厦门大学
XIAMEN UNIVERSITY

神经网络

谭　忠

数 学 科 学 学 院
SCHOOL OF MATHEMATICAL SCIENCES

目录
CONTENTS

Part 1

源头问题与当今应用

在早期的1943年, 在商业计算机诞生之前, McCulloch 和 Pitts 开始探索人工神经元的小网络怎样模仿类似大脑的思考方式.在随后的几年, 计算机不断提高的可用性允许神经网络在简单的模式识别任务上被测试.但是进程缓慢，一个关键的原因在于"人工神经网络学习算法"受到了机器性能的限制.

一个主要的地标, 在神经网络的发展历史上, 是 Frank Rosenblatt 所创造的感知机(perceptron, 1958), 感知机显式地在大脑的神经架构之上进行建模.感知机学习算法 允许一个感知机去学习, 去联系输入和输出, 以人类大脑学习的方式.但也存在很大的局限性。

今天 “神经网络” 已是一个相当大的、多学科交叉的学科领域. 它能解决我们生活中的各种问题，其中深度神经网络能够对数据进行预测，卷积神经网络可应用与语音识别、图像处理、图像识别；生成多抗网络能够生成逼真的图像和视频。

“神经网络”已经是一个相当大、多学科交叉的学科领域. 各相关学科对神经网络的定义多种多样，本课程采用目前使用得最广泛得一种，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做出的交互反应**”. 我们在机器学习中谈及神经网络时指的是“神经网络学习”，或者说，是机器学习与神经网络这两个学科领域的交叉部分。

神经网络中最基本的成分是神经元模型, 即上述定义中的 “简单单元”. 在生物神经网络中, 每个神经元与其他神经元相连, 当它 “兴奋” 时, 就会向相连的神经元发送化学物质, 从而改变这些神经元内的电位; 如果某神经元的电位超过了一个 “阈值”, 那么它就会被激活, 即 “兴奋” 起来, 向其他神经元发送化学物质.

这个模型可以用下面这样图表示

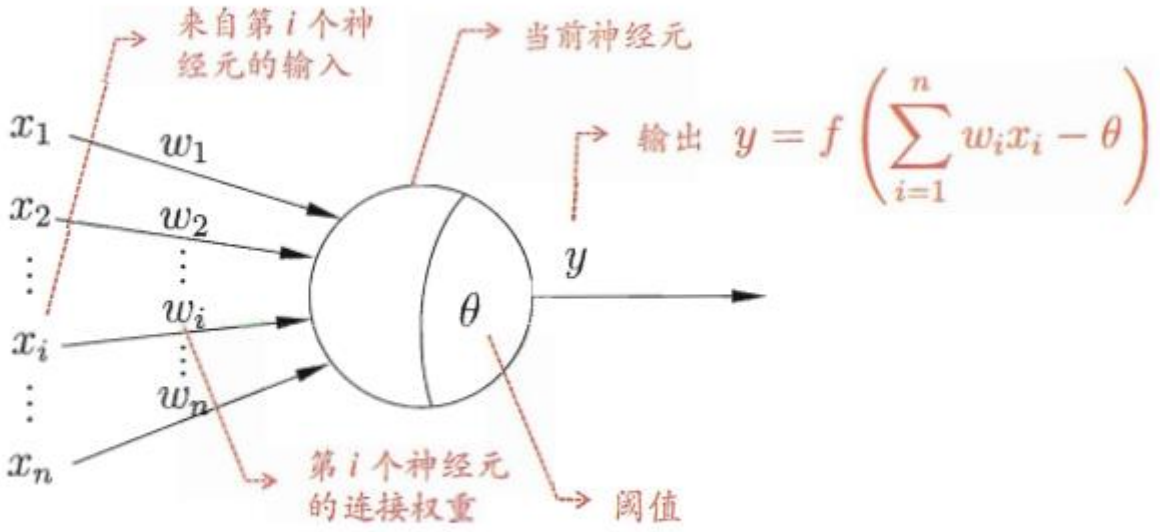


图 5.1　M-P 神经元模型

把许多个这样的神经元按一定的层次结构连接起来, 就得到了神经网络.

事实上, 从计算机科学的角度看, 我们可以先不考虑神经网络是否真的模拟了生物神经网络, 只需将一个神经网络视为包含了许多参数的数学模型, 这个模型是若干个函数, 例如 $y_j = f(\sum_i w_i x_i - \theta_j)$ 相互(嵌套)代入而得. 有效的神经网络学习算法大多以数学证明为支撑.

