

(3 条消息) 神经网络的 BP 算法推导详解_知识搬运工的博客 - CSDN 博客_bp 神经网络算法步骤

一个神经网络程序包含以下几部分内容。

1. 数据表达和特征提取。对于一个非深度学习神经网络，主要影响其模型准确度的因素就是数据表达和特征提取。同样的一组数据，在欧式空间和非欧空间，就会有着不同的分布。有时候换一种思考问题的思路就会使得问题变得简单。所以选择合适的数据表达可以极大的降低解决问题的难度。同样，在机器学习中，特征的提取也不是一种简单的事。在一些复杂问题上，要通过人工的方式设计有效的特征集合，需要很多的时间和精力，有时甚至需要整个领域数十年的研究投入。例如，PCA 独立成分分析就是特征提取中常用的手段之一。但是很多情况下，人为都很难提取出合适的特征。

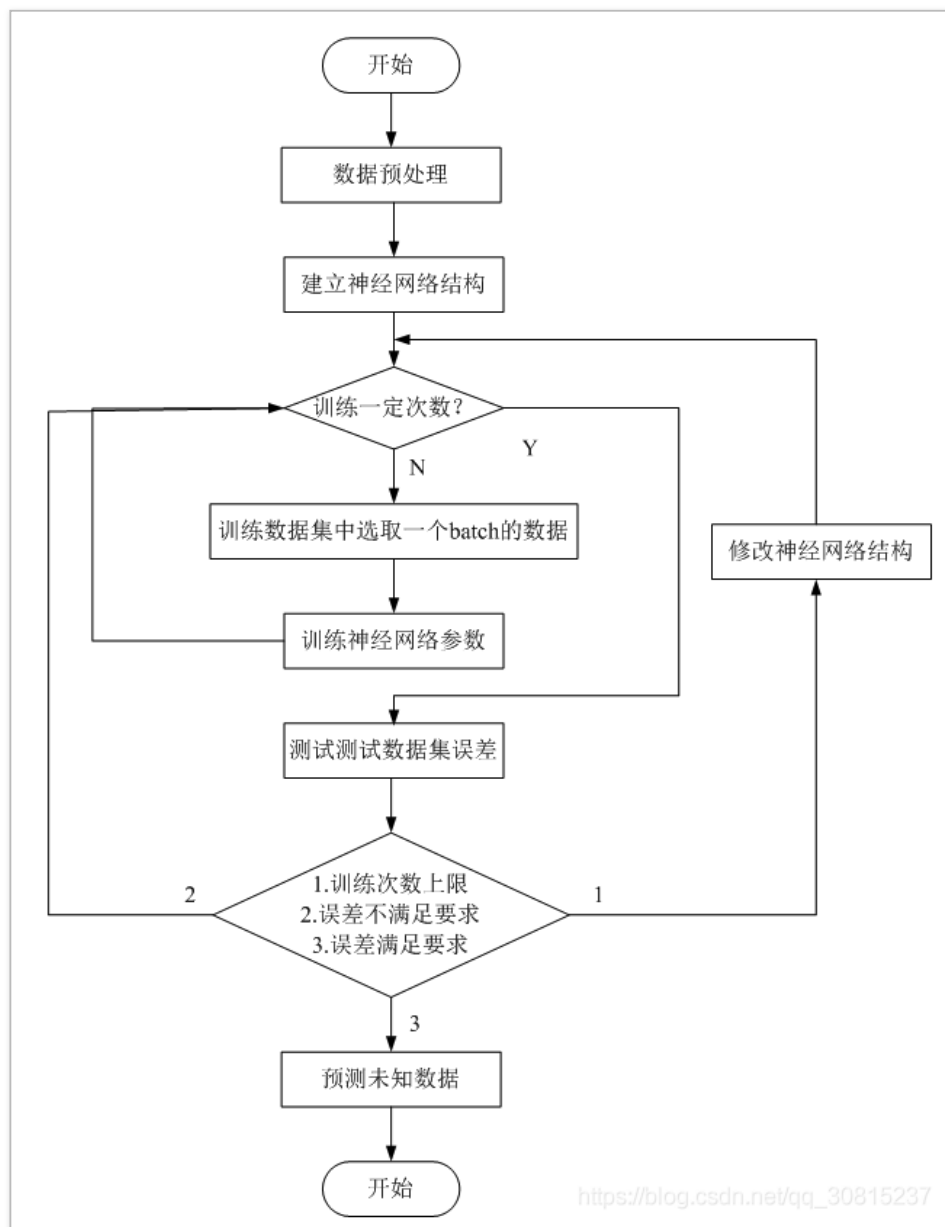
由于不同问题下，可以提取不同的特征向量，这里将不做具体介绍。事实上深度学习解决的核心问题之一就是自动地将简单的特征组合成更加复杂的特征，并使用这些组合特征解决问题。

2. 定义神经网络的结构。由神经网络发展的历史可知，不同结构的神经网络在不同的问题下得到的效果不同。因此分析问题，选择与问题合适的神经网络结构也同样重要。

3. 训练神经网络的参数。使用训练数据集训练神经网络。主要是利用神经网络输出误差反向传播修正神经网络中的参数，甚至结构。反向传播过程中，步长选择对神经网络的训练有着重要的影响，在此基础上产生了多种训练方法。将在后面介绍。

