第 7 章 深度学习

7.1 引言

第6.5节介绍的通用近似定理指出：对于采用适当激活函数的单隐层前馈神经网络，只要隐含层神经元数量足够多，理论上就能以任意精度逼近任何紧集上的连续函数。然而，该定理仅提供了存在性证明，并未给出如何构造或学习相应网络参数的方法。在实际应用中，核心问题在于如何有效地确定合适的网络参数。如第6.5节所述，含有隐含层的神经网络，其损失函数关于参数通常是非凸的，且普遍存在大量非全局最优的局部极小值点。第6.4节所讨论的反向传播算法利用反向传播计算梯度，再采用梯度优化方法更新参数。由于其局部搜索的特性，它容易陷入局部极小值，并且对参数初始值较为敏感。因此，在 2002 年（对比散度算法被提出）之前，我们面临的困境是：尽管神经网络在理论上具备强大的函数表征能力，但由于其优化问题的非凸性以及反向传播算法本身的局限性，实践中如何获得一组具有良好泛化性能的深层网络参数，当时仍然是一个尚未得到有效解决的难题。

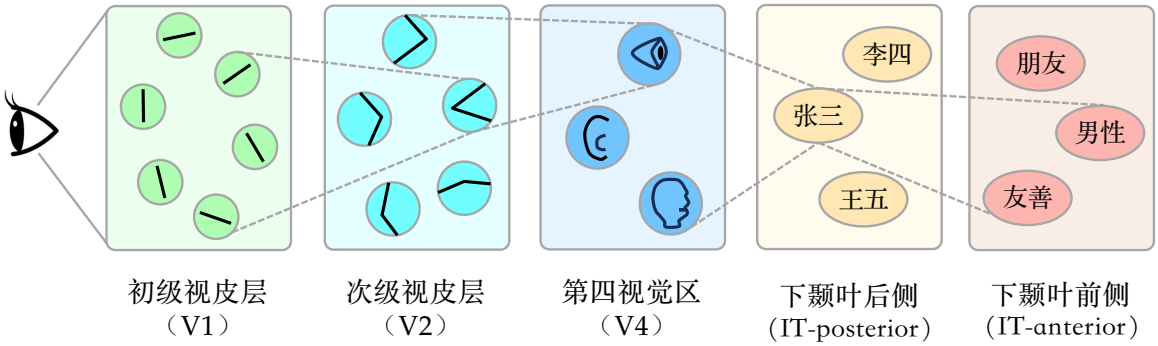


图 7.1: 灵长类动物腹侧视觉通路的层级化组织结构与表征演化。视觉信息沿该通路（从左至右）依次经过初级视皮层（V1）、次级视皮层（V2）、第四视觉区（V4），最终抵达下颞叶皮层（IT）。随着加工层级的升高，神经元的感受野逐渐增大，表征也由低级视觉特征逐步演化为高级语义表征。V1 主要编码局部边缘、方向、线段或条纹等基础视觉特征；V2 通过组合多个 V1 特征，形成更复杂的轮廓与纹理；V4 编码复杂的几何模式与物体形状；在下颞叶后侧，神经元开始形成与个体身份相关的表征（如特定个体的身份识别）；而在下颞叶前侧，表征进一步抽象为更高层次的语义属性信息，如性别、性格与社会关系等。此处描述的是一种总体上的功能变化趋势，并不意味着不同脑区之间存在严格的功能分工，其具体神经机制仍有待进一步研究。

另一方面，通用近似定理被证明同样适用于包含多个隐层的前馈神经网络。随之而来的疑问是：神经网络结构应当向“宽度”还是“深度”方向扩展？即在计算资源（如总参数量）相同的情况下，增加隐层神经元的数量（变宽）与增加网络的层数（变深），哪种方式能更有效地提升模型的表征能力与泛化性能？受灵长类动物腹侧视觉系统（Ventral Visual System）层级化组织结构（如图7.1所示）的启发，研究者们通过理论分析与实验验证，归纳出多层深度结构相较于浅层结构所具备的以下一些优点 [63]：

- **更高效的学习和迁移能力**。当底层神经元学会捕捉大多数物体共享的普适性通用特征（如边缘、方向和纹理等）之后，高层神经元可以通过对这些特征的组合与复用，快速适应并识别新的物体，从而显著提高学习效率与样本利用率。此外，随着网络层数的加深，特征复用的潜在组合能力呈指数级增长，使得高层能够构建出更加丰富且多样化的抽象表示。具体而言，假设第二层基于第一层的 K 个特征组合形成某一中间特征，而第三层再基于第二层的 K 个特征进一步组合出更高层的抽象特征，则第三层对第一层原始特征的有效组合规模可达 K^2。
- **缓解连接爆炸问题**。考虑一个仅包含两层的网络，若要求同一物体在视网膜任意位置出现时，均能激活第二层中的同一神经元，则该神经元必须与第一层中所有可能参与该物体表征的神经元建立连接，从而导致连接数量随特征维度、物体种类及出现位置的组合呈指数级增长。相比之下，多层结构通过将复杂识别任务分解为多个层级阶段，使得每一层神经元仅需响应前一层局部区域内神经元活动的特定模式。随着层数加深，神经元的感受野逐渐扩大，实现从局部到全局的逐级整合。这种机制在减少单个神经元输入连接规模的同时，仍能够完成从整个输入空间（如视网膜）到高层语义空间（如下颞叶前部）的复杂映射。
- **三维物体识别与视角泛化**。层级视觉系统能够通过对三维物体的有限个二维视图的学习，泛化到该物体在其他视角下的二维呈现。这一能力源于：同一物体在不同视角下所形成的图像并非相互独立，而是在结构上具有内在关联。具体而言，不同视角下的二维图像通常共享大量局部特征与结构，仅在位置、尺度、旋转以及透视变换等方面存在变化。因此，无论观察角度如何改变，物体往往仍保留相对稳定的局部特征（如边缘、角点、纹理片段及基本轮廓）。层级视觉系统首先由底层提取这些局部特征，而高层神经元通过对其进行逐级组合与整合，将来自不同视角的输入归入同一三维物体的统一表征。当系统已学习若干典型的二维视图后，新的视图只要仍位于该物体已知视图的变化范围内，即可以因共享特征而被映射到相邻表示，从而正确识别为同一物体。
- **无需显式分割的端到端识别**。学习完成后，系统的行为类似于一种关联记忆，即将输入图像的特征表示直接映射到语义信息。低层负责提取边缘、纹理等局部特征，高层通过学习到的统计规律对这些特征进行整合与抽象。整个识别过程基于局部特征的累积匹配与竞争机制，即当物体的某个关键特征在视野中出现时，对应高层神经元被激活的可能性随之提高。因此，该过程并不遵循“先分割、后识别”的工作模式，而是通过端到端的方式完成。系统通过将高维输入投影至习得的特征空间，并利用高层神经元之间的竞争来选择最可能的识别结果。这种机制使得系统在面对遮挡时，仍能基于部分可见特征做出合理推断。
- **计算高效与低能耗**。在总体神经元数量相同的情况下，多层深度结构通过特征复用，使得每一层的宽度（神经元数量）显著小于实现相同性能的浅层网络。每层通常仅需激活部分神经元参与计算，从而形成稀疏的响应模式。在保持表达能力的同时，这种分层结构不仅提高每层网络的计算速度，也契合生物视觉系统在感知过程中对低能耗的要求。

从机器学习的角度来看，分层架构还显著降低了每层网络的训练难度。其原因在于，每一层神经元只需学习如何从前一层有限数量的特征中进行有效组合，以提取对最终识别任务有用

的表征，从而避免在更大规模的原始输入空间中进行搜索。接下来，我们先介绍杰弗里·辛顿所提出的对比散度算法，并探讨如何利用该算法对深度置信网络进行逐层预训练。正是这一突破性的方法，为训练更深的神经网络奠定了基础。

7.2 深度置信网络

如前文所述，**深度置信网络**（Deep Belief Network）通常采用逐层预训练的方式进行构建，其中每一层的基本组件为**受限玻尔兹曼机**（Restricted Boltzmann Machine）。受限玻尔兹曼机的结构如图7.2所示，之所以称为“受限”，是因为与一般的玻尔兹曼机相比，其同一层内的神经元之间不允许存在连接，从而形成了一种典型的二分图结构。受限玻尔兹曼机由可见层（底层）和隐藏层（上层）组成。与霍普菲尔德网络相似，基础的受限玻尔兹曼机属于离散模型，其所有神经元通常取二值状态（即 0 或 1）。底层之所以被称为“可见层”（Visible Layer），这是因为该层直接对应输入样本，各神经元的状态可由观测数据直接给定，因此是“可见”的。受限玻尔兹曼机可以视为对霍普菲尔德网络施加结构性约束后所得到的一种特例：即将神经元划分为两组（可见层与隐藏层），并去除同一组内神经元之间的连接，同时将其中一组（可见层）与输入样本相对应，用于表示观测数据。隐藏层则用于学习输入数据背后的潜在表示与统计依赖关系。

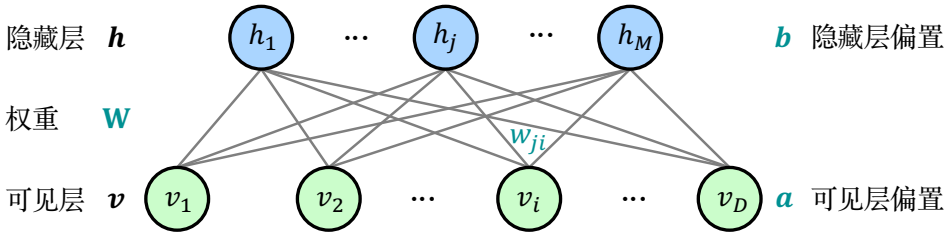


图 7.2: 受限玻尔兹曼机结构示意图。该模型可表示为一个二分图（Bipartite Graph），由可见层（底层）和隐藏层（上层）组成。两层之间通过无向连接相互作用，而同一层内的神经元之间则不存在连接（即“受限”的含义）。此外，每个神经元均带有一个偏置参数。基础的受限玻尔兹曼机的可见层与隐藏层的神经元状态通常取二值形式（即 0 或 1）。

与霍普菲尔德网络的能量函数（详见公式6.65）类似，我们可以为受限玻尔兹曼机定义如下能量函数：

$$E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) = -\boldsymbol{v}^\top \mathbf{W}\boldsymbol{h} - \boldsymbol{a}^\top \boldsymbol{v} - \boldsymbol{b}^\top \boldsymbol{h} \tag{7.1}$$

$$= -\sum_{i=1}^{D}\sum_{j=1}^{M} w_{ji} v_i h_j - \sum_{i=1}^{D} a_i v_i - \sum_{j=1}^{M} b_j h_j \tag{7.2}$$

其中 $\boldsymbol{v} \in \{0,1\}^D$ 表示可见层神经元的状态向量，$\boldsymbol{h} \in \{0,1\}^M$ 表示隐藏层神经元的状态向量，而 $\boldsymbol{\theta} = \{\mathbf{W}, \boldsymbol{a}, \boldsymbol{b}\}$ 为模型参数集合。在参数中，w_{ji} 为可见层第 i 个神经元与隐藏层第 j 个神经元之间的对称连接权重，a_i 为可见层第 i 个神经元的偏置，b_j 则为隐藏层第 j 个神经元的偏置。

接下来，我们将基于上述能量函数来定义受限玻尔兹曼机处于状态 $\{\boldsymbol{v},\boldsymbol{h}\}$ 的联合概率分布。

在能量模型中，这一概率定义遵循统计力学中的玻尔兹曼分布（Boltzmann Distribution）。玻尔兹曼分布将系统处于某一状态的概率表示为该状态的能量以及系统温度的函数，其表达式如下：

$$p_s = \frac{\exp\left(-\frac{E_s}{\kappa_B \cdot T}\right)}{\sum_{k=1}^{K} \exp\left(-\frac{E_k}{\kappa_B \cdot T}\right)} \tag{7.3}$$

其中 p_s 表示系统处于状态 s 的概率，E_s 为系统处于状态 s 时的能量，κ_B 表示玻尔兹曼常数（Boltzmann Constant），T 为系统的绝对温度，而 K 则表示系统所有可能状态的总数。在统计物理学中，乘积 $\kappa_B \cdot T$ 代表热能尺度（Thermal Energy Scale），用于衡量系统内微观粒子（如原子或分子）因热运动所具有的典型能量量级。在机器学习建模中，通常简化并将该项设定为常数，即令 $\kappa_B T = 1$。因此，受限玻尔兹曼机处于状态 $\{\boldsymbol{v}, \boldsymbol{h}\}$ 的联合概率分布定义如下：

$$p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta}) = \frac{\exp(-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}))}{\mathcal{Z}(\boldsymbol{\theta})} \tag{7.4}$$

$$\mathcal{Z}(\boldsymbol{\theta}) = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta})) \tag{7.5}$$

其中 $\mathcal{Z}(\boldsymbol{\theta})$ 被称为配分函数（Partition Function）。在概率模型中，它作为归一化因子，用于确保所有可能状态的概率之和为 1。如公式7.4所示的概率分布形式，可以做如下直观解释。在讨论霍普菲尔德网络时，我们已知系统状态对应的能量越高，其稳定性越差。因此，在动力学演化过程中，系统往往会从高能量状态向低能量状态“下落”，并最终收敛到某个稳定状态（即局部极小值）。因此，在概率建模中，高能量状态由于其不稳定性，出现的概率较低；而低能量状态则更为稳定，对应的发生概率较高。状态的发生概率与其能量呈“负相关”关系，这也解释了在指数函数中对能量项引入负号的原因。此外，为了保证概率的非负性以及归一化条件（即所有状态概率之和为 1），需要对能量的相反数取指数（确保非负性），并除以所有可能状态对应的指数和（即配分函数），从而构成一个合法的概率分布。

我们的目标是优化受限玻尔兹曼机的参数 $\boldsymbol{\theta}$，使得模型生成实际观测数据（即输入样本）的概率最大化。从能量模型的角度来看，这相当于将实际观测到的数据映射为系统能量较低的状态。为了计算观测数据的似然函数，我们需要对隐藏层状态进行边缘化处理。通过对所有可能的隐藏层神经元状态向量 $\boldsymbol{h}$ 进行求和，我们可以将 $\boldsymbol{h}$ 从联合概率分布（公式7.4）中除去，从而得到仅关于可见层神经元状态向量 $\boldsymbol{v}$ 的边缘分布：

$$\begin{aligned}
p(\boldsymbol{v}|\boldsymbol{\theta}) &= \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta})) \\
&= \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \sum_{\boldsymbol{h}} \exp\left(\boldsymbol{v}^\top \mathbf{W}\boldsymbol{h} + \boldsymbol{a}^\top \boldsymbol{v} + \boldsymbol{b}^\top \boldsymbol{h}\right) \\
&= \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \exp\left(\boldsymbol{a}^\top \boldsymbol{v}\right) \prod_{j=1}^{M} \sum_{h_j \in \{0,1\}} \exp\left(\sum_{i=1}^{D} w_{ij} v_i h_j + b_j h_j\right) \quad (e^{(x+y)} = e^x \cdot e^y) \\
&= \frac{1}{\mathcal{Z}(\boldsymbol{\theta})} \exp\left(\boldsymbol{a}^\top \boldsymbol{v}\right) \prod_{j=1}^{M} \left(1 + \exp\left(\sum_{i=1}^{D} w_{ij} v_i + b_j\right)\right) \quad (\text{将 } h_j \text{ 取 0 和 1 代入})
\end{aligned} \tag{7.6}$$

为了确定如何调整参数 $\boldsymbol{\theta}$ 以最大化公式7.6所示的似然函数，我们需要计算对数似然函数关于参数 $\boldsymbol{\theta}$ 的梯度：

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \ln p(\boldsymbol{v}|\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \left(\ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})} \right) - \nabla_{\boldsymbol{\theta}} \left(\ln \mathcal{Z}(\boldsymbol{\theta}) \right) \\
&= \nabla_{\boldsymbol{\theta}} \left(\ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})} \right) - \nabla_{\boldsymbol{\theta}} \left(\ln \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})} \right) \\
&= -\frac{1}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \\
&\quad + \frac{1}{\sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}} \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \\
&= -\sum_{\boldsymbol{h}} \frac{e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \\
&\quad + \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \frac{e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}}{\sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \\
&= -\sum_{\boldsymbol{h}} \frac{\frac{e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}}{\mathcal{Z}(\boldsymbol{\theta})}}{\sum_{\boldsymbol{h}} \frac{e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}}{\mathcal{Z}(\boldsymbol{\theta})}} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \quad \text{（分子和分母同时除以 } \mathcal{Z}(\boldsymbol{\theta})\text{）} \\
&\quad + \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \frac{e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}}{\mathcal{Z}(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \quad \left(\mathcal{Z}(\boldsymbol{\theta}) = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}\right) \\
&= -\sum_{\boldsymbol{h}} \frac{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}{p(\boldsymbol{v}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) + \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) \quad \text{（}p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})\text{ 定义）} \\
&= \underbrace{-\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}_{\text{通过分布 } p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta}) \text{ 对梯度求期望}} + \underbrace{\sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})}_{\text{通过分布 } p(\boldsymbol{h},\boldsymbol{v}|\boldsymbol{\theta}) \text{ 对梯度求期望}} \quad \left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})} = p(\boldsymbol{h}|\boldsymbol{v})\right) \\
&= -\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) + \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta})
\end{aligned} \tag{7.7}
$$

结合公式7.1定义的能量函数，对数似然关于各参数的梯度可以表示为：

$$\nabla_{\mathbf{W}} \ln p(\boldsymbol{v}|\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})}[\boldsymbol{v}\boldsymbol{h}^{\top}] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}[\boldsymbol{v}\boldsymbol{h}^{\top}] \tag{7.8}$$

$$\nabla_{\boldsymbol{a}} \ln p(\boldsymbol{v}|\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})}[\boldsymbol{v}] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}[\boldsymbol{v}] \tag{7.9}$$

$$\nabla_{\boldsymbol{b}} \ln p(\boldsymbol{v}|\boldsymbol{\theta}) = \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})}[\boldsymbol{h}] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}[\boldsymbol{h}] \tag{7.10}$$

从以上三个梯度公式可以看出，各参数的优化方向具有统一的结构，即条件分布 $p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})$ 下的期望减去联合分布 $p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})$ 下的期望。具体而言，第一项被称为正相，亦称数据相关项。它是在给定观测数据条件下的期望，旨在驱动模型向实际观测样本靠近，从而降低观测数据点处的能量；第二项则被称为负相，亦称模型相关项。其作用是抑制模型对非真实样本（即模型生成的“伪数据”）赋予过高的概率。通过提高这些区域的能量，使模型分布逐步逼近真实数据分布。在获得对数似然函数关于参数的梯度后，可以利用梯度方向对参数进行迭代优化。以对称权重

矩阵 $\mathbf{W}$ 为例，其更新量 $\Delta\mathbf{W}$ 的计算公式如下（偏置参数 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的更新以此类推）：

$$\Delta\mathbf{W} = \eta\left(\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})}[\boldsymbol{v}\boldsymbol{h}^\top] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}[\boldsymbol{v}\boldsymbol{h}^\top]\right) \tag{7.11}$$

其中 η 是学习率。在给定观测数据的情况下，条件分布下的期望（即数据相关项）较易求得，而联合分布下的期望（即模型相关项）由于涉及配分函数 $\mathcal{Z}(\boldsymbol{\theta})$，在实际计算中通常是不可行的（Intractable）。由于可见层和隐藏层神经元均取二值状态，通过直接枚举准确计算该期望需要对 2^{D+M}（其中 D 和 M 分别为可见层与隐藏层的神经元个数）种可能的状态进行遍历，具有指数级的计算复杂性。为了解决这一难题，我们需要引入高效的近似估计方法。由杰弗里 · 辛顿所提出的对比散度算法仅需少量的采样步数即可有效地对该期望值进行近似估计（虽然是有偏的）。在详细介绍对比散度算法之前，我们先讨论如何计算数据相关项中的条件概率。

根据条件概率公式，在给定可见层 $\boldsymbol{v}$ 的条件下，隐藏层 $\boldsymbol{h}$ 的概率分布为（为简化表示，推导中省略参数 $\boldsymbol{\theta}$）：

$$\begin{aligned}
p(\boldsymbol{h}|\boldsymbol{v}) &= \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})} \\
&= \frac{\frac{1}{\mathcal{Z}}\exp(-E(\boldsymbol{v},\boldsymbol{h}))}{\frac{1}{\mathcal{Z}}\sum_{\boldsymbol{h}}\exp(-E(\boldsymbol{v},\boldsymbol{h}))} \quad \text{（基于能量函数的分布定义）} \\
&= \frac{\exp\left(\boldsymbol{a}^\top\boldsymbol{v}\right)\prod_{j=1}^{M}\exp\left(\sum_{i=1}^{D}w_{ij}v_ih_j + b_jh_j\right)}{\exp\left(\boldsymbol{a}^\top\boldsymbol{v}\right)\prod_{j=1}^{M}\sum_{h_j\in\{0,1\}}\exp\left(\sum_{i=1}^{D}w_{ij}v_ih_j + b_jh_j\right)} \quad \text{（能量函数的定义代入）} \\
&= \prod_{j=1}^{M}\frac{\exp\left(h_j\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)\right)}{\sum_{h_j\in\{0,1\}}\exp\left(h_j\left(\sum_{i=1}^{D}w_{ij}v_i + b_j\right)\right)} \\
&= \prod_{j=1}^{M}\frac{\left(\exp\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)\right)^{h_j}}{\left(1+\exp\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)\right)^{h_j}\left(1+\exp\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)\right)^{1-h_j}} \\
&= \prod_{j=1}^{M}\left(\frac{\exp\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)}{1+\exp\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)}\right)^{h_j}\left(\frac{1}{1+\exp\left(b_j + \sum_{i=1}^{D}w_{ij}v_i\right)}\right)^{1-h_j} \\
&= \prod_{j=1}^{M}\sigma\left(\sum_{i=1}^{D}w_{ij}v_i + b_j\right)^{h_j}\left(1-\sigma\left(\sum_{i=1}^{D}w_{ij}v_i + b_j\right)\right)^{1-h_j}
\end{aligned} \tag{7.12}$$

（其中 $\sigma(\cdot)$ 为 Sigmoid 函数）

从公式7.12可以看出，在给定可见层状态时，隐藏层各神经元的状态是相互独立的（表现为联合概率可分解为各分量概率的乘积）。因此，单个隐藏层神经元 h_j 取值为 1 的条件概率可表示为：

$$p(h_j = 1|\boldsymbol{v}) = \sigma\left(\sum_{i=1}^{D}w_{ij}v_i + b_j\right) \tag{7.13}$$

同理，由于受限玻尔兹曼机的结构对称性，在给定隐藏层状态时，可见层神经元 v_i 取值为 1 的

条件概率同样具有如下 Sigmoid 函数形式：

$$p(v_i = 1|\boldsymbol{h}) = \sigma\left(\sum_{j=1}^{M} w_{ij}h_j + a_i\right) \tag{7.14}$$

从上述推导可知，数据相关项中的条件概率可由公式7.12直接计算得到。对于模型相关项，辛顿提出采用吉布斯采样（Gibbs Sampling）方法来近似估算联合分布 $p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta})$。吉布斯采样（将在第11章中详细介绍）的基本思想是：当直接从复杂的联合分布 $p(\boldsymbol{v}, \boldsymbol{h})$ 中采样较为困难，而各个变量的条件分布易于采样时，我们可以每次只更新一个变量（其余变量固定），并且依据相应的条件分布进行采样。具体而言，可以先给定 $\boldsymbol{v}$ 的条件下，根据 $p(\boldsymbol{h}|\boldsymbol{v})$ 采样得到 $\boldsymbol{h}$，再在给定新采样得到的 $\boldsymbol{h}$ 条件下，根据 $p(\boldsymbol{v}|\boldsymbol{h})$ 重新采样 $\boldsymbol{v}$，并不断迭代这一过程。对于受限玻尔兹曼机，首先基于给定样本，通过公式7.13采样 $\boldsymbol{h}$，随后在给定 $\boldsymbol{h}$ 的条件下，利用公式7.14采样 $\boldsymbol{v}$，并进行 K 轮交替采样，以近似联合分布 $p(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta})$。当迭代次数为 K 时，则对应的学习算法被称为 CD-K 算法，其中 CD 表示对比散度（Contrastive Divergence）。在实际应用中，CD-1（即仅迭代采样一次）通常就能取得良好的训练效果。对比散度中的“散度”是指 KL 散度（Kullback-Leibler Divergence），通常用于衡量两个概率分布之间的差异。而“对比散度”名称的由来则在于其核心思想：通过比较观测数据分布与模型分布之间的 KL 散度，以及经过 K 步采样得到的重构分布与模型分布之间的 KL 散度。在实际计算中，算法并不直接计算这两个 KL 散度，而是通过对比观测数据与重构样本的相关统计量（如公式7.7所示），并以此计算近似梯度来更新模型参数。

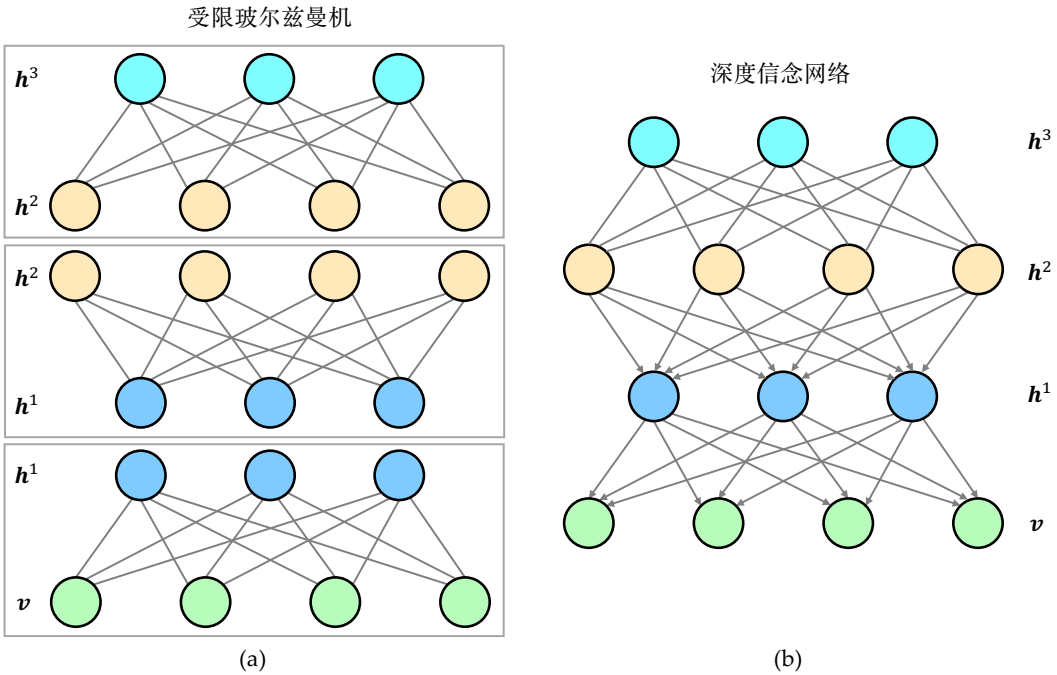


图 7.3: 由受限玻尔兹曼机构建深度置信网络示意图。(a) 逐层训练的受限玻尔兹曼机结构。各层依次独立训练，其中上层受限玻尔兹曼机以下一层受限玻尔兹曼机提取的隐层特征（即隐藏层状态向量）作为训练数据。(b) 由多个已训练好的受限玻尔兹曼机堆叠而成的深度置信网络。

利用上述对比散度算法完成训练后，受限玻尔兹曼机能够从可见层（输入样本）中提取出隐层特征表示（即隐藏层状态向量），并能根据该特征表示又重构出原始输入。如图7.3(a) 所示，该训练过程可以通过堆叠的方式多次重复。当下层受限玻尔兹曼机训练完成后，其提取的隐层特征表示将作为上一层受限玻尔兹曼机的训练样本。这种贪婪逐层训练的机制，使得模型能够逐层构建更加抽象的特征表示。当多层受限玻尔兹曼机依次训练完成后，即可将其按顺序堆叠，构建出如图7.3(b) 所示的深度置信网络。

值得注意的是，受限玻尔兹曼机的层间连接构成了一个无向图，这体现了其对称性：模型既能根据可见层输入计算隐藏层，也能将隐藏层作为输入来重构可见层。相比之下，深度置信网络则是一种混合概率模型。除了最顶部的两层之间保持无向连接外，其余层之间的连接均是有向的。这种结构源自深度置信网络的联合概率分布的分解形式。以图7.3(b) 为例，其联合分布可表示为：

$$p(\boldsymbol{v}, \boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3|\boldsymbol{\theta}) = p(\boldsymbol{v}|\boldsymbol{h}^1; \boldsymbol{\theta})p(\boldsymbol{h}^1|\boldsymbol{h}^2; \boldsymbol{\theta})p(\boldsymbol{h}^2, \boldsymbol{h}^3|\boldsymbol{\theta}) \tag{7.15}$$

在上式中，等号右侧的前两个因子为条件概率分布，在图中对应有向连接（由高层指向低层的生成过程）；而最后一项则代表顶部两层所构成的联合分布，在图中对应无向连接。

深度置信网络的逐层预训练流程如算法7.1所示。在该算法中，我们以重构误差是否降低至预设阈值作为判断当前受限玻尔兹曼机训练是否充分。一般而言，随着训练的进行，重构误差通常呈下降趋势，并逐渐趋于稳定，但并不保证单调下降或达到预设阈值。需要注意的是，重构误差较小并不意味着模型分布已经充分接近数据分布。当采样链混合较慢时，即使模型尚未学到理想的数据分布，经过少量采样步骤得到的重构结果也可能与原始输入十分接近，从而产生较小的重构误差。因此，在实际训练中，通常会设置最大训练轮数或迭代次数，并将重构误差作为辅助监测指标，而不是判断模型训练是否充分的唯一依据。所谓“无监督”是指模型在训练过程中仅依赖未标注的原始数据（预训练阶段不需要人工标注的样本），因此在理论上可以利用大量的领域样本进行学习。预训练的核心作用是将网络参数引导至一个较好的初始值空间，使初始化后的网络能够捕捉并生成目标领域的数据分布。在实际应用中，通常还需结合具体任务，对预训练后的深度置信网络进行有监督微调。例如：对于分类任务，通常在最顶层添加一个 Softmax 层；对于回归任务，则添加一个线性映射层。随后，利用带标签的数据，通过反向传播算法对整个网络参数进行联合优化，从而提升模型在特定任务上的性能。这种“无监督预训练 + 有监督微调”的训练范式有效缓解了深层神经网络在直接采用反向传播训练时对参数初始化较为敏感的问题，并且推动了深度学习早期的发展，形成了经典的“两阶段训练”方法。

值得注意的是，在执行特定任务时，深度置信网络的推理是从底层向高层方向进行，这在直观上似乎与图7.3(b) 中所示的箭头方向相反。其原因在于，深度置信网络在预训练阶段所建模的是数据的生成过程，即关注“模型如何产生这些数据”。而当我们利用预训练好的模型执行分类或回归任务时，实质上是在进行一种“逆向过程”，即根据输入的数据去推测其最可能的潜在原因（如类别标签或拟合目标）。因此，模型的生成方向是自顶向下，而贪婪逐层预训练通常按自底向上的顺序进行，推理阶段则也是自底向上的判别过程。

算法 7.1: 深度置信网络逐层预训练算法

输入: 训练样本集 $\mathcal{D}=\{\boldsymbol{x}_i\}_{i=1}^N$、网络层数 L、学习率 η、重构误差阈值 ϵ、批量大小 B 和吉布斯采样迭代次数 K。

初始化: 随机初始化各层权重与偏置参数 $\boldsymbol{\theta}$（其中偏置的初始值可设为 0）;
for $l \leftarrow 1$ **to** L **do**
  if $l=1$ **then**
    训练集为 $\mathcal{D}$;
  else
    训练集为 $l-1$ 层提取的隐层特征表示;
  重构误差初始化: $r \leftarrow \infty$;
  while $\frac{1}{B}r \geq \epsilon$ **do**
    梯度清零: $\mathbf{g} \leftarrow 0$;
    重构误差清零: $r \leftarrow 0$;
    从训练集中随机采样 B 个样本;
    for $i \leftarrow 1$ **to** B **do**
      使用公式7.12根据第 i 个训练样本可见层输入 $\boldsymbol{v}_i^{(0)}$ 计算隐藏层表示 $\boldsymbol{h}_i^{(0)}$;
      for $k \leftarrow 1$ **to** K **do**
        给定 $\boldsymbol{h}_i^{(k-1)}$ 利用条件分布 $p(\boldsymbol{v}|\boldsymbol{h};\boldsymbol{\theta})$ 采样得到 $\boldsymbol{v}_i^{(k)}$（使用公式7.14）;
        给定 $\boldsymbol{v}_i^{(k)}$ 利用条件分布 $p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})$ 采样得到 $\boldsymbol{h}_i^{(k)}$（使用公式7.13）;
      根据公式7.7计算对数似然关于各参数的梯度 $\nabla_{\boldsymbol{\theta}} \ln p(\boldsymbol{v}|\boldsymbol{\theta})$，其中数据相关项通过 $\boldsymbol{v}_i^{(0)}$ 和 $\boldsymbol{h}_i^{(0)}$ 计算，而模型相关项通过 $\boldsymbol{v}_i^{(K)}$ 和 $\boldsymbol{h}_i^{(K)}$ 计算;
      梯度累积: $\mathbf{g} \leftarrow \mathbf{g} + \nabla_{\boldsymbol{\theta}} \ln p(\boldsymbol{v}|\boldsymbol{\theta})$;
      重构误差累积: $r \leftarrow r + \|\boldsymbol{v}_i^{(0)} - \boldsymbol{v}_i^{(K)}\|^2$;
    更新参数: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\frac{1}{B} \cdot \mathbf{g}$;

输出: 预训练后的深度置信网络各层参数 $\boldsymbol{\theta}$。

在实际应用中，输入样本通常并非离散的二值形式，而是以实数值数据为主（如图像数据）。因此，需要将受限玻尔兹曼机扩展到能够处理实值输入的情况。这里介绍一种代表性的扩展，即高斯受限玻尔兹曼机（Gaussian Restricted Boltzmann Machine），其能量函数定义如下:

$$E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}) = -\sum_{i=1}^{D}\sum_{j=1}^{M} w_{ij}\frac{v_i}{\tau_i}h_j + \sum_{i=1}^{D}\frac{(v_i-a_i)^2}{2\tau_i^2} - \sum_{j=1}^{M} b_j h_j \tag{7.16}$$

其中 $\boldsymbol{v} \in \mathbb{R}^D$ 表示可见层神经元的状态向量，$\boldsymbol{h} \in \{0,1\}^M$ 表示隐藏层神经元的状态向量，而 $\boldsymbol{\theta}=\{\mathbf{W},\boldsymbol{a},\boldsymbol{b},\boldsymbol{\tau}\}$ 为模型参数集合。可见层状态向量 $\boldsymbol{v}$ 的边缘分布（即似然函数）的形式为:

$$p(\boldsymbol{v}|\boldsymbol{\theta}) = \sum_{\boldsymbol{h}} \frac{\exp(-E(\boldsymbol{v},\boldsymbol{h};\boldsymbol{\theta}))}{\int_{\boldsymbol{v}'}\sum_{\boldsymbol{h}}\exp\left(-E(\boldsymbol{v}',\boldsymbol{h};\boldsymbol{\theta})\right)d\boldsymbol{v}'} \tag{7.17}$$

根据能量函数公式7.16，可以推导出以下条件概率分布：

$$p(h_j = 1|\boldsymbol{v}) = \sigma\left(\sum_{i=1}^{D} w_{ij}\frac{v_i}{\tau_i} + b_j\right) \tag{7.18}$$

$$p(v_i = x|\boldsymbol{h}) = \frac{1}{\sqrt{2\pi}\tau_i}\exp\left(-\frac{\left(x - a_i - \tau_i\sum_{j=1}^{M} h_j w_{ij}\right)^2}{2\tau_i^2}\right) \tag{7.19}$$

其中 $\sigma(\cdot)$ 是 Sigmoid 函数。由公式7.19可知，在给定隐藏层状态的条件下，每个可见层神经元均服从一维高斯分布，其方差由 τ_i^2 控制，而均值则由隐藏层神经元激活值的加权组合确定（即在偏置 a_i 的基础上产生相应的偏移）。对数似然函数关于权重 w_{ij} 的偏导数为：

$$\frac{\partial \ln p(\boldsymbol{v}|\boldsymbol{\theta})}{\partial w_{ij}} = \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v};\boldsymbol{\theta})}\left[\frac{1}{\tau_i}v_i h_j\right] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{\theta})}\left[\frac{1}{\tau_i}v_i h_j\right] \tag{7.20}$$

高斯受限玻尔兹曼机仍可以采用对比散度算法进行训练。在优化过程中，参数 $\boldsymbol{\tau}$ 既可以作为可调参数与其他参数一起进行优化，也可以在初始化后保持固定不变。

随着深度学习技术的发展，尤其是可用训练数据规模的显著增长，以及如 ReLU 激活函数（详见第7.3节）和残差连接（详见第7.5节）等有效缓解梯度消失问题的方法不断被提出与完善，目前在训练深层模型时已较少采用上述逐层预训练的方法。然而，在算力资源匮乏、训练算法尚不成熟的年代，深度置信网络所采取的“无监督预训练 + 有监督微调”策略及其在各项任务上相较传统机器学习方法所取得的显著性能提升，成为人工神经网络在 2006 年前后重新受到广泛关注的重要转折点。

