第九章　人工神经网络

9.1　简介

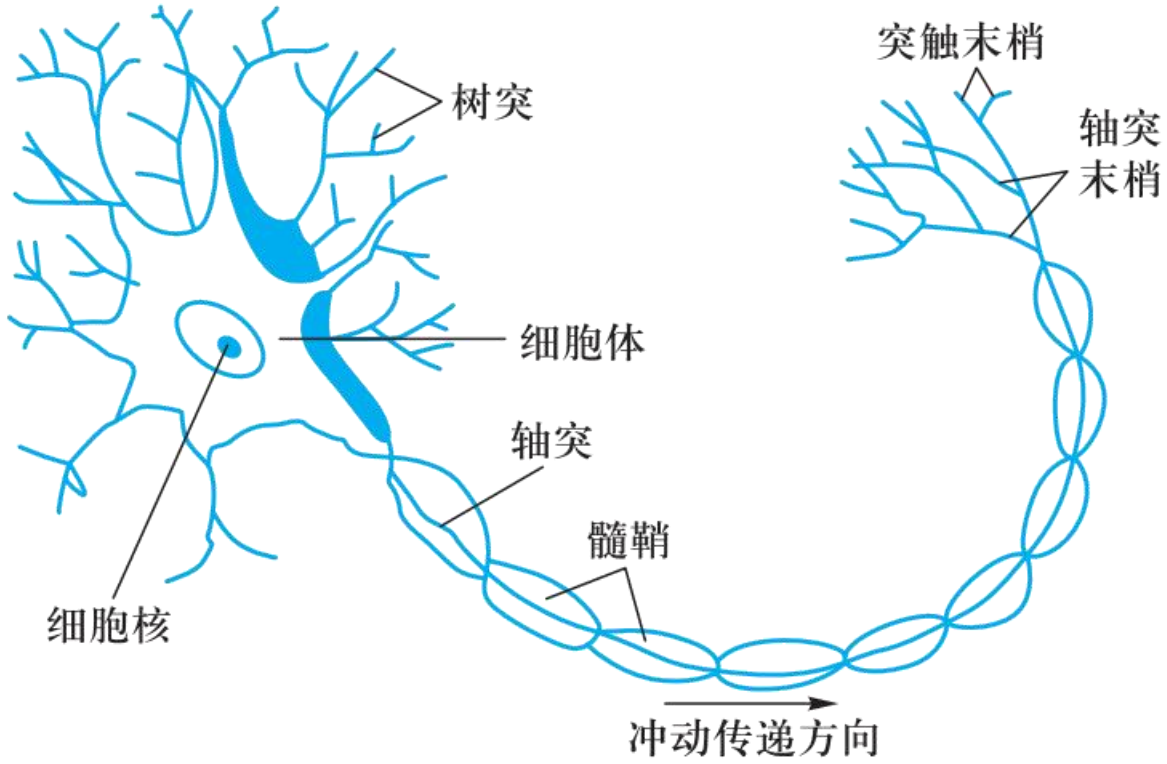
9.1 简 介

- 1943 年, 心理学家麦卡洛克 (W.McCulloch) 和数理逻辑学家皮茨 (W.Pitts)在分析和总结神经元 (neuron) 基本特性的基础上, 首先提出了神经元的数学模型[104]. 到目前为止, 人工神经网络 (后面简称为神经网络) 已经发展成了一个相当大的、多学科交叉的学科领域. 对神经网络的定义已有很多, 本书采用了科霍嫩 (Kohonen) 在 1988 年提出的定义, 即 “神经网络是由具有适应性的简单单元组成的广泛并行互连的网络, 它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应”[105]. 在机器学习中谈论神经网络时指的是 “神经网络学习”, 或者, 是机器学习与神经网络这两个学科领域的交叉部分.

9.2　人工神经元

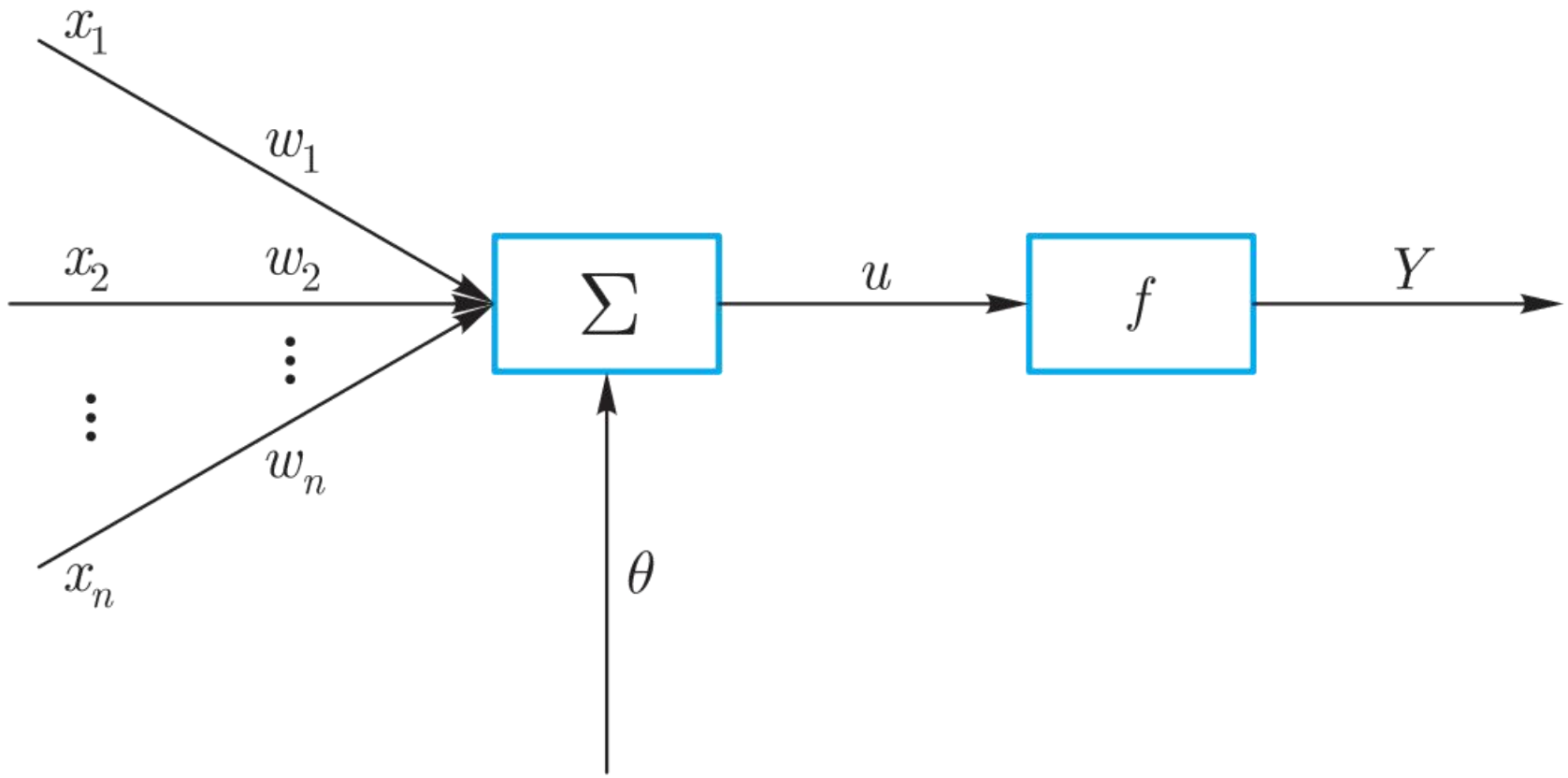
9.2 人工神经元

- 神经网络起源于对生物神经元的研究, 如图 9.1 所示生物神经元包括细胞体、树突、轴突等部分, 其中树突是用于接收输入信息, 输入信息经过突触处理, 当达到一定条件时通过轴突传出, 此时神经元处于激活状态; 反之没有达到相应条件, 则神经元处于抑制状态.



9.2 人工神经元

- 神经网络的基本节点是人工神经元 (后面简称为神经元), 其工作原理是仿照生物神经元提出的, 如 9.2 所示. 输入值经过加权和偏置后, 由激活函数处理后输出.



