**EX3 – Neural Network Learning**

input\_layer\_size = 400; % 20x20 Input Images of Digits

hidden\_layer\_size = 25; % 25 hidden units

num\_labels = 10; % 10 labels, from 1 to 10

load('ex4data1.mat');

m = size(X, 1);

initial\_Theta1 = randInitializeWeights(input\_layer\_size, hidden\_layer\_size);

initial\_Theta2 = randInitializeWeights(hidden\_layer\_size, num\_labels);

initial\_nn\_params = [initial\_Theta1(:) ; initial\_Theta2(:)];

%% =================== Part 1: Training NN ===================

options = optimset('MaxIter', 100);

lambda = 1;

costFunction = @(p) nnCostFunction(p, ...

input\_layer\_size, ...

hidden\_layer\_size, ...

num\_labels, X, y, lambda);

[nn\_params, cost] = fmincg(costFunction, initial\_nn\_params, options);

Theta1 = reshape(nn\_params(1:hidden\_layer\_size \* (input\_layer\_size + 1)), ...

hidden\_layer\_size, (input\_layer\_size + 1));

Theta2 = reshape(nn\_params((1 + (hidden\_layer\_size \* (input\_layer\_size + 1))):end), ...

num\_labels, (hidden\_layer\_size + 1));

%% ================= Part 2: Implement Predict =================

pred = predict(Theta1, Theta2, X);

fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) \* 100);

function W = **randInitializeWeights**(L\_in, L\_out)

%RANDINITIALIZEWEIGHTS Randomly initialize the weights of a layer with L\_in

%incoming connections and L\_out outgoing connections

W = zeros(L\_out, 1 + L\_in);

epsilon\_init = sqrt(6)/(sqrt(L\_in + L\_out));

W = rand(L\_out, 1 + L\_in) \* 2 \* epsilon\_init;

end

function g = **sigmoid**(z)

%SIGMOID Compute sigmoid functoon

g = 1.0 ./ (1.0 + exp(-z));

end

function g = **sigmoidGradient**(z)

%SIGMOIDGRADIENT returns the gradient of the sigmoid function

%evaluated at z

g = zeros(size(z));

T = sigmoid(z);

g = T.\*(1-T);

end

function p = **predict**(Theta1, Theta2, X)

%PREDICT Predict the label of an input given a trained neural network

m = size(X, 1);

num\_labels = size(Theta2, 1);

p = zeros(size(X, 1), 1);

h1 = sigmoid([ones(m, 1) X] \* Theta1');

h2 = sigmoid([ones(m, 1) h1] \* Theta2');

[dummy, p] = max(h2, [], 2);

end

function [J grad] = **nnCostFunction**(nn\_params, ...

input\_layer\_size, ...

hidden\_layer\_size, ...

num\_labels, ...

X, y, lambda)

%NNCOSTFUNCTION Implements the neural network cost function for a two layer

%neural network which performs classification

Theta1 = reshape(nn\_params(1:hidden\_layer\_size \* (input\_layer\_size + 1)), ...

hidden\_layer\_size, (input\_layer\_size + 1));

Theta2 = reshape(nn\_params((1 + (hidden\_layer\_size \* (input\_layer\_size + 1))):end), ...

num\_labels, (hidden\_layer\_size + 1));

m = size(X, 1);

%y(value) -> y(vector)

yd = eye(num\_labels);

y = yd(y,:);

a1 = [ones(m,1) X];

a2 = sigmoid(a1\*Theta1');

a2 = [ones(size(a1,1),1) a2];

h\_theta = sigmoid(a2\*Theta2');

%Cost Function

J = -(1./m)\*sum(sum(y.\*log(h\_theta) + (1-y).\*log(1-h\_theta))) ...

+ lambda/(2\*m)\*(sum(sum(Theta1(:,2:end).^2))+sum(sum(Theta2(:,2:end).^2)));

D3 = h\_theta - y;

D2 = (D3\*Theta2(:,2:end)).\*sigmoidGradient(a1\*Theta1');

Temp1 = Theta1;

Temp1(:,1) = 0;

Temp2 = Theta2;

Temp2(:,1) = 0;

%Delta

Theta1\_grad = (1./m)\*(D2'\*a1) + (lambda/m)\*Temp1;

Theta2\_grad = (1./m)\*(D3'\*a2) + (lambda/m)\*Temp2;

grad = [Theta1\_grad(:) ; Theta2\_grad(:)];

end