4. 使用训练好的神经网络预测未知数据。训练神经网络的目的就是对未知数据预测。

具体过程可以由如下流程图表示：



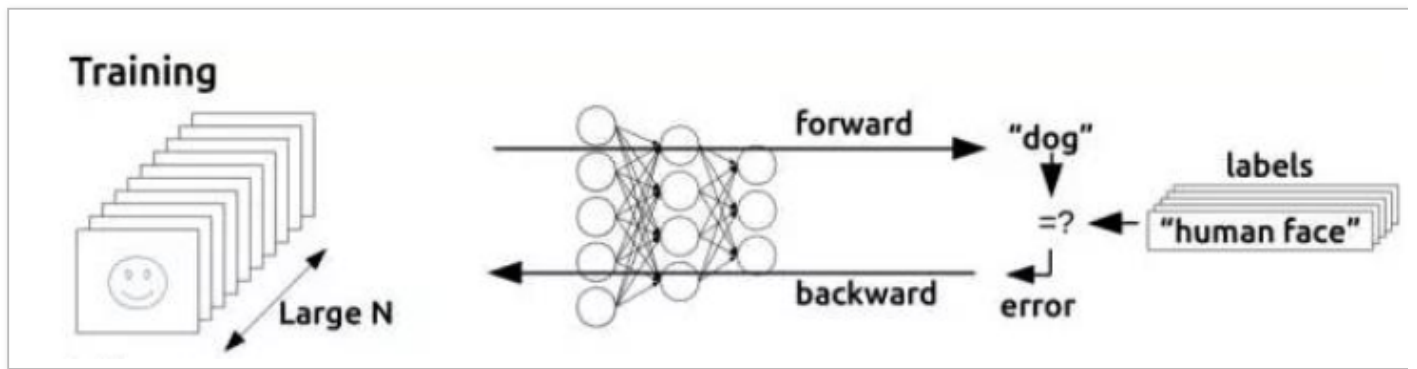
一. BP 算法的提出及其算法思想

神经网络主要是由三个部分组成的, 分别是: 1) 网络架构 2) 激活函数 3) 找出最优权重值的参数学习算法.

BP 算法就是目前使用较为广泛的一种参数学习算法.

BP(back propagation) 神经网络是 1986 年由 Rumelhart 和 McClelland 为首的科学家提出的概念, 是一种按照误差逆向传播算法训练的多层前馈神经网络。

既然我们无法直接得到隐层的权值, 能否先通过输出层得到输出结果和期望输出的误差来间接调整隐层的权值呢? BP 算法就是采用这样的思想设计出来的算法, 它的基本思想: 学习过程由信号的正向传播 (求损失) 与误差的反向传播 (误差回传) 两个过程组成。如图所示为 BP 算法模型示意图:



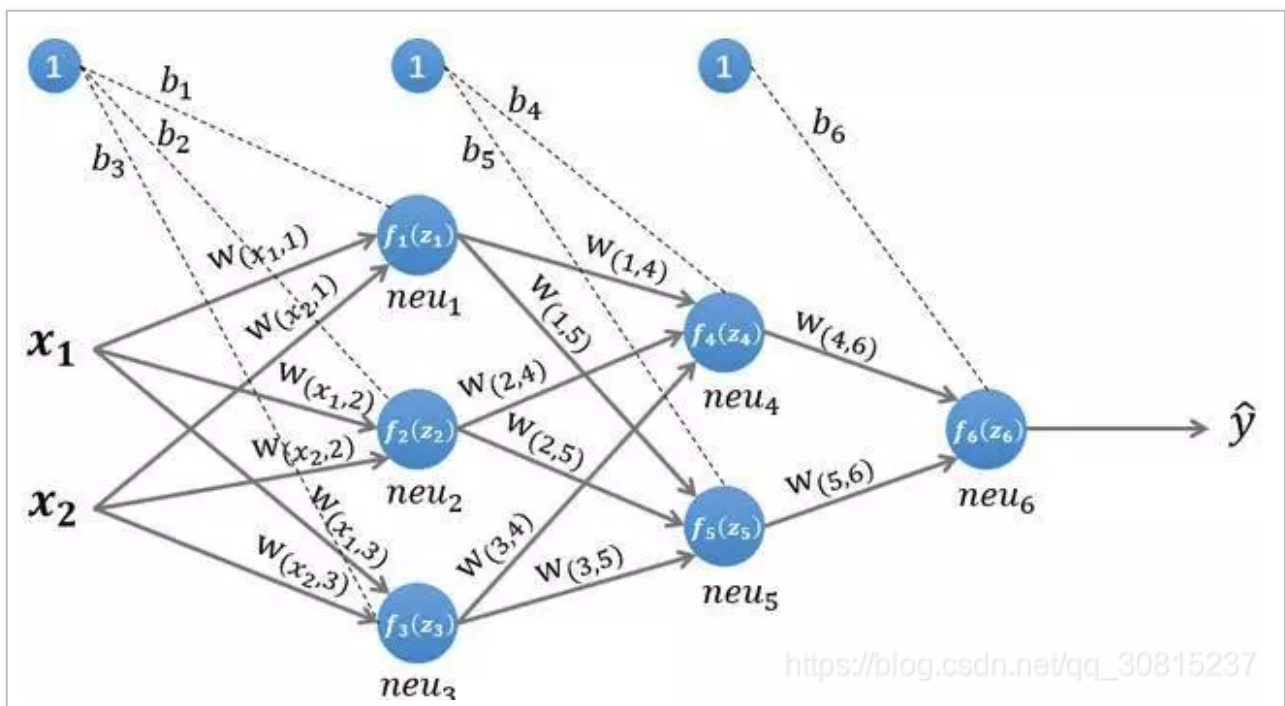
二. BP 算法

BP 算法的一般流程：

1. 正向传播 FP(求损失). 在这个过程中, 我们根据输入的样本, 给定的初始化权重值 W 和偏置项的值 b , 计算最终输出值以及输出值与实际值之间的损失值. 如果损失值不在给定的范围内则进行反向传播的过程; 否则停止 W, b 的更新.
2. 反向传播 BP(回传误差). 将输出以某种形式通过隐层向输入层逐层反传, 并将误差分摊给各层的所有单元, 从而获得各层单元的误差信号, 此误差信号即作为修正各单元权值的依据。

由于 BP 算法是通过传递误差值 δ 进行更新求解权重值 W 和偏置项的值 b , 所以 BP 算法也常常被叫做 δ 算法.

