predict.m:

```
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);

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

% ====================== YOUR CODE HERE ======================
% Instructions: Complete the following code to make predictions using
%               your learned neural network. You should set p to a
%               vector containing labels between 1 to num_labels.
%
% Hint: The max function might come in useful. In particular, the max
%       function can also return the index of the max element, for more
%       information see 'help max'. If your examples are in rows, then, you
%       can use max(A, [], 2) to obtain the max for each row.
%

h1 = sigmoid([ones(m, 1) X] * Theta1');
h2 = sigmoid([ones(m, 1) h1] * Theta2');
[temp, p] = max(h2, [], 2);

%
=========================================================================

end
```

nnCostFunction.m

```
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.
%
```

```
K = num_labels;

Y = eye(K)(y,:); % [5000, 10]

% Part 1
a1 = [ones(m, 1), X]; % results in [5000, 401]
a2 = sigmoid(Theta1 * a1'); % results in [25, 5000]
a2 = [ones(1, size(a2, 2)); a2]; % results in [26, 5000]
h = sigmoid(Theta2 * a2); % results in [10, 5000]

costPositive = -Y .* log(h)';
costNegative =  (1 - Y) .* log(1 - h)';
cost = costPositive - costNegative;

J = (1/m) * sum(cost(:));

% Part 1.4 regularization
Theta1Filtered = Theta1(:,2:end);
Theta2Filtered = Theta2(:,2:end);
reg = (lambda / (2*m)) * (sumsq(Theta1Filtered(:)) + sumsq(Theta2Filtered(:)));
J = J + reg;

% Part 2: Implement the backpropagation algorithm to compute the gradients

Delta1 = 0;
Delta2 = 0;
for t = 1:m
  a1 = [1; X(t,:)'];
  z2 = Theta1 * a1;
  a2 = [1; sigmoid(z2)];
  z3 = Theta2 * a2;
  a3 = sigmoid(z3);

  yt = Y(t,:)';
  d3 = a3 - yt;

  % step 3
  d2 = (Theta2Filtered' * d3) .* sigmoidGradient(z2);

  Delta2 = Delta2 + (d3 * a2');
  Delta1 = Delta1 + (d2 * a1');
end



Theta1_grad = (1/m) * Delta1;
Theta2_grad = (1/m) * Delta2;

% Part 3: Implement regularization with the cost function and gradients.
Theta1_grad(:,2:end) = Theta1_grad(:,2:end) + ((lambda / m) * Theta1Filtered);
```

```
Theta2_grad(:,2:end) = Theta2_grad(:,2:end) + ((lambda / m) * Theta2Filtered);


% -------------------------------------------------------------

% =========================================================================

% Unroll gradients
grad = [Theta1_grad(:) ; Theta2_grad(:)];


end
```

sigmoidGradient.m

```
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).

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


% =============================================================


end
```

randInitializeWeights.m

```
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 = 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 column row 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 row of W corresponds to the parameters for the bias units
%


epsilon = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon - epsilon;


% =========================================================================

end
```

screenshot:

starter code — gnuplot — 80×24

```
Loading and Visualizing Data ...
Program paused. Press enter to continue.

Loading Saved Neural Network Parameters ...

Training Set Accuracy: 97.520000
Program paused. Press enter to continue.
```

starter code — gnuplot — 80×24

```
Loading and Visualizing Data ...
Program paused. Press enter to continue.

Loading Saved Neural Network Parameters ...

Feedforward Using Neural Network ...
Cost at parameters (loaded from ex3weights): 0.287629
(this value should be about 0.287629)

Program paused. Press enter to continue.
```

starter code — gnuplot — 80×24

```
Loading and Visualizing Data ...
Program paused. Press enter to continue.

Loading Saved Neural Network Parameters ...

Feedforward Using Neural Network ...
Cost at parameters (loaded from ex3weights): 0.287629
(this value should be about 0.287629)

Program paused. Press enter to continue.

Checking Cost Function (w/ Regularization) ...
Cost at parameters (loaded from ex3weights): 0.383770
(this value should be about 0.383770)
Program paused. Press enter to continue.

Evaluating sigmoid gradient...
Sigmoid gradient evaluated at [1 -0.5 0 0.5 1]:
  0.000000 0.000000 0.000000 0.000000 0.000000

Program paused. Press enter to continue.
```

starter code — less — 80×24

```
Loading Saved Neural Network Parameters ...

Feedforward Using Neural Network ...
Cost at parameters (loaded from ex3weights): 0.287629
(this value should be about 0.287629)

Program paused. Press enter to continue.

Checking Cost Function (w/ Regularization) ...
Cost at parameters (loaded from ex3weights): 0.383770
(this value should be about 0.383770)
Program paused. Press enter to continue.

Evaluating sigmoid gradient...
Sigmoid gradient evaluated at [1 -0.5 0 0.5 1]:
  0.196612 0.235004 0.250000 0.235004 0.196612

Program paused. Press enter to continue.
```

starter code — gnuplot –

```
Initializing Neural Network Parameters ...

Checking Backpropagation...
  -9.2783e-03  -9.2783e-03
   8.8991e-03   8.8991e-03
  -8.3601e-03  -8.3601e-03
   7.6281e-03   7.6281e-03
  -6.7480e-03  -6.7480e-03
  -3.0498e-06  -3.0498e-06
   1.4287e-05   1.4287e-05
  -2.5938e-05  -2.5938e-05
   3.6988e-05   3.6988e-05
  -4.6876e-05  -4.6876e-05
  -1.7506e-04  -1.7506e-04
   2.3315e-04   2.3315e-04
  -2.8747e-04  -2.8747e-04
   3.3532e-04   3.3532e-04
  -3.7622e-04  -3.7622e-04
  -9.6266e-05  -9.6266e-05
   1.1798e-04   1.1798e-04
  -1.3715e-04  -1.3715e-04
   1.5325e-04   1.5325e-04
  -1.6656e-04  -1.6656e-04
```

starter code — gnuplot — 80×24

```
   1.1715e-01   1.1715e-01
   7.5480e-02   7.5480e-02
   1.2570e-01   1.2570e-01
  -4.0759e-03  -4.0759e-03
   1.6968e-02   1.6968e-02
   1.7634e-01   1.7634e-01
   1.1313e-01   1.1313e-01
   8.6163e-02   8.6163e-02
   1.3229e-01   1.3229e-01
  -4.5296e-03  -4.5296e-03
   1.5005e-03   1.5005e-03
The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your backpropagation implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 2.26112e-11
```

```
The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your backpropagation implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 2.26112e-11


Cost at (fixed) debugging parameters (w/ lambda = 10): 0.576051
(this value should be about 0.576051)

Program paused. Press enter to continue.
```

```
Iteration    50 | Cost: 4.941781e-01
Program paused. Press enter to continue.

Visualizing Neural Network...

Program paused. Press enter to continue.

Training Set Accuracy: 95.200000
```