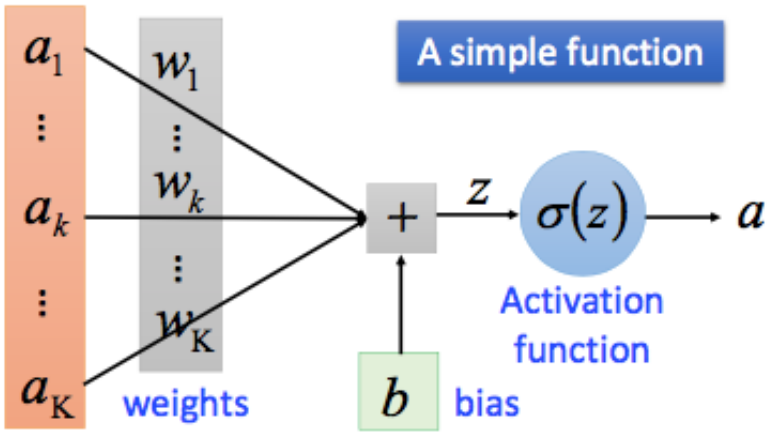
深度学习系列（1）：神经网络与反向传播算法

一、神经元

首先我们从最简单的神经网络——神经元讲起，以下即为一个神经元（Neuron）的图示：

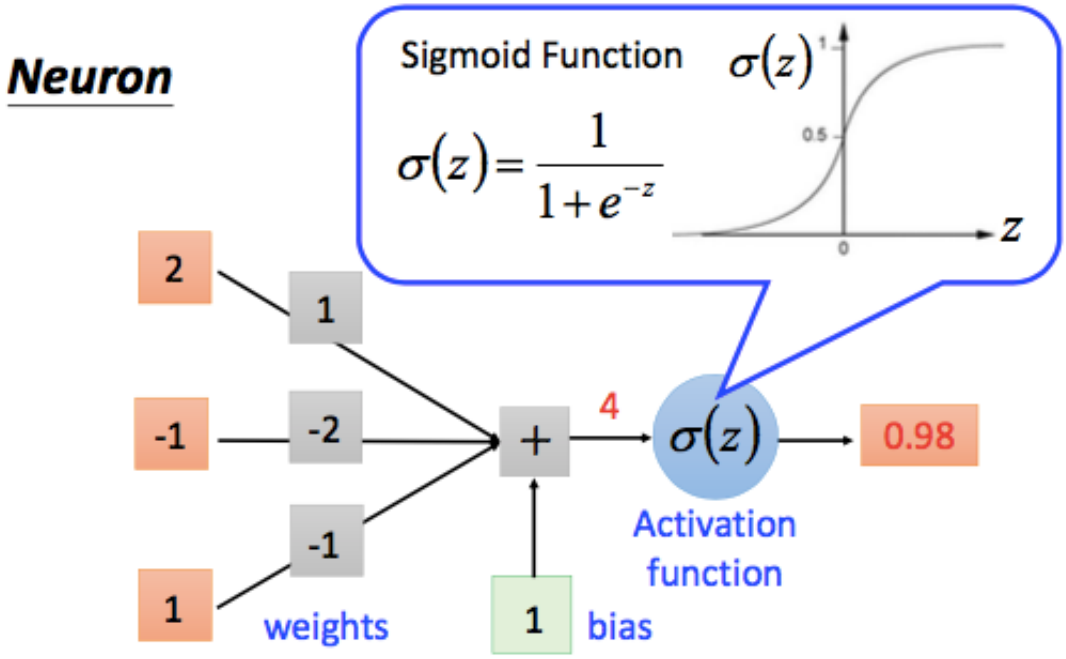
$$z = a_1 w_1 + \cdots + a_k w_k + \cdots + a_K w_K + b$$



这个神经元是一个以$x_1, x_2, \cdots, x_K$以及截距b为输入值的运算单元，其输出为

$$\alpha = \sigma\left(w^T a + b\right) = \sigma\left(w_1 a_1 + w_2 a_2 + \cdots + w_K a_K + b\right)$$

其中w为权值项，b为偏置项，函数σ被称为“激活函数”。之前在学习感知机的时候，我们知道感知机的激活函数是阶跃函数；而当我们说神经元的时，激活函数往往选择sigmoid函数或tanh函数。激活函数的作用就是将之前加法器输出的函数值z进行空间映射，如下图所示：

