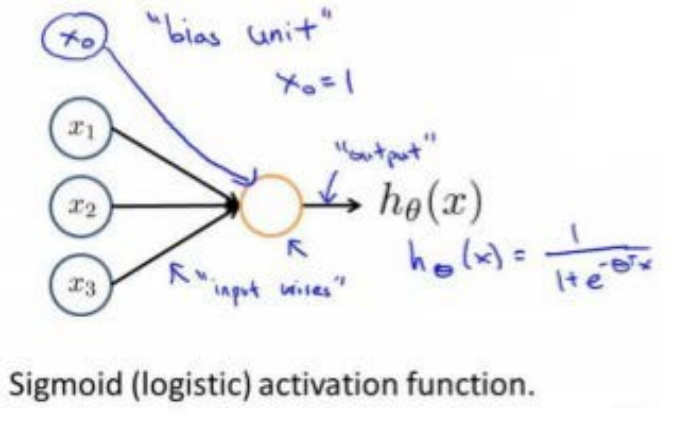
非线性假设

线性回归和逻辑回归都有一个缺点：特征太多时，计算负荷会很大。

假设有两个特征x_1, x_2，当我们使用x_1, x_2 的多次项式进行预测时，假设函数可能是$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta 3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 \dots)$。如果我们有很多个特征，用他们来构建一个非线性的多项式模型，其特征组合数量会很多。这对于一般的逻辑回归来说需要计算的特征太多了。

