第 6 章 人工神经网络

6.1 神经元模型

“神经网络” 一词最早源于对生物系统信息处理机制的数学建模与模拟尝试。为了区别于生物学意义上的神经系统，这类模型通常被称为人工神经网络（Artificial Neural Network）。尽管早期研究深受神经科学发现的启发，但随着学科的发展，现代人工神经网络已逐渐脱离对生物合理性（Biological Plausibility）的追求，演变为机器学习领域中一类通用的模式识别模型。

在神经科学的发展史上，西班牙神经学家圣地亚哥 · 拉蒙-卡哈尔（Santiago Ramón y Cajal）发挥了关键作用。他改进了意大利生物学家卡米洛 · 高尔基（Camillo Golgi）发明的硝酸银染色法，使神经细胞在显微镜下能够清晰可见（典型神经元结构如图6.1(a) 所示[1]）。基于他的观察，拉蒙-卡哈尔提出了著名的神经元学说（Neuron Doctrine），指出神经元在解剖和功能上是独立的离散个体，而非当时主流观点所认为的连续网络。这一学说奠定了现代神经科学的理论基石。因高尔基与拉蒙-卡哈尔在神经系统结构研究上的卓越贡献，他们在 1906 年共同获得了诺贝尔生理学或医学奖。

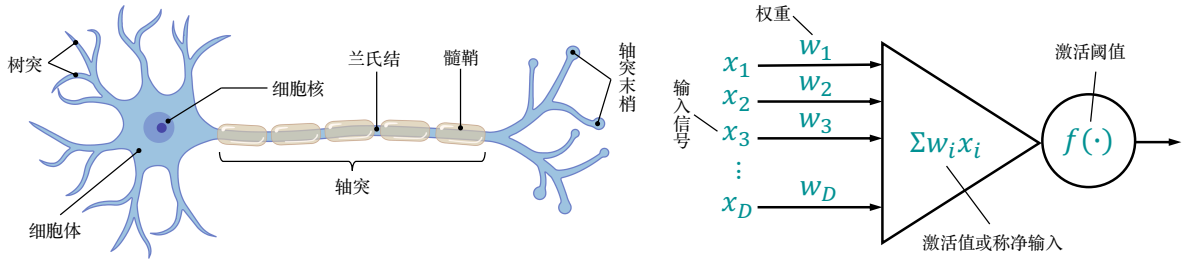
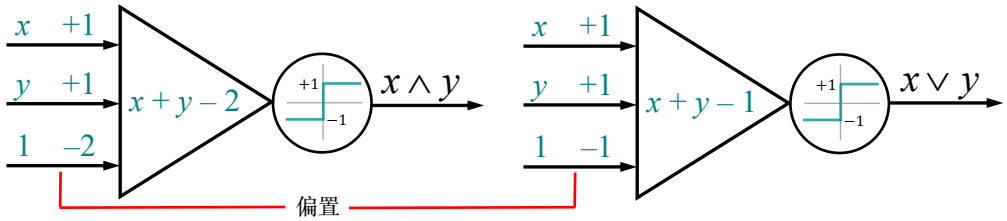


图 6.1: 生物神经元与人工神经元的结构对比示意。(a) 典型的生物神经元主要由细胞体和胞突（用于接收信号的树突与用于传导信号的轴突）组成。(b) 人工神经元模型是对生物神经元功能的抽象数学模拟。其中 x_i 表示来自前一层神经元的输入信号，而 w_i 为对应的权重，用于表征各输入信号对当前神经元的影响强度。所有输入经加权求和得到激活值（亦称净输入）$\sum_{i=1}^{D} w_i x_i$，该激活值与阈值进行比较（或通过非线性激活函数映射），最终产生输出信号。

如图6.1(a) 所示，典型的生物神经元主要由细胞体（Soma）、树突（Dendrites）和轴突（Axon）三部分组成。其中，树突负责接收来自其他神经元的传入信号，信号在细胞体内完成整合后，再通过轴突传导至下游神经元。神经元之间通过突触（Synapse）相互连接，构成了复杂的神经网络。突触具有可塑性（Synaptic Plasticity），即神经元之间的连接强度可随神经活动而动态改变。1925 年至 1926 年，英国电生理学家埃德加 · 阿德里安（Edgar Adrian）通过对神经纤维的电生理记录分析发现，神经元之间并非通过连续模拟信号来传递信息，而是依靠全或无的动作电位（脉冲）序列来进行通信。这意味着，只有当刺激达到特定阈值时，神经元才会发放标准强度的

[1]在全脑尺度上，若综合形态、分子和功能标准进行分类，神经元的类型可能多达上千种。

动作电位；若刺激不足，神经元则保持静息状态，不发放脉冲。尽管单个动作电位具有恒定的强度，但其承载的信息主要编码在脉冲发放的时间模式与频率中。通常而言，刺激越强，单位时间内产生的动作电位的频率就越高。阿德里安因揭示神经系统的信息传导机制，于 1932 年获得诺贝尔生理医学奖。这种“全或无”的特性启发了人工神经网络中阈值函数的设计，而“频率编码”机制，即刺激强度被转化为脉冲发放频率的非线性映射，则为人工神经网络引入非线性激活函数来表征神经元激活状态提供了生物学依据。



x	y	$x+y-2$	输出
1	1	0	+1
1	0	−1	−1
0	1	−1	−1
0	0	−2	−1

(a)

x	y	$x+y-1$	输出
1	1	1	+1
1	0	0	+1
0	1	0	+1
0	0	−1	−1

(b)

图 6.2: McCulloch-Pitts 神经元逻辑门实现示例。(a) 实现逻辑“与”（AND）运算的神经元配置。(b) 实现逻辑“或”（OR）运算的神经元配置。

既然神经系统由离散的神经元组成，每个神经元便可视为一个独立的逻辑处理单元。在这种思想的指引下，美国神经生理学家沃伦 · 麦卡洛克（Warren Sturgis McCulloch）和美国逻辑学家沃尔特 · 哈里 · 皮茨（Walter Harry Pitts）于 1943 年提出了首个神经元的数学模型，即著名的 McCulloch-Pitts 模型（简称 M-P 模型）。如图6.1(b) 所示，M-P 神经元接收来自 D 个输入信号 $x_1, \ldots, x_D$，并通过一组权重 $\{w_1, \ldots, w_D\}$ 对输入信号进行整合，整合方式为加权求和：

$$z = \sum_{i=1}^{D} w_i x_i \tag{6.1}$$

其中，权重取值 w_i 可正可负，类比于生物神经元突触的兴奋或抑制机制。正权重代表兴奋性（Excitatory）信号，趋向于激活神经元，而负权重则代表抑制性（Inhibitory）信号，趋向于抑制神经元的发放。在获得整合后的激活值 z 之后，受生物神经元“全或无”特性的启发，模型采用如下阈值函数作为激活函数（其中 t 为阈值）：

$$\sigma(z) = \begin{cases} +1 & \text{若 } z \geq t \\ -1 & \text{若 } z < t \end{cases} \tag{6.2}$$

有趣的是，麦卡洛克和皮茨提出神经元数学模型时，他们的灵感并非直接来源于突触的生化机制，而是来自数理逻辑。麦卡洛克当时深受莱布尼茨（Leibniz）的形式逻辑思想和图灵（Turing）

的计算理论影响。他意识到，如果生物神经元遵循“全或无”放电规律，那么通过神经元的互联，就可以在大脑中实现逻辑运算。如图6.2所示，通过配置适当的连接权重与激活阈值，单个 M-P 神经元即可实现逻辑“与”和“或”等基础运算。图6.2中所展示的神经元是 M-P 神经元的另一种等价表示，除了接收两个变量 x 和 y 的输入外，还接收一个数值恒定为 1 的输入项，其对应的权重称为偏置（Bias）。在此表示中，偏置等于原阈值的相反数，同时将激活函数的阈值统一设为零。这种表示形式使偏置能够像其他权重相同的方式进行调节。该偏置与线性模型中的偏置项作用类似。从这一角度看，每个 M-P 神经元本质上是一个带激活函数的广义线性模型。

尽管 M-P 神经元通过适当的配置可以实现基本逻辑门功能，但一旦设置完成，其所有参数保持固定，模型本身并不具备学习能力。单个人工神经元自适应学习能力的实现，要等到 14 年后由美国神经科学家弗兰克 · 罗森布拉特（Frank Rosenblatt）于 1957 年提出的感知机（Perceptron）模型及其学习算法。感知机又常被称为感知器。

6.2 感知机

在首个人工神经元模型提出之后，随之而来的核心问题是如何使神经元模型具备自适应学习能力。在此期间，神经科学关于学习过程中突触可塑性（Synaptic Plasticity）的机制也取得了重要进展。1949 年，加拿大心理学家唐纳德 · 奥丁 · 赫布（Donald Olding Hebb）提出赫布理论（Hebbian Theory），又称赫布规则（Hebb's Rule）。该理论的核心思想是：如果神经元 A 参与并导致了神经元 B 的重复持续激发，这两个神经元或其中一个将会发生某些生长过程或代谢变化，从而增强 A 激活 B 的能力。这一理论通常被总结为：“一起激发的神经元将更紧密地连在一起”（Cells that fire together, wire together）。基于赫布理论的突触权重更新规则可以形式化为：

$$\Delta w_{ij} = \eta z_i z_j \tag{6.3}$$

其中 w_{ij} 表示由突触前神经元 j 指向突触后神经元 i 的连接强度（即权重），$\eta > 0$ 为学习率，而 z_i 和 z_j 分别表示神经元 i 和 j 的激活值。赫布理论所定义的突触权重更新规则并不直接适用于有监督学习。这是因为该规则仅基于神经元之间放电的相关性进行权重调整，并未考虑和利用输入样本的期望输出或类别标签信息。

在提出之初，赫布理论更多地被视为一种富有洞见的假说。直到 20 世纪 60 年代至 70 年代，奥地利裔美国神经科学家埃里克 · 理查德 · 坎德尔（Eric Richard Kandel）通过对海兔（Aplysia）神经系统的系统性实验，从分子和细胞层面揭示了突触强度变化（突触可塑性）与学习和记忆过程之间的紧密关联，并为突触可塑性作为学习与记忆的重要神经生物学基础提供了实验证据。这一发现不仅使坎德尔获得 2000 年诺贝尔生理学或医学奖，也为人工神经网络研究确立了基本范式：所谓的“学习”，本质上就是网络中连接权重的动态调整过程。

罗森布拉特于 1957 年提出了感知机模型，这一时间点正好处于赫布规则提出之后、坎德尔揭示突触可塑性神经生物学机制之前。与上述神经科学理论相吻合，感知机通过调整单个神经元的连接权重来实现学习。该学习算法已在第3.5节中详细介绍，这里我们进行简要回顾。

感知机主要用于二分类任务。给定包含 N 个样本的训练数据集 $\mathcal{D}=\{(\boldsymbol{x}_i,t_i)\}_{i=1}^{N}$，并约定：若样本 $\boldsymbol{x}_i$ 属于类别 $\mathcal{C}_1$，则对应的标签 $t_i=+1$；若属于类别 $\mathcal{C}_2$，则 $t_i=-1$。设任意样本 $\boldsymbol{x}$ 有 D 维特征向量，即 $\boldsymbol{x}=(x_1,\ldots,x_D)^\top$，则单个感知机的输出可表示为：

$$y(\boldsymbol{x})=\sigma\left(\sum_{i=1}^{D}w_ix_i+b\right)=\sigma\left(\boldsymbol{w}^\top\boldsymbol{x}+b\right) \tag{6.4}$$

其中 $\sigma(\cdot)$ 为阈值为零的激活函数（即阶跃函数），而 b 为偏置项。为了符合人工神经网络的形式化表达习惯，此处将偏置项与其他权重参数分开表示，而不是将其并入扩展特征向量中。从上述表达式可以看出，感知机的结构与 M-P 神经元完全一致。二者的主要区别在于，感知机具备基于训练数据的参数学习算法，从而能够根据样本自动调整权重。

我们希望权重向量 $\boldsymbol{w}=(w_1,\ldots,w_D)^\top$ 和偏置 b 满足如下条件：当样本 $\boldsymbol{x}$ 属于类别 $\mathcal{C}_1$ 时，其激活值 $\boldsymbol{w}^\top\boldsymbol{x}+b>0$；而当 $\boldsymbol{x}$ 属于类别 $\mathcal{C}_2$ 时，则 $\boldsymbol{w}^\top\boldsymbol{x}+b<0$。将两种情况统一表示，可写为 $(\boldsymbol{w}^\top\boldsymbol{x}+b)t>0$，其中 $t\in\{+1,-1\}$ 为类别标签。对于某个具体样本 $\boldsymbol{x}_i$，若 $(\boldsymbol{w}^\top\boldsymbol{x}_i+b)t_i>0$，则该样本已被正确分类，无需调整参数；而当 $(\boldsymbol{w}^\top\boldsymbol{x}_i+b)t_i<0$ 时，说明该样本被误分类，需要对参数进行更新。罗森布拉特提出的更新思想十分直观：当样本被误分类时，应当调整参数，使 $(\boldsymbol{w}^\top\boldsymbol{x}_i+b)t_i$ 向增大的方向变化，即将其推向正值区域。使函数值增加最快的方向即为该函数的梯度方向。因此，权重向量和偏置的更新公式为：

$$\Delta\boldsymbol{w}=\eta\boldsymbol{x}_it_i \tag{6.5}$$

$$\Delta b=\eta t_i \tag{6.6}$$

其中 η 为学习率，用于控制每次参数更新的步长。

表 6.1: 二分类示例训练数据集

样本	1	2	3	4	5	6	7	8	9	10
x_1	1.0	9.4	2.5	8.0	0.5	7.9	7.0	2.8	1.2	7.8
x_2	1.0	6.4	2.1	7.7	2.2	8.4	7.0	0.8	3.0	6.1
t	+1	−1	+1	−1	+1	−1	−1	+1	+1	−1

