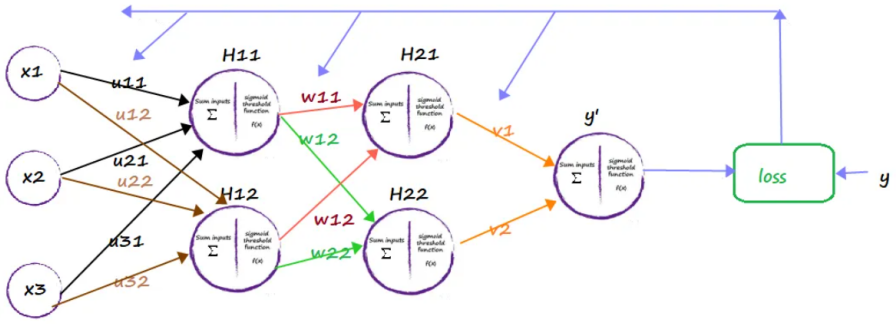


BP神经网络

远小数
专注于人工智能、数学建模的知识分享

关注他

9 人赞同了该文章

BP(Back Propagation) 算法是神经网络深度学习中最重要的算法之一，了解BP算法可以让我们更理解神经网络深度学习模型训练的本质，属于内功修行的部分。

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，是应用最广泛的神经网络模型之一 。Minsky和Papert在颇具影响力的"*perceptron*" 一书中指出，简单的感知器只能求解线性问题，能够求解非线性问题的网络应该具有感知层，但是对隐藏层神经元的学习规则还没有合理的理论依据。

从前面介绍的感知器学习规则来看，其权值的调整取决于期望输出与实际输出之差：

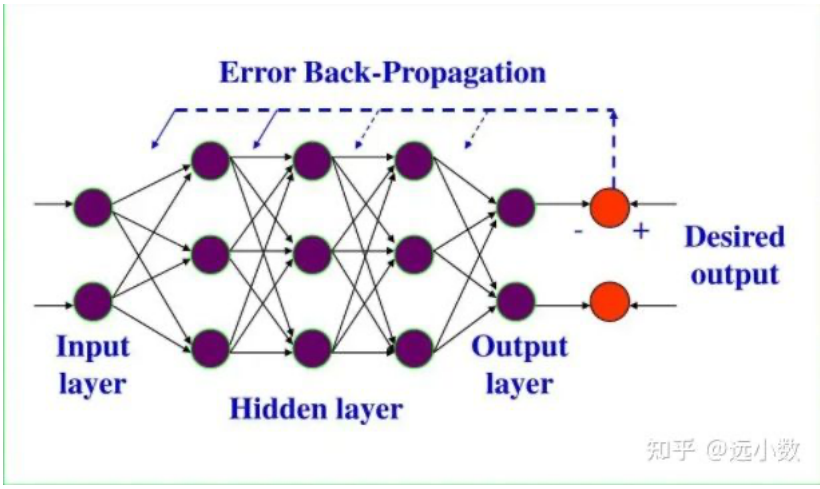
$$\Delta w_i = \eta(t-y)x_i$$

但是对于各个隐藏层的节点来说，不存在已知的期望输出，因而该学习规则不能用于隐藏层的权值调整。

BP算法的基本思想是：学习过程由信号的正向传播和误差的反向传播两个过程组成。

正向传播时，把样本的特征从输入层进行输入，信号经过各个隐藏层的处理后，最后从输出层传出。对于网络的实际的输出与期望输出之间的误差，把误差信号从最后一层逐层反传，从而获得各个层的误差学习信号，然后再根据误差学习信号来修正各层神经元的权值。

这种信号正向传播与误差反向传播，然后各层调整权值的过程是周而复始地进行的。权值不断调整的过程，也就是网络学习训练的过程。进行此过程直到网络输出误差减小到预先设置的阈值以下，或者超过预先设置的最大训练次数。



代价函数

代价函数也称为损失函数(Loss Function 或 Cost Function)。

代价函数并没有准确的定义，一般我们可以理解为是一个人为定义的函数，我们可以利用这个函数来优化模型的参数。最简单且常见的一个代价函数是均方差(MSE)代价函数，也称为二次代价函数：

$$E = \frac{1}{2N}(T-N)^2 = \frac{1}{2N}\sum_{i=1}^{N}(t_i - y_i)^2$$

矩阵可以用大写字母来表示，这里的T表示真实标签，Y表示网格输出，i表示第i个数据。N表示训练样本的个数（注意，这里的N是一个大于0的整数，不是矩阵）

$T-Y$可以得到每个训练样本与真实标签的误差。误差的值有正有负，我们可以求平方，把所有的误差都变成正的，然后除以$2N$。这里的2没有特别的含义，主要是我们对均方差代价函数求导的时候，式子中的2次方的2可以跟分母中的2约掉，使得公式推导看起来更整齐简洁。除以N表示求每个样本的误差平均的平均值。

