# 第3章

# 深度学习算法基础

机器学习从大量数据中挖掘信息，学习如何完成任务，是计算机视觉处理任务的关键技术。深度学习是目前机器学习领域最受关注的分支，相比于需要手工特征提取的传统机器学习，深度学习自动从数据中学习复杂特征，并组合成更复杂的特征来完成任务，深度学习可以理解为传统神经网络的拓展。

## 3.1 机器学习

机器学习算法可以从大量数据中总结出潜在的规律，寻找出有用的知识，建立合适的模型，并利用这个模型解决包括计算机视觉在内的各种任务[1]。

根据数据使用情况，机器学习算法通常可以分为有监督学习、无监督学习、半监督学习等几种类型[2]。有监督学习和无监督学习的区别在于数据集有没有真实标签。有监督学习的训练数据集中包含了样本与对应的标签；无监督学习的数据集没有标签，例如聚类学习，需要通过算法将数据集中有相似特征的样本聚合为若干类别，这些不同的类别称为“簇”。半监督学习在实际应用场景中很常见，由于对数据进行标注的成本较高，往往需要付出大量的人力物力，而未标注的数据则较容易获得，半监督学习便是将少量有标注的样本与大量未标注的样本结合使用，学习到数据的特征表达。

与前面三种学习方式不同，强化学习是机器学习的另一个领域，不需要大量数据，而是在与环境不断交互的过程中进行学习。强化学习强调场景中的智能体如何基于环境进行行动，每一步行动都有可能得到奖励或者惩罚，通过与环境不断交互，以取得最大化的预期收益。

在实践应用中，使用最广泛的是有监督学习方法，常用的有监督分类技术可以分为线性、非线性、集成三个不同类别，分别以支持向量机（SVM）、核函数支持向量机、AdaBoost分类器为代表。

## 3.2　神经网络的基本组成

神经网络(Neural Network,NN)是实现机器学习任务的常见方法,主要由数据驱动对生物神经系统进行建模,在机器学习领域取得了出色的效果[3]。人工神经网络又称神经元网络,其基本单元为人工神经元(Artificial Neuron,AN),是学者们从人脑神经元中获得灵感并设计的一种具有简洁数学表示的计算单元。

### 3.2.1　神经元

神经系统中每个神经元是一个可以接收、发射脉冲信号的细胞,如图 3-1 所示。外部刺激通过神经末梢转化为电信号,并通过轴突将该信号进行传递。而这个脉冲信号的强度也决定了该神经元能否被激活,只有激活状态下的神经元才会输出脉冲信号至下一个神经元,非激活的神经元不输出脉冲。无数个这样的神经元构成了神经中枢,神经中枢可以综合各种信号,使得人体对外部刺激做出反应。

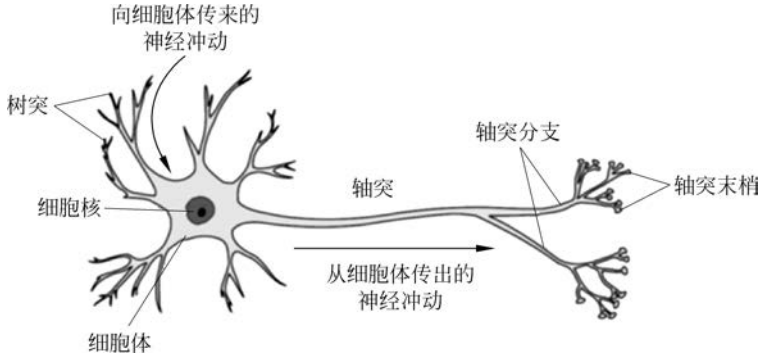



**图 3-1　神经元示意图**[4]

人工神经元的工作机制与生物神经系统中的神经元类似,每个神经元会接收若干输入信号,产生单个输出。对于每一个输入,都有一个权重与其关联,线性组合后再经过一个非线性的激活函数作为输出,由此来模拟生物神经元中激活与非激活的两种状态。如果用 $x_1,x_2,\cdots,x_n$ 表示 $n$ 个输入,用 $y$ 表示输出,则有

$$y=f(w_1x_1+w_2x_2+\cdots+w_nx_n+b) \tag{3-1}$$

其中,$f(\cdot)$表示激活函数,$\{w_1,w_2,\cdots,w_n\}$表示权重系数,$b$ 表示一个与输入无关的偏置量(bias)。

图 3-2 是人工神经元的简要结构,通常也把这样的人工神经元称为感知器(perceptron),其中的模型参数需要通过学习来得到。单神经元模型已经可以用来作为一个简单的决策模型,通过输入信号来决定是否做出响应。但在真实世界中的实际决策模型要更加复杂,这时便需要将多个神经元进行连接,组成神经网络,即多层感知器(Multi-Layer Perceptron,MLP)[5]。

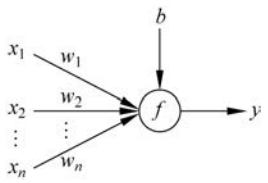



**图 3-2　人工神经元示意图**

### 3.2.2 神经网络的结构

一个完整的神经网络通常由输入层、隐藏层、输出层组成。输入层一般为输入神经网络中的样本信息，其维数决定了输入层的神经元个数。隐藏层指所有在输入层之后输出层之前的层，用于处理一些不对用户展示的中间步骤，通常隐藏层的层数可以根据任务的复杂程度自由设置[5]。当隐藏层层数较多时，就可以称其为深度神经网络[6]。输出层负责产生预测值，例如，当使用神经网络处理分类任务时，输出层的神经元个数为该任务中的类别数，每个神经元的输出值为该样本属于每个类别的概率。