7.3 卷积神经网络

在数字信号处理中，通常将一个系统视为从输入信号 $x[n]$ 到输出信号 $y[n]$ 的映射，其中 n 表示离散时间（时刻）。当系统满足**线性时不变**（Linear Time-invariant）性质，则对于任意输入信号的输出均可表示为如下**离散卷积**（Discrete Convolution）形式：

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m] \tag{7.21}$$

其中 $h[n]$ 称为系统的单位脉冲响应（Unit Impulse Response），也常被称为卷积核（kernel）。它本质上是一个与 $x[n]$ 类似的离散序列，用于描述系统对不同时刻输入的响应强度。因此，公式7.21表示：系统在时刻 n 的输出是所有历史时刻（也可能包括未来时刻）输入信号的加权和。由于该映射具有线性特征，且系统的行为不随时间变化（即在任何时刻 n，系统均以相同的加权方式产生输出），故称之为线性时不变系统。简而言之，任意满足线性性与时不变性的系统，其输出都可以表示为输入信号与其单位脉冲响应的卷积。值得注意的是，这里所提到的“核（Kernel）”与第4章核方法中的核函数（Kernel Function）并非同一概念。本节中的“核”通常指一组用于加权求和的系数，其本质是对不同时刻输入信号贡献的权重分配；而核方法中的核函数则用于度量两个样本（通常表示为特征向量）之间的相似性。

如果将上述 $x[n]$ 和 $h[n]$ 视为以离散时间步为自变量的函数，则公式7.21通常可以写为：

$$(x * h)[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m] \tag{7.22}$$

其中 $*$ 表示卷积运算符。若将上述定义推广到连续的情形，则函数 $f(t)$ 和 $g(t)$ 的卷积定义为：

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau \tag{7.23}$$

图7.4展示了同一红色矩形脉冲函数分别与另一相同形状的矩形脉冲函数及一个指数衰减函数进行卷积的结果。卷积操作不仅可以对输入信号（蓝色）起到平滑与延迟的作用，还可以通过设计不同形式的 $g(t-\tau)$ 来从输入信号中提取特定的模式或特征。如图7.4(a) 所示，当输入信号中出现与 $g(t-\tau)$ 形态相匹配的信号时（即两者重叠程度最大），卷积值达到最大，从而在 $t=0$ 处产生最强的响应输出。这一性质也是后续将介绍的卷积神经网络（Convolutional Neural Network）能够利用多个可学习的卷积核来识别各种局部模式或特征（如边缘、纹理等）的基本原理。

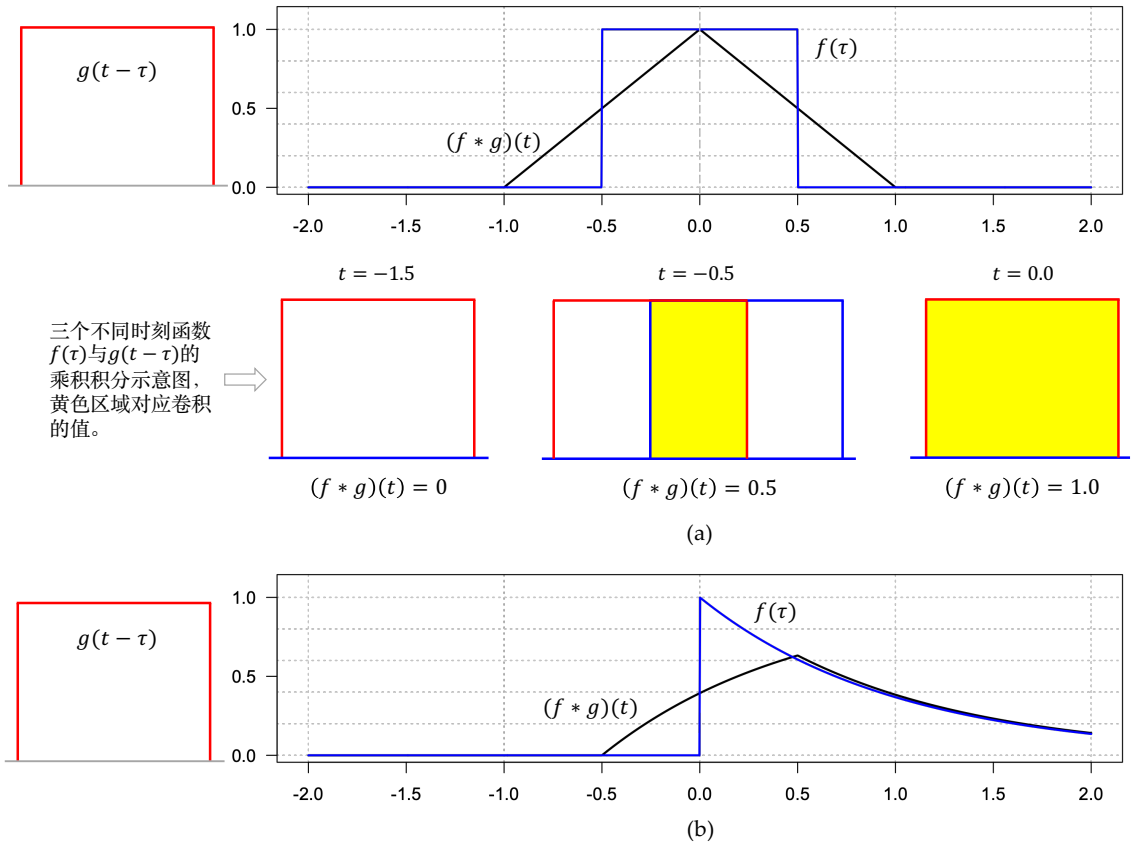


图 7.4: 连续卷积示意图。(a) 函数 $f(\tau)$（蓝色）与函数 $g(t-\tau)$（红色）的卷积结果 $(f*g)(t)$（黑色）。随着红色矩形脉冲沿横轴从左向右滑动，每一时刻 t 的卷积值对应于两个函数乘积的积分（即重叠区域的面积）。当红色函数与蓝色函数的重叠程度最大时（即 $t=0$），卷积值达到最大值。(b) 使用相同的红色矩形脉冲函数 $g(t-\tau)$ 与指数衰减函数 $f(\tau)$ 进行卷积得到的结果（黑色曲线）。输出曲线体现了系统对输入信号（蓝色）的平滑和延时效应。

在信号处理中，卷积运算通常需要先对卷积核进行时间翻转（源自 $g(t-\tau)$ 中变量 τ 前面的负号），随后再与输入信号进行滑动加权求和。这种翻转操作保证了卷积运算满足交换律，即 $(f*g)(t)=(g*f)(t)$。而在卷积神经网络的实现中，通常并不需要对卷积核进行翻转。因此所采用的运算形式从严格意义上更接近于**互相关**（Cross-correlation），其离散形式为：

$$(x\star h)[n]=\sum_{m=-\infty}^{\infty}x[m]h[n+m] \tag{7.24}$$

其中符号 $\star$ 表示互相关运算。此外，上式中求和符号的上下限 $-\infty$ 和 ∞ 体现的是数学上严谨性的通用定义。在理论分析中，我们假设信号 $x[n]$ 与卷积核 $h[n]$ 均定义在整个整数域上的序列，因而可以描述任意长度的信号。在实际应用中，信号长度通常是有限的（例如长度为 15），为了与上述通用的数学形式保持一致，通常约定在信号定义范围之外，其取值为 0（即零填充）。

互相关本质上是一个用于衡量两个序列（或函数）之间相似性的算子。直观而言，它描述了一个信号（通常称为核）在另一个信号上“滑动”时，两者在不同位置上的匹配程度。其计算过程可以理解为：将一个较短的模板（即核）放置在较长信号的起始位置，在当前重叠区域内逐元素相乘，并将所有乘积求和，得到该位置的输出值。随后，将模板向右移动一个单位步长，并重复上述过程。在某个位置上得到的输出值越大，表示该位置处模板与输入信号局部结构的相似程度越高。这种“滑动”计算机制确保了算子具备**平移等变性**（Translation Equivariance）：即无论目标特征出现在输入信号的任何位置，只要其形态与模板（或核）相似，均能够被有效地识别与提取出来，并且其响应位置会随目标特征位置的变化而保持同步偏移。

在机器学习中，上述互相关运算相当于使用同一组可学习的权重，对当前输入及其邻域的特征进行加权求和，从而提取更抽象或更高层的特征表示。对于一维时序数据，该过程等价于在时间轴上使用固定大小的滑动窗口，对当前时刻及其前后邻域数据进行加权聚合；对于二维数据（如图像），该窗口在空间上扩展为一个局部感受野，通常表现为在上下左右方向上扩展后的矩形或正方形区域；而在三维数据（如视频或三维医学影像）中，该局部感受野进一步扩展为三维体素块（Voxel），即在空间与时间或深度维度上扩展后的立方体。

尽管从严谨的数学定义来看，卷积神经网络中的核心运算更接近于互相关，但为了遵循已形成的习惯表述，本书在后续章节中仍将其称为“卷积”。卷积神经网络最早在图像识别与处理任务中取得了突破性成功。因此，接下来主要以图像识别任务为例来讨论卷积神经网络的基本结构与工作机制。

图像通常可以表示为一个二维矩阵。图7.5给出了一个 3×3 卷积核作用于一个 4×4 二维输入矩阵（即由 4×4 个像素构成的图像）的示例。为便于理解，卷积核中的元素被简单设置为从 1 到 9 的数值。为了在输入的每一个位置上都能够应用相同的卷积操作，通常需要在输入的边界处进行填充（Padding），其大小一般由卷积核的尺寸决定。以 $k\times k$ 卷积核为例（其中 k 为奇数），常见的填充大小为 $\lfloor\frac{k-1}{2}\rfloor$（向下取整）。为了尽量减小填充对卷积结果的影响，通常采用零值填充。在图7.5中，相邻位置的卷积计算区域相对于前一位置发生了一个单位的平移，即向右或向下移动一个格子（对应图像的一个像素）。该相邻两次卷积计算之间的位移称为步长（Stride）。卷积核每次仅移动一个步长是最常见的设置，因为这种方式能够较好地保留输入中的

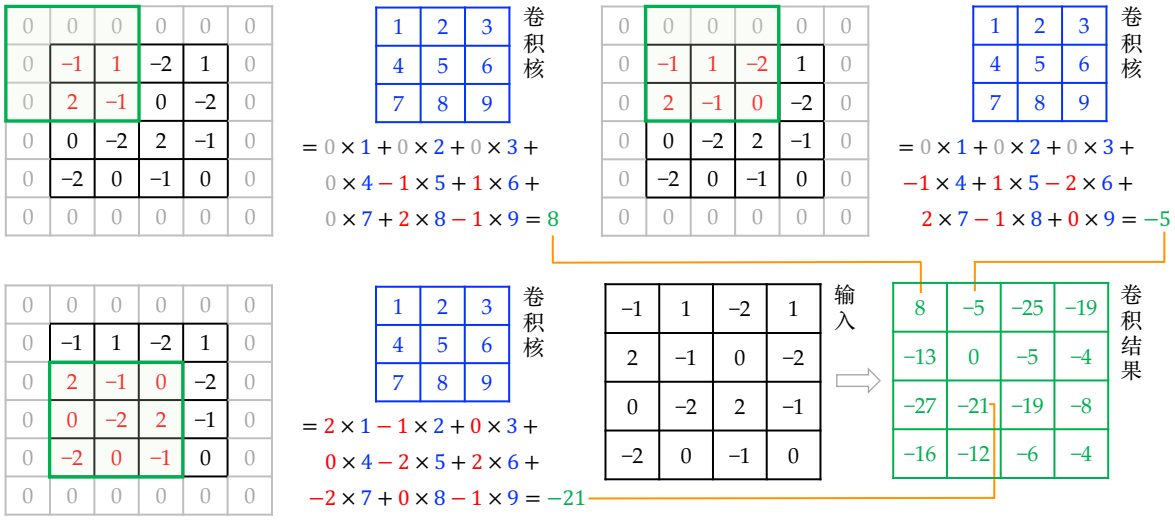


图 7.5: 二维卷积示意图。输入为一个 4×4 的矩阵，所采用的卷积核大小为 3×3（以蓝色矩阵表示）。为了在输入的每一个位置上都能够应用相同的卷积操作，通常需要在输入的边界处进行填充。为减小填充对结果的影响，一般采用零值填充（图中灰色边框表示填充区域）。左上子图展示了 3×3 卷积核作用于输入第一个位置时的计算结果，右上子图展示了同一卷积核作用于输入第二个位置时的结果。通过对比上方两个子图可以看出，表示当前计算区域的绿色方框向右平移了一格，即卷积的步长为 1（实际应用中步长也可设为大于 1）。左下子图展示了卷积核作用于输入某一中间区域时的结果，右下子图的绿色矩阵则给出了完整卷积运算后的输出结果。

细节信息。但在实际应用中，步长也可以设置为大于 1。当采用适当的填充方式并将步长设置为 1 时，卷积输出的空间尺寸可以与输入保持一致（如图7.5右下子图所示）。

对于输入图像，上述卷积操作可以理解为：使用同一组卷积核参数，在图像的所有可能位置上进行滑动，并与每个与卷积核尺寸相同的局部区域进行内积运算。向量之间的内积在一定程度上可以视为两者相似性的度量。为了形象地理解这一过程，可以将其类比为声学中的共振：想象你在一个放置着调音叉的房间里弹钢琴。只有当你弹出的音频频率（对应图像中出现的某种特征）接近调音叉的固有频率（对应卷积核所刻画的模式）时，调音叉才会产生明显的共振。类似地，当卷积核滑动到与其模式匹配度较高的图像区域时，各对应元素的乘积往往数值较大且为正，累加后得到较大的响应；而当匹配程度较低时，各乘积项往往相互抵消，导致最终的累加结果较小。因此，卷积运算可以看作一种 “模式匹配” 或 “特征检测” 机制：卷积核在图像上滑动，对与其匹配的局部结构产生较强响应，从而突出特定类型的特征。

如前文互相关性质的讨论类似，卷积运算同样具有重要的平移等变性。这意味着无论特征出现在图像的哪个位置，与其匹配的卷积核都能捕捉到它，并将捕捉到的特征输出到特征图的对应位置。当输入图像中的目标特征发生空间位移时，特征图中捕捉到该特征的位置也会发生相应的位移，但特征内容和特征值大小保持不变。特征图（Feature Map）是输入数据（如图像）经过卷积核提取特征后所产生的二维矩阵（如图7.5右下方的绿色方阵）。通过在空间上共享同一组卷积核参数，模型能够在不同位置检测到相似的特征。这种参数共享机制使得同一特征无

需在所有空间位置分别学习，从而显著减少了总体的参数量。卷积的这一特性也回应了本章开篇提到的多层深度结构能够有效缓解连接爆炸问题。当物体或特征（可出现在图像的任意位置）被某个卷积核检测到之后，与该卷积结果相连接的高层网络可以通过进一步的特征抽象与压缩，实现对该物体的识别。相比之下，若采用全连接的结构，由于需要对图像中不同位置的同一特征分别建模，往往需要在所有可能的位置重复学习相似的权重参数。当需要识别的物体或特征数量不断增加时，这种方式会导致参数规模呈指数增长，从而引发连接组合爆炸问题。

图7.6展示了三种典型卷积核作用于同一图像时的效果。边缘检测算子旨在提取物体的轮廓信息。以图中 3×3 的卷积核为例，其中心元素取较大正值（即为 8），周围权重均为 -1。当该卷积核作用于像素值相似或平滑的区域（即低频区域）时，加权求和的结果趋近于 0（表现为黑色背景）。而当该卷积核滑过像素值剧烈变化的边界（即高频区域）时，中心像素与周围像素的差异被显著放大，产生较大的正值或负值响应，在视觉上表现为清晰的轮廓线条。锐化核同样用于增强图像中的高频信息，即强化相邻像素之间的差异，从而提升图像的清晰度与对比度。然而，与边缘检测核不同，锐化核不仅强调局部差异，还在一定程度上保留原始图像信息，因此其作用效果更偏向于“增强”，而并非对非边缘信息的完全抑制。锐化核可以分解为以下恒等算子与拉普拉斯算子的叠加：

$$\underbrace{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}}_{\text{锐化核}} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{恒等算子}} + \underbrace{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}}_{\text{拉普拉斯算子}}$$

上式等式右侧的第一个矩阵（即恒等算子）用于保留中心像素的原始值，而第二个矩阵（即拉普拉斯算子）则通过对比中心像素与其邻域像素的差异来刻画局部变化程度。当中心像素与邻域像素的取值接近时，该差分项趋于抵消；而在边缘区域，该项会放大差异，使亮的一侧更亮、暗的一侧更暗，从而增强对比度。模糊核（平滑核）的作用相对直观，其本质是对邻域像素进行加权平均，从而抑制局部高频变化与噪声，并在视觉上产生模糊效果。需要指出的是，上述三类典型卷积核的参数是人为设计的，用于说明其作用机理。在实际的卷积神经网络中，卷积核参数一般先随机初始化，然后利用反向传播算法进行训练和优化得到。

彩色图像通常采用 RGB 编码，其每个像素由红（Red）、绿（Green）、蓝（Blue）三个颜色分量组成，每个分量的取值通常为 0 至 255 的整数。因此，为了生成一个特征图，卷积运算不再由单一的二维卷积核完成，而是需要多组与输入通道（Channel）数相同的二维卷积核参数。如图7.7所示，在多通道卷积过程中，每一个特征图（图中绿色矩阵）的生成包括两个步骤：首先，各输入通道分别与对应的二维卷积核进行卷积操作；然后，将各通道在对应位置的卷积结果逐元素相加，从而得到最终的二维输出特征图。由此可见，生成一个特征图所需的卷积参数通常是一个三维张量（Tensor）。张量是多维数组，它是标量、向量和矩阵在更高维空间中的推广。其中，零阶张量对应标量，一阶张量对应向量，二阶张量对应矩阵，而三阶及以上统称为高阶张量。彩色图像最初输入通常包含红、绿、蓝三个通道。然而，随着卷积神经网络层数的加深，后

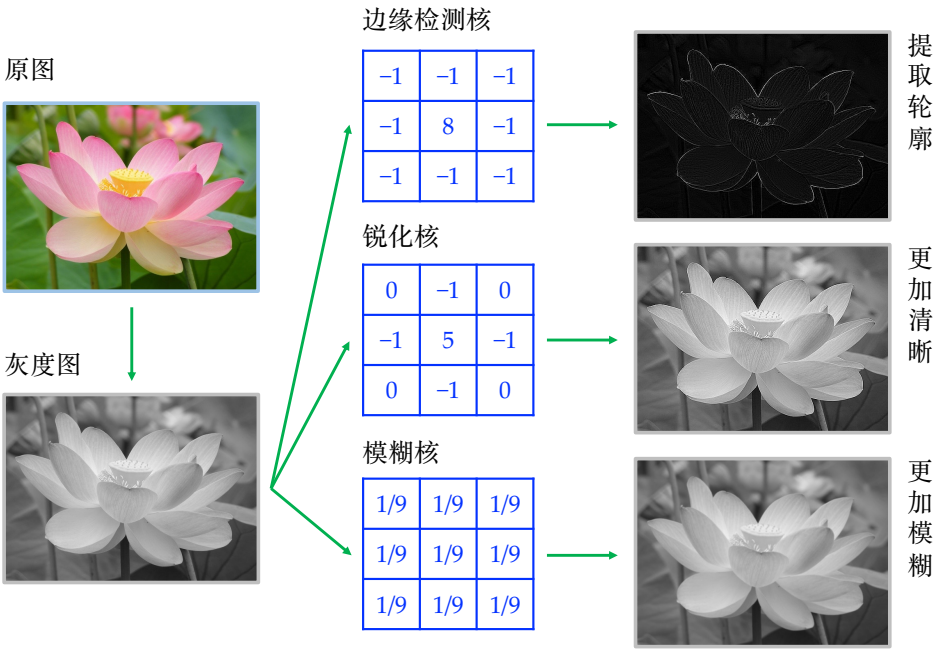


图 7.6: 三个不同卷积核作用于同一图像的效果示意。为了突显卷积操作的效果，先将彩色原图转换为灰度图像。灰度表示的图像中，每个像素的取值范围为 0（黑色）至 255（白色）。边缘检测核可用于提取图像中的轮廓与边缘信息；锐化核能够增强局部对比度，使得图像细节更加清晰；而模糊核则通过平滑邻域像素值，使图像较原始灰度图呈现出更强的平滑与模糊效果。

续卷积层所处理的通道数是由前一层所生成的特征图数量决定。通道与特征图这两个术语虽在不同语境下常被互换使用，但侧重点有所不同：通常将网络的输入维度（张量在深度方向上的维度）称为“通道”，而将卷积操作后得到的具有空间结构的二维（或三维）数据称为“特征图”。

在计算卷积结果时，除了将卷积核与输入局部区域进行内积外，通常还会引入一个偏置项。从数学角度看，这种“卷积之后加偏置”的形式构成一个仿射变换（Affine Transformation），即在线性变换的基础上再加上一个平移项。卷积运算之后通常还需要进行非线性变换。在早期的神经网络中，常使用 Sigmoid 或 Tanh 作为激活函数，但由于它们在输入绝对值较大时容易进入饱和区，从而导致梯度消失问题，后来逐渐被 ReLU（Rectified Linear Unit）等函数所取代（如图7.8所示）。在卷积层之间引入非线性变换的目的是为了提升网络的函数近似能力。由于卷积操作本质上仍是线性变换，若多层卷积之间不加入非线性变换，则连续的线性变换仍然等价于单一的线性变换，从而显著限制模型的表达能力。

图7.8展示了三种常用的非线性激活函数。其中 Sigmoid 函数的值域为 $(0,1)$，这意味着其输出始终为正值。这一性质使得以 Sigmoid 作为激活函数的某一层输出恒为正，从而导致连接到下一层同一神经元的权重的梯度更新方向趋于一致（整体为正或为负）。如第6.4节所述，权重 w_{ij} 的梯度通常由误差项 δ_j 与输入 x_i 相乘得到。当采用单样本梯度更新，且各输入分量均为正时，连接到同一隐层神经元 h_j 的所有权重 w_{ij} 的梯度符号都由误差项 δ_j 决定。因此，在一次梯度更新中，这些权重只能同时增大或同时减小，难以实现部分权重增大、部分权重减小的差异

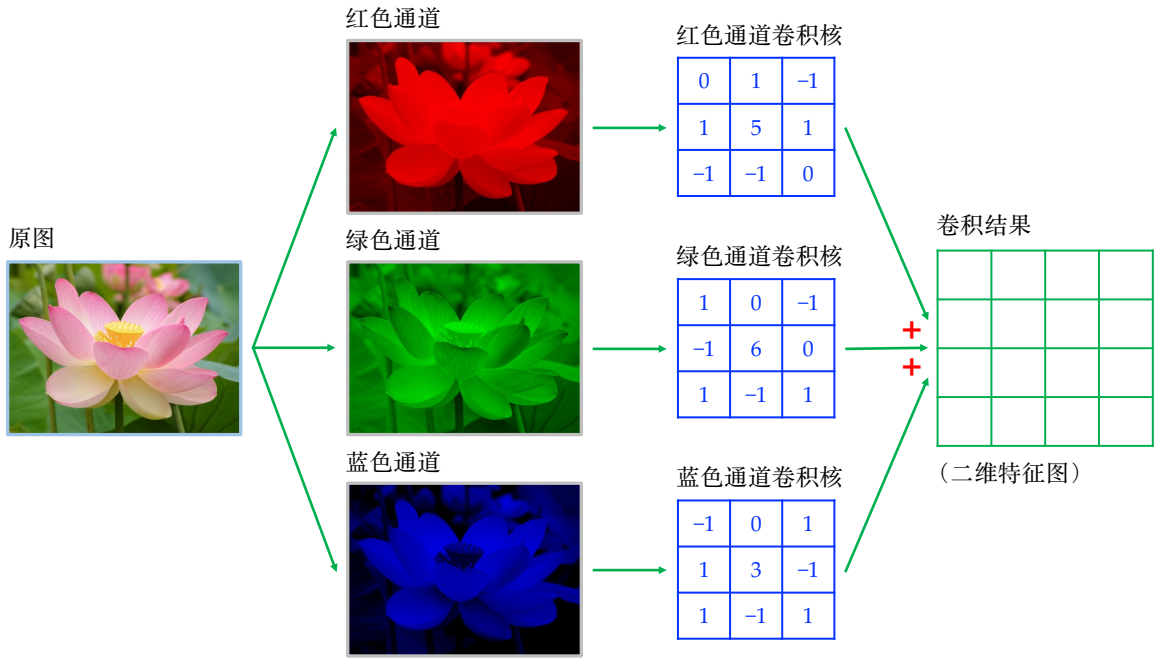
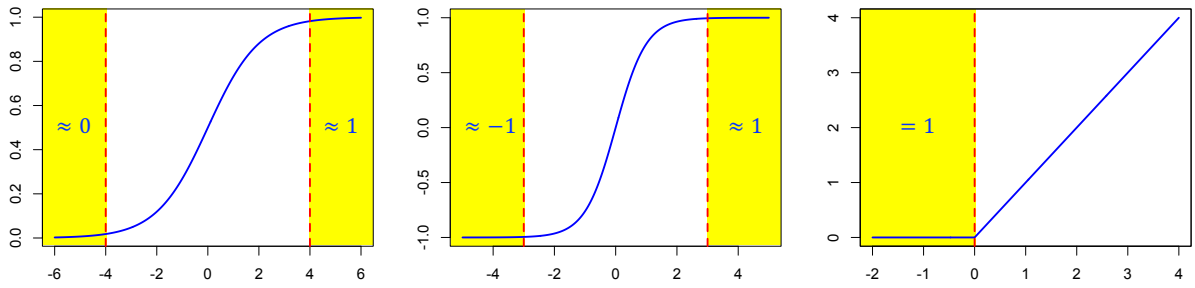


图 7.7: 多通道卷积示意图。彩色图像通常采用 RGB 编码，即每个像素由三个通道组成，分别对应红（Red）、绿（Green）、蓝（Blue）三个分量。在多通道卷积中，卷积核需要在输入的每个通道上分别进行卷积操作。随后，将各通道对应位置的卷积结果逐元素相加，从而得到最终的二维输出特征图。因此，卷积核不再是单一的二维矩阵，而是一个三维张量（Tensor），其通道数必须与输入的通道数一致（此例为 RGB 这三个通道，但在深层网络中该数值由上一层输出的通道数或特征图数量决定）。对于不同的输入通道，卷积核在该维度上对应一组独立的二维参数（通过训练自适应学习获得），从而能够从各通道中提取并融合互补的特征信息。

化调整。这种限制可能影响参数更新的灵活性，从而降低网络的优化效率。双曲正切函数 Tanh 在一定程度上弥补了 Sigmoid 函数这一缺陷，其输出范围为 $(-1, 1)$，且均值为 0。这种零中心化特性使得下一层的输入信号正负分布均衡，从而令梯度更新的方向更加多样化，通常能显著加快网络的收敛速度。然而，Sigmoid 和 Tanh 函数在输入绝对值较大时均会进入饱和区（图中黄色区域）。一旦进入饱和区，其导数将趋近于 0，从而引发梯度消失问题，使模型难以继续有效更新参数。ReLU 函数在保持非线性表达能力的同时，在正半轴区域（输入大于 0）上不再出现饱和现象。具体而言，对于输入大于 0 的区域，其导数恒为 1。这意味着，当神经元处于激活状态（即输出大于 0）时，反向传播的梯度不会因该激活函数的导数而衰减，从而在一定程度上缓解深层网络中的梯度消失问题。通常情况下，使用 ReLU 的网络收敛速度显著快于使用 Tanh 的网络，这主要得益于其在正区间的非饱和线性特性。但需要注意的是，ReLU 的输出始终非负，因此仍然存在非零中心化的问题，这在一定程度上会影响网络的优化效率。在实际应用中，这一问题通常可以通过归一化（Normalization）得到有效缓解。我们将在第7.6节中介绍该方法。

卷积结果在经过非线性变换后，通常会紧接着进行池化（Pooling）操作。池化的主要作用是对输入的局部区域进行下采样（Downsampling），从而减小特征图的空间尺寸并降低分辨率。



(a) Sigmoid$(x) = \frac{1}{1+e^{-x}}$ (b) Tanh$(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (c) ReLU$(x) = \max(0, x)$

图 7.8: 三种常用的非线性激活函数。(a)Sigmoid 函数。其值域在 $(0, 1)$ 区间，即输出始终为正。这一性质会导致：在使用单个样本进行梯度更新时，指向同一神经元的各权重梯度在更新方向上趋于一致（同正或同负），从而由于难以实现差异化的权重更新而影响优化效率。(b) 双曲正切函数 Tanh。其输出范围为 $(-1, 1)$，在适当对称的输入分布下，均值为 0。零中心化特性使得梯度的更新方向更加多样化，通常有助于加快网络收敛速度。然而，Sigmoid 和 Tanh 函数在输入绝对值较大时均存在饱和区（图中黄色区域），一旦进入饱和区，其导数趋近于 0，从而导致梯度消失。(c)ReLU 函数。在输入大于 0 的区域，其导数恒为 1。这意味着，当神经元处于激活状态（即输出大于 0）时，反向传播的梯度不会因该激活函数的导数而衰减，从而在一定程度上缓解深层网络中的梯度消失问题。但由于其输出始终非负，存在非零中心化问题，会对优化效率产生一定影响（之后通过归一化方法缓解）。

以图7.9中 2×2 的最大池化为例，其在每个局部区域（包含相邻的 4 个特征值）中仅保留数值最大的元素，从而得到下采样后的特征图。常见的池化方式包括最大池化（Max-pooling）、最小池化（Min-pooling）和平均池化（Average-pooling）。其中，最大池化应用最为广泛，因为它能够保留局部区域中最显著（响应值最大）的特征。相比之下，平均池化会将强响应与周围较弱的响应（甚至噪声）进行平均，从而在一定程度上导致重要特征被 “平滑” 或 “淡化”，使得原本鲜明的特征变得模糊。然而，在希望下采样的同时尽量保留原有整体信息时，平均池化仍然是一种合理选择。最小池化提取局部区域中最小的响应值，这在许多实际场景中并无实际意义。例如，在经过 ReLU 激活函数后，大量背景区域的特征值为 0，最小池化往往会保留这些无效的零值。与卷积操作不同，池化通常不需要在输入边界进行填充。这是因为池化本质上是一种下采样过程，其目标是主动减小特征图的空间尺寸以降低后续的计算复杂性，因此无需像卷积那样通过填充（Padding）来维持输出尺寸。池化操作通常也不需要可学习的参数。

池化的步长（Stride）通常与其窗口的尺寸相同。如图7.9所示，相邻池化区域之间通常不存在重叠。然而，在一些卷积神经网络设计中，也可以采用步长小于窗口尺寸的设置，从而使不同池化区域之间产生部分重叠。最大池化的一个重要特性是能够引入局部**平移不变性**（Translation Invariance）。这意味着当目标特征在视野内发生小幅位移时，只要该特征仍然落在同一个池化窗口（或局部区域）内，其输出的最大值通常保持不变。如图7.10所示，最大池化与卷积在处理空间位移时表现出不同特性：前者在局部范围内对目标特征（以红色小方块表示）的小幅平移具有一定鲁棒性，使输出对位置变化不敏感，从而增强模型对目标特征在视野中轻微位置变化的泛化能力；而后者则体现为平移等变性，即特征图的响应会随着输入中目标特征位置的变化

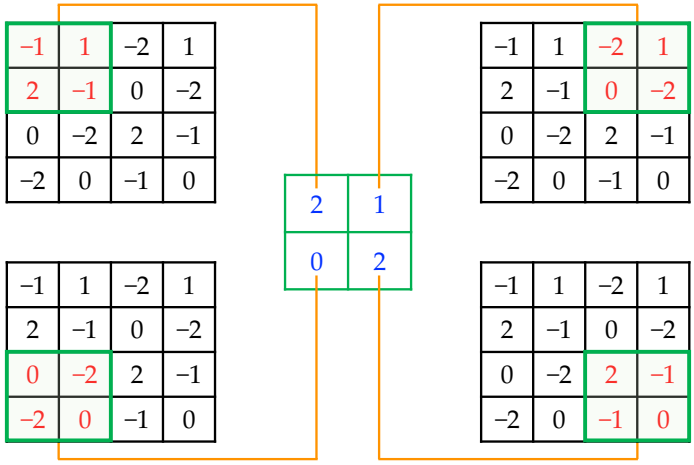


图 7.9: 最大池化示意。对一个 4×4 的特征图应用尺寸为 2×2 的最大池化进行下采样。在每个由相邻 4 个特征值所组成的局部区域中，选取并保留其中的最大值，从而得到一个尺寸为 2×2 的输出特征图（图中央的绿色方阵）。

而发生对应的空间位移。

接下来，我们以 2018 年图灵奖得主杨立昆（Yann LeCun）[1]提出的经典网络 LeNet-5 为例[47]，介绍卷积神经网络的基本结构。需要说明的是，我们以该网络为主要框架，并且结合前文讨论的内容进行了适度的调整，以更好地体现卷积神经网络在后续发展中的一些主流设计思想与变化。

LeNet-5 卷积神经网络的结构如图7.11所示。网络输入为 32×32 像素的灰度手写数字图像，因此初始输入仅包含一个通道。网络的主体由两组连续的 “卷积 + 非线性变换 + 最大池化” 操作构成。其中，卷积层均采用 5×5 的卷积核，非线性激活函数采用 Tanh，池化层则采用 2×2 的窗口尺寸。需要注意的是，在 LeNet-5 提出时尚未普遍采用填充来保持卷积前后特征图的尺寸一致。因此在不使用填充的情况下，特征图尺寸会在每次卷积后逐层缩小。具体而言，对于 5×5 卷积核，若希望保持卷积前后空间尺寸不变，需要在输入四周填充 2 个像素；否则，每次卷积后特征图在高和宽方向上均减少 4 个像素（即上下、左右各缩减 2 个像素）。鉴于手写数字通常位于图像中心区域，边缘像素所包含的信息对识别任务的重要性相对较低，因此这种边界信息的损失通常不会显著影响模型性能。我们以第二个卷积层为例来说明卷积层参数量的计算方法。该层属于多通道卷积，其输入由 6 个通道（即 6 幅特征图）组成。对于每个输出特征图，其卷积核尺寸为 $6 \times 5 \times 5$。加上每个输出特征图对应一个偏置参数，生成单个特征图需要 151 个参数。由于该层共有 16 个特征图，其总参数量为 $2,416 = 16 \times 151$。

为了使模型更符合当前的设计，我们将原始 LeNet-5 中的平均池化替换为效果更优的最大池化。在最后一个池化层之后，网络生成了 16 幅 5×5 的特征图。通过展平操作，该三维张量转换为一个 400 维 $(16 \times 5 \times 5)$ 的特征向量，这可视为对输入图像高层语义特征的压缩表示。最后，

[1]全名为杨 · 安德烈 · 勒昆（Yann André Le Cun），法国计算机科学家。

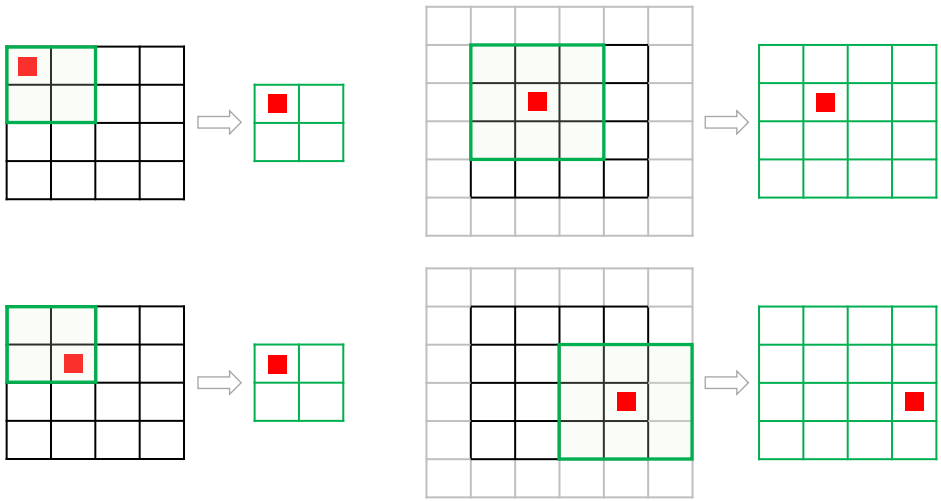


图 7.10: 平移不变性与平移等变性对比图。(a) 最大池化所带来的局部平移不变性：当目标特征（以红色小方块表示）在局部范围内发生小幅位移时，只要其仍落在同一池化区域内，输出的最大值通常保持不变，从而增强模型对局部位置扰动的鲁棒性。(b) 卷积运算所具有的平移等变性：当输入图像中的目标特征发生空间位移时，其对应卷积后的特征图也会发生相同方向与幅度的位移，表现为特征响应在空间上的同步平移。

该特征向量依次通过三个全连接层。前两个全连接层在线性变换后采用 Tanh 激活函数，而输出层则采用 Softmax 函数，将原始预测得分（Logits）转换为各类别的概率分布，从而实现对 0 至 9 共十个数字类别的分类。需要说明的是，原始 LeNet-5 的最后一层采用了高斯连接（Gaussian Connection），此处改为目前更常用于分类任务的 “线性层 + Softmax” 形式。网络通常以交叉熵损失函数为优化目标，并通过反向传播算法进行端到端的参数学习与优化。

