实验报告-ex4

编程软件：MATLAB，notepad++

编程环境：Windows10

作业任务：本次实验旨在了解用神经网络算法中的back-propagation算法处理one-vs-all logistic回归的问题，解决识别手写数字的实例问题。有ex4一个函数，函数中套用了相应的函数，所以需要根据这两个目录函数的指示和实验手册，完成对应的5个函数。

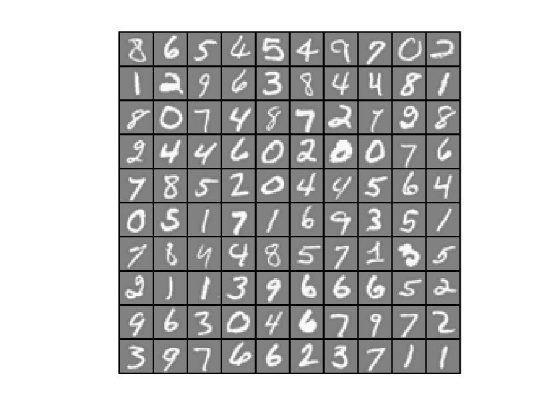
1 Neural Networks

这次实验的目的和上次一样：识别0-9的手写体数字。只是这次实验利用了back-prop算法来自行习得参数Θ，而不是已知。

1.1 Visualizing the data

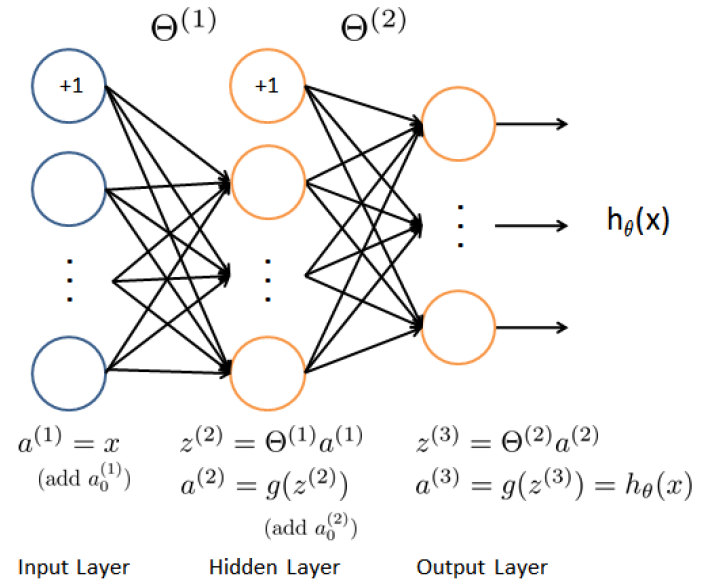
这一部分与ex3的training set一样。图像的像素是20×20，每个像素由一个浮点数来表示该位置的灰度。所以这个20×20的像素就转化为1×400的向量，而每一行代表一个training example，所以得到的X是一个5000×400的矩阵。结果向量y就是一个5000×1的向量，每个元素代表该手写数字的值。

