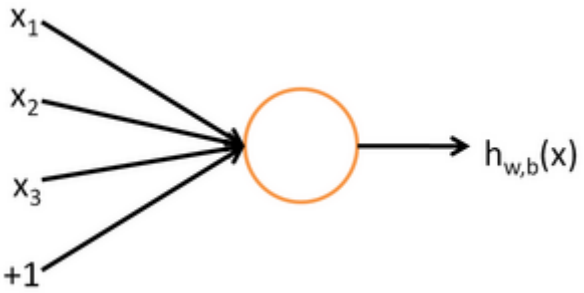
神经网络--反向传播详细推导过程

概述

以监督学习为例，假设我们有训练样本集 $(x^{(i)}, y^{(i)})$，那么神经网络算法能够提供一种复杂且非线性的假设模型 $h_{W,b}(x)$，它具有参数 W, b，可以以此参数来拟合我们的数据。

为了描述神经网络，我们先从最简单的神经网络讲起，这个神经网络仅由一个“神经元”构成，以下即是这个“神经元”的图示：



这个“神经元”是一个以 x_1, x_2, x_3 及截距 $+1$ 为输入值的运算单元，其输出为 $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$，其中函数 $f : \Re \mapsto \Re$ 被称为“激活函数”。在本教程中，我们选用 sigmoid 函数作为**激活函数** $f(\cdot)$

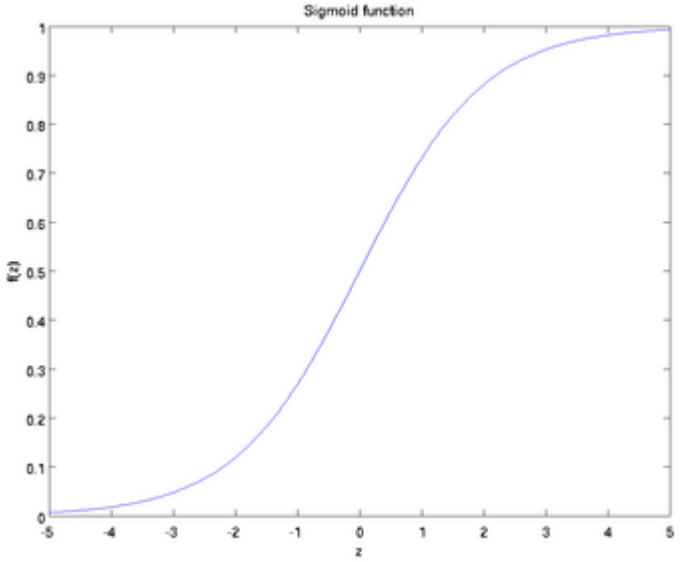
$$f(z) = \frac{1}{1 + \exp(-z)}.$$

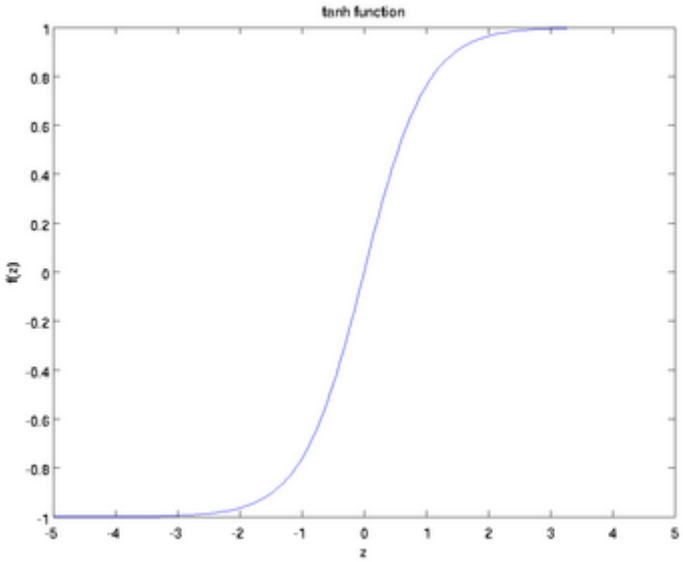
可以看出，这个单一“神经元”的输入－输出映射关系其实就是一个逻辑回归（logistic regression）。