Part 2

神经网络思想与建模方法

一、感知机与多层网络

感知机由两层神经元组成, 输入层接收外界输入信号后传递给输出层, 输出层是 M-P 神经元, 亦称“阈值逻辑单元”.

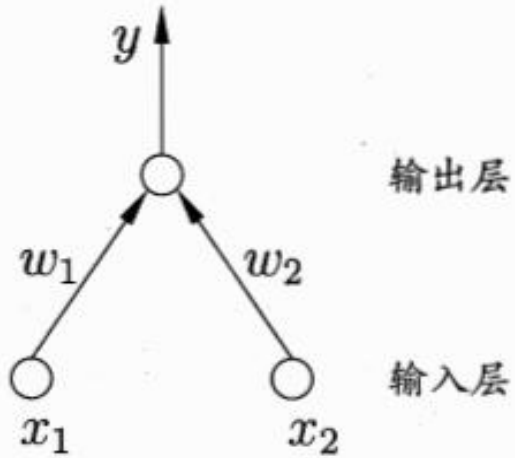
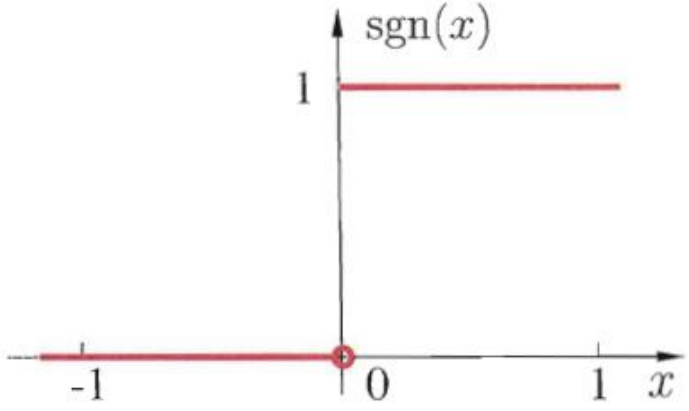


图 5.3 两个输入神经元的感知机网络结构示意图

感知机能容易地实现逻辑与、或、非运算. 注意到 $y = f(\sum_i w_i x_i - \theta)$, 假定 f 是如图的阶跃函数, 有

- "与" $(x_1 \wedge x_2)$: 令 $w_1 = w_2 = 1, \theta = 2$, 则 $y = f(1 \cdot x_1 + 1 \cdot x_2 - 2)$, 仅在 $x_1 = x_2 = 1$ 时, $y = 1$;
- "或" $(x_1 \vee x_2)$: 令 $w_1 = w_2 = 1, \theta = 0.5$, 则 $y = f(1 \cdot x_1 + 1 \cdot x_2 - 0.5)$, 当 $x_1 = 1$ 或 $x_2 = 1$ 时, $y = 1$;
- "非" $(\neg x_1)$: 令 $w_1 = -0.6, w_2 = 0, \theta = -0.5$, 则 $y = f(-0.6 \cdot x_1 + 0. x_2 + 0.5)$, 当 $x_1 = 1$ 时, $y = 0$; 当 $x_1 = 0$ 时, $y = 1$.



$$\operatorname{sgn}(x) = \begin{cases} 1, & x \geqslant 0; \\ 0, & x < 0. \end{cases}$$

(a) 阶跃函数

更一般地, 给定训练数据集, 权重 $w_i(i = 1,2, ..., n)$ 以及阈值 θ 可通过学习得到. 阈值 θ 可看作一个固定输入为 -1.0 的 “哑结点” 所对应的连接权重w_{n+1}, 这样, 权重和阈值的学习就可统一为权重的学习. 感知机学习规则非常简单, 对训练样例 $(\boldsymbol{x}, y)$, 若当前感知机的输出为 $\hat{y}$, 则感知机权 重将这样调整:

$$w_i \leftarrow w_i + \Delta w_i,$$
$$\Delta w_i = \eta(y - \hat{y})x_i,$$

其中 $\eta \in (0,1)$ 称为学习率.