公式可以用矩阵形式来表达，也可以拆分为用$\sum$来累加各个训练样本的真实标签与网络输出的误差的平方。

梯度下降法

在求解机器学习算法的模型参数，即无约束优化问题时，梯度下降（Gradient Descent）是最常采用的方法之一，另一种常用的方法是最小二乘法。这里就对梯度下降法做一个完整的总结。

梯度

在微积分里面，对多元函数的参数求∂偏导数，把求得的各个参数的偏导数以向量的形式写出来，就是梯度。比如函数$f(x,y)$, 分别对x,y求偏导数，求得的梯度向量就是$(\partial f/\partial x, \partial f/\partial y)^T$, 简称$grad f(x,y)$或者$\nabla f(x,y)$。对于在点$(x_0,y_0)$的具体梯度向量就是$(\partial f/\partial x_0, \partial f/\partial y_0)^T$. 或者$\nabla f(x_0,y_0)$，如果是3个参数的向量梯度，就是$(\partial f/\partial x, \partial f/\partial y, \partial f/\partial z)^T$,以此类推。

那么这个梯度向量求出来有什么意义呢？他的意义从几何意义上讲，就是函数变化增加最快的地方。具体来说，对于函数$f(x,y)$,在点(x_0,y_0)，沿着梯度向量的方向就是$(\partial f/\partial x_0, \partial f/\partial y_0)^T$的方向是$f(x,y)$增加最快的地方。或者说，沿着梯度向量的方向，更加容易找到函数的最大值。反过来说，沿着梯度向量相反的方向，也就是$-(\partial f/\partial x_0, \partial f/\partial y_0)^T$的方向，梯度减少最快，也就是更加容易找到函数的最小值。

梯度下降和梯度上升

在机器学习算法中，在最小化损失函数时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数，和模型参数值。反过来，如果我们需要求解损失函数的最大值，这时就需要用梯度上升法来迭代了。

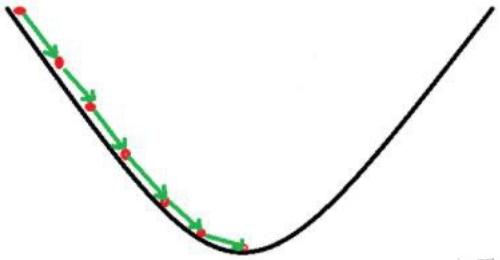
梯度下降法和梯度上升法是可以互相转化的。比如我们需要求解损失函数$f(\theta)$的最小值，这时我们需要用梯度下降法来迭代求解。但是实际上，我们可以反过来求解损失函数 $-f(\theta)$的最大值，这时梯度上升法就派上用场了。

下面来详细总结下梯度下降法。

梯度下降法

首先来看看梯度下降的一个直观的解释。比如我们在一座大山上的某处位置，由于我们不知道怎么下山，于是决定走一步算一步，也就是在每走到一个位置的时候，求解当前位置的梯度，沿着梯度的负方向，也就是当前最陡峭的位置向下走一步，然后继续求解当前位置梯度，向这一步所在位置沿着最陡峭最易下山的位置走一步。这样一步步的走下去，一直走到觉得我们已经到了山脚。当然这样走下去，有可能我们不能走到山脚，而是到了某一个局部的山峰低处。

从上面的解释可以看出，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。



在详细了解梯度下降的算法之前，我们先看看相关的一些概念。

1. 步长（Learning rate）：步长决定了在梯度下降迭代的过程中，每一步沿梯度负方向前进的长度。用上面下山的例子，步长就是在当前这一步所在位置沿着最陡峭最易下山的位置走的那一步的长度。

2.特征（feature）：指的是样本中输入部分，比如2个单特征的样本（$x(0), y(0)$），（$x(1), y(1)$）,则第一个样本特征为$x(0)$，第一个样本输出为$y(0)$。

