Neural Network

Homework 1

Ning Ma, A50055399

1 Perceptron

1. For 2-D, the decision boundary is

$$w_1 x_1 + w_2 x_2 = \theta \tag{1}$$

Assume x is a point on the boundary. So, we have

$$w^T x = \theta \tag{2}$$

Let x^0 be the point from which we want to compute the distance to the line. So, the distance is

$$\frac{w^\top (x - x^0)}{||w||_2} = \frac{w^\top x - w^\top x^0}{||w||_2}$$

So, the distance from the origin to the boundary is

$$\frac{w^\top x - w^\top 0}{||w||_2} = \frac{w^\top x}{||w||_2} \tag{3}$$

2.

(a) The learning rule is

$$w_j(t+1) = w_j(t) + \alpha(Teacher - Output)x_j \tag{4}$$

(b) The perceptron learning goes as the following:
The final weights and threshold are w_1 = -1, w_2 = -1, $theta$ = -1.

(c) This solution is not unique. For example, $w_1 = -2$, $w_2 = -2$, and $\theta = -2$ is another solution. Namely, the solution will not change if you add same constant to all the parameters.

Table 1: perceptron learning for NAND

x_1	x_2	w_1	w_2	Net	$Output$	$Teacher$	$theta$
1	1	0	0	0	1	0	0
0	0	-1	-1	0	0	1	1
0	1	-1	-1	-1	0	1	0
1	1	-1	0	-1	1	0	-1
1	0	-2	-1	-2	0	1	0
-	-	-1	-1	-	-	-	-1

(d) i. See code file 'readProcessData.m' for details. After preprocessing, the data are unitless. If the data have different units, different data points may have very different numerical value but actually represent the same physical measurement. Secondly, if the original data is very large, this preprocessing may avoid overflowing and let the learning converge appropriately.

ii. According to Figure 1, the class are linearly separable for most feature spaces.

iii. See code for more details. For the stopping criteria, I choose to stop when the iteration reaches 1000 or the error drops below 0.05, whichever occurs first.

iv. The test error rate is just 3.33%

v. my learning rate is 0.05. When I slightly increase the learning rate, the perceptron learning will converge a little bit fast and vice versa. However, if my learning rate is too large, the learning procedure will not converge.

2 Logistic and Softmax Regression

1.

$$E(\theta) = -\sum_{i=1}^{N} \left(y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right) \tag{5}$$

$$\frac{\partial E(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \frac{\partial E(\theta)}{\partial h_\theta(x^{(i)})} \frac{\partial h_\theta(x^{(i)})}{\partial \theta_j} \tag{6}$$

where

$$\frac{\partial E(\theta)}{\partial h_\theta(x^{(i)})} = -\frac{y^{(i)} - h_\theta(x^{(i)})}{h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))} \tag{7}$$

