神经网络与反向传播算法

描述一个神经网络，最简洁清晰的方法就是看**前馈传播公式**。至于网络中各个名词与数值的命名与记号，其实也都是由**前馈公式**逆推决定的。（比如初始是0还是1；权重的层号是k还是k-1；向量是行还是列；矩阵左乘还是右乘、转不转置……）命名与记号不同，后续所有的公式也会相应的变得天花乱坠。本文根据公式最简洁的原则，设定一个明确的标准。并推导出该情况下的反向传播算法公式。

有了这个基准，在查阅不同的资料时，可以根据本文明确写出的定义和公式，相应调整记号，达成文献的“互译”。出于这个原则，本文尽量把定义写清楚，公式也同时给出向量和分量两种类型。

神经网络的结构

设样本集 $\mathcal{D}=\{(x^i,t^i)\}_{i=1}^N$, x^i 对应的输出为 y^i .

其中每个样本 $x=(x_1,\cdots,x_m)^T\in\mathbb{R}^{\mathrm{m}\times 1}$ ，对应的样本 $y=(y_1,\cdots,y_d)^T\in\mathbb{R}^{\mathrm{d}\times 1}$.

假设网络有1个**输入层** l^0 ， $N-1$ 个**隐藏层** $l^1,\cdots,l^{N-1}$ ，1个**输出层** l^N 。

用 $\{a^{(k)}=(a_1^{(k)},a_2^{(k)},\cdots)\}_{k=1}^N$ 代表网络 l^k 层的**输入**（经过激活函数前），
$\{z^{(k)}=(z_1^{(k)},z_2^{(k)},\cdots)\}_{k=0}^N$ 代表网络 l^k 层的**输出**（经过激活函数后）。

$\{W^{(k)}\}_{k=1}^N$ 表示从 l^{k-1} 层到 l^k 层的**权重**，具体而言， $W_{ij}^{(k)}$ 表示从 l^{k-1} 层的神经元 $z_j^{(k-1)}$ 到 l^k 层的神经元 $a_i^{(k)}$ 的权重。$\{b^{(k)}\}_{k=1}^N$ 表示从 l^{k-1} 层到 l^k 层的**偏置**，具体而言， $b_i^{(k)}$ 表示从 l^{k-1} 层的神经元到 l^k 层的神经元 $a_i^{(k)}$ 的偏置。

$\{h^{(k)}(\cdot)\}_{k=1}^N$ 表示 l^k 层的**激活函数**，一般而言，除输出层外其余各层的激活函数都是相等的。

前馈公式：

> - 网络输入：（输入层 l^0 的输出即为某个样本 x）
>
> $$z^{(0)}=x \tag{1}$$
>
> 分量形式：
>
> $$z_i^{(0)}=\tilde{x}_i \qquad ,\forall i$$
>
> - 层间传播：（ l^{k-1} 层的输出线性组合加上偏置即为 l^k 层的输入）
>
> $$a_i^{(k)}=\sum_j W_{ij}^{(k)}z_j^{(k-1)}+b_i^{(k)} \qquad ,\forall i$$
>
> 向量形式：
>
> $$a^{(k)}=W^{(k)}\cdot z^{(k-1)}+b^{(k)} \tag{2}$$
>
> - 层内传播（神经元的输入经过激活函数变为输出）：
>
> $$z_i^{(k)}=h^{(k)}\left(a_i^{(k)}\right) \qquad ,\forall i$$
>
> 向量形式（我们假设 $h^{(k)}(\cdot)$ 作用在向量上即相当于分别作用在其每个分量上）：
>
> $$z^{(k)}=h^{(k)}\left(a^{(k)}\right) \tag{3}$$
>
> ---
>
> - 网络输出：（ l^N 层的输出即为网络的输出 y）

> $$y = z^{(N)} \tag{4}$$
>
> 分量形式：
>
> $$y_i = z_i^{(N)} \qquad , \forall i$$

最终前馈部分的全部公式为：

$$\begin{cases} z^{(0)} = x \\ a^{(k)} = W^{(k)} \cdot z^{(k-1)} + b^{(k)} & , \forall i = 1, \cdots, N \\ z^{(k)} = h^{(k)}\left(a^{(k)}\right) & , \forall i = 1, \cdots, N \\ y = z^{(N)} \end{cases} \tag{5}$$

反向传播算法（BP）

定义模型的总损失

$$E = \sum_{i=1}^{N} l(y^i, t^i) \tag{6}$$

其中 $l(\cdot)$ 是单个样本的损失函数。

由于 y^i 由 $x^i, W^{(k)}, b^{(k)}$ 共同决定，因此 $E = E(W^{(k)}, b^{(k)}; x^i, t^i)$ 。在样本给定的情况下，改变参数 $W^{(k)}$ 和 $b^{(k)}$ 可以改变损失的大小，这就是神经网络里调参的目的。