接下来，我们利用表6.1所示的二分类训练数据集，简要说明感知机算法的训练过程。表中输入样本的特征向量是二维的，因此对应的感知机模型包含三个参数：权重 w_1，w_2 和偏置 b。假设参数随机初始化为：

$$w_1=0.75,\quad w_2=0.5,\quad b=-0.6$$

并将学习率设为 $\eta=0.2$。首先取第一个样本 $\boldsymbol{x}_1=(1.0,1.0)^\top$，感知机的输出为：

$$\sigma(0.75\times1+0.5\times1-0.6\times1)=\sigma(0.65)=1$$

预测结果与真实标签一致，因此无需进行参数更新。我们接着使用第二个样本 $\boldsymbol{x}_2=(9.4,6.4)^\top$，感知机的预测输出为：

$$\sigma(0.75\times9.4+0.5\times6.4-0.6\times1)=\sigma(9.65)=1$$

然而该样本的真实标签为 -1，因此该样本被错误分类，需要对参数进行调整。根据感知机更新公式6.5和6.6，分别对权重向量和偏置进行如下更新：

$$\boldsymbol{w}^{(2)} = \boldsymbol{w}^{(1)} + \Delta\boldsymbol{w} = \boldsymbol{w}^{(1)} + \eta\boldsymbol{x}_i t_i = \begin{bmatrix} 0.75 \\ 0.50 \end{bmatrix} - 0.2 \begin{bmatrix} 9.4 \\ 6.4 \end{bmatrix} = \begin{bmatrix} -1.13 \\ -0.78 \end{bmatrix}$$

$$b^{(2)} = b^{(1)} + \Delta b = b^{(1)} + \eta t_i = -0.6 - 0.2 = -0.8$$

其中参数的上标表示对应的迭代步数。算法继续选取第三个样本进行计算，直至遍历完数据集中所有的样本，完成一个完整的训练轮次（Epoch）。需要注意的是，当发生参数更新时，后续样本应立即使用更新后的参数进行预测。使用表6.1中的训练数据，大约经过 500 个训练轮次后，权重向量和偏置逐渐收敛至：

$$w_1 = -1.3, \quad w_2 = -1.1, \quad b = 10.9$$

该结果在二维特征空间中对应一条线性决策边界（如图6.3所示）。权重和偏置的调整可以理解为对这条直线进行平移和旋转，从而逐步优化决策边界，使其更好地将两类样本分开。

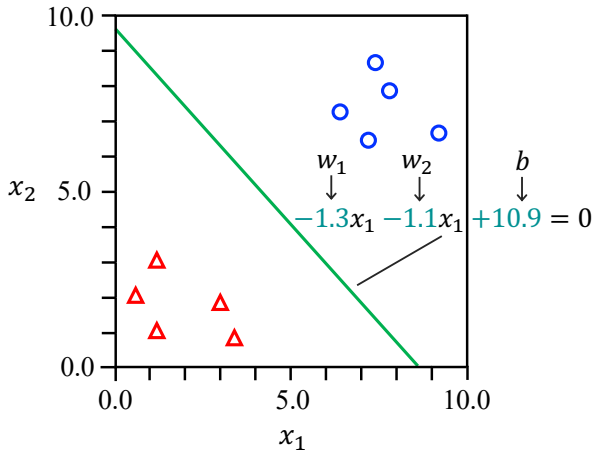


图 6.3: 感知机算法基于表6.1中的训练样本学习得到的线性决策边界 (绿色直线)。红色三角形和蓝色圆圈分别表示两类样本。

感知机算法在当时引发了学术界的广泛关注。其简洁的结构与明确的学习规则，使人们一度对基于神经元模型构建通用智能系统寄予厚望。然而，1969 年美国科学家马文 · 李 · 闵斯基（Marvin Lee Minsky）和美国南非裔数学家西摩 · 奥布里 · 佩珀特（Seymour Aubrey Papert）在他们的著作《感知机》（Perceptrons）[47] 中指出，单个感知机甚至无法解决简单的异或（XOR）问题。根据图6.4(a) 所示的异或真值表，感知机的参数需要同时满足以下四个不等式：

$$w_1 \times 1 + w_2 \times 1 + b < 0 \tag{6.7}$$

$$w_1 \times 1 + w_0 \times 0 + b \geq 0 \tag{6.8}$$

$$w_1 \times 0 + w_2 \times 1 + b \geq 0 \tag{6.9}$$

$$w_1 \times 0 + w_2 \times 0 + b < 0 \tag{6.10}$$

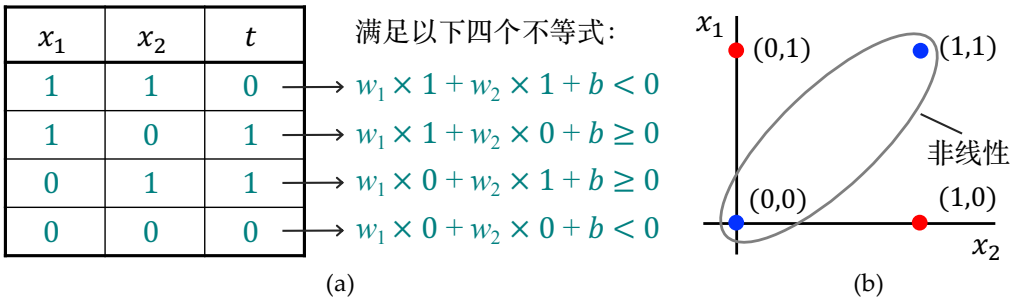


图 6.4: 感知机在异或（XOR）问题上的局限性分析。(a) 异或真值表及其对应的线性不等式组。(b) 异或问题在二维空间中是线性不可分的。红色两点应归为一类，而蓝色两点归属另一类。

将不等式6.8和6.9相加，可得：

$$w_1 + w_2 \geq -2b$$

又由于不等式6.10要求 $b < 0$，因此 $-2b > -b$。结合上式可得：

$$w_1 + w_2 \geq -b$$

然而，不等式6.7要求：

$$w_1 + w_2 < -b$$

两者直接矛盾。因此，该不等式组无解。

单个感知机本质上是一个线性分类器。在二维特征空间中，它的决策边界是一条直线；在 D 维特征空间中，则表现为 $D-1$ 维的线性超平面。单个感知机只能处理线性可分问题，无法解决异或（XOR）等线性不可分问题。闵斯基和佩珀特的这一结论揭示了单个感知机的局限性，也直接导致人工神经网络研究进入了长达十余年的低潮期。

6.3 多层感知机

既然单个感知机无法解决线性不可分问题，那么通过增加神经元的数量并引入多层结构，是否可以突破这一局限？这一问题促使研究者探索更复杂的网络结构，从而推动人工神经网络从单层向多层发展。

我们将单个感知机扩展为如图6.5所示的网络拓扑结构。神经网络的拓扑结构通常用有向图来表示，这样能够较直观地刻画节点之间的连接关系。除输入节点外，图中的每个节点均对应一个感知机，而有向边表示神经元之间的连接关系，其方向指明信息在网络中的传播方向以及节点之间的影响路径。

图6.5展示的是一个全连接（Fully Connected）网络，即每个神经元都会接收来自前一层所有神经元的输入。虽然多层感知机（Multilayer Perceptron）通常采用全连接拓扑结构，但网络的连接也可以是稀疏（Sparse）结构，即部分神经元之间可能没有连接。此外，图中所示的网络仅包含一个隐藏层，但可以将其扩展为包含多个隐藏层的结构。

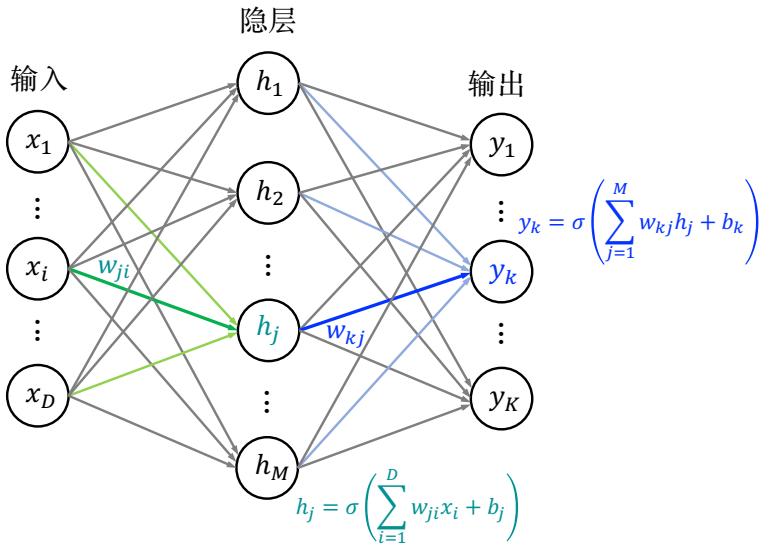


图 6.5: 含一个隐层的多层感知机。除输入节点外，网络中每个节点（以圆圈表示）均代表一个感知机。有向箭头表示连接权重，其中 w_{ji} 表示第 i 维输入到第 j 个隐藏层神经元之间的连接权重，而 w_{kj} 表示第 j 个隐藏层神经元到第 k 个输出神经元之间的连接权重。

从图6.5可以看出，D 维输入向量 $\boldsymbol{x} = (x_1, \ldots, x_D)^\top$ 经过了两次变换。第一次变换通过输入层到隐层的权重 w_{ji} 以及每个隐层神经元的偏置 b_j，得到 M 维隐层输出 $\boldsymbol{h} = (h_1, \ldots, h_M)^\top$，其中隐层第 j 个神经元的输出计算公式为：

$$h_j = \sigma\left(\sum_{i=1}^{D} w_{ji}x_i + b_j\right) \tag{6.11}$$

其中 $\sigma(\cdot)$ 为非线性激活函数，多层感知机中通常使用如下 Sigmoid 函数：

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{6.12}$$

得到隐层表示 $\boldsymbol{h}$ 后，第二次变换通过隐层到输出层的权重 w_{kj} 和输出层神经元的偏置 b_k，得到最终的 K 维输出向量 $\boldsymbol{y} = (y_1, \ldots, y_K)^\top$，其中输出层第 k 个神经元的输出计算如下：

$$y_k = \sigma\left(\sum_{j=1}^{M} w_{kj}h_j + b_k\right) \tag{6.13}$$

因此，整个网络可以表示为如下的复合函数形式：

$$y_k = \sigma\left(\sum_{j=1}^{M} w_{kj}\sigma\left(\sum_{i=1}^{D} w_{ji}x_i + b_j\right) + b_k\right) \tag{6.14}$$

值得注意的是，隐藏层和输出层中的每个神经元均采用 Sigmoid 非线性激活函数，而不是原始感知机中使用的不可微阶跃函数。之所以采用这种非线性激活函数，是因为它在定义域内是可微的，即在各点都存在导数。函数的可微性对于网络参数的优化至关重要，因为许多常用的学习算法需要利用梯度信息来更新神经网络模型的参数。关于可微性对参数优化的具体影响，我们将在第6.4节中详细讨论。

在线性模型一章中已知，我们通常会先采用一组固定的基函数 $\boldsymbol{\phi}(\boldsymbol{x})$ 对输入 $\boldsymbol{x}$ 进行变换，以提取特征表示；随后利用训练数据集学习权重参数 $\boldsymbol{w}$，从而构建线性模型 $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{w}$。若在此基础上进一步施加非线性激活函数 $\sigma(\cdot)$，则可以将模型扩展为广义线性模型：$f(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{w}\right)$。若将公式6.13中的隐藏层输出 h_j 视为对输入 $\boldsymbol{x}$ 的某种基函数变换，则公式6.14所表示的神经网络可以理解为一种广义线性模型。然而，多层感知机与传统广义线性模型的本质区别在于：线性模型的基函数通常是预先固定的，而多层感知机的基函数具有自适应性。具体而言，通过调整输入层到隐层的权重 w_{ji} 及偏置 b_j，网络可以改变这些基函数的形式，从而更好地适应训练数据。借助适当的学习算法（将在下一节中介绍），可以从训练数据中自动学习到隐层表示，这一过程在某种意义上等价于以自适应的方式构建适合任务的基函数。此外，需要强调的是：多层感知机的隐藏层必须包含非线性变换。若缺少这种非线性变换，即便堆叠多个隐藏层，整个网络也会退化为一个广义线性模型。这是因为多个连续的线性变换在数学上等价于单次线性变换，从而失去了多层结构带来的表征优势。

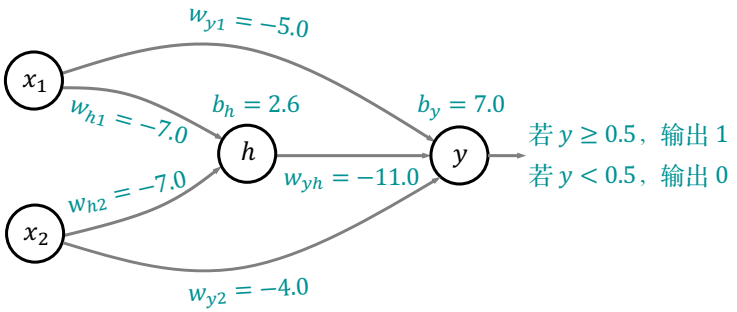
如图6.5所示的多层感知机包含两次变换（输入层到隐藏层、隐藏层到输出层），因此是一个两层网络。需要注意的是，学术界对层数的定义存在不同的惯例。部分文献会将输入层也计入总层数，称其为三层网络。本书统一以变换的次数（即带有参数的层数）来确定网络的层数。神经网络的层数通常也被称为网络的**深度**（Depth），而网络的**宽度**（Width）则是指每一层中神经元的数量。

一般而言，输入层神经元的个数由输入向量的维度确定，而输出层神经元的个数则取决于具体的学习任务。对于多分类任务，输出层神经元的数量通常等于类别的个数。在进行预测时，选择输出值（或响应值）最大的神经元所对应的类别作为预测类别。对于二分类问题，既可以使用两个输出神经元分别表示两个类别；也可以使用一个输出神经元，并通过设定阈值来判断输入样本属于哪个类别。对于回归任务，输出层神经元的数量则取决于需要预测目标值的个数。

隐藏层神经元的数量（即宽度）是模型设计中的关键超参数，其数量的选择通常依赖于具体问题及数据特征。通常情况下，隐藏层宽度会设定为大于输入维度，以便通过多种组合方式对原始输入进行变换，从而提取潜在的（Latent）、对任务有用的高维特征。如果隐藏层神经元的数量小于输入维度，则相当于对原始输入进行了降维或压缩，这可能会丢失部分信息。然而，在某些情况下，例如原始输入中包含较多噪声，或希望获得更加紧致（Compact）的特征表示时，也会有意将隐藏层神经元的数量设置为小于输入维度。

