Programming Exercise 3:
Multi-class Classification and Neural Networks
September 2020

Machine Learning Course (Stanford University)

This exercise shows the problem of multi-class classification in recognizing hand-written digits and the implementation of the solution using one-vs-all logistic regression and neural networks(feedforward only for now).

1 Multi-class Classification

1.1 Challenge

Automated handwritten digit recognition is widely used today - from recognizing zip codes (postal codes) on mail envelopes to recognizing amounts written on bank checks. The exercise's scope is in the recognition of handwritten digits (from 0 to 9), exploring two different approaches: one-vs-all logistic regression and neural nets.

1.2 Dataset

Handwritten digits

Images

- 5000 training examples
- Each training example is 20 x 20 pixels (grayscale)
- Each pixel is represented by a floating point number indicating the grayscale intensity at that location

    Flattening

- 20 x 20 pixels is "unrolled"/"flattened" into a 400x400 dimensional vector.
- Each of these training examples becomes a single row in our data matrix X

    Resulting data

- Therefore, this gives us a 5000 by 400 matrix X
    - Where every row is a training example of a handwritten digit image

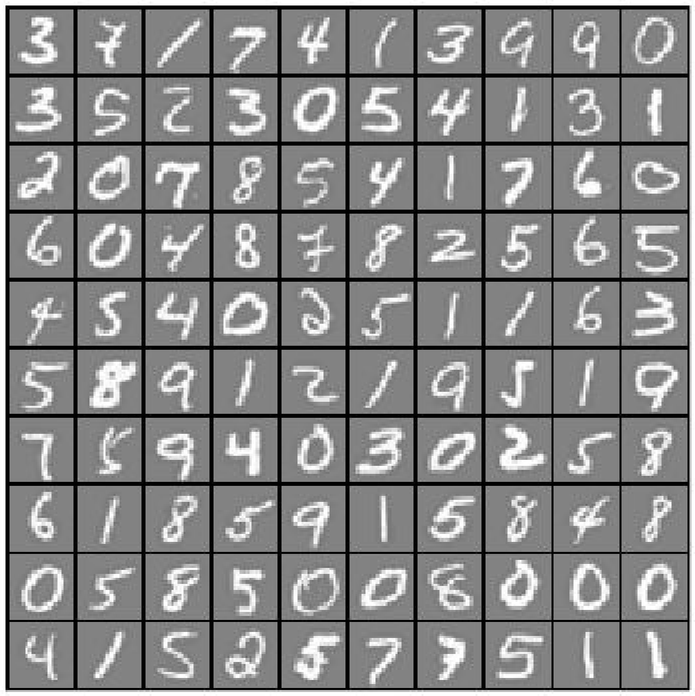
Ground Truth Labels

- 5000 dimensional vector y

    Implementation Caution!

- To make things more compatible with Octave/MATLAB indexing, where there is no zero index, we have mapped the digit zero to the value ten.
    - Therefore, a “0” digit is labeled as “10”, while the digits “1” to “9” are labeled as “1” to “9” in their natural order.

1.3 Visualizing the Data



1.4 Vectorizing Logistic Regression

Instructions:
You will be using multiple one-vs-all logistic regression models to build a multi-class classifier.

- Since there are 10 classes, you will need to train **10 separate** logistic regression classifiers.
- To make this training efficient, it is important to ensure that your code is well vectorized.

1.4.1 Vectorizing the cost function

This is the same as in exercise #2.

1.4.2 Vectorizing the gradient

This is the same as in exercise #2.

1.4.3 Vectorizing regularized logistic regression

This is the same as in exercise #2.

