决问题的提示方式，即为零样本提示。关于这一方法的更多讨论，以及通过增加示例实现的单样本和少样本学习，将在下一节详细展开。

上下文学习

学习可以在推理过程中发生。上下文学习（in-context learning）正是这样一种方法：提示中包含若干问题求解的演示，LLM 从这些演示中“学习”如何解决新问题。由于这一过程不更新模型参数，上下文学习可被视为无需额外训练或微调，便能有效激活和重组预训练知识的手段。这使得 LLM 能够快速适应新任务，拓展了预训练模型在无任务特定调整下的能力边界。

我们可以通过对比零样本学习、单样本学习和少样本学习来具体说明上下文学习。在上一节中，我们已经看到了一个语法纠错的零样本学习示例——它直接要求模型纠正句子，不提供任何示例。在单样本学习中，我们添加一个已完成的纠错示例作为演示，让 LLM 从这个新增的经验中学习：

系统 你是一位乐于助人的助手，擅长语法纠正。

用户 请将下面的句子修改为语法正确的英文：

输入：There is many reasons to celebrate.

输出：There are many reasons to celebrate.

用户 请将下面的句子修改为语法正确的英文：

输入：She don't like going to the park.

输出：_____

在此基础上增加更多演示，便构成少样本学习：

系统 你是一位乐于助人的助手，擅长语法纠正。

用户 请将下面的句子修改为语法正确的英文：

输入：There is many reasons to celebrate.

输出：There are many reasons to celebrate.

用户 请将下面的句子修改为语法正确的英文：

输入：Me and my friend goes to the gym every day.

输出：My friend and I go to the gym every day.

用户 请将下面的句子修改为语法正确的英文：

输入：She don't like going to the park.

输出：_____

在少样本学习中，我们实质上提供了一个从输入到输出的映射模式，LLM 尝试遵循这一模式进行预测。上下文学习的有效性在很大程度上依赖于提示质量和模型的基础能力：一方面，我们需要通过提示工程精心设计演示内容，以帮助模型更有效地从中学习；另一方面，能力更强的 LLM 往往能更好地利用上下文学习执行新任务。例如，若希望 LLM 将因纽特语翻译为英语，但模型在预训练阶段严重缺乏因纽特语数据，那么无论怎样设计提示，都很难获得高质量的翻译。此时，更好的策略是用更多相关数据继续训练模型，而非一味优化提示。

关于上下文学习为何在预训练过程中涌现，以及为何在推理时起作用，一个直观的理解是：预训练模型已经获得了解决问题的能力，但面对新问题时可能同时存在多种合理的预测路径。提供演示可以引导模型遵循“正确”的路径。此外，研究者也从贝叶斯推断 [?]、梯度下降 [??]、线性回归 [?] 和元学习 [?] 等角度对上下文学习进行了理论解释。

3.2.2 进阶提示方法

我们已经介绍了与 LLM 提示相关的基本概念，并展示了许多用于 NLP 任务的提示。现在我们来探讨几种增强提示有效性的技术。

提示工程策略

设计提示是高度经验性的。总的来说，对于同一任务，有很多方法可以提示 LLM，我们需要进行多次试错才能找到一个令人满意的提示。为了更高效地编写好的提示，可以遵循某些策略。常见的提示原则示例包括：

- **尽可能清晰地描述任务。**当我们应用 LLM 解决问题时，我们需要提供对问题的精确、具体、清晰的描述，并指示 LLM 按照我们的期望执行。当我们希望

LLM 的输出满足某些期望时，这一点尤其重要。例如，假设我们对气候变化感到好奇。一个要求 LLM 提供一些信息的简单提示是

告诉我关于气候变化的信息。

由于这个指令太过笼统，LLM 可能会生成一个涉及气候变化任何方面的回应，而这可能与我们的特定兴趣不符。在这种情况下，我们可以使用更具体、更详细的提示。现在，假设我们打算向一个 10 岁的孩子解释气候变化。一个这样的例子是

向一个 10 岁的孩子解释气候变化的原因和影响。谈谈它如何影响天气、海平面和气温。同时，提及人们正在做的一些有助于解决问题的事情。请尝试用简单的语言解释，并且不超过 500 个词。

- **引导 LLM 思考。** LLM 已经展现出了惊人的“思考”能力。一个常见的例子是，发展成熟的 LLM 在被认为极具挑战性的数学推理任务中取得了令人印象深刻的表现。在提示工程中，LLM 的“思考”能力需要通过适当的提示来激活，特别是对于那些需要大量推理的问题。在许多情况下，被指示“思考”的 LLM 与被指示直接执行任务的同一个 LLM 相比，可以产生完全不同的结果。例如，[7]发现，仅仅在每个提示的末尾附加“Let’s think step by step”就可以提高 LLM 在多个推理任务上的性能。有多种方法可以提示 LLM 进行“思考”。一种方法是指示 LLM 在得出最终答案之前，生成推理问题的步骤。例如，考虑一个解决数学问题的任务。下面是该任务的一个简单提示。

你是一位数学家。你将收到一个数学问题。请解决这个问题。

由于解决数学问题需要详细的推理过程，如果 LLM 试图直接得出答案，很可能会犯错。因此，我们可以明确要求 LLM 在得出结论之前遵循给定的推理过程。

你是一位数学家。在解决数学问题时，你将遵循以下详细的推理步骤。

第 1 步：问题解读。

数学家仔细倾听你的问题，并理解你提出的数学挑战的复杂细节。

第 2 步：策略制定。

数学家凭借其渊博的知识，选择针对特定类型数学问题（无论是代数、微积分还是几何）的最有效策略。

第 3 步：详细计算。

数学家以精准和专业的方式，逐步执行必要的计算，并遵守所有数学原则。

第 4 步：解答回顾。

在提供最终答案之前，数学家会仔细检查计算的准确性，并为解答提供简洁的解释或理由。

你将收到一个数学问题。请解决这个问题。

{*problem*}

另一种方法是通过多轮交互，先让模型给出初步答案，再提示其评估、修正，从而引出更可靠的最终结果。

- **提供参考信息。**如前一节所述，我们可以在提示中包含示例，让 LLM 通过这些示例进行情境学习，从而了解如何执行任务。事实上，鉴于 LLM 卓越的语言理解能力，我们可以在提示中加入任何类型的文本，这样模型就可以基于更丰富的上下文进行预测。在许多应用中，我们拥有与用户查询相关的各种信息。我们通常不希望 LLM 进行无约束的预测，而是希望 LLM 生成的输出局限于相关文本。一个这样的例子是 RAG，其中用户查询的相关文本是通过调用 IR 系统提供的，然后我们提示 LLM 基于这些提供的相关文本生成回应。下面的提示展示了一个例子。

你是一位能为输入查询生成答案的专家。现在你收到了一个查询和相应的上下文信息。请基于此上下文信息生成答案。注意，你需要用自己的话来提供答案，而不是简单地从提供的上下文中复制。

上下文信息: {*IR-result*}

查询: {*query*}

在处理现实世界的问题时，我们通常拥有有助于产生更好答案的先验知识和额外信息。在提示中考虑这些信息通常有助于改善结果。当上下文足够可靠时，甚至可以要求模型仅基于给定文本回答，避免幻觉。

上面，我们只讨论了几种编写好提示的策略。此外，提示的呈现格式（如使用分段、引号或代码风格）也对结果有显著影响，这些细节需要在实践中不断体会。当然，还有很多这样的方法，需要通过实践来形成自己的方法。感兴趣的读者可以参考各种在线文档以获取更多信息，例如 OpenAI 关于 GPT 系列模型的手册³。

思维链

思维链（*CoT*）使 LLM 能够针对复杂问题生成逐步的推理过程，从而以更接近人类认知的方式处理任务。与直接输出结论不同，CoT 要求 LLM 自行产生推理步骤，或从提示中给出的详尽推理演示中学习。为具体说明 CoT，不妨考虑文献中常用的一类代数计算问题。设问题为：

计算数字 2、4 和 6 的平均值。

可直接将该问题作为输入，提示 LLM 作答。

Q: Please calculate the average of the numbers 2, 4, and 9.

A: The answer is 6.

此时 LLM 通常难以直接给出正确结果。一种简单的改进措施是在提示中加入同类问题的问答示例，使 LLM 从中获得参照。

Q: Please calculate the average of the numbers 1, 3, 5, and 7.

A: The answer is 4.

Q: Please calculate the average of the numbers 2, 4, and 9.

A: The answer is 7.

然而，仅提供问答对仍不足以使 LLM 掌握正确的推理方式。CoT 方法的核心概念在于，LLM 不仅学习问题和答案之间的映射关系，还能从推导答案所涉及的详细步骤中获益。为此，可在提示中加入若干推理步骤，构造 CoT 风格的提示。

Q: Please calculate the mean square of the numbers 1, 3, 5, and 7.

A: Calculate the square of each number: $1^2 = 1$, $3^2 = 9$, $5^2 = 25$, and $7^2 = 49$. Sum the squares, $1 + 9 + 25 + 49 = 84$. There are 4 numbers in total. Divide the sum by the number of items, $84/4 = 21$. The answer is 21.

³参见 <https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results>.

Q: Please calculate the average of the numbers 2, 4, and 9.

A: Calculate $2 + 4 + 9$, which equals 15. There are three numbers. Divide the total sum by the count, resulting in $15/3 = 5$. The answer is 5.

上例中以绿色标示的部分即为推理步骤。通过展示同类问题的完整推理过程，LLM 得以学会如何进行逻辑推导，从而生成通向正确答案的解题路径。

重点提示

思维链（CoT）是提升复杂推理最有效的提示技巧，其本质是强制模型生成中间推理步骤，将多步问题拆解为可验证的序列。无论是少样本（提供推理示例）还是零样本（添加“逐步思考”指令），都能显著提高数学、逻辑等任务的正确率。关键在于确保推理步骤完整且连贯，而非仅关注最终答案。

CoT 提示在实际应用中表现出若干优势。其一，CoT 将复杂问题拆解为一系列顺序化的子步骤，这一过程在一定程度上模拟了人类解决复杂问题的行为模式，因此尤其适用于需要多步推理的任务。其二，CoT 使推理过程更加透明，所有中间步骤均可见，便于使用者理解并解释结论的形成依据。其三，当使用者能够看到并理解推理逻辑时，其对模型输出的信任度往往会提高，这在医学、教育和金融等对可解释性要求较高的领域尤为重要。

上述方法需要提供一个或多个包含推理步骤的示例，通常称为少样本 CoT。与之相对，零样本 CoT 无需提供示例，而是通过在提示中加入特定指令，要求 LLM 执行逐步推理。例如，下面给出一个零样本 CoT 提示的实例。

Q: Please calculate the average of the numbers 2, 4, and 9.

A: Let's think step-by-step.

We have three numbers: 2, 4, and 9. Add these numbers together, $2 + 4 + 9 = 15$. Determine how many numbers there are, which in this case is three. The average is calculated by dividing the total sum by the number of elements. Completing the division gives $15/3 = 5$. So the answer is 5.

在“让我们一步一步思考”这一指令的引导下，LLM 被要求生成详细的推理步骤。?] 的研究指出，此类指令有时会使 LLM 只输出推理过程而不给出最终结论。此时，可通过第二轮提示将第一轮的和输出结合，再交由 LLM 继续生成，从而提取出正确答案。此外，除“一步一步思考”外，其他类似指令（如“请先展示思考过程”或“让我们有逻辑地分析”）同样可用于引导推理。

在结束本节之前，有必要审视 CoT 在实际应用中面临的若干局限。其一，少样

本 CoT 依赖高质量的多步推理示例，而这些示例的获取（无论是自动还是手动）均存在一定难度。其二，缺乏将复杂问题分解为适当子步骤的标准方法论，这在很大程度上依赖于使用者的经验。其三，中间步骤的累积错误可能对最终结论的准确性产生不利影响。关于 CoT 优缺点更为全面的讨论，可参考近年来的综述文献 [??]。

3.2.3 学习提示

前面介绍的提示策略均为人工设计。手工设计高质量提示不仅费力，而且存在明显局限：第一，不同大语言模型对提示的敏感程度各异，要找到通用而有效的提示往往需要大量试错，成本高昂；第二，完全依赖人类直觉容易遗漏那些不太直观但可能十分有效的提示，限制了提示的多样性；第三，人工编写的提示时常复杂冗长，既增加了推理计算开销，也不一定带来性能增益。

为克服这些问题，研究者提出了一系列自动学习提示的技术，目标是自动创建、优化和表示提示，以便更高效地解决下游任务。围绕这一目标，本节讨论以下两个核心问题：

- 如何为语言模型自动设计和优化提示？
- 除文本字符串外，提示是否还有其他表示形式？如何学习这些表示？

提示优化

自动提示优化的目标是用算法发现针对特定任务的最优离散提示，这可以视为自动机器学习 (AutoML) 的一种实例，与神经架构搜索 (NAS) 的思路颇为相似。一个通用的提示优化框架通常包含三个关键组件：

- **提示搜索空间**。定义所有可供探索的提示。例如，可以围绕若干种子提示，通过编辑、扩展等方式生成多样化的候选。
- **性能评估**。对每个候选提示，将其送入语言模型并在验证集上衡量下游任务表现，以此作为提示质量的度量。
- **搜索策略**。决定如何遍历搜索空间。通常采用迭代方式：每一步选取当前最有希望的一组提示进行评估，并据此生成新的候选，循环往复，直至输出表现最佳的提示。

目前广泛使用的一类方法利用大语言模型自身来实现上述组件，形成基于大语言模型的提示优化。以 [?] 的工作为例，该方法交替执行以下步骤：

- **初始化**。构造候选提示池。既可以由人工编写少量初始提示，也可以借助语言模型，根据任务描述或少量输入-输出示例自动生成初始提示。

- **评估**。将候选池中的每个提示喂给语言模型，在验证集上计算预定义的性能指标（如准确率或对数似然），得到对应的质量分数。
- **修剪**。若候选池过大，可依据评估分数保留得分较高的一部分提示，舍弃其余，以减轻后续步骤的计算负担。
- **扩展**。以当前保留的提示为基础，利用语言模型生成语义相近或有变化的新提示（例如通过改写、局部编辑等方式），并加入候选池，从而不断探索更广的搜索空间。

评估、修剪、扩展三步可以反复进行，使系统在迭代中逐步发现更优的提示。扩展阶段的具体实现可以多种多样：它既可以是对提示进行释义或施加编辑操作，也可以引入反馈-修订循环，让语言模型根据输出质量对提示进行自我修正。但无论形式如何变化，核心理念始终一致——将提示设计转化为一个可由语言模型驱动搜索与优化问题。

需要指出的是，提示实际上具有结构化特征，通常包含指令、示例等多个组成部分。目前大多数自动提示优化工作侧重于学习更好的**指令**，即生成能有效引导模型完成任务的文本说明；同时，研究也在向自动选择或生成示例等方向延伸。

软提示

自然语言提示（即硬提示）虽然直观，但往往冗长且计算开销大，反复输入相同长提示更是低效。注意到硬提示作为离散 token 序列，实际上会被大语言模型编码为连续向量，这引出一个自然的问题：能否直接使用这些连续表示作为更紧凑的提示？为此，我们引入软提示（soft prompt）的概念，视其为提示的隐藏、分布式表示。与人类可读的硬提示不同，软提示是模型内部的、可适应的隐式模式，以实值向量形式存在，更便于模型处理。

Translate the sentence into Chinese.

Consider it done!

例如，提示“Translate the sentence into Chinese.”可视为硬提示，其 token 序列经 LLM 转换为向量序列 $\mathbf{h}_1 \dots \mathbf{h}_5$ ，这些向量即可看作一种软提示（图 3.4）。但软提示不必来自自然语言的翻译；它们可以完全脱离具体文本，直接作为连续参数学习得到。这样的表示密集、低维且可优化，训练和推理成本远低于长硬提示，在重复使用同一提示的场景中尤为实用。

上述思想催生了一系列通过学习连续向量来高效适配大语言模型的方法。典型代表是参数高效微调中的软提示：**前缀微调**（prefix tuning）在每一 Transformer 层的

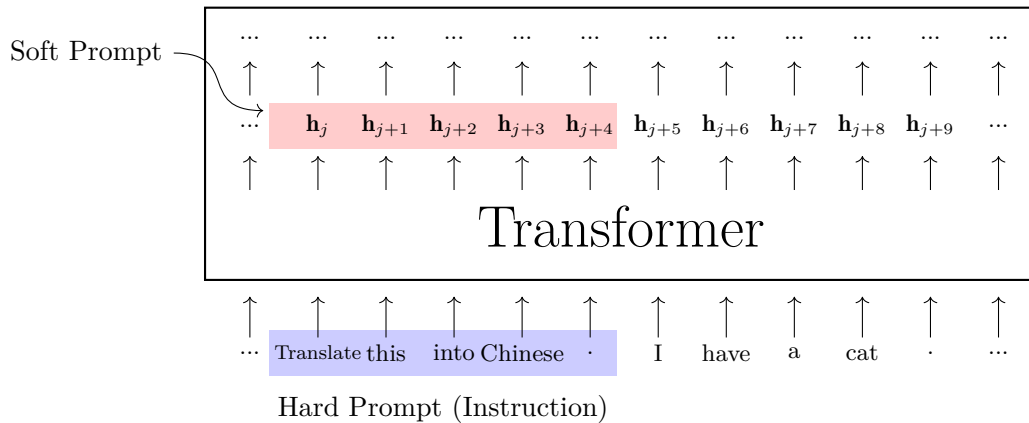


图 3.4: 硬提示与软提示示意图。这里，硬提示是我们为了执行任务而输入给 LLM 的指令。LLM 正常对该指令进行编码，其对应的中间表示可被视为一种软提示。

输入前添加可训练的前缀向量作为软提示 [?]；**提示微调**（prompt tuning）则仅在嵌入层加入可学习的软提示嵌入，而保持模型其余参数不变 [?]。这些软提示通过与原有表示交互，将任务知识注入模型，以极低的参数成本实现适配。

另一类方法从上下文压缩的视角学习软提示。为替代冗长的指令或示例，可将完整上下文压缩为少量连续向量（软提示），并通过知识蒸馏或循环聚合训练得到。例如，[?] 将长上下文划分为片段，逐步使用摘要 token 累积记忆，最终生成固定大小的软提示表示整个上下文。此类压缩表示进一步拓展了软提示在高效推理中的应用。

总而言之，软提示架起了高效计算与灵活任务适配的桥梁，已成为提示工程与参数高效微调的核心概念之一。

3.3 大模型微调技术

3.3.1 微调概述

预训练完成后，大型语言模型（LLM）可被直接用于解决各类 NLP 任务。其工作方式本质上是**文本补全**：给定一段上下文，模型生成最可能的后续文本。形式化地，设上下文为 token 序列 $\mathbf{x} = x_0 \dots x_m$ ，待生成序列为 $\mathbf{y} = y_1 \dots y_n$ ，则推理过程可定义为

$$\begin{aligned} \hat{\mathbf{y}} &= \arg \max_{\mathbf{y}} \log \Pr(\mathbf{y}|\mathbf{x}) \\ &= \arg \max_{\mathbf{y}} \sum_{i=1}^n \log \Pr(y_i | x_0, \dots, x_m, y_1, \dots, y_{i-1}) \end{aligned} \quad (3.12)$$

这与标准语言模型的对数概率形式完全一致，区别仅在于预测从位置 $m+1$ 开始。因此，只需将任务包装成合适的**提示模板**，就能让同一个 LLM 处理多种问题。例如，判断句子语法是否正确，可构造如下输入：

John seems happy today.

Question: Is this sentence grammatically correct?

Answer: _____

模型在上下文后生成 “Yes” 即给出了判断。同样地，机器翻译可以使用 “Translate this sentence from English into Chinese:” 这样的指令模板，甚至可以写成类似代码的配置块。这种统一范式的强大之处在于，无需为每个任务设计专门的模型结构。

然而，让 LLM 稳定地 “听懂” 指令并非天然具备的能力。一种直接的思路是将大量（指令，响应）样本混入预训练数据，但这需要海量标注，且从头训练的计算代价极高。因此，目前主流且高效的做法是**指令微调**（instruction fine-tuning）：在预训练模型的基础上，用一套规模小得多但质量高的指令数据集 $\mathcal{D}_{\text{tune}}$ 对参数进行轻微调整，从而激活模型在预训练中已习得的通用语言知识，使其泛化到各种指令驱动的任务上。

指令微调数据由输入-输出对构成。输入部分融合了任务指令、可选的系统提示以及用户提供的内容，输出部分则为对应的标准回答。以下给出了几个典型的微调样本，它们覆盖了分类、生成、推理和代码等不同任务，并且同一任务（如语法判别）可以对应风格迥异的指令，从而增加数据多样性：

Read the following sentence, and answer whether it is grammatically correct.

LLMs are powerful models but are expensive to build.

Yes

Write a short story about three characters: a boy named Andy, his dog, and his friend Jane.

In the heart of a bustling city, Andy, his loyal dog Rocket, and his friend Jane discovered a hidden garden beneath the glow of twilight...

If you buy 5 apples and each apple costs \$1.20, how much do you spend in total?

\$6.00

Write a Python program to calculate the sum of squares of the following numbers: 1, 2, 10, -9, 78.

```
numbers = [1,2,10,-9,78]
sum_of_squares = sum(x**2 for x in numbers)
print(sum_of_squares)
```

这些示例中，输入部分（蓝色）在计算损失时会被屏蔽，模型仅学习预测下划线标注的输出部分。研究指出，扩大微调任务的数量和指令的多样性可以显著提升模型性能 [?]。但值得强调的是，微调数据的体量远小于预训练语料——通常数万至数十万条高质量样本即已足够 [?]，而预训练则需要数十亿甚至数万亿 token，两者在计算开销上差了几个数量级。

更为关键的是，经过指令微调后，模型往往表现出**零样本泛化**（*zero-shot learning*）能力：它不仅能完成训练时见过的指令类型，还能应对从未被显式训练过的全新任务描述 [?]。这种能力正是生成式 LLM 与早期 BERT 等“针对每个下游任务独立微调”范式的分水岭。背后的原理通常被理解为，预训练已经储备了充足的语言理解和推理基础，指令微调只是引入了一种监督信号，将潜藏的指令遵循与任务解决能力“唤醒”。

微调的数学形式与预训练高度相似，但目标函数聚焦于输出段。设预训练得到的参数为 $\hat{\theta}$ ，微调数据集为 $\mathcal{D}_{\text{tune}}$ ，则最优参数 $\tilde{\theta}$ 为：

$$\tilde{\theta} = \arg \max_{\hat{\theta}^+} \sum_{\text{sample} \in \mathcal{D}_{\text{tune}}} \mathcal{L}_{\hat{\theta}^+}(\text{sample}) \quad (3.13)$$

对每个样本，我们将其切分为输入段 $\mathbf{x}_{\text{sample}}$ 和输出段 $\mathbf{y}_{\text{sample}}$ ，损失函数只作用于后者：

$$\mathcal{L}_{\hat{\theta}^+}(\text{sample}) = -\log \Pr(\mathbf{y}_{\text{sample}} | \mathbf{x}_{\text{sample}})_{\hat{\theta}^+} \quad (3.14)$$

在具体实现中，前向传播照常对整个拼接序列进行计算，但反向传播时，误差梯度仅流经 $\mathbf{y}_{\text{sample}}$ 对应的位置。考虑这样一个序列：

$$\underbrace{\langle s \rangle \text{ Square this number . 2 .}}_{\text{Input}} \quad \underbrace{\text{The result is 4 .}}_{\text{Output}}$$

损失仅基于“The result is 4 .”这几个 token 计算并回传，输入部分不产生任何梯度。这种方式确保模型专注于学习如何根据指令生成正确的响应，而非重复记忆输入内容。

指令微调在实际操作中仍需要一定的工程调参——学习率、批大小、训练步数等超参数需要仔细摸索。不过，由于其数据规模和计算量远不及预训练，多次实验的整

体成本是完全可控的。正是这种低成本、高效率的特性，使得微调成为适配 LLM 的通用技术栈核心：除了指令微调，它还被广泛用于将基座模型转化为对话助手、适配超长上下文、进行领域知识注入等。后续章节将深入探讨更高效的微调算法以及在更多场景下的应用。

3.3.2 有监督微调方法

微调数据获取

与预训练可以方便地获取海量原始文本不同，SFT 依赖高质量的人工标注或精心筛选的指令 - 响应对。数据的准确性、多样性和任务相关性直接决定了微调后模型的性能。研究表明，数万条高质量样本便足以有效激活模型的指令遵循能力，而低质量数据反而会损害模型。因此，数据的构建与审查成为 SFT 中最为耗时却最为关键的环节。微调数据的获取方式主要有人工生成和自动生成两类。

人工生成数据 一种直接的方法是招募人类标注员为感兴趣的任務创建输入-输出对。与传统 NLP 标注不同，为 LLM 构造微调数据需要更多的步骤。首先需要设计提示模板来描述任务。例如，为英汉翻译任务可定义如下模板：

Instruction	Translate the text from English to Chinese.
User Input	{*text*}
Output	{*translation*}

收集源文本与目标文本对（例如 “How’s the weather today?” → “今天天气怎么样？”），将变量 `{*text*}` 和 `{*translation*}` 替换为具体句子，即可形成微调样本：

Instruction	Translate the text from English to Chinese.
User Input	How’s the weather today?
Output	<u>今天天气怎么样？</u>

如此构造的 $(\mathbf{x}, \mathbf{y})$ 对可直接用于微调。

这里的困难在于同一任务存在多种编写提示的方式，不同标注员产出的模板质量和风格各异。一种广泛采用的策略是为现有 NLP 任务开发提示模板 `[? ? ?]`：标注员在了解原始任务描述和示例后，用自己的表达方式写出提示。模板确定后，利用原任务已有的标注数据便可批量生成微调样本。此外，也可直接使用互联网上的自然数据，如从问答网站收集问答对来构建开放域问答数据 `[?]`，以保证问题类型的广覆盖。

众包是另一重要途径：收集真实用户提出的问题，由人工回答或先由 LLM 生成再经人工修正，从而捕获传统 NLP 任务未涵盖的“新”问题。大量研究表明，提升微调数据的多样性有助于增强 LLM 的鲁棒性和泛化能力，我们将在第 3.3.2 节进一步讨论。

自动生成数据 人工方法受限于标注者的经验和创造力，覆盖范围与扩展效率有限。自动生成方法中，最直接的方式是利用一个已微调好的 LLM 为收集到的问题批量生成答案，形成合成微调样本。为进一步解决输入本身多样性的问题，**自指令** *Self-instruct*[?] 提供了一种系统化的方案：维护一个任务指令池，初始包含少量人工编写的种子任务及样本。每次从池中抽取若干条指令作为示例，提示 LLM 生成一条新的指令；随后再提示 LLM 为该指令生成对应的用户输入和输出，构成完整的微调样本。新样本经过启发式规则过滤（如去除与池中已有数据高度重复的样本）后加入池中，如此迭代便可扩充出大规模、多样化的微调数据集。

实际应用中，生成顺序可灵活调整。例如，对于文本分类等任务，为避免 LLM 倾向于生成某一固定类别的伪样本，可采用输入反转（input reversal）策略：先指定输出的类别标签，再提示 LLM 生成与其匹配的输入 [?]。自指令及其变体已被广泛应用于 LLM 开发，Alpaca、Vicuna 等模型的微调数据集均包含一定比例的合成数据 [?]。后续工作进一步引入进化算法以生成更多样化的指令 [?]，或将合成数据用作自提升的监督信号 [?]；近期在预训练阶段使用合成数据同样引起了关注 [? ?]。

在只有少量种子数据的小样本场景下，自指令技术也可充当数据增强手段，从种子集出发引导 LLM 生成更多样本，实现数据层面的自举。这种利用模型自身生成训练数据的实践在自然语言处理中由来已久，例如早年在句法分析和统计机器翻译中便取得了显著成效 [? ?]。

监督微调

将前面介绍的指令微调付诸实践的最直接方式，就是**监督微调**（Supervised Fine-Tuning, SFT）。如前所述，该方法利用已标注的输入 - 输出对来调整模型参数，使模型学会根据指令生成期望的响应。本节首先对这一过程进行形式化描述，然后将其扩展到多轮对话场景，最后讨论实践中需要关注的几个关键问题。

单轮预测的形式化。设输入序列为 $\mathbf{x} = x_0 \dots x_m$ （包含指令和用户提供的內容），对应的输出序列为 $\mathbf{y} = y_1 \dots y_n$ 。SFT 数据集 $\mathcal{D}$ 由大量这样的 $(\mathbf{x}, \mathbf{y})$ 对所构成。下表给出了一些典型示例：

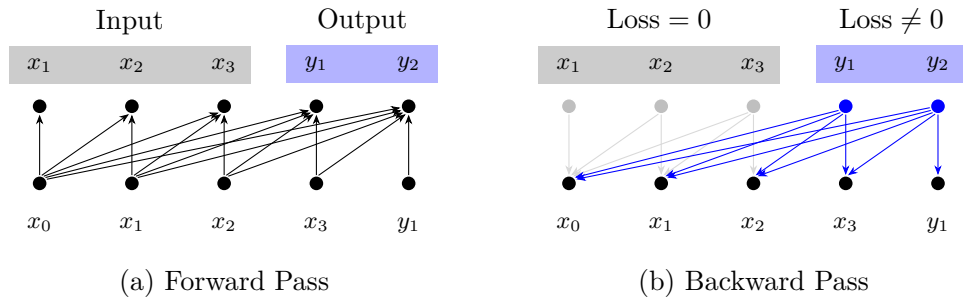


图 3.5: LLM 的监督微调图示。我们将输入和输出拼接成一个单一序列。在前向传播过程中，我们照常运行 LLM。在反向传播过程中，我们仅计算输出部分的损失，而将输入部分的损失简单地设置为 0。

$\mathbf{x}$ (指令 + 用户输入)	$\mathbf{y}$ (输出)
Summarize the following article. Article: In recent years, solar energy has seen ...	{*summary*}
Analyze the sentiment of the following review. Review: I absolutely loved the new dining experience...	Positive
Translate the following sentence into French. Sentence: practice indeed helps.	La pratique aide effectivement.
Classify the following email as spam or not spam. Text: Congratulations! You've won a \$500 gift card...	Spam

我们的目标是最大化在给定输入下生成正确输出的条件概率。沿用概述部分的记号，令 $\hat{\theta}$ 为预训练得到的参数，则微调后的最优参数 $\tilde{\theta}$ 由下式给出（为书写简洁，后文将用 θ 表示从 $\hat{\theta}$ 开始调整后的参数）：

$$\tilde{\theta} = \arg \max_{\theta} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log \Pr_{\theta}(\mathbf{y}|\mathbf{x}) \quad (3.15)$$

其中 $\log \Pr_{\theta}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^n \log \Pr_{\theta}(y_i|\mathbf{x}, \mathbf{y}_{<i})$ ，即只对输出部分的 token 计算交叉熵损失。

与标准语言模型训练不同，这里我们不关心输入序列 $\mathbf{x}$ 自身的生成概率。在实际实现中，通常将 $[\mathbf{x}, \mathbf{y}]$ 拼接为一个完整序列进行前向传播，而在反向传播时强行将输入部分的损失设为 0，仅保留输出部分的梯度。这一过程可以借助链式法则清晰地表达为

$$\log \Pr_{\theta}(\mathbf{x}, \mathbf{y}) = \underbrace{\log \Pr_{\theta}(\mathbf{x})}_{\text{设为0}} + \underbrace{\log \Pr_{\theta}(\mathbf{y}|\mathbf{x})}_{\text{实际计算损失}} \quad (3.16)$$

图 3.5 给出了这种“前向计算全序列，反向仅传播输出部分”的训练方式示意图。由此可见，SFT 本质上就是一种修改了损失掩码的标准语言模型训练。

扩展到多轮对话。上述单轮预测假定每次交互相互独立，但在聊天机器人等真实应用中，往往需要进行多轮对话。假设一段对话包含 K 轮，每轮依次为用户输入 $\mathbf{x}^k$ 和助手回复 $\mathbf{y}^k$ ，整体构成序列 $\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K, \mathbf{y}^K$ 。我们希望模型在每一轮都能基于已有的对话历史生成恰当的回复，即最大化

$$\tilde{\theta} = \arg \max_{\theta} \sum_{k=1}^K \log \Pr_{\theta}(\mathbf{y}^k | \mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^k) \quad (3.17)$$

一种朴素的实现方式是对每个 k 分别运行一次 LLM，但这样做的效率极低。更高效的做法与单轮情形类似：将整个对话序列 $[\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K, \mathbf{y}^K]$ 视为一个长序列，在单次前向传播中完成所有轮次的损失计算。应用链式法则展开：

$$\begin{aligned} \log \Pr_{\theta}(\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K, \mathbf{y}^K) &= \underbrace{\log \Pr_{\theta}(\mathbf{x}^1)}_{\text{设为0}} + \underbrace{\log \Pr_{\theta}(\mathbf{y}^1 | \mathbf{x}^1)}_{\text{损失}} + \dots \\ &+ \underbrace{\log \Pr_{\theta}(\mathbf{x}^K | \mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{y}^{K-1})}_{\text{设为0}} \\ &+ \underbrace{\log \Pr_{\theta}(\mathbf{y}^K | \mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K)}_{\text{损失}} \end{aligned} \quad (3.18)$$

我们只需将每一处用户输入项对应的对数概率损失设为 0，仅计算助手回复部分的损失，便能在单次运行中达到式 (3.17) 所描述的目标。图 3.6 展示了多轮对话 SFT 的训练范式。最终，整体训练目标依然可以简洁地写为

$$\tilde{\theta} = \arg \max_{\theta} \sum_{\text{seq} \in \mathcal{D}} \log \Pr_{\theta}(\text{seq}) \quad (3.19)$$

其中 seq 即上述完整对话序列，损失掩码模式保证了模型仅学习如何生成助手回复，而不会去预测用户提问。

总而言之，监督微调是连接通用预训练模型与具体应用场景的核心桥梁。通过精心设计单轮或多轮的指令数据，并配合合理的训练策略，我们能够高效地赋予 LLM 强大且泛化的任务解决能力。

3.3.3 高效微调方法

使用更少数据进行微调

指令微调常依赖大规模数据集，例如 FLAN 汇集了 1836 个任务、约 1500 万个样本 [?]。在如此庞大的数据上对大型语言模型进行全参数更新，计算开销极高。参数高效方法通过只更新少量参数来缓解这一问题，但微调数据本身仍可能包含合成噪

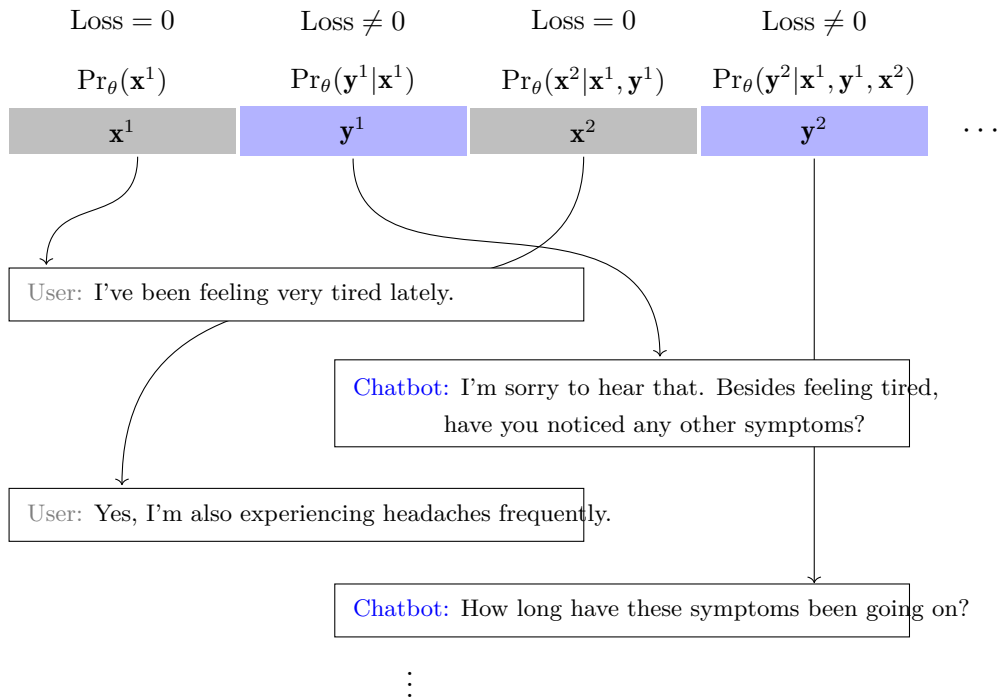


图 3.6: 对话模型的监督微调示意图。在此, LLM 充当聊天机器人, 根据对话历史响应每个请求。对话在用户和聊天机器人之间交替进行。在 SFT 中, 我们像在标准 LLM 中一样将整个对话视为一个序列, 但仅针对 LLM 的响应计算损失。

声和偏差。另一种高效思路是从数据入手：只选用最相关、最有影响力的样本进行微调，在保持模型更新质量的同时大幅压缩数据量。形式化地，给定大规模数据集 $\mathcal{D}$ ，我们希望选出一个高信息密度的子集 $\mathcal{S}^* \subset \mathcal{D}$ ，使基于该子集微调得到的模型 $\theta^*(\mathcal{S}^*)$ 在目标任务上的损失尽可能逼近使用全量数据的效果：

$$\mathcal{S}^* = \arg \min_{\mathcal{S} \subset \mathcal{D}, |\mathcal{S}|=k} \mathcal{L}_{\text{task}}(\theta^*(\mathcal{S})). \quad (3.20)$$

多项工作验证了这一思路。[?] 精心设计提示并收集样本，构建了仅包含 1000 个样本的指令遵循数据集 LIMA。使用该数据微调的 LLaMA 65B 模型，其竞争力可与投入更多微调努力的模型相媲美，甚至更优：在人类偏好评估中，LIMA-65B 对 DaVinci003 的胜率达 58%，而用 52k 条数据微调的 Alpaca-65B 仅获 47% 的胜率。这说明模型的多任务响应能力无需在所有指令类型上微调即可激发。[?] 则利用 GPT-3.5 为每条指令-回答对打分，从 Alpaca-52K 中筛选出 9000 个最高质量样本构成 AlpaGasus-9K；仅用该子集微调，在 AlpacaEval 上的表现即持平甚至超过全量数据。这些结果进一步印证了“数据质量优先于数量”的观点。此外，LESS 等方法通过计算每个训练样本对验证损失的梯度影响来筛选最有价值的样本：

$$\mathcal{I}(z) \approx -\nabla_{\theta} \ell(z)^{\top} \mathbf{H}^{-1} \nabla_{\theta} \ell(\mathcal{D}_{\text{val}}), \quad (3.21)$$

其中 $\mathbf{H}$ 为海森矩阵。选择影响力最大的子集，可高效驱动模型收敛。总体而言，使用更少但更高质量的数据训练 NLP 模型，往往能带来更好效果。

上述发现与 NLP 的传统认知有所不同：复杂任务的执行能力可以通过少量标注数据激活，而非依赖大规模监督训练。一种解释是，生成正确响应的能力已在预训练阶段习得，但指令-响应的映射在推理时并非高概率路径。微调只需对模型进行微小调整即可使其遵循指令，所需的训练量远少于预训练。这与**表面对齐假说 (superficial alignment hypothesis)** 密切相关：知识的主体在预训练期间已经建立，后续的微调或对齐仅对表层行为进行修正，不会显著改变底层知识库 [?]。图 3.7 直观地展示了这一思想：预训练构筑了模型能力的大厦，微调只是完成表层的意图对齐装修。

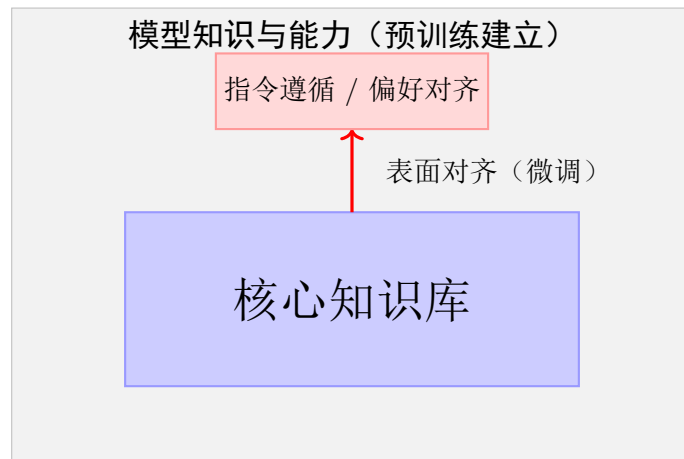


图 3.7: 表面对齐假说示意：预训练完成知识与能力的主体构建，微调仅需少量数据实现浅层的行为对齐。

参数高效微调方法

即使有监督微调的数据量远小于预训练，在数十亿参数的模型上进行全参数微调仍消耗可观的 GPU 显存与计算时间。这促使了参数高效微调 (Parameter-Efficient Fine-Tuning, PEFT) 方法的产生。与更新全部参数的全参数微调 (Full Fine-Tuning, FFT) 不同，PEFT 冻结预训练模型的大部分权重，仅训练少量附加参数或特定层。早期的 PEFT 方法通过在 Transformer 层之间插入小型全连接网络 (*adapter*) [?]，或在输入端添加可学习的连续提示向量 (如 *prefix tuning* 和 *prompt tuning*) 来引导生成 [??]。但这些方法往往在推理时引入额外延迟，或在复杂任务上难以追平全参数微调的性能上限。

为克服上述局限，[?] 提出了低秩自适应 (Low-Rank Adaptation, *LoRA*)。其核心假设是：微调过程中的权重更新 ΔW 具有较低的“内在秩”。如图 3.8 所示，对于预训练权重 $W_0 \in \mathbb{R}^{d \times k}$ ，LoRA 将其更新分解为两个低秩矩阵的乘积：

$$W' = W_0 + \Delta W = W_0 + BA, \quad \text{其中 } B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}, r \ll \min(d, k). \quad (3.22)$$

训练时 W_0 被冻结，仅优化 B 和 A 。秩 r 通常取 4、8 或 16，可带来巨大的参数缩减。以 LLaMA-7B 的某个注意力投影矩阵 ($d = k = 4096$) 为例，若 $r = 16$ ，原始矩阵参数量约 $4096^2 \approx 16.7\text{M}$ ，而 LoRA 的参数量仅 $2 \times 4096 \times 16 \approx 131\text{K}$ ，减少超过 100 倍。

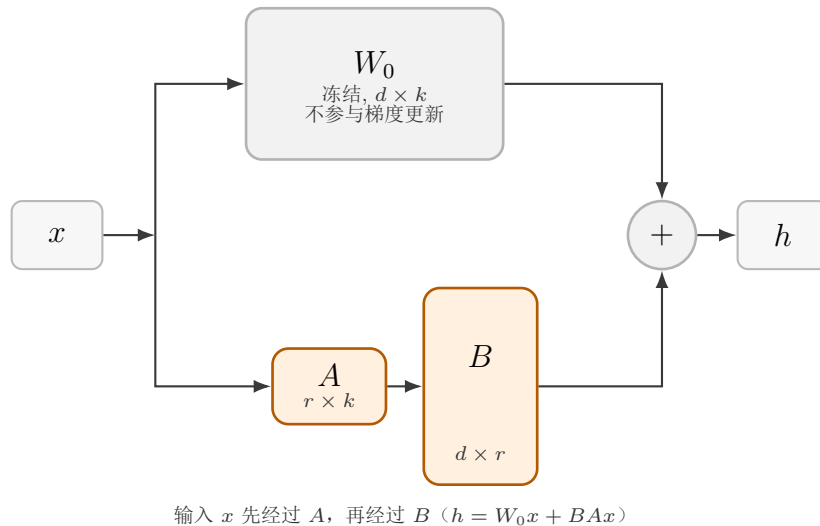


图 3.8: LoRA 示意图。输入 x 从左侧进入，分两条路径向右传播：上方路径穿过冻结的预训练权重 W_0 ；下方路径依次穿过可训练的低秩矩阵 A ($r \times k$ ，先将输入压缩到秩 r) 和 B ($d \times r$ ，再展开回维度 d)，两条路径最终在右侧汇合相加，得到输出 $h = W_0x + BAx$ 。

LoRA 的一大优势是训练后低秩矩阵可直接合并回原权重 ($W' = W_0 + BA$)，推理时不会引入任何额外延迟。近年来涌现出众多 LoRA 变体：QLoRA 将 W_0 以 4-bit NormalFloat 量化存储并配合分页优化器，使得在单张 48GB 消费级 GPU 上微调 65B 模型成为可能 [?]；DoRA 则将权重矩阵解耦为幅度与方向分别学习，进一步提升了微调的稳定性和任务表现 [?]

表 3.4 对比了全参数微调与主流 PEFT 方法在 7B 模型上的显存占用（混合精度训练）。PEFT 方法将可训练参数量降低数个数量级，显著缓解了显存压力，使大模型微调走向“平民化”。

表 3.4: 全参数微调与 PEFT 方法的资源对比（以 7B 模型，序列长度 512，批次大小 1 为例）

方法	可训练参数占比	GPU 显存占用 (估)	推理延迟
全参数微调 (FFT)	100%	~55 GB	无额外
Adapter	3%–5%	~20 GB	略有增加
Prefix/Prompt Tuning	<1%	~16 GB	略有增加
LoRA ($r=16$)	0.1%–0.5%	~18 GB	无增加
QLoRA (4-bit)	0.1%–0.5%	~10 GB	无增加

重点提示

LoRA 通过低秩分解将权重更新压缩为两个小矩阵的乘积，仅训练极少量参数即可达到接近全量微调的性能。其核心优势在于推理时零延迟（可合并回原权重），且显存占用大幅降低。需注意秩 r 的选择会影响表现，通常 8~16 即可，过大则失去高效性。

PEFT 方法之所以能用极少参数达到全参数微调的效果，可从**内在维度** (*intrinsic dimensionality*) 假说得到解释 [?]。该假说认为，尽管预训练模型参数规模庞大，但适配特定下游任务的有效解空间实际上处于一个低维子流形中。如图 3.9 所示，预训练已将参数 θ_0 置于良好的损失盆地内，微调只需在其周围的低维子空间中寻找适合任务的 θ^* ：

$$\theta^* = \theta_0 + P \cdot \delta, \quad \delta \in \mathbb{R}^m, m \ll |\theta|, \quad (3.23)$$

其中 P 为投影矩阵。LoRA 这类低秩方法恰好契合了低维更新的特性，能高效捕捉任务特定的变化，无需在高维参数空间进行昂贵搜索。

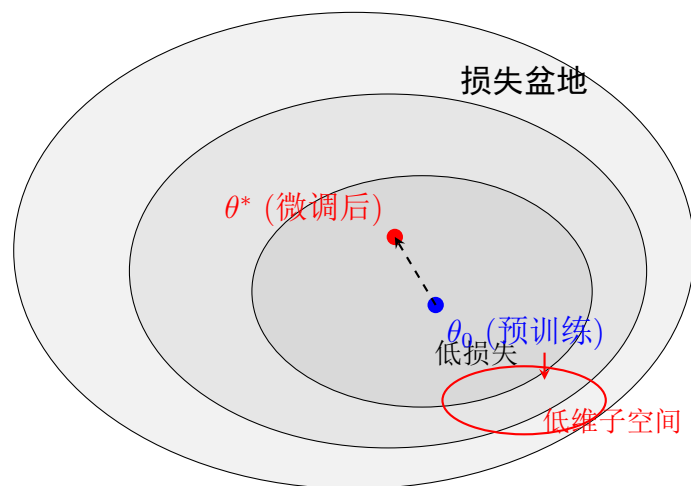


图 3.9: 内在维度假说示意：预训练模型处于宽广的损失盆地中，微调只需在低维子空间内进行微小扰动即可适配下游任务。

3.4 大模型偏好对齐技术

3.4.1 对齐概述

指令微调让大语言模型能够适应那些可被清晰定义的任务，这类问题本质上属于 *alignment* 问题。对齐，就是引导模型的行为以符合人类意图，我们期望模型不仅能遵循指令，还要做到无偏见、真实且无害。例如，一个负责任的模型应当拒绝回答“如何制造武器”这类有害请求。与对齐紧密相关的是人工智能安全，其目标在于构建安全且有益的智能系统，确保系统在正常使用乃至被滥用或恶意攻击的情况下，都

能保持鲁棒、安全，且行为符合人类的主观预期。对齐之所以困难，是因为人类价值观复杂多样且不断变化，有时我们只有看到模型的回应，才能判断它是否符合期望。因此，对齐不单是针对预设任务的微调，更是一个需要模型与真实世界不断交互、持续学习的宏大问题。

在大规模无标签数据上完成预训练后，对齐通常会经历两个步骤：

- 监督微调 (*SFT*)。使用新的、面向特定任务的标注数据继续训练模型。最常用的技术是指令微调，使模型与“遵循指令”这一预期行为对齐。
- 从人类反馈中学习。即使经过监督微调，模型仍可能生成不真实、有偏见或有害的内容。此时可以收集人类对模型输出的评估反馈，并利用这些反馈进一步训练模型，实现更深层的对齐。

一种主流的从人类反馈中学习的方法，是将其当作一个强化学习问题，称为 *reinforcement learning from human feedback (RLHF)*。该方法包含两个组件：

- *Agent*。“智能体”就是我们要训练的大语言模型。它从环境接收文本，并输出另一段文本返回给环境，其策略由模型本身的分布定义： $\Pr(\mathbf{y}|\mathbf{x})$ 。
- *Reward Model*。奖励模型充当了环境的替身，它负责对智能体生成的输出序列给出一个数值奖励，以此表征输出的好坏。

RLHF 涉及两个学习任务：一、利用人类反馈训练奖励模型；二、在奖励模型的指导下，用强化学习算法优化策略。关键步骤可以归纳为：a) 通过预训练和指令微调获得初始策略；b) 为每个输入生成多个候选输出，并收集人类对这些输出的偏好排序；c) 基于排序结果学习奖励模型；d) 以奖励模型为监督信号，对策略进行强化学习微调。图 3.10 展示了这一流程。

下面我们通过一个具体的例子来理解 RLHF 的基本思想。假设我们已有一个经过预训练和指令微调的大语言模型，用户输入：

怎样才能更环保地生活？

模型通过采样生成了 4 个不同的回答：

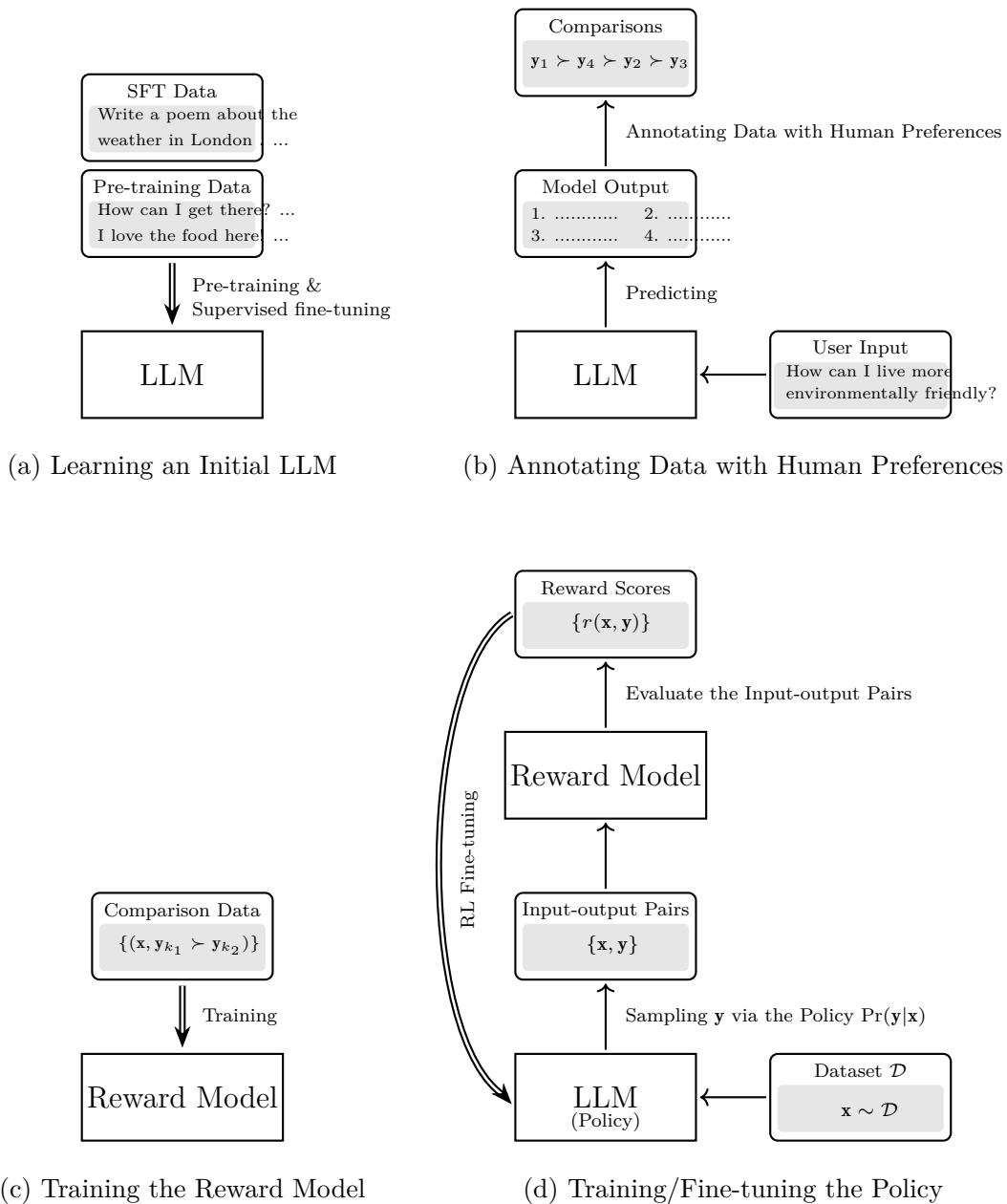


图 3.10: RLHF 概述。其中涉及 4 个关键步骤：a) 使用预训练和监督微调训练初始 LLM（即策略）；b) 通过对 LLM 的输出进行排序来收集人类偏好数据；c) 使用排序结果训练奖励模型；d) 基于奖励模型对策略进行 RL 微调。双线箭头表示训练或微调。

输出 1 ($\mathbf{y}_1$):	考虑改用电动汽车或自行车代替传统汽车, 以减少碳排放, 保护我们的星球。
输出 2 ($\mathbf{y}_2$):	尝试极简主义生活方式, 减少所拥有的物品, 从而降低消费以及制造和处置带来的环境影响。
输出 3 ($\mathbf{y}_3$):	脱离电网, 自己用可再生能源发电并收集雨水, 实现完全自给自足, 摆脱对不可再生资源的依赖。
输出 4 ($\mathbf{y}_4$):	多支持本地农产品, 这既能减少食品长途运输的碳足迹, 又能享受新鲜健康的食物。

让标注者直接为每个输出打出绝对分数往往比较困难, 但对他们来说, 判断输出之间的相对优劣则要容易得多。于是可以得到一个偏好排序, 例如:

$$\mathbf{y}_1 \succ \mathbf{y}_4 \succ \mathbf{y}_2 \succ \mathbf{y}_3$$

利用这类排序数据, 我们便可以训练奖励模型。奖励模型通常采用与目标大语言模型相似但规模更小的架构。给定输入 $\mathbf{x}$ 和输出 $\mathbf{y}_k$, 将它们拼接成一个序列 $\text{seq}_k = [\mathbf{x}, \mathbf{y}_k]$, 并在末尾添加一个特殊符号 (如 $\langle s \rangle$), 然后取 Transformer 最顶层该符号位置的输出作为整个序列的表示, 最后经过一个线性层得到奖励分数 $R(\mathbf{x}, \mathbf{y}_k)$ 。训练时常采用成对排序损失:

$$\text{Loss}_\omega(\mathcal{D}_r) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_{k_1}, \mathbf{y}_{k_2}) \sim \mathcal{D}_r} \log(\text{Sigmoid}(R_\omega(\mathbf{x}, \mathbf{y}_{k_1}) - R_\omega(\mathbf{x}, \mathbf{y}_{k_2}))) \quad (3.24)$$

其中 ω 为奖励模型参数, $\mathcal{D}_r$ 是由输入和输出对构成的偏好数据集。如果模型预测的排序与人类标注不一致, 该损失就会施加惩罚。通过最小化这一损失, 我们得到奖励模型 $R_\omega(\cdot)$, 它可以为任意输入-输出对提供连续的奖励分数, 成为后续策略学习的监督信号。

在策略学习阶段, 一个直接的目标是最大化期望奖励:

$$\tilde{\theta} = \arg \max_{\tilde{\theta}^+} \mathbb{E}_{(\mathbf{x}, \mathbf{y}_{\tilde{\theta}^+}) \sim \mathcal{D}_{\text{rlft}}} R_\omega(\mathbf{x}, \mathbf{y}_{\tilde{\theta}^+}) \quad (3.25)$$

其中 $\mathbf{x}$ 从输入数据集中采样, $\mathbf{y}_{\tilde{\theta}^+}$ 则从当前策略分布 $\text{Pr}_{\tilde{\theta}^+}(\mathbf{y}|\mathbf{x})$ 中采样。实践中通常会采用更高级的强化学习算法, 如 *proximal policy optimization (PPO)*, 以获得更稳定的训练效果, 详细算法将在后续章节展开。

3.4.2 奖励模型构建方法

奖励模型在通用强化学习框架中扮演着核心角色, 是计算价值函数的基础。本节将讨论如何训练这样的奖励模型。

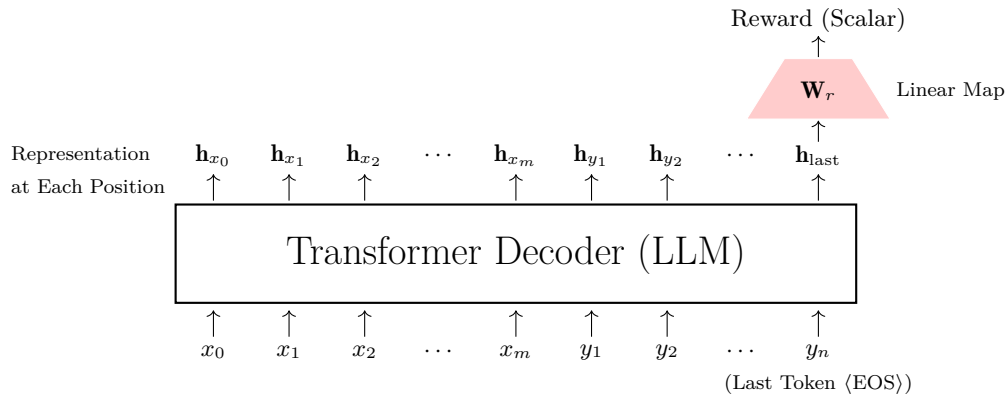


图 3.11: 基于 Transformer 的奖励模型架构。模型主体仍是一个 LLM，其 Transformer 解码器作为序列表示模型。我们抽取解码器在最后一个位置的表示作为整个序列 $[\mathbf{x}, \mathbf{y}]$ 的表示，再通过一个线性映射将其转换为标量奖励分数。