需要指出的是，原始输入图像中每个像素的取值范围为 0 至 255 的整数，其均值期望约为 $127.5 = \frac{0+255}{2}$（实际应根据训练数据计算均值）。若直接进行卷积运算，输出特征值的绝对值通常仍然较大，易落入 Tanh 激活函数的饱和区，导致激活值趋近于 1，从而引发梯度消失问题。为缓解这一问题，通常首先将输入数据归一化到 $[0, 1]$ 区间。对于灰度图像，只需将像素值除以 255。随后，为了使数据分布具有零均值和单位方差，从而有助于网络训练过程的数值稳定性（详见第7.6节讨论），通常还会对归一化后的数值进行标准化处理，即减去均值并除以标准差。

图7.12展示了卷积神经网络不同层所学习到特征的可视化。该可视化以一组鸟类图像作为输入，并基于相应层神经元的激活值进行重建。从图中可以观察到，网络的低层（靠近输入端）主要捕捉边缘、方向和纹理等基础视觉特征。随着层数的加深，感受野逐渐扩大，中间层开始编码更具结构性的局部模式，例如鸟类的头部、喙部及爪部特征。在更高层，网络已能够整合更大范围的上下文信息，从而形成对复杂结构和语义模式的表征。在此示例中，高层表示已经能够分辨出不同鸟类的完整形态。这些高层特征经过后续的全连接层映射，最终与具体的语义类别建立对应关系，从而实现对输入图像的识别与分类。

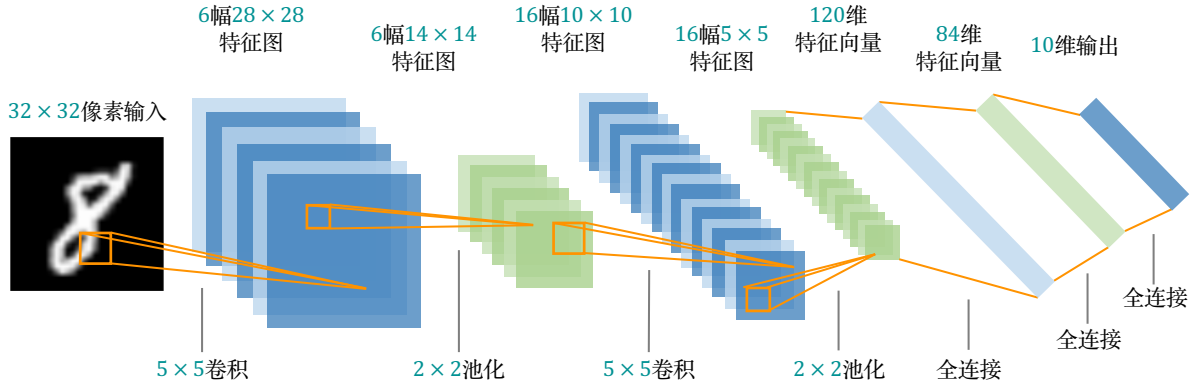
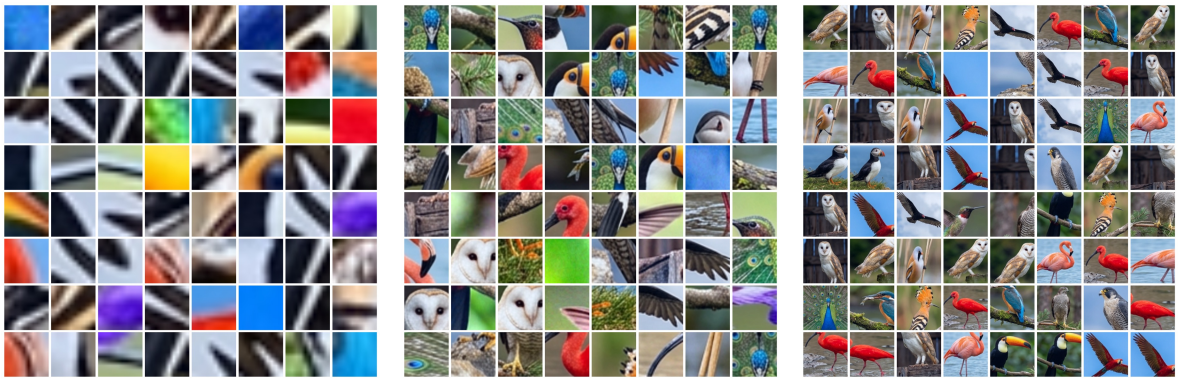


图 7.11: LeNet-5 卷积神经网络结构示意图。输入为 32×32 像素的灰度手写数字图像（类别为 0 至 9 的十个数字）。网络主体由两组连续的卷积与最大池化层构成。在最后一个池化层之后，将得到的 $16 \times 5 \times 5$ 多通道特征图展平为 400 维的向量，随后通过三层全连接层进行特征变换与降维，最终输出 10 维向量，对应 0 至 9 十个数字类别的分类结果。此图重绘自文献 [47]。



(a) (b) (c)

图 7.12: 卷积神经网络不同层所学习到特征的可视化。(a) 低层特征（如边缘与纹理）。(b) 中层特征（如局部结构与简单形状）。(c) 高层特征（如全局结构与复杂模式）。

7.4 循环神经网络

为了使神经网络具备处理序列数据的能力，网络需要能够将当前输入与历史信息进行有效整合，并利用整合后的信息进行预测。如何有效地整合历史信息形成“记忆”，并在该记忆中尽可能保留与后续预测相关的关键信息，是设计并构建此类网络的核心。由于记忆需要不断与新的输入进行融合，因此在时刻 t 形成的记忆（或状态）$\boldsymbol{h}_t$ 通常依赖于上一时刻的记忆 $\boldsymbol{h}_{t-1}$ 以及当前输入。若将 $\boldsymbol{h}_t$ 视为网络在时刻 t 的隐藏状态，则其取值由前一时刻的隐藏状态 $\boldsymbol{h}_{t-1}$ 与当前输入共同决定，从而形成一种随时间递归更新的状态演化机制。与前馈神经网络中信息沿单向传播不同，此类网络在结构上引入了跨时间步的循环依赖关系，使得当前状态不仅影响后续状态，也间接影响未来的输出。因此，**循环神经网络**（Recurrent Neural Network）是一类神经元之间存在循环连接（自身指向并影响其后续时刻取值），且通常用于处理序列数据的神经网络。

实现当前时刻 t 的输入 $\boldsymbol{x}_t$ 与上一时刻的记忆（即隐藏状态）$\boldsymbol{h}_{t-1}$ 整合，并生成当前记忆 $\boldsymbol{h}_t$ 的基本方式是：分别使用两组权重矩阵对 $\boldsymbol{x}_t$ 和 $\boldsymbol{h}_{t-1}$ 进行线性变换，然后将结果相加。该整合方式可以简单理解为当前记忆是由之前记忆与当前输入的加权和所构成。在此基础上，通常还会引入非线性激活函数，以增强模型的表达能力，从而得到如下形式：

$$\boldsymbol{a}_t = \mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{V}\boldsymbol{x}_t + \boldsymbol{b} \tag{7.25}$$

$$\boldsymbol{h}_t = \sigma(\boldsymbol{a}_t) \tag{7.26}$$

其中 $\mathbf{U}$ 为状态转移矩阵，用于捕捉状态随时间演化的规律；$\mathbf{V}$ 为输入权重矩阵，用于提取当前时刻的输入特征；$\boldsymbol{b}$ 为偏置项，$\boldsymbol{a}_t$ 为 t 时刻隐藏层的净输入，而 $\sigma(\cdot)$ 为非线性激活函数，通常取 Tanh 或 Sigmoid 函数。在得到当前隐藏状态 $\boldsymbol{h}_t$ 之后，网络的输出通常通过另一组权重矩阵从该状态中计算得到：

$$\boldsymbol{y}_t = \mathbf{W}\boldsymbol{h}_t \tag{7.27}$$

上述基本循环神经网络的结构如图7.13(a) 所示，其中隐藏状态 $\boldsymbol{h}$ 包含一个指向自身的反馈环路，表示当前时刻的状态取值受到前一时刻状态的影响，这也是“循环”神经网络名称的由来。图7.13(b) 则展示了该网络按时间步 t 展开的示意图，其中不同时间步对应的网络结构在形式上是相同的，并且共享同一组权重参数。值得注意的是，这种循环结构也赋予了网络处理变长序列的能力。由于各时间步共享同一组权重参数，网络可以根据输入序列的长度在时间维度上进行展开，从而适应不同长度的输入，而无需为每种序列长度单独定义网络参数。

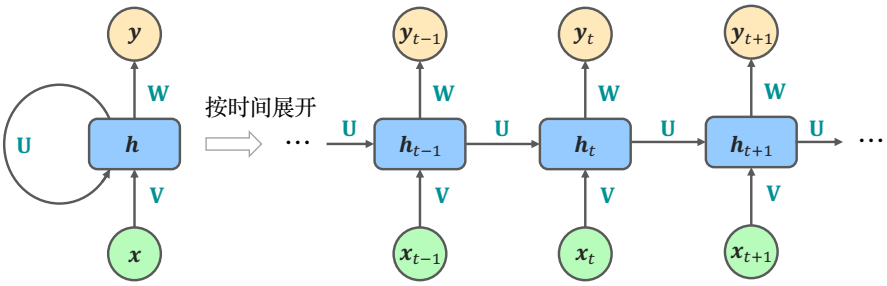


图 7.13: 循环神经网络及其按时间步展开的示意图。(a) 基本循环神经网络的结构示意。(b) 循环神经网络在时间维度上的展开形式，其中各时间步共享同一组权重参数。

在序列预测任务中，循环神经网络可以在每个时间步产生输出。这意味着网络可以在序列的每一时刻均产生预测，而不一定需要等到整个序列结束后才进行预测。例如，在基于历史成交价格预测股票价格走势的任务中，模型可以在多个时间点上持续进行预测，以捕捉市场随时间变化的动态过程。设某一长度为 T 的时序样本为 $\boldsymbol{x}_{1:T} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$，其对应的每个时间步的预测目标为 $\boldsymbol{t}_{1:T} = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_T)$。我们可以在每个时间步定义损失函数 $\mathcal{L}_t$，该损失由模型在时刻 t 的预测 $\boldsymbol{y}_t$ 与对应目标 $\boldsymbol{t}_t$ 计算得到。若为回归任务，可采用平方误差损失；若为分类任务，则可采用交叉熵损失。整个序列的总损失函数可以定义为各时间步损失的平均（或累加），其形

式如下：

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_t \tag{7.28}$$

循环神经网络的训练目标同样是通过调整其参数 $\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{V}, \mathbf{W}, \boldsymbol{b}\}$ 来最小化损失函数。在采用反向传播算法对参数进行更新和优化时，需要计算总损失 $\mathcal{L}$ 关于各参数的梯度。根据公式7.28，总损失 $\mathcal{L}$ 关于参数 $\boldsymbol{\theta}$ 的梯度可以通过对各时间步的损失 $\mathcal{L}_t$ 分别求梯度并取平均得到。以将 $\mathcal{L}_t$ 关于状态转移矩阵 $\mathbf{U}$ 的梯度为例，可以先按照图7.13(b) 所示的方式将网络在时间维度上展开，然后沿着与前向计算相反的方向（即沿图中箭头的反方向）传播误差信号，这一过程称为**随时间反向传播**（Backpropagation Through Time）。根据复合函数求导的链式法则，当计算 $\mathcal{L}_t$ 对较早时刻 $k\,(k < t)$ 的隐藏状态 $\boldsymbol{h}_k$ 及其对应参数矩阵 $\mathbf{U}$ 的梯度时，梯度需要穿过从时刻 t 到 k 的所有中间隐藏状态。这一过程会产生如下偏导数的连乘形式：

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_k} = \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_t}\left(\prod_{i=k+1}^{t}\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}}\right) \tag{7.29}$$

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} = \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{h}_k}\cdot\frac{\partial \boldsymbol{h}_k}{\partial \mathbf{U}} \tag{7.30}$$

公式7.29中的连乘项 $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}}$ 可以进一步分解为：

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{a}_i}\cdot\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{h}_{i-1}} \tag{7.31}$$

其中 $\boldsymbol{a}_i$ 为 i 时刻隐藏层的净输入。由于 $\boldsymbol{h}_i = \sigma(\boldsymbol{a}_i)$，且激活函数 $\sigma(\cdot)$ 是作用于 $\boldsymbol{a}_i$ 每个分量的逐元素运算，因此 $\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{a}_i}$ 对应的雅可比矩阵（Jacobian Matrix）是一个对角矩阵，其形式为：

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{a}_i} = \operatorname{diag}\left(\sigma'(\boldsymbol{a}_i)\right) = \begin{bmatrix} \sigma'(a_{i,1}) & 0 & \cdots & 0 \\ 0 & \sigma'(a_{i,2}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'(a_{i,M}) \end{bmatrix} \tag{7.32}$$

其中 $a_{i,j}\,(j \in \{1, \ldots, M\})$ 表示向量 $\boldsymbol{a}_i$ 的第 j 个分量，$\sigma'(\cdot)$ 为激活函数的导数，而 M 为隐藏层神经元的数量（即向量 $\boldsymbol{h}_i$ 的维度）。根据公式7.25，我们可得：

$$\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{h}_{i-1}} = \mathbf{U} \tag{7.33}$$

其中 $\mathbf{U}$ 是一个 $M \times M$ 的矩阵。结合公式7.32和7.33的结果，我们可以得到相邻时刻隐状态间的雅可比矩阵为：

$$\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \operatorname{diag}\left(\sigma'(\boldsymbol{a}_i)\right)\mathbf{U} \tag{7.34}$$

计算跨越多个时间步的隐状态之间的梯度时，需要将从 $k+1$ 到 t 时刻的所有雅可比矩阵连乘：

$$\prod_{i=k+1}^{t}\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \prod_{i=k+1}^{t}\operatorname{diag}\left(\sigma'(\boldsymbol{a}_i)\right)\mathbf{U} \tag{7.35}$$

根据矩阵**谱范数**（Spectral Norm）的次乘性（Submultiplicativity），即 $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|$，可以

得到该矩阵连乘的范数上界：

$$\left\| \prod_{i=k+1}^{t} \text{diag}\left(\sigma'(\boldsymbol{a}_i)\right) \mathbf{U} \right\| \leq \prod_{i=k+1}^{t} \underbrace{\left\|\text{diag}\left(\sigma'(\boldsymbol{a}_i)\right)\right\|}_{\leq 1} \cdot \|\mathbf{U}\| \tag{7.36}$$

矩阵谱范数在数值上等于矩阵的最大奇异值（若矩阵为对称矩阵，则谱范数等价于其最大特征值的绝对值）。直观而言，谱范数表征了矩阵对向量长度的“最大放大倍数”，即该矩阵所描述的线性变换在作用于所有单位向量时能够产生的最大伸缩比例。不等式7.36右侧对角矩阵 $\text{diag}\left(\sigma'(\boldsymbol{a}_i)\right)$ 的谱范数（即最大奇异值）总是小于或等于 1，其原因在于该矩阵的最大奇异值就是对角元素绝对值的最大值，而这些对角元素正是所使用激活函数的导数值。若采用 Tanh 作为激活函数，其导数为：

$$\text{Tanh}'(x) = 1 - \text{Tanh}^2(x) \tag{7.37}$$

由于 $\text{Tanh}(x) \in (-1, 1)$，因此 $\text{Tanh}'(x) \in (0, 1]$，其谱范数上界为 1。若采用 Sigmoid 激活函数，其导数为：

$$\text{Sigmoid}'(x) = \text{Sigmoid}(x)(1 - \text{Sigmoid}(x)) \tag{7.38}$$

其中 $\text{Sigmoid}(x) \in (0, 1)$，因此 $\text{Sigmoid}'(x) \in (0, 0.25]$。当 $\text{Sigmoid}(x) = 0.5$（即输入 $x = 0$）时，导数取得最大值 $0.25 = 0.5(1 - 0.5)$，故其谱范数上界为 0.25。综上所述，在采用上述两类常见激活函数时，$\|\text{diag}\left(\sigma'(\boldsymbol{a}_i)\right)\| \leq 1$ 恒成立。这意味着误差信号在通过激活函数层时，必然会面临不同程度的衰减。因此，若要求梯度在跨越多个时间步传播时不至于迅速衰减，则需要权重矩阵 $\mathbf{U}$ 的谱范数足够大，以补偿激活函数导数所引起的梯度收缩。然而，过大的 $\|\mathbf{U}\|$ 会严重损害训练过程的稳定性，并且增加过拟合的风险。矩阵 $\mathbf{U}$ 的谱范数表征了其在主方向上的拉伸倍数。当 $\|\mathbf{U}\|$ 过大时，输入中的微小扰动会被显著放大并传递到输出，使得损失函数对输入变化较为敏感，进而导致梯度幅值过大（甚至出现梯度爆炸），从而影响训练的稳定性。此外，模型通常应对输入的微小扰动保持一定的鲁棒性，而过大的 $\|\mathbf{U}\|$ 容易使模型过度拟合训练数据中的噪声或异常模式，从而降低泛化能力。为缓解过拟合，常用的方法（如权重正则化或权重衰减）通常会倾向于减小权重矩阵 $\mathbf{U}$ 的范数，但这又加剧了梯度的收缩。由此可见，基础的循环神经网络陷入了两难困境：为了缓解梯度消失，需要较大的谱范数来抵消激活函数引起的梯度衰减；而为了维持训练稳定性与模型的泛化能力，又要求较小的谱范数。

在实际应用中，与梯度消失相比，权重矩阵谱范数过大所引起的训练不稳定（甚至导致无法收敛）通常具有更直接且严重的负面影响。两害相较取其轻，在权衡中往往倾向于通过正则化等手段限制权重矩阵的谱范数，以确保训练过程的稳定性。然而，这种约束在一定程度上会加剧梯度在时间维度上的衰减。对于基础循环神经网络而言，这种倾向表现为模型无法将时刻 t 的损失信号 $\mathcal{L}_t$ 有效回传至相隔较远的时刻 k，导致模型难以捕捉序列中的长程依赖关系（Long-range Dependencies）。正是这一困境促使研究者们通过引入门控机制（Gating Mechanism）来缓解梯度消失问题。所谓“门控”，是指通过模拟物理开关的启闭程度（通常取值在 $[0, 1]$ 区间）来调节信息的通过比例，从而实现对信息流动的选择性控制。

长短期记忆网络（Long Short-term Memory）是一种改进的循环神经网络结构 [28]，简称 LSTM，用于缓解基础循环神经网络在建模长序列时难以捕捉长程依赖关系的问题。LSTM 通过引入显式的记忆单元或称细胞状态（Cell State），并结合门控机制对信息的写入、保留与输出进行选择性控制，使得信息能够在较长时间跨度上有效传递，从而显著缓解梯度消失问题。如图7.14所示，LSTM 单元的前向传播过程（从输入到输出）可由以下公式描述：

$$\boldsymbol{f}_t = \sigma_g\left(\mathbf{U}_f \boldsymbol{h}_{t-1} + \mathbf{V}_f \boldsymbol{x}_t + \boldsymbol{b}_f\right) \tag{7.39}$$

$$\boldsymbol{i}_t = \sigma_g\left(\mathbf{U}_i \boldsymbol{h}_{t-1} + \mathbf{V}_i \boldsymbol{x}_t + \boldsymbol{b}_i\right) \tag{7.40}$$

$$\boldsymbol{o}_t = \sigma_g\left(\mathbf{U}_o \boldsymbol{h}_{t-1} + \mathbf{V}_o \boldsymbol{x}_t + \boldsymbol{b}_o\right) \tag{7.41}$$

$$\tilde{\boldsymbol{c}}_t = \sigma_c\left(\mathbf{U}_c \boldsymbol{h}_{t-1} + \mathbf{V}_c \boldsymbol{x}_t + \boldsymbol{b}_c\right) \tag{7.42}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{7.43}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \sigma_h(\boldsymbol{c}_t) \tag{7.44}$$

其中 $\boldsymbol{x}_t \in \mathbb{R}^D$ 表示时刻 t 的输入向量，其维度为 D；$\boldsymbol{h}_t \in \mathbb{R}^M$ 表示时刻 t 的隐状态（即输出）向量，其维度为 M。$\boldsymbol{f}_t$, $\boldsymbol{i}_t$, $\boldsymbol{o}_t \in \mathbb{R}^M$ 分别表示时刻 t 的遗忘门、输入门和输出门的激活向量，用于控制历史信息的保留、新信息的写入以及当前状态的输出比例。$\boldsymbol{c}_t \in \mathbb{R}^M$ 表示时刻 t 的细胞状态，它是整合历史信息与当前输入、形成长短期记忆的核心载体。之所以使用 “细胞” 一词，是为了形象地描述 LSTM 单元如同生物细胞一般，具有相对持久且独立的内部状态，并且通过门控机制对信息的流入、保留与输出进行调节。$\tilde{\boldsymbol{c}}_t \in \mathbb{R}^M$ 表示时刻 t 的候选细胞状态（Candidate Cell State）。之所以称为 “候选”，是因为 $\tilde{\boldsymbol{c}}_t$ 并不会直接成为新的细胞状态，而是先经过输入门 $\boldsymbol{i}_t$ 的调制（即 $\boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$）之后，才被有选择地累加至细胞状态 $\boldsymbol{c}_t$ 中。$\mathbf{U} \in \mathbb{R}^{M \times M}$ 为状态转移矩阵，$\mathbf{V} \in \mathbb{R}^{M \times D}$ 为输入权重矩阵，而 $\boldsymbol{b} \in \mathbb{R}^M$ 为偏置向量。这些权重矩阵和偏置向量共有四组，它们的下标 f, i, o 和 c 分别对应遗忘门、输入门、输出门和候选细胞状态计算时所使用的参数。$\sigma_g(\cdot)$ 通常为 Sigmoid 函数，用于产生 $[0, 1]$ 之间的门控系数。系数取值为 0 时表示完全阻断信息通过；取值为 1 时表示允许信息完全通过；而取值在 $(0, 1)$ 区间时，则表示允许相应比例的信息通过；$\sigma_c(\cdot)$ 通常为 Tanh 激活函数，用于生成候选细胞状态；$\sigma_h(\cdot)$ 可选 Tanh 函数或恒等映射 $\sigma_h(x) = x$。符号 $\odot$ 表示逐元素相乘（Hadamard Product）。

在 LSTM 单元的前向传播过程中（如图7.14所示），首先利用前一时刻的隐状态 $\boldsymbol{h}_{t-1}$ 与当前时刻的输入 $\boldsymbol{x}_t$，根据公式7.39、7.40和7.41分别计算遗忘门 $\boldsymbol{f}_t$、输入门 $\boldsymbol{i}_t$ 和输出门 $\boldsymbol{o}_t$。由于都采用了 Sigmoid 激活函数，这些门控向量所有元素的取值均处于 $[0, 1]$ 区间，它们表示之后在逐元素相乘时对应特征被保留的比例。接着，根据公式7.42计算出候选细胞状态 $\tilde{\boldsymbol{c}}_t$，它表示从当前输入 $\boldsymbol{x}_t$ 与隐状态 $\boldsymbol{h}_{t-1}$ 中提取出的、拟存入长期记忆的候选信息。随后，利用遗忘门 $\boldsymbol{f}_t$ 与输入门 $\boldsymbol{i}_t$ 分别对上一时刻的细胞状态 $\boldsymbol{c}_{t-1}$ 和当前时刻的候选细胞状态 $\tilde{\boldsymbol{c}}_t$ 进行调制。调制后的两部分信息通过加法进行整合，从而完成对细胞状态 $\boldsymbol{c}_t$ 的更新（即形成新的记忆）。这是 LSTM 最为核心的设计：它将需要保留的历史信息（即 $\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$）与需要写入的新信息（即 $\boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$）以线性相加的方式融合，从而为梯度在时间维度上的传播提供了相对 “通畅” 的路径，在一定程度

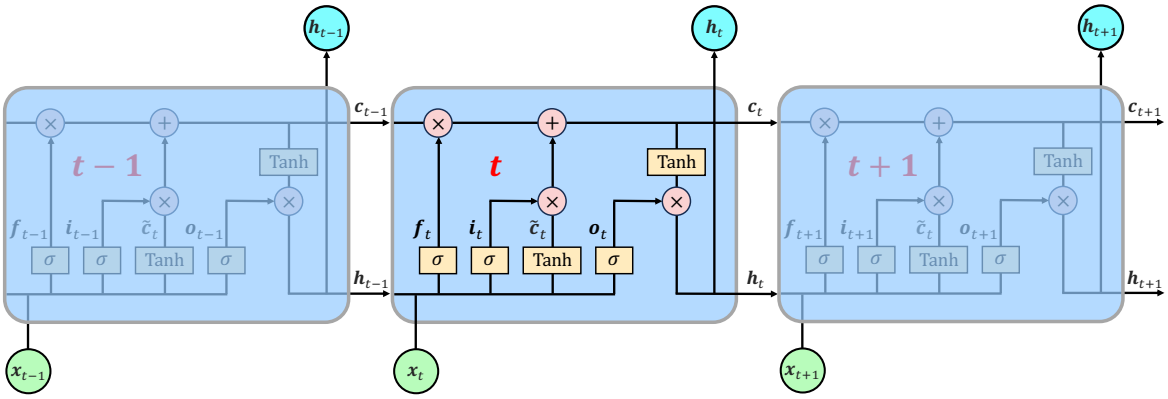


图 7.14: 长短期记忆网络（LSTM）单元的结构示意图。图中展示了由遗忘门 $\boldsymbol{f}_t$、输入门 $\boldsymbol{i}_t$ 和输出门 $\boldsymbol{o}_t$ 构成的门控机制，以及用于长程信息传递的细胞状态 $\boldsymbol{c}_t$。其中 σ 表示 Sigmoid 函数。

上缓解了梯度消失问题（这一点将在后文进一步详述）。最后，输出门 $\boldsymbol{o}_t$ 从经过 $\sigma_h(\cdot)$ 变换后的细胞状态 $\boldsymbol{c}_t$ 中提取与当前时刻 t 相关的输出 $\boldsymbol{h}_t$。由于存储在 $\boldsymbol{c}_t$ 中的信息并不一定全部有助于当前预测，输出门实现了对记忆内容的选择性过滤。例如，若使用 LSTM 建模由两个不同周期信号叠加而成的序列，在某一时刻可能只与其中一个信号的周期相关。此时，输出门会选择性地从 $\sigma_h(\boldsymbol{c}_t)$ 提取与当前相关的那部分信息，而与两个信号均相关的完整信息仍保存在细胞状态 $\boldsymbol{c}_t$ 中，以便在后续合适的时刻再被提取和利用。

从上述讨论可看出，LSTM 中的细胞状态 $\boldsymbol{c}_t$ 在功能上类似于基础循环神经网络中的隐状态 $\boldsymbol{h}_t$，二者都用于对历史信息与当前输入进行整合并加以存储。为了分析 LSTM 相较于基础循环神经网络为何能够更有效地缓解梯度消失问题，我们采用类似于公式7.31的分析方法，先根据公式7.43来计算相邻时刻细胞状态之间的雅可比矩阵：

$$\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} = \frac{\partial(\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1})}{\partial \boldsymbol{c}_{t-1}} + \frac{\partial(\boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t)}{\partial \boldsymbol{c}_{t-1}} \tag{7.45}$$

上式右侧的遗忘门 $\boldsymbol{f}_t$、输入门 $\boldsymbol{i}_t$ 以及候选细胞状态 $\tilde{\boldsymbol{c}}_t$ 均通过隐状态 $\boldsymbol{h}_{t-1}$ 间接地依赖于 $\boldsymbol{c}_{t-1}$。从严格推导的角度来看，每一项都需要进一步应用链式法则展开。然而，参照我们在基础循环神经网络中关于梯度消失的分析，这些通过非线性激活函数生成的门控和候选细胞状态项，其计算形式与公式7.25和公式7.26的组合（即为 $\boldsymbol{h}_t = \sigma(\mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{V}\boldsymbol{x}_t + \boldsymbol{b})$）一致。在跨越多个时间步进行传播时，其对应的雅可比矩阵连乘项同样受限于类似公式7.36的范数上界，从而导致其梯度贡献迅速衰减。因此，在后续分析中，我们忽略这些在时间维度上快速衰减的间接链式展开项，从而突出梯度传播的主要路径。在忽略这些快速衰减项之后，相邻时刻细胞状态之间的雅可比矩阵可近似简化为公式7.45中仅保留细胞状态 $\boldsymbol{c}_{t-1}$ 对其自身的偏导项。考虑到 $\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$ 为逐元素相乘运算，其对应的雅可比矩阵为以遗忘门向量 $\boldsymbol{f}_t$ 为对角元素的对角矩阵：

$$\frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{c}_{t-1}} \approx \frac{\partial(\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1})}{\partial \boldsymbol{c}_{t-1}} = \mathrm{diag}(\boldsymbol{f}_t) \tag{7.46}$$

在计算跨越多个时间步的细胞状态之间的梯度时，需要将从 $k+1$ 到 t 时刻的所有雅可比矩阵

进行连乘：

$$\prod_{i=k+1}^{t} \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{c}_{i-1}} \approx \prod_{i=k+1}^{t} \mathrm{diag}(\boldsymbol{f}_i) \tag{7.47}$$

对比公式7.35和7.47可以看出，LSTM 相较于基础循环神经网络（RNN）之所以能够有效地缓解梯度消失，并建模更长程的依赖关系，主要有以下几点原因：

- 跨多个时间步的梯度连乘项仅由门控向量构成的对角矩阵所组成，而不再包含权重矩阵。这避免了传统 RNN 中由于权重矩阵反复相乘所可能引起的梯度缩放或数值不稳定问题。特别是当权重矩阵谱范数小于 1 时，会进一步加剧梯度的衰减。
- 每一时刻的遗忘门向量并非固定常数，而是基于当时输入与前一时刻隐状态动态生成的。这种设计使门控系数能够随时间步自适应变化，从而避免了基础 RNN 梯度计算的雅可比矩阵中共享同一权重矩阵带来的系统性偏差。理论上模型可以通过学习，使得在需要长期记忆的序列区间内遗忘门的各元素接近 1，从而使梯度能够以近似无衰减（约等于 1）的方式传播，形成所谓的 “恒定误差传送带”（Constant Error Carousel）。
- 由于梯度连乘中不再包含权重矩阵，使得权重参数能够摆脱 “必须增大谱范数来补偿激活函数导数所带来的梯度收缩” 的束缚，这在一定程度上降低了训练的难度，并有助于提升模型的泛化能力。

若将梯度的传播比作车辆在道路上的行驶，传统 RNN 就像是一条每隔一段距离就设有减速带（激活函数导数）且路面摩擦系数固定（共享权重矩阵）的普通道路。无论初始动能（梯度信号）多强，在经过层层阻碍后，能量都会迅速损耗殆尽，导致信号无法触达远端。相比之下，LSTM 则构建了一条由 “智能道闸”（门控机制）动态调节的信息高速公路。当识别到重要信息时，遗忘门（道闸）会完全开启，形成一条阻力近乎为零的传输通道，使得梯度信号能够几乎无损地 “滑行” 至序列终点，从而更有效地捕捉长程依赖关系。在实际应用中，基础 RNN 通常只能较好地建模约数十个时间步范围内的依赖关系，而 LSTM 则能够在更长时间跨度上捕捉相应的依赖关系，通常可扩展至数百个时间步（或甚至更远）。需要注意的是，这里的 “时间步数” 并不是指整个输入序列的总长度，而是指序列中任意两个相关信息之间的间隔长度，即依赖关系所跨越的时间步距离。

如果训练样本的完整序列可以预先获得（而非实时生成的流式数据），LSTM 不仅可以按时间顺序从前向后处理序列，从而捕捉和整合历史信息，还可以按时间逆序从后向前处理序列，用于建模并融合未来信息。以语言数据为例，给定一段文本或一句话，正向读取能够整合当前词汇的上文信息，而反向读取则能融合该词汇的下文语义。通过构建双向 LSTM（Bidirectional LSTM），模型可以同时捕捉单词与其前后上下文之间的依赖关系。此外，LSTM 还可以通过堆叠（Stacking）的方式形成深层结构。具体而言，将底层 LSTM 在每一时刻的输出（即隐状态 $\boldsymbol{h}$）作为高层 LSTM 同一时刻的输入，这种层次化结构能够有效地建模更为抽象的高层语义特征。如图7.15所示，ELMo（Embeddings from Language Models）[57] 是将双向 LSTM 与多层堆叠结构相结合的典型模型，并被成功用于构建预训练语言模型。

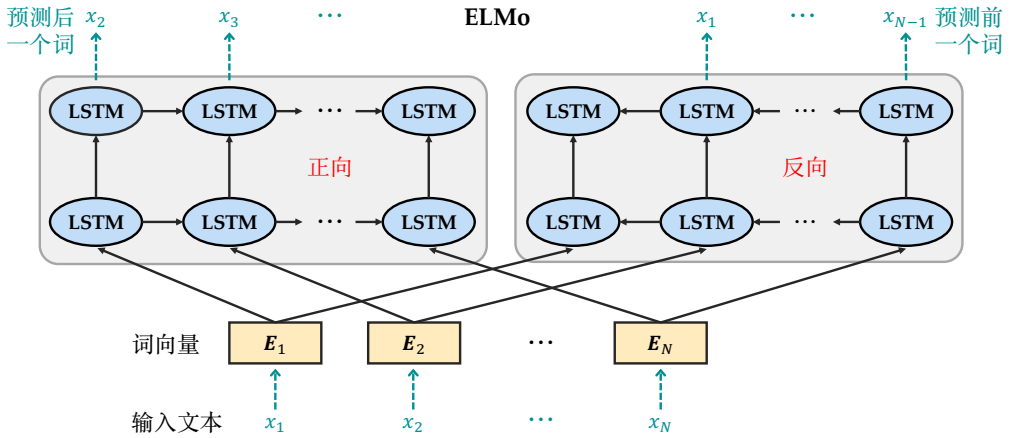


图 7.15: ELMo 模型结构示意图 [57]。该模型采用两层堆叠的双向 LSTM 结构，其中正向 LSTM 负责在给定前文的条件下预测下一个词，反向 LSTM 则负责在给定后文的条件下预测前一个词。正向与反向 LSTM 的参数彼此独立，但在各自的每一时刻均共享同一组参数。

传统的**语言模型**（Language Model）旨在建模词序列的概率分布，其核心任务是在给定一组连续词汇的前提下预测下一个单词，本质上是一种从前向后的正向预测。而 ELMo 将其扩展为双向语言模型，通过同时利用前文和后文信息来学习更丰富的上下文表示。语言模型的预训练是指利用大规模未经人工标注的文本对模型参数进行训练（初始化）。完成预训练后的模型能够捕捉丰富的通用语言特征，并可被迁移应用至词法分析、句法解析及语义建模等各类下游**自然语言处理**[2]任务。在预训练阶段，正向 LSTM 的训练目标是最大化序列的前向因式分解概率：

$$p(x_1, \ldots, x_N) = \prod_{i=1}^{N} p(x_i | x_1, \ldots, x_{i-1}) \tag{7.48}$$

其中 N 表示输入文本的长度（即单词总数），而 x_i 表示该文本序列中的第 i 个单词。相应地，反向 LSTM 的训练目标是最大化序列的反向因式分解概率：

$$p(x_1, \ldots, x_N) = \prod_{i=1}^{N} p(x_i | x_{i+1}, \ldots, x_N) \tag{7.49}$$

ELMo 模型的总体训练目标是同时最大化正向与反向语言模型的对数似然之和。需要注意的是，输入的单词首先会被映射为对应的向量表示（词向量），这些词向量（随机初始化）和两层双向 LSTM 的参数在预训练过程中会被联合学习与共同优化[3]。在利用大规模文本完成上述预训练之后，不同层的 LSTM 会逐渐形成层次化的语言表征。底层 LSTM 往往能够捕捉到更具象的语言学特征，如词性、词形变化及局部句法结构，而高层 LSTM 则更倾向于学习抽象的语义特征、词义消歧及更长程的依赖关系。在应用于下游自然语言处理任务时，ELMo 为每个单词生成的向量表示由正向与反向 LSTM 的隐状态拼接而成，这使得每个单词的表示都能够整合其完整的上

[2] 自然语言处理（Natural Language Processing, NLP）是人工智能和语言学的交叉学科，旨在研究计算机处理、理解与生成人类语言的技术。

[3] 原始的 ELMo 采用字符级卷积神经网络等组件构造词的上下文无关表示。

下文信息。需要注意的是，在预训练阶段并不会采用这种“双向拼接”的方式，其目的是避免泄露所需预测的信息，即防止模型在预测当前词时已经“看到”该词本身的信息。

由于 LSTM 单元的内部结构相对复杂、参数量较多，通常需要更多的训练样本以及更长的训练时间才能达到较好的优化效果。因此，研究者们在 LSTM 的基础上提出了若干简化的变体，而**门控循环单元**（Gated Recurrent Unit, GRU）[15] 是较具代表性的一种。GRU 的前向传播过程可以由以下公式描述：

$$\boldsymbol{z}_t = \sigma_g\left(\mathbf{U}_z\boldsymbol{h}_{t-1} + \mathbf{V}_z\boldsymbol{x}_t + \boldsymbol{b}_z\right) \tag{7.50}$$

$$\boldsymbol{r}_t = \sigma_g\left(\mathbf{U}_r\boldsymbol{h}_{t-1} + \mathbf{V}_r\boldsymbol{x}_t + \boldsymbol{b}_r\right) \tag{7.51}$$

$$\tilde{\boldsymbol{h}}_t = \sigma_h\left(\mathbf{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \mathbf{V}_h\boldsymbol{x}_t + \boldsymbol{b}_h\right) \tag{7.52}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \tag{7.53}$$