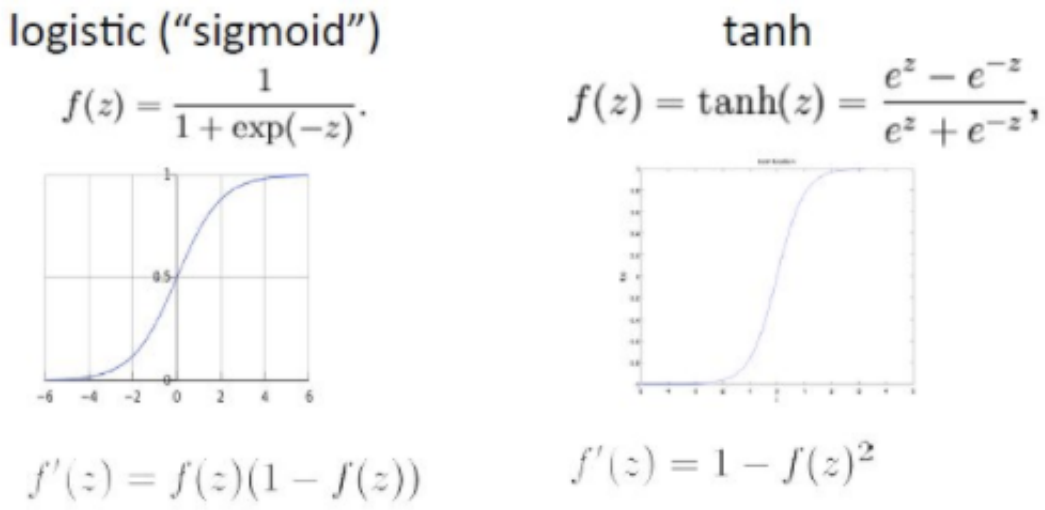


可以看出，这个单一神经元的输入输出的映射关系其实就是一个逻辑回归（logistic regression）。

关于sigmoid阶跃函数的性质，在逻辑回归中已经了解过了，有一个等式我们会用到：$f'(z)=f(z)(1-f(z))$。现在我们简要看一下双曲正切函数（tanh）。它的表达式为：

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

它们图像为

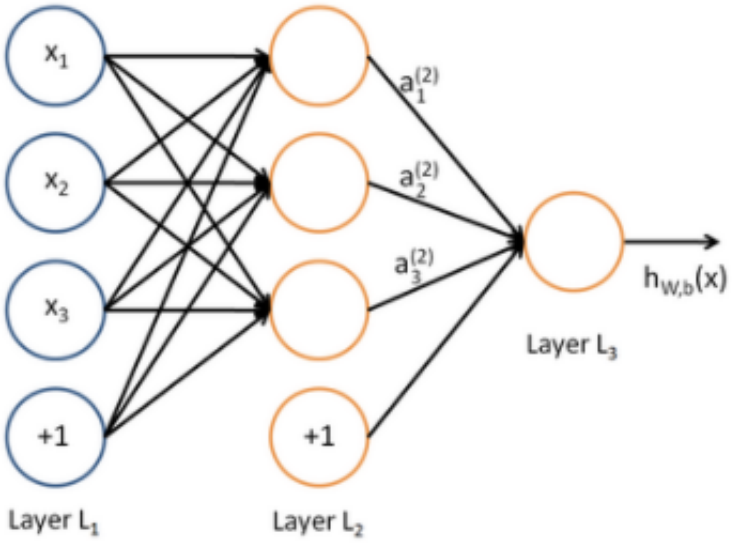


tanh（z）函数是sigmoid函数的一种变体，它的取值范围为[-1,1]，而不是sigmoid函数的[0,1]，它的导数为$f'(z)=1-(f(z))^2$

二、神经网络模型

2.1 神经网络模型

所谓神经网络就是将许多神经元联结在一起，这样，一个神经元的输出就可以是另一神经元的输入。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，最右边的一层叫做输出层（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，如此命名是因为我们不能在训练样本中观测到它们的值。同时可以看到，以上神经网络的例子中有3个输入单元（偏置单元不算在内），三个隐藏单元及一个输出单元。

我们用n_l来表示神经网络的层数，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_{nl}。本例神经网络有参数$(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$，其中$W_{ij}^{(l)}$是第$l$层第$j$个单元与第$l+1$层第$i$单元之间的联接参数（其实就是连接线上的权重，注意标号前后顺序），$b_i^{(l)}$是第$l+1$层第i个单元的偏置项。偏置单元没有输入，即没有单元连向偏置单元，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个单元的激活值（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$h_{W,b}(x)$来计算结果。本例中神经网络的计算步骤如下：

