

深度学习

教学课程组

2021年

- 参考教材：吴飞，《人工智能导论：模型与算法》，高等教育出版社
- 在线课程(MOOC)：<https://www.icourse163.org/course/ZJU-1003377027>
- 在线实训平台（智海-Mo）：https://mo.zju.edu.cn/classroom/class/zju_ai_2021
- 在线共享资源（智海在线）：<http://www.wiscean.cn/online/intro/zju-01>

提纲

一、深度学习历史发展

二、前馈神经网络

三、卷积神经网络

四、循环神经网络

五、深度生成学习

六、深度学习应用

深度学习的历史发展

1943 年，神经科学家 Warren McCulloch 和逻辑学家 Walter Pitts 合作提出了“McCulloch-Pitts (MCP) neuron”的思想。MCP 可对输入信号线性加权组合，再用符号函数来输出线性加权组合结果，以模拟大脑复杂活动模式。MCP 是最早的神经网络雏形。

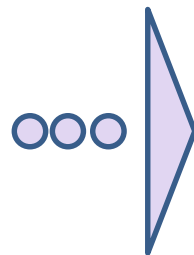
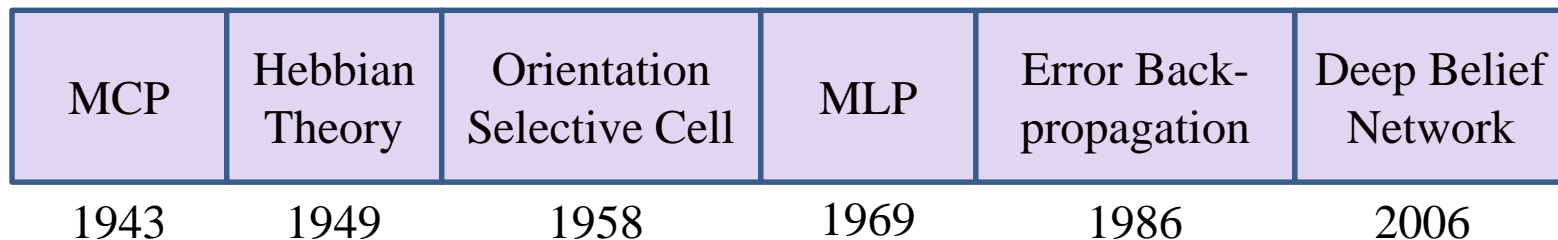
“我们在科学史上第一次知道了‘我们是怎么知道的’ (for the first time in the history of science, we know how we know)”

赫布理论(Hebbian theory): 神经元之间持续重复经验刺激可导致突触传递效能增加。

Neurons that fire together, wire together.

“神经元之间突触的强弱变化是学习与记忆的生理学基础”这一理论为联结主义人工智能研究提供了认知神经心理学基础。

1958 年，David Hubel 和 Torsten Wiesel 在实验中发现小猫后脑皮层中不同视觉神经元与瞳孔所受刺激之间存在某种对应关系，由此发现了一种被称为“方向选择性细胞 (orientation selective cell)”的神经元细胞，从而揭示了“视觉系统信息分层处理”这一机制。



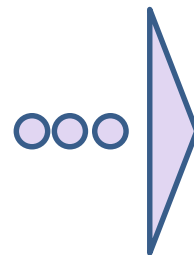
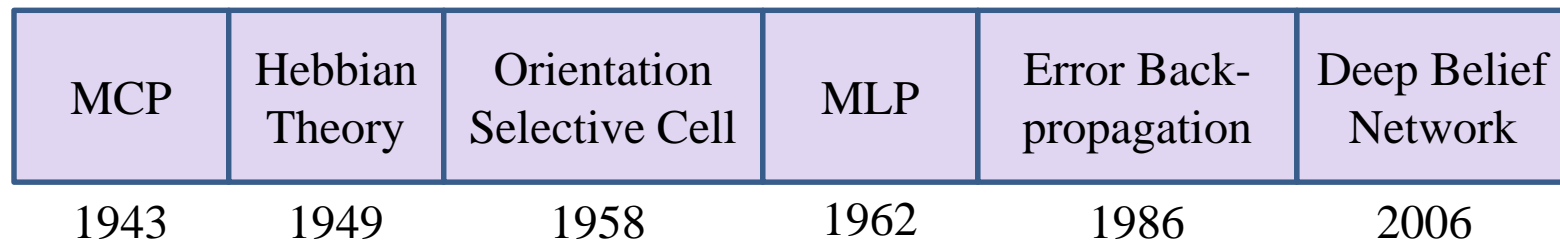
深度学习的历史发展

神经网络研究的突破来自于Frank Rosenblatt在20世纪50年代所提出的“感知机(perceptron)”模型。

由于感知机中没有包含非线性变换操作的隐藏层，因此感知机表达能力较弱（如无法解决异或问题）。

最早由Werbos提出[Werbos 1974] [Werbos 1990]、并且由Rumelhar和Hinton等人[Rumelhart 1986]完善的误差后向传播(error backpropagation)算法解决了多层感知机中参数优化这一难题。

2006年，Hinton在《Science》等期刊上发表了论文，首次提出了“深度信念网络(deep belief network)”模型[Hinton 2006]，在相关分类任务上可取得性能超过了传统浅层学习模型（如支持向量机），使得深度架构引起了大家的关注。



提纲

一、深度学习的历史发展

二、前馈神经网络

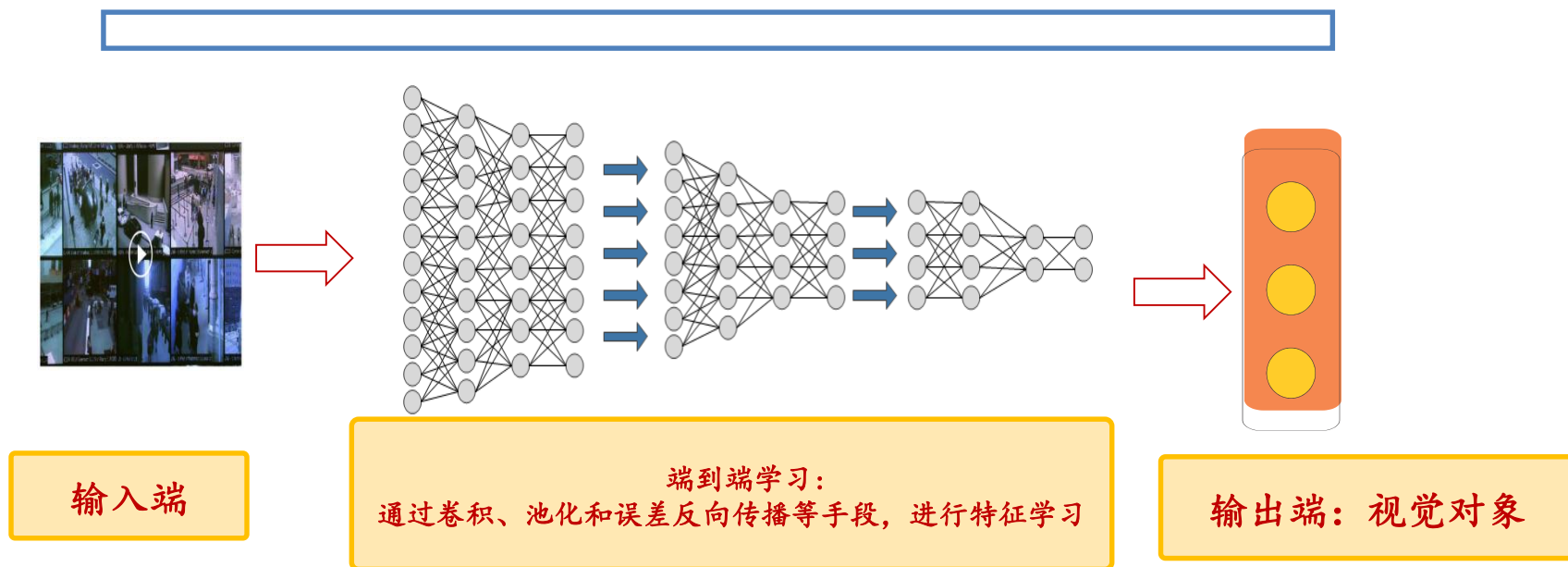
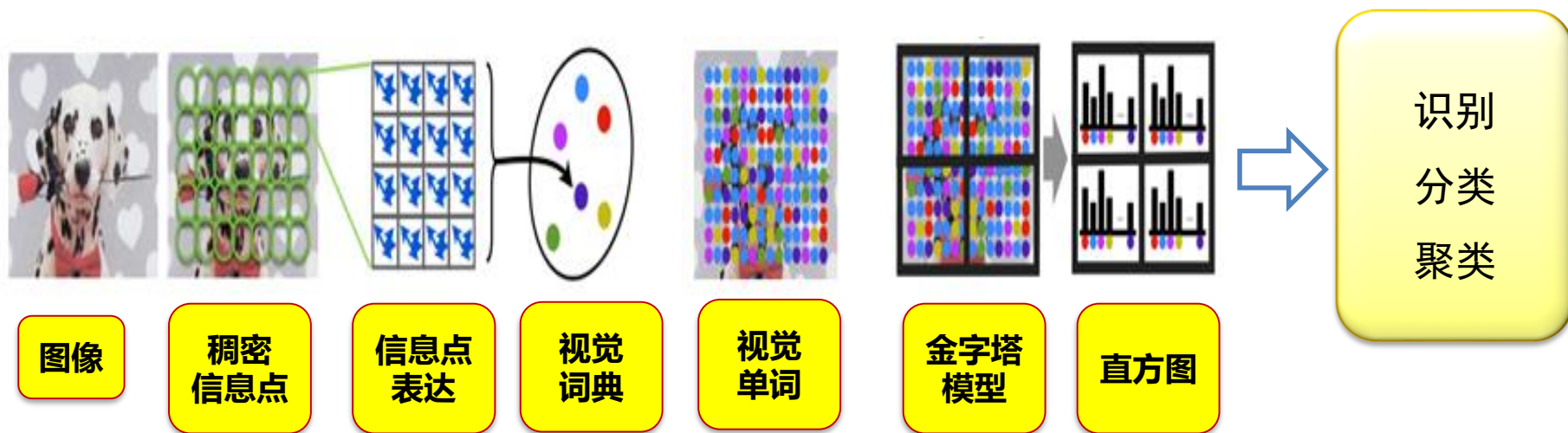
三、卷积神经网络

四、循环神经网络

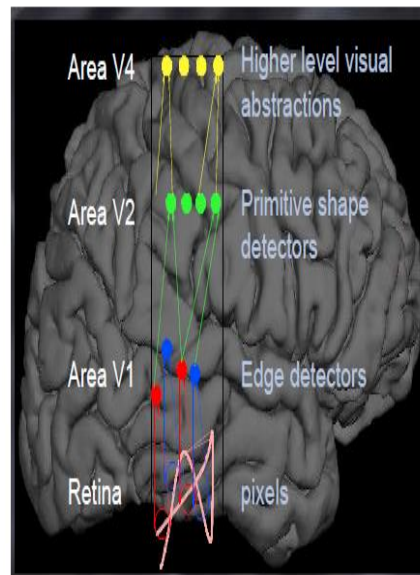
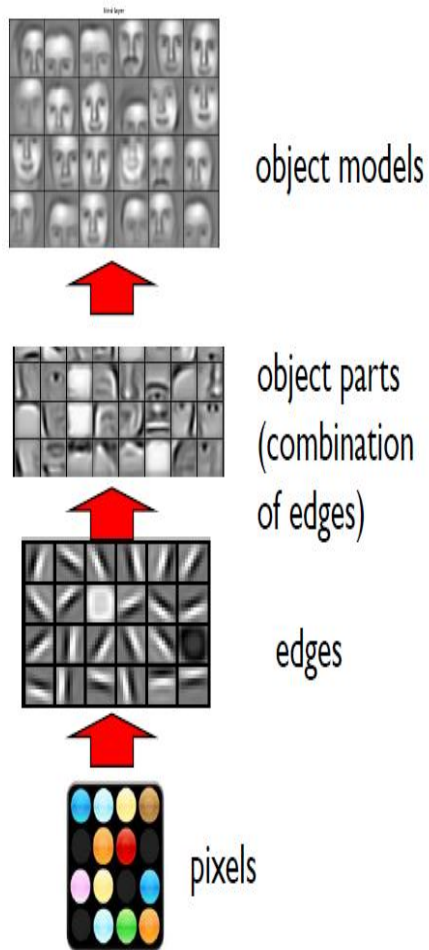
五、深度生成学习

六、深度学习应用

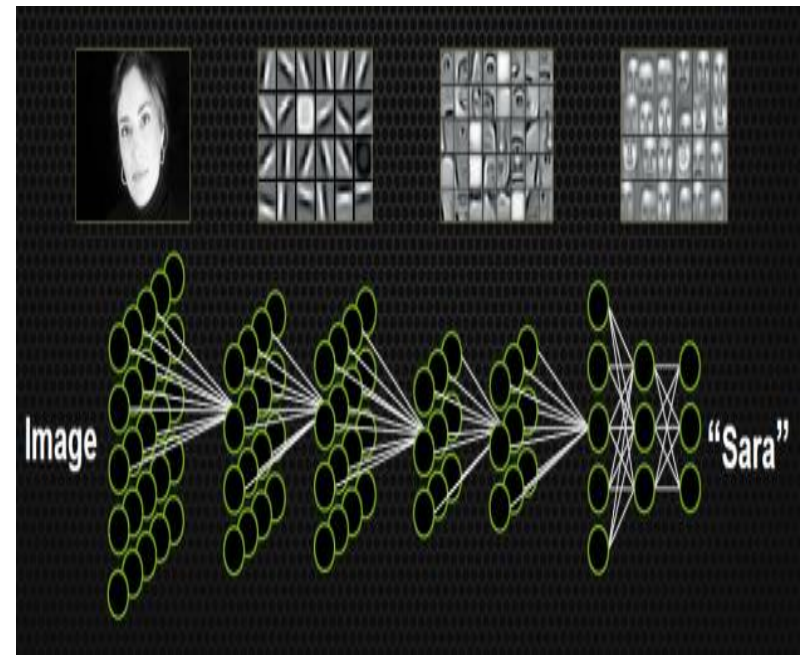
浅层学习 Versus 深度学习：从分段学习到端到端学习



深度学习：以端到端的方式逐层抽象、逐层学习



Slide credit: Andrew Ng



- 深度学习所得模型可视为一个复杂函数
- 非线性变换与映射的过程：像素点→语义

神经元

在生物学中，神经元细胞有兴奋与抑制两种状态。大多数神经元细胞在正常情况下处于抑制状态，一旦某个神经元受到刺激并且电位超过一定的阈值后，这个神经元细胞就被激活，处于兴奋状态，并向其他神经元传递信息。基于神经元细胞的结构特性与传递信息方式，神经科学家Warren McCulloch和逻辑学家Walter Pitts合作提出了“McCulloch–Pitts (MCP) neuron”模型[McCulloch 1943]。在人工神经网络中，MCP模型成为人工神经网络中的最基本结构。MCP模型结构如图6.1所示。

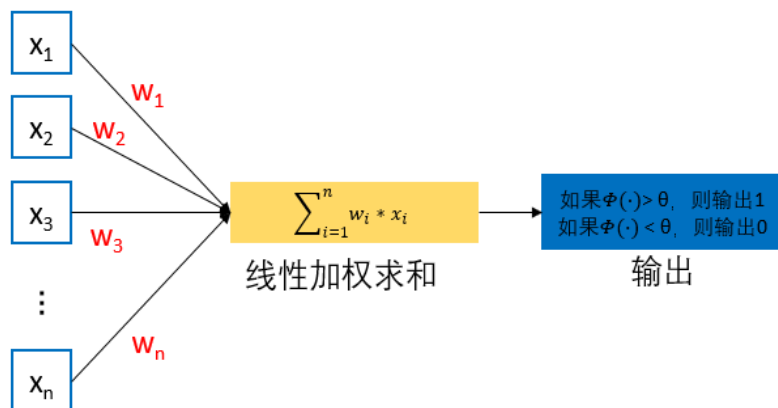


图6.1 MCP模型结构

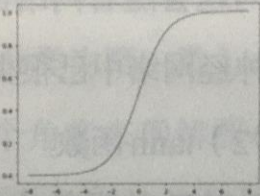
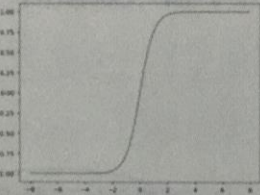
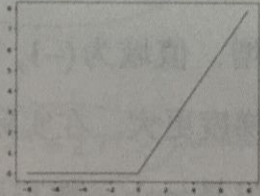
给定 n 个二值化（0或1）的输入数据 $x_i(1 \leq i \leq n)$ 与连接参数 $w_i(1 \leq i \leq n)$ ，MCP神经元模型对输入数据线性加权求和，然后使用函数 $\Phi(\cdot)$ 将加权累加结果映射为0或1，以完成两类分类的任务：

$$y = \Phi\left(\sum_{i=1}^n w_i x_i\right)$$

其中 w_i 为预先设定的连接权重值（一般在0和1中一个值或者1和-1中取一个值），用来表示其所对应输入数据对输出结果的影响（即权重）。 $\Phi(\cdot)$ 将输入端数据与连接权重所得线性加权累加结果与预先设定阈值 θ 进行比较，根据比较结果输出1或0。

常用的激活函数：对输入信息进行非线性变换

表6.1 激活函数基本性质

名称	函数	图像	导数	值域
sigmoid	$f(x) = \frac{1}{1+e^{-x}}$		$f'(x) = f(x) * (1 - f(x))$	$(0, 1)$
tanh	$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$		$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{cases}$		$f'(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ 1, & \text{for } x > 0 \end{cases}$	$[0, +\infty)$

神经网络使用非线性函数作为激活函数（activation function），通过对多个非线性函数进行组合，来实现对输入信息的非线性变换

常用的激活函数：softmax函数

Softmax函数一般用于多分类问题中，其将输入数据 x_i 映射到第 i 个类别的概率 y_i 如下计算：

$$y_i = \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}}$$

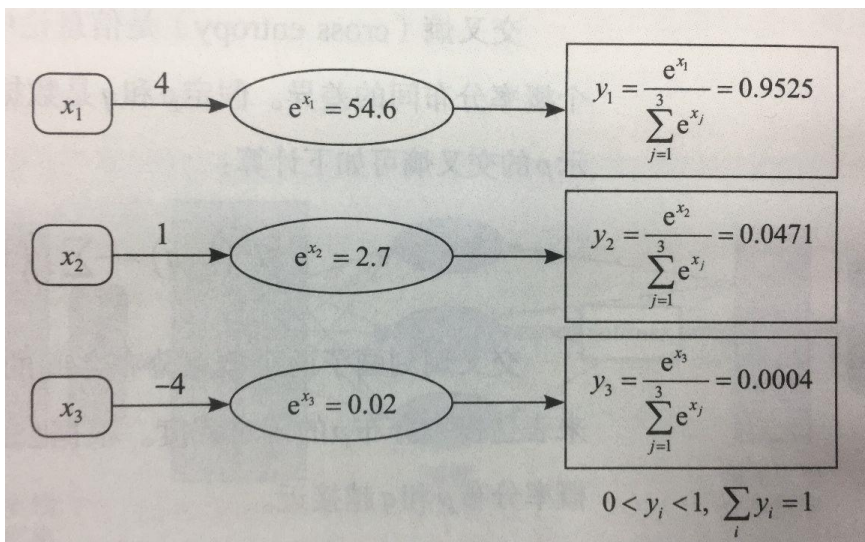


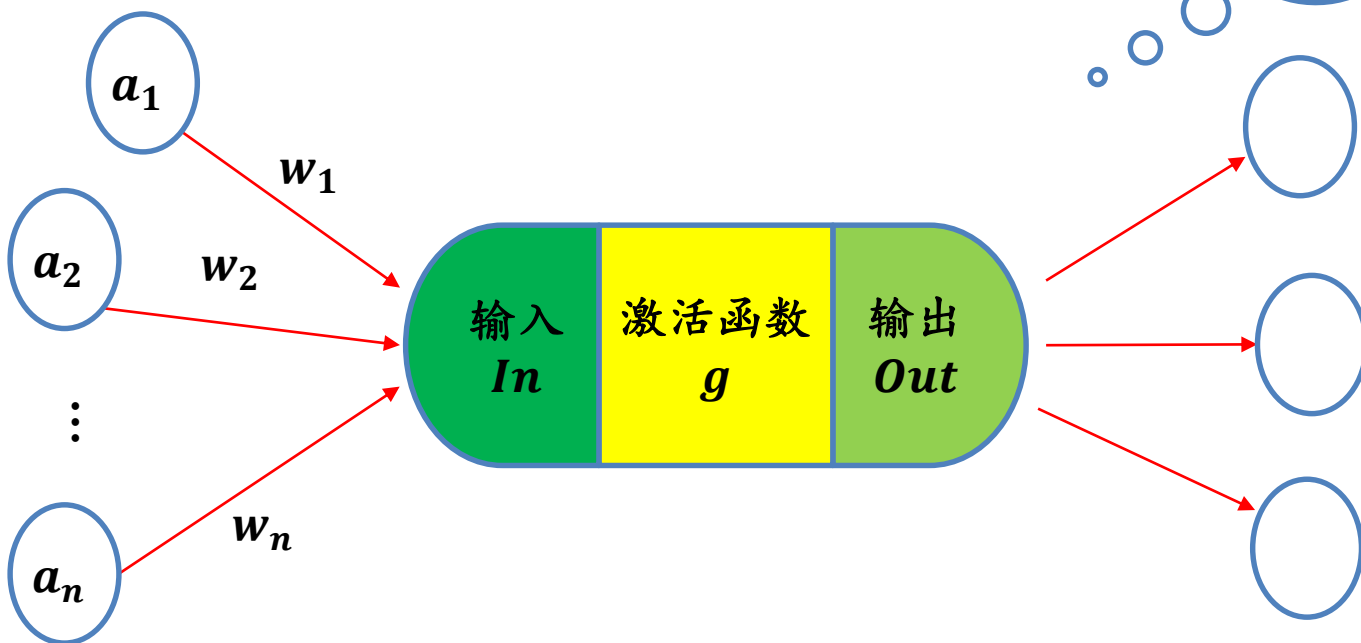
图6.2 三分类问题的 softmax 输出示意

- 对于取值为4、1和-4的 x_1 、 x_2 和 x_3 ，通过softmax变换后，将其映射到(0,1)之间的概率值。
- 由于Softmax输出结果的值累加起来为1，因此可将输出概率最大的作为分类目标。