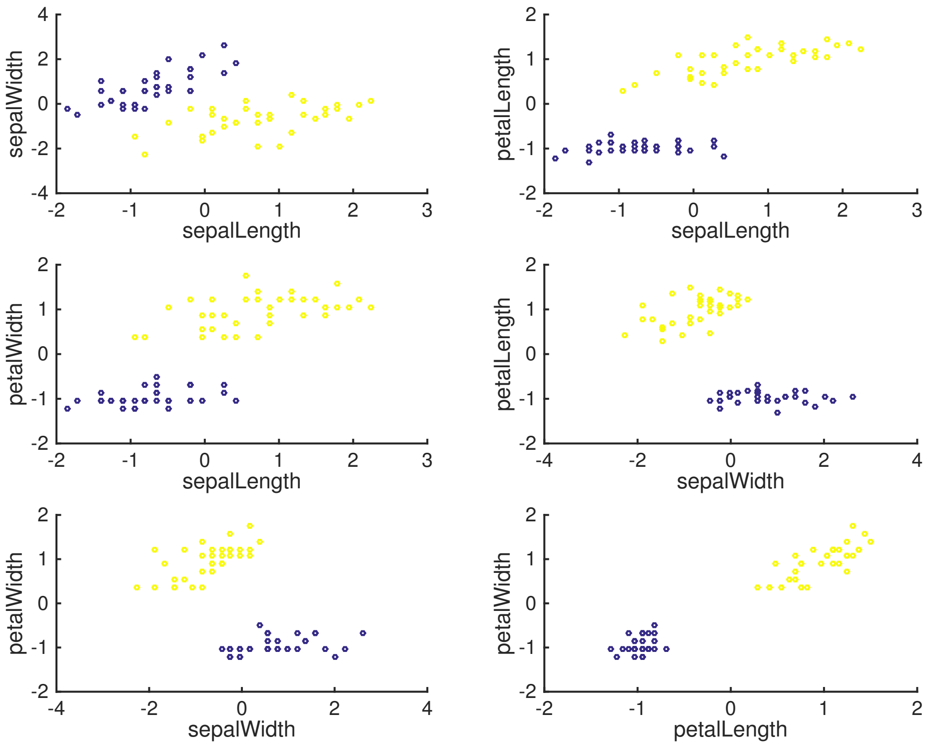

Figure 1: scatter plot

and

$$\frac{\partial h_\theta(x^{(i)})}{\partial \theta_j} = h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))x_j^{(i)} \tag{8}$$

Thus, we have

$$\frac{\partial E(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \frac{\partial E(\theta)}{\partial h_\theta(x^{(i)})} \frac{\partial h_\theta(x^{(i)})}{\partial \theta_j} \tag{9}$$

$$= \sum_{i=1}^{N} (h_\theta(x^{(i)}) - y_i)x_j^{(i)} \tag{10}$$

2.

$$E(\theta) = -\sum_{i=1}^{N}\sum_{l=0}^{K} 1_{\{y^{(i)}=l\}} \log \frac{\exp(\theta^{(l)\top} x^{(i)})}{\sum_{j=0}^{K} \exp(\theta^{(j)\top} x^{(i)})} \tag{11}$$

$$= -\sum_{i=1}^{N}\sum_{l=0}^{K} 1_{\{y^{(i)}=l\}} \big(\theta^{(l)\top} x^{(i)} - \log \sum_{j=0}^{K} \exp(\theta^{(j)\top} x^{(i)})\big) \tag{12}$$

So, we have

$$\frac{\partial E(\theta)}{\partial \theta^{(k)}} = -\sum_{i=1}^{N} \left[1_{\{y^{(i)}=k\}} x^{(i)} - \sum_{l=0}^{K} 1_{\{y^{(i)}=l\}} \frac{x^{(i)} \exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=0}^{K} \exp(\theta^{(j)\top} x^{(i)})} \right] \tag{13}$$

$$= -\sum_{i=1}^{N} \left[1_{\{y^{(i)}=k\}} x^{(i)} - \frac{x^{(i)} \exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=0}^{K} \exp(\theta^{(j)\top} x^{(i)})} \right] \tag{14}$$

$$= -\sum_{i=1}^{N} \left[x^{(i)} \left(1_{\{y^{(i)}=k\}} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right] \tag{15}$$

3.
See code file 'readOneMNIST.m' for details. Before appending 1 to the x-vector, I divide each vector by 255 to avoid overflowing.
4.

(a) The result of the 10 2-way classification is in Table 2

Table 2: 10 2-way classification

$0-all$	$1-all$	$2-all$	$3-all$	$4-all$	$5-all$	$6-all$	$7-all$	$8-all$	$9-all$
0.977	0.982	0.957	0.951	0.957	0.950	0.960	0.955	0.921	0.935

(b) The overall test accuracy is 0.846.

5.

(a) See Figure 2. We can see that the training accuracy increases as the iteration number.

(b) The test accuracy on the test data is 0.860.

(c) The test accuracy is a little bit hight than the one-vs-all logistic regression. It is because softmax regression can directly computes the probability of each class, and then vote for the one with highest probability. So, softmax regression will provide a little bit more accurate result.

