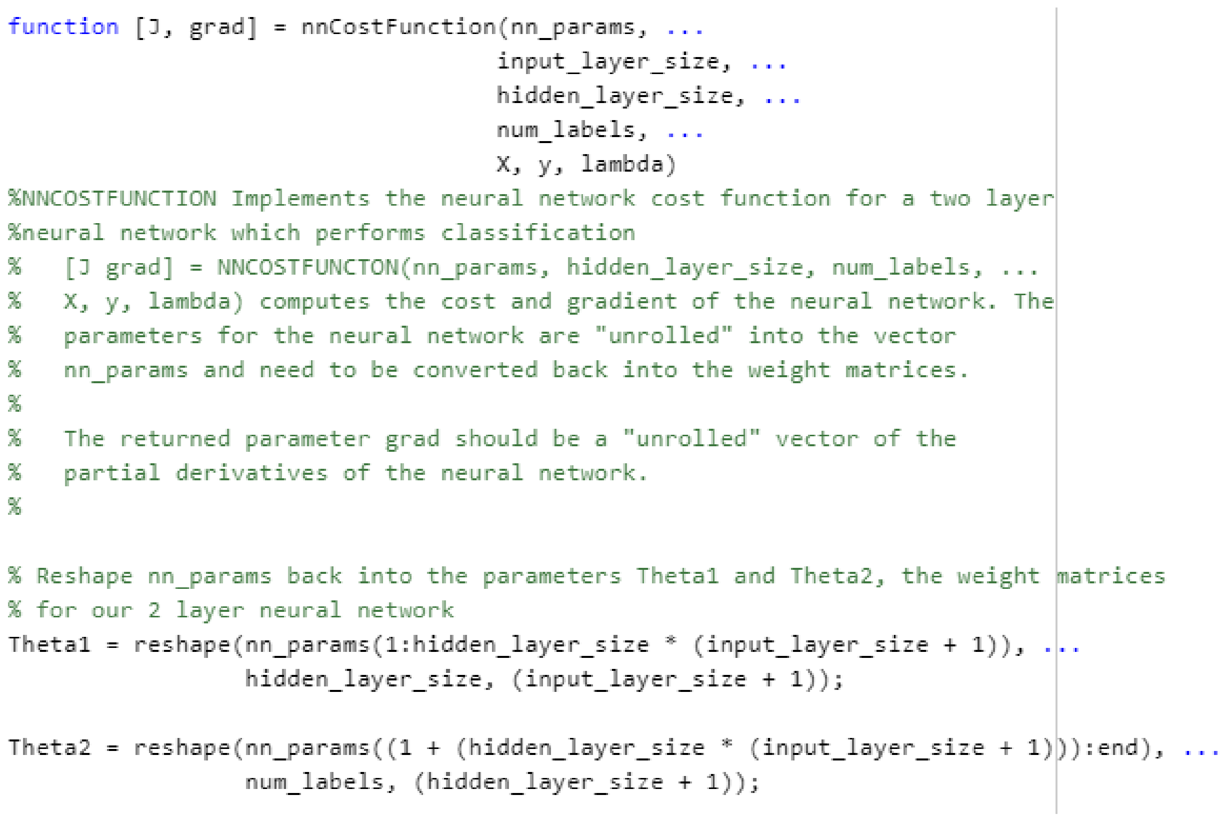
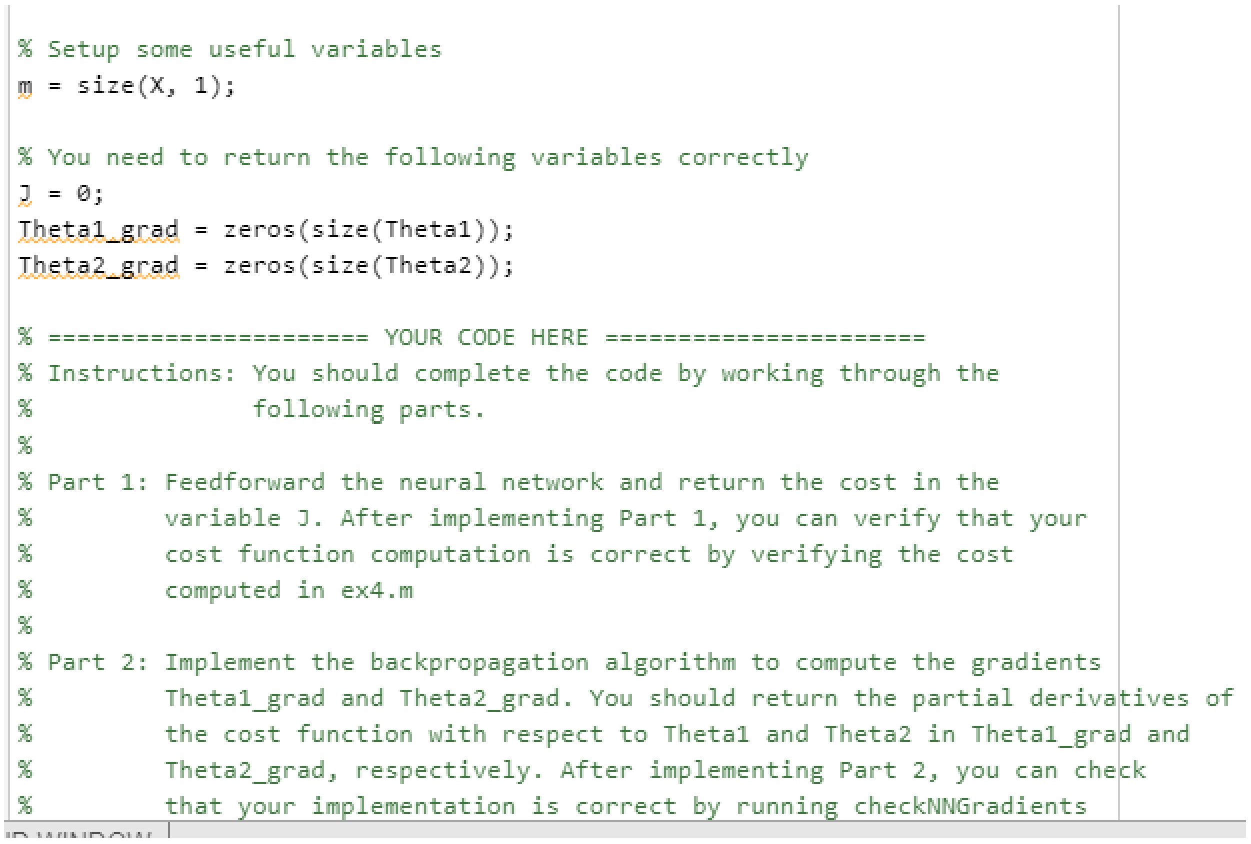
```
function g = sigmoidGradient(z)
%SIGMOIDGRADIENT returns the gradient of the sigmoid function
%evaluated at z
%   g = SIGMOIDGRADIENT(z) computes the gradient of the sigmoid function
%   evaluated at z. This should work regardless if z is a matrix or a
%   vector. In particular, if z is a vector or matrix, you should return
%   the gradient for each element.

g = zeros(size(z));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the gradient of the sigmoid function evaluated at
%               each value of z (z can be a matrix, vector or scalar).

g = sigmoid(z).*(1 - sigmoid(z));
```

```
function W = randInitialzeWeights(L_in, L_out)
%RANDINITIALIZEWEIGHTS Randomly initialize the weights of a layer with L_in
%incoming connections and L_out outgoing connections
%   W = RANDINITIALIZEWEIGHTS(L_in, L_out) randomly initializes the weights
%   of a layer with L_in incoming connections and L_out outgoing
%   connections.