在 RLHF 中，奖励模型是一个将输入和输出 token 序列对映射为标量的神经网络。给定输入 $\mathbf{x}$ 和输出 $\mathbf{y}$ ，奖励值可以表示为

$$r = \text{Reward}(\mathbf{x}, \mathbf{y}) \quad (3.26)$$

其中 $\text{Reward}(\cdot)$ 即奖励模型。 r 衡量的是在给定输入 $\mathbf{x}$ 的条件下，输出 $\mathbf{y}$ 与期望行为的一致程度。如前所述，我们假设 $\mathbf{x}$ 和 $\mathbf{y}$ 都是完整的文本，这意味着奖励模型需要基于整个序列的语义来给出评判。在具体实现中，我们会将 $\mathbf{x}$ 和 $\mathbf{y}$ 拼接后送入模型，模型在序列的每个位置都会产生一个输出，但我们只取序列末尾（即 $\mathbf{y}$ 的最后一个 token 之后）的表示来计算最终奖励；中间位置的输出可以约定为 0 或其他占位值。为保持符号简洁，下文直接用 $r(\mathbf{x}, \mathbf{y})$ 表示奖励模型 $\text{Reward}(\mathbf{x}, \mathbf{y})$ 。

实现奖励模型的方式有很多。一种简单的做法是基于预训练的大语言模型来构建。具体地，我们将 $\mathbf{x}$ 和 $\mathbf{y}$ 拼接成一个单一的 token 序列 $\text{seq}_{\mathbf{x}, \mathbf{y}} = [\mathbf{x}, \mathbf{y}]$ ，将其输入一个预训练的 LLM，并从最顶层 Transformer 层的每个位置获取表示。我们取出最后一个位置的表示 $\mathbf{h}_{\text{last}}$ ，并通过一个线性变换将其映射为标量：

$$r(\mathbf{x}, \mathbf{y}) = \mathbf{h}_{\text{last}} \mathbf{W}_r \quad (3.27)$$

其中 $\mathbf{h}_{\text{last}}$ 是 d 维向量， $\mathbf{W}_r$ 是 $d \times 1$ 的线性映射矩阵。该架构如图 3.11 所示。

训练奖励模型的第一步是收集关于若干生成输出的人类反馈。给定输入 $\mathbf{x}$ ，我们先用 LLM 生成多个候选输出 $\{\mathbf{y}_1, \dots, \mathbf{y}_N\}$ 。人类反馈通常可以通过以下几种方式获得：

- **成对比较（成对排序）**：给定两个不同的输出，人类专家指出哪一个更好。
- **评分**：人类专家为每个输出单独给出一个数值分数（例如 1–5 分）。

- **列表排序：**人类专家直接对一组输出进行整体排序。

这里我们聚焦于成对比较反馈，因为它是 RLHF 中最简单且最常用的人类反馈形式之一。在该设置下，每次从候选池 $\{\mathbf{y}_1, \dots, \mathbf{y}_N\}$ 中随机抽取两个输出 $(\mathbf{y}_a, \mathbf{y}_b)$ ，呈现给人类专家，并要求其根据清晰度、相关性、准确性等标准表明偏好。偏好结果可以编码为二元标签： $\mathbf{y}_a \succ \mathbf{y}_b$ 表示 $\mathbf{y}_a$ 更受偏好， $\mathbf{y}_b \succ \mathbf{y}_a$ 则相反。

为了从成对比较数据中学习奖励模型，我们需要一个将偏好结果与奖励分数联系起来的概率模型。*Bradley-Terry model* [?] 正是这样一种简单且广泛使用的模型，它估计一个项目比另一个项目更受偏好的概率。适配到当前符号， $\mathbf{y}_a$ 比 $\mathbf{y}_b$ 更受偏好的概率可写为

$$\begin{aligned} \Pr(\mathbf{y}_a \succ \mathbf{y}_b \mid \mathbf{x}) &= \frac{e^{r(\mathbf{x}, \mathbf{y}_a)}}{e^{r(\mathbf{x}, \mathbf{y}_a)} + e^{r(\mathbf{x}, \mathbf{y}_b)}} \\ &= \frac{e^{r(\mathbf{x}, \mathbf{y}_a) - r(\mathbf{x}, \mathbf{y}_b)}}{e^{r(\mathbf{x}, \mathbf{y}_a) - r(\mathbf{x}, \mathbf{y}_b)} + 1} \\ &= \text{Sigmoid}(r(\mathbf{x}, \mathbf{y}_a) - r(\mathbf{x}, \mathbf{y}_b)) \end{aligned} \quad (3.28)$$

训练奖励模型时，我们希望奖励模型赋予被偏好输出更高的分数，也就是最大化上述偏好概率。由此可以得到基于 Bradley-Terry 模型的损失函数

$$\mathcal{L}_r(\phi) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \sim \mathcal{D}_r} [\log \Pr(\mathbf{y}_a \succ \mathbf{y}_b \mid \mathbf{x})] \quad (3.29)$$

其中 $(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b)$ 采样自包含输入及其偏好输出对的人类标注数据集 $\mathcal{D}_r$ ， ϕ 表示奖励模型的参数（包括 Transformer 解码器的参数和线性映射矩阵 $\mathbf{W}_r$ ）。实际中通常假设样本均匀采样，因此可以用求和代替期望：

$$\mathcal{L}_r(\phi) = -\frac{1}{|\mathcal{D}_r|} \sum_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \in \mathcal{D}_r} \log \Pr(\mathbf{y}_a \succ \mathbf{y}_b \mid \mathbf{x}) \quad (3.30)$$

训练的目标是找到最小化该损失的最优参数 $\hat{\phi}$ ：

$$\hat{\phi} = \arg \min_{\phi} \mathcal{L}_r(\phi) \quad (3.31)$$

由于奖励模型本身也是一个 LLM，我们可以直接复用标准的 Transformer 训练流程，只需将交叉熵损失替换为公式 (3.30) 中的成对比较损失。训练完成后，我们便可用训练好的奖励模型 $r_{\hat{\phi}}(\cdot)$ 来为目标 LLM 的对齐提供监督信号。

值得注意的是，尽管我们使用成对比较数据来训练奖励模型，但在后续的对齐阶段，奖励模型是独立地为每个输入-输出对打分的。成对排序的训练目标使奖励模型对输出之间的细微质量差异足够敏感，而我们依赖其输出的连续标量分数来指导策略优化。这样做的一个优势在于：无论训练时选用何种具体的排序损失，只要最终能获

得一个可靠的标量奖励函数，它就可以无缝嵌入到统一的 RLHF 对齐框架中。

3.4.3 基于 PPO 的强化学习人类反馈训练

有了奖励模型之后，下一步就是让大语言模型学会生成高奖励的回复。直观地，我们希望找到策略 π_θ ，使其在输入分布下的期望奖励最大：

$$J(\theta) = \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})} [r(\mathbf{x}, \mathbf{y})], \quad (3.32)$$

同时还要保持生成文本的流畅与合理，避免为追求奖励而输出无意义内容。直接优化上式几乎不可能，因为文本生成是离散且不可微的。强化学习中的策略梯度定理为此提供了可行方案。

策略梯度方法的核心思想是：通过采样估计目标函数 $J(\theta)$ 的梯度，再利用梯度上升更新参数。为了降低梯度估计的方差，通常不直接使用原始奖励，而是引入优势函数 $A(s_t, a_t)$ ，衡量在状态 s_t 下采取动作 a_t 比平均水平好多少。优势可通过一个独立的价值模型 V 来近似，例如用时序差分误差 $r_t + \gamma V(s_{t+1}) - V(s_t)$ 作为估计。在此框架下，一条轨迹 $\tau = (s_1, a_1, \dots, s_T, a_T)$ 的效用可写为

$$U(\tau; \theta) = \sum_{t=1}^T \log \pi_\theta(a_t | s_t) A(s_t, a_t), \quad (3.33)$$

优化目标即最小化负期望效用：

$$\mathcal{L}(\theta) = -\mathbb{E}_\tau [U(\tau; \theta)]. \quad (3.34)$$

这个形式的直观解释很清晰：若某动作的优势为正，就提高其对数概率；若为负，则压低概率。

将上述形式迁移到语言生成任务，输入为 $\mathbf{x}$ ，模型逐 token 生成回答 $\mathbf{y} = (y_1, \dots, y_T)$ 。状态是已生成的前缀 $(\mathbf{x}, \mathbf{y}_{<t})$ ，动作是下一个 token y_t 。损失函数变为

$$\mathcal{L}(\theta) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y})} \left[\sum_{t=1}^T \log \pi_\theta(y_t | \mathbf{x}, \mathbf{y}_{<t}) A(\mathbf{x}, \mathbf{y}_{<t}, y_t) \right]. \quad (3.35)$$

在典型的 RLHF 设定中，数据集只有输入 $\mathbf{x}$ ，输出 $\mathbf{y}$ 由策略 π_θ 自身采样，所以实际使用的损失为

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})} [U(\mathbf{x}, \mathbf{y}; \theta)]. \quad (3.36)$$

然而，每次更新都从当前策略重新采样所有输出，计算开销极大。一种更高效的做法是：固定一个旧策略（称为参考策略 $\pi_{\theta_{\text{ref}}}$ ）来收集数据，再利用重要性采样将旧

数据重用于新策略的评估。对于完整序列，有如下等价关系：

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) \right]. \quad (3.37)$$

该形式称为替代目标，它将采样和评估解耦，让我们可以用同一批旧样本多次更新策略。落实到 token 层面，效用函数中的对数概率被替换为当前策略与参考策略的概率比：

$$U(\tau; \theta) = \sum_{t=1}^T \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{ref}}}(a_t | s_t)} A(s_t, a_t). \quad (3.38)$$

比率大于 1 意味着当前策略比参考策略更偏好该动作，反则反之。

单纯使用概率比会带来新的问题：比率波动剧烈，梯度的方差可能非常大，导致训练不稳定。为解决这一问题，近端策略优化（PPO）同时引入了两种约束。第一，对单步概率比进行截断，将其限制在 $[1 - \epsilon, 1 + \epsilon]$ 的区间内，避免某一步更新幅度过大：

$$U_{\text{clip}}(\tau; \theta) = \sum_{t=1}^T \min \left(\frac{\pi_{\theta}}{\pi_{\theta_{\text{ref}}}}, \text{clip} \left(\frac{\pi_{\theta}}{\pi_{\theta_{\text{ref}}}}, 1 - \epsilon, 1 + \epsilon \right) \right) A(s_t, a_t). \quad (3.39)$$

（为简洁省略了时间下标；实际使用中会根据优势的正负分别采用不同的截断方式，使更新更为保守。）第二，在目标函数中加入显式的 KL 惩罚项，强制当前策略的概率分布整体不要偏离参考策略太远。惩罚项取两策略在序列上的对数概率差：

$$\text{Penalty} = \sum_{t=1}^T \left[\log \pi_{\theta}(a_t | s_t) - \log \pi_{\theta_{\text{ref}}}(a_t | s_t) \right]. \quad (3.40)$$

将二者结合，得到 PPO 的最终目标

$$U_{\text{ppo}}(\tau; \theta) = U_{\text{clip}}(\tau; \theta) - \beta \text{Penalty}, \quad (3.41)$$

其中 β 控制惩罚的强度。这一设计既防止了局部更新过于激进，也在全局上保证了策略平稳演进，因此被广泛用于大语言模型的 RLHF 训练。

映射回语言模型的符号，对于输入 $\mathbf{x}$ 和输出 $\mathbf{y}$ ，完整的 PPO 目标可以写成

$$\begin{aligned} U(\mathbf{x}, \mathbf{y}; \theta) = & \sum_{t=1}^T \min \left(\frac{\pi_{\theta}(y_t | \mathbf{x}, \mathbf{y}_{<t})}{\pi_{\theta_{\text{ref}}}(y_t | \mathbf{x}, \mathbf{y}_{<t})}, \text{clip}(\cdots, 1 - \epsilon, 1 + \epsilon) \right) A(\mathbf{x}, \mathbf{y}_{<t}, y_t) \\ & - \beta \sum_{t=1}^T [\log \pi_{\theta}(y_t | \cdot) - \log \pi_{\theta_{\text{ref}}}(y_t | \cdot)]. \end{aligned} \quad (3.42)$$

相应地，策略训练的损失为 $\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot | \mathbf{x})} [U(\mathbf{x}, \mathbf{y}; \theta)]$ 。其中优势 A 由价值模型提供，奖励则来自自己训练好的奖励模型。

实现上述训练流程需要同时维护四个基于 Transformer 解码器的模型：

- **奖励模型** r_ϕ : 将输入与输出拼接映射为一个标量奖励，由人类偏好数据训练得到。
- **价值模型** V_ω : 预测从当前状态开始的未来累积奖励，为计算优势提供基线，常与奖励模型共享架构。
- **参考模型** $\pi_{\theta_{\text{ref}}}$: 策略训练的起点（如经过指令微调的模型），在 RLHF 过程中参数冻结，为重要性采样和 KL 惩罚提供基准。
- **目标策略** π_θ : 即我们需要训练的语言模型，在奖励与价值的双重监督下不断更新。

这些模型的训练是交替进行的：首先用预训练模型初始化上述四个模型；接着收集人类偏好数据训练奖励模型；然后固定奖励模型和参考模型，同时优化价值模型和策略。每一步中，策略生成输出序列，价值模型给出优势估计，策略通过最小化 PPO 损失来更新，价值模型则通过最小化价值预测的均方误差来更新。循环往复，直到策略能够稳定地生成符合人类偏好的回复。

3.4.4 直接偏好优化

基于 PPO 的 RLHF 虽然有效，但需要同时维护奖励模型、价值模型、参考模型和策略四个网络，训练流程较为复杂。一个自然的想法是：能否跳过显式的奖励建模，直接利用人类偏好数据优化语言模型？**直接偏好优化（DPO）**正是沿着这一思路提出的方法 [?]。图 3.12 展示了标准 RLHF 方法与 DPO 方法的对比。

回顾 RLHF 中策略训练的出发点，它是一个带 KL 惩罚的目标函数

$$\tilde{\theta} = \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} \left[-r(\mathbf{x}, \mathbf{y}) + \beta (\log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - \log \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x})) \right], \quad (3.43)$$

其中 $r(\mathbf{x}, \mathbf{y})$ 是奖励模型给出的分数，超参数 β 控制策略与参考模型 $\pi_{\theta_{\text{ref}}}$ 的接近程度。该目标本质上鼓励模型生成高奖励且不偏离太远的回复。

这个优化问题存在一个重要的闭式解。通过将目标视为关于分布 π_θ 的泛函并求极值，可以证明最优策略 π^* 满足

$$\pi^*(\mathbf{y}|\mathbf{x}) \propto \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp\left(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y})\right). \quad (3.44)$$

也就是说，最优策略是在参考策略的基础上，按奖励的指数倍数重新加权得到的分布。对该式两边取对数并整理，可以反过来把奖励表示为策略的函数

$$r(\mathbf{x}, \mathbf{y}) = \beta \log \frac{\pi^*(\mathbf{y}|\mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x})} + \beta \log Z(\mathbf{x}), \quad (3.45)$$

其中 $Z(\mathbf{x})$ 是与 $\mathbf{y}$ 无关的归一化常数。

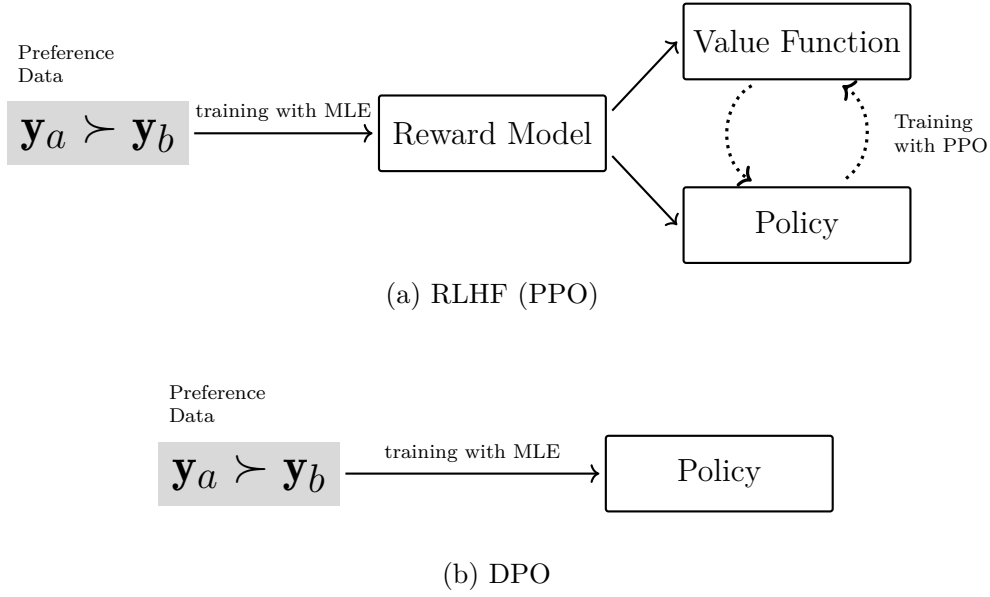


图 3.12: 标准的 RLHF (PPO) 与 DPO。在 RLHF 中，人类偏好数据用于训练一个奖励模型，该模型随后被用于训练策略以及价值函数。在 DPO 中，人类偏好数据的使用更加直接，策略直接在此数据上进行训练，无需训练奖励模型。

现在，将这一奖励表达式代入训练奖励模型时所使用的 Bradley-Terry 偏好模型中。在 RLHF 框架下，我们假设人类偏好 $\mathbf{y}_a \succ \mathbf{y}_b$ 的概率为

$$\Pr(\mathbf{y}_a \succ \mathbf{y}_b \mid \mathbf{x}) = \sigma(r(\mathbf{x}, \mathbf{y}_a) - r(\mathbf{x}, \mathbf{y}_b)), \quad (3.46)$$

其中 σ 是 sigmoid 函数。用式 (3.45) 替换奖励后，两项中的 $\beta \log Z(\mathbf{x})$ 恰好相消，得到

$$\Pr_{\theta}(\mathbf{y}_a \succ \mathbf{y}_b \mid \mathbf{x}) = \sigma\left(\beta \log \frac{\pi_{\theta}(\mathbf{y}_a \mid \mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}_a \mid \mathbf{x})} - \beta \log \frac{\pi_{\theta}(\mathbf{y}_b \mid \mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}_b \mid \mathbf{x})}\right). \quad (3.47)$$

这里的关键在于，归一化常数 $Z(\mathbf{x})$ 被消掉了，偏好概率仅依赖于策略和参考模型输出的对数比。这意味着我们不再需要显式训练奖励模型，可以直接用人类偏好数据训练策略。

由此，DPO 的损失函数直接定义为负对数似然，形式与训练奖励模型时几乎一致，只是参数变成了策略的 θ ：

$$\mathcal{L}_{\text{dpo}}(\theta) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \sim \mathcal{D}_r} \left[\log \Pr_{\theta}(\mathbf{y}_a \succ \mathbf{y}_b \mid \mathbf{x}) \right]. \quad (3.48)$$

最小化这一损失，等价于增大被人类偏好的回复相较于参考模型的概率比，同时压低不被偏好的回复的概率比。

与 PPO 相比，DPO 将整个对齐过程简化为一个**监督式的分类任务**：只需从数据集中读取人类偏好对，直接更新语言模型参数，无需在线采样、价值网络和奖励网络。这使得 DPO 的训练更加稳定、样本效率更高，且易于实现。当然，DPO 本质上是一

种离线方法，无法像在线 RL 那样通过主动探索发现超越固定数据集的新策略，但它凭借极简的流程已经成为目前最流行的人类偏好对齐方法之一。

重点提示

直接偏好优化（DPO）省去了奖励模型的训练和强化学习采样，将偏好对齐转化为一个分类损失：通过比较两个输出相对于参考模型的概率比来直接更新策略。这一方法简化了流程且训练更稳定，但本质上是离线的，无法像在线 RL 那样探索新路径。理解其与 RLHF 的异同是掌握对齐技术的关键。

3.4.5 偏好数据自动生成

基于人类偏好学习是当前对齐大语言模型有效且流行的方法，但标注偏好数据成本高昂。人类反馈不仅扩展性有限，还因其固有的主观性容易引入偏见。为此，AI反馈方法可以突破人类标注的限制，提升可扩展性与一致性。

生成偏好数据与生成指令微调数据类似，可直接利用大语言模型。给定输入，先让模型生成成对输出，再提示其为每对输出标注偏好。下面是一个让大语言模型为一对客服回复标注偏好标签的示例。

设想一个客户服务场景，客户提出了如下请求。你将审阅两个回复，并指出哪一个更优。请注意，好的回复应礼貌、清晰、简洁，直接回应客户的关切，提供有用的信息或解决方案，并保持积极的语气。

请求：

您好，我注意到我的订单还没到，按计划几天前就该送达了。能否帮我查一下目前的状态？谢谢！

回复 A：

非常抱歉给您造成延误，我完全理解这多么令人失望。我们正在尽最大努力尽快为您解决。

回复 B：

嗨，这种事难免的！您的包裹该到的时候自然会到，不用着急。

偏好回复 A。

收集到偏好标签后，即可结合输出对和输入来训练奖励模型。为提升标注质量，可加入示例或采用思维链（CoT）等高级提示技术，例如在提示中展示带有推理过程的偏好示例。

除了偏好标签，我们还可以获得每个标签的概率 [?]。一种简单做法是从模型输出的概率中提取标签 token（如“A”“B”）的概率，再使用 Softmax 等归一化技术转

换为分布，以此作为训练奖励模型的逐点监督信号。

数据生成虽然容易扩展，但准确性和多样性依然关键，这不仅关乎偏好标注，还涉及模型输入与输出的质量，通常需要借助不同模型、多样提示和上下文示例来保证 [?]。 [?] 指出，无论通过人类反馈还是 AI 反馈训练，成对偏好数据的多样性都十分重要。

AI 反馈高度可扩展且相对客观，但更适合有明确客观指标的任务。反之，当需要对齐人类价值观、主观偏好或复杂情境时，人类反馈更具优势。实践中可将二者结合，兼顾人类洞察与 AI 反馈的可扩展性。

3.5 大语言模型推理增强方法

3.5.1 推理概述

历史上，人工智能研究将推理分为三种主要的逻辑形式：**演绎**（从已知前提推导具体结论，如数学证明）、**归纳**（从具体例子推断一般规则或模式）和**溯因**（为一组观察结果寻找最合理的解释） [?]。长期以来，主流机器学习和自然语言处理（NLP）研究主要集中在归纳任务上。例如，文本分类和命名实体识别等经典 NLP 应用可以被视为归纳任务：模型被训练以从大型标注数据集中泛化出映射函数。

然而，随着大语言模型（LLM）的发展，研究人员正试图解决更困难的推理问题，如复杂数学、智能体规划（agentic planning）和科学发现，这些通常需要严格的演绎和溯因推理。现代 LLM 研究表明，标准的预训练目标（即在海量文本语料库上进行下一个词预测）足够强大，能够帮助模型内化语言和世界的知识，包括潜在的推理能力。然而，主要挑战在于推理在预训练阶段并未被显式建模。因此，当面对复杂查询时，LLM 会尝试直接输出最终答案，而不是执行逐步的逻辑推导。

例如，考虑一个简单的应用题：

如果约翰的年龄是玛丽的两倍，而玛丽在 3 年后将是 10 岁，那么约翰现在多大？

基础的 LLM 可能会通过简单关联关键词“两倍”和“10”输出一个错误的猜测，如“20”。相比之下，具备推理能力的 LLM 可能会在回答前显式生成逻辑链：

玛丽现在的年龄是 $10 - 3 = 7$ 岁。因此，约翰的年龄是 $7 \times 2 = 14$ 岁。

要使用 LLM 解决复杂的演绎和溯因任务，必须显式激活这些潜在的推理能力，无论是通过思维链（CoT）等提示技术，还是通过强化学习等先进的后训练范式。

为了更好地理解推理问题，我们可以使用认知心理学中的双过程理论（dual-process theory）来审视 LLM 的能力：系统 1 和系统 2 思维 [?]。

- **系统 1（直觉思维）：**在人类认知中，系统 1 自动、快速地运行，几乎或根本不需要耗费精力。在 LLM 的语境下，系统 1 思维对应于模型在单次前向传播中直接输出最终答案的范式。这种模式效率很高，通常足以应对简单的问答等归纳任务。然而，当面对复杂的多步问题时，系统 1 思维容易出现逻辑跳跃和幻觉。
- **系统 2（审慎思维）：**这种模式缓慢且有意识，需要逐步的逻辑推导。对于 LLM 而言，当引导模型在推理时分配额外的计算资源以显式生成中间推理路径时，系统 2 思维便被激活。通过将思考过程表示为文本，LLM 能够分解复杂任务，验证中间结论，并在得出最终答案前维持一条连贯的逻辑链。

对于 LLM 推理而言，核心目标是激活系统 2 的能力。为了理解这一点，我们必须超越传统的符号逻辑。在经典人工智能中，推理通常被定义为在结构化知识库上应用硬编码的逻辑规则。然而，对于 LLM 来说，推理是一种内在的文本生成过程。它可以被定义为生成一系列连贯的中间步骤以得出最终答案的能力。在某种意义上，LLM 中的推理是一种为更难的问题动态分配更多推理计算资源的机制。归根结底，LLM 的推理是将潜在的系统 2 思维显式编码为一系列思考步骤（推理步骤）的能力。

推理在 NLP 研究中至关重要，因为它在多个具有挑战性的领域有着广泛的应用：

- **数学：**解决数学问题需要严格的逻辑。模型不能仅仅根据文本模式猜测答案。相反，它需要逐步应用数学规则来计算正确结果。
- **编程：**编写软件远比简单地预测下一个词要困难。模型必须根据特定的用户需求和严格的代码逻辑生成代码。特别是在解决复杂问题时，将抽象需求转化为可运行的代码需要逐步推理。
- **智能体规划：**当人工智能作为自主智能体（如网络助手或机器人）行动时，它必须将庞大、复杂的目标分解为小的、可执行的步骤。它还需要提前规划，并在出现问题时调整其行动。
- **科学发现：**协助科学家需要分析数据、提出新假设并对其进行仔细验证。这在很大程度上依赖于逐步的逻辑演绎和溯因。

3.5.2 测试时缩放

大语言模型的卓越性能在很大程度上得益于训练阶段的计算扩展——通过使用更多数据、更大模型和更多算力来提升能力。然而，对于数学证明、逻辑推理等复杂任务，仅靠训练扩展是不够的。这类任务需要模型按步骤进行逻辑推演，而所有可能

的推理路径是无法在训练中穷尽的。一种更高效的理念是将部分计算资源从训练转移到推断阶段，让模型在运行时动态地构建推理过程，即“花更多时间思考”。这种在模型推断阶段通过增加计算量来提升推理表现的方法，就称为测试时缩放（test-time scaling，也称测试时扩展）。

测试时缩放的核心优势在于计算资源分配的灵活性。对于简单问题，模型可以快速作答；对于困难问题，则可以分配更多的算力，允许模型探索不同的推理思路，从而有更大概率得到正确答案。实践表明，一个规模较小的模型，若在测试时被给予额外的时间进行推理，其表现常常能超越那些被要求立即给出答案的大模型。正因如此，测试时缩放成为增强大语言模型推理能力的一条极具吸引力的技术路径。接下来的两个小节将分别从生成多条推理路径和构建搜索过程这两个角度，介绍实现测试时缩放的典型方法。

采样多条推理路径

实现测试时缩放最直接的方法，就是为同一个输入生成多条不同的推理路径，而非仅生成一条。通过扩大推理路径的候选池，模型就获得了更多“尝试”不同逻辑走向并找到正确答案的机会。

要生成多样化的路径，需要在文本生成中注入可控的随机性。常用的策略是温度采样：在预测下一个词时，模型原本会输出一个概率分布。引入温度参数 $T > 0$ 后，会在归一化指数函数之前对各个词的得分进行缩放。较高的温度会令概率分布变得平坦，使原本低概率的词也有机会被选中，从而鼓励模型探索更广泛的语言选择。为避免完全随机的退化输出，温度采样常与词表截断技术结合：

- **Top- k 采样**：每步只保留概率最高的 k 个候选词，重新分配概率后从中采样。
- **核采样（Top- p 采样）**：动态地选择累计概率刚刚超过阈值 p 的最小候选词集合，适应不同语境下的不确定性。

通过这些策略，对于输入查询 $\mathbf{x}$ ，大语言模型可生成 N 条候选推理路径：

$$\mathcal{Y}^{\text{top}} = \{\hat{\mathbf{y}}^1, \dots, \hat{\mathbf{y}}^N\}. \quad (3.49)$$

这里的 N 代表了测试时计算预算。 N 越大，探索的路径就越多。接下来，需要从这组候选路径中选出最优的一条或综合出最终答案。两种经典方法是 Best-of- N （ N 选优）和自一致性。

1. Best-of- N (BoN) 该方法引入一个验证器 v 为每条候选路径打分，分值反映其逻辑合理性与正确性。最终选择得分最高的路径：

$$\hat{\mathbf{y}}_{\text{best}} = \arg \max_{\hat{\mathbf{y}} \in \{\hat{\mathbf{y}}^1, \dots, \hat{\mathbf{y}}^N\}} v(\mathbf{x}, \hat{\mathbf{y}}). \quad (3.50)$$

可以将其类比为一次课堂测验： N 个学生各自写下解题过程，老师（验证器）批阅所有草稿并挑出得分最高的一份。显然，Best-of- N 的效果严重依赖验证器的质量。如果验证器容易被表面上正确但逻辑谬误的答案欺骗，就可能导致选中的路径不可靠。验证器的设计将在第 3.5.3 节中详细讨论。

2. 自一致性 自一致性绕过显式的评分器，转而利用“多数真理”的直觉：对于复杂推理题，通常有多条不同的推理路径能导向同一个正确答案，而错误路径的答案则较为分散。因此，我们只需从每条路径中抽取出最终答案 $\hat{a}^k = \text{Extract}(\hat{\mathbf{y}}^k)$ ，然后对 N 个答案进行多数投票：

$$\hat{a}_{\text{sc}} = \arg \max_{a \in A} \sum_{k=1}^N \mathbb{I}(\text{Extract}(\hat{\mathbf{y}}^k) = a), \quad (3.51)$$

其中 $\mathbb{I}(\cdot)$ 是指示函数。这种方法简单且无需额外训练，在答案形式明确的任务中非常有效。

在使用以上方法时，一个不可忽视的实践要点是平衡路径的**多样性与质量**。较高的温度能提升多样性，增加覆盖正确推理的机会，但也可能使单条路径的文本质量下降，甚至出现逻辑混乱。因此，需要根据任务特点仔细调节采样参数（温度 T 以及 k 或 p 的值），在探索的广度与单路输出的可靠性之间找到最优点。

重点提示

测试时缩放通过增加推理时的计算量（如生成多条候选路径或进行树搜索）来提升复杂任务表现。其中，自一致性利用多数投票筛选答案，无需额外验证器；而 Best-of- N 则依赖可靠的评分模型。实践时需平衡路径多样性与质量，过高的温度虽能增加覆盖却可能引入噪声。缩放收益随计算量增加会趋于饱和，根本上限仍受模型自身知识储备制约。

推理的搜索策略

除了通过增加采样数量来扩展候选池，测试时缩放还可以从更结构化的视角来实现：将寻找最优推理路径视为一个搜索问题。其核心思想是，当我们给予模型额外的算力去“思考”时，本质上是在允许它探索一个由中间推理步骤构成的逻辑空间，并以系统化的方式定位其中的高价值路径。

要应用经典搜索算法，首先需要将大语言模型的推理过程映射为搜索的标准组件：

- **搜索状态**：状态是模型当前已构建的上下文。在步骤 t ，状态 s_t 由原始问题 $\mathbf{x}$ 和已生成的部分推理内容拼接而成。
- **动作**：生成下一个推理步骤，即一段文本片段 $\bar{\mathbf{y}}_{t+1}$ 。

- **状态转移**：将新生成的片段追加到当前状态末尾，形成 $s_{t+1} = (s_t, \bar{y}_{t+1})$ 。
- **状态评估函数**：给当前状态打分，评估其通往正确答案的潜力。评估可基于简单规则，也可使用训练好的神经网络。
- **终止条件**：当模型输出明确的最终答案，或搜索步数达到预设上限时停止。

基于这套形式化描述，多种成熟的搜索算法便可直接用于推理。最简单的如广度优先或深度优先搜索，模型每步可以展开多个候选思路并向前探索，必要时回溯。对于需要长远规划或存在延迟反馈的任务，蒙特卡洛树搜索能够动态平衡探索新想法与利用已知有希望的方向，有时还会结合外部反馈或自我修正来改进搜索质量（例如思维树等工作 [?]）。

需要指出，通过搜索来扩展测试时计算虽能稳定提升推理表现，但其收益并非随搜索量线性增长。当搜索强度超过某个阈值后，性能提升会趋于饱和。根本原因在于，模型本身的知识储备构成了能力的上限：若模型完全不具备解决某一问题所需的核心知识，再庞大的搜索也无法凭空创造出正确的推理逻辑；而过量的搜索还容易生成大量语义重复的路径，徒增计算开销而贡献甚微。

3.5.3 基于可校验奖励的强化学习方法

在推理增强中，我们通常需要一个验证器来判断中间步骤或最终答案的正确性，从而为强化学习提供奖励信号。常见的验证器有三类：给出步骤级标量分数的过程奖励模型（PRM），以自然语言输出评语的生成式验证器，以及依靠外部工具严格判断的形式符号验证器。下面我们依次介绍它们的工作原理和训练方式。

1. 过程奖励模型 验证推理结果主要有两种范式：结果奖励模型（ORMs）和过程奖励模型（PRMs）[?]。ORM 仅评估最终答案，虽然容易训练，但它存在“假阳性”问题。考虑一个简单的数学问题：

求解 x ： $2x + 4 = 10$ 。

模型可能生成如下推理路径：

$2x + 4 = 10$ 意味着 x 必须是 10 的一半，即 5。

但是 $5 - 2 = 3$ 。

因此， $x = 3$ 。

虽然最终答案（3）正确，但逻辑步骤完全是错的。如果只用 ORM，这条路径可能获得高分，从而强化了错误的推理。为了解决这个问题，PRM 会评估路径中每一

步的正确性。给定推理路径 $\mathbf{y} = \{\bar{\mathbf{y}}_1, \dots, \bar{\mathbf{y}}_{n_p}\}$, PRM 基于输入 $\mathbf{x}$ 和之前的步骤 $\bar{\mathbf{y}}_{<t}$, 为当前步骤 $\bar{\mathbf{y}}_t$ 打一个正确性分数 r_t :

$$r_t = v_{\text{prm}}(\bar{\mathbf{y}}_t | \mathbf{x}, \bar{\mathbf{y}}_{<t}). \quad (3.52)$$

这里的 $v_{\text{prm}}(\cdot)$ 是一个训练好的神经网络。整个路径的验证分数可以取所有步骤分的最小值（即只有“最薄弱的一环”也合理，才认为推理正确）:

$$v(\mathbf{x}, \mathbf{y}) = \min_{1 \leq t \leq n_p} v_{\text{prm}}(\bar{\mathbf{y}}_t | \mathbf{x}, \bar{\mathbf{y}}_{<t}). \quad (3.53)$$

当然也可以简单地对步骤分取平均。

训练 PRM 需要步骤级的监督信号，构建训练数据通常有两种方法:

- **人工标注:** 人类专家将模型生成的中间步骤标记为“正确”或“错误”。这种方法准确，但成本高、难以扩展，尤其在需要领域专家的高级任务上。
- **通过 Rollout 自动验证:** 如果我们拥有已知最终答案的数据集，就可以用自动方法估计步骤价值 [?]。从中间步骤 $\bar{\mathbf{y}}_t$ 出发，让模型继续生成多条完整路径。如果这些路径中很大比例最终得到了正确答案 a^* ，我们就可以认为该步骤走在正确的轨道上。步骤的估计值 $\hat{r}_t$ 可以定义为达到正确答案的期望概率:

$$\hat{r}_t = \mathbb{E}_{\mathbf{y}_{>t} \sim \pi_\theta} [\mathbb{I}(\text{Extract}(\mathbf{y}_{>t}) = a^*)]. \quad (3.54)$$

有了这些估计值后，就可以训练 PRM 去预测 $\hat{r}_t$ 。

2. 生成式验证器 上面的验证器都需要修改模型架构并额外训练（例如加上分类头）。另一种自然的想法是直接利用大语言模型的文本生成能力来评价推理步骤，这就是**生成式验证**（generative verification）[?]。它就像一位老师在试卷边缘写评语，而不只是给一个分数。

为了看清它是怎么工作的，我们来看一个例子。假设模型在求解方程组:

问题: 已知 $2x + y = 10$ 和 $x - y = 2$, 求解 x 和 y 。

模型已经生成了两个正确的步骤:

从第二个方程, 用 y 表示 x : $x = y + 2$ 。

将 x 代入第一个方程: $2(y + 2) + y = 10$ 。

接下来它尝试第三步, 但出现了计算错误:

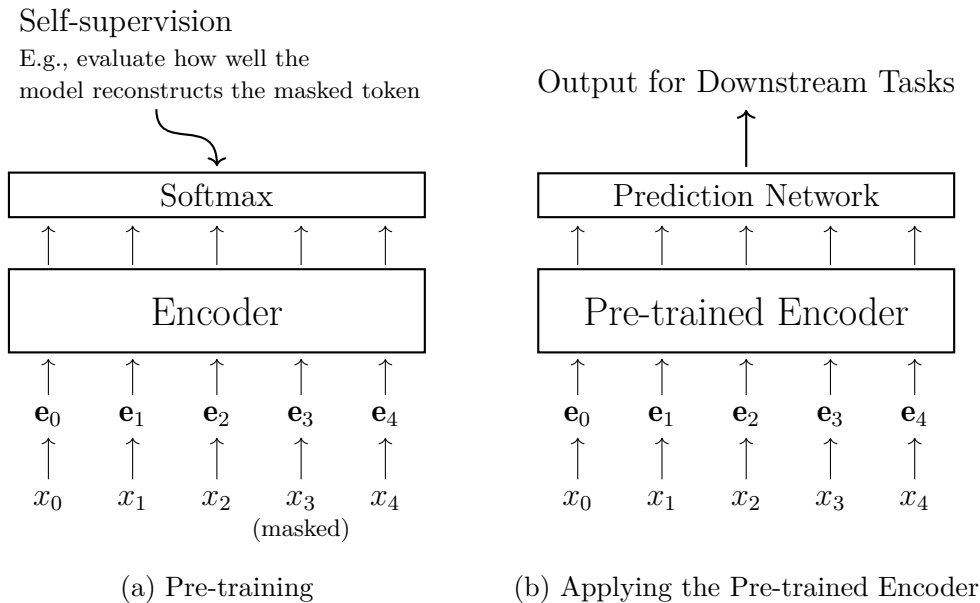


图 3.13: 生成式验证

展开方程得到 $2y + 2 + y = 10$ ，化简为 $3y + 2 = 10$ 。

我们把问题、前两步以及当前步骤拼接起来，再加上一条验证指令（例如“给定前面的步骤，检查当前步骤中的逻辑是否正确并解释你的结论”），一起输入模型。生成式验证器可能会输出这样的文本评论：

评论：错误。分配 2 到 $(y + 2)$ 时必须对括号内每一项都乘以 2，应得到 $2y + 4$ ，而不是 $2y + 2$ 。正确展开为 $2y + 4 + y = 10$ ，化简为 $3y + 4 = 10$ 。

形式化地，设查询为 $\mathbf{z}$ ，验证指令为 $\mathbf{c}$ ，推理历史为 $\bar{\mathbf{y}}_{\leq t}$ 。将它们拼接后输入大语言模型，即可生成评论 $\mathbf{p}_t$ ：

$$\mathbf{p}_t \sim \Pr(\cdot | \mathbf{z}, \bar{\mathbf{y}}_{\leq t}, \mathbf{c}). \quad (3.55)$$

这直接利用了预训练语言模型的生成能力，无需额外训练。图 3.13 展示了这一流程。

当某些搜索算法需要数值分数时，我们可以测量模型在评论开头输出“正确”等指示词的概率，将验证分数定义为：

$$v(\mathbf{z}, \bar{\mathbf{y}}_{\leq t}, \mathbf{c}) = \Pr(y_{\text{anchor}} = \text{“Correct”} | \mathbf{z}, \bar{\mathbf{y}}_{\leq t}, \mathbf{c}). \quad (3.56)$$

与标量奖励模型相比，生成式验证器有几个优点。第一，它可解释——文本评语能清楚地说明判断依据，方便开发者诊断问题。第二，它支持自我纠正——评语通常会指出错误的具体位置和修正方法，模型可以直接据此修改当前状态，而不必丢弃整条路径。第三，它对架构的改动最小——同一个具有强大推理能力的大语言模型可以

同时充当求解器和验证器。

3. 形式符号系统作为验证器 神经验证器虽然强大，但仍然可能出现幻觉或逻辑错误。对于数学、代码等需要严格精确的任务，我们还可以使用**形式符号系统**作为验证器。例如，在代码优化中，形式化等价检查器（如 Alive2）可以自动验证新生成的代码是否保持了与原始代码完全相同的语义。这类系统严格遵循数学和逻辑规则，判断是 100% 可靠的。

要用上形式验证，模型必须将推理步骤生成为可执行代码或形式化陈述，而不仅仅是自然语言。比如，在解决数学问题时，模型可以写一个 Python 脚本来执行某步中间计算，然后由外部工具运行这段代码。验证分数可以直接基于执行是否成功来定义：

$$v(\mathbf{x}, \bar{\mathbf{y}}_{<t}, \bar{\mathbf{y}}_t) = \begin{cases} 1, & \text{if } E(\bar{\mathbf{y}}_t) \text{ executes successfully} \\ 0, & \text{otherwise.} \end{cases} \quad (3.57)$$

如果执行失败，形式环境通常会返回一条错误消息（如 `TypeError`），这条消息可以反馈给模型，帮助它理解错误并修正逻辑。

这种方法完全消除了验证器本身产生“假阳性”的可能。不过，它也存在局限：它要求模型具备较强的编码或形式化能力，将抽象推理转化为严格代码并不容易；而且，对于写文章、规划旅行这类开放式任务，推理很难映射成刚性规则，形式系统并不适用。

3.5.4 从推理大模型到智能体

推理模型不仅提升了语言模型在静态问题上的表现，更催生了一类能够感知、规划、行动与反思的自主智能体。本节介绍三种将“系统 2”推理注入智能体构建的核心方法：通过显式思维链驱动行动的 ReAct 框架、基于失败经验进行自我纠正的 Reflexion 方法，以及将规划建模为树搜索的 Tree of Thoughts 算法。这些方法共同揭示了推理如何成为连接认知与行动的桥梁。

ReAct 框架

传统的提示链方法通常一次性生成完整计划，难以应对动态环境中的不确定性。ReAct (Reasoning and Acting) 框架 [?] 将问题求解建模为一个交替生成“思维”与“动作”的序列，使模型能够在执行过程中不断根据观察结果调整计划。

具体而言，在时间步 t ，模型接收当前上下文 c_t （包括任务描述、工具说明以及

历史交互), 先采样一个思维 T_t , 再基于该思维采样一个动作 A_t :

$$T_t \sim \pi_\theta(\cdot | c_t), \quad (3.58)$$

$$A_t \sim \pi_\theta(\cdot | c_t, T_t). \quad (3.59)$$

环境根据 A_t 返回观察 O_t , 新的上下文更新为 $c_{t+1} = c_t \oplus (T_t, A_t, O_t)$ 。这一循环持续进行, 直到模型输出一个结束动作或达到最大步数。

一个经典的例子是让智能体在电商网站为用户选购笔记本电脑。原始 LLM 可能直接生成一个含错误参数的搜索指令, 而 ReAct 智能体的交互过程如下:

思维: 用户需要视频剪辑用的电脑, 需要高性能 GPU 和至少 32 GB 内存。我先用关键词搜索。

动作: `search("视频剪辑笔记本电脑")`

观察: 返回 50 条商品, 但摘要中没有显示内存信息。

思维: 我需要筛选内存条件。页面上有“高级筛选”按钮, 我可以点击它并设置内存为 32 GB。

动作: `click("高级筛选"); select("内存", "32GB")`

观察: 列表刷新, 剩余 8 个匹配商品。

这种“思维—动作—观察”的循环带来了两个关键优势: (1) 思维为动作提供了可解释的决策依据; (2) 每一轮观察都可以用来修正后续的推理方向, 使系统具备在线适应能力。在实际部署中, 动作集通常被实现为工具 API 调用, 模型通过生成特定格式的文本 (如 `Action: search("...")`) 来触发工具, 环境则将执行结果以文本形式回传。

ReAct 的局限性在于其本质是线性链式推理, 面对包含多个分支或需要长程前瞻的复杂任务时, 可能陷入局部次优甚至无法找到可行路径。为此, 后续方法引入了更结构化的搜索机制。

Reflexion 方法

在现实交互中, 即使是一次完整的 ReAct 轨迹也可能以失败告终。与其从头重试, Reflexion 方法 [?] 赋予智能体“从错误中学习”的能力: 每次失败后, 模型会生成一段文本化的反思, 并将其存入长期记忆, 作为后续尝试的提示。

形式化地, 记任务输入为 $\mathbf{x}$, 第 k 次尝试生成的轨迹为 τ_k , 最终环境给出的二进制成功信号为 $s_k \in \{0, 1\}$ 。若 $s_k = 0$, 系统将当前轨迹与失败信号输入大语言模型, 让其总结失败原因和修正策略, 生成反思文本 R_k :

$$R_k = \text{LLM}(\mathbf{x}, \tau_k, \text{"Task failed. Analyze the error and suggest a fix."}). \quad (3.60)$$

反思 R_k 被追加到记忆 $\mathbf{M}$ 中。在第 $k+1$ 次尝试时，智能体除了任务描述 $\mathbf{x}$ 外，还会看到记忆 $\mathbf{M}$ 作为额外上下文，从而有意识地规避之前犯过的错误。整个过程可以看作在语言空间中进行的一种轻量级“经验回放”。

仍以选购电脑为例，假设某次尝试中智能体在筛选后直接选择了一款外观漂亮的机型，却忽略了 GPU 型号，导致最终购买结果不符合要求。失败后，Reflexion 模块可能输出如下反思：

反思：上次失败是因为我只关注了内存筛选，而没有验证 GPU 是否符合视频剪辑的需求。在下一步选择具体产品时，必须检查其 GPU 是否为独立高性能显卡（如 RTX 4060 及以上）。

在下一轮尝试中，这条反思被加入提示，智能体在筛选后会主动检查每款候选品的 GPU 参数，从而显著提高成功率。

Reflexion 方法的优雅之处在于它完全复用了同一个大语言模型来实现“反思者”的角色，无需额外训练。不过，反思的质量很大程度上依赖于模型本身的诊断能力；对于需要精密逻辑的任务，单纯的文本反思可能不足以纠正常见的计算或逻辑谬误，此时可结合后续的树搜索或形式验证器进行强化。

Tree of Thoughts 算法

当任务具有明确的中间状态和可评价的子目标时，仅靠线性的“思维—动作”链或记忆化的反思仍显不足。Tree of Thoughts (ToT) 算法 [?] 将规划形式化为在“思维树”上的搜索问题：每一个节点代表一个部分推理结果，边代表从当前思考状态到下一步思考的过渡，搜索算法负责在树的生长过程中探索有希望的分支并剪除低质量路径。

ToT 的一次典型执行包含两个核心组件：

- **候选生成**：给定当前状态 s ，大语言模型一次性生成 k 个不同的下一步思维 $\{t^{(1)}, \dots, t^{(k)}\}$ 。
- **状态评估**：使用同一大语言模型（或一个专用的验证器）为每个候选思维打分。一种常见的方法是让模型以特定句式（如“这一步骤正确/可能/不可能的”）进行评价，然后将其对“正确”等锚定词的概率映射为分值：

$$V(s, t^{(i)}) = P_{\theta}(y_{\text{anchor}} = \text{“certain”} \mid s, t^{(i)}, \text{评估指令}). \quad (3.61)$$

根据不同的任务特性，可以选择宽度优先搜索（BFS）或深度优先搜索（DFS）。以 BFS 为例，算法在每一层保留分值最高的 b 个状态，然后基于这些状态继续生成下一步候选，直至达到终止条件（如找到公认的最优答案或预算耗尽）。

Tree of Thoughts 在需要全局规划和回溯的任务（如数学证明、创意写作大纲生成、复杂策略游戏）上取得了远超简单 ReAct 的表现。但它的计算开销更大，因为每次扩展都需要多次调用大语言模型；因此，动态分配计算资源、在简单步骤上减少采样数量，是提升其实用性的重要方向。

总结起来，ReAct 提供了智能体与环境交互的基本骨架，Reflexion 为其注入了从失败中自我改进的元认知能力，而 Tree of Thoughts 则通过结构化搜索赋予了智能体应对复杂规划问题的深度推理手段。这些方法共同构成了当前基于大语言模型构建推理型智能体的技术基石。

3.6 本章小结

本章围绕大语言模型从预训练到应用部署的完整技术链路，构建了一套系统性的认知框架。我们从最基础的语言建模目标出发——利用链式法则将序列联合概率分解为逐步条件概率的乘积，进而理解了仅解码器 Transformer 架构如何为这一概率分解提供可计算的神经网络实现。预训练的本质被还原为一个看似朴素却极其强大的目标：在海量文本上最大化逐 token 预测的对数似然。缩放定律则揭示了模型参数量与训练数据量之间的幂律关系，Chinchilla 定律进一步指出二者必须等比例增长才能达到计算最优，这一洞见直接指导了从 GPT-3 的 0.5T 到 Qwen3 的 36T 训练数据的规模跃迁。分布式训练中的数据并行、流水线并行与张量并行则为这种规模扩张提供了工程上的可行性保障。

在预训练奠定的通用能力之上，我们学习了如何通过提示学习将模型适配到具体任务。零样本、单样本与少样本提示构成了上下文学习的基本谱系，而思维链提示通过强制模型生成中间推理步骤，显著提升了多步推理任务的准确率。提示工程从人工设计走向自动化优化，软提示则突破了离散文本的限制，将提示表示为可学习的连续向量，在效率与灵活性之间取得了平衡。

微调技术是将通用预训练模型转化为任务专用模型的核心手段。有监督微调通过精心设计的指令-响应对激活模型的指令遵循能力，其数据量远小于预训练却能有效唤醒潜在能力，这与表面对齐假说高度一致。参数高效微调方法，尤其是 LoRA，通过低秩分解将权重更新压缩为两个小矩阵的乘积，以极少的可训练参数达到接近全量微调的效果，且推理时零额外延迟，使大模型微调真正走向了普及化。

偏好对齐技术解决了模型“能做什么”与“应该做什么”之间的鸿沟。RLHF 通过人类偏好排序训练奖励模型，再以 PPO 等强化学习算法优化策略，使模型输出符合人类期望。DPO 则通过数学推导将奖励建模与策略优化合二为一，将复杂的四模型训练流程简化为一个监督式分类任务，大幅降低了对齐的工程门槛。偏好数据的自动生成进一步缓解了对人类标注的依赖，使对齐过程具备了更强的可扩展性。

推理增强是本章的最后一个主题，也是当前最活跃的研究前沿。测试时缩放将计算资源从训练阶段转移到推断阶段，通过采样多条推理路径或构建结构化搜索来

动态分配“思考时间”。基于可校验奖励的强化学习为推理过程提供了可靠的监督信号，过程奖励模型、生成式验证器和形式符号系统各有适用场景。从推理模型到智能体的演进——ReAct 的“思维-动作-观察”循环、Reflexion 的失败反思机制、Tree of Thoughts 的树搜索规划——标志着大语言模型正从被动的文本生成器走向能够感知、规划、行动与自我修正的自主系统。

展望未来，大语言模型的技术演进仍面临诸多开放性问题的挑战。缩放定律尚未触及天花板，但算力与数据的边际成本正在攀升，如何在有限资源下实现更高效的训练与推理将是持续的工程挑战。对齐问题远未解决，人类价值观的复杂性与动态性决定了这不可能是一次性的训练目标，而需要模型在与真实世界的持续交互中不断校准。推理能力的边界仍在拓展，如何将形式化验证的严谨性与神经网络的灵活性深度融合，如何让智能体在开放环境中实现长程规划与鲁棒决策，都是亟待突破的方向。此外，多模态融合、长上下文处理、模型安全与可解释性等议题也将在后续章节中逐步展开。本章所建立的预训练、提示、微调、对齐与推理增强的技术体系，既是理解当前大模型能力边界的钥匙，也是通往更强大、更安全、更通用的人工智能系统的起点。

课后练习

1. 设一个 token 序列为 $\{x_0, x_1, x_2, x_3\}$ ，其中 $x_0 = \langle s \rangle$ 。根据链式法则，写出该序列的联合概率 $\Pr(x_0, x_1, x_2, x_3)$ 的分解式。
2. 判断下列说法是否正确，并简要说明理由：“根据 Chinchilla 缩放定律，若要提升模型性能，应优先增加模型参数量，而非增加训练数据量。”
3. 下面哪个选项是思维链（Chain-of-Thought）提示的核心特征？
A. 在提示中提供多个完整的问答示例
B. 强制模型生成中间的推理步骤后再给出答案
C. 使用可训练的连续向量代替文本指令
D. 要求模型直接输出最终结果以提高效率
4. 在有监督微调（SFT）中，为什么需要将损失函数仅作用于输出部分（即助手的回复），而不作用于输入部分（即用户的指令）？请简要解释。
5. 对于一个参数量为 $d \times k$ 的预训练权重矩阵，若采用 LoRA 进行微调，秩取 r ，则新增的可训练参数量是多少？（用 d 、 k 、 r 表示）
6. 在 RLHF 流程中，Bradley-Terry 模型用于训练奖励模型时，其损失函数使用成对比较数据而非绝对评分数据。这种做法的优势是什么？
7. 某模型在数学推理任务上表现不佳，你作为技术人员，可以从哪些技术层面尝试改进？请至少写出两个方面，并简要说明理由。

第四章 深度学习中的连续动力学机制

作者：张俊翔、叶凯阳、肖桐

参考：Ordinary Differential Equations in Vision and Language

学习目标

- 理解离散状态更新、连续动力系统与数值离散之间的双向联系，掌握常微分方程、初值问题、流映射和数值求解的基本概念。
- 能够从连续深度视角解释残差网络与 Transformer 的表示演化，区分结构类比与严格等价，并理解 Neural ODE 及伴随灵敏度法的基本思想。
- 掌握确定性概率输运的核心工具，能够说明连续性方程、连续归一化流与 Flow Matching 的训练目标及其差异。
- 理解随机微分方程、反向 SDE、分数函数、去噪分数匹配与概率流 ODE 之间的关系，并能比较确定性与随机性生成轨迹。
- 了解连续时间马尔可夫链和离散扩散的基本机制，能够比较嵌入空间扩散、词元空间扩散以及自回归与非自回归生成方案。
- 能够结合稳定性、刚性、误差控制和采样效率，对不同连续动力学模型及其数值实现作出分析与选择。

4.1 连续动力学的数学基础

在前面的章节中，我们已经学习了深度学习的基本组成与计算过程。从模型训练的角度看，优化算法通过多次迭代不断调整模型参数；从模型结构的角度看，输入数据依次经过不同的网络模块，并逐步转化为更高层次的隐藏表示。这些训练和推理过程最终都由计算机执行，因此通常表现为有限个离散步骤上的状态更新。

离散表示能够直接说明系统如何从当前状态到达下一状态，却不会显式描述两个离散步骤之间可能存在的演化过程。例如，当模型参数从 θ_k 更新为 θ_{k+1} 时，离散算法只记录更新前后的参数，并不直接刻画参数在这两个状态之间如何连续变化。类似地，网络前向传播通常只给出不同层上的隐藏表示，而不描述相邻层之间的中间状态。

连续动力学提供了另一种描述方式。其基本思想是，将离散状态看作某条连续轨迹上的采样点，并使用状态的瞬时变化率描述轨迹在每个位置附近的局部演化。在这一视角下，研究对象不再只是彼此分离的状态序列，而是一条由局部变化不断累积形

成的完整轨迹。

需要强调的是，这种连续化并不意味着深度学习模型在计算机中真正以连续方式运行。模型的训练与推理仍由有限次离散运算完成。连续动力学的作用，是为这些离散计算建立一个理想化的连续描述：既可以从离散状态序列出发，研究其可能逼近的连续演化规律；也可以先定义连续动力系统，再通过数值方法将其转化为实际可执行的离散更新。因此，离散计算与连续动力学之间存在如下双向联系：

$$\text{离散状态更新} \xrightarrow{\text{连续化}} \text{连续动力学} \xrightarrow{\text{数值离散}} \text{可执行的离散计算}. \quad (4.1)$$

本节首先以前介绍过的梯度下降为例，说明离散参数更新如何与连续演化规律建立联系；随后介绍常微分方程中的状态、动力学、初始条件和轨迹等基本概念；最后讨论如何通过积分表示完整的状态演化，以及如何利用数值方法近似求解连续动力系统。

重点提示

新概念：连续动力系统用状态的瞬时变化率描述演化规律。状态 $\mathbf{x}(t)$ 表示系统在时刻 t 的位置，向量场 $f(\mathbf{x}, t)$ 给出该位置处的局部速度，初始条件决定从哪一点出发；三者共同确定一条随时间演化的轨迹。

重难点解析：从梯度下降或网络层更新过渡到微分方程，是在步长趋小意义下建立的理想化连续描述，并非任意有限步长下的严格恒等。反过来，用 Euler、Runge-Kutta 等方法求解 ODE 时又会重新得到离散更新；步长、方法阶数与稳定域共同决定近似误差和数值稳定性。

4.1.1 从离散更新到连续演化

在前面的章节中，我们已经介绍了梯度下降方法。设 $\boldsymbol{\theta}_k \in \mathbb{R}^d$ 表示第 k 次迭代后的模型参数， $L(\boldsymbol{\theta})$ 表示训练损失，则梯度下降按照负梯度方向更新参数：

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k), \quad (4.2)$$

其中， $\eta > 0$ 为学习率，用于控制每次参数更新的幅度。

式 (4.2) 描述的是相邻两次迭代之间的离散参数变化。为了从连续演化的角度观察这一过程，可以将学习率 η 看作相邻两个训练时刻之间的间隔，并定义

$$t_k = k\eta, \quad t_{k+1} - t_k = \eta. \quad (4.3)$$

于是，梯度下降可以改写为

$$\frac{\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k}{t_{k+1} - t_k} = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k). \quad (4.4)$$