通过在网络中引入隐藏层和非线性激活函数，多层感知机能够对输入特征进行多层逐次的非线性变换，从而显著提升模型的表示能力。此前单个感知机无法解决的异或问题，可以通过如图6.6(a) 所示的两层网络，并配合适当的权重和偏置设置来实现。值得注意的是，该网络结构包含了一种从输入 x_1 和 x_2 直接指向输出 y 的跨层连接 (Skip Connection)。虽然这种跨层连接通常不出现在典型的多层感知机中，但它为输入特征到输出预测提供了一条直接通路。我们将在第7章中看到，类似的跨层连接在较深网络的训练中具有特殊作用和重要意义。

最后需要指出，多层感知机这一术语其实并不十分准确，在一定程度上存在误用。其主要原因如下：



(a)

x_1	x_2	y	输出
1	1	0.11	0
1	0	0.86	1
0	1	0.94	1
0	0	0.03	0

(b)

图 6.6: 求解异或（XOR）问题的两层神经网络。由于网络仅包含一个隐藏层神经元 h 和一个输出神经元 y，为了简化符号，相应的权重和偏置的下标分别使用 h 和 y。(a) 神经网络拓扑结构。(b) 网络具体输入输出对应关系。

- 原单个感知机使用的不可微阶跃函数被 Sigmoid 等可微非线性激活函数所取代，因此每个神经元本质上不再是单个感知机，而更类似于一个逻辑回归模型。
- 除了网络层数增加之外，每一层不再只有单个神经元，而是由多个神经元构成，即网络的深度和宽度都有所增加。

6.4 反向传播算法

前面的讨论假设多层网络的权重和偏置参数已经针对具体任务进行了合理设置，但实际上，面临的主要挑战是如何基于训练数据自动优化这些网络参数，也就是人工神经网络的学习问题。

我们先讨论回归问题。假设给定一个包含 N 个样本的训练数据集 $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{t}_i)\}_{i=1}^N$，其中 $\boldsymbol{x}_i$ 表示第 i 个样本的 D 维输入向量，其对应的预测目标为 K 维向量 $\boldsymbol{t}_i$。我们仍然采用如图6.5所示的两层网络结构。由于是回归任务，我们采用如下平方误差损失函数：

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{y}_i - \boldsymbol{t}_i)^2 \tag{6.15}$$

其中 $\boldsymbol{\theta}$ 表示网络中所有可调权重和偏置参数的集合，而 $\boldsymbol{y}_i = (y_1, \ldots, y_K)^\top$ 表示网络对于第 i 个样本 $\boldsymbol{x}_i$ 的预测输出。每一维预测输出 y_k 的具体计算方式如下：

$$y_k = g\left(\sum_{j=1}^{M} w_{kj}\sigma\left(\sum_{i=1}^{D} w_{ji}x_i + b_j\right) + b_k\right) \tag{6.16}$$

其中 $g(z) = z$ 为恒等激活函数（Identity Function）。与公式6.14对比可知，此处将输出层神经元的非线性激活函数 Sigmoid 替换为恒等函数。这样做的原因在于，回归任务通常要求模型能够输出任意实数值，而 Sigmoid 等非线性激活函数通常具有有界性，例如 Sigmoid 函数会将输出限制在 $(0, 1)$ 区间内，因此无法有效拟合目标值超出该范围的回归问题。

若将公式6.16所示的计算过程写成向量和矩阵形式，则可表示为：

$$\boldsymbol{y}_i = \mathbf{W}_2\sigma\left(\mathbf{W}_1\boldsymbol{x}_i + \boldsymbol{b}_1\right) + \boldsymbol{b}_2 \tag{6.17}$$

其中矩阵 $\mathbf{W}_1$ 和向量 $\boldsymbol{b}_1$ 分别表示第一层网络的所有权重和偏置，而矩阵 $\mathbf{W}_2$ 和向量 $\boldsymbol{b}_2$ 则分别表示第二层网络的权重和偏置参数。我们的学习目标是通过调整所有参数 $\boldsymbol{\theta} = \{\mathbf{W}_1, \boldsymbol{b}_1, \mathbf{W}_2, \boldsymbol{b}_2\}$ 来最小化损失函数（即公式6.15）。

与简单的线性模型不同，我们不能通过将损失函数 $\mathcal{L}(\boldsymbol{\theta})$ 关于参数 $\boldsymbol{\theta}$ 求偏导并令导数为零（即 $\nabla\mathcal{L}(\boldsymbol{\theta}) = 0$）来直接解出最优参数 $\boldsymbol{\theta}$。其原因之一在于，网络隐层中引入了非线性激活函数，并且各层之间存在嵌套的复合映射（如公式6.17所示），导致损失函数对参数的导数是高度非线性的方程组，通常不存在闭式解。此外，即便是只有一个隐藏层的神经网络，也存在权重**空间对称性**（Weight-space Symmetries）。具体而言，如果交换隐藏层中任意第 i 个和第 j 个神经元的所有连接权重（包括输入和输出连接）及偏置，网络的最终输出不会发生变化。当隐藏层有 M 个神经元时，这种交换方式共有 $M!$（M 的阶乘）种可能，即对隐藏层神经元的全排列。这意味着网络参数空间中存在大量等价解，进一步表明难以通过解析方法直接求解网络的最优参数。

由于无法利用公式 $\nabla\mathcal{L}(\boldsymbol{\theta}) = 0$ 求出解析解，我们只能借助迭代数值方法。连续非线性函数的优化是一个广泛研究的课题，已有大量文献探讨如何高效求解此类问题。大多数优化方法遵循如下逻辑：首先设有一个初始参数 $\boldsymbol{\theta}^{(0)}$；随后在每一迭代步 t 中，通过在参数空间中找到一个能够使目标函数下降的更新方向 $\Delta\boldsymbol{\theta}^{(t)}$，然后按照如下规则对参数进行更新：

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)} \tag{6.18}$$

为了确定更新方向 $\Delta\boldsymbol{\theta}$（此处省略了迭代步数的上标来简化符号表示），我们考察损失函数在当前参数 $\boldsymbol{\theta}$ 处的一阶泰勒展开：

$$\mathcal{L}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})\Delta\boldsymbol{\theta} + o(\|\Delta\boldsymbol{\theta}\|) \tag{6.19}$$

其中 $o(\|\Delta\boldsymbol{\theta}\|)$ 为一阶余项，表示比线性项更高阶的无穷小项。为了使损失函数下降，参数更新方向 $\Delta\boldsymbol{\theta}$ 应与当前梯度方向相反。若令 $\Delta\boldsymbol{\theta} = -\eta\nabla\mathcal{L}(\boldsymbol{\theta})$（其中 $\eta > 0$ 为一个足够小的值），代入公式6.19并忽略高阶无穷小项，可得：

$$\mathcal{L}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) - \eta\|\nabla\mathcal{L}(\boldsymbol{\theta})\|^2 \tag{6.20}$$

由于梯度的平方范数 $\|\nabla\mathcal{L}(\boldsymbol{\theta})\|^2$ 在梯度不为零时恒大于零，因此只要步长 η 足够小（使得高阶无穷小项可以忽略），可以保证：

$$\mathcal{L}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) < \mathcal{L}(\boldsymbol{\theta}) \tag{6.21}$$

这表明沿着负梯度方向对参数进行调整能够降低损失函数的值，这一结论也构成了**梯度下降法**（Gradient Descent）的理论基础。若当前参数的梯度为零，则在一阶近似下无法找到使损失函数进一步下降的方向，此时该参数点称为驻点（Stationary Point）。

图6.7展示了梯度下降法单步参数更新的示意图。为了直观起见，图中绘制的是二维空间的误差曲线，而实际优化通常发生在高维空间中的误差曲面上。在每一次迭代中，模型参数沿负梯度方向，以设定的学习步长对当前参数 $\boldsymbol{\theta}^{(t)}$ 进行更新，从而得到新的参数 $\boldsymbol{\theta}^{(t+1)}$。根据梯度下降法的原理，新参数对应的损失函数值通常小于原参数的损失值。这一过程常被形象地称为“下山法”，即将参数优化过程类比为从高山上往下走。在当前位置视野所及的范围内（即一阶

近似）找到能够下山（使得损失函数下降）的最快方向（即梯度的反方向），沿这个方向走到新的位置，然后再观察和确定下一步下山的最佳方向。如此循环，逐步走到山下（即图中的局部极小值点）。由于该方法基于一阶估计，学习步长的选取比较重要。步长过大可能会导致参数在极小值附近发生震荡，而步长过小则会导致收敛速度过慢。在后续章节中，我们将讨论为什么神经网络的损失函数通常存在多个局部极小值，并且其中许多局部极小值并非全局最优解。

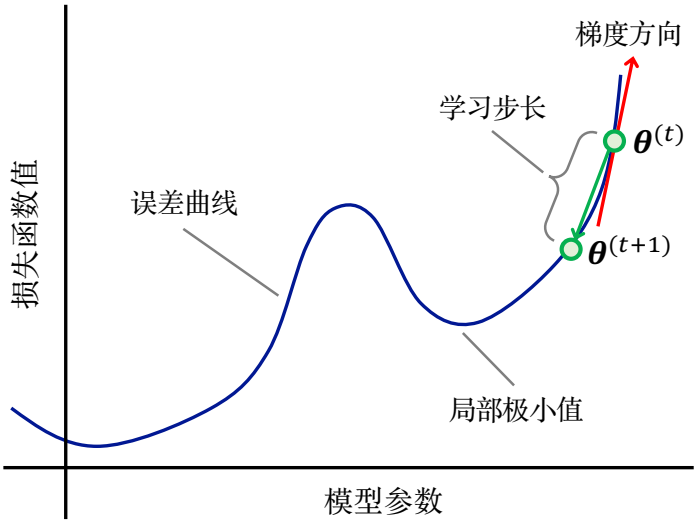


图 6.7: 梯度下降法参数更新示意图。模型参数沿负梯度方向，以设定的学习步长对当前参数 $\boldsymbol{\theta}^{(t)}$ 进行更新，从而得到新的参数 $\boldsymbol{\theta}^{(t+1)}$，使损失函数值逐步减小。

针对人工神经网络的多层结构，结合复合函数求导的链式法则实现的梯度下降法即为**反向传播算法**（Backpropagation）[59]。该算法的基本过程如下：首先利用一个或多个训练样本计算网络的损失函数值；随后从输出层开始，按照输出层到隐藏层（如果存在多个隐藏层，还包括隐藏层之间），再到输入层的顺序，逐层计算各层参数对损失函数的梯度（通常要求激活函数可微）；最后根据计算得到的梯度，对网络中的所有参数进行更新。这一过程通常需要迭代多次，直到满足预先设定的停止条件，例如达到最大迭代次数或损失函数梯度的范数低于设定阈值。

接下来，我们以如图6.5所示的两层网络结构和平方误差损失函数（公式6.15）为例，具体说明反向传播算法的计算流程。在掌握基本计算流程之后，我们可以自然地将其扩展到更深的多层网络结构和其他类型的损失函数。

对于给定的输入样本 $\boldsymbol{x}$，网络从输入到损失函数值的完整前向计算过程如下：

$$\boldsymbol{x} \xrightarrow{w_{ji},b_j} \boldsymbol{v} \xrightarrow{\sigma(\cdot)} \boldsymbol{h} \xrightarrow{w_{kj},b_k} \boldsymbol{y} \xrightarrow{\boldsymbol{t}} \mathcal{L} \tag{6.22}$$

其中新引入的符号 $\boldsymbol{v}$ 表示隐藏层的净输入向量，其第 j 个神经元的取值为 $\sum_{i=1}^{D} w_{ji}x_i + b_j$，$\sigma(\cdot)$ 为 Sigmoid 非线性激活函数，$\boldsymbol{h} = \sigma(\boldsymbol{v})$ 为隐藏层的输出向量，而 $\mathcal{L}$ 则表示基于网络预测输出 $\boldsymbol{y}$ 和目标向量 $\boldsymbol{t}$ 计算得到的损失函数值。由于输出层神经元采用了恒等激活函数，因此可以忽略该激活函数的影响。此时，输出向量 $\boldsymbol{y}$ 的第 k 个分量的取值为 $\sum_{j=1}^{M} w_{kj}h_j + b_k$，即输出层不再对线性组合后的净输入施加额外的变换。

神经网络的前向预测按公式6.22所示的箭头方向进行，而计算参数梯度则按相反方向进行。这也正是“反向传播”名称的由来。我们首先计算平方误差损失函数 $\mathcal{L}$ 关于输出向量 $\boldsymbol{y}$ 的梯度。对于给定的输入样本 $\boldsymbol{x}$ 及其对应的网络输出 $\boldsymbol{y}$ 与目标向量 $\boldsymbol{t}$，单样本的平方误差损失函数为：

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2 \tag{6.23}$$

其中 K 为输出层神经元的数量。为了使推导过程更加直观，我们针对每一个具体维度进行推导。对于输出层的第 k 个维度，其偏导数为：

$$\frac{\partial \mathcal{L}}{\partial y_k} = y_k - t_k \tag{6.24}$$

为了简化后续推导，我们引入符号 δ_k 来表示损失函数关于输出层第 k 维输出 y_k 的偏导数，即定义：

$$\delta_k = y_k - t_k \tag{6.25}$$

由于输出层神经元取消了非线性激活函数（采用恒等激活函数代替），符号 δ_k 也表示损失函数关于输出层第 k 维净输入的偏导数。在神经网络学习算法中，δ_k 又通常被称为**输出层误差项**，其值等于预测值与目标值之间的差异。

接下来，我们计算损失函数关于连接隐藏层与输出层的权重 w_{kj} 的偏导数。根据链式法则，可得：

$$\frac{\partial \mathcal{L}}{\partial w_{kj}} = \frac{\partial \mathcal{L}}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{kj}} = \delta_k h_j \tag{6.26}$$

同理，损失函数关于输出层神经元偏置 b_k 的偏导数为：

$$\frac{\partial \mathcal{L}}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial y_k} \cdot \frac{\partial y_k}{\partial b_k} = \delta_k \times 1 = \delta_k \tag{6.27}$$

