误差逆传播算法（BP）

前言

我们已经知道了关于神经网络的基础。下面我们需要思考一个问题：我们知道单层的感知机没办法胜任实际任务，在这种问题的基础上我们肯定想到的是使用多层网络，但是，我们如何有效地训练包含隐藏层的多层网络?

于是，我们就有了反向传播算法。简单来说，BP算法是一种用于训练人工神经网络（ANN）的监督学习算法。它的核心工作可以概括为：***“教会”一个多层神经网络如何通过调整内部参数（连接权重和偏置），使得其输出结果尽可能接近我们期望的目标值***。也就是说，我们的侧重还是在“连接权重和偏置”（~~离不开子~~

想象一下你在教一个孩子认动物。你给他看一张猫的图片（输入），他会给出一个答案（输出），比如“狗”。你会告诉他“不对，这是猫”（计算误差）。接下来关键的一步是：孩子需要知道是哪个判断环节出了错——是耳朵的形状看错了？还是胡须的特征没注意到？然后他会有针对性地调整自己的判断依据。

反向传播算法所类似的事情：

- 前向传播：输入数据从输入层，经过隐藏层，逐层计算，最终产生输出（孩子的猜测）。
- 反向传播：将输出层的误差，沿着与之前相反的方向，逐层反向传播回去，从而计算出每个神经元连接权重的“责任”（误差梯度）。
- 参数更新：根据计算出的梯度，使用优化算法（如梯度下降法）来更新网络中的每一个权重和偏置（调整判断依据）。

所以BP不是网络，而是一个训练网络的“教练”，他制定了一套训练方案。

注：下面内容参考西瓜书，并作注解（~~权威~~

算法推导

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，下图给出了一个拥有 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反馈网络结构，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 $\alpha_h=\sum_{d}^{i=1}v_{ih}x_i$，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 个神经元的输出。假设隐层和输出层神经元都是用Sigmoid函数。