式 (4.4) 左侧表示模型参数在相邻两个训练时刻之间的平均变化率。

如果将离散参数序列 $\{\theta_k\}$ 看作一条连续参数轨迹 $\theta(t)$ 在时刻 $\{t_k\}$ 上的采样, 并考虑学习率逐渐减小的情形, 则上述平均变化率可以进一步过渡为参数关于训练时间的瞬时变化率:

$$\frac{d\theta(t)}{dt} = -\nabla_{\theta} L(\theta(t)). \quad (4.5)$$

由此, 原本由有限步参数更新描述的训练过程, 可以被理想化为一条随连续时间变化的参数轨迹。离散梯度下降直接给出相邻两次迭代之间的参数变化, 而式 (4.5) 则通过参数的瞬时变化率描述其局部演化规律。

这一例子表明, 离散更新过程在适当条件下可以与连续状态演化建立联系。式 (4.5) 已经具有常微分方程的基本形式。为了进一步理解其中的状态变量、瞬时变化率、动力学函数以及完整演化轨迹, 下一小节将正式介绍常微分方程的基本概念。

4.1.2 常微分方程

上一小节以梯度下降为例, 将离散的参数更新序列理想化为一条连续参数轨迹 $\theta(t)$, 并使用参数关于训练时间的瞬时变化率描述其局部演化。其中, 梯度流

$$\frac{d\theta(t)}{dt} = -\nabla_{\theta} L(\theta(t)) \quad (4.6)$$

规定了模型参数在任意时刻应当如何变化, 是连续动力系统的一个具体例子。相较于只记录相邻两个离散状态, 连续动力学更关注系统在任意时刻的局部变化规律。

微分方程正是描述这类连续变化的基本数学工具。它通过建立未知函数与其导数之间的关系, 刻画系统状态如何随自变量发生变化。本节将首先介绍微分方程的基本概念, 并重点讨论常微分方程 (Ordinary Differential Equation, ODE) 的相关技术;

为了直观理解这些概念, 先从最简单的匀速直线运动说起。设一个质点沿直线运动, 令 $t \in \mathbb{R}$ 表示当前时刻, $x(t)$ 表示质点在时刻 t 的位置。在这一表述中, t 是函数 x 的自变量, $x(t)$ 是位置函数在时刻 t 处的取值, 也称为状态变量或因变量。为了简化记号, 在不会引起混淆时, 可以使用 x 来代替 $x(t)$ 。

对于沿直线运动的质点, 速度刻画了位置随时间变化的快慢。用微积分的语言来说, 速度就是位置函数 $x(t)$ 对时间 t 的瞬时变化率, 即

$$\frac{dx(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}. \quad (4.7)$$

该式表示, 当时间间隔 Δt 趋近于零时, 平均速度的极限就是时刻 t 的瞬时速度。

对于匀速直线运动而言, 速度始终为常数 v , 因此我们得到:

$$\frac{dx(t)}{dt} = v. \quad (4.8)$$

这就是一个最简单的**常微分方程** (ordinary differential equation, ODE), 即未知函数 x 只依赖单个自变量, 即时间 t 。相反地, 如果未知函数依赖多个自变量, 例如同时依赖时间和空间, 则相应方程称为**偏微分方程** (Partial differential equation, PDE)。

更一般地, 系统状态 $x(t)$ 的变化率未必是常数, 而是也可能由当前状态和时间共同决定。此时可以用函数 $f(\cdot)$ 表示这种变化的规律:

$$\frac{dx(t)}{dt} = f(x(t), t). \quad (4.9)$$

这是一种更一般的常微分方程的形式。它刻画的不是状态 $x(t)$ 在某一时刻的静态取值, 而是状态随时间变化的瞬时规律; 其中, $f(x(t), t)$ 表示系统在状态 $x(t)$ 和时刻 t 下的变化率。

式 (4.9) 也可以写成微分形式:

$$dx(t) = f(x(t), t) dt. \quad (4.10)$$

这一写法直观地表示: 在一个很小的时间间隔 dt 内, 状态变化量 $dx(t)$ 可以近似为当前变化率与时间增量的乘积。后续介绍数值积分方法和随机过程时, 这种形式会更加方便。

回到匀速直线运动, 由于 $f(x(t), t) = v$, 式 (4.9) 退化为式 (4.8)。对其两边积分可得

$$x(t) = vt + x(0), \quad (4.11)$$

其中 $x(0)$ 是 $t = 0$ 时的初始位置。因此, 若初始位置 $x(0)$ 不同, 即使速度 v 相同, 也会得到不同的运动轨迹; 从几何上看, 该方程的解集是 (t, x) 平面上一族斜率相同、截距不同的直线。这个简单的例子说明, 微分方程的解通常需要结合初始条件才能唯一确定。

当 $f(\cdot)$ 显式依赖于位置 $x(t)$ 时, 系统展现出的变化规律会更加丰富。一个典型例子是线性回归过程: 假设系统状态离平衡点越远, 其回到平衡点的趋势越强。若这种变化率与当前状态成线性关系, 即对于某个常数 $k > 0$, 则有 $f(x(t), t) = -kx(t)$ 。这类方程被称为**线性常微分方程**。这类方程尤其值得关注, 因为它们通常具有闭式解析解, 即可以用显式的表达式直接写出系统状态随时间变化的规律, 而不必完全依赖数值近似方法。

然而, 在许多实际应用中, $f(\cdot)$ 是非线性的。例如质点所处的实际环境导致运动规律包含 $x(t)^2$ 或 $\sin(x(t))$ 等非线性项, 此时对应的方程便属于**非线性常微分方程**。与线性方程相比, 非线性方程通常更难求得闭式解析解, 即使解在理论上存在, 也未必能够写成明确的公式形式。因此, 在处理复杂系统时, 往往需要借助数值方法对其进行近似求解。由于后续章节讨论的问题大多涉及非线性常微分方程, 如何有效求得近似解将成为我们关注的重点。

需要注意的是, 虽然 $f(\cdot)$ 接收 $x(t)$ 和 t 两个参数, 但 $x(t)$ 本身也是 t 的函数。因此,

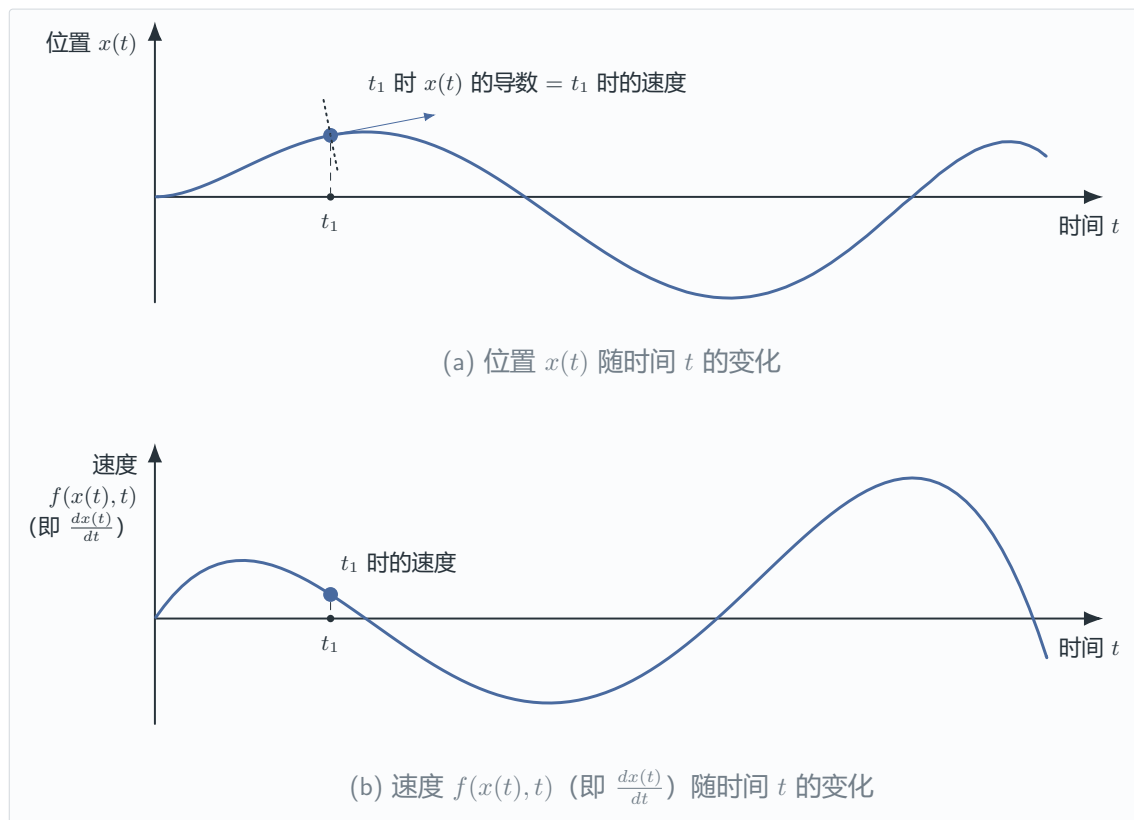


图 4.1: 常微分方程因变量 ($x(t)$) 及其导数 ($\frac{dx(t)}{dt}$) 的曲线图。虽然该微分方程需要将 $x(t)$ 作为 $f(\cdot)$ 的输入, 但 $x(t)$ 本身由时间驱动。因此, 若轨迹固定, 则速度 $f(\cdot)$ 隐式地成为仅关于 t 的函数。故可直接绘制速度 $f(\cdot)$ 与 t 的关系曲线。

沿着任意一条给定的轨迹, 速度都可以看作关于 t 的复合函数。在这种情况下, 我们可以将 $f(\cdot)$ 绘制为关于 t 的函数, 其中任意时刻的函数值都对应于导数 $\frac{dx(t)}{dt}$ 。图 4.1 给出了一个简单的例子。

然而在实际问题中, 状态轨迹 $x(t)$ 通常是未知的, 因此函数 $f(\cdot)$ 对时间 t 的具体依赖关系也往往难以直接确定。此外, $f(\cdot)$ 对时间的显式依赖还会带来额外的复杂性; 此时质点的运动规律并非固定不变, 而是会随时间发生变化。为了强调这一点, 我们可以采用记号 $f_t(\cdot)$ 表示 $f(\cdot, t)$ 。这意味着, 质点的速度不仅取决于其当前位置, 也取决于当前时刻; 即使质点在不同时间到达相同的位置, 也可能具有不同的速度。

图 4.1 分别展示了状态 $x(t)$ 及其导数 $\frac{dx(t)}{dt}$ 随时间的变化。虽然常微分方程中的函数 $f(\cdot)$ 需要以 $x(t)$ 作为输入, 但 $x(t)$ 本身由时间驱动。因此, 当状态轨迹确定后, 速度 $f(x(t), t)$ 便可以隐式地看作仅关于时间 t 的函数。

从更根本的角度看, 这类常微分方程定义了一个**动力系统**。在这一框架下, 变量 $x(t)$ 表示系统的**状态**, 用于描述系统在某一特定时刻的整体情况; 函数 $f(\cdot)$ 则刻画系统的**动力学**, 也常被称为**动力学规则**, 用于说明系统状态在任意瞬间如何发生变化。

从这一视角出发, 函数是否显式依赖时间, 实际上对应着对动力学规则本身的分类。如果动力学规则保持不变, 并且只依赖于系统当前的状态, 即 $\frac{dx(t)}{dt} = f(x(t))$, 那么该系统称为**自治系统**; 如果动力学规则本身会随时间变化, 即 $\frac{dx(t)}{dt} = f(x(t), t)$, 那

么该系统称为非自治系统。

非自治系统能够描述更加复杂的动态过程，但其分析和求解通常也比自治系统更加困难。例如，在第 4.2 节中，我们将说明一组堆叠的 Transformer 层可以从非自治动力系统的角度加以理解。

符号	含义
t	自变量，通常表示时间。
$x(t)$	因变量，表示系统在时刻 t 的状态。
$\frac{dx(t)}{dt}$	状态的导数，表示状态的瞬时变化率，例如速度。
$f(x(t), t)$	控制系统演化的函数，也称为动力学规则，用于描述系统的动力学。
$\frac{dx(t)}{dt} = f(x(t), t)$	非自治常微分方程，其动力学显式依赖于时间。
$\frac{dx(t)}{dt} = f(x(t))$	自治常微分方程，其动力学仅依赖于系统状态，即动力学规则不随时间变化。
$dx(t) = f(x(t), t) dt$	常微分方程的微分形式，常用于数值积分和随机微积分。
$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$	向量值常微分方程，用于描述状态向量 $\mathbf{x}(t) \in \mathbb{R}^d$ 所表示的多维系统，其中 $f(\cdot)$ 定义了一个向量场。

表 4.1: 常微分方程中的基本概念

到目前为止，我们的讨论主要集中于标量变量和标量函数，即状态 $x(t)$ 是一个标量变量，函数 $f(\cdot)$ 是一个标量函数。一个自然的推广是在向量空间上定义常微分方程。令 $\mathbf{x}(t) \in \mathbb{R}^d$ 表示一个包含 d 个不同分量的向量状态，例如神经网络某一层中的神经元激活值。相应地，函数 $f(\cdot)$ 被定义为一个向量值映射 $f: \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ 。此时，常微分方程可以写为

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t). \quad (4.12)$$

在式 (4.12) 中，导数 $\frac{d\mathbf{x}(t)}{dt}$ 表示对向量 $\mathbf{x}(t)$ 的各个分量分别求导。从标量状态推广到高维向量状态后，常微分方程具有了更加丰富的几何解释。在前面的例子中，标量函数只表示一条曲线的斜率；而向量函数 $f(\cdot)$ 则在状态空间中定义了一个**向量场**。对于状态空间中的每一个点 $\mathbf{x}(t)$ ，向量 $f(\mathbf{x}(t), t)$ 都给出了系统瞬时运动的大小和方向。

图 4.2 给出了两个二维向量场的例子。每个箭头表示系统位于相应位置时的瞬时运动方向与大小；从不同初始状态出发并沿箭头方向连续推进，便得到由同一向量场诱导的不同状态轨迹。

尽管我们将系统状态推广到了 d 维空间，常微分方程的基本概念和性质仍然成立。

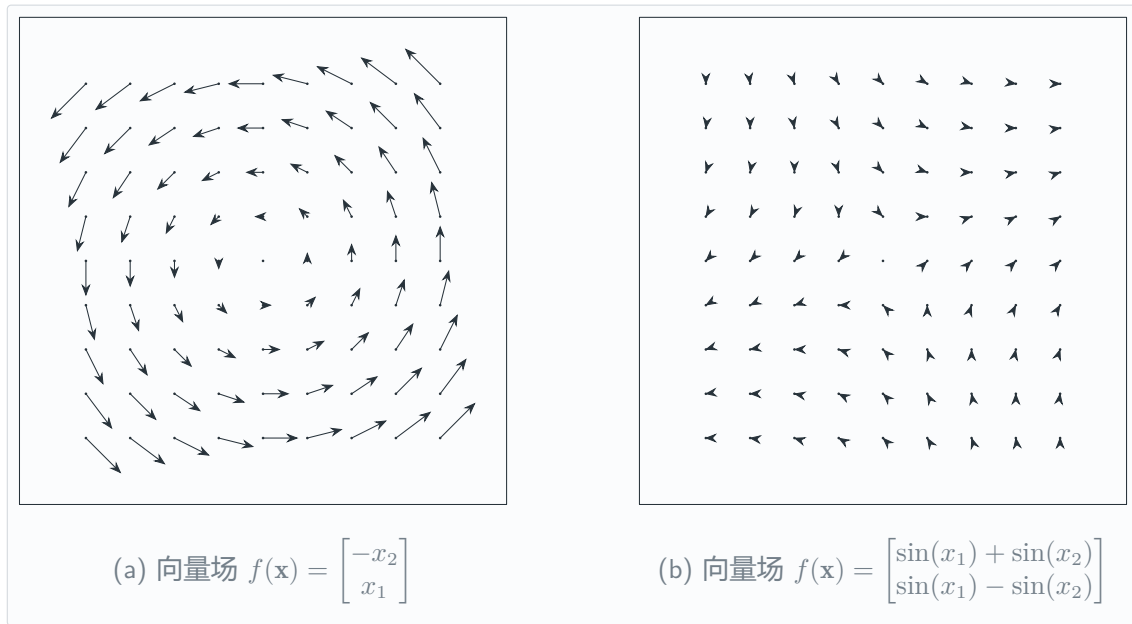


图 4.2: 二维平面中向量场的两个示例。每个箭头表示状态位于该位置时的瞬时变化方向与大小；沿向量场积分可得到系统的状态轨迹。(a) 线性向量场产生圆形流线；(b) 非线性向量场产生更复杂的流线。

为了使本节的表述更加简洁，后续讨论仍将主要采用标量记号；而在之后的章节中，我们将主要使用向量值常微分方程来描述更加复杂的问题。表 4.1 总结了目前介绍的主要概念。

4.1.3 常微分方程的求解：积分表示与数值方法

我们已经说明，常微分方程（ODE）刻画的是状态 $x(t)$ 的导数。一个自然的问题是：为什么我们要用导数来定义系统，而不是直接描述状态 $x(t)$ 本身？事实上，如果整个轨迹 $x(t)$ 都已知，那么 ODE 的描述是多余的，之所以要引入 ODE，是因为在许多实际应用中，我们无法得到 $x(t)$ 的解析表达式。我们通常只给定系统的动力学函数 $f(\cdot)$ 以及初始条件 $x(0)$ 。这类似于驾驶汽车：我们可以很容易知道当前时刻的速度，但要精确确定相对于起点的位置却很困难。在 ODE 中，速度对应 $f(\cdot)$ ，起点对应 $x(0)$ ，而我们的目标是重建整个轨迹 $x(t)$ 的“路径”。

在这种情况下，我们需要求解常微分方程（ODE）。这通常被称为初值问题（initial value problem, IVP），其中给定一个 ODE 以及一个特定约束（例如初始状态 $x(0)$ ），我们希望从其导数恢复任意时刻 t 的状态 $x(t)$ 。根据微积分基本定理，时刻 t 的状态可以看作初始状态 $x(0)$ 加上从 0 到 t 这段时间内状态的总变化量。这个总变化量可以表示为导数（也就是速度）在区间 $[0, t]$ 上的积分，因此，我们可以通过积分写出 ODE

的解

$$\begin{aligned} x(t) &= x(0) + \int_0^t \frac{dx(\tau)}{d\tau} d\tau \\ &= x(0) + \int_0^t f(x(\tau), \tau) d\tau. \end{aligned} \quad (4.13)$$

如图 4.3 所示, 该公式具有非常直观的几何解释。积分项 $\int_0^t f(x(\tau), \tau) d\tau$ 表示函数 $f(x(\tau), \tau)$ 在 $\tau = 0$ 到 $\tau = t$ 区间上的带符号曲线下的面积。该面积量化了状态 $x(t)$ 在区间 $[0, t]$ 上的总累积变化。因此, 该等式表明: 时刻 t 的状态可以通过将这一总变化 (面积) 加到初始状态 $x(0)$ 上得到。

然而, 这种积分形式在 ODE 求解中带来了计算上的挑战: 尽管方程形式看起来很简单, 但由于被积函数 $f(x(\tau), \tau)$ 依赖于未知的状态轨迹 $x(\tau)$ 本身, 而获得状态轨迹又需要被积函数来积分, 因此无法直接计算。此外, 即使轨迹是显式给定的, 由于动力系统 (例如由深度神经网络建模) 通常具有复杂性与非线性, 解析计算该积分也往往十分困难。因此, 我们必须转向数值方法来近似该积分。其核心思想是离散化: 不再在每一个无穷小时间刻度上连续追踪 $x(t)$ 的演化, 而是在一系列离散时间点 $\{t_0, t_1, \dots, t_N\}$ 上对轨迹进行近似。由于在有限个点上计算函数 $f(\cdot)$ 的代价较低, 因此离散化提供了一种高效获得近似解的方法。

从黎曼积分的角度来看, 离散化可以被理解为对 (4.13) 中的积分进行近似: 将积分区间划分为若干小区间, 并对由导数函数定义的矩形面积进行求和。图 4.4 给出了示意图。为形式化该过程, 我们在区间 $[0, t]$ 上定义时间网格, 该网格包含 $N + 1$ 个点 $t_0 < t_1 < \dots < t_N$, 其中 $t_0 = 0$ 为初始时刻, $t_N = t$ 为终止时刻。

为简化表示, 我们通常假设时间间隔为常数, 即

$$\Delta t = t_{k+1} - t_k. \quad (4.14)$$

其中 Δt 被称为步长 (step size)。

我们的目标是计算一系列状态 $\{\bar{x}_0, \bar{x}_1, \dots, \bar{x}_N\}$, 其中每个 $\bar{x}_k$ 用于近似真实状态 $x(t_k)$ 。从当前状态 $\bar{x}_k$ 到下一状态 $\bar{x}_{k+1}$ 的转移, 需要近似计算动力系统在小区间 $[t_k, t_{k+1}]$ 上的积分。数学上, 该离散模型可以写为递推形式

$$\bar{x}_{k+1} = \bar{x}_k + \text{Approx} \left(\int_{t_k}^{t_{k+1}} f(x(\tau), \tau) d\tau \right). \quad (4.15)$$

其中函数 $\text{Approx}(\cdot)$ 用于近似局部积分。在离散 ODE 模型中, 该函数通常根据 $f(\cdot)$ 在局部几何结构上的取值来定义, 例如

$$\text{Approx} \left(\int_{t_k}^{t_{k+1}} f(x(\tau), \tau) d\tau \right) = \Delta t \cdot \Psi(\bar{x}_k, t_k, \Delta t, f). \quad (4.16)$$

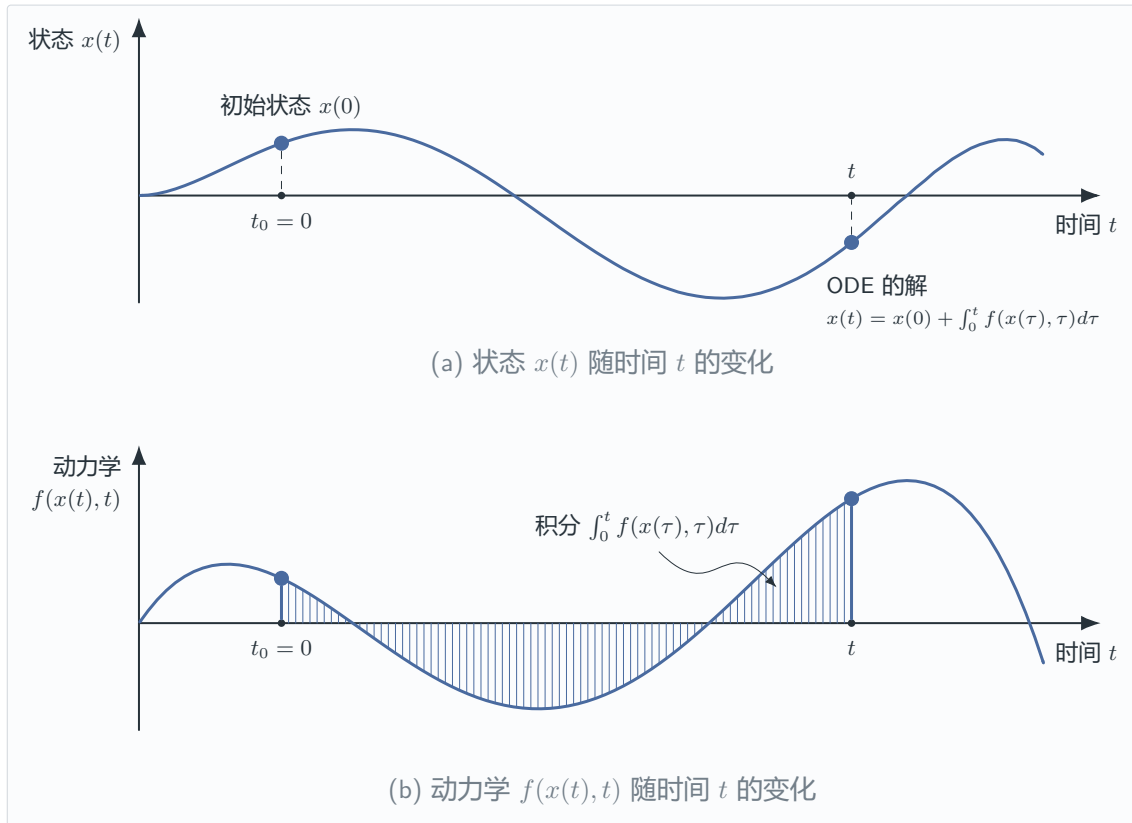


图 4.3: 通过积分求解常微分方程的示意图。蓝色阴影表示动力学函数在区间上的积分，即相对于初始状态的累积变化；将该积分加到初始状态即可得到终止状态。

因此，式 (10) 可以改写为

$$\bar{x}_{k+1} = \bar{x}_k + \Delta t \cdot \Psi(\bar{x}_k, t_k, \Delta t, f). \quad (4.17)$$

其中 $\Psi(\cdot)$ 被称为增量函数 (increment function) 或阶跃函数 (step function)。它利用在离散点上计算得到的 $f(\cdot)$ 来估计区间上的积分。不同的数值方法对应不同的 $\Psi(\cdot)$ 选择。例如，如果 $\Psi(\cdot)$ 取为左端点处的斜率 $f(\bar{x}_k, t_k)$ ，则得到经典的 Euler 方法 (类似左黎曼和)。此时增量函数为动力学函数本身，即 $\Psi(\bar{x}_k, t_k, \Delta t, f) = f(\bar{x}_k, t_k)$ 。代入式 (12) 得到 Euler 方法的更新形式

$$\bar{x}_{k+1} = \bar{x}_k + \Delta t \cdot f(\bar{x}_k, t_k). \quad (4.18)$$

由于每一步仅需一次函数计算，Euler 方法计算效率较高。然而，当步长 Δt 较大或动力系统变化较快时，其精度往往较差。为克服这一问题，数值 ODE 求解方法通常分为两类：高阶单步方法与多步方法。

- 高阶单步方法。在数值分析中，方法的阶数描述误差随 $\Delta t \rightarrow 0$ 的收敛速度。如果局部误差为 $O(\Delta t^{p+1})$ ，则称该方法为 p 阶方法。高阶方法通过在区间 $[t_k, t_{k+1}]$ 内的多个点评估 $f(\cdot)$ 来更准确估计平均斜率。例如，四阶 Runge-Kutta 方法 (RK4)

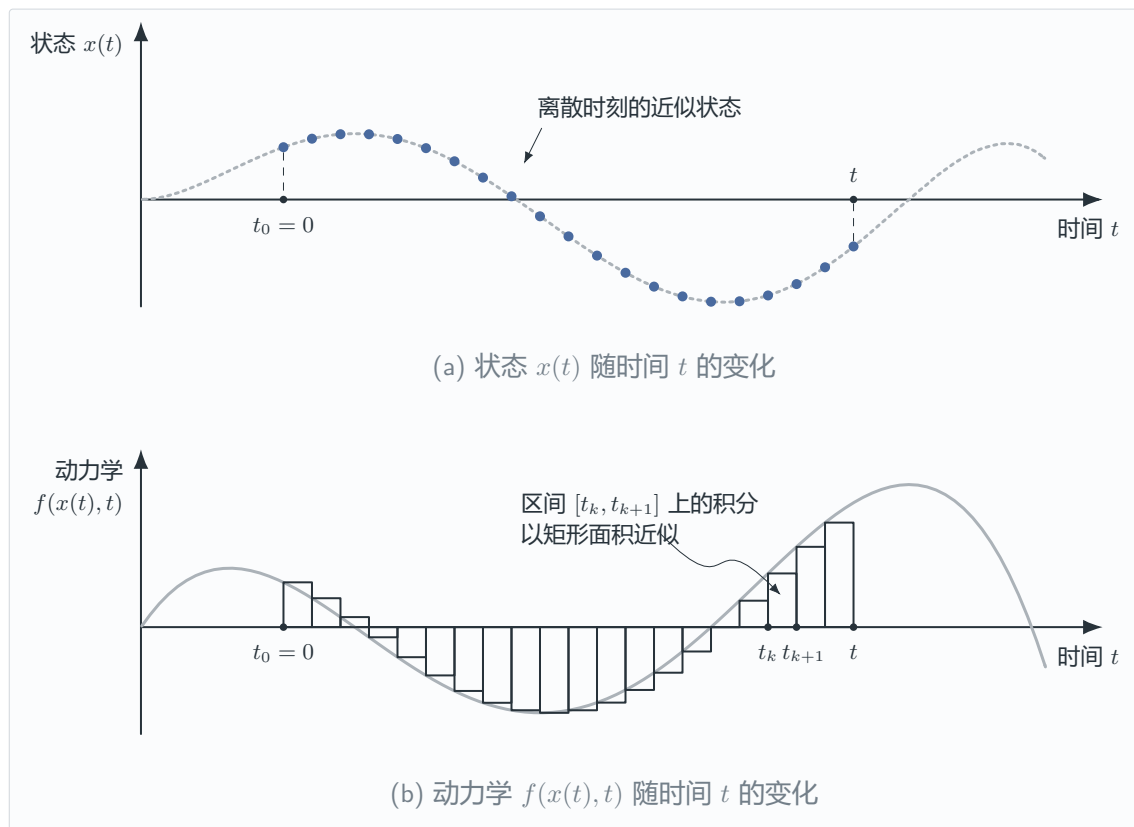


图 4.4: 常微分方程离散化示意图。函数 f 给出状态在各时刻的瞬时变化率；蓝色圆点表示离散时间节点上的状态近似，矩形面积 $\Delta t \cdot \Psi$ 近似一个时间步内的积分增量。

并不只使用当前时刻的斜率来推进状态，而是在每个时间步内计算四个局部斜率：一个位于区间起点，两个用于估计区间中点，另一个用于估计区间终点。随后，RK4 对这四个斜率进行加权平均，并用该平均斜率来近似整个时间步内的状态变化。因此，相比显式欧拉法只使用起点斜率，RK4 能够更准确地刻画一个时间步内轨迹的弯曲变化，从而在不使用极小步长的情况下仍获得较高精度。

- 多步方法。与单步方法不同，多步方法（如 Euler 和 RK4）只依赖当前状态并丢弃历史信息，而多步方法利用轨迹历史信息，通过对过去导数值进行拟合（如多项式拟合）来预测下一步积分。例如 Adams-Bashforth 方法属于此类。其优点是计算效率高，可以复用已有函数评估结果；但缺点是非自启动，需要先用单步方法初始化前若干步。

此外，还存在改进 ODE 求解器的其他方法，例如自适应步长方法。在该方法中，步长 Δt 不再固定，而是根据局部误差动态调整。本文不对这些方法展开细节讨论，相关内容可参考数值分析教材。

在进一步讨论之前，需要澄清本文所用记号。通常函数记号 $x(t)$ 表示连续时间状态，而在机器学习与随机过程领域中， x_t 常用于表示时刻 t 的状态。在数值求解中， x_k （或 $\bar{x}_k$ ）表示第 k 步的数值近似解。

在前述讨论中，我们区分了数值近似解 $\bar{x}_k$ 与真实解析解 $x(t_k)$ 。但为了简化记号

并与文献保持一致，后续我们用 x_k 表示第 k 步的状态近似值，即

$$x_{k+1} = x_k + \Delta t f(x_k, t_k). \quad (4.19)$$

有时也会使用 $x(t)$ 来表示对真实状态的近似。

总结而言，给定由动力学 f 和初始状态 $x(0)$ 定义的 ODE 系统，可以将 ODE 求解器视为一个函数 $\phi(\cdot)$ 。对于任意 $t \geq 0$ ，该函数返回状态的近似解，记为

$$x(t) = \phi(f, x(0), t). \quad (4.20)$$

同时满足初始条件 $x(0) = \phi(f, x(0), 0)$ 。

重新审视梯度下降 在本节开头，我们从梯度下降的离散参数更新出发，引出了连续时间下的梯度流方程。在介绍了常微分方程及其数值求解方法后，现在可以更准确地理解二者之间的关系。

考虑梯度流

$$\frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t)). \quad (4.21)$$

它是一般常微分方程

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$$

在状态变量取为模型参数 $\mathbf{x}(t) = \boldsymbol{\theta}(t)$ ，动力学函数取为负梯度

$$f(\boldsymbol{\theta}(t), t) = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t))$$

时的一个特殊情形。表 4.2 总结了梯度流、欧拉法与梯度下降之间的对应关系。

表 4.2: 梯度流、显式欧拉法与梯度下降之间的对应关系

基本概念	一般 ODE 与欧拉法	梯度流与梯度下降
系统状态	$\mathbf{x}(t)$ 或 $\mathbf{x}_k$	模型参数 $\boldsymbol{\theta}(t)$ 或 $\boldsymbol{\theta}_k$
动力学函数	$f(\mathbf{x}(t), t)$	$-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t))$
连续动力学	$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$	$\frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t))$
离散步长	时间步长 Δt	学习率 η
显式欧拉更新	$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t f(\mathbf{x}_k, t_k)$	$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k)$

具体而言，对梯度流方程采用步长为 $\Delta t = \eta$ 的显式欧拉法，可得

$$\begin{aligned} \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + \eta [-\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k)] \\ &= \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k), \end{aligned} \quad (4.22)$$

这正是梯度下降的更新公式。因此，从连续动力学的角度看，梯度流描述参数沿负梯度方向连续运动的理想化轨迹，而梯度下降则通过有限步长对这条连续轨迹进行离散近似；其中，学习率既控制每次参数更新的幅度，也对应数值求解中的时间步长。

将梯度下降与常微分方程联系起来后，我们便可以借助动力系统和数值分析中的概念，进一步理解优化算法的训练行为。例如，梯度下降可以看作梯度流的显式欧拉离散，其中学习率对应数值求解的时间步长。当学习率过大时，离散更新可能无法准确跟随连续梯度流，从而出现损失振荡甚至训练发散。当损失函数在不同方向上的变化尺度相差很大时，这种现象还与常微分方程中的多时间尺度和刚性问题密切相关。

这一连续动力学视角也可以自然地扩展到动量方法。考虑带动量的梯度下降

$$\mathbf{v}_{k+1} = \mu \mathbf{v}_k - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k), \quad (4.23)$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{v}_{k+1}, \quad (4.24)$$

其中， $\mathbf{v}_k$ 表示由过去更新逐步累积形成的动量， $\mu \in [0, 1)$ 为动量系数。与普通梯度下降只根据当前位置的梯度更新不同，动量方法还保留了参数此前的运动趋势，因此可以减弱某些方向上的来回振荡，并在梯度方向较为一致时加快参数前进。

在适当的时间缩放下，上述离散更新可以对应如下二阶常微分方程：

$$\frac{d^2 \boldsymbol{\theta}(t)}{dt^2} + \gamma \frac{d\boldsymbol{\theta}(t)}{dt} = -\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}(t)), \quad (4.25)$$

其中，二阶导数描述参数运动的惯性，一阶导数项表示阻尼，损失函数的负梯度则提供驱动参数下降的作用力。因此，普通梯度下降可以理解为一阶梯度流，而带动量的优化方法则对应具有惯性和阻尼的二阶动力系统。这说明，常微分方程不仅提供了描述连续参数轨迹的工具，也为理解不同优化算法的结构与行为提供了统一视角。

由此，本节开头提出的离散与连续之间的双向联系在梯度下降中得到了一个具体体现：离散梯度下降可以通过连续化得到梯度流，而梯度流又可以通过显式欧拉法重新离散为梯度下降。后续章节将沿用这一基本思想，从连续动力学及其数值离散的角度理解神经网络中的表示演化。

课后练习

1. 说明离散状态更新、连续动力学和数值离散三者之间的关系。为什么从有限步长的梯度下降得到梯度流只能理解为理想化连续化？
2. 设 $L(\theta) = \frac{1}{2}a\theta^2$ ，其中 $a > 0$ 。写出对应的梯度流，并用步长 $\Delta t = \eta$ 作一次显式 Euler 离散。所得更新何时会因步长过大而不收敛到0？
3. 对初值问题 $dx/dt = -x$ ， $x(0) = 2$ ，分别写出解析解，并用步长 $\Delta t = 0.25$ 的显式 Euler 法计算 $x(0.25)$ 和 $x(0.5)$ 的近似值。
4. 比较显式 Euler 法、RK4 与自适应步长方法所利用的局部信息、计算代价

和误差控制方式。为什么高阶方法不能简单理解为“多堆几层”？

5. 若梯度下降使用变化的学习率 $\eta_k > 0$ ，请设计一组非均匀训练时刻 t_k ，使离散更新仍能写成平均变化率等于负梯度的形式。

4.2 表示空间中的连续动力学

在深度学习中，输入经过网络变换后形成的中间特征称为**表示**（representation），所有可能表示构成的空间称为**表示空间**（representation space）。与训练过程中不断更新的模型参数不同，隐藏表示在模型参数固定后，随着网络前向传播而逐层变化。因此，深层网络的计算过程可以看作输入表示在表示空间中的一系列状态变换。

在 Transformer 中，自注意力模块和前馈网络不断更新序列的隐藏表示。由于这些模块通常采用残差形式，其逐层更新与常微分方程数值求解中的离散状态推进具有相似结构。因此，可以将隐藏表示视为系统状态，将不同网络层视为表示演化过程中的离散计算步骤。这一视角不仅可以用于理解 Transformer 中表示的逐层变化，还可以借助高阶方法、隐式方法以及稳定性理论分析和设计不同的网络结构。

常微分方程还可以进一步直接用于构造神经网络模块。与 Transformer 子层直接输出更新后的隐藏表示不同，神经常微分方程使用神经网络 f_θ 描述隐藏表示在当前状态下的瞬时变化率：

$$\frac{d\mathbf{H}(t)}{dt} = f_\theta(\mathbf{H}(t), t). \quad (4.26)$$

其中，神经网络接收当前隐藏表示 $\mathbf{H}(t)$ 和时间 t ，并输出其变化率。给定输入表示后，常微分方程求解器在指定时间区间内反复计算这一变化率，并通过数值积分得到最终的输出表示。因此，神经网络负责定义表示的局部演化规律，而常微分方程求解器负责将这些局部变化累积为完整的表示变换。这种模型称为神经常微分方程（Neural Ordinary Differential Equation, Neural ODE）。

本节首先介绍 Transformer 的表示更新与常微分方程数值离散之间的联系，随后讨论常微分方程视角对网络结构设计、稳定性与刚性分析的启发，最后介绍神经常微分方程及其训练方法。

重点提示

新概念：将层索引视为离散时间、将隐藏表示视为状态后，残差更新可以解释为表示轨迹的一次数值推进。Neural ODE 则直接用神经网络参数化瞬时变化率，并由 ODE 求解器在连续深度区间上累积这些局部变化。

重难点解析：标准 Transformer 与显式 Euler 法具有“当前状态加局部增量”的结构对应，但其层间参数通常不共享，归一化、注意力和前馈子层也带来额外结构，因此不能据此断言它严格求解某个固定 ODE。只有明确连续模型、参数随

深度的变化方式、步长和离散格式后，数值分析中的阶数与稳定性结论才可直接使用。

4.2.1 Transformer 的离散动力学解释

前一节已经说明，数值方法通过有限次状态推进来近似连续轨迹。将研究对象换成 Transformer 的隐藏表示后，自注意力模块和前馈网络所产生的残差更新也可以写成“当前状态加局部增量”的形式。这种结构上的对应使得 Transformer 的逐层计算可以从离散动力系统的角度理解，但并不意味着注意力机制本身就是微分方程，也不意味着 Transformer 与某一种数值方法在所有意义下严格等价。下面将围绕状态、层索引和更新函数之间的对应关系，说明 ODE 数值离散如何为表示演化提供一种统一表述。

表示状态与残差更新。 设序列长度为 n ，隐藏维数为 d ，第 l 个残差子层输入的隐藏表示记为 $\mathbf{H}_l \in \mathbb{R}^{n \times d}$ 。矩阵的每一行对应一个词元在当前深度处的上下文表示，而 $\boldsymbol{\theta}_l$ 表示该子层的模型参数。对于预归一化 Transformer，一个自注意力子层或前馈网络子层都可以统一写成

$$\mathbf{H}_{l+1} = \mathbf{H}_l + \alpha_l F_l(\text{LN}(\mathbf{H}_l); \boldsymbol{\theta}_l), \quad (4.27)$$

其中 F_l 表示子层变换，LN 表示层归一化， $\alpha_l > 0$ 是残差分支的尺度，通常取 $\alpha_l = 1$ 。式 (4.27) 中需要区分两类量： $\boldsymbol{\theta}_l$ 属于参数空间，在训练过程中由优化算法更新； $\mathbf{H}_l$ 属于表示空间，在给定参数后随一次前向传播逐层变化。这里研究的动力学是后一种变化，而不是参数优化轨迹。

为了建立连续描述，取一组深度网格

$$t_0 < t_1 < \cdots < t_L, \quad h_l = t_{l+1} - t_l,$$

并考虑表示空间上的非自治动力系统

$$\frac{d\mathbf{H}(t)}{dt} = f(\mathbf{H}(t), t; \boldsymbol{\theta}(t)). \quad (4.28)$$

对式 (4.28) 在 $[t_l, t_{l+1}]$ 上作一次显式 Euler 离散化，可得

$$\mathbf{H}(t_{l+1}) \approx \mathbf{H}(t_l) + h_l f(\mathbf{H}(t_l), t_l; \boldsymbol{\theta}(t_l)). \quad (4.29)$$

若令 $\mathbf{H}_l \leftrightarrow \mathbf{H}(t_l)$ 、 $\alpha_l \leftrightarrow h_l$ ，并把 $F_l \circ \text{LN}$ 看作局部向量场的一次离散实现，则式 (4.27) 与式 (4.29) 具有相同的“当前状态加步长乘局部变化率”结构。各个量的对应关系如下表所示：

Transformer 中的量	连续动力系统量的量	作用
子层索引 l	网格时刻 t_l	标记推进位置
隐藏表示 $\mathbf{H}_l$	状态 $\mathbf{H}(t_l)$	描述当前表示
子层变换 F_l	向量场 f	给出局部更新方向
残差尺度 α_l	步长 h_l	控制单步更新幅度
参数 $\boldsymbol{\theta}_l$	$\boldsymbol{\theta}(t_l)$	参数化局部动力学

这一对应首先是一种**结构解释**。标准 Transformer 的 F_l 可以随层改变，层归一化、注意力掩码和其他离散操作也未必来自某个预先给定的光滑向量场；同时，单位残差尺度并不一定是逼近连续轨迹所需的“小步长”。因此，不能由式 (4.29) 推出 Transformer 严格等于某个 ODE 的 Euler 求解结果。更准确地说，ODE 视角为不同层的状态推进提供了共同的数学语言，并允许用截断误差、稳定域和步长等概念考察这种推进。

一个完整 Transformer 层的分裂形式。若以 $\mathcal{A}_l$ 和 $\mathcal{B}_l$ 分别表示吸收了层归一化的自注意力变换和前馈网络变换，则完整一层可写为

$$\begin{aligned}\widetilde{\mathbf{H}}_l &= \mathbf{H}_l + h_l \mathcal{A}_l(\mathbf{H}_l; \boldsymbol{\theta}_l^{\text{att}}), \\ \mathbf{H}_{l+1} &= \widetilde{\mathbf{H}}_l + h_l \mathcal{B}_l(\widetilde{\mathbf{H}}_l; \boldsymbol{\theta}_l^{\text{ffn}}).\end{aligned}\quad (4.30)$$

自注意力在词元维度聚合上下文信息，前馈网络则对各位置的表示施加非线性变换；二者直接产生隐藏表示的增量。若连续动力学被形式化为

$$\frac{d\mathbf{H}}{dt} = \mathcal{A}(\mathbf{H}, t) + \mathcal{B}(\mathbf{H}, t),$$

那么式 (4.30) 可理解为先推进 $\mathcal{A}$ 所描述的动力学，再从中间状态推进 $\mathcal{B}$ 所描述的动力学。这种顺序执行类似于一阶算子分裂，也说明两个子层的排列次序会影响最终表示。图 4.5 概括了 Transformer 子层更新与 ODE 数值推进之间的结构对应。

由此，Transformer 的一次前向传播可以被看作表示状态沿网络深度经历的一串离散推进。这个解释的价值不在于把已有网络重新命名为 ODE，而在于为下一步提出更具体的问题：若一种残差结构类似于一阶显式推进，那么改变积分格式，是否会产生具有不同精度、计算量和稳定性质的网络模块？

4.2.2 数值积分视角下的网络结构设计

离散动力学的解释不仅用于重述已有结构，还提供了比较和设计网络模块的数值分析语言。标准残差更新主要利用当前表示给出一次局部推进，而高阶方法会在一个更新区间内综合多个局部变化率或中间表示；隐式方法则让待求的下一状态参与更新方程，因而需要预测-校正、定点迭代或其他方程求解过程。本小节将由这一差异出发，讨论数值积分格式如何启发多阶段、历史复用和隐式求解式的网络结构，而不是

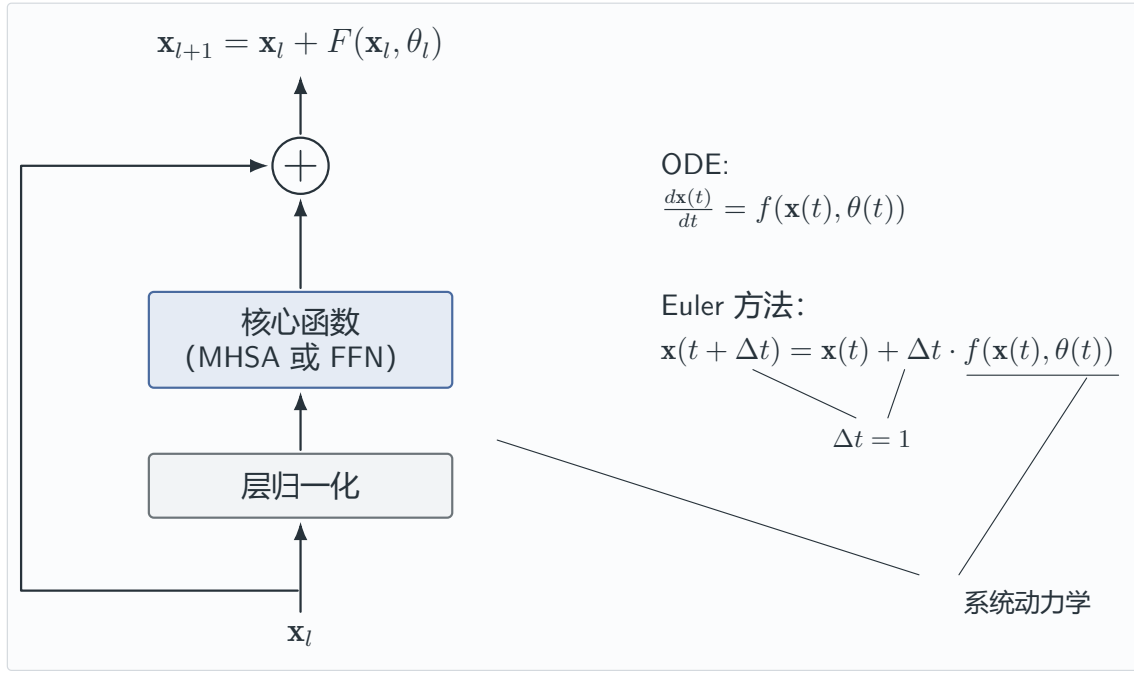


图 4.5: Transformer 的离散表示更新与连续动力系统状态推进之间的结构对应。

简单地把“高阶”理解为增加网络层数。

算子分裂与子层排列。把注意力和前馈网络看作两个同时作用于表示的向量场，

$$\frac{d\mathbf{H}}{dt} = \mathcal{A}(\mathbf{H}) + \mathcal{B}(\mathbf{H}),$$

则标准的串行子层可对应于 Lie-Trotter 型分裂：在一个步长内先近似求解 $\mathcal{A}$ ，再近似求解 $\mathcal{B}$ 。这种分裂通常只有一阶精度，而且不具备交换对称性；交换两个子层的顺序一般会改变输出。若希望获得对称的二阶组合，可采用“半步前馈-整步注意力-半步前馈”的 Strang 型结构。若 $\Phi_h^{\mathcal{A}}$ 与 $\Phi_h^{\mathcal{B}}$ 分别表示两个子动力学推进时间 h 的流映射，则

$$\mathbf{H}_{l+1} = \Phi_{h_l/2}^{\mathcal{B}} \circ \Phi_{h_l}^{\mathcal{A}} \circ \Phi_{h_l/2}^{\mathcal{B}}(\mathbf{H}_l). \quad (4.31)$$

在经典数值分析中，式 (4.31) 要具有二阶精度，需要满足一定条件。首先，前后两个半步对应的必须是同一个子动力学算子，并且每个子流映射都需要被足够准确地求解。对于显式依赖时间的非自治系统，还需要对时间变量采用与分裂格式一致的处理方式。如果仅使用普通残差子层去近似每个流映射 Φ ，或者在两次前馈变换中采用完全独立的参数，那么得到的网络模块虽然仍然具有类似“B-A-B”的三明治结构，但并不能自动继承 Strang 分裂方法的二阶误差性质。换言之，数值积分格式提供的是一种网络设计思路，而具体模型是否保留相应的理论性质，还取决于算子的参数共享方式、子模块求解精度以及实际实现方式。

多阶段高阶更新。数值方法的“阶”描述误差随步长缩小时的收敛速度。对足够光滑的问题和稳定的单步格式，局部截断误差为 $O(h_l^{p+1})$ 时，在固定积分区间上的全局误差通常为 $O(h_l^p)$ ，相应方法称为 p 阶方法；多步格式还需要相应的零稳定性条件。

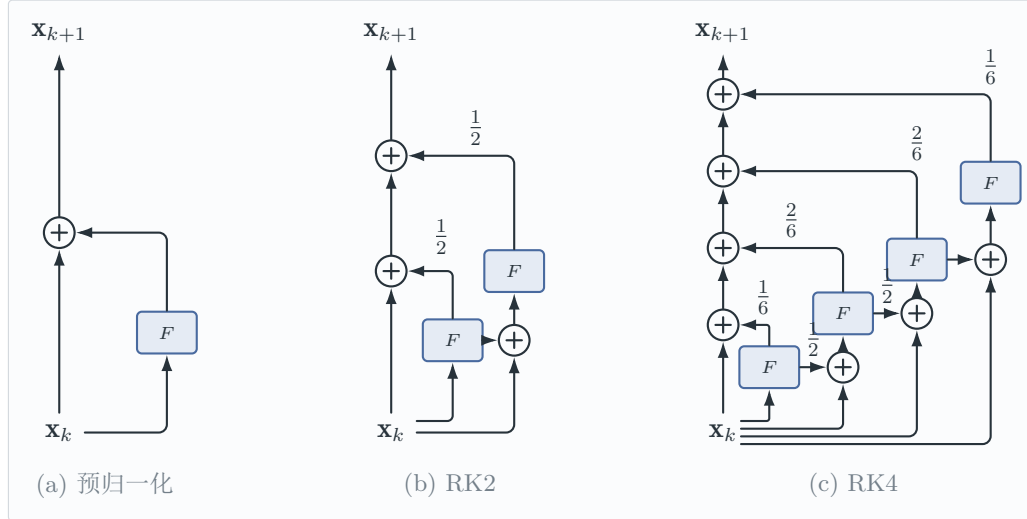


图 4.6: 基于数值 ODE 求解器启发的 Transformer 子层架构对比。(a) 基线预归一化结构对应一阶显式 Euler 型更新；(b) 与 (c) 分别展示受 RK2 和 RK4 启发的多阶段更新。中间状态与局部变化率在同一宏观更新区间内被组合。

这里的 p 并不表示网络堆叠了 p 倍的外部层数。高阶单步方法通常在同一个更新区间内评估多个中间斜率。例如，可用二阶 Heun 格式构造两阶段模块：

$$\begin{aligned}
 \mathbf{K}_1 &= F(\mathbf{H}_l, t_l; \boldsymbol{\theta}_l), \\
 \mathbf{K}_2 &= F(\mathbf{H}_l + h_l \mathbf{K}_1, t_l + h_l; \boldsymbol{\theta}_l), \\
 \mathbf{H}_{l+1} &= \mathbf{H}_l + \frac{h_l}{2} (\mathbf{K}_1 + \mathbf{K}_2).
 \end{aligned} \tag{4.32}$$

$\mathbf{K}_1$ 是起点处的变化率， $\mathbf{K}_2$ 在预测的终点处重新评估变化率，二者的平均值近似区间上的平均斜率。若两次评估共享 $\boldsymbol{\theta}_l$ ，模块的参数量不必随阶段数成比例增加，但函数评估次数和计算量仍会增加。更高阶 Runge-Kutta 格式遵循同一思想：在一个宏观更新内组合多个中间状态，而不是把“高阶”简单等同于“更深”。此外，阶段数与误差阶只是低阶格式中常见的对应，二者在一般情形下并不相同。

图 4.6 将基线预归一化子层与两阶段、四阶段更新并列展示。该对照强调，高阶格式所增加的是一个宏观更新内部的函数评估阶段，而不是简单增加外部网络深度。

历史复用与多步更新。高阶近似也可以利用已经计算过的历史变化率。例如，在等步长 h 下，二步 Adams-Bashforth 型更新为

$$\mathbf{H}_{l+1} = \mathbf{H}_l + h \left[\frac{3}{2} F_l(\mathbf{H}_l) - \frac{1}{2} F_{l-1}(\mathbf{H}_{l-1}) \right]. \tag{4.33}$$

它复用前一层已经得到的局部更新，可在较少新增函数评估的情况下利用轨迹历史；代价是需要保存历史表示或历史变化率，而且网络的首个更新必须由单步格式启动。式 (4.33) 具有经典二阶精度的前提是， F_{l-1} 与 F_l 是同一个足够光滑的非自治向量场在相邻网格时刻的函数值，并且网格等步长、启动值也达到所需精度。若各层参数彼

此独立，使 F_l 只是一组任意变化的层函数，该更新仍可作为历史复用结构，但不再自动具有 Adams–Bashforth 的二阶保证。从架构角度看，跨层聚合、稠密连接或显式缓存旧层更新都可与这种多步思想建立联系，但只有在系数和状态对应关系明确时，才能讨论其数值阶。

隐式更新与内部求解。显式格式由当前状态直接计算下一状态；隐式 Euler 格式则定义

$$\mathbf{H}_{l+1} = \mathbf{H}_l + h_l F(\mathbf{H}_{l+1}, t_{l+1}; \boldsymbol{\theta}_l). \quad (4.34)$$

未知的 $\mathbf{H}_{l+1}$ 同时出现在等式两侧，所以式 (4.34) 不是一次普通前向计算。一种基本实现是先用显式步给出预测值，再作定点迭代：

$$\begin{aligned} \mathbf{H}_{l+1}^{(0)} &= \mathbf{H}_l + h_l F(\mathbf{H}_l, t_l; \boldsymbol{\theta}_l), \\ \mathbf{H}_{l+1}^{(r+1)} &= \mathbf{H}_l + h_l F(\mathbf{H}_{l+1}^{(r)}, t_{l+1}; \boldsymbol{\theta}_l). \end{aligned} \quad (4.35)$$

当 F 对状态的 Lipschitz 常数为 L_F 且 $h_l L_F < 1$ 时，上述映射在相应区域内为压缩映射，定点迭代具有基本的收敛保证。实际网络也可采用有限次校正、拟牛顿法或学习到的求解器近似隐式解。因此，“隐式”指下一状态由隐式方程定义，而不是指隐藏层不可见；其主要代价是内部迭代和停止准则带来的额外计算。

格式	使用的信息	主要潜力	主要代价
显式单步	当前表示处的一次变化率	结构简单、可并入普通残差块	精度和稳定域受步长限制
多阶段高阶	同一区间内的多个中间变化率	提高局部近似阶，参数可在阶段间共享	增加函数评估次数
多步	当前及若干历史变化率	复用历史计算	需要启动方法和跨层缓存
隐式	包含未知下一状态的变化率	往往具有更大的稳定域	需要求根或迭代校正

4.2.3 表示动力学的稳定性与刚性

尽管常微分方程为模型设计提供了理论解释，但在将相关概念应用于深度神经网络时，底层动力系统的稳定性仍然是一个挑战。从常微分方程的角度来看，Transformer 的稳定性与信息流的平滑性以及常微分方程的刚性密切相关。

将 Transformer 建模为常微分方程的一个问题是，学习到的动力学可能是刚性的。**刚性**指的是隐藏状态连续动力学以极不相同的速率演化的现象，例如，某些分量变化迅速，而其他分量变化缓慢。因此，刚性常微分方程是指标准显式数值方法严重不稳定的常微分方程，需要非常小的步长才能稳定地求解该方程。在神经网络的背景下，这意味着标准显式求解器（或固定深度的残差连接）可能无法有效地传播信号，因为

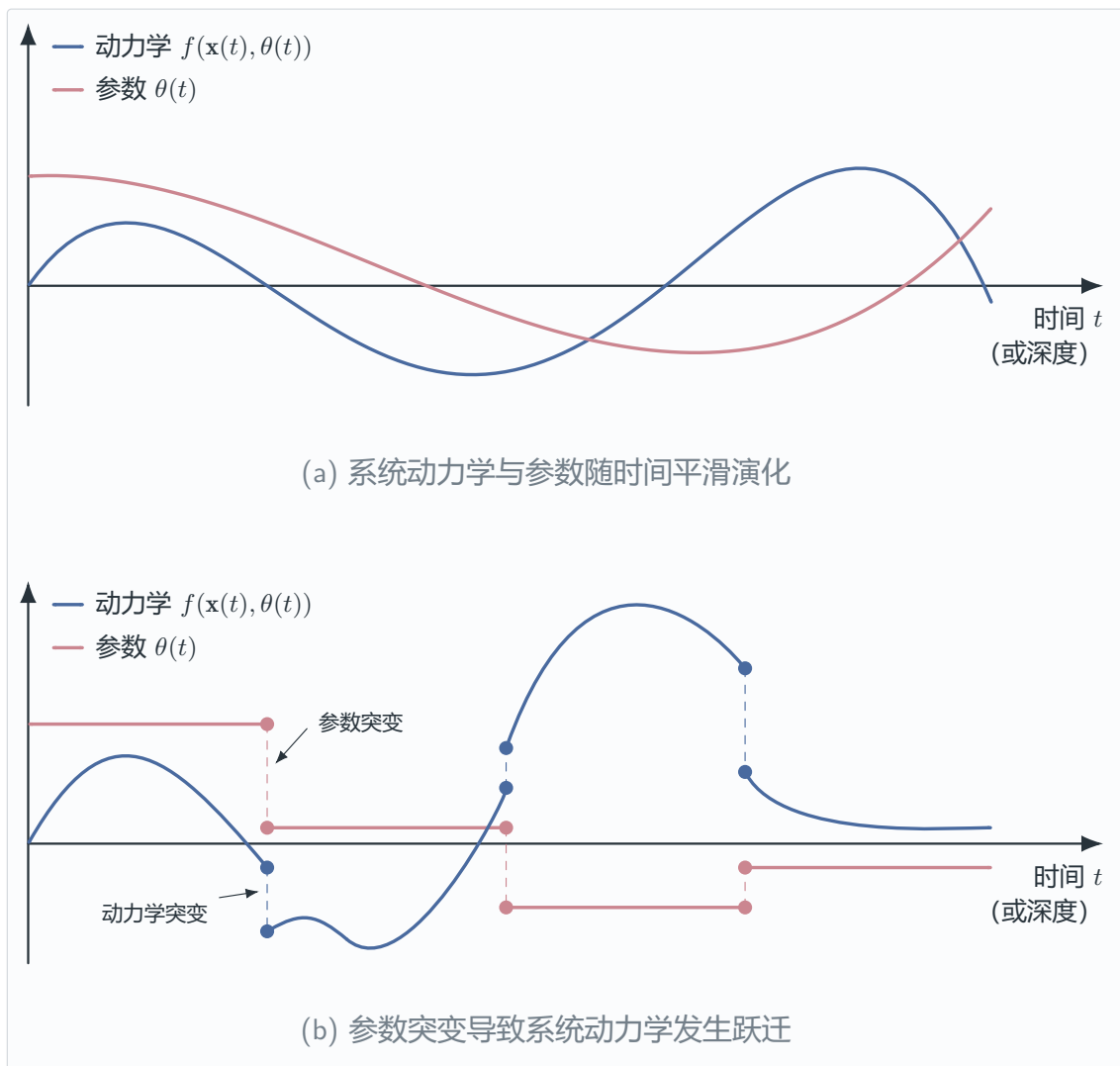


图 4.7: 系统动力学因参数突变而发生变化的示意图。蓝线表示系统动力学 $f(\mathbf{x}(t), \theta(t))$ 的演化, 红线表示参数 $\theta(t)$ 的演化。子图 (a) 展示参数和动力学连续平滑变化的理想情况; 子图 (b) 展示标准 Transformer 中离散层参数导致动力学发生阶跃变化的情况。