$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$

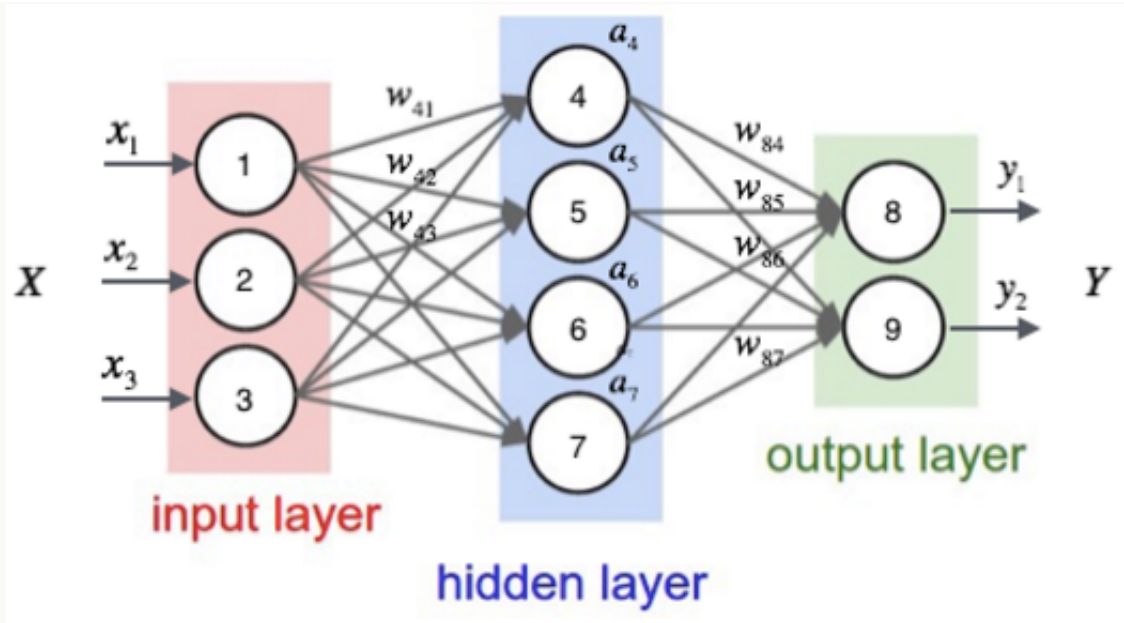
$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right)$$

$$a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}\right)$$

$$h_{w,b}(x) = a_1^{(3)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(2)}\right)$$

2.2 具体举例

接下来举一个具体的例子来说明这个过程，我们先给神经网络的每个单元写上编号。



图中，输入层有三个节点，我们将其依次编号为1，2，3；隐藏层的4个节点，编号依次为4，5，6，7；最后输出层的两个节点编号为8，9。因为我们这个神经网络是全连接网络，所以可以看到每个节点都和上一层的所有节点有链接。比如我们可以看到隐藏层的节点4，它和输入层的三个节点1，2，3之间都有连接，其连接上的权重分别为w_{41}, w_{42}, w_{43}。那么，我们怎样计算节点4的输出值a_4呢?

为了计算节点4的输出值，我们必须先得到其所有上游节点（也就是节点1，2，3）的输出值。节点1、2、3是输入层的节点，所以，他们的输出值就是向量$\vec{x}$。按照上图画出的对应关系，可以看到节点1、2、3的输出值分别是x_1, x_2, x_3。

一旦我们有了节点1、2、3的输出值，我们就可以计算节点4的输出值a_4：

$$a_4 = f(\vec{w} \cdot \vec{x}) = f(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + w_{4b})$$

其中w_{4b}是节点4的偏置项，图中没有画出来。而w_{41}, w_{42}, w_{43}分别为节点1、2、3到节点4连接的权重，在给权值w_{ij}编号时，我们把目标节点的编号i放在前面，把源节点的编号i放在后面。

同样，我们可以继续计算出节点5、6、7的输出值a_5, a_6, a_7。这样，隐藏层的4个节点的输出值就计算完成了，我们就可以接着计算输出层的节点8的输出值y_1：

$$y_1 = f(\vec{w} \cdot \vec{x}) = f(w_{84}a_4 + w_{85}a_5 + w_{86}a_6 + w_{87}a_7 + w_{8b})$$