3. 假设函数（hypothesis function）：在监督学习中，为了拟合输入样本，而使用的假设函数，记为$h_\theta(x)$。比如对于单个特征的m个样本（$x(i), y(i)$）$(i = 1, 2, \dots m)$,可以采用拟合函数如下：$h_\theta(x) = \theta_0 + \theta_1 x$。

4. 损失函数（loss function）：为了评估模型拟合的好坏，通常用损失函数来度量拟合的程度。损失函数极小化，意味着拟合程度最好，对应的模型参数即为最优参数。在线性回归中，损失函数通常为样本输出和假设函数的差取平方。比如对于m个样本（xi, yi）$(i = 1, 2, \dots m)$,采用线性回归，损失函数为：

$$J(\theta_0, \theta_1) = \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

其中x_i表示第i个样本特征，y_i表示第i个样本对应的输出，$h_\theta(x_i)$为假设函数。

梯度下降的详细算法

1. 先决条件： 确认优化模型的假设函数和损失函数。

比如对于线性回归，假设函数表示为

$$h_\theta(x_1,x_2,...x_n)=\theta_0+\theta_1x_1+...+\theta_nx_n,$$

其中$\theta_i(i=0,1,2...n)$为模型参数，$x_i(i=0,1,2...n)$为每个样本的n个特征值。这个表示可以简化，我们增加一个特征$x_0=1$，这样

$$h_\theta(x_0,x_1,...x_n)=\sum_{i=0}^{n}\theta_ix_i$$

。

同样是线性回归，对应于上面的假设函数，损失函数为：

$$J(\theta_0,\theta_1...,\theta_n)=\frac{1}{2m}\sum_{j=1}^{m}(h_\theta(x_0^{(j)},x_1^{(j)},...x_n^{(j)})-y_j)^2$$

1. 算法相关参数初始化：主要是初始化$\theta_0,\theta_1...,\theta_n$,算法终止距离$\varepsilon$以及步长$\alpha$。在没有任何先验知识的时候，我喜欢将所有的$\theta$初始化为$0$， 将步长初始化为1。在调优的时候再优化。

3.算法过程：

1）确定当前位置的损失函数的梯度，对于θ_i,其梯度表达式如下：

$$\frac{\partial}{\partial\theta_i}J(\theta_0,\theta_1...,\theta_n)$$

2）用步长乘以损失函数的梯度，得到当前位置下降的距离，即$\alpha\frac{\partial}{\partial\theta_i}J(\theta_0,\theta_1...,\theta_n)$对应于前面登山例子中的某一步。

3）确定是否所有的θ_i,梯度下降的距离都小于ε，如果小于ε则算法终止，当前所有的$\theta_i(i=0,1,...n)$即为最终结果。否则进入步骤4.

4）更新所有的θ，对于θ_i，其更新表达式如下。更新完毕后继续转入步骤1.

$$\theta_i=\theta_i-\alpha\frac{\partial}{\partial\theta_i}J(\theta_0,\theta_1...,\theta_n)$$

梯度下降的算法调优

在使用梯度下降时，需要进行调优。哪些地方需要调优呢?

1. 算法的步长选择。在前面的算法描述中，我提到取步长为1，但是实际上取值取决于数据样本，可以多取一些值，从大到小，分别运行算法，看看迭代效果，如果损失函数在变小，说明取值有效，否则要增大步长。前面说了。步长太大，会导致迭代过快，甚至有可能错过最优解。步长太小，迭代速度太慢，很长时间算法都不能结束。所以算法的步长需要多次运行后才能得到一个较为优的值。
2. 算法参数的初始值选择。 初始值不同，获得的最小值也有可能不同，因此梯度下降求得的只是局部最小值；当然如果损失函数是凸函数则一定是最优解。由于有局部最优解的风险，需要多次用不同初始值运行算法，关键损失函数的最小值，选择损失函数最小化的初值。

3.归一化。由于样本不同特征的取值范围不一样，可能导致迭代很慢，为了减少特征取值的影响，可以对特征数据归一化，也就是对于每个特征x，求出它的期望$\overline{x}$ 和标准差$std(x)$，然后转化为：

$$\frac{x-\overline{x}}{std(x)}$$

这样特征的新期望为0，新方差为1，迭代速度可以大大加快。

梯度下降三大方法

1.批量梯度下降法（Batch Gradient Descent） 批量梯度下降法，是梯度下降法最常用的形式，具体做法也就是在更新参数时使用所有的样本来进行更新，这个方法对应于前面的线性回归的梯度下降算法，也就是说梯度下降详细算法就是批量梯度下降法。

$$\theta_j = \theta_i - \alpha \sum_{j=1}^{m} (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots x_n^{(j)}) - y_j) x_i^{(j)}$$