9.2 人工神经元

- 从图 9.2 所示的神经元示意图来看, 对某一个神经元来说, 它可以同时接受 n 个输入信号, 分别用 $x_1, x_2, \cdots, x_n$ 来表示. 每个输入端与神经元之间有联接权值 $\omega_1, \omega_2, \cdots, \omega_n$, 神经元的总输入为对每个输入进行加权求和, 同时加上偏置 θ, 即

$$u = \sum_{i=1}^{n} \omega_i x_i + \theta. \tag{9.2.1}$$

- 神经元的输出 Y 是对 u 的映射,

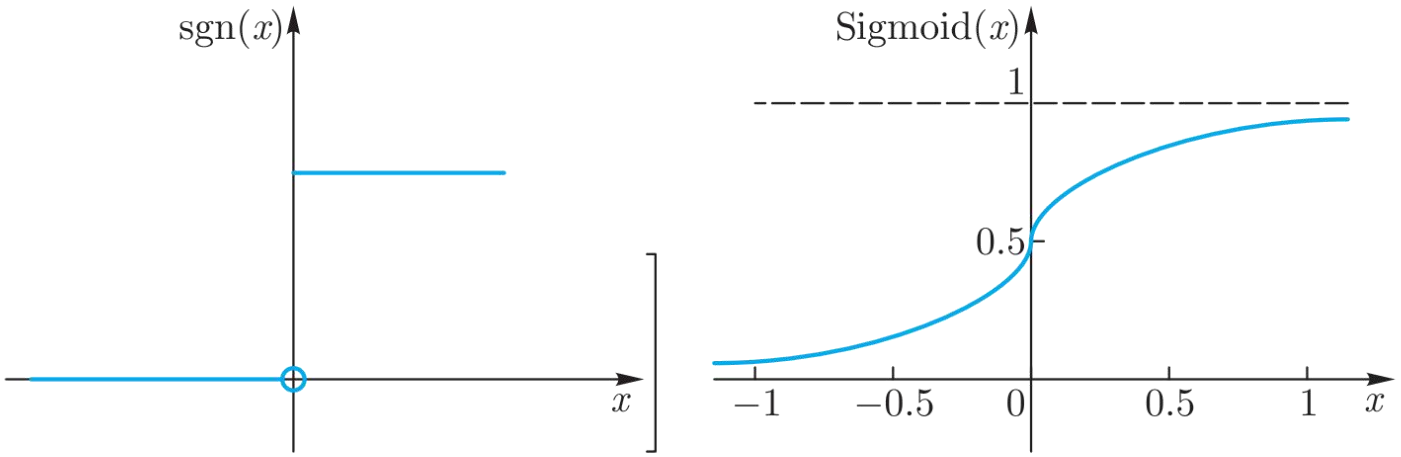
$$Y = f(u) = f\left(\sum_{i=1}^{n} \omega_i x_i + \theta \right), \tag{9.2.2}$$

  - 其中, f 为激活函数. 注意, 在神经网络章节部分, 我们沿用计算机的数学符号, n 表示变量个数 (对应前面的 p), m 表示样本量 (对应前面的 n), 以及样本向量 $(x_1, \cdots, x_n)$ 和 $(X_1, \cdots, X_p)$ 与前面的理解是一样的.

9.2 人工神经元

- 激活函数能够给神经元引入非线性因素, 使得神经网络可以逼近未知非线性函数, 这样神经网络就可以应用到更多的非线性模型中. 而理想的激活函数包括像图 9.3(a) 中 sgn(x) 函数那样的跃阶函数, 可以将输入神经元的值映射为 “抑制” 或 “兴奋” 的输出, 也就是 0 和 1[106]. 其中, $\operatorname{sgn}(x)=\begin{cases}1, & x \geq 0\\ 0, & x<0\end{cases}$.

  然而跃阶函数的性质比较差, 它是不连续、不光滑的, 因此实际常用 Sigmoid 函数作为激活函数, 如图 9.3(b), 它把输入神经元的值映射到了 (0,1) 的范围内进行输出. 其中, $\operatorname{sigmoid}(x)=\dfrac{1}{1+\mathrm{e}^{-x}}$.



(a) 跃阶函数　　(b) Sigmoid函数

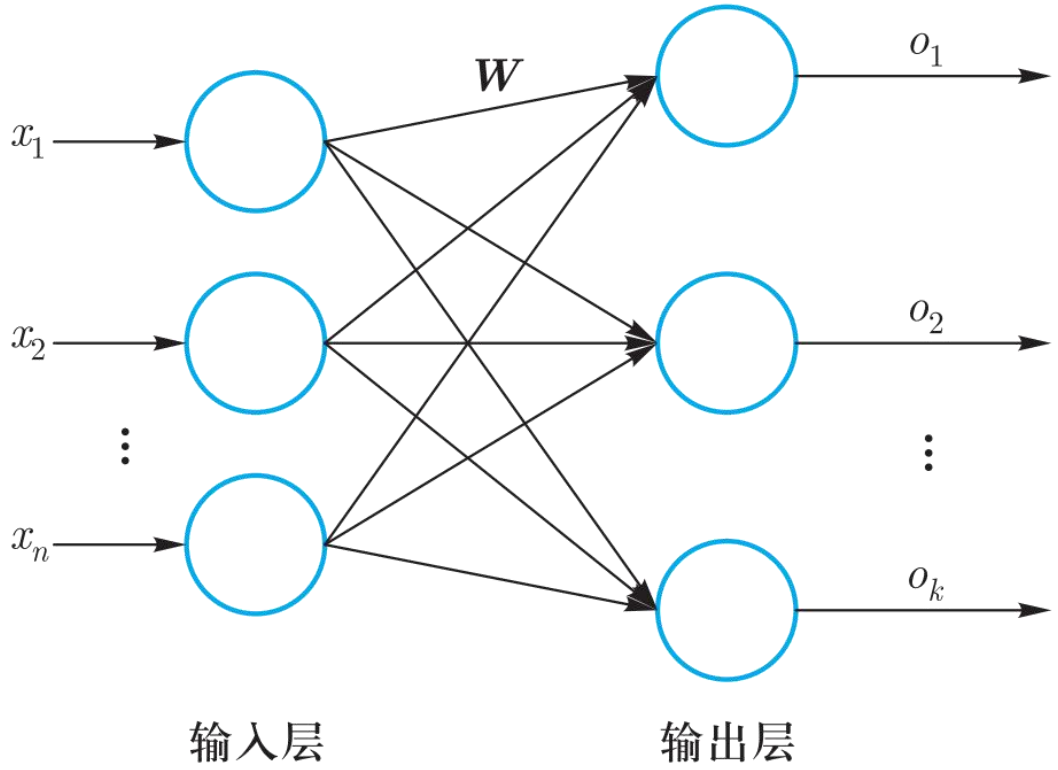
9.3　前馈神经网络

9.3 前馈神经网络

- 在实际问题中, 不可能只使用一个神经元, 会使用非常多的神经元进行计算. 一般情况下, 神经元分层排列, 每个神经元只接受前一层的输入, 并输出到下一层, 网络没有下一层到上一层的回路信号, 这种网络称为前馈神经网络 (feedforward neural networks). 前馈并不意味着网络中信号不能向后传, 而是指网络拓扑结构上不存在环或回路. 前馈神经网络的第一层为输入层 (input layer), 最后一级为输出层 (output layer), 输入层与输出层之间的各层统一称为隐含层 (hidden layer). 隐含层和输出层神经元都是拥有激活函数的功能神经元. 一个网络可以只包含一个隐含层, 也可以包含多个隐含层. 在这个意义下感知器可以理解为一种常见的前馈神经网络.

9.3.1 单层感知器

- 感知器, 也称感知机, 是弗兰克·罗森布拉特 (Frank Rosenblatt) 提出的一种神经网络[107]. 如图 9.4 所示, 这是一个简单的感知器逻辑图, 该感知器具有两层, 第一层为输入层, 将输入的值传递给下一层; 第二层为计算单元, 并将结果输出. 因此, 该网络也被称为单层感知器, 它能容易实现与、或、非等逻辑运算, 也常用于线性分类问题.



9.3.1 单层感知器

- 假设输入模式为 n 维特征向量 $\boldsymbol{X}=(x_1, x_2, \cdots, x_n)^{\mathrm{T}}$, 则感知器的输入层应有 n 个神经元. 若输出类别有 k 个, 则输出层应包含 k 个神经元, 即$\boldsymbol{O}=(o_1, o_2, \cdots, o_k)^{\mathrm{T}}$. 输入层的第 j 个神经元与输出层的第 i 个神经元的连接权值为 ω_{ji} , 则第 i 个神经元的输出为

$$o_i = f\left(\sum_{j=1}^{n}\left(\omega_{ji}x_j + \theta_i\right)\right). \tag{9.3.1}$$

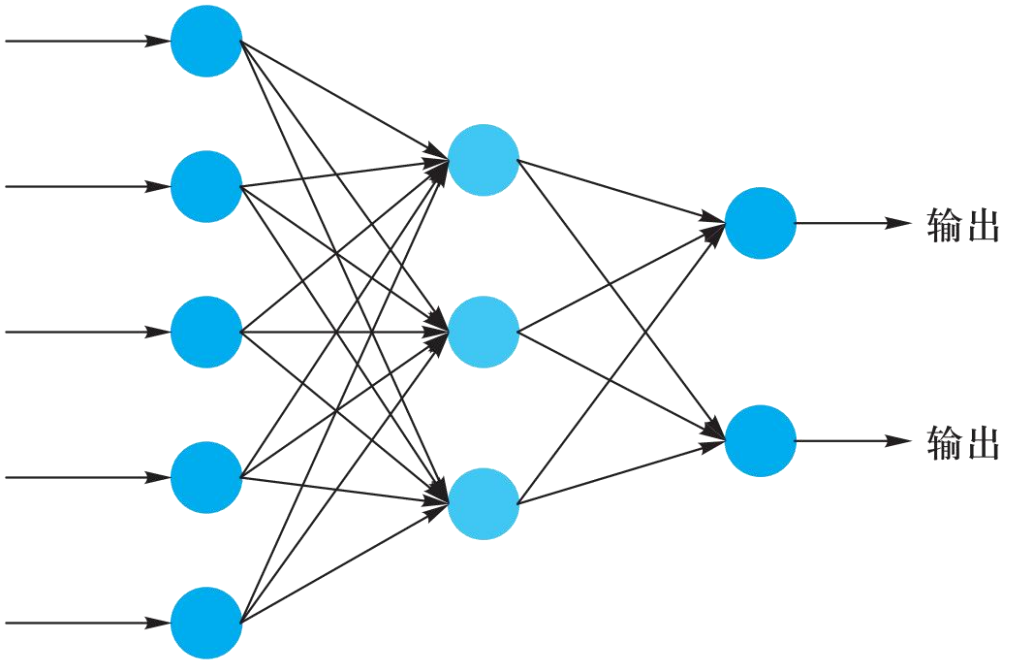
- 如果感知器模型是一种针对二分类问题中的神经网络模型, 其激活函数为符号函数 sgn(x), k 值为 2, 感知器模型通过输入层接受输入信息 $\boldsymbol{X}=(x_1, x_2, \cdots, x_n)^{\mathrm{T}}$, 其输出可以表示为

$$o_i = \mathrm{sgn}\left(\sum_{j=1}^{n}\left(\omega_{ji}x_j + \theta_i\right)\right). \tag{9.3.2}$$