下面我们将以下图所示的神经网络, 该图所示是一个三层神经网络, 两层隐藏层和一层输出层, 输入层有两个神经元, 接收输入样本, 为网络的输出。



三. 前馈计算的过程

为了理解神经网络的运算过程，我们需要先搞清楚前馈计算，即数据沿着神经网络前向传播的计算过程，以上图所示的网络为例，输入的样本为：

$$\vec{a} = (x_1, x_2)$$

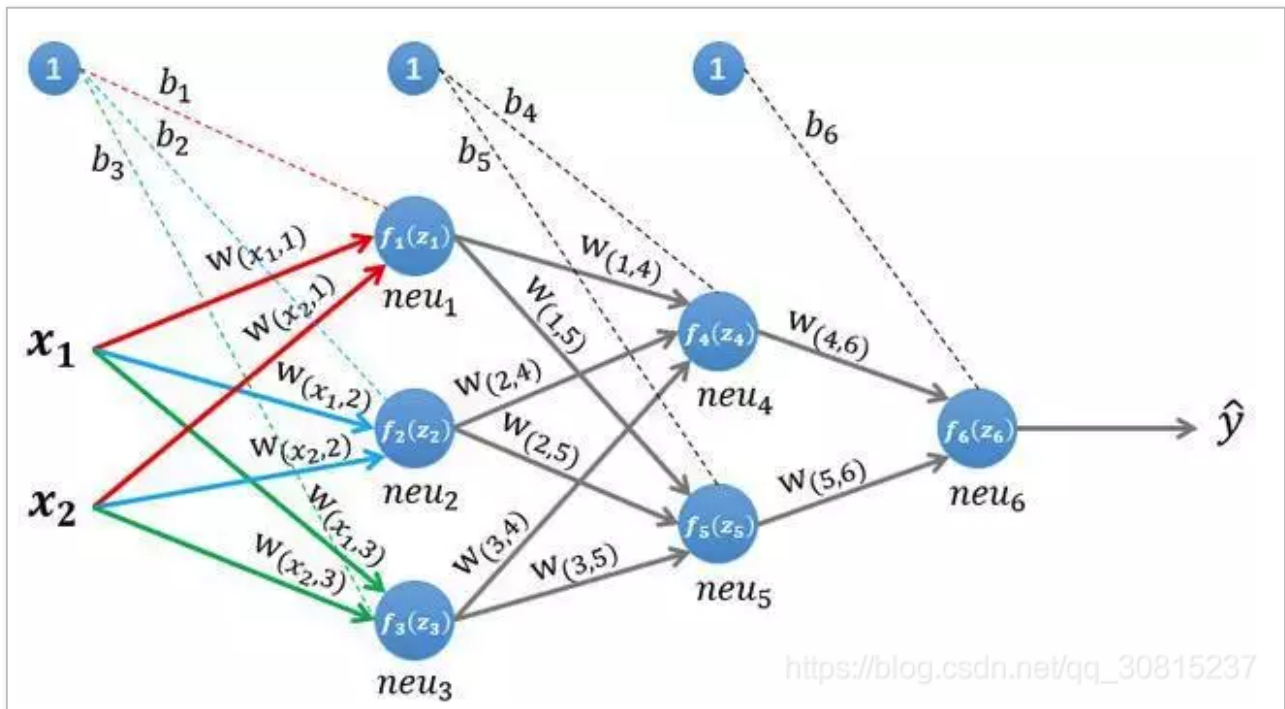
三层网络的参数定义为：

$$W^{(1)} = \begin{bmatrix} W_{(x_1,1)}, W_{(x_2,1)} \\ W_{(x_1,2)}, W_{(x_2,2)} \\ W_{(x_1,3)}, W_{(x_2,3)} \end{bmatrix}, \quad b^{(1)} = [b_1, b_2, b_3]$$

$$W^{(2)} = \begin{bmatrix} W_{(1,4)}, W_{(2,4)}, W_{(3,4)} \\ W_{(1,5)}, W_{(2,5)}, W_{(3,5)} \end{bmatrix}, \quad b^{(2)} = [b_4, b_5]$$

