1、参数值θ计算

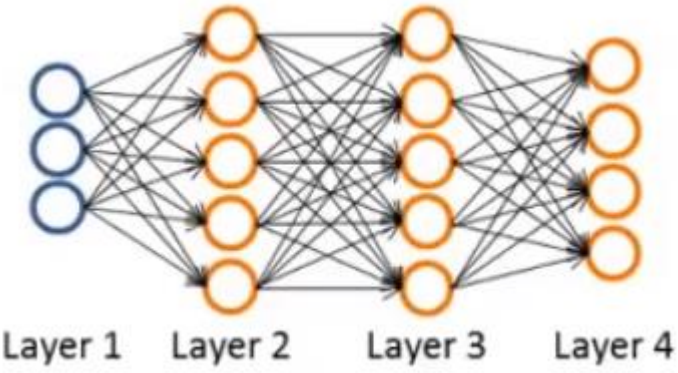
在讲了神经网络的多分类问题以及代价函数计算公式后，接下来就是要利用梯度下降算法或是高等梯度下降算法进行参数值的计算了。这里采用第二种方式，即高等梯度下降算法。

首先得手动编写一个 costFunction 函数，返回代价值和当前偏导数。

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

$\frac{\lambda}{2m}\sum_{L=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ji}^{(l)}\right)^2$，这是之前的代价函数计算公式。

求偏导数，这里由于中间有很多隐藏层，因此不可能同时将所有的层的权重矩阵计算出来，所以这里引入反向传播的概念，即通过最后一层输出层，反向的将之前一直到第 2 层的偏导数求出来。如果层数一共有 L 层那么权重矩阵的个数为 L-1 个，因此要迭代 L-1 次，才可以将代价值和偏导数求出来。这边完成了高等梯度下降算法中第一次计算。



这是一个含有 4 层的神经网络，给定一个训练样本(x，y)，根据前向传播的知识我们可以得到

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$

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

这对应的是输出层第 j 个输出与真实结果中第 j 个的误差，同样可以利用向量化的思想一次性计算出第 4 层的误差值。利用反向传播的思想，我们可以得到第 2 层和第 3

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

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

层误差向量。

可以证明，$\frac{\partial}{\partial \Theta_{ij}^{(L)}} J(\Theta) = a_j^{(L)} \delta_i^{(L+1)}$即为第 L 层权重矩阵中第 i 个输出中的第 j 个参数偏导数。这样一来，所有的偏导数和代价值都求出来了，可以使用高等梯度下降算法计算每次迭代后的参数θ了。

这只是针对只有一个输入项和多个输出项所计算出来的偏导数，如果有多个输入该怎么办呢？我们可以考虑将 m 个输入作为迭代次数，每次计算出来每个$a_j^{(L)} \delta_i^{(L+1)}$，然后对应项相加，最后计算总的偏导数，这里分两种情况：

$$D_{ij}^{(l)} = \frac{1}{m} a_j^{(l)} \delta_i^{(l+1)} \qquad \text{if j} = 0$$

$$D_{ij}^{(l)} = \frac{1}{m} a_j^{(l)} \delta_i^{(l+1)} + \lambda \Theta_{ij}^{(l)} \qquad \text{if j} \neq 0$$

2、高等梯度下降算法

到目前为止，我们已经计算出了代价值和偏导数，不过我们第一次传入 costFuntion 和迭代过程形成的偏导数都是以矩阵的形式来表达的，在高等梯度下降算法中要求必须使用向量的形式，这就涉及到一个矩阵与向量之间的转化。

先来看一个例子，现有 3 个权重矩阵，分别为 A=ones（10，11）、

B=2*ones（10，11）、C=3*ones（1，11），要求在传入 costFunction 时为向量化的形式，可以使用[A(:);B(:);C(:)]转化为向量形式，然后再 costFunction 中，由于要利用正向传播和反向传播计算 z、a、δ，所以用矩阵形式更方便，因此再利用 reshape(theta(1,110),10,11)形成矩阵，最后算完偏导数后，再转化为向量形式输出偏导数结果。这样，fimunc 函数才可正常进行。

3、梯度检验

再利用 a、δ计算偏导数时，难免会出现一些 bug，这取决与个人实现求偏导数时的算法有没有问题。检验的方法是利用双侧数值计算求偏导数，最后在一次迭代完成以后，检验双方是不是近似相等，否则认为前者计算θ出现了问题，需要修改。

```
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                    /(2*EPSILON);
end;
```

需说明一点的是，数值计算偏导是一个漫长的过程，如果每次都要检验，则θ向量有多少个元素就要检验多少次，再乘以总督迭代次数，可以想象，非常耗时。好的做法是，当检验几次成功以后，在下一次运行代码时，就应关掉检验程序。

4、随机初始化

无论是在梯度下降算法还是在高等梯度下降算法中，都需要提供一个初始的θ值，在神经网络这部分中，我么要提供 L-1 层的权重矩阵，如果还想以前那样所有的θ值都初始化为 0，那么结果就是所有的权重矩阵中每一行所有的θ值相同。因为初始化为 0 后，所计算得到的每一层 a 值中各个单元值是一样的，因此在利用$\frac{\partial}{\partial\Theta_{ij}^{(L)}}J(\Theta)=a_j^{(L)}\delta_i^{(L+1)}$是，结果就是一样的。试想一下，到最后迭代完成以后，所有的高级特征都是线性相关的，那么意义何在呢，使用神经网络预测模型，就是要将起初简答的特征值越来越复杂化，抽象化，因此每个权重矩阵的每一行都应保持不同。

因此这里引入了随机初始化的概念，首先另$\varepsilon\in[-\epsilon,+\epsilon]$，$\epsilon$接近于0，然后初始化权重矩阵，使用 rand(10,11)*(2*ε)- ε，接下来，就可以使用高等梯度下降算法进行求解了。