我们接下来沿公式6.22所示箭头的反方向继续推进。遵循反向传播的过程，若要计算损失函数关于输入层至隐藏层的权重 w_{ji} 和隐藏层偏置 b_j 的偏导数，必须先求得损失函数对隐藏层输出 $\boldsymbol{h}$ 和隐藏层净输入 $\boldsymbol{v}$ 的梯度。我们先来考察损失函数关于隐藏层输出的梯度。在全连接网络结构中，隐藏层的任意一个神经元输出都会影响所有输出层神经元的净输入。因此，在计算损失函数对隐藏层第 j 个神经元输出 h_j 的偏导数时，需要先分别计算各输出层神经元对该 h_j 的梯度贡献，再将这些贡献进行累加。综上所述，根据链式法则可得：

$$\frac{\partial \mathcal{L}}{\partial h_j} = \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_j} = \sum_{k=1}^{K} \delta_k \cdot \frac{\partial y_k}{\partial h_j} = \sum_{k=1}^{K} \delta_k w_{kj} \tag{6.28}$$

随后，根据隐藏层激活函数 $h_j = \sigma(v_j)$，我们可以按如下公式计算损失函数关于隐藏层第 j 个神经元净输入 v_j 的偏导数：

$$\frac{\partial \mathcal{L}}{\partial v_j} = \frac{\partial \mathcal{L}}{\partial h_j} \cdot \frac{\partial h_j}{\partial v_j} = \left(\sum_{k=1}^{K} \delta_k w_{kj}\right) \cdot \frac{\partial h_j}{\partial v_j} = \left(\sum_{k=1}^{K} \delta_k w_{kj}\right) \sigma'(v_j) = \left(\sum_{k=1}^{K} \delta_k w_{kj}\right) h_j(1 - h_j) \tag{6.29}$$

在上述推导中，我们利用了 Sigmoid 激活函数 $\sigma(z)$ 对自变量 z 的导数具有如下简洁形式：

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{6.30}$$

类似地，我们引入符号 δ_j 来表示损失函数关于隐藏层第 j 个神经元净输入 v_j 的偏导数。该项通常被称为**隐藏层误差项**，它刻画了隐藏层神经元对全局损失的敏感程度，即隐藏层神经元净输入发生微小扰动时对损失函数值影响的方向与强度。

至此，我们可以根据链式法则计算损失函数关于连接输入层与隐藏层权重 w_{ji} 的偏导数：

$$\frac{\partial \mathcal{L}}{\partial w_{ji}} = \frac{\partial \mathcal{L}}{\partial v_j} \cdot \frac{\partial v_j}{\partial w_{ji}} = \delta_j \cdot \frac{\partial v_j}{\partial w_{ji}} = \delta_j x_i \tag{6.31}$$

同理，损失函数关于隐藏层偏置 b_j 的偏导数为：

$$\frac{\partial \mathcal{L}}{\partial b_j} = \frac{\partial \mathcal{L}}{\partial v_j} \cdot \frac{\partial v_j}{\partial b_j} = \delta_j \cdot \frac{\partial v_j}{\partial b_j} = \delta_j \times 1 = \delta_j \tag{6.32}$$

图6.8展示了在误差反向传播过程中，计算损失函数关于输入层与隐藏层之间连接权重 w_{ji} 梯度时所涉及的相关变量及其相互关系。此外，由公式6.26和6.31可以看出，神经网络中权重梯度的计算具有统一而简洁的形式，即权重梯度等于该权重所连接神经元的误差与输入的乘积。具体而言，从隐藏层到输出层的权重梯度等于输出层误差项与隐藏层输出（即输出层输入）的乘积，而从输入层到隐藏层的权重梯度则等于隐藏层误差项与输入层输入的乘积。

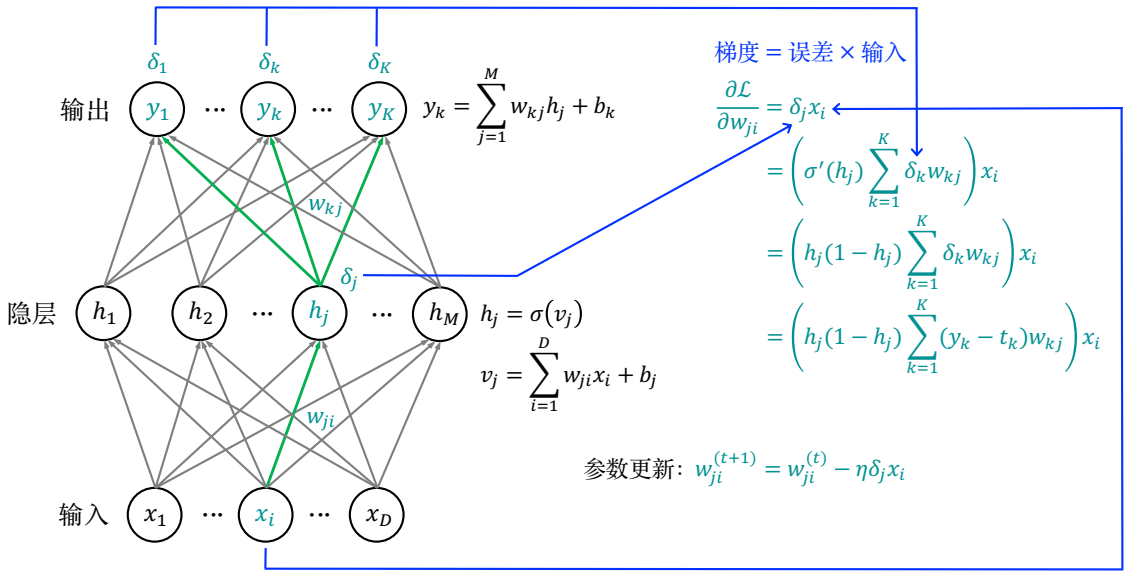


图 6.8: 两层全连接神经网络中输入层与隐藏层连接权重的梯度计算示意图。图中绿色有向箭头和变量表示在反向传播过程中参与权重 w_{ji} 梯度计算的各依赖项。

接下来我们讨论分类问题。在神经网络设计中，对于二分类问题，通常也在输出层配置两个神经元，分别对应两个不同的类别。采用这种设计的原因在于：如果仅使用一个输出神经元，则通常需要额外设定一个阈值，以判断输入样本应被划分到哪一个类别。而阈值这一超参数的选择可能会对模型的预测性能产生影响，其选择过程会增加模型调试的复杂度。因此，为了保持网络模型的通用性，无论是二分类问题还是多分类问题，我们此处均采用统一的处理方式，即都将其视为多分类问题，并让输出层神经元的个数等于类别的数量。

为了使网络最后一层的输出能够被解释为对应各个类别的概率分布，通常将输出层的激活

函数替换为 Softmax 函数。此时，输出层第 k 个神经元的净输入为：

$$a_k = \sum_{j=1}^{M} w_{kj} h_j + b_k \tag{6.33}$$

然后通过 Softmax 函数对净输入进行变换后，可以得到第 k 个类别的预测概率输出 y_k：

$$y_k = \frac{\exp(a_k)}{\sum_{s=1}^{K} \exp(a_s)} \tag{6.34}$$

使用 Softmax 作为输出层激活函数可以保证每个神经元的输出为非负值，且所有神经元的输出之和为一，从而可以将输出向量 $\boldsymbol{y} = (y_1, \ldots, y_K)^\top$ 解释为各类别的预测概率分布。

在多分类任务中，通常采用交叉熵损失函数。给定输入样本 $\boldsymbol{x}$ 及其对应的预测输出向量 $\boldsymbol{y}$，若目标向量 $\boldsymbol{t} = (t_1, \ldots, t_K)^\top$ 采用 1-of-K 编码（独热编码）表示类别，则该样本的损失函数为：

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{k=1}^{K} t_k \ln y_k \tag{6.35}$$

我们先来计算交叉熵损失函数关于网络输出向量 $\boldsymbol{y}$ 的梯度。对于输出向量的第 k 个分量 y_k 的偏导数为：

$$\frac{\partial \mathcal{L}}{\partial y_k} = -\frac{t_k}{y_k} \tag{6.36}$$

接下来，我们将计算损失函数关于输出层第 k 个神经元净输入 a_k 的偏导数。根据 Softmax 函数的定义，输出层每个神经元的净输入都会影响所有输出神经元的最终输出。因此，在计算损失函数对输出层第 k 个神经元净输入 a_k 的偏导数时，需要先分别计算各神经元最终输出对该 a_k 的梯度贡献，然后再将这些贡献进行累加。综上所述，根据链式法则可得：

$$\frac{\partial \mathcal{L}}{\partial a_k} = \sum_{s=1}^{K} \frac{\partial \mathcal{L}}{\partial y_s} \cdot \frac{\partial y_s}{\partial a_k} = \sum_{s=1}^{K} -\frac{t_s}{y_s} \cdot \frac{\partial y_s}{\partial a_k} \tag{6.37}$$

对于上式中的 $\frac{\partial y_s}{\partial a_k}$，我们要分 $s = k$ 和 $s \neq k$ 两种情况进行分析。当 $s \neq k$ 时，我们有：

$$\begin{aligned}
\frac{\partial y_s}{\partial a_k} &= \frac{-\exp(a_s)\exp(a_k)}{\left(\sum_{i=1}^{K} \exp(a_i)\right)^2} \\
&= -\frac{\exp(a_s)}{\sum_{i=1}^{K} \exp(a_i)} \cdot \frac{\exp(a_k)}{\sum_{i=1}^{K} \exp(a_i)} \\
&= -y_s y_k
\end{aligned} \tag{6.38}$$

当 $s = k$ 时，则有：

$$\begin{aligned}
\frac{\partial y_s}{\partial a_k} &= \frac{\exp(a_k)}{\sum_{i=1}^{K} \exp(a_i)} + \frac{-\exp(a_k)\exp(a_k)}{\left(\sum_{i=1}^{K} \exp(a_i)\right)^2} \\
&= \frac{\exp(a_k)}{\sum_{i=1}^{K} \exp(a_i)} \left(1 - \frac{\exp(a_k)}{\sum_{i=1}^{K} \exp(a_i)}\right) \\
&= y_k(1 - y_k)
\end{aligned} \tag{6.39}$$

将上述两种结果代入公式6.37，可得：

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a_k} &= \sum_{s=1}^{K} -\frac{t_s}{y_s} \cdot \frac{\partial y_s}{\partial a_k} \\ &= -\frac{t_k}{y_k} y_k(1-y_k) + \sum_{s \neq k} -\frac{t_s}{y_s}(-y_s y_k) \\ &= -t_k + t_k y_k + \sum_{s \neq k} t_s y_k \\ &= -t_k + y_k \sum_{s=1}^{K} t_s \quad (\sum_{s=1}^{K} t_s = 1) \\ &= y_k - t_k \end{aligned} \tag{6.40}$$

综合公式6.24和6.40可以发现，无论是回归问题还是分类问题，损失函数关于输出层第 k 个神经元净输入的偏导数均具有统一且简洁的形式，即预测输出减去目标值。也就是说，在回归任务中，采用平方误差损失配合输出层恒等激活函数；在分类任务中，采用交叉熵损失配合输出层 Softmax 激活函数，损失函数关于输出层净输入的梯度均可表示为 $y_k - t_k$。这一结果具有非常直观的解释：当 $y_k - t_k > 0$ 时，梯度为正，在梯度下降法下参数的调整方向为负，即通过减小第 k 个输出神经元的净输入来降低预测值 y_k；而当 $y_k - t_k < 0$ 时，梯度为负，则调整方向为正，即通过增加第 k 个输出神经元的净输入来提高预测值 y_k。通过这样的调整，网络输出会逐步向目标值 t_k 靠近。

算法 6.1: 反向传播算法（小批量随机梯度下降）

输入: 训练样本集 $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{t}_i)\}_{i=1}^{N}$、损失函数 $\mathcal{L}(\boldsymbol{\theta})$、学习步长 η、批量大小 B 和最大迭代次数 H。

初始化: 随机初始化网络权重与偏置参数 $\boldsymbol{\theta}$，并设当前迭代次数 $h = 1$；

while $h \leq H$ **do**

- 梯度清零：$\mathbf{g} \leftarrow 0$；
- 从训练集中随机采样 B 个样本；
- **for** $i \leftarrow 1$ **to** B **do**
  - **前向传播:** 计算样本 $\boldsymbol{x}_i$ 在各隐藏层和输出层的输出；
  - **反向传播:** 利用链式法则计算损失函数 $\mathcal{L}(\boldsymbol{\theta})$ 关于参数 $\boldsymbol{\theta}$ 的梯度 $\nabla \mathcal{L}(\boldsymbol{\theta})$；
  - **梯度累积:** $\mathbf{g} \leftarrow \mathbf{g} + \nabla \mathcal{L}(\boldsymbol{\theta})$；
- **计算更新:** $\Delta \boldsymbol{\theta} \leftarrow -\eta \frac{1}{B} \cdot \mathbf{g}$；
- **更新参数:** $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$；
- **收敛判断:** 当验证集上的损失在连续若干次迭代中不再下降时，则提前终止算法；
- $h \leftarrow h + 1$；

输出: 经训练后得到的神经网络参数 $\boldsymbol{\theta}$。

此前讨论梯度计算时，我们仅考虑单个样本。然而在神经网络的实际训练中，通常会计算多个样本的梯度之后，再用其平均值来对网络参数进行一次更新。这样做的主要原因是：单个样本可能包含噪声，导致梯度方向剧烈波动，造成训练过程不稳定。使用小批量样本（Mini-Batch）

并取其平均梯度，可以获得更可靠的梯度估计，从而提高训练的整体稳定性。然而，批量大小也不宜过大。一方面，随着批次中所包含的样本数量增加，梯度估计准确性的增益逐渐减小，而单次参数更新的计算量则始终呈线性增加；另一方面，过大的单批样本数量可能降低网络跳出局部极小值的能力（详见后续讨论）。在实践中，我们通常会根据设定的批量大小将训练集划分为若干个批次（Batch），每个批次进行一次参数更新。当遍历完数据集中所有样本后，即完成一个训练轮次（Epoch）。每经过一个轮次，通常还会对样本顺序进行重洗（Shuffling），然后再重新划分批次。这是为了确保网络在不同轮次中观察到的样本顺序具有随机性，避免网络由于记忆固定的样本序列而陷入过拟合，或者产生周期性的梯度估计偏差。