神经元

神经元是深度学习模型中基本单位，可以如下刻画神经元功能：

1. 对相邻前向神经元输入信息进行加权累加： $In = \sum_{i=1}^n w_i * a_i$
2. 对累加结果进行非线性变换（通过激活函数）： $g(x)$
3. 神经元的输出： $Out = g(In)$



损失函数

损失函数 (Loss Function) 又称为代价函数 (Cost Function)，用来计算模型预测值与真实值之间的误差。损失函数是神经网络设计中的一个重要组成部分。通过定义与任务相关的良好损失函数，在训练过程中可根据损失函数来计算神经网络的误差大小，进而优化神经网络参数。

两种最常用损失函数：

- 均方误差损失函数
- 交叉熵损失函数。

常用的损失函数

均方误差损失函数

均方误差损失函数通过计算预测值和实际值之间距离（即误差）的平方来衡量模型优劣。假设有 n 个训练数据 x_i ，每个训练数据 x_i 的真实输出为 y_i ，模型对 x_i 的预测值为 \hat{y}_i 。该模型在 n 个训练数据下所产生的均方误差损失可定义如下：

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

常用的损失函数

交叉熵损失函数

交叉熵（cross entropy）是信息论中的重要概念，主要用来度量两个概率分布间的差异。假定 p 和 q 是数据 x 的两个概率分布，通过 q 来表示 p 的交叉熵可如下计算：

$$H(p, q) = - \sum_x p(x) * \log q(x)$$

交叉熵刻画了两个概率分布之间的距离，旨在描绘通过概率分布 q 来表达概率分布 p 的困难程度。根据公式不难理解，交叉熵越小，两个概率分布 p 和 q 越接近。

常用的损失函数

交叉熵损失函数

假设数据 x 属于类别1。记数据 x 的类别分布概率为 y ，显然 $y = (1,0,0)$ 代表数据 x 的实际类别分布概率。记 \hat{y} 代表模型预测所得类别分布概率。

那么对于数据 x 而言，其实际类别分布概率 y 和模型预测类别分布概率 \hat{y} 的交叉熵损失函数定义为：

$$\text{cross entropy} = -y \times \log(\hat{y})$$

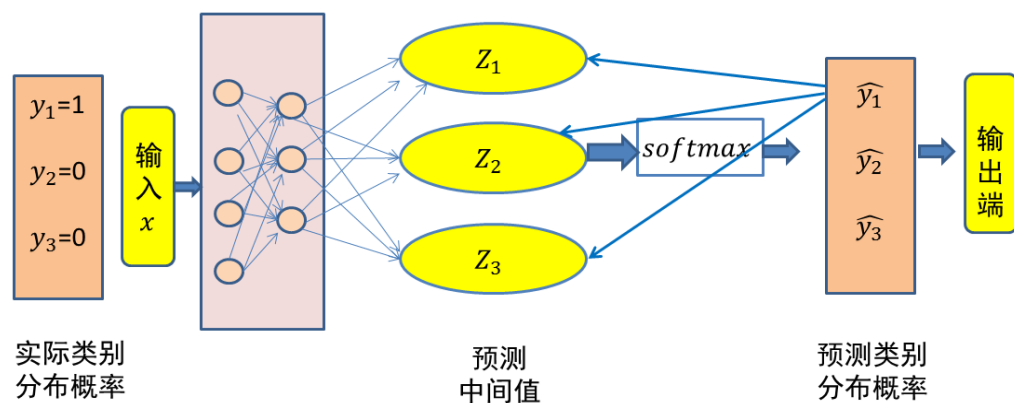


图6.3 三类分类问题中输入 x 的交叉熵损失示意图 (x 属于第一类)

交叉熵损失函数

很显然，一个良好的神经网络要尽量保证对于每一个输入数据，神经网络所预测类别分布概率与实际类别分布概率之间的差距越小越好，即交叉熵越小越好。于是，可将交叉熵作为损失函数来训练神经网络。

图6.3给出了一个三个类别分类的例子。由于输入数据 x 属于类别1，因此其实际类别概率分布值为 $y = (y_1, y_2, y_3) = (1, 0, 0)$ 。经过神经网络的变换，得到了输入数据 x 相对于三个类别的预测中间值 (z_1, z_2, z_3) 。然后，经过Softmax函数映射，得到神经网络所预测的输入数据 x 的类别分布概率 $\hat{y} = (\hat{y}_1, \hat{y}_2, \hat{y}_3)$ 。根据前面的介绍， \hat{y}_1 、 \hat{y}_2 和 \hat{y}_3 为 $(0, 1)$ 范围之间的一个概率值。由于样本 x 属于第一个类别，因此希望神经网络所预测得到的 \hat{y}_1 取值要远远大于 \hat{y}_2 和 \hat{y}_3 的取值。

常用的损失函数

交叉熵损失函数

训练中可利用如下交叉熵损失函数来对模型参数进行优化： $\text{cross entropy} = -(y_1 \times \log(\hat{y}_1) + y_2 \times \log(\hat{y}_2) + y_3 \times \log(\hat{y}_3))$

在上式中， y_2 和 y_3 均为0、 y_1 为1，因此交叉熵损失函数简化为： $-y_1 \times \log(\hat{y}_1) = -\log(\hat{y}_1)$ 。

在神经网络训练中，要将输入数据实际的类别概率分布与模型预测的类别概率分布之间的误差（即损失）从输出端向输入端传递，以便来优化模型参数。下面简单介绍根据交叉熵计算得到的误差从 \hat{y}_1 传递给 z_1 和 z_2 （ z_3 的推导与 z_2 相同）的情况。

$$\begin{aligned}\frac{\partial \hat{y}_1}{\partial z_1} &= \frac{\partial}{\partial z_1} \left(\frac{e^{z_1}}{\sum_k e^{z_k}} \right) = \frac{(e^{z_1})' \times \sum_k e^{z_k} - e^{z_1} \times e^{z_1}}{(\sum_k e^{z_k})^2} \\ &= \frac{e^{z_1}}{\sum_k e^{z_k}} - \frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_1}}{\sum_k e^{z_k}} = \hat{y}_1(1 - \hat{y}_1)\end{aligned}$$

（注意：在推导中应用了 $(e^x)' = e^x$ 、 $(\frac{v}{u})' = \frac{v'u - vu'}{u^2}$ 等公式）

由于交叉熵损失函数 $-\log(\hat{y}_1)$ 对 \hat{y}_1 求导的结果为 $-\frac{1}{\hat{y}_1}$ ， $\hat{y}_1(1 - \hat{y}_1)$ 与 $-\frac{1}{\hat{y}_1}$ 相乘为 $\hat{y}_1 - 1$ 。这说明一旦得到模型预测输出 \hat{y}_1 ，将该输出减去1就是交叉损失函数相对于 z_1 的偏导结果。

$$\frac{\partial \hat{y}_1}{\partial z_2} = \frac{\partial}{\partial z_2} \left(\frac{e^{z_1}}{\sum_k e^{z_k}} \right) = \frac{0 \times \sum_k e^{z_k} - e^{z_1} \times e^{z_2}}{(\sum_k e^{z_k})^2} = -\frac{e^{z_1}}{\sum_k e^{z_k}} \times \frac{e^{z_2}}{\sum_k e^{z_k}} = -\hat{y}_1 \hat{y}_2$$

同理，交叉熵损失函数导数为 $-\frac{1}{\hat{y}_1}$ ， $-\hat{y}_1 \hat{y}_2$ 与 $-\frac{1}{\hat{y}_1}$ 相乘结果为 \hat{y}_2 。这意味对于除第一个输出节点以外的节点进行偏导，在得到模型预测输出后，只要将其保存，就是交叉损失函数相对于其他节点的偏导结果。在 z_1 、 z_2 和 z_3 得到偏导结果后，再通过链式法则（后续介绍）将损失误差继续往输入端传递即可。

常用的损失函数

交叉熵损失函数

- 在上面的例子中，假设所预测中间值 (z_1, z_2, z_3) 经过Softmax映射后所得结果为 $(0.34, 0.46, 0.20)$ 。由于已知输入数据 x 属于第一类，显然这个输出不理想而需要对模型参数进行优化。如果选择交叉熵损失函数来优化模型，则 (z_1, z_2, z_3) 这一层的偏导值为 $(0.34 - 1, 0.46, 0.20) = (-0.66, 0.46, 0.20)$ 。
- 可以看出，Softmax和交叉熵损失函数相互结合，为偏导计算带来了极大便利。偏导计算使得损失误差从输出端向输入端传递，来对模型参数进行优化。在这里，交叉熵与Softmax函数结合在一起，因此也叫Softmax损失（Softmax with cross-entropy loss）。

感知机模型：单层感知机

早期的感知机结构和MCP模型相似，由一个输入层和一个输出层构成，因此也被称为“单层感知机”。感知机的输入层负责接收实数值的输入向量，输出层则能输出1或-1两个值。

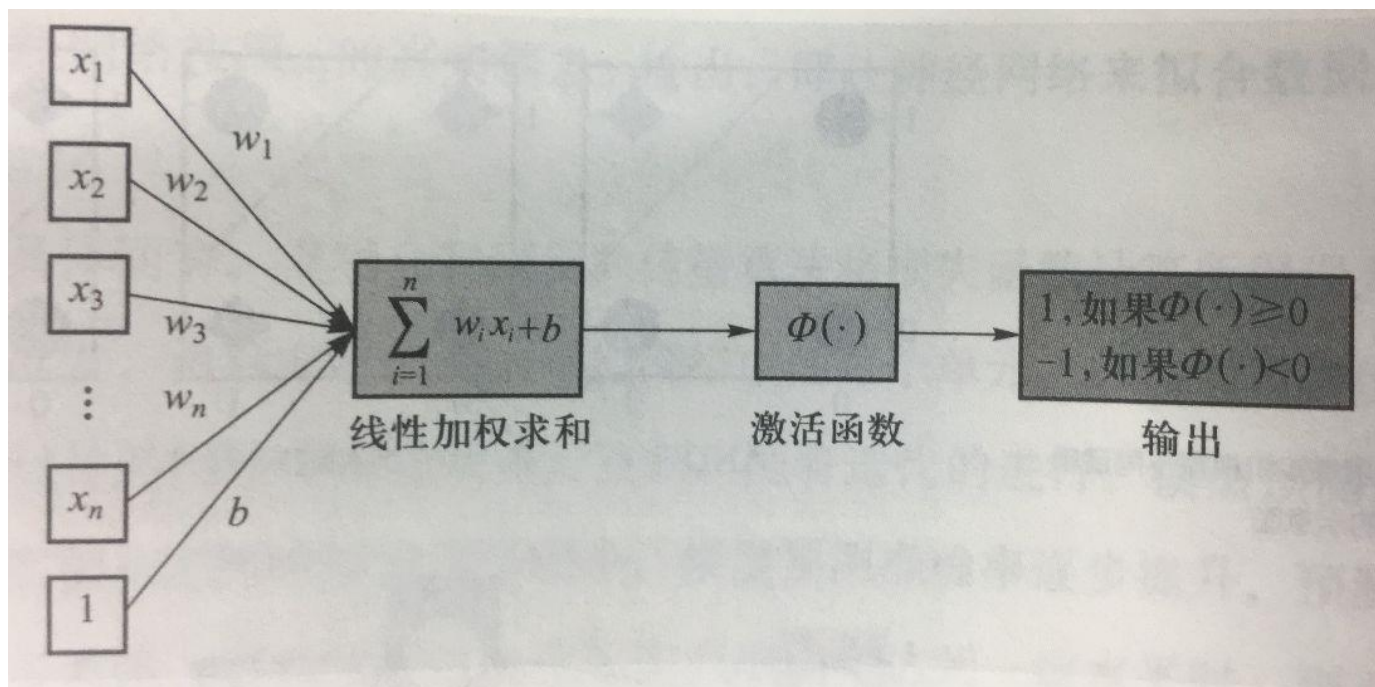


图6.4 感知机模型

感知机模型：单层感知机

单层感知机可被用来区分线性可分数据。在图6.5中，逻辑与(AND)、逻辑与非(NAND)和逻辑或(OR)为线性可分函数，所以可利用单层感知机来模拟这些逻辑函数。但是，由于逻辑异或(XOR)是非线性可分的逻辑函数，因此单层感知机无法模拟逻辑异或函数的功能。

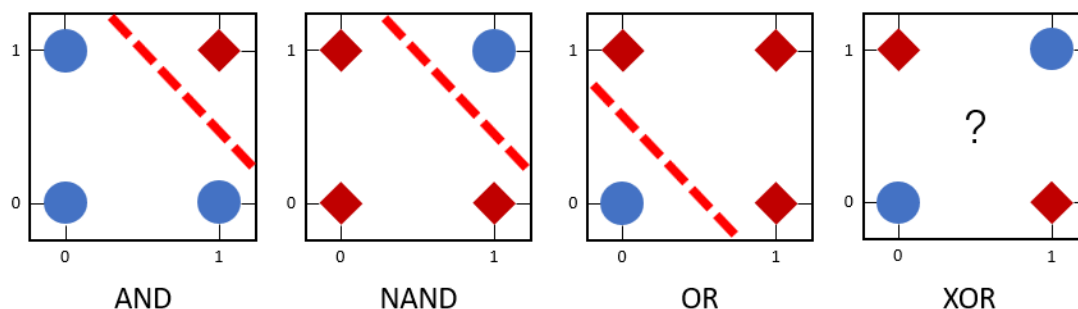


图6.5 单层感知机模拟不同逻辑函数功能的示意图

感知机模型：多层感知机

- 多层感知机由输入层、输出层和至少一层的隐藏层构成。网络中各个隐藏层中神经元可接收相邻前序隐藏层中所有神经元传递而来的信息，经过加工处理后将信息输出给相邻后续隐藏层中所有神经元。
- 各个神经元接受前一级的输入，并输出到下一级，模型中没有反馈
- 层与层之间通过“全连接”进行链接，即两个相邻层之间的神经元完全成对连接，但层内的神经元不相互连接。
- 前馈神经网络（feedforward neural network）

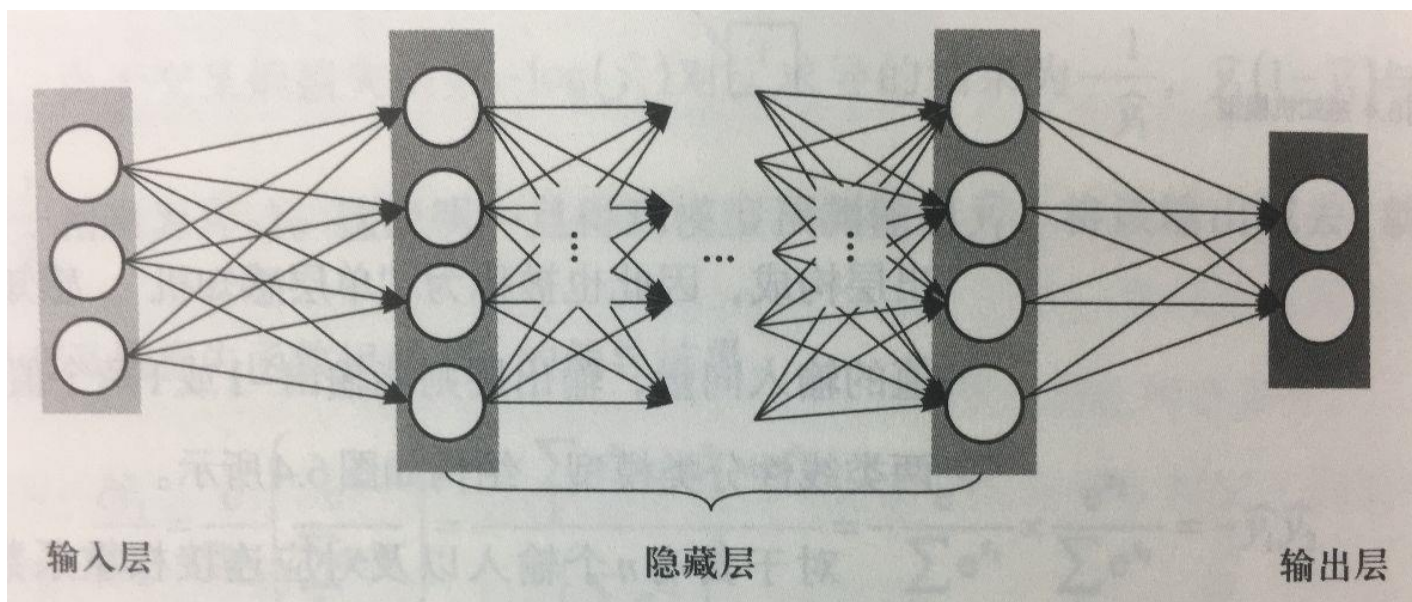
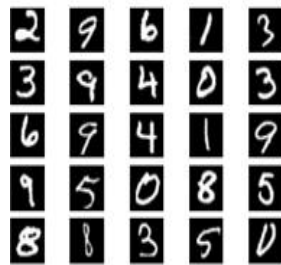


图6.6 多层感知机模型

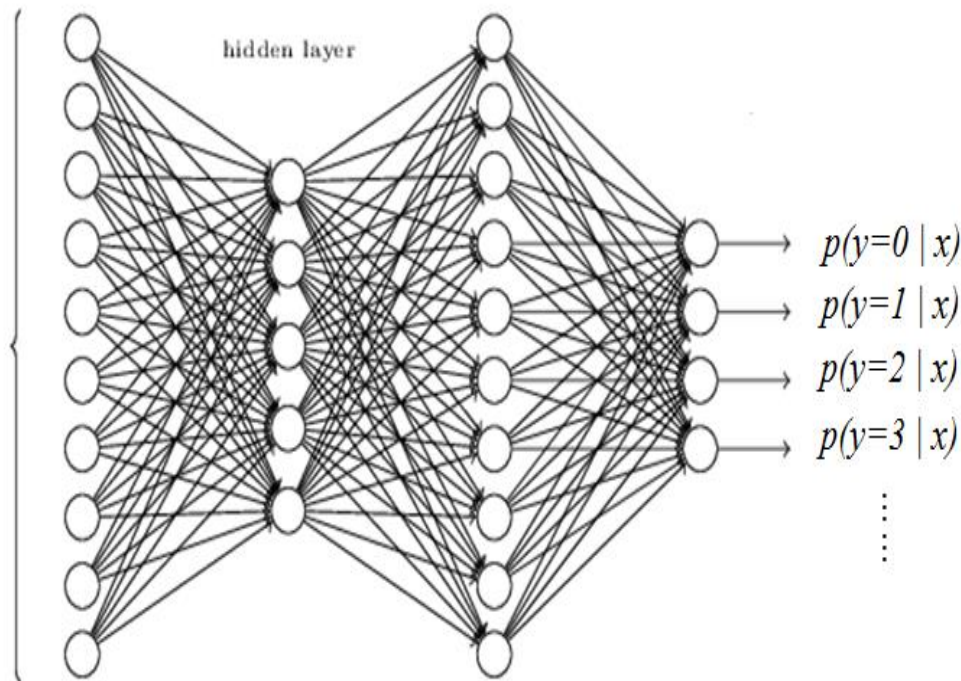
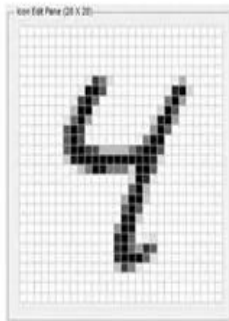
问题：如何优化网络参数？

$$w_{ij}(1 \leq i \leq n, 1 \leq j \leq m)$$

n 为神经网络层数、 m 为每层中神经元个数



input layer
(784 neurons)



参数优化：梯度下降 (Gradient Descent)

梯度下降算法是一种使得**损失函数**最小化的方法。一元变量所构成函数 f 在 x 处梯度为：

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$



- 在多元函数中，梯度是对每一变量所求导数组成的向量
- 梯度的反方向是函数值下降最快的方向，因此是损失函数求解的方向

梯度下降 (Gradient Descent)

- 假设损失函数 $f(x)$ 是连续可微的多元变量函数，其泰勒展开如下(Δx 是微小的增量):

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)(\Delta x)^2 + \cdots + \frac{1}{n!}f^{(n)}(x)(\Delta x)^n$$

$$f(x + \Delta x) - f(x) \approx (\nabla f(x))^T \Delta x$$

- 因为我们的目的是最小化损失函数 $f(x)$ ，则 $f(x + \Delta x) < f(x)$ ，于是 $(\nabla f(x))^T \Delta x < 0$
- 在 $(\nabla f(x))^T \Delta x = \|\nabla f(x)\| \|\Delta x\| \cos \theta$ 中， $\|\nabla f(x)\|$ 和 $\|\Delta x\|$ 分别为损失函数梯度的模和下一轮迭代中 x 取值增量的模，两者均为正数。为了保证损失误差减少，只要保证 $\cos \theta < 0$ 。当 $\theta = 180^\circ$ 时， $\cos \theta = -1$ ，这时损失函数减少的幅度值 $(\nabla f(x))^T \Delta x$ 取到最小。

梯度下降 (Gradient Descent)

