# 17.4.21（反向传播算法）

神经网络

```
%
s = sigmoid(X*(all_theta'));

[maxnum, maxindex] = max(s, [], 2);

p = maxindex;
% =========================================
```

一对多类，分类问题，其中的max函数是计算每一行的最大值，并返回该最大值，及其对应的下标。

一对多类，是分成N个二分类问题，求出结果看哪个类对应的假设函数最大，预测值对应的就是这个类。

```
s1 = sigmoid(X*(Theta1'));
s1 = [ones(size(s1,1), 1) s1];
s2 = sigmoid(s1*(Theta2'));

[~, maxindex] = max(s2, [], 2);

p = maxindex;
```

简单神经网络算法，就是套了两层sigmoid函数。

神经网络的cost function

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

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。

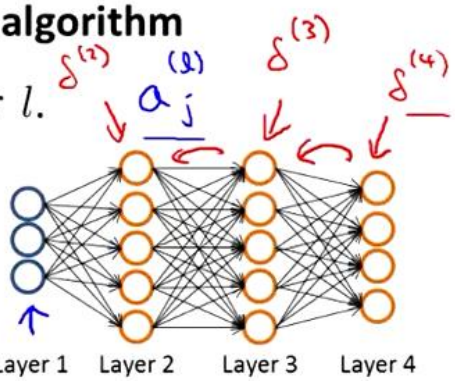
上图的代价函数就是把隐藏层中所有的激励函数和其对应的theta的cost function的加和。

Delte 反向传播算法

cost function 的偏导数 就是 激励函数的激励值（输出）与delte的乘积。

## Gradient computation: Backpropagation algorithm

Intuition: $\delta_j^{(l)}$ = "error" of node $j$ in layer $l$.




For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

(No $\delta^{(1)}$)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

(ignoring $\lambda$; if $\lambda = 0$)

补充：

$$g'(z) = \frac{d}{dz} g(z) = g(z)(1 - g(z))$$

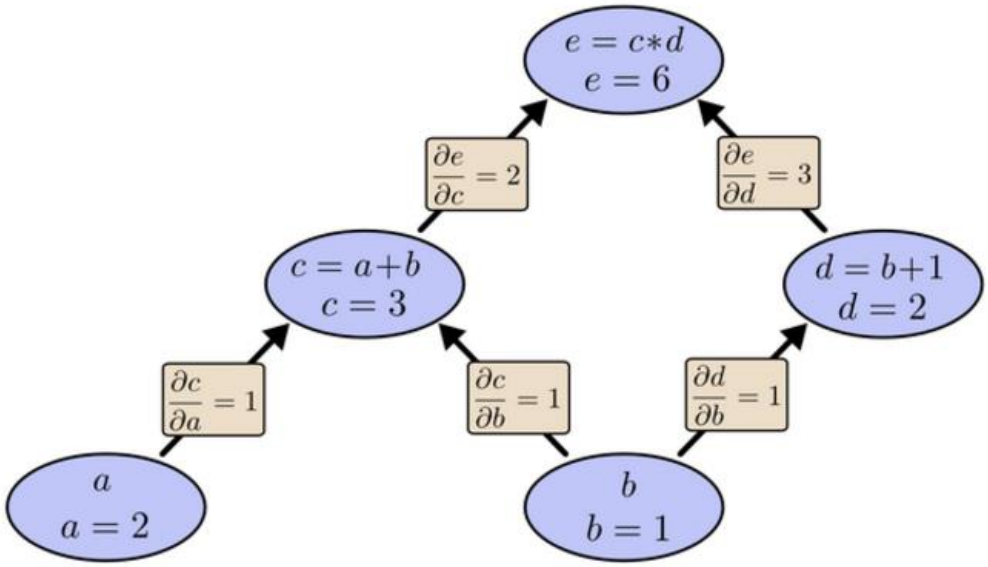
$$\text{sigmoid}(z) = g(z) = \frac{1}{1 + e^{-z}}.$$

神经网络求出cost function 和 grad ，然后用梯度下降来修改theta，使cost function越来越小。

delte 误差就是从最后一层，输出层开始，用预测值与实际值的误差往前反方向推上一层的sigmoid函数，也就是激励函数求得的激励值与实际值的误差，（用以前的方法就是说假设函数求得的预测值与实际值的误差），从输出层一点点的反向传播，求得每一层的误差，然后才能一点点的修正，更好的拟合训练集，更好的预测。和梯度下降算法中一点点的到达最优值一样，反向传播算法是一种更好的优化我们的model的一种算法，强大到可以优化很复杂的model（就是其中有多层函数的model）。

具体实现如上图中，l-1层的delte就等于本层的theta（权重矩阵）的转置乘 l 层的delte，然后元素乘g(z(l))的导数（ z 就是就是本层的输入，g(z)就是本层的激励函数，本项就等于本层输出值点乘（1-本层输出值））。

为了求出a=2, b=1时，e的梯度，我们可以先利用偏导数的定义求出不同层之间相邻节点的偏导关系，如下图所示。




利用链式法则我们知道：

$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a}$ 以及 $\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$。

链式法则在上图中的意义是什么呢？其实不难发现，$\frac{\partial e}{\partial a}$的值等于从a到e的路径上的偏导值的乘积，而$\frac{\partial e}{\partial b}$的值等于从b到e的路径1(b-c-e)上的偏导值的乘积加上路径2(b-d-e)上的偏导值的乘积。也就是说，对于上层节点p和下层节点q，要求得$\frac{\partial p}{\partial q}$，需要找到从q节点到p节点的所有路径，并且对每条路径，求得该路径上的所有偏导数之乘积，然后将所有路径的 "乘积" 累加起来才能得到$\frac{\partial p}{\partial q}$的值。