Neural Networks Learning

Adapted from Coursera

This homework and lab have you implementing the backpropagation network.

This uses the same data as in last week's lab – a set of 5000 training examples where each training example contains 400 features (a set of 20x20 pixel images).

The neural network will have three layers – input, hidden, and output.

Inputs:

- data set containing X and y (X being the 5000 training examples, y being their classification.
- Initial weights for the network – Theta1 and Theta2. Theta 1 is 25x401 – 25 units. Theta2 is 10x26 – 10 digit classes

1. Implement the cost function, nnCostFunction without regularization. With the initial Theta1 and Theta2, you should have a cost of around 0.287692.

2. Next, create a regularized version of the cost function, nnRCostFunction. If you use the loaded in Theta1 and Theta2 and lambda of 1, you should get a cost of around 0.383770.

3. Implement the sigmoidGradient function. If you use the following call:

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

   You should see results of:

   0.196612 0.235004 0.250000 0.235004 0.196612

4. If you got the correct values for the Cost earlier, you can use your cost function to numerically calculate the gradient values to do a check (you learned how to do this in the video lectures). Complete the code for computeNumericalGradient.m to do this.

5. Now, time for backprop! Here's the algorithm:

   You can do this either in a for-loop to handle each training data value separately or you can vectorize it to handle all of them at once. In this description, we'll give the formulas for handling one example at a time.

1. Set the input layer's values to X. You need to add a +1 term to include the bias unit.

$a^{(1)} = x^{(t)}$, where t means the t-th training example.

2. Do a feedforward pass to compute the activations for layers 2 and 3. So calculate $z^{(2)}$, $a^{(2)}$, $z^{(3)}$, and $a^{(3)}$.

3. Now, you need to do the backpropagation. You'll need to use a logical array for the output where $y_k = 1$ means that the example belongs to class k and $y_k=0$ means it does not.

4. Calculate the error in cost for the output layer, layer 3.

$\delta_k^{(3)} = (a_k^{(3)} - y_k)$

5. For the hidden layer,

$\delta^{(2)} = (\Theta^{(2)T} \delta^{(3)} .* g'(z^{(2)})$

6. Accumulate the gradients. You will need to remove or skip $\delta_0^{(2)}$

$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

7. Finally, calculate the (unregularized) gradients my multiplying by 1/m :

$D_{i,j}^{(l)} = (1/m) \Delta_{ij}^{(l)}$

6. Check your gradient using checkNNGradients. You should see a difference less than 1e-9.

7. Next, compute your gradients using Regularization. The good news is that you can just add the regularization term for the gradients after computing backprop:

   $D_{i,j}^{(l)} = (1/m) \Delta_{ij}^{(l)} + (\lambda/m)\Theta_{ij}^{(l)}$ for $j \geq 1$

   The trick here is you don't regularize the first column of $\Theta^{(l)}$ because it is used for the bias term.

8. Now, use fmincg to learn your parameters.

9. Finally write a predict function to get the predicted labels. You should get an accuracy of around 95.3% although it can vary by around 1% because of the random initializations.