其中 $\boldsymbol{x}_t \in \mathbb{R}^D$ 表示时刻 t 的输入向量，其维度为 D；$\boldsymbol{h}_t \in \mathbb{R}^M$ 表示时刻 t 的隐状态（同时也是输出）向量，其维度为 M。$\boldsymbol{z}_t$ 和 $\boldsymbol{r}_t$ 分别表示更新门（Update Gate）和重置门（Reset Gate）的激活向量。更新门用于控制隐状态的更新比例，它决定了多少比例的历史信息 $\boldsymbol{h}_{t-1}$ 需要被保留，以及多少比例的新信息 $\tilde{\boldsymbol{h}}_t$ 需要被写入（见公式7.53）。重置门则用于确定前一时刻隐状态 $\boldsymbol{h}_{t-1}$ 对当前候选隐状态 $\tilde{\boldsymbol{h}}_t$ 的贡献程度（见公式7.52），它允许模型根据当前输入和历史信息有选择地“遗忘”与未来无关的信息。更新后的隐状态 $\boldsymbol{h}_t$ 按更新门 $\boldsymbol{z}_t$ 所设定的比例来整合上一时刻的隐状态 $\boldsymbol{h}_{t-1}$ 和当前时刻的候选隐状态 $\tilde{\boldsymbol{h}}_t$。$\mathbf{U} \in \mathbb{R}^{M\times M}$ 为状态转移矩阵，$\mathbf{V} \in \mathbb{R}^{M\times D}$ 为输入权重矩阵，而 $\boldsymbol{b} \in \mathbb{R}^M$ 为偏置向量。这些权重矩阵和偏置向量共有三组，它们的下标 z, r 和 h 分别表示更新门、重置门和候选隐状态计算时所涉及的参数。$\sigma_g(\cdot)$ 通常为 Sigmoid 函数，用于产生 $[0, 1]$ 之间的门控系数，而 $\sigma_h(\cdot)$ 通常为 Tanh 激活函数，用于生成候选隐状态。研究表明，在机器翻译、语音识别和音乐建模等时序任务中，GRU 的性能通常与 LSTM 相当。由于 GRU 结构更为简洁、参数量更少，在相同训练条件下往往具有更高的训练效率，即更易于优化与收敛。

7.5 残差神经网络

虽然多层深度结构能够通过特征的逐层复用与抽象，有效提取更高层次的语义表示，从而在一定程度上提升模型的表达能力与泛化性能，但随着网络深度的不断增加，又引发了新的问题。具体而言，深层网络仍然可能面临梯度消失或爆炸等问题 [4, 29]。即使采用适当的参数初始化（详见第7.6.4节）和归一化方法（详见第7.6.3节）来缓解这些问题，简单地增加网络深度仍可能使模型更加难以优化，甚至导致训练误差上升。当网络深度增加到一定程度时，更深网络的性能反而可能不如较浅的网络 [32]。这一现象被称为深层网络的退化问题（Degradation Problem）：即简单地增加网络深度，反而可能导致训练误差和测试误差同时上升。需要强调的是，这种性能下降并非由于过拟合所引起。过拟合通常表现为训练误差较低而测试误差升高，而退化问题则表现为训练误差与测试误差同时上升。这表明，对于更深层网络的有效学习与优化仍然存在困难和挑战。

在传统的深层神经网络中，梯度在反向传播过程中需要经过权重矩阵的层层连乘。若中间层权重矩阵的谱范数偏小，梯度将随着网络深度的增加而呈指数级衰减，即产生梯度消失现象，从而使深层参数难以得到有效更新。为缓解这一问题，2015 年提出的残差学习（Residual Learning）[33] 将网络中的模块（通常由若干层构成）的学习目标从“直接逼近目标映射”转变为“拟合输入与目标之间的残差”。换言之，在残差神经网络（Residual Neural Network）中，每个模块不再试图从输入直接“生成完整输出”，而是在前一模块输出（或阶段性预测）的基础上进行“增量偏差修正”，从而使深层网络更易于优化。

假设深度网络中的某一模块需要学习的目标映射为 $g(\boldsymbol{x})$，其中 $\boldsymbol{x}$ 为该模块的输入向量。若将该模块的学习目标改为拟合目标输出 $g(\boldsymbol{x})$ 与输入 $\boldsymbol{x}$ 之间的残差，则该模块可表示为：

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) - \boldsymbol{x} \tag{7.54}$$

因此，原始的目标映射可以改写为：

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{x} \tag{7.55}$$

此时，学习目标从直接建模目标映射 $g(\boldsymbol{x})$ 转化为学习一个残差映射 $f(\boldsymbol{x})$（Residual Mapping）与一个恒等映射 $\boldsymbol{x}$（Identity Mapping）之和。实现该学习目标转换的网络结构示意图如图7.16所示，其中引入的恒等映射通常被称为捷径连接（Shortcut Connection）或跳跃连接（Skip Connection），因为这一连接不仅在前向传播中提供了一条信息的直接传递路径，也在反向传播中为梯度提供了直达的“高速”通道。如图7.16所示的残差模块中，残差分支 $f(\boldsymbol{x})$ 由三层子网络构成：输入 $\boldsymbol{x}$ 首先经过一个线性变换层，随后通过 ReLU 激活函数进行非线性变换，最后再次经过线性变换后与恒等映射（即捷径分支）的 $\boldsymbol{x}$ 相加。需要注意的是，最后的相加操作为逐元素加法，因此残差分支 $f(\boldsymbol{x})$ 的输出维度通常需要与输入 $\boldsymbol{x}$ 的维度一致，这也是在网络结构中使用残差连接需要满足的基本条件。残差连接也可应用于卷积神经网络中，通常也要求残差模块输入与输出特征图在空间尺寸以及通道数上保持一致（或通过额外的投影变换进行匹配）。

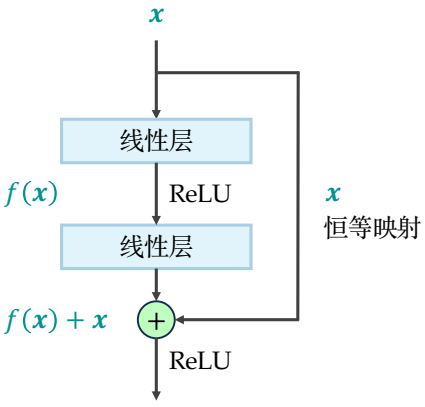


图 7.16: 残差模块结构示意图。该模块的残差分支 $f(\boldsymbol{x})$ 由三层子网络构成：输入 $\boldsymbol{x}$ 首先经过一个线性变换层（如普通线性层或卷积层），随后通过 ReLU 激活函数进行非线性变换，最后再次经过线性变换得到输出。该输出再与捷径分支上的恒等映射 $\boldsymbol{x}$ 进行逐元素相加，得到最终输出。

引入残差模块之所以能够使更深的神经网络更容易训练，并且在一定情况下可以通过增加网络的深度来提升性能，主要有以下两个原因：

- 假设残差模块的输出为 $g(\boldsymbol{x})$，其学习的残差映射为 $f(\boldsymbol{x})$，则输出可表示为 $g(\boldsymbol{x}) = f(\boldsymbol{x})+\boldsymbol{x}$。在反向传播过程中，损失函数 $\mathcal{L}$ 关于该模型输入 $\boldsymbol{x}$ 的梯度为：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \frac{\partial \mathcal{L}}{\partial g(\boldsymbol{x})} \cdot \frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \tag{7.56}$$

其中上式右侧的第二项为：

$$\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} + \mathbf{I} \tag{7.57}$$

上式（与 LSTM 中相邻细胞状态间梯度计算公式7.45类似）来自恒等映射分支的单位矩阵 $\mathbf{I}$ 至关重要。即便残差分支的梯度 $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ 过小而趋近于零，整体梯度仍然可以通过恒等映射分支直接进行反向传播，而不必完全依赖残差分支。这相当于在网络中构建了一条梯度传播的“高速公路”，使得梯度能够更加有效地跨层传递至浅层网络，从而缓解深层网络中的梯度消失问题。

- 将学习目标从“直接逼近目标映射”转变为“拟合输入与目标之间的残差”，从而显著降低了优化难度。以数值逼近为例：假设最终目标是将输入映射为 9.456。在传统结构中，每个模块都需尝试“完整”映射使得最终输出逼近目标值。而在残差架构下，第一个模块可先将输入映射至 9，后续模块则仅需在前一模块输出 9 的基础上补充残差 0.4 以达到 9.4，再由下一模块补充残差 0.05 以达到 9.45，如此这般逐步逼近目标值。在这种逐层修正的机制下，随着残差模块数量（即网络深度）的增加，网络模型能够以逐层“修正误差”的方式不断提高预测精度，同时降低单个模块的学习负担与难度。

残差模块的引入使得训练深达数百层甚至上千层的神经网络成为可能。图7.17展示了一个经典的包含 152 层的残差神经网络（ResNet-152），并将其应用于光学字符识别（Optical Character Recognition）任务。光学字符识别主要是从图像中识别并提取其中包含的文字内容（包括版面结构）。该网络的输入为 224×224 像素的图像。网络首先通过一个 7×7 的卷积核生成 64 幅特征图。由于该卷积层的步长为 2，输出的分辨率降至 112×112。随后，网络采用核大小为 3×3、步长为 2 的最大池化层（四周填充为 1）进行下采样，得到尺寸为 $56 \times 56 \times 64$ 的特征张量。

接下来，网络依次经过四个核心阶段（在图7.17中分别以黄、绿、红、青四色标示）。每个阶段由多个残差模块堆叠而成，而每个残差模块通常包含三层卷积。该类残差模块采用典型的“瓶颈结构”（Bottleneck Design）：首先通过 1×1 卷积压缩通道数（例如在绿色阶段将通道数从 256 降至 128），之后在较低通道数上使用较大的 3×3 卷积核来扩大感受野，并提取空间特征（避免了直接在高维通道上使用大卷积核所带来的参数量剧增），最后再通过 1×1 卷积恢复并扩充通道数，以提升特征捕捉的丰富程度。这里的“瓶颈结构”是指残差模块在中间层对特征通道进行压缩，使整体结构呈现“两端通道数较大、中间通道数较小”的形态，类似于中间收窄的瓶颈或沙漏结构。这种设计能够在保持或提升模型表达能力的同时，显著减少参数量与计算代价。

每个残差模块由上述三层卷积构成残差分支，并配有一条对应的捷径分支。与基础残差模块不同，此处的捷径连接需要额外的参数来完成映射。这是因为残差模块的输入与输出在通道

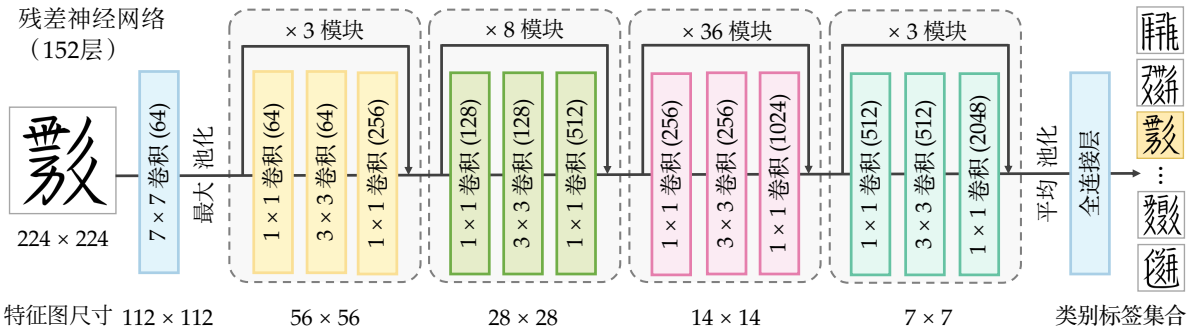


图 7.17: 经典的 152 层残差神经网络（ResNet-152）的结构示意图。图中类似 “7 × 7 卷积 (64)” 表示使用 7 × 7 卷积核生成 64 个通道（特征图）的卷积层。网络主要由四个核心阶段构成（分别以黄、绿、红、青四色表示），每一阶段由多个残差模块堆叠而成。每个残差模块由三层卷积构成的残差分支以及一条对应的捷径分支组成。残差模块上方标注的 “×3” 表示该残差模块在对应阶段中重复堆叠的次数。该示例网络被应用于西夏文字识别任务。

数上不一致，因此需要在捷径分支上采用核为 1 × 1 卷积进行投影变换，以对齐特征图的空间尺寸和通道数，并确保残差分支与捷径分支的输出可以进行逐元素相加。需要注意的是，捷径分支中的投影变换通常不包含非线性激活函数，且往往不使用偏置项，因此本质上是一个用于维度匹配的线性变换。因此，尽管该分支引入了额外参数，但由于其结构简单且提供跨层连接，仍然能够成为梯度反传的捷径。

在网络的末端，采用 7 × 7 的平均池化层进行降维，将特征图转化为 1 × 1 × 2048 的张量。最后，通过全连接层将该 2048 维特征向量映射到目标任务所需的类别数。

7.6 网络优化方法

由于深度神经网络的优化本质上是一个复杂的高维非凸优化问题，其参数空间不仅存在众多的局部极小值，还广泛分布着鞍点、陡峭悬崖和平坦区域等极端损失地貌（Loss Landscape）。这些因素严重影响了优化算法的收敛效率与寻优能力，从而使整个训练过程具有较大的挑战性。在网络结构与损失函数确定的前提下，训练策略与调参技巧往往会直接影响模型的最终性能，甚至可能带来显著差异。因此，深度神经网络的训练有时被形象地称为是一个 “艺术多于科学” 的过程。除了了解神经网络的主要类型与结构，以及如何针对具体任务设计合适的损失函数外，系统掌握有效的优化方法同样至关重要。接下来，我们将从参数优化算法、数据预处理、参数初始化、归一化方法、正则化方法和超参数选择这六个方面，对其代表性方法进行简要介绍。

7.6.1 参数优化算法

在参数优化算法方面，最基础的方法是**随机梯度下降**（Stochastic Gradient Descent, SGD）。在实际应用中，该算法通常与小批量（Mini-batch）策略相结合：即在每次迭代中，从训练集中随机抽取包含 B 个样本的子集来近似估计损失函数 $\mathcal{L}(\boldsymbol{\theta})$ 关于模型参数 $\boldsymbol{\theta}$ 的梯度。第 t 次迭代

时的批量梯度 $\mathbf{g}^{(t)}$ 计算公式如下：

$$\mathbf{g}^{(t)} = \frac{1}{B}\sum_{i=1}^{B}\nabla\mathcal{L}_i(\boldsymbol{\theta}^{(t)}) \tag{7.58}$$

其中 $\nabla\mathcal{L}_i(\boldsymbol{\theta}^{(t)})$ 表示基于第 i 个样本在当前参数下计算的梯度。在获得该小批量样本的梯度估计之后，参数按照如下方式进行更新：

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t)} - \eta\mathbf{g}^{(t)} \tag{7.59}$$

其中 η 是学习率，它决定了参数更新的步长。如前所述，批量大小 B 不宜过小，否则梯度估计容易受随机噪声的影响，进而因梯度估计的方差过大而导致训练过程的不稳定；但也不宜过大，否则可能降低优化过程的随机性，从而削弱模型跳出鞍点或较差局部极小值的能力。学习率 η 的选取通常与批量 B 的大小呈正比。这是因为，随着批量中样本数量的增加，虽然梯度估计的期望保持不变，但其方差会随之减小。由于梯度估计更为稳健，我们可以使用较大的学习率来加速收敛，从而提升训练效率。

在神经网络的训练过程中，学习率 η 通常不会保持恒定，而是会随着训练的推进逐渐减小。这种策略被称为学习率衰减（Decay）或退火（Annealing）。采用这种策略主要有以下两个方面的原因：

- 在训练初期，尤其是在参数随机初始化的情况下，模型通常距离 “最优解”（更准确地说是某个较优的局部极小值）较远。此时，采用较大的学习率有助于模型快速跨越平坦区域或局部障碍，从而加速趋近该局部极小值。然而，当模型进入该极小值附近的 “谷底” 区域时，若学习率依然取较大值，参数更新步长可能超过该区域内进行有效优化的尺度，导致参数在谷底两侧反复振荡，甚至因步长过大而 “跳出” 该局部极小值区域。
- 在采用小批量随机梯度下降时，每次迭代得到的梯度估计不可避免地包含随机噪声。在训练后期，随着模型逐渐接近收敛，真实的梯度将趋近于零。若仍保持较大的学习率，梯度中的噪声成分会被放大，导致参数在收敛点附近进行无意义的 “随机游走”，难以实现精细收敛。通过在训练后期降低学习率，可以有效减小由于噪声引起的更新方差，使模型能够更加平稳地收敛至目标局部最小值。

基于上述原因，学习率可以采用如下的线性衰减策略：

$$\eta^{(t)} = (1-\alpha)\eta^{(0)} + \alpha\eta^{(\tau)} \tag{7.60}$$

其中比例系数 $\alpha = \frac{t}{\tau}$ 随迭代次数 t 线性增加（$0 \le t \le \tau$）。上述公式表示学习率在训练过程中从初始值 $\eta^{(0)}$ 线性下降至最终值 $\eta^{(\tau)}$。当迭代次数 $t \ge \tau$ 后，学习率将保持在 $\eta^{(\tau)}$ 不变。在实践中，最终学习率 $\eta^{(\tau)}$ 通常设定为初始学习率 $\eta^{(0)}$ 的 1% 左右。

在训练的初期，梯度的数值通常较大且方向波动较强，尤其是在参数随机初始化的情况下，各参数之间尚未形成有效的协同关系。此时若直接采用较大的学习率，容易导致参数更新幅度过大，从而引起训练过程的不稳定。为了缓解这一问题，在实践中常采用预热（Warm-up）策略，即在训练开始的若干个迭代步中使用较小的学习率，等梯度下降到一定程度后再恢复到初始的学习率 $\eta^{(0)}$。并在此基础上，再结合相应的学习率衰减策略，使学习率在后续训练过程中逐渐

减小。由此，学习率在整体上呈现出“先升后降”的变化趋势。

在上述随机梯度下降法中，参数更新仅依赖于当前小批量样本所估计的即时梯度，因而未能充分利用优化路径中的长期趋势。从直觉上看，如果当前梯度方向与历史梯度方向较为一致，说明在多个迭代步骤中梯度大致指向同一方向，此时可以在该方向上适当增大更新幅度；反之，若当前梯度方向与历史方向差异较大甚至相反，则应在该方向上采取更加保守的更新策略，以避免不稳定的参数振荡。为了实现这一目标，可以对历史梯度进行加权累积，并与当前梯度共同用于参数更新，这种方法称为**带动量的随机梯度下降法**。在物理学中，动量（Momentum）被定义为物体质量与速度的乘积，体现了物体维持其运动状态的惯性[4]。在优化过程中，我们可以将累积的梯度视为速度（Velocity），而将当前计算出的即时梯度视为作用在物体上的力（Force）。根据物理学定律，力作用于物体产生加速度，进而改变其运动速度。当连续多个迭代步的梯度方向保持一致时，参数更新的“速度”会迅速累积。模型应在该确定性较强的方向上“加速前进”，从而利用“惯性”快速冲过较浅的局部极小值或梯度极其平缓的区域。而在梯度方向不稳定或随机噪声较大的情况下，动量机制能够通过加权累积所带来的平滑效应抵消掉部分矛盾的分量，从而在缓解振荡的同时，显著提升整体收敛效率。

在带动量的随机梯度下降法中，参数的更新不再仅取决于当前的梯度，而是通过维护一个“速度”向量 $\Delta\boldsymbol{\theta}^{(t)}$ 来积累历史更新的方向，其递推形式可表示为：

$$\Delta\boldsymbol{\theta}^{(t)} = \gamma\Delta\boldsymbol{\theta}^{(t-1)} - \eta\mathbf{g}^{(t)} \tag{7.61}$$

其中 $\gamma \in [0,1)$ 称为动量因子（通常设为 0.9），它决定了历史更新方向对当前更新的影响程度。η 表示学习率，而 $\mathbf{g}^{(t)}$ 表示当前迭代步中由小批量样本得到的梯度估计。随后，我们利用计算得到的“速度”向量完成参数更新：

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t)} + \gamma\Delta\boldsymbol{\theta}^{(t-1)} - \eta\mathbf{g}^{(t)} \tag{7.62}$$

在基本的随机梯度下降法中，所有参数共享同一个全局学习率。然而在实际应用中，不同参数对应的梯度尺度往往存在显著差异：某些参数的梯度较大，从而更新幅度较大；而另一些参数的梯度较小，其更新幅度也相对有限。若统一采用相同的学习率，可能导致前者更新过于剧烈，而后者更新不足，从而影响整体的优化效果。**AdaGrad 算法**（Adaptive Gradient Algorithm）针对这一问题提出了相应的解决方法 [23]。该算法的核心思想是为每个参数维护一个自适应的学习率。AdaGrad 算法通过累积各参数历史梯度的平方和，来衡量该参数在优化过程中被更新的“频率”或“强度”。在更新时，算法根据该平方和对学习率进行动态缩放：梯度平方和较大的参数，其学习率会被相应减小；而梯度平方和较小的参数，则会保持相对较大的学习率。

具体而言，在第 t 次迭代时，AdaGrad 算法首先计算各参数历史梯度平方的累积：

$$\mathbf{G}^{(t)} = \sum_{i=1}^{t} \mathbf{g}^{(i)} \odot \mathbf{g}^{(i)} \tag{7.63}$$

其中符号 $\odot$ 表示逐元素乘法，而 $\mathbf{g}^{(i)}$ 为第 i 次迭代时的梯度向量。由此得到的 $\mathbf{G}^{(t)}$ 反映了各参

[4]准确地说，惯性（Inertia）是物体抗拒其运动状态被改变的性质，并由其质量唯一量度。

数在历史迭代中的梯度平方和。然后，参数的更新量 $\Delta\boldsymbol{\theta}^{(t)}$ 按照如下公式计算：

$$\Delta\boldsymbol{\theta}^{(t)} = -\frac{\eta}{\sqrt{\mathbf{G}^{(t)}+\epsilon}} \odot \mathbf{g}^{(t)} \tag{7.64}$$

其中 η 为全局学习率，ϵ 是一个非常小的正数（通常取值为 10^{-7} 或 10^{-8}），用于避免分母过小而引发数值不稳定问题。需要注意的是，公式7.64中的除法、平方根和乘法运算均以逐元素方式进行。上式中的 $\frac{\eta}{\sqrt{\mathbf{G}^{(t)}+\epsilon}}$ 可视为一种**自适应学习率**，它根据历史梯度平方的累积对全局学习率进行缩放，使得梯度累计较大的参数对应较小的更新步长，而梯度累计较小的参数则保持相对较大的更新步长。

然而，AdaGrad 算法存在一个显著的缺陷：由于有效学习率在迭代过程中呈单调递减趋势，在训练后期，有效学习率往往会变得过小，导致参数无法得到充分优化。这一问题源于 $\mathbf{G}^{(t)}$ 对历史梯度平方进行了无限制累加。随着迭代次数 t 的增加，分母项 $\sqrt{\mathbf{G}^{(t)}}+\epsilon$ 会不断单调递增，迫使有效学习率最终趋近于零，导致模型在尚未达到最优解时就过早收敛。**RMSprop 算法**针对 AdaGrad 在后期学习率过快衰减的问题进行了改进 [75]，其核心思想是借鉴动量方法，用梯度平方的指数移动平均来替代简单的累加。这种改进不仅避免了分母的无限增长，还赋予了算法一定的“遗忘”能力，使其能够自动弱化较早时期的梯度信息，从而使学习率能够根据当前和近期的梯度尺度进行动态调整。在第 t 次迭代中，梯度平方的指数平均值 $\mathbf{G}^{(t)}$ 的计算公式如下：

$$\mathbf{G}^{(t)} = \gamma\mathbf{G}^{(t-1)} + (1-\gamma)\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)} \tag{7.65}$$

其中 $\gamma \in [0,1)$ 为衰减率，通常取值为 0.9。参数的更新量 $\Delta\boldsymbol{\theta}^{(t)}$ 仍然按公式7.64进行计算。

Adam（Adaptive Moment Estimation）算法结合了动量法和 RMSprop 算法的优点 [44]。它既能够通过对梯度的累积来捕捉参数更新的长期趋势，又能够基于梯度平方实现自适应的学习率调整，使其成为目前深度学习领域应用最为广泛的优化算法之一。Adam 算法通过维护两个向量来更新参数：一是梯度的一阶矩估计（即梯度的指数加权平均），用于捕捉梯度的方向和动量；二是梯度的二阶矩估计（即梯度平方的指数加权平均），用于捕捉梯度的尺度及其变化。对于第 t 次迭代，这两个向量的递推计算公式如下：

$$\mathbf{M}^{(t)} = \gamma_1\mathbf{M}^{(t-1)} + (1-\gamma_1)\mathbf{g}^{(t)} \tag{7.66}$$

$$\mathbf{G}^{(t)} = \gamma_2\mathbf{G}^{(t-1)} + (1-\gamma_2)\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)} \tag{7.67}$$

其中 $\gamma_1, \gamma_2 \in [0,1)$ 是衰减率，在实际应用中通常分别设为 0.9 和 0.999。$\mathbf{g}^{(t)}$ 为当前步的小批量梯度估计。梯度的一阶矩估计 $\mathbf{M}^{(t)}$ 和二阶矩估计 $\mathbf{G}^{(t)}$ 通常被初始化为零向量。然而，这种初始化方式会导致矩估计的值在训练初期严重偏向于零。尤其当衰减率 γ_1 和 γ_2 较接近 1（即倾向于保留更多历史信息）时，这种初始阶段的偏差尤为显著。为了消除初始化带来的系统性偏差，Adam 算法使用以下公式对原矩估计进行修正，使其成为无偏估计：

$$\hat{\mathbf{M}}^{(t)} = \frac{\mathbf{M}^{(t)}}{1-\gamma_1^t} \tag{7.68}$$

$$\hat{\mathbf{G}}^{(t)} = \frac{\mathbf{G}^{(t)}}{1-\gamma_2^t} \tag{7.69}$$

上述修正的合理性在于：通过分母项 $1-\gamma^t$ 补偿了指数加权平均在初始阶段因零初始化而导致的数值亏损，从而确保修正后的矩估计在统计期望上与真实的矩保持一致（假设在局部窗口内梯度服从相同的分布）。随着迭代次数 t 的增加，修正因子 $1-\gamma^t$ 将趋近于 1，此时偏差修正的影响逐渐消退，算法平稳过渡至标准的指数加权平均形式。Adam 算法中参数的更新量 $\Delta\boldsymbol{\theta}^{(t)}$ 按如下公式计算：

$$\Delta\boldsymbol{\theta}^{(t)} = -\frac{\eta}{\sqrt{\hat{\mathbf{G}}^{(t)}} + \epsilon} \odot \hat{\mathbf{M}}^{(t)} \tag{7.70}$$

其中 η 为全局学习率，在实际应用中通常取 0.001，而 ϵ 为一个非常小的正数，通常取 10^{-8}。在深度学习中被广泛采用的 **AdamW**（Adam with Decoupled Weight Decay）是在 Adam 算法的基础上引入权重衰减策略后的一种改进 [50]，我们将在第7.6.5节中对其进行详细介绍。

此前讨论的优化算法通常只对每个参数的更新量进行独立的缩放，如 AdaGrad 算法通过为每个参数维护其历史梯度平方的累积，实现学习率的自适应调整。然而，在神经网络中，权重矩阵内部的参数往往存在较强的相关性。如果忽略这种相关性，可能导致冗余更新，以及由于梯度矩阵奇异值分布不均而引发的振荡或收敛缓慢等问题。冗余更新现象表现为：若权重矩阵中不同特征分量高度相关，则其梯度方向将趋于一致。在传统的逐元素优化框架下，所涉及的参数会在相同的方向上被重复且过度地更新，从而导致表示能力的退化（其他有效更新方向被相对抑制）与计算资源的浪费。此外，对于权重矩阵 $\mathbf{W}$，设其梯度 $\mathbf{G}$ 可进行如下奇异值分解：

$$\mathbf{G} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top \tag{7.71}$$

其中 $\mathbf{U}$ 和 $\mathbf{V}$ 分别为左、右奇异向量构成的正交矩阵，$\boldsymbol{\Sigma}$ 为对角矩阵，其对角线上的非负元素即为矩阵 $\mathbf{G}$ 的奇异值。在实际优化过程中，损失函数在不同方向上的曲率可能存在较大差异，使其在参数空间中呈现 “细长山谷” 的形状，导致优化算法容易在陡峭方向上发生振荡，而在平坦方向上进展缓慢。此外，梯度更新矩阵的奇异值分布也可能高度不均，即某些方向对应的奇异值较大，而另一些方向则较小，使得参数更新主要集中在少数方向上。**Muon**（Momentum Orthogonalized by Newton-Schulz）算法的基本思想是将矩阵 $\mathbf{G}$ 映射为一个正交矩阵 $\mathbf{O} = \mathbf{U}\mathbf{I}\mathbf{V}^\top$（其中 $\mathbf{I}$ 是单位阵），并利用该正交矩阵对权重参数进行更新。该操作相当于在保持奇异向量方向不变的同时，将更新矩阵的奇异值调整为相同的大小，从而使参数在不同奇异方向上的更新幅度更加均衡，避免少数方向主导参数更新。同时，该类更新在一定程度上有助于控制权重矩阵的谱范数，避免在奇异值较大的方向上产生过度拉伸，从而提高训练的稳定性。

与动量法类似，Muon 算法 [43] 首先为每个权重矩阵维护一个动量（或称 “速度”）矩阵 $\mathbf{M}^{(t)}$，其递推形式可表示为：

$$\mathbf{M}^{(t)} = \gamma\mathbf{M}^{(t-1)} + \mathbf{G}^{(t)} \tag{7.72}$$

$$\mathbf{G}^{(t)} = \frac{1}{B}\sum_{i=1}^{B}\nabla\mathcal{L}_i(\mathbf{W}^{(t)}) \tag{7.73}$$

其中 $\gamma \in [0,1)$ 为动量因子，$\mathbf{G}^{(t)}$ 表示在第 t 次迭代步中，利用大小为 B 的小批量样本计算得到的损失函数 $\mathcal{L}$ 关于权重矩阵 $\mathbf{W}^{(t)}$ 的平均梯度估计。然后，算法利用牛顿-舒尔茨迭代（Newton-

表 7.1: 代表性参数优化算法总结

参数优化算法	参数更新量计算公式	
随机梯度下降法（SGD）	$\Delta\boldsymbol{\theta}^{(t)} = -\eta\mathbf{g}^{(t)}$	最基本的参数优化方法
带动量的随机梯度下降法	$\Delta\boldsymbol{\theta}^{(t)} = \gamma\Delta\boldsymbol{\theta}^{(t-1)} - \eta\mathbf{g}^{(t)}$	考虑梯度长期变化趋势
AdaGrad 算法	$\Delta\boldsymbol{\theta}^{(t)} = -\frac{\eta}{\sqrt{\mathbf{G}^{(t)}+\epsilon}} \odot \mathbf{g}^{(t)}$ $\mathbf{G}^{(t)} = \sum_{i=1}^{t} \mathbf{g}^{(i)} \odot \mathbf{g}^{(i)}$	梯度平方累加对学习率进行动态缩放实现自适应学习率
RMSprop 算法	$\Delta\boldsymbol{\theta}^{(t)} = -\frac{\eta}{\sqrt{\mathbf{G}^{(t)}+\epsilon}} \odot \mathbf{g}^{(t)}$ $\mathbf{G}^{(t)} = \gamma\mathbf{G}^{(t-1)} + (1-\gamma)\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$	在 AdaGrad 基础上，将梯度平方累加改为指数移动平均
Adam 算法	$\Delta\boldsymbol{\theta}^{(t)} = -\frac{\eta}{\sqrt{\hat{\mathbf{G}}^{(t)}+\epsilon}} \odot \hat{\mathbf{M}}^{(t)}$ $\hat{\mathbf{M}}^{(t)} = \mathbf{M}^{(t)}/(1-\gamma_1^t)$ $\hat{\mathbf{G}}^{(t)} = \mathbf{G}^{(t)}/(1-\gamma_2^t)$ $\mathbf{M}^{(t)} = \gamma_1\mathbf{M}^{(t-1)} + (1-\gamma_1)\mathbf{g}^{(t)}$ $\mathbf{G}^{(t)} = \gamma_2\mathbf{G}^{(t-1)} + (1-\gamma_2)\mathbf{g}^{(t)} \odot \mathbf{g}^{(t)}$	相当于动量法和 RMSprop 的结合
AdamW 算法	$\Delta\boldsymbol{\theta}^{(t)} = -\eta\left(\frac{1}{\sqrt{\hat{\mathbf{G}}^{(t)}+\epsilon}} \odot \hat{\mathbf{M}}^{(t)} + \lambda\boldsymbol{\theta}^{(t)}\right)$ λ 为权重衰减系数，其它同 Adam 算法	在 Adam 基础上引入权重衰减
Muon 算法	$\Delta\mathbf{W}^{(t)} = -\eta\mathbf{O}^{(t)}$ $\mathbf{O}^{(t)} = \text{Newton-Schulz}(\mathbf{M}^{(t)})$ $\mathbf{M}^{(t)} = \gamma\mathbf{M}^{(t-1)} + \mathbf{G}^{(t)}$ $\mathbf{G}^{(t)} = \frac{1}{B}\sum_{i=1}^{B} \nabla\mathcal{L}_i(\mathbf{W}^{(t)})$	用正交化后的梯度矩阵对参数进行更新，仅适用于权重矩阵
注：梯度以小批量样本估计，即 $\mathbf{g}^{(t)} = \frac{1}{B}\sum_{i=1}^{B} \nabla\mathcal{L}_i(\boldsymbol{\theta}^{(t)})$		

Schulz Iteration）[7, 35] 将动量矩阵 $\mathbf{M}^{(t)}$ 转换为其对应的正交矩阵（或接近正交）：

$$\mathbf{O}^{(t)} = \text{Newton-Schulz}(\mathbf{M}^{(t)}) \tag{7.74}$$

值得注意的是，牛顿-舒尔茨迭代在实现矩阵正交化的过程中仅涉及矩阵乘法运算。相比于涉及奇异值分解等运算的传统正交化方法，该迭代法更适合在现代 GPU 硬件上进行并行计算，因此在这些硬件上运行时具有较高的计算效率。最后，Muon 算法利用正交化后的矩阵对权重参数 $\mathbf{W}$ 进行更新：

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \Delta\mathbf{W}^{(t)} = \mathbf{W}^{(t)} - \eta\mathbf{O}^{(t)} \tag{7.75}$$

其中 η 为学习率，亦可根据实际需求替换为自适应学习率或应用衰减策略。由于 Muon 算法中包含矩阵的正交化过程，目前仅适用于二维权重矩阵的优化（例如全连接层或卷积层的权重）。对于标量参数或一维向量参数（如偏置项），通常仍需配合 AdamW 或随机梯度下降法等主流优化算法进行联合训练。表7.1总结了上述代表性参数优化算法中参数更新量的计算方式及其核心设计思想。

7.6.2 数据预处理

在介绍数据预处理方法之前，我们先讨论输入数据的分布如何影响损失函数的形态，以及对参数优化过程的影响。为便于直观理解，我们以最简单的二元线性函数 $f(\boldsymbol{x})$ 为例进行说明：

$$f(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + b \tag{7.76}$$

其中 x_1 和 x_2 表示输入向量 $\boldsymbol{x}$ 在两个维度上的特征取值，w_1 和 w_2 是与之对应的权重参数，而 b 为偏置项。由于偏置项仅对输出值进行整体平移，并不影响损失函数在参数空间的形状与特性。为了简化分析，我们在此将其设为零（即 $b = 0$），则 $f(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 = \boldsymbol{w}^\top \boldsymbol{x}$。给定包含 N 个样本的训练数据集 $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$，并采用以下平方误差作为损失函数：

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i \right)^2 \tag{7.77}$$

以上线性模型的平方误差损失函数是一个凸二次函数。然而，损失函数可能是非凸的，其在不同参数位置的曲率也可能存在较大差异。因此，为了分析不同曲率对参数优化过程的影响，我们不再局限于上述线性模型，而是考虑一般的二阶连续可微损失函数 $\mathcal{L}(\boldsymbol{w})$。为了分析损失函数在参数更新过程中的变化趋势，我们在当前参数 $\boldsymbol{w}$ 处对损失函数进行二阶泰勒展开：

$$\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \nabla\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w} + \frac{1}{2}\Delta\boldsymbol{w}^\top \mathbf{H} \Delta\boldsymbol{w} + o\left(\|\Delta\boldsymbol{w}\|\right)^2 \tag{7.78}$$

其中 $\nabla\mathcal{L}(\boldsymbol{w})$ 为一阶梯度，$\mathbf{H} = \nabla^2\mathcal{L}(\boldsymbol{w})$ 为海森矩阵（即损失函数关于参数的二阶导数矩阵），而 $o\left(\|\Delta\boldsymbol{w}\|\right)^2$ 表示高阶无穷小项。该泰勒展开式提供了损失函数的二阶近似，刻画了当参数从 $\boldsymbol{w}$ 沿方向 $\Delta\boldsymbol{w}$ 移动时，损失函数的局部变化情况。一阶梯度描述了损失曲面的“局部倾斜方向”，其负梯度方向对应局部下降最快的方向；而海森矩阵 $\mathbf{H}$ 则描述了该损失曲面的“曲率”，反映了梯度的变化速率。