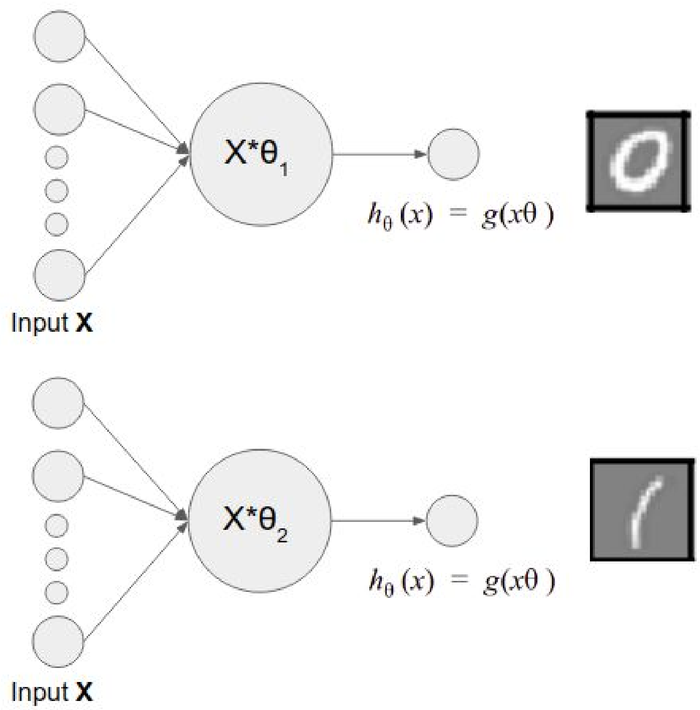
The whole lrCostFunction is as follows:

```
function [J, grad] = lrCostFunction(theta, X, y, lambda)
%LRCOSTFUNCTION Compute cost and gradient for logistic regression with
%regularization
%   J = LRCOSTFUNCTION(theta, X, y, lambda) computes the cost of using
%   theta as the parameter for regularized logistic regression and the
%   gradient of the cost w.r.t. to the parameters.

% Initialize some useful values
m = length(y); % number of training examples

% Variables to return
J = 0;
grad = zeros(size(theta));

size_t= size(theta);

h = sigmoid(X*theta);
J = (1/m)*(sum( ( (-y)'*log(h) ) - (1-y)'*(log(1-h)) )) + (lambda/(2*m))*sum((theta(2:size_t)).^2);

grad_theta=zeros(size_t);
grad_theta(2:size_t)=(lambda/m)*(theta(2:size_t));
grad = (1/m)*((X')*(h - y)) + grad_theta;

end
```

1.5 One-vs-all Classification

As said before, we trained 10 different logistic regression classifiers here.
The question is **how**. Let’s break it down little by little.

The image below shows only the case for class 0 and 1 but the same should apply to 2-9 in our case.



What’s the difference?

They differed on the learned parameters θ; thus, the model itself.

And that is because in the first place, the ***learning phase*** is also different.