$$W^3 = [W_{(4,6)}, W_{(5,6)}], \quad b^{(3)} = [b_6]$$

第一层隐藏层的计算



第一层隐藏层有三个神经元：neu₁、neu₂和 neu₃该层的输入为：

$$z^{(1)} = W^{(1)} * (\vec{a})^T + (b^{(1)})^T$$

以单个神经元为例，则其输入为：

$$z_1 = w_{(x_1,1)} * x_1 + w_{(x_2,1)} * x_2 + b_1$$

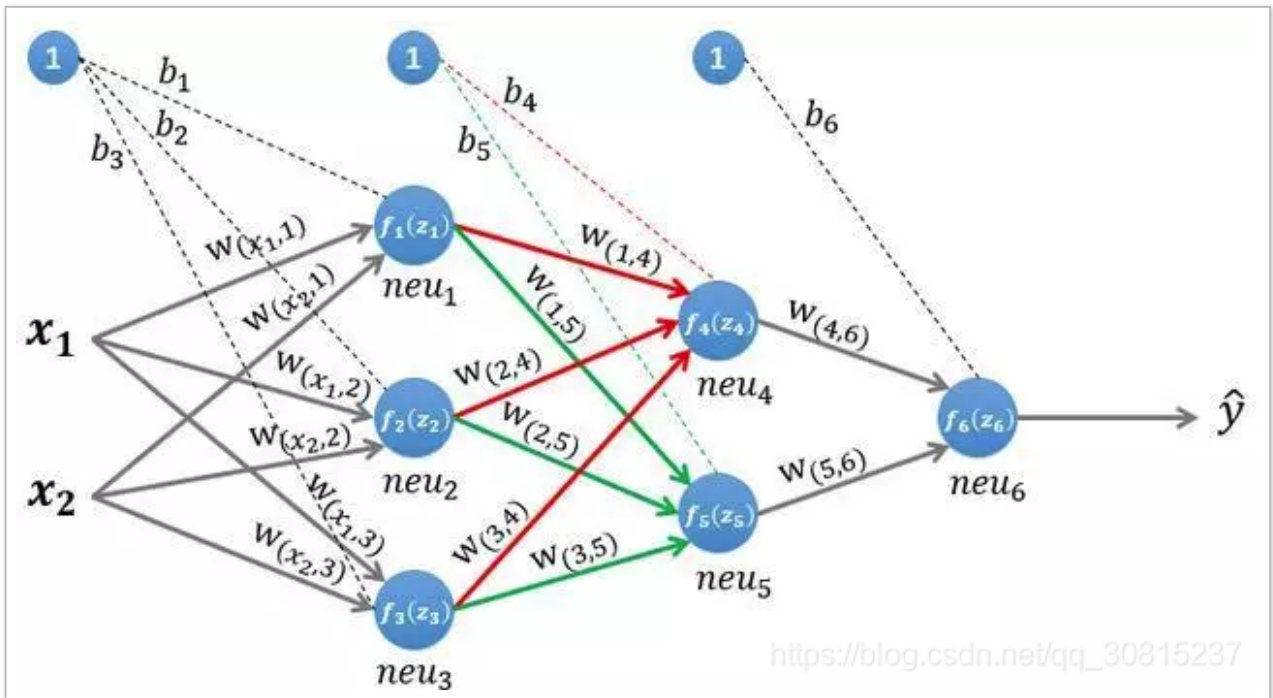
$$z_2 = w_{(x_1,2)} * x_1 + w_{(x_2,2)} * x_2 + b_2$$

$$z_3 = w_{(x_1,3)} * x_1 + w_{(x_2,3)} * x_2 + b_3$$

假设我们选择函数 $f(x)$ 作为该层的激活函数（图中的激活函数都标了一个下标，一般情况下，同一层的激活函数都是一样的，不同层可以选择不同的激活函数），那么该层的输出为：

$f_1(z_1)$ 、 $f_2(z_2)$ 和 $f_3(z_3)$ 。

第二层隐藏层的计算：



第二层隐藏层有两个神经元：neu₄和 neu₅。该层的输入为：

$$z^{(2)} = W^{(2)} * [z_1, z_2, z_3]^T + (b^{(2)})^T$$

即第二层的输入是第一层的输出乘以第二层的权重，再加上第二层的偏置。因此得到

z_4 和 z_5 的输入分别为：

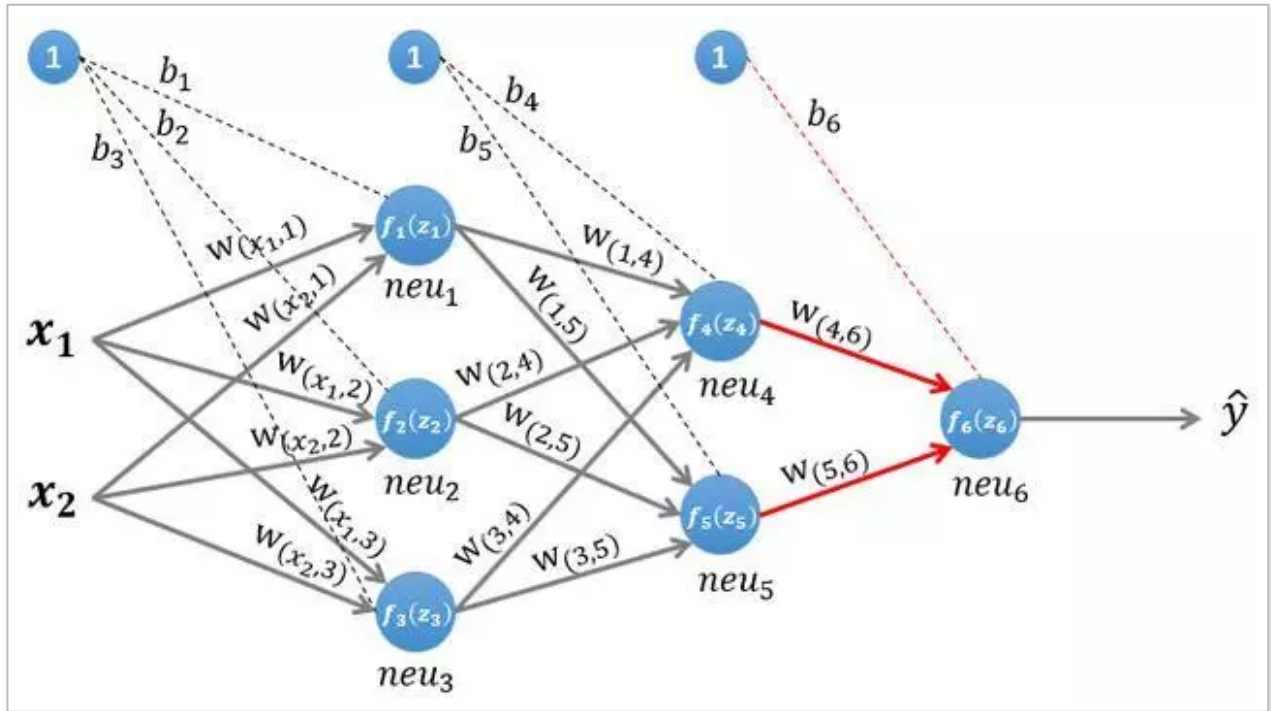
$$z_4 = w_{(1,4)} * z_1 + w_{(2,4)} * z_2 + w_{(3,4)} * z_3 + b_4$$