图 3-3 展示了一个神经网络结构，其中输入层包含 4 个输入值，2 层隐藏层分别包含 5 个和 3 个节点，输出层包含 1 个输出值，网络中每个节点视为一个神经元，将神经元按输入层、各隐藏层、输出层进行分布。同时该神经网络模型为最基本的全连接网络，输入层的 4 个节点与第 1 隐藏层的 5 个节点连接，第 1 隐藏层的 5 个节点与第 2 隐藏层的 3 个节点连接，第 2 隐藏层的 3 个节点与输出层的节点连接。像图 3-3 这样层内的神经元互不相连，每层的每个节点都与下一层的全部神经元节点两两连接的网络，称为全连接网络（简称全连接）[7]。

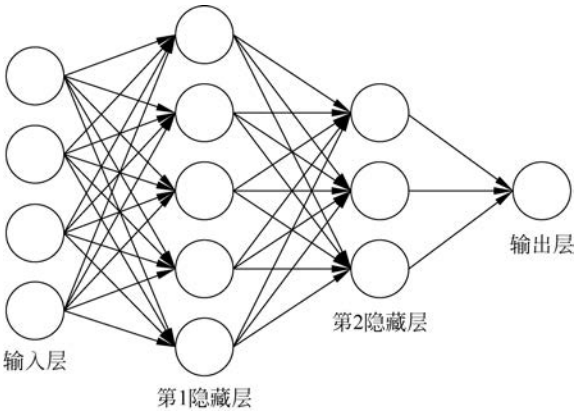



**图 3-3 神经网络的结构**

## 3.3 神经网络的计算

神经网络的计算是网络进行学习训练的过程，与机器学习中基于梯度下降训练的模型类似[8]，也主要包含正向传播、反向传播、梯度下降三部分。

### 3.3.1 激活函数

如上所述，类似生物神经系统的激活机制，神经元在接收其他神经元释放的神经递质时，只有当刺激达到一定强度时才会将信号进行传递[9]。因此，在神经网络中，激活函数（activation function）起到非常重要的作用[10]，是对输入的一种非线性映射操作，也是神经

网络能学习到非线性特征的一个重要原因，没有激活函数的神经网络，只是一些线性映射层的多层堆叠，无论怎么增加层数，也很难从数据中学习到表达能力强的特征。

在传统的神经网络中，常用的激活函数有 Sigmoid 和 Tanh，其中 Sigmoid 是使用较多的激活函数，由如下公式定义：

$$\text{Sigmoid}(x)=\frac{1}{1+e^{-x}}\in(0,1) \tag{3-2}$$

图 3-4 分别给出了 Sigmoid 函数及其导数，Sigmoid 函数单调递增，值域范围限制在(0,1)，这也很形象地描绘了神经元受到激活的情形——原点左侧几乎没有被激活，而右侧能得到较好的激活。此外，可以看出 Sigmoid 函数在左右两侧非常平坦，对应的 Sigmoid 激活函数导数趋于 0，这样在根据链式法则反向传播时，往往会导致梯度趋于 0，容易引起梯度消失的问题。