虽然本系列教程采用 sigmoid 函数，但你也可以选择双曲正切函数（tanh）：

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

以下分别是 sigmoid 及 tanh 的函数图像





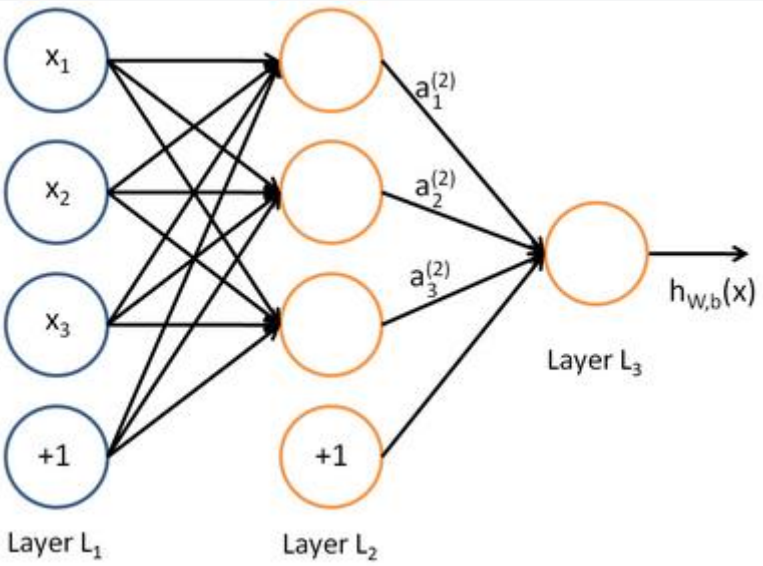
$\tanh(z)$ 函数是 sigmoid 函数的一种变体，它的取值范围为 $[-1, 1]$，而不是 sigmoid 函数的 $[0, 1]$。

注意，与其它地方（包括 OpenClassroom 公开课以及斯坦福大学 CS229 课程）不同的是，这里我们不再令 $x_0 = 1$。取而代之，我们用单独的参数 b 来表示截距。

最后要说明的是，有一个等式我们以后会经常用到：如果选择 $f(z) = 1/(1+\exp(-z))$，也就是 sigmoid 函数，那么它的导数就是 $f'(z) = f(z)(1-f(z))$（如果选择 tanh 函数，那它的导数就是 $f'(z) = 1-(f(z))^2$，你可以根据 sigmoid（或 tanh）函数的定义自行推导这个等式。

神经网络模型

所谓神经网络就是将许多个单一“神经元”联结在一起，这样，一个“神经元”的输出就可以是另一个“神经元”的输入。例如，下图就是一个简单的神经网络：



我们使用圆圈来表示神经网络的输入，标上“$+1$”的圆圈被称为**偏置节点**，也就是截距项。神经网络最左边的一层叫做**输入层**，最右的一层叫做**输出层**（本例中，输出层只有一个节点）。中间所有节点组成的一层叫做**隐藏层**，因为我们不能在训练样本集中观测到它们的值。同时可以看到，以上神经网络的例子中有 3 个**输入单元**（偏置单元不计在内），3 个**隐藏单元**及一个**输出单元**。

我们用 n_l 来表示网络的层数，本例中 $n_l = 3$，我们将第 l 层记为 L_l，于是 L_1 是输入层，输出层是 L_{n_l}。本例神经网络有参数 $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$，其中 $W_{ij}^{(l)}$ （下面的式子中用到）是第 l 层第 j 单元与第 $l+1$ 层第 i 单元之间的联接参数（其实就是连接线上的权重，注意标号顺序），$b_i^{(l)}$ 是第 $l+1$ 层第 i 单元的偏置项。因此在本例中，$W^{(1)} \in \Re^{3\times 3}$，$W^{(2)} \in \Re^{1\times 3}$。注意，没有其他单元连向偏置单元(即偏置单元没有输入)，因为它们总是输出 $+1$。同时，我们用 s_l 表示第 l 层的节点数（偏置单元不计在内）。

我们用 $a_i^{(l)}$ 表示第 l 层第 i 单元的**激活值**（输出值）。
当 $l = 1$ 时，$a_i^{(1)} = x_i$，也就是第 i 个输入值（输入值的第 i 个特征）。