由于我们有m个样本，这里求梯度的时候就用了所有m个样本的梯度数据。

2.随机梯度下降法（Stochastic Gradient Descent)

随机梯度下降法，其实和批量梯度下降法原理类似，区别在与求梯度时没有用所有的m个样本的数据，而是仅仅选取一个样本j来求梯度。对应的更新公式是：

$$\theta_j = \theta_i - \alpha (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots x_n^{(j)}) - y_j) x_i^{(j)}$$

随机梯度下降法，和前面的批量梯度下降法是两个极端，一个采用所有数据来梯度下降，一个用一个样本来梯度下降。自然各自的优缺点都非常突出。对于训练速度来说，随机梯度下降法由于每次仅仅采用一个样本来迭代，训练速度很快，而批量梯度下降法在样本量很大的时候，训练速度不能让人满意。对于准确度来说，随机梯度下降法用于仅仅用一个样本决定梯度方向，导致解很有可能不是最优。对于收敛速度来说，由于随机梯度下降法一次迭代一个样本，导致迭代方向变化很大，不能很快的收敛到局部最优解。

那么，有没有一个中庸的办法能够结合两种方法的优点呢？有！这就是下面的小批量梯度下降法。

3.小批量梯度下降法（Mini-batch Gradient Descent）

小批量梯度下降法是批量梯度下降法和随机梯度下降法的折中，也就是对于m个样本，我们采用x个来迭代，$1

$$\theta_j = \theta_i - \alpha \sum_{j=t}^{t+x-1} (h_\theta(x_0^{(j)}, x_1^{(j)}, \dots x_n^{(j)}) - y_j) x_i^{(j)}$$

梯度下降算法和其他无约束算法的比较

在机器学习中的无约束优化算法，除了梯度下降以外，还有前面提到的最小二乘法，此外还有牛顿法和拟牛顿法。

梯度下降法和最小二乘法相比，梯度下降法需要选择步长，而最小二乘法不需要。梯度下降法是迭代求解，最小二乘法是计算解析解。如果样本量不算很大，且存在解析解，最小二乘法比起梯度下降法要有优势，计算速度很快。但是如果样本量很大，用最小二乘法由于需要求一个超级大的逆矩阵，这时就很难或者很慢才能求解解析解了，使用迭代的梯度下降法比较有优势。

梯度下降法和牛顿法/拟牛顿法相比，两者都是迭代求解，不过梯度下降法是梯度求解，而牛顿法/拟牛顿法是用二阶的海森矩阵的逆矩阵或伪逆矩阵求解。相对而言，使用牛顿法/拟牛顿法收敛更快。但是每次迭代的时间比梯度下降法长。

Delta学习规则

Delata学习规则是一种利用梯度下降法的一般性学习规则，其实就是利用梯度下降法来最小化代价函数。 例如我们所介绍的均方差代价函数，一个样本的均方差代价函数公式定义如下：

