神经网络

1. 神经元模型

“神经网络是由具有适应性的简单单元组成的广泛并行互联的网络，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。”神经网络最基本的成分是神经元（neuron）模型

1.1 M-P神经元模型

在这个模型中，神经元接收到来自n个其他神经元传递过来的输入信号。这些输入信号通过带全权重的连接（connection）进行传递，神经元接收到的总输入值将与神经元的阈值进行比较，然后通过**“激活函数”**（activation function）处理以产生神经元的输出。

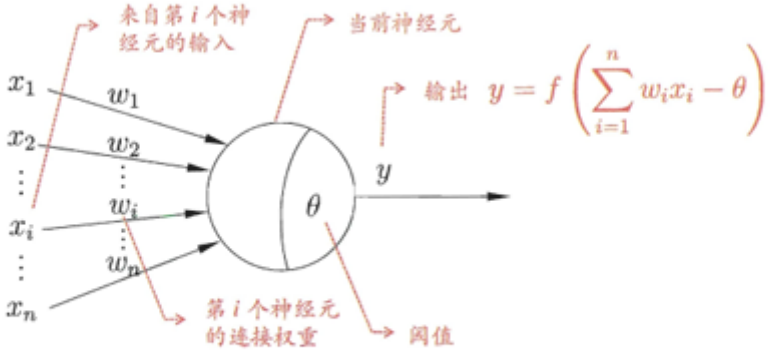
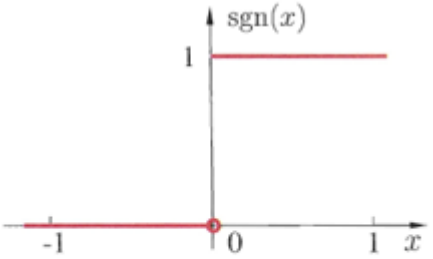


图 5.1 M-P 神经元模型

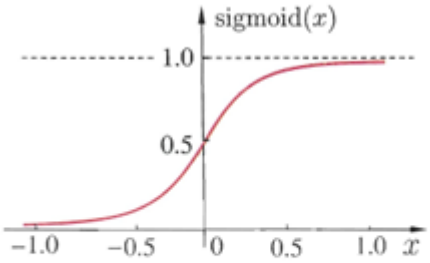
理想中的激活函数是如图所示的阶跃函数



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

(a) 阶跃函数

它将输入值映射为输出值“0”或“1”,显然“1”对应于神经元兴奋，“0”对应于神经元抑制，然而，阶跃函数具有不连续，不光滑等不好的性质。因此实际采用**Sigmoidj函数**作为激活函数。典型的激活函数如图所示，他把可能在较大范围内变化的输入值挤压到（0，1）输出值范围内，因此有时也称为**“挤压函数”**（squashing function）。



$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

(b) Sigmoid 函数

2. 感知机与多层网络

感知机（Perceceotron）由两层神经元组成，如图，输入层接收外层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"（threshold logic unit）。

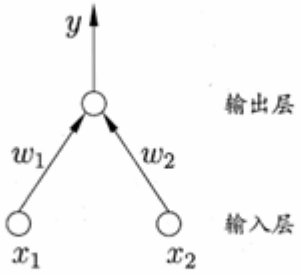


图 5.3 两个输入神经元的感知机网络结构示意图

感知机能容易的实现逻辑与，或，非运算，注意到$y = f(\sum_i w_i x_i - \theta)$,假定f是sigmoid函数。

- "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 $y = f(-0.6 \cdot x_1 + 0 \cdot x_2 + 0.5)$, 当 $x_1 = 1$ 时, $y = 0$; 当 $x_1 = 0$ 时, $y = 1$.

一般地，给定训练数据集，权重W1（i=1,2,....,n） x_i 是 $\boldsymbol{x}$ 对应于第 i 个输入神经元的分量. 以及阈值0可通过学习得到。法制0可看作一个固定输入为-1.0的"哑节点"（dummy node）所对应的连接权重wn+1这样，权重和阈值的学习就可以统一为权重的学习，感知机学习规则：对训练样例（x,y），若当前感知机的输出为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(learning rate). 从式(5.1) 可看出, 若感知机对训练样例 $(\boldsymbol{x}, y)$ 预测正确, 即 $\hat{y} = y$, 则感知机不发生变化, 否则将根据错误的程度进行权重调整.