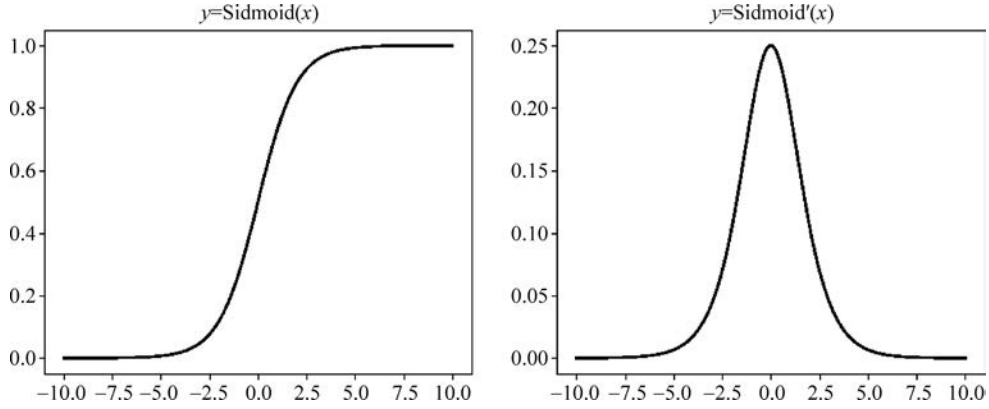



**图 3-4　Sigmoid 函数及其导数**

另一个常用的激活函数为双曲正切函数 Tanh，定义如下：

$$\text{Tanh}(x)=\frac{1-e^{-2x}}{1+e^{-2x}}\in(-1,1) \tag{3-3}$$

Tanh 函数及其导数如图 3-5 所示。与 Sigmoid 相比，该激活函数将输入数据压缩到了−1～1，且均值为 0，但也同样存在梯度消失的问题。

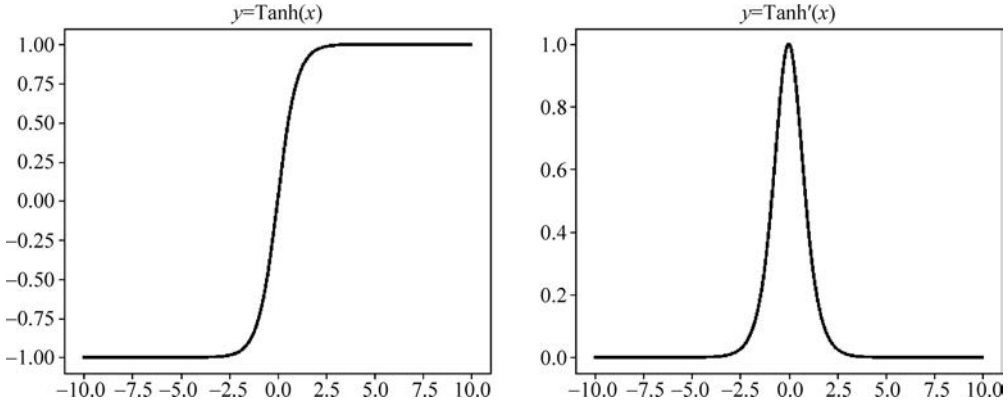



**图 3-5　Tanh 函数及其导数**

### 3.3.2 正向传播

在神经网络训练阶段，神经网络中的数据流动分为正向传播和反向传播两个环节。正向传播是指数据沿神经网络的输入层到输出层的顺序，依次与中间隐藏层的参数进行运算，一直传播到输出的过程。针对每个网络节点（神经元），正向传播过程就是节点在获得输入数据后进行线性变换和激活运算的两步计算过程。3.3.1 节讲述了单个神经元的线性变换和激活运算过程，而对于复杂的深层神经网络模型一般采用向量化运算方法来改进正向传播计算过程。

以图 3-3 的神经网络结构为例，输入层的信息表示为矩阵 $\boldsymbol{X}=(x_1,x_2,\cdots,x_i,\cdots,x_n)$，其中 $n=4$ 表示有 4 个输入，$x_i$ 表示第 $i$ 个输入的具体信息。当把每层对应的权重系数和偏置量也都矩阵化以后，网络的正向传播构成也就转换为向量处理的过程。因此，图 3-3 神经网络第 1 层（由于输入层不计入神经网络的层数，因此神经网络第 1 层即第 1 隐藏层）的中间值矩阵 $\boldsymbol{A}^1$ 可表示为

$$\boldsymbol{A}^1=f^1(\boldsymbol{W}^1\boldsymbol{X}+\boldsymbol{b}^1) \tag{3-4}$$

同理，对于神经网络第 2 层（第 2 隐藏层）的输出 $\boldsymbol{A}^2$，可由网络第 1 层的输出 $\boldsymbol{A}^1$ 经过加权偏移与激活计算得到；而网络第 3 层（输出层）输出 $\hat{\boldsymbol{Y}}$ 可由网络第 2 层的输出 $\boldsymbol{A}^2$ 经过加权偏移与激活计算得到，其过程如图 3-6 所示。

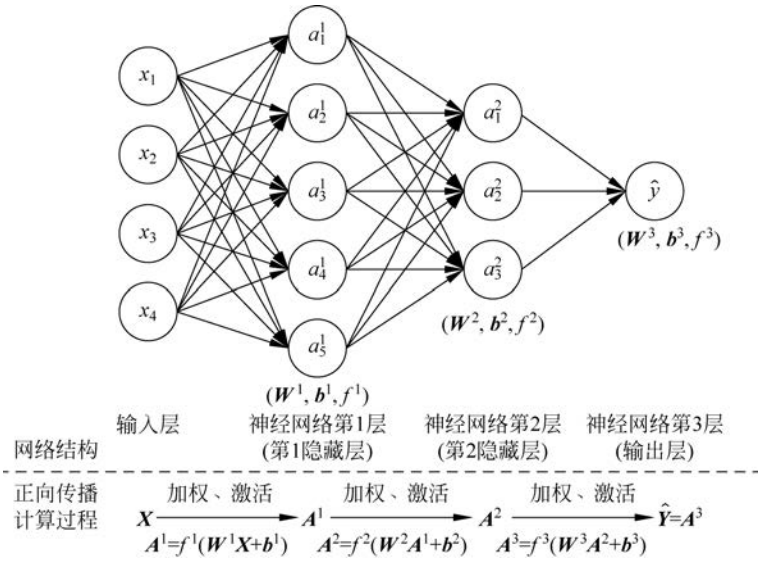



**图 3-6 神经网络的正向传播计算过程图**

### 3.3.3 反向传播

神经网络在正向传播得到输出后，可以计算预测值与实际值的误差，从而根据误差函数相对于网络参数的梯度进行网络学习；而反向传播便是应用链式法则，在神经网络模型中

逐层反向求解损失函数对所有待优化参数的梯度，并一步步更新中间权重和偏置参数。

在反向传播过程中，通常采用 delta 法则，其关键是采用梯度下降法来逐步逼近神经网络各隐藏层的最佳权重参数。假设图 3-6 中神经网络最终的预测输出结果为 $\boldsymbol{Y}$，而经过前向传播得到的实际输出为 $\hat{\boldsymbol{Y}}$，两者存在一定的误差。为了方便理解神经网络的参数学习过程，以均方误差为例计算两者之间的损失。

$$E(\boldsymbol{X},\boldsymbol{W},\boldsymbol{b})=\frac{1}{2}\sum(\boldsymbol{Y}-\hat{\boldsymbol{Y}})^2 \tag{3-5}$$