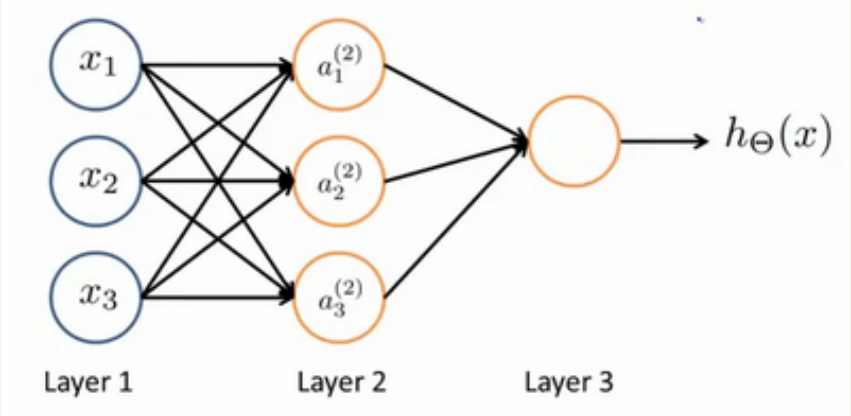
普通的逻辑回归模型，不能有效地处理这么多的特征，这时候我们需要神经网络。

模型表示



一个激活单元接收多个输入，也就是x，然后经过处理输出一些值。

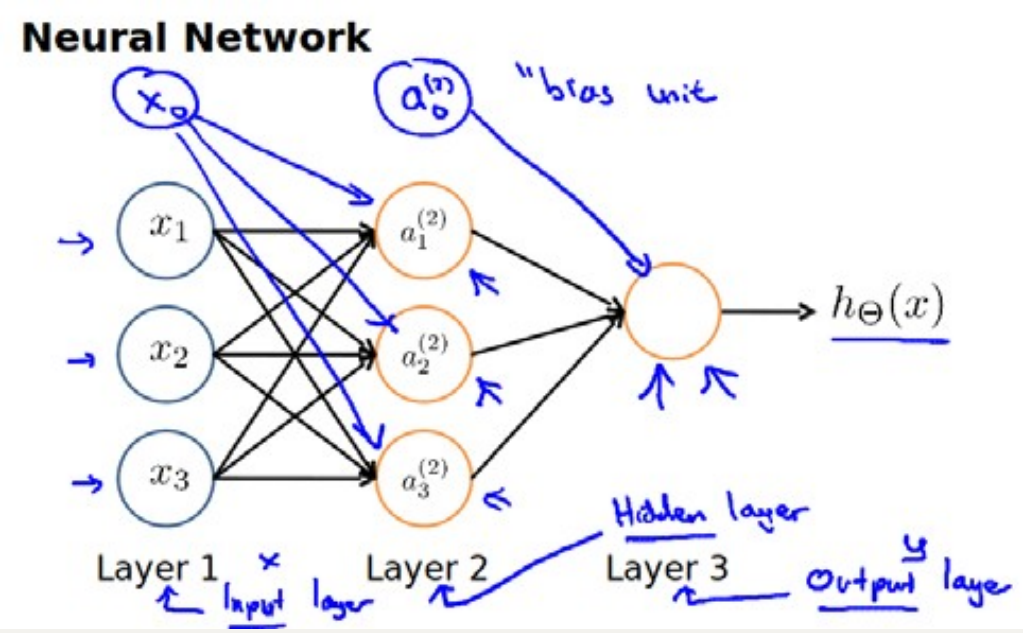
> *在神经网络中，参数θ又可被成为权重（**weight**）。*



其中x_1, x_2, x_3是输入单元（**input units**），我们将原始数据输入给它们。 a_1, a_2, a_3是中间单元，它们负责将数据进行处理，然后呈递到下一层。 最后是输出单元，它负责计算$h_\theta(x)$。

> *有时候会在输入层中添加x_0，他被叫做偏差单位(bias unit)*

神经网络模型是许多逻辑单元按照不同层级组织起来的网络，每一层的输出变量都是下一层的输入变量。



下面引入一些标记法来帮助描述模型：

- 每一条连线都对应一个θ
- $a_i^{(j)}$ 代表第j 层的第 i 个激活单元
- $\theta^{(j)}$代表从第 j 层映射到第$j+1$ 层时的权重的矩阵，例如$\theta^{(1)}$代表从第一层映射到第二层的权重的矩阵。其尺寸为：以第 $j+1$层的激活单元数量为行数，以第 j 层的激活单元数加一为列数的矩阵，也就是$s_{j+1} \times (s_j + 1)$。例如：上图所示的神经网络中$\theta^{(1)}$的尺寸为 3*4。

> *理解$\theta^{(1)}$的尺寸为3x4：x和a的之间的每一条连线都对应一个θ值，这些θ构成3x4矩阵，为什么是3x4？因为有4个输入变量x_0到x_3，下一层有3个变量a_1到a_3*

对于上图所示的模型，激活单元和输出分别表达为，Θ_{10}的下标代表x_0到a_1的映射：

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \; a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

把x, θ, a 分别用矩阵表示：

$$X = \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix}, \quad \theta = \begin{matrix} \theta_{10} & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \theta_{33} \end{matrix}, \quad a = \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix}$$

我们可以得到$\theta \cdot X = a$ 。

向量化表示

我们把$\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$写成$z_1^{(2)}$

> *注：这里上标2表示这些值和第2层有关，1表示第一行*
>
> *z为输入，a为输出，$z^{(0)} = x$*

$a_1^{(2)} = g(z_1^{(2)})$

再写成向量形式：

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad z^{(2)} = \begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{pmatrix}$$

可以写成矩阵相乘：

$z^{(2)} = \Theta^{(1)} x$

其中：

$$\Theta^{(1)} = \begin{pmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{pmatrix}$$

$a^{(2)} = g(z^{(2)})$

$$g\left(\begin{bmatrix}\theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)}\\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)}\\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)}\end{bmatrix}\times\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}\right)=g\left(\begin{bmatrix}\theta_{10}^{(1)}x_0+\theta_{11}^{(1)}x_1+\theta_{12}^{(1)}x_2+\theta_{13}^{(1)}x_3\\ \theta_{20}^{(1)}x_0+\theta_{21}^{(1)}x_1+\theta_{22}^{(1)}x_2+\theta_{23}^{(1)}x_3\\ \theta_{30}^{(1)}x_0+\theta_{31}^{(1)}x_1+\theta_{32}^{(1)}x_2+\theta_{33}^{(1)}x_3\end{bmatrix}\right)=\begin{bmatrix}a_1^{(2)}\\a_2^{(2)}\\a_3^{(2)}\end{bmatrix}$$