感知机只能解决线性可分的问题，它甚至无法解决对于异或这种简单的非线性可分问题，如图：

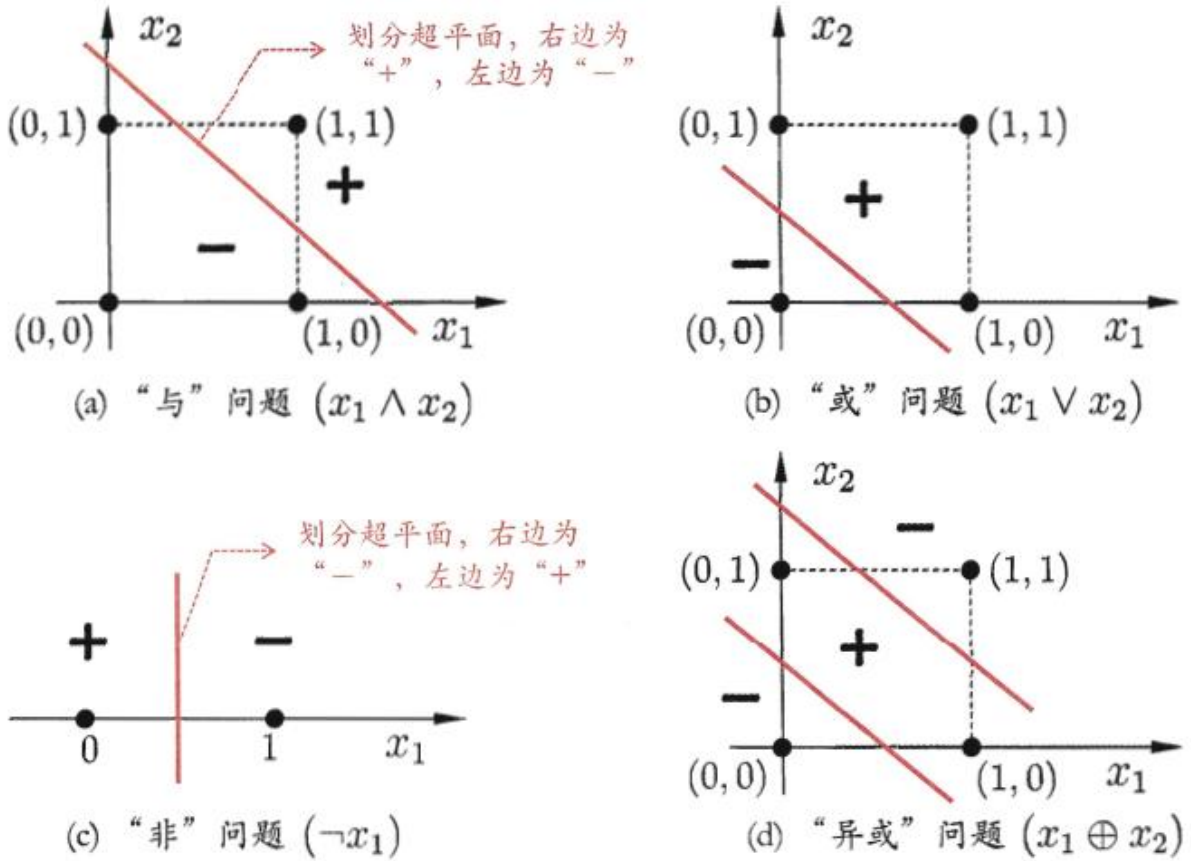
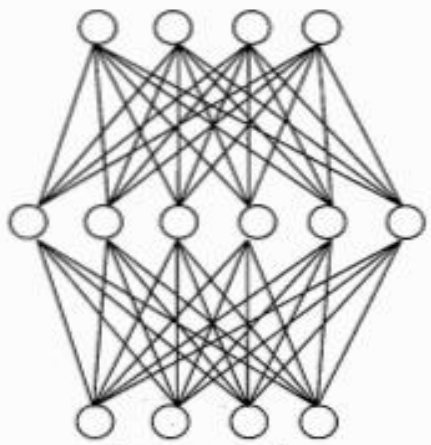
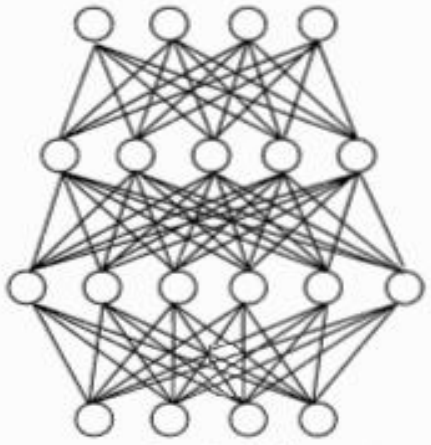


图 5.4 线性可分的“与” “或” “非”问题与非线性可分的“异或”问题

解决非线性可分问题需要用到如图所示的多层神经元，它由输入层、隐藏层和输出层构成



(a) 单隐层前馈网络



(b) 双隐层前馈网络

图 5.6　多层前馈神经网络结构示意图

二、误差逆传播算法

多层网络的学习能力比单层感知机强得多. 欲训练多层网络, 简单感知机学习规则显然不够了, 需要更强大的学习算法. 误差逆传播(简称BP)算法就是其中最杰出的代表, 如图:

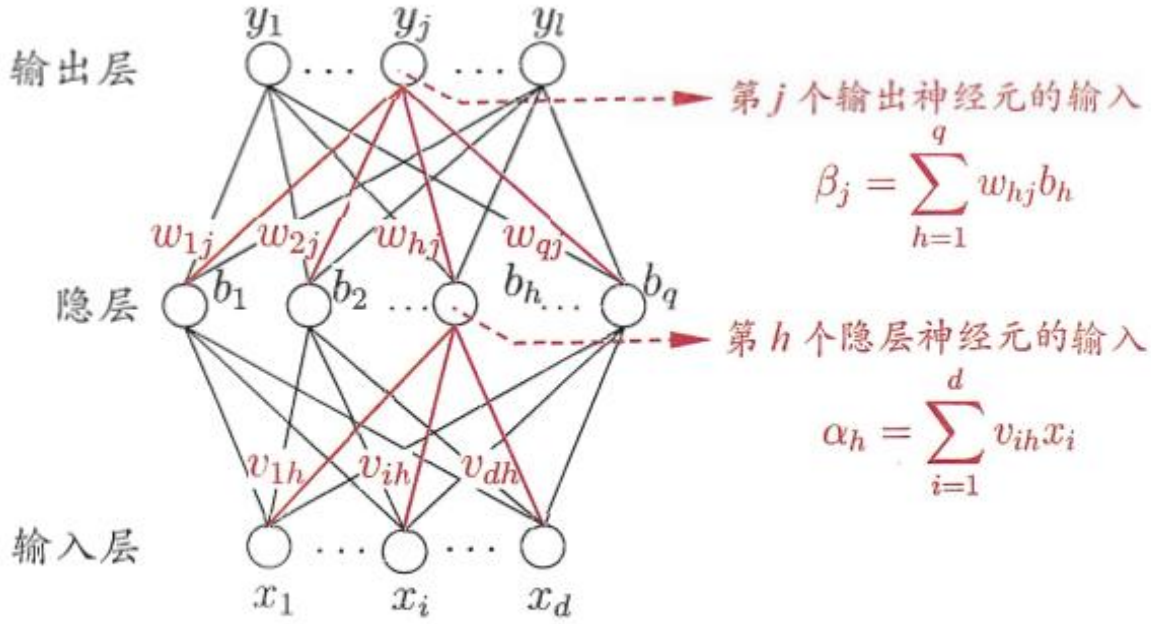


图 5.7 BP 网络及算法中的变量符号

对训练例 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$, 假定神经网终的输出为 $\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, \ldots, \hat{y}_l^k)$, 即

$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

则网络在 $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ 上的均方误差为

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

网络中有 $(d+l+1)q+l$ 个参数需确定：输入层到隐层的 $d \times q$ 个权值、隐层到输出层的 $q \times l$ 个权值、 q 个隐层神经元的阈值、l 个输出层神经元的阈值. BP 是一个**迭代学习算法**, 在迭代的每一轮中采用广义的感知机学习规则对参数进行更新估计, 任意参数 v 的更新估计式为

$$v \leftarrow v+\Delta v$$

下面我们以隐层到输出层的连接权 w_{hj} 为例来进行推导.

BP 算法基于梯度下降策略, 以目标的负梯度方向对参数进行调整.