其中，$E$ 表示均方误差函数，$\boldsymbol{W}$ 和 $\boldsymbol{b}$ 分别表示神经网络各隐藏层的权重和偏置。反向传播的目标是确定一个参数矩阵 $\boldsymbol{W}$ 和偏置矩阵 $\boldsymbol{b}$ 使得损失函数 $E$ 的值足够小。

梯度下降法是从随机初始化位置开始，每次都以较小的步幅向着误差曲面下降最快的方向微调参数，经过多轮迭代更新，直到达到误差最小值点，即全局最优点。在神经网络反向传播训练中，梯度下降法通过计算损失函数 $E$ 对参数 $\boldsymbol{W}$ 和 $\boldsymbol{b}$ 的偏导数来确定误差曲面下降最快的方向。因此，为了使 $E$ 最小化，参数 $\boldsymbol{W}$ 和 $\boldsymbol{b}$ 的修正变化量可以具体表示为式(3-6)和式(3-7)。

$$\Delta\boldsymbol{W}=-\eta\frac{\partial E}{\partial\boldsymbol{W}} \tag{3-6}$$

$$\Delta\boldsymbol{b}=-\eta\frac{\partial E}{\partial\boldsymbol{b}} \tag{3-7}$$

其中，$\eta$ 为神经网络的学习率，表示沿着当前梯度方向下降的步幅，这就是 delta 法则。

在神经网络的训练学习过程中，正向传播与反向传播是交替进行的，反向传播需要在正向传播中得到的中间变量来进行计算，而每次正向传播前都需要反向传播来更新参数。

### 3.3.4　优化算法

当模型和损失函数较简单时，可以通过求解析解的方式来最小化误差；但如果模型或者损失函数较复杂，难以求得解析解，便只能通过一些迭代优化算法来降低损失函数的值，即数值解，在深度学习中，人们也把这些优化算法称为优化器。常用的优化器有很多种，例如批量梯度下降(Batch Gradient Descent，BGD)及其变种、自适应的方法(AdaGrad、Adam等)[11]。每种优化器都有其利弊，需要根据任务选择合适的优化算法。本节主要介绍较为经典的随机梯度下降(Stochastic Gradient Descent，SGD)[12]。

梯度下降使用整个训练集的数据来计算损失函数 $L(w)$ 对模型参数的梯度 $\nabla_w L(w)$，并沿梯度的反方向来更新参数以最小化损失函数。但在一次更新中，都要遍历整个训练集，BGD 的计算量巨大，会造成训练过程缓慢且不稳定。与 BGD 相比，随机梯度下降 SGD[11-12] 每次更新时会对每个样本(或者每个批次的样本)进行梯度更新，可能每次更新并不都是朝着整体最优的方向，但训练速度快，学习率设置合适的前提下仍能较好地收敛。当然，SGD 也有其局限性，例如更新频繁、容易产生振荡、容易收敛到局部最小值等，也有一些工作通过加入动量[13]、加入自适应的方法对 SGD 进行改进。

针对随机梯度下降存在的问题，可以选择使用自适应优化学习率的方法。自适应梯度(Adaptive Gradient，AdaGrad)优化算法对每个不同的参数调整不同的学习率，对于变化比

较频繁的参数使用更小的步长进行更新，而变化较为稀疏的参数则用较大的步长来更新；但如果 AdaGrad 没有在前期找到较优解，后期学习率进一步降低则更难趋向最优解。均方根反向传播(Root Mean Squared Propagation，RMSProp)通过将 AdaGrad 中的梯度积累改变为指数加权移动平均，并结合这个值来调节学习率的变化，使得其能够在不稳定的目标函数下很好地收敛。自适应矩阵估计(Adaptive Moment Estimation，Adam)结合了 AdaGrad 与 RMSProp 两种优化算法的优点，收敛速度更快，同时又综合考虑之前时间的梯度动量，从而计算更新步长。

为了最小化网络的损失函数，学习率是一个非常关键的超参数。这个超参数决定了权重更新的快慢，如果学习率设置较低，则网络训练会十分缓慢，相反，如果学习率设置得较高，则可能会跳出最优解，使网络收敛到不理想的结果。因此，通常希望设置一个较为理想的学习率，来尽可能地减少网络的损失。

在训练神经网络时，一种常见的学习率设置方法是在初始时使用较大的学习率，然后在后期将学习率减小。但由于在刚开始训练时，网络模型的权重是随机初始化的，此时使用较大的学习率可能就会导致模型的不稳定振荡。于是，有人提出了 Warmup 预热学习率的方式，如图 3-7 所示，最开始训练的几个 Epoch 使用较小的学习率(learning rate)，然后使学习率慢慢增大，直到模型逐渐趋于稳定后达到预先设置的学习率进行训练，此后学习率再慢慢衰减，这样可以使得模型的收敛速度更快，得到的效果更佳。