之后调用displayData函数，画出原来图像的样子。（该函数已给出）



1.2 Model representation

本次实验的神经网络架构只用到了3层，即input、hidden、output，结构如下：

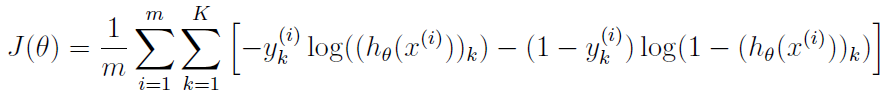


第二层的hidden是有25个单元，输出为10个单元（恰好对应0 ~ 9十个数字）

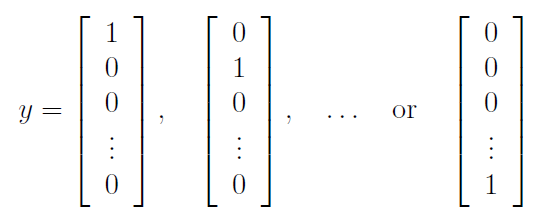
1.3 Feedforward and cost function

这一部分需要完成cost function和gradient两个重要式子。

首先是cost function，回顾其在神经网络中的式子如下：

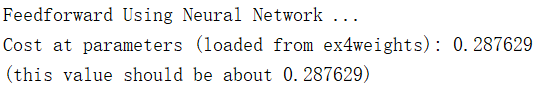


其中K是结果层的个数。另外需要注意，输出的每个y不再是0-9这十个数字，而应该转化成向量。故原来的向量y应该转化成m×K的矩阵。



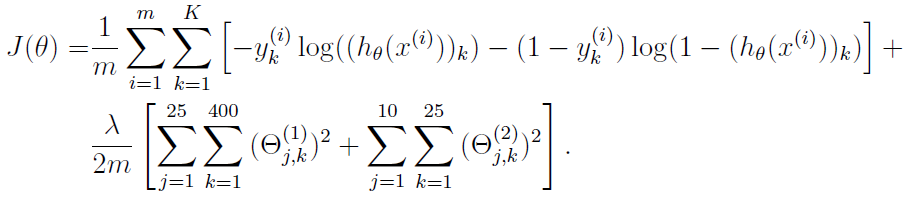
需要注意，传入的Θ是经过unroll之后的向量，所以在进行向量化运算前，需要先将其转化为矩阵（代码中已经给出）。与之类似，返回的grad也应该是unroll之后的向量，而代码得到的直接结果是矩阵，所以需要通过unroll将其转换（代码也已给出）。

结果正确：

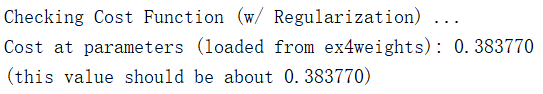


1.4 Regularized cost function

这里在上一部分的基础上加上了正则化部分，公式为：



需要注意的是Θ(1)和Θ(2)的第一列，即bias不需要进行正则化。类似于ex3中的处理方法，将两个Θ的第一列全部置零。剩下可直接在不带正则化的结果J后面加上两个Θ的所有元素平方的和。

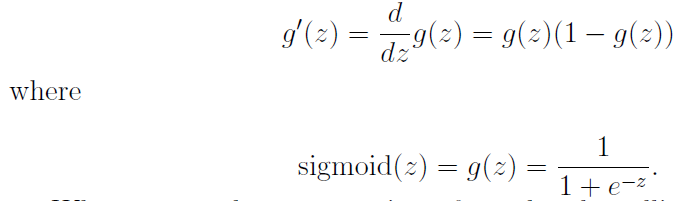


2 Backpropagation

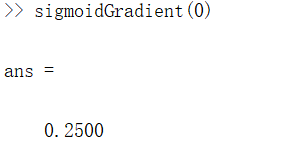
上一大部分只运用了前面向前传播的知识计算了cost function。在这一大部分，需要利用向后传播的知识计算gradient，然后就可以用fmincg计算合适的parameter了。

2.1 Sigmoid gradient

为了在后面更简便地计算gradient，可实现计算出所需要的sigmoid函数的导数的表达式。



需要完善sigmoidGradient(z)函数，这个函数的实现较为简单，直接利用已有公式sigmoid带入计算即可。



2.2 Random initialization

在神经网络部分，为了防止symmetry问题的产生，需要对parameter进行随机初始化。方法就是对已知维度的Θ进行随机赋值，此时的值范围为[-1,1]。我们要做的就是将其限制在[-ε, ε]。在这里，我们令ε=0.12.

2.3 Backpropagation

现在要做的就是用反向传播实现gradient的计算了。一共分5步实现，其中前4步要在1:m的循环中进行，计算Δ。最后一步才是在循环外计算gradient。

1. 利用向前传播算法计算每一层的z和a。注意a=g(z)。需要注意的是，要根据当前a(l)计算a(l+1)时，要先加上bias。

2. 计算δ3，即最后一层的“error”。直接用假设值-对应数据项的真实值即可，注意转置问题。

3. 计算hidden层的δ。，注意计算完成后要舍弃bias位置。

4. 计算gradient中的Δ。

5. 计算没有正则化的gradient。即直接除以m即可。

2.4 Gradient checking

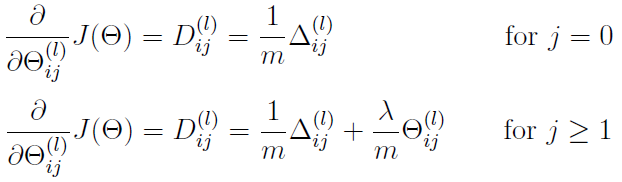
这一部分实现了用数学公式计算gradient以对梯度下降法算出的gradient的验证。在代码文件中已经给出这一部分的代码，不需要自己实现。发现在computeNumericalGradient(J, theta)中，用了一个循环完成了θ-ε和θ+ε向量的表示，最后返回的则是数学计算得到的gradient值。