%
%   Note that W should be set to a matrix of size(L_out, 1 + L_in) as
%   the first column of W handles the "bias" terms
%

% You need to return the following variables correctly
W = zeros(L_out, 1 + L_in);

% ====================== YOUR CODE HERE ======================
% Instructions: Initialize W randomly so that we break the symmetry while
%               training the neural network.
%
% Note: The first column of W corresponds to the parameters for the bias unit
%
epsilon_init = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon_init - epsilon_init;
```



```
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
%   [J grad] = NNCOSTFUNCTON(nn_params, hidden_layer_size, num_labels, ...
%   X, y, lambda) computes the cost and gradient of the neural network. The
%   parameters for the neural network are "unrolled" into the vector
%   nn_params and need to be converted back into the weight matrices.
%
%   The returned parameter grad should be a "unrolled" vector of the
%   partial derivatives of the neural network.
%

% Reshape nn_params back into the parameters Theta1 and Theta2, the weight matrices
% for our 2 layer neural network
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));
```



```
% Setup some useful variables
m = size(X, 1);

% You need to return the following variables correctly
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));

% ====================== YOUR CODE HERE ======================
% Instructions: You should complete the code by working through the
%               following parts.
%
% Part 1: Feedforward the neural network and return the cost in the
%         variable J. After implementing Part 1, you can verify that your
%         cost function computation is correct by verifying the cost
%         computed in ex4.m
%
% Part 2: Implement the backpropagation algorithm to compute the gradients
%         Theta1_grad and Theta2_grad. You should return the partial derivatives of
%         the cost function with respect to Theta1 and Theta2 in Theta1_grad and
%         Theta2_grad, respectively. After implementing Part 2, you can check
%         that your implementation is correct by running checkNNGradients
%         Note: The vector y passed into the function is a vector of labels
%               containing values from 1..K. You need to map this vector into a
%               binary vector of 1's and 0's to be used with the neural network
%               cost function.
%
%         Hint: We recommend implementing backpropagation using a for-loop
%               over the training examples if you are implementing it for the
%               first time.
%
% Part 3: Implement regularization with the cost function and gradients.
%
%         Hint: You can implement this around the code for
%               backpropagation. That is, you can compute the gradients for
%               the regularization separately and then add them to Theta1_grad
%               and Theta2_grad from Part 2.
%

% Setup some useful variables
m = size(X, 1);
num_labels = size(Theta2, 1);
```

```
% 1. Feed-forward to compute h = a3.
a1 = [ones(1, m); X'];
z2 = Theta1 * a1;
a2 = [ones(1, m); sigmoid(z2)];
a3 = sigmoid(Theta2 * a2);

% Eplode y into 10 values with Y[i] : = i == y.
Y = zeros(num_labels, m);
Y(sub2ind(size(Y), y', 1:m)) = 1;

% Compute the non-regularized error. Fully vectorized, at the expense of
% having an expandad Y in memory (which is 1/40th the size of X, so it
% should be fine).
J = (1/m) * sum(sum(-Y .* log(a3) - (1 - Y) .* log(1 - a3)));

% Add regularied error. Drop the bias term in the 1st columns.
J = J + (lambda / (2*m)) * sum(sum(Theta1(:, 2:end) .^2));
J = J + (lambda / (2*m)) * sum(sum(Theta2(:, 2:end) .^2));
```

```
% 2. Backpropagate to get graident information.
d3 = a3 - Y;
d2 = (Theta2' * d3) .*[ones(1, m); sigmoidGradient(z2)];

% Vectorized ftw:
Theta2_grad = (1/m) * d3 * a2';
Theta1_grad = (1/m) * d2(2:end ,:) * a1';

% Add gradient regularizaiton
Theta2_grad = Theta2_grad + (lambda / m) * ([zeros(size(Theta2, 1), 1), Theta2(:, 2:end)]);
Theta1_grad = Theta1_grad + (lambda / m) * ([zeros(size(Theta1, 1), 1), Theta1(:, 2:end)]);
```