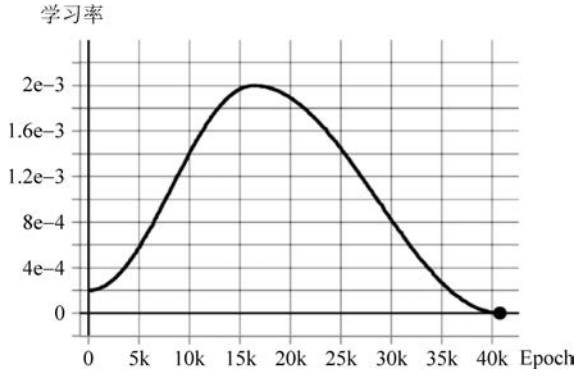



**图 3-7　Warmup 的学习率策略**

## 3.4　卷积神经网络的基本组成

卷积神经网络(Convolutional Neural Network，CNN)是指在网络结构中用卷积运算来代替一般的矩阵乘法实现加权操作的神经网络，是深度学习中一类非常重要的神经网络结构，也是一种广泛应用在计算机视觉领域的网络类型[14]。在结构上，卷积神经网络一般由卷积层、池化层、全连接层组成[15]。

### 3.4.1　卷积层

卷积层是卷积神经网络的核心组成部分，也是卷积神经网络能够提取非线性特征的重

要依据，它由一系列参数可学习的卷积核(kernel)集合而成，其中卷积核也称为滤波器(filter)。对于每个卷积层，都存在一个卷积核，如果将卷积核矩阵中对应的数值视为传统神经网络中的权重系数，则卷积层的正向传播可通过神经元权重与输入数据的卷积计算实现。

如图 3-8 所示，在计算机视觉处理领域，输入数据一般为输入的图像，而卷积核通常为一个正方形矩阵，通常也被称作滤波器或者卷积模板。将卷积核在输入图像上沿高度或者宽度滑动，输入图像在卷积窗口内的每个元素与卷积核上对应位置的元素相乘求和，便得到了该窗口内的卷积输出，这个输出就是该局部邻域内的特征。

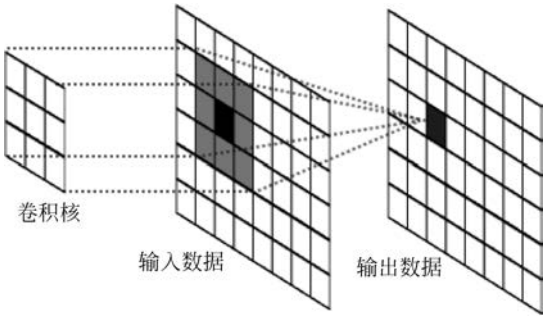



**图 3-8　卷积过程示意图**

使用一个卷积核遍历整幅图像时，卷积后的输出与原图的尺寸是不同的。这主要是因为图像最边缘的像素点无法与卷积核的中心重合，无法映射到输出的特征图，导致输出的特征图尺寸相比较原图会缩小。同样地，也应该考虑卷积核在输入图像上滑动的准则，不同准则下得到的输出尺寸也有差异。为了避免卷积操作之后使图像尺寸变小的问题，常在图像的外围进行填充操作(padding)。

填充指在原始图像的四周填充一些元素，通常填充值为 0，填充 0 的行或者列数由卷积核的大小决定。例如，当卷积核大小为 3×3 时，只需要在输入图像的四周填充 1 行或 1 列 0 元素，就能保证卷积后的输出与原图的尺寸一致。

如图 3-9 所示，填充后的卷积将卷积核从图像的左上角开始滑动，卷积核每次滑动 1 个像素，这个滑动的像素步长称为步幅，可以人为设定。当步幅为 1 时，卷积核每完成一次卷积将向右移动一个像素(1 列)；完成该行的卷积后，卷积核返回最左端并向下平移一个像素(1 行)。有时为了满足某些特定的任务或需求，也可以将宽和高上的步幅分别设置。

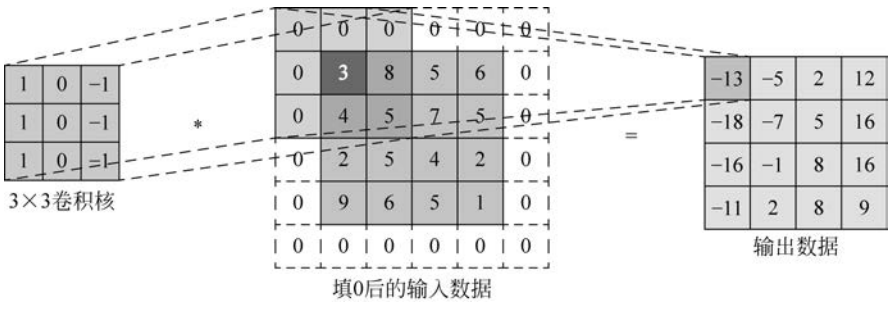
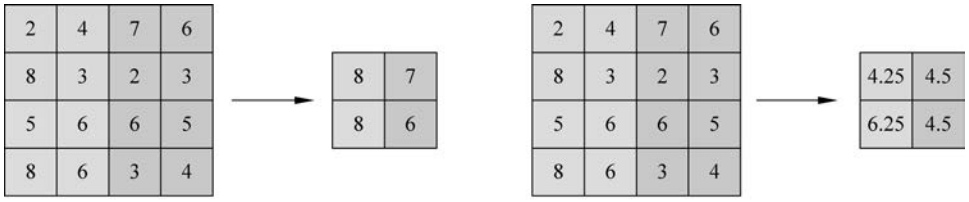