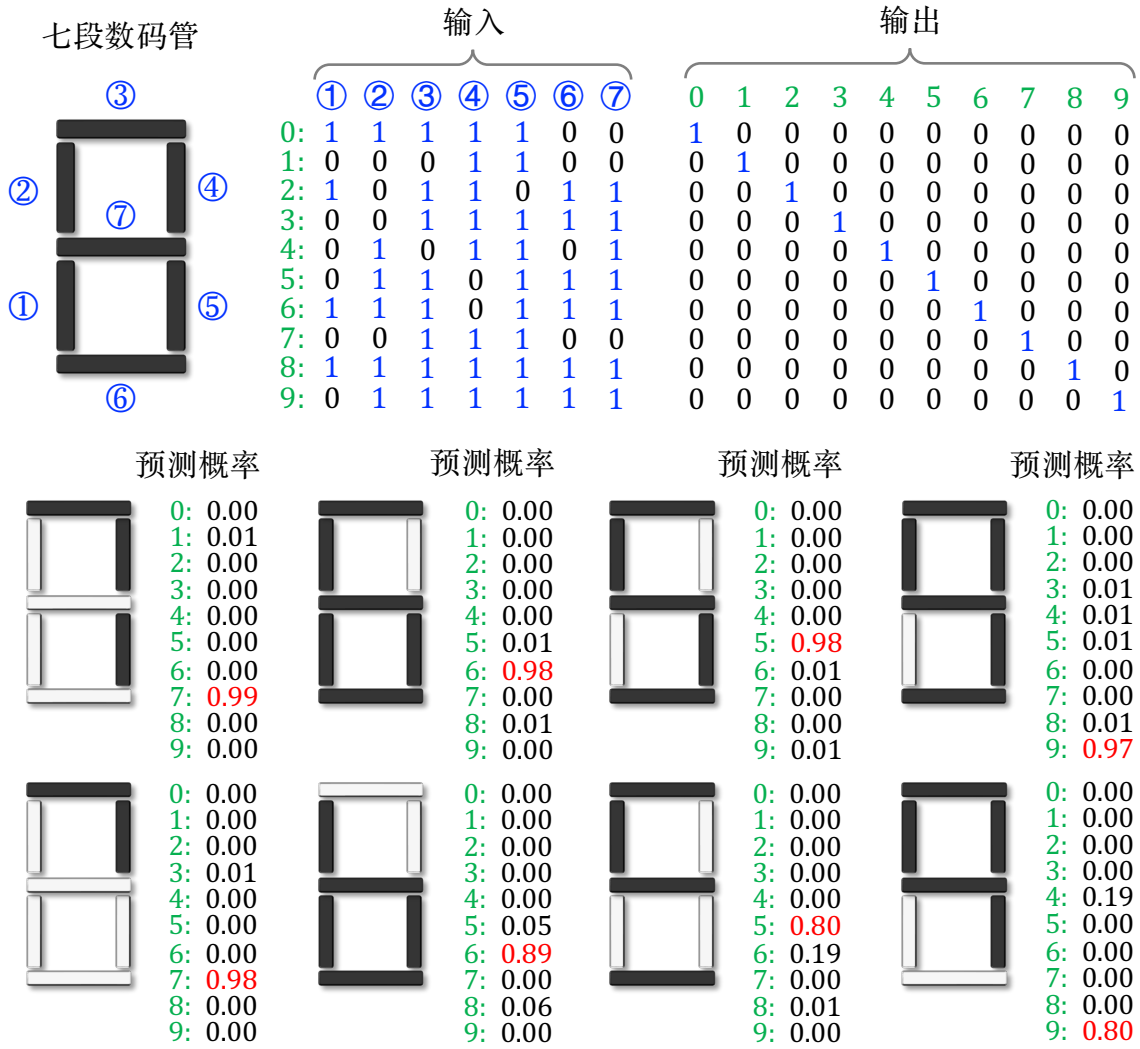


图 6.9: 两层全连接神经网络在数字识别中的示例。数字由七段数码管表示：对应位置取值为 1 表示该段点亮，取值为 0 表示该段熄灭。图上方给出了神经网络输入与输出的编码方式，图下方展示了训练完成的神经网络对不同形状数字的识别结果及其预测概率。其中，第一排为标准数字表示，第二排为不规则数字表示。从结果可以看出，即使是训练过程中未出现过的不规则数字形式，网络的识别结果仍与人类判断基本一致，说明该神经网络具有较好的泛化能力。

算法6.1展示了反向传播算法（小批量随机梯度下降版本）的完整训练过程。其中，第 9 行计算了一个批次样本的平均负梯度，并乘以学习率，以确定网络参数的更新量。在网络训练中，通常会采用早停（Early Stopping）策略来防止过拟合。早停的判定准则有多种选择，这里主要根据验证集上的性能变化趋势（算法第 11 行）来判断是否在达到最大迭代次数前停止训练。

我们通过一个简单的数字识别任务来演示两层全连接神经网络的建模能力及其泛化性能。如图6.9所示，计算器和交通信号灯等通常采用七段数码管显示数字。通过对这七个发光段进行固定编号，可以将任何显示模式表示为一个特征向量：若某段点亮记为 1，熄灭则记为 0。由此，每个数字均可由一个 7 维二进制编码表示，可以作为神经网络的输入。由于该任务的目标是将输入划分为 0 至 9 这 10 个类别，神经网络的输出层共有 10 个神经元，并且采用 1-of-K（独热编码）表示目标向量，隐藏层神经元数量则设为 18 个。该任务属于分类问题，因此采用交叉熵损失函数，并在输出层使用 Softmax 激活函数。

训练集仅包含 10 个标准样本，即数字 0 至 9 的标准数码管表示。对训练后的神经网络进行测试（见图6.9下方的样例）可以发现：对于标准输入，网络能够给出正确预测，且正确类别的输出概率较高，表明模型对预测结果具有较高的置信度。更值得关注的是，对于一些不规则或存在缺损的数字表示（如图 6.9 最下方一排例子），尽管这些模式在训练阶段并未出现，但网络的识别结果仍与人类的判断基本一致。这表明神经网络并非简单地“记忆”训练数据，而是通过学习提取出了具有一定泛化能力的特征表示。

6.5 通用近似定理

多层感知机是典型的前馈神经网络（Feedforward Neural Network）。作为人工神经网络最基础的架构之一，**前馈神经网络**的显著特点在于信息的单向流动性：信号从输入层出发，逐层经过一个或者多个隐藏层传递，最终到达输出层。在该拓扑结构中，既不存在同层神经元之间的侧向连接，也不存在后层指向前层的反馈连接。大量理论研究表明，前馈神经网络具有极强的函数表达能力，理论上能够以任意精度逼近任何连续函数。1989 年，达特茅斯学院（Dartmouth College）的乔治 · 西本科 (George Cybenko) 证明了以下通用近似定理 [16]：

定理 6.1 (通用近似定理)

设 $\mathcal{C}([0,1]^D, \mathbb{R})$ 为定义在单位超立方体 $[0,1]^D$ 上的实值连续函数集合。对于该集合中的任意函数 $f : [0,1]^D \to \mathbb{R}$ 以及任意 $\epsilon > 0$，存在一个有界、严格单调递增且非常数的连续函数 σ，以及一个正整数 M、一组实数 w_j, b_j 和向量 $\boldsymbol{w}_j \in \mathbb{R}^D\,(j = 1, \ldots, M)$，使得函数

$$g(\boldsymbol{x}) = \sum_{j=1}^{M} w_j \sigma\left(\boldsymbol{w}_j^{\top} \boldsymbol{x} + b_j\right) \tag{6.41}$$

能够一致近似于函数 f，即满足：

$$\sup_{\boldsymbol{x}} \left| f(\boldsymbol{x}) - g(\boldsymbol{x}) \right| < \epsilon, \boldsymbol{x} \in [0,1]^D \tag{6.42}$$

♡

事实上，上述定理中的函数 σ 只要不是多项式函数，通用近似定理依然成立。从公式6.41可看出，模型的表达能力（即其能够表示或逼近函数的复杂度）主要由参数 M 的大小决定。若将该式视为一个单隐层（两层）全连接神经网络，则 M 对应于隐藏层神经元的数量。因此，从直观上讲，该定理表明：对于两层神经网络，只要增加隐藏层神经元的数量（即增加网络的宽度），理论上就能够逼近任意复杂的非线性函数（图6.10展示了单隐层网络对于四种不同函数的拟合结果）。此外，虽然上述定理是以标量输出（一维）为例给出的，但其结论同样适用于多维输出的情形。只需相应地增加隐层神经元规模及其权重与偏置，即可实现对多维输出函数的近似。

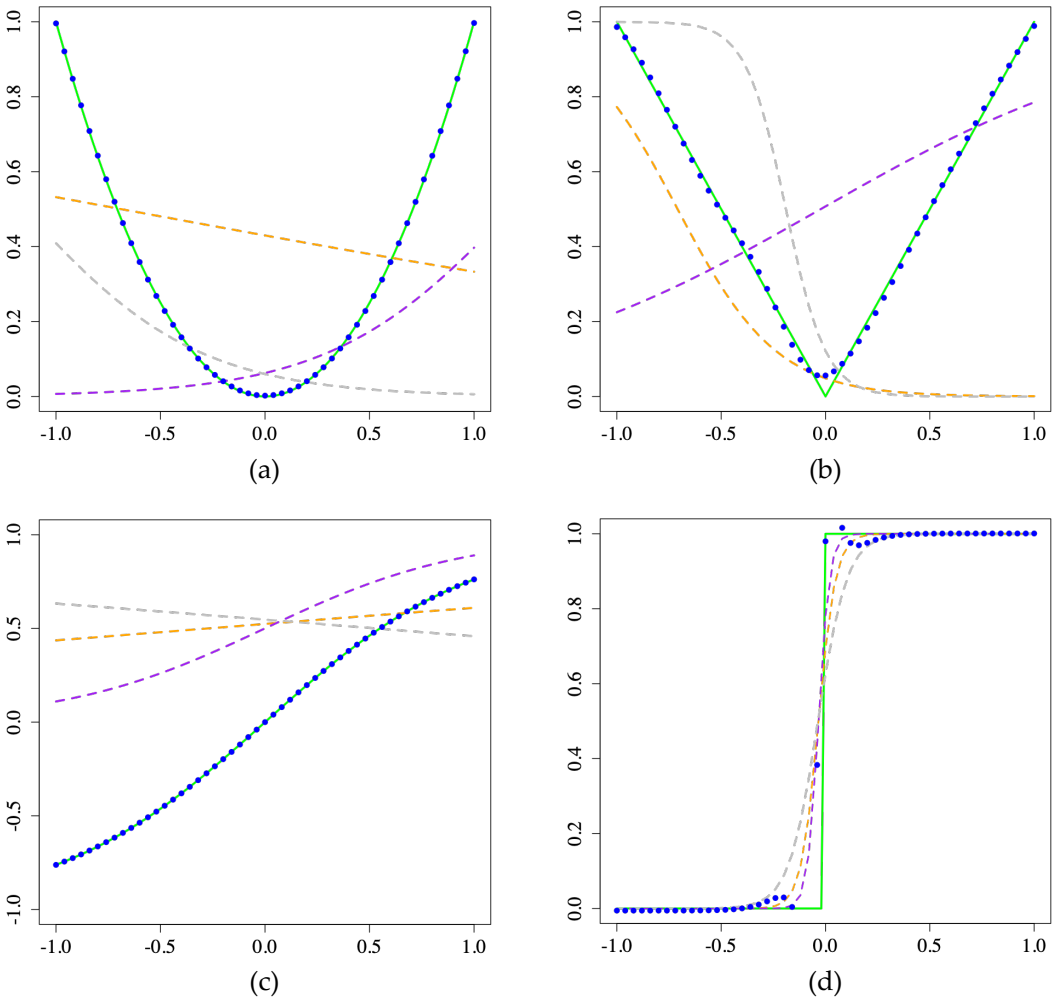


图 6.10: 单隐藏层神经网络拟合能力示意图。该网络包含一个输入和一个输出，隐藏层设有 3 个神经元。训练时使用了 30 个在区间 $[-1, 1]$ 上等距采样的样本，而图中显示的测试集则使用同一区间内的 50 个等距采样点。训练时采用平方误差损失函数，隐藏层神经元使用 Sigmoid 激活函数，而输出层神经元使用恒等激活函数。图中绿色曲线表示应拟合的目标函数，蓝点表示网络预测输出。橙色、紫色和灰色虚线分别表示三个隐藏层神经元的输出。最终网络预测值由这三个隐藏层输出的线性组合与输出层偏置相加得到。(a) 平方函数 $f(x) = x^2$。(b) 绝对值函数 $f(x) = |x|$。(c) 双曲正切函数 $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$。(d) 单位阶跃函数（Heaviside Step Function）$\mathrm{H}(x) = 0, x < 0;\ \mathrm{H}(x) = 1, x \geq 0$。

需要强调的是，该定理仅从存在性角度证明了单隐层神经网络作为通用拟合器的潜力，其前提是假设网络的权重和偏置已被设置为恰当的值。在实际应用中，仍然需要有效的学习算法来确定这些参数。在第6.4节已经介绍了如何使用反向传播算法来学习网络参数，但我们在后续分析中会发现，神经网络的损失函数曲面在参数空间中存在大量的局部极小值点，其中许多并非全局极小值。这会导致基于梯度下降的反向传播算法容易陷入局部极小值，并且其训练结果对网络的初始值比较敏感，不同的随机初始值可能会使网络收敛到不同的局部极小值点。

为了分析反向传播算法收敛结果的性质，我们考察损失函数在当前参数 $\boldsymbol{\theta}$ 处的二阶泰勒展开形式：

$$\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \nabla\mathcal{L}(\boldsymbol{\theta})^\top\Delta\boldsymbol{\theta} + \frac{1}{2}\Delta\boldsymbol{\theta}^\top\mathbf{H}\Delta\boldsymbol{\theta} + o\left(\|\Delta\boldsymbol{\theta}\|\right)^2 \tag{6.43}$$

其中 $\mathbf{H} = \nabla^2\mathcal{L}(\boldsymbol{\theta})$ 为海森矩阵（即损失函数关于参数的二阶导数矩阵），而 $o\left(\|\Delta\boldsymbol{\theta}\|\right)^2$ 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，用以刻画当参数从 $\boldsymbol{\theta}$ 沿方向 $\Delta\boldsymbol{\theta}$ 移动时，损失函数值的局部变化情况。