How different is the *learning phase?*

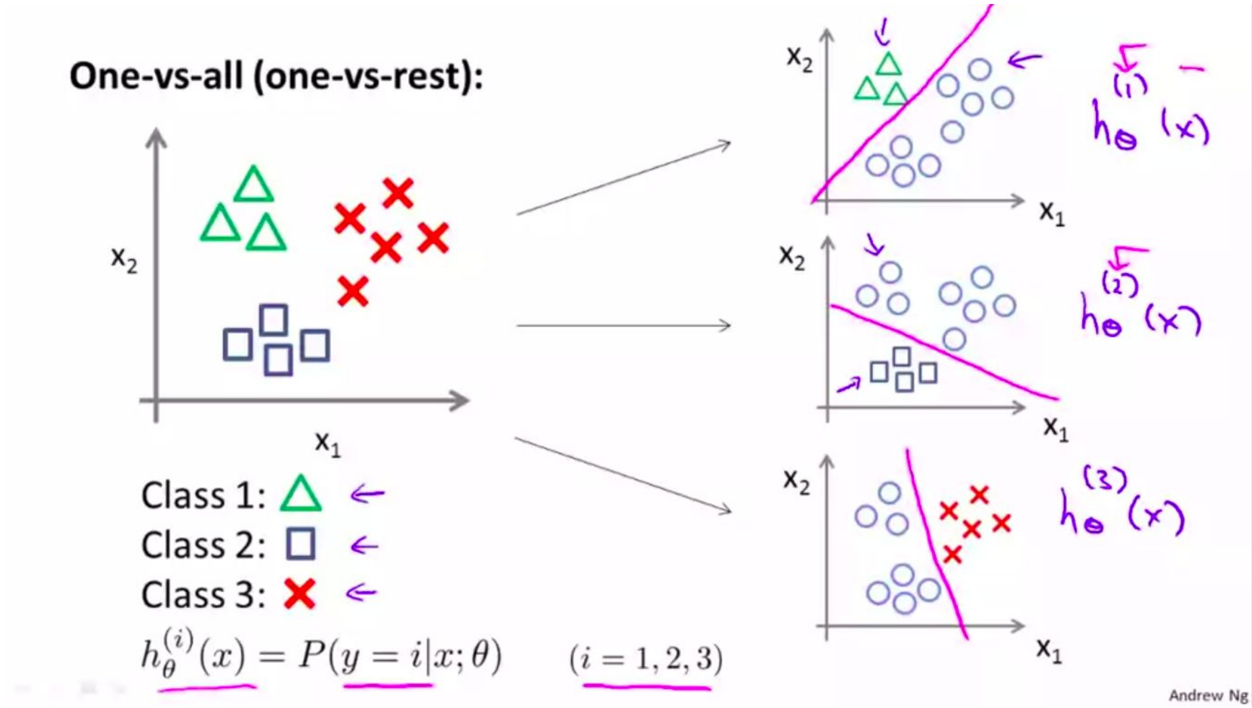
They differ in the ground truth label, ***y***.
Say you have the y = [0, 1, 1, 3, 2] for your first 5 images.

So for the

- first model for class 0, you have y = [1, 0, 0, 0, 0].
- Second model for class 1, you have y = [0, 1, 1, 0, 0].
- and so on.

So we can see here that we transformed our outputs from integers to 1s and 0s.
That sense is the reason why we call this one-vs-all or one-vs-rest since when we consider a class x, we treat the rest as non-x class.

Recall from the lecture:



So we now dive deep to the implementation.

1.5.1 One-vs-all Training

Instruction:

- In this part of the exercise, you will implement one-vs-all classification by training multiple regularized logistic regression classifiers, one for each of the K classes in our dataset
  *In the handwritten digits dataset, K = 10, but your code should work for any value of K

- **Parameters and on Multiple Classes**
  - You should now complete the code in oneVsAll.m to train one classifier for each class.
    - your code should return all the classifier parameters in a matrix $\Theta \in R^{K\times(N+1)}$
      - where each row of Θ corresponds to the learned logistic regression parameters for one class.

      You can do this with a “for”-loop from 1 to K, training each classifier independently.

Ground truth Labels (y)

- The *y* argument to this function is a vector of labels from 1 to 10, where we have mapped the digit “0” to the label 10 (to avoid confusions with indexing).

- When training the classifier for class $k \in \{1, ..., K\}$, you will want a m-dimensional vector of labels y, where $y_j \in 0, 1$ indicates whether the j-th training instance belongs to class k ($y_j = 1$), or if it belongs to a different class ($y\ j = 0$).
  *You may find logical arrays helpful for this task.

Use `fmincg` instead of `fminunc`. See more on the documentation.