**图 3-9　填充后的卷积过程**

### 3.4.2 池化层

池化往往在卷积操作之后出现，其最直观的功能是缩小特征图的尺寸，这样一方面可以压缩参数量，简化网络的计算；另一方面还能聚合特征，使网络提取到不同尺度的信息。

池化是将某一位置的输出用该位置相邻输出的某种总体统计特征替换的操作。通常用池化窗口（一般为 2×2）对相邻区域进行限定，这一点与卷积类似，只是不像卷积核一样需要优化参数。与卷积核类似，池化窗口每次滑动的步长也可根据步幅来确定。常用的池化有最大池化与平均池化，最大池化便是在池化窗口中取最大值，平均池化则是在池化窗口中取平均值，图 3-10 为最大池化与平均池化的示例。




(a) 最大池化　　(b) 平均池化

**图 3-10　池化过程的计算（见彩插）**

### 3.4.3 ReLU 激活函数

激活函数可以给神经网络加入非线性因素，以提升线性模型的特征表达能力，是神经网络的重要组成部分。早期的神经网络普遍采用 3.3.1 节中所介绍的 Sigmoid 和 Tanh 激活函数。如前所述，Sigmoid 激活函数在正负饱和区的梯度都趋于 0，容易引起梯度弥散问题；而 Sigmoid 和 Tanh 激活函数都包含指数子项计算，使得计算难度较大。因此，为了解决上述问题，修正线性单元（Rectified Linear Unit，ReLU，又称线性整流函数）代替了传统的激活函数。

近几年，ReLU 激活函数已经成为深度学习领域中使用较多的激活函数，它的表达式如下：

$$\mathrm{ReLU}(x)=\begin{cases}x, & x\geqslant 0\\ 0, & x<0\end{cases}\in[0,+\infty) \tag{3-8}$$

如图 3-11 所示，ReLU 激活函数的优点是在激活区域的导数为恒定的非 0 值，缓解了梯度消失问题的同时也能加快网络的收敛速度。但由于原点左侧的导数恒为 0，在梯度反传时流经该神经元后梯度就都变成了 0，之后的权重就无法得到更新，造成了“死亡神经元”的问题。针对这个问题，一些其他激活函数被提出[10]，但仍不妨碍 ReLU 使深度学习往前迈进了一大步。

### 3.4.4 全连接层

在神经网络中，全连接是指某一层的节点都与其上一层的全部节点连接。在卷积神经

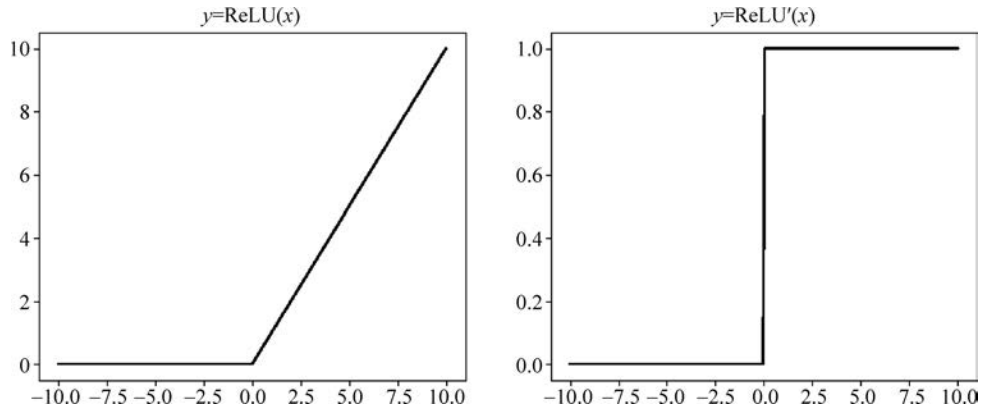



**图 3-11　ReLU 激活函数及其导数**

网络中，常说的全连接层（Fully Connected Layers，FC 层）往往位于卷积神经网络的末端，其作用是将前面卷积层产生的特征图展开成一个一维向量，即输入图像包含高级语义信息的特征向量，在整个卷积神经网络中起到“分类器”的作用。例如，对于分类任务，往往在多个全连接层后得到一个维数等于类别数的特征向量，该向量的每一维即为某个类别的概率。

## 3.5　深度学习模型的训练技巧

深度学习网络模型通常是一种非线性的神经网络模型，其所采用的损失函数是一个非凸函数。在模型迭代优化过程中，最小化损失函数在本质上可以看作一种非凸优化问题，因此会存在许多局部最优解。当深层神经网络进行梯度反传时，损失误差在经过每一层的传递都会不断衰减，有可能出现梯度消失问题。深度学习网络模型一般具有很大的参数量，也为模型的优化训练带来巨大挑战。同时，深度学习网络模型的训练往往依赖数据样本，样本的数量与多样性会直接影响模型最终的性能。如果训练样本太少或者网络模型过于复杂，则容易出现过拟合现象，导致模型鲁棒性和泛化能力变差。