假设公式6.43中的海森矩阵 $\mathbf{H}$ 为半正定，即对于任意微小的变化 $\Delta\boldsymbol{\theta}$ 均有 $\Delta\boldsymbol{\theta}^\top\mathbf{H}\Delta\boldsymbol{\theta} \geq 0$，且损失函数在当前参数 $\boldsymbol{\theta}$ 处的梯度为零，即 $\nabla\mathcal{L}(\boldsymbol{\theta}) = 0$。则根据二阶泰勒展开可知，$\mathcal{L}(\boldsymbol{\theta})$ 是损失函数的一个局部极小值。原因如下：

$$\mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) \approx \mathcal{L}(\boldsymbol{\theta}) + \underbrace{\nabla\mathcal{L}(\boldsymbol{\theta})^\top\Delta\boldsymbol{\theta}}_{=0} + \frac{1}{2}\underbrace{\Delta\boldsymbol{\theta}^\top\mathbf{H}\Delta\boldsymbol{\theta}}_{\geq 0} \;\Rightarrow\; \mathcal{L}(\boldsymbol{\theta}) \leq \mathcal{L}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) \tag{6.44}$$

即在 $\boldsymbol{\theta}$ 附近沿任意方向移动，损失函数都不会下降，因此 $\boldsymbol{\theta}$ 是局部极小值。若损失函数是定义在凸集（即集合中任意两点之间的线段都完全位于该集合内）上的凸函数，则该函数的任何局部极小值必然是其全局最小值。这个性质可以通过以下简单推导说明。

我们知道，如果函数 $\mathcal{L}(\boldsymbol{\theta})$ 满足以下不等式，则该函数为凸函数（Convex Function）：

$$\mathcal{L}((1-\lambda)\boldsymbol{\theta}_1 + \lambda\boldsymbol{\theta}_2) \leq (1-\lambda)\mathcal{L}(\boldsymbol{\theta}_1) + \lambda\mathcal{L}(\boldsymbol{\theta}_2) \tag{6.45}$$

其中 $\boldsymbol{\theta}_1$ 和 $\boldsymbol{\theta}_2$ 为函数定义域内的任意两点，并且 $0 \leq \lambda \leq 1$。若 $\mathcal{L}(\boldsymbol{\theta})$ 是一个局部极小值点，但假设它并非全局极小值点，那么必然存在另一个点 $\boldsymbol{\theta}^*$，使得：

$$\mathcal{L}(\boldsymbol{\theta}^*) < \mathcal{L}(\boldsymbol{\theta}) \tag{6.46}$$

由于定义域是凸集，则对于任意的 $\lambda \in (0,1]$，两点的线性组合 $\bar{\boldsymbol{\theta}} = (1-\lambda)\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^*$ 依然位于定义域内。根据凸函数的定义（公式6.45），我们将 $\bar{\boldsymbol{\theta}}$ 代入函数 $\mathcal{L}(\boldsymbol{\theta})$，可得：

$$\mathcal{L}(\bar{\boldsymbol{\theta}}) = \mathcal{L}((1-\lambda)\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^*) \leq (1-\lambda)\mathcal{L}(\boldsymbol{\theta}) + \lambda\mathcal{L}(\boldsymbol{\theta}^*) \tag{6.47}$$

由于我们假设 $\mathcal{L}(\boldsymbol{\theta}^*) < \mathcal{L}(\boldsymbol{\theta})$，且 $\lambda > 0$，我们可以得到：

$$(1-\lambda)\mathcal{L}(\boldsymbol{\theta}) + \lambda\mathcal{L}(\boldsymbol{\theta}^*) < (1-\lambda)\mathcal{L}(\boldsymbol{\theta}) + \lambda\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) \tag{6.48}$$

结合公式6.47和公式6.48，我们可推得：

$$\mathcal{L}(\bar{\boldsymbol{\theta}}) < \mathcal{L}(\boldsymbol{\theta}) \tag{6.49}$$

另一方面，当 $\lambda \to 0$ 时，点 $\bar{\boldsymbol{\theta}} = (1-\lambda)\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^*$ 会无限趋近于 $\boldsymbol{\theta}$。只要我们取（其中 ϵ 是一个

充分小的正实数)：

$$\lambda < \frac{\epsilon}{\|\boldsymbol{\theta}^* - \boldsymbol{\theta}\|} \tag{6.50}$$

则有：

$$\begin{aligned}
\bar{\boldsymbol{\theta}} &= (1-\lambda)\boldsymbol{\theta} + \lambda\boldsymbol{\theta}^* \Rightarrow \\
\bar{\boldsymbol{\theta}} &= \boldsymbol{\theta} + \lambda(\boldsymbol{\theta}^* - \boldsymbol{\theta}) \Rightarrow \\
\bar{\boldsymbol{\theta}} - \boldsymbol{\theta} &= \lambda(\boldsymbol{\theta}^* - \boldsymbol{\theta}) \Rightarrow \\
\|\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}\| &= \lambda\|\boldsymbol{\theta}^* - \boldsymbol{\theta}\| \Rightarrow \quad \text{（对等式两边同时取范数）} \\
\|\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}\| &< \left(\frac{\epsilon}{\|\boldsymbol{\theta}^* - \boldsymbol{\theta}\|}\right) \cdot \|\boldsymbol{\theta}^* - \boldsymbol{\theta}\| \Rightarrow \quad \text{（使用公式6.50）} \\
\|\bar{\boldsymbol{\theta}} - \boldsymbol{\theta}\| &< \epsilon
\end{aligned}$$

此时，$\bar{\boldsymbol{\theta}}$ 落入 $\boldsymbol{\theta}$ 的 ϵ 领域内。根据局部极小值的定义，则应有 $\mathcal{L}(\bar{\boldsymbol{\theta}}) \geq \mathcal{L}(\boldsymbol{\theta})$，这显然与公式6.49相矛盾。因此，局部极小值不是全局最小值的假设是不成立的。这就证明了：凸函数的任意局部极小值必然是全局最小值。

遗憾的是，含一个或多个隐藏层的神经网络的损失函数关于参数通常是非凸的(Non-convex)。其原因可以通过前述的权重空间对称性得到直观解释。假设存在一组参数配置 $\boldsymbol{\theta}_1$ 使损失函数取得局部极小值。根据权重空间的对称性，我们通过交换隐藏层中第 i 个和第 j 个神经元的所有连接权重（包括输入与输出连接）及偏置，可以得到另一组参数配置 $\boldsymbol{\theta}_2$。由于两者的网络输出完全一致，因此 $\boldsymbol{\theta}_2$ 也是一个具有相同函数值的局部极小值。现在考虑这两组参数的线性组合，例如取它们的平均值 $\bar{\boldsymbol{\theta}} = \frac{1}{2}(\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2)$。若损失函数是关于参数的凸函数，则根据凸函数的定义(公式6.45)，该中点处的函数值应满足 $\mathcal{L}(\bar{\boldsymbol{\theta}}) \leq \mathcal{L}(\boldsymbol{\theta}_1) = \mathcal{L}(\boldsymbol{\theta}_2)$。然而，在神经网络中通常并非如此。参数配置 $\bar{\boldsymbol{\theta}}$ 实际上相当于在保持其他参数不变的情况下，将第 i 个和第 j 个神经元的所有权重和偏置设置为完全相同的值（即都取原参数的平均值)。这种设置会使两个神经元在网络中执行完全相同的计算，从而降低网络的表达能力（原本可以提取两种特征，现在仅能提取一种特征)。其结果是，损失函数在 $\bar{\boldsymbol{\theta}}$ 处的值通常会升高，从而在 $\boldsymbol{\theta}_1$ 和 $\boldsymbol{\theta}_2$ 之间形成一个“波峰”。这种在两个等价局部极小值之间存在较高函数值区域的现象说明损失函数在参数空间中不满足凸性条件。因此，包含单个或多个隐藏层的神经网络，其参数学习问题通常是非凸优化问题。

我们通过一个具体实例来说明神经网络损失函数在参数空间中的非凸性。考虑一个简单的只含两个样本的回归任务：当输入 $x = -1$ 时，目标输出为 0；当输入 $x = 1$ 时，目标输出为 1。为此，我们构建一个包含单个输入神经元、单个输出神经元和两个隐藏层神经元的网络（结构见图6.11)。由于这是回归问题，我们采用平方误差作为损失函数。接下来，我们证明图6.11(a)所示的网络参数设置对应损失函数的一个局部极小值。该两层神经网络可表示为以下函数形式：

$$f(x) = w_{y1} \cdot \sigma(w_{1x} \cdot x + b_1) + w_{y2} \cdot \sigma(w_{2x} \cdot x + b_2) + b_y \tag{6.51}$$

其中 $\sigma(\cdot)$ 表示 Sigmoid 激活函数。为了简化后续分析，我们忽略参数为零的项及其影响，可以

将上述函数进一步简化为：

$$f(x) = w_{y1} \cdot \sigma(w_{1x} \cdot x) \tag{6.52}$$

将参数的具体取值代入，并根据如下平方误差损失函数进行计算：

$$\frac{1}{2}\sum_{i=1}^{2}(f(x_i) - t_i)^2 \tag{6.53}$$

总损失约为 4.5×10^{-5}，其值已非常接近于零。损失函数关于权重参数 w_{1x} 的一阶偏导数为：

$$\frac{\partial \mathcal{L}}{\partial w_{1x}} = \sum_{i=1}^{2} \delta_i \cdot w_{y1} \cdot \sigma'(w_{1x} \cdot x_i)x_i \tag{6.54}$$

其中 $\delta_i = f(x_i) - t_i$ 表示第 i 个样本的输出层误差项。将具体参数数值代入，可得该点处的一阶偏导数约为 -8.9×10^{-5}，也已趋近于零。为了进一步分析该极值点的性质，我们对公式6.54中的权重参数 w_{1x} 再次求偏导，得到如下二阶偏导数：

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial w_{1x}^2} &= \sum_{i=1}^{2}\left(\frac{\partial \delta_i}{\partial w_{1x}} \cdot w_{y1} \cdot \sigma'(w_{1x} \cdot x_i)x_i + \delta_i \cdot w_{y1} \cdot \sigma''(w_{1x} \cdot x_i)x_i^2\right) \\
&= \sum_{i=1}^{2}\left(w_{y1}^2 \cdot \sigma'(w_{1x} \cdot x_i)^2 x_i^2 + \delta_i \cdot w_{y1} \cdot \sigma''(w_{1x} \cdot x_i)x_i^2\right) \\
&\approx \sum_{i=1}^{2}\left(w_{y1}^2 \cdot \sigma'(w_{1x} \cdot x_i)^2 x_i^2\right) \quad （由于 \delta_i \approx 0）\\
&> 0
\end{aligned} \tag{6.55}$$

我们对公式6.54中的权重参数 w_{y1} 求偏导，可得：

$$\frac{\partial^2 \mathcal{L}}{\partial w_{1x} \partial w_{y1}} = \sum_{i=1}^{2} \delta_i \cdot \sigma'(w_{1x} \cdot x_i)x_i \approx 0 \quad （由于 \delta_i \approx 0） \tag{6.56}$$

损失函数关于权重参数 w_{y1} 的一阶偏导数为：

$$\frac{\partial \mathcal{L}}{\partial w_{y1}} = \sum_{i=1}^{2} \delta_i \cdot \sigma(w_{1x} \cdot x_i) \tag{6.57}$$

对上式关于权重参数 w_{y1} 再次求偏导，可得：

$$\frac{\partial^2 \mathcal{L}}{\partial w_{y1}^2} = \sum_{i=1}^{2} \frac{\partial \delta_i}{\partial w_{y1}} \cdot \sigma(w_{1x} \cdot x_i) = \sum_{i=1}^{2} \sigma(w_{1x} \cdot x_i)^2 > 0 \tag{6.58}$$

对公式6.57中的权重参数 w_{1x} 求偏导数，可得：

$$\begin{aligned}
\frac{\partial^2 \mathcal{L}}{\partial w_{y1} \partial w_{1x}} &= \sum_{i=1}^{2}\left(\frac{\partial \delta_i}{\partial w_{1x}} \cdot \sigma(w_{1x} \cdot x_i) + \delta_i \cdot \sigma'(w_{1x} \cdot x_i)x_i\right) \\
&\approx \sum_{i=1}^{2}\left(w_{y1} \cdot \sigma'(w_{1x} \cdot x_i)\sigma(w_{1x} \cdot x_i)x_i\right) \quad （由于 \delta_i \approx 0）\\
&\approx 6.56 \times 10^{-3} \quad （代入具体数值计算）
\end{aligned} \tag{6.59}$$

综合公式6.55、6.56、6.58和6.59的计算结果，可得该神经网络参数的海森矩阵：

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 \mathcal{L}}{\partial w_{1x}^2} & \frac{\partial^2 \mathcal{L}}{\partial w_{1x} \partial w_{y1}} \\ \frac{\partial^2 \mathcal{L}}{\partial w_{y1} \partial w_{1x}} & \frac{\partial^2 \mathcal{L}}{\partial w_{y1}^2} \end{bmatrix} = \begin{bmatrix} \mathrm{H}_{11} > 0 & \mathrm{H}_{12} \approx 0 \\ \mathrm{H}_{21} \approx 6.56 \times 10^{-3} & \mathrm{H}_{22} > 0 \end{bmatrix}$$

此矩阵的所有顺序主子式（即一阶主子式 H_{11} 与二阶主子式 $\mathrm{H}_{11} \times \mathrm{H}_{22} - \mathrm{H}_{21} \times \mathrm{H}_{12}$）均大于零，因此该海森矩阵是正定的。这证明了该组参数配置对应于损失函数的一个局部极小值点。