我们令 $z^{(2)}=\theta^{(1)}x$，则 $a^{(2)}=g(z^{(2)})$ ，计算后添加 $a_0^{(2)}=1$。 计算输出的值为：

$$g\left(\begin{bmatrix}\theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)}\end{bmatrix}\times\begin{bmatrix}a_0\\a_1^{(2)}\\a_2^{(2)}\\a_3^{(2)}\end{bmatrix}\right)=g\left(\theta_{10}^{(2)}a_0^{(2)}+\theta_{11}^{(2)}a_1^{(2)}+\theta_{12}^{(2)}a_2^{(2)}+\theta_{13}^{(2)}a_3^{(2)}\right)=h_\theta(x)$$

这只是针对训练集中一个训练实例所进行的计算。如果我们要对整个训练集进行计算，我们需要将训练集特征矩阵进行转置，使得同一个实例的特征都在同一列里。即：
$z^{(2)}=\Theta^{(1)}\times X^T$

$a^{(2)}=g(z^{(2)})$

其实神经网络就像是**logistic regression**，只不过我们把**logistic regression**中的输入向量 $[x_1\sim x_3]$ 变成了中间层的$\left[a_1^{(2)}\sim a_3^{(2)}\right]$, 即：
$h_\theta(x)=g\left(\Theta_0^{(2)}a_0^{(2)}+\Theta_1^{(2)}a_1^{(2)}+\Theta_2^{(2)}a_2^{(2)}+\Theta_3^{(2)}a_3^{(2)}\right)$ 我们可以把a_0,a_1,a_2,a_3看成更为高级的特征值，也就是x_0,x_1,x_2,x_3的进化体，并且它们是由 x与θ决定的，因为是梯度下降的，所以a是变化的，并且变得越来越厉害，所以这些更高级的特征值远比仅仅将 x次方厉害，也能更好的预测新数据。 这就是神经网络相比于逻辑回归和线性回归的优势。

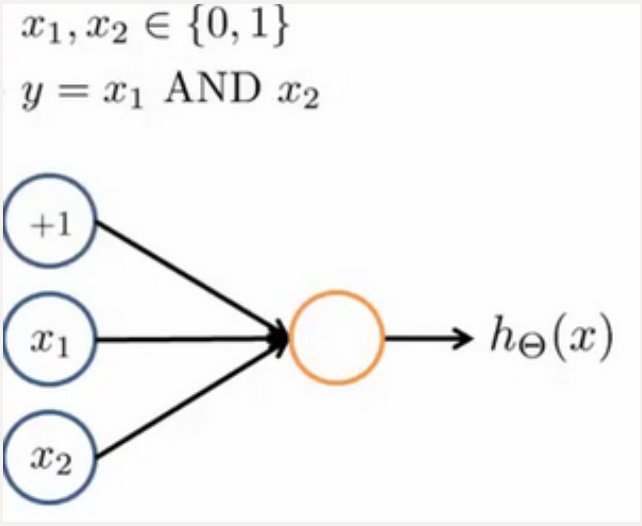
上面进行的讨论中只是将特征矩阵中的一行（一个训练实例）喂给了神经网络，我们需要将整个训练集都喂给我们的神经网络算法来学习模型。