为了克服深度学习网络模型在训练过程中难以优化的问题，通常会引入多种训练优化技巧来分别解决模型计算量大、过拟合、梯度弥散、参数量大等问题[14]。下面介绍几种在训练中常用的技巧方法，包括归一化（normalization）、丢弃法（dropout）、权重衰减（weight decay）以及参数初始化（weights initialization）等。

### 3.5.1　归一化

为了加快深度网络模型的收敛速度，同时缓解网络中梯度弥散问题，归一化处理是深度神经网络训练中一个非常重要的技巧，使得深层网络模型的训练更加容易和稳定。在百万量级数据的大规模神经网络训练中，常采用计算训练集中的批量（batch）数据模式。数据在送入网络时，都是成批输入的，但如果不同批次数据分布差异较大或训练集与测试集数据分布差异较大，就会导致神经网络的性能降低，变得难以训练或产生过拟合现象。因此，针对每一批数据的归一化处理目前已几乎成为所有卷积神经网络的通用技巧。

在深度学习中，常见的归一化方法有批归一化（Batch Normalization，BN）[16]、层归一化（Layer Normalization，LN）[17]、实例归一化（Instance Normalization，IN）[18]、组归一化（Group Normalization，GN）[19]等，如图 3-12 所示。

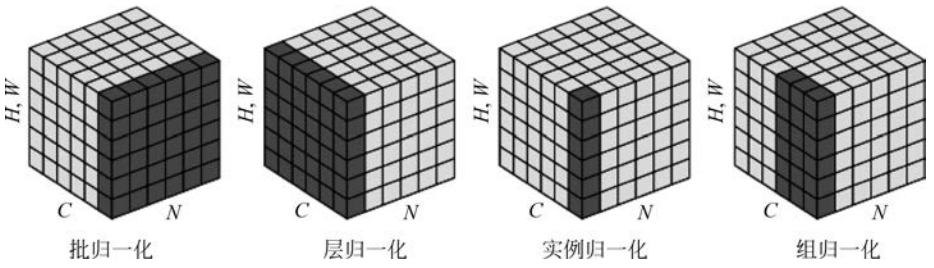



**图 3-12　常见的归一化方法[19]**

批归一化（BN）使得训练较深的神经网络成为可能，并且使网络可以更好更迅速地收敛[16]。批归一化操作一般在卷积层之后、激活函数之前，将输入的一个批次特征图中的每个通道求解均值和方差，即对每个通道在这一批样本中做归一化操作。这样会使得整个神经网络在不同层的数值都相对稳定，也可以减少训练时梯度爆炸等情况的发生。

值得注意的一点是，在使用批归一化训练时，往往希望批大小尽可能大一些，这样可以让一个批次内的均值和方差更准确；但在测试时，单个样本的输出不应该依赖某个批次的分布，这时可以使用整个训练集的均值和方差来处理测试集的每个批次，具体来说，对于测试样本归一化时的均值，可以通过直接计算训练集中所有批数据均值的平均值来求得，对于方差则使用训练集中每个批数据方差的无偏估计。

层归一化（LN）与 batch 无关，而是在特征图的通道、长与宽的维度上进行标准化，每个样本都计算独立的均值和方差[17]。层归一化不依赖批大小的特点使得其适合处理序列化的数据，例如自然语言处理中的循环神经网络，但其在卷积神经网络中的表现不如批归一化等方法。

实例归一化（IN）将统计范围进一步缩小至单个通道的特征图，在特征图的每一通道上计算均值和方差，而与批大小和特征图的通道数都无关[18]。

组归一化（GN）的统计方式介于层归一化与实例归一化之间，将某一特征图的不同通道分为多个组，然后对每个组进行归一化[19]。其也可以避免批大小对训练的影响，在计算机视觉任务中有着不错的表现。

总之，BN、LN、IN、GN 这四种归一化方式只是区分在统计数据维度选择上的不同，而它们的计算过程基本一致，主要分为计算数据均值、计算数据方差、去均值方差处理将数据归一化到 0 均值 1 方差分布上、变化重构出网络所需要学到的分布。

### 3.5.2　丢弃法

深度学习网络模型在训练过程中容易出现过拟合问题，当模型参数较多、训练样本较少时，训练得到的模型往往会过度拟合数据，从而导致在训练集和验证集上的预测准确率很高，但在测试集上表现却很差。丢弃法（dropout）是深度学习网络中一种常用的抑制过拟合

方法，其做法是在模型训练过程中，随机选择一些隐藏层神经元并暂时丢弃，每次前向传播和梯度反传时只有输入层、输出层以及其余部分隐藏层神经元被激活。在整个迭代优化过程中选择的丢弃神经元都是随机的，即每个隐藏层神经元权重都有一定的概率不进行更新。

图 3-13 是 dropout 的示意图，由于 dropout 在每次迭代优化时按照一定概率选择部分神经元进行激活，将另一部分神经元从网络中暂时丢弃，所以能有效降低网络模型的计算量，并且有效缓解网络模型的过拟合，达到一定的正则化效果。