$$E = \frac{1}{2}(T-Y)^2 = \frac{1}{2}(t-y)^2 = \frac{1}{2}(t-f(WX))^2$$

误差E是W的函数，我们可以使用梯度下降法来最小化E的值，权值矩阵的变化ΔW等于负的学习率（$-\eta$）乘以E对W求导：

$$\Delta W = -\eta E' = \eta X^T(t-y)f'(WX) = \eta X^T\delta$$

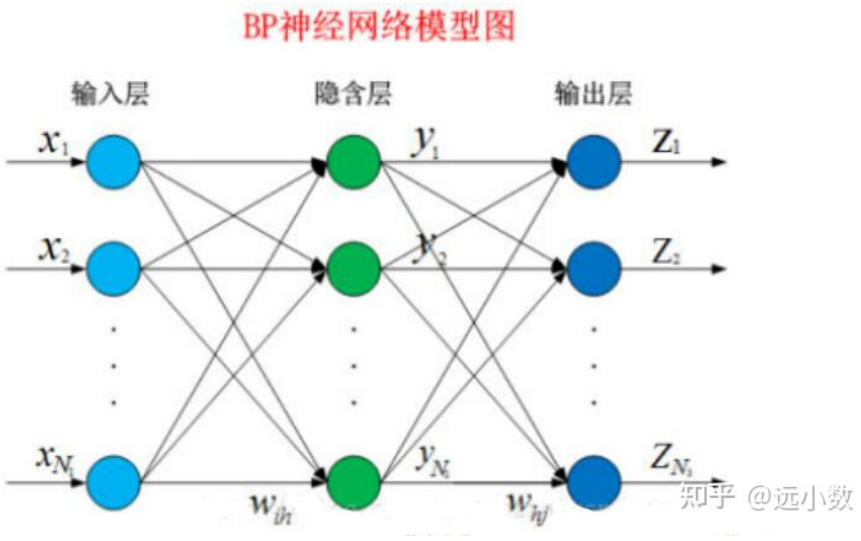
注意，这里的X和W都是矩阵，所以这里求导的时候是对矩阵W进行求导，矩阵求导的方式和单元素求导的方式有一些不同。对单个w元素的权值变化计算：

$$\Delta w_i = -\eta E' = \eta x_i(t-y)f'(WX) = \eta x_i\delta$$

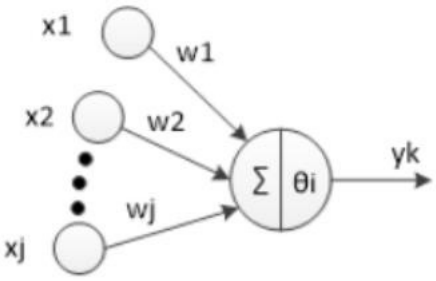
BP神经网络模型

BP神经网络学习算法可以说是目前最成功的神经网络学习算法。显示任务中使用神经网络时，大多数是使用BP算法进行训练. 在我看来BP神经网络就是一个”万能的模型+误差修正函数“，每次根据训练得到的结果与预想结果进行误差分析，进而修改权值和阈值，一步一步得到能输出和预想结果一致的模型。举一个例子：比如某厂商生产一种产品，投放到市场之后得到了消费者的反馈，根据消费者的反馈，厂商对产品进一步升级，优化，从而生产出让消费者更满意的产品。这就是BP神经网络的核心。

下面就让我们来看看BP算法到底是什么东西。BP网络由输入层、隐藏层、输出层组成。 给定训练集$D = (x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$,其中$x_n \epsilon R^d$， $y_n \epsilon R^l$,表示输入示例由d个属性组成，输出l维实值变量。现在，我们看看如何求得输出值，以及怎么由输出值调整权值和阈值。



神经元是以生物研究及大脑的响应机制而建立的拓扑结构网络，模拟神经冲突的过程，多个树突的末端接受外部信号，并传输给神经元处理融合，最后通过轴突将神经传给其它神经元或者效应器。神经元的拓扑结构如图：



对于第i个神经元，x_1、x_2、$\ldots$、x_j为神经元的输入，输入常为对系统模型关键影响的自变量，w_1、w_2、$\ldots$、w_j为连接权值调节各个输入量的占重比。将信号结合输入到神经元有多种方式，选取最便捷的线性加权求和可得$neti$神经元净输入:

$$Net_{in} = \sum_{i=1}^{n} w_i * x_i$$

θ_i表示该神经元的阈值，根据生物学中的知识，只有当神经元接收到的信息达到阈值是才会被激活。因此，我们将Net_{in}和θ_j进行比较，然后通过激活函数处理以产生神经元的输出。 激活函数：激活函数这里我们不多重述。如果输出值有一定的范围约束，比如用来分类，一般我们用的最多的是$Sigmod$函数，它可以把输入从负无穷大到正无穷大的信号变换成0到1之间输出。如果没有约束的话，我们可以使用线性激活函数(即权值相乘之和)。这样我们得到的输出为:

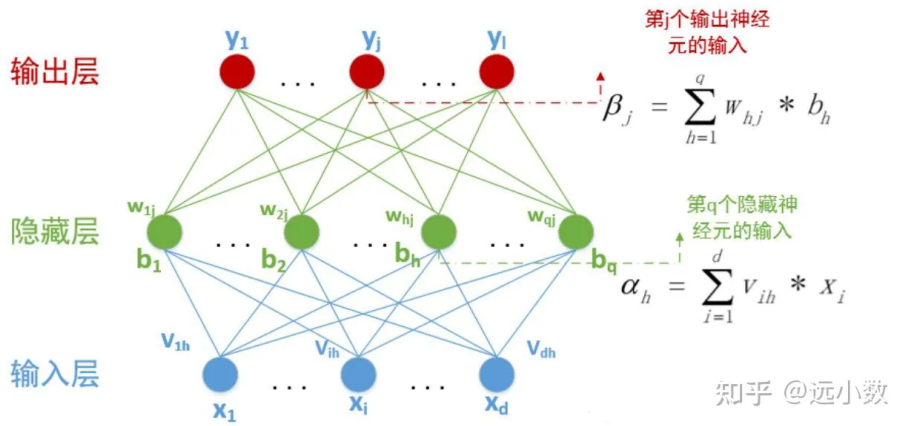
$$y_j = f(Net_{in} - \theta_j)$$

我们可以将公式化简一下，设第一个输入值永远为θ,权值为-1，则我们可以得到公式:

$$y_j = f(\sum_{i=0}^{n} w_i * x_i)$$

其中$w_0 = -1, x_0 = \theta_j$,其中f为选择的激活函数。

已经知道在BP神经网络模型中，我们有三层结构，输入层、隐藏层、输出层,因此输入层到隐藏层的权值，设为v_{ih},隐藏层第h个神经元的阈值我们设为γ_h。隐藏层到输出层的权值，设为w_{hj},输出层第j个神经元的阈值我们用θ_j表示。在下面这张图里，有d输入神经元,q个隐藏神经元，隐藏有q个隐藏神经元阈值，l个输出神经元，因此有l个输出神经元阈值。



其中β_j中的$b_h = f(\alpha_h - \gamma_h)$。隐藏层和输出层的激活函数，在这里我们暂时全部用$Sigmod$函数。

在某个训练示例(x_k, y_k)中，假设神经网络的训练输出为$y_k = (y_1^k, y_2^k, \cdots, y_l^k)$,输出为$l$维向

量，其中

$$y_i^{k'} = f(\beta_i - \theta_i)$$

那么这次预测结果的误差我们可以用最小二乘法表示：

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

而我们现在要做的就是根据这个误差去调整（$d+l+1$）$q+l$个参数的值，一步一步缩小E_k。那么从现在开始，我们就要进入数学的世界了。这里我们使用最常用的算法：梯度下降法来更新参数。函数永远是沿着梯度的方向变化最快，那么我们对每一个需要调整的参数求偏导数，如果偏导数>0,则要按照偏导数相反的方向变化；如果偏导数<0，则按照此方向变化即可。于是我们使用-1*偏导数则可以得到参数需要变化的值。同时我们设定一个学习速率η，这个学习速率不能太快，也不能太慢。太快可能会导致越过最优解；太慢可能会降低算法的效率。(具体设多少就属于玄学调参的领域了)。因此我们可以得到一个参数调整公式：

$$Param+ = -\eta\frac{\partial E_k}{\partial Param}$$

首先我们看看隐藏层到输出层的权值调整值：

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

综合我们可得：

$$\Delta w_{hj} = -\eta\frac{\partial E_k}{\partial w_{hj}} = -\eta g_i b_h = -\eta(y_j^{k'} - y_j^k)\cdot y_j^{k'}\cdot(1-y_j^{k'})\cdot b_h\cdots$$

同理：

$$\Delta\theta_j = -\eta\frac{\partial E_k}{\partial\theta_j} = -\eta\frac{\partial E_k}{\partial y_j^{k'}}\cdot\frac{\partial y_j^{k'}}{\partial\theta_j} = \eta\cdot g_j\cdots$$

我们再看看Δv_{ih}的值怎么求,一个v权值会影响所有的β,

$$\Delta v_{ih} = -\eta e_h x_i\cdots\cdots\cdots\cdots$$

$$\Delta\gamma_h = \eta e_h\cdots\cdots\cdots$$

其中

$$e_h = (\sum_{j=1}^{l}\frac{\partial E_k}{\partial\beta_j}\cdot\frac{\partial\beta_j}{\partial b_j})\cdot f'(\alpha_h-\gamma_h) = (\sum_{j=1}^{l}(y_j^{k'}-y_j^k)\cdot f'(\beta_j-\theta_j)\cdot w_{hj})\cdot f'(\alpha_h-\gamma_h)\cdots\cdots\cdots$$

至此，我们所有得公式都推导完毕了，剩下做的就是设定一个迭代终止条件，可以是误差小于一定值时终止递归，也可以是设定迭代次数。这样一个BP神经网络模型就算是设计结束。