要调参优化损失 E ，那么计算损失函数相对于参数 $W^{(k)}$ 和 $b^{(k)}$ 的梯度 $\frac{\partial E}{\partial W^{(k)}}$ 和 $\frac{\partial E}{\partial b^{(k)}}$ 就十分重要。但注意到 E 的表达式（6）中没有显式的 $W^{(k)}$ 和 $b^{(k)}$ ，而神经网络的复杂结构让写出显式的 $W^{(k)}$ 和 $b^{(k)}$ 变得几乎不可能，因此求导就变成了看起来十分困难的一个问题。

在这种多层网络复合映射的情况下，我们想到的自然且唯一的工具就是复合函数的求导准则了。

为了符号的简单清晰，我们下面考虑单个样本 $E = l(y, t)$ 的情况。因为对于多个样本的情况，总损失函数只是单个样本损失的简单加和，因此梯度也只是简单加和，对推导过程没有影响。

根据复合函数的求导准则，我们一层一层的看：

> **第N层：**
>
> $$\begin{aligned} \frac{\partial E}{\partial b^{(N)}} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial b^{(N)}} \\ &= \frac{\partial E}{\partial z^{(N)}} \cdot \frac{\partial z^{(N)}}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial b^{(N)}} \\ \frac{\partial E}{\partial W^{(N)}} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial W^{(N)}} \\ &= \frac{\partial E}{\partial z^{(N)}} \cdot \frac{\partial z^{(N)}}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial W^{(N)}} \end{aligned}$$
>
> 而等式右边的每一项都是可以求的。

> **第N-1层：**
>
> $$\begin{aligned} \frac{\partial E}{\partial b^{(N-1)}} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial z^{(N-1)}} \cdot \frac{\partial z^{(N-1)}}{\partial a^{(N-1)}} \cdot \frac{\partial a^{(N-1)}}{\partial b^{(N-1)}} \\ &= \frac{\partial E}{\partial z^{(N)}} \cdot \frac{\partial z^{(N)}}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial z^{(N-1)}} \cdot \frac{\partial z^{(N-1)}}{\partial a^{(N-1)}} \cdot \frac{\partial a^{(N-1)}}{\partial b^{(N-1)}} \\ \frac{\partial E}{\partial W^{(N-1)}} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial z^{(N-1)}} \cdot \frac{\partial z^{(N-1)}}{\partial a^{(N-1)}} \cdot \frac{\partial a^{(N-1)}}{\partial W^{(N-1)}} \\ &= \frac{\partial E}{\partial z^{(N)}} \cdot \frac{\partial z^{(N)}}{\partial a^{(N)}} \cdot \frac{\partial a^{(N)}}{\partial z^{(N-1)}} \cdot \frac{\partial z^{(N-1)}}{\partial a^{(N-1)}} \cdot \frac{\partial a^{(N-1)}}{\partial W^{(N-1)}} \end{aligned}$$

> 尽管已经十分复杂，但此时等式右边的每一项也都是可以求的。

懒得往下继续写了。总之，以此类推，通过观察可以发现，在计算第 $N-k$ 层的参数的导数时，需要计算的项包括了计算从第 N 层到第 $N-k+1$ 层的参数的导数的绝大多数项。（事实上，除了最后一项都包括了）那么如果我们记录之前计算的中间结果，那么计算导数的计算量就会被大大简化。这就是反向传播算法的思想。具体方法如下：

如果我们记中间变量[1]

$$\delta^{(k)} := \frac{\partial E}{\partial a^{(k)}} \tag{7}$$

即

$$\delta_j^{(k)} := \frac{\partial E}{\partial a_j^{(k)}} \tag{7'}$$

那么要计算的损失函数相对于参数 $W^{(k)}$ 和 $b^{(k)}$ 的梯度就可以直接写出：

$$\begin{gathered}\frac{\partial E}{\partial b^{(k)}} = \frac{\partial E}{\partial a^{(k)}} \cdot \frac{\partial a^{(k)}}{\partial b^{(k)}} = \frac{\partial E}{\partial a^{(k)}} \\ \frac{\partial E}{\partial W^{(k)}} = \frac{\partial E}{\partial a^{(k)}} \cdot \frac{\partial a^{(k)}}{\partial W^{(k)}}\end{gathered} \tag{8}$$

注意到 $\frac{\partial a^{(k)}}{\partial W^{(k)}}$ 是向量对矩阵求导的形式，没有办法用矩阵乘法的方式简洁的表示计算结果[2]，故我们逐分量计算

$$\text{i.e.} \qquad \begin{gathered}\frac{\partial E}{\partial b_i^{(k)}} = \delta_i^{(k)} \\ \frac{\partial E}{\partial W_{ij}^{(k)}} = \delta_i^{(k)} \cdot z_j^{(k-1)}\end{gathered} \tag{8'}$$

这就实现了用 $\delta^{(k)}$ 简便计算 $W^{(k)}$ 和 $b^{(k)}$ 。

下一步，则是用 $\delta^{(k)}$ 计算 $\delta^{(k-1)}$，用后一层的导数计算前一层的导数，即所谓“反向传播”的本意。仍然用复合函数的求导法则可得

$$\delta^{(k-1)} = \frac{\partial E}{\partial a^{(k-1)}} = \frac{\partial E}{\partial a^{(k)}} \cdot \frac{\partial a^{(k)}}{\partial z^{(k-1)}} \cdot \frac{\partial z^{(k-1)}}{\partial a^{(k-1)}}$$