η 通常设置为一个小正数, 例如 0.1.

需注意的是, 感知机只有输出层神经元进行激活函数处理, 即只拥有一层功能神经元(functional neuron), 其学习能力非常有限. 事实上, 上述与、或、非问题都是线性可分(linearly separable)的问题. 可以证明 [Minsky and Papert, 1969], 若两类模式是线性可分的, 即存在一个线性超平面能将它们分开, 如图 5.4(a)-(c) 所示, 则感知机的学习过程一定会收敛(converge) 而求得适当的权向量 $\boldsymbol{w} = (w_1; w_2; \ldots; w_{n+1})$; 否则感知机学习过程将会发生振荡(fluctuation), $\boldsymbol{w}$ 难以稳定下来, 不能求得合适解, 例如感知机甚至不能解决如图 5.4(d) 所示的异或这样简单的非线性可分问题.

"非线性可分" 意味着用线性超平面无法划分.

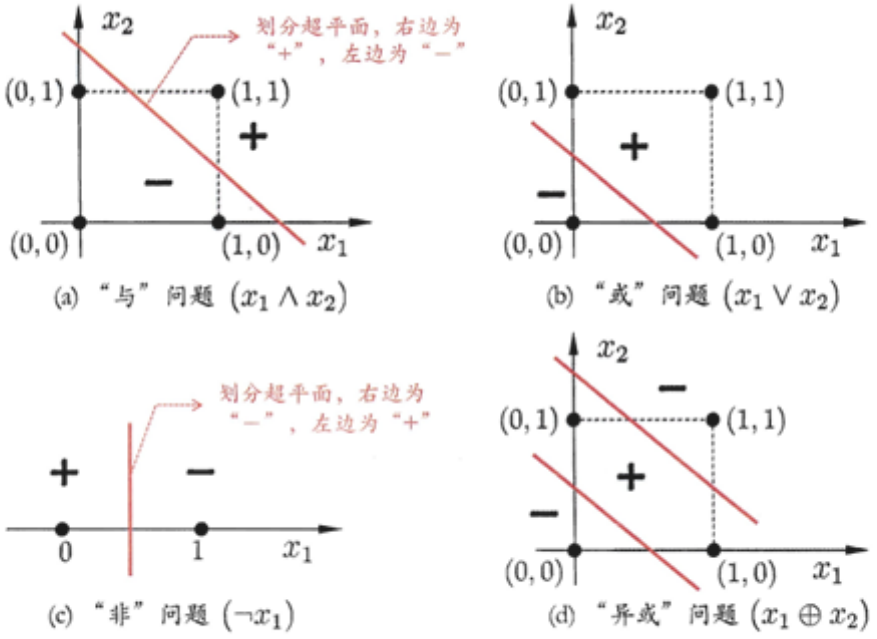


图 5.4 线性可分的 "与" "或" "非" 问题与非线性可分的 "异或" 问题

要解决非线性可分问题, 需考虑使用多层功能神经元. 例如图 5.5 中这个简单的两层感知机就能解决异或问题. 在图 5.5(a)中, 输出层与输入层之间的一层神经元, 被称为隐层或隐含层(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.