$$z_5 = w_{(1,5)} * z_1 + w_{(2,5)} * z_2 + w_{(3,5)} * z_3 + b_5$$

该层的输出分别为：

$f_4(z_4)$ 和 $f_5(z_5)$ 。

输出层的计算



输出层只有一个神经元：neu₆。该层的输入为：

$$z^{(3)} = W^{(3)} * [z_4, z_5]^T + (b^{(3)})^T$$

即：

$$z_6 = W_{(4,6)} * z_4 + W_{(5,6)} * z_5 + b_6$$

因为该网络要解决的是一个二分类问题，所以输出层的激活函数也可以使用一个 Sigmoid 型函数，神经网络最后的输出为

$$f_6(z_6)。$$

四. 反向传播的计算

我们已经了解了数据沿着神经网络前向传播的过程，这一节我们来介绍更重要的反向传播的计算过程。假设我们使用随机梯度下降的方式来学习神经网络的参数，损失函数定义为

$L(y, \hat{y})$ ，其中 y 是该样本的真实类标。使用梯度下降进行参数的学习，我们必须计算出损失函数关于神经网络中各层参数（权重 w 和偏置 b ）的偏导数。

假设我们要对第 k 层隐藏层的参数

$W^{(k)}$ 和求偏导数 $b^{(k)}$ 。假设 $z^{(k)}$ 代表第 k 层神经元的输入，即

$$z^{(k)} = W^{(k)} * n^{(k-1)} + b^{(k)}$$

其中

$n^{(k-1)}$ 为前一层神经元的输出，则根据链式法则有：

$$\frac{\partial L(y, \hat{y})}{\partial W^{(k)}} = \frac{\partial L(y, \hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial W^{(k)}}$$

$$\frac{\partial L(y, \hat{y})}{\partial b^{(k)}} = \frac{\partial L(y, \hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial b^{(k)}}$$

计算偏导数 1:

前面说过，第 k 层神经元的输入为：

$$z^{(k)} = W^{(k)} * n^{(k-1)} + b^{(k)},$$

因此可以得到：

$$\frac{\partial z^{(k)}}{\partial W^{(k)}} = \begin{bmatrix} \frac{\partial (W_{1:}^{(k)} * n^{(k-1)} + b^{(k)})}{\partial W^{(k)}} \\ \vdots \\ \frac{\partial (W_{m:}^{(k)} * n^{(k-1)} + b^{(k)})}{\partial W^{(k)}} \end{bmatrix} \xRightarrow{\text{初等变换}} (n^{(k-1)})^T$$

$$\frac{\partial z^{(k)}}{\partial b^{(k)}} = \begin{bmatrix} \frac{\partial (W_{1:}^{(k)} * n^{(k-1)} + b_1)}{\partial b_1} & \dots & \frac{\partial (W_{1:}^{(k)} * n^{(k-1)} + b_1)}{\partial b_m} \\ \vdots & \dots & \vdots \\ \frac{\partial (W_{m:}^{(k)} * n^{(k-1)} + b_m)}{\partial b_1} & \dots & \frac{\partial (W_{m:}^{(k)} * n^{(k-1)} + b_m)}{\partial b_m} \end{bmatrix}$$

上式中，

$W_{m:}^{(k)}$ 代表第 k 层神经元的权重矩阵的第 m 行， $W_{mn}^{(k)}$ 代表第 k 层神经元的权重矩阵的第 m 行中的第 n 列。

假设我们要计算第一层隐藏层的神经元关于权重矩阵的导数，则有：

$$\frac{\partial z^{(1)}}{\partial W^{(1)}} = (x_1, x_2)^T = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\frac{\partial z^{(1)}}{\partial b^{(1)}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

计算偏导数 2:

偏导数

$\frac{\partial L(y, \hat{y})}{\partial z^{(k)}}$ 又称为误差项（也称为“灵敏度”），其值的大小代表了第一层神经元对于最终总误差的影响大小。根据第一节的前向计算，我们知道第 $k + 1$ 层的输入与第 k 层的输出之间的关系为：

$$z^{(k+1)} = W^{(k+1)} * n^{(k)} + b^{k+1}$$

其中：

$n^{(k)} = f_k(z_k)$, 根据链式法则，我们可以得到：

$$\begin{aligned} \delta^{(k)} &= \frac{\partial L(y, \hat{y})}{\partial z^{(k)}} \\ &= \frac{\partial n^{(k)}}{\partial z^{(k)}} * \frac{\partial z^{(k+1)}}{\partial n^{(k)}} * \frac{\partial L(y, \hat{y})}{\partial z^{(k+1)}} \\ &= \frac{\partial n^{(k)}}{\partial z^{(k)}} * \frac{\partial z^{(k+1)}}{\partial n^{(k)}} * \delta^{(k+1)} \\ &= f'_k(z^{(k)}) * ((W^{(k+1)})^T * \delta^{(k+1)}) \end{aligned}$$

由上式我们可以看到，第 k 层神经元的误差项

$\delta^{(k)}$ 是由第 $k + 1$ 层的误差项乘以第 $k + 1$ 层的权重，再乘以第 k 层激活函数的导数（梯度）得到的。

这就是误差的反向传播。

现在我们已经计算出了偏导数，则分别表示为：

$$\frac{\partial L(y, \hat{y})}{\partial W^{(k)}} = \frac{\partial L(y, \hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial W^{(k)}} = \delta^{(k)} * (n^{(k-1)})^T$$

$$\frac{\partial L(y, \hat{y})}{\partial b^{(k)}} = \frac{\partial L(y, \hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial b^{(k)}} = \delta^{(k)}$$