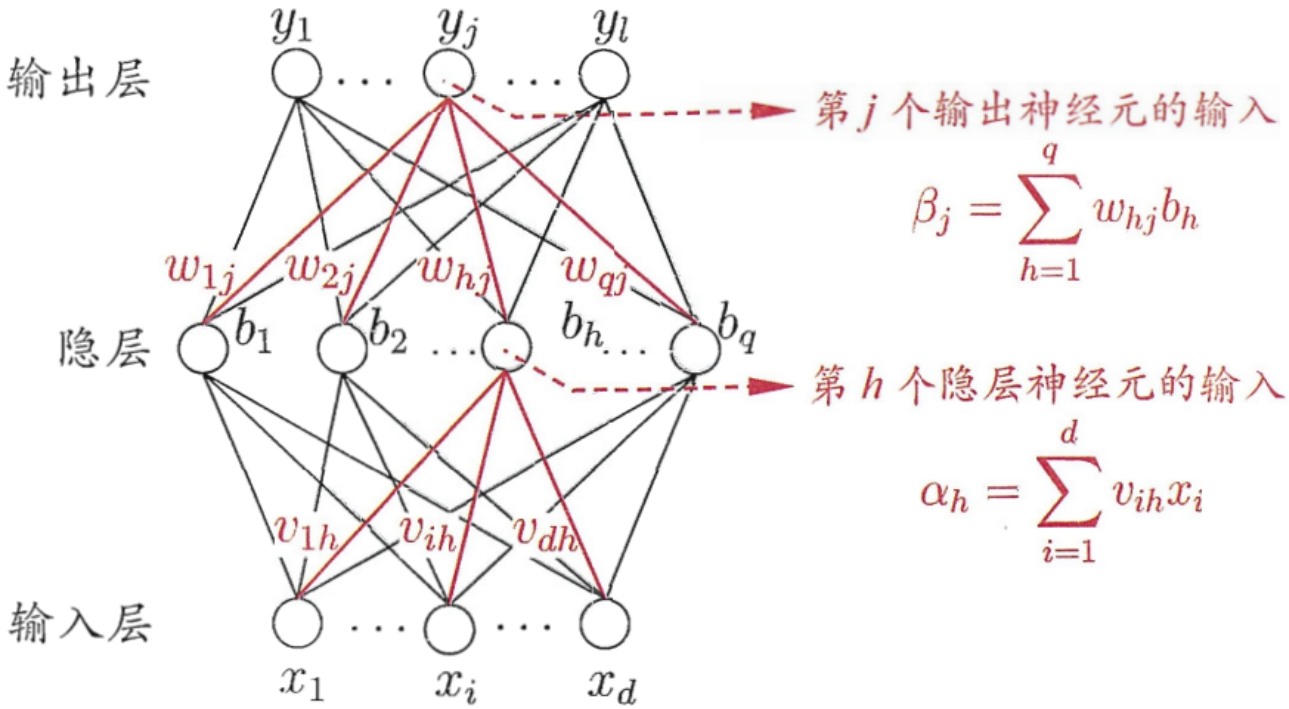


图 5.7　BP 网络及算法中的变量符号

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

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

这里的 β_j 是第 j 个神经元的输入，θ_j 代表第 j 个神经元的阈值，输入减去阈值得到的数带入激活函数就得到了本节点的输出

但是我们还需要知道其输出值与真实值的均方误差：

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

这里的的均方差是用来表示该节点的误差的，其目的是为了后面的最小化误差。至于一些参数，比如 $\frac{1}{2}$ 是为了好求导，后边的 $\hat{y}_j^k - y_j^k$ 这是预测值减去真实值，得到误差。

在上图中，网络有 $(d + l + 1)q + l$ 个参数需要确认：其展开就是 dq 和 ql 个互联接的权，q 和 l 个阈值。BP算法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 感知机与多层网络 中的 (5.1) 类似，任意参数 v 的更新估计式为

$$v \leftarrow v + \Delta v, \quad (5.5)$$

下面我们以上图中的隐层到输出层的连接权 w_{hj} 为例来进行推导。

参数求解

BP算法基于梯度下降（GD，gradient descent）策略，以目标的负梯度方向对参数进行调整。我们现在可以将目光关注到 (5.4) 的误差 E_k 上了。给定学习率 η ，直接对该式子进行求梯度（对矩阵中的 w_{hj} 求偏导）：

$$\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}}, \quad (5.6)$$

根据链式法则，我们得到了（不知道什么是链式法则的回去重修微积分）：

$$\frac{\partial E_k}{\partial w_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}, \quad (5.7)$$

根据 β_j 的定义，有：

$$\frac{\partial \beta_j}{\partial w_{hj}} = \frac{\partial \sum_{h=1}^{q} w_{hj} b_h}{\partial w_{hj}} = b_h, \quad (5.8)$$

现在，我们来看Sigmoid函数，对其进行求导得到：

$$\begin{aligned} f'(x) &= \frac{0 \times (1+e^{-x}) - 1 \times (-e^{-x})}{(1+e^{-x})^2} \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \times \frac{e^{-x}}{1+e^{-x}} \\ &= \frac{1}{1+e^{-x}} \times \left(1 - \frac{1}{1+e^{-x}}\right) \\ &= f(x)(1-f(x)) \end{aligned}, \quad (5.9)$$

将 $\hat{y}_j^k = f(\beta_j - \theta_j), (5.3)$ 代入 (5.9) 得到：

$$\begin{aligned} \frac{\partial \hat{y}_j^k}{\partial \beta} &= f'(\beta_j - \theta_j) \\ &= f(\beta_j - \theta_j)(1 - f(\beta_j - \theta_j)), \\ &= \hat{y}_j^{k\prime} = \hat{y}_j^k (1 - \hat{y}_j^k) \end{aligned} \quad (5.9.1)$$