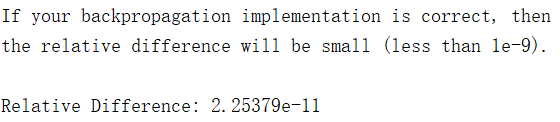
需要注意的是，我们在验证gradient计算方法的正确性时一般选用较小的input，而不是直接使用training set得到。所以需要一个checkNNGradients函数来随机生成一个较小的input与验证gradient计算算法的正确性。当正确后，要关闭该验证算法。

2.5 Regularized Neural Networks

前面已经用反向传播算法正确实现了非正规化的gradient，这一部分就是需要加上正规化后的gradient。



可以从公式中看出，Θ的第一列不应该被正规化，按照之前的方法，将Θ的第一列置零即可。



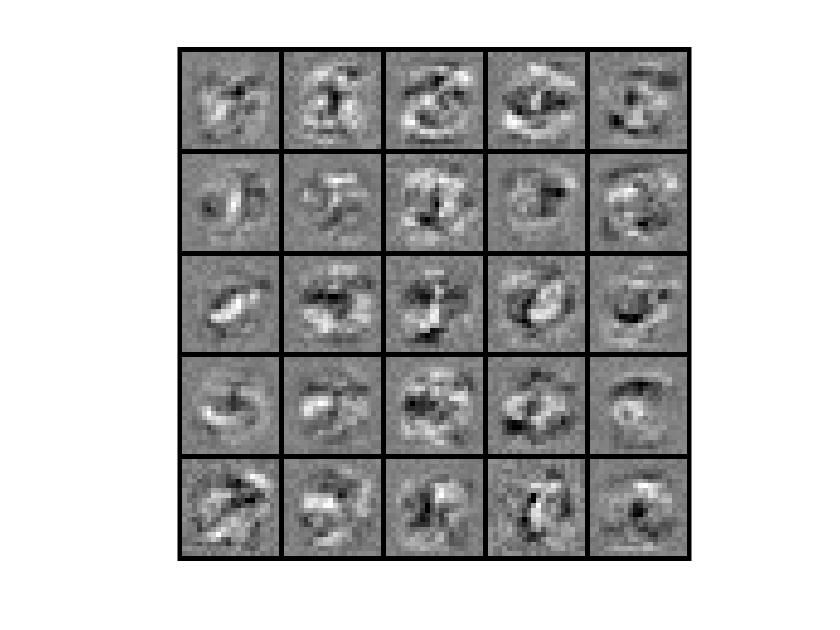
2.6 Learning parameters using fmincg

这一部分就比较简单了，因为前面把准备工作都完成了，所以这一部分只需要使用optimized function fmincg计算parameter的最佳值即可。

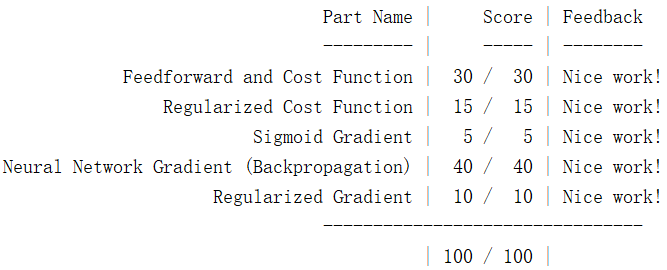


3 Visualizing the hidden layer

这一部分将hidden layer进行了可视化处理，如下图：



○ 到此所有评分实验部分成功完成！



3.1 Optional (ungraded) exercise

这一部分是通过变换λ和迭代次数来对神经网络算法有更好的了解。如果λ的值过小导致不能很好地正则化，那么可能会出现overfitting的问题。