The results you need to get in ex3

ex3

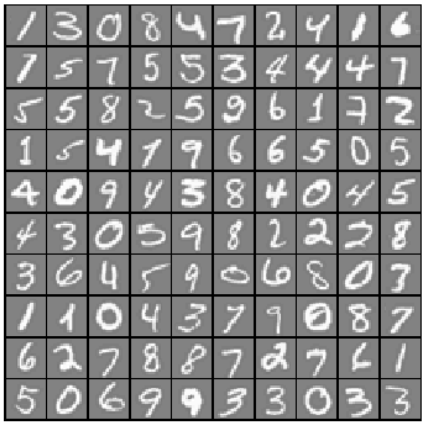
Part1

第一部分 多元分类

1.手写数字的数据可视化

ex3.m中实现：

```
%训练数据被存储到X，y中，无需再定义变量
load('ex3data1.mat');
%m为数据集的列数，这里是5000行
m = size(X, 1);
%随机选取100个数据行
rand_indices = randperm(m);
%数字打乱后返回一个向量
sel = X(rand_indices(1:100), :);
%调用数字展示函数displayData.m
displayData(sel);
```



2.向量化逻辑回归

为了将数字分成10类，需要生成10个逻辑回归分类器，而使用向量运算可以使训练过程更加高效，

（1）损失函数向量化

```
t=sigmoid(X*theta);
J=(-y'*log(t)-(1-y)'*log(1-t))/m;
```

（2）向量化梯度

```
grad=(t-y)/m;
```

（3）对正则化后的逻辑回归向量化

```
t=sigmoid(X*theta);
J=(-y'*log(t)-(1-y)'*log(1-t))/m+lambda*sum(theta(2:end).^2)/(2*m);
theta(1)=0;
grad=X‘*(t-y)/m+lambda/m*theta;
```

Trick：A[:,3:5]=B[:,1:3] ；使用end关键字；使用sum求和函数

此时提交：

```
==
==                                Part Name |     Score | Feedback
==                                --------- |     ----- | --------
==           Regularized Logistic Regression |  30 /  30 | Nice work!
==            One-vs-All Classifier Training |   0 /  20 |
==          One-vs-All Classifier Prediction |   0 /  20 |
==        Neural Network Prediction Function |   0 /  30 |
==                                           ---------------------------------
==                                                     |  30 / 100 |
--
```

3.One-vs-all 一对多分类

通过实现多个正则化逻辑回归来实现一对多的分类效果，在本次习题中只是分成10类，但是代码需要能够使样本分成任意数量类，完成One-vs-all.m代码来为每一个类别生成一个逻辑回归分类器，需要返回一个包含全部分类器的所有参数的参数矩阵，K✖（N+1），每一整行元素代表训练后的一个类的逻辑回归分类器参数。我们可以通过for循环来对每个分类器进行独立的训练，y变量为元素为1-10的标签向量，此时10个分类器均已经对样本进行过一次判断，结果为0/1。最终ex3.m代码会根据One-vs-all.m的多个二元分类器（即整个参数矩阵）训练出多元分类器。

trick：数组a=数值b代表着a的全部元素均需要进行判别运算。返回只包含0/1值的逻辑数组；使用fmincg函数进行高阶的梯度下降运算，类似于fminunc，但是在参数较多时可以更加的高效,本次习题中单个分类器包含了401个参数。

代码实现;

```
%样本个数
m = size(X, 1);
%特征数
n = size(X, 2);
% all_theta为最终需要返回的参数矩阵
all_theta = zeros(num_labels, n + 1);
%加入插入向量的样本矩阵
X = [ones(m, 1) X];
options = optimset('GradObj', 'on', 'MaxIter', 50);
%for循环求all_theta参数矩阵，注意每次求一个分类器均需要重新设置initial_theta向量
for c=1:num_labels
    initial_theta = zeros(n + 1, 1);
    [theta]=fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)),initial_theta, options);
    all_theta(c,:)=theta';
end
```