对误差 E_k, 给定学习率 η, 有

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}}.$$

注意到 w_{hj} 先影响到第 j 个输出层神经元的输入值 β_j, 再影响到其输出值 $\hat{y}_j^k$, 然后影响到 E_k, 有

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}.$$

根据 β_j 的定义, 显然有

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h.$$

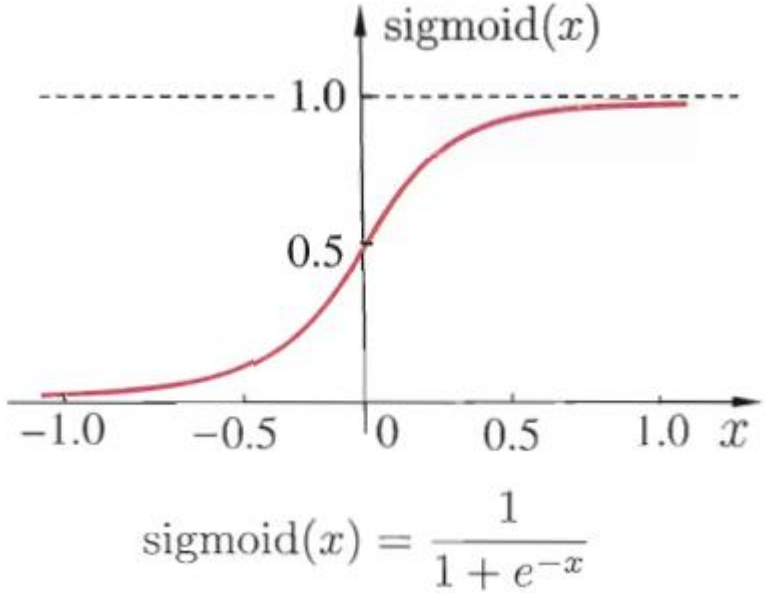


图 5.2 中的 Sigmoid 函数有一个很好的性质:

$$f'(x) = f(x)(1 - f(x))$$

于是根据 E_k 和 $\hat{y}_j^k$的表达式, 有

$$g_j = -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j}$$

$$= -\left(\hat{y}_j^k - y_j^k\right) f'\left(\beta_j - \theta_j\right)$$

$$= \hat{y}_j^k\left(1 - \hat{y}_j^k\right)\left(y_j^k - \hat{y}_j^k\right)$$

综合上面各式，就得到了BP 算法中关于 w_{hj} 的更新公式

$$\Delta w_{hj} = \eta g_j b_h.$$

类似可得

$$\begin{aligned} \Delta\theta_j &= -\eta g_j \\ \Delta v_{ih} &= \eta e_h x_i \\ \Delta\gamma_h &= -\eta e_h \end{aligned}$$

其中

$$e_h = -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h}$$

$$= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h)$$

$$= \sum_{j=1}^{l} w_{hj} g_j f'(\alpha_h - \gamma_h)$$

$$= b_h(1 - b_h) \sum_{j=1}^{l} w_{hj} g_j$$

需注意的是, BP 算法的目标是要最小化训练集 D 上的累积误差

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k$$

但我们上面介绍的 “标准 BP 算法” 每次仅针对一个训练样例更新连接权和阈值, 也就是说算法的更新规则是基于单个的 E_k 推导而得.

如果类似地推导出基于累积误差最小化的更新规则, 就得到了累积误差逆传播算法. 一般来说, 标准 BP 算法每次更新只针对单个样例, 参数更新得非常频繁, 而且对不同样例进行更新的效果可能出现“抵消”现象. 因此, 为了达到同样的累积误差极小点, 标准 BP 算法往往需进行更多次数的迭代. 累积BP算法直接针对累积误差最小化, 其参数更新的频率低得多. 但在很多任务中, 累积误差下降到一定程度之后, 进一步下降会非常缓慢, 这时标准 BP往往会更快获得较好的解, 尤其是在训练集 D 非常大时.

只需一个包含足够多神经元的隐层, 多层前馈网络就能以任意精度逼近任意复杂度的连续函数. 然而, 如何设置隐层神经元的个数仍是个未解决问题, 实际应用中通常靠“试错法”调整. BP 神经网络经常遭遇过拟合, 其训练误差持续降低, 但测试误差却可能上升. 有两种策略常用来缓解BP网络的过拟合.

第一种策略是“**早停**”，将数据分成训练集和验证集, 训练集用来计算梯度、更新连接权和阈值, 验证集用来估计误差, 若训练集误差降低但验证集误差升高, 则停止训练, 同时返回具有最小验证集误差的连接权和阈值.

第二种策略是 “**正则化**” ,其基本思想是在误差目标函数中增加一个用于描述网络复杂度的部分, 例如连接权与阈值的平方和. 仍令 E_k 表示第 k 个训练样例上的误差, w_i 表示连接权和阈值, 则误差目标函数改变为

$$E = \lambda \frac{1}{m} \sum_{k=1}^{m} E_k + (1-\lambda) \sum_{i} w_i^2,$$

其中 $\lambda \in (0,1)$ 用于对经验误差与网络复杂度这两项进行折中, 常通过交叉验证法来估计.

三、全局极小与局部极小

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. 每次迭代中根据梯度确定搜索方向. 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, 如果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, 我们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, 人们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, 从中进行选择有可能获得更接近全局最小的结果.

2、使用 “模拟退火” 技术 .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.

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.

注意，大多用于跳出局部极小点的算法都是启发式的，缺少理论上的保障.