利用矩阵求导[3]的方式可写作

$$\delta^{(k-1)} = \delta^{(k)} \cdot W^{(k)} \cdot \operatorname{diag}\left\{h^{(k+1)\prime}\left(a^{(k-1)}\right)\right\} \tag{9}$$

其中 $\operatorname{diag} v$ 表示 一个以向量 v 的各分量为对角元的对角矩阵。如果搞不清楚矩阵形式的计算方式，老老实实写成分量形式计算就好了，最终结果为：

$$\delta_j^{(k-1)} = \left(\sum_i \delta_i^{(k)} W_{ij}^{(k)}\right) \cdot h^{(k-1)\prime}\left(a_j^{(k-1)}\right) \tag{9'}$$

最终反向传播算法的公式总结为：

$$\begin{cases} \delta_j^{(N)} = \frac{\partial E}{\partial y} \cdot h^{(N)\prime}\left(a_j^N\right) \qquad ,\forall j \\ \frac{\partial E}{\partial b_i^{(k)}} = \delta_i^{(k)} \qquad ,\forall i, \quad \forall k = 1, \cdots, N \\ \frac{\partial E}{\partial W_{ij}^{(k)}} = \delta_i^{(k)} \cdot z_j^{(k-1)} \qquad ,\forall i,j, \quad \forall k = 1, \cdots, N \\ \delta_j^{(k-1)} = \left(\sum_i \delta_i^{(k)} W_{ij}^{(k)}\right) \cdot h^{(k-1)\prime}\left(a_j^{(k-1)}\right) \qquad ,\forall j, \quad \forall k = 2, \cdots, N \end{cases} \tag{10}$$

注意

1. 数据规范化的情况

有的时候，公式（2）会写成

$$a^{(k)} = \tilde{W}^{(k)} \cdot \tilde{z}^{(k-1)} \tag{2'}$$

的形式。其中 $\tilde{z}^{(k-1)}$ 就是在 原本的 $\tilde{z}^{(k-1)}$ 的基础上增加一维 $\tilde{z}_0^{(k-1)} = 1$ ，相应的 $W^{(k)}$ 也会增加一列变成 $\tilde{W}^{(k)}$ ，这样将 $+b^{(k)}$ 就统一进了矩阵乘法之中。这是机器学习的常见套路了，在此不多解释。在这个情况下，计算时要注意的变化，简而言之就是：

1. **前馈传播时，只有$\tilde{z}^{(k)}$ 和 $\tilde{W}^{(k)}$ 的列有0号分量，只需注意 $\tilde{z}_0^{(k)} = 1$ 不需要被计算即可。**
2. **反向传播时，只有$\tilde{z}^{(k)}$ 和 $\tilde{W}^{(k)}$ 的列有0号分量，不需额外注意，正常计算全部导数值即可。**

总之这里根据你编程序的代码实际运行即可。

2. 向量形式公式的不同形式

在反向传播算法中，分量形式的公式是没有歧义的。但向量形式的公式（9）有不同的写法。歧义点主要有2个：

1. 转置问题： $\delta^{(k)} \cdot W^{(k)}$ 还是 ${W^{(k)}}^T \cdot \delta^{(k)}$?

   这里主要是看 $\delta^{(k)}$ 是行向量还是列向量，根据不同的布局规则有不同的结果，总之要把矩阵乘法的维数对齐。本人统一采用分子布局（同本人大学数学分析课的规则），就是列向量对标量求导还是列向量的规则。主要好处是复合函数的求导准则和原来相同。
2. Hadamard积： $* \cdot \operatorname{diag}\left\{h^{(k+1)\prime}\left(a^{(k-1)}\right)\right\}$ 在某些文章里会写作 $* \odot h^{(k+1)\prime}\left(a^{(k-1)}\right)$

   后者的运算叫作Hadamard积，定义为 $A \odot B = (a_{ij} * b_{ij})_{mxn}$ ，即对应分量相乘。这两种写法的意思是相同的，后者的好处是编程的时候计算方便，不需要额外把向量转化成对角矩阵的步骤。

总结

前馈传播——公式（5），具体解释（1-4）

反向传播——公式（10），具体解释（7-9‘）

拓展

在反向传播算法中，我们用来记忆之前的计算过程的中间变量 $\delta^{(k)} := \frac{\partial E}{\partial a^{(k)}}$ ，但其实这并不是唯一的方式，读者可以尝试用 $\frac{\partial E}{\partial z^{(k)}}$ 作为中间变量，推导一下此时的“反向传播算法”会是什么样子，检验一下自己的理解和计算能力。

[1]注意，根据分子布局规则，$\delta^{(k)}$ 作为标量对列向量求导的结果，是行向量。直接从分量角度考虑可以规避这些细节。

[2]见我还没写的文章《向量函数的求导》

[3]见我还没写的文章《向量函数的求导》