同理，我们还可以计算出y_2的值。这样输出层所有节点的输出值计算完毕，我们就得到了在输入向量$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$时，神经网络的输出向量$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$。这里我们也看到，输出向量的维度和输出层神经元个数相同。

2.3 神经网络的矩阵表示

神经网络的计算如果用矩阵来表示会很方便，我们先来看看隐藏层的矩阵表示。
首先我们把隐藏层4个节点的计算依次排列出来：

$$a_4 = f(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + w_{4b})$$

$$a_5 = f(w_{51}x_1 + w_{52}x_2 + w_{53}x_3 + w_{5b})$$

$$a_6 = f(w_{61}x_1 + w_{62}x_2 + w_{63}x_3 + w_{6b})$$

$$a_7 = f(w_{71}x_1 + w_{72}x_2 + w_{73}x_3 + w_{7b})$$

接着，定义神经网络的输入向量$\vec{x}$和隐藏层每个节点的权重向量$\overrightarrow{w_j}$。令

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix}$$

$$\overrightarrow{w_4} = [w_{41}, w_{42}, w_{43}, w_{4b}]$$

$$\overrightarrow{w_5} = [w_{51}, w_{52}, w_{53}, w_{5b}]$$

$$\overrightarrow{w_6} = [w_{61}, w_{62}, w_{63}, w_{6b}]$$

$$\overrightarrow{w_7} = [w_{71}, w_{72}, w_{73}, w_{7b}]$$

代入之前的一组式子，得到

$$a_4 = f(\overrightarrow{w_4} \cdot \vec{x})$$

$$a_5 = f(\overrightarrow{w_5} \cdot \vec{x})$$

$$a_6 = f(\overrightarrow{w_6} \cdot \vec{x})$$

$$a_7 = f(\overrightarrow{w_7} \cdot \vec{x})$$

现在，我们把上述计算a_4, a_5, a_6, a_7的四个式子写到一个矩阵里面，每个式子作为矩阵的一行，就可以利用矩阵来表示他们的计算了。令

$$\vec{a} = \begin{bmatrix} a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}$$

$$\vec{W} = \begin{bmatrix} \vec{w_4} \\ \vec{w_5} \\ \vec{w_6} \\ \vec{w_7} \end{bmatrix} = \begin{bmatrix} w_{41} & w_{42} & w_{43} & w_{4b} \\ w_{51} & w_{52} & w_{53} & w_{5b} \\ w_{61} & w_{62} & w_{63} & w_{6b} \\ w_{71} & w_{72} & w_{73} & w_{7b} \end{bmatrix}$$

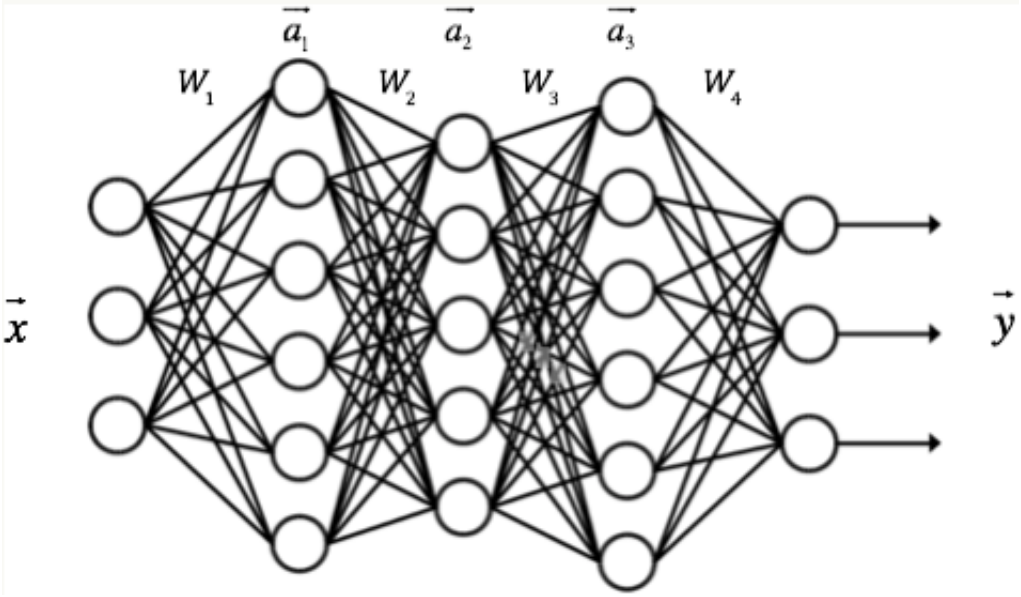
$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ .. \\ .. \end{bmatrix}\right) = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \cdots \end{bmatrix}$$