This is our function for onevsAll training.

```
function [all_theta] = oneVsAll(X, y, num_labels, lambda)
%ONEVSALL trains multiple logistic regression classifiers and returns all
%the classifiers in a matrix all_theta, where the i-th row of all_theta
%corresponds to the classifier for label i
%   [all_theta] = ONEVSALL(X, y, num_labels, lambda) trains num_labels
%   logistic regression classifiers and returns each of these classifiers
%   in a matrix all_theta, where the i-th row of all_theta corresponds
%   to the classifier for label i

% Some useful variables
m = size(X, 1); % 5000
n = size(X, 2); % 400
X = [ones(m, 1) X];

all_theta = zeros(num_labels, n + 1); % variable to return

initial_theta = zeros(n+1, 1);
options = optimset('GradObj', 'on', 'MaxIter', 50);

for c=1:num_labels, %for every class c
  all_theta(c,:) = ...
  fmincg (@(t)(lrCostFunction(t, X, (y == c), lambda)), ...
  initial_theta, options);
end
end
```

1.5.2 One-vs-all Prediction

After training, we have all the parameters for the models in the 10 classes (0-9 numbers) in one matrix.

Then we do the same as before, however, our ***h*** or hypothesis won't be anymore a single output, instead we will have output--the probabilities on the 10 classes. To get the single output, we get the max of those 10 probabilities.

Example:

h = [0.2, 0.1, 0.99, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9)

max(h) = 0.99 at index 3

```
function p = predictOneVsAll(all_theta, X)
%PREDICT Predict the label for a trained one-vs-all classifier. The labels
%are in the range 1..K, where K = size(all_theta, 1).
%  p = PREDICTONEVSALL(all_theta, X) will return a vector of predictions
%  for each example in the matrix X. Note that X contains the examples in
%  rows. all_theta is a matrix where the i-th row is a trained logistic
%  regression theta vector for the i-th class. You should set p to a vector
%  of values from 1..K (e.g., p = [1; 3; 1; 2] predicts classes 1, 3, 1, 2
%  for 4 examples)

m = size(X, 1);
num_labels = size(all_theta, 1);

% You need to return the following variables correctly
p = zeros(size(X, 1), 1);

% Add ones to the X data matrix
X = [ones(m, 1) X];

h = sigmoid(X*all_theta');
disp(h)
[max_h, p] = max(h, [], 2)

end

```

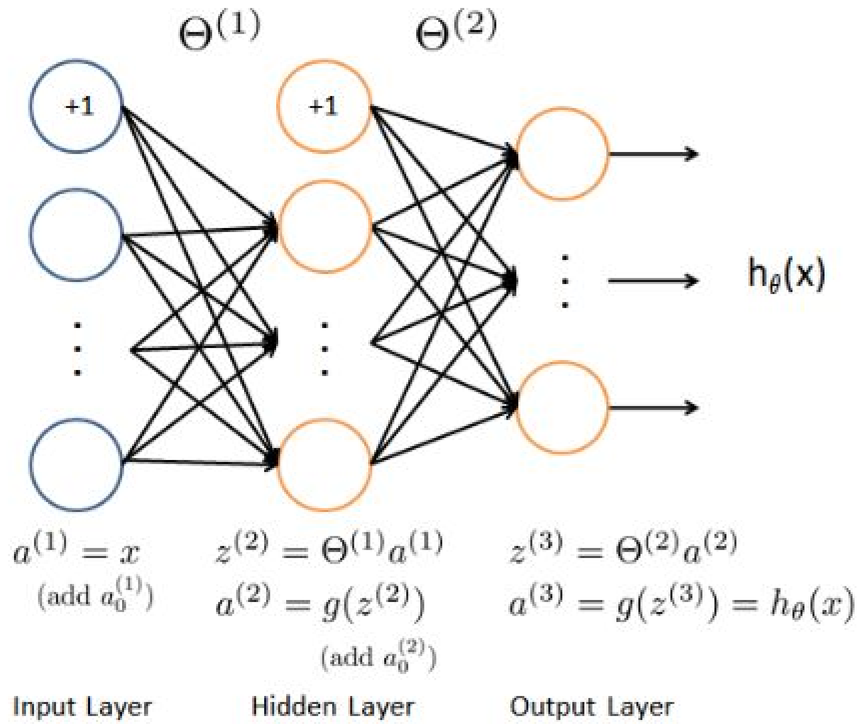
The training accuracy is ~94.9%.