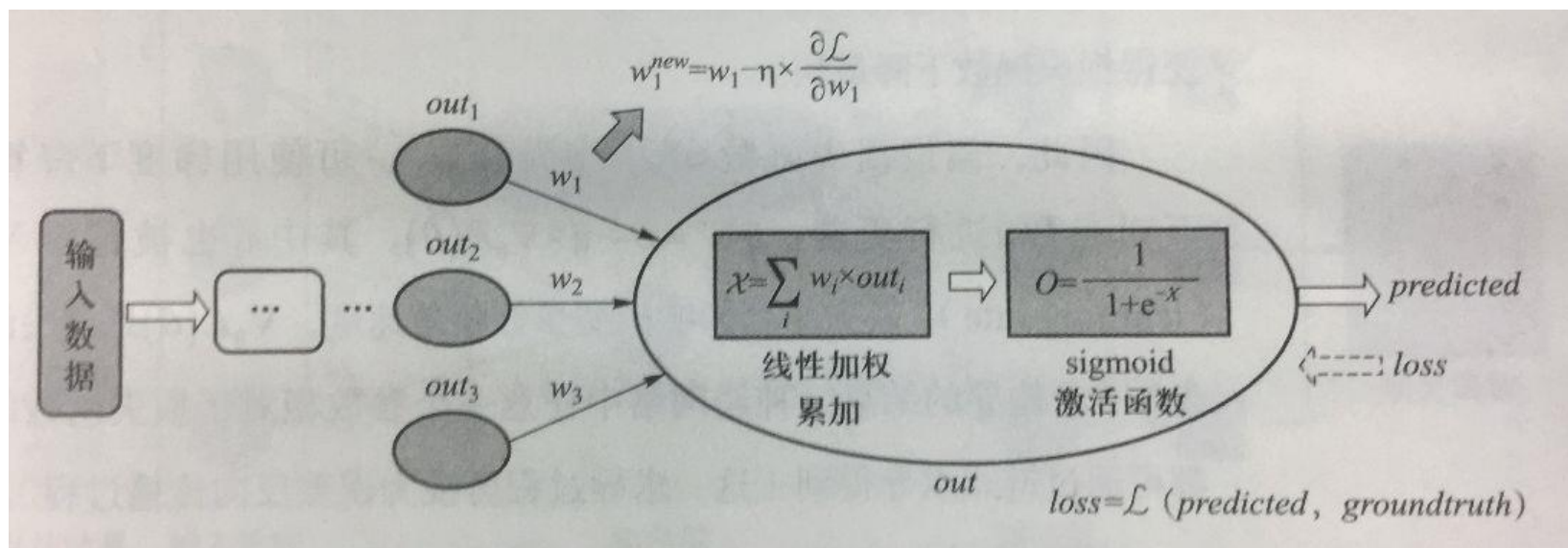
当 $\theta = 180^\circ$ 时， $f(x + \Delta x)$ 和 $f(x)$ 之间的差值为： $f(x + \Delta x) - f(x) = \|\nabla f(x)\| \|\Delta x\| \cos \theta = -\|\Delta x\| \|\nabla f(x)\|$ 。这说明只要沿着损失函数梯度的反方向选取 x 的增量 Δx ，就会保证损失误差下降最多、下降最快，犹如从山峰处沿着最陡峭路径可快速走到山谷。在前面的推导中忽略了损失函数的二阶导数以及其高阶导数的取值，因此在实际中引入步长 η ，用 $x - \eta \nabla f(x)$ 来更新 x （在具体实现时 η 可取一个定值）。

从下面的公式可以计算得到变量的增量为 $-\eta \nabla f(x)$

$$\underbrace{x - \eta \nabla f(x)}_{\text{变量更新后的取值}} - \underbrace{x}_{\text{变量当前取值}} = \underbrace{-\eta \nabla f(x)}_{\text{变量的增量}}$$

参数优化：误差反向传播 (error back propagation, BP)

- BP算法是一种将输出层误差反向传播给隐藏层进行参数更新的方法。
- 将误差从后向前传递，将误差分摊给各层所有单元，从而获得各层单元所产生的误差，进而依据这个误差来让各层单元负起各自责任、修正各单元参数。



6.7 链式求导与模型参数更新示意图

参数优化：误差反向传播 (error back propagation, BP)

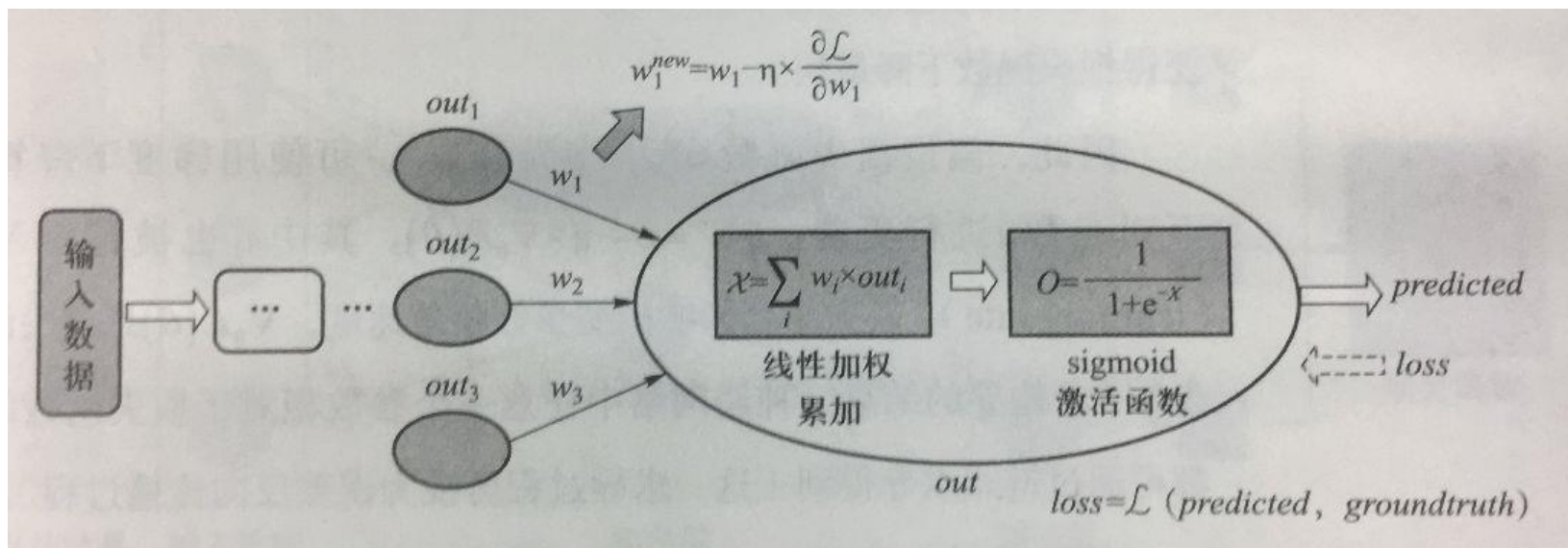


图6.7 链式求导与模型参数更新示意图

为了使损失函数 \mathcal{L} 取值减少（从而保证模型预测结果与实际结果之间的差距越来越小），需要求取损失函数 \mathcal{L} 相对于 w_1 的偏导，然后按照损失函数梯度的反方向选取一个微小的增量，来调整 w_1 的取值，就能够保证损失函数取值减少。即将 w_1 变为 $w_1 - \eta \frac{d\mathcal{L}}{dw_1}$ 后，能使得损失误差减少。这里 $\frac{d\mathcal{L}}{dw_1}$ 为损失函数 \mathcal{L} 对 w_1 的偏导。于是，需要求取损失函数 \mathcal{L} 对变量参数 w_1 的偏导。

参数优化：误差反向传播 (error back propagation, BP)

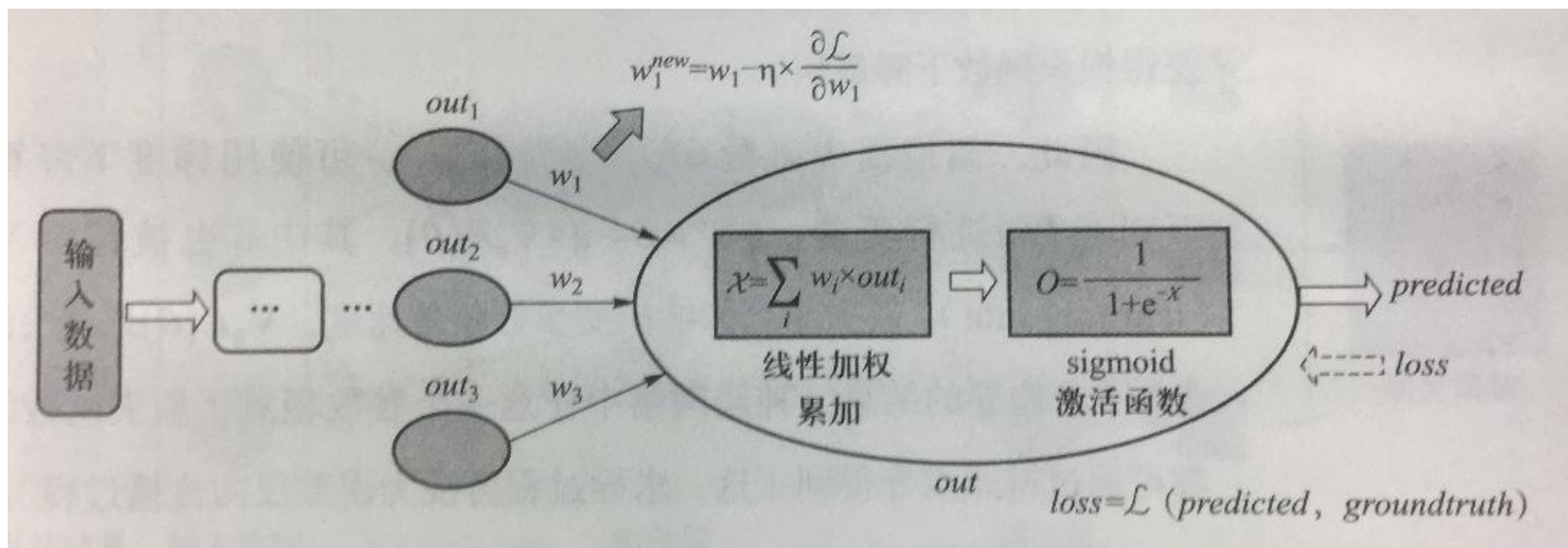


图6.7 链式求导与模型参数更新示意图

- 由于 w_1 与加权累加函数 X 和 sigmoid 函数均有关，因此 $\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dO} \frac{dO}{dX} \frac{dX}{dw_1}$ 。在这个链式求导中： $\frac{d\mathcal{L}}{dO}$ 与损失函数的定义有关； $\frac{dO}{dX}$ 是对 sigmoid 函数求导，结果为 $\frac{1}{1+e^{-X}} \times (1 - \frac{1}{1+e^{-X}})$ ； $\frac{dX}{dw_1}$ 是加权累加函数 $w_1 \times out_1 + w_2 \times out_2 + w_3 \times out_3$ 对 w_1 求导，结果为 out_1 。
- 链式求导实现了损失函数对某个自变量求偏导，好比将损失误差从输出端向输入端逐层传播，通过这个传播过程来更新该自变量取值。梯度下降法告诉我们，只要沿着损失函数梯度的反方向来更新参数，就可使得损失函数下降最快。

参数优化：误差反向传播 (error back propagation, BP)

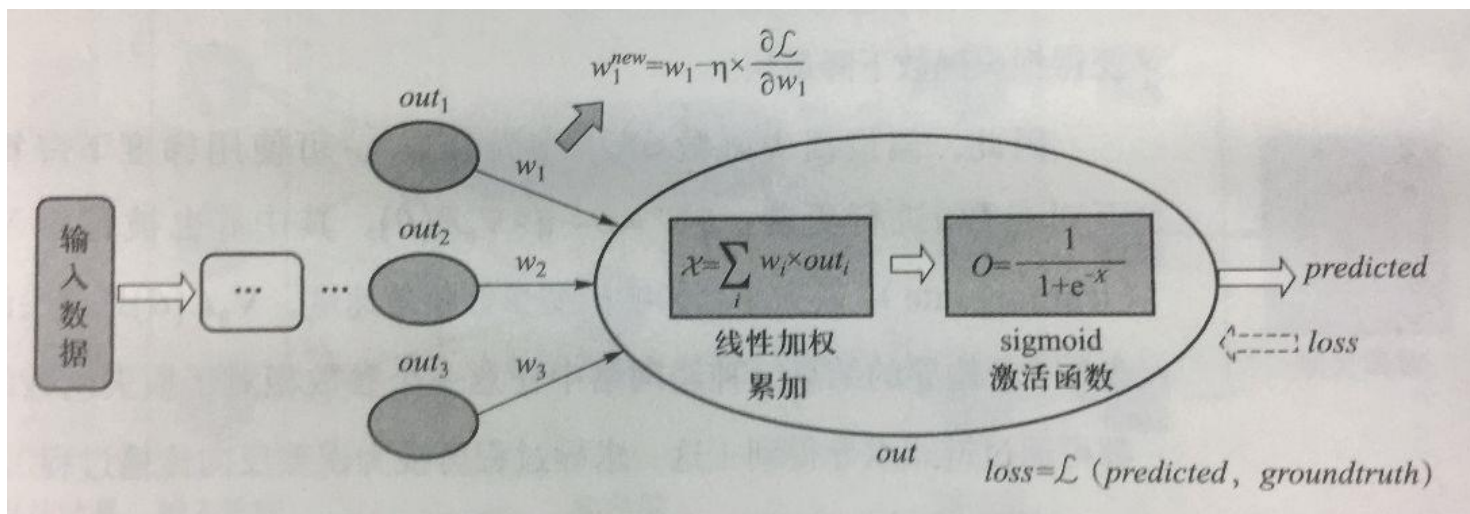


图6.7 链式求导与模型参数更新示意图

- 参数 w_1 在下一轮迭代中的取值被调整为： $w_1^{new} = w_1 - \eta \times \frac{d\mathcal{L}}{dw_1} = w_1 - \eta \times \frac{d\mathcal{L}}{dO} \frac{dO}{dX} \frac{dX}{dw_1} = w_1 - \eta \times \frac{d\mathcal{L}}{dO} \times \frac{1}{1+e^{-X}} \times (1 - \frac{1}{1+e^{-X}}) \times out_1$ 。
- 按照同样的方法，可调整 w_2 、 w_3 以及其它图中未显示参数的取值。经过这样的调整，在模型参数新的取值作用下，损失函数 $\mathcal{L}(\theta)$ 会以最快下降方式逐渐减少，直至减少到最小值（即全局最小值）。

参数优化：误差反向传播 (error back propagation, BP)

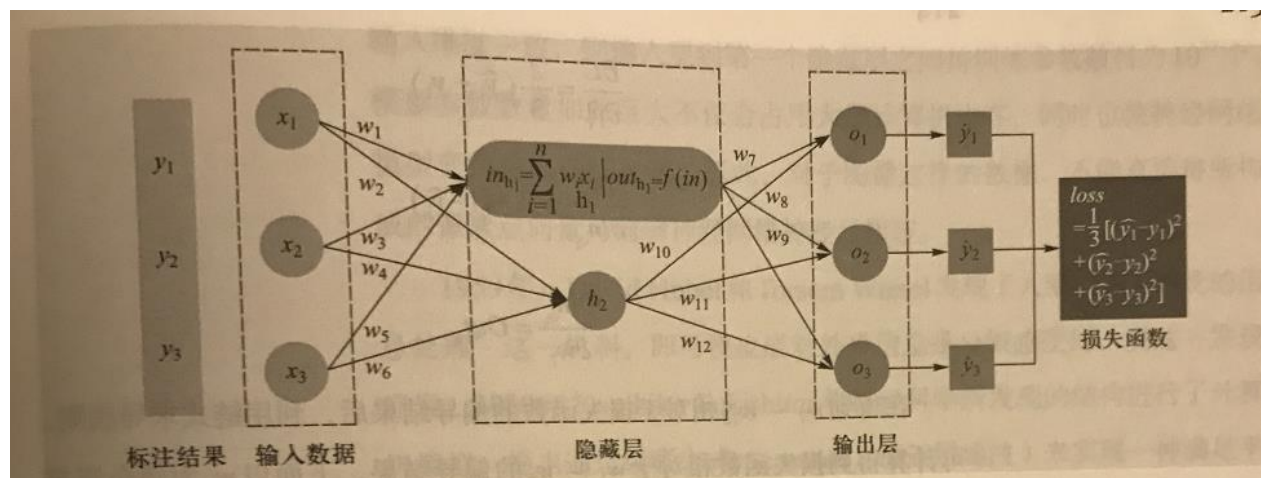


图6.8 包含一个隐藏层的多层感知机

- 通过一个三类分类的具体例子来介绍神经网络中参数更新过程。给定一个包含输入层、一层隐藏层和输出层的多层感知机，其中隐藏层由两个神经元构成。
- 网络使用Sigmoid函数作为神经元的激活函数，使用均方损失函数来计算网络输出值与实际值之间的误差。

参数优化：误差反向传播 (error back propagation, BP)

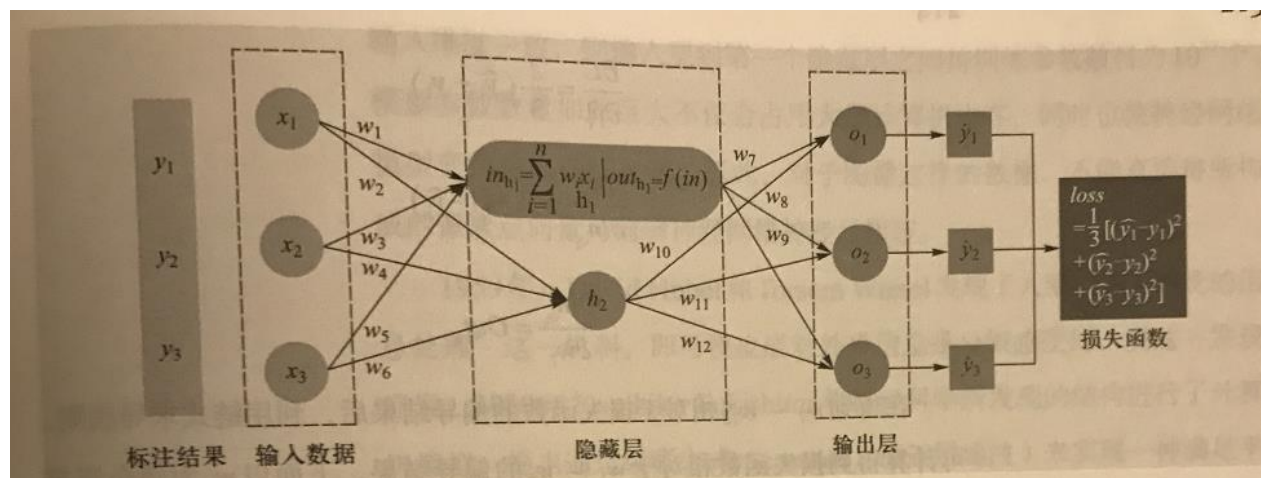


图6.8 包含一个隐藏层的多层感知机

每个神经元完成如下两项任务：1) 对相邻前序层所传递信息进行线性加权累加；2) 对加权累加结果进行非线性变换。假设样本数据输入为 (x_1, x_2, x_3) ，其标注信息为 (y_1, y_2, y_3) 。在三类分类问题中，对于输入数据 (x_1, x_2, x_3) ， y_1 、 y_2 和 y_3 中只有一个取值为1、其余两个取值为0。

参数优化：误差反向传播 (error back propagation, BP)

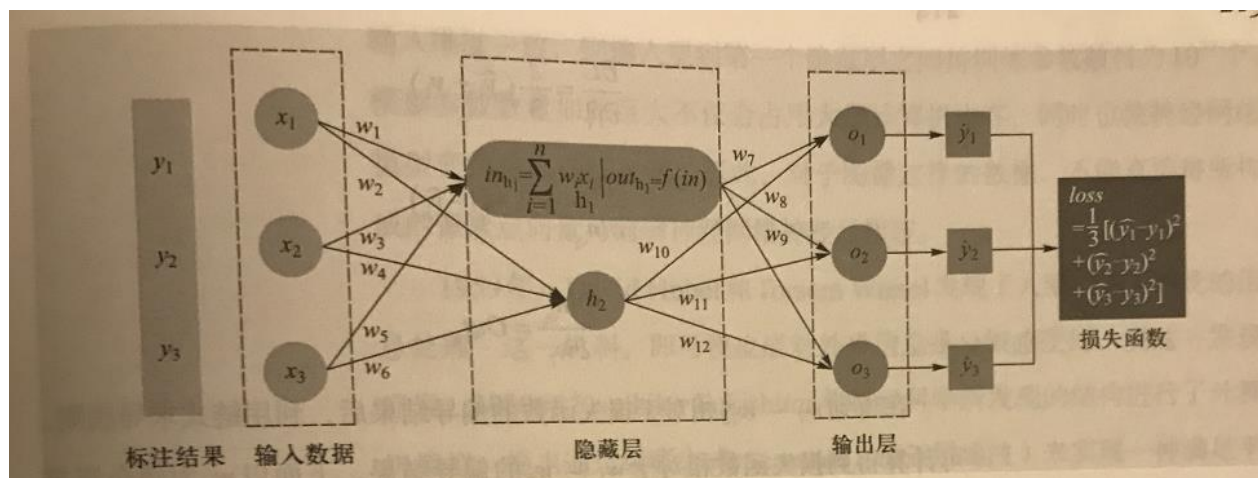


图6.8 包含一个隐藏层的多层感知机

单元	相邻前序神经元传递信息线性累加	非线性变换
h_1	$In_{h_1} = w_1 * x_1 + w_3 * x_2 + w_5 * x_3$	$Out_{h_1} = \text{sigmoid}(In_{h_1})$
h_2	$In_{h_2} = w_2 * x_1 + w_4 * x_2 + w_6 * x_3$	$Out_{h_2} = \text{sigmoid}(In_{h_2})$
o_1	$In_{o_1} = w_7 * Out_{h_1} + w_{10} * Out_{h_2}$	$\hat{y}_1 = Out_{o_1} = \text{sigmoid}(In_{o_1})$
o_2	$In_{o_2} = w_8 * Out_{h_1} + w_{11} * Out_{h_2}$	$\hat{y}_2 = Out_{o_2} = \text{sigmoid}(In_{o_2})$
o_3	$In_{o_3} = w_9 * Out_{h_1} + w_{12} * Out_{h_2}$	$\hat{y}_3 = Out_{o_3} = \text{sigmoid}(In_{o_3})$