对于给定参数集合 W, b，我们的神经网络就可以按照函数 $h_{W,b}(x)$ 来计算输出结果。本例神经网络的计算步骤如下：

$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

我们用 $z_i^{(l)}$ 表示第 l 层第 i 单元输入加权和（包括偏置单元），比如，$z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(1)}$，则 $a_i^{(l)} = f(z_i^{(l)})$。

这样我们就可以得到一种更简洁的表示法。这里我们将激活函数 $f(\cdot)$ 扩展为用向量（分量的形式）来表示，
即 $f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$，那么，上面的等式可以更简洁地表示为：

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{aligned}
$$

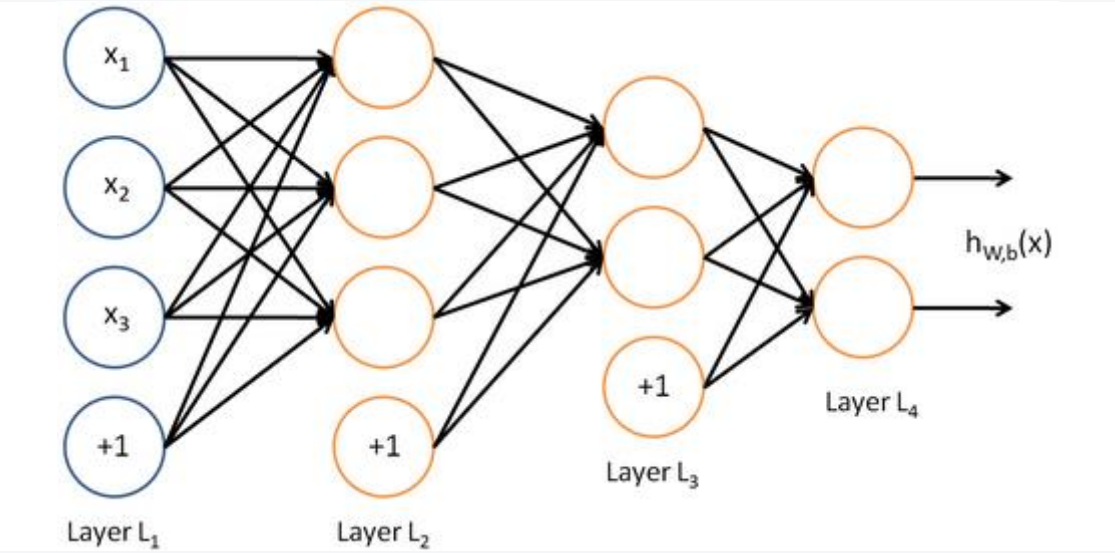
我们将上面的计算步骤叫作**前向传播**。回想一下，之前我们用 $a^{(1)} = x$ 表示输入层的激活值，那么给定第 l 层的激活值 $a^{(l)}$ 后，第 $l+1$ 层的激活值 $a^{(l+1)}$ 就可以按照下面步骤计算得到：

$$
\begin{aligned}
z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\
a^{(l+1)} &= f(z^{(l+1)})
\end{aligned}
$$

将参数矩阵化，使用矩阵－向量运算方式，我们就可以利用线性代数的优势对神经网络进行快速求解。

目前为止，我们讨论了一种神经网络，我们也可以构建另一种**结构**的神经网络（这里结构指的是神经元之间的联接模式），也就是包含多个隐藏层的神经网络。最常见的一个例子是 n_l 层的神经网络，第 1 层是输入层，第 n_l 层是输出层，中间的每个层 l 与层 $l+1$ 紧密相联。这种模式下，要计算神经网络的输出结果，我们可以按照之前描述的等式，按部就班，进行前向传播，逐一计算第 L_2 层的所有激活值，然后是第 L_3 层的激活值，以此类推，直到第 L_{n_l} 层。这是一个**前馈**神经网络的例子，因为这种联接图没有闭环或回路。

