**神经网络反向传播主程序**

%% Initialization

clear ; close all; clc

%% Setup the parameters you will use for this exercise

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

% (note that we have mapped "0" to label 10)

times = 30;

step = 10;

times\_start = 30;

times\_end= times\_start + step\*times;

acc\_times = zeros(1,times+1);

for i = 0:times

fprintf(['------------------------第 %d 轮--------------------------\n'],i);

%% =========== Part 1: 加载和可视化数据 =============

% We start the exercise by first loading and visualizing the dataset.

% You will be working with a dataset that contains handwritten digits.

% Load Training Data

load('ex1data.mat');

m = size(X, 1);

% Randomly select 100 data points to display

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

sel = sel(1:100);

displayData(X(sel, :));

fprintf('数据已经加载完成并可视化。\n');

%% ================ Part 2: 加载参数 ================

% In this part of the exercise, we load some pre-initialized neural network parameters.

% Load the weights into variables Theta1 and Theta2

load('ex1weights.mat');

% Unroll parameters

nn\_params = [Theta1(:) ; Theta2(:)];

fprintf('参数已经加载完成。\n');

%% ================ Part 3: 计算成本（前向） ================

% To the neural network, you should first start by implementing the feedforward part of the neural network that returns the cost only.

% You should complete the code in nnCostFunction.m to return cost.

% After implementing the feedforward to compute the cost,

% you can verify that your implementation is correct by verifying that you get the same cost as us for the fixed debugging parameters.

%

% We suggest implementing the feedforward cost \*without\* regularizationfirst so that it will be easier for you to debug.

% Later, in part 4, you will get to implement the regularized cost.

% Weight regularization parameter (we set this to 0 here).

lambda = 0;

J = nnCostFunction(nn\_params, input\_layer\_size, hidden\_layer\_size, ...

num\_labels, X, y, lambda);

fprintf(['Cost at parameters (loaded from ex1weights): %f '...

'\n(this value should be about 0.288401 )\n'], J);

fprintf('\n第三部分完成\n');

%% =============== Part 4: 实现正则化 ===============

% Once your cost function implementation is correct,

% you should now continue to implement the regularization with the cost.

% Weight regularization parameter (we set this to 1 here).

lambda = 1;

J = nnCostFunction(nn\_params, input\_layer\_size, hidden\_layer\_size, ...

num\_labels, X, y, lambda);

fprintf(['Cost at parameters (loaded from ex1weights): %f '...

'\n(this value should be about 0.408577)\n'], J);

fprintf('第四部分完成。\n');

%% ================ Part 5: 计算激活函数Sigmoid的梯度 ================

% Before you start implementing the neural network,

% you will first implement the gradient for the sigmoid function.

% You should complete the code in the sigmoidGradient.m file.

g = sigmoidGradient([1 -0.5 0 0.5 1]);

fprintf('Sigmoid gradient evaluated at [1 -0.5 0 0.5 1]:\n ');

fprintf('%f ', g);

fprintf('\n\n');

fprintf('第五部分完成。\n');

%% ================ Part 6: 初始化参数 ================

% In this part of the exercise,

% you will be starting to implment a two layer neural network that classifies digits.

% You will start byimplementing a function to initialize the weights of the neural network

% (randInitializeWeights.m)

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

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

% Unroll parameters

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

fprintf('第六部分完成。\n');

%% =============== Part 7: 实现反向传播 ===============

% Once your cost matches up with ours, you should proceed to implement the backpropagation algorithm for the neural network.

% You should add to the code you've written in nnCostFunction.m to return the partial derivatives of the parameters.

% Check gradients by running checkNNGradients

checkNNGradients;

fprintf('第七部分完成，但计算出的梯度偏大。\n');

%% =============== Part 8: 实现正则化 ===============

% Once your backpropagation implementation is correct,

% you should now continue to implement the regularization with the cost and gradient.

% Check gradients by running checkNNGradients

lambda = 3;

checkNNGradients(lambda);

% Also output the costFunction debugging values

debug\_J = nnCostFunction(nn\_params, input\_layer\_size, ...

hidden\_layer\_size, num\_labels, X, y, lambda);

fprintf(['\n\nCost at (fixed) debugging parameters (w/ lambda = 10): %f ' ...

'\n(this value should be about 0.648928)\n\n'], debug\_J);

fprintf('第八部分完成。\n');

%% =================== Part 9: 训练神经网络 ===================

% You have now implemented all the code necessary to train a neural network.

% To train your neural network, we will now use "fmincg", which is a function which works similarly to "fminunc".

% Recall that these advanced optimizers are able to train our cost functions efficiently as long as we provide them with the gradient computations.

%%%%%%%%%%%%%%%%%%%%%%%%%% 参数调节 %%%%%%%%%%%%%%%%%%%%%%%%%%

% After you have completed the assignment, change the MaxIter to a larger value to see how more training helps.

test\_num = times\_start + step\*i

options = optimset('MaxIter', test\_num);

% You should also try different values of lambda

lambda = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Create "short hand" for the cost function to be minimized

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

input\_layer\_size, ...

hidden\_layer\_size, ...

num\_labels, X, y, lambda);

% Now, costFunction is a function that takes in only one argument (the neural network parameters)

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

% Obtain Theta1 and Theta2 back from nn\_params

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));

fprintf('第九部分完成。\n');

%% ================= Part 10: 可视化权重 =================

% You can now "visualize" what the neural network is learning by displaying the hidden units

% to see what features they are capturing in the data.

displayData(Theta1(:, 2:end));

fprintf('第十部分完成。\n');

%% ================= Part 11: 实现预测 =================

% After training the neural network, we would like to use it to predict the labels.

% You will now implement the "predict" function to use the neural network to predict the labels of the training set.

% This lets you compute the training set accuracy.

pred = predict(Theta1, Theta2, X);

acc\_times(i+1)=mean(double(pred == y)) \* 100;

fprintf('\n训练集准确性: %f\n', acc\_times(i+1));

fprintf('======================全部完成=====================\n');

end

figure();

f1=plot(times\_start:step:times\_end,acc\_times);

set(f1, 'LineWidth', 1.2); % 使用句柄修改线条粗细

title("训练次数对准确率的影响")

xlabel('训练次数')

ylabel('准确率')

grid on;grid minor;