我们可以知道：每一个a都是由上一层所有的x和每一个x所对应的决定的。

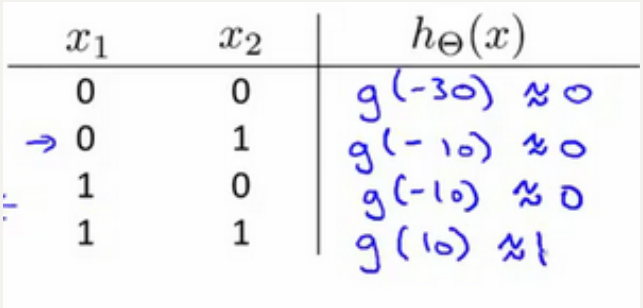
（我们把这样从左到右的算法称为前向传播算法(**FORWARD PROPAGATION**)）

举例理解

当$\theta_0 = -30, \theta_1 = 20, \theta_2 = 20$时， 下面这个例子表示AND函数



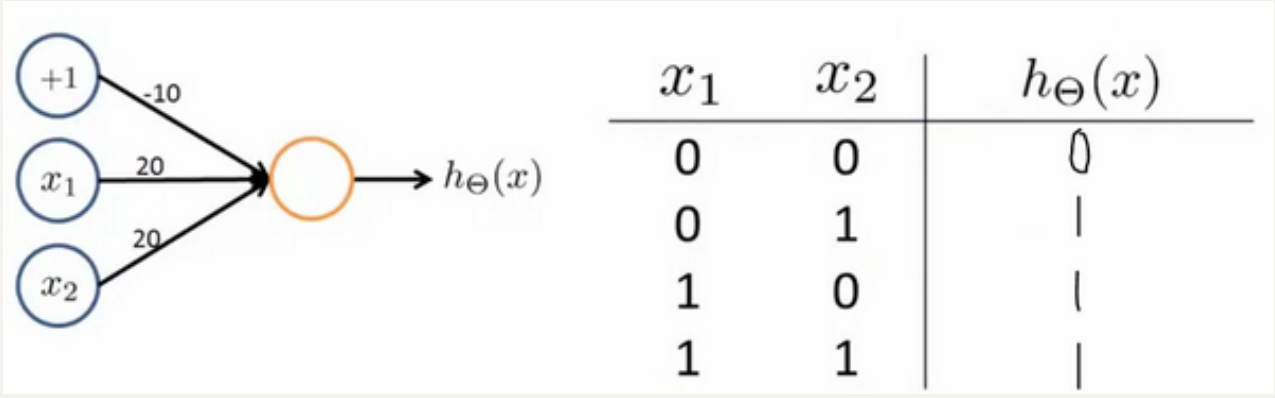
$h_\theta(x)$即为：$h_\Theta(x) = g\left(-30 + 20x_1 + 20x_2\right)$