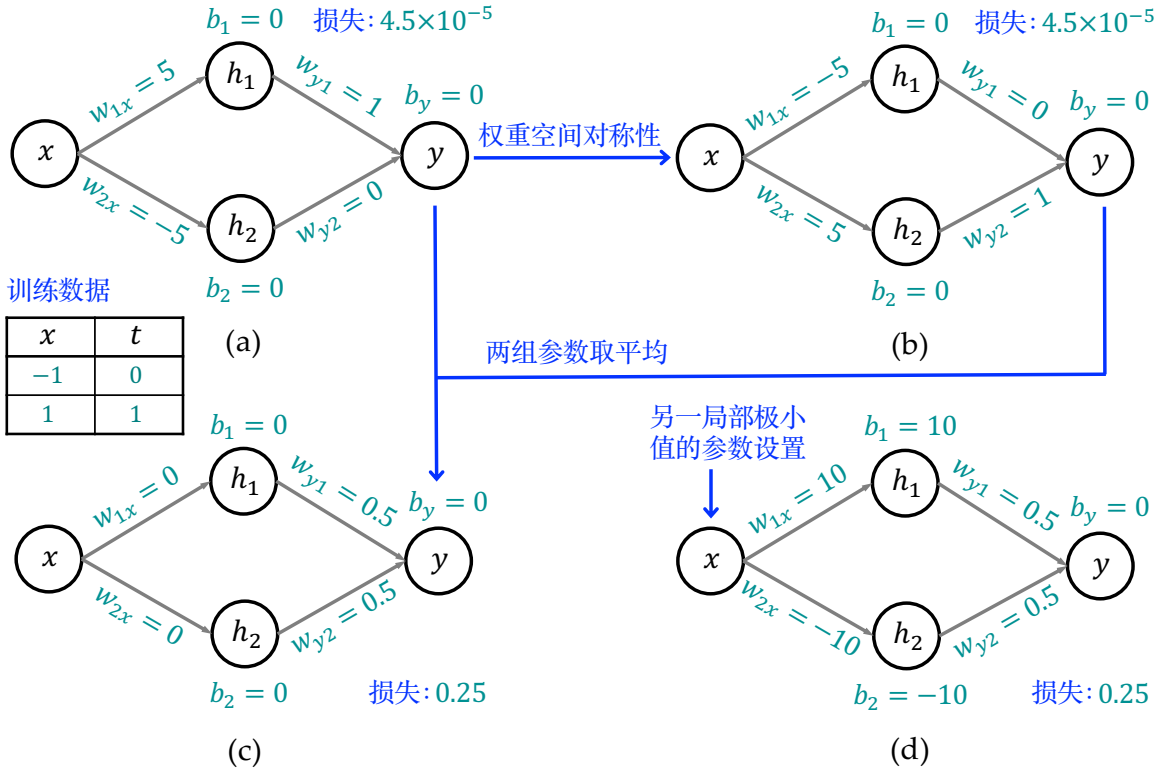


图 6.11: 神经网络损失函数非凸性与部分局部极小值非全局最优的实例说明。(a) 局部极小值点：对应一组特定的网络参数配置，其损失函数值极小。(b) 等价极小值点：利用权重空间对称性，通过置换 (a) 中两个隐藏层神经元的权重和偏置得到另一参数配置。两组配置的损失函数值完全相等。(c) 非凸性验证：将 (a) 与 (b) 的参数进行线性组合（即取均值）得到的参数配置。该点的损失值显著升高，表明两点之间存在“波峰”，从而证明了损失函数的非凸性。(d) 次优局部极小值：另一组参数配置，其海森矩阵可以证明为正定，确认为局部极小值点。但由于其损失值远高于 (a)，说明该点并非全局最优解。

如图6.11(b) 所示，我们可以利用权重空间对称性，通过交换隐藏层神经元 h_1 与 h_2 的所有输入权重、输出权重及其偏置，构造出另一个具有完全相同损失函数值的网络参数配置。现将图6.11(a) 与 (b) 所示的两组参数进行线性组合（此处简单取算术平均），得到如图6.11(c) 所示的网络参数取值。经计算，该网络的损失函数值为 0.25。这一结果表明，参数均值处的损失函数值显著大于两组参数损失函数值的线性组合，显然违背了凸函数的定义（见公式 6.45）。因此，该实例说明神经网络的损失函数在参数空间中是非凸的。此外，对于相同的网络结构，我们还可以构造出如图6.11(d) 所示的另一组参数配置。尽管该参数配置的损失值同样为 0.25，但可以

证明其海森矩阵是正定的，即该参数配置是另一个局部极小值点。结合权重空间对称性，我们同样可以构造出与其等价的另一个局部极小值。这一实例进一步说明，神经网络通常存在大量局部极小值，并且这些参数配置往往并非全局最小值。

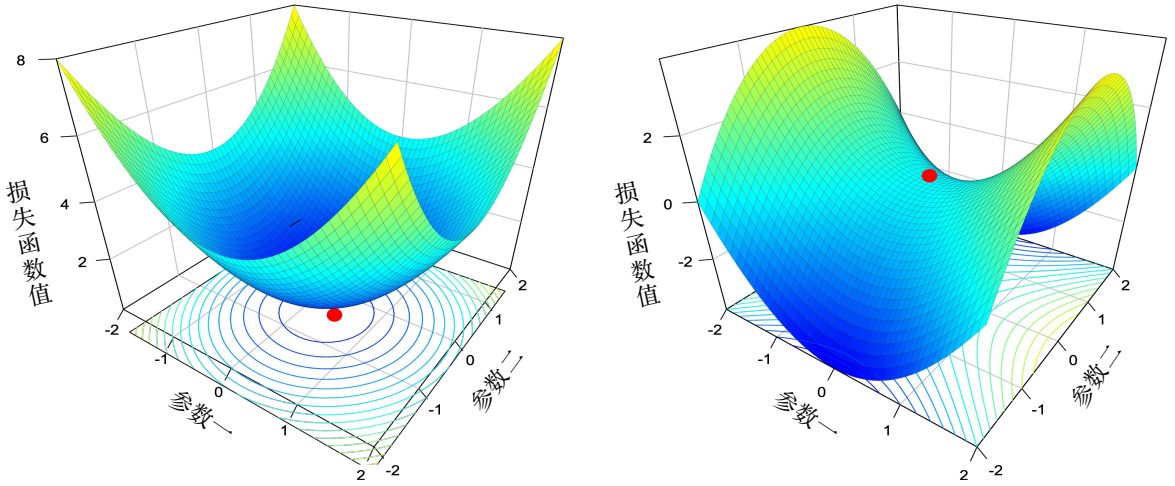


图 6.12: 神经网络参数空间中的损失函数曲面示意图。(a) 局部极小值点：红点表示被相对平滑区域包围的局部极小值。在该点附近，参数沿任意方向的微小调整都会导致损失函数值上升。(b) 鞍点（红色）：该点在某些参数维度上处于极小值，而在另一些维度上则处于极大值。具体表现为：沿某一特定方向调整参数可以降低损失函数值，而沿另一方向调整则会导致损失值增加。

图6.12展示了两幅神经网络参数空间中损失函数曲面的示意图。其中图6.12(a) 描绘了一个理想化的局部极小值点（以红点表示），该点被一个相对平滑的区域包围。在学习率设置合理的前提下，优化算法能够平稳地收敛至该极小值。然而在实际应用中，损失曲面往往并不如此平滑。一种常见情况是曲面呈“悬崖”状，即梯度范数可能变得非常大，从而引发梯度“爆炸”（Exploding Gradient）。与“悬崖”结构相对的另一种极端情形是平坦区域（Plateau）。当损失曲面过于平坦时，梯度范数趋近于零，从而引发梯度消失（Vanishing Gradient）现象，使得参数更新几乎陷入停滞。图6.12(b) 则展示了损失曲面中的一个鞍点（Saddle Point），也以红色表示。在该点处，损失函数在某些参数维度上处于极小值，而在另一些维度上则处于极大值。具体表现为：沿特定方向调整参数可以降低损失函数值，而沿另一方向调整则会导致损失值增加。在数学上，鞍点是一类梯度为零的驻点（Stationary Point），若不同维度的梯度分量因接近零而失去更新的动力，模型便会“困陷”于此点。在实践中，采用小批量随机梯度下降法是缓解上述问题的有效手段之一。由于小批量样本在梯度估计中引入了统计噪声，这种随机扰动有助于模型在训练过程中“挣脱”平坦区间的束缚，并利用噪声带来的位移寻找下降方向，从而有效“逃离”鞍点。

训练神经网络最常用的反向传播算法，本质上是利用损失函数对参数的一阶梯度来确定参数的调整方向。在实际训练中，我们很少直接采用泰勒展开中的二阶近似（如牛顿法）来加速收敛，其主要原因有二：首先，现代神经网络的参数规模通常非常庞大，计算并存储海森矩阵的计算复杂度与内存开销过高；其次，即便在局部区域内海森矩阵是正定的，受损失函数非凸

性影响，二阶优化算法也仅能保证加速收敛至距离初始点最近的局部极小值，而无法确保该点是全局最小值，亦或具备良好的泛化性能。

由于神经网络损失函数在参数空间中表现出高度的非凸性，并且存在大量的局部极小值，反向传播算法在理论上只能保证收敛至某一驻点或局部极小值。在实际应用中，尽管我们通常难以确定、也并不追求收敛至全局极小值。然而反向传播算法对参数初始化具有较强的敏感性。网络参数的初始取值在很大程度上决定了优化过程的演化路径，即最终落入损失曲面上的哪个局部极小值，从而影响了模型最终的泛化性能。

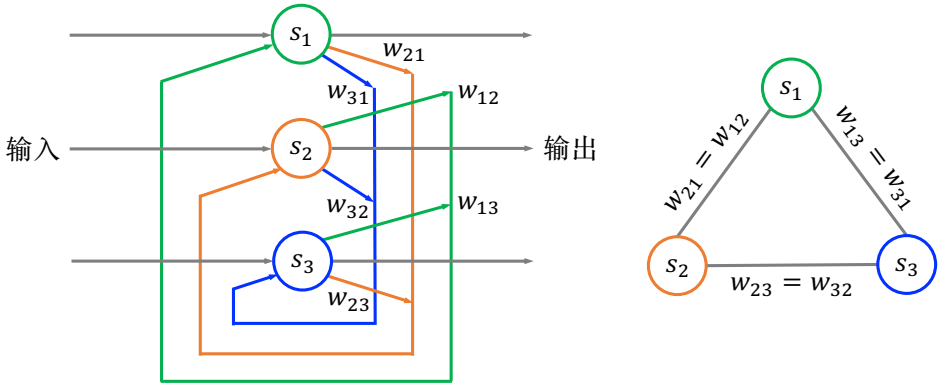
1992 年，伯恩哈德 · 波塞（Bernhard Boser）等学者在美国加州大学伯克利分校提出了支持向量机 [9]。支持向量机在预测精度上通常能够达到与当时神经网络相当的水平。由于其优化目标函数具备凸性，使其训练过程在计算上更加稳定且易于实现（详见第4.4介绍）。这一进展引发了随后十年对核方法（详见第4章）的广泛研究。转机出现在 2002 年，2018 年图灵奖及 2024 年诺贝尔物理学奖得主、英国裔加拿大计算机学家和心理学家杰弗里 · 埃弗里斯特 · 辛顿 (Geoffrey Everest Hinton）提出了对比散度（Contrastive Divergence）算法，为深层神经网络的训练提供了一种有效途径。该算法通过逐层无监督预训练的方式对网络参数进行初始化（详见第7章），在一定程度上缓解了多层网络模型难以训练和优化的问题。这一进展重新激发了研究者对神经网络的兴趣，并推动了深度学习（Deep Learning）的发展。

6.6 霍普菲尔德网络

1982 年，美国科学家约翰 · 霍普菲尔德（John Joseph Hopfield）提出了联想记忆（Associative Memory）网络，即后来广为人知的霍普菲尔德网络（Hopfield Network）[34]。该网络模型将统计物理方法引入神经网络研究，是早期神经网络理论的重要奠基性工作之一。霍普菲尔德网络的核心目标是模拟生物系统的联想记忆功能，使网络能够根据给定的输入模式，从其内部存储中检索出对应的记忆模式。理想的联想记忆系统应具备良好的鲁棒性，即当输入模式存在缺失、模糊或受到噪声干扰时，网络仍能够恢复出正确的原始模式。霍普菲尔德网络在一定程度上具备这种鲁棒的联想记忆能力。因其在人工神经网络领域的开创性贡献，霍普菲尔德与杰弗里 · 辛顿共同获得了 2024 年诺贝尔物理学奖。从技术演进的角度来看，霍普菲尔德网络所体现的能量最小化思想对后续研究产生了深远影响。例如，辛顿提出的受限玻尔兹曼机（Restricted Boltzmann Machine）继承并发展了这一思想，并作为深度神经网络中逐层无监督预训练的重要组成模块。

霍普菲尔德网络可以看作是一种单层且具有侧向连接的网络，可以用图6.13(a) 所示的有向图表示（以三个神经元组成的网络为例）。每个神经元会接收来自其他所有神经元的输入，但不接收自身的输入。由于权重具有对称性，即第 i 个神经元到第 j 个神经元的权重 w_{ji} 等于第 j 个神经元到第 i 个神经元的权重 w_{ij}，该网络还可以用图6.13(b) 所示的无向图进行简化表示。从拓扑结构上看，霍普菲尔德网络由于引入了同层神经元之间的侧向连接，并不属于前馈神经网络，而是被归为循环神经网络（Recurrent Neural Network）。在给定初始输入后，网络会进入动态演化过程。某一神经元的输出会通过侧向连接影响其他神经元的输入，进而反作用于自身，因此

每个神经元的状态会随时间不断更新。只有当所有神经元的状态不再随时间变化时，网络才达到稳态，此时的输出即为给定输入下的最终结果。



(a) (b)

图 6.13: 由三个神经元组成的霍普菲尔德网络。(a) 采用有向图表示的网络拓扑结构。每个神经元接收来自其他神经元的输入（不包括自身）。(b) 由于权重的对称性，网络还可用无向图进行简化表示。

经典霍普菲尔德网络中的每个神经元通常仅具有两种离散状态，可用 $\{-1,1\}$ 和 $\{0,1\}$ 来表示。在本节中，我们采用 $\{-1,1\}$ 的编码方式来展开讨论。在设定网络各神经元的初始状态（即输入模式）后，第 i 个神经元的状态 s_i 按以下公式进行更新：

$$s_i = \sigma\left(\sum_{j=1}^{N} w_{ij}s_j + b_i\right) \tag{6.60}$$

其中 $w_{jj} = 0$（即神经元不存在自反馈连接），而 b_i 为第 i 个神经元的偏置。需要说明的是，在霍普菲尔德网络的经典文献中，研究者通常更倾向于使用阈值这一概念（阈值为偏置的相反数）。为了保持本书前后符号的一致性，此处仍沿用偏置 b_i 来表示阈值。公式中的 $\sigma(\cdot)$ 为如下形式的阶跃函数：

$$\sigma(z) = \begin{cases} +1 & \text{若 } z \geq 0 \\ -1 & \text{若 } z < 0 \end{cases} \tag{6.61}$$

由于选择采用 $\{-1,1\}$ 的编码方式，公式6.60中的偏置（即阈值的相反数）可以简单地设为 0。若采用 $\{0,1\}$ 的编码方案，为了达到等价的判定效果，偏置通常需设为 -0.5。将偏置设为零可以简化计算，这也是本节采用 $\{-1,1\}$ 编码而非 $\{0,1\}$ 编码的原因之一。