- 一般地, 给定训练数据集和感知器后, 权重W以及阈值θ可通过学习得到. 当把阈值θ看作一个固定输入为1的“哑节点”(dummy node) 并将所对应的连接权也记为W(包含θ) 时, 权重和阈值的学习就可统一为权重的学习.

9.3.2 多层感知器

- 单层感知器只能解决线性可分的问题, 多层感知器可以突破这一局限, 实现输入和输出之间的非线性映射. 多层感知器的神经元层级之间采用全连接的方式, 上层的神经元的输出作为输入推送给下一层的所有神经元. 多层感知器中, 除了输入层和输出层, 中间可以有多个隐含层. 最简单的情况是只含一个输入层、一个隐含层和一个输出层, 即三层感知器, 也叫单隐层前馈神经网络, 如图 9.5 所示.



9.3.2 多层感知器

- 若第 l 层的第 i 个神经元的输出为 $a_i^{(l)}$, 其与上一层第 j 个神经元的连接权值为 $\omega_{ji}^{(l-1)}$, 偏置为 $\theta_i^{(l)}$; 与下一层的第 k 个神经元的连接权值为 $\omega_{ik}^{(l)}$, 偏置为 $\theta_k^{(l+1)}$. 则该神经元的输入为
- $z_i^{(l)}=\sum_j \omega_{ji}^{(l-1)} a_j^{(l-1)}+\theta_i^{(l)}$, 其中 $a_j^{(l-1)}$ 为上一层第 j 个神经元的输出,则第 l 层第 i 个神经元的输出为

$$a_i^{(l)}=f\left(z_i^{(l)}\right)=f\left(\sum_j \omega_{ji}^{(l-1)} a_j^{(l-1)}+\theta_i^{(l)}\right). \tag{9.3.3}$$

9.3.2 多层感知器

- 若第 l 层的第 i 个神经元的输出为 $a_i^{(l)}$, 其与上一层第 j 个神经元的连接权值为 $\omega_{ji}^{(l-1)}$, 偏置为 $\theta_i^{(l)}$; 与下一层的第 k 个神经元的连接权值为 $\omega_{ik}^{(l)}$, 偏置为 $\theta_k^{(l+1)}$. 则该神经元的输入为
- $z_i^{(l)}=\sum_j \omega_{ji}^{(l-1)} a_j^{(l-1)}+\theta_i^{(l)}$, 其中 $a_j^{(l-1)}$ 为上一层第 j 个神经元的输出,则第 l 层第 i 个神经元的输出为

$$a_i^{(l)}=f\left(z_i^{(l)}\right)=f\left(\sum_j \omega_{ji}^{(l-1)} a_j^{(l-1)}+\theta_i^{(l)}\right). \tag{9.3.3}$$

  - 其中, f 为激活函数.
- 从上面可以看出, 人工神经网络的学习过程, 就是根据训练数据来调整神经元之间的连接权值 (connection weight) 和每个功能神经元的偏置; 换言之, 神经网络学到的东西, 蕴含在连接权值与偏置中.

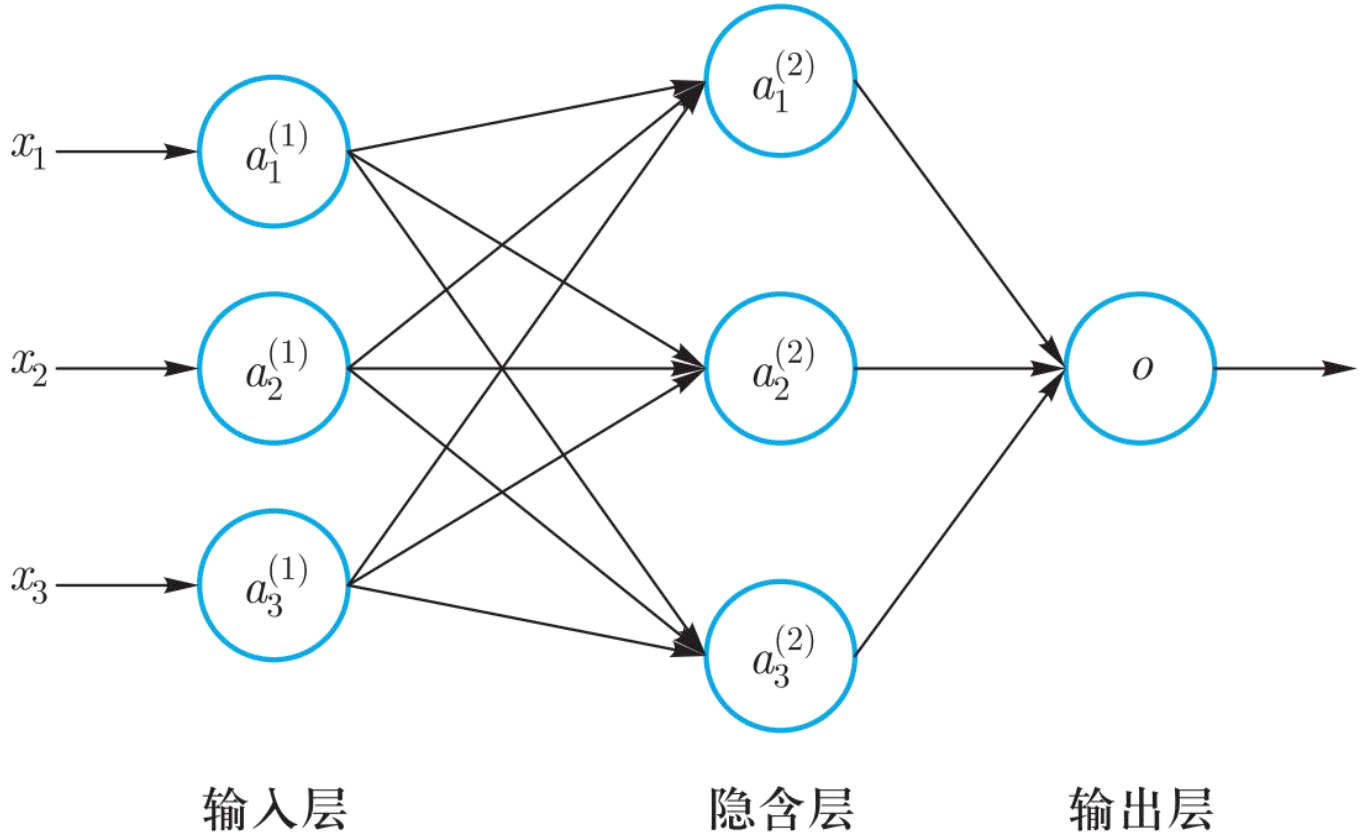
9.4　神经网络的正向与反向传播算法

9.4 神经网络的正向与反向传播算法

- 在前几节中, 我们已经学习了神经网络的基本知识. 由于神经网络的结构是多种多样的, 在本节中将以单隐层神经网络和双隐层神经网络为例, 介绍神经网络具体的学习过程是怎样的.