神经网络也可以有多个输出单元。比如，下面的神经网络有两层隐藏层：L_2 及 L_3，输出层 L_4 有两个输出单元。



要求解这样的神经网络，需要样本集 $(x^{(i)}, y^{(i)})$，其中 $y^{(i)} \in \Re^2$。如果你想预测的输出是多个的，那这种神经网络很适用。（比如，在医疗诊断应用中，患者的体征指标就可以作为向量的输入值，而不同的输出值 y_i 可以表示不同的疾病存在与否。）

反向传导算法

假设我们有一个固定样本集 $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$，它包含 m 个样例。我们可以用批量梯度下降法来求解神经网络。具体来讲，对于单个样例 (x, y)，其代价函数为：

$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2.$$

这是一个（二分之一的）方差代价函数。给定一个包含 m 个样例的数据集，我们可以定义整体代价函数为：

$$\begin{aligned}J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2\end{aligned}$$

以上关于$J(W,b)$定义中的第一项是一个均方差项。第二项是一个规则化项（也叫**权重衰减项**），其目的是减小权重的幅度，防止过度拟合。

[注：通常权重衰减的计算并不使用偏置项 $b_i^{(l)}$，比如我们在 $J(W,b)$ 的定义中就没有使用。一般来说，将偏置项包含在权重衰减项中只会对最终的神经网络产生很小的影响。如果你在斯坦福选修过 CS229（机器学习）课程，或者在 YouTube 上看过课程视频，你会发现这个权重衰减实际上是课上提到的贝叶斯规则化方法的变种。在贝叶斯规则化方法中，我们将高斯先验概率引入到参数中计算 MAP（极大后验）估计（而不是极大似然估计）。]

权重衰减参数 λ 用于控制公式中两项的相对重要性。在此重申一下这两个复杂函数的含义：$J(W,b;x,y)$ 是针对单个样例计算得到的方差代价函数；$J(W,b)$ 是整体样本代价函数，它包含权重衰减项。

以上的代价函数经常被用于分类和回归问题。在分类问题中，我们用 $y = 0$ 或 1，来代表两种类型的标签（回想一下，这是因为 sigmoid 激活函数的值域为 $[0,1]$；如果我们使用双曲正切型激活函数，那么应该选用 -1 和 $+1$ 作为标签）。对于回归问题，我们首先要变换输出值域（译者注：也就是 y），以保证其范围为 $[0,1]$（同样地，如果我们使用双曲正切型激活函数，要使输出值域为 $[-1,1]$）。

我们的目标是针对参数 W 和 b 来求其函数 $J(W,b)$ 的最小值。为了求解神经网络，我们需要将每一个参数 $W_{ij}^{(l)}$ 和 $b_i^{(l)}$ 初始化为一个很小的、接近零的随机值（比如说，使用正态分布 $Normal(0,\epsilon^2)$ 生成的随机值，其中 ϵ 设置为 0.01），之后对目标函数使用诸如批量梯度下降法的最优化算法。因为 $J(W,b)$ 是一个非凸函数，梯度下降法很可能会收敛到局部最优解；但是在

实际应用中，梯度下降法通常能得到令人满意的结果。最后，需要再次强调的是，要将参数进行随机初始化，而不是全部置为0。如果所有参数都用相同的值作为初始值，那么所有隐藏层单元最终会得到与输入值有关的、相同的函数（也就是说，对于所有i，$W_{ij}^{(1)}$都会取相同的值，那么对于任何输入x都会有：$a_1^{(2)} = a_2^{(2)} = a_3^{(2)} = \ldots$）。随机初始化的目的是使**对称失效**。

梯度下降法中每一次迭代都按照如下公式对参数W和b进行更新：

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)$$

其中α是学习速率。其中关键步骤是计算偏导数。我们现在来讲一下**反向传播**算法，它是计算偏导数的一种有效方法。

我们首先来讲一下如何使用反向传播算法来计算$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y)$和$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y)$，这两项是单个样例$(x,y)$的代价函数$J(W,b;x,y)$的偏导数。一旦我们求出该偏导数，就可以推导出整体代价函数$J(W,b)$的偏导数：