下面以一个由 5 个神经元所组成的霍普菲尔德网络为例，说明该网络如何从受干扰的输入模式中恢复出与之最相近的已存储原始模式。假设该网络存储了以下两个模式：

$$\boldsymbol{\xi}^1 = (1,-1,1,-1,1)^\top$$
$$\boldsymbol{\xi}^2 = (-1,1,-1,1,-1)^\top$$

存储以上两个模式的权重矩阵为：

$$\mathbf{W}=\begin{bmatrix}0 & -0.4 & 0.4 & -0.4 & 0.4\\ -0.4 & 0 & -0.4 & 0.4 & -0.4\\ 0.4 & -0.4 & 0 & -0.4 & 0.4\\ -0.4 & 0.4 & -0.4 & 0 & -0.4\\ 0.4 & -0.4 & 0.4 & 0.4 & 0\end{bmatrix} \tag{6.62}$$

其中权重矩阵 $\mathbf{W}$ 的第 i 行、第 j 列元素 w_{ij} 表示第 j 个神经元指向第 i 个神经元的连接权重。关于网络权重的具体学习方法将在后文中详述。若将模式 $\boldsymbol{\xi}^1$ 作为初始输入，则网络的状态更新过程如下（所有偏置均为零）：

$$\boldsymbol{s}=\sigma\left(\mathbf{W}\boldsymbol{\xi}^1\right)=\sigma\left(\begin{bmatrix}0 & -0.4 & 0.4 & -0.4 & 0.4\\ -0.4 & 0 & -0.4 & 0.4 & -0.4\\ 0.4 & -0.4 & 0 & -0.4 & 0.4\\ -0.4 & 0.4 & -0.4 & 0 & -0.4\\ 0.4 & -0.4 & 0.4 & 0.4 & 0\end{bmatrix}\begin{bmatrix}1\\-1\\1\\-1\\1\end{bmatrix}\right)=\sigma\begin{bmatrix}1.6\\-1.6\\1.6\\-1.6\\1.6\end{bmatrix}=\begin{bmatrix}1\\-1\\1\\-1\\1\end{bmatrix} \tag{6.63}$$

即经过一次迭代，网络即可收敛到稳定状态，而该稳定状态正是存储的模式 $\boldsymbol{\xi}^1$。同理，若以 $\boldsymbol{\xi}^2$ 作为输入，则 $\boldsymbol{s}=\sigma\left(\mathbf{W}\boldsymbol{\xi}^2\right)=\boldsymbol{\xi}^2$。若输入模式为存在扰动的 $\boldsymbol{\xi}^3=(1,-1,-1,-1,1)^\top$，经过迭代（本例中为一次迭代，理论上可能需要多次），网络将收敛到与该输入最相近的存储模式 $\boldsymbol{\xi}^1$。若输入模式改为 $\boldsymbol{\xi}^4=(-1,1,1,1,-1)^\top$，则迭代后网络将收敛到与之最相近的模式 $\boldsymbol{\xi}^2$。很容易看出，输入 $\boldsymbol{\xi}^3$ 与模式 $\boldsymbol{\xi}^1$ 更为相似，而输入 $\boldsymbol{\xi}^4$ 与模式 $\boldsymbol{\xi}^2$ 更为接近。上述例子表明，通过合理设置霍普菲尔德网络的权重，网络确实能够从带有扰动的输入（即模糊或部分缺失的记忆）中恢复出准确的存储模式（即原始记忆）。这种特性赋予网络一定的纠错能力和联想检索功能。

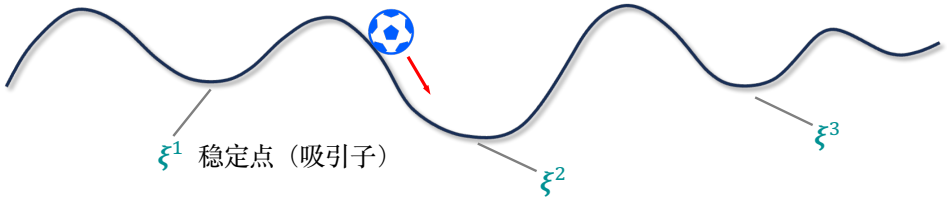


图 6.14: 霍普菲尔德网络能量地貌示意图。当代表网络状态的蓝色小球落入模式 $\boldsymbol{\xi}^2$ 区域内时，网络状态会沿能量曲面下降，最终收敛至极小值点 $\boldsymbol{\xi}^2$。

从本质上讲，霍普菲尔德网络的构建过程是通过设计权重矩阵，将所有待存储的模式向量转化为系统的吸引子（Attractors）。在一些理想化或近似分析中，在数学上相当于将权重矩阵的特征向量近似设置为希望存储的模式。当系统状态在迭代过程中不断与权重矩阵相乘时，状态向量会向对应特征向量方向演化，并最终稳定于该特征方向。在物理意义上，这些稳态对应于系统能量地貌（Energy Landscape）中的局部极小值点。如图6.14所示，当代表网络当前状态的蓝色小球落入模式 $\boldsymbol{\xi}^2$ 的吸引盆（Basin of Attraction）区域时，系统动力学产生的“引力”会驱动状态沿能量曲面不断下降，最终收敛至预设的极小值点 $\boldsymbol{\xi}^2$（系统的吸引子之一）。

霍普菲尔德网络的学习目标是使网络的能量函数最小化，其能量函数的形式受统计力学中伊辛模型（Ising Model）所启发。在物理学中，伊辛模型通过如下哈密顿量 H 来描述磁性物质微观状态的总能量：

$$H = -\sum_{\langle i,j \rangle} J_{ij} s_i s_j - \sum_i h_i s_i \tag{6.64}$$

其中 $s_i \in \{-1, 1\}$ 表示第 i 个格点的自旋状态（分别代表自旋向下或向上）。这些自旋通常排列在特定的晶格结构上。$\langle i, j \rangle$ 表示相邻的格点对，J_{ij} 为相互作用系数（或称耦合系数），而 h_i 则表示作用于第 i 个格点上的外磁场。霍普菲尔德将这一统计物理模型引入神经网络，并定义了如下形式的能量函数：

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} s_i s_j - \sum_{i=1}^{N} b_i s_i \tag{6.65}$$

其中 N 为神经元的总数，w_{ij} 为神经元之间的连接权重，而 b_i 为偏置。公式6.65右侧的第一项体现了赫布理论的核心思想：若两个神经元 s_i 与 s_j 的状态相同（同为激活或同为抑制），增加它们之间的连接权重 w_{ij} 将导致能量函数值下降；反之，若状态不同，减小（或取负值）其连接权重同样会降低总能量。这表明网络倾向于强化状态一致的神经元间的关联。对于公式右侧的第二项：若采用 $s_i \in \{-1, 1\}$ 编码方式，偏置可设为零，此时该项消失。若采用 $s_i \in \{0, 1\}$ 编码方式（其中 1 表示激活状态，0 表示抑制状态），根据前述讨论，偏置 b_i（即阈值的相反数）可以统一设置为 -0.5。总能量的最小化要求公式6.65右侧第二项的值尽可能小。当 $b_i = -0.5$ 时，这一项变为 $0.5 \sum_{i=1}^{N} s_i$。因此，为了降低能量，系统会抑制 s_i 取 1 的数量。从信息处理的角度来看，这实际上施加了一种稀疏性约束，要求模型以尽可能少的激活神经元来表示记忆模式。这种稀疏化表示能够有效降低不同存储模式之间发生重叠和干扰的概率，从而提升网络存储记忆模式的容量（详见下文讨论）。

接下来，我们证明对于公式6.65所定义的能量函数，在神经元状态发生变化时，其能量总是单调递减的。不失一般性，假设在 t 时刻仅有第 i 个神经元的状态 s_i 发生了改变。我们定义第 i 个神经元的状态变化量为：

$$\Delta s_i = s_i^{(t)} - s_i^{(t-1)} \tag{6.66}$$

相应的能量函数变化量 ΔE 可推导如下：

$$\begin{aligned}
\Delta E &= -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} \Delta s_i s_j - b_i \Delta s_i \\
&= -\sum_{i=1}^{N} w_{ij} \Delta s_i s_j - b_i \Delta s_i \quad \text{（由于权重对称性，即 } w_{ij} = w_{ji}\text{）} \\
&= -\underbrace{\left(\sum_{i=1}^{N} w_{ij} s_j + b_j \right)}_{\text{第 } i \text{ 个神经元的净输入}} \Delta s_i
\end{aligned} \tag{6.67}$$

从上式可以看出，若在 t 时刻第 i 神经元的净输入为正，只有当 $s_i^{(t-1)} = -1$ 时，状态才会发生改变（变为 1）。此时 $\Delta s_i = 1 - (-1) = 2$。代入公式6.67可知 $\Delta E < 0$，总能量下降。若净输入为负，只有当 $s_i^{(t-1)} = 1$ 时，状态才会发生改变（变为 -1）。此时 $\Delta s_i = -1 - 1 = -2$。代入公式6.67可得 $\Delta E < 0$，总能量同样下降。

霍普菲尔德网络的权重矩阵 $\mathbf{W}$ 可以按如下公式一次性设置：

$$\mathbf{W} = \frac{1}{N}\sum_{k=1}^{K} \boldsymbol{\xi}^k \left(\boldsymbol{\xi}^k\right)^\top \tag{6.68}$$

其中 N 为神经元的总数，K 为存储模式的数量，而 $\boldsymbol{\xi}_i^k \in \{-1, 1\}^N$ 表示第 k 个存储模式的向量表示。从数学形式上看，权重矩阵 $\mathbf{W}$ 等于所有模式向量自外积之和的均值。当有新模式 $\boldsymbol{\xi}$ 需要加入存储记忆时，只需将 $\frac{1}{N}\boldsymbol{\xi}\boldsymbol{\xi}^\top$ 直接累加至原有的权重矩阵即可。需要注意的是：权重矩阵所用对角线元素均应取零值，即 $w_{ii} = 0$。从公式6.68可以看出，该权重矩阵的计算本质上是赫布规则的应用，即在存储某一模式时，若两个神经元的状态相同（同号），则增加它们之间的连接强度；若状态不同（异号），则降低其连接强度。

下面简要讨论为何霍普菲尔德网络的权重矩阵可以采用上述简单形式进行设置与更新。在接下来的讨论中，我们采用 $s_i \in \{-1, 1\}$ 编码方式，并设所有的偏置项均为零。设网络有 N 个神经元，并且存储了 K 个模式 $\{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^K\}$。为了使这些模式成为系统的稳定吸引子，网络的权重矩阵 $\mathbf{W}$ 需要满足如下条件：即当网络的输入为任一模式 $\boldsymbol{\xi}^k = (\xi_1^k, \ldots, \xi_N^k)$ 时，系统状态应保持不变：

$$\xi_i^k = \sigma\left(\sum_{j\neq i}^{N} w_{ij}\xi_j^k\right) \tag{6.69}$$

根据权重计算公式6.68，权重矩阵的元素 w_{ij} 被设置为：

$$w_{ij} = \frac{1}{N}\sum_{k=1}^{K} \xi_i^k \xi_j^k \quad (i \neq j) \tag{6.70}$$

将上式代入公式6.69，并令 v_i 表示第 i 个神经元的净输入（即 $s_i = \sigma(v_i)$），则有：

$$v_i = \sum_{j\neq i}^{N}\left(\frac{1}{N}\sum_{h=1}^{K} \xi_i^h \xi_j^h\right)\xi_j^k \tag{6.71}$$

我们将求和项中 $h = k$（即当前网络需稳定到的目标模式）与 $h \neq k$（其他模式）的项进行分离，可得：

$$v_i = \underbrace{\frac{1}{N}\sum_{j\neq i}^{N} \xi_i^k \xi_j^k \xi_j^k}_{\text{信号项}} + \underbrace{\frac{1}{N}\sum_{j\neq i}^{N}\sum_{h\neq k}^{K} \xi_i^h \xi_j^h \xi_j^k}_{\text{串扰项}} \tag{6.72}$$

由于 $\xi_j^k \in \{-1, 1\}$，所以 $(\xi_j^k)^2 = 1$。上式中的第一项（即信号项）可以简化为：

$$\frac{N-1}{N}\xi_i^k \tag{6.73}$$

在神经元规模 N 较大时，该信号项近似为 ξ_i^k。若式中第二项（即串扰项）的影响可以忽略或其

值接近于零，则系统状态将稳定于目标模式 $\boldsymbol{\xi}^k$。公式6.72中的串扰项可以改写为如下形式：

$$\sum_{h\neq k}^{K} \xi_i^h \underbrace{\left(\frac{1}{N} \sum_{j\neq i}^{N} \xi_j^h \xi_j^k \right)}_{\boldsymbol{\xi}^h \text{ 与 } \boldsymbol{\xi}^k \text{ 的互相关性}} \tag{6.74}$$

上式括号中的项表示模式 $\boldsymbol{\xi}^h$ 与 $\boldsymbol{\xi}^k$ 之间的互相关性，即两者在特征空间中的相似程度。若所有存储模式彼此正交，即对于任意两个不同的模式 $\boldsymbol{\xi}^h$ 和 $\boldsymbol{\xi}^k$ $(h \neq k)$，均满足内积 $(\boldsymbol{\xi}^h)^\top \boldsymbol{\xi}^k = 0$，则串扰项将完全消失。然而，若模式之间存在较高的相关性，或者存储模式的数量过多，超过了网络的理论存储容量上限，系统能量函数中就会产生伪吸引子（Spurious State），即在原本没有存储模式的位置出现了能量局部极小值点，表现为网络可能会在这些位置上进入稳态，从而形成“虚假记忆”。对于二值霍普菲尔德网络，其理论存储容量上限约为 $0.14N$，其中 N 为网络中神经元的数量（相关证明详见文献 [1]）。这意味着当存储的模式数量 K 超过此临界值时，网络难以有效抑制串扰项的影响，因而更容易被伪吸引子所吸引并收敛到错误的稳态。

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

[9] 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.

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

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

[12] 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.

[13] 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.

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

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

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

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

[18] 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.

[19] 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).

[20] 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).

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

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

[23] 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.

[24] 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.

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

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

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

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

[34] 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.

[35] 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.

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

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

[38] 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.

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

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

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

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

[43] 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.

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

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

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

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

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

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

[50] 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.

[51] 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.

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

[53] 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).

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

[55] 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.

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

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

[58] 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.

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

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

[61] 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).

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

[63] 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.

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

[65] 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).

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

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

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

[69] 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.

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

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

[72] 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.

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

[74] 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.