代入前面的一组式子，得到

$$\vec{a} = f(W \cdot \vec{x})$$

在上式中，f是激活函数，在本例中为sigmoid函数；W是某一层的权重矩阵；$\vec{x}$是某层的输入向量；$\vec{a}$是某层的输出向量。它说明了神经网络的每一层的作用实际上就是先将输入向量左乘一个数组进行线性变换，得到一个新的向量，然后再对这个向量逐元素应用一个激活函数。

每一层的算法都是一样的。比如，对于包含一个输入层，一个输出层和三个隐藏层的神经网络，我们假设其权重举证分别为W_1, W_2, W_3, W_4,每个隐藏层的输出分别是$\vec{a_1}, \vec{a_2}, \vec{a_3}$，神经网络的输入为$\vec{x}$，神经网络的输入为$\vec{y}$，如下图所示：



则每一层的输出向量的计算可以表示为：

$$\vec{a_1} = f(W_1 \cdot \vec{x})$$

$$\vec{a_2} = f(W_2 \cdot \vec{a_1})$$

$$\vec{a_3} = f(W_3 \cdot \vec{a_2})$$

$$\vec{y} = f(W_4 \cdot \vec{a_3})$$

这就是神经网络输出值的计算方法。

三、反向传导算法

3.1 损失函数与正则化项

假设我们有一个固定样本集$\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$,它包含$m$个样本。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：

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

这是一个平方误差损失函数。对于包含m个样本的数据集，我们可以定义整体的损失函数为：

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

$$= \left[\frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \parallel h_{W,b}\left(x^{(i)}\right) - y^{(i)} \parallel^2 \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ij}^{(l)}\right)^2$$

以上关于$J(W,b)$定义中的第一项是均方误差项，第二项是一个正则化项，也叫权重衰减项，其目的就是减小权重的幅度，防止过度拟合。权重衰减参数λ用于控制公式中两项的相对重要性。需要注意的是，$J(W,b;x,y)$是针对单个样本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。

3.2 反向传播算法

反向传播算法其实就是链式求导法则的应用。然而，这个如此简单且显而易见的方法，却是在Roseblatt剔除感知机算法将近30年之后才被发明和普及的。接下来，我们用链式求导法则来推导反向传播算法。

按照机器学习的通用套路，我们先确定神经网络的目标函数，然后用随机梯度下降优化算法去求目标函数最小值时的参数值。

假设我们的参数集合为$\theta = \{w_1, w_2, \cdots, b_1, b_2 \cdots\}$，设初始参数为$\theta^0$，将损失函数$L(\theta)$分别对参数求导：

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta)/\partial w_1 \\ \partial L(\theta)/\partial w_2 \\ \cdots \\ \partial L(\theta)/\partial b_1 \\ \partial L(\theta)/\partial b_2 \\ \cdots \end{bmatrix}$$