BP算法推导总结

我们最后推导得到的结论也就是权值调整的公式为：

$$\Delta W^h = \eta (Y^{h-1})^T \delta^h$$

这里的ΔW^h表示第h层权值矩阵W的变化，η表示学习率，Y^{h-1}表示网络第$h-1$层的输出，δ^h表示第h层的学习信号。

η是人为设置的超参数：只要把数据传入网络中就可以计算出Y^{h-1}网络第$h-1$层的输出，所以这里要重点关注的是第h层的学习信号δ^h。学习信号有两个不同的公式，输出层的学习信号公式为：

$$\delta^{h+1} = (T - Y^{h+1}) f'(Y^h W^{h+1})$$

这里的δ^{h+1}表示输出层的学习信号；T表示数据的标签值；Y^{h+1}表示模型的预测值；f'表示激活函数的导数；$Y^h W^{h+1}$表示输出层信号的汇总。

T是已知的数据标签；Y^{h+1}可以通过传入数据计算得到；激活函确定以后，f'也是已知的；Y^h可以通过传入数据计算得到；W^{h+1}在网络进行随机初始化以后也确定下来了。所以这个公式里面的所有值都是已知的，或者可以通过计算得到。把δ^{h+1}计算出来以后，代入式中就可以计算出输出层的权值矩阵要如何调整。

除输出层外，剩下的网络层的学习信号的公式都是：

$$\delta^h = \delta^{h+1} (W^{h+1})^T f'(Y^{h-1} W^h)$$

从式中我们可以看到，第h层的学习信号δ^h跟它下一层$h+1$层的学习信号δ^{h+1}有关系，并且其还跟它的下一层$h+1$层的权值矩阵的转置$(W^{h+1})^T$以及跟$f'(Y^{h-1} W^h)$有关。

所以我们在使用BP算法的时候，需要先根据网络预测的误差计算最后一层的学习信号，其次再计算倒数第二层的学习信号，最后再计算倒数第三层的学习信号，以此类推，从后向前计算。因此BP算法叫做误差反向传播算法。计算得到每一层的学习信号以后，再根据权值调整的公式来计算每一层的权值矩阵如何调整，最后对所有层的权值矩阵进行更新。

使用BP神经网络求解异或问题

```
import numpy as np
import matplotlib.pyplot as plt
# 输入数据
X = np.array([[0,0],[0,1], [1,0],[1,1]])
# 标签
T = np.array([[0],[1],[1],[0]])
# 定义一个2层的神经网络：2-10-1
# 输入层2个神经元，隐藏层10个神经元，输出层1个神经元
# 输入层到隐藏层的权值初始化，2行10列
W1 = np.random.random([2,10])
# 隐藏层到输出层的权值初始化，10行1列
W2 = np.random.random([10,1])
# 初始化偏置值，偏置值的初始化一般可以取0，或者一个比较小的常数，如0.1
# 隐藏层的10个神经元偏置
b1 = np.zeros([10])
# 输出层的1个神经元偏置
b2 = np.zeros([1])
# 学习率设置
lr = 0.1
# 定义训练周期数
epochs = 100001
# 定义测试周期数
test = 5000
# 定义sigmoid函数
def sigmoid(x):
 return 1/(1+np.exp(-x))
```