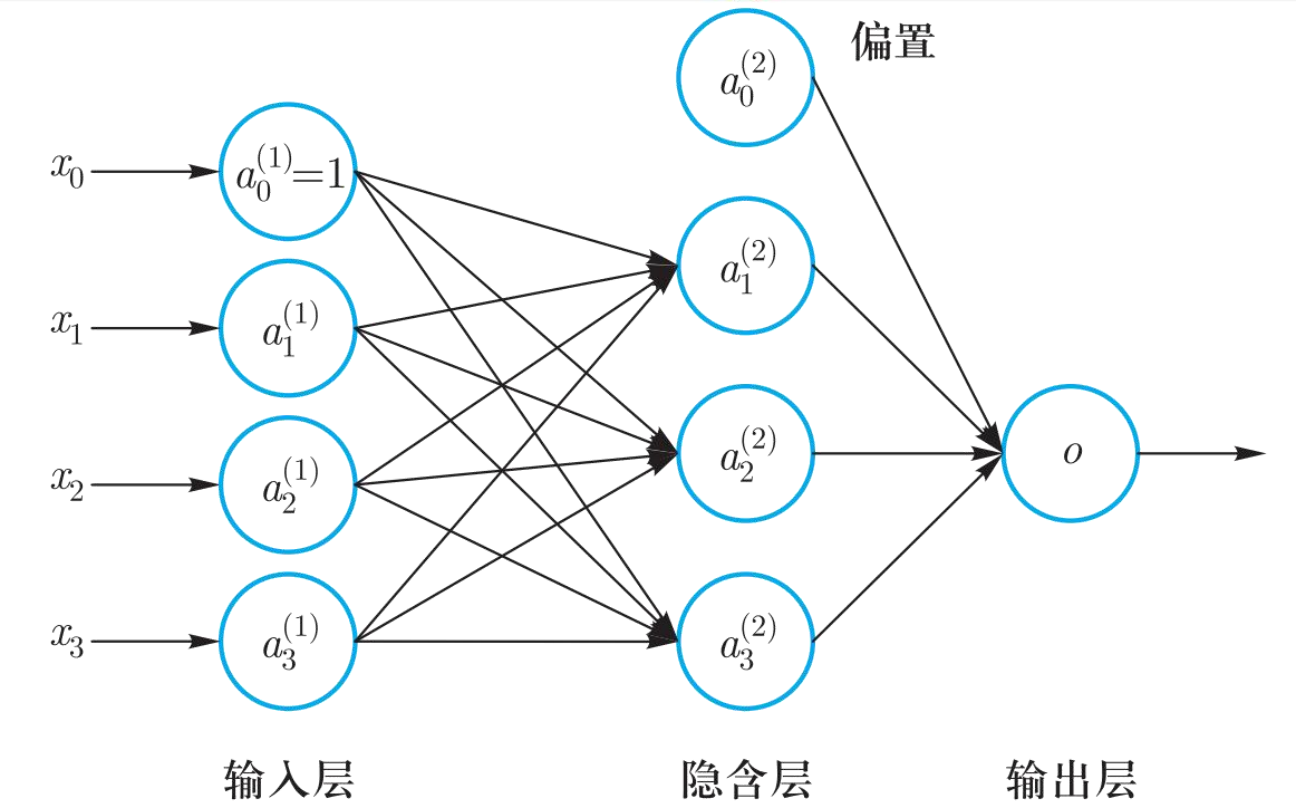
9.4.1 神经网络的正向传播

- 先以简单的单隐层神经网络为例 (图 9.6), 其中第一层是输入层单元, 输入原始数据. 第二层是隐含层单元, 对数据进行处理, 然后呈递到下一层. 最后是输出单元, 计算后, 输出计算结果.



9.4.1 神经网络的正向传播

- 一般情况下, 为了提高拟合能力, 需要增加偏置项, 如图 9.7 为一个 3 层的神经网络, 其中第一层为输入层, 最后一层为输出层, 中间一层为带偏置的隐含层.



9.4.1 神经网络的正向传播

- 下面引入一些标记法来帮助描述模型:
- $a_i^{(j)}$ 代表第 j 层的第 i 个单元,$\omega^{(j)}$代表从第 j 层映射到第 $j+1$ 层时的权重的矩阵, 例如 ω 代表从第一层映射到第二层的权重的矩阵. 其尺寸为:以第 j+1 层的单元数量为行数, 以第 j 层的激活单元数加一为列数的矩阵.例如: 图 9.7 所示的神经网络中如$\omega^{(1)}$的尺寸为 $3\times 4=12$.
- 对于图 9.7 所示的模型, 激活单元和输出分别表达为

$$a_1^{(2)}=f\left(\omega_{10}^{(1)}x_0+\omega_{11}^{(1)}x_1+\omega_{12}^{(1)}x_2+\omega_{13}^{(1)}x_3\right), \tag{9.4.1}$$

$$a_2^{(2)}=f\left(\omega_{20}^{(1)}x_0+\omega_{21}^{(1)}x_1+\omega_{22}^{(1)}x_2+\omega_{23}^{(1)}x_3\right), \tag{9.4.2}$$

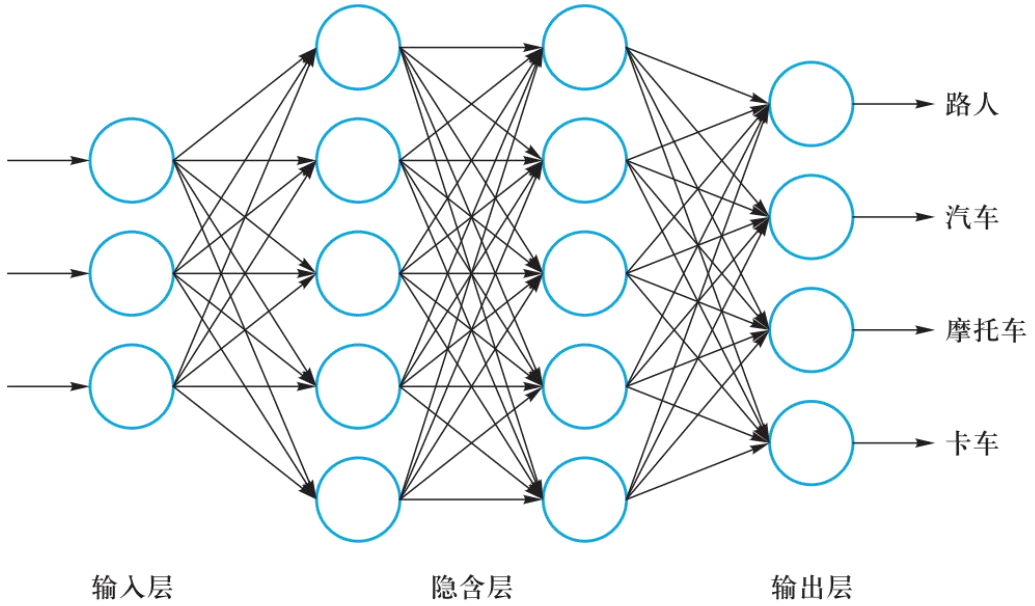
$$a_3^{(2)}=f\left(\omega_{30}^{(1)}x_0+\omega_{31}^{(1)}x_1+\omega_{32}^{(1)}x_2+\omega_{33}^{(1)}x_3\right), \tag{9.4.3}$$

$$O=f\left(\omega_{10}^{(2)}a_0^{(2)}+\omega_{11}^{(2)}a_1^{(2)}+\omega_{12}^{(2)}a_2^{(2)}+\omega_{13}^{(2)}a_3^{(2)}\right). \tag{9.4.4}$$

9.4.1 神经网络的正向传播

- 上述情况是一种两分类的神经网络模型, 但当有更多种分类时, 比如以下这种情况, 该怎么办?

- 假如要训练一个神经网络算法来识别路人、汽车、摩托车和卡车, 在输出层我们应该有 4 个值. 例如, 第一个值为 1 或 0 用于预测是不是行人, 第二个值用于判断是不是汽车, 以此类推. 输入向量有三个维度, 两个中间层,输出层4个神经元分别用来表示4类, 也就是每一个数据在输出层都会出现$(a, b, c, d)^{\mathrm{T}}$, 且 $(a, b, c, d)^{\mathrm{T}}$ 中仅有一个为 1, 其余为 0 时, 表示当前类. 图 9.8是该神经网络的结构示例



9.4.1 神经网络的正向传播

- 神经网络算法的输出结果为以下四种可能情形之一:

$$\begin{pmatrix}1\\0\\0\\0\end{pmatrix},\begin{pmatrix}0\\1\\0\\0\end{pmatrix},\begin{pmatrix}0\\0\\1\\0\end{pmatrix},\begin{pmatrix}0\\0\\0\\1\end{pmatrix}.$$

9.4.2 神经网络的损失函数

- 首先引入一些便于稍后讨论的符号:
- 假设神经网络的训练样本有 m 个, 第 i 个样本的输入为 $\boldsymbol{X}_i=(x_{i0},x_{i1},\cdots,x_{in})^{\mathrm{T}}$,若是分类问题, 则第 i 个样本的标签为 $\boldsymbol{Y}_i=(Y_{i1},\cdots,Y_{ik})^{\mathrm{T}}$, 其整体映射函数为 $G(\boldsymbol{X})$, 若权重 W 为变量, 则整体映射函数记为 $G_W(\boldsymbol{X})$. L 表示神经网络层数, S_l 表示每层的神经元个数 $(l=1,2,\cdots,L)$, 则 S_L 代表最后一层中神经元的个数. 若该神经网络是 k 分类问题, 那么显然 $S_L=k$.
- 神经网络学习的过程, 实际上就是最小化损失函数的过程. 而损失函数是用来估计模型的预测值 $G(\boldsymbol{X})$ 与真实值 $\boldsymbol{Y}$ 的不一致程度, 通常用误差函数进行衡量. 对于连续函数, 当 $G(\boldsymbol{X})$ 和 $\boldsymbol{Y}$ 维度为一时, 其损失函数常定义为

$$J(W)=\frac{1}{2m}\left[\sum_{i=1}^{m}\left(G_W(\boldsymbol{X}_i)-\boldsymbol{Y}_i\right)^2\right]+\lambda\|W\|_2^2. \tag{9.4.5}$$

  - 式 (9.4.5) 右侧的前半部分为经验风险函数, 后半部分为正则化项, λ 为正则化因子, 由于是 2 范数的平方项, 称为L_2 正则化项, 也可以是其他形式的正则化项.