更一般的, 常见的神经网络是形如图 5.6 所示的层级结构, 每层神经元与下一层神经元全互连, 神经元之间不存在同层连接, 也不存在跨层连接. 这样的神经网络结构通常称为“多层前馈神经网络”(multi-layer feedforward neural

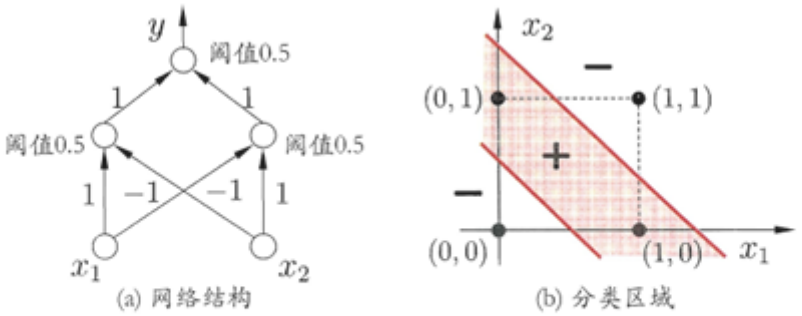


图 5.5 能解决异或问题的两层感知机

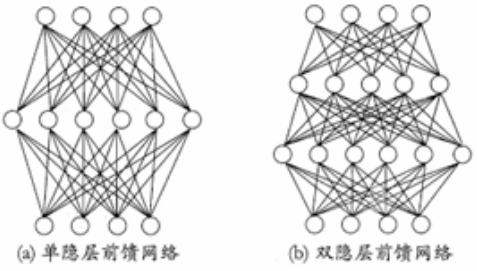


图 5.6 多层前馈神经网络结构示意图

“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; 参见 5.5.5 节.

networks), 其中输入层神经元接收外界输入, 隐层与输出层神经元对信号进行加工, 最终结果由输出层神经元输出; 换言之, 输入层神经元仅是接受输入, 不进行函数处理, 隐层与输出层包含功能神经元. 因此, 图 5.6(a) 通常被称为“两层网络”. 为避免歧义, 本书称其为“单隐层网络”. 只需包含隐层, 即可称为多层网络. 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

即神经元连接的权重.

“连接权”(connection weight) 以及每个功能神经元的阈值; 换言之, 神经网络“学”到的东西, 蕴涵在连接权与阈值中.

3. 误差逆传播算法（error backpropagation）

多层网路的学习能力比单层感知机强得多，预训练多层网络，简单感知机学习规则显然不够，需要更强大的学习算法，误差逆传播（error backpropagation ,简称BP）算法就是其中最接触的代表，它是迄今最成功的神经网络学习算法，现实任务中使用神经网络时候，大多在使用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，如训练递归神经网络，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。

离散属性需先进行处理: 若属性值间存在"序"关系则可进行连续化; 否则通常转化为 k 维向量, k 为属性值数. 参见 3.2 节.

下面我们来看看 BP 算法究竟是什么样. 给定训练集 $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $\boldsymbol{y}_i \in \mathbb{R}^l$, 即输入示例由 d 个属性描述, 输出 l 维实值向量. 为便于讨论, 图 5.7 给出了一个拥有 d 个输入神经元、l 个输出神经元、q 个隐层神经元的多层前馈网络结构, 其中输出层第 j 个神经元的阈值用 θ_j 表示, 隐层第 h 个神经元的阈值用 γ_h 表示. 输入层第 i 个神经元与隐层第 h 个神经元之间的连接权为 v_{ih}, 隐层第 h 个神经元与输出层第 j 个神经元之间的连接权为 w_{hj}. 记隐层第 h 个神经元接收到的输入为 $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$, 输出层第 j 个神经元接收到的输入为 $\beta_j = \sum_{h=1}^{q} w_{hj} b_h$, 其中 b_h 为隐层第 h 个神经

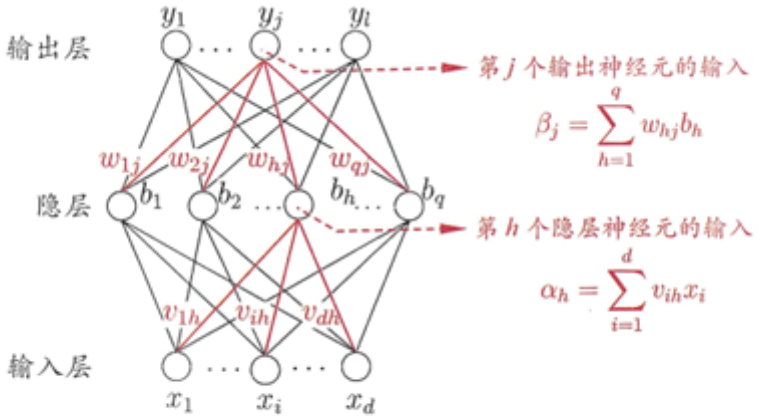


图 5.7 BP 网络及算法中的变量符号

元的输出. 假设隐层和输出层神经元都使用图 5.2(b) 中的 Sigmoid 函数.