表6.2 在图6.8中神经网络各单元计算结果

参数优化：误差反向传播 (error back propagation, BP)

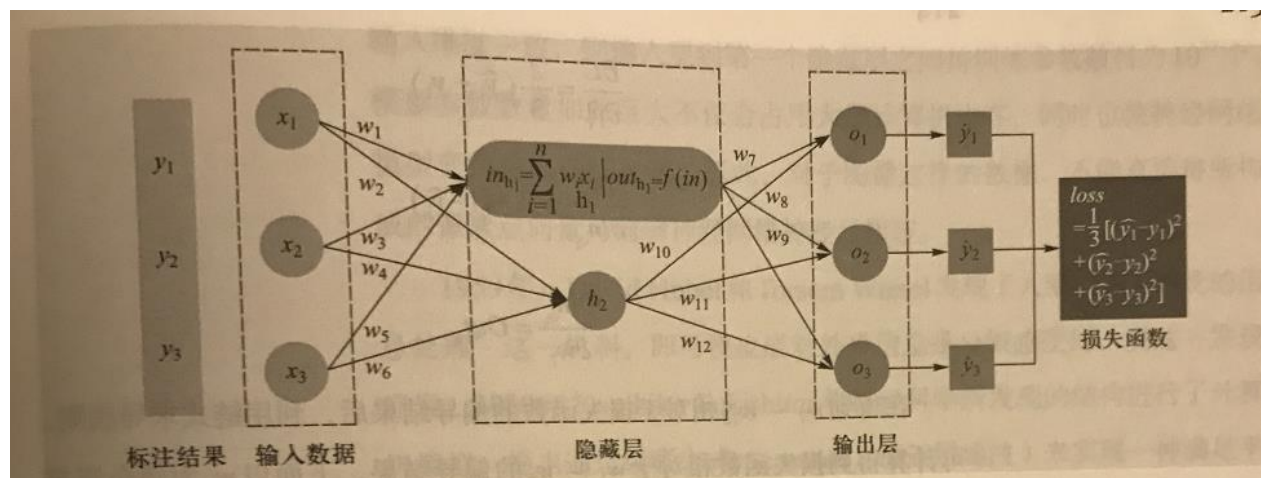


图6.8 包含一个隐藏层的多层感知机

一旦神经网络在当前参数下给出了预测结果 $(\hat{y}_1, \hat{y}_2, \hat{y}_3)$ 后，通过均方误差损失函数来计算模型预测值与真实值 (y_1, y_2, y_3) 之间误差，记为 $loss = \frac{1}{3} \sum_{i=1}^3 (\hat{y}_i - y_i)^2$ 。

接下来通过梯度下降和误差反向传播方法，沿着损失函数梯度的反方向来更改参数变量取值，使得损失函数快速下降到其最小值。由于损失函数对 $w_7 \sim w_{12}$ 的偏导计算相似，在此以 w_7 为例来介绍如何更新 w_7 这一参数取值。记损失函数为 $\mathcal{L}(w)$ ， $\mathcal{L}(w) = \frac{1}{3} \sum_{i=1}^3 (\hat{y}_i - y_i)^2$ 。

参数优化：误差反向传播 (error back propagation, BP)

损失函数 \mathcal{L} 对参数 w_7 的偏导可如下计算：

$$\delta_7 = \frac{\partial \mathcal{L}}{\partial w_7} = \frac{\partial \mathcal{L}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial \ln_{o_1}} * \frac{\partial \ln_{o_1}}{\partial w_7}$$

上述公式中每一项结果如下：

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_1} = (x + a)^n = \frac{2}{3} (\hat{y}_1 - y_1)$$

$$\frac{\partial \hat{y}_1}{\partial \ln_{o_1}} = \hat{y}_1 * (1 - \hat{y}_1)$$

$$\frac{\partial \ln_{o_1}}{\partial w_7} = \text{Out}_{h_1}$$

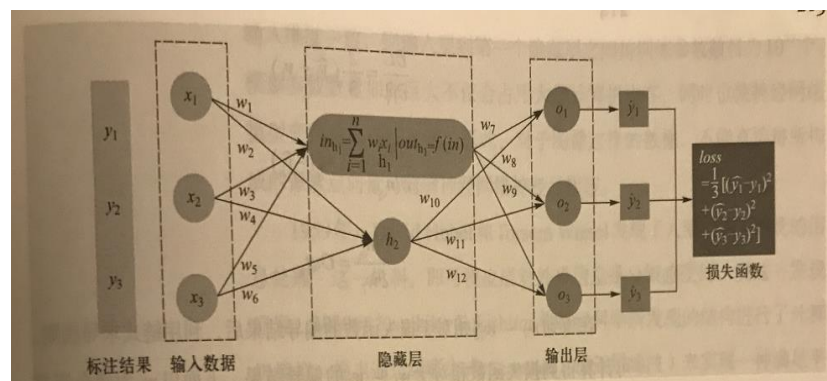


图6.8 包含一个隐藏层的多层感知机

参数优化：误差反向传播 (error back propagation, BP)

下面以 w_1 为例介绍损失函数 \mathcal{L} 对 w_1 的偏导 δ_1 。

$$\begin{aligned}
 \delta_1 &= \frac{\partial \mathcal{L}}{\partial w_1} \\
 &= \frac{\partial \mathcal{L}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial w_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial w_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_3} * \frac{\partial \hat{y}_3}{\partial w_1} \\
 &= \frac{\partial \mathcal{L}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial \ln_{o_1}} * \frac{\partial \ln_{o_1}}{\partial \text{Out}_{h_1}} * \frac{\partial \text{Out}_{h_1}}{\partial \ln_{h_1}} * \frac{\partial \ln_{h_1}}{\partial w_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial \ln_{o_2}} \\
 &\quad * \frac{\partial \ln_{o_2}}{\partial \text{Out}_{h_1}} * \frac{\partial \text{Out}_{h_1}}{\partial \ln_{h_1}} * \frac{\partial \ln_{h_1}}{\partial w_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_3} * \frac{\partial \hat{y}_3}{\partial \ln_{o_3}} * \frac{\partial \ln_{o_3}}{\partial \text{Out}_{h_1}} * \frac{\partial \text{Out}_{h_1}}{\partial \ln_{h_1}} \\
 &\quad * \frac{\partial \ln_{h_1}}{\partial w_1} \\
 &= \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial \ln_{o_1}} * \frac{\partial \ln_{o_1}}{\partial \text{Out}_{h_1}} + \frac{\partial \mathcal{L}}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial \ln_{o_2}} * \frac{\partial \ln_{o_2}}{\partial \text{Out}_{h_1}} + \frac{\partial \mathcal{L}}{\partial \hat{y}_3} * \frac{\partial \hat{y}_3}{\partial \ln_{o_3}} \right) \\
 &\quad * \frac{\partial \text{Out}_{h_1}}{\partial \ln_{h_1}} * \frac{\partial \ln_{h_1}}{\partial w_1} \\
 &= (\delta_7 + \delta_8 + \delta_9) * \frac{\partial \text{Out}_{h_1}}{\partial \ln_{h_1}} * \frac{\partial \ln_{h_1}}{\partial w_1}
 \end{aligned}$$

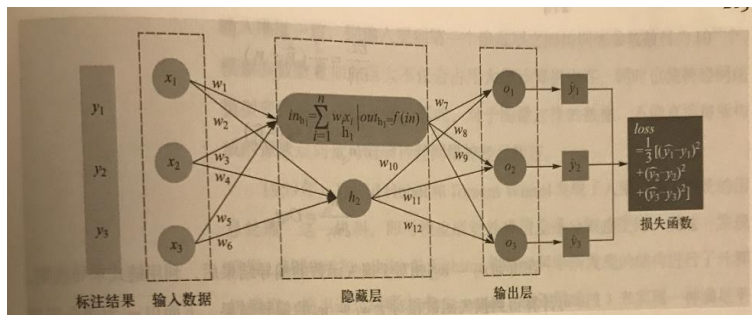
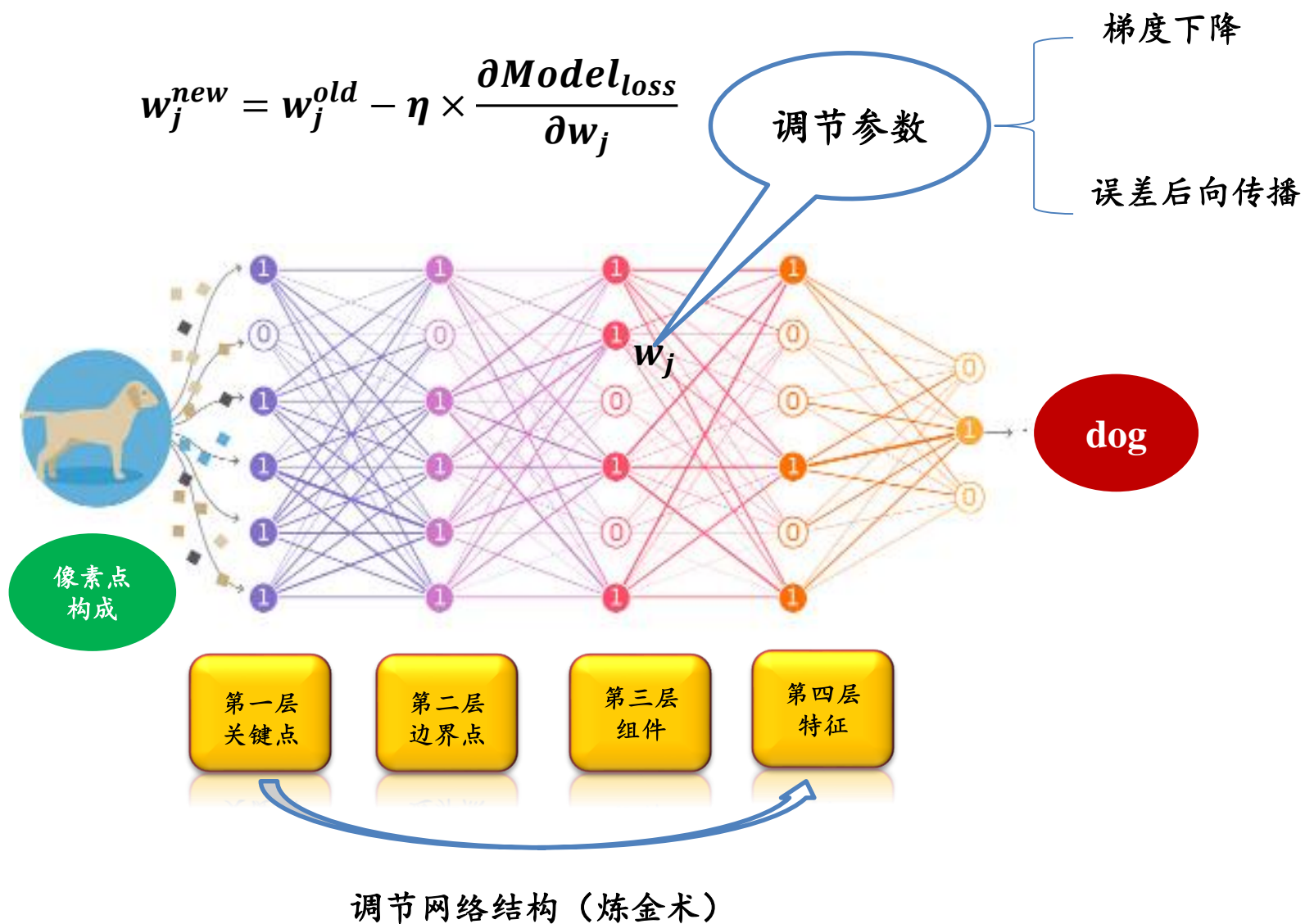


图6.8 包含一个隐藏层的多层感知机

在计算得到所有参数相对于损失函数 \mathcal{L} 的偏导结果后，利用梯度下降算法，通过 $w_i^{\text{new}} = w_i - \eta * \delta_i$ 来更新参数取值。然后不断迭代，直至模型参数收敛，此时损失函数减少到其最小值。

机器学习的能力在于拟合和优化



提纲

一、深度学习的历史发展

二、前馈神经网络

三、卷积神经网络

四、循环神经网络

五、深度生成学习

六、深度学习应用

卷积神经网络(convolution neural network, CNN)

- 在前馈神经网络中，输入层的输入数据直接与第一个隐藏层中所有神经元相互连接。如果输入数据是一幅图像，需要把灰度图像（二维矩阵）或彩色图像（三维矩阵）转换为向量形式。则给定一幅分辨率 1000×1000 的灰度图像，输入数据为一个1,000,000维的向量，如果输入数据与第一个隐藏层中所有神经元均相连，且第一个隐藏层维度和输入维度一致，则输入层到第一个隐藏层之间待训练参数数目为 10^{12} 个，模型参数数量如此巨大不仅会占用大量计算机内存，同时也使神经网络模型变得难以训练收敛。因此，对于图像这样的数据，不能直接将所构成的像素点向量与前馈神经网络神经元相连。
- 1959年，David Hubel和Torsten Wiesel发现了人脑“视觉系统的信息处理”这一机制，即可视皮层对外界信息是分级感受的。受这一发现启发，1980年Kunihiko Fukushima将神经科学所发现的结构进行了计算机模拟，提出通过级联方式（cascade，即逐层滤波）来实现一种满足平移不变性的网络Neocognitron，这就是卷积神经网络的前身。20世纪90年代，LeCun等人，设计了一种被称为LeNet-5的卷积神经网络用于手写体识别，初步确立了卷积神经网络的基本结构。
- 如今，虽然针对不同应用场景的卷积神经网络结构已变得愈发复杂、性能也变得更加强健，但是究其本质，这些复杂网络结构仍是以卷积操作与池化操作为核心构建而成。

卷积神经网络(convolution neural network, CNN)

- 图像中像素点具有很强的空间依赖性，卷积（convolution）就是针对像素点的空间依赖性来对图像进行处理的一种技术。
- 在图像卷积计算中，需要定义一个卷积核（kernel）。卷积核是一个二维矩阵，矩阵中数值为对图像中与卷积核同样大小的子块像素点进行卷积计算时所采用的权重。
- 卷积核中的权重系数 w_i 是通过数据驱动机制学习得到，其用来捕获图像中某像素点及其邻域像素点所构成的特有空间模式。一旦从数据中学习得到权重系数，这些权重系数就刻画了图像中像素点构成的空间分布不同模式。

卷积神经网络：卷积操作

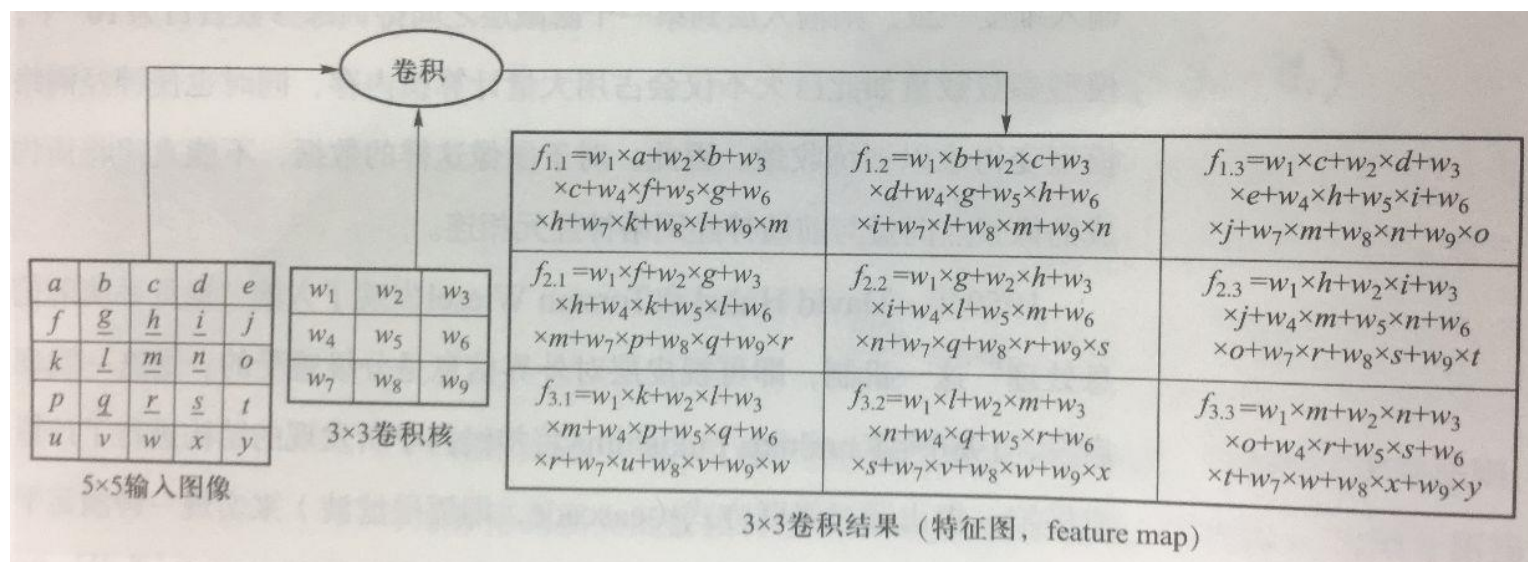


图6.9 图像卷积例子

给定一个权重分别为 $w_i (1 \leq i \leq 9)$ 、大小为 3×3 的卷积核以及一个 5×5 大小灰度图像。该卷积核对图像的卷积操作就是分别以图像中 g 、 h 、 i 、 l 、 m 、 n 、 q 、 r 和 s 所在像素点位置为中心，形成一个 3×3 大小的图像子块区域，然后用卷积核所定义权重 w_i 对图像子块区域内每个像素点进行加权累加，所得结果即为图像子块区域中心像素点被卷积后（即滤波后）的结果。滤波也可视为在给定卷积核权重前提下，记住了邻域像素点之间的若干特定空间模式、忽略了某些空间模式。

卷积神经网络：卷积操作

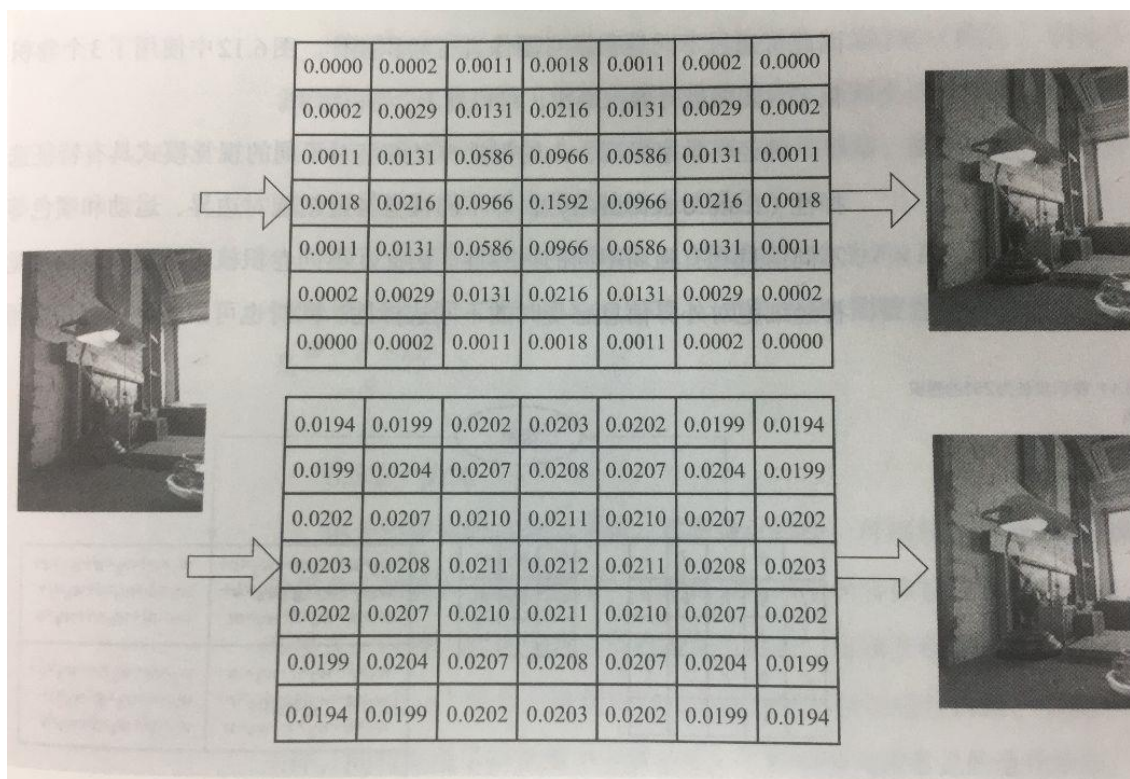
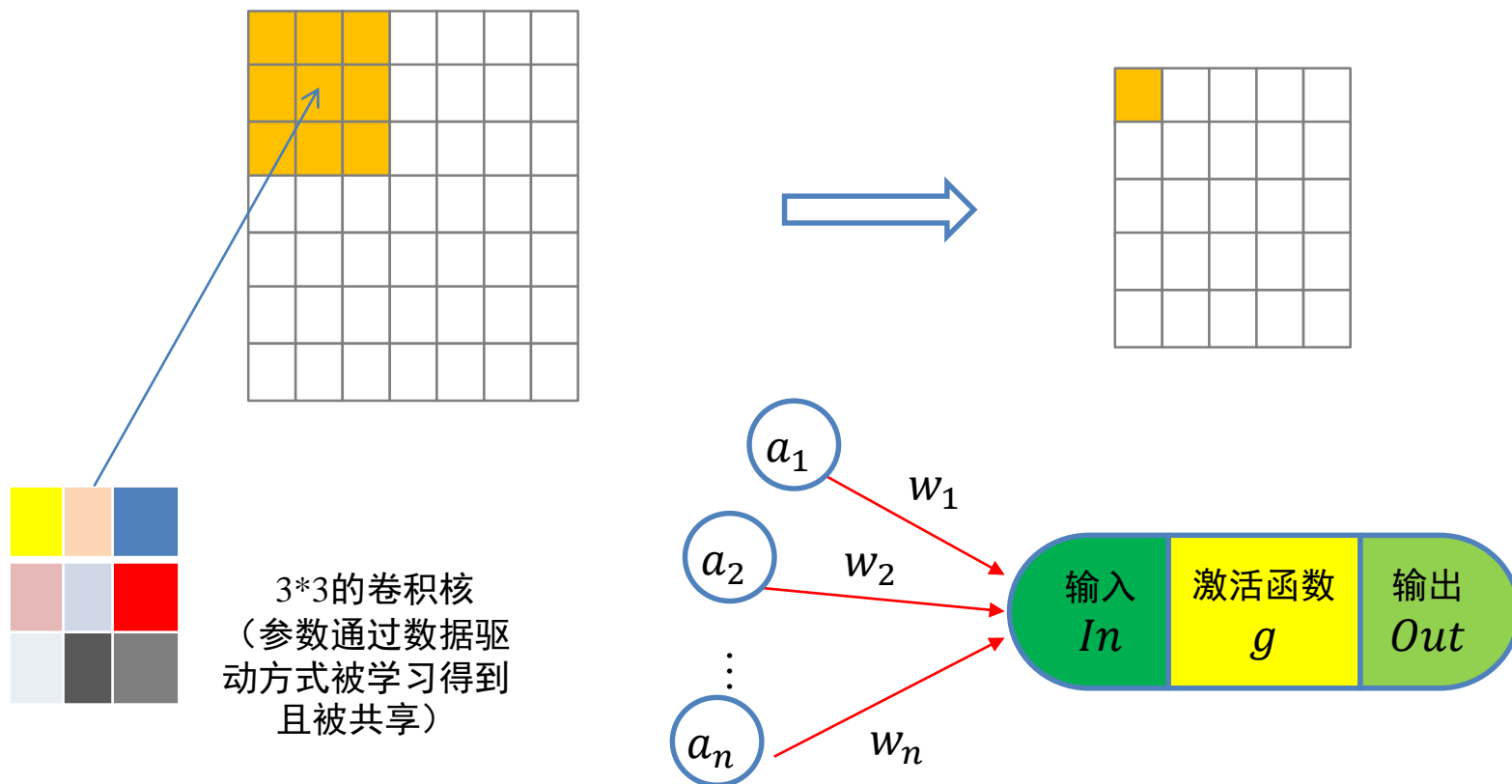


