Problem Set 3

Due Sunday, November 19, 2017 at 11:55pm

How to Submit

Create one .zip file (**not** .rar or something else) of your code and written answers and submit it via `ilearn.ucr.edu`. Your zip file should contain the following 3 files

- trainneuralnet.m
- nneval.m
- result.pdf

plus any other matlab functions your code depends on that you wrote.

Each file should include at the top (in comments if necessary)

- Your name & UCR student ID number
- The date
- The course (CS 171) & assignment number (PS 3)

Note: Do not use any Matlab function that is in a toolbox for this problem set.

Problem 1. [25 pts]

In this problem you are implement a two-layer (2 layers of weights, 1 hidden layer of units) neural network for classification with all non-linearities as sigmoids. The file `usps_all.mat` contains the data for hand-written digits. The supplied function `getusps` will load them in and produce a training set and a testing set for distinguishing the "7s" from the "9s."[1] In this case the Y values are either 0 or 1 (instead of -1 or +1), because that matches a sigmoid output more naturally.

You are to produce a plot of the testing error rate as a function of λ (the regularization strength) for three different numbers of hidden units: 5, 10, and 50. The supplied code `runusps` does this.

It calls two functions you should write: `[W1,W2] = trainneuralnet(X,Y,nhid,lambda)` and `predY = nneval(X,W1,W2)`. Getting a neural network to converge can be a little tricky. Please follow the guidelines below.

1. Start the weights randomly chosen using `randn` (that is each weight is selected from a normal distribution with mean 0 and unit standard deviation). Then divide all weights by 10 to make them closer to 0.
2. Each layer should have an "offset" unit (to supply a 1 to the next layer), except the output layer.
3. For a problem this small, use batch updates. That is, the step is based on the sum of the gradients for each element in the training set.
4. For the step size start with $\eta = 0.1$.
5. For real neural network training, people use rules like those discussed on `http://sebastianruder.com/optimizing-gradient-descent/index.html`, but they are more complex than needed for this simple example. Instead, every 1000 iterations, check to see if the loss function (including the regularization part) has decreased over the past 1000 iterations. If it has not, divide η by 10 and continue.
6. Check the maximum gradient element (before multiplying it by η). If its absolute value is less than 10^{-3}, stop the algorithm.

[1]Neural networks can be used to do the full 10-way classification problem, but we will keep it simple for this assignment.

A few notes to help

1. Display the loss function's value every 1000 iterations (or so). It should generally be getting smaller.
2. The smaller λ is and the more units, the more difficult the optimization will be for gradient descent.
3. Write a function to do forward propagation and one to do backward propagation. Write a function to evaluate the loss function. In general, break things up to keep things straight.
4. Processing the entire batch at once is much more efficient in Matlab. For instance, if you have the input as a vector, `x` the activations of the first layer is `W1*x`. However, if you have a set of inputs (for different examples) all lined up as columns in a matrix, then `W1*X` is a matrix, where each column is the activations for a different input. So, you can do an entire batch with one multiplication (and no loops).

The `runusps` function will save a PDF (`result.pdf`) of the results. Include this file in your submission.