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

以上两行公式稍有不同，第一行比第二行多出一项，是因为权重衰减是作用于W而不是b。

反向传播算法的思路如下：给定一个样例(x,y)，我们首先进行“前向传导”运算，计算出网络中所有的激活值，包括$h_{W,b}(x)$的输出值。之后，针对第l层的每一个节点i，我们计算出其“残差”$\delta_i^{(l)}$，该残差表明了该节点对最终输出值的残差产生了多少影响。对于最终的输出节点，我们可以直接算出网络产生的激活值与实际值之间的差距，我们将这个差距定义为$\delta_i^{(n_l)}$（第n_l层表示输出层）。对于隐藏单元我们如何处理呢？我们将基于节点（译者注：第$l+1$层

节点）残差的加权平均值计算 $\delta_i^{(l)}$，这些节点以 $a_i^{(l)}$ 作为输入。下面将给出反向传导算法的细节：

1. 进行前馈传导计算，利用前向传导公式，得到 $L_2, L_3, \ldots$ 直到输出层 L_{n_l} 的激活值。
2. 对于第 n_l 层（输出层）的每个输出单元 i，我们根据以下公式计算残差：

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

[译者注：

$$\begin{aligned}\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{n_l}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 \\ &= \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - a_j^{(n_l)})^2 = \frac{\partial}{\partial z_i^{n_l}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - f(z_j^{(n_l)}))^2 \\ &= -(y_i - f(z_i^{(n_l)})) \cdot f'(z_i^{(n_l)}) = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})\end{aligned}$$

]

3. 对 $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ 的各个层，第 l 层的第 i 个节点的残差计算方法如下：

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

{译者注：

$$
\begin{aligned}
\delta_i^{(n_l-1)} &= \frac{\partial}{\partial z_i^{n_l-1}} J(W,b;x,y) = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = \frac{\partial}{\partial z_i^{n_l-1}} \frac{1}{2} \sum_{j=1}^{S_{n_l}} (y_j - \\
&= \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - a_j^{(n_l)})^2 = \frac{1}{2} \sum_{j=1}^{S_{n_l}} \frac{\partial}{\partial z_i^{n_l-1}} (y_j - f(z_j^{(n_l)}))^2 \\
&= \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} f(z_j^{(n_l)}) = \sum_{j=1}^{S_{n_l}} -(y_j - f(z_j^{(n_l)})) \cdot f'(z \\
&= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{n_l-1}} = \sum_{j=1}^{S_{n_l}} \left(\delta_j^{(n_l)} \cdot \frac{\partial}{\partial z_i^{n_l-1}} \sum_{k=1}^{S_{n_l-1}} f(z_k^{n_l-1}) \cdot W_{jk}^{n_l-1} \right) \\
&= \sum_{j=1}^{S_{n_l}} \delta_j^{(n_l)} \cdot W_{ji}^{n_l-1} \cdot f'(z_i^{n_l-1}) = \left(\sum_{j=1}^{S_{n_l}} W_{ji}^{n_l-1} \delta_j^{(n_l)} \right) f'(z_i^{n_l-1})
\end{aligned}
$$