图6.10 同一幅图像被两个不同卷积核计算结果

如果卷积核中心位置的权重系数越小且与其它卷积权重系数差别越小，则卷积所得到图像滤波结果越模糊，这被称为图像平滑操作

卷积神经网络：卷积操作

7*7大小的图像，通过3*3大小卷积矩阵以1的步长进行卷积操作，可得到5*5大小的卷积结果



卷积神经网络：卷积操作

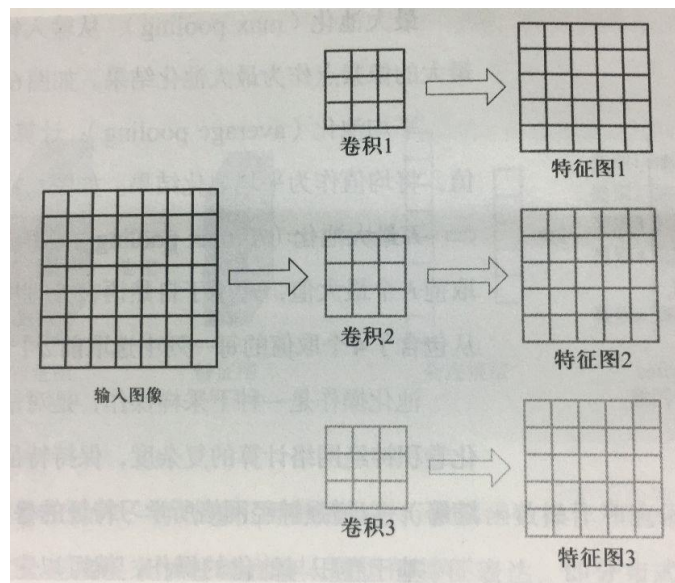
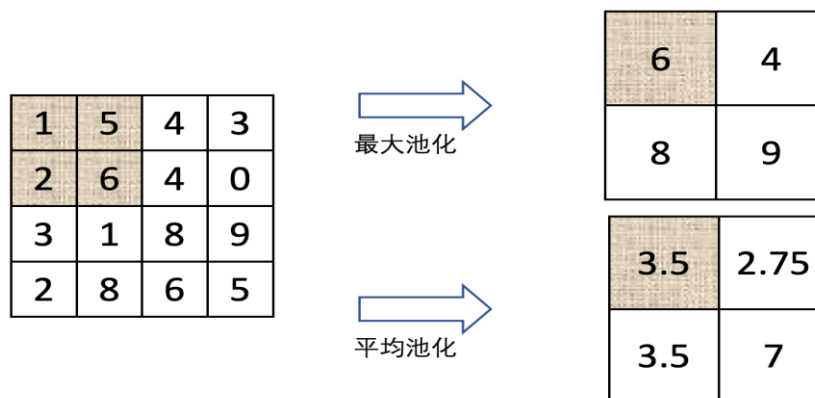


图6.12 3个卷积核对图像进行卷积示意图

神经科学家发现，人的视觉神经细胞对不同的视觉模式具有特征选择性（Feature Selectivity），即不同视觉神经细胞对边界、运动和颜色等不同信息具有强弱不同的选择性。因此，不同卷积核可被用来刻画视觉神经细胞对外界信息感受时的不同选择性。同时也可以看到，卷积所得结果中，每个输出点的取值仅依赖于其在输入图像中该点及其邻域区域点的取值，与这个区域之外的其他点取值均无关，该区域被称为感受野（receptive field），正所谓“管中窥豹、见微知著”。在卷积神经网络中，感受野是卷积神经网络每一层输出的特征图（feature map）上的像素点在输入图像上映射的区域大小。也就是说，感受野是特征图上一个点对应输入图像上的区域。

卷积神经网络：池化操作



由于图像中存在较多冗余，在图像处理中，可用某一区域子块的统计信息（如最大值或均值等）来刻画该区域中所有像素点呈现的空间分布模式，以替代区域子块中所有像素点取值，这就是卷积神经网络中池化(pooling)操作。池化操作对卷积结果特征图进行约减，实现了下采样，同时保留了特征图中主要信息。

卷积神经网络：池化操作

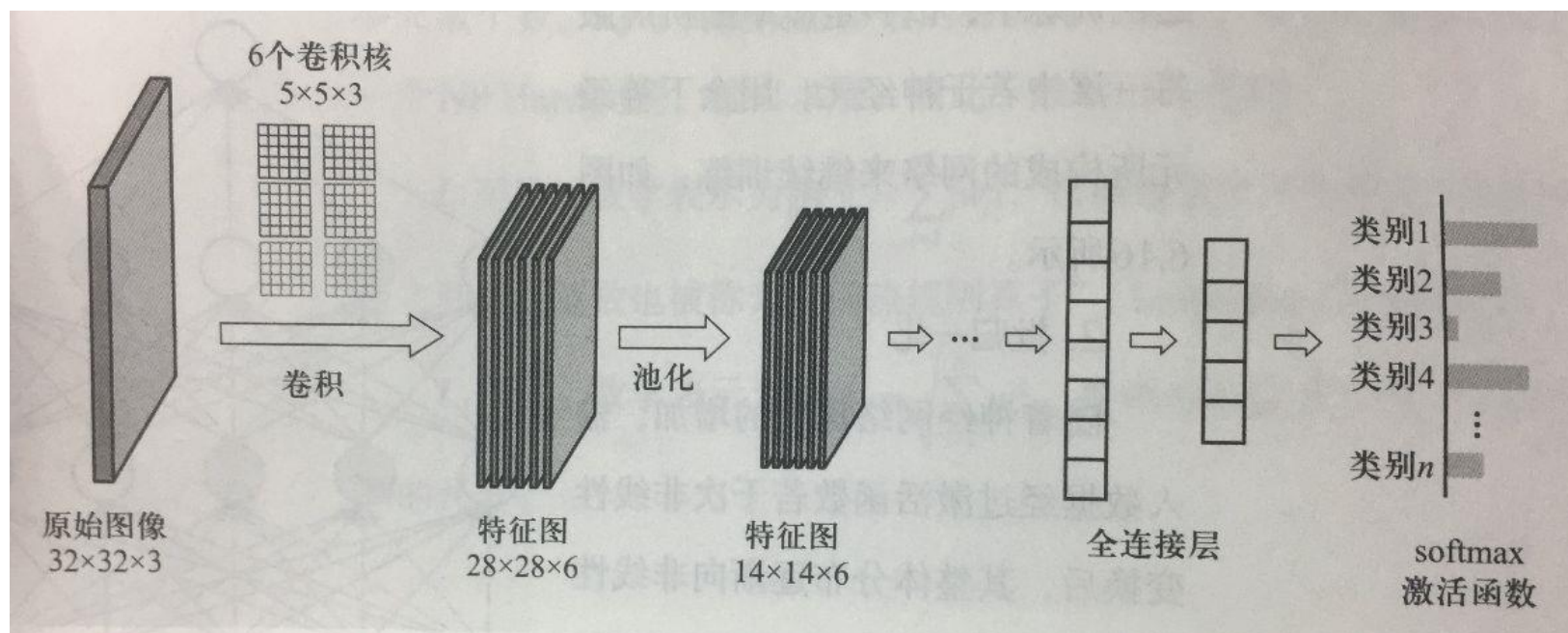


图6.15 基于卷积神经网络的图像分类示意图

对于输入的海量标注数据，通过多次迭代训练，卷积神经网络在若干次卷积操作、接着对卷积所得结果进行激活函数操作和池化操作下，最后通过全连接层来学习得到输入数据的特征表达，即分布式向量表达(distributed vector representation)。

神经网络正则化

为了缓解神经网络在训练过程中出现的过拟合问题，需要采取一些正则化技术来提升神经网络的泛化能力(generalization)

- **Dropout**
- **Batch-Normalization**
- **L1-Norm & L2-Norm**

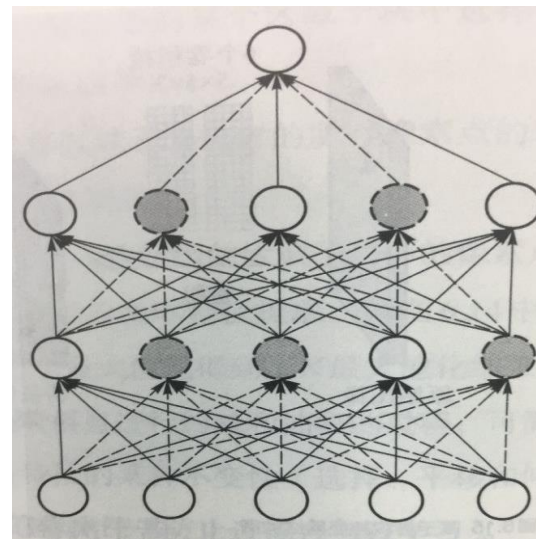


图6.16 使用Dropout的神经网络模型

L_1 范数：数学表示为 $\|W\|_1 = \sum_{i=1}^N |w_i|$ ，指模型参数 W 中各个元素的绝对值之和。

L_1 范数也被称为“稀疏规则算子” (Lasso regularization) 。

L_2 范数：数学表示为 $\|W\|_2 = \sqrt{\sum_{i=1}^N w_i^2}$ ，指模型参数 W 中各个元素平方和的开方。

提纲

一、深度学习的历史发展

二、前馈神经网络

三、卷积神经网络

四、循环神经网络

五、深度生成学习

六、深度学习应用

循环神经网络

- 循环神经网络 (Recurrent Neural Network, RNN) 是一类处理序列数据 (如文本句子、视频帧等) 时所采用的网络结构。先前所介绍的前馈神经网络或卷积神经网络所需要处理的输入数据一次性给定, 难以处理存在前后依赖关系的数据。
- 循环神经网络的本质是希望模拟人所具有的记忆能力, 在学习过程中记住部分已经出现的信息, 并利用所记住的信息影响后续结点输出。循环神经网络在自然语言处理, 例如语音识别、情感分析、机器翻译等领域有重要应用。

循环神经网络

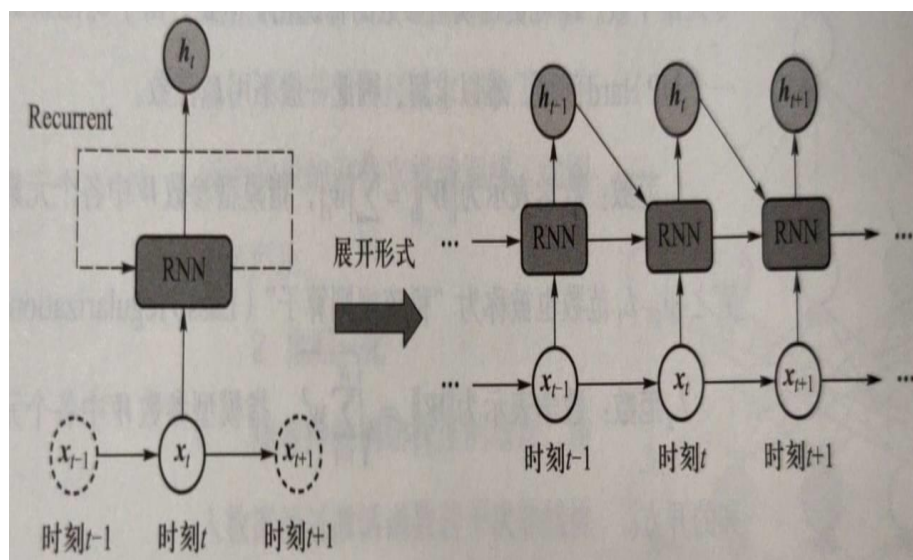


图6.17 循环神经网络结构及其展开形式示意图

时刻 t 所得到的隐式编码 h_t 是由上一时刻隐式编码 h_{t-1} 和当前输入 x_t 共同参与生成的，这可认为隐式编码 h_{t-1} 已经“记忆”了 t 时刻之前的时序信息，或者说前序时刻信息影响了后续时刻信息的处理。与前馈神经网络和卷积神经网络在处理时需要将所有数据一次性输入不同，这体现了循环神经网络可刻画序列数据存在时序依赖这一重要特点。

由“循环”两字可知，循环神经网络在处理数据过程中构成了一个循环体。对于一个序列数据，在每一时刻 t ，循环神经网络单元会读取当前输入数据 x_t 和前一时刻输入数据 x_{t-1} 所对应的隐式编码结果 h_{t-1} ，一起生成 t 时刻的隐式编码结果 h_t 。接着将 h_t 后传，去参与生成 $t+1$ 时刻输入数据 x_{t+1} 的隐式编码 h_{t+1} 。

循环神经网络

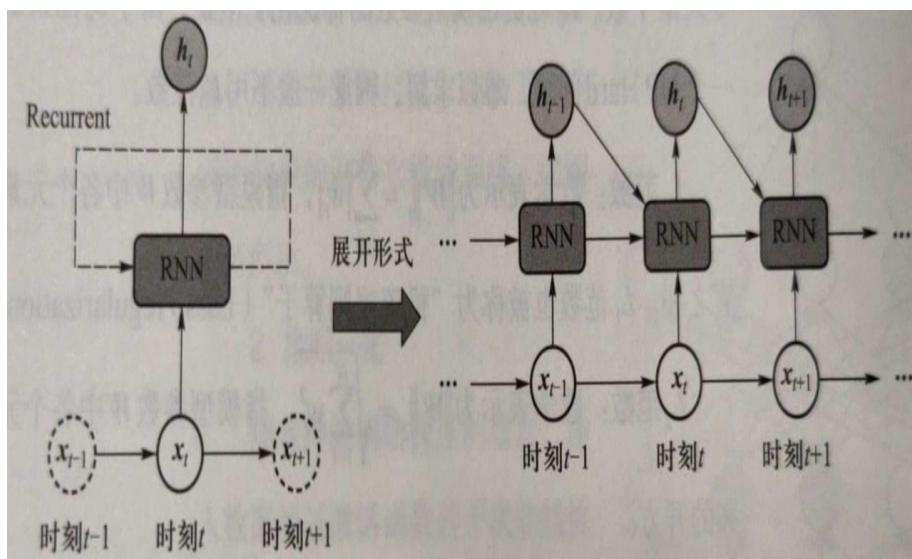


图6.17 循环神经网络结构及其展开形式示意图

为了更加直观展示在循环神经网络中前序时刻信息被“记住”来影响当前时刻信息编码，这里对上述公式进行如下变化：

$$\begin{aligned} h_t &= \Phi(U \times x_t + W \times h_{t-1}) = \Phi(U \times x_t + W \times \Phi(U \times x_{t-1} + W \times h_{t-2})) \\ &= \Phi \left(U \times \underbrace{x_t}_{t \text{ 时刻输入}} + W \times \Phi \left(U \times \underbrace{x_{t-1}}_{t-1 \text{ 时刻输入}} + W \times \Phi \left(U \times \underbrace{x_{t-2}}_{t-2 \text{ 时刻输入}} + \dots \right) \right) \right) \end{aligned}$$

从上式可见，当前时刻编码中，的确均包含了历史信息，这也说明了循环神经网络能够记忆的原因，使得一些需要记忆历史过往信息的任务能够被循环神经网络有效处理。

在时刻 t ，一旦得到当前输入数据 x_t ，循环神经网络会结合前一时刻 $t-1$ 得到的隐式编码 h_{t-1} ，如下产生当前时刻隐式编码 h_t ：

$$h_t = \Phi(U \times x_t + W \times h_{t-1})$$

这里 $\Phi(\cdot)$ 是激活函数，一般可为Sigmoid或者Tanh激活函数，使模型能够忘掉无关的信息，同时更新记忆内容。 U 与 W 为模型参数。从这里可看出，当前时刻的隐式编码输出 h_t 不仅仅与当前输入数据 x_t 相关，与网络已有的“记忆” h_{t-1} 也有着密不可分的联系。

循环神经网络

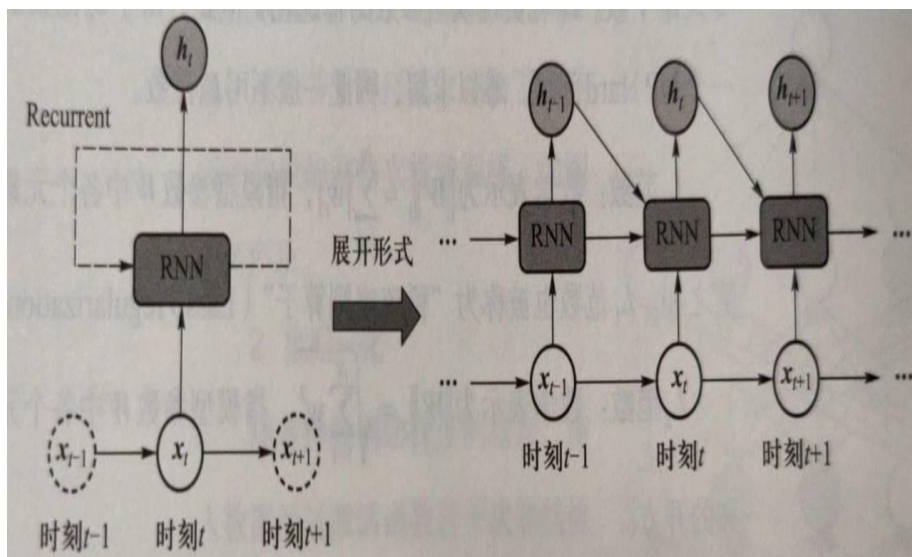


图6.17 循环神经网络结构及其展开形式示意图

- 按照时间将循环神经网络展开后，可以得到一个和前馈神经网络相似的网络结构。
- 这个网络结构可利用反向传播算法和梯度下降算法来训练模型参数，这种训练方法称为“沿时间反向传播算法（backpropagation through time, BPTT）”。
- 由于循环神经网络每个时刻都有一个输出，所以在计算循环神经网络的损失时，通常需要将所有时刻（或者部分时刻）上的损失进行累加。

$$h_t = \Phi(U \times x_t + W \times h_{t-1}) = \Phi(U \times x_t + W \times \Phi(U \times x_{t-1} + W \times h_{t-2}))$$

$$= \Phi \left(U \times \underbrace{x_t}_{t \text{ 时刻输入}} + W \times \Phi \left(U \times \underbrace{x_{t-1}}_{t-1 \text{ 时刻输入}} + W \times \Phi \left(U \times \underbrace{x_{t-2}}_{t-2 \text{ 时刻输入}} + \dots \right) \right) \right)$$