在尝试捕捉这些快速的瞬态变化时, 若没有足够精细的离散化, 它们容易产生数值不稳定性。经典数值分析文献指出, 标准欧拉法或龙格-库塔法等显式方法的稳定区域有限。当应用于刚性问题时, 它们通常需要很小的步长以避免数值发散。相比之下, 隐式方法通常更适用于刚性问题, 因为它们具有更大的稳定区域。

对于 Transformer 而言, 由于层的离散性, 这个问题更为严重。在这些模型中, 参数 θ 是独立的, 并且在不同层之间可能存在巨大差异。这与稳定求解常微分方程所需的平滑动力学假设相矛盾。回顾常微分方程公式 $\frac{d\mathbf{x}(t)}{dt} = F(\mathbf{x}(t), \theta(t))$, 其中 $\theta(t)$ 表示随时间 t 连续变化的参数。然而, 在标准 Transformer 中, 每层的参数是固定的, 从而将 $\theta(t)$ 离散化为一个阶梯函数 (即 $\theta_1, \theta_2, \dots$)。这些参数的突变导致系统动力学从一层到下一层发生急剧变化。例如, 在大语言模型中, 经常观察到前几层的参数值明显不同, 导致不同层之间的中间表示存在显著差异。图 4.7 展示了此问题的示意图。

为了缓解由刚性和参数突变引起的数值不稳定性，很自然地会约束函数 F 的变异性。实现此目标的一种常见方法是通过 Lipschitz 连续性条件。具体来说，为确保常微分方程良态且能稳定求解，必须存在一个常数 K 使得 $\|F(\mathbf{H}_1) - F(\mathbf{H}_2)\| \leq K\|\mathbf{H}_1 - \mathbf{H}_2\|$ 。在神经网络中，Lipschitz 常数与每层权重矩阵的谱范数（即矩阵的最大奇异值）密切相关。约束谱范数可以有效地限制 Lipschitz 常数。这种约束不仅平滑了动力学，还确保了信号通过网络深度的放大受到控制。这有助于防止数值发散。

更广泛地，我们可以使用显式正则化技术来实现更高的稳定性。可以考虑两种常见策略：

参数正则化。为了强制 Transformer 中动力学随深度的演化更平滑，可以在损失函数中引入一个惩罚项。以下方程展示了一个惩罚项的例子

$$\mathcal{L}_{\text{param}} = \lambda_{\text{param}} \sum_{k=1}^{L-1} \|\theta_{k+1} - \theta_k\|_2^2, \quad (4.36)$$

其中 λ_{param} 是惩罚项的权重， L 是 Transformer 层堆叠的深度¹。此项约束相邻层具有相似的参数。在极限情况下（例如，当 $\lambda_{\text{param}} \rightarrow \infty$ 时），此约束强制 $\theta_{k+1} \approx \theta_k$ ，这导致了参数共享架构。在此类模型中， θ 在各层间变为常数，非自治常微分方程转化为一个自治系统。这实际上将深度维度视为循环神经网络的时间步长。一般而言，自治系统更稳定且参数效率更高。

状态正则化。此方法直接控制层间输出的变化。例如，我们可以惩罚相邻层输出之间的较大差异，如下所示

$$\mathcal{L}_{\text{state}} = \lambda_{\text{state}} \sum_{k=1}^{L-1} \|\mathbf{H}_{k+1} - \mathbf{H}_k\|_2^2, \quad (4.37)$$

其中 λ_{state} 是该正则化项的权重。从这个意义上说，像层归一化这样的归一化方法可以被视为在 Transformer 中正则化隐藏状态的一种有效方式。另一种方法涉及将先前层的输出作为输入来产生组合输出。此类层组合技术已广泛用于训练深度神经网络。从数值分析的角度来看，这些架构可以解释为多步数值积分器（例如，线性多步方法），其中下一状态的近似依赖于先前状态的历史，而不仅仅是紧邻的前一个状态。这种序列依赖有效地平滑了隐藏状态 $\mathbf{H}(t)$ 的轨迹。通过聚合来自多个先前步骤的信息，即使离散变换变化迅速，信号的传播也能相对稳健。

4.2.4 从离散深度到神经常微分方程

常微分方程视角为理解和设计深度网络提供了一种连续动力学框架。在传统深度网络中，表示变换通常被描述为有限层网络的逐层计算过程：每一层根据当前状态产生新的表示，多个离散变换组合形成最终输出。从动力系统角度看，这一过程可以理

¹此处，一个 Transformer 层包含两个子层。 θ_k 表示整个层的所有参数集合。

解为表示状态在高维空间中的逐步演化，网络层对应离散时间步，而每层计算则决定状态在该步中的变化方向。

基于这一思想，ODE 不仅可以作为一种工具用于解释已有网络结构，还可以进一步用于重新设计网络的计算范式。前者关注的是：如何将离散层网络理解为连续动力系统的近似；后者则关注：如何直接利用连续动力系统定义网络的表示演化过程。

神经常微分方程（Neural Ordinary Differential Equation, Neural ODE）正是在这一背景下提出的连续深度模型。与固定层数的神经网络不同，Neural ODE 不再显式规定一系列离散状态更新，而是通过学习表示空间中的连续动力规律，由神经网络定义状态在任意时刻的瞬时变化率：

$$\frac{dH(t)}{dt} = f_{\theta}(H(t), t), \quad H(t_0) = H_0. \quad (4.38)$$

其中， $H(t)$ 表示连续深度变量下的隐藏状态， f_{θ} 表示由神经网络参数化的动力场（vector field）。需要强调的是， f_{θ} 并不是直接完成一次完整的表示变换，而是描述当前状态在演化过程中的瞬时变化方向，真正的状态变化轨迹由 ODE 求解器积分得到。

因此，Neural ODE 改变的是网络描述计算过程的方式：传统深度网络显式定义有限数量的状态转移，而 Neural ODE 学习的是连接这些状态的连续演化规律。换言之，网络不再直接回答“下一层表示是什么”，而是学习“当前表示应以何种速度和方向演化”。

表 4.3: 固定深度网络与 Neural ODE 模块的比较

比较维度	固定深度残差网络	Neural ODE 模块
状态更新	由预先排列的离散网络层直接完成	由连续向量场定义，并通过 ODE 求解器积分得到
基本对象	离散状态转移 $H_l \rightarrow H_{l+1}$	连续轨迹 $H(t)$ 的演化过程
参数形式	通常每一层具有独立参数 θ_l	通常共享动力学参数 θ ，并可显式依赖时间 t
计算步数	网络结构设计时固定	由求解器误差容差和轨迹复杂度动态决定
中间状态	对应固定层输出	对应 ODE 求解过程中的内部时间节点
误差控制	由网络结构和参数训练间接影响	可通过积分步长和误差容差显式控制

从数学上看，Neural ODE 的状态转移满足积分形式：

$$H(t_1) = H(t_0) + \int_{t_0}^{t_1} f_{\theta}(H(\tau), \tau) d\tau. \quad (4.39)$$

由于 f_{θ} 通常具有复杂的非线性结构，且积分路径 $H(\tau)$ 未知，上述积分一般无法通过解析方法求解，而是需要通过数值常微分方程求解器（如欧拉法、龙格-库塔法

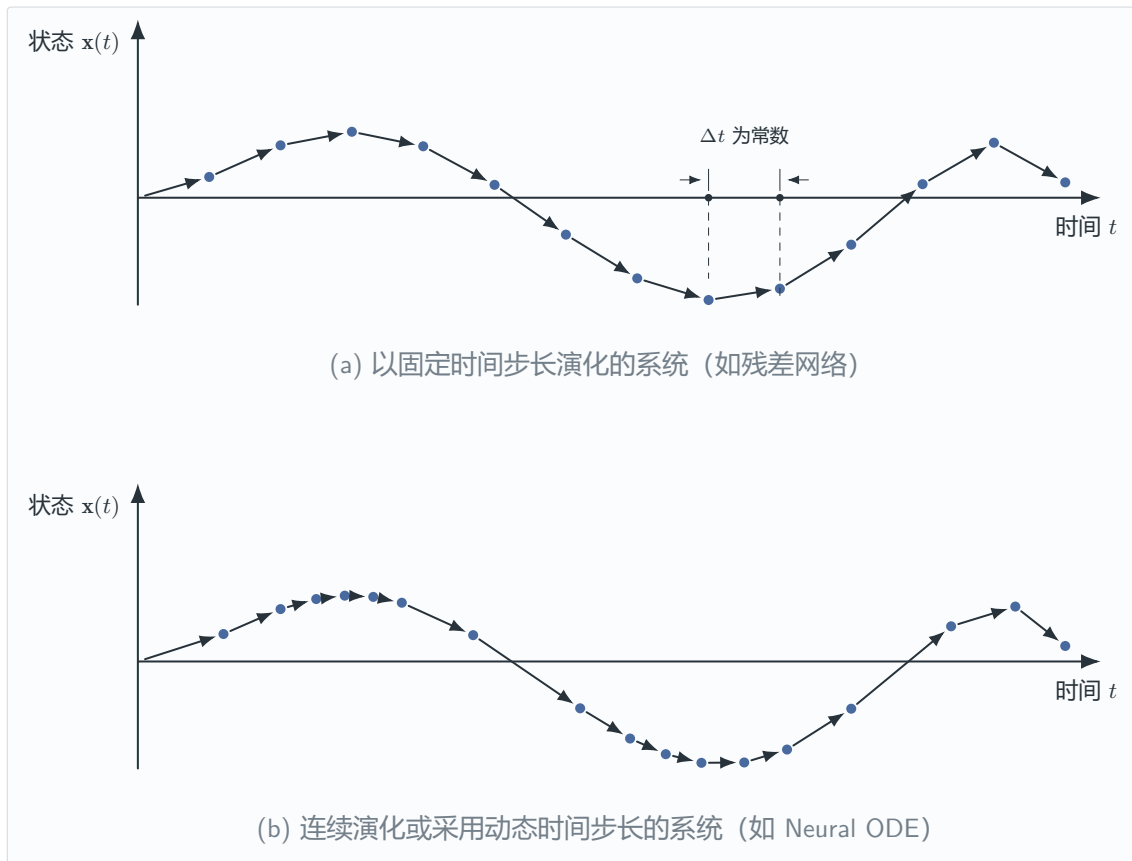


图 4.8: 固定步长与自适应步长求解常微分方程的示意图。(a) 固定步长方法在整个区间采用相同的时间步长；(b) 自适应步长方法根据局部轨迹复杂度与误差估计动态选择时间步长。

或更先进的方法) 计算得到:

$$H(t_1) = \text{ODESolve}(f_\theta, H(t_0), [t_0, t_1]). \quad (4.40)$$

其中, 神经网络负责计算当前状态下的变化率, 而 ODE 求解器负责不断调用该变化率, 并通过数值积分获得最终状态。

固定步长求解器预先规定各次状态推进的时间间隔, 而自适应求解器会依据局部误差估计动态调整步长。图 4.8 展示了两类求解策略在同一连续轨迹上的取点差异。

神经常微分方程的训练可视为标准深度神经网络训练任务, 通过计算损失函数对参数 θ 的梯度来迭代更新参数。直接方法是沿求解器运算过程执行标准反向传播, 但这往往因需要存储整个轨迹的中间状态以应用链式法则而产生高昂内存开销。内存瓶颈可能限制精细求解器或长时积分的使用。为此, 神经常微分方程通常采用**伴随灵敏度方法**进行训练, 该方法能以与积分步数无关的恒定内存成本计算梯度。我们将在第 4.2.5 节详细讨论训练细节。

简而言之, 神经常微分方程使用神经网络 $f(\cdot)$ 来近似系统动力学 $\frac{d\mathbf{H}(t)}{dt}$ 。在训练阶段, 我们优化该网络以建模这些动力学行为; 在测试阶段, 则可通过求解神经常微

分方程直接恢复系统在任意时刻的状态。

4.2.5 神经常微分方程的训练与伴随灵敏度法

伴随灵敏度方法（或称伴随方法）是源自最优控制领域的一种经典技术，用于计算由微分方程支配的系统的灵敏度。在深度学习中，它可以被解释为连续时间的反向传播。其核心思想是引入一组辅助变量，称为**伴随状态**，用于追踪损失的梯度如何随时间反向演化。它通过求解第二个微分方程来计算模型参数的梯度，而不是将链式法则显式地应用于前向传播的离散操作。

在将其与标准反向传播进行比较时，应强调这一区别。在一个朴素的实现（通常称为“先离散化再优化”）中，人们会将数值 ODE 求解器视为具有有限层数的计算图，并直接通过求解器的内部操作进行反向传播。然而，由于 ODE 求解器可能需要数千个小步骤才能达到高精度，标准反向传播需要存储每个步骤的中间激活值。这导致内存使用量随积分步数线性增长（ $O(T)$ ），达到难以承受的程度。换句话说，使用大量步长进行 ODE 求解的离散化使得训练内存密集，有时甚至不可行。

形式上，我们将伴随状态 $\mathbf{a}(t)$ 定义为损失函数 $\mathcal{L}$ 相对于任意给定时间 t 的隐藏状态 $\mathbf{x}(t)$ 的梯度

$$\mathbf{a}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}. \quad (4.41)$$

直观上， $\mathbf{a}(t)$ 表示最终损失对时间 t 时状态微小扰动的敏感性。由于损失是基于最终状态 $\mathbf{x}(t_1)$ 计算的，因此伴随状态的边界条件是已知的

$$\mathbf{a}(t_1) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}. \quad (4.42)$$

需要注意的是，虽然 $\mathbf{x}(t)$ 是从 t_0 到 t_1 正向求解的，但伴随状态 $\mathbf{a}(t)$ 必须使用 $\mathbf{a}(t_1)$ 作为反向积分的初始值，从 t_1 到 t_0 反向求解。图 4.9 展示了该方法的高层示意图。

伴随动力学

我们不通过求解器的步骤进行反向传播，而是通过求解一个新的常微分方程来计算 $\mathbf{a}(t)$ 。为了理解 $\mathbf{a}(t)$ 的动力学，考虑使用步长为 Δt 的简单欧拉方法对前向传播进行离散化。从 t 到 $t + \Delta t$ 的状态更新为

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \theta, t). \quad (4.43)$$

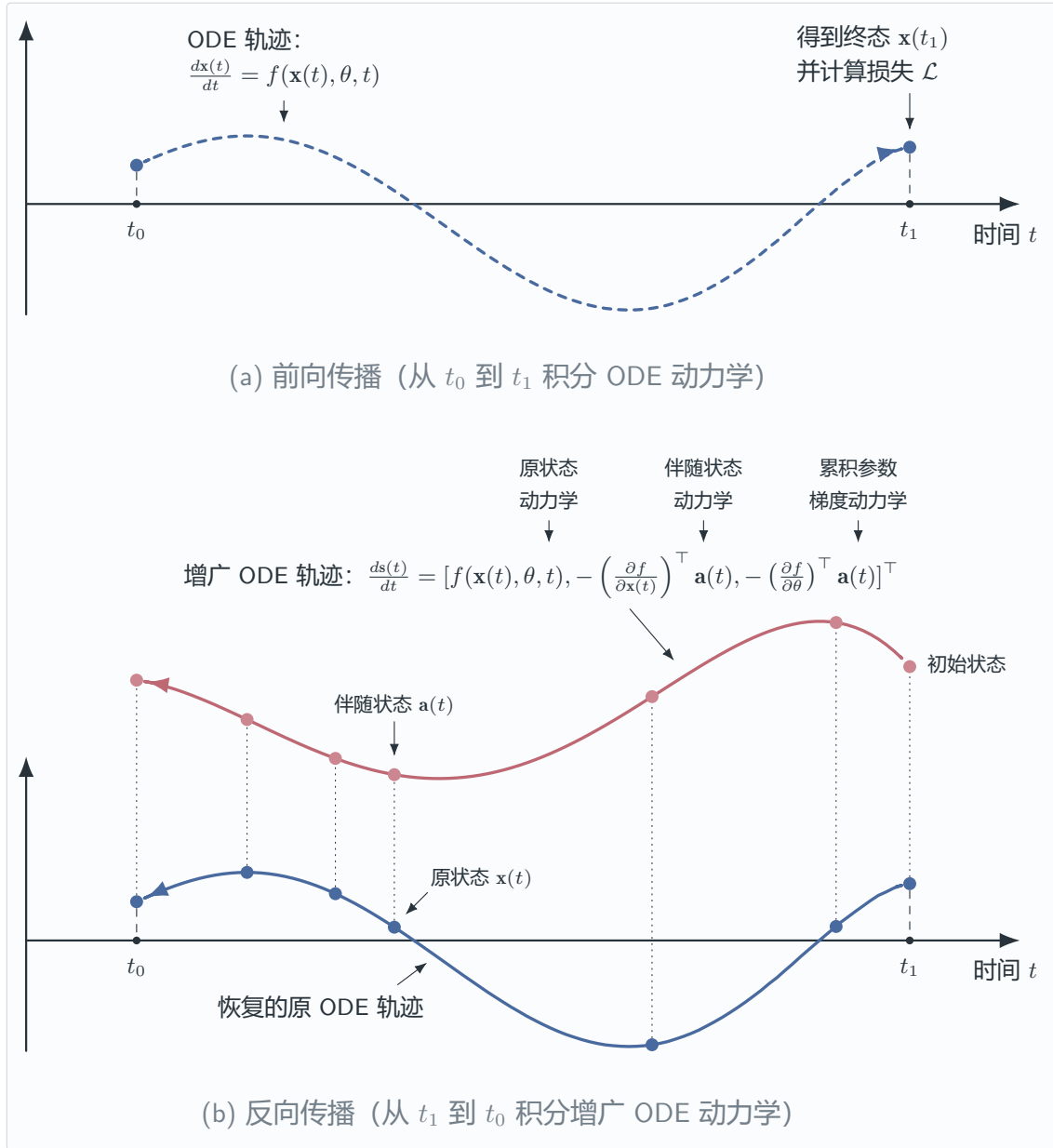


图 4.9: 神经常微分方程中伴随灵敏度方法的示意图

应用链式法则，损失关于隐藏状态 $\mathbf{x}(t)$ 的梯度可以表示为

$$\begin{aligned}
 \mathbf{a}(t) &= \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)} \\
 &= \left(\frac{\partial \mathbf{x}(t + \Delta t)}{\partial \mathbf{x}(t)} \right)^\top \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t + \Delta t)} \\
 &= \left(\frac{\partial \mathbf{x}(t + \Delta t)}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t + \Delta t),
 \end{aligned} \tag{4.44}$$

其中 $\frac{\partial \mathbf{x}(t + \Delta t)}{\partial \mathbf{x}(t)} \in \mathbb{R}^{d \times d}$ 是雅可比矩阵，而 $\mathbf{a}(t), \mathbf{a}(t + \Delta t) \in \mathbb{R}^d$ 是列向量。

将方程 (4.43) 代入方程 (4.44) 得到

$$\begin{aligned}
 \mathbf{a}(t) &= \left(\frac{\partial(\mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \theta, t))}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t + \Delta t) \\
 &= \left(\frac{\partial \mathbf{x}(t) + \partial \Delta t \cdot f(\mathbf{x}(t), \theta, t)}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t + \Delta t) \\
 &= \left(\mathbf{I} + \Delta t \cdot \frac{\partial f(\mathbf{x}(t), \theta, t)}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t + \Delta t).
 \end{aligned} \tag{4.45}$$

重新整理此方程，得到伴随状态的差商

$$\frac{\mathbf{a}(t + \Delta t) - \mathbf{a}(t)}{\Delta t} = - \left(\frac{\partial f(\mathbf{x}(t), \theta, t)}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t + \Delta t). \tag{4.46}$$

通过取极限 $\Delta t \rightarrow 0$ ，我们得到伴随状态的连续时间动力学。这导出了描述 $\mathbf{a}(t)$ 瞬时变化率的微分方程

$$\frac{d\mathbf{a}(t)}{dt} = - \left(\frac{\partial f(\mathbf{x}(t), \theta, t)}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t). \tag{4.47}$$

这里， $\frac{\partial f}{\partial \mathbf{x}}$ 是动力学函数的雅可比矩阵。这个方程告诉我们，伴随状态由一个线性常微分方程定义，该方程依赖于原始函数 $f(\cdot)$ 的雅可比矩阵。

类似地，我们可以推导出参数 θ 的梯度。在每个时间步 t ，参数 θ 对状态变化做出微小贡献，进而影响损失。时间 t 处的瞬时梯度贡献是损失对状态的敏感度 ($\mathbf{a}(t)$) 乘以状态变化对参数的敏感度 ($\frac{\partial f}{\partial \theta}$)。总梯度是这些瞬时贡献在整个轨迹上的累积（积分）

$$\frac{\partial \mathcal{L}}{\partial \theta} = - \int_{t_1}^{t_0} \left(\frac{\partial f(\mathbf{x}(t), \theta, t)}{\partial \theta} \right)^\top \mathbf{a}(t) dt. \tag{4.48}$$

注意，积分是从 t_1 到 t_0 反向进行的，这与反向传播的反向过程一致。

增广常微分方程与内存效率

式 (4.47) 和 (4.48) 给出了精确的梯度，但它们依赖于所有时刻 t 的状态轨迹 $\mathbf{x}(t)$ 。在标准的反向传播中，这需要存储前向传播中的所有中间状态 $\mathbf{x}(t)$ ，导致 $O(T)$ 的内存开销。

伴随灵敏度方法的关键优势在于通过反向时间重建状态轨迹来避免这种存储。由于描述状态演化的常微分方程是确定性的且通常是可逆的，我们可以通过反向运行动力学从最终状态 $\mathbf{x}(t_1)$ 恢复 $\mathbf{x}(t)$ ²。

²常微分方程在简单的函数意义上并非天生可逆，但这一概念适用于机器学习中的神经常微分方程和流（即随

为了高效地计算梯度，我们构建一个包含原始状态、伴随状态和累积参数梯度的增广状态

$$\mathbf{s}(t) = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{a}(t) \\ \frac{\partial \mathcal{L}}{\partial \theta}(t) \end{bmatrix}, \quad (4.49)$$

其中 $\frac{\partial \mathcal{L}}{\partial \theta}(t)$ 是从 t_1 到 t 的反向积分，即 $\frac{\partial \mathcal{L}}{\partial \theta}(t) = -\int_{t_1}^t \left(\frac{\partial f(\mathbf{x}(\tau), \theta, \tau)}{\partial \theta} \right)^\top \mathbf{a}(\tau) d\tau$ 。这个组合系统的动力学从 t_1 到 t_0 联合反向求解：

$$\frac{d\mathbf{s}(t)}{dt} = \begin{bmatrix} f(\mathbf{x}(t), \theta, t) \\ -\left(\frac{\partial f}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t) \\ -\left(\frac{\partial f}{\partial \theta} \right)^\top \mathbf{a}(t) \end{bmatrix}, \quad (4.50)$$

初始条件为

$$\mathbf{s}(t_1) = \begin{bmatrix} \mathbf{x}(t_1) \\ \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)} \\ \mathbf{0} \end{bmatrix}. \quad (4.51)$$

增广动力学的第一个分量就是原始的常微分方程。这意味着当求解器反向积分时，它会即时重建轨迹 $\mathbf{x}(t)$ 。在任何特定时间 t ，评估雅可比矩阵（在第二和第三分量中）所需的 $\mathbf{x}(t)$ 值在当前增广状态 $\mathbf{s}(t)$ 中即可获得。因此，我们不需要存储前向传播的中间状态，内存复杂度从 $O(T)$ 降低到 $O(1)$ 。本质上，伴随方法以重新求解状态轨迹的计算开销为代价，换取了内存的节省。需要注意的是，这个增广常微分方程可以使用现成的数值求解器来求解。人们也可以根据特定需求，通过简单地调整容差参数来平衡计算效率和数值精度。

总而言之，神经常微分方程的训练过程可以概括为以下两个步骤：

1. **前向传播**：从 t_0 到 t_1 求解常微分方程以得到 $\mathbf{x}(t_1)$ ，并计算损失 $\mathcal{L}$ 。
2. **反向传播**：在 t_1 处构造增广状态的初始值： $\mathbf{s}(t_1) = [\mathbf{x}(t_1), \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}, \mathbf{0}]^\top$ 。使用常微分方程求解器从 t_1 到 t_0 反向积分增广动力学方程 $\frac{d\mathbf{s}(t)}{dt} = [f(\mathbf{x}(t), \theta, t), -\left(\frac{\partial f}{\partial \mathbf{x}(t)} \right)^\top \mathbf{a}(t), -\left(\frac{\partial f}{\partial \theta} \right)^\top \mathbf{a}(t)]^\top$ 。

课后练习

1. 对残差更新 $\mathbf{H}_{l+1} = \mathbf{H}_l + \alpha_l F_l(\mathbf{H}_l)$ ，指出隐藏表示、层索引、残差尺度和子层变换分别对应显式 Euler 更新中的哪些量。

时间变化的解)。非正式地说，可逆性意味着逆转变换以找到先前的状态或在空间之间映射。

2. 设二阶 Heun 型模块满足 $K_1 = F(H_l)$ 、 $K_2 = F(H_l + hK_1)$ 、 $H_{l+1} = H_l + \frac{h}{2}(K_1 + K_2)$ 。当 $F(H) = H$ 、 $H_l = 1$ 、 $h = 0.2$ 时，计算一次更新结果。
3. 什么是表示动力学中的刚性？请从参数平滑、状态平滑或隐式更新中选择两种策略，说明它们如何改善稳定性。
4. 比较固定深度残差网络与 Neural ODE 在参数形式、计算步数、中间状态和误差控制方面的差异。若轨迹在某段时间内弯曲剧烈，自适应求解器应如何调整步长？
5. 写出伴随状态的终端条件和反向动力学，并说明伴随灵敏度法为何能够降低内存占用，以及它为此付出的计算或数值代价。

4.3 确定性概率动力学

上一节主要讨论了连续动力系统在表示学习中的作用，即如何描述单个隐藏状态随连续深度变量演化形成一条确定性的轨迹。在这一视角下，研究对象是给定初始状态后的表示变换过程，即关注“一个状态如何随动力系统运动”。

然而，生成模型关注的问题有所不同。生成模型的目标并非仅仅变换某一个确定的样本，而是学习数据背后的概率分布，并建立不同分布之间的转换关系。例如，在生成任务中，模型需要学习如何将简单的随机分布逐渐转化为复杂的数据分布。因此，连续动力系统的研究对象需要从单个状态进一步推广到由大量样本共同构成的概率分布。

当同一个确定性动力系统作用于不同初始样本时，每个样本都会沿自身轨迹演化，而所有样本轨迹的整体运动将诱导概率分布随时间发生连续变化。因此，连续生成模型需要从“一个状态如何运动”进一步转向“概率质量如何随动力系统重新分布”。

这一转变涉及两个相互关联的问题：一方面，需要建立样本轨迹与概率分布演化之间的数学联系，即利用流映射和连续性方程描述分布输运过程；另一方面，需要设计可学习的动力学机制，从数据中恢复这种分布变化规律。

本节主要介绍包括连续归一化流（Continuous Normalizing Flow, CNF）和流匹配（Flow Matching, FM）等基于确定性连续动力系统的生成方法。前者通过同时追踪样本轨迹与概率密度变化，构建可逆的连续生成模型；后者则通过预先构造概率路径，直接学习产生该路径的连续动力学。二者均采用确定性的样本输运过程，但在路径构造方式和训练目标上存在差异。

本节将首先从单个样本轨迹推广到概率分布的连续输运，建立流映射与质量守恒之间的关系；随后介绍连续归一化流与 Flow Matching 的基本思想，并讨论概率路径选择对轨迹几何性质和数值求解难度的影响。

重点提示

新概念：确定性 ODE 不仅把单个初始样本映射为一条轨迹，还会把整族初始样本共同推向新的分布。流映射描述样本位置如何变化，推前分布描述概率质量被搬运到哪里，连续性方程则以局部守恒形式刻画密度随时间的演化。

重难点解析：CNF 通过散度积分追踪沿轨迹的对数密度变化，适合显式似然建模；Flow Matching 预先构造概率路径，并以向量场回归学习产生该路径的速度。二者都在生成时积分 ODE，但监督信号和训练时是否需要求解完整轨迹并不相同；还应始终核对数据到先验与先验到数据的时间方向。

4.3.1 从样本轨迹到概率输运

常微分方程从一个初始状态出发可以确定一条轨迹，但仅研究单条轨迹无法说明一组随机样本的统计结构如何变化。为此，需要同时考虑来自初始分布的所有样本，并考察它们在同一动力学作用下被推向哪些位置。由样本级状态演化诱导出的分布级变化就是概率输运，它把前文的状态动力学扩展为生成建模所需的分布动力学。

给定状态空间中的一个初始样本 $x(0) \in \mathbb{R}^d$ ，以及一个随时间变化的向量场 $f : \mathbb{R}^d \times [0, T] \rightarrow \mathbb{R}^d$ ，该样本按照如下常微分方程演化：

$$\frac{dx(t)}{dt} = f(x(t), t), \quad x(0) = x_0. \quad (4.52)$$

这里的“确定性”是指：当初始状态 x_0 和向量场 f 给定后，轨迹不再包含额外的随机扰动，而是由动力系统唯一决定。不同的初始状态会产生不同的演化轨迹；如果考虑大量初始样本，这些轨迹的整体运动将进一步诱导样本分布随时间发生变化。

将初始状态推进到时刻 t 的映射记为流映射 $\Phi_{0 \rightarrow t}$ ，即：

$$x(t) = \Phi_{0 \rightarrow t}(x(0)) = x(0) + \int_0^t f(x(s), s) ds. \quad (4.53)$$

于是，分布 p_t 不是独立指定的，而是由初始分布 p_0 和流映射共同诱导。这一关系称为**推前 (pushforward)**：

$$p_t = (\Phi_{0 \rightarrow t})_{\#} p_0. \quad (4.54)$$

严格地说，对任意可测集合 $A \subseteq \mathbb{R}^d$ ，有：

$$\mathbb{P}(x(t) \in A) = \mathbb{P}(x(0) \in \Phi_{0 \rightarrow t}^{-1}(A)). \quad (4.55)$$

等价地，对任意适当的测试函数 φ ，

$$\mathbb{E}_{x(t) \sim p_t} [\varphi(x(t))] = \mathbb{E}_{x(0) \sim p_0} [\varphi(\Phi_{0 \rightarrow t}(x(0)))] . \quad (4.56)$$

式 (4.56) 揭示了样本视角与分布视角之间的对应关系：左侧关心时刻 t 下分布 p_t 的统计量，右侧则通过将初始样本 $x(0)$ 经过流映射推进后计算相同的统计量。因此，在有限训练样本条件下，也可以将经验分布中的每个样本沿上述 ODE 推进，并利用推进后的样本集合近似演化后的分布 p_t 。

本节统一采用：

$$p_0 = p_{\text{data}}, \quad p_T = p_{\text{prior}}$$

的时间方向：前向输运过程将数据分布逐渐转化为易于处理的先验分布，而生成过程则从 p_T 中采样，并沿同一流映射的逆方向回到 $t = 0$ 。如果流映射可逆，则有：

$$p_0 = (\Phi_{T \rightarrow 0})_{\#} p_T, \quad \Phi_{T \rightarrow 0} = \Phi_{0 \rightarrow T}^{-1}. \quad (4.57)$$

这一约定同时区分了两个层面：式中的 ODE 描述单个样本的位置变化，而推前关系 (4.54) 描述所有样本轨迹共同诱导的概率质量变化，如图 4.10 所示。

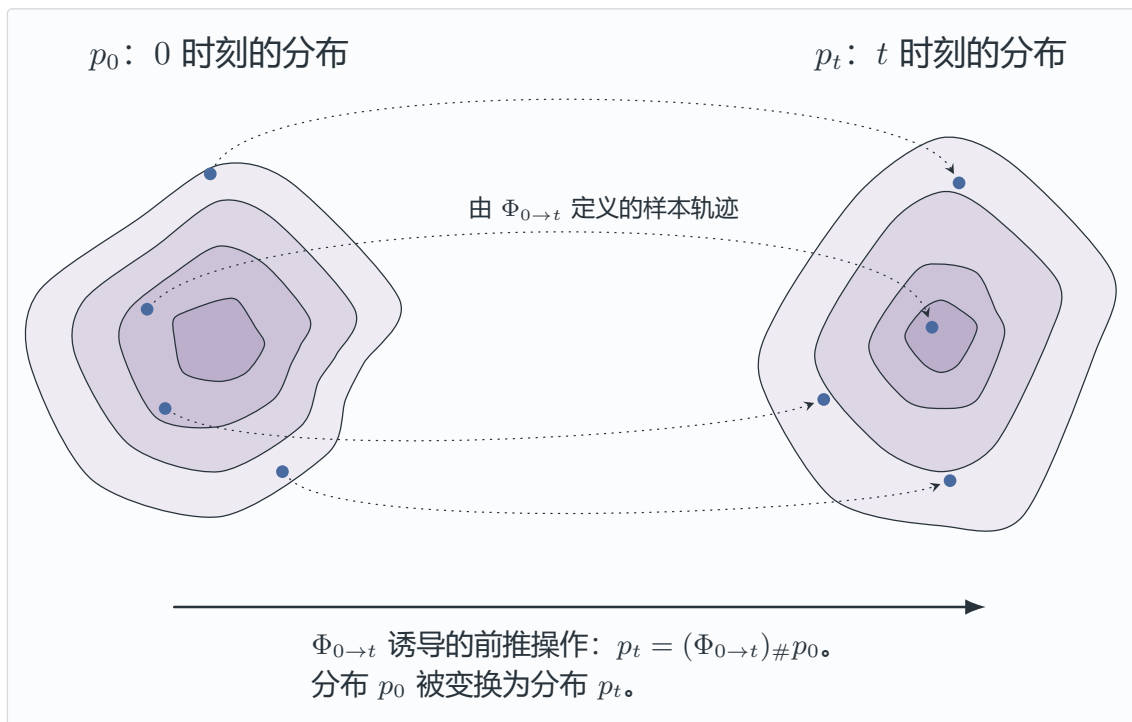


图 4.10: 同一向量场作用于所有初始样本时，样本轨迹的集合诱导出概率分布的推前。

推前关系给出了概率输运的整体定义，但尚未说明 $\Phi_{0 \rightarrow t}$ 在什么条件下可逆，也没有给出密度 $p_t(\mathbf{x})$ 在局部如何变化。下一小节将分别通过流映射的性质和连续性方程回答这两个问题。

4.3.2 流映射与连续性方程

描述概率输运既要知道每个初始点在给定时刻到达何处，也要保证运动过程中概率质量既不凭空产生也不消失。流映射汇总了向量场在一段时间内对所有初始状态的累积作用，连续性方程则从局部角度刻画密度变化与概率通量之间的平衡。二者分别连接样本轨迹和分布守恒，为后续计算密度随流的变化奠定基础。

为了同时描述任意起止时刻，引入两参数流映射

$$\Phi_{s \rightarrow t}(\mathbf{x}_s) = \mathbf{x}_s + \int_s^t f(\mathbf{x}(\tau), \tau) d\tau, \quad \mathbf{x}(s) = \mathbf{x}_s. \quad (4.58)$$

在解唯一存在的时间区间内，它满足恒等关系和复合关系

$$\Phi_{s \rightarrow s} = \text{Id}, \quad \Phi_{r \rightarrow t} = \Phi_{s \rightarrow t} \circ \Phi_{r \rightarrow s}, \quad r \leq s \leq t. \quad (4.59)$$

式 (4.59) 表明，先从 r 推进到 s ，再从 s 推进到 t ，与直接从 r 推进到 t 得到同一状态。对于显式依赖时间的非自治向量场，应使用这种带起止时刻的记号；只有在动力学不显式依赖时间时，才可以进一步把它写成常见的单参数流形式。

若 $f(\mathbf{x}, t)$ 关于 $\mathbf{x}$ 局部 Lipschitz，并且解在所讨论的时间区间内存在，则初值问题的解具有唯一性。唯一性排除了两条不同轨迹在同一时刻相交后再分开的情形，并给出

$$\Phi_{s \rightarrow t}^{-1} = \Phi_{t \rightarrow s}. \quad (4.60)$$

若向量场对状态还具有足够的连续可微性，流映射及其逆映射也连续可微，此时 $\Phi_{s \rightarrow t}$ 是一个微分同胚。这一光滑可逆性使得变量替换、雅可比矩阵和密度演化公式都有明确含义。需要注意，理论上的可逆流不等同于数值计算中的逐位可逆：有限步长和舍入误差仍可能使正向求解后再反向求解无法完全恢复初值。

流映射描述“点到哪里”，连续性方程则描述“概率质量怎样通过某处”。设 S_s 是时刻 s 的一个区域，并令 $S_t = \Phi_{s \rightarrow t}(S_s)$ 随流一起运动。由于确定性输运只移动概率质量而不创造或消灭概率质量，

$$\int_{S_t} p_t(\mathbf{x}) d\mathbf{x} = \int_{S_s} p_s(\mathbf{x}) d\mathbf{x}. \quad (4.61)$$

对任意随流区域都要求式 (4.61) 成立，可得到密度的连续性方程

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla_{\mathbf{x}} \cdot (p_t(\mathbf{x}) f(\mathbf{x}, t)) = 0. \quad (4.62)$$

其中， $p_t(\mathbf{x}) f(\mathbf{x}, t)$ 称为概率通量。若某一小区域的净流出为正，该区域中的概率密度就会下降；若净流入为正，密度则会上升。将散度项展开，可写成

$$\frac{\partial p_t}{\partial t} + f^\top \nabla_{\mathbf{x}} p_t + p_t \nabla_{\mathbf{x}} \cdot f = 0. \quad (4.63)$$

前两项合在一起正是沿样本轨迹观察到的密度全导数。因此，在 $p_t(\mathbf{x}(t)) > 0$ 处，

$$\frac{d}{dt} \log p_t(\mathbf{x}(t)) = -\nabla_{\mathbf{x}} \cdot f(\mathbf{x}(t), t). \quad (4.64)$$

如果散度为正，邻近轨迹形成的局部体积膨胀，密度沿轨迹降低；如果散度为负，局部体积收缩，密度升高；散度为零则表示局部体积保持。

从有限时间的角度看，令

$$J_{s \rightarrow t}(\mathbf{x}_s) = \frac{\partial \Phi_{s \rightarrow t}(\mathbf{x}_s)}{\partial \mathbf{x}_s}$$

为流映射的雅可比矩阵。多元变量替换公式给出

$$p_t(\Phi_{s \rightarrow t}(\mathbf{x}_s)) = p_s(\mathbf{x}_s) |\det J_{s \rightarrow t}(\mathbf{x}_s)|^{-1}. \quad (4.65)$$

式 (4.65) 记录一段时间内的累积体积变化，而式 (4.64) 记录其瞬时变化率。二者是同一质量守恒原理的有限时间形式和微分形式。连续归一化流正是利用后一种形式，避免直接计算整个流映射的高维雅可比行列式。

4.3.3 连续归一化流

流映射描述了单个样本在向量场作用下的位置变化，而连续归一化流 (continuous normalizing flow, CNF) 进一步关注整个概率分布如何随流演化。其核心思想是：利用一个连续可逆动力系统，将复杂的数据分布连续变换为简单的先验分布，同时保持概率质量守恒。

直观地看，这类似于将一团染料释放到流动的水中。虽然水流会使染料云发生拉伸、压缩和变形，但染料总质量保持不变。CNF 正是利用这种连续变形机制，建立简单分布与复杂数据分布之间的可逆映射。

设状态空间中的样本满足神经常微分方程

$$\frac{d\mathbf{x}(t)}{dt} = f_{\theta}(\mathbf{x}(t), t), \quad (4.66)$$

其中 f_{θ} 是由神经网络参数化的向量场。与普通 Neural ODE 只关注状态轨迹不同，CNF 还需要追踪轨迹对应的概率密度变化。

考虑初始时刻 $t = 0$ 的任意区域 $\mathcal{S}_0$ 。在流映射 $\Phi_t(\cdot)$ 作用下，该区域演化为

$$\mathcal{S}_t = \Phi_t(\mathcal{S}_0). \quad (4.67)$$

由于概率质量既不会产生也不会消失，区域内概率保持不变：

$$\int_{\mathcal{S}_t} p_t(\mathbf{x}) d\mathbf{x} = \int_{\mathcal{S}_0} p_0(\mathbf{z}) d\mathbf{z}. \quad (4.68)$$

令

$$\mathbf{x} = \Phi_t(\mathbf{z}),$$

根据多元变量替换公式，有

$$\int_{\mathcal{S}_0} p_t(\Phi_t(\mathbf{z})) \left| \det \frac{\partial \Phi_t(\mathbf{z})}{\partial \mathbf{z}} \right| d\mathbf{z} = \int_{\mathcal{S}_0} p_0(\mathbf{z}) d\mathbf{z}. \quad (4.69)$$

由于该关系对于任意区域 $\mathcal{S}_0$ 成立，因此得到密度的逐点变换关系：

$$p_t(\mathbf{x}(t)) \left| \det \frac{\partial \mathbf{x}(t)}{\partial \mathbf{x}(0)} \right| = p_0(\mathbf{x}(0)). \quad (4.70)$$

进一步整理得到连续流中的变量替换公式：

$$p_t(\mathbf{x}(t)) = p_0(\mathbf{x}(0)) \left| \det \frac{\partial \mathbf{x}(t)}{\partial \mathbf{x}(0)} \right|^{-1}. \quad (4.71)$$

其中雅可比行列式表示流映射造成的局部体积变化。如果流使空间发生膨胀，则单位体积内包含的概率质量减少；如果流使空间压缩，则概率密度相应增加。图 4.11 直观展示了连续流映射下样本分布及局部密度的同步变化。

然而，直接计算式 (4.71) 中从 0 到 t 的完整雅可比行列式在高维空间中代价很高。CNF 的关键思想是考虑无穷小时间变化。

对数密度关于时间求导，可以得到瞬时变量替换公式：

$$\begin{aligned} \frac{d \log p_t(\mathbf{x}(t))}{dt} &= -\text{Tr} \left(\frac{\partial f_\theta(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} \right) \\ &= -\nabla_{\mathbf{x}} \cdot f_\theta(\mathbf{x}(t), t). \end{aligned} \quad (4.72)$$

其中

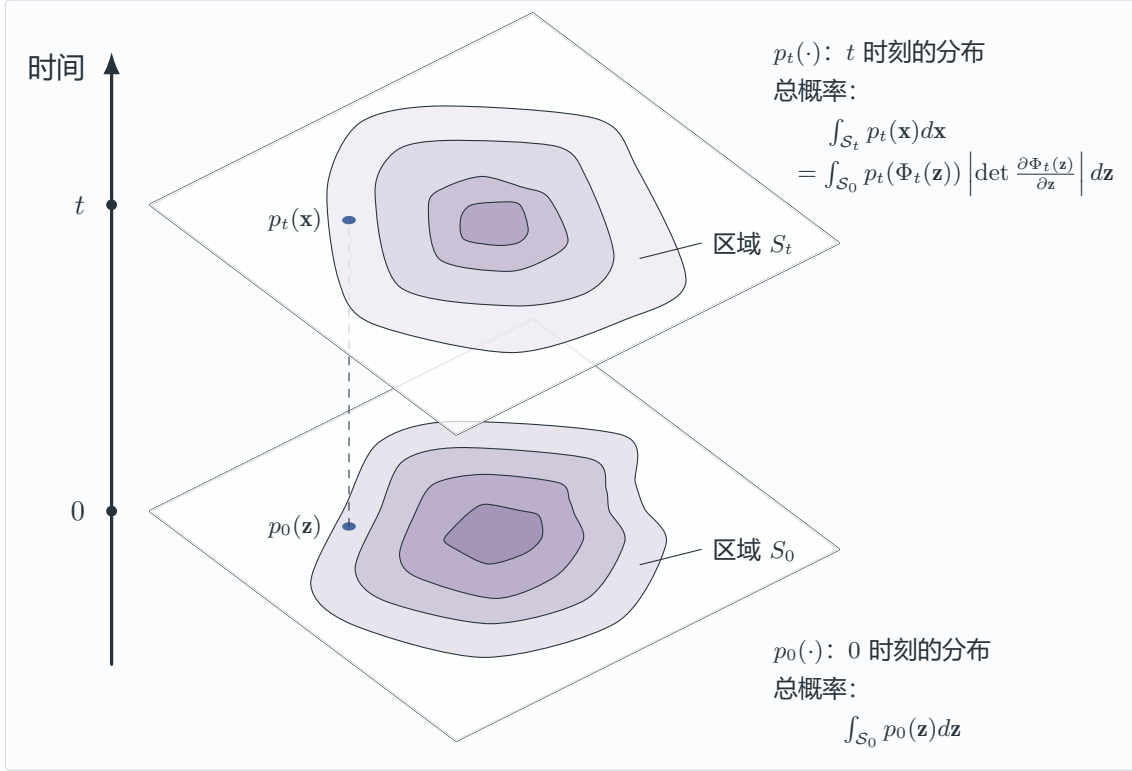


图 4.11: 连续归一化流中的概率输运过程。简单分布中的样本经过连续流映射后，形成复杂数据分布；流过程中的拉伸和压缩对应概率密度的变化。

$$\text{Tr} \left(\frac{\partial f_{\theta}}{\partial \mathbf{x}} \right)$$

是向量场雅可比矩阵的迹，也称为散度。它描述了局部体积变化速率，而无需显式计算完整雅可比行列式。

因此，状态变量和概率密度可以组成增广动力系统：

$$\frac{d}{dt} \begin{bmatrix} \mathbf{x}(t) \\ \log p_t(\mathbf{x}(t)) \end{bmatrix} = \begin{bmatrix} f_{\theta}(\mathbf{x}(t), t) \\ -\text{Tr}(\frac{\partial f_{\theta}}{\partial \mathbf{x}}) \end{bmatrix}. \quad (4.73)$$

在本章采用的数据到先验的时间方向约定下：

$$p_0 = p_{\text{data}}, \quad p_T = p_{\text{prior}},$$

沿时间方向积分式 (4.72)，得到

$$\log p_0(\mathbf{x}(0)) = \log p_T(\mathbf{x}(T)) + \int_0^T \text{Tr} \left(\frac{\partial f_{\theta}(\mathbf{x}(t), t)}{\partial \mathbf{x}(t)} \right) dt. \quad (4.74)$$

因此，CNF 使用同一个神经网络向量场同时完成三项任务：

- 将数据分布连续编码到简单先验分布；
- 从先验分布反向积分生成数据样本；
- 沿轨迹计算样本的概率密度。

相比离散归一化流需要设计特殊结构以保证雅可比行列式可计算，CNF 只需要计算雅可比矩阵的迹，因此可以使用任意神经网络参数化向量场。在高维情况下，迹还可以通过 Hutchinson 估计器等随机方法近似，从而降低计算成本。

模型训练完成后，生成样本无需计算密度项，只需从先验采样

$$\mathbf{x}(T) \sim p_T, \quad (4.75)$$

然后沿式 (4.66) 反向积分至 $t = 0$ ，即可得到生成样本。

不过，CNF 的似然训练仍然需要在优化过程中不断求解 ODE，并计算散度项。下一节介绍的 Flow Matching 则进一步改变训练方式：通过预先构造概率路径，直接监督路径上的局部速度，从而避免对未知流进行反复积分。

4.3.4 Flow Matching

连续归一化流可以通过似然学习向量场，但密度演化和散度计算会带来额外代价。如果能够预先构造连接先验分布与数据分布的概率路径，就可以直接监督路径上的局部速度，而不必先完整模拟一个未知流。Flow Matching 正是沿这一思路把动力学学习转化为向量场回归问题；本小节将介绍条件路径如何提供可计算的训练信号，以及局部条件速度如何汇集成整体的概率流。

首先固定端点的联合分布，

$$(\mathbf{X}(0), \mathbf{X}(T)) \sim q_{0,T}, \quad \mathbf{X}(0) \sim p_0, \quad \mathbf{X}(T) \sim p_T.$$

给定一对端点，通过可微插值映射 I_t 构造条件路径

$$\mathbf{X}(t) = I_t(\mathbf{X}(0), \mathbf{X}(T)), \quad I_0(\mathbf{x}_0, \mathbf{x}_T) = \mathbf{x}_0, \quad I_T(\mathbf{x}_0, \mathbf{x}_T) = \mathbf{x}_T. \quad (4.76)$$

当端点对随机变化时， $\mathbf{X}(t)$ 的边缘分布便形成一族 $\{p_t\}_{t \in [0, T]}$ 。与先运行一个待学习的 ODE 不同，式 (4.76) 允许直接在任意时刻构造训练状态。

沿一条给定条件路径的目标速度为

$$u_t(\mathbf{X}(t) | \mathbf{X}(0), \mathbf{X}(T)) = \frac{\partial}{\partial t} I_t(\mathbf{X}(0), \mathbf{X}(T)). \quad (4.77)$$

最常用的例子是端点之间的线性插值：

$$\mathbf{X}(t) = \left(1 - \frac{t}{T}\right) \mathbf{X}(0) + \frac{t}{T} \mathbf{X}(T), \quad (4.78)$$

相应的条件速度为

$$u_t = \frac{\mathbf{X}(T) - \mathbf{X}(0)}{T}. \quad (4.79)$$

对于固定端点对，式 (4.78) 是一条直线，式 (4.79) 在时间上保持常数。由于端点、时间、中间状态和目标速度都能直接采样或计算，它们可以作为监督数据训练神经向量场 $v_{\theta}(\mathbf{x}, t)$ 。

条件 Flow Matching 的平方损失为

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E} \left[\|v_{\theta}(\mathbf{X}(t), t) - u_t(\mathbf{X}(t) \mid \mathbf{X}(0), \mathbf{X}(T))\|_2^2 \right], \quad (4.80)$$

其中期望依次取自

$$t \sim \mathcal{U}[0, T], \quad (\mathbf{X}(0), \mathbf{X}(T)) \sim q_{0,T}, \quad \mathbf{X}(t) = I_t(\mathbf{X}(0), \mathbf{X}(T)).$$

一次训练迭代只需采样端点和时间，解析构造中间状态与条件速度，再进行普通的向量回归；无须先用当前网络从0数值积分到 t 。这里的“无模拟训练”仅指不需要模拟待学习的生成 ODE，并不意味着推断阶段不需要 ODE 求解器。

同一位置 $\mathbf{x}$ 和时刻 t 可能对应多组端点，因而也可能对应多个条件速度。平方损失的最优解不是记住某一条条件轨迹，而是这些速度在当前状态条件下的平均：

$$v^*(\mathbf{x}, t) = \mathbb{E} [u_t(\mathbf{X}(t) \mid \mathbf{X}(0), \mathbf{X}(T)) \mid \mathbf{X}(t) = \mathbf{x}]. \quad (4.81)$$

在适当的正则条件下，式 (4.81) 给出的边缘向量场满足

$$\frac{\partial p_t(\mathbf{x})}{\partial t} + \nabla_{\mathbf{x}} \cdot (p_t(\mathbf{x}) v^*(\mathbf{x}, t)) = 0, \quad (4.82)$$

因而产生的恰是条件路径混合后的边缘概率路径。这一步解释了为什么可以用容易计算的条件速度监督一个只接收 $(\mathbf{x}, t)$ 的全局向量场。图 4.12 展示了多条条件轨迹的局部速度如何通过条件期望汇集为边缘向量场。

推断时，从 $\mathbf{x}(T) \sim p_T$ 出发，将

$$\frac{d\mathbf{x}(t)}{dt} = v_{\theta}(\mathbf{x}(t), t) \quad (4.83)$$

从 T 反向求解到0。例如，反向欧拉更新为

$$\mathbf{x}(t - h) \approx \mathbf{x}(t) - h v_{\theta}(\mathbf{x}(t), t), \quad h > 0. \quad (4.84)$$

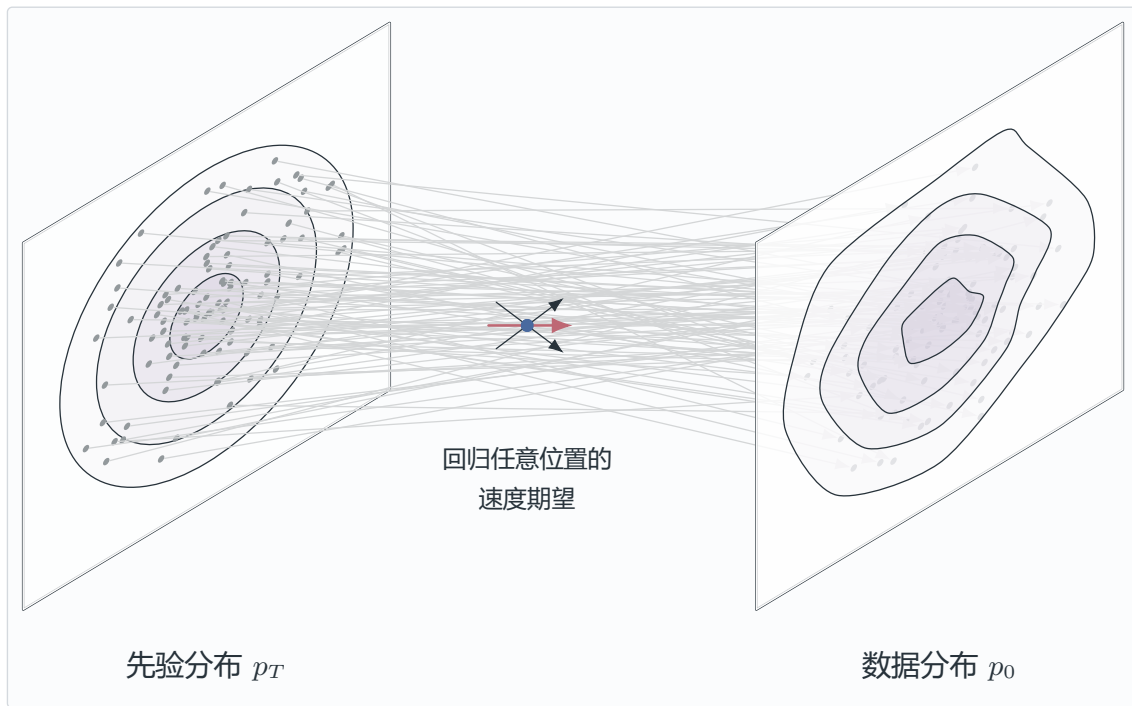


图 4.12: 流匹配中的边缘向量场学习示意图。灰色线条表示由不同端点对确定的条件轨迹，蓝点表示同一时刻附近的中间状态；红色箭头表示在给定当前状态后对条件速度取期望所得的边缘向量场。

这里没有另行把向量场定义为负号；负号来自时间步长由 t 走向 $t - h$ 。若 v_θ 准确逼近边缘向量场，反向流便将先验 p_T 输运回数据分布 p_0 。模型误差与数值积分误差则共同决定最终样本对目标分布的近似程度。

4.3.5 概率路径设计

如前所述，流匹配旨在学习一个向量场 $v_\theta(\mathbf{x}(t), t)$ ，该向量场能够诱导一个概率流，从而将简单的先验分布 p_T 转换为复杂的数据分布 p_0 。其核心目标是最小化学到的速度场与将样本沿选定路径传输的真实条件速度之间的差异（公式 4.80）。然而，问题源于数据分布 p_0 与噪声分布 p_T 之间的耦合。

流匹配模型的初始训练阶段，通常称为 1-修正流，数据点 $\mathbf{x}(0)$ 和噪声点 $\mathbf{x}(T)$ 从其各自的分布中随机配对。这种随机配对导致一种现象：高维潜在空间中的轨迹经常相交。当多条直线轨迹在时空中的同一点 $\mathbf{x}(t)$ 相交时，学到的边际向量场 $v_\theta(\mathbf{x}(t), t)$ （即这些不同相交速度的期望）会变得弯曲。因此，在积分常微分方程时实际遵循的路径不再是直的，这导致在数值求解器中采用大步长时会产生更多的离散化误差 [??]。在修正流框架中，可以通过减少条件路径之间的干扰来改善噪声分布与数据分布之间的耦合。首先，使用随机配对 $(\mathbf{x}(0), \mathbf{x}(T))$ 训练一个 1-修正流 v_θ^1 。然后，使用训练好的模型 v_θ^1 生成一个新的合成数据集。具体来说，对于一组噪声样本 $\{\mathbf{x}_i(T)\}$ ，求解常微分方程 $\frac{d\mathbf{x}(t)}{dt} = -v_\theta^1(\mathbf{x}, t)$ 以获得对应的生成数据样本 $\{\hat{\mathbf{x}}_i(0)\}$ 。这样就创建了一组确定性的、重新连接的对 $(\hat{\mathbf{x}}_i(0), \mathbf{x}_i(T))$ 。最后，使用这些对齐的对来训练一个新模

型 v_θ^2 (称为 2-修正流)。由于 $\hat{\mathbf{x}}_i(0)$ 是通过第一个模型的动力学从 $\mathbf{x}_i(T)$ 生成的, 这些潜在空间中的对是高度对齐的。因此, 得到的向量场明显更加线性。

对直线轨迹的探索根植于最优传输原理和 Wasserstein 距离的最小化。该领域的一个重要结果是, 对于从高斯分布到两个高斯混合分布的修正流, 只需几次修正就足以实现直线流。这一见解支撑了 InstaFlow 和 Stable Diffusion 等模型的经验成功, 这些模型利用修正流的直线化特性来减少函数评估次数, 同时保持样本质量 [?]。这些商业化模型的成功也表明, 采用修正流的扩散模型可以很好地扩展用于高质量图像生成。

轨迹的直线性还与学到的向量场的 Lipschitz 常数有关。如果 1-修正流是 L -Lipschitz 的, 则可以防止附近的噪声样本映射到任意远的数据点。然而, 在实践中, L 可能很大, 导致在生成的早期阶段传输路径出现显著弯曲。这种弯曲会降低一步欧拉采样的精度, 可能需要进一步的修正或更先进的采样调度。

