Problem Set 4
Due Monday, May 16, 2016 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

- runq1.m
- trainneuralnet.m
- q2out.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
- Your UCR student ID number
- The date
- The course (CS 171)
- The assignment number (PS 4)

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

Problem 1. [15 pts]

In this problem you are to train a support vector machine to perform well on spam classification. The files `spamtrainX.data` and `spamtrainY.data` contain 3000 instances of features (X) and labels (Y) for e-mail spam classification. You are to write code that uses these to train a spam classifier and then predict the correct labels for the examples in `spamtestX.data`.

The file `learnsvm.m` has a function that will learn the weights and bias term (which is calculated separately for a SVM, so do not add a constant feature) for a linear support vector machine. You do not need to understand how it works. It takes a parameter `C`, the penalty for training points violating the margin.

You need to write a function `runq1` that takes no parameters and returns the predicted (+1 or -1) labels for the testing points as a vector. You should use the last 25% of the training data as a validation set in order to select the proper value of `C`. I would recommend trying values of `C` logarithmically spaced between 10^-2 and 10^3 (see the Matlab command `logspace`).

Problem 2. [35 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 `toy.data` contains the X (first 2 columns) and Y (last column) values for a 2D problem. In this case the Y values are either 0 or 1 (instead of -1 or +1), because that matches a sigmoid output more naturally.

There are no testing data for this. Instead, you are to produce an output plot of the decision surface. To help, two functions are provided:

- `getgridpts`: This takes the X matrix and returns a set of X points, layout out on a grid.
- `plotdecision`: This takes the training X, the training Y, the grid X points (from getgridpts) and the grid Y points (the values your classifier predicts for the grid X points, and plots the decision surface.

The pair of functions can be used as follows.

```
% load in and set up trainX as the training X pts and trainY as the Y pts
gridX = getgridpts(trainX);
% train classifier
% use classifer to predict labels for gridX, call this the vector gridY
plotdecision(trainX,trainY,gridX,gridY)
```

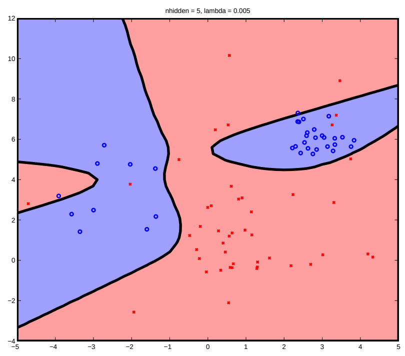
The supplied function `runq2` learns a neural network with different numbers of hidden units and different λ (regularization). It calls a function your should write: `trainneuralnet(X,Y,nhid,lambda)`. Getting a neural network to converge can be a little tricky. Please follow the guidelines below.

1. Start the weights each uniformly randomly sampled between -1 and $+1$.
2. 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.
3. For the step size on iteration i, use $\eta_i = \frac{1000}{25000+i}$. I tuned this by hand and it should work for these problems. For real neural network training, people use rules like those discussed on `http://sebastianruder.com/optimizing-gradient-descent/index.html`, but they are too complex for this class.
4. Check the maximum gradient value (before multiplying it by η_i). If it is less than 0.001, stop the algorithm.
5. The method will probably take 100,000 – 500,000 iterations to converge.

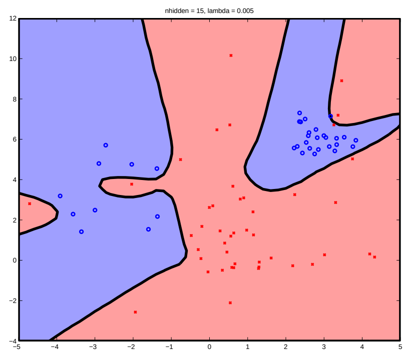
At the end, your function needs to plot the decision surface (see above) and return the two matrices of weights (the first for the input-to-hidden layer and the second for the hidden-to-output layer).

A few notes to help

1. Every 1000 iterations (or so), plot the surface so that you can see what is going on (see `drawnow`).
2. Display the loss function's value every 1000 iterations (or so). It should generally be getting smaller.
3. Write a function to do forward propagation and one to do backward propagation so that everything doesn't get mixed up in one big function.
4. My answers for two of the nine plots look like those below. Note that because of the random starting weights, your solutions might look a little different



nhid = 5, $\lambda = 0.005$



nhid = 15, $\lambda = 0.005$

The `runq2` function will save a PDF (`q2out.pdf`) of the results. Include this file in your submission.