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

$$\hat{y}_j^k = f(\beta_j - \theta_j) \,, \tag{5.3}$$

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

$$E_k = \frac{1}{2} \sum_{j=1}^{l} (\hat{y}_j^k - y_j^k)^2 \,. \tag{5.4}$$

图 5.7 的网络中有 $(d + l + 1)q + l$ 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, 即与式(5.1)类似, 任意参数 v 的更新估计式为

$$v \leftarrow v + \Delta v \,. \tag{5.5}$$

下面我们以图 5.7 中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(gradient descent)策略, 以目标的负梯度方向对参数进行调整. 对式(5.4)的误差 E_k, 给定学习率 η, 有

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} \,. \tag{5.6}$$

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

这就是“链式法则”.

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} . \tag{5.7}$$

根据 β_j 的定义, 显然有

$$\frac{\partial \beta_j}{\partial w_{hj}} = b_h . \tag{5.8}$$

图 5.2 中的 Sigmoid 函数有一个很好的性质:

$$f'(x) = f(x)(1 - f(x)) , \tag{5.9}$$

于是根据式(5.4)和(5.3), 有

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) . \end{aligned} \tag{5.10}$$

将式(5.10)和(5.8)代入式(5.7), 再代入式(5.6), 就得到了BP 算法中关于 w_{hj} 的更新公式

$$\Delta w_{hj} = \eta g_j b_h . \tag{5.11}$$

类似可得

$$\Delta \theta_j = -\eta g_j , \tag{5.12}$$

$$\Delta v_{ih} = \eta e_h x_i , \tag{5.13}$$

$$\Delta \gamma_h = -\eta e_h , \tag{5.14}$$

式(5.13)和(5.14)中

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

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

$$= b_h(1-b_h) \sum_{j=1}^{l} w_{hj} g_j \ . \tag{5.15}$$

常设置为 $\eta = 0.1$.

学习率 $\eta \in (0,1)$ 控制着算法每一轮迭代中的更新步长, 若太大则容易振荡, 太小则收敛速度又会过慢. 有时为了做精细调节, 可令式(5.11) 与(5.12) 使用 η_1, 式(5.13) 与(5.14) 使用 η_2, 两者未必相等.

图 5.8 给出了 BP 算法的工作流程. 对每个训练样例, BP 算法执行以下操作: 先将输入示例提供给输入层神经元, 然后逐层将信号前传, 直到产生输出层的结果; 然后计算输出层的误差(第 4–5 行), 再将误差逆向传播至隐层神经元(第 6 行), 最后根据隐层神经元的误差来对连接权和阈值进行调整(第 7 行). 该迭代过程循环进行, 直到达到某些停止条件为止, 例如训练误差已达到一个很小的值. 图 5.9 给出了在 2 个属性、5 个样本的西瓜数据上, 随着训练轮数的增加, 网络参数和分类边界的变化情况.

停止条件与缓解 BP 过拟合的策略有关.

输入: 训练集 $D = \{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{m}$;
学习率 η.