循环神经网络

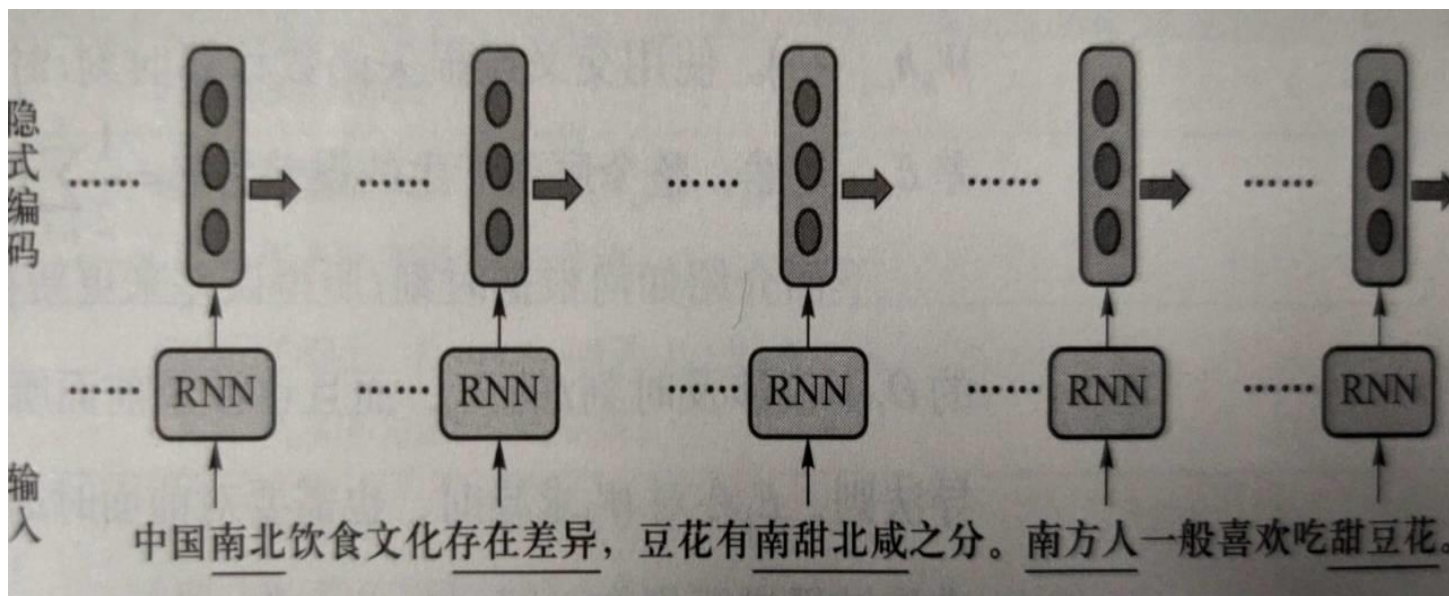


图6.18 循环神经网络中的应用示例

“南北”、“存在差异”、“南甜北咸”、“南方人”和“甜豆花”等单词之间存在前后依赖，这种依赖会被有效利用起来，得到各自单词的隐式编码以及句子的向量表达。

循环神经网络

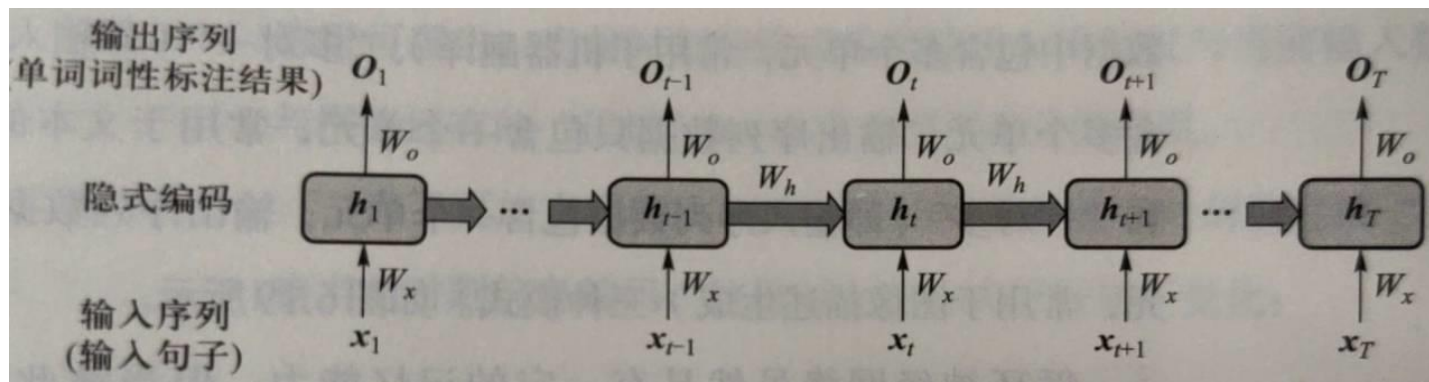


图6.20 循环神经网络中梯度传递示意图

以词性标注为例子给出了一个输入序列数据 $(x_1, \dots, x_{t-1}, x_t, x_{t+1}, \dots, x_T)$ 被循环神经网络处理的示意图。在图6.20中，时刻 t 输入数据 x_t （文本中的单词）的隐式编码为 h_t 、其真实输出为 y_t （单词词性）、模型预测 x_t 的词性是 O_t 。参数 W_x 将 x_t 映射为隐式编码 h_t 、参数 W_o 将 h_t 映射为预测输出 O_t 、 h_{t-1} 通过参数 W_h 参与 h_t 的生成。图中 W_x 、 W_o 和 W_h 是复用参数。

循环神经网络

假设时刻 t 隐式编码如下得到： $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$ 。使用交叉熵损失函数计算时刻 t 预测输出与实际输出的误差 E_t 。显然，整个序列产生的误差为 $E = \frac{1}{2} \sum_{t=1}^T E_t$ 。

下面介绍如何根据时刻 t 所得误差来更新参数 W_x 。在时刻 t 计算所得 O_t 不仅涉及到了时刻 t 的 W_x ，而且也涉及了前面所有时刻的 W_x ，按照链式求导法则， E_t 在对 W_x 求导时候，也需要对前面时刻的 W_x 依次求导，然后再将求导结果进行累加，即：

$$\frac{\partial E_t}{\partial W_x} = \sum_{i=1}^t \frac{\partial E_t}{\partial O_t} \frac{\partial O_t}{\partial h_t} \left(\prod_{j=i+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_i}{\partial W_x}$$

$$\text{其中 } \prod_{j=i+1}^t \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=i+1}^t \tanh' \times W_h$$

令 $t = 3$ ，则有：

$$\frac{\partial E_3}{\partial W_x} = \frac{\partial E_3}{\partial O_3} \frac{\partial O_3}{\partial h_3} \frac{\partial h_3}{\partial W_x} + \frac{\partial E_3}{\partial O_3} \frac{\partial O_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W_x} + \frac{\partial E_3}{\partial O_3} \frac{\partial O_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_x}$$

由于 \tanh 函数的导数取值位于0到1区间，对于长序列而言，若干多个0到1区间的小数相乘，会使得参数求导结果很小，引发**梯度消失问题**。 E_t 对 W_h 的求导类似，这里就不列出了。为了解决梯度消失问题，长短时记忆模型（Long Short-Term Memory, LSTM）被提出。

循环神经网络

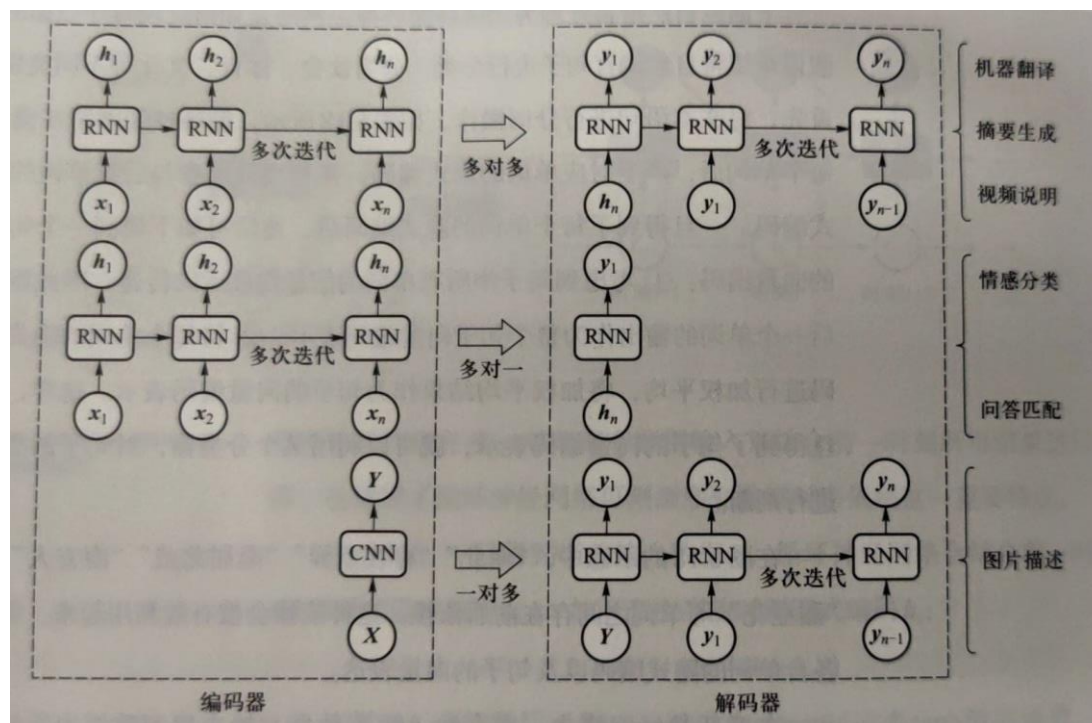


图6.19 输入与输出不同情况下循环神经网络结构示例

根据输入序列数据与输出序列数据中所包含“单元”的多寡，循环神经网络可以实现“多对多”（即输入和输出序列数据中包含多个单元，常用于机器翻译）、“多对一”（即输入序列数据包含多个单元、输出序列数据只包含一个单元，常用于文本的情感分类）和“一对多”（即输入序列数据包含一个单元、输出序列数据包含多个单元，常用于图像描述生成）三种模式。

长短时记忆网络

- 与简单的循环神经网络结构不同，长短时记忆网络（Long Short-Term Memory, LSTM）中引入了**内部记忆单元**（internal memory cell）和**门**（gates）两种结构来对当前时刻输入信息以及前序时刻所生成信息进行整合和传递。在这里，内部记忆单元中信息可视为对“历史信息”的累积。
- 常见的LSTM模型中有输入门(input gate)、遗忘门(forget gate)和输出门(output gate)三种门结构。对于给定的当前时刻输入数据 x_t 和前一时刻隐式编码 h_{t-1} ，输入门、遗忘门和输出门通过各自参数对其编码，分别得到三种门结构的输出 i_t 、 f_t 和 o_t 。
- 在此基础上，再进一步结合前一时刻内部记忆单元信息 c_{t-1} 来更新当前时刻内部记忆单元信息 c_t ，最终得到当前时刻的隐式编码 h_t 。

长短时记忆网络

表6.3 长短时记忆网络中所用符号及其含义

符号	内容描述或操作
x_t	时刻 t 的输入数据
i_t	输入门的输出: $i_t = \text{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$ (W_{xi} 、 W_{hi} 和 b_i 为输入门的参数)
f_t	遗忘门的输出: $f_t = \text{sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$ (W_{xf} 、 W_{hf} 和 b_f 为遗忘门的参数)
o_t	输出门的输出: $o_t = \text{sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$ (W_{xo} 、 W_{ho} 和 b_o 为输出门的参数)
c_t	内部记忆单元的输出: $c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ (W_{xc} 、 W_{hc} 和 b_c 为记忆单元的参数。输入门 i_t 控制有多少信息流入当前时刻内部记忆单元 c_t 、遗忘门控制上一时刻内部记忆单元 c_{t-1} 中有多少信息可累积到当前时刻内部记忆单元 c_t)
h_t	时刻 t 输入数据的隐式编码: $h_t = o_t \odot \tanh(c_t)$ $= o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c))$ (注: 输入门、遗忘门和输出门的信息 i_t 、 f_t 、 o_t 一起参与得到 h_t)
\odot	两个向量中对应元素按位相乘 (element-wise product)。 如: $[10 \ 6 \ 3 \ 7] \odot [0.2 \ 0.1 \ 0.8 \ 0.5] = [2 \ 0.6 \ 2.4 \ 3.5]$

长短时记忆网络

当给定序列数据中当前时刻 t 的输入数据 x_t 和前一时刻隐式编码 h_{t-1} 时，可按照如下公式来计算内部记忆单元信息和隐式编码：

$$\text{输入门信息输出: } i_t = \text{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$\text{遗忘门信息输出: } f_t = \text{sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

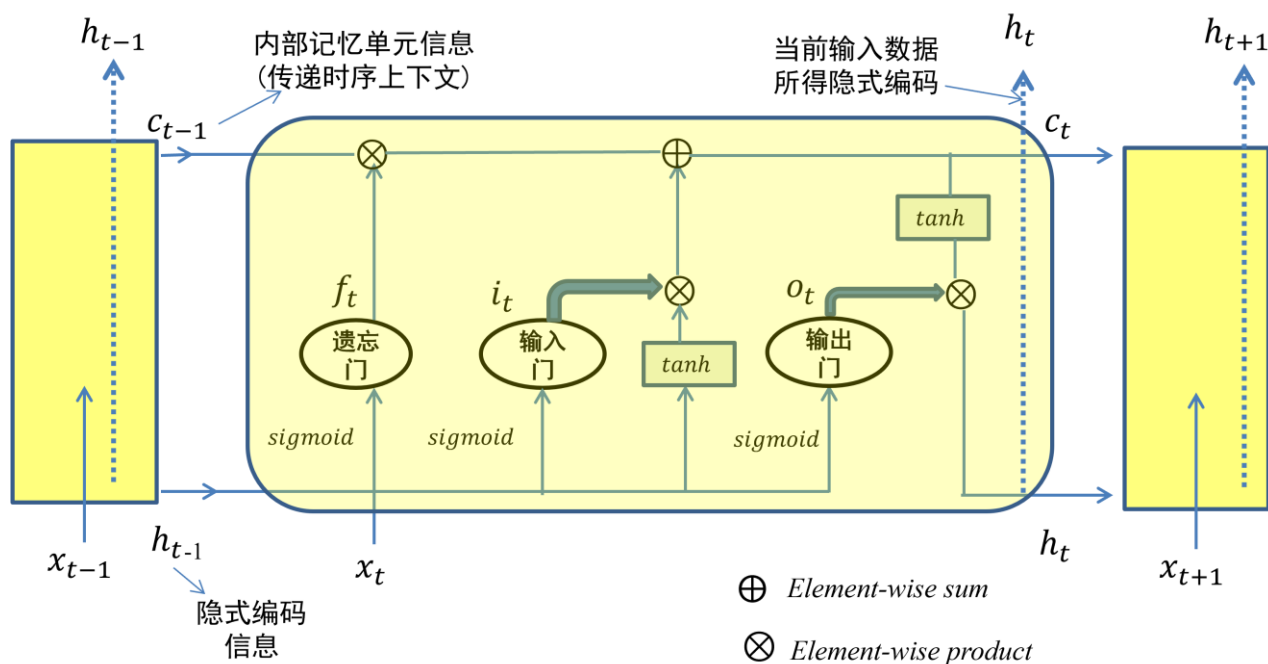
$$\text{输出门信息输出: } o_t = \text{sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$\text{内部记忆单元信息输出: } c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$\text{隐式编码输出: } h_t = o_t \odot \tanh(c_t) \text{ 或者 } h_t = o_t \odot \tanh(f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c))$$

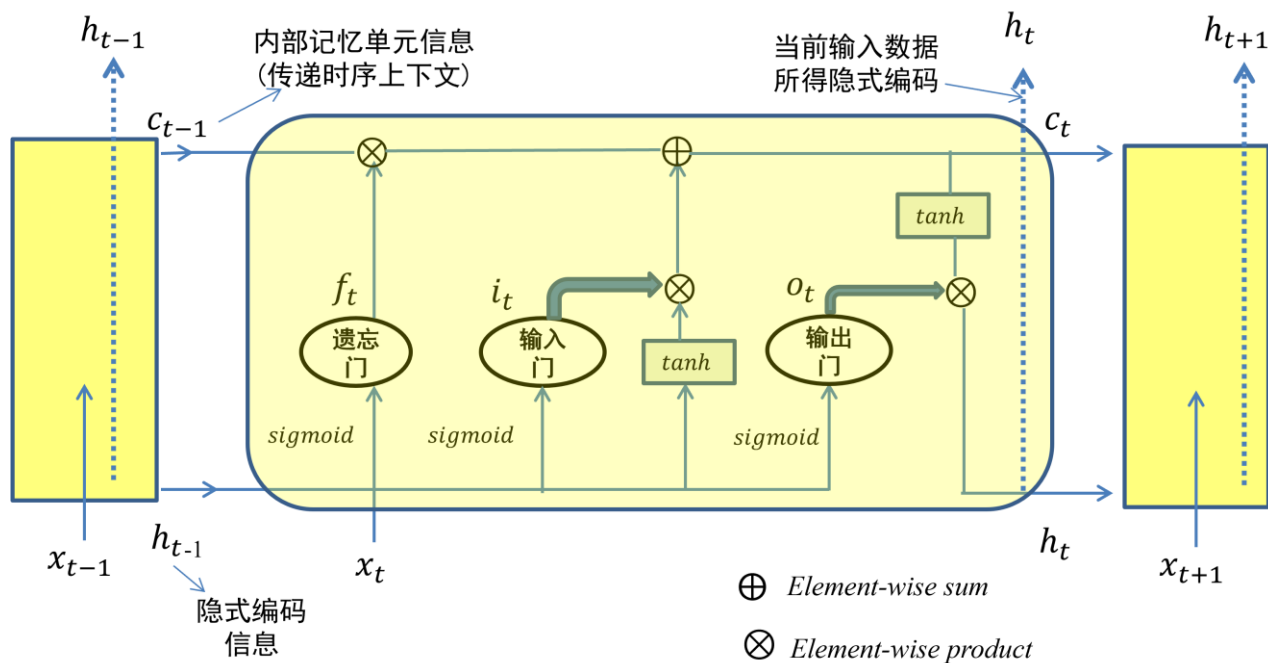
输入门、遗忘门和输出门通过各自参数对当前时刻输入数据 x_t 和前一时刻隐式编码 h_{t-1} 处理后，利用 sigmoid 对处理结果进行非线性映射，因此三种门结构的输出 i_t 、 f_t 和 o_t 值域为 $(0,1)$ 。正是由于三个门结构的输出值为位于0到1之间的向量，因此其在信息处理中起到了“调控开关”的“门”作用。三个门结构所输出向量的维数、内部记忆单元的维数和隐式编码的维数均相等。

长短时记忆网络



- ◆ 在每个时刻 t 中只有内部记忆单元信息 c_t 和隐式编码 h_t 这两种信息起到了对序列信息进行传递的作用。每个时刻 t 只有 h_t 作为本时刻的输出以用于分类等处理。
- ◆ 以输入门 $i_t = \text{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$ 为例，在 sigmoid 函数作用下，假设输入门 i_t 的取值为 $(0.05, 0.1, 0.8)$ ，则通过 $i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ 中按位相乘 \odot 操作，输入门会“阻断”一些信息进行后续操作（如0.05和0.1所对应按位相乘的信息）、也会“放入”一些信息进行后续操作（如0.8所对应按位相乘的信息）。当然，如果门的某一位取值为0，表示这一位所对应信息为全闭、禁止信息通过；如果门的某一位取值为1，表示这一位所对应的信息为全开、容许信息全部通过。遗忘门不容许一些信息后续传递，从另外一个侧面而言是为了关注需要关注信息。

长短时记忆网络



- 从 $c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ 可以看出，对当前时刻 t 所对应内部记忆单元中信息的调整涉及到遗忘门信息、 $t-1$ 时刻所对应内部记忆单元中信息、输入门信息和 $t-1$ 时刻的隐式编码。
- 由于 sigmoid 函数的值域在 $(0,1)$ 之间，输入门、遗忘门和输出门三种门结构采用 sigmoid 这一非线性映射函数而起到了信息“控制门”的作用。在内部记忆单元和隐式编码中，使用了 \tanh 这一非线性映射函数而没有继续使用 sigmoid 非线性映射函数，其原因在于 \tanh 函数的值域为 $(-1,1)$ ，使得 \tanh 函数在进行信息整合时可起到信息“增（为正）”或“减（为负）”的效果。

长短时记忆网络

LSTM如何克服梯度消失

对于 $c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ ，有如下求导结果的存在：

$$\frac{\partial c_t}{\partial c_{t-1}} = f_t + \frac{\partial f_t}{\partial c_{t-1}} \times c_{t-1} + \dots\dots$$

可见， $\frac{\partial c_t}{\partial c_{t-1}}$ 求导的结果至少大于等于 f_t ，即遗忘门的输出结果。如果遗忘门选择保留旧状态，则这一求导结果就接近1（或者接近向量1），使得梯度是存在的，从而避免了梯度消失问题。也就是说，LSTM通过引入门结构，在从 t 到 $t+1$ 过程中引入加法来进行信息更新，避免了梯度消失问题。

整体来看，内部记忆单元信息 c_t 和隐式编码 h_t 这两种信息起到了对序列信息进行传递的作用。内部记忆单元信息 c_t 好比人脑的长时记忆、隐式编码 h_t 代表了短时记忆。

门控循环单元神经网络

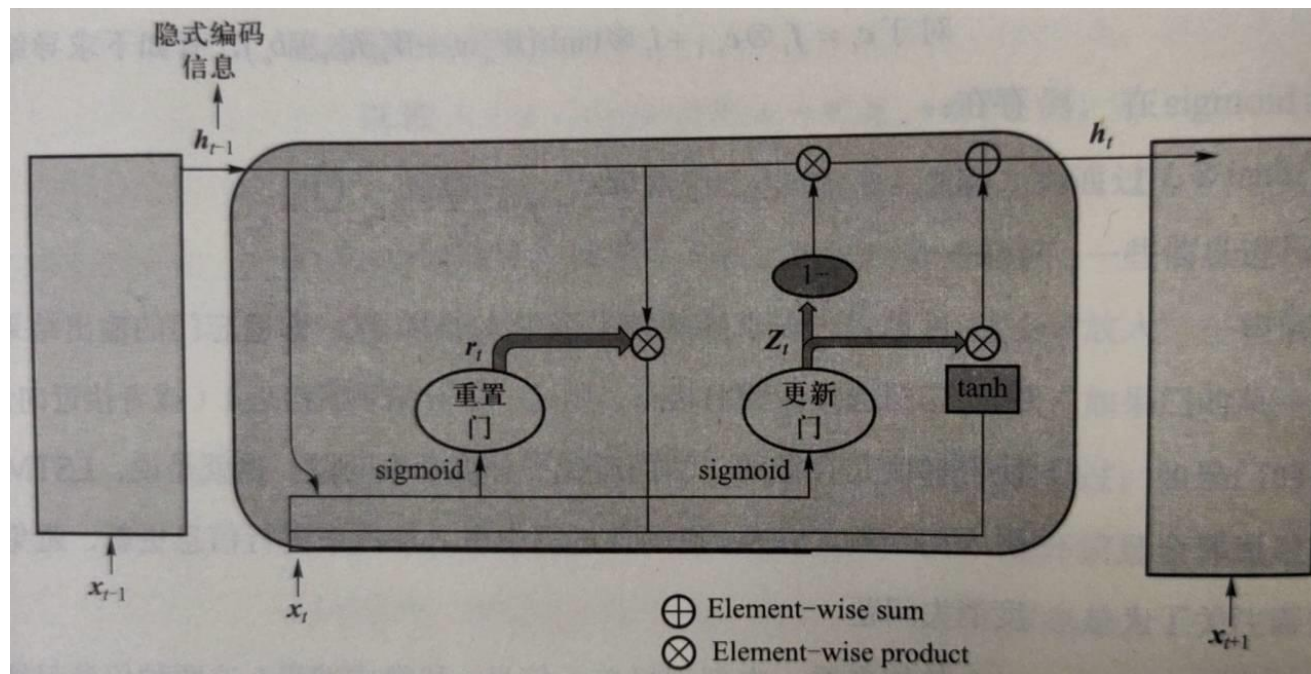


图6.22 GRU网络结构图

提纲

一、深度学习的历史发展

二、前馈神经网络

三、卷积神经网络

四、循环神经网络

五、深度生成学习

六、深度学习应用

深度生成学习模型 (deep generative learning model)