考虑损失函数 $\mathcal{L}(\boldsymbol{w})$ 在当前参数 $\boldsymbol{w}$ 处，以学习率 η 沿负梯度方向 $\Delta\boldsymbol{w} = -\eta\nabla\mathcal{L}(\boldsymbol{w})$ 更新后的变化情况：

$$\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) - \mathcal{L}(\boldsymbol{w}) = \underbrace{-\eta\|\nabla\mathcal{L}(\boldsymbol{w})\|^2}_{\text{一阶项}} + \underbrace{\frac{1}{2}\eta^2 \nabla\mathcal{L}(\boldsymbol{w})^\top \mathbf{H} \nabla\mathcal{L}(\boldsymbol{w})}_{\text{二阶项}} + o\left(\|\Delta\boldsymbol{w}\|\right)^2 \tag{7.79}$$

上式中的一阶项始终为负（或零），表示沿负梯度方向更新所带来的损失下降，这也是梯度下降法有效的根本原因。二阶项则刻画了局部曲率对下降过程的修正，其影响取决于海森矩阵 $\mathbf{H}$ 的性质。根据参数所处区域的几何特征，可分为以下三种典型情况：

- 若当前参数处于损失函数具有负曲率的区域（如局部极大值附近或形似“山峰”的区域），则海森矩阵 $\mathbf{H}$ 可能为负定或半负定，即 $\mathbf{H}$ 的所有特征值都小于或等于零。在这种情况下，$\nabla\mathcal{L}(\boldsymbol{w})^\top \mathbf{H} \nabla\mathcal{L}(\boldsymbol{w}) \leq 0$。因此，当沿负梯度方向更新参数时，二阶泰勒展开中的二阶项与一阶项均为非正。这意味着负曲率可能进一步促进损失函数的下降，使损失函数的实际下降幅度大于仅考虑一阶项时的近似预测值。换言之，负曲率在一定程度上起到了“加速”损失函数下降的作用。
- 若当前参数处于损失函数的局部凸区域（如局部极小值附近），则海森矩阵 $\mathbf{H}$ 是正定或

半正定的，即所有特征值都大于或等于零（$\nabla\mathcal{L}(\boldsymbol{w})^\top\mathbf{H}\nabla\mathcal{L}(\boldsymbol{w}) \geq 0$）。此时，沿负梯度方向更新参数所产生的二阶项为非负值，会在一定程度上抵消一阶项带来的损失下降。当学习率过大或局部曲率较大时，二阶项甚至可能超过一阶项的下降幅度，导致损失函数不降反升。此外，如果海森矩阵在不同方向上的特征值差异较大，损失函数便可能呈现狭长的“峡谷”形状，形成所谓的病态性（Ill-conditioning）问题。在深度网络训练中，往往会观察到梯度范数并未显著减小，但二阶项却因曲率变陡而剧增。为了防止因二阶项成为主导项而造成损失函数不降反升（即越过极小值点导致发散），不得不减小学习率 η，但这会导致整体学习速度下降。

- 若当前参数位于鞍点附近，则海森矩阵 $\mathbf{H}$ 是不定的，既包含正特征值，也包含负特征值。二阶项的正负取决于梯度向量 $\nabla\mathcal{L}(\boldsymbol{w})$ 与 $\mathbf{H}$ 的哪些特征向量方向的重合度更高。

由于损失函数关于参数的二阶导数的海森矩阵 $\mathbf{H}$ 是对称的，我们可以对其进行**特征分解**（Eigendecomposition）或称**谱分解**（Spectral Decomposition）：

$$\mathbf{H} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top \tag{7.80}$$

其中 $\mathbf{Q}$ 为正交矩阵，其列向量为矩阵 $\mathbf{H}$ 的特征向量，而 $\boldsymbol{\Lambda}$ 为对角矩阵，其对角线元素为 $\mathbf{H}$ 的特征值。因为海森矩阵的特征值刻画了损失函数在各特征向量方向上的曲率。当最大特征值 $\lambda_{\max}$ 与最小特征值 $\lambda_{\min}$ 相差较大时，损失函数的曲面将呈现出如图7.18所示的两侧陡峭（曲率较大）、中间狭长平坦（曲率较小）的“峡谷”形状。这种极度不均匀的曲率分布会使基于梯度下降的优化过程变得困难，主要体现在以下两个方面：

- 在曲率最大的方向上，海森矩阵的特征值 $\lambda_{\max}$ 较大，此时二阶项主要由该特征值对应的特征向量方向主导，因此损失函数的二阶泰勒展开公式7.79可近似改写为（忽略高阶无穷小项）：

$$\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) - \mathcal{L}(\boldsymbol{w}) \approx -\eta\|\nabla\mathcal{L}(\boldsymbol{w})\|^2 + \frac{1}{2}\eta^2\lambda_{\max}\|\nabla\mathcal{L}(\boldsymbol{w})\|^2 \tag{7.81}$$

为了确保损失函数下降，即 $\mathcal{L}(\boldsymbol{w}+\Delta\boldsymbol{w}) - \mathcal{L}(\boldsymbol{w}) < 0$，必须满足：

$$0 < \eta < \frac{2}{\lambda_{\max}} \tag{7.82}$$

一旦学习率 η 超过该阈值，二阶项（随 η^2 增长）将超过一阶项（随 η 线性增长），从而导致损失函数不降反升，甚至发散。然而，在曲率较小的平坦方向上，较大的学习率则有助于加快收敛。上述约束（公式7.82）使学习率必须取较小值，从而导致在平坦方向上（即图7.18中损失函数等高线分布稀疏的方向）的收敛速度显著降低。

- 在狭长的谷底附近（尚未完全落入谷底而处于峡谷侧壁时），梯度往往主要由陡峭方向的分量主导。若采用稍大的学习率进行更新，由于谷底区域较为狭窄，参数往往会从一侧峡谷壁跨越谷底，直接冲到另一侧的峡谷壁上。而在另一侧，由于曲率同样较大，更新方向再次发生反转，使得参数返回原侧。如此反复，导致参数在峡谷两侧来回振荡，呈现出典型“之”字形（Zigzagging）轨迹。这种振荡使得大量更新被消耗在横向摆动上，而非沿谷底朝最优解方向推进，从而显著降低整体优化效率。在极端情况下，若曲率过于陡峭，多

次更新使得梯度范数或参数模长随迭代次数迅速增大，从而引发数值不稳定问题，甚至可能发生浮点溢出，即所谓的“数值爆炸”现象。

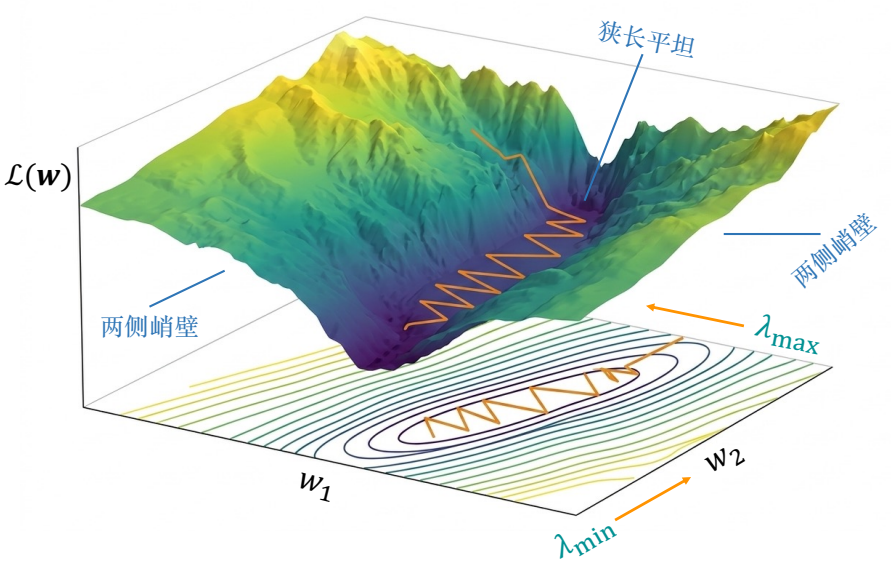


图 7.18: 病态条件对梯度下降影响的示意图。当海森矩阵的最大特征值 λ_{max} 与最小特征值 λ_{min} 存在显著差异时，损失函数曲面呈现“两侧陡峭、中间狭长且平坦”的峡谷形态。其中两侧峭壁对应曲率较大的方向，而中间狭长平坦区域对应曲率的较小方向。从损失曲面下方的等高线图中可以观察到：沿 λ_{max} 对应的特征向量方向的等高线分布密集，表明该方向上的微小参数变化会引起损失函数的剧烈波动；反之，沿 λ_{min} 对应的特征向量方向的等高线分布稀疏，说明该方向上的损失变化较为平缓。这种不平衡导致了优化的困境：为避免在陡峭方向上因步长过大而产生发散，迫使学习率取较小的值，这使得在平坦方向上进展缓慢。如图中橙色折线路径所示，梯度下降法在峡谷两壁间呈现出“之”字形来回振荡，而沿谷底朝最优解方向的收敛较为缓慢。

从上述分析可以看出，梯度下降法的优化难度主要取决于损失函数在不同方向上的曲率是否均衡。若某一方向的曲率显著大于其他方向，则损失函数曲面会呈现出如图7.18所示的“峡谷”结构。随着不同方向曲率差异的增大，峡谷中部将变得更加狭长，而两侧则更加陡峭。为了避免训练过程中可能出现的发散或数值不稳定问题，学习率必须受到最陡方向的约束（即公式7.82）而取较小值。这一限制会显著减缓平坦方向上的收敛速度，并使参数更新轨迹呈现出更加曲折的“之”字形路径。我们已知，海森矩阵的特征值刻画了损失函数在各特征向量方向上的曲率大小。从直观上看，海森矩阵最大特征值与最小特征值之比的绝对值衡量了损失曲面上最为极端的曲率不平衡程度，从而反映了该损失函数优化问题的难易程度。

对于上述海森矩阵这类正规矩阵[5]（Normal Matrix），其**条件数**（Condition Number）可以表示为绝对值最大的特征值 $|\lambda_{max}(\mathbf{H})|$ 与绝对值最小的特征值 $|\lambda_{mim}(\mathbf{H})|$ 之比：

$$\kappa(\mathbf{H}) = \frac{|\lambda_{max}(\mathbf{H})|}{|\lambda_{min}(\mathbf{H})|} \tag{7.83}$$

[5]在数学上，若复方阵 $\mathbf{A}$ 满足 $\mathbf{A}^*\mathbf{A} = \mathbf{A}\mathbf{A}^*$（其中 $\mathbf{A}^*$ 为 $\mathbf{A}$ 的共轭转置），则称矩阵 $\mathbf{A}$ 为正规矩阵。实对称矩阵是正规矩阵的特例。

更通用的定义是：对于任意非奇异矩阵 $\mathbf{A}$（即矩阵可逆），其条件数 $\kappa(\mathbf{A})$ 定义为矩阵范数与其逆矩阵范数的乘积：

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| \tag{7.84}$$

其中 $\|\mathbf{A}\|$ 表示矩阵 $\mathbf{A}$ 的范数，而 $\mathbf{A}^{-1}$ 是矩阵 $\mathbf{A}$ 的逆矩阵。若取常用的矩阵 ℓ_2 范数（谱范数），则矩阵 $\mathbf{A}$ 的条件数可以表示为最大奇异值与最小奇异值的比值：

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})} \tag{7.85}$$

其中 $\sigma_{\max}(\mathbf{A})$ 和 $\sigma_{\min}(\mathbf{A})$ 分别是矩阵 $\mathbf{A}$ 的最大与最小奇异值。由于奇异值均为非负实数，因此无需像公式7.83中基于特征值的定义那样引入绝对值。

从数值分析的角度来看，矩阵的条件数刻画了与该矩阵相关的计算问题对输入扰动的敏感程度，即其数值稳定性。一般而言，条件数越大，表示输出对输入中微小误差的放大效应越强，从而使问题的求解过程越不稳定。在第3.3节开篇的例子中，模型对拟合目标的微小扰动表现出较高敏感性，从而导致参数求解过程出现数值不稳定，其根本原因在于模型所涉及矩阵 $\mathbf{\Phi}$ 的条件数过大（$\kappa(\mathbf{\Phi}) = 2.0005/(4.999875 \times 10^{-5}) \approx 4 \times 10^4$）。

回到我们关于损失函数海森矩阵 $\mathbf{H}$ 的讨论，若该矩阵的条件数较大，则说明损失函数在不同方向上的曲率差异显著，优化问题也就越“病态”（Ill-conditioned）；反之，当条件数越接近 1 时，各方向曲率较为均衡，此时问题更接近“良态”（Well-conditioned），梯度下降通常也更容易稳定收敛。从几何上看，海森矩阵的条件数 $\kappa(\mathbf{H})$ 决定了损失函数等高线的形状：条件数越大，等高线越趋于扁平且拉长，最终形成典型的“峡谷”结构。将海森矩阵的特征分解 $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$ 代入损失函数泰勒展开的二阶项，可得：

$$\nabla\mathcal{L}(\boldsymbol{w})^\top \mathbf{H} \nabla\mathcal{L}(\boldsymbol{w}) = \Big(\mathbf{Q}^\top \nabla\mathcal{L}(\boldsymbol{w})\Big)^\top \mathbf{\Lambda} \Big(\mathbf{Q}^\top \nabla\mathcal{L}(\boldsymbol{w})\Big) = \boldsymbol{z}^\top \mathbf{\Lambda} \boldsymbol{z} = \sum_i \lambda_i z_i^2 \tag{7.86}$$

其中 $\boldsymbol{z} = \mathbf{Q}^\top \nabla\mathcal{L}(\boldsymbol{w})$ 是将梯度向量 $\nabla\mathcal{L}(\boldsymbol{w})$ 投影至由海森矩阵特征向量构成的正交基空间中，从而得到梯度在各特征方向上的分量 z_i。随后，由对角矩阵 $\mathbf{\Lambda}$ 根据对应的特征值 λ_i（即曲率）对投影分量的平方 z_i^2 进行加权。若将梯度 $\nabla\mathcal{L}(\boldsymbol{w})$ 看作输入，则该二阶项的本质是梯度在不同曲率方向上的分量被不同的特征值进行了非均匀的缩放。接下来，我们将延续本节开篇的示例，具体讨论海森矩阵 $\mathbf{H}$ 及其条件数的计算过程，并探讨它们与输入数据分布之间的内在联系。

海森矩阵 $\mathbf{H}$ 可以通过对损失函数（公式7.77）关于参数 $\boldsymbol{w}$ 求二阶导数来获得：

$$\mathbf{H} = \frac{1}{N}\sum_{i=1}^{N} \begin{bmatrix} \frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial w_1^2} & \frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial w_1 \partial w_2} \\ \frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 \mathcal{L}(\boldsymbol{w})}{\partial w_2^2} \end{bmatrix} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^\top \tag{7.87}$$

其中模型为如公式7.76所示的简单二元线性函数。接下来，我们可以证明：在对数据进行中心化处理后，该海森矩阵等价于输入数据 $\boldsymbol{x}$ 的协方差矩阵。在统计学中，样本协方差矩阵的定义为：

$$\mathbf{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_i - \mathbb{E}(\boldsymbol{x}))(\boldsymbol{x}_i - \mathbb{E}(\boldsymbol{x}))^\top \tag{7.88}$$

若数据经过中心化处理，即样本均值为零 ($\mathbb{E}(\boldsymbol{x}) = \mathbf{0}$)，则上式中的协方差矩阵可进一步化简为：

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \mathbf{0})(\boldsymbol{x}_i - \mathbf{0})^\top = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i\boldsymbol{x}_i^\top \tag{7.89}$$

对比公式7.87和公式7.89可知，在上述条件下，损失函数的海森矩阵等于输入数据的协方差矩阵。因此，海森矩阵的特征值也等于协方差矩阵的特征值。

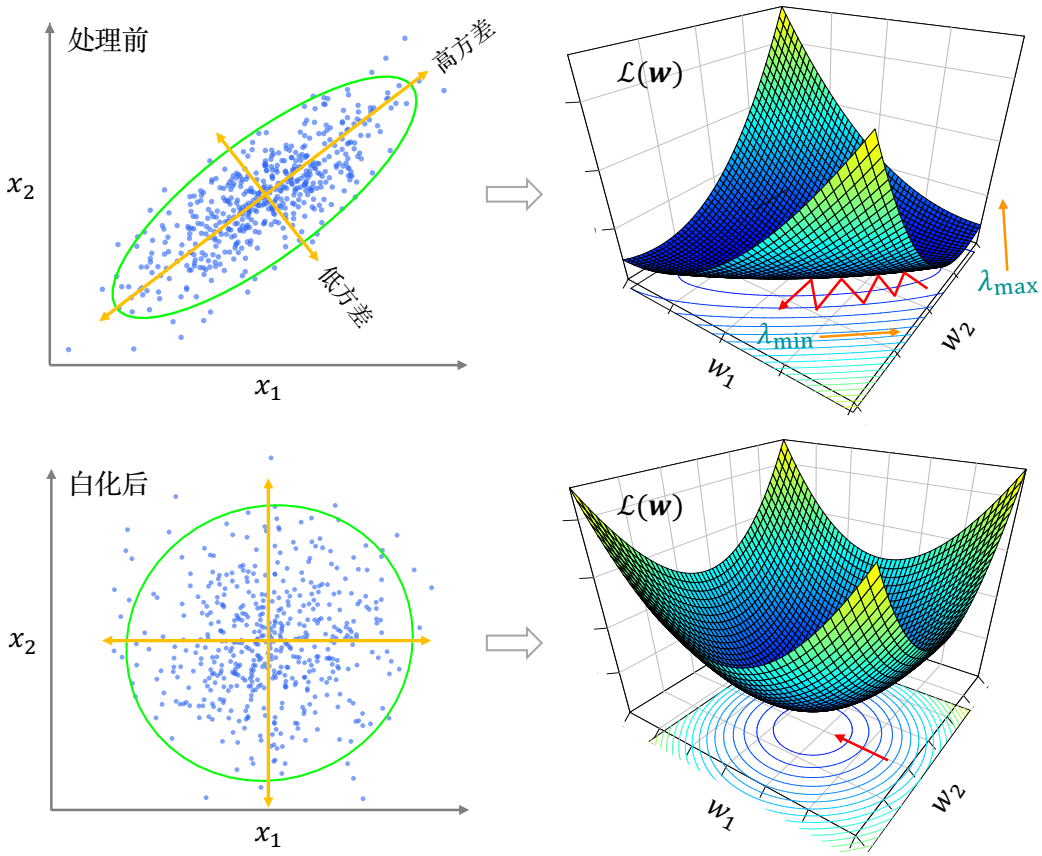


图 7.19: 数据白化对损失曲面形态的重塑作用。经过白化处理后，损失函数曲面由原来狭长的“峡谷”地貌被重塑为平滑均匀的“碗状”形态，这有助于提升优化过程的收敛速度与数值稳定性。在右下方子图中，红色箭头表示局部负梯度方向，该方向更直接地指向损失函数的极小值，因此可以采用相对较大的学习率以加快收敛。从几何角度来看，损失函数在参数空间中呈现的“盆地”形态及其拉伸程度，本质上是输入数据在其特征空间中“云朵”形状的镜像。

输入数据协方差矩阵的特征值刻画了数据在各主成分方向上的方差大小。从几何联系上看，输入数据协方差矩阵在各主成分方向上的特征值（即数据的方差）与损失函数在对应方向上的曲率密切相关，即数据方差越大，损失函数在该方向上的曲率也越大。为了使海森矩阵的条件数趋近于 1，从而将“病态”优化问题转为“良态”，可以对输入数据进行基于主成分分析（Principal Component Analysis）的**白化**（Whitening）处理。白化过程通过对数据进行去相关并统一各主成分方向上的方差，将原本尺度不均的特征空间转化为各向同性的空间。如图7.19所示，输入数据经过白化处理后，损失函数曲面从原来的狭长的“峡谷”地貌重塑为平滑均匀的“碗状”形态。

这种数据预处理方法能够显著提升优化算法的收敛速度与数值稳定性。

基于主成分分析的白化，本质上是先去相关，再归一化方差。对于已中心化的数据 $\boldsymbol{x}$（即已减去均值），白化通过如下线性变换得到：

$$\boldsymbol{z} = \mathbf{W}\boldsymbol{x} \tag{7.90}$$

其目标是使变换后数据的协方差矩阵为单位矩阵：

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{z}_i - \underbrace{\mathbb{E}[\boldsymbol{z}]}_{=0})(\boldsymbol{z}_i - \underbrace{\mathbb{E}[\boldsymbol{z}]}_{=0})^{\top} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{z}_i\boldsymbol{z}_i^{\top} = \mathbf{I} \tag{7.91}$$

由于单位矩阵 $\mathbf{I}$ 的对角线元素均为 1、非对角线元素均为 0，因此经过白化变换后的数据向量 $\boldsymbol{z}$ 在各个维度上相互不相关，且每一维的方差均被归一化为 1。从几何角度来看，这一过程消除了原始数据在不同方向上的方差差异，使其协方差结构由各向异性变为各向同性，即数据在各个方向上具有相同的方差。

白化通常包括以下几个步骤。首先，对原始数据 $\boldsymbol{x}$ 进行中心化处理：

$$\dot{\boldsymbol{x}} = \boldsymbol{x} - \mathbb{E}[\boldsymbol{x}] \tag{7.92}$$

接着，计算中心化数据的协方差矩阵：

$$\boldsymbol{\Sigma} = \mathbb{E}[\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}^{\top}] = \frac{1}{N}\sum_{i=1}^{N}\dot{\boldsymbol{x}}_i\dot{\boldsymbol{x}}_i^{\top} \tag{7.93}$$

随后，对协方差矩阵进行特征分解：

$$\boldsymbol{\Sigma} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\top} \tag{7.94}$$

在此基础上，根据特征分解结果构造白化变换矩阵：

$$\mathbf{W} = \boldsymbol{\Lambda}^{-1/2}\mathbf{Q}^{\top} \tag{7.95}$$

最后，对中心化后的数据进行线性变换，得到白化后的表示：

$$\boldsymbol{z} = \mathbf{W}\dot{\boldsymbol{x}} = \boldsymbol{\Lambda}^{-1/2}\mathbf{Q}^{\top}\dot{\boldsymbol{x}} \tag{7.96}$$

上述白化变换可以分解为以下两个步骤。首先，通过正交变换（旋转）：

$$\dot{\boldsymbol{x}}' = \mathbf{Q}^{\top}\dot{\boldsymbol{x}} \tag{7.97}$$

将中心化数据 $\dot{\boldsymbol{x}}$ 投影到以协方差矩阵特征向量为基的空间中。虽然此时各维度的相关性已消除，但各维度的方差仍不统一（第 i 维的方差为 λ_i）。然后，通过对各维度分量除以其标准差，实现方差的归一化，使所有维度的方差均变为 1：

$$z_i = \frac{\dot{x}_i'}{\sqrt{\lambda_i}} \tag{7.98}$$

其中 λ_i 为对角矩阵 $\boldsymbol{\Lambda}$ 中的第 i 个对角元素。

我们可以证明，经过上述白化变换后的数据 $\boldsymbol{z}$ 的协方差矩阵为单位矩阵：

$$
\begin{aligned}
\mathrm{Cov}[\boldsymbol{z},\boldsymbol{z}] &= \mathbb{E}\left[(\boldsymbol{z}-\mathbb{E}[\boldsymbol{z}])(\boldsymbol{z}-\mathbb{E}[\boldsymbol{z}])^\top\right] \\
&= \mathbb{E}[\boldsymbol{z}\boldsymbol{z}^\top] \quad （线性变换\ \boldsymbol{z}=\mathbf{W}\dot{\boldsymbol{x}}\ 不会改变零均值性，即\ \mathbb{E}[\boldsymbol{z}]=0）\\
&= \mathbb{E}\left[(\mathbf{W}\dot{\boldsymbol{x}})(\mathbf{W}\dot{\boldsymbol{x}})^\top\right] \\
&= \mathbf{W}\mathbb{E}[\dot{\boldsymbol{x}}\dot{\boldsymbol{x}}^\top]\mathbf{W}^\top \quad （期望的线性性质）\\
&= \mathbf{W}\boldsymbol{\Sigma}\mathbf{W}^\top \\
&= \boldsymbol{\Lambda}^{-1/2}\mathbf{Q}^\top\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top\left(\boldsymbol{\Lambda}^{-1/2}\mathbf{Q}^\top\right)^\top \quad （代入\ \mathbf{W}=\boldsymbol{\Lambda}^{-1/2}\mathbf{Q}^\top\ 和\ \boldsymbol{\Sigma}=\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top）\\
&= \boldsymbol{\Lambda}^{-1/2}\boldsymbol{\Lambda}\boldsymbol{\Lambda}^{-1/2} \quad （利用正交性\ \mathbf{Q}^\top\mathbf{Q}=\mathbf{I}）\\
&= \mathbf{I} \quad （\boldsymbol{\Lambda}\ 为对角阵）
\end{aligned}
$$

以上推导中，我们假设协方差矩阵的特征值严格为正。

虽然数据白化能显著改善损失函数曲面的形态，但由于其涉及协方差矩阵的计算与特征值分解，在输入特征维度较高（如图像）的情况下，计算代价过于高昂。因此，实践中通常采用**标准化**（Standardization），亦称 Z 值归一化（Z-score Normalization）来替代。标准化通过平移和缩放将输入数据的每一维特征独立地调整为零均值和单位方差的分布。为了简化表达，以下仅对第 j 维特征标准化进行描述：

$$x'_{ij} = \frac{x_{ij}-\mu_j}{\sigma_j} \tag{7.99}$$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij} \tag{7.100}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij}-\mu_j)^2 \tag{7.101}$$

其中 x'_{ij} 表示第 i 个样本在第 j 维特征标准化后的结果，N 为样本总数，μ_j 和 σ_j 分别为该特征在训练数据上的均值与标准差。需要强调的是，不同于白化操作会消除特征间的相关性（即令协方差矩阵非对角元素都为 0），标准化是对每一输入特征维度分别独立进行。标准化实际上是忽略了特征间的相关性，仅通过平移和缩放操作使协方差矩阵对角线元素进行了归一化。由于标准化未能消除特征间的相关性，模型参数之间仍然可能存在耦合关系。这意味着损失函数的等高线虽在尺度上得到了修正，但其主轴方向并未与坐标轴对齐。因此，局部负梯度方向往往不会像白化后那样直指极小值中心，而是会受特征相关性的影响产生偏转，导致优化轨迹呈现一定程度的“之”字形振荡。这也是标准化不如白化“理想”的一个重要原因。对同一原始数据进行标准化与白化后的对比如图7.20所示。

数据预处理（如标准化与白化）不仅能够调整输入特征的数值尺度，在一定条件下还可以改善损失函数海森矩阵的条件数。从优化的角度来看，这本质上相当于在参数学习之前对损失函数海森矩阵的条件数进行“预设”（Preconditioning），或称预条件化。也就是通过调整输入数

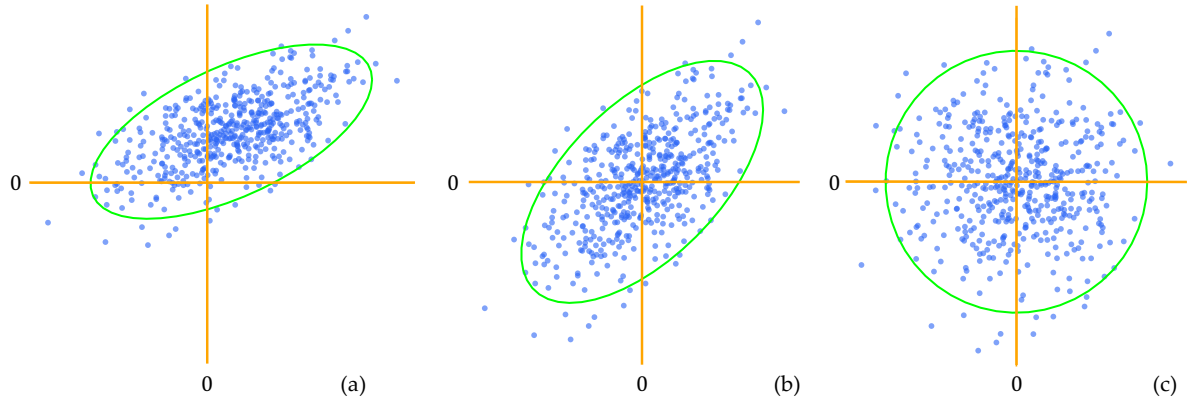


图 7.20: 数据标准化与白化后的对比。(a) 原始数据分布。(b) 标准化后的数据分布：各特征维度被平移至零均值并缩放至单位方差，但是特征之间的相关性仍然存在。(c) 白化后的数据分布：在标准化的基础上进一步消除特征之间的线性相关性，使数据在各个方向上具有相同的方差，呈现各向同性的协方差结构。

据的分布，尽可能改善海森矩阵的条件数，使参数空间中的损失函数曲面从狭长的“峡谷”重塑为更加规整的“圆碗”，从而消除不同特征尺度对梯度下降的干扰。事实上，许多参数优化算法也试图通过调整参数的更新方向或步长，缓解优化过程中不同方向上的更新不均衡问题。例如，动量法通过累积历史梯度，减轻梯度在陡峭方向上的反复振荡，并加快在梯度方向相对一致的方向上的推进。自适应学习率算法（如 AdaGrad 和 RMSprop）则根据各参数历史梯度的大小，动态调整相应的学习率，使梯度较大的参数通常采用较小的更新步长，而梯度较小的参数可以采用相对较大的步长。Muon 则通过对动量累积得到的梯度更新矩阵进行近似正交化处理，使不同奇异方向上的更新幅度更加均衡。这些方法虽然采用了不同的机制，但都旨在改善参数更新的效率与稳定性，使模型即使面临不同方向上尺度差异较大、优化条件较差的情况，也能够更加有效地进行训练。

对于回归任务，通常也会对目标输出进行标准化处理（并在推理阶段再进行反标准化，即乘以标准差后加上均值），其主要作用是统一损失函数的数值尺度以及相应梯度的量级。特别是在多维输出的情况下，如果不同维度的量纲差异较大，那么未标准化前，大尺度的输出维度会在损失函数中占据主导地位。这在优化上等价于使某些方向的曲率显著增大，从而导致或加剧问题的病态性。需要注意的是，在进行数据中心化和标准化之前，通常应先对异常值（Outliers）进行适当处理（如剔除或截断）。这是因为异常值会显著影响均值和方差的估计，从而扭曲预处理后的数据分布，进而削弱标准化等预处理方法的效果。

7.6.3 归一化方法

上一节我们讨论了当输入数据协方差矩阵的特征值差异较大时，可能导致损失函数在不同参数方向上的曲率差异增大，使海森矩阵的条件数变大，从而引发病态优化问题。类似的现象也可能出现在深度神经网络的各个层中。由于前一层的输出构成了后一层的输入，因此，从直

观上看，如果前一层输出特征的协方差矩阵具有差异较大的特征值，就意味着这些特征在不同方向上的变化程度存在显著差异。这可能进一步影响后一层权重参数的优化条件，使其损失函数在不同方向上呈现不同的曲率，从而增加参数优化的难度。然而，由于前层的输出分布会随着其参数的更新而不断发生偏移，即所谓的“内部协变量偏移”（Internal Covariate Shift），我们无法像处理原始输入数据那样，通过一次性的预处理来完成数据标准化或白化。因此，针对神经网络内部的“数据预处理”必须动态进行。

在具体实现上，我们首先排除白化处理。这是因为白化需要对协方差矩阵进行估计与分解，计算复杂度过高，不适合在大规模训练过程中频繁使用。相较之下，通常采用计算代价较小的标准化方法，并且以基于小批量数据的方式进行，即**批量归一化**（Batch Normalization）[39]。该方法的核心想法是利用当前训练的小批量样本来近似估计标准化所需的均值和方差。具体而言，在深度神经网络某一层计算得到一小批样本的输出后，按如下方式计算该批次样本在第 j 维上的均值 μ_j 与方差 σ_j^2：

$$\mu_j = \frac{1}{B}\sum_{i=1}^{B} x_{ij} \tag{7.102}$$

$$\sigma_j^2 = \frac{1}{B}\sum_{i=1}^{B}(x_{ij} - \mu_j)^2 \tag{7.103}$$

其中 B 表示小批量的大小，x_{ij} 为当前批次中第 i 个样本在第 j 维特征上的激活值。得到批次统计量后，我们通过下式对该特征进行标准化处理：

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \tag{7.104}$$

其中 ϵ 为一个极小的正数（如 10^{-5}），用于防止分母为零并提高数值计算的稳定性。经过上述变换后，该层输出在当前小批量内的每一维特征均被标准化为近似零均值、单位方差的分布。

从直觉上看，批量归一化应该作用于两层神经网络之间，即在每一层完成如下线性变换与非线性变换之后引入：

$$\boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \quad \text{（线性变换）} \tag{7.105}$$

$$\boldsymbol{h} = g(\boldsymbol{z}) \quad \text{（非线性变换）} \tag{7.106}$$

其中 $\boldsymbol{x}$ 表示当前层的输入，$\mathbf{W}$ 为该层的权重矩阵，$\boldsymbol{b}$ 为偏置向量，$\boldsymbol{z}$ 为该层的净输入，而 $g(\cdot)$ 表示所采用的非线性激活函数，最终得到的 $\boldsymbol{h}$ 为该层的输出（或称激活值）。实际上，文献 [39] 建议将批量归一化置于线性变换之后、非线性变换之前，即作用于净输入 $\boldsymbol{z}$ 而非输出值 $\boldsymbol{h}$。其主要原因在于，线性变换后的数据分布通常比经过非线性激活后的分布更适合进行归一化处理。在线性变换 $\boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ 之后，各维特征都是多个输入分量的加权和。根据中心极限定理，在一定条件下，多个随机变量的加权和可能近似服从高斯分布。因此，线性变换后的数据分布往往具有相对对称、连续的特点，而对其进行标准化，可以有效调整非线性激活函数输入的均值和方差，使其保持在较为合适的数值范围内。相比之下，若将批量归一化置于非线性变换之后，

输出分布可能因非线性函数的作用而发生明显变化。例如，ReLU 激活函数会将所有负值映射为零，使输出分布在零点处聚集大量取值，并呈现明显的不对称性。如果此时再进行标准化，虽然仍然可以调整输出的均值和方差，但无法消除这种由非线性变换产生的分布特征，而且原本经 ReLU 变换后得到的零值也可能重新变为非零值。因此，将批量归一化置于非线性激活函数之前，可以更直接地调整激活函数的输入分布，同时保留非线性激活函数本身产生的输出特征。

然而，将批量归一化置于线性变换与非线性变换之间也会引入新的问题。由于上述归一化操作会将净输入各维度调整为零均值、单位方差的分布（在近似高斯分布的假设下，大多数取值集中在 $[-3,3]$ 区间内，即约 99.73% 的概率落在正负三个标准差范围内），这会显著影响压缩型非线性函数的作用。以 Sigmoid 激活函数为例，这一区间恰好对应其近似线性的中心区域。如果网络各层的输入均被压缩到这一范围内，则激活函数的非线性作用会被削弱，使其难以充分利用不同输入区间的非线性特性，从而影响模型的表达能力。为了解决这一问题，批量归一化在标准化之后，为每一维特征引入两个可学习参数：缩放参数 α_j 与偏移参数 β_j，并对标准化后的净输入进行如下仿射变换：

$$z'_j = \alpha_j \hat{z}_j + \beta_j \tag{7.107}$$

其中 z'_j 是第 j 维仿射变换后的值。这一仿射变换仍然赋予了模型“恒等映射”的能力。当 $\beta_j = \mu_j$ 且 $\alpha_j = \sqrt{\sigma_j^2 + \epsilon}$ 时，有 $z'_j = z_j$，即可以还原未经归一化的净输入。因此，批量归一化在提升优化效率的同时，并不会在表达能力上对模型施加额外限制。

综上所述，批量归一化主要包括以下三个步骤：首先，计算每一维特征在当前小批量上的均值与方差；其次，利用这些统计量对输入进行标准化；最后，对标准化后的结果进行缩放与平移变换。需要强调的是，缩放参数与偏移参数是可学习的。在模型训练过程中，批量归一化可以被视作一个独立的网络层，其缩放参数 α 与偏移参数 β 可以与其他网络参数一起，通过梯度下降方法进行联合优化。这使得该批量归一化层不仅能够对各层的输入分布进行规范化，还能够根据输入样本及其他参数的变化，自适应地调整输出的尺度与偏移，从而提升模型特征表达的灵活性。此外，由于不同小批量所估计的均值与方差因存在随机波动而略有差异，这种由批量统计带来的噪声在一定程度上相当于一种正则化，有助于缓解模型的过拟合。

标准批量归一化训练时使用当前批次统计量，而推理时通常使用训练过程中估计并保存的总体统计量（如均值和方差）。对卷积特征，通常按通道，在批次与空间位置维度上共同计算均值和方差，并为每个通道学习一组缩放、偏移参数。

上述批量归一化是在样本维度上进行的，更准确地说，是在小批量内对每一维特征分别进行归一化处理。这通常要求输入样本的特征维度是一致的，即不同样本同一维度的特征能够对齐。因此，批量归一化比较适合图像等可以通过缩放或裁剪等手段统一输入尺寸的情况。然而，对于序列数据，样本长度往往是不一致的。即使在组装小批量时选取长度相近的样本，批次内仍不可避免地存在长度差异。此时，若采用零值填充等来强行对齐特征维度以计算均值与方差，填充值将显著干扰这些统计量的估计，导致均值与方差在不同批次之间剧烈波动，从而损害训练的稳定性。为使归一化机制也能够有效应用于序列建模等场景，研究者提出了**层内归一化**或