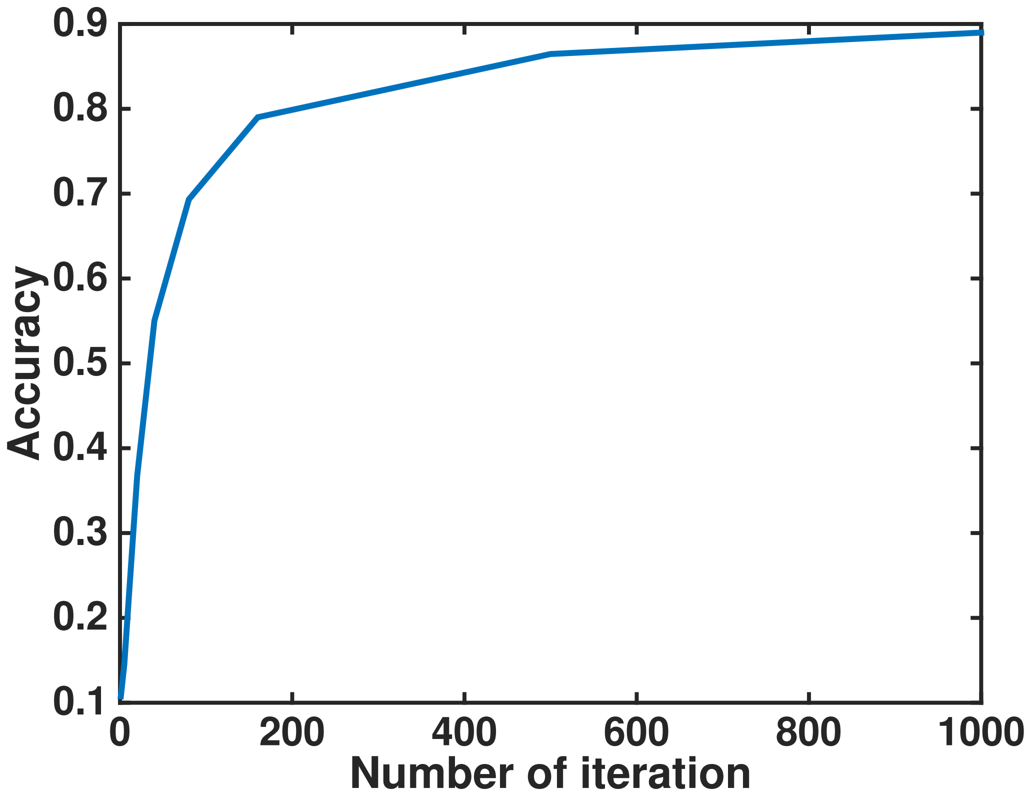

Figure 2: Accuracy vs. Iteration

3 Appendix

The following is the source code

3.1 Code for Peceptron

```
%%%%%%%%%%%%%%%%getProcessData.m%%%%%%%%%%%%%%%%%%%%%%
%dependency: readProcessData
[trainData, trainLabel] = readProcessData('./iris/iris_train.data');
[testData, testLabel] = readProcessData('./iris/iris_test.data');
save('processedData.mat', 'trainData', 'trainLabel', 'testData', 'testLa

%%%%%%%%%%%%%%%%%%readProcessData.m%%%%%%%%%%%%%%%%%%%%
function [data, label] = readProcessData(file)
f = fopen(file);
```

```
rawData = textscan(f,'%f %f %f %f %s','Delimiter',',');
data = cell2mat(rawData(:,1:4)); %ignor
label = cellfun(@(x) strcmp(x,'Iris-setosa'),rawData{:,5});

meanData = repmat(mean(data,1), size(data,1),1);
stdData = repmat(std(data,1),size(data,1),1);

data = (data - meanData)./ stdData;
data = [ones(size(data,1),1),data];

%%%%%%%%%%%%%%HW1Perceptron.m%%%%%%%%%%%%%
%dependency: perceptronLearning.m, perceptronrror.m
load('processedData.mat');

%(d) ii.
attributes = {'sepalLength', 'sepalWidth','petalLength','petalWidth'};

fig = figure();
subIndex = 1;
for i = 1:(length(attributes) - 1)
    for j = (i + 1) : length(attributes)
    subplot(3,2,subIndex);
    colorVec = 10*(trainLabel == 1) + 250*(trainLabel == 0);
    scatter(trainData(:,i + 1), trainData(:,j + 1), 5, colorVec);
    xlabel(attributes{i});
    ylabel(attributes{j});
    subIndex = subIndex + 1;
    end
end

saveas(fig,'./figure/scatterPlot.fig');

%%(d) iii.
w0 = 1- rand(size(trainData,2),1);
w = perceptronLearning(trainData, trainLabel, w0);

%%(d) iv
error = perceptronError(testData, testLabel, w);
save('testError','error');
```

```
%%%%%%%%%%%%perceptronLearning.m%%%%%%%%%%%%%%%
function w = perceptronLearning(data, label, w0)
[nTrain, ~] = size(data);
maxIter = 10000;
errorCriteria = 0.05;
step = 0.05;

w = w0;
errorCount = 0;
sampleCount = 0;
%data(i,:) = [1 x1 x2 x3 ... xk], which has already included bias term
for k = 1:maxIter
    i = randi(nTrain,1);
    %equivalent to : output = data(i,2:end)*w(2:end) >= -w(1);
    output = (data(i,:)*w) >= 0;
    errorCount = errorCount + abs(output - label(i));
    sampleCount = sampleCount + 1;
    error = errorCount / sampleCount;

    %if the training error is below the threshold, stop training
    if (error ~= 0 && error < errorCriteria)
        display(k);
        break;
    end
    %update the weight
    w = w + step*(label(i) - output)*data(i,:)';
end


%%%%%%%%%%%%perceptronError.m%%%%%%%%%%%%
function error = perceptronError(data, label, w)
errorCount = sum(abs((data*w >= 0) - label));
error = errorCount / size(data,1);
```