过程:

1: 在$(0,1)$范围内随机初始化网络中所有连接权和阈值
2: **repeat**
3: **for all** $(\boldsymbol{x}_k, \boldsymbol{y}_k) \in D$ **do**
4: 根据当前参数和式(5.3) 计算当前样本的输出 $\hat{\boldsymbol{y}}_k$;
5: 根据式(5.10) 计算输出层神经元的梯度项 g_j;
6: 根据式(5.15) 计算隐层神经元的梯度项 e_h;
7: 根据式(5.11)-(5.14) 更新连接权 w_{hj}, v_{ih} 与阈值 θ_j, γ_h
8: **end for**
9: **until** 达到停止条件

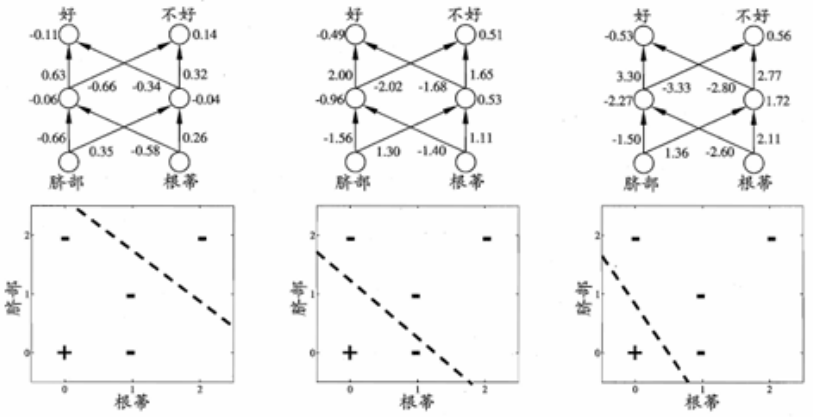
输出: 连接权与阈值确定的多层前馈神经网络

图 5.8 误差逆传播算法

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

$$E = \frac{1}{m} \sum_{k=1}^{m} E_k \ , \tag{5.16}$$

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, 图 5.8 中算法的更新规则是基于单个的 E_k 推导而得. 如



(a) 第 25 轮　(b) 第 50 轮　(c) 第 100 轮

图 5.9　在2个属性、5个样本的西瓜数据上, BP网络参数更新和分类边界的变化情况

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(accumulated error backpropagation) 算法. 累积 BP 算法与标准 BP 算法都很常用. 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, 它在读取整个训练集 D 一遍后才对参数进行更新, 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 非常大时更明显.

读取训练集一遍称为进行了“一轮”(one round, 亦称 one epoch)学习.

标准 BP 算法和累积 BP 算法的区别类似于随机梯度下降(stochastic gradient descent, 简称 SGD)与标准梯度下降之间的区别.

[Hornik et al., 1989] 证明, 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决问题, 实际应用中通常靠“试错法”(trial-by-error)调整.

正是由于其强大的表示能力, 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”(early stopping): 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”(regularization) [Barron, 1991; Girosi et al., 1995], 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

引入正则化策略的神经网络与第 6 章的 SVM 已非常相似.

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(5.16) 改变为

$$E = \lambda \frac{1}{m} \sum_{k=1}^{m} E_k + (1-\lambda) \sum_i w_i^2 , \tag{5.17}$$

增加连接权与阈值平方和这一项后, 训练过程将会偏好比较小的连接权和阈值, 使网络输出更加“光滑”, 从而对过拟合有所缓解.

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.