在本章之前的介绍中，神经网络模型从数据中提取出高层语义在数据中所蕴含的“模式”，并利用这些模式实现对数据的分类和检测等，这种模型通常称为判别模型，判别模型不关心数据如何生成，它只关心数据蕴含哪些模式以及如何将数据进行分类。与之相对的模型类型被称为生成模型(generative model)，生成模型需要学习目标数据的分布规律，以合成属于目标数据空间的新数据。

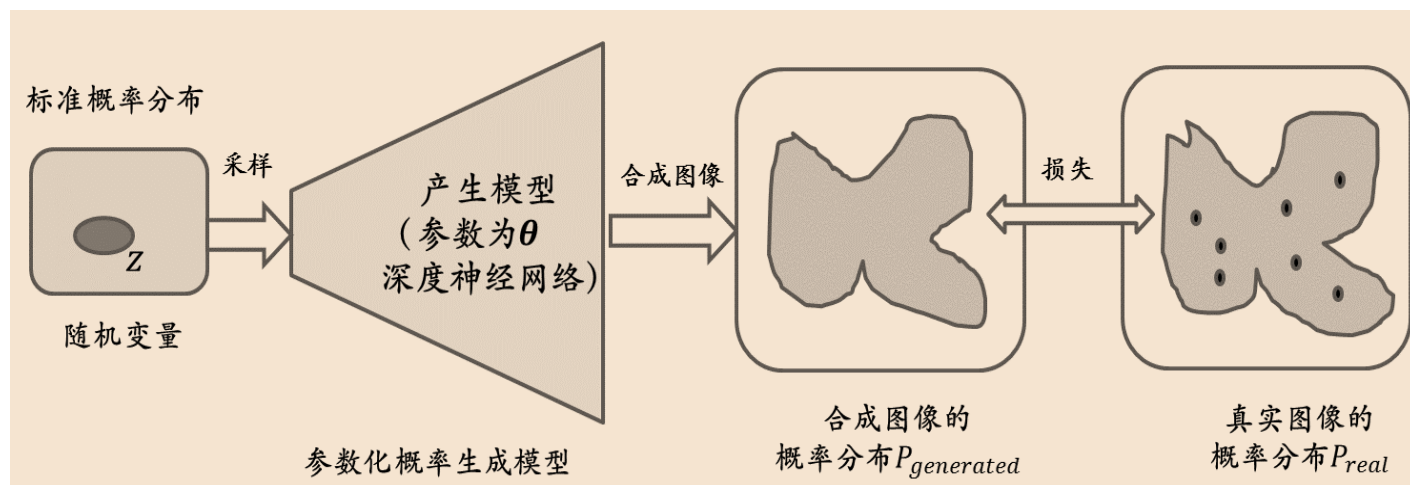


图6.23 生成模型的示意图

变分自编码器 (variational auto-encoder, VAE)、自回归模型 (Autoregressive models) 与生成对抗网络 (generative adversarial network, GAN) 等.....

生成对抗网络 (deep generative learning model)

生成对抗网络是由Ian Goodfellow等人于2014年提出的一种生成模型[Goodfellow2014]，该模型可视为两个神经网络相互竞争的零和游戏（zero-sum game），“以子之矛，陷子之盾”，最终达到纳什均衡。

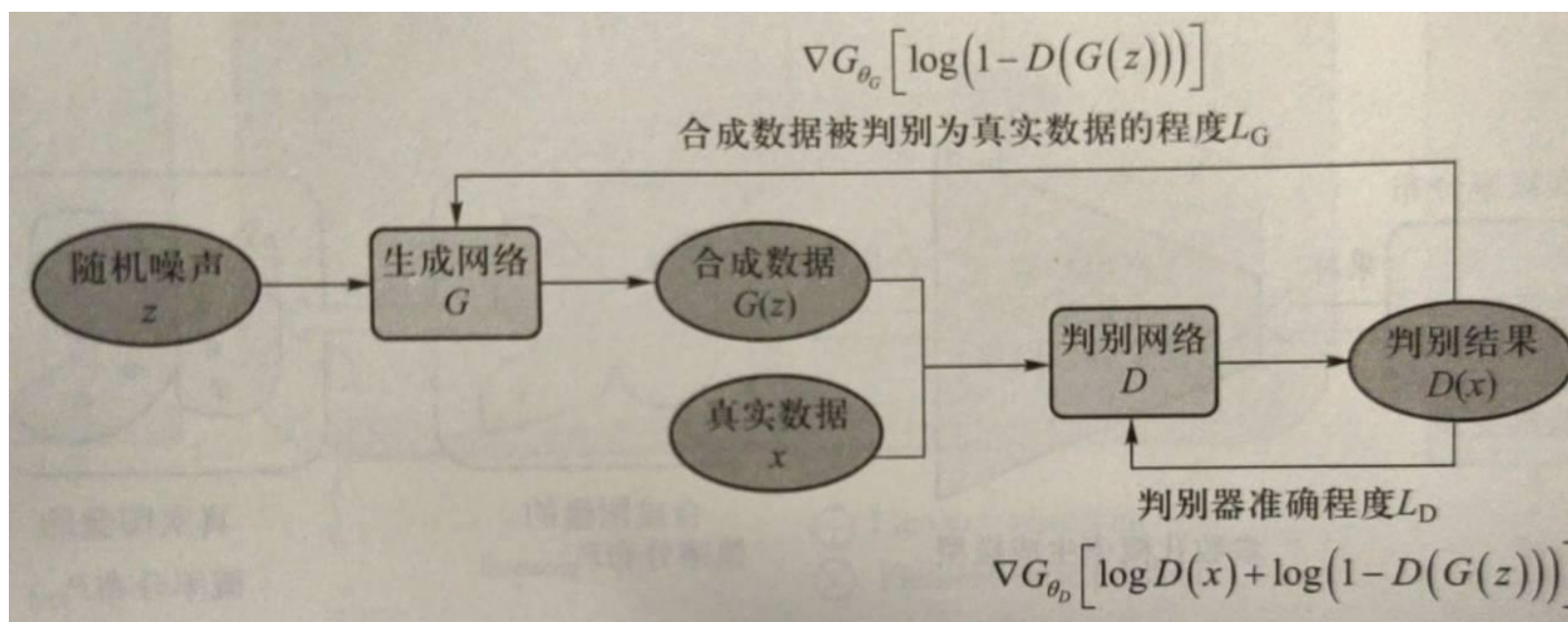


图6.24 生成对抗网络模型示意图

生成对抗网络 (deep generative learning model)

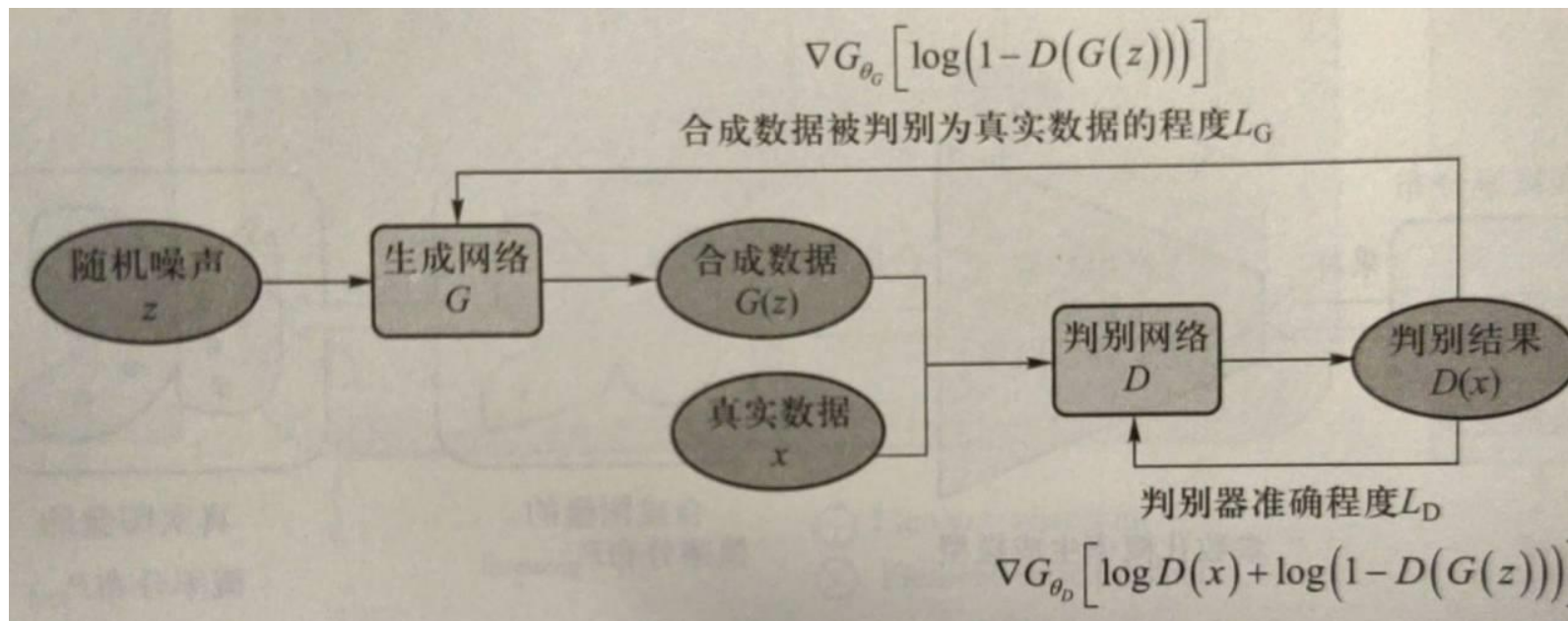


图6.24 生成对抗网络模型示意图

生成对抗网络由一个生成器 (generator, 简称G) 和一个判别器 (discriminator, 简称D) 组成。GAN的核心是通过生成器和判别器两个神经网络之间的竞争对抗, 不断提升彼此水平以使得生成器所生成数据 (人为伪造数据) 与真实数据相似, 判别器无法区分真实数据和生成数据。

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

生成对抗网络 (deep generative learning model)

算法6.5.1 训练生成对抗网络

输入：神经网络 D 、 G ，噪声分布 $p_z(z)$ ，真实数据分布 $p_{data}(x)$

输出：神经网络参数 θ_d 、 θ_g

算法步骤：

每轮训练循环执行：

训练 k 轮判别器：

从噪声分布 $p_z(z)$ 中采样 m 个样本 $\{z^{(1)}, z^{(2)}, z^{(3)} \dots z^{(m)}\}$

从真实数据分布 $p_{data}(x)$ 中采样 m 个样本 $\{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$

沿梯度上升方向更新判别器参数：

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

从噪声分布 $p_z(z)$ 中采样 m 个样本 $\{z^{(1)}, z^{(2)}, z^{(3)} \dots z^{(m)}\}$

沿梯度下降方向更新判别器参数：

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)})))$$

在训练初期，因为生成数据质量较低，判别器 D 可以轻易将其区分，此时 $\log(1 - D(G(z)))$ 无法为生成器 G 提供显著的梯度，因此可以使用 $-\log D(G(z))$ 来代替损失函数。这么做并不会影响其能够收敛至最优解的性质，且可以在训练初期提供足够的梯度。

在后续的应用中，发现GAN具有收敛困难，且非常依赖于合适的交替训练轮次选择等不足。为此，Martin Arjovsky等人提出了Wasserstein GAN[Arjovsky 2017]，对模型结构进行了一些修改。在Wasserstein GAN中，模型输出不使用以概率值形式输出的sigmoid激活函数，而是直接根据输出值大小评价其效果。相应地，目标函数中也不需要再取对数：

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [D(x)] - \mathbb{E}_{z \sim p_z(z)} [D(G(z))]$$

条件生成对抗网络

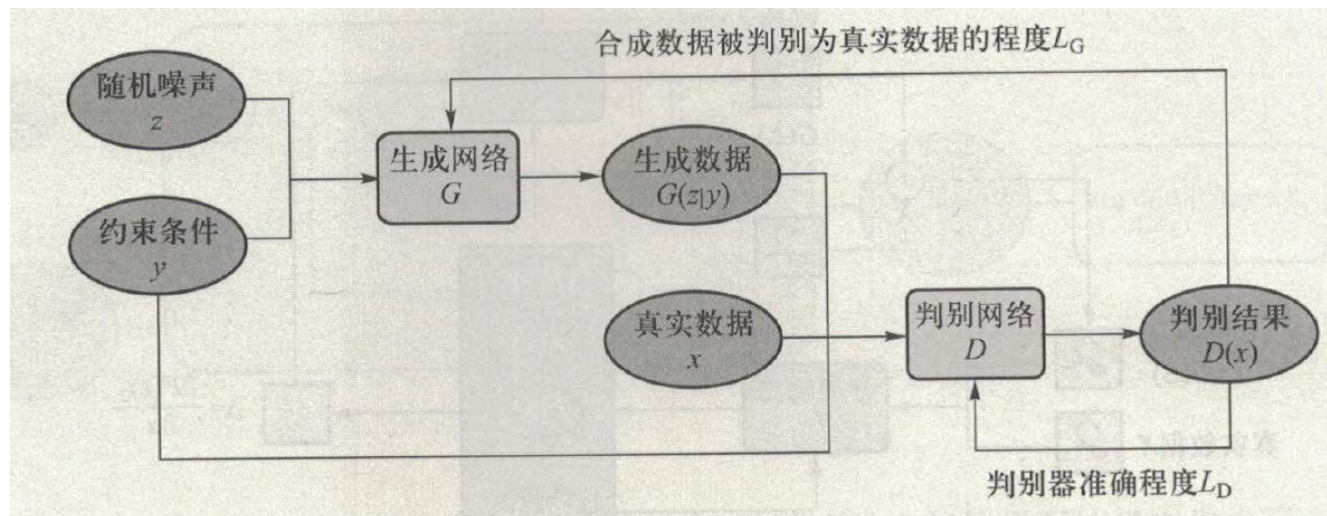


图6.25 条件生成对抗网络模型示意图

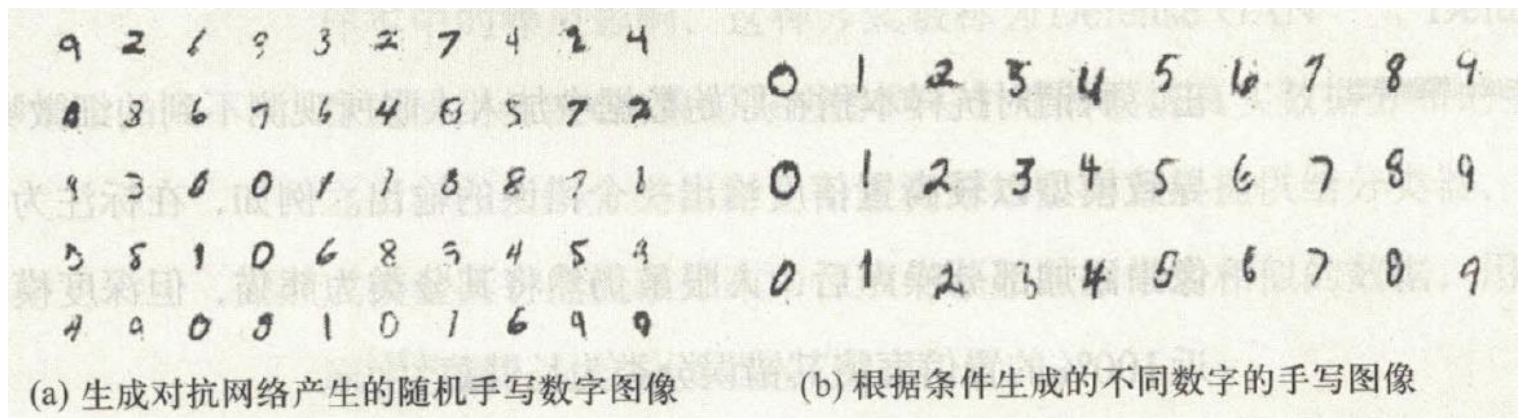


图6.26 不同生成对抗网络模型产生的手写数字图像

用生成对抗网络抵御对抗样本攻击

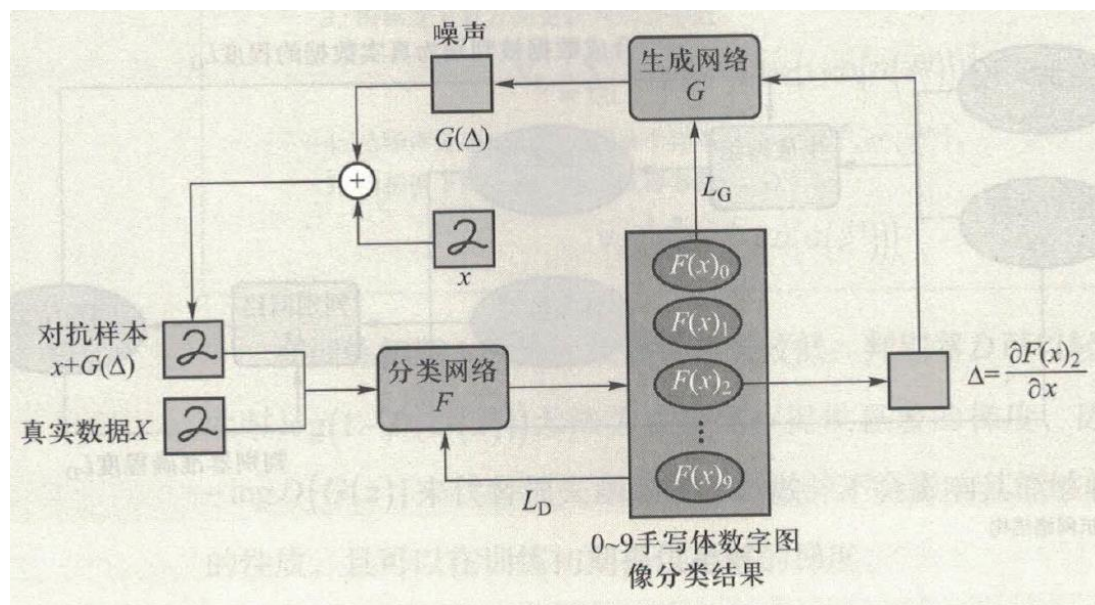


图6.27 对抗生成训练器模型

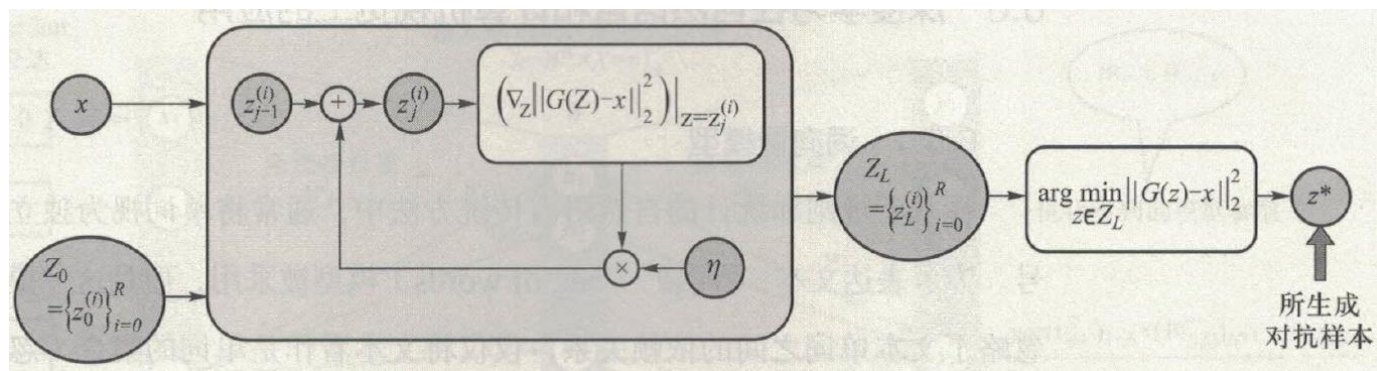


图6.29 Defense GAN模型示意图

用生成对抗网络抵御对抗样本攻击

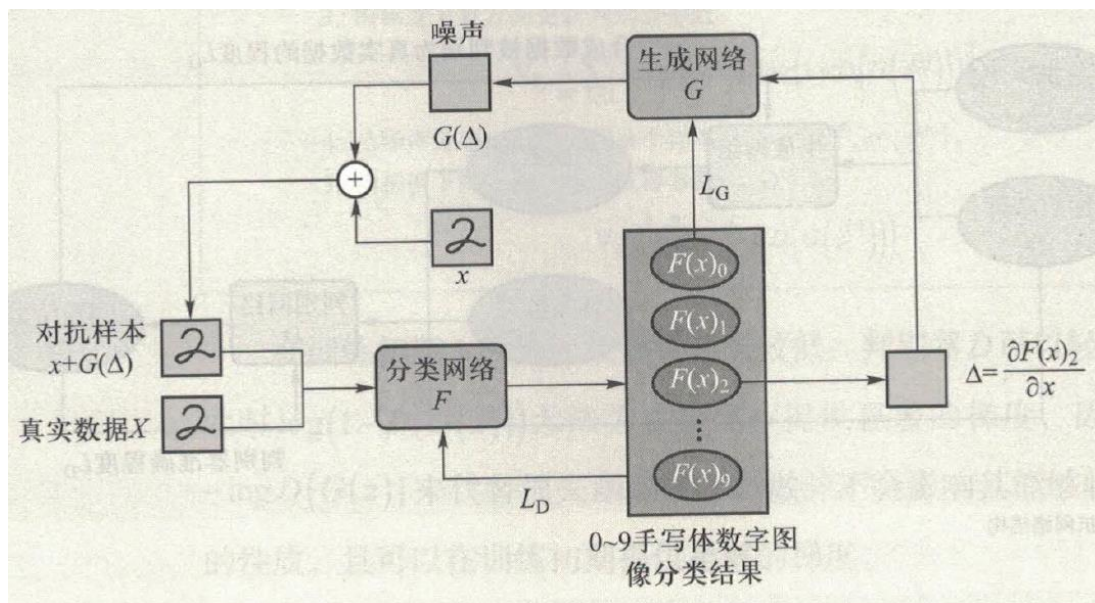


图6.27 对抗生成训练器模型

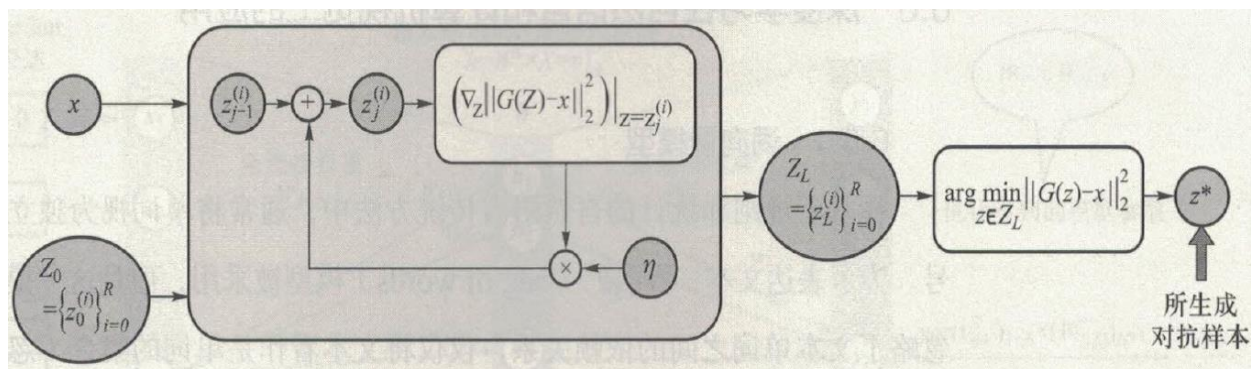
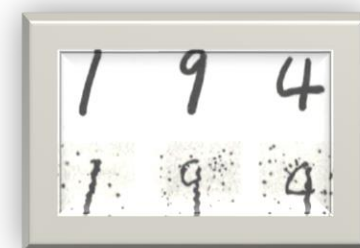


