### 9.4 实现注意：展开参数

参考视频: 9 - 4 - Implementation Note\_ Unrolling Parameters (8 min).mkv

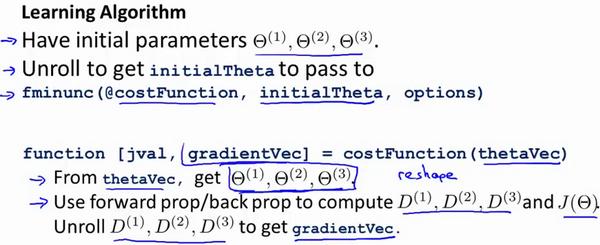
在上一段视频中，我们谈到了怎样使用反向传播算法计算代价函数的导数。在这段视频中，我想快速地向你介绍一个细节的实现过程，怎样把你的参数从矩阵展开成向量，以便我们在高级最优化步骤中的使用需要。

图片包含 屏幕截图

描述已自动生成

图片包含 文字

描述已自动生成



### 9.5 梯度检验

参考视频: 9 - 5 - Gradient Checking (12 min).mkv

当我们对一个较为复杂的模型（例如神经网络）使用梯度下降算法时，可能会存在一些不容易察觉的错误，意味着，**虽然代价看上去在不断减小，但最终的结果可能并不是最优解。**

为了避免这样的问题，我们采取一种叫做**梯度的数值检验（Numerical Gradient Checking）**方法。这种方法的思想是通过估计梯度值来检验我们计算的导数值是否真的是我们要求的。

对梯度的估计采用的方法是在代价函数上沿着切线的方向选择离两个非常近的点然后计算两个点的平均值用以估计梯度。**即对于某个特定的 ，我们计算出在 - 处和 + 的代价值（是一个非常小的值，通常选取 0.001），然后求两个代价的平均，用以估计在 处的代价值。**

图片包含 地图, 文字

描述已自动生成

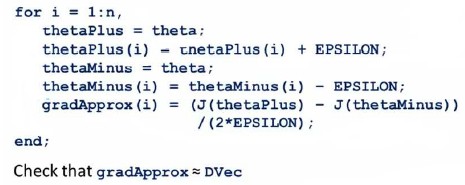
**Octave** 中代码如下：

gradApprox = (J(theta + eps) – J(theta - eps)) / (2\*eps)

**当是一个向量时，我们则需要对偏导数进行检验。**因为代价函数的偏导数检验只针对一个参数的改变进行检验，下面是一个只针对进行检验的示例：

最后我们还需要对通过反向传播方法计算出的偏导数进行检验。

根据上面的算法，计算出的偏导数存储在矩阵 中。检验时，我们要将该矩阵展开成为向量，同时我们也将 矩阵展开为向量，我们针对每一个 都计算一个近似的梯度值，将这些值存储于一个近似梯度矩阵中，最终将得出的这个矩阵同 进行比较。



### 9.6 随机初始化

参考视频: 9 - 6 - Random Initialization (7 min).mkv

任何优化算法都需要一些初始的参数。到目前为止我们都是初始所有参数为0，这样的初始方法对于逻辑回归来说是可行的，但是对于神经网络来说是不可行的。如果我们令所有的初始参数都为0，这将意味着我们第二层的所有激活单元都会有相同的值。同理，如果我们初始所有的参数都为一个非0的数，结果也是一样的。

我们通常初始参数为正负之间的随机值，假设我们要随机初始一个尺寸为10×11的参数矩阵，代码如下：

Theta1 = rand(10, 11) \* (2\*eps) – eps

### 9.7 综合起来

参考视频: 9 - 7 - Putting It Together (14 min).mkv

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图, 文字

描述已自动生成

小结一下使用神经网络时的步骤：

**网络结构：**第一件要做的事是选择网络结构，即决定选择多少层以及决定每层分别有多少个单元。

第一层的单元数即我们训练集的特征数量。

最后一层的单元数是我们训练集的结果的类的数量。

如果隐藏层数大于1，确保每个隐藏层的单元个数相同，通常情况下隐藏层单元的个数越多越好。

我们真正要决定的是隐藏层的层数和每个中间层的单元数。

**训练神经网络：**

1. 参数的随机初始化
2. 利用正向传播方法计算所有的
3. 编写计算代价函数 的代码
4. 利用反向传播方法计算所有偏导数
5. 利用数值检验方法检验这些偏导数
6. 使用优化算法来最小化代价函数
7. 图片包含 地图, 文字

   描述已自动生成