轨迹修正的实际实现也涉及架构改进。研究人员已用复杂的多模态扩散 Transformer (MMDiT) 取代了简单的 U-Net 架构。例如, Stable Diffusion 3 引入了 MMDiT 架构 [?], 而 Flux.1 进一步将修正流 Transformer 架构扩展到 120 亿参数 [?]。这表明扩散模型可以扩展到非常大的规模, 同时保持高采样效率。

为了进一步提高效率和稳定性, 也可以将分治策略应用于修正流。例如, 分段修正流 (PeRFlow) 将生成轨迹划分为几个离散的时间窗口 [?]。它不是拉直全局流, 而是单独拉直每个区间内的片段, 这种局部化降低了模拟训练轨迹的计算成本, 并最小化了数值误差。此外, 研究人员通过先进的引导技术解决了条件生成的稳定性问题。例如, Rectified-CFG++ 采用了一种自适应预测器-校正器方案 [?]。每个生成步骤首先执行条件更新, 将样本锚定在学到的传输路径附近, 然后基于条件速度与无条件速度之间的差异应用流形感知校正。这确保了采样轨迹保持在数据流形的一个稳定管状邻域内。

课后练习

1. 一维样本满足 $dx/dt = 2$, 且 $x(0) \sim \mathcal{N}(0, 1)$ 。写出流映射 $\Phi_{0 \rightarrow t}$ 和时刻 t 的边缘分布 p_t 。
2. 对线性向量场 $f(\mathbf{x}, t) = a\mathbf{x}$, 求其散度以及沿轨迹的对数密度变化率。若 $a > 0$, 概率密度沿轨迹为何会下降?
3. 写出 CNF 同时追踪状态与对数密度的增广动力系统, 并说明它为什么只需要雅可比迹, 而不需要完整流映射的高维雅可比行列式。
4. 比较连续归一化流与 Flow Matching 的训练目标、监督信号和生成阶段。二者分别在哪些情形下更适合用于显式似然建模或直接向量场学习?
5. 随机端点配对即使采用直线条件路径, 学到的边缘轨迹仍可能弯曲。请解释原因, 并设计从 1-修正流得到 2-修正流的基本训练流程。

4.4 随机概率动力学

上一节讨论了如何利用确定性向量场推动样本运动，并由大量样本轨迹共同实现概率分布之间的连续输运。在这一框架下，一旦给定初始状态和向量场，样本的演化轨迹便由相应的常微分方程唯一确定。

然而，确定性输运并不是连续时间生成建模的唯一方式。另一类重要方法允许样本在演化过程中持续受到随机扰动，从而以随机动力学的方式连接数据分布与简单先验分布。与确定性流相比，此时即使从相同的初始状态出发，不同的随机扰动也可能产生不同的样本路径；但从整体上看，大量随机轨迹仍然刻画了一族随时间连续变化的概率分布。

扩散模型正是建立在这一思想之上的代表性方法。本节将主要从随机微分方程的视角理解扩散生成过程，将其视为概率分布在连续时间中的随机演化，而不以传统扩散模型的离散时间构造为叙述起点。诸如 DDPM 中具体的离散加噪过程、变分下界推导、网络参数化形式及工程实现细节，虽然对于完整理解扩散模型十分重要，但并非本节讨论的重点。

本节也不旨在覆盖扩散模型的所有理论与实现细节，而是希望建立一套统一的连续时间观点：理解随机扰动如何推动概率分布演化，正向过程与反向生成过程如何建立联系，以及随机动力学如何进一步关联到确定性的概率流描述。通过这一视角，扩散模型可以被放置在更一般的连续时间生成框架中，并与前文介绍的确定性概率输运方法形成对照。

图 4.13 从结构上对比了自编码器的单次编码-解码映射与扩散模型的多时刻概率演化。这里的重点并非把两者视为同一种模型，而是突出扩散模型在数据分布与先验分布之间显式定义了正向扰动和反向生成过程。

下面先建立随机微分方程的基本构成，再依次讨论正向扩散、反向 SDE、分数学习与概率流 ODE，最后回到数值采样问题，说明连续时间模型如何落实为有限步生成算法。

重点提示

新概念：分数函数 $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ 指向当前密度增长最快的方向。它把正向 SDE 造成的概率扩散转化为可逆转的局部信息：将其加入反向漂移后，可以从简单先验逐步恢复数据分布。

重难点解析：反向 SDE 的漂移修正系数为 $g(t)^2$ ，而共享相同边缘分布的概率流 ODE 使用 $\frac{1}{2}g(t)^2$ 。二者的等价仅指每个时刻的边缘分布一致，并不表示样本路径相同。推导和实现时还要同时核对反向时间增量的符号，避免把“方程中的漂移方向”和“数值积分的时间方向”混为一谈。

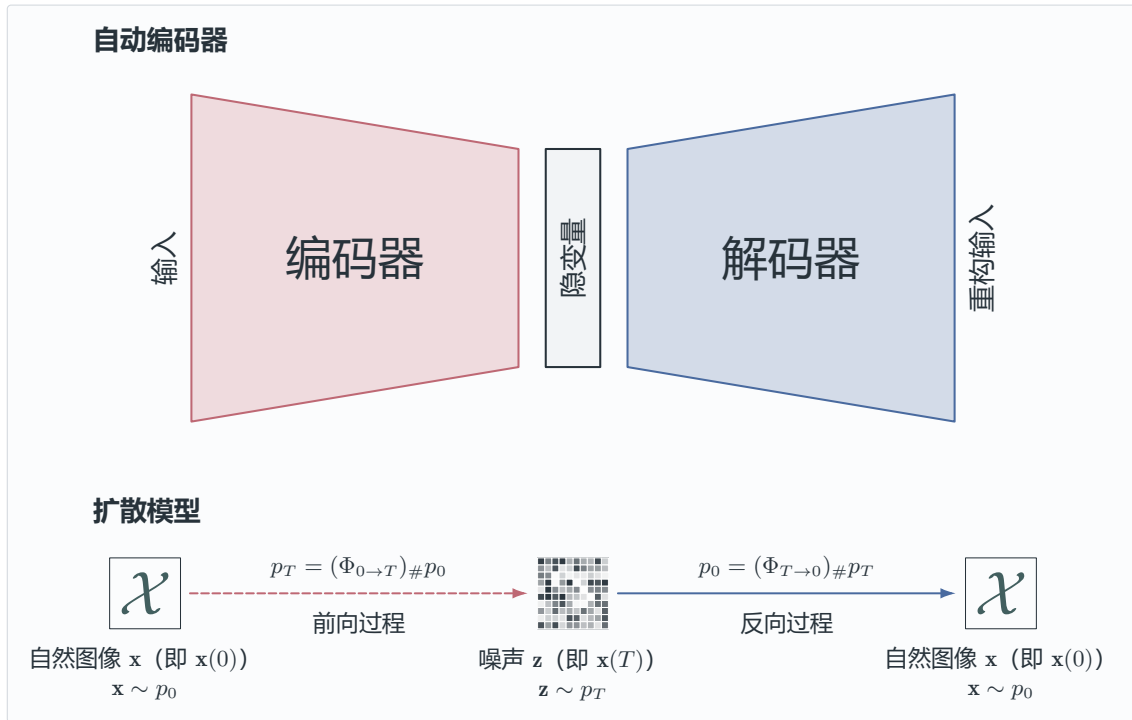


图 4.13: 自编码器与扩散模型的对比。上部为自编码器通过编码器和解码器完成的数据重构；下部为扩散模型在数据分布 p_0 与先验分布 p_T 之间建立的正向扰动过程和反向生成过程。

4.4.1 随机微分方程

常微分方程只用向量场描述给定状态处的确定性变化率，不足以表达持续注入的随机扰动。随机微分方程在确定性漂移之外加入由布朗运动驱动的扩散项，从而把“平均演化趋势”和“随机波动强度”纳入同一个连续时间模型。本小节将先澄清漂移、扩散系数和随机增量各自的作用，为区分样本路径与边缘分布的演化作准备。

随机微分方程的基本形式: 随机微分方程 (SDE) 是一种包含一个或多个随机过程项的微分方程。通常，它描述一个同时受到确定性力（可预测的趋势）和随机噪声（不可预测的波动）影响的系统的演化。

为了理解这一点，首先回顾描述状态向量 $\mathbf{x}(t)$ 随时间 t 演化的标准常微分方程 (ODE): $\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t)$ 。这里， $f(\cdot)$ 是一个描述状态速度的确定性向量场。给定初始条件 $\mathbf{x}(0)$ ，未来的轨迹就完全确定了。

然而，在扩散模型中，需要持续地向数据引入随机噪声。为了形式化这一点，必须在方程中添加一个随机项。由于标准布朗运动处处不可微，不能简单地写成 $\frac{d\mathbf{x}(t)}{dt} = f + \text{noise}$ 。相反，应以包含随机分量的微分形式写出方程：

$$d\mathbf{x}(t) = \underbrace{f(\mathbf{x}(t), t)dt}_{\text{deterministic}} + \underbrace{g(t)d\mathbf{w}(t)}_{\text{stochastic}}. \quad (4.85)$$

这是一个通用的伊藤 (Itô) SDE。该方程的组成部分定义如下：

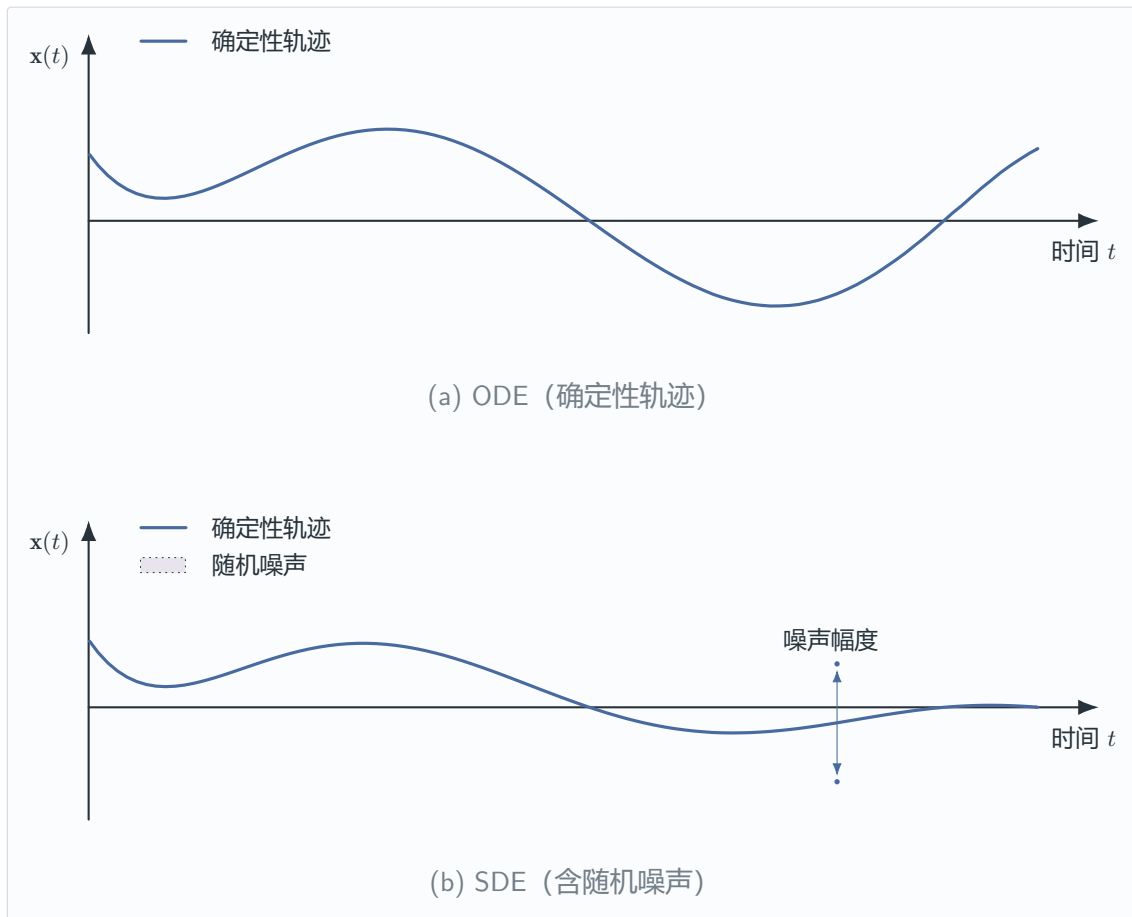


图 4.14: 确定性轨迹与随机轨迹对比。ODE 为给定初值指定确定性轨迹；SDE 的扩散项持续注入高斯噪声，使相同初始状态可对应不同的样本轨迹。

- 漂移系数 $f(\mathbf{x}(t), t) \in \mathbb{R}^d$ 表示演化的确定性部分，类似于常微分方程中的向量场。它决定了演化的一般趋势。
- 扩散系数 $g(t)$ 通常是一个标量函数，用于控制在时间 t 注入系统的随机噪声的幅度。
- 维纳过程（也称为布朗运动） $\mathbf{w}(t) \in \mathbb{R}^d$ 表示随机性的来源。增量 $d\mathbf{w}(t)$ 可以视为一个均值为 $\mathbf{0}$ 、协方差为 $dt \cdot \mathbf{I}$ 的无穷小高斯噪声向量。

方程 (4.85) 被称为前向 SDE。通过使用这个 SDE，我们定义了一个前向过程，当时间 t 从 0 流向 T 时，该过程逐渐向数据样本 $\mathbf{x}(0) \sim p_{\text{data}}$ 添加噪声。在此过程中，分布 $p_t(\mathbf{x}(t))$ 扩散，并且 $\mathbf{x}(T)$ 的分布最终将收敛到一个已知的先验分布（通常是 $\mathcal{N}(\mathbf{0}, \mathbf{I})$ ），如图 4.14 所示。

4.4.2 正向扩散过程

有了随机动力学的一般形式，还需要选择怎样的漂移和扩散强度，才能把复杂的数据分布平稳地转化为简单先验。正向扩散过程通过随时间逐步削弱数据信号并增加

噪声来完成这一任务，其噪声调度决定中间分布、终点先验以及后续反演的难度。本小节将从这一设计目标出发说明正向过程如何构造训练所需的带噪样本。

方差爆炸与方差保持随机微分方程：现在转向定义前向随机微分方程，即在方程(4.85)中定义漂移系数 $f(\mathbf{x}(t), t)$ 和扩散系数 $g(t)$ 。构建前向 SDE 有多种方法，这里考虑两种常用方法，它们分别对应噪声条件分数网络 (NCSN) 和去噪扩散概率模型 (DDPM) 的连续时间极限，称为**方差爆炸 (VE)**和**方差保持 (VP)** SDE。

在 **NCSN** 框架中，前向过程通过一系列递增的高斯噪声水平 $\{\sigma_1, \sigma_2, \dots, \sigma_N\}$ (其中 $\sigma_1 < \sigma_2 < \dots < \sigma_N$) 扰动数据。考虑一个离散过程，其中第 i 步的状态 $\mathbf{x}_i$ 通过从 $\mathbf{x}_{i-1}$ 添加必要噪声量使总方差从 σ_{i-1}^2 增加到 σ_i^2 演化而来，其更新规则为：

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2} \boldsymbol{\epsilon}_i, \quad (4.86)$$

其中 $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 表示第 i 步的增量噪声。该模型确保若从数据 $\mathbf{x}_0$ 出发，第 i 步在给定 $\mathbf{x}_0$ 条件下的分布具有 σ_i^2 的方差。

现在转向连续极限。定义关于时间 t 的连续方差函数 $\sigma(t)$ 。当步数趋近无穷大 ($N \rightarrow \infty$) 时，离散时间间隔趋近于零 ($\Delta t \rightarrow 0$)。相邻步间的方差差变为微分：

$$\sigma_i^2 - \sigma_{i-1}^2 \approx \frac{d\sigma^2(t)}{dt} \Delta t. \quad (4.87)$$

回忆随机微积分中，维纳过程增量 $d\mathbf{w}$ 与 $\sqrt{\Delta t}$ 成比例 (即 $\sqrt{\Delta t} \boldsymbol{\epsilon}_i \approx d\mathbf{w}(t)$)。通过替换方差差可重写更新规则：

$$\mathbf{x}_i - \mathbf{x}_{i-1} \approx \sqrt{\frac{d\sigma^2(t)}{dt} \Delta t} \boldsymbol{\epsilon}_i. \quad (4.88)$$

取极限 $\Delta t \rightarrow 0$ 后，得到连续时间 SDE：

$$d\mathbf{x}(t) = \sqrt{\frac{d\sigma^2(t)}{dt}} d\mathbf{w}(t). \quad (4.89)$$

该 SDE 中漂移系数定义为 $f(\mathbf{x}(t), t) = \mathbf{0}$ ，扩散系数为 $g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$ 。由于分布方差随 $t \rightarrow T$ 单调递增，此 SDE 将数据分布映射至具有不断增大方差的噪声分布，因此称为**方差爆炸 SDE**。

以图像生成为例，可以把 $\mathbf{x}_0$ 看作一张清晰图像的像素向量。随着时间 t 增大，系统不断向图像中注入高斯噪声：开始时，图像中的轮廓和纹理仍然清晰可辨；随后，细节逐渐被噪声掩盖；当 t 接近终止时刻 T 时，原始图像的语义结构几乎完全消失，状态接近高方差的随机噪声。

这里的扩散系数

$$g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$$

可以理解为单位时间内噪声方差增加的速度： $\sigma^2(t)$ 增长越快，图像在该时刻受到的随机扰动就越强。因此，方差爆炸 SDE 描述的是一个连续加噪过程：样本没有确定性的漂移趋势，而是在不断累积的随机扰动下，由具有复杂结构的数据逐渐演化为高方差噪声。（缺图）

DDPM 框架采用不同的噪声注入策略。不同于简单地增加方差噪声，它在每一步重新缩放数据以保持方差有界。在离散公式中，前向过程由方差调度 $0 < \beta_1 < \beta_2 < \dots < \beta_N < 1$ 定义，从 $\mathbf{x}_{i-1}$ 到 $\mathbf{x}_i$ 的转移由下式给出：

$$\mathbf{x}_i = \sqrt{1 - \beta_i} \mathbf{x}_{i-1} + \sqrt{\beta_i} \boldsymbol{\epsilon}_i, \quad (4.90)$$

其中 $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 。与 NCSN 公式不同，系数 $\sqrt{1 - \beta_i}$ 会收缩前一状态的均值。该设计确保若输入数据服从标准正态分布，则每步边缘分布仍保持标准正态，即方差得以保持。

此时增量 $\mathbf{x}_i - \mathbf{x}_{i-1}$ 可表示为：

$$\mathbf{x}_i - \mathbf{x}_{i-1} = (\sqrt{1 - \beta_i} - 1) \mathbf{x}_{i-1} + \sqrt{\beta_i} \boldsymbol{\epsilon}_i. \quad (4.91)$$

为获得连续时间 SDE，再次令 $N \rightarrow \infty$ 并引入连续函数 $\beta(t)$ 使得 $\beta_i \approx \beta(t) \Delta t$ 。利用泰勒展开近似 $\sqrt{1 - x} \approx 1 - \frac{1}{2}x$ （当 x 较小时），有 $\sqrt{1 - \beta(t) \Delta t} - 1 \approx -\frac{1}{2} \beta(t) \Delta t$ 。将其代入式(4.91) 并应用缩放规则 $\sqrt{\beta(t) \Delta t} \boldsymbol{\epsilon}_i \approx \sqrt{\beta(t)} d\mathbf{w}(t)$ ，可得极限：

$$d\mathbf{x}(t) = -\frac{1}{2} \beta(t) \mathbf{x}(t) dt + \sqrt{\beta(t)} d\mathbf{w}(t). \quad (4.92)$$

此 SDE 中漂移系数为 $f(\mathbf{x}(t), t) = -\frac{1}{2} \beta(t) \mathbf{x}(t)$ ，扩散系数为 $g(t) = \sqrt{\beta(t)}$ 。漂移项作为恢复力将状态拉向原点（零均值），从而抵消扩散项注入的方差。因此，若初始分布具有单位方差，整个过程方差将保持恒定。故式(4.92) 称为方差保持 SDE。

为简洁起见，此处未明确定义参数 $\{\sigma_i\}$ （或 $\sigma(t)$ ）和 $\{\beta_i\}$ （或 $\beta(t)$ ）的函数形式。值得注意的是，DDPM 中常引入辅助变量（如 $\alpha_i = 1 - \beta_i$ 和 $\bar{\alpha}_i = \prod_{s=1}^i \alpha_s$ ）以简化推导与实现。调度参数的精心设计可使模型在终止时刻接近预先选定的先验分布。

虽然这里重点是通过连续时间 SDE 公式建立统一的理论基础，但实践中扩散模型通常通过离散化步骤实现（见表 4.4）。NCSN 和 DDPM 方法中引入的离散变量 $\mathbf{x}_i$ 本质上是连续过程的数值近似。实际实现中，通过迭代这些离散时间步 $\{\mathbf{x}_i\}$ 来模拟 SDE 演化。

4.4.3 反向 SDE 与分数函数

正向过程能够产生从数据到噪声的随机轨迹，生成任务却要求从先验噪声回到数据分布。随机过程的时间反转表明，反向 SDE 除了包含前向漂移和扩散信息，还依赖各时刻对数密度关于状态的梯度，即分数函数。因此，反向生成的核心困难从“如

条目	方差爆炸 (NCSN)	方差保持 (DDPM)
漂移系数 $f(\mathbf{x}(t), t)$	$\mathbf{0}$	$-\frac{1}{2}\beta(t)\mathbf{x}(t)$
扩散系数 $g(t)$	$\sqrt{\frac{d\sigma^2(t)}{dt}}$	$\sqrt{\beta(t)}$
连续时间 SDE	$d\mathbf{x}(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}d\mathbf{w}(t)$	$d\mathbf{x}(t) = -\frac{1}{2}\beta(t)\mathbf{x}(t)dt + \sqrt{\beta(t)}d\mathbf{w}(t)$
迭代扩散 (离散)	$\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\epsilon_i$	$\mathbf{x}_i = \sqrt{1 - \beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\epsilon_i$
单步扩散 (离散)	$\mathbf{x}_i = \mathbf{x}_0 + \sigma_i\epsilon$	$\mathbf{x}_i = \sqrt{\bar{\alpha}_i}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_i}\epsilon$

表 4.4: VP 和 VE SDE 的连续时间与离散时间形式。 $\mathbf{x}(t)$ 、 $\mathbf{w}(t)$ 、 $\sigma(t)$ 和 $\beta(t)$ 是时间 t 的连续函数； $\mathbf{x}_i$ 是离散时间步 i 的状态。

何添加噪声”转化为“如何获得未知中间分布的分数”，这也自然引出下一小节的学习方法。

在本节采用的状态无关标量扩散系数 $g(t)$ 的设定下，前向 SDE 存在一个对应的反向时间 SDE。如果从 $t = T$ 到 0 反向模拟这个过程，就可以从先验分布生成数据样本。反向 SDE 由下式给出：

$$d\mathbf{x}(t) = [f(\mathbf{x}(t), t) - g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))] dt + g(t) d\bar{\mathbf{w}}(t). \quad (4.93)$$

其中 dt 表示一个无穷小的负时间增量， $d\bar{\mathbf{w}}(t)$ 是反向时间中的一个标准维纳过程，而 $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ 是概率密度对数相对于数据的梯度，也称为分数函数。给定一个前向 SDE（方程(4.85)），如果能够访问分数函数，就可以使用反向 SDE（方程(4.93)）将样本从先验噪声 p_T 传输回数据分布 p_0 。

将反向 SDE 用于生成建模，需要具备两个基本要素。首先，需要选取合适的前向 SDE，使复杂的数据分布 p_0 能够逐渐转化为易于采样的先验分布 p_T ；上一小节介绍的 VE SDE 与 VP SDE 正是两类常用构造。其次，需要获得反向漂移项中的分数函数

$$\mathbf{s}_t(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}). \quad (4.94)$$

一旦这两个要素确定，便可以从先验分布出发，通过数值求解反向 SDE 逐步生成数据样本。

因此，在选定前向扩散过程之后，反向生成所面临的主要困难便集中在分数函数的获取上。中间时刻的边缘分布 $p_t(\mathbf{x})$ 由数据分布经过随机扩散得到，通常既没有可计算的归一化密度，也无法直接求出其对数梯度。实际训练中能够获得的通常只有来自 p_0 的数据样本，而不是真实分数 $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ 。这意味着，分数函数无法像普通监督学习中的标签一样直接使用。

4.4.4 去噪分数匹配

反向 SDE 表明，生成过程所需要获得的核心对象是各时刻边缘分布的分数函数

$$\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)).$$

然而，边缘分布 $p_t(\mathbf{x}(t))$ 由数据分布经过随机扩散得到，通常没有可直接计算的密度表达式，因而无法将其真实分数作为监督信号。

前向扩散过程是人为设计的，因此，给定原始数据 $\mathbf{x}(0)$ 后，条件转移分布

$$p(\mathbf{x}(t) | \mathbf{x}(0))$$

及其条件分数通常容易计算。去噪分数匹配正是利用这一性质，以已知的条件分数构造训练目标，从而避免直接计算未知边缘分布的分数，并将分数估计转化为可以由神经网络求解的回归问题。

以加性高斯扰动为例，在时刻 t ，带噪样本可以写为

$$\mathbf{x}(t) = \mathbf{x}(0) + \sigma(t)\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (4.95)$$

因此，其条件转移分布为

$$p(\mathbf{x}(t) | \mathbf{x}(0)) = \mathcal{N}(\mathbf{x}(0), \sigma(t)^2 \mathbf{I}). \quad (4.96)$$

该条件分布关于带噪状态 $\mathbf{x}(t)$ 的分数可以显式计算为

$$\begin{aligned} \nabla_{\mathbf{x}(t)} \log p(\mathbf{x}(t) | \mathbf{x}(0)) &= -\frac{\mathbf{x}(t) - \mathbf{x}(0)}{\sigma(t)^2} \\ &= -\frac{\boldsymbol{\epsilon}}{\sigma(t)}. \end{aligned} \quad (4.97)$$

因此，只需从数据集中采样 $\mathbf{x}(0)$ ，再按照式 (4.95) 生成带噪样本，便可以同时得到可计算的条件分数目标。去噪分数匹配由此绕开了未知边缘密度 $p_t(\mathbf{x}(t))$ 的显式求解。

去噪分数匹配的基本等价性表明，训练一个神经网络 $s_\theta(\mathbf{x}(t), t)$ 来近似这个条件分数，等价于学习边缘分数 $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ 。因此，生成建模的目标可以有效地简化为一个去噪回归问题。

形式上，希望最小化估计分数与真实条件分数之间的期望平方欧几里得距离：

$$\mathcal{L}(\theta) = \mathbb{E}_{t, \mathbf{x}(0), \boldsymbol{\epsilon}} \left[\lambda(t) \left\| s_\theta(\mathbf{x}(t), t) - \left(-\frac{\boldsymbol{\epsilon}}{\sigma(t)} \right) \right\|^2 \right], \quad (4.98)$$

其中 $\lambda(t)$ 是一个正权重函数。一个常见的选择是 $\lambda(t) = \sigma(t)^2$ ，这可以平衡不同噪声水平下分数的大小。

通过代入此权重并重新整理项，目标简化为：

$$\mathcal{L}(\theta) = \mathbb{E}_{t, \mathbf{x}(0), \epsilon} [\|\epsilon + \sigma(t)s_\theta(\mathbf{x}(0)) + \sigma(t)\epsilon, t\|^2]. \quad (4.99)$$

令噪声预测网络满足 $\epsilon_\theta(\mathbf{x}(t), t) = -\sigma(t)s_\theta(\mathbf{x}(t), t)$ ，上述目标就是标准高斯噪声 ϵ 的均方误差预测目标。

上述方法与**基于能量的模型**（EBM）存在理论联系。在统计物理学中，概率分布通常通过能量函数 $E_\theta(\mathbf{x})$ 表示为 $p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z_\theta}$ ，其中 Z_θ 是归一化常数或配分函数。

对分数函数而非似然本身进行建模的一个优点是，它可以绕开难以处理的归一化常数 Z_θ 的计算。对于图像这样的高维连续数据，计算 $Z_\theta = \int \exp(-E_\theta(\mathbf{x})) d\mathbf{x}$ 通常难以处理。基于分数的模型避免了这个问题，因为对数配分函数关于数据的梯度为零，即 $\nabla_{\mathbf{x}} \log Z_\theta = 0$ 。因此有

$$\begin{aligned} \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) &= \nabla_{\mathbf{x}} (-E_\theta(\mathbf{x}) - \log Z_\theta) \\ &= -\nabla_{\mathbf{x}} E_\theta(\mathbf{x}). \end{aligned} \quad (4.100)$$

这一特性允许在不评估配分函数的情况下训练未归一化的模型。

一旦分数网络 $s_\theta(\mathbf{x}, t)$ 训练完成，就可以用它来生成新样本。具体来说，将学习到的近似值 $s_\theta(\mathbf{x}(t), t) \approx \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ 代入方程(4.93)，得到

$$d\mathbf{x}(t) = [f(\mathbf{x}(t), t) - g(t)^2 s_\theta(\mathbf{x}(t), t)] dt + g(t) d\bar{\mathbf{w}}(t). \quad (4.101)$$

在测试时，从先验分布（例如 $\mathcal{N}(\mathbf{0}, \mathbf{I})$ ）初始化 $\mathbf{x}(T)$ ，并使用欧拉-丸山方法等数值求解器，从 $t = T$ 到 $t = 0$ 反向积分该方程。通过逐步去除噪声，这个过程将初始的噪声样本转化为目标数据分布的样本。

4.4.5 概率流 ODE

分数模型训练完成后，反向 SDE 给出一种随机采样方式，但同一族边缘概率分布并不只对应随机样本路径。通过把扩散对密度的作用重新写成确定性的概率通量，可以构造与该 SDE 具有相同边缘分布的概率流 ODE。这一结果并不意味着两者的单条轨迹相同，而是为同一分布演化提供了随机与确定性两种实现，并使前一节的 ODE 求解工具能够再次用于生成。

基于分数的生成模型中一个重要的理论结果是：扩散型随机微分方程描述的随机过程具有边缘分布等价的确定性表示。对于本节形式的 SDE，即扩散系数是状态无关的标量函数 $g(t)$ ，存在一个对应的确定性 ODE，其轨迹产生的边缘概率密度 $\{p_t(\mathbf{x}(t))\}_{t=0}^T$ 与随机微分方程相同。

该常微分方程被称为概率流 ODE，其表达式为

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t) - \frac{1}{2}g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t)). \quad (4.102)$$

推导过程依赖于 Fokker-Planck 方程（又称 Kolmogorov 前向方程），该方程描述随机微分方程动力学下概率密度函数 $p_t(\mathbf{x})$ 的时间演化。可以证明，式(4.102) 中 ODE 导出的连续性方程与式(4.85) 中 SDE 的 Fokker-Planck 方程完全一致。因此，如果采样 $\mathbf{x}(T) \sim p_T$ 并逆向求解该 ODE（从 T 到 0），所得样本 $\mathbf{x}(0)$ 将服从数据分布 p_0 ，这与逆向随机微分方程在边缘分布上的结果相同。

这种等价性为随机视角与确定性视角建立了联系。需注意式(4.102) 依赖于分数函数 $\nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$ 。这意味着一旦训练好分数模型 $s_\theta(\mathbf{x}(t), t)$ ，即可直接用于构建概率流 ODE：

$$\frac{d\mathbf{x}(t)}{dt} \approx f(\mathbf{x}(t), t) - \frac{1}{2}g(t)^2 s_\theta(\mathbf{x}(t), t). \quad (4.103)$$

在实际应用中，相比随机微分方程方法，采用常微分方程进行采样具有多项优势。首先，给定固定初始噪声向量 $\mathbf{x}(T)$ 时，生成过程是确定性的。因此可以获得唯一的流映射，并能通过正向积分 ODE 将真实样本反演为对应的潜在噪声表示。其次，ODE 的轨迹通常比 SDE 更平滑，可以采用具有自适应步长的常微分方程求解器。第三，利用瞬时变量变换公式，可以计算概率流 ODE 下数据的对数似然。

由 Fokker-Planck 方程推导概率流 ODE。

下面给出概率流 ODE 的详细推导。考虑定义的前向 SDE

$$d\mathbf{x}(t) = f(\mathbf{x}(t), t)dt + g(t)d\mathbf{w}(t). \quad (4.104)$$

其中 $f(\mathbf{x}(t), t)$ 是漂移系数， $g(t)$ 是扩散系数， $\mathbf{w}(t)$ 是标准维纳过程。为简化符号，下文省略 $\mathbf{x}(t)$ 中的参数 t 。

状态变量 $\mathbf{x}$ 的概率密度函数 $p_t(\mathbf{x})$ 的时间演化由 Fokker-Planck 方程控制：

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla \cdot [f(\mathbf{x}, t)p_t(\mathbf{x})] + \frac{1}{2}g(t)^2 \Delta p_t(\mathbf{x}). \quad (4.105)$$

其中 $\nabla \cdot$ 表示散度算子， $\Delta = \nabla \cdot \nabla$ 是拉普拉斯算子。等式右边的第一项表示由于漂移引起的概率质量平流，第二项表示由随机噪声引起的扩散。

为了推导等效的 ODE，将方程(4.105) 重写为连续性方程的形式。连续性方程仅涉及一阶导数，其形式为

$$\frac{\partial p_t(\mathbf{x})}{\partial t} = -\nabla \cdot [\tilde{f}(\mathbf{x}, t)p_t(\mathbf{x})], \quad (4.106)$$

其中 $\tilde{f}(\mathbf{x}, t)$ 对应于确定性 ODE 的向量场。

关键步骤是将扩散项 $\Delta p_t(\mathbf{x})$ （涉及二阶导数）表示为一阶项的散度。利用恒等式 $\nabla p_t(\mathbf{x}) = p_t(\mathbf{x}) \nabla \log p_t(\mathbf{x})$ 将拉普拉斯算子重写为

$$\begin{aligned}\Delta p_t(\mathbf{x}) &= \nabla \cdot (\nabla p_t(\mathbf{x})) \\ &= \nabla \cdot (p_t(\mathbf{x}) \nabla \log p_t(\mathbf{x})).\end{aligned}\quad (4.107)$$

将此恒等式代回 Fokker-Planck 方程(4.105)，可以将各项组合在单个散度算子下：

$$\begin{aligned}\frac{\partial p_t(\mathbf{x})}{\partial t} &= -\nabla \cdot [f(\mathbf{x}, t) p_t(\mathbf{x})] + \frac{1}{2} g(t)^2 \nabla \cdot [p_t(\mathbf{x}) \nabla \log p_t(\mathbf{x})] \\ &= -\nabla \cdot \left[f(\mathbf{x}, t) p_t(\mathbf{x}) - \frac{1}{2} g(t)^2 p_t(\mathbf{x}) \nabla \log p_t(\mathbf{x}) \right] \\ &= -\nabla \cdot \left[\left(f(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(\mathbf{x}) \right) p_t(\mathbf{x}) \right].\end{aligned}\quad (4.108)$$

将此结果与方程(4.106) 中的连续性方程进行比较，如果将有效的确定性向量场定义为

$$\tilde{f}(\mathbf{x}, t) = f(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(\mathbf{x}), \quad (4.109)$$

则两个方程匹配。因此，该确定性 ODE 产生的边缘分布 $p_t(\mathbf{x})$ 与原始 SDE 定义的随机过程相同，即

$$\frac{d\mathbf{x}}{dt} = \tilde{f}(\mathbf{x}, t) = f(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(\mathbf{x}). \quad (4.110)$$

4.4.6 数值采样与高效生成

无论采用反向 SDE 还是概率流 ODE，训练得到的模型都只提供局部的分数或向量场，最终样本仍需通过数值求解逐步累积这些局部变化。离散步数过少会放大求解误差，步数过多又会增加神经网络评估成本。因此，本小节将从求解格式、步长选择和轨迹形状三个方面讨论采样效率，并说明高阶或专用求解器、预测-校正以及轨迹简化方法如何在速度与精度之间取得平衡。

反向 SDE 的数值求解。

为了从训练好的扩散模型中生成样本，需要从初始样本 $\mathbf{x}(T) \sim p_T$ 出发，逆向模拟反向时间 SDE（式(4.93)）直至 $t = 0$ 。由于复杂数据分布通常无法获得闭式解，需要采用数值求解器来近似连续轨迹。

最基础且广泛使用的方法是 Euler-Maruyama 方法，该方法将欧拉法推广至 SDE 情形。首先将时间区间 $[0, T]$ 离散化为 N 步： $t_N = T > t_{N-1} > \dots > t_0 = 0$ 。对于正

的步长 $\Delta t = t_{i+1} - t_i$ ，从 t_{i+1} 到 t_i 的反向 SDE 可通过下式近似：

$$\mathbf{x}_{t_i} \approx \mathbf{x}_{t_{i+1}} - [f(\mathbf{x}_{t_{i+1}}, t_{i+1}) - g(t_{i+1})^2 s_\theta(\mathbf{x}_{t_{i+1}}, t_{i+1})] \Delta t + g(t_{i+1}) \sqrt{\Delta t} \boldsymbol{\epsilon}_{i+1}. \quad (4.111)$$

其中 $\boldsymbol{\epsilon}_{i+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 。需要注意的是，由于这里进行的是时间逆向积分，数值求解器中的计算步骤对应于物理时间 t 的负增量。将 Euler-Maruyama 离散化应用于反向 VP SDE，可得到与离散扩散模型采样规则相联系的更新形式。

SDE 框架的一个优势在于能够将数值积分与马尔可夫链蒙特卡洛 (MCMC) 方法结合。例如，通过使用预测-校正方法，可将数值离散化与基于分数的校正解耦。在预测步骤中，数值求解器根据当前状态 $\mathbf{x}_{t_{i+1}}$ 估计下一时间步 $\mathbf{x}_{t_i}$ 的状态，该步骤使过程沿时间逆向演化。在预测步骤之后，对 $\mathbf{x}_{t_i}$ 应用 MCMC 方法（如朗之万动力学）进行多次迭代。分数函数 $s_\theta(\mathbf{x}_{t_i}, t_i)$ 提供分布 p_{t_i} 的对数密度梯度，因此运行朗之万动力学可将预测样本推向边缘分布 p_{t_i} 的高密度区域，同时保持时间 t_i 不变。

与流匹配路径的比较。

虽然从随机微分方程导出的概率流 ODE 提供了确定性的生成过程，但它本质上是随机扩散过程的副产品。方程(4.102) 中的向量场

$$f(\mathbf{x}(t), t) - \frac{1}{2}g(t)^2 \nabla_{\mathbf{x}(t)} \log p_t(\mathbf{x}(t))$$

由前向扩散 SDE（例如方差爆炸或方差保持）的具体选择决定。这引发了一个问题：能否直接学习概率流 ODE 的向量场，而不依赖于中间的扩散公式？此外，能否设计这个向量场使其具有更直的轨迹，从而更容易进行数值积分？

这催生了流匹配框架。流匹配是一种无模拟训练连续归一化流的方法，它允许灵活地定义概率路径。其核心思想是直接用一个神经网络向量场 $v(\mathbf{x}(t), \theta, t)$ （或 $v_\theta(\mathbf{x}(t), t)$ ）去回归一个能够生成期望概率路径 $p_t(\mathbf{x}(t))$ 的目标向量场 $u_t(\mathbf{x}(t))$ 。基于流匹配的方法与基于分数的方法可作如下比较：

- **基于分数的方法。**前向过程是一个固定的高斯扩散过程。对应的概率流 ODE 具有一个由漂移项和分数项组成的向量场。由此产生的轨迹通常较为弯曲，因为扩散过程按照固定的调度破坏信息。因此，求解反向 ODE 往往需要较多步数或专用求解器来精确追踪这些路径。
- **流匹配方法。**与由随机微分方程决定动力学不同，流匹配允许直接设计路径。两点之间最简单的传输映射是一条直线；对于给定端点对，沿线性插值路径的条件向量场在时间上为常数。这样的 ODE 通常更容易数值求解。

图 4.15 从轨迹几何和数值离散的角度概括了上述差异。

高阶与专用求解器。

将扩散模型部署到实际应用中的一个瓶颈是其在推理阶段的高计算成本。提高生成效率的方法包括高阶数值求解器、轨迹修正和一致性模型。

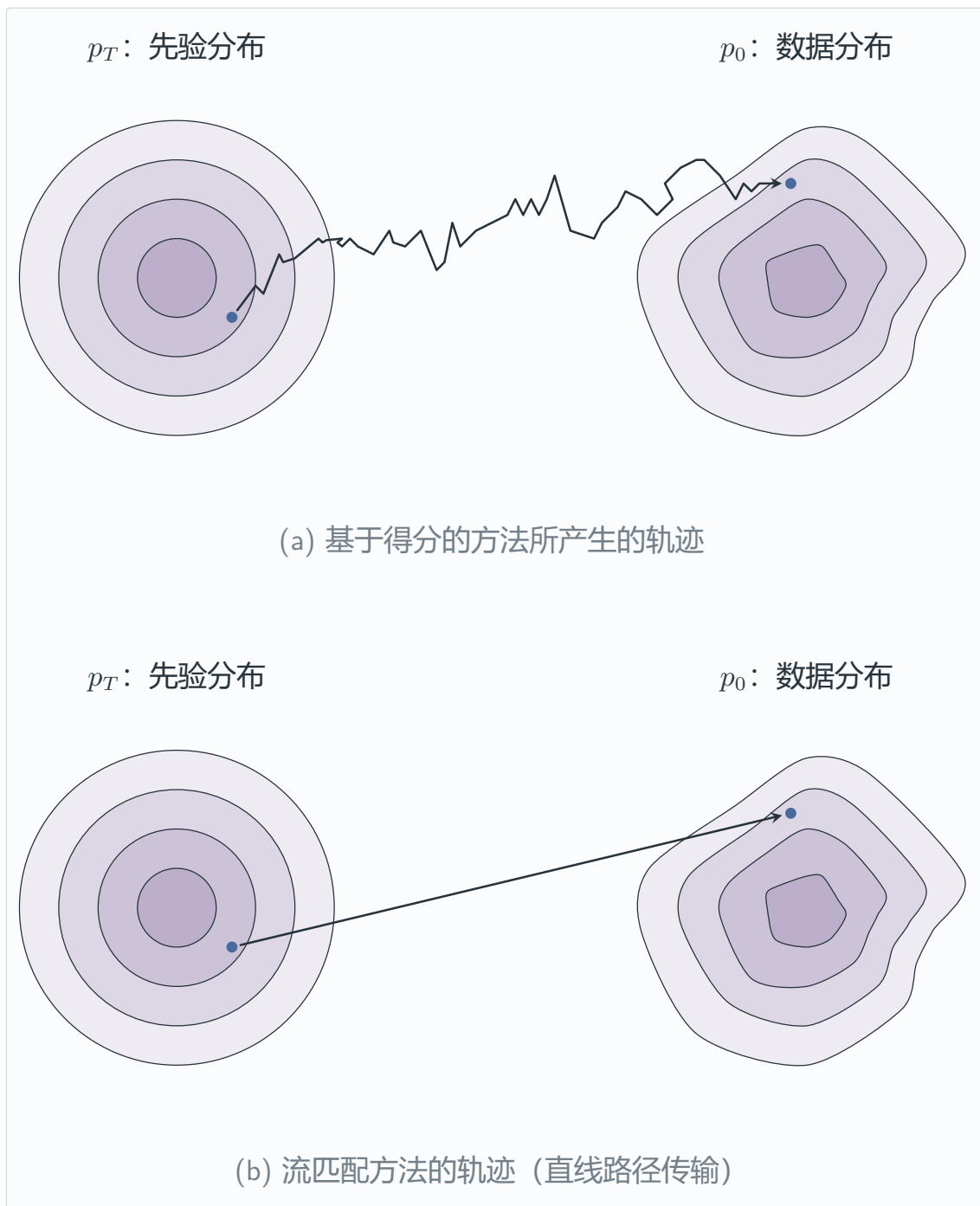


图 4.15: 从先验噪声分布 p_T 到数据分布 p_0 的生成轨迹对比。基于分数的方法经由扩散过程确定概率路径；流匹配则允许显式设计条件路径。

最直接的加速方法之一是采用高阶数值求解器。当轨迹弯曲时，像一阶欧拉法这样的标准求解器需要许多步才能减小离散化误差。为了解决这个问题，可以利用扩散 ODE 特有的数学结构构造高阶求解器。

例如，DPM-Solver 和 DPM-Solver++ 利用扩散 ODE 的半线性结构来解析计算解的线性部分，从而将非线性部分简化为指数加权积分。其核心原理在于常数变易公式，该公式将 ODE 分离为线性漂移项和非线性的分数项。通过解析求解线性部分，

求解器只需使用 $(k - 1)$ 阶多项式来近似非线性分数函数。DPM-Solver++ 通过使用数据预测参数化和动态阈值处理来防止解漂移到有效数据范围之外，从而进一步提高稳定性。DPM-Solver-v3 则采用预测器-校正器框架，通过内部预测步骤细化积分近似，并结合由经验模型统计确定的系数改善少步采样。

除了上述方法，还可从几何和数学结构出发设计求解器。近似平均方向求解器 (AMED-Solver) 利用采样轨迹通常位于低维子空间这一观察，通过学习预测流的平均方向减少求解步数。扩散指数积分采样器 (DEIS) 在变换后的对数 ρ 空间中拟合多项式，以减小离散化误差。统一预测器-校正器方法 (UniPC) 引入校正步骤并重用预测器中已经评估过的噪声预测，从而在不增加神经函数评估的情况下提高精度阶数。扩散模型的伪数值方法 (PNDM) 将去噪过程视为在数据高密度区域对应的流形上求解微分方程，并将数值步骤分解为梯度部分和非线性转移部分。此外，阐明扩散模型 (EDM) 使用二阶 Heun 求解器，并通过将网络输入、输出和损失缩放到单位方差来稳定训练动态。

轨迹修正。

流匹配旨在学习一个向量场 $v_\theta(\mathbf{x}(t), t)$ ，该向量场能够诱导一个概率流，从而将简单的先验分布 p_T 转换为复杂的数据分布 p_0 。其核心目标是最小化学到的速度场与将样本沿选定路径传输的真实条件速度之间的差异。然而，实际轨迹还受到数据分布 p_0 与噪声分布 p_T 之间耦合方式的影响。

在初始训练阶段，通常称为 1-修正流，数据点 $\mathbf{x}(0)$ 和噪声点 $\mathbf{x}(T)$ 从其各自的分布中随机配对。这种随机配对导致高维潜在空间中的条件轨迹经常相交。当多条直线轨迹在时空中的同一点 $\mathbf{x}(t)$ 相交时，学到的边缘向量场 $v_\theta(\mathbf{x}(t), t)$ （即这些不同相交速度的期望）会变得弯曲。因此，在积分 ODE 时实际遵循的路径不再是直的，这会使采用大步长时产生更多离散化误差。

在修正流框架中，可以通过减少条件路径之间的干扰来改善噪声分布与数据分布之间的耦合。首先，使用随机配对 $(\mathbf{x}(0), \mathbf{x}(T))$ 训练一个 1-修正流 v_θ^1 。然后，使用训练好的模型 v_θ^1 生成一个新的合成数据集。具体来说，对于一组噪声样本 $\{\mathbf{x}_i(T)\}$ ，沿物理时间 t 从 T 反向积分 ODE

$$\frac{d\mathbf{x}(t)}{dt} = v_\theta^1(\mathbf{x}(t), t)$$

以获得对应的生成数据样本 $\{\hat{\mathbf{x}}_i(0)\}$ 。等价地，若引入递增的反向时间 $\tau = T - t$ ，则该方程写为 $d\mathbf{x}/d\tau = -v_\theta^1(\mathbf{x}, T - \tau)$ 。这样就创建了一组确定性的、重新连接的样本对 $(\hat{\mathbf{x}}_i(0), \mathbf{x}_i(T))$ 。最后，使用这些对齐的样本对训练新模型 v_θ^2 ，即 2-修正流。由于 $\hat{\mathbf{x}}_i(0)$ 是通过第一个模型的动力学从 $\mathbf{x}_i(T)$ 生成的，这些潜在空间中的样本对高度对齐，因此得到的向量场更加线性。

轨迹的直线性还与学到的向量场的 Lipschitz 常数有关。如果 1-修正流是 L -Lipschitz 的，则可以防止附近的噪声样本映射到任意远的数据点。然而，在实践中， L 可能很大，导致在生成的早期阶段传输路径出现显著弯曲。这种弯曲会降低一步欧

拉采样的精度，可能需要进一步修正或采用更合适的采样调度。

为了进一步提高效率和稳定性，也可以将分治策略应用于修正流。例如，分段修正流（PeRFlow）将生成轨迹划分为几个离散的时间窗口。它不是拉直全局流，而是分别拉直每个区间内的轨迹片段；这种局部化降低了模拟训练轨迹的计算成本，并减小数值误差。

一致性模型。

一致性建模代表了一种从迭代积分转向直接映射的方法。不同于沿轨迹路径逐步求解，一致性模型旨在学习一个函数，该函数能够将概率流 ODE 轨迹上任意时间步的任意点直接映射至其原点。这种自一致性要求同一条路径上的所有点生成相同的输出，因此可以通过单步或少步生成获得样本。

一致性模型通常通过两种机制训练：一致性训练与一致性蒸馏。虽然一致性训练支持从零开始训练，但一致性蒸馏可以把预训练扩散模型（教师模型）的多步轨迹提炼至一致性模型（学生模型）。蒸馏损失通过最小化学生模型在当前时间步 t 的预测与相邻时间步预测之间的差异，将教师模型的多步轨迹压缩为较少的直接跳跃。

潜在一致性模型（LCM）将一致性蒸馏应用于高分辨率图像合成模型的潜在空间。通过提炼潜在 ODE 的概率流，LCM 在保持母模型生成能力的同时降低推理延迟。为缓解重训练大型骨干网络的计算开销，LCM-LoRA 进一步采用参数高效的微调策略，训练轻量级低秩自适应模块作为加速器。

随着采样步数增加，早期一致性模型的生成质量可能下降。轨迹一致性蒸馏（TCD）受指数积分器启发，在半线性框架中重构一致性函数，以支持对任意中间步骤的映射；分阶段一致性模型（PCM）则将轨迹划分为多个子轨迹，从而在避免随机误差累积的前提下实现确定性多步采样。

需注意的是，虽然一致性模型针对单步生成进行了优化，但它们同样支持多步一致性采样。通过执行少量采样步并对中间输出重新编码，模型可以迭代修正离散化误差，从而在推理速度与样本保真度之间提供灵活的权衡。

课后练习

1. 对伊藤 SDE $d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)d\mathbf{w}$ ，分别说明漂移、扩散系数和布朗增量的作用。它与 ODE 的样本轨迹有何根本区别？
2. 某二维 SDE 在短时间 $\Delta t = 0.01$ 内的漂移为 $(1, -2)^\top$ ，扩散系数为 3。求 Euler-Maruyama 单步增量的均值、协方差矩阵和每个坐标随机部分的标准差。
3. 若 $p_t(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \mathbf{I})$ ，求其分数函数，并解释分数向量的几何方向以及它在反向 SDE 中的作用。
4. 去噪分数匹配为什么能够使用可计算的条件分数来学习未知的边缘分数？请结合高斯扰动写出常用的条件分数监督目标。

5. 写出前向 SDE 对应的反向 SDE 与概率流 ODE，并说明二者“具有相同边缘分布”不等于“具有相同样本路径”。在采样、反演和似然计算方面它们各有什么特点？

4.5 离散状态上的连续时间动力学

前两节讨论的确定性与随机概率动力学都建立在连续状态空间上：无论样本由 ODE 还是 SDE 驱动，状态都能够在向量空间中作无穷小变化。现在转向另一类情形：时间变量仍然连续，但系统状态取自离散集合。对于文本词元、类别标签等对象，两个离散状态之间通常不存在可直接解释为有效样本的连续中间值；相应的样本轨迹不再沿光滑曲线移动，而是在连续时间中以随机时刻发生离散跳转。连续时间马尔可夫链及其生成器正是刻画这类动力学的基本工具。

语言生成同时提供了连接连续与离散方法的典型场景。一种做法是先把词元映射到连续嵌入空间，再沿用连续扩散模型；另一种做法则直接在词元空间定义破坏、去噪和跳转过程。为说明这两条路线如何与前文的概率动力学相衔接，下面先区分自回归与非自回归序列生成，随后依次讨论嵌入空间扩散、连续时间马尔可夫链、离散扩散、掩码扩散以及相关的连续松弛与流方法。

重点提示

新概念：连续时间马尔可夫链的时间变量连续、状态空间离散。生成器矩阵的非对角元素给出状态间的瞬时跳转率，对角元素保证每行和为零；它在离散状态空间中承担了类似连续向量场的角色，并诱导概率分布满足主方程。

重难点解析：嵌入扩散在连续向量空间中加噪，便于复用 SDE 和分数模型，但最终需要舍入为词元，可能出现表示平均化和长度处理问题；离散扩散直接在词元空间定义替换或掩码跳转，语义对象始终合法，却需要设计可处理的转移核与反向过程。扩散式 NAR 可以并行更新位置，但仍需多次去噪，不能把完整生成复杂度简单视为 $O(1)$ 。

4.5.1 令牌序列生成

我们首先介绍自回归（AR）生成和非自回归（NAR）生成的概念，随后说明扩散模型如何采用 NAR 式的并行位置更新。

自回归生成与非自回归生成

令牌序列生成（简称**序列生成**）是自然语言处理中的基础任务，也是**大语言模型**（LLMs）的基石 [?]。主流方法是自回归（AR）生成³。自然语言处理中通常采用从左到右的生成方式，即根据已生成的所有前序令牌逐个预测后续令牌。形式化地，设 $\mathbf{x} = \{x_1, \dots, x_n\}$ 为令牌序列，其概率通过概率链式法则定义为：

$$\begin{aligned}\Pr(\mathbf{x}) &= \Pr(x_1, \dots, x_n) \\ &= \prod_{i=1}^n \Pr(x_i | x_{<i}).\end{aligned}\quad (4.112)$$

其中 x_i 表示当前步生成的令牌， $x_{<i}$ 表示所有已生成的前序令牌。 $\Pr(x_i | x_{<i})$ 是下一令牌的条件概率，体现了从左到右生成的本质。实现 $\Pr(x_i | x_{<i})$ 的方式多样，常用方法是使用神经网络建模该概率，例如 LSTM 和 Transformer 都是神经语言建模中的经典架构。

多数文本生成任务需要基于用户提供的上下文信息进行条件生成。我们将这类上下文信息记为 $\mathbf{c}$ ，可广义视为变量序列 $\mathbf{c} = \{c_1, \dots, c_m\}$ 。此时生成任务转化为建模给定 $\mathbf{c}$ 时目标序列 $\mathbf{x}$ 的概率：

$$\Pr(\mathbf{x} | \mathbf{c}) = \prod_{i=1}^n \Pr(x_i | x_{<i}, \mathbf{c}). \quad (4.113)$$

此处 $\mathbf{c}$ 可以是离散令牌序列（如提示词）或实值向量序列（如神经网络生成的表示）。通常 $\mathbf{c}$ 可根据具体任务定义，例如在翻译中作为源语句，在问答中作为提示词，或作为外部记忆中的历史表示。

自回归生成的核心优势在于其简洁性，由此衍生出多项优良特性：既符合人类语言产出的认知过程，也适合对话、叙事等任务；还能自然处理变长序列——模型持续生成直至输出序列结束符（EOS），而非填充固定长度模板。但自回归方式长期受学界诟病的缺点包括：生成速度慢，由于第 i 步依赖第 $i-1$ 步，无法并行生成，导致 100 个令牌的句子需顺序执行 100 次前向计算，实时应用计算成本高；上下文视野受限，预测 x_i 时仅能观测左侧上下文 $x_{<i}$ ，无法前瞻右侧语义。

非自回归（NAR）生成通过打破序列依赖链提供替代方案 [?]。与逐个生成词元不同，非自回归模型并行地生成序列中的所有词元。对于最简单的单步 NAR 模型，通常假设目标词元在给定输入条件下相互独立。此时，序列 $\mathbf{x}$ 的概率可分解为：

$$\Pr(\mathbf{x} | \mathbf{c}) = \prod_{i=1}^n \Pr(x_i | \mathbf{c}) \quad (4.114)$$

³统计学中，回归指估计因变量（Y）与自变量（X）的关系。前缀 *auto* 源自希腊语“自我”，因此自回归表示用变量自身的滞后值作为自变量进行回归分析。即不通过 X 预测 Y ，而是用 $Y_{\text{昨日}}$ 预测 $Y_{\text{今日}}$



图 4.16: 自回归生成与非自回归生成对比。 $\mathbf{c} = c_1, c_2$ 表示用户提供的上下文， $\mathbf{x} = x_1, x_2, x_3, x_4$ 表示目标序列。我们的目标是对给定 $\mathbf{c}$ 条件下 $\mathbf{x}$ 的概率进行建模。(a) 在自回归生成中，词元按序预测， x_i 的预测依赖于历史信息 $\mathbf{x}_{<i}$ 和 $\mathbf{c}$ 。(b) 在非自回归生成中，序列中的所有词元并行预测。图 (b) 中的虚线表示一个由 4 个掩码词元组成的模板（给定目标长度时）。

在这种形式中，词元 x_i 的预测不依赖于其他目标词元 x_j ($j \neq i$)。因此，非自回归生成具有高效的推理性能。通过解耦词元之间的依赖关系，模型可以在单次前向传播中预测整个序列，从而将相对于序列长度的串行深度从 $O(n)$ 减少到 $O(1)$ 。这里的 $O(1)$ 只描述一次并行生成或更新；扩散式 NAR 模型仍需执行多个去噪时间步，其总采样深度并不是关于扩散步数的 $O(1)$ 。同时，多步模型可以借助不断更新的全序列状态表达位置间依赖，并不等同于始终采用上述一次性乘积分布。这种并行性使得现代 GPU 能够得到更高效的利用，因此对于延迟要求苛刻的实时应用而言，非自回归模型具有很大吸引力。此外，非自回归模型在生成过程中可以充分利用过去和未来位置的信息。图 4.16 展示了自回归生成与非自回归生成之间的区别。

然而需要指出的是，条件独立性假设也存在局限性。例如，在自然语言中，一个句子往往存在多种有效的补全方式。非自回归模型由于无法知晓其他位置已选择了哪些词元，可能会混合不同的有效模式，从而导致所谓的多峰性问题⁴。

扩散建模作为非自回归生成

尽管存在上述限制，非自回归生成的并行位置更新仍与扩散模型通常同时处理全部数据维度的方式相适应。因此，自然语言处理中的许多扩散模型采用 NAR 式的生成范式。为了理解这种联系，将非自回归生成的概念推广到单步预测之外是有帮助的。让我们将生成过程重新想象为一个随时间 t 演化的动力系统。我们将 $\mathbf{s}(t)$ 定义为系统在时间 t 的中间状态。这个抽象状态 $\mathbf{s}(t)$ 作为一个统一的表示：它既可以是一系列离散变量（例如，被破坏的标记），也可以是一系列连续向量（例如，标记嵌入）。在生成过程中，系统从 $t = T$ 到 $t = 0$ 在时间上反向演化。该系统的状态定义如下