9.4.2 神经网络的损失函数

- 对于二分类问题, 逻辑斯谛回归中常用的整体损失函数为

$$J(W)=-\frac{1}{m}\sum_{i=1}^{m}\left[\boldsymbol{Y}_i\log G_W(\boldsymbol{X}_i)+(1-\boldsymbol{Y}_i)\log\left(1-G_W(\boldsymbol{X}_i)\right)\right]+\lambda\|W\|_2^2. \quad (9.4.6)$$

- 在传统逻辑斯谛回归中, 只有一个输出变量, 但是在神经网络中, 可以有很多输出变量. 若 $G_W(\boldsymbol{X})$ 是一个维度为 k 的向量, $G_W(\boldsymbol{X})_j$ 表示 $G_W(\boldsymbol{X})$的第 j 个维度, 并且训练集中的因变量也是同样维度的一个向量, 因此损失函数会比逻辑斯谛回归更加复杂一些, 则损失函数为

$$\begin{aligned}J(W)=&-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}\left[\boldsymbol{Y}_{ij}\log G_W(\boldsymbol{X}_i)_j+(1-\boldsymbol{Y}_{ij})\log\left(1-\left(G_W(\boldsymbol{X}_i)\right)_j\right)\right]\\&+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_t}\sum_{j=1}^{S_t+1}\left(\omega_{ij}^{(l)}\right)^2.\end{aligned} \quad (9.4.7)$$

- 这个看起来复杂很多的损失函数, 其背后的思想还是一样的, 即希望通过损失函数来观察算法预测的结果与真实情况的误差有多大. 唯一不同的是,对于每一行特征, 都会给出 k 个预测, 基本上可以利用循环, 对每一行特征都预测 k 个不同结果, 然后再利用循环在 k 个预测中选择可能性最高的一个,将其与 $\boldsymbol{Y}$ 中的实际数据进行比较.

9.4.2 神经网络的损失函数

- 在确定了模型与损失函数后, 通过梯度下降算法就可以让神经网络进行学习了, 但是, 使用梯度下降算法有一个前提: 该算法需要提前知道当前点的梯度, 然而事实并非如此, 如果仔细观察过此模型的损失函数之后, 就会发现这个函数相当复杂, 其导函数难以计算, 而在更加复杂的神经网络模型中, 对损失函数求导往往是不可能的, 因此需要通过其他方法来进行实现.

9.4.3 反向传播算法

- 1974 年, Paul Werbos[108] 首次给出了训练神经网络的学习算法——反向传播算法 (back propagation, BP), 这个算法可以高效地计算每一次迭代过程中的梯度, 它是迄今最成功的神经网络学习算法. 在实际使用神经网络时, 大多是在使用 BP 算法进行训练. 值得指出的是, BP 算法不仅可用于多层前馈神经网络, 还可用于其他类型的神经网络, 例如训练递归神经网络[109]. 但通常说 “BP 网络” 时, 一般是指用 BP 算法训练的多层前馈神经网络.
- 下面举一个简单的例子来说明 BP 算法是如何运作的.
- 假设训练集只有一个样本 $(\boldsymbol{X},\boldsymbol{Y})$, 使用四层神经网络进行训练, 其中$k=4$, $S_L=4$(如图 9.8), 激活函数为 Sigmoid 函数, 损失函数为式 (9.4.5), 为简单起见，$\boldsymbol{Y}$ 是 k 类输出变量，且用损失函数

$$J(W)=\frac{1}{2}\sum_{j=1}^{k}\left(G_W(\boldsymbol{X})_j-Y_j\right)^2=\frac{1}{2}\sum_{j=1}^{k}\left(a_j^{(L)}-Y_j\right)^2,$$

9.4.3 反向传播算法

- 该网络反向传播算法为:

算法 1

$$a^{(1)} = \boldsymbol{X} \tag{9.4.8}$$

$$z^{(2)} = W^{(1)} a^{(1)} \tag{9.4.9}$$

$$a^{(2)} = f\left(z^{(2)}\right)\left(add\ a_0^{(2)}\right) \tag{9.4.10}$$

$$z^{(3)} = W^{(2)} a^{(2)} \tag{9.4.11}$$

$$a^{(3)} = f\left(z^{(3)}\right)\left(add\ a_0^{(3)}\right) \tag{9.4.12}$$

$$z^{(4)} = W^{(3)} a^{(3)} \tag{9.4.13}$$

$$a^{(4)} = f\left(z^{(4)}\right) \tag{9.4.14}$$

9.4.3 反向传播算法

- 一般从最后一层的误差开始计算, 定义 $\delta^{(i)}=\dfrac{\partial J(W)}{\partial z^{(i)}}$ 来表示误差, 则

$$\delta_j^{(4)}=\frac{\partial J(W)}{\partial z_j^{(4)}}=\left(a_j^{(4)}-Y_j\right)f'\left(z_j^{(4)}\right),$$

  - 再利用这个误差来计算前一层的误差

$$\delta^{(3)}=\frac{\partial J(W)}{\partial z^{(3)}}=\frac{\partial J(W)}{\partial z^{(4)}}=\frac{\partial z^{(4)}}{\partial z^{(3)}}=\left(W^{(3)}\right)^{\mathrm{T}}\delta^{(4)}*f'\left(z^{(3)}\right),$$

  - 其中 * 表示矩阵对应元素相乘, $f'\left(z^{(3)}\right)$ 是激活函数的导数, 而 $\left(W^{(3)}\right)^{\mathrm{T}}\delta^{(4)}$ 则是权重导致的误差和. 下一步是计算第二层的误差

$$\delta^{(2)}=\left(W^{(2)}\right)^{\mathrm{T}}\delta^{(3)}*f'\left(z^{(2)}\right).$$

  - 第一层是输入变量, 不存在误差.

9.4.3 反向传播算法

- 有了所有误差的表达式后, 便可以计算损失函数的偏导数了, 即不做任何正则化处理时有

$$\frac{\partial J(W)}{\partial \omega_{ji}^{(l)}} = \frac{\partial J(W)}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial \omega_{ji}^{(l)}} = a_i^{(l)} \delta_j^{(l+1)}.$$

- BP 算法的学习过程实际上由正向传播过程和反向传播过程组成. 在正向传播过程中, 输入信息通过输入层经隐含层, 逐层处理并传向输出层. 如果在输出层得不到期望的输出值, 则取输出与期望的误差的平方和作为目标函数, 转入反向传播, 逐层求出目标函数对各神经元权值的偏导数, 构成目标函数对权值向量的梯量, 作为修改权值的依据, 网络的学习在权值修改过程中完成. 误差达到所期望值时, 网络学习结束.

9.4.4 全局最小与局部极小

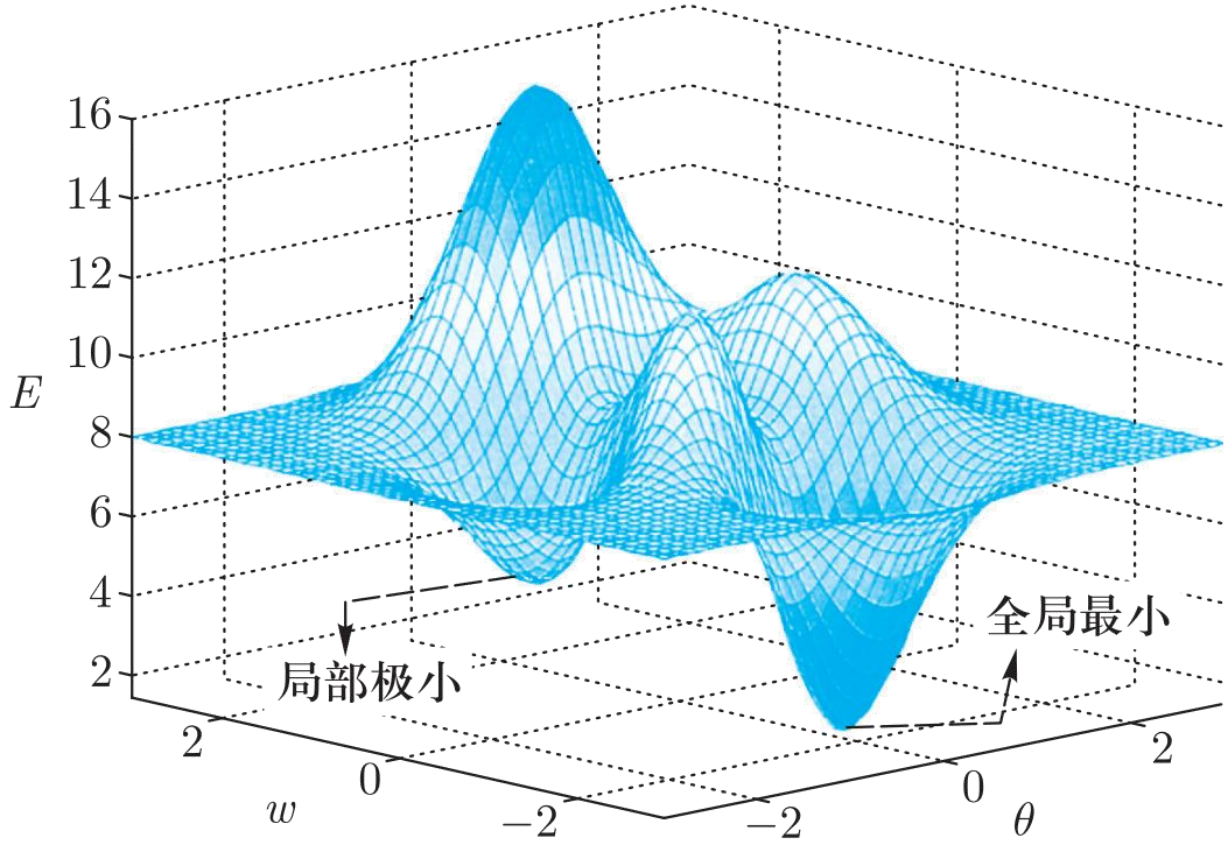
- 若用 J 表示神经网络在训练集上的误差, 则它显然是关于连接权值和偏置的函数. 此时, 神经网络的训练过程可看作一个参数寻优过程, 即在参数空间中, 寻找一组最优参数使得 J 最小.
- 我们常会谈到两种 “最优”: 局部极小 (local minimum) 和全局最小 (globalminimum). 对ω^* 和θ^* ,若存在$\varepsilon>0$ 使得

$$\forall(\omega,\theta)\in\left\{(\omega;\theta)\left\|(\omega;\theta)-(\omega^*;\theta^*)\right\|\leq\varepsilon\right\},$$