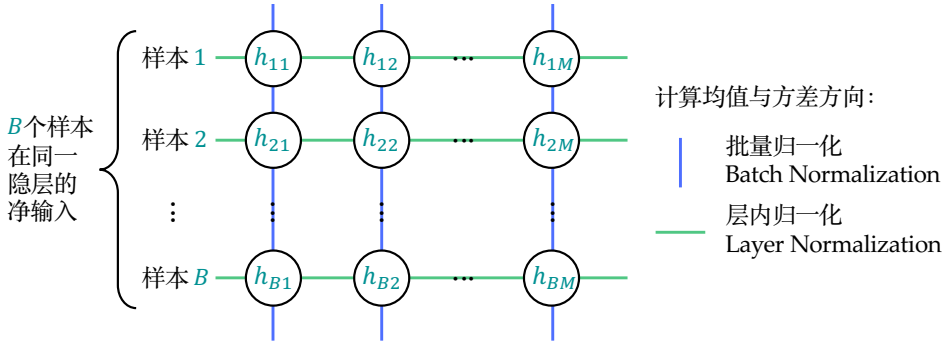


图 7.21: 批量归一化与层内归一化在统计量计算方向上的对比示意图。图中以同一隐层（包含 M 个神经元）中的 B 个样本为例，其中 h_{ij} 表示第 i 个样本在该层第 j 个神经元上的净输入。批量归一化沿蓝色直线所示方向，在样本维度上计算归一化所需的均值与方差；而层内归一化则沿绿色线条所示方向，在单个样本内部的特征维度上计算相应的统计量。

层归一化（Layer Normalization）[3]。与批量归一化不同，层内归一化不再跨多个样本来计算统计量，而是在单个样本内部，对同一层的所有神经元进行归一化。在处理序列任务的循环神经网络中，这意味着在每一个时间步，层内归一化都会独立地对该时刻同一隐藏层的神经元进行归一化，从而避免序列长度差异对归一化过程的影响。具体而言，层内归一化针对同一层内所有神经元的净输入计算均值与方差：

$$\mu_l = \frac{1}{M}\sum_{i=1}^{M} a_{i,l} \tag{7.108}$$

$$\sigma_l^2 = \frac{1}{M}\sum_{i=1}^{M}(a_{i,l} - \mu_l)^2 \tag{7.109}$$

其中 $a_{i,l}$ 表示第 l 层第 i 个神经元的净输入，M 为该层神经元的总数，μ_l 和 σ_l^2 分别为在该层内计算得到的均值与方差。与批量归一化类似，层内归一化同样引入了可学习的缩放参数 $\alpha_{i,l}$ 与偏移参数 $\beta_{i,l}$，用于对标准化后的净输入进行仿射变换：

$$\hat{a}_{i,l} = \frac{a_{i,l} - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \quad \text{（标准化）} \tag{7.110}$$

$$a'_{i,l} = \alpha_{i,l}\hat{a}_{i,l} + \beta_{i,l} \quad \text{（仿射变换）} \tag{7.111}$$

其中 ϵ 为一个极小的正数。这一仿射变换旨在恢复模型在标准化过程中可能损失的表征能力，并确保随后的非线性激活函数能够有效发挥作用。批量归一化与层内归一化所需统计量（即均值与方差）在计算方向上的对比示意如图7.21所示。

7.6.4 参数初始化

参数初始化是指在神经网络训练开始之前，为权重和偏置赋予合适的初始值，从而为后续优化提供良好的起点，以提升训练的稳定性并加速收敛。通常最佳的参数初始化方法是预训练，

但预训练本身也依赖于合理的初始化。在许多实际应用中，往往也需要从头开始训练网络模型，因此参数初始化仍然是一个基础且关键的问题。参数初始化通常需要着重考虑以下两个方面：

- **打破参数的对称性**。若将同一层内的所有权重初始化为相同的值，则在反向传播过程中，这些参数将接收到完全相同的梯度信号。这意味着无论训练如何进行，这些参数始终执行相同的更新，从而使对应神经元学习到相同的特征表示，模型退化为仅能提取单一模式，从而失去了表达和组合复杂模式的能力。打破对称性最简单的方法是随机初始化，这赋予每个神经元独特的“起点”，使其能够学习到多样化的模式。
- **信息传递的稳定性**。在深层神经网络中，信号（前向传播的激活值与反向传播的梯度）需要通过多次矩阵运算，经过数十甚至上百层的传播。初始化应确保信号在传播过程中既能保持足够的强度以避免梯度消失，又不至于过强而引发数值溢出或梯度爆炸。因此，参数初始化需要使得信号在多层前向与反向传播过程中，其方差保持在相对稳定的范围内（即满足所谓的方差齐性条件），以保证训练过程的有效性与数值稳定性。

为了同时实现上述两个目标，最常用的方法是预先设定某种合适的概率分布，并从中进行随机采样以完成参数的初始化。接下来，我们先来讨论权重参数应服从何种分布才能满足方差齐性条件。所谓**方差齐性**，是指在神经网络逐层的信息传递过程中，相邻两层神经元激活值的方差保持一致，即输入与输出的激活值方差尽可能相等。若这一条件不满足，可以假设每一层输出激活值方差与输入激活值方差之比为 r，即每经过一层后方差被放大或缩小的比例。则经过 L 层传播之后，激活值的方差将变为原来的 r^L 倍。当 $r>1$ 时，该值随层数呈指数增长；当 $r<1$ 时，则呈指数衰减，由此会导致信号在深层网络中逐层放大或衰减，从而引发数值爆炸或梯度消失等问题。为简化分析，我们暂不考虑非线性激活函数的影响（即假设激活函数为恒等映射），并忽略偏置项。此时，某一层神经网络的输入 $\boldsymbol{x}$ 与输出 $\boldsymbol{z}$ 之间的线性关系可表示为：

$$z_j = \sum_{i=1}^{M_{\text{in}}} w_{ji} x_i \tag{7.112}$$

其中 x_i 表示该层第 i 个输入神经元的激活值，z_j 表示第 j 个输出神经元的激活值，w_{ji} 是连接输入神经元 i 与输出神经元 j 的权重，而 M_{in} 表示输入神经元的总数（即该层的扇入数）。下面我们推导该层第 j 个输出神经元激活 z_j 的方差：

$$\begin{aligned}
\text{Var}[z_j] &= \text{Var}\left[\sum_{i=1}^{M_{\text{in}}} w_{ji} x_i\right] \\
&= \sum_{i=1}^{M_{\text{in}}} \text{Var}[w_{ji}] \cdot \text{Var}[x_i] \quad \text{（假设 } w_{ji} \text{ 与 } x_i \text{ 相互独立，且 } \mathbb{E}[w_{ji}]=0 \text{ 及 } \mathbb{E}[x_i]=0\text{）} \\
&= M_{\text{in}} \cdot \text{Var}[w_{ji}] \cdot \text{Var}[x_i] \quad \text{（同层权重独立同分布）}
\end{aligned}$$

根据上述推导结果，若要求输出激活值的方差 $\text{Var}[z_j]$ 与输入激活值的方差 $\text{Var}[x_i]$ 保持一致（即方差比率为 1），则权重方差必须满足：

$$\frac{\text{Var}[z_j]}{\text{Var}[x_i]} = M_{\text{in}} \cdot \text{Var}[w_{ji}] = 1 \quad \Rightarrow \quad \text{Var}[w_{ji}] = \frac{1}{M_{\text{in}}} \tag{7.113}$$

在反向传播过程中，损失函数 $\mathcal{L}$ 关于输入神经元的梯度 $\frac{\partial \mathcal{L}}{\partial x_i}$ 取决于其关于输出神经元的梯度 $\frac{\partial \mathcal{L}}{\partial z_j}$。根据链式法则，其关系如下：

$$\frac{\partial \mathcal{L}}{\partial x_i} = \sum_{j=1}^{M_{\text{out}}} w_{ji} \cdot \frac{\partial \mathcal{L}}{\partial z_j} \tag{7.114}$$

其中 M_{out} 表示当前层的输出神经元总数（即该层的扇出数）。我们计算梯度的方差：

$$\begin{aligned}
\text{Var}\left[\frac{\partial \mathcal{L}}{\partial x_i}\right] &= \text{Var}\left[\sum_{j=1}^{M_{\text{out}}} w_{ji} \cdot \frac{\partial \mathcal{L}}{\partial z_j}\right] \\
&= \sum_{j=1}^{M_{\text{out}}} \text{Var}\left[w_{ji} \cdot \frac{\partial \mathcal{L}}{\partial z_j}\right] \quad \text{（假设权重 } w_{ji} \text{ 与梯度相互独立，且梯度的均值为 0）} \\
&= M_{\text{out}} \cdot \text{Var}[w_{ji}] \cdot \text{Var}\left[\frac{\partial \mathcal{L}}{\partial z_j}\right]
\end{aligned}$$

同理，若要求梯度方差在反向传播过程中保持尺度不变，则需要满足：

$$\frac{\text{Var}\left[\frac{\partial \mathcal{L}}{\partial x_i}\right]}{\text{Var}\left[\frac{\partial \mathcal{L}}{\partial z_j}\right]} = M_{\text{out}} \cdot \text{Var}[w_{ji}] = 1 \quad \Rightarrow \quad \text{Var}[w_{ji}] = \frac{1}{M_{\text{out}}} \tag{7.115}$$

在实际网络结构中，每一层的输入神经元数量 M_{in} 与输出神经元数量 M_{out} 通常并不相等。为了同时兼顾前向和反向传播的数值稳定性，文献 [29] 建议取 M_{in} 和 M_{out} 算术平均值的倒数作为权重的方差：

$$\text{Var}[w_{ji}] = \frac{2}{M_{\text{in}} + M_{\text{out}}} \tag{7.116}$$

由于均匀分布易于采样，并且能使权重的初始值在给定区间内分布得均匀且分散（直观上降低多个权重取相同或相近数值的可能性），因此常被用作初始化的基础分布。假设权重的初始值服从均匀分布 $\text{U}(x|-a,a)$，并要求该分布的方差满足公式7.116所给出的条件。根据均匀分布的方差公式 $\text{Var}[x] = \frac{a^2}{3}$，我们可以得到边界参数 a 的取值为：

$$\frac{a^2}{3} = \frac{2}{M_{\text{in}} + M_{\text{out}}} \quad \Rightarrow \quad a = \sqrt{\frac{6}{M_{\text{in}} + M_{\text{out}}}} \tag{7.117}$$

综上所述，将权重 w_{ji} 初始化为如下均匀分布：

$$\text{U}\left(-\sqrt{\frac{6}{M_{\text{in}} + M_{\text{out}}}}, \sqrt{\frac{6}{M_{\text{in}} + M_{\text{out}}}}\right) \tag{7.118}$$

即为广泛应用的 **Xavier 初始化**（亦称 Glorot 初始化）。

我们接下来考虑非线性激活函数对梯度传播的影响。不同的激活函数在反向传播时，会对梯度产生不同程度的缩放。若采用 Tanh 激活函数，其导数取值范围为 $(0,1]$（详见公式7.37）；而若采用 Sigmoid 激活函数，其导数取值范围为 $(0,0.25]$（见公式7.38）。为了抵消激活函数导数小于 1 可能导致的梯度逐层衰减，通常需要在初始化时对权重的方差引入一个缩放因子 τ^2 进行补偿，而这等价于将初始化均匀分布的边界参数 a 乘上相应的补偿因子 τ。由于 Sigmoid 函数最大导数仅为 Tanh 函数最大导数的 $\frac{1}{4}$，为了在反向传播中保持相同的梯度尺度，Sigmoid 对应

的补偿因子 τ 通常设定为 Tanh 的 4 倍。

如果采用 ReLU 激活函数，它会将小于零的输入截断为零。在激活前的净输入近似服从以零为中心的对称分布的假设下，约有一半的神经元输出为 0。这一截断操作会使激活值的方差（ReLU 输出的均值通常大于 0，因此严格说应是二阶矩）相较于以恒等映射作为激活函数时的情形缩小了约一半。基于上述分析，文献 [34] 建议将权重的方差设定为：

$$\mathrm{Var}[w_{ji}] = \frac{2}{M_{\text{in}}} \tag{7.119}$$

将上式与公式7.116对比可以看出，其分子仍然为 2，但分母仅依赖于该层的输入神经元数量 M_{in}，这反映了由于 ReLU 的截断效应，使有效参与传播的神经元数量减少，从而需要相应增大权重方差以维持信号强度。通常某一层的输入神经元数量 M_{in} 与输出神经元数量 M_{out} 并不相等。上述公式侧重于维持前向传播的信号强度（即基于该层的扇入数）；类似地，若侧重于保证梯度反向传播的稳定性，也可将权重方差设为 $\frac{2}{M_{\text{out}}}$（即基于该层的扇出数）。由于 M_{in} 和 M_{out} 在量级上通常较为接近，一般默认采用 $\frac{2}{M_{\text{in}}}$ 作为权重初始化的方差。若权重采用均匀分布进行初始化，则相应的分布形式调整为：

$$\mathrm{U}\left(-\sqrt{\frac{6}{M_{\text{in}}}}, \sqrt{\frac{6}{M_{\text{in}}}}\right) \tag{7.120}$$

这种针对 ReLU 激活函数而改进的权重初始化方法被称为 **He 初始化**（亦称 Kaiming 初始化）。

上述 Xavier 和 He 初始化均是对权重矩阵中的各个元素进行独立随机采样。尽管这类方法在统计意义上能够维持信号方差的稳定性，但并不能保证输入向量经过权重矩阵线性变换后，其范数保持不变。从理论上看，保持信号范数不变是缓解梯度爆炸与消失问题的重要条件之一。为此，文献 [67] 提出将权重矩阵初始化为正交矩阵，即满足 $\mathbf{W}\mathbf{W}^\top = \mathbf{I}$（或 $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$）。我们将在后续分析中可以证明，采用正交矩阵对权重进行初始化，在理想条件下能够在前向传播与反向传播过程中保持激活值与梯度的范数不变。需要注意的是，严格意义上的正交矩阵必须是方阵，因此只有当输入与输出神经元数量相等时，权重矩阵才能满足严格的正交性要求。对于非方阵的权重矩阵，也可以采用半正交初始化，使其行向量或列向量保持正交归一。**正交初始化**在循环神经网络中尤为有效。这是因为循环连接或结构要求各时间步的隐藏状态维度一致，从而天然满足方阵条件。由于正交矩阵的所有特征值模长均为 1，其谱范数也为 1。在跨时间步传播梯度的过程中，该性质能够避免梯度发生指数级放大或衰减，从而有效改善长序列训练中的梯度稳定性问题。不过，实际的梯度传播还受到激活函数等因素的影响，因此正交初始化并不能完全消除梯度消失或爆炸问题。

我们首先证明，正交权重矩阵 $\mathbf{W}$ 在前向传播过程中能够保持输入向量 $\boldsymbol{x}$ 的 ℓ_2 范数不变：

$$\|\mathbf{W}\boldsymbol{x}\|^2 = (\mathbf{W}\boldsymbol{x})^\top(\mathbf{W}\boldsymbol{x}) = \boldsymbol{x}^\top\mathbf{W}^\top\mathbf{W}\boldsymbol{x} = \boldsymbol{x}^\top\mathbf{I}\boldsymbol{x} = \boldsymbol{x}^\top\boldsymbol{x} = \|\boldsymbol{x}\|^2$$

由此可见，经过正交矩阵线性变换后，向量的 ℓ_2 范数与输入向量 $\boldsymbol{x}$ 相同。对于反向传播过程，设该层的输出为 $\boldsymbol{z} = \mathbf{W}\boldsymbol{x}$，我们同样可以证明梯度的 ℓ_2 范数保持不变：

$$\left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}\right\|^2 = \left\|\mathbf{W}^\top \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\right\|^2 = \left(\mathbf{W}^\top \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\right)^\top \left(\mathbf{W}^\top \cdot \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\right) = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\right)^\top \mathbf{W}\mathbf{W}^\top \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\right) = \left\|\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}}\right\|^2$$

直观上，正交矩阵的作用等同于在空间中进行“旋转”或“旋转加镜像（反射）”。它仅改变向量的方向，而不产生拉伸或压缩效应，因此在前向和反向传播过程中均不会改变向量的范数。在神经网络中，两次线性变换之间通常会引入非线性激活函数。非线性变换会在一定程度上改变向量的范数。当权重矩阵采用正交初始化时，通常需要为正交矩阵乘以一个缩放系数 τ，以补偿非线性变换引起的范数衰减。以 ReLU 激活函数为例，其在 0 附近的平均梯度可以近似为 0.5。为了保持范数不变，缩放系数 τ 需满足以下条件：

$$0.5\tau^2 = 1 \quad \Rightarrow \quad \tau = \sqrt{2}$$

正交初始化的具体实现通常可以分为三个步骤：首先，从均值为 0、方差为 1 的标准高斯分布中随机采样，生成一个初始矩阵；然后，对该矩阵进行奇异值分解，得到两个正交矩阵，并选取其中一个正交矩阵；最后，将选取的正交矩阵乘以适当的缩放系数 τ，以完成最终的权重矩阵的参数初始化。需要注意的是，上述结论仅适用于初始化时权重矩阵所对应的线性变换。在后续训练过程中，参数更新一般并不能自动保持权重矩阵的正交性。此外，即使权重矩阵保持正交，非线性激活函数的雅可比矩阵仍可能导致梯度衰减。权重矩阵的常见初始化方法总结如表7.2所示。

相比权重矩阵，偏置项的初始化通常较为简单，因为偏置不直接参与矩阵乘法的累积效应，并对信号尺度的影响相对较小。实践中，常将偏置初始化为 0。在使用 ReLU 作为激活函数时，有时也会将偏置初始化为一个较小的正数（如 0.01），以增加训练初期神经元“活跃”（被激活）的概率，从而降低出现“死亡神经元”（Dead Neurons）的风险。此外，在长短期记忆网络（LSTM）的遗忘门中，通常将偏置初始化为 1，以在训练初期倾向于“记忆”而非“遗忘”。

表 7.2: 代表性权重初始化方法总结

初始化方法	分布	方差	核心思想
Xavier（Glorot）初始化	$\mathrm{U}\left(-\tau\sqrt{\frac{6}{M_{\text{in}}+M_{\text{out}}}}, \tau\sqrt{\frac{6}{M_{\text{in}}+M_{\text{out}}}}\right)$	$\frac{2\tau^2}{M_{\text{in}}+M_{\text{out}}}$	维持前后向传播的方差齐性
He（Kaiming）初始化	$\mathrm{U}\left(-\sqrt{\frac{6}{M_{\text{in}}}}, \sqrt{\frac{6}{M_{\text{in}}}}\right)$	$\frac{2}{M_{\text{in}}}$	补偿 ReLU 引起二阶矩减半
正交初始化	$\mathbf{W}\mathbf{W}^\top = \mathbf{I}$ 或 $\mathbf{W}^\top\mathbf{W} = \mathbf{I}$		保持线性变换前后向量范数

7.6.5 正则化方法

神经网络训练的核心目标是降低模型在真实数据分布上的期望损失，这一指标在统计学习理论中被称为风险（Risk）或期望泛化误差（Expected Generalization Error）。需要强调的是，理论上的风险是基于未知的真实分布计算的。然而在实践中，我们通常无法获得真实分布 $p(\boldsymbol{x}, y)$，只能获取包含有限 N 个样本的训练数据集 $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$。因此，学习过程转化为最小化基于训练数据集（即模型“经历”或遇到过的）定义的**经验风险**（Empirical Risk）：

$$\mathbb{E}_{\boldsymbol{x},y\sim\hat{p}(\boldsymbol{x},y)}\left[\mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}), y)\right] = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f(\boldsymbol{x}_i;\boldsymbol{\theta}), y_i) \tag{7.121}$$

其中 $\hat{p}(\boldsymbol{x}, y)$ 表示由训练数据集所定义的**经验分布**（Empirical Distribution），它通常与真实数据分布 $p(\boldsymbol{x}, y)$ 存在差异，$\boldsymbol{\theta}$ 表示神经网络的可学习参数集合，而 $\mathcal{L}(f(\boldsymbol{x};\boldsymbol{\theta}), y)$ 则是衡量模型预测 $f(\boldsymbol{x};\boldsymbol{\theta})$ 与真实观测值 y 之间差异的损失函数。这种通过最小化训练样本平均损失来训练模型的方法被称为**经验风险最小化**（Empirical Risk Minimization）。由于经验分布与真实分布之间存在偏差，过度追求经验风险最小化往往会导致模型陷入过拟合（Overfitting），即模型在训练集上表现优异，但在未见过的测试样本上性能大幅下降。为此，通常需要在最小化经验风险的基础上辅以相应的正则化（Regularization）方法，以限制模型复杂度并提高模型的泛化能力。

由于神经网络的每个神经元可视为一个广义线性模型，因此，在线性模型中常用的 ℓ_1 或 ℓ_2 等正则化方法，也可以自然地推广至神经网络的权重参数中。以 ℓ_2 为例，引入该正则化项后的**目标函数**（Objective Function）或**代价函数**（Cost Function）可表示为：

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f(\boldsymbol{x}_i;\boldsymbol{\theta}), y_i) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2 \tag{7.122}$$

其中 $\lambda > 0$ 为正则化系数，用于控制经验风险与模型复杂度之间的权衡。对代价函数关于参数 $\boldsymbol{\theta}$ 求梯度，可得：

$$\nabla\mathcal{J}(\boldsymbol{\theta}) = \nabla\mathcal{L}(\boldsymbol{\theta}) + \lambda\boldsymbol{\theta} \tag{7.123}$$

其中 $\nabla\mathcal{L}(\boldsymbol{\theta})$ 表示经验风险关于参数的梯度。若采用随机梯度下降（SGD）算法，则参数的更新公式为：

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta\nabla J(\boldsymbol{\theta}^{(t)}) \\ &= \boldsymbol{\theta}^{(t)} - \eta\left(\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) + \lambda\boldsymbol{\theta}^{(t)}\right) \\ &= \underbrace{(1-\eta\lambda)\boldsymbol{\theta}^{(t)}}_{\text{权重衰减}} - \eta\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})\end{aligned}$$

其中 $\eta > 0$ 为学习率。以上更新公式中的第一项 $(1-\eta\lambda)\boldsymbol{\theta}^{(t)}$ 表明，在每一次参数更新中，参数不仅沿着梯度方向进行调整，还会额外乘以一个小于 1 的缩放因子 $(1-\eta\lambda)$，从而使参数整体向零点方向收缩。这种在每一步迭代中对参数施加“比例性衰减”的机制，被称为**权重衰减**（Weight Decay）。值得注意的是，通常不会对偏置施加与权重相同的衰减或 ℓ_2 正则。其主要原因在于，与决定输入特征组合关系的权重参数不同，偏置参数主要用于调整神经元的激活阈值或模型输出的基准水平，通常不需要对其大小进行同等程度的约束。因此，对偏置施加正则化往往难以像约束权重那样有效地控制模型复杂度，反而可能限制模型对数据分布的适应能力，增加不必要的拟合偏差。

上述讨论表明，在使用标准的随机梯度下降（SGD）算法时，在代价函数中加入 ℓ_2 正则项与直接对参数施加权重衰减是等价的。然而，当采用 Adam 这种自适应学习率的优化算法时，这种等价性将不再成立。回顾 Adam 算法，其参数更新量 $\Delta\boldsymbol{\theta}^{(t)}$ 的计算公式为：

$$\Delta\boldsymbol{\theta}^{(t)} = -\frac{\eta}{\sqrt{\hat{\mathbf{G}}^{(t)} + \epsilon}} \odot \hat{\mathbf{M}}^{(t)} \tag{7.124}$$

其中 $\hat{\mathbf{M}}^{(t)}$ 和 $\hat{\mathbf{G}}^{(t)}$ 分别为修正后的梯度一阶矩和二阶矩的无偏估计。若仍然直接在代价函数中引入 ℓ_2 正则，则该正则项产生的梯度分量 $\lambda\boldsymbol{\theta}^{(t)}$ 会在计算参数更新量 $\Delta\boldsymbol{\theta}^{(t)}$ 时被纳入一阶矩 $\hat{\mathbf{M}}^{(t)}$ 的计算中，并最终在更新时被二阶矩的平方根 $\sqrt{\hat{\mathbf{G}}^{(t)}}$ 所除。这会导致正则化强度在不同参数上表现出不一致性：对于二阶矩较大的参数，正则化约束会被削弱；而对于二阶矩较小的参数，正则化强度反而会被放大。此外，二阶矩估计主要反映参数在优化过程中历史梯度的大小或“强度”。若对 Adam 算法不做相应调整，会造成参数更新强度越大，反而受到正则化项的约束较小，而参数更新强度越小则会受到更强的正则化作用，这显然是不合理的。因此，文献 [50] 提出在使用 Adam 优化算法时，不应将正则项合并至代价函数中计算梯度，而应将正则化与梯度更新分离，并直接对权重参数进行衰减。这种改进后的算法被称为 **AdamW**（Adam with Decoupled Weight Decay），其具体步骤是：参数首先按照 Adam 算法进行梯度更新，随后再独立执行一次权重衰减操作：

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta\lambda\boldsymbol{\theta}^{(t)} \tag{7.125}$$

综上所述，AdamW 算法的核心思想在于将“梯度更新”与“参数衰减”解耦：梯度的更新方向由 Adam 优化算法决定，而参数向零点收缩的强度则由权重衰减负责。这种解耦机制确保了正则化强度不再受自适应学习率的干扰，从而恢复了正则化在优化过程中的预期行为和作用。

在基于梯度的优化过程中，可能会出现梯度骤增甚至爆炸的情况，导致参数获得异常大的更新量，进而使优化轨迹偏离正常的路径。为了提升优化过程的稳定性，通常会采用**梯度截断**（Gradient Clipping）策略，对梯度的数值范围或梯度向量的模长进行限制。梯度截断通过抑制过大的梯度，避免参数因异常梯度而发生剧烈变化，从而提高优化过程的稳定性。与此同时，这种对梯度更新的限制也会改变模型在参数空间中的优化轨迹，并可能间接影响模型的泛化性能。因此，梯度截断也被视为一种广义的正则化方法。

梯度截断主要分为按值截断与按模截断两种方式：

- **按值截断**：对梯度向量中的每个分量分别进行限制。通常需预设一个闭区间 $[a, b]$，若梯度的某一分量小于 a，则将其强制设为 a；若大于 b，则设为 b；若落在区间 $[a, b]$ 内，则保持不变。这种操作实际上是将梯度分量限制在特定的上下界内。
- **按模截断**：对整个梯度向量的范数进行约束。通常设定一个最大范数阈值 c，若当前梯度向量的 ℓ_2 范数 $\|\mathbf{g}\|$ 超过了该阈值，则将梯度按比例缩放为：

$$\mathbf{g} \leftarrow \frac{c}{\|\mathbf{g}\|}\mathbf{g} \tag{7.126}$$

这种方式的优势在于，它能够将梯度的范数限制为 c，同时保持梯度的更新方向不变。

随机失活（Dropout）是一种计算代价较低但非常有效的正则化方法，用于缓解神经网络的过拟合问题。如图7.22所示，在训练过程中，随机失活会以一定概率将非输出层神经元的输出置零（即“失活”），从而在每一次迭代中相当于从完整网络中随机采样出一个子网进行训练；而在推理阶段，则恢复使用完整的网络。这种机制使得推理结果可被视为多个子网预测结果的近似集成。因此，采用随机失活训练的网络本质上可以看作是一个由大量共享参数的子网络构成的集成模型。集成学习通常能在保持模型偏差基本不变的同时显著降低预测方差，因而随机失活

方法能够有效地提升神经网络模型的泛化性能。

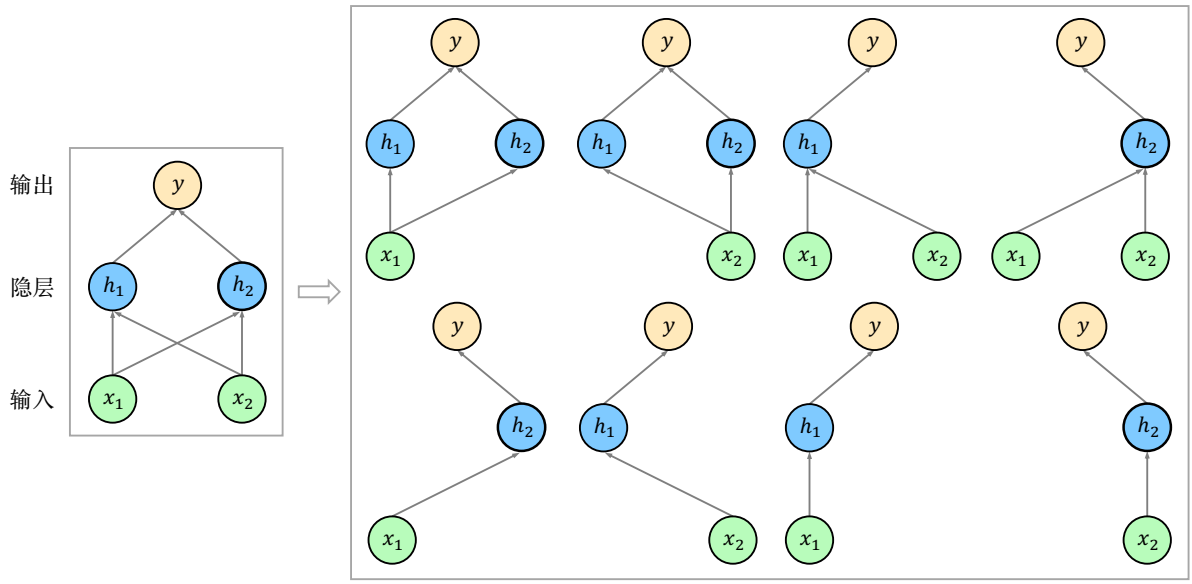


图 7.22: 随机失活（Dropout）方法本质上可以理解为在训练一个由大量子网组成的集成模型（Ensemble），这些子网通过以一定概率从完整网络中随机移除非输出单元而得到。以一个包含两个输入神经元、两个隐藏神经元和一个输出神经元所构成的基础网络为例，其四个非输出层神经元理论上共可衍生出 $2^4 = 16$ 种可能的子网，图中仅展示了其中 8 种典型的子网。

在使用随机失活训练时，需预设一个固定的失活概率 p（常取 $p = 0.5$），并利用伯努利分布来决定非输出层神经元的存活状态。假设某神经元的原始输出为 z，则在训练阶段，该神经元输出的期望为：

$$(1-p)\cdot z + p\cdot 0 = (1-p)\cdot z \tag{7.127}$$

而在推理阶段，由于不再启用随机失活机制，神经元的输出为确定值 z（即恢复为原始值）。这会导致训练与推理阶段在输出尺度上的不一致。一种直接的修正方法是在推理阶段将所有非输出层神经元的输出乘以 $(1-p)$，以匹配训练阶段的期望输出。然而，为了避免在推理阶段引入额外的计算开销，实践中常采用逆随机失活（Inverted Dropout）策略，即在训练阶段将未失活神经元的输出除以 $(1-p)$（即进行缩放补偿）；在推理阶段，无需进行任何额外处理，直接运行完整的神经网络即可保持输出期望的一致性。

神经网络训练时，通常会预设一个最大的迭代次数（或训练轮数）。当达到最大迭代次数时，训练过程将会自动终止。若在达到最大迭代次数之前提前终止训练，则称为**早停**（Early Stop）。早停是防止模型过拟合的重要手段之一。早停最常用且有效的方法是在训练集之外额外划分出一个验证集（Validation Set），并在训练过程中持续监控模型在验证集上的性能指标（如准确率）或验证损失。当验证集性能指标不再提升或验证损失不再下降时，便提前终止训练。需要注意的是，在神经网络训练过程中，验证集性能通常会存在一定的波动。有时模型可能在连续若干个训练步骤中验证集性能不再提升，但随后又继续改善。因此，在判断模型是否真正停止提升

之前，通常需要观察多个训练步骤，而不能仅依据单次性能变化做出停止的判断。实践中，通常会设置一个“耐心值”（Patience），即允许验证集性能在连续若干个训练步骤内暂时不提升，以避免由于局部波动而导致的过早停止。此外，由于验证集规模较小或训练数据噪声较大等原因，验证集性能指标或验证损失有时会出现较明显的随机波动。为了更稳定地观察验证集性能的变化趋势，常常会采用滑动平均或指数滑动平均等方法，对多个训练步骤上的验证集性能指标或验证损失进行平滑处理，从而更稳健地判断模型性能的整体变化趋势。在使用早停策略的过程中，通常还会配套使用检查点（Checkpoint）机制，即在训练过程中始终保存验证集性能最优的一组模型参数。如此，即使后续训练过程中模型性能下降，或者训练过程意外终止，也能够恢复到验证集性能最佳时的模型参数。

在其他因素保持不变的前提下，神经网络的性能通常随高质量训练样本数量的增加而提升。在训练数据有限的情况下，**数据增强**（Data Augmentation）是提高模型鲁棒性与泛化能力的重要方法之一。数据增强是指在不破坏样本语义（即不改变输入样本与其对应输出或标签之间的映射关系）的前提下，通过对原始训练样本进行扰动、变换或加噪，从而扩充训练数据规模的方法。以图像识别为例，常用的数据增强方法包括几何变换（如旋转、翻转、缩放、平移）以及像素级扰动（如加噪、调整对比度等）。这些变换和扰动实质上是向模型引入了关于任务的不变性先验（例如翻转图像是为了让模型学习“翻转不变性”），从而使模型能够学习到对各种变换和扰动更鲁棒的特征表示。

在分类任务中，通常使用独热编码（One-hot Encoding）来表示类别标签，并且结合交叉熵（Cross-entropy）损失函数进行训练。独热编码要求模型对正确类别的预测概率接近于 1，而对其他类别趋近于 0。这要求网络最末的线性层输出（即 Softmax 前的 Logits）中，正确类别的值相对于其他类别趋向于正无穷大。这种过于“极端”的优化目标容易导致模型产生过度“自信”（Overconfidence），即让模型倾向于输出过于尖锐的概率分布，从而降低模型的泛化能力。此外，由于大规模数据集中不可避免地存在标注噪声，强制让模型预测“非 1 即 0”的硬标签（Hard Labels），模型会过度拟合这些错误标签，从而放大噪声样本对训练过程的负面影响。为了缓解上述问题，**标签平滑**（Label Smoothing）通过对独热编码标签引入少量“平滑噪声”，使模型在预测时留有“余地”，从而提升模型的鲁棒性与泛化能力。具体而言，原始独热编码的硬标签：

$$t_i = \begin{cases} 1 & \text{若样本属于第 } i \text{ 类} \\ 0 & \text{否则} \end{cases} \tag{7.128}$$

修改为如下形式的软标签（Soft Labels）：

$$t_i = \begin{cases} 1-\epsilon & \text{若样本属于第 } i \text{ 类} \\ \dfrac{\epsilon}{K-1} & \text{否则} \end{cases} \tag{7.129}$$

其中 $\epsilon \in (0,1)$ 为一个较小的平滑系数，而 K 则为类别总数。通过这种方式，标签平滑将一小部分概率质量分配给其他类别，从而避免模型输出过于极端或尖锐的概率分布，并减弱少量错误标注样本对模型的影响。

上述标签平滑方法将从目标类别中减少的概率质量 ϵ 平均分配给所有非目标类别。然而，在

某些分类任务中，不同类别之间往往具有不同程度的语义相似性。因此，更合理的做法是对与目标类别更相近的类别分配更高的概率，而对差异较大的类别分配更低的概率。例如，当目标类别为“猫”时，可以为“老虎”等语义更接近的类别分配相对较大的概率，而为“汽车”等无关类别分配较小的概率。这种基于类别相似性的标签平滑方法在一定程度上可以反映类别之间的语义关系（即引入了类别间的先验结构信息），从而为模型提供更加合理的监督信号。然而，该方法通常需要事先获得类别之间可靠且准确的相似度度量，而这在实践中往往较为困难。如果类别相似性估计不准确，反而可能引入错误的先验偏置，从而对模型训练产生不利影响。

如果存在一个性能较好的教师模型（通常具有更大的参数规模，且在更大规模数据上经过充分训练），则一种更有效的“软标签”学习方法是**知识蒸馏**（Knowledge Distillation）[36]。在知识蒸馏中，学生模型的学习目标不再是传统的独热编码或简单平滑后的标签，而是拟合教师模型所产生的输出分布（通常为教师模型 Softmax 层的预测概率或经缩放后的 Logits 分值）。与独热编码相比，教师模型的预测分布通常更加平滑，并能够体现不同类别之间的相似性与竞争关系，从而传递更丰富的类别结构信息。此外，知识蒸馏不仅可以让学生模型“模仿”教师模型的最终输出，还可以进一步扩展到中间层特征的对齐，即通过学习使学生模型产生与教师模型相同或相似的隐层表示，从而获得更丰富的特征表示能力。总体而言，知识蒸馏能够将参数规模较大、性能更优的教师模型中所蕴含的“知识”有效迁移至参数规模更小的学生模型中，使得小模型在保持较低计算与存储开销的同时，仍然能够获得接近教师模型的性能表现。