四、其他神经网络

神经网络模型有非常多，这里对几个常见的进行简单的介绍。

1、**RBF（径向基函数）网络**是一种单隐层前馈神经网络, 它使用径向基函数作为隐层神经元激活函数, 而输出层则是对隐层神经元输出的线性组合. 假定输入为 d 维向量 $\boldsymbol{x}$, 输出为实值, 则RBF网络可表示为

$$\varphi(\boldsymbol{x}) = \sum_{i=1}^{q} w_i \rho(\boldsymbol{x}, \boldsymbol{c}_i),$$

其中 q 为隐层神经元个数, $\boldsymbol{c}_i$ 和 w_i 分别是第 i 个隐层神经元所对应的中心和权重, $\rho(\boldsymbol{x}, \boldsymbol{c}_i)$ 是径向基函数, 这是某种沿径向对称的标量函数, 通常定义为样本 $\boldsymbol{x}$ 到数据中心 c_i 之间欧氏距离的单调函数. 常用的高斯径向基函数形如

$$\rho(\boldsymbol{x}, \boldsymbol{c}_i) = e^{-\beta_i \|\boldsymbol{x} - \boldsymbol{c}_i\|^2}.$$

通常采用两步过程来训练 RBF 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 w_i 和 β_i.

2、ART网络

竞争型学习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” 原则.

ART(Adaptive Resonance Theory, 自适应谐振理论) 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.

竞争方式：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，距离最小者胜利。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时，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，使得以后在接收相似输入样本时该模式类会计算出更大的相似度，从而使得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网络可以较好地缓解竞争型学习中的“可塑性-稳定性窘境”，可塑性是指神经网络要有学习新知识的能力，稳定性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**或**在线学习**。

3、SOM网络

SOM(Self-Organizing Map, 自组织映射)网络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 5.11 所示, SOM 网络中的输出层神经元以矩阵方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.

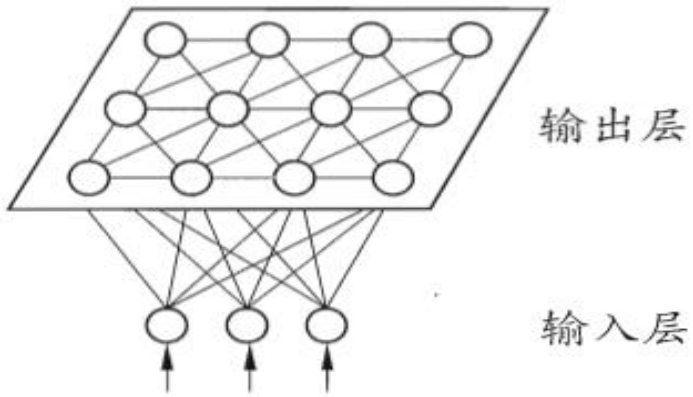


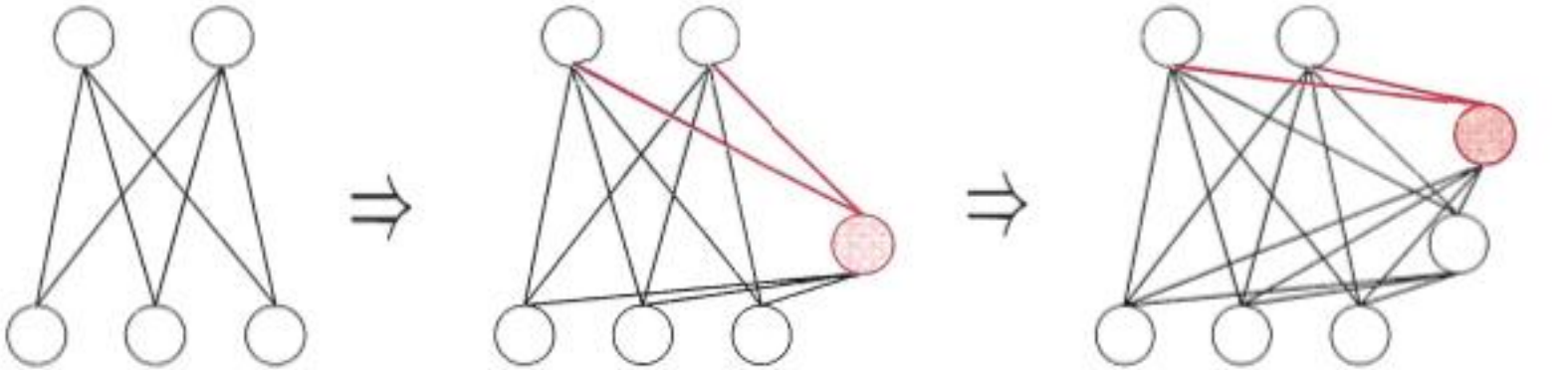
图 5.11 SOM 网络结构

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. 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.