说实话，看到这我有点蒙了，上面的推导不太能看懂，涉及到迭代。。。。。

假设每一层网络激活后的输出为

$f_i(x)$ ，其中 i 为第 i 层， x 代表第 i 层的输入，也就是第 $i-1$ 层的输出， f 是激活函数，那么得出：

$$f_{i+1} = f(f_i * w_{i+1} + b_{i+1}), \text{ 记为 } f_{i+1} = f(f_i * w_{i+1}).$$

$$\frac{\partial Loss}{\partial w_n} = \frac{\partial Loss}{\partial f_n} * \frac{\partial f_n}{\partial w_n} = \frac{\partial Loss}{\partial f_n} * f' * f_{n-1}$$

 Blank Equation

$$\frac{\partial Loss}{\partial f_n} = \frac{\partial Loss}{\partial f_{n+1}} * \frac{\partial f_{n+1}}{\partial f_n}$$

可以看出，这是一个类似迭代的公式，前一层的参数更新，有赖于后一层的损失。

下面这种推导可能更好理解一些：

BP 神经网络分为两个过程

1. 工作信号正向传递子过程

2. 误差信号逆向传递过程

在一般的 BP 神经网络中，单个样本有 m 个输入和 n 个输出，在输入层和输出层之间还有若干个隐藏层，实际上 1989 年时就已经有人证明了一个万能逼近定理：

所以说一个三层的神经网络就可以实现一个任意从 m 维到 n 维的一个映射。这三层分别是 输入层、隐藏层、输出层。在 BP 神经网络中，输入层和输出层的节点数目都是固定的，关键的就是在于隐藏层数目的选择，隐藏层数目的选择决定了神经网络工作的效果，一般而言，有一个关于隐藏层数目的经验公式

$$h = \sqrt{m + n} + a$$

其中 h 为隐藏层节点数目， m 为输入层节点数目， n 为输出层节点数目 a 为 1-10 之间的调节常数。一般而言如果数据多的话我们可以设 a 稍微大一点，而数据不是太多的时候就设置的小一点防止过拟合。

正向传递过程：

设节点 i 与 节点 j 之间的权值为

$w_{i,j}$ ，每个节点的输出值为 y_j ，具体的计算方法如下：

$$S_j = \sum_{i=0}^{m-1} w_{i,j} x_i + b_j$$

$$y_j = f(S_j)$$

其中 f 为激活函数，一般选择 sigmoid 函数或者线性函数。

BP 神经网络的关键之处就在于反向误差的传播

假设我的第 j 个输出结果为

d_j 则误差函数如下所示：

$$E(w, b) = \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2$$

BP 神经网络的墓地就是通过不断修改 w 值和 b 值使得误差达到最小（一旦 w b 值全部确定以后，一个输入就对应着一个输出，所以只要让误差最小就可以了）而调整误差的方法就是使用梯度下降法不断减小误差。

我们选择激励函数为

$$f(x) = \frac{A}{1 + e^{-\frac{x}{B}}}$$

对于介于隐藏层与输出层之间的权值 $w_{i,j}$ 由偏微分公式我们可以得到

$$\Delta w(i, j) = -\eta \frac{\partial E(w, b)}{\partial w(i, j)}$$

对于激励函数求导（为了方便我们得到整体的导数）我们可以得到

$$f'(x) = \frac{f(x)[A - f(x)]}{AB}$$

然后对于

$w_{i,j}$ 的偏导数我们也可以求出

$$\begin{aligned}
\frac{\partial E(w, b)}{\partial w_{ij}} &= \frac{1}{\partial w_{ij}} \cdot \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2 \\
&= (d_j - y_j) \cdot \frac{\partial d_j}{\partial w_{ij}} \\
&= (d_j - y_j) \cdot f'(S_j) \cdot \frac{\partial S_j}{\partial w_{ij}} \\
&= (d_j - y_j) \cdot \frac{f(S_j) [A - f(S_j)]}{AB} \cdot \frac{\partial S_j}{\partial w_{ij}} \\
&= (d_j - y_j) \cdot \frac{f(S_j) [A - f(S_j)]}{AB} \cdot x_i \\
&= \delta_{ij} \cdot x_i
\end{aligned}$$

其中：

$$\delta_{i,j} = (d_j - y_j) * \frac{f(S_j)[A - f(S_j)]}{AB}$$

对于 b_j 的导数为：

$$\frac{\partial E(w, b)}{\partial b_j} = \delta_{ij}$$

这就是著名的学习规则，通过改变神经元之间的连接权值来减少系统实际输出和期望输出的误差，这个规则又叫做 Widrow-Hoff 学习规则或者纠错学习规则。

对最后一层处理完毕之后我们开始对前一层进行处理，首先将误差通过权值向前传递得到上一层的误差，那么同样的对于上一层使用梯度下降法最小化误差就可以了：

$$\begin{aligned}
\frac{\partial E(w, b)}{\partial w_{ki}} &= \frac{1}{\partial w_{ki}} \cdot \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2 \\
&= \sum_{j=0}^{n-1} (d_j - y_j) \cdot f'(S_j) \cdot \frac{\partial S_j}{\partial w_{ki}} \\
&= \sum_{j=0}^{n-1} (d_j - y_j) \cdot f'(S_j) \cdot \frac{\partial S_j}{\partial x_i} \cdot \frac{\partial x_i}{\partial S_i} \cdot \frac{\partial S_i}{\partial w_{ki}} \\
&= \sum_{j=0}^{n-1} \delta_{ij} \cdot w_{ij} \cdot \frac{f(S_i) [A - f(S_i)]}{AB} \cdot x_k \\
&= x_k \cdot \sum_{j=0}^{n-1} \delta_{ij} \cdot w_{ij} \cdot \frac{f(S_i) [A - f(S_i)]}{AB} \\
&= \delta_{ki} \cdot x_k
\end{aligned}$$

<http://blog.csdn.net/zhenlong3205237>

其中

$$\delta_{k,i} = \sum_{j=0}^{n-1} \delta_{i,j} * w_{i,j} * \frac{f(S_j)[A - f(S_j)]}{AB}$$