⁴此问题可能导致重复、遗漏或语法不连贯，例如当有效实体为“New York”或“London”时，模型却可能在位置 i 选择“New”，在位置 $i+1$ 选择“London”。

- 起始点 $\mathbf{s}(T)$ 。这表示从先验分布（例如高斯噪声向量或完全随机的标记）中抽取的样本。它不包含任何语义信息。
- 中间状态 $\mathbf{s}(t)$ （当 $0 < t < T$ 时）。这些状态表示不断演化的潜在变量。它们可视为目标数据的带噪近似。
- 终止点 $\mathbf{s}(0)$ 。状态 $\mathbf{s}(0)$ 表示完全去噪后的信号。最终，通过确定性或随机性映射（例如取整或 argmax 操作）将 $\mathbf{s}(0)$ 投影回有效的标记序列 $\mathbf{x} = \{x_1, \dots, x_n\}$ 。

在这个过程中，我们需要将条件 $\mathbf{c}$ 合并到每个状态 $\mathbf{s}(t)$ 中，以形成联合状态 $(\mathbf{s}(t), \mathbf{c})$ 。因此，演化可以描述为

$$(\mathbf{s}(T), \mathbf{c}) \rightarrow \dots \rightarrow (\mathbf{s}(t), \mathbf{c}) \rightarrow \dots \rightarrow (\mathbf{s}(0), \mathbf{c}) \rightarrow \mathbf{x}. \quad (4.115)$$

最终的映射 $(\mathbf{s}(0), \mathbf{c}) \rightarrow \mathbf{x}$ 通常通过使用一个单独的系统来执行（例如，将嵌入映射到标记）。步骤 $(\mathbf{s}(T), \mathbf{c}) \rightarrow \dots \rightarrow (\mathbf{s}(t), \mathbf{c}) \rightarrow \dots \rightarrow (\mathbf{s}(0), \mathbf{c})$ 可以被视为一个动力系统的演化，就像标准扩散模型中那样。这个过程如图 4.17 所示。

在本节的剩余部分，我们将使用扩散模型作为实现非自回归生成的工具。我们根据它们如何定义动力系统 $\mathbf{s}(t)$ 的状态空间对这些方法进行分类。我们首先介绍嵌入空间扩散，它将标记映射到连续向量以应用连续高斯扩散。然后我们讨论离散扩散，它直接在离散标记上定义破坏和去噪过程。

4.5.2 连续空间中的扩散：嵌入空间扩散

在本小节中，我们介绍嵌入空间扩散框架，该框架将连续扩散模型适配于词元序列生成。

通用框架

将扩散模型应用于文本的一种直观方法是，将离散词元映射到连续嵌入空间 $\mathbb{R}^d$ 中，从而使扩散和生成过程可以在连续域中进行。几项重要工作都基于这种嵌入空间扩散框架 [1, 2, 3]。在这些方法中，第一步是将序列中的每个词元转换为一个 d 维实值向量（称为词元嵌入）。形式上，令 V 为一个大小为 $|V|$ 的词表。每个词元 $x \in V$ 通常表示为一个独热向量，并投影为一个稠密向量。然后，一个词元序列 $\mathbf{x} = \{x_1, \dots, x_n\}$ 可以通过一个嵌入函数 $\text{Embed}(\cdot)$ 映射为一个连续向量序列，如下所示：

$$\begin{aligned} \text{Embed}(\mathbf{x}) &= [\mathbf{e}_1, \dots, \mathbf{e}_n], \\ &= \mathbf{e} \end{aligned} \quad (4.116)$$

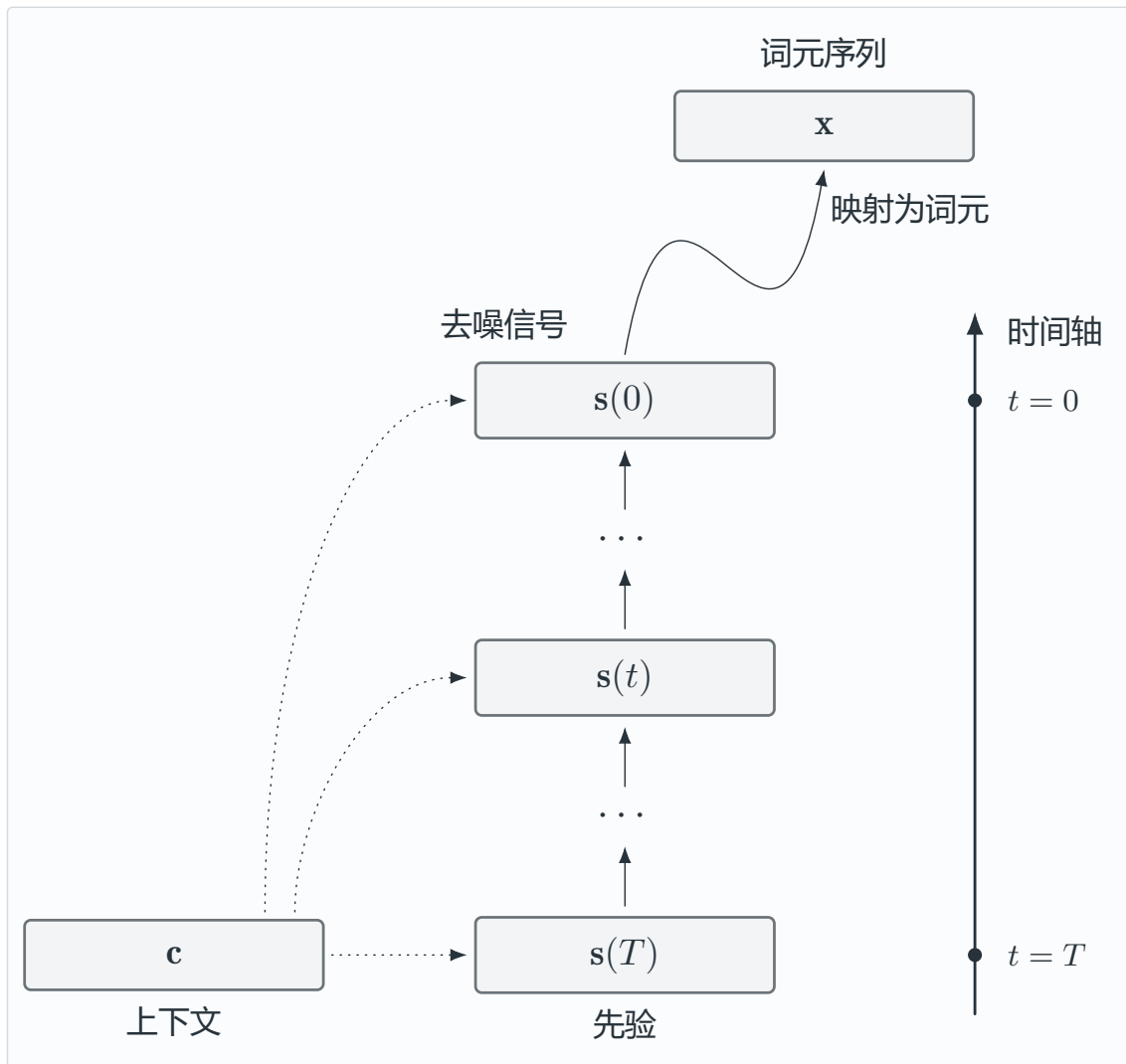


图 4.17: 基于扩散模型的非自回归生成示意图。生成过程被建模为状态 $s(t)$ 随时间 t 的演化过程。其中 $s(T)$ 表示起始点（先验分布）， $s(0)$ 表示终点（完全去噪信号）。上下文信息 c 在生成过程中被注入到状态变量。最终输出是从 $s(0)$ 映射得到的词元序列。

其中 $e_i \in \mathbb{R}^d$ 表示位置 i 处的嵌入， $e \in \mathbb{R}^{d \times n}$ 表示嵌入序列（堆叠为矩阵）。同样，我们可以将上下文词元序列 c 转换为嵌入序列 $e_c \in \mathbb{R}^{d \times n_c}$ 。

定义 $\text{Embed}(\cdot)$ 的一种简单方法是使用一个静态嵌入矩阵 $E \in \mathbb{R}^{d \times |V|}$ [??]。这种方法虽然简单，但将词元视为独立单元，从而忽略了它们在序列中的上下文依赖关系。或者，可以将整个词元序列映射到一个上下文化表示序列中。例如，可以使用预训练的大语言模型来编码词元及其周围的上下文 [??]。请注意，这种变换不限于词元与向量之间的一一对应关系，也不受限于固定的序列长度 n 。相反，我们可以使用更高层次的抽象将词元序列映射到一个不同长度 n' 的潜在连续序列（其中 n' 不必等于 n ）。例如，通过使用变分自编码器，可以将离散序列压缩为更稠密、语义更丰富的潜在表示。在这种潜在扩散范式 [?] 中，扩散过程作用于这些压缩后的潜在变量，而不是原始的词元嵌入。

一旦词元被表示为嵌入，生成问题就可以用标准方式通过扩散模型来形式化。形

式上, 令 $\mathbf{e}_c$ 为上下文嵌入序列。我们定义 $\bar{\mathbf{s}}(t)$ 为结合了 $\mathbf{e}_c$ 和 $\mathbf{s}(t)$ 的增广状态:

$$\bar{\mathbf{s}}(t) = [\mathbf{e}_c, \mathbf{s}(t)]. \quad (4.117)$$

我们将 $\mathbf{e}_c$ 和 $\mathbf{e}$ 的拼接视为连续扩散过程的初始状态 $\bar{\mathbf{s}}(0) = [\mathbf{e}_c, \mathbf{e}] \in \mathbb{R}^{d \times (n_c + n)}$ 。然后, 我们定义一个前向过程, 在时间 $t \in [0, T]$ 内逐渐向目标嵌入添加高斯噪声, 同时保持上下文嵌入固定。它将有意义的词元嵌入转换为标准高斯噪声 $\mathcal{N}(\mathbf{0}, \mathbf{I})$ 。

如前所述, 有几种模型可供选择。例如, 可以用随机微分方程描述目标嵌入的扩散过程, 而把上下文嵌入作为固定条件:

$$\begin{aligned} d\mathbf{e}_c &= 0, \\ d\mathbf{s}(t) &= f(\mathbf{s}(t), \mathbf{e}_c, t)dt + g(t)d\mathbf{w}. \end{aligned} \quad (4.118)$$

其中 $f(\mathbf{s}(t), \mathbf{e}_c, t)$ 是漂移系数, $g(t)$ 是扩散系数, $\mathbf{w}$ 是标准维纳过程。生成过程对应于条件反向时间随机微分方程, 它从目标序列部分为高斯噪声的状态 $\bar{\mathbf{s}}(T) = [\mathbf{e}_c, \mathbf{s}(T)]$ 重建目标嵌入:

$$d\mathbf{s}(t) = [f(\mathbf{s}(t), \mathbf{e}_c, t) - g(t)^2 \nabla_{\mathbf{s}(t)} \log p_t(\mathbf{s}(t) | \mathbf{e}_c)] dt + g(t)d\bar{\mathbf{w}}. \quad (4.119)$$

其中 $\bar{\mathbf{w}}$ 是反向时间维纳过程, $\nabla_{\mathbf{s}(t)} \log p_t(\mathbf{s}(t) | \mathbf{e}_c)$ 是给定上下文嵌入后的条件得分函数。也可以使用流匹配来学习以 $\mathbf{e}_c$ 为条件的向量场, 将目标部分的分布从高斯噪声推送到数据分布。这些方法的详细讨论在此省略, 相关的连续扩散模型已在前面的随机概率动力学一节中讨论。

与计算机视觉模型的一个关键区别在于, 在自然语言处理中, 我们通常使用 Transformer 作为学习得分函数或向量场的主干架构。给定增广状态 $\bar{\mathbf{s}}(t) = [\mathbf{e}_c, \mathbf{s}(t)] \in \mathbb{R}^{d \times (n_c + n)}$, Transformer 可以读取上下文块和目标块, 但只有与目标序列对应的输出块用于估计条件得分或向量场, 上下文块本身不参与扩散更新, 如图 4.18 所示。与图像生成 (其空间维度通常是固定的) 不同, 自然语言序列的长度本质上是可变的。因此, 我们需要在推理过程中确定目标序列长度 n 。此外, 尽管扩散和生成过程在连续空间中进行以获得 $\mathbf{s}(0)$, 但目标是产生离散词元。因此, 需要一个额外的模块将生成的连续嵌入解码回离散词元。我们将在下面简要讨论这两个挑战。

长度预测

长度预测是自然语言处理中一个反复出现的问题, 已有悠久的历史。一种直接的方法是根据上下文明确预测目标序列的长度。例如, 可以通过将源长度乘以特定比率来简单地推导目标长度。这一概念可以追溯到统计机器翻译 (SMT) 的早期, 当时使用繁衍模型来确定与单个源词对应的目标词数量 [?]。在非自回归生成模型的发展过程中也采用了类似的机制 [?]。在早期的非自回归生成模型 (通常是编码器-解码

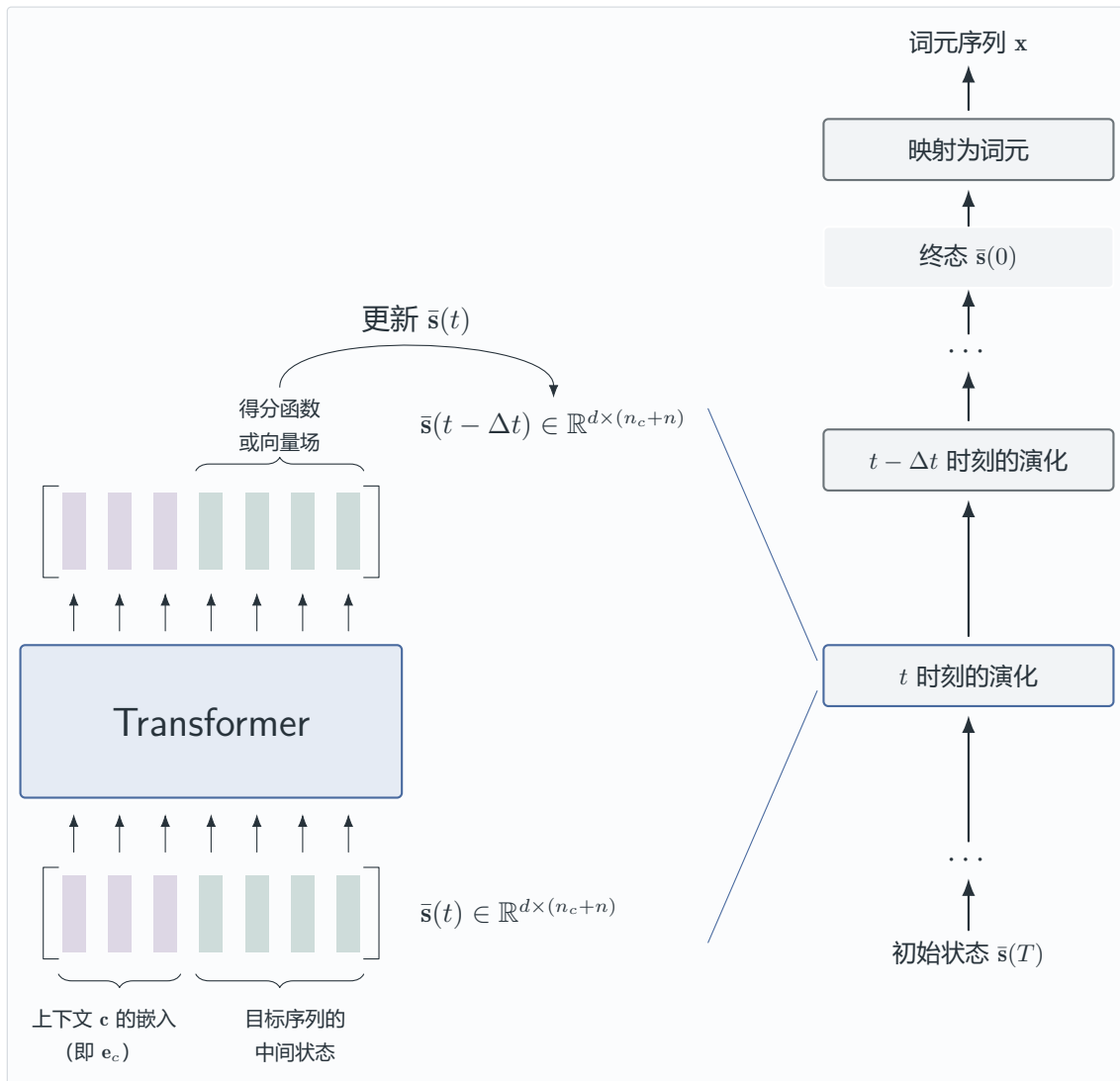


图 4.18: 基于 Transformer 的生成过程示意图。生成（即反向扩散）过程从目标序列的标准高斯噪声初始化状态 $\bar{s}(T)$ 开始。在每个时间步 t ，状态 $\bar{s}(t)$ 表示上下文嵌入与含噪目标嵌入的拼接。Transformer 模型基于 $\bar{s}(t)$ 估计得分函数（或向量场），随后用于将状态更新至 $\bar{s}(t - \Delta t)$ 。该迭代过程持续进行直至达到干净状态 $\bar{s}(0)$ ，最终被解码为离散标记。注意上下文嵌入 e_c 在更新过程中保持固定。

器架构）中，编码器中集成了一个繁衍预测器。该预测器估计每个编码器隐藏状态应被复制到解码器的次数，从而在解码之前规划目标序列长度。在这类方法中，长度预测被构建为对概率 $\Pr(n|\mathbf{c})$ 进行建模。预测器通常是一个神经网络，例如多层感知机（MLP），其训练目标是最大化给定上下文条件下真实长度的似然。在推理过程中，通过选择概率最高的长度 n^* 来确定目标长度。

显式长度预测方法简单，但常常导致重复和幻觉，因为模型会试图用无意义或冗余的令牌来填充预分配窗口中未使用的容量。相反，当前大多数扩散语言模型采用隐式长度确定方法。这种方法与自回归生成的行为一致，允许模型自行发出终止信号。在实践中，生成过程从一个足够长的窗口（例如，最大序列长度）开始。一旦该过程完成，并且连续嵌入已被解码为离散令牌，系统会识别第一个终止标记（如 EOS 令

牌) 的出现位置。该位置被视为句子的逻辑结尾, 其后的所有令牌都将被丢弃。

最近的研究也通过引入动态长度适应机制, 解决了预分配最大长度窗口带来的计算效率低下的问题 [??]。这些方法不依赖于仅通过 EOS 令牌进行事后截断, 而是将长度调制直接集成到扩散和生成过程中。通过离散的插入和删除操作或灵活的去噪调度, 它们可以动态调整序列长度。

舍入

如上所述, 生成过程在 $t = 0$ 时终止, 产生一个连续向量序列 $\mathbf{s}(0)$ 。我们将 $\mathbf{s}(0)$ 内生成的嵌入序列表示为 $\hat{\mathbf{e}} = [\hat{\mathbf{e}}_1, \dots, \hat{\mathbf{e}}_n]$, 其中每个 $\hat{\mathbf{e}}_i \in \mathbb{R}^d$ 。由于我们的目标是生成文本, 必须将这些连续向量映射回词汇表 V 中的离散词元。此操作通常被称为**舍入或解码**。

最直接的舍入方法是**最近邻搜索**。假设我们使用预训练的嵌入矩阵 $\mathbf{E} \in \mathbb{R}^{d \times |V|}$ 来表示词元嵌入, 其中每一列对应一个词元 $v \in V$ 的嵌入。那么, 我们可以选择在特定距离度量下, 其嵌入与生成的向量 $\hat{\mathbf{e}}_i$ 最接近的词元。由于扩散模型通常使用 MSE 损失进行训练, 欧几里得距离是一个自然的选择

$$x_i = \arg \min_{v \in V} \|\hat{\mathbf{e}}_i - \mathbf{e}_v\|_2^2, \quad (4.120)$$

其中 $\mathbf{e}_v$ 表示 $\mathbf{E}$ 中对应词元 v 的列。或者, 也可以采用可训练的投影层后接 softmax 函数, 类似于标准自回归语言模型中的输出层。在这种情况下, 词元 x_i 的概率由 Softmax 函数给出。

相比之下, 对于基于潜在或上下文嵌入的模型 (例如使用 VAE 或 LLM 表示的模型), 简单的最近邻搜索不适用, 因为生成的潜在变量并不直接对应静态的词元嵌入。在这些方法中, 需要一个可学习的解码器将连续的潜在序列映射回离散的词元空间。解码过程通常涉及一个神经网络, 该网络为每个潜在向量预测词汇表上的概率分布, 如同自回归生成模型一样。

舍入并非易事, 这源于连续扩散潜在空间与词元嵌入的离散性质之间的不匹配。标准的扩散训练目标 (如 MSE) 鼓励模型预测数据的期望值。在语言的上下文中, 如果分布是多模态的 (例如, 下一个词元可能是“猫”或“狗”), 模型可能会预测它们嵌入的平均值: $(\mathbf{e}_{\text{猫}} + \mathbf{e}_{\text{狗}})/2$ 。这个平均向量位于有效嵌入之间的间隙空间中。简单地将此向量舍入到最近邻, 可能会导致一个语义宽泛的词元 (例如“动物”), 或者如果嵌入空间不是完全平滑的, 则可能导致一个完全不相关的词元。这种现象通常表现为生成的向量未能承诺于一个特定的离散模式。

为了缓解这个问题, 研究人员引入了辅助目标和架构修改。[?] 提出了一个可训练的舍入参数, 并向训练目标中添加了一个辅助正则化项。该项明确鼓励模型生成接近 $\mathbf{E}$ 中有效嵌入的向量, 从而减少推理过程中的舍入误差。[?] 引入了自条件的概念, 即将去噪词元的估计值投影回嵌入空间, 并用作下一步的输入。这有效地将中间状态

钳制在更接近有效数据流形的位置，并使最终状态 $\mathbf{s}(0)$ 能够可靠地映射到离散词元。

4.5.3 词元空间中的扩散：离散扩散

虽然嵌入空间中的扩散能很好地融入标准扩散模型的框架，但它引入了将连续向量舍入回离散词元的非平凡挑战。为了规避这一问题，并使生成过程更自然地与语言的离散特性对齐，越来越多的研究开始探索离散扩散。在这一范式中，污染过程直接在离散词汇空间 V 上进行操作，从而消除了对嵌入投影和舍入的需求。

CTMC 视角

为了直接在离散词汇表 V 上建模扩散和生成过程，我们摒弃了连续空间中常用的高斯噪声和随机微分方程（SDE）。在离散状态空间中，过程在连续时间内演化，但仅通过瞬时跳跃改变状态：它保持恒定一段随机停留时间，然后跳转到一个新状态。自然描述这种概率跳跃的数学框架是连续时间马尔可夫链（CTMC）。正如 SDE 框架将扩散视为步长趋近于零时随机游走的极限，CTMC 框架将离散扩散视为时间步数趋近于无穷时离散时间马尔可夫链的极限。

由于此处的 CTMC 在离散的 token 上运行，我们使用 $\mathbf{x}(t)$ 而不是 $\mathbf{s}(t)$ 来表示代表 token 序列的状态。令 $\{\mathbf{x}(t)\}_{t \geq 0}$ 为一个随机过程，其中 $\mathbf{x}(t)$ 是时间 t 时由 n 个 token 组成的序列。令 $\mathbf{x}$ 和 $\mathbf{y}$ 表示离散空间 V^n 中的特定状态（即 token 序列）。概率分布 $p_t(\mathbf{x}) = \Pr(\mathbf{x}(t) = \mathbf{x})$ 的演化由 Kolmogorov 前向方程支配

$$\frac{dp_t(\mathbf{x})}{dt} = \sum_{\mathbf{y} \in V^n} p_t(\mathbf{y}) [\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{y}\mathbf{x}}, \quad (4.121)$$

其中 $\mathbf{Q}^{\text{seq}}(t)$ 是整个序列的（时间相关的）生成器矩阵。项 $[\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{y}\mathbf{x}}$ 表示从状态 $\mathbf{y}$ 到状态 $\mathbf{x}$ 的瞬时转移率，它是大矩阵 $\mathbf{Q}^{\text{seq}}(t)$ 中的一个元素。

然而，状态空间大小 $|V|^n$ 是指数级大的，使得完整的矩阵 $\mathbf{Q}^{\text{seq}}(t)$ 在计算上不可行。为了解决这个问题，我们通常在前向过程中假设 token 位置之间相互独立。在此假设下，转移在每个位置上独立发生。因此，我们可以使用一个更小的单 token 生成器矩阵 $\mathbf{Q}^{\text{tok}}(t) \in \mathbb{R}^{|V| \times |V|}$ 来建模该过程，该矩阵作用于单个 token。因此， $\mathbf{Q}^{\text{seq}}(t)$ 的复杂高维动力学可以分解为 n 个并行的过程，每个过程都由相同的单 token 生成器 $\mathbf{Q}^{\text{tok}}(t)$ 支配。图 4.19 展示了方程 (4.121) 的示意图。

需要注意的是，虽然 CTMC 通过方程 (4.121) 中的前向方程提供了连续时间的视角，但 NLP 中实用的离散扩散模型通常运行在离散的时间网格 $t \in \{0, 1, \dots, T\}$ 上。在这种情况下，过程通过相邻时间步之间的转移核（或矩阵）来实现，而 CTMC 表达式可以理解为这些离散化动力学的连续时间极限。

使用 CTMC，我们将前向和后向过程描述如下：

- **前向过程。**在连续时间马尔可夫链（CTMCs）中，我们并非通过添加高斯噪声，

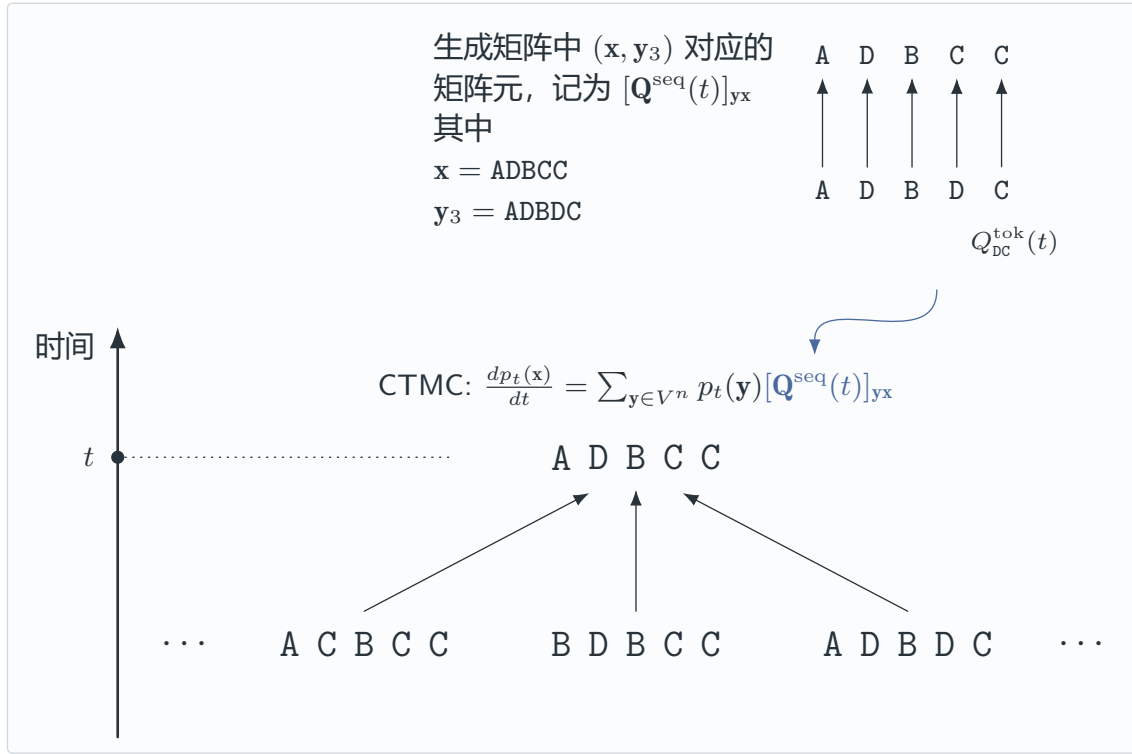


图 4.19: 连续时间马尔可夫链在标记序列生成中的示意图。该连续时间马尔可夫链在时间 t 的动力学方程为 $\frac{dp_t(\mathbf{x})}{dt} = \sum_{\mathbf{y} \in V^n} p_t(\mathbf{y}) [\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{y}\mathbf{x}}$ 。该方程表明, 观测到目标序列 $\mathbf{x}$ (例如 ADBCC) 的概率变化率, 是所有可能源序列 (例如 $\mathbf{y}_1$ 、 $\mathbf{y}_2$ 、 $\mathbf{y}_3$) 的概率之和, 并以序列级转移速率 $[\mathbf{Q}^{\text{seq}}(t)]_{\mathbf{y}\mathbf{x}}$ 加权。全局转移速率由底层的标记级动力学决定。例如, 从序列 $\mathbf{y}_3$ 到 $\mathbf{x}$ 的转移涉及单个标记的变化, 例如第四个标记从 D 变为 C 的速率 $Q_{\text{DC}}^{\text{tok}}(t)$ 。请注意, 虽然连续时间马尔可夫链在连续时间中建模动力学, 但状态转移本身是离散的跳跃, 而非连续变化。

而是通过随机标记替换来破坏文本。这一过程由预定义的生成器矩阵 $\mathbf{Q}^{\text{tok}}(t)$ 决定。常用的破坏方案包括:

- **均匀扩散:** 标记以均匀速率跳转到词汇表中的任意其他标记。这类似于连续空间中的高斯噪声, 后者将数据分布推向标准正态分布。在此情况下, 分布会收敛为词汇表 V 上的均匀分布。
- **掩码扩散 (吸收态):** 标记以特定速率转移至特殊的 [MASK] 标记且不可逆, 这对应于马尔可夫链中的吸收态。
- **反向过程。** 随机过程中的一个标准结果表明, 如果前向过程是一个具有生成器 $\mathbf{Q}^{\text{tok}}(t)$ 的 CTMC, 那么从 T 到 0 反向运行的后向过程也是一个 CTMC [?]。其生成速率矩阵 $\tilde{\mathbf{Q}}^{\text{tok}}(t)$ 由下式给出:

$$\tilde{Q}_{vu}^{\text{tok}}(t) = Q_{uv}^{\text{tok}}(t) \frac{p_t(u)}{p_t(v)}, \quad (4.122)$$

其中 $u, v \in V$ 代表词汇表中的特定 token 值, 且上述非对角速率在 $p_t(v) > 0$ 时

定义。这里， $\tilde{Q}_{vu}^{\text{tok}}(t)$ 表示反向过程中从值 v 转移到值 u 的速率， $p_t(v)$ 是 token 在时间 t 取值为 v 的边缘概率；反向生成器的对角元由每一行元素之和为零的条件确定。

上述方程建立了 CTMC 与离散扩散模型之间的基本联系。它是反向时间 SDE 公式的离散类比，该公式将反向漂移与得分函数 $\nabla \log p_t(\mathbf{x})$ 联系起来。在离散情况下，概率比 $p_t(u)/p_t(v)$ 充当“离散得分”，引导生成过程朝向高概率的 token。需要强调的是，方程 (4.122) 首先描述单词元的边缘 CTMC；当完整序列的各位置存在相关性时，精确反向过程应定义在序列状态空间 V^n 上，并使用联合序列概率之比。因此，仅依赖单词元边缘比的逐位置参数化一般并不足够。实践中，Transformer 通过读取完整的损坏序列来条件化各位置的跳转速率，从而表达跨位置依赖；在前向生成元只允许单位位置跳转的设定下，这种全序列条件化的速率参数化在结构上可以表示相应的精确反向生成元。

方程 (4.122) 是时间反转 CTMC 的理论刻画，其中反向速率依赖于真实的边缘分布 $p_t(\cdot)$ 。在用于文本的离散扩散模型中，我们通常不显式地估计 $p_t(\cdot)$ 。相反，反向动力学是通过反向后验 $\Pr(\mathbf{x}(t - \Delta t) | \mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$ 的贝叶斯分解来构建的。因此，方程 (4.122) 主要用于提供直觉，说明为什么概率比或后验重加权会引导反向过程朝向高概率的 token。在实践中，我们使用神经网络（通常是 Transformer）对反向动力学进行参数化，该网络预测一个去噪分布，例如 $p_\theta(\mathbf{x}(0) | \mathbf{x}(t), \mathbf{c})$ ⁵。

基于 CTMC 的扩散建模中的一个代表性例子是？] 提出的**结构化去噪扩散概率模型**（D3PM）。虽然早期尝试离散扩散通常依赖于简单的均匀转移概率，但 D3PM 通过结构化转移矩阵将该框架推广到支持任意噪声过程。重要的是，？] 证明了破坏过程的选择会显著影响样本质量。他们引入了几种超越标准均匀扩散的结构化转移方案。在模型参数化方面，D3PM 不同于噪声预测范式。由于噪声在离散空间中不是加性的，D3PM 被训练为直接从噪声输入中预测干净的数据分布 $p_\theta(\mathbf{x}(0) | \mathbf{x}(t), \mathbf{c})$ 。

需要注意的是，由于在训练期间可以获得真实值 $\mathbf{x}(0)$ ，我们可以直接计算真实的反向后验 $q(\mathbf{x}(t - \Delta t) | \mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$ 。因此，这个真实后验作为模型转移概率的监督信号。

形式上，这个反向后验对应于前一个时间步的分布。遵循扩散模型中的常见符号，我们用 q 表示前向条件分布。那么，反向后验可以使用贝叶斯规则推导：

$$q(\mathbf{x}(t - \Delta t) | \mathbf{x}(t), \mathbf{x}(0), \mathbf{c}) = \frac{q(\mathbf{x}(t) | \mathbf{x}(t - \Delta t), \mathbf{c}) q(\mathbf{x}(t - \Delta t) | \mathbf{x}(0), \mathbf{c})}{q(\mathbf{x}(t) | \mathbf{x}(0), \mathbf{c})}. \quad (4.123)$$

这里，序列上的概率分布分解为独立的 per-token 转移，例如：

$$q(\mathbf{x}(t) | \mathbf{x}(0), \mathbf{c}) = \prod_{i=1}^n q(x_i(t) | x_i(0), \mathbf{c}), \quad (4.124)$$

⁵在大多数设置中，前向破坏过程被选择为独立于 $\mathbf{c}$ 。我们在符号中保留 $\mathbf{c}$ 是为了匹配条件生成的设定，并允许更一般的情况，即噪声过程可能依赖于 $\mathbf{c}$ 。

其中每个概率 $q(x_i(t)|x_i(0), \mathbf{c})$ 由单 token 生成器 $\mathbf{Q}^{\text{tok}}(t)$ 决定⁶。在实践中，对于像均匀扩散和掩码扩散这样的结构化 $\mathbf{Q}^{\text{tok}}(t)$ 矩阵，前向转移概率 $q(x(t)|x(0), \mathbf{c})$ 具有简单的解析闭式解。对于均匀扩散，我们有：

$$q(x(t)|x(0), \mathbf{c}) = \alpha_t \cdot \mathbb{I}(x(t) = x(0)) + (1 - \alpha_t) \cdot \frac{1}{|V|}, \quad (4.125)$$

其中 α_t 是一个依赖于时间的标量调度参数，控制信号保留率。类似地，对于吸收态（掩码）扩散，概率质量被导向一个特殊的 [MASK] token。相应的闭式转移概率由下式给出：

$$q(x(t)|x(0), \mathbf{c}) = \begin{cases} \alpha_t, & \text{if } x(t) = x(0), \\ 1 - \alpha_t, & \text{if } x(t) = [\text{MASK}], \\ 0, & \text{otherwise.} \end{cases} \quad (4.126)$$

这些解析解使我们能够绕过前向过程中计算昂贵的矩阵运算。有了这些易于处理的前向概率，后向过程通过将神经网络对原始 token 序列的预测 $p_\theta(\mathbf{x}(0)|\mathbf{x}(t), \mathbf{c})$ 代入上述推导的贝叶斯分解来实现。这产生了一个反向转移分布 $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ ，我们可以从中采样。图 4.20 展示了在假设 $\Delta t = 1$ 的情况下，D3PM 中均匀扩散的前向和后向过程。

为了训练模型，我们希望最大化观测数据 $\log p_\theta(\mathbf{x}(0)|\mathbf{c})$ 的对数似然。然而，这需要所有可能的轨迹 $\mathbf{x}(1:T)$ 进行边缘化，这是不可行的。通过引入前向过程 $q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})$ 作为辅助分布，我们可以将对数边缘概率重写为一个期望：

$$\begin{aligned} \log p_\theta(\mathbf{x}(0)|\mathbf{c}) &= \log \sum_{\mathbf{x}(1:T)} q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c}) \frac{p_\theta(\mathbf{x}(0:T)|\mathbf{c})}{q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})} \\ &= \log \mathbb{E}_{q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})} \left[\frac{p_\theta(\mathbf{x}(0:T)|\mathbf{c})}{q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})} \right]. \end{aligned} \quad (4.127)$$

应用 Jensen 不等式，我们将对数移到期望内部，推导出一个可处理的证据下界 (ELBO)：

$$\log p_\theta(\mathbf{x}(0)|\mathbf{c}) \geq \mathbb{E}_{q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})} \left[\log \frac{p_\theta(\mathbf{x}(0:T)|\mathbf{c})}{q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})} \right]. \quad (4.128)$$

通过将联合分布 $p_\theta(\mathbf{x}(0:T)|\mathbf{c})$ 和前向轨迹 $q(\mathbf{x}(1:T)|\mathbf{x}(0), \mathbf{c})$ 展开为各自条件分布的乘积，然后重新排列项，负 ELBO 可以分解为每步 KL 散度项的和（关于离散设

⁶准确地说， $q(x_i(t)|x_i(0), \mathbf{c})$ 对应于单 token 转移概率矩阵 $\mathbf{P}^{\text{tok}}(0, t) = \exp\left(\int_0^t \mathbf{Q}^{\text{tok}}(\tau) d\tau\right)$ 的 $(x_i(0), x_i(t))$ 元素（假设随时间可交换）。从物理上讲， $\mathbf{P}^{\text{tok}}(0, t)$ 表示在区间 $[0, t]$ 内 token 之间转移的总概率质量。由于马尔可夫性质，该矩阵也可以被视为连续离散时间步上转移矩阵的乘积：如果我们将区间离散化为 $0 = t_0 < t_1 < \dots < t_k = t$ ，那么 $\mathbf{P}^{\text{tok}}(0, t) = \mathbf{P}^{\text{tok}}(t_{k-1}, t_k) \times \dots \times \mathbf{P}^{\text{tok}}(t_0, t_1)$ 。

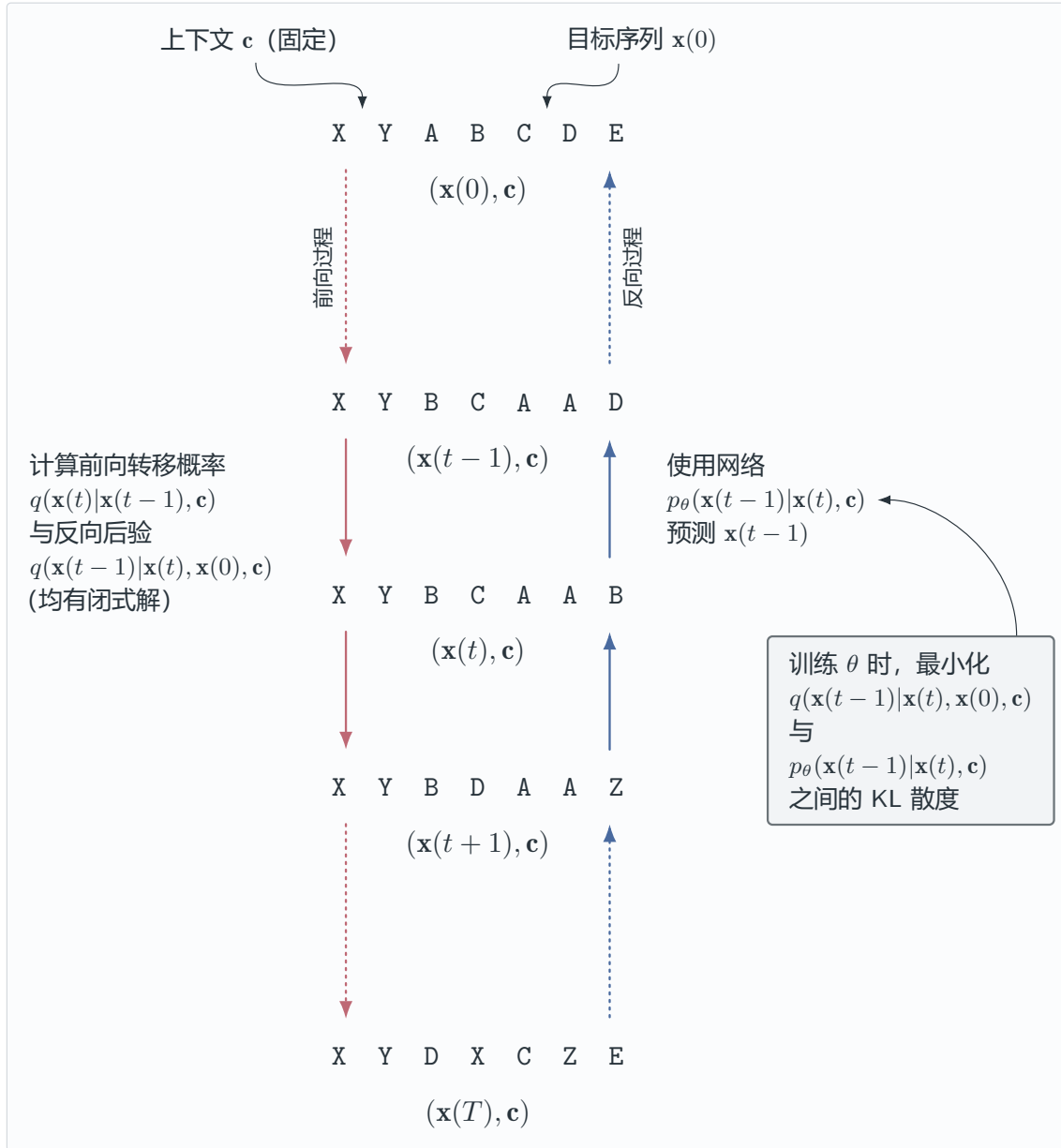


图 4.20: D3PM 均匀扩散中前向与后向过程的示意图。在前向过程中, 状态在每个 $\Delta t = 1$ 的时间步跳转到一个新状态, 初始状态为 $(\mathbf{x}(0), c)$, 最终状态为 $(\mathbf{x}(T), c)$ 。在前向步骤中, 我们计算前向转移概率 $q(\mathbf{x}(t)|\mathbf{x}(t-1), c)$, 然后可用其计算反向后验 $q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), c)$ 以训练后向模型。在后向过程中, 模型沿时间反向演化。在每个后向步骤中, 我们使用网络 $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), c)$ 预测词元序列 $\mathbf{x}(t-1)$, 然后继续处理 $t-1$ 。网络的训练目标是, 在 q 的期望下, 最小化 $q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), c)$ 与 $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), c)$ 之间的 KL 散度。

置下的完整推导, 请参见 [?]]。模型最小化的最终训练目标由下式给出:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(\mathbf{x}(1:T)|\mathbf{x}(0), c)} \left[-\log p(\mathbf{x}(T)) + \sum_{t=1}^T \text{KL} \left(q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), c) \parallel p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), c) \right) \right] \quad (4.129)$$

在这个目标中，项 $q(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{x}(0), \mathbf{c})$ 表示反向转移的真实后验。这里的一个关键优势是，对于结构化的前向过程（如均匀噪声或掩码），这个后验分布可以使用贝叶斯规则高效地以闭式计算。关于第一项，由于先验 $p(\mathbf{x}(T))$ 通常是一个没有可学习参数的固定分布（例如均匀分布或确定性的掩码状态）， $-\log p(\mathbf{x}(T))$ 相对于 θ 是常数，可以省略。因此，核心的学习信号驱使模型 p_θ 将其反向转移概率与前向动力学所决定的理想反向路径对齐。

掩码扩散模型

掩码扩散模型（Masked Diffusion Models, MDMs）是离散扩散的一种专门实现，其转移速率矩阵 $\mathbf{Q}^{\text{tok}}(t)$ 被设计为将概率质量移向一个特殊的吸收态，记作 [MASK] 标记 [? ?]。这种方法提供了一个独特的实践优势：一旦一个标记在前向过程中转移到掩码状态，它就会停留在那里，从而创建了一条通向完全损坏状态的单向路径。

前向过程逐步将原始序列中的标记替换为 [MASK]。它从一个完全观测到的序列开始，逐渐破坏信息，直到整个序列变为仅包含掩码的序列。在实践中，我们将时间离散化，并在离散时间步 $\{0, 1, \dots, T\}$ 上实现该过程为一个离散时间马尔可夫链。令 $\mathbf{x}(t)$ 为时间 $t \in \{0, 1, \dots, T\}$ 的损坏序列，其中 $\mathbf{x}(0)$ 是干净数据， $\mathbf{x}(T) = [\text{MASK}]^n$ ， $x_i(t)$ 是 $\mathbf{x}(t)$ 的第 i 个标记。在前向过程中，假设各位置之间独立，掩码扩散可以写为每个标记的吸收转移：

$$q(\mathbf{x}(t)|\mathbf{x}(0), \mathbf{c}) = \prod_{i=1}^n q(x_i(t)|x_i(0), \mathbf{c}), \quad (4.130)$$

其中单标记前向核为：

$$q(x_i(t)|x_i(0), \mathbf{c}) = \alpha_t \cdot \mathbb{I}(x_i(t) = x_i(0)) + (1 - \alpha_t) \cdot \mathbb{I}(x_i(t) = [\text{MASK}]). \quad (4.131)$$

这里 $\alpha_t \in [0, 1]$ 是一个单调递减的调度函数，控制着在时间 t 保留多少信号。等价地，我们可以将 $\mathbf{x}(t)$ 视为通过采样一个二元掩码指示向量 $\mathbf{m}_t \in \{0, 1\}^n$ 生成，其中：

$$m_{t,i} \sim \text{Bernoulli}(1 - \alpha_t), \quad (4.132)$$

$$x_i(t) = \begin{cases} [\text{MASK}], & m_{t,i} = 1, \\ x_i(0), & m_{t,i} = 0. \end{cases} \quad (4.133)$$

因此，扩散轨迹可以解释为重复增加掩码位置的集合，直到最终所有位置都被掩码，如图 4.21 所示。

后向过程从 $t = T$ 运行到 $t = 0$ 。给定上下文 $\mathbf{c}$ 和一个初始的完全掩码序列 $\mathbf{x}(T) = [\text{MASK}]^n$ ，MDM 迭代地将 [MASK] 标记替换为真实标记，直到在 V^n 中获得

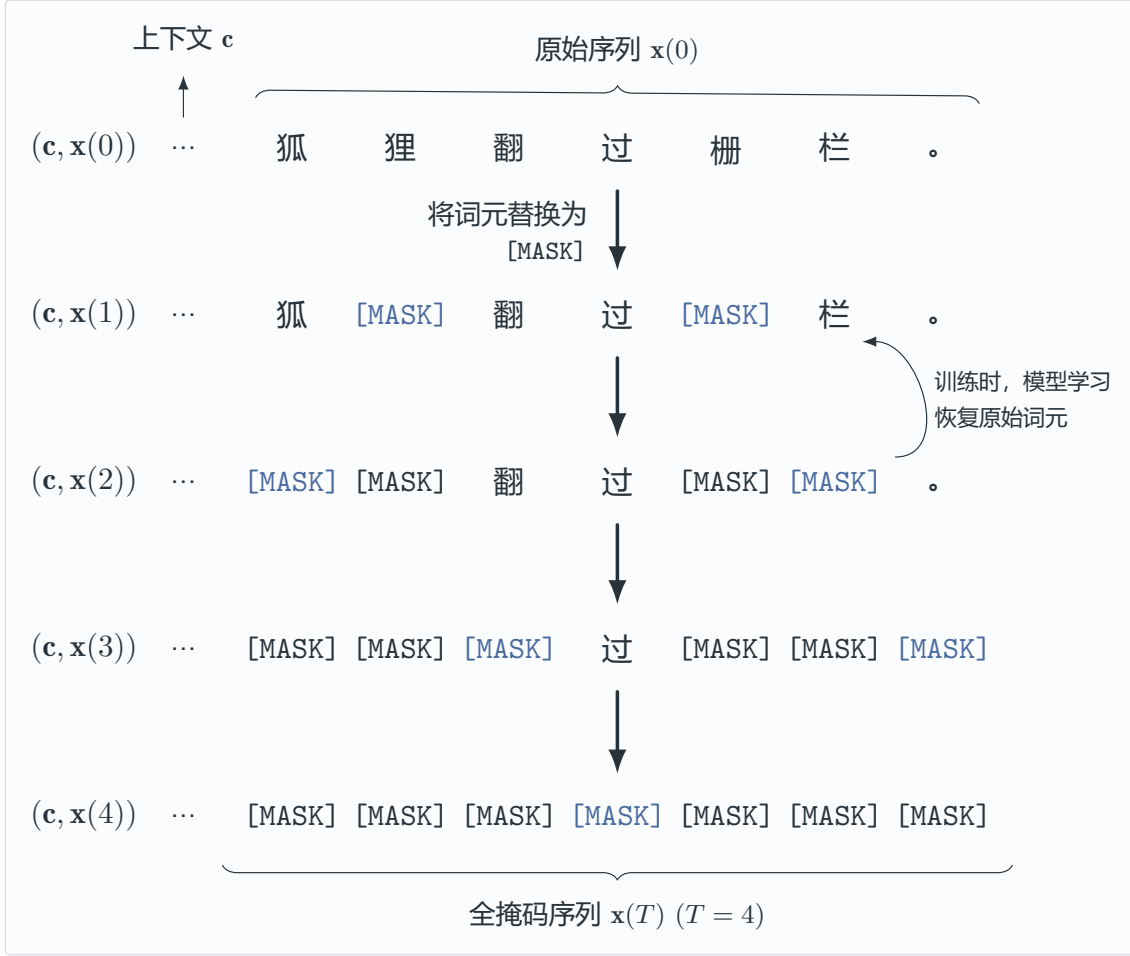


图 4.21: MDM 前向过程示意图, 其中原始标记序列被迭代地转换为完全掩码序列。在此过程中, 标记被替换为 [MASK] 符号, 直至整个序列被掩码。注意, 上下文 $\mathbf{c}$ 在整个过程中始终保持未掩码状态。为训练 MDM, 模型学习在固定上下文 $\mathbf{c}$ 和部分损坏序列的条件下, 恢复被掩码的标记。

一个完全实例化的序列。形式上, 我们定义一个参数化的逆向马尔可夫链:

$$p_{\theta}(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c}), \quad t = T, T-1, \dots, 1, \quad (4.134)$$

这里 $p_{\theta}(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ 是逆向条件分布 $\Pr_{\theta}(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ 的记法, 下标 θ 强调逆向时间动力学是由 θ 参数化的。

一种简单的参数化方法是让网络预测干净标记的去噪分布:

$$p_{\theta}(\mathbf{x}(0)|\mathbf{x}(t), \mathbf{c}) = \prod_{i=1}^n p_{\theta}(x_i(0)|\mathbf{x}(t), \mathbf{c}). \quad (4.135)$$

然后, 通过仅更新掩码位置来构建逆向转移, 同时保持未掩码的标记不变。为了与前

向调度 α_t 一致，令

$$\rho_t = \frac{\alpha_{t-1} - \alpha_t}{1 - \alpha_t}, \quad (4.136)$$

其中 ρ_t 表示在已知位置于时刻 t 被掩码的条件下，该位置在反向步骤 $t \rightarrow t-1$ 中恢复为词元的概率。对于 $v \in V \cup \{\text{[MASK]}\}$ ，每个位置的逆向核可以写为：

$$p_\theta(x_i(t-1) = v | \mathbf{x}(t), \mathbf{c}) = \begin{cases} \mathbb{I}(v = x_i(t)), & x_i(t) \neq \text{[MASK]}, \\ 1 - \rho_t, & x_i(t) = \text{[MASK]}, v = \text{[MASK]}, \\ \rho_t \pi_{\theta,t,i}(v | \mathbf{x}(t), \mathbf{c}), & x_i(t) = \text{[MASK]}, v \in V, \end{cases} \quad (4.137)$$

其中 $\pi_{\theta,t,i}(\cdot | \mathbf{x}(t), \mathbf{c})$ 是由模型产生的在 V 上的分类分布。一个常见的选择是：

$$\pi_{\theta,t,i}(v | \mathbf{x}(t), \mathbf{c}) = p_\theta(x_i(0) = v | \mathbf{x}(t), \mathbf{c}), \quad v \in V, \quad (4.138)$$

即，直接从预测的干净标记分布中采样缺失的标记。

给定一个示例数据集 $(\mathbf{x}(0), \mathbf{c}) \sim p_{\text{data}}$ ，MDMs 通过最大化干净序列的条件似然（或等价地最小化其负对数似然）来学习逆向模型 p_θ 。与其他扩散模型类似，我们可以使用 ELBO 来训练 MDMs（另见第 4.5.3 节中的 D3PM）。

尽管上述训练目标看起来有些繁琐，但在实践中，一个更简单且广泛使用的方法是训练模型以恢复掩码位置处的原始标记。我们采样一个时间步 $t \sim U\{1, \dots, T\}$ ，然后采样一个损坏序列 $\mathbf{x}(t) \sim q(\mathbf{x}(t) | \mathbf{x}(0), \mathbf{c})$ 。我们定义掩码索引集为：

$$\mathcal{M}(t) = \{i \in \{1, \dots, n\} : x_i(t) = \text{[MASK]}\}. \quad (4.139)$$

损失函数则定义为掩码位置上的条件负对数似然，如下所示：

$$\begin{aligned} \mathcal{L}_{\text{denoise}}(\theta) &= \mathbb{E}_{(\mathbf{x}(0), \mathbf{c}) \sim p_{\text{data}}} \mathbb{E}_{t \sim U[T]} \mathbb{E}_{\mathbf{x}(t) \sim q(\mathbf{x}(t) | \mathbf{x}(0), \mathbf{c})} \left[- \sum_{i \in \mathcal{M}(t)} \log p_\theta(x_i(0) | \mathbf{x}(t), \mathbf{c}) \right]. \end{aligned} \quad (4.140)$$

可选地，可以对时间步进行重新加权以强调某些噪声水平：

$$\mathcal{L}_{\text{denoise}}(\theta) = \mathbb{E} \left[\gamma_t \cdot \left(- \sum_{i \in \mathcal{M}(t)} \log p_\theta(x_i(0) | \mathbf{x}(t), \mathbf{c}) \right) \right], \quad (4.141)$$

其中 $\gamma_t \geq 0$ 是一个预定义的调度函数。直观上，类似于课程学习，小的 t 对应轻度损坏（即较容易的去噪），而大的 t 对应重度损坏（即较难的去噪）。权重 γ_t 可以用来平衡这些状态。

训练完成后，我们使用学习到的逆向核 $p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c})$ 通过从完全掩码状态向后运行马尔可夫链来生成文本。推理过程从以下开始：

$$\mathbf{x}(T) = [\text{MASK}]^n, \quad (4.142)$$

并迭代地采样：

$$\mathbf{x}(t-1) \sim p_\theta(\mathbf{x}(t-1)|\mathbf{x}(t), \mathbf{c}), \quad t = T, T-1, \dots, 1. \quad (4.143)$$

在方程 (4.137) 的每标记因式分解下，逆向步骤仅可能更新掩码位置。对于每个满足 $x_i(t) = [\text{MASK}]$ 的位置，先采样

$$b_{t,i} \sim \text{Bernoulli}(\rho_t), \quad (4.144)$$

再令

$$x_i(t-1) = \begin{cases} x_i(t), & x_i(t) \neq [\text{MASK}], \\ [\text{MASK}], & x_i(t) = [\text{MASK}], b_{t,i} = 0, \\ \text{Cat}(\pi_{\theta,t,i}(\cdot|\mathbf{x}(t), \mathbf{c})), & x_i(t) = [\text{MASK}], b_{t,i} = 1. \end{cases} \quad (4.145)$$

其中 $\text{Cat}(\cdot)$ 表示从其参数指定的分类分布中采样一个离散标记。于是，并非所有掩码位置都在同一步被填充；模型在已填充上下文的条件下逐步恢复缺失标记，直到所有位置都变为未掩码。

除了按 ρ_t 独立采样各位置，也可以在实践中采用显式的去掩码数量调度。令 k_t 为从 t 到 $t-1$ 时要新变为未掩码的标记数量，并令 $\mathcal{U}(t) \subseteq \mathcal{M}(t)$ 为要更新的掩码位置的选定子集，且 $|\mathcal{U}(t)| = k_t$ 。那么一个常见的实现是：

$$x_i(t-1) = \begin{cases} x_i(t), & i \notin \mathcal{U}(t), \\ \text{Cat}(\pi_{\theta,t,i}(\cdot|\mathbf{x}(t), \mathbf{c})), & i \in \mathcal{U}(t). \end{cases} \quad (4.146)$$

调度函数 $\{k_t\}_{t=1}^T$ 在速度与准确性之间进行权衡：较大的 k_t 导致更少的迭代次数，但要求模型在高不确定性下填充许多标记；而较小的 k_t 产生更多的细化步骤，但增加了计算量。

一个简单的选择是确定性调度，即每步揭示固定的比例（例如，在 t 上呈线性或余弦变化），使得掩码标记的期望数量从 n 减少到 0。另一种常见策略是基于模型置信度选择 $\mathcal{U}(t)$ 。给定 $\pi_{\theta,t,i}$ ，我们可以定义一个置信度分数为：

$$s_{t,i} = \max_{v \in V} \pi_{\theta,t,i}(v|\mathbf{x}(t), \mathbf{c}), \quad (4.147)$$

然后我们可以首先对置信度最高的位置进行去掩码。这种“先易后难”的启发式方法倾向于通过用可靠的标记锚定序列来稳定生成，然后再填充更模糊的标记。

4.5.4 离散扩散的评注

在本小节中，我们讨论在词元空间进行离散扩散时的若干建模考量。

对非吸收式替换扩散的再思考

吸收式的 [MASK] 构造具有一种吸引人的信息单调性：前向过程仅移除信息，而反向过程则仅将其填充回来。这种设计与 Transformer 的去噪过程非常契合，并且最近在大规模扩散语言模型的预训练中展现了强大的可扩展性 [??]。尽管如此，重新审视非吸收式替换扩散方法（例如均匀扩散）仍然是有价值的，在这些方法中，词元可能在整条轨迹中被反复替换。