3.2 Code for Logistic and Softmax Regression

3.2.1 Code for Logistic Regression

```
%%%%%%%%%%%%%%%%%%%%%%%%%%% readData.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%dependency: readOneMNIST.m
```

```
[trainImages, trainLabels] = readOneMNIST('train-images-idx3-ubyte', 'tr
[testImages, testLabels] = readOneMNIST('t10k-images-idx3-ubyte', 't10k-

save('./data/trainData','trainImages','trainLabels');
save('./data/testData', 'testImages', 'testLabels');

%%%%%%%%%%%%%%%%%%%%%%%%%%% readOneMNIST.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read 'number' images together with labels from images/labels files
% Each images is represented as a 28 x 28 + 1= 785 dimension vector wher
% the first element '1' represents the intercept.
% Each image vector comes with the corresponding label
%Input:
%imageFiel: the image files
%labelFile: the label files
%radNum: the number of read files
%Output:
%images: a number x 785 matrix. Each row represents one image vector
%labels: labels of selected images
function [images,labels] = readOneMNIST(imageFile, labelFile, readNum)
    addpath ../../share/
    % open and read information of image files
    fidImage = fopen(imageFile, 'r', 'b');
    %get the magic number which is 2051 for image file
    magicNum = fread(fidImage, 1, 'int32');
    if magicNum ~= 2051
        error('Invalid image file header');
    end

    %how images are there in this data set
    count = fread(fidImage, 1, 'int32');
    if count < readNum
        error('Trying to read too many digits');
    end

    %open and read information of label file
    fidLabel = fopen(labelFile, 'r', 'b');
    magicNum = fread(fidLabel, 1, 'int32');
    %get the magic number which is 2049 for label file
    if magicNum ~= 2049
        error('Invalid label file header');
```

```
    end
    count = fread(fidLabel, 1, 'int32');
    if count < readNum
        error('Trying to read too many digits');
    end

    %get hight and width of each image
    h = fread(fidImage, 1, 'int32');
    w = fread(fidImage, 1, 'int32');

    %images matrix
    images = zeros(readNum, h*w);
    %label vector

    for i=1:readNum
        oneImage = (fread(fidImage, w*h, 'uint8'));
        images(i, :) = oneImage;
    end
    images = [ones(size(images,1),1), images / 255];

    %images = images;

    labels = fread(fidLabel, readNum, 'uint8');

    fclose(fidImage);
    fclose(fidLabel);
end



%%%%%%%%%%%%%%%%%%%%%%HW1Part4.m%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load('./data/testData.mat');
load('./data/trainData.mat');

%testProb: each row corresponds to test probability of 10 2-way classifi
%test image
[nTest, nFeature] = size(testImages);
testProb = zeros(nTest, 10);