4、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终结构. 级联相关(Cascade-Correlation) 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 5.12 所示, 新的隐层神经元逐渐加入, 从而创建起层级结构. 当新的隐层神经元加入时, 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.



(a) 初始状态　(b) 增加一个隐层结点　(c) 增加第二个隐层结点

图 5.12 级联相关网络的训练过程. 新的隐结点加入时, 红色连接权通过最大化新结点的输出与网络误差之间的相关性来进行训练.

5、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 网络是最常用的递归神经网络之一, 其结构如图 5.13 所示, 它的结构与多层前馈网络很相似, 但隐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, 作为隐层神经元在下一时刻的输入. 隐层神经元通常采用 Sigmoid 激活函数, 而网络的训练则常通过推广的BP算法进行

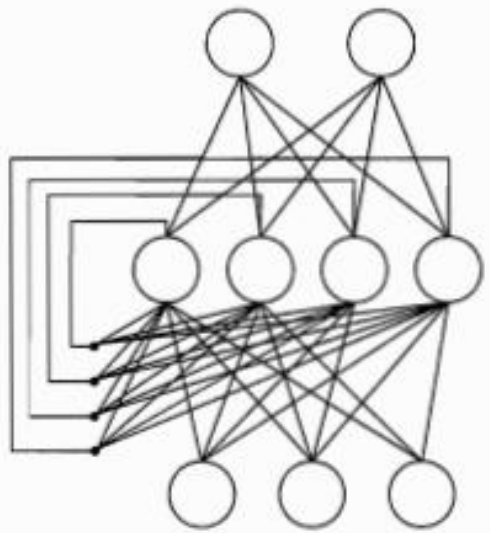
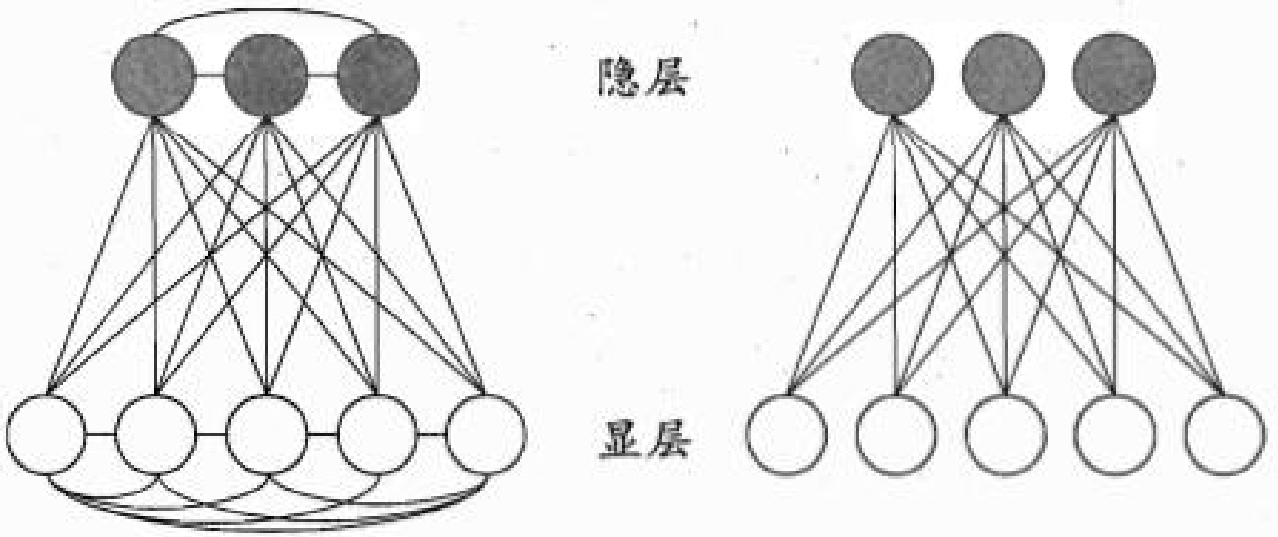


图 5.13 Elman 网络结构

6、Boltzmann机

神经网络中有一类模型是为网络状态定义一个“能量”，能量最小化时网络达到理想状态, 而网络的训练就是在最小化这个能量函数. Boltzmann 机就是一种“基于能量的模型” (energy-based model), 常见结构如图 5.14(a) 所示, 其神经元分为两层: 显层与隐层. 显层用于表示数据的输入与输出, 隐层则被理解为数据的内在表达.