所以我们有：$h_\Theta(x) \approx \mathrm{x}_1 \mathrm{AND}\, \mathrm{x}_2$

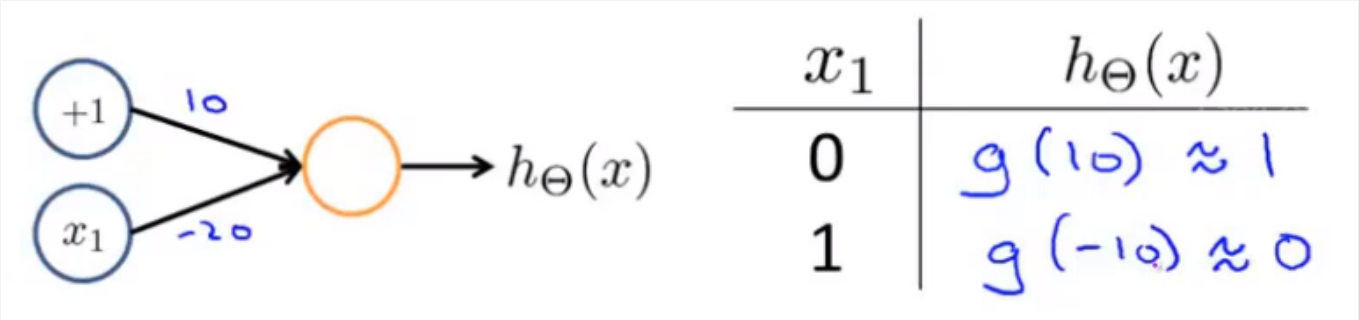
所以我们的：$h_\Theta(x)$

OR函数的实现方法就是将θ取值改为$(-10, 20, 20)$



x_1	x_2	$h_\Theta(x)$
0	0	0
0	1	1
1	0	1
1	1	1

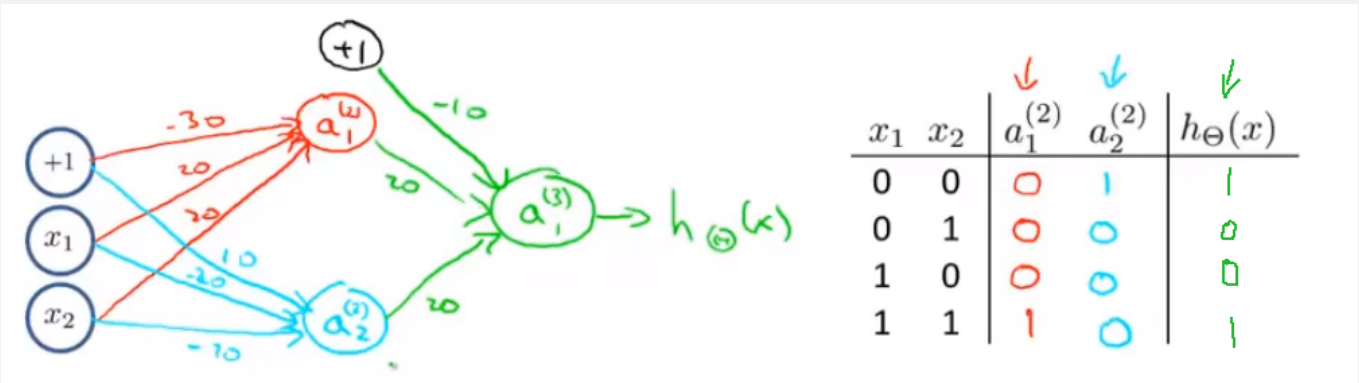
下图的神经元（两个权重分别为 10，-20）可以被视为作用等同于逻辑非（**NOT**）：



x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

现在我们有了与或非三个 函数，那么我们就能用他们来组成更复杂的函数。

例如我们要实现**XNOR** 功能（输入的两个值必须一样，均为1或均为0），即
$\text{XNOR} = (\text{x}_1 \text{ AND } \text{x}_2) \text{ OR } ((\text{NOT } \text{x}_1) \text{ AND } (\text{NOT } \text{x}_2))$



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

这张图中红色节点的参数为(-30, 20, 20)他是用来计算$((\text{NOT } \text{x}_1) \text{ AND } (\text{NOT } \text{x}_2))$，蓝色节点的参数为(10, -20, -10)，它是我们之前提到的AND函数，最后将他们两个的结果输入到绿色节点，也就是经过OR运算，得到最后的假设函数。