图6.29 Defense GAN模型示意图

提纲

一、深度学习的历史发展

二、前馈神经网络

三、卷积神经网络

四、循环神经网络

五、深度生成学习

六、深度学习应用

自然语言中词向量生成

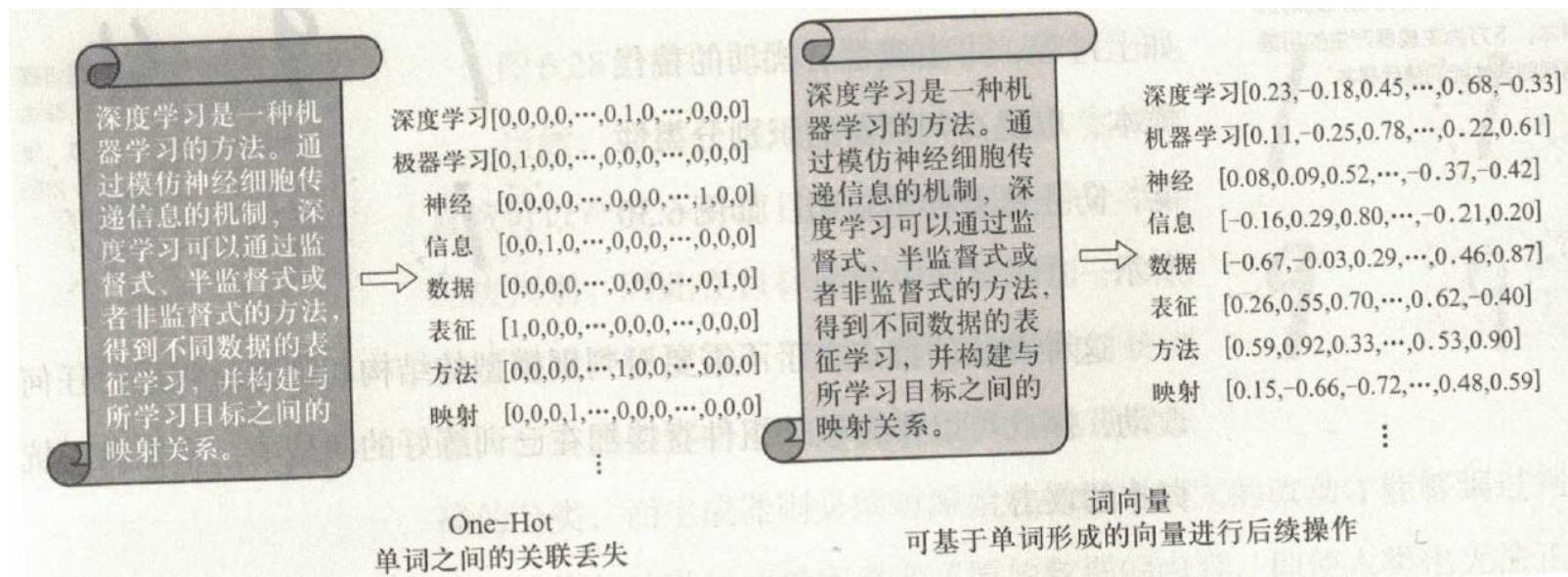


图6.31 词典模型与词向量模型表示文本

为了表达文本，“词袋”（Bag of Words）模型被采用，但是这一模型忽略了文本单词之间的依赖关系，仅仅将文本看作是单词的集合（忽略了单词之间的先后次序）。在词袋模型中，一个单词按照词典序被表示为一个词典维数大小的向量（被称为one-hot vector），向量中该单词所对应位置按照其在文档中出现与否取值为1或0。1表示该单词在文档中出现、0表示没有出现。每个单词被表达为单词向量后，一些与自然语言相关工作（如聚类、同义词计算、主题挖掘等）可以依次开展。

自然语言中词向量生成

- 使用词袋模型来表示单词时，往往会遭遇维度灾难的问题。除此之外，这种表达方法无法有效计算单词与单词之间的语义相似度。比如，“高兴”与“愉快”两个单词的向量表达会很不相同，虽然其具有很强的语义相似性。
- 为了刻画不同单词之间的语义相关性，研究人员希望使用一种分布式向量表达 (distributed vector representation) 对不同单词进行表达。利用深度学习模型，可将每个单词表征为 N 维实数值的分布式向量。这样一来可以把对文本内容分析简化为 N 维向量空间中的向量运算，如两个单词在向量空间上所计算结果可用来衡量这两个单词之间的相似度。用深度学习算法生成每个单词的向量表达，词向量 (Word2Vec) 是较为经典的模型。

Word2Vec 模型

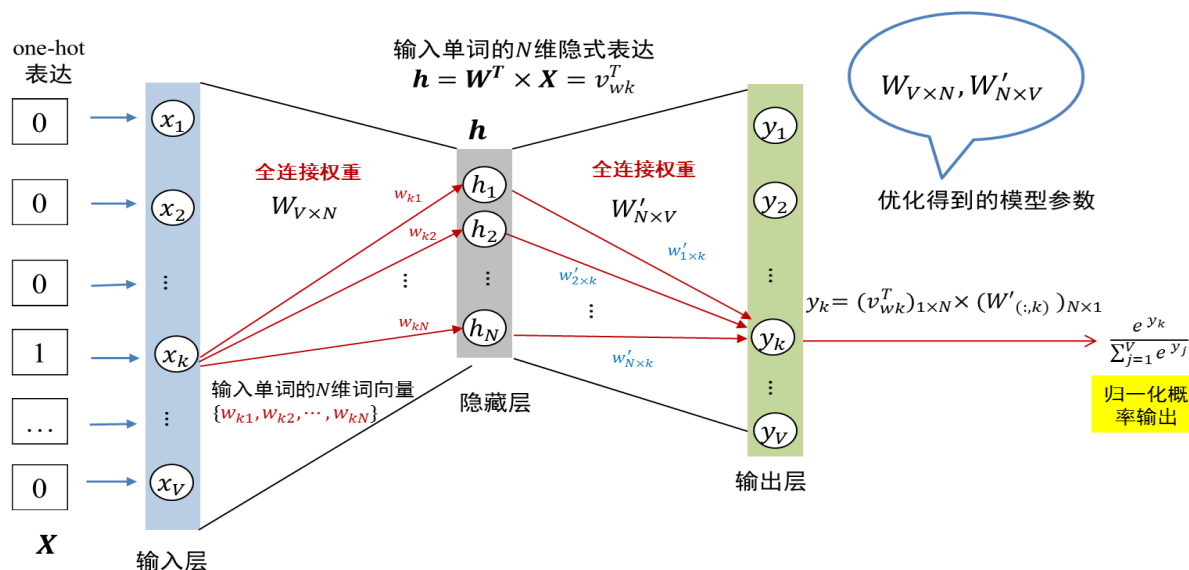


图6.32 单个单词的词向量(WordVec)生成示意图

假设词典中有 V 个不同的单词，现在考虑如何生成第 k 个单词的 N 维词向量（如图6.32所示）。首先，将该单词表示成 V 维one-hot 向量 X ，向量 X 中第 k 个位置取值为1、其余位置取值均为0。隐藏层神经元大小为 N ，每个神经元记为 $h_i (1 \leq i \leq N)$ 。向量 X 中每个 $x_i (1 \leq i \leq V)$ 与隐藏层神经元是全连接，连接权重矩阵为 $W_{V \times N}$ 。输出层是 V 维归一化的概率值，其中 y_k 对应第 k 个单词归一化的概率值，显然其取值应该远大于其它输出所对应的归一化概率值。

Word2Vec 模型

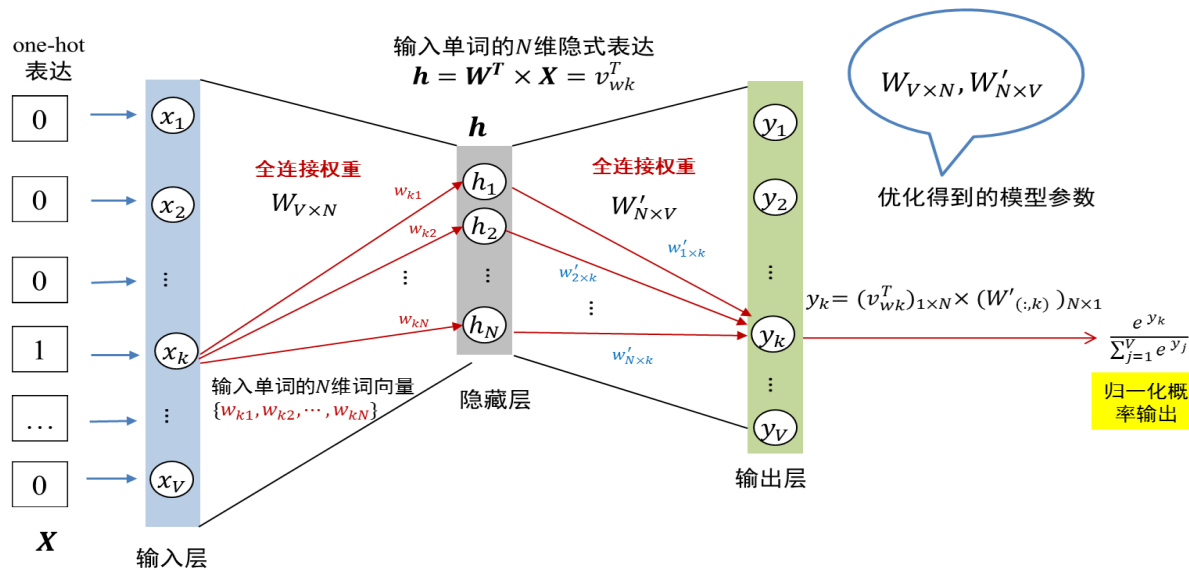


图6.32 单个单词的词向量(WordVec)生成示意图

在图6.32中，输入层中 x_k 与隐藏层连接权重为 $\{w_{k1}, w_{k2}, \dots, w_{kN}\}$ ，输入单词的 N 维隐式表达 $h = W^T \times X = v_{wk}^T$ 。所得到的隐藏层结果与输出层是全连接，全连接权重矩阵为 $W'_{N \times V}$ 。从图中可知，第 k 个单词对应的输出为 $y_k = (v_{wk}^T)_{1 \times N} \times (W'_{(:,k)})_{N \times 1}$ 。这里 $W'_{(:,k)}$ 指的从连接权重矩阵 $W'_{N \times V}$ 中取出第 k 列的向量。接着对 y_k 通过Softmax函数进行归一化，得到第 k 个单词对应的归一化概率输出。显然，第 k 个单词归一化的概率值应该远远大于输出层中其他位置所对应的归一化概率值。

这里 $W_{V \times N}$ 和 $W'_{N \times V}$ 为模型参数。一旦训练得到了图6.5.2的神经网络，就可以将输入层中 x_k 与隐藏层连接权重为 $\{w_{k1}, w_{k2}, \dots, w_{kN}\}$ 作为第 k 个单词 N 维词向量(word vector)。

Word2Vec 模型

为了训练词向量模型，可将训练目标如下进行形式化描述：

$$\begin{aligned}\text{word2vec model} &= \max P(\text{输出表达} | \text{输入表达}) \\ &= \max \log \left(\frac{e^{y_k^*}}{\sum_{j=1}^V e^{y_j}} \right) = \max (y_k^* - \log \sum_{j=1}^V e^{y_j})\end{aligned}$$

其中星号 (*) 表示预测值。对上式取其相反数，令其最小，则得到损失函数：

$$\text{loss} = -y_k^* + \log \sum_{j=1}^V e^{y_j}$$

这样就可以利用梯度下降和误差后向传播来优化训练参数 $W_{V \times N}$ 和 $W'_{N \times V}$ 了。

通常Word2Vec的模型参数有两种训练模式。一种是Continuous Bag-of-Words (CBoW)，即根据某个单词所处的上下文单词来预测该单词。另一种是Skip-gram，即利用某个单词来分别预测该单词的上下文单词。

CBOW 模型

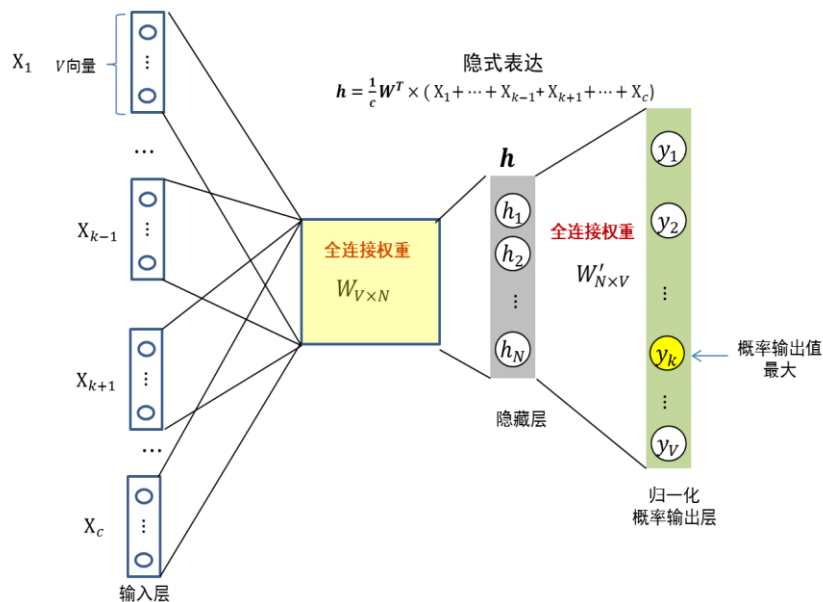


图6.33 基于CBOW模型(continuous bag-of-words model)的单词向量(Word2Vec)生成示意图

图6.33给出了基于CBOW来得到词向量的示意图。给出包含 c 个单词的句子 $\{x_1, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_c\}$ 。在CBOW中，可用 x_k 的前序 $k-1$ 个单词和后续单词 $c-k$ 个单词一起来预测 x_k 。每个输入单词仍然是 V 维向量（可以是one-hot的表达），隐式编码为每个输入单词所对应编码的均值

$$h = \frac{1}{c-1} W^T \times (x_1 + \dots +$$

$x_{k-1} + x_{k+1} + \dots + x_c)$ 。对于计算得到的隐式编码，要求其输出归一化概率中，单词 x_k 所对应的 y_k 取值最大，即用 x_k 的上下文来预测 x_k 。定义好相应的损失函数，可训练得到参数 $W_{V \times N}$ 和 $W'_{N \times V}$ 。

CBOW 模型

无论是CBOW还是Skip-Gram模式，随着单词个数数目增加，模型参数数目也迅速上升。例如：如果单词个数为100，希望每个单词所对应的词向量维度为300，则模型隐藏层的神经元数量为300、模型连接权重参数为 $2 \times 100 \times 300 = 60,000$ ；如果单词个数为1,000，则模型参数为 $2 \times 1000 \times 300 = 600,000$ 。

为此，Word2Vec模型在训练过程中使用了两个巧妙方法来加快模型训练，分别为层次化Softmax（Hierarchical Softmax）与负采样（Negative Sampling）。

图像分类和目标定位

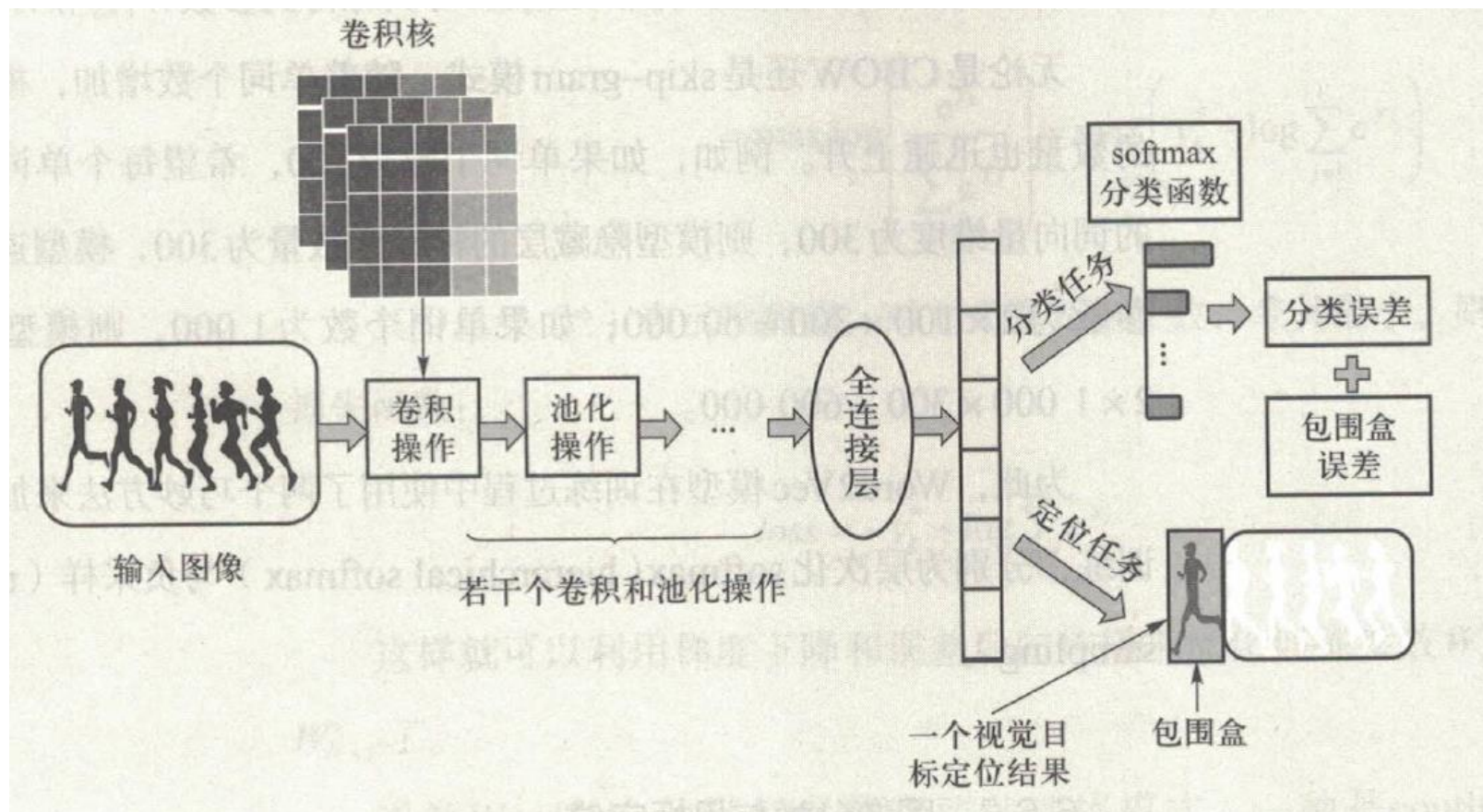


图6.34 利用卷积神经网络进行图像分类与目标定位示意图