运行：

```
Training One-vs-All Logistic Regression...
Iteration    50 | Cost: 1.379516e-02
Iteration    50 | Cost: 5.725251e-02
Iteration    50 | Cost: 6.400463e-02
Iteration    50 | Cost: 3.543449e-02
Iteration    50 | Cost: 6.182145e-02
Iteration    50 | Cost: 2.140899e-02
Iteration    50 | Cost: 3.560987e-02
Iteration    50 | Cost: 8.587522e-02
Iteration    50 | Cost: 7.913039e-02
Iteration    50 | Cost: 1.003794e-02
```

此时提交：

```
Iteration    26 | Cost: 3.391340e-01
Iteration    22 | Cost: 6.631438e-02
Iteration    30 | Cost: 1.002636e-01
Iteration    16 | Cost: 3.598119e-01
==
==                                  Part Name |     Score | Feedback
==                                  --------- |     ----- | --------
==            Regularized Logistic Regression |  30 /  30 | Nice work!
==             One-vs-All Classifier Training |  20 /  20 | Nice work!
==           One-vs-All Classifier Prediction |   0 /  20 |
==         Neural Network Prediction Function |   0 /  30 |
==                                  ---------------------------------
==                                            |  50 / 100 |
```

4.使用One-vs-all分类进行预测

完成predictOne-vs-all.m进行预测，返回每个样本所属数字种类（1-10）的向量，其原理是使用每个分类器对样本进行计算，返回每种预测生成的概率，最高的即为最有可能对应的数字类别。

注：这里使用到Octave的max函数，如果参数为矩阵将会返回由矩阵的列向量的最大值

```
m = size(X, 1);
num_labels = size(all_theta, 1);
p = zeros(m, 1);
X = [ones(m, 1) X];
%通过max函数返回样本最大概率构成的向量和及对应的索引
temp=sigmoid(X*all_theta');
[s,p]=max(temp,[],2);
```

ex3.m调用OneVsAll函数

```
lambda = 0.1;
[ ] = oneVsAll(X, y, num_labels, lambda);
```

最后会对准确性进行检测：

```
Training Set Accuracy: 95.020000
```

此时提交：

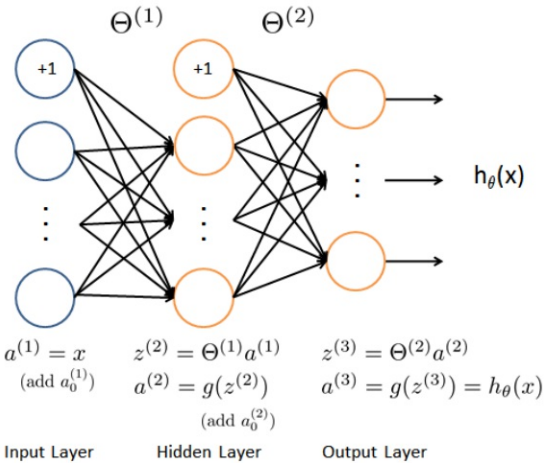
```
==
==                                  Part Name |     Score | Feedback
==                                  --------- |     ----- | --------
==            Regularized Logistic Regression |  30 /  30 | Nice work!
==             One-vs-All Classifier Training |  20 /  20 | Nice work!
==           One-vs-All Classifier Prediction |  20 /  20 | Nice work!
==         Neural Network Prediction Function |   0 /  30 |
==                                  ---------------------------------
==                                            |  70 / 100 |
==
```

Part2

第二部分 神经网络

由于逻辑回归始终只是一个线性分类器，无法形成更加复杂的假设，所以训练神经网络来重新进行判别，能够做出非线性假设表征复杂模型。使用前向传播Forward Propagation 算法进行判别，下一个练习使用反向传播算法来学习神经网络参数。后面使用ex3_nn.m运行结果。