首先，均匀替换是一个简单且行为良好的基线。D3PM 中使用的均匀替换核就是一个典型例子。在每一步中，一个词元要么被保留，要么被一个从词汇表中均匀采样的元素所替换。这种策略能得到闭式转移概率，从而保持训练的可处理性。此外，由于这种破坏在原则上是可逆的，反向过程不受不可逆吸收汇的约束。从概念上讲，均匀替换是添加各向同性高斯噪声的离散类比。它驱使分布趋向于一个简单的先验，同时保持对前向边缘分布的分析控制。

其次，在非吸收式替换扩散中，反向更新是密集的。在掩码扩散中，未掩码的词元通常被保留，只有被掩码的位置会被更新。而在非吸收式替换扩散中，每个位置都保持随机性，因此一个自然的反向核可以在每一步更新所有词元。这类似于连续时间扩散模型的结构，其中所有维度在整个轨迹中都被精炼。动态过程可以修正任何组成部分以提高全局一致性，而不是将可见坐标视为固定。

第三，非吸收式替换在语言中可能较为困难。如果词元可以被多次破坏，那么噪声序列 $\mathbf{x}(t)$ 可能包含局部流畅但全局不一致的片段，而反向模型必须同时推断缺失的语义并纠正错误的词元。这与掩码扩散形成对比，在掩码扩散中生成的词元通常是干净的。从连续时间马尔可夫链的视角看，非吸收式替换对应于一个生成器 $\mathbf{Q}^{\text{tok}}(t)$ ，其所有词元对之间具有非零的非对角速率，因此概率质量在 V 上持续循环，而不是流入单一的汇。随着 t 的增加，信噪比可能比吸收式扩散下降得更快，因为即使是被观测到的词元也是不可靠的。

最近的研究探索了将吸收式掩码与额外的替换噪声相结合的方法，以诱导一种纠正行为。通过这种方式，模型可以检测并修复损坏的词元，而不仅仅是填充被掩码的位置 [??]。这种混合视角表明，非吸收式噪声不仅仅是掩码扩散的竞争者，还可以作为一种互补机制来增强鲁棒性和自校正能力。

一个简单的想法是考虑结构化的替换策略。在自然语言处理中，一种自然的方法

是使用依赖于词元或结构感知的转移速率来偏置替换，例如：

$$Q_{uv}^{\text{tok}}(t) = w_{uv}(t), \quad (4.148)$$

其中 $w_{uv}(t)$ 建模了混淆性，它可以是频率感知的、与子词相关的或基于嵌入相似性的。其动机是使早期噪声在语义上是局部的（易于去噪），而使晚期噪声是全局混合的（接近一个简单的先验）。这类似于连续扩散中的方差保持与方差爆炸设计选择，但现在表示为离散转移结构上的调度。

另一个有趣的方向是在掩码式吸收扩散与均匀替换扩散之间进行插值。例如，可以在生成器层面考虑一种组合：

$$\mathbf{Q}^{\text{tok}}(t) = \lambda_t \cdot \mathbf{Q}_{\text{mask}}^{\text{tok}}(t) + (1 - \lambda_t) \cdot \mathbf{Q}_{\text{unif}}^{\text{tok}}(t), \quad (4.149)$$

其中模型在某些噪声水平下表现得像掩码扩散，而在其他水平下则保留在未掩码词元之间分配概率质量的能力。最近的研究已经探索了这种组合，以同时获得两种机制的优势（来自吸收式破坏的稳定性和来自替换式破坏的灵活性）。

掩码扩散建模与掩码语言建模

在一般的“损坏-去噪”框架下，掩码扩散模型的训练目标与掩码语言建模（MLM）非常接近。例如，在类 BERT 模型中，我们采样一个掩码模式，并训练一个双向 Transformer，使其能够根据部分观察到的上下文来预测掩码位置处的原始词元 [?]。如果我们将式 (4.133) 解释为以掩码率 $(1 - \alpha_t)$ 采样一个掩码指示符 $\mathbf{m}_t$ ，那么式 (4.140) 中的去噪损失本质上就是一个 MLM 交叉熵损失，区别仅在于损坏程度通过 α_t 被时间明确索引，并且网络以扩散步数 t 为条件。因此，主要的概念差异不在于监督信号，而在于去噪器的部署方式。BERT 的 MLM 主要是一种用于表示学习的预训练目标，而掩码扩散模型则将相同的去噪基本操作解释为一个逆时间转移核，并从一个完全掩码的状态开始迭代应用它，以生成一个自然的词元序列。从这个意义上说，现代的掩码扩散模型在架构上通常看起来像 BERT，但它们将 MLM 从一个预训练任务转变为一个序列生成的过程。

掩码扩散模型与 MLM 之间的联系揭示了一个有趣的方向：由于掩码扩散中的前向过程是一种特定的损坏选择，许多为 NLP 预训练开发的加噪策略可以被重新解释为离散扩散的替代前向核。也就是说，除了独立的词元掩码之外，我们还可以改变被损坏的对象（词元与文本片段）、损坏的方式（掩码、替换或置换）以及损坏程度随时间演变的方式（手动设计或学习得到的调度方案）。以下是在设计掩码扩散模型时可以考虑的一些常见加噪策略。

- **非均匀分布的标记替换。**与均匀替换标记不同，可以从频率调整的分布中采样替换项，或从学习到的提议分布中采样。例如 ELECTRA 模型，其中一个小型生成器提出合理的替换项，判别器则被训练来检测它们 [?]。从扩散的角度看，

这种更困难的非均匀替换可视为注入了比随机标记更具语义混淆性的噪声，这可能会鼓励更强的条件去噪器，并缩小训练时使用的损坏与推理时的不确定性之间的差距。

- **片段损坏。** T5 系列模型将连续的片段替换为单个哨兵标记，并学习重建缺失的内容 [?]。对于扩散式生成，片段损坏是自然的：逆向过程可被解释为逐步将粗糙的缺失区域细化为详细的文本。这与在结构化槽位上迭代去噪而非在独立标记空白处去噪的直觉相符。然而，这种方法的一个挑战在于，将多个标记替换为单个哨兵标记会改变序列长度，这违反了标准扩散模型的固定维度假设。为解决此问题，我们可以采用类似于 SpanBERT 的就地片段掩码 [?]，即连续标记被掩码但序列长度保持不变。或者，可以扩展前向核以显式建模插入和删除操作，从而处理可变长度的轨迹 [??]。
- **标记重排与置换噪声。** BART 引入了带有标记删除和句子置换等损坏的去噪预训练。该方法要求模型不仅学习内容恢复，还要学习重新排序 [?]。将其重新定义为扩散前向核，则意味着存在一类更广泛的离散动态过程，它们同时扰动身份信息 and 位置信息。这种方法可以导致逆向过程执行联合的“编辑 + 重排序”细化，而非纯粹的填充。
- **对抗性加噪。** 另一个方向是利用模型引导的扰动或辅助网络生成更具挑战性的损坏。用于 NLP 的对抗性训练方法（例如，嵌入级扰动）表明，更强、结构化的噪声可以提高鲁棒性 [?]。虽然许多对抗性方法是在连续嵌入空间中定义的，但其核心思想适用于扩散建模。我们可以选择在每个噪声级别上对去噪器施加最大压力的损坏方式，这原则上可以使学习到的逆向动态在迭代采样下更加稳定。
- **学习到的加噪调度。** 与其手动固定调度 α_t ，不如尝试学习在每个步骤中应用多少以及何种类型的噪声，例如，通过优化目标组合或选择最能服务于下游用途的损坏机制 [?]。在扩散建模中，这促使学习时间非齐次的损坏策略，以塑造轨迹，从而获得更好的样本质量或更少的生成步骤。

单纯形与 Logit 动态作为连续松弛方法

我们已经看到，连续时间马尔可夫链 (CTMC) 的表述将离散扩散过程建模为分布动态。尽管样本路径 $\mathbf{x}(t) \in V^n$ 在分类状态间跳跃，但通过式 (4.121) 中的 Kolmogorov 方程，边缘分布 $p_t(\cdot)$ 会随时间 t 平滑演化。这一观察启发了我们对语言建模的有用松弛方法：不再将中间状态视为硬标记序列，而是用单纯形上的概率列向量表示每个位置：

$$\mathbf{p}_i(t) \in \Delta^{|V|-1}, \quad (4.150)$$

$$\mathbf{p}_i(t)[v] = \Pr(x_i(t) = v \mid \mathbf{c}). \quad (4.151)$$

我们将这些向量聚合为矩阵 $\mathbf{P}(t) = [\mathbf{p}_1(t), \dots, \mathbf{p}_n(t)] \in \mathbb{R}^{|V| \times n}$ ，其中每列位于单纯形上。这将离散轨迹转换为单纯形积空间上的连续曲线，从而可以应用 ODE/SDE 工具的同时保持标记的分类特性。

对于生成器为 $\mathbf{Q}^{\text{tok}}(t) \in \mathbb{R}^{|V| \times |V|}$ 的标记级 CTMC，单个位置的边缘分布满足 Kolmogorov 正向方程：

$$\frac{d\mathbf{p}_i(t)}{dt} = [\mathbf{Q}^{\text{tok}}(t)]^\top \mathbf{p}_i(t), \quad (4.152)$$

这是单纯形上的一个 ODE。注意 $\mathbf{Q}^{\text{tok}}(t)$ 的行和为零，因此总概率守恒。在噪声过程中各位置独立的假设下，每个 $\mathbf{p}_i(t)$ 都遵循相同的 ODE 演化。这为语言建模提供了 ODE 视角：离散扩散可视为选择由 $\mathbf{Q}^{\text{tok}}(t)$ 诱导的单纯形上线性正向向量场，而非 $\mathbb{R}^d$ 中的高斯 SDE。

直接在概率空间操作可能在单纯形边界附近（例如 one-hot 顶点）存在数值问题。常用替代方案是通过 logit 向量 $\mathbf{z}_i(t) \in \mathbb{R}^{|V|}$ 参数化 $\mathbf{p}_i(t)$ ：

$$\mathbf{p}_i(t) = \text{Softmax}(\mathbf{z}_i(t)). \quad (4.153)$$

由此可在 logit 空间定义连续时间动态：

$$\frac{d\mathbf{z}_i(t)}{dt} = \mathbf{v}_\theta(\mathbf{z}_i(t), t, \mathbf{c}), \quad (4.154)$$

其中 $\mathbf{v}_\theta$ 是由 Transformer 定义的向量场（各位置共享）。直观上， $\mathbf{z}_i(t)$ 如同连续细化的“软标记”，终端解码 $\mathbf{z}_i(0) \mapsto x_i$ 可通过 $\text{Softmax}(\mathbf{z}_i(0))$ 的 $\arg \max$ 实现。当需要可微采样时，也可使用 Gumbel-Softmax 分布等分类采样的松弛方法 [??]。

这种 logit 视角阐明了为何连续松弛常与嵌入空间扩散相似，却更接近离散建模：状态保持词汇对齐（每位置维度 $|V|$ ）而非任意嵌入空间 $\mathbb{R}^d$ 。它还直接关联到式 (4.152) 的 CTMC 边缘 ODE：选择的 $\mathbf{Q}^{\text{tok}}(t)$ 决定了 $\mathbf{p}_i(t)$ 的特定演化，可通过 Softmax 映射重新表达为 $\mathbf{z}_i(t)$ 的演化。

如前面的随机概率动力学一节所述，在连续扩散中，逆向时间动态既可表示为涉及分数 $\nabla \log p_t(\mathbf{s})$ 的 SDE，也可表示为概率流 ODE。在离散扩散中，精确的逆向 CTMC 速率取决于概率比 $p_t(u)/p_t(v)$ （式 (4.122)），其作用类似于离散分数。当我们状态提升至 $\mathbf{p}(t)$ 或 $\mathbf{z}(t)$ 时，可类似地将逆向过程解释为学习从简单先验（如全掩码或均匀分布）流向集中于表示真实文本的 one-hot 顶点附近的分布的向量场。最近的离散分数建模研究表明，可以定义避免显式计算 $p_t(\cdot)$ 但仍产生理论合理的逆向动态的可处理训练目标 [??]。

总结而言，我们概述单纯形与 logit 松弛的实践优势如下：

- **更高效的 ODE 求解器与更少的步数。**一旦将逆向动力学表达为如式 (4.154) 所示的 ODE，我们就可以使用自适应 ODE 求解器或粗粒度离散化来减少采样

步数，这与连续扩散和流模型中使用的加速策略相呼应（参见前面的数值采样与高效生成小节）。

- **与流匹配的兼容性。**在单纯形上，流匹配自然地对应于学习将软标记分布向尖锐的、类似数据的分类分布传输的 $\mathbf{v}_\theta$ ，而无需在训练时依赖显式的基于似然的 ELBO。
- **离散与连续之间的谱系。**掩码扩散（硬 [MASK] 状态）与嵌入扩散（连续嵌入）可被视为两个端点。单纯形/逻辑动态占据了一个中间地带。状态是连续的，但仍可直接解释为标记概率，并且到文本的最终投影是定义良好的。

与迭代优化的关联

离散扩散的逆向过程与一系列关于非自回归生成中**迭代优化**的研究密切相关 [?]。在迭代优化中，模型并非一次性获得最终序列，而是通过多轮优化迭代，反复生成完整序列并逐步修正其部分内容。例如条件掩码语言模型中的掩码预测方法，该方法从全 [MASK] 模板出发，交替执行并行词元预测与不确定位置子集的重新掩码操作，直至收敛或达到固定迭代次数 [?]。

该方法可表述为与式 (4.146) 相似的形式。设 $\mathbf{x}^{(r)}$ 表示第 r 轮优化后的序列， $\pi_{\theta,i}^{(r)}(v|\mathbf{x}^{(r)}, \mathbf{c}) = p_\theta(x_i^{(r+1)} = v|\mathbf{x}^{(r)}, \mathbf{c})$ 为第 r 轮位置 i 的预测分布，则优化步骤可表示为：

$$x_i^{(r+1)} = \begin{cases} x_i^{(r)}, & i \notin \mathcal{U}^{(r)}, \\ \text{Cat} \left(\pi_{\theta,i}^{(r)}(\cdot|\mathbf{x}^{(r)}, \mathbf{c}) \right), & i \in \mathcal{U}^{(r)}, \end{cases} \quad (4.155)$$

其中 $\mathcal{U}^{(r)}$ 为第 r 轮待更新位置集合。典型情况下 $\mathcal{U}^{(r)}$ 由低置信度定义，而简单扩散实现中常采用时间调度策略。换言之，掩码扩散采样可视为一种结构化优化过程——其掩码模式随时间 t （即迭代优化中的迭代次数）单调演化，而经典优化方法允许基于模型不确定性的非单调决策（如对已填充词元重新掩码）。这一差异至关重要：单调解掩过程稳定但可能固化早期错误，而非单调优化虽可修正错误却可能因选择规则不明确导致振荡。

近期扩散语言模型已开始显式融合迭代优化中标准的非单调修正行为。一种方法是在推理阶段重新掩码低置信度词元，将扩散采样与类掩码预测的复查机制结合 [??]；另一种方法是通过修改前向破坏过程，使逆向模型不仅学习填充空白还需修正可见错误词元。这可通过混合吸收掩码噪声与替换噪声（如均匀替换）实现，从而产生天然支持修正与编辑的逆向过程 [?]。这些方法与第 4.5.4 节所述方案存在交集，也呼应了我们关于组合不同扩散策略的讨论。

从扩散模型视角亦可解读迭代优化。式 (4.152)–(4.154) 中的单纯形与逻辑松弛为迭代优化提供了微分方程视角。若将每次优化步骤视为软词元状态的微小更新，则迭代优化即成为朝向尖锐近单点分布的连续时间传输的显式数值离散化。这与近期离散

流匹配 formulations 相连接——后者旨在学习可用较少优化步骤采样同时保持质量的连续时间流 [??]。由此观之，扩散建模、基于流的建模与经典迭代优化可视为同一主题的变体，其主要差异在于：中间状态表示方式（硬词元 vs. 软分布）、更新集 $\mathcal{U}$ 选择策略（调度 vs. 置信度）、以及动态过程框架（马尔可夫过程 vs. 通用学习优化算子）。

扩散变换器

离散扩散在自然语言处理领域变得具有竞争力的一个实际原因是其逆向动力学可以通过变换器进行参数化。该架构是当前最具可扩展性的骨干网络，在语言建模领域占据主导地位。术语扩散变换器（DiT）在视觉领域应用最为广泛，其中用变换器骨干网络替换 U-Net 能够产生强大的缩放行为 [?]。详细介绍变换器超出了本文的范围。我们建议感兴趣的读者参阅相关论文 [??]。

在自然语言处理中，变换器提供了一种强大的机制，可以将一个词元序列编码为等长的实值表示序列。根据这些表示的定义和解释方式，此类模型可用于学习从输入词元序列到任意目标的映射。在扩散建模中，变换器必须被条件化于扩散时间 t （或离散索引）。这通常通过将学习到的时间嵌入注入到每一层来实现，例如，通过加性偏置，或一个时间词元进行交叉注意力。网络的输出可以有多种解释方式，这取决于所选择的参数化方法。在连续空间扩散中，常见的选择包括预测得分 $\nabla_s \log p_t(\mathbf{s})$ 、加性噪声 ϵ 、去噪数据 $\mathbf{s}(0)$ ，或是线性组合上述变量的速度变量 [??]。这些参数化方法会导致不同的离散化方案和稳定性特性，但本质上都可以视为学习一个时间索引的向量场，该向量场将概率质量从一个简单的基础分布输送到数据分布。

基于流的视角

鉴于 Transformer 提供的建模灵活性，我们无需局限于使用显式的基于似然的 ELBO（见公式 (4.129)）来训练扩散模型。一个自然的替代方案是流匹配。在连续设定下，流匹配学习一个时间依赖的向量场，其常微分方程将一个简单的基础分布输送到数据分布。对于语言而言，关键困难在于状态定义在离散的乘积空间 V^n 上，因此流必须定义为要么是词元上概率质量的动态（离散状态视角），要么是定义在连续松弛（如概率单纯形或 logit 空间）上的常微分方程（连续状态视角）。最近的**离散流匹配**方法提供了这两种视角，并使它们在现代语言建模的规模下变得实用 [???]。

一个有用的起点是公式 (4.152) 中已经引入的单纯形表示。我们不将 $\mathbf{x}(t)$ 视为离散的词元序列，而是将中间状态视为每个位置的分类概率

$$\mathbf{P}(t) = [\mathbf{p}_1(t), \dots, \mathbf{p}_n(t)] \in \mathbb{R}^{|V| \times n}, \quad (4.156)$$

其中

$$\sum_{v \in V} \mathbf{p}_i(t)[v] = 1. \quad (4.157)$$

在这种表示下，流是定义在单纯形乘积空间上的一个常微分方程，

$$\frac{d\mathbf{p}_i(t)}{dt} = \mathbf{v}_\theta(\mathbf{p}_i(t), t, \mathbf{c}), \quad (4.158)$$

$$\sum_{v \in V} \mathbf{v}_\theta(\mathbf{p}_i(t), t, \mathbf{c})[v] = 0, \quad (4.159)$$

其中切向约束确保了概率守恒。采样因此成为一个常微分方程积分问题，这与视觉领域类似：给定来自基分布的 $\mathbf{p}(T)$ ，反向积分公式 (4.159) 以获得 $\mathbf{p}(0)$ ，并通过 $\arg \max$ 或分类采样解码出词元。

剩下的任务是指定 $\mathbf{v}_\theta$ 的训练目标。离散流匹配通过定义一族在源分布和数据分布之间插值的概率路径来实现这一点，然后使模型向量场与这些路径诱导的瞬时概率速度相匹配。更具体地说，对于每个训练样本 $\mathbf{x}(0)$ ，定义 V^n 上或每个词元边缘分布上的条件路径 $q_t(\cdot | \mathbf{x}(0), \mathbf{c})$ ，并抽取一个带噪样本 $\mathbf{x}(t) \sim q_t(\cdot | \mathbf{x}(0), \mathbf{c})$ 。流匹配的回归目标是沿着该路径的条件速度，

$$\mathbf{v}^*(\mathbf{x}(t), t, \mathbf{x}(0), \mathbf{c}) = \frac{d\mathbb{E}[\mathbf{1}_{\mathbf{x}(t)} | \mathbf{x}(0), \mathbf{c}]}{dt}, \quad (4.160)$$

其中 $\mathbf{1}_{\mathbf{x}(t)}$ 是 $\mathbf{x}(t)$ 的独热表示。最终的目标函数与连续条件流匹配类似，区别在于状态和速度受到单纯形几何以及端点分类性质的约束。

加噪核可以与之前讨论的扩散语言模型中使用的核非常相似。对于每个位置，我们可以定义一个在干净点质量和基分布 $\bar{p}_i(\cdot)$ 之间的时间依赖混合，

$$q_t(x_i | x_i(0), \mathbf{c}) = \alpha_t \cdot \mathbb{I}(x_i = x_i(0)) + (1 - \alpha_t) \cdot \bar{p}_i(x_i), \quad (4.161)$$

其中调度 α_t 从 1 递减到 0。这即使在样本路径是离散的情况下，也能在单纯形中诱导出一条平滑的概率路径。

对公式 (4.161) 关于时间求导，得到目标速度的解析形式

$$\mathbf{v}_i^*(x_i(t), t, x_i(0), \mathbf{c}) = \dot{\alpha}_t \cdot [\mathbf{1}_{x_i(0)} - \bar{p}_i], \quad (4.162)$$

其中 $\dot{\alpha}_t = d\alpha_t/dt$ 表示调度的时间导数。由于 α_t 随前向时间从 1 递减到 0，故 $\dot{\alpha}_t \leq 0$ 。因此，该条件速度在前向加噪方向把概率质量从目标词元 $x_i(0)$ 移向基分布 $\bar{p}_i$ ；生成时则沿相反时间方向把概率质量移回目标词元。

为了训练模型，我们需要一个仅依赖于时间 t 时可用信息的条件速度。与标准流

匹配一样，训练目标定义为

$$\mathcal{L}_{\text{DFM}}(\theta) = \mathbb{E}_{t, \mathbf{x}(0), \mathbf{x}(t)} [\|\mathbf{v}_\theta(\mathbf{x}(t), t, \mathbf{c}) - \mathbf{v}^*(\mathbf{x}(t), t, \mathbf{x}(0), \mathbf{c})\|^2]. \quad (4.163)$$

上述单纯形常微分方程视角描述了一条连续的边缘分布曲线，但它没有指定在生成过程中离散样本应如何演化。一个互补的视角是通过一个由时间依赖的生成器矩阵 $\mathbf{Q}^{\text{tok}}(t) \in \mathbb{R}^{|V| \times |V|}$ 参数化的连续时间马尔可夫链来表示离散流。边缘速度 $\mathbf{v}(t)$ 与生成器之间的联系由 Kolmogorov 前向方程给出

$$\frac{d\mathbf{p}_i(t)}{dt} = [\mathbf{Q}^{\text{tok}}(t)]^\top \mathbf{p}_i(t), \quad (4.164)$$

其中 $\mathbf{p}_i(t)$ 是表示词元上概率分布的向量。通过为特定目标词元 z 展开这个矩阵乘法，我们可以将速度解释为一个通量平衡方程

$$\mathbf{v}_i(t)[z] = \sum_{y \neq z} \underbrace{\mathbf{p}_i(t)[y] Q_{yz}^{\text{tok}}(t)}_{\text{inflow from } y \rightarrow z} - \sum_{y \neq z} \underbrace{\mathbf{p}_i(t)[z] Q_{zy}^{\text{tok}}(t)}_{\text{outflow from } z \rightarrow y}. \quad (4.165)$$

这里， $Q_{yz}^{\text{tok}}(t)$ 表示从词元 y 跳转到 z 的瞬时速率。注意，对角线元素满足 $Q_{yy}^{\text{tok}}(t) = -\sum_{z \neq y} Q_{yz}^{\text{tok}}(t)$ 以确保概率守恒。

虽然公式 (4.162) 导出的边缘速度 $\mathbf{v}^*$ 固定了概率的净变化，但它并未唯一确定转移速率 $\mathbf{Q}^{\text{tok}}$ 。为了描述从基分布返回目标词元的生成过程，引入反向时间 $\tau = T - t$ ，并令 $\alpha_\tau^+ = \alpha_{T-\tau}$ 。此时 $d\alpha_\tau^+/d\tau = -\dot{\alpha}_{T-\tau} \geq 0$ 。近期工作中一个常见选择是只允许非目标词元跳向目标样本 $x_i(0)$ [?]，其条件目标速率为

$$\tilde{Q}_{yz}^{*, \text{tok}}(\tau | x_i(0)) = \frac{d\alpha_\tau^+/d\tau}{1 - \alpha_\tau^+} \cdot \mathbb{I}(z = x_i(0)), \quad y \neq x_i(0), \quad \alpha_\tau^+ < 1. \quad (4.166)$$

该式的非对角速率非负，因此定义了合法的生成器；其对角元取为对应行其余速率之和的负值，从而保证概率守恒。这个生成器驱动任何非目标词元 y 直接跳转到目标 $x_i(0)$ ，并产生所需的反向边缘演化。

因此，模型被参数化以预测这个生成器矩阵，训练目标也变成了速率匹配损失，而非公式 (4.163) 中的向量回归。在生成过程中，可以使用 Gillespie 方法 [?] 或 Tau-Leaping [?] 等算法模拟该过程，并执行离散跳转 $x(\tau) \rightarrow x(\tau + \Delta\tau)$ ，这些跳转在统计上遵循学习到的概率流。

至此可以看到，连续时间并不要求状态本身连续：在连续状态空间中，ODE 以向量场规定局部速度，SDE 进一步加入随机扩散；在离散状态空间中，CTMC 则以生成器规定瞬时跳转速率。三类模型都通过局部动力学诱导随时间演化的概率分布，只是在状态表示、轨迹形式和数值模拟方式上有所不同。

课后练习

1. 写出条件自回归模型和单步非自回归模型的概率分解，并比较二者关于序列长度的串行深度。为什么扩散式 NAR 的完整生成复杂度不能简单写成 $O(1)$?
2. 比较嵌入空间扩散中的显式长度预测、EOS 截断和动态长度适应。最近邻舍入为什么可能产生语义平均化问题？请给出两种缓解思路。
3. 连续时间马尔可夫链的生成器矩阵需要满足哪些条件？对二状态生成器 $\mathbf{Q} = \begin{pmatrix} -2 & 2 \\ 1 & -1 \end{pmatrix}$ ，说明每个非对角元素和对角元素的含义。
4. 词表大小为 $|V| = 4$ ，信号保留率为 $\alpha_t = 0.7$ 。分别计算均匀扩散中原词元与任一其他词元的概率，以及吸收式掩码扩散中原词元与 [MASK] 的概率。
5. 从状态表示、局部动力学、训练目标和采样方法四个方面，比较嵌入空间扩散与词元空间离散扩散。若任务要求中间状态始终是合法词元，应优先选择哪条路线？为什么？

4.5.5 本章小结

第五章 多模态机器学习方法与模型

作者：赵润松、陈丹、刘晓倩

前面几章我们建立了机器学习的数学基础与优化方法、强化学习、大语言模型的预训练与微调（包括指令微调与偏好对齐），以及深度学习中的连续动力学与基于流的生成模型。这些内容大多围绕单一模态、尤其是文本数据展开。然而，真实世界的信息通常以多种模态并存：图像是定义在二维空间上的像素信号，语音与音频是随时间变化的声学信号，文本则是离散的符号序列。它们在数据结构、噪声特性与语义粒度上差异显著，却又常常指向相同的语义——同一只猫，可以是一张照片、一句描述，也可以是一段叫声（图 5.1）。

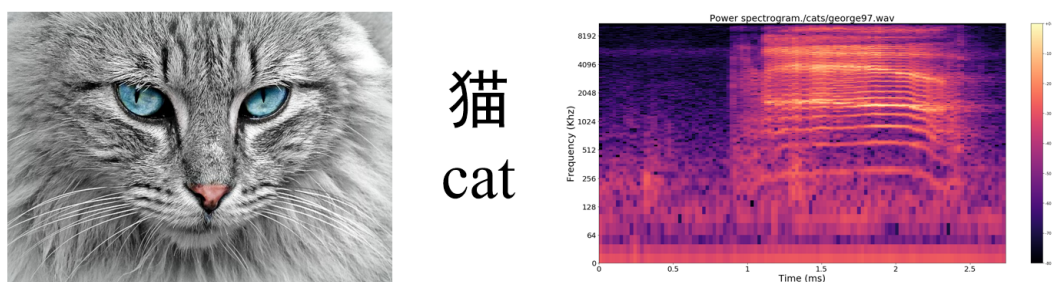


图 5.1: 同一语义“猫”在不同模态中的具体形式：一张猫的图片、文本“猫”（cat）与猫叫声的频谱。表示学习与跨模态对齐所追求的，正是让这三者在向量空间中彼此接近。

本章讨论**多模态机器学习**：研究图像、语音/音频与文本等不同模态如何被统一表示、对齐、建模与生成。全章分为三个部分，依次递进。第一部分是多模态的**表示学习与跨模态对齐**，解决不同模态如何被编码为向量并进入统一语义空间；第二部分是**多模态基础模型的结构与训练**，说明图像、语音如何接入大语言模型，形成理解与交互能力；第三部分是**条件生成建模**，把文本生成图像、图像编辑、语音合成与音频生成，统一表述为在给定条件下学习一个分布并采样的问题。

学习目标

- 掌握图像、语音与文本在计算机中的原始表示形式，以及将其转化为向量表示的动机与方法；
- 理解向量与向量空间的含义，明确“语义相近则表示相近”这一表示学习目标，并据此理解跨模态对齐；
- 掌握以对比学习为代表的跨模态对齐方法及其在图文场景中的实现，了解相关代表工作的发展脉络；

- 理解图像、语音如何接入大语言模型，构成具备多模态理解能力的基础模型；
- 掌握条件生成建模的统一框架，能够将文本生成图像、图像编辑、语音与音频生成，表述为条件分布 $p(x \mid c)$ 的学习与采样问题。

5.1 多模态表示学习与跨模态对齐

本节讨论两个基础问题：其一，图像、语音与文本如何从原始数据转化为模型可处理的**向量表示**；其二，不同模态的表示如何在同一空间中建立**语义对应**，即跨模态对齐。这两个问题是后续多模态模型的前提。

5.1.1 单模态表示学习：从原始信号到向量

数据在计算机中的原始形式。 讨论表示学习之前，需要先明确各模态在计算机中以何种形式存在（见图 5.2），因为后续处理都建立在这一原始形式之上。

图像在计算机中通常是一个数值矩阵。屏幕上的图像由许多小方格构成，每个小方格称为一个**像素**（pixel）。一幅灰度图像可表示为一个 $H \times W$ 的矩阵，其中第 (i, j) 个元素是一个整数（常取 0 至 255），表示该像素的明暗，0 为黑、255 为白。彩色图像一般由红、绿、蓝三个通道叠加而成，对应一个 $H \times W \times 3$ 的三维数组。于是一幅 224×224 的彩色图像，大致相当于 $224 \times 224 \times 3 \approx 1.5 \times 10^5$ 个整数。

语音/音频在计算机中通常是一串随时间排列的数值。声音是空气压强的振动，麦克风按固定的时间间隔（采样率，如每秒 16000 次）测量这种振动的强弱，每次测量得到一个数值，称为一个**采样点**，其取值即**振幅**。因此一段语音可视为一个一维的长向量，长度大致正比于时长。

文本在计算机中通常是一串编号。先准备一张**词表**（vocabulary），把可能出现的词（或子词）逐一编号；一段文本便被转写为这些编号构成的序列。例如词表中“猫”的编号为 352，文本中的“猫”便以整数 352 进入模型。

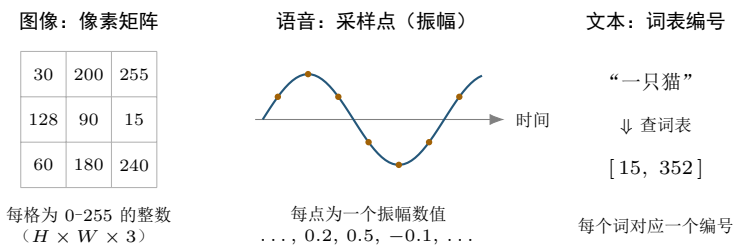


图 5.2: 三种模态在计算机中的原始形式：图像是像素整数矩阵，语音是随时间的振幅采样序列，文本是词表编号序列。

为什么要用向量表示。 上述原始形式虽是数字，却不利于直接学习。一个重要原因是，原始数字之间的数值接近并不等于语义接近：像素值相近的两幅图像内容可能并不相干，内容相近的两幅图像像素值也可能相差很大；词表中编号相邻的两个词往往含义无关。我们希望得到的，是一种语义相近则表示也相近的形式。

为此，机器学习常用**向量**来表示对象。一个向量是一组有序的实数 $\mathbf{x} = (x_1, x_2, \dots, x_d)$ ，其中每个分量 x_k 可理解为对象在某个特征上的取值。也就是说，**向量可以看作对象若干特征的集合**。例如，可设想用不同分量分别刻画一幅图像“是否有毛发”“是否四足”“是否有轮子”等语义特征；若两幅图像在这些特征上取值接近，其向量也接近。真实模型学到的特征并非人为指定，而是从数据中自动学习，但“向量作为特征集合”这一直观通常是成立的。

向量空间与相似度。 全体 d 维向量构成一个 d 维**向量空间**。在这个空间中，每个对象对应一个点，对象间的“相似程度”可用点之间的距离或夹角来量化。常用两种度量：欧氏距离 $\|\mathbf{x} - \mathbf{y}\|_2$ 衡量远近；余弦相似度

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|} \quad (5.1)$$

衡量方向的一致程度，取值越接近 1 表示方向越一致。**表示学习**的目标，可以概括为学习一个从原始数据到向量的映射，使得**语义相近的对象在向量空间中距离较小、语义无关的对象距离较大**（图 5.3）。这一点是理解后续内容的关键：跨模态对齐所追求的，正是让表达相同语义的不同模态对象，在同一向量空间中彼此接近。

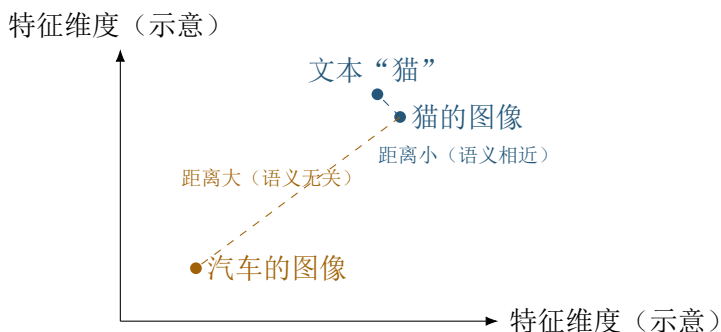


图 5.3: 表示学习的目标：语义相近的对象（无论来自哪个模态）在向量空间中距离较小，语义无关的对象距离较大。

Token：表示的基本单元。 处理图像、语音、文本时，通常不会把整个对象压成单个向量，而是先切分为若干**基本单元**，每个单元各对应一个向量，从而把对象表示为一串向量。这种基本单元称为 **token**。文本的 token 一般是词或子词；图像的 token 常取 16x16 的图块 (**patch**)；语音的 token 可取声学帧或声学单元。把不同模态都整理为“token 序列”的一个好处是：以自注意力为核心的序列模型（Transformer 与大

语言模型)可以较为统一地处理它们,这为多模态与大语言模型的结合提供了结构上的便利。

图像表示：卷积与视觉 token。 如前所述,原始像素通常不适合直接作为语义表示,需要一个**编码器**把像素逐层加工为高层特征。图像编码中最经典的运算是**卷积**(convolution)。

卷积的核心是一个小的权重矩阵,称为**卷积核** K (例如 3×3)。它在输入图像上滑动,在每个位置与所覆盖的局部区域做逐元素相乘再求和,得到输出的一个数值;所有位置的输出组成一张**特征图**。以二维情形为例,记输入为 I 、卷积核为 K ,则输出特征图 S 在位置 (i, j) 处为

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n). \quad (5.2)$$

式 (5.2) 表明:卷积核只关注一小块局部区域(称为**局部感受野**),用于检测该处是否出现某种局部模式,如某方向的边缘或某种纹理;同一卷积核在整幅图像上共享使用(图 5.4)。参数共享带来两点好处:一是显著减少参数量,二是带来一定的**平移不变性**——同一模式出现在不同位置时,可由同一卷积核检测到。

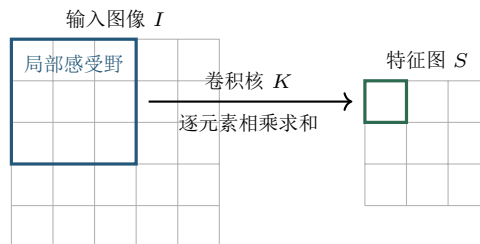


图 5.4: 卷积运算:卷积核在输入上滑动,每个局部感受野经“逐元素相乘再求和”得到特征图的一个值。

以边缘检测为例:取一个用于检测竖直边缘的 3×3 卷积核(如 Sobel 核),按式 (5.2) 在图像上滑动,即可在灰度变化剧烈的竖直边缘处得到较大响应、在颜色平坦处得到接近 0 的响应,从而“点亮”物体的竖直轮廓(图 5.5);换用不同的卷积核,则可分别检测水平边缘、斜向纹理等不同局部模式。

图像编码器的发展脉络。 把多层卷积叠加起来,可从低层的边缘、纹理逐层组合出更高层的语义,这类网络称为**卷积神经网络**(CNN)。CNN 的发展大体经历了从 AlexNet[?] 确立深度卷积网络在图像识别上的有效性,到 VGG[?]、ResNet[?] 通过残差连接把网络显著加深的过程。另一类做法不依赖卷积,而是把图像切成固定大小的图块(patch):例如把 224×224 的图像切成 14×14 个 16×16 的图块,每个图块经线性映射成一个向量,即一个**视觉 token**,随后用自注意力建模这些 token 间的关系,这便是 Vision Transformer (其结构见图 5.6) [?]。两类结构的训练目标类似,都是让所得视觉表示服务于具体任务(如分类),并使语义相近的图像表示彼此接近。

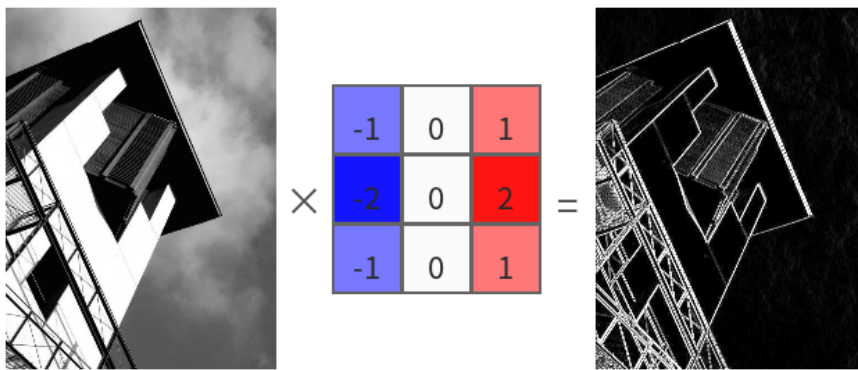


图 5.5: 卷积核用于边缘检测：竖直边缘检测核（中间，Sobel 核）对左侧图像做卷积，得到右侧突出竖直轮廓的特征图。

经过编码后，图像被表示为一组视觉 token，其序列形式与文本 token 较为接近，这为二者进入同一模型创造了条件；不过二者在语义粒度上仍有差异，视觉 token 更接近局部区域特征，而文本 token 通常已对应较完整的语义单位。

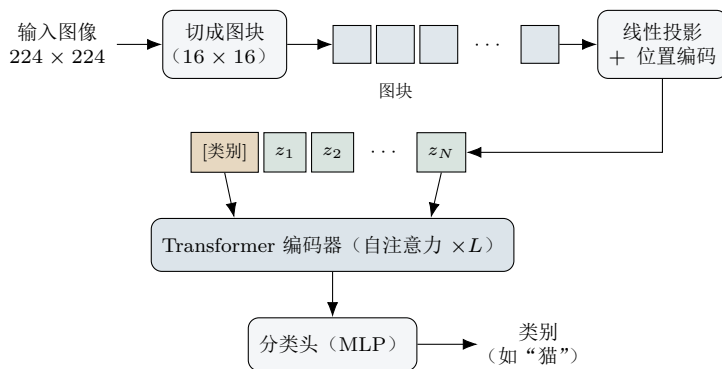


图 5.6: Vision Transformer (ViT)：图像切成固定大小的图块，每块经线性投影加位置编码成为一个视觉 token；连同一个额外的“[类别]” token 一起送入 Transformer 编码器，最后由分类头输出类别。

语音表示：从波形到声学 token。 语音在计算机中的原始形式是一维**波形** (waveform)，即按时间顺序记录的振幅采样点序列。若采样率为 16000 Hz，则一秒语音包含 16000 个数值。波形完整保留了原始声学信号，但其数值随时间快速振荡，且会同时混合说话人音色、发音内容、语速、情绪、环境噪声等多种因素，因此通常不直接把整段波形作为高层语义表示。

一种经典处理方式是先把波形转换为**频谱图** (spectrogram)。具体而言，将连续波形切分为许多短时片段，称为**帧** (frame)；常见帧长约为 20–25 ms，相邻帧之间可有重叠。对每一帧进行短时傅里叶变换 (short-time Fourier transform, STFT)，即可得到该短时间内不同频率成分的强弱。若记波形为 $x[n]$ 、窗函数为 $w[n]$ 、第 t 帧的频

率索引为 k ，则其复频谱可写为

$$X(t, k) = \sum_n x[n + tH] w[n] e^{-j2\pi kn/N}, \quad (5.3)$$

其中 H 是帧移， N 是每帧长度。通常取其模平方 $|X(t, k)|^2$ 作为能量，因此频谱图可以理解为一个“时间 $\times$ 频率”的二维矩阵：横轴表示时间，纵轴表示频率，每个位置的数值表示该时刻附近某个频率成分的能量。

人耳对不同频率的分辨能力并不均匀，低频区域更敏感，因此语音处理中还常将频谱投影到**梅尔尺度**（Mel scale）上，得到**梅尔频谱**（Mel spectrogram）。其基本做法是用一组按梅尔尺度分布的三角滤波器，对频谱能量进行加权汇总，再取对数：

$$M(t, m) = \log \left(\sum_k F_m(k) |X(t, k)|^2 + \epsilon \right), \quad (5.4)$$

其中 $F_m(k)$ 表示第 m 个梅尔滤波器。梅尔频谱保留了语音中与发音内容、音高、共振峰和韵律相关的重要结构，同时压缩了不必要的频率细节，因此长期以来是语音识别、说话人识别和语音生成中的常用输入特征。波形、频谱图与梅尔频谱三种表示的对照见图 5.7。

无论输入是原始波形还是梅尔频谱，语音编码器的目标都是把每一帧或一小段连续信号映射为一个高维向量，从而得到按时间排列的**声学 token** 序列。这里的声学 token 通常是连续向量，并不一定对应一个完整的音节、词或字符；其语义粒度往往比文本 token 更细，还会保留部分说话人、情绪和韵律信息。与图像中“图块 $\rightarrow$ 视觉 token”的过程类似，语音也可视为“时间帧 $\rightarrow$ 声学 token”的过程；区别在于，图像主要在二维空间上展开，而语音沿一维时间轴展开，前后 token 的顺序具有更强的因果性和动态性。同一句话在不同语速下持续时间不同，因此语音 token 序列通常也比对应文本 token 序列长得多。

文本表示：分词、词表与词嵌入。 文本的表示相对直接，其流程如图 5.8 所示：先把一段文字切分为基本单元（词或子词），称为**分词**；每个单元在**词表**中对应一个编号；再通过一张可学习的**词嵌入**（word embedding）表，把编号映射为向量。于是一句话最终成为一串词向量，与图像的视觉 token、语音的声学 token 在形式上一致。

对照图 5.8，以“一只猫”为例：先切分为“一只”“猫”两个 token，查词表得到各自编号，再经词嵌入表取出对应向量。词嵌入的意义与前文一致——把离散编号转化为特征向量，使语义相近的词（如“猫”与“猫咪”）向量也相近。

5.1.2 自监督学习与预训练表示

有了编码器，还需回答它如何学到有用的表示。若依赖“数据+人工标签”的监督训练，则受限于标注的规模与成本；而图像、语音、文本的无标注数据较为丰富。

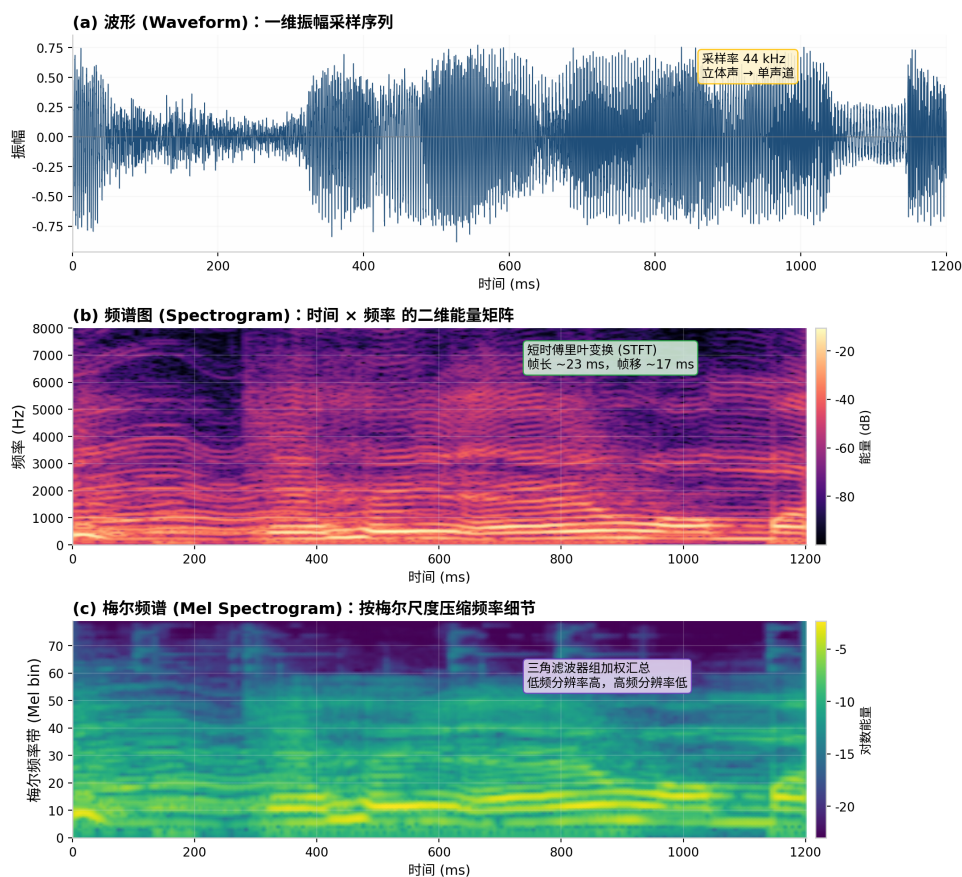


图 5.7: 语音的三种常见表示: (a) 波形, 按时间排列的一维振幅采样序列; (b) 频谱图, 经短时傅里叶变换得到的“时间 × 频率”二维能量矩阵; (c) 梅尔频谱, 按梅尔尺度对频率细节加权压缩后的结果。

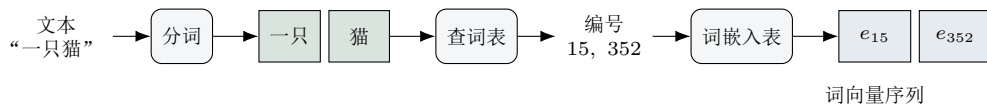


图 5.8: 文本表示流程: 分词得到子词 token, 查词表得到编号, 再由词嵌入表映射为词向量序列。

自监督学习的基本思想，是从数据自身构造监督信号，无需人工标注，从而能够利用大规模数据进行**预训练**，得到可迁移到下游任务的通用表示。下面着重说明其在图像上的两类典型训练目标。

掩码重建。 随机遮挡输入的一部分，让模型根据未遮挡部分重建被遮挡的内容。在图像上，可遮挡若干图块再要求模型恢复其像素或特征；要重建得较好，模型往往需要学到图像的结构与语义规律。其训练目标是 최소화重建误差，与文本中的掩码语言建模在形式上相近。这类方法的代表包括 MAE[?] 与 BEiT[?]。以 MAE 为例（图 5.9），它随机遮住约 75% 的图块，只把剩余可见图块送入编码器，再由一个轻量解码器重建被遮部分；要补得合理，模型必须理解物体结构与上下文（例如由半只火烈鸟推断其全貌）。训练完成后，这套编码器学到的通用表示可迁移到分类、检测等下游任务。

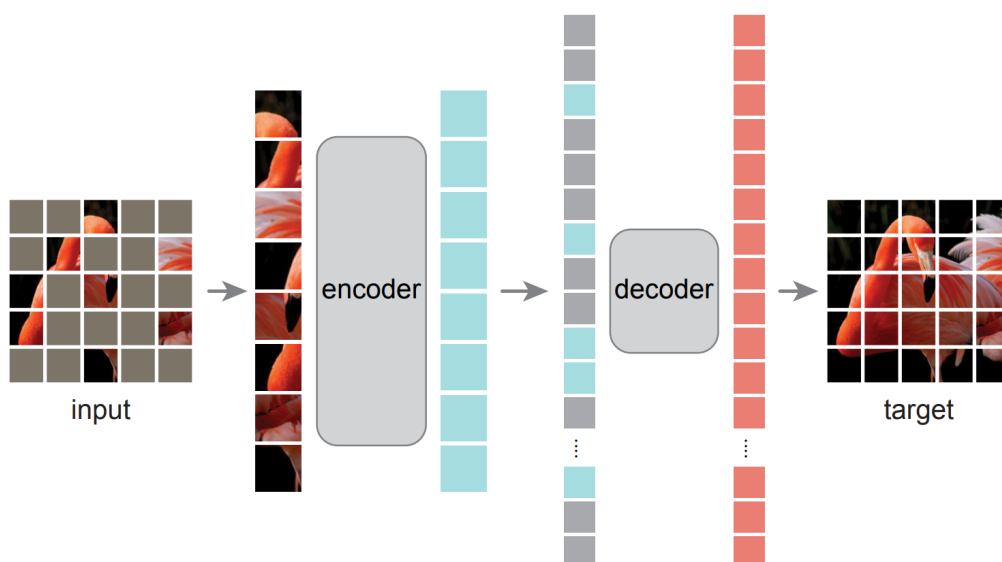


图 5.9: 掩码自编码器 (MAE): 随机遮住大部分图块 (左, input), 仅将可见图块送入编码器, 再由解码器重建被遮部分, 训练目标是还原原图 (右, target)。

对比学习。 对同一对象施加两种不同的随机变换（如对图像做不同的裁剪与颜色扰动），得到的两个视图视为同一语义，称为**正样本对**，要求其表示接近；来自不同对象的视图为**负样本对**，要求其表示远离。其训练目标使正样本对的相似度高于负样本对，从而学到对无关变化（如位置、颜色）较不敏感、对语义较敏感的表达。代表工作包括 SimCLR[?] 与 MoCo[?]; 此外 DINO[?] 等自蒸馏方法也取得了较好效果。下一小节的跨模态对齐，可视为对比学习思想从“不同视图”到“不同模态”的推广。

语音的自监督。 语音同样有海量的无标注录音，因此也希望用自监督的方式学到通用表示。它的核心思路与图像的掩码重建一脉相承，可以概括成一句话：**遮住一段语**

音，再让模型根据前后听到的内容把它“猜”回来（图 5.10）。为了猜得准，模型必须理解语音里稳定的发音规律与上下文，这正是我们想要的表示。

不过语音有一个和图像、文本都不同的特点：它是连续变化的声波，相邻的采样点几乎一模一样，信息高度冗余。如果直接要求模型逐点还原原始波形，它只要照抄邻近的点就能蒙混过关，学不到有用的东西。因此语音自监督一般不去还原波形本身，而是预测更“抽象、稳定”的目标。按预测目标的不同，有两种有代表性的做法。

第一种是**从若干候选里认出正确答案**，代表是 wav2vec 2.0[?]。它先用一个卷积网络把波形转成一串声学向量，随机遮住其中一段，再让 Transformer 根据没被遮住的上下文，为被遮位置给出预测；训练时同时准备好该位置真正的向量和若干来自别处的“干扰向量”，要求模型从中认出正确的那一个。这本质上就是上一段对比学习的思想在时间轴上的应用。

第二种是**先给语音编号，再做“完形填空”**，代表是 HuBERT[?]。它先用聚类（例如对声学特征做 K-means）把每一帧归入某一类，得到一个类别编号，并把这些编号当作无需人工标注的“伪标签”；预训练时遮住一段语音，让模型根据上下文预测被遮位置属于哪个编号。这样，原本连续的语音就被看成一串离散“单元”，预测任务与文本的掩码语言建模非常相似，模型也因此更容易学到音素一级的结构。

两种做法的预测目标不同（认向量还是猜编号），但共享同一个骨架：**遮挡，再用上下文预测**。这也再次说明，图像、文本与语音的自监督在思想上是相通的。

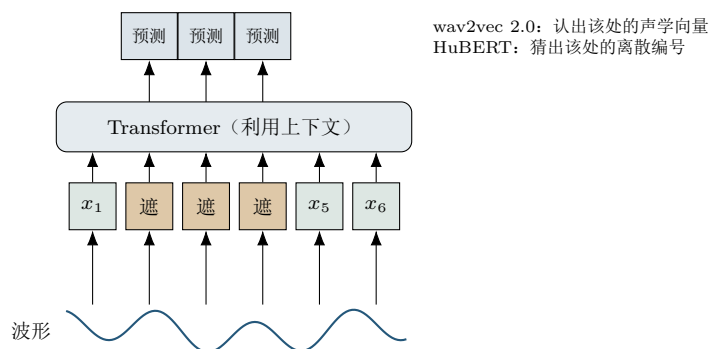


图 5.10: 语音自监督的共同思路：遮住一段语音，让模型根据前后未遮挡的上下文预测被遮部分。wav2vec 2.0 预测该处的声学向量，HuBERT 预测该处的离散编号。

5.1.3 跨模态对齐

前两小节使每个模态各自得到了向量表示，但不同模态的编码器多是各自训练的，其向量分处不同空间，通常难以直接比较。**跨模态对齐**要解决的问题是：把不同模态的表示纳入同一个向量空间，使表达相同语义的对象（如一幅猫的图像与文本“猫”）在该空间中较为接近。

实现对齐的一个常用途径，是利用天然存在的**配对数据**。例如互联网上大量“图像配文字”的样本，本身就标明了哪幅图与哪句话语义对应。据此可训练图像编码

器与文本编码器，使配对样本的表示靠近、非配对样本的表示远离——这正是对比学习从“同一模态的不同视图”推广到“不同模态的配对”的结果。这一思路的代表是 CLIP[?] 与几乎同期的 ALIGN[?]：前者使用较高质量的图文数据，后者表明利用规模更大但噪声更多的网络图文数据同样可行。此后的工作在此基础上做了不少改进，例如 SigLIP[?] 用 sigmoid 形式的损失替代批内 softmax，以改善大批量训练下的稳定性与效率。

以 CLIP 为例，取一批 N 对图文。图像编码器把 N 幅图编码为向量 $\mathbf{I}_1, \dots, \mathbf{I}_N$ ，文本编码器把 N 句话编码为向量 $\mathbf{T}_1, \dots, \mathbf{T}_N$ ，并归一化到同一空间。计算任意一幅图 $\mathbf{I}_i$ 与任意一句话 $\mathbf{T}_j$ 的相似度（可用内积 $\mathbf{I}_i \cdot \mathbf{T}_j$ 表示），得到一个 $N \times N$ 的相似度矩阵（图 5.11）：对角线元素 $\mathbf{I}_i \cdot \mathbf{T}_i$ 为真实配对，应取较高值；非对角线元素为错误配对，应取较低值。训练即调整两个编码器，使该矩阵趋于“对角线高、其余低”。

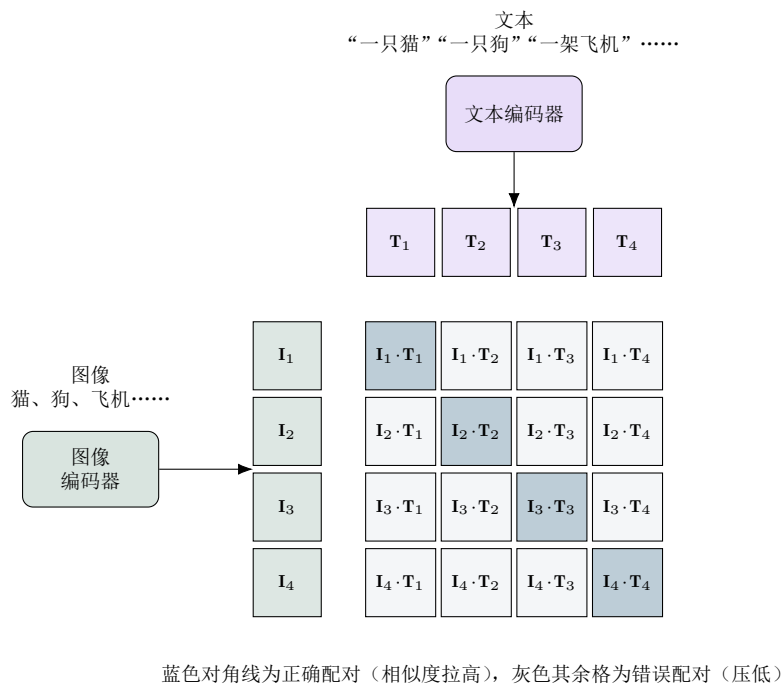


图 5.11: 图文对比对齐 (CLIP): 图像编码器输出 $\mathbf{I}_i$ 、文本编码器输出 $\mathbf{T}_j$ ，两两做内积得到 $N \times N$ 相似度矩阵。训练目标是使对角线元素 $\mathbf{I}_i \cdot \mathbf{T}_i$ 较高、非对角线元素较低。

把这一目标写成损失函数，即对比损失 (InfoNCE)。对第 i 幅图，其“图到文”方向的损失为

$$\ell_i^{\text{图} \rightarrow \text{文}} = -\log \frac{\exp(\cos(\mathbf{I}_i, \mathbf{T}_i)/\tau)}{\sum_{j=1}^N \exp(\cos(\mathbf{I}_i, \mathbf{T}_j)/\tau)}. \quad (5.5)$$

该式可逐项理解：分子是第 i 幅图与其正确配文的相似度，分母是该图与批内全部 N 句话相似度之和。整个分式是一个 softmax，表示“在这 N 句话中，模型判定第 i 句为正确配文的概率”；取负对数后，概率越接近 1、损失越小。 τ 为温度参数， τ 越小，