(a) Boltzmann机　　(b) 受限Boltzmann机

图 5.14　Boltzmann 机与受限 Boltzmann 机

Boltzmann 机中的神经元都是布尔型的, 即只能取 0、1 两种状态, 状态 1 表示激活, 状态 0 表示抑制. 令向量 $s \in \{0,1\}^n$ 表示 n 个神经元的状态, w_{ij} 表示神经元 i 与 j 之间 的连接权, θ_i 表示神经元 i 的阈值, 则状态向量 s 所对应的 Boltzmann 机能量定义为

$$E(\boldsymbol{s}) = -\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} w_{ij}s_i s_j - \sum_{i=1}^{n}\theta_i s_i.$$

若网络中的神经元以任意不依赖于输入值的顺序进行更新, 则网络最终将达到 Boltzmann 分布, 此时状态向量 s 出现的概率将仅由其能量与所有可能状态向量的能量确定:

$$P(s) = \frac{e^{-E(s)}}{\sum_t e^{-E(t)}}.$$

Boltzmann 机的训练过程就是将每个训练样本视为一个状态向量, 使其出现的概率尽可能大. 标准的 Boltzmann 机是一个全连接图, 训练网络的复杂度很高, 这使其难以用于解决现实任务. 现实中常采用受限 Boltzmann 机(Restricted Boltzmann Machine, 简称 RBM). 如图 5.14(b) 所示, 受限 Boltzmann 机仅保留显层与隐层之间的连接, 从而将 Boltzmann 机结构由完全图简化为二部图.

受限 Boltzmann 机常用 “对比散度” (Contrastive Divergence, 简称 CD)算法来进行训练. 假定网络中有 d 个显层神经元和 q 个隐层神经元, 令 $\boldsymbol{v}$ 和 $\boldsymbol{h}$ 分别表示显层与隐层的状态向量, 则由于同一层内不存在连接, 有

$$P(\boldsymbol{v}|\boldsymbol{h}) = \prod_{i=1}^{d} P(v_i|\boldsymbol{h})$$

$$P(\boldsymbol{h}|\boldsymbol{v}) = \prod_{j=1}^{q} P(h_j|\boldsymbol{v})$$

CD 算法对每个训练样本 $\boldsymbol{v}$, 先根据$P(\boldsymbol{h}|\boldsymbol{v})$计算出隐层神经元状态的概率分布, 然后根据这个概率分布采样得到 $\boldsymbol{h}$; 此后, 从 $\boldsymbol{h}$ 产生 $\boldsymbol{v}'$, 再 从 $\boldsymbol{v}'$ 产生 $\boldsymbol{h}'$; 连接权的更新公式为

$$\Delta w = \eta(\boldsymbol{v}\boldsymbol{h}^{\mathsf{T}} - \boldsymbol{v}'\boldsymbol{h}^{\mathsf{T}}).$$

Part 3

案例分析

我们使用威斯康星州乳腺癌数据集，该数据集可以从sklearn中导入。

In[4]:

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): \n{}".format(cancer.keys()))
```

Out[4]:

```
cancer.keys():
dict_keys(['feature_names', 'data', 'DESCR', 'target', 'target_names'])
```

In[100]:

```
from sklearn.neural_network import MLPClassifier
X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, random_state=0)
mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)
print("Accuracy on training set:
{:.2f}".format(mlp.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(mlp.score(X_test,
y_test)))
```

Out[100]:

Accuracy on training set: 0.94

Accuracy on test set: 0.92

此时已经取得足够好的效果，但与其他模型相比稍显逊色。于是再次对数据进行预处理并增加迭代次数。神经网络也要求所有输入特征的变化范围相似，最理想的情况是均值为 0、方差为 1。

In[101]:

```
mean_on_train = X_train.mean(axis=0) # 计算训练集中每个特征的平均值
std_on_train = X_train.std(axis=0) # 计算训练集中每个特征的标准差
X_train_scaled = (X_train - mean_on_train) / std_on_train
# 减去平均值，然后乘以标准差的倒数
# 对测试集做相同的变换（使用训练集的平均值和标准差）
X_test_scaled = (X_test - mean_on_train) / std_on_train
mlp = MLPClassifier(alpha=1,max_iter=1000,random_state=0)
mlp.fit(X_train_scaled, y_train)
print("Accuracy on training set: {:.3f}".format(
 mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled,
y_test)))
```

Out[101]:

Accuracy on training set:0.988

Accuracy on test set: 0.972

性能有了明显的提升。

感谢大家！