threshold = 3*10^-3;
```

```
theta0 = rand(nFeature,1) - 0.5;
step = 5*10^-6;
numIter = 2000;

for class =  1:10
    tic
    theta = twoWayClassifier(trainImages, trainLabels, theta0, step, num
    testProb(:,class) = logisticFunc(testImages,theta);
    toc
end

save('./data/logisticTestProb', 'testProb');
twoWayCorrectCount = zeros(1,10);

%(a) compute overal test accuracy
%voteLabels: the label with largest probability for every class
[maxProbs, voteLabels] = max(testProb, [], 2);
overalCorrectCount = sum((voteLabels - 1) == testLabels);


%(b)compute two-way classification accuracy
for class = 1:10
    thisClass = (testLabels == (class - 1)) & (testProb(:,class) >= 0.5)
    notThisClass = (testLabels ~= (class - 1)) & (testProb(:,class) < 0.
    twoWayCorrectCount(class) = sum(thisClass | notThisClass);
end

overalCorrectRate = overalCorrectCount / nTest;
twoWayCorrectRate = sum(twoWayCorrectCount,1) / nTest;

save('./data/logisticOveralAccuracy','overalCorrectRate');
save('./data/logisticTwoWayAccuracy','twoWayCorrectRate');

%%%%%%%%%%%%%%%%%%%% logisticFunc.m %%%%%%%%%%%%%%%%%%%%%%%%%%
%return the value of logistic function
% x: 1 x K
% theta : K x 1
function value = logisticFunc(X, theta)
value = 1./(1 + exp(- X*theta));
```

```
%%%%%%%%%%%%%%%%%%%%%% logisticCost.m %%%%%%%%%%%%%%%%%%%%%%%
function cost = logisticCost(X,y, theta)
h_theta = logisticFunc(X,theta);
cost = -sum(y.*log(h_theta) + (1 - y).*log(1 - h_theta));

%%%%%%%%%%%%%%%%%%%%%%logisticGradient %%%%%%%%%%%%%%%%%%%%%%
function grad = logisticGradient(X,y, theta)
hx = logisticFunc(X, theta);
grad = X'*(hx - y);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%% LogisticGradientDescent.m %%%%%%%%%%%%%%%
%the optimal parameters
%Using gradient descent to compute the optimal paramters for logistic
%rgression.
%X is nxm matrix. Each row is one data point and each column is one feat
%y is nx1 vector representing the labels
function theta = LogisticGradientDescent(X, y, theta0, ...
    step, numIter, threshold)
theta = theta0;
%threshold = 3*10^-3;
%numIter = 2000;
%step = 5*10^-6;

for iter = 1:numIter
    preTheta = theta;
    %iter;
    %cost = logisticCost(X, y, theta);
    gradient = logisticGradient(X,y, theta);
    %sum(theta)
    %sum(gradient)
    %norm(gradient)
    %logisticGradient(X,y, theta);
    theta = theta - step*gradient;
    diff = norm(theta - preTheta);
    if (diff < threshold)
        break;
    end

end
```

```
end

%%%%%%%%%%%%%%%%%%%%%% twoWayClassifier.m %%%%%%%%%%%%%%%%
function theta = twoWayClassifier(trainData, trainLabel, theta0, step, n
newLabel = (trainLabel == (class - 1)) * 1;
theta = LogisticGradientDescent(trainData, newLabel, theta0,step, numIte
```