计算$\nabla L(\theta^0)$，参数更新

$$\theta^1 = \theta^0 - \eta \nabla L(\theta^0)$$

计算$\nabla L(\theta^1)$，参数更新

$$\theta^2 = \theta^1 - \eta \nabla L(\theta^1)$$

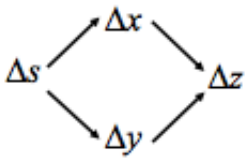
因为推导过程需要用到链式法则，具体如下图所示：

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$



$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

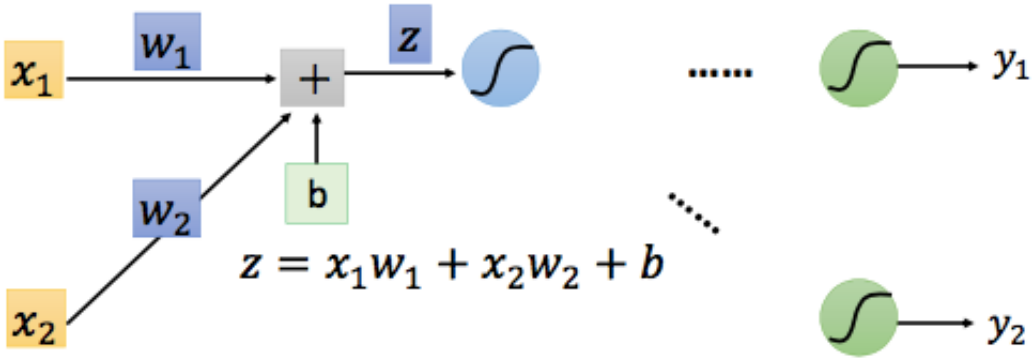
我们定义整体损失函数为：

$$L(\theta) = \sum_{n=1}^{N} C^n(\theta)$$

对参数w求偏导：

$$\frac{\partial L(\theta)}{\partial w} = \sum_{n=1}^{N} \frac{\partial C^n(\theta)}{\partial w}$$

因此我们只需要求出单个样例的偏导数，就可以推导出整体损失函数的偏导数。根据链式法则，对于某一个节点，如下所示：



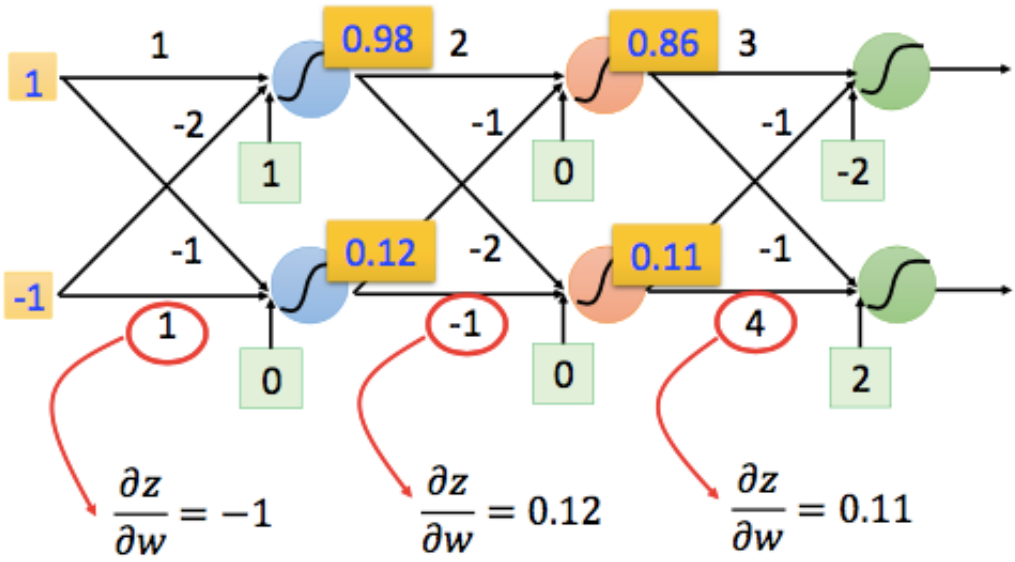
$$\frac{\partial C}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$$