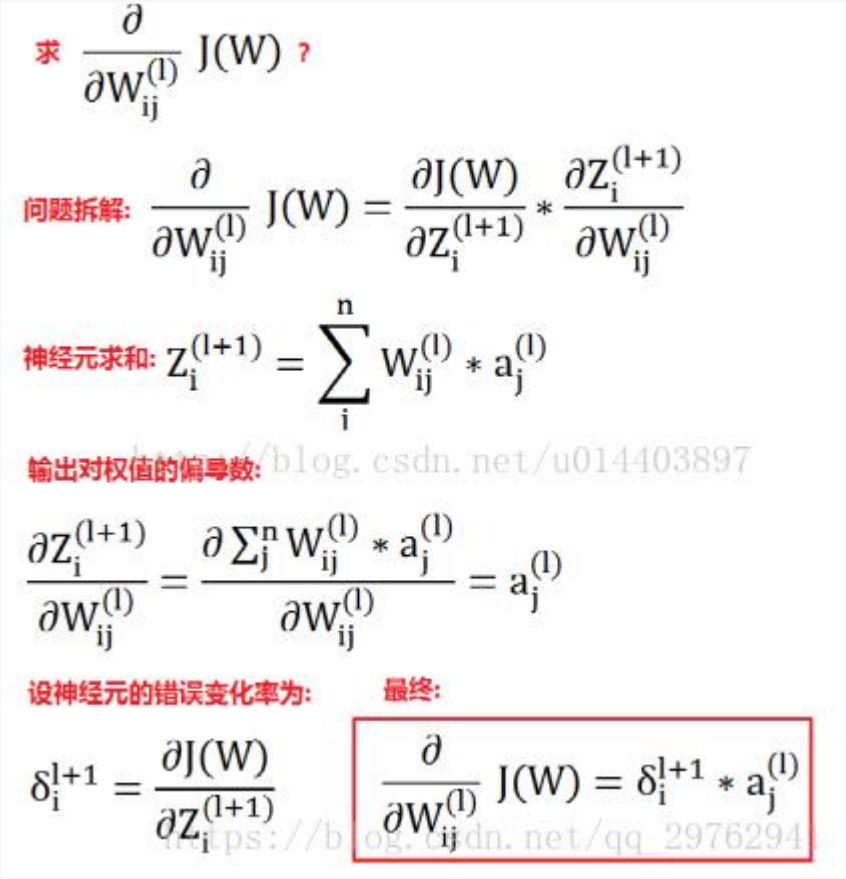
将上式中的$n_l - 1$与n_l的关系替换为l与$l + 1$的关系，就可以得到：

$$
\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})
$$

以上逐次从后向前求导的过程即为“反向传导”的本意所在。

4. 计算我们需要的偏导数，计算方法如下：

$$
\begin{aligned}
\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) &= a_j^{(l)} \delta_i^{(l+1)} \\
\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) &= \delta_i^{(l+1)}.
\end{aligned}
$$



最后，我们用矩阵-向量表示法重写以上算法。我们使用“$\bullet$” 表示向量乘积运算符（在 Matlab 或 Octave 里用“.*”表示，也称作阿达马乘积）。若 $a = b \bullet c$，则 $a_i = b_i c_i$。在上一个教程中我们扩展了 $f(\cdot)$ 的定义，使其包含向量运算，这里我们也对偏导数 $f'(\cdot)$ 也做了同样的处理（于是又有 $f'([z_1, z_2, z_3]) = [f'(z_1), f'(z_2), f'(z_3)]$）。

那么，反向传播算法可表示为以下几个步骤：

1. 进行前馈传导计算，利用前向传导公式，得到 $L_2, L_3, \ldots$ 直到输出层 L_{n_l} 的激活值。
2. 对输出层（第 n_l 层），计算：

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. 对于 $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$ 的各层，计算：

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. 计算最终需要的偏导数值：

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$
$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

实现中应注意：在以上的第 2 步和第 3 步中，我们需要为每一个 i 值计算其 $f'(z_i^{(l)})$。假设 $f(z)$ 是 sigmoid 函数，并且我们已经在前向传导运算中得到了 $a_i^{(l)}$。那么，使用我们早先推导出的 $f'(z)$表达式，就可以计算得到 $f'(z_i^{(l)}) = a_i^{(l)}(1 - a_i^{(l)})$。

最后，我们将对梯度下降算法做个全面总结。在下面的伪代码中，$\Delta W^{(l)}$ 是一个与矩阵 $W^{(l)}$ 维度相同的矩阵，$\Delta b^{(l)}$ 是一个与 $b^{(l)}$ 维度相同的向量。注意这里“$\Delta W^{(l)}$”是一个矩阵，而不是“Δ 与 $W^{(l)}$ 相乘”。下面，我们实现批量梯度下降法中的一次迭代：

1. 对于所有 l，令 $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$ （设置为全零矩阵或全零向量）
2. 对于 $i = 1$ 到 m，
    1. 使用反向传播算法计算 $\nabla_{W^{(l)}} J(W,b;x,y)$ 和 $\nabla_{b^{(l)}} J(W,b;x,y)$。
    2. 计算 $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W,b;x,y)$。
    3. 计算 $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W,b;x,y)$。
3. 更新权重参数：

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$
$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

现在，我们可以重复梯度下降法的迭代步骤来减小代价函数 $J(W,b)$ 的值，进而求解我们的神经网络。