于是，我们就根据 (5.4) 、 (5.3) 和 $(5.9.1)$ ，定义 g_j ：

$$
\begin{aligned}
g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\
&= -\frac{\partial \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta}, \quad (5.10) \\
&= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\
&= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)
\end{aligned}
$$

将式 (5.10) 和 (5.8) 代入 (5.7) 得到：

$$
\begin{aligned}
\frac{\partial E_k}{\partial w_{hj}} &= \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}} \\
&= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) \cdot b_h \\
&= f(\beta_j - \theta_j)(1 - f(\beta_j - \theta_j))(y_j^k - f(\beta_j - \theta_j)) \cdot b_h \quad , \quad (5.7.1) \\
&= f(\sum_{h=1}^{q} w_{hj} b_h - \theta_j)(1 - f(\sum_{h=1}^{q} w_{hj} b_h - \theta_j))(y_j^k - f(\sum_{h=1}^{q} w_{hj} b_h - \theta_j)) \cdot b_h \\
&= -g_j \cdot b_h
\end{aligned}
$$

这是完整的公式，当然我们前边定义了 g_j ，代换得到：

$$
\text{隐层-输出层权} \Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}} = \eta g_j \cdot b_h, \quad (5.11)
$$

不过，这种只是其中的一个参数，这样的参数我们还有 3 个🤯：分别对隐层和输出层的参数进行参数求解，得到

$$
\text{输出层阈值} \Delta \theta_j = -\eta g_j, \quad (5.12)
$$

$$
\text{输入层-隐层权} \Delta v_{ih} = \eta e_h x_i, \quad (5.13)
$$

$$
\text{隐层阈值} \Delta \gamma_h = -\eta e_h, \quad (5.14)
$$

详细的推导大家可以参考南瓜书，这部分还是太吃操作了（

这三个式子中存在和 g_j 一样的为了简化表示，新定义部分：

$$
\begin{aligned}
e_h &= -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \\
&= -\sum_{j=1}^{l} \frac{\partial E_k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} f'(\alpha_h - \gamma_h) \\
&= \sum_{j=1}^{l} w_{hj} g_j f'(\alpha_h - \gamma_h) \qquad , \quad (5.15) \\
&= b_h(1-b_h) \sum_{j=1}^{l} w_{hj} g_j \\
&= \text{哈基米no南北路多（bushi}
\end{aligned}
$$

学习率

学习率 $\eta \subseteq (0,1)$ 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。

流程

👉图给出了 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👇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。

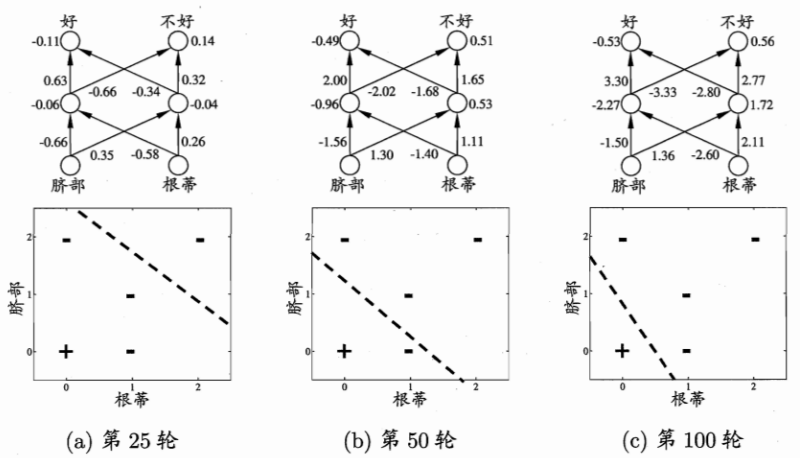
输入： 训练集 $D = \{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{m}$；
学习率 η.

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



(a) 第 25 轮　(b) 第 50 轮　(c) 第 100 轮

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

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k\,, \quad (5.16)$$

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

注：*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

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

其中 $\lambda \subseteq (0,1)$ 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。