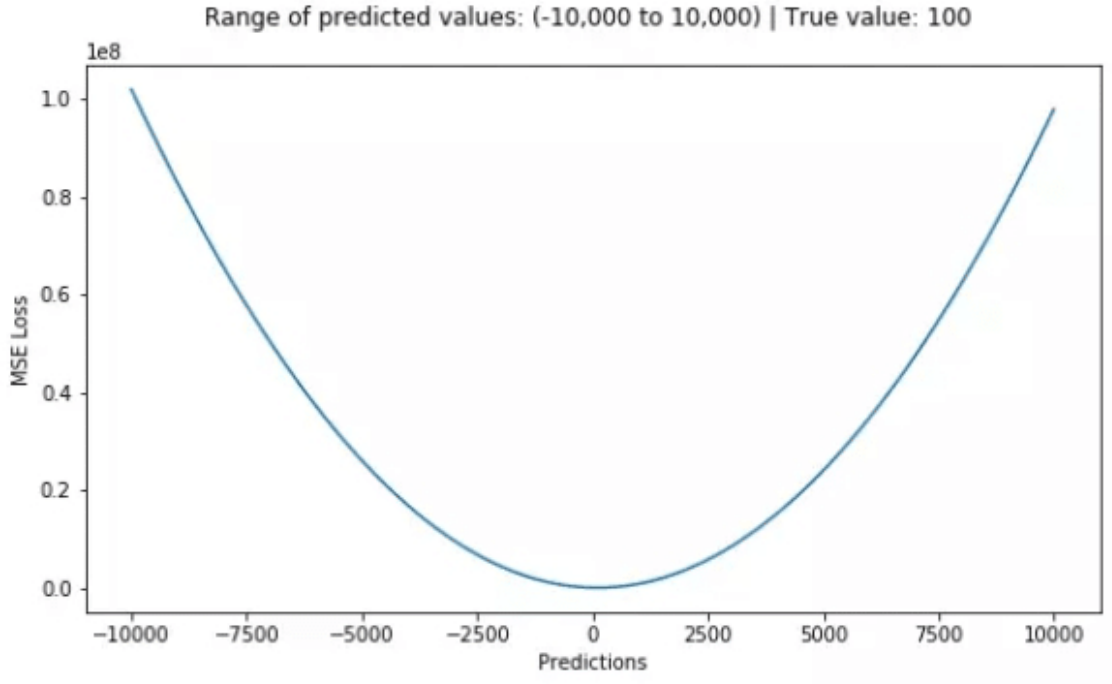
上述计算步骤通过链式求导实现对神经元权重进行更新

3.1 均方误差 (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

均方误差（MSE）是度量预测值于真实值之间的差的平方均值，它只考虑误差的平均大小，不考虑其方向，但由于经过平方，让真实值偏离较多的预测值比偏离较小的预测值受到的更多的权重惩罚。再加上MSE的数学特性很好，这使得计算梯度变得更加容易。

下图是MSE 函数的图像，其中目标值是 100，预测值的范围从 -10000 到 10000，Y 轴代表的 MSE 取值范围是从0 到正无穷，并且在预测值为 100 处变得最小。



MSE損失（Y軸）-預測值（X軸）

又被称为L2 损失 或 L2范式损失/平方损失

3.2 梯度下降算法（Gradient Descent）

B.4 梯度下降法

一阶方法仅使用目标函数的一阶导数, 不利用其高阶导数.

梯度下降法(gradient descent)是一种常用的一阶(first-order)优化方法, 是求解无约束优化问题最简单、最经典的方法之一.

考虑无约束优化问题 $\min_{\boldsymbol{x}} f(\boldsymbol{x})$, 其中 $f(\boldsymbol{x})$ 为连续可微函数. 若能构造一个序列 $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ 满足

$$f(\boldsymbol{x}^{t+1}) < f(\boldsymbol{x}^t), \quad t = 0, 1, 2, \ldots \tag{B.15}$$

则不断执行该过程即可收敛到局部极小点. 欲满足式(B.15), 根据泰勒展式有

$$f(\boldsymbol{x} + \Delta\boldsymbol{x}) \simeq f(\boldsymbol{x}) + \Delta\boldsymbol{x}^{\mathrm{T}} \nabla f(\boldsymbol{x}) \,, \tag{B.16}$$

于是, 欲满足 $f(\boldsymbol{x} + \Delta\boldsymbol{x}) < f(\boldsymbol{x})$, 可选择

$$\Delta\boldsymbol{x} = -\gamma \nabla f(\boldsymbol{x}) \,, \tag{B.17}$$

每步的步长 γ_t 可不同.

其中步长 γ 是一个小常数. 这就是梯度下降法.

L-Lipschitz条件是指对于任意 $\boldsymbol{x}$, 存在常数 L 使得 $\|\nabla f(\boldsymbol{x})\| \leqslant L$ 成立.