  - 都有$J(\omega;\theta)\geqslant J(\omega^*;\theta^*)$成立, 则$(\omega^*;\theta^*)$为局部极小解; 若对参数空间中的任意 $(\omega;\theta)$ 都有 $J(\omega;\theta)\geqslant J(\omega^*;\theta^*)$, 则 $(\omega^*;\theta^*)$ 为全局最小解. 直观地看, 局部极小解是参数空间中的某个点, 其邻域点的误差函数值均不小于该点的函数值; 全局最小解则是指参数空间中所有点的误差函数值均不小于该点的误差函数值. 两者对应的值分别称为误差函数的局部极小值和全局最小值.
- 显然, 参数空间内梯度为零的点, 只要其误差函数值小于邻点的误差函数值, 就是局部极小点; 可能存在多个局部极小值, 但却只会有一个全局最小值.

9.4.4 全局最小与局部极小

- 也就是说, “全局最小” 一定是 “局部极小”, 反之则不成立. 例如, 图 9.9中有两个局部极小, 但只有其中之一是全局最小. 显然, 在参数寻优过程中是希望找到全局最小.



9.4.4 全局最小与局部极小

- 基于梯度的搜索是使用最为广泛的参数寻优方法. 在此类方法中, 我们从某些初始解出发, 迭代寻找最优参数值. 每次迭代中, 先计算误差函数在当前点的梯度, 然后根据梯度确定搜索方向. 例如, 由于负梯度方向是函数值下降最快的方向, 因此梯度下降法就是沿着负梯度方向搜索最优解. 若误差函数在当前点的梯度为零, 则已达到局部极小, 更新量将为零, 这意味着参数的迭代更新将在此停止. 显然, 如果误差函数仅有一个局部极小, 那么此时找到的局部极小就是全局最小; 然而, 若误差函数具有多个局部极小, 则不能保证找到的解是全局最小. 对后一种情形, 称参数寻优陷入了局部极小, 这显然不是我们所希望的.

- 在现实任务中, 常采用以下策略来试图 “跳出” 局部极小, 从而进一步接近全局最小:

  - 1. 以多组不同参数值初始化多个神经网络, 按标准方法训练后, 取其中误差最小的解作为最终参数. 这相当于从多个不同的初始点开始搜索, 这样就可能陷入不同的局部极小, 从中选择有可能获得更接近全局最小的结果.
  - 2. 使用 “模拟退火”(simulated annealing, SA) 技术[110]. 模拟退火在每一步都以一定的概率接受比当前解更差的结果, 从而有助于 “跳出” 局部极小.在每步迭代过程中, 接受 “次优解” 的概率要随着时间的推移而逐渐降低, 从而保证算法稳定.

9.4.4 全局最小与局部极小

- 3. 使用随机梯度下降. 与标准梯度下降法精确计算梯度不同, 随机梯度下降法在计算梯度时加入了随机因素. 于是, 即便陷入局部极小点, 它计算出的梯度仍可能不为零, 这样就有机会跳出局部极小继续搜索.

此外, 遗传算法 (genetic algorithms, GA)[111] 也常用来训练神经网络以更好地逼近全局最小. 需注意的是, 上述用于跳出局部极小的技术大多是启发式智能算法, 理论上尚缺乏保障.

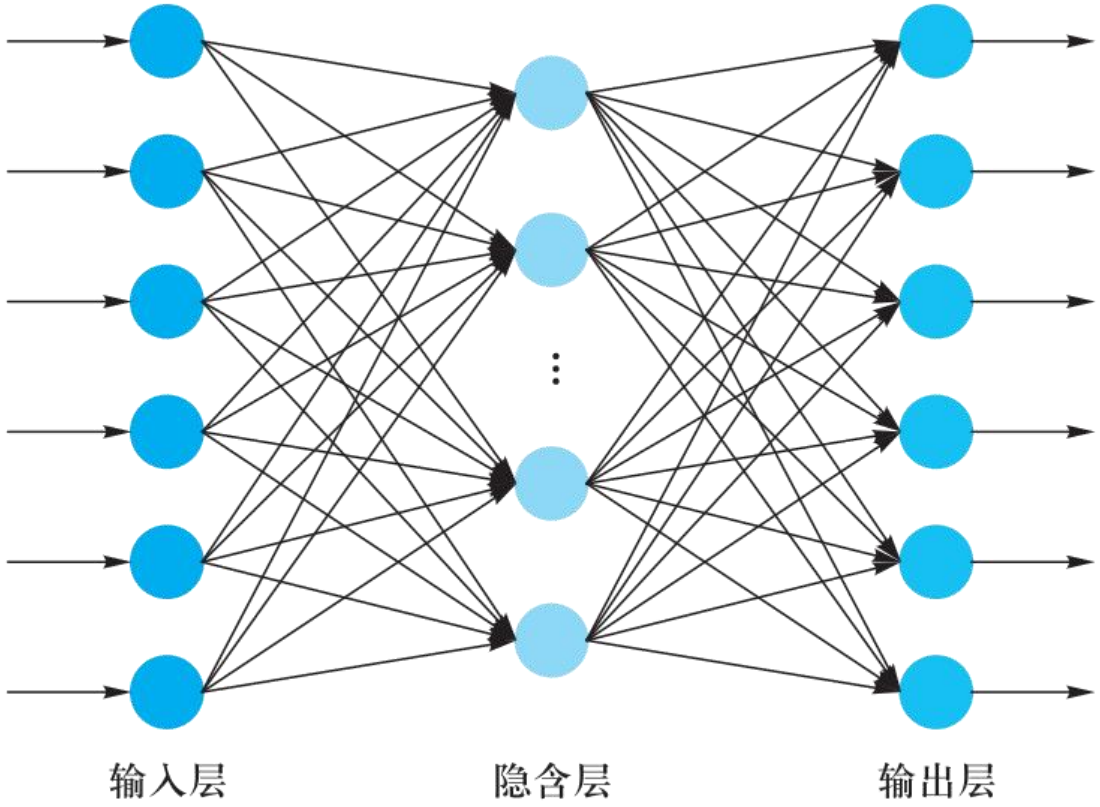
9.5　径向基网络

9.5 径向基网络

- 径向基函数神经网络 (简称为径向基 (radial basis function, RBF) 网络)是一种局部逼近型网络模型. 所谓局部逼近型网络, 是指网络输出仅与少数几个连接权重相关, 对于每个参与模型训练的样本, 通常仅有少数与其相关的权重需要进行更新. 这种局部性的参数更新方式有利于加速模型的训练.
- 机器学习中回归任务的本质是根据已知离散数据集求解与之相符的连续函数, 基本求解思路是对已知的离散数据进行拟合, 使得拟合函数与已知离散数据的误差在某种度量意义下达到最小. RBF 网络对于此类问题的求解思路则是通过对已知离散数据进行插值的方式确定网络模型参数.
- RBF 神经网络一共分为三层, 如图 9.10 所示. 第一层为输入层, 由信号源节点组成; 第二层为隐含层, 隐含层中神经元的激活函数, 即径向基函数是对中心点径向对称且衰减的非负线性函数, 该函数是局部响应函数. 局部响应函数一般要根据具体问题设置相应的隐含层神经元个数; 第三层为输出层,是对输入模式做出的响应, 输出层根据线性权重进行调整, 采用的是线性优化策略, 因而学习速度相对较快. 该网络将径向基函数作为隐单元的 “基” 构成隐含层空间, 将输入直接映射到隐空间.

9.5 径向基网络

- RBF 网络的基本思想是: 用 RBF 作为隐单元的 “基” 构成隐含层空间,这样就可以将输入直接映射到隐空间, 而不需要通过 “权” 连接. 当 RBF 的中心点确定以后, 这种映射关系也就确定了. 而隐含层空间到输出空间的映射是线性的, 即网络的输出是隐单元输出的线性加权和, 此处的权即为网络可调参数. 隐含层的作用是把向量从低维度映射到高维度, 这样低维度线性不可分的情况到高维度就可以变得线性可分了, 这即是核函数的思想. 这样网络由输入到输出的映射是非线性的, 而网络输出对可调参数而言却又是线性的. 网络的权就可由线性方程组直接解出, 从而大大加快学习速度并避免局部极小问题.