3.2.2 Code for Softmax Regression

```
%%%%%%%%%%%%%%%%%%%% Hw1Part5.m %%%%%%%%%%%%%%%%%%%%
%Dependency: softmaxGradientDescent.m, softmaxAccuracy
load('./data/testData.mat');
load('./data/trainData.mat');

[nTrian, nFeature] = size(trainImages);
K = 10;
theta0 = rand(K, nFeature) - 0.5;
step = 5*10^-6;
threshold = 3*10^-3;

%%part (a)
numIters = [1 5 10 20 40 80 160 500 1000];
trainAccuracyIter = zeros(1, length(numIters));

for i = 1:length(numIters)
    theta = theta0;
    tic
    numIter = numIters(i);
    for iter = 1:numIter
        theta = theta - step*softmaxGradient(trainImages,trainLabels, th
    end
    trainAccuracyIter(i) = softmaxAccuracy(trainImages,trainLabels,theta
    toc
end
save('./data/softmaxTestAccuracyIter','trainAccuracyIter');
h = plot(numIters, trainAccuracyIter,'linewidth',3);
xlabel('Number of iteration');
ylabel('Accuracy');
set(gca,'fontWeight','bold','FontSize',20,'linewidth',2)
saveas(h,'./data/accuracyVsIteration.fig');
```

```
%%part (b)
theta = softmaxGradientDescent(trainImages, trainLabels, theta0, step, 2
testAccuracy = softmaxAccuracy(testImages,testLabels,theta);
save('./data/softmaxTestAccuracy','testAccuracy');

%%%%%%%%%%%%%%%% sofmaxFunc.m %%%%%%%%%%%%%%%%%%%%%%%%
function prob = softmaxFunc(X,theta,K)
%prob = P(y^{i} = k|x^{i};theta)
thetaX = exp(theta*X');
sumThetaX = repmat(sum(thetaX,1),K,1);
prob = thetaX./sumThetaX;
end

%%%%%%%%%%%%%%%%%%%% softmaxCost.m %%%%%%%%%%%%%%%%%%%%%
%theta is K x nFeature: each row is the theta  for one specific class
%X is N x nFeature
function cost = softmaxCost(X,y,theta, K)
%thetaX = exp[ theta_1* x_1, theta_1*x_2, ..., theta_1*x_N;
%                    .            .          .         .
%                    .            .          .         .
%                    .            .          .         .
%              theta_K* x_1, theta_K* x_1,..., theta_K* x_1;
%]
[N, ~] = size(X);
thetaX = exp(theta*X');
sumThetaX = repmat(sum(thetaX,1),K,1);

Y = repmat(y', K, 1);
Kmatrix = repmat((0:(K - 1))', 1, N);
labelLogic = (Y == Kmatrix);

cost = -sum(sum(labelLogic.*(log(thetaX./sumThetaX))));


%%%%%%%%%%%%%%%%% softmaxGradient.m %%%%%%%%%%%%%%%%%
%theta is K x nFeature: each row is the theta  for one specific class
%X is N x nFeature
function grad = softmaxGradient(X,y,theta, K)
%thetaX = exp[ theta_1* x_1, theta_1*x_2, ..., theta_1*x_N;
%                    .            .          .         .
```

```
%                               .                  .                .              .
%                               .                  .                .              .
%                    theta_K* x_1, theta_K* x_2,..., theta_K* x_1;
%]

%prob = P(y^{i} = k|x^{i};theta)
prob = softmaxFunc(X, theta,K);

%labelLogic  = 1{y^{i} = k}
[N, ~] = size(X);
labelLogic = (repmat(y', K,1) == repmat((0:(K - 1))',1,N));

%1{y^{i} = k} - P(y^{i} = k|x^{i};theta)
diff = labelLogic*1 - prob;

%gradient with respect to theta
grad = -diff*X;
end

%%%%%%%%%%%%%%%% softmaxGradientDescent.m %%%%%%%%%%%%%%%%%%%%%
%theta is  K x nFeature
%Dependency: softmaxGradient
function theta  = softmaxGradientDescent(X, y, theta0, step, numIter, th
theta = theta0;
%numIter = 2000;
%threshold = 3*10^-3;
%step = 5*10^-6;

for iter = 1:numIter
    %iter
    preTheta = theta;
    theta = theta - step*softmaxGradient(X,y, theta, K);
    diff = norm(theta - preTheta,'fro');
    if (diff < threshold)
        break;
    end
end

%%%%%%%%%%%%%%%% softmaxAccuracy.m %%%%%%%%%%%%%%%%%%
%Given the classifier, compute the accuracy of this classifier
```

```
%on the 'data'
%theta is K x nFeature. Each row is the theta for one class.
function accuracy = softmaxAccuracy(data,labels,theta)

[nTrain, ~] = size(data);
correctCount = 0;
for i = 1:nTrain
    image = data(i,:);
    prob = exp(theta*image');
    [~, maxLabel] = max(prob);
    if (maxLabel - 1) == labels(i);
        correctCount = correctCount + 1;
    end
end
accuracy = correctCount / nTrain;
```