若目标函数 $f(\boldsymbol{x})$ 满足一些条件, 则通过选取合适的步长, 就能确保通过梯度下降收敛到局部极小点. 例如若 $f(\boldsymbol{x})$ 满足 L-Lipschitz 条件, 则将步长设置为 $1/(2L)$ 即可确保收敛到局部极小点. 当目标函数为凸函数时, 局部极小点就对应着函数的全局最小点, 此时梯度下降法可确保收敛到全局最优解.

当目标函数 $f(\boldsymbol{x})$ 二阶连续可微时, 可将式(B.16)替换为更精确的二阶泰勒展式, 这样就得到了牛顿法(Newton's method). 牛顿法是典型的二阶方法, 其迭代轮数远小于梯度下降法. 但牛顿法使用了二阶导数 $\nabla^2 f(\boldsymbol{x})$, 其每轮迭代中涉及到海森矩阵(A.21)的求逆, 计算复杂度相当高, 尤其在高维问题中几乎不可行. 若能以较低的计算代价寻找海森矩阵的近似逆矩阵, 则可显著降低计算开销, 这就是拟牛顿法(quasi-Newton method).

4. 全局最小与局部最小

若用 E 表示神经网络在训练集上的误差, 则它显然是关于连接权 $\boldsymbol{w}$ 和阈值 θ 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 E 最小.

这里的讨论对其他机器学习模型同样适用.

我们常会谈到两种“最优”: “局部极小”(local minimum)和“全局最小”(global minimum). 对 $\boldsymbol{w}^*$ 和 θ^*, 若存在 $\epsilon > 0$ 使得

$$\forall\ (\boldsymbol{w};\theta) \in \{(\boldsymbol{w};\theta) \mid \|(\boldsymbol{w};\theta) - (\boldsymbol{w}^*;\theta^*)\| \leqslant \epsilon\}\ ,$$

都有 $E(\boldsymbol{w};\theta) \geqslant E(\boldsymbol{w}^*;\theta^*)$ 成立, 则 $(\boldsymbol{w}^*;\theta^*)$ 为局部极小解; 若对参数空间中的任意 $(\boldsymbol{w};\theta)$ 都有 $E(\boldsymbol{w};\theta) \geqslant E(\boldsymbol{w}^*,\theta^*)$, 则 $(\boldsymbol{w}^*;\theta^*)$ 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 $E(\boldsymbol{w}^*;\theta^*)$ 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. 也就是说, “全局最小”一定是“局部极小”, 反之则不成立. 例如, 图 5.10 中有两个局部极小, 但只有其中之一是全局最小. 显然,我们在参数寻优过程中是希望找到全局最小.

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, 我们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

感知机更新规则式 (5.1) 和 BP 更新规则式 (5.11)-(5.14) 都是基于梯度下降.

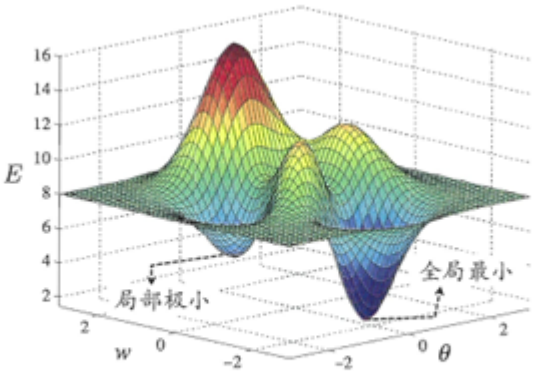


图 5.10 全局最小与局部极小

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:

- 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.
- 使用“模拟退火”(simulated annealing) 技术 [Aarts and Korst, 1989]. 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.

但是也会造成“跳出”全局最小.

- 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) [Goldberg, 1989] 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, 理论上尚缺乏保障.