容易得到

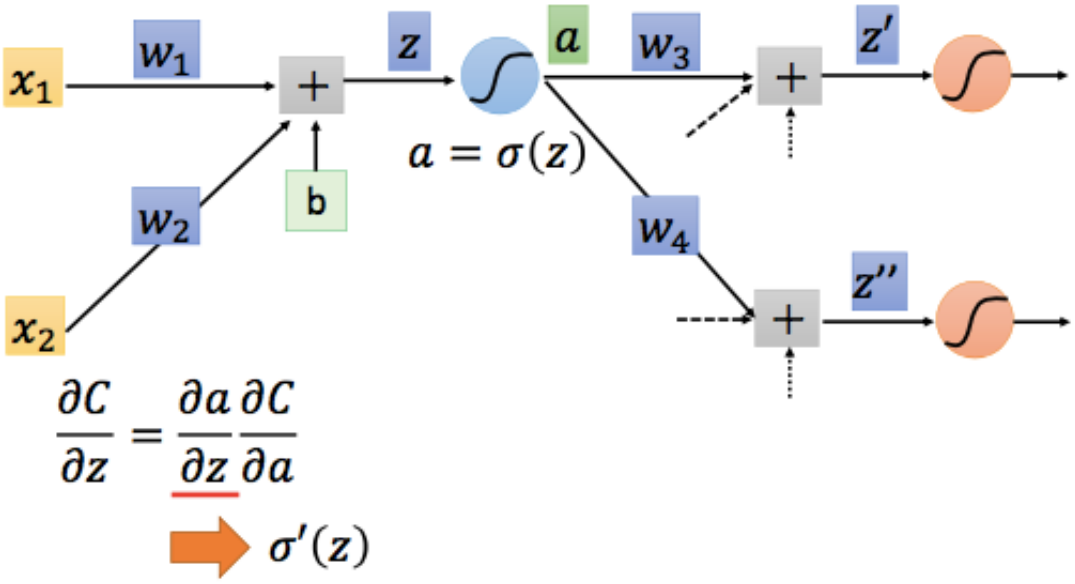
$$\partial z / \partial w_1 = x_1$$

$$\partial z / \partial w_2 = x_2$$

我们可以利用前向传导的方法计算出所有层的$\frac{\partial z}{\partial w}$



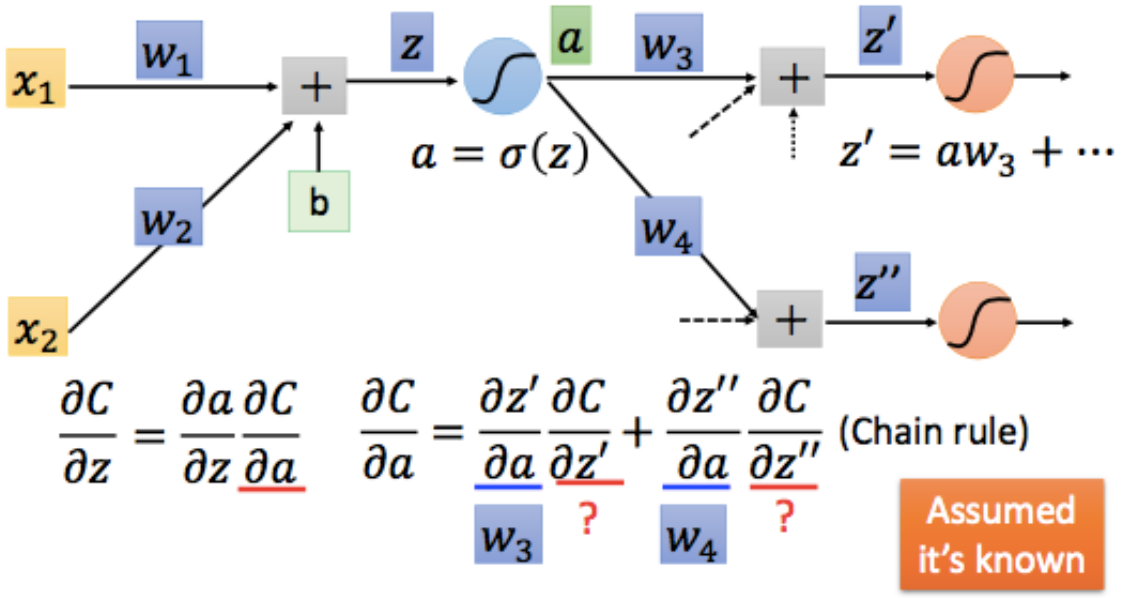
我们已经求出了整个偏导数的左半部分，接下来看右半部分，即$\frac{\partial C}{\partial z}$。



根据链式法则得到：

$$\frac{\partial C}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial C}{\partial a}$$

对于$\frac{\partial a}{\partial z}$，我们知道就是激活函数对加法器的偏导，知道了激活函数便知道了$\frac{\partial a}{\partial z}$，我们设其求导结果为$\partial'(z)$，因为$z$在前向传播中已经确定，所以$\partial'(z)$其实是一个常数。接下来看$\frac{\partial C}{\partial a}$