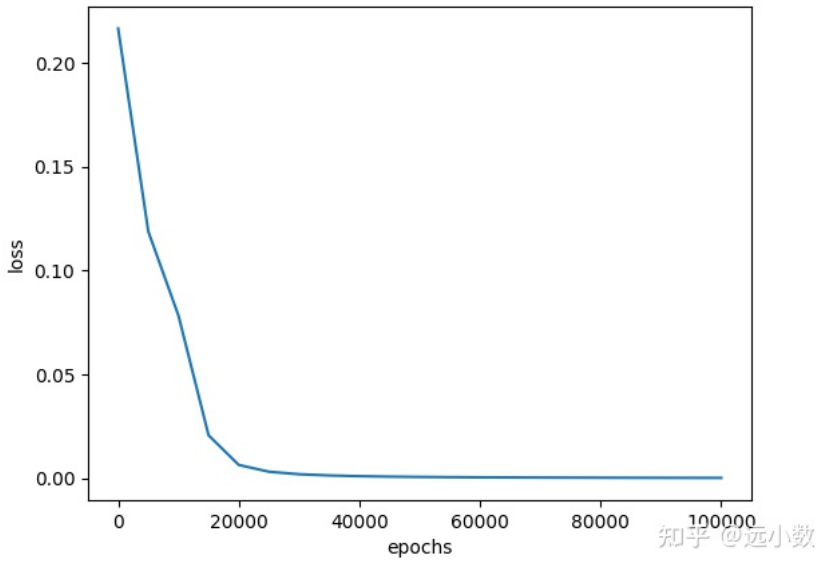
```
# 定义sigmoid函数导数
def dsigmoid(x):
 return x*(1-x)
# 更新权值和偏置值
def update():
 global X,T,W1,W2,lr,b1,b2
 # 隐藏层输出
 L1 = sigmoid(np.dot(X,W1) + b1)
 # 输出层输出
 L2 = sigmoid(np.dot(L1,W2) + b2)
 # 根据公式4.41，求输出层的学习信号
 delta_L2 = (T - L2) *dsigmoid(L2)
 # 根据公式4.42，求隐藏层的学习信号
 delta_L1 = delta_L2.dot(W2.T) *dsigmoid(L1)
 # 根据公式4.44，求隐藏层到输出层的权值改变
 # 由于一次计算了多个样本，所以需要求平均
 delta_W2 =lr*L1.T.dot(delta_L2) /X.shape[0]
# 根据公式4.45，求输入层到隐藏层的权值改变
# 由于一次计算了多个样本，所以需要求平均
 delta_W1 =lr*X.T.dot(delta_L1) / X.shape[0]
 # 更新权值
 W2 =W2 + delta_W2
 W1 =W1 + delta_W1
 # 改变偏置值
 # 由于一次计算了多个样本，所以需要求平均
 b2 =b2 +lr*np.mean(delta_L2, axis=0)
 b1 =b1 +lr*np.mean(delta_L1, axis=0)
# 定义空list用于保存loss
loss = []
# 训练模型
for i in range(epochs):
 # 更新权值
    update()
 # 每训练5000次计算一次loss值
    if i %test == 0:
 # 隐藏层输出
       L1 = sigmoid(np.dot(X,W1) + b1)
 # 输出层输出
       L2 =sigmoid(np.dot(L1,W2) + b2)
 # 计算loss值
       print('epochs:',i,'loss:',np.mean(np.square(T - L2) / 2))
 # 保存loss值
       loss.append(np.mean(np.square(T - L2) / 2))
# 画图训练周期数与loss的关系图
plt.plot(range(0,epochs,test),loss)
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()
# 隐藏层输出
L1 = sigmoid(np.dot(X,W1) + b1)
# 输出层输出
L2 = sigmoid(np.dot(L1,W2) + b2)
print('output:')
print(L2)
# 因为最终的分类只有0和1，所以我们可以把
# 大于等于0.5的值归为1类，小于0.5的值归为0类
def predict(x):
 if x>=0.5:
  return 1
 else:
  return 0
# map会根据提供的函数对指定序列做映射
# 相当于依次把L2中的值放到predict函数中计算
# 然后打印出结果
print ('predict:')
for  i in map(predict,L2):
```

```
    print(i)
```

运行结果如下：

```
epochs: 0 loss: 0.21655021948422454
epochs: 5000 loss: 0.11886693767654921
epochs: 10000 loss: 0.0784134455934812
epochs: 15000 loss: 0.020728082691587067
epochs: 20000 loss: 0.006474064948643635
epochs: 25000 loss: 0.003216904125830181
epochs: 30000 loss: 0.0020068415981549523
epochs: 35000 loss: 0.0014139696548422505
epochs: 40000 loss: 0.0010726101163828885
epochs: 45000 loss: 0.0008545799251234986
epochs: 50000 loss: 0.0007049564200818229
epochs: 55000 loss: 0.0005967480043781333
epochs: 60000 loss: 0.0005153047148555323
epochs: 65000 loss: 0.0004520544201019757
epochs: 70000 loss: 0.00040167730296948004
epochs: 75000 loss: 0.00036071199148032226
epochs: 80000 loss: 0.00032681711228389016
epochs: 85000 loss: 0.0002983566300128549
epochs: 90000 loss: 0.00027415545154439193
epochs: 95000 loss: 0.00025334936598236956
epochs: 100000 loss: 0.0002352896211791448
output:
[[0.02123791]
 [0.9775524 ]
 [0.98023771]
 [0.02316949]]
predict:
0
1
```

知乎 ··· 写文章



发布于 2022-03-23 20:50

bp神经网络 深度学习（Deep Learning） 神经网络