多任务学习（Multi-task Learning）是另一种常用的“隐式”正则化方法，其核心思想是在同一模型中共享全部或部分参数，同时优化多个相关任务的加权损失函数。当多个任务之间存在较强相关性并且能够相互补充信息时，多任务学习尤为有效。它也可以被理解为另一种形式的数据增强：来自不同任务的监督信号对共享参数和中间表示施加了额外约束，迫使模型学习更通用、更稳定的特征表达。例如，在计算机视觉中，可以在图像分类的同时引入目标检测或语义分割任务；在自然语言处理中，在进行情感分析的同时引入语法分析或实体识别任务；在自动驾驶中，同时预测车辆的行驶轨迹与周围障碍物的类型。从优化角度来看，多任务学习通过多个任务损失函数的共同约束，使模型参数不会过度拟合某一个任务的特定模式，而是在多个任务之间寻求折中解。这种“相互牵制”的作用会限制参数的搜索空间，使其更倾向于学习任务之间共享的稳定特征表示，从而提升模型的鲁棒性与泛化能力。

7.6.6 超参数优化

神经网络模型的构建与训练涉及众多超参数，其取值在很大程度上决定了模型性能的上限。表7.3列出了神经网络中常见的主要超参数，其中网络结构部分以卷积神经网络为例。若采用其他类型的网络结构，相应的结构类超参数会发生替换或增减。此外，部分超参数还可以进一步细化。例如，学习率不仅可以表示为一个固定值（即恒定学习率），还可以细分为初始学习率和最终学习率。超参数既可以是连续变量（如学习率、权重衰减系数），也可以是离散变量（如批量大小、网络层数）。不同超参数之间的组合及其相互作用，会对模型的收敛速度和最终性能产

表 7.3: 神经网络的主要超参数

类别	超参数
训练优化	学习率（Learning Rate） 学习率衰减策略及比例系数（Scheduler） 批量大小（Batch Size） 最大训练轮次（Epoch）或最大迭代步数（Step） 预热（Warm-up）迭代步数 优化算法类型（Optimizer） 梯度一阶矩衰减率或动量（Momentum）因子 梯度二阶矩衰减率（Decay Rate）
正则化方法	权重衰减（Weight Decay）或 ℓ_2 正则系数 随机失活（Dropout）概率 梯度截断（Gradient Clipping）阈值 早停（Early Stop）的耐心值（Patience） 标签平滑（Label Smoothing）系数 数据增强（Data Augmentation）方法及比例 多任务学习（Multi-task Learning）各任务损失权重
网络结构 （以卷积神经网络为例）	网络层数或网络深度（Depth） 隐层维度或网络宽度（Width） 激活函数类型（Nonlinear Activation） 卷积通道数（Channels）或特征图数量（Feature Maps） 卷积核大小（Kernel Size）及步长（Stride）

生显著影响。因此，超参数选择与优化本质上是一个典型的组合优化问题，即在有限计算资源与训练时间情况下，寻找一组能够使模型在验证集上获得最佳泛化性能的超参数配置。

最简单的超参数优化方法是**网格搜索**（Grid Search）。该方法首先为每个超参数预先设定若干候选取值，随后通过穷举所有可能的参数组合来训练模型，并根据各组合在验证集上的性能表现来择优。虽然不同组合的训练可以并行化，但其总体计算开销随超参数数量及其候选取值数量呈指数增长。特别是当需要优化的超参数较多时，搜索空间会迅速膨胀，使得在有限计算资源下难以完成全面且有效的参数搜索。

由于神经网络最终性能对不同超参数的敏感程度并不相同，即某些关键超参数会显著影响模型性能，而另一些超参数在合理范围内变化时对最终结果的影响则相对有限。基于这一观察，文献 [6] 提出了**随机搜索**（Random Search）策略：不再穷举所有可能的超参数组合，而是在预先设定的超参数空间中进行随机采样，选取若干组合来训练模型，并根据其在验证集上的性能表现选择最优组合。该方法的核心逻辑在于：只要关键超参数的最优值或近似最优值在随机采样过程中被覆盖，即使该组合中其他非关键超参数并非最优（其影响相对较小），模型仍可以获得接近最优的性能表现。以包含一个关键超参数和一个非关键超参数的情况为例，并假设总计算资源允许进行 9 次训练。如果采用网格搜索，将形成 3×3 的组合网格，此时关键超参数仅被尝

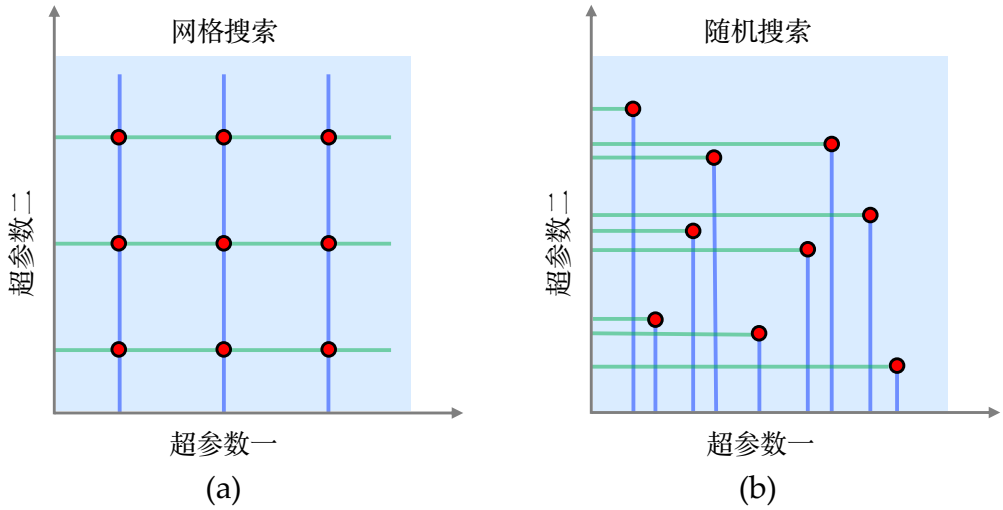


图 7.23: 网格搜索与随机搜索的对比示意图。红色圆点表示超参数采样点。在相同的搜索预算（图中均为 9 次评估）下，对于同一超参数空间，随机搜索 (b) 较网格搜索 (a) 能够覆盖更多不同的取值，从而具有更高的搜索效率。

试 3 种不同取值。而在随机搜索中，则会在超参数的合理范围内独立进行 9 次采样，从而使关键超参数有 9 次机会探索不同取值，显著提高其覆盖最优区域的概率（如图7.23所示）。随机搜索不需要像网格搜索那样将某一超参数的每个取值与其他超参数的所有候选取值进行组合，而是以随机采样的方式对整个超参数空间进行探索。因此，随机搜索通常比网格搜索具有更高的搜索效率，并且常作为初步探索方法，用于快速定位较优的超参数区域，随后再在该区域内进行局部精细搜索。然而，当重要超参数数量较多且彼此存在较强耦合关系时，纯随机采样仍可能错过全局最优或近似最优的超参数组合。

为了降低网格搜索或随机搜索在超参数优化过程中的盲目性，需要构建一种能够刻画超参数配置与神经网络性能之间关系的模型，从而为后续的超参数搜索提供方向性指引。这类模型相当于真实神经网络的代理或低成本替代。它无需将每组超参数都在真实神经网络上进行高代价的训练与测试，就能够近似预测不同超参数配置所对应的模型性能。因此，这种模型被称为代理模型（Surrogate Model）。在获得代理模型后，便可以结合相应的搜索策略（即采集函数），更高效地探索最有可能提升神经网络性能的超参数配置。

高斯过程（参见第4.3节）是构建此类代理模型的常用方法 [71]。与传统方法不同，它并不显式假设代理模型具有某种固定的函数形式，而是假设代理模型所有预测值的联合分布服从多维正态分布，且该分布的协方差矩阵由刻画输入空间中超参数配置间相似度的格拉姆矩阵确定。格拉姆矩阵中的每个元素对应两个超参数配置之间的核函数值，从而反映它们在输入空间中的相似程度。这一假设具有直观的合理性：即相似的超参数配置通常应产生相近的神经网络性能。此外，高斯过程具有较强的灵活性：我们可以通过选择不同形式的核函数来定义“相似性”的度量方式，从而调节预测函数的平滑程度以及先验假设的强弱。采用高斯过程建模的另一个重要优点在于，它能够同时提供预测的均值 $\mu(\boldsymbol{\lambda})$ 与方差 $\sigma^2(\boldsymbol{\lambda})$（其中 $\boldsymbol{\lambda}$ 表示某一超参数配置）。这

使得在超参数搜索过程中可以同时考虑“性能提升的期望”（由均值体现）与“预测的不确定性”（由方差体现），从而在利用已知区域与探索未知不确定区域之间实现更合理的权衡。

初始代理模型仍然需要少量经实测的超参数配置及其对应的真实性能值来进行构建。具体而言，我们可以先从超参数空间中随机采样若干组超参数配置，并分别在这些配置下训练神经网络，从而获得其实际性能表现。随后，利用这些输入（超参数配置）与输出（真实性能值）所构成的训练集来构建基于高斯过程的初始代理模型。之后，将使用采集函数（Acquisition Function）来获取下一个值得尝试的超参数配置。置信上界（Upper Confidence Bound, UCB）是一种经典的采集函数，其核心思想是在预测均值的基础上，引入不确定性（以标准差衡量）作为探索奖励。然而，超参数优化的目标通常是最小化验证集损失，因此更常使用其变体，即置信下界（Lower Confidence Bound, LCB），其定义为：

$$\mathrm{LCB}(\boldsymbol{\lambda}) = \mu(\boldsymbol{\lambda}) - \kappa \cdot \sigma(\boldsymbol{\lambda}) \tag{7.130}$$

其中 $\mu(\boldsymbol{\lambda})$ 与 $\sigma(\boldsymbol{\lambda})$ 分别是代理模型所给出的预测均值与标准差，$\kappa \geq 0$ 则用于控制探索与利用之间的权衡。下一步需要尝试的超参数配置通过最小化置信下界得到，即：

$$\boldsymbol{\lambda}_* = \arg\min_{\boldsymbol{\lambda}} \mathrm{LCB}(\boldsymbol{\lambda}) \tag{7.131}$$

为了直观理解 κ 的作用，假设有两个超参数配置 $\boldsymbol{\lambda}_1$ 和 $\boldsymbol{\lambda}_2$，代理模型对其进行预测的损失均值与标准差分别为：

$$\mu(\boldsymbol{\lambda}_1) = 0.7,\ \sigma(\boldsymbol{\lambda}_1) = 0.1$$
$$\mu(\boldsymbol{\lambda}_2) = 0.8,\ \sigma(\boldsymbol{\lambda}_2) = 0.5$$

若设 $\kappa = 0.2$，则有：

$$\mathrm{LCB}(\boldsymbol{\lambda}_1) = 0.7 - 0.2 \times 0.1 = 0.68 \text{（选择 } \boldsymbol{\lambda}_1\text{）}$$
$$\mathrm{LCB}(\boldsymbol{\lambda}_2) = 0.8 - 0.2 \times 0.5 = 0.70$$

若设 $\kappa = 0.5$，则有：

$$\mathrm{LCB}(\boldsymbol{\lambda}_1) = 0.7 - 0.5 \times 0.1 = 0.65$$
$$\mathrm{LCB}(\boldsymbol{\lambda}_2) = 0.8 - 0.5 \times 0.5 = 0.55 \text{（选择 } \boldsymbol{\lambda}_2\text{）}$$

由此可见，当 κ 较小时，更偏向选择损失均值较低（对应性能更优）的配置，即倾向于利用已有的低损失（高性能）区域（降低对不确定性的奖励）；而当 κ 较大时，则更加重视不确定性，从而更倾向于探索预测不确定性较高的区域。

既然代理模型能够预测不同超参数配置对应性能的概率分布，那么更合理且稳健的策略是优先选择那些“期望上能够带来最大性能提升”的超参数配置。期望提升（Expected Improvement, EI）是另一种经典的、且更为常用的采集函数，其基本思想是：选择能够相对于当前最优结果带来最大期望改进的超参数配置。其定义为：

$$\mathrm{EI}(\boldsymbol{\lambda}) = \int_{-\infty}^{+\infty} \max\left[0, \mathcal{L}_* - f(\boldsymbol{\lambda})\right] \cdot p(f|\boldsymbol{\lambda}, \mathcal{D})\, df \tag{7.132}$$

其中 $\mathcal{L}_*$ 表示当前已观测到的最小验证损失（即当前最优性能），$\mathcal{D}$ 为已有的观测数据集，$\boldsymbol{\lambda}$ 为待评估的候选超参数配置，$f(\boldsymbol{\lambda})$ 表示高斯过程代理模型在 $\boldsymbol{\lambda}$ 处的预测值，其服从如下正态分布：

$$p(f|\boldsymbol{\lambda},\mathcal{D}) = \mathcal{N}(f|\mu(\boldsymbol{\lambda}),\sigma^2(\boldsymbol{\lambda})) \tag{7.133}$$

公式7.132中的 $\max[0,\mathcal{L}_* - f(\boldsymbol{\lambda})]$ 衡量了在 $\boldsymbol{\lambda}$ 处的预测值相较于当前最优值 $\mathcal{L}_*$ 的改进量（若预测损失 $f(\boldsymbol{\lambda})$ 大于 $\mathcal{L}_*$，则改进量为 0）。通过最大化该期望值得到的 $\boldsymbol{\lambda}$，即为预期能带来最大性能改善（即验证损失下降最多）的超参数配置。

最小化置信下界 LCB($\boldsymbol{\lambda}$) 或最大化期望提升 EI($\boldsymbol{\lambda}$) 通常是复杂的非凸优化问题，往往难以获得解析解。此外，由于超参数空间一般同时包含连续变量与离散变量，也难以直接应用梯度下降等基于导数的优化方法。因此，一种较为实用的解法是在超参数空间中随机采样足够数量的候选配置，并利用代理模型对这些配置进行快速评估，然后依据相应的采集函数选择最优的超参数组合作为下一步实际尝试的配置。在获得最优的超参数配置后，需要使用该配置对神经网络进行实际训练与验证，并将得到的真实性能结果加入已有观测数据集 $\mathcal{D}$ 中，并且据此更新基于高斯过程的代理模型。上述过程体现了贝叶斯分析的核心思想：将当前轮次得到的代理模型视为下一轮的先验分布，通过实际观测获取新的样本数据，再利用这些数据对模型进行更新，从而得到更加准确的后验分布。因此，这类方法被称为**贝叶斯优化** (Bayesian Optimization)。

除了高斯过程外，还可以使用随机森林（详见第5.4节）来替代高斯过程构建代理模型。在随机森林代理模型中，预测均值由“森林”中所有决策树预测结果的算术平均值给出，而预测的不确定性（方差）则通过各棵树预测值之间的差异进行估计。随机森林同样适用于超参数空间同时包含连续变量与离散变量的情形，并且在计算效率上（尤其是在高维）通常优于高斯过程。然而，随机森林提供的不确定性估计属于经验性近似，因此在不确定性度量的严谨性和精确性方面通常不及具有严格统计学理论支撑的高斯过程。

尽管贝叶斯优化具有严谨的统计学理论基础，但其明显的局限性在于难以实现高效的并行搜索。由于其本质上是一种序贯决策模型，即每一轮超参数配置的选择都依赖于上一轮的观测结果，导致后续探索通常需要等待当前配置评估结束后方可进行。为了更好地支持并行搜索，并在优化过程中充分利用中间评估结果以动态调整计算资源分配，研究者提出了**资源重配方法**。这类方法通常不显式构建代理模型，而是将超参数优化问题视为在有限计算资源下，尽快识别并剔除性能较差的配置，从而将更多资源分配给更有潜力的候选配置。其中较具代表性的方法是逐步减半（Successive Halving）算法 [42]。其基本思想是：首先为一组数量较多的候选超参数配置分配相同且较小的训练预算（如较少的训练轮数或迭代次数）进行并行训练；随后在当前预算耗尽后，根据验证集性能对这些配置进行评估，并淘汰其中表现较差的一半配置；接着对剩余配置增加训练预算（如按固定倍数扩展），并重复训练、评估、淘汰的过程，直至筛选出最优的配置。然而，这类方法也存在一定风险：如果某些配置在初期收敛较慢但最终表现出较高性能（即具有较大的后期潜力），它们可能在早期阶段被过早淘汰。因此，这类方法对预算分配策略、早停规则以及评估指标的选择（尤其是在训练早期阶段）具有较高的敏感性。

传统超参数优化方法通常将超参搜索与模型训练视为两个独立的阶段：首先通过大量尝试

确定最优超参数配置，然后再利用该配置重新开始正式的模型训练。然而，这类方法存在两个主要局限。首先，在超参数搜索阶段所消耗的计算资源通常无法直接用于最终模型训练；其次，这类方法通常只能得到一组固定的超参数配置，难以根据训练过程中不同阶段的需求动态调整超参数。由于大规模神经网络训练过程高度复杂，最优超参数配置往往随训练进程而变化，单一静态的配置难以获得最优性能。为此，DeepMind 团队于 2017 年提出了**基于种群的训练方法**（Population-Based Training）[41]，该方法能够在训练神经网络权重的同时动态优化超参数。基于种群的训练方法借鉴了遗传算法的思想：性能较好的模型个体能够将其“基因”（即模型权重与超参数）传递给表现较差的个体，并在此基础上进行随机变异，从而通过高效的并行搜索逐步逼近最优或近似最优解。该训练方法通常包含以下四个主要步骤：

- 首先，随机采样一组超参数配置形成初始种群，每个模型个体都拥有各自的超参数及随机初始化的网络权重；
- 随后，所有个体在并行环境下独立训练固定步数（例如若干训练轮次或一定数量的迭代）；
- 接着，在阶段性训练结束后，根据验证集性能对所有个体进行评估与排序。对于表现较好的个体，保留并继续训练；而对于表现较差的个体（例如位于后 20% 的个体），则丢弃其当前权重与超参数，并复制种群中最优个体的权重与超参数；
- 最后，为了避免种群中的个体逐渐趋于一致，从而丧失种群多样性，复制后的个体还会以一定概率对超参数进行随机扰动或变异（例如将学习率乘以 0.8 或 1.2 等）。

上述过程不断重复，直至耗尽预设的计算资源。基于种群的训练方法的优势在于，它能够自动学习超参数随训练过程动态变化的策略。与传统“先搜索、后训练”的超参数优化方法不同，该方法在超参数搜索过程中已经同步完成了模型训练，因此最终还能直接获得已训练完成的模型。

7.7 Transformer

虽然长短期记忆网络（LSTM）通过引入显式的记忆单元与门控机制，在一定程度上缓解了基础循环神经网络难以捕捉长程依赖关系的问题，但其有效的建模距离依然受限（通常认为仅能建模数百个时间步之内的依赖关系）。此外，当 LSTM 作为编码器对一个序列生成特征向量表示时，通常需要将序列最后一个时刻的隐状态作为整个序列的表示。然而，LSTM 各时刻隐状态的生成存在严格的时序依赖，即每一时刻的隐状态计算均以前序状态为前提，因此序列中的各时间步无法并行计算。这种串行计算方式限制了模型对序列进行高效编码的能力，也使得难以利用现代并行硬件（如 GPU）快速生成给定序列的特征表示。

人类语言（即自然语言）的处理通常要求模型具备较强的长程依赖关系建模能力。例如，在对话系统中，模型需要利用对话的历史上下文；在篇章理解任务中，模型则需要捕捉整个篇章中对于同一人物、事件或论点在不同位置的描述，以及这些描述之间的关联关系。在处理较长的自然语言文本时，基础循环神经网络和 LSTM 采用了一种“压缩”式的记忆方式，即将历史信息与当前输入编码为固定维度的记忆向量。然而，这种固定容量的表示方式不可避免地会导致部分信息被“遗忘”。相比之下，更有效建模长程依赖关系的方法是“检索”。在这种方法中，不同

时刻的历史信息都会被保留下来，并以如键值对（Key-value Pairs）的形式进行组织。模型可以根据当前输入构造“查询”（Query）从历史信息中动态寻找与当前输入最相关的内容，并将检索得到的信息进行融合。这种方法理论上能够处理任意长度的历史信息，并且有效建模长程依赖关系。目前应用广泛的 Transformer 架构正是这一“检索”思想的体现 [77]，其核心实现方式是自注意力机制（Self-attention），更准确地说，是基于点积的**自注意力机制**（Dot-Product Attention）。之所以称为“自注意力”，是因为注意力计算发生在同一输入序列内部的不同位置之间，即序列中的每个位置都会基于自身的信息，动态关注序列中其他位置的信息。

假设输入序列的长度为 T，并且对于序列中任一位置或时刻 j，都有一个查询向量 $\boldsymbol{q}_j = (q_{j1}, \cdots, q_{jD})^\top$，一个键向量 $\boldsymbol{k}_j = (k_{j1}, \cdots, k_{jD})^\top$ 和一个值向量 $\boldsymbol{v}_j = (v_{j1}, \cdots, v_{jD})^\top$。其中，查询向量可以理解为当前位置‘想获取什么信息”，键向量表示当前位置“能提供什么信息”，而值向量则表示当前位置“实际存储的信息”。在基于点积的自注意力机制中，首先利用当前位置的查询向量与序列中所有位置（包括其自身）的键向量分别计算点积，从而得到对应的注意力分值（Attention Score）。由于向量点积能够衡量两个向量间的相似程度，因此注意力分值越大，表示对应位置与当前位置的相关性越强，其矩阵运算形式如下：

$$\begin{bmatrix} k_{11} & \cdots & k_{1D} \\ \vdots & \ddots & \vdots \\ k_{j1} & \cdots & k_{jD} \\ \vdots & \ddots & \vdots \\ k_{T1} & \cdots & k_{TD} \end{bmatrix} \begin{bmatrix} q_{j1} \\ \vdots \\ q_{jD} \end{bmatrix} = \begin{bmatrix} s_{j1} \\ \vdots \\ s_{jj} \\ \vdots \\ s_{jT} \end{bmatrix} \tag{7.134}$$

此处假设查询向量、键向量和值向量均为 D 维。在实际应用中，查询向量与键向量的维度必须保持一致以进行点积运算，而值向量的维度则可以有所不同。获得注意力分值后，需将其转化为归一化概率分布，即使所有分值非负且总和为 1，从而反映不同位置相对的相关或重要程度。实现这一转化最常用的方法是采用 Softmax 函数。需要注意的是，随着维度 D 的增大，点积结果的数值尺度通常也会随之增大，这可能使 Softmax 函数进入梯度较小的饱和区域，从而影响模型训练的稳定性。因此，在执行 Softmax 变换之前，通常会将注意力分值除以缩放因子 $\sqrt{D}$。在得到注意力权重（即概率分布）之后，自注意力机制会对所有位置的值向量进行加权求和，并将所得结果 $(r_{j1}, \cdots, r_{jD})^\top$ 作为当前位置新的特征表示：

$$a_{j1} \begin{bmatrix} v_{11} \\ \vdots \\ v_{1D} \end{bmatrix} + \cdots + a_{jj} \begin{bmatrix} v_{j1} \\ \vdots \\ v_{jD} \end{bmatrix} + \cdots + a_{jT} \begin{bmatrix} v_{T1} \\ \vdots \\ v_{TD} \end{bmatrix} = \begin{bmatrix} r_{j1} \\ \vdots \\ r_{jD} \end{bmatrix} \tag{7.135}$$

第 j 个位置的注意力计算完整过程如图7.24所示。关于为何可以将值向量的加权和作为当前位置新特征表示的问题，可以理解为一种上下文信息融合的过程：该表示通过注意力权重对序列中不同位置的信息进行加权，从而整合全局上下文信息，并生成上下文相关的特征表示。以自然语言处理为例，词汇“苹果”具有典型的多义性：它既可以表示一种水果，也可以指代特定的品牌或公司。在不考虑语境的情况下，该词的语义表征是模糊且具有歧义的。然而，若上下文中

出现了“品尝”、“香甜”或“采摘”等词，模型通过自注意力机制捕捉到这些关联，便能判定其语义偏向水果（通过融入这些词的语义特征，其特征向量表示向水果类别的语义空间偏转）；若上下文中出现了“手机”、“屏幕”或“市值”等词，模型则会倾向于将其解释为品牌或公司的含义。

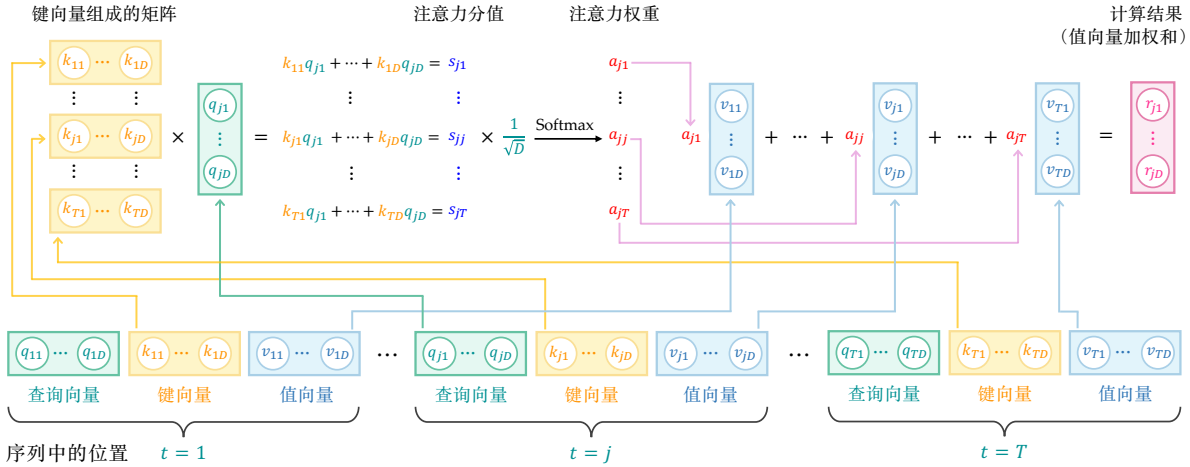


图 7.24: 基于点积的自注意力机制示意图。序列中每个位置都有对应的查询向量 $\boldsymbol{q}$，键向量 $\boldsymbol{k}$ 和值向量 $\boldsymbol{v}$。在计算第 j 个位置的注意力输出时，首先使用该位置的查询向量 $\boldsymbol{q}_j$ 与序列中所有位置的键向量逐一进行点积，得到注意力分值；随后对缩放后的分值（即除以 $\sqrt{D}$）应用 Softmax 函数，将其转换为注意力权重（概率分布）；最后利用该权重对所有值向量进行加权求和，并将结果 $(r_{j1}, \cdots, r_{jD})^\top$ 作为第 j 个位置更新后的特征表示。

单次注意力计算通常只能建模一种特定的语义依赖关系。类似于卷积神经网络中引入多组不同的卷积核以提取多种特征图，也可以通过多组注意力从不同特征空间中捕捉序列内多样化的语义依赖关系，这就是**多头注意力**（Multi-head Attention）机制。为了实现多头注意力，在序列的每个位置上需要构造多组不同的查询向量、键向量和值向量。这些向量通常是通过对同一输入向量（或前层输出）分别施加不同的线性变换（由不同的可学习参数矩阵确定）而得到的。

假设某一位置的输入为一个 M 维向量 $\boldsymbol{x}$。对于第 i 个注意力头（$i = 1, \ldots, H$），分别通过三组可学习的线性变换矩阵 $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i \in \mathbb{R}^{D \times M}$ 将该输入向量 $\boldsymbol{x}$ 映射到相应的特征子空间，从而得到该位置第 i 个注意力头的查询向量 $\boldsymbol{q}^i$、键向量 $\boldsymbol{k}^i$ 和值向量 $\boldsymbol{v}^i$：

$$\boldsymbol{q}^i = \mathbf{W}_q^i \boldsymbol{x}, \quad \boldsymbol{k}^i = \mathbf{W}_k^i \boldsymbol{x}, \quad \boldsymbol{v}^i = \mathbf{W}_v^i \boldsymbol{x} \tag{7.136}$$

对于序列中的其他位置，同样使用这三组变换矩阵生成第 i 个注意力头的查询向量、键向量和值向量。在各注意力头独立计算得到该位置的输出向量 $\boldsymbol{r}^i$ 之后，将所有注意力头的输出向量进行拼接，并通过另一个线性变换矩阵 $\mathbf{W}_o \in \mathbb{R}^{HD \times M}$ 对拼接后的向量进行线性整合，最终生成与输入向量维度一致的输出向量 $\boldsymbol{h} \in \mathbb{R}^M$：

$$\boldsymbol{h} = \left((\boldsymbol{r}^1)^\top, \cdots, (\boldsymbol{r}^H)^\top \right) \mathbf{W}_o \tag{7.137}$$

因此，多头注意力模块的输入向量与输出向量具有相同的维度，这种等维映射使得多头注意力模块能够方便地在深度网络中进行层层堆叠。值得注意的是，对于给定的输入序列，所有位置

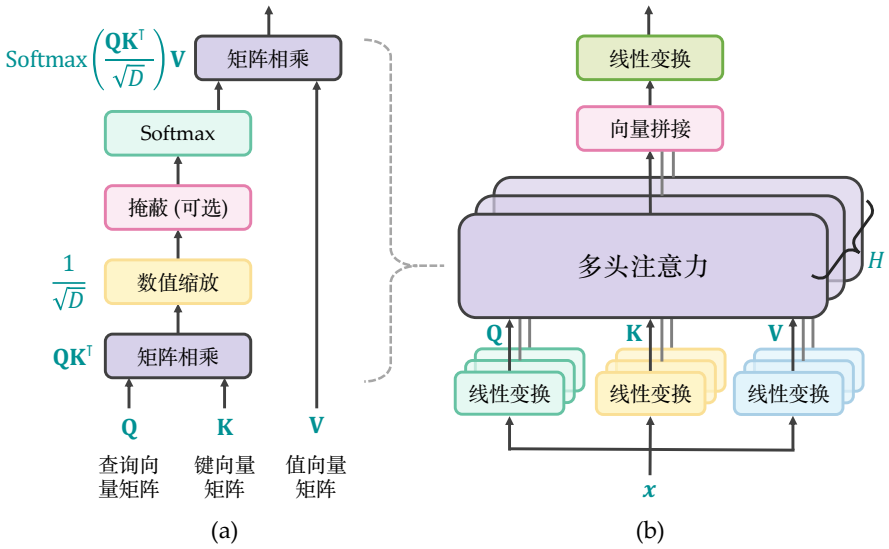


图 7.25: 自注意力机制示意图。(a) 点积注意力的单次计算流程。(b) 多头注意力计算过程。首先通过多组不同的线性变换矩阵将输入向量 $\boldsymbol{x}$ 映射至不同的子空间，分别生成各注意力头的查询向量、键向量与值向量；然后各注意力头独立执行点积注意力的所有计算步骤；最后将所有注意力头的输出向量进行拼接，并通过线性变换进行整合，得到最终输出表示。

的注意力输出都可以通过矩阵运算并行计算得到：

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{D}}\right)\mathbf{V} \tag{7.138}$$

其中矩阵 $\mathbf{Q}$、$\mathbf{K}$ 和 $\mathbf{V}$ 的各行分别对应序列中各位置的查询向量、键向量和值向量。Softmax 按矩阵的行进行归一化，从而得到每个位置相对于序列中所有位置的注意力权重。

图7.25展示了点积注意力及多头注意力的计算流程。在单次注意力计算中，包含一个可选的掩蔽（Masking）操作。引入该操作的原因在于处理序列生成的因果约束。当注意力机制用于编码器（Encoder）时（即将给定序列转换为包含上下文语义信息的高层表示），序列各位置均可自由融合双向的上下文信息；然而，当其用于解码器（Decoder）进行序列生成时，模型需要遵循**自回归**（Auto-regressive）原则。具体而言，在推理阶段，解码器通常采用逐位生成的策略，即利用当前已经生成的部分序列预测下一位置的输出，并将该输出追加到已有序列中，再继续后续预测。在这一过程中，当前位置显然不应访问尚未生成的“未来”信息。为了使模型在训练阶段能够模拟这种自回归生成过程，需要在注意力计算中引入掩蔽操作。具体做法是：将注意力得分矩阵中对应“当前位置之后”位置的分值强制设为负无穷，使得经过 Softmax 归一化后的注意力权重趋近于零。这样既能够有效防止未来信息泄露到当前预测中，也能够保证训练与推理阶段模型行为的一致性。这种引入了掩蔽操作、并限制模型只能关注当前位置及其之前信息的自注意力机制，也被称为**因果注意力**（Causal Attention）。多头注意力中，除了每个注意力头都需要独立完成点积注意力的全部计算步骤之外，还包括：在注意力计算之前，通过多组线性变换矩阵为各注意力头分别生成查询向量、键向量和值向量；并在注意力计算之后，将各注意力头的输出向量进行拼接，并通过线性变换对其进行整合输出。

如图7.26所示，Transformer 架构由编码器和解码器两部分组成。编码器由多个相同结构的模块堆叠而成，每个模块由多头注意力子层和前馈网络子层组成，每个子层均集成了残差连接与层内归一化[6]。多头注意力子层的计算过程可表示为：

$$\text{MHA-Sublayer}(\boldsymbol{x}) = \boldsymbol{x} + \text{MHA}(\text{LayerNorm}(\boldsymbol{x})) \tag{7.139}$$

其中 $\boldsymbol{x}$ 为该子层的输入向量，$\text{MHA}(\cdot)$ 表示多头注意力模块，$\text{LayerNorm}(\cdot)$ 为层内归一化。上式中的相加操作实现了残差连接。随后，数据进入前馈网络子层，其计算过程如下：

$$\text{FFN-Sublayer}(\boldsymbol{x}) = \boldsymbol{x} + \text{FFN}(\text{LayerNorm}(\boldsymbol{x})) \tag{7.140}$$

其中前馈网络 $\text{FFN}(\cdot)$ 通常由两个线性变换层及中间的非线性激活函数组成，其具体形式为：

$$\text{FFN}(\boldsymbol{x}) = \mathbf{W}_2\,\text{ReLU}(\mathbf{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 \tag{7.141}$$

其中 $\mathbf{W}_1$ 和 $\mathbf{W}_2$ 分别为两层线性变换的权重矩阵，而 $\boldsymbol{b}_1$ 和 $\boldsymbol{b}_2$ 为对应的偏置向量。通常情况下，第一个线性层会将特征维度进行升维，而第二个线性层将其映射回原始维度。

Transformer 的解码器同样由多个结构相同的模块堆叠而成，每个模块由因果注意力子层、交叉注意力子层和前馈网络子层组成。其中前馈网络子层与编码器中前馈网络子层的结构相同。引入因果注意力子层是为了在序列生成时遵循自回归原则，通过在注意力计算中加入掩蔽操作，使注意力权重矩阵呈现下三角形状（如图7.26右下蓝色网格所示）。该矩阵上三角元素均为零，从而阻断了模型对当前位置之后“未来信息”的访问。**交叉注意力**（Cross-attention）子层则负责建立解码器与编码器输出之间的关联。以英译中的翻译任务为例，编码器处理输入的英文语句，而解码器负责生成对应的中文语句。在生成每一个中文单词时，模型需要通过交叉注意力机制在完整的英文语句中检索与当前生成最相关的语义信息。因此，该子层的键向量 $\mathbf{K}$ 和值向量 $\mathbf{V}$ 来自编码器的顶层输出，而查询向量则源自解码器当前生成的中间表征。

注意力机制能够有效建模序列中的长程依赖关系。当应用于编码器时，所有位置的注意力计算可以并行进行（不同位置共享同一组参数），从而显著提升序列编码的效率。然而，与逐位处理序列输入的循环神经网络不同，注意力机制无法天然捕捉序列内部的时序关系。若不显式引入位置信息，则无论如何改变序列中元素的排列顺序，每个元素的注意力计算结果（以加权和形式计算）都不会发生变化，即注意力机制对输入元素的排列顺序具有**置换等变性**。在实际的序列建模中，位置相关的时序信息至关重要。以自然语言为例，句子并非词汇的无序堆砌，而是按照语义与语法规则有序组织的序列。为了在模型中补充位置相关的时序特征，Transformer 引入了**位置编码**（Positional Encoding），即在每个位置的输入向量（在自然语言处理中通常为词向量）上叠加一个表征其在序列中位置的向量，从而使输入向量表示同时包含内容信息与位置

[6]原始 Transformer 采用层后归一化（Post-layer Normalization），即将层内归一化放置在多头注意力或前馈网络及残差连接之后。这种方式会干扰残差路径中的恒等映射，导致深层网络训练的不稳定。层前归一化（Pre-layer Normalization）将层内归一化放置在多头注意力或前馈网络之前，从而缓解了上述问题。层前归一化还能使输入多头注意力和前馈网络的数据分布更加稳定，有利于这些网络层参数的优化。因此，此处采用目前更为广泛采纳的层前归一化。