2 One-vs-all VS. Neural Nets

One-vs-all logistic regression works but only at some point. It cannot form complex hypotheses as it is only a linear classifier. To have more complex boundaries or nonlinear hypothesis, we can use Neural Networks.

3 Neural Networks

3.1 Model Representation



Input Layer

- Since the size of our images are of size 20x20, we will also do flattening here resulting to 400 pixels per image in the 5000 training samples.
- Don't forget the bias unit

Hidden Layer

- In our case, we put 25 units
- Don't forget the bias unit

Output Layer

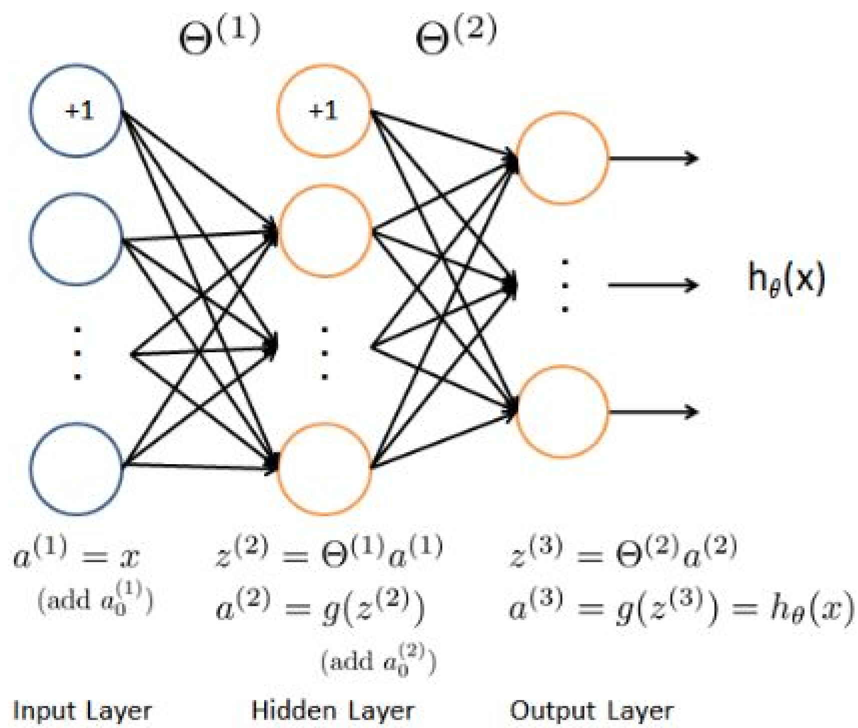
- This has 10 units corresponding to the 10 classes.

For now, the weights are given and loaded.

```
% Load saved matrices from file
load('ex3weights.mat');

% The matrices Theta1 and Theta2 will now be in your Octave
% environment
% Theta1 has size 25 x 401
% Theta2 has size 10 x 26
```

3.2 Feedforward Propagation and Prediction

Copying the same network above:



The implementation is straightforward:

```
function p = predict(Theta1, Theta2, X)
%PREDICT Predict the label of an input given a trained neural network
%   p = PREDICT(Theta1, Theta2, X) outputs the predicted label of X given the
%   trained weights of a neural network (Theta1, Theta2)

% Useful values
m = size(X, 1);
num_labels = size(Theta2, 1);

p = zeros(size(X, 1), 1); % variable to return

a_1 = [ones(m, 1) X];

z_2 = a_1*Theta1'; % result is 16x4
a_2 = sigmoid(z_2);

a_2 = [ones(size(a_2,1),1) a_2]

z_3 = a_2*Theta2';
a_3 = sigmoid(z_3);

[max_h, p] = max(a_3, [], 2)

end
```

The training accuracy became ~97.5%.

3 Scores

```
==
==                                   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 |
```