> *在第2层加上了个偏差单位(bias unit) +1。*

我们就得到了一个能实现 XNOR 运算符功能的神经网络。

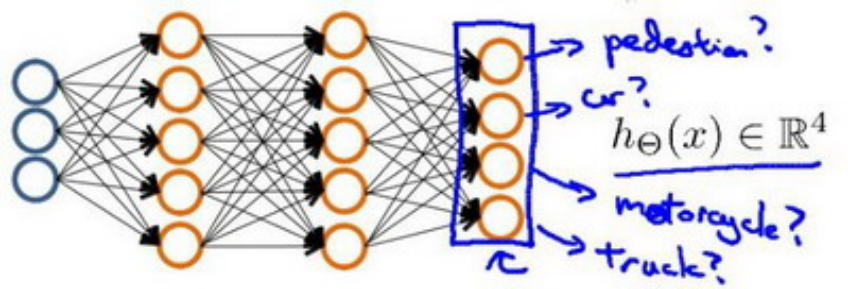
按这种方法我们可以逐渐构造出越来越复杂的函数，也能得到更加厉害的特征值。

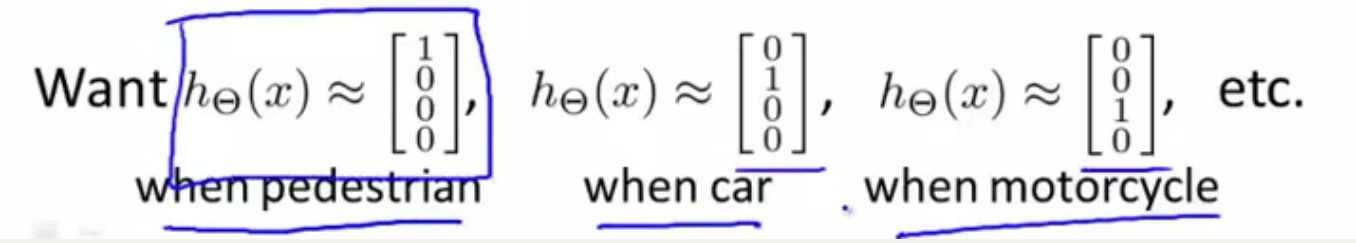
这就是神经网络的厉害之处。

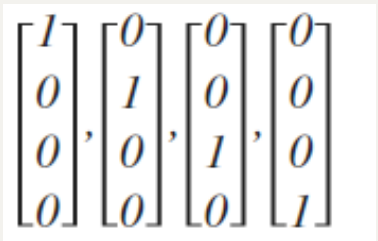
多分类

当我们有不止两种分类时（也就是$y = 1, 2, 3 \ldots.$），比如以下这种情况，该怎么办？如果我们要训练一个神经网络算法来识别路人、汽车、摩托车和卡车，在输出层我们应该有4个值。例如，第一个值为1或0用于预测是否是行人，第二个值用于判断是否为汽车。

输入向量x有三个维度，两个中间层，输出层4个神经元分别用来表示4类，也就是每一个数据在输出层都会出现$[a\ b\ c\ d]^T$，且a, b, c, d中仅有一个为1，表示当前类。下面是该神经网络的可能结构示例：







代价函数

首先引入一些便于稍后讨论的新标记方法：

假设神经网络的训练样本有m个，每个包含一组输入x和一组输出信号y，L表示神经网络层数，S_I表示每层的**neuron**个数(S_l表示输出层神经元个数)，S_L代表最后一层中处理单元的个数。

将神经网络的分类定义为两种情况：二类分类和多类分类，

二类分类：$S_L=0, y=0\, or\, 1$表示哪一类；

K类分类：$S_L=k, y_i=1$表示分到第i类；$(k>2)$

逻辑回归问题中我们的代价函数为：

$$J\left(\theta\right)=-\frac{1}{m}\left[\sum_{i=1}^{m}y^{(i)}\log h_\theta(x^{(i)})+\left(1-y^{(i)}\right)log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]+\frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2$$

在逻辑回归中，我们只有一个输出变量，又称标量（**scalar**），也只有一个因变量y，但是在神经网络中，我们可以有很多输出变量，我们的$h_\theta(x)$是一个维度为K的向量，并且我们训练集中的因变量也是同样维度的一个向量，因此我们的代价函数会比逻辑回归更加复杂一些，为：$h_\theta\left(x\right)\in\mathbb{R}^K\ \left(h_\theta\left(x\right)\right)_i=i^{th}\text{output}$

$$J(\Theta)=-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{k}{y_k}^{(i)}\log(h_\Theta(x^{(i)}))_k+\left(1-y_k^{(i)}\right)\log\left(1-\left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right]+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