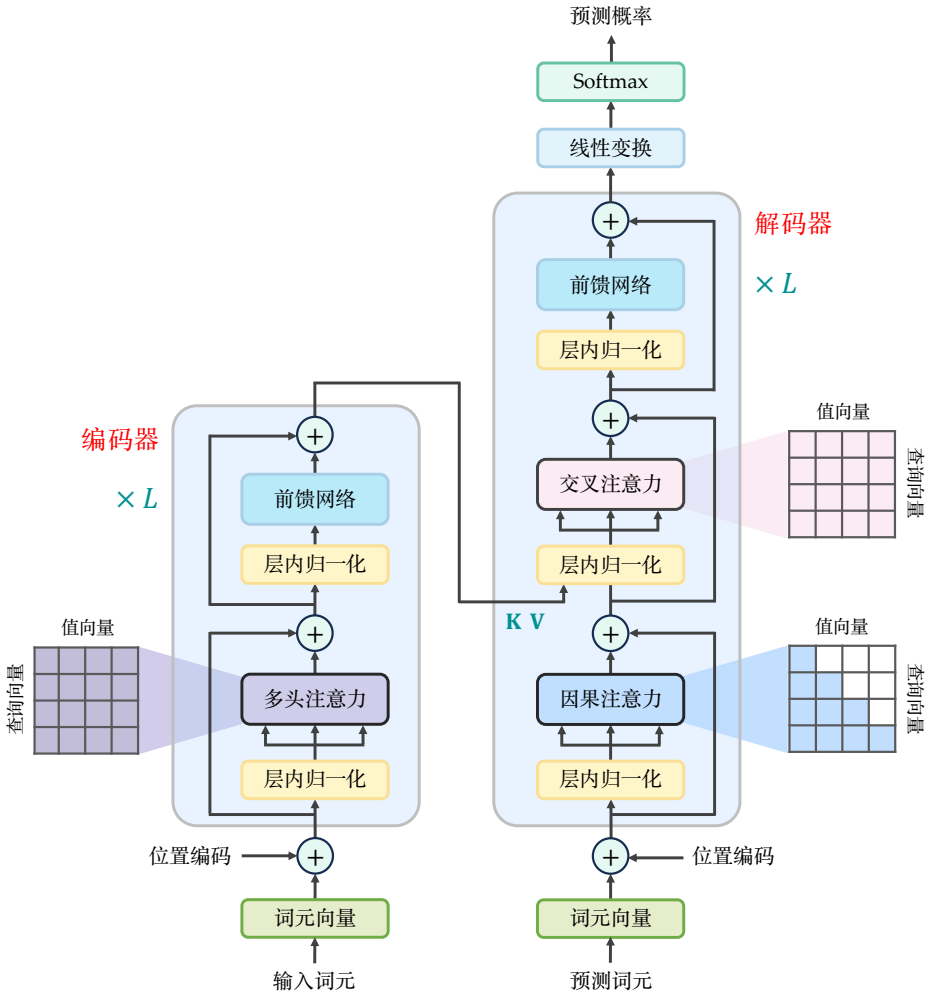


图 7.26: Transformer 架构示意图。Transformer 由编码器和解码器两部分组成，两部分均由多个结构相同的模块堆叠而成。编码器的每个模块由多头注意力子层和前馈网络子层组成，解码器的每个模块则由因果注意力子层、交叉注意力子层以及前馈网络子层构成。

信息。位置编码向量的维度与词向量相同，其各维分量通过如下方式计算得到：

$$\mathrm{PE}(j, 2i) = \sin\left(\frac{j}{10000^{\frac{2i}{M}}}\right) \tag{7.142}$$

$$\mathrm{PE}(j, 2i+1) = \cos\left(\frac{j}{10000^{\frac{2i}{M}}}\right) \tag{7.143}$$

其中 j 表示序列中的位置索引，i 表示位置编码的维度（$2i$ 与 $2i+1$ 分别对应偶数维与奇数维），而 M 表示词向量的维度。位置编码的每个特征维度都对应一个特定频率的正余弦信号，其波长覆盖从 2π 到 $10000 \cdot 2\pi$ 的等比数列。

上述位置编码会根据元素在序列中的绝对位置分配不同的编码。例如，为序列中位于第一的元素分配位置编码 $\boldsymbol{p}_1$（按公式7.142和7.143计算），为第二个元素分配 $\boldsymbol{p}_2$，以此类推。由于这种编码方式完全依赖于元素在整个序列中的绝对位置，因此被称为**绝对位置编码**（Absolute

Positional Encoding）。这类绝对位置编码存在长度外推性较差和缺乏平移不变性的问题。长度外推性较差是指：假设模型在训练阶段所见到的最大序列长度为 T（例如 $T = 2^{10} = 1024$），则在推理阶段，当输入序列长度超过 1024 时，模型性能往往会明显下降。换言之，模型难以将其在长度范围 $[1, 1024]$ 内学习到的表示与建模能力有效泛化到更长的序列上。在实际训练中，样本长度不可能无限延伸，否则不仅会降低训练效率，还会显著提高硬件资源需求（注意力机制的计算与存储复杂度随序列长度呈平方增长）。此外，绝对位置编码另一个重要局限是其缺乏平移不变性。仍以自然语言处理为例，一个固定的短语（如“机器学习”）既可能出现在句首，也可能出现在句中或句尾。从语义理解的角度来看，无论该短语处于句中何种位置，模型都应能一致地捕捉“机器”与“学习”之间的相关性与依赖关系。然而，在绝对位置编码方式下，同一短语在不同位置会叠加不同的位置编码，从而改变注意力得分的计算结果，进而破坏了平移不变性。

引入位置编码的主要目的是为了解决注意力机制的置换等变性问题。理想的位置编码方式应能够在注意力得分计算过程中，为模型提供序列中不同位置之间相对距离或位置信息。具体而言，设两个元素分别位于第 m 个和第 n 个位置，则在注意力（点积）计算后，模型应能够在尽量不干扰内容信息的前提下，保留与相对位置差 $m - n$ 相关的信息。这一性质也有助于模型处理长度超过训练阶段最大序列长度 T 的输入。即使推理时某些元素的位置超出了 T，只要它们之间的相对距离仍处于训练阶段覆盖的范围内，相应的位置关系仍可按照相同的方式进行表示，从而为模型向更长序列的泛化提供有利条件。此外，由于相对位置编码关注元素之间的相对距离，因此同一局部模式即使出现在序列的不同位置，其内部元素之间的相对位置关系仍然保持不变，有助于模型识别出现在不同位置的相似模式。以自然语言处理为例，无论某个短语（如“机器学习”）出现在句首、句中还是句尾，“机器”与“学习”之间的相对距离始终一致，因此模型在计算二者之间的注意力得分时基本也不会发生变化。按照上述思想设计的位置编码方式被称为**相对位置编码**（Relative Positional Encoding）。其中，具有代表性的方法之一是旋转位置编码 RoPE（Rotary Position Embedding）[74]。与绝对位置编码通常直接叠加到输入向量上的方式不同，相对位置编码通常直接作用于注意力计算过程。

为了更容易理解旋转位置编码 RoPE 的原理，我们先从二维查询向量与键向量的情形出发（注意力得分的计算仅与查询向量和键向量有关，而与值向量无关），再进一步推广到任意维度的情况。假设存在一个位于序列第 m 个位置的查询向量 $\boldsymbol{q}$ 和一个位于第 n 个位置的键向量 $\boldsymbol{k}$，我们的目标是寻找一种变换 f，用于在向量中引入位置信息，使得变换后的查询向量与键向量在进行点积计算后，所得到的注意力分值 s 仅依赖于向量本身的内容信息以及二者之间的相对位置差 $m - n$，即：

$$f(\boldsymbol{q}, m)^\top f(\boldsymbol{k}, n) = s(\boldsymbol{q}, \boldsymbol{k}, m - n) \tag{7.144}$$

RoPE 的核心思想是：将变换 f 设计为对向量按其位置在复平面进行旋转的操作。经旋转的向量在进行点积时，各自对应的绝对旋转角度会相互抵消，点积结果仅保留相对位置差有关的信息。

具体而言，二维查询向量 $\boldsymbol{q} = [q_1, q_2]^\top$ 和键向量 $\boldsymbol{k} = [k_1, k_2]^\top$ 可以在复平面上分别表示为

如下复数形式：

$$z_q = q_1 + iq_2$$
$$z_k = k_1 + ik_2$$

其中 i 为虚数单位（$i^2 = -1$）。在复数域中，两个复数的内积通常通过共轭乘法来实现。复数的共轭（Conjugate）是指保持实部不变，并将虚部取反。因此，则 z_k 的共轭复数为：

$$\overline{z_k} = k_1 - ik_2$$

将 z_q 与 $\overline{z_k}$ 相乘，可得：

$$\begin{aligned} z_q\overline{z_k} &= (q_1 + iq_2)(k_1 - ik_2) \\ &= q_1k_1 + iq_2k_1 - iq_1k_2 - i^2q_2k_2 \\ &= \underbrace{(q_1k_1 + q_2k_2)}_{\text{实部}} + i\underbrace{(q_2k_1 - q_1k_2)}_{\text{虚部}} \quad (i^2 = -1) \end{aligned}$$

从以上的推导过程可以看出，两个二维向量在复平面表示下共轭乘积的实部与它们在原始实数空间中的点积结果一致，即：

$$\text{Re}[z_q\overline{z_k}] = \boldsymbol{q}^\top \boldsymbol{k} = q_1k_1 + q_2k_2$$

其中 Re[·] 表示取复数的实部。从几何角度来看，复数乘以 $e^{i\theta}$ 会在保持模长不变的情况下，使其在复平面上逆时针旋转 θ 弧度。同时，若采用极坐标形式（模长与辐角）表示复数，则两个复数相乘时，其结果对应于模长相乘、辐角相加；若其中一个复数取共轭，则对应于模长相乘、辐角相减。基于这一特性，RoPE 将位置变换函数 f 定义为复平面上的旋转操作：

$$f(\boldsymbol{q}, m) = z_q e^{im\theta}$$
$$f(\boldsymbol{k}, n) = z_k e^{in\theta}$$

其中位置索引 m 与 n 分别决定查询向量 $\boldsymbol{q}$ 与键向量 $\boldsymbol{k}$ 的旋转角度。注意力得分可表示为：

$$s(\boldsymbol{q}, \boldsymbol{k}, m-n) = \text{Re}\left[f(\boldsymbol{q}, m) \cdot \overline{f(\boldsymbol{k}, n)}\right] = \text{Re}\left[\left(z_q e^{im\theta}\right)\left(\overline{z_k} e^{-in\theta}\right)\right] = \text{Re}\left[z_q\overline{z_k} e^{i(m-n)\theta}\right] \tag{7.145}$$

从上式可以看出，虽然变换 f 将绝对位置 m 和 n 用于复平面旋转角的计算，但经过共轭相乘并提取实部后，最终的注意力得分仅取决于原始向量的内容信息（即 $\text{Re}[z_q\overline{z_k}] = \boldsymbol{q}^\top \boldsymbol{k}$）以及二者之间相对位置差 $m - n$ 相关的项。

上述复平面中的旋转操作，可以等价地转换为对查询向量或键向量施加一个二维旋转矩阵。根据欧拉公式（Euler's Formula），我们有：

$$e^{im\theta} = \cos m\theta + i \sin m\theta \tag{7.146}$$

将二维查询向量 $\boldsymbol{q}$ 的复数表示 $z_q = q_1 + iq_2$ 乘以旋转算子 $e^{im\theta}$，可得：

$$\begin{aligned} z_q e^{im\theta} &= (q_1 + iq_2)e^{im\theta} \\ &= (q_1 + iq_2)(\cos m\theta + i\sin m\theta) \\ &= \underbrace{(q_1\cos m\theta - q_2\sin m\theta)}_{\text{实部}} + i\underbrace{(q_1\sin m\theta + q_2\cos m\theta)}_{\text{虚部}} \end{aligned}$$

若将上述结果的实部记为 $\tilde{q}_1$，虚部记为 $\tilde{q}_2$，则有：

$$\begin{bmatrix} \tilde{q}_1 \\ \tilde{q}_2 \end{bmatrix} = \begin{bmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \tilde{\boldsymbol{q}} \tag{7.147}$$

同理，将二维键向量 $\boldsymbol{k}$ 对应的复数表示乘以旋转算子 $e^{in\theta}$，亦可写为等价的实数矩阵乘法：

$$\begin{bmatrix} \tilde{k}_1 \\ \tilde{k}_2 \end{bmatrix} = \begin{bmatrix} \cos n\theta & -\sin n\theta \\ \sin n\theta & \cos n\theta \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \tilde{\boldsymbol{k}} \tag{7.148}$$

将转换后的向量 $\tilde{\boldsymbol{q}}$ 与 $\tilde{\boldsymbol{k}}$ 进行点积运算 $\tilde{\boldsymbol{q}}^\top\tilde{\boldsymbol{k}}$，即实部 $\tilde{q}_1$ 与 $\tilde{k}_1$ 乘积，以及虚部 $\tilde{q}_2$ 与 $\tilde{k}_2$ 乘积（两个复数的虚部乘积为实数）之和（即去掉了一个复数的实部与另一个复数的虚部相乘所构成的虚数项），其结果等价于复数 $z_q e^{im\theta}$ 与 $z_k e^{in\theta}$ 的共轭 $\overline{z_k}e^{-in\theta}$ 相乘后的实部：

$$\text{Re}\left[z_q\overline{z_k}e^{i(m-n)\theta}\right] = \underbrace{(q_1k_1 + q_2k_2)\cos((m-n)\theta)}_{\text{原向量点积结果经三角函数缩放后的值}} + \underbrace{(q_1k_2 - q_2k_1)\sin((m-n)\theta)}_{\text{与相对位置差 } m-n \text{ 相关的项}} \tag{7.149}$$

从上式可以看出，经过旋转矩阵变换后的查询向量与键向量的点积结果既保留了原向量之间的内容相关性，还显式引入了与相对位置差 $m-n$ 有关的信息。

将上述二维情形推广到 D 维空间时，可以将整个 D 维向量划分为 $D/2$ 个二维子空间，并对每一个二维子空间分别施加类似的旋转变换。对应的 D 维旋转矩阵可表示为（不同背景颜色为一对二维子空间）：

$$\begin{bmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{D/2} & -\sin m\theta_{D/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{D/2} & \cos m\theta_{D/2} \end{bmatrix} \tag{7.150}$$

其中 $\theta_i = 10000^{-2(i-1)/D}$, $i \in \{1, \ldots, D/2\}$ 是预定义的频率基数集合，用于控制各二维子空间对应的旋转频率。此外，RoPE 整体上具有远程衰减（Long-term Decay）的特性，即随着相对位置差的绝对值 $|m-n|$ 增大，不同位置之间的注意力点积在整体上呈现减弱的趋势。其原因在于，RoPE 采用不同频率的旋转变换，使得相距较远的位置之间更容易出现不同频率分量相互抵消的现象，从而使点积幅度的上界在整体上呈现下降趋势。需要注意的是，这种衰减并非严格单调的（表现为局部振荡、总体下降的趋势），也不意味着任意两个位置之间的注意力得分都会

随着距离增加而减小。这种“位置距离越远，相关性越弱”的性质，直观上与自然语言以及各类时序信号的实际规律较为吻合，因此较适合作为位置编码。

如图7.26所示，Transformer 解码器顶部通常通过 Softmax 层预测生成序列中的下一个单词，即在整个词汇表中进行多分类。以英语为例，仅《牛津英语词典》就收录超过 60 万词条（包括历史词、废弃词和各种变体等），而现代英语中常用词汇的规模通常也达到 17 万至 25 万之间。词汇表规模越大，分类任务的难度通常越高，因为模型需要在更多候选词之间进行区分与选择。此外，在 Transformer 编码器和解码器的输入端，都需要将离散的单词映射为连续的向量表示。这种向量表示通常称为**词向量**（Word Embedding）或词嵌入（即将离散词汇嵌入到连续的语义空间中）。假设词向量维度为 D，词汇表中包含的词数为 K，则词向量矩阵所对应的参数量为 $K \times D$。减小词汇表规模不仅能够降低模型分类预测的难度，还可以有效减少模型的参数量。

中文可以采用“字”作为基本单位来降低词表规模 [78]，英文也可以将较长单词拆分为多个子词（Subword）。例如，单词“beautiful”可以拆分为“beau”、“ti”和“ful”三个子词（Sub-word）。这些子词类似于英语中的词根、前缀或后缀，能够在多个单词之间共享和复用，因此也有助于减小词汇表规模。后续研究表明，在经过大规模预训练（将在后文详细介绍）后，子词切分粒度和拆分方式对模型的语义理解与语言生成能力影响相对有限。因而转向基于给定语料（即包含多国语言的大规模文本）的词表自动构建方法。其中，代表性方法之一是字符对编码（Byte Pair Encoding, BPE）[69]。该方法首先将语料视为字符（或字节）构成的序列，然后统计整个语料中相邻字符对出现的频率，并将出现频率最高的字符对合并为一个新的词元（Token）。随后继续在更新后的序列上重复这一过程，直到达到预设的词表大小（更准确地说，是达到预设的词元数量）。在这一过程中，先前已经合并得到的词元还可以继续与其他字符或词元进一步合并，从而形成更长的词元。这种基于词元的词表构建方式不仅能够有效控制词表规模，还对训练过程中并未出现的未见词（Out-of-vocabulary）具有较好的泛化能力，因为大多数未见词通常可以拆分为若干已知词元的组合。采用词元级词表后，Transformer 编码器和解码器的输入都以词元作为基本单位，而解码器顶层分类类别数量也与词表中的词元总数一致。

Transformer 架构本身具有庞大的参数量。在提出该架构的经典工作 [77] 中，作者分别给出了基础版（Base）和加强版（Big）两种不同规模的模型配置。其中，较小规模的基础版模型的编码器与解码器均由 6 个模块堆叠而成，注意力头数为 8（每个注意力头的维度为 64），隐藏层维度为 $512 = 8 \times 64$，前馈网络子层的中间层维度为 2048，总参数量约为 $6,500$ 万。较大规模的加强版模型的编码器与解码器仍由 6 个模块堆叠而成，注意力头数增加到 16（每个注意力头的维度仍为 64），隐藏层维度提升至 $1024 = 16 \times 64$，前馈网络子层的中间层维度扩展到 $4,096$，总参数量达到了 2.13 亿。随着大语言模型时代的到来，作为典型代表的 GPT-3 将这一架构的解码器扩展到了 96 个堆叠模块，注意力头数达到 96（每个注意力头的维度为 128），隐藏层维度骤升至 $12,288 = 96 \times 128$，其前馈网络子层的中间层维度高达 $49,152$，使得总参数规模达到 $1,750$ 亿。面对如此海量的网络参数，传统的随机初始化方法已难以为继。正如前文所述，最佳的参数初始化方法是预训练。只有通过预训练让模型在海量文本中学习通用的语言统计规律与语义表示，才能为下游任务的微调（Fine-tuning）提供具有良好泛化能力和鲁棒性的初始参数。

T5（Text-to-text Transfer Transformer）[61] 提出了一种基于“文本到文本”或“序列到序列”的预训练方法。其核心想法是：首先在输入文本中随机删除若干连续的片段，并使用特殊标记替换这些被删除的片段；然后基于 Transformer 的编码器对残缺文本进行编码，再由解码器生成被删除的片段。例如，对于如下原始文本：

```
The cute cat walks in the campus .
```

随机删除若干连续片段之后，可以得到：

```
The cute [X1] in the [X2] .
```

其中 `[X1]` 与 `[X2]` 是用于表示缺失片段的特殊标记。模型的训练目标是让解码器按如下方式生成并恢复所有被删除的片段：

```
[X1] cat walks [X2] campus
```

在预训练过程中，Transformer 的编码器、解码器以及词向量等所有参数均通过该目标函数进行端到端联合训练，并采用随机梯度下降法进行优化。

与 T5 仅重构缺失片段不同，BART（Bidirectional and Auto-regressive Transformer）[48] 将预训练目标进一步扩展为“去噪自回归”：即对原始文本施加多种形式的破坏或噪声扰动，然后训练模型从受损的输入中生成完整的原始文本。常见的破坏或加噪方式包括词汇级或片段级随机遮蔽、删除、替换，以及随机打乱输入顺序等。T5 和 BART 均属于典型的基于“编码器-解码器”（Encoder-decoder）结构的预训练方法，通过统一的序列到序列生成目标对编码器和解码器进行联合优化。

在 T5 和 BART 出现之前，BERT（Bidirectional Encoder Representations from Transformers）提出了掩码语言模型（Masked Language Model），并仅对 Transformer 的编码器部分进行预训练 [21]。**掩码语言模型**的基本思想是：首先对原始文本中的若干词进行随机遮蔽，然后利用 Transformer 编码器提取其上下文特征，并在编码器顶部附加线性映射与 Softmax 分类层（如图7.27(a) 所示），以预测被遮蔽的词汇。例如，将用于介绍 T5 训练目标的原始文本进行随机遮蔽后，可以得到：

```
The cute [MASK] [MASK] in the [MASK] .
```

其中特殊标记 `[MASK]` 表示对应位置的词被遮蔽。模型需要结合上下文信息，通过自注意力机制生成该位置的上下文特征表示，并据此预测被遮蔽的词。尽管 BERT 的训练目标包含对缺失词的预测，但由于其并不包含自回归式的解码器部分，因此主要能力在于对输入文本进行特征提取与语义编码，更适用于如文本分类和序列标注等文本理解任务，通常并不能直接用于文本生成任务。在编码过程中，由于整个输入序列是完整可见的，编码器的自注意力机制允许每个位置同时利用其前后位置的信息，从而使 BERT 具备双向上下文建模与表征能力（即同时兼顾上文与下文信息）。

与 BERT 采用 Transformer 编码器的架构不同，GPT（Generative Pre-trained Transformer）仅采用 Transformer 的解码器部分，并去除了用于连接编码器与解码器的交叉注意力层 [60]。其预训练目标遵循“下一词元预测”（Next-token Prediction）范式，即在给定上文的条件下，预测文本序列中的下一个词元。尽管 GPT 仅使用 Transformer 解码器架构（如图7.27(b) 所示），但它不仅能够用于文本生成，也可以用于文本编码和语言理解。在文本生成过程中，GPT 采用自回归

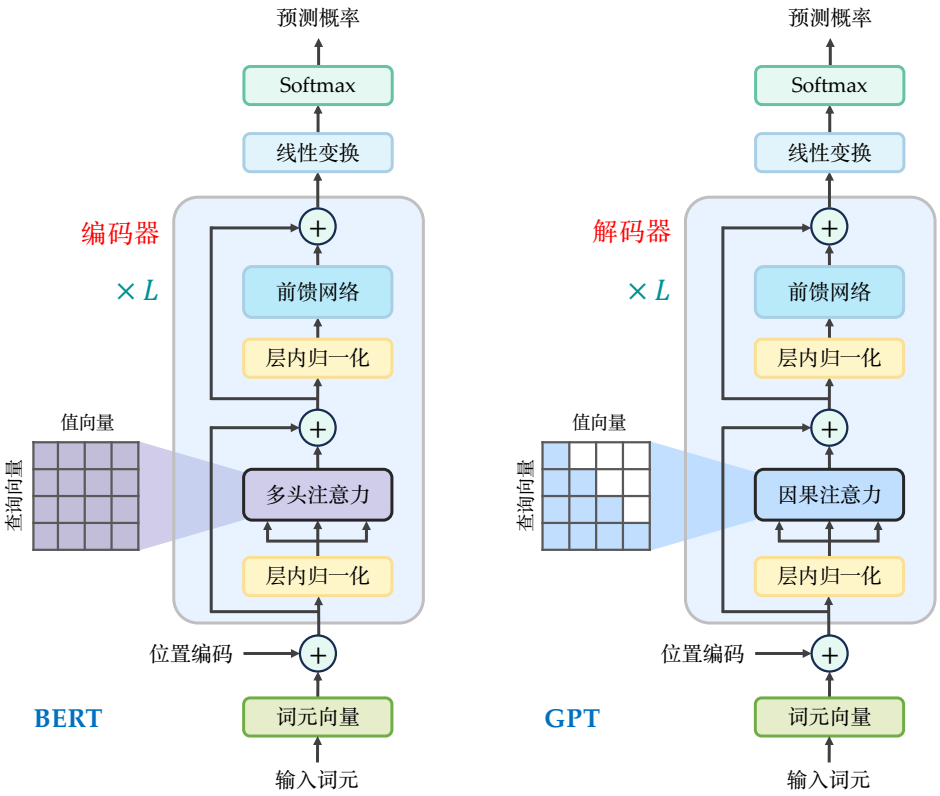


图 7.27: BERT 与 GPT 的架构及注意力机制对比。(a)BERT 仅采用 Transformer 的编码器结构，其注意力机制为双向注意力，可同时利用上下文信息。(b)GPT 仅采用 Transformer 解码器结构，其注意力机制为单向的因果注意力，只能关注当前位置及之前的输入信息。

生成策略：模型首先根据已有上文预测下一个词元，然后将该词元追加到已有上文末尾；随后再基于扩展后的上文继续预测下一个词元，如此循环迭代，直到生成表示结束的特殊标记为止。在用于文本编码时，通常将解码器顶层输出的最后一个词元的隐表示作为整个输入文本的语义表示。

GPT 所采用的“给上文、补下文”的下一词元预测的预训练范式，与许多自然语言处理任务的形式高度一致，因此比较适合在大规模预训练的基础上，针对各类下游任务进行微调。例如，机器翻译可视为给定源语言的上文、生成目标语言下文的过程；问题回答可看成给定提问内容的上文、生成标准回答下文的过程；多轮对话可视为给定交互历史的上文、生成适合回复下文的过程。即使是传统的文本分类任务，也可以通过补充提示（Prompt）的方式转化为生成任务。例如，在需要分类的文本之后补充“根据以上对电影的评价，在好评、中评、差评中做出判断”的提示，便能引导 GPT 模型生成“好评”、“中评”或“差评”之一作为分类结果。研究表明，在经过大规模预训练之后，GPT 这类仅基于解码器的模型在多种语言任务上的性能可以达到甚至超过同时包含编码器与解码器的模型 [2]。此外，单向因果注意力在自回归推理时能够通过键值

主要符号表

数学符号

$\boldsymbol{x}$	向量
y	标量
D	向量维度
x_j	向量的第 j 个分量（向量分量为标量）
f	输入到输出的某种映射
$\mathbf{X}^\top$	矩阵转置
$\mathbf{X}^{-1}$	逆矩阵
$\|\mathbf{X}\|$	矩阵的行列式
$\mathrm{tr}(\mathbf{X})$	矩阵的迹（矩阵所有特征值之和，也等于主对角线元素的总和）
$\ln$	以自然常数 e 为底的对数
e^x 或 $\exp(x)$	自然常数 e 的指数函数
$f'(x)$	一阶导数
$f''(x)$	二阶导数
$\nabla f(\boldsymbol{x})$	一阶梯度（梯度向量）
$\nabla^2 f(\boldsymbol{x})$	二阶梯度（海森矩阵）
$\frac{\partial f(x)}{\partial x}$	一阶偏导
$\frac{\partial^2 f(x)}{\partial x^2}$	二阶偏导

机器学习

$\mathcal{D}$	数据集
N	样本数量
$\boldsymbol{x}_i$	第 i 个训练样本
y_i	第 i 个训练样本对应的输出
$\kappa(\boldsymbol{x}, \boldsymbol{x}')$	核函数
$\mathcal{L}(\cdot)$	损失函数
$\mathcal{J}(\cdot)$	目标函数或代价函数

概率统计

$\mathbb{E}[x]$	均值或期望
$\text{Var}[x]$	方差
$\text{Cov}[x, z]$	随机变量 x 和 z 之间的协方差
$\Gamma(z)$	伽玛函数
$\mathcal{N}(x\|\mu, \sigma^2)$	高斯分布，其中 μ 为均值，σ 为标准差
$\mathcal{N}(x\|\mu, \lambda^{-1})$	高斯分布，其中 μ 为均值，λ 为精度
$\text{Bern}(x\|\mu)$	伯努利分布
$\text{Bin}(k\|\mu, N)$	二项分布
$\text{Beta}(\mu\|\alpha, \beta)$	贝塔分布
$\text{Poisson}(k\|\lambda)$	泊松分布
$\text{Exp}(\lambda\|\beta)$	指数分布
$\text{Gamma}(\lambda\|\alpha, \beta)$	伽玛分布，其中 α 为形状参数，β 为逆尺度参数
$\text{Mult}(\boldsymbol{m}\|\boldsymbol{\mu}, N)$	多项分布
$\text{Dir}(\boldsymbol{\mu}\|\boldsymbol{\alpha})$	狄利克雷分布
$\mathcal{N}(\boldsymbol{x}\|\boldsymbol{\mu}, \boldsymbol{\Sigma})$	多维高斯分布，其中 $\boldsymbol{\mu}$ 为均值向量，$\boldsymbol{\Sigma}$ 为协方差矩阵
$\mathcal{N}(\boldsymbol{x}\|\boldsymbol{\mu}, \boldsymbol{\Lambda})$	多维高斯分布，其中 $\boldsymbol{\mu}$ 为均值向量，$\boldsymbol{\Lambda}$ 为精度矩阵
$\mathcal{W}(\boldsymbol{\Lambda}\|\mathbf{W}, \eta)$	威沙特分布，其中 $\mathbf{W}$ 为尺度矩阵，η 为自由度
$\text{NormalGamma}(\mu, \lambda\|\upsilon, \nu, \alpha, \beta)$	高斯-伽玛分布
$\text{NormalWishart}(\boldsymbol{\mu}, \boldsymbol{\Lambda}\|\boldsymbol{\upsilon}, \nu, \mathbf{W}, \eta)$	高斯-威沙特分布
$\text{St}(x\|\mu, \tau, \eta)$	学生氏分布，其中 μ 为均值，τ 为精度，η 为自由度
$\text{St}(\boldsymbol{x}\|\boldsymbol{\mu}, \boldsymbol{\Lambda}, \eta)$	多维学生氏分布，其中 $\boldsymbol{\mu}$ 为均值向量，$\boldsymbol{\Lambda}$ 为精度矩阵
$\text{Laplace}(x\|\mu, \tau)$	拉普拉斯分布，其中 μ 为位置参数，τ 为尺度参数
$\chi^2(x\|\eta)$	卡方分布，其中 η 为自由度
$\text{U}(x\|a, b)$	连续型均匀分布，其中 a 为下界、b 为上界 $(a < b)$
$\text{H}[x]$	信息熵，其中 x 为随机变量
$\text{H}[y\|x]$	条件熵，其中 x 和 y 为随机变量
$\text{H}[p, q]$	分布 p 和 q 之间的交叉熵
$D_{\text{KL}}(p\|\|q)$	分布 p 和 q 之间的 KL 散度

参考文献

[1] Daniel J. Amit, Hanoch Gutfreund, and Haim Sompolinsky. “Storing infinite numbers of patterns in a spin-glass model of neural networks”. In: *Physical Review Letters* 55.14 (1985), pp. 1530–1533.

[2] Mikel Artetxe et al. “On the role of bidirectionality in language model pre-training”. In: *Findings of the Conference on Empirical Methods in Natural Language Processing* (2022), pp. 3973–3985.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv: 1607.06450* (2016).

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[5] Jame O. Berger. *Statistical decision theory and Bayesian analysis (second edition)*. Springer, 1980.

[6] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research* 13.2 (2012).

[7] Jeremy Bernstein and Laker Newhouse. “Old optimizer, new norm: An anthology”. In: *Proceedings of the 16th Annual Workshop on Optimization for Machine Learning* (2024), pp. 1–19.

[8] J Besag. “On spatio-temporal models and Markov fields”. In: *Transactions of the 7th Prague Conference on Information Theory, Statistical Decision Functions and Random Processes* (1974), pp. 47–75.

[9] Julian Besag. “On the statistical analysis of dirty pictures”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 48.3 (1986), pp. 259–279.

[10] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[11] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. 1992, pp. 144–152.

[12] George E.P. Box and Norman Richard Draper. *Empirical model-building and response surfaces*. John Wiley & Sons, 1987.

[13] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[14] Tom Brown et al. “Language models are few-shot learners”. In: *Proceedings of the Conference on Neural Information Processing Systems* 33 (2020), pp. 1877–1901.

[15] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder–decoder for statistical machine translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (2014), pp. 1724–1734.

[16] Peter Clifford. “Markov random fields in statistics”. In: *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley* (1990), pp. 19–32.

[17] Thomas H. Cormen et al. *Introduction to algorithms (third edition)*. The MIT Press, 2022.

[18] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley, 1991.

[19] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[20] Morris H. DeGroot and Mark J. Schevish. *Probability and statistics (forth edition)*. China Machine Press, 2012.

[21] Jacob Devlin et al. “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2019), pp. 4171–4186.

[22] Alexey Dosovitskiy et al. “An image is worth 16×16 words: Transformers for image recognition at scale”. In: *Proceedings of the International Conference on Learning Representations* (2021).

[23] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.7 (2011).

[24] Walter D. Fisher. “On grouping for maximum homogeneity”. In: *Journal of the American Statistical Association* 53.284 (1958), pp. 789–798.

[25] Roger Fletcher. *Practical methods of optimization (second edition)*. Wiley, 1987.

[26] Yoav Freund and Robert E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139.

[27] Stuart Geman and Donald Geman. “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1984), pp. 721–741.

[28] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. “Learning precise timing with LSTM recurrent networks”. In: *Journal of Machine Learning Research* 3 (2002), pp. 115–143.

[29] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), pp. 249–256.

[30] Ian J Goodfellow et al. “Generative adversarial nets”. In: *Proceedings of the Conference on Neural Information Processing Systems* 27 (2014).

[31] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction (second edition)*. Springer, 2009.

[32] Kaiming He and Jian Sun. “Convolutional neural networks at constrained time cost”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 5353–5360.

[33] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[34] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1026–1034.

[35] Nicholas J. Higham. *Functions of matrices: Theory and computation*. SIAM, 2008.

[36] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv: 1503.02531* (2015).

[37] Geoffrey E. Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural Computation* 14.8 (2002), pp. 1771–1800.

[38] John J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558.

[39] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the International Conference on Machine Learning* (2015), pp. 448–456.

[40] Arieh Iserles. *A first course in the numerical analysis of differential equations*. 44. Cambridge university press, 2009.

[41] Max Jaderberg et al. “Population based training of neural networks”. In: *arXiv: 1711.09846* (2017).

[42] Kevin Jamieson and Ameet Talwalkar. “Non-stochastic best arm identification and hyperparameter optimization”. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics* (2016), pp. 240–248.

[43] Keller Jordan et al. *Muon: An optimizer for hidden layers in neural networks*. 2024. URL: https://kellerjordan.github.io/posts/muon/.

[44] Diederik P. Kingma and Jimmy L. Ba. “Adam: A method for stochastic optimization”. In: *Proceedings of the International Conference on Learning Representations* (2015).

[45] Josef Kittler and Janos Föglein. “Contextual classification of multispectral pixel data”. In: *Image and vision computing* 2.1 (1984), pp. 13–29.

[46] Solomon Kullback and Richard A. Leibler. “On information and sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86.

[47] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (2002), pp. 2278–2324.

[48] Mike Lewis et al. “BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), pp. 7871–7880.

[49] Yaron Lipman et al. “Flow matching guide and code”. In: *arXiv: 2412.06264* (2024).

[50] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *arXiv: 1711.05101* (2017).

[51] Calvin Luo. “Understanding diffusion models: A unified perspective”. In: *arXiv: 2208.11970* (2022).

[52] Marvin Minsky and Seymour Papert. *Perceptrons: An introduction to computational geometry*. The MIT Press, 1969.

[53] Kevin P. Murphy. *Machine learning: A probabilistic perspective*. The MIT Press, 2012.

[54] Brady Neal et al. “A modern take on the bias-variance tradeoff in neural networks”. In: *arXiv: 1810.08591* (2018).

[55] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Proceedings of the Conference on Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[56] Judea Pearl. *Probabilistic reasoning in intelligent systems*. Morgan kaufmann, 2014.

[57] Matthew E. Peters et al. “Deep Contextualized Word Representations”. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2018), pp. 2227–2237.

[58] John C. Platt. “Fast training of support vector machines using sequential minimal optimization”. In: *Advances in Kernel Methods* (1999), pp. 185–208.

[59] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *Proceedings of the International Conference on Learning Representations* (2016).

[60] Alec Radford et al. “Improving language understanding by generative pre-training”. In: *OpenAI’s Technical Report* (2018).

[61] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67.

[62] Scott Reed et al. “Generative adversarial text to image synthesis”. In: *Proceedings of the International Conference on Machine Learning* (2016), pp. 1060–1069.

[63] Edmund T. Rolls. *Cerebral cortex: principles of operation*. Oxford University Press, 2016.

[64] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention* (2015), pp. 234–241.

[65] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[66] Leonard J. Savage. *The subjective basis of statistical practice*. University of Michigan, 1961.

[67] Andrew M Saxe, James L McClelland, and Surya Ganguli. “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”. In: *Proceedings of the International Conference on Learning Representations* (2014).

[68] Robert E. Schapire. “The strength of weak learnability”. In: *Machine learning* 5.2 (1990), pp. 197–227.

[69] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural machine translation of rare words with subword units”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (2016), pp. 1715–1725.

[70] Claude Elwood Shannon. “A mathematical theory of communication”. In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423.

[71] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical Bayesian optimization of machine learning algorithms”. In: *Proceedings of the Conference on Neural Information Processing Systems* 25 (2012).

[72] Yang Song et al. “Score-based generative modeling through stochastic differential equations”. In: *Proceedings of the International Conference on Learning Representations* (2021).

[73] Gilbert Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press, 2016.

[74] Jianlin Su et al. “Roformer: Enhanced transformer with rotary position embedding”. In: *Neurocomputing* 568 (2024), p. 127063.

[75] Tijmen Tieleman and Geoffrey E. Hinton. “Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural Networks for Machine Learning* 4.2 (2012), pp. 26–26.

[76] Antonio Torralba, Phillip Isola, and William Freeman. *Foundations of computer vision*. The MIT Press, 2024.

[77] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the Conference on Neural Information Processing Systems* 30 (2017).

[78] Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. “Deep learning for Chinese word segmentation and POS tagging”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (2013), pp. 647–657.

[79] Ji Zhu et al. “Multi-class AdaBoost”. In: *Statistics and its Interface* 2.3 (2009), pp. 349–360.

[80] Hui Zou and Trevor Hastie. “Regularization and variable selection via the elastic net”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.2 (2005), pp. 301–320.