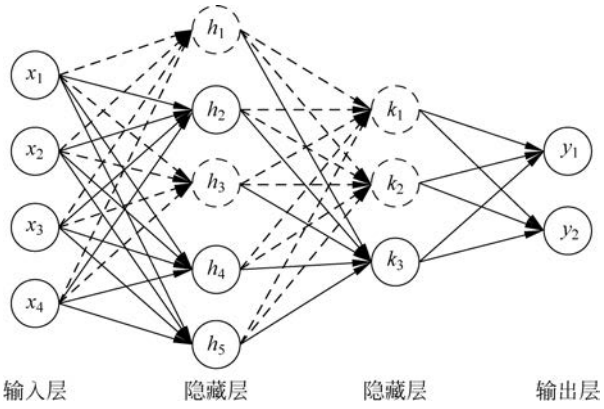



**图 3-13　dropout 示意图**

### 3.5.3　权重衰减

在训练网络的过程中，一般通过观察误差函数来判断模型训练是否成功。有时误差函数已经降到很低，但用这个训练好的网络测试其他数据却得到较差的表现，产生过拟合现象。为了在反向传播过程中减小模型过拟合现象，权重衰减是深度学习网络模型训练中一种常用的方法，又称为 $L_2$ 范数正则化。

在优化模型参数时，往往通过最小化目标函数或者损失函数来实现，即 $\min\limits_{w} L(w;X,y)$，其中 $w$ 为待优化的参数，$X$、$y$ 分别为训练样本与对应的标签。为了让模型能够适应不同的数据集，避免过拟合，往往会希望权重尽可能小，于是考虑加入正则项 $\Omega(w)=\frac{1}{2}\|w\|_2^2$ 作为参数范数惩罚减小权重。添加正则项后的目标函数变为

$$L_{\text{new}}(w;X,y)=L(w;X,y)+\alpha\Omega(w) \tag{3-9}$$

其中 $\alpha$ 是调节惩罚项权重的参数，该超参数通常是一个大于零的常数，值越大表示在损失函数中所占比重越大，则模型学到的权重参数会越接近 0。如式(3-9)所示，在原来损失函数的基础上增加正则项作为惩罚项，从而对模型的权重参数进行约束。该惩罚项是由预先设定的一个超参数作为权重衰减项的系数，使得梯度多了一个 $w$ 的一次项，于是在更新参数时都会有一个常数衰减 $w \leftarrow (1-\eta\alpha)w-\eta\nabla_w L_{\text{new}}(w;X,y)$，其中 $\eta$ 为学习率，$\nabla_w L_{\text{new}}(\cdot)$ 为加了正则化约束的目标函数对参数 $w$ 的梯度。由此可见，在计算完损失进行梯度反传时，正则项会使得权重参数先乘以一个小于 1 的数，然后减去不包含正则项的梯度。所以，模型的权重参数能够在迭代优化中不断衰减，有效地降低模型过拟合的可能性。

### 3.5.4 参数初始化

参数初始化是深度学习网络模型在开始训练之前需要完成的一个关键过程，直接影响网络模型能否高效且精准地收敛。为了便于理解，假设模型所有的隐藏层都采用相同的激活函数，并且对模型所有参数进行全零初始化，那么在反向传播的时候，每个隐藏层神经元的参数会计算得到相同的梯度，在参数迭代更新之后每个神经元的参数依旧是相同的，相当于每个隐藏层只有一个神经元起作用，这显然不合理。

因此，通常采用高斯分布或者均匀分布等方式对模型参数进行随机初始化，使每个神经元具有不同的初始参数，以保证在反向传播时获得不同的梯度，从而使网络模型参数快速准确地收敛到全局最优值。

## 3.6 本章小结

深度学习作为具有多级表示的表征学习方法，与传统机器学习方法相比，深度学习更加复杂，能够学习到更加抽象的模式和特征，也因此在计算机视觉领域得到了广泛的应用。本章从机器学习开始，依次介绍了人工神经网络和卷积神经网络的基础算法和原理、深度学习模型的优化计算技巧等内容。随着更多精度更高、速度更快、性能更强大的深度学习网络的出现，深度学习算法与模型的设计和优化必将在计算机视觉领域发挥更加重要的作用。

## 参考文献

[1] CZUM J M. Dive into deep learning[J]. Journal of the American College of Radiology, 2020, 17(5).

[2] 周志华. 机器学习[M]. 北京：清华大学出版社. 2016.

[3] Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks[J]. arXiv preprint arXiv:1312.6199, 2013.

[4] 邱锡鹏. 神经网络与深度学习[M]. 北京：机械工业出版社. 2020.

[5] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[6] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.

[7] Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

[8] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[9] Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.

[10] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[11] Kingma D P, Ba J. ADAM: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[12] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic

optimization[J]. Journal of machine learning research,2011,12(7): 2121-2159.

[13] Qian N. On the momentum term in gradient descent learning algorithms[J]. Neural networks,1999,12(1): 145-151.

[14] Howard A G,Zhu M,Chen B,et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861,2017.

[15] Tan M,Le Q. Efficientnet: Rethinking model scaling for convolutional neural networks[C]// International conference on machine learning. PMLR,2019: 6105-6114.

[16] Ioffe S,Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International conference on machine learning. PMLR,2015: 448-456.

[17] Ba J L,Kiros J R,Hinton G E. Layer normalization[J]. arXiv preprint arXiv:1607.06450,2016.

[18] Ulyanov D, Vedaldi A, Lempitsky V. Instance normalization: The missing ingredient for fast stylization[J]. arXiv preprint arXiv:1607.08022,2016.

[19] Wu Y,He K. Group normalization[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 3-19.