9.5 径向基网络

- 对于给定训练样本集 $D = \{(\boldsymbol{X}_1, Y_1),(\boldsymbol{X}_2, Y_2), \cdots ,(\boldsymbol{X}_m, Y_m)\}$, 其中$Y_i$ 为 $\boldsymbol{X}_i$ 所对应的连续真实值, 插值的目标是找到某个函数 f 使得

$$f\left(\boldsymbol{X}_i\right) = Y_i, \qquad i = 1, 2, \cdots, m. \tag{9.5.1}$$

  - 通常 f 为非线性函数, 故其函数图像为一个插值曲面, 该曲面经过数据集D 中所有样本点. 为求解函数 f, 可考虑选择 m 个与样本点对应的基函数$\Phi_1, \Phi_2, \cdots, \Phi_m$, 并将 f 近似表示为这组基函数的线性组合, 即有

$$f\left(\boldsymbol{X}\right) = \sum_{i=1}^{m} \omega_i \Phi_i. \tag{9.5.2}$$

- 只需确定上式中权重参数 $\omega_1, \omega_2, \cdots, \omega_m$ 取值, 就可确定插值函数 f 的具体形式, 进而将上述插值问题转换为如下非线性模型的参数问题:

$$\Phi_i = \Phi\left(\left\|\boldsymbol{X} - \boldsymbol{X}_i\right\|\right), \tag{9.5.3}$$

  - 其中 Φ_i 为非线性函数, 其函数自变量为输入数据 $\boldsymbol{X}$ 到中心点 $\boldsymbol{X}_i$ 的距离.

9.5 径向基网络

- 由于从中心点 $\boldsymbol{X}_i$ 到相同半径的球面上任意点的距离相等, 即距离具有径向相同性, 故通常将这种以距离作为自变量的基函数, 称为径向基函数. 将径向基函数的具体形式代入式中, 则可得到插值函数 f 如下形式:

$$f(\boldsymbol{X})=\sum_{i=1}^{m}\omega_i\Phi\left(\left\|\boldsymbol{X}-\boldsymbol{X}_i\right\|\right). \tag{9.5.4}$$

- 将数据集 D 中任意样本 $(\boldsymbol{X}_j, Y_j)$ 代入可得

$$Y_j=\sum_{i=1}^{m}\omega_i\Phi\left(\left\|\boldsymbol{X}-\boldsymbol{X}_i\right\|\right). \tag{9.5.5}$$

  - 将数据集D中数据均代入式 (9.5.5) 可得到由m个线性方程组成的线性方程组. 记径向基函数的取值 $\Phi\left(\left\|\boldsymbol{X}_j-\boldsymbol{X}_i\right\|\right)$ 为 Φ_{ij}, 则可将该线性方程组表示为如下矩阵形式:

$$\boldsymbol{\Phi\omega}=\boldsymbol{Y}, \tag{9.5.6}$$

  - 其中 $\boldsymbol{\Phi}=(\Phi_{ij})_{m\times m}$, $\omega=\left(\omega_1,\omega_2,\cdots,\omega_m\right)^{\mathrm{T}}$, $\boldsymbol{Y}=(Y_1, Y_2, \cdots, Y_m)^{\mathrm{T}}$.

9.5 径向基网络

- 径向基函数可以有多种选择, 如高斯径向基函数、反演 S 型径向基函数等.
  - (1) 高斯径向基函数:

$$G\left(\boldsymbol{X},\boldsymbol{X}_i\right)=\exp\left(-\frac{1}{2\delta_i^2}\left\|\boldsymbol{X}-\boldsymbol{X}_i\right\|^2\right); \tag{9.5.7}$$

  - (2) 反演 S 型径向基函数:

$$R\left(\boldsymbol{X},\boldsymbol{X}_i\right)=\frac{1}{1+\exp\left(\dfrac{\left\|\boldsymbol{X}-\boldsymbol{X}_i\right\|^2}{\delta_i^2}\right)}; \tag{9.5.8}$$

  - 其中 δ_i 为扩展常数, 其取值越大, 数据分布范围越宽.
- 针对分类问题, 高斯径向基函数常用

$$G\left(\boldsymbol{X},c_i\right)=\exp\left(-\beta_i\left\|\boldsymbol{X}-c_i\right\|^2\right),i=1,\cdots,k,$$

9.5 径向基网络

- 其中 c_i 称为中心. 通常采用两步过程来训练该网络: 第一步, 确定神经元中心 c_i, 常用的方式包括随机采样、聚类等; 第二步, 利用 BP 算法等来确定参数 ω_i 和 β_i. 第 j 个样本的 RBF 可用线性方程组表示为

$$\Phi_j W \xrightarrow{\text{激活}} Y_j,$$

- 其中 $\Phi_j = (\Phi_{ji})_{1\times k}$, $\Phi_{ji} = G(X_j, C_i)$, $W = (W_{ir})_{k\times r}$, $Y_j = (Y_{j1}, \cdots, Y_{jr})$.

9.6　其他常见的神经网络

9.5 径向基网络

- 神经网络模型中包含了大量的不同模型, 就算相同模型也可以通过改变网络的结构变得不同, 因此本书不能详尽地列举所有的模型, 所以选择了最常见的几种模型进行了简单的介绍.

1. ART 网络模型

- 竞争型学习 (competitive learning) 是神经网络中一种常用的无监督学习策略, 在使用该策略时, 网络的输出神经元相互竞争, 每一时刻仅有一个竞争获胜的神经元被激活, 其他神经元的状态被抑制. 这种机制亦称“胜者通吃”(winner-take-all) 原则.

- 自适应谐振理论 (adaptive resonance theory, ART) 网络[112] 是竞争型学习的重要代表. 该网络由比较层、识别层、识别阈值和重置模块构成. 其中,比较层负责接收输入样本, 并将其传递给识别层神经元. 识别层每个神经元对应一个模式类, 神经元数目可在训练过程中动态增长以增加新的模式类.

- 在接收到比较层的输入信号后, 识别神经元之间相互竞争以产生获胜神经元. 竞争的最简单方式是, 计算输入向量与每个识别层神经元所对应的模式类的代表向量之间的距离, 距离最小者胜.

9.5 径向基网络

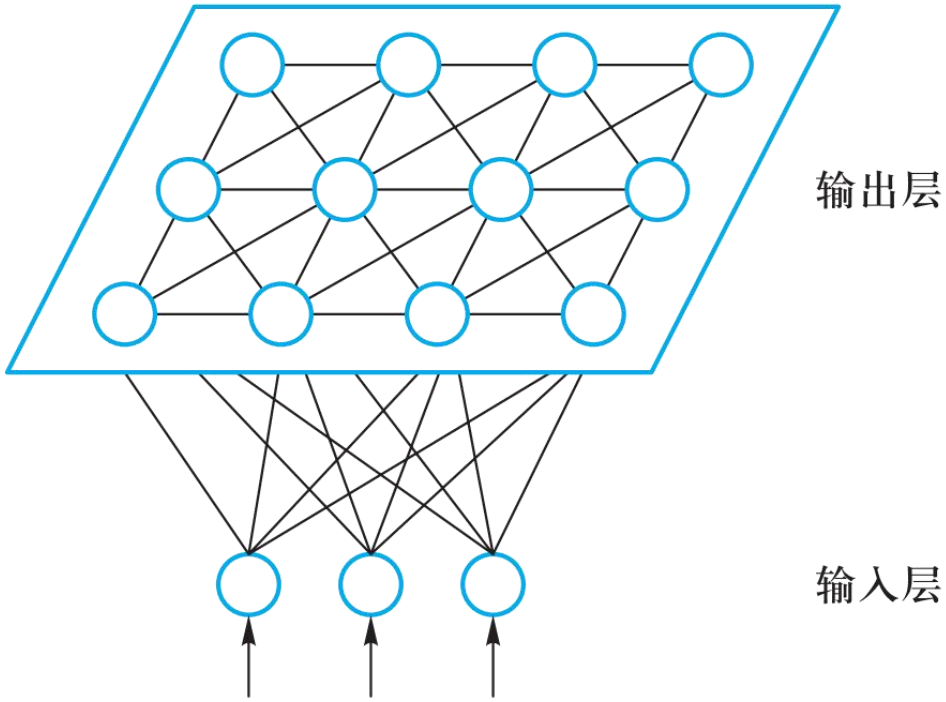
- 获胜神经元将向其他识别层神经元发送信号, 抑制其激活. 若输入向量与获胜神经元所对应的代表向量之间的相似度大于识别阈值, 则当前输入样本将被归为该代表向量所属类别,同时, 网络连接权将会更新, 使得以后在接收到相似输入样本时该模式类会计算出更大的相似度, 从而使该获胜神经元有更大可能获胜; 若相似度不大于识别阈值, 则重置模块将在识别层增设一个新的神经元, 其代表向量就设置为当前输入向量.

