ex3

Note: a change to displayData.m for MacOS users: (link)

Note: if your images are upside-down, use flipud() to reverse the data. This is due to a change in gnuplot()'s defaults.

Tips on lrCostFunction():

- When completed, this function is identical to your costFunctionReg() from ex2, but using vectorized methods. See the ex2 tutorials for the cost and gradient - they use vectorized methods.
- ex3.pdf tells you to first implement the unregularized parts, then to implement the regularized parts. Then you test your code, and then submit it for grading.
- Do not remove the line "grad = grad(:)" from the end of the lrCostFunction.m script template. This line guarantees that the grad value is returned as a column vector.

oneVsAll() tutorial

predictOneVsAll() tutorial (updated)

predict() tutorial (for the NN forward propagation - updated)

Tutorial: ex3 oneVsAll()

Tom Mosher

Mentor[Assignment: Multi-class Classification and Neural Networks](#) · [2 years ago](#) · Edited

all_theta is a matrix, where there is a row for each of the trained thetas. In the exercise example, there are 10 rows, of 401 elements each. You know this because that's how all_theta was initialized in line 15 of the script template.

(note that the submit grader's test case doesn't have 401 elements or 10 rows - your function must work for any size data set - so use the "num_labels" variable).

Each call to fmincg() returns a theta vector. Be sure you use the lambda value provided in the function header.

You then need to copy that vector into a row of all_theta.

The oneVsAll.m script template contains several Hints and a code example to guide your work.

The "y == c" statement creates a vector of 0's and 1's for each value of 'c' as you iterate from 1 to num_labels. Those are the effective 'y' values that are used for training to detect each label.

Type these commands in your workspace to see how to copy a vector into a matrix:

```
Q = zeros(5,3)      % create a test matrix of all-zeros
v = [1 2 3]'        % create a column vector
Q(2,:) = v          % copy v into the 2nd row of Q
```

The syntax "(2,:)" means "use all columns of the 2nd row".

======

keywords: tutorial onevsall

ex3: tutorial for predictOneVsAll()

Tom Mosher

MentorWeek 4 · 2 years ago · Edited

The code you add to predictOneVsAll.m can be as little as two lines:

1. one line to calculate the sigmoid() of the product of X and all_theta. X is (m x n), and all_theta is (num_labels x n), so you'll need a transposition to get a result of (m x num_labels)
2. one line to return the classifier which has the max value. The size will be (m x 1). Use the "help max" command in your workspace to learn how the max() function returns two values.

Note that your function must return the predictions as a column vector - size (m x 1). If you return a row vector, the script will not compute the accuracy correctly.

ex3 tutorial for predict()

Tom Mosher

MentorWeek 4 · 2 years ago · Edited

Here is an outline for forward propagation using the vectorized method. This is an implementation of the formula in Figure 2 on Page 11 of ex3.pdf.

1. Add a column of 1's to X (the first column), and it becomes 'a1'.
2. Multiply by Theta1 and you have 'z2'.
3. Compute the sigmoid() of 'z2', then add a column of 1's, and it becomes 'a2'
4. Multiply by Theta2, compute the sigmoid() and it becomes 'a3'.
5. Now use the max(a3, [], 2) function to return two vectors - one of the highest value for each row, and one with its index. Ignore the highest values. Keep the vector of the indexes where the highest values were found. These are your predictions.

Note: When you multiply by the Theta matrices, you'll have to use transposition to get a result that is the correct size.

Note: The predictions must be returned as a column vector - size (m x 1). If you return a row vector, the script will not compute the accuracy correctly.

Note: Not getting the correct results? In the hidden layer, be sure you use sigmoid() first, then add the bias unit.

------ dimensions of the variables ---------

a1 is (m x n), where 'n' is the number of features including the bias unit

Theta1 is (h x n) where 'h' is the number of hidden units

a2 is (m x (h + 1))

Theta2 is (c x (h + 1)), where 'c' is the number of labels.

a3 is (m x c)

p is a vector of size (m x 1)

Ex3 Test Cases

Chirag Uttamsingh

MentorGeneral Discussion · 2 years ago · Edited by moderator

Here are the test cases for ex3 by Tom Mosher:

lrCostFunction - regularized (ex3.pdf Section 1.3.3):

```
% input
theta = [-2; -1; 1; 2];
X = [ones(5,1) reshape(1:15,5,3)/10];
y = [1;0;1;0;1] >= 0.5;       % creates a logical array
lambda = 3;
[J grad] = lrCostFunction(theta, X, y, lambda)

% results
J =  2.5348
grad =

   0.14656
  -0.54856
   0.72472
   1.39800
```

Note: your cost function must return the gradient as a column vector (size n x 1), NOT as a row vector (1 x n).

```
====
```

oneVsAll:

```
%input:
X = [magic(3) ; sin(1:3); cos(1:3)];
y = [1; 2; 2; 1; 3];
num_labels = 3;
lambda = 0.1;
[all_theta] = oneVsAll(X, y, num_labels, lambda)
%output:
all_theta =
  -0.559478   0.619220  -0.550361  -0.093502
  -5.472920  -0.471565   1.261046   0.634767
   0.068368  -0.375582  -1.652262  -1.410138
====
```

predictOneVsAll:

```
% input:
all_theta = [1 -6 3; -2 4 -3];
X = [1 7; 4 5; 7 8; 1 4];
predictOneVsAll(all_theta, X)
%output:
ans =
   1
   2
   2
   1
```

Note: your prediction function should NOT include any use of a fixed threshold. Select the classifier with the maximum output.

=====

predict:

```
Theta1 = reshape(sin(0 : 0.5 : 5.9), 4, 3);
Theta2 = reshape(sin(0 : 0.3 : 5.9), 4, 5);
X = reshape(sin(1:16), 8, 2);
p = predict(Theta1, Theta2, X)
% you should see this result
p =
  4
  1
  1
  4
  4
  4
  4
  2
```

Note: your prediction function should NOT include any use of a fixed threshold. Select the classifier with the maximum output.

Here are the values for the "a3" layer in the test case for predict().

```
a3 =

   0.53036   0.54588   0.55725   0.56352
   0.54459   0.54298   0.53754   0.52875
   0.49979   0.49616   0.49288   0.49024
   0.41357   0.42199   0.43736   0.45844
   0.37321   0.40368   0.44349   0.48911
   0.42073   0.45935   0.50210   0.54464
   0.50962   0.53216   0.55173   0.56659
   0.54882   0.55033   0.54738   0.54021
======
```