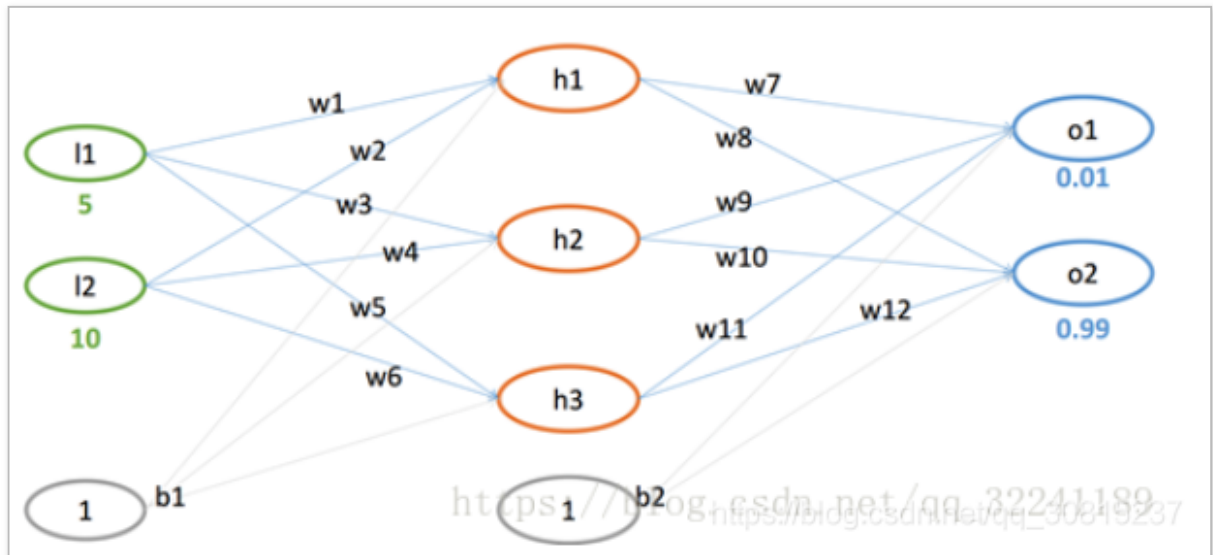
剩下的就是根据梯度下降法更新 w 与 b 的值，以使得误差最小

$$w_{i,j} = w_{i,j} - \eta_1 * \frac{\partial E(w, b)}{\partial w_{i,j}} = w_{i,j} - \eta_1 * \delta_{i,j} * x_i$$

$$b_j = b_j - \eta_2 * \frac{\partial E(w, b)}{\partial b_j} = b_j - \eta_2 * \delta_{i,j}$$

BP 算法的示例

已知如图所示的网络结构.



设初始权重值 w 和偏置项 b 为:

$$w = (0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65)$$

$b = (0.35, 0.65)$ --> 这里为了便于计算, 假设从输入层到隐层之间, 隐层到输出层之间的偏置项 b 恒定.

1) FP 计算过程

① 从输入层到隐层 (其实这里的 b 可以看成是一个特征)

$$h1 = w1 * l1 + w2 * l2 + b1 * 1 = 0.1 * 5 + 0.15 * 10 + 0.35 * 1 = 2.35$$

$$h2 = w3 * l1 + w4 * l2 + b1 * 1 = 0.2 * 5 + 0.25 * 10 + 0.35 * 1 = 3.85$$

$$h3 = w5 * l1 + w6 * l2 + b1 * 1 = 0.3 * 5 + 0.35 * 10 + 0.35 * 1 = 5.35$$

则各个回归值经过激活函数变换后的值为:

$$out_{h1} = \frac{1}{1 + e^{-h1}} = \frac{1}{1 + e^{-2.35}} = 0.9129342$$

$$out_{h2} = \frac{1}{1 + e^{-h2}} = \frac{1}{1 + e^{-3.85}} = 0.9791637$$

$$out_{h3} = \frac{1}{1 + e^{-h3}} = \frac{1}{1 + e^{-5.35}} = 0.9952743$$

② 隐层到输出层

$$net_o1 = outh1*w7 + outh2*w9 + outh3*w11 + b2*1 = 2.35*0.4 + 3.85*0.5 + 5.35*0.6 + 0.65 = 2.10192$$

$$net_o2 = outh1*w8 + outh2*w10 + outh3*w12 + b2*1 = 2.35*0.45 + 3.85*0.55 + 5.35*0.65 + 0.65 = 2.24629$$

则经过激活函数变换得到:

$$out_o1 = 0.89109 \text{ (真实值 } 0.01), out_o2 = 0.90433 \text{ (真实值 } 0.99)$$

此时的平方和误差为:

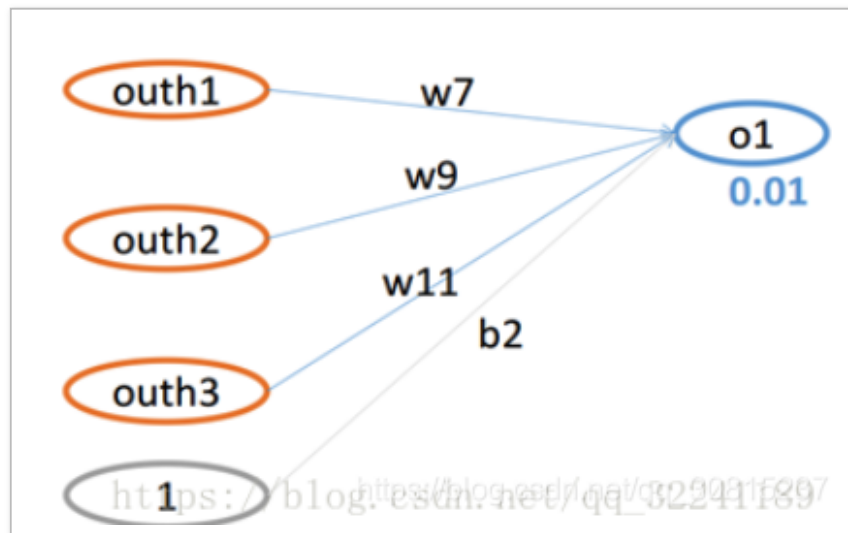
$$E_{total} = E_{o1} + E_{o2} = \frac{1}{2}(0.01 - 0.89109)^2 + \frac{1}{2}(0.99 - 0.90433)^2 = 0.1391829$$

与真实值不符, 需要进行 BP 反馈计算.