- ART 比较好地缓解了竞争型学习中的 “可塑性–稳定性窘境”(stability_x0002_plasticity dilemma), 可塑性是指神经网络要有学习新知识的能力, 而稳定性则是指神经网络在学习新知识时要保持对旧知识的记忆. 这就使得 ART 网络具有一个很重要的优点: 可进行增量学习 (incremental learning) 或在线学习 (online learning). 早期的 ART 网络只能处理布尔型输入数据, 此后 ART发展成了一个算法族, 包括能处理包括实值输入的 ART2 网络、结合模糊处理的 FuzzyART 网络, 以及可进行监督学习的 ARTMAP 网络等.

2. 自组织映射网络模型

- 自组织映射 (self-organizing map, SOM) 网络[113] 是一种竞争学习型的无监督神经网络, 它能将高维输入数据映射到低维空间 (通常为二维), 同时保持输入数据在高维空间的拓扑结构, 即将高维空间中相似的样本点映射到网络输出层中的邻近神经元.

- 如图 9.11 所示, SOM 网络中的输出层神经元以矩阵形式排列在二维空间中, 每个神经元都拥有一个权向量, 网络在接收输入向量后, 将会确定输出层获胜神经元, 它决定了该输入向量在低维空间中的位置. SOM 的训练目标就是为每个输出层神经元找到合适的权向量, 以达到保持拓扑结构的目的.



- SOM 的训练过程很简单: 在接收到一个训练样本后, 每个输出层神经元会计算该样本与自身携带的权向量之间的距离, 距离最近的神经元成为竞争获胜者, 称为最佳匹配单元 (best matching unit). 然后, 最佳匹配单元及其邻近神经元的权向量将被调整, 以使得这些权向量与当前输入样本的距离缩小.这个过程不断迭代, 直至收敛.

3. 级联相关网络

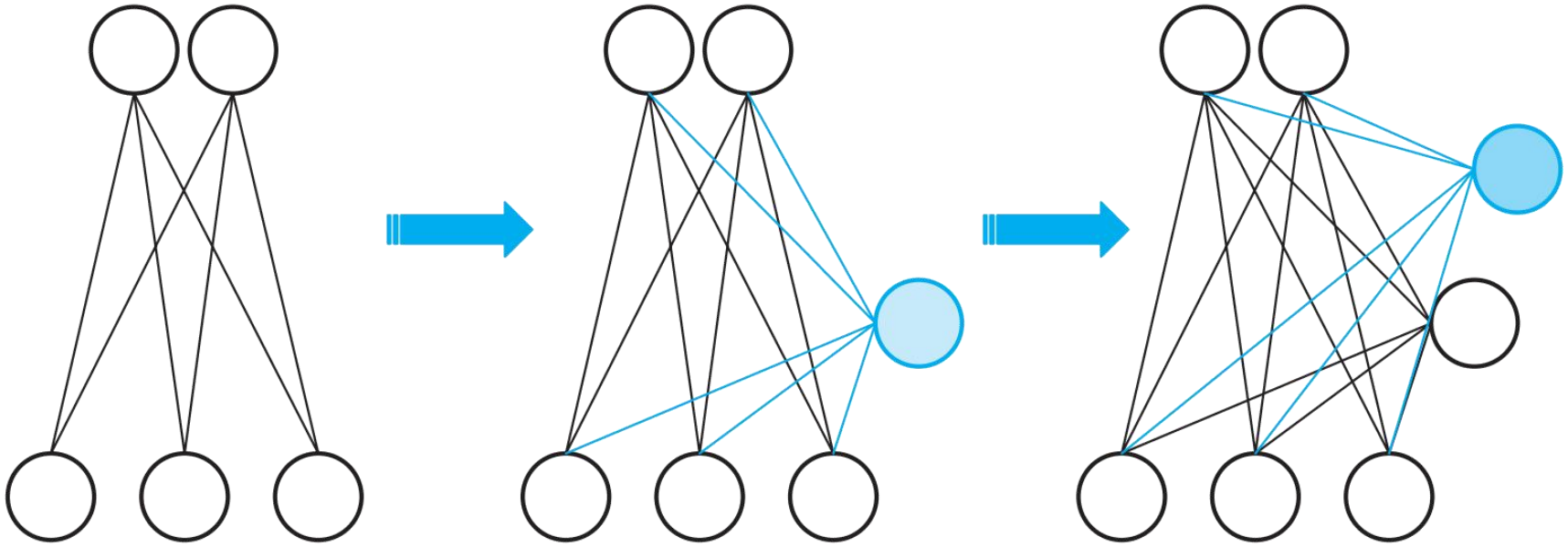
- 一般的神经网络模型通常假定网络结构是事先固定的, 训练的目的是利用训练样本来确定合适的连接权、阈值等参数. 与此不同, 结构自适应网络则将网络结构也当作学习的目标之一, 并希望能在训练过程中找到最符合数据特点的网络结构. 级联相关 (cascade-correlation) 网络[114] 是结构自适应网络的重要代表.
- 级联相关网络有两个主要成分: 级联和相关. 级联是指建立层次连接的层级结构. 在开始训练时, 网络只有输入层和输出层, 处于最小拓扑结构; 随着训练的进行, 如图 9.12 所示, 新的隐含层神经元逐渐加入, 从而创建起层级结构. 当新的隐含层神经元加入时, 其输入端连接权值是冻结固定的. 相关是指通过最大化新神经元的输出与网络误差之间的相关性 (correlation) 来训练相关的参数.

- SOM 的训练过程很简单: 在接收到一个训练样本后, 每个输出层神经元会计算该样本与自身携带的权向量之间的距离, 距离最近的神经元成为竞争获胜者, 称为最佳匹配单元 (best matching unit). 然后, 最佳匹配单元及其邻近神经元的权向量将被调整, 以使得这些权向量与当前输入样本的距离缩小.这个过程不断迭代, 直至收敛.

3. 级联相关网络

- 一般的神经网络模型通常假定网络结构是事先固定的, 训练的目的是利用训练样本来确定合适的连接权、阈值等参数. 与此不同, 结构自适应网络则将网络结构也当作学习的目标之一, 并希望能在训练过程中找到最符合数据特点的网络结构. 级联相关 (cascade-correlation) 网络[114] 是结构自适应网络的重要代表.
- 级联相关网络有两个主要成分: 级联和相关. 级联是指建立层次连接的层级结构. 在开始训练时, 网络只有输入层和输出层, 处于最小拓扑结构; 随着训练的进行, 如图 9.12 所示, 新的隐含层神经元逐渐加入, 从而创建起层级结构. 当新的隐含层神经元加入时, 其输入端连接权值是冻结固定的. 相关是指通过最大化新神经元的输出与网络误差之间的相关性 (correlation) 来训练相关的参数.

9.5 径向基网络



(a) 初始状态　(b) 增加一个隐含层节点　(c) 增加第二个隐含层节点

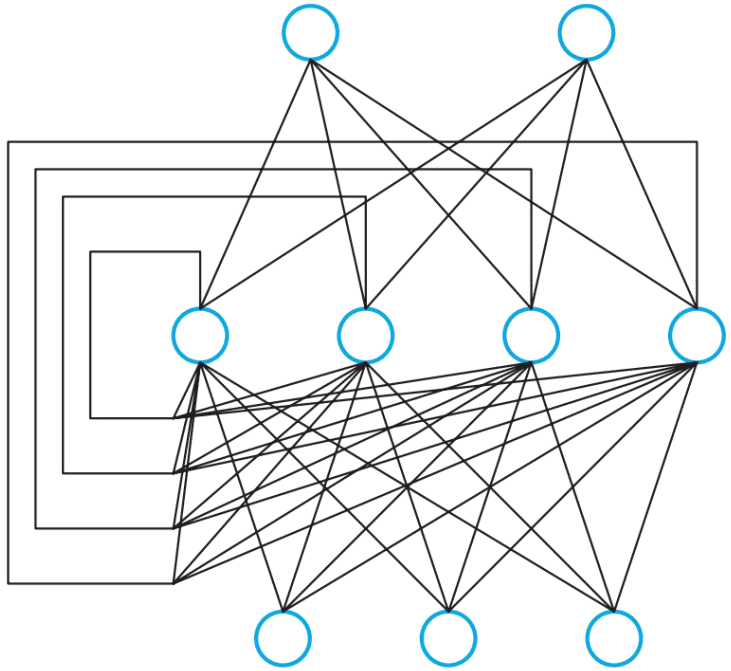
- 与一般的前馈神经网络相比, 级联相关网络无须设置网络层数、隐含层神经元数目, 且训练速度较快, 但其在数据较小时易陷入过拟合.

9.5 径向基网络

4. Elman 网络

- 与前馈神经网络不同, 递归神经网络 (recurrent neural networks) 允许网络中出现环形结构, 从而可让一些神经元的输出反馈回来作为输入信号. 这样的结构与信息反馈过程, 使得网络在 t 时刻的输出状态不仅与 t 时刻的输入有关, 还与 $t-1$ 时刻的网络状态有关, 从而能处理与时间有关的动态变化.

- Elman 网络[115] 是最常用的递归神经网络之一, 其结构如图 9.13 所示,它的结构与多层前馈网络很相似, 但隐含层神经元的输出被反馈回来, 与下一时刻输入层神经元提供的信号一起, 作为隐含层神经元在下一时刻的输入.隐含层神经元通常采用 Sigmoid 激活函数, 而网络的训练则常通过推广的 BP算法进行[109].



9.7　神经网络实践

实践代码