这个看起来复杂很多的代价函数背后的思想还是一样的，我们希望通过代价函数来观察算法预测的结果与真实情况的误差有多大，唯一不同的是，**对于每一行特征，我们都会给出K个预测**，基本上我们可以利用循环，对每一行特征都预测K个不同结果，然后在利用循环在K个预测中选择可能性最高的一个，将其与y中的实际数据进行比较。

正则化的那一项只是排除了每一层θ_0后，每一层的θ **矩阵的和**。最里层的循环j循环所有的行（由s_{l+1} 层的激活单元数决定），循环i则循环所有的列，由该层（s_l层）的激活单元数所决定。即：$h_\theta(x)$与真实值之间的距离为每个样本-每个类输出的加和，对参数进行**regularization**的**bias**项处理所有参数的平方和。

> *可以将相邻两层间的θ参数堪称一个矩阵，ij分别代表行列，矩阵大小为$s_{l+1}xs_l$，正则项就是对矩阵每一项求和。代价函数希望结果最小，也就是该网络K个输出单元的预测值都最小，因此需要累加每个输出单元的结果，使其总和最小。*

反向传播算法

> *这部分比较难，看这篇文章容易理解，通俗理解神经网络BP传播算法*

我的理解是，反向传播算法就是梯度下降算法，只不过用到了链式求导法则而已。

我们从最后一层的误差开始计算，误差是激活单元的预测（$a^{(4)}$）与实际值（y^k）之间的误差，（$k=1:k$）。我们用δ来表示误差，则：$\delta^{(4)}=a^{(4)}-y$ 我们利用这个误差值来计算前一层的误差：$\delta^{(3)}=\left(\Theta^{(3)}\right)^T\delta^{(4)}*g'\left(z^{(3)}\right)$ 其中 $g'(z^{(3)})$是 S 形函数的导数，$g'(z^{(3)})=a^{(3)}*(1-a^{(3)})$。而$(\Theta^{(3)})^T\delta^{(4)}$则是权重导致的误差的和。下一步是继续计算第二层的误差：$\delta^{(2)}=(\Theta^{(2)})^T\delta^{(3)}*g'(z^{(2)})$ 因为第一层是输入变量，不存在误差。我们有了所有的误差的表达式后，便可以计算代价函数的偏导数了，假设$\lambda=0$，即我们不做任何正则化处理时有：$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)=a_j^{(l)}\delta_i^{l+1}$

l 代表目前所计算的是第几层。

j 代表目前计算层中的激活单元的下标，也将是下一层的第j个输入变量的下标。

i 代表下一层中误差单元的下标，是受到权重矩阵中第i行影响的下一层中的误差单元的下标。

如果我们考虑正则化处理，并且我们的训练集是一个特征矩阵而非列向量。在上面的特殊情况中，我们需要计算每一层的误差单元来计算代价函数的偏导数。在更为一般的情况中，我们同样需要计算每一层的误差单元，但是我们需要为整个训练集计算误差单元，此时的误差单元也是一个矩阵，我们用$\Delta_{ij}^{(l)}$来表示这个误差矩阵。第 l 层的第 i 个激活单元受到第 j 个参数影响而导致的误差。

算法如下：

for i=1:m // 遍历训练集 $set\quad a^{(i)}=x^{(i)}$ 执行正向传播算法计算$a^{(l)}$ for l=1, 2, 3, ..., L 使用 $\delta^{(L)}=a^{(L)}-y^i$ 执行反向传播算法计算所有前一层的误差向量$\Delta_{ij}^{(l)}=\Delta_{ij}^{(l)}+a_j^{(l)}\delta_i^{l+l}$

即首先用正向传播方法计算出每一层的激活单元$a^{(l)}$，利用训练集的结果与神经网络预测的结果求出最后一层的误差，然后利用该误差运用反向传播法计算出直至第二层的所有误差。

在求出了$\Delta_{ij}^{(l)}$之后，我们便可以计算**代价函数的偏导数**了，计算方法如下：
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} \; if \; j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} \; if \; j = 0$