1.如何表现神经网络模型



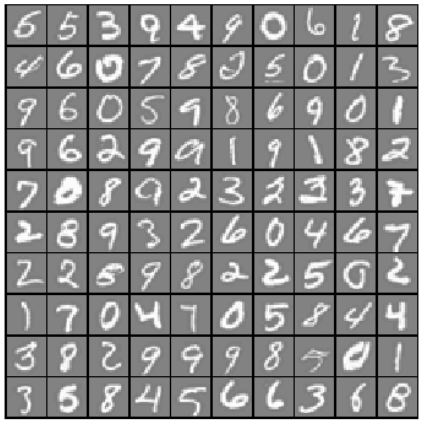
%同样挑选100个样本数据进行显示

load('ex3data1.mat');

sel = randperm(size(X, 1));

sel = sel(1:100);

displayData(X(sel, :));



%加载权重数据存储至Theta1（401*25）和Theta2（10*26）中

load（“ex3weights.mat”）;

2.前向传播算法以及推算

完成predict.m文件，同样会返回最大逻辑回归概率所对应的标签

注：需要为样本添加偏差项

Pridict.m文件代码实现：

```
X=[ones(m,1) X];
temp1=sigmoid(X*Theta1');
temp2=[ones(m,1) temp1];
temp3=sigmoid(temp2*Theta2');
[s,p]=max(temp3,[],2);
```

ex3_nn文件代码实现：

```
%进行推算
pred = predict(Theta1, Theta2, X);
%计算准确率
fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

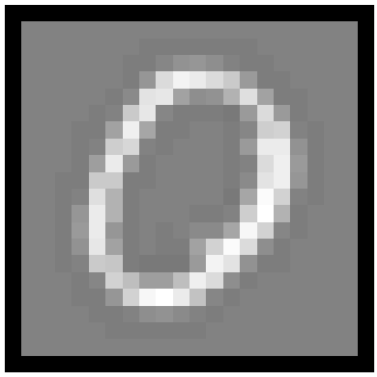
结果如下图:

```
Loading Saved Neural Network Parameters ...

Training Set Accuracy: 97.520000
```

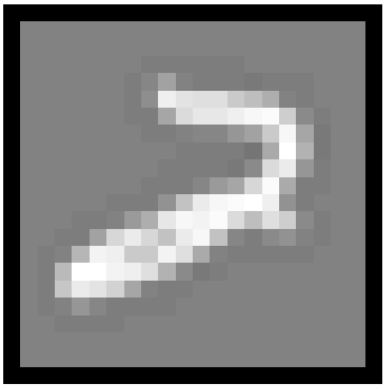
```
%进行推算，随即显示一张图，推算其对应的数字
rp = randperm(m);
for i = 1:m
    % 挨个显示图片
    fprintf('\nDisplaying Example Image\n');
    displayData(X(rp(i), :));
    %推算单张图片所对应的数字
    pred = predict(Theta1, Theta2, X(rp(i),:));
    fprintf('\nNeural Network Prediction: %d (digit %d)\n', pred, mod(pred, 10));
    % Pause
    fprintf('Program paused. Press enter to continue.\n');
    pause;
end
```

部分输出结果如下：



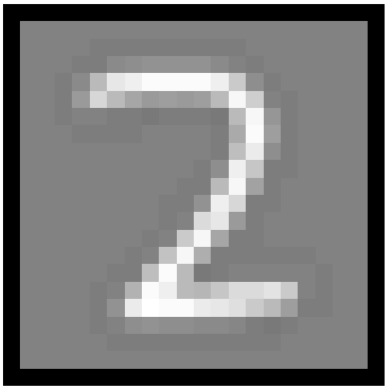
```
Displaying Example Image

Neural Network Prediction: 10 (digit 0)
```



```
Displaying Example Image

Neural Network Prediction: 2 (digit 2)
```



```
Displaying Example Image

Neural Network Prediction: 2 (digit 2)
```

最终提交：

```
==
==                                  Part Name |     Score | Feedback
==                                  --------- |     ----- | --------
==               Regularized Logistic Regression |  30 /  30 | Nice work!
==                One-vs-All Classifier Training |  20 /  20 | Nice work!
==              One-vs-All Classifier Prediction |  20 /  20 | Nice work!
==            Neural Network Prediction Function |  30 /  30 | Nice work!
==                                  --------------------------------
==                                            | 100 / 100 |
==
```