根据链式求导法则

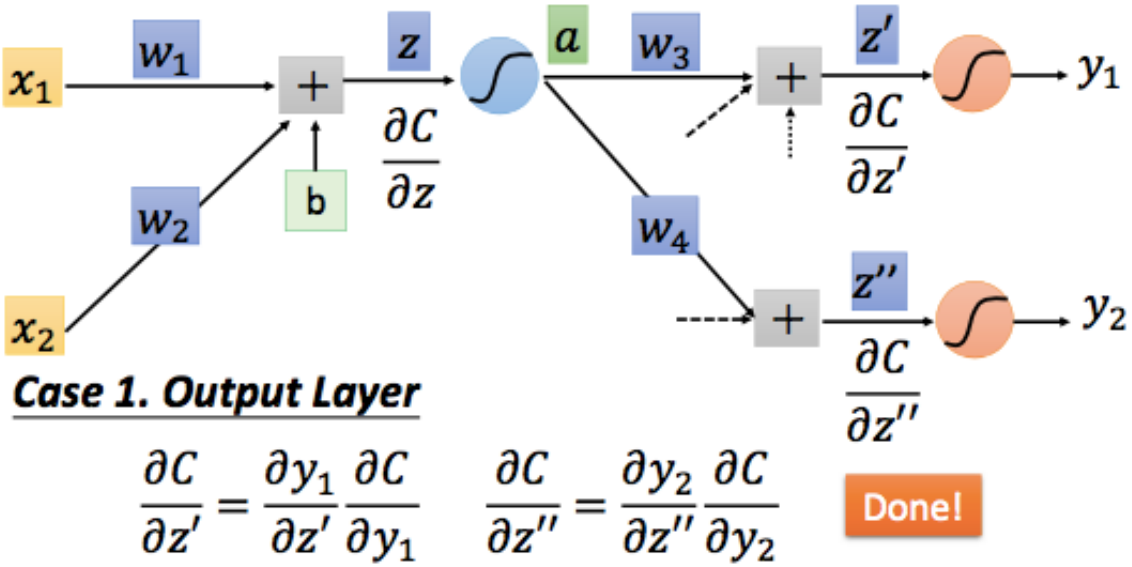
$$\frac{\partial C}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial C}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial C}{\partial z''}$$

易知$\frac{\partial z'}{\partial a}$即为权值，而$\frac{\partial C}{\partial z'}$假设其已知，则我们可以得到

$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3\frac{\partial C}{\partial z'} + w_4\frac{\partial C}{\partial z''}\right]$$

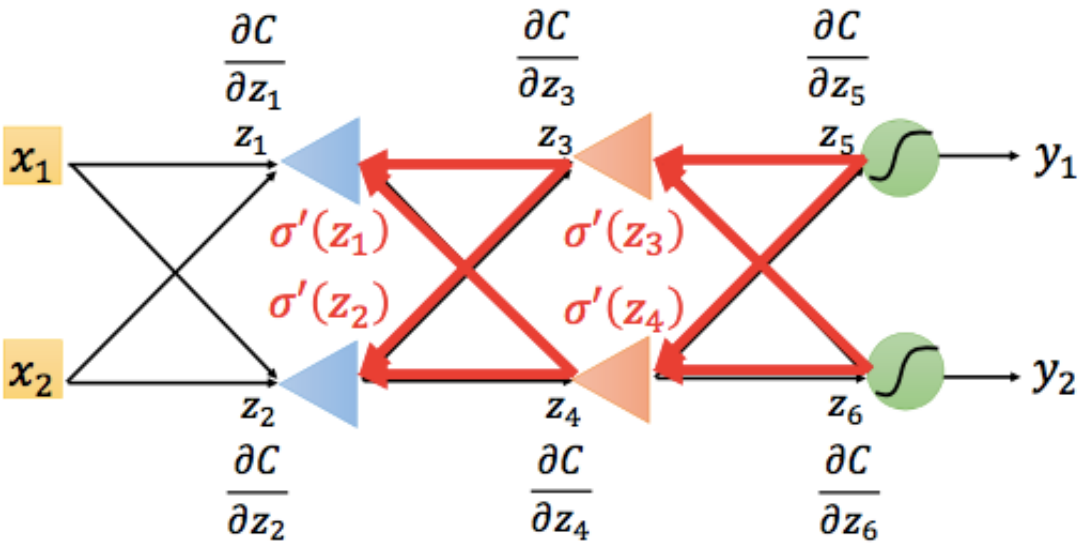
而对于$\frac{\partial C}{\partial z}$的求导，我们需要区分输出层和隐藏层两种情况：

第一种情况。如果已经是输出层了，如下图所示



我们可以直接求得。

第二种情况。如果还处于隐藏层，我们可以根据上述算法不断递归的计算$\partial C/\partial z$，直到抵达输出层。



最后总结一下，我们根据前向传播算法求得所有的$\frac{\partial z}{\partial w}$，根据反向传播算法求得所有的$\frac{\partial C}{\partial z}$（需要用到前向传播算法求得的$\frac{\partial a}{\partial z}$，即$\sigma'(z)$）。这样就可以用更新公式对参数进行迭代更新了。