5. 其他常见神经网络

5.1 RBF神经网络

5.1.1 径向基函数

径向基函数是取值仅仅依赖于离原点距离的实值函数即$\varphi(x)=\varphi(||\ x\ ||)$(也可以是离任意点的距离,距离一般使用欧式距离[欧式径向基函数],也可以是其他距离函数),任何满足该特性的函数都φ都叫做径向基函数,通常定义为空间中任一点x到某一中心c之间区民距离的单调函数

常见的径向基函数包括(定义$r=|\ x-ci\ ||\ ciO$称为径向基函数的扩展函数(方差),

它反应了函数图像的宽度,σ越小,宽度越窄,函数越具有选择性)

$Gauss$(高斯)函数:$\varphi(r)=exp(-2\sigma 2r2)$

反演S型函数:$\varphi(r)=1+exp(\sigma 2r21)$

拟多二次函数:$\varphi(r)=(r2+\sigma 2)211$

$$\text{拟多二次函数:}\ \varphi(r)=\frac{1}{(r^2+\sigma^2)^{\frac{1}{2}}}$$

$$\text{反演}S\text{型函数:}\ \varphi(r)=\frac{1}{1+\exp\left(\frac{r^2}{\sigma^2}\right)}$$

拟多二次函数：$\varphi(r)=\frac{1}{(r^2+\sigma^2)^{\frac{1}{2}}}$

5.1.2 结构

RBF网络是一种三层前向网络，第一层为由信号源节点组成的输入层，第二层为隐层，隐单元数视问题需要而定，隐单元的变换函数为非负非线性的函数RBF（径向基函数），第三层为输出层，输出层是对隐层神经元输出的线性组合

RBF（Radial Basis Function ,径向基函数）网络是一种单隐层前馈神经网络，它是由径向基函数作为隐层神经元激活函数，而输出层则是对隐藏神经元输出的线性组合，假定输入为d维向量x，输出为实值，则RBF网络可表示为

$$\varphi(\boldsymbol{x})=\sum_{i=1}^{q} w_i \rho(\boldsymbol{x}, \boldsymbol{c}_i)\ , \tag{5.18}$$

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 $\boldsymbol{c}_i$ 之间欧氏距离的单调函数. 常用的高斯径向基函数形如

$$\rho(\boldsymbol{x}, \boldsymbol{c}_i)=e^{-\beta_i\|\boldsymbol{x}-\boldsymbol{c}_i\|^2}\ . \tag{5.19}$$

[Park and Sandberg, 1991] 证明, 具有足够多隐层神经元的 RBF 网络能以任意精度逼近任意连续函数.

通常采用两步过程来训练 RBF 网络: 第一步, 确定神经元中心 $\boldsymbol{c}_i$, 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.

5.2 ART神经网络

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.

ART(Adaptive Resonance Theory, 自适应谐振理论)网络 [Carpenter and Grossberg, 1987] 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.

模式类可认为是某类别的“子类”.

在接收到比较层的输入信号后, 识别层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. 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.

这就是“胜者通吃”原则的体现.

显然, 识别阈值对ART网络的性能有重要影响. 当识别阈值较高时, 输入样本将会被分成比较多、比较精细的模式类, 而如果识别阈值较低, 则会产生比较少、比较粗略的模式类.

ART比较好地缓解了竞争型学习中的“可塑性-稳定性窘境”(stability-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).

增量学习是指在学得模型后, 再接收到训练样例时, 仅需根据新样例对模型进行更新, 不必重新训练整个模型, 并且先前学得的有效信息不会被“冲掉”; 在线学习是指每获得一个新样本就进行一次模型更新. 显然, 在线学习是增量学习的特例, 而增量学习可视为“批模式”(batch-mode)的在线学习.

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.

5.3 SOM网络

SOM(Self-Organizing Map, 自组织映射)网络 [Kohonen, 1982] 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.

亦称“自组织特征映射”(Self-Organizing Feature Map)、Kohonen 网络.

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.

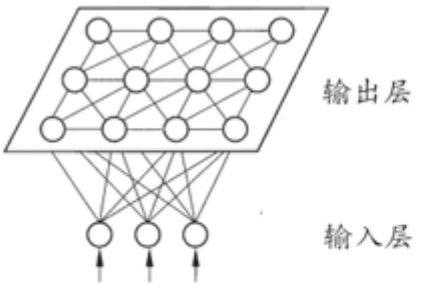


图 5.11 SOM 网络结构