2) BP 计算过程

这里的 BP 计算我们分为两个部分. ①隐层到输出层的参数 W 的更新 ②从输入层到隐层的参数 W 的更新.

在这里, 我们主要讲述第一部分隐层到输出层的参数 W 的更新.



首先, 运用梯度下降法求解 W7 的值.

目标函数:

 , 由于此时求解 W7, 所以只与  有关, 则此时的损失函数为: 

可以看出,

 为凸函数 (开口向上), 有最小值且最小值在导数为 0 的点上.

又有 $net_o1 = outh1*w7 + outh2*w9 + outh3*w11 + b2*1$,

$outo1 = f(net_o1)$

则对 W7 求偏导数得到:

$$\begin{aligned} \frac{\partial E_{o1}}{\partial w_7} &= \frac{\partial E_{o1}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial net_{o1}} \cdot \frac{\partial net_{o1}}{\partial w_7} \\ &= \frac{1}{2} \times 2 \times (target_{o1} - out_{o1}) \cdot f'(net_{o1}) \cdot outh1 \end{aligned}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_7} = \frac{1}{2} * 2 * (target_{o1} - out_{o1}) * out_{o1} * (1 - out_{o1}) out_{h1}$$

, 写错了吧

又有 W 的更新公式为:



这样将上式带入 W 的更新公式 (5) 就可以求得更新后的 W 值. 最后, 得到了 $W7$ 的更新值为:



同理可以求得 $W9, W11, b2$ 的更新值, 这里就不在一一叙述了.

代码实现:

```
from numpy import *
class CBpnet:
#第一阶段 数据预处理阶段
    #构造函数, 输入训练数据trainx, 输出数据y
    def __init__(self, trainx, trainy):
        self.hidenum=2
        self.error=1;
        self.e=0;
        self.learningrata=0.9#默认学习率为0.9
        self.trainy=self.__normalize__(trainy)#训练输出数据归一化
        self.data1=self.__normalize__(trainx)#训练输入数据归一化
        mx,nx=shape(trainx)

        #方案1 用0作为初始化参数 测试
        self.weight1=zeros((self.hidenum,mx));#默认隐藏层有self.hidenum个神经元, 输入层与隐藏层的链接权值
        self.b1=zeros((self.hidenum,1));
        my,ny=shape(trainy)
        self.weight2=zeros((my,self.hidenum))#隐藏层与输出层的链接权值
        self.b2=zeros((my,1));
        #方案2、采用随机初始化为-1~1之间的数 测试
        self.weight1=2*random.random((self.hidenum,mx))-1
        self.weight2=2*random.random((my,self.hidenum))-1#隐藏层与输出层的链接权值
#训练数据归一化至0~1
    def __normalize__(self, trainx):
        minx,maxx=self.__MaxMin__(trainx)
        return (trainx-minx)/(maxx-minx)
    def __MaxMin__(self, trainX):
        n,m=shape(trainX)
        minx=zeros((n,1))
        maxx=zeros((n,1))
        for i in range(n):
            minx[i,0]=trainX[i,:].min();
            maxx[i,0]=trainX[i,:].max();
        return minx,maxx
```

#第二阶段 数据训练阶段

```
def Traindata(self):
    mx,nx=shape(self.data1)
    #随机梯度下降法
    for i in range(mx):
        #第一步、前向传播
        #隐藏层
        outdata2=self.__ForwardPropagation__(self.data1[:,i],self.weight1,self.b1)#隐藏层节点数值
        #输出层 outdatatemp为隐藏层数值
        outdata3=self.__ForwardPropagation__(outdata2,self.weight2,self.b2)

        self.e=self.e+(outdata3-self.trainy[:,i]).transpose()*(outdata3-self.trainy[:,i])
        self.error=self.e/2.
        #计算每一层的残差
        sigma3=(1-outdata3).transpose()*outdata3*(outdata3-self.trainy[:,i])
        sigma2=((1-outdata2).transpose()*outdata2)[0,0]*(self.weight2.transpose()*sigma3)
        #计算每一层的偏导数
        w_derivative2=sigma3*outdata2.transpose()
        b_derivative2=sigma3

        w_derivative1=sigma2*self.data1[:,i].transpose()
        b_derivative1=sigma2
        #梯度下降公式
        self.weight2=self.weight2-self.learningrata*w_derivative2
        self.b2=self.b2-self.learningrata*b_derivative2

        self.weight1=self.weight1-self.learningrata*w_derivative1
        self.b1=self.b1-self.learningrata*b_derivative1

def __ForwardPropagation__(self,indata,weight,b):
    outdata=weight*indata+b
    outdata=1./(1+exp(-outdata))
    return outdata
```

注意：代码中使用的激活函数为 $f(x) = \frac{1}{1 + e^{-x}}$ ，即上述公式推导中的A，B 全为 1 的情况。

from: <https://www.cnblogs.com/zk71124720/p/8551811.html>

from : <https://www.jianshu.com/p/97795a193272>

from: <https://blog.csdn.net/hjimce/article/details/45457181>

from: <https://blog.csdn.net/zhelong3205/article/details/78688476>

from: https://blog.csdn.net/qq_32241189/article/details/80305566