对“正确配对显著高于错误配对”的要求越严格。将“图到文”与对称的“文到图”两个方向的损失相加，即为完整的对齐目标。

经过对齐，文本与图像被纳入同一向量空间，于是“用文字检索图像”“判断图文是否相符”等任务，大体可归结为在该空间中计算相似度。更重要的是，对齐后的文本表示常被用作图像生成的条件——5.3 的文本生成图像，往往依赖这种图文对应关系，使文本描述能够引导图像的生成方向。

对齐后的图文空间还能直接用于分类，且无需为新类别重新训练：给定一张图片，把若干候选类别分别写成文本“一张关于某类别的照片”（如“猫”“狗”“飞机”），编码后与图像向量计算相似度，相似度最高者即预测类别。只要更换候选文本，就能识别训练时未专门标注过的新类别，这称为**零样本分类**。

语音、音频的对齐。 语音和文本之间天然存在配对关系。例如，语音识别数据提供“语音-文字转写”，语音翻译数据提供“语音-文字翻译”。语音编码器和文本编码器分别将二者映射为向量：

$$\mathbf{s}_i = f_{\text{speech}}(S_i), \quad \mathbf{u}_i = f_{\text{text}}(T_i). \quad (5.6)$$

匹配的语音-文本为正样本，不匹配的组合为负样本。训练仍采用式 (5.5) 的对比损失，使正确配对的表示相互靠近，错误配对的表示相互远离，如图 5.12 所示。

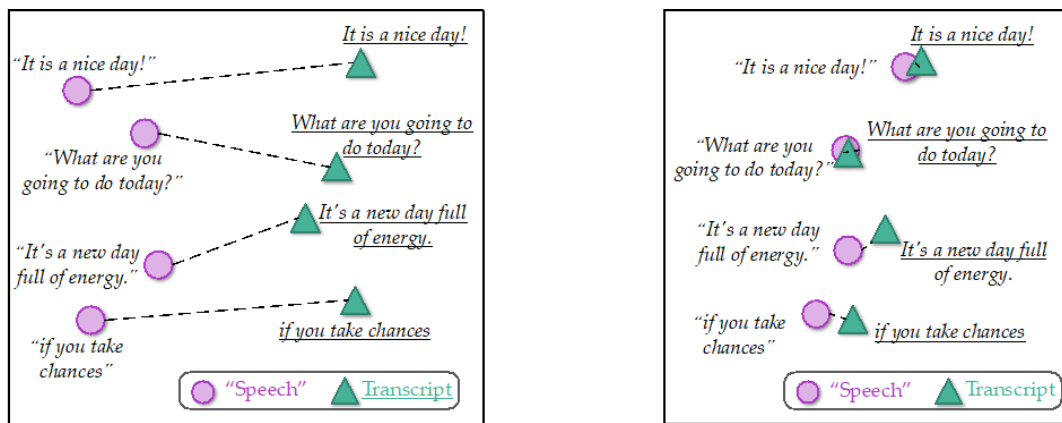


图 5.12: 语音-文本表示的对齐过程。左图为对齐前，右图为对齐后；圆形表示语音，三角形表示对应的文字转写。

句级与词级对齐。 句级对齐将整段语音与整句话分别压缩为一个向量，适合语音检索和语音翻译。词级对齐则进一步确定某段语音帧对应哪个文本 token。例如，WACO[?] 通过拉近语音词片段与对应文本词的表示，提升低资源语音翻译效果。

音频-文本对齐。 更广义的音频还包括音乐、动物叫声、交通声和环境声等。这类音频通常与“狗在吠叫”“雨声伴随雷声”等文字描述配对。CLAP[?] 将 CLIP 中的图像编码器替换为音频编码器，在共享空间中对齐音频和文字描述，如图 5.13 所示。

需要注意。 文字转写主要描述语音中的“说了什么”，通常不会完整记录说话人、情

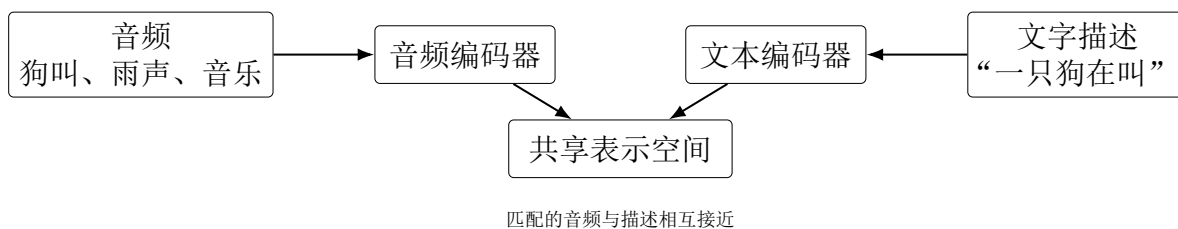


图 5.13: 音频-文本对齐的基本结构。

绪、音色和韵律。因此，只使用转写进行对齐时，模型更容易学习语言内容；若要保留副语言信息，还需加入说话人、情绪或韵律相关的训练目标。

5.2 多模态基础模型结构与训练方法

本节讨论如何把图像、语音等非文本模态接入大语言模型，构成能够理解多模态输入、并按指令完成任务的**多模态基础模型**。需要说明的是，这类模型一般并非把图像模型、语音模型与语言模型简单并置，其关键的机器学习问题，是不同表示空间之间的映射与联合优化。大语言模型自身的预训练、指令微调与对齐方法已在第三部分介绍，本节着重于多模态特有的结构与训练环节。

5.2.1 多模态基础模型的通用结构

现有多模态基础模型大多可归入同一通用结构（图 5.14）：**模态编码器**、**连接模块**与**大语言模型**三段相连。模态编码器（即 5.1 所述的图像或语音编码器）把非文本输入编码为高层表示；大语言模型负责语义理解与文本生成；二者之间的连接模块负责把模态表示映射到大语言模型可接受的输入空间。

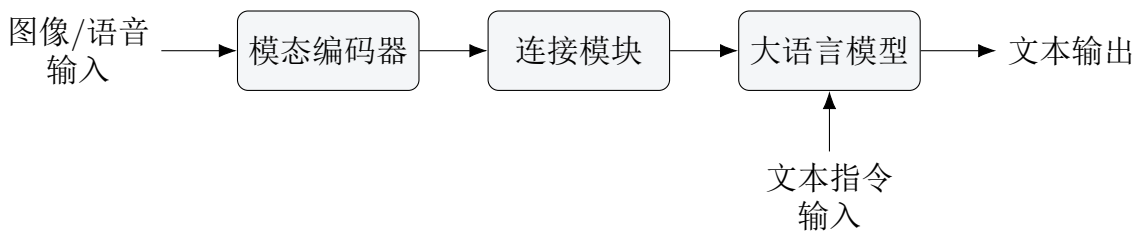


图 5.14: 多模态基础模型的通用结构：编码器负责感知，连接模块负责把模态表示映射到语言模型空间，文本指令与映射后的模态表示一并输入大语言模型，由其完成理解与表达。

按融合发生的早晚看三类结构。上述通用结构之外，还需回答一个问题：来自不同模态的信息，究竟在模型的哪一步结合到一起？按**融合发生的早晚**，大致可分为三类（图 5.15），它们正对应连接方式由浅到深的不同选择：

- **后期融合**：图像先经一个独立的视觉编码器完整编码，再经投影与文本拼接后接入语言模型，融合发生得较晚、较浅。各模块相对独立，最易复用现成的视觉编码器与语言模型。代表是 LLaVA[?]，其连接模块只是一个轻量投影层。上文给出的通用结构即以这类为典型。
- **中间层融合**：图像与文本先各自编码，再通过**交叉注意力**在编码器与语言模型之间交互。代表是 BLIP-2[?]：它在冻结的图像编码器与冻结的语言模型之间插入一个 Q-Former，用一组可学习查询向量、以交叉注意力从图像特征中提取出少量视觉向量，再送入语言模型；Flamingo[?] 则在语言模型各层插入门控交叉注意力以引入视觉信息。
- **早期融合**：把图像编码成嵌入后，与文本嵌入交错成同一序列，交由一个联合训练的统一模型处理；融合发生在输入端、程度最深，模型甚至可以统一地理解并生成图像与文本。代表是 Emu[?]，它把视觉嵌入与文本 token 放入同一自回归序列端到端训练。

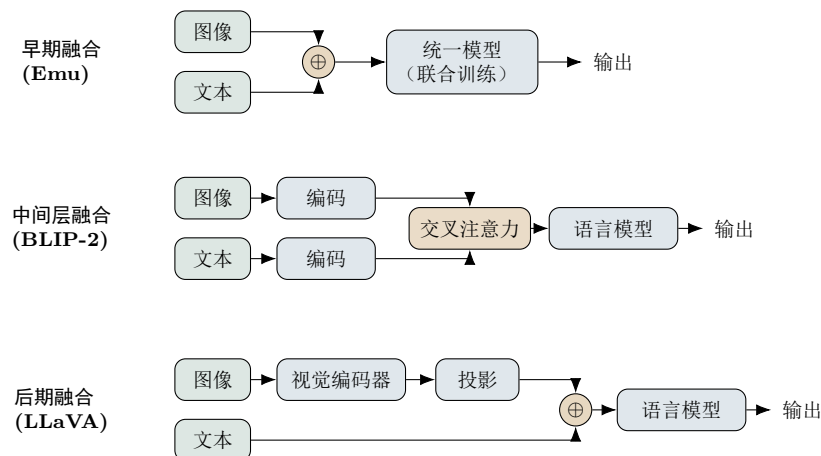


图 5.15: 三类融合按“融合发生的早晚”区分（ $\oplus$ / 交叉注意力为融合点，由上到下依次后移）：早期融合（Emu）在输入端就把图文嵌入并入同一序列、统一建模；中间层融合（BLIP-2）各自编码后用交叉注意力在编码器与语言模型之间交互；后期融合（LLaVA）让视觉先经独立编码器完整编码，末端才与文本拼接接入语言模型。

三者的差别，直观上就是图 5.15 中融合点由上至下逐渐后移：越靠早融合，模态间交互越深、越依赖联合训练；越靠后融合，各模块越独立、越易复用已有的视觉编码器与语言模型。需要说明的是，这一划分并非绝对，实际模型可能兼有多种方式；连接模块的具体形式将在 5.2.2 展开。

连接模块与分阶段训练。 无论属于上述哪一类，连接模块的作用都是把模态编码器输出的向量，转换为若干与文本 token 处于同一空间的向量，相当于把图像或语音“翻译”为语言模型可读取的一段输入。训练上常采用**分阶段**策略：先冻结模态编码器与大语言模型、只训练连接模块，使两个表示空间先行对齐，这样既降低计算开销，又有助于保留语言模型已有的能力；随后再对整体做小幅微调。

语音的接入。 语音和音频同样可以按照“编码器-连接模块-大语言模型”的结构接入。首先，语音编码器将波形或梅尔频谱转换为声学表示：

$$\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_T.$$

连接模块再对这些表示进行**时序压缩**和空间投影，得到较短的语音 token 序列：

$$\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_K, \quad K \ll T.$$

这些语音 token 与文本指令一起输入大语言模型，使其能够完成语音问答、语音翻译和音频理解等任务，如图 5.16 所示。

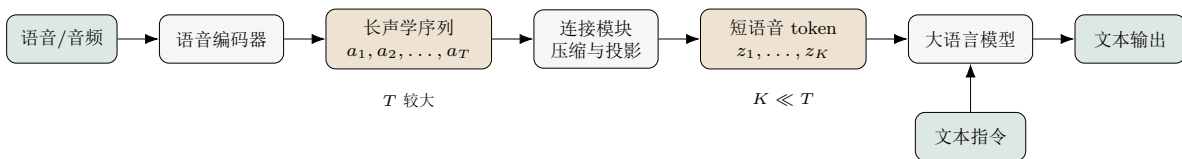


图 5.16: 语音接入大语言模型的基本结构。连接模块先压缩较长的声学序列，再将其投影为大语言模型可读取的语音 token。

语音与图像接入的主要区别在于，语音沿时间轴连续展开，声学 token 通常数量更多。如果直接将全部声学 token 输入语言模型，不仅计算开销较大，还可能挤占文本上下文长度。因此，连接模块常采用卷积下采样、池化、查询向量或注意力压缩等方法，在尽量保留语义和时间顺序的同时缩短序列。

语音的融合。 语音首先经过语音编码器，被转换为按时间排列的声学 token $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_T$ 。这些 token 与文本或图像信息融合时，同样落在前面“融合发生早晚”的框架内：可以把声学 token、图像 token 和文本 token 拼接成统一序列，再交给同一个 Transformer 处理（偏早期融合）；也可以保留独立的语音编码器，让文本 token 通过交叉注意力读取语音特征（偏中间层融合），如图 5.17 所示。

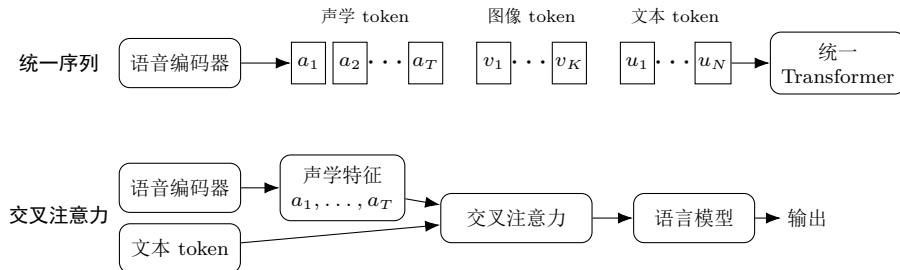


图 5.17: 语音与其他模态的两种常见融合方式。上：将声学、图像和文本 token 拼接为统一序列；下：文本通过交叉注意力读取语音特征。

与文本相比，语音 token 数量更多，并且具有明确的时间顺序。因此，融合前通常需要通过卷积或池化进行下采样，以缩短序列；同时加入位置编码，保留“先说什

么、后说什么”的时序关系。对于语音问答、语音翻译等任务，模型还应避免在处理当前时刻时破坏语音原有的时间结构。

5.2.2 视觉语言模型：图像理解中的多模态建模

把上述通用结构配以图像编码器，即得到**视觉语言模型**（VLM）：能够接收图像输入，并围绕图像进行描述、问答与推理。从机器学习角度看，视觉语言模型把图像表示作为条件，使大语言模型在该条件下进行文本生成，即建模条件分布 $p(\text{文本} | \text{图像})$ 。

其工作方式如图 5.18 所示：图像经编码器与连接模块转化为若干视觉 token，置于文本提示词之前，与文本 token 共同构成大语言模型的输入；模型随后如同处理普通文本上下文那样，自回归地逐词生成回答。对模型而言，图像相当于上下文的一部分，区别在于这部分来自图像。

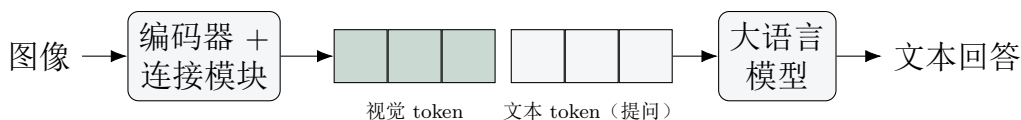


图 5.18: 视觉语言模型：图像被编码为视觉 token，与文本 token 拼接后输入大语言模型，由模型在图像条件下生成回答。

代表工作：连接方式的对比。 不同视觉语言模型的主要差别，往往在于**连接模块**的设计，即如何把视觉编码器的输出接入大语言模型。图 5.19 对比了三种有代表性的做法，三者的视觉编码器（绿色）与大语言模型（蓝色）大体相同，差异集中在中间的连接模块（黄色）。

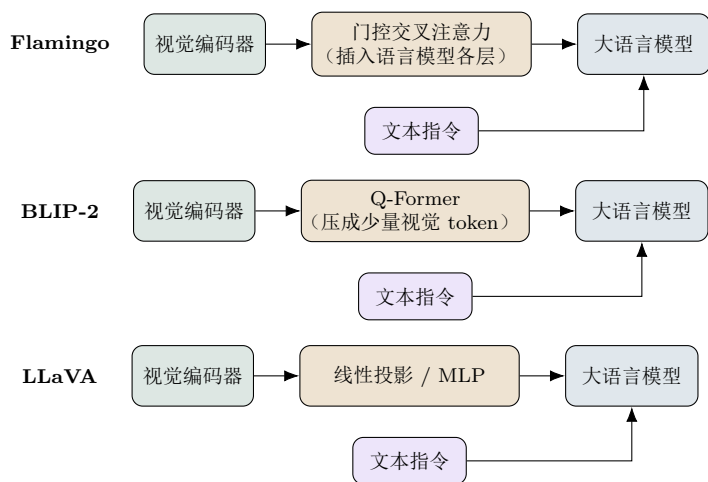


图 5.19: 三种代表性视觉语言模型连接方式对比：视觉经编码器与连接模块（黄色）接入大语言模型，文本指令（紫色）同样输入大语言模型；三者的差异主要在于中间连接模块的形式。

对照图 5.19：第一行的 **Flamingo**[?] 在冻结的语言模型各层中插入**门控交叉注意力**，让文本在生成时按需“查看”视觉特征，适合处理图文交错的序列与少样本

学习；第二行的 **BLIP-2**[?] 在冻结的图像编码器与冻结的大语言模型之间接入一个轻量的 **Q-Former**：它带有一组可学习的查询向量（如 32 个），通过交叉注意力从图像特征中提取出固定数量的视觉向量（数量只由查询个数决定，与图像分辨率、图块数量无关），经一个线性层投影后作为“软提示”送入语言模型；训练时只更新 Q-Former 与该投影层，并分两阶段进行——先把 Q-Former 接在图像编码器上学习图文对齐的视觉表示，再把其输出接入语言模型学习看图生成文本，因而参数高效；第三行的 **LLaVA**[?] 采用最简单的**线性投影（或 MLP）**，把视觉特征直接映射到词向量空间，充当视觉 token。可以看出，连接模块从复杂的交叉注意力逐步走向更轻量的投影，而模型效果并未因此下降——这与下面要讲的训练方式密切相关。

三种连接模块的具体结构。 图 5.19 只标出了连接模块的名称，下面进一步说明每一种“黄色方块”内部到底是什么。**线性投影**（图 5.20）最简单：把视觉编码器输出的每个特征向量，用一个矩阵 W （或两层 MLP）映射到与词向量相同的维度，所得向量直接当作视觉 token 插入文本序列。**Q-Former**（图 5.21）引入一组可学习的查询向量，通过交叉注意力从大量图像特征中“问”出所需信息，输出固定数量的少量 token，从而把成百上千个图像特征压缩到几十个，显著缩短序列。**门控交叉注意力**（图 5.22）不改动文本序列，而是在语言模型的层中插入一个交叉注意力子层，让文本隐状态去读取视觉特征；其输出先乘一个门控 $\tanh(\alpha)$ 再残差加回， α 初始为 0，使训练之初几乎不影响原语言模型，因而更稳定。可以看出，从线性投影到 Q-Former 再到门控交叉注意力，连接模块在结构复杂度和对原模型的改动程度上依次增加。

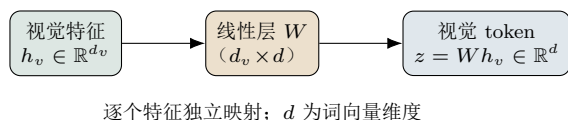


图 5.20: 线性投影连接模块 (LLaVA)：用一个矩阵 W （或两层 MLP）把视觉特征映射到与词向量同维的空间，直接充当视觉 token。

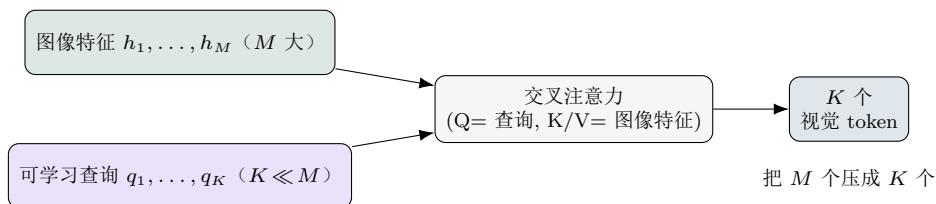


图 5.21: Q-Former 连接模块 (BLIP-2)：一组可学习查询向量通过交叉注意力从大量图像特征中提取信息，输出固定的少量视觉 token，实现压缩。

训练范式的演进。 除连接结构外，训练方式也在演进。早期以图文对齐或图文生成的预训练为主，例如 CLIP[?] 训练出的图文对齐视觉编码器，常被后续模型直接复用为视觉塔。此后**视觉指令微调**成为关键一步：LLaVA 借助较强的纯文本模型，自动构造“图像+指令+回答”数据对模型进行微调，使其能较好地遵循人类指令；

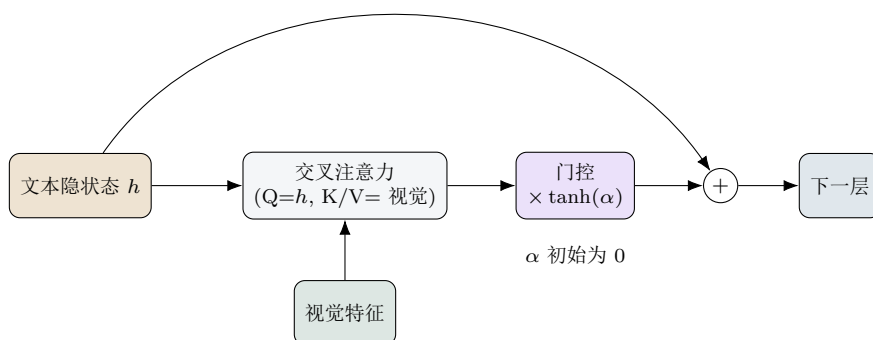


图 5.22: 门控交叉注意力连接模块 (Flamingo): 在冻结的语言模型层中插入交叉注意力读取视觉特征, 输出经门控 $\tanh(\alpha)$ 缩放后残差加回; α 初始为 0, 初始时几乎不干扰原模型。

InstructBLIP[?]、Qwen-VL[?] 等则在指令数据与模型结构上进一步扩展。需要说明的是, 这些模型在不同任务上各有优劣, 且“主流”随时间变化, 此处仅将其视为发展过程中的代表, 而非确定的优劣排序。

重点提示

视觉语言模型可能出现**视觉幻觉**现象: 模型有时会描述出图像中并不存在的内容。其成因之一, 可能是语言先验较强——模型从大量文本中习得了“某场景通常包含某物”的统计倾向, 以致在图像未提供相应证据时仍作出断言。抑制幻觉是多模态对齐训练关注的问题之一。

举一个具体例子: 给定一张“厨房餐桌上放着水果”的图片与问题“图中有几个苹果?”, 模型先由视觉编码器把图像转成视觉 token, 与问题文本一起输入大语言模型, 再据图像内容作答“有两个苹果”; 若图中并无苹果而模型仍答“有一个苹果”, 便是上述视觉幻觉。

5.2.3 语音语言模型: 语音理解中的序列建模

与视觉语言模型相对应, **语音语言模型**把语音表示接入大语言模型, 以完成语音识别、语音理解、语音问答与语音对话等任务, 其结构大体遵循 5.2.1 的通用框架。二者的主要差异在于数据结构: 图像更多体现空间结构, 语音更多体现时间结构, 且语音中常同时包含语言内容、说话人信息、情感与噪声。因此语音语言模型的一个核心, 是把连续声学信号压缩、映射为大语言模型可处理的语义序列。

从语音识别到语音理解。 传统语音识别主要学习

$$p(\text{转写文本} \mid \text{语音}),$$

目标是准确回答“语音中说了什么”。语音语言模型则进一步结合文本指令，学习

$$p(\text{回答} | \text{语音}, \text{指令}),$$

因此输出不必是逐字转写，也可以是问题答案、翻译结果或对话回复。二者的区别如图 5.23 所示。

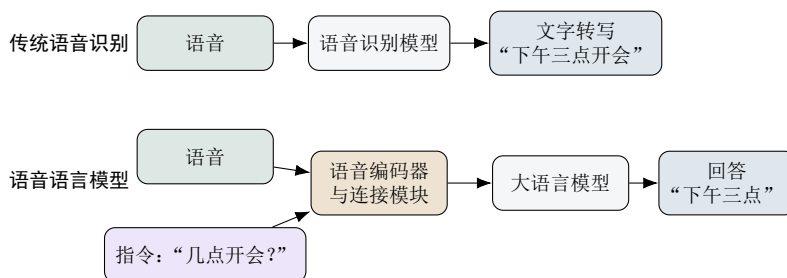


图 5.23: 从语音识别到语音语言模型：传统模型主要输出逐字转写，语音语言模型则结合语音和指令完成问答、翻译与对话等任务。

举例来说，给定一段语音“几点开会？”，传统语音识别只输出其逐字转写“几点开会”，即建模 $p(\text{转写文本} | \text{语音})$ ；而语音语言模型可结合指令直接作答，例如根据日程回答“下午三点”，即建模 $p(\text{回答} | \text{语音}, \text{指令})$ 。同一段语音，前者是“转写”，后者是“理解并回应”。

语音如何进入语言模型。 常见方法有两类，如图 5.24 所示。

1. **连续特征接入**：语音编码器输出连续声学表示，再由连接模块进行压缩和投影，使其维度与文本词向量一致。Qwen-Audio[?] 和 SALMONN[?] 等模型主要采用这种结构。
2. **离散 token 接入**：先将语音量化为有限集合中的离散单元，再把这些单元加入语言模型词表，使同一个模型能够统一处理文本 token 和语音 token。SpeechGPT[?] 采用了这种思路。

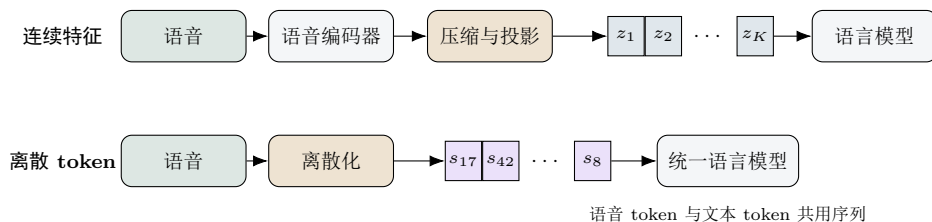


图 5.24: 语音 token 接入语言模型的两种方式。上：将连续声学特征压缩并投影到词向量空间；下：将语音离散化，与文本 token 统一建模。

语音与文本 token 的对齐。 连续接入方法通过连接模块，使语音表示落入语言模型能够读取的词向量空间；离散接入方法则让语音 token 与文本 token 使用同一个自回归模型。训练时，一段语音通常与转写、翻译或回答配对，模型根据语音和指令预测目标文本：

$$\mathcal{L}_{\text{speech}} = - \sum_{l=1}^L \log p(y_l | y_{<l}, \mathbf{z}_{1:K}, \mathbf{q}), \quad (5.7)$$

其中 $\mathbf{z}_{1:K}$ 为语音表示， $\mathbf{q}$ 为文本指令， y_l 为第 l 个目标文本 token。模型必须从语音中提取与答案有关的内容，才能正确预测后续文本，从而逐渐建立语音与文本语义之间的对应关系。

训练能力的逐步扩展。 语音语言模型通常经过以下三个阶段：

1. **语音-文本训练：**利用语音与转写或翻译数据，先学习语音内容与文字之间的对应关系。
2. **语音指令微调：**加入“语音+问题+回答”数据，使模型从简单转写扩展到问答、翻译、摘要和信息提取。
3. **语音对话训练：**使用多轮对话数据，使模型结合前文，持续理解用户语音并生成连贯回复。

Whisper[?] 是大规模多语言语音识别与语音翻译的代表，说明了大规模语音-文本训练可以显著增强跨语言和跨场景的识别能力。在此基础上，SpeechGPT[?] 进一步使用离散语音表示，使语言模型能够接收和生成语音；Qwen-Audio[?] 将训练范围扩展到语音、音乐和环境声等多种音频任务；SALMONN[?] 则把语音编码器、通用音频编码器与语言模型结合，用于语音识别、翻译、音频问答和情感识别等任务。

现实语音中的困难。 真实语音比书面文本更加复杂，主要体现在以下几方面：

- **口语化与方言：**语音中常有重复、停顿、口头语和不完整句子，不同地区的口音与方言也可能在训练数据中分布不均。
- **噪声与混响：**交通声、音乐和远距离录音会遮盖发音内容，使声学表示不稳定。
- **多人重叠：**多人同时说话时，模型还需判断“谁说了什么”，往往需要结合语音分离或说话人分段。
- **长语音与实时性：**语音 token 序列很长，过度压缩可能丢失细节，而保留全部 token 又会增加计算量和回答延迟。

重点提示

语音理解并不完全等于“先转写，再让语言模型回答”。逐字转写主要保留“说了什么”，却可能丢失语气、情绪、停顿、说话人和环境声音。因此，语音语言模型通常仍需直接读取声学表示，而不能只依赖转写文本。

5.2.4 多模态指令微调与对齐

模型具备图像或语音输入能力后，一般仍需通过**指令微调**学会理解用户意图并按规定格式作答，再通过**偏好对齐**使输出更符合人类偏好与安全要求。指令微调、RLHF 与 DPO 等方法本身已在第三部分讨论，本小节只说明其 在多模态情形下的特殊之处。

其特殊性主要体现在两方面。其一，**指令数据须包含非文本模态**：多模态指令数据通常由“图像/语音+指令+目标回答”构成，如图像问答、图像描述、图文推理等；其构造成本较高，实践中常借助已有的强文本模型，依据图像的文字标注自动生成大量问答样本（LLaVA 即采用了类似做法）。其二，**对齐须兼顾模态一致性**：除一般的有用性与安全性外，多模态对齐往往还需抑制前述视觉幻觉，使模型的回答尽量忠实于实际输入的图像或语音，而非依赖语言先验作出臆测。

语音侧的指令数据。 语音指令数据通常由

语音输入 + 文本指令 + 目标回答

组成。它不再只要求模型逐字转写语音，而是要求模型根据语音内容完成指定任务。常见数据形式如图 5.25 所示。

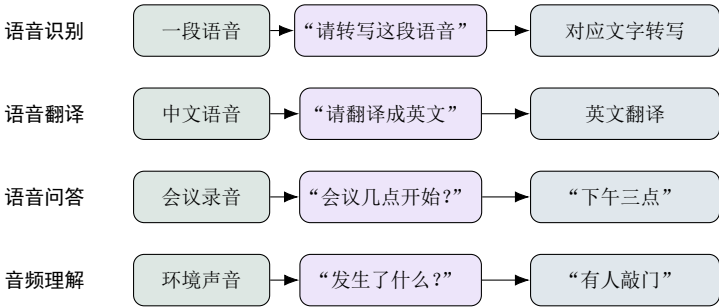


图 5.25: 语音侧指令数据的常见形式：同一模型根据不同指令，对语音完成转写、翻译、问答或音频理解。

已有的语音识别和语音翻译数据可直接改写为指令格式。例如，将“语音-转写”样本包装为“请转写这段语音-目标转写”，将“语音-翻译”样本包装为“请翻译成某种语言-目标译文”。对于语音问答和摘要，还可以先利用语音转写、场景标签或人工标注，再由较强的文本模型生成问题与参考答案。SpeechGPT[?]、Qwen-Audio[?] 和 SALMONN[?] 等工作均通过混合多种语音和音频任务，使模型能够根据指令选择不同的处理方式。

5.2.5 图像—语音—文本联合建模

现实信息往往多模态并存：视频同时包含画面、声音与字幕；课堂同时包含板书、讲解与文字资料。**联合建模**要求模型在同一上下文中同时处理图像、语音与文本，分别对应空间结构、时间结构与符号结构，并在统一表示中完成跨模态的理解与生成。其主要挑战，在于协调这几种异质结构，尽量保持空间、时间与语义上的一致。从双模态系统推广到三模态及以上的统一系统，是近年多模态模型的一个发展方向，一些较新的模型已尝试在统一框架内原生支持多种模态的输入与输出。

以语音/音频为核心的联合场景。 在视频、课堂和会议等场景中，语音或音频通常沿时间连续展开，可作为组织其他模态的**时间轴**。模型需要把同一时刻附近的画面、声音和字幕组合起来，才能完整理解正在发生的事件。

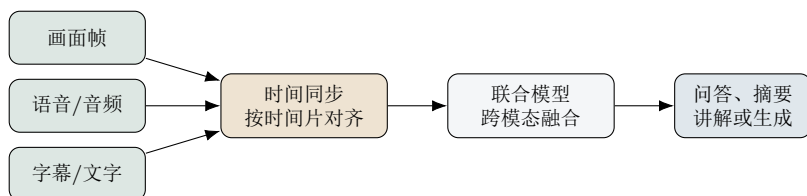


图 5.26: 以语音或音频时间轴为核心的联合建模：先对齐同一时间片内的画面、声音与字幕，再进行统一理解与生成。

视频理解。 画面提供人物、物体和动作，语音提供说话内容，环境声音则可提示画面外发生的事件。例如，仅凭画面可能看不出门外有人，而敲门声能够补充这一信息；字幕还可以帮助模型理解口音较重或受噪声干扰的语音。

看图配音语音讲解。 模型也可以同时接收图像和文本指令，再生成相应的语音讲解。例如，根据一幅医学图像生成口头说明，或根据课件内容生成教学讲解。此时不仅要保证语音内容与图像一致，还要控制语速、停顿和语气，使生成结果适合具体场景。

长上下文中的时间同步。 对于长视频、课堂录音或多人会议，模型通常先将输入切分为若干时间片，形成

$$(V_1, A_1, T_1), (V_2, A_2, T_2), \dots, (V_L, A_L, T_L),$$

其中 V_l 、 A_l 和 T_l 分别表示第 l 个时间片内的画面、音频和文字。模型需要保留这些时间片的先后顺序，并正确判断一句话、一个动作和一种声音是否发生在同一时刻。若时间对齐错误，就可能把后面的声音解释为前面的画面，从而产生错误的事件描述。

5.3 条件生成建模与多模态内容生成

本节讨论多模态内容的生成。文本生成图像、图像编辑、语音合成、文本生成音频等任务表面各异，但大体可统一表述为同一个数学问题：在给定条件 c 下，学习目

标内容 x 的条件分布并从中采样，

$$\text{学习 } p(x | c) \text{ 并从中采样得到新的 } x. \quad (5.8)$$

其中条件 c 可为文本、图像、语音或指令，目标 x 可为图像、语音或音频。本节先建立条件生成的基本概念，再依次讨论文本生成图像、图像编辑、语音生成、音频生成与跨模态交互生成。阅读每一种任务时，建议对照式 (5.8)，明确其中的 c 与 x 各指什么。

5.3.1 条件生成模型基础

概率分布与采样。 理解生成模型，先要明确两个基本概念。**概率分布**是对一个随机对象所有可能取值给出概率（离散情形）或概率密度（连续情形）的描述。以图像为例，可设想在“所有可能图像”构成的高维空间上存在一个分布 $p(x)$ ：真实、自然的图像所在区域概率较高，杂乱无意义的像素组合概率较低。**采样**则是按某分布抽取一个具体取值。需要强调，机器学习中所说的“生成”，大体上可以理解为从一个分布中采样。

这一概念对文本与图像是相通的。语言模型在生成时，每一步给出“下一个词”在词表上的概率分布，再据此抽取一个词，逐步连成句子——这就是**采样文字**。图像生成则是在图像分布上抽取一个高维向量（即一幅图像）——这就是**采样向量**。无论目标是离散的词还是连续的图像向量，生成大体都遵循“先有分布、再采样”的模式。

判别模型与生成模型。 据此可区分两类模型。**判别模型**学习的是条件 $p(y | x)$ ，即由输入预测标签，关注类别之间的分界；先前多数任务属此类。**生成模型**学习的是数据本身的分布 $p(x)$ ，学成之后即可采样出新的、训练集中未必出现过的样本。**条件生成模型**进一步引入条件 c ，学习 $p(x | c)$ ，从而能按指定条件采样，例如要求“生成一只猫”而非任意图像。在多模态生成中，条件 c 正是连接不同模态的纽带：文本可作为图像生成的条件，原图与指令可作为图像编辑的条件。

常见生成建模方法。 如何建模并从复杂分布中采样，历史上发展出若干方法（表 5.1）：生成对抗网络 [?]、变分自编码器 [?]、扩散模型 [?]，以及第四部分已讨论的基于流的生成模型。本节将以扩散模型为主线，因其近年在图像、语音、音频生成中应用较广。各方法的数学细节不在此展开；它们的共同点在于都要回答同一个问题：如何建模数据分布并从中采样。

表 5.1: 常见生成建模方法（仅列核心思想）

方法	核心思想	代表
自回归	按顺序逐元素建模 $p(x) = \prod_t p(x_t x_{<t})$ ，逐步采样	语言模型、图像自回归
变分自编码器	引入隐变量，优化重建与分布约束的折中	VAE
生成对抗网络	生成器与判别器对抗，提升样本逼真度	GAN
扩散模型	正向逐步加噪、反向逐步去噪并采样	DDPM
基于流的模型	以可逆变换在简单分布与数据分布间建立映射	Normalizing Flow

5.3.2 文本生成图像

文本生成图像建模的是 $p(\text{图像} | \text{文本})$ ：给定一段文字描述，采样出与之语义较为一致的图像。这一方向经历了较明显的方法演变：早期曾用生成对抗网络直接由文本生成图像；DALL·E[?] 将图像离散为 token 后以自回归方式生成；此后扩散模型逐渐成为较主流的一类方法，GLIDE[?]、DALL·E 2[?]、Imagen[?] 在开放域文本生成图像上取得了较好效果，而潜在扩散模型 [?]（即 Stable Diffusion 的基础）通过在低维潜在空间中做扩散，显著降低了计算成本。下面以扩散模型为例，较完整地说明其训练与采样过程，并解释它如何实现式 (5.8) 所要求的“建模分布并采样”。

基本思想：加噪与去噪。 第四章已从连续时间随机微分方程的角度介绍了扩散模型的正向扰动与反向生成过程；本节沿用一致的记号（以 $\mathbf{x}$ 记图像数据、 p_{data} 记数据分布、 $\mathcal{N}(\mathbf{0}, \mathbf{I})$ 记先验分布），并从条件生成的角度，说明文本如何作为条件引导反向过程生成图像。为便于直观理解，这里采用离散时间步的表述，其思想与第四章的连续时间视角一致。扩散模型包含两个方向相反的过程（图 5.27）。**正向过程**对一幅真实图像 $\mathbf{x}_0 \sim p_{\text{data}}$ 逐步加入高斯噪声，经过 T 步后得到接近纯噪声的 $\mathbf{x}_T$ ；这一过程是预先设定的，无需学习。**反向过程**则需要学习：训练一个神经网络，使其能从带噪图像中估计所含噪声，从而每步去除一部分噪声。学会“去噪”之后，生成时便从先验分布 $\mathcal{N}(\mathbf{0}, \mathbf{I})$ 采样出发，反复去噪，逐步还原出一幅图像。

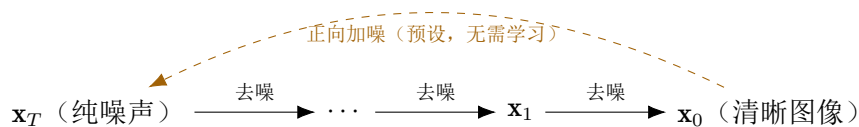


图 5.27: 扩散模型：正向逐步加噪（虚线），反向逐步去噪并采样（实线）。生成从纯噪声开始，沿反向链得到图像。

训练目标。 正向过程加噪到第 t 步，可一次写出

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (5.9)$$

其中系数 $\bar{\alpha}_t$ 随 t 增大而减小，表示越靠后原图成分越少、噪声成分越多。训练一个网络 $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c})$ ，令其根据带噪图 $\mathbf{x}_t$ 、步数 t 与条件 $\mathbf{c}$ 估计所加噪声 $\boldsymbol{\epsilon}$ ，目标是估计得尽量准确：

$$\mathcal{L}_{\text{diff}} = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}, t} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c}) \right\|_2^2. \quad (5.10)$$

此处的条件 $\mathbf{c}$ 即文本：文字经文本编码器（常用 5.1.3 中与图像对齐过的文本表示）转为向量后注入网络，使每一步去噪都依赖于文本，从而引导生成朝符合描述的方向进行。

如何采样：从噪声到图像。 训练完成后，可按如下方式采样，对应式 (5.8) 的“从 $p(\mathbf{x} | \mathbf{c})$ 采样”。先从先验分布采样初值 $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ；随后令 t 从 T 递减到 1，反复执行

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{c}) \right) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (5.11)$$

即每一步先用网络估计并去除一部分噪声，再注入少量随机噪声 $\sigma_t \mathbf{z}$ 以维持采样的随机性；迭代至 $t = 1$ 得到的 $\mathbf{x}_0$ 即为生成的图像。可以看出，这条反向链条实际上定义了一个以 $\mathbf{c}$ 为条件、可从中采样的分布 $p_\theta(\mathbf{x}_0 | \mathbf{c})$ ，它对应我们要建模的 $p(\mathbf{x} | \mathbf{c})$ 。由于每次采样的初值不同，同一段文字往往可生成多样的结果。

直观地看，这条采样链条就是让图像“从噪声中逐步显影”：起初画面几乎全是噪点，随着式 (5.11) 的去噪步骤反复推进，轮廓、颜色与细节逐渐浮现，最终得到清晰图像。

增强文本一致性：无分类器引导。 为加强生成对文本的服从程度，实践中常采用**无分类器引导** [?]：在每一步同时计算“给定条件”与“不给条件”两种噪声估计，并放大二者之差，

$$\tilde{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, \mathbf{c}) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset) + w (\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{c}) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \emptyset)), \quad (5.12)$$

其中 $\emptyset$ 表示不提供条件，引导强度 $w > 1$ 。括号内是“顾及文本”相对“不顾文本”所多出的方向，乘以 w 即放大对文本的服从。一般而言， w 越大，结果越贴合文本，但样本多样性可能下降，需要权衡。

举一个完整的例子：给定文本提示词“一只戴着红色帽子的柯基犬坐在草地上，阳光明媚”，模型从纯噪声出发、在该文本条件下逐步去噪，生成与描述语义一致的图像；改变提示词中的属性（如帽子颜色、背景、画风），生成结果会相应变化；而提高无分类器引导强度 w ，结果会更贴合文字，但多样性随之下降。

5.3.3 图像编辑：约束条件下的生成

文本生成图像是从噪声生成全新图像，**图像编辑**则要在在一幅已有图像上做修改，因而多出一项关键约束：在引入目标改动的同时，尽量保持原图的主体、结构或风格不变。其建模形式可写为

$$p(\text{编辑后图像} \mid \text{原图}, \text{编辑指令}, \text{掩码或结构条件}). \quad (5.13)$$

例如把背景替换为雪山时，模型需改变背景，同时尽量保持人物的面部、姿态与主体结构。图像编辑的一个核心，正是在“目标改动”与“原图保持”之间取得平衡。其一般形式如图 5.28 所示：以原图和某种**编辑条件**为输入，交由生成模型（多为扩散模型）产生结果。

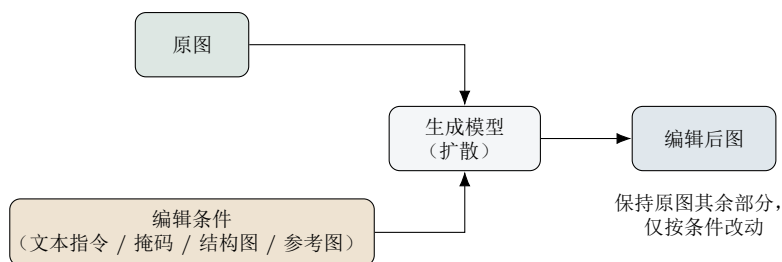


图 5.28: 图像编辑的一般形式：以原图与编辑条件为输入，生成在保持原图其余部分的同时按条件改动的结果。不同代表工作的差异，主要在于“编辑条件”的形式。

对照图 5.28，代表工作的区别主要在于图中“编辑条件”取何种形式，据此可分为以下几类：

- **图像修复与扩展**：在指定的缺失或外延区域生成内容，由掩码约束“仅在此处生成”。
- **文本引导编辑**：以自然语言描述目标，在尽量保留原图的前提下实现该描述，如 SDEdit[?] 与 Prompt-to-Prompt[?]
- **指令式编辑**：直接给出编辑命令（如“将天空改为黄昏”），模型据此修改，如 InstructPix2Pix[?]
- **结构控制与身份保持**：以边缘图、深度图、人体姿态等作为额外约束以固定构图，如 ControlNet[?]; 或借助少量参考图保持特定对象的身份，如 DreamBooth[?]

以结构控制为例：当希望生成图像严格遵循某种构图时，可提供一张**结构条件图**——例如人体姿态骨架、线稿或深度图——与文本提示一起输入模型，模型便在保持该结构的前提下按文本生成内容。给定同一姿态骨架，配以不同文本（“穿红裙的女孩”“穿盔甲的战士”），即可得到姿态一致而外观不同的图像。

重点提示

评价图像编辑一般不能只看新内容是否真实，还需考察三点：是否较准确地实现了目标改动；不应改动之处是否得到保持；局部修改是否与整体较为协调。这三者之间的平衡，是图像编辑较之纯生成更为困难之处。

再以指令式编辑为例：给定一张“人物站在城市街道前”的原图与编辑指令“把背景换成雪山”，模型在保持人物面部、姿态与主体结构不变的前提下，仅改动背景区域；若把指令改为“将白天改为黄昏”，则整体光照随之改变，而主体保持一致。

5.3.4 语音生成：连续信号的条件生成

将式 (5.8) 中的目标 x 取为语音、条件 c 取为文本及说话人、情感等，即得到**语音合成**：

$$p(\text{语音} \mid \text{文本}, \text{说话人}, \text{风格}, \text{情感}). \quad (5.14)$$

它与图像生成的一个根本差异在于：图像多为空间结构的一次性生成，语音则是随时间展开的连续序列生成。较好的语音合成不仅要准确传达文本内容，通常还需生成较自然的音色、韵律、语速与停顿。

文本到语音的基本流程。 典型语音合成系统包含三个部分，如图 5.29 所示。首先，**文本前端**将文字规范化，并转换为字符、音素等语言单元；随后，**声学模型**根据文本预测梅尔频谱或声学 token；最后，**声码器**将这些中间表示还原为可播放的语音波形。

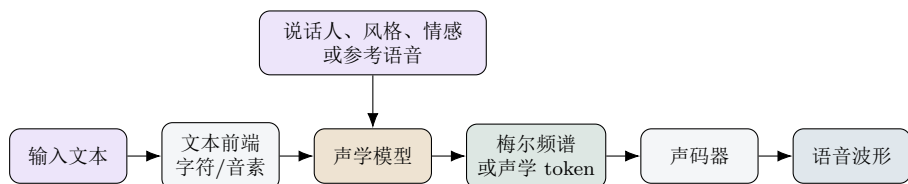


图 5.29: 文本到语音的基本流程。声学模型根据文本和说话人、风格等条件生成声学表示，再由声码器转换为连续语音波形。

举例来说，给定文本“欢迎来到多模态课程”与一段参考语音（指定音色），模型先由文本前端与声学模型生成梅尔频谱，再由声码器还原为语音波形，输出以参考说话人的音色朗读该文本；此时改变情感标签（如“高兴/平静”）或参考语音，便可在内容不变的前提下改变语气与音色，这正体现了语音生成的“一对多”特性。

音色与韵律控制。 同一句文本可以对应多种正确读法，因此语音生成是一种“一对多”任务。说话人条件决定声音像谁，音高、时长和能量共同影响语速、重音、停顿和语气；情感标签则可控制高兴、悲伤或平静等表达方式。模型需要同时满足三项基本要求：内容易懂、声音自然，并符合指定的说话人和表达风格。

零样本合成与声音克隆。 多说话人模型可以从一段参考语音中提取说话人的音色和风格，再使用该声音朗读新的文本。若模型能够为训练中未出现的说话人直接生成语音，

而不需要重新训练，便称为**零样本语音合成**。参考语音提供“用谁的声音说”，输入文本提供“说什么”，二者应尽量相互独立。

连续时间信号的特点。图像生成主要组织二维空间中的像素或图像 token，而语音生成必须沿时间轴保持顺序。一个字或音素通常对应若干连续声学帧，因此模型还需学习文本与语音之间近似单调的时间对应关系。对齐错误可能造成漏词、重复、异常停顿或语速不稳定；生成时间越长，还越难保持音色和韵律的一致。

代表工作。 Tacotron 2[?] 采用“文本到梅尔频谱，再由 WaveNet 声码器生成波形”的两阶段结构；FastSpeech 2[?] 使用非自回归生成，并显式建模时长、音高和能量，提高了速度与可控性。VITS[?] 通过变分推断、流模型和对抗训练，将声学建模与波形生成进行端到端联合训练。VALL-E[?] 则将语音表示为离散神经音频编码 token，像语言模型预测文本 token 一样预测语音 token，并可根据较短的参考语音进行零样本合成。

重点提示

声音克隆的安全与伦理。声音具有明显的个人身份属性，未经许可复制他人声音，可能被用于冒充、虚假信息和诈骗。因此，实际系统应确认声音授权，明确标识合成内容，限制高风险人物或场景的克隆，并结合音频水印、来源记录和伪造语音检测等措施降低滥用风险 [?]。

5.3.5 音频生成：声音事件与声景的条件生成

除语音外，还有更广泛的声音：环境声、音乐片段、声音事件与复杂声景。令模型据文字描述生成此类声音，即一般**音频生成**，建模 $p(\text{音频} | \text{文本描述})$ 。它与语音合成都需生成连续声音信号，但语音有较明确的语言内容与发音规则，一般音频则更强调声音事件、环境层次与时间上的变化，且常有多个声源叠加，建模难度往往更高。

文本到音频的基本流程。 文本到音频生成不是朗读文字，而是根据文字描述还原相应的声音。例如，输入“雨夜的街道上有汽车经过”，模型需要生成雨声、车辆声及相应的空间环境。其基本流程如图 5.30 所示：文字描述先被编码为条件表示，生成模型据此产生频谱、潜在表示或离散音频 token，最后再还原为音频波形。

声音事件与声景生成。 较简单的目标是生成单个**声音事件**，如狗叫、玻璃破碎或汽车鸣笛；更复杂的**声景**则包含多个同时或依次出现的声源。例如，“雨声中，远处有汽车驶过，随后传来雷声”同时规定了声音种类、背景环境和发生顺序。模型不仅要生成每种声音，还要确定它们何时开始、持续多久以及如何相互叠加：先是持续的雨声，中间叠入由远及近的车声，最后才出现雷声。与语音合成不同，这里并没有文字发音，重点在于声音事件的种类、层次与时间结构。

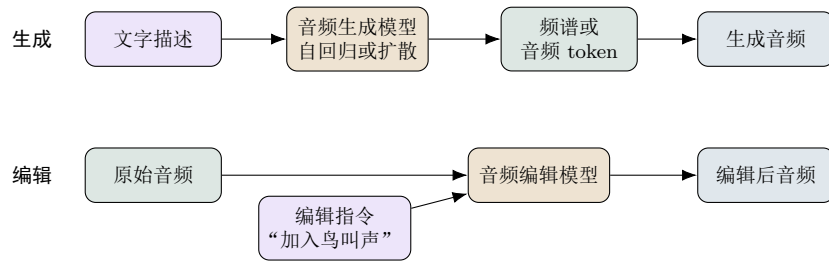


图 5.30: 文本引导的音频生成与编辑。生成模型从文字描述产生新的音频；编辑模型则根据文字指令修改已有音频。

因此，音频生成通常需要满足三个要求：声音内容与文字描述一致，音质具有较强的真实感，并且多个事件在时间上的顺序正确。同一段文字也可能对应许多合理声音，因此模型学习的不是唯一答案，而是符合描述的声音分布。

音频上的扩散建模。 扩散模型训练时逐步向真实音频表示加入噪声，生成时则从随机噪声出发，在文字条件的引导下反复去噪：

$$\mathbf{z}_T \longrightarrow \mathbf{z}_{T-1} \longrightarrow \cdots \longrightarrow \mathbf{z}_0 \longrightarrow \text{音频}. \quad (5.15)$$

为降低计算量，模型通常不直接在高采样率波形上扩散，而是先将音频压缩为频谱或低维潜在表示，在潜在空间完成去噪，再通过解码器和声码器恢复波形。文字表示在每一步去噪中提供条件，使最终声音逐渐接近描述中的事件和环境。

文本引导的音频编辑。 音频生成还可扩展为对已有声音的修改。模型同时接收原始音频和编辑指令，完成“加入雨声”“删除狗叫”“将钢琴替换为吉他”或“修复缺失片段”等操作。编辑时既要改变指定内容，又应尽量保留无关部分，避免整段音频的音色、背景或节奏发生不必要的变化。

代表工作。 AudioGen[?] 将音频量化为离散 token，再使用自回归模型根据文本逐步预测声音；AudioLDM[?] 和 Make-An-Audio[?] 则将扩散模型应用于压缩后的音频表示，提高生成效率与音质。Make-An-Audio 2[?] 进一步强调声音事件的先后顺序和长音频中的时间一致性。AUDIT[?] 则使用“原始音频+编辑指令+目标音频”数据训练潜在扩散模型，可执行声音添加、删除、替换和片段修复等编辑任务。

总体而言，音频生成是上一小节语音合成从“人类说话声”向一般声音的推广。语音合成重点保证文字发音和说话人特征正确，而一般音频生成还需组织多个声源、背景环境与时间变化，生成具有完整声景结构的连续声音。

5.3.6 跨模态交互生成

前述任务多为单一条件、单向生成。更接近实际应用的，是**多条件、多模态、多轮**的交互生成：

$$p(\text{目标模态} \mid \text{文本, 图像, 语音, 历史上下文, 用户指令}). \quad (5.16)$$

例如：依据图像生成语音讲解；依据语音指令修改图像；依据图像与文本共同生成配套音效；或在多轮对话中不断修订生成结果。其主要难点在于不同模态的结构不一致——图像较重空间、语音/音频较重时间、文本较重逻辑——模型需在生成过程中尽量同时保持语义对应、空间一致、时间同步与用户意图一致。这与本讲前两节相呼应：往往需先具备跨模态对齐与多模态理解能力，才能实现较为可控的多模态生成。近年也出现了一些尝试在统一框架内支持“任意模态到任意模态”生成的工作，如 CoDi[?]。

以语音/音频为目标或条件的交互生成。 语音和音频既可以作为模型需要生成的**目标**，也可以作为理解用户意图的**条件**，如图 5.31 所示。

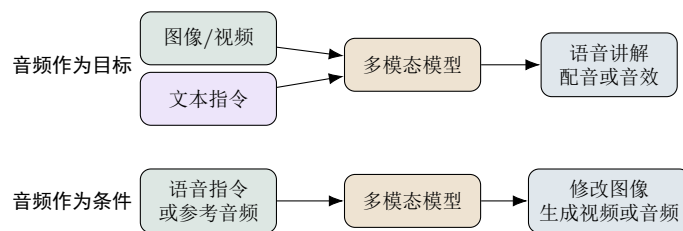


图 5.31: 语音和音频参与交互生成的两种形式：上，依据图像、视频和文字生成语音或音效；下，将语音指令或参考音频作为条件控制其他模态的生成。

视频配音与看图讲解。模型可以根据画面内容和文本要求生成语音讲解。例如，输入一段教学视频，模型需要说明画面中的步骤；输入人物视频时，则可生成与人物动作和场景匹配的配音。此时不仅要保证“说的内容”与画面一致，还要使停顿、语速和音效出现的时间与视频同步。

语音驱动的修改。用户也可以直接用语音提出修改要求，如“把背景音乐调小一点”、“在开门时加入脚步声”或“用更平静的语气重新讲一遍”。模型需要结合当前生成结果和历史指令，只修改用户指定的部分，并尽量保留其他内容。

多轮交互可写为

$$Y_t \sim p(Y_t \mid I, A, Q_{\leq t}, Y_{< t}), \quad (5.17)$$

其中 I 表示图像或视频， A 表示参考语音或音频， $Q_{\leq t}$ 表示截至当前轮的用户指令， $Y_{< t}$ 表示此前生成结果。模型每一轮都需根据新指令修订结果，而不是完全重新生成。

一致性与可控性。这类系统通常需要同时考察以下几方面：

- **语义一致性：**生成的语音或音效是否与图像、视频和文字描述相符；

- **时间同步**：声音事件、动作与字幕是否在正确时间出现；
- **指令遵循**：模型是否只修改用户指定的内容；
- **声音质量**：语音是否自然、易懂，音频是否真实且无明显杂音；
- **跨轮保持**：多轮修改后，人物、音色、场景和未编辑内容是否保持一致。

CoDi[?] 等统一多模态生成模型尝试将文本、图像、视频和音频纳入同一框架，使一个模态或多个模态的组合能够控制另一模态的生成。这类模型的关键不只是“能够生成多种模态”，还在于不同输出之间应描述同一事件，并在空间、时间和语义上保持协调。

重点提示

本节讨论的是较为通用的条件生成能力。要在真实场景中落地，通常还需应对噪声、实时性、安全性与可靠性等更严苛的要求。

5.4 本章小结

本章围绕图像、语音/音频与文本的统一处理，按三个层次构建了多模态机器学习的认知框架。在**表示与对齐**层面，我们说明了各模态在计算机中的原始形式，以及将其编码为向量的动机——用向量空间中的距离刻画语义相似度，使语义相近的对象表示相近；在此基础上，通过自监督预训练获得通用表示，并以对比学习为代表，把不同模态纳入同一空间实现跨模态对齐。

在**结构与理解**层面，我们给出了“模态编码器—连接模块—大语言模型”的通用结构，说明图像、语音如何借此接入大语言模型，构成视觉语言模型与语音语言模型，并通过多模态指令微调与对齐获得按指令处理多模态输入的能力。

在**条件生成**层面，我们把文本生成图像、图像编辑、语音合成与音频生成统一表述为条件分布 $p(\mathbf{x} | \mathbf{c})$ 的学习与采样问题，并以扩散模型为例，说明了如何训练去噪网络、如何从先验噪声逐步采样出图像，以及文本条件如何引导生成。

课后练习

1. 用一句话说明表示学习所追求的目标，并解释为什么希望做到“语义相近的对象，其向量表示也相近”。
2. 一张 224×224 的 RGB 图像，按 16×16 的图块（patch）不重叠地切分，可得到多少个视觉 token？
3. 判断下列说法是否正确并简要说明理由：“视觉语言模型与文本生成图像建模的是同一个条件分布。”

4. 以 CLIP 为例，简述跨模态对齐用什么样的训练目标，把图像与文本纳入同一个向量空间。
5. 某单声道语音的采样率为 16 kHz、时长为 2 秒，问其波形包含多少个采样点？
6. 在基于扩散模型的文本生成图像中，无分类器引导的强度 w 增大，会对生成结果的“文本一致性”和“多样性”分别产生什么影响？
7. 扩散正向过程为 $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}$ 。若 $\bar{\alpha}_t = 0.64$ ，求原图 $\mathbf{x}_0$ 与噪声 $\boldsymbol{\epsilon}$ 前的系数各为多少。
8. 视觉语言模型与语音语言模型在整体结